ADOBE® COLDFUSION® 10
アプリケーションの開発

法律上の注意

最終更新日 **2012/7/12**

コンテンツ

第 5 章：ColdFusion アプリケーションのビルディングブロック

第 6 章：CFML アプリケーションの開発

第 7 章：データへのアクセスおよび使用

第 8 章：ColdFusion ORM

第 9 章：ColdFusion と HTML 5

第 15 章：Web 要素および外部オブジェクトの使用

第 16 章：外部リソースの使用

第 1 章：ColdFusion 10 の新機能

ColdFusion 10 は、Adobe の取り組みによって創造的な発展を遂げました。Web 開発者が堅牢なインターネットアプリケーションを構築するための支援を行います。このリリースは、比類なき HTML5 のビルトインサポートに加えて、強化されたセキュリティ、改良されたスケジューラー、Web サービスサポートを備えています。このリリースのハイライトは、REST サービス、Microsoft Exchange Server 2010 のサポート、および CFML の大幅な機能強化です。

このリリースの大きなポイントは次のとおりです。

JRun から Tomcat への変更

ColdFusion 10 スタンドアローンインストールでは、JRun の代わりに Tomcat が組み込まれます。

以前のバージョンの ColdFusion インストーラーでは、マルチサーバーインストールの作成が可能でしたが、ColdFusion 10 インストールでは、スタンドアローンインストールのみが可能です。エンタープライズ版またはデベロッパー版のライセンスを保有している場合は、ColdFusion をスタンドアローンモードでインストールした後に、複数のインスタンスおよびクラスタを作成できます。

詳しくは、『Adobe ColdFusion インストール』を参照してください。

セキュリティの機能強化

セキュリティの機能強化により、セキュリティの脆弱性、特に XSS および CSRF 攻撃の脅威が減少します。

また、このリリースには、ColdFusion セッションを効果的に管理するための機能強化も含まれています。

詳しくは、361 ページの「ColdFusion 10 でのセキュリティの機能強化」を参照してください。

ColdFusion WebSocket

株式、チャート作成、オンラインゲーム、ソーシャルネットワーク、様々な目的のダッシュボード、監視などを目的とするリアルタイムアプリケーションを ColdFusion WebSocket を使用して開発します。

ColdFusion では、WebSocket プロトコル用のメッセージングレイヤーを提供することで WebSocket を実装しています。これは、CFML と JavaScript を使用して簡単に制御できます。

詳しくは、563 ページの「ColdFusion WebSocket の使用」を参照してください。

言語の機能強化

- （クエリ用の）for-in 構文のサポート
- タグ本文でのファイルコンテンツの指定
- ColdFusion 関数用の呼び出しスタック
- アプリケーションのメタデータを取得する関数

- ディスクの空き容量の詳細を取得する関数
- アプリケーション固有のメモリ内ファイルシステム
- MIME タイプの検証によるセキュアなファイルのアップロード
- CFC の暗黙のコンストラクター
- CFC メソッドのメソッド連結
- CFC の暗黙の表記法
- 新しい ArraySlice 関数
- 新しい merge パラメーターによる arrayAppend のサポート
- 新しい format パラメーターを LSParseDateTime 関数に追加
- 新しい runOnce 属性を cfinclude に追加
- 新しい timeout 属性を cfstoredproc に追加
- 新しい maxLength 属性を cfparam に追加
- 新しい dateTimeFormat 関数と lsDateTimeFormat 関数
- 新しい reEscape 関数
- replaceList 関数での区切り文字の使用
- arraySort、listSort および structSort 関数に対する変更
- 暗黙的な構造体での :（コロン）の区切り文字の使用のサポート
- インターフェイスシグネチャでの output 属性の無視
- FUNCTION を ColdFusion データ型に変更
- クエリのループ処理における動的参照のサポート
- 新しい invoke 関数
- cfpop の新しい secure 属性
- cfloop の新しい group 属性
- for-in 構文における Java 配列のサポート
- queryAddRow および queryNew 関数の強化
- 新しい listRemoveDuplicates 関数
- XPath 2.0 と XSLT 2.0 シンタックスのサポート

ColdFusion のクロージャ

詳しくは、122 ページの「クロージャの使用」を参照してください。

Java 統合の強化

Java ライブラリをカスタムパスからロードします。

詳しくは、1132 ページの「ColdFusion 10 での拡張された Java 統合」を参照してください。

ColdFusion ORM 検索

ColdFusion ORM のインデックス作成および検索機能を使用します。

ColdFusion ORM を使用するアプリケーションを開発するときは、検索機能により全文検索が容易になります。クエリテキストに基づいて、検索条件に一致するすべての永続的エンティティを読み込むことができます。

詳しくは、559 ページの「ColdFusion ORM 検索」を参照してください。

Solr の機能強化

- データベースのインデックス作成でのデータ読み込みハンドラーの使用
- 動的なカスタムフィールドに基づいたインデックス作成および検索
- 個別のコレクションのリロード
- 検索用の言語の追加
- 検索システムのセキュリティ保護
- インデックスを作成したドキュメントの自動コミット
- 特定のフィールドまたはドキュメント全体の検索結果向上の促進

詳しくは、481 ページの「ColdFusion 10 での Solr の機能強化」を参照してください。

スケジューラーの機能強化

詳細で拡張可能な、整然とした方法でタスクをスケジュールします。このリリースでは、Quartz スケジューリングサービスがサポートされます。

詳しくは、399 ページの「スケジューラーの使用」を参照してください。

Microsoft Exchange Server 2010 との統合

Adobe ColdFusion は、Microsoft Exchange Server 2010 SP1 と対話できます。拡張機能は、次の操作での効率性をもたらす Microsoft Exchange Web Services（EWS）のサポートを提供します。

- 作成、変更、削除などのフォルダー操作。
- Exchange 組織の会議室および会議室リストの取得。
- 効率的なスケジュールに役立つユーザー空き時間情報。
- 会話の詳細の検索、コピー、移動、会話が読まれているかどうかのステータスなどの会話操作。

詳しくは、1219 ページの「Microsoft Exchange Server 2010 への接続」を参照してください。

クライアントとサーバー間での遅延ロード

ColdFusion ORM をバックエンドで使用し、Flex をフロントエンドで使用する、アプリケーションの関連エンティティの必要に応じたロードが、このリリースでは可能です。

アプリケーションでは、メインエンティティを取得し、関連エンティティは取得しないようにすることができます。クライアントアプリケーションが関連エンティティにアクセスを試みたときにのみ、エンティティが読み込まれます。

詳しくは、613 ページの「クライアントとサーバー間での遅延ロード」を参照してください。

Web サービスの機能強化

ColdFusion 10 には、Axis 2 Web サービスフレームワークが統合されています。これにより、Web サービスでは WSDL 2 仕様、SOAP 1.2 プロトコル、およびドキュメントリテラルラップスタイルを使用できます。

また、機能拡張により、ColdFusion 9 での Web サービスの使用時に発生する可能性があった相互運用性の問題の多くが解決されています。

詳しくは、1090 ページの「ColdFusion 10 での Web サービスの機能強化」を参照してください。

ColdFusion での RESTful Web サービス

ColdFusion Zeus では、REST（Representational State Transfer）サービスを作成して公開できます。クライアントは、HTTP/HTTPS リクエストを使用してこのサービスを使用できます。

詳しくは、1093 ページの「ColdFusion での RESTful Web サービス」を参照してください。

メディアプレーヤーの機能拡張

このリリースでの機能拡張は次をサポートします。

- HTML 5 ビデオ用の再生機能
- Flash Player がインストールされていない場合の HTML 5 ビデオプレーヤーへのフォールバック
- ブラウザーに依存しないビデオコントロール
- Flash ビデオの動的ストリーミング
- メディアプレーヤー用の高度なスキニング
- Flash ビデオの再生リスト
- Open Source Media Framework（OSMF）で構築されたプラグインを使用するメディアプレーヤーの拡張。例として、
- YouTube サーバーでのビデオの再生
- リニアモードおよび非リニアモードでのビデオ内での広告の表示による、ステージビデオサポートの使用
- ビデオへのタイトルの追加

詳しくは、586 ページの「メディアプレーヤーの機能拡張」を参照してください。

地理位置情報の表示

cfmap で showUser 属性が指定されている場合、地図上でのユーザーの位置を表示します。この機能は、HTML 5 準拠ブラウザーでのみ動作します。

詳しくは、592 ページの「地理位置情報の表示」を参照してください。

クライアントサイドのチャート作成

ColdFusion Zeus はクライアントサイドのチャート作成をサポートします。これは、既存のサーバーサイドのチャート作成機能（従来どおり提供されます）に対する追加です。クライアントサイドのチャート作成は以下をサポートします。

- 動的および対話形式のチャート作成：チャートの変更、スタイルの追加、新しい系列またはプロットの追加を行います。
- 適切なフォールバック機能を備えた主要なチャート形式：HTML 5、Flash、SVG、または VML のチャートを使用します。
- ブラウザーが HTML 5 のチャート作成関連機能をサポートしていない場合、チャートは Flash で表示されます。同様に、
- Flash がサポートされていない場合、チャートは HTML で表示されます。
- サーバーサイドのチャート作成と同様の機能：サーバーサイドのチャート作成機能のほとんどを、クライアントサイドのチャート作成で使用できます。
- 従来のチャートと新しいチャート：現在のチャートタイプに加えて、新しいチャートセットを提供します。
- サーバーへのトリップが最小：すべてのユーザー対話について、サーバーレベルでのチャート生成と比較した場合。

詳しくは、592 ページの「クライアントサイドのチャート作成」を参照してください。

キャッシュ機能の強化

- アプリケーション別のキャッシュ処理
- Ehcache の使用によるクエリのキャッシュ処理の強化

詳しくは、254 ページの「ColdFusion 10 でのキャッシュ機能の強化」を参照してください。

ColdFusion Administrator を使用したサーバーの更新

ColdFusion Administrator（サーバー更新／更新）を使用して、製品アップデートがあるかどうかを確認します。アップデートには、ColdFusion Zeus のホットフィックスおよびセキュリティホットフィックスが含まれる場合があります。

詳しくは、『Adobe ColdFusion 設定と管理』を参照してください。

ColdFusion Administrator のセキュアプロファイル

ColdFusion では、ColdFusion Administrator で選択した設定を有効または無効にして、ColdFusion サーバーのセキュリティを強化することができます。ColdFusion のインストール時に、セキュアプロファイル画面で入力を要求されたときにオプションを選択して、セキュアプロファイルを有効にすることができます。さらに、ColdFusion Administrator へのアクセスを許可する IP アドレスのカンマ区切りリストを指定することもできます。詳しくは、Enabling Secure Profile for ColdFusion Administrator を参照してください。

第 2 章：はじめに

『Adobe® ColdFusion® 9 アプリケーションの開発』ガイドでは、Adobe ColdFusion を使用してインターネットアプリケーションを開発するためのツールについて説明します。本マニュアルは、ColdFusion を学習している Web アプリケーションプログラマや、ColdFusion のプログラミング知識を拡充したいプログラマを対象としています。本マニュアルを読めば、ColdFusion に用意されているツールを使用してさまざまな種類の複雑な Web アプリケーションを開発するための基礎知識を習得できます。

本マニュアルの使用について

本マニュアルには、CFML に関する基本的な情報だけでなく、詳細な情報も記載されています。ColdFusion の基本機能を使用したことがある方や、Adobe ColdFusion Administrator の「ファーストステップ」をご覧になった方には、より使いやすい構成になっています。本マニュアルを読むときは、CFML 言語の要素に関する詳細な情報が記載されている『CFML リファレンス』を参照することをお勧めします。

Adobe ColdFusion 10 マニュアルについて

ColdFusion のマニュアルは、あらゆるユーザーを総合的にサポートできるように作成されています。

マニュアルセット

ColdFusion のマニュアルセットには、次のマニュアルが含まれています。

タイトル	説明
Adobe® ColdFusion® 10 インストール	Windows、Macintosh、Solaris、Linux、および AIX 環境でのシステムインストールおよび基本設定について説明します。
Adobe® ColdFusion® 10 設定と管理	ColdFusion の管理作業 (サーバー設定の管理、データソースの設定、セキュリティの管理、ColdFusion アプリケーションのデプロイ、キャッシュ、CFX タグのセットアップ、ColdFusion サーバーモニタを使用したサーバーアクティビティの監視、Web サーバーの設定など) について説明します。
Adobe® ColdFusion® 10 アプリケーションの開発	ダイナミック Web アプリケーションの開発方法について説明します。このガイドには、CFML プログラミング言語と ColdFusion の機能 (ColdFusion Web サービス、ColdFusion ポートレット、ColdFusion ORM、AJAX サポート、Flex および AIR 統合など) の使用方法と、他社の製品およびテクノロジー (Microsoft Office、OpenOffice、SharePoint など) との統合に関する詳細情報が含まれています。
Adobe® ColdFusion® 10 CFML リファレンス	すべての ColdFusion タグ、関数、変数に関する説明、シンタックス、使用方法、コード例が含まれています。

オンラインマニュアルの参照

ColdFusion のすべてのマニュアルは、HTML 形式および Adobe Acrobat PDF (Portable Document Format) 形式でオンラインから入手できます。オンラインマニュアルを表示するには、ColdFusion ヘルプ＆サポートページ（www.adobe.com/go/learn_cfu_support_jp）にアクセスしてください。これらのオンラインマニュアルでは、新しいコメントを追加したり、既存のコメントを参照することもできます。

第 3 章：ColdFusion の概要

Adobe ColdFusion を使用すれば、ダイナミックなインターネットアプリケーションを作成できます。

インターネットアプリケーションと Web アプリケーションサーバーについて

ColdFusion は、Web アプリケーションサーバーで実行するインターネットアプリケーションを開発するためのツールです。

Web ページとインターネットアプリケーションについて

インターネットは、スタティックな HTML ページの集まりから、アプリケーションをデプロイするためのプラットフォームへと発展してきました。スタティックな Web ページの集合であったインターネットは、まず、ダイナミックなインタラクティブコンテンツへと進化しました。一定のコンテンツしか表示できないページで商品やサービスを宣伝するだけであった企業は、ダイナミックページを導入することで、e コマースから業務プロセスの管理まで、広範なビジネス活動をインターネットで展開できるようになりました。たとえば書店であれば、スタティックな HTML ページを使用することで、店舗の地図や提供しているサービス (特別注文など) を掲示したり、サイン会などの今後のイベントを告知したりすることができますが、ダイナミックな Web サイトを構築すれば、顧客からオンラインで注文を受けたり、本の感想を投稿できるページを用意したり、顧客の好みに応じて購入アドバイスを表示したりすることができます。

さらに最近では、さまざまなアプリケーションを実装するためのインフラストラクチャとしてインターネットが活用されています。XML、Web サービス、J2EE (Java 2 Platform, Enterprise Edition)、Microsoft .NET などのテクノロジーが登場したことで、ビジネス活動のさまざまな側面がインターネットに統合できるようになりました。インターネットを使用すれば、顧客サービス、受注、受注処理、請求などの広範なビジネス活動を統合することができます。

Adobe ColdFusion は、ダイナミックな Web サイトやインターネットアプリケーションをすばやく簡単に構築できるアプリケーション開発環境です。ColdFusion を使用すれば、複雑なテクノロジーに精通していなくても高度な Web サイトやインターネットアプリケーションが開発でき、スキルの高い開発者は最新のインターネットテクノロジーも活用できます。

Web アプリケーションサーバーについて

通常、Web ブラウザがリクエストを行うと、Microsoft IIS (Internet Information Server) や Apache Web サーバーなどの Web サーバーがそれを処理して、リクエストされた情報をブラウザに返します。Web サーバーが返す情報としては、HTML ファイルや FLA ファイルなどがあります。

Web サーバーの役割は、リクエストが届くのを待ち受けてそのリクエストにできるだけ早く応答することなので、Web サーバーが実行できることは限られています。たとえば、Web サーバーでは次のタスクは行えません。

- データベース、他のリソース、または他のアプリケーションの操作
- ユーザーの好みやリクエストに基づいたカスタム情報の提供
- ユーザー入力の検証

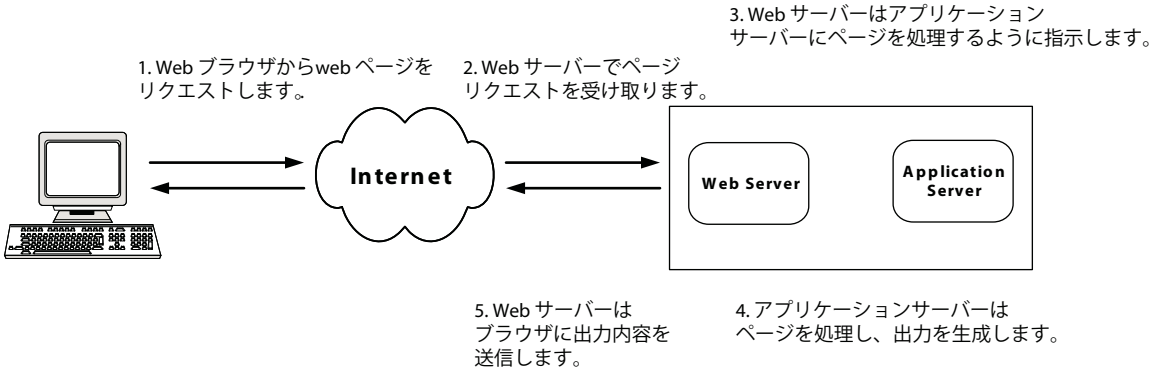
Web サーバーが行うのは、基本的に、情報を見つけて Web ブラウザに返すことだけです。

Web サーバーの機能を拡張するには、**Web** アプリケーションサーバーを使用します。Web アプリケーションサーバーは、前述のようなタスクを行えるように Web サーバーの機能を拡張するためのプログラムです。

Web サーバーと Web アプリケーションサーバーの連携

Web サーバーと Web アプリケーションサーバーが連携してページリクエストを処理する過程を、次の手順に示します。

1. ユーザーがブラウザに URL を入力してページをリクエストすると、Web サーバーがそのリクエストを受信します。
2. Web サーバーはファイル名の拡張子を調べて、そのページを Web アプリケーションサーバーで処理するかどうかを判断します。次のいずれかの処理が行われます。
   - シンプルな Web ページ (拡張子が HTM または HTML のページなど) をユーザーがリクエストした場合は、Web サーバーがそのリクエストを処理し、ブラウザにファイルを送信します。
   - Web アプリケーションサーバーで処理する必要があるページ (ColdFusion リクエストであれば拡張子が CFM、CFML、または CFC のページ) をユーザーがリクエストした場合は、Web サーバーから Web アプリケーションサーバーにそのリクエストが渡されます。Web アプリケーションサーバーはページを処理して、その結果を Web サーバーに送信します。Web サーバーは受信した結果をブラウザに返します。このプロセスを次の図に示します。



Web アプリケーションサーバーを使用すれば、プログラムの命令を実行して、Web ブラウザで処理可能な出力を生成できるので、データを活用したインタラクティブな Web サイトを実現できます。たとえば、次のことが行えます。

- 他のデータベースアプリケーションにクエリーを発行してデータを取得する。
- フォーム要素にデータをダイナミックに入力する。
- Flash データをダイナミックに生成する。
- アプリケーションのセキュリティを提供する。
- HTTP、FTP、LDAP、POP、SMTP などの標準プロトコルを使用して他のシステムを統合する。
- ショッピングカートや e コマースの Web サイトを作成する。
- ユーザーがフォームを送信したら電子メールメッセージを返す。
- キーワード検索の結果を返す。

ColdFusion について

Adobe ColdFusion は、ダイナミックなインターネットアプリケーションを作成するためのスクリプト環境を備えたサーバーです。CFML (ColdFusion Markup Language) は、習得しやすいタグベースのスクリプト言語で、エンタープライズデータへのアクセス機能や、強力なビルトイン検索機能、グラフ作成機能などを備えています。ColdFusion を使用すれば、ダイナミックな Web サイト、コンテンツパブリッシュシステム、セルフサービス型アプリケーション、コマースサイトなどを簡単に構築してデプロイできます。

ColdFusion ページは、Web アプリケーションの作成に使用するプレーンテキストファイルです。ColdFusion アプリケーションを作成するには、すべてのコードを自分で入力してもかまいませんし、一部のエディタで提供されているウィザードを使用して大半のコードを生成することもできます。

ColdFusion ページの保存

ColdFusion サーバーで処理するページは、ColdFusion がインストールされているコンピュータに保存する必要があります。ローカルサーバー (ColdFusion が稼動しているコンピュータ) 上でページを作成している場合は、そのコンピュータにページを保存します。リモートサーバーを使用している場合は、そのサーバーにページを保存します。

J2EE 設定を使用している場合は、通常、ColdFusion の **Web** アプリケーションのルートディレクトリの下に ColdFusion ページを保存します。たとえば、JRun の J2EE 設定でデフォルトのディレクトリ構造を使用している場合は、<JRun のルートディレクトリ >/servers/cfusion/cfusion-ear/cfusion-war の下にページを保存します。

ColdFusion ページのテスト

作成したコードが期待どおりに動作するか確認するには、http://localhost/test/mypage.cfm などの該当する URL にアクセスして、ColdFusion ページをブラウザで表示します。ビルトイン Web サーバーを使用している場合は、http://localhost:8500/test/cfpage.cfm などのように、使用するポートを URL で指定します。localhost は、ページをローカルで表示する場合にのみ有効です。

注意：Vista では、::1 というアドレスが localhost に相当します。ColdFusion の GetLocalHostIP 関数を使用すれば localhost の IP アドレスを取得できます。

リモートサイトの URL にアクセスする場合は、http://< サーバーの IP アドレス >/test/mypage.cfm のように、ColdFusion がインストールされているサーバーのサーバー名または IP アドレスを指定します。J2EE 設定の ColdFusion を使用している場合は、http://< サーバー >/< コンテキストルート >/mypage.cfm のように、必要に応じてコンテキストルートも指定します。たとえば、EAR ファイルをデプロイし、cfconroot というデフォルトのコンテキストルートを使用している場合は、http://localhost/cfconroot/test/mypage.cfm と指定します。

ColdFusion の要素

ColdFusion は、次のコア要素から構成されます。

- ColdFusion スクリプト環境
- CFML
- ColdFusion Administrator

ColdFusion スクリプト環境

ColdFusion スクリプト環境は、インターネットアプリケーションを開発するための効率的なモデルを提供します。この ColdFusion スクリプト環境の中核をなしているのが、CFML (ColdFusion Markup Language) です。CFML はタグベースのプログラミング言語で、Web プログラミングに必要な低レベル処理の多くが高レベルのタグや関数にカプセル化されています。

CFML

CFML (ColdFusion Markup Language) は HTML に似たタグベースの言語で、固有のタグや関数が用意されています。CFML を使用すれば、データベースコマンド、条件演算子、高度な書式制御関数などを使用して標準の HTML ファイルを拡張し、保守を行いやすい Web アプリケーションを迅速に作成できます。また、HTML の拡張以外にも CFML を活用できます。たとえば、Flash 要素と CFML だけで構成される Flash 出力を作成したり、他のアプリケーションで利用可能な Web サービスを作成したりすることもできます。

詳細については、13 ページの「CFML の要素」を参照してください。

CFML タグ

CFML のタグは HTML のタグに似ています。開始タグと (多くの場合) 終了タグがあり、各タグは山括弧で囲まれています。終了タグの前にはスラッシュ記号 (/) があり、すべてのタグの名前は cf で始まります。たとえば、次のようになります。

```
<cftagname>
    tag body text and CFML
</cftagname>
```

CFML は、一種のアブストラクションレイヤーです。インターネットアプリケーションのプログラミングで必要な多くの低レベル処理が隠蔽されているので、生産性の向上が図れます。また、CFML は強力で柔軟性が高いので、ファイル、データベース、レガシーシステム、メールサーバー、FTP サーバー、オブジェクト、コンポーネントなどを統合するアプリケーションを簡単に構築できます。

CFML タグを使用すれば、さまざまな処理が行えます。条件処理やループ処理などのプログラミング構文を使用したり、グラフ作成や全文検索などのサービスを利用することもできます。また、FTP、SMTP/POP、HTTP などのプロトコルにアクセスすることもできます。次の表に、代表的な ColdFusion タグの例を示します。

タグ	用途
cfquery	データベースへの接続を確立し (確立されていない場合)、クエリーを実行して結果を ColdFusion 環境に返します。
cfoutput	ColdFusion の関数、変数、式などの処理結果を使用して出力を表示します。
cfset	ColdFusion 変数に値を設定します。
cfmail	アプリケーション変数、クエリー結果、またはサーバーファイルを使用して、アプリケーションから SMTP メールメッセージを送信します (メールを受信するには cfpop という別のタグを使用します)。
cfchart	アプリケーションデータやクエリー結果を、Flash、JPG、または PNG 形式の棒グラフや円グラフなどに変換します。
cfobject	COM (Component Object Model) コンポーネント、Enterprise JavaBeans などの Java オブジェクト、一般的な CORBA (Object Request Broker Architecture) オブジェクトなどの、他のプログラミング言語で作成されたオブジェクトを呼び出します。

CFML タグの詳細については、『CFML リファレンス』を参照してください。

CFML 関数と CFScript

CFML では、文字列操作、データ管理、システム機能などのさまざまな処理がビルトイン関数として用意されています。また、CFScript というビルトインスクリプト言語も用意されています。CFScript を使用すれば、JavaScript などの一般的なプログラミング言語に近い形でコードを記述できます。

CFML の拡張

CFML では、カスタムタグやユーザー定義関数 (UDF) を作成したり、COM、C++、Java のコンポーネント (JSP タグライブラリなど) を統合したりすることで、CFML を拡張できます。また、ColdFusion コンポーネント (CFC) を作成することで、関連する関数やプロパティをカプセル化してそのインターフェイスを提供できます。

これらの機能を使用すれば、構築するアプリケーションや Web サイトに適した再利用可能な機能を簡単に作成できます。

CFML 開発ツール

Adobe® Dreamweaver® CS3 を使用すれば、ColdFusion アプリケーションを効率的に開発できます。Dreamweaver には、CFML のデバッグツールなど、ColdFusion の開発を効率化しサポートするさまざまな機能が用意されています。CFML は HTML に似たテキスト形式で記述されており、ColdFusion ページで HTML を使用することも多いので、HTML エディタやメモ帳などのテキストエディタを使用して ColdFusion アプリケーションを作成することもできます。

ColdFusion 9 にはラインデバッガが用意されており、Eclipse™ または Adobe Flex™ Builder™ で ColdFusion アプリケーションをデバッグすることができます。

ColdFusion Administrator

ColdFusion Administrator では、ColdFusion アプリケーションサーバーの設定や管理が行えます。ColdFusion Administrator は、セキュリティ保護された Web ベースのアプリケーションで、インターネットに接続しているコンピュータから Web ブラウザでアクセスできます。これには、ColdFusion サーバーのステータスを確認できるサーバーモニタが含まれています。

ColdFusion Administrator の詳細については、『ColdFusion 設定と管理』を参照してください。

J2EE と ColdFusion アーキテクチャについて

インターネットソフトウェアの市場が成熟するにつれて、ColdFusion アプリケーションなどの分散インターネットアプリケーションに必要なインフラストラクチャサービスが標準化されてきました。今日最も広く採用されている標準は J2EE (Java 2 Platform, Enterprise Edition) 仕様です。J2EE は、複数のオペレーティングシステムでデータベース、プロトコル、およびオペレーティングシステム機能を利用するための、共通インフラストラクチャサービスのセットを提供します。

ColdFusion と J2EE プラットフォームについて

ColdFusion は Java テクノロジープラットフォーム上に実装されています。データベース接続、ネーミングサービスやディレクトリサービス、その他のランタイムサービスなどの基本サービスの多くは、J2EE アプリケーションサーバーを使用して提供されています。ColdFusion は、組み込みの J2EE サーバーを使用するように設定する (サーバー設定にする) ことも、独立した J2EE アプリケーションサーバーに J2EE アプリケーションとしてデプロイする (マルチサーバー設定または **J2EE** 設定にする) ことも可能です。ColdFusion エンタープライズ版には、JRun J2EE アプリケーションサーバーの完全機能版が含まれています。または、IBM WebSphere や BEA WebLogic などのサードパーティの J2EE サーバーにデプロイすることもできます。

ColdFusion 設定の詳細については、『ColdFusion インストール』を参照してください。

ColdFusion のスクリプト環境は J2EE プラットフォーム上に実装されているので、J2EE プラットフォームの強力な機能が活用できるとともに、使いやすいスクリプト環境やビルトインサービスも利用できます。ColdFusion は J2EE プラットフォーム上に構築されているので、J2EE や Java の機能を ColdFusion アプリケーションに簡単に統合できます。たとえば、ColdFusion ページでは次のことが行えます。

- JSP (Java Server Pages) や Java サーブレットとセッションデータを共有する。
- カスタムの JSP タグライブラリをインポートして、ColdFusion のカスタムタグと同様に使用する。
- J2EE Java API、JavaBeans、Enterprise JavaBeans などの Java オブジェクトを統合する。

ColdFusion での J2EE 機能の使用方法の詳細については、1109 ページの「CFML アプリケーションへの J2EE および Java 要素の統合」を参照してください。

第 4 章：CFML プログラミング言語

CFML の要素

CFML には、タグ、関数、定数、変数、式、CFScript などの基本要素が用意されています。これらの要素を使用すれば、インタラクティブな Web アプリケーションを簡単に開発できます。

CFML の基礎

CFML はダイナミックアプリケーションを開発するためのツールであり、プログラミング言語のさまざまな特徴を利用できますそれらの特徴には、関数、式、変数、定数、フロー制御構造 (if-then やループなど) が含まれます。また、CFML には " 言語内の言語 " である CFScript が用意されており、JavaScript に似たシンタックスでさまざまな処理を記述できます。

これらの要素や、コメント、データ型、エスケープ文字、予約語などの基本的な CFML エンティティを使用すれば、複雑なアプリケーションを作成できます。

コメント

ColdFusion のコメントと HTML のコメントはよく似ています。ただし、ハイフンを 2 つではなく 3 つ挿入する点が異なります。次に例を示します。

```
<!--- This is a ColdFusion Comment. Browsers do not receive it. --->
```

ColdFusion サーバーは、Web サーバーにページを返す前に、そのページに含まれている ColdFusion コメントをすべて削除します。したがって、ブラウザで受信されるページにコメントは含まれていません。ユーザーがページのソースを表示してもコメントは表示されません。

CFML コメントは、タグ本文以外にも、開始タグ、関数呼び出し、# 記号で囲まれた変数テキストの中に挿入できます。ColdFusion では、コメント内のテキストは無視されます。

```
<cfset MyVar = var1 <!--- & var2 --->>
<cfoutput>#Dateformat(now() <!---, "dddd, mmmm yyyy" --->)#</cfoutput>
```

この方法は、式の一部や、オプションの属性や引数などを一時的にコメントアウトしたい場合に便利です。

次の例のように、コメントをネストすることもできます。

```
<!--- disable this code
<!--- display error message --->
<cfset errormessage1="Oops!">
<cfoutput>
#errormessage1#
</cfoutput>
--->
```

このようにネストは、アプリケーションのテスト時に、特定のコードセクションを一時的に無効にしたい場合に便利です。

コメントの中にコメントを埋め込むことは可能ですが、このテクニックを使用する際は注意が必要です。

注意：コメントは、<cf_My<!--- New --->CustomTag> のようにタグ名や関数名の中に挿入することはできません。また、IsDefined("My<!--- New --->Variable") のように文字列の中に挿入することもできません。

タグ

ColdFusion のタグは、情報を処理するように ColdFusion サーバーに指示するためのマークです。ColdFusion サーバーで処理されるのは、タグの本文のみです。ColdFusion の外にあるテキストはそのまま Web サーバーに返されます。ColdFusion ではさまざまなビルトインタグが使用でき、カスタムタグも作成できます。

タグのシンタックス

ColdFusion のタグは、HTML のタグと同じ形式で記述します。タグは山括弧 (< と >) で囲まれており、0 個以上の属性を指定できます。多くの ColdFusion タグには本文があります。つまり、開始タグと終了タグの間に、処理するテキストを記述します。次に例を示します。

```
<cfoutput>
    Hello #YourName#! <br>
</cfoutput>
```

cfset や cfhttp のように、本文がないタグもあります。次の例のように、必要な情報はすべて開始文字 (<) と終了文字 (>) の間に挿入します。

```
<cfset YourName="Bob">
```

注意：cfset タグは他のタグと異なり、本文も引数も持ちません。このタグは、変数に値を代入するステートメントを囲みます。cfset タグでは、結果変数に値を代入せずに関数を呼び出すこともできます。

本文を持つことができるタグでも、必要な情報を属性ですべて指定できる場合は、必ずしも本文を記述する必要はありません。タグに本文がない場合は、終了タグを省略して、開始タグの終了文字 (>) の前にスラッシュを置くこともできます。たとえば、次のようになります。

```
<cfprocessingdirective pageencoding="euc-jp" />
```

ほとんどの場合は、この例のように **attributeName**=" **attributeValue**" の形式を使用して、タグの中で属性を直接指定します。または、すべての属性を 1 つの構造体に格納して、その構造体を単一の attributeCollection 属性で指定することもできます。たとえば、次のように記述します。

```
<tagname attributeCollection="#structureName#">
```

cfmodule を除くすべてのビルトイン ColdFusion タグでは、attributeCollection 属性を指定した場合はその他の属性は指定できません。この方法は、タグの引数の個数や値が処理結果によって異なる場合に便利です。この場合の例を次に示します。

```
<!--- Configure dynamic attribute variables. --->
    <cfparam name="theURL" default="http://www.adobe.com">
    <cfparam name="resolveURL" default="yes">

<!--- Code that dynamically changes values for attributes can go here. --->

<!--- Create an arguments structure using variables. --->
    <cfset myArgs=StructNew()>
    <cfset myArgs.url="#theURL#">
<!--- Include a user name and password only if they are available. --->
    <cfif IsDefined("username")>
    <cfset myArgs.username="#username#">
</cfif>
<cfif IsDefined("password")>
    <cfset myArgs.password="#password#">
</cfif>
<cfset myArgs.resolveURL="#resolveURL#">
<cfset myArgs.timeout="2">

<!--- Use the myArgs structure to specify the cfhttp tag attributes. --->
    <cfhttp attributeCollection="#myArgs#">
    <cfoutput>
        #cfhttp.fileContent#
    </cfoutput>
```

注意：attributeCollection 属性は、cfmodule タグおよびカスタムタグで使用する場合と、その他のタグで使用する場合とでは、その性質が異なります。cfmodule タグおよびカスタムタグでは、attributeCollection 属性に加えて、カスタムタグ属性を明示的に指定することができます。また、cfmodule タグでは、attributeCollection 属性に name および template 属性を含めることはできません。これらの属性は、cfmodule タグで直接指定します。

attributeCollection 属性は、次を除くすべてのタグで使用できます。

cfargument	cfelseif	cflogout	cfset
cfbreak	cffunction	cfloop	cfsilent
cfcase	cfif	cfparam	cfswitch
cfcatch	cfimport	cfprocessingdirective	cftry
cfcomponent	cfinterface	cfproperty	
cfdefaultcase	cflogin	cfrethrow	
cfelse	cfloginuser	cfreturn	

ビルトインタグ

ビルトインタグは、ColdFusion の中核をなす機能です。これらのタグを使用すれば、次のようなさまざまな処理が行えます。

- 変数の操作
- インタラクティブフォームの作成
- データベースへのアクセスおよび操作
- データの表示
- ColdFusion ページでの実行フローの制御
- エラー処理
- ColdFusion ページの処理

- CFML アプリケーションフレームワークの管理
- ファイルおよびディレクトリの操作
- COM、Java、CORBA オブジェクト、実行可能プログラムなどの外部ツールおよびオブジェクトの使用
- mail、http、ftp、pop などのプロトコルの使用

各タグの詳細については、『CFML リファレンス』を参照してください。

カスタムタグ

ColdFusion ではカスタムタグを作成できます。作成できるカスタムタグには、次の 2 つのタイプがあります。

- ColdFusion ページである CFML カスタムタグ
- Java、C++ などのプログラミング言語で記述する CFX タグ

カスタムタグを使用すれば、頻繁に利用するビジネスロジックや表示コードをカプセル化できます。頻繁に利用するコードをカスタムタグとして用意しておけば、さまざまな場所から呼び出せるようになります。また、カスタムタグを使用すると、複雑なロジックを単純な 1 つのインターフェイスにまとめることができます。この方法を使用すれば、開発したコードを簡単に配布できます。タグのロジックにアクセスできないようにタグを暗号化することもできます。

Adobe ColdFusion コンポーネントの情報交換サイト (www.adobe.com/go/learn_cfu_cfdevcenter_jp) では、無償または有償で提供されている、さまざまなカスタムタグにアクセスできます。たとえば、クライアントのブラウザで Cookie や JavaScript が有効であるかどうかを確認するカスタムタグや、あるリストボックスから別のリストボックスに項目を移動するカスタムタグなどが用意されています。これらのタグの多くは、ソースコードとともに無償で提供されています。

CFML カスタムタグ

CFML でカスタムタグを記述する場合は、ビルトインタグや他のカスタムタグを含む、ColdFusion 言語のすべての機能を利用できます。CFML カスタムタグには本文と終了タグを含めることができます。CFML カスタムタグは CFML で記述するので、Java などのプログラミング言語の知識は必要ありません。CFML カスタムタグは、ユーザー定義関数と比べて多くの機能を実現できますが、効率は劣ります。

CFML カスタムタグの詳細については、206 ページの「カスタム CFML タグの作成と使用」を参照してください。CFML カスタムタグ、ユーザー定義関数、CFX タグなどの ColdFusion コードの再利用方法の詳細および比較については、143 ページの「ColdFusion 要素の作成」を参照してください。

CFX タグ

CFX タグは、Java、C++ などのプログラミング言語で記述する ColdFusion カスタムタグです。CFX タグでは、ランタイム環境へのアクセスなど、使用する言語に用意されているツールやリソースをすべて利用できます。また、CFX タグはコンパイルされているので、一般に CFML カスタムタグよりも実行速度が速くなります。クロスプラットフォームに対応している CFX タグもありますが、COM オブジェクトや Windows API を利用しているタグなど、多くの CFX タグはプラットフォーム固有です。

CFX タグの詳細については、221 ページの「CFXAPI カスタムタグの構築」を参照してください。

関数および演算子としてのタグ

ColdFusion には、CFML タグに対応するさまざまな関数や演算子言語要素が用意されています。これらの要素を既存の CFScript 言語に組み合わせると、完全に CFScript 内で多数の関数や CFC を定義できます。

新しい関数と演算子は、次のタグカテゴリに分類されます。

- 本文を持たないタグ (cfexit や cfinclude など)
- 本文を持つ言語タグ (cflock や cftransaction など)
- 本文を持つサービスタグ (cfmail や cfquery など)

- コンポーネントや関数を定義して使用するためのタグ (cfcomponent、cfinterface、cfimport、cffunction、cfproperty、cfargument)。詳細については、113 ページの「CFScript でのコンポーネントと関数の定義」を参照してください。

本文を持たないタグ

基本的な ColdFusion タグの一部に対応する CFScript 演算子が追加されました。これらの演算子では、標準のタグ属性の一部を指定できますが、カスタム属性は使用できません。これらの演算子は値を返しません。

CFML タグとそれに対応する CFScript シンタックスを次のリストに示します。

- cfabort: abort ["message"];
- cfexit: exit ["methodName"];
- cfinclude: include "template";
- cfparam: param [type] name [=defaultValue];

  param 属性で name=value のペアをいくつでも指定できるようになりました。また、param では、<cfparam> のすべての属性を名前 / 値のペアとして指定できます。

  次に例を示します。

  ```
  <cfscript>
  param name="paramname" default="value" min="minvalue" max="maxvalue" pattern="pattern"
  </cfscript>
  ```

- cfrethrow: rethrow;
- cfthrow: throw "message";

ステートメントパラメータの詳細については、『CFML リファレンス』で対応するタグ属性の説明を参照してください。

本文を持つ言語レベルのタグ

ColdFusion には、次に示す言語 (コンパイラ) レベルのタグ (本文を持つタグ) の機能を提供する CFScript 要素が含まれています。これらのタグは、本文内のコードの実行を管理します。

- cflock: lock
- cfthread: thread
- cftransaction: transaction

また、スレッドとトランザクションのサポートには、threadJoin や transactionCommit などの関数も含まれます。これらの関数を使用すると、タグと関数のどちらを使用して定義されているかにかかわらず、あらゆるスレッドやトランザクションを管理できます。

lock、thread、および transaction 操作のシンタックスを次に示します。

```
operationName attributeName1=value1 attributName2=value2...
{body contents }
```

cflock

lock 操作には、特別な特徴や制限はありません。cflock タグの属性はすべて、有効な操作パラメータとして使用できます。次のコードでは、lock 操作を使用しています。

```
lock scope = "request" timeout = "30" type = "Exclusive" {
request.number = 1;
writeoutput("E-Turtleneck has now sold "& request.number &"
turtlenecks!");
}
```

cftransaction

transaction 操作を使用するには、begin アクションパラメータを指定します。transaction の一般的な形式を次に示します。

```
TRANSACTION action="begin" [isolation="isolationValue"] {
transaction code
}
```

transaction ブロック内では、トランザクションを管理するために次のメソッドを呼び出します。

- transactionCommit()
- transactionRollback([savepoint])
- transactionSetSavepoint([savepoint])

savepoint パラメータは、セーブポイントを識別する文字列です。

注意：これらのメソッドは、cftransaction タグの本文でも使用できます。

transaction 操作は、ネストすることもできます。ネストされたトランザクションの詳細については、『CFML リファレンス』の cftransaction を参照してください。

ネストされた transaction 操作の使用例を次に示します。

```
<cfscript>
    qry = new Query();
    qry.setDatasource("test");
    qry.setSQL("delete from art where artid=62");
    qry.execute();
    TRANSACTION action="begin"
    {writeoutput("Transaction in cfscript test");
    TRANSACTION action="begin" {
    qry.setSQL("insert into art(artid, artistid, artname, description, issold, price)
    values ( 62, 1, 'art12', 'something', 1, 100)");
    qry.execute();}
    transactionSetSavepoint("sp01");
    qry.setSQL("update art set artname='art45' where artid=62");
    qry.execute();
    transactionSetSavepoint("sp02");
    qry.setSQL("update art set artname='art56' where artid=62");
    qry.execute();
    transactionrollback("sp02");
    transactioncommit();
    }
</cfscript>
```

cfthread

thread 操作を使用するには、run アクションパラメータを指定します。thread は、操作本文のコードを実行します。thread ブロックの一般的な形式を次に示します。

```
THREAD name="text" [action="run"] [priority="priorityValue"
application-specific attributes] {
thread code
}
```

thread 操作本文のコードは、単一の ColdFusion スレッドで実行されます。本文の外側にあるコードは、このスレッドの一部としては実行されません。スレッドを管理するには、次のメソッドを使用します。

- threadTerminate(threadName)

  この関数を使用すると、threadName パラメータで指定されたスレッドが終了します。この関数の動作は、cfthread action="terminate" と同じです。

- threadJoin([[threadName], timeout])

  この関数を使用すると、指定されたスレッドに現在のスレッドが結合されます。現在のスレッドは、指定されたスレッドが完了するか、タイムアウト期間が経過するまで (いずれかの条件が先に満たされるまで) 待機します。thread 関数ブロック内における現在のスレッドはそのブロックスレッドに属し、thread 関数ブロック外における現在のスレッドはページスレッドになります。threadName パラメータでは、ページスレッドに結合するスレッドをカンマで区切って指定します。この属性を省略すると、現在のスレッドは、すべての ColdFusion スレッドの実行が完了するまで待機します。timeout パラメータでは、指定したスレッドの処理が完了するまで呼び出し元のスレッドを待機させる最大時間をミリ秒単位で指定します。未完了のスレッドがある場合でも、タイムアウト期間が経過すると、現在のスレッドの処理が直ちに開始されます。この属性を省略した場合、現在のスレッドは、指定されたすべてのスレッドの実行が完了するまで待機します。

  注意： これらの関数は、cftransaction タグを使用して作成したトランザクションと組み合わせて使用することもできます。

本文を持つサービスタグ

ColdFusion には、次のサービスタグに対応するオブジェクトが用意されており、これらのオブジェクトは CFC として実装されます。

- cfftp
- cfhttp
- cfmail
- cfpdf
- cfquery
- cfstoredproc

これらのタグは本文を持ち、クエリーの実行やメールの送信などのサービスを提供します。これらのタグの多くは action 属性を持ちますが、残りのタグは暗黙的なアクション (execute など) を持ちます。cfmail と cfpdf を除く各サービスタグでは、該当するプロパティセットを含むコンポーネントが返され、データにアクセスするにはそれらのプロパティに対して getter を呼び出す必要があります。

注意：以前は、getName() メソッドおよび getResult() メソッドを呼び出すと、クエリーの結果セット、pdf オブジェクト、または ftp 接頭辞などのデータが返されていましたが、現在は、その代わりに、適切なプロパティセットを含むコンポーネントが返されるようになりました。

オブジェクト名は、タグ名から cf 接頭辞を取り除いたものになります (ftp など)。これらのオブジェクトは、子タグの機能 (cfmailpart や cfmailparam など) もサポートします。

注意：暗黙的な setter が使用されているときに cfmailpart や cfmailparam などの子タグが追加された場合、それらのタグは CFC 変数スコープ内に追加されるため、スレッドセーフティの問題が発生する可能性があります。したがって、サービスごとに新しいコンポーネントを作成することをお勧めします。属性の状態を保持する必要がある場合、初期化された属性値を保持するには、そのコンポーネントに対して duplicate() を使用します。

これらのタグを関数で使用するには :

1. サービスオブジェクトをインスタンス化します。
2. オブジェクトの属性と子タグを設定します。

3 オブジェクトに対してアクションを実行します。

注意：対応するタグとは異なり、これらの関数では、アプリケーション固有のパラメータを使用できません。使用できるパラメータは、ColdFusion で直接サポートされるパラメータのみです。

手順 1 : サービスオブジェクトのインスタンス化
関数オブジェクト (メールオブジェクトなど) を作成するには、次の例のように new 演算子または createobject() 関数を使用します。

```
myMail = new mail(server="sendmail.myCo.com");
```

手順 2a : 属性の管理
属性は次に示すいずれかの方法で設定できます。

- 次の例のように、オブジェクトをインスタンス化するときに、オブジェクトイニシャライザに対する name=value 形式のパラメータとして設定する。

  ```
  myMail = new mail(server="sendmail.myCo.com");
  ```

- 次の例のように、オブジェクトのアクションメソッドに対する name=value 形式のパラメータとして設定する。

  ```
  Q = myQuery.execute(sql="select * from art");
  ```

- 次の例のように、属性設定メソッドを使用する。

  ```
  myMail.setSubject("Hi");
  ```

  注意：getAttributeName 関数を使用して、AttributeName で指定された属性の値を取得することはできません。その場合は、代わりに GetAttributes(AttributeName) を使用します。

- 次の関数を使用する。

  ```
  SetAttributes(attrib1=value,attrib2=value,...);
  GetAttributes([attribName1[,attribName2]]....);
  ClearAttributes([attribName1[,attribName2]]...);
  ```

注意：ストアドプロシージャの result 属性を指定した場合に getPrefix() を呼び出すと、executionTime,statusCode,cached が返されます。result 属性を指定していない場合に getPrefix() を呼び出すと、executionTime と statusCode のみが返されます。

手順 2b: 子タグの操作の管理
すべてのサービスオブジェクトは、子タグを持つタグに対応します。たとえば、cfmail は、子タグとして cfmailpart および cfmailparam を持ちます。

子タグの機能を指定するには、次のメソッドを使用します。

- httpObj.addParam
- mailObj.addParam
- mailObj.addPart
- pdfObj.addParam
- queryObj.addParam
- storedProcObj.addParam
- storedProcObj.addProcResult

次に例を示します。

```
mailObj.addparam(file="#ExpandPath('test.txt')#");
mailObj.addPart(name="foo",type="html",charset="utf-8",
body="This is a test message.");
```

次の関数を呼び出して子タグの設定を消去することもできます。

- httpObj.clearParams
- mailObj.clearParams
- mailObj.clearParts
- pdfObj.clearParams
- queryObj.clearParams
- storedProcObj.clearParams
- storedProcObj.clearProcResults

オブジェクトに対して複数の add メソッドを使用した場合に clear メソッドを使用すると、すべてのメソッドで設定されたすべて値が消去されます。

手順 3 : サービスアクションの実行

cfmail と cfpdf を除くサービスタグには、結果を返すアクションがあります。cfpdf や cfftp などの一部のタグには、action 属性があります。これらのタグの場合、各アクションは、サービスオブジェクトに対するメソッドに対応します。たとえば、ftp オブジェクトのアクションメソッドには、open、close、listDir、getFile、rename など、さまざまなものがあります。ただし、サービスタグからデータを返す方法が変更されています。現在は、該当するプロパティセットを含むコンポーネントが返さるようになっているため、データにアクセスするにはそれらのプロパティに対して getter を呼び出す必要があります。

注意：PDF オブジェクトには 2 つのアクションメソッドがありますが、それらの名前は、対応する cfftp アクション属性の値 (getInfo および setInfo) とは異なり、getPDFInfo および setPDFInfo になっています。これは、PDF の info 属性の set メソッドおよび get メソッドと名前が競合しないようにするための措置です。

cfhttp、cfmail、cfquery、および cfstoredproc タグには、action 属性がありません。これらのタグは 1 種類のアクションのみを実行します。これらのアクションを cfscript で実行するには、次の関数を呼び出します。

- httpObj.send()
- mailObj.send()
- queryObj.execute()
- storedProcObj.execute()

通常、アクションの結果を取得するには、次の例のように、アクションメソッドの結果を変数に代入します。

```
Q = qry.execute(sql="select * from art");
```

注意：アクションに対して指定した属性は、そのアクションに対してのみ有効であり、アクションが完了すると消去されます。

サービスコードの例 : mail、ftp、および http

cfscript での mail、http、および ftp サービスの使用例を次に示します。

```
<!---mail and ftp service --->
<cfscript>
    m = new mail();
<!---mail service --->
    m.setTo("x@adobe.com");
<!---set attribute using implicit setter --->
    m.setSubject("Hi");
    m.setBody("test mail");
<!---users need to use 'body' to specify cfmail and cfmailpart content --->
    m.addparam(file="#E xpandPath('test.txt')#");
<!---add cfmail param tags --->
    m.addPart(type="html",charset="utf-8",body="some
    mailpart content");
<!---add cfmailpart tags --->
    m.send(to="y@abc.com" ....);
<!---attributes can be overriden when sending mail --->
    m.clear();
<!---clearAttributes(),clearParams() and clearParts() can also be used to clear --->
    individual items, if needed
<!---ftp service --->
    f = new ftp(server="s",username="u",password="p");
<!---check if a specified directory already exists (note the usage of getPrefix ())--->
    f.existsDir(directory ="some_dir").getPrefix().returnValue ? WriteOutput("Directory
    exists"):WriteOutput("Directory does not exist");
    <!---list directory contents (note the usage of getResult() and getPrefix() --->
    r = f.listDir(directory="some_dir",name="dirContents");
    dirContents = r.getResult();
    r.getPrefix().succeeded ? WriteOutput("List Directory operation successful") :
</cfscript>
<!---http service --->
<cfscript>
    httpObj = new http();
    <!---example 1 --->
        <!---add params--->
        httpObj.addParam(type="cgi", Name="Content-type", value =
        "application/x-www-form-urlencoded",encoded="no");
        httpObj.addParam(type="body",value="test1=value1&test2=
        value2&arraytest=value1&arraytest=value2");
    <!---assign the component returned to a variable--->
        r = httpObj.send(url="http://localhost:8500/
        project1/cfscript_test_files/thread-
        safe/http/_cfhttpparam_body.cfm",method="POST");
    <!---use getPrefix() to dump the cfhttp prefix --->
        writedump(r.getPrefix());
    <!---example 2 --->

    <!---using attributes that return a query --->
        r = httpObj.send(url="
        http://localhost:8500/language_enhancements_2/cfscript_test_files/thread-
        safe/http/vamsee.txt")",name="myqry", firstrowasheaders="no",method="GET");
    <!---dump result and name attributes data --->
        writedump(r.getPrefix());
        writedump(r.getResult());
</cfscript>
```

cfftp の場合、返されたコンポーネントに対して使用できる getter には次のものがあります。

- getPrefix()

  タグ接頭辞 cfftp を返します。これは、cfftp 操作の後に使用可能になる構造体です。

- getResult()

  action="listDir" である場合にのみ使用できます。

cfhttp の場合、返されたコンポーネントに対して使用できる getter には次のものがあります。

- getPrefix()

  タグの実行後に使用可能になる cfhttp 接頭辞 (構造体) を返します。

- getResult()

  columns、delimiter、firstrowasheaders、name、textQualifier など、クエリーオブジェクトを返すように ColdFusion に指示する属性が指定されている場合にのみ使用できます。

クエリーサービスの例

```
<cfscript>
    qryObj = new createObject("component","com.adobe.coldfuison.query").init();
    <!---r here is no longer the query recordset but a component --->
    r = qryObj.execute(sql="select * from art",
    datasource="cfdocexamples",result="myresult",name="myquery");
    <!---new way to access the data --->
    resultset =r.getResult();
    prefixData = r.getPrefix();
    writedump(resultset);
    writedump(prefixData);
    <!---Using QoQ--->
    qryObj.setAttributes(myquery=resultset);
    r = qryObj.execute(sql="select * from myquery", dbtype="query");
    writedump(r.getResult());
    writedump(r.getPrefix());
</cfscript>
```

返されたコンポーネントに対して使用できる getter には次のものがあります。

- getPrefix()

  クエリーの実行後に使用可能になる結果の構造体を返します。

- getResult()

  クエリー (SELECT クエリー) によって返された結果セットを返し、その他のタイプの SQL ステートメントまたはクエリー (INSERT、UPDATE、DELETE など) の場合はエラーを返します。

PDF の例

name 属性が指定されたアクションを実行すると、そのたびに新しい pdf がユーザーに返されます。

次のコードは、PDF での一般的なアクション示しています。

```
<cfscript>
    pdfObj = new pdf();
    x = pdfObj.read(source=#sourcefile#, name="PDFInfo");
    x = pdfObj.processddx(ddxfile="#tocddx#",inputfiles="#inputStruct#",outputfiles=
        "#outputStruct#",name="ddxVar");
    x = pdfObj.addWatermark(source="#pdf1#",image="#image1#", pages="1",
        overwrite="yes", name="test2");
    x = pdfObj.removewatermark(source="#pdf1#", name="temp");
    x = pdfObj.deletePages(source="#destfolder#dest.pdf",pages="2-4", name="deltest");
    pdfObj.addparam(source="#pdf1#", pages="1-2,4");
    pdfObj.merge(destination="#destfolder#merge-oneBigFile-5.pdf", overwrite="yes");
    pdfObj.thumbnail(source="#pdf1#", overwrite="yes");
    pdfObj.setInfo(source="#pdf1#", info="#{Author="Donald Duck"}#",
                    destination="#destfolder#pdfinfo.pdf", overwrite="yes");
    pdfObj.write(source="myBook", destination="#destfolder#write1.pdf", version="1.4",
                overwrite="yes");
    pdfObj.protect(source="MyPdfVar", password="adobe", permissions="none",
                    newuserpassword="newuserpw", newownerpassword="newownerpw");
</cfscript>
```

Storedproc の例

次のコードは、storedproc サービスオブジェクトの使用例を示しています。

```
<cfscript>
    sp = new storedproc();
    <!---add cfprocparam tags --->
        sp.addParam(TYPE = "IN", CFSQLTYPE="CF_SQL_VARCHAR", VALUE="David",
        DBVARNAME="@firstname");
        sp.addParam(TYPE="IN", CFSQLTYPE="CF_SQL_VARCHAR", VALUE="Peterson",
        DBVARNAME="@lastname", null ="yes");
        sp.add Param(TYPE="OUT", CFSQLTYPE="CF_SQL_INTEGER", variable="MyCount",
        DBVARN AME="@MyCount");
    <!---add cfprocresult  tags --->
        sp.addProcResult(NAME = "home r", RESULTSET = 1);
        sp.addProcResult( NAME = "home r2", RESULTSET = 2);
        sp.addProcResult(NAME = "home r3", RESULTSET = 3) ;
    <!---execute stored proc--->
        r = sp.execute(procedure="sp_weird",datasource="some_dsn",result="r");
        writedump(r.getProcResultSets());
    <!---changed from sp.getProcResults()--->
        writedump(r.getProcResultSets ("home r3"));
        writedump(r.getPrefix());
    <!---changed from sp.getResult()--->
        writedump(r.getProcOutVariables());
    <!---changed from sp.getProcVars()--->
</cfscript>
```

返されたコンポーネントに対して使用できる getter には次のものがあります。

- getPrefix()

  プロシージャの実行後に使用可能になる cfstoredproc 接頭辞 (構造体) を返します。

- getProcResultsets()

  ストアドプロシージャによって返された結果セットを返します。

- getProcOutVariables()

  プロシージャによって設定された OUT 変数または INOUT 変数を返します。

関数

関数は、通常、データを処理して結果を返します。また、ユーザー定義関数 (UDF) も作成できます。UDF はカスタム関数とも呼ばれます。

関数の一般的な形式を次に示します。

```
functionName([argument1[, argument2]]...)
```

Now 関数など、引数を持たない関数もあります。引数を持つ関数では、引数をカンマで区切って指定します。引数には必須のものとオプションのものがあります。すべての ColdFusion 関数は値を返します。たとえば、Round(3.14159) は値 3 を返します。

ビルトイン関数

ColdFusion のビルトイン関数を使用すれば、さまざまなタスクが行えます。その例を次に示します。

- 配列、リスト、構造体などの、複雑なデータ変数の作成および操作
- クエリーの作成および操作
- 文字列や日付時刻値の作成、解析、操作、および書式制御
- ダイナミックデータの値の評価
- 変数値の型の調査
- データの形式変換
- 数学演算の実行
- システム情報やリソースの取得

ColdFusion 関数のアルファベット順リストとカテゴリ別リストについては、『CFML リファレンス』の ColdFusion Functions を参照してください。

ビルトイン関数は、ColdFusion ページの任意の場所で使用できます。cfset タグや cfoutput タグでは、表示データの作成などのためにビルトイン関数がよく使用されます。たとえば次のコードは、October 24, 2007 の形式で今日の日付を表示します。

```
<cfoutput>#DateFormat(Now(), "mmmm d, yyyy")#</cfoutput>
```

このコードでは 2 つの関数をネストしています。Now 関数は、現在の日時を表す ColdFusion 日付時刻値を返します。DateFormat 関数は、Now 関数で返された値を指定の文字列表記に変換します。

関数は、CFScript スクリプトでも役立ちます。CFScript の中で ColdFusion タグを使用することはできないので、スクリプトで ColdFusion の機能にアクセスするには関数を使用する必要があります。

暗黙的な Get 関数と Set 関数

ColdFusion コンポーネントは、パブリックの setter および getter メソッドを持つプライベートプロパティをサポートします。この動作を利用してコンポーネントプロパティへの直接アクセスを防止することにより、オブジェクト指向プログラミングがサポートされます。

たとえば、myCFC コンポーネントの MyProp プロパティを設定して取得するには、次のコードを使用します。

```
myCFC.setMyProp(27);
theProp = myCFC.getMyProp();
```

setter メソッドと getter メソッドを持つプロパティの特徴を次に示します。

- set メソッドで代入されたコンポーネントプロパティは、CFC に対してプライベートな Variables スコープに属します。get メソッドまたは set メソッドを呼び出す以外にプロパティを取得または再設定する方法はありません。

- プロパティに type 属性の値が含まれる場合は、setter 関数に渡されるデータが ColdFusion によって検証されます。default 属性はプロパティに影響を与えず、プロパティの初期値は設定されません。
- 直接代入ステートメント (myCFC.MyProp=27 など) を使用すると、cfproperty タグでプロパティが指定されている場合でも、標準の This スコープ変数が CFC に作成されます。This スコープ変数は、set メソッドと get メソッドを使用してアクセスするプロパティから独立しています。ただし、This スコープ変数の名前は、set メソッドと get メソッドを使用してアクセスするプロパティと同じ名前にすることができます。
- CFC 外部からのプロパティへのアクセスを制御するには、cfproperty タグの getter 属性と setter 属性を使用します。

  setter 属性の値を true に設定すると、アプリケーションコードによるプロパティの設定が許可されます (デフォルトの動作)。

  値を false に設定すると、CFC 内からのプロパティ設定のみが許可されます。getter 属性も同様に機能します。
- 明示的な set メソッドまたは get メソッドは、暗黙的な set メソッドおよび get メソッドよりも優先されます。したがって、CFC に明示的な setMyProp メソッドを持つ MyProp プロパティがある場合、アプリケーションコードで setMyProp() 関数を呼び出すと、暗黙的な関数ではなく、コード内で呼び出した関数が使用されます。

validate 属性と validateparams 属性

<cfproperty> の validate 属性では、このプロパティに対して暗黙的な setter が呼び出されたときにデータの検証に使用するバリデータを指定します。指定可能なバリデータには次のものがあります。

- string
- boolean
- integer
- numeric
- date
- time
- creditcard: mod10 アルゴリズムに準拠する 13 ～ 16 桁の数値です。
- email: 有効な電子メールアドレスです。
- eurodate: 日付時刻値です。いずれの日付の部分も dd/mm/yy の形式でなければなりません。区切り文字として /、-、または . を使用できます。
- regex: validateparams で指定されたパターンと入力を照合します。
- ssn: 米国の社会保障番号です。
- telephone: 米国の標準の電話番号です。
- UUID: Home Universally Unique Identifier (ユニバーサル固有識別子) です。形式は 'XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXXXXXX' で、'X' は 16 進数です。
- guid: Universally Unique Identifier (ユニバーサル固有識別子) です。形式は "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX" で、'X' は 16 進数です。
- zipcode: 米国の 5 桁または 9 桁形式の郵便番号です。

<cfproperty> の validateparams 属性では、validate 属性で指定されたバリデータに必要なパラメータを指定します。これは、暗黙的な構造体表記法で指定する必要があります。

- min: validate が integer/numeric/ である場合の最小値
- max: validate が integer/numeric/ である場合の最大値
- minLength: validate が string である場合の文字列の最小長

- maxLength: validate が string である場合の文字列の最大長
- pattern: validate 属性で指定されたバリデータが regex である場合の正規表現式

たとえば、次のコードは、従業員の電子メール、郵便番号、および年齢に対するバリデータを設定しています。

注意：年齢に関しては、値が整数であるかどうかが validate で検査され、その値が指定された範囲内にあるかどうかが validateparams によって検査されます。

```
<!---Setting validators for an employee's e-mail, age, and zipcode--->
<cfcomponent>
<cfproperty name="mail" validate="email">
<cfproperty name="zip" validate="zipcode">
<cfproperty name="age" validate="integer" validateparams="{min=18,max=60}"> </cfcomponent>
```

ユーザー定義関数

ユーザーが独自に作成した関数はユーザー定義関数 (UDF) と呼ばれます。作成した関数は、ColdFusion 式や CFScript で使用できます。UDF は、ビルトイン CFML 関数を使用できる任意の場所で呼び出せます。UDF を作成するには、cffunction タグまたは CFScript の function ステートメントを使用します。cffunction タグを使用して作成する UDF には、ColdFusion タグおよび関数を含めることができます。CFScript で作成する UDF には、関数のみを含めることができます。UDF はスタンドアローンの関数として作成することも、ColdFusion コンポーネント内にカプセル化することもできます。

ユーザー定義関数を使用すれば、頻繁に使用するロジックやオペレーションを 1 つのユニットにカプセル化できます。一度記述したコードは繰り返し使用できます。UDF を使用することで、コーディングの一貫性を維持し、CFML を効率よく構造化できます。

ユーザー定義関数の例としては、対数計算などの数学ルーチン、数字の金額を "2 ドル 3 セント " などの文字列に変換する文字列操作ルーチン、暗号化や復号化を行うルーチンなどがあります。

注意：www.cflib.org の Common Function Library Project には、ユーザー定義関数の無償ライブラリがあります。

ユーザー定義関数の詳細については、150 ページの「ユーザー定義関数の作成と呼び出し」を参照してください。

ColdFusion コンポーネント

ColdFusion コンポーネントは、複数の関連する関数をカプセル化します。ColdFusion コンポーネントは、関連するユーザー定義関数や変数の集まりであり、コンポーネントコンテンツへのアクセスを提供および管理するための機能も備えています。ColdFusion コンポーネントでは、データをプライベートにすることができます。プライベートに設定したデータは、コンポーネント内のどの関数 (メソッド) からもアクセスできますが、そのコンポーネントを使用するアプリケーションからはアクセスできません。

ColdFusion コンポーネントには、次の機能があります。

- 関連するサービスを 1 つのユニットで提供できます。
- Web サービスを作成して、インターネットで公開できます。
- Flash クライアントから直接呼び出せる ColdFusion サービスを提供できます。
- データの隠蔽、継承、パッケージ、イントロスペクションなど、オブジェクト指向プログラミングの特徴的な機能が使用できます。

ColdFusion コンポーネントの使用方法の詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

定数

定数の値は、プログラムの実行中に変化しません。定数は "Robert Trent Jones" や 123.45 などの単純なスカラー値で、式や関数の中で使用できます。定数には、整数、実数、日付時刻値、ブール値、テキスト文字列などがあります。ColdFusion では、定数に名前を付けることはできません。

変数

変数は、ColdFusion 式でオペランドとしてよく使用されます。変数値は設定および再設定が可能で、属性として CFML タグに渡したり、パラメータとして関数に渡したりすることができます。多くの場合、定数の代わりに変数を使用することができます。

ColdFusion には、サーバーに関する情報を提供するさまざまなビルトイン変数が用意されています。これらは ColdFusion タグによって返されます。ColdFusion のビルトイン変数のリストについては、『CFML リファレンス』の Reserved Words and Variables を参照してください。

変数は、次の 2 つの特性によって分類できます。

- 変数のスコープ。情報が利用可能な場所、および変数の継続期間を示します。
- 変数値のデータ型。数値、文字列、日付などの、変数が表す情報の種類を示します。

データ型の一覧については、28 ページの「データ型」を参照してください (データ型は定数値にも適用されます)。データ型、スコープ、それらの使用方法などの ColdFusion 変数の詳細については、34 ページの「ColdFusion 変数の使用」を参照してください。

式

ColdFusion の式は、オペランドと演算子から構成されます。オペランドは、"Hello" や MyVariable などの定数や変数です。文字列連結演算子 (&)、除算演算子 (/) などの演算子は、オペランドに作用する動詞です。ColdFusion 関数も演算子のように作用します。

最も基本的な式は単一のオペランドで構成され、演算子がありません。複雑な式は複数のオペランドと演算子で構成されます。たとえば、次のステートメントはすべて ColdFusion 式です。

```
 12
MyVariable
(1 + 1)/2
"father" & "Mother"
Form.divisor/Form.dividend
Round(3.14159)
```

変数の使用方法の詳細については、34 ページの「ColdFusion 変数の使用」を参照してください。式および演算子の詳細については、58 ページの「式と # 記号の使用」を参照してください。

データ型

ColdFusion では変数のデータ型を明示的に宣言しないので、タイプレスな環境であるように見えます。

しかし実際は、ColdFusion のデータ (変数が表すデータや定数) にはデータ型があります。コンピュータでデータが格納される方法は、データ型によって異なります。

ColdFusion データ型は次のカテゴリに分類されます。

カテゴリ	説明および型
単純	単一の値を表します。単純データ型は ColdFusion 式で直接使用できます。ColdFusion の単純データ型には次のものがあります。 • 文字列 : 単一引用符 (') または二重引用符 (") で囲んだ一連の英数字 ("This is a test" など) • 整数 : 引用符で囲まれていない一連の数字 (356 など) • 実数 (-3.14159 など) • ブール値 : True、Yes、1 で真を表し、False、No、0 で偽を表します。ブール値の大文字と小文字は区別されません。 • 日付時刻値 : ColdFusion ではさまざまな日付形式がサポートされています。詳細については、39 ページの「日時の形式」を参照してください。
複合	データのコンテナ。複合変数は通常、複数の値を表します。ColdFusion のビルトイン複合データ型には次のものがあります。 • 配列 • 構造体 • クエリー
バイナリ	GIF ファイルや実行可能プログラムファイルの内容などの生データ。
オブジェクト	COM、CORBA、Java、Web サービス、および ColdFusion コンポーネントオブジェクト。cfobject タグやその他の専用タグを使用して作成およびアクセスする複雑なオブジェクト。

注意 : ColdFusion には、無限精度の 10 進数を表すデータ型は用意されていませんが、そのような数値は文字列として表現でき、無限精度の 10 進演算をサポートする関数も用意されています。詳細については、『CFML リファレンス』の PrecisionEvaluate を参照してください。

ColdFusion のデータ型の詳細については、34 ページの「ColdFusion 変数の使用」を参照してください。

フロー制御

ColdFusion には、ページの実行方法を制御するためのタグが用意されています。これらのタグは、if、then、else などのプログラミング言語のフロー制御ステートメントに対応します。ColdFusion のフロー制御タグには次のものがあります。

タグ	用途
cfif、cfelseif、cfelse	式が True であるか False であるかに基づいて、実行するコードセクションを選択します。
cfswitch、cfcase、cfdefaultcase	式の値に基づいて、実行するコードセクションを選択します。case では、True、False 以外の条件判断も行うことができます。
cfloop、cfbreak	リストのエントリ、構造体または外部オブジェクトのキー、クエリー列のエントリ、インデックス、条件式の値などに基づいて、コードをループします。
cfabort、cfexit	ColdFusion ページまたはカスタムタグの処理を終了します。

CFScript にも一連のフロー制御ステートメントが用意されています。CFScript でのフロー制御ステートメントの使用方法については、98 ページの「CFML スクリプト言語による ColdFusion ページの拡張」を参照してください。フロー制御タグの使用方法の詳細については、『CFML リファレンス』の該当するタグに関するリファレンスページを参照してください。

cfif、cfelseif、および cfelse

cfif、cfelseif、および cfelse タグでは、次のように if-then-else 条件が処理されます。

1. cfif タグで条件がテストされ、True の場合は本文が実行されます。
2. 前の cfif または cfelseif のテスト条件が False であった場合は、cfelseif タグで別の条件がテストされ、その条件が True の場合はその本文が実行されます。
3. cfelse タグはオプションで、cfif タグおよび 0 個以上の cfelseif タグの後に記述します。前にあるタグのテスト条件がすべて False であった場合に、その本文が実行されます。

   次の例に、cfif、cfelseif、および cfelse タグの使用方法を示します。type 変数の値が "Date" の場合は日付が表示されます。"Time" の場合は時刻が表示されます。それ以外の場合は時刻と日付の両方が表示されます。

```
<cfif type IS "Date">
    <cfoutput>#DateFormat(Now())#</cfoutput>
<cfelseif type IS "Time">
    <cfoutput>#TimeFormat(Now())#</cfoutput>
<cfelse>
    <cfoutput>#TimeFormat(Now())#, #DateFormat(Now())#</cfoutput>
</cfif>
```

cfswitch、cfcase、および cfdefaultcase

cfswitch、cfcase、および cfdefaultcase タグを使用すると、式の値に基づいてさまざまなコードブロックを選択できます。これらのタグは次のように処理されます。

1. cfswitch タグによって式が評価されます。cfswitch タグ本文には、1 つまたは複数の cfcase タグが含まれ、オプションで cfdefaultcase タグが含まれています。
2. cfswitch タグ本文のそれぞれの cfcase タグでは値が指定されています。cfswitch タグの式の評価値に一致する値が指定されている cfcase タグの本文コードが実行され、cfswitch タグが終了されます。2 つの cfcase タグで同じ条件が指定されている場合は、エラーが発生します。
3. cfswitch タグの評価値に一致する cfcase タグがなく、cfswitch タグの本文に cfdefaultcase タグが含まれている場合は、cfdefaultcase タグの本文コードが実行されます。

注意：cfdefaultcase タグは常に必要なわけではありませんが、cfcase タグを使用するときは cfswitch ステートメントの最後にこのタグを記述するのがよいプログラミングスタイルといえます。

cfif タグで複数の cfelseif タグを指定するよりも、cfswitch タグを使用するほうがパフォーマンスおよびわかりやすさの点で優れています。switch 処理は通常、月やリクエスト識別子などの文字列変数に基づいてアクションを選択する場合に使用します。

switch 処理の例を次に示します。

```
<cfoutput query = "GetEmployees">
<cfswitch expression = #Department#>
    <cfcase value = "Sales">
        #FirstName# #LastName# is in <b>Sales</b><br><br>
    </cfcase>
    <cfcase value = "Accounting">
        #FirstName# #LastName# is in <b>Accounting</b><br><br>
    </cfcase>
    <cfcase value = "Administration">
        #FirstName# #LastName# is in <b>Administration</b><br><br>
    </cfcase>
    <cfdefaultcase>#FirstName# #LastName# is not in Sales,
        Accounting, or Administration.<br>
    </cfdefaultcase>
</cfswitch>
</cfoutput>
```

cfloop と cfbreak

cfloop タグは、タグの属性で指定された条件に基づいてタグ本文を 0 回以上ループします。cfbreak タグは cfloop タグを終了します。

cfloop

cfloop タグには次のタイプのループがあります。

ループタイプ	説明
インデックス作成	タグ本文をループし、ループするたびに指定の数だけカウンタ変数をインクリメントして、カウンタが指定の値になるまで続行します。
条件演算	条件を確認して、条件が True の場合にタグ本文を実行します。
クエリー	クエリー内の行ごとにタグ本文を 1 回ループします。
リスト、ファイル、または配列	リストのエントリ、ファイルの行、または配列の項目ごとにタグ本文を 1 回ループします。
コレクション	ColdFusion 構造体内のキーまたは COM/DCOM オブジェクト内の項目ごとにタグ本文を 1 回ループします。

簡単なインデックスループの例を次に示します。

```
 <cfloop index = "LoopCount" from = 1 to = 5>
The loop index is <cfoutput>#LoopCount#</cfoutput>.<br>
</cfloop>
```

簡単な条件ループの例を次に示します。このコードは次の操作を行います。

1. 4 番目のエントリに "kumquats" という単語が含まれている 10 要素からなる配列を設定します。
2. "kumquats" が含まれている配列要素を検出するか配列の最後に到達するまで、配列をループします。
3. "**kumquats**" を検出したかどうかを示すブール変数の値と、ループを終了したときの配列インデックスを出力します。

```
 <cfset myArray = ArrayNew(1)>
<!--- Use ArraySet to initialize the first ten elements to 123 --->
<cfset ArraySet(myArray, 1, 10, 123)>
<cfset myArray[4] = "kumquats">

<cfset foundit = False>
<cfset i = 0>
<cfloop condition = "(NOT foundit) AND (i LT ArrayLen(myArray))">
    <cfset i = i + 1>
    <cfif myArray[i] IS "kumquats">
        <cfset foundit = True>
    </cfif>
</cfloop>
<cfoutput>
i is #i#<br>
foundit is #foundit#<br>
</cfoutput>
```

注意：終了条件を指定しないと、無限条件ループに陥ります。この例では、<cfset i = i + 1> ステートメントがないと無限ループになります。無限ループを終了するには、ColdFusion アプリケーションサーバーを停止する必要があります。

cfbreak

cfbreak タグは cfloop タグを終了します。特定の条件が満たされたときにループを終了するには、cfif タグの中でこのタグを使用します。クエリーループでの cfbreak タグの使用例を次に示します。

```
<cfloop query="fruitOrder">
    <cfif fruit IS "kumquat">
        <cfoutput>You cannot order kumquats!<br></cfoutput>
        <cfbreak>
    </cfif>
    <cfoutput>You have ordered #quantity# #fruit#.<br></cfoutput>
</cfloop>
```

cfabort と cfexit

cfabort タグは、その位置で現在のページの処理を停止します。cfabort タグの前までに処理されたすべての内容が、ユーザーまたは呼び出したタグに返されます。表示するエラーメッセージを指定することもできます。特定の状態 (エラーなど) が発生した場合にページの処理を停止するには、cfif タグの本文に cfabort タグを記述します。

cfexit タグは、カスタムタグの処理を制御します。このタグは、ColdFusion カスタムタグの中でのみ使用できます。詳細については、216 ページの「タグ実行の終了」および『CFML リファレンス』を参照してください。

大文字と小文字の区別

ColdFusion では大文字と小文字が区別されません。たとえば、cfset、CFSET、CFSet、および cfsEt はすべて cfset タグを表します。ただし、プログラム内では大文字と小文字の使用方法を統一することをお勧めします。たとえば、次のようにします。

- 大文字と小文字の使用方法を定めて統一します。いくつかのタグ名を小文字で記述した場合は、すべてのタグ名を小文字で記述します。
- 1 つの変数では大文字と小文字の使用方法を統一します。たとえば、1 つのページの中で同じ変数を myvariable と記述したり MyVariable と記述したりすることは避けます。

  大文字と小文字が区別される言語 (JavaScript など) を CFML とともに使用する場合は、アプリケーションページでエラーが発生しないように、これらのルールに従ってください。

特殊文字

ColdFusion では、二重引用符 (")、単一引用符 (')、およびシャープ記号 (#) には特別な意味があります。これらの文字を文字列に含める場合は、その文字を 2 回入力します。たとえば、## は 1 つの # 文字を表します。

単一引用符 (') や二重引用符 (") をエスケープする必要があるかどうかは、場合によって異なります。二重引用符 (") の文字列内では、単一引用符やアポストロフィ (') をエスケープする必要はありません。単一引用符 (') の文字列内では、二重引用符 (") をエスケープする必要はありません。

引用符 (') と二重引用符 (") を同時に使用した場合の特殊文字のエスケープの例を次に示します。

```
<cfset mystring = "We all said ""Happy birthday to you.""">
<cfset mystring2 = 'Then we said "How old are you now?"'>
<cfoutput>
    #mystring#<br>
    #mystring2#<br>
    Here is a number sign: ##
</cfoutput>
```

出力は次のようになります。

```
We all said "Happy birthday to you."
Then we said "How old are you now?"
Here is a number sign: #
```

予約語

他のプログラミングツールと同様に、ColdFusion の変数、UDF、およびカスタムタグにも、使用できない単語や名前があります。ColdFusion の要素と混同するような名前は避けてください。たとえばビルトイン構造体名など、ColdFusion で予約されている単語を使用すると、ColdFusion データが上書きされることがあります。

ColdFusion の変数、ユーザー定義関数名、およびカスタムタグ名に使用できない単語を次に示します。場合によっては、これらの単語を使用しても問題ないことがありますが、エラーを避けるために一切使用しないことをお勧めします。すべての予約語のリストについては、『CFML リファレンス』を参照してください。

- Now、Hash などのビルトイン関数名
- Form、Session などのスコープ名
- cf で始まる任意の名前。ただし、CFML カスタムタグを直接呼び出す場合は、カスタムタグページ名の前に cf_ を付けます。
- NE、IS などの演算子
- Error、File などのビルトインデータ構造体の名前
- RecordCount、CGI 変数名などのビルトイン変数の名前
- for、default、continue などの CFScript 言語の要素名

また、次の語句のいずれかで終わるフォームフィールド名も作成しないでください。ただし、非表示フォームフィールド名を使用してフォームフィールドの検証ルールを指定する場合は例外です。フォームフィールド検証の詳細については、692 ページの「データの取得および形式設定の概要」を参照してください。

- _integer
- _float
- _range
- _date
- _time
- _eurodate

ColdFusion では大文字と小文字が区別されないので、IS、Is、iS、is はすべて予約語です。

CFScript

CFScript は言語内の言語です。CFScript は、JavaScript に似たスクリプト言語ですが、JavaScript よりも簡単に使用できます。また、JavaScript と異なり、CFScript は ColdFusion サーバーでのみ動作し、クライアントシステムでは動作しません。CFScript では、ColdFusion のすべての関数と、スクリプトのスコープで使用できるすべての変数を使用できます。

CFScript を使用すると、ColdFusion のロジックを簡潔かつ効果的に記述できます。CFScript は、次のような目的でよく使用されます。

- 変数設定の簡略化と高速化
- コンパクトなフロー制御構造の構築
- ユーザー定義関数へのビジネスロジックのカプセル化

次のサンプルスクリプトでは、配列に値を設定して、"key" で始まる最初の配列エントリを検索します。このコードでは、変数の設定、ループ構造、スクリプトコードブロック、関数呼び出しなどを CFScript で記述する方法が示されています。また、結果を表示するために cfoutput タグを使用しています。出力に CFScript を使用することもできますが、通常は cfoutput タグのほうが簡単です。

```
 <cfscript>
strings = ArrayNew(1);
strings[1]="the";
strings[2]="key to our";
strings[4]="idea";
for( i=1 ; i LE 4 ; i = i+1 )
{
    if(Find("key",strings[i],1))
        break; }
</cfscript>
<cfoutput>Entry #i# starts with "key"</cfoutput><br>
```

CFScript を使用してユーザー定義関数を作成することができます。

CFScript の詳細については、98 ページの「CFML スクリプト言語による ColdFusion ページの拡張」を参照してください。ユーザー定義関数の詳細については、150 ページの「ユーザー定義関数の作成と呼び出し」を参照してください。

ColdFusion 変数の使用

Adobe ColdFusion 変数は、ColdFusion 式でオペランドとしてよく使用されます。変数値は設定および再設定が可能で、属性として CFML タグに渡したり、パラメータとして関数に渡したりすることができます。多くの場合、定数の代わりに変数を使用することができます。

ColdFusion 変数を作成して使用するには、次の方法を知っておく必要があります。

- さまざまなデータ型を変数で表す方法
- データ型を変換する方法
- 変数が存在するさまざまなスコープ
- スコープの使用方法
- 変数を正しく使用する方法

変数の作成

ほとんどの ColdFusion 変数は、変数に値を代入することによって作成します。ただし、配列を作成するには ArrayNew 関数を使用する必要があります。通常、変数を作成するには cfset タグを使用します。また、cfparam タグを使用したり、CFScript で代入ステートメントを使用したりすることもできます。データオブジェクトを作成するタグでも変数が作成されます。たとえば、cfquery タグではクエリーオブジェクト変数が作成されます。

ColdFusion では、特定のタグや演算の結果に関する情報を提供するいくつかの変数が自動的に作成されます。また、Client や Server などの特定のスコープでも自動的に変数が生成されます。これらの特殊な変数の詳細については、『CFML リファレンス』の Reserved Words and Variables や、これらの変数を作成する CFML タグの説明を参照してください。

まだ作成していない変数を使用しようとするとエラーが発生します。たとえば、入力が完了していないフォームのデータを処理するとエラーが発生することがあります。そのようなエラーを防ぐために、変数を使用する前にその変数が存在するかどうかをテストします。変数の存在テストの詳細については、55 ページの「変数の存在の確認」を参照してください。

変数の作成方法の詳細については、52 ページの「スコープでの変数の作成および使用」を参照してください。

変数のネーミング規則

フォームフィールド名、カスタム関数、ColdFusion コンポーネントの引数名などの ColdFusion の変数名は、Java ネーミング規則および次のガイドラインに従っている必要があります。

- 変数名は文字、アンダースコア、または Unicode 通貨記号で始まる必要があります。

- 2 文字目以降には、文字、数字、アンダースコア文字、および Unicode 通貨記号を何文字でも続けることができます。
- 変数名にスペースを含めることはできません。
- クエリー結果は変数の一種であるため、同じ名前のローカル変数は上書きされます。
- ColdFusion 変数では大文字と小文字は区別されません。ただし、大文字の使用方法を統一するとコードが読みやすくなります。
- クエリーに使用するフィールドをフォームに含めるときは、対応するデータベースフィールド名と同じフォームフィールド名にします。
- 構造体のコンポーネントやオブジェクト名はピリオドで区切ります。変数スコープと変数名もピリオドで区切ります。Cookie および Client スコープ内の変数を除き、単純変数の名前にはピリオドを使用できません。ピリオドの使用方法の詳細については、44 ページの「変数の参照でのピリオドの使用」を参照してください。

次のルールは、変数名には適用されますが、フォームフィールドおよび引数名には適用されません。

- 変数名の前には、その変数のスコープを接頭辞として付けます。ローカル変数名の Variables 接頭辞は省略可能ですが、その他のスコープについては、すべて接頭辞を付けてください。スコープ接頭辞を使用すると変数名がより明確になり、コードの効率が高まります。多くの場合、スコープ接頭辞は必須です。詳細については、51 ページの「スコープについて」を参照してください。

  注意：場合によっては、既存の変数名をシャープ記号 (#) で囲むことが必要になります。これは ColdFusion が文字列や HTML テキストから変数を区別し、変数名ではなくその値を挿入するために必要です。詳細については、63 ページの「# 記号の使用」を参照してください。

変数の特性

変数は次の特性によって分類できます。

- 変数値のデータ型。数値、文字列、日付などの、変数が表す情報の種類を示します。
- 変数のスコープ。情報が利用可能な場所、および変数の継続期間を示します。

データ型

ColdFusion では、ユーザーが変数に型を割り当てたり、ColdFusion が変数名に型を関連付けたりすることはないので、タイプレスな環境であるように見えます。しかし実際は、変数が表すデータには型があります。データ型は、式の評価や関数の引数の評価に影響を与えます。ColdFusion では、式を評価するときに多くのデータ型を他のデータ型に自動的に変換できます。数値や文字列などの単純データの場合は、変数が式の中や関数の引数として使用されるまでデータ型は問題になりません。

ColdFusion 変数のデータ型は、次のいずれかのカテゴリに属しています。

単純 単一の値。ColdFusion の式で直接使用できます。数値、文字列、ブール値、日付時刻値などがあります。

バイナリ GIF ファイルや実行可能プログラムファイルの内容などの生データ。

複合 データのコンテナ。通常、複数の値を表します。ColdFusion のビルトイン複合データ型には、配列、構造体、クエリー、XML ドキュメントオブジェクトなどがあります。

配列などの複合変数は ColdFusion 式で直接使用できませんが、複合変数の単純データ型要素は式で使用できます。

たとえば、myArray という数値の 1 次元配列がある場合、式 myArray * 5 は使用できませんが、配列の 3 番目の要素に 5 を乗算する式 myArray[3] * 5 は使用できます。

オブジェクト 複合構造。データと関数演算の両方がカプセル化されているのが一般的です。次の表に、ColdFusion で使用可能なオブジェクトのタイプと、使用方法の参照先を示します。

オブジェクトのタイプ	参照先
COM (Component Object Model)	1158 ページの「CFML アプリケーションでの COM および CORBA オブジェクトの統合」
CORBA (Common Object Request Broker Architecture)	1158 ページの「CFML アプリケーションでの COM および CORBA オブジェクトの統合」
Java	1109 ページの「CFML アプリケーションへの J2EE および Java 要素の統合」
ColdFusion コンポーネント	172 ページの「ColdFusion コンポーネントの構築と使用」
Web サービス	1061 ページの「Web サービスの使用」

データ型に関する注意

ColdFusion 変数には型が割り当てられませんが、変数に格納されているデータの型という意味で、便宜上 " 変数の型 " という表現を使用します。

ColdFusion では、フォームフィールドおよびクエリーパラメータに含まれているデータの型を検証できます。詳細については、697 ページの「変数の存在のテスト」および 419 ページの「cfqueryparam の使用」を参照してください。

cfdump タグは、変数 (ColdFusion の複合データ構造体も含む) の内容をすべて表示します。このタグは、複合データおよびそのデータを処理するコードのデバッグに便利です。

ColdFusion には、変数のデータ型を識別するための次の関数が用意されています。

- IsArray
- IsBinary
- IsBoolean
- IsImage
- IsNumericDate
- IsObject
- IsPDFObject
- IsQuery
- IsSimpleValue
- IsStruct
- IsXmlDoc

また、文字列を他のデータ型で表したり変換したりできるかどうかを判断する次の関数も用意されています。

- IsDate
- IsNumeric
- IsXML

ColdFusion では、null データ型は使用されません。ただし、データベースや Java オブジェクトなどの外部ソースから null 値を受け取った場合は、単純値として使用されるまで null 値が維持され、使用時に空の文字列 ("") に変換されます。また、Java オブジェクトを呼び出す際に JavaCast 関数を使用して、ColdFusion の空の文字列を Java の null に変換することができます。

数値

ColdFusion では整数と実数がサポートされています。式には整数と実数を混在させることができます。たとえば、1.2 + 3 は 4.2 になります。

整数

ColdFusion では、-2,147,483,648 ～ 2,147,483,647 (32 ビット符号付き整数) の範囲の整数がサポートされています。この範囲外の値を変数に代入することもできますが、その場合は最初に文字列として保管されます。数式でその文字列を使用すると、その値を保持したまま浮動小数点数値に変換されますが、次の例のように精度が低くなります。

```
<cfset mybignum=12345678901234567890>
<cfset mybignumtimes10=(mybignum * 10)>
<cfoutput>mybignum is: #mybignum#</cfoutput><br>
<cfoutput>mybignumtimes10 is: #mybignumtimes10# </cfoutput><br>
```

この例では次の出力が生成されます。

mybignum is: 12345678901234567890

mybignumtimes10 is: 1.23456789012E+020

実数

実数は小数部がある数値であり、浮動小数点数とも呼ばれます。ColdFusion の実数の範囲は、約 -10300 ～ 10300 です。実数の有効数字は最大 12 桁です。整数と同様に、変数に 13 桁以上の値を代入することもできますが、その場合は文字列としてデータが保管されます。その文字列を数式で使用すると、文字列が実数に変換されるので、精度が低くなることがあります。

実数は指数表現で表すことができます。その形式は **x**E**y** です。**x** は 1.0 以上 10 未満の正または負の実数で、**y** は整数です。指数表現の数値は、x の 10y 倍を表します。たとえば、$4.0E^2$ は、4 の 10^2 倍、つまり 400 になります。同様に、$2.5E^{-2}$ は、2.5 の 10^{-2} 倍、つまり 0.025 になります。指数表現は、絶対値が非常に大きな数値や小さな数値を表現するときに役立ちます。

BigDecimal 数

ColdFusion には、任意の長さの 10 進数 (1234567890987564.234678503059281 など) を表す特殊な BigDecimal データ型は用意されていません。このような数値は文字列として表現されます。ただし、BigDecimal 値を含む数式を受け取って計算し、結果の BigDecimal 値を文字列で返すことができる PrecisionEvaluate 関数があります。詳細については、『CFML リファレンス』の PrecisionEvaluate を参照してください。

文字列

ColdFusion では、テキスト値は文字列に保管されます。文字列を指定するには、文字列を単一引用符 (') または二重引用符 (") で囲みます。たとえば、次の 2 つの文字列は同じです。

" これは文字列です "

' これは文字列です '

空の文字列を表現する場合は、次の方法を使用します。

- "" (間に何もない二重引用符のペア)
- ' (間に何もない単一引用符のペア)

文字列のサイズは、ColdFusion サーバーで利用可能なメモリの制限を受ける他は自由です。ただし、長いテキストの取得 (CLOB) に対するデフォルトのサイズ制限は 64K です。ColdFusion Administrator を使用してデータベース文字列転送のサイズ制限を引き上げることは可能ですが、それによってサーバーのパフォーマンスが低下するおそれがあります。この制限を変更するには、データソースの [詳細設定] ページにある長いテキストの取得を有効にするオプションをオンにします。

引用符と # 記号のエスケープ

単一引用符 (') で囲まれている文字列に単一引用符を含めるには、2 つの単一引用符を使用します。これが単一引用符のエスケープです。次の例では、エスケープされた単一引用符を使用しています。

```
<cfset myString='This is a single-quotation mark: '' This is a double-quotation mark: "'>
<cfoutput>#mystring#</cfoutput><br>
```

二重引用符 (") で囲まれている文字列に二重引用符を含めるには、2 つの二重引用符を使用します。これが二重引用符のエスケープです。次の例では、エスケープされた二重引用符を使用しています。

```
<cfset myString="This is a single-quotation mark: ' This is a double-quotation mark: """>
<cfoutput>#mystring#</cfoutput><br>
```

文字列は二重引用符 (") または単一引用符 (') のどちらでも囲めるので、この 2 つの例では次に示す同じテキストが表示されます。

This is a single-quotation mark: ' This is a double-quotation mark: "

文字列に # 記号を挿入するには、次のように # 記号をエスケープする必要があります。

```
"This is a number sign ##"
```

リスト

ColdFusion にはリストを操作する関数はありますが、リストデータ型はありません。ColdFusion のリストは、複数のエントリを区切り文字で区切った文字列として表されます。

リストのデフォルトの区切り文字はカンマ (,) です。他の文字を使用してリスト要素を区切る場合は、リスト関数で区切り文字を指定する必要があります。複数の区切り文字を指定することもできます。たとえば、次のように、カンマまたはセミコロンを区切り文字と解釈するように指定できます。

```
<cfset MyList="1,2;3,4;5">
<cfoutput>
List length using ; and , as delimiters: #listlen(Mylist, ";,")#<br>
List length using only , as a delimiter: #listlen(Mylist)#<br>
</cfoutput>
```

この例を実行すると、次の出力が表示されます。

; と , を区切り文字に指定した場合のリストの長さ : 5

, のみを区切り文字に指定した場合のリストの長さ : 3

区切り文字は 1 文字であることが必要です。たとえば、2 つの連続したハイフンを区切り文字に指定することはできません。

リスト内で 2 つの区切り文字が連続している場合、空の要素は無視されます。たとえば、MyList が "1,2,,3,,4,,,5" であり、区切り文字がカンマである場合は、リストの要素数は 5 つと見なされ、リスト関数では "1,2,3,4,5" として処理されます。

ブール値

ブール値は、真または偽を表します。ColdFusion には、これらの値を表す 2 つの特殊な定数 True および False があります。たとえば、ブール式 1 IS 1 は True と評価されます。式 "Monkey" CONTAINS "Money" は False と評価されます。

次の例のように、式の中でブール定数を直接使用できます。

```
<cfset UserHasBeenHere = True>
```

ブール式では、True、0 以外の数値、および文字列 "Yes"、"1"、"True" はすべて真を表し、False、0、および文字列 "No"、"0"、"False" はすべて偽を表します。

ブール値の評価では大文字と小文字は区別されません。たとえば、True、TRUE、および true はすべて同じです。

日付時刻値

ColdFusion では日付時刻値の演算が行えます。日付時刻値では、西暦 100 年から 9999 年の日時を識別できます。日付または時刻のみを指定することもできますが、ColdFusion では日付時刻オブジェクトと呼ばれる単一のデータ型を使用して、日付、時刻、および日付時刻値が処理されます。

ColdFusion には、日付時刻値を作成および操作する関数や、値の全部または一部をさまざまな形式で返す関数が多数用意されています。

次のように定数を使用して、cfset タグで日付時刻値を直接入力できます。

```
<cfset myDate = "October 30, 2001">
```

この場合、情報は文字列として保管されます。日付時刻関数を使用した場合は、別の単純データ型である日付時刻オブジェクトとして値が保管されます。日時の指定には、できる限り CreateDate や CreateTime などの日付時刻関数を使用してください。日付時刻関数を使用すれば、無効な形式で日時を指定するおそれがなくなり、日付時刻オブジェクトを直接作成できます。

日時の形式

米国の標準の日付形式を使用して、日付、時刻、または日時を直接入力できます。年の値が 2 桁で指定されている場合、0 ～ 29 は 21 世紀の日付として処理されます。30 ～ 99 は 20 世紀の日付として処理されます。時刻値は、秒単位まで使用できます。次の表に、有効な日時の形式を示します。

指定データ	使用形式
日付	October 30, 2003 Oct 30, 2003 Oct. 30, 2003 10/30/03 2003-10-30 10-30-2003
時刻	02:34:12 2:34a 2:34am 02:34am 2am
日付と時刻	有効な日付形式と時刻形式の任意の組み合わせ。たとえば、次のような形式が有効です。 October 30, 2003 02:34:12 Oct 30, 2003 2:34a Oct. 30, 2001 2:34am 10/30/03 02:34am 2003-10-30 2am 10-30-2003 2am

ロケール固有の日時

ColdFusion には、現在のロケールに固有の形式で日時 (および数値や通貨値) を入出力できる関数が用意されています。ロケールとは、英語 (米国) やフランス語 (スイス) など、言語および地域を識別する情報のことです。米国標準形式以外の形式で日時を入出力するには、これらの関数を使用します。ロケールを指定するには SetLocale 関数を使用します。その例を次に示します。

```
<cfset oldlocale = SetLocale("French (Standard)")>
<cfoutput>#LSDateFormat(Now(), "ddd, dd mmmm, yyyy")#</cfoutput>
```

この例では次のような行が出力されます。

mar., 03 juin, 2003

各国語対応関数の詳細については、366 ページの「グローバル化アプリケーションの開発」および『CFML リファレンス』を参照してください。

ColdFusion での日時の保管方法

ColdFusion では、日時は日付時刻オブジェクトとして保管および操作されます。日付時刻オブジェクトでは、タイムラインデータが実数として保管されます。この保管方法は処理効率が高く、多くのデータベースシステムで採用されています。日付時刻オブジェクトでは、1 日は 2 つの連続した整数の差になります。日付時刻値の時刻の部分は、実数の小数部分に保管されます。値 0 は 12:00 AM 12/30/1899 を表します。

算術演算を使用して日付時刻値を直接操作することも可能ですが、コードが読みにくくなり、管理も難しくなる場合があります。代わりに、ColdFusion の日付時刻操作関数を使用してください。これらの関数の詳細については、『CFML リファレンス』を参照してください。

バイナリデータ型とバイナリエンコード

バイナリデータ (バイナリオブジェクトとも呼ばれます) は、GIF ファイルや実行可能プログラムファイルの内容などの生データです。通常、バイナリデータを直接使用することはありませんが、cffile タグを使用して変数にバイナリファイルを読み込んで、文字列バイナリエンコードに変換して電子メールでファイルを転送することができます。

文字列バイナリエンコードとは、Web 上で送信可能な文字列形式でバイナリ値を表したもののことです。ColdFusion では、次に示す 3 つのバイナリエンコード形式がサポートされています。

エンコード	形式
Base64	バイナリデータのすべてのバイトを 6 ビットずつに分割してエンコードします。これによって、バイナリデータおよび ANSI 以外の文字データを破損することなく電子メールで転送できるようになります。Base64 アルゴリズムは、IETF RFC 2045 で定義されています。詳細については、www.ietf.org/rfc/rfc2045.txt を参照してください。
Hex	0 ～ 9 および A ～ F の文字を使用して、各バイトを 2 文字の 16 進数値 (3A など) で表します。
UU	UNIX UUencode アルゴリズムを使用してデータを変換します。

ColdFusion には、文字列データ、バイナリデータ、文字列エンコードされたバイナリデータの間で変換を行うための次の関数が用意されています。

関数	説明
BinaryDecode	エンコードされたバイナリデータを含む文字列をバイナリオブジェクトに変換します。
BinaryEncode	バイナリデータをエンコード文字列に変換します。
CharsetDecode	指定の文字エンコードを使用して、文字列をバイナリデータに変換します。
CharsetEncode	指定の文字エンコードを使用して、バイナリオブジェクトを文字列に変換します。
ToBase64	文字列およびバイナリデータを、Base64 でエンコードされたデータに変換します。
ToBinary	Base64 でエンコードされたデータをバイナリデータに変換します。BinaryDecode 関数には、ToBase64 の機能が含まれています。
ToString	ほとんどの単純データ型を文字列データに変換します。数値、日付時刻オブジェクト、およびブール値を変換できます。日付時刻オブジェクトは ODBC タイムスタンプ文字列に変換されます。バイナリデータを文字列に変換するには、CharsetEncode 関数を使用することをお勧めします。

複合データ型

配列、構造体、およびクエリーは、ColdFusion のビルトイン複合データ型です。構造体およびクエリーはデータ値ではなくデータのコンテナなので、オブジェクトと呼ばれることがあります。

配列および構造体の使用方法の詳細については、75 ページの「配列および構造体の使用」を参照してください。

配列

配列は複数の値を格納する方法の **1** つで、**1** 次元または **2** 次元以上の表形式に似た構造を持ちます。配列を作成するには、関数または代入ステートメントを使用します。

- ColdFusion の ArrayNew 関数を使用すると、配列が作成され、初期次元が指定されます。たとえば次の行では、空の 2 次元配列が作成されます。

```
<cfset myarray=ArrayNew(2)>
```

- 直接代入を使用すると、配列が作成され、配列要素が挿入されます。たとえば次の行では、2 次元配列が作成され、1 行目の 2 列目の値が現在の日付に設定されます。

```
<cfset myarray[1][2]=Now()>
```

前の例に示すように、要素を参照するには、次元ごとに 1 つずつ設定される数値インデックスを使用します。

最大 3 つまでの次元を含む配列を直接作成できます。配列のサイズや次元数に制限はありません。4 次元以上の配列を作成するには、配列の配列を作成します。

作成した配列の内容は、関数や直接参照を使用して操作できます。

新しい変数に既存の配列を代入すると、新しい配列が作成され、元の配列の内容が新しい配列にコピーされます。次の例では、元の配列のコピーが作成されます。

```
<cfset newArray=myArray>
```

配列の使用方法の詳細については、75 ページの「配列および構造体の使用」を参照してください。

構造体

ColdFusion の構造体は、キーと値のペアで構成されます。キーはテキスト文字列で、値にはすべての ColdFusion データ型 (構造体を含む) を使用できます。構造体によって、互いに関連する一連の変数を 1 つの名前でグループ化することができます。構造体を作成するには、ColdFusion の StructNew 関数を使用します。たとえば次の行では、**depts** という新しい空の構造体が作成されます。

```
<cfset depts=StructNew()>
```

また、構造体のキーに値を代入する方法で新しい構造体を作成することもできます。たとえば次の行では、**MyStruct** という新しい構造体が作成されます。この構造体は **MyValue** という 1 つのキーを持ち、その値は 2 になります。

```
<cfset MyStruct.MyValue=2>
```

注意：ColdFusion 7 以前のバージョンでは、このコードを実行すると、"MyStruct.MyValue" という名前の Variables スコープ変数が作成され、値が 2 に設定されていました。

作成した構造体は、関数や直接参照を使用して、キーと値のペアの追加などの操作を行えます。

構造体に保存されている要素を参照するには、次のいずれかの方法を使用します。

- **StructureName**.**KeyName**
- **StructureName**["**KeyName**"]

その例を次に示します。

```
depts.John="Sales"
depts["John"]="Sales"
```

新しい変数に既存の構造体を代入しても、新しい構造体は作成されません。この場合、新しい変数は、元の構造体変数と同じデータ (元の構造体変数が保存されているメモリ内の場所) を参照します。つまり、どちらの変数も同じオブジェクトを参照します。

たとえば、次の行で新たに作成される myStructure2 という変数は、myStructure 変数と同じ構造体を参照します。

```
<cfset myStructure2=myStructure>
```

myStructure2 の内容を変更すると、myStructure の内容も変更されます。構造体の内容をコピーするには、構造体などの複合データ型の内容をコピーする ColdFusion の Duplicate 関数を使用します。

構造体のキー名には、構造体や配列などの複合データオブジェクトの名前を指定することができます。この方法を使用すれば、非常に複雑な構造体を作成できます。

構造体の使用方法の詳細については、75 ページの「配列および構造体の使用」を参照してください。

クエリー

クエリーオブジェクト (クエリー、クエリー結果、レコードセットとも呼ばれます) は ColdFusion の複合データ型で、名前を持つ列のセットとしてデータを表現します。これは、データベーステーブルの列のセットに似ています。データをクエリーオブジェクトとして返すタグの例には、次のものがあります。

- cfquery
- cfdirectory
- cfhttp
- cfldap
- cfpop
- cfprocresult

これらのタグでは、name 属性でクエリーオブジェクトの変数名を指定します。また、QueryNew 関数もクエリーオブジェクトを作成します。

新しい変数にクエリーを代入しても、クエリーオブジェクトはコピーされません。この場合は、どちらの名前も同じレコードセットデータを参照します。たとえば次の行では、myQuery 変数と同じレコードセットを参照する myQuery2 という変数が新たに作成されます。

```
<cfset myQuery2 = myQuery>
```

myQuery のデータを変更すると、myQuery2 にもその変更が反映されます。

クエリー列を参照するには、クエリー名、ピリオド、列名の形式で指定します。たとえば、次のようになります。

```
myQuery.Dept_ID
```

タグ属性でクエリー名を指定する cfoutput や cfloop などのタグ内では、クエリー列を参照するときにクエリー名を指定する必要はありません。

クエリー列は、1 次元配列と同じ方法でアクセスできます。たとえば次の行では、myQuery クエリーの Employee 列の 2 行目の内容を変数 myVar に代入します。

```
<cfset myVar = myQuery.Employee[2]>
```

注意：クエリー内の 1 つの行全体 (すべての列) を配列表記法で参照することはできません。たとえば、myQuery[2] は myQuery クエリーオブジェクトの 2 番目の行全体を意味しません。

構造体およびクエリーの操作

構造体変数とクエリー変数はオブジェクトへの参照です。これらのデータ型には、次のセクションに示すルールが適用されます。

オブジェクトに対する複数の参照

複数の変数が 1 つの構造体またはクエリーオブジェクトを参照している場合、オブジェクトへの参照が 1 つでもあれば、そのオブジェクトは存続します。その例を次に示します。

```
<cfscript> depts = structnew();</cfscript>
<cfset newStructure=depts>
<cfset depts.John="Sales">
<cfset depts=0>
<cfoutput>
    #newStructure.John#<br>
    #depts#
</cfoutput>
```

この例を実行すると、次の出力が表示されます。

Sales

0

<cfset depts=0> タグの実行後、depts 変数は構造体を参照しません。この変数は値 0 の単純変数です。ただし、変数 newStructure は引き続き元の構造体オブジェクトを参照します。

オブジェクトへのスコープの割り当て

他のスコープの新しい変数にクエリーまたは構造体を代入すると、そのクエリーまたは構造体に別のスコープが割り当てられます。たとえば次の行では、myquery というローカル変数を使用して、Server.SScopeQuery というサーバー変数を作成します。

```
<cfset Server.SScopeQuery = myquery>
```

サーバースコープのクエリー変数をクリアするには、次のように再度クエリーオブジェクトを代入します。

```
<cfset Server.SScopeQuery = 0>
```

この行を実行すると、元のオブジェクトへの参照はサーバースコープから削除されますが、その他の参照は削除されません。

オブジェクトのコピーと複製

Duplicate 関数を使用すると、構造体またはクエリーオブジェクトのコピーを作成できます。コピーを変更しても、元の構造体またはクエリーオブジェクトには影響しません。

クエリー列の使用

query 属性を持つ cfloop、cfoutput、cfmail などのタグの外部では、クエリー列を配列として処理できます。ただし、クエリー列の参照は、常に意図どおりに機能するとは限りません。このセクションでは、次の cfquery タグで生成される結果を例に取って、クエリー列の参照について説明します。

```
<cfquery dataSource="cfdocexamples" name="myQuery">
    SELECT FirstName, LastName
    FROM Employee
</cfquery>
```

クエリー列内の要素を参照するには、配列のインデックスに行番号を指定します。たとえば、次のどちらの行でも "ben" という単語が表示されます。

```
<cfoutput> #myQuery.Firstname[1]# </cfoutput><br>
<cfoutput> #myQuery["Firstname"][1]# </cfoutput><br>
```

配列インデックスを使用せずに、myQuery.Firstname や myQuery["Firstname"] のようにクエリー列を参照した場合は、やや複雑になります。この 2 つの参照形式では、生成される結果が異なります。

myQuery.Firstname という参照は、その列の最初の行に自動変換されます。たとえば次の行では、"ben" という単語が出力されます。

```
<cfset myCol = myQuery.Firstname >
<cfoutput>#mycol#</cfoutput>
```

しかし、次の行ではエラーメッセージが表示されます。

```
<cfset myCol = myQuery.Firstname >
<cfoutput>#mycol[1]#</cfoutput><br>
```

Query["Firstname"] を参照した場合、列の最初の行への自動変換は行われません。たとえば次の行では、複合型を単純値に変換できないことを示すエラーメッセージが表示されます。

```
<cfoutput> #myQuery['Firstname']# </cfoutput><br>
```

同様に次の行では、列の 2 行目の値である名前 "marjorie" が出力されます。

```
<cfset myCol = myQuery["Firstname"]>
<cfoutput>#mycol[2]#</cfoutput><br>
```

ただし、単純値が要求される代入式では、クエリー列が最初の行の値に自動変換されます。たとえば次の行では、名前 "ben" が 2 回表示されます。

```
<cfoutput> #myQuery.Firstname# </cfoutput><br>
<cfset myVar= myQuery['Firstname']>
<cfoutput> #myVar# </cfoutput><br>
```

ColdFusion 10 では FUNCTION は ColdFusion データ型

次に例を示します。

```
<cfscript>
public FUNCTIOn FUNCTION a()
{
return variables.b;
}
public FUNCTION b()
{
return "Hal";
}
writeoutput(a());
writedump(a().getMetadata());
writedump(a.getMetadata());
writedump(x.getMetadata());
</cfscript>
```

変数の参照でのピリオドの使用

ColdFusion では、構造体、クエリー、XML ドキュメントオブジェクト、外部オブジェクトなどの複合変数の各要素は、ピリオド (.) を使用して区切ります。たとえば、MyStruct.KeyName のようにします。また、変数名と変数スコープ識別子を区切るときにもピリオドを使用します。たとえば、Variables.myVariable や CGI.HTTP_COOKIE のようにします。

Cookie および Client スコープの変数 (これらは常に単純変数型になります) を除き、通常は、単純変数の名前にピリオドを含めることはできません。ただし、この要件を満たしていないレガシーコードやサードパーティコードに対応できるように、いくつかの例外が設けられています。

詳細については、51 ページの「スコープについて」、75 ページの「配列および構造体の使用」、および 1028 ページの「XML および WDDX の使用」を参照してください。

変数とピリオドについて

次のセクションでは、MyVar.a.b という変数を例に取って、ColdFusion で変数値を取得および設定するときのピリオドの使用方法について説明します。

変数の取得

ColdFusion では、変数の名前にピリオドが含まれていても、変数の値を正しく取得できます。たとえば、<cfset Var2 = myVar.a.b> や IsDefined(myVar.a.b) という式では、次の手順に従って MyVar.a.b が取得されます。

1. 内部で保持されている名前テーブル (シンボルテーブル) で、myVar が検索されます。
2. myVar が複合オブジェクト (スコープを含む) の名前である場合は、オブジェクト内の a という要素が検索されます。

   myVar が複合オブジェクトの名前でない場合は、myVar.a が複合オブジェクトの名前であるかどうかが確認され、手順 3 が省略されます。
3. myVar が複合オブジェクトの名前である場合は、a が複合オブジェクトであるかどうかが確認されます。
4. myVar.a が複合オブジェクトの名前である場合は、b が単純変数の名前であるかどうかが確認され、b の値が返されます。

   myVar が複合オブジェクトであり、a が複合オブジェクトでない場合は、a.b が単純変数の名前であるかどうかが確認され、その値が返されます。

   myVar.a が複合オブジェクトでない場合は、myVar.a.b が単純変数の名前であるかどうかが確認され、その値が返されます。

このようにして変数名が正しく解決され、その値が取得されます。

名前にピリオドが含まれている単純変数は、配列表記を使用して取得することもできます。この配列表記では、スコープ名 (または単純変数が含まれている複合変数) を配列名として使用します。角括弧の中では、単純変数の名前を単一引用符 (') または二重引用符 (") で囲んで記述します。

配列表記では、キーの可能な組み合わせを ColdFusion が解析して確認する必要がないので、通常のドット表記よりも効率的です。たとえば、次の行はどちらも myVar.a.b の値を出力しますが、2 番目のほうが効率的です。

```
<cfoutput>myVar.a.b is: #myVar.a.b#<br></cfoutput>
<cfoutput>myVar.a.b is: #Variables["myVar.a.b"]#<br></cfoutput>
```

変数の設定

変数値を設定する場合は、変数を取得するときのような柔軟性はありません。これは、作成または設定する変数の型を決定する必要があるからです。したがって、設定する変数名のルールはより厳格になります。このルールは、変数名の最初の部分が Cookie または Client スコープ識別子であるかどうかによって異なります。

たとえば、次のコードがあるとします。

```
<cfset myVar.a.b = "This is a test">
```

変数 myVar が存在していない場合は、次の処理が行われます。

1. myVar という構造体が作成されます。
2. 構造体 myVar の中に a という構造体が作成されます。
3. myVar.a の中に b というキーが作成されます。
4. このキーに " これはテストです " という値が設定されます。

myVar または myVar.a が存在し、そのいずれも構造体でない場合は、エラーになります。

つまり、変数を取得するときと同じルールを使用して、まだ存在しない名前が検出されるまで変数名が解析されます。次に、b というキーを作成するために必要な構造体が作成され、そのキーに値が割り当てられます。

ただし、最初のピリオドの前にある名前が Cookie、Client、ColdFusion のいずれかである場合は、別のルールが使用されます。この場合は、スコープ名の後に続くすべてのテキスト (ピリオドも含む) が、単純変数の名前と見なされます。これは、Cookie および Client スコープの変数は常に単純変数だからです。次のコードを実行すると、myVar.a.b という名前の Client スコープの単純変数が 1 つ作成されます。

```
<cfset Client.myVar.a.b = "This is a test">
<cfdump var=#Client.myVar.a.b#>
```

ピリオドを持つ変数の作成

構造体のドット表記を除き、ピリオドを含んだ変数名は作成しないでください。ただし、外部データソースの変数名との互換性の維持や、変数名にピリオドを使用している既存のコードの統合など、ピリオドの使用を避けられない場合に、これを処理するためのメカニズムが用意されています。次のセクションでは、ピリオドを含んだ単純変数名を作成する方法について説明します。

括弧を使用してピリオドを持つ変数を作成する

連想配列構造体 (85 ページの「構造体の表記法」を参照) の表記法を使用して、ピリオドを含んだ変数名を作成できます。その手順は次のとおりです。

- 構造体のキーとして変数を参照します。ColdFusion ではすべてのスコープが構造体と見なされているので、これは常に実行できます。スコープの詳細については、51 ページの「スコープについて」を参照してください。
- ピリオドを含んだ変数名を、角括弧の中に、単一引用符 (') または二重引用符 (") で囲んで記述します。

その例を次に示します。

```
<cfset Variables['My.Variable.With.Periods'] = 12>
<cfset Request["Another.Variable.With.Periods"] = "Test variable">
<cfoutput>
    My.Variable.With.Periods is: #My.Variable.With.Periods#<br>
    Request.Another.Variable.With.Periods is:
        #Request.Another.Variable.With.Periods#<br>
</cfoutput>
```

ピリオドを持つ Client および Cookie 変数の作成

ピリオドを含んだ Client または Cookie 変数を作成するには、単純にその変数に値を代入します。たとえば次の行では、User.Preferences.CreditCard という Cookie が作成されます。

```
<cfset Cookie.User.Preferences.CreditCard="Discover">
```

データ型の変換

ColdFusion では、式の演算要件 (関数の引数の要件など) に合わせて、データ型が自動的に変換されます。したがって通常は、データ型の互換性や変換に注意を払う必要はありませんが、ColdFusion でどのようにデータ値が評価され、データ型が変換されるかを理解しておくと、エラーの防止やコードの効率化に役立ちます。

オペレーション志向の評価方法

従来のプログラミング言語では、さまざまな型のオブジェクトを式で組み合わせることは厳格に制限されています。たとえば、C++ や Basic などの言語の場合、式 ("8" * 10) はエラーになります。乗算演算子のオペランドとして 2 つの数値が必要であるにもかかわらず、"8" という文字列が指定されているからです。このような言語でプログラムを作成するときは、データ型を変換してエラーを防ぐ必要があります。たとえば、前の式は (ToNumber("8") * 10) と記述する必要があります。

しかし、ColdFusion では、式 ("8" * 10) は数値 80 と評価され、エラーは発生しません。ColdFusion では、乗算演算子を処理するときに、オペランドが自動的に数値に変換されます。"8" は数値 8 に正常に変換できるので、式は 80 と評価されます。

ColdFusion では、式および関数は次の順序で処理されます。

1 式中の各演算子について、必要なオペランドが決定されます。たとえば、乗算演算子には数値のオペランドが必要であり、CONTAINS 演算子には文字列のオペランドが必要です。

  関数の場合は、関数の各引数に必要な型が決定されます。たとえば、Min 関数には引数として 2 つの数値が必要であり、Len 関数には 1 つの文字列が必要です。

2 すべてのオペランドまたは関数の引数が評価されます。

3 オペランドや引数の型が必要な型と異なる場合は、それらがすべて変換されます。変換に失敗した場合はエラーが生成されます。

型の変換

ColdFusion の式評価メカニズムは強力ですが、すべてのデータを自動的に変換できるとは限りません。たとえば、式 "eight" * 10 では、文字列 "eight" を数値 8 に変換できないのでエラーが発生します。したがって、データ型の変換ルールを理解する必要があります。

次の表に、実行される変換を示します。最初の列は変換元の値を示します。その他の列は、各データ型に変換された結果を示します。

値	ブール値	数値	日付時刻値	文字列
"Yes"	True	1	エラー	"Yes"
"No"	False	0	エラー	"No"
True	True	1	エラー	"Yes"
False	False	0	エラー	"No"
数値	数値が 0 でない場合は True、0 の場合は False。	数値	この章の「日付時刻値」を参照してください。	数値の文字列表現 ("8" など)
文字列	"Yes" の場合は True。 "No" の場合は False。 0 に変換できる場合は、False。 0 以外の数値に変換できる場合は、True。	"1,000" や "12.36E-12" など、数値を表す場合は、対応する数値に変換されます。日付時刻値を表す場合 (次の列を参照) は、対応する日付時刻オブジェクトの数値に変換されます。	"{ts '2001-06-14 11:30:13'}" などの ODBC の日付、時刻、またはタイムスタンプである場合や、米国の標準日時形式 (完全なまたは省略された月名が使用されている場合も含む) である場合は、対応する日付時刻値に変換されます。 曜日や特殊な句読点が含まれている場合はエラーになります。 通常はダッシュ、スラッシュ、およびスペースが有効です。	文字列
日付	エラー	日付時刻オブジェクトの数値	日付	ODBC タイムスタンプ

ColdFusion では、配列、クエリー、COM オブジェクトなどの複合型は他の型に変換できません。ただし、複合型の単純データ要素を他の単純データ型に変換することはできます。

型変換に関する注意事項

次のセクションでは、型変換に関する詳細なルールおよび注意事項について説明します。

cfoutput タグ

cfoutput タグは、常にデータを文字列として表示します。したがって、cfoutput タグで変数が表示される前に、文字列以外のすべてのデータに対して型変換ルールが適用されます。たとえば、cfoutput タグは日付時刻値を ODBC タイムスタンプとして表示します。

ブール値への変換では大文字と小文字が区別されない

ColdFusion の式評価では大文字と小文字は区別されないので、Yes、YES、yes は同じと見なされます。また、False、FALSE、false や、No、NO、no や、True、TRUE、true も同じと見なされます。

バイナリデータの変換

ColdFusion では、バイナリデータを自動的に他のデータ型に変換することはできません。バイナリデータを変換するには、ToBase64 および ToString 関数を使用します。詳細については、40 ページの「バイナリデータ型とバイナリエンコード」を参照してください。

日付時刻データの変換

日付時刻値が確実に実数として表現されるようにするには、変数に 0 を加算します。その例を次に示します。

```
<cfset mynow = now()>
Use cfoutput to display the result of the now function:<br>
<cfoutput>#mynow#</cfoutput><br>
Now add 0 to the result and display it again:<br>
<cfset mynow = mynow + 0>
<cfoutput>#mynow#</cfoutput>
```

2003 年 6 月 6 日、午後 1 時 6 分である場合は、次のように出力されます。

```
Use cfoutput to display the result of the now function:
{ts '2003-06-03 13:06:44'}
Now add 0 to the result and display it again:
37775.5463426
```

数値の変換

整数と実数の両方が含まれている式を評価すると、結果は実数になります。実数を整数に変換するには、ColdFusion 関数を使用します。Int、Round、Fix、および Ceiling 関数を使用すると実数が整数に変換されますが、数値の小数部の処理はそれぞれ異なります。

名前に _integer または _range という接尾辞が付いている非表示フォームフィールドを使用してフォーム入力フィールドを検証する場合は、フィールドに入力された実数が整数に丸められ、その整数がアクションページに渡されます。

名前に _integer、_float、または _range という接尾辞が付いている非表示フォームフィールドを使用してフォーム入力フィールドを検証する場合は、入力されたデータにドル記号を含むドル金額やカンマ付き数値が含まれていても入力は有効と見なされ、値からドル記号やカンマが削除されて、その整数または実数がアクションページに渡されます。

ColdFusion には、任意精度の 10 進数 (BigDecimal 数) を表す固有のデータ型は用意されていません。ColdFusion では、そのような数値はまず文字列として保存されます。その値を式で使用すると数値型に変換され、多くの場合精度が落ちます。精度を維持する PrecisionEvaluate メソッドを使用すれば、BigDecimal 精度演算を使用して文字列式が評価され、長い数値の文字列として結果が返されます。詳細については、『CFML リファレンス』の PrecisionEvaluate を参照してください。

評価および型変換の問題

次のセクションでは、型の評価および変換時に発生する可能性がある問題について説明します。

True または False と変数との比較

次に示す 2 つの cfif タグの例は、どちらも同じ結果が生成されるように見えるかもしれません。

```
<cfif myVariable>
    <cfoutput>myVariable equals #myVariable# and is True
    </cfoutput>
</cfif>
<cfif myVariable IS True>
    <cfoutput>myVariable equals #myVariable# and is True
    </cfoutput>
</cfif>
```

しかし実際は、常に同じになるとは限りません。myVariable に 12 などの数値が代入されている場合は、最初の例の結果のみが生成されます。2 番目の例の場合、IS 演算子は特定のデータ型を必要とせず、2 つの値の一致をテストするだけなので、myVariable の値がブール値に変換されることはありません。したがって、値 12 と定数 True が比較されることになります。この 2 つは等しくないので、何も出力されません。ただし、myVariable が 1、"Yes"、または True の場合は、ブール値 True とそれらが同じであると見なされるので、どちらの例でも同じ結果が出力されます。

次に示すコードでは、変数の値 12 とブール値 False は等しくないので、出力ステートメントが実行されます。

```
<cfif myVariable IS NOT False>
    <cfoutput>myVariable equals #myVariable# and IS NOT False
    </cfoutput>
</cfif>
```

したがって、変数が True か False かをテストする場合は、IS 比較演算子を使用せず、<cfif **testvariable**> でテストするようにしてください。この問題は、型があいまいな式の評価という問題の一例です。次のセクションで、この一般的な問題について説明します。

型があいまいな式および文字列

文字列を必要としない式 (IS や GT などのすべての比較演算子も含む) が ColdFusion で評価されるときには、各文字列値を数値または日付時刻オブジェクトに変換できるかどうかが確認されます。変換できる場合は、対応する数値または日付時刻値 (これは数値として保管されます) に変換されます。その後で、その数値が式で使用されます。

1a や 2P のような短い文字列では、予期しない結果が生成されることがあります。ColdFusion は、1 文字の "a" を AM、1 文字の "P" を PM と解釈します。したがって、文字列が意図せず日付時刻値として解釈されることがあります。

同様に、文字列を数値として解釈できる場合は、予期しない結果が生成されることがあります。

たとえば、ColdFusion では次のように式が解釈されます。

式	解釈
<cfif "1a" EQ "01:00">	1:00am が 1:00am である場合
<cfif "1P" GT "2A">	1:00pm が 2:00am より遅い場合
<cfset age="4a"> <cfset age=age + 7>	変数 age が 4:00 am と見なされて日付時刻値 0.16666666667 に変換され、7 が加算されて 7.16666666667 になります。
<cfif "0.0" is "0">	0 が 0 である場合

文字列比較のこのようなあいまいさを防ぐには、比較演算子ではなく、ColdFusion の文字列比較関数 Compare および CompareNoCase を使用します。

また、日付時刻値として解釈できる文字列であるかどうかを IsDate 関数で調べたり、比較する前に文字列に文字を追加して間違って解釈されないようにすることもできます。

ODBC がサポートされていない場合の日付時刻関数およびクエリー

CFML には、Now、CreateDate、CreateTime、CreateDateTime など、日付時刻オブジェクトを返す関数が多く用意されています。ColdFusion では、日付時刻オブジェクトが文字列に変換されるときには、ODBC (Open Database Connectivity) タイムスタンプ値が作成されます。したがって、ODBC のエスケープシーケンスをサポートしていないデータベースドライバで日付を使用したり、クエリーオブクエリーで SQL を使用したりすると、予期しない結果が生成されることがあります。

データベースドライバで ODBC 形式の日付がサポートされていない場合は、SQL を使用してデータベースにデータを挿入するときや、WHERE 節でデータベースからデータを選択するときに DateFormat 関数を使用して、そのドライバで有効な形式に日付時刻値を変換してください。これはクエリーオブクエリーにも当てはまります。

たとえば次の SQL ステートメントでは、クエリーオブクエリーで DateFormat 関数を使用して、将来の MyDate 値を持つ行を選択しています。

```
<cfquery name="MyQofQQ" dbtype="query">
SELECT *
FROM DateQuery
WHERE MyDate >= '#DateFormat(Now())#'
</cfquery>
```

次のクエリーオブクエリーは失敗し、" エラー : 有効な日付ではありません " のようなエラーメッセージが表示されます。これは、ColdFusion の Now 関数が ODBC タイムスタンプを返すからです。

```
<cfquery name="MyQofQQ" dbtype="query">
SELECT *
FROM DateQuery
WHERE MyDate >= '#now()#'
</cfquery>
```

オーバーロードされた Java メソッドでの JavaCast の使用

Java では、メソッドをオーバーロードすることで、パラメータのデータ型のみが異なる同じ名前のメソッドを 1 つのクラス内に用意することができます。実行時には、呼び出しで渡されたパラメータの型に基づいて、Java 仮想マシンが適切なメソッドを選択します。ColdFusion では型が明示されていないので、仮想マシンでどのメソッドが選択されるかは予測できません。

ColdFusion の JavaCast 関数を使用して変数の Java 型を指定すれば、正しいメソッドが選択されるようになります。その例を次に示します。

```
<cfset emp.SetJobGrade(JavaCast("int", JobGrade))>
```

JavaCast 関数は、Java データ型を示す文字列と、その型を設定する変数の 2 つのパラメータを取ります。指定できる Java データ型は、Boolean、int、long、float、double、および String です。

JavaCast 関数の詳細については、『CFML リファレンス』を参照してください。

引用符の使用

ColdFusion で文字列データが正しく解釈されるようにするには、文字列を単一引用符 (') または二重引用符 (") で囲みます。たとえば "10/2/2001" は、日付時刻オブジェクトに変換可能な文字列と評価されます。しかし、10/2/2001 は数式 5/2001 と解釈されるので、0.00249875062469 と評価されます。

式評価での型変換の例

ColdFusion の式評価の例を次に示します。

例 1

```
2 * True + "YES" - ('y' & "es")
```

文字列としての結果値 : "2"

説明 : (2*True) は 2、("YES"- "yes") は 0 なので、2 + 0 は 2 になります。

例 2

```
"Five is " & 5
```

文字列としての結果値 : "Five is 5"

説明 : 5 は文字列 "5" に変換されます。

例 3

```
DateFormat("October 30, 2001" + 1)
```

文字列としての結果値 : "31-Oct-01"

説明 : 加算演算子によって、文字列 "October 30, 2001" が日付時刻オブジェクトに変換され、さらに数値に変換されます。その数値に 1 が加算されます。DateFormat 関数の引数は日付時刻オブジェクトである必要があるので、加算された結果が日付時刻オブジェクトに変換されます。日付時刻オブジェクトに 1 が加算されたことで、変換結果は 1 日後の October 31, 2001 になります。

スコープについて

各変数には、どのように設定されるか (コードで設定されるのか ColdFusion で設定されるのか)、コード内のどの部分で使用できるか、どの期間保持されるかという条件が定められており、この条件に基づいて変数を分類できます。これらの条件は、一般に変数のスコープと呼ばれています。よく使用されるスコープとしては、作成する変数のデフォルトスコープである Variables スコープや、HTTP リクエストの存続期間でのみ使用できる Request スコープなどがあります。

注意 : ユーザー定義関数もスコープに属しています。詳細については、168 ページの「関数のスコープの指定」を参照してください。

スコープタイプ

ColdFusion スコープの説明を次の表に示します。

スコープ	説明
Application	サーバー上に存在する単一の名前付きアプリケーションに関連付けられている変数が含まれます。アプリケーションの名前は、cfapplication タグの name 属性か、"Application.cfc" の This.name 変数設定で指定されます。詳細については、300 ページの「永続データとロックの使用」を参照してください。
Arguments	ユーザー定義関数や ColdFusion コンポーネントメソッドの呼び出しで渡される変数。詳細については、157 ページの「Arguments スコープについて」を参照してください。
Attributes	カスタムタグページおよびスレッドでのみ使用できます。呼び出したページまたは cfthread タグによってタグの属性に渡された値が含まれます。詳細については、206 ページの「カスタム CFML タグの作成と使用」および 326 ページの「ColdFusion スレッドの使用」を参照してください。
Caller	カスタムタグページでのみ使用できます。カスタムタグの Caller スコープは、呼び出したページの Variables スコープへの参照です。カスタムタグページで Caller スコープを使用して作成または変更したすべての変数は、呼び出したページの Variables スコープでアクセスできます。詳細については、206 ページの「カスタム CFML タグの作成と使用」を参照してください。
CGI	ページがリクエストされたコンテキストを識別する環境変数が含まれます。使用可能な変数は、ブラウザやサーバーソフトウェアによって異なります。よく使用される CGI 変数のリストについては、『CFML リファレンス』の Reserved Words and Variables を参照してください。
Client	特定のクライアントに関連付けられている変数が含まれます。Client 変数を使用すれば、ユーザーがアプリケーションで別のページに移動してもそのステートを維持できます。Client 変数は、複数のブラウザセッションにまたがって使用できます。デフォルトでは、Client 変数はシステムレジストリに保管されますが、Cookie またはデータベースに保管することもできます。Client 変数は複合データ型にすることはできませんが、名前にピリオドを含めることは可能です。詳細については、300 ページの「永続データとロックの使用」を参照してください。
Cookie	Cookie としてユーザーのブラウザに保持されている変数が含まれます。通常、Cookie はブラウザでファイルに保管されるので、複数のブラウザセッションおよびアプリケーションにまたがって使用できます。メモリ内のみの Cookie 変数も作成できますが、これはブラウザを閉じると使用できなくなります。Cookie スコープの変数名にはピリオドを含めることができます。
Flash	SWF ムービーと ColdFusion の間で送受信される変数。詳細については、595 ページの「Flash Remoting サービスの使用」を参照してください。
Form	フォームの送信によってフォームページからアクションページに渡された変数が含まれます。HTML の form タグを使用する場合は、method="post" を使用する必要があります。詳細については、692 ページの「データの取得および形式設定の概要」を参照してください。
Local (関数ローカル)	ユーザー定義関数または ColdFusion コンポーネントメソッドの中で宣言され、該当する関数が実行されている間のみ存在する変数が含まれます。詳細については、150 ページの「ユーザー定義関数の作成と呼び出し」を参照してください。

スコープ	説明
Request	単一の HTTP リクエストの存続期間に利用可能なデータを保持します。Request スコープは、カスタムタグおよびネストカスタムタグを含む、リクエストに応じて処理されるすべてのページで使用可能です。 このスコープは、ネストされた (子 / 親) タグで役立ちます。変数をロックしなくても済むように、Application スコープの代わりに使用することもできます。Request スコープの使用方法については複数の章で説明しています。
Server	現在の ColdFusion サーバーに関連付けられている変数が含まれます。このスコープを使用すれば、複数のアプリケーションにまたがってすべての ColdFusion ページで使用可能な変数を定義できます。詳細については、300 ページの「永続データとロックの使用」を参照してください。
Session	特定のクライアントに関連付けられ、クライアントの特定のセッションでのみ維持される変数が含まれます。この変数はサーバーのメモリに保管され、アクティブでない状態が一定時間続いたらタイムアウトするように設定できます。詳細については、300 ページの「永続データとロックの使用」を参照してください。
This	ColdFusion コンポーネント内、または ColdFusion 構造体などの上位オブジェクトに属する cffunction タグ内でのみ存在します。コンポーネントインスタンスまたは上位オブジェクトが存続している間のみ存在します。This スコープのデータにコンポーネントまたはコンテナの外からアクセスするには、インスタンス名またはオブジェクト名を接頭辞として使用します。
ThisTag	カスタムタグページでのみ使用できます。ThisTag スコープはタグの現在の呼び出しに対してアクティブになります。カスタムタグの中にタグがネストされている場合、ネストされたタグを呼び出す前に設定した ThisTag スコープ値は、ネストされたタグから呼び出したタグに戻ったときにすべて保持されます。 ThisTag スコープには、タグの実行モードを識別する変数、タグによって生成された内容を示す変数、終了タグがあるかどうかを示す変数の 3 つのビルトイン変数が用意されています。 ネストされたカスタムタグでは、cfassociate タグを使用することで、呼び出したタグの ThisTag スコープに値を返せます。詳細については、213 ページの「タグインスタンスデータへのアクセス」を参照してください。
Thread	特定の ColdFusion スレッドで作成および変更される変数が含まれます。ただし、この変数の読み取りは、そのスレッドを作成したページ内のすべてのコードで行えます。各スレッドの Thread スコープは、cfthread スコープのサブスコープです。詳細については、326 ページの「ColdFusion スレッドの使用」を参照してください。
スレッドローカル	特定の ColdFusion スレッドでのみ使用可能な変数。詳細については、326 ページの「ColdFusion スレッドの使用」を参照してください。
URL	現在のページを呼び出すために使用された URL の中で指定されているパラメータが含まれます。パラメータは、www.MyCompany.com/inputpage.cfm?productCode=A12CD1510&quantity=3 のように、?variablename = value[&variablename=value...] という形式で URL に追加されます。 同じ名前のパラメータが URL に複数含まれている場合、ColdFusion URL スコープの変数には、それらすべてのパラメータをカンマで区切った値が格納されます。たとえば、http://localhost/urlparamtest.cfm?param=1&param=2&param=3 という URL の場合、ColdFusion ページの URL.param 変数値は 1,2,3 になります。
Variables	cfset タグや cfparam タグで作成された任意の型の変数のデフォルトスコープ。Variables スコープの変数は、その変数を作成したページ、またはそのページに含まれているページでのみ使用できます。「Caller スコープ」も参照してください。 CFC で作成された Variables スコープの変数は、コンポーネントとその関数でのみ使用できます。コンポーネントのインスタンス化やその関数の呼び出しを行うページでは使用できません。

重要：データの破損を防ぐために、Session、Application、または Server スコープ変数を使用するコードをロックしてください。詳細については、300 ページの「永続データとロックの使用」を参照してください。

スコープでの変数の作成および使用

次の表に、各種のスコープで変数を作成および参照する方法を示します。スコープで変数を作成するメカニズムの詳細については、34 ページの「変数の作成」を参照してください。

スコープ接頭辞 (タイプ)	参照に接頭辞が必要	使用可能な場所	作成方法
Application	必要	1 つのアプリケーション内の複数のクライアントで、複数のブラウザセッションにまたがって使用できます。アプリケーション変数を使用するコードは cflock ブロックで囲んでください。	変数の作成時に接頭辞 Application を指定します。
Arguments	不要	ユーザー定義関数の本文内、または ColdFusion コンポーネントメソッド内で使用できます。	関数呼び出しで引数を渡す呼び出しページによって作成されます。
Attributes	必要	カスタムタグページまたはスレッド内で使用できます。	カスタムタグの場合、カスタムタグの属性でカスタムタグページに値を渡す呼び出しページによって作成されます。 スレッドの場合、属性の値を指定する cfthread タグによって作成されます。
Caller	カスタムタグページでは必要。 呼び出しページでは不要。Variables 接頭辞はオプションです。	カスタムタグページでは、Caller スコープ接頭辞を付けます。 カスタムタグを呼び出すページではローカル変数 (Variables スコープ) として使用できます。	カスタムタグページでは、変数の作成時に接頭辞 Caller を指定します。 呼び出しページでは、変数の作成時に接頭辞 Variables を指定するかどうかはオプションです。
Cffile	必要	cffile の呼び出しの後で使用できます。	cffile タグによって作成されます。
CGI	不要	すべてのページで使用できます。値は最新のブラウザリクエストによって変化します。	Web サーバーによって作成されます。ブラウザリクエストから生成されるサーバー環境変数が含まれます。
Client	不要	1 つのアプリケーションの 1 つのクライアントで、複数のブラウザセッションにまたがって使用できます。	変数の作成時に接頭辞 Client を指定します。
Cookie	不要	アプリケーションおよびページの 1 つのクライアントで、複数のブラウザセッションにまたがって使用できます。	cfcookie タグによって作成されます。変数の作成時に接頭辞 Cookie を指定して、メモリ内のみの Cookie を設定することもできます。
Flash	必要	Flash クライアントから呼び出される ColdFusion ページまたは ColdFusion コンポーネントで使用できます。	ColdFusion クライアントアクセスによって作成されます。Flash に値を割り当てます。Flash.result および Flash.pagesize 変数に値を割り当てることができます。
Form	不要	フォームのアクションページ、およびそのアクションページで呼び出されるカスタムタグで使用できます。アクションページではないフォームページでは使用できません。	form タグまたは cfform タグによって作成されます。フォーム送信時に使用される、フォーム本文内のフォームフィールドタグの値 (input など) が含まれます。変数名はフォームフィールドの名前です。
Local	不要	ユーザー定義関数または ColdFusion コンポーネントメソッドの本文内で、関数が実行されている間のみ使用できます。	次のいずれかの方法で作成されます。 • 関数またはメソッド定義内で、cfset タグの var キーワード、または CFScript の var ステートメントを使用して作成します。 • 関数またはメソッドの変数を作成する際に、ローカルキーワードを指定して作成します。
Request	必要	作成したページ、および変数の作成後に現在の HTTP リクエストの間に実行されている任意のページ (カスタムタグやネストされたカスタムタグを含む) で使用できます。	変数の作成時に接頭辞 Request を指定します。
Server	必要	ColdFusion サーバーの任意のページで使用できます。サーバー変数を使用するコードはすべて cflock ブロックで囲んでください。	変数の作成時に接頭辞 Server を指定します。

スコープ接頭辞 (タイプ)	参照に接頭辞が必要	使用可能な場所	作成方法
Session	必要	1 つのアプリケーションの 1 つのクライアントで、1 つのブラウザセッションでのみ使用できます。Session スコープ変数を使用するコードは cflock ブロックで囲んでください。	変数の作成時に接頭辞 Session を指定します。
This	必要	ColdFusion コンポーネント内、または cffunction タグを使用して作成され、オブジェクト、構造体、またはスコープ内に配置されたユーザー定義関数の本文内で使用できます。その変数を含むページ内では、コンポーネントインスタンスまたはその変数を含むオブジェクトを介して使用できます。	コンポーネント内または関数内では、変数の作成時に接頭辞 This を指定します。 その変数を含むページ内では、変数の作成時にコンポーネントインスタンスまたはその関数を含むオブジェクトを接頭辞として指定します。
ThisTag	必要	カスタムタグページで使用できます。	タグで変数を作成するときに接頭辞 ThisTag を指定するか、またはネストされたカスタムタグで cfassociate タグを使用して作成します。
Thread	スレッド名。 変数を作成するスレッド内では、キーワード thread も使用できます。	リクエスト内の任意のコードで使用できます。	変数の作成時に、キーワード thread またはスレッド名を接頭辞として使用します。 Thread 変数はスレッド内でのみ作成できます。
スレッドローカル (接頭辞なし)	なし	cfthread タグによって作成されたスレッド内で使用できます。	変数の作成時に接頭辞を使用しないで作成します。変数名の前にキーワード var を使用することもできます。
URL	不要	URL のターゲットページで使用できます。	システムによって作成されます。ページにアクセスするときに使用した URL クエリー文字列で渡されたパラメータが含まれます。
Variables (ローカル)	不要	現在のページで使用できます。フォームページがアクションページでないかぎり、フォームのアクションページからはアクセスできません。カスタムタグを呼び出しているページで使用されているこのスコープの変数は、Caller スコープを使用することでカスタムタグ内からアクセスできます。ただし、ネストされたカスタムタグからはアクセスできません。	変数の作成時に接頭辞 Variables を指定するかどうかはオプションです。ColdFusion スレッド内で Variables スコープ変数を作成する場合は、Variables 接頭辞を使用する必要があります。

スコープの使用

次のセクションでは、さまざまなスコープで変数を作成および使用する方法について詳しく説明します。

スコープが指定されていない変数の評価

スコープ接頭辞のない変数名が使用されている場合は、次の順で各スコープが検索されて、所属するスコープが確認されます。

1. ローカル (関数ローカル、UDF および CFC のみ)
2. Arguments
3. スレッドローカル (スレッド内の場合のみ)
4. クエリー (実際にはスコープではありません。クエリーループ内の変数)
5. Thread
6. Variables
7. CGI
8. Cffile

9 URL

10 Form

11 Cookie

12 Client

スコープを指定しないと ColdFusion が変数を検索する必要があるので、すべての変数にスコープを指定するとパフォーマンスが向上します。

これら以外のスコープの変数にアクセスするには、変数名にスコープ識別子の接頭辞を付ける必要があります。

スコープと CFX タグ

ColdFusion のスコープは、C++ や Java などのプログラミング言語で記述されたカスタムタグである CFX (ColdFusion Extension) タグには適用されません。CFX タグを呼び出す ColdFusion ページでは、タグ属性を使用してデータを CFX タグに渡す必要があります。CFX タグでは、Java のリクエストおよびレスポンスインターフェイスや C++ のリクエストクラスを使用して、データのやり取りを行う必要があります。

Java の setVariable レスポンスインターフェイスメソッドや、C++ の CCFX::SetVariable メソッドは、呼び出したページの Variables スコープにデータを返します。これは、ColdFusion カスタムタグで Caller スコープ変数を設定するのと同じです。

構造体としてのスコープの使用

ColdFusion では、すべての名前付きスコープを構造体として使用できます。CFScript を使用して定義した UDF (ユーザー定義関数) の関数ローカルスコープは、構造体として使用できません。

名前付きスコープの変数は、構造体の要素として参照できます。この方法を使用するには、構造体名としてスコープ名を指定し、キー名として変数名を指定します。たとえば、Request スコープに MyVar という変数がある場合は、次のいずれの方法でも参照できます。

```
Request.MyVar
Request["MyVar"]
```

同様に、CFML の構造体関数を使用してスコープの内容を操作できます。構造体の使用方法の詳細については、75 ページの「配列および構造体の使用」を参照してください。

重要：セッション変数をクリアするときは StructClear(Session) を呼び出さないでください。これを呼び出すと、SessionID、CFID、および CFtoken ビルイトイン変数も削除されるので、セッションが終了してしまいます。StructClear を使用してアプリケーション変数を削除したい場合は、Session スコープの構造体に変数を格納してから、その構造体をクリアします。たとえば、すべてのアプリケーション変数を Session.MyVars に格納し、StructClear(Session.MyVars) を呼び出してそれらの変数をクリアします。

変数の存在の確認

ColdFusion では、存在しない変数値を使用しようとするとエラーが発生します。したがって、値がダイナミックに割り当てられる変数は、変数値の存在を確認してから使用する必要があります。たとえば、アプリケーションでフォームを使用している場合は、フィールドデータの入力を必須にしたり、フィールドにデフォルト値を設定したり、フィールド変数値の存在を確認するなどの対策を施す必要があります。

変数を使用する前に存在を確認する方法の例を次に示します。

- IsDefined 関数を使用すれば、変数が存在するかどうかをテストできます。
- cfparam タグを使用すれば、変数をテストできます。値がない場合はデフォルト値を設定することもできます。

- cfinput タグで hidden 属性を指定することにより、必須フィールドにデータを入力しないユーザーに対してヘルプメッセージを表示させることができます。この方法の詳細については、697 ページの「必須入力フィールド」を参照してください。

変数の存在のテスト

アプリケーションページ内に特定の変数が存在するかどうかをテストするには、IsDefined 関数を使用します。配列に特定のエントリが存在するかどうかを調べるには、ArrayIsDefined 関数を使用します。構造体に特定のキーが存在するかどうかを調べるには、StructKeyExists 関数を使用します。

たとえば、チェックボックスが設定されていないフォームを送信した場合、アクションページではチェックボックスの変数を受け取りません。次のフォームアクションページの例では、[Contractor] チェックボックスの Form 変数が存在することを使用前に確認しています。

```
<cfif IsDefined("Form.Contractor")>
    <cfoutput>Contractor: #Form.Contractor#</cfoutput>
    </cfif>
```

IsDefined 関数に渡す引数は、常に引用符で囲む必要があります。IsDefined 関数の詳細については、『CFML リファレンス』を参照してください。

値を表示する前に配列に要素が存在するかどうかをテストするには、次のような形式を使用します。

```
</cfoutput>
    <cfloop index="i" from="1" to="#Arraylen(myArray)#">
        <cfif ArrayIsDefined(myArray, i)>
            #i#: #myArray[i]#<br>
        </cfif>
    </cfloop>
</cfoutput>
```

IsDefined 関数と異なり、ArrayIsDefined 関数では変数名を引用符で囲みません。

定義されていない変数を評価しようとすると、ページが処理できず、エラーメッセージが表示されます。この問題の原因を特定するには、ColdFusion Administrator のデバッグ機能を有効にするか、エディタのデバッガを使用します。ColdFusion Administrator のデバッグ情報には、どの変数がアプリケーションページに渡されているかが示されます。

変数の存在に関する注意事項

存在をテストする変数が、構造体として使用可能なスコープに属している場合は、IsDefined 関数を使用するよりも、StructKeyExists 関数を使用するほうが、若干パフォーマンスが向上することがあります。

Form.fieldnames というビルトイン変数の内容を参照して、どの Form 変数が存在するか調べることもできます。この変数には、フォームから送信されたすべてのフィールドのリストが含まれています。ただし、フォームのテキストフィールドは必ずアクションページに送信され、ユーザーがデータを入力しなかった場合は空の文字列が含まれる可能性があります。

cfparam タグの使用

cfparam タグを使用すると変数の存在を確認できます。また、変数が存在しない場合のデフォルト値も設定できます。cfparam タグのシンタックスは次のとおりです。

```
<cfparam name="VariableName"
    type="data_type"
    default="DefaultValue">
```

注意：パラメータのデータ型を検証する type 属性の使用方法については、『CFML リファレンス』を参照してください。

変数の存在をテストする cfparam タグでは、テスト結果の処理方法を 2 通りの中から選択できます。

- 必須変数の存在を調べるために name 属性のみを指定する方法。変数が存在しない場合は、ページの処理が停止され、エラーメッセージが表示されます。

- オプション変数の存在を調べるために name 属性と default 属性を指定する方法。変数が存在する場合は処理が継続され、値は変更されません。変数が存在しない場合は変数が作成され、default 属性の値が設定され、処理が続行されます。

次の例では、cfparam タグでオプション変数が存在するかどうかを確認して、存在しない場合はデフォルト値を設定します。

```
<cfparam name="Form.Contract" default="Yes">
```

例 : 変数のテスト

name 属性を持つ cfparam を使用すると、処理を進める前にページまたはカスタムタグが受け取るべき変数を明確に定義できます。これによって、コードが読みやすくなるだけでなく、保守とデバッグが容易になります。

たとえば、次の cfparam タグは、このページでは StartRow と RowsToFetch という 2 つのフォーム変数が必要であることを示します。

```
<cfparam name="Form.StartRow">
    <cfparam name="Form.RowsToFetch">
```

いずれかのフォーム変数がない状態でこれらのタグを持つページが呼び出されると、エラーが発生し、ページの処理が停止します。デフォルトではエラーメッセージが表示されますが、275 ページの「エラー処理」の説明に従ってエラーを処理することもできます。

例 : デフォルト値の設定

次の例では、cfparam タグを使用してオプション変数の存在を確認します。変数が存在する場合は処理が継続されます。変数が存在しない場合は作成され、デフォルトの値が設定されます。

```
<cfparam name="Cookie.SearchString" default="temple">
<cfparam name="Client.Color" default="Gray">
<cfparam name="ShowExtraInfo" default="No">
```

条件論理式を使用する代わりに cfparam タグを使用して、URL 変数や Form 変数のデフォルト値を設定することもできます。たとえば、次のコードをアクションページに含めれば、SelectedDepts 変数が必ず存在するようになります。

```
<cfparam name="Form.SelectedDepts" default="Marketing,Sales">
```

データの検証

多くの場合、単に入力データが存在していても、正しい形式でなければ不十分です。たとえば、日付フィールドのデータは日付の形式で入力されている必要があります。年収フィールドのデータは、数値または通貨の形式で入力されている必要があります。データの有効性を確認するには、次のようなさまざまな方法があります。

- type 属性を指定した cfparam タグを使用して、変数を検証する。
- IsValid 関数を使用して変数を検証する。
- SQL の WHERE 節内で cfqueryparam タグを使用して、クエリーパラメータを検証する。
- 検証属性を持つ cfform コントロールを使用する。
- hidden 属性を指定した input タグを使用して、フォームの入力フィールドの内容を検証する。

注意：cfparam、cfqueryparam、および form タグを使用したデータ検証は、サーバーによって行われます。cfform タグおよび非表示フィールドを使用した検証は、サーバーにデータが送信される前に、ユーザーのブラウザで JavaScript によって行われます。

フォームおよび変数のデータ検証の詳細については、729 ページの「データの検証」を参照してください。クエリーパラメータの検証の詳細については、419 ページの「cfqueryparam の使用」を参照してください。

カスタムタグおよび UDF (ユーザー定義関数) への変数の受け渡し

次のセクションでは、CFML で記述したカスタムタグまたはユーザー定義関数や、Java または C++ で記述した CFX カスタムタグにデータを渡す場合のルールについて説明します。

CFML のタグおよび UDF への変数の受け渡し

変数を CFML カスタムタグに属性として渡した場合や、ユーザー定義関数に引数として渡した場合に、その変数のプライベートコピーが渡されるのか、呼び出しページ内の変数への参照のみが渡されるのかは、次のルールによって決まります。

- 単純変数および配列は、データのコピーとして渡されます。単純変数が複数含まれている式を引数に指定した場合は、式の評価結果が関数またはタグにコピーされます。
- 構造体、クエリー、および cfobject オブジェクトは、オブジェクトへの参照として渡されます。

呼び出しページ内のデータのコピーがタグまたは関数に渡された場合は、カスタムタグまたは関数内で変数を変更しても、呼び出しページ内の変数の値は変更されません。変数が参照として渡された場合は、カスタムタグまたは関数内で変数を変更すると、呼び出しページ内の変数値も変更されます。

カスタムタグに変数を渡す場合は、変数名を # 記号で囲む必要があります。関数に変数を渡す場合は、変数名を # 記号で囲みません。たとえば、次のコードでは 3 つの Form 変数を使用してユーザー定義関数を呼び出します。

```
<cfoutput>
    TOTAL INTEREST: #TotalInterest(Form.Principal, Form.AnnualPercent,Form.Months)#<br>
</cfoutput>
```

次の例では、MyString および MyArray という 2 つの変数を使用してカスタムタグを呼び出します。

```
<cf_testTag stringval=#MyString# arrayval=#MyArray#>
```

CFX タグへの変数の受け渡し

配列、構造体、または cfobject オブジェクトを CFX タグに渡すことはできません。クエリーについては、CFX タグを呼び出すときに query 属性を使用して渡すことができます。通常は、単純データ型を CFX タグに渡すと文字列に変換されます。ただし、Java リクエストインターフェイスの getIntAttribute メソッドを使用すると、渡された整数値を取得できます。

式と # 記号の使用

CFML では、cfoutput などの Adobe ColdFusion タグ、文字列、および式の中に式を挿入するときには、その式を # 記号で囲みます。また、変数名や文字列に変数を含めて、ダイナミック式やダイナミック変数を作成することもできます。

式

ColdFusion の式は、オペランドと演算子から構成されます。定数と変数はオペランドです。乗算記号などの演算子は、オペランドに対して作用する動詞です。関数も演算子の一種と見なすことができます。

最も基本的な式は単一のオペランドで構成され、演算子がありません。複雑な式は複数の演算子とオペランドで構成されます。ColdFusion の式の例を次に示します。

```
 12
MyVariable
a++
(1 + 1)/2
"father" & "Mother"
Form.divisor/Form.dividend
Round(3.14159)
```

演算子はオペランドに対して作用します。引数が 1 つだけの関数など、一部の演算子にはオペランドが 1 つしかありません。算術演算子や論理演算子など、多くの演算子にはオペランドが 2 つあります。オペランドが 2 つある式は、一般に次のように記述します。

```
Expression Operator Expression
```

式の間に演算子を置きます。それぞれの式は、単純なオペランド (変数または定数) にすることも、演算子と式で構成されるサブ式にすることも可能です。サブ式を使用すれば複雑な式が作成できます。たとえば、式 (1 + 1)/2 の場合、1 + 1 は 1 つの演算子と 2 つのオペランドで構成されるサブ式です。

演算子のタイプ

ColdFusion には、5 つのタイプの演算子があります。

- 算術
- ブール値
- 決定 (比較)
- 文字列
- 三項

関数も、オペランドに対して作用するので演算子と見なすことができます。

算術演算子

次の表で、算術演算子について説明します。

演算子	説明
+ - * /	加減乗除を行うための基本的な算術演算子です。 除算の場合は、右側のオペランドに 0 を指定できません。
++ --	インクリメントおよびデクリメント。変数を 1 増減します。 変数を増減した後に式で使用する場合は、x = ++ i のように記述します。これをプリインクリメントまたはプリデクリメントと呼びます。逆に、式で使用した後に変数を増減する場合は、x = i++ のように記述します。これをポストインクリメントまたはポストデクリメントと呼びます。たとえば、変数 i の初期値が 7 である場合、x = ++i とすると x の値は 8 になりますが、x=i++ とすると x の値は 7 になります。いずれの場合も、i の値は 8 になります。 これらの演算子は、f().a++ など、関数を含む式では使用できません。また、-++x のような式は使用できますが、---x や +++x は意味があいまいであるため、エラーになります。このような場合は、-(--x) や +(++x) のように、括弧を使用して演算子をグループ化できます。
+= -= *= /= %=	複合代入演算子。右辺の変数が、式の要素および結果変数の両方として使用されます。したがって、a += b という式は a = a +b と同じです。 複合代入演算子は、1 つの式に 1 つのみ使用できます。
+ -	単項算術演算子。数値の符号を設定します。
MOD または %	モジュロ。数値を除数で割った余りを返します。その結果には、除数と同じ符号が付きます。演算子の右側の値には整数を使用してください。数字以外の値を使用するとエラーが発生します。実数を指定した場合は小数部分が無視されます。たとえば、11 MOD 4.7 は 3 になります。
¥	整数の除算。整数を別の整数で割ります。結果も整数になります。たとえば、9¥4 は 2 になります。右側のオペランドには 0 を指定できません。
^	指数。指数で累乗した結果を返します。キャレット文字 (^) は数字と指数の間に挿入します。たとえば、2^3 は 8 になります。基数および指数には実数や負の値も使用できます。ただし、-1^.5 のように虚数になる式の結果は、文字列 "-1.#IND になります。ColdFusion では、虚数および複素数はサポートされていません。

ブール演算子

ブール (論理) 演算子は、論理結合子および否定の演算を実行します。ブール演算子のオペランドは、ブール値 (True/False) です。次の表で、ブール演算子について説明します。

演算子	説明
NOT または !	引数の値を反転します。たとえば、NOT True は False、NOT False は True です。
AND または &&	両方の引数が True の場合は True を返します。そうでない場合は False を返します。たとえば、True AND True は true ですが、True AND False は False です。
OR または ||	引数のいずれか一方でも True である場合は True を返します。そうでない場合は False を返します。たとえば、True OR False は True ですが、False OR False は False です。
XOR	排他的論理和 (Exclusive or)。1 つの引数のみが True である場合は True を返します。そうでない場合は False を返します。両方の引数が True または False である場合は False を返します。たとえば、True XOR True は False ですが、True XOR False は True です。
EQV	等価 (Equivalence)。両方のオペランドが True または False である場合は True を返します。EQV 演算子は XOR 演算子の逆です。たとえば、True EQV True は True ですが、True EQV False は False です。
IMP	包含 (Implication)。ステートメント A IMP B は、論理ステートメント "A ならば B" と同値です。A が True で B が False の場合にのみ、A IMP B は False です。そうでない場合はすべて True です。

決定演算子

ColdFusion の決定 (比較) 演算子は、結果としてブール値 True/False を生成します。多くの演算には、等価な演算子形式が複数存在します。たとえば、IS と EQ は同じ演算を実行します。次の表で、決定演算子について説明します。

演算子	説明
IS EQUAL EQ	大文字と小文字を区別しないで、2 つの値を比較します。2 つの値が同じである場合は、True を返します。
IS NOT NOT EQUAL NEQ	IS の逆です。大文字と小文字を区別しないで、2 つの値を比較します。2 つの値が同じでない場合は、True を返します。
CONTAINS	左側の値に右側の値が含まれている場合は、True を返します。
DOES NOT CONTAIN	CONTAINS の逆です。左側の値に右側の値が含まれていない場合は、True を返します。
GREATER THAN GT	左側の値が右側の値よりも大きい場合は、True を返します。
LESS THAN LT	GREATER THAN の逆です。左側の値が右側の値よりも小さい場合は、True を返します。
GREATER THAN OR EQUAL TO GTE GE	左側の値が右側の値よりも大きいか等しい場合は、True を返します。
LESS THAN OR EQUAL TO LTE LE	左側の値が右側の値よりも小さいか等しい場合は、True を返します。

注意：CFScript 式に限り、次の決定演算子も使用できます。これらは、タグ内の式では使用できません。== (EQ)、!= (NEQ)、> (GT)、< (LT)、>= (GTE)、<= (LTE)

決定演算子のルール

決定演算子には、次のルールが適用されます。

- CONTAINS または DOES NOT CONTAIN 以外の決定演算子を含んだ式が評価されるときには、まず、データを数値に変換できるかどうかが判定されます。変換できる場合は、データの数値比較が行われます。変換できない場合は、文字列比較が行われます。これによって、予期しない結果が生成される場合があります。詳細については、48 ページの「評価および型変換の問題」を参照してください。
- CONTAINS または DOES NOT CONTAIN を含んだ式が評価されるときには、文字列比較は行われません。式 A CONTAINS B は、B が A の部分文字列である場合に True と評価されます。したがって、次の式は True です。

```
123.45 CONTAINS 3.4
```

- ColdFusion の決定演算子は、大文字と小文字を区別せずに文字列を比較します。したがって、次の式は True です。

```
"a" IS "A"
```

- 決定演算子で文字列比較が行われる場合は、文字列の左から右に評価され、それぞれの位置で文字のソート順に基づいて判定が行われます。異なる文字が最初に現れた位置で、文字列の関係が判定されます。したがって、次の式は True です。

```
"ab" LT "aba"
"abde" LT "ac"
```

文字列演算子

文字列演算子は文字列を操作します。次の表で、文字列演算子について説明します。

演算子	説明
&	文字列を連結します。
&=	複合連結。右辺の変数が、連結演算の要素および結果変数の両方として使用されます。したがって、a &= b という式は a = a & b と同じです。 複合代入演算子は、1 つの式に 1 つのみ使用できます。

注意：クエリーオブクエリーでは、連結演算子として || を使用します。

三項演算子

三項演算子は、3 つのオペランドを取る決定演算子です。この演算子では、ブール式に基づいて変数に値が代入されます。この演算子の形式は次のようになります。

```
(Boolean expression)? expression1 : expresson2
```

ブール式が true と評価された場合、演算子の結果は expression1 となり、true でない場合は expression2 となります。

例 :

```
<cfset c = (a GT b)? a : b >
```

a が b より大きい場合は、c に値 a が代入されます。そうでない場合は、c に値 b が代入されます。

丸括弧にはブール値として評価される式であればどのような式でも入れることができ、a および b には有効な式であればどのような式でも指定できます。この演算子は他の式の中にネストできます。

演算子の優先順位と評価順序

式中の演算子が評価される順序は、演算子の優先順位によって決まります。優先順位は次のとおりです。EQUALS や GREATER THAN OR EQUAL TO など、演算子の代替名の一部は、簡潔のために省略しています。

```
 Unary +, Unary -
^
*, /
\
MOD
+, -
&
EQ, NEQ, LT, LTE, GT, GTE, CONTAINS, DOES NOT CONTAIN, ==, !=, >, >=, <, <=
NOT, !
AND, &&
OR, ||
XOR
EQV
IMP
```

標準以外の順序で評価するには、式を括弧で囲みます。次に例を示します。

- 6 - 3 * 2 は 0 です。
- (6 - 3) * 2 は 6 です。

括弧はネストできます。式中の演算子の評価順序がわからない場合は、括弧を使用して評価順序を明示してください。

関数を演算子として使用する

関数は演算子の形式の 1 つです。ColdFusion 関数は値を返すので、関数の結果をオペランドとして使用できます。関数引数は式です。たとえば、次の式は有効です。

- Rand()
- UCase("This is a text: ") & ToString(123 + 456)

関数のシンタックス

次の表に、関数のシンタックスと使用方法のガイドラインを示します。

使用方法	例
引数なし	Function()
基本形式	Function(Data)
関数のネスト	Function1(Function2(Data))
複数の引数	Function(Data1, Data2, Data3)
文字列の引数	Function(' これはデモです。') Function(' これはデモです。')
式の引数	Function1(X*Y, Function2(" テキスト "))

すべての関数は値を返します。次の cfset タグの例では、Now 関数で返された値が変数に設定されます。

```
<cfset myDate = DateFormat(Now(), "mmmm d, yyyy")>
```

関数で返される値を直接使用して、複合式を作成できます。次に例を示します。

```
Abs(Myvar)/Round(3.14159)
```

式に関数を挿入する方法の詳細については、63 ページの「# 記号の使用」を参照してください。

関数のオプションの引数

必須の引数の後にオプションの引数を取る関数があります。オプションの引数を省略した場合は、デフォルトで、あらかじめ定義された値が使用されます。次に例を示します。

- Replace("Eat and Eat", "Eat", "Drink") は "Drink and Eat" を返します。
- Replace("Eat and Eat", "Eat", "Drink", "All") は "Drink and Drink" を返します。

このように結果が異なるのは、Replace 関数が、置換範囲を指定する 4 番目のオプション引数を持っているからです。そのデフォルト値は "One" なので、初めの例では最初の "Eat" のみが "Drink" に置換されます。2 番目の例では、4 番目の引数が指定されているので、すべての "Eat" が "Drink" に置換されます。

式の評価と関数

ColdFusion では、関数が実行される前に、関数の属性が式として評価されます。したがって、関数の属性には ColdFusion 式を指定することができます。たとえば、次のようなコードがあるとします。

```
 <cfset firstVariable = "we all need">
<cfset myStringVar = UCase(firstVariable & " more sleep!")>
```

ColdFusion サーバーで 2 行目が実行されるときには、次の処理が行われます。

1. 文字列の連結を含んだ式が識別されます。
2. firstVariable 変数が文字列 "we all need" として評価されます。
3. 文字列 "we all need" と文字列 " more sleep!" が連結されて "we all need more sleep!" が取得されます。
4. 文字列 "we all need more sleep!" が UCase 関数に渡されます。
5. 文字列の引数 "we all need more sleep!" が UCase 関数で処理され、"WE ALL NEED MORE SLEEP!" が取得されます。
6. 変数 myStringVar に文字列値 "WE ALL NEED MORE SLEEP!" が代入されます。

上記の手順 1 ～ 3 は、関数が処理される前に実行されます。

1 つの式での複数の代入の使用

代入を連結すると、1 つのステートメントで複数の変数に同じ値を代入できます。これには式の結果の連結代入も含まれます。次のコードは連結代入の例を示しています。

```
a=b=c=d*5
```

複数の代入で var 演算子を使用することもできますが、この演算子を含む変数は他の変数よりも前に置く必要があります。次に例を示します。

```
//The following line is valid.
var a = var b = c = d*5
//The following line is not valid.
// a = b = var c = d*5
```

記号の使用

CFML では、# 記号は特別な意味を持ちます。ColdFusion サーバーでは、CFML テキスト (cfoutput タグの本文テキストなど) で # 記号が検出されると、# 記号で囲まれたテキストが変数または関数であるかどうかが確認されます。


記号はポンド記号とも呼ばれます。

変数または関数である場合は、テキストとそれを囲む # 記号が、変数値または関数の結果に置き換えられます。変数または関数でない場合はエラーが発生します。

たとえば、変数 Form.MyFormVariable の現在の値を出力するには、次のように変数名を # 記号で区切る (囲む) 必要があります。

```
<cfoutput>Value is #Form.MyFormVariable#</cfoutput>
```

この例では、変数 Form.MyFormVariable が、変数の値に置き換えられます。

記号を使用するときは、次のガイドラインに従ってください。

- # 記号は、プレーンテキストと変数または関数を区別するために使用します。
- # 記号で囲む変数または関数は 1 つのみです。たとえば、#Variables.myVar# や #Left(myString, position)# のようにします。ただし、#Left(trim(myString), position)# のように、# 記号で囲まれた関数に別の関数をネストすることができます。
- 1 + 2 のような複合式を # 記号で囲まないでください。cfoutput ブロックではそのような記述も可能ですが (たとえば <cfoutput>1 足す 1 は #1 + 1# です。</cfoutput> とすることは可能ですが)、その場合は、ロジックとプレゼンテーションが混在することになります。
- 不要な # 記号を使用すると処理に時間がかかるので、# 記号は必要な箇所でのみ使用してください。

記号を使用して変数名を作成する方法については、67 ページの「代入式で # 記号を使用して変数名を構築する」を参照してください。

ColdFusion タグの属性値での # 記号の使用

タグ属性の中に、# 記号で囲んだ変数、関数、または式を挿入することができます。たとえば、変数 CookieValue の値が "MyCookie" である場合、次の行の cfcookie の value 属性は、"The value is MyCookie" に設定されます。

```
<cfcookie name="TestCookie" value="The value is #CookieValue#">
```

次の例のように、属性値として使用する変数を囲む引用符は省略できます。

```
<cfcookie name = TestCookie value = #CookieValue#>
```

ただし、属性値をすべて引用符で囲むほうが HTML コーディングスタイルに一致します。

文字列式を使用して属性値を作成する場合は、次の例のように、式中の文字列を単一引用符 (') で囲むことによって、属性値を囲む引用符と区別します。

```
<cfcookie name="TestCookie2" value="The #CookieValue & 'ate the cookie!'#">
```

注意：cfset タグで、変数に別の変数の値を代入するときは、# 記号を使用する必要はありません。たとえば、<cfset newVar = oldVar> タグでは、新しい変数 newVar に oldVar 変数の値が代入されます。

タグ本文での # 記号の使用

次に示すタグの本文では、# 記号で囲むことで、変数や関数を使用できます。

- cfoutput
- cfquery
- cfmail

次に例を示します。

```
<cfoutput>
    Value is #Form.MyTextField#
</cfoutput>

<cfoutput>
    The name is #FirstName# #LastName#.
</cfoutput>

<cfoutput>
    The value of Cos(0) is #Cos(0)#
</cfoutput>
```

記号を省略すると、cfoutput ステートメントの出力には、値ではなくテキストが表示されます。

記号で囲まれた 2 つの式は、直接並べることができます。たとえば、次のようになります。

```
<cfoutput>
    "Mo" and "nk" is #Left("Moon", 2)##Mid("Monkey", 3, 2)#
</cfoutput>
```

これによって、次のテキストが表示されします。

"Mo" と "nk" で Monk です。

記号を重ねて使用しても、エスケープされた # 文字としては解釈されません。

文字列での # 記号の使用

文字列の中では、# 記号で囲むことで、変数や関数を使用できます。次に例を示します。

```
<cfset TheString = "Value is #Form.MyTextField#">
<cfset TheString = "The name is #FirstName# #LastName#.">
<cfset TheString = "Cos(0) is #Cos(0)#">
```

このテキストは、変数の値または関数で返された値で自動的に置き換えられます。たとえば、次の 2 つの cfset ステートメントは同じ結果を出力します。

```
<cfset TheString = "Hello, #FirstName#!">
<cfset TheString = "Hello, " & FirstName & "!">
```

文字列内で # 記号を省略すると、値ではなくテキストが文字列に表示されます。たとえば、次の 2 つの cfset ステートメントは同じ結果を出力します。

```
<cfset TheString = "Hello, FirstName!">
<cfset TheString = "Hello, " & "First" & "Name!">
```

cfoutput ステートメントの場合と同様に、文字列内で 2 つの式を直接並べることができます。次に例を示します。

```
<cfset TheString = "Monk is #Left("Moon", 2)##Mid("Monkey", 3, 2)#">
```

"Moon" および "Monkey" を囲む二重引用符 (") は、""Moon"" および ""Monkey"" のようにエスケープする必要はありません。# 記号の間のテキストは式と見なされ、その値が文字列に挿入される前に式として評価されるからです。

記号のネスト

式中で # 記号をネストできる場合があります。次の例では、# 記号をネストしています。

```
<cfset Sentence = "The length of the full name is #Len("#FirstName# #LastName#")#">
```

この例では、# 記号がネストしているので、変数 FirstName と LastName の値が文字列内に挿入されてから、その長さが Len 関数によって計算されます。

記号をネストすると、式が複雑になります。ネストしないように記述したほうが、読みやすく効率もよくなるのが一般的です。たとえば、上のコード例は、# 記号をネストしないで次のように記述することができます。

```
<cfset Sentence2 = "The length of the full name is #Len(FirstName & " " & LastName)#">
```

次のコードは、結果は同じですがさらに読みやすくなっています。

```
 <cfset FullName = "#FirstName# #LastName#">
<cfset Sentence = "The length of the full name is #Len(FullName)#">
```

次のように、関数の引数を # 記号で囲むのは誤りです。

```
 <cfset ResultText = "#Len(#TheText#)#">
<cfset ResultText = "#Min(#ThisVariable#, 5 + #ThatVariable#)#">
<cfset ResultText = "#Len(#Left("Some text", 4)#)#">
```

これらのステートメントはエラーになります。一般的なルールとして、関数の引数は # 記号で囲まないでください。

式中での # 記号の使用

不要な # 記号を使用すると式がわかりにくくなり、処理に時間がかかるので、式中の # 記号は必要な場合にのみ使用してください。変数を参照するときは、次の例のようにすることをお勧めします。

```
 <cfset SomeVar = Var1 + Max(Var2, 10 * Var3) + Var4>
```

次の例では不要な # 記号が使用されているので、前のステートメントよりも効率が悪くなります。

```
 <cfset #SomeVar# = #Var1# + #Max(Var2, 10 * Var3)# + #Var4#>
```

ダイナミック式およびダイナミック変数

多くの ColdFusion プログラマにとって、ダイナミック式を使用したコードを作成したり見たりする機会はあまりないかもしれませんが、ショッピングカートアプリケーションのように変数名が予測できない状況では、ダイナミック変数名が役に立ちます。

また、ColdFusion には IIf 関数も含まれています。この関数は、ほとんど場合、ダイナミック式なしで使用されます。この関数は、引数をダイナミックに評価します。ダイナミックに評価されないようにする場合は、DE 関数を使用します。IIF 関数の使用方法の詳細については、71 ページの「IIF 関数の使用」を参照してください。

ダイナミック変数について

ダイナミック変数とは、ダイナミックに名前が付けられる変数のことで、多くの場合、変化しない部分と変化する部分から構成されます。たとえば次の例では、変化する接頭辞と変化しない接尾辞によって変数名がダイナミックに生成されます。

```
 <cfset "#flavor#_availability" = "out of stock">
```

このようにダイナミック変数を使用する場合は、ダイナミック評価は不要です。

ダイナミック式とダイナミック評価について

ダイナミック式では、変数値だけでなく、実際の式も実行時に決定されます。つまりダイナミック式では、変数値だけでなく、変数名などの式の構造が実行時に構築されます。

ダイナミック式は、文字列式、つまり文字列に含まれている式 (引用符で囲まれた式) を使用して作成します。ダイナミック評価とは、文字列式を評価する処理のことです。ダイナミック評価を実行する関数は、Evaluate と IIf のみです。

ダイナミック評価は次の手順で実行されます。

1 文字列式が、文字列ではなく通常の式として処理されます。

2 式が解析されて、式の要素が決定されるとともに、式のシンタックスが検証されます。

3 式が評価されます。その際には、変数が値に置き換えられ、関数が呼び出され、その他の必要な操作が行われます。

この処理によって、変化する部分を持つダイナミック式が解釈されます。ただし、この処理ではかなりのオーバーヘッドが生じます。

ダイナミック式は、配列や構造体がサポートされる前の、初期バージョンの ColdFusion では重要な機能でした。現在でも特定の状況では役に立ちますが、構造体の機能や、連想配列の表記法を使用して構造体の要素にアクセスする機能を使用すれば、効率よく簡単にダイナミックなデータ管理が行えます。配列および構造体の使用方法については、75 ページの「配列および構造体の使用」を参照してください。

変数名の作成方法の選択

ダイナミック変数名が必要になる例を 2 つ示します。

- フォームのフィールドの数や名前がダイナミックに変化するフォームアプリケーション。この場合、フォームからアクションページに渡されるのはそのフィールドの名前と値のみです。アクションページでは、実際に作成されたフィールド名 (変数名) は把握できますが、すべてのフィールド名を把握することは不可能です。
- 次の条件が当てはまる場合
  - ColdFusion がカスタムタグを複数回呼び出す。
  - カスタムタグの結果を、毎回異なる変数に返す必要がある。
  - カスタムタグの結果を返す変数を、呼び出し側のコードが指定できる。

  この場合、カスタムタグは結果を返す変数名をあらかじめ把握できないので、それを属性値として取得します。

いずれの場合も、Evaluate 関数を使用したダイナミック式で変数名を構築する必要があるように思えるかもしれませんが、72 ページの「例 : ダイナミックショッピングカート」のようにダイナミック変数名を使用すれば、より効率的に同じ結果を得ることができます。

これは、ダイナミック評価は常に避けるべきであるという意味ではありません。しかし、ダイナミック評価はパフォーマンスコストを必要とするので、まず次の方法で同じ結果が得られないかを確認してください。

- 配列 (インデックス変数を使用)
- 式を使用して連想配列の表記法で構造体の要素にアクセスする
- 変数名をダイナミックに生成する

ダイナミック評価を使用しないダイナミック変数名

ColdFusion では、変化する部分を組み合わせてインラインで変数名を生成する方法を常に使用できるわけではありませんが、一般的な用途にはこの方法で対応できます。

代入式で # 記号を使用して変数名を構築する

cfset の代入式の左辺で、テキストと変数名を組み合わせて変数名を構築することができます。たとえば次のコードでは、変数 Product12 に文字列値 "Widget" が設定されます。

```
 <cfset ProdNo = 12>
<cfset "Product#ProdNo#" = "Widget">
```

この方法で変数名を構築する場合は、等号の左辺のテキストをすべて引用符で囲む必要があります。

この方法は、配列を使用する方法よりも効率が落ちます。次の例では、より少ない処理で前の例と同じ目的を達成できます。

```
 <cfset MyArray=ArrayNew(1)>
<cfset prodNo = 12>
<cfset myArray[prodNo] = "Widget">
```

ダイナミック変数の制約

代入式の左辺に引用符で囲んだダイナミック変数名を使用する場合、その名前は、単純変数名か、object.property の表記法を使用した複合名 (MyStruct.#KeyName# など) のいずれかである必要があります。ダイナミック変数名に配列を含めることはできません。たとえば、次のコードはエラーになります。

```
<cfset MyArray=ArrayNew(1)>
<cfset productClassNo = 1>
<cfset productItemNo = 9>
<cfset "myArray[#productClassNo##productItemNo#]" = "Widget">
```

ただし、代入式の左辺で引用符を使用せずに、配列のインデックス値を変数からダイナミックに作成することは可能です。たとえば、前述のサンプルコードは、最後の行を次の行に置き換えれば正しく動作します。

```
<cfset myArray[#productClassNo# & #productItemNo#] = "Widget">
```

構造体の参照をダイナミックに構築する

連想配列の表記法を使用して構造体を参照することができるので、変数を使用して構造体の参照をダイナミックに作成することができます。連想配列の表記法の詳細については、85 ページの「構造体の表記法」を参照してください。連想配列の表記法で構造体を参照するときには、インデックスの括弧内に ColdFusion 式を使用することができます。たとえば、product_1、product_2、... というキーを持つ productName 構造体がある場合、次のコードを実行すると productName.product_3 の値が表示されます。

```
<cfset prodNo = 3>
<cfoutput>
    Product_3 Name: #ProductName["product_" & prodNo]#
<cfoutput>
```

この形式を使用したショッピングカート管理の例については、72 ページの「例 : ダイナミックショッピングカート」を参照してください。

ダイナミック評価の使用

ダイナミック評価とダイナミック式には、いくつかの特徴や注意事項があります。

ColdFusion ダイナミック評価関数

次の表に、ダイナミック評価を実行する関数と、ダイナミック式の評価に役立つ関数を示します。

関数	用途
DE	引数内の二重引用符 (") をエスケープし、結果を二重引用符で囲みます。DE 関数は、出力文字列が IIF 関数で評価されないようにしたい場合に便利です。 IIF 関数で DE 関数を使用する例については、71 ページの「IIF 関数の使用」を参照してください。
Evaluate	1 つまたは複数の文字列式を引数に取り、それらの内容を式と見なして、左から右にダイナミックに評価していきます。左側の式の評価結果が、右側の式に影響を与えることがあります。右端の引数の評価結果を返します。 この関数の詳細については、69 ページの「Evaluate 関数について」を参照してください。
IIf	ブール条件式を評価します。この式が True であるか False であるかによって、2 つの文字列式のいずれかをダイナミックに評価し、その結果を返します。IIF 関数は、cfif タグを HTML にインラインで組み込みたい場合に便利です。 この関数の使用例については、71 ページの「IIF 関数の使用」を参照してください。
PrecisionEvaluate	任意精度の 10 進演算が可能である点を除いて、Evaluate 関数と同じです。数式内のオペランドが 10 進数であり (12947834.986532 など)、ColdFusion の数値データ型で正確に表現できないほど長い場合、この関数は任意精度の演算を行って結果を計算し、その結果を任意長の数値の文字列として返します。この関数の詳細については、『CFML リファレンス』の PrecisionEvaluate を参照してください。
SetVariable	最初の引数で表現されている変数に、2 番目の引数で指定されている値を設定します。現在、適切な形式で記述された ColdFusion ページでは、この関数を使用する必要はなくなりました。71 ページの「SetVariable 関数に関する注意事項」を参照してください。

関数の引数の評価に関する注意事項

ColdFusion では、引数の値が関数に渡される前に、引数の評価が必ず実行されることに注意する必要があります。

たとえば、次の DE 関数の例を考えてみます。

```
<cfoutput>#DE("1" & "2")#</cfoutput>
```

この結果は """1"" & ""2""" となるように思うかもしれませんが、ColdFusion では次のような処理が行われるので、実際には "12" と表示されます。

1 式 "1" & "2" が文字列 "12" と評価されます。

2 文字列 "12" (引用符は除く) が DE 関数に渡されます。

3 DE 関数が呼び出され、12 の前後に引用符が付加されます。

同様に、DE(1 + 2) という式の場合は、1 + 2 が整数 3 と評価されて、関数に渡されます。関数によって整数 3 が文字列に変換され、リテラル引用符で囲まれて "3" という結果が返されます。

Evaluate 関数について

Evaluate 関数は、1 つまたは複数の文字列式を引数に取り、それらの内容を式と見なして左から右にダイナミックに評価していき、右端の引数の評価結果を返します。

ここでは、次のコードを例に取って、Evaluate 関数と ColdFusion 変数の処理方法を説明します。

```
 <cfset myVar2="myVar">
<cfset myVar="27/9">
<cfoutput>
    #myVar2#<br>
    #myVar#<br>
    #Evaluate("myVar2")#<br>
    #Evaluate("myVar")#<br>
    #Evaluate(myVar2)#<br>
    #Evaluate(myVar)#<br>
</cfoutput>
```

コードの説明

このコードが ColdFusion で処理される手順を、次の表に示します。

コード	説明
<cfset myVar2="myVar"> <cfset myVar="27/9">	2 つの変数に次の文字列が設定されます。 myVar 27/9
<cfoutput> #myVar2# #myVar# 	各変数に代入されている myVar および 27/9 という値が表示されます。
#Evaluate("myVar2")# 	文字列 "myvar2" (引用符は除く) が Evaluate 関数に渡され、次の処理が行われます。 1. 変数 myVar2 と評価されます。 2. myVar2 変数の値である文字列 "myvar" (引用符は除く) が返されます。

コード	説明
#Evaluate("myVar")# 	文字列 "myvar" (引用符は除く) が Evaluate 関数に渡され、次の処理が行われます。 1. 変数 myVar と評価されます。 2. myVar 変数の値である文字列 "27/9" (引用符は除く) が返されます。
#Evaluate(myVar2)# 	変数 myVar2 が文字列 "myVar" と評価され、この文字列 (引用符は除く) が Evaluate 関数に渡されます。残りの処理は、前のコードと同じです。
#Evaluate(myVar)# </cfoutput>	変数 myVar が文字列 "27/9"(引用符は除く) と評価されます。この文字列が Evaluate 関数に渡され、次の処理が行われます。 1. この文字列が式 27/9 と評価されます。 2. 除算が実行されます。 3. その結果である値 3 が返されます。

このように、ダイナミック式を使用すると、式を評価するためにかなりのオーバーヘッドが生じ、コードもわかりにくくなります。したがって、インデックス配列や構造体などの単純な方法で同じ処理が行える場合は、ダイナミック式を使用しないことをお勧めします。

Evaluate 関数の使用の回避

Evaluate 関数を使用すると、処理オーバーヘッドが増大します。ほとんどの場合、この関数を使用する必要はありません。次の例は、Evaluate 関数を使用するかどうかを検討したほうがよいケースを示しています。

例 1

次のようなコードでは、Evaluate 関数を使用したくなるかもしれません。

```
<cfoutput>1 + 1 is #Evaluate(1 + 1)#</cfoutput>
```

このコードは動作しますが、次のコードのほうが効率的です。

```
<cfset Result = 1 + 1>
<cfoutput>1 + 1 is #Result#</cfoutput>
```

例 2

この例は、Evaluate 関数の代わりに連想配列の表記法を使用する方法を示しています。この方法は強力です。その理由は次のとおりです。

- ほとんどの ColdFusion スコープは構造体としてアクセスできます。
- 連想配列の表記法で構造体を参照するときには、インデックスに ColdFusion 式を使用することができます。連想配列の表記法で構造体を参照する方法の詳細については、85 ページの「構造体の表記法」を参照してください。

次の例では、Evaluate 関数を使用して変数名を生成しています。

```
<cfoutput>
    Product Name: #Evaluate("Form.product_#i#")#
</cfoutput>
```

このコードは、ショッピングカートに入っている不定数の品目に対してエントリが作成されるフォームで使用されているものです。ショッピングカートの各品目に対して、製品名フィールドが用意されます。このフィールドの名前は、product_1、product_2 という形式になっています。数字の部分は、ショッピングカート内の製品エントリを表します。この例では、ColdFusion によって次の処理が行われます。

1. 変数 i がその値 (1 など) に置き換えられます。
2. その変数値と "Form.product_" が連結され、その結果 (Form.product_1) が Evaluate 関数に渡されます。以降の手順は、この関数によって実行されます。

3 変数 product_1 が解析され、変数の実行可能表現が生成されます。ColdFusion はパーサーを実行する必要があるので、単純変数であってもこの手順には時間がかかります。

4 変数表現が評価されます。たとえば、" 空気銃 " などと評価されます。

5 変数の値が返されます。

次の例を使用すれば、前の例と同じ結果を、より効率的に得ることができます。

```
<cfoutput>
    ProductName: #Form["product_" & i]#
</cfoutput>
```

このコードでは、ColdFusion によって次の処理が行われます。

1 連想配列のインデックスの括弧内にある式が評価され、文字列 "product_" に変数 i の値を連結するものと解釈されます。

2 変数 i の値が 1 と決定されます。

3 文字列と変数値が連結されて product_1 が取得されます。

4 その結果が Form 構造体のキー値として使用され、Form[product_1] が取得されます。この連想配列の表記法を使用した参照でも、object.attribute という形式の参照である Form.product_1 と同じ値 (この例では " 空気銃 ") が取得されます。

このコードではダイナミック評価を使用していませんが、文字列と変数を使用することで、構造体への参照を同じくダイナミックに作成しています。

SetVariable 関数に関する注意事項

次のような方法を使用して変数名をダイナミックに生成すれば、SetVariable 関数の使用を避けることができます。たとえば、次の 2 つのコードは同等です。

```
<cfset SetVariable("myVar" & i, myVal)>
<cfset "myVar#i#" = myVal>
```

2 番目のコードでは、変数名 myVar#i# が引用符で囲まれているので、ColdFusion によってその名前の評価が行われ、# 記号で囲まれているテキストが変数または関数として処理されます。#i# は変数 i の値に置き換えられます。i の値が 12 の場合、このコードは次のコードと等しくなります。

```
<cfset myVar12 = myVal>
```

この方法の詳細については、67 ページの「代入式で # 記号を使用して変数名を構築する」を参照してください。

IIF 関数の使用

IIf 関数は、次のコードを簡略化したものです。

```
<cfif argument1>
    <cfset result = Evaluate(argument1)>
<cfelse>
    <cfset result = Evaluate(argument2)>
</cfif>
```

この関数は、result 変数の値を返します。これは、JavaScript および Java の ? : 演算子に似ており、コードを簡潔に記述できます。したがって、IIF 関数は、ダイナミック式を使用しない場合にも便利です。

IIF 関数では、評価を行わないリテラル文字列は DE 関数の中に入れる必要があります。次に例を示します。

```
<cfoutput>
    #IIf(IsDefined("LocalVar"), "LocalVar", DE("The variable is not defined."))#
</cfoutput>
```

文字列 "The variable is not defined." を DE 関数で囲まないと、IIF 関数が文字列の内容を式として評価しようとするので、エラーになります。この場合は、無効なパーサー構文エラーになります。

IIF 関数は、HTML コードにインラインで ColdFusion ロジックを組み込む場合に便利ですが、ダイナミック式の評価を必要としない場合には、処理に時間がかかるという欠点があります。

次の例では、IIF を使用して表の行の背景色を 1 行おきに白と灰色に設定します。また、DE 関数を使用して、ColdFusion が色文字列を評価しないようにしています。

```
<cfoutput>
    <table border="1" cellpadding="3">
        <cfloop index="i" from="1" to="10">
            <tr bgcolor="#IIF( i mod 2 eq 0, DE("white"), DE("gray") )#">
            <td>
                hello #i#
            </td>
            </tr>
        </cfloop>
    </table>
</cfoutput>
```

このコードは、IIF や DE を使用しない次の例よりも簡潔です。

```
<cfoutput>
<table border="1" cellpadding="3">
    <cfloop index="i" from="1" to="10">
        <cfif i mod 2 EQ 0>
            <cfset Color = "white">
        <cfelse>
            <cfset Color = "gray">
        </cfif>
        <tr bgcolor="#color#">
            <td>
                hello #i#
            </td>
        </tr>
    </cfloop>
</table>
</cfoutput>
```

例 : ダイナミックショッピングカート

次の例では、連想配列の表記法を使用することで、ダイナミック式の評価を行わずに変数名をダイナミックに作成および操作しています。

ショッピングカートのように、必要な出力がダイナミックに生成され変化するようなアプリケーションでは、変数名をダイナミックに生成する必要があります。ショッピングカートでは、カートのエントリ数やその内容を予測することは不可能です。また、フォームを使用しているので、アクションページが受け取るのはフォームフィールドの名前と値を持つ Form 変数のみに限られます。

次の例では、ショッピングカートの内容が表示され、ユーザーは注文内容を編集して送信できます。例を単純にするために、ユーザーがカートに項目を入れるのではなく、CFScript を使用して自動的にショッピングカートの内容を生成しています。より完全な例では、ユーザーの選択した項目をショッピングカートに入れることになります。またこの例では、注文の確認や発注などのビジネスロジックも省略されています。

フォームの作成

1 エディタでファイルを作成します。

```
 <html>
<head>
    <title>Shopping Cart</title>
</head>
<cfscript>
CartItems=4;
Cart = ArrayNew(1);
for ( i=1; i LE cartItems; i=i+1)
{
    Cart[i]=StructNew();
    Cart[i].ID=i;
    Cart[i].Name="Product " & i;
    Cart[i].SKU=i*100+(2*i*10)+(3*i);
    Cart[i].Qty=3*i-2;
}
</cfscript>

<body>
Your shopping cart has the following items.<br>
You can change your order quantities.<br>
If you don't want any item, clear the item's check box.<br>
When you are ready to order, click submit.<br>
<br>
<cfform name="ShoppingCart" action="ShoppingCartAction.cfm" method="post">
<table>
    <tr>
    <td>Order?</td>
    <td>Product</td>
    <td>Code</td>
    <td>Quantity</td>
    </tr>
    <cfloop index="i" from="1" to="#cartItems#">
        <tr>
        <cfset productName= "product_" & Cart[i].ID>
        <cfset skuName= "sku_" & Cart[i].ID>
        <cfset qtyname= "qty_" & Cart[i].ID>
        <td><cfinput type="checkbox" name="itemID" value="#Cart[i].ID#" checked>
             </td>
        <td><cfinput type="text" name="#productName#" value="#Cart[i].Name#"
             passThrough = "readonly = 'True'"></td>
        <td><cfinput type="text" name="#skuName#" value="#Cart[i].SKU#"
             passThrough = "readonly = 'True'"></td>
        <td><cfinput type="text" name="#qtyName#" value="#Cart[i].Qty#">
             </td>
        </tr>
    </cfloop>
</table>
<input type="submit" name="submit" value="submit">
</cfform>

</body>
</html>
```

2 このファイルに "ShoppingCartForm.cfm" という名前を付けて保存します。

コードの説明

このコードについて、次の表で説明します。

コード	説明

<cfscript>
CartItems=4;
Cart = ArrayNew(1);
for (i=1; i LE cartItems; i=i+1)
{
Cart[i]=StructNew();
Cart[i].ID=i;
Cart[i].Name="Product " & i;
Cart[i].SKU=i*100+(2*i*10)+(3*i);
Cart[i].Qty=3*i-2;
}
</cfscript>
	構造体の配列としてショッピングカートを作成します。各構造体には、カート品目 ID、製品名、SKU 番号、およびそのカートでの品目の注文個数が含まれます。CartItems の数だけループし、ループカウンタに基づいて構造体の変数に値を設定して、ショッピングカートに製品を入れます。実際のアプリケーションでは、一般に、名前、SKU、および数量の値は別のページで設定します。

<cfform name="ShoppingCart" action="ShoppingCartAction.cfm" method="post"> <table> <tr> <td>Order?</td> <td>Product</td> <td>Code</td> <td>Quantity</td> </tr>	フォームとそれに埋め込まれたテーブルを開始します。ユーザーが送信ボタンをクリックすると、フォームのデータが "ShoppingCartAction.cfm" ページに送信されます。 テーブルによってフォームの形式を整えています。テーブルの最初の行は、列のヘッダです。その後の各行に、カート内の品目のデータが表示されます。
<cfloop index="i" from="1" to="#cartItems#"> <tr> <cfset productName= "product_" & Cart[i].ID> <cfset skuName= "sku_" & Cart[i].ID> <cfset qtyname= "qty_" & Cart[i].ID> <td><cfinput type="checkbox" name="itemID" value="#Cart[i].ID#" checked> </td> <td><cfinput type="text" name="#productName#" value="#Cart[i].Name#" passThrough = "readonly = 'True'"></td> <td><cfinput type="text" name="#skuName#" value="#Cart[i].SKU#" passThrough = "readonly = 'True'"></td> <td><cfinput type="text" name="#qtyName#" value="#Cart[i].Qty#"> </td> </tr> </cfloop> </table>	ショッピングカートのエントリをループして、カートフォームをダイナミックに生成しています。各ループで、フィールドタイプ識別子 (sku_ など) の後にカート品目 ID (Cart[i].ID) を付加して、フォームフィールドの name 属性に使用する変数を生成しています。 すべてのチェックボックスに対して、"itemID" という同じ名前を使用しています。したがって、アクションページに送信される itemID の値は、すべてのチェックボックスフィールドの値が列挙されたリストになります。各品目のチェックボックスフィールドの値は、カート品目 ID です。 行の各列は、カート品目の構造体のエントリになっています。passthrough 属性によって、製品名と SKU フィールドが読み取り専用に設定されています。単一引用符 (') を使用していることに注意してください。cfinput タグの passthrough 属性の詳細については、『CFML リファレンス』を参照してください。チェックボックスはデフォルトで選択されています。
<input type="submit" name="submit" value="submit"> </cfform>	送信ボタンを作成してフォームを終了します。

アクションページの作成

1 エディタでファイルを作成します。

2 次のテキストを入力します。

```
 <html>
<head>
    <title>Your Order</title>
</head>
<body>
<cfif isDefined("Form.submit")>
    <cfparam name="Form.itemID" default="">
    <cfoutput>
        You have ordered the following items:<br>
        <br>
        <cfloop index="i" list="#Form.itemID#">
            ProductName: #Form["product_" & i]#<br>
            Product Code: #Form["sku_" & i]#<br>
            Quantity: #Form["qty_" & i]#<br>
            <br>
        </cfloop>
    </cfoutput>
</cfif>
</body>
</html>
```

3 このファイルに "ShoppingCartAction.cfm" という名前を付けて保存します。

4 ブラウザで "ShoppingCartform.cfm" を開き、チェックボックスと数量の値を変更し、送信ボタンをクリックします。

コードの説明
このコードについて、次の表で説明します。

コード	説明
`<cfif isDefined("Form.submit")>`	フォーム送信によって呼び出された場合にのみ、このページの CFML を実行します。これは、フォームページとアクションページを分離している場合には必要ありませんが、1 つの ColdFusion ページにフォームとアクションを記述している場合には必要です。
`<cfparam name="Form.itemID" default="">`	Form.itemID のデフォルトを空の文字列に設定します。これは、ユーザーがすべてのチェックボックスをオフにしてフォームを送信しても (製品 ID が 1 つも送信されない場合でも)、ColdFusion でエラーが表示されないようにするためです。
`<cfoutput>` `You have ordered the following items: ` ` `<cfloop index="i" list="#Form.itemID#">` `ProductName: #Form["product_" & i]# ` `Product Code: #Form["sku_" & i]# ` `Quantity: #Form["qty_" & i]# ` ` `</cfloop>` `</cfoutput>` `</cfif>`	注文された各品目について、名前、SKU 番号、および数量を表示します。 フォームページから送信される Form.itemID は、すべてのチェックボックスの value 属性が列挙されたリストです。この属性は、選択されたショッピングカート品目の ID を表します。リストの値をインデックスに使用してループし、注文した各品目を出力しています。 連想配列の表記法を使用して Form スコープに構造体としてアクセスし、配列のインデックスで式を使用してフォーム変数名を生成しています。この式では、フィールド名のフィールドタイプ接頭辞を表す文字列 ("sku_" など) に、ショッピングカートの ItemID 番号を表す変数 i (これはループのインデックス変数として使用されています) を連結しています。

配列および構造体の使用

Adobe ColdFusion では、ダイナミックな多次元配列がサポートされています。配列を使用すると、ColdFusion のアプリケーションコードで強力な処理が行えます。

Adobe ColdFusion では、キーと値のペアのリストを管理する構造体もサポートされています。構造体には、他の構造体や複合データ型を値として含めることができるので、複雑なデータを柔軟に管理できる強力なツールとして使用できます。

配列について

配列は、データを保持するためのテーブル構造です。大きさと次元が明確に定義されたスプレッドシートのテーブルに似ています。

ColdFusion では、データを一時的に保存する場所として配列がよく使用されます。たとえば、オンラインで商品を注文できるサイトの場合は、ショッピングカートの内容を配列に保持しておくことができます。配列を使用すると、注文が確定されるまでの間にショッピングカートの内容が変化しても、データベースに情報をコミットせずに済むようになります。

配列の基本概念

ColdFusion 配列の説明では、次の用語を使用します。

配列の次元 配列構造の相対的な複雑さの度合い。

インデックス 特定の次元における要素の位置。通常は、my1Darray[1]、my2Darray[1][1]、my3Darray[1][1][1] のように角括弧で囲みます。

配列の要素 : 配列のインデックスが示す位置に保存されているデータ。

最も単純な配列は、1 次元の配列です。これは、テーブルの単一の行に似ています。1 次元配列には、名前 (変数名) と数値インデックスがあります。インデックス番号は、配列内の 1 つのエントリ (セル) を参照します。

したがって、次のステートメントでは、MyArray という 1 次元配列の 5 番目のエントリの値が "Robert" に設定されます。

```
<cfset MyArray[5] = "Robert">
```

2 次元配列は、簡単なテーブルに似ています。3 次元配列は、キューブ状のデータと考えることができます。ColdFusion で直接作成できる配列は、3 次元までです。4 次元以上の配列は、複数のステートメントを使用して作成します。

my2darray[1][3]="Paul" というシンタックスは、My2dArray という 2 次元配列のインデックス [1][3] の要素に、値 'Paul' を代入することを表します。

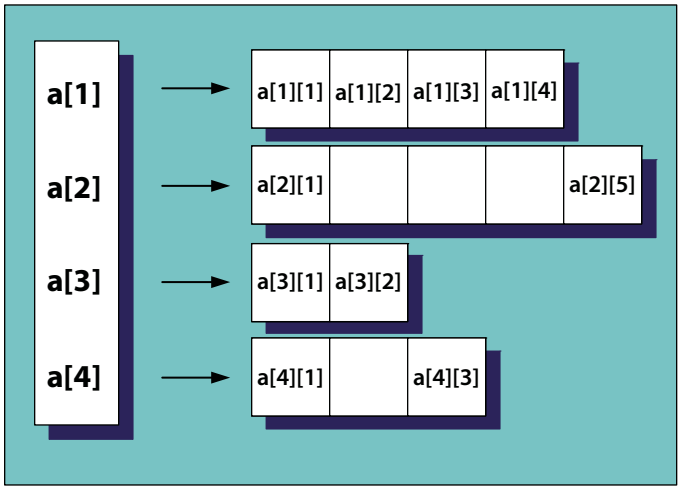
ColdFusion 配列について

ColdFusion の配列は、一般的な配列と異なり、ダイナミックな構造を持っています。たとえば、一般的な配列はサイズが一定で対称形です。これに対して、ColdFusion の配列では、データの追加または削除が可能で、各行の長さが異なります。

一般的な 2 次元配列は、個々のセルで構成された固定サイズのテーブルに似ています。

ColdFusion の 2 次元配列は、実際には、一連の 1 次元配列が含まれている 1 次元配列です。1 つの行を構成する各配列は、他の列とは独立して拡大したり縮小したりできます。

次の図は、ColdFusion の 2 次元配列です。



ColdFusion の 3 次元配列は、1 次元配列を 3 回ネストした配列です。一般的な 3 次元配列と ColdFusion の 3 次元配列にも同様な違いがありますが、紙面で表すのは難しいので 2 次元で説明します。

ダイナミックな配列は、データを追加すると拡大され、データを削除すると縮小されます。

配列の基本的な使用方法

配列要素の参照

配列要素を参照するには、インデックスを角括弧で囲んで、**arrayName[x]** のように記述します。**x** は、指定するインデックスです。ColdFusion の配列インデックスは、1 から数えます。たとえば、firstname という配列の最初の要素を参照するには、firstname[1] と記述します。2 次元配列では、myarray[1][1] のように 2 つの座標を使用してインデックスを指定します。

角括弧の中では、次のように、ColdFusion 変数や式を使用してインデックスを指定することができます。

```
<cfset myArray=ArrayNew(1)>
<cfset myArray[1]="First Array Element">
<cfset myArray[1 + 1]="Second Array" & "Element">
<cfset arrayIndex=3>
<cfset arrayElement="Third Array Element">
<cfset myArray[arrayIndex]=arrayElement>
<cfset myArray[++arrayIndex]="Fourth Array Element">
<cfdump var=#myArray#>
```

注意：IsDefined 関数では、配列要素が存在するかどうかはテストできません。したがって、未定義の配列要素にアクセスする可能性がある場合は、そのコードを try ブロックの中に配置して、要素が存在しない場合の例外を catch ブロックで処理してください。

配列の作成

ColdFusion での配列の作成方法には、明示的な方法と暗黙的な方法の 2 通りがあります。明示的な方法では、関数を使用して配列を宣言し、それにデータを代入します。暗黙的な方法では、代入ステートメントを使用します。単純な配列も複雑な (多次元の) 配列も作成できます。

関数による配列の作成

配列を明示的に作成するには、arrayNew 関数を使用して、配列の次元を指定します。次に例を示します。

```
<cfset myNewArray=ArrayNew(2)>
```

この行によって、myNewArray という 2 次元配列が作成されます。この方法を使用して作成できる配列は、3 次元までです。

配列を作成したら、配列要素を追加します。配列要素を参照するには、要素のインデックスを使用します。

たとえば、firstname という 1 次元配列を作成します。

```
<cfset firstname=ArrayNew(1)>
```

この firstname 配列にはデータがなく、長さも指定されていません。ここで、配列にデータを追加します。

```
<cfset firstname[1]="Coleman">
<cfset firstname[2]="Charlie">
<cfset firstname[3]="Dexter">
```

これらの名前を配列に追加すると、配列の長さは 3 になります。

配列の暗黙的な作成と使用

配列を暗黙的に作成する場合は、ArrayNew 関数を使用しません。その代わり、代入ステートメントの左辺で新しい変数名を指定し、代入ステートメントの右辺で配列表記を使用します。次に例を示します。

```
<cfset firstnameImplicit=["Coleman","Charlie","Dexter"]>
```

この単一のステートメントは、77 ページの「関数による配列の作成」で firstname 配列を作成するために使用した 4 つのステートメントに相当します。

配列を暗黙的に作成する場合、代入ステートメントの右辺では、配列の内容を囲む角括弧 ([]) と、個々の配列要素を区切るカンマを使用します。要素はリテラル値 (この例の文字列など)、変数、または式です。変数を指定する場合、変数名は引用符で囲みません。

空の配列を暗黙的に作成するには、次の例のようにします。

```
<cfset myArray = []>
```

また、単一のエントリを割り当てることで、配列を暗黙的に作成することもできます。次に例を示します。

```
<cfset chPar[1] = "Charlie">
<cfset chPar[2] = "Parker">
```

暗黙的な作成を行うときは、配列をネストしたり、構造体をネストしたり、配列と構造体をネストしたりすることはできません。したがって、単一の暗黙的なステートメントで多次元配列を作成することはできません。たとえば、次のどちらのステートメントも有効ではありません。

```
 <cfset myArray = [[],[]]>
<cfset jazzmen = [["Coleman","Charlie"],["Hawkins", "Parker"]]
```

2 次元配列を作成する場合は、次のように記述します。

```
 <cfset ch = ["Coleman", "Hawkins"]>
<cfset cp = ["Charlie", "Parker"]>
<cfset dg = ["Dexter", "Gordon"]>
<cfset players = [ch, cp, dg]>
```

配列を暗黙的に作成する場合、ダイナミック変数は使用できません。たとえば、次の式ではエラーが発生します。

```
 <cfset i="CP">
<cfset "#i#"=["Charlie","Parker"]>
```

複雑な多次元配列の作成

ColdFusion では、ダイナミックな多次元配列がサポートされています。ArrayNew 関数を使用して配列を宣言する場合は、次元数を指定できます。また、非対称配列を作成したり、配列を配列要素としてネストすることで配列の次元数を増やすことができます。

ある配列 (配列 1) を別の配列 (配列 2) の要素として割り当てると、配列 1 が配列 2 の中にコピーされることに注意してください。コピー元の配列 1 は、配列 2 とは関係なくそのまま存在します。したがって、2 つの配列の内容はそれぞれ独立に変更できます。

非対称配列を理解するには、実際の配列を見るのが効果的です。次の例では、非対称の多次元配列を作成し、作成した配列の構造を cfdump タグで表示しています。まだデータを含んでいない配列要素もあります。

```
 <cfset myarray=ArrayNew(1)>
<cfset myotherarray=ArrayNew(2)>
<cfset biggerarray=ArrayNew(3)>

<cfset biggerarray[1][1][1]=myarray>
<cfset biggerarray[1][1][1][10]=3>
<cfset biggerarray[2][1][1]=myotherarray>
<cfset biggerarray[2][1][1][4][2]="five deep">

<cfset biggestarray=ArrayNew(3)>
<cfset biggestarray[3][1][1]=biggerarray>
<cfset biggestarray[3][1][1][2][3][1]="This is complex">
<cfset myarray[3]="Can you see me">

<cfdump var=#biggestarray#><br>
<cfdump var=#myarray#>
```

注意：cfdump タグを使用すると、配列の内容をすべて表示できます。このタグは、配列や配列処理コードのデバッグに役立ちます。

コードの説明

このコードについて、次の表で説明します。

コード	説明

`<cfset myarray=ArrayNew(1)>`
`<cfset myotherarray=ArrayNew(2)>`
`<cfset biggerarray=ArrayNew(3)>`
	1 次元配列、2 次元配列、3 次元配列の 3 つの空の配列を作成します。

`<cfset biggerarray[1][1][1]=myarray>` `<cfset biggerarray[1][1][1][10]=3>`	3 次元配列 biggerarray の要素 [1][1][1] に、1 次元配列をコピーします。その配列の要素 [1][1][1][10] に 3 を代入します。 これで biggerarray 配列が非対称になりました。たとえば、この配列には要素 [1][1][2][1] がありません。
`<cfset biggerarray[2][1][1]=myotherarray>` `<cfset biggerarray[2][1][1][4][2]="five deep">`	3 次元配列の要素 [2][1][1] を 2 次元配列にし、要素 [2][1][1][4][2] に値 "five deep" を代入します。 これで biggerarray 配列がさらに非対称になりました。

`<cfset biggestarray=ArrayNew(3)>`
`<cfset biggestarray[3][1][1]=biggerarray>`
`<cfset biggestarray[3][1][1][2][3][1]="This is complex">`
	2 つ目の 3 次元配列を作成します。この配列の要素 [3][1][1] に biggerarray 配列をコピーし、要素 [3][1][1][2][3][1] に値を代入します。
	この配列は、複雑で非対称です。

`<cfset myarray[3]="Can you see me">`	myarray の要素 [3] に値を代入します。
`<cfdump var=#biggestarray#> ` `<cfdump var=#myarray#>`	cfdump を使用して biggestarray と myarray の構造を表示します。 myarray には "Can you see me" というエントリがありますが、biggestarray にはありません。これは、biggestarray には元の myarray の値がコピーされており、その後で myarray が変更されても biggestarray には反映されないからです。

暗黙的に作成された配列の使用

暗黙的に作成された配列は、関数 (ユーザー定義関数を含む) やタグの中で直接使用できます。たとえば、次のコードでは暗黙的な配列を 2 つ使用しています。1 つは ColdFusion 関数の中で、もう 1 つはユーザー定義関数の中で使用しています。

```
<cffunction name="sumarray">
    <cfargument name="inarray" type="array">
       <cfset result = 0>
    <cfloop array="#inarray#" index="i" >
        <cfset result += i>
        </cfloop>
        <cfreturn result>
</cffunction>

<cfoutput>
Summed Implicit array [#ArrayToList([1,2,3,4,5,6])#]: #sumarray([1,2,3,4,5,6])#<br />
</cfoutput>
```

配列への要素の追加

配列に要素を追加するには、要素に値を代入するか、または ColdFusion 関数を使用します。

代入による配列要素の追加

次の cfset タグのように、配列要素の値を定義することで、配列に要素を追加できます。

```
<cfset myarray[5]="Test Message">
```

指定したインデックスに要素が存在しない場合は、ColdFusion によって要素が作成されます。指定したインデックスに要素が存在する場合は、ColdFusion によって新しい値に置き換えられます。既存のデータの上書きを避けるには、次のセクションで説明するように、ArrayInsertAt 関数を使用します。

指定したインデックより小さいインデックスで存在しない要素があっても、その要素は未定義のままになります。未定義の配列要素は、値を代入するまで使用できません。たとえば次のコードでは、配列を作成してインデックス 4 に要素を作成しています。要素 4 の内容は出力されますが、存在しない要素 3 を出力しようとするとエラーが発生します。

```
 <cfset myarray=ArrayNew(1)>
<cfset myarray[4]=4>
<cfoutput>
    myarray4: #myarray[4]#<br>
    myarray3: #myarray[3]#<br>
</cfoutput>
```

関数を使用した配列要素の追加

次の配列関数を使用して配列にデータを追加できます。

関数	説明
ArrayAppend	配列の末尾に配列要素を作成します。
ArrayPrepend	配列の先頭に配列要素を作成します。
ArrayInsertAt	指定のインデックス位置に配列要素を挿入します。

ColdFusion の配列はダイナミックなので、配列で要素の追加や削除を行うと、それより大きい番号のインデックス値がすべて変更されます。たとえば次のコードは、まず 2 つの要素配列を作成し、配列の内容を表示します。次に、ArrayPrepend を使用して配列の先頭に新しい要素を挿入し、その結果を表示します。インデックス 1 および 2 にあったデータは、インデックス 2 および 3 に移動しています。

```
 <!--- Create an array with three elements. --->
<cfset myarray=ArrayNew(1)>
<cfset myarray[1]="Original First Element">
<cfset myarray[2]="Original Second Element">
<!--- Use cfdump to display the array structure --->
<cfdump var=#myarray#>
<br>
<!--- Add a new element at the beginning of the array. --->
<cfscript>
    ArrayPrepend(myarray, "New First Element");
</cfscript>
<!--- Use cfdump to display the new array structure. --->
<cfdump var=#myarray#>
```

これらの配列関数の詳細については、『CFML リファレンス』を参照してください。

配列からの要素の削除

特定のインデックスにあるデータを配列から削除するには、ArrayDeleteAt 関数を使用します。データ値を 0 に設定したり、空の文字列を設定したりしても、要素は削除されません。配列からデータを削除すると、その配列のサイズは、次の例のようにダイナミックに変化します。

```
 <!--- Create an array with three elements --->
<cfset firstname=ArrayNew(1)>
<cfset firstname[1]="Robert">
    <cfset firstname[2]="Wanda">
    <cfset firstname[3]="Jane">
<!--- Delete the second element from the array --->
<cfset temp=ArrayDeleteAt(firstname, 2)>
<!--- Display the array length (2) and its two entries,
    which are now "Robert" and "Jane" --->
<cfoutput>
    The array now has #ArrayLen(firstname)# indexes<br>
    The first entry is #firstname[1]#<br>
        The second entry is #firstname[2]#<br>
    </cfoutput>
```

ArrayDeleteAt 関数によって、元の 2 番目の要素が削除されます。これによって配列のサイズが変わり、エントリは 2 つになります。現在の 2 番目の要素は、元の 3 番目の要素です。

配列のコピー

単純変数 (数値、文字列、ブール値、および日付時刻値) の配列をコピーするには、新しい変数名に元の配列を代入します。ArrayNew を使用してあらかじめ配列を作成しておく必要はありません。新しい変数に既存の配列を代入すると、ColdFusion によって配列が作成され、元の配列の内容が新しい配列にコピーされます。次の例では、まず配列を作成して 2 つの要素を挿入します。次に元の配列をコピーし、コピーした配列の要素の 1 つを変更してから、両方の配列を表示します。結果に示されるように、元の配列は変更されませんが、コピーには新しい 2 番目の要素が追加されます。

```
 <cfset myArray=ArrayNew(1)>
<cfset myArray[1]="First Array Element">
<cfset myArray[2]="Second Array Element">
<cfset newArray=myArray>
<cfset newArray[2]="New Array Element 2">
<cfdump var=#myArray#><br>
<cfdump var=#newArray#>
```

配列に複合変数 (構造体、クエリーオブジェクト、COM オブジェクトなどの外部オブジェクト) が含まれている場合は、新しい変数に元の配列を代入しても、元の配列の完全なコピーとはなりません。配列構造はコピーされますが、新しい配列には複合データのコピーではなく、そのデータへの参照が含まれています。この動作を説明するために、次のコードを実行します。

```
 Create an array that contains a structure.<br>
<cfset myStruct=StructNew()>
<cfset myStruct.key1="Structure key 1">
<cfset myStruct.key2="Structure key 2">
<cfset myArray=ArrayNew(1)>
<cfset myArray[1]=myStruct>
<cfset myArray[2]="Second array element">
<cfdump var=#myArray#><br>
<br>
Copy the array and dump it.<br>
<cfset myNewArray=myArray>
<cfdump var=#myNewArray#><br>
<br>
Change the values in the new array.<br>
<cfset myNewArray[1].key1="New first array element">
<cfset myNewArray[2]="New second array element">
<br>
Contents of the original array after the changes:<br>
<cfdump var=#myArray#><br>
Contents of the new array after the changes:<br>
<cfdump var=#myNewArray#>
```

新しい配列を変更すると、元の配列の構造体の内容も変更されます。

複合変数が含まれている配列の完全なコピーを作成するには、Duplicate 関数を使用します。

配列データの設定

配列要素には、クエリー、構造体、他の配列など、任意の値を保存できます。配列の値は、代入ステートメントを使用して設定できます。また、ArraySet、ArrayAppend、ArrayInsertAt、ArrayPrepend などの関数を使用して配列のデータを設定することもできます。これらの関数は、既存の配列にデータを追加するときに役立ちます。

特に、次のテクニックを使用すると便利です。

- ArraySet 関数を使用した配列データの設定
- cfloop タグを使用した配列データの設定
- クエリーからの配列データの設定

ArraySet 関数を使用した配列データの設定

ArraySet 関数を使用すれば、1 次元配列または多次元配列の 1 つの次元に、空の文字列や 0 などの初期値を設定できます。この関数は、特定のサイズの配列を用意する必要はあるが、個別のデータを追加する必要はない場合に便利です。たとえば、すべての配列インデックスを参照できるようにしておく場合などに使用します。空の文字列のように、値が含まれていない配列インデックスを参照すると、エラーが発生します。

ArraySet 関数の形式を次に示します。

```
ArraySet (arrayname, startrow, endrow, value)
```

次の例では、配列 myarray のインデックス 1 から 100 を空の文字列で初期化します。

```
ArraySet (myarray, 1, 100, "")
```

cfloop タグを使用した配列データの設定

cfloop タグは、配列にデータを設定するときによく使用される効率的な方法です。次の例では、cfloop タグと MonthAsString 関数を使用して、単純な 1 次元配列に月の名前を設定しています。2 番目の cfloop によって、配列内のデータがブラウザに出力されます。

```
<cfset months=arraynew(1)>

<cfloop index="loopcount" from=1 to=12>
    <cfset months[loopcount]=MonthAsString(loopcount)>
</cfloop>

<cfloop index="loopcount" from=1 to=12>
    <cfoutput>
        #months[loopcount]#<br>
    </cfoutput>
</cfloop>
```

2 次元配列や 3 次元配列でのネストループの使用

2 次元配列や 3 次元配列の値を出力するには、配列データを返すためにネストループを使用します。1 次元配列のデータを出力するには、前の例のように、1 つの cfloop で十分です。多次元配列では、配列の各レベルで別個のループカウンタを設定します。

2 次元配列での cfloop タグのネスト

次の例では、ネストされた cfloop タグを使用して、2 次元配列のデータを出力しています。同様に、ネストされた cfloop タグを使用して配列データを設定しています。

```
 <cfset my2darray=arraynew(2)>
<cfloop index="loopcount" from=1 to=12>
    <cfloop index="loopcount2" from=1 to=2>
        <cfset my2darray[loopcount][loopcount2]=(loopcount * loopcount2)>
    </cfloop>
</cfloop>

<p>The values in my2darray are currently:</p>

<cfloop index="OuterCounter" from="1" to="#ArrayLen(my2darray)#">
    <cfloop index="InnerCounter" from="1"to="#ArrayLen(my2darray[OuterCounter])#">
        <cfoutput>
            <b>[#OuterCounter#][#InnerCounter#]</b>:
            #my2darray[OuterCounter][InnerCounter]#<br>
        </cfoutput>
    </cfloop>
</cfloop>
```

3 次元配列での cfloop タグのネスト

3 次元配列では、cfloop タグをもう 1 回ネストします。この例では、コードを短くするため、配列値の設定は行っていません。

```
 <cfloop index="Dim1" from="1" to="#ArrayLen(my3darray)#">
    <cfloop index="Dim2" from="1" to="#ArrayLen(my3darray[Dim1])#">
        <cfloop index="Dim3" from="1"to="#ArrayLen(my3darray[Dim1][Dim2])#">
            <cfoutput>
                <b>[#Dim1#][#Dim2#][#Dim3#]</b>:
                    #my3darray[Dim1][Dim2][Dim3]#<br>
            </cfoutput>
        </cfloop>
    </cfloop>
</cfloop>
```

クエリーからの配列データの設定

クエリーから配列のデータを設定するときは、次の点に注意してください。

- すべてのクエリーデータを配列に一度に追加することはできません。通常、クエリーから配列データを設定するには、ループ処理が必要です。
- クエリー列のデータは、配列と同じシンタックスを使用して参照できます。たとえば、myquery.col_name[1] とすると、myquery というクエリーの col_name という列の最初の行のデータを参照できます。
- cfloopquery= ループの内側では、クエリーの変数を参照するためにクエリー名を指定する必要はありません。

cfset タグで次のシンタックスを使用すると、配列インデックスの値を定義できます。

```
 <cfset arrayName[index]=queryColumn[row]>
```

次の例では、cfloop タグを使用して、サンプルデータソースの 4 つのデータ列を配列 myarray に設定します。

```
 <!--- Do the query --->
<cfquery name="test" datasource="cfdocexamples">
    SELECT Emp_ID, LastName, FirstName, Email
    FROM Employees
</cfquery>

<!--- Declare the array --->
<cfset myarray=arraynew(2)>

<!--- Populate the array row by row --->
<cfloop query="test">
    <cfset myarray[CurrentRow][1]=Emp_ID>
    <cfset myarray[CurrentRow][2]=LastName>
    <cfset myarray[CurrentRow][3]=FirstName>
    <cfset myarray[CurrentRow][4]=Email>
</cfloop>

<!--- Now, create a loop to output the array contents --->
<cfset total_records=test.recordcount>
<cfloop index="Counter" from=1 to="#Total_Records#">
    <cfoutput>
        ID: #MyArray[Counter][1]#,
        LASTNAME: #MyArray[Counter][2]#,
        FIRSTNAME: #MyArray[Counter][3]#,
        EMAIL: #MyArray[Counter][4]# <br>
    </cfoutput>
</cfloop>
```

この例では、クエリーオブジェクトのビルトイン変数である CurrentRow を使用して、配列の最初の次元のインデックスを指定しています。

配列関数

次の関数を使用して、配列の作成、編集、および操作を行うことができます。

関数	説明
ArrayAppend	指定された配列の末尾に配列要素を付加します。
ArrayAvg	指定された配列の値の平均を返します。
ArrayClear	指定された配列のすべてのデータを削除します。
ArrayDeleteAt	指定された配列の指定されたインデックスの要素を削除し、配列のサイズを変更します。
ArrayInsertAt	指定された配列の指定されたインデックスにデータを持つ要素を挿入し、配列のサイズを変更します。
ArrayIsDefined	指定された配列が定義されている場合は True を返します。
ArrayIsEmpty	指定された配列にデータがない場合は True を返します。
ArrayLen	指定された配列の長さを返します。
ArrayMax	指定された配列内の最大の数値を返します。
ArrayMin	指定された配列内の最小の数値を返します。
ArrayNew	指定された次元を持つ配列を作成します。
ArrayPrepend	指定された配列の先頭に配列要素を追加します。
ArrayResize	指定された最小要素数に配列を設定し直します。
ArraySet	1 次元配列の指定された範囲の要素に、指定された値を設定します。

関数	説明
ArraySort	指定された配列を、数値順またはアルファベット順に要素を並べ替えて返します。
ArraySum	指定された配列内の値の合計を返します。
ArraySwap	指定されたインデックスの配列値を入れ替えます。
ArrayToList	指定された 1 次元配列を、指定された文字で区切られたリストに変換します。
IsArray	値が配列である場合は True を返します。
ListToArray	指定された文字で区切られたリストを配列に変換します。

これらの関数の詳細については、『CFML リファレンス』を参照してください。

配列を返す関数を使用する場合に、関数呼び出しステートメントで特定の要素配列を直接参照できるようになりました。たとえば、次の例では myFunc() 関数から返される配列の 5 番目の要素を参照しています。

```
myFunc()[5]
```

構造体について

ColdFusion の構造体は、キーと値のペアで構成されます。構造体によって、互いに関連する一連の変数を 1 つの名前でグループ化することができます。ColdFusion の構造体はダイナミックに定義できます。

構造体を使用すると、関連する値を個々の値ではなく 1 つのユニットとして参照できます。たとえば、従業員リストを管理する場合は、氏名、住所、電話番号、ID 番号などの個人情報を保持する構造体を作成できます。これらの情報は、個別の変数の集まりではなく、**employee** という構造体として参照できます。

構造体のキーは、文字列であることが必要です。キーに関連付ける値には、有効な ColdFusion 値またはオブジェクトが使用できます。文字列や整数、または配列や別の構造体などの複合オブジェクトを関連付けることができます。構造体にはあらゆる種類のデータを含めることができるので、複雑なデータを柔軟に表現できる強力なツールとして使用できます。

構造体の表記法

ColdFusion では、構造体の内容を参照するための表記法として、次の 3 つの方法がサポートされています。使用する表記法はアプリケーションの必要性に応じて選択します。

表記法	説明
Object.property	obj.prop でオブジェクト obj のプロパティ prop を参照できます。この表記法 (ドット表記法) は、次の例のような単純な代入に便利です。 depts.John="Sales" この表記法は、プロパティ名 (キー) が事前にわかっており、特殊文字、数字、空白を含まない文字列である場合にのみ使用できます。プロパティ (キー) がダイナミックな場合は、ドット表記法を使用できません。
連想配列	キー名が事前にわからない場合や、キー名にスペース、数字、または特殊文字が含まれている場合に使用できます。この表記法では、インデックスが文字列である配列として構造体を記述します。たとえば、次のようになります。 depts["John"]="Sales" depts[employeeName] = "Sales" 連想配列のインデックスには変数 (employeeName など) を使用できます。したがって、リテラルキー名はすべて引用符で囲みます。 変数を使用した連想配列の参照方法については、68 ページの「構造体の参照をダイナミックに構築する」を参照してください。
構造体	構造体表記法は、構造体を作成して初期値を設定する場合にのみ使用します。構造体のデータにアクセスしたりデータを更新したりするときには使用しません。この表記法は、代入式の右辺でのみ使用します。この表記法の形式は次のとおりです。 {keyName=value[,keyName=value]...} 角括弧と省略記号 ([]...) は、繰り返しが可能なオプションの内容であることを表します。 次の例では、構造体表記法を使用して構造体を作成しています。 <cfset name={firstName = "John", lastName = "Smythe"}>

複合構造体の参照

構造体の中に別の構造体が含まれている場合は、object.property 表記法または連想配列表記法を拡張することで、ネストされた構造体のデータを参照できます。両方の表記法を組み合わせて使用することもできます。

たとえば、構造体 structure1 にキー key1 があり、その値が struct2key1、struct2key2... というキーを持つ構造体である場合は、次の参照方法のいずれを使用しても、key1 の構造体の最初のキーのデータにアクセスできます。

```
 Structure1.key1.Struct2key1
Structure1["key1"].Struct2key1
Structure1.key1["Struct2key1"]
Structure1["key1"]["Struct2key1"]
```

次の例に、複合構造体の内容を参照するさまざまな方法を示します。

```
 <cfset myArray=ArrayNew(1)>
<cfset myArray[1]="2">
<cfset myArray[2]="3">
<cfset myStruct2=StructNew()>
<cfset myStruct2.struct2key1="4">
<cfset myStruct2.struct2key2="5">
<cfset myStruct=StructNew()>
<cfset myStruct.key1="1">
<cfset myStruct.key2=myArray>
<cfset myStruct.key3=myStruct2>
<cfdump var=#myStruct#><br>

<cfset key1Var="key1">
<cfset key2Var="key2">
<cfset key3Var="key3">
<cfset var2="2">

<cfoutput>
Value of the first key<br>
#mystruct.key1#<br>
#mystruct["key1"]#<br>
#mystruct[key1Var]#<br>
<br>
Value of the second entry in the key2 array<br>
#myStruct.key2[2]#<br>
#myStruct["key2"][2]#<br>
#myStruct[key2Var][2]#<br>
#myStruct[key2Var][var2]#<br>
<br>
Value of the struct2key2 entry in the key3 structure<br>
#myStruct.key3.struct2key2#<br>
#myStruct["key3"]["struct2key2"]#<br>
#myStruct[key3Var]["struct2key2"]#<br>
#myStruct.key3["struct2key2"]#<br>
#myStruct["key3"].struct2key2#<br>
<br>
</cfoutput>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明

`<cfset myArray=ArrayNew(1)>`
`<cfset myArray[1]="2">`
`<cfset myArray[2]="3">`
`<cfset myStruct2=StructNew()>`
`<cfset myStruct2.struct2key1="4">`
`<cfset myStruct2.struct2key2="5">`
`<cfset myStruct=StructNew()>`
`<cfset myStruct.key1="1">`
`<cfset myStruct.key2=myArray>`
`<cfset myStruct.key3=myStruct2>`
	3 つのエントリ (文字列、配列、および埋め込まれた構造体) を持つ構造体を作成します。

`<cfdump var=#myStruct#> `	構造体全体を表示します。

`<cfset key1Var="key1">`
`<cfset key2Var="key2">`
`<cfset key3Var="key3">`
	myStruct の各キーの名前および数字 2 を値として持つ変数を作成します。

コード	説明
<cfoutput> Value of the first key #mystruct.key1# #mystruct["key1"]# #mystruct[key1Var]# 	次の表記法を使用して、key1 (文字列) エントリの値を出力します。 • object.property 表記法 • 定数を使用した連想配列表記法 • 変数を使用した連想配列表記法
  Value of the second entry in the key2 array #myStruct.key2[2]# #myStruct["key2"][2]# #myStruct[key2Var][2]# #myStruct[key2Var][var2]# 	次の表記法を使用して、key2 (配列) の 2 番目のエントリの値を出力します。 • object.property 表記法 • 定数を使用した連想配列表記法 • 変数を使用した連想配列表記法 • 配列と配列インデックスの両方に変数を使用した連想配列表記法
Value of the struct2key2 entry in the key3 structure #myStruct.key3.struct2key2# #myStruct["key3"]["struct2key2"]# #myStruct[key3Var]["struct2key2"]# #myStruct.key3["struct2key2"]# #myStruct["key3"].struct2key2# 	次の表記法を使用して、key3 (埋め込まれた構造体) の 2 番目のエントリの値を出力します。 • object.property 表記法 • 2 つの定数を使用した連想配列表記法 • 変数と定数を使用した連想配列表記法 • object.property 表記法と連想配列表記法 • 連想配列表記法と object.property 表記法

構造体の作成と使用

以降のサンプルコードでは、**employee** という構造体を使用しています。これは、社内情報システムに新しい従業員を追加するために使用されています。

構造体の作成

ColdFusion での構造体の作成方法には、明示的な方法と暗黙的な方法の 2 通りがあります。明示的な方法では、関数を使用して構造体を作成し、代入ステートメントまたは関数を使用してデータを設定します。暗黙的な方法では、代入ステートメントを使用します。

関数による構造体の作成

次のように、StructNew 関数を使用して、構造体に変数名を割り当てることで構造体を作成できます。

```
<cfset structName = StructNew()>
```

たとえば、次のシンタックスを使用すると、departments 構造体を作成できます。

```
<cfset departments = StructNew()>
```

このステートメントによって空の構造体が作成されるので、データを追加できます。

構造体の暗黙的な作成

空の構造体を暗黙的に作成するには、次の例のようにします。

```
<cfset myStruct = {}>
```

変数にデータを代入することによって構造体を作成できます。たとえば、次の各行では、myStruct という名前の構造体が作成されます。この構造体は 1 つの要素 name を持ち、その値は Adobe Systems Incorporated です。

```
<cfset coInfo.name = "Adobe Systems Incorporated">
<cfset coInfo["name"] = "Adobe Systems Incorporated">
<cfset coInfo = {name = "Adobe Systems Incorporated"}>
```

3 番目の例のように構造体表記法を使用して構造体を作成する場合は、複数の構造体フィールドを作成できます。その例を次に示します。

```
<cfset coInfo={name="Adobe Systems Incorporated", industry="software"}
```

暗黙的な作成を行うときは、構造体をネストしたり、配列をネストしたり、構造体と配列をネストしたりすることはできません。たとえば、次の行はエラーになります。

```
<cfset myStruct = {structKey1 = {innerStructKey1 = "innerStructValue1"}}>
```

同様に、構造体表記法の内部で、代入式の左辺に object.property 表記法を使用することはできません。たとえば、次のステートメントはエラーになります。

```
<cfset myStruct={structKey1.innerStructKey1 = "innerStructValue1"}>
```

代わりに、次のように複数のステートメントを使用します。

```
<cfset innerStruct1 = {innerStructKey1 = "innerStructValue1"}
<cfset myStruct1={structKey1 = innerStruct1}>
```

構造体を暗黙的に作成する場合、ダイナミック変数は使用できません。たとえば、次の式ではエラーが発生します。

```
<cfset i="coInfo">
<cfset "#i#"={name = ""Adobe Systems Incorporated"}>
```

関数およびタグにおける暗黙的に作成された構造体の使用

暗黙的に作成された構造体は、関数 (ユーザー定義関数を含む) やタグの中で直接使用できます。たとえば、次のコードを実行すると、暗黙的に作成された構造体がダンプされます。

```
<cfdump var="#{Name ="28 Weeks Later", Time = "7:45 PM"}#">
```

次の例に示すように、構造体表記法の内部では配列表記法を使用できます。

```
<cfset student = {firstName="Jane", lastName="Janes", grades=[91, 78, 87]}>
```

構造体要素の追加と更新

構造体に要素を追加したり、構造体の要素を更新したりするには、要素に値を代入するか、または ColdFusion 関数を使用します。直接代入するほうがシンプルで効率的です。

構造体にキーと値のペアを追加するには、次の例のように構造体キーの値を定義します。

```
<cfset myNewStructure.key1="A new structure with a new key">
<cfdump var=#myNewStructure#>
<cfset myNewStructure.key2="Now I've added a second key">
<cfdump var=#myNewStructure#>
```

次のコードでは、cfset と object.property 表記法を使用して、departments.John という構造体要素を作成し、John の部門を Sales から Marketing に変更しています。さらに、連想配列表記法を使用して、部門を Facilities に変更しています。部門を変更するたびに、その結果を表示しています。

```
<cfset departments=structnew()>
<cfset departments.John = "Sales">
<cfoutput>
    Before the first change, John was in the #departments.John# Department<br>
</cfoutput>
<cfset Departments.John = "Marketing">
<cfoutput>
    After the first change, John is in the #departments.John# Department<br>
</cfoutput>
<cfset Departments["John"] = "Facilities">
<cfoutput>
    After the second change, John is in the #departments.John# Department<br>
</cfoutput>
```

構造体とキーに関する情報の取得

構造体やキーに関する情報を取得するには、ColdFusion 関数を使用します。

構造体に関する情報の取得

値が構造体であるかどうかを調べるには、次のように IsStruct 関数を使用します。

```
IsStruct(variable)
```

この関数は、変数 **variable** が ColdFusion 構造体である場合に True を返します。**variable** が java.util.Map インターフェイスを実装する Java オブジェクトである場合も、True を返します。

構造体のインデックスは数値ではないので、構造体に存在する名前と値のペアの数を調べるには、次の例のように、StructCount 関数を使用します。

```
StructCount(employee)
```

特定の構造体がデータを含んでいるかどうかを調べるには、次のように、StructIsEmpty 関数を使用します。

```
StructIsEmpty(structure_name)
```

構造体が空の場合は True を返し、データを含んでいる場合は False を返します。

特定のキーとその値の検索

構造体に特定のキーが存在するかどうかを調べるには、次のように、StructKeyExists 関数を使用します。

```
StructKeyExists(structure_name, "key_name")
```

構造体名は引用符で囲みません。キー名は二重引用符 (") で囲みます。たとえば次のコードでは、MyStruct.MyKey が存在する場合にその値を表示します。

```
<cfif StructKeyExists(myStruct, "myKey")>
    <cfoutput> #mystruct.myKey#</cfoutput><br>
</cfif>
```

StructKeyExists 関数で、キー名を変数で表すことによって、キーをダイナミックにテストできます。この場合は、変数を引用符で囲みません。たとえば次のコードでは、GetEmployees クエリーのレコードをループして、クエリーの LastName フィールドに一致するキーが myStruct 構造体に存在するかどうかをテストします。一致するキーが検出された場合は、クエリーの LastName とそれに対応する構造体のエントリが表示されます。

```
<cfloop query="GetEmployees">
<cfif StructKeyExists(myStruct, LastName)>
    <cfoutput>#LastName#: #mystruct[LastName]#</cfoutput><br>
</cfif>
</cfloop>
```

キーの名前が事前にわかっている場合は、次の例のように、ColdFusion の IsDefined 関数を使用できます。

```
IsDefined("structure_name.key")>
```

キーがダイナミックである場合や、キーに特殊文字が含まれている場合は、StructKeyExists 関数を使用します。

注意：構造体エントリが存在するかどうかをテストするときは、IsDefined を使用するよりも、StructKeyExists を使用するほうが効率的です。ColdFusion のスコープは構造体として使用できるので、変数の存在を確認するときに StructKeyExists を使用すれば、効率を向上させることができます。

構造体のキーのリストの取得

CFML 構造体のキーのリストを取得するには、次のように StructKeyList 関数を使用します。

```
<cfset temp=StructKeyList(structure_name, [delimiter])>
```

区切り文字には任意の文字を指定できます。デフォルトはカンマです。

構造体のキーの配列を取得したい場合は、次のように StructKeyArray 関数を使用します。

```
<cfset temp=StructKeyArray(structure_name)>
```

注意：StructKeyList 関数と StructKeyArray 関数では、キーが返される順序に特定の規則はありません。結果をソートするには、ListSort または ArraySort 関数を使用します。

構造体のコピー

ColdFusion には、構造体をコピーしたり、構造体の参照を作成したりする方法がいくつか用意されています。次の表に、それらの方法と用途を示します。

方法	用途
Duplicate 関数	構造体の完全なコピーを作成します。構造体、クエリー、その他のオブジェクトなど、すべてのデータを元の構造体から新しい構造体にコピーします。したがって、構造体のコピーを変更しても、他の構造体には影響を与えません。 この関数は、構造体を新しいスコープに完全に移動する場合に便利です。特に、Application などのロックする必要があるスコープに構造体が作成されている場合は、Request などのロックする必要がないスコープに複製してから、ロックが必要なスコープの構造体を削除できます。
StructCopy 関数	構造体のシャローコピーを作成します。構造体を作成し、元の構造体の最上位にあるすべての単純変数と配列の値を新しい構造体にコピーします。ただし、元の構造体に含まれている構造体、クエリー、その他のオブジェクト、またはこれらのオブジェクトの中のデータはコピーされません。この場合は、元の構造体内のオブジェクトへの参照が新しい構造体に作成されます。したがって、元の構造体でこれらのオブジェクトを変更すると、コピーされた構造体の対応するオブジェクトも変更されます。 ほとんどの場合、この関数の代わりに Duplicate 関数を使用できます。
変数への代入	構造体への参照 (エイリアス) を作成します。ある変数名を使用してデータを変更した場合、別の名前でその構造体を参照している変数でもデータが変更されます。 この方法は、ローカル変数を別のスコープに追加する場合や、元のスコープから変数を削除せずに変数のスコープを変更する場合に便利です。

構造体のコピー、複製、変数代入の違いを、次の例で説明します。

```
 Create a structure<br>
<cfset myNewStructure=StructNew()>
<cfset myNewStructure.key1="1">
<cfset myNewStructure.key2="2">
<cfset myArray=ArrayNew(1)>
<cfset myArray[1]="3">
<cfset myArray[2]="4">
<cfset myNewStructure.key3=myArray>
<cfset myNewStructure2=StructNew()>
<cfset myNewStructure2.Struct2key1="5">
<cfset myNewStructure2.Struct2key2="6">
<cfset myNewStructure.key4=myNewStructure2>
<cfdump var=#myNewStructure#><br>
<br>
A StructCopy copied structure<br>
<cfset CopiedStruct=StructCopy(myNewStructure)>
<cfdump var=#CopiedStruct#><br>
<br>
A Duplicated structure<br>
<cfset dupStruct=Duplicate(myNewStructure)>
<cfdump var=#dupStruct#><br>
<br>
A new reference to a structure<br>
<cfset structRef=myNewStructure>
<cfdump var=#structRef#><br>

<br>
Change a string, array element, and structure value in the StructCopy copy.<br>
<br>
<cfset CopiedStruct.key1="1A">
<cfset CopiedStruct.key3[2]="4A">
<cfset CopiedStruct.key4.Struct2key2="6A">
Original structure<br>
<cfdump var=#myNewStructure#><br>
Copied structure<br>
<cfdump var=#CopiedStruct#><br>
Duplicated structure<br>
<cfdump var=#DupStruct#><br>
Structure reference
<cfdump var=#structRef#><br>
<br>
Change a string, array element, and structure value in the Duplicate.<br>
<br>
<cfset DupStruct.key1="1B">
<cfset DupStruct.key3[2]="4B">
<cfset DupStruct.key4.Struct2key2="6B">
Original structure<br>
<cfdump var=#myNewStructure#><br>
Copied structure<br>
<cfdump var=#CopiedStruct#><br>
Duplicated structure<br>
<cfdump var=#DupStruct#><br>
Structure reference
<cfdump var=#structRef#><br>
<br>
Change a string, array element, and structure value in the reference.<br>
<br>
<cfset structRef.key1="1C">
<cfset structRef.key3[2]="4C">
```

```
<cfset structRef.key4.Struct2key2="6C">
Original structure<br>
<cfdump var=#myNewStructure#><br>
Copied structure<br>
<cfdump var=#CopiedStruct#><br>
Duplicated structure<br>
<cfdump var=#DupStruct#><br>
Structure reference
<cfdump var=#structRef#><br>
<br>
Clear the original structure<br>
<cfset foo=structclear(myNewStructure)>
Original structure:<br>
<cfdump var=#myNewStructure#><br>
Copied structure<br>
<cfdump var=#CopiedStruct#><br>
Duplicated structure<br>
<cfdump var=#DupStruct#><br>
Structure reference:<br>
<cfdump var=#structRef#><br>
```

構造体要素および構造体の削除

構造体からキーとその値を削除するには、次のように、StructDelete 関数を使用します。

```
StructDelete(structure_name, key [, indicateNotExisting ])
```

indicateNotExisting 引数は、指定されたキーが存在しない場合の処理を指定します。この関数は、デフォルトでは常に True を返します。ただし、**indicateNotExisting** 引数に True が指定されている場合は、キーが存在するときは True を返し、キーが存在しないときは False を返します。

StructClear 関数を使用すると、構造体のインスタンスを保持したまま構造体内のすべてのデータを削除できます。

```
StructClear(structure_name)
```

StructCopy 関数を使用してコピーした構造体を StructClear 関数を使用して削除すると、指定された構造体は削除されますが、コピーには影響を与えません。

次の例のように、複数の参照を持つ構造体を StructClear で削除すると、その構造体の内容が削除され、すべての参照が空の構造体をポイントします。

```
<cfset myStruct.Key1="Adobe">
Structure before StructClear<br>
<cfdump var="#myStruct#">
<cfset myCopy=myStruct>
<cfset StructClear(myCopy)>
After Clear:<br>
myStruct: <cfdump var="#myStruct#"><br>
myCopy: <cfdump var="#myCopy#">
```

構造体のループ

次の例のように、構造体をループすることでその内容を出力できます。

```
 <!--- Create a structure and set its contents. --->
<cfset departments=structnew()>

<cfset val=StructInsert(departments, "John", "Sales")>
<cfset val=StructInsert(departments, "Tom", "Finance")>
<cfset val=StructInsert(departments, "Mike", "Education")>

<!--- Build a table to display the contents --->
<cfoutput>
<table cellpadding="2" cellspacing="2">
    <tr>
        <td><b>Employee</b></td>
        <td><b>Department</b></td>
    </tr>
    <!--- Use cfloop to loop through the departments structure.
    The item attribute specifies a name for the structure key. --->
    <cfloop collection=#departments# item="person">
        <tr>
            <td>#person#</td>
            <td>#Departments[person]#</td>
        </tr>
    </cfloop>
</table>
</cfoutput>
```

構造体の例

構造体は、複数の変数を 1 つの名前でグループ化するときに役立ちます。次の例では、構造体を使用してフォームから情報を収集し、その情報を cf_addemployee というカスタムタグに送信します。カスタムタグの作成および使用については、206 ページの「カスタム CFML タグの作成と使用」を参照してください。

サンプルファイル "newemployee.cfm"

次の ColdFusion ページでは、構造体を作成し、それを使用してデータベースにデータを追加しています。このページで呼び出している cf_addemployee というカスタムタグは、"addemployee.cfm" ファイルで定義します。

```
 <html>
<head>
<title>Add New Employees</title>
</head>

<body>
<h1>Add New Employees</h1>
<!--- Action page code for the form at the bottom of this page. --->

<!--- Establish parameters for first time through --->
<cfparam name="Form.firstname" default="">
<cfparam name="Form.lastname" default="">
<cfparam name="Form.email" default="">
<cfparam name="Form.phone" default="">
<cfparam name="Form.department" default="">

<!--- If at least the firstname form field is passed, create
         a structure named employee and add values. --->
<cfif #Form.firstname# eq "">
    <p>Please fill out the form.</p>
<cfelse>
<cfoutput>
     <cfscript>
        employee=StructNew();
```

```
        employee.firstname = Form.firstname;
        employee.lastname = Form.lastname;
        employee.email = Form.email;
        employee.phone = Form.phone;
        employee.department = Form.department;
     </cfscript>

<!--- Display results of creating the structure. --->
     First name is #StructFind(employee, "firstname")#<br>
     Last name is #StructFind(employee, "lastname")#<br>
     EMail is #StructFind(employee, "email")#<br>
     Phone is #StructFind(employee, "phone")#<br>
     Department is #StructFind(employee, "department")#<br>
    </cfoutput>

<!--- Call the custom tag that adds employees. --->
    <cf_addemployee empinfo="#employee#">
</cfif>

<!--- The form for adding the new employee information --->
<hr>
<form action="newemployee.cfm" method="Post">
First Name: 
<input name="firstname" type="text" hspace="30" maxlength="30"><br>
Last Name: 
<input name="lastname" type="text" hspace="30" maxlength="30"><br>
EMail: 
<input name="email" type="text" hspace="30" maxlength="30"><br>
Phone: 
<input name="phone" type="text" hspace="20" maxlength="20"><br>
Department: 
<input name="department" type="text" hspace="30" maxlength="30"><br>

<input type="Submit" value="OK">
</form>
<br>
</body>
</html>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明
<cfparam name="Form.firstname" default=""> <cfparam name="Form.lastname" default=""> <cfparam name="Form.email" default=""> <cfparam name="Form.phone" default=""> <cfparam name="Form.department" default="">	すべてのフォームフィールドにデフォルト値を設定します。これによって、このページが最初に表示された時点でそれらが確実に存在し、テストできるようにしています。
<cfif #Form.firstname# eq ""> Please fill out the form.	firstname フィールドの値をテストします。このフィールドは必須です。このページが最初に表示された時点では、テストは False になります。 Form.firstname 変数にデータが存在しない場合は、フォームへの入力を求めるメッセージを表示します。

<cfelse>
<cfoutput>
<cfscript>
employee=StructNew();
employee.firstname = Form.firstname;
employee.lastname = Form.lastname;
employee.email = Form.email;
employee.phone = Form.phone;
employee.department = Form.department;
</cfscript>
 	
<!--- Display results of creating the structure. --->
First name is #StructFind(employee, "firstname")#
 	
Last name is #StructFind(employee, "lastname")#
 	
EMail is #StructFind(employee, "email")#
 	
Phone is #StructFind(employee, "phone")#
 	
Department is #StructFind(employee, "department")#
 	
</cfoutput>
	Form.firstname にテキストが含まれている場合は、ユーザーからフォームが送信されています。
	CFScript を使用して構造体 employee を作成し、フォームフィールドのデータを代入します。
	次に、この構造体の内容を表示します。

<cf_addemployee empinfo="#employee#"> </cfif>	cf_addemployee カスタムタグを呼び出して、employee 構造体のコピーを empinfo 属性に渡します。 duplicate 関数を使用することで、元の employee 構造体ではなく、その複製をカスタムタグに渡しています。このテクニックを使用しなくてもこの例は正常に動作しますが、呼び出しページの構造体の内容がカスタムタグによって変更されるのを防止できるので、常にこのテクニックを使用することをお勧めします。
<form action="newemployee.cfm" method="Post"> First Name: <input name="firstname" type="text" hspace="30" maxlength="30"> Last Name: <input name="lastname" type="text" hspace="30" maxlength="30"> EMail: <input name="email" type="text" hspace="30" maxlength="30"> Phone: <input name="phone" type="text" hspace="20" maxlength="20"> Department: <input name="department" type="text" hspace="30" maxlength="30"> <input type="Submit" value="OK"> </form>	データフォーム。ユーザーが [OK] をクリックすると、フォームのデータがこの ColdFusion ページに送信されます。

サンプルファイル "addemployee.cfm"

次のファイルは、従業員を追加するカスタムタグの例です。従業員の情報は、employee 構造体 (empinfo 属性) によって渡されます。キーの自動生成機能がサポートされていないデータベースでは、Emp_ID も追加します。

```
<cfif StructIsEmpty(attributes.empinfo)>
    <cfoutput>
        Error. No employee data was passed.<br>
    </cfoutput>
    <cfexit method="ExitTag">
<cfelse>
    <!--- Add the employee --->
    <cfquery name="AddEmployee" datasource="cfdocexamples">
        INSERT INTO Employees
            (FirstName, LastName, Email, Phone, Department)
        VALUES (
        '#attributes.empinfo.firstname#' ,
        '#attributes.empinfo.lastname#' ,
        '#attributes.empinfo.email#' ,
        '#attributes.empinfo.phone#' ,
        '#attributes.empinfo.department#' )
    </cfquery>
</cfif>
<cfoutput>
    <hr>Employee Add Complete
</cfoutput>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明
`<cfif StructIsEmpty(attributes.empinfo)> <cfoutput> Error. No employee data was passed.  </cfoutput> <cfexit method="ExitTag">`	empinfo 属性が指定されずにカスタムタグが呼び出された場合は、エラーメッセージを表示してタグを終了します。
`<cfelse> <!--- Add the employee ---> <cfquery name="AddEmployee" datasource="cfdocexamples"> INSERT INTO Employees (FirstName, LastName, Email, Phone, Department) VALUES ('#attributes.empinfo.firstname#' , '#attributes.empinfo.lastname#' , '#attributes.empinfo.email#' , '#attributes.empinfo.phone#' , '#attributes.empinfo.department#') </cfquery> </cfif>`	empinfo 構造体に渡された従業員データを、cfdocexamples データベースの Employees テーブルに追加します。 StructFind 関数ではなく、構造体エントリへの直接参照を使用します。 データベースで Emp_ID キーの自動生成がサポートされていない場合は、Emp_ID エントリをフォームに追加し、クエリーにも追加します。
`<cfoutput> <hr>Employee Add Complete </cfoutput>`	終了メッセージを表示します。このコードは、cfelse ブロックの内側に置く必要はありません。empinfo 構造体が空の場合は cfexit タグが実行されるので、このコードが実行されることはありません。

構造体関数

ColdFusion アプリケーションでは、次の関数を使用して構造体の作成や管理が行えます。次の表で、各関数の目的を説明します。また、他の方法ではなくその関数を使用すべきかどうかを判断するのに役立つ情報を簡単に示します。

StructDelete 以外の関数では、参照されたキーまたは構造体が存在しないと例外が発生します。

これらの関数の詳細については、『CFML リファレンス』を参照してください。

関数	説明
Duplicate	構造体の完全なコピーを返します。
IsStruct	指定された変数が ColdFusion 構造体であるか、または java.util.Map インターフェイスを実装する Java オブジェクトである場合に True を返します。
StructAppend	ある構造体を別の構造体に追加します。
StructClear	指定された構造体からすべてのデータを削除します。
StructCopy	構造体のシャローコピーを返します。埋め込まれているすべてのオブジェクトは、元の構造体のオブジェクトへの参照になります。ほとんどの場合、この関数は Duplicate 関数に置き換えることができます。
StructCount	指定された構造体内のキーの数を返します。
StructDelete	指定された構造体から指定された項目を削除します。
StructFind	指定された構造体内の指定されたキーに関連付けられている値を返します。この関数は、連想配列表記法を使用して構造体要素にアクセスするのと同じです。
StructFindKey	指定されたキー名を構造体で検索し、検出したキーに関するデータを含んでいる配列を返します。
StructFindValue	文字列や数値などの指定された単純データ値を構造体で検索し、構造体でのその値の位置に関する情報を含んでいる配列を返します。
StructGet	指定されたパスの構造体に含まれているサブ構造体への参照を返します。この関数は、構造体への直接参照を使用するのと同じです。この関数を間違って構造体でない変数に使用すると、その値が空の構造体に置き換えられます。
StructInsert	指定された構造体に、指定されたキーと値のペアを挿入します。この関数は、直接代入ステートメントとは異なり、指定されたキーが構造体に存在すると、デフォルトでエラーが発生します。
StructIsEmpty	指定された構造体がデータを含んでいるかどうかを示します。構造体にデータがない場合は True を返し、データがある場合は False を返します。
StructKeyArray	指定された構造体のキーの配列を返します。
StructKeyExists	指定された構造体に指定されたキーがある場合は True を返します。この関数を IsDefined 関数の代わりに使用して、構造体として使用可能なスコープに変数が存在するかどうかを確認できます。
StructKeyList	指定された構造体のキーのリストを返します。
StructNew	新しい構造体を返します。
StructSort	ソート条件に従ってソートされた構造体のキー名の配列を返します。
StructUpdate	指定されたキーを、指定された値で更新します。この関数は、直接代入ステートメントと異なり、構造体またはキーが存在しないとエラーが発生します。

CFML スクリプト言語による ColdFusion ページの拡張

Adobe ColdFusion には、サーバーサイドのスクリプト言語である CFScript が用意されています。CFScript を使用すると、スクリプトのシンタックスで ColdFusion の機能を利用できます。この JavaScript に似た言語を使用すれば、タグを使用せずに ColdFusion と同じ制御フローを作成できます。また、CFScript を使用してユーザー定義関数を作成することもできます。作成したユーザー定義関数は、ColdFusion 式が記述できる場所であればどこでも使用できます。

CFScript について

CFScript は言語内の言語です。JavaScript に似たスクリプト言語ですが、JavaScript よりも簡単に使用できます。また、JavaScript と異なり、CFScript は ColdFusion サーバーでのみ動作し、クライアントシステムでは動作しません。CFScript コードでは、ColdFusion のすべての関数および式を使用でき、そのコードのスコープで使用できるすべての ColdFusion 変数にアクセスできます。

CFScript を使用すると、ColdFusion のロジックを簡潔かつ効果的に記述できます。CFScript は、次のような目的でよく使用されます。

- 変数設定の簡略化と高速化
- JavaScript に似たコンパクトなフロー制御構造の構築
- ユーザー定義関数の作成

CFScript では、関数や式を直接記述できます。代入式や関数を cfset タグで囲む必要はありません。また、多くの場合、cfset タグよりも CFScript の代入式のほうが処理が速くなります。

CFScript では、多くのプログラマにとって ColdFusion タグよりも馴染みのある、一連の決定構造やフロー制御構造が使用できます。

変数設定だけでなく、その他のオペレーションでも、通常はタグよりも CFScript のほうが若干速くなります。

ユーザー定義関数 (UDF) は CFScript を使用して作成できます (カスタム関数とも呼ばれます)。UDF は、標準の ColdFusion 関数と同じ方法で呼び出すことができます。ColdFusion のビルトインタグにない機能をカスタムタグで作成できるのと同様に、ColdFusion のビルトイン関数にない機能を UDF で作成することができます。UDF は、データ操作ルーチンや数学演算ルーチンを作成する場合などによく使用されます。

CFScript の中で ColdFusion タグを使用することはできませんが、それらに相当する関数や CFScript ステートメントが用意されています。詳しくは、101 ページの「タグに相当する CFScript 機能」を参照してください。

タグと CFScript の比較

CFML タグと CFScript で同じ処理を記述する例を次に示します。それぞれの例では、フォームから送信されたデータを取得して、構造体に格納しています。姓フィールドまたは部門フィールドが入力されていない場合は、メッセージを表示します。

CFML タグを使用する場合

```
<cfif IsDefined("Form.submit")>
    <cfif (Form.lastname NEQ "") AND (Form.department NEQ "")>
        <cfset employee=structnew()>
        <cfset employee.firstname=Form.firstname>
        <cfset employee.lastname=Form.lastname>
        <cfset employee.email=Form.email>
        <cfset employee.phone=Form.phone>
        <cfset employee.department=Form.department>
        <cfoutput>
            Adding #Form.firstname# #Form.lastname#<br>
        </cfoutput>
    <cfelse>
        <cfoutput>
            You must enter a Last Name and Department.<br>
        </cfoutput>
        </cfif>
</cfif>
```

CFScript を使用する場合

```
<cfscript>
    if (IsDefined("Form.submit")) {
        if ((Form.lastname NEQ "") AND (Form.department NEQ "")) {
            employee=StructNew();
            employee.firstname=Form.firstname;
            employee.lastname=Form.lastname;
            employee.email=Form.email;
            employee.phone=Form.phone;
            employee.department=Form.department;
            WriteOutput("Adding #Form.firstname# #Form.lastname# <br>");
            }
        else
            WriteOutput("You must enter a Last Name and Department.<br>");
        }
</cfscript>
```

ColdFusion 9 での言語の機能強化

ColdFusion 9 では、新しい言語構文、タグサポートの強化、新しいキーワード、CFC として実装されるスクリプト関数、新しい操作のサポートなど、言語に関するさまざまな機能が強化されています。

タグに相当する新しい CFScript 機能

相当する CFScript 機能があるタグを次の表に示します。

タグ	相当する CFScript 機能
cfabort	abort
cfcomponent	component
cfcontinue	continue
cfdirectory <Cfdirectory action=list/> のみ	ディレクトリ関数 DirectoryCreate、DirectoryDelete、DirectoryList、および DirectoryRename。
cfdump	writedump
cfexit	exit
cffinally	finally
cfimport	import
cfinclude	include
cfinterface	interface
cflocation	location
cflog	writelog
cfparam	param
cfprocessingdirective	pageencoding
cfproperty	property
cfrethrow	rethrow
cfthread	thread

タグ	相当する CFScript 機能
cfthrow	throw
cftrace	trace
cftransaction	transaction

ColdFusion 9 で追加されたスクリプト関数

ColdFusion 9 で追加されたスクリプト関数を次の表に示します。

関数	相当する ColdFusion タグ
ftp	cfftp
http	cfhttp
mail	cfmail
pdf	cfpdf
query	cfquery
storedproc	cfstoredproc

スクリプト関数の詳細については、『CFML リファレンス』の Script Functions Implemented as CFCs を参照してください。

ColdFusion 9 で追加された予約語

- import
- finally
- local (関数宣言の内部)
- interface
- pageencoding

CFScript でサポートされる機能

タグに相当する CFScript 機能

タグ	相当する CFScript の機能
cfabort	abort
cfbreak	break。CFScript には continue ステートメントもありますが、これに相当する CFML タグはありません。
cfcase	case
cfcatch	catch
cfcomponent	component
cfcontinue	continue

タグ	相当する CFScript の機能
cfcookie	Cookie スコープのメモリ内変数への直接代入。直接代入を使用して、ユーザーのシステムに保存される永続 Cookie を設定することはできません。
cfdefaultcase	default
cfdirectory <Cfdirectory action=list/> のみ	ディレクトリ関数 DirectoryCreate、DirectoryDelete、DirectoryList、および DirectoryRename。
cfdump	writedump
cfelse	else
cfelseif	elseif
cfexit	exit
cffile	ファイル関数 FileDelete、FileSeek、FileSkipBytes、および FileWriteLine。
cffinally	finally
cffunction	function
cfimage	Image functions。
cfif	if
cfimport	import <cfimport path=""> に相当するのは、cfscript 内の import インポートのみです。cfscript では、<cfimport taglib=""> を使用できません。
cfinclude	include
cfinterface	interface
cflocation	location
cflock	lock
cflog	writelog
cfloop	• インデックスを使用した cfloop : for ループ • 条件付き cfloop : while ループおよび do while ループ • 構造体に対する cfloop : for in ループ。クエリー、リスト、またはオブジェクトに相当するものはありません。
cfobject	createobject、new
cfoutput	writeoutput
cfparam	param
cfprocessingdirective	pageencoding
cfproperty	property
cfrethrow	rethrow
cfreturn	return
cfsavecontent	savecontent

タグ	相当する CFScript の機能
cfset	var var x =1; は <cfset var x =1> に相当します。 代入ステートメント x =1; は <cfset x =1> に相当します。 local.x=1; は <cfset var x =1> に相当します。
cfswitch	switch
cfthread	thread
cfthrow	throw
cftrace	trace
cftransaction	transaction
cftry	try

例

次の例では CFScript でクエリー結果をループ処理しています。

```
...
<cfscript>
// Loop through the qGetEmails RecordSet
for (x = 1; x <= qGetEmails.RecordCount; x=x+1) {
     This_id = qGetEmails.Emails_id[x];
     This_Subject = qGetEmails.Subject[x];
     This_RecFrom = qGetEmails.RecFrom[x];
     This_SentTo = qGetEmails.SentTo[x];
     This_dReceived = qGetEmails.dReceived[x];
      This_Body = qGetEmails.Body[x];
... // More code goes here.
}
</cfscript>
```

予約語

ColdFusion 関数の名前と、ColdFusion 式で予約されている語 (NOT、AND、IS など) に加えて、CFScript では次の語が予約されています。これらの語を、スクリプトコードの中で変数または識別子として使用しないでください。

break	do	import	var
case	else	in	while
catch	finally	interface	
try	for	pageencoding	
continue	function	return	
default	if	switch	

スクリプト関数

スクリプト関数のリストについては、101 ページの「ColdFusion 9 で追加されたスクリプト関数」を参照してください。

CFScript 言語

CFScript 言語のシンタックスは他のスクリプト言語と似ており、同じ種類の要素を持っています。

CFScript の識別

<cfscript> タグと </cfscript> タグの間の領域が、CFScript の領域です。cfscript の領域内に他の CFML タグを含めることはできません。次に最小のスクリプトを示します。

```
<cfscript>
a = 2;
</cfscript>
```

変数

CFScript 変数は、数値、文字列、配列、クエリー、オブジェクトなど、任意の ColdFusion 型として使用できます。CFScript コードは、そのスクリプトが含まれているページにあるすべての変数を読み書きできます。これには、セッション変数、アプリケーション変数、サーバー変数など、すべての共有スコープが含まれます。

式と演算子

CFScript では、すべての CFML 式がサポートされています。CFML 式には、演算子 (+、-、EQ など) や、すべての CFML 関数が含まれます。

いくつかの比較演算子は、CFScript でのみ使用でき、CFML タグでは使用できません。それらに対応する CFML 演算子は、CFScript でも使用できます。次の表に、CFScript でのみ使用できる演算子と、それに対応する CFML タグの演算子 (CFScript でも使用できます) を示します。

CFScript の演算子	CFML の演算子	CFScript の演算子	CFML の演算子
==	EQ	!=	NEQ
<	LT	<=	LTE
>	GT	>=	GTE

CFML の式、演算子、および関数の詳細については、58 ページの「式と # 記号の使用」を参照してください。

ステートメント

CFScript では、次のステートメントがサポートされています。

代入	for-in	try-catch
関数呼び出し	while	function (関数定義)
if-else	do-while	var (カスタム関数内のみ)
switch-case-default	break	return (カスタム関数内のみ)
for	continue	

ステートメントには、次のルールが適用されます。

- ステートメントの終わりには、セミコロンを付けます。
- 改行は無視されます。単一のステートメントを複数の行に記述できます。
- 空白は無視されます。たとえば、セミコロンの前にスペース文字があっても問題ありません。
- 複数のステートメントを 1 つの論理的なステートメントユニットにまとめるには、中括弧 ({}) を使用します。
- 特に説明がない限り、ステートメントの本文では任意の ColdFusion 式を使用できます。

注意：function、var、および return ステートメントの詳細については、151 ページの「CFScript での関数の定義」を参照してください。

ステートメントブロック

中括弧 ({}) は、複数の CFScript ステートメントをグループ化し、1 つのユニットまたはステートメントとして扱われるようにします。このシンタックスを使用すると、次のように、条件ステートメントでコードブロックを作成することができます。

```
if(score GT 0)
{
    result = "positive";
    Positives = Positives + 1;
}
```

この例では、score が 0 よりも大きい場合に両方の代入ステートメントが実行されます。このステートメントがコードブロックに入っていない場合は、最初の行のみが実行されます。

中括弧のみを独立した行として記述する必要はありません。たとえば、前述の例では、if ステートメントと同じ行に左括弧が置かれています。左括弧をどこに記述するかはプログラマによってさまざまです。ただし、少なくとも右括弧は独立した行に記述するほうが、コードが読みやすく、コードブロックが識別しやすくなります。

コメント

CFScript では、単一行と複数行の 2 つの形式のコメントが使用できます。

単一行コメントは、2 つのスラッシュ (//) で始まり、同じ行の行末で終わります。次に例を示します。

```
//This is a single-line comment.
//This is a second single-line comment.
```

複数行コメントは、/* マーカーで始まり、*/ マーカーまでコメントと見なされます。次に例を示します。

```
/*This is a multiline comment.
     You do not need to start each line with a comment indicator.
     This line is the last line in the comment. */
```

コメントには、次のルールが適用されます。

- コメントは、行頭から始める必要はありません。アクティブなコードと同じ行に、コメントを続けることができます。たとえば、次の行は有効です。

  ```
  MyVariable = 12; // Set MyVariable to the default value.
  ```

- 複数行コメントが終了した同じ行に、アクティブなコードを続けることもできます。たとえば、よいコーディングスタイルではありませんが、次の行は有効です。

  ```
  End of my long comment */ foo = "bar";
  ```

- 次のように、複数行コメントを単一行に記述してもかまいません。

  ```
  /*This is a single-line comment using multiline format. */
  ```

- コメント行の中に別の /* および */ マーカーをネストすることはできません。
- CFScript で CFML コメント (<!--- および --->) を使用することはできません。

JavaScript との相違点

CFScript と JavaScript は似ていますが、いくつかの重要な相違点があります。JavaScript と異なる CFScript の機能を、次に示します。

- CFScript の ColdFusion 式は、JavaScript 式のスーパーセットでもサブセットでもありません。特に、ColdFusion 式ではビット単位演算子がサポートされていません。また、ColdFusion の MOD 演算子または % 演算子は、JavaScript の % 演算子と動作が異なります。ColdFusion の演算子は、小数部分を無視して整数演算を行います。ColdFusion 式では、JavaScript でサポートされていない EQV、IMP、CONTAINS、および DOES NOT CONTAIN 演算子がサポートされています。

- 変数宣言 (var キーワード) は、ユーザー定義関数およびスレッドでのみ使用します。
- CFScript では大文字と小文字が区別されません。
- すべてのステートメントはセミコロンで終わり、改行は無視されます。
- 代入式はステートメントであり、式ではありません。したがって、代入演算を評価する必要がある状況で代入式を使用することはできません。
- Window や Document などの JavaScript オブジェクトは利用できません。
- CFScript は、ColdFusion サーバーでのみ処理されます。クライアントサイド CFScript は存在しません。

CFScript の制限

CFScript の中で ColdFusion タグを使用することはできませんが、cfoutput などの ColdFusion タグの中に cfscript ブロックを含めることはできます。

CFScript ステートメントの使用

CFScript では、次のタイプのステートメントを使用できます。

- 代入ステートメントおよび関数
- 条件処理ステートメント
- ループステートメント

代入ステートメントおよび関数の使用

CFScript の代入ステートメントは、cfset タグに相当します。このステートメントは次の形式で記述します。

```
lval = expression;
```

eval は、任意の ColdFusion 変数の参照です。次に例を示します。

```
x = "positive";
y = x;
a[3]=5;
structure.member=10;
ArrayCopy=myArray;
```

CFScript では、ColdFusion 関数 (UDF を含む) を直接呼び出せます。たとえば、次の行は有効な CFScript ステートメントです。

```
StructInsert(employee,"lastname",FORM.lastname);
```

条件処理ステートメントの使用

CFScript では、次の条件処理ステートメントが使用できます。

- if および else ステートメント。これは、cfif、cfelseif、および cfelse タグに相当します。
- switch、case、および default ステートメント。これは、cfswitch、cfcase、および cfdefaultcase タグに相当します。

if および else ステートメントの使用

if および else ステートメントのシンタックスを次に示します。

```
if(expr) statement [else statement]
```

最も簡単な if ステートメントの例を示します。

```
if(value EQ 2700)
    message = "You've reached the maximum";
```

簡単な if-else ステートメントの例を示します。

```
if(score GT 1)
    result = "positive";
else
    result = "negative";
```

CFScript には elseif ステートメントはありません。cfelseif タグに相当する処理を行うには、次の例のように、else ステートメントの直後に if ステートメントを記述します。

```
if(score GT 1)
    result = "positive";
else if(score EQ 0)
    result = "zero";
else
    result = "negative";
```

条件処理を記述する部分では、一般的な条件処理ステートメントと同じように、中括弧を使用して複数のステートメントを囲むことができます。

```
if(score GT 1) {
    result = "positive";
    message = "The result was positive.";
    }
else {
    result = "negative";
    message = "The result was negative.";
    }
```

注意：多くの場合、括弧を適切に使用することで、コードの読みやすさを向上させることができます (処理上は不要な括弧であっても)。

switch および case ステートメントの使用

switch ステートメントと、その従属ステートメントである case および default ステートメントのシンタックスを次に示します。

```
switch (expression) {
        case constant: [case constant:]... statement(s) break;
        [case constant: [case constant:]... statement(s) break;]...
        [default: statement(s)] }
```

switch ステートメントを使用するときは、次のルールおよび推奨事項に従ってください。

- 1 つの switch ステートメント内に、ブール値定数と数値定数を混在させることはできません。
- constant は定数である必要があります。変数、関数、または他の式は指定できません。
- 複数の case**constant:** ステートメントを、1 つまたは複数のステートメントの前に並べることができます。この場合は、それらの case のいずれかが true になれば、その後のステートメントが実行されます。これによって、1 つのコードブロックを複数の条件に対応させることができます。
- 同じ定数値を複数の case で指定することはできません。
- case ステートメントブロックのコロンの後のステートメントは、中括弧で囲む必要はありません。switch 式に等しい定数値が見つかると、break ステートメントに達するまで、以降のすべてのステートメントが実行されます。
- case ステートメントの末尾にある break ステートメントは、switch ステートメントを終了させるために使用します。break ステートメントを省略しても、エラーにはなりません。このステートメントが存在しない場合は、次の case ステートメントに記述されているステートメントもすべて実行されます。これは、次の **case** 条件が **false** であっても行われます。このような動作を意図して行うことは少なく、ほとんどの場合は break の記述忘れです。
- 1 つの switch ステートメントブロックで使用できる default ステートメントは、1 つのみです。式の値に等しい case ステートメント定数がない場合は、default ブロックに記述されているステートメントが実行されます。

- default ステートメントは常に必要なわけではありませんが、case ステートメントを使用するときはその最後に default ステートメントを記述するのがよいプログラミングスタイルといえます。case ステートメントの最後に default ステートメントを記述する場合は、default ブロックコードの最後に break ステートメントを置く必要があります。
- default ステートメントは必須ではありませんが、式が取りうる値を case 定数ですべて列挙していない場合は、このステートメントを使用します。

次の switch ステートメントでは、name 変数の値に基づいて処理を行います。

1. name が John または Robert の場合は、male 変数と found 変数の両方を True に設定します。
2. name が Mary の場合は、male 変数を False に、found 変数を True に設定します。
3. その他の場合は、found 変数を False に設定します。

```
switch(name) {
    case "John": case "Robert":
        male=True;
        found=True;
        break;
    case "Mary":
        male=False;
        found=True;
        break;
    default:
        found=False;
} //end switch
```

ループステートメントの使用

CFScript には、CFML タグよりも豊富なループ構文が用意されています。一般的なプログラミング言語やスクリプト言語と同じように、効果的なループ構文を作成できます。CFScript では、次のループ構文を使用できます。

- For
- While
- Do-while
- For-in

また、ループ処理を制御する continue および break ステートメントも使用できます。

for ループの使用

for ループの形式を次に示します。

```
for (initial-expression; test-expression; final-expression) statement
```

initial-expression および **final-expression** は、次のいずれかです。

- 1 つの代入式 (x=5 や loop=loop+1 など)
- 任意の ColdFusion 式 (SetVariable("a",a+1) など)
- 空

test-expression は、次のいずれかです。

- 次のような任意の ColdFusion 式

```
A LT 5
index LE x
status EQ "not found" AND index LT end
```

- 空

注意：test expression は、ループを繰り返す前に再評価されます。ループ内のコードによって test expression の任意の部分を変更すると、ループの繰り返し回数に影響を及ぼす可能性があります。

statement は、セミコロンで終了する単一のステートメントか、または中括弧で囲まれているステートメントブロックです。

ColdFusion では、for ループは次のように処理されます。

1 **initial expression** が評価されます。

2 **test-expression** が評価されます。

3 **test-expression** が False の場合は、ループが終了され、**statement** の後にあるコードが処理されます。

test-expression が true の場合は、

a **statement** (またはステートメントブロック) が実行されます。

b **final-expression** が評価されます。

c 手順 2 に戻ります。

ループ構文の代表的な使い方は、ループを繰り返すたびにインデックス変数を増加させていく方法です (それ以外の使い方もできます)。

次のループサンプルでは、10 要素からなる配列の各要素に、そのインデックス番号を設定します。

```
for(index=1;
    index LTE 10;
    index = index + 1)
    a[index]=index;
```

より複雑な次の例では、2 つの方法を使用しています。

- 中括弧を使用して複数のステートメントをグループ化し、1 つのブロックにしています。
- 空条件ステートメントを使用しています。ループの制御ロジックは、すべてステートメントブロックの中に含まれています。

```
<cfscript>
strings=ArrayNew(1);
ArraySet(strings, 1, 10, "lock");
strings[5]="key";
indx=0;
for( ; ; ) {
    indx=indx+1;
    if(Find("key",strings[indx],1)) {
        WriteOutput("Found key at " & indx & ".<br>");
        break;
        }
    else if (indx IS ArrayLen(strings)) {
        WriteOutput("Exited at " & indx & ".<br>");
        break;
        }
}
</cfscript>
```

この例では、ループを作成するときの重要な注意事項が示されています。それは、ループを確実に終了させる必要があるということです。この例で elseif ステートメントを記述し忘れると、配列に "key" が含まれていなかった場合に、ColdFusion が無限ループに陥ります (システムエラーが発生して停止しない限り)。無限ループを終了させるには、サーバーを停止する必要があります。

また、この例では、インデックスの処理に関する 2 つの注意事項も示されています。1 つは、この形式のループではインデックスを初期化する必要があることです。もう 1 つは、どこでインデックスが増加するかを考慮して処理を行う必要があることです。この場合は、ループの開始時にインデックスが増加するので、最初のループで 1 になるように、インデックスの初期値を 0 にします。

while ループの使用

while ループの形式を次に示します。

```
while (expression) statement
```

while ステートメントは、次のように処理されます。

1. **expression** が評価されます。
2. **expression** が True の場合は、次が実行されます。
   a. **statement** (セミコロンで終了する単一のステートメントか、または中括弧で囲まれているステートメントブロック) が実行されます。
   b. 手順 1 に戻ります。

   expression が False の場合は、次のステートメントに処理が移ります。

   次の例では、while ループを使用して、10 要素からなる配列に 5 の倍数を代入します。

```
a = ArrayNew(1);
loop = 1;
while (loop LE 10) {
    a[loop] = loop * 5;
    loop = loop + 1;
    }
```

他のループと同様に、while の **expression** が False になる時点があることを確認し、インデックスの処理に注意する必要があります。

do-while ループの使用

do-while ループは while ループに似ていますが、ループのステートメントブロックが実行された後にループ条件がテストされる点が異なります。do-while ループの形式を次に示します。

```
do statement while (expression);
```

dowhile ステートメントは、次のように処理されます。

1. **statement** (セミコロンで終了する単一のステートメントか、または中括弧で囲まれているステートメントブロック) が実行されます。
2. **expression** が評価されます。
3. **expression** が True の場合は、手順 1 に戻ります。

expression が False の場合は、次のステートメントに処理が移ります。

次の例では、while ループの例と同様に、10 要素からなる配列に 5 の倍数を代入します。

```
a = ArrayNew(1);
loop = 1;
do {
a[loop] = loop * 5;
loop = loop + 1;
}
while (loop LE 10);
```

配列値を代入した後にループインデックスをインクリメントしているので、loop 変数は 1 で初期化し、この値が 10 以下であることを確認しています。

次の例は、前の 2 つの例と同じ結果になりますが、インデックスをインクリメントしてから配列値を代入しているので、インデックスは 0 で初期化し、終了条件としてインデックスの値が 10 未満であることを確認しています。

```
a = ArrayNew(1);
loop = 0;
do {
    loop = loop + 1;
    a[loop] = loop * 5;
}
while (loop LT 10);
```

クエリー結果をループ処理する例を次に示します。

```
<cfquery ... name="myQuery">
... sql goes here...
</cfquery>
<cfscript>
if (myQuery.RecordCount gt 0) {
     currRow=1;
     do {
       theValue=myQuery.myField[CurrRow];
       currRow=currRow+1;
     } while (currRow LTE myQuery.RecordCount);
}
</cfscript>
```

for-in ループの使用

for-in ループは、ColdFusion 構造体内の各要素をループします。このループの形式を次に示します。

```
for (variable in structure) statement
```

variable は任意の ColdFusion 識別子で、ColdFusion が構造体内をループする際に、構造体のキー名が格納されます。**structure** は、既存の ColdFusion 構造体の名前である必要があります。**statement** は、セミコロンで終了する単一のステートメントか、または中括弧で囲まれているステートメントブロックです。

次の例では、3 つの要素を持つ構造体を作成します。その後、構造体をループして、各キーの名前と値を表示します。この例に含まれている中括弧は必須ではありませんが、長い WriteOutput 関数の内容を識別しやすくするために使用しています。一般的に、ループなどの構造化された制御フローでは、中括弧を使用すると読みやすくなります。

```
myStruct=StructNew();
myStruct.productName="kumquat";
mystruct.quality="fine";
myStruct.quantity=25;
for (keyName in myStruct) {
    WriteOutput("myStruct." & Keyname & " has the value: " &
            myStruct[keyName] &"<br>");
}
```

注意：cfloop タグとは異なり、CFSCript の for-in ループには、クエリーおよびリストをループ処理する機能は組み込まれていません。

continue および break ステートメントの使用

continue および break ステートメントを使用すると、ループ内の処理を制御できます。

- continue ステートメントは、次のループ繰り返しの先頭までスキップするように指示します。
- break ステートメントは、現在のループまたは case ステートメントを終了します。

continue の使用

continue ステートメントは、現在のループ繰り返しを終了し、ループ内の以降のコードをすべてスキップして、次のループ繰り返しの先頭にジャンプします。たとえば、次のコードは 1 つの配列をループし、空の文字列以外の値を表示します。

```
for ( loop=1; loop LE 10; loop = loop+1) {
    if(a[loop] EQ "") continue;
    WriteOutput(loop);
}
```

このコードスニペットをテストするには、10 以上の要素からなり、いくつかの要素に空以外の文字列が含まれている、a という配列を最初に作成する必要があります。

continue ステートメントは、配列や構造体をループするときに、空の文字列などの特定の値を持つ配列要素または構造体メンバーの処理をスキップするのに便利です。

break の使用

break ステートメントは、現在のループまたは case ステートメントを終了します。処理は、次の CFScript ステートメントから続行されます。case のステートメント処理ブロックは、break ステートメントを使用して終了します。また、次の例のように、無限ループを防止するために特定のテストケースで break ステートメントを実行することもできます。このスクリプトは、配列をループし、key という値を含んでいる配列インデックスを出力します。配列の末尾でループを終了させるために、条件テストと break ステートメントを使用しています。

```
strings=ArrayNew(1);
ArraySet(strings, 1, 10, "lock");
strings[5]="key";
strings[9]="key";
indx=0;
for( ; ; ) {
    indx=indx+1;
    if(Find("key",strings[indx],1)) {
        WriteOutput("Found a key at " & indx & ".<br>");
        }
    else if (indx IS ArrayLen(strings)) {
        WriteOutput("Array ends at index " & indx & ".<br>");
        break;
    }
}
```

for-in 構文（配列の場合）

CFScript で for-in 構文を使用して、配列に対するループを実行できます。

例

```
public String foo(array a)
{
    for(var item in a)
    {
      writedump(item);
    }
}
```

for-in 構文（クエリの場合）

配列や構造体に対するループ処理と同様に、for-in 構文を使用すると、CFScript のクエリオブジェクトをループ処理できます。

例

この例では、クエリ結果は変数 arts に格納されており、for-in ループを使用して結果セットをループ処理しています。for-in 構文で使用されている変数 row は、キーとしてクエリ列を含む構造体です。arts.currentrow を使用して現在の行を参照できます。

```
<cfquery name="arts" datasource="cfartgallery">
    select * from art
</cfquery>
<cfscript>
    cols = listToArray(listsort(arts.columnlist, "textnocase"));
    for(row in arts)
    {
        for(col in cols)
            writeoutput(arts.currentrow & " ..." & col & ": " & row[col] & "<br>");
        writeoutput("<hr>");
    }
</cfscript>
```

注意：recordcount や currentrow などのクエリ結果変数にアクセスするには、クエリ名を前に付ける必要があります（例を参照）。

for ループ内の var 宣言

注意：この機能は、ColdFusion 9 アップデート 1 をインストールしてある場合にのみ適用されます。

for-in 構文で var インラインを使用して、構造体と配列の両方の変数をローカルスコープにバインドできます。

例

```
public String foo(struct s)
{
    for(var item in s)
    {
      writedump(item & ": " & s[item]);
    }
    writedump(local);
}
```

配列の例については、112 ページの「for-in 構文（配列の場合）」を参照してください。

for-in 構文における Java 配列のサポート

for-in 構文で、ネイティブな ColdFusion 配列に加えて、Java 配列がサポートされるようになりました。次に例を示します。

```
cfscript>
                a = CreateObject("java","java.util.Arrays").AsList(ToString("CF,10,Zeus").split(","));
                for (var1 in a) {
                                WriteOutput(var1);
                }
</cfscript>
```

CFScript でのコンポーネントと関数の定義

ColdFusion は、CFC を定義するためのシンタックス (インターフェイス、関数、プロパティ、パラメータを含む) を完全に CFScript 内でサポートします。ただし、現時点で CFScript 関数としてサポートされるのは、特定の ColdFusion タグに限定されます。このセクションでは、コンポーネント定義のシンタックスについて説明します。

関数としてのタグについて詳しくは、101 ページの「タグに相当する CFScript 機能」を参照してください。

基本的なシンタックス

コンポーネントを定義するためのシンタックスを次に示します。

```
/**
* ColdFusion treats plain comment text as a hint.
* You can also use the @hint metadata name for hints.
* Set metadata, including, optionally, attributes, (including custom
* attributes) in the last entries in the comment block, as follows:
*@metadataName metadataValue
...
*/
component attributeName="attributeValue" ... {
body contents
}
```

簡単なコンポーネント定義の例を次に示します。

```
/**
* Simple Component.
*/
component {
/**
* Simple function.
*/
public void function foo() {
WriteOutput("Method foo() called<br>");
}
}
```

CFScript 内だけでコンポーネントを定義するときは、ページで cfscript タグを使用する必要はありません。この場合、component キーワードの前に配置できるのは、コメント (メタデータの割り当てを含む) とインポート演算子のみです。Adobe では、この形式を使用することを積極的に推奨しています。コンポーネントプロパティは、次のように指定します。

```
/**
/*@default defaultValue
* @attrib1Name attrib1Value
* ...
*/
property [type]propName;
```

プロパティ名の前に型を指定する場合は、"type" キーワードを使用する必要はなく、特定の型の名前のみを使用します。いずれの形式を使用する場合も、name 属性の値を設定する必要があります。その他のプロパティ属性 (type など) は、すべてオプションです。cfproperty タグと同様、プロパティ演算子は、コンポーネント定義の冒頭にある左中括弧の直後に配置します。

関数を定義するシンタックスは、コンポーネント定義と類似しています。

```
/**
*Comment text, treated as a hint.
*Set metadata, including, optionally, attributes, in the last entries
*in the comment block, as follows:
*@metadataName metadataValue
...
*/
access returnType function functionName(arg1Type arg1Name="defaultValue1"
arg1Attribute="attributeValue...,arg2Type
arg2Name="defaultValue2" arg2Attribute="attributeValue...,...)
functionAttributeName="attributeValue" ... {
body contents
}
```

関数の定義では、関数の引数 (引数の型、デフォルト値、属性など) をすべて指定します。

関数の定義の例を次に示します。

```
/**
* @hint "This function displays its name and parameter."
*/
public void function foo(String n1=10)
description="does nothing" hint="overrides hint" (
WriteOutput("Method foo() called<br> Parameter value is " & n1);
}
```

required キーワードを指定した場合、その引数は必須になります。required キーワードを指定しない場合、その引数はオプションになります。

次に例を示します。

```
public function funcname(required string argument1)
```

インターフェイスの定義は、コンポーネントの定義と同じパターンに従い、cfinterface タグによって定義されるインターフェイスと同じ一般ルールと制約が適用されます。次の簡単なコードでは、文字列の引数を 1 つ取り、引数のデフォルト値が "Hello World!" に設定された関数を 1 つだけ含むインターフェイスを定義しています。

```
interface {
function method1(string arg1="Hello World!");
function method2 (string arg1="Goodbye World!");
...
}
```

関数を 1 つだけ含む簡単なコンポーネント定義の例を次に示します。

```
/**
* Component defined in CFScript
* @output true
*/
component extends="component_01" {
/**
* Function that displays its arguments and returns a string.
* @returnType string
*/
public function method_03(argOne,argTwo) {
WriteOutput("#arguments.argOne# ");
WriteOutput("#arguments.argTwo# ");
return("Arguments written.");
}
}
```

属性の設定

定義のシンタックスでは、次の 2 つの方法で属性を設定できます。

- 要素の直前にあるコメントの末尾で、次のような形式を使用します。

  ```
  /**
  *Comment
  *@attributeName1 attributeValue
  *@attributeName2 attributeValue
  *...
  */
  ```

- 標準的な属性 / 値の割り当て表記を使用する要素宣言で、次のような形式を使用します。

  ```
  component extends="component_01"
  ```

要素宣言で設定された属性値は、コメントセクションで設定された値より優先されます。したがって、両方の場所で hint などの属性を設定した場合、コメントセクションの値は無視され、要素宣言の値のみが使用されます。

ページエンコードの指定

コンポーネントの文字エンコードを指定するには、コンポーネント本文の冒頭で pageencoding 処理ディレクティブを指定します。pageencoding ディレクティブは、コンポーネントに対してのみ指定できます。次のコード例は、ディレクティブの使用方法を示しています。

```
// this is a component
/**
*@hint "this is a hint for component"
*/
component displayname="My Component" {
pageencoding "Cp1252" ;
//
// The rest of the component definition goes here.
//
}
```

注意：現時点では、CFScript を使用して suppresswhitespace 処理ディレクティブを指定することはできません。

コンポーネントのメタデータへのアクセス

コンポーネント、関数、パラメータ、またはプロパティのメタデータにアクセスするには、GetMetadata 関数を使用します。コンポーネントのメタデータには、そのコンポーネントのプロパティと関数 (属性た関数パラメータなど) が含まれます。

メタデータの構造と内容の詳細については、『CFML リファレンス』の GetMetaData を参照してください。

次のコードは、コンポーネントメタデータの使用方法を示しています。

```
//Create an instance of a component.
theComponent=createObject("Component" "myComponent");
// Get the component metadata.
theMetadata = getMetadata(theComponent);
// The component properties are in an array. Display the name
// of the first property in the array.
writeoutput("Property name: " & theMetadata.properties[1].name);
```

カスタムメタデータの作成のサポート

注意：この機能を使用するには、ColdFusion 9 アップデート 1 をインストールする必要があります。

次のいずれかの方法で、スクリプトのシンタックスでカスタムメタデータを関数の引数に指定できます。

- 引数で、キーと値のペアのスペース区切りのリストを使用します。
- 注釈内で、@arg1.custommetadata "custom value" を使用します。

例

custom.cfm

```
cfscript>
                writeoutput(new custom().foo(10));
</cfscript>
```

custom.cfc

```
/**
 * custom metadata for a cfc defined using annotation as well as key-value pairs
 * @cfcMetadata1 "cfc metadata1"
 */
component cfcMetadata2 = "cfc metadata2"
{
                /**
                 * custom metadata for a property defined using annotation as well as key-value pairs
                 * @propMetadata1 "property metadata1"
                 */
                property type="numeric" name="age" default="10" propMetadata2="property metadata2";

                /**
                 * custom metadata for a function/argument using both annotation and key-value pairs
                 * @arg1.argmdata1 "arg metadata1"
                 * output true
                 * @fnMetadata1 "function metadata1"
                 */
                public string function foo(required numeric arg1=20 argmdata2="arg metadata2")
fnMetadata2="function metadata2"
                {
                                writedump(getmetadata(this));
                                return arg1;
                }
}
```

明示的な関数ローカルスコープ

ColdFusion には、明示的な関数ローカルスコープがあります。このスコープの変数は、関数の実行時にのみ存在し、その関数内でのみ使用できます。関数ローカルスコープ変数を宣言するには、変数の代入時にスコープ名として Local を指定するか、var キーワードを使用します。また、var キーワードは、関数定義の冒頭だけでなく、どの場所でも使用できるようになりました。

注意：現在、local はスコープ名になっているので、変数名または引数名として使用しないでください。local を変数名または引数名として使用した場合、その変数または引数は無視されます。

次のコードは、関数ローカルスコープの使用方法を示しています。

```
<cffunction name="foo" output="true">
<cfset var x = 5>
<cfset local.y=local.x*4>
<cfset var z=[local.x,local.y]>
<cfset local.u.v={z="2"}>
<cfset zz="in Variables Scope">
<cfdump var="#local#">
</cffunction>;
```

関数ローカルスコープの詳細については、34 ページの「ColdFusion 変数の使用」を参照してください。

システムレベル関数の使用

<cfscript> モードで、次の基本的な言語構文を使用できるようになりました。

- throw
- writedump
- writelog
- location
- trace

これらの関数を呼び出すには、name="value" のペアまたは位置表記法を使用して引数を渡します。位置表記法の場合は、引数の順序に従う必要があります。各構文の引数の順序については、『CFML リファレンス』を参照してください。

引数を name=value のペアとして渡す例を次に示します。

```
<cfscript>
    writedump(var=myquery, label="query", show="name", format = "text");
</cfscript>
```

引数を位置引数として渡す例を次に示します。

```
<cfscript>
    writedump(myquery, false, html, name);
</cfscript>
```

位置引数を使用するときは、パラメータをすべて指定する必要はありません。たとえば、1 番目と 3 番目の引数を指定する場合は、2 番目の引数として空の値を追加できます。ただし、これは 2 番目の引数がブール型である場合には当てはまりません。この場合は、true または false を指定する必要があります。

注意：関数の呼び出し時に、位置引数と名前付き引数を同時に使用することはできません。たとえば、writedump 構文で var 属性と output 属性のみを使用する必要がある場合は、writedump(myquery,html) を使用します。

cfimport を使用した import 操作と new 操作

CFScript では、import 操作と new 操作がサポートされています。現在、"New" はキーワードになっています。ただし、これは予約語ではないため、変数名として使用できます。

CFC をインポートするには、cfimport タグまたは import スクリプト演算子を使用します。import 操作では、指定したコンポーネントの内容が現在の名前空間に格納され、変換されたコンポーネントパスがメモリにキャッシュされます。import アクションは、現在のページに対してのみ有効です。CFC を Application.cfm にインポートする場合、CFC は、アプリケーションのその他のページにはインポートされません。

インポートしたコンポーネントは、ドット区切りのパス名を使用せずに直接参照します。キャッシュされたコンポーネントの実行時間は、インポートしていない CFC を使用する場合より高速になります。

注意：cfobject タグ、cfobject タグ、および CreateObject 関数を使用する場合も、変換されたコンポーネントパスがキャッシュされます。ただし、これらのタグや関数では、イニシャライザ関数は呼び出されません。

スクリプトで import ステートメントを使用するには、次のシンタックスを使用します。

```
import "cfc_filepath"
```

ほとんどのパスは引用符で囲まなくても問題ありません。パスを引用符で囲む必要があるのは、ディレクトリ名または CFC 名にハイフンが含まれている場合のみです。

cfimport タグで CFC のインポートがサポートされるようになり、インポートする CF ファイルのパスを path 属性で指定するようになりました。

import 関数、または path 属性を含む cfimport タグは、ページの冒頭でのみ使用します。これらを別の場所で使用した場合は、ページの冒頭に配置した場合と同じ結果になります。したがって、標準的なコーディングスタイルでは、import タグまたは import 演算子をファイルの冒頭に配置します。cfimport タグは、cfcomponent タグの前に配置できます。CFScript の import ステートメントは、コンポーネントステートメントの後に配置する必要があります。

ColdFusion Administrator の [サーバーの設定]-[キャッシュ機能] ページに、コンポーネントキャッシュのオプションとコンポーネントキャッシュをクリアするボタンが追加されました。変換されたコンポーネントパスが ColdFusion でキャッシュされないようにするには、コンポーネントキャッシュのオプションをオフにします。変換されたコンポーネントパスをキャッシュからすべて削除するには、[コンポーネントキャッシュを直ちにクリア] ボタンをクリックします。

注意：いずれの場合も、CFC の com.adobe.coldfusion.* 名前空間は自動的にインポートされます。このパスを明示的にインポートする必要はありません。

new 演算子は、CFC のインスタンスを作成します。この演算子は、cfobject タグおよび CreateObject 関数と同等です。new は、CFScript 演算子として使用することも、CFScript ブロック外部の代入ステートメント (cfset タグなど) で使用することもできます。ColdFusion には、これに対応する cfnew タグがありません。

new 操作のシンタックスを次に示します。

```
cfObject=new cfcPath(constructorParam1,...)
```

または

```
cfObject=new cfcPath(arg1=constructorParam1Value,...)
```

フォルダ名または CFC 名にハイフンが含まれている場合は、次のシンタックスを使用します。

```
cfObject=new "cfc-path"(constructorParam1,...)
```

import 演算子を使用して CFC を含むディレクトリをインポートする場合、cfcPath の値は、CFC ファイル名になります。コンストラクタパラメータは、位置形式にすることも、name="value" の形式にすることもできます。new 演算子を使用すると、ColdFusion では次の処理が実行されます。

1. CFC の initmethod コンストラクタメソッドが検索されます。見つかった場合は、コンポーネントがインスタンス化され、initmethod が実行されます。
2. initmethod コンストラクタメソッドが見つからない場合は、init コンストラクタメソッドが検索されます。見つかった場合は、コンポーネントがインスタンス化され、initmethod が実行されます。
3. いずれのメソッドも存在しない場合は、new 操作によってコンポーネントがインスタンス化されますが、コンストラクタは呼び出されません。

注意：initmethod または init 関数を自動的に呼び出すのは new 演算子のみです。new 演算子は init または initmethod から返された値を返し、戻り値が void である場合は、CFC のインスタンスを返します。cfobject タグおよび CreateObject 関数はこれらの関数を呼び出さないので、カスタムの初期化コードを使用する場合は、そのコードを明示的に呼び出す必要があります。

例外処理

ColdFusion には、CFScript 内で例外処理を行うために try と catch という 2 つのステートメントが用意されています。これらのステートメントは、CFML の cftry タグと cfcatch タグに相当します。

注意：ColdFusion における例外処理については、275 ページの「エラー処理」を参照してください。

例外処理のシンタックスとルール

CFScript の例外処理コードの形式を次に示します。

```
try {
    Code where exceptions will be caught
}
catch(exceptionType exceptionVariable) {
    Code to handle exceptions of type exceptionType
    that occur in the try block
}
...
catch(exceptionTypeN exceptionVariableN) {
    Code to handle exceptions of type
    exceptionTypeN that occur in the try block
}
finally {
    Code that will execute whether there is an exception or not.
}
```

注意：CFScript では、catch および finally ステートメントは try ブロックの後に配置します。try ブロックの中には含めないでください。この構造は、cfcatch および ***cffinally*** タグを本文に含める必要がある cftry タグと異なります。

try ステートメントを使用する場合は、catch ステートメントも使用する必要があります。catch ブロックでは、**exceptionVariable** 変数に例外のタイプが渡されます。この変数は、cfcatch タグの cfcatch.Type ビルトイン変数に相当します。

finally ブロックはオプションです。このブロックに含まれるコードは常に実行され、try ブロックと catch ブロックに含まれるコードの後に実行されます。

例外処理の例

次のコードは、CFScript での例外処理を示しています。このコードでは、CreateObject 関数を使用して Java オブジェクトを作成しています。catch ステートメントは、CreateObject 関数で例外が発生した場合にのみ実行されます。表示される情報には、例外メッセージが含まれています。この except.Message 変数は、渡された Java 例外オブジェクトで Java の getMessage メソッドを呼び出すことに相当します。finally ブロックのメッセージは、catch ブロックのメッセージの後に表示されます。

```
<cfscript>
    try {
        emp = CreateObject("Java", "Employees");
    }
    catch(any excpt) {
        WriteOutput("The application was unable to perform a required operation.<br>
    Please try again later.<br>If this problem persists, contact
    Customer Service and include the following information:<br>
                #excpt.Message#<br>");
    }
    finally {
    writeoutput("<br>Thank you for visiting our web site.<br>come back soon!");
    }
</cfscript>
```

CFScript の例

次の例では、CFScript の次の機能を使用しています。

- 変数への代入
- 関数呼び出し
- For ループ
- If-else ステートメント
- WriteOutput 関数
- Switch ステートメント

この例では、cfscript 以外の ColdFusion タグをまったく使用せずに、CFScript のみを使用しています。ここでは、コースの受験者の構造体を作成します。この構造体には、合格した学生と不合格の学生の 2 つの配列が含まれています。また、不合格の学生の一部 (全員ではない) について、その不合格理由を保持する構造体も作成します。そして、合格した受験者を表示した後に、不合格の学生とその不合格理由を表示します。

```
<html>
<head>
    <title>CFScript Example</title>
</head>
<body>
<cfscript>

    //Set the variables

    acceptedApplicants[1] = "Cora Cardozo";
    acceptedApplicants[2] = "Betty Bethone";
    acceptedApplicants[3] = "Albert Albertson";
    rejectedApplicants[1] = "Erma Erp";
    rejectedApplicants[2] = "David Dalhousie";
    rejectedApplicants[3] = "Franny Farkle";
    applicants.accepted=acceptedApplicants;
    applicants.rejected=rejectedApplicants;

    rejectCode=StructNew();
    rejectCode["David Dalhousie"] = "score";
    rejectCode["Franny Farkle"] = "too late";

    //Sort and display accepted applicants

    ArraySort(applicants.accepted,"text","asc");
    WriteOutput("The following applicants were accepted:<hr>");
    for (j=1;j lte ArrayLen(applicants.accepted);j=j+1) {
        WriteOutput(applicants.accepted[j] & "<br>");
    }
    WriteOutput("<br>");

    //sort and display rejected applicants with reasons information

    ArraySort(applicants.rejected,"text","asc");
    WriteOutput("The following applicants were rejected:<hr>");
    for (j=1;j lte ArrayLen(applicants.rejected);j=j+1) {
        applicant=applicants.rejected[j];
        WriteOutput(applicant & "<br>");
        if (StructKeyExists(rejectCode,applicant)) {
            switch(rejectCode[applicant]) {
                case "score":
                    WriteOutput("Reject reason: Score was too low.<br>");
                    break;
                case "late":
                    WriteOutput("Reject reason: Application was late.<br>");
                    break;
                default:
                    WriteOutput("Rejected with invalid reason code.<br>");
            } //end switch
        } //end if
        else {
            WriteOutput("Reject reason was not defined.<br>");
        } //end else
        WriteOutput("<br>");
    } //end for
</cfscript>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明

<cfscript>
//Set the variables
acceptedApplicants[1] = "Cora Cardozo";
acceptedApplicants[2] = "Betty Bethone";
acceptedApplicants[3] = "Albert Albertson";
rejectedApplicants[1] = "Erma Erp";
rejectedApplicants[2] = "David Dalhousie";
rejectedApplicants[3] = "Franny Farkle";
applicants.accepted=acceptedApplicants;
applicants.rejected=rejectedApplicants;
 	
rejectCode=StructNew();
rejectCode["David Dalhousie"] = "score";
rejectCode["Franny Farkle"] = "too late";
	合格した受験者と不合格の受験者の 2 つの 1 次元配列を作成します。各配列のエントリは順不同になっています。
	構造体を作成し、構造体の要素に各配列を代入します。
	不合格の受験者の不合格コードを保持する構造体を作成します。rejectedCode 構造体には、すべての不合格者は含まれていません。また、あるエントリで規定外の不合格コードを代入しています。この構造体のキーにはスペースが含まれているので、連想配列表記法を使用して構造体の要素を参照しています。

ArraySort(applicants.accepted,"text","asc");
WriteOutput("The following applicants were accepted:<hr>");
for (j=1;j lte ArrayLen(applicants.accepted);j=j+1) {
WriteOutput(applicants.accepted[j] & "
");
}
WriteOutput("
");
	合格した受験者をアルファベット順にソートします。
	見出しを表示します。
	合格した受験者をループし、その名前を出力します。単一ステートメントのループなので中括弧は不要ですが、読みやすくするために付けています。
	合格した受験者リストの末尾に改行を付け加えます。

ArraySort(applicants.rejected,"text","asc"); WriteOutput("The following applicants were rejected:<hr>");	rejectedApplicants 配列をアルファベット順にソートし、見出しを出力します。

for (j=1;j lte ArrayLen(applicants.rejected);j=j+1) {
applicant=applicants.rejected[j];
WriteOutput(applicant & "
");
	不合格の受験者をループします。
	applicant 変数に受験者名を代入します。これによって、コードが読みやすくなり、このブロックの後にある rejectCode 配列で簡単に参照できるようになります。
	受験者名を出力します。

if (StructKeyExists(rejectCode,applicant)) {
switch(rejectCode[applicant]) {
case "score":
WriteOutput("Reject reason: Score was too low.
");
break;
case "late":
WriteOutput("Reject reason: Application was late.
");
break;
default:
WriteOutput("Rejected with invalid reason code.
");
} //end switch
} //end if
	rejectCode 構造体をチェックして、その受験者に不合格コードが設定されているか確認します。
	コードが設定されている場合は、switch ステートメントで不合格コードの値を調べます。
	規定されたコードのいずれかに不合格コードの値が一致する場合は、不合格理由の説明を表示します。それ以外の場合 (default の場合) は、不合格コードが有効でないことを示すメッセージを表示します。
	ブロック末尾のコメントは、制御フローを明確にするためのものです。

else {
WriteOutput("Reject reason was not defined.
");
} //end else
	その受験者のエントリが rejectCode 構造体の中にない場合は、理由が定義されていないことを示すメッセージを表示します。

WriteOutput(" "); } //end for </cfscript>	それぞれの不合格者の後に、空白行を表示します。 不合格者を処理するループを終了します。 CFScript を終了します。

クロージャの使用

クロージャは内部関数です。内部関数は、外部関数の変数にアクセスできます。外部関数にアクセスすることで、内部関数にアクセスできます。次の例を参照してください。

```
<cfscript>
function helloTranslator(String helloWord)
    {
        return function(String name)
            {
                return "#helloWord#, #name#";
            };
    }
helloInHindi=helloTranslator("Namaste");
helloInFrench=helloTranslator("Bonjour");
writeoutput(helloInHindi("Anna"));
//closure is formed.
//Prints Namaste, Anna.
writeoutput("<br>");
writeoutput(helloInFrench("John"));
//Prints Bonjour, John.
</cfscript>
```

この例では、外部関数はクロージャを返します。外部関数には helloHindi 変数を使用してアクセスします。この関数は helloWord 引数を設定します。この関数ポインターを使用して、クロージャを呼び出します。例えば、helloInHindi("Anna") のようにします。外部関数の実行後であっても、クロージャは外部関数によって変数セットにアクセスできることに注意してください。

この場合、クロージャを使用して 2 つの新しい関数が作成されます。1 つは名前に Namaste を追加します。もう 1 つは名前に Bonjour を追加します。helloInHindi および helloInFrench はクロージャです。これらの関数の本文は同じですが、異なる環境を格納します。

内部関数は、外部関数が返った後でも実行できます。内部関数が実行可能な場合に、クロージャが形成されます。

例で示されているように、外部関数から返った後でも、内部関数は外部関数の変数にアクセスできます。クロージャは、作成時の環境への参照を保持しています。例えば、外部関数のローカル変数の値などです。これにより、クロージャは簡単に使用できる便利な機能になります。

クロージャについて詳しくは、http://jibbering.com/faq/notes/closures を参照してください。

ColdFusion でのクロージャ

クロージャは次のようなカテゴリにできます。

- 名前を指定しないでインラインで定義。次のようにして使用できます。
    - 変数、配列項目、構造体および変数スコープに代入できます。関数から直接返すことができます。

      例

```
<cfscript>
function operation(string operator)
    {
        return function(numeric x, numeric y)
        {
            if(operator eq "add")
                {
                return x + y;
                }
            else if(operator eq "subtract")
                {
                return x - y;
                }
        };
    }
myval_addition=operation("add");
myval_substraction=operation("subtract");
writeoutput(myval_addition(10,20));
writeoutput("<br>");
writeoutput(myval_substraction(10,20));
</cfscript>
```

この例では、外部関数は演算子を設定します。myval_addition および myval_substraction はクロージャです。これらは、外部関数によって設定された条件に基づいてデータを処理します。

- 関数およびタグ引数としてインラインで定義。

  例

```
<cfscript>
function operation(numeric x, numeric y, function logic)
    {
        var result=logic(x,y);
        return result;
    }
add = operation(10,20, function(numeric N1, numeric N2)
    {
        return N1+N2;
    });
subtract = operation(10,20, function(numeric N1, numeric N2)
    {
        return N1-N2;
    });
</cfscript>
<cfdump var="#add#">
<cfdump var="#substract#">
```

この例では、関数 operation の引数 logic がクロージャです。operation を呼び出すとき、インラインクロージャが引数として渡されます。この匿名クロージャには、数値を処理するためのロジックが含まれます（addition または subtraction）。この場合、ロジックは動的であり、クロージャとして関数に渡されます。

クロージャは変数に代入可能

変数にクロージャを代入できます。

例

```
var c2 = function () {..}
```

注意：変数にクロージャを代入する場合は、スクリプト形式のシンタックスのみがサポートされます。

クロージャは戻り値の型として使用可能

クロージャを戻り値の型として使用できます。

注意：戻り値の型がクロージャの場合は、Function キーワードの先頭を大文字にすることをお勧めします。

例

```
Function function exampleClosure(arg1)
{
    function exampleReturned(innerArg)
    {
        return innerArg + owner.arg1;
    }
    /*
    return a reference to the inner function defined.
    */
return exampleReturned;
}
```

キーと値のペアによるクロージャの呼び出し

関数呼び出しの場合と同じように、キーと値のペアを渡してクロージャを呼び出すことができます。

例

```
var c2 = function(arg1, arg1) {..}
c2(arg1=1, arg2=3);
```

関数の外でクロージャを変数に代入可能

関数の外で変数にクロージャを代入できます。

例

```
hello = function (arg1)
{
    writeoutput("Hello " & arg1);
};
hello("Mark");
```

引数のコレクションによるクロージャの呼び出し

例

```
var c2 = function(arg1, arg1) {..}
argsColl = structNew();
argsColl.arg1= 1;
argsColl.arg2= 3;
c2(argumentCollection = argsColl);
```

クロージャと関数

クロージャは、作成時に認識できるように変数のコピーを保持します。グローバル変数（ColdFusion 固有スコープなど）およびローカル変数（宣言または外部関数のローカルおよび引数スコープを含む）は、クロージャの作成時に保持されます。関数は静的です。

次の表では、定義される方法に基づくクロージャのスコープの詳細を示します。

クロージャが定義される状況	スコープ
CFC 関数内	クロージャ引数スコープ、外側の関数のローカルスコープおよび引数スコープ、this スコープ、変数スコープ、スーパースコープ
CFM 関数内	クロージャ引数スコープ、外側の関数のローカルスコープおよび引数スコープ、this スコープ、変数スコープ、スーパースコープ
関数引数として	クロージャ引数スコープ、変数スコープ、this スコープ、スーパースコープ（CFC コンポーネント内で定義されている場合）

クロージャでは、スコープが指定されていない変数の検索順序は次のとおりです。

1 クロージャのローカルスコープ

2 クロージャの引数スコープ

3 可能な場合は、外部関数のローカルスコープ

4 可能な場合は、所有者関数のローカルスコープ

5 ColdFusion のビルトインスコープ

注意：クロージャはユーザー定義関数を呼び出せません。これは、クロージャのコンテキストは保持されますが、関数のコンテキストは保持されないためです。これにより、誤った結果になります。例えば、クロージャがキャッシュされた場合、クロージャは後で適切に呼び出して使用できますが、関数は使用できません。

クロージャ関数

クロージャ関数は次のとおりです。

isClosure

説明

クロージャの名前かどうかを調べます。

戻り値

名前をクロージャとして呼び出せる場合は True、呼び出せない場合は False。

カテゴリ

決定関数

シンタックス

isClosure(closureName)

関連項目

他の決定関数。

履歴

ColdFusion 10 Beta：この関数が追加されました。

パラメータ

パラメータ	説明
closureName	クロージャの名前です。引用符で囲まないでください。 定義されている変数名または関数名ではない場合は、エラーになります。

使用方法

クロージャの名前かどうかを判定するには、この関数を使用します。

例

```
<cfscript>
    isClosure(closureName)
    {
     // do something
    }
    else
    {
     // do something
    }
</cfscript>
```

関数 isCustomFunction に対する変更

クロージャは関数オブジェクトですが、カスタム関数とは見なされません。

関数は次の値を返すようになっています。

- True：名前をカスタム関数として呼び出せる場合。
- False：名前をクロージャとして呼び出せる場合。

使用例

次の例では、ColdFusion のクロージャを効果的に使用できる方法を説明します。

例 - クロージャを使用した配列のフィルター処理

次の例では、所在地、年齢、呼称に基づいて従業員をフィルター処理します。フィルター処理には単一の関数を使用します。フィルター処理のロジックは、クロージャとして関数に提供されます。そのフィルター処理ロジックは動的に変更されます。

例

1 変数を定義する employee.cfc ファイルを作成します。

```
/**
* @name employee
* @displayname ColdFusion Closure Example
* @output false
* @accessors true
*/
component
{
property string Name;
property numeric Age;
property string designation;
property string location;
property string status;
}
```

2 従業員の配列を作成します。この CFC には filterArray() 関数も含まれます。クロージャ filter は、関数の引数です。この関数にアクセスするときに、フィルター処理のロジックをクロージャとして渡します。

```
<!---filter.cfc--->
<cfcomponent>
<cfscript>
//Filter the array based on the logic provided by the closure.
function filterArray(Array a, function filter)
    {
        resultarray = arraynew(1);
            for(i=1;i<=ArrayLen(a);i++)
            {
                if(filter(a[i]))
                ArrayAppend(resultarray,a[i]);
            }
        return resultarray;
    }
function getEmployee()
    {
//Create the employee array.
    empArray = Arraynew(1);
    ArrayAppend(empArray,new employee(Name="Ryan", Age=24, designation="Manager", location="US"));
    ArrayAppend(empArray,new employee(Name="Ben", Age=34, designation="Sr Manager", location="US"));
    ArrayAppend(empArray,new employee(Name="Den", Age=24, designation="Software Engineer",
location="US"));
    ArrayAppend(empArray,new employee(Name="Ran", Age=28, designation="Manager", location="IND"));
    ArrayAppend(empArray,new employee(Name="Ramesh", Age=31, designation="Software Engineer",
location="IND"));
    return empArray;
    }
</cfscript>
</cfcomponent>
```

3 フィルター処理のロジックを提供するクロージャを指定して filterArray() 関数にアクセスする CFM ページを作成します。filterArray() 関数は、従業員データを所在地、年齢、呼称の 3 つの方法でフィルター処理するために使用されます。関数にアクセスするたびに、クロージャ内のフィルター処理ロジックが変更されます。

```
<!---arrayFilter.cfm--->
<cfset filteredArray = arraynew(1)>
<cfset componentArray = [3,6,8,2,4,7,9]>
<cfscript>
obj = CreateObject("component", "filter");
// Filters employees from India
filteredArray = obj.filterArray(obj.getEmployee(), function(a)
    {
    if(a.getLocation()=="IND")
        return 1;
    else
        return 0;
    });
writedump(filteredArray);
//Filters employees from india whos age is above thirty
filteredArray = obj.filterArray(obj.getEmployee(), closure(a)
    {
    if((a.getLocation()=="IND") && (a.getAge()>30))
        return 1;
    else
        return 0;
    });
writedump(filteredArray);
// Filters employees who are managers
filteredArray = obj.filterArray( obj.getEmployee(), function(a)
    {
    if((a.getdesignation() contains "Manager"))
        return 1;
    else
        return 0;
    });
writedump(filteredArray);
</cfscript>
```

関数での正規表現の使用

正規表現を使用すると、Adobe ColdFusion 関数を使用して文字列のマッチングを行えます。特に、次の関数で正規表現を使用できます。

- REFind
- REFindNoCase
- REMatch
- REMatchNoCase
- REReplace
- REReplaceNoCase

cfinput タグおよび cftextinput タグでは、JavaScript の正規表現を使用します。JavaScript と ColdFusion では、正規表現のシンタックスが若干異なります。JavaScript の正規表現については、709 ページの「cfform タグによるダイナミックフォームの作成」を参照してください。

正規表現について

ColdFusion の Find 関数や Replace 関数で実行できるような通常の文字列マッチングでは、検索する文字列のパターンと、検索対象となる文字列を指定します。次の例では、" BIG " というパターンの文字列を検索し、検索された場合はその文字列インデックスを返します。文字列インデックスとは、検索文字列における文字列パターンの開始位置です。

```
<cfset IndexOfOccurrence=Find(" BIG ", "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

この検索では、文字列パターンを正確に (具体的に) 指定する必要があります。そのパターンに正確に一致するものが検出されなかった場合、Find はインデックス 0 を返します。文字列パターンを具体的に指定する必要があるので、ダイナミックなデータマッチングは困難です。

次の例では、正規表現を使用して同じ検索を行います。この例では、前後をスペースで囲まれた、大文字のみで構成される最初の文字列を、検索文字列の中から検索します。

```
<cfset IndexOfOccurrence=REFind(" [A-Z]+ ", "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

正規表現 " [A-Z]+ " は、先頭に 1 文字のスペースがあり、その後に大文字が何文字か続き、最後に 1 文字のスペースがある文字列パターンに一致します。したがって、この正規表現は、文字列 " BIG " だけでなく、前後にスペースがある任意の大文字の文字列に一致します。

デフォルトでは、正規表現のマッチングでは大文字と小文字が区別されます。大文字と小文字を区別しない場合は、REFindNoCase 関数および REReplaceNoCase 関数を使用します。

大量のテキストデータをダイナミックに処理する状況はよく発生するので、複雑な ColdFusion アプリケーションを作成する上で正規表現は非常に重要です。

ColdFusion の正規表現関数の使用

ColdFusion には、正規表現を使用できる次の 4 つの関数が用意されています。

- REFind
- REFindNoCase
- REMatch
- REMatchNoCase
- REReplace
- REReplaceNoCase

REFind および REFindNoCase は、正規表現を使用して特定のパターンの文字列を検索し、パターンが検索された場所を表す文字列インデックスを返します。たとえば、次の関数は、最初に検索された文字列 " BIG " のインデックスを返します。

```
<cfset IndexOfOccurrence=REFind(" BIG ", "Some BIG BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

次に出現する文字列 " BIG " を検索するには、REFind 関数をもう一度呼び出す必要があります。1 つの検索文字列を繰り返し検索して、正規表現に一致する文字列をすべて検索する例については、139 ページの「一致したサブ式に関する情報の取得」を参照してください。

REReplace および REReplaceNoCase は、正規表現を使用して文字列内を検索し、その正規表現に一致する文字列パターンを、別の文字列に置き換えます。これらの関数では、最初に一致した文字列パターンのみを置き換えることも、すべての文字列パターンを置き換えることもできます。

正規表現を使用する ColdFusion 関数の詳細については、『CFML リファレンス』を参照してください。

正規表現の基本的なシンタックス

最も単純な正規表現は、リテラル文字のみで構成される正規表現です。リテラル文字は、検索するテキストを正確に (具体的に) 指定する場合に使用します。たとえば、次の正規表現関数 REFind では、Find 関数を使用した場合と同様に、文字列パターン " BIG " が検索されます。

```
<cfset IndexOfOccurrence=REFind(" BIG ", "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

この例では、具体的な文字列パターン " BIG " に正確に一致するものが検索されます。

正規表現を最大限に活用するには、次の例のように、リテラル文字、文字セット、特殊文字を組み合わせて使用します。

```
<cfset IndexOfOccurrence=REFind(" [A-Z]+ ", "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

この正規表現で使用されているリテラル文字は、先頭および末尾のスペース文字です。文字セットは、角括弧で囲まれている部分です。この文字セットは、A ～ Z の任意の大文字を 1 つ検索することを表します。角括弧の後にあるプラス符号 (+) は、その文字セットの繰り返しを検索することを表す特殊文字です。

この正規表現から + を削除して " [A-Z] " とした場合は、リテラルスペースの後に 1 文字の大文字があり、その後に 1 文字のスペースがある文字列に一致します。この正規表現は、" B " には一致しますが、" BIG " には一致しません。REFind 関数で 0 が返される場合は、正規表現に一致する文字列が検索されなかったことを表します。

リテラル文字、文字セット、特殊文字を使用すれば、複雑な正規表現を作成できます。あらゆるプログラミング言語と同様に、正規表現も慣れるに従ってさまざまな処理が記述できるようになります。この例は、ごく基本的な使用方法にすぎません。その他の例については、141 ページの「正規表現の例」を参照してください。

正規表現のシンタックス

正規表現のシンタックスにはいくつかの基本ルールと方式があります。

文字セットの使用

正規表現の角括弧の中では、1 文字のマッチングに使用する一連の文字 (文字セット) を定義します。たとえば、正規表現 " [A-Za-z] " は、前後にスペースがある 1 文字の大文字または小文字に一致します。文字セット内のハイフンは、文字の範囲を表します。

正規表現 " B[IAU]G " は、文字列 " BIG "、" BAG "、および " BUG " には一致しますが、文字列 " BOG " には一致しません。

" B[IA][GN] " のように文字セットを連結すると、対応する各文字を連結したものに一致します。この正規表現は、スペースの後に "B" が続き、その後に "I" または "A" が続き、その後に "G" または "N" が続き、最後にスペースがある文字列に一致します。つまり、" BIG "、" BAG "、" BIN "、および " BAN " に一致します。

正規表現 [A-Z][a-z]* は、1 文字の大文字で始まり、その後に 0 文字以上の小文字が続く単語に一致します。右角括弧の後の * は、その文字セットの 0 回以上の繰り返しに一致するように指定する特殊文字です。

注意： * が適用されるのはその直前の文字セットだけで、正規表現全体ではありません。

右角括弧の後の + は、その文字セットの 1 回以上の繰り返しを検索するように指定します。正規表現 "[A-Z]+" は、前後にスペースがある 1 文字以上の大文字に一致します。したがって、この正規表現は、" BIG " だけでなく、" LARGE "、" HUGE "、" ENORMOUS " など、前後にスペースがある任意の大文字の文字列に一致します。

特殊文字を使用する場合の注意事項

後ろに * が付いている正規表現は、その 0 回の繰り返しにも一致します。つまり、空の文字列にも一致します。次に例を示します。

```
<cfoutput>
    REReplace("Hello","[T]*","7","ALL") - #REReplace("Hello","[T]*","7","ALL")#<BR>
</cfoutput>
```

これは、次のような結果になります。

```
REReplace("Hello","[T]*","7","ALL") - 7H7e7l7l7o7
```

正規表現 [T]* は、空の文字列にも一致します。この場合は、まず "Hello" の "H" の前にある空の文字列に一致します。"ALL" 引数が指定されているので、REReplace は正規表現のすべてのインスタンスを置き換えます。したがって、"e" の前にある空の文字列から "o" の前にある空の文字列まで同様に置換されます。

これは、おそらく意図した結果ではないと思われます。問題を回避する方法は、場合によって異なります。たとえば、[T]* の代わりに、少なくとも 1 つの "T" がなければ一致しない [T]+ を使用することで解決できる場合があります。または、[T]* の後ろにパターンを追加することで解決できる場合もあります。

次の例では、正規表現の末尾に "W" を追加しています。

```
<cfoutput>
    REReplace("Hello World","[T]*W","7","ALL") -
        #REReplace("Hello World","[T]*W","7","ALL")#<BR>
</cfoutput>
```

この式では、より予測しやすい次のような結果が返されます。

```
REReplace("Hello World","[T]*W","7","ALL") - Hello 7orld
```

文字の繰り返しの検索

文字の繰り返しを検索文字列の中から検索することができます。たとえば、正規表現 "a{2,4}" は、"a" が 2 ～ 4 回繰り返されている部分に一致します。つまり、"aa"、"aaa"、"aaaa" には一致しますが、"a" や "aaaaa" には一致しません。次の例では、REFind 関数はインデックス 6 を返します。

```
<cfset IndexOfOccurrence=REFind("a{2,4}", "hahahaaahaaaahaaaaahhh")>
<!--- The value of IndexOfOccurrence is 6--->
```

正規表現 "[0-9]{3,}" は、3 桁以上の整数に一致します。"123"、"45678" には一致しますが、1 桁や 2 桁の整数には一致しません。

文字の繰り返しを検索するには、次のシンタックスを使用します。

1 {**m**,**n**}

m は 0 以上、**n** は **m** 以上です。**m** 回～ **n** 回の繰り返しに一致します。

式 {0,1} は、特殊文字 ? と同じです。

2 {**m**,}

m は 0 以上です。**m** 回以上の繰り返しに一致します。{,**n**} というシンタックスは使用できません。

式 {1,} は特殊文字 + と同じです。{0,} は * と同じです。

3 {**m**}

m は 0 以上です。**m** 回の繰り返しにのみ一致します。

正規表現における大文字と小文字の区別

ColdFusion の正規表現関数には、大文字と小文字を区別するものと、区別しないものがあります。REFind と REReplace は大文字と小文字を区別して検索しますが、REFindNoCase と REReplaceNoCase は区別しません。

大文字と小文字を区別する関数でも、区別しないで検索するように正規表現を記述することができます。大文字と小文字を区別しないようにするには、個々の文字を文字セットに置き換えます。たとえば、大文字と小文字を区別する REFind または REReplace 関数で正規表現 [Jj][Aa][Vv][Aa] を使用すれば、次のどの文字列パターンも検索できます。

- JAVA
- java
- Java
- jAva
- その他のすべての大文字と小文字の組み合わせ

サブ式の使用

括弧を使用して、正規表現の一部をサブ式としてグループ化すれば、1 つのユニットとして処理できます。たとえば、正規表現 "ha" は、この文字列の 1 回の繰り返しに一致しますが、正規表現 "(ha)+" は、"ha" の 1 回以上の繰り返しに一致します。

次の例では、正規表現 "B(ha)+" を使用して、文字 "B" の後に文字列 "ha" が 1 回以上繰り返されている文字列を検索します。

```
<cfset IndexOfOccurrence=REFind("B(ha)+", "hahaBhahahaha")>
<!--- The value of IndexOfOccurrence is 5 --->
```

サブ式では、特殊文字 | を使用して "OR" 論理演算を行えます。次の正規表現を使用すると、"jelly" または "jellies" という単語を検索できます。

```
<cfset IndexOfOccurrence=REFind("jell(y|ies)", "I like peanut butter and jelly">
<!--- The value of IndexOfOccurrence is 26--->
```

特殊文字の使用

正規表現では、次の特殊文字が定義されています。

```
+ * ? . [ ^ $ ( ) { | \
```

特殊文字をリテラル文字として使用できる場合もあります。たとえば、文字列内のプラス符号 (+) を検索する場合は、次のように、プラス符号の前に円記号 (¥) を付けてエスケープする必要があります。

```
"\+"
```

次の表で、正規表現で使用する特殊文字について説明します。

特殊文字	説明
¥	特殊文字の前に円記号 (¥) を付けると、その特殊文字のリテラル文字に一致します。つまり、円記号 (¥) は特殊文字のエスケープに使用します。 たとえば、"¥+" はプラス符号に、"¥¥" は円記号に一致します。
.	ピリオド (.) は、改行文字を含む、すべての文字に一致します。 改行文字以外の文字に一致させるには、[^#chr(13)##chr(10)#] を使用します。この正規表現は、ASCII のキャリッジリターンおよびラインフィードコードを除外しています。対応するエスケープコードは、¥r および ¥n です。
[]	指定された文字セット内の任意の 1 文字に一致します。 たとえば、"[akm]" は、"a"、"k"、または "m" のいずれかの文字に一致します。文字セット内のハイフンは、文字の範囲を表します。たとえば、[a-z] は任意の小文字 1 文字に一致します。 文字セットの最初の文字がキャレット (^) である場合は、文字セット内の文字以外の任意の文字に一致します。空の文字列には一致しません。 たとえば、"[^akm]" は、"a"、"k"、"m" のいずれでもない任意の文字に一致します。キャレット (^) は、文字セットの最初の文字として使用した場合にのみ、この特殊な意味を持ちます。

特殊文字	説明
^	正規表現の先頭でキャレット (^) を指定すると、一致する文字列は検索文字列の先頭にある文字列に限定されます。 たとえば、"^ColdFusion" という正規表現は、文字列 "ColdFusion lets you use regular expressions" には一致しますが、文字列 "In ColdFusion, you can use regular expressions" には一致しません。
$	正規表現の末尾でドル記号 ($) を指定すると、一致する文字列は検索文字列の末尾にある文字列に限定されます。 たとえば、"ColdFusion$" という正規表現は、文字列 "I like ColdFusion" には一致しますが、文字列 "ColdFusion is fun" には一致しません。
?	後ろに疑問符 (?) が付いている文字セットまたはサブ式は、その文字セットまたはサブ式の 0 回または 1 回の繰り返しに一致します。 たとえば、"xy?z" は "xyz" または "xz" に一致します。
|	OR 文字 (|) を使用すると、2 つの正規表現から選択できます。 たとえば、"jell(y|ies)" は "jelly" または "jellies" に一致します。
+	後ろにプラス符号 (+) が付いている文字セットまたはサブ式は、その文字セットまたはサブ式の 1 回以上の繰り返しに一致します。 たとえば、[a-z]+ は、1 文字以上の小文字に一致します。
*	後ろにアスタリスク (*) が付いている文字セットまたはサブ式は、その文字セットまたはサブ式の 0 回以上の繰り返しに一致します。 たとえば、[a-z]* は 0 文字以上の小文字に一致します。
()	括弧を使用して、正規表現の一部をサブ式としてグループ化すれば、1 つのユニットとして処理できます。 たとえば (ha)+ は、"ha" の 1 回以上の繰り返しに一致します。
(?x)	正規表現の先頭でこれを指定すると、正規表現内の空白が無視されます。また、## を使用して行の最後にコメントを付けることもできます。一致させたいスペースは円記号 (¥) でエスケープします。 たとえば、次の正規表現には ## で始まるコメントが含まれています。このコメントは検索時には無視されます。 reFind("(?x) one ##first option |two ##second option |three¥ point¥ five ## note escaped spaces ", "three point five")
(?m)	正規表現の先頭でこれを指定すると、複数行モードで特殊文字 ^ および $ が一致するようになります。 ^ とともに使用すると、検索文字列全体の先頭だけでなく、ラインフィード文字または chr(10) によって表される改行の先頭にも一致するようになります。$ とともに使用した場合は、検索文字列全体の末尾だけでなく、改行の末尾にも一致するようになります。 複数行モードでは、キャリッジリターンまたは chr(13) は改行文字として認識されません。 次の例では、複数行モードで文字列 "two" を検索します。 #reFind("(?m)^two", "one#chr(10)#two")# この例では、chr(10) ラインフィードの後の "two" に一致したことを示す 4 が返されます。(?m) を指定しなかった場合、^ は文字列の先頭にのみ一致するので、この文字列では一致するものがありません。 (?m) を使用しても、¥A または ¥Z の動作は変わりません。これらは常に文字列の先頭または末尾に一致します。¥A および ¥Z の詳細については、135 ページの「エスケープシーケンスの使用」を参照してください。
(?i)	REFind() の正規表現の先頭でこれを指定すると、大文字と小文字を区別せずに比較が行われます。 たとえば、次のコードは、インデックス 1 を返します。 #reFind("(?i)hi", "HI")# (?i) を省略した場合は、正規表現に一致するものがないので、インデックス 0 が返されます。

特殊文字	説明
(?=...)	正規表現の先頭でこれを指定すると、正規表現の検索時に肯定先読みが行われます。 サブ式の前にこれを付けると、ColdFusion はそのサブ式に肯定先読みを使用します。 肯定先読みでは、通常の括弧と同様に、囲まれたサブ式が存在するかどうかがテストされますが、一致結果にはその内容が含まれません。単に、その内容が式の他の部分に隣接しているかどうかだけがテストされます。 たとえば、URL からプロトコルを抽出する式を考えてみます。 <cfset regex = "http(?=://)"> <cfset string = "http://"> <cfset result = reFind(regex, string, 1, "yes")> mid(string, result.pos[1], result.len[1]) この例では、文字列 "http" が返されます。先読みの括弧を使用すると、"://" の存在は確認されますが、結果には "://" が含まれません。先読みを使用しなった場合は、結果に余分な "://" が含まれます。 先読みの括弧はテキストを取得しないので、バックリファレンスの番号付けでは、これらのグループはスキップされます。バックリファレンスの詳細については、137 ページの「バックリファレンスの使用」を参照してください。
(?!...)	正規表現の先頭でこれを指定すると、否定先読みが行われます。否定先読みは、(?=...) の肯定先読みに似ていますが、そのサブ式が存在しないかどうかがテストされる点が異なります。 先読みの括弧はテキストを取得しないので、バックリファレンスの番号付けでは、これらのグループはスキップされます。バックリファレンスの詳細については、137 ページの「バックリファレンスの使用」を参照してください。
(?:...)	サブ式に "?:" という接頭辞を付けると、バックリファレンスで使用するテキストとして取得されなくなります。それ以外は通常のサブ式と同様に機能します。

[a-z] のように文字セットで特殊文字を使用するときは、次の点に注意する必要があります。

- ColdFusion では、ハイフン (-) は常に範囲記号として解釈されます。したがって、角括弧内の文字セットにリテラル文字としてハイフン (-) を含めたい場合は、通常の特殊文字のようにエスケープすることはできません。文字セットにリテラル文字のハイフン (-) を含めるには、文字セットの最後の文字として指定する必要があります。
- 文字セットに右角括弧 (]) を含めるには、[1-3¥]A-z] のように円記号 (¥) でエスケープします。文字セット指定子の外にある] 文字は、エスケープする必要はありません。

エスケープシーケンスの使用

エスケープシーケンスとは、円記号 (¥) が前にある正規表現の特殊文字です。エスケープシーケンスは、正規表現で特殊文字を表すのによく使用されます。たとえば、エスケープシーケンス ¥t は正規表現でのタブ記号を表します。エスケープシーケンス ¥d は [0-9] と同じで、任意の数字を表します。ColdFusion のエスケープシーケンスでは大文字と小文字が区別されます。

次の表に、ColdFusion でサポートされているエスケープシーケンスを示します。

エスケープシーケンス	説明
¥b	英数字から英数字以外、または英数字以外から英数字に変わる境界を指定します。 たとえば、文字列 " Big" では、スペース (英数字以外) と "B" (英数字) の間に境界があります。 次の例では、正規表現にエスケープシーケンス ¥b が含まれているので、検索文字列の最後にある文字列 "Big" は検索されますが、単語 "ambiguous" の一部である "big" は検索されません。 reFindNoCase("¥bBig¥b", "Don't be ambiguous about Big.") <!--- IndexOfOccurrence の値は 26 です。---> 文字セット内で使用すると ([¥b] など)、バックスペースが指定されます。
¥B	文字タイプが変化しない境界を指定します。たとえば、英数字 2 文字または英数字以外の 2 文字が並んだ部分の境界に一致します。¥b とは逆です。
¥A	文字列の先頭のアンカーを指定します。特殊文字 ^ に似ています。 ただし、^ と異なり、¥A と (?m) を組み合わせても、検索文字列内の改行の先頭には一致しません。
¥Z	文字列の末尾のアンカーを指定します。特殊文字 $ に似ています。 ただし、$ と異なり、¥Z と (?m) を組み合わせても、検索文字列内の改行の末尾には一致しません。
¥n	改行文字
¥r	キャリッジリターン
¥t	タブ
¥f	フォームフィード
¥d	任意の数字。[0-9] と同じです。
¥D	数字以外の任意の文字。[^0-9] と同じです。
¥w	任意の英数字またはアンダースコア (_)。[[:word:]] と同じです。
¥W	英数字以外の任意の文字 (アンダースコアを除く)。[^[:word:]] と同じです。
¥s	タブ、スペース、改行文字、キャリッジリターン、フォームフィードなどの任意の空白文字。[¥t¥n¥r¥f] と同じです。
¥S	空白以外の任意の文字。[^ ¥t¥n¥r¥f] と同じです。
¥¥x	文字の 16 進表現。d は 16 進数の数字です。
¥ddd	文字の 8 進表現。d は 8 進数の数字で、¥000 ～ ¥377 の形式です。

文字クラスの使用

正規表現の文字セットには、文字クラスを含めることができます。文字クラスは、次の例のように角括弧で囲みます。

```
REReplace ("Adobe Web Site","[[:space:]]","*","ALL")
```

このコードでは、スペースがすべて * で置き換えられて、次の文字列が生成されます。

```
Adobe*Web*Site
```

文字クラスは、文字セット内で他の式と組み合わせることができます。たとえば、正規表現 [[:space:]123] は、スペース、1、2、または 3 を検索します。次の例でも、正規表現内で文字クラスが使用されています。

```
<cfset IndexOfOccurrence=REFind("[[:space:]][A-Z]+[[:space:]]",
    "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

次の表に、ColdFusion でサポートされている文字クラスを示します。これらのクラスを使用した正規表現は、ASCII または ISO-8859 の文字だけでなく、Unicode 文字にも一致します。

文字クラス	一致
:alpha:	任意のアルファベット文字。
:upper:	任意の大文字のアルファベット文字。
:lower:	任意の小文字のアルファベット文字。
:digit:	任意の数字。¥d と同じです。
:alnum:	任意の英数字。
:xdigit:	任意の 16 進数字。[0-9A-Fa-f] と同じです。
:blank:	スペースまたはタブ。
:space:	任意の空白文字。¥s と同じです。
:print:	任意の英数字、句読点、または空白文字。
:punct:	任意の句読点。
:graph:	任意の英数字または句読点。
:cntrl:	[:upper:]、[:lower:]、[:alpha:]、[:digit:]、[:punct:]、[:graph:]、[:print:]、または [:xdigit:] の文字クラスに含まれない、任意の文字。
:word:	任意の英数字およびアンダースコア (_)。¥w と同じです。
:ascii:	16 進数で 0 ～ 7F の範囲の ASCII 文字。

バックリファレンスの使用

括弧を使用すると、正規表現のコンポーネントをサブ式にグループ化できます。たとえば、正規表現 "(ha)+" は、文字列 "ha" の 1 回以上の繰り返しに一致します。

ColdFusion では、サブ式を使用すると追加の処理が行われます。つまり、検索文字列内のサブ式に一致する部分が自動的に保存されて、正規表現の中で再利用できるようになります。保存されたサブ式のテキストを参照することを、バックリファレンスと呼びます。

バックリファレンスは、たとえば "the the" や "is is" のような重複している単語を文字列内から検索する場合に使用します。次の例では、バックリファレンスを使用して検索文字列内の重複している単語をすべて検索し、それらをアスタリスクに置き換えます。

```
REReplace("There is is coffee in the the kitchen",
    "[ ]+([A-Za-z]+)[ ]+\1"," * ","ALL")
```

この正規表現を使用すると、重複している "is" と "the" が検出されて、スペースで囲まれたアスタリスクに置き換えられ、次の文字列が返されます。

```
There * coffee in * kitchen
```

正規表現 [ ]+([A-Za-z]+)[ ]+¥1 は、次のように解釈されます。

サブ式 ([A-Za-z]+) で表される 1 文字以上の文字列が、1 つ以上のスペース [ ]+ で囲まれており、その後に最初のサブ式 ¥1 で一致した文字列と同じ文字列が続いている文字列が検索されます。

サブ式で一致した文字列を参照するには、円記号の後に数字 **n** を付けます (**¥n**)。最初のサブ式は ¥1、2 番目のサブ式は ¥2 のように記述します。次のセクションでは、複数のバックリファレンスを使用する例を示します。

置換文字列でのバックリファレンスの使用

REReplace 関数および REReplaceNoCase 関数の置換文字列では、バックリファレンスを使用できます。たとえば、次のシンタックスを使用すると、テキスト文字列内の最初の重複単語を 1 つの単語に置き換えることができます。

```
REReplace("There is is a cat in in the kitchen",
    "([A-Za-z ]+)\1","\1")
```

この結果、次の文が返されます。

```
"There is a cat in in the kitchen"
```

次のコードのように、REReplace の 4 番目のパラメータである **scope** (オプション) を使用して、すべての重複語を置き換えることもできます。

```
REReplace("There is is a cat in in the kitchen",
    "([A-Za-z ]+)\1","\1","ALL")
```

この結果、次の文字列が返されます。

```
"There is a cat in the kitchen"
```

次の例では、2 つのバックリファレンスを使用して、文中の "apples" と "pears" を入れ替えます。

```
<cfset astring = "apples and pears, apples and pears, apples and pears">
<cfset newString = REReplace("#astring#", "(apples) and (pears)",
    "\2 and \1","ALL")>
```

この例では、サブ式 (apples) を ¥1、サブ式 (pears) を ¥2 として参照しています。REReplace 関数は次の文字列を返します。

```
"pears and apples, pears and apples, pears and apples"
```

注意：検索文字列や置換文字列でバックリファレンスを使用するためには、正規表現内で括弧を使用して、対応するサブ式を作成する必要があります。そうでない場合は、例外が発生します。

バックリファレンスを使用した置換での大文字小文字の変換

REReplace 関数および REReplaceNoCase 関数の置換文字列では、置換する文字を大文字または小文字に変換する特殊文字を使用できます。次の表で、これらの特殊文字について説明します。

特殊文字	説明
¥u	次にある文字を大文字に変換します。
¥l	次にある文字を小文字に変換します。
¥U	¥E が現れるまで、すべての文字を大文字に変換します。
¥L	¥E が現れるまで、すべての文字を小文字に変換します。
¥E	¥U または ¥L を終了します。

¥u や他のコードを置換文字列にリテラルとして含めたい場合は、¥¥u のように円記号 (¥) を重ねてエスケープします。

たとえば、次のステートメントは、大文字の文字列 "HELLO" を小文字の "hello" に置き換えます。この例では、バックリファレンスを使用して置換しています。

```
REReplace("HELLO", "([[:upper:]]*)", "Don't shout\scream \L\1")
```

この例の結果は、文字列 "Don't shout¥scream hello" になります。

置換文字列での特殊文字のエスケープ

置換文字列の中でバックリファレンスおよび大文字小文字の変換記号をエスケープするには、円記号 (¥) を使用します。たとえば、置換文字列にリテラル "¥u" を含めるには、"¥¥u" のようにしてエスケープします。

バックリファレンスからのサブ式の除外

デフォルトでは、1 組の括弧には、サブ式をグループ化する機能と、一致したテキストを取得してバックリファレンスで再利用できるようにする機能の両方があります。ただし、サブ式の先頭に "?:" を挿入すると、バックリファレンスで使用するテキストとして取得されなくなります。それ以外は通常のサブ式と同様に機能します。

これは、含まれているグループの数が異なるサブ式の間で選択を行っているために、バックリファレンスの番号付けが困難である場合に便利です。たとえば、Bonjour|Hi|Hello と Bond の間に "Mr." を挿入する式で、グループをネストして Hi と Hello を表している場合を考えてみます。

```
<cfset regex = "(Bonjour|H(?:i|ello))( Bond)">
<cfset replaceString = "\1 Mr.\2">
<cfset string = "Hello Bond">
#REReplace(string, regex, replaceString)#
```

この例は、"Hello Mr. Bond" を返します。Hi/Hello グループの取得を禁止しなかった場合、¥2 バックリファレンスは "Bond" ではなくそのグループを参照してしまうので、"Hello Mr.ello" という結果が返されます。

一致したサブ式に関する情報の取得

REFind 関数および REFindNoCase 関数は、検索文字列内にある、正規表現と最初に一致した文字列の位置を返します。次の例の検索文字列には、正規表現に一致する文字列が 2 つありますが、最初の文字列のインデックスのみが返されます。

```
<cfset IndexOfOccurrence=REFind(" BIG ", "Some BIG BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

正規表現に一致する箇所をすべて検索するには、REFind 関数および REFindNoCase 関数を複数回呼び出す必要があります。

REFind 関数および REFindNoCase 関数には、オプションの第 3 パラメータがあり、検索文字列のどのインデックスから検索を開始するかを指定できます。デフォルトでは、開始位置のインデックスは 1 (文字列の先頭) です。

この例で、正規表現の 2 番目の一致箇所を検索するには、開始インデックスを 8 に指定して REFind を呼び出します。

```
<cfset IndexOfOccurrence=REFind(" BIG ", "Some BIG BIG string", 8)>
<!--- The value of IndexOfOccurrence is 9 --->
```

この場合は、2 つ目の " BIG " の開始インデックスであるインデックス 9 が返されます。

2 つ目の文字列を検索するためには、最初の文字列がインデックス 5 にあり、長さが 5 であるという情報が必要です。しかし、REFind では、文字列の開始インデックスは返されますが、長さは返されません。したがって、2 回目の REFind を実行するためには、一致した文字列の長さを調べるか、または正規表現でサブ式を使用する必要があります。

REFind 関数および REFindNoCase 関数では、一致したサブ式に関する情報を取得できます。これらの関数の第 4 パラメータである ReturnSubExpression を True に設定すると、pos と len の 2 つの配列を持つ CFML 構造体が返されます。この構造体には、正規表現のサブ式に一致したテキスト文字列の位置と長さが含まれています。次に例を示します。

```
<cfset sLenPos=REFind(" BIG ", "Some BIG BIG string", 1, "True")>
<cfoutput>
    <cfdump var="#sLenPos#">
</cfoutput><br>
```

pos 配列の要素 1 には、正規表現に一致した文字列の、検索文字列における開始インデックスが含まれています。len 配列の要素 1 には、一致した文字列の長さが含まれています。この例では、最初の "BIG" 文字列のインデックスは 5 であり、長さも 5 です。正規表現に一致する文字列が存在しない場合、pos 配列と len 配列は、値が 0 の要素を 1 つ含みます。

返された情報は、mid などの他の文字列関数で使用できます。次の例は、検索文字列内の正規表現に一致する文字列を返します。

```
<cfset myString="Some BIG BIG string">
<cfset sLenPos=REFind(" BIG ", myString, 1, "True")>
<cfoutput>
    #mid(myString, sLenPos.pos[1], sLenPos.len[1])#
</cfoutput>
```

pos 配列の 2 番目以降の各要素には、各サブ式に最初に一致した検索文字列内の文字列の位置が含まれます。len 配列の 2 番目以降の各要素には、各サブ式に一致した文字列の長さが含まれます。

前述の例では、正規表現 " BIG " にはサブ式が含まれていません。したがって、REFind 関数によって返される構造体の各配列の要素は 1 つです。

前述の例を実行した後、REFind をもう一度呼び出して、正規表現の次の一致箇所を検索できます。2 回目の呼び出しでは、最初の呼び出しで得た情報を使用します。

```
<cfset newstart = sLenPos.pos[1] + sLenPos.len[1] - 1>
<!--- subtract 1 because you need to start at the first space --->
<cfset sLenPos2=REFind(" BIG ", "Some BIG BIG string", newstart, "True")>
<cfoutput>
    <cfdump var="#sLenPos2#">
</cfoutput><br>
```

正規表現にサブ式が含まれている場合は、pos と len の要素 1 の後の各要素で、各サブ式に一致する検索文字列内の最初の文字列の位置と長さが示されます。

次の例で、式 [A-Za-z]+ は、正規表現のサブ式です。式 ([A-Za-z]+)[ ]+ によって、"is is" が最初に一致します。

```
<cfset sLenPos=REFind("([A-Za-z]+)[ ]+\1",
    "There is is a cat in in the kitchen", 1, "True")>
<cfoutput>
    <cfdump var="#sLenPos#">
</cfoutput><br>
```

sLenPos.pos[1] と sLenPos.len[1] には、正規表現全体の一致に関する情報が含まれています。配列要素 sLenPos.pos[2] と sLenPos.len[2] には、最初のサブ式 ("is") に関する情報が含まれています。REFind では、正規表現の最初の一致に関する情報のみが返されるので、sLenPos 構造体には、2 番目の一致である "in in" に関する情報は含まれません。

次の例の正規表現では、2 つのサブ式を使用しています。したがって、返される構造体の各配列には、正規表現全体に最初に一致した文字列、最初のサブ式に最初に一致した文字列、および 2 番目のサブ式に最初に一致した文字列の、位置と長さが含まれます。

```
<cfset sString = "apples and pears, apples and pears, apples and pears">
<cfset regex = "(apples) and (pears)">
<cfset sLenPos = REFind(regex, sString, 1, "True")>
<cfoutput>
    <cfdump var="#sLenPos#">
</cfoutput>
```

サブ式の使用方法の詳細については、『CFML リファレンス』の ColdFusion 関数に関する章の REFind および REFindNoCase のセクションを参照してください。

最短一致の指定

正規表現の修飾子である ?、*、+、{min,} および {min,max} では、式の繰り返し回数の最小、最大、またはその両方が指定されます。デフォルトでは、正規表現に一致する最長の文字列が検索文字列の中から検索されます。この動作を最長一致と呼びます。

たとえば、正規表現 "(.*)" を使用して文字列 "one two" を検索するとします。正規表現 "(.*)" は、次の両方に一致します。

- one
- one two

デフォルトでは、正規表現に一致する最長の文字列が検索文字列の中から検索されます。次のコードは、この例の結果を示します。

```
<cfset sLenPos=REFind("<b>(.*)</b>", "<b>one</b> <b>two</b>", 1, "True")>
<cfoutput>
    <cfdump var="#sLenPos#">
</cfoutput><br>
```

このように、文字列の開始位置は 1 でその長さは 21 です。これは、2 つの一致候補のうちの長いほうに対応します。

しかし、このデフォルトの動作を無効にして、正規表現に一致する最短の文字列を検索することもできます。ColdFusion には、最短の文字列に一致するように指定する、最短一致修飾子が用意されています。次の表で、これらの式について説明します。

式	説明
*?	* の最短一致バージョン
+?	+ の最短一致バージョン
??	? の最短一致バージョン
{min,}?	{min,} の最短一致バージョン
{min,max}?	{min,max} の最短一致バージョン
{n}?	{n} と同じです。表記上の一貫性を保つためにサポートされています。

前の例に変更を加えて、最短一致のシンタックスを使用すると、次のようなコードになります。

```
<cfset sLenPos=REFind("<b>(.*?)</b>", "<b>one</b> <b>two</b>", 1, "True")>
<cfoutput>
    <cfdump var="#sLenPos#">
</cfoutput><br>
```

このように、正規表現で検索された文字列の長さは 10 です。これは文字列 "one" に対応します。

正規表現の例

次の表に、正規表現の例と、それに一致するものを示します。

式	説明
[¥?&]value=	URL に含まれている URL パラメータ値
[A-Z]:(¥¥[A-Z0-9_]+)+	大文字の DOS/Windows パスのうち、ドライブのルートではなく、英数字とアンダースコアのみで構成されているパス
^[A-Za-z][A-Za-z0-9_]*	修飾子のない ColdFusion 変数
([A-Za-z][A-Za-z0-9_]*)(¥.[A-Za-z][A-Za-z0-9_]*)?	修飾子を 1 つしか持たない ColdFusion 変数 (たとえば、Form.VarName には一致し、Form.Image.VarName には一致しない)
(¥+|-)?[1-9][0-9]*	0 で始まらない整数 (符号が付いている場合はそれも含む)
(¥+|-)?[0-9]+(¥.[0-9]*)?	実数値
(¥+|-)?[1-9]¥.[0-9]*E(¥+|-)?[0-9]+	科学的記数法で表された実数値
a{2,4}	"a" の 2 回～ 4 回の繰り返し (aa、aaa、aaaa)
(ba){3,}	3 ペア以上の "ba"(bababa、babababa など)

CFML の正規表現

次の表に、CFML での正規表現関数の使用例を示します。

式	戻り値
REReplace (CGI.Query_String, "CFID=[0-9]+[&]*", "")	パラメータ CFID とその数値を削除したクエリー文字列
REReplace("I Love Jellies", "[[:lower:]]","x","ALL"	I Lxxx Jxxxxxx
REReplaceNoCase("cabaret","[A-Z]", "G","ALL")	GGGGGGG
REReplace (Report,"¥$[0-9,]*¥.[0-9]*","$***.**")", "")	変数 Report の文字列値で、ドル形式の正の数値をすべて "$***.**" に置き換えたもの。
REFind ("[Uu]¥.?[Ss]¥.?[Aa]¥.?", Report)	変数 Report 内で略語 USA が最初に現れる位置。大文字と小文字は区別されず、文字の後にピリオドがある場合も含まれます。
REFindNoCase("a+c","ABCAACCDD")	4
REReplace("There is is coffee in the the kitchen","([A-Za-z]+)[ ]+¥1","*","ALL")	There * coffee in * kitchen
REReplace(report, "<[^>]*>", "", "All")	変数 report の文字列値から、HTML タグをすべて削除したもの

正規表現のタイプ

プログラマが利用できる正規表現にはさまざまなタイプがあります。たとえば、JavaScript、Perl、POSIX の正規表現はすべて異なります。それぞれの正規表現には、独自のシンタックス仕様があり、必ずしも他の正規表現と互換性があるわけではありません。

ColdFusion では、Perl 準拠の正規表現がサポートされていますが、次の例外があります。

- ピリオド (.) は、常に改行文字にも一致します。
- 置換文字列では、バックリファレンス変数に $n ではなく ¥n を使用します。ColdFusion では、置換文字列に含まれるすべての $ がエスケープされます。
- 置換文字列で円記号 (¥) をエスケープする必要はありません。円記号 (¥) は ColdFusion によってエスケープされます。ただし、大文字小文字の変換シーケンスまたはそのエスケープバージョン (¥u または ¥¥u) は例外です。
- 埋め込まれている修飾子 ((?i) など) は、グループ内にある場合でも、常に式全体に作用します。
- 置換文字列では、¥ はサポートされません。¥u¥L の組み合わせおよび ¥l¥U の組み合わせもサポートされません。

次の Perl ステートメントはサポートされていません。

- 戻り読み (?<=) (<?!)
- ¥¥x
- ¥N
- ¥p
- ¥C

正規表現については、『詳説正規表現』(原書『Mastering Regular Expressions』Jeffrey E. F. Friedl 著、歌代和正監訳、春遍雀来、鈴木武生共訳、1999 年 4 月オライリージャパン発行、ISBN : 4-900900-45-1、www.oreilly.co.jp) を参照してください。

第 5 章：ColdFusion アプリケーションのビルディングブロック

ColdFusion 要素の作成

ColdFusion 要素を作成して、コードの構成要素 (ビルディングブロック) として使用できます。一度作成した要素は、コピーすることなく、さまざまな場所で再利用できます。

作成できる CFML 要素について

Adobe ColdFusion には、アプリケーションで何度も再利用できるコードセクションを作成するための方法や要素が数多く用意されています。これらの要素の多くは、ColdFusion のビルトイン機能を拡張するためにも使用できます。ColdFusion に用意されている方法および要素は、次のとおりです。

- cfinclude タグを使用してインクルードする ColdFusion ページ
- ユーザー定義関数 (UDF)
- ColdFusion コンポーネント
- カスタム CFML タグ
- CFX (ColdFusion Extension) タグ

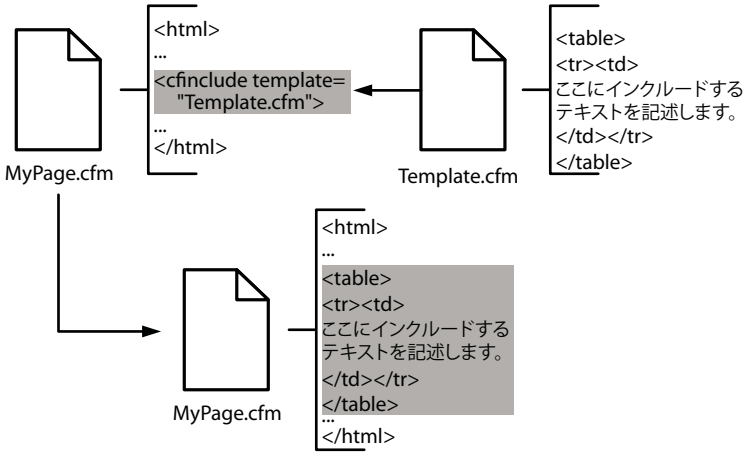
ColdFusion では、これらの要素の他に、次の方法で開発した要素も使用できます。

- JSP タグライブラリの JSP タグ。JSP タグの使用方法の詳細については、1109 ページの「CFML アプリケーションへの J2EE および Java 要素の統合」を参照してください。
- Java ランタイム環境や JavaBeans のオブジェクトなどの Java オブジェクト。Java オブジェクトの使用方法の詳細については、1109 ページの「CFML アプリケーションへの J2EE および Java 要素の統合」を参照してください。
- Microsoft COM (Component Object Model) オブジェクト。COM オブジェクトの使用方法の詳細については、1158 ページの「CFML アプリケーションでの COM および CORBA オブジェクトの統合」を参照してください。
- CORBA (Common Object Request Broker Architecture) オブジェクト。CORBA オブジェクトの使用方法の詳細については、1158 ページの「CFML アプリケーションでの COM および CORBA オブジェクトの統合」を参照してください。
- Web サービス。Web サービスの使用方法の詳細については、1061 ページの「Web サービスの使用」を参照してください。

cfinclude タグによるページのインクルード

cfinclude タグを使用すれば、ColdFusion ページに別の ColdFusion ページの内容を追加できます。インクルードしたページに含まれているコードは、インクルード先のページに存在しているものと見なされて処理されます。これによって、一度記述した ColdFusion 要素を何度も再利用する (さまざまなページで使用する) ことができます。同じコードを複数のページにコピーして保守する代わりに、1 つのページにコードを保存して、そのページをさまざまなページから参照することができます。たとえば、cfinclude タグを使用して、複数のページにヘッダやフッタを挿入することができます。この方法を使用すれば、ヘッダやフッタのデザインを変更する場合でも、1 つのファイルの内容を変更するだけで済みます。

インクルードしたページに含まれているコードは、実際には別のファイルに存在していますが、インクルード先のページに存在しているものと見なされて処理されます。インクルードされるページに cfinclude タグでパラメータを渡すことはできませんが、インクルードされるページでは、インクルード先のページにあるすべての変数にアクセスできます。次の図に、このモデルを示します。



cfinclude タグの使用

cfinclude タグを使用して、ある ColdFusion ページを別の ColdFusion ページにインクルードする場合、インクルード先のページのことを呼び出しページと呼びます。ColdFusion は cfinclude タグを検出すると、呼び出しページのタグを、インクルードしたページの処理結果で置き換えます。インクルードされるページでは、呼び出しページの変数を設定することもできます。

次の行はサンプルの cfinclude タグです。

```
<cfinclude template = "header.cfm">
```

注意：CFML のコードブロックを、複数のページにまたがって記述することはできません。たとえば、ある ColdFusion ページで cfoutput ブロックを開いた場合は、そのページ内でそのブロックを閉じる必要があります。インクルードされるページにブロックの終了部分を記述することはできません。

ColdFusion では、次のようにしてインクルードファイルが検索されます。

- template 属性によって、呼び出しページが格納されているディレクトリからの相対パスが指定されます。
- template 値がスラッシュ (/) で始まっている場合は、ColdFusion Administrator の [マッピング] ページで指定されているディレクトリでインクルードファイルが検索されます。

重要：ページでそのページ自身をインクルードしないでください。これを行うと、無限処理ループに陥ります。この問題を解決するには、ColdFusion サーバーを停止します。

呼び出しページにおけるコードのインクルード

1 ロゴを表示する "header.cfm" という ColdFusion ページを作成します。ここでは次の行を使用しますが、より詳細なコードを記述して完全なヘッダを定義してもかまいません。

```
<img src="mylogo.gif">
<br>
```

この例を実行するには、ロゴを GIF ファイルとして "header.cfm" と同じディレクトリに用意しておく必要があります。

2 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Test for Include</title>
</head>
<body>
    <cfinclude template="header.cfm">
</body>
</html>
```

3 このファイルに "includeheader.cfm" という名前を付けて保存し、ブラウザで表示します。

ヘッダがロゴとともに表示されます。

推奨される使用方法

次の場合は cfinclude タグの使用を検討してください。

- ページのヘッダやフッタ
- 理解および管理しやすいように、大規模なページを複数の論理的な部分に分割する場合
- パラメータを必要としないか、または関数やタグのモデルに納まらない、多くの場所で使用される大規模なコードスニペット

ユーザー定義関数について

ユーザー定義関数 (UDF) を使用すると、渡した引数に応じて戻り値が返されるアプリケーション要素を作成できます。UDF を定義するには、CFScript または cffunction タグを使用します。この 2 つの方法にはいくつか違いがありますが、最も重要なのは次の点です。

- cffunction タグを使用する場合は、関数に CFML タグを含めることができます。
- CFScript を使用して関数を作成する場合は、CFML タグを含めることはできません。

UDF は、標準の ColdFusion 関数と同じようにアプリケーションページで使用することができます。よく使用するアルゴリズムやプロシージャを関数にしておけば、そのプロシージャが必要になったときに、ColdFusion のビルトイン関数と同じようにいつでも呼び出すことができます。たとえば次の行では、関数 MyFunct を呼び出して、2 つの引数を渡しています。

```
<cfset returnValue=MyFunct(Arg1, Arg2)>
```

関連する関数は、1 つの ColdFusion コンポーネントにグループ化することができます。詳細については、146 ページの「ColdFusion コンポーネントの使用」を参照してください。

カスタムタグと同様に、UDF も簡単に他のユーザーに配布できます。たとえば、Common Function Library Project (www.cflib.org) は、CFML ユーザー定義関数のオープンソースのコレクションです。

推奨される使用方法

UDF の代表的な使用方法には次のものがあります。

- データ操作ルーチン (配列を反転する関数など)
- 文字列や日付に関するルーチン (文字列が有効な IP アドレスかどうかを確認する関数など)
- 算術演算ルーチン (標準の三角関数や統計演算、ローン返済計算など)
- COM や CORBA などを使用して外部の関数を呼び出すルーチン (Windows ファイルシステムドライブの空き領域を確認するルーチンなど)

次の場合は UDF の使用を検討してください。

- いくつかの引数を渡し、結果を処理して値を戻す必要がある場合。UDF は、複合値 (複数の単純値を含む構造体など) を戻すことができます。
- データ操作関数などの論理ユニットを作成する場合。
- 再帰的に処理を行う必要がある場合。
- コードを他のユーザーに配布する場合。

UDF でもカスタム CFML タグでも同じ目的が達成できる場合は、まず UDF の作成を検討します。カスタムタグを使用するよりも UDF を実行するほうがシステムのオーバーヘッドが少ないからです。

詳細情報

ユーザー定義関数の詳細については、150 ページの「ユーザー定義関数の作成と呼び出し」を参照してください。

ColdFusion コンポーネントの使用

ColdFusion コンポーネント (CFC) は、関連する関数や各関数の引数などが含まれた、ColdFusion テンプレートです。CFC には、関数や引数を定義したり値を返したりするのに必要な CFML タグが含まれています。ColdFusion コンポーネントは、.cfc 拡張子を使用して保存されます。

CFC を使用すれば、オブジェクトが持つ便利な機能を、シンプルな CFML で利用できます。関連する関数を 1 つのユニットにまとめることができる CFC は、さまざまな関数を提供するオブジェクトまたはクラスシェルとして活用できます。

ColdFusion コンポーネントでは、データをプライベートにすることができます。プライベートに設定したデータは、コンポーネント内のどの関数 (メソッド) からもアクセスできますが、そのコンポーネントを使用するアプリケーションからはアクセスできません。

ColdFusion コンポーネントには、次の機能があります。

- 関連するサービスを 1 つのユニットで提供できます。
- Web サービスを作成して、インターネットで公開できます。
- Flash クライアントから直接呼び出せる ColdFusion サービスを提供できます。
- データの隠蔽、継承、パッケージ、イントロスペクションなど、オブジェクト指向プログラミングの特徴的な機能が使用できます。

推奨される使用方法

次の場合は ColdFusion コンポーネントの使用を検討してください。

- Web サービスを作成する場合。ColdFusion で Web サービスを作成するには、コンポーネントを使用する必要があります。
- Flash クライアントから呼び出せるサービスを作成する場合。
- 関連する関数のライブラリを作成する場合 (特に、それらの関数間でデータを共有する必要がある場合)。
- ロールおよびリクエスタの所在に基づいた統合アプリケーションセキュリティメカニズムを使用する場合。
- オブジェクト指向でコードを開発する場合。オブジェクト指向では、オブジェクトをメソッドで操作し、既存のオブジェクトの機能を拡張するオブジェクトを作成できます。

詳細情報

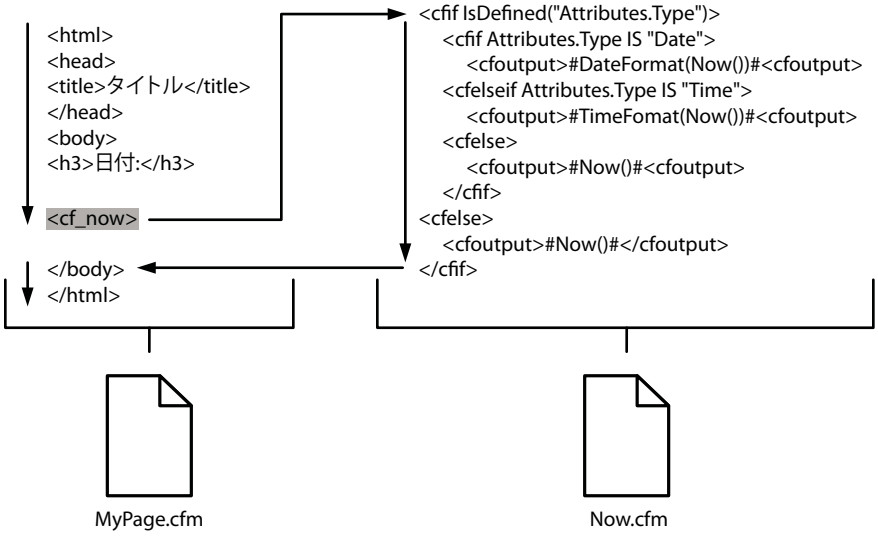
ColdFusion コンポーネントの使用方法の詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

カスタム CFML タグの使用

CFML で作成されたカスタムタグは、ColdFusion タグと同じように動作します。次のことが可能です。

- 引数を取ることができます。
- 開始タグおよび終了タグとともにタグ本文を記述できます。
- ColdFusion で開始タグが検出されたときに、特定の処理を行うことができます。
- ColdFusion で終了タグが検出されたときに、開始タグと異なる処理を行うことができます。
- タグ本文には、ColdFusion の有効なページコンテンツを記述できます。たとえば、ColdFusion のビルトインタグやカスタムタグ (ネストタグ)、JSP タグや JavaScript も使用できます。
- 再帰的に呼び出せます。正しく設計されていれば、タグ本文でそのタグ自身を呼び出すことができます。
- 呼び出しページの共通スコープ (Variables スコープ) に値を戻せます。ただし、関数のような、値を直接戻す機能はありません。

カスタムタグページも、cfinclude タグを使用してインクルードする ColdFusion ページも、どちらも ColdFusion ページですが、両者は処理方法が異なります。呼び出しページがカスタムタグを呼び出すと、カスタムタグページに処理が渡されて、呼び出しページはその処理が完了するまで待機します。カスタムタグが終了すると、呼び出しページに処理 (および場合によってデータ) が戻されます。その後、呼び出しページはその処理を完了します。このプロセスを次の図に示します。矢印は、ColdFusion がページを処理するフローを示します。



カスタム CFML タグの呼び出し

ビルトインタグとは異なり、カスタム CFML タグは次の 3 つの方法で実行できます。

- 直接タグを呼び出します。
- cfmessagebox タグを使用してタグを呼び出します。
- cfimport タグを使用してカスタムタグライブラリディレクトリをインポートします。

CFML カスタムタグを直接呼び出すには、タグファイル名の前に cf_ を付けて .cfm 拡張子を外した文字列を、山括弧 (<>) で囲みます。たとえば、ファイル "mytag.cfm" で定義したカスタムタグを呼び出すには、次の行のようにします。

```
<cf_myTag>
```

タグに本文がある場合は、次のように、同じタグ名の前にスラッシュ (/) を付けたタグで終了します。

```
</cf_myTag>
```

cfmodule および cfimport タグを使用してカスタム CFML タグを呼び出す方法については、206 ページの「カスタム CFML タグの作成と使用」を参照してください。

推奨される使用方法

ColdFusion カスタムタグを使用すると、複雑なコードやプログラミングロジックを簡潔なユニットとして表すことができ、CFML のデザインスキームを保つことができます。カスタムタグは簡単に配布して他のユーザーと共有できます。たとえば、ColdFusion Developer Exchange には、高度なジョブを実行するさまざまなカスタムタグのライブラリがあります。www.adobe.com/go/learn_cfu_exchange_jp を参照してください。

次の場合は CFML カスタムタグの使用を検討してください。

- 本文と終了タグを持つタグ形式の構造が必要で、本文のコンテンツが事前に確定していない場合。
- 特定の処理を開始タグ、終了タグ、またはその両方に関連付ける場合。
- タグ本文で " 子 " タグ (サブタグ) を使用した論理構造を作成できるようにする場合。この構造は、サブタグで個々のフォームフィールドを表す cfform タグに似ています。
- 関数のように使用する (呼び出しコードで戻り値を直接使用する) 必要がない場合。
- 再帰的に処理を行う必要がある場合。
- 機能が複雑である場合。
- 利用しやすい形で他のユーザーにコードを配布したい場合。

UDF でもカスタム CFML タグでも同じ目的が達成できる場合は、まず UDF の作成を検討してください。カスタムタグを使用するよりも UDF を実行するほうがシステムのオーバーヘッドが少ないからです。

詳細情報

カスタム CFML タグの詳細については、206 ページの「カスタム CFML タグの作成と使用」を参照してください。

CFX タグの使用

CFX (ColdFusion Extension) タグは、Java または C++ で作成するカスタムタグです。CFML では実現できない処理を行いたい場合によく使用されます。CFX タグを使用すれば、これまでに作成した Java や C++ のコードを ColdFusion アプリケーションで活用することができます。CFML カスタムタグと異なり、CFX タグは本文や終了タグを持つことができません。

CFX タグでは、ページ変数に情報を渡したり、呼び出しページにテキストを出力したりして、呼び出しページに情報を返すことができます。

CFX タグでは、次のことが行えます。

- 任意の数のカスタム属性を持つことができます。
- ColdFusion クエリーの作成および操作が行えます。
- クライアントに返す HTML をダイナミックに生成できます。
- 呼び出しページの変数を設定できます。
- 標準的な ColdFusion エラーメッセージとして処理される例外を返すことができます。

CFX タグの呼び出し

CFX タグを使用するには、クラス名の前に cfx_ を付けた文字列を、山括弧で囲みます。たとえば、MyCFXClass クラスで定義した CFX タグを呼び出して 1 つの属性を渡すには、次の行を使用します。

```
<cfx_MyCFXClass myArgument="arg1">
```

推奨される使用方法

CFX タグを使用すれば、C++ や Java を使用してコードを記述することができます。cfobject タグでも、Java クラスや COM オブジェクトを作成して使用できますが、CFX タグを使用すれば、cfobject タグにない次のビルトイン機能が利用できます。

- CFX タグのほうが、CFML コードで簡単に呼び出せます。CFX タグは、通常のタグと同じように CFML コードに直接記述でき、標準のタグ形式を使用して引数を渡すことができます。
- ColdFusion には、CFX タグの開発に役立つ、Java や C++ のコードで使用できる定義済みのクラスが用意されています。リクエストの処理をサポートするクラスや、エラーレポートの作成に役立つクラス、クエリー管理用のクラスなどが用意されています。

次の場合は CFX タグを使用したほうが便利です。

- ColdFusion アプリケーションに組み込みたい機能が、既に C++ または Java で作成されている場合。
- 必要な機能が ColdFusion 要素では作成できない場合。
- cfobject タグを使用してネイティブの Java オブジェクトや COM オブジェクトをインポートするのではなく、新しい機能をタグ形式で提供したい場合。
- ColdFusion に用意されている、CFX コード開発用の Java クラスや C++ クラスを利用したい場合。

詳細情報

CFX タグの詳細については、221 ページの「CFXAPI カスタムタグの構築」を参照してください。

ColdFusion コードの再利用方法の選択

次の表に、コードを再利用する代表的な目的を示し、それぞれの目的に対して候補となる方法を示します。**P** は、推奨される方法 (Preferred) であることを示します。推奨される方法が複数あることもあります。**A** は、状況によって有用な代替方法 (Alternative) であることを示します。

この表には CFX タグは含まれません。CFX タグを使用するのは、C++ または Java で機能を作成する必要がある場合のみです。CFX タグの使用方法の詳細については、148 ページの「CFX タグの使用」を参照してください。

用途	cfinclude タグ	カスタムタグ	UDF	コンポーネント
複数のページで使用する必要があるコード (CFML、HTML、スタティックテキストなど) を作成する	P			
ヘッダやフッタをデプロイする	P			
あるページを別のページに挿入する	P			
小さなユニットにページを分割する	P			
呼び出しページの変数を使用する	A	P	P	
再帰的な処理を行う		P	P	P
他のユーザーにコードを配布する		P	P	P
HTML または CFML のテキスト本文を操作する		P		
サブタグを使用する		P		
計算やデータ操作などの処理を提供する		A	P	
任意の数の入力値を取り、(多くの場合複雑な) 結果を返す機能を単独のユニットとして提供する		A	P	

用途	cfinclude タグ	カスタムタグ	UDF	コンポーネント
毎回名前が変わる変数を使用する		A	P	P
Flash クライアントからアクセスできるようにする		A	A	P
ビルトインのユーザーセキュリティ機能を使用する			A	P
複数の関連する関数やプロパティをカプセル化する				P
Web サービスを作成する				P
オブジェクト指向でコーディングする				P

ユーザー定義関数の作成と呼び出し

頻繁に呼び出すアルゴリズムやプロシージャをカスタム関数として定義すれば、Adobe ColdFusion のアプリケーションページで再利用することができます。

ユーザー定義関数について

ColdFusion では、ユーザー定義関数 (UDF) と呼ばれる、ユーザー独自のカスタム関数を作成することができます。UDF は、標準の ColdFusion 関数と同じようにアプリケーションページで使用することができます。関連する関数を ColdFusion コンポーネントにグループ化して、作成した関数を整理することもできます。詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

よく使用するアルゴリズムやプロシージャを関数にしておけば、そのプロシージャが必要になったときにいつでも呼び出すことができます。そのプロシージャを変更する必要がある場合でも、1 つの場所でコードを変更するだけで済みます。作成した関数は、タグ属性の中、出力におけるシャープ記号 (#) の間、CFScript コードの中など、ColdFusion 式が使用できる場所であればどこでも使用できます。UDF の代表的な使用方法には次のものがあります。

- データ操作ルーチン (配列を反転する関数など)
- 文字列や日付に関するルーチン (文字列が有効な IP アドレスかどうかを確認する関数など)
- 算術演算ルーチン (標準の三角関数や統計演算、ローン返済計算など)
- COM や CORBA などを使用して外部の関数を呼び出すルーチン (Windows ファイルシステムドライブの空き領域を確認するルーチンなど)

ユーザー定義関数、ColdFusion コンポーネント、カスタムタグの使い分けについては、143 ページの「ColdFusion 要素の作成」を参照してください。

注意：Common Function Library Project (www.cflib.org) は、CFML ユーザー定義関数のオープンソースのコレクションです。

ユーザー定義関数の作成

UDF を作成する前に、UDF を定義する場所や、関数の作成に CFML を使用するか CFScript を使用するかを決定します。

ユーザー定義関数の作成場所の決定

関数は、次の場所で定義できます。

- ColdFusion コンポーネント。ColdFusion コンポーネントで定義した関数は、184 ページの「ColdFusion コンポーネントの使用」の説明に従って使用します。

- 関数を呼び出すページ。ページ内の関数呼び出しより下の場所で関数を定義することもできますが、これはよいコーディングスタイルとはいえず、コードが読みにくくなります。
- cfinclude タグを使用してインクルードするページ。cfinclude タグは、関数を呼び出す前に実行する必要があります。たとえば、アプリケーションのすべての関数を単一のページで定義しておき、その関数を使用するページの先頭で cfinclude タグを使用することができます。
- 任意のページの、呼び出しページからアクセス可能な特定のスコープ。UDF スコープの詳細については、168 ページの「関数のスコープの指定」を参照してください。
- "Application.cfc" または "Application.cfm" ページ。詳細については、231 ページの「ColdFusion アプリケーションの設計と最適化」を参照してください。

関数を定義する場所の選択方法については、168 ページの「"Application.cfm" および関数インクルードファイルの使用」および 168 ページの「関数のスコープの指定」を参照してください。

CFScript による関数の作成について

CFScript で関数を定義するには、function ステートメントを使用します。CFScript での関数の定義には、次の特徴と制限があります。

- JavaScript や多くのプログラミング言語に近いシンタックスで関数を定義できます。
- CFScript は、式や条件操作などのビジネスロジックを効率的に記述できます。
- CFScript 関数の定義には、CFML タグを含めることはできません。

2 の累乗を返す関数を CFScript で定義する例を次に示します。

```
<cfscript>
function twoPower(exponent) {
    return 2^exponent;
}
</cfscript>
```

CFScript を使用した関数の定義方法の詳細については、151 ページの「CFScript での関数の定義」を参照してください。

CFScript での関数の定義

CFScript の関数は、JavaScript の関数に似た方法で定義できます。単一の CFScript ブロックで複数の関数を定義できます。

注意：CFScript の使用方法の詳細については、98 ページの「CFML スクリプト言語による ColdFusion ページの拡張」を参照してください。

CFScript 関数の定義のシンタックス

CFScript 関数を定義するシンタックスは次のとおりです。

```
function functionName( [argName1[, argName2...]] )
    {
        CFScript Statements
    }
```

次の表に、関数の変数を示します。

関数の変数	説明
functionName	関数の名前。標準の ColdFusion 関数の名前や "cf" で始まる名前は使用できません。同じ名前を使用して 2 つの異なる関数を定義することはできません。関数名にピリオドを含めることはできません。
argName1...	関数で必要な引数の名前。関数に渡す引数の数は、関数定義冒頭の括弧内にある引数の数以上である必要があります。必須の引数を呼び出しページで省略した場合は、引数の数が一致しないことを示すエラーが発生します。

関数定義の本文は、空であっても中括弧で囲む必要があります。

次の 2 つのステートメントは、関数定義内でのみ使用できます。

ステートメント	説明
varvariableName = expression;	関数のローカル変数 (関数の変数) を作成および初期化します。この変数は、関数の内部でのみ意味を持ちます。同じ関数を次に呼び出しても、前回の値は保存されていません。関数本文の中では、他のスコープに同名の変数があっても、関数の変数が優先されます。関数の変数にはスコープ識別子の接頭辞を付けません。変数名にピリオドを含めることはできません。変数の初期値は、expression の評価結果です。expression には、定数や別の UDF などで構成される、任意の有効な ColdFusion 式を指定できます。 すべての var ステートメントは、関数宣言の先頭 (その他のすべてのステートメントの前) に配置する必要があります。変数を宣言するときには、初期化を行います。引数と同じ名前の変数を定義することはできません。 1 つの var ステートメントで初期化できる変数は 1 つのみです。 関数内でのみ使用する変数 (ループカウンタや一時変数も含む) は、すべて var ステートメントを使用して初期化してください。
returnexpression;	expression (変数など) を評価し、関数を呼び出したページにその値を返し、関数を終了します。ColdFusion の任意の変数型を返すことができます。

簡単な CFScript の例

次の関数の例では、2 つの引数を加算して、その結果を返します。

```
 <cfscript>
function Sum(a,b) {
    var sum = a + b;
    return sum;
}
</cfscript>
```

この例では、1 行で関数の変数を宣言し、式を使用して戻り値を設定しています。次のようにこの関数を単純化して、関数の変数を使用しないようにすることも可能です。

```
 function MySum(a,b) {Return a + b;}
```

関数定義の本文は、ステートメントが 1 つであっても、中括弧で囲む必要があります。

注意：関数の引数は、関数のローカルスコープにはコピーされません。スコープが指定されていない変数が呼び出された場合は、最初に引数スコープ、次にローカルスコープが検索されます。

タグによる関数の作成について

CFML で UDF を定義するには、cffunction タグを使用します。cffunction タグのシンタックスには、次の特徴と制限があります。

- このシンタックスは、スクリプティングやプログラミングの経験はないが CFML や HTML の経験はある開発者にとってわかりやすいものになっています。
- 任意の ColdFusion タグを関数定義に含めることができます。これによって、たとえば、データベースにアクセスする関数を作成できます。
- CFScript コードを関数定義に埋め込むことができます。
- cffunction タグの属性を使用すれば、関数を実行できるユーザーを制限したり、関数にアクセス可能な方法を指定したりすることができます。

次のコードでは、cffunction タグを使用して、累乗の関数を定義します。

```
<cffunction name="twoPower" output=True>
    <cfargument name="exponent">
    <cfreturn 2^exponent>
</cffunction>
```

cffunction タグを使用した関数の定義方法の詳細については、153 ページの「cffunction タグによる関数の定義」を参照してください。

cffunction タグによる関数の定義

cffunction タグおよび cfargument タグを使用すると、CFScript を使用しなくても CFML で関数を定義できます。

ColdFusion コンポーネントについては、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。cffunction タグの詳細については、『CFML リファレンス』を参照してください。

cffunction タグの関数定義の形式

cffunction タグでは、次の形式を使用して関数を定義します。

```
<cffunction name="functionName" [returnType="type" roles="roleList"
        access="accessType" output="Boolean"]>
    <cfargument name="argumentName" [Type="type" required="Boolean"
        default="defaultValue">]
    <!--- Function body code goes here. --->
    <cfreturn expression>
</cffunction>
```

角括弧 ([]) はオプションの引数を示します。cfargument タグの数に制限はありません。

cffunction タグでは、この関数を呼び出すときに使用する名前を指定します。次の表に示すように、オプションで関数の他の特性を指定することもできます。

属性	説明
name	関数名。
returnType	(オプション) 関数が返すデータの型。有効な標準の型名は、any、array、binary、Boolean、date、guid、numeric、query、string、struct、uuid、variableName、xml、および void です。他の名前を指定する場合は、その名前を持つ ColdFusion コンポーネントを引数として使用する必要があります。 ここで指定した型に自動変換できない型のデータを関数が返そうとすると、エラーが発生します。たとえば、関数が数値の計算結果を返す場合、文字列または数値の returnType 属性は有効ですが、配列は無効です。
roles	(オプション) このメソッドを実行できるセキュリティロールのカンマ区切りのリスト。この属性を省略すると、この関数へのユーザーのアクセスが制限されません。 この属性を使用すると、現在のユーザーが cfloginuser タグを使用してログインしており、この属性で指定されているいずれかのロールのメンバーである場合にのみ、関数が実行されます。それ以外の場合は、承認されていないアクセスの例外が発生します。ユーザーセキュリティの詳細については、337 ページの「アプリケーションの保護」を参照してください。
output	(オプション) 関数本文内の表示可能な出力の処理方法を決定します。 このオプションを指定しなかった場合は、関数の本文が通常の CFML として処理されます。したがって、関数定義の本文内のテキストと cfoutput タグの結果が、関数を実行するたびに表示されます。 true または yes を指定した場合は、関数の本文が cfoutput タグの中にあるものとして処理されます。変数や式をシャープ記号 (#) で囲むと、その変数の値や式の結果が表示されます。 false または no を指定した場合は、関数が cfsilent タグの中にあるものとして処理されます。関数は出力を表示しません。関数の結果を表示する処理は、その関数を呼び出したコードで行うことになります。

関数の必須引数には cfargument タグを使用する必要があります。すべての cfargument タグは、cffunction タグの本文で他の CFML コードを記述する前に使用する必要があります。したがって、cffunction 開始タグの直後に cfargument を置きます。cfargument タグは次の属性を持ちます。

属性	説明
name	引数名
type	(オプション) 引数のデータ型。関数に渡されるデータの型。有効な標準の型名は、any、array、binary、Boolean、date、guid、numeric、query、string、struct、uuid、および variableName です。他の名前を指定する場合は、その名前を持つ ColdFusion コンポーネントを引数として使用する必要があります。 ここで指定した型に自動変換できない型のデータが関数に渡された場合には、エラーが発生します。たとえば、引数の type 属性が数値の場合、配列を使用して関数を呼び出すことはできません。
required	(オプション) 引数が必須であるかどうかを指定するブール値。true に設定した場合、その引数を省略して関数を呼び出すとエラーが発生します。デフォルト値は false です。default 属性を指定した場合は、required 属性を指定する必要はありません。 関数を呼び出すときは、引数名を指定しないこともあります。したがって、cffunction を定義するときは、必須引数を定義するすべての cfargument タグを、オプション引数を定義する cfargument タグよりも前に置く必要があります。
default	(オプション) 引数に値が渡されなかった場合に使用する、オプション引数のデフォルト値。この属性を指定した場合、required 属性は無視されます。

注意：オプションの引数は、cfargument タグで定義しなくてもかまいません。この機能は、関数の引数の数が不確定な場合に役立ちます。cfargument タグで定義していないオプション引数は、Arguments スコープ配列でその位置を指定して参照できます。詳細については、159 ページの「配列としての Arguments スコープの使用」を参照してください。

ユーザー定義関数での CFML タグの使用

関数を CFScript で定義せずに cffunction タグで定義する場合の最も重要な利点は、関数内で CFML タグを使用できることです。これによって、ColdFusion タグを必要とする処理 (データベース検索など) をカプセル化することができます。また、cfoutput タグが使用できるので、簡潔なコードで呼び出しページに出力を表示できます。

注意：パフォーマンスを向上させるには、ColdFusion 関数で cfparam タグを使用しないようにします。代わりに、cfset タグを使用します。

次の関数は、従業員の部門 ID を検索して返します。この関数は、従業員 ID を表す 1 つの引数を取り、cfdocexamples の Employee テーブルから対応する部門 ID を検索します。

```
<cffunction name="getDeptID" >
    <cfargument name="empID" required="true" type="numeric">
    <cfset var cfdocexamples="">
    <cfquery dataSource="cfdocexamples" name="deptID">
        SELECT Dept_ID
        FROM Employee
        WHERE Emp_ID = #empID#
    </cfquery>
    <cfreturn deptID.Dept_ID>
</cffunction>
```

関数定義のルール

CFScript または cffunction タグを使用して定義する関数には、次のルールが適用されます。

- 関数名は一意である必要があります。既存の変数または UDF と異なる名前にする必要があります。ただし、ColdFusion 拡張セキュリティ関数名は使用できます。
- ユーザー定義関数には、CFC のビルトイン関数と同じ名前を付けることができます (CFM のビルトイン関数と同じ名前は使用できません)。
- 次の名前を使用してユーザー定義関数を作成することはできません。
  - writedump
  - writelog

- location
- throw
- trace

- 関数名の先頭に **cf** という文字は使用できません。たとえば、CF_MyFunction、cfmyFunction、cfxMyFunction は無効な UDF 名です。
- 関数の再定義や、オーバーロードはできません。ある関数定義がアクティブである場合、同じ名前で別の関数を定義するとエラーが発生します。
- 関数の定義はネストできません。つまり、関数定義の内部で別の関数を定義することはできません。
- 関数は再帰的に定義できます。つまり、関数定義の本文でその関数自身を呼び出すことができます。
- 関数は、必ずしも値を返す必要はありません。

UDF の作成方法には、タグを使用する方法と、CFScript を使用する方法があります。どちらの方法にも、利点と欠点があります。

ユーザー定義関数の呼び出し

作成した関数は、cfoutput タグのシャープ記号 (#) の間、CFScript の中、タグ属性値の中など、式が使用できる場所であればどこでも呼び出せます。関数で別の関数を呼び出すこともできます。また、関数を別の関数の引数として使用することもできます。

UDF は、次の 2 通りの方法で呼び出すことができます。

- 名前を使用せずに位置で引数を識別する方法。これは、ビルトイン関数を呼び出す場合と同じです。
- 名前を使用して引数を識別する方法。これは、タグで属性を指定する場合と同じです。

どの関数でも、いずれかの方法を選択して呼び出すことができます。ただし、名前を使用して引数を識別する場合は、関数定義で指定されている引数名と同じ名前を使用します。名前を使用する方法と使用しない方法を混在させて関数を呼び出すことはできません。

ユーザー定義関数の例として、TotalInterest 関数を考えます。この関数は、元金、年利、およびローン期間 (月) に基づいて、ローンの支払い金額を計算します。この関数の定義については、166 ページの「ユーザー定義関数の例」を参照してください。次のように、フォームのアクションページで、引数名を使用せずに関数を呼び出すことができます。

```
 <cfoutput>
    Interest: #TotalInterest(Form.Principal, Form.Percent, Form.Months)#
</cfoutput>
```

次のように、引数名を使用して関数を呼び出すこともできます。

```
 <cfoutput>
Interest: #TotalInterest(principal=Form.Principal, annualPercent=Form.Percent,
    months=Form.Months)#
</cfoutput>
```

関数での引数および変数の操作

推奨する引数ネーミングスタイル

引数には、その用途がわかる名前を付けるようにしてください。たとえば、次のコードで混乱が発生することはまずありません。

```
<cfscript>
    function SumN(Addend1,Addend2)
    { return Addend1 + Addend2; }
</cfscript>
<cfset x = 10>
<cfset y = 12>
<cfoutput>#SumN(x,y)#</cfoutput>
```

これに似た次のコードでは、プログラミングエラーが発生する可能性が高くなります。

```
<cfscript>
    function SumN(x,y)
    { return x + y; }
</cfscript>
<cfset x = 10>
<cfset y = 12>
<cfoutput>#SumN(x,y)#<cfoutput>
```

引数の受け渡し

次のデータ型は、その値が関数に渡されます。

- 整数
- 実数
- 文字列 (リストを含む)
- 日付時刻オブジェクト
- 配列

したがって、関数の定義とそれを呼び出すコードが同じ ColdFusion ページにある場合でも、関数内で引数が変更されたことで、関数呼び出しに使用した変数が変更されることはありません。

クエリー、構造体、COM オブジェクトなどの外部オブジェクトは、その参照が関数に渡されます。したがって、関数内でこれらの引数が変更されると、呼び出しコード内の変数の値も変更されます。

引数の受け渡しの影響の例については、156 ページの「複合データの受け渡し」を参照してください。

複合データの受け渡し

構造体、クエリー、および COM オブジェクトなどの複合オブジェクトは、その参照がユーザー定義関数に渡されます。したがって、呼び出し側と同じデータがユーザー定義関数でも使用されます。配列は、その値がユーザー定義関数に渡されます。ユーザー定義関数は配列データの新しいコピーを受け取るので、呼び出しページの配列がユーザー定義関数によって変更されることはありません。配列は、他の複合データ型と異なる処理が必要です。

構造体、クエリー、およびオブジェクトの受け渡し

呼び出し側の構造体、クエリー、またはオブジェクトを関数で変更するには、その変数を引数として渡します。関数は呼び出し側の構造体への参照を取得するので、関数でその構造体に加えた変更は、すべて呼び出し側の変数に反映されます。呼び出し側にその構造体を返す必要はありません。関数から戻った呼び出しページでは、関数に渡した構造体変数を使用して、変更されたデータにアクセスできます。

呼び出し側の構造体、クエリー、またはオブジェクトが関数で変更されないようにしたい場合は、Duplicate 関数を使用してその複製を作成してから、それを関数に渡します。

配列の受け渡し

呼び出し側の配列を関数で変更したい場合は、関数で配列を受け取って変更し、それを関数の return ステートメントで呼び出し側に返すのが最も簡単です。呼び出し側では、引数として渡した変数に戻り値を代入します。

配列を直接渡して返す方法を次の例に示します。この例の doubleOneDArray 関数は、1 次元配列の各要素の値を 2 倍にします。

```
 <cfscript>
//Initialize some variables
//This creates a simple array.
a=ArrayNew(1);
a[1]=2;
a[2]=22;
//Define the function.
function doubleOneDArray(OneDArray) {
    var i = 0;
    for ( i = 1; i LE arrayLen(OneDArray); i = i + 1)
        { OneDArray[i] = OneDArray[i] * 2; }
    return OneDArray;
}
//Call the function.
a = doubleOneDArray(a);
</cfscript>
<cfdump var="#a#">
```

この解決方法は簡単ですが、適切でない場合もあります。

- この方法では、配列全体を 2 回コピーする必要があります。1 回は関数を呼び出すとき、もう 1 回は関数を返すときです。この方法は大きな配列の場合は非効率的であり、この関数を頻繁に呼び出すとパフォーマンスが低下する場合があります。
- ステータス変数などの他の目的に戻り値を使用できます。

return ステートメントを使用して呼び出し側に配列を返す方法が適切でない場合は、この配列を構造体内の要素として渡し、構造体内の配列値を変更します。呼び出しページでは、UDF に渡した構造体変数を使用して、変更されたデータにアクセスできます。

次のコードは、構造体内の配列を使用して前述の例を書き直した例です。この関数では、呼び出しページにステータス True を返し、構造体を使用して配列データを戻しています。

```
 <cfscript>
//Initialize some variables.
//This creates a simple array as an element in a structure.
arrayStruct=StructNew();
arrayStruct.Array=ArrayNew(1);
arrayStruct.Array[1]=2;
arrayStruct.Array[2]=22;
//Define the function.
function doubleOneDArrayS(OneDArrayStruct) {
    var i = 0;
    for ( i = 1; i LE arrayLen(OneDArrayStruct.Array); i = i + 1)
        { OneDArrayStruct.Array[i] = OneDArrayStruct.Array[i] * 2; }
    return True;
}
//Call the function.
Status = doubleOneDArrayS(arrayStruct);
WriteOutput("Status: " & Status);
</cfscript>
</br>
<cfdump var="#arrayStruct#">
```

呼び出しページと関数では、配列を参照するために同じ構造体要素名 (この場合は Array) を使用します。

Arguments スコープについて

関数のすべての引数は、固有のスコープである Arguments スコープに属しています。

Arguments スコープは、関数呼び出しが存続している間のみ存在します。関数が戻ると、このスコープとその変数は廃棄されます。

ただし、Arguments スコープが廃棄されても、関数に参照が渡された変数 (構造体やクエリーオブジェクトなど) は廃棄されません。呼び出しページで引数として使用した変数は存在し続け、関数によって引数に加えられた変更は、呼び出しページのその変数に反映されます。

Arguments は特殊なスコープで、配列または構造体として扱うことができます。この Arguments スコープの二面性を利用すると、次のような状況で引数を簡単に処理できます。

- CFScript を使用して関数を定義する。
- cffunction タグを使用して関数を定義する。
- 引数名 = 値という形式を使用して引数を渡す。
- 引数値のみを列挙して渡す。
- 関数に、宣言されていないオプションの引数を渡す。

Arguments スコープの内容

Arguments スコープとその内容には次のルールが適用されます。

- このスコープには、関数に渡されたすべての引数が含まれています。
- cffunction で関数を定義した場合は、関数に引数が渡されなくても、宣言した引数ごとにエントリの " スロット " が常に用意されます。宣言した (オプションの) 引数が渡されなかった場合、その引数に対応するスコープエントリは空になります。

  CFScript で定義した関数を呼び出す場合は、関数定義で宣言したすべての引数に値を渡します。したがって、CFScript 呼び出しの Arguments スコープに空のスロットはありません。

このルールを次の例に示します。次のように関数が宣言されているとします。

```
<cffunction name="TestFunction">
    <cfargument name="Arg1">
    <cfargument name="Arg2">
</cffunction>
```

この関数は、次の行のように、1 つの引数を使用して呼び出すことができます。

```
<cfset TestFunction(1)>
```

その結果、Arguments スコープは次のようになります。

配列として使用		**構造体として使用**	
エントリ	**値**	**エントリ**	**値**
1	1	Arg1	1
2	未定義	Arg2	未定義

この例では、スコープには引数が 2 つ定義されているので、次の関数は値 2 を返します。

```
ArrayLen(Arguments)
StructCount(Arguments)
```

ただし、Arguments スコープの 2 番目の要素の内容は未定義なので、次のテストは false を返します。

```
Isdefined("Arguments.Arg2")
testArg2 = Arguments[2]>
Isdefined("testArg2")
```

注意：IsDefined 関数では、配列要素が存在するかどうかはテストできません。配列の要素を検索するには、ArrayContains 関数を使用します。

配列としての Arguments スコープの使用

Arguments スコープを配列として参照する場合には、次のルールが適用されます。

- 名前を使用せずに引数を渡した場合、配列のインデックスは、関数呼び出しでの引数の位置を表します。
- 名前を使用して引数を渡した場合、配列のインデックスは、関数定義で引数が宣言されている位置を表します。
- 名前を使用して引数を渡し、関数定義で宣言されている引数の一部を渡さなかった場合、Arguments 配列では、渡さなかった引数に対応するインデックスが空のエントリになります。このルールは、cffunction タグで作成した関数にのみ適用されます。
- 名前を使用して、関数定義で宣言されていないオプション引数を渡した場合、その引数の配列インデックスは、次の値の合計になります。
  - 関数定義で名前を使用して宣言されている引数の数
  - その関数に渡した、関数定義で名前が宣言されていない引数の中での、そのオプション引数の順番

  ただし、このように引数名を使用するのは、よいプログラミングスタイルとはいえません。関数を呼び出すときに常に同じオプション引数名を使用するとは限らないからです。

これらのルールを次の例に示します。この例では、関数の Arguments 配列の内容を表示する簡単な関数を定義し、さまざまな引数の組み合わせを使用して関数を呼び出します。

```
<cffunction name="TestFunction" >
    <cfargument name="Arg1">
    <cfargument name="Arg2">
    <cfloop index="i" from="1" to="#ArrayLen(Arguments)#">
        <cfoutput>Argument #i#: #Arguments[i]#<br></cfoutput>
    </cfloop>
</cffunction>

<strong>One Unnamed argument</strong><br>
<cfset TestFunction(1)>
<strong>Two Unnamed arguments</strong><br>
<cfset TestFunction(1, 2)>
<strong>Three Unnamed arguments</strong><br>
<cfset TestFunction(1, 2, 3)>
<strong>Arg1:</strong><br>
<cfset TestFunction(Arg1=8)>
<strong>Arg2:</strong><br>
<cfset TestFunction(Arg2=9)>
<strong>Arg1=8, Arg2=9:</strong><br>
<cfset TestFunction(Arg1=8, Arg2=9)>
<strong>Arg2=6, Arg1=7</strong><br>
<cfset TestFunction(Arg2=6, Arg1=7)>
<strong>Arg1=8, Arg2=9, Arg3=10:</strong><br>
<cfset TestFunction(Arg1=8, Arg2=9, Arg3=10)>
<strong>Arg2=6, Arg3=99, Arg1=7</strong><br>
<cfset TestFunction(Arg2=6, Arg3=99, Arg1=7)>
```

注意：Arguments スコープは配列として使用できますが、IsArray(Arguments) 関数は常に false を返し、cfdump タグはスコープを構造体として表示します。

構造体としての Arguments スコープの使用

Arguments スコープを構造体として参照する場合には、次のルールが適用されます。

- 引数名を構造体のキーとして使用します。たとえば、関数定義に Principal 引数が含まれている場合は、Arguments.Principal のようにして引数を参照します。

  次のルールも適用されますが、これらのルールを使用したコードは作成しないでください。プログラムをわかりやすくするために、関数定義で名前を使用して宣言した引数のみに Arguments 構造体を使用してください。関数定義で宣言していないオプション引数には、Arguments スコープを配列として使用してください。

- 関数定義でオプション引数に名前を付けておらず、関数呼び出しでその名前を指定している場合は、関数呼び出しで指定した名前が使用できます。たとえば、名前のないオプションの引数があり、その引数に myOptArg という名前を指定して関数を呼び出した場合は、関数の本文でその引数を Arguments.myOptArg として参照できます。ただし、これはよいプログラミングスタイルとはいえません。関数定義の内容が、その関数を呼び出すコードで使用されている名前に依存してしまうからです。

CFScript での Arguments スコープの使用

関数は、オプションの引数を取る場合があります。オプション引数は、関数を呼び出すときに必ずしも指定する必要はありません。関数に渡された引数の数を調べるには、次の関数を使用します。

```
ArrayLen(Arguments)
```

CFScript を使用して関数を定義する場合、その関数内でオプション引数を取得するには、Arguments スコープを使用する必要があります。たとえば、次の SumN 関数は、2 つ以上の数値を加算します。この関数は 2 つの必須引数を取り、任意の数の追加のオプション引数をサポートしています。初めの 2 つの引数 (必須) は、Arg1 および Arg2、または Arguments[[1] および Arguments[2] として参照できます。3 番目、4 番目などの追加のオプション引数は、Arguments[3]、Arguments[4] などのようにしてアクセスします。

```
function SumN(Arg1,Arg2) {
    var arg_count = ArrayLen(Arguments);
    var sum = 0;
    var i = 0;
    for( i = 1 ; i LTE arg_count; i = i + 1 )
    {
        sum = sum + Arguments[i];
    }
    return sum;
}
```

この関数では、次のどの関数呼び出しも有効です。

```
SumN(Value1, Value2)
SumN(Value1, Value2, Value3)
SumN(Value1, Value2, Value3, Value4)
```

など。

このコードでは、Arg1 や Arg2 という引数変数を直接使用していません。これは、これらの値が常に Arguments 配列の最初の 2 つの要素であり、配列を順にループしたほうが処理が簡単だからです。関数定義で Arg1 および Arg2 を使用すると、関数に引数が 1 つしか渡されなかった場合や 1 つも渡されなかった場合に、エラーが発生します。

注意：関数本文で必須引数を参照するときは、引数名を使用する方法と、Arguments スコープの配列または構造体で位置を指定する方法を混在させないでください。2 つの方法を混在させると、混乱やエラーが発生しやすくなります。

cffunction 定義での Arguments スコープの使用

cffunction タグを使用して関数を定義するときは、cfargument タグですべての引数名が宣言されている場合は、名前を使用して引数を直接参照するのが一般的です。Arguments スコープ識別子を使用する場合は、157 ページの「Arguments スコープについて」に示したルールに従ってください。

CFScript で定義した関数内で Arguments スコープを使用する方法の詳細については、160 ページの「CFScript での Arguments スコープの使用」を参照してください。

関数内変数

関数は、Arguments スコープだけでなく、関数内でのみ存続する変数を使用することができます。これらの変数は、同じ関数を次に呼び出しても、前回の値は保存されていません。関数が終了すると、このスコープ内の変数はすべて削除されます。

CFScript で関数内変数を作成するには、var ステートメントを使用します。他の変数と異なり、関数内変数にスコープ名の接頭辞を付ける必要はありません。

関数内変数の使用

CFScript の UDF で、関数固有の変数 (ループインデックスや、関数呼び出しの間でのみ必要な一時変数など) を宣言するときは、常に var ステートメントを使用してください。これによって、これらの変数が関数内でのみ使用可能になり、他のスコープの変数名と競合することが避けられます。呼び出しページに同じ名前の変数があっても、この 2 つの変数は独立しており、互いに影響を与えることはありません。

たとえば、ColdFusion ページの cfloop タグでインデックス変数 i を使用しており、タグ本文で呼び出している CFScript の UDF でも関数内変数としてループインデックス i を使用している場合、呼び出しページのループインデックスの値が UDF によって変更されることはありません。同様に、UDF のインデックスが呼び出しページによって変更されることはありません。したがって、この cfloop タグの本文でこの関数を呼び出しても問題はありません。

一般に、CFScript の UDF 内でのみ使用する変数は、関数の引数や共有スコープ変数を除き、すべて var ステートメントを使用して宣言します。ただし、たとえば関数が呼び出されるたびにインクリメントされるカウンタなど、複数の関数呼び出しにまたがって変数値を保持する必要がある場合は、別のスコープを使用します。

呼び出し側の変数の参照

呼び出しページで使用可能な任意の変数を、そのページから呼び出された関数で使用したり変更したりすることができます。使用可能な変数としては、呼び出し側の Variables (ローカル) スコープの変数が含まれます。関数側では、その関数が呼び出しページに含まれているかのように、呼び出し側の変数にアクセスすることができます。たとえば、呼び出しページに Customer_name というローカル変数があり、関数スコープに Customer_name という変数がない場合は、関数で Customer_name または Variables.Customer_name という参照 (後者のほうがよいコーディングスタイルです) を使用すれば、その変数の読み取りや変更が行えます。同様に、関数内で作成したローカル変数を、関数呼び出しの後に、呼び出しページの任意の場所で使用することができます。関数を呼び出す前に変数を使用することはできません。

ただし、通常は、呼び出し側の変数を関数内で直接使用するのは避けてください。呼び出し側の変数を使用すると、呼び出し側との間に依存関係ができることになります。呼び出し側のコードでは、関数で使用されている変数名と同一の変数名を常に使用する必要があります。多くのページからこの関数を呼び出す場合、これを守るのは容易ではありません。

呼び出し側と関数の間でデータのやり取りを行うときは、これらの問題を避けるため、関数の引数と戻り値のみを使用してください。呼び出しページの変数を関数内で直接参照しないでください。これによって、呼び出しコードの変数を気にすることなく、アプリケーション内 (または複数のアプリケーション) の任意の場所からその関数が使用できるようになります。

ただし、実際のプログラミングでは、この推奨事項にも例外があります。たとえば、次のことが可能です。

- Application スコープや Session スコープのカウンタ変数などの、共有スコープ変数を使用できます。
- Request スコープを使用して、関数内で使用する変数を保存できます。詳細については、169 ページの「スタティック変数および定数での Request スコープの使用」を参照してください。
- 呼び出し側と関数側で常に同じ変数名を使用するように徹底できるのであれば、呼び出し側のデータを直接操作する文脈依存型の関数を作成できます。

注意：呼び出し側の単純変数 (関数に参照が渡されない変数) を関数で直接変更する必要がある場合は、構造体の中にその変数を格納して、その構造体を引数として関数に渡します。

引数の使用

関数の引数名は、呼び出し側の変数名と同じにすることができますが (値は同じにはなりません)、コードがわかりにくくなるので、このようなことは行わないでください。

引数の存続期間については次のルールが適用されます。

- 単純変数および配列の引数はその値が渡されるので、その名前と値は関数が実行されている間のみ存在します。
- 構造体、クエリー、COM オブジェクトなどのオブジェクトはその参照が渡されるので、引数の名前は関数が実行されている間のみ存在しますが、データは関数が戻った後でも存続し、呼び出し側の変数名を使用してアクセスできます。呼び出し側の変数と引数には異なる名前を付けることができます。

注意：関数内変数と同じ名前を持つ別のスコープの変数を関数で使用する必要がある場合は、その変数に Variables や Form などのスコープ識別子の接頭辞を付けてください。ただし、他のスコープの変数を関数内で直接使用するのは、多くの場合よいコーディングスタイルではありません。

UDF でのエラー処理

ColdFusion では、UDF でのエラー次の方法で処理できます。

- 関数内でエラーメッセージを直接表示する。
- 関数のステータス情報を呼び出しページに返す。
- try/catch または cftry/cfcatch ブロック、cfthrow および cfrethrow タグを使用して例外を処理および生成する。

どの方法を使用するかは、関数およびアプリケーションの環境や、プログラマのプログラミングスタイルによって異なります。ただし、2 番目または 3 番目の方法か、この 2 つの組み合わせを使用することをお勧めします。

エラーメッセージの表示

関数でエラーがないかどうかをテストし、WriteOutput 関数を使用してエラーメッセージをユーザーに直接表示できます。この方法は、単純な入力エラーをユーザーにすぐフィードバックするときに便利です。この方法は、単独で使用したり、他の 2 つのエラー処理方法と組み合わせて使用したりできます。

たとえば、次の変更を加えた "Hello world" 関数は、フォームに名前が入力されていない場合にエラーメッセージを表示します。

```
 <cfform method="POST" action="#CGI.script_name#">
    <p>Enter your Name: 
    <input name="name" type="text" hspace="30" maxlength="30">
    <input type="Submit" name="submit" value="OK">
</cfform>
<cfscript>
    function HelloFriend(Name) {
        if (Name is "") WriteOutput("You forgot your name!");
        else WriteOutput("Hello " & name &"!");
        return "";
    }
    if (IsDefined("Form.submit")) HelloFriend(Form.name);
</cfscript>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明
<cfform method="POST" action="#CGI.script_name#"> <p>Enter your Name: <input name="name" type="text" hspace="30" maxlength="30"> <input type="Submit" name="submit" value="OK"> </cfform>	名前の入力を要求する簡単なフォームを作成します。 script_name CGI 変数を使用して、URL を指定せずにこのページに渡します。 名前が入力されなかった場合は、名前フィールドとして空の文字列が渡されます。

<cfscript>
function HelloFriend(Name) {
if (Name is "") WriteOutput("You forgot your name!");
else WriteOutput("Hello " & name &"!"); return "";
}
if (IsDefined("Form.submit"))
HelloFriend(Form.name);
</cfscript>
	"Hello name!" を表示する関数を定義します。最初に、引数が空の文字列であるかどうかを確認します。空の文字列である場合は、エラーメッセージを表示します。
	それ以外の場合は、挨拶を表示します。
	空の文字列を返します。呼び出し側では戻り値を使用しません。if または else ステートメントの本文は、1 つのステートメントであるため、中括弧で囲む必要はありません。
	HelloFriend 関数が呼び出されるのは、フォームを送信してこのページを呼び出した場合のみです。それ以外の場合は、フォームが表示されるのみです。

ステータス情報の提供

関数で修正作業を行えないなど、関数でエラーを直接処理できないか、処理すべきではない場合があります。このような場合は、関数から呼び出しページに情報を返すことができます。呼び出しページは、エラー情報を処理して正常に機能できる必要があります。

ステータス情報を提供するメカニズムとしては、次のようなものがあります。

- 戻り値を使用して、関数のステータスのみを示します。たとえば、ブール値で成功か失敗かを示します。または、ステータスコードを返すこともできます。たとえば、成功した場合は 1 を返し、失敗した場合はさまざまなエラータイプを規定の番号で返すことができます。この方法を使用する場合、成功したときの処理結果は、呼び出し側の変数を設定することで渡す必要があります。
- 戻り値の変数ではなく、呼び出し側が使用できるステータス変数に、成功、失敗、またはその他の失敗情報を設定します。この方法を使用すると、関数から呼び出し側に直接結果を返すことができます。この方法では、戻り値と構造体の引数のみを使用してステータスを呼び出し側に返します。

これらの方法にはバリエーションがあり、それぞれに利点と欠点があります。どの方法を使用するかは、関数のタイプ、関数を使用するアプリケーション、プログラマのコーディングスタイルによって異なります。

166 ページの「ユーザー定義関数の例」で使用した関数を修正した次の例では、ステータス変数を使用する方法のバリエーションを使用しています。この例では、エラー情報は次の 2 つの形式で提供されます。

- エラーが発生した場合は、利息の代わりに -1 を返します。ローンでマイナスの利息を支払うことはないので、この値でエラーを示すことができます。
- さらに、エラー説明用の変数が含まれている構造体に、エラーメッセージが書き込まれます。メッセージは構造体の中にあるので、呼び出しページと関数の両方で使用できます。

TotalInterest 関数

エラーを処理できるように変更した TotalInterest 関数は、次のようになります。166 ページの「ユーザー定義関数の例」の例から変更されたコードは、太字で示されています。

```
<cfscript>
function TotalInterest(principal, annualPercent, months, status) {
    Var years = 0;
    Var interestRate = 0;
    Var totalInterest = 0;
    principal = trim(principal);
    principal = REReplace(principal,"[\$,]","","ALL");
    annualPercent = Replace(annualPercent,"%","","ALL");
    if ((principal LE 0) OR (annualPercent LE 0) OR (months LE 0)) {
        Status.errorMsg = "All values must be greater than 0";
        Return -1;
    }
    interestRate = annualPercent / 100;
    years = months / 12;
    totalInterest = principal*(((1+ interestRate)^years)-1);
    Return DollarFormat(totalInterest);
}
</cfscript>
```

コードの説明

前のバージョンから変更または追加されたコードについて、次の表で説明します。最初のコードの説明については、166 ページの「ユーザー定義関数の例」を参照してください。

コード	説明
`function TotalInterest(principal, annualPercent, months, status)`	この関数は、新しいステータス構造体を引数として取るようになりました。関数で行った変更が呼び出し側に反映されるように、構造体を使用しています。

`if ((principal LE 0) OR (annualPercent LE 0) OR (months LE 0)) {`
`Status.errorMsg = "All values must be greater than 0";`
`Return -1;`
`}`
	元金、利率、および返済期間がすべて 0 よりも大きい数であることを確認します。
	そうでない場合は、Status 構造体の errorMsg キー (唯一のキー) に、説明用の文字列を設定します。呼び出し側に -1 を返し、それ以上の処理を行わずに終了します。

関数の呼び出し

この関数を呼び出すコードは、次のようになります。166 ページの「ユーザー定義関数の例」の例から変更されたコードは、太字で示されています。

```
<cfset status = StructNew()>
<cfset myInterest = TotalInterest(Form.Principal,
    Form.AnnualPercent,Form.Months, status)>
<cfif myInterest EQ -1>
    <cfoutput>
        ERROR: #status.errorMsg#<br>
    </cfoutput>
<cfelse>
    <cfoutput>
        Loan amount: #Form.Principal#<br>
        Annual percentage rate:
            #Form.AnnualPercent#<br>
        Loan duration: #Form.Months# months<br>
        TOTAL INTEREST: #myInterest#<br>
    </cfoutput>
</cfif>
```

コードの説明

変更または追加されたコードについて、次の表で説明します。

コード	説明
`<cfset status = StructNew()>`	関数ステータスを保持する構造体を作成します。
`<cfset myInterest = TotalInterest (Form.Principal, Form.AnnualPercent, Form.Months, status)>`	関数を呼び出します。ここでは、関数はステータス変数を含む 4 つの引数を必要とします。
`<cfif myInterest EQ -1> <cfoutput> ERROR: #status.errorMsg#  </cfoutput>`	関数が -1 を返した場合は、エラーが発生しています。status.errorMsg 構造体キーに格納されているメッセージを表示します。
`<cfelse> <cfoutput> Loan amount: #Form.Principal#  Annual percentage rate: #Form.AnnualPercent#  Loan duration: #Form.Months# months  TOTAL INTEREST: #myInterst#  </cfoutput> </cfif>`	関数が -1 を返さない場合は、利息を返します。入力値および関数の戻り値を表示します。

例外の使用

CFScript で作成する UDF では、try および catch ステートメントを使用して例外を処理できます。cffunction タグで作成する UDF では、cftry、cfcatch、cfthrow、および cfrethrow タグを使用できます。例外を利用する方法は、標準的なプログラミング言語の多くの関数でエラー処理方法として採用されている方法と同じです。例外を利用すると、エラーを効果的に処理できます。特に、例外を使用すると、関数のコードとエラー処理コードを分離することができます。また、テストや分岐を行う必要がないので、他の方法より実行時の効率が向上します。

例外を利用する方法は、次の 2 つの段階から構成されます。

- UDF コードの実行によって生成された例外を処理する段階
- 無効なデータや、処理を継続するとエラーが発生する状況を UDF が検出したときに、例外を生成する段階

UDF での例外処理

UDF でも、他の ColdFusion アプリケーションで try/catch ブロックを使用するのと同じ状況で、try/catch ブロックを使用してください。該当する状況としては、外部リソース (Java、COM、または CORBA のオブジェクトや、データベース、ファイルなど) を関数で使用する場合が一般的です。アプリケーションデータが無効である状況は、例外として処理するのではなく、できる限りその状況を回避してください。たとえば、あるフォームフィールドを除数として使用している場合は、ゼロによる除算を例外として処理するのではなく、ユーザーがそのフィールドにゼロを入力しないようにします。

例外が検出された場合、関数は次のいずれかの方法を使用して例外を処理できます。

- 回復可能なエラー (たとえば、再試行によって解決できるデータベースタイムアウトなど) である場合は、問題の解決を試みます。
- 162 ページの「エラーメッセージの表示」の説明に従って、メッセージを表示します。
- 163 ページの「ステータス情報の提供」の説明に従って、エラーステータスを返します。
- cffunction タグで定義されている UDF の場合は、呼び出し側の ColdFusion ページで検出されるように、カスタム例外を投げるか、再度例外を投げます。例外を投げる方法と、例外を再度投げる方法の詳細については、287 ページの「ColdFusion タグでのランタイム例外の処理」を参照してください。

UDF での例外の生成

cffunction タグを使用して関数を定義する場合は、cfthrow タグおよび cfrethrow タグを使用して、関数を呼び出したページにエラーを投げることができます。UDF でエラーが検出された場合はいつでも、メッセージを表示したりエラーステータスを返したりする代わりに、この方法を使用できます。たとえば次のコードは、163 ページの「ステータス情報の提供」の例を書き直したもので、cffunction タグおよび CFML を使用して、いずれかのフォームの値が正の値でない場合に例外を投げて処理します。

無効なデータを識別して例外を投げる行は太字で示しています。他の行は、前の例のコードと同じです。ただし、不要な文字を削除するコードは、エラーチェックコードの前に置く必要があります。

```
<cffunction name="TotalInterest">
    <cfargument name="principal" required="Yes">
    <cfargument name="annualPercent" required="Yes">
    <cfargument name="months" required="Yes">
    <cfset var years = 0>
    <cfset var interestRate = 0>
    <cfset var totalInterest = 0>

    <cfset principal = trim(principal)>
    <cfset principal = REReplace(principal,"[\$,]","","ALL")>
    <cfset annualPercent = Replace(annualPercent,"%","","ALL")>

    <cfif ((principal LE 0) OR (annualPercent LE 0) OR (months LE 0))>
        <cfthrow type="InvalidData" message="All values must be greater than 0.">
    </cfif>

    <cfset interestRate = annualPercent / 100>
    <cfset years = months / 12>
    <cfset totalInterest = principal*
            (((1+ interestRate)^years)-1)>
    <cfreturn DollarFormat(totalInterest)>
</cffunction>
```

関数を呼び出して例外を処理するコードは、次のようになります。変更した行は太字で示しています。

```
<cftry>
    <cfset status = StructNew()>
    <cfset myInterest = TotalInterest(Form.Principal, Form.AnnualPercent,
        Form.Months, status)>
    <cfoutput>
        Loan amount: #Form.Principal#<br>
        Annual percentage rate: #Form.AnnualPercent#<br>
        Loan duration: #Form.Months# months<br>
        TOTAL INTEREST: #myInterest#<br>
    </cfoutput>
<cfcatch type="InvalidData">
    <cfoutput>
        #cfcatch.message#<br>
    </cfoutput>
</cfcatch>
```

</cftry>

ユーザー定義関数の例

次の単純な関数は、元金、年利、およびローン期間 (月) に基づいて、その期間に支払う利息の総額を返します。オプションで、利率にはパーセント記号を、また元金にはドル記号とカンマ区切り文字を使用できます。

次のように TotalInterest 関数をフォームのアクションページの cfoutput タグ内で使用できます。

```
<cfoutput>
    Loan amount: #Form.Principal#<br>
    Annual percentage rate: #Form.AnnualPercent#<br>
    Loan duration: #Form.Months# months<br>
    TOTAL INTEREST: #TotalInterest(Form.Principal, Form.AnnualPercent, Form.Months)#<br>
</cfoutput>
```

CFScript を使用した関数の定義

```
 <cfscript>
function TotalInterest(principal, annualPercent, months) {
    Var years = 0;
    Var interestRate = 0;
    Var totalInterest = 0;
    principal = trim(principal);
    principal = REReplace(principal,"[\$,]","","ALL");
    annualPercent = Replace(annualPercent,"%","","ALL");
    interestRate = annualPercent / 100;
    years = months / 12;
    totalInterest = principal*(((1+ interestRate)^years)-1);
    Return DollarFormat(totalInterest);
}
</cfscript>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明
`function TotalInterest(principal, annualPercent, months) {`	TotalInterest 関数の定義を開始します。元金、年利、およびローン期間 (月) の 3 つの変数が必要です。

`Var years = 0;`
`Var interestRate = 0;`
`Var totalInterest = 0;`
	この関数で使用する中間変数を宣言し、これらを 0 に初期化します。すべての var ステートメントは、他のすべての関数コードの前に配置する必要があります。

`principal = trim(principal);`
`principal = REReplace(principal,"[\$,]","","ALL");`
`annualPercent = Replace(annualPercent,"%","","ALL");`
`interestRate = annualPercent / 100;`
`years = months / 12;`
	principal 引数から前後のスペースを削除します。
	principal 引数からドル記号 ($) やカンマ (,) 文字を削除して、数値を取得します。
	annualPercent 引数からパーセント記号 (%) を削除して数値を取得し、パーセントの値を 100 で割って利率を取得します。
	ローンを月単位から年単位に変換します。

`totalInterest = principal*`
`(((1+ interestRate)^years)-1);`
`Return DollarFormat(totalInterest);`
`}`
	支払利息の合計を計算します。Return ステートメントで値を計算することもできますが、この例では中間的な totalInterest 変数を使用してコードを読みやすくしています。US 通貨の文字列として書式設定された結果を返します。
	関数の定義を終了します。

cffunction タグを使用した関数の定義

次のコードは、CFScript ステートメントを同等の CFML タグに置き換えます。

```
<cffunction name="TotalInterest">
    <cfargument name="principal" required="Yes">
    <cfargument name="annualPercent" required="Yes">
    <cfargument name="months" required="Yes">
    <cfset var years = 0>
    <cfset var interestRate = 0>
    <cfset var totalInterest = 0>
    <cfset principal = trim(principal)>
    <cfset principal = REReplace(principal,"[\$,]","","ALL")>
    <cfset annualPercent = Replace(annualPercent,"%","","ALL")>
    <cfset interestRate = annualPercent / 100>
    <cfset years = months / 12>
    <cfset totalInterest = principal*
            (((1+ interestRate)^years)-1)>
    <cfreturn DollarFormat(totalInterest)>
</cffunction>
```

UDF の効果的な使用方法

ここでは、ユーザー定義関数をより効果的に使用する方法について説明します。

ColdFusion コンポーネントでの関数の使用

ほとんどの場合、UDF を最も効果的に使用する方法は、CFC 内で使用することです。CFC の詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

"Application.cfm" および関数インクルードファイルの使用

ColdFusion ページで UDF を使用するには、次の方法があります。

- 少数の UDF を継続して呼び出す場合は、それらを "Application.cfm" ページで定義します。
- 少数のページでのみ UDF を呼び出す場合は、"Application.cfm" では定義しません。
- 大量の UDF を使用する場合は、UDF を定義するための専用の ColdFusion ページを用意して、それらを定義します。UDF 定義ページは、UDF を呼び出す任意のページでインクルードできます。

次のセクションでは、ColdFusion ページで UDF を使用するその他の方法について説明します。

関数のスコープの指定

ユーザー定義関数の名前は、基本的には ColdFusion の変数です。ColdFusion の変数はデータの名前です。関数の名前は、CFML のコードセグメントの名前 (参照) です。したがって、変数と同様に、関数もスコープに属しています。

関数とスコープについて

ColdFusion 変数と同様に、UDF もスコープ内に存在します。

- UDF を定義すると、ColdFusion によって Variables スコープに配置されます。
- 変数をスコープに割り当てるのと同様に、新しいスコープ内の名前に UDF を代入することによって、UDF をスコープに割り当てることができます。たとえば次の行では、MyFunc という UDF を Request スコープに割り当てます。

  ```
  <cfset Request.MyFunc = Variables.MyFunc>
  ```

  これで、Request.MyFunc を呼び出すことによって、Request スコープ内の任意のページからこの関数を使用できます。

関数スコープの選択

次の表に、各関数スコープの利点と欠点を示します。

スコープ	注意事項
Application	関数を、アプリケーションのすべての呼び出しで使用可能にします。Application スコープ内の UDF へのアクセスはマルチスレッドです。したがって、UDF の複数のコピーを同時に実行することができます。
Request	現在の HTTP リクエストが存続している間、関数を使用可能にします。これには、すべてのカスタムタグ内やネストされたカスタムタグ内などが含まれます。このスコープは、ページ内だけでなく、そのページが呼び出すカスタムタグ内、またはネストされたカスタムタグ内で関数を使用する場合に役立ちます。
Server	その関数を、1 台のサーバー上のすべてのページで使用できるようにします。ほとんどの場合、このスコープは最適な選択ではありません。クラスタ化されたシステムでは、関数が 1 台のサーバーで使用可能になるだけであり、さらにその関数を使用するすべてのコードが cflock ブロック内に存在する必要があるからです。
Session	現在のユーザーセッションの間、関数をすべてのページで使用できるようにします。このスコープには、Application スコープを上回る重要な利点はありません。

Request スコープの使用

次の方法を使用すれば、アプリケーションページやカスタムタグで使用する関数を効果的に管理できます。

1. 関数定義ページで、関数を定義します。
2. 関数ページで、関数を Request スコープに割り当てます。
3. cfinclude タグを使用して、関数定義ページをアプリケーションページにインクルードします。カスタムタグページではインクルードしません。
4. 常に Request スコープを使用して、関数を呼び出します。

この方法を使用すれば、リクエストごとに一度だけ関数をインクルードすればよく、そのリクエストが存続している間それらの関数を使用できます。たとえば、"myFuncs.cfm" ページを作成し、次のようなシンタックスを使用して関数を定義し、Request スコープに割り当てます。

```
function MyFunc1(Argument1, Argument2)
{ Function definition goes here }
Request.MyFunc1 = MyFunc1
```

アプリケーションページで "myFuncs.cfm" ページをインクルードします。

```
<cfinclude template="myfuncs.cfm">
```

アプリケーションページおよびすべてのカスタムタグ (およびネストされたカスタムタグ) では、次のようにして関数を呼び出します。

```
Request.MyFunc1(Value1, Value2)
```

スタティック変数および定数での Request スコープの使用

161 ページの「呼び出し側の変数の参照」で説明したルールには例外があります。ここでは、関数で Request スコープ内に変数を定義します。ただし、これは次のような状況が発生した場合の解決策の一例です。

- 関数で大量の変数を初期化する場合
- 変数に次のいずれかの特性がある場合
  - 変数がスタティックであることが必要な場合。関数内でのみ使用される変数で、その値を関数で変更でき、複数の関数呼び出しにまたがってその値を保持する必要がある場合
  - 変数が名前付き定数である場合。つまり、変数の値が固定されている場合。
- アプリケーションページ (および任意のカスタムタグ) で、その関数を複数回呼び出す場合
- 変数名がその関数以外で使用されない場合

これらの状況では、関数の変数を Function スコープではなく Request スコープで定義することで、効率を向上させ、処理時間を節約できます。関数は Request スコープの変数についてテストし、存在しない場合はそれらを初期化します。その後の呼び出しでは変数が存在するので、関数は変数を初期化しません。

www.cflib.org から入手できる、Ben Forta の作成した NumberAsString 関数では、このテクニックが利用されています。

関数の引数としての関数名の使用

関数名は ColdFusion の変数なので、関数の名前を別の関数に引数として渡すことができます。このテクニックを使用すると、ある関数で別の関数をコンポーネントとして使用することができます。たとえば、ある演算関数を呼び出す際に、演算全体の一部分を実行する関数の名前をこの関数に渡すことができます。

この方法を使用すれば、1 つの関数を使用してさまざまな種類の計算をこなすことができます。たとえば、異なる形式の利息を計算できます。最初の関数でフレームワークを提供し、そのフレームワークに名前を渡す関数では、呼び出しページで必要な個別のアルゴリズムを実装できます。

次の簡単な例で、この使用方法を示します。binop 関数は、特定の二項演算を実行する関数の名前と、その関数の 2 つのオペランドを取る、一般化された関数です。binop 関数は指定された関数を呼び出し、オペランドをこの関数に渡すだけです。このコードでは 1 つの演算関数である合計関数を定義しています。より完全な実装では、複数の二項演算を定義することになります。

```
 <cfscript>
function binop(operation, operand1, operand2)
{ return (operation(operand1, operand2)); }
function sum(addend1, addend2)
{ return addend1 + addend2;}
x = binop(sum, 3, 5);
writeoutput(x);
</cfscript>
```

UDF を使用したクエリー結果の処理

cfloop タグなど、query 属性を持つタグの本文で UDF を呼び出した場合、クエリー列の名前を指定した引数では、列全体ではなく、その列の単一の要素が渡されます。したがって、関数では単一のクエリー要素を操作する必要があります。

たとえば次のコードでは、単一の名前と姓を結合して氏名を作成する関数を定義しています。その後、cfdocexamples データベースにクエリーを実行して全従業員の名前と姓を取得し、cfoutput タグを使用してクエリー内をループし、クエリーの各行に対してこの関数を呼び出しています。

```
 <cfscript>
function FullName(aFirstName, aLastName)
    { return aFirstName & " " & aLastName; }
</cfscript>

<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT FirstName, LastName
    FROM Employee
</cfquery>

<cfoutput query="GetEmployees">
#FullName(FirstName, LastName)#<br>
</cfoutput>
```

一般に、多数のクエリー列を操作する関数は、クエリーをループするタグの外で使用します。関数にクエリーを渡し、関数内でそのクエリーをループします。たとえば次の関数は、クエリー列内のテキストを大文字に変更します。引数にはクエリー名を渡します。

```
function UCaseColumn(myquery, colName) {
    var currentRow = 1;
    for (; currentRow lte myquery.RecordCount; currentRow = currentRow + 1)
    {
        myquery[colName][currentRow] = UCase(myquery[colName][currentRow]);
    }
    Return "";
}
```

次のコードでは、スクリプトを使用して UCaseColumn 関数を呼び出し、GetEmployees クエリー内のすべての姓を大文字に変換しています。その後、cfoutput を使用してこのクエリーをループし、列の内容を表示しています。

```
<cfscript>
    UCaseColumn(GetEmployees, "LastName");
</cfscript>
<cfoutput query="GetEmployees">
    #LastName#<br>
</cfoutput>
```

UDF の識別と確認

IsCustomFunction 関数を使用すれば、名前が UDF を表すかどうかを判断できます。IsCustomFunction 関数では、指定された引数が存在しない場合はエラーが発生します。したがって、この関数を呼び出す前に、IsDefined 関数を呼び出すなどの方法で、その名前が存在することを確認してください。次のコードに、この使用例を示します。

```
<cfscript>
if(IsDefined("MyFunc"))
    if(IsCustomFunction(MyFunc))
        WriteOutput("MyFunc is a user-defined function");
    else
        WriteOutput("Myfunc is defined but is NOT a user-defined function");
else
    WriteOutput("MyFunc is not defined");
</cfscript>
```

IsCustomFunction の引数は引用符で囲みません。これによって、引数自体が関数であるかどうかを判断できます。

Evaluate 関数の使用

ユーザー定義関数で、文字列を含んでいる引数に Evaluate 関数を使用している場合は、引数として使用するすべての変数名にスコープ識別子を含める必要があります。これによって、関数内変数との競合を避けることができます。

次の例は、引数を評価した結果を返します。この例では、スコープを指定した変数名である Variables.myname を引数に渡すと、期待した結果であるその引数の値が返されます。しかし、Variables スコープ識別子のない変数名である myname を引数に渡すと、関数内のローカル変数の値が返されます。

```
<cfscript>
    myname = "globalName";
    function readname(name) {
        var myname = "localName";
        return (Evaluate(name));
    }
</cfscript>

<cfoutput>
<!--- This one collides with local variable name. --->
    The result of calling readname with myname is:
        #readname("myname")# <br>
<!--- This one finds the name passed in. --->
    The result of calling readname with Variables.myname is:
        #readname("Variables.myname")#
</cfoutput>
```

再帰的な処理

再帰関数とは、自分自身を呼び出している関数のことです。再帰関数は、前の結果を使用して同じ処理を何度も繰り返すことで問題を解決できる場合に便利です。次の例で示されている階乗計算は、再帰的な処理が役立つ一例です。ハノイの塔ゲームも、再帰アルゴリズムを使用して解決できます。

再帰関数では、ループコードと同様に、関数が確実に停止するように条件を指定する必要があります。そうしないと、システムエラーが発生するか、ColdFusion サーバーを停止するまで関数が実行され続けます。

次の例では、数の階乗を計算します。つまり、1 からその数までのすべての整数の積を計算します。たとえば、4 の階乗は 4 × 3 × 2 × 1 = 24 です。

```
function Factorial(factor) {
    If (factor LTE 1)
        return 1;
    else
        return factor * Factorial(factor -1);
}
```

この関数は、1 より大きい数で呼び出された場合、受け取った数よりも 1 小さい数を使用して自分自身を呼び出します。その結果に元の引数を掛けて、結果を返します。したがって、この関数は要素が減算されて 1 になるまで自分自身を呼び出します。最後の再帰的呼び出しで 1 が返され、その前の呼び出しで 2 * 1 が返され、以下同様に処理されて、最初の呼び出しによって最終結果が返されます。

重要：再帰関数が自分自身を何度も呼び出すと、スタックオーバーフローが発生する場合があります。再帰関数を作成した場合は、最大回数の再帰呼び出しが行われる条件下でその関数をテストして、スタックオーバーフローが発生しないことを確認してください。

ColdFusion コンポーネントの構築と使用

互いに関連性のあるデータや関数は、ColdFusion コンポーネント (CFC) ファイルで定義することができます。CFC ページを使用すれば、関連するアクションを 1 つのファイルにまとめて、プログラムを簡略化できます。CFC を使用したアプリケーションの作成については、アドビの Web サイト www.adobe.com/jp/ を参照してください。

ColdFusion コンポーネントについて

CFC (ColdFusion コンポーネント) は、拡張子 .cfc を使用して保存されたファイルです。CFC には、データおよび関数を格納することができます。CFC に格納されているデータのことをプロパティといいます。CFC でも cffunction タグを使用して関数を定義できますが、これらは一般に関数ではなくメソッドと呼ばれます。

CFC の定義が記述されているページは、コンポーネントページとも呼ばれます。コンポーネントページは、通常の CFML ページと同じタグや関数を使用して作成します。また、いくつかの特殊なタグ (cfcomponent タグなど) やタグ属性も使用できます。

CFC では、関連する複数のメソッドを定義できます。ColdFusion のカスタムタグと異なり、複数のメソッドで定義された関連するさまざまなアクションを、1 つの CFC で実行できます。これらのメソッドでは、メタデータやスコープなどのデータコンテキストを共有したり、特定のデータベースやテーブルセットを管理したりできます。たとえば、特定のデータベースやテーブルのレコードを挿入、更新、削除、および取得するメソッドを、1 つの CFC に定義することができます。

CFC とオブジェクト指向プログラミング

CFC を使用すれば、オブジェクト指向の手法で ColdFusion コードを開発できます。ただし、オブジェクト指向プログラミングを行わなくても CFC は使用できます。CFC には、カプセル化、継承、イントロスペクションなどのオブジェクト指向の機能が用意されています。CFC のオブジェクト指向機能は、JavaScript などの他の言語のオブジェクト指向要素に似ています。

コードとデータの両方を CFC などの単一のオブジェクトにまとめることを、カプセル化といいます。カプセル化を行うことで、実際のコードを理解しなくても、ユーザーが CFC にデータを渡したり、CFC から結果を取得したりできるようになります。カプセル化を使用すると、CFC に渡されるデータを検証することができます。また、データ型の強制を行ったり、必須パラメータを確認したり、必要に応じてデフォルト値を割り当てることもできます。

CFC は、別の CFC からメソッドとプロパティを継承できます。継承を使用すれば、親コンポーネントの基本部分を流用して、個別のコンポーネントを複数構築することができます。詳細については、197 ページの「Super キーワードの使用」を参照してください。

CFC では、自分自身に関する情報を提供するイントロスペクション機能がサポートされています。コンポーネントページを HTML ブラウザで直接表示するか、ColdFusion および Adobe Dreamweaver CS3 のコンポーネントブラウザでコンポーネントページを調べるか、または CFML GetMetadata 関数を使用すれば、コンポーネントに関する情報を参照できます。この情報には、コンポーネントのパス、プロパティ、メソッド、説明用の特殊な属性やタグで指定できる追加情報などが含まれます。詳細については、200 ページの「イントロスペクションによるコンポーネント情報の取得」を参照してください。

構築した ColdFusion コンポーネント内のメソッドは、呼び出すことで使用できます。ただし、通常は、CFC のインスタンスを作成してから、CFC のメソッドを呼び出したり、プロパティを参照したりします。

CFC が有用な場合

CFC を使用すれば、次のことが行えます。

- 構造化された再利用可能なコードの開発
- Web サービスの作成
- Flash Remoting 要素の作成
- 非同期 CFC の使用

構造化された再利用可能なコードの開発

CFC は、表示要素と論理要素を分離してデータベースクエリーをカプセル化する、構造化されたアプリケーションを開発するときに有用です。CFC を使用すれば、UDF (ユーザー定義関数) やカスタムタグのように、必要に応じて再利用可能なアプリケーション機能を作成できます。コンポーネントの機能を変更、追加、または削除する場合でも、1 つのコンポーネントファイルを変更するだけで済みます。

CFC には、UDF およびカスタムタグよりも優れた点がいくつかあります。それらは CFC 固有の特性によって提供される利点であり、次のようなものがあります。

- 関連するメソッドを 1 つのコンポーネントにグループ化し、関連するコンポーネントをパッケージにグループ化する機能
- 複数のメソッドで共有可能なプロパティ
- コンポーネント固有のスコープである This スコープ
- Super キーワードの使用などによる、基本コンポーネントからのコンポーネントメソッドおよびプロパティの継承
- アクセス制御
- CFC メソッド、プロパティ、メタデータのイントロスペクション

CFC は、すべての点において優れているわけではないので、コードの再利用方法として無条件に選択することはできません。CFC のインスタンス化には時間がかかります。CFC のインスタンスを作成するよりも、カスタムタグを処理するほうが時間がかかりません。また、カスタムタグを処理するよりも、UDF を実行するほうが時間がかかりません。ただし、いったん CFC をインスタンス化すれば、CFC メソッドの呼び出しによるオーバーヘッドは、相当するカスタムタグと同程度になります。したがって、独立した単一用途のカスタムタグや UDF には CFC を使用しないでください。CFC は、関連する複数のメソッド、特にプロパティを共有するメソッドを作成する場合に有用です。

UDF やカスタムタグなどの、ColdFusion コードの再利用方法の詳細については、143 ページの「ColdFusion 要素の作成」を参照してください。

Web サービスの作成

ColdFusion では、CFC メソッドを Web サービスとして自動的に公開できます。CFC メソッドを Web サービスとして公開するには、メソッドの cffunction タグで access="remote" 属性を指定します。必要な Web サービス記述言語 (WSDL : Web Services Description Language) コードがすべて生成され、CFC メソッドがエクスポートされます。ColdFusion での Web サービス作成の詳細については、1061 ページの「Web サービスの使用」を参照してください。

Flash Remoting 要素の作成

Flash Remoting を利用した Adobe® Flash® アプリケーションでは、ColdFusion コンポーネントをビジネスロジックに簡単に活用できます。CFC では、cffunction タグで関数に名前を付けてアプリケーションロジックを定義し、cfreturn タグで Flash に結果を返します。

注意：Flash アプリケーションとやり取りを行う ColdFusion コンポーネントメソッドでは、cffunction タグの access 属性を remote に設定します。

Flash Remoting 用の CFC 作成の詳細については、605 ページの「Flash での CFC の使用」を参照してください。

非同期 CFC の使用

ColdFusion には、CFC に対して非同期でメッセージを送信するためのイベントゲートウェイが用意されています。このゲートウェイを使用して CFC を呼び出せば、CFC が完了したり値を返したりするのを待たずに、処理を続行することができます。このゲートウェイを使用した非同期 CFC は、次の用途に使用できます。

- コレクションのインデックスの再作成
- 情報のロギング
- バッチ処理の実行

非同期 CFC の使用方法の詳細については、1242 ページの「イベントゲートウェイについて」を参照してください。

ColdFusion コンポーネントの作成

CFC を作成するときには、コンポーネントページでメソッド (ColdFusion のユーザー定義関数) を作成します。メソッドにデータを渡すには、パラメータを使用します。メソッドはその関数を実行し、cfreturn タグで指定されている場合はデータを返します。

CFC では変数も定義できます。CFC では、こうした変数のことをプロパティと呼びます。

CFC の作成に使用するタグ

CFC の作成には、次のタグを使用します。これらのタグは、CFC を定義する CFML ページで使用します。

タグ	説明
cfcomponent	コンポーネントを定義し、イントロスペクション用の属性を指定します。詳細については、175 ページの「ColdFusion コンポーネントの構築」を参照してください。
cffunction	コンポーネントのメソッド (関数) を定義し、イントロスペクション用の属性を指定します。詳細については、175 ページの「コンポーネントメソッドの定義」を参照してください。
cfargument	メソッドのパラメータ (引数) を定義し、イントロスペクション用の属性を指定します。詳細については、178 ページの「メソッドパラメータの定義と使用」を参照してください。
cfproperty	Web サービスを提供する CFC の変数を定義します。コンポーネントプロパティの説明も提供できます。詳細については、183 ページの「cfproperty タグ」を参照してください。

CFC の要素

CFC には次の特性があります。

- CFC は、ファイル名拡張子が .cfc の単一の CFML ページです。コンポーネント名はファイル名と同じです。たとえば、ファイルが "myComponent.cfc" の場合、コンポーネント名は myComponent です。
- ページは、cfcomponent タグで囲まれます。このタグの外側にコードを記述することはできません。
- コンポーネントページは、メソッド (関数)、プロパティ (データ)、またはその両方を定義します。多くの CFC にはメソッドか、メソッドとプロパティがありますが、プロパティのみを含む CFC も作成できます。
- CFC のメソッドを定義するには、cffunction タグを使用します。CFScript の function ステートメントは、簡単なメソッドを作成できますが、メソッドのアクセス制御、メタデータの提供、戻り値の型の指定、生成した出力の制御を行うオプションは用意されていません。
- コンポーネントページの cffunction 定義の外側にはコードを記述することができます。このコードは、CFC がインスタンス化されたときに実行されます。または、CFC のメソッドを呼び出すたびに実行されます。

ColdFusion コンポーネントの構築

ColdFusion コンポーネントの作成には、cfcomponent および cffunction タグを使用します。cffunction タグは、単独では機能を持ちません。cfcomponent タグは、CFML で作成して cffunction タグで囲んだ機能を記述するためのエンベロープになります。次の例は、2 つのメソッドを持つコンポーネントの基本構造を示しています。

```
<cfcomponent>
    <cffunction name="firstMethod">
         <!--- CFML code for this method goes here. --->
    </cffunction>
    <cffunction name="secondMethod">
         <!--- CFML code for this method goes here. --->
    </cffunction>
</cfcomponent>
```

コンポーネントメソッドの定義

コンポーネントメソッドを定義するには、cffunction タグを使用します。次の例では、getall および getsalary という 2 つのメソッドを持つ CFC を定義します。

```
<cfcomponent>
    <cffunction name="getall" output="false" returntype="query">
        <cfset var queryall="">
        <cfquery name="queryall" datasource="cfdocexamples">
            SELECT * FROM EMPLOYEE
        </cfquery>
        <cfreturn queryall>
    </cffunction>
    <cffunction name="getsalary" output="false">
        <cfset var getNamesandSalary="">
            <cfquery name="getNamesandSalary" datasource="cfdocexamples">
                SELECT FirstName, LastName, Salary FROM EMPLOYEE
            </cfquery>
        <cfreturn getNamesandSalary>
    </cffunction>
</cfcomponent>
```

コンポーネントメソッドは ColdFusion の関数なので、メソッドの機能やコーディングテクニックの大部分は、ユーザー定義関数の場合と同じです。cffunction タグによる関数の作成方法の詳細については、150 ページの「ユーザー定義関数の作成と呼び出し」を参照してください。CFC メソッドは、他の ColdFusion 関数と同様に、出力を生成して情報を直接表示することも、メソッドを呼び出したコードまたはクライアントに値を返すこともできます。

次に示す cffunction タグの属性は CFC 専用です。

- displayname および hint 属性。CFC に関する情報を記述します。詳細については、182 ページの「CFC に関する説明を含める」を参照してください。
- access 属性。CFC へのアクセスを制御します。詳細については、199 ページの「アクセスセキュリティの使用」を参照してください。

cffunction タグの詳細については、『CFML リファレンス』を参照してください。

関連メソッドが設定された CFC の定義

CFC を定義する際に、1 つの CFC に関連メソッドをまとめることは適切なプログラミングスタイルです。たとえば、ユーザー関連の処理を行うすべてのメソッド (addUser、editUser、storeUserPreferences など) を 1 つの CFC に格納したり、関連する算術関数を 1 つの CFC にグループ化することができます。また、ショッピングカートに必要なすべてのメソッドおよびプロパティを含めることもできます。次の CFC には、2 つのコンポーネントメソッド (getEmp および getDept) を定義する 2 つの cffunction タグが含まれています。コンポーネントメソッドを呼び出すと、ExampleApps データベースに対してクエリーが実行されます。cfreturn タグによって、メソッドの呼び出し元であるクライアントまたはページにクエリーの結果が返されます。

```
<cfcomponent>
    <cffunction name="getEmp">
        <cfset var empQuery="">
         <cfquery name="empQuery" datasource="cfdocexamples" dbtype="ODBC">
            SELECT FIRSTNAME, LASTNAME, EMAIL
            FROM tblEmployees
         </cfquery>
         <cfreturn empQuery>
    </cffunction>
    <cffunction name="getDept">
        <cfset var deptQuery="">
        <cfquery name="deptQuery" datasource="cfdocexamples" dbtype="ODBC">
            SELECT *
            FROM tblDepartments
         </cfquery>
         <cfreturn deptQuery>
    </cffunction>
</cfcomponent>
```

別ファイルへの実行可能コードの配置

実行可能コードは、メインのコンポーネント定義ページとは別のファイルに配置できます。メソッド実行コードを別ファイルに配置することで、プロパティ初期化コード、メタ情報、およびメソッド定義シェルを、実行可能メソッドの定義コードと分離できます。これによって、コードをモジュール化し、CFML ページが長く複雑になるのを防ぐことができます。

コンポーネントメソッドコードを分離するには、コンポーネント定義ページで cfinclude タグを使用して、コンポーネントメソッドコードが含まれているページを呼び出します。

注意：メソッドが引数を取る場合や、呼び出しページにデータを返す場合、cfargument タグや cfreturn タグはコンポーネント定義ページに記述する必要があります。インクルードページに記述することはできません。

cfinclude タグによるコンポーネントメソッドの作成

1 次のコードを持つ "tellTime.cfc" ファイルを作成します。

```
<cfcomponent>
    <cffunction name="getUTCTime">
        <cfinclude template="getUTCTime.cfm">
    <cfreturn utcStruct.Hour & ":" & utcStruct.Minute>
    </cffunction>
</cfcomponent>
```

2 次のコードを持つ ColdFusion ページを作成し、それに "getUTCTime.cfm" という名前を付けて "tellTime.cfc" と同じディレクトリに保存します。

```
<cfscript>
    serverTime=now();
    utcTime=GetTimeZoneInfo();
    utcStruct=structNew();
    utcStruct.Hour=DatePart("h", serverTime);
    utcStruct.Minute=DatePart("n", serverTime);
    utcStruct.Hour=utcStruct.Hour + utcTime.utcHourOffSet;
    utcStruct.Minute=utcStruct.Minute + utcTime.utcMinuteOffSet;
    if (utcStruct.Minute LT 10) utcStruct.Minute = "0" & utcStruct.Minute;
</cfscript>
```

この例では、getUTCTime メソッドの定義で cfinclude タグを使用して、"getUTCTime.cfm" ファイルを呼び出しています。"getUTCTime.cfm" のコードは、現在の時刻を UTC 時間形式で計算し、時と分の値を構造体に挿入します。"tellTime.cfc" 内のメソッドはその構造体の情報を使用して、現在の UTC 時間を文字列として呼び出しページに返します。インクルードページに cfreturn ステートメントを含めることはできません。

インスタンスデータの初期化

一部のコンポーネントではインスタンスデータを使用できます。これは、コンポーネントインスタンスが存在する間のみ保持されるデータです。たとえば、ショッピングカートコンポーネントでは、ユーザーがショッピングカートに入れた商品の ID や数量を格納するためのインスタンスデータなどを使用します。一般に、インスタンスデータは、そのデータを作成、削除、または変更できる複数のメソッドによって共有されます。

CFC のインスタンスデータは、CFC のインスタンスを作成した場合にのみ参照できます。CFC 内からインスタンスデータを参照するには、this.firstvariable のように this 接頭辞を使用します。呼び出しページからインスタンスデータを参照するには、objectname.ivarname のように、ドット表記法でコンポーネントのインスタンス名とインスタンスデータ名を指定します。コンポーネントをインスタンス化せずにメソッドを呼び出した場合、通常、そのメソッドを持つコンポーネントにインスタンスデータはありません。

インスタンスデータの初期化は、コンポーネント定義の冒頭で、メソッドを定義する前に行います。このコードは、コンポーネントがインスタンス化されたとき、たとえば cfobject タグでコンポーネントインスタンスが作成されたときなどに実行されます。このインスタンスデータの初期化コードは、インスタンスが作成されるときにのみ実行され、主にコンポーネントのプロパティを " 構築 (コンストラクト)" するために使用されることから、コンストラクタコードとも呼ばれます。

コンストラクタコードでは任意の CFML タグまたは関数を使用でき、データベースのクエリーやデータの検証および操作など、あらゆる ColdFusion 処理を実行できます。コンポーネントを拡張して別のコンポーネントを作成した場合は、親コンポーネントのコンストラクタコードが、子コンポーネントのコンストラクタコードの前に実行されます。

注意：ColdFusion では、初期化コードをコンポーネント定義の冒頭に配置する必要はありませんが、冒頭に配置するのがよいプログラミングスタイルです。

次の例は、ショッピングカート CFC のコンストラクタコードを示します。

```
<cfcomponent>
    <!--- Initialize the array for the cart item IDs and quantities. --->
    <cfset This.CartData = ArrayNew(2)>
    <!--- The following variable has the ID of the "Special Deal" product for
            this session. --->
    <cfset This.Special_ID = RandRange(1, 999)>
```

スコープの詳細については、193 ページの「This スコープ」および 193 ページの「Variables スコープ」を参照してください。

CFC のインスタンスを初期化しコンストラクタとして機能する init() という名前のメソッドを定義すると便利です。init() メソッドは定数を初期化して、呼び出しページにコンポーネントのインスタンスを返すことができます。次のコードは、init() メソッドの例を示しています。

```
<cfcomponent displayname="shoppingCart">
    <cffunction name="init" access="public" output="no" returntype="shoppingCart">
        <cfargument name="shoppingCartID" type="UUID" required="yes">
        <cfset variables.shoppingCartID = arguments.shoppingCartID>
        <cfreturn this>
    </cffunction>

    <!--- Additional methods go here. --->
</cfcomponent>
```

この例の init() メソッドでは Variables スコープを使用して、ショッピングカート ID を CFC の任意の場所で利用できるようにしています。スコープの詳細については、193 ページの「CFC 変数とスコープ」を参照してください。

メソッドパラメータの定義と使用

メソッドにデータを渡すには、パラメータを使用します。コンポーネントメソッドパラメータを定義するには、cffunction タグの本文で cfargument タグを使用します。複数のパラメータを定義するには、複数の cfargument タグを使用します。このタグでは、パラメータの名前と、次の項目を指定できます。

- パラメータが必須かどうか
- 必要なデータ型
- デフォルトの引数値
- CFC イントロスペクション用の表示名およびヒントのメタデータ

注意：位置でパラメータを識別するなどの方法を使用すれば、cfargument タグを使用せずに CFC メソッドを作成できますが、多くの CFC メソッドは cfargument タグを使用します。

例 : convertTemp.cfc

"convertTemp.cfc" ファイルは、次のように構成されています。

```
<cfcomponent>
    <!--- Celsius to Fahrenheit conversion method. --->
    <cffunction name="ctof" output="false">
        <cfargument name="temp" required="yes" type="numeric">
        <cfreturn ((temp*9)/5)+32>
    </cffunction>

    <!--- Fahrenheit to Celsius conversion method. --->
    <cffunction name="ftoc" output="false">
        <cfargument name="temp" required="yes" type="numeric">
        <cfreturn ((temp-32)*5/9)>
    </cffunction>
</cfcomponent>
```

コードの説明

convertTemp CFC には、温度を変換する 2 つのメソッドが含まれています。このコードおよびその機能について、次の表で説明します。

コード	説明
<cfcomponent>	コンポーネントを定義します。
<cffunction name="ctof" output="false">	ctof メソッドを定義します。 このメソッドは出力を表示しないことを指定します。
<cfargument name="temp" required="yes" type="numeric">	ctof メソッドの temp パラメータを作成します。このパラメータは必須であり、受け付ける値は数値 (numeric) であることを指定しています。

コード	説明
<cfreturn ((temp*9)/5)+32>	メソッドが返す値を定義します。
</cffunction>	メソッドの定義を終了します。
<cffunction name="ftoc" output="false">	ftoc メソッドを定義します。 このメソッドは出力を表示しないことを指定します。
<cfargument name="temp" required="yes" type="numeric">	ftoc メソッドの temp パラメータを作成します。このパラメータは必須であり、受け付ける値は数値 (numeric) であることを指定しています。
<cfreturn ((temp-32)*5/9)>	メソッドが返す値を定義します。
</cffunction>	メソッドの定義を終了します。
</cfcomponent>	コンポーネントの定義を終了します。

例 : tempConversion.cfm

ColdFusion ページ "tempConversion.cfm" は、変換する温度を入力して変換の種類を選択するための HTML フォームです。ユーザーが送信ボタンをクリックすると、ColdFusion によって "processForm.cfm" ページのアクションが実行されます。ファイル "tempConversion.cfm" は、"convertTemp.cfc" と同じディレクトリにあり、次のように構成されています。

```
<cfform action="processForm.cfm" method="POST">
    Enter the temperature:
        <input name="temperature" type="text"><br>
        <br>
    Select the type of conversion:<br>
        <select name="conversionType">
            <option value="CtoF">Celsius to Farenheit</option>
            <option value="FtoC">Farenheit to Celsius</option>
        </select><br><br>
    <input name="submitform" type="submit" value="Submit">
</cfform>
```

例 : processForm.cfm

ColdFusion ページ "processForm.cfm" は、"tempConversion.cfm" ページのフォームにユーザーが入力した情報に基づいて、適切なコンポーネントメソッドを呼び出します。このページは、"convertTemp.cfc" と同じディレクトリに配置します。

```
<cfif #form.conversionType# is "CtoF">
    <cfinvoke component="convertTemp" method="ctof" returnvariable="newtemp"
        temp=#form.temperature#>
    <cfoutput>#form.temperature# degrees Celsius is #newtemp# degrees
        Farenheit.</cfoutput>
    <cfelseif #form.conversionType# is "FtoC">
        <cfinvoke component="convertTemp" method="ftoc"
            returnvariable="newtemp" temp=#form.temperature#>
        <cfoutput>#form.temperature# degrees Fahrenheit is #newtemp# degrees
            Celsius.</cfoutput>
</cfif>
```

コードの説明

ファイル "processForm.cfm" は、適切なコンポーネントメソッドを呼び出します。このコードおよびその機能について、次の表で説明します。

コード	説明
<cfif form.conversionType is "CtoF">	"tempConversion.cfm" ページのフォームで摂氏から華氏への変換をユーザーが選択した場合に、この cfif ブロック内のコードを実行します。
<cfinvoke component="convertTemp" method="ctof"returnvariable="newtemp"arguments.temp="#form.temperature#">	convertTemp コンポーネントのインスタンスを作成することなく、convertTemp コンポーネントの ctof メソッドを呼び出します。メソッドの結果変数として newtemp を指定します。ユーザーがフォームに入力した温度の値を、ctof メソッドの cfargument タグで指定した temp 変数に代入します。ctof メソッドを呼び出すと、この temp 変数は Arguments スコープに割り当てられます。変数とスコープの詳細については、193 ページの「CFC 変数とスコープ」を参照してください。
<cfoutput>#form.temperature# degrees Celsius is #newtemp#bdegrees Fahrenheit.</cfoutput>	ユーザーがフォームに入力した温度、"degrees Celsius is" というテキスト、ctof メソッドから得られた新しい温度の値、"degrees Fahrenheit." というテキストを表示します。
<cfelseif #form.conversionType# is "FtoC">	"tempConversion.cfm" ページのフォームで華氏から摂氏への変換をユーザーが選択した場合に、この cfelseif ブロック内のコードを実行します。
<cfinvoke component="converttemp" method="ftoc"returnvariable="newtemp" temp=#form.temperature#>	convertTemp コンポーネントのインスタンスを作成することなく、convertTemp コンポーネントの ftoc メソッドを呼び出します。メソッドの結果変数として newtemp を指定します。ユーザーがフォームに入力した温度の値を、ftoc メソッドの cfargument タグで指定した temp 変数に代入します。ftoc メソッドを呼び出すと、この temp 変数は Arguments スコープに割り当てられます。変数とスコープの詳細については、193 ページの「CFC 変数とスコープ」を参照してください。
<cfoutput>#form.temperature# degrees Fahrenheit is #newtemp#degrees Celsius.</cfoutput>	ユーザーがフォームに入力した温度、"degrees Fahrenheit is " というテキスト、ftoc メソッドから得られた新しい温度の値、"degrees Celsius." というテキストを表示します。
</cfif>	cfif ブロックを閉じます。

この例を実行するには、ブラウザで "tempConversion.cfm" ページを表示します。フォームのテキストボックスに値を入力すると、その値が変数 form.temperature に格納されます。次に "processForm.cfm" ページで処理が実行されます。このページは、値を form.temperature として参照します。いずれかのメソッドを呼び出すと、cfinvoke タグが form.temperature の値を temp に代入します。temp は、適切なメソッドの cfargument タグで指定されている引数です。convertTemp コンポーネント内の適切なメソッドが必要な計算を実行し、新しい値を newtemp として返します。

cfargument タグの詳細については、『CFML リファレンス』を参照してください。

コンポーネントメソッド定義内のパラメータ値にアクセスするには、Arguments スコープに対して構造体や配列の表記法を使用します。次の例では、lastName 引数を Arguments.lastName として参照しています。また、Arguments[1] として参照することもできます。# 記号を使用して #lastname# のように引数に直接アクセスすることもできますが、#Arguments.lastname# のようにスコープを明示するのがよいプログラミングスタイルです。また、配列または構造体の表記法を使用して、複数のパラメータをループすることができます。

Arguments スコープの詳細については、194 ページの「Arguments スコープ」を参照してください。

コンポーネントメソッド定義でのパラメータの定義

次の内容のコンポーネントを作成し、"corpQuery.cfc" という名前を付けて Web のルートディレクトリの下のディレクトリに保存します。

```
<cfcomponent>
    <cffunction name="getEmp">
        <cfargument name="lastName" type="string" required="true"
                hint="Employee last name">
        <cfset var empQuery="">
         <cfquery name="empQuery" datasource="cfdocexamples">
            SELECT LASTNAME, FIRSTNAME, EMAIL
            FROM tblEmployees
            WHERE LASTNAME LIKE '#Arguments.lastName#'
         </cfquery>
            <!--- Use cfdump for debugging purposes. --->
         <cfdump var=#empQuery#>
    </cffunction>
    <cffunction name="getCat" hint="Get items below specified cost">
    <cfargument name="cost" type="numeric" required="true">
        <cfset var catQuery="">
        <cfquery name="catQuery" datasource="cfdocexamples">
            SELECT ItemName, ItemDescription, ItemCost
            FROM tblItems
            WHERE ItemCost <= #Arguments.cost#
         </cfquery>
            <!--- Use cfdump for debugging purposes. --->
         <cfdump var=#catQuery#>
    </cffunction>
</cfcomponent>
```

この例では、cfargument で次の属性を指定しています。

- name 属性でパラメータ名を定義しています。
- lastName 引数の type 属性では、パラメータとしてテキスト文字列が必要であることを指定しています。cost 引数の type 属性では、パラメータとして数値が必要であることを指定しています。これらの属性は、データベースに送信する前にデータを検証します。
- required 属性では、パラメータが必須であり、指定しなかった場合は例外が発生することを指定しています。
- パラメータ値には Arguments スコープを使用してアクセスできます。

結果の提供

ColdFusion コンポーネントは、次の方法で情報を提供できます。

- 出力を生成して呼び出しページに表示する。
- 変数を返す。

アプリケーションでは、この 2 つの方法のいずれか、または両方を使用できます。どの方法を使用するかは、アプリケーションでの必要性やプログラマのコーディングスタイルによって異なります。たとえば、ビジネスロジックを実行する CFC メソッドの多くは、結果を変数として返します。出力を直接表示する CFC メソッドの多くは、出力生成用のモジュラーユニットとして設計されており、ビジネスロジックは実行しません。

出力の表示

メソッド内にテキスト、HTML コード、または CFML タグの生成出力が含まれている場合は、出力を明示的に抑制していない限り、コンポーネントメソッドを呼び出したクライアントに生成出力としてそれらが返されます。クライアントが Web ブラウザである場合は、Web ブラウザにそれらの結果が表示されます。たとえば、次の getLocalTime1 コンポーネントメソッドは、そのメソッドを呼び出したページにローカル時刻を直接表示します。

```
<cfcomponent>
    <cffunction name="getLocalTime1">
        <cfoutput>#TimeFormat(now())#</cfoutput>
    </cffunction>
</cfcomponent>
```

Flash Remoting を使用して呼び出されるコンポーネントメソッドや、Web サービスとして呼び出されるコンポーネントメソッドでは、この方法で結果を提供することはできません。

結果の変数を返す

コンポーネントメソッドの定義では、cfreturn タグを使用することで、変数データとしてクライアントに結果を返すことができます。たとえば、次の getLocalTime2 コンポーネントメソッドは、メソッドを呼び出した ColdFusion ページや他のクライアントに、変数としてローカル時刻を返します。

```
<cfcomponent>
    <cffunction name="getLocalTime">
        <cfreturn TimeFormat(now())>
    </cffunction>
</cfcomponent>
```

結果を受け取った ColdFusion ページや Flash アプリケーションなどのクライアントは、必要に応じてこの変数データを使用できます。

注意：URL の使用またはフォームの送信によって CFC が呼び出された場合、変数は WDDX パケットとして返されます。Flash Remoting や他の CFC インスタンスによって呼び出された CFC では、This スコープを返す必要はありません。

値を返すときには、文字列、整数、配列、構造体、CFC のインスタンスなど、すべてのデータ型を使用できます。CFScript の return ステートメントと同様に、cfreturn タグは単一の変数を返します。したがって、一度に複数の結果値を返したい場合は、構造体を使用します。メソッドで出力を表示したくない場合は、cffunction タグで output="false" を指定します。

cfreturn タグの使用方法の詳細については、『CFML リファレンス』を参照してください。

CFC に関する説明を含める

ColdFusion では、いくつかの方法を使用して、CFC に関する説明をコンポーネント定義に含めることができます。この説明は、イントロスペクションを使用して CFC に関する情報を表示したときや、GetMetadata または GetComponentMetaData 関数を呼び出してコンポーネントのメタデータを取得したときに使用されます。CFC に説明を含める方法としては、次のものが用意されています。

- displayname および hint 属性
- ユーザー定義メタデータ属性
- cfproperty タグ

情報の表示の詳細については、200 ページの「イントロスペクションによるコンポーネント情報の取得」を参照してください。

displayname および hint 属性

cfcomponent、cffunction、cfargument、および cfproperty タグには、displayname および hint 属性があります。

displayname 属性を使用すると、コンポーネント、属性、メソッド、またはプロパティにわかりやすい名前を付けることができます。この属性は、イントロスペクションを使用したときに、コンポーネントまたはメソッド名の横か、パラメータ情報の行に丸括弧で囲まれて表示されます。

コンポーネント、メソッド、または引数に関するより詳しい説明を提供するには、hint 属性を使用します。この属性は、イントロスペクションを表示したときに、コンポーネントやメソッドの記述から独立した行、および引数の記述の末尾に表示されます。

メタデータ属性

cfcomponent、cffunction、cfargument、および cfproperty タグでは、任意のメタデータ情報を属性として含めることができます。メタデータ属性を作成するには、メタデータ属性の名前と値を指定します。たとえば次の cfcomponent タグでは、Author 属性がメタデータ属性です。この属性は、関数のパラメータとして使用するためではなく、この CFC の作成者を示すために挿入されています。

```
<cfcomponent name="makeForm" Author="Bean Lapin">
```

メタデータ属性は、ColdFusion の処理で使用されることはなく、ColdFusion の標準のイントロスペクションでも表示されませんが、GetMetaData または GetComponentMetaData 関数を使用してメタデータを取得することでアクセスおよび表示できます。属性名は、CFC 要素のメタデータ構造体のキーになります。

メタデータ属性は、説明以外の用途にも利用できます。アプリケーションでは、GetMetadata 関数を使用して、コンポーネントインスタンスのメタデータ属性を取得できます。また、GetComponentMetaData 関数を使用すれば、インターフェイスやインスタンス化していないコンポーネントのメタデータを取得できます。これらの方法でメタデータを取得したら、その値に基づいて処理を行うことができます。たとえば、次のような cfcomponent タグを持つ mathCFC コンポーネントがあるとします。

```
<cfcomponent displayname="Math Functions" MetaType="Float">
```

この場合、次のコードを持つ ColdFusion ページでは、MetaTypeInfo 変数に Float が設定されます。

```
<cfobject component="mathCFC" name="MathFuncs">
<cfset MetaTypeInfo=GetMetadata(MathFuncs).MetaType>
```

注意：すべてのメタデータの値は、GetMetadata 関数から返されるメタデータ構造体内の文字列によって置き換えられます。したがって、カスタムメタデータ属性では式を使用しないでください。

cfproperty タグ

cfproperty タグは、次に示すように、WSDL で使用する複合データ型を作成したり、コンポーネントプロパティの説明を提供したりするのに使用します。

- ColdFusion Web サービスの WSDL で使用する複合データ型を作成します。詳細については、1076 ページの「ColdFusion コンポーネントによる Web サービスのデータ型の定義」を参照してください。
- ColdFusion イントロスペクションにコンポーネントプロパティの説明を提供します。cfproperty タグの標準属性の値がイントロスペクション情報として示されます。

注意：cfproperty タグを使用しても、変数の作成や値の代入は行われません。このタグは、情報提供のためにのみ使用されます。プロパティを作成して値を設定するには、cfset タグまたは CFScript の代入ステートメントを使用します。

ColdFusion コンポーネントの保存とネーミング

次の表に、コンポーネントファイルを保存できる場所と、さまざまな場所からアクセス可能かどうかを示します。

	URL	フォーム	Flash Remoting	Web サービス	ColdFusion ページ
現在のディレクトリ	N/A	可	N/A	N/A	可
Web ルート	可	可	可	可	可
ColdFusion マッピング	不可	不可	不可	不可	可
カスタムタグのルート	不可	不可	不可	不可	可

注意：ColdFusion マッピングおよびカスタムタグのルートは、Web のルート内に存在する場合があります。この場合、ColdFusion マッピングおよびカスタムタグのルートは、URL、フォーム、Flash Remoting、Web サービス呼び出しなどのリモートリクエストからアクセスできます。

同じディレクトリに複数のコンポーネントを保存した場合、それらのコンポーネントはコンポーネントパッケージのメンバーになります。関連する CFC はパッケージにグループ化することができます。アプリケーションでは、修飾されたコンポート名を使用して、ディレクトリ内のコンポーネントを参照できます。" 修飾 " とは、アクセス可能なディレクトリのサブディレクトリ名から、そのコンポーネントが含まれているディレクトリまでの各ディレクトリを、ピリオドで区切ってコンポーネント名の前に付けることをいいます。たとえば次の例は、price というコンポーネントの修飾名です。

```
catalog.product.price
```

この例では、前の表に示すように、ColdFusion がコンポーネントを検索するディレクトリの "catalog¥product" サブディレクトリに "price.cfc" ファイルが存在している必要があります。修飾名を使用してコンポーネントを参照した場合、ColdFusion は 190 ページの「CFC の場所の指定」に示されている手順でコンポーネントを検索します。

わかりやすいネーミング規則を使用することは重要です。パッケージ化されたアプリケーションの一部としてコンポーネントをインストールする場合は特に重要です。

ColdFusion コンポーネントの使用

CFC は、次の 2 通りの方法で使用できます。

1. CFC オブジェクトをインスタンス化する方法。これによって、CFC のインスタンスが作成されます。その後で、インスタンスのメソッドを呼び出します。CFC のメソッドおよびデータは、インスタンスの要素としてアクセスできます。また、cfinvoke タグでインスタンスを使用して、CFC メソッドを呼び出すこともできます。CFC をインスタンス化すると、CFC のデータはそのインスタンスが存在する限り保持されます。メソッドを呼び出すたびにインスタンス作成のオーバーヘッドが発生することはありません。

   CFC をインスタンス化して CFC にデータを保持します。CFC を同一ページで複数回使用する場合は、処理効率を上げるため、CFC をインスタンス化してからメソッドを呼び出してください。

   Flash Remoting および Web サービスによってリモートで実行されるメソッドは、実行前に必ず CFC のインスタンスを作成します。

2. CFC をインスタンス化せずに、CFC のメソッドを呼び出す方法。これを、メソッドの一時的な呼び出しといいます。この場合も CFC のインスタンスが作成されますが、このインスタンスは、メソッドが呼び出されてから結果が返されるまでの間しか存在しません。複数の呼び出しにまたがってデータが保持されることはなく、CFML の別の場所で再利用できる CFC のインスタンスが ColdFusion で保持されることはありません。CFML リクエストで CFC を複数回使用する場合は、メソッドを呼び出す前に CFC のインスタンスを作成することをお勧めします。メソッドの一時的な呼び出しを頻繁に行う場合は、CFC メソッドではなく、ユーザー定義関数の作成を検討してください。

永続 CFC を作成するには、Session や Application などの永続スコープに CFC インスタンスを割り当てます。この方法を使用すれば、ショッピングカートやログインユーザーなど、複数のセッションにまたがって存続する必要のあるオブジェクトの CFC を作成できます。また、アプリケーション固有のデータやメソッドを提供する CFC も作成できます。

CFC を使用するためのタグ

次の表に、CFC のインスタンス化や呼び出しに使用するタグを示します。CFC のインスタンス化や呼び出しを行う CFML ページでは、これらのタグを使用します。

タグ	説明
cfinvoke	CFC のメソッドを呼び出します。
cfinvokeargument	パラメータの名前と値をコンポーネントメソッドに渡します。
cfobject	CFC のインスタンスを作成します。
CreateObject	CFC のインスタンスを作成します。

CFC を呼び出す方法

ColdFusion では、さまざまな方法を使用して、CFC のインスタンス化や CFC メソッドの呼び出しを行えます。次の表に、それらの方法 (ColdFusion タグや関数など) を示します。

呼び出す方法	説明	詳細情報
cfinvoke タグ	コンポーネントメソッドを呼び出します。CFC インスタンスのメソッドを呼び出したり、メソッドを一時的に呼び出したりすることができます。	185 ページの「cfinvoke タグによる CFC メソッドの呼び出し」を参照してください。
cfset タグと代入ステートメント	メソッドを呼び出して、コンポーネントインスタンスのプロパティにアクセスします。	187 ページの「CFScript および CFML でのコンポーネントの直接使用」を参照してください。
URL (HTTP GET)	URL 文字列にコンポーネント名とメソッド名を指定して、コンポーネントメソッドを一時的に呼び出します。	188 ページの「URL によるコンポーネントメソッドの呼び出し」を参照してください。
フォームコントロール (HTTP POST)	HTML の form タグと input タグおよびそれらの属性を使用して、コンポーネントメソッドを一時的に呼び出します。	189 ページの「フォームによるコンポーネントメソッドの呼び出し」を参照してください。
Flash Remoting	ActionScript は、コンポーネントメソッドを一時的に呼び出すことができます。	595 ページの「Flash Remoting サービスの使用」を参照してください。
Web サービス	cfinvoke タグと CFScript で ColdFusion の Web サービスを利用します。外部アプリケーションも、CFC メソッドを Web サービスとして利用できます。	1061 ページの「Web サービスの使用」を参照してください。

CFC のインスタンス化

1 つの ColdFusion リクエストで CFC を複数回使用する場合や、CFC で永続的にプロパティを使用する場合は、cfobject タグまたは CreateObject 関数を使用して CFC をインスタンス化してから、メソッドを呼び出します。

次の例では、cfobject タグを使用して tellTime CFC のインスタンスを作成します。

```
<cfobject component="tellTime" name="tellTimeObj">
```

次の例では、CreateObject 関数を使用して、同じコンポーネントを CFScript でインスタンス化します。

```
tellTimeObj = CreateObject("component", "tellTime");
```

cfinvoke タグによる CFC メソッドの呼び出し

cfinvoke タグは、CFC インスタンスのメソッドを呼び出したり、CFC メソッドを一時的に呼び出したりできます。また、cfinvoke タグを使用して CFC 内から CFC メソッドを呼び出すこともできます。

CFC インスタンスのメソッドの呼び出し

CFC インスタンスのコンポーネントメソッドを呼び出すには、cfinvoke タグを使用して、次の項目を指定します。

- CFC インスタンス名を # 記号で囲んで component 属性に指定します。
- メソッド名を method 属性に指定します。
- 任意のパラメータを指定します。パラメータの受け渡しの詳細については、191 ページの「cfinvoke タグでのメソッドパラメータの受け渡し」を参照してください。
- コンポーネントメソッドが結果を返す場合は、結果を格納する変数の名前を returnVariable 属性に指定します。

次の手順は、現在の UTC とローカル時刻を表示するアプリケーションの作成方法を示しています。

1 次のコードを使用して、"tellTime2.cfc" という名前のファイルを作成します。

```
<cfcomponent>
    <cffunction name="getLocalTime" access="remote">
        <cfreturn TimeFormat(now())>
    </cffunction>
    <cffunction name="getUTCTime" access="remote">
        <cfscript>
            serverTime=now();
            utcTime=GetTimeZoneInfo();
            utcStruct=structNew();
            utcStruct.Hour=DatePart("h", serverTime);
            utcStruct.Minute=DatePart("n", serverTime);
            utcStruct.Hour=utcStruct.Hour + utcTime.utcHourOffSet;
            utcStruct.Minute=utcStruct.Minute + utcTime.utcMinuteOffSet;
            if (utcStruct.Minute LT 10) utcStruct.Minute = "0" & utcStruct.Minute;
        </cfscript>
        <cfreturn utcStruct.Hour & ":" & utcStruct.Minute>
    </cffunction>
</cfcomponent>
```

この例では、getLocalTime と getUTCTime の 2 つのコンポーネントメソッドを定義しています。

2 次のコードを使用して ColdFusion ページを作成し、tellTime コンポーネントと同じディレクトリに保存します。

```
<!--- Create the component instance. --->
<cfobject component="tellTime2" name="tellTimeObj">
<!--- Invoke the methods. --->
<cfinvoke component="#tellTimeObj#" method="getLocalTime" returnvariable="localTime">
<cfinvoke component="#tellTimeObj#" method="getUTCTime" returnvariable="UTCTime">
<!--- Display the results. --->
<h3>Time Display Page</h3>
<cfoutput>
    Server's Local Time: #localTime#<br>
    Calculated UTC Time: #UTCTime#
</cfoutput>
```

この例では、cfobject タグを使用して tellTime コンポーネントのインスタンスを作成し、cfinvoke タグを使用してインスタンスの getLocalTime メソッドと getUTCTime メソッドを呼び出しています。ここでは、CFC のメソッドに機能ロジックが実装されています。このロジックが呼び出しページに結果を返すと、呼び出しページでその結果が表示されます。この構造によって、表示機能とロジック機能が分離されているので、再利用性の高いコードになっています。

コンポーネントメソッドの一時的な呼び出し

ColdFusion ページまたはコンポーネントでは、cfinvoke タグを使用して、永続 CFC インスタンスを作成することなく、コンポーネントメソッドを呼び出すことができます。

コンポーネントメソッドを一時的に呼び出すには、cfinvoke タグを使用して次の項目を指定します。

- コンポーネントの名前またはパスを component 属性に指定します。
- メソッド名を method 属性に指定します。
- 任意のパラメータを指定します。パラメータの受け渡しの詳細については、191 ページの「cfinvoke タグでのメソッドパラメータの受け渡し」を参照してください。
- コンポーネントメソッドが結果を返す場合は、結果を格納する変数の名前を returnVariable 属性に指定します。

次の手順は、ローカル時刻を表示するアプリケーションの作成方法を示しています。

1 次のコンポーネントを作成し、"tellTime.cfc" という名前で保存します。

```
<cfcomponent>
    <cffunction name="getLocalTime">
        <cfoutput>#TimeFormat(now())#</cfoutput>
    </cffunction>
</cfcomponent>
```

この例では、現在の時刻を表示する getLocalTime というメソッドを持つコンポーネントを定義しています。

2 次のコードを使用して ColdFusion ページを作成し、tellTime コンポーネントと同じディレクトリに保存します。

```
<h3>Time Display Page</h3>
<b>Server's Local Time:</b>
<cfinvoke component="tellTime" method="getLocalTime">
```

この例では、cfinvoke タグを使用して、永続 CFC インスタンスを作成することなく、getLocalTime コンポーネントメソッドを呼び出しています。

CFC 定義内での cfinvoke タグの使用

cfinvoke タグを使用して、コンポーネント定義の中からコンポーネントメソッドを呼び出すことができます。たとえば、コンポーネントの他のメソッドにサービスを提供するユーティリティメソッドを呼び出すことができます。この方法で cfinvoke タグを使用する場合は、次の例のように、インスタンスの作成や cfinvoke タグでのコンポーネント名の指定は行いません。

```
<cfcomponent>
    <cffunction name="servicemethod" access="public">
        <cfoutput>At your service...<br></cfoutput>
    </cffunction>
    <cffunction name="mymethod" access="public">
        <cfoutput>We're in mymethod.<br></cfoutput>
        <!--- Invoke a method in this CFC. --->
        <cfinvoke method="servicemethod">
    </cffunction>
</cfcomponent>
```

注意：メソッドを、そのメソッドが定義されているコンポーネント定義の中から呼び出す場合は、This スコープを使用しないでください。使用するとアクセス権がリセットされます。

ダイナミックなメソッド名を使用したメソッド呼び出し

変数データ (フォームの入力など) に基づいてコンポーネントメソッドを効率的に呼び出したい場合は、cfinvoke タグを使用するのが唯一の方法です。この場合、Form.method などの変数名を method 属性の値として使用します。次の例では、ユーザーはフォームからレポートを選択します。

```
<select name="whichreport">
    <option value="all">Complete Report</option>
    <option value="salary">Salary Information</option>
</select>
```

次に、ユーザーが選択した項目に基づいて、cfinvoke タグで適切なメソッドを呼び出します。

```
<cfinvoke component="getdata" method="#form.whichreport#" returnvariable="queryall">
```

CFScript および CFML でのコンポーネントの直接使用

CFScript や CFML タグの中では、コンポーネントインスタンスのメソッドを直接呼び出せます。コンポーネントメソッドを直接呼び出すには、CreateObject 関数または cfobject タグを使用してコンポーネントをインスタンス化します。その後、インスタンス名の後にピリオドとメソッド名を指定することで、メソッドを呼び出します。メソッドがパラメータを取らない場合でも、メソッド名の後には常に丸括弧を使用します。

このシンタックスは、cfset タグ内や、cfoutput タグ本文の # 記号で囲まれた箇所など、ColdFusion 関数を使用できる場所であればどこでも使用できます。

CFScript でのコンポーネントメソッドの呼び出し

次の例は、CFScript でコンポーネントメソッドを呼び出す方法を示しています。

```
<!--- Instantiate once and reuse the instance.--->
<cfscript>
    tellTimeObj=CreateObject("component","tellTime");
    WriteOutput("Server's Local Time: " & tellTimeObj.getLocalTime());
    WriteOutput("<br> Calculated UTC Time: " & tellTimeObj.getUTCTime());
</cfscript>
```

この例では、3 つの CFScript ステートメントで次の処理が行われます。

1. CreateObject 関数で、tellTime CFC が tellTimeObj としてインスタンス化されます。
2. 最初の WriteOutput 関数で、テキストと、tellTimeObj インスタンスの getLocalTime メソッドによって返された結果が表示されます。
3. 2 番目の WriteOutput 関数で、テキストと、tellTimeObj インスタンスの getUTCTime メソッドによって返された結果が表示されます。

CFScript では、methodName() のように、標準の関数シンタックスでメソッド名を使用します。

CFML でのコンポーネントメソッドの呼び出し

次の例では、CFML タグを使用して、CFScript の例と同じ結果を生成します。

```
<cfobject name="tellTimeObj" component="tellTime">
<cfoutput>
    Server's Local Time: #tellTimeObj.getLocalTime()#<br>
    Calculated UTC Time: #tellTimeObj.getUTCTime()#
</cfoutput>
```

コンポーネントデータへの直接アクセス

CFScript および cfset の代入ステートメントでは、コンポーネントの This スコープのデータに直接アクセスできます。たとえば、ユーザーデータを格納する CFC に This.lastUpdated というプロパティがある場合は、次のようなコードを使用できます。

```
<cfobject name="userDataCFC" component="userData">
<cfif DateDiff("d", userDataCFC.lastUpdated, Now()) GT 30>
    <!--- Code to deal with older data goes here. --->
</cfif>
```

詳細については、193 ページの「This スコープ」を参照してください。

フォームおよび URL による CFC メソッドの呼び出し

URL で CFC を指定するか、HTML および CFML フォームタグを使用して、CFC メソッドを直接呼び出すことができます。HTTP リクエストはすべて一時的であるため、これらのメソッドは一時的にしかメソッドを呼び出せません。永続 CFC インスタンスを作成することはできません。

URL によるコンポーネントメソッドの呼び出し

URL を使用してコンポーネントメソッドを呼び出すには、URL クエリー文字列の標準のシンタックスである名前と値のペアを使用して、URL の末尾にメソッド名を追加します。1 回の URL リクエストで呼び出せるコンポーネントは 1 つのみです。次に例を示します。

```
http://localhost:8500/tellTime.cfc?method=getLocalTime
```

注意：URL 呼び出しを行えるようにするには、cffunction タグの access 属性を remote に設定します。

URL を使用してコンポーネントメソッドにパラメータを渡すには、次のように、URL クエリー文字列の標準のシンタックスである名前と値のペアを使用して、URL の末尾にパラメータを追加します。

```
http://localhost:8500/corpQuery.cfc?method=getEmp&lastName=camden
```

URL で複数のパラメータを渡すには、次のように、アンパサンド文字 (&) を使用して名前と値のペアを区切ります。

```
http://localhost:8500/corpQuerySecure.cfc?method=getAuth&store=women&dept=shoes
```

注意：データを保護し、機密情報が Web を介して漏洩するのを防ぐために、URL 文字列には機密情報を指定しないでください。パスワード、住所、電話番号など、ユーザーの個人情報はすべて機密情報と考えられます。

URL を使用してアクセスした CFC メソッドが出力を直接表示する場合は、ユーザーのブラウザに出力が表示されます。cffunction タグに output="No" を指定すると、出力を表示しないようにすることができます。CFC が cfreturn タグを使用して結果を返す場合は、そのテキストが HTML 編集形式に変換され (特殊文字が HTML のエスケープシーケンスに置き換えられ)、その結果が WDDX パケットに格納され、そのパケットが HTML に挿入されてクライアントに返されます。

フォームによるコンポーネントメソッドの呼び出し

ColdFusion フォームまたは HTML フォームを使用してメソッドを呼び出す方法は次のとおりです。

- form または cfform タグの action 属性に、CFC ファイル名またはパスを指定します。
- 非表示のフォームフィールドで、次のように CFC メソッドを指定します。

```
<form action="myComponent.cfc" method="POST">.
    <input type="Hidden" name="method" value="myMethod">
```

- あるいは、POST メソッドを使用してフォームを送信する場合は、ファイル名の後に ?method=< メソッド名 > を付けます。< メソッド名 > の部分では CFC メソッド名を指定します。たとえば、次のように指定します。この方法は、GET メソッドでは使用できません。

```
<form action="myComponent.cfc?method=myMethod" method="POST">.
```

- コンポーネントメソッドの各パラメータに対応する input タグを作成します。タグの name 属性でメソッドのパラメータ名を指定する必要があります。フィールド値がパラメータ値として使用されます。
- CFC メソッドを定義する cffunction タグで、access="remote" 属性を指定します。

フォームから呼び出した CFC メソッドが出力を直接表示する場合は、ユーザーのブラウザに出力が表示されます。また、cffunction タグの output 属性を使用すれば、出力を表示しないようにすることができます。CFC が cfreturn タグを使用して結果を返す場合は、そのテキストが HTML 編集形式に変換されて WDDX パケットに格納され、そのパケットが HTML に含められてクライアントに返されます。

1 次の内容の "corpFind.cfm" ファイルを作成します。

```
<h2>Find People</h2>
<form action="components/corpQuery.cfc?method=getEmp" method="post">
    <p>Enter employee's last Name:</p>
    <input type="Text" name="lastName">
    <input type="Hidden" name="method" value="getEmp">
    <input type="Submit" title="Submit Query"><br>
</form>
```

この例では、form タグの action 属性で corpQuery コンポーネントを指定し、getEmp メソッドを呼び出しています。

2 次の例に示す "corpQuery.cfc" ファイルを作成します。cffunction タグでは access="remote" を指定しています。

```
<cfcomponent>
    <cffunction name="getEmp" access="remote">
        <cfargument name="lastName" required="true">
        <cfset var empQuery="">
         <cfquery name="empQuery" datasource="cfdocexamples">
            SELECT LASTNAME, FIRSTNAME, EMAIL
            FROM tblEmployees
            WHERE LASTNAME LIKE '#arguments.lastName#'
         </cfquery>
         <cfoutput>Results filtered by #arguments.lastName#:</cfoutput><br>
         <cfdump var=#empQuery#>
    </cffunction>
</cfcomponent>
```

3 Web ブラウザを開いて、次の URL を入力します。

```
http://localhost/corpFind.cfm
```

ColdFusion によって検索フォームが表示されます。値を入力して [クエリーの送信] ボタンをクリックすると、ブラウザに結果が表示されます。

ColdFusion や基本的な HTML 以外の方法による CFC へのアクセス

Flash Remoting を利用した Flash アプリケーションでは、ColdFusion コンポーネントをビジネスロジックに簡単に活用できます。また、CFC をエクスポートすれば、任意のアプリケーションから Web サービスとして CFC メソッドにアクセスできます。

Flash Remoting アプリケーションとやり取りを行う ColdFusion コンポーネントメソッドでは、cffunction タグの access 属性を remote に設定します。

Flash Remoting 用の CFC 作成の詳細については、595 ページの「Flash Remoting サービスの使用」を参照してください。

正しい形式で CFC が構築されていれば、ColdFusion で自動生成される WSDL ファイルを参照することによって、任意のアプリケーション (ColdFusion アプリケーション、Java アプリケーション、JSP ページ、.Net アプリケーションなど) から Web サービスとしてその CFC にアクセスできます。

コンポーネントの WSDL 定義を表示するには、次のように、URL にコンポーネントの Web アドレスを指定し、その後に ?wsdl を付けます。

```
http://localhost:8500/MyComponents/arithCFC.cfc?wsdl
```

CFC を Web サービスとして使用する方法の詳細については、1061 ページの「Web サービスの使用」を参照してください。

CFC の場所の指定

コンポーネントのインスタンス化や呼び出しを行うときは、コンポーネント名のみを指定するか、または修飾パスを指定することができます。修飾パスを指定するには、ディレクトリ名を円記号 (¥) ではなくピリオド (.) で区切ります。たとえば、myApp.cfcs.myComponent は、myApp¥cfcs¥myComponent.cfc に定義されているコンポーネントを表します。詳細については、183 ページの「ColdFusion コンポーネントの保存とネーミング」を参照してください。

指定された CFC は、次のルールに従って検索されます。

- cfinvoke タグまたは cfobject タグ、または CreateObject 関数を使用して CFML ページから CFC にアクセスする場合は、次の順序でディレクトリが検索されます。
  1. 呼び出し CFML ページのローカルディレクトリ
  2. Web ルート
  3. ColdFusion Administrator の [カスタムタグのパス] ページで指定されているディレクトリ
- コンポーネント名のみを指定した場合は、これらの各ディレクトリで、指定されたコンポーネントが検索されます。

- myApp.cfcs.myComponent のような修飾パスを指定した場合は、これらの各ディレクトリで、パスの最初の要素に一致するディレクトリ (この例では myApp) が検索されます。一致するディレクトリが見つかると、そのディレクトリの下で、指定されたパスのファイル (この例では "myApp¥cfcs¥myComponent.cfc") が検索されます。

  注意：最初のパス要素に一致するディレクトリが見つかっても、そのディレクトリの下に CFC が見つからない場合は、見つからないことを示すエラーが返され、別のディレクトリの検索は行われません。

- URL、フォームフィールド、Flash Remoting、または Web サービス呼び出しを使用して、CFC メソッドをリモートで呼び出した場合は、Web ルートを基準として指定のパスが検索されます。ローカル Web ページで直接指定したフォームフィールドおよび URL の場合は、ページディレクトリを基準とするパスも検索されます。

  注意：UNIX および Linux システムの場合、CFC 名またはカスタムタグ名とファイル名は、次のように照合されます。まず、すべて小文字の名前を持つファイルが検索されます。見つからない場合は、大文字小文字が CFML に一致するファイルが検索されます。たとえば、<cfobject name="myObject" Component="myComponent"> と指定すると、最初に **"mycomponent.cfc"** が検索され、見つからない場合は **"myComponent.cfc"** が検索されます。

メソッドへのパラメータの受け渡し

CFC のメソッドにパラメータを渡すには、cfinvoke タグを使用する方法、メソッドを直接呼び出す方法、URL でパラメータを渡す方法があります。

cfinvoke タグでのメソッドパラメータの受け渡し

cfinvoke タグを使用する場合、CFC メソッドにパラメータを渡すには、次のような方法があります。

- cfinvoke タグの属性として name="value" 形式で渡します。
- cfinvoke タグの argumentcollection 属性で渡します。
- cfinvoke タグの本文で cfinvokeargument タグを使用します。

1 つの呼び出しの中で、これらの方法を自由に組み合わせることができます。複数の方法で同じ名前を指定した場合は、次の優先順位に基づいて値が使用されます。

1. cfinvokeargument タグ
2. cfinvoke 属性の名前と値のペア
3. argumentcollection 引数

属性形式によるパラメータの受け渡し

cfinvoke タグでは、次の例のように、タグ属性の名前と値のペアとしてパラメータを渡すことができます。

```
<cfinvoke component="authQuery" method="getAuthSecure"
    lastName="#session.username#" pwd="#url.password#">
```

この例では、パラメータが lastName および pwd 属性として渡されています。

注意：cfinvoke タグの属性名は予約されているので、パラメータ名としては使用できません。予約済みの属性名は、component、method、argumentCollection、および returnVariable です。詳細については、『CFML リファレンス』を参照してください。

argumentCollection 属性によるパラメータの受け渡し

構造体にパラメータを保存すれば、cfinvoke タグの argumentCollection 属性を使用して、その構造体を直接渡すことができます。この方法は、CFC に引数として渡す値が既存の構造体またはスコープ (Forms スコープなど) に含まれている場合や、条件処理またはループ処理を使用してパラメータを作成する場合に便利です。

argumentCollection 構造体を渡す場合は、構造体の各キーがパラメータの名前になります。

次の例では、Form スコープを UserDataCFC コンポーネントの addUser メソッドに渡しています。このメソッドでは、各フォームフィールド名をパラメータ名として使用しています。メソッドの本文では、取得したフォームフィールドの内容を使用して、データベースにユーザーを追加できます。

```
<cfinvoke component="UserDataCFC" method="addUser" argumentCollection="#Form#">
```

cfinvokeargument タグによるパラメータの受け渡し

cfinvoke タグの本文でパラメータを渡すには、cfinvokeargument タグを使用します。cfinvokeargument タグを使用すると、たとえば、ユーザーの入力に基づいて異なるパラメータを渡す条件処理を作成できます。

次の例では、corpQuery コンポーネントを呼び出しています。

```
<cfinvoke component="corpQuery" method="getEmp">
    <cfinvokeargument name="lastName" value="Wilder">
</cfinvoke>
```

cfinvokeargument タグで、lastName パラメータをコンポーネントメソッドに渡しています。

次の例では、生成するレポートをユーザーがフォームで既に選択しています。このアクションページでは、getdata および reports コンポーネントをインスタンス化した後に、doquery コンポーネントインスタンスを呼び出しています。このインスタンスは、queryall にクエリー結果を返します。次に、doreport コンポーネントインスタンスを呼び出し、cfinvokeargument タグを使用してクエリー結果を doreport インスタンスに渡しています。このインスタンスで出力が生成されます。

```
<cfobject component="getdata" name="doquery">
<cfobject component="reports" name="doreport">
    <cfinvoke component="#doquery#" method="#form.whichreport#" returnvariable="queryall">
    <cfinvoke component="#doreport#"method="#form.whichreport#">
        <cfinvokeargument name="queryall" value="#queryall#">
    </cfinvoke>
```

メソッドの直接呼び出しでのパラメータの受け渡し

メソッドの直接呼び出しで CFC メソッドにパラメータを渡すには、次の 3 つの方法があります。

1 次の CFScript の例のように、カンマ区切りの name="value" エントリの形式でパラメータを渡します。

```
authorized = securityCFC.getAuth(name="Almonzo", Password="LauRa123");
```

2 argumentCollection 構造体にパラメータを格納して渡します。次のコードは、前の例と同等です。

```
argsColl = structNew();
argsColl.username = "Almonzo";
argsColl.password = "LauRa123";
authorized = securityCFC.getAuth(argumentCollection = argsColl);
```

3 各パラメータをカンマで区切った定位置パラメータを渡します。次の例では、getAuth メソッドを呼び出し、名前とパスワードを定位置パラメータとして渡しています。

```
authorized = securityCFC.getAuth("Almonzo", "LauRa123");
```

注意：ColdFusion 関数で定位置パラメータおよびコンポーネントメソッドを使用する方法については、150 ページの「ユーザー定義関数の作成」を参照してください。

URL でのパラメータの受け渡し

CFC メソッドに URL でパラメータを渡すことができます。それには、次に示すように、URL クエリー文字列の標準のシンタックスである名前と値のペアを URL に追加します。

```
http://localhost:8500/CompanyQuery.cfc?method=getEmp&lastName=Adams
```

CFC 変数とスコープ

CFC では、ColdFusion スコープやローカル変数を使用できます。

注意：コンポーネントで使用される Super キーワードもスコープとして扱われることがあります。Super キーワードの詳細については、197 ページの「Super キーワードの使用」を参照してください。

This スコープ

This スコープは CFC 内で使用でき、すべての CFC メソッドによって共有されます。また、基本 (親) コンポーネント (CFC が子コンポーネントの場合)、CFC をインスタンス化するページ、および CFC によってインクルードされるすべての CFML ページでも This スコープが使用できます。

CFC 内では、次のように This という接頭辞を使用して、This スコープ変数の定義やアクセスを行います。

```
<cfset This.color="green">
```

呼び出しページでは、CFC のインスタンス名を接頭辞として使用すれば、CFC の This スコープ変数の定義やアクセスが行えます。たとえば、car という名前の CFC があり、その中で <cfset This.color="green"> と指定している場合、その CFC をインスタンス化した ColdFusion ページでは、その CFC の color プロパティを #car.color# として参照できます。

This スコープの変数値は、CFC インスタンスが存在する限り存続します。したがって、CFC インスタンスメソッドの複数の呼び出しにまたがって保持されます。

注意：This スコープ識別子は、JavaScript および ActionScript の This キーワードに機能が似ています。CFC は Java クラスモデルには準拠しておらず、ColdFusion と Java では This キーワードの動作が異なります。This は、Java ではプライベートスコープですが、ColdFusion ではパブリックスコープです。

Variables スコープ

CFC の Variables スコープは、CFC のプライベートスコープです。このスコープには、CFC 本文 (初期化コードまたはコンストラクタコード) および CFC メソッドで定義されている変数が含まれます。CFC 内で設定されている Variables スコープの変数には、CFC を呼び出したページからはアクセスできません。

CFC の Variables スコープには、CFC のインスタンス化や呼び出しを行うページの Variables スコープで定義または保持されている変数は含まれません。ただし、Variables を引数として CFC メソッドに渡せば、CFC を呼び出したページの Variables スコープに CFC からアクセスできます。

Variables スコープの変数を設定するには、Variables 接頭辞を付けた名前か、または接頭辞の付いていない名前に値を代入します。

Variables スコープの値は、CFC インスタンスが存在する限り存続します。したがって、CFC インスタンスメソッドの複数の呼び出しにまたがって保持されます。

Variables スコープはインクルードページで使用でき、インクルードページで宣言されている Variables スコープの変数はコンポーネントページで使用できます。

注意：Variables スコープは、関数ローカルスコープとは異なります。関数ローカルスコープは関数内のプライベート変数を宣言するために使用します。関数ローカルの変数を定義するには、常に Local スコープ名の var キーワードを使用します。

例 : Variables スコープの共有

次の例では、Variables を引数として CFC メソッドに渡すことによって、CFC を呼び出したページの Variables スコープに CFC からアクセスする方法を示しています。また、Variables スコープが CFC のプライベートスコープであることも示しています。

次の例は "callGreetMe.cfm" ページのコードです。

```
 <cfset Variables.MyName="Wilson">
<cfobject component="greetMe" name="myGreetings">
<cfoutput>
    Before invoking the CFC, Variables.Myname is: #Variables.MyName#.<br>
    Passing Variables scope to hello method. It returns: #myGreetings.hello(Variables.MyName)#.<br>
After invoking the CFC, Variables.Myname is: #Variables.MyName#.<br>
</cfoutput>
<cfinvoke component="greetMe" method="VarScopeInCfc">
```

次の例は greetMe CFC のコードです。

```
 <cfcomponent>
<cfset Variables.MyName="Tuckerman">
    <cffunction name="hello">
        <cfargument name="Name" Required=true>
        <cfset Variables.MyName="Hello " & Arguments.Name>
        <cfreturn Variables.MyName>
    </cffunction>
    <cffunction name="VarScopeInCfc">
        <cfoutput>Within the VarScopeInCfc method, Variables.MyName is: #variables.MyName#<br></cfoutput>
    </cffunction>
</cfcomponent>
```

この例では、"callGreetMe.cfm" ページは次の処理を行います。

1. そのページの Variables スコープの MyName 変数に **Wilson** を設定します。
2. **Variables.MyName** の値を表示します。
3. greetMe CFC を呼び出し、Variables スコープをパラメータとして渡します。
4. greetMe CFC によって返された値を表示します。
5. **Variables.MyName** の値を表示します。
6. VarScopeInCfc メソッドを呼び出します。これにより、CFC 内での **Variables.MyName** の値が表示されます。

ブラウザで "callGreetMe.cfm" ページを表示すると、次のように表示されます。

```
 Before invoking the CFC, Variables.Myname is: Wilson.
Passing Variables scope to hello method. It returns: Hello Wilson.
After invoking the CFC, Variables.Myname is: Wilson.
Within the VarScopeInCfc method, Variables.MyName is: Tuckerman
```

Arguments スコープ

Arguments スコープはメソッド内にのみ存在し、メソッドの外で使用することはできません。このスコープには、次の方法で渡された変数などの、メソッドに渡された変数が含まれています。

- cfinvoke タグに属性名を指定して渡された変数
- cfinvoke タグの cfargumentcollection 属性に渡された変数
- cfinvokeargument タグに渡された変数
- Web サービス、Flash Remoting、直接 URL、フォームの送信のいずれかを使用してメソッドが呼び出されたときに、属性またはパラメータとしてメソッドに渡された変数

Arguments スコープの変数にアクセスするには、構造体表記法 (Arguments.variablename) または配列表記法 (Arguments[1] または Arguments["variablename"]) を使用します。

Arguments スコープは、CFC メソッドの複数の呼び出しにまたがって保持されません。

Arguments スコープ内の変数は、メソッドでインクルードしたページで使用できます。

他の変数スコープ

CFC は、Form、URL、Request、CGI、Cookie、Client、Session、Application、Server、および Flash スコープを呼び出しページと共有します。これらのスコープ内の変数は、CFC でインクルードしたすべてのページでも使用できます。これらの変数には、CFC に特有の動作はありません。

関数のローカル変数

cffunction タグまたは CFScript の function 定義の中で Var キーワードを使用して宣言した変数は、その変数を定義したメソッド内でのみ使用でき、そのメソッドが呼び出されてから結果が返されるまでの間のみ存在します。関数定義の外で Var キーワードを使用することはできません。

注意：宣言している関数内でしか使用しない変数には、常に Var キーワードまたは Local スコープ名を使用します。

関数のすべてのローカル変数は、どの CFML コードよりも前に、関数定義の冒頭に定義します。次に例を示します。

```
<cffunction ...>
    <cfset Var testVariable = "this is a local variable">
        <!--- Function code goes here. --->
    <cfreturn myresult>
</cffunction>
```

cfargument タグによるすべての引数宣言は、cfset タグによるどの変数定義よりも前に行う必要があります。また、最初に cfscript タグを使用し、宣言する変数を Var キーワードを使用してスクリプトで定義することもできます。

Session スコープなどの永続スコープに CFC を格納し、関数が終了したら解放する必要のあるデータが関数に存在する場合は、関数のローカル変数を使用します。

ローカル変数は、CFC メソッドの複数の呼び出しにまたがって保持されることはありません。

ローカル変数は、メソッドでインクルードしたページで使用できます。

CFC の効果的な使用

このセクションでは、CFC を効果的に使用するテクニックについて説明します。

- コードの構築と再利用
- セキュリティ保護された CFC の構築
- イントロスペクションによるコンポーネント情報の取得

コードの構築と再利用

コンポーネントの継承と Super キーワードは、オブジェクト指向の構造化された ColdFusion コンポーネントを作成するための重要な機能です。

コンポーネントの継承 既存のコンポーネント (基本コンポーネントと呼びます) を再利用して、複数のコンポーネント (サブクラスまたはサブコンポーネントと呼びます) を派生させることができます。通常は、基本コンポーネントで一般的な性質を定義し、サブコンポーネントでそれぞれの目的に特化した性質を定義します。サブクラスでは、基本コンポーネントのコードを再定義せずに継承することも、必要に応じてコードを上書き (オーバーライド) することもできます。

Super キーワード 基本コンポーネントのメソッドをオーバーライドしたコンポーネントの中で、基本コンポーネントの元のメソッドを実行できます。したがって、サブクラス化したコンポーネントでメソッドをオーバーライドしても、元のメソッドが利用できなくなることはありません。

コンポーネントの継承の使用

コンポーネントの継承を使用すれば、あるコンポーネントのメソッドやプロパティを、別のコンポーネントにインポートできます。継承によって作成されたコンポーネントは、元のコンポーネントのメソッドやプロパティを共有します。親の CFC を拡張 (extends) した CFC をインスタンス化すると、親の CFC のインスタンスデータが初期化されます。

コンポーネントの継承とは、コンポーネントの間に **is a** 関係を定義することと考えられます。たとえば、"president.cfc" というコンポーネントが "manager.cfc" からメソッドとプロパティを継承し、その "manager.cfc" が "employee.cfc" からメソッドとプロパティを継承しているとします。この場合は、"president.cfc" **is a** "manager.cfc" ("president.cfc" は "manager.cfc" の一種である) という関係、および "manager.cfc" **is a** "employee.cfc" ("manager.cfc" は "employee.cfc" の一種である) という関係が成り立っています。同時に、"president.cfc" **is a** "employee.cfc" ("president.cfc" は "employee.cfc" の一種である) という関係も成り立っています。

この例では、"employee.cfc" が基本コンポーネントであり、他のコンポーネントのベースになっています。manager コンポーネントは、employee コンポーネントを拡張 (extends) したもので、employee コンポーネントのすべてのメソッドおよびプロパティを持ち、さらに独自のメソッドおよびプロパティも持っています。president コンポーネントは、manager コンポーネントを拡張 (extends) したものです。president コンポーネントは、manager コンポーネントのサブコンポーネントまたは子コンポーネントと呼ばれます。同様に、manager コンポーネントは、employee コンポーネントの子コンポーネントです。

1 次の内容の "employee.cfc" ファイルを作成します。

```
<cfcomponent>
    <cfset This.basesalary=40*20>
</cfcomponent>
```

2 次の内容の "manager.cfc" ファイルを作成します。

```
<cfcomponent extends="employee">
    <cfset This.mgrBonus=40*10>
</cfcomponent>
```

この例では、cfcomponent タグの extends 属性で employee コンポーネントを指定しています。

3 次の内容の "president.cfc" ファイルを作成します。

```
<cfcomponent extends="manager">
    <cfset This.prezBonus=40*20>
</cfcomponent>
```

この例では、cfcomponent タグの extends 属性で manager コンポーネントを指定しています。

4 次の内容の "inherit.cfm" ファイルを作成し、前の手順で作成したコンポーネントと同じディレクトリに保存します。

```
<cfobject name="empObj" component="employee">
<cfobject name="mgrObj" component="manager">
<cfobject name="prezObj" component="president">
<cfoutput>
    An employee's salary is #empObj.basesalary# per week.<br>
    A manager's salary is #mgrObj.basesalary + mgrObj.mgrBonus# per week.<br>
    A president's salary is #prezObj.basesalary + prezObj.mgrBonus +
        prezObj.PrezBonus# per week.
</cfoutput>
```

ブラウザで "inherit.cfm" ファイルを表示すると、manager コンポーネントは、基本コンポーネントの "employee.cfc" で定義されている basesalary を参照します。president コンポーネントは、employee コンポーネントで定義されている basesalary と、manager コンポーネントで定義されている mgrBonus の両方を参照します。manager コンポーネントは、president コンポーネントの親クラスです。

"component.cfc" ファイルの使用

すべての CFC は、ColdFusion の "WEB-INF/cftags/component.cfc" コンポーネントを自動的に拡張します。"WEB-INF" ディレクトリは、埋め込み J2EE サーバーを使用して設定された ColdFusion では、"<ColdFusion のルートディレクトリ >/wwwroot" にあります。J2EE サーバー上にデプロイした ColdFusion では、<ColdFusion のルートディレクトリ > にあります。この CFC は、長さゼロのファイルとして配布されます。これを使用すれば、ColdFusion アプリケーションサーバーインスタンス内のすべての CFC に継承させるコアメソッドやプロパティを定義できます。

注意：新しいバージョンの ColdFusion をインストールすると、既存の "component.cfc" ファイルが新しいバージョンに置き換えられます。したがって、アップグレードする前に、"component.cfc" ファイルに追加したすべてのコードを保存し、アップグレード後の新しい "component.cfc" ファイルにそれらのコードをコピーしてください。

Super キーワードの使用

Super キーワードは、Extends 属性を使用して別の CFC を拡張している CFC でのみ使用します。ColdFusion スコープとは異なり、Super キーワードは変数には使用せず、CFC メソッドにのみ使用します。CFC を呼び出した ColdFusion ページでは使用できません。

Super キーワードを使用すれば、現在の CFC の親の CFC で定義されているメソッドを参照することができます。たとえば、employee、manager、および president CFC にはそれぞれ getPaid() メソッドが含まれています。manager CFC は、employee CFC を拡張したものです。したがって、manager CFC でメソッド名に接頭辞 Super を付ければ、オーバーライドした getPaid() メソッドの元のバージョン (employee CFC で定義されている getPaid() メソッド) を使用できます。

1 次の内容の "employee.cfc" ファイルを作成します。

```
<cfcomponent>
    <cffunction name="getPaid" returntype="numeric">
        <cfset var salary=40*20>
        <cfreturn salary>
    </cffunction>
</cfcomponent>
```

2 次の内容の "manager.cfc" ファイルを作成します。

```
<cfcomponent extends="employee">
    <cffunction name="getPaid" returntype="numeric">
        <cfset var salary=1.5 * Super.getPaid()>
        <cfreturn salary>
    </cffunction>
</cfcomponent>
```

3 次の内容の "president.cfc" ファイルを作成します。

```
<cfcomponent extends="manager">
    <cffunction name="getPaid" returntype="numeric">
        <cfset var salary=1.5 * Super.getPaid()>
        <cfreturn salary>
    </cffunction>
</cfcomponent>
```

4 次の内容の "payday.cfm" ファイルを作成し、前の手順で作成したコンポーネントと同じディレクトリに保存します。

```
<cfobject name="empObj" component="employee">
<cfobject name="mgrObj" component="manager">
<cfobject name="prezObj" component="president">
<cfoutput>
    <cfoutput>
        An employee earns #empObj.getPaid()#.<br>
        A manager earns #mgrObj.getPaid()#.<br>
        The president earns #prezObj.getPaid()#.
    </cfoutput>
</cfoutput>
```

この例では、子コンポーネントの getPaid() メソッドが、親コンポーネントの getPaid() メソッドを呼び出しています。子の getPaid() メソッドは、親の getPaid() メソッドから返された給与を使用して、適切な金額を算出しています。

インクルードページでは Super キーワードを使用できます。

注意：Super キーワードでさかのぼれるのは、1 レベルの継承のみです。複数レベルの継承を使用している場合であっても、Super キーワードでアクセスできるのは、現在のコンポーネントの直接の親のみです。このセクションの例では、この制限に対処するために、メソッドをチェーン状に呼び出しています。

コンポーネントパッケージの使用

同じディレクトリに保管されているコンポーネントは、コンポーネントパッケージを構成するメンバーです。コンポーネントパッケージを使用すれば、名前の競合が回避でき、コンポーネントをデプロイしやすくなります。

- ColdFusion は、まず現在のディレクトリで CFC を検索します。2 つのコンポーネントが単一ディレクトリにパッケージとして格納されており、修飾パスではなくコンポーネント名のみを使用して一方のコンポーネントからもう一方のコンポーネントを参照した場合、ColdFusion は常に、まずパッケージディレクトリでコンポーネントを検索します。したがって、アプリケーションのコンポーネントをパッケージに構造化すれば、コンポーネントコードを共有することなく、同じコンポーネント名を使用できます。
- メソッドの cffunction タグで access="package" 属性を使用すると、メソッドへのアクセスは同じパッケージ内のコンポーネントに制限されます。他のパッケージのコンポーネントは、完全修飾コンポーネント名で指定しても、このメソッドを使用できません。アクセスセキュリティの詳細については、199 ページの「アクセスセキュリティの使用」を参照してください。

cfinvoke タグによるパッケージ化されたコンポーネントメソッドの呼び出し

1. Web ルートディレクトリに、**appResources** という名前のディレクトリを作成します。
2. "appResources" ディレクトリに、**components** という名前のディレクトリを作成します。
3. 185 ページの「CFC インスタンスのメソッドの呼び出し」で作成した "tellTime2.cfc" ファイルと、176 ページの「別ファイルへの実行可能コードの配置」で作成した "getUTCTime.cfm" ファイルを、"components" ディレクトリにコピーします。
4. 次の内容の "timeDisplay.cfm" ファイルを作成し、Web ルートディレクトリに保存します。

```
<!--- Create the component instance. --->
<cfobject component="appResources.components.tellTime2" name="tellTimeObj">
<!--- Invoke the methods. --->
<cfinvoke component="#tellTimeObj#" method="getLocalTime"
    returnvariable="localTime" >
<cfinvoke component="#tellTimeObj#" method="getUTCTime"
    returnvariable="UTCTime" >
<!--- Display the results. --->
<h3>Time Display Page</h3>
<cfoutput>
    Server's Local Time: #localTime#<br>
    Calculated UTC Time: #UTCTime#
</cfoutput>
```

   ディレクトリ構造の区切り文字にはドット (.) を使用します。コンポーネント名の前にディレクトリ名を付けます。

5. "timeDisplay.cfm" ファイルをブラウザで開きます。

次の例では、CFScript を使用して呼び出します。

```
<cfscript>
helloCFC = createObject("component", "appResources.components.catQuery");
helloCFC.getSaleItems();
</cfscript>
```

次の例では、URL を使用して呼び出します。

```
http://localhost/appResources/components/catQuery.cfc?method=getSalesItems
```

永続スコープでの CFC の使用

CFC インスタンスは、Session または Application スコープに置くことができます。これによって、そのスコープが存在する限り、コンポーネントのプロパティが保持されます。たとえば、ショッピングカートアプリケーションで CFC を使用して、ユーザーのセッションが続いている間ショッピングカートの内容を保持するとします。ショッピングカート CFC を Session スコープに置けば、コンポーネントのプロパティを使用してカートの内容を保存できます。たとえば次の行は、Session スコープに shoppingCart コンポーネントのインスタンスを作成します。

```
<cfobject name="Session.myShoppingCart" component="shoppingCart">
```

永続スコープの CFC プロパティを操作するコードは、永続スコープを操作する通常のコードと同様に、ロックする必要があります。したがって、次のタイプのアプリケーションコードは、両方ともロックする必要があります。

- 永続スコープの CFC インスタンスのプロパティを直接操作するコード
- 永続スコープの CFC インスタンスのプロパティを操作するインスタンスメソッドを呼び出すコード

単一の永続スコープに複数の CFC インスタンスを置く場合は、各 CFC インスタンスに名前付きロックを作成できます。ロックの詳細については、300 ページの「永続データとロックの使用」を参照してください。

注意：Session スコープの CFC はシリアル化できないので、たとえばサーバー間のセッションフェイルオーバーなどのために、それらをクラスタ化されたセッションに使用することはできません。

セキュリティ保護された ColdFusion コンポーネントの構築

ColdFusion コンポーネントでは、アクセスセキュリティ、ロールベースのセキュリティ、またはプログラムセキュリティを使用して、コンポーネントメソッドへのアクセスを制限できます。

アクセスセキュリティの使用

CFC のアクセスセキュリティを使用して、コンポーネントにアクセスできるコードを制限できます。CFC メソッドへのアクセスを指定するには、次のように cffunction の access 属性を指定します。

タイプ	説明
private	そのメソッドが宣言されているコンポーネント、およびそのコンポーネントを拡張したコンポーネントでのみ使用できます。これは、Java の private キーワードではなく protected キーワードに似ています。
package	そのメソッドが宣言されているコンポーネント、そのコンポーネントを拡張したコンポーネント、またはパッケージ内の他のコンポーネントでのみ使用できます。パッケージは、単一のディレクトリに定義されているすべてのコンポーネントで構成されます。パッケージの詳細については、198 ページの「コンポーネントパッケージの使用」を参照してください。
public	ローカルで実行中の ColdFusion ページまたはコンポーネントメソッドで使用できます。
remote	ローカルまたはリモートで実行中の ColdFusion ページまたはコンポーネントメソッドで使用できます。また、ローカルまたはリモートのクライアントで、URL、フォーム送信、Flash Remoting、または Web サービスを介して使用できます。

ロールベースのセキュリティの使用

cffunction タグで roles 属性を指定すれば、指定のロールでログインしたユーザーのみがメソッドを実行できるようになります。アクセスを許可されていないユーザーがメソッドを呼び出そうとすると、例外が返されます。

次の例では、ファイルを削除するコンポーネントメソッドを作成します。

```
<cfcomponent>
    <cffunction
         name="deleteFile" access="remote" roles="admin,manager" output="no">
        <cfargument name="filepath" required="yes">
        <cffile action="DELETE" file=#arguments.filepath#>
    </cffunction>
</cfcomponent>
```

この例の cffunction タグでは、roles 属性を使用して、アクセスを許可するユーザーロールを指定しています。この例では、ロールが admin および manager のユーザーのみがこの関数にアクセスできます。複数のロールはカンマで区切ります。

ColdFusion の cflogin タグやロールベースセキュリティなどの、ColdFusion のセキュリティの詳細については、337 ページの「アプリケーションの保護」を参照してください。

プログラムセキュリティの使用

メソッド内に独自のセキュリティを実装して、リソースを保護することもできます。たとえば、ColdFusion 関数の IsUserInAnyRole を使用して、ユーザーが特定のロールに属しているかどうかを調べることができます。次に例を示します。

```
<cffunction name="foo">
    <cfif IsUserInRole("admin")>
        … do stuff allowed for admin
    <cfelseif IsUserInRole("user")>
        … do stuff allowed for user
    <cfelse>
        <cfoutput>unauthorized access</cfoutput>
        <cfabort>
    </cfif>
</cffunction>
```

イントロスペクションによるコンポーネント情報の取得

ColdFusion では、次のような方法でコンポーネントに関する情報を取得できます。

- ブラウザでのコンポーネントページのリクエスト
- ColdFusion コンポーネントブラウザの使用
- Adobe® Dreamweaver® の [コンポーネント] パネルの使用
- GetMetaData 関数の使用

コンポーネントに関するこれらの情報は、開発を行うときに、最新の API リファレンスとして活用できます。

注意：イントロスペクションで表示する説明を CFC に含める方法については、182 ページの「CFC に関する説明を含める」を参照してください。

ブラウザでのコンポーネントページのリクエスト

Web ブラウザで、コンポーネントメソッドを指定せずに直接 CFC にアクセスすると、次の処理が実行されます。

1. "<ColdFusion のルートディレクトリ >/wwwroot/CFIDE/componentutils" にある "cfcexplorer.cfc" ファイルにリクエストが転送されます。
2. 必要に応じて、ColdFusion RDS または Administrator のパスワードが cfcexplorer コンポーネントによって要求されます。
3. cfcexplorer コンポーネントによって HTML の記述が生成され、ブラウザに返されます。

ColdFusion コンポーネントブラウザの使用

ColdFusion で使用できるコンポーネントは、"<ColdFusion のルートディレクトリ >/wwwroot/CFIDE/componentutils/componentdoc.cfm" にあるコンポーネントブラウザを使用してブラウズすることもできます。

このブラウザには 3 つのペインがあります。

- 左上のペインには、ColdFusion がアクセスできるすべての CFC パッケージがリストされます。また、[すべてのコンポーネント] および [更新] のリンクがあります。
- 左下のペインには、CFC コンポーネント名がリストされます。ブラウザが最初に表示されたときや、上のペインで [すべてのコンポーネント] リンクをクリックした場合は、使用可能なすべてのコンポーネントが下のペインにリストされます。左上のペインでパッケージ名をクリックした場合は、そのパッケージ内のコンポーネントのみが下のペインにリストされます。
- 右側のペインには、最初はすべてのコンポーネントのパスがリストされます。左下のペインでコンポーネント名をクリックすると、200 ページの「ブラウザでのコンポーネントページのリクエスト」の説明のように、ColdFusion のイントロスペクションページが右側のペインに表示されます。

注意：RDS ユーザー名が有効な場合、コンポーネントブラウザでは、管理者または RDS の単一パスワードを持つルート管理者ユーザー (admin) が受け入れられます。

Dreamweaver の [コンポーネント] パネルの使用

Dreamweaver の [コンポーネント] パネルには、使用可能なすべてのコンポーネントとそのメソッド、メソッドパラメータ、およびプロパティがリストされます。パネルのコンテキストメニューには、コンポーネントの作成、選択したコンポーネントの編集、コンポーネントを呼び出すコードの挿入、およびコンポーネントやコンポーネント要素の詳細情報の表示を行うためのオプションがあります。[詳細の取得] オプションを選択すると、200 ページの「ブラウザでのコンポーネントページのリクエスト」の説明のように、ColdFusion イントロスペクションページが表示されます。Dreamweaver での CFC の表示および編集の詳細については、Dreamweaver のオンラインヘルプを参照してください。

GetMetaData 関数の使用

CFML の GetMetaData 関数は、CFC インスタンスのすべてのメタデータを含んだ構造体を返します。この構造体を使用すれば、cfdump タグを使用するよりも CFC に関するデータを多く取得できます。これには、次の情報が含まれます。

- コンポーネントタグのすべての属性 (メタデータ専用の属性など) およびコンポーネントパス。
- コンポーネント内の各メソッド (関数) に関する詳細情報を含んだ構造体の配列。この情報には、メタデータ専用関数やパラメータ属性などの、すべての属性が含まれます。
- 各関数の構造体の Parameters 要素。cfargument タグで指定されているパラメータの配列が含まれます。各パラメータの情報には、任意のメタデータ専用属性も含まれます。
- cfproperty タグで指定されているプロパティに関する情報。

CFC のメタデータの表示

1 次のコードを使用して、"telltime.cfc" と同じディレクトリに "tellAboutCfcs.cfm" ファイルを作成します。

```
<!--- Create an instance of the component. --->
<cfobject component="tellTime" name="tellTimeObj">
<!--- Create a structure. --->
<cfset aboutcfc=structNew()>
<!--- Populate the structure with the metadata for the
            tellTimeObj instance of the tellTime CFC. --->
<cfset aboutcfc=GetMetaData(tellTimeObj)>
<cfdump var="aboutcfc">
```

2 ブラウザで "tellAboutCfcs.cfm" ファイルを表示します。

コンポーネントタグの使用方法やメタデータ専用属性の指定方法など、CFC メタデータの指定方法の詳細については、182 ページの「CFC に関する説明を含める」を参照してください。

ColdFusion コンポーネントの例

『ColdFusion アプリケーションの開発』のコード例では、コード、特にクエリーを再利用しているものがいくつかあります。それらのコード例では、CFC のメリットを示すために、次に示す CFC の適切なメソッドを呼び出しています。構造化された再利用可能なコードを作成する場合には CFC を使用するのが一般的ですが、本マニュアルの一部のコード例では、ColdFusion の特定の要素を明示するために、CFC を呼び出すのではなく、CFML ページ内でクエリーを実行していることがあります。

```
<cfcomponent>
    <cffunction name="allemployees" access="public" output="false"
        returntype="query">
        <cfset var getNames="">
        <cfquery name="getNames" datasource="cfdocexamples">
            SELECT * FROM Employee
        </cfquery>
    </cffunction>

    <cffunction name="namesalarycontract" access="public" output="false"
        returntype="query">
        <cfset var EmpList="">
        <cfquery name="EmpList" datasource="cfdocexamples">
            SELECT Firstname, Lastname, Salary, Contract
            FROM Employee
        </cfquery>
    </cffunction>

    <cffunction name="fullname" access="public" output="false"
        returntype="query">
        <cfset var engquery="">
        <cfquery name="engquery" datasource="cfdocexamples">
            SELECT FirstName || ' ' || LastName AS FullName
            FROM Employee
        </cfquery>
    </cffunction>

    <cffunction name="bydept" access="public" output="false" returntype="query">
        <cfset var deptquery="">
        <cfquery name="deptquery" datasource="cfdocexamples">
            SELECT Dept_ID, FirstName || ' ' || LastName
            AS FullName
            FROM Employee
            ORDER BY Dept_ID
        </cfquery>
    </cffunction>

    <cffunction name="employeebyURLID" access="public" output="false"
        returntype="query">
        <cfset var GetRecordtoUpdate="">
        <cfquery name="GetRecordtoUpdate" datasource="cfdocexamples">
            SELECT * FROM Employee
            WHERE Emp_ID = #URL.Emp_ID#
        </cfquery>
```

```
    </cffunction>

    <cffunction name="deleteemployee" access="public" output="false"
        returntype="void">
        <cfset var DeleteEmployee="">
        <cfquery name="DeleteEmployee" datasource="cfdocexamples">
            DELETE FROM Employee
            WHERE Emp_ID = #Form.Emp_ID#
        </cfquery>
    </cffunction>

    <cffunction name="distinctlocs"access="public" output="false"
        returntype="query">
        <cfset var GetDepartments="">
        <cfquery name="GetDepartments" datasource="cfdocexamples">
            SELECT DISTINCT Location
            FROM Departmt
        </cfquery>
    </cffunction>
</cfcomponent>
```

CFC の暗黙のコンストラクター

CFC をインスタンス化するときに、キーと値のペアまたは構造体として値を渡すことにより、CFC のプロパティを初期化できます。

例えば、プロパティ firstName および lastName を持つ CFC employee.cfc があるものとします。

次のシンタックスを使用して、名と姓を設定できます。

```
emp=new employee(firstName="Tom", lastName="Nash");
```

または、次のように暗黙の構造体を使用できます。

```
emp=new Employee({firstName="Tom", lastName="Nash"});
```

init() メソッドを明示的に定義するか、またはコンポーネントの initmethod 属性でメソッドを提供することにより、init() メソッドが CFC に対して定義されている場合は、暗黙のコンストラクターは呼び出されません。

暗黙のコンストラクターをサポートする場合は、accessors=true を cfcomponent で指定するか、または定義されているプロパティに setter 関数を用意します。

特定のプロパティに setter = false を指定した場合、または CFC に対してプロパティが定義されていない場合、値は設定されません。

どちらの場合も、onMissingMethod が CFC に対して定義されている場合は、それが呼び出されます。

例 1

次の例では、CFC のインスタンス化時にプロパティ firstName は値 Tom に設定されますが、プロパティ lastName は setter = false が指定されているので設定されません。また、プロパティ lastName の場合は onMissingMethod が呼び出されます。

employee.cfm

```
<cfscript>
    emp = new employee(firstname="Tom", lastname="Nash");
    writeOutput("<u><b>Employee Details</b></u>: " & "<br><br>");
    writeOutput("First Name: " & emp.getFirstname() & "<br>");
</cfscript>
```

employee.cfc

```
<cfcomponent accessors="TRUE">
    <cfproperty name="firstname" type="string" setter="true"/>
    <cfproperty name="lastname" type="string" setter="false"/>
    <cfproperty name="age" type="numeric"/>
    <cffunction name="onMissingMethod">
        <cfargument name="missingMethodName"/>
        <cfargument name="missingMethodArguments"/>

        <cfoutput>
            onMissingMethod() called for method call - #arguments.missingMethodName#
            <hr>
        </cfoutput>
    </cffunction>
</cfcomponent>
```

例 2

次の例では、employee1.cfc と employee2.cfc の両方に init() メソッドが定義されています。したがって、CFC のプロパティ firstName および lastName は初期化されません。しかし、employee.cfc では、init() メソッドが定義されていないため、プロパティ firstName は初期化されます。

employee1.cfc

```
<cfcomponent accessors="TRUE">
    <cfproperty name="firstname" type="string" setter="true"/>
    <cfproperty name="lastname" type="string" setter="false"/>
    <cffunction name="init">
        <cfreturn this>
    </cffunction>
    <cffunction name="onMissingMethod">
        <cfargument name="missingMethodName"/>
        <cfargument name="missingMethodArguments"/>
        <cfoutput>
            onMissingMethod() called for method call - #arguments.missingMethodName#
            <hr>
        </cfoutput>
    </cffunction>
</cfcomponent>
```

employee2.cfc

```
<cfcomponent accessors="TRUE" initmethod=foo>
    <cfproperty name="firstname" type="string" setter="true"/>
    <cfproperty name="lastname" type="string" setter="false"/>
    <cffunction name="foo">
        <cfreturn this>
    </cffunction>
    <cffunction name="onMissingMethod">
        <cfargument name="missingMethodName"/>
        <cfargument name="missingMethodArguments"/>
        <cfoutput>
            onMissingMethod() called for method call - #arguments.missingMethodName#
            <hr>
        </cfoutput>
    </cffunction>
</cfcomponent>
```

employee.cfm

```
<cfscript>
    emp = new employee(firstname="Tom", lastname="Nash");
    writeOutput("<u><b>Employee Details</b></u>: " & "<br><br>");
    writeOutput("First Name: " & emp.getFirstname() & "<br>");
    writeOutput("<hr>");
    emp1 = new employee1(firstname="Tom", lastname="Nash");
    writeOutput("First Name: " & emp1.getFirstname() & "<br>");
    writeOutput("<hr>");
    emp2 = new employee2(firstname="Tom", lastname="Nash");
    writeOutput("First Name: " & emp2.getFirstname() & "<br>");
    writeOutput("<hr>");
</cfscript>
<cfoutput>
    onMissingMethod() called for method call - #arguments.missingMethodName#
    <hr>
</cfoutput>
```

CFC メソッドのメソッド連結

次のようにして CFC メソッドを連結できます。

```
emp=new employee();
emp.setFirstName("Tom").setLastName("Nash").setAge("30");
```

連結は次の条件でのみ動作します。

- 属性 accessors が true に設定されている場合
- メソッドが見つかるまで
- エラーがない場合

または

- プロパティの setter 関数が定義されている場合

例

次の例の連結が動作するのは、lastName までです。これは、age の setter が false に設定されているためです。

```
<cfcomponent accessors="TRUE">
    <cfproperty name="firstname" type="string" setter="true"/>
    <cfproperty name="lastname" type="string" setter="true"/>
    <cfproperty name="age" type="numeric" setter="false"/>
    <cffunction name="init">
        <cfreturn this>
    </cffunction>
</cfcomponent>
```

CFC の暗黙の表記法

この機能を使用すると、setter を指定することなく、単純変数への代入として CFC プロパティを設定できます。つまり、プロパティの setter メソッドを呼び出すのではなく、プロパティのフィールドを設定することで、プロパティを設定できます。

同様に、プロパティの getter を呼び出すのではなく、プロパティの名前を参照することによって、プロパティの値にアクセスできます。

例

Application.cfc

```
component
{
    this.name = "MyApplication";
    this.invokeImplicitAccessor = true;
}
```

employee.cfm

```
<cfscript>
    emp = new emp();
    emp.firstname = "Tom";
    emp.lastname = "Nash";
    emp.age = 30;
    writeOutput("First Name = " & emp.firstname & "<br>");
    writeOutput("last Name = " & emp.emp.lastname & "<br>");
    writeOutput("Age = " & emp.age & "<br>");
</cfscript>
```

employee.cfc

```
<cfcomponent accessors="TRUE">
    <cfproperty name="firstname" type="string" setter="true"/>
    <cfproperty name="lastname" type="string" setter="false"/>
    <cfproperty name="age" type="numeric"/>
    <cffunction name="onMissingMethod">
        <cfargument name="missingMethodName"/>
        <cfargument name="missingMethodArguments"/>
        <cfoutput>
            onMissingMethod() called for method call -#arguments.missingMethodName#
            <hr>
        </cfoutput>
    </cffunction>
</cfcomponent>
```

この例では、Application.cfc で invokeImplicitAccessor を true に設定した場合にのみ、CFC の暗黙の表記法が機能します。それ以外の場合は、値はコンポーネントの This スコープに Post されます。

カスタム CFML タグの作成と使用

共通のコードをカプセル化するカスタム CFML タグを使用して、CFML を拡張することができます。

カスタムタグの作成

カスタムタグを使用すると、ColdFusion で提供されているタグに独自のタグを追加して、CFML を拡張できます。定義したカスタムタグは、cfquery や cfoutput などの標準の CFML タグと同じように、ColdFusion ページの中で使用できます。

カスタムタグを使用すれば、アプリケーションロジックをカプセル化して、任意の ColdFusion ページから参照できます。カスタムタグは、さまざまなプログラミング作業で簡単に利用でき、アプリケーションの開発効率やコードの再利用性を高めることができます。

たとえば、誕生日のメッセージを生成する cf_happybirthday というカスタムタグを作成します。このタグは、ColdFusion ページで次のように使用します。

```
<cf_happybirthday name="Ted Cantor" birthDate="December 5, 1987">
```

このタグが含まれているページを ColdFusion で処理すると、次のメッセージが出力されます。

```
December 5, 1987 is Ted Cantor's Birthday.
Please wish him well.
```

また、カスタムタグは本文と終了タグを持つこともできます。次に例を示します。

```
<cf_happybirthdayMessge name="Ellen Smith" birthDate="June 8, 1993">
    <p> Happy Birthday Ellen!</p>
    <p> May you have many more!</p>
</cf_happybirthdayMessge>
```

このタグによって次のメッセージが出力されます。

```
June 8, 1993 is Ellen Smith's Birthday.
Happy Birthday Ellen!
May you have many more!
```

終了タグの使用の詳細については、214 ページの「終了タグの処理」を参照してください。

カスタムタグの作成と呼び出し

カスタムタグは単一の ColdFusion ページで実装します。作成した ColdFusion ページからカスタムタグを呼び出すには、そのページのファイル名の前に接頭辞 cf_ を挿入します。カスタムタグを参照するページは、呼び出しページと呼ばれます。

1 現在の日付を表示する ColdFusion ページ (カスタムタグページ) を作成します。

```
<cfoutput>#DateFormat(Now())#</cfoutput>
```

2 このファイルに "date.cfm" という名前を付けて保存します。

3 次の内容の ColdFusion ページ (呼び出しページ) を作成します。

```
<html>
<head>
    <title>Date Custom Tag</title>
</head>
<body>

    <!--- Call the custom tag defined in date.cfm --->
    <cf_date>

</body>
</html>
```

4 このファイルに "callingdate.cfm" という名前を付けて保存します。

5 ブラウザで "callingdate.cfm" を表示します。

このカスタムタグは、DD-MMM-YY (日 - 月 - 年) の形式で現在の日付を返します。

この例からもわかるように、CFML でのカスタムタグの作成方法は、ColdFusion ページの作成方法と同じです。HTML だけでなく、CFML の構文もすべて使用できます。ネーミング規則は、開発スタイルに合った方法を自由に使用できます。わかりやすい固有の名前を付けると、だれでも簡単に正しいタグを見つけることができます。

注意：ColdFusion ページではタグ名の大文字と小文字は区別されませんが、UNIX のカスタムタグのファイル名は小文字である必要があります。

カスタムタグページの保存

カスタムタグページは、次のいずれかの場所に保存する必要があります。

- 呼び出しページと同じディレクトリ
- "cfusion¥CustomTags" ディレクトリ
- "cfusion¥CustomTags" ディレクトリのサブディレクトリ
- ColdFusion Administrator で指定したディレクトリ

複数のディレクトリに存在するアプリケーションの間で特定のカスタムタグを共有したい場合は、"cfusion¥CustomTags" ディレクトリに配置します。このディレクトリでは、サブディレクトリを作成してカスタムタグを整理できます。ColdFusion は、カスタムタグが見つかるまで、"CustomTags" ディレクトリ内の各サブディレクトリを反復検索します。

場合によっては、同じ名前のカスタムタグを複数使用することがあります。目的のタグが呼び出されるようにするには、呼び出しページと同じディレクトリにそのカスタムタグをコピーします。または、cfmodule タグの template 属性を使用して、そのカスタムタグの絶対パスを指定します。cfmodule の詳細については、次のセクションを参照してください。

cfmodule タグによるカスタムタグの呼び出し

カスタムタグページの場所を指定したい場合は、cfmessagebox タグを使用してカスタムタグを呼び出します。カスタムタグを使用すると名前の競合が発生するおそれがある場合や、変数を使用してダイナミックにカスタムタグを呼び出す必要がある場合は、cfmodule タグを使用すると便利です。

このタグでは、template 属性と name 属性のいずれかを使用する必要がありますが、両方を同時に使用することはできません。次の表で、cfmodule の基本的な属性について説明します。

属性	説明
template	name 属性を使用しない場合は必須です。cfinclude の template 属性と同じです。この属性は次の操作を行います。 • 呼び出しページのディレクトリを基準とした相対パスを指定します。 • パスの値が "/" で始まる場合は、ColdFusion Administrator で明示的にマッピングされているディレクトリ内で、インクルードファイルが検索されます。 例 : <cfmodule template="../MyTag.cfm"> は、親ディレクトリ内のカスタムタグファイルを参照しています。
name	template 属性を使用しない場合は必須です。CustomTags ルートディレクトリの下のサブディレクトリ名を、ピリオドで区切って指定します。 例 : <cfmodule name="MyApp.GetUserOptions"> では、ColdFusion ルートディレクトリの下の CustomTags¥MyApp にある "GetUserOptions.cfm" ファイルを指定しています。
attributes	カスタムタグの属性

たとえば次のコードは、呼び出しページの親ディレクトリにある "mytag.cfm" ページで定義されているカスタムタグを実行します。

```
<cfmodule template="../mytag.cfm">
```

cfmessagebox タグの使用方法の詳細については、『CFML リファレンス』を参照してください。

cfimport タグによるカスタムタグの呼び出し

cfimport タグを使用して、タグライブラリのディレクトリからカスタムタグをインポートすることができます。次の例では、"myCustomTags" ディレクトリからタグをインポートしています。

```
<cfimport prefix="mytags" taglib="myCustomTags">
```

インポートしたら、次の例のように、インポート時に設定した接頭辞を使用してカスタムタグを呼び出します。

```
<mytags:customTagName>
```

customTagName は、"customTagName.cfm" という名前の ColdFusion アプリケーションページを表しています。タグに属性を渡す場合は、次のように呼び出しの中に含めます。

```
<mytags:custom_tag_name attribute1=val_1 attribute2=val_2>
```

次の例のように、カスタムタグを呼び出すときに終了タグを使用することもできます。

```
<mytags:custom_tag_name attribute1=val_1 attribute2=val_2>
...
</mytags:custom_tag_name>
```

終了タグを持つタグでは、カスタムタグページが 2 回呼び出されます。1 回目は開始タグ、2 回目は終了タグに対して呼び出されます。ColdFusion での終了タグの処理方法、およびカスタムタグで終了タグの処理を記述する方法の詳細については、214 ページの「終了タグの処理」を参照してください。

cfimport タグを使用する利点の 1 つは、カスタムタグのディレクトリ構造を定義して、カテゴリごとにカスタムタグを整理できることです。たとえば、すべてのセキュリティタグを 1 つのディレクトリにまとめ、すべてのインターフェイスタグを別のディレクトリにまとめることができます。これにより、各ディレクトリからタグをインポートして、別の接頭辞を付けることができます。

```
<cfimport prefix="security" taglib="securityTags">
<cfimport prefix="ui" taglib="uiTags">
...
<security:validateUser name="Bob">
...
<ui:greeting name="Bob">
...
```

接頭辞を見れば、どの場所からインポートしたカスタムタグであるかが判別できるので、コードが読みやすくなります。

カスタムタグの保護

ColdFusion のセキュリティフレームワークでは、タグファイルやタグディレクトリに対して個別にアクセス制限を設定できます。これは、チーム開発で役立つ重要な保護機能です。詳細については、『ColdFusion 設定と管理』を参照してください。

既存のカスタムタグへのアクセス

CFML でカスタムタグを作成する前に、Adobe の情報交換サイト (www.adobe.com/go/learn_cfu_cfdevcenter_jp) にアクセスして、無償または有償で提供されているカスタムタグを確認してください。必要なタグが見つかる場合があります。

タグはさまざまなカテゴリに分類されており、フリーウェア、シェアウェア、または商用ソフトウェアとしてダウンロードできます。また、各タグのシンタックスや使用方法も参照できます。このギャラリーには、カスタムタグに関する豊富な背景情報やオンラインディスカッションフォーラムがあります。

先頭に cf_ が付いているタグは、CFML カスタムタグです。先頭に cfx_ が付いているタグは、Java または C++ で作成された ColdFusion Extension です。CFX タグの詳細については、221 ページの「CFXAPI カスタムタグの構築」を参照してください。

必要なタグが見つからない場合は、CFML でカスタムタグを独自に作成できます。

カスタムタグへのデータの受け渡し

カスタムタグにデータを渡せるようにすると、柔軟な処理が行えるようになります。そのためには、タグ属性などのデータを呼び出しページから渡せるカスタムタグを作成します。

カスタムタグとの間の値のやり取り

カスタムタグは独立した ColdFusion ページなので、カスタムタグと呼び出しページの間で、変数などのデータが自動的に共有されることはありません。呼び出しページからカスタムタグにデータを渡すには、標準の HTML タグや CFML タグと同様に、カスタムタグに属性の名前と値のペアを指定します。

たとえば、NameYouEntered 変数の値を cf_getmd タグに渡すには、次のようにカスタムタグを呼び出します。

```
<cf_getmd Name=#NameYouEntered#>
```

複数の属性をカスタムタグに渡すには、次のようにタグ内でスペースを使用して属性を区切ります。

```
<cf_mytag Firstname="Thadeus" Lastname="Jones">
```

カスタムタグでは、Attributes スコープを使用して、タグに渡された属性にアクセスします。たとえば "getmd.cfm" ページでは、渡された属性を Attributes.Name として参照します。"mytag.cfm" カスタムタグページでは、渡された属性を Attributes.Firstname および Attributes.Lastname として参照します。

また、呼び出しページの変数にカスタムタグページからアクセスすることもできます。それには、呼び出しページのローカル変数の先頭に Caller. を付けます。ただし、この方法でカスタムタグに情報を渡すのはあまりよい方法とはいえません。カスタムタグで使用する特定の名前の変数を、すべての呼び出しページで作成する必要があるからです。属性を使用してパラメータを渡すほうが、柔軟性の高いカスタムタグを作成できます。

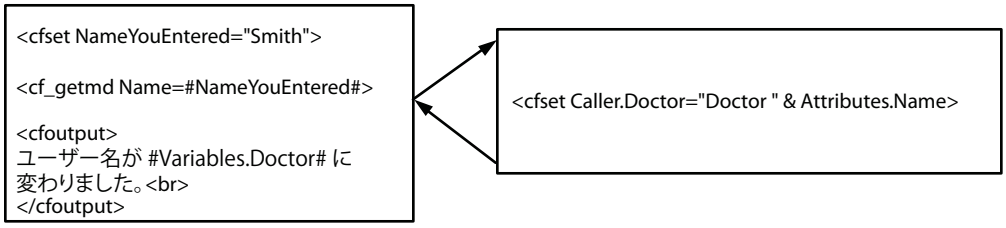
カスタムタグ内で作成した変数は、タグの処理が終了すると削除されます。したがって、呼び出しページに情報を返す場合は、呼び出しページの Caller スコープにその情報を返します。カスタムタグの外からカスタムタグの変数にアクセスすることはできません。

たとえば、呼び出しページの変数 Doctor を設定するには、"getmd.cfm" ページで次のコードを使用します。

```
<cfset Caller.Doctor="Doctor " & Attributes.Name>
```

変数 Doctor が呼び出しページに存在しない場合は、このステートメントによってその変数が作成されます。変数が存在する場合は、カスタムタグによってその変数が上書きされます。

次の図は、呼び出しページの変数とカスタムタグの間の関係を示しています。



A. 呼び出しページ **B.** getmd.cfm

カスタムタグでよく使用される方法の 1 つが、結果を返す変数の名前を属性で入力として受け取る方法です。たとえば、呼び出しページで、結果を返すために使用する変数名として returnHere を渡します。

```
<cf_mytag resultName="returnHere">
```

"mytag.cfm" では、次のコードを使用してその結果を返します。

```
<cfset "Caller.#Attributes.resultName#" = result>
```

カスタムタグで呼び出しページの変数を上書きしないように注意してください。変数の上書きが発生しにくくなるようなネーミング規則を使用してください。たとえば、返す変数の先頭に customtagname_ などの接頭辞を付けます。customtagname はカスタムタグの名前です。

注意：HTTP リクエストや現在のアプリケーションに関連するデータは、カスタムタグページからアクセスできます。このデータには、Form、URL、Cgi、Request、Cookie、Server、Application、Session、および Client スコープの変数が含まれます。

タグ属性の使用方法のまとめ

カスタムタグの属性値は、名前と値のペアとして呼び出しページからカスタムタグページに渡されます。CFML カスタムタグでは、必須の属性とオプションの属性が使用できます。カスタムタグ属性は、次の CFML コーディング規格に準拠します。

- 属性はすべて Attributes スコープに渡されます。
- 渡された属性を参照するには、Attributes.***attribute_name*** というシンタックスを使用して、カスタムタグページのローカル変数と区別します。
- 属性では、大文字と小文字は区別されません。

- 属性は、タグ内で任意の順番でリストできます。
- タグを呼び出すときは、属性名と値のペアをスペースで区切る必要があります。
- 渡す値にスペースが含まれている場合は、二重引用符 (") で囲む必要があります。
- 呼び出しページから渡されたオプションの属性をテストしてデフォルト値を割り当てるには、カスタムタグの冒頭で default 属性を含む cfparam タグを使用します。次に例を示します。

```
<!--- The value of the variable Attributes.Name comes from the calling page. If
    the calling page does not set it, make it "Who". --->
<cfparam name="Attributes.Name" default="Who">
```

- カスタムタグの冒頭で、cfparam タグ、または IsDefined 関数を含む cfif タグを使用して、呼び出しページから渡される必要のある必須属性をテストします。たとえば次のコードは、カスタムタグに Name 属性が渡されなかった場合に処理を中断します。

```
<cfif not IsDefined("Attributes.Name")>
    <cfabort showError="The Name attribute is required.">
</cfif>
```

属性を持つカスタムタグの例

次の例では、渡された属性を使用して呼び出しページの変数 Doctor に値を設定するカスタムタグを作成します。

1 次の内容の ColdFusion ページ (呼び出しページ) を作成します。

```
<html>
<head>
    <title>Enter Name</title>
</head>
<body>
<!--- Enter a name, which could also be done in a form. --->
<!--- This example simply uses a cfset. --->
<cfset NameYouEntered="Smith">

<!--- Display the current name. --->
<cfoutput>
Before you leave this page, you're #Variables.NameYouEntered#.<br>
</cfoutput>

<!--- Go to the custom tag. --->
<cf_getmd Name="#NameYouEntered#">
<!--- Come back from the Custom tag --->

<!--- Display the results of the custom tag. --->
<cfoutput>
You are now #Variables.Doctor#.<br>
</cfoutput>
</body>
</html>
```

2 このページに callingpage.cfm という名前を付けて保存します。

3 次の内容の別のページ (カスタムタグ) を作成します。

```
<!--- The value of the variable Attributes.Name comes from the calling page.
    If the calling page does not set it, make it "Who". --->
<cfparam name="Attributes.Name" default="Who">

<!--- Create a variable called Doctor, make its value "Doctor "
    followed by the value of the variable Attributes.Name.
    Make its scope Caller so it is passed back to the calling page.
--->
<cfset Caller.Doctor="Doctor " & Attributes.Name>
```

4 このページに getmd.cfm という名前を付けて保存します。

5 "callingpage.cfm" ファイルをブラウザで開きます。

呼び出しページで getmd カスタムタグが使用され、結果が表示されます。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfset NameYouEntered="Smith">`	呼び出しページで、変数 NameYouEntered を作成して値 Smith を代入します。
`<cfoutput>` `Before you leave this page, you're` `#Variables.NameYouEntered#. ` `</cfoutput>`	呼び出しページで、カスタムタグを呼び出す前に、NameYouEntered 変数の値を表示します。
`<cf_getmd Name="#NameYouEntered#">`	呼び出しページで、getmd カスタムタグを呼び出します。Name 属性に、ローカル変数 NameYouEntered の値を渡します。
`<cfparam name="Attributes.Name" default="Who">`	カスタムタグページで、呼び出しページから渡された Name 変数を Attributes スコープで取得します。呼び出しページから属性が渡されなかった場合は、値 Who を代入します。
`<cfset Caller.Doctor="Doctor " & Attributes.Name>`	カスタムタグページで、Doctor 変数を作成します。その際には、Caller スコープを使用して、呼び出しページのローカル変数として作成されるようにします。 値は、"Doctor" という文字列に Attributes.Name 変数の値を付加したものにします。
`<cfoutput>` `You are now #Variables.Doctor#. ` `</cfoutput>`	呼び出しページで、カスタムタグページから返された Doctor 変数の値を表示します。この例では、変数がローカル変数として返されたという事実を強調するために、Variables スコープの接頭辞を使用しています。

CFML 構造体によるカスタムタグ属性の受け渡し

予約属性 attributeCollection を使用すれば、構造体を使用してカスタムタグに属性を渡すことができます。attributeCollection 属性に渡す構造体では、キーとして属性名を指定し、値として属性値を指定します。カスタムタグを呼び出すときは、attributeCollection と他の属性を組み合わせることができます。

attributeCollection 属性で指定された構造体のキーと値のペアは、カスタムタグページの Attributes スコープにコピーされます。この動作は、カスタムタグを呼び出すときに attributeCollection の各エントリを個別の属性として指定した場合と同じです。attributeCollection を使用して渡された属性をカスタムタグページで参照するには、Attributes.CustomerName、Attributes.Department_number などのように、通常の属性と同じ方法を使用します。

注意：カスタムタグを直接使用する場合や、cfmodule タグを使用してカスタムタグを呼び出す場合は、タグ属性と attributeCollection 属性を組み合わせて使用できます。タグ属性を使用する方法と attributeCollection 構造体を使用する方法の両方で同じ名前の属性を渡した場合、カスタムタグにはタグ属性のみが渡され、属性コレクションの対応する属性は無視されます。標準の (ビルトイン) ColdFusion タグを使用する場合は、タグ属性と attributeCollection 属性を組み合わせて使用することはできません。

attributeCollection 属性は、カスタムタグ属性のコレクションを保持する構造体用に予約されています。attributeCollection がそのようなコレクションを参照していない場合は、テンプレート例外が発生します。

次の例では、attributeCollection 属性を使用して、4 つの属性のうちの 2 つを渡します。

```
<cfset zort=StructNew()>
<cfset zort.x = "-X-">
<cfset zort.y = "-Y-">
<cf_testtwo a="blab" attributeCollection=#zort# foo="16">
```

"testtwo.cfm" に次のコードが含まれている場合は、

```
---custom tag ---<br>
<cfoutput>#attributes.a# #attributes.x# #attributes.y#
    #attributes.foo#</cfoutput><br>
--- end custom tag ---
```

次のステートメントが出力されます。

```
---custom tag ---
blab -X- -Y- 16
--- end custom tag ---
```

attributeCollection を使用すれば、特定のカスタムタグの Attributes スコープ全体を別のカスタムタグに渡すことができます。最初のカスタムタグが 2 番目のカスタムタグを呼び出しており、最初のタグのすべての属性を 2 番目のタグに渡す必要がある場合などに、この方法を使用すると便利です。

たとえば、次のコードを使用してカスタムタグを呼び出します。

```
<cf_first attr1="foo" attr2="bar">
```

最初のカスタムタグのすべての属性を 2 番目のカスタムタグに渡すには、"first.cfm" に次のステートメントを含めます。

```
<cf_second attributeCollection="#attributes#">
```

"second.cfm" の本文では、渡されたパラメータを次のように参照します。

```
<cfoutput>#attributes.attr1#</cfoutput>
<cfoutput>#attributes.attr2#</cfoutput>
```

カスタムタグの管理

複数の開発者がいる環境でカスタムタグをデプロイする場合や、タグを一般に配布する場合は、ColdFusion の高度なセキュリティ機能やテンプレートのエンコード機能を利用できます。

ColdFusion のセキュリティフレームワークでは、タグファイルやタグディレクトリに対して個別にアクセス制限を設定できます。この制限は、チーム開発で役立つ重要な保護機能です。詳細については、337 ページの「アプリケーションの保護」を参照してください。

コマンドラインユーティリティ cfcompile を使用して、カスタムタグファイルを Java クラスファイルやバイトコードにプリコンパイルできます。詳細については、『ColdFusion 設定と管理』の Using the cfcompile utility を参照してください。

カスタムタグの実行

ここでは、終了タグや本文テキストの処理などの、カスタムタグの実行方法について説明します。

タグインスタンスデータへのアクセス

カスタムタグページの実行時には、タグインスタンスに関するデータが thisTag 構造体に保持されます。カスタムタグから thisTag 構造体にアクセスして、タグの処理を制御できます。これは、タグ固有の File 変数 (File スコープとも呼ばれます) に似ています。

次の表に示す変数が ColdFusion によって生成され、thisTag 構造体に保持されます。

変数	説明
ExecutionMode	カスタムタグの実行モードを保持します。有効な値は、"start"、"end"、および "inactive" です。
HasEndTag	終了タグを使用してカスタムタグが呼び出されているかどうかを識別します。コードの検証に使用します。終了タグが使用されている場合、HasEndTag は true に設定されます。それ以外の場合は false に設定されます。
GeneratedContent	タグによって生成された内容を表します。これには、ColdFusion の変数や関数などのアクティブな内容の結果を含む、タグ本文の内容がすべて含まれます。この内容は変数として処理できます。
AssocAttribs	cfassociate を使用して、ネストタグの属性を親タグで利用できるようにしている場合に、すべてのネストタグの属性が保持されます。詳細については、218 ページの「高レベルのデータ交換」を参照してください。

次の例では、カスタムタグから thisTag 構造体の ExecutionMode 変数にアクセスします。

```
<cfif thisTag.ExecutionMode is 'start'>
```

終了タグの処理

これまでのカスタムタグの例では、次のように、開始タグのみを使用してカスタムタグを参照していました。

```
<cf_date>
```

この場合は、ColdFusion によってカスタムタグページ "date.cfm" が呼び出されてタグが処理されます。

これに対して、開始タグと終了タグの両方を持つカスタムタグを作成することもできます。たとえば、次のタグには開始タグと終了タグがあります。

```
<cf_date>
    ...
</cf_date>
```

終了タグを持つタグでは、カスタムタグページ "date.cfm" が 2 回呼び出されます。1 回目は開始タグ、2 回目は終了タグに対して呼び出されます。"date.cfm" ページでは、現在の呼び出しが開始タグに対するものか、終了タグに対するものかを判断して、適切な処理を実行できます。

次のような終了タグの簡易表記を使用した場合も、カスタムタグページが 2 回呼び出されます。

```
<cf_date/>
```

カスタムタグは、cfmodule タグを使用して、次の例のように呼び出すこともできます。

```
<cfmodule ...>
    ...
</cfmodule>
```

cfmessagebox で終了タグを指定した場合は、開始タグと終了タグの両方があるカスタムタグと同じように、カスタムタグが呼び出されます。

終了タグが使用されているかどうかの確認

終了タグを必要とするカスタムタグを作成できます。終了タグが必要である場合は、カスタムタグページで thisTag.HasEndTag を使用して、ユーザーが終了タグを使用したかどうかを確認できます。

たとえば、"date.cfm" に次のコードを含めて、終了タグが使用されているかどうかを調べます。

```
<cfif thisTag.HasEndTag is 'False'>
    <!--- Abort the tag--->
    <cfabort showError="An end tag is required.">
</cfif>
```

タグの実行モードの確認

変数 thisTag.ExecutionMode には、カスタムタグページの呼び出しモードが含まれています。変数の値は、次のいずれかです。

Start 終了タグを処理するモード

End 終了タグを処理するモード

Inactive カスタムタグがネストタグを使用しているモード詳細については、216 ページの「カスタムタグのネスト」を参照してください。

終了タグが明示的に使用されていない場合は、ColdFusion によってカスタムタグページが Start モードで 1 回だけ呼び出されます。

テキストを太字にする "bold.cfm" というカスタムタグページは、次のように記述できます。

```
<cfif thisTag.ExecutionMode is 'start'>
    <!--- Start tag processing --->
    <B>
<cfelse>
    <!--- End tag processing --->
    </B>
</cfif>
```

このタグを使用してテキストを太字に変換できます。

```
<cf_bold>This is bold text</cf_bold>
```

cfswitch を使用してカスタムタグの実行モードを判断することもできます。

```
<cfswitch expression=#thisTag.ExecutionMode#>
    <cfcase value= 'start'>
        <!--- Start tag processing --->
    </cfcase>
    <cfcase value='end'>
        <!--- End tag processing --->
    </cfcase>
</cfswitch>
```

終了タグ使用時の注意事項

カスタムタグを作成する際に、開始タグと終了タグにどのように処理を配分するかは、タグの機能に大きく依存しますが、次のルールを参考にすると判断しやすくなります。

- 入力された属性の検証、デフォルト値の設定、必須の終了タグの存在確認などは、開始タグで行います。
- 開始タグと終了タグの間でタグに渡された本文テキストなどの実際のタグ処理は、終了タグで行います。本文テキストの詳細については、215 ページの「本文テキストの処理」を参照してください。
- 出力の実行は、開始タグか終了タグのいずれかで行います。2 つのタグに処理を配分しないでください。

本文テキストの処理

本文テキストとは、カスタムタグを呼び出すときに開始タグと終了タグの間に含めるテキストです。次に例を示します。

```
<cf_happybirthdayMessge name="Ellen Smith" birthDate="June, 8, 1993">
    <p> Happy Birthday Ellen!</p>
    <p> May you have many more!</p>
</cf_happybirthdayMessge>
```

この例では、開始タグの次にある 2 行のコードが本文テキストです。

thisTag.GeneratedContent 変数を使用すれば、カスタムタグの本文テキストにアクセスできます。この変数には、タグに渡されたすべての本文テキストが含まれています。このテキストはタグの処理時に変更できます。thisTag.GeneratedContent 変数の内容は、タグの出力の一部としてブラウザに返されます。

開始タグの処理中は、thisTag.GeneratedContent 変数は常に空です。開始タグの処理中に生成された出力は、カスタムタグの生成された内容の一部とは見なされません。

thisTag.GeneratedContent 変数を使用すると、カスタムタグの生成された内容にアクセスして変更できます。生成された内容とは、カスタムタグの本文を処理した結果を意味します。この内容には、本文のすべてのテキストおよび HTML コード、ColdFusion の変数、式、および関数の評価結果、さらに子孫タグによって生成された結果が含まれます。この変数の値を変更すると、生成される内容も変わります。

子孫タグによって生成された HTML をコメントアウトするタグの例を示します。このタグの実装は、次のようになります。

```
<cfif thisTag.ExecutionMode is 'end'>
    <cfset thisTag.GeneratedContent ='<!--#thisTag.GeneratedContent#-->'>
</cfif>
```

タグ実行の終了

カスタムタグ内では通常、エラーをチェックしパラメータを確認します。このチェックで、必須属性が未設定であるなどの重大なエラーを検出した場合は、cfabort を使用してタグを中断できます。

cfexit タグでもカスタムタグの実行を終了できます。カスタムタグを作成するときは、cfabort タグよりも cfexit タグのほうが柔軟な処理が可能です。cfexit タグの method 属性で、実行を再開する場所を指定します。たとえば、cfexit タグの最初の子から処理を再開したり、終了タグマーカーの直後から再開したりできます。

また、method 属性を使用して、タグ本文を再実行することもできます。これによって、cfloop の動作をエミュレートして、高レベルの繰り返しを実行できます。

次の表に、cfexit の動作を示します。

Method 属性の値	**cfexit 呼び出しの場所**	**動作**
ExitTag (デフォルト)	ベースページ	cfabort のように動作します。
	ExecutionMode=start	終了タグの後から処理を継続します。
	ExecutionMode=end	終了タグの後から処理を継続します。
ExitTemplate	ベースページ	cfabort のように動作します。
	ExecutionMode=start	本文内の最初のサブタグから処理を継続します。
	ExecutionMode=end	終了タグの後から処理を継続します。
Loop	ベースページ	エラー
	ExecutionMode=start	エラー
	ExecutionMode=end	本文内の最初のサブタグから処理を継続します。

カスタムタグのネスト

カスタムタグの本文テキストから他のカスタムタグを呼び出すことを、タグのネストと呼びます。ColdFusion には、cfgraph と cfgraphdata、cfhttp と cfhttpparam、cftree と cftreeitem などのネストタグが用意されています。タグをネストすることで、関連する機能を提供できます。

次の例では、cftree タグの中に cftreeitem タグがネストされています。

```
<cftree name="tree1"
    required="Yes"
    hscroll="No">
    <cftreeitem value=fullname
        query="engquery"
        queryasroot="Yes"
        img="folder,document">
</cftree>
```

呼び出し側のタグは、祖先タグ、親タグ、またはベースタグと呼ばれます。祖先タグに呼び出されるタグは、子孫タグ、子タグ、またはサブタグと呼ばれます。祖先タグとすべての子孫タグは、協調しているタグと呼ばれます。

タグをネストするには、親タグに終了タグが必要です。

次の表に、ネストタグ間の関係を表す用語を示します。

呼び出しタグ	呼び出しタグ内にネストされているタグ	説明
祖先	子孫	祖先タグは、開始タグと終了タグの間に他のタグを含んでいるタグです。子孫タグは、タグによって呼び出されているタグです。
親	子	親と子の関係は、祖先と子孫の関係と同じです。
ベースタグ	サブタグ	ベースタグは、子孫 (サブタグと呼ばれます) で cfassociate によって明示的に関連付けられている祖先です。

複数のレベルのネストタグを作成できます。この場合、サブタグは、その中のサブタグに対するベースタグになります。終了タグがあるタグは、別のタグの祖先になることができます。

ネストカスタムタグには 3 つの処理モードがあります。ベースタグでは、変数 thisTag.ExecutionMode を使用して処理モードを確認できます。

ネストしているカスタムタグ間でのデータ交換

カスタムタグの重要な特徴の 1 つは、協調しているカスタムタグの間でユーザーの介入なしに複雑なデータを交換し、各タグの実装をカプセル化して隠蔽できる点にあります。

ネストタグを使用する場合は、次の点について検討する必要があります。

- どのデータを公開するか。
- どのタグの間でやり取りを行うか。
- データ交換のソースとターゲットをどのように識別するか。
- データ交換にどのような CFML メカニズムを使用するか。

公開するデータ

制限の少ない環境では、開発者の生産性を最大にするために、CFML カスタムタグから協調しているタグにすべてのデータを公開できます。

カスタムタグの開発時には、協調しているタグからアクセスや変更が可能な変数をすべて文書化します。他のカスタムタグと協調するカスタムタグを作成する場合は、

文書化されていないデータを変更しないように十分注意する必要があります。

カプセル化を維持するには、タグデータへのアクセス操作や変更操作をすべてカスタムタグで行うようにします。たとえば、「このタグの実装では変数 MyQueryResults にクエリー結果を保持しています」と文書化してユーザーにその変数を直接操作させるのではなく、MyQueryResult を操作するためのネストカスタムタグを用意します。これによって、ユーザーがカスタムタグの実装で変更を行うのを防ぐことができます。

Variable スコープと特殊変数

ネストタグの変数には Request スコープを使用します。Request スコープは、ベースページ、ベースページがインクルードするすべてのページ、ベースページが呼び出すすべてのカスタムタグページ、およびインクルードページやカスタムタグページが呼び出すすべてのカスタムタグページで利用できます。1 つの同じタグにネストしていない、互いに協調しているカスタムタグの間では、リクエスト構造体を使用してデータを交換できます。Request スコープは Request という構造体として表されます。

アクセス可能なデータの場所

ページ内の 2 つのカスタムタグはさまざまな方法で関連付けることができます。祖先と子孫の関係は、タグのネストの順番に影響するので重要です。

タグページが実行されている間、そのタグの子孫はアクティブではありません。つまり、子孫タグにはインスタンスデータがありません。したがって、タグがアクセスできるのは祖先のデータのみです。子孫のデータにはアクセスできません。祖先のデータは、現在のページおよび実行時タグコンテキストスタック全体から利用可能です。タグコンテキストスタックとは、現在のタグ要素からネストタグ階層を上にたどって、リクエストのベースページの先頭に達するまでのパスのことです。これには、インクルードページ内のタグやカスタムタグ参照も含まれます。cfinclude タグもカスタムタグも、タグコンテキストスタックに含まれます。

高レベルのデータ交換

ネストカスタムタグを作成すると生産性が大幅に向上しますが、複雑なネストタグ階層を追跡するのは手間がかかります。cfassociate タグを使用すれば、子タグの状態を親タグに伝えることができます。サブタグでこのタグを使用すると、サブタグの属性をベースタグに提供できます。

cfhttp/cfhttpparam や cftree/cftreeitem のように、データ検証を行って祖先タグにデータを渡すためにのみ子孫タグを使用する場合があります。cfassociate タグを使用すれば、この処理をカプセル化することができます。

cfassociate タグは、次のように記述します。

```
<cfassociate baseTag="tagName" dataCollection="collectionName">
```

baseTag 属性には、このタグの属性を提供するベースタグの名前を指定します。dataCollection 属性には、サブタグのデータをベースタグに提供するために使用する構造体の名前を指定します。デフォルト値は AssocAttribs です。dataCollection 属性を指定する必要があるのは、ベースタグが複数のタイプのサブタグを持つ可能性がある場合のみです。これは、タグのタイプごとに属性のコレクションを維持する場合に役立ちます。

注意：終了タグが必要なカスタムタグの場合、dataCollection 属性によって参照される構造体を処理するコードは、終了タグのコードに含める必要があります。

サブタグ内で cfassociate が検出されると、サブタグの属性が自動的にベースタグに保存されます。それらの属性は構造体として、thisTag.**collectionName** という配列の末尾に追加されます。

cfassociate タグでは、次の処理が行われます。

```
<!--- Get base tag instance data --->
    <cfset data = getBaseTagData(baseTag)>
    <!--- Create a string with the attribute collection name --->
    <cfset collection_Name = "data.#dataCollection#">
    <!--- Create the attribute collection, if necessary --->
    <cfif not isDefined(collectionName)>
        <cfset #collection_Name# = arrayNew(1)>
    </cfif>
    <!--- Append the current attributes to the array --->
    <cfset temp=arrayAppend(evaluate(collectionName), attributes)>
```

ベースタグ内のサブタグ属性にアクセスするコードは、次のようになります。

```
<!--- Protect against no subtags --->
    <cfparam Name='thisTag.assocAttribs' default=#arrayNew(1)#>

    <!--- Loop over the attribute sets of all sub tags --->
    <cfloop index=i from=1 to=#arrayLen(thisTag.assocAttribs)#>

        <!--- Get the attributes structure --->
        <cfset subAttribs = thisTag.assocAttribs[i]>
        <!--- Perform other operations --->

    </cfloop>
```

祖先データへのアクセス

祖先のデータはすべて構造体オブジェクトに含まれています。

次の関数を使用すると、祖先データにアクセスできます。

- GetBaseTagList : 大文字の祖先タグ名をカンマで区切ったリストを文字列として返します。最初のリスト要素は現在のタグです。現在のタグがネストタグである場合、2 番目の要素は親タグ名になります。トップレベルのタグに対してこの関数を呼び出すと、空の文字列が返されます。
- GetBaseTagData(InstanceNumber =1) : 指定した名前を持つ n 番目の祖先の (ローカル変数だけでなく) すべての変数を含んでいるオブジェクトを返します。デフォルトでは、最も近い祖先が返されます。指定した名前の祖先がない場合や、祖先でデータが公開されていない場合 (cfif など) は、例外が発生します。

例 : 祖先データへのアクセス

この例では、2 つのカスタムタグと、各カスタムタグを呼び出す簡単なページを作成します。最初のカスタムタグによって、2 番目のタグを呼び出します。2 番目のタグによって、タグのステータスをレポートし、祖先についての情報を提供します。

呼び出しページの作成

1 次の内容の ColdFusion ページ (呼び出しページ) を作成します。

```
Call cf_nesttag1 which calls cf_nesttag2<br>
<cf_nesttag1>
<hr>

Call cf_nesttag2 directly<br>
<cf_nesttag2>
<hr>
```

2 このページに "nesttest.cfm" という名前を付けて保存します。

最初のカスタムタグページの作成

1 次の内容の ColdFusion ページを作成します。

```
<cf_nesttag2>
```

2 このページに "nesttag1.cfm" という名前を付けて保存します。

2 番目のカスタムタグページの作成

1 次の内容の ColdFusion ページを作成します。

```
<cfif thisTag.executionmode is 'start'>
    <!--- Get the tag context stack. The list looks something like
    "MYTAGNAME, CALLINGTAGNAME, ..." --->
    <cfset ancestorlist = getbasetaglist()>

    <!--- Output your own name. You are the first entry in the context stack. --->
    <cfoutput>
    <p>I'm custom tag #ListGetAt(ancestorlist,1)#</p>

    <!--- Output all the contents of the stack a line at a time. --->
    <cfloop index="loopcount" from="1" to="#listlen(ancestorlist)#">
        Ancestorlist entry #loopcount# n is #ListGetAt(ancestorlist,loopcount)#<br>
    </cfloop><br>
    </cfoutput>

    <!--- Determine whether you are nested inside a custom tag. Skip the first
        element of the ancestor list, i.e., the name of the custom tag I'm in. --->
    <cfset incustomtag = ''>
    <cfloop index="elem"
        list="#listrest(ancestorlist)#">
        <cfif (left(elem, 3) eq 'cf_')>
            <cfset incustomtag = elem>
            <cfbreak>
        </cfif>
    </cfloop>

    <cfif incustomtag neq ''>
        <!--- Say that you are there. --->
        <cfoutput>
            I'm running in the context of a custom tag named #inCustomTag#.<p>
        </cfoutput>

        <!--- Get the tag instance data. --->
        <cfset tagdata = getbasetagdata(incustomtag)>

        <!--- Find out the tag's execution mode. --->
        I'm located inside the
        <cfif tagdata.thisTag.executionmode neq 'inactive'>
            custom tag code either because it is in its start or end execution mode.
        <cfelse>
            body of the tag
        </cfif>
        <p>
    <cfelse>
        <!--- Say that you are lonely. --->
        I'm not nested inside any custom tags. :^( <p>
    </cfif>
</cfif>
```

2 このページに "nesttag2.cfm" という名前を付けて保存します。

3 "nesttest.cfm" ファイルをブラウザで開きます。

CFXAPI カスタムタグの構築

場合によっては、ColdFusion で実行する実行可能ファイルとしてアプリケーション要素を開発することが、最適なアプリケーション開発方法であることもあります。たとえば、アプリケーションに必要な機能が、現在利用できる CFML の機能では実現できない場合があります。また、アプリケーションの特定の処理のパフォーマンスを改善したい場合もあります。あるいは、アプリケーションの問題点を解決できるコードが既に作成されており、それを ColdFusion アプリケーションに導入したい場合も考えられます。

CFX API (ColdFusion Extension Application Programming Interface) を使用して Java または C++ に基づくカスタム ColdFusion タグを開発すれば、これらの要件を満たすことができます。

CFX タグとは

CFX (ColdFusion Extension) タグは、ColdFusion Extension Application Programming Interface を使用して記述するカスタムタグです。CFML では実現できない機能を開発する場合や、頻繁に実行するタスクのパフォーマンスを改善したい場合によく利用されます。

また、既存のアプリケーション機能を ColdFusion アプリケーションに組み込む場合にも CFX タグが利用できます。つまり、使用可能なコードが既にある場合は、CFX タグを利用することで、そのコードを簡単にアプリケーションで使用できます。

CFX タグでは、次のことが行えます。

- 任意の数のカスタム属性を処理することができます。
- カスタムの形式で ColdFusion クエリーを使用して操作できます。
- ODBC 以外の規格を使用した情報ソースとやり取りを行う ColdFusion クエリーを生成できます。
- クライアントに返す HTML をダイナミックに生成できます。
- タグを呼び出した ColdFusion アプリケーションページ内の変数を設定できます。
- 標準的な ColdFusion エラーメッセージとして処理される例外を返すことができます。

CFX タグは、C++ または Java を使用して構築できます。

注意：ColdFusion には、カスタムタグ以外にも、再利用可能なコードを作成する方法がいくつか用意されています。これらの方法の詳細については、143 ページの「ColdFusion 要素の作成」を参照してください。

Java を使用して CFX タグの開発を始める前に

Java を使用して CFX タグの開発を始める前に、Java 開発環境を設定します。また、CFX タグを作成する前に、ここに記載されている例を見ておくと参考になります。

Java のサンプル CFX タグ

Java で CFX タグを開発する前に、サンプルの CFX タグを見ておくと参考になる場合があります。サンプルの Java ソースファイルは、Windows の場合は、メインインストールディレクトリの "cfx¥java¥distrib¥examples" サブディレクトリにあります。UNIX システムの場合は、"cfx/java/examples" ディレクトリにあります。次の表にサンプルタグを示します。

例	アクション	説明
HelloColdFusion	個人宛の挨拶を表示します。	CFX タグを作成するのに必要な最小限の実装です。
ZipBrowser	ZIP アーカイブのコンテンツを取得します。	ColdFusion クエリーを生成し、このクエリーを呼び出しページに返します。
ServerDateTime	ネットワークサーバーから日付と時刻を取得します。	数値属性を使用して属性を検証し、呼び出しページ内の変数を設定します。
OutputQuery	HTML テーブルに ColdFusion クエリーを返します。	入力として ColdFusion クエリーを処理し、例外を返し、ダイナミック出力を生成します。
HelloWorldGraphic	JPEG 形式の "Hello World!" グラフィックを生成します。	Java で作成した CFX タグからグラフィックをダイナミックに生成して返します。

Java で CFX タグを開発するための開発環境の設定

Java CFX タグの構築には、Sun の Java 開発キット (JDK) などのさまざまな Java 開発環境を使用できます。JDK は、Sun の Web サイト (http://java.sun.com/j2se) からダウンロードできます。

開発、デバッグ、およびプロジェクト管理のための統合環境を提供する、市販の Java IDE を使用することをお勧めします。

クラスパスの設定

Java CFX タグを構築するための開発環境を設定するには、必要なサポートクラスが Java コンパイラから認識されるようにする必要があります。これらのクラスは、"cfx.jar" アーカイブで提供されています。このアーカイブは、次のいずれかのディレクトリにあります。

サーバー設定 <ColdFusion のルートディレクトリ >/wwwroot/WEB-INF/lib

J2EE 設定 <ColdFusion Web アプリケーションのルートディレクトリ >/WEB-INF/lib

実際の環境でコンパイラクラスパスを設定する方法については、Java 開発ツールのマニュアルを参照してください。

"cfx.jar" アーカイブには、com.allaire.cfx パッケージにクラスが格納されています。これらのクラスは、Java CFX タグを開発およびデプロイする場合に必要となります。

新しい Java CFX タグを作成する場合は、"WEB-INF/classes" ディレクトリにコンパイルします。これによって、開発、デバッグ、およびテストプロセスが簡略化されます。

開発とテストが終了したら、ColdFusion で認識できるクラスパス上の任意の場所に Java CFX タグをデプロイできます。

Java のカスタマイズと設定

ColdFusion Administrator の [サーバーの設定]-[Java と JVM] ページを使用すると、Java 開発環境をカスタマイズできます。たとえば、クラスパスや Java システムプロパティをカスタマイズしたり、代替 JVM (Java Virtual Machine: Java 仮想コンピュータ) を指定したりできます。詳細については、ColdFusion Administrator のオンラインヘルプを参照してください。

Java CFX タグの作成

Java CFX タグを作成するには、カスタムタグインターフェイスを実装するクラスを作成します。このインターフェイスには processRequest というメソッドが 1 つ含まれています。このメソッドには、タグで処理する Request および Response オブジェクトが渡されます。

次の手順の例では、呼び出しページにテキスト文字列を書き込む cfx_MyHelloColdFusion という単純な Java CFX タグを作成します。

1 次のコードを使用して、エディタでソースファイルを作成します。

```
import com.allaire.cfx.* ;

public class MyHelloColdFusion implements CustomTag {
    public void processRequest( Request request, Response response )
    throws Exception {
    String strName = request.getAttribute( "NAME" ) ;
    response.write( "Hello, " + strName ) ;
    }
}
```

2 このファイルに "MyHelloColdFusion.java" という名前を付けて、"WEB_INF/classes" ディレクトリに保存します。

3 Java コンパイラを使用して、Java ソースファイルをクラスファイルにコンパイルします。JDK に同梱されているコマンドツールを使用する場合は、"classes" ディレクトリで次のコマンドラインを実行します。

```
javac -classpath cf_root\WEB-INF\lib\cfx.jar MyHelloColdFusion.java
```

注意：このコマンドは、Java コンパイラ (javac.exe) がパスに含まれている場合にのみ動作します。パスに含まれていない場合は、完全修飾パスを指定してください。たとえば、Windows の場合は c:¥jdk1.3.1_01¥bin¥javac、UNIX の場合は /usr/java/bin/javac などのように指定します。

コンパイル時にエラーが発生した場合は、ソースコードが正しく入力されているかどうかを確認してください。エラーが発生しなければ、初めての Java CFX タグが正しく作成できたことになります。

ColdFusion ページからの CFX タグの呼び出し

ColdFusion ページから Java CFX タグを呼び出すには、ColdFusion Administrator の [CFX タグ] ページで登録した CFX タグの名前を使用します。この名前は、接頭辞 cfx_ の後ろにクラス名 (.class 拡張子は除く) を付加したものにします。

ColdFusion Administrator での Java CFX タグの登録

1 ColdFusion Administrator で、[拡張機能]-[CFX タグ] を選択します。

2 [Java CFX の登録] をクリックします。

3 タグ名 (たとえば、cfx_MyHelloColdFusion) を入力します。

4 拡張子 .class を除いたクラス名 (たとえば、MyHelloColdFusion) を入力します。

5 (オプション) 説明を入力します。

6 送信ボタンをクリックします。

これで、ColdFusion ページからタグを呼び出すことができます。

ColdFusion ページからの CFX タグの呼び出し

1 HelloColdFusion カスタムタグを呼び出す次の ColdFusion ページ (.cfm) をエディタで作成します。

```
<html>
<body>
    <cfx_MyHelloColdFusion NAME="Les">
</body>
</html>
```

2 ColdFusion ページを配置するためのディレクトリにファイルを保存します。たとえば、Windows の場合は "c:¥inetpub¥wwwroot¥cfdocs¥testjavacfx.cfm" に保存し、UNIX の場合は "/home/docroot/cfdocs/testjavacfx.cfm" に保存します。

3 CFX タグをまだ登録していない場合は、ColdFusion Administrator で登録します (230 ページの「CFX タグの登録」を参照)。

4 次のような適切な URL を使用してブラウザからページをリクエストします。

http://localhost/cfdocs/testjavacfx.cfm

ColdFusion によってページが処理され、"Hello, Les." と表示するページが返されます。エラーが返される場合は、ソースコードが正しく入力されているかどうかを確認してください。

ColdFusion Administrator での CFX タグの削除

1 ColdFusion Administrator で、[拡張機能]-[CFX タグ] を選択します。

2 [登録された CFX タグ] リストで、削除するタグの [コントロール] 列の [削除] アイコンをクリックします。

リクエストの処理

Java CFX タグを実装するには、processRequest メソッドに渡される Request および Response オブジェクトを操作する必要があります。また、ColdFusion クエリーを処理する必要のある CFX タグでは、Query オブジェクトもインターフェイスする必要があります。Request オブジェクト、Response オブジェクト、および Query オブジェクトは、"WEB-INF/lib/cfx.jar" アーカイブ内の com.allaire.cfx パッケージに含まれています。

これらのオブジェクトの詳細については、『CFML リファレンス』の ColdFusion Java CFX Reference を参照してください。Request オブジェクト、Response オブジェクト、および Query オブジェクトを使用する Java CFX タグの完全な例については、225 ページの「ZipBrowser の例」を参照してください。

Request オブジェクト

Request オブジェクトは、CustomTag インターフェイスの processRequest メソッドに渡されます。次の表に、タグに渡された属性 (クエリーなど) の取得や、グローバルタグ設定の取得に使用する Request オブジェクトのメソッドを示します。

メソッド	説明
attributeExists	このタグに属性が渡されたかどうかをチェックします。
debug	タグに debug 属性があるかどうかをチェックします。
getAttribute	渡された属性の値を取得します。
getAttributeList	タグに渡されたすべての属性名のリストを取得します。
getIntAttribute	渡された属性の整数値を取得します。
getQuery	このタグにクエリーが渡された場合は、そのクエリーを取得します。
getSetting	グローバルカスタムタグ設定の値を取得します。

これらのインターフェイスに関する詳細なリファレンス情報については、『CFML リファレンス』を参照してください。

Response オブジェクト

Response オブジェクトは、CustomTag インターフェイスの processRequest メソッドに渡されます。次の表に、出力の書き込み、クエリーの生成、および呼び出しページ内の変数の設定に使用する Response オブジェクトのメソッドを示します。

メソッド	説明
write	呼び出しページにテキストを出力します。
setVariable	呼び出しページの変数を設定します。
addQuery	呼び出しページにクエリーを追加します。
writeDebug	デバッグストリームにテキストを出力します。

これらのインターフェイスに関する詳細なリファレンス情報については、『CFML リファレンス』を参照してください。

Query オブジェクト

Query オブジェクトは、ColdFusion クエリーを操作するためのインターフェイスを提供します。次の表に、名前、行数、および列名の取得や、データ要素の取得および設定を行う Query オブジェクトのメソッドを示します。

メソッド	説明
getName	クエリーの名前を取得します。
getRowCount	クエリーの行数を取得します。
getColumnIndex	クエリー列のインデックスを取得します。
getColumns	クエリー列の名前を取得します。
getData	クエリーからデータ要素を取得します。
addRow	クエリーに新しい行を追加します。
setData	クエリー内にデータ要素を設定します。

これらのインターフェイスに関する詳細なリファレンス情報については、『CFML リファレンス』を参照してください。

Java CFX タグのライフサイクル

Java CFX オブジェクトのインスタンスは、Java CFX タグが呼び出されるたびに新しく作成されます。したがって、CustomTag オブジェクトのメンバーにそのリクエスト固有のインスタンスデータを保存しても問題はありません。CustomTag のすべてのインスタンスからアクセス可能なデータやオブジェクトを保存するには、スタティックデータメンバーを使用します。その場合は、データへのすべてのアクセスがスレッドセーフであることを確認する必要があります。

ZipBrowser の例

次に、Request オブジェクト、Response オブジェクト、および Query オブジェクトの使用例を示します。この例では、java.util.zip パッケージを使用して、cfx_ZipBrowser という Java CFX タグを実装します。この Java CFX タグは、ZIP ファイルをブラウズするタグです。

注意：cfx_ZipBrowser が実装されている Java ソースファイル "ZipBrowser.java" は、"<ColdFusion のルートディレクトリ >/cfx/java/distrib/examples" (サーバー設定) または "<ColdFusion Web アプリケーションのルートディレクトリ >/WEB-INF/cfusion/cfx/java/distrib/examples" (J2EE 設定) ディレクトリにあります。このタグを実装するには、"ZipBrowser.java" をコンパイルします。

このタグの archive 属性では、ブラウズする ZIP アーカイブの完全修飾パスを指定します。また、このタグの name 属性では、呼び出しページに返すクエリーを指定します。返されるクエリーには、Name、Size、および Compressed という 3 つの列が含まれます。

たとえば、パス c:¥logfiles.zip にあるアーカイブの内容をクエリーして結果を出力するには、次の CFML コードを使用します。

```
<cfx_ZipBrowser
    archive="C:\logfiles.zip"
    name="LogFiles">

<cfoutput query="LogFiles">
#Name#,#Size#, #Compressed# <BR>
</cfoutput>
```

ZipBrowser の Java の実装は次のとおりです。

```
import com.allaire.cfx.* ;
import java.util.Hashtable ;
import java.io.FileInputStream ;
import java.util.zip.* ;

public class ZipBrowser implements CustomTag {
    public void processRequest( Request request, Response response )
        throws Exception {
        // Validate that required attributes were passed.
        if (!request.attributeExists( "ARCHIVE" ) || !request.attributeExists( "NAME" ) ) {
            throw new Exception(
        "Missing attribute (ARCHIVE and NAME are both " +
            "required attributes for this tag)" ) ;
        }
     // get attribute values
        String strArchive = request.getAttribute( "ARCHIVE" ) ;
        String strName = request.getAttribute( "NAME" ) ;

    // create a query to use for returning the list of files
        String[] columns = { "Name", "Size", "Compressed" } ;
        int iName = 1, iSize = 2, iCompressed = 3 ;
        Query files = response.addQuery( strName, columns ) ;

    // read the ZIP file and build a query from its contents
        ZipInputStream zin = new ZipInputStream( new FileInputStream(strArchive) ) ;
        ZipEntry entry ;
        while ( ( entry = zin.getNextEntry()) != null ) {
            // Add a row to the results.
                int iRow = files.addRow() ;

                // populate the row with data
                files.setData( iRow, iName, entry.getName() ) ;
                files.setData( iRow, iSize, String.valueOf(entry.getSize()) ) ;
                files.setData( iRow, iCompressed,
                    String.valueOf(entry.getCompressedSize())) ;

                // Finish up with entry.
                zin.closeEntry() ;
            }

            // Close the archive.
            zin.close() ;
    }
}
```

Java CFX タグをデバッグする方法

Java CFX タグは、通常の Java アプリケーションのような、固有のプロセスとして実行される独立したアプリケーションではありません。Java CFX タグは、既存のプロセスによって作成され、実行されます。したがって、他のプロセスによってロードされた Java クラスを対話型デバッガでデバッグする手法が使用できないので、Java CFX タグのデバッグは難しくなります。

次のいずれかの方法を使用すると、この制約を回避できます。

- 必要に応じてデバッグ情報を出力することによって、ColdFusion で実行しながら CFX タグをデバッグする。
- ブレークポイントの設定、コードのステップ実行、変数値の表示などのデバッグ機能をサポートする Java IDE (Integrated Development Environment) を使用して CFX タグをデバッグする。
- 特殊な com.allaire.cfx デバッグクラスを使用して、ColdFusion から対話型デバッガをオフラインで実行してリクエストをデバッグする。

デバッグ情報の出力

対話型デバッガが普及する前は、プログラムに出力ステートメントを挿入して、変数値や制御パスなどの情報を表示することでプログラムをデバッグするのが一般的でした。新しいプラットフォームが登場したときは、そのプラットフォーム向けの高度なデバッグテクノロジーが提供されるまで、プログラマはこの従来の方法を使用することになります。

本番サーバーで実行しながら Java CFX タグをデバッグする場合は、この従来の方法を使用する必要があります。Response.write メソッドを使用してデバッグテキストを出力する方法の他に、debug="On" 属性を使用して Java CFX タグを呼び出す方法も使用できます。この属性によって、リクエストがデバッグモードで実行されていることを表すフラグが CFX タグに設定されます。したがって CFX タグでは、このフラグを検出して詳細なデバッグ情報を生成できます。たとえば、デバッグモードで HelloColdFusion CFX タグを呼び出すには、次の CFML コードを使用します。

```
<cfx_HelloColdFusion name="Robert" debug="On">
```

CFX タグが debug 属性を使用して実行されたかどうかを調べるには、Request.debug メソッドを使用します。タグの実行が終了した後に特別なデバッグブロックにデバッグ出力を書き込むには、Response.writeDebug メソッドを使用します。これらのメソッドの使用方法の詳細については、『CFML リファレンス』の ColdFusion Java CFX Reference を参照してください。

Java IDE でのデバッグ

Java IDE を使用して Java CFX タグをデバッグすることができます。つまり、Java CFX タグの開発とデバッグを 1 つの環境で行うことができます。

1. IDE を起動します。
2. プロジェクトのプロパティ (または IDE のプロジェクト設定) で、CFX クラスが "<Web のルートディレクトリ >¥WEB-INF¥classes" ディレクトリまたはシステムクラスパスにあることを確認します。
3. ライブラリ "<ColdFusion のルートディレクトリ >/wwwroot/WEB-INF/lib/cfx.jar" (J2EE 設定では "<ColdFusion Web アプリケーションのルートディレクトリ >/WEB-INF/lib/cfx.jar") および "<ColdFusion のルートディレクトリ >/runtime/lib/jrun.jar" (サーバー設定のみ) が、クラスパスに含まれていることを確認します。
4. プロジェクト設定で、main クラスを jrunx.kernel.JRun に設定し、アプリケーションパラメータを -startdefault に設定します。
5. ブレークポイントの設定、シングルステップ実行、変数の表示などのデバッグアクションを実行して、アプリケーションをデバッグします。

デバッグクラスの使用

Java CFX タグを ColdFusion から分離して開発およびデバッグするには、com.allaire.cfx パッケージにある 3 つの特別なデバッグクラスを使用します。これらのクラスを使用すると、Java 開発環境の対話型デバッガのコンテキスト内で、CFX タグの processRequest メソッドの呼び出しをシミュレートできます。この 3 つのデバッグクラスは次のとおりです。

- DebugRequest : Request インターフェイスの実装。カスタム属性、設定、クエリーを使用してリクエストを初期化できます。
- DebugResponse : Response インターフェイスの実装。リクエストの処理結果を出力できます。
- DebugQuery : Query インターフェイスの実装。名前、列、およびデータセットを使用してクエリーを初期化できます。

main メソッドの実装

1. Java CFX クラスの main メソッドを作成します。
2. main メソッド内で、DebugRequest、DebugResponse、および DebugQuery を初期化します。テスト用の適切な属性およびデータを使用します。
3. Java CFX タグのインスタンスを作成して、その processRequest メソッドを呼び出し、DebugRequest および DebugResponse オブジェクトを渡します。

4 DebugResponse.printResults メソッドを呼び出して、生成された内容、変数セット、生成されたクエリーなどの、リクエストの結果を出力します。

上記の手順に従って main メソッドを実装したら、シングルステップ対話型デバッガを使用して Java CFX タグをデバッグできます。main クラスとして Java CFX クラスを指定し、必要に応じてブレークポイントを設定してデバッグを開始します。

例 : デバッグクラス

デバッグクラスの使用例を次に示します。

```
import java.util.Hashtable ;
import com.allaire.cfx.* ;

public class OutputQuery implements CustomTag {
    // debugger testbed for OutputQuery
    public static void main(String[] argv) {
        try {
            // initialize attributes
            Hashtable attributes = new Hashtable() ;
            attributes.put( "HEADER", "Yes" ) ;
            attributes.put( "BORDER", "3" ) ;

            // initialize query

            String[] columns = { "FIRSTNAME", "LASTNAME", "TITLE" } ;

            String[][] data = {
                { "Stephen", "Cheng", "Vice President" },
                { "Joe", "Berrey", "Intern" },
                { "Adam", "Lipinski", "Director" },
                { "Lynne", "Teague", "Developer" } };

            DebugQuery query = new DebugQuery( "Employees", columns, data ) ;

            // create tag, process debugging request, and print results
            OutputQuery tag = new OutputQuery() ;
            DebugRequest request = new DebugRequest( attributes, query ) ;
            DebugResponse response = new DebugResponse() ;
            tag.processRequest( request, response ) ;
            response.printResults() ;
        }
        catch( Throwable e ) {
            e.printStackTrace() ;
        }
}

    public void processRequest(Request request, Response response) throws Exception {
        // ...code for processing the request...
    }
}
```

C++ を使用した CFX タグの開発

C++ で CFX タグを開発することができます。

C++ のサンプル CFX タグ

C++ で CFX タグを開発する前に、ColdFusion に含まれている 2 つの CFX タグを見ておくと参考になります。これらの例は、CFXAPI を理解するのに役立ちます。2 つのサンプルタグは次のとおりです。

- CFX_DIRECTORYLIST : ディレクトリをクエリーして、ディレクトリ内のファイルのリストを取得します。

- CFX_NTUSERDB (Windows のみ) : Windows NT ユーザーを追加および削除します。

Windows の場合、これらのタグは "<ColdFusion のルートディレクトリ >¥cfx¥examples" ディレクトリにあります。UNIX の場合、これらのタグは "<ColdFusion のルートディレクトリ >/coldfusion/cfx/examples" ディレクトリにあります。

C++ 開発環境の設定

次に示すコンパイラは、UNIX プラットフォームで有効な CFX コードを生成します。

プラットフォーム	コンパイラ
Solaris	Sun Workshop C++ コンパイラ、バージョン 5.0 以降 (Solaris で CFX コードをコンパイルする場合、gcc は使用できません)
Linux	Gnu C++ コンパイラ - gcc

C++ コンパイラを使用してカスタムタグを構築する前に、このコンパイラが CFX API ヘッダファイル "cfx.h" を検索できるように設定する必要があります。Windows では、CFX API インクルードディレクトリをグローバルインクルードパスのリストに追加します。Windows の場合、このディレクトリは "<ColdFusion のルートディレクトリ >¥cfx¥include" です。UNIX の場合、このディレクトリは "<ColdFusion のルートディレクトリ >/cfx/include" です。UNIX では、"-I < インクルードパス >" をコンパイル行で指定する必要があります。ディレクトリリストの例については、"cfx/examples" ディレクトリの Makefile を参照してください。

C++ CFX タグのコンパイル

Windows および UNIX 上で構築する CFX タグはスレッドセーフであることが必要です。Solaris で CFX タグをコンパイルするときは、Sun コンパイラの -mt スイッチを使用してください。

UNIX での C++ ライブラリファイルの配置

UNIX システムでは、C++ ライブラリファイルは、LD_LIBRARY_PATH または SHLIB_PATH (HP-UX のみ) に含まれている任意のディレクトリに配置できます。

C++ CFX タグの実装

C++ の CFX タグでは、C++ の CCFXRequest class によって表されるタグリクエストオブジェクトを使用して処理を行います。このオブジェクトは、アプリケーションページからカスタムタグに対して行われたリクエストを表します。カスタムタグのメインプロシージャには、リクエストオブジェクトのインスタンスを指すポインタが渡されます。カスタムタグでは、リクエストオブジェクトに用意されているメソッドを使用して処理を行います。CFX API のクラスおよびメンバーの詳細については、『CFML リファレンス』の ColdFusion C++ CFX Reference を参照してください。

注意 : 存在しない C++ CFX プロシージャまたはエントリポイントを UNIX 上で呼び出すと、JVM がクラッシュします。

C++ CFX タグのデバッグ

デバッグセッションを設定すると、デバッガ内からのカスタムタグの実行や、ブレークポイント、シングルステップなどの設定ができます。

Windows でのデバッグ

Visual C++ 環境内で、カスタムタグをデバッグできます。

1. デバッグオプションを使用して C++ CFX タグを構築します。
2. ColdFusion を再起動します。
3. Visual C++ を起動します。

4 [ビルド]-[デバッグの開始]-[プロセスへアタッチ] を選択します。

5 "jrunsvc.exe" を選択します。

他のすべての Java プログラムをシャットダウンすることをお勧めします。

6 CFX タグを呼び出す任意の ColdFusion ページを実行します。

7 [ファイル]-[開く] を使用して、ブレークポイントを設定するファイルを VisualDev で開きます。

8 CFX プロジェクトにブレークポイントを設定します。

最適な設定場所は、ProcessRequest() です。次にこのページを実行すると、ブレークポイントで実行が中断されます。

CFX タグの登録

ColdFusion アプリケーションで CFX タグを使用するには、まず ColdFusion Administrator の [拡張機能] にある [CFX タグ] ページで登録します。

1 ColdFusion Administrator で、[拡張機能]-[CFX タグ] を選択します。

2 [C++ CFX の登録] をクリックします。

3 タグ名 (たとえば、cfx_MyNewTag) を入力します。

4 .dll サーバーライブラリのフィールドが空の場合は、ファイルパスを入力します。

5 [プロシージャ] はデフォルトエントリのままにします。

6 タグの開発中は、[ロードしたライブラリの維持] ボックスの選択を解除します。

タグが完成して運用環境で使用できるようになった後は、パフォーマンスを向上させるために、このオプションを選択して DLL をメモリに常駐させておくことができます。

7 (オプション) 説明を入力します。

8 送信ボタンをクリックします。

これで、ColdFusion ページからタグを呼び出すことができます。

CFX タグの削除

1 ColdFusion Administrator で、[拡張機能]-[CFX タグ] を選択します。

2 [登録された CFX タグ] リストで、削除するタグの [コントロール] 列の [削除] アイコンをクリックします。

第 6 章：CFML アプリケーションの開発

ColdFusion アプリケーションの設計と最適化

さまざまなアプリケーション要素を利用し、サーバー上で適切にアプリケーションを構築すれば、Adobe ColdFusion ページを通じて効果的なインターネットアプリケーションを提供できます。"Application.cfc" および "Application.cfm" ファイルやさまざまなコーディング方法を使用して、アプリケーションの効率を最適化できます。

アプリケーションについて

アプリケーションという用語にはさまざまな意味があります。ゲストブックのようなシンプルなアプリケーションもあれば、カタログページ、ショッピングカート、レポート機能などを完備したインターネット商取引システムのような高度なアプリケーションもあります。

ColdFusion では、アプリケーションという用語は特定の意味で使用されます。ColdFusion アプリケーションは、次のような特徴を持っています。

- 連携して機能し共通リソースを共有する ColdFusion ページから構成されます。
- アプリケーション内のすべてのページは、"Application.cfc" ファイルまたは cfapplication タグでの指定に基づいて、同じアプリケーション名および設定を共有します。
- アプリケーション内のすべてのページは、Application スコープの変数を共有します。
- アプリケーション全体に適用可能な、特定のイベント (リクエストの開始、セッションの終了など) 用のイベントハンドラを記述できます。

会社の Web サイトのように、ユーザーにとって単一アプリケーションのように見えるものでも、複数の ColdFusion アプリケーションから構成されている場合があります。

ColdFusion アプリケーションは J2EE アプリケーションではありません。ただし、"Application.cfc" ファイルまたは cfapplication タグでアプリケーション名を指定しなければ、Application スコープは J2EE アプリケーションサーブレットコンテキストに対応するものとなります。

アクセスがないまま ColdFusion アプリケーションのタイムアウト期間が経過した場合や、サーバーが停止した場合、そのアプリケーションは終了します。アプリケーションがタイムアウトすると、Application スコープの変数はすべて解放されます。したがって、タイムアウト期間については、Application スコープのメモリをクリアする必要性とスコープを作成し直すオーバーヘッドとのバランスを考慮して設定する必要があります。多くの場合、アプリケーションタイムアウト期間は 2 日間に設定されます。

ColdFusion のアプリケーションとセッションは互いに独立しています。たとえば、アプリケーションがタイムアウトした場合でも、アクティブなユーザーセッションは継続し、そのセッションコンテキスト (ユーザーの Session スコープの変数など) はアプリケーションの終了や再起動の影響を受けません。

Web アプリケーションを ColdFusion アプリケーションとして構築する方法に一定のルールはありませんが、次のガイドラインが参考になります。

- アプリケーションページには、共通する一般的な目的を持たせます。たとえば、Web 店舗は通常、単一の ColdFusion アプリケーションです。
- ColdFusion アプリケーションのページの多くは、シングルログインなどのメカニズムを提供するために、データや共通コード要素を共有します。
- アプリケーションページには、共通の外観を持たせます。これは、同じヘッダページやフッタページ、共通のエラーメッセージテンプレートなどの共通のコード要素を使用することで実現できます。

ColdFusion アプリケーションの要素

ColdFusion アプリケーションを開発する前に、アプリケーションの構造と、アプリケーション全体のニーズや問題を処理する方法を決定しておきます。特に、次のことをすべて考慮する必要があります。

- アプリケーションの全体的なフレームワーク
- 再利用可能なアプリケーション要素
- 共有変数
- アプリケーションイベントと "Application.cfc" ファイル
- アプリケーションレベルの設定と機能
- アプリケーションのセキュリティとユーザーの識別

アプリケーションフレームワーク

アプリケーションフレームワークとは、アプリケーションの全体的な構造と、その構造をディレクトリ構造やアプリケーションページに反映させる方法のことです。単一のアプリケーションフレームワークでは、複数の ColdFusion アプリケーションを組み合わせて、1 つの Web サイトやインターネットアプリケーションを構築します。ColdFusion アプリケーションは、さまざまな方法で構築できます。たとえば、Fusebox というアプリケーション開発手法は、ColdFusion Web アプリケーションの開発によく採用されるフレームワークです。Fusebox の詳細については、www.fusebox.org を参照してください。

本書では、特定のアプリケーションフレームワークの使用方法や開発方法については説明しません。ここでは、ColdFusion に用意されているフレームワーク構築ツール ("Application.cfc" ファイルなど) や、アプリケーションのディレクトリ構造がアプリケーションに与える影響、およびディレクトリ構造をマッピングする方法について説明します。アプリケーションフレームワークのマッピングの詳細については、235 ページの「アプリケーション構造の作成」を参照してください。

再利用可能なアプリケーション要素

ColdFusion には、頻繁に使用する機能の提供や CFML の拡張に使用できる再利用可能な要素が数多く用意されています。これらの要素には次のものがあります。

- ユーザー定義関数 (UDF)
- CFML カスタムタグ
- ColdFusion コンポーネント (CFC)
- CFX (ColdFusion Extension) タグ
- cfinclude タグを使用してインクルードするページ

これらの要素の概要と、使用する要素の選択基準については、143 ページの「ColdFusion 要素の作成」を参照してください。

共有変数

次の ColdFusion 変数スコープでは、現在の HTTP リクエストのスコープを超えてデータが保持されます。

変数スコープ	変数を使用できる範囲
Server	1 つのサーバー上にあるすべてのアプリケーションと、すべてのクライアント

変数スコープ	変数を使用できる範囲
Application	1 つのアプリケーション内の、すべてのクライアントに対するすべてのページ
Client	1 つのアプリケーション内の、単一のクライアントブラウザに対する複数のブラウザセッション
Session	1 つのアプリケーション内の、単一のクライアントブラウザに対する単一のブラウザセッション

含まれるデータの整合性を保つためのロックの使用方法などの、これらの変数の使用方法については、300 ページの「永続データとロックの使用」を参照してください。

アプリケーションイベントと "Application.cfc" ファイル

アプリケーションイベントとは、アプリケーションのライフサイクル内で発生する個々の事象のことです。イベントが発生するたびに、"Application.cfc" ファイル (アプリケーション CFC とも呼ばれます) に実装した、そのイベントに対応するメソッドが ColdFusion によって実行されます。"Application.cfc" ファイルでは、アプリケーションの設定を定義し、アプリケーションイベントを処理するメソッドを実装します。

次のイベントを処理するアプリケーション CFC メソッドを実装できます。

イベント	トリガ
アプリケーション起動	動作していないアプリケーション内の任意のページに対する最初のリクエストが ColdFusion によって処理開始されると発生します。
アプリケーション終了	アプリケーションのタイムアウト期間が経過するか、サーバーがシャットダウンすると発生します。
セッション開始	既存のセッションに属さないリクエストの結果として新しいセッションが作成されると発生します。
セッション終了	セッションのタイムアウト期間が経過すると発生します。
リクエスト開始	ColdFusion でリクエスト (HTTP リクエスト、イベントゲートウェイへのメッセージ、SOAP リクエスト、Flash Remoting リクエスト) が受信されると発生します。
リクエスト	ColdFusion によるリクエスト開始イベントの処理が完了した直後に発生します。このイベントのハンドラでは、主にリクエスト内容のフィルタ処理を行います。リクエスト開始イベントとリクエストイベントの違いについては、242 ページの「"Application.cfc" によるリクエストの管理」を参照してください。
リクエスト終了	ColdFusion でリクエストに関するすべてのページと CFC の処理が完了すると発生します。
例外	例外が try/catch ブロックで処理されなかった場合に発生します。

"Application.cfc" ファイルでは、アプリケーション名や、そのアプリケーションで Session 変数をサポートするかどうかなどの、アプリケーション全体に関する設定も定義します。

アプリケーションイベントと "Application.cfc" ファイルの使用方法の詳細については、237 ページの「"Application.cfc" でのアプリケーションおよびイベントハンドラの定義」を参照してください。

アプリケーションレベルのその他の設定と機能

新しく作成するコードでアプリケーションレベルの設定、変数、および関数を定義する際には、ColdFusion MX 7 よりも前に使用していた方法を使用しないことをお勧めします。"Application.cfc" ファイルとその変数およびメソッドのほうが高機能であり、構造も論理的に階層化されているので、今後はこの方法を使用してください。

"Application.cfc" ファイルがない場合は、ColdFusion でアプリケーションのページが処理されるたびに、次の 2 つのページが処理されます (存在する場合)。

- "Application.cfm" ページが、アプリケーションの各ページの前に処理されます。
- "OnRequestEnd.cfm" ページが、アプリケーションの各ページの後に処理されます。

注意：UNIX システムでは大文字と小文字が区別されます。UNIX で確実にページが動作するように、"Application.cfm" の A および "OnRequestEnd.cfm" の O、R、E は常に大文字にしてください。

"Application.cfm" ページでは、アプリケーションを定義します。このページには、アプリケーション名を指定する cfapplication タグを含めることができます。また、このページに記述したコードはアプリケーションのすべてのページに対して処理されます。このページでは、アプリケーションレベルの設定、関数、および機能を定義できます。

"OnRequestEnd.cfm" ページは、"Application.cfm" ページよりも使用機会が少ないページですが、すべてのアプリケーションページの後で実行する共通のクリーンアップコードを実装したり、ダイナミックなフッタページを指定したりすることができます。


ページで cflocation タグを実行している場合、"OnRequestEnd.cfm" ページは実行されません。

"Application.cfm" ページと "OnRequestEnd.cfm" ページの詳細については、248 ページの「"Application.cfm" ページの使用」を参照してください。アプリケーションディレクトリ構造におけるこれらのページの配置については、235 ページの「アプリケーション構造の作成」を参照してください。

注意：ColdFusion アプリケーションは、"Application.cfc"、"Application.cfm"、および "OnRequestEnd.cfm" ページを使用しなくても作成できます。ただし、アプリケーションの各ページで cfapplication タグを使用して共通のアプリケーション要素を定義するよりも、"Application.cfm" ページを使用するほうが簡単です。

アプリケーションごとの設定

アプリケーションごとに次の項目を設定します。

- マッピング
- カスタムタグのパス

これらの設定は、指定されたアプリケーションに限り、ColdFusion Administrator のサーバーサイドの設定よりも優先されます。アプリケーションごとの設定を指定しても、サーバー全体の設定は変更されません。アプリケーションごとの設定を行うには、まず、ColdFusion Administrator の [設定] ページでアプリケーションごとの設定を有効にします。その後、"Application.cfc" ファイルでマッピングまたはカスタムタグのパスを設定します。

アプリケーションごとの設定で指定したカスタムタグは、ColdFusion Administrator で定義されているカスタムタグよりも優先されます。たとえば、同じ名前のカスタムタグが 2 つあり、Administrator の設定とアプリケーションごとの設定でそれらの場所が異なる場合は、アプリケーションごとの設定で指定されているカスタムタグが最初に取得されます。

注意：アプリケーションごとの設定は、"Application.cfc" ファイルを使用するアプリケーションでのみサポートされており、"Application.cfm" を使用するアプリケーションではサポートされていません。Administrator の [メモリ変数] ページでアプリケーション変数を無効にしている場合、アプリケーションごとの設定は機能しません。

アプリケーションごとのマッピングの設定

1. ColdFusion Administrator の [設定] ページにある [アプリケーションごとの設定の有効化] オプションにチェックを付けます。
2. 次のようなコードを "Application.cfc" ファイルに含めます。

```
<cfset THIS.mappings["/MyMap"]="c:\inetpub\myStuff">
```

または

```
<cfset StructInsert(THIS.mappings, "/MyMap", "c:\inetpub\myStuff")>
```

2 つ目の形式を使用する場合は、先に THIS.mappings 構造体を作成する必要があります。

アプリケーションごとのカスタムタグのパスの設定

1 ColdFusion Administrator の [設定] ページにある [アプリケーションごとの設定の有効化] オプションにチェックを付けます。

2 次のようなコードを "Application.cfc" ファイルに含めます。

```
<cfset customtagpaths = "c:\mapped1,c:\mapped2">
<cfset customtagpaths = ListAppend(customtagpaths,"c:\mapped3")>
<cfset This.customtagpaths = customtagpaths>
```

アプリケーションのセキュリティとユーザーの識別

すべてのアプリケーションは、悪意のあるユーザーにリソースを不正に使用されないようにする必要があります。また、多くのアプリケーションでは、サイト内のユーザーアクセスが可能な領域の管理、ユーザーが実行できる操作の管理、ユーザー固有のコンテンツの提供などを行うために、ユーザーの識別機能が必要になります。ColdFusion には、これらの問題に対処するために、次のようなアプリケーションセキュリティが用意されています。

リソース (ファイルベースまたはディレクトリベースの) セキュリティ 特定のディレクトリにあるアプリケーションページがアクセスできる ColdFusion リソース (タグ、関数、データソースなど) を制限します。アプリケーションのディレクトリ構造をデザインするときには、アプリケーションのリソースセキュリティの必要性を検討する必要があります。

ユーザー (プログラムによる) セキュリティ ユーザーが利用できるアプリケーション機能を制限する方法として、認証 (ログイン) メカニズムとロールベースの承認メカニズムが用意されています。ユーザーセキュリティではユーザー ID も利用できるので、この ID に基づいてページコンテンツをカスタマイズできます。ユーザーセキュリティを実装するには、cflogin タグや cfloginuser タグなどのセキュリティコードをアプリケーションに含めます。

セキュリティの実装の詳細については、337 ページの「アプリケーションの保護」を参照してください。

アプリケーション構造の作成

ColdFusion アプリケーションを設計するときは、ディレクトリとファイルから構成される構造としてそのコンテンツを作成します。この作業はディレクトリ構造のマッピングと呼ばれ、ColdFusion アプリケーションの設計における重要な手順の 1 つです。アプリケーションの構築を始める前に、アプリケーションのルートディレクトリを設定します。アプリケーションページは、ルートディレクトリのサブディレクトリに保存します。

ColdFusion によるアプリケーション定義ページの検索および処理方法

ColdFusion では、アプリケーション固有の要素が定義されている "Application.cfc"、"Application.cfm"、および "OnRequestEnd.cfm" ページが、次の規則に従って検索され処理されます。ColdFusion のファイル検索方法を理解しておくと、アプリケーション構造を作成するときに役立ちます。

ColdFusion で個々のページリクエストが処理されるたびに、次の処理が行われます。

1 ColdFusion でリクエストの処理が開始されると、次の処理が行われます。

- ページのディレクトリで "Application.cfc" ファイルが検索されます。このファイルがある場合は、CFC の新しいインスタンスが作成され、初期イベントが処理されて、ファイルの検索が停止します。ColdFusion では、リクエストごとに新しい CFC インスタンスが作成され、その初期化コードが実行されます。
- リクエストされたページのディレクトリに "Application.cfc" ファイルがない場合は、"Application.cfm" ファイルが検索されます。このファイルがある場合は、リクエストされたページの先頭に "Application.cfm" ページが論理的にインクルードされて、ファイルの検索が停止します。
- リクエストされたページのディレクトリに "Application.cfc" ファイルも "Application.cfm" ファイルもない場合は、ディレクトリツリーの階層が上方向にたどられていき、各ディレクトリでまず "Application.cfc" が検索され、なければ "Application.cfm" が検索されます。この検索はルートディレクトリ ("C:¥" など) に到達するまで続けられます。

"Application.cfc" ファイルまたは "Application.cfm" ファイルが見つかった場合は、そのページが処理され、ファイルの検索が停止します。

2 リクエストされたページのコンテンツが処理されます。

3 ColdFusion でリクエストの処理が終了するときには、次の処理が行われます。

- "Application.cfc" がある場合は、その CFC の onRequestEnd メソッドが実行され、その CFC インスタンスが解放されます。
- "Application.cfc" がなく、"Application.cfm" がある場合は、現在のページで使用された "Application.cfm" ページと同じディレクトリで "OnRequestEnd.cfm" が検索されます。検索はそのディレクトリでのみ行われるので、他のディレクトリに存在する "OnRequestEnd.cfm" ページは実行されません。また、アプリケーションページでエラーまたは例外が発生した場合や、アプリケーションページで cfabort タグまたは cfexit タグが実行された場合も、"OnRequestEnd.cfm" ページは実行されません。

アプリケーションページおよび設定は、次の規則に従って処理されます。

- 1 つのリクエストに対して処理される "Application.cfc" または "Application.cfm" ページは、1 つのみです。ColdFusion ページ内に追加の ColdFusion ページを指す cfinclude タグがあっても、その追加ページをインクルードするときに "Application.cfc" および "Application.cfm" ページの検索は行われません。
- cfapplication タグがページに含まれている場合は、まず "Application.cfc" または "Application.cfm" が処理され、それから cfapplication タグが処理されます。このタグによって、アプリケーションファイルで指定した設定が、cfapplication タグ属性で設定したアプリケーション名や動作などによって上書きされます。
- 複数の "Application.cfc" ファイル、"Application.cfm"、および cfapplication タグで、同じアプリケーション名を使用することもできます。その場合は、同じ名前を持つすべてのページで、同じアプリケーション設定や Application スコープが共有され、このスコープ内のすべての変数が設定および取得されます。セッションタイムアウトなどの設定がファイルによって異なる場合は、cfapplication タグのパラメータか、最後に処理されたファイルのパラメータ設定が使用されます。

注意：UNIX プラットフォームでアプリケーションを実行する場合は、大文字と小文字が区別されるので、"Application.cfc"、"Application.cfm"、および "OnRequestEnd.cfm" の大文字と小文字を正確に使い分ける必要があります。

ディレクトリ構造の定義

アプリケーション固有のルートディレクトリを使用してアプリケーションのディレクトリ構造を定義すると、次の利点が得られます。

開発 アプリケーションページファイルが系統立てて整理されるので、アプリケーションの開発と保守が簡単になります。

可搬性 アプリケーションページファイル内のコードを変更することなく、サーバー間またはサーバー内で、アプリケーションを簡単に移動できます。

アプリケーションレベルの設定 同じディレクトリの下にあるアプリケーションページでは、アプリケーションレベルの設定と機能を共有できます。

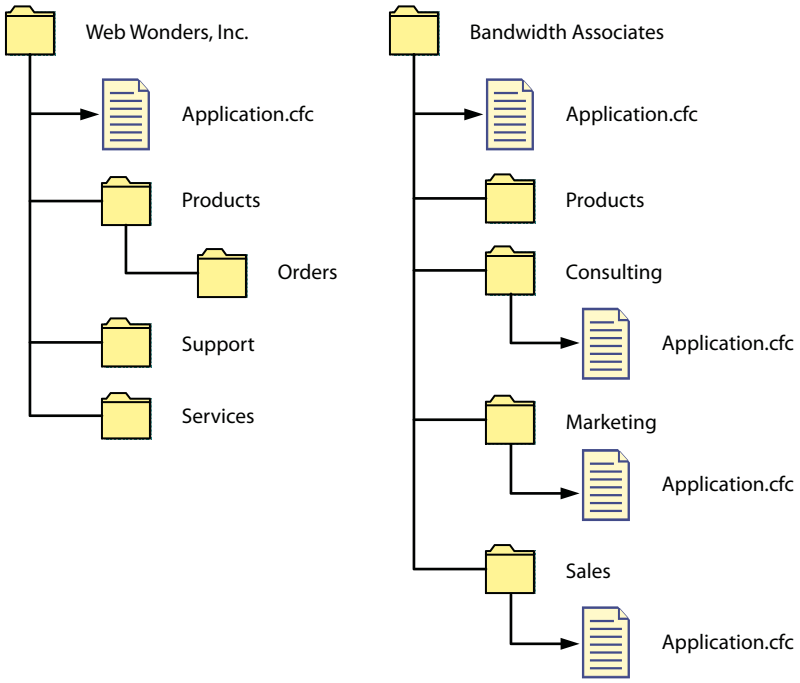
セキュリティ 同じディレクトリの下にあるアプリケーションページでは、Web サーバーのセキュリティ設定を共有できます。

アプリケーション固有のディレクトリ階層にアプリケーションを配置する際には、単一のアプリケーション定義 ("Application.cfc" または "Application.cfm") ページをアプリケーションのルートディレクトリに置いて使用することも、異なるアプリケーション定義ページを複数のディレクトリに置いてアプリケーションのセクション別に管理することもできます。

複数のアプリケーション定義ページを使用して異なるアプリケーション名を付ければ、論理的な Web アプリケーションが複数の ColdFusion アプリケーションに分割されます。また、複数のアプリケーション定義ページを使用して同じアプリケーション名を付ければ、1 つのアプリケーション内でサブセクションごとに異なるコードが実装されます。

次の図のディレクトリツリーは、アプリケーションフレームワークの 2 つの実装方法を示しています。

- 左側の例の Web Wonders, Inc. という会社では、アプリケーションのルートディレクトリに 1 つの "Application.cfc" ファイルをインストールして、すべてのアプリケーションページリクエストを処理しています。
- 右側の例の Bandwidth Associates では、個々の "Application.cfc" ファイルの設定を使用して、ColdFusion アプリケーションを部門レベルで作成しています。ルートの "Application.cfc" ファイルの設定を使用して処理されるのは、Products アプリケーションのページのみです。Consulting、Marketing、および Sales のディレクトリには、それぞれ専用の "Application.cfc" ファイルがあります。



"Application.cfc" でのアプリケーションおよびイベントハンドラの定義

"Application.cfc" ファイルでは、アプリケーション全体に適用する設定と変数、およびアプリケーションイベントハンドラを定義します。

- アプリケーション全体に適用する設定および変数としては、ページ処理の設定、デフォルト変数、データソース、スタイル設定、その他のアプリケーションレベルの定数などがあります。
- アプリケーションイベントハンドラは CFC メソッドの一種で、アプリケーションの有効期間中に特定のイベントが発生すると ColdFusion によって自動的に実行されます。該当するイベントは、アプリケーション起動、アプリケーション終了、セッション開始、セッション終了、リクエスト開始、リクエスト実行、リクエスト終了、および例外です。

アプリケーションレベルの設定および変数の定義

アプリケーションを作成する際には、アプリケーション全体に適用するプロパティや特性を多数設定できます。これには次のような項目が含まれます。

- アプリケーション名
- アプリケーションプロパティ (Client 変数、Application 変数、Session 変数の管理オプションを含む)
- ページ処理オプション
- デフォルト変数、データソース、スタイル設定、および他のアプリケーションレベルの定数

デフォルト変数の設定については、241 ページの「onApplicationStart におけるアプリケーションのデフォルト変数と定数の設定」を参照してください。

アプリケーションのネーミング

アプリケーションを定義し、アプリケーション名を付けるには、"Application.cfc" の初期化セクションで、メソッド定義よりも前に This.name 変数を設定します。同じアプリケーション名を使用すれば、複数のページを同じ論理アプリケーションの構成要素として定義できます。

ColdFusion では、名前を持たないアプリケーションがサポートされています。これは、JSP タグおよびサーブレットと相互運用する必要がある ColdFusion アプリケーションにおいて便利です。名前を持たないアプリケーションの作成を検討するのは、ColdFusion ページで Application スコープまたは Session スコープのデータを既存の JSP ページやサーブレットと共有する必要がある場合のみとしてください。名前のないアプリケーションを 1 つのサーバーインスタンス上に複数作成することはできません。名前のないアプリケーションの使用については、1114 ページの「ColdFusion ページおよび JSP ページまたはサーブレット間でのデータ共有」を参照してください。

アプリケーションプロパティの設定

アプリケーションプロパティは、"Application.cfc" の初期化コードで This スコープの変数を設定することによって指定します。これは、cfapplication タグで設定できるのと同じプロパティです。次の表に、アプリケーションプロパティの設定に使用する This スコープ変数とその使用方法を示します。

変数	デフォルト	説明
applicationTimeout	Administrator の値	アプリケーション (すべての Application スコープ変数も含む) の有効期間を示す実数値 (日数)。この変数を生成するには createTimeSpan 関数を使用します。
clientManagement	False	アプリケーションで Client スコープ変数をサポートするかどうかを示します。
clientStorage	Administrator の値	Client 変数を格納する場所を示します。Cookie、レジストリ、またはデータソース名を指定できます。
loginStorage	Cookie	ログイン情報を Cookie スコープと Session スコープのいずれに格納するかを示します。
scriptProtect	Administrator の値	変数をクロスサイトスクリプティング攻撃から保護するかどうかを示します。
sessionManagement	False	アプリケーションで Session スコープ変数をサポートするかどうかを示します。
sessionTimeout	Administrator の値	ユーザーセッション (すべての Session 変数も含む) の有効期間を示す実数値 (日数)。この変数を生成するには createTimeSpan 関数を使用します。
setClientCookies	True	クライアントブラウザに CFID Cookie および CFTOKEN Cookie を送信するかどうかを示します。
setDomainCookies	False	クライアント識別用の CFID および CFTOKEN の値と、Cookie を使用して格納される Client スコープの変数について、ドメイン Cookie を使用するかどうかを示します。False の場合はホスト固有の Cookie が使用されます。クラスタで実行されるアプリケーションの場合は True に設定します。

次の例は "Application.cfc" の冒頭部分のコードで、アプリケーション名とプロパティを設定しています。

```
 <cfcomponent>
<cfset This.name = "TestApplication">
<cfset This.clientmanagement="True">
<cfset This.loginstorage="Session">
<cfset This.sessionmanagement="True">
<cfset This.sessiontimeout="#createtimespan(0,0,10,0)#">
<cfset This.applicationtimeout="#createtimespan(5,0,0,0)#">
```

これらの設定の詳細については、『CFML リファレンス』の cfapplication を参照してください。

ページ処理オプションの設定

cfsetting タグを使用すれば、アプリケーションの全ページに適用する次のページ処理属性を指定できます。

属性	用途
showDebugOutput	デバッグ出力を表示するかどうかを指定します。ColdFusion Administrator でデバッグが無効になっている場合、この設定でデバッグを有効にすることはできません。Administrator でデバッグ出力が有効になっている場合に、デバッグ出力を表示するか非表示にするかをこの設定で指定します。
requestTimeout	ページリクエストのタイムアウトを指定します。タイムアウトまでに ColdFusion でページの処理が完了できなかった場合は、エラーが生成されます。この設定は、ColdFusion Administrator での設定よりも優先されます。外部の LDAP サーバーや Web サービスプロバイダなど、特に低速の外部リソースにアプリケーションやページでアクセスすることが多い場合は、この設定を使用してページのタイムアウト時間を増やします。
enableCFOutputOnly	cfoutput タグに含まれていないテキストを出力しないようにします。ColdFusion ページに含まれている不要なテキストを非表示にしたい場合に、この設定が役立ちます。

cfsetting は個別のページで使用されることが多いタグですが、"Application.cfc" ファイルでも使用できます。たとえば、マルチアプリケーション環境で使用して、ColdFusion Administrator の設定をアプリケーションごとに上書きできます。

コンポーネントの初期化コードでも、アプリケーション全体に適用する cfsetting タグを記述できます。その場合は通常、次の例のように This スコープのアプリケーションプロパティ設定に続いて記述します。

```
 <cfcomponent>
<cfscript>
    This.name="MyAppl";
    This.clientmanagement="True";
    This.loginstorage="Session" ;
    This.sessionmanagement="True" ;
    This.sessiontimeout=CreateTimeSpan(0,0,1,0);
</cfscript>
<cfsetting showdebugoutput="No" enablecfoutputonly="No">
```

この例に含まれる cfsetting タグは、アプリケーション内のすべてのページに影響します。アプリケーション全体の設定は、イベントメソッド (onRequestStart など) の中や個別の ColdFusion ページで上書きできます。

アプリケーションイベントハンドラの使用

次の表では、実装可能なアプリケーションイベント CFC メソッドと、それらがトリガされるタイミングについて説明します。

メソッド	実行のタイミング
onApplicationStart	アプリケーションが最初に起動するときに発生します。つまり、任意のページに対する最初のリクエストが処理されるときか、CFC メソッドがイベントゲートウェイインスタンス、Flash Remoting リクエスト、または Web サービス呼び出しによって最初に呼び出されるときに発生します。 このメソッドは、データソース名など、アプリケーション全体で使用する (Application スコープの) 変数を設定するのに便利です。
onApplicationEnd	アプリケーションが終了するときに発生します。つまり、アプリケーションがタイムアウトするか、サーバーがシャットダウンすると発生します。
onSessionStart	既存のセッション (ColdFusion イベントゲートウェイセッションを含む) に属さないリクエストの結果として、新しいセッションが作成されると発生します。このイベントが発生するには、アプリケーションでセッションが有効になっている必要があります。
onSessionEnd	セッションのタイムアウト期間が経過すると発生します。このイベントは、アプリケーションのタイムアウトやサーバーのシャットダウンでは発生**しません**。
onRequestStart	ColdFusion でリクエスト (ブラウザなどからの HTTP リクエスト、イベントゲートウェイへのメッセージ、SOAP リクエスト、Flash Remoting リクエスト) が受信されると発生します。

メソッド	実行のタイミング
onRequest	onRequestStart イベントが完了すると発生します。このメソッドは、リクエストされたページコンテンツに対するフィルタの役割を果たします。
onRequestEnd	リクエストに関するすべてのページや CFC の処理が完了すると発生します。"OnRequestEnd.cfm" ページと同等です。
onMissingTemplate	ColdFusion で、存在しないページに対する要求が受信されると発生します。
onError	例外が try/catch ブロックで処理されなかった場合に発生します。

ColdFusion でリクエストが受信されると、Application CFC がインスタンス化され、"Application.cfc" のコードが次の順序で実行されます。

- ファイル冒頭の CFC 初期化コード
- onApplicationStart (そのアプリケーションでまだ実行されていない場合)
- onSessionStart (そのセッションでまだ実行されていない場合)
- onRequestStart
- onRequest か、onRequest メソッドがない場合はリクエストされたページ
- onRequestEnd

次のメソッドは特定のイベントによってトリガされます。

- onApplicationEnd
- onSessionEnd
- onMissingTemplate
- onError

onApplicationEnd メソッドと onSessionEnd メソッドは、ページリクエストのコンテキストでは実行されないので、リクエスト変数にアクセスすることや、ユーザーに対して情報を表示することはできません。onMissingTemplate メソッドは、存在しない CFML ページが URL で指定されるとトリガされます。OnError メソッドは、リクエストのコンテキストで実行されるとは限りません。コンテキストは Event 引数を使用して判別します。

"Application.cfc" によるアプリケーションの管理

アプリケーションの設定と管理には、onApplicationStart メソッドと onApplicationEnd メソッドを使用します。複数のページやリクエストで使用され、アプリケーション内のすべてのコードから一貫して利用可能であることが必要なリソースは、これらのメソッドによって制御します。そのようなリソースとしては、データソース、アプリケーションカウンタ (ページヒット変数など)、すべてのページに適用するスタイル情報などがあります。

onApplicationStart メソッドは、サーバーの起動後に、アプリケーションの任意のページに対する最初のリクエストが ColdFusion で受信されると実行されます。onApplicationEnd メソッドは、アプリケーションサーバーがシャットダウンするか、アクセスがないままアプリケーションのタイムアウト期間が経過すると実行されます。

次に、これらのメソッドを使用するいくつかの方法について説明します。詳細については、『CFML リファレンス』で onApplicationStart および onApplicationEnd のエントリを参照してください。

アプリケーションユーティリティ関数の定義

"Application.cfc" で定義した関数のうち共有スコープに属していないものは、デフォルトではその CFC 内の他のメソッドからのみ使用可能です。

"Application.cfc" で onRequest メソッドを実装している場合、"Application.cfc" で定義したすべてのユーティリティ関数は、ターゲットページでも直接使用できます。これは、"Application.cfc" とターゲットページでは同じ Variables スコープが共有されるためです。

"Application.cfc" 内のメソッドだけでなく複数のページでユーティリティ関数を使用する必要があり、onRequest メソッドを使用しない場合は、別の CFC でその関数を定義し、その CFC を実行してアクセスすることをお勧めします。他の ColdFusion ページと同様に、"Application.cfc" でも、ColdFusion Administrator の [マッピング] ページで設定したディレクトリパス内のすべての CFC にアクセスできます。したがって、この方法を使用してユーティリティ関数を複数のアプリケーションで共有します。

リクエストページで使用するユーティリティ関数を "Application.cfc" で定義し、onRequest メソッドを使用しない場合は、次に示すようなコードを使用して、それらの関数を何らかの ColdFusion スコープ (Request スコープなど) に明示的に配置します。

```
<cffunction name="theFunctionName" returntype="theReturnType">
    <!--- Function definition goes here. --->
</cffunction>

<cffunction name="OnRequestStart">
    <!--- OnRequestStart body goes here --->
        <cfset Request.theFunctionName=This.theFunctionName>
</cffunction>
```

リクエストページには、次のようなコードを記述します。

```
<cfset myVar=Request.theFunctionName(Argument1...)>
```

この方法で定義した関数では、This スコープと Variables スコープがリクエストの "Application.cfc" ファイルと共有されます。

onApplicationStart におけるアプリケーションのデフォルト変数と定数の設定

"Application.cfc" でデフォルト変数とアプリケーションレベルの定数を設定できます。たとえば、次のことが行えます。

- データソースの指定および可用性の確保
- ドメイン名の指定
- フォントやカラーなどのスタイル設定
- その他のアプリケーションレベル変数の設定

"Application.cfc" の onApplicationStart メソッドで Application スコープの変数を設定する場合、それらの変数をロックする必要はありません。

onApplicationStart メソッドを実装する方法の詳細については、『CFML リファレンス』の onApplicationStart を参照してください。

onApplicationEnd メソッドの使用

onApplicationEnd メソッドは、アプリケーションのシャットダウン時にクリーンアップ処理が必要な場合に使用できます。たとえば、メモリ内にあるデータをデータベースに保存することや、アプリケーションの終了をログに記録することができます。このメソッドはリクエストに関連付けられていないので、ユーザーページにデータを表示することはできません。このメソッドで例外が発生した場合でもアプリケーションは終了します。頻繁に使用されるアプリケーションでは、サーバーのシャットダウン時を除き、このメソッドが実行されることはほとんどありません。

onApplicationEnd メソッドを実装する方法の詳細については、『CFML リファレンス』の onApplicationEnd を参照してください。

"Application.cfc" によるセッションの管理

ユーザーセッションの設定と管理には、onSessionStart メソッドと onSessionEnd メソッドを使用します。ユーザーが 1 回のブラウザセッションでサイトにアクセスする間に複数のページで使用されるリソースは、これらのメソッドによって制御できます。アプリケーション内のページをユーザーが最初にリクエストするとセッションが開始し、そのセッションがタイムアウトするとセッションは終了します。Session スコープと Session 変数の詳細については、300 ページの「永続データとロックの使用」を参照してください。

セッションリソースには、セッションの全期間にわたって必要なデータ (口座番号やショッピングカートの内容など) を格納する変数や、セッション中に複数のページから使用するメソッドおよびデータが含まれている CFC などがあります。

注意：onSessionStart メソッドのコードはセッションが開始するときにしか実行されないので、cflogin タグや基本ログイン処理は記述しないでください。ユーザーのログアウトが処理できず、完全なセキュリティを確保できません。

onSessionStart メソッドの使用

このメソッドは、セッションデータ (ユーザー設定やショッピングカートの内容など) の初期化や、アクティブなセッションの数を監視するなどの用途に適しています。このメソッドでセッション変数を設定する場合、Session スコープをロックする必要はありません。

詳細については、『CFML リファレンス』で onSessionStart のエントリを参照してください。

onSessionEnd メソッドの使用

このメソッドは、セッション終了時に任意のクリーンアップ処理を実行するために使用します。セッション終了の詳細については、313 ページの「セッションの終了」を参照してください。たとえば、セッション関連データ (ショッピングカートの内容や、ユーザーが注文処理を完了したかどうかなど) をデータベースに保存することや、セッション終了のログをファイルに記録することができます。このメソッドはリクエストに関連付けられていないので、ユーザーページにデータを表示することはできません。

注意：アプリケーションが終了しても、セッションは終了しないので onSessionEnd メソッドは呼び出されません。詳細については、『CFML リファレンス』で onSessionEnd のエントリを参照してください。

"Application.cfc" によるリクエストの管理

ColdFusion でリクエストを管理する方法としては、onRequestStart、onRequest、および onRequestEnd の 3 つがあります。ColdFusion では、リクエスト (これらのメソッドを含む) が次のように処理されます。

1. リクエストの開始時には、必ず onRequestStart が処理されます。
2. onRequest メソッドが実装されている場合はそれが処理され、実装されていない場合はリクエストされたページが処理されます。onRequest メソッドを実装する場合は、リクエストされたページを onRequest メソッド内で明示的に呼び出します。
3. リクエストの終了時には、必ず onRequestEnd が処理されます。

次のように "Application.cfc" のリクエストメソッドを使用すると、リクエストを管理できます。

onRequestStart メソッドの使用

このメソッドはリクエストの開始時に実行されます。これはユーザーの承認 (ログイン処理) や、リクエスト固有の変数の初期化 (パフォーマンス統計の収集など) に適しています。

onRequestStart メソッドを使用して onRequest メソッドを使用しなかった場合は、onRequestStart コードの処理が完了すると、ColdFusion によってリクエストが自動的に処理されます。

注意："Application.cfm" ファイルに onRequest メソッドを記述しなかった場合、onRequestStart メソッドとリクエストされたページの間で Variables スコープは共有されません。ただし、Request スコープの変数は共有されます。

詳細については、『CFML リファレンス』で onRequestStart のエントリを参照してください。

ユーザー認証

アプリケーションでユーザーのログインが必要な場合は、onRequestStart メソッドに認証コード (cflogin タグ、またはこのタグを呼び出すコードなど) を記述します。これによって、各リクエストの開始時にユーザーが認証されるようになります。セキュリティおよびログイン作成の詳細については、337 ページの「アプリケーションの保護」を参照してください。
Adobe Dreamweaver の CF ログインウィザードで生成した認証コードを使用する例については、『CFML リファレンス』の onRequestStart を参照してください。

onRequest メソッドの使用

onRequest メソッドには、onRequestStart メソッドとの重要な違いが 1 つあります。それは、onRequest メソッドによってユーザーのリクエストがさえぎられるという点です。これは、次の 2 つのことを意味します。

- リクエストを明示的に (たとえば cfinclude タグを使用して) 呼び出さない限り、リクエストは処理されません。したがって、onRequest メソッドを使用することで、リクエストされたページコンテンツをフィルタ処理することや、表示するページまたはページコンテンツを決定するスイッチを実装することができます。
- cfinclude を使用してリクエストを処理すると、その CFC インスタンスとリクエストされたページの間で Variables スコープが共有されます。したがって、その CFC のすべてのメソッドからそのページの Variables スコープの変数を設定できます。また、onRequestEnd メソッドを使用して、インクルードページで設定または変更された Variable スコープの任意の変数にアクセスできます。たとえば、ページ内で使用される変数を onRequestStart メソッドまたは onRequest メソッドで設定することができます。

このメソッドをフィルタとして使用するには、次の例のように cfsavecontent タグの内側に cfinclude タグを記述します。

```
<cffunction name="onRequest">
    <cfargument name = "targetPage" type="String" required=true/>
    <cfsavecontent variable="content">
        <cfinclude template=#Arguments.targetPage#>
    </cfsavecontent>
    <cfoutput>
        #replace(content, "report", "MyCompany Quarterly Report", "all")#
    </cfoutput>
</cffunction>
```

詳細については、『CFML リファレンス』で onRequest のエントリを参照してください。

onRequestEnd メソッドの使用

onRequestEnd メソッドでは、各リクエストの終了時に実行するコードを記述します。ColdFusion MX 6.1 までのバージョンでは、そのようなコードは "OnRequestEnd.cfm" ページに記述していました。このメソッドの代表的な用途としては、ダイナミックなフッタページの挿入などがあります。例については、『CFML リファレンス』の onSessionEnd を参照してください。

注意： "Application.cfm" ファイルに onRequest メソッドを記述しなかった場合、onRequestEnd メソッドとリクエストされたページの間で Variables スコープは共有されません。ただし、Request スコープの変数は共有されます。

詳細については、『CFML リファレンス』で onRequestEnd のエントリを参照してください。

"Application.cfc" でのエラー処理

以降のセクションでは、"Application.cfc" でエラーを処理する方法について簡単に説明します。エラーページとエラー処理の詳細については、275 ページの「エラー処理」を参照してください。onError メソッドを実装する方法の詳細については、『CFML リファレンス』の onError を参照してください。

"Application.cfc" でのエラー処理方法

"Application.cfc" では、次の方法を組み合わせてエラーを処理します。

- イベントメソッド (onApplicationStart、onRequestStart など) の中で try/catch エラーハンドラを使用して、イベントメソッド内で発生する例外を処理します。
- onError メソッドを実装します。CFML コード内の try/catch ハンドラで直接処理されない例外は、すべてこのメソッドに送られます。cfthrow タグをこのメソッドで使用すれば、そこで処理しないエラーをすべて ColdFusion に処理させることができます。
- cferror タグは、アプリケーションの初期化コード内で使用します。これは cfcomponent タグの後の、通常はアプリケーションの This スコープ変数を設定するコードの後に記述します。これらのタグでは、onError メソッドを実装しない場合のエラー処理方法、または onError メソッドでエラーが発生した場合の処理方法を指定します。また、アプリケーション固有の検証エラーハンドラを実装することもできます。それには、たとえば次のようなタグを CFC の初期化コード内に記述します。

```
<cferror type="VALIDATION" template="validationerrorhandler.cfm">
```

- 以上のいずれの方法でも処理されないエラーは、ColdFusion のデフォルトのエラーメカニズムによって処理されます。デフォルトのメカニズムには、サイト全体のエラーハンドラ (ColdFusion Administrator で指定) や、ビルトインのデフォルトエラーページなどがあります。

これらの方法を使用すれば、エラーメッセージにアプリケーション固有の情報 (連絡先情報、アプリケーションやバージョンの識別子など) を含めることができ、また、アプリケーション内のすべてのエラーメッセージを一貫した方法で表示できます。"Application.cfc" ページを使用することで、アプリケーション全体にわたる高度なエラー処理を実装できます。たとえば、エラー処理メソッドで特定のメッセージを表示したり、構造化されたエラー処理メカニズムを実現することができます。

注意 : onError メソッドは、アプリケーションイベントメソッド onSessionEnd および onApplicationEnd 内で発生したエラーを処理しますが、リクエストコンテキストが存在していないので、ユーザーに対してメッセージは表示されません。onError 関数を使用すると、セッション終了時またはアプリケーション終了時に発生したエラーがログに記録されます。

onError メソッドによるサーバーサイド検証エラーの処理

サーバーサイド検証エラーは、実際には ColdFusion 例外です。したがって、アプリケーションで onError メソッドを使用している場合はこのメソッドにエラーが送られるので、そこで処理するか ColdFusion に渡して処理させる必要があります。

サーバーサイド検証エラーを識別するには、Arguments.< 例外 >.StackTrace フィールドに coldfusion.filter.FormValidationException が含まれるかどうかを確認します。その後、エラーを onError ルーチンで直接処理するか、またはこのエラーを再度投げて、ColdFusion デフォルトの検証エラーページか、"Application.cfc" 初期化コードの cferror タグで指定したページに処理させます。

例 : onError メソッドによるエラー処理

次に示す "Application.cfc" ファイルには、エラーを次のように処理する onError メソッドが含まれています。

- サーバーサイド検証エラーである場合は、ColdFusion にエラーを投げて処理させ、その結果として標準の検証エラーメッセージを表示します。
- その他のタイプの例外である場合は、エラーが発生したイベントメソッドの名前を表示し、例外情報をダンプします。この例では、"Application.cfc" のメソッドではなく CFM ページでしかエラーを生成していないので、イベント名は常に空の文字列になります。

```
<cfcomponent>
<cfset This.name = "BugTestApplication">
<cffunction name="onError">
    <!--- The onError method gets two arguments:
            An exception structure, which is identical to a cfcatch variable.
            The name of the Application.cfc method, if any, in which the error
            happened.
    <cfargument name="Except" required=true/>
    <cfargument type="String" name = "EventName" required=true/>
    <!--- Log all errors in an application-specific log file. --->
    <cflog file="#This.Name#" type="error" text="Event Name: #Eventname#" >
    <cflog file="#This.Name#" type="error" text="Message: #except.message#">
    <!--- Throw validation errors to ColdFusion for handling. --->
    <cfif Find("coldfusion.filter.FormValidationException",
                    Arguments.Except.StackTrace)>
        <cfthrow object="#except#">
    <cfelse>
        <!--- You can replace this cfoutput tag with application-specific
                error-handling code. --->
        <cfoutput>
            <p>Error Event: #EventName#</p>
            <p>Error details:<br>
            <cfdump var=#except#></p>
        </cfoutput>
    </cfif>
</cffunction>
</cfcomponent>
```

この例をテストするには、次のコードを含んだ CFML ページを "Application.cfc" ファイルと同じページに作成し、テキスト入力フィールドに、有効なテキストや有効でないテキストを入力します。

```
<cfform>
    This box does Integer validation:
    <cfinput name="intinput" type="Text" validate="integer" validateat="onServer"><br>
    Check this box to throw an error on the action page:
    <cfinput type="Checkbox" name="throwerror"><br>
    <cfinput type="submit" name="submitit">
    </cfform>
<cfif IsDefined("form.fieldnames")>
    <cfif IsDefined("form.throwerror")>
        <cfthrow type="ThrownError" message="This error was thrown from the bugTest action page.">
    <cfelseif form.intinput NEQ "">
        <h3>You entered the following valid data in the field</h3>
        <cfoutput>#form.intinput#</cfoutput>
    </cfif>
</cfif>
```

注意：サーバーサイド検証エラーの詳細については、729 ページの「データの検証」を参照してください。

例 : 完全な "Application.cfc"

次の例は、すべてのアプリケーションイベントハンドラの基本的な使用方法を示す、単純な "Application.cfc" ファイルです。

```
 <cfcomponent>
<cfset This.name = "TestApplication">
<cfset This.Sessionmanagement=true>
<cfset This.Sessiontimeout="#createtimespan(0,0,10,0)#">
<cfset This.applicationtimeout="#createtimespan(5,0,0,0)#">

<cffunction name="onApplicationStart">
    <cftry>
        <!--- Test whether the DB that this application uses is accessible
                by selecting some data. --->
        <cfquery name="testDB" dataSource="cfdocexamples" maxrows="2">
            SELECT Emp_ID FROM employee
        </cfquery>
        <!--- If we get database error, report it to the user, log the error
            information, and do not start the application. --->
        <cfcatch type="database">
            <cfoutput>
                This application encountered an error.<br>
                Please contact support.
            </cfoutput>
            <cflog file="#This.Name#" type="error"
                text="cfdocexamples DB not available. message: #cfcatch.message#
                Detail: #cfcatch.detail# Native Error: #cfcatch.NativeErrorCode#">
            <cfreturn False>
        </cfcatch>
    </cftry>
    <cflog file="#This.Name#" type="Information" text="Application Started">
    <!--- You do not have to lock code in the onApplicationStart method that sets Application scope
variables. --->
    <cfscript>
        Application.availableResources=0;
        Application.counter1=1;
        Application.sessions=0;
    </cfscript>
    <!--- You do not need to return True if you don't set the cffunction returntype attribute. --->
    </cffunction>

<cffunction name="onApplicationEnd">
    <cfargument name="ApplicationScope" required=true/>
    <cflog file="#This.Name#" type="Information"
        text="Application #ApplicationScope.applicationname# Ended">
</cffunction>


<cffunction name="onRequestStart">
    <!--- Authentication code, generated by the Dreamweaver Login Wizard,
        makes sure that a user is logged in, and if not displays a login page. --->
    <cfinclude template="mm_wizard_application_include.cfm">
<!--- If it's time for maintenance, tell users to come back later. --->
    <cfscript>
        if ((Hour(now()) gt 1) and (Hour(now()) lt 3)) {
            WriteOutput("The system is undergoing periodic maintenance.
                Please return after 3:00 AM Eastern time.");
            return false;
        } else {
            this.start=now();
        }
    </cfscript>
</cffunction>

<cffunction name="onRequest">
    <cfargument name = "targetPage" type="String" required=true/>
```

```
    <cfsavecontent variable="content">
        <cfinclude template=#Arguments.targetPage#>
    </cfsavecontent>
    <!--- This is a minimal example of an onRequest filter. --->
    <cfoutput>
        #replace(content, "report", "MyCompany Quarterly Report", "all")#
    </cfoutput>
</cffunction>

<!--- Display a different footer for logged in users than for guest users or
         users who have not logged in. --->

<cffunction name="onRequestEnd">
    <cfargument type="String" name = "targetTemplate" required=true/>
    <cfset theAuthuser=getauthuser()>
    <cfif ((theAuthUser EQ "guest") OR (theAuthUser EQ ""))>
        <cfinclude template="noauthuserfooter.cfm">
    <cfelse>
        <cfinclude template="authuserfooter.cfm">
    </cfif>
</cffunction>

<cffunction name="onSessionStart">
    <cfscript>
        Session.started = now();
        Session.shoppingCart = StructNew();
        Session.shoppingCart.items =0;
    </cfscript>
    <cflock timeout="5" throwontimeout="No" type="EXCLUSIVE" scope="SESSION">
        <cfset Application.sessions = Application.sessions + 1>
    </cflock>
    <cflog file="#This.Name#" type="Information" text="Session:
        #Session.sessionid# started">
</cffunction>

<cffunction name="onSessionEnd">
    <cfargument name = "SessionScope" required=true/>
    <cflog file="#This.Name#" type="Information" text="Session:
            #arguments.SessionScope.sessionid# ended">
</cffunction>

<cffunction name="onError">
    <cfargument name="Exception" required=true/>
    <cfargument type="String" name = "EventName" required=true/>
    <!--- Log all errors. --->
    <cflog file="#This.Name#" type="error" text="Event Name: #Eventname#">
    <cflog file="#This.Name#" type="error" text="Message: #exception.message#">
```

```
    <!--- Some exceptions, including server-side validation errors, do not
             generate a rootcause structure. --->
    <cfif isdefined("exception.rootcause")>
        <cflog file="#This.Name#" type="error"
            text="Root Cause Message: #exception.rootcause.message#">
    </cfif>
    <!--- Display an error message if there is a page context. --->
    <cfif NOT (Arguments.EventName IS onSessionEnd) OR
            (Arguments.EventName IS onApplicationEnd)>
        <cfoutput>
            <h2>An unexpected error occurred.</h2>
            <p>Please provide the following information to technical support:</p>
            <p>Error Event: #EventName#</p>
            <p>Error details:<br>
            <cfdump var=#exception#></p>
        </cfoutput>
    </cfif>
    </cffunction>

</cfcomponent>
```

"Application.cfm" から "Application.cfc" への移行

"Application.cfm" を使用している既存のアプリケーションを、"Application.cfc" を使用するように移行するには、次の作業を行います。

- cfapplication タグを、このタグの属性に対応する "Application.cfc" の This スコープの変数を設定する CFC 初期化コードで置き換えます。
- onApplicationStart メソッドに、Application スコープの変数を初期化する任意のコードと、アプリケーションの起動時にのみ実行する任意のアプリケーション固有コードを記述します。"Application.cfm" では、このようなコードは多くの場合、Application スコープの switch 変数が存在するかどうかをテストするブロックの中に記述されています。変数をテストするコードや、Application スコープの変数を設定するコードを囲んでいる Application スコープのロックを削除します。
- onSessionStart メソッドに、Session スコープの変数を初期化する任意のコードと、セッションの起動時にのみ実行する任意のアプリケーション固有コードを記述します。初期化対象の Session スコープ変数の存在をテストするコードや、Session スコープの変数を設定するコードを囲んでいる Session スコープのロックを削除します。
- onRequestStart メソッドに、cflogin タグおよび関連する認証コードを記述します。
- onRequest メソッドに Variables スコープの変数を設定する任意のコードを記述し、method.Targetpage 変数の Arguments で指定されているページをインクルードする cfinclude タグを追加します。
- onRequestEnd メソッドに、"OnRequestEnd.cfm" ページに含まれている任意のコードを記述します。
- cferror タグを onError イベントメソッドで置き換えるかどうか検討します。置き換えない場合は、その cferror タグを CFC 初期化コード内に記述します。

"Application.cfm" ページの使用

"Application.cfc" ファイルを使用しない場合は、"Application.cfm" ページを使用して、アプリケーションレベルの設定および関数を定義します。

アプリケーションのネーミング

cfapplication タグは、アプリケーション名を指定し、1 つの論理アプリケーションを構成するページ群を定義するために使用します。アプリケーション名を指定する cfapplication タグを各ページに記述することもできますが、通常はこのタグを "Application.cfm" ファイルに記述します。次に例を示します。

```
<cfapplication name="SearchApp">
```

注意：cfapplication タグの name 属性に設定する値は、最大 64 文字に制限されています。

クライアント変数、アプリケーション変数、およびセッション変数のオプションの設定

クライアントステートや永続変数を使用する場合は、cfapplication タグで次のように指定します。

- Client スコープ変数を使用するには、clientManagement=True を指定します。
- Session スコープ変数を使用するには、sessionManagment=True を指定します。

必要に応じて次のことも行えます。

- Application スコープ変数と Session スコープ変数に対してアプリケーション固有のタイムアウトを設定する。これらの設定は、ColdFusion Administrator で設定するデフォルト値よりも優先されます。
- Client スコープ変数の保管方法を指定する。この設定は、ColdFusion Administrator で設定する方法よりも優先されます。
- クライアントブラウザで Cookie を使用しないように指定する。

これらのオプションの設定については、300 ページの「永続データとロックの使用」および『CFML リファレンス』を参照してください。

ページ処理設定の定義

cfsetting タグを使用すると、アプリケーション内の全ページに適用するページ処理属性を指定できます。詳細については、239 ページの「ページ処理オプションの設定」を参照してください。

アプリケーションのデフォルト変数と定数の設定

デフォルト変数とアプリケーションレベルの定数は、"Application.cfm" ページで設定します。たとえば、次の値を指定します。

- データソース
- ドメイン名
- フォントやカラーなどのスタイル設定
- その他の重要なアプリケーションレベルの変数

通常、"Application.cfm" ページでは cfinclude タグを使用して、アプリケーションページで必要な共通コード (ユーザー定義関数など) のライブラリをインクルードします。

ログインの処理

アプリケーションでユーザーのログインが必要な場合、通常は cflogin タグを "Application.cfm" ページに記述します。ユーザーのログインを管理する "Application.cfm" ページなどの、セキュリティとログイン作成の詳細については、337 ページの「アプリケーションの保護」を参照してください。

エラー処理

"Application.cfm" ページで cferror タグを使用すると、次の例に示すように、リクエストエラー、検証エラー、または例外エラーを処理するためのアプリケーション固有のページを指定できます。この方法を使用すれば、エラーメッセージにアプリケーション固有の情報 (連絡先情報、アプリケーションやバージョンの識別子など) を含めることができ、また、アプリケーション内のすべてのエラーメッセージを一貫した方法で表示できます。

エラーページとエラー処理の詳細については、275 ページの「エラー処理」を参照してください。

例 : "Application.cfm" ページ

次の例では、"Application.cfm" ページでよく使用されるテクニックをいくつか用いた、サンプルの "Application.cfm" ファイルを示します。簡略化するために、ログイン処理は省略しています。ログインの例については、337 ページの「アプリケーションの保護」を参照してください。

```
 <!--- Set application name and enable Client and Session variables. --->
<cfapplication name="Products"
    clientmanagement="Yes"
    clientstorage="myCompany"
    sessionmanagement="Yes">

<!--- Set page processing attributes. --->
<cfsetting showDebugOutput="No">

<!--- Set custom global error handling pages for this application.--->
<cferror type="request"
    template="requesterr.cfm"
    mailto="admin@company.com">
<cferror type="validation"
    template="validationerr.cfm">

<!--- Set the Application variables if they aren't defined. --->
<!--- Initialize local app_is_initialized flag to false. --->
<cfset app_is_initialized = False>
<!--- Get a read-only lock. --->
<cflock scope="application" type="readonly" timeout=10>
<!--- Read init flag and store it in local variable. --->
    <cfset app_is_initialized = IsDefined("Application.initialized")>
</cflock>
<!--- Check the local flag. --->
<cfif not app_is_initialized>
<!--- Application variables are not initialized yet.
        Get an exclusive lock to write scope. --->
    <cflock scope="application" type="exclusive" timeout=10>
        <!--- Check the Application scope initialized flag since another request
            could have set the variables after this page released the read-only
            lock. --->
        <cfif not IsDefined("Application.initialized")>
            <!--- Do initializations --->
            <cfset Application.ReadOnlyData.Company = "MyCompany">
            <!--- and so on --->
            <!--- Set the Application scope initialization flag. --->
            <cfset Application.initialized = "yes">
```

```
        </cfif>
    </cflock>
</cfif>

<!--- Set a Session variable.--->
<cflock timeout="20" scope="Session" type="exclusive">
    <cfif not IsDefined("session.pagesHit")>
        <cfset session.pagesHit=1>
    <cfelse>
        <cfset session.pagesHit=session.pagesHit+1>
    </cfif>
</cflock>

<!--- Set Application-specific Variables scope variables. --->
<cfset mainpage = "default.cfm">
<cfset current_page = "#cgi.path_info#?#cgi.query_string#">

<!--- Include a file containing user-defined functions called throughout
        the application. --->
<cfinclude template="commonfiles/productudfs.cfm">
```

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfapplication name="Products" clientmanagement="Yes" clientstorage="myCompany" sessionmanagement="Yes">`	アプリケーションの名前を指定し、Client および Session スコープ変数を有効にし、クライアントの変数ストアを myCompany データソースに設定します。
`<cfsetting showDebugOutput="No">`	ColdFusion Administrator でデバッグが有効になっていても、デバッグ出力が表示されないようにします。
`<cferror type="request" template="requesterr.cfm" mailto="admin@company.com"> <cferror type="validation" template="validationerr.cfm">`	アプリケーションで発生するリクエストエラーと検証エラー用のカスタムエラーハンドラを指定します。リクエストエラーハンドラでは、メールアドレスを指定します。
`<cfset app_is_initialized = False>`	Application スコープ変数が設定されていない場合は設定します。Application スコープ変数の設定に使用されるテクニックの詳細については、300 ページの「永続データとロックの使用」を参照してください。
`<cflock timeout="20" scope="Session" type="exclusive"> <cfif not IsDefined("session.pagesHit")> <cfset session.pagesHit=1> <cfelse> <cfset session.pagesHit=session.pagesHit+1> </cfif> </cflock>`	このセッションで参照されたページ数をカウントする Session スコープの pagesHit 変数を設定します。変数が存在しない場合は作成し、存在する場合はインクリメントします。
`<cfset mainpage = "default.cfm"> <cfset current_page = "#cgi.path_info#?#cgi.query_string#">`	アプリケーション全体を通して使用される 2 つの Variables スコープ変数を設定します。current_page 変数はダイナミックに作成します。この値はリクエストごとに変わります。
`<cfinclude template="commonfiles/productudfs.cfm">`	アプリケーションの多くのページで使用されるユーザー定義関数のライブラリをインクルードします。

ColdFusion アプリケーションの最適化

ColdFusion アプリケーションの最適化にはさまざまな方法があります。ColdFusion の最適化作業の大半は、開発およびコーディングを適切に行うことと関係します。たとえば、データベースを適切に設計および使用することは、効率的な ColdFusion アプリケーションの重要な要件です。

本マニュアルでは、さまざまな ColdFusion トピックにおいて、そのトピックに関連する最適化テクニックを紹介しています。ここでは、ColdFusion の最適化ツールと方法 (特に最適化のために CFML キャッシュタグを使用する方法) の概要を説明します。また、アプリケーション最適化の重要なテーマの 1 つであるデータベース使用の最適化についても説明します。

ColdFusion Administrator には、ColdFusion ページおよび SQL クエリーのキャッシュオプションが用意されています。これらのオプションについては、ColdFusion Administrator のオンラインヘルプおよび『ColdFusion 設定と管理』を参照してください。

実行速度の遅いページを特定するのに役立つデバッグテクニックについては、379 ページの「アプリケーションのデバッグとトラブルシューティング」を参照してください。

ColdFusion の最適化に関する追加情報については、Adobe ColdFusion サポートセンター (www.adobe.com/go/learn_cfu_support_jp) を参照してください。

変更頻度の低い ColdFusion ページのキャッシュ

ColdFusion ページの中には、出力があまり変化しないものもあります。たとえば、データベースから業者のリストを抽出するアプリケーションや、四半期ごとの要約を生成するアプリケーションが考えられます。通常は、アプリケーションページに対するリクエストが ColdFusion で受信されると、レポートの作成やリストの生成と表示に必要なビジネスロジックや表示処理がすべて実行されます。したがって、処理結果があまり変化しないページの場合は、同じ処理にプロセッサリソースや帯域幅が使用されることになり、効率的ではありません。

cfcache タグを使用すると、ページリクエストの処理結果である HTML をサーバーの一時ファイルにキャッシュすることができます。これによって、ページがリクエストされるたびに HTML を生成せずに済むようになります。キャッシュ済みの ColdFusion ページに対するリクエストを受信した場合は、ColdFusion ページを処理せずに、生成済みの HTML ページを取得します。また、クライアントサイドでページをキャッシュすることもできます。以前に参照したページのキャッシュコピーがクライアントブラウザにある場合は、ページを再送信せずに、クライアントにあるページを使用するようにブラウザに指示します。

注意：cfcache タグのキャッシュメカニズムでは、異なる URL は異なるページと見なされます。したがって、http://www.mySite.com/view.cfm?id=1 と http://www.mySite.com/view.cfm?id=2 は 2 つの独立したキャッシュページになります。URL のパラメータセットごとに独立してページがキャッシュされるので、パラメータに応じて出力が変化するページにも対応できます。

cfcache タグの使用

ページ結果をキャッシュするには、ColdFusion ページのテキスト出力コードよりも前に cfcache タグを記述します。このタグを使用すると、次の情報を指定できます。

- サーバー、クライアントシステム、またはその両方にページ結果をキャッシュするかどうか。デフォルトでは両方にキャッシュします。このデフォルトは、すべてのユーザーに共通のページに対して最適です。ページにクライアント固有の情報が含まれる場合や、ColdFusion のユーザーセキュリティでページを保護する場合は、cfcache タグの action 属性を ClientCache に設定します。
- キャッシュページを保存するサーバーのディレクトリ。デフォルトのディレクトリは "<ColdFusion のルートディレクトリ >/cache" です。アプリケーションごとに個別のキャッシュディレクトリを作成することをお勧めします。それによって、cfcache タグの flush アクションで、誤って複数のアプリケーションのキャッシュが一度に消去されるのを防ぐことができます。
- ページをキャッシュに保管してから自動的に消去するまでの期間。

また、Web サーバー上のキャッシュページにアクセスするための属性 (Web サーバーで必要なユーザー名とパスワード、ポート、プロトコルが HTTP であるか HTTPS であるかなど) も指定できます。

cfcache タグは、ページ内の出力生成コードよりも前 (通常はページ本文の冒頭) に記述します。たとえば、次のタグでは、クライアントとサーバーの両方にページがキャッシュされます。サーバーでは、"e:/temp/page_cache" ディレクトリにページがキャッシュされます。キャッシュページの保持期間は 1 日です。

```
<cfcache timespan="#CreateTimespan(1, 0, 0, 0)#" directory="e:/temp/page_cache">
```

重要："Application.cfm" または "Application.cfc" ページでテキストを表示する場合 (ヘッダページをインクルードする場合など) は、キャッシュするページだけでなく "Application.cfm" ページでも cfcache タグを使用してください。これを行わないと、キャッシュされた各ページに **"Application.cfm"** ページの出力が **2** 回表示されます。

キャッシュされたページの消去

ページのコードを変更すると、そのページのキャッシュは自動的に消去されます。期限切れのページも自動的に消去されます。

cfcache タグで action="flush" 属性を使用すると、キャッシュされたページをすぐに消去できます。必要に応じて、消去するキャッシュページが含まれているディレクトリや、消去するページを識別する URL パターンを指定することもできます。URL パターンを指定しない場合は、ディレクトリ内のすべてのページが消去されます。この URL パターンにアスタリスク (*) ワイルドカードを含めると、URL の可変部分を指定できます。

cfcache タグを使用してページのキャッシュを消去すると、サーバー上のキャッシュが削除されます。その消去したページがクライアントシステム上でもキャッシュされている場合、その ColdFusion ページにクライアントが次回アクセスするときにクライアントのキャッシュが削除され、新しいコピーがキャッシュされます。

次の例では、"e:/temp/page_cache/monthly" ディレクトリにある HR で始まるページのキャッシュをすべて消去します。

```
<cfcache action="flush" directory="e:/temp/page_cache/monthly" expirURL="HR*">
```

キャッシュページで使用されているデータを更新するための ColdFusion ページがある場合は、更新を実行するページに cfcache タグを含めて、該当データを使用するすべてのページを消去します。

cfcache タグの詳細については、『CFML リファレンス』を参照してください。

ColdFusion ページの部分キャッシュ

場合によっては、同じページの中に、ページを表示するたびに生成する必要があるダイナミック情報と、変更頻度の低いダイナミック情報が混在していることがあります。ColdFusion 9 では、キャッシュを制御する機能がさらに強化されています。ページの部分 (フラグメント) をキャッシュすると、変更頻度の低いコンテンツをキャッシュ内に格納できます。次に、フラグメントキャッシングの例を示します。

```
<!--- Greet the us er. --->
<cf output>
    W elcome to our home page.<br>
    The time is #TimeFormat(Now())#.<br>
    Your lucky number is: #RandRa nge(1,1000)#<br>
    <hr>
</cfoutput>
<!--- If the flag is false, query the DB, and save an image of
    the results output to a variable. ---> <cfcache action="optimal" timespan="#createtimespan(0,1,0,0)#"
idletime="#createtimespan(0,0,30,0)#"><!--- Perform database query. ---><cfquery dataSource="cfartgallery"
name="specialQuery">    SELECT * from art
</cfquery>
<!--- Calculate sale price and display the results. --->
        <h2>Check out the following specials</h2>
        <table>
<cfoutput query="specialQuery">
            <tr>
                <td>#artid#</td>
                <td>#Description#</td>
                <td>#price#</td>
       </tr>
</cfoutput>
        </table></cfcache><hr><p>Thank you for visiting us!</p>
```

この例では、太字で示されているコードにより、テーブル art に含まれるすべてのアートレコードを表示するページフラグメントが作成されます。キャッシュは 1 時間後 (またはアイドル状態が 30 分間続いた後) に無効になるように設定されています。キャッシュが無効になった後に、このページへの最初のアクセスが発生するとキャッシュが再作成され、更新された情報があればその情報が表示されます。

ColdFusion 10 でのキャッシュ機能の強化

アプリケーション別のキャッシュ処理

キャッシュ処理をサーバーレベルまたはアプリケーション別に指定することができます。

キャッシュ設定をアプリケーションレベルに設定するには、次の例に示すように、Application.cfc でそのアプリケーション専用のキャッシュ設定ファイル（ehcache.xml）を指定します。

絶対パスを指定する場合：

```
this.cache.configfile = "c:/cachesettings/ehcache.xml";
```

Application.cfc を基準にした相対パスを指定する場合：

```
this.cache.configfile = "ehcache.xml";
```

キャッシュ処理をアプリケーションレベルに指定した場合、すべてのキャッシュ関数では、そのアプリケーション専用のキャッシュ設定が使用されます。

例

Application.cfc

```
<cfcomponent>
    <cfscript>
        this.name = "appSpecificCacheTest";
        this.cache.configfile = "ehcache.xml";
       this.applicationTimeout = createtimespan(0,0,0,5);

        function onApplicationStart(){
            writelog("In onApplicationStart()");
        }
        function onApplicationEnd(){
            writelog("In onApplicationEnd()");
        }
    </cfscript>
</cfcomponent>
```

cache.cfm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>cfhttp</title>
</head>
<!--- End of header --->
<body>
        <!--- remove object from Application Specific cache --->
        <cfif ArrayLen(cacheGetAllIds()) gt 0>
            <cfset cacheRemove(ArrayToList(cacheGetAllIds()))>
        </cfif>

        <cfset obj1 = structNew()>
        <cfset obj1.name = "xyz">
        <cfoutput>Starting to write to cache..</cfoutput>
        <cfset cachePut("obj1",obj1)>
        <br/>
        <cfoutput>Done!!</cfoutput>

        <cfoutput>Trying to fetch cached item...</cfoutput>
        <cfset obj = cacheGet("obj1")>
        <br/>
        <cfdump var="#obj#">


        <cfscript>
           sleep(15000);
        </cfscript>

        <cfoutput>Trying to fetch cached item after 15 seconds...<br/></cfoutput>
        <cfset obj = cacheGet("obj1")>
        <cfdump var="#obj#">
</body>
</html>
```

ehcache.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="ehcache.xsd">

    <diskStore path="java.io.tmpdir"/>

    <cacheManagerEventListenerFactory class="" properties=""/>
    <defaultCache
     maxElementsInMemory="5"
     eternal="false"
     timeToIdleSeconds="30"
     timeToLiveSeconds="5"
     overflowToDisk="true"
     diskSpoolBufferSizeMB="30"
     maxElementsOnDisk="10"
     diskPersistent="false"
     diskExpiryThreadIntervalSeconds="3600"
     memoryStoreEvictionPolicy="LRU"
     clearOnFlush="true"
    />


    <cache name="app1cache"
     maxElementsInMemory="5"
     eternal="false"
     timeToIdleSeconds="60"
     timeToLiveSeconds="5"
     overflowToDisk="false"
     diskSpoolBufferSizeMB="30"
     maxElementsOnDisk="10"
     diskPersistent="false"
     diskExpiryThreadIntervalSeconds="3600"
     memoryStoreEvictionPolicy="LRU"
     clearOnFlush="true"/>

</ehcache>
```

Ehcache の使用によるクエリのキャッシュ処理の強化

Ehcache の使用

すべてのキャッシュクエリは、Ehcache を使用してキャッシュされます。

デフォルトのクエリキャッシュ領域とカスタムクエリキャッシュ領域

すべてのキャッシュクエリは、デフォルトのクエリ領域に保存されます。サーバー固有のキャッシュとアプリケーション固有のキャッシュには、それぞれ個別のデフォルトのクエリ領域が存在します。

cfquery で cacheRegion 属性を使用すると、ユーザー定義のクエリキャッシュ領域を指定できます。

クエリキャッシュ領域を指定しない場合は、アプリケーションレベルまたはサーバーレベルの（どちらが適用される場合であっても）デフォルトのキャッシュ領域が使用されます。

キャッシュ ID

cacheID 属性を使用して、各キャッシュの一意の識別子にクエリを関連付けることができます。キャッシュ関数を使用する際に、指定する ID を引用符で囲むことができます。

キャッシュ内では、クエリは、（object キャッシュや template キャッシュのように）query の type で保存されます。

内部キャッシュへのフォールバック

（Ehcache を使用せずに）内部キャッシュにフォールバックしてクエリをキャッシュするには、次のいずれかを行います。

- **サーバーレベル：**次の設定を変更します。

  「内部キャッシュを使用してクエリを保存する」オプションにチェックを付けます。

- **アプリケーションレベル：**次の設定に true を指定します。

  this.cache.useinternalquerycache=true|false。デフォルト値は false です。

クエリ制限の指定

キャッシュ内の最大クエリ数の制限を指定するには、Application.cfc で次の設定にクエリの数を指定します。

this.cache.querysize

Amazon S3 ストレージの使用

ColdFusion ユーザーは、Amazon S3 にデータを保存できるようになりました。このサポートは、ファイルやディレクトリを入力または出力として使用するタグと関数の大部分に渡ります。

Amazon S3 にファイルを保存するには、ディスクにファイルを保存するときと同じように実行します。ファイルが Amazon S3 に存在することを示すには、s3:// 接頭辞を使用します。例 : s3://testbucket/sample.txt

Amazon S3

Amazon S3 を使用するには、Amazon の S3 アカウントを持っている必要があります。

Amazon S3 に関する概念や詳細については、AmazonS3 ドキュメントを参照してください。

Amazon S3 へのアクセス

次のいずれかの URL 形式を使用して、ColdFusion から Amazon S3 にアクセスします。

- s3://bucket/x/y/sample.txt

  ここでの bucket はバケットの名前で、残りの URL 部分は Amazon S3 オブジェクトのキー名です。

  この場合は、Application.cfc で次の認証情報を指定します。

  ```
  <cfscript>
          this.name ="Object Operations";
          this.s3.accessKeyId = "key_ID";
          this.s3.awsSecretKey = "secret_key";
          this.s3.defaultLocation="location";
          </cfscript>
  ```

  次に例を示します。

  <cffile action="write" output="S3 Specification" file="s3://testbucket/sample.txt"/>

- s3://accessKeyId:awsSecretKey@bucket/x/y/sample.txt

  - この形式には、accessKeyId と awsSecretKey を含めて指定します。
  - @ は、認証情報の終わりを示すしるしの役目を果たします。

  注意：URL と Application.cfc の両方で accessKeyId と awsSecretKey を指定した場合は、URL で指定した値が優先されます。

例

```
<cffile action="write" output="S3 Specifications"
file="s3://accessKeyID:awsSecretKey@bucket/x/y/sample.txt"/>
```

Amazon S3 統合の強化

Amazon S3 にファイルをアップロードするときのパフォーマンスの向上に加えて、ColdFusion 10 では、ファイルを複数のパーツに分割して、それらのパーツを並行してアップロードするマルチパートアップロードをサポートしています。

マルチパートアップロードのサポートを設定するには、Application.cfc で次の設定を指定します。

```
this.s3.minsizeformultipart=filesize _in_MB
```

指定したサイズは最小サイズとして扱われ、このサイズを超えるファイルのアップロードは、マルチパートアップロードとして処理されます。このオプションは、サイズが非常に大きいファイルをアップロードする必要がある場合に役立ちます。

サポートされる操作

Amazon S3 でサポートされる操作は、次のとおりです。

- バケットの作成および削除
- バケットの場所の取得
- オブジェクトの読み込み、書き込み、コピー、および削除
- バケットのキーのリスト表示
- オブジェクトまたはバケットのメタデータの取得および設定
- オブジェクトまたはバケットの ACL の取得および設定

バケットの操作

バケットの操作 (作成、削除、またはリスト表示) を行うには、cfdirectory タグまたは directory 関数を使用します。

操作	使用するタグ	関数	例
作成	cfdirectory action="create" directory 属性では、バケットのパスのみを指定できます。その他のパスが含まれていると、エラーが発生します。 その他の属性はすべて無視されます。S3 バケットを作成すると、デフォルトのバケットの場所は US になります。この場所は、storeLocation 属性を使用して変更することができます。 storeLocation は、cfdirectory タグに追加された新しい属性です。 バケットに ACL を指定できます。ACL を作成するには、storeACL 属性を使用します。この属性は、構造体の値を持ちます。詳細については、260 ページの「アクセス制御の設定」を参照してください。	DirectoryCreate	`<cfdirectory action="create" directory="s3://bucket1"/>`
キーのリスト表示	cfdirectory action="list"	DirectoryList	`<cfdirectory action="list" directory="s3://bucket1/X/y" />` Amazon S3 にはディレクトリの概念がないので、バケットに含まれるオブジェクトのキー名 (つまりフルパス) が返されます。 この場合の directory 属性には、検索対象のオブジェクトが含まれるパス (たとえば s3://bucket1) を指定します。バケット名に続くパスは、リスト表示の操作を実行するための接頭辞として使用され、この接頭辞に一致するすべてのオブジェクトが返されます。 この場合、recurse、type、および sort 属性は無視されます。
削除	cfdirectory action="delete"	DirectoryDelete	`<cfdirectory action="delete" directory="s3://bucket1"/>`

注意：バケットが存在していてアクセス可能であるかどうかを確認するには、directoryExists 関数を使用します。

オブジェクトの操作

すべてのオブジェクトの操作は、ファイルの操作 (読み込み、書き込み、コピー、および削除) に似ています。そのため、cffile タグとファイル関数を使用して操作を実行できます。次の表に、一般的なシナリオを示します。

操作	使用するタグ	関数	例
読み込み	cffile action="read"	FileRead	`<cffile action="read" file="s3://testbucket/test.txt" variable="data"/>`
書き込み	cffile action="write"	FileWrite	`<cffile action="write" output="#data#" file="s3://testbucket/test.txt"/>`
削除	cffile action="delete"	FileDelete	`<cffile action="delete" file="s3://testbucket/test.txt"/>`
コピー	cffile action="copy"	FileCopy	`<cffile action="copy" source="s3://testbucket/test.txt" destination="s3://bucket2/a/b.txt"/>`

サポートされる関数は次のとおりです。

- FileIsEOF
- FileReadBinary

- Filecopy
- FileReadLine
- FileExists
- FileWriteln
- FileOpen
- FileClose
- FileRead
- FileDelete

cfdirectory action="create" タグの新しい属性

追加された属性	説明	例
storeLocation	作成したバケットの場所を変更する場合に使用します。場所は、EU または US のいずれかです。デフォルトの場所は US です。	`<cfdirectory action="create" directory="s3://<bucketname>" storelocation="US"> <cfdirectory action="create" directory="s3://<bucketname>" storelocation="EU">`
storeACL	構造体の配列で、各構造体は Permission または Grant (260 ページの「ACLObject」を参照) を表します。	`<cfdirectory action="create" directory="s3://<bucketname>" storeACL="ACLObject">`

アクセス制御の設定

Amazon S3 では、バケットおよびオブジェクトにアクセス制御リスト (ACL) を設定できます。バケットとオブジェクトの ACL は独立しています。これらは別々に管理する必要があります。また、オブジェクトの ACL は、バケットの ACL から継承されません。

ACL は複数の Grant で構成され、各 Grant には Grantee と Permission が含まれます。S3 では、次の 3 つのタイプの Grantee を使用できます。

- group
- email
- canonical (ID)

使用可能な権限 (Permission) は、次のとおりです。

- read
- write
- read_acp
- write_acp
- full_control

詳細については、Amazon S3 ACL Documentation を参照してください。

ACLObject

ACLObject は構造体の配列で、各構造体は ACL の Grant を表します。Grantee の詳細は、次のとおりです。

group Group キー (値は all、authenticated または log_delivery のいずれか) と permission を指定する必要があります。

email email キーと permission を指定する必要があります。

canonical Id キーと permission を指定する必要があります。displayName はオプションです。

ACLObject のサンプル

```
all_read = {group="all", permission="read"};
   owner_full = {email="xxx@yyy.com", permission="full_control"};
   aclObj = [owner_full, all_read];
```

アクセス制御関数

storeSetACL

説明

オブジェクトまたはバケットに ACL を設定します。

戻り値

なし

シンタックス

StoreSetACL(url, ACLObject)

パラメータ

パラメータ	説明
url	Amazon S3 の URL (コンテンツまたはオブジェクト) です。
ACLObject	構造体の配列で、各構造体は Permission または Grant (260 ページの「ACLObject」を参照) を表します。

履歴

ColdFusion 9 アップデート 1: この関数が追加されました。

使用方法

すべての権限を設定する場合に、この関数を使用します。この関数は、既存のすべての権限を上書きします。現在の状況で設定したもののみが存在します。

例

```
<cftry>
    <cfset dir = "s3://bucket_name">
    <cfif !directoryExists(dir)>
        <cfset directorycreate(dir)>
    </cfif>

    <cfset perm = structnew()>
    <cfset perm.group = "all">
    <cfset perm.permission = "read">
    <cfset perm1 = structnew()>
    <cfset perm1.email = "email ID">
    <cfset perm1.permission = "FULL_CONTROL">
    <cfset myarrray = arrayNew(1)>
    <cfset myarrray = [perm,perm1]>
    <cfset fileWrite("#dir#/test.txt","This is to test all users permission")>

    <cfset StoreSetACL("#dir#/textl.txt","#myarrray#")>
    <cfset test = StoreGetACL ("#dirkey#/test.txt") >

    <cfdump var= "test">

    <cfcatch>
        <cfdump var="#cfcatch#">
    </cfcatch>
</cftry>
```

storeAddACL

説明

オブジェクトまたはバケットの既存の ACL に、ACL を追加します。

戻り値

なし

シンタックス

StoreAddACL(url, ACLObject)

パラメータ

パラメータ	説明
url	Amazon S3 の URL (コンテンツまたはオブジェクト) です。
ACLObject	構造体の配列で、各構造体は Permission または Grant (260 ページの「ACLObject」を参照) を表します。

履歴

ColdFusion 9 アップデート 1: この関数が追加されました。

使用方法

既存の権限に追加を行う場合に、この関数を使用します。

例

```
<cftry>
    <cfset dir = "s3://bucket_name/">
    <cfset perm = structnew()>
    <cfset perm.group = "authenticated">
    <cfset perm.permission = "READ">
    <cfset perm1 = structnew()>
    <cfset perm1.email = "email_ID">
    <cfset perm1.permission = "READ_ACP">
    <cfset myarrray = [perm,perm1]>
    <cfif NOT DirectoryExists(dir)>
        <cfset directoryCreate(dir)>
    </cfif>
    <cfset fileWrite("#dir#/Sample.txt","This is to test StoreAddACL")>
    <cfset StoreAddACL("#dir#","#myarrray#")>
    <cfset test = StoreGetACL(dirkey)>
        <cfdump var="#test#">
    <cfcatch>
        <cfdump var="#cfcatch#">
    </cfcatch>
</cftry>
```

storeGetACL

説明

オブジェクトまたはバケットの ACL を取得します。

戻り値

ACLObject を返します。

シンタックス

StoreGetACL(url, ACLObject)

パラメータ

パラメータ	説明
url	Amazon S3 の URL (コンテンツまたはオブジェクト) です。
ACLObject	構造体の配列で、各構造体は Permission または Grant (260 ページの「ACLObject」を参照) を表します。

履歴

ColdFusion 9 アップデート 1: この関数が追加されました。

例

```
<cfset dir = "s3://bucket_Name">
    <cfif NOT DirectoryExists(dir)>
         <cfset directoryCreate(dir)>
        </cfif>
    <cfset test = StoreGetACL("#dir#")>
    <cfdump var="#test#">
```

メタデータの使用

Amazon S3 では、オブジェクトとバケットの両方のメタデータを指定できます。

次の 2 つの関数を使用して、オブジェクトまたはバケットのメタデータを取得および設定できます。

StoreGetMetadata

説明
オブジェクトまたはバケットに関連するメタデータを返します。

戻り値
オブジェクトのメタデータまたはバケットのメタデータ

シンタックス
StoreGetMetadata(url)

パラメータ

パラメータ	説明
url	Amazon S3 の URL (バケットまたはオブジェクト) です。

履歴
ColdFusion 9 アップデート 1: この関数が追加されました。

例

```
<cfdump var = #StoreGetMetadata("bucket_Name")#>
```

StoreSetMetadata

説明
バケットまたはオブジェクトのメタデータを設定します。

戻り値
なし

シンタックス
StoreSetMetadata(url,Struct)

パラメータ

パラメータ	説明
url	Amazon S3 の URL (バケットまたはオブジェクト) です。
struct	メタデータを表します。メタデータの標準キーの一覧については、265 ページの「標準キー」を参照してください。 また、標準のメタデータとは別にカスタムメタデータを設定することもできます。

履歴
ColdFusion 9 アップデート 1: この関数が追加されました。

例

```
<cfscript>
      mydate = #Now()#;
      hello = structNew();
      hello.color = "grey";
/cfscript>

<cfset dir = "s3://mycfbucket">
      <cffile action="write" file="#dir#/hello5.txt" output="Sample s3 text">

      <cfset StoreSetMetadata("#dir#/hello5.txt","#hello#")>
      <cfset test = StoreGetMetadata("#dir#/hello5.txt")>

<cfdump var="#test#">
```

標準キー

メタデータの標準キーは、次のとおりです。

オブジェクト用

- last_modified
- date
- owner
- etag
- content_length
- content_type
- content_encoding
- content_disposition
- content_language
- content_md5
- md5_hash

バケット用

- date
- owner

セキュリティに関する注意事項

S3 のバケットまたはオブジェクトには、サンドボックス機能を適用できません。これは、Amazon S3 では、独自のセキュリティ機能でサンドボックス機能を処理しているからです。

サポートされる関数

fileOpen	fileClose	fileCopy	fileDelete
fileExists	fileisEOF	fileMove	fileWrite
fileRead	fileReadBinary	fileReadLine	fileSetLastModified
getFileInfo	getDirectoryFromPath	directoryCreate	directoryDelete
directoryExists	directoryList	imageNew	imageRead
imageWrite	imageWriteBase64	isImageFile	isPDFFile

サポートされるタグ

すべての cffile action	すべての cfdirectory action (rename 以外)	cfdocument	cffeed
cfftp	cfimage	cfloop	すべての cfimage action

ColdFusion 10 での機能強化

Amazon S3 にファイルをアップロードするときのパフォーマンスの向上に加えて、ColdFusion 10 では、ファイルを複数のパーツに分割して、それらのパーツを並行してアップロードするマルチパートアップロードをサポートしています。

マルチパートアップロードのサポートを設定するには、Application.cfc で次の設定を指定します。

this.s3.minsizeformultipart=filesize _in_MB

指定したサイズは最小サイズとして扱われ、このサイズを超えるファイルのアップロードは、マルチパートアップロードとして処理されます。このオプションは、サイズが非常に大きいファイルをアップロードする必要がある場合に役立ちます。

制約

- 次のタグはサポートされません。
  - cfpdf
  - cfpdfform
- 次の関数はサポートされません。
  - Linux または Unix でファイルの属性を設定する FileSetAccessMode
  - Windows でファイルの属性を設定する FilesSetAttribute
  - cfzip で、Amazon S3 オブジェクトはソースに指定できません。
  - cfexecute タグの outputfile 属性で出力先に S3 オブジェクトを使用すると、[<exe> が完了する前に、タイムアウトになりました。] のエラーが発生します。また、サーバーコンソールで NullPointerException が発生します。
  - fileMove 関数を使用する場合は、ソースオブジェクトと移動先オブジェクトが同じバケット名である必要があります。つまり、Amazon S3 オブジェクトを別のバケットや他のファイルシステムに移動することはできません。

メモリ内ファイルの使用

メモリベースの仮想ファイルシステムを使用すると、一時的なデータの処理速度が向上します。メモリ内ファイルは、ディスクに書き込まれるのではなく、RAM に保存されます。メモリ内ファイルは、ディスク上のファイルと同様に扱われますが、より高速に処理できます。

ColdFusion でメモリ内ファイルを使用すると、ダイナミックコードの実行が容易になります。メモリ内ファイルは、ファイルやディレクトリを入力または出力として使用するタグと関数の大部分でサポートされています。

メモリ内ファイルは、ディスク上のファイルと同様に使用できますが、RAM 上に存在していることを示すために、接頭辞 ram:/// を付ける必要があります。たとえば、ram:///a/b/dynamic.cfm のように指定します。

ダイナミック CFM ファイルの書き込みおよび実行

次のシンタックスは、メモリ内ファイルに CFM データを書き込む方法を示しています。

```
<cffile action="write" output="#cfml#"
file="ram:///filename.cfm"/>
```

次のシンタックスの例は、メモリ内 CFM ファイルの使用方法を示しています。

- 論理パスを取るタグの場合は、Administrator でマッピングを定義します。cfinclude タグを使用して、メモリ内 CFM ページを実行します。

```
<cfinclude template="/inmemory/filename.cfm">
```

タグで使用できるように ram:/// のマッピングを作成します。この例の場合は、/inmemory が ram:/// を指すマッピングです。

- 絶対パスを取るタグの場合は、次のようなシンタックスを使用します。

```
<cffile action="append" file="ram:///a/b/dynamic.cfm" output="I'm appending">
```

注意：Application.cfm をメモリ内ファイルとして指定することはできません。これを指定した場合は、無視されます。

例

次のコードは、イメージをメモリ内ファイルとして書き込む方法を示しています。

```
<cffile action="readBinary" variable="myImage" file="#ExpandPath('./')#/blue.jpg">
<cffile action="write" output="#myImage#" file="ram:///a.jpg">
<cfif FileExists("ram:///a.jpg")>
<cfoutput>a.jpg exists</cfoutput>
<cfelse>
<cfoutput>a.jpg Doesn't exists</cfoutput>
</cfif>
```

ダイナミック CFC ファイルの書き込みおよびインスタンス化

次のシンタックスは、メモリ内ファイルに CFC コードを書き込む方法を示しています。

```
<cffile action="write" output="#cfcData#"
file="ram:///filename.cfc"/>
```

次のシンタックスは、メモリ内の CFC ファイルをインスタンス化する方法を示しています。

```
<cfset cfc=CreateObject("component","inmemory.filename")/>
```

この例では、inmemory が ram:/// を指す ColdFusion マッピングです。

注意：Application.cfc をメモリ内ファイルとして指定することはできません。これを指定した場合は、無視されます。

例

次のコードを実行すると、CFC がメモリ内ファイルとして書き込まれます。

```
<cffile action="read" file="#ExpandPath('./')#/dynamic.cfc" variable="Message">
<cffile action="write" file="ram:///cfc/dynamic.cfc" output="#Message#">
```

コンポーネントメソッドを呼び出すには :

```
<cfinvoke component="inmemory.cfc.dynamic" method ="method1" returnVariable="returnVariable">
<cfinvokeargument name="paramOne" value="hello">
</cfinvoke>
<cfoutput>#returnVariable#</cfoutput>
```

メモリ内ファイルシステムの使用

ここには、メモリ内ファイルにアクセスして使用する際に役立つ情報が記載されています。

メモリ内ファイルの使用

- RAM 内に保存された CFC は、ディスク上の CFM ファイルから呼び出せます。同様に、メモリ内の CFM ファイルからは、ディスク上に保存された CFC を呼び出せます。

- ある CFC が RAM 内の同じディレクトリにある別の CFC を拡張する場合は、相対パスを使用できます。たとえば、a.cfc と b.cfc が RAM 内の同じディレクトリに存在する場合、a.cfc では、次のコードに示す相対パスを使用して b.cfc を拡張できます。

```
<cfcomponent name="a" extends"b">
…
</cfcomponent>
```

- メモリ内の ColdFusion インターフェイスは、メモリ内の CFC と同じ方法で使用できます。

サポートされる関数

メモリ内ファイルに対しては、次のファイル関数を使用できます。

- FileIsEOF
- FileReadBinary
- Filemove
- Filecopy
- FileReadLine
- FileExists
- FileOpen
- FileWriteln
- FileClose
- FileRead
- FileDelete
- DirectoryExists
- FileSetLastModified
- GetFileInfo
- GetDirectoryFromPath
- GetFileFromPath
- ImageNew
- ImageRead
- ImageWrite
- ImageWriteBase64
- IsImageFile
- IsPDFFile
- FileSetLastModified

例

次のシンタックスは、FileSetLastModified() 関数の使用方法を示しています。

```
<cftry>
<cffile action="write" file="ram:///a.txt" output="Testing the function FileSetLastModified">
<cfset date="12/12/2007">
<cfscript>
FileSetLastModified("ram:///a.txt", "#date#");
sleep(1000);
WriteOutput(#GetFileInfo("ram:///a.txt").lastmodified#);
</cfscript>
<cfcatch>
<cfset PrintException(cfcatch)>
</cfcatch>
</cftry>
<cf_expectedresults>{ts '2007-12-12 00:00:00'}
</cf_expectedresults>
```

ファイル操作

メモリ内ファイルに対しては、次のファイル操作を実行できます。

- ディレクトリ別の操作 : 作成、削除、一覧表示、名前変更。
- ファイル別の操作 : コピー、作成、書き込み、追加書き込み、削除、名前変更、属性の作成、モード変更、移動、読み取り。

例

次のコードは、ファイルとディレクトリを操作する方法を示しています。

```
<cfdirectory action = "create" directory = "ram://src" >
<cfdirectory action = "create" directory = "ram://des" >
<cfdirectory action = "rename" directory = "ram:///CurrentDir" newDirectory = "NewDir">
<cfdirectory action="list" directory="ram://" name="listDir" recurse="yes" >
<cfdump var="#listDir#">
<cffile action="write" file = "ram://src/test.txt" output = "Release Description">
<cffile action="copy" source="ram://src/test.txt" destination="ram://des/final.txt" >
<cffile action="rename" source = "ram:///src/message.txt" destination = "ram:///des/test.txt">
<cffile action ="move" source = "ram:///des/test.txt" destination = "c:\des\move.txt">
```

ドキュメント処理とイメージ処理

次の例に示すように、メモリ内イメージファイルは、すべてのイメージ処理およびドキュメント処理で使用できます。

```
<cfimage action="captcha" fontSize="15" width="180" height="50" text="readMe"
destination="ram:///readMe.jpg"
difficulty="medium">
<cfimage source="ram://aiden02.png" action="convert" destination="#ExpandPath("./")#/blue1.JPG"
overwrite="yes">
<cfdocument format="pdf" filename="ram://Sample.pdf" overwrite="yes">Sample Text</cfdocument>
```

カスタムタグ

メモリ内の CFM ページと CFC からはカスタムタグを呼び出せますが、カスタムタグ自体はディスク上に存在している必要があります。メモリ内のカスタムタグはサポートされていません。

タグでのメモリ内ファイルの使用

メモリ内ファイルは、次のタグで指定できます。

- cfcontent

- cfdocument
- cfdump
- cfexchange
- cfexecute
- cffeed
- cfhttp
- cfftp
- cfimage
- cfloop
- cfpresentation
- cfprint
- cfreport
- cfzip

cfcontent タグの使用例

```
<cfcontent file="ram:///a.jpg" type="image/jpeg" deletefile="yes">
```

権限の追加

ColdFusion では、既存のサンドボックスセキュリティ設定を使用して、RAM 上のディレクトリやファイルにアクセス権限を追加できます。サンドボックスセキュリティは、ディスク上のディレクトリに対してのみ設定でき、RAM 上のディレクトリに対しては設定できません。メモリ内のディレクトリとファイルへのアクセスは、既存の論理ファイルシステムのサンドボックスを通じてのみ制限できます。

したがって、メモリ内ファイルに対してサンドボックスセキュリティを設定するには、ディスク上のディレクトリに対して既に指定したサンドボックスを選択します。

ram:/// ディレクトリは、デフォルトで、保護対象フォルダのリストに追加され、読み取り、書き込み、実行、および削除の権限が設定されます。したがって、すべての RAM ファイルには、サンドボックスで指定された権限がデフォルトで適用されます。ディスク上のファイルに適用されるセキュリティ制限は、メモリ内ファイルにもすべて適用されます。

メモリ内ファイルに対してサンドボックスセキュリティを設定するには :

1. ColdFusion Administrator の [セキュリティ]-[サンドボックスセキュリティ] ページを開きます。

   [サンドボックスセキュリティ許可] ページが表示されます。

2. [セキュリティサンドボックスの追加] ボックスで、ディスク上のディレクトリを指定して [追加] をクリックします。
3. [ファイル / ディレクトリ] で、メモリ内ファイルのパスを指定します。たとえば、ディレクトリの場合は ram:///a/b のように、ファイルの場合は ram:///a/b/dynamic.cfm のように指定します。
4. 必要な権限を選択し、[ファイル / パスの追加] をクリックして [終了] をクリックします。

サンドボックスセキュリティの詳細については、『ColdFusion 設定と管理』を参照してください。

VFS 情報へのアクセス

GetVFSMetaData 関数を使用すると、VFS 情報にアクセスできます。この関数は、入力としてパラメータ RAM を取ります。

この関数は、次の情報を含む構造体を返します。

- メモリ内仮想ファイルシステムのサポートが有効になっているかどうか
- メモリ内仮想ファイルシステムに割り当てられるメモリ容量の上限 (バイト単位)
- 使用メモリと空きメモリ

たとえば、<cfdump var="#getVFSMetaData("ram")#"> のように使用します。

注意：ColdFusion Administrator の [設定] ページには、メモリ内仮想ファイルシステムを有効化または無効化するオプションがあります。メモリ内仮想ファイルシステムのメモリ制限をメガバイト (MB) 単位で指定することもできます。

メモリ内ファイルの削除

メモリ内ファイルは、サーバーが稼動している限り RAM 内に残ります。これらのファイルを消去するには、cffile/cfdirectory で action=delete を指定します。たとえば、RAM ディレクトリ "ram://a/b" に存在する内容をすべて削除する場合は、<cfdirectory action="delete" directory="ram:///a/b" recurse="yes"> というコードを使用します。

制約

- RAM 上のファイルとディレクトリの名前では、大文字と小文字が区別されます。
- メモリ内ファイルにアクセスするときは、マッピングまたは絶対パスを使用する必要があります。ファイルおよびディレクトリへの相対パスは、サポートされません。
  - 正しい例 : ram:///a/b/
  - 誤った例 : ram:///a/b/../..
- HTTP または HTTPS プロトコルを使用してメモリ内ファイルにアクセスすることはできません。代わりに、ram:///< ファイル > を使用します。たとえば、ram:///a/b/test.cfm のように指定します。
- DDX ファイルとフォントファイルは、メモリ内ファイルとしてアクセスできません。
- メモリ内ファイルに対して次の関数を使用することはできません。
  - Linux または Unix でファイルの属性を設定する FileSetAccessMode
  - Windows でファイルの属性を設定する FilesSetAttribute
- 次のタグはサポートされません。
  - cfpdf
  - cfpdfform
- 次のシナリオはサポートされません。
  - cfreport タグは、RAM 上のテンプレートレポートを受け入れません。したがって、次のタグは機能しません。

    ```
    <cfreport format="PDF" template="ram:///myReport1.cfr" filename="ram:///test.pdf" overwrite="yes">
    ```

    この場合、myReport1.cfr はディスク上に存在している必要があります。
  - cfimport タグは、RAM 上のタグライブラリを受け入れません。たとえば、次のタグは機能しません。

    ```
    <cfimport prefix="custom" taglib="ram:///a/b/mytags.jar">
    ```
  - 次のシンタックスに示すような、複数のファイルシステムにまたがる名称変更はサポートされません。

    ```
    <cffile action="rename" source="ram:///src1/message2.txt" destination="#ExpandPath('./')#/123.txt">.
    ```

    このような場合は、ファイルを移動して対処してください。

- cfexecute タグを使用する場合、実行可能ファイルは、RAM 上ではなく、ローカルファイルシステム上に存在している必要があります。ただし、次のコードに示すように、出力先はメモリ内ファイルに設定できます。

```
<cfexecute name="C:\WINDOWS\System32\netstat.exe" arguments="-e" outputFile="ram:///output.txt"
timeout="1">
</cfexecute>
<cfset thisPath=ExpandPath("*.*")>
```

アプリケーション固有のメモリ内ファイルシステム

ColdFusion 9 では、メモリ内ファイルシステムはサーバーレベルでのみサポートされています。しかし、今回のリリースでの機能強化により、アプリケーション固有のメモリ内ファイルシステムを使用できるようになりました。これにより、仮想ファイルシステムでアプリケーションの分離を実現できます。つまり、あるアプリケーションによってメモリ内ファイルシステムに作成されたファイルには、別のアプリケーションからアクセスできません。

この設定は、Application.cfc で次のように指定できます。

変数	説明
this.inmemoryfilesystem.enabled	アプリケーション用のメモリ内ファイルシステムを有効にするには、値を true に設定します。 これはデフォルトの設定です。
this.inmemoryfilesystem.size	メモリ内ファイルシステムのメモリ制限（MB 単位）を指定します。 また、この値は ColdFusion Administrator でも指定できます（サーバーの設定／設定／メモリ内仮想ファイルシステムのアプリケーションごとのメモリ制限）。 小さい方の値が優先されます。

仮想ファイルシステム：HTTP、FTP および ZIP のサポート

ColdFusion 10 では、FTP または HTTP プロトコルによってネットワーク経由でファイルにアクセスできます。また、ZIP のサポートも提供されます。

規則については、Apache Commons Documentation を参照してください。

例

FTP の場合

```
<cffile
    action = "read"
    file = "ftp://Administrator:Password@Host_Name/ReadmeLater.htm"    variable = "varvar">
<cffile action="write" file="ftp://Administrator:Password@Host_Name/ReadmeLater.htm" output="new stuff
added">
```

ZIP の場合

```
<cffile
    action = "read"
    file="zip:#ExpandPath('./')#/hello.zip!/hello.txt"
    variable = "varRead1">
```

データベース使用の最適化

アプリケーションの効率が悪くなる最も一般的な原因は、不十分なデータベース設計、および不適切または非効率的なデータベースの使用です。アプリケーションを設計するときは、データベースやデータベースから取得した情報の使用方法を工夫してください。たとえば、SQL クエリーから多くの製品の価格を平均する場合は、ColdFusion でループを使用するよりも、SQL を使用したほうがより効率的に平均値を得ることができます。

データベースの使用を最適化するのに役立つ ColdFusion 機能としては、cfstoredproc タグと、cfquery タグの cachedWithin 属性の 2 つがあります。

ストアドプロシージャの使用

cfstoredproc タグを使用すると、ColdFusion でデータベース管理システムのストアドプロシージャを使用できます。ストアドプロシージャとは、名前を付けてコンパイルされ、データベースシステムに保管されている SQL ステートメントのシーケンスのことです。ストアドプロシージャにより SQL ステートメントのプログラミングロジックがカプセル化され、データベースシステムはストアドプロシージャを効率的に実行できるように最適化されています。したがって、ストアドプロシージャは cfquery タグより高速です。

cfprocparam タグを使用してストアドプロシージャにパラメータを送信し、cfprocresult タグを使用してストアドプロシージャによって返されるレコードセットを取得します。

次の例では、3 つの結果セットを返す Sybase ストアドプロシージャを実行します。この例ではそのうちの 2 つを使用します。このストアドプロシージャはステータスコードと 1 つの出力パラメータを返し、それを例で表示しています。

```
 <!--- cfstoredproc tag --->
<cfstoredproc procedure = "foo_proc" dataSource = "MY_SYBASE_TEST"
    username = "sa" password = "" returnCode = "Yes">

    <!--- cfprocresult tags --->
    <cfprocresult name = RS1>
    <cfprocresult name = RS3 resultSet = 3>

    <!--- cfprocparam tags --->
    <cfprocparam type = "IN"
        CFSQLType = CF_SQL_INTEGER
            value = "1">
    <cfprocparam type = "OUT"CFSQLType = CF_SQL_DATE
        variable = FOO>
<!--- Close the cfstoredproc tag. --->
</cfstoredproc>

<cfoutput>
    The output param value: '#foo#'<br>
</cfoutput>

<h3>The Results Information</h3>
<cfoutput query = RS1>
    #name#,#DATE_COL#<br>
</cfoutput>
```

```
<br>
<cfoutput>
    <hr>
    Record Count: #RS1.recordCount#<br>
    Columns: #RS1.columnList#<br>
    <hr>
</cfoutput>

<cfoutput query = RS3>
    #col1#,#col2#,#col3#<br>
</cfoutput>
<br>
<cfoutput>
    <hr><br>
    Record Count: #RS3.recordCount#<br>
    Columns: #RS3.columnList#<br>
    <hr>

    The return code for the stored procedure is: '#cfstoredproc.statusCode#'<br>
</cfoutput>
```

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfstoredproc procedure = "foo_proc" dataSource = "MY_SYBASE_TEST" username = "sa" password = "" returnCode = "Yes">`	ストアドプロシージャ foo_proc を MY_SYBASE_TEST データソースで実行します。cfstoredproc の statusCode 変数に、ストアドプロシージャによって返されるステータスコードを設定します。
`<cfprocresult name = RS1>` `<cfprocresult name = RS3 resultSet = 3>`	ストアドプロシージャから 2 つのレコードセットを取得します。この 2 つはストアドプロシージャによって返される最初と 3 番目の結果セットです。

`<cfprocparam type = "IN" CFSQLType = CF_SQL_INTEGER value = "1">`
`<cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE variable = FOO>`
`<!--- Close the cfstoredproc tag. --->`
`</cfstoredproc>`
	ストアドプロシージャの 2 つのパラメータ (入力パラメータと出力パラメータ) を指定します。入力パラメータに 1、出力を取得する ColdFusion 変数として FOO を設定します。
	cfstoredproc タグ本文を終了します。

`<cfoutput>` `The output param value: '#foo#' ` `</cfoutput>` `<h3>The Results Information</h3>` `<cfoutput query = RS1>` `#name#,#DATE_COL# ` `</cfoutput>` ` ` `<cfoutput>` `<hr>` `Record Count: #RS1.recordCount# ` `Columns: #RS1.columnList# ` `<hr>` `</cfoutput>` `<cfoutput query = RS3>` `#col1#,#col2#,#col3# ` `</cfoutput>` ` ` `<cfoutput>` `<hr> ` `Record Count: #RS3.recordCount# ` `Columns: #RS3.columnList# ` `<hr>` `The return code for the stored procedure is:` `'#cfstoredproc.statusCode#' ` `</cfoutput>`	ストアドプロシージャの実行結果を表示します。 • 出力パラメータ値。 • 名前と DATE_COL 変数によって識別される最初のレコードセット内の 2 つの列の内容。これらの変数の値は、ページの別の場所で設定します。 • 最初のレコードセットの行数と列名。 • col1、col2、および col3 という変数によって識別されるもう 1 つのレコードセット内の列の内容。 • レコードセットの行数と列名。 • ストアドプロシージャによって返されるステータス値。

ストアドプロシージャ作成の詳細については、データベース管理ソフトウェアのマニュアルを参照してください。cfstoredproc タグの使用方法の詳細については、『CFML リファレンス』を参照してください。

cfquery タグの cachedWithin 属性の使用

cfquery タグの cachedWithin 属性は、データベースクエリーの結果を一定の時間保存するように指定します。最初のページリクエストでデータベースにアクセスすれば、一定の時間が経過するまでデータベースへのクエリーは実行されません。cachedWithin 属性を使用すると、すぐに変更されることのないデータベースへのアクセスのオーバーヘッドを大幅に限定できます。

このテクニックが役立つのは、特定の時間における変更のみがデータベースに含まれている場合や、データベースの変更頻度が低く最新の結果が必ずしも必要でない場合などです。

cachedWithin 属性値を (日数、時間、分、秒の形式で) 指定するには、CreateTimeSpan 関数を使用します。たとえば、次のコードでは、cfdocexamples データソースの Employees テーブルの内容を取得した結果を 1 時間キャッシュに保持します。

```
<cfquery datasource="cfdocexamples" name="master"
    cachedWithin="#CreateTimeSpan(0,1,0,0)#">
    SELECT * FROM Employees
</cfquery>
```

ユーザーへの視覚的フィードバック

アプリケーションでデータの処理に長い時間を要する場合は、何らかの問題があると考えたユーザーがページを再びリクエストすることを防ぐために、処理が実行中であることを視覚的なフィードバックによって示すことをお勧めします。これによってアプリケーションの処理効率が最適化されることはありませんが、ユーザーはアプリケーションのレスポンスが向上したように感じます。

cfflush タグを使用すると、692 ページの「データの取得および形式設定の概要」で示すようにデータの一部をユーザーに返すことができます。

また、cfflush タグを使用して、進行状況表示バーを作成することもできます。

エラー処理

Adobe ColdFusion には、アプリケーションで発生するエラーに対応するためのツールやテクニックが数多く用意されています。そのツールとしては、エラー処理メカニズムやエラーロギングツールなどがあります。

ユーザー入力の検証の詳細については、692 ページの「データの取得および形式設定の概要」および 709 ページの「cfform タグによるダイナミックフォームの作成」を参照してください。デバッグの詳細については、379 ページの「アプリケーションのデバッグとトラブルシューティング」を参照してください。

ColdFusion におけるエラー処理について

デフォルトでは、ColdFusion でエラーが発生すると固有のエラーメッセージが生成されます。ColdFusion に用意されているさまざまなツールやテクニックを使用すれば、エラー情報をカスタマイズしたり、エラー発生時に独自の処理を行うことができます。エラー処理テクニックには、次のものがあります。

- 次の場合に ColdFusion のカスタムページを表示できます。
  - ColdFusion ページが見つからない場合 ([見つからないテンプレートハンドラ] ページ)

- ページの処理中に発生した例外エラーが、他のいずれの方法でも処理されなかった場合 ([サイト全体のエラーハンドラ] ページ)

これらのページは、ColdFusion Administrator の [サーバーの設定] の [設定] ページで指定します。詳細については、ColdFusion Administrator のヘルプを参照してください。

- cferror タグを使用して、特定のエラータイプを処理する ColdFusion ページを指定します。
- cftry タグ、cfcatch タグ、cfthrow タグ、および cfrethrow タグを使用して、例外エラーが発生したページで直接その例外エラーを検出し、処理します。
- CFScript 内で try および catch ステートメントを使用して例外を処理します。
- "Application.cfc" で onError イベントを使用して、アプリケーションページの try/catch コードで処理されなかった例外エラーを処理します。
- エラーをロギングします。ColdFusion では、デフォルトで一定のエラーがロギングされます。cflog タグを使用すると、他のエラーをロギングすることができます。

以降のトピックでは、次の情報について説明します。

- ColdFusion エラーのタイプと ColdFusion によるエラーの処理方法に関する基本事項
- cferror タグを使用してエラー処理ページを指定する方法
- エラーをロギングする方法
- ColdFusion 例外の処理方法

注意：ここでは cftry および cfcatch タグの使用方法について説明しますが、CFScript の try および catch ステートメントについては説明しません。ここに記載されている例外処理の一般的な説明は、タグと CFScript ステートメントの両方に適用されます。ただし、CFScript で使用するコードおよび適用される情報は、タグのものとは異なります。CFScript での例外処理の詳細については、162 ページの「UDF でのエラー処理」を参照してください。

エラーについて

エラーを捉える観点にはさまざまなものがあります。たとえば、エラーをその原因で捉えることができます。また、その影響、特にアプリケーションがエラーを修復できるかという観点から捉えることもできます。さらに、ColdFusion の動作という観点から捉えることもできます。ここでは、これらについてエラーの捉え方について説明します。

エラー原因と修復

エラーには多様な原因が考えられます。原因によっては、エラーを修復できる場合があります。修復可能なエラーとは、アプリケーションでエラー原因を識別でき、その問題についてアクションを実行することができるものを指します。エラーの中には、タイムアウトエラーのように、エラーが検出されたことをユーザーに通知せずに修復できるものもあります。リクエストされたページが存在しないというエラーは修復できません。このエラーの場合、アプリケーションはエラーメッセージの表示のみを行います。

検証エラーのように、リクエストの処理は継続できないが、エラー固有のレスポンスを返せるエラーも、修復可能と考えられます。たとえば、数値が必要な箇所にテキストを入力したために発生するエラーは、アプリケーションでエラーを認識でき、データフィールドを再表示して、エラー原因を提示してデータの再入力を促すことができるので、修復可能と考えることができます。

エラーの中には、状況によって修復できないものもあります。たとえば、タイムアウトエラーが発生した場合はリクエストを再実行するのが一般的ですが、リクエストが常にタイムアウトとなるような場合についても対策を立てておく必要があります。

エラー原因は、次の表にリストされている大まかなカテゴリに分類されます。

カテゴリ	説明
プログラムエラー	コードシンタックス内またはプログラムロジック内に原因があります。CFML を Java クラスにコンパイルするときに ColdFusion コンパイラによってプログラムシンタックスエラーが識別され、レポートされます。アプリケーションロジック内のエラーを検出することは困難です。デバッグのツールおよびテクニックの詳細については、379 ページの「アプリケーションのデバッグとトラブルシューティング」を参照してください。 ColdFusion のシンタックスエラーとは異なり、SQL シンタックスエラーは実行時にのみ検出されます。
データエラー	通常、データエラーはユーザーの入力エラーです。検証テクニックを使用してユーザーの入力データ内のエラーを識別し、ユーザーがエラーを修正できるようにします。
システムエラー	システムエラーは、データベースシステムの問題、サーバー上での過剰な需要によるタイムアウト、システム内のメモリ不足エラー、ファイルエラー、ディスクエラーなどのさまざまな原因から発生します。

これらのカテゴリは、ColdFusion がエラーを分類する方法に完全にはマッピングされませんが、エラーについて考える有効な方法を提供し、コード内のエラーの予防や処理を行う上で役立ちます。

ColdFusion のエラータイプ

ColdFusion のエラーを効果的に管理するためには、ColdFusion によるエラー分類方法とエラー処理方法について理解する必要があります。次の表に、ColdFusion でのエラー分類の詳細を示します。

タイプ	説明
例外	通常の処理の継続を阻害するエラー。ColdFusion の例外はすべて、根本的に Java 例外です。
テンプレートの欠如	検出できない ColdFusion ページを要求する HTTP リクエスト。存在しない ColdFusion ページがブラウザによってリクエストされる場合に生成されます。 テンプレート欠如エラーは、cfinclude タグやカスタムタグでターゲットが検出できない場合に発生するインクルード欠如例外とは異なります。
フォームフィールドのデータ検証	サーバーサイドのフォームフィールド検証エラーは、ColdFusion の特殊な例外です。cfform 属性または非表示 HTML フォームフィールドを使用して、サーバーサイドフォーム検証を指定します。cfparam タグなどの、その他のすべてのタイプのサーバーサイド検証では、ランタイム例外が生成されます。フォームフィールド検証の詳細については、729 ページの「データの検証」を参照してください。 ColdFusion には、サーバーサイドフォームフィールド検証用のエラーページがあらかじめ用意されています。cferror タグには type 属性が用意されており、この属性を使用してカスタムエラーページでこうしたエラーを処理できますが、"Application.cfc" で onError 処理、または try/catch エラー処理を使用する場合、エラーはアプリケーション例外として表示されます。"Application.cfc" でのフォームフィールド検証の処理の詳細については、244 ページの「onError メソッドによるサーバーサイド検証エラーの処理」を参照してください。

注意：onSubmit および onBlur を使用したフォームフィールド検証テクニックは、クライアントのブラウザで JavaScript または Flash 検証を使用するものです。

ColdFusion 例外について

ColdFusion エラーの大部分は例外です。ColdFusion 例外は、次の 2 通りの考え方で分類できます。

- 発生する状況
- 例外のタイプ

例外が発生する状況

ColdFusion エラーが発生する可能性があるのは、CFML が Java にコンパイルされるときと、その結果の Java を実行したとき (ランタイム例外と呼ばれている) の 2 回です。

コンパイラ例外

コンパイラ例外は、CFML を Java にコンパイルするときに ColdFusion が検出するプログラミングエラーです。コンパイラ例外は ColdFusion ページが実行可能コードに変換される前に発生するので、コンパイラ例外が発生したページでその例外を処理することはできませんが、他のページでこれらのエラーを処理することができます。詳細については、282 ページの「コンパイラ例外の処理」を参照してください。

ランタイム例外

ランタイム例外は、コンパイル済みの ColdFusion Java コードの実行時に発生します。これは、アプリケーションの通常の命令の流れを妨げるイベントです。例外は、システムエラーまたはプログラム論理エラーから発生することがあります。ランタイム例外には次のものがあります。

- ODBC ドライバや CORBA サーバーなどの外部サービスからのエラーレスポンス
- CFML エラー、cfthrow、または cfabort タグの結果
- ColdFusion の内部エラー

ColdFusion の例外のタイプ

ColdFusion の例外にはタイプが存在し、cferror、cfcatch、および cfthrow といったエラー処理タグで指定します。cferror または cfcatch タグでは、指定したタイプの例外のみが処理されます。例外タイプを指定する識別子は、次のタイプカテゴリに分類されます。

- 基本
- カスタム
- 拡張
- Java クラス

注意：cfthrow タグでは、カスタムエラータイプ名および Application 基本タイプのみを使用してください。他のすべてのビルトイン例外タイプ名は、システムによって識別される特定のエラータイプを表しているので、自分のコードで識別するエラーに対しては使用しないでください。

基本例外タイプ

カスタム例外を除くすべての ColdFusion 例外は、基本タイプのカテゴリに属しています。基本カテゴリを構成する各タイプによって、ColdFusion 例外が大まかに分類されています。次の表で、基本例外タイプについて説明します。

タイプ	タイプ名	説明
データベースエラー	Database	SQL ステートメントの失敗や ODBC の問題などの、データベース操作の失敗によって発生するエラー。
インクルードファイル欠如エラー	MissingInclude	cfinclude タグ、cfmessagebox タグ、および cferror タグによって指定されたファイルが欠如している場合に発生するエラー。cferror タグで missingInclude エラーが生成されるのは、タグ内で指定したタイプのエラーが発生した場合のみです。 MissingInclude エラータイプは、テンプレートエラーのサブカテゴリです。MissingInclude エラータイプを特別に処理しなくても Template エラータイプを処理すれば、テンプレートエラーハンドラによってこれらのエラーが検出されます。MissingInclude エラーは実行時に検出されます。
テンプレートエラー	Template	無効なタグおよび属性名などの、一般的なアプリケーションページエラー。Template エラーの大部分は、実行時ではなくコンパイル時に検出されます。
オブジェクト例外	Object	オブジェクトを操作する ColdFusion コードで発生する例外。
セキュリティ例外	Security	セキュリティを操作する ColdFusion コードで発生する検出可能な例外。

タイプ	タイプ名	説明
式の例外	Expression	1 と "a" を足すなど、式の評価の失敗によって発生する例外。
ロック例外	Lock	cflock の重要なセクションがタイムアウトになったり、実行時に失敗したりするなど、ロック操作の失敗によって発生する例外。
cfthrow によって生成されたアプリケーション定義による例外イベント	Application	cfthrow タグによって生成される、タイプを指定しないカスタムの例外、またはタイプを Application として指定するカスタムの例外。
すべての例外	Any	すべての例外。この表内のすべてのタイプ、および他のエラーハンドラで処理されないすべての例外が含まれます。この例外には、予期されない内部エラーおよび外部エラーも含まれます。

注意：Any タイプには、java.lang.Exception という Java オブジェクトタイプのエラーがすべて含まれます。java.lang.Throwable エラーは含まれません。すべての Throwable エラーを検出するには、cfcatch タグの type 属性で java.lang.Throwable を指定します。

カスタム例外

cfthrow タグ内で、MyCustomErrorType などのカスタム例外のタイプ名を指定すると、独自のタイプの例外を生成できます。次に、cfcatch または cferror タグにカスタムタイプ名を指定して、例外を処理します。カスタムタイプ名は、基本タイプや Java 例外クラスなどのビルトインタイプ名とは異なるものにしてください。

拡張例外タイプ

拡張例外タイプは、特定の限定された例外タイプで構成されます。これらのタイプは、下位互換性を維持するために ColdFusion でサポートされています。

Java 例外クラス

ColdFusion の各例外は、基本、カスタム、または拡張タイプに加え、特定の Java 例外クラスにも属しており、これによって識別されます。例外の標準エラー出力では、スタックトレースの 1 行目で、例外の Java クラスが識別されます。

たとえば、整数の変数に ArrayIsEmpty などの配列関数を使用すると、ColdFusion では Expression 例外の基本タイプと coldfusion.runtime.NonArrayException Java クラスに属する例外が生成されます。

一般的に、多くのアプリケーションでは、例外の識別に Java 例外クラスを使用する必要はありません。ただし、Java クラス名を使用して非 CFML Java オブジェクト内の例外を検出することができます。たとえば、次の行によってすべての Java 入力および出力の例外を検出することができます。

```
<cfcatch type="java.io.IOException">
```

ColdFusion によるエラーの処理方法

次に、ColdFusion でのエラーの処理方法について簡単に説明します。詳細については、以降の各トピックで説明します。

テンプレート欠如エラー

ColdFusion で検出できないページをユーザーがリクエストした場合、Administrator の [サーバーの設定] にある [見つからないテンプレートハンドラ] フィールドで [見つからないテンプレートハンドラ] ページが指定されていれば、そのページを使用してエラー情報が表示されます。指定されていなければ、標準のエラーメッセージが表示されます。

フォームフィールド検証エラー

ユーザーが、onServer または非表示フォームフィールドサーバーサイドデータ検証を使用する HTML タグに無効なデータを入力すると、ColdFusion では次の処理が実行されます。

1 アプリケーション CFC ("Application.cfc") に onError イベントハンドラメソッドが設定されている場合は、このメソッドが呼び出されます。

2 "Application.cfc" の初期化コードまたは "Application.cfm" ページに、検証エラーハンドラを指定する cferror が設定されている場合は、指定されたエラーページが表示されます。

3 それ以外の場合は、デフォルトのヘッダ、エラーを説明する箇条書きリスト、およびデフォルトのフッタで構成される標準形式のエラー情報が表示されます。

非表示フォームフィールド検証の使用方法の詳細については、729 ページの「データの検証」を参照してください。
"Application.cfc" の詳細については、231 ページの「ColdFusion アプリケーションの設計と最適化」を参照してください。

コンパイラ例外エラー

ColdFusion で発生したコンパイラ例外の処理方法は、リクエストされたページとインクルードページのどちらでエラーが発生したのかによって異なります。

- cfinclude または cfmessagebox タグによってアクセスされるページでエラーが発生した場合、あるいは、cf_ 表記法を使用してアクセスするカスタムタグページでエラーが発生した場合、そのエラーは、そのタグにアクセスしているページ内のランタイム例外として処理されます。これらのエラーの処理方法の詳細については、ランタイム例外エラーを参照してください。
- リクエストされたページでエラーが直接発生し、Administrator の [設定] の [サイト全体のエラーハンドラ] フィールドにエラーハンドラページが指定されている場合は、指定されたエラーページが表示されます。それ以外の場合は、280 ページの「エラーメッセージと標準エラー形式」で説明する標準のエラーメッセージ形式を使用してエラーがレポートされます。

ランタイム例外エラー

ColdFusion でランタイム例外が発生する場合は、次の表の中で最初に該当する条件のアクションが実行されます。

条件	アクション
エラーのあるコードが cftry タグ内にあり、例外タイプが cfcatch タグ内に指定されている場合	cfcatch タグ内のコードが実行されます。 cftry ブロック内にこのエラーの cfcatch タグがない場合は、適切な cferror ハンドラまたはサイト全体のエラーハンドラがテストされます。
ColdFusion アプリケーションに、onError メソッドが設定された "Application.cfc" が含まれる場合	onError メソッドのコードが実行されます。onError メソッドの使用方法の詳細については、243 ページの「"Application.cfc" でのエラー処理」を参照してください。
cferror タグによって、その例外タイプの例外エラーハンドラが指定されている場合	cferror タグによって指定されているエラーページが使用されます。
Administrator の [サーバーの設定] にある [見つからないテンプレートハンドラ] フィールドでエラーハンドラページが指定されている場合	Administrator の設定によって指定されているカスタムエラーページが使用されます。
cferror タグによってリクエストエラーハンドラが指定されている場合	cferror タグによって指定されているエラーページが使用されます。
デフォルト	標準のエラーメッセージ形式が使用されます。

たとえば、cftry ブロック内にない CFML コードで例外が発生し、"Application.cfc" に onError メソッドが設定されておらず、cferror タグにこのエラータイプを処理するページが指定されている場合は、指定されているエラーページが使用されます。

エラーメッセージと標準エラー形式

アプリケーションでエラーが処理されない場合は、ColdFusion によってユーザーのブラウザに診断メッセージが表示されます。

また、エラー情報がログファイルに書き込まれるので、後で見直すことができます。エラーロギングの詳細については、285 ページの「cflog タグによるエラーのロギング」を参照してください。

標準エラー形式は、次の表に示す情報で構成されます。常にすべてのセクションが表示されるとは限りません。

セクション	説明
エラーの説明	(通常は) オンラインによる、簡単なエラー説明。
エラーメッセージ	エラーの詳細な説明。表示されるエラーメッセージの診断情報はエラーのタイプによって異なります。たとえば、タグに無効な属性が指定されている場合は、有効なすべてのタグ属性のリストがこのセクションに表示されます。
エラーの場所	エラーが発生したページおよび行番号と、その行の周辺の CFML セクション。このセクションは、エラーによっては表示されない場合があります。 エラーの原因となった行が、ColdFusion で問題が検出された場所よりも何行か前にある場合があります。その場合は、エラーの原因となった行が表示に含まれないことがあります。
リソース	マニュアル、ナレッジベース、その他のリソースなど、問題の解決に役立つ情報へのリンク。
エラー環境情報	エラーの原因となったリクエストに関する情報。すべてのエラーメッセージには次の項目が含まれています。 • ユーザーのブラウザ • ユーザーの IP アドレス • エラーの日付と時刻
スタックトレース	例外発生時の Java スタック。例外の特定の Java クラスを含んでいます。これは Adobe テクニカルサポートに問い合わせる場合に役立ちます。 スタックトレースはデフォルトでは閉じています。トレースを表示するには、見出しをクリックしてください。

エラーの原因が明確に識別されていないメッセージを受け取った場合は、使用可能メモリやディスク容量など、主なシステムパラメータを調べてください。

エラー処理方法の決定

279 ページの「ColdFusion によるエラーの処理方法」で説明したように、ColdFusion には、エラー処理、特に例外処理のオプションが多数あります。ここでは、使用するエラー処理形式の決定方法について説明します。

テンプレート欠如エラーの処理

テンプレート欠如エラーは、.cfm で終わる見つからないページに対する HTTP リクエストを受け取ったときに発生します。独自のテンプレート欠如エラーページを作成して、アプリケーション固有の情報や外観を表示することができます。テンプレート欠如エラーページは、Administrator の [設定] ページで指定します。

欠如エラーのページでは、CFML タグおよび変数を使用できます。特に、リクエストされたページを識別するために、テキスト内で次のように CGI.script_name 変数を使用できます。

```
<cfoutput>The page #Replace(CGI.script_name, "/", "")# is not available.<br>
Make sure that you entered the page correctly.<br>
</cfoutput>
```

このコードでは、表示をわかりやすくするために、Replace 関数でスクリプト名から先頭のスラッシュ記号を削除しています。

フォームフィールド検証エラーの処理

サーバーサイドのフォームフィールド検証には、エラーの原因をわかりやすく説明するデフォルトの検証エラーメッセージが用意されています。ただし、このエラーメッセージの形式をカスタマイズしたり、サービスの問い合わせ先電話番号やアドレスなどの情報を追加したい場合もあります。そのような場合は、"Application.cfc" の初期化コードで cferror タグの

Validation 属性を使用します。または、"Application.cfm" ページで独自の検証方法を指定します。「検証エラーページの例」では、このようなページの例を紹介しています。"Application.cfc" の onError メソッドにフォームフィールド検証エラーの処理コードを記述することもできます。

コンパイラ例外の処理

コンパイラ例外は、ColdFusion でページコードが実行される前に検出されるので、発生したページで直接コンパイラ例外を処理することはできません。コンパイラ例外はすべて、開発プロセスの過程で修正してください。このエラーの原因を識別して修正するには、379 ページの「アプリケーションのデバッグとトラブルシューティング」で説明されている、レポートされたエラーメッセージおよびコードのデバッグテクニックを使用してください。

cfinclude または cfmessagebox タグを使用してアクセスするページで発生するコンパイラ例外は、cftry ブロックで cfinclude または cfmodule タグを囲むことで、ランタイムエラーとして処理できます。アクセス先のページで発生したコンパイラ例外は、ベースページのランタイムエラーとして検出されます。ただし、コンパイラエラーの正しい解決方法はアプリケーションをデプロイする前にエラーを除去することなので、このような対処方法で " 解決 " するのは避けてください。

ランタイム例外の処理

例外の処理にはさまざまな方法がありますが、適切な方法はアプリケーションやその要件によって異なります。次の表に、適切なテクニックの選択に役立つ情報を示します。

方法	用途
cftry	特定のコード部分で例外が発生する可能性があり、コード固有のエラーメッセージを表示するなどの、コンテキスト固有の方法でその例外を処理したい場合は、cftry ブロックの中にそのコード部分を配置します。 cftry ブロックは、例外から回復できる場合に使用します。たとえば、タイムアウトとなったオペレーションの再試行や、別のリソースへのアクセスを試行できます。また、欠如しているリソースが必要ない場合など、その例外がアプリケーションに悪影響を及ぼさない場合にも、cftry タグを使用して処理を継続できます。 詳細については、287 ページの「ColdFusion タグでのランタイム例外の処理」を参照してください。
"Application.cfc" の onError メソッド	"Application.cfc" に onError メソッドを実装することで、アプリケーション内の複数のコードセクションで生成されるアプリケーション固有の例外に対して、一貫した処理方法を提供できます。"Application.cfc" を使用した例外処理の詳細については、243 ページの「"Application.cfc" でのエラー処理」を参照してください。
cferror で例外固有のエラーハンドラページを指定する	cferror タグを使用することで、特定の例外タイプに対するエラーページを指定します。これらのページではエラーから回復できませんが、エラーの原因や防止方法などを表示できます。 詳細については、283 ページの「cferror タグによるカスタムエラーメッセージの指定」を参照してください。
cferror でリクエストエラーページを指定する	cferror タグを使用することで、修復できない例外に対してアプリケーション固有のカスタムメッセージを表示するリクエストエラーハンドラを指定します。アプリケーション内のすべてのページに適用されるように、"Application.cfc" 初期化コードまたは "Application.cfm" ページにこのタグを追加します。 リクエストエラーページでは CFML タグを使用できませんが、エラー変数は表示できます。したがって、通常のエラー情報は表示できますが、エラー固有の手順を提示することはできません。通常、リクエストページでは、エラー変数の値とサポートへの問い合わせ情報などのアプリケーション固有の情報を表示します。 コード例については、「リクエストエラーページの例」を参照してください。
サイト全体のエラーハンドラページ	Administrator でサイト全体のエラーハンドラを指定することで、サーバー上のすべてのアプリケーションにおいて、他のいずれの方法でも処理されなかったすべての例外を一貫した外観と内容で処理します。 リクエストページと同様に、サイト全体のエラーハンドラではエラーを修復できません。ただし、エラー変数の他に CFML タグも使用できます。 サイト全体のエラーハンドラを使用すると、ColdFusion のデフォルトのエラーメッセージが表示されなくなるので、ユーザーに提供する情報を制限することができます。また、デフォルトの問い合わせ先情報やその他の手順をすべてのユーザーに提供することができます。

cferror タグによるカスタムエラーメッセージの指定

カスタムエラーページを使用すると、ユーザーに提供するエラー情報を制御できます。カスタムエラーページはエラータイプ別に指定できるので、エラータイプに応じた処理を実行できます。たとえば、リクエストのタイムアウトなどの修復可能なエラーを処理するためのページを作成できます。また、エラーメッセージをアプリケーションの外観と調和したものにすることもできます。

次のタイプのカスタムエラーメッセージのページを指定できます。

タイプ	説明
Validation	サーバーサイドのフォームフィールドデータ検証エラーを処理します。検証エラーページには、CFML タグを含めることはできませんが、エラーページの変数は表示できます。 この属性を使用できるのは、"Application.cfc" の初期化コードまたは "Application.cfm" ページのみです。他のページで使用しても、効果はありません。したがって、1 つのアプリケーションで指定できる検証エラーページは 1 つのみです。このページはすべてのサーバーサイド検証エラーに適用されます。
Exception	特定の例外エラーを処理します。例外のタイプごとに個別のエラーページを指定できます。
Request	他のいずれの方法でも処理されなかった例外を処理します。リクエストエラーページは、CFML 言語プロセッサの終了後に実行されます。したがって、リクエストエラーページで CFML タグを使用することはできませんが、エラーページの変数は表示できます。リクエストエラーページは、他のエラーハンドラでエラーが発生した場合のバックアップとして役立ちます。

カスタムエラーページの指定

カスタムエラーページは、cferror タグを使用して指定します。検証エラーに対するカスタムエラーページを指定する場合は、"Application.cfc" の初期化コードまたは "Application.cfm" ページにこのタグを配置する必要があります。例外エラーおよびリクエストエラーに対するカスタムエラーページを指定する場合は、各アプリケーションページで設定できます。ただし、一般的にカスタムエラーページはアプリケーション全体に適用するものなので、これらの cferror タグは "Application.cfc" または "Application.cfm" ファイルにも配置するとより効果的です。これらのタグの使用方法の詳細については、231 ページの「ColdFusion アプリケーションの設計と最適化」を参照してください。

cferror タグには次の表にリストされている属性があります。

属性	説明
Type	ColdFusion がこのページを表示する原因となるエラーのタイプ。Exception、Request、Validation のいずれかです。
Exception	Exception タイプの場合にのみ使用します。このページを表示する原因となる例外または例外のカテゴリを指定します。277 ページの「ColdFusion 例外について」で説明しているいずれかのタイプを指定できます。
Template	表示する ColdFusion ページ。
MailTo	(オプション) 電子メールアドレス。cferror タグによって、エラーページの error.mailTo 変数にこの値が設定されます。エラー通知を送信するように伝えるメッセージをエラーページで表示するときに、error.mailTo の値を使用できます。ColdFusion によって自動的にメッセージが送信されることはありません。

次の cferror タグでは、ロックされているコードで発生した例外に対するカスタムエラーページを指定し、このタイプのエラーが発生するたびに通知を送信できるように、使用可能な電子メールアドレスをこのエラーページに知らせます。

```
<cferror type = "exception"
    exception = "lock"
    template = "../common/lockexcept.cfm"
    mailto = "server@mycompany.com">
```

cferror タグの詳細については、『CFML リファレンス』を参照してください。

エラーアプリケーションページの作成

次の表に、エラーアプリケーションページに適用されるルールおよび注意事項を示します。

タイプ	注意事項
Validation	• CFML タグは使用できません。 • HTML タグは使用できます。 • シャープ記号 (#) で囲むことによって、Error.InvalidFields、Error.validationHeader、および Error.validationFooter 変数を使用できます。 • 他の CFML 変数を使用することはできません。
Request	• CFML タグは使用できません。 • HTML タグは使用できます。 • シャープ記号 (#) で囲むことによって、Error.Diagnostics などの 9 つの CFML エラー変数を使用できます。 • 他の CFML 変数を使用することはできません。
Exception	• タグ、関数、変数を含むすべての CFML シンタックスを使用できます。 • 9 つの標準 CFML エラー変数および cfcatch 変数を使用できます。両方のタイプの変数に、接頭辞として、Error または cferror のいずれかを使用できます。 • 他のアプリケーション定義の CFML 変数を使用できます。 • 任意の CFML 変数を表示するには、cfoutput タグを使用します。

次の表に、エラーページで使用できる変数を示します。また、289 ページの「cfcatch ブロック内の例外情報」にリストされているすべての例外変数も、例外ページエラーで使用できます。これらの変数を使用するには、cfcatch 接頭辞を cferror または error に置き換えます。たとえば、エラーページで例外メッセージを使用するには、error.message のように参照します。

一般に、本番用の例外ページやリクエストページでは、error.diagnostics 変数の内容などの、詳細なエラー情報は表示すべきではありません。例外ページでは、詳細なエラー情報をユーザーに表示するのではなく、管理者のアドレスに電子メールを送信したり、cflog タグを使用して情報をロギングするのが普通です。cflog タグの使用方法の詳細については、285 ページの「cflog タグによるエラーのロギング」を参照してください。

リクエストエラーページの例

リクエストエラーに対するカスタムエラーページの例を次に示します。

```
 <html>
<head>
<title>Products - Error</title>
</head>
<body>

<h2>Sorry</h2>

<p>An error occurred when you requested this page.</p>
<p>Please send e-mail with the following information to #error.mailTo# to report
    this error.</p>

<table border=1>
<tr><td><b>Error Information</b> <br>
    Date and time: #error.DateTime# <br>
    Page: #error.template# <br>
    Remote Address: #error.remoteAddress# <br>
    HTTP Referer: #error.HTTPReferer#<br>
</td></tr></table>

<p>We apologize for the inconvenience and will work to correct the problem.</p>
</body>
</html>
```

検証エラーページの例

検証エラーに対する簡単なカスタムエラーページの例を次に示します。

```
 <html>
<head>
<title>Products - Error</title>
</head>
<body>

<h2>Data Entry Error</h2>

<p>You failed to correctly complete all the fields
in the form. The following problems occurred:</p>

#error.invalidFields#

</body>
</html>
```

cflog タグによるエラーのロギング

『ColdFusion 設定と管理』で説明されているように、ColdFusion には、ログファイルの生成、管理、および表示のための機能が数多く用意されています。また、ColdFusion ログにエントリを追加する cflog タグもあります。

デフォルトのエラーハンドラを使用する場合は、ColdFusion によってデフォルトのログにエラーが自動的にロギングされます。それ以外の場合は、エラー処理コードで cflog タグを使用して、ログエントリを生成します。

cflog タグでは、次の情報を指定できます。

- メッセージをロギングするカスタムファイルまたは標準 ColdFusion ログファイル。
- ログファイルに記述するテキスト。これには、ロギング可能なすべてのエラーおよび cfcatch 変数の値を含めることができます。
- メッセージの厳格度 (タイプ)。Information、Warning、Fatal、Error のいずれか。
- アプリケーション名、スレッド ID、システムの日付または時刻をロギングするかどうか。デフォルトでは、すべての情報がロギングされます。

たとえば、次のページのように、例外エラー処理ページで cflog タグを使用して、アプリケーション固有のログファイルにエラー情報をロギングすることができます。

```
 <html>
<head>
<title>Products - Error</title>
</head>
<body>

<h2>Sorry</h2>

<p>An error occurred when you requested this page.
The error has been logged and we will work to correct the problem.
We apologize for the inconvenience. </p>

<cflog type="Error"
        file="myapp_errors"
        text="Exception error --
            Exception type: #error.type#
            Template: #error.template#,
            Remote Address: #error.remoteAddress#,
            HTTP Reference: #error.HTTPReferer#
            Diagnostics: #error.diagnostics#">

</body>
</html>
```

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明

`<cflog type="Error"`
`file="myapp_errors"`
`text="Exception error --`
`Exception type: #error.type#`
`Template: #error.template#,`
`Remote Address: #error.remoteAddress#,`
`HTTP Reference: #error.HTTPReferer#`
`Diagnostics: #error.diagnostics#">`
	このページが処理されると、ColdFusion ログディレクトリ内の "myapp_errors.log" ファイルにエントリがロギングされます。このエントリはエラーメッセージとして識別され、例外タイプ、エラーが発生したページのパス、そのページを呼び出したリモートアドレス、およびこのエラーの診断メッセージを含むエラーメッセージが記述されます。

実在しないカスタムタグを呼び出して、このページでエラーが検出された場合は、次のようなログファイルエントリが生成されます (わかりやすくするために改行を追加しています)。

```
"Error","web-13","12/19/01","11:29:07",MYAPP,"Exception error --
        Exception type: coldfusion.runtime.CfErrorWrapper
        Template: /MYStuff/MyDocs/exceptiontest.cfm,
        Remote Address: 127.0.0.1,
        HTTP Reference:
        Diagnostics: Cannot find CFML template for custom tag testCase. Cannot
        find CFML template for custom tag testCase. ColdFusion attempted looking
        in the tree of installed custom tags but did not find a custom tag with
        this name."
```

このテキストには、カンマで区切られた、次のエントリのリストが含まれています。

- cflog の type 属性によって指定されたログエントリのタイプ
- 実行中だったスレッドの ID
- エントリがロギングされた日付
- エントリがロギングされた時刻

- "Application.cfc" の初期化コードで This.application 変数を使用して指定されているアプリケーション名、または、cfapplication タグ (たとえば "Application.cfm" ファイルで設定) によって指定されているアプリケーション名
- cflog の text 属性によって指定されたメッセージ

ColdFusion タグでのランタイム例外の処理

例外は、データベース操作の失敗、インクルードファイルの欠如、開発者指定のイベントなど、ColdFusion ページ内の通常の命令の流れを中断するイベントです。ColdFusion では通常、例外が発生すると処理が中断し、エラーメッセージまたはエラーページ (cferror タグ、または、Administrator の [設定] ページの [サイト全体のエラーハンドラ] オプションで指定したもの) が表示されます。ただし、ColdFusion の例外処理タグを使用すると、ランタイム例外を ColdFusion ページで直接検出して処理することができます。

アプリケーションページで直接例外を処理する機能を使用すれば、次のことが行えます。

- 現在のアプリケーションページのコンテキスト内で、特定のエラーに適した応答が行えます。
- 可能な場合にはエラーから回復できます。

例外処理タグ

ColdFusion では、次の表に示す例外処理タグが使用できます。

タグ	説明
cftry	タグ本文の処理時に例外が発生した場合は、例外処理用の cfcatch タグを探し、cfcatch タグ本文のコードを実行します。
cfcatch	cftry タグ本文のコードで発生した例外が、このタグの属性で指定されている例外タイプに一致する場合、このタグ本文のコードを実行します。 cftry タグ本文でのみ使用します。
cfthrow	ユーザー指定の例外を生成します。
cfrethrow	現在の cfcatch ブロックを終了し、同じタイプの新しい例外を生成します。 cfcatch タグ本文でのみ使用します。

cftry タグおよび cfcatch タグの使用

cftry タグを使用すると、ユーザーにエラーデータをレポートする以外に、次のことを実行できます。

- ユーザーに警告を出さずに処理を継続できるように、エラーから回復するコードを含めることができます。
- エラーを発生させる特定のコードに対して、カスタマイズされたエラーメッセージを作成できます。

たとえば、cftry を使用して、ユーザー登録フォームからデータベースにデータを入力するコード内のエラーを検出できます。cfcatch コードでは、次のことを行えます。

1. リソースが単に一時的に使用できなかった場合に、クエリーを再度実行して処理を継続します。
2. 再試行に失敗した場合は、次のことを行います。
   - ユーザーにカスタムメッセージを表示します。
   - データを電子メールアドレスに送信し、問題解決後に社内の担当者によってデータを入力できるようにします。

リソースの可用性が保証されていないデータベース、ファイル、LDAP サーバーなどの外部リソースにアクセスするコードは、Try/Catch ブロックの使用に適しています。

try/catch コードの構造

コードで発生した例外を直接処理したい場合は、cftry ブロックにそのコードを含める必要があります。したがって、アプリケーションページ全体を cftry ブロックで囲むのは有力な方法です。cftry ブロックの後には cfcatch ブロックを配置します。このブロックは、潜在的なエラーに対応します。cftry ブロック内で例外が発生すると、その例外のタイプに対応する cfcatch ブロックに処理が渡されます。

cftry および cfcatch を使用してエラー処理を行うためのアウトラインを次に示します。

```
<cftry>
    Put your application code here ...
    <cfcatch type="exception type1">
        Add exception processing code here ...
    </cfcatch>
    <cfcatch type="exception type2">
        Add exception processing code here ...
    </cfcatch>
    ...
    <cfcatch type="Any">
        Add exception processing code appropriate for all other exceptions here ...
    </cfcatch>
</cftry>
```

try/catch コードのルールおよび注意事項

cftry タグおよび cfcatch タグを使用する場合は、次のルールおよび注意事項に従ってください。

- cfcatch タグは、cftry タグ本文内の他のすべてのコードの後ろに配置します。
- cftry ブロックはネストすることができます。たとえば、次の構造は有効です。

```
<cftry>
    code that may cause an exception
    <cfcatch ...>
        <cftry>
            First level of exception handling code
            <cfcatch ...>
                Second level of exception handling code
            </cfcatch
        </cftry>
    </cfcatch>
</cftry>
```

  最初のレベルの例外処理コードで例外が発生した場合は、内部にある cfcatch ブロックで例外が検出され、処理されます。cfcatch ブロック内の例外は、そのブロックと同じレベルにある cfcatch ブロックでは処理できません。

- ColdFusion では常に、最後に発生した例外が処理されます。たとえば、cftry ブロックで例外が発生して cfcatch ブロックに処理が渡され、その cfcatch ブロックで例外が発生したがそれを処理するハンドラがない場合は、cfcatch ブロックの例外に対してデフォルトのエラーメッセージが表示されます。最初の例外に関する情報は表示されません。
- 現在のタグが他のタグの中にネストされている場合に例外が発生すると、適当な cftry/cfcatch の組み合わせが見つかるまで、またはスタックの終わりに到達するまで、CFML プロセッサによって開始タグのスタック全体が確認されます。
- cftry と cfcatch は、アプリケーションページ内の例外発生場所または診断情報に基づいて例外を処理する場合に使用します。
- cfcatch タグ全体 (対応するすべての cftry タグも含む) は、単一の ColdFusion ページに配置する必要があります。<cftry> 開始タグと </cftry> 終了タグを別々のページに配置することはできません。
- cfcatch ブロックで正常にエラーを処理できない場合は、295 ページの「cfrethrow タグの使用」で説明している cfrethrow タグの使用を検討してください。
- 無視しても安全な例外である場合は、次のように、本文を持たない cfcatch タグを使用します。

```
<cfcatch Type = Database />
```

- 問題がある場合は、例外が発生しやすいタグを、cftry と cfcatch の特殊な組み合わせで囲み、タグの例外をすぐに切り離します。

cfcatch ブロック内の例外情報

cfcatch タグの本文内では、cfcatch オブジェクトを使用して、アクティブな例外のプロパティにアクセスできます。次に、このオブジェクトの内容について説明します。

標準の cfcatch 変数

次の表で、ほとんどの cfcatch ブロック内で利用できる変数について説明します。

プロパティ変数	説明
cfcatch.Detail	CFML コンパイラからの詳細メッセージ。このメッセージには HTML 形式が含まれることがあり、どのタグが例外を返したかを調べるために利用できます。 CFScript の catch ステートメントでも、exceptionVariable パラメータを使用して cfcatch.Detail の値にアクセスできます。
cfcatch.ErrorCode	cfthrow タグの errorCode 属性によってこのコード値が設定されることがあります。Type="Database" の場合、cfcatch.ErrorCode は、cfcatch.SQLState と同じ値を持ちます。 それ以外の場合、cfcatch.ErrorCode の値は空の文字列になります。
cfcatch.ExtendedInfo	カスタムのエラーメッセージ情報。これは、type 属性が Application またはカスタムタイプである cfcatch タグにのみ返されます。 それ以外の場合、cfcatch.ExtendedInfo の値は空の文字列になります。
cfcatch.Message	例外のデフォルトの診断メッセージ (診断メッセージが設定されている場合)。使用できる診断メッセージがなければ、この変数は空の文字列になります。 CFScript の catch ステートメントでも、exceptionVariable パラメータの値に cfcatch.Message の値が含まれています。
cfcatch.RootCause	例外の " 根本原因 " として JVM によってレポートされる、Java サーブレットの例外。
cfcatch.TagContext	タグスタック内の各タグの情報が含まれている構造体の配列。タグスタックは、現在開いている各タグで構成されます。
cfcatch.Type	文字列として返される例外のタイプ。

注意：cfdump タグを使用して cfcatch 変数を表示する場合、値を持たない変数は表示されません。

cfcatch.TagContext 変数には、タグ情報構造体の配列が含まれています。各構造体は、ColdFusion で例外が検出されたときのアクティブタグコンテキストの 1 レベルを表しています。つまり、例外発生時に開いている各タグについて、1 つの構造体があります。たとえば、カスタムタグページ上のタグで例外が発生した場合、タグコンテキストには、呼び出されたカスタムタグおよびエラーが発生したタグについての情報が表示されます。

配列の位置 1 にある構造体は、例外が検出されたときに実行されていたタグを表します。ArrayLen の位置にある構造体は、コンパイラによって例外が検出されたときに実行されていたタグスタック内の最初のタグを表します。

次の表に、tagContext 構造体の属性をリストします。

エントリ	説明
Column	廃止 (下位互換性のために維持)。常に 0 です。
ID	その中で例外が発生しているタグ。CFScript 内の例外は、2 つの疑問符 (??) によって示されます。直接呼び出されるものを含め、すべてのカスタムタグは、CFMODULE として識別されます。
Line	そのタグが配置されているページ上の行。

エントリ	説明
Raw_Trace	そのエラーの Java スタックトレース。
Template	このタグが含まれているアプリケーションページのパス名。
Type	ページのタイプ。常に ColdFusion ページです。

データベースの例外

例外タイプがデータベースである場合は、次の追加の変数が常に使用できます。

プロパティ変数	説明
cfcatch.NativeErrorCode	この例外に関連付けられているネイティブエラーコード。データベースドライバによって返されるエラーコードは、失敗したデータベース操作の診断に役立ちます。cfcatch.NativeErrorCode の値は、ドライバによって異なります。 エラーコードが指定されていない場合、cfcatch.nativeErrorCode の値は -1 になります。クエリーオブクエリーの値は 0 になります。
cfcatch.SQLState	この例外に関連付けられている SQLState コード。データベースドライバによって返されるエラーコードは、失敗したデータベース操作の診断に役立ちます。SQLState コードは、ネイティブエラーコードと比較すると、データベースシステム全体でより一貫したものになっています。 ドライバによって SQLState 値が与えられない場合、cfcatch.SQLState の値は -1 になります。
cfcatch.Sql	データソースに送信された SQL ステートメント。
cfcatch.queryError	データベースドライバによってレポートされたエラーメッセージ。
cfcatch.where	クエリーで cfqueryparam タグを使用する場合は、クエリーパラメータの名前 / 値のペアです。

式の例外

次の変数は、式の例外に対してのみ使用できます。

プロパティ変数	説明
cfcatch.ErrNumber	式の内部エラー番号。type="Expression" の場合にのみ有効です。

ロック例外

cflock タグのエラーに関連する例外では、次の追加情報が使用できます。

プロパティ変数	説明
cfcatch.lockName	問題のロックの名前。ロックの名前が判明していない場合は "anonymous" に設定されます。
cfcatch.lockOperation	失敗した操作。失敗した操作が判明していない場合は "unknown" に設定されます。

インクルードファイル欠如の例外

cfinclude タグで指定されているファイルが欠如しているためにエラーが発生した場合は、次の追加の変数が使用できます。

プロパティ変数	説明
cfcatch.missingFileName	欠如ファイルの名前

cftry タグの使用 : 例

次の例では、cftry タグと cfcatch タグの使用方法を示します。この例では、ここにリストしたサンプルの多くで使用されている cfdocexamples データソースと、サンプルのインクルードファイル "includeme.cfm" を使用します。

cfquery ステートメントの実行中に例外が発生すると、cfcatchtype="Database" 例外ハンドラにアプリケーションのページフローが切り替わります。cfcatchtype="Database" ハンドラが終了すると、cftry ブロックの後ろにある次のステートメントから処理が再開されます。同様に、cfcatchtype="MissingInclude" ブロックは、cfinclude タグによって発生した例外を処理します。

```
 <!--- Wrap code you want to check in a cftry block --->
<cfset EmpID=3>
<cfparam name="errorCaught" default="">
<cftry>
    <cfquery name="test" datasource="cfdocexamples">
        SELECT Dept_ID, FirstName, LastName
        FROM Employee
        WHERE Emp_ID=#EmpID#
    </cfquery>

    <html>
    <head>
    <title>Test cftry/cfcatch</title>
    </head>
    <body>
    <cfinclude template="includeme.cfm">
    <cfoutput query="test">
        <p>Department: #Dept_ID#<br>
        Last Name: #LastName#<br>
        First Name: #FirstName#</p>
    </cfoutput>

<!--- Use cfcatch to test for missing included files. --->
<!--- Print Message and Detail error messages. --->
<!--- Block executes only if a MissingInclude exception is thrown. --->
    <cfcatch type="MissingInclude">
        <h1>Missing Include File</h1>
        <cfoutput>
        <ul>
            <li><b>Message:</b> #cfcatch.Message#
            <li><b>Detail:</b> #cfcatch.Detail#
            <li><b>Filename:</b> #cfcatch.MissingFileName#
        </ul>
        </cfoutput>
        <cfset errorCaught = "MissingInclude">
    </cfcatch>

<!--- Use cfcatch to test for database errors.--->
<!--- Print error messages. --->
<!--- Block executes only if a Database exception is thrown. --->
    <cfcatch type="Database">
        <h1>Database Error</h1>
        <cfoutput>
        <ul>
            <li><b>Message:</b> #cfcatch.Message#
            <li><b>Native error code:</b> #cfcatch.NativeErrorCode#
            <li><b>SQLState:</b> #cfcatch.SQLState#
            <li><b>Detail:</b> #cfcatch.Detail#
            </ul>
```

```
        </cfoutput>
        <cfset errorCaught = "Database">
    </cfcatch>

<!--- Use cfcatch with type="Any" --->
<!--- to find unexpected exceptions. --->
    <cfcatch type="Any">
        <cfoutput>
            <hr>
            <h1>Other Error: #cfcatch.Type#</h1>
            <ul>
                <li><b>Message:</b> #cfcatch.Message#
                <li><b>Detail:</b> #cfcatch.Detail#
            </ul>
        </cfoutput>
        <cfset errorCaught = "General Exception">
    </cfcatch>
</body>
</html>
</cftry>
```

このコードをテストするには、次の手順を行います。

コードのテスト

1. "includeme.cfm" ファイルがないことを確認して、ページを表示します。cfcatch type="MissingInclude" ブロックにより、エラーが表示されます。
2. 空でない "includeme.cfm" ファイルを作成してページを表示します。データベースが正しく構成されていれば従業員のエントリが表示され、エラーは発生しません。
3. cfquery タグで、次の行を

   ```
   FROM Employee
   ```

   次のコードに変更します。

   ```
   FROM Employer
   ```

   ページを表示します。すると、cfcatch type="Database" ブロックによってエラーメッセージが表示されます。
4. Employer を Employee に変更します。

   cfoutput の次の行を

   ```
   <p>Department: #Dept_ID#<br>
   ```

   次のコードに変更します。

   ```
   <p>Department: #DepartmentID#<br>
   ```

   ページを表示します。すると、cfcatch type="Any" ブロックによって式のエラーを示すエラーメッセージが表示されます。
5. DepartmentID を Dept_ID に戻して、ページを再度表示します。ページは正しく表示されます。

   テキストエディタで ¥CFusion¥Log¥MyAppPage.log を開きます。次のような、ヘッダ行、初期化行、および 4 つの詳細行が表示されます。

```
 "Severity","ThreadID","Date","Time","Application","Message"
"Information","web-0","11/20/01", "16:27:08",, "cf_root\runtime\servers\default\logs\ MyAppPage.log
initialized"
"Information","web-0","11/20/01","16:27:08",,
    "Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: MissingInclude"
"Information","web-1","11/20/01","16:27:32",,"
    Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: "
"Information","web-0","11/20/01","16:27:49",,
    "Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: Database"
"Information","web-1","11/20/01","16:28:21",,
    "Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: General Exception"
"Information","web-0","11/20/01","16:28:49",,
    "Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: "
```

コードの説明

このコードについて、次の表で説明します。

コード	説明
`<cfset EmpID=3>` `<cfparam name="errorCaught" default="">`	従業員 ID を有効な値に初期化します。実際のアプリケーションでは、フォームまたは他のソースからこの値を取得します。 デフォルトの errorCaught 変数の値を空の文字列 (エラーが検出されなかったことを示す) に設定します。 これらの行を cftry ブロックに入れる必要はありません。

`<cftry>`
`<cfquery name="test" datasource="cfdocexamples">`
`SELECT Dept_ID, FirstName, LastName`
`FROM Employee`
`WHERE Emp_ID=#EmpID#`
`</cfquery>`
	cftry ブロックを開始します。ここからブロックの終わりまでの間の例外を、cfcatch タグによって検出できます。
	cfdocexamples データベースにクエリーを実行し、EmpID 変数によって識別される従業員のデータを取得します。

`<html>` `<head>` `<title>Test cftry/cfcatch</title>` `</head>` `<body>` `<cfinclude template="includeme.cfm">` `<cfoutput query="test">` `<p>Department: #Dept_ID# ` `Last Name: #LastName# ` `First Name: #FirstName#</p>` `</cfoutput>`	HTML ページを開始します。このセクションには、エラーが発生しない場合に情報を表示するコードがすべて含まれています。 "includeme.cfm" ページをインクルードします。 test クエリーから得たユーザー情報のレコードを表示します。

`<cfcatch type="MissingInclude">`
`<h1>Missing Include File</h1>`
`<cfoutput>`
`<ul>`
`<li><b>Message:</b> #cfcatch.Message#`
`<li><b>Detail:</b> #cfcatch.Detail#`
`<li><b>Filename:</b> #cfcatch.MissingFileName#`
`</ul>`
`</cfoutput>`
`<cfset errorCaught = "MissingInclude">`
`</cfcatch>`
	cfinclude タグで指定したページが見つからない場合に発生した例外を処理します。
	cfcatch 変数を使用して、ColdFusion の基本的なエラーメッセージ、詳細メッセージ、および見つからなかったファイルの名前を表示します。
	errorCaught 変数をエラータイプの値に設定します。

コード	説明
<cfcatch type="Database"> Database Error <cfoutput> Message: #cfcatch.Message# Native error code: #cfcatch.NativeErrorCode# SQLState: #cfcatch.SQLState# Detail: #cfcatch.Detail# </cfoutput> <cfset errorCaught = "Database"> </cfcatch>	データベースへのアクセスで発生した例外を処理します。 cfcatch 変数を使用して、ColdFusion の基本的なエラーメッセージ、データベースシステムによってレポートされるエラーコードと SQL ステート、および詳細エラーメッセージを表示します。 errorCaught 変数をエラータイプの値に設定します。
<cfcatch type="Any"> <cfoutput> <hr> Other Error: #cfcatch.Type# Message: #cfcatch.Message# Detail: #cfcatch.Detail# </cfoutput> <cfset errorCaught = "General Exception"> </cfcatch>	cftry ブロックで発生したその他の例外を処理します。 情報 (この場合は、インクルードファイルの内容) の表示後にエラーが発生することがあるので、メッセージテキストを記述する前に区切りを付けます。 ColdFusion の基本メッセージおよび詳細エラーメッセージを表示します。 errorCaught 変数をエラータイプの値に設定します。
</body> </html> </cftry>	HTML ページを終了し、cftry ブロックを終了します。

cfthrow タグの使用

cfthrow タグを使用して、独自のカスタム例外を発生させることができます。cfthrow タグを使用する場合は、次の情報のいずれかまたはすべてを指定します。

属性	説明
type	エラーのタイプ。これは、InvalidProductCode など、そのアプリケーションでのみ意味を持つカスタムタイプにすることができます。デフォルトのタイプである Application を指定することもできます。Database や MissingTemplate などのあらかじめ定義されている ColdFusion エラータイプを使用することはできません。
message	エラーを示す簡単なテキストメッセージ。
detail	エラーを説明するより詳細なテキストメッセージ。
errorCode	アプリケーション固有のエラーコード。このフィールドは、アプリケーションで数値のエラーコードを使用する場合に役立ちます。
extendedInfo	アプリケーション向けの、使用方法についての追加情報。

これらの値はすべてオプションです。cfcatch ブロック内の属性値および例外タイプのエラーページにアクセスするには、cfcatch.extendedInfo のように、属性の前に cfcatch または error を置きます。デフォルトの ColdFusion エラーハンドラによって、[メッセージ] ペインにメッセージと詳細値が表示され、その他の値は [エラー診断情報] ペインに表示されます。

発生させたエラーの検出と表示

cfcatch タグでカスタム例外を検出するには、cfcatch の type 属性に次のいずれかの値を使用します。

- cfthrow タグで指定したカスタム例外タイプ。
- cfthrow タグで指定したタイプの最初の部分に階層的に一致するカスタム例外タイプ。詳細については、カスタムエラータイプ名の階層を参照してください。
- Application。type 属性が Application の例外、または type 属性が指定されていない例外が検出されます。
- Any。特定のタイプ用の cfcatch タグによって検出されなかった例外がすべて検出されます。

同様に、cferror タグでこれらのタイプを指定すると、発生したエラーに関する情報が、指定したエラーページに表示されます。

cfthrow タグによって例外が生成されるため、リクエストエラーハンドラまたはサイト全体のエラーハンドラでもこれらのエラーを表示できます。

カスタムエラータイプ名の階層

カスタム例外のタイプには、Java クラスのネーミング規則に似た方法で名前を付けることができます。つまり、ドメイン名を逆順にして、その後にプロジェクト識別子を置きます。次に例を示します。

```
<cfthrow
    type="com.myCompany.myApp.Invalid_field.codeValue"
    errorcode="Dodge14B">
```

この完全修飾ネーミング規則は必ずしも必要ではありません。たとえば、myApp.Invalid_field.codeValue または codeValue のように、より短いネーミング規則を使用することもできます。

このネーミング規則には規則以上の意味があります。ColdFusion ではこの名前階層を利用して、考えられるエラーハンドラの階層を選択します。たとえば、次の cfthrow ステートメントを使用すると想定します。

```
<cfthrow type="MyApp.BusinessRuleException.InvalidAccount">
```

このエラーは、次のいずれの cfcatch エラーハンドラでも処理されます。

```
<cfcatch type="MyApp.BusinessRuleException.InvalidAccount">
<cfcatch type="MyApp.BusinessRuleException">
<cfcatch type="MyApp">
```

このエラーは、最も正確に一致するハンドラによって処理されます。この場合は、MyApp.BusinessRuleException.InvalidAccount ハンドラが実行されます。ただし、次の cfthrow タグを使用する場合は、

```
<cfthrow type="MyApp.BusinessRuleException.InvalidVendorCode
```

MyApp.BusinessRuleException ハンドラがエラーを受け取ります。

タイプの比較では、大文字と小文字は区別されません。

cfthrow タグが有用な場合

cfthrow タグは、アプリケーションがアプリケーション固有のエラーを識別して処理できる場合に使用します。cfthrow タグの典型的な使用例として、カスタムデータ検証の実装があります。また、カスタムタグページから呼び出しページにエラーを投げる場合にも cfthrow タグが役立ちます。

たとえば、パスワードを設定するためのフォームアクションページまたはカスタムタグでは、入力されたパスワードが最低文字数を満たしているか、文字と数字の両方を含んでいるかなどを判断し、違反したパスワードルールを示すメッセージとともにエラーを投げます。cfcatch ブロックではそのエラーを処理し、ユーザーに問題の訂正方法を指示します。

cfrethrow タグの使用

cfrethrow タグによって、エラーハンドラの階層を作成できます。このタグは、現在の cfcatch ブロックを終了し、次のレベルのエラーハンドラに例外を " 再度投げる (rethrow する)" ように ColdFusion に指示します。このように、特定のエラータイプ向けのエラーハンドラでエラーが解決できない場合は、より汎用性のあるエラーハンドラに対して再度エラーを投げることができます。cfrethrow タグは cfcatch タグ本文内でのみ使用できます。

cfrethrow タグのシンタックス

次の擬似コードでは、cfrethrow タグを使用してエラー処理階層を作成する方法を示します。

```
<cftry>
    <cftry>
        Code that might throw a database error
        <cfcatch Type="Database">
            <cfif Error is of type I can Handle>
                Handle it
            <cfelse>
                <cfrethrow>
            </cfif
        </cfcatch>
    </cftry>
    <cfcatch Type="Any">
        General Error Handling code
    </cfcatch>
</cftry>
```

この例ではデータベースエラーを例に挙げていますが、最も内側のエラータイプには任意の cfcatch タイプ属性を使用できます。

cfrethrow タグを使用する場合は、次のルールに従ってください。

- エラー処理階層のレベルごとに 1 つのタグを使用して、cftry タグをネストします。各レベルでは、そのエラー詳細レベルに対する cfcatch タグを記述します。
- 最も汎用的なエラー検出コードを、最も外側の cftry ブロックに配置します。
- 最も限定的なエラー検出コードを、最も内側の cftry ブロックに配置します。
- 例外エラーが発生する可能性のあるコードを、最も内側の cftry ブロックの冒頭に配置します。
- 最も外側の cftry ブロックを除き、各 cfcatch ブロックの末尾に cfrethrow タグを配置します。

例 : ネストされたタグ、cfthrow、および cfrethrow の使用

次の例では、ネストされた cftry ブロックや cfthrow および cfrethrow タグなど、これまでに説明したテクニックの多くを示します。この例には、簡単な呼び出しページおよびカスタムタグページが含まれています。

- 呼び出しページでは、データベース内で検索する名前を示す 1 つの属性を渡してカスタムタグを呼び出すとともに、カスタムタグから例外が投げられた場合にはその処理も行います。
- カスタムタグでは、cfdocexamples データベース内のすべてのレコードを姓で照合して検出し、Caller 変数に結果を返します。メインデータベースへの接続に失敗した場合は、バックアップデータベースへの接続を試みます。

呼び出しページ

この呼び出しページは、より大きなアプリケーションページの一部です。単純にするため、検索する名前はハードコードしています。

```
<cftry>
    <cf_getEmps EmpName="Jones">
    <cfcatch type="myApp.getUser.noEmpName">
        <h2>Oops</h2>
        <cfoutput>#cfcatch.Message#</cfoutput><br>
    </cfcatch>
</cftry>
<cfif isdefined("getEmpsResult")>
    <cfdump var="#getEmpsResult#">
</cfif>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明
<cftry> <cf_getEmps EmpName="Jones">	cftry ブロックで、cf_getEmps カスタムタグ ("getEmps.cfm") を呼び出します。
<cfcatch type="myApp.getUser.noEmpName"> Oops <cfoutput>#cfcatch.Message#</cfoutput> </cfcatch>	受け取った属性が有効でないことを示す例外がタグから投げられた場合は、その例外を検出し、カスタムタグの cfthrow タグで設定されたメッセージ変数を使用してメッセージを表示します。
<cfif isdefined("getEmpsResult")> <cfdump var="#getEmpsResult#"> </cfif>	タグから結果が返された場合は、cfdump タグを使用してこれを表示します。本番用のアプリケーションでは cfdump タグは使用しません。

カスタムタグページ

カスタムタグページでは、指定の名前をデータベースで検索し、一致したレコードを呼び出しページの getEmpsResult 変数に返します。このページでは、エラー条件を処理するために、複数の cftry ブロックをネストしています。詳細については、例の後にある「コードの説明」を参照してください。

次のコードに "getEmps.cfm" という名前を付けて、呼び出しページと同じディレクトリに保存します。

```
 <!--- If the tag didn't pass an attribute, throw an error to be handled by
            the calling page --->
<cfif NOT IsDefined("attributes.EmpName")>
    <cfthrow Type="myApp.getUser.noEmpName"
        message = "Last Name was not supplied to the cf_getEmps tag.">
    <cfexit method = "exittag">
<!--- Have a name to look up --->
<cfelse>
<!--- Outermost Try Block --->
    <cftry>

<!--- Inner Try Block --->
        <cftry>
<!--- Try to query the main database and set a caller variable to the result --->
            <cfquery Name = "getUser" DataSource="cfdocexamples">
                SELECT *
                FROM Employee
                WHERE LastName = '#attributes.EmpName#'
            </cfquery>
            <cfset caller.getEmpsResult = getuser>
<!--- If the query failed with a database error, check the error type
            to see if the database was found --->
            <cfcatch type= "Database">
                <cfif (cfcatch.SQLState IS "S100") OR (cfcatch.SQLState IS
                    "IM002")>

<!--- If the database wasn't found, try the backup database --->
<!--- Use a third-level Try block --->
                    <cftry>
                        <cfquery Name = "getUser" DataSource="cfdocexamplesBackup">
                            SELECT *
                            FROM Employee
                            WHERE LastName = '#attributes.EmpName#'
                        </cfquery>
                            <cfset caller.getEmpsResult = getuser>

<!--- If still get a database error, just return to the calling page
            without setting the caller variable. There is no cfcatch body.
            This might not be appropriate in some cases.
            The Calling page ends up handling this case as if a match was not
            found --->
                            <cfcatch type = "Database" />
```

```
<!--- Still in innermost try block. Rethrow any other errors to the next
            try block level --->
                        <cfcatch type = "Any">
                            <cfrethrow>
                        </cfcatch>
                    </cftry>

<!--- Now in second level try block.
            Throw all other types of Database exceptions to the next try
                block level --->
                <cfelse>
                    <cfrethrow>
                </cfif>
            </cfcatch>
<!--- Throw all other exceptions to the next try block level --->
            <cfcatch type = "Any">
                    <cfrethrow>
            </cfcatch>
        </cftry>

<!--- Now in Outermost try block.
         Handle all unhandled exceptions, including rethrown exceptions, by
            displaying a message and exiting to the calling page.--->
        <cfcatch Type = "Any">
            <h2>Sorry</h2>
            <p>An unexpected error happened in processing your user inquiry.
                Please report the following to technical support:</p>
            <cfoutput>
                Type: #cfcatch.Type#
                Message: #cfcatch.Message#
            </cfoutput>
            <cfexit method = "exittag">
        </cfcatch>
    </cftry>
</cfif>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明

`<cfif NOT IsDefined("attributes.EmpName")>`
`<cfthrow Type="myApp.getUser.noEmpName"`
`message = "Last Name was not supplied to the cf_getEmps tag.">`
`<cfexit method = "exittag">`
	呼び出しページで EmpName 属性が指定されたかどうか確認します。指定されていない場合は、問題を示すカスタムエラーを投げ、このタグを終了します。呼び出しページでは、投げられたエラーを処理します。

`<cfelse>` `<cftry>`	このタグに EmpName 属性がある場合は、最も外側の try ブロックで残りの処理を実行します。その末尾にある cfcatch ブロックでは、他のいずれの方法でも検出されなかった例外を処理します。

`<cftry>`
`<!--- Try to query the main database and set a caller variable to the result --->`
`<cfquery Name = "getUser" DataSource="cfdocexamples">`
`SELECT *`
`FROM Employee`
`WHERE LastName = '#attributes.EmpName#'`
`</cfquery>`
`<cfset caller.getEmpsResult = getuser>`
	2 番目のネストされた try ブロックを開始します。このブロックは、データベースクエリーでの例外を検出します。
	例外がない場合は、呼び出しページの getEmpsResult 変数にクエリーの結果を設定します。

コード	説明
<cfcatch type= "Database"> <cfif (cfcatch.SQLState IS "S100") OR (cfcatch.SQLState IS "IM002")> <cftry> <cfquery Name = "getUser" DataSource="cfdocexamplesBackup"> SELECT * FROM Employee WHERE LastName = '#attributes.EmpName#' </cfquery> <cfset caller.getEmpsResult = getuser>	クエリーによってデータベースエラーが返された場合は、データベースにアクセスできなかったことがエラーの原因であるかどうかを確認します (SQLState 変数の S100 または IM002 の値によって示されます)。 データベースが検出されなかった場合は、ネストされている 3 番目の try ブロックを開始し、バックアップデータベースへのアクセスを試行します。この try ブロックでは、この 2 番目のデータベースへのアクセスによる例外を検出します。 データベースの問い合わせが正常に行われた場合は、呼び出しページの getEmpsResult 変数にクエリーの結果を設定します。
<cfcatch type = "Database" />	2 番目のデータベースクエリーがデータベースエラーで失敗した場合は、何も表示せずに終了します。このデータベースタイプの cfcatch タグには本文がないので、そのまま終了します。呼び出しページは getEmpsResult 変数を得ることができません。データベースに一致が検出されなかったのか、回復不可能なデータベースエラーが発生したのかはわかりませんが、一致が検出されなかったことは判明しています。
<cfcatch type = "Any"> <cfrethrow> </cfcatch> </cftry>	その他の理由で 2 番目のデータベースクエリーが失敗した場合は、次の try ブロックにエラーを投げます。 最も内側にある try ブロックを終了します。
<cfelse> <cfrethrow> </cfif> </cfcatch>	2 番目の try ブロックでは、データベースを検出できなかった以外の理由で失敗した最初のデータベースクエリーを処理します。 次のレベルである最も外側の try ブロックにエラーを再度投げます。
<cfcatch type = "Any"> <cfrethrow> </cfcatch> </cftry>	2 番目の try ブロックで、その他の例外を検出し、それらを最も外側の try ブロックに再度投げます。 2 番目の try ブロックを終了します。
<cfcatch Type = "Any"> Sorry An unexpected error happened in processing your user inquiry. Please report the following to technical support: <cfoutput> Type: #cfcatch.Type# Message: #cfcatch.Message# </cfoutput> <cfexit method = "exittag"> </cfcatch> </cftry> </cfif>	最も外側の try ブロックで、例外タイプおよび例外のエラーメッセージを含んでいるエラーメッセージを表示することによって、例外を処理します。ネストされた try ブロックに含まれているコードを除き、このブロックでは何も実行していないので、この cfcatch タグでは、ネストされたブロックから再度投げられたエラーのみが処理されます。 カスタムタグを終了し、呼び出しページに戻ります。 catch ブロック、try ブロック、および最初の cfif ブロックを終了します。

コードのテスト

この例で、エラーが発生して処理されるさまざまな状況をテストするには、次の操作を行います。

- 呼び出しページで、属性名を My Attrib などの他の値に変更します。その後、元に戻します。
- 最初の cfquery タグで、データソース名を NoDatabase などの無効なデータソース名に変更します。
- 最初のデータソース名を無効なものにしたまま、2 番目の cfquery タグのデータソースを cfdocexamples に変更します。
- カスタム例外を投げる cfthrow タグをコード内のさまざまな場所に挿入して、その結果を確認します。

永続データとロックの使用

Adobe ColdFusion には、複数のリクエストにまたがってデータを保持できる変数スコープがいくつか用意されています。その変数スコープには、Client、Application、Session、および Server スコープがあります。これらのスコープを使用すると、永続的にデータを保存して、複数のページやアプリケーションでデータを共有することができます。これらのスコープを永続スコープとして使用します。特に、Client スコープと Session スコープを使用して、複数のリクエストにまたがってユーザー情報を保持します。

ColdFusion では、特定のコードセクションへのアクセスをロックすることで、同時にまたは予期しない順序でコードが実行されたり、使用するデータがアクセスされたりするのを防止できます。このロック機能は、永続スコープ内のデータだけでなく、外部ソースのデータを含むすべての共有データの一貫性を維持するために重要です。

永続スコープを使用してアプリケーションを開発し、ロックを使用してデータの一貫性を維持できます。

永続スコープ変数について

次の表に示す 4 つの変数スコープを使用すれば、複数のアプリケーションやユーザーにまたがって使用する必要があるデータや、現在のリクエストのスコープを超えて維持する必要のあるデータを保持できます。

変数スコープ	説明
Client	1 つのアプリケーション内の、単一のクライアントブラウザに対する複数のブラウザセッションで使用できる変数が含まれています。ブラウザセッションについては、309 ページの「セッションとは」を参照してください。 クライアントの環境設定など、長期間にわたって保管するクライアント固有の情報を格納するのに便利です。 データは Cookie、データベースエントリ、またはレジストリ値として保管されます。クライアント変数は、タイムアウト期間が経過したら無効化することができます。 変数名に Client スコープ接頭辞を使用する必要はありませんが、接頭辞を使用すると効率が向上し、メンテナンスも行いやすくなります。
Session	1 つのアプリケーション内の、単一のクライアントブラウザに対する単一のブラウザセッションで使用できる変数が含まれています。 ショッピングカートの内容など、クライアントがアプリケーションを使用している間保持するクライアント固有の情報を格納するのに便利です。 データはメモリに保管され、アクティブでない状態が一定時間続くか、またはサーバーがシャットダウンされるとタイムアウトになります。 ColdFusion Administrator では、標準の ColdFusion セッション管理と J2EE セッション管理の 2 種類のセッション管理から選択できます。セッション管理のタイプについては、310 ページの「ColdFusion セッション管理と J2EE セッション管理」を参照してください。 変数名には Session スコープ接頭辞を使用します。
Application	1 つのアプリケーション内の、すべてのクライアントに対するすべてのページで使用できる変数が含まれています。 連絡先情報など、時間とともに変化する可能性があるために変数内に保管する必要があるアプリケーション固有の情報を格納するのに便利です。 データはメモリに保管され、アクティブでない状態が一定時間続くか、またはサーバーがシャットダウンされるとタイムアウトになります。 変数名には Application スコープ接頭辞を使用します。
Server	1 つのサーバー上にあるすべてのアプリケーションと、すべてのクライアントで使用できる変数が含まれています。 ページヒットカウンタの値など、サーバーのすべてのページに適用される情報を格納するのに便利です。 データはメモリに保管されます。この変数がタイムアウトになることはありませんが、作成した変数は削除できます。また、サーバーの実行が停止されるとすべてのサーバー変数は自動的に削除されます。 変数名には Server スコープ接頭辞を使用します。

次のセクションでは、これらの変数のすべてまたは一部に共通する情報について説明します。後続のセクションでは、アプリケーションでの Client、Session、Application、および Server スコープの使用方法とコードのロックについて詳しく説明します。

ColdFusion 永続変数と ColdFusion 構造体

すべての永続スコープは、ColdFusion 構造体として使用できます。したがって、ColdFusion の構造体関数を使用して、Client、Session、Application、および Server スコープの内容にアクセスしたり操作したりできます。ここでは、特定のスコープの特徴や制限については説明しますが、それらの関数の使用方法の詳細については説明しません。

注意：StructClear 関数を使用して Server スコープからデータをクリアすることはできますが、変数の名前は削除されず、値のみが削除されます。また、Server.os および Server.ColdFusion 構造体の内容は削除されません。StructClear 関数を使用して Session スコープまたは Application スコープをクリアすると、ビルトイン変数を含むスコープ全体がクリアされます。StructClear 関数を使用して Client スコープをクリアすると、サーバーのメモリから変数がクリアされますが、保管されている変数のコピーは削除されません。

ColdFusion の永続変数の問題

Session、Application、および Server スコープの変数は ColdFusion サーバーのメモリに保持されます。これは次のことを意味しています。

- サーバーの実行が停止されると、これらのスコープ内の変数はすべて失われます。
- これらのスコープ内の変数はクラスタ内のサーバー間では共有されません。
- 競合状態が生じるのを避け、データの一貫性を保つために、これらのスコープの変数を変更するすべてのコード、および値が変化する可能性のあるこれらのスコープの変数を読み取るすべてのコードは、アクセスをロックする必要があります。

注意：J2EE セッション管理を使用しており、サーバーを再起動してもセッションデータが保持されるように J2EE サーバーを設定している場合、セッション変数はサーバーの再起動後も保持されます。

また、サーバークラスタで (複数のサーバー上でアプリケーションが実行される可能性がある状況で) クライアント変数を使用するときには注意が必要です。

メモリ変数のロック

ColdFusion は複数のリクエストが Session、Application、および Server スコープ変数を共有できるマルチスレッドシステムであるため、複数のリクエストによって同じデータへのアクセスや変更が同時に行われる可能性があります。ColdFusion は J2EE 環境で実行されるため、同時にデータにアクセスすることはできません。したがって、複数のリクエストによって重大なシステムエラーが発生することはありません。しかし、特にページで複数の変数を変更している場合は、そのようなリクエストによってデータ値に矛盾が発生することがあります。

セッション、アプリケーション、およびサーバー変数でのデータエラーを防ぐには、これらのスコープ内でデータを読み書きするコードをロックします。詳細については、316 ページの「cflock によるコードのロック」を参照してください。

クラスタシステムでの変数の使用

メモリ変数はメモリ内に保管されるので、クラスタ内のすべてのサーバーでは使用できません。したがって、通常は、クラスタ環境で Session、Application、または Server スコープ変数を使用することはありません。ただし、次のような状況では、クラスタシステムでこれらのスコープ変数を使用します。

- クラスタリングシステムで " スティッキー " セッションがサポートされている場合、1 つのユーザーセッションは 1 つのサーバー上で実行され続けることがクラスタリングシステムによって保証されます。この場合は、単一のサーバー上で使用するのと同じようにセッション変数を使用できます。
- データベースなどから一貫して設定されるライトワンスの変数 (一度設定された後は読み取り専用となる変数) については、Application スコープ変数および Server スコープ変数をクラスタで使用できます。

クラスタシステムでクライアント変数を使用するには、変数を Cookie として保管するか、すべてのサーバーで使用可能なデータベースに保管します。データベースに保管する場合は、1 台のサーバー上でのみ、ColdFusion Administrator の [クライアント変数] にある [クライアントストアの追加 / 編集] ページで [アクセスされなかったクライアントのデータを破棄] オプションを選択します。

クライアントステート管理

Web はステートレスシステムであるため、ブラウザから Web サーバーへの接続が行われるたびに、Web サーバー側ではそれらが別々の接続と見なされます。しかし、多くのアプリケーションでは、ユーザーがアプリケーションのページ間を移動するのを追跡する必要があります。これがクライアントステート管理の定義です。

ColdFusion では、ユーザーがアプリケーションのページ間を移動するに従って、ブラウザに関する変数をシームレスに記録することによって、クライアントステートを維持することができます。これらの変数は、クライアントステートを記録する他の方法 (URL パラメータ、非表示フォームフィールド、HTTP Cookie など) の代わりに使用することができます。

クライアント変数およびセッション変数について

ColdFusion には、クライアントステート管理用の 2 つのツールであるクライアント変数とセッション変数が用意されています。どちらのタイプの変数も特定のクライアントに関連付けられていますが、次の表で説明するように、これらは管理方法や使用方法が異なります。

変数タイプ	説明
Client	データは Cookie、データベースエントリ、またはレジストリエントリとして保存されます。データはサーバーを再起動しても保持されますが、そのデータへの最初のアクセスと保存は、メモリに保管されたデータよりも時間がかかります。 それぞれのデータ保管域には、固有のタイムアウト期間があります。データベースおよびレジストリデータのタイムアウトは、ColdFusion Administrator で指定できます。ColdFusion では、Cookie クライアント変数は約 10 年後に無効になるように設定されます。 データは、ユーザーごとおよびアプリケーションごとに保管されます。たとえば、クライアント変数を Cookie として保管すると、ユーザーは、サーバーによって提供される各 ColdFusion アプリケーションに対して個別の Cookie を持ちます。 クライアント変数は、数値、日付、または文字列などの単純変数である必要があります。配列、構造体、クエリーオブジェクト、または他のオブジェクトは使用できません。 クライアント変数名には、ピリオドを含めることができます。たとえば、My.ClientVar は、簡単なクライアント変数の有効な名前ですが、コードを明確にするために、このような名前は使用しないでください。 クライアント変数の参照時には、その接頭辞としてスコープ名を付ける必要はありません。ただし、Client 接頭辞を使用しないと、別のスコープの同じ名前を持つ変数を意図せず参照してしまうおそれがあります。接頭辞を使用すると、パフォーマンスが最適化されるだけでなく、プログラムがより明確になります。 クライアント変数を使用するコードをロックする必要はありません。 Cookie またはクラスタシステムの共有データベースに保存されているクライアント変数を使用できます。
Session	データはメモリに保管されるので、アクセスが速くなります。 クライアントブラウザによるアクセスがないままタイムアウト期間が経過すると、データは失われます。タイムアウトは、ColdFusion Administrator、"Application.cfc" の初期化コード、または "Application.cfm" で指定します。 クライアント変数と同様に、データは単一のクライアントおよびアプリケーションでのみ使用可能です。 変数には任意の ColdFusion データ型を保管できます。 すべての変数名に、その接頭辞として Session スコープ名を付ける必要があります。 競合状態を避けるために、セッション変数を使用するコードをロックする必要があります。 システムが、単一のサーバーにセッションを限定する " スティッキー " セッションをサポートしている場合にのみ、クラスタシステムでセッション変数を使用できます。

単一のブラウザセッションでのみ必要な値に対しては、通常、クライアント変数よりもセッション変数が適しています。クライアント変数は、クライアントの設定など、複数のブラウザセッションにまたがって使用するクライアント固有のデータに対して使用します。

クライアント ID の保持

Web はステートレスシステムであるため、クライアントを管理するには複数のリクエストにまたがってクライアントの情報を保持できる方法が必要です。通常は Cookie を使用しますが、アプリケーションページ間で情報を渡す方法も使用できます。次のセクションでは、ColdFusion がさまざまな設定および環境でクライアントの ID を保持する方法について説明し、クライアントステート管理で発生する可能性のある問題について説明します。

クライアントの識別子について

クライアント変数およびセッション変数を使用するには、ColdFusion でクライアントを識別できる必要があります。ColdFusion では通常、次の 2 つの Cookie の値をクライアントのシステムに設定してクライアントを識別します。

- **CFID**: 連番のクライアント識別子
- **CFToken**: 乱数のクライアントセキュリティトークン

これらの Cookie は、ColdFusion に対してクライアントを一意に識別し、また Session スコープおよび Client スコープの一部としてその変数のコピーも維持します。クライアント Cookie を使用しないようにアプリケーションを設定することもできますが、その場合は、アプリケーションが呼び出すすべてのページにこれらの変数を渡します。Cookie を使用しないクライアント情報およびセッション情報の保持については、303 ページの「Cookie が使用できない場合のクライアント変数とセッション変数の使用」を参照してください。

セッション変数に対して、ColdFusion セッション管理ではなく、J2EE サーブレットセッション管理を使用するように ColdFusion を設定できます。このセッション管理方法では、CFID および CFToken の値を使用せずに、クライアントサイドの jsessionid セッション管理 Cookie を使用します。J2EE セッション管理の使用方法の詳細については、310 ページの「ColdFusion セッション管理と J2EE セッション管理」を参照してください。

Cookie が使用できない場合のクライアント変数とセッション変数の使用

ユーザーがブラウザで Cookie を無効にしていることはよくあります。この場合、ColdFusion はクライアントのステートを自動的に保持することはできません。アプリケーションページ間でクライアントの ID 情報を渡せば、Cookie を使用せずにクライアント変数またはセッション変数を使用できます。ただし、この方法には、次のような重大な制限があります。

1. クライアント変数は事実上セッション変数と同じですが、使用できないデータがクライアントデータストアに残される点が異なります。

   クライアントのシステムには ID 情報が保持されないので、次回ユーザーがログオンしても、ColdFusion ではそのユーザーが以前のクライアントであると識別できません。したがって、ColdFusion は、そのユーザーに対して新規のクライアント ID を作成する必要があります。前のセッションのユーザー情報は使用できませんが、その情報は ColdFusion によって削除されるまでクライアントのデータ保管域に残っています。ColdFusion Administrator の [アクセスされなかったクライアントのデータを破棄] オプションをオフにすると、このデータは削除されません。

   したがって、Cookie の無効化をユーザーに認める場合は、クライアント変数を使用しないでください。Cookie を使用しないでクライアントの情報を保持するには、ログイン時に固有の ID の入力をユーザーに要求します。ユーザーの ID をキーとして使用することで、データベースにユーザー固有の情報を保存できます。

2. ColdFusion では、ユーザーがブラウザでページを直接リクエストするたびに新規セッションが作成されます。これは、セッションまたはクライアントを示すステート情報が新規リクエストに含まれていないからです。

注意: "Application.cfc" 内で This.setClientCookies 変数を No に設定するか、cfapplication タグの setClientCookies 属性を No に設定すると、ColdFusion がクライアント情報を Cookie としてブラウザに送信するのを防ぐことができます。

Cookie を使用せずに ColdFusion セッション変数を使用する場合は、各ページから任意のページを呼び出すときに、リクエスト URL の一部として CFID 値と CFToken 値を渡す必要があります。ページに HTML の hrefa= リンク、cflocation タグ、form タグ、または cfform タグが含まれている場合は、これらのタグで使用する URL の中に CFID および CFToken の値を含める必要があります。J2EE セッション管理を使用する場合は、jsessionid 値をページリクエストの中で渡します。ColdFusion クライアント変数と J2EE セッション変数を使用する場合は、CFID 値、CFToken 値、および jsessionid 値を URL の中で渡します。

注意：CFID と CFTOKEN を URL の中で渡すと、次のようになります。セッションが存在する場合、URL の中の CFID と CFTOKEN は無視されます。セッションが存在しない場合は、URL の中の CFID と CFTOKEN に基づいてセッションが有効かどうか検証され、セッションが有効であれば、そのセッションが使用されます。セッションが有効でなければ、新しいセッションが作成されます。このとき再度、CFID と CFTOKEN が生成さます。

ColdFusion には、次のことが行える URLSessionFormat 関数が用意されています。

- クライアントが Cookie を受け入れない場合は、必要なすべてのクライアント ID 情報を URL に自動的に追加します。
- クライアントが Cookie を受け入れる場合は、情報を追加しません。

URLSessionFormat 関数は、必要な識別子を自動的に調べて、必要な情報のみを送信します。この関数を使用すれば、必要なときに必要な情報のみを送ることができ、コーディングも行いやすいので、各 URL の情報を手動でエンコードするよりも、安全かつ強力にクライアント ID をサポートできます。

URLSessionFormat 関数を使用するには、関数内にリクエスト URL を含めます。たとえば、次の cfform タグは別のページにリクエストを送信し、必要に応じてクライアント ID を送信します。

```
<cfform method="Post" action="#URLSessionFormat("MyActionPage.cfm")#>
```

複数の URLSessionFormat 関数で同じページ URL を使用する場合は、この関数でページ **URL** を処理した結果を変数に保持すれば、わずかですがパフォーマンスを向上させることができ、コードを簡素化できます。次に例を示します。

```
<cfset myEncodedURL=URLSessionFormat(MyActionPage.cfm)>
<cfform method="Post" action="#myEncodedURL#">
```

クライアント識別子とセキュリティ

クライアント識別子では、セキュリティに関係する次のような問題を考慮する必要があります。

- CFToken 識別子の一意性および複雑性の確保
- セッション識別子の可用性の制限

次のセクションではこれらの問題について説明します。

CFToken の一意性とセキュリティの確保

デフォルトでは、CFToken 識別子には 8 桁の乱数が使用されます。ほとんどの環境下のユーザーは、この形式の CFToken を使用して一意かつ安全に識別できます。この数を生成する ColdFusion メソッドで使用されている暗号強度の乱数ジェネレータには、サーバーが開始するときにのみシードが設定されます。

ただし、ColdFusion Administrator の [設定] ページを有効にすることにより、より複雑な CFToken 識別子を生成できます。[cftoken 用の UUID の使用] オプションを有効にすると、16 桁の 16 進数の乱数を ColdFusion UUID の前に追加することで、CFToken 値が作成されます。作成される CFToken 識別子は次のようになります。

```
3ee6c307a7278c7b-5278BEA6-1030-C351-3E33390F2EAD02B9
```

セッションセキュリティの提供

ColdFusion では、Client スコープおよび標準の Session スコープに同じクライアント識別子が使用されます。CFToken および CFID 値は、クライアントを一定時間識別するのに使用されるので、通常はユーザーのブラウザに Cookie として保存されます。これらの Cookie は、クライアントのブラウザで削除されるまで、かなり長い間保持されます。したがって、セッションごとに異なるユーザー識別子を使用する場合と比べて、それらの変数がハッカーにアクセスされる危険性が高くなります。

ユーザーの CFToken および CFID Cookie を入手したハッカーは、盗んだ CFToken および CFID Cookie を使用して、ユーザーのセッション中に Web ページにアクセスし、ユーザーのデータにアクセスできます。このような事態が実際に発生する可能性は低いといえますが、理論的に不可能ではありません。

この脆弱性を解決するには、ColdFusion Administrator の [メモリ変数] ページの [J2EE セッション変数の使用] オプションをオンにします。J2EE セッション管理メカニズムはセッションごとに新しいセッション識別子を作成し、CFToken および CFID Cookie の値を使用しません。

セキュリティ関連の変更

ColdFusion 9 アップデート **1** にアップグレードすると、次のセキュリティ関連の仕様が適用されます。

- CFID、CFTOKEN、および jsessionid に httpOnly が指定されます。これにより、セッション情報がクロスサイトスクリプティング (XSS) の攻撃によって漏洩する可能性が下がります。
- セッション Cookie を httpOnly にするには、Dcoldfusion.sessioncookie.httponly=true のシステムプロパティを設定します。
- セッション Cookie の httpOnly のサポートは、使用するアプリケーションサーバーによって異なります。
  - Tomcat および JBoss の場合、JSESSIONID の httpOnly はサポートされていません。
  - JRun では、システムプロパティ Dcoldfusion.sessioncookie.httponly=true を jvm.config ファイルに追加します。
  - その他のアプリケーションサーバーの場合、セッション Cookie の httpOnly サポートの詳細については、関連するマニュアルを参照してください。

クライアント変数の設定と使用

特定のクライアントおよびアプリケーションに関連付けられているデータや、複数のユーザーセッションにまたがって保持する必要のあるデータには、クライアント変数を使用します。ユーザーディスプレイやコンテンツの環境設定などの長期間に及ぶ情報には、クライアント変数を使用します。

クライアント変数の有効化

クライアント変数を有効にするには、"Application.cfc" の初期化コードで This.clientmanagement="True" を指定するか、"Application.cfm" ファイルで cfapplication タグの clientmanagement 属性を Yes に設定します。たとえば、SearchApp というアプリケーションでクライアント変数を有効にするには、このアプリケーションの "Application.cfm" ページで次の行を使用します。

```
<cfapplication NAME="SearchApp" clientmanagement="Yes">
```

クライアント変数の保管方法の選択

デフォルトでは、クライアント変数はレジストリ内に保管されます。しかし、ほとんどの場合は、情報をクライアント Cookie として保管するか SQL データベースに保管するほうが適しています。

ColdFusion Administrator の [クライアント変数] ページでは、デフォルトのクライアント変数の保管場所を制御できます。デフォルトの場所を上書きするには、"Application.cfc" で This.clientStorage の値を指定するか、cfapplication タグに clientStorage 属性を設定します。

クライアント変数の保管方法としては、次の値を指定できます。

- Registry (デフォルト)。クライアント変数は、キー HKEY_LOCAL_MACHINE¥SOFTWARE¥Macromedia¥ColdFusion¥CurrentVersion¥Clients の下に保管されます。
- ColdFusion Administrator で設定したデータソースの名前
- Cookie

一般に、クライアント変数はデータベースに保管するのが最も効果的です。Registry オプションがデフォルトですが、レジストリにはクライアントデータの保管に関する重大な制限があります。レジストリはクラスタシステムでは使用できず、UNIX でレジストリにクライアント変数を保存することはサポートされていません。

Cookie 保管域の使用

クライアント変数の保管方法を Cookie に設定した場合、ColdFusion によって作成される Cookie には、アプリケーションの名前が付けられます。Cookie 内にクライアントデータを保管する方法には、多数のクライアントに対応できるという利点がありますが、いくつかの制限も存在します。特に、クライアントがブラウザの Cookie を無効にしていると、クライアント変数は機能しません。

Cookie にクライアント変数を保管する前に、次の制限事項に注意してください。

- cfflush タグの後で設定したクライアント変数はブラウザに送信されないので、変数の値は保存されません。
- 一部のブラウザでは、特定のホストからの Cookie を 20 個までしか受け付けません。ColdFusion では、CFID および CFToken 識別子に 2 個の Cookie が使用されます。また、HitCount、TimeCreated、LastVisit などのクライアントに関するグローバルデータを保持するために、cfglobals という Cookie が作成されます。したがって、使用できるアプリケーションの数は、クライアントとホストの組み合わせごとに最大 17 個に制限されます。
- ブラウザの中には、Cookie 1 個あたり 4 KB というサイズ制限を持つものがあります。ColdFusion で作成される Cookie 内では、英数字以外のデータは URL のエンコード方式でエンコードされて 3 倍の長さになります。したがって、クライアントごとに大量のデータを保管することは望ましくありません。4 KB を超えるエンコード済みのクライアントデータを保管しようとすると、ColdFusion でエラーが発生します。

データベース保管の設定

クライアント変数の保管先としてデータベースを指定する場合は、クライアント変数を保管するデータテーブルを常に手動で作成する必要はありません。

使用しているデータベースでデータベーステーブルの SQL 作成がサポートされていることを ColdFusion で識別できる場合は、あらかじめデータベースを作成します。[メモリ変数] ページの [クライアントストアとして追加するためのデータソースの選択] ボックスの [追加] ボタンをクリックすると、[クライアントストアの追加 / 編集] ページが表示されます。このページには、[クライアントデータベーステーブルの作成] 選択ボックスがあります。このオプションを選択すると、ColdFusion によって必要なテーブルがデータベース内に作成されます。データベースに既に必要なテーブルがある場合、このオプションは表示されません。

データベースでテーブルの SQL 作成がサポートされていない場合や、ODBC ソケット [Macromedia] ドライバを使用してデータベースにアクセスしている場合は、データベースツールを使用してクライアント変数のテーブルを作成します。CDATA テーブルと CGLOBAL テーブルを作成してください。

CDATA テーブルには次の列が必要です。

列	データ型
cfid	CHAR(64)、TEXT、VARCHAR、または最大 64 文字の可変長文字列を格納できる任意のデータ型
app	CHAR(64)、TEXT、VARCHAR、または最大 64 文字の可変長文字列を格納できる任意のデータ型
data	MEMO、LONGTEXT、LONG VARCHAR、CLOB、または不定長の長い文字列を格納できる任意のデータ型

CGLOBAL テーブルには次の列が必要です。

列	データ型
cfid	CHAR(64)、TEXT、VARCHAR、または最大 64 文字の可変長文字列を格納できる任意のデータ型
data	MEMO、LONGTEXT、LONG VARCHAR、CLOB、または不定長の長い文字列を格納できる任意のデータ型
lvisit	TIMESTAMP、DATETIME、DATE、または日付時刻値を保管できる任意のデータ型

注意：データ型の名前は、データベースによって異なります。この表で示した名前は一般的なものであり、使用するデータベースによっては別の名前が使用されていることがあります。

パフォーマンスを向上させるには、これらのテーブルの作成時にインデックスを作成します。CDATA テーブルでは、cfid および app 列のインデックスを作成します。CGLOBAL テーブルでは、cfid 列のインデックスを作成します。

アプリケーションにおけるクライアント変数保管域の指定

クライアント変数のデフォルトの保管場所を上書きするには、"Application.cfc" の初期化コードで This.clientstorage の値を設定するか、cfapplication タグの clientStorage 属性を使用します。

次の例では、"Application.cfc" ファイルでクライアント変数を mydatasource データソースに保管するように指定しています。

```
<cfscript>
    This.name"SearchApp";
    This.clientManagement="Yes";
    This.clientStorage="mydatasource";
</cfscript>
```

次の例では、"Application.cfm" ファイルで前の例と同じ設定を行っています。

```
<cfapplication name"SearchApp"
    clientmanagement="Yes"
    clientstorage="mydatasource">
```

クライアント変数の使用

アプリケーションでクライアント変数を有効にすると、特定のクライアントに関連する長期的な情報を追跡することができます。

クライアント変数は、文字列、数値、リスト、ブール値、または日付時刻値の単純データ型である必要があります。配列、レコードセット、XML オブジェクト、クエリーオブジェクト、または他のオブジェクトは使用できません。複合データ型をクライアント変数として保管する必要がある場合は、cfwddx タグを使用して、データを WDDX 形式 (文字列) に変換し、変換した WDDX データを保管します。そのデータを読み取るときには cfwddx タグを使用してデータを元に戻します。WDDX の使用方法の詳細については、1057 ページの「WDDX の使用」を参照してください。

注意：クライアント変数データを WDDX 形式で保存する場合、レジストリおよび SQL Server では約 4K、ORACLE では約 2K のサイズ制限があります。

クライアント変数の作成

クライアント変数を作成しその値を設定するには、cfset タグまたは cfparam タグを使用し、さらに変数の接頭辞として Client スコープ識別子を使用します。次に例を示します。

```
<cfset Client.FavoriteColor="Red">
```

このようにクライアント変数を設定すると、変数が設定されたクライアントによってアクセスされるアプリケーションの任意のページ内で、その変数を使用できます。

次の例では、cfparam タグを使用してクライアントパラメータの存在を調べ、パラメータが存在していない場合にはデフォルト値を設定します。

```
<cfparam name="Client.FavoriteColor" default="Red">
```

クライアント変数へのアクセスと変更

クライアント変数には、他のタイプの変数と同じシンタックスを使用してアクセスします。クライアント変数は、他の ColdFusion 変数が使用できるすべての箇所で使用できます。

たとえば、特定のユーザー用に設定された好みの色を表示するには、次のようなコードを使用します。

```
<cfoutput>
    Your favorite color is #Client.FavoriteColor#.
</cfoutput>
```

クライアントの好みの色を変更するには、次のようなコードを使用します。

```
<cfset Client.FavoriteColor = Form.FavoriteColor>
```

標準クライアント変数

Client スコープには、アプリケーションで使用できる、次のような読み取り専用のビルトイン変数があります。

変数	説明
Client.CFID	通常 Cookie としてクライアントシステムに保管されるクライアント ID
Client.CFToken	通常 Cookie としてクライアントシステムに保管されるクライアントセキュリティトークン
Client.URLToken	この値は、J2EE セッション管理が有効化されているかどうかによって決まります。 セッション管理なし、または ColdFusion セッション管理 : CFID=IDNum&CFTOKEN=tokenNum 形式の、CFID および CFToken 値の組み合わせ。この変数は、クライアントで Cookie がサポートされておらず、CFID 変数と CFToken 変数をページ間で渡す場合に有用です。 J2EE セッション管理 : CFID=IDNum&CFTOKEN=tokenNum&jsessionid=SessionID 形式の、CFID、CFToken、およびセッション ID 値の組み合わせ。
Client.HitCount	クライアントにより実行されたページリクエストの回数
Client.LastVisit	クライアントが最後にアプリケーションにアクセスした日付
Client.TimeCreated	ColdFusion でクライアントを識別するための CFID 変数と CFToken 変数が最初に作成された時刻

注意：ColdFusion では、ビルトインクライアント変数の値の削除および変更を行うことはできますが、通常は行わないでください。

Cookie を許可しないブラウザをアプリケーションでサポートする場合は、Client.CFID、Client.CFToken、および Client.URLToken 変数を使用します。Cookie を許可しないブラウザをサポートする方法の詳細については、303 ページの「Cookie が使用できない場合のクライアント変数とセッション変数の使用」を参照してください。

Client.HitCount と時刻情報の変数を使用して、ユーザーがサイトにアクセスする頻度と最後にサイトにアクセスした日付に応じて動作をカスタマイズできます。たとえば、次のコードでは、ユーザーがサイトを最後にアクセスした日付が表示されます。

```
<cfoutput>
    Welcome back to the Web SuperShop. Your last
    visit was on #DateFormat(Client.LastVisit)#.
</cfoutput>
```

クライアント変数のリストの取得

特定のクライアントに関連付けられているカスタムのクライアントパラメータのリストを取得するには、次のように GetClientVariablesList 関数を使用します。

```
<cfoutput>#GetClientVariablesList()#</cfoutput>
```

GetClientVariablesList 関数は、現在のアプリケーションに設定されているクライアント変数名のカンマ区切りのリストを返します。システムが提供する標準クライアント変数 (CFID、CFToken、URLToken、HitCount、TimeCreated、および LastVisit) は、リスト内には返されません。

クライアント変数の削除

クライアント変数を削除するには、StructDelete 関数または DeleteClientVariable 関数を使用します。例えば、次の行は同等です。

```
<cfset IsDeleteSuccessful=DeleteClientVariable("MyClientVariable")>
<cfset IsDeleteSuccessful=StructDelete(Client, "MyClientVariable")>
```

ColdFusion Administrator の [クライアント変数] ページを使用すると、タイムアウト期間を設定できます。アクティブでない状態がこのタイムアウト期間続くと、レジストリまたはデータソースに保管されているクライアント変数が削除されます。クライアント変数がレジストリに保管されている場合のデフォルト値は 10 日間です。クライアント変数がデータソースに保管されている場合は 90 日間です。

注意：システムが提供するクライアント変数 (CFID、CFToken、URLToken、HitCount、TimeCreated、および LastVisit) を削除することはできません。

クライアント変数と cflocation の併用

cflocation タグを使用して .dbm または .cfm で終了するパスに ColdFusion を転送すると、Client.URLToken 変数が自動的に URL に付加されます。この動作は、addtoken="No" 属性を cflocation タグに追加することで無効にすることができます。

クライアント変数のキャッシュ

ColdFusion でクライアント変数を読み書きすると、クライアントデータへのアクセスのオーバーヘッドを減らすために、メモリに変数がキャッシュされます。したがって、ColdFusion がクライアントデータストアにアクセスするのは、最初にクライアントデータストアの値を読み取るとき、またはリクエスト終了時に値を設定するときのみです。クライアント変数へのその他の参照では、ColdFusion のメモリ内のキャッシュされた値が使用されるので、ページがより速く処理されます。

クライアント変数データベースのエクスポート

クライアント変数データベースが Windows のシステムレジストリに保存されており、そのデータベースを別のコンピュータに移動する必要があるときは、クライアント変数が保存されているレジストリキーをエクスポートし、それを新しいサーバーに移動できます。システムレジストリでは、レジストリエントリをエクスポートしたり、インポートしたりできます。

Windows でレジストリからクライアント変数データベースをエクスポートするには :

1. レジストリエディタを開きます。
2. 次のキーを検索し、選択します。

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\Macromedia\ColdFusion\CurrentVersion\Clients
   ```

3. [レジストリ] メニューで、[レジストリファイルの書き出し] をクリックします。
4. レジストリファイルの名前を入力します。

レジストリファイルを作成したら、そのファイルを新しいマシンにコピーし、レジストリエディタの [レジストリ] メニューから [レジストリファイルの取り込み] をクリックすることによって、そのファイルをインポートできます。

注意：UNIX システムでは、レジストリエントリは "/opt/coldfusion/registry/cf.registry" に保持されます。これは直接コピーや編集が可能なテキストファイルです。

セッション変数の設定と使用

1 回のサイトアクセスまたは一連のリクエストに対して変数が必要なときは、セッション変数を使用します。たとえば、セッション変数を使用して、ショッピングアプリケーションでユーザーがショッピングカートに入れたアイテムを保存できます。複数回のアクセスにまたがって変数を保持する必要がある場合は、クライアント変数を使用します。

重要：同じセッション変数が複数の場所からアクセスされて競合状態が生じる可能性がある場合は、セッション変数を使用するコードを cflock タグで囲む必要があります。cflock タグの使用方法の詳細については、316 ページの「cflock によるコードのロック」を参照してください。

セッションとは

セッションとは、特定のアプリケーションの一連のページを表示するために、単一のクライアントからサーバーに対して行われるすべての接続のことを意味します。セッションは、ユーザーおよびアプリケーションごとに固有です。つまり、アプリケーションのすべてのユーザーはそれぞれ個別のセッションを持ち、個別のセッション変数にアクセスします。

論理的には、クライアントがアプリケーションに最初に接続したときにセッションが開始され、そのクライアントが最後の接続を終えたときにセッションが終了します。ただし、Web はステートレスなので、いつセッションが終了したのかを正確に定義できないことがあります。ユーザーがアプリケーションの使用を終えたときにセッションが終了したと考えるのが妥当ですが、ユーザーがアプリケーションの使用を終えたのか、まだ何か行おうとしているのかを Web アプリケーションで判断できることはあまりありません。

したがって、タイムアウトに指定された時間を超えてアクティブでない状態が続くと、セッションは常に終了するようになっています。タイムアウト期間内にユーザーがアプリケーションのページにアクセスしないと、ColdFusion では、セッションが終了したと解釈され、そのセッションに関連付けられているすべての変数がクリアされます。

セッション変数のデフォルトのタイムアウトは 20 分間です。このデフォルトのタイムアウトは、ColdFusion Administrator の [サーバーの設定] 領域にある [メモリ変数] ページで変更できます。

"Application.cfc" で This.sessionTimeout の値を設定するか、cfapplication タグの sessionTimeout 属性を使用することで、特定のアプリケーションについてセッション変数のタイムアウト期間を設定することもできます。その場合、ColdFusion Administrator で設定したデフォルトの値は無効になります。ただし、ColdFusion Administrator の [メモリ変数] ページに設定されている最大セッションタイムアウト値よりも大きい値をタイムアウト値として設定することはできません。

セッション終了とセッション変数の削除の詳細については、313 ページの「セッションの終了」を参照してください。

ColdFusion セッション管理と J2EE セッション管理

ColdFusion サーバーでは、次の 2 つのタイプのセッション管理を使用できます。

- ColdFusion セッション管理
- J2EE サーブレットセッション管理

ColdFusion セッション管理では、ColdFusion クライアント管理と同じ方法でクライアントが識別されます。

J2EE セッション管理は、ColdFusion セッション管理に比べて次の点で優れています。

- J2EE セッション管理ではセッション固有のセッション識別子である jsessionid が使用されます。これは個々のセッションが開始するたびに新たに作成されます。
- 複数の ColdFusion ページ、およびそこから呼び出す JSP ページや Java サーブレットとの間で、セッション変数を共有できます。
- Session スコープはシリアル化可能 (後で元のオブジェクトに完全に復元できるバイトシーケンスに変換可能) です。ColdFusion セッション管理では、Session スコープはシリアル化できません。シリアル化可能なスコープのみが、サーバー間で共有できます。

したがって、次のような場合には J2EE セッション管理の使用を検討してください。

- セッションのセキュリティレベルを最大にする場合、特にクライアント変数も使用する場合
- 単一のアプリケーション内の複数の ColdFusion ページ、および JSP ページまたはサーブレット間でセッション変数を共有する場合
- クライアント ID の Cookie を Client スコープで使用できるようにしながら、セッションを手動で終了できるようにする場合
- クラスタ化されたセッション、たとえばサーバー間のセッションフェイルオーバーをサポートする場合

セッション変数の設定と有効化

セッション変数を使用するには、次の 2 か所でセッション変数を有効にします。

- ColdFusion Administrator
- "Application.cfc" の初期化コードの This.sessionManagement 変数、またはアクティブな cfapplication タグ

ColdFusion Administrator、"Application.cfc"、および cfapplication タグには、変数のタイムアウトなどのセッション変数の動作を設定する機能もあります。

ColdFusion Administrator でのセッション変数の選択と有効化

セッション変数を使用するには、ColdFusion Administrator の [メモリ変数] ページでセッション変数を有効にする必要があります。デフォルトでは有効になっています。ColdFusion Administrator の [メモリ変数] ページでは、次のことも行えます。

- ColdFusion セッション管理 (デフォルト) または J2EE セッション管理の選択。
- デフォルトのセッションタイムアウトの変更。この値はアプリケーションのコードで上書きできます。このタイムアウトのデフォルト値は 20 分間です。
- 最大セッションタイムアウトの指定。この値を超えるタイムアウト値をアプリケーションのコードで設定することはできません。このタイムアウトのデフォルト値は 2 日間です。

アプリケーションでのセッション変数の有効化

セッション変数は、"Application.cfc" ファイルの初期化コード内、または "Application.cfm" ファイルの cfapplication タグで有効にします。

セッション変数を有効にするには、"Application.cfc" ファイルの初期化コード内で、cfcomponent タグよりも後で、次のことを行います。

- This.sessionManagement="Yes" を設定します。
- This.name にアプリケーションの名前を設定します。
- 必要に応じて、This.sessionTimeout にアプリケーション固有のセッションタイムアウト値を設定します。CreateTimeSpan 関数を使用して、タイムアウトの日数、時間、分、および秒を指定します。

セッション変数を有効にするには、"Application.cfm" ファイルで次のことを行います。

- sessionManagement="Yes" を設定します。
- name 属性を使用して、アプリケーションの名前を指定します。
- 必要に応じて、sessionTimeout 属性を使用してアプリケーション固有のセッションタイムアウト値を設定します。CreateTimeSpan 関数を使用して、タイムアウトの日数、時間、分、および秒を指定します。

次のサンプルコードでは、GetLeadApp アプリケーションのセッション管理を有効にし、アクティブでない状態が 45 分間続くとタイムアウトするようにセッション変数を設定しています。

```
<cfapplication name="GetLeadApp"
    sessionmanagement="Yes"
    sessiontimeout=#CreateTimeSpan(0,0,45,0)#>
```

セッション変数へのセッションデータの保管

セッション変数は、セッションレベルのデータを保管するように設計されています。セッション変数は、ショッピングカートの内容やスコアカウンタなど、そのアプリケーションのすべてのページでユーザーセッションの間に必要な情報を保管するのに便利です。

セッション変数を使用すると、ユーザーがアプリケーションの任意のページに初めてアクセスしたときに、ユーザー固有のデータでアプリケーションを初期化することができます。この情報は、ユーザーがそのアプリケーションを使用している間使用できます。たとえば、ユーザーがアプリケーションの任意のページに初めてアクセスしたときに、そのユーザーの設定情報をデータベースから 1 回だけ取得することができます。ユーザーセッション中はこの情報を使用できるので、設定情報を何度も取得するオーバーヘッドを避けることができます。

標準のセッション変数

ColdFusion のセッション変数を使用する場合は、Session スコープに用意されている読み取り専用の 4 つのビルトイン変数が使用できます。J2EE セッション管理を使用する場合は、Session スコープの 2 つのビルトイン変数が使用できます。これらの変数を ColdFusion ページで使用するのは、通常、Cookie を受け付けないブラウザをアプリケーションでサポートする場合のみです。Cookie を許可しないブラウザをサポートする方法の詳細については、303 ページの「Cookie が使用できない場合のクライアント変数とセッション変数の使用」を参照してください。次の表に、ビルトインセッション変数を示します。

変数	説明
Session.CFID	ColdFusion セッション管理のみ : 通常 Cookie としてクライアントシステムに保管されるクライアント ID。
Session.CFToken	ColdFusion セッション管理のみ : 通常 Cookie としてクライアントシステムに保管されるクライアントセキュリティトークン。
Session.URLToken	ColdFusion セッション管理 : CFID=IDNum&CFTOKEN=tokenNum 形式の、CFID および CFToken 値の組み合わせ。これは、クライアントで Cookie がサポートされておらず、CFID 変数と CFToken 変数をページ間で渡す必要がある場合に有用です。 J2EE セッション管理 : CFID=IDNum&CFTOKEN=tokenNum&jsessionid=SessionID 形式の、CFID Cookie、CFToken Cookie、および J2EE セッション ID の組み合わせ。
Session.SessionID	セッション固有の識別子。 ColdFusion セッション管理 : アプリケーション名と CFID および CFToken 値の組み合わせ。 J2EE セッション管理 : jsessionid 値。

注意 : ColdFusion では、ビルトインセッション変数の値の削除および変更を行うことはできますが、通常は行わないでください。

クライアント変数および ColdFusion セッション管理を有効にすると、ColdFusion では Client スコープおよび Session スコープの CFID、CFToken、および URLtoken 変数に同じ値が使用されます。これらの変数の値は、共通のソースであるクライアントの CFID Cookie および CFTOKEN Cookie から取得されます。

J2EE セッション管理を使用する場合、Session スコープに Session.CFID または Session.CFToken 変数は含まれませんが、Session.URLToken および Session.SessionID 変数は含まれます。この場合、Session.SessionID は J2EE セッション ID であり、Session.URLToken は J2EE セッション ID の前に文字列 jsessionid= が付いたものになります。

セッション変数のリストの取得

セッション変数のリストを取得するには、次のように StructKeyList 関数を使用します。

```
<cflock timeout=20 scope="Session" type="Readonly">
    <cfoutput> #StructKeyList(Session)# </cfoutput>
</cflock>
```

重要 : セッション変数にアクセスするコードは常に cflock タグで囲んでください。

セッション変数の作成と削除

新しいセッション変数を作成するには、次のように、標準的な代入ステートメントを使用します。

```
<cflock timeout=20 scope="Session" type="Exclusive">
    <cfset Session.ShoppingCartItems = 0>
</cflock>
```

セッション変数を削除するには、次のように、structdelete タグを使用します。

```
<cflock timeout=20 scope="Session" type="Exclusive">
    <cfset StructDelete(Session, "ShoppingCartItems")>
</cflock>
```

注意：cflocation タグを使用している CFML テンプレートでセッション変数を設定しても、その変数に値が設定されないことがあります。詳細については、テクニカルノート (www.adobe.com/go/tn_18171) を参照してください。

セッション変数へのアクセスと変更

セッション変数にアクセスするには、通常の変数と同じシンタックスを使用します。ただし、セッション変数に対してアクセスまたは変更を行うコードはロックします。

たとえば、ショッピングカート内のアイテムの数を表示するには、次のコードを使用します。

```
<cflock timeout=20 scope="Session" type="Exclusive">
    <cfoutput>
        Your shopping cart has #Session.ShoppingCartItems# items.
    </cfoutput>
</cflock>
```

ショッピングカート内のアイテムの数を増やすには、次のコードを使用します。

```
<cflock timeout=20 scope="Session" type="Exclusive">
    <cfset Session.ShoppingCartItems = Session.ShoppingCartItems + 1>
</cflock>
```

セッションの終了

セッションの終了と Session スコープ変数の削除については、次のルールが適用されます。

- ColdFusion セッション管理を使用している場合は、クライアントからアクセスがないままセッションタイムアウト期間が経過すると、ColdFusion によってセッションが終了され、Session スコープの変数がすべて削除されます。ユーザーがブラウザを閉じてもセッションは終了しません。
- J2EE セッション管理を使用している場合は、クライアントからアクセスがないままセッションタイムアウト期間が経過すると、ColdFusion によってセッションが終了され、Session スコープの変数がすべて削除されます。ただし、ブラウザからは同じセッション ID が送信され続けます。この ID は、ブラウザがアクティブである限り、そのブラウザインスタンスとのセッションで再利用されます。
- ユーザーがログアウトしてもセッションは終了せず、Session スコープの変数は削除されません。
- 多くの場合、次のようなコードによって Session スコープをクリアすればセッションは事実上終了します。ただし、これには次のリストで説明する重要な制限および代替手段があります。

  ```
  <cfset StructClear(Session)>
  ```

- Session スコープをクリアしてもセッション ID はクリアされないので、ブラウザから送信されてくるリクエストでは、ブラウザが終了するまで同じセッション ID が使用されます。また、Session スコープにログイン情報を保管していても、Session スコープをクリアすることでユーザーをログアウトさせることはできません。ユーザーをログアウトさせるには必ず cflogout タグを使用してください。
- J2EE セッション管理を使用する場合は、次のようにしてセッションを無効化できます。

  ```
  <cfset getPageContext().getSession().invalidate()>
  ```

  この行では、サーブレットページのコンテキストへのポインタを作成し、内部メソッドを呼び出してセッションをリセットしています。これによって、セッション ID Session スコープ変数を含むすべてのセッション情報がクリアされ、ログインをセッションに保管している場合は、以降のリクエストに使用されるログイン情報もクリアされます。ただし、現在のリクエストが終了するまで、セッション情報は使用可能なまま存続します。セッションを無効化した後で、ブラウザがそのアプリケーションにアクセスしようとすると、セッションがタイムアウトするまで無効セッション例外が発生します。

  注意：1 つのリクエストで、セッションを廃棄してセッションを作成することはできません。これは、新しいセッションの作成時にはセッション Cookie の返送が行われるからです。

- クライアント Cookie を使用しない場合は、セッションの CFID と CFTOKEN (J2EE セッションの場合はさらに jsessionid) の値を URL クエリー文字列で渡している間のみ、Session スコープとログインステートが使用できます。ただし、それらの値を使用するのをやめた後も、セッションタイムアウト期間が経過するまでメモリ内のセッションデータは存続します。

アプリケーション変数の設定と使用

アプリケーション変数は、アプリケーション内のすべてのページ、つまり同じアプリケーション名を持つページで使用できます。アプリケーション変数は永続変数なので、値をページ間で簡単に渡すことができます。アプリケーション変数は、アプリケーション名、背景色、データソース名、連絡先などの情報に使用できます。

アプリケーション名は通常、アプリケーションの "Application.cfm" ページで cfapplication タグを使用して設定します。アプリケーション名は、Application.applicationName 変数に保管されます。

クライアント変数やセッション変数とは異なり、アプリケーション変数にクライアント名 (クライアント ID) を関連付ける必要はありません。アプリケーション変数は、アプリケーションのページを使用する任意のクライアントで使用可能です。

重要：同じアプリケーション変数が複数の場所からアクセスされて競合状態が生じる可能性がある場合は、アプリケーション変数を使用するコードを cflock タグで囲む必要があります。cflock タグの使用方法の詳細については、316 ページの「cflock によるコードのロック」を参照してください。

アプリケーション変数の設定と有効化

アプリケーション変数を使用するには、次の作業を行います。

- ColdFusion Administrator でアプリケーション変数が有効にされていることを確認します。デフォルトでは有効になっています。
- "Application.cfc" の初期化コードで This.name 変数を設定するか、現在のページの cfapplication タグに name 属性を設定することで、アプリケーション名を指定します。

注意：ColdFusion では、J2EE アプリケーションとの互換性を保つため、名前のないアプリケーションがサポートされています。詳細については、1115 ページの「名前のない ColdFusion Application および Session スコープ」を参照してください。

また、ColdFusion Administrator では次の情報も指定できます。

- 変数のデフォルトのタイムアウト。タイムアウト期間が経過してもアプリケーション内のすべてのページがアクティブにならない場合は、ColdFusion によってすべてのアプリケーション変数が削除されます。"Application.cfc" ファイルまたは cfapplication タグを使用することで、この値を特定のアプリケーションについて上書きできます。このタイムアウトのデフォルト値は 2 日間です。
- タイムアウトの最大値。この値を超えるタイムアウト値をアプリケーションのコードで設定することはできません。このタイムアウトのデフォルト値は 2 日間です。

"Application.cfc" で This.applicationTimeout 変数を使用するか、cfapplication タグの applicationTimeout 属性を使用して、特定のアプリケーションでのアプリケーション変数のタイムアウト期間を設定できます。

アプリケーション変数へのアプリケーションデータの保管

アプリケーション変数は、アプリケーションが実行されているクライアントに関係なく、アプリケーションのすべてのページで必要な情報を保管するのに便利です。アプリケーション変数を使用すると、たとえば、最初のユーザーがアプリケーションの任意のページにアクセスしたときに、アプリケーションを初期化できます。この情報は無期限に使用できるので、繰り返し初期化するオーバーヘッドを避けることができます。

アプリケーション変数に保管されているデータは、アプリケーション内のすべてのページで使用可能であり、アクティブでない状態で一定期間が経過するまで、または ColdFusion Server がシャットダウンされるまで使用できるので、アプリケーションのグローバルな永続データとして使用するのに便利です。

ただし、アプリケーションを実行しているどのクライアントでも同じアプリケーション変数が使用されるので、クライアント固有の情報やセッション固有の情報としては使用できません。特定のクライアント用の変数が必要な場合は、クライアント変数またはセッション変数を使用します。

アプリケーション変数の使用

アプリケーション変数は、書き込む頻度が低い情報に対して使用するのが一般的です。ほとんどの場合、これらの変数の値は 1 回だけ (多くの場合、アプリケーションを最初に開始するときに) 設定します。これらの変数の値は、アプリケーションやセッションの有効期間中に何回も参照されます。

同じアプリケーション変数が複数の場所からアクセスされて競合状態が生じる可能性がある場合は、Application スコープ変数にデータを書き込むコードや、値が変化する可能性のある Application スコープ変数を読み取るコードは、すべて cflock タグで囲む必要があります。

Application スコープの各変数は、そのアプリケーション内のすべてのリクエストによってメモリで共有されるので、使用方法を誤るとこれらの変数がボトルネックになる場合があります。あるリクエストが Application スコープの特定の変数を読み書きしているときは、そのコードがその変数へのアクセスを終えるまで、その変数を使用する他のリクエストはすべて待機する必要があります。この問題は、ロックに要する時間が長くなるほど深刻になります。多くのユーザーがこのアプリケーションに同時にアクセスしている場合に Application スコープの変数を頻繁に使用すると、アプリケーションのパフォーマンスが低下します。アプリケーションで多くのアプリケーション変数を使用している場合は、変数が Application スコープに存在する必要があるかどうか、Session スコープ変数や Request スコープ変数に置き換えられないかどうかを検討してください。

Application スコープには、1 つのビルトイン変数 Application.applicationName があります。これには、cfapplication タグで指定したアプリケーション名が含まれています。

アプリケーション変数には、セッション変数を使用するのと同じ方法でアクセスし、操作します。ただし、変数の接頭辞に Session ではなく Application を使用する点と、Session をロックスコープとして指定する点が異なります。セッション変数の使用例については、312 ページの「セッション変数の作成と削除」および 313 ページの「セッション変数へのアクセスと変更」を参照してください。

一度設定された後は読み取り専用となるライトワンスのアプリケーション変数の効率的なロック方法については、322 ページの「アプリケーション変数の効果的なロック」を参照してください。

サーバー変数の使用

サーバー変数は、単一の ColdFusion サーバーに関連付けられています。サーバー変数は、サーバー上で実行するすべてのアプリケーションで使用できます。この変数は、サーバーのグローバルヒット数などの、クライアントやアプリケーション全体でアクセスされるデータに対して使用します。

サーバー変数はタイムアウトしませんが、サーバーがシャットダウンすると失われます。サーバー変数は削除できます。

サーバー変数は単一のサーバーに保管されます。したがって、サーバークラスタで ColdFusion を使用する場合、サーバー変数は適切に機能しません。

サーバー変数のアクセス方法や操作方法は、変数の接頭辞として Server を使用することを除き、セッション変数およびアプリケーション変数の場合と同じです。

重要：同じサーバー変数が複数の場所からアクセスされて競合状態が生じる可能性がある場合は、サーバー変数を使用するコードを cflock タグで囲む必要があります。ビルトインサーバー変数へのアクセスをロックする必要はありません。

ColdFusion には、次の読み取り専用の標準ビルトインサーバー変数があります。

変数	説明
Server.ColdFusion.AppServer	ColdFusion が使用している J2EE アプリケーションサーバーの名前。ColdFusion Server Edition では、アプリケーションサーバーが統合されているので、この名前は JRun 4 になります。
Server.ColdFusion.Expiration	ColdFusion ライセンスが切れる試用期限の日付 (トライアル版の場合)
Server.ColdFusion.ProductLevel	サーバーの製品レベル (Enterprise など)
Server.ColdFusion.ProductName	製品の名前 (ColdFusion)
Server.ColdFusion.ProductVersion	実行中のサーバーのバージョン番号 (6,0,0 など)
Server.ColdFusion.Rootdir	ColdFusion がインストールされているディレクトリ ("C:¥cfusion" など)
Server.ColdFusion.SerialNumber	このサーバーのインストール用に割り当てられたシリアル番号
Server.ColdFusion.SupportedLocales	English (US) や Spanish (Standard) など、サーバーでサポートされているロケール
Server.OS.AdditionalInformation	オペレーティングシステムにより提供される追加情報 (Service Pack 番号など)
Server.OS.arch	プロセッサのアーキテクチャ (Intel Pentium プロセッサの x86 など)
Server.OS.BuildNumber	特定のオペレーティングシステムのビルド番号 (1381 など)
Server.OS.Name	オペレーティングシステムの名前 (Windows NT など)
Server.OS.Version	オペレーティングシステムのバージョン番号 (4.0 など)

cflock によるコードのロック

cflock タグは、ColdFusion コードへの同時アクセスを制御します。cflock タグを使用すると、次のことが行えます。

- Session、Application、および Server スコープ (ColdFusion スレッドを使用するアプリケーションでは Request および Variables スコープも) に保存されている共有データに対してアクセスや操作を行うコードセクションを保護します。
- 他のアプリケーションや ColdFusion タグが書き込みのためにファイルを開いたことが原因でファイルの更新が失敗しないようにします。
- スレッドセーフでない CFX API を使用して記述された ColdFusion Extension タグによって、アプリケーションが同時にアクセスされないようにします。この機能は、同時アクセスを防ぐことなく共有 (グローバル) データ構造体を使用する (スレッドセーフでない) CFX タグで重要です。ただし、Java の CFX タグは、CFX タグのアクセスがロックされていない場合に一貫性を損なう可能性がある共有リソースにもアクセスできます。
- スレッドセーフでないデータベースに、アプリケーションが同時にアクセスしないようにします。この機能はほとんどのデータベースシステムでは必要ありません。

ColdFusion は、複数のページリクエストを一度に処理できるマルチスレッド Web アプリケーションサーバーです。したがって、複数のリクエストによって、同じ情報またはリソースに同時にアクセスが試みられる可能性があります。

ColdFusion はスレッドセーフなので、1 つの変数が同時に変更されることはありませんが、これらのリクエストが正しい順序で情報にアクセスすることは保証されません。複数のページ、または同じページの複数回の呼び出しによって、データが同時に書き込まれたり、読み取りと書き込みが同時に行われたりすると、データの不整合が生じる可能性があります。これについては、次のロックシナリオの例のセクションで説明します。

また、同時に行われた複数のリクエストを適切に処理できない外部リソース (ファイル、データベース、CFX タグなど) も同様です。ColdFusion では、2 つのコードがそれらのリソースにアクセスする可能性が常にあります。また、そのような共有リソースに対して正しい順序でアクセスが行われ、データが正しく処理されることも保証されません。

これらのリソースにアクセスするコードをロックすれば、1 つのリソースに同時にアクセスできるスレッドが 1 つに限定されるので、競合状態を避けることができます。

ロックシナリオの例

次の例では、ColdFusion コードをロックする必要があるシナリオを示します。ロックが必要なシナリオは数多くありますが、このセクションではそのうちの 2 つを紹介します。

共有変数の読み書き

チケット販売総数のカウンタがアプリケーション全体で使用されており、ログインページに次のようなコードがあるとします。

```
<cfset Application.totalTicketsSold = Application.totalTicketsSold + ticketOrder>
```

このコードが実行されると、ColdFusion では次の処理が行われます。

1 一時保管場所から Application.totalTicketsSold の現在の値が取得されます。

2 この値がインクリメントされます。

3 Application スコープに結果が戻されます。

Application.totalTicketsSold の初期値が 160 であるとした場合、2 つのチケット注文がほぼ同時に行われると、次の順序で処理が行われる可能性があります。

1 注文 1 がチケット販売総数を 160 と読み取ります。

2 注文 2 がチケット販売総数を 160 と読み取ります。

3 注文 1 が 160 枚に 5 枚の注文を追加して、165 という計算結果を得ます。

4 注文 2 が 160 枚に 3 枚の注文を追加して、163 という計算結果を得ます。

5 注文 1 が Application.totalTicketsSold に値 165 を保存します。

6 注文 2 が Application.totalTicketsSold に値 163 を保存します。

処理の結果、チケット販売総数は不正確な値になりました。このアプリケーションでは、収容人数を超すチケットを販売してしまうおそれがあります。

このような問題が発生しないように、カウンタをインクリメントするコードを次のようにロックします。

```
<cflock scope="Application" timeout="10" type="Exclusive">
    <cfset Application.totalTicketsSold = Application.totalTicketsSold + ticketOrder>
</cflock>
```

この cflock タグを使用すると、タグ本文が ColdFusion で処理されている間に、他のスレッドが Application スコープにアクセスできなくなります。したがって、2 番目のトランザクションは最初のトランザクションが終了するまで処理されず、次のような順序で処理が行われるようになります。

1 注文 1 がロックタグに到達し、Application スコープのロックを取得します。

2 注文 1 がチケット販売総数を 160 と読み取ります。

3 注文 2 がロックタグに到達します。Application スコープに対してアクティブなロックが存在するので、ColdFusion はロックが解除されるのを待ちます。

4 注文 1 が 160 枚に 5 枚の注文を追加して、165 という計算結果を得ます。

5 注文 1 が Application.totalTicketsSold に値 165 を保存します。

6 注文 1 がロックタグを終了します。Application スコープのロックが解除されます。

7 注文 2 が Application スコープのロックを取得し、処理を開始します。

8 注文 2 がチケット販売総数を 165 と読み取ります。

9 注文 2 が 165 枚に 3 枚の注文を追加して、168 という計算結果を得ます。

10 注文 2 が Application.totalTicketsSold に値 168 を保存します。

11 注文 2 がロックタグを終了し、Application スコープのロックを解放します。ColdFusion が他の注文を処理できるようになります。

処理の結果、Application.totalTickesSold は正しい値になりました。

複数の変数の一貫性の確保

アプリケーションでは、共有スコープの複数の変数を一度に設定することがよくあります。たとえば、ユーザーがフォームから複数の値を送信した場合などがあります。ユーザーがフォームを送信した後に、戻るボタンをクリックし、別のデータを入力してフォームを再送信した場合、前のセクションで説明したのと似たような過程で、2 回の送信データがアプリケーションで混同される可能性があります。

たとえば、Session スコープのショッピングカートに、注文品目に関する情報を保管しているとします。ユーザーが、セージグリーンのサイズ 36 の半ズボンを指定した品目選択ページを送信した後に、水色のサイズ 34 の半ズボンを指定して再送信した場合、アプリケーションで 2 つの注文の情報が混同されて、セージグリーンのサイズ 34 の半ズボンが注文されたかのように処理される可能性があります。

関連するセッション変数の設定コードをすべて単一の cflock タグ内に配置すれば、すべての変数がまとめて設定されます。つまり、すべての変数を設定する操作が**最小単位**、つまり単一の操作として扱われます。これは、トランザクションがすべて実行されるか、または何も実行されない、データベーストランザクションに似ています。この例では、最初の注文ですべての設定が行われた後に、2 番目の注文によってそれらが置き換えられます。

アプリケーションでのロックの使用例については、323 ページの「cflock の例」を参照してください。

ライトワンス変数での cflock タグの使用

設定が行えるのはアプリケーション内の **1** か所のみで、他のすべての場所では読み取り (UDF の場合は呼び出し) しか行えない変数またはユーザー定義関数名は、Session、Application、または Server スコープで読み取りや呼び出しを行う場合でも、cflock を使用する必要はありません。このようなデータはライトワンスと呼ばれます。たとえば、"Application.cfm" で Application スコープまたは Session スコープの変数を設定し、他のページでその変数を設定していない場合、その変数を設定するコードはロックしますが、その値を読み取るコードをロックする必要はありません。"Application.cfc" の対応する開始メソッド (たとえば Application スコープの変数であれば onApplicationStart) でその変数を設定している場合は、変数を設定するコードもロックする必要はありません。

ライトワンスデータを使用するコードをロックしないことには、アプリケーションのパフォーマンスが向上するという利点がありますが、リスクもあります。変数への書き込みが 1 回しか行われないことを保証する必要があります。たとえば、ユーザーがブラウザの更新ボタンや戻るボタンを押した場合でも、変数の再書き込みが行われないようにする必要があります。また、将来コードを修正するときに、アプリケーション内の複数の箇所でその変数を設定しないようにする必要がありますが、この制限を徹底するのは難しいことがあります。

cflock タグの使用

cflock タグを使用すると、複数のリクエストで同じコードが同時に実行されることが避けられるので、共有しているデータ構造体、ファイル、または CFX タグの整合性を保つことができます。cflock は、データにアクセスするコードセクションやデータを設定するコードセクションを保護するタグであり、変数自体を保護するタグではないことに注意してください。

コードへのアクセスを保護するには、そのコードを cflock タグで囲みます。次に例を示します。

```
<cflock scope="Application" timeout="10" type="Exclusive">
    <cfif not IsDefined("Application.number")>
        <cfset Application.number = 1>
    </cfif>
</cflock>
```

ロックのタイプ

cflock タグには、type 属性で指定する 2 種類のロックモードがあります。

排他的ロック (デフォルトのロックタイプ) ロックされたコードを処理できるリクエストは 1 つのみになります。あるリクエストが排他的ロックを取得している間、他のリクエストはそのタグ内のコードを実行できません。

セッション変数、アプリケーション変数、およびサーバー変数を作成または変更するコードは、すべて排他的な cflock タグで囲みます。

読み取り専用ロック 同じスコープまたは名前に対して排他的ロックが実行されていない場合は、複数のリクエストを同時に実行できます。あるリクエストが排他的ロックを取得している間、他のリクエストはそのタグ内のコードを実行できません。

セッション変数、アプリケーション変数、またはサーバー変数を単に読み取るまたはテストするコードは、読み取り専用の cflock タグで囲みます。読み取り専用ロックを指定するには、次のように、cflock タグで type="readOnly" 属性を設定します。

```
<cflock scope="Application" timeout="10" type="readOnly">
    <cfif IsDefined("Application.dailyMessage")>
        <cfoutput>#Application.dailyMessage#<br></cfoutput>
    </cfif>
</cflock>
```

読み取り専用のロックタグ内で共有変数を設定することは可能ですが、この場合、ロックの利点は失われてしまいます。したがって、読み取り専用の cflock タグの本文内で、セッション変数、アプリケーション変数、またはサーバー変数を設定しないように注意してください。

注意：ロックをネストしてロックのタイプを変更することはできません。つまり、読み取り専用ロックの中に、同じ名前またはスコープの排他的ロックをネストすることはできません。そのような排他的ロックは常にタイムアウトします。同じように、排他的ロックの中に同じ名前またはスコープの読み取り専用ロックをネストすることもできません。これは効果がありません。

スコープと名前のロック

cflock タグは、コードセクションへの同時アクセスを防ぐものであり、変数への同時アクセスを防ぐものではありません。同じ変数にアクセスするコードセクションが 2 つある場合は、それらのコードセクションを同期して、同時に実行されないようにする必要があります。それには、同じ scope 属性または name 属性を使用してロックを識別します。

注意：排他的ロックは、必ずしも識別する必要はありません。識別子を省略すると匿名ロックになり、cflock タグブロック内のコードを他のコードと同期できなくなります。匿名ロックは、1 つのコードブロックでのみ使用されるリソースを保護するときにはエラーの原因になりませんが、よいプログラミングスタイルとはいえません。読み取り専用ロックは、常に識別する必要があります。

scope 属性によるデータアクセスの制御

セッション変数、アプリケーション変数、またはサーバー変数にアクセスするコードをロックする場合は、cflock の scope 属性を使用してアクセスを同期します。

この属性には、次のいずれかの値を設定できます。

スコープ	説明
Server	サーバー上でこの属性を持つすべてのコードセクションで単一のロックを共有します。

スコープ	説明
Application	同じアプリケーションでこの属性を持つすべてのコードセクションで単一のロックを共有します。
Session	アプリケーションの同じセッションで実行される、この属性を持つすべてのコードセクションで単一のロックを共有します。
Request	同じリクエストで実行される、この属性を持つすべてのコードセクションで単一のロックを共有します。このスコープを使用するのは、アプリケーションで cfthread タグを使用して単一のリクエストで複数のスレッドを作成している場合のみです。Request スコープをロックすると、Variables スコープデータへのアクセスもロックされます。Request スコープのロックの詳細については、332 ページの「スレッドデータやリソースに対するアクセスのロック」を参照してください。

複数のコードセクションでロックを共有する場合は、次のルールが適用されます。

- type 属性が Exclusive である cflock タグブロック内のコードが実行されている場合、同じ scope 属性を持つ cflock タグブロック内のコードは実行できません。そのコードは、排他的ロックを持つコードが完了するまで待機する必要があります。
- type 属性が readOnly である cflock タグブロック内のコードが実行されている場合、scope 属性が同じで type 属性が readOnly である他の cflock タグブロック内のコードは実行できますが、scope 属性が同じで type 属性が Exclusive であるブロックは実行できず、読み取り専用ロックを持つすべてのコードが完了するまで待機する必要があります。ただし、読み取り専用ロックがアクティブで、スコープまたは名前が同じである排他的ロックのコードが実行を待機している場合、排他的リクエストがキューに入れられた後で作成された、スコープまたは名前が同じである読み取り専用リクエストは、排他的ロックのコードが実行されて完了するまで待機する必要があります。

name 属性によるファイルや CFX タグへのアクセスのロック制御

cflock の name 属性を使用してロックを識別することもできます。この属性は、ファイルアクセスを管理するコードや、スレッドセーフでない CFX タグを呼び出すコードをロックして保護するために使用します。

name 属性を使用するときは、特定のファイルや CFX タグにアクセスするすべてのコードセクションに対して同じ名前を指定します。

ロックのタイムアウトの制御および最小化

cflock タグには、timeout 属性を指定してください。timeout 属性は、ロックが取得できない場合にロックが解放されるのを待つ最大時間を秒単位で指定します。デフォルトでは、タイムアウト期間内にロックが取得できないと、ロックタイプの例外エラーが生成されます。この例外エラーは、cftry タグおよび cfcatch タグを使用して処理できます。

cflock の throwOnTimeout 属性を No に設定した場合は、タイムアウトになると、</cflock> 終了タグに続く行から処理が続行されます。ロックを取得する前にタイムアウトが発生した場合、cflock タグ本文内のコードは実行されません。したがって、常に実行する必要がある CFML では throwOnTimeout 属性を使用しないでください。

通常は、ロックの取得に数秒以上かかることはありません。しかし、タイムアウト期間が長すぎると、リクエストスレッドが長時間にわたってブロックされるので、スループットが大幅に低下する可能性があります。タイムアウト値は、タイムアウトが発生しすぎない範囲で可能な限り小さい値にします。

必要のないタイムアウトが発生しないように、ロックするコードの数はできるだけ減らしてください。可能な限り、ビジネスロジックやデータベースクエリーはロックせず、変数を設定または読み取るコードのみをロックしてください。次のような方法が役立ちます。

1 時間のかかるアクティビティは cflock タグの外で実行します。

2 その結果を Variables スコープの変数に代入します。

3 その Variables スコープ変数の値を、cflock ブロックの中で共有スコープ変数に代入します。

たとえば、クエリーの結果をセッション変数に割り当てる場合は、まず、ロックされていないコードで Variables スコープ変数を使用してクエリー結果を取得します。それから、ロックされたコードセクションの中で、セッション変数にクエリー結果を代入します。この方法を次のコードに示します。

```
<cfquery name="Variables.qUser" datasource="#request.dsn#">
    SELECT FirstName, LastName
    FROM Users
    WHERE UserID = #request.UserID#
</cfquery>
<cflock scope="Session" timeout="5" type="exclusive">
    <cfset Session.qUser = Variables.qUser>
</cflock>
```

ロックの規模の検討

ロック方法を検討するときは、コードブロックを小さくして多数のロックを使用するのか、コードブロックを大きくして少数のロックを使用するのかを検討します。ロックの規模を決める単純なルールはないので、さまざまなオプションでパフォーマンスをテストして判断します。ただし、次の問題に注意してください。

- コードブロックを大きくすると、そのブロックの処理時間が長くなります。したがって、ロックが解放されるまでアプリケーションが待機する回数が増える可能性があります。
- ロックを行うには時間がかかります。ロックが多いほど、コードのロックに要する時間が増えます。

ロックのネストとデッドロックの防止

cflock タグのネストやネーミングが不適切であると、デッドロック (コードのブロック) が発生することがあります。ロックをネストするときは、一貫した順序で cflock タグをネストし、一貫したロックスコープ (または名前) を使用する必要があります。

デッドロックとは、ページ内の特定のセクションがロックされて、どのリクエストからも実行できなくなった状態のことをいいます。ページ内の保護されたセクションへのリクエストは、タイムアウトになるまですべてブロックされます。次の表に、デッドロックが発生するシナリオを示します。

ユーザー 1	**ユーザー 2**
Session スコープをロックします。	Application スコープをロックします。
Application スコープをロックしようとしますが、Application スコープは既にユーザー 2 によってロックされています。	Session スコープをロックしようとしますが、Session スコープは既にユーザー 1 によってロックされています。

いずれのリクエストも、もう一方のリクエストが完了するのを待っているので、処理を続行できません。両方のリクエストがデッドロックの状態になっています。

デッドロックが発生した場合は、いずれのユーザーもデッドロックから抜け出せなくなります。これは、ロックがタイムアウトしてデッドロックが解決されるまで、両方のリクエストの実行がブロックされるからです。

また、異なるタイプのロックをネストした場合にもデッドロックが発生することがあります。たとえば、読み取り専用ロックの中に、同じスコープまたは名前の排他的ロックをネストした場合などがあります。

デッドロックを防止するには、一貫した順序でコードセクションをロックし、また一貫した名前をロックに付ける必要があります。特に、Server、Application、および Session スコープへのアクセスをロックする場合は、次の順序に従ってください。

1 Session スコープをロックします。cflock タグで scope="Session" を指定します。

2 Application スコープをロックします。cflock タグで scope="Application" を指定します。

3 Server スコープをロックします。cflock タグで scope="Server" を指定します。

4 Server スコープのロックを解除します。

5 Application スコープのロックを解除します。

6 Session スコープのロックを解除します。

注意：特定のスコープをロックする必要がない場合は、上記のリストのロックおよびロック解除の手順を省略することができます。たとえば、Server スコープをロックする必要がない場合は、手順 3 と 4 を省略することができます。

Request スコープへの共有変数のコピー

次の方法を使用すれば、リクエスト時に共有スコープ変数を何度もロックせずに済みます。

1. "Application.cfc" の onRequestStart メソッドまたは "Application.cfm" ページで、排他的ロックを使用して、共有スコープの変数を Request スコープにコピーします。
2. ColdFusion ページでリクエストを行う間は、Request スコープの変数を使用します。
3. "Application.cfc" の onRequestEnd メソッドまたは "OnRequestEnd.cfm" ページで、排他的ロックを使用して、Request スコープの変数を共有スコープにコピーします。

この方法を使用した場合は、最後のリクエストのみが共有スコープに反映されます。たとえば、2 つのリクエストが同時に実行されており、両方のリクエストで共有スコープからデータ値がコピーされ、変更が加えられた場合は、最後に完了したリクエストのデータが共有スコープに保存され、それ以前のリクエストのデータは保存されません。

アプリケーション変数の効果的なロック

アプリケーション変数をロックすると、サーバーのパフォーマンスが低下することがあります。これは、Application スコープの変数を使用するすべてのリクエストが単一のロックを待機しなければならないからです。この問題は、一度設定された後は読み取り専用になるライトワンス変数でも発生します。その値を読み取るためには、変数が存在することを確認し、存在しない場合は設定する必要があるからです。

次のような方法を使用してアプリケーション変数の存在をテストし、存在しない場合は設定するようにすれば、この問題を最小限に抑えることができます。

1. 変数が初期化されているかどうかを示すフラグ変数を、Application スコープに用意します。読み取り専用ロックを使用して、このフラグが存在するかどうかを確認し、その結果をローカル変数に代入します。
2. cflock ブロックの外で、ローカル変数の値をテストします。
3. アプリケーション変数が初期化されていないことが示されている場合は、Application スコープの排他的ロックを取得します。
4. そのロックの中で Application スコープのフラグを再テストし、手順 1 ～ 4 の間に別のページによって変数が設定されていないことを確認します。
5. 変数が設定されていない場合は設定し、Application スコープのフラグを true に設定します。
6. 排他的ロックを解除します。

この方法を次のコードに示します。

```
<!--- Initialize local flag to false. --->
<cfset app_is_initialized = False>
<!--- Get a readonly lock --->
<cflock scope="application" type="readonly">
    <!--- read init flag and store it in local variable --->
    <cfset app_is_initialized = IsDefined("APPLICATION.initialized")>
</cflock>
<!--- Check the local flag --->
<cfif not app_is_initialized >
<!--- Not initialized yet, get exclusive lock to write scope --->
    <cflock scope="application" type="exclusive">
        <!--- Check nonlocal flag since multiple requests could get to the
                exclusive lock --->
        <cfif not IsDefined("APPLICATION.initialized") >
            <!--- Do initializations --->
            <cfset APPLICATION.varible1 = someValue >
             ...
            <!--- Set the Application scope initialization flag --->
            <cfset APPLICATION.initialized = "yes">
        </cfif>
    </cflock>
</cfif>
```

cflock の例

次の例では、さまざまな状況で cflock ブロックを使用する方法を示します。

アプリケーション変数、サーバー変数、およびセッション変数を使用する例

この例では、cflock を使用して、Application、Server、および Session スコープ内の変数が正しく更新されるようにします。

この例では、ロックがタイムアウトした場合の例外は処理していません。したがって、ロックタイムアウトに関するデフォルトの例外エラーページがユーザーに表示されます。

次のサンプルコードは "Application.cfm" ファイルで使用できます。

```
<cfapplication name="ETurtle"
    sessiontimeout=#createtimespan(0,1,30,0)#
    sessionmanagement="yes">

<!--- Initialize the Session and Application
variables that will be used by E-Turtleneck. Use
the Session lock scope for the session variables. --->

<cflock scope="Session"
    timeout="10" type ="Exclusive">
    <cfif not IsDefined("session.size")>
        <cfset session.size = "">
    </cfif>
    <cfif not IsDefined("session.color")>
        <cfset session.color = "">
    </cfif>
</cflock>

<!--- Use the Application scope lock for the Application.number variable.
This variable keeps track of the total number of turtlenecks sold.
The following code implements the scheme shown in the Locking Application
variables effectively section --->

<cfset app_is_initialized = "no">
```

```
<cflock scope="Application" type="readonly">
    <cfset app_is_initialized = IsDefined("Application.initialized")>
</cflock>
<cfif not app_is_initialized >
    <cflock scope="application" timeout="10" type="exclusive">
        <cfif not IsDefined("Application.initialized") >
            <cfset Application.number = 1>
            <cfset Application.initialized = "yes">
        </cfif>
    </cflock>
</cfif>

<!--- Always display the number of turtlenecks sold --->

<cflock scope="Application"
    timeout="10"
    type ="ReadOnly">
    <cfoutput>
    E-Turtleneck is proud to say that we have sold
    #Application.number# turtlenecks to date.
    </cfoutput>
</cflock>
```

残りのサンプルコードは、顧客が注文を行うアプリケーションページで使用します。

```
<html>
<head>
<title>cflock Example</title>
</head>

<body>
<h3>cflock Example</h3>

<cfif IsDefined("Form.submit")>

<!--- Lock session variables --->
<!--- Note that we use the automatically generated Session
    ID as the order ID --->
<cflock scope="Session"
    timeout="10" type="ReadOnly">
    <cfoutput>Thank you for shopping E-Turtleneck.
    Today you have chosen a turtleneck in size
    <b>#form.size#</b> and in the color <b>#form.color#</b>.
    Your order ID is #Session.sessionID#.
    </cfoutput>
</cflock>

<!--- Lock session variables to assign form values to them. --->

<cflock scope="Session"
    timeout="10"
    type="Exclusive">
    <cfparam name=Session.size default=#form.size#>
    <cfparam name=Session.color default=#form.color#>
</cflock>
<
!--- Lock the Application scope variable application.number to
update the total number of turtlenecks sold. --->

<cflock scope="Application"
    timeout="30" type="Exclusive">
    <cfset application.number=application.number + 1>
</cflock>
```

```
<!--- Show the form only if it has not been submitted. --->
<cfelse>
<form action="cflock.cfm" method="Post">

<p> Congratulations! You have just selected
the longest-wearing, most comfortable turtleneck
in the world. Please indicate the color and size
you want to buy.</p>

<table cellspacing="2" cellpadding="2" border="0">
<tr>
<td>Select a color.</td>
<td><select type="Text" name="color">
        <option>red
        <option>white
        <option>blue
        <option>turquoise
        <option>black
        <option>forest green
        </select>
    </td>
</tr>
<tr>
    <td>Select a size.</td>
    <td><select type="Text" name="size">
        <option>small
        <option>medium
        <option>large
        <option>xlarge
        </select>
    </td>
</tr>
<tr>
    <td></td>
    <td><input type="Submit" name="submit" value="Submit">
    </td>
</tr>
</table>
</form>
</cfif>

</body>
</html>
```

注意：この簡単な例では、"Application.cfm" ページによって Application.number 変数の値を表示しています。"Application.cfm" ファイルは各 ColdFusion ページのどのコードよりも先に処理されるので、送信ボタンを押した後で表示される値には、新規の注文内容は反映されません。この問題の解決策としては、"OnRequestEnd.cfm" ページを使用して、アプリケーションの各ページの下にその値を表示する方法があります。

ファイルシステムへのアクセスを同期する例

次の例では、cflock ブロックを使用してファイルシステムへのアクセスを同期化する方法を示します。cflock タグを使用して cffile タグを保護しているので、別のリクエストで同じタグを使用して書き込み用にファイルを開いているときには、データを追加できません。

Append 操作に 30 秒以上かかると、このコードの排他的ロックを取得するために待機しているリクエストがタイムアウトになる可能性があります。また、この例では、name 属性にダイナミックな値を使用することで、ファイルごとに別個のアクセス制御を行っています。したがって、あるファイルへのアクセスをロックしても、その他のファイルへのアクセスが遅れることはありません。

```
<cflock name=#filename# timeout=30 type="Exclusive">
    <cffile action="Append"
        file=#fileName#
        output=#textToAppend#>
</cflock>
```

ColdFusion Extension を保護する例

次の例では、スレッドセーフでない CFX タグのカスタムタグラッパーを構築する方法を示します。このラッパーでは、cflock タグの中で、スレッドセーフでない CFX タグに属性を渡しています。

```
<cfparam name="Attributes.AttributeOne" default="">
<cfparam name="Attributes.AttributeTwo" default="">
<cfparam name="Attributes.AttributeThree" default="">

<cflock timeout=5
        type="Exclusive"
        name="cfx_not_thread_safe">
    <cfx_not_thread_safe attributeone=#attributes.attributeone#
        attributetwo=#attributes.attributetwo#
        attributethree=#attributes.attributethree#>
</cflock>
```

ColdFusion スレッドの使用

Adobe ColdFusion では、スレッドを使用することで、ColdFusion ページまたは CFC で複数のストリームを同時に実行することができます。

ColdFusion スレッドについて

スレッドとは、独立した実行ストリームのことです。ページまたは CFC では、複数のスレッドを同時かつ非同期に実行できるので、CFML で非同期処理を行うことができます。

スレッドは、次の 2 つのタイプの広範なアクティビティに役立ちます。

- 複数のアクションが同時に発生する可能性がある場合。
- 次のアクションを開始するために現在のアクションの完了を待つ必要がない場合。

スレッドの代表的な使用例を次に示します。

- 応答に時間がかかる複数の外部ソースから情報を集めるアプリケーションでは、各ソースから情報を取得するコードを別個のスレッドで実行できます。これによって、すべてのリクエストをすばやく開始することができます。アプリケーションが待つ必要があるのは最後の応答のみであり、次のリクエストを開始するために前のリクエストの応答を待つ必要はありません。この方法は、RSS または Atom のフィードアグリゲータで利用できます。
- 多数のメールメッセージを送信するページでは、メールメッセージを送信するコードを別個のスレッドで実行します。これによって、メール送信コードの完了を待たずにページの処理を再開できます。ページレベルの処理が完了し、アプリケーションで別のページの処理が開始された後でも、メールメッセージを送信するスレッドは処理を続行できます。
- アプリケーションでは、ユーザーがサイトにログインするたびに、ユーザーデータの保守 (更新クエリーの使用、レコードの削除など) を行います。保守作業を別のスレッドで実行すれば、ユーザーはログイン後すぐに応答を取得でき、更新作業の完了を待たずに済みます。

ColdFusion でページが処理されるときには、単一のスレッド (ページスレッド) でページが実行されます。cfthread タグを使用すると、ページスレッドから独立して実行可能な追加のスレッドを作成できます。また、スレッド処理を同期させて、たとえば作成したスレッドが処理を完了するまでページスレッドを待機させることもできます。

ColdFusion スレッドの作成と管理

ColdFusion スレッドを作成および管理するには、cfthread タグと Sleep 関数を使用します。スレッドの管理機能を使用すれば、次の操作が行えます。

- スレッドの実行を開始できます。
- スレッドの処理を一時的に中断できます。あるスレッドが別のスレッドの処理を待つ必要があり、その後、スレッドを結合せずに両方のスレッドの処理を続行する必要がある場合は、このアクションが役立ちます。
- スレッドを終了できます。スレッドの終了は、スレッドでエラーが発生した場合や、スレッドが長時間処理中になっている場合などに行われます。
- 他のスレッドの処理が完了するのを待ってから、ページまたはスレッドの処理を再開できます。これをスレッドの結合 (join) と呼びます。スレッドの結合は、あるスレッドで別のスレッドの結果が必要な場合などに行われます。たとえば、複数のスレッドを使用して複数のニュースフィードを取得している場合は、すべてのフィードスレッドを結合してその結果を表示することができます。

各スレッドでは、cfthread タグ本文に記述されているコードが実行されます。通常は、タグ本文コードの処理が完了するとスレッドが終了します。

スレッドの開始

スレッドを開始するには、action 属性に run という値を指定した cfthread タグを使用します。cfthread タグ本文内の CFML コードは別個のスレッドで実行され、ページリクエストスレッドの処理が再開されます。スレッドを作成できるのはページスレッドのみです。cfthread タグで作成されたスレッドの中で子スレッドを作成することはできないので、スレッドの多重ネストは行えません。

スレッドを開始するときは、オプションで高、標準 (デフォルト)、または低の優先度レベルを指定して、プロセッサがそのスレッドに費やす相対時間を指定できます。ページレベルのコードは常に標準の優先度で実行されるので、ページよりも多いまたは少ない処理時間をスレッドに与えることができます。

スレッド属性の使用方法の詳細については、331 ページの「Attributes スコープとスレッド属性」を参照してください。

スレッドの中断

場合によっては、あるスレッドを処理するときに別のスレッドの一部の処理が完了するのを待つ必要があるが、すべての処理が完了するまで待てないために、それらのスレッドを結合できないことがあります。たとえば、複数のスレッドで必要な初期化処理を行った後に、追加の処理を続行するスレッドなどが考えられます。他のスレッドは、初期化処理が完了するまで待つ必要があります。

このような同期を行うには、Sleep 関数を使用するか、action 属性に sleep を指定した cfthread タグを使用します。これらを使用すると、指定の期間だけスレッド処理を中断できます。たとえば、コードループで条件変数をテストし、特定の期間だけスリープした後に、その条件を再度テストします。条件が true になった場合 (または特定の値に達したか、他のテストが true になった場合) は、ループを終了して、スレッドの処理を再開します。

次の例では、あるスレッドでスリープ関数を使用して、別のスレッドの特定のアクションが完了するのを待ちます。

```
<!--- ThreadA loops to simulate an activity that might take time. --->
<cfthread name="threadA" action="run">
        <cfset thread.j=1>
        <cfloop index="i" from="1" to="1000000">
        <cfset thread.j=thread.j+1>
    </cfloop>
</cfthread>

<!--- ThreadB loops, waiting until threadA finishes looping 40000 times.
            the loop code sleeps 1/2 second each time. --->
<cfthread name="threadB" action="run">
    <cfscript>
        thread.sleepTimes=0;
        thread.initialized=false;
        while ((threadA.Status != "TERMINATED") && (threadA.j < 400000)) {
            sleep(500);
            thread.sleeptimes++;
        }
        // Don't continue processing if threadA terminated abnormally.
        If (threadA.Status != "TERMINATED") {
            thread.initialized=true;
            // Do additional processing here.
        }
    </cfscript>
</cfthread>

<!Join the page thread to thread B. Don't join to thread A.--->
<cfthread action="join" name="threadB" timeout="10000" />

<!--- Display the thread information. --->
<cfoutput>
    current threadA index value: #threadA.j#<br />
    threadA status: #threadA.Status#<br>
    threadB status: #threadB.Status#<br>
    threadB sleepTimes: #threadB.sleepTimes#<br>
    Is threadB initialized: #threadB.initialized#<br>
</cfoutput>
```

スレッドの終了

処理が完了できなくなった (ハングした) スレッドは、システムリソースを占有し続けます。したがってアプリケーションでは、ハングしたスレッドがないか確認し、ある場合は終了することをお勧めします。また、処理に時間がかかりすぎてアプリケーションやサーバーの応答性を著しく低下させているスレッドについても、終了を検討してください。

スレッドを終了するには、次のコードのように、action 属性に terminate を指定した cfthread タグを使用します。

```
<!--- Thread1 sleeps to simulate an activity that might hang. --->
<cfthread name="thread1" action="run">
        <cfset thread.j=1>
    <cfset sleep(50000) >
    </cfthread>

<!--- Thread2 loops to simulate an activity that takes less time. --->
<cfthread name="thread2" action="run">
        <cfset thread.j=1>
        <cfloop index="i" from="1" to="10">
        <cfset thread.j=thread.j+1>
    </cfloop>
</cfthread>

<!--- The page thread sleeps for 1/2 second to let thread
         processing complete. --->
<cfset sleep(500) >

<!--- The page thread loops through the threads and terminates
            any that are still running or never started.
        Note the use of the cfthread scope and associative array
        notation to reference the dynamically named threads without
        using the Evaluate function. --->
<cfloop index="k" from="1" to="2">
<cfset theThread=cfthread["thread#k#"]>
    <cfif ((theThread.Status IS "RUNNING") || (theThread.Status IS "NOT_STARTED"))>
        <cfthread action="terminate" name="thread#k#" />
    </cfif>
</cfloop>

<!--- Wait 1/2 second to make ensure the termination completes --->
<cfset sleep(500) >

<!--- Display the thread information. --->
<cfoutput>
    thread1 index value: #thread1.j#<br />
    thread1 status: #thread1.Status#<br>
    thread2 index value: #thread2.j#<br />
    thread2 status: #thread2.Status#<br>
</cfoutput>
```

注意：ColdFusion サーバーモニタを使用して、ハングしたスレッドの確認と終了を自動的に行うこともできます。

スレッドの結合

複数のスレッドを結合するには、action 属性に join を指定した cfthread タグを使用します。スレッドの結合は、他のいくつかのスレッドが完了してから現在のスレッドを続行したい場合に便利です。たとえば、ページで複数のスレッドを開始してそれらを結合すれば、スレッドの処理結果をページで使用することができます。join アクションを実行すると、デフォルトでは、指定されたすべてのスレッドが処理を完了するまで、現在のスレッドの処理が中断されます。

timeout 属性を使用すれば、結合するスレッドの完了を待つ期間を、ミリ秒単位で指定できます。指定した時間までに完了していないスレッドがあっても、そのスレッドが完了するのを待たずに、現在のスレッドで処理が再開されます。

たとえば次のコードは、現在のスレッド (多くの場合、メインページスレッド) に 3 つのスレッドを結合します。現在のスレッドは、他のスレッドの完了を最大 6 秒間待った後、完了していないスレッドがあっても処理を再開します。

```
<cfthread action="join" name="t1,t2,t3" timeout="6000"/>
```

timeout 属性値を 0 (デフォルト値) にすると、結合するスレッドがすべて完了するまで、現在のスレッドは待機し続けます。現在のスレッドがページスレッドである場合は、ページのタイムアウトが指定されていても、スレッドが結合されるまでページは待機し続けます。通常は、ハングしたスレッドがあっても処理が再開されるように、timeout 値を指定します。

スレッドデータの使用

単一のリクエストで複数のスレッドが同時に実行されることがあるので、アプリケーションでは、あるスレッドのデータが別のスレッドのデータに不適切な影響を与えないように考慮する必要があります。ColdFusion には、スレッドデータの管理に使用できる複数のスコープや、ページレベルデータにスレッドがアクセスする際の問題を回避するためのリクエストレベルのロックメカニズムが用意されています。また、スレッド固有の出力やスレッドに関する情報 (スレッドのステータスや処理時間など) を提供するメタデータ変数も用意されています。

スレッドのスコープ

各スレッドには、次の 3 つの特殊なスコープがあります。

- スレッドローカルスコープ
- Thread スコープ
- Attributes スコープ

スレッドローカルスコープ

スレッドローカルスコープは、スレッドでのみ使用可能な変数が含まれている暗黙的なスコープであり、スレッドの有効期間のみ存在します。cfthread タグ本文内でスコープ名の接頭辞を指定せずに定義した変数は、すべてスレッドローカルスコープに配置され、他のスレッドによるアクセスや変更は行えません。

スレッドローカル変数を作成するには、次の行のように、cfthread タグ本文でスコープ接頭辞を指定せずに変数に代入します。

```
<cfset var index=1>
<cfset index=1>
```

この 2 つの行はほぼ同じですが、var キーワードを使用する代入コードは、cfthread タグの直後 (他の CFML タグよりも前) に置く必要がある点が異なります。

Thread スコープ

Thread スコープには、スレッド固有の変数と、スレッドに関するメタデータが含まれています。このスコープにデータを書き込めるのは、このスコープを所有しているスレッドのみですが、このスコープの変数値の読み取りは、リクエスト内のページスレッドや他のすべてのスレッドでも行えます。スレッドの処理が完了する前にページが完了しても、ページおよびそのページから開始されたすべてのスレッドが完了するまで、Thread スコープのデータは存続します。

Thread スコープ変数を作成するには、cfthread タグ本文で、キーワード Thread またはスレッドの名前 (myThread など) を接頭辞として使用します。次に示す Thread スコープ変数の作成例は同等です。

```
<cfset Thread.myValue = 27>
<cfset myThread.myValue = 27>
```

スレッドの外部からスレッドの Thread スコープ変数にアクセスするには、次の例のように、スレッドの名前を変数の接頭辞として使用します。

```
<cfset nextValue=myThread.myValue + 1>
```

Thread スコープ変数は、スレッドを作成したページ、またはそのページで作成された他のスレッドでのみ使用可能です。その他のページからデータにアクセスすることはできません。あるページから別のページの Thread スコープデータにアクセスする必要がある場合は、そのデータをデータベースまたはファイルに格納して、そこからアクセスする必要があります。

各スレッドの Thread スコープは、cfthread という特別なスコープのサブスコープです。このスコープの存続期間は、リクエストの存続期間か、リクエストによって開始された最後のスレッドが完了するまでの期間の、いずれか長いほうになります。したがって、myThread1 と myThread2 という 2 つのスレッドがある場合は、すべてのスレッドとリクエストが完了

するまで、cfthread.myThread1 および cfthread.myThread2 としてそれらの Thread スコープにアクセスできます。ほとんどの場合、直接 cfthread スコープを使用する必要はありませんが、次の 2 つの場合に cfthread スコープ名を使用できます。

1 スレッド名をダイナミックに生成する場合は、次のコードのように、cfthread スコープで連想配列表記法を使用すれば、Evaluate 関数を使用せずに済みます。

```
<cfset threadname="thread_#N#">
...
<!--- The following two lines are equivalent --->
<cfset threadscopeForNthThread = cfthread[threadname] >
<cfset threadscopeForNthThread = Evaluate(threadname) >
```

2 スレッドと同じ名前の変数が Variables スコープにある場合は、スレッド名の前に cfthread を付ければ、そのスレッドの Thread スコープにのみアクセスできます。スコープ名を付けないと、Variables スコープ変数にアクセスするか、エラーが発生します。

Attributes スコープとスレッド属性

Attributes スコープには、スレッドに個別に渡された属性や、attributeCollection 属性で渡された属性が含まれています。Attributes スコープは、そのスレッド内でのみ、スレッドの有効期間にのみ使用可能です。

スレッドに属性変数が渡されるときには、すべての属性変数の完全なコピー (ディープコピー) が作成されます。したがって、スレッド内の変数の値は、ページスレッドなどの他のスレッド内の対応する変数の値から独立しています。たとえば、CFC インスタンスを属性としてスレッドに渡すと、スレッドを作成したときの This スコープの内容を含めた、CFC の新しい完全なコピーがスレッドに渡されます。スレッドの外で元の CFC を変更しても (たとえば CFC 関数を呼び出しても)、スレッドに渡されたコピーには影響しません。同様に、スレッド内の CFC インスタンスを変更しても、元の CFC インスタンスには影響しません。

データは複製されるので、他のスレッドによって属性値が変更されることはありません。したがって、スレッドに渡される値はスレッドセーフです。データを複製したくない場合は、個別の属性や attributeCollection 属性を通じてスレッドに渡す方法は使用できません。代わりに、スレッドがアクセスできるスコープにデータを保持します。属性としてスレッドに渡せないオブジェクトの例として、複製できないシングルトン CFC があります。シングルトン CFC は、共有スコープに入れて、スレッドによってアクセスする必要があります。詳細については、331 ページの「他のスコープの使用」を参照してください。

すべての属性で値がコピーされるので、たとえば次のコードのように、ループ内で複数のスレッドを作成するときに、特定の属性に対してスレッドごとに異なる値をダイナミックに渡すことができます。作成される各スレッドでは、ディレクトリ内の異なるファイルがコピーされます。

```
<cfloop query="dir">
    <cfset threadname = "thread_" & #i#>
    <cfset i=i+1>
    <cfthread name="#threadname#" filename="#dir.name#">
        <cffile action="COPY" source="#src#\#filename#"
            destination="#dest#\#filename#\">
    </cfthread>
</cfloop>
```

他のスコープの使用

スレッドは、すべての ColdFusion スコープにアクセスできます。ページによって実行されるすべてのスレッドは、同じ Variables および This スコープを共有します。リクエスト内で実行されるすべてのスレッドは、同じ Form、URL、Request、CGI、Cookie、Session、Application、Server および Client スコープを共有します。複数のスレッドがスコープ内のデータの変更を試みる可能性がある場合は、これらのスコープへのアクセスをロックする必要があります。ロックしないと、スレッド間でデッドロックが発生する場合があります。詳細については、332 ページの「スレッドデータやリソースに対するアクセスのロック」を参照してください。

スレッドはすべてのスコープにアクセスできますが、リクエストページの処理が終了している場合は、Session、Cookie、Request などのスコープに書き込めないことがあります。

スコープの優先順位

cfthread タグ本文で、変数にスコープ接頭辞を指定しなかった場合は、次の順序でスコープが検索されて、所属するスコープが確認されます。

1. 関数ローカル (スレッド内の関数定義内のみ)
2. スレッドローカル
3. Attributes
4. Variables
5. Thread/cfthread

その他のスコープは、標準の検索順序で確認されます。

スレッドデータやリソースに対するアクセスのロック

アプリケーションで複数のスレッドを使用する場合は、スレッドセーフでない共有リソースがスレッドで同時に使用または変更されないようにしてください。たとえば、次のような点に注意してください。

- Variables または Request スコープの変数が複数のスレッドによって変更される可能性がある場合は、Request スコープのロックを使用して、その変数を使用するコードへのアクセスを制御し、デッドロックや競合状態を回避します。同様に、Variables スコープのビルトインデータ構造体やサブスコープ (Forms 変数など) が複数のスレッドによって変更される可能性がある場合は、それらにアクセスするコードを Request スコープのロックで囲みます。
- その他の共有リソースについても、複数のスレッドによって同時にアクセスされないようにします。たとえば、複数のスレッドで同一の FTP 接続を使用することはできません。この動作を回避するには、リソースを使用するコードを名前付きの cflock タグ内に配置します。同じリソースを使用するすべてのコードは、同じ name 属性を使用した cflock タグで囲みます。

コードのロックの詳細については、cflock および 316 ページの「cflock によるコードのロック」を参照してください。

メタデータ変数

Thread スコープには、スレッドに関する情報 (メタデータ) を提供する次の変数が含まれています。

変数	説明
Elapsedtime	これまでスレッドの処理に費やされたプロセッサ時間。
Error	cfcatch タグでアクセス可能なキーが含まれている ColdFusion エラー構造体。この変数が値を持つのは、処理されないエラーがスレッドの実行中に発生した場合のみです。スレッドエラーの処理の詳細については、334 ページの「ColdFusion スレッドエラーの処理」を参照してください。
Name	スレッド名。
Output	スレッドによって生成される出力テキスト。スレッドは出力を直接表示することができません。詳細については、334 ページの「スレッド出力の処理」を参照してください。
Priority	スレッド作成時に指定したスレッド処理の優先度。
Starttime	スレッドが処理を開始した時刻。
Status	スレッドの現在のステータス。アプリケーションにおける Status の使用方法については、333 ページの「スレッドステータスの使用」を参照してください。

Thread スコープの他の変数と同様に、スレッド名を変数の接頭辞として指定することで、ページのすべてのスレッドからスレッドメタデータを使用できます。たとえば、myThread1 スレッドの現在の経過時間をページスレッドで取得するには、myThread1.ElapsedTime 変数を使用します。

メタデータが使用可能な期間は、スレッドが作成されたときから、ページおよびそのページで開始されたすべてのスレッドが処理を完了するときまでです。したがって、スレッドが完了する前にページが完了しても、メタデータは依然として使用可能です。これによって、スレッドの処理中または処理後に、スレッド出力、エラー情報、処理情報を取得できます。

スレッドの操作

マルチスレッドアプリケーションで使用されるビルディングブロックには、次のものがあります。

- ループ内でのスレッドの開始
- スレッドの処理ステータスに関する情報の取得
- スレッドの結果の表示
- スレッドエラーの処理
- スレッドでのデータベーストランザクションの使用

ループ内でのスレッドの開始

スレッドは非同期で実行されるので、ページレベルの変数はスレッドの実行中に変化する可能性があります。したがって、cfloop の中でスレッドを開始し、スレッド内のコードでループイテレータ (インデックス変数、クエリー名、リストアイテムなど) を使用する場合は、ループイテレータを属性としてスレッドに渡します。

ループ内でスレッドを使用する例を次に示します。この例では、インデックスを使用した cfloop タグで、5 つのスレッドを開始しています。各スレッドには、threadIndex 属性を通じて現在のループインデックスの値を渡しています。スレッドの中では、スレッドの threadIndex 属性の値と、ページの cfloop インデックス (pageIndex) の現在の値を、配列エントリとして追加しています。ページでは、スレッドを結合した後に、配列の内容を表示しています。この例を実行すると (何回か実行すると特によくわかりますが)、配列にデータが保存されるときには、pageIndex の値が threadIndex の値よりも大きくなっている場合があることに気付きます。また、いくつかのスレッドで pageIndex が同じ値になっている場合もあります。threadIndex の値は、どのスレッドでも常に正しい値になります。

```
<cfloop index="pageIndex" from="1" to="5">
    <cfthread name="thr#pageIndex#" threadIndex="#pageIndex#" action="run">
        <cfset Variables.theOutput[threadIndex]="Thread index attribute:" &
            threadIndex & "   Page index value: " & pageIndex>
    </cfthread>
</cfloop>

<cfthread action="join" name="thr1,thr2,thr3,thr4,thr5" timeout=2000/>

<cfloop index="j" from="1" to="5">
    <cfoutput>#theOutput[j]# <br /></cfoutput>
</cfloop>
```

スレッドステータスの使用

Thread スコープの status メタデータ変数を使用すると、ページやページ内で開始されたスレッドで、任意のスレッドのステータスを確認できます。スレッドが異常終了していたり、ハングしている場合には、ページ処理コードで必要なアクションを行うことができます。status 変数が取りうる値は、次のとおりです。

値	説明
NOT_STARTED	スレッドはキューで待機しており、まだ処理されていません。
RUNNING	スレッドは正常に実行中です。
TERMINATED	次のいずれかの処理の結果として、スレッドは実行を停止しました。 • terminate アクションを持つ cfthread タグによってスレッドが停止した。 • スレッドを停止させるようなエラーがスレッドで発生した。 • ColdFusion の管理者がサーバーモニタを使用してスレッドを停止した。
COMPLETED	スレッドは正常に終了しました。
WAITING	action="join" を指定した cfthread タグがスレッドで実行されており、結合対象のスレッドがまだ完了していません。

アプリケーションでは、スレッドステータスを確認して処理を管理できます。たとえば、あるスレッドの結果を使用する必要があるアプリケーションで、タイムアウトを指定してそのスレッドを結合している場合は、結合後に COMPLETED ステータスをチェックすることで、スレッドがタイムアウトせずに処理を完了したことを確認できます。同様に、正常に開始または完了しない可能性のあるスレッドのステータス値を確認して、必要に応じて終了することができます。328 ページの「スレッドの終了」の例では、スレッドステータスを確認し、RUNNING または NOT_STARTED ステータスを持つスレッドをすべて終了しています。

スレッド出力の処理

競合を避けるため、出力を表示できるのはページスレッドのみに制限されています。したがって、名前付きスレッドには次のような制限があります。

- スレッド内で生成されたすべての出力 (HTML、プレーンテキスト、cfoutput タグの生成出力など) は、Thread スコープの output メタデータ変数に保存されます。この変数の内容は、ページレベルのコードで **threadName**.output 変数にアクセスすることで表示できます。
- (HTML 出力などのページテキストを生成するのではなく) 出力をクライアントに直接送信するすべてのタグとタグアクションは、スレッド内では機能しません。たとえば、cfdocument または cfreport タグをスレッドで使用するには、filename 属性を指定します。また、cfpresentation タグを使用するには、directory 属性を使用します。

ColdFusion スレッドエラーの処理

スレッド内でエラーが発生しても、ページレベルの処理は影響を受けず、エラーメッセージは生成されません。スレッドコードで発生したエラーが try/catch ブロックを使用して処理されなかった場合、エラーのあるスレッドは終了します。ページレベルのコードまたは他のスレッドでは、スレッドメタデータの Error 変数からエラー情報を取得し、エラーを適宜処理することができます。

ページまたはアプリケーションベースのエラー処理テクニックは、スレッドの実行中に発生したエラーの管理には使用できません。したがって、cferror タグまたは onError アプリケーションイベントハンドラをスレッドエラーに使用することはできません。代わりに、次のテクニックのいずれかを使用します。

1. cftry/cfcatch タグまたは CFScript の try/catch ステートメントを cfthread 本文で使用して、スレッド内でエラーを処理します。
2. ページおよび他のスレッドで使用可能な Thread スコープの **threadName**.Error 変数内のスレッドエラー情報を使用して、スレッド外でエラーを処理します。アプリケーションコードでは、この変数を使用してエラー情報を確認できます。たとえば、エラーが発生した可能性のあるスレッドを結合した場合は、**threadname**.status 変数を確認して、スレッドが異常終了したことを示す terminated が値として設定されていないか調べることができます。その後、**threadName**.Error 変数をチェックして、終了の原因に関する情報を調べることができます。

データベーストランザクションの処理

データベーストランザクションは、複数のスレッドにまたがることはできません。たとえば、あるページに次の構造体があるとします。

```
<cftransaction>
    <cfthread name ="t1" ...>
        <cfquery name="q1" ...>
            ...
        </cfquery>
    </cfthread>
    <cfquery name="q2" ...>
        ...
    </cfquery>
    <cfthread action="join" name="t1" ... />
</cftransaction>
```

この場合、クエリー q1 は、クエリー q2 を含むトランザクションに含まれていません。両方のクエリーをトランザクションに含めるには、次のような構造体を使用して、完全なトランザクションを単一のスレッドに入れる必要があります。

```
<cfthread name ="t1" ...>
    <cftransaction>
        <cfquery name="q1" ...>
            ...
        </cfquery>
        <cfquery name="q2" ...>
            ...
        </cfquery>
    </cftransaction>
</cfthread>
<cfthread action="join" name="t1" ... />
```

ColdFusion ツールによるスレッド使用の制御

ColdFusion Administrator の [サーバーの設定]-[リクエストの調整] ページの [タグ制限の設定] セクションでは、同時に実行可能な cfthread によって開始されたスレッドの最大数を指定できます。この最大数を超えた cfthread リクエストはキューに入れられ、実行中のスレッドが終了するとキューに入れられたスレッドが開始されます。

例 : 複数の RSS フィードの取得

次の例では、3 つのスレッドを使用して 3 つの RSS フィードの結果を取得します。ユーザーは、指定された 3 つのフィードをすべて使用してフォームを送信する必要があります。アプリケーションでは、6 秒間のタイムアウトを指定してスレッドを結合し、フィードのタイトルと、個々の項目タイトルをリンクとして表示します。

```
<!--- Run this code if the feed URL form has been submitted. --->
<cfif isDefined("Form.submit")>
    <cfloop index="i" from="1" to="3">
        <!--- Use array notation and string concatenation to create a variable
                 for this feed. --->
        <cfset theFeed = Form["Feed"&i]>
        <cfif theFeed NEQ "">
            <!--- Use a separate thread to get each of the feeds. --->
            <cfthread action="run" name="t#i#" feed="#theFeed#">
                <cffeed source="#feed#"
                    properties="thread.myProps"
                    query="thread.myQuery">
            </cfthread>
        <cfelse>
            <!--- If the user didn't fill all fields, show an error message. --->
            <h3>This example requires three feeds.<br />
            Click the Back button and try again.</h3>
            <cfabort>
        </cfif>
    </cfloop>

    <!--- Join the three threads. Use a 6 second timeout. --->
    <cfthread action="join" name="t1,t2,t3" timeout="6000" />

    <!--- Use a loop to display the results from the feeds. --->
    <cfloop index="i" from="1" to="3">
        <!--- Use the cfthread scope and associative array notation to get the
                Thread scope. --->
        <cfset feedResult=cfthread["t#i#"]>
        <!--- Display feed information only if you got items,
                 for example, the feed must complete before the join. --->
        <cfif isDefined("feedResult.myQuery")>
            <cfoutput><h2>#feedResult.myProps.title#</h2></cfoutput>
            <cfoutput query="feedResult.myQuery">
                <p><a href="#RSSLINK#">#TITLE#</a></p>
            </cfoutput>
        </cfif>
    </cfloop>

</cfif>

<!--- The form for entering the feeds to aggregate. --->
<cfform>
    <h3>Enter three RSS Feeds</h3>
    <cfinput type="text" size="100" name="Feed1" validate="url"
        value="http://rss.adobe.com/events.rss?locale=en"><br />
    <cfinput type="text" size="100" name="Feed2" validate="url"
        value="http://weblogs.macromedia.com/dev_center/index.rdf"><br />
    <cfinput type="text" size="100" name="Feed3" validate="url"
        value="http://rss.adobe.com/studio.rss?locale=en"><br />
    <cfinput type="submit" name="submit">
</cfform>
```

アプリケーションの保護

リソースセキュリティ (Adobe ColdFusion スタンダード版) やサンドボックスセキュリティ (Adobe ColdFusion エンタープライズ版) は、特定のリソース (タグ、ファイルなど) に対するアクセスを制限するためのセキュリティです。ColdFusion Administrator を使用してサンドボックスセキュリティやリソースセキュリティを設定すれば、このセキュリティを使用してアプリケーションを構築できます。

ユーザーセキュリティは、ユーザー ID に基づくセキュリティです。ユーザーセキュリティも Adobe ColdFusion アプリケーションに実装できます。

Administrator で管理するセキュリティ機能の使用方法の詳細については、『ColdFusion 設定と管理』を参照してください。

ColdFusion のセキュリティ機能

ColdFusion には、ColdFusion アプリケーションの構築やデプロイに役立つ、拡張可能で詳細なセキュリティ機能が用意されています。

ColdFusion では、次のタイプのセキュリティリソースが提供されています。

開発 ColdFusion Administrator はパスワードで保護されています。また、Dreamweaver からデータソースにアクセスするためのパスワードを指定できます。Administrator のセキュリティパスワードの設定の詳細については、ColdFusion Administrator のオンラインヘルプを参照してください。

CFML 機能 CFML 言語には、アプリケーションのセキュリティ強化に役立つ次のような機能があります。

- cfqueryparam タグ : 悪意のある SQL 式がユーザーによって入力されるのを防ぐのに役立ちます。このタグを使用したデータベースセキュリティの詳細については、419 ページの「cfqueryparam によるセキュリティの強化」を参照してください。
- Scriptprotect 設定：この設定を使用して、クロスサイトスクリプティング攻撃から保護することができます。ただし、Scriptprotect を使用しても、完全な保護が保証されるわけではありません。

  この値は、ColdFusion Administrator の「グローバルなスクリプト保護」設定、Application.cfc の This.scriptprotect 変数、または対応する cfapplication タグの scriptprotect 属性で設定します。この機能の詳細については、『CFML リファレンス』の cfapplication を参照してください。"Application.cfc" の詳細については、237 ページの「"Application.cfc" でのアプリケーションおよびイベントハンドラの定義」を参照してください。
- 暗号化関数とハッシュ関数 : Encrypt 関数、Decrypt 関数、および Hash 関数により、データの暗号化と復号およびハッシュ " フィンガープリント " の生成に使用するための安全なアルゴリズムを選択できます。安全なアルゴリズムは、基盤となる Java セキュリティメカニズムでサポートされているいくつかのアルゴリズムの中から選択できます。暗号化の場合は、AES、Blowfish、DES、および Triple DES があります。詳細については、『CFML リファレンス』の Encrypt、Decrypt、および Hash の各関数を参照してください。
- データ検証ツール : ColdFusion には、悪意のあるフォームデータがユーザーから送信されないようにする機能など、フォーム入力やその他のデータ値を検証するための多彩なツールが用意されています。データ検証の詳細については、729 ページの「データの検証」を参照してください。セキュリティと検証に関する具体的な情報については、733 ページの「セキュリティに関する注意事項」を参照してください。

リソース / サンドボックス ColdFusion Administrator では、選択したタグや機能、データソース、ファイル、ホストアドレスなどの、ColdFusion リソースへのアクセスを制限できます。スタンダード版では、すべての ColdFusion アプリケーションに適用するリソース制限セットを 1 つのみ設定できます。

エンタープライズ版では、ColdFusion ページの場所に基づいて、異なるリソース制限セットを持つ複数のサンドボックスを設定できます。セキュリティ保護された領域にアプリケーションを配置することで、そのアプリケーションからのリソースアクセスを柔軟に制限できます。

ユーザー ColdFusion アプリケーションでは、アプリケーションページを使用するユーザーにログインを要求することができます。ユーザーにはロール (グループとも呼びます) を割り当てることができます。ColdFusion ページでは、ログインしたユーザーのロールまたは ID を確認し、ユーザーの行える操作をその情報に基づいて決定できます。ユーザーセキュリティは、認証および承認のセキュリティとも呼ばれます。

注意："<ColdFusion のルートディレクトリ >/bin" ディレクトリにある cfencode ユーティリティを使用して、配布する ColdFusion ページを暗号化して判読できなくすることもできます。このテクニックを使用しても、ハッカーにページの内容を知られる可能性はありますが、ページを詳細に調べられることは防止できます。cfencode ユーティリティは、OS X では使用できません。

リソースセキュリティおよびサンドボックスセキュリティについて

ColdFusion には、次の 2 つのレベルのリソースベースセキュリティがあります。

- ColdFusion スタンダード版のリソースベースセキュリティを、リソースセキュリティと呼びます。このセキュリティでは、ColdFusion リソースへのアクセス制限セットを 1 つ指定して、すべての ColdFusion アプリケーションに適用します。
- ColdFusion エンタープライズ版のリソースベースセキュリティを、サンドボックスセキュリティと呼びます。サンドボックスセキュリティは、リソースセキュリティのスーパーセットです。サンドボックスセキュリティでは、複数のサンドボックスを作成して、それらを異なるディレクトリに適用できます。サンドボックスごとに制限セットを指定して、サンドボックスディレクトリ内やそのサブディレクトリ内のすべての ColdFusion ページに適用できます。サンドボックスのサブディレクトリとなるサンドボックスを作成した場合は、そのサブディレクトリのルールによって、親ディレクトリのルールが上書きされます。

ColdFusion Administrator のリソースセキュリティページ (スタンダード版) およびサンドボックスセキュリティページ (エンタープライズ版) を使用して、リソースベースのセキュリティを有効にできます。ColdFusion スタンダード版のセキュリティページでは、すべての ColdFusion アプリケーションに適用するリソース設定を設定できます。ColdFusion エンタープライズ版のセキュリティページでは、複数のサンドボックスを作成して、サンドボックスごとにリソース制限を設定できます。

リソースの制御

ColdFusion では、次のリソースへのアクセスを制御できます。

リソース	説明
データソース	指定したデータソースへのアクセスを可能にします。
CF タグ	外部リソースにアクセスする CFML タグを使用できないようにします。次のタグのいずれかまたはすべてを使用できないようにディレクトリのページを設定できます。 cfcollection、cfcontent、cfcookie、cfdirectory、cfdocument、cfexecute、cffile、cfftp、cfgridupdate、cfhttp、cfhttpparam、cfindex、cfinsert、cfinvoke、cfldap、cflog、cfmail、cfobject、cfobjectcache、cfpop、cfquery、cfregistry、cfreport、cfschedule、cfsearch、cfstoredproc、cftransaction、cfupdate
CF 関数	外部リソースにアクセスする CFML 関数を使用できないようにします。次の関数のいずれかまたはすべてを使用できないようにページを設定できます。 CreateObject (COM、Java、Web Service)、DirectoryExists、ExpandPath、FileExists、GetBaseTemplatePath、GetDirectoryFromPath、GetFileFromPath、GetGatewayHelper、GetProfileString、GetTempDirectory、GetTempFile、GetTemplatePath、SendGatewayMessage、SetProfileString
ファイル / ディレクトリ	読み取り、書き込み、実行、および削除アクセスを、指定のディレクトリ、ディレクトリツリー、またはファイルに設定します。
サーバー / ポート	ColdFusion から IP アドレスおよびポート番号へのアクセスを制御します。ホスト名または数値アドレスを指定できます。また、個々のポートおよびポート範囲も指定できます。

注意：リソースセキュリティおよびサンドボックスセキュリティの設定の詳細については、『ColdFusion 設定と管理』および ColdFusion Administrator のオンラインヘルプを参照してください。

サンドボックスセキュリティ

ColdFusion エンタープライズ版では、サンドボックスセキュリティを使用して、複数のルールセットをそれぞれ異なるディレクトリ構造に適用できます。これによって、共有ホスティング環境を分割することができ、目的や所有者が異なる多数のアプリケーションを 1 つのサーバー上で安全に実行できます。複数のアプリケーションでホストを共有している場合は、アプリケーションごとに別個のディレクトリ構造を設定し、各アプリケーションが固有のデータソースとファイルのみにアクセスできるようにルールを適用します。

サンドボックスセキュリティを使用すると、各機能コンポーネントに設定するアクセス権に基づいて、アプリケーションの構造を作成および分割できます。たとえば、従業員用の問い合わせ関数と、人事部用の機密データの作成、アクセス、および変更関数を作成する場合は、アプリケーションを次のように構築できます。

- 人事部用のページは、ほとんどのアクティビティを許可するアクセスルールを持つ単一のディレクトリに配置します。
- 従業員用のページは、変更可能なファイルや使用可能なタグを制限するルールを持つ別のディレクトリに配置します。
- 人事部用と従業員用の両方の関数で必要なページは、適切なアクセスルールを持つ 3 番目のディレクトリに配置します。

ユーザーセキュリティについて

ユーザーセキュリティを使用すると、アプリケーションの表示内容をセキュリティルールを使用して決定できます。ユーザーセキュリティには次の 2 つの要素があります。

認証 ユーザーが入力した ID およびパスワードに基づいて、有効なユーザーがログインしていることを確認します。ColdFusion (または、Web サーバー認証を使用する場合は Web サーバー) は、ユーザーがログインしている間はユーザー ID 情報を保持します。

承認 ログインしたユーザーがページを使用したり操作を実行できることを確認します。承認は通常、ユーザーが所属する 1 つまたは複数のロール (グループとも呼びます) に基づいて行います。たとえば、従業員データベースでは、すべてのユーザーを、従業員ロールまたは契約社員ロールのいずれかのメンバーとして分類できます。さらに、部門、役職、または職務を表すロールでも分類できます。たとえば、1 人のユーザーを、次のロールの一部またはすべてのメンバーにすることができます。

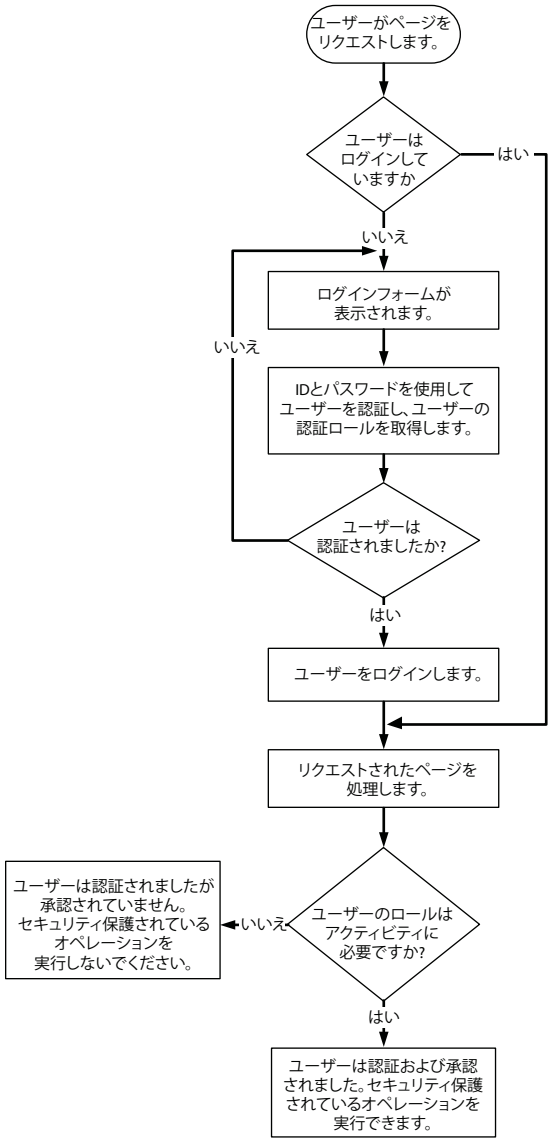
- 従業員
- 人事部
- 給付金
- 管理職

ロールを使用すると、個別のユーザーの情報を管理しなくても、アプリケーションリソースへのアクセスを制御できます。たとえば、ある会社では、イントラネットで ColdFusion を使用しています。イントラネットのページは人事部が管理しています。すべての従業員はこのページにアクセスして、最新の会社方針、行事予定、社内の人材募集などの会社情報を得ることができます。この場合は、すべての従業員が情報を読み取れるようにするとともに、承認された特定の人事部員のみが情報の追加、更新、または削除を行えるようにする必要があります。

ユーザーがログインしたら、ユーザー情報データストアからユーザーのロールを取得し、それに基づいて各ページまたは機能へのアクセスを許可します。通常、ユーザー情報はデータベース、LDAP ディレクトリ、またはその他のセキュリティ保護された情報ストアに保管します。

ユーザー ID は承認にも使用できます。たとえば、前述のイントラネットサイトでは、従業員が自分の給与、職務レベル、職務評価などのカスタマイズされた情報を表示できます。従業員は他の従業員の個人情報を表示できませんが、管理職は他の従業員の情報を表示して更新することができます。この場合は、ユーザー ID とロールの両方を使用することで、適切なユーザーのみに機密データのアクセスや操作を許可できます。

次の図に、ユーザーの認証および承認を行うための一般的な制御フローを示します。次のセクションでは、この図に基づいて、ColdFusion でのユーザーセキュリティの実装方法を説明します。



ユーザーの認証

ColdFusion では、次の認証形式のいずれか、または両方を使用して、アプリケーションを保護します。

- Web サーバー認証 : Web サーバーでユーザーを認証します。有効なログイン ID を持たないユーザーが Web サイトにアクセスするのを防ぎます。
- アプリケーション認証 : ColdFusion アプリケーションでユーザーを認証します。有効なログイン ID を持たないユーザーがアプリケーションにアクセスするのを防ぎます。

Web サーバー認証

すべての主要な Web サーバーでは、HTTP 基本認証がサポートされています。Web サーバーによっては、Digest HTTP 認証や Microsoft NTLM 認証などの、他の認証方式もサポートされています。

注意：Dreamweaver および Studio MX では、RDS による NTLM セキュリティはサポートされていません。したがって、これらのアプリケーションでは、NTLM セキュリティを使用して保護されているディレクトリに ColdFusion RDS サーブレット ("<ColdFusion のルートディレクトリ >/CFIDE/main/ide.cfm") がある場合は、RDS を使用できません。

Web サーバー認証では、次のように、ユーザーが特定のディレクトリのページにアクセスするときにログインが必要になります。

1. セキュリティ保護されたディレクトリ内のページをユーザーが初めてリクエストすると、そのページに資格情報 (ユーザー ID とパスワード) が必要であることが、Web サーバーからブラウザに通知されます。

   HTTP 基本認証では、ユーザー ID とパスワードが base64 エンコードの文字列としてリクエストごとに送信されます。すべてのページトランザクションでは SSL (Secure Sockets Layer) を使用して、ユーザー ID とパスワードを未承認のアクセスから保護してください。SSL と keytool ユーティリティの詳細については、474 ページの「LDAP サーバーのセキュリティについて」を参照してください。
2. ユーザーの資格情報を要求するプロンプトがブラウザに表示されます。
3. ユーザーが資格情報を入力し、ブラウザがその情報を元のリクエストとともに Web サーバーに送信します。
4. Web サーバーは、独自のユーザー認証メカニズムを使用してユーザー ID およびパスワードを確認します。
5. ユーザーがログインに成功すると、ブラウザは認証情報をキャッシュします。以後、ユーザーがページをリクエストするたびに、HTTP 認証ヘッダに認証情報を格納して送信します。
6. Web サーバーはリクエストされたページを処理し、ブラウザからの以降のすべてのページリクエスト (HTTP 認証ヘッダが含まれているリクエスト) も同様に処理します (リクエストされたページに対してログイン情報が有効な場合)。

ColdFusion のセキュリティ機能を使用せずに、Web サーバー認証のみを使用することができます。この場合は、Web サーバーのインターフェイスを通してすべてのユーザーセキュリティを設定および管理します。

また、Web サーバー認証と ColdFusion アプリケーション認証の両方を使用して、承認に ColdFusion セキュリティを使用することもできます。Web サーバーで HTML 基本認証を使用する場合は、ColdFusion で cflogin タグを使用して、ユーザーが Web サーバーにログインするときに入力したユーザー ID およびパスワードにアクセスできます。Web サーバーで Digest 認証または NTLM 認証を使用する場合、cflogin タグは通常はユーザー ID を取得し、パスワードは取得しません。

したがって、ユーザー名とパスワードによるユーザー認証は Web サーバーで行うことができ、アプリケーションでログインページを表示する必要はありません。cflogin タグと cfloginuser タグを使用して ColdFusion ユーザーセキュリティシステムにユーザーをログインさせ、IsUserInAnyRole 関数と GetAuthUser 関数を使用してユーザーの承認を確認します。この形式のセキュリティの詳細については、347 ページの「Web サーバー認証のセキュリティシナリオ」を参照してください。

注意：ユーザーが Web サーバー認証を使用してログインし、ColdFusion アプリケーション認証を使用してログインしていない場合、GetAuthUser タグは Web サーバーのユーザー ID を返します。この機能を使用すると、Web サーバーで認証を行うとともに、アプリケーションでユーザー ID に基づく承認を行うことができます。

アプリケーション認証

アプリケーション認証を使用する場合は、Web サーバーのセキュリティを使用しません。ユーザーの認証や承認は、すべてアプリケーションで行います。アプリケーションでログインページを表示し、LDAP ディレクトリやデータベースなどの独自の承認ストアを使用してユーザー ID およびログインを照合し、cfloginuser タグを使用してユーザーを ColdFusion にログインさせます。ColdFusion ページまたはページ上の特定のコードを実行する前には、IsUserInAnyRole 関数および GetAuthUser 関数を使用して、ユーザーのロールまたは ID を確認できます。アプリケーション認証の使用例については、347 ページの「アプリケーション認証のセキュリティシナリオ」を参照してください。

ColdFusion 認証の保管と永続性

ColdFusion アプリケーションの認証情報がブラウザおよび ColdFusion によって保持される方法およびその有効期限は、次の条件によって異なります。

- ユーザーのブラウザで Cookie が有効になっているか

- アプリケーションがログイン保管のために Session スコープをサポートしているか

注意：Session スコープの詳細については、309 ページの「セッション変数の設定と使用」を参照してください。Cookie スコープにはブラウザから送信された Cookie が格納されます。Cookie の使用方法の詳細については、『CFML リファレンス』の cfcookie を参照してください。

認証と Cookie

HTTP はコネクションレスなので、複数の Web ページにまたがってログインを維持するには、ブラウザから送信された固有の識別子をサーバー上のソフトウェアでチェックして、現在のユーザーが認証されていることを確認する必要があります。通常、この処理には、開いているブラウザウィンドウをすべて閉じると自動的に破棄されるメモリ内のみの Cookie が使用されます。固有の Cookie とその使用方法は、アプリケーションがログイン保管のために Session スコープをサポートしているかどうかによって異なります。

注意：Cookie を使用しないユーザーログインの詳細については、342 ページの「Cookie を利用しない ColdFusion セキュリティの使用」を参照してください。

Session スコープの使用

次の操作を行うと、ログイン情報は Cookie スコープではなく Session スコープで保持されます。

- ColdFusion Administrator と、"Application.cfc" の初期化コードまたは cfapplication タグで、Session スコープを有効にします。
- "Application.cfc" の初期化コードまたは cfapplication タグで、loginStorage="Session" を指定します。

  ログイン情報を Session スコープに保持する場合、認証の詳細情報は Session.cfauthorization 変数に保管され、セッションの Cookie 情報を使用してユーザーが識別されます。セッションベースの認証には、永続性の低いログイン保管にはない次のような利点があります。
- ユーザーがログインした後は、サーバーとブラウザの間でユーザー ID やパスワードがやり取りされることはありません。
- ログイン情報とセッションは、同一のタイムアウトを共有します。セッションとログインを手動で同期する必要はありません。
- サーバークラスタを使用する場合、Session スコープのログイン ID はクラスタ内のすべてのサーバーで使用できます。サーバーのクラスタリングの詳細については、『ColdFusion 設定と管理』を参照してください。

Session スコープを有効化していない場合、認証情報は永続スコープに保持されません。代わりに、詳細なログイン情報は、ユーザー名、パスワード、およびアプリケーション名を含む base64 エンコード文字列として、メモリ内のみの Cookie (CFAUTHORIZATION_< アプリケーション名 >) に置かれます。ユーザーがログインしている間は、クライアントがページをリクエストするたびに、この Cookie が Web サーバーに送信されます。すべてのページトランザクションでは SSL を使用して、ユーザー ID とパスワードを未承認のアクセスから保護してください。

Cookie を利用しない ColdFusion セキュリティの使用

ユーザーのブラウザで Cookie がサポートされていない場合は、有効期間が限定された ColdFusion セキュリティを実装します。この場合は、cflogin タグを使用せずに、cfloginuser タグのみを使用します。cfloginuser タグを cflogin タグの外部で使用する必要があるのは、この場合のみです。

ブラウザ Cookie を使用しない場合、cfloginuser タグの有効期間は単一の HTTP リクエストに限定されます。独自の認証メカニズムを用意して、ColdFusion ログイン ID を使用するページごとに cfloginuser を呼び出します。

ColdFusion のセキュリティタグおよび関数の使用

ColdFusion には、次のようなユーザーセキュリティタグおよび関数があります。

タグまたは関数	用途
cflogin	ユーザー認証コードやログインコードのコンテナ。このタグの本文は、ユーザーがログインしていない場合にのみ実行されます。アプリケーションベースのセキュリティを使用する場合は、cflogin タグの本文に、ユーザーが指定した ID とパスワードをログイン ID のデータソース、LDAP ディレクトリ、またはその他のレポジトリと対照して確認するためのコードを挿入します。このタグの本文では、cfloginuser タグを使用して (または cfloginuser タグを含んだ ColdFusion ページをインクルードして)、認証されたユーザーを ColdFusion で識別します。
cfloginuser	ユーザーを ColdFusion で識別します (ColdFusion にログインさせます)。ユーザーの ID、パスワード、およびロールを指定します。このタグは一般に、cflogin タグの中で使用します。 cfloginuser タグに本文はありませんが、name、password、および roles の 3 つの属性が必要です。roles 属性は、ログインユーザーが属するロール識別子のカンマ区切りのリストです。このリスト内のスペースはロール名の一部と見なされるので、カンマの後にスペースを続けないでください。 ユーザーが ColdFusion にログインしている間、セキュリティ関数はユーザー ID およびロール情報にアクセスします。
cflogout	現在のユーザーをログアウトします。ユーザー ID とロールの情報がサーバーから削除されます。このタグを使用しなかった場合、346 ページの「ユーザーのログアウト」の説明のように、ユーザーは自動的にログアウトされます。 cflogout タグには属性も本文もありません。
cfNTauthenticate	ColdFusion サーバーが稼動している NT ドメインに対してユーザー名とパスワードを認証し、必要に応じてユーザーのグループを取得します。
cffunction	roles 属性を指定すると、そのロールのいずれかに属するユーザーがログインしている場合にのみ、関数が実行されます。
IsUserInAnyRole	現在のユーザーが、指定したロールのメンバーである場合に True を返します。
GetAuthUser	現在ログインしているユーザーの ID を返します。 このタグは、最初に cfloginuser タグによるログインの有無を確認します。そのようなログインがない場合は、Web サーバーログイン (cgi.remote_user) の有無を確認します。

cflogin タグの使用

cflogin タグは、現在ログインしているユーザーがいない場合にのみ実行されます。このタグには、ColdFusion ログインの特性を制御する次の 3 つのオプション引数があります。

属性	用途
idleTimeout	idleTimeout 期間内にページリクエストが行われなかった場合は、ユーザーをログアウトします。デフォルト値は 1,800 秒 (30 分) です。ログイン情報を Session スコープに保管している場合、この値は無視されます。
applicationToken	ColdFusion ページの cfapplication タグで指定した特定のアプリケーションでのみログインを有効にします。デフォルト値は現在のアプリケーション名です。
cookieDomain	ユーザーがログイン済みであることを示す Cookie のドメインを指定します。cookieDomain はクラスタ環境 (x.acme.com、x2.acme.com など) で使用します。この属性によって、クラスタ内のすべてのコンピュータで Cookie が機能します。

ログイン ID スコープと applicationToken 属性

cflogin タグによって作成されるログイン ID は、その cflogin タグを実行したページが含まれているディレクトリ内およびそのサブディレクトリ内のページでのみ有効です。したがって、ユーザーが別のディレクトリツリーにあるページをリクエストした場合、現在のログイン資格情報はそれらのページへのアクセスに対しては無効です。このセキュリティ制限によって、同じユーザー名およびパスワードを異なるアプリケーションセクション (たとえば、UserFunctions ツリーと SecurityFunctions ツリー) に使用して、セクションごとに異なるロールをユーザーに設定できます。

ColdFusion では、applicationToken 値を使用して、このルールを実行する固有の識別子を生成します。applicationToken のデフォルト値は、cfapplication タグまたは "Application.cfc" の初期化コードで指定した現在のアプリケーション名です。通常の使用方法では、cflogin タグで applicationToken 値を指定する必要はありません。

インターネットドメインの指定

cookieDomain 属性を使用して、ユーザーがログイン済みであることを示す Cookie のドメインを指定します。cookieDomain はクラスタ環境 (www.acme.com、www2.acme.com など) で使用します。この属性によって、クラスタ内のすべてのコンピュータで Cookie が機能します。たとえば、Cookie が acme.com ドメインのすべてのサーバーで機能するようにするには、cookieDomain=".acme.com" と指定します。ドメイン名を指定するには、名前をピリオドで始めます。

重要：Cookie ドメインを広く設定すると、意図しないアプリケーションやサーバーから Cookie にアクセスされる可能性があるので、設定する前によく検討してください。たとえば、給与処理アプリケーションをクラスタ化して payroll1.acme.com、payroll2.acme.com などで実行する場合、Cookie ドメインを .acme.com のように広く設定すると、非公開の情報が test.acme.com のテスト用コンピュータからも見えてしまう可能性があります。

ユーザー ID およびパスワードの取得

ページが次のいずれかに応答して実行中の場合、cflogin タグでは、cflogin.username および cflogin.password という 2 つの変数を持つビルトインの cflogin 構造体を使用できます。

- j_username および j_password という入力フィールドを含むログインフォームの送信。
- HTTP 基本認証を使用し、ユーザー名とパスワードが入っている認証ヘッダを含むリクエスト。
- setCredentials メソッドで設定が行われている、Flash Remoting の gatewayConnection オブジェクトからのメッセージ。
- NTLM または Digest 認証を使用するリクエスト。この場合、ユーザー名とパスワードは、Authorization ヘッダに設定される前に一方向アルゴリズムを使用してハッシュされます。ColdFusion は Web サーバーからユーザー名を取得し、cflogin.password 値に空の文字列を設定します。

このうち最初の 3 つはアプリケーション認証で使用される方法であり、最後の 1 つは Web サーバー認証で使用される方法です。cflogin 構造体は、ログインフォームを表示する方法に関係なく使用可能な、ユーザーのログイン ID およびパスワードを取得するための一貫したインターフェイスです。

重要：ログインフォームでは、ユーザー名とパスワードが暗号化されずに送信されます。HTTP 基本認証では、ユーザー名とパスワードが base64 エンコードの文字列としてリクエストのたびに送信されます。これは容易にプレーンテキストに戻すことができる形式です。この方法は、https リクエストや、パスワードのセキュリティが問題にならない場合にのみ使用してください。

次に、認証のためのログイン情報をアプリケーションに渡す方法について説明します。

ログインフォームによるユーザー情報の取得

ログインフォームからユーザー ID とパスワードを取得するアプリケーションを構築する場合は、ユーザーのログイン情報を含んだ cflogin 構造体が存在するかどうかを cflogin タグで確認します。この構造体が存在しない場合は、ログインフォームを表示します。フォームの表示にはログインページで cfinclude タグを使用するのが一般的です。次のコードにその方法を示します。

"Application.cfc" の onRequestStart メソッド、またはこのメソッドから呼び出す ColdFusion ページか CFC メソッドで、次のコードを実行します。

```
<cflogin>
    <cfif NOT IsDefined("cflogin")>
        <cfinclude template="loginform.cfm">
        <cfabort>
     <cfelse>
    <!--- Code to authenticate the user based on the cflogin.user and
        cflogin.password values goes here. --->
    <!--- If User is authenticated, determine any roles and use a line like the
        following to log in the user. --->
        <cfloginuser name="#cflogin.name#" Password = "#cflogin.password#"
            roles="#loginQuery.Roles#">
    </cfif>
</cflogin>
```

単純なログインフォームの例を次に示します。

```
<cfform name="loginform" action="#CGI.script_name#?#CGI.query_string#"
        method="Post">
    <table>
     <tr>
            <td>user name:</td>
            <td><cfinput type="text" name="j_username" required="yes"
                message="A user name is required"></td>
     </tr>
     <tr>
            <td>password:</td>
            <td><cfinput type="password" name="j_password" required="yes"
                message="A password is required"></td>
     </tr>
    </table>
    <br>
    <input type="submit" value="Log In">
</cfform>
```

ブラウザダイアログボックスによるユーザー情報の取得

アプリケーション認証では、ログインフォームを使用する代わりに、ブラウザが表示する標準的なログインダイアログボックスを使用できます。この方法を使用するには、ユーザーがログインしていない場合やログインに失敗した場合 (つまり、有効な cflogin 構造体がない場合) に、cflogin タグの本文で HTTP ステータス 401 をブラウザに返します。これによって、ブラウザにログインダイアログボックスが表示されます。このダイアログボックスのログインボタンをユーザーがクリックすると、ブラウザがログイン情報を HTTP Authorization ヘッダとして ColdFusion に返します。この情報は cflogin タグの cflogin 構造体に設定されます。

この方法の利点は単純であることです。ログインフォームを用意する必要がなく、ユーザー側にも見慣れたログインページが表示されます。ただし、セキュリティには注意が必要です。ユーザー名とパスワードは base64 エンコードの文字列として、ログイン時だけでなくリクエストのたびにブラウザから送信されます。すべてのページトランザクションでは SSL (Secure Sockets Layer) を使用して、ユーザー ID とパスワードを未承認のアクセスから保護してください。

注意：この方法を使用するには、ブラウザベースのログインフォームがサポートされるように Web サーバーを正しく設定する必要があります。たとえば IIS 5 では、匿名アクセスを許可し、基本認証と統合 Windows 認証を無効にします。

次の cflogin タグでは、ユーザーがまだログインしていなければログインフォームを表示します。

```
<cflogin>
    <cfif NOT IsDefined("cflogin")>
        <cfheader statuscode="401">
        <cfheader name="www-Authenticate" value="Basic
            realm=""MM Wizard #args.authtype# Authentication""">
    </cfif>
    <cfabort>
    <cfelse>
        <!--- code to authenticate the user based on the cflogin.user and
            cflogin.password values goes here. --->
</cflogin>
```

Flash Remoting によるユーザーのログイン

Flash と Flash Remoting を使用したリッチインターネットアプリケーションを開発する場合は、ColdFusion アプリケーションに Flash ログイン用の特別なコードを挿入する必要はありません。Flash Remoting ゲートウェイから渡されたユーザー ID とパスワードは、cflogin タグ内の cflogin 構造体で取得できます。

Flash コードでは、ActionScript の SetCredentials メソッドを使用してログイン情報を ColdFusion に送信します。Flash SWF ファイルでユーザー ID およびパスワードのフィールドを表示して、次のように、それらのフィールドの内容を setCredentials メソッドに渡します。

```
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
    gatewayConnection = NetServices.createGatewayConnection();
    gatewayConnection.setCredentials(userID, password);
    myService = gatewayConnection.getService("securityTest.thecfc", this);
}
```

Flash Remoting の使用方法の詳細については、595 ページの「Flash Remoting サービスの使用」および 608 ページの「Flash Remoting Update の使用」を参照してください。

ユーザーのログアウト

ColdFusion のユーザー承認情報および認証情報は、ユーザーがログインしてから次のいずれかの条件が発生するまでの間有効です。

- アプリケーションで cflogout タグを使用してユーザーをログアウトした場合。通常、ユーザーがログアウトのリンクまたはボタンをクリックした場合に実行します。
- ログイン情報の保管に Session スコープを使用しており、セッションが終了した場合。
- ログイン情報の保管に Session スコープを使用しておらず、cflogin タグの idleTimeout の期間内に新しいページがリクエストされなかった場合。
- ログイン情報の保管に Session スコープを使用していないか、J2EE ベースのセッション ID を使用しているときに、ユーザーがすべてのブラウザウィンドウを閉じた場合。

cflogout タグを使用してユーザーをログアウトさせても、そのユーザーのセッションは閉じませんが、セッションログイン保管域を使用している場合は、Session スコープからログイン情報 (Session.cfauthorization 変数) が削除されます。セッション終了の詳細については、313 ページの「セッションの終了」を参照してください。

重要：Web サーバー認証を使用している場合や、HTTP 基本認証ヘッダを使用するフォーム認証を使用している場合は、ユーザーがブラウザを閉じるまで (場合によってはすべての開いているブラウザウィンドウを閉じるまで)、ブラウザはアプリケーションに認証情報を送り続けます。したがって、ユーザーがログアウトしてアプリケーションで cflogout タグが実行された後でも、ブラウザが閉じられるまでは、ログアウトしたユーザーのユーザー ID とパスワードが cflogin タグ内の cflogin 構造体に保持されます。このため、ユーザーがログアウト後にブラウザを閉じなかった場合、別のユーザーがそのユーザーのログインを使用してページにアクセスすることができます。

セキュリティシナリオ

次に、2 つの詳細なセキュリティシナリオについて説明します。最初のシナリオでは、Web サーバーでユーザーおよびパスワードのデータベースを照合して認証を行います。2 番目のシナリオでは、すべての認証および承認を ColdFusion で行います。

Web サーバー認証のセキュリティシナリオ

Web サーバー認証を使用するアプリケーションでは、次のような処理を行います。このシナリオの実装例については、352 ページの「Web サーバー認証によるユーザーセキュリティの例」を参照してください。

1. ユーザーがブラウザを起動して、サーバー上の特定のディレクトリにあるページを初めてリクエストすると、Web サーバーはログインページを表示して、ユーザーをログインさせます。すべてのユーザー認証は、Web サーバーが処理します。
2. ColdFusion ページがリクエストされたので、Web サーバーは ColdFusion にリクエストを渡します。
3. ColdFusion は、ColdFusion ページに対するリクエストを受信すると、"Application.cfc" をインスタンス化して onRequestStart メソッドを実行します。"Application.cfc" ではなく "Application.cfm" ページを使用している場合は、リクエストされたページを実行する前に "Application.cfm" ページの内容を実行します。onRequestStart メソッドまたは "Application.cfm" ページには、cflogin タグが含まれています。ユーザーが ColdFusion にログインしていない場合は、cflogin タグ本文を実行します。cfloginuser タグがこのアプリケーションに対して正常に実行され、ユーザーがまだログアウトしていない場合、ユーザーはログインしています。
4. ブラウザログインで送られたユーザー ID およびパスワードは、cflogin.name および cflogin.password 変数に格納されます。cflogin タグ本文内のコードでは、それらを使用して次の処理を行います。Digest または NTLM web サーバー認証では、cflogin.password 変数は空の文字列になります。
   a. ユーザーとロールが保持されている情報ストアを使用してユーザー名を照合します。単純なアプリケーションでは、ユーザーロールと管理者ロールの 2 つのロールが使用されます。CFML は、**Admin** というユーザー ID を使用してログオンしたユーザーに Admin ロールを割り当て、他のすべてのユーザーに User ロールを割り当てます。
   b. ユーザーの ID、パスワード、およびロールを使用して cfloginuser タグを呼び出し、ColdFusion でそのユーザーを識別します。
5. "Application.cfc" または "Application.cfm" ページの処理が完了すると、ColdFusion はリクエストされたアプリケーションページを処理します。
6. 特定のロールのユーザーのみに許可されるコードを実行する前には、IsUserInAnyRole 関数を使用して、ユーザーがそのロールに属しているかどうかを確認します。
7. ユーザー ID を判断するには、GetAuthUser 関数を使用します。たとえば、この ID を使用して表示をカスタマイズできます。また、この ID をデータベースキーとして使用し、ユーザー固有のデータを取得することもできます。

重要：Web サーバー認証を使用している場合や、HTTP 基本認証ヘッダを使用するフォーム認証を使用している場合は、ユーザーがブラウザを閉じるまで (場合によってはすべての開いているブラウザウィンドウを閉じるまで)、ブラウザはアプリケーションに認証情報を送り続けます。したがって、ユーザーがログアウトしてアプリケーションで cflogout タグが実行された後でも、ブラウザが閉じられるまでは、ログアウトしたユーザーのユーザー ID とパスワードが cflogin タグ内の cflogin 構造体に保持されます。このため、ユーザーがログアウト後にブラウザを閉じなかった場合、別のユーザーがそのユーザーのログインを使用してページにアクセスすることができます。

アプリケーション認証のセキュリティシナリオ

独自の認証を実行するアプリケーションでは、次のような処理を行います。このシナリオの実装例については、354 ページの「アプリケーションベースのユーザーセキュリティの例」を参照してください。

1. ColdFusion は、ColdFusion ページに対するリクエストを受信すると、"Application.cfc" をインスタンス化して onRequestStart メソッドを実行します。"Application.cfc" ではなく "Application.cfm" ページを使用している場合は、リ

クエストされたページを実行する前に "Application.cfm" ページの内容を実行します。onRequestStart メソッドまたは "Application.cfm" ページには、cflogin タグが含まれています。ユーザーがログインしていない場合は、cflogin タグ本文を実行します。cfloginuser タグが現在のセッションで実行され、ユーザーが cflogout タグでログアウトしていない場合、ユーザーはログインしています。

2 cflogin タグ本文のコードで、ユーザー ID とパスワードをログインフォームなどから受信したかどうかを確認します。

3 ユーザー ID もパスワードも受信していない場合は、cflogin タグ本文のコードで、ユーザー ID とパスワードを要求するログインフォームを表示します。

このフォームから最初にリクエストされたページにログイン情報が戻されると、onRequestStart メソッドまたは "Application.cfm" ページの cflogin タグが再び実行されます。cflogin タグ本文のコードでは、データベース、LDAP ディレクトリ、またはその他のポリシーストアを使用してユーザー名とパスワードを照合し、ユーザーが有効であることを確認してユーザーのロールを取得します。

4 ユーザー名とパスワードが有効であれば、cflogin タグ本文のコードでユーザーの ID、パスワード、およびロールを使用して cfloginuser タグを呼び出し、ColdFusion でそのユーザーを識別します。

5 ユーザーがログインしている場合、特定のロールのユーザーのみに許可されるコードを実行する前には、IsUserInAnyRole 関数を使用して、ユーザーがそのロールに属しているかどうかを確認します。

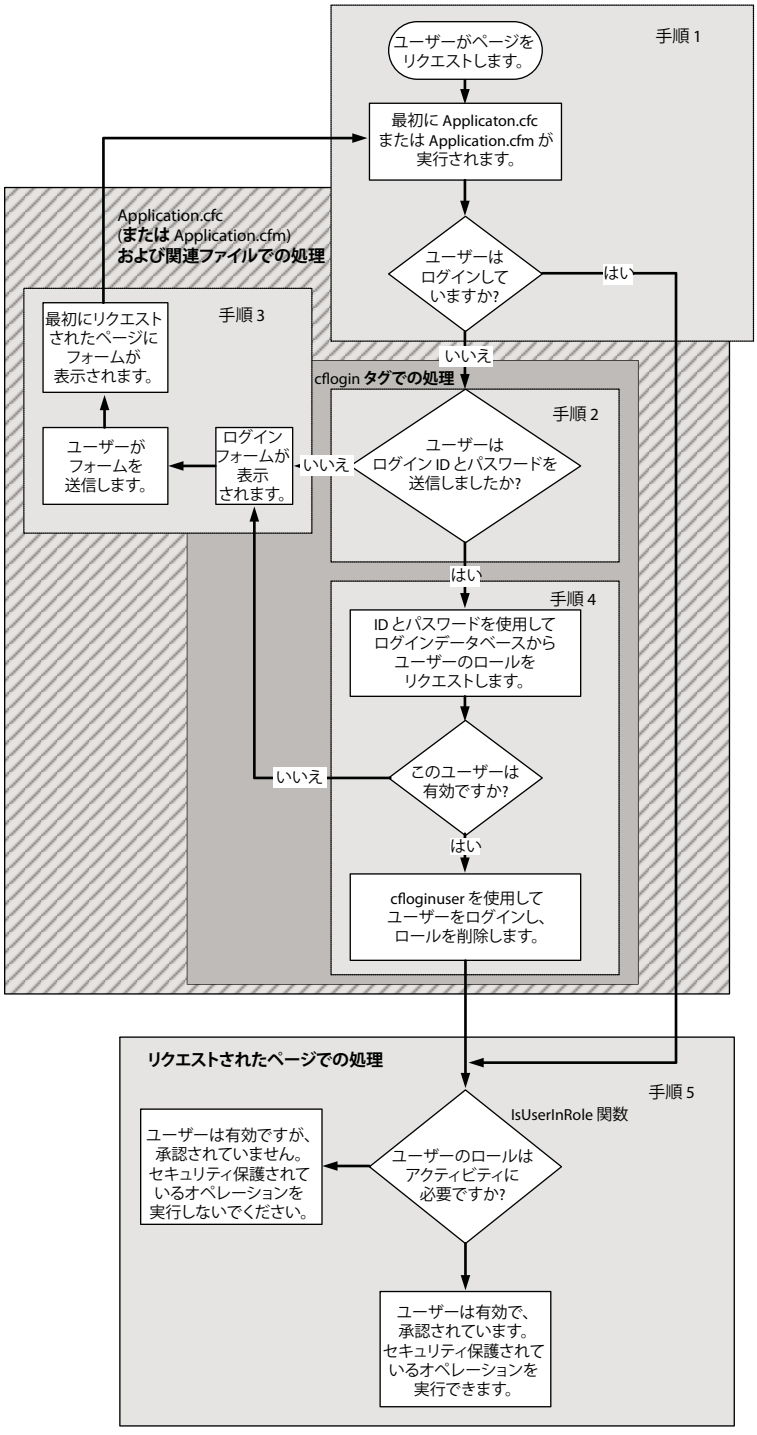
ユーザー ID を判断するには、GetAuthUser 関数を使用できます。たとえば、この ID を使用して表示をカスタマイズできます。また、この ID をデータベースキーとして使用し、ユーザー固有のデータを取得することもできます。

6 各アプリケーションページでは、cflogout タグを使用してユーザーをログアウトするログアウトフォームへのリンクを表示します。通常、このログアウトリンクは、すべてのページに表示するページヘッダに配置します。ログアウトフォームは、"Application.cfc" 内 (たとえば、onRequestStart メソッドや onRequestEnd メソッド内)、または "Application.cfm" ページ内で定義することもできます。

このシナリオは、ユーザーセキュリティを実装する一例にすぎません。たとえば、管理者機能を含んでいるフォルダにあるページなど、特定のページでのみユーザーにログインを要求するアプリケーションもあります。ユーザーセキュリティの実装を設計するときは、次の点に注意してください。

- cflogin タグ本文のコードは、ログインしているユーザーがいない場合にのみ実行されます。
- アプリケーション認証を行う場合は、ユーザーから ID を取得し、セキュリティ保護された資格情報ストアを使用してその情報をテストするためのコードを、独自に作成します。
- ユーザーを認証したら、次に cfloginuser タグを使用して、ユーザーを ColdFusion にログインさせます。

次の図に、この制御フローを示します。簡略化するため、ログアウトオプションは省略しています。



ユーザーセキュリティの実装

次の方法でセキュリティを実装できます。

Dreamweaver ログインウィザードの使用

ColdFusion によって、Dreamweaver の [コマンド] メニューにログインウィザードコマンドがインストールされます。このコマンドを使用すると、ユーザーの認証および承認を管理するためのページ群を生成できます。

このウィザードでは、ログイン情報の認証方法を指定するように求められます。次のいずれかのオプションを選択します。

- **Simple**: 単一のユーザー ID およびパスワードをウィザードで指定します。すべてのユーザーはログイン時にこの情報を入力する必要があります。このオプションはテストに使用できます。また、生成されたファイルをテンプレートとして使用し、認証コードをより複雑なコードに置き換えることもできます。たとえば、データベースを使用して ID とパスワードを検証するコードなどです。
- **NT domain**: NT ドメインをウィザードで指定します。そのドメインに対してクエリーを発行するコードがウィザードによって生成されます。
- **LDAP**: LDAP サーバーおよびポート、ログインデータへのアクセスに必要なユーザー名およびパスワード、そのユーザー名の検索を開始するために使用する識別名 (DN) を指定します。そのユーザー ID とパスワードを使用して LDAP サーバーにクエリーを発行するコードがウィザードによって生成されます。

ログイン情報の入力を要求する方法について、次のいずれかを指定するように求められます。

- Browser Dialog Box
- ColdFusion Login Form

ログインウィザードで生成されるコードの構造

ウィザードによって、指定したディレクトリまたはサイト内の次のファイルが生成または変更されます。

Application.cfc このファイルが存在しない場合は、onRequestStart メソッドのみを持つファイルがウィザードによって作成されます。アプリケーション名の指定や他のメソッドの生成は行われません。ファイルが存在し、onRequestStart メソッドが含まれていない場合は、このメソッドが追加されます。"Application.cfc" および onRequestStart メソッドが存在する場合は、必要なコードがこのメソッドの先頭に挿入されます。結果として onRequestStart メソッドには、"mm_wizard_application_include.cfm" を指定する cfinclude タグが含まれます。また、ログアウトボタンを持つ単純なフォームも含まれており、アプリケーションの各ページの一番上にこのフォームが表示されます。

注意：ウィザードによって "Application.cfc" ファイルが作成されたら、そのファイルを編集して、アプリケーション名を指定してください。"Application.cfc" の詳細については、231 ページの「ColdFusion アプリケーションの設計と最適化」を参照してください。

mm_wizard_application_include.cfm ウィザードのフィールドに指定された情報を使用して、CFC メソッドのいくつかの引数が設定されます。次に、それらを使用して、マスタログイン CFC である mm_wizard.authenticate の performlogin メソッドが呼び出されます。

mm_wizard_authenticate.cfc この CFC には、ユーザー認証とログインに関するすべてのロジックが含まれます。この CFC は次のメソッドから構成されます。

- 認証メソッド ntauth、ldapauth、および simpleauth は、有効なログイン情報を使用してユーザーの名前と ID を照合し、そのユーザーが認証されたかどうかを返します。これらのメソッドによるユーザー認証方法と具体的な戻り値については、各メソッドを参照してください。
- performLogin メソッドはマスタログインメソッドです。この中には cflogin タグが含まれており、ログインフォームの表示と必要な認証メソッドの呼び出しが行われます。認証メソッドの戻り値で有効なユーザーであることが示された場合は、このメソッドによってユーザーがログインされます。
- logout メソッドはユーザーをログアウトします。ログインページのタイプとして [Browser Dialog Box] を指定した場合は、ブラウザウィンドウを閉じるために、このメソッドから closeBrowser メソッドが呼び出されます。この動作が必要なのは、ユーザーがログアウトした後もブラウザから古いログイン資格情報が送信され続け、cflogin タグがそれを使用して再びユーザーをログインさせてしまうからです。

- closeBrowser メソッドは、ブラウザのウィンドウを閉じるか、ブラウザを閉じてログアウトを完了するように指示するメッセージを表示します。どちらの処理を行うかは、ブラウザのタイプによって異なります。

mm_wizard_login.cfm このファイルには ColdFusion ログインフォームが含まれています。どのオプションを選択しても生成されますが、[Browser Dialog] ログインを指定した場合は使用されません。

"index.cfm" または "mm_wizard_index.cfm" そのディレクトリに "index.cfm" ページが存在しない場合は、ウィザードによって生成されます。"index.cfm" ページが存在する場合は、"mm_wizard_index.cfm" ページが生成されます。これらのページを使用すると、生成されたログインコードを、アプリケーションの実装前にテストすることや、標準のアプリケーションページを一切使用せずにテストすることができます。ログインをテストするには、ブラウザで "index.cfm" ページを開きます。

アプリケーションに応じたログインコードの変更

ログインウィザードで作成されるのは、ユーザー認証のための基本的なフレームワークです。アプリケーションのニーズに合わせるには、このフレームワークをカスタマイズします。セキュリティ関連では、次のような変更がよく行われます。

- cflogin タグにおけるユーザー固有のロール情報の提供
- データベースを使用したユーザー認証

ユーザー固有のロール情報の提供

ログインウィザードは、すべてのユーザーに単一のロールを設定します。"mm_wizard_authenticate.cfc" 内の performlogin メソッドは、ロールを "user" に設定するようにハードコードされています。ロールの扱いは認証ルーチンによって異なります。詳細については、"mm_wizard_authenticate.cfc" のコードを参照してください。アプリケーションでロールを使用して承認を行う場合は、有効なロール情報を取得して返すように認証メソッドを変更し、また、cfloginuser タグの roles 属性でその情報を使用するように performlogin メソッドを変更します。

データベースを使用したユーザー認証

データベースを使用してユーザー ID とパスワードを管理している場合は、シンプル認証を指定してログインフレームワークを作成し、データベースを使用するようにコードを変更します。データベースを使用するようにコードを変更する簡単な方法を、次の手順で説明します。ただし、これだけではコードのクリーンアップは完全ではありません (特に、ハードコードされたユーザー名とパスワードの削除などは示していません)。実際のアプリケーションではその他のクリーンアップも必要です。

次のコードを置き換えます。

```
 <cfif sUserName eq uUserName AND sPassword eq uPassword>
    <cfset retargs.authenticated="YES">
<cfelse>
    <cfset retargs.authenticated="NO">
</cfif>
<cfreturn retargs>
```

変更後のコードは次のようになります。

```
 <cfquery name="loginQuery" dataSource="#Application.DB#" >
    SELECT *
    FROM Users
    WHERE UserName = <cfqueryparam value="#uUserName#" CFSEQLType=
        'CF_SQL_VARCHAR'AND password = <cfqueryparam value="#uPassword#"
        CFSEQLType='CF_SQL_VARCHAR'>
</cfquery>

<cfif loginQuery.recordcount gt 0>
    <cfset retargs.authenticated="YES">
    <cfset retargs.roles=loginQuery.roles>
<cfelse>
    <cfset retargs.authenticated="NO">
</cfif>
<cfreturn retargs>
```

注意：セキュリティ強化のために、パスワードはハッシュ処理することをお勧めします。データベースにパスワードをそのまま格納することは避け、hash 関数で作成した安全なパスワードフィンガープリントをデータベースに格納するようにしてください。ユーザーがパスワードを入力したら、送信された文字列を Hash 関数で処理し、その結果の値をデータベースの値と比較します。

Web サーバー認証によるユーザーセキュリティの例

次の例では、Web サーバーベースの基本認証と、ユーザーおよび管理者の 2 つのロールを使用してユーザーセキュリティを実装する方法を示します。

この例には、次の 2 つの ColdFusion ページがあります。

1. "Application.cfc" ページでは、ColdFusion セキュリティシステムにユーザーをログインさせ、ユーザー ID に基づいて特定のロールをユーザーに割り当てます。

   このページには、ユーザーをログアウトするための 1 つのボタンとロジックを備えたフォームも含まれています。これは、各ページの一番上に表示されます。

2. "securitytest.cfm" ページは、サンプルのアプリケーションページです。ログインしたユーザーのロールが表示されます。

この簡単な例には、ユーザーログアウトインターフェイスは含まれていません。"Application.cfc" ページと同じディレクトリに独自のページを追加して、セキュリティ動作をテストします。

例 : "Application.cfc"

"Application.cfc" ページは、次のように構成されます。

```
 <cfcomponent>
<cfset This.name = "Orders">
<cffunction name="OnRequestStart">
    <cfargument name = "request" required="true"/>
    <cflogin>
        <cfif IsDefined("cflogin")>
            <cfif cflogin.name eq "admin">
                <cfset roles = "user,admin">
            <cfelse>
                <cfset roles = "user">
            </cfif>
            <cfloginuser name = "#cflogin.name#" password = "#cflogin.password#"
                roles = "#roles#" />
        <cfelse>
            <!--- This should never happen. --->
            <h4>Authentication data is missing.</h4>
            Try to reload the page or contact the site administrator.
        <cfabort>
        </cfif>
    </cflogin>
</cffunction>
</cfcomponent>
```

コードの説明

"Application.cfc" の onRequestStart メソッドは、アプリケーションの各 ColdFusion ページのコードよりも前に実行されます。"Application.cfc" ページとその実行タイミングの詳細については、231 ページの「ColdFusion アプリケーションの設計と最適化」を参照してください。

"Application.cfc" の CFML コードおよびその機能について、次の表で説明します。

コード	説明

`<cfcomponent>`
`<cfset This.name = "Orders">`
`<cffunction name="OnRequestStart">`
`<cfargument name = "request" required="true"/>`
	アプリケーションを識別し、個々のリクエストの開始時に実行される onRequestStart メソッドを開始します。このページのログイン情報は、このアプリケーションのみに適用されます。

`<cflogin>`
`<cfif IsDefined("cflogin")>`
`<cfif cflogin.name eq "admin">`
`<cfset roles = "user,admin">`
`<cfelse>`
`<cfset roles = "user">`
`</cfif>`
	ログインしているユーザーがいない場合に実行されます。
	Web サーバーによってユーザーが正しくログインされたかどうかを確認します。ログインしていない場合は、cflogin 変数がありません。
	ユーザーの ID に基づいてロール変数を設定します。"admin" という名前のユーザーには admin ロールを割り当てます。他のすべてのユーザーには user ロールを割り当てます。

`<cfloginuser name = "#cflogin.name#" password = "#cflogin.password#" roles = "#roles#" />`	ColdFusion セキュリティシステムにユーザーをログインさせ、ユーザーのパスワード、名前、およびロールを指定します。パスワードと名前は、cflogin 構造体から直接取得します。

`<cfelse>`
`<!--- This should never happen. --->`
`<h4>Authentication data is missing.</h4>`
`Try to reload the page or contact the site administrator.`
`<cfabort>`
	このコードは通常は実行されませんが、ユーザーが何らかの方法で Web サーバーにログインせずにこのページを表示した場合は、このメッセージが表示され、ColdFusion によってリクエスト処理が停止されます。

`</cfif>`
`</cflogin>`
`</cffunction>`
`</cfcomponent>`
	if/else ブロックを終了します。
	cflogin タグ本文を終了します。
	onRequestStart メソッドを終了します。
	Application コンポーネントを終了します。

例 : "securitytest.cfm"

"securitytest.cfm" ページでは、アプリケーションページにおける ColdFusion のユーザー承認機能の使用方法を示します。Web サーバーでは認証されたユーザーが存在することが確認され、"Application.cfc" ページでは表示するページ内容を示すロールがユーザーに割り当てられます。"securitytest.cfm" ページでは、IsUserInAnyRole 関数と GetAuthUser 関数を使用して、表示する情報を制御します。

"securitytest.cfm" ページは、次のように構成されます。

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Basic authentication security test page</title>
</head>

<body>
<cfoutput>
    <h2>Welcome #GetAuthUser()#!</h2>
</cfoutput>

ALL Logged-in Users see this message.<br>
<br>
<cfscript>
    if (IsUserInRole("admin"))
        WriteOutput("Users in the admin role see this message.<br><br>");
    if (IsUserInRole("user"))
        WriteOutput("Everyone in the user role sees this message.<br><br>");
</cfscript>

</body>
</html>
```

コードの説明

"securitytest.cfm" ページの CFML コードおよびその機能について、次の表で説明します。

コード	説明

`<cfoutput>`
`<h2>Welcome #GetAuthUser()#!</h2>`
`</cfoutput>`
	ユーザーは既に "Application.cfc" によってログインされています。ユーザーのログイン ID を含む挨拶を表示します。

`ALL Logged-in Users see this message. ` ` `	このメッセージはすべてのユーザーに表示されます。このページは、ユーザーがログインするまで表示されません。
`<cfscript>` `if (IsUserInRole("admin"))` `WriteOutput("Users in the admin role see this message.  ");` `if (IsUserInRole("user"))` `WriteOutput("Everyone in the user role sees this message.  ");` `</cfscript>`	ユーザーが有効なロールに所属しているかどうかをテストします。ユーザーがいずれかのロールに所属している場合は、ロール名を含むメッセージを表示します。 所属するロールに応じた 1 つのメッセージがユーザーに表示されます。

アプリケーションベースのユーザーセキュリティの例

次の例では、ユーザーを認証し、許可されたリソースのみを表示および使用できるようにするユーザーセキュリティの実装方法を示します。

この例には、次の 3 つの ColdFusion ページがあります。

- "Application.cfc" ページの認証ロジックでは、ユーザーがログインしているかどうかを確認し、ユーザーがログインしていなければログインページをリクエストし、ログインページからのデータを認証します。ユーザーが認証されると、ログインします。

  このページには、ユーザーをログアウトするための 1 つのボタンとロジックを備えたフォームも含まれています。これは、各ページの一番上に表示されます。

- "loginform.cfm" ページでは、ログインフォームを表示します。このページのコードは、"Application.cfc" に含めることもできます。
- "securitytest.cfm" ページは、サンプルのアプリケーションページです。ログインしたユーザーのロールが表示されます。

"Application.cfc" ページと同じディレクトリに独自のページを追加して、セキュリティ動作をテストします。

この例では、ColdFusion とともにインストールされた cfdocexamples データベースの LoginInfo テーブルからユーザー情報を取得します。このデータベースは、UserID、Password、および Roles フィールドを持つ任意のデータベースに置き換えることができます。このサンプルデータベースには、次のデータが含まれています。

UserID	Password	Roles
BobZ	Ads10	Employee,Sales
JaniceF	Qwer12	Contractor,Documentation
RandalQ	ImMe	Employee,Human Resources,Manager

ロールの文字列ではスペースにも意味があるので、Roles フィールドのカンマ区切り文字の後にスペースを続けないでください。

例 : "Application.cfc"

"Application.cfc" ページは、次のように構成されます。

```
 <cfcomponent>
<cfset This.name = "Orders">
<cfset This.Sessionmanagement="True">
<cfset This.loginstorage="session">

<cffunction name="OnRequestStart">
    <cfargument name = "request" required="true"/>
    <cfif IsDefined("Form.logout")>
        <cflogout>
    </cfif>

    <cflogin>
        <cfif NOT IsDefined("cflogin")>
            <cfinclude template="loginform.cfm">
            <cfabort>
        <cfelse>
            <cfif cflogin.name IS "" OR cflogin.password IS "">
                <cfoutput>
                    <h2>You must enter text in both the User Name and Password fields.
                    </h2>
                </cfoutput>
                <cfinclude template="loginform.cfm">
                <cfabort>
            <cfelse>
                <cfquery name="loginQuery" dataSource="cfdocexamples">
                SELECT UserID, Roles
```

```
                FROM LoginInfo
                WHERE
                    UserID = '#cflogin.name#'
                    AND Password = '#cflogin.password#'
                </cfquery>
                <cfif loginQuery.Roles NEQ "">
                    <cfloginuser name="#cflogin.name#" Password = "#cflogin.password#"
                        roles="#loginQuery.Roles#">
                <cfelse>
                    <cfoutput>
                        <H2>Your login information is not valid.<br>
                        Please Try again</H2>
                    </cfoutput>
                    <cfinclude template="loginform.cfm">
                    <cfabort>
                </cfif>
            </cfif>
        </cfif>
    </cflogin>

    <cfif GetAuthUser() NEQ "">
        <cfoutput>
                <form action="securitytest.cfm" method="Post">
                <input type="submit" Name="Logout" value="Logout">
            </form>
        </cfoutput>
    </cfif>

</cffunction>
</cfcomponent>
```

コードの説明

"Application.cfc" ページは、アプリケーションの各 ColdFusion ページのコードよりも前に実行されます。
"Application.cfc" ページとその実行タイミングの詳細については、231 ページの「ColdFusion アプリケーションの設計と最適化」を参照してください。

"Application.cfc" の CFML コードおよびその機能について、次の表で説明します。

コード	説明

`<cfcomponent>`
`<cfset This.name = "Orders">`
`<cfset This.Sessionmanagement="True">`
`<cfset This.loginstorage="session">`
 	
`<cffunction name="OnRequestStart">`
`<cfargument name = "request" required="true"/>`
	アプリケーションを識別し、セッション管理を有効にして、Session スコープへのログイン情報の保管を有効にします。
	個々のリクエストの開始時に実行される onRequestStart メソッドの定義を開始します。

`<cfif IsDefined("Form.logout")>`
`<cflogout>`
`</cfif>`
	ユーザーがログアウトフォームを送信したら、そのユーザーをログアウトします。その結果、次の cflogin タグが実行されます。

`<cflogin>`
`<cfif NOT IsDefined("cflogin")>`
`<cfinclude template="loginform.cfm">`
`<cfabort>`
	ログインしているユーザーがいない場合に実行されます。
	ユーザーがログインフォームを送信したかどうかをテストします。送信していない場合は、cfinclude を使用してフォームを表示します。ログインフォームで j_username および j_password 入力フィールドが使用された場合のみ、ビルトインの cflogin 変数が存在し、この変数にユーザー名とパスワードが含まれます。
	cfabort タグにより、このページの以降のコードは処理されません。

コード	説明
<cfelse> <cfif cflogin.name IS "" OR cflogin.password IS ""> <cfoutput> You must enter text in both the User Name and Password fields. </cfoutput> <cfinclude template="loginform.cfm"> <cfabort>	ユーザーがログインフォームを送信すると実行されます。 名前とパスワードの両方にデータが含まれているかどうかをテストします。いずれかの変数が空の場合は、メッセージを表示して、ログインフォームを表示します。 cfabort タグにより、このページの以降のコードは処理されません。
<cfelse> <cfquery name="loginQuery" dataSource="cfdocexamples"> SELECT UserID, Roles FROM LoginInfo WHERE UserID = '#cflogin.name#' AND Password = '#cflogin.password#' </cfquery>	ユーザーがログインフォームを送信し、両方のフィールドにデータが含まれている場合に実行されます。 cflogin 構造体の name および password エントリを使用して、データベースでユーザーレコードを検索し、ユーザーのロールを取得します。
<cfif loginQuery.Roles NEQ ""> <cfloginuser name="#cflogin.name#" Password = "#cflogin.password#" roles="#loginQuery.Roles#">	クエリーによって Roles フィールドのデータが返された場合は、ユーザーの名前とパスワード、およびデータベースの Roles フィールドを使用してユーザーをログインさせます。このアプリケーションでは、どのユーザーもいずれかのロールを持つ必要があります。
<cfelse> <cfoutput> <H2>Your login information is not valid. Please Try again</H2> </cfoutput> <cfinclude template="loginform.cfm"> <cfabort>	クエリーがロールを返さない場合に実行されます。データベースが有効な場合は、ユーザー ID およびパスワードに一致するエントリがないことを意味します。メッセージを表示し、次にログインフォームを表示します。 cfabort タグにより、このページの以降のコードは処理されません。
</cfif> </cfif> </cfif> </cflogin>	loginquery.Roles テストコードを終了します。 フォームエントリに空の値があるかどうかのテストを終了します。 フォームエントリが存在するかどうかのテストを終了します。 cflogin タグ本文を終了します。
<cfif GetAuthUser() NEQ ""> <cfoutput> <form action="securitytest.cfm" method="Post"> <input type="submit" Name="Logout" value="Logout"> </form> </cfoutput> </cfif>	ユーザーがログインしている場合は、[ログアウト] ボタンを表示します。 ユーザーがボタンをクリックすると、アプリケーションのロジック上のエントリページである "index.cfm" にフォームが送信されます。 次にユーザーをログアウトし、ログインフォームを表示します。ユーザーが再びログインすると、"index.cfm" が表示されます。
</cffunction> </cfcomponent>	onRequestStart メソッドを終了します。 Application コンポーネントを終了します。

例 : "loginform.cfm"

"loginform.cfm" ページは、次のように構成されます。

```
 <H2>Please Log In</H2>
<cfoutput>
    <form action="#CGI.script_name#?#CGI.query_string#" method="Post">
        <table>
            <tr>
                <td>user name:</td>
                <td><input type="text" name="j_username"></td>
            </tr>
            <tr>
                <td>password:</td>
                <td><input type="password" name="j_password"></td>
            </tr>
        </table>
        <br>
        <input type="submit" value="Log In">
    </form>
</cfoutput>
```

コードの説明

"loginform.cfm" ページの CFML コードおよびその機能について、次の表で説明します。

コード	説明
<H2>Please Log In</H2> <cfoutput> <form action="#CGI.script_name#?#CGI.query_string#" method="Post"> <table> <tr> <td>user name:</td> <td><input type="text" name="j_username"></td> </tr> <tr> <td>password:</td> <td><input type="password" name="j_password"></td> </tr> </table> <input type="submit" value="Log In"> </form> </cfoutput>	ログインフォームを表示します。 CGI 変数を使用して、フォームの action 属性を構築します。クエリー文字列変数は疑問符 (?) の後に指定します。"loginform.cfm" は "Application.cfc" で cfinclude タグを使用してアクセスされているので、この CGI 変数は最初にリクエストされたページの変数を表します。 ユーザー ID とパスワードをリクエストし、ユーザーの入力内容を newurl 変数によって指定されたページに送信します。 フィールド名 j_username および j_password を使用します。これらの値を持つフォームフィールドが、ColdFusion によって cflogin タグ内の cflogin.name および cflogin.password 変数に自動的に挿入されます。

例 : "securitytest.cfm"

"securitytest.cfm" ページでは、アプリケーションページにおける ColdFusion のユーザー承認機能の使用方法を示します。"Application.cfc" では、ページの内容を表示する前に、認証されたユーザーが存在することを確認します。"securitytest.cfm" ページでは、IsUserInAnyRole 関数と GetAuthUser 関数を使用して、表示する情報を制御します。

"securitytest.cfm" ページは、次のように構成されます。

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Security test page</title>
</head>

<body>
<cfoutput>
    <h2>Welcome #GetAuthUser()#!</h2>
</cfoutput>

ALL Logged-in Users see this message.<br>
<br>
<cfscript>
    if (IsUserInRole("Human Resources"))
        WriteOutput("Human Resources members see this message.<br><br>");
    if (IsUserInRole("Documentation"))
        WriteOutput("Documentation members see this message.<br><br>");
    if (IsUserInRole("Sales"))
        WriteOutput("Sales members see this message.<br><br>");
    if (IsUserInRole("Manager"))
        WriteOutput("Managers see this message.<br><br>");
    if (IsUserInRole("Employee"))
        WriteOutput("Employees see this message.<br><br>");
    if (IsUserInRole("Contractor"))
        WriteOutput("Contractors see this message.<br><br>");
</cfscript>

</body>
</html>
```

コードの説明

"securitytest.cfm" ページの CFML コードおよびその機能について、次の表で説明します。

コード	説明
<cfoutput> Welcome #GetAuthUser()#! </cfoutput>	ユーザーのログイン ID を含む挨拶を表示します。
ALL Logged-in Users see this message. 	このメッセージはすべてのユーザーに表示されます。このページは、ユーザーがログインするまで表示されません。

<cfscript>
if (IsUserInRole("Human Resources"))
WriteOutput("Human Resources members see this message.
 	
");
if (IsUserInRole("Documentation"))
WriteOutput("Documentation members see this message.
 	
");
if (IsUserInRole("Sales"))
WriteOutput("Sales members see this message.
 	
");
if (IsUserInRole("Manager"))
WriteOutput("Managers see this message.
 	
");
if (IsUserInRole("Employee"))
WriteOutput("Employees see this message.
 	
");
if (IsUserInRole("Contractor"))
WriteOutput("Contractors see this message.
 	
");
</cfscript>
	ユーザーが有効なロールに所属しているかどうかをテストします。ユーザーがいずれかのロールに所属している場合は、ロール名を含むメッセージを表示します。
	所属するロールに応じた 1 つのメッセージがユーザーに表示されます。

LDAP ディレクトリによるセキュリティ情報の管理

LDAP ディレクトリは、セキュリティ情報の保管によく使用されます。次の cflogin タグの例では、LDAP ディレクトリを確認し、ユーザーを認証してユーザーロールを取得します。

LDAP ディレクトリを ColdFusion で使用する方法の詳細については、452 ページの「LDAP ディレクトリの管理」を参照してください。

```
 <cfapplication name="Orders" sessionmanagement="Yes" loginstorage="Session">
<cflogin>
    <cfif isDefined("cflogin")>
        <!--- setting basic attributes --->
        <cfset LDAP_root = "o=mycompany.com">
        <cfset LDAP_server = "ldap.mycompany.com">
        <cfset LDAP_port = "389">

        <!--- Create the prefix and suffix parts of the user's DN. --->
        <cfset userPrefix = "cn=">
        <cfset userSuffix = ",ou=Users,o=mycompany.com">

        <!--- Concatenate the user's DN and use it to authenticate. --->
        <cfset LDAP_username = userPrefix&cflogin.name&userSuffix>

        <!--- This filter will look for groups for containing the user's ID. --->
        <cfset userfilter =
            "(&(objectClass=groupOfUniqueNames)(uniqueMember=#LDAP_username#))">

        <!--- Search for groups containing the user's dn.
            The groups represent the user's roles.
            NOTE: Your LDAP permissions must allow authenticated users to search.
            groups. --->
        <cftry>
            <cfldap action="QUERY"
                name="auth"
                attributes="cn"
                referral="yes"
                start="#LDAP_root#"
                scope="SUBTREE"
                server="#LDAP_server#"
                port="#LDAP_port#"
                filter="#userfilter#"
                username="#LDAP_username#"
```

```
                password="#cflogin.password#"
            >
            <cfcatch type="any">
                <cfif FindNoCase("Invalid credentials", cfcatch.detail)>
                    <cfoutput>
                        <script>alert("User ID or Password invalid for user:
                            #cflogin.name#")</script>
                    </cfoutput>
                    <cfabort>
                <cfelse>
                    <cfoutput>
                        <script>alert("Unknown error for user: #cflogin.name#
                            #cfcatch.detail#")</script>
                    </cfoutput>
                    <cfabort>
                </cfif>
            </cfcatch>
        </cftry>

<!--- If the LDAP query returned a record, the user is valid. --->
        <cfif auth.recordcount>
            <cfloginuser name="#cflogin.name#" password="#cflogin.password#"
                roles="#valueList(auth.cn)#">
        </cfif>
    </cfif>
</cflogin>
```

コードの説明

このコードおよびその機能について、次の表で説明します。簡略化するために、コメントおよび一部のタブ文字は省略しています。

コード	説明
<cflogin> <cfif isDefined("cflogin")> <!--- setting basic attributes ---> <cfset LDAP_root = "o=mycompany.com"> <cfset LDAP_server = "ldap.mycompany.com"> <cfset LDAP_port = "389"> <!--- Create the prefix and suffix parts of the user's DN. ---> <cfset userPrefix = "cn="> <cfset userSuffix = ",ou=Users,o=mycompany.com"> <!--- Concatenate the user's DN and use it to authenticate. ---> <cfset LDAP_username = userPrefix&cflogin.name&userSuffix> <!--- This filter will look for groups for containing the user's ID. ---> <cfset userfilter = "(&(objectClass=groupOfUniqueNames)(uniqueMember=#LDAP_username#))">	cflogin タグ本文を開始します。cfldap タグで属性として使用するいくつかの変数に値を設定します。 LDAP サーバーへのバインドに使用する識別名 (DN) を作成するための、接頭辞と接尾辞の値を設定します。 cflogin.name に接頭辞と接尾辞を連結してユーザーのバインド DN を作成します。この変数は LDAP サーバーに対してユーザーを認証するために使用します。 ディレクトリを検索してユーザーの所属グループを取得するためのフィルタを設定します。グループはユーザーの組織内におけるロールを表します。
<cftry> <cfldap action="QUERY" name="auth" attributes="cn" referral="yes" start="#LDAP_root#" scope="SUBTREE" server="#LDAP_server#" port="#LDAP_port#" filter="#userfilter#" username="#LDAP_username#" password="#cflogin.password#" >	cftry ブロックで、ユーザーの連結 DN を使用して LDAP サーバーに対する認証を実行し、そのユーザーが属するグループの一般名 (cn) 属性を取得します。認証が失敗した場合は、LDAP サーバーからエラーが返されます。 メモ : このクエリーで結果が返されるためには、認証済みユーザーがグループを検索することが LDAP 権限により許可されている必要があります。
<cfcatch type="any"> <cfif FindNoCase("Invalid credentials", cfcatch.detail)> <cfoutput> <script>alert("User ID or Password invalid for user: #cflogin.name#")</script> </cfoutput> <cfabort> <cfelse> <cfoutput> <script>alert("Unknown error for user: #cflogin.name# #cfcatch.detail#")</script> </cfoutput> <cfabort> </cfif> </cfcatch> </cftry>	例外を検出します。 エラー情報に "invalid credentials" という文字列が含まれているかどうかをテストします。含まれている場合は、識別名またはパスワードのいずれかが無効であることを意味します。その場合は、ダイアログボックスにその問題を示すエラーメッセージを表示します。 それ以外の場合は、一般的なエラーメッセージを表示します。 エラーを検出した場合は、エラーの説明を表示した後、cfabort タグでリクエストの処理を中止します。 cfcatch ブロックと cftry ブロックを終了します。
<cfif auth.recordcount> <cfloginuser name="#cflogin.name#" password="#cflogin.password#" roles="#valueList(auth.cn)#"> </cfif> </cfif> </cflogin>	承認クエリーから有効なレコードが返されたら、ユーザーをログインさせます。valueList 関数を使用して、そのユーザーの所属グループのカンマ区切りリストを作成し、cfloginuser の roles 属性にそのリストを渡します。 先頭の cfifisDefined("cflogin") ブロックを終了します。 cflogin タグ本文を終了します。

ColdFusion 10 でのセキュリティの機能強化

ColdFusion 10 でのセキュリティの機能強化により、XSS および CSRF 攻撃に対する脆弱性が低下します。この機能強化は、ColdFusion セッションの効果的な管理にも役立ちます。このリリースには、他の脆弱性を軽減する修正も含まれます。

XSS 攻撃

クロスサイトスクリプティング（XSS）攻撃は、Web ブラウザーによって実施されるクライアントサイドのセキュリティメカニズムをバイパスします。これらのメソッドでは、エンコーディングに Open Web Application Security Project（OWASP）の Enterprise Security API を使用します。攻撃者は、悪意のあるスクリプトを Web ページに挿入して、ブラウザーに格納されている情報にアクセスします。

- cform タグ内の name 属性の値として許される文字は、英数字、_（アンダースコア）、-（ハイフン）、:（コロン）、.（ドット）のみです。これにより、scriptsrc フィールドに格納される XSS を防ぎます。
- XSS 攻撃に対する脆弱性を軽減するために追加される新しいエンコーディングメソッドは、encodeForHTML、encodeForHTMLAttribute、encodeForJavaScript、encodeForCSS、および encodeForURL です。コンテキストに応じてユーザー入力をエンコードします。入力文字列をデコードするため、canonicalize メソッドが追加されています。

CSRF 攻撃

クロスサイトリクエストフォージェリ（CSRF）は、ユーザーが認証された Web アプリケーションで、ユーザーに意図しない操作を強制的に実行させるものです。例えば、電子メールやチャットを使用してアクセス権のあるユーザーに URL を送信します。URL リンクをクリックすると、ユーザーは攻撃者が意図する操作を実行させられます。

- 次のメソッドを使用して、CSRF に対する脆弱性を軽減できます。
  - CSRFGenerateToken：ランダムなトークンを返し、セッション内に格納します。
  - CSRFVerifyToken：指定されたトークンとキーを、セッションに格納されているものと同じかどうか検証します。

  例：CSRFGenerateToken

  次の例では、ユーザーは値を入力して送信します。ページではトークンが生成され、別の ColdFusion ページが呼び出されます。

```
<cfset csrfToken=CSRFGenerateToken() />
<cfform method="post" action="sayHello.cfm">
    <cfinput name="userName" type="text" >
    <cfinput name="token" value="#csrfToken#" type="hidden" >
    <cfinput name="submit"  value="Say Hello!!" type="submit" >
</cfform>
```

  次のページ sayHello.cfm では、生成されたトークンが検証され、CSRFverifyToken(token) の出力が表示されます。

```
<cfset token=form.token>
    <cfset validate = CSRFverifyToken(token)>
<cfoutput >#validate#</cfoutput>
```

  注意：CSRF に対する保護のためには SessionManagement を有効にします。管理コンソールでセッション変数を無効にすると、CSRF 保護が無効になります。

セッションの向上

ColdFusion のセッション Cookie を効率よく管理できます。

CF セッション Cookie（CFID、CFTOKEN、CFAuthorization_<app-name>）

セッション Cookie を管理するための新しい機能は次のとおりです。

- ColdFusion セッション Cookie の次のプロパティを、サーバーレベルまたはアプリケーションレベルで設定できます。
  - httponly：デフォルト値は true
  - secure：デフォルト値は false
  - domain

- timeout：デフォルト値は 30 年

次の例で示すように、Application.cfm で設定を構造体として指定することにより、アプリケーションレベルでセッション Cookie を設定できます。

```
<cfset cookiest = {httponly='true', timeout=createTimeSpan(0, 0, 0, 10),
secure='true',domain=".domain.com"}>
<cfset cookieast = {timeout=createTimeSpan(0, 0, 00, 10)}>
<cfapplication name="sessionCookies_appcfm_allSetting" sessionmanagement="Yes"
sessiontimeout="#createTimeSpan(0,0,03,0)#" scriptprotect="all" sessioncookie=#cookiest#
authcookie=#cookieast#>
```

注意：アプリケーションレベルの設定は、サーバーレベルの設定より優先されます。

サーバーレベルでセッション Cookie を設定するには、次の新しい管理 API を使用し、パラメーター getRuntimeProperty および setRuntimeProperty を指定します。これらのメソッドは CFIDE¥adminapi¥runtime.cfc ファイルで使用できます。

次の例では、getRuntimeProperty() メソッドを使用して Cookie のパラメーターを取得する方法を説明します。Cookie パラメーターの設定は、setRuntimeProperty() メソッドを使用して同様の方法で行います。

```
GetRuntimeProperty("HttpOnlySessionCookie");
GetRuntimeProperty("SecureSessionCookie");
GetRuntimeProperty("SessionCookieTimeout");
GetRuntimeProperty("SessionCookieDomain");
```

- Application.cfc で次のように指定することにより、アプリケーションレベルでセッション Cookie を設定できます。
  - this.sessioncookie.httponly="true"
  - this.sessioncookie.secure="true"
  - this.sessioncookie.domain="value"
  - this.sessioncookie.timeout="value"（日数）
  - this.authcookie.timeout= "value"（デフォルトでは -1。Cookie はブラウザーが開くまで有効です。）

  注意：SetDomainCookies プロパティを定義し、アプリケーションおよびサーバーのレベルでドメインに対するセッション Cookie を設定できます。この場合、優先順位は、アプリケーションの設定、サーバーの設定、SetDomianCookies プロパティの順になります。

  注意: ColdFusion 9.01 で追加されたシステムプロパティ coldfusion.sessioncookie.httponly=true はこのリリースでは不要になったため、削除されました。

  注意: CFCookie タグおよび CFHeader タグを使用した ColdFusion Cookie および承認 Cookie の操作は、アプリケーションレベルまたはサーバーレベルの設定で制御できます。application.cfc または application.cfm に、sessioncookie.disableupdate=true および authcookie.disableupdate=true を追加します。CFIDE¥adminapi¥runtime.cfc で、GetRuntimeProperty("CFInternalCookieDisableUpdate") メソッドおよび SetRuntimeProperty("CFInternalCookieDisableUpdate", "true/false") メソッドを使用することもできます。ColdFusion Administrator でタグを設定するには、サーバーの設定／メモリ変数／セッション Cookie 設定を使用します。「ColdFusion のタグ / 関数を使用して ColdFusion の内部 Cookie を更新できないようにします」を選択または選択解除します。

- <cflogin> タグでは、パスワードはキャッシュに保存されます。認証済みのセッションを長くするために、cfhome/lib/auth-ehcache.xml ファイル内にある authcache を変更して、diskPeristent を有効にすることができます。永続化のために使用するディレクトリは、セキュリティ保護されている必要があります。

CRLF 攻撃

キャリッジリターン（ASCII 13、¥r）ラインフィード（ASCII 10、¥n）（CRLF）攻撃は、HTTP レスポンス分割（HTTP Response Splitting）とも呼ばれます。この攻撃では、HTTP ストリームに CRLF が挿入されます。通常は、HTTP パラメーターまたは URL を変更することで行われます。それにより、CRLF をアプリケーションに挿入し、応答に含めることができます。プロキシおよびキャッシュによって解釈された CRLF は、重大なセキュリティの問題をもたらします。

- cfheader、cfcontent、cfmail、cfmailpart、cfmailparam などのヘッダーを作成するタグに、CRLF 攻撃に対する保護が追加されています。

情報開示

この機能は、情報の開示におけるセキュリティ関連の問題を改善します。

- このバージョンでは、すべてのサービスに対するパスワードが暗号化されます。

  パスワードのシードの変更は、実行中のサーバーに負荷がかかっていないときにのみ行ってください。そうしないと、サーバーで予期しない動作が発生します。

新しい HMAC メソッド

ハッシュベースのメッセージ認証コード（HMAC）は、データの整合性および送信されるメッセージの信憑性を確認するために使用されます。暗号ハッシュ機能と秘密鍵が組み合わされています。暗号ハッシュ機能には、Message Digest 5（MD5）、Secure Hash Algorithm（SHA）などを使用できます。

CFScript での cfcookie のサポート

CFScript では Cookie を構造体として設定できます。次のパラメーターを設定できます。

- expires
- value
- name
- secure
- httponly
- domain
- path
- preservecase
- encodevalue

例 1

```
<cfscript >
cookie.mytest = {value="Adobe",expires="10",secure="true",domain=".adobe.com",path="/coldfusion"};
</cfscript>
```

例 2

```
<cfscript>
    cookie_example = structNew();
    cookie_example.value = "example";
    cookie_example.expires = "10";
    cookie_example.secure = "true";
    cookie.mycookie = cookie_example;
</cfscript>
```

その他の変更

- httponly Cookie のサポートは、J2EE 1.6 をサポートする Tomcat で利用できます。
- 新しいパラメーター numIteration が、hash() メソッドに追加されています。

  このオプションのパラメーターでは、ハッシュを繰り返す回数を指定します。更新された hash() メソッドは次のとおりです。

  ```
  hash(Object message, String algorithm, String encoding, int no-of-iterations)
  ```

  1 番目の引数には、String 型または Byte[] 型のオブジェクトを指定できます。

  例

  ```
  <!--- Do the following if the form is submitted. --->
  <cfif IsDefined("Form.UserID")>
  <!--- query the data base. --->
  <cfquery name = "CheckPerson" datasource = "cfdocexamples">
      SELECT PasswordHash FROM SecureData WHERE UserID = <cfqueryparam value = "#Form.userID#" cfsqltype
  = 'CF_SQL_VARCHAR'>
  </cfquery>
  <!--- Compare query PasswordHash field and the hashed form password  and display the results. --->
  <cfoutput>
      <cfif Hash(Form.password, "SHA","",4) is not checkperson.passwordHash> User ID #Form.userID# or
  password is not valid. Try again.
      <cfelse> Password is valid for User ID #Form.userID#.
      </cfif>
  </cfoutput>
  </cfif>
  <!--- Form for entering ID and password. --->
  <form action="#CGI.SCRIPT_NAME#" method="post">
  <b>User ID: </b> <input type = "text" name="UserID" ><br>
  <b>Password: </b> <input type = "text" name="password" ><br><br>
  <input type = "Submit" value = "Encrypt my String"> </form>
  ```

- 強化された <cflogin> および承認の Cookie。

  クラスタ環境では、スティッキーセッションを有効にします。スティッキーセッションが有効でない場合は、次のようにします。

  注意：分散キャッシュを追加する別の方法が Ehcache の Web サイトに記載されています。

  - クラスタ環境用の認証キャッシュを設定します。クラスタ内の各キャッシュに対して以下を行います。

    1. CF_instance/lib/auth-ehcache.xml を開きます。
    2. 「Mandatory Default Cache configuration」という文字列を探し、次のエントリを追加します。

       ```
       <!-- distributed caching settings part 1 starts -->
       <cacheManagerPeerProviderFactory
           class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
           properties="peerDiscovery=automatic, multicastGroupAddress=230.0.0.1,
           multicastGroupPort=4446, timeToLive=32"/>
       <cacheManagerPeerListenerFactory
       class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"
           properties="port=40002, socketTimeoutMillis=3000"/>
       <!-- distributed caching settings part 1 ends -->
       ```

    3. 上のエントリで、cacheManagerPeerListenerFactory プロパティポートを更新します。インスタンスごとに固有である必要があります。
    4. 「<cache name="authcache"」という文字列を探します。
    5. 「clearOnFlush="true">」の後に次のエントリを追加します。

```
clearOnFlush="true">
<cacheEventListenerFactory class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
properties="replicateAsynchronously=false, replicatePuts=true, replicatePutsViaCopy=false,
replicateUpdates=true, replicateUpdatesViaCopy=true, replicateRemovals=true"
propertySeparator=","/>
</cache>
```

注意：ColdFusion Administrator は、クラスタセットアップをサポートしていません。

注意：Remember Me タイプの機能の場合、または認証キャッシュを長時間有効にしておく場合は、認証キャッシュの設定を変更します。例えば、タイムアウトを長くしたり、永続キャッシュを有効にしたりします。

注意：認証のセキュリティを強化するには、新しい Cookie 設定を使用します。要件に応じて、サーバーレベルまたはアプリケーションレベルで設定を行います。

- 次の場合は、一方の ColdFusion Administrator からログアウトします。
  - 同じホストから、ColdFusion（10）Administrator と古いバージョンの ColdFusion Administrator にログインしている。
- RDS アクセスを使用している場合は、ColdFusion Administrator で、データソースとセキュリティ保護されたファイルパス権限を設定できます。
- 新しいサンドボックスのデフォルト値は、より安全になるように変更されています。
- ColdFusion では、action 属性が <cfform> タグで指定されていない場合、現在の URL を使用して同じものを生成しません。問題がある場合は、ColdFusion-install-dir¥cfusion¥bin¥jvm.config ファイルに -Dcoldfusion.encodeformaction=true というエントリを追加してください。

グローバル化アプリケーションの開発

Adobe ColdFusion では、インターネット用のダイナミックアプリケーションを開発できます。場合によっては、複数の国および地域のユーザーが ColdFusion アプリケーションにアクセスすることがあります。そのようなアプリケーションを設計する場合は、さまざまな地域のユーザーに対応できるように、アプリケーションのグローバル化を検討する必要があります。

グローバル化の概要

グローバル化を行えば、サポートされている言語を使用しているすべてのユーザーがアプリケーションを使用できるようになります。たとえば、アプリケーションの実装に使用する文字セット以外の文字セットでもデータを入力できるようにして、日本語のフォームデータを送信できる英語の Web サイトを作成したり、韓国語のパラメータ値をリクエスト URL に含められるようにすることができます。

また、数値、日付、通貨、および時刻が含まれているデータを処理するアプリケーションでは、国や地域によって形式が異なるこれらのデータ型を、適切に処理することができます。

さらに、英語以外の言語でアプリケーションを開発することもできます。たとえば、デフォルトの文字エンコードが Shift-JIS で、ColdFusion ページに日本語の文字が表示され、インターフェイスが日本語で表示される日本語のアプリケーションを開発できます。

アプリケーションをグローバル化するには、必要に応じて次のようなアクションを行います。

- 複数の言語で入力が行えるようにします。
- 複数のロケールで形式設定された日付、時刻、通貨、および数値を処理できるようにします。
- 複数の文字セットが使用されているフォーム、データベース、HTTP 接続、電子メールメッセージなどのデータを処理できるようにします。

- 英語以外の言語が含まれている ColdFusion ページを作成します。

グローバル化の定義

グローバル化の定義にはさまざまなものがありますが、ここでは、複数の言語で共有可能な基盤に、できるだけ多くのアプリケーション機能を組み込む設計プロセスのことをグローバル化と呼びます。

グローバル化は次の 2 つの部分から構成されます。

国際化対応 表現形式に関係なくデータを認識し、処理し、応答できる、言語に依存しないアプリケーション機能を開発します。つまり、ある言語でできることは他の言語でもできるようにします。たとえば、テキストのコピー & ペーストを考えてみます。コピー & ペースト操作は、テキストに含まれている言語に関係なく実行できることが求められます。ColdFusion アプリケーションの処理ロジックでも、数値計算やデータベースクエリーなどの操作を言語に関係なく実行できることが求められます。

ローカライズ 言語に依存しない共有機能に、ロケール固有のインターフェイスを適用します。このインターフェイスは、スキンとも呼ばれます。たとえば、特定の言語 (日本語など) のインターフェイスを構成する一連のメニュー、ボタン、およびダイアログボックスを開発します。開発したインターフェイスは、基盤となるアプリケーションの言語に依存しない機能と組み合わせて使用します。ローカライズの一環として、言語固有の方法で表現された入力を処理し、その言語に合ったレスポンスを出力する機能を作成します。

アプリケーションのグローバル化の重要性

インターネットには国境がありません。顧客は、世界中のあらゆる場所からいつでも Web サイトにアクセスできます。サイトへのアクセスを英語などの特定の言語に限定しない場合は、グローバル化の問題を検討してください。

アプリケーションをグローバル化する理由の 1 つは、エラーと顧客の混乱を避けるためです。たとえば、1/2/2003 という形式の日付は、米国では 2003 年 1 月 2 日と解釈されますが、ヨーロッパ諸国では 2003 年 2 月 1 日と解釈されます。

アプリケーションをグローバル化するもう 1 つの理由は、通貨を正しい形式で表示するためです。商品の正しい価格が 15,000 円ではなく、15,000 米ドルであることがわかったときに、お客様がどう感じるかを考えてみてください。

また、顧客からのフィードバックやその他のテキスト入力を Web サイトで受け付けることもあります。その場合は、複数の言語のさまざまな文字セットでフィードバックが行えるように検討する必要があります。

ColdFusion でのグローバル化のサポート

ColdFusion は Java で実装されています。Java アプリケーションである ColdFusion では、Java のグローバル化機能を使用します。たとえば、ColdFusion の内部では、Unicode 文字セットを使用してすべての文字列が保管されています。Unicode を使用しているので、ColdFusion では、どのような言語のテキストデータも表現できます。

また、ColdFusion には、アプリケーションのグローバル化をサポートするためのタグや関数が多数用意されています。これらのタグや関数を使用すると、ロケールの設定、日付および通貨形式の変換、ColdFusion ページの出力エンコードの制御などが行えます。

文字セット、文字エンコード、ロケール

グローバル化について検討する際は、アプリケーションで認識する文字セットまたは文字エンコードと、アプリケーションでデータの形式設定に使用するロケールという 2 つの項目について考慮する必要があります。

文字セットは文字のコレクションです。たとえば、ラテンアルファベットは英語の記述に使用する文字セットであり、A ～ Z の小文字と大文字がすべて含まれています。フランス語の文字セットには、英語で使用される文字セットの他に、"é"、"à"、"ç" などの特殊文字が含まれています。

日本語では、ひらがな、カタカナ、および漢字の 3 種類の文字が使用されます。ひらがなとカタカナは、それぞれ 46 個の文字と 2 つのアクセント記号が含まれている音標文字です。漢字には、日本で独自に発展した中国の表意文字が含まれています。日本語では 10,000 個以上の異なる文字がサポートされており、英語よりもはるかに大きな文字セットを使用します。

ColdFusion アプリケーションでテキストを処理するためには、そのテキストで使用されている文字セットをアプリケーションが認識する必要があります。文字エンコードとは、文字セットの定義と、データを表すために使用されるデジタルコードとの間のマッピングのことです。

文字セットと文字エンコードという用語は同じ意味で使用されることも多く、ほとんどの場合、特定の文字エンコードでエンコードする文字セットは 1 つのみです。ただし、常にそうであるとは限りません。Unicode 文字セットの文字エンコードは複数あります。文字エンコードの詳細については、368 ページの「文字エンコードについて」を参照してください。

注意：ColdFusion では、属性名、構造フィールドキー、および関数パラメータ名の文字エンコードを示すために、文字セットという用語を使用します。

ロケールは、ユーザーの言語と文化の設定を表します。ロケールによって、日付、通貨、時刻、数値データの表示形式が制御されます。たとえば、ロケール English (US) では、次のように通貨値が表示されます。

$100,000.00

一方、ロケール Portuguese (Portugese) では、次のように通貨値が表示されます。

R$ 100.000

日付、時刻、通貨、および数値データを顧客に正しく表示するには、顧客のロケールを把握する必要があります。ロケールの詳細については、370 ページの「ロケール」を参照してください。

文字エンコードについて

文字エンコードとは、文字セットに含まれている個々の文字と、コンピュータで表現可能な数値との間のマッピングのことです。これらの数値はシングルバイトまたはマルチバイトで表すことができます。たとえば、ASCII エンコードでは、7 ビットを使用してラテンアルファベット、句読点、および制御文字を表します。

日本語のテキストを表すには、Shift-JIS、EUC-JP、ISO-2022-JP などの日本語エンコードを使用します。これらのエンコードはわずかに異なりますが、いずれの文字セットにも、日本語で使用される約 10,000 個の文字を含む一般的な文字セットが含まれています。

文字エンコードでは、次の用語が使用されます。

SBCS シングルバイト文字セット。ASCII または ISO 8859-1 などの、1 文字につき 1 バイトでエンコードされる文字セットです。

DBCS ダブルバイト文字セット。Shift-JIS などの、文字セットを 2 バイト以下でエンコードする方法です。Shift-JIS などの、2 バイト文字エンコードスキームの多くには、1 バイトでエンコードされる文字と 2 バイトでエンコードされる文字が混在します。一方、UCS-2 などはすべての文字で 2 バイトを使用します。

MBCS マルチバイト文字セット。文字によって異なる長さのバイト列でエンコードされる文字セットです。UTF-8 などがこれに該当します。

次の表に一般的な文字エンコードを示しますが、ブラウザおよび Web サーバーではこれ以外にも多くの文字エンコードがサポートされています。

エンコード	タイプ	説明
ASCII	SBCS	英語およびインドネシア語で使用される 7 ビットエンコード。
Latin-1 (ISO 8859-1)	SBCS	多くの西ヨーロッパ言語で使用される 8 ビットエンコード。
Shift_JIS	DBCS	16 ビットの日本語エンコード。 メモ : CFML 属性でこの名前を指定するときは、ハイフン (-) ではなくアンダースコア文字 (_) を使用します。

エンコード	タイプ	説明
EUC-KR	DBCS	16 ビットの韓国語エンコード。
UCS-2	DBCS	2 バイトの Unicode エンコード。
UTF-8	MBCS	マルチバイトの Unicode エンコード。ASCII は 7 ビット、ヨーロッパおよび多くの中東言語で使用される非 ASCII 文字は 2 バイト、ほとんどのアジア言語の文字は 3 バイトです。

W3C (World Wide Web Consortium) では、インターネットでサポートされているすべての文字エンコードのリストを管理しています。この情報には、www.w3.org/International/O-charset.html でアクセスできます。

コンピュータでは、文字エンコードの変換が必要になることがよくあります。特に、インターネットでよく使用されている文字エンコードは、Java または Windows では使用されていません。インターネットでは、シングルバイト文字セット、またはマルチバイト文字セット (シングルバイト文字を使用可能な DBCS 文字セットも含む) が一般的に使用されています。これらの文字セットは、各文字が最小限必要なバイト数だけを使用するため、データを最も効率的に送信できます。現在、Web では Latin 文字が最も多く使用されており、Web で使用される文字エンコードのほとんどが 1 バイトの Latin 文字を表します。

しかし、コンピュータで最も効率的にデータが処理できるのは、各文字が同じバイト数を使用している場合です。したがって、Windows でも Java でも、内部処理には 2 バイトエンコードが使用されています。

Java Unicode 文字エンコード

ColdFusion の内部では、Java Unicode 規格を使用して文字データを表現しています。この規格は Unicode 文字セットの UCS-2 エンコードに対応します。Unicode 文字セットは、主要なヨーロッパおよびアジア文字セットをすべて含む、多くの言語を表現することができます。したがって、ColdFusion では、Unicode でサポートされているすべての言語で表されたテキストの受信、保管、処理、および表示を行うことができます。

ColdFusion ページの処理に使用される Java 仮想マシン (JVM) は、ColdFusion ページや他の情報ソースで使用されている文字エンコードを UCS-2 に変換します。ColdFusion のページやデータでサポートされるエンコードの種類は、使用する JVM によって異なりますが、Web で使用されるほとんどのエンコードはサポートされています。同様に、JVM はその内部 UCS-2 表現を、クライアントへのレスポンスに使用する文字エンコードに変換します。

デフォルトでは、ColdFusion は、ブラウザに送信するテキストデータを UTF-8 形式で表します。UTF-8 は可変長のエンコードで Unicode 文字セットを表します。ASCII 文字は 1 バイトで送信されます。ヨーロッパおよび中東の文字のほとんどは 2 バイトで送信され、日本、韓国、および中国の文字は 3 バイトで送信されます。UTF-8 の利点の 1 つは、シングルバイト ASCII 文字のみを処理できるシステムで認識される形式で ASCII 文字セットデータを送信するとともに、マルチバイト文字表現を処理する柔軟性を備えていることです。

ColdFusion が返すテキストデータのデフォルト形式は UTF-8 ですが、Java でサポートされている任意の文字セットでページを返すように指定することもできます。たとえば、日本語 Shift-JIS 文字セットを使用してテキストを返すことができます。同様に、ColdFusion はさまざまな文字セットが含まれているデータを処理できます。詳細については、372 ページの「サーバー出力のページエンコードの決定」を参照してください。

文字エンコードの変換の問題

サポートされている文字セットは文字エンコードによって異なるので、あるエンコードで受信したテキストを別のエンコードで表示すると、エラーが発生する可能性があります。たとえば、Windows Latin-1 文字エンコードである Windows-1252 には、16 進表現で範囲 80-9F の文字が含まれますが、ISO 8859-1 にはその範囲の文字は含まれません。したがって、次のような状況では、範囲 80-9F にある文字 (ユーロ記号 Ä など) は正しく表示されません。

- あるファイルが Windows-1252 でエンコードされており、範囲 80-9F の文字が含まれています。
- ColdFusion で、cffile タグに Windows-1252 エンコードを指定して、このファイルを読み込みます。
- ColdFusion で、cfcontent タグに ISO-8859 を指定して、このファイルのコンテンツを表示します。

その他の文字エンコード間の変換、たとえば日本語の Windows デフォルトエンコードのファイルを読み込んで Shift-JIS で表示する場合にも、同様の問題が生じる可能性があります。こうした問題を回避するために、入力と表示には同じエンコードを使用してください。

ロケール

ロケールは、ユーザーの言語と文化の設定を表します。ロケールによって、次の項目の形式が制御されます。

- 日付
- 時刻
- 数値
- 通貨

ColdFusion では、使用される JVM でサポートされているすべてのロケールがサポートされています。

注意：現在の JVM バージョン (1.4.2 まで) では、アラビア語ロケールで使用されるアラビアヒンズー数字、ヒンズー語ロケールで使用されるヒンズー数字などの、ローカルな数字はサポートされていません。ColdFusion はすべてのロケールでアラビア数字を使用します。

ロケール名

ColdFusion では、ロケール名を指定するために 2 つの形式がサポートされています。1 つは標準の Java ロケール名です。もう 1 つは、ColdFusion 6.1 以前で使用されていた ColdFusion ネーミング規則です。

- すべてのロケールは、次の要素を組み合わせた名前で指定できます。
  - 言語を識別する 2 つの小文字。たとえば、英語は en、中国語は zh。
  - オプションとして、下線と 2 つの大文字。これにより、地域による言語の違いを識別できます。たとえば、米国は US、香港は HK と表します。

    たとえば、en_US は米国英語を、es_MX はメキシコスペイン語を表します。Sun 1.4.2 JVM でサポートされている Java ロケール識別子の一覧とその意味については、
    http://java.sun.com/j2se/1.4.2/docs/guide/intl/locale.doc.html を参照してください。

ColdFusion MX 7 よりも前の ColdFusion では、サポートされているロケールセットが限定されており、言語名の後に地域を丸括弧で囲んだ識別子を使用していました (地域を指定しない言語もあります)。たとえば、English (US) や German (Standard) のように指定できます。こうした名前は現在でも引き続きサポートされています。一覧については、『CFML リファレンス』の SetLocale を参照してください。

Server.coldfusion.supportedlocales 変数は、指定可能なロケール名のカンマ区切りのリストです。

また、ユーザーが認識できる形式でロケール名を返す、GetLocaleDisplayName 関数も用意されています。この関数により、français (フランス) など、ユーザーの言語を使用してロケールを表示することができます。

ロケールの決定

ColdFusion では、ロケールの値を次のように決定します。

- デフォルトでは、JVM ロケールが使用されます。デフォルトの JVM ロケールはオペレーティングシステムのロケールです。ColdFusion Administrator の [Java と JVM の設定] ページの [JVM 引数] フィールドを使用すれば、ColdFusion の JVM ロケール値を明示的に設定できます。次に例を示します。

  ```
  -Duser.language=de -Duser.country=DE.
  ```

- SetLocale 関数で設定されたロケールは現在のリクエストの終了まで、または、現在のリクエストの途中で再度の SetLocale 呼び出しによる再設定が行われるまで保持されます。

- リクエストに複数の SetLocale 関数がある場合、ロケールに依存する ColdFusion タグおよび関数 (**LS** で始まる関数など) は、現在のロケール設定に基づいてデータを形式設定します。レスポンスの Content-Language HTTP ヘッダの値は、リクエスタ (通常はクライアントブラウザ) にレスポンスを送信する前に最後に処理された SetLocale 関数によって決まります。ページをリクエストしたブラウザは、Content-Language ヘッダで指定されている言語の規則に従ってレスポンスを表示します。
- cfflush タグの後の SetLocale 関数は無視されます。

ロケールの使用

SetLocale 関数は、ColdFusion で日付、時刻、数値、および通貨値を出力するために使用するデフォルトの形式を決定します。ColdFusion の現在のロケール設定を取得するには、GetLocale 関数を使用します。または、GetLocaleDisplayName 関数を使用して、ユーザーが認識できる形式でロケール名を取得することができます。SetLocale をまだ呼び出していない場合、GetLocale は JVM のロケールを返します。

現在のロケールが与える影響は次の 2 つです。

- ColdFusion で出力される日付、時刻、通貨、または数値の形式設定に影響を与えます。ColdFusion ページでは、ロケールを何回でも変更して、さまざまなロケールの表記規則に従って情報を表示することができます。これによって、正しく形式設定された多様な通貨値を含むページを出力できます。
- ColdFusion がクライアントにページを返すときには、HTTP の Content-Language ヘッダが挿入されます。この情報は、ページの最後のロケール設定から取得されます。

注意：以前のバージョンの ColdFusion では、デフォルトのロケールはオペレーティングシステムのロケールではなく、常に英語でした。日本語版の ColdFusion では、デフォルトのロケールは日本語でした。

次のサンプルコードでは LSCurrencyFormat 関数を使用して、ColdFusion でサポートされているすべてのロケールの通貨単位を使用して 100,000 という値を出力します。このコードを実行すると、ブラウザに返されるデータに対するロケールの影響を確認できます。

```
 <p>LSCurrencyFormat returns a currency value using the locale convention.
<!--- loop through list of locales; show currency values for 100,000 units --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
    index = "locale" delimiters = ",">
<cfset oldlocale = SetLocale(locale)>
    <cfoutput><p><b><I>#locale#</I></b><br>
        Local: #LSCurrencyFormat(100000, "local")#<br>
        International: #LSCurrencyFormat(100000, "international")#<br>
        None: #LSCurrencyFormat(100000, "none")#<br>
        <hr noshade>
    </cfoutput>
</cfloop>
```

この例では、ColdFusion 変数の Server.Coldfusion.SupportedLocales を使用しています。この変数には、ColdFusion でサポートされているすべてのロケールのリストが含まれています。

ColdFusion でのリクエストの処理

ColdFusion ページを要求する HTTP リクエストを受信した ColdFusion は、リクエスト URL を物理的なファイルパスに解決し、ファイルのコンテンツを読み込んで解析します。ColdFusion ページは、使用している JVM でサポートされている任意の文字エンコードでエンコードできますが、ColdFusion で正しいエンコードが識別できるように指定する必要があります。

サーバー上の ColdFusion ページのコンテンツには、スタティックデータ (ColdFusion で処理されない HTML やプレーンテキストなど) と、CFML で記述されたダイナミックコンテンツがあります。スタティックコンテンツはブラウザへのレスポンスに直接書き込まれ、ダイナミックコンテンツは ColdFusion で処理されます。

Web サイトのデフォルトの言語が、その Web サイトに接続しているクライアントの言語と異なる場合があります。たとえば、フランス語のコンピュータから英語の Web サイトに接続することがあります。ColdFusion でレスポンスを生成するときは、ユーザーが期待している形式を使用する必要があります。この形式には、文字セットとロケールが含まれます。

次に、処理するファイルの文字セットを ColdFusion が判断する方法、およびクライアントに対するレスポンスの文字セットおよびロケールの決定方法について説明します。

ColdFusion ページの文字エンコードの判断

ColdFusion ページに対するリクエストを受け取ると、ColdFusion はそのページを開き、コンテンツを処理して、リクエスト送信元のブラウザに結果を返します。ColdFusion ページを処理するには、そのページコンテンツを ColdFusion が解釈する必要があります。

そのために ColdFusion が使用する情報の 1 つが、ColdFusion ページの BOM (Byte Order Mark) です。BOM はテキストストリームの先頭にある特殊文字で、ページで使用されているマルチバイト文字のバイトの順序を指定します。次の表に、一般的な BOM 値を示します。

エンコード	BOM シグネチャ
UTF-8	EF BB BF
UTF-16 ビッグエンディアン	FE FF
UTF-16 リトルエンディアン	FF FE

BOM 文字をサポートしているエディタを使用すれば、BOM 文字を CFML ページに簡単に挿入できます。Dreamweaver を含む多くの Web ページ開発ツールが BOM 文字の挿入をサポートしています。Dreamweaver では、[ページプロパティ] の [エンコーディング] の選択に基づいて BOM が自動的に設定されます。

ページに BOM を含めない場合は、cfprocessingdirective タグを使用してページの文字エンコードを設定できます。ただし、BOM が含まれているページに cfprocessingdirective タグを挿入する場合は、cfprocessingdirective タグで指定する情報が BOM と一致している必要があります。そうでない場合、ColdFusion によってエラーが返されます。

次の手順で、ColdFusion が ColdFusion ページのエンコード形式を認識する方法を説明します。

ページのエンコードの判断 (ColdFusion が行う処理)

1. ページで BOM が指定されている場合は、BOM が使用されます。

   ファイルには BOM 文字を設定することをお勧めします。

2. cfprocessingdirective タグが指定されている場合は、その pageEncoding 属性が使用されます。この属性の使用方法の詳細については、『CFML リファレンス』の cfprocessingdirective タグを参照してください。

3. いずれも指定されていない場合は、JVM のデフォルトのファイル文字エンコードが使用されます。デフォルトでは、これはオペレーティングシステムのデフォルト文字エンコードです。

サーバー出力のページエンコードの決定

クライアントにレスポンスを返すためには、レスポンスのデータに使用するエンコードを決定する必要があります。ColdFusion では、デフォルトで Unicode UTF-8 形式を使用して文字データが返されます。

ColdFusion ページ (.cfm ページ) のレスポンスには、デフォルトで Unicode UTF-8 が使用されます。これは、ページ内に HTML の meta タグを挿入しても影響を受けません。したがって、次のコード例では、レスポンスの文字セットは変更**されません**。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type"
    content="text/html;
    charset="Shift_JIS">
</head>
...
```

この例では、レスポンスに使用される文字セットは UTF-8 のままです。出力文字セットを設定するには、cfcontent タグを使用します。

ColdFusion ページで cfcontent タグを使用すると、その指定がレスポンスのデフォルトの文字エンコードよりも優先されます。文字セットなどのページ出力の MIME タイプを指定するには、次のように cfcontent タグの type 属性を使用します。

```
<cfcontent type="text/html charset=EUC-JP">
```

注意：ColdFusion には、HTTP リクエスト、リクエストヘッダ、ファイル、メールメッセージなどの特定の要素のエンコードを指定する属性もあります。詳細については、373 ページの「文字エンコードを制御するためのタグおよび関数」および 375 ページの「ColdFusion でのデータ処理」を参照してください。

以降のセクションでは、グローバル化に使用する ColdFusion タグおよび関数について説明し、個々のソースにおけるグローバル化の問題について解説します。

アプリケーションのグローバル化に使用するタグと関数

ColdFusion には、グローバル化アプリケーションの開発に使用できるタグおよび関数が多数あります。

文字エンコードを制御するためのタグおよび関数

次のタグおよび関数を使用して、ColdFusion が生成および解釈するテキストの文字エンコードを指定できます。

タグまたは関数	属性またはパラメータ	用途
cfcontent	type	結果をクライアントブラウザに返すためのエンコードを指定します。詳細については、372 ページの「サーバー出力のページエンコードの決定」を参照してください。
cffile	charset	ファイルに書き込むデータのエンコード、または読み込むファイルのエンコードを指定します。詳細については、377 ページの「ファイルデータ」を参照してください。
cfheader	charset	HTTP ヘッダ値をエンコードするための文字エンコードを指定します。
cfhttp	charset	HTTP リクエストの文字エンコードを指定します。
cfhttpparam	mimeType	ファイルの MIME メディアタイプを指定します。ファイルの文字エンコードを位置表記として指定することができます。
cfmail	charset	ヘッダを含めたメールメッセージの文字エンコードを指定します。
cfmailpart	charset	マルチパート仕様のメールメッセージの 1 つのパートの文字エンコードを指定します。
cfprocessingdirective	pageEncoding	ColdFusion で処理するページの文字エンコードを識別します。詳細については、372 ページの「ColdFusion ページの文字エンコードの判断」を参照してください。
CharsetDecode	encoding	指定したエンコードの文字列をバイナリオブジェクトに変換します。
CharsetEncode	encoding	バイナリオブジェクトを、指定したエンコードの文字列に変換します。

タグまたは関数	属性またはパラメータ	用途
GetEncoding		Form または URL スコープのテキストの文字エンコードを返します。
SetEncoding	charset	Form または URL スコープのテキストの文字エンコードを指定します。フォーム入力の文字セットまたは URL の文字セットが UTF-8 エンコード形式でない場合に使用されます。
ToBase64	encoding	Base 64 に変換する文字列の文字エンコードを指定します。
ToString	encoding	指定の文字エンコードでエンコードした文字列を返します。
URLDecode	charset	デコードする URL の文字エンコードを指定します。
URLEncodedFormat	charset	URL に使用する文字エンコードを指定します。

ロケールを制御および使用するための関数

ColdFusion には、ロケールを指定および識別し、ロケールに基づいてテキストの形式を設定する次の関数があります。

タグまたは関数	用途
GetLocale	現在のロケール設定を返します。
GetLocaleDisplayName	特定のロケールの言語でロケール名を返します。デフォルト値は、ロケールの言語での現在のロケールです。
LSCurrencyFormat	数値をロケール固有の通貨形式の文字列に変換します。ユーロを使用する国の場合、結果は JVM のバージョンによって異なります。
LSDateFormat	日付時刻値の日付の部分を、ロケール固有の日付形式の文字列に変換します。
LSEuroCurrencyFormat	数値をロケール固有の通貨形式の文字列に変換します。ユーロを通貨として使用する国では、ユーロを使用して形式設定します。
LSIsCurrency	文字列が現在のロケールで有効な通貨表現かどうかを判別します。
LSIsDate	文字列が現在のロケールで有効な日付時刻値表現かどうかを判別します。
LSIsNumeric	文字列が現在のロケールで有効な数値表現かどうかを判別します。
LSNumberFormat	数値をロケール固有の数値形式の文字列に変換します。
LSParseCurrency	現在のロケールの通貨値である文字列を、形式設定された数値に変換します。ユーロを使用する国の場合、結果は JVM のバージョンによって異なります。
LSParseDateTime	現在のロケールの有効な日時表現である文字列を、日付時刻オブジェクトに変換します。
LSParseEuroCurrency	現在のロケールの通貨値である文字列を、形式設定された数値に変換します。ユーロを通貨とする国ではユーロを使用する必要があります。
LSParseNumber	現在のロケールの有効な数値表現である文字列を、形式設定された数値に変換します。
LSTimeFormat	日付時刻値の時刻の部分を、ロケール固有の日付形式の文字列に変換します。
SetLocale	ロケール設定を指定します。

注意：名前が **LS** で始まる多くの関数には、LS という接頭辞の付かない関数も存在します。そのような関数として、DateFormat、IsDate、IsNumeric、NumberFormat、ParseDateTime、および TimeFormat があります。これらの関数は英語 (US) ロケールの規則に従います。

LS 関数を呼び出す前に SetLocale 関数を呼び出さなかった場合は、JVM で定義されているロケールが使用されます。これは通常、オペレーティングシステムのロケールです。

次の例では、LSDateFormat 関数を使用して、ColdFusion でサポートされている各ロケールの形式で現在の日付を表示します。

```
<!--- This example shows LSDateFormat --->
<html>
<head>
<title>LSDateFormat Example</title>
</head>
<body>
<h3>LSDateFormat Example</h3>
<p>Format the date part of a date/time value using the locale convention.
<!--- loop through a list of locales; show date values for Now()--->
<cfloop list = "#Server.Coldfusion.SupportedLocales#"
    index = "locale" delimiters = ",">
    <cfset oldlocale = SetLocale(locale)>

    <cfoutput><p><B><I>#locale#</I></B><br>
        #LSDateFormat(Now(), "mmm-dd-yyyy")#<br>
        #LSDateFormat(Now(), "mmmm d, yyyy")#<br>
        #LSDateFormat(Now(), "mm/dd/yyyy")#<br>
        #LSDateFormat(Now(), "d-mmm-yyyy")#<br>
        #LSDateFormat(Now(), "ddd, mmmm dd, yyyy")#<br>
        #LSDateFormat(Now(), "d/m/yy")#<br>
        #LSDateFormat(Now())#<br>
        <hr noshade>
    </cfoutput>
</cfloop>
</body>
</html>
```

その他のグローバル化タグおよび関数

グローバル化アプリケーションを開発するときは、グローバル化アプリケーション用のタグや関数の他に、次の関数も役立ちます。

- すべての文字列処理関数。詳細については、『CFML リファレンス』の ColdFusion Functions にある文字列関数のリストを参照してください。
- オペレーティングシステムのタイムゾーンを返す GetTimeZoneInfo 関数。

ColdFusion でのデータ処理

アプリケーションのグローバル化に関する問題の多くは、ColdFusion でサポートされている多様なソースからのデータの処理方法に関連するものです。たとえば、次のものが含まれます。

- 文字エンコードの一般的な問題
- ロケール固有のコンテンツ
- URL および HTML フォームからの入力データ
- ファイルデータ
- データベース
- 電子メール
- HTTP
- LDAP
- WDDX
- COM
- CORBA

- 検索とインデックス作成

文字エンコードの一般的な問題

以前のバージョンの ColdFusion 用に開発されたアプリケーションでは、文字の長さがバイト長に等しいという前提で開発されているものがあります。そのようなアプリケーションを ColdFusion で使用するとエラーが発生する場合があります。文字列のバイト長は、文字エンコードによって異なります。

ロケール固有のコンテンツ

マルチロケールコンテンツの生成

複数のロケールのユーザーをサポートするアプリケーションで、ロケールに応じて異なる出力を生成する場合は、リクエストごとに SetLocale 関数を呼び出して、そのリクエストに対するロケールを設定します。処理が完了したら、ロケールを前の値に戻します。有用なテクニックとして、ユーザーが選択したロケールを Session 変数に保存し、セッション中の各ユーザーリクエストに対して、その Session 変数の値を使用してロケールを設定するという方法があります。

ユーロのサポート

ユーロはヨーロッパ諸国の通貨です。ColdFusion は正しく形式設定されたユーロ値の読み取りと書き込みをサポートしています。サポートされている他の通貨と異なり、ユーロはどの特定の国 (ロケール) にも関連付けられていません。LSCurrencyFormat および LSParseCurrency 関数の動作は基盤となる JVM に依存し、通貨の表記規則は JVM によって異なります。Sun JVM の 1.3 リリースではユーロがサポートされておらず、従来の国別の通貨が使用されます。1.4 リリースでは、2002 年現在でユーロ地域に含まれているすべての通貨に対してユーロが使用されます。ユーロがサポートされていない JVM を使用している場合は、LSEuroCurrencyFormat および LSParseEuroCurrency 関数を使用して、通貨にユーロを使用しているロケールでユーロ値を形式設定および解析できます。

URL および HTML フォームからの入力データ

Web アプリケーションサーバーに文字データを渡す方法には、リクエスト URL パラメータから渡す方法と、フォームデータとして渡す方法があります。

HTTP 1.1 標準では、URL とメッセージヘッダに使用できる文字は US-ASCII 文字 (0 ～ 127) のみと規定されています。したがって、ASCII 以外の文字を URL のアドレスおよびパラメータに使用する場合は、それらの文字をエンコードする必要があります。それには、"%xx" という 16 進数形式を使用して文字をエスケープ (URL エンコード) します。ただし、URL エンコードによって、Web ドキュメントでの URL の使用方法が決まるわけではありません。URL のエンコード方法だけが指定されます。

フォームデータではメッセージヘッダを使用して、リクエスト (Content ヘッダ) やレスポンス (Accept ヘッダ) のエンコードを指定します。クライアントとサーバー間のコンテンツネゴシエーションではこの情報が使用されます。

さまざまな文字エンコードで入力された URL およびフォームデータを処理するには、いくつかの方法があります。

URL 文字列の処理

サーバーへの URL リクエストには、名前と値のペアが含まれている場合があります。たとえば、次の URL には名前と値のペアが含まれています。

http://company.com/prod_page.cfm?name=Stephen;ID=7645

前述したとおり、US-ASCII 以外の文字エンコードを使用して入力された URL 文字は、16 進数形式で URL エンコードされます。しかし、Web サーバーでは、URL 文字列の文字はデフォルトでシングルバイト文字であると見なされます。

ASCII 以外の文字を URL で使用するための一般的な方法の 1 つに、URL の文字エンコードを定義する名前と値のペアを URL に含める方法があります。たとえば、次の URL では、**encoding** というパラメータを使用して URL パラメータの文字エンコードを定義しています。

http://company.com/prod_page.cfm?name=Stephen;ID=7645;encoding=Latin-1

"product_name.cfm" ページでは、encoding パラメータの値を確認してから、他の名前と値のペアを処理することができます。これによって、パラメータを正しく処理することができます。

SetEncoding 関数を使用して、URL パラメータの文字エンコードを指定することもできます。SetEncoding 関数は、2 つのパラメータを取ります。最初のパラメータは変数スコープを指定し、2 番目のパラメータはそのスコープで使用する文字エンコードを指定します。URL パラメータは URL スコープに保存されるので、関数のスコープパラメータには "URL" を指定します。

たとえば、Shift-JIS を使用して URL パラメータが渡される場合、そのパラメータには次のようにしてアクセスできます。

```
<cfscript>
    setEncoding("URL", "Shift_JIS");
    writeoutput(URL.name);
    writeoutput(URL.ID);
</cfscript>
```

注意：Shift-JIS 文字エンコードを指定するときは、ハイフン (-) ではなくアンダースコア (_) の Shift_JIS 属性を使用します。

フォームデータの処理

HTML の form タグと ColdFusion の cfform タグを使用すると、ユーザーがページにテキストを入力して、そのテキストをサーバーに送信できます。form タグは、シングルバイト文字データのみを処理するように設計されています。ColdFusion では文字列の保管に 1 文字につき 2 バイトを使用しているので、フォーム入力の各バイトは 2 バイト表現に変換されます。

ダブルバイトテキストをフォームに入力すると、2 バイトで 1 文字であることが認識されず、各バイトが 1 文字として解釈されます。これによって、次の例のように入力テキストが破損します。

1. 顧客が、3 つのダブルバイト文字をフォームに入力します。これは 6 つのバイトで表現されます。
2. この 6 つのバイトが、フォームから ColdFusion に 6 文字として返されます。ColdFusion では、入力された各バイトに 2 バイトが使用され、合計 12 バイトの表現に変換されます。
3. これらの文字を出力すると、破損した情報がユーザーに表示されます。

この問題を回避するには、SetEncoding 関数を使用して、入力フォームテキストの文字エンコードを指定します。SetEncoding 関数は、2 つのパラメータを取ります。最初のパラメータは変数スコープを指定し、2 番目のパラメータはそのスコープで使用する文字エンコードを指定します。フォームパラメータは Form スコープに保存されるので、関数のスコープパラメータには "Form" を指定します。入力テキストがダブルバイトの場合は、テキストの 2 バイト表現が保持されます。

次の例では、フォームデータに韓国語の文字が含まれることを指定します。

```
<cfscript>
    setEncoding("FORM", "EUC-KR");
</cfscript>
<h1> Form Test Result </h1>
<strong>Form Values :</strong>

<cfset text = "String = #form.input1# , Length = #len(Trim(form.input1))#">
<cfoutput>#text#</cfoutput>
```

ファイルデータ

テキストファイルの読み取りと書き込みを行うには、cffile タグを使用します。cffile タグで読み取り、書き込み、コピー、移動、または追加するテキストは、デフォルトでは JVM デフォルトファイル文字エンコード形式であると見なされます。これは通常、システムのデフォルト文字エンコードです。cffile action="Read" では、ファイルの先頭に BOM (Byte Order Mark) があるかどうかも確認されます。BOM がある場合は、その文字エンコードが使用されます。

ファイル文字エンコードが JVM 文字エンコードに一致しない場合は、問題が生じる可能性があります。特に、1 つの文字に使用されるバイト数が両者のエンコードで異なる場合は、問題が発生します。

たとえば、JVM デフォルトファイル文字エンコードが ISO 8859-1 で、ファイルの文字エンコードが Shift-JIS であるとします。ISO 8859-1 では各文字に対して 1 バイトを使用し、Shift-JIS では多くの文字に対して 2 バイト表現を使用します。このファイルを cffile タグで読み込むと、各バイトが ISO 8859-1 文字として処理され、対応する 2 バイト Unicode 表現に変換されます。元の文字は Shift-JIS 形式なのでデータは破損し、1 つの 2 バイト Shift-JIS 文字が 2 つの Unicode 文字に変換されます。

JVM デフォルト文字エンコードと異なるエンコードが使用されているテキストを cffile タグで正しく読み書きするには、charset 属性を渡します。次の例のように、読み書きするデータの文字エンコードを値として指定します。

```
<cffile action="read"
    charset="EUC-KR"
    file = "c:\web\message.txt"
    variable = "Message" >
```

データベース

ColdFusion アプリケーションは、サポートされているデータベースタイプに対応するドライバを使用してデータベースにアクセスします。クライアントのネイティブ言語のデータ型から SQL データ型への変換は、ドライバマネージャ、データベースクライアント、またはサーバーによって透過的に行われます。たとえば、JDBC API で使用する文字データ (SQL CHAR、VARCHAR) は、Unicode エンコード文字列で表現されます。

データベース管理者はデータソースを設定し、通常は、文字を使用する列データの文字エンコードを指定する必要があります。Oracle、Sybase、Informix などの主要ベンダの多くでは、Unicode UTF-8 や UTF-16 などの多くの文字エンコードを使用して文字データを保管できます。

ColdFusion で提供されているデータベースドライバでは、データベースのネイティブ形式から ColdFusion Unicode 形式へのデータ変換が正しく処理されます。データベースにアクセスするために、追加の処理は必要はありません。ただし、データベース管理者に相談してデータベースの文字エンコードの対応状況を確認してください。

電子メール

ColdFusion は、cfmail、cfmailparam、および cfmailpart タグを使用して電子メールメッセージを送信します。

デフォルトでは、メールは UTF-8 エンコードで送信されます。ColdFusion Administrator の [メール] ページを使用すれば、デフォルトのエンコードを変更できます。また、cfmail および cfmailpart タグの charset 属性を使用して、特定のメールメッセージ、または、マルチパート仕様のメールメッセージの特定のパートに対して、文字エンコードを指定できます。

HTTP

ColdFusion では、cfhttp および cfhttpparam タグと GetHttpRequestData 関数を使用して、HTTP 通信が行えます。

cfhttp タグは HTTP リクエストの作成をサポートします。cfhttp タグはデフォルトで Unicode UTF-8 エンコードを使用してデータを渡しますが、charset 属性を使用して文字エンコードを指定することもできます。cfhttpparam タグの mimeType 属性を使用して、ファイルの MIME タイプおよび文字セットを指定することもできます。

LDAP

ColdFusion では、cfldap タグによって LDAP (Lightweight Directory Access Protocol) がサポートされています。LDAP では UTF-8 エンコード形式が使用されるので、取得したすべてのデータを他のデータと組み合わせて、安全に操作できます。LDAP をサポートするために、追加の処理は特に必要ありません。

WDDX

ColdFusion では、cfwddx タグがサポートされています。WDDX (Web Distributed Data Exchange) データは UTF-8 エンコードとして保管されるので、ダブルバイト文字エンコードは自動的にサポートされます。WDDX でダブルバイト文字を処理するために、特別な処理は必要ありません。

COM

ColdFusion では、cfobjecttype="com" タグによって COM がサポートされています。COM インターフェイスで使用されるすべての文字列データは、ダブルバイト文字をサポートするワイド文字 (wchars) を使用して構築されます。COM オブジェクトとやり取りを行うために、特別な処理は必要はありません。

CORBA

ColdFusion では、cfobjecttype="corba" タグによって CORBA がサポートされています。CORBA 2.0 インターフェイス定義言語 (IDL) の基本型 "String" では、フル 8 ビット (256) を使用して文字を表す Latin-1 文字エンコードが使用されていました。

バージョン 2.1 以降の CORBA では、Java 型の char および string に対応する IDL 型 wchar および wstring がサポートされているので、バージョン 2.1 以降の CORBA を使用している限り、ダブルバイト文字をサポートするために特別な処理を行う必要はありません。

ただし、wchar および wstring がサポートされていないバージョンの CORBA を使用している場合、サーバーでは、シングルバイトのテキスト表現を前提とする char および string データ型が使用されます。

検索とインデックス作成

ColdFusion では、cfindex、cfcollection、および cfsearch タグによって Verity 検索がサポートされています。ColdFusion の CD には、多言語検索をサポートするための Verity Language Packs が含まれています。これをインストールすると、さまざまな言語がサポートされます。

アプリケーションのデバッグとトラブルシューティング

Adobe ColdFusion には、アプリケーションの問題を解決するための詳細なデバッグ情報が用意されています。デバッグ情報を提供するように ColdFusion を設定し、cftrace タグと cftimer タグを使用してコード実行の詳細情報を取得できます。さらに、コードを実行する前にツールを使用して検証したり、特定の問題のトラブルシューティングを行うこともできます。

注意：Adobe Dreamweaver には、ColdFusion のデバッグ出力を表示および使用するための統合ツールが用意されています。このツールの使用方法については、Dreamweaver のオンラインヘルプを参照してください。

ColdFusion Administrator によるデバッグの設定

ColdFusion では、ブラウザによってリクエストされるすべてのアプリケーションページに関する重要なデバッグ情報を提供できます。ColdFusion Administrator を使用して、有効にするデバッグ情報およびその表示方法を指定できます。ここでは、Administrator の設定について簡単に説明します。詳細については、[デバッグ] ページのオンラインヘルプを参照してください。

[デバッグの設定] ページ

Administrator の [デバッグ出力の設定] ページの次のオプションを使用して、デバッグ出力に表示する情報を指定できます。

オプション	説明
Robust 例外情報の有効化	例外エラーページに次の情報を表示します。デフォルトはオフです。 • エラーが発生したページのパスおよび URL • エラーが識別されたコードの行番号およびその周辺のコード • SQL ステートメントおよびデータソース • Java スタックトレース
デバッグの有効化	デバッグ出力を有効にします。このオプションをオフにすると、cftrace および cftimer 呼び出しのすべての出力を含め、デバッグ情報は表示されなくなります。デフォルトはオフです。 本番用のサーバーでは、デバッグ出力を無効にしてください。ユーザーにデバッグ情報が表示されなくなるので、セキュリティが向上します。また、サーバーの応答時間も短縮されます。デバッグ出力を特定の IP アドレスに限定することもできます。詳細については、381 ページの「[デバッグする IP アドレス] ページ」を参照してください。
デバッグ出力形式の選択	デバッグ出力を表示する方法を決定します。 • "classic.cfm" テンプレート (デフォルト) を指定すると、情報は HTML 形式のプレーンテキストとしてページの下部に表示されます。 • "dockable.cfm" テンプレートを指定すると、DHTML を使用して、デバッグ情報が別のウィンドウに拡張ツリー形式で表示されます。このウィンドウは、フローティングペインにするか、またはブラウザウィンドウにドッキングすることができます。ドッキング可能な出力形式の詳細については、384 ページの「"dockable.cfm" 出力形式の使用」を参照してください。
実行時間のレポート	HTTP リクエストの結果実行される ColdFusion ページをリストし、実行時間を表示します。処理時間が指定の値を超過したページは赤で強調表示されます。要約表示または詳細なツリー構造表示を選択できます。
一般的なデバッグ情報	リクエストに関する一般情報 (ColdFusion のバージョン、テンプレート、タイムスタンプ、ユーザーロケール、ユーザーエージェント、ユーザー IP、ホスト名など) を表示します。
データベースアクティビティ	SQL データソースおよびストアドプロシージャへのアクセスに関するデバッグ情報を表示します。デフォルトで選択されています。
例外情報	リクエストの処理中に発生したすべての ColdFusion 例外をリストします。デフォルトで選択されています。
トレース情報	各 cftrace タグのエントリを表示します。このオプションをオフにすると、デバッグ出力にはトレース情報が含まれませんが、出力ページには、inline="Yes" と指定されている cftrace タグの情報が含まれます。デフォルトで選択されています。 cftrace タグの使用方法の詳細については、386 ページの「cftrace タグによる実行のトレース」を参照してください。
変数	ColdFusion 変数値の表示を有効にします。このオプションをオフにすると、デバッグ出力ですべての ColdFusion 変数の表示が無効になります。デフォルトで選択されています。 このオプションを有効にすると、選択したスコープの変数値が表示されます。Variables、Attributes、Caller、および ThisTag を除くすべての ColdFusion スコープを選択して表示できます。セキュリティ上の理由から、Application、Server、および Request 変数の表示は、デフォルトで無効になっています。
パフォーマンス監視の有効化	標準の NT パフォーマンスモニタアプリケーションを使用して、実行中の ColdFusion アプリケーションサーバーの情報を表示できるようにします。
CFSTAT の有効化	cfstat コマンドラインユーティリティを使用してリアルタイムでパフォーマンスを監視できるようにします。このユーティリティを使用すれば、NT System Monitor アプリケーションを使用せずに、ColdFusion が NT System Monitor に出力する情報と同じ情報を表示できます。cfstat ユーティリティの詳細については、『ColdFusion 設定と管理』を参照してください。

[デバッグする IP アドレス] ページ

デバッグ出力を有効にすると、デフォルトではローカルユーザーのみが (つまり IP アドレス 127.0.0.1 を経由した場合のみ) 出力を表示できます。デバッグ出力を表示できるユーザーの IP アドレスを追加したり、ローカルユーザーが出力を表示できないようにすることも可能です。デバッグメッセージを受信できるアドレスを指定するには、Administrator の [デバッグする IP アドレス] ページを使用します。

注意：困難な問題の原因解明などのために、本番用のサーバーでデバッグを有効にする必要がある場合は、[デバッグする IP アドレス] ページを使用して開発システムのみに出力を制限し、クライアントにデバッグ情報が表示されないように設定します。

ブラウザページのデバッグ情報の使用

Administrator で設定した ColdFusion デバッグ出力は、HTML リクエストが完了するたびに表示されます。この出力では、リクエスト終了時のサーバーのステータスが示されます。リクエストの処理中にデバッグ情報を表示する方法については、386 ページの「cftrace タグによる実行のトレース」を参照してください。

dockable.cfm のデバッグ出力形式では、デバッグ出力が折り畳まれた形式で表示されます。次のセクションでは、デバッグの各セクションに表示される情報およびその利用方法について説明します。

一般デバッグ情報

デバッグの一般的な情報を表示します。classic.cfm の出力形式では、デバッグ出力の冒頭に情報が表示され、見出しはありません。

一般デバッグ情報の内容は次のとおりです。この表は、"classic.cfm" 出力テンプレート表示で使用される名前の一覧です。

名前	説明
ColdFusion	ColdFusion のバージョン
Template	リクエストされたテンプレート ("dockable.cfm" 形式では [ページ] と呼ばれ、ページ概要のセクションに表示されます)。
TimeStamp	リクエストが完了した時刻 ("dockable.cfm" 形式では [日付] と呼ばれ、ページ概要のセクションに表示されます)。
Locale	情報の処理方法 (特にメッセージの言語) を指定する地域と言語の設定
User Agent	HTTP リクエストを生成したブラウザの ID
Remote IP	HTTP リクエストを生成したクライアントシステムの IP アドレス
Host Name	リクエストを実行した ColdFusion サーバーを実行しているホストの名前

実行時間のレポート

[実行時間のレポート] セクションには、リクエストの処理に要した時間が表示されます。"Application.cfc" ページ、"Application.cfm" ページ、"OnRequestEnd.cfm" ページ、(使用している場合は) CFML カスタムタグ、cfinclude タグでインクルードされるページ、ColdFusion コンポーネント (CFC) ページなど、リクエストに必要なすべてのページを処理するための所要時間に関する情報が表示されます。


特定のコードブロックの実行時間を表示するには、cftimer タグを使用します。

実行時間は次の 2 つの形式で表示できます。

- 要約
- ツリー

注意：ページの作成や変更を行った場合は、そのページを初めて実行するときよりも、2 回目に実行するときのほうが、実行時間が大幅に短くなります。ページを初めて使用するときは、ColdFusion によってそのページが Java バイトコードにコンパイルされ、サーバーに保存されてメモリにロードされます。以降、そのページを変更せずに使用するときは、コードを再コンパイルする必要がないので時間が短縮されます。

要約実行時間形式

要約形式では、リクエスト時に処理された ColdFusion ページごとに 1 つのエントリが表示されます。ページが複数回にわたって処理された場合でも、要約には 1 回だけ表示されます。たとえば、あるカスタムタグが 1 つのリクエストで 3 回呼び出された場合、出力には 1 回だけ表示されます。

次の表に、表示フィールドを示します。

列	説明
Total Time	ページのすべてのインスタンス、および使用されるすべてのページの処理に要した合計時間。たとえば、リクエストによってページが 2 回処理され、そのページに他のページが含まれている場合、合計時間には両方のページを 2 回処理するために必要な時間が含まれます。
Avg Time	このページの各インスタンス、および使用されているページを処理するための平均時間。[Avg Time] に [Count] を乗算すると [Total Time] になります。
Count	リクエストに対してページが処理された回数
Template	ページのパス名

ページアイコンはリクエストされたページを示します。

平均処理時間が ColdFusion Administrator の [デバッグの設定] ページで設定された強調表示値を超えるページは、すべて赤で表示されます。

最後から 2 番目の行には、ページの解析、コンパイル、ロード、開始処理、および終了処理に要した時間が表示されます。この値は、各ページの実行時間には含まれません。最後の行には、リクエストの処理に要した合計時間が表示されます。

ツリー実行時間形式

ツリー実行時間形式は、ColdFusion における各ページの処理過程を示す階層型の詳細な表示です。あるページで 2 番目のページをインクルードまたは呼び出している場合、2 番目のページは最初のページの下にインデントされて表示されます。各ページは、使用されるたびに 1 回表示されます。したがって、あるページがリクエスト処理時に 3 回呼び出された場合は、ツリーにも 3 回表示されます。ツリー表示を使用すれば、処理時間とページの処理順序の両方が確認できます。

要約表示と同様に、実行時間 (括弧内) は、リストされているページや、そのページの処理に必要なすべてのページ (ツリー内のインデントされているすべてのページ) の処理時間を示します。

この図の出力から、次の情報がわかります。

- "Application.cfm" ページをリクエストの一部として処理するための所要時間は 0 ミリ秒でした。
- リクエストされたページは "tryinclude.cfm" でした。このページと、このページの実行に必要なすべてのページを処理するための所要時間は、203 ミリ秒でした。このページに直接記述されているコードを処理するための所要時間は 71 ミリ秒 (203 - 93 - 16 - 23) でした。
- "mytag2.cfm" ページは 3 回処理されました。すべての処理の合計所要時間は 93 ミリ秒で、平均処理時間は 31 ミリ秒でした。このページは他のページを呼び出していません。
- "mytag1.cfm" ページは 2 回処理されました。すべての処理の合計所要時間は 78 ミリ秒で、平均処理時間は 39 ミリ秒でした。この時間には "mytag2.cfm" (このタグは mytag2 カスタムタグを呼び出します) の処理時間も含まれています。したがって、このページに直接記述されているコードの平均所要時間は 8 ミリ秒で、合計処理時間は 16 ミリ秒でした。

- "includeme.cfm" ページを処理するための所要時間は約 62 ミリ秒でした。この処理時間には "mytag1.cfm" の処理時間が含まれていることから、"mytag2.cfm" を 1 回処理する時間も加算されています。したがって、このページに直接記述されているコードの所要時間は 23 ミリ秒 (62 - 39) でした。
- 特定のページに関連付けられていない処理の所要時間は、125 ミリ秒でした。
- 総処理時間は、125 + 203 の合計の 328 ミリ秒でした。

データベースアクティビティ

Administrator の [デバッグの設定] ページで [データベースアクティビティ] を選択すると、データベースアクセスに関する情報がデバッグ出力に含まれます。

SQL クエリー

SQL クエリーのセクションには、SQL クエリーを生成したタグや、キャッシュされたデータベースクエリーを取得するタグ (cfquery、cfinsert、cfgridupdate、cfupdate) に関する情報が示されます。

次の情報が表示されます。

- クエリーが存在するページ
- クエリーが生成された時刻
- クエリー名
- キャッシュされたクエリーの結果かどうか
- SQL ステートメント。CFML 変数や cfqueryparam タグなどのダイナミックな要素の処理結果を含みます。この情報は、SQL ステートメントの ColdFusion での処理結果がすべて含まれるので役立ちます。
- データソース名
- 返されたレコードの数。0 はクエリーに一致がないことを示します。
- クエリー実行時間
- cfqueryparam タグから渡されたクエリーパラメータ値

ストアドプロシージャ

[ストアドプロシージャ] セクションには、cfstoredproc タグを使用してデータベース管理システムのストアドプロシージャを実行した結果に関する情報が表示されます。

次の情報が表示されます。

- ストアドプロシージャ名
- データソース名
- クエリー実行時間
- クエリーが存在するページ
- クエリーが生成された時刻
- cfprocparam タグで指定したプロシージャパラメータの送受信結果を示すテーブル。ctype、CFSQLType、value、variable、および dbVarName 属性が含まれます。OUT および INOUT パラメータの variable 情報には、戻り値が含まれます。
- cfprocresult タグで指定して返されたプロシージャ結果セットがリストされているテーブル

例外

Administrator の [デバッグの設定] ページで [例外情報] を選択すると、アプリケーションページの処理中に発生したすべての ColdFusion 例外のリストがデバッグ出力に含まれます。

この例外情報には、アプリケーションコードまたは ColdFusion で検出され処理されたすべてのアプリケーション例外に関する情報が含まれます。

例外は、アプリケーションの通常のフローを妨げるイベントです。275 ページの「エラー処理」で説明したように、予測可能な例外はアプリケーションで検出して、可能であれば回復することが望まれますが、アプリケーションをデバッグしている場合は、例外が検出されたことが警告されるようにしておくと便利です。たとえば、ファイルが見つからない場合は、cffile 例外を検出してバックアップファイルやデフォルトファイルを使用することができますが、デバッグ出力で例外情報を有効にしておくと、このような状況が発生したときにすぐにわかります。

トレースポイント

Administrator の [デバッグの設定] ページで [トレース情報] を選択すると、インラインで結果を表示するタグを含む、すべての cftrace タグの結果がデバッグ出力に含まれます。したがって、リクエスト処理時に検出されたすべてのトレースポイントの履歴がデバッグ出力に含まれます。

cftrace タグの使用方法の詳細については、386 ページの「cftrace タグによる実行のトレース」を参照してください。

スコープ変数

Administrator の [デバッグの設定] ページで [変数] オプションと変数スコープを選択すると、選択したスコープのすべての変数の値がデバッグ出力に表示されます。デバッグ出力に表示される値は、現在のページの処理がすべて完了した後の値です。

選択したスコープ変数を表示することで、アプリケーション変数などの永続スコープ変数に対する処理の影響を判断できます。この方法は、例外の発生しない問題を調査するときに役立ちます。

Form、URL、および CGI スコープは、リクエストのステートを調査するときに役立ちます。これらのスコープを表示すると、ページの動作に影響を与える次のようなパラメータを調査できます。

URL 変数 HTTP リクエストパラメータを識別します。

フォーム変数 アクションページに送信されるフォームフィールドを識別します。

CGI 変数 リクエストに従ってサーバー環境を表示します。

同様に、Client、Session、Application、および Server スコープ変数はアプリケーションのグローバルステートを示すので、各ページが ColdFusion の永続変数のステートに与える影響をトレースするときに役立ちます。

"dockable.cfm" 出力形式の使用

dockable.cfm の出力形式には、classic.cfm のデバッグ表示に含まれない機能がいくつかあります。

アプリケーションページの選択

各ページの下部には、次の表に示す 2 つのボタンが表示されます。

ボタン	説明
Debug This page	選択したフレームのデバッグ情報を表示するように指示します。現在のフレームとして選択した場合、またはアプリケーションがフレームを使用していない場合、デバッグペインが更新されます。
Floating/Docked debug pane	ペインを、指定されたフレームの左側に固定するかどうかを切り替えます。

デバッグペインの機能

デバッグペインには次の機能があります。

- 例外などのデバッグ情報カテゴリは、カテゴリ見出しの前のプラスまたはマイナス記号 (+ または -) をクリックして、展開したり閉じたりすることができます。[変数] セクションに表示される各スコープのデータ型も、展開したり閉じたりすることができます。

- デバッグペインの上部には、cgi.script_name 変数によって識別された、デバッグ対象のアプリケーションページの URL が表示されます。このリンクをクリックすると、ページが更新され、生成されたデバッグ情報が表示されます。ブラウザの更新ボタンまたはキーを使用して、ページやデバッグ情報を更新することもできます。
- デバッグペインには、ページのパスまたは URL を入力できるボックスも表示されます。[Go] ボタンをクリックすると、ColdFusion によってそのページが処理され、デバッグペインが更新されて新しいページのデバッグ情報が表示されます。

CFML でのデバッグ情報の制御

次のセクションでは、CFML タグおよび関数を使用してデバッグ情報およびトレース情報を表示または非表示にする方法について説明します。

個別クエリーのデバッグ情報の生成

cfquery タグの debug 属性は、Administrator の [デバッグの設定] ページの [データベースアクティビティ] 設定よりも優先されます。debug 属性が機能するのは、[デバッグの設定] ページでデバッグ出力を有効にしている場合のみです。

- ColdFusion Administrator で [データベースアクティビティ] を選択しても、debug="No" と指定すれば、そのクエリーの SQL および統計はデバッグ出力に表示されません。
- ColdFusion Administrator で [データベースアクティビティ] を選択していなくても、debug="Yes" または debug と指定すれば、そのクエリーの SQL および統計がデバッグ出力に表示されます。

たとえば、ColdFusion Administrator で [データベースアクティビティ] を選択していなくても、次のコードを使用すると、このクエリーについてのみ、クエリー実行時間、返されたレコード数、ColdFusion ページ、タイムスタンプ、およびデータソースに送信された SQL ステートメントを表示できます。

```
<cfquery name="TestQuery" datasource="cfdocexamples" debug>
    SELECT * FROM TestTable
</cfquery>
```

たとえば、頻繁に呼び出されるカスタムタグでクエリーを使用している場合は、debug 属性を使用してクエリーデバッグ情報の生成を無効にすれば、呼び出しページのクエリーデバッグ情報のみが表示されます。

また、cfstoredproc タグで debug 属性を指定すれば、ストアドプロシージャ固有のデバッグ情報を表示できます。

cfsetting タグによるデバッグ出力の制御

cfsetting タグの showDebugOutput 属性を使用すると、特定のページのデバッグ出力をオフにできます。この属性でデバッグ出力を制御できるのは、ColdFusion Administrator の [デバッグの設定] ページでデバッグ出力を有効にしている場合のみです。この属性のデフォルト値は Yes です。次のタグを使用すると、現在のページのすべてのデバッグ出力が表示されなくなります。

```
<cfsetting showDebugOutput="No">
```

このタグを "Application.cfc" ファイルの初期化コードか "Application.cfm" ページに追加すると、アプリケーションのすべてのデバッグ出力が表示されなくなります。また、特定のページの cfsetting タグで showDebugOutput="Yes" と指定すれば、そのページについてデバッグ出力の設定を上書きできます。逆に、アプリケーションのデバッグをオンのままにし、特定のページで cfsetting showDebugOutput="No" タグを使用すれば、デバッグ出力によってエラーや混乱が発生する可能性のあるページのデバッグを無効にできます。

さらに、ColdFusion Administrator へのアクセス権がなく、ColdFusion Administrator でデバッグが有効にされている場合は、showDebugOutput 属性を使用してデバッグ出力を制御できます。

IsDebugMode 関数によるコードの選択的実行

IsDebugMode 関数は、デバッグが有効な場合に True を返します。この関数を cfif タグの条件で使用すれば、デバッグ出力が有効な場合にのみコードを実行することができます。IsDebugMode 関数を使用すると、デバッグモードで任意のコードを実行できるので、cftrace タグよりも柔軟に情報を処理および表示できます。

IsDebugMode 関数を使用すると、デバッグが有効な場合のみ、情報を選択的にログに記録できます。ログ出力を制御できるので、トレース情報をブラウザに表示することなくログに記録することができます。たとえば、デバッグが有効な場合に次のコードを実行すると、アプリケーションページ、現在時刻、および 2 つの変数の値が、ログファイル "MyAppSilentTrace.log" に記録されます。

```
<cfquery name="MyDBQuery" datasource="cfdocexamples">
    SELECT *
    FROM Employee
</cfquery>
<cfif IsDebugMode()>
    <cflog file="MyAppSilentTrace" text="Page: #cgi.script_name#,
    completed query MyDBQuery; Query Execution time:
    #cfquery.ExecutionTime# Status: #Application.status#">
</cfif>
```

頻繁に cfdump タグでデバッグする場合は、このタグを <cfif IsDebugMode()> タグで囲み、<cfif IsDebugMode()><cfdump var=#myVar#></cfif> のようにします。このようにすると、本番用コードに cfdump タグが残っていても、デバッグ出力が無効なときには表示されなくなります。

cftrace タグによる実行のトレース

cftrace タグを使用すると、cftrace タグの実行時点でのアプリケーションのステートに関するデバッグデータが表示され、ログに記録されます。このタグは、アプリケーションの実行時に特定の情報の " スナップショット " を取得するために使用します。

cftrace タグについて

cftrace タグは、次の情報を提供します。

- cftrace タグの type 属性によって指定される厳格度識別子
- cftrace タグが実行された時刻を示すタイムスタンプ
- リクエストの処理開始から今回の cftrace タグの実行までに経過した時間
- リクエストの前回の cftrace タグと今回のこのタグの間の経過時間。今回がリクエストで処理される最初の cftrace タグである場合は、出力に "1st trace" と示されます。この情報は、インライントレース出力には表示されず、ログと標準のデバッグ出力にのみ表示されます。
- cftrace タグを呼び出したページの名前
- cftrace 呼び出しが配置されているページ上の行
- category 属性によって指定されたトレースカテゴリ
- text 属性によって指定されたメッセージ
- cftrace 呼び出しで var 属性に指定された 1 つの変数の名前とその値

一般的な cftrace タグの例を次に示します。

```
<cftrace category="UDF End" inline = "True" var = "MyStatus"
    text = "GetRecords UDF call has completed">
```

cftrace タグ出力は、次のいずれかまたは両方の方法で表示できます。

- **デバッグ出力のセクションとして表示** : デバッグ出力にトレース情報を表示するには、Administrator の [デバッグの設定] ページで [トレース情報] を選択します。
- **アプリケーションページにインラインで表示** : cftrace タグに inline 属性を指定すると、トレース出力がページの cftrace タグの位置に表示されます (インラインの cftrace タグが cfsilent タグブロックの中にある場合は、出力が表示されません)。

cftrace タグは、ColdFusion Administrator の [デバッグの設定] ページで [デバッグの有効化] を選択した場合のみ実行されます。トレースの結果をデバッグ出力に表示するには、さらに [デバッグの設定] ページで [トレース情報] を選択する必要があります。選択しなかった場合、トレース情報はログとインライントレースには表示されますが、デバッグ出力には表示されません。

注意 : インラインのトレースタグが使用されているページは、すべてのページ処理が完了した後、デバッグテンプレートからデバッグ出力が表示される前にブラウザに送信されます。したがって、トレースタグからページが終了するまでの間にエラーが発生した場合は、そのタグのトレースが表示されない可能性があります。

次の図に、インライントレースメッセージの例を示します。

次の表に、表示される情報の一覧を示します。

エントリ	説明
	cftrace の呼び出しで指定されたトレースタイプ (厳格度)。この場合は Information です。
[CFTRACE 13:21:11.011]	cftrace タグが実行された時刻
[501 ms]	現在のリクエストを処理するために cftrace タグまでに要した時間
[C:¥CFusion¥wwwroot¥MYStuff¥mydocs¥tractest.cfm]	cftrace タグが含まれているページの Web サーバーでのパス
@ line:14	cftrace タグがある行番号
[UDF End]	cftrace タグの category 属性の値
GetRecords UDF 呼び出しは完了しました。	cftrace タグの text 属性。変数はすべてその値に置き換えられます。
MyStatus Success	cftrace タグの var 属性で指定された変数の名前とその値

すべての cftrace 出力は、ColdFusion インストールディレクトリの "logs¥cftrace.log" ファイルに記録されます。

ログファイルのエントリは、次のようになります。

```
"Information","web-29","04/01/02","13:21:11","MyApp","[501 ms (1st trace)]
[C:\ColdFusion9\wwwroot\MYStuff\mydocs\tractest.cfm @ line: 14] - [UDF End] [MyStatus = Success] GetRecords
UDF call has completed "
```

このエントリは、フィールドがカンマで区切られ、二重引用符 (") で囲まれた、標準の ColdFusion ログ形式です。トレース出力で表示される情報は、最後のメッセージフィールドに示されます。

次の表に、トレースメッセージとログエントリの内容を示します。ログファイル形式の詳細については、285 ページの「cflog タグによるエラーのロギング」を参照してください。

エントリ	説明
Information	cftrace 呼び出しで指定された厳格度
web-29	コードを実行したサーバースレッド
04/01/02	トレースがロギングされた日付
13:21:11	トレースがロギングされた時刻
MyApp	cfapplication タグで指定されているアプリケーション名
[501 ms (1st trace)]	現在のリクエストを処理するために cftrace タグまでに要した時間。これは、このリクエストで処理された最初の cftrace タグです。以前に処理された cftrace タグがある場合は、前回の cftrace タグと今回のタグの間隔 (ミリ秒単位) が括弧内に示されます。
[C:¥CFusion¥wwwroot¥MYStuff¥mydocs¥tracetest.cfm @ line: 14]	トレースタグが配置されているページのパスと、そのページの cftrace タグの行番号
[UDF End]	cftrace タグの category 属性の値
[MyStatus = Success]	cftrace タグの var 属性で指定された変数の名前とその値変数が配列や構造体などの複合データ型である場合、ログには変数値と、変数のトップレベルのエントリの数 (構造体のトップレベルのキーの数など) が記録されます。
GetRecords UDF 呼び出しは完了しました。	cftrace タグの text 属性。変数はすべてその値に置き換えられます。

トレースの使用

cftrace タグは、その名前のとおり、アプリケーションの実行をトレースするためのタグです。これは、次のような目的で利用できます。

- タグまたはコードセクションの実行時間の測定。この機能は、処理に時間のかかるタグや処理で有効です。一般的な使用例として、cfquery、cfldap、cfftp、cffile など、外部リソースにアクセスするすべての ColdFusion タグが含まれます。タグまたはコードブロックの実行時間を測定するには、測定するコードの前および後で cftrace タグを呼び出します。
- データ構造体などの、内部変数の値の表示。たとえば、データベースクエリーの結果をそのまま表示できます。
- 変数の中間値の表示。たとえば、このタグを使用すると、文字列関数を使用して部分文字列を選択したり形式を設定したりする前に、文字列値の内容をそのまま表示できます。
- 処理の進行状況の表示とロギング。たとえば、アプリケーションのページの最初や、重要なタグまたは重要な関数の呼び出しの前に cftrace 呼び出しを指定できます。複雑なアプリケーションでは大量のログファイルが生成される場合があるので、この方法は注意して使用してください。
- ページに多数の cfif タグと cfelseif タグがネストされている場合は、各条件ブロックに cftrace タグを指定すると、実行フローをトレースできます。この場合、条件変数をメッセージまたは var 属性で使用します。
- ColdFusion サーバーがハングした場合に、特定のコードブロック (または cfx タグ、COM オブジェクト、その他のサードパーティコンポーネントの呼び出し) が原因と思われる場合は、原因と思われるコードの前後に cftrace タグを追加すると、その箇所の開始と終了をログに記録できます。

cftrace タグの呼び出し

cftrace タグは次の属性を取ります。すべての属性はオプションです。

属性	用途
abort	ブール値です。True を指定すると、このタグの直後で現在のリクエストの処理を中止します。この属性は、cftrace タグの直後に cfabort タグを指定するのと同じです。デフォルト値は False です。この属性が True の場合、cftrace 呼び出しの出力は "cftrace.log" ファイルのみに表示されます。ファイル内の行には、"[ABORTED]" というテキストが含まれます。
category	ユーザー定義のトレースタイプカテゴリを指定するテキスト文字列。この属性を使用すると、複数のトレース行をカテゴリ別に識別または処理できます。たとえば、ログのエントリをカテゴリ別にソートすることができます。 category 属性は、多目的なトレースポイントを識別するように設計されています。たとえば、" カスタムタグ終了 " カテゴリを使用して、カスタムタグが呼び出しページに処理を返すポイントを識別できます。また、特定のカスタムタグ名をカテゴリで識別するなどの方法で、詳細なカテゴリを使用することもできます。 カテゴリテキストには、ColdFusion の単純変数をシャープ記号 (#) で囲んで含めることができますが、配列、構造体、またはオブジェクトを含めることはできません。
inline	ブール値です。True を指定すると、トレース出力をページにインラインで表示します。デフォルト値は False です。 inline 属性を使用すると、cftrace 呼び出しが処理される場所にトレース結果を表示できます。これにより、ビジュアルキューが ColdFusion ページに直接表示されます。 また、デバッグ情報のセクションにもトレース出力が表示されます。
text	このトレースポイントについて説明するテキストメッセージ。テキスト出力には、ColdFusion の単純変数をシャープ記号 (#) で囲んで含めることができますが、配列、構造体、またはオブジェクトを含めることはできません。
type	ColdFusion ロギング厳格度タイプ。インライントレース表示および "dockable.cfm" 出力形式では、各タイプのシンボルが表示されます。デフォルトのデバッグ出力ではタイプ名が表示され、ログファイルでも使用されます。タイプ名は、次のいずれかである必要があります。 Information (デフォルト) Warning Error Fatal Information
var	表示する単一の変数の名前。この属性では、文字列などの単純変数や、構造体名などの複合変数を指定できます。変数名は、シャープ記号で囲まないでください。 複合変数は、インライン出力では cfdump 形式で表示されます。デバッグ表示およびログファイルでは、値ではなく複合変数の要素の数がレポートされます。 この属性を使用すると、ページには通常表示されない内部変数や、ページ処理が進行中の変数の中間値を表示できます。 関数の戻り値を表示するには、メッセージ内に関数を指定します。var 属性は関数を評価できないので、属性内で関数を使用しないでください。

注意：インライントレース出力を指定し、cftrace タグを cfsilent タグブロック内に入れた場合、トレース情報はインラインで表示されませんが、標準のデバッグ出力には表示されます。

次の cftrace タグは、386 ページの「cftrace タグについて」の出力例とログエントリに示されている情報を表示します。

```
<cftrace abort="False" category="UDF End" inline = "True" text = "GetRecords UDF
    call has completed" var = "MyStatus">
```

cftimer タグによるコードブロックの実行時間の測定

cftimer タグを使用すると、CFML コード内の指定したセクションの実行時間を表示できます。

時間表示の使用

このタグを使用して、コードブロックの実行時間を確認することができます。これは、ColdFusion デバッグ出力に過剰な実行時間が示されているが、長時間実行しているコードブロックを特定できない場合に有効です。

このタグを使用するには、ColdFusion Administrator の [デバッグの設定] ページでデバッグを有効にします。また、[デバッグの設定] ページで [タイマー情報] チェックボックスをオンにして、cftimer タグを明示的に使用可能にする必要があります。

cftimer タグのみのデバッグを有効にし、**HTML** コメントで時間情報を表示すれば、本番環境のユーザーを邪魔することなく時間情報を生成することができます。

cftimer タグの呼び出し

cftimer タグによる時間情報の表示場所は、次の中から選択できます。

- **インライン** : </cftimer> タグの後に時間情報を表示します。
- **アウトライン** : 測定するコードの冒頭に時間情報を表示し、測定するコードの周囲にボックスを描画します。これを実行するには、ブラウザが HTML FIELDSET 属性をサポートしている必要があります。
- **コメント** : **<!--label**: **elapsed-timems -->** の形式で HTML コメント内に時間情報を表示します。デフォルトのラベルは cftimer です。
- **デバッグ** : デバッグ出力の CFTimer Times という見出しの下に時間情報を表示します。

次の例では、cftimer タグを複数回呼び出しています。各呼び出しでは、異なる type 属性を使用しています。

```
 <HTML>
<body>
<h1>CFTIMER test</h1>
<!--- type="inline" --->
    <cftimer label="Query and Loop Time Inline" type="inline">
        <cfquery name="empquery" datasource="cfdocexamples">
                select *
                from Employees
        </cfquery>

        <cfloop query="empquery">
            <cfoutput>#lastname#, #firstname#</cfoutput><br>
        </cfloop>
    </cftimer>
<hr><br>
<!--- type="outline" --->
    <cftimer label="Query and CFOUTPUT Time with Outline" type="outline">
        <cfquery name="coursequery" datasource="cfdocexamples">
                select *
                from CourseList
        </cfquery>

    <table border="1" width="100%">
        <cfoutput query="coursequery">
        <tr>
        <td>#Course_ID#</td>
        <td>#CorName#</td>
        <td>#CorLevel#</td>
        </tr>
        </cfoutput>
        </table>
    </cftimer>
<hr><br>
<!--- type="comment" --->
    <cftimer label="Query and CFOUTPUT Time in Comment" type="comment">
        <cfquery name="parkquery" datasource="cfdocexamples">
                select *
                from Parks
        </cfquery>
```

```
<p>Select View &gt; Source to see timing information</p>
    <table border="1" width="100%">
        <cfoutput query="parkquery">
        <tr>
        <td>#Parkname#</td>
        </tr>
        </cfoutput>
        </table>
    </cftimer>

<hr><br>
<!--- type="debug" --->
    <cftimer label="Query and CFOUTPUT Time in Debug Output" type="debug">
        <cfquery name="deptquery" datasource="cfdocexamples">
                select *
                from Departments
        </cfquery>
<p>Scroll down to CFTimer Times heading to see timing information</p>
    <table border="1" width="100%">
        <cfoutput query="deptquery">
        <tr>
        <td>#Dept_ID#</td>
        <td>#Dept_Name#</td>
        </tr>
        </cfoutput>
        </table>
    </cftimer>
</body>
```

コードアナライザの使用

コードアナライザには、次の 2 つの用途があります。

- アプリケーションの CFML シンタックスを検証できます。アナライザは、ページで ColdFusion コンパイラを実行しますが、コンパイルしたコードは実行しません。コンパイラが検出するエラーをレポートします。
- 非互換性とその重大度に関する情報が提供され、必要に応じて修正方法が提示されます。
- ColdFusion の動作が以前のバージョンと異なる部分を識別できます。アナライザは、次のような種類の機能を識別します。
    - **サポートされていない機能** : 使用するとエラーが発生します。たとえば、closable 属性は、ボーダーレイアウト (type="border" の cflayout) 内の cflayoutarea タグではサポートされていません。
    - **非推奨の機能** : 現在は使用できますが、使用はお勧めしません。今後のリリースでは使用できなくなる可能性があります。非推奨の機能の現在の動作が、以前のリリースと異なる場合もあります。たとえば、cfcache タグでは、directory、cachedirectory、port、および protocol 属性は非推奨になっています。
    - **変更された動作** : 以前のバージョンと動作が異なる場合があります。たとえば、ColdFusion 9 で cfcache タグを終了タグ (</cfcache>) なしで使用した場合、現在のページのみがキャッシュされるのではなく (旧リリースの動作)、リクエスト全体がキャッシュされます。
    - **新機能 :** ColdFusion 9 に新しく追加された機能です。たとえば、CFM で throw をユーザー定義関数として使用した場合、throw がビルトインの ColdFusion 関数であることがアナライザによって通知され、名前の変更が推奨されます。CFC で throw をユーザー定義関数として使用した場合、throw がビルトイン関数であることがアナライザによって通知され、先頭にオブジェクトスコープを付加するよう推奨されます。新機能の詳細については、What's New in ColdFusion 9 を参照してください。

コードアナライザは ColdFusion Administrator から実行できます。[デバッグとロギング] ページのリストから [コードアナライザ] を選択します。

注意：コードアナライザでは、検証対象ページは実行されません。したがって、実行時に属性値がダイナミックに指定されたときに無効になる属性の組み合わせは検出できません。

一般的な問題のトラブルシューティング

ここでは、発生する可能性のある一般的な問題と、その解決方法について説明します。

ColdFusion のトラブルシューティングの詳細については、www.adobe.com/go/learn_cfu_troubleshoot_jp の「ColdFusion Support Center Testing and Troubleshooting」ページを参照してください。技術的な問題を防ぎアプリケーションのパフォーマンスを向上する一般的なチューニングや予防策については、テクニカルノート 13810 の ColdFusion の技術的ヒントに関する記事を参照してください。この記事へのリンクは、「Testing and Troubleshooting」ページの先頭近くに表示されます。

CFML シンタックスエラー

問題：次のようなエラーメッセージが表示される。

```
Encountered "function or tag name" at line 12, column 1.
Encountered "\"" at line 37, column 20.
Encountered "," at line 24, column 61.
Unable to scan the character '\"' which follows "" at line 38, column 53.
```

これらのエラーは通常、対応しない <、"、または # 文字があることを示します。このエラーの代表的な原因は、コードを囲む引用符、変数名を囲むシャープ記号、またはタグの閉じ忘れです。識別された行とその前の行のコードに、不足している文字がないかどうかを確認してください。

エラーメッセージに示される行番号は、エラーの原因になった行**ではない**場合があります。エラーメッセージの行番号は、エラーが発生した結果、ColdFusion コンパイラで処理できないコードが検出された最初の行を表します。

問題：理解できないエラーメッセージが表示される。

各 CFML タグに、対応する終了タグがあることを確認してください (終了タグが必要なタグである場合)。代表的なミスとして、cfquery、cfoutput、cftable、または cfif タグの終了タグを忘れることがあります。

前の問題と同様に、エラーメッセージに示される行番号は、エラーの原因になった行**ではない**場合があります。これは、エラーが発生した結果、ColdFusion コンパイラで処理できないコードが検出された最初の行を表します。エラーメッセージにレポートされている行に問題がないように見える場合は、その前のコードを調べて、不足しているテキストがないかどうか確認してください。

問題：属性または値が無効である。

無効な属性または属性値を使用すると、エラーメッセージが返されます。このようなシンタックスエラーは、CFML コードアナライザを使用して防げます。386 ページの「cftrace タグによる実行のトレース」も参照してください。

問題：構造体、配列、クエリーオブジェクト、WDDX エンコード変数などの複合データ変数の構造または内容に問題があると思われる。

cfdump タグを使用して、変数の構造や内容をテーブル形式で表示します。たとえば、relatives という名前の構造体をダンプするには、次の行を使用します。変数名はシャープ記号 (#) で囲みます。

```
<cfdump var=#relatives#>
```

データソースのアクセスおよびクエリー

問題：データベースに接続できない。

接続する前にデータソースを作成します。接続エラーの原因には、ファイルの保存場所、ネットワーク接続、データベースクライアントライブラリの設定などの問題が考えられます。

アプリケーションのソースファイルでデータソースを参照する前に、データソースを作成してください。まず、ColdFusion Administrator の [データソース] ページにある [確認] ボタンをクリックすると、データベースとの接続を検証できます。そのページから基本的な接続を行うことができない場合は、データベース管理者に相談して問題を解決してください。

また、データソース名のスペルを確認してください。

問題 : クエリーの実行に時間がかかりすぎる。

デバッグ出力の [クエリー] セクションからデータベースのクエリー解析ツールにクエリーをコピー & ペーストします。次に、データベースサーバーのクエリーオプティマイザによって生成された実行プランを取得し、解析します。この方法は、データベース管理システムによって異なります。クエリーに時間がかかる代表的な原因は、データの取得を最適化するインデックスがないことです。通常、可能であればテーブルスキャン (または " クラスタされたインデックス " のスキャン) を実行しないでください。

HTTP/URL

問題 : ColdFusion がフォーム入力内容を正しくデコードできない。

ColdFusion サーバーに送信するフォームの method 属性は Post である必要があります。次に例を示します。

```
<form action="test.cfm" method="Post">
```

問題 : URL パラメータにスペースを含めると、ブラウザに警告が表示されるか、または完全な URL 文字列が送信されない。

ブラウザによっては、URL パラメータ中のスペースを %20 というエスケープシーケンスに自動的に置き換えるものもありますが、エラーを表示したり、最初の文字までの URL 文字列だけを送信するブラウザもあります。Netscape 4.7 などは後者のブラウザです。

URL 文字列にスペースを含めることはできません。スペースを含める必要がある場合は、プラス記号 (+) または標準の HTTP スペース文字用エスケープシーケンス (%20) を使用してください。これらの要素は ColdFusion でスペースに正しく変換されます。

このエラーが発生する代表的なシナリオとしては、スペースが含まれているデータベーステキストフィールドからダイナミックに URL を生成する場合があります。この問題を回避するには、URL のダイナミックに生成される部分には数値のみを含めてください。

または、URLEncodedFormat 関数を使用して、自動的にスペースを %20 エスケープシーケンスに置換することもできます。URLEncodedFormat 関数の詳細については、『CFML リファレンス』を参照してください。

ColdFusion デバッガの使用

Adobe ColdFusion では、個々のページのデバッグ情報が提供されますが、ただし、複雑な開発作業には、堅牢でインタラクティブなデバッガが必要です。ColdFusion には、Eclipse または Adobe Flash Builder で ColdFusion アプリケーションを開発するときに使用できるラインデバッガが用意されています。ラインデバッガでは、ブレークポイントの設定、コードのステップオーバー、ステップイン、ステップアウト、変数の検査を行えます。ColdFusion のログファイルを表示することもできます。

ColdFusion デバッガについて

ColdFusion デバッガは Eclipse プラグインであり、Eclipse のデバッグパースペクティブで動作します。ColdFusion デバッガを使用すると、次のようなデバッグタスクを実行できます。

- ブレークポイントの設定
- 変数の表示

- 関数呼び出しのステップオーバー、ステップイン、ステップアウト

ColdFusion デバッガのインストールとアンインストール

ColdFusion デバッガを使用するには、次のソフトウェアをインストールしておく必要があります。

- Eclipse バージョン 3.1.2、Eclipse バージョン 3.2、Flex Builder 2、または Flash Builder
- ColdFusion 9

ColdFusion デバッガをインストールするには、ColdFusion Eclipse プラグインをインストールします。詳細については、『ColdFusion インストール』を参照してください。

デバッガーを使用するための ColdFusion の設定

デバッガを使用するには、ColdFusion Administrator でデバッグを有効にします。

1. ColdFusion Administrator で、[デバッグとロギング]-[デバッガの設定] を選択します。
2. [ラインデバッグの許可] オプションを有効にします。
3. デバッグに使用するポートを指定します (表示されるデフォルトとは異なる場合)。
4. 同時デバッグセッションの最大数を指定します (デフォルトとは異なる場合)。
5. 「変更の送信」をクリックします。
6. 次の操作を行うことで、リクエストがタイムアウトになる時間を増やす必要がある場合があります。
   a. サーバーの設定／設定を選択します。
   b. [リクエストタイムアウト] オプションを有効にします。
   c. 300 などの適切な数をテキストボックスに入力します。
7. デバッガサーバーは、手順 3 で指定したのとは別のポートから、Eclipse クライアントからのコマンドをリスンします。ColdFusion では、デフォルトで、使用可能なポートをランダムに使用してデバッガーサーバーを起動します。このため、ColdFusion (つまり、デバッガサーバー) がファイアウォールの外側になり、デバッガがリスンするランダムのポートをファイアウォールがブロックする問題が発生する可能性があります。

   この問題を防ぐには、デバッガサーバーの固定ポート番号を指定し、このポートをファイアウォール内で使用できるようにします。デバッガサーバーの固定ポート番号を設定するには、ColdFusion Administrator の [Java と JVM] ページ (または J2EE Application Server 用の適切な場所) で、次の JVM 引数を指定し、portNumber を使用するポートに置き換えます。

   ```
   -DDEBUGGER_SERVER_PORT=portNumber
   ```

8. ColdFusion を再起動します。J2EE 設定の ColdFusion を実行している場合は、指定のデバッグポートを使用してデバッグモードでサーバーを再起動します。
9. デバッグ設定を変更するには、Eclipse で [ウィンドウ]-[設定]-[ColdFusion]-[デバッグの設定] を選択します。ホームページの URL を指定して、[ホーム] ボタンをクリックしたときにデバッガの [デバッグ出力バッファー] に表示するページを設定できます。また、デバッグ可能なファイルタイプの拡張子や、デバッガに認識させる変数スコープを指定することもできます。サイズの大きなファイルをデバッグする際のパフォーマンスを向上させるために、情報が不要なスコープのチェックはすべて選択解除してください。

   注意：デバッグしているテンプレートの ColdFusion エラーが発生した行でデバッガを停止させるには、[デバッグとロギング]-[デバッグ出力の設定] を選択し、[Robust 例外情報の有効化] チェックボックスを選択します。
10. RDS サーバーを設定するには、Eclipse で [ウィンドウ]-[設定]-[ColdFusion]-[RDS 構成] を選択します。

Eclipse と同じコンピュータ上で ColdFusion を実行している場合、localhost はデフォルトで設定されます。追加の RDS サーバーを使用する場合は、設定情報を入力します。

11 Eclipse と同じコンピュータ上で ColdFusion を実行していない場合は、Eclipse で [ウィンドウ]-[設定]-[ColdFusion]-[マッピングのデバッグ] を選択します。ColdFusion サーバー上のファイルを開くために Eclipse が使用するパスと、Eclipse で編集されたファイルを検出するために ColdFusion が使用するパスを指定します。

マッピングを行うことで、Eclipse と ColdFusion が同じファイルを処理するようにします。たとえば、D:¥MyCoolApp に Eclipse プロジェクトがあるとします。このプロジェクトで編集したファイルは、ColdFusion サーバーにデプロイするために W:¥websites¥MyCoolSite¥ にコピーします。ColdFusion サーバーはこれを D:¥Shared¥websites¥MyCoolSite として認識します。Eclipse のマッピングで、Eclipse ディレクトリは D:¥MyCoolApp であり、サーバーは D:¥Shared¥websites¥MyCoolSite であると指定します。Eclipse は、ファイルパス (D:¥MyCoolApp¥index.cfm) を、ColdFusion サーバーが認識するパス (D:¥Shared¥websites¥MyCoolSite¥index.cfm) に変換します。クライアントとサーバー間の通信の詳細を確認するには、ColdFusion Administrator の JVM 引数に次を追加します。

```
-DDEBUGGER_TRACE=true
```

12 サーバー設定の ColdFusion を実行していない場合は、実行するアプリケーションサーバーの設定ファイルまたはスタートアップスクリプトで Java デバッグパラメーターを指定します。パラメータは次のようになります。

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=<port_number>
```

ColdFusion Administrator のデバッガーの設定ページで指定したのと同じポート番号を指定する必要があります。

サーバー設定の ColdFusion を実行している場合は、ColdFusion Administrator のデバッガーの設定ページを使用すると、ColdFusion によってこれらのデバッグパラメーターが jvm.config ファイルに書き込まれます。

13 サーバー設定の ColdFusion を実行しておらず、アプリケーションサーバーが JRE 1.6 で実行されていない場合は、アプリケーションサーバーで実行している JDK バージョンの tools.jar ファイルを、ColdFusion の ¥lib フォルダーにコピーします。例えば、JRE 1.4 で JRun を実行している場合は、JDK 1.4 の tools.jar ファイルを ColdFusion の ¥lib フォルダーにコピーする必要があります。

14 サーバーバージョンの ColdFusion を実行しており、JRE 1.6 以外のバージョンの JRE を jvm.config ファイルで指定する場合は、jvm.config ファイルで指定されている JDK バージョンの tools.jar ファイルを ColdFusion の ¥lib フォルダーにコピーします。

注意：マルチサーバー設定で実行している ColdFusion アプリケーションをデバッグするには、次のコマンドを使用してコマンドラインから ColdFusion サーバーを開始します。

```
jrun -config <path_to_jvm_config> -start <server_name>
```

Eclipse でのデバッガのテストおよび変更

Eclipse で [ウィンドウ]-[設定]-[ColdFusion]-[RDS 構成] を選択し、[デバッガのテスト] をクリックすると、デバッガサーバーが正しく設定されたかどうかを判断できます。

RDS 設定は、RDS ファイルビューまたは RDS データビューから [Edit RDS Preferences] ボタンをクリックすると簡単に変更できます。

デバッグパースペクティブについて

ColdFusion プラグインをインストールし、ColdFusion でデバッガを有効にし、Eclipse を設定したら、Eclipse で ColdFusion デバッガを使用できるようになります。これは Eclipse のデバッグパースペクティブで使用可能です。

デバッグパースペクティブには、次の項目があります。

- [デバッグ] ペイン : 完了した各セッションの結果が保持されます。このペインの上部には次のボタンが表示されます。
  - 再開 : デバッグセッションを再開します。

- 中断 : デバッグセッションを一時停止します。
- 終了 : デバッグセッションを停止します。
- 切断 : リモートでデバッグしている場合に、選択されたデバッグターゲットからデバッガを切断します。
- 終了したすべての起動を除去 - 終了しているデバッグターゲットをすべて表示からクリアします。
- ステップイン : インクルードしたコード、UDF、CFC などを含めて、コードを行単位で実行します。
- ステップオーバー : インクルードしたコード、UDF、CFC などを除外して、コードを行単位で実行します。
- ステップリターン : インクルードしたコード、UDF、CFC などから呼び出し元のページに戻ります。
- フレームにドロップ : 指定されたスタックフレームに再入します。これは、逆に進み、プログラムを途中で再開するのと同じです。
- ステップフィルタ / ステップデバッグの使用 : すべてのステップ関数にステップフィルタが適用されるようにします。
- メニュー : ビューの管理、システムスレッドの表示、修飾名の表示、モニタの表示を行うメニューを表示します。

- [変数] ペイン : 変数スコープを含む、現在の変数が表示されます。このペインの上部には次のボタンが表示されます。
  - 型名の表示 : 変数の型を表示します。
  - 論理構造の表示 : このボタンはサポートされていません。
  - すべて縮小表示 : パネル内の情報を折り畳んで、変数の型のみが表示されるようにします。
- [ブレークポイント] ペイン : ColdFusion アプリケーションのブレークポイントがリストされます。このペインの上部には次のボタンが表示されます。
  - 選択されたブレークポイントを除去 : ブレークポイントを削除します。
  - すべてのブレークポイントを除去 : ブレークポイントをすべて削除します。
  - 選択したターゲットでサポートされるブレークポイントを表示 : 現在デバッグ中のターゲットのブレークポイントを表示します。
  - ブレークポイント用のファイルへジャンプ - 選択したブレークポイントが設定されているファイルに移動します。
  - すべてのブレークポイントをスキップ : ブレークポイントをすべて無視します。
  - すべて展開 : ペイン内の情報を展開します。
  - すべて縮小表示 : ペイン内の情報を折り畳みます。
  - デバッグビューにリンク : アプリケーションの実行を停止したときに、選択されたブレークポイントをデバッグビューで強調表示します。
  - Java 例外ブレークポイントの追加 : 選択されたブレークポイントに達したときに返す Java 例外を指定できます。
  - メニュー : [ブレークポイント] ペインに表示する情報のタイプを指定できます。
- [デバッグ出力バッファー] : 2 つのペインがあります。[ブラウザ] ペインには、アプリケーション実行中にブラウザに表示される内容が表示されます。[サーバー出力バッファー] ペインにはデバッグ出力が表示されます。
- 編集ペイン : 開いているソースファイルごとに 1 つずつ、ソースペインが積み重ねて表示されます。
- [アウトライン] ペイン : 現在のソースファイルの内容がアウトライン形式で表示されます。

ColdFusion デバッガの使用

ColdFusion Administrator でデバッガを有効にし、Eclipse を設定すると、Eclipse プロジェクト内にある ColdFusion ページをデバッグできるようになります。

ColdFusion デバッガを使用すると、次のタスクを行えます。

- ブレークポイントの設定
- 行単位でのコードの実行
- 変数の検査

ColdFusion アプリケーションのデバッグの開始

1 Eclipse プロジェクト内のデバッグ対象ファイルを開きます。

CFML ソースと同じフォルダ内に Eclipse プロジェクトを作成する必要はありません。別のフォルダ内にプロジェクトを作成し、そのプロジェクトの下にフォルダを作成してから、そのフォルダを CFML ソースがあるフォルダにリンクできます。

2 Eclipse ワークベンチの右上隅にある [デバッグ] をクリックすると、デバッグパースペクティブに移動します。

3 [ウィンドウ]-[ビューの表示]-[デバッグ出力バッファー] を選択し、アプリケーションからの出力と、ブラウザでのアプリケーションの表示を確認します。

4 [ウィンドウ]-[設定] を選択して、デバッグセッションのホームページ、デバッグ可能なファイルタイプの拡張子、[変数] ペインに表示する変数の変数スコープを指定します。「OK」をクリックします。

ホームページは、[デバッグ出力バッファー] ペインで [ホーム] ボタンをクリックすると [デバッグ出力バッファー] ペインに表示されるページです。

5 [編集] ペインにソースが表示されているファイルのデバッグを開始するには、Eclipse ツールバーの [デバッグ] アイコンをクリックします。

6 [新規] をクリックして、新規のデバッグ設定を作成します。

7 アクティブなデバッグセッションのホームページを指定します。

これは、[デバッグ出力バッファー] ペインでデバッグセッションのホームボタンをクリックすると [デバッグ出力バッファー] ペインに表示されるページです。

8 [デバッグ] をクリックしてデバッグセッションを開始します。

注意：テンプレートをデバッグしているときに、そのページをブラウズしようとしたり、更新しようとしたりすると、デバッガが予期しない動作をする場合があります。

ブレークポイントの設定

CFML ファイル内でブレークポイントを設定すると、特定の場所でページの実行を停止できます。ブレークポイントを特定の行に設定すると、CFML の実行はその行の直前で停止します。例えば、次の CFML ページの 3 行目にブレークポイントを設定した場合、<cfset myName = "Wilson"> の前で実行が停止します。

```
 <cfset yourName = "Tuckerman">
<cfoutput>Your name is #yourName#.</cfoutput>
<cfset myName = "Wilson"
```

ブレークポイントを設定してページをデバッグする前に、そのページを実行してページをコンパイルします。これにより、デバッグ時のパフォーマンスが向上します。プロジェクトに所属していないファイルにブレークポイントを設定することはできません。

1 Eclipse で、ブレークポイントを設定するファイルを開きます。

2 ブレークポイントを設定したい行を強調表示して、次のいずれかの操作を行います。

- エディタ領域の左に表示されるマーカーバーをダブルクリックします。
- 右クリックし、[ブレークポイントの切り替え] を選択します。

- Alt+Shift+B を押します。

ブレークポイントが設定された行の頭に青のドットが表示されます。

現在の Eclipse プロジェクトで設定されているブレークポイントのリストは、[ブレークポイント] パネルで確認できます。

Eclipse デバッガの ColdFusion ブレークポイントには、次の 4 つの状態があります。

- 使用可能で有効 : 有効な場所にあるブレークポイントです。塗りつぶされた青の丸印で表され、ここに到達するとコードの実行が停止されます。
- 未解決 : ColdFusion では、メモリ内にロードされているページにブレークポイントが設定されます。ページが変更されており、まだ実行されていない状況では、ColdFusion によってサーバー上で認識されるページとソースが同期していません。この状況では、ColdFusion によって、ブレークポイントの設定対象行が無効と見なされることがあります。ただし、未実行のページを実行すれば、その行は有効になります。このタイプのブレークポイントは疑問符 (?) アイコンで表されます。

  パフォーマンス上の理由により、未解決のブレークポイントの解決は、ページの実行ごとには試行されません。ページを変更して実行した場合にのみ、解決が試行されます。未解決と表示されているブレークポイントが有効であると思われる場合は、ブレークポイントを削除して再度設定してください。
- 無効 : Eclipse で編集されている CFML がメモリ内の CFML と同一であり、無効な行にブレークポイントが設定されていると ColdFusion で判断された場合、ブレークポイントは赤の X として表示されます。
- 使用不可

行単位でのコードの実行

[ステップイン]、[ステップオーバー]、[ステップリターン] ボタンを使用して、行単位で CFML アプリケーションを実行することができます。[ステップイン] は、インクルードファイル (UDF や CFC など) の中に入るときに使用します。[ステップオーバー] ボタンは、インクルードファイル (UDF や CFC など) を迂回して CFML アプリケーション内を進むときに使用します。[ステップリターン] ボタンは、インクルードファイル (UDF や CFC など) から呼び出し元のページに戻るときに使用します。

ステップを正しく機能させるには、コンパイル済みクラスのキャッシュをクリアします。これを行うには、以前のバージョンの ColdFusion でコンパイルした CFML ページをすべて再コンパイルします。サイズの大きいファイルでは、ステップやブレークポイントが低速になる場合があります。パフォーマンスを向上させるために、Eclipse で [ウィンドウ]-[設定]-[ColdFusion]-[デバッグの設定] を選択し、情報が不要なスコープのチェックはすべて選択解除してください。

cfset タグなどの CFML 命令では、ステップインは使用しないでください。ステップインは、ステップオーバーよりもパフォーマンスに負荷がかかります。ステップインは、UDF、CFC、カスタムタグ、およびインクルードファイルで使用します。

関数、タグ、ファイルにステップインする場合、Eclipse では、開いているプロジェクトのいずれかでファイルが表示されている必要があります。ステップイン対象のファイルは開いている Eclipse プロジェクト内にある必要があります。

場合によっては、デバッガが特定の行で停止していても、Eclipse 3.2.1 でスタックトレースが表示されず、ステップボタンが無効になっていることがあります。ステップボタンを有効にするには、[デバッグ] ウィンドウでデバッガサーバーのインスタンスをクリックします。スタックトレースを表示するには、[ステップイン] または [ステップアウト] のいずれかをクリックします。

変数の検査

コードの実行を監視するときには、[変数] パネルに変数の値とスコープを表示できます。[変数] パネルには、CFML コード実行時の変数のスコープと値が表示されます。[設定] ダイアログボックスで選択したスコープに属している変数のみが [変数] ペインに表示されます。

ColdFusion のログファイルの表示

Log File Viewer を使用すると、ColdFusion で生成されるすべてのログファイルの内容を簡単に表示できます。

1 Eclipse で、[ウィンドウ]-[ビューの表示]-[その他]-[ColdFusion]-[CF Log Viewer] を選択します。

2 ログファイルの詳細を表示するには、ファイルの名前をダブルクリックします。

3 別のフォルダのログファイルを含めるには、[Add Log Folder] ボタンをクリックし、フォルダを選択して [OK] をクリックします。

4 リストからフォルダを削除するには、フォルダを選択して [Remove Log Folder] ボタンをクリックします。この操作を行っても、コンピュータのファイルシステムからそのフォルダは削除されません。

5 コンピュータのファイルシステムからログファイルを削除するには、[Delete Log File] ボタンをクリックします。

6 詳細ペインの内容を削除するには、[Menu] ボタンをクリックして [Clear Log] をクリックします。

7 詳細ペインの内容を更新するには、[Menu] ボタンをクリックして [Refresh Log] をクリックします。

スケジューラーの使用

ColdFusion 10 のスケジューラー拡張機能を使用すると、きめ細かく、拡張可能な、整然とした方法でタスクをスケジュールできます。この拡張には次のものが含まれます。

- **Quartz スケジューリングサービス：**詳しくは、http://www.quartz-scheduler.org/ を参照してください。
- **グルーピング：**次に示すように、タスクを異なるグループにまとめます。

```
<cfschedule
 action=update
 task=t1 url="www.adobe.com"
 group="G1"/>
```

  タスクをグループにまとめると、同じグループのすべてのタスクをまとめて再開または一時停止でき、タスクごとに繰り返す必要がありません。

- **アプリケーション固有のタスク：**サーバーレベルでのスケジュールタスクとは別に、アプリケーションのみが認識できるアプリケーションレベルのタスクをスケジュールできます。次に例を示します。

```
<cfschedule
 action=update
 task=t1
 url="www.adobe.com"
 group="G1"
 mode="application"/>
```

  デフォルトのモードは server です。

- **イベント処理：**リスナーをスケジュールされたタスクにアタッチします。

  例えば、次のことを行うリスナー CFC を作成できます。

  - タスクが完了したら、すべての関係者にメールを送信します（onTaskEnd）
  - タスクを実行する必要があるかどうかを判断します（onTaskStart）
  - 例外が発生した場合、タスクを再実行します（onError）
  - URL を呼び出す代わりにメソッド内で提供されているコードを実行します（execute）

```
<cfcomponent implements="CFIDE.scheduler.ITaskEventHandler">
    <cffunction name="onTaskStart" returntype="boolean">
        <cfargument name="context" type="struct"/>
        <cfmail from="a@adobe.com" subject="Scheduler_Scenario_Testing" to="a@adobe.com">
            The Report is about to be generated.
        </cfmail>
        <cfreturn true>
    </cffunction>
    <cffunction name="onMisfire" returntype="void">
        <cfargument name="context" type="struct" required="false"/>
        <cfmail from="a@adobe.com" subject="Scheduler_Scenario_Testing" to="a@adobe.com">
            The Report generation task has misfired.
        </cfmail>
    </cffunction>
    <cffunction name="onTaskEnd" access="public" returntype="void">
        <cfargument name="context" type="struct" required="false"/>
        <cfmail from="a@adobe.com" subject="Scheduler_Scenario_Testing" to="a@adobe.com">
            The Report generation task has Completed.
        </cfmail>
    </cffunction>
    <cffunction name="onError" returntype="void">
        <cfargument name="context" type="struct" required="false"/>
        <cfmail from="a@adobe.com" subject="Scheduler_Scenario_Testing" to="a@adobe.com">
            The Report generation task has errored out.
        </cfmail>
    </cffunction>
        <cffunction name="execute" returntype="void">
        <cffile action="append" file="#Expandpath('.')#/log.txt" output="<br><b>In Execute</b><br>">
    </cffunction>
</cfcomponent>
```

注意：リスナーは CFIDE.scheduler.ITaskEventHandler.cfc を拡張する必要があります。

注意：イベント処理関数の context 引数は構造体で、次のキーが含まれます。onMisfire、onTaskEnd および onTaskStart の関数では、キーは group、mode および task です。onError 関数では、キーは exceptionMessage、group、mode および task です。

- **チェーン：**従属タスクを定義できます。親タスクが実行されると、すべての従属タスクも実行されます。

```
<cfschedule
 action=update
 task=t1
 url="www.adobe.com"
 group="G1"
 mode="application"
 onComplete="t2:DEFAULT:application">
```

この例では、onComplete の値は <task>:<group>:<mode> の形式で指定する必要があります。

注意：サーバー固有タスクの後にアプリケーション固有タスクをチェーンすることはできません。

- **クラスタ：**クラスタセットアップでスケジューラーを実行できます。現在、クラスタは JDBC ジョブストアでのみ動作します。機能には、ロードバランスとジョブフェイルオーバーが含まれます。1 つのアプリケーションで、クラスタセットアップと非クラスタセットアップの両方を使用できます。タスクはどちらのセットアップでも実行できます。

  すべてのクラスタサーバーのシステム時刻が一致している必要があります。

  クラスタセットアップを有効にするには、ColdFusion Administrator を使用します。詳しくは、Scheduled Tasks を参照してください。

- **cron コマンド：**cron コマンドを使用してスケジュールされたタスクをトリガーできます。cron 式は、7 個のサブ式（Seconds、Minutes、Hours、Day-of-Month、Month、Day-of-Week、Year（オプションフィールド））で構成される文字列であり、スケジュールの詳細を個別に記述します。

サブ式はスペース文字で区切ります。

例えば、"0 */2 3-10,21-23 * * ?" の場合、タスクが、毎日 AM 3:00 から AM 10:00 までの間と PM 9:00 から PM 11:58 までの間、2 分おきに実行されます。

詳しくは、http://www.quartz-scheduler.org/docs/tutorial/TutorialLesson06.html を参照してください。

- **タスクの優先度設定：**タスクの優先度を設定できます。同時に開始する必要のある 50 個のタスクがあるものとします。しかし、現在使用できるワーカースレッドは 10 個だけです。そこで、優先度の高い最初の 10 タスクを他より先に開始します。優先度は任意の整数値です。デフォルト値は 5 です。
- **日付の除外：**スケジュールプロセスから日付を除外できます。

  例えば、12 月 25 日を除いて 9 月 1 日から 12 月 30 日までジョブを実行するように設定できます。

  除外の指定は、カンマ区切りの日付のリスト、単独の日付、または日付範囲で行うことができます。

  次に例を示します。

  ```
  <cfschedule .... exclude="date1 TO date2"  .../>

  <cfschedule  ... exclude="02/02/2011,03/03/2011"  .../>
  ```

  date1 から date2 まで（両端を含む）のすべての日付を、スケジュールから除外します。日付文字列の配列または日付オブジェクトの配列を指定できます。
- **エラーの場合：**実行するアクションを指定できます。

  ```
  <cfschedule task="job1" onException="REFIRE,PAUSE,INVOKEHANDLER" ....>
  ```

  つまり、ジョブの実行中にエラーが発生した場合に、タスクの再実行、ジョブの一時停止、またはアタッチされているイベントハンドラーの onError メソッドの呼び出しを行うかどうかを決定できます。
- **タスクが誤って実行された場合：**実行するアクションを指定できます。

  つまり、再実行、再スケジュール、またはアタッチされているイベントハンドラーの onMisfire メソッドの呼び出しを行うかどうかです。

  ```
  <cfschedule task="trigger1" onmisfire="FIRE_NOW
  ,NOW_EXISTING,NOW_REMAINING,NEXT_EXISTING,NEXT_REMAINING,INVOKEHANDLER..../>
  ```

  スケジューラーがシャットダウンしたため、またはジョブを実行するために使用できるスレッドがスレッドプールにないために、永続トリガーが時間どおりに動作しない場合、誤実行が発生します。タスクが誤実行したものと見なされるしきい値は 60 秒です。

  注意：すべての誤実行したタスクは、scheduler.log に記録されます。このファイルは、J2EE 設定では cf_root¥WEB-INF¥cfusion¥logs¥scheduler.log に、サーバー設定では cf_root¥logs¥scheduler.log にあります。
- **一時停止および再開／すべて一時停止およびすべて再開：**次のように、グループ全体のタスクを一時停止または再開します。

  ```
  <cfschedule group="group1" action="pause" .......>

  <cfschedule group="group1" action="resume" .......>
  ```

  これは、次のようにサーバーレベルまたはアプリケーションレベルで指定できます。

  ```
  <cfschedule  action="pauseall" mode=application/>

  <cfschedule  action="resumeall" mode=server/>
  ```
- **タスクの一覧表示：**サーバーレベルまたはアプリケーションレベルですべてのスケジュールされたタスクを一覧表示します。

  ```
  <cfschedule  action="list" mode="application" result ="res" />
  ```

- **再試行：**ジョブの実行結果が例外になった場合、次のように、再試行回数になるまで再実行を続けるようにプリセットできます。

  ```
  <cfschedule task="job1" onException="REFIRE" retryCount="3" ....>
  ```

- **繰り返し：**特定のスケジュールを繰り返す回数を指定します。

  例えば、Job1 を今日の午後 2 時に実行し、10 分間隔で 20 回繰り返す、といったことができます。

  この場合、スケジュールの終了時刻を指定する必要はありません。

  ```
  <cfschedule task="task1" repeat="20" interval="60" ....>
  ```

- **Quartz のカスタマイズ：**熟練したユーザーは、cfusion¥lib¥quartz にある quartz.properties を使用して Quartz をカスタマイズできます。ColdFusion に付属する Quartz の現在のバージョンは 2.02 です。

第 7 章：データへのアクセスおよび使用

データベースおよび SQL の概要

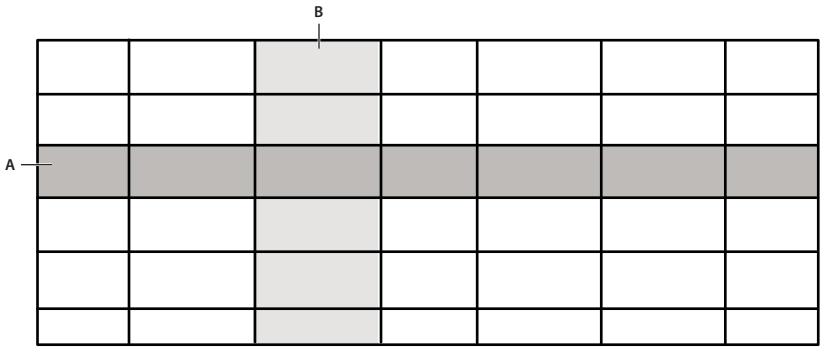
Adobe ColdFusion を使用すると、データベースのデータに対してアクセスおよび変更を行うダイナミックアプリケーションを作成できます。ColdFusion アプリケーションを開発するために、データベースに関する詳細な知識は必要ありませんが、データベースと SQL に関する基本的な概念やテクニックは理解しておく必要があります。

SQL Server、Oracle、DB2 などの各データベースサーバーには、固有の機能およびプロパティがあります。詳細については、データベースサーバーに付属のマニュアルを参照してください。

データベースとは

データベースには、情報を格納するための構造が定義されています。通常、データベースはいくつかのテーブルで構成されます。テーブルは、互いに関連する項目の集まりです。テーブルは、列と行が格子状に配置されたものと考えることができます。ColdFusion では、主に Oracle、DB2、SQL Server などのリレーショナルデータベースが使用されます。

次の図に、データベーステーブルの基本的なレイアウトを示します。



データベーステーブルのレイアウト
A. 行 **B.** 列

1 つの列は、テーブルのすべての行からデータを 1 つずつ集めたものです。1 つの行には、そのテーブルで定義されている各列の項目が 1 つずつ含まれています。

たとえば、会社の従業員の ID、氏名、肩書きなどの情報が格納されているテーブルがあるとします。各行はデータレコードとも呼ばれ、1 つの行が 1 人の従業員に対応します。レコード内の 1 つの列の値は、レコードのフィールドと呼ばれます。

次の図に、会社の従業員情報が含まれている employees というテーブルの例を示します。

EmpID	LastName	FirstName	Title	DeptID	Email	Phone
4	Smith	John	Engineer	3	jsmith	x5833

employees テーブルの例

employee 4 のレコードには次のフィールド値が含まれています。

- LastName フィールドは "Smith" です。
- FirstName フィールドは "John" です。
- Title フィールドは "Engineer" です。

この例では、EmpID フィールドがテーブルのプライマリキーフィールドとして使用されています。プライマリキーには、各レコードを一意に識別するための識別子が含まれています。プライマリキーとして使用できるフィールドには、従業員 ID、部品番号、顧客番号などがあります。プライマリキーを格納する列は、データベーステーブルの作成時に指定するのが普通です。

テーブルにアクセスしてテーブルデータの読み込みや変更を行うには、SQL プログラミング言語を使用します。たとえば、次の SQL ステートメントを実行すると、このテーブル内の部門 ID が 3 である行がすべて返されます。

```
SELECT * FROM employees WHERE DEPTID=3
```

注意：ここでは、SQL のキーワードおよびシンタックスは常に大文字で表記します。テーブル名および列名には、大文字と小文字の両方を使用します。

複数のデータベーステーブルの使用

データベースを設計するときは、複数のテーブルに情報を分割して格納するのが一般的です。次の図には、従業員の情報テーブルと、従業員の住所テーブルの 2 つが示されています。

EmpID	LastName	FirstName	Title	DeptID	Email	Phone
1	Jones	Joe	Engineer	3	jjones	x5844
2	Davis	Ken	Manager	4	kdavis	x5854
3	Baker	Mary	Engineer	3	mbaker	x5876
4	Smith	John	Engineer	3	jsmith	x5833
5	Morris	Jane	Manager	3	jmorris	x5833

employees テーブル

EmpID	Street	City	State	Zip
1	4 Main St.	Newton	MA	02158
2	10 Oak Dr.	Newton	MA	02161
3	15 Main St.	Newton	MA	02158
4	56 Maple Ln.	Newton	MA	02160
5	25 Elm St.	Newton	MA	02160

addresses テーブル

この例の各テーブルには、EmpID という列が含まれています。employees テーブルの行と addresses テーブルの行の関連付けは、この列によって行われます。

たとえば、1 人の従業員に関するすべての情報を取得するには、同じ値の EmpID を使用して、employees テーブルの行と addresses テーブルの行をリクエストします。

複数のテーブルを使用する利点の 1 つは、既存のテーブルの構造を変更しなくても新しい情報を格納するテーブルを追加できることです。たとえば、給与情報を追加するには、データベースに新しいテーブルを追加して、その最初の列に従業員 ID を格納し、以降の列に現在の給与、前の給与、ボーナス額、および 401(k) 積立金の天引き率を格納します。

また、小さいテーブルへのアクセスは大きいテーブルへのアクセスより効率的です。たとえば、従業員の住所を更新する場合は addresses テーブルのみを更新すればよく、データベース内の他のテーブルにアクセスする必要はありません。

データベースのアクセス許可

多くのデータベース環境では、データベースを使用するユーザーのアクセス権をデータベース管理者が定義します。これにはユーザー名とパスワードを使用するのが一般的です。ユーザーがデータベースに接続を試みると、ユーザー名とパスワードが有効かどうかが確認され、次に、ユーザーに応じたアクセス制限が課されます。

アクセス権によって、ユーザーの行える処理が次のように制限されます。

- データの読み取り
- データの読み取りと行の追加
- データの読み取り、行の追加、および既存のテーブルの変更

ColdFusion では、ColdFusion Administrator を使用してデータソースと呼ばれるデータベース接続を定義します。接続を定義するときには、ColdFusion がデータベースに接続するために使用するユーザー名とパスワードを指定します。データベース側では、このユーザー名とパスワードに基づいてアクセスを制御できます。

データソースの作成の詳細については、『ColdFusion 設定と管理』を参照してください。

コミット、ロールバック、およびトランザクション

データベースのデータにアクセスする前に、次のデータベースの概念を理解しておくことが重要です。

- コミット
- ロールバック
- トランザクション

データベースに対して永続的な変更を行うことを、データベースのコミットといいます。たとえば、データベースに新しい行を書き込んでも、データベースの変更がコミットされるまで実際の書き込みは行われません。

ロールバックは、データベースへの変更を取り消す処理です。たとえば、データベースに新しい行を書き込んでも、コミットする前ならば書き込みをロールバックできます。コミット後は書き込みをロールバックできません。

ほとんどのデータベースではトランザクションがサポートされています。1 回のトランザクションは、1 つまたは複数の SQL ステートメントで構成され、データベースの読み取り、変更、書き込みなどを行えます。トランザクションを終了するには、そのトランザクションのすべての変更をコミットするか、またはロールバックします。

データベースに複数の書き込みを行う必要があり、それらがすべて正常に行われたことを確認してから書き込みをコミットしたい場合は、トランザクションが役立ちます。この場合は、すべての書き込みを 1 つのトランザクション内で実行し、書き込みを行うたびにエラーを確認します。いずれかの書き込みでエラーが発生したら、すべての書き込みをロールバックします。すべての書き込みが正常に終了したら、トランザクションをコミットします。

たとえば銀行では、口座から口座への振替処理を 1 つの処理単位にカプセル化するためにトランザクションを使用できます。普通預金口座から当座預金口座に振り替える処理では、当座預金口座に入金されない限り普通預金口座から出金されないようにする必要があります。両方の処理を同じトランザクション内で実行すれば、当座預金口座の更新に失敗した場合に普通預金口座からの出金をロールバックできます。

ColdFusion では、cftransaction タグを使用することで、ロールバックとコミットを制御するためのデータベーストランザクションを実装できます。詳細については、『CFML リファレンス』を参照してください。

データベース設計のガイドライン

以上の基本的な説明から、データベースの設計には次のようなルールがあることがわかります。

- 各レコードには、プライマリキーとして使用する一意の識別子 (従業員 ID、部品番号、顧客番号など) が含まれている必要があります。通常、プライマリキーには、リレーショナルデータベースのテーブルで各レコードを一意に識別するための ID が保持されている列を使用します。データベースでは、複数の列をプライマリキーとして使用することもできます。
- 列を定義するときは、その列の SQL データ型 (たとえば、給与列は数値など) を定義します。
- データベースの実装を成功させるためには、ユーザーのニーズを調査し、そのニーズをデータベースの設計に取り入れることが不可欠です。同じ組織内でもデータに対するニーズは変化していきますが、優れた設計のデータベースなら変化にも対応できます。

使用するデータベースソフトウェアやデータベース管理システム (DBMS) の機能を習得するには、実際の製品のマニュアルを参照してください。

SQL の使用

ここでは、SQL とその基本的なシンタックスについて説明し、SQL ステートメントの例を示します。この説明を読めば、ColdFusion の使用を開始できます。SQL の詳細については、データベースに付属の SQL リファレンスを参照してください。

クエリーは、データベースへのリクエストです。クエリーによって、データベースのデータの取得、データベースへの新規データの書き込み、データベースの既存情報の更新、データベースのレコードの削除などが行えます。

SQL (Structured Query Language) は、データベースクエリーを作成するための ANSI/ISO 標準のプログラミング言語です。ColdFusion でサポートされているすべてのデータベースで SQL がサポートされており、データベースにアクセスするすべての ColdFusion タグで SQL ステートメントを渡すことができます。

SQL の例

ColdFusion で最も頻繁に使用される SQL ステートメントは、SELECT ステートメントです。SELECT ステートメントはデータベースからデータを読み取り、ColdFusion に返します。たとえば、次の SQL ステートメントは、employees テーブルからすべてのレコードを読み取ります。

```
SELECT * FROM employees
```

このステートメントは、"employees テーブルからすべての列を選択する " という意味です。ワイルドカード記号 * はすべての列を示します。

Dreamweaver MX 2004、Adobe Dreamweaver CS3、または HomeSite+ のビルトインのクエリービルダでは、取得するテーブルやレコードを選択することで、グラフィカルな操作で SQL ステートメントを構築できます。

テーブルのすべての行ではなく、特定の行のみを取得したいことも多くあります。次の例では、employees テーブル内の DeptID 列の値が 3 である行がすべて返されます。

```
SELECT * FROM employees WHERE DeptID=3
```

このステートメントは、"DeptID が 3 である行を employees テーブルからすべて選択する " という意味です。

また、テーブル内のどの列を返すかを指定することもできます。たとえば次のようにすると、テーブル内のすべての列ではなく、特定の列が返されます。

```
SELECT LastName, FirstName FROM employees WHERE DeptID=3
```

このステートメントは、"DeptID が 3 である行の FirstName 列と LastName 列を employees テーブルから選択する " という意味です。

テーブルからのデータ読み込みだけでなく、SQL INSERT ステートメントを使用してデータをテーブルに書き込むこともできます。次のステートメントは、新しい行を employees テーブルに追加します。

```
INSERT INTO employees(EmpID, LastName, Firstname) VALUES(51, 'Doe', 'John')
```

SQL シンタックスの基本要素

次の表では、主な SQL コマンドの要素を簡単に説明します。

ステートメント

SQL ステートメントは、常に SQL の動詞で始まります。次のキーワードは、一般的に使用される SQL の動詞です。

キーワード	説明
SELECT	指定されたレコードを取得します。
INSERT	新しい行を追加します。
UPDATE	指定された行の値を変更します。
DELETE	指定された行を削除します。

ステートメントの節

次のキーワードは、詳細な SQL ステートメントを作成する場合に使用します。

キーワード	説明
FROM	操作を行うデータテーブルを指定します。

キーワード	説明
WHERE	操作に対する条件を設定します。
ORDER BY	指定された順序で結果セットを並べ替えます。
GROUP BY	指定された選択リスト項目によって結果セットをグループ化します。

演算子

次の基本的な演算子は、条件の指定と論理演算および数値演算に使用します。

演算子	説明
AND	両方の条件が満たされている必要があることを指定します。
OR	少なくとも 1 つの条件が満たされている必要があることを指定します。
NOT	除外する条件を指定します。
LIKE	パターンとの照合を行います。
IN	値のリストとの照合を行います。
BETWEEN	値の範囲との照合を行います。
=	等しい
<>	等しくない
<	より小さい
>	より大きい
<=	より小さいか、または等しい
>=	より大きいか、または等しい
+	加算
-	減算
/	除算
*	乗算

データベースでの大文字と小文字の区別

ColdFusion は、大文字と小文字が区別されないプログラミング環境です。大文字と小文字が区別されない環境では、次のステートメントがいずれも同等と見なされます。

```
<cfset foo="bar">
<CFSET FOO="BAR">
<CfSet FOO="bar">
```

しかし、多くのデータベース、特に UNIX のデータベースでは大文字と小文字が区別されます。大文字と小文字が区別される場合は、SQL クエリーで列名やテーブル名を指定するときに、大文字と小文字を正確に使い分ける必要があります。

たとえば、大文字と小文字が区別されるデータベースでは、次のクエリーは同等ではありません。

```
SELECT LastName FROM EMPLOYEES
SELECT LASTNAME FROM employees
```

大文字と小文字が区別されるデータベースでは、employees と EMPLOYEES は 2 つの異なるテーブルと見なされます。

ご使用のデータベースで大文字と小文字がどう扱われるかについては、該当する製品のマニュアルを参照してください。

SQL に関する注意事項

ColdFusion で SQL を作成する際は、次のガイドラインに従ってください。

- ColdFusion 変数を SQL 式で使用し、変数値が単一引用符を含む文字列の場合、ColdFusion が引用符を解釈しないようにするには、変数を PreserveSingleQuotes 関数に設定します。その例を次に示します。

```
<cfset List = "'Suisun', 'San Francisco', 'San Diego'">
<cfquery name = "GetCenters" datasource = "cfdocexamples">
    SELECT Name, Address1, Address2, City, Phone
    FROM Centers
    WHERE City IN (#PreserveSingleQuotes(List)#)
</cfquery>
```

- SQL については、ここに記載されている以外にも多数の注意事項があります。適切な SQL ガイドを購入して参照することをお勧めします。
- クエリーを正しく実行するには、参照するデータソース、列、およびテーブルが存在している必要があります。
- DBMS のベンダによっては、非標準の SQL シンタックス (方言) が製品で使用されていることがあります。ColdFusion では SQL の検証は行われません。SQL はデータベースに渡されてから検証されるので、ご使用のデータベースでサポートされているすべてのシンタックスを自由に使用できます。非標準の SQL の使用方法については、DBMS のマニュアルを参照してください。

データベースからのデータの読み取り

データベースからデータを読み取るには、SQL SELECT ステートメントを使用します。この SQL ステートメントの一般的なシンタックスは次のとおりです。

```
SELECT column_names
    FROM table_names
    [ WHERE search_condition ]
    [ GROUP BY group_expression ] [HAVING condition]
    [ ORDER BY order_condition [ ASC | DESC ] ]
```

角括弧 ([]) 内のステートメントはオプションです。

注意：データベースによっては、SELECT に追加オプションがあります。SELECT のシンタックスの詳しい説明については、該当する製品のマニュアルを参照してください。

SELECT ステートメントの結果

データベースによって SELECT ステートメントが処理されると、リクエストされたデータを含むレコードセットが返されます。レコードセットの形式は、行と列を持つテーブルです。たとえば、次のクエリーを実行すると、

```
SELECT * FROM employees WHERE DeptID=3
```

クエリーによってデータベーステーブルが返されます。SELECT ステートメントによって ColdFusion に返されるデータはデータベーステーブルの形式なので、ColdFusion でその結果に対して SQL クエリーを実行することができます。この機能はクエリーオブクエリーと呼ばれます。クエリーオブクエリーの詳細については、413 ページの「データのアクセスおよび取得」を参照してください。

次の例では、SELECT ステートメントを使用してテーブルから特定の列のセットのみを返します。

```
SELECT LastName, FirstName FROM employees WHERE DeptID=3
```

結果のフィルタ

SELECT ステートメントを使用すると、クエリーの結果を絞り込んで、特定の条件に一致するレコードのみを返すことができます。たとえば、部門 3 の従業員のすべてのデータベースレコードにアクセスする場合は、次のクエリーを使用します。

```
SELECT * FROM employees WHERE DeptID=3
```

WHERE 節を使用して、複数の条件を結合できます。たとえば、次の例では 2 つの条件を使用しています。

```
SELECT * FROM employees WHERE DeptID=3 AND Title='Engineer'
```

結果のソート

デフォルトでは、SQL クエリーから返されるレコードに対して、データベースによるソートは行われません。同じクエリーを複数回実行した場合でも、返されるレコードが毎回同じ順序で並んでいる保証はありません。

レコードを特定の順序にソートする必要がある場合は、SQL ステートメントで、データベースから返されるレコードをソートするように指定できます。これを行うには、SQL ステートメントに ORDER BY 節を含めます。

たとえば、次の SQL ステートメントでは、LastName 列に基づいてレコードがソートされたテーブルが返されます。

```
SELECT * FROM employees ORDER BY LastName
```

次のように、ORDER BY 節で複数のフィールドを組み合わせてソート順を指定することもできます。

```
SELECT * FROM employees ORDER BY DepartmentID, LastName
```

このステートメントでは、まず部門 ID でソートされ、次に部門別に姓でソートされた結果が返されます。

列のサブセットの取得

データベーステーブルから列のサブセットのみを取得したい場合があります。次の例では、FirstName、LastName、および Phone 列のみが返されます。この例は、すべての従業員の電話番号を表示する Web ページを構築する場合などに役立ちます。

```
SELECT FirstName, LastName, Phone FROM employees
```

ただし、このクエリーではテーブルの行がアルファベット順に並びません。ソートするには、次のように ORDER 節を SQL に含めます。

```
SELECT the FirstName, LastName, Phone
    FROM employees
    ORDER BY LastName, FirstName
```

列のエイリアスの使用

SQL ステートメントの結果としてふさわしくない列名が使用されていることがあります。たとえば、ColdFusion の予約語 (EQ など) と同じ名前の列がデータベースに含まれている場合などが考えられます。このような場合は、次のようにクエリーの中で列名を変更することができます。

```
SELECT EmpID, LastName, EQ as MyEQ FROM employees
```

このクエリーの結果には、EmpID、LastName、および MyEQ という列が含まれます。

複数のテーブルへのアクセス

データベースでは、関連する情報を複数のテーブルに分割して保存していることがあります。1 回のクエリーで複数のテーブルから情報を抽出することができます。それには、次のように SELECT ステートメントに複数のテーブル名を指定します。

```
SELECT LastName, FirstName, Street, City, State, Zip
    FROM employees, addresses
    WHERE employees.EmpID = addresses.EmpID
    ORDER BY LastName, FirstName
```

この SELECT ステートメントでは、EmpID フィールドを使用して 2 つのテーブルを結合しています。このクエリーでは、EmpID 列にテーブル名の接頭辞を付けています。これは、各テーブルに EmpID という名前の列があるために必要になります。複数のテーブルに同じ列名がある場合は、列名の前にテーブル名を付けます。

この場合、LastName および FirstName の情報は employees テーブルから抽出され、Street、City、State、および ZIP の情報は addresses テーブルから抽出されます。この出力は、従業員向けニュースレターの送付先リストを生成する場合などに使用できます。

複数のテーブルを参照する SELECT ステートメントの結果は、対応する行の情報を結合した単一の結果テーブルになります。結合 (join) とは、複数の行の情報を組み合わせて 1 行の結果を生成することを意味します。この場合、結果のレコードセットは次のような構造になります。

LastName	FirstName	Street	City	State	Zip

EmpID フィールドを使用して 2 つのテーブルの情報を結合したにもかかわらず、出力には EmpID フィールドが含まれていない点に注意してください。

データベースの変更

SQL を使用すると、次に示す方法でデータベースを変更できます。

データベースへのデータの挿入

データベースに情報を書き込むには、SQL の INSERT ステートメントを使用します。書き込みによって、新しい行がデータベーステーブルに追加されます。INSERT ステートメントの基本的なシンタックスは次のとおりです。

```
INSERT INTO table_name(column_names) VALUES(value_list)
```

指定する項目は次のとおりです。

- **column_names** には、カンマ区切りの列のリストを指定します。
- **value_list** には、カンマ区切りの値のリストを指定します。値の順序と列名の順序は対応している必要があります。

注意：データベースによっては、INSERT に追加オプションがあります。INSERT のシンタックスの詳しい説明については、該当する製品のマニュアルを参照してください。

たとえば、次の SQL ステートメントは、新しい行を employees テーブルに追加します。

```
INSERT INTO employees(EmpID, LastName, Firstname) VALUES(51, 'Smith', 'John')
```

このステートメントによって、employees テーブルに行が作成され、行の EmpID、LastName、および FirstName フィールドの値が設定されます。行の残りのフィールドには null が設定されます。**null** は、フィールドに値が含まれていないことを意味します。

テーブルを作成するときには、テーブルおよびテーブルの列にプロパティを設定できます。列のプロパティの中には、そのフィールドで null 値をサポートするかどうかを設定するプロパティがあります。null をサポートしているフィールドは、INSERT ステートメントで省略できます。新しい行を追加すると、省略したフィールドには自動的に null が設定されます。

null をサポートしていないフィールドについては、INSERT ステートメントの中でフィールドの値を指定します。指定しないとエラーが発生します。

クエリー内の LastName および FirstName の値は引用符で囲んでいます。これは、これらの列が文字列を格納するように定義されているからです。数値データを引用符で囲む必要はありません。

データベースのデータの更新

テーブルの行の値を更新するには、SQL の UPDATE ステートメントを使用します。UPDATE を使用すると、テーブル内の特定の行またはすべての行のフィールドを更新できます。UPDATE ステートメントのシンタックスは次のとおりです。

```
UPDATE table_name
    SET column_name1=value1, ... , column_nameN=valueN
    [ WHERE search_condition ]
```

注意：データベースによっては、UPDATE に追加オプションがあります。UPDATE のシンタックスの詳しい説明については、該当する製品のマニュアルを参照してください。

レコードのプライマリキーフィールドは更新しないでください。通常、そのような操作はデータベースによって禁止されています。

UPDATE ステートメントでは、SELECT ステートメントと同様にオプションの WHERE 節を使用して、変更する列を指定します。次の UPDATE ステートメントでは、John Smith の電子メールアドレスを更新します。

```
UPDATE employees SET Email='jsmith@mycompany.com' WHERE EmpID = 51
```

UPDATE を使用するときは注意が必要です。たとえば、WHERE 節を指定しない次のようなステートメントがあるとします。

```
UPDATE employees SET Email = 'jsmith@mycompany.com'
```

これを実行すると、テーブル内のすべての行の Email フィールドが更新されます。

データベースからのデータの削除

テーブルから行を削除するには、DELETE ステートメントを使用します。DELETE ステートメントのシンタックスは次のとおりです。

```
DELETE FROM table_name
    [ WHERE search_condition ]
```

注意：データベースによっては、DELETE に追加オプションがあります。DELETE のシンタックスの詳しい説明については、該当する製品のマニュアルを参照してください。

次の形式のステートメントを使用すると、テーブルからすべての行を削除できます。

```
DELETE FROM employees
```

DELETE ステートメントでは通常、WHERE 節を指定してテーブルの特定の行を削除します。たとえば、次のステートメントを使用すると、テーブルから John Smith が削除されます。

```
DELETE FROM employees WHERE EmpID=51
```

複数のテーブルの更新

これまでの例では、単一のデータベーステーブルの変更方法について説明してきました。しかし、データベースでは複数のテーブルを使用している場合があります。

複数のテーブルを更新するには、テーブルごとに 1 つの INSERT ステートメントを使用し、すべての INSERT ステートメントを 1 回のデータベーストランザクションで実行するという方法があります。トランザクションには 1 つまたは複数の SQL ステートメントが含まれており、それらを 1 つの単位としてロールバックまたはコミットできます。トランザクション内のステートメントが 1 つでも失敗した場合は、トランザクション全体をロールバックすることで、そのトランザクション内で既に実行された書き込みを取り消すことができます。選択、更新、および削除についても同じ方法を使用できます。次の例では、cftransaction タグを使用して、複数の SQL ステートメントをまとめています。

```
<cftransaction>

<cfquery name="qInsEmp" datasource="cfdocexamples">
    INSERT INTO Employees (FirstName,LastName,EMail,Phone,Department)
    VALUES ('Simon', 'Horwith', 'SHORWITH','(202)-797-6570','Research and Development')
</cfquery>

<cfquery name="qGetID" datasource="cfdocexamples">
    SELECT MAX(Emp_ID) AS New_Employee
    FROM Employees
</cfquery>

</cftransaction>
```

データのアクセスおよび取得

ColdFusion には、データベースからデータを取得したり、クエリーデータを操作したりするためのタグが用意されています。cfquery タグを使用してデータソースをクエリーしたり、cfoutput タグを使用してクエリー結果を Web ページに出力したりできます。また、cfqueryparam タグを使用してアプリケーションのセキュリティリスクを低減できます。

ダイナミックデータの操作

Web アプリケーションページは、データをダイナミックに公開できるという点でスタティックな Web ページと異なります。ブラウザページでのユーザーの操作に応じて、データベースにクエリーを実行したり、LDAP またはメールサーバーに接続したり、COM、DCOM、CORBA、Java オブジェクトを使用して実行時にデータの取得、更新、挿入、削除を行ったりすることができます。

ColdFusion の開発者にとって、データソースという用語は、ローカルやネットワーク上にある、さまざまなタイプの構造化コンテンツを意味します。Web サイト、LDAP サーバー、POP メールサーバー、およびさまざまな形式のドキュメントに対してクエリーを実行することができます。しかし、アプリケーションで最も利用されているのはデータベースであり、この場合、データソースという用語は、ColdFusion からデータベースにアクセスするためのエントリポイントという意味で使用されます。

ここでは、cfdocexamples というデータソースからデータを取得するクエリーを構築します。

データベースクエリーの作成では、次のものを使用します。

- ColdFusion データソース
- cfquery タグ
- SQL コマンド

データの取得

実行時にデータベースにクエリーを実行して、データを取得することができます。取得したデータはレコードセットと呼ばれ、ページ上にクエリーオブジェクトとして保管されます。クエリーオブジェクトは、レコードセット値と RecordCount、CurrentRow、ColumnList、SQL、Cached、および SQLParameter クエリー変数を含んでいる特殊なエンティティです。クエリーオブジェクトの名前は cfquery タグの name 属性で指定します。クエリーオブジェクトのことを、単にクエリーと呼ぶこともあります。

cfquery タグの簡単な例を次に示します。

```
<cfquery name = "GetSals" datasource = "cfdocexamples">
    SELECT * FROM Employee
    ORDER BY LastName
</cfquery>
```

注意：クエリーのレコードセットについて説明する場合は、" レコードセット " と " クエリーオブジェクト " を同じ意味で使用します。詳細については、433 ページの「クエリーオブクエリーの使用」を参照してください。

データベースからデータを取得するには、次の手順を行います。

- ページ上で cfquery タグを使用して、データベースへの接続方法を ColdFusion に指示します。
- cfquery ブロック内に SQL コマンドを記述して、データベースから取得するデータを指定します。
- クエリーオブジェクトを参照し、cfoutput、cfgrid、cftable、cfgraph、cftree などのデータ表示タグでそのデータ値を使用します。

cfquery タグ

cfquery タグは最も頻繁に使用される CFML タグの 1 つです。このタグは、クエリーによって返されるデータの取得および参照に使用します。ページ上で cfquery タグが認識されると、次の処理が実行されます。

- 指定されたデータソースに接続します。
- ブロック内に含まれている SQL コマンドが実行されます。
- 結果セット値がクエリーオブジェクトとしてページに返されます。

cfquery タグのシンタックス

次のコードに、cfquery タグのシンタックスを示します。

```
<cfquery name="EmpList" datasource="cfdocexamples">
    SQL code...
</cfquery>
```

このクエリーコードによって、ColdFusion で次の処理が実行されます。

- cfdocexamples データソース ("cfdocexamples.mdb" データベース) に接続します。
- 指定された SQL コードが実行されます。
- 取得されたデータがクエリーオブジェクト EmpList に保管されます。

クエリーを作成してデータを取得するときは、次のガイドラインに従ってください。

- cfquery タグはブロックタグであるため、<cfquery> 開始タグと </cfquery> 終了タグの両方を使用します。
- クエリーの name および datasource 属性は、cfquery 開始タグに記述します。
- クエリーの実行時に行う処理をデータベースに指示するには、cfquery ブロック内に SQL ステートメントを配置します。
- SQL 内でリテラル文字列を参照するときは、単一引用符 (') を使用します。たとえば、SELECT * FROM mytable WHERE FirstName='Jacob' は、名前フィールドが Jacob であるレコードを mytable からすべて抽出します。
- 属性値は、"attrib_value" のように二重引用符 (") で囲みます。
- cfquery タグ内でデータソースを参照する前に、そのデータソースが ColdFusion Administrator 内に存在することを確認してください。
- SQL ステートメントで参照している列やテーブルが存在しない場合、クエリーは失敗します。
- cfoutput、cfgrid、cftable、cfgraph、cftree などの表示タグで、クエリーの名前を指定して、クエリーデータを参照します。
- ColdFusion で返されるデータベース列からは、テーブルおよび所有者を表す接頭辞は削除されます。たとえば、Employee.Emp_ID をクエリーすると Employee. が削除され、Emp_ID として返されます。列名に重複がある場合は、

エイリアスを使用することで対応できます。詳細については、433 ページの「クエリーオブクエリーの使用」を参照してください。

- SQL ステートメントに、MIN、MAX、COUNT などの SQL 予約語を使用することはできません。予約語はデータベースによって異なるので、予約語のリストについてはデータベースのマニュアルを参照してください。
- SQL で COMPUTE AVG() を使用した場合、ColdFusion 9 では avg() が列名として返されます。以前のバージョン (ColdFusion 5 および ColdFusion MX 7) では ave() が列名として返されていました。
- データベーストリガによって返された結果を取得するには、次の接続パラメータを接続文字列に追加します。

  ```
  AlwaysReportTriggerResults=true
  ```

  このパラメータは、データベーストリガ (データベースに格納され、テーブルが変更されると実行されるプロシージャ) によって生成された結果をドライバからどのように報告するかを指定します。Microsoft SQL Server 2005 の場合、これには、データ定義言語 (DDL) イベントによって起動されるトリガが含まれます。true に設定した場合は、トリガによって生成された結果を含むすべての結果がドライバから返されます。複数のトリガ結果がある場合は、1 つずつ返されます。個々のトリガ結果を取得するには、Statement.getMoreResults メソッドを使用します。警告やエラーが発生した場合は、結果内で報告されます。

クエリーの構築

前述のように、クエリーを構築するには cfquery タグと SQL を使用します。

注意：以降の多くの手順では、"cfdocexamples.mdb" データベースに接続する cfdocexamples データソースを使用しています。このデータソースは、デフォルトでインストールされています。データソースの追加および設定については、『ColdFusion 設定と管理』を参照してください。

テーブルに対するクエリーの実行

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>Employee List</title>
</head>
<body>
<h1>Employee List</h1>
<cfquery name="EmpList" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary, Contract
    FROM Employee
</cfquery>
</body>
</html>
```

注意：クエリーは ColdFusion コンポーネントに含めて、構造化された再利用可能なコードにすることをお勧めします。ただし、ここでは、簡略化のために ColdFusion ページの本文にクエリーを記述しています。ColdFusion コンポーネントの使用方法の詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

2 このページに "emplist.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。たとえば、Windows コンピュータ上のデフォルトのパスは次のようになります。

C:¥CFusion¥wwwroot¥myapps¥

3 Web ブラウザに次の URL を入力します。

http://localhost/myapps/emplist.cfm

ヘッダのみが表示されます。

4 ブラウザでソースを表示します。

ColdFusion によって EmpList データセットが作成されますが、ブラウザに返されるのは HTML およびテキストのみです。ページのソースを見ると、HTML タグと見出しの " 従業員リスト " しか含まれていません。ページにデータセットを表示するには、データを出力するタグと変数をコーディングします。

コードの説明

作成したクエリーによって、cfdocexamples データベースからデータが取得されます。このコードの太字の部分について、次の表で説明します。

コード	説明
<cfquery name="EmpList" datasource="cfdocexamples">	cfdocexamples データソースで指定されているデータベースに対してクエリーを実行します。
SELECT FirstName, LastName, Salary, Contract FROM Employee	Employee テーブルの FirstName、LastName、Salary、Contract フィールドからデータを取得します。
</cfquery>	cfquery ブロックを終了します。

クエリーデータの出力

クエリーを定義したら、cfoutput タグと query 属性を使用して、レコードセットからデータを出力できます。query 属性を使用するときは、次の点に注意してください。

- cfoutput ブロックに含まれているコード全体が、データベースから返されたレコードセットの行ごとに 1 回ずつループされます。
- データをページに出力するには、cfoutput ブロック内で具体的な列名を参照します。
- cfoutput ブロックの中または前後にテキスト、CFML タグ、および HTML タグを配置して、ページ上のデータの形式を設定できます。
- クエリー列を参照するときにクエリー名を指定する必要はありませんが、クエリー名の接頭辞を使用することをお勧めします。たとえば、cfoutput タグで Emplist クエリーを指定した場合、Emplist クエリーの Firstname 列は Firstname で参照できますが、クエリー名の接頭辞を使用して Emplist.Firstname とすることをお勧めします。次の手順でもこの方法を使用しています。

cfoutput タグではさまざまな属性を使用できますが、通常は、query 属性を使用して既存のクエリーの名前を定義します。

1 次のように "emplist.cfm" を編集します。

```
<html>
<head>
<title>Employee List</title>
</head>
<body>
<h1>Employee List</h1>
<cfquery name="EmpList" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary, Contract
    FROM Employee
</cfquery>
<cfoutput query="EmpList">
#EmpList.FirstName#, #EmpList.LastName#, #EmpList.Salary#, #EmpList.Contract#<br>
</cfoutput>
</body>
</html>
```

2 このファイルを保存して Web ブラウザで表示します。

従業員のリストがブラウザに表示されます。データが 1 行ずつ表示されます。

注意：必要な場合は、ブラウザを更新して変更を反映します。

これで、データベースからデータを取得して表示する ColdFusion のアプリケーションページが作成されました。この時点では、出力の体裁は調整されていないので、形式を設定する必要があります。詳細については、692 ページの「データの取得および形式設定の概要」を参照してください。

コードの説明

クエリー結果がページに表示されます。このコードの太字の部分について、次の表で説明します。

コード	説明
<cfoutput query="EmpList">	EmpList クエリーで取得した情報を表示します。
#EmpList.FirstName#, #EmpList.LastName#, #EmpList.Salary#, #EmpList.Contract#	各レコードの FirstName、LastName、Salary、および Contract フィールドの値を、カンマとスペースで区切って表示します。
 	各レコードの末尾に改行を挿入します (次の行に進みます)。
</cfoutput>	cfoutput ブロックを終了します。

クエリーの出力に関する注意事項

クエリーの結果を出力するときは、次のガイドラインに従ってください。

- cfoutput タグで結果を表示する前に、cfquery でデータを取得する必要があります。同じページに両方のタグを記述することもできますが、クエリーは ColdFusion コンポーネントに含め、結果は別のページで出力することをお勧めします。詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。
- クエリーのすべてのレコードのデータを出力するには、cfoutput タグで query 属性を使用してクエリー名を指定します。
- 列の値を出力するには、その列が存在し、アプリケーションに取得されている必要があります。
- cfquery 属性を使用する cfoutput ブロックでは、クエリー変数にクエリー名の接頭辞を追加できます (Emplist.FirstName など)。
- query 属性の値は、他の属性と同様に、二重引用符 (") で囲みます。
- 列の現在の値を出力するには、他の変数を出力する場合と同様に、列名をシャープ記号 (#) で囲みます。
- 変数参照の末尾に
 タグを追加すれば、クエリーの各行がそれぞれ新しい行に表示されます。

クエリー結果に関する情報の取得

cfquery タグを使用してデータベースにクエリーを実行すると、データ (レコードセット) とクエリー変数が取得されます。これらはクエリーオブジェクトに格納されています。次の表で、クエリー変数について説明します。クエリー変数はクエリーのプロパティとも呼ばれます。

変数	説明
RecordCount	クエリーによって返されたレコードの総数
ColumnList	アルファベット順に並べられたクエリー列のカンマ区切りリスト
SQL	実行された SQL ステートメント
Cached	クエリーがキャッシュされたかどうか
SQLParameters	cfqueryparam の値の順序配列
ExecutionTime	クエリーの処理に要した累積時間 (ミリ秒単位)

CFML コードでは、データベーステーブルの列と同じ方法でこれらの変数を使用します。これらの変数を格納するための構造体の名前は、result 属性で指定します。この構造体の名前を使用して、次の例で示すようにクエリー変数を参照します。

ページへのクエリ情報の出力

1 次のように "emplist.cfm" を編集します。

```
<cfset Emp_ID = 1>
<cfquery name="EmpList" datasource="cfdocexamples" result="tmpResult">
    SELECT FirstName, LastName, Salary, Contract
    FROM Employee
    WHERE Emp_ID = <cfqueryPARAM value = "#Emp_ID#"
                          CFSQLType = "CF_SQL_INTEGER">
</cfquery>
<cfoutput query="EmpList">
    #EmpList.FirstName#, #EmpList.LastName#, #EmpList.Salary#, #EmpList.Contract#<br>
</cfoutput> <br>
<cfoutput>
    The query returned #tmpResult.RecordCount# records.<br>
    The query columns are:#tmpResult.ColumnList#.<br>
    The SQL is #tmpResult.SQL#.<br>
    Whether the query was cached: #tmpResult.Cached#.<br>
    Query execution time: #tmpResult.ExecutionTime#.<br>
</cfoutput>
<cfdump var="#tmpResult.SQLParameters#">
```

2 このファイルを保存して Web ブラウザで表示します。

従業員の数が従業員リストの下に表示されます。必要に応じて、ブラウザを更新し、スクロールして RecordCount 出力を表示します。

コードの説明

クエリーで取得されたレコードの数が表示されます。このコードおよびその機能について、次の表で説明します。

コード	説明
<cfoutput>	以降を表示します。
The query returned	"The query returned" というテキストを表示します。
#EmpList.RecordCount#	EmpList クエリーで取得されたレコードの数を表示します。
records.	"records." というテキストを表示します。
</cfoutput>	cfoutput ブロックを終了します。

クエリー変数に関する注意事項

クエリー変数を使用するときは、次のガイドラインに従ってください。

- クエリー変数の値をページに出力するには、cfoutput ブロックでクエリー変数を参照します。
- クエリー変数の参照をシャープ記号 (#) で囲んで、変数の名前をその現在値に置き換えます。
- RecordCount または ColumnList プロパティを出力する場合は、cfoutput タグで query 属性を使用しないでください。これを行うと、クエリーの行数と同じ回数その値が出力されてしまいます。代わりに、変数にクエリー名の接頭辞を追加します。

cfqueryparam によるセキュリティの強化

DBMS (Database Management System) の中には、1 回のクエリーで複数の SQL ステートメントを送信できるものがあります。この場合、ハッカーが悪意のある SQL ステートメントを既存のパラメータの後に追加して、ダイナミックなクエリーで URL やフォーム変数を改変するおそれがあります。クエリー文字列にパラメータを渡すときは、潜在的なセキュリティリスクに十分に注意してください。このようなリスクは、ColdFusion、ASP、CGI などの多くの開発環境に存在する可能性があります。cfqueryparam タグを使用すると、このリスクを低減できます。

クエリー文字列のパラメータについて

クエリー文字列にパラメータを渡している場合は、意図した情報のみが渡されるように対策が必要です。たとえば、次の ColdFusion クエリーには WHERE 節が含まれています。これは、フォームの LastName フィールドで指定された姓に一致するデータベースエントリのみを選択するために使用されています。

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary
    FROM Employee
    WHERE LastName='#Form.LastName#'
</cfquery>
```

この場合、次のような悪意のある URL を使用してページが呼び出されるおそれがあります。

http://myserver/page.cfm?Emp_ID=7%20DELETE%20FROM%20Employee

その結果、ColdFusion は次のクエリーを実行しようとします。

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT * FROM Employee
    WHERE Emp_ID = 7 DELETE FROM Employee
</cfquery>
```

Emp_ID 列の整数値だけでなく、SQL ステートメントとして実行可能な悪意のある文字列コードも渡されています。このクエリーが正常に実行されると、Employee テーブルからすべての行が削除されます。このような操作が行えるのは明らかに望ましくありません。これを防ぐには、クエリー文字列のパラメータの内容を評価します。

cfqueryparam の使用

cfqueryparam タグを使用して、クエリー文字列のパラメータを評価し、SQL ステートメント内で ColdFusion 変数を渡すことができます。このタグは、変数の値をデータベースに送信する前に評価します。cfqueryparam タグの cfsqltype 属性に、該当するデータベース列のデータ型を指定します。次の例では、cfdocexamples データソースの Emp_ID 列は整数であるため、cf_sql_integer の cfsqltype を指定しています。

```
<cfquery name="EmpList" datasource="cfdocexamples">
    SELECT * FROM Employee
    WHERE Emp_ID = <cfqueryparam value = "#Emp_ID#"
                              cfsqltype = "cf_sql_integer">
</cfquery>
```

cfqueryparam タグによって、Emp_ID の値が整数のデータ型であることが確認されます。クエリー文字列に整数でないものが含まれている場合 (たとえばテーブルを削除する SQL ステートメントが含まれている場合)、cfquery タグは実行されません。その代わりに、cfqueryparam タグは次のエラーメッセージを返します。

CFSQLTYPE 'CF_SQL_INTEGER' 用の無効なデータ '7 DELETE FROM Employee' です。

文字列に対する cfqueryparam の使用

文字列を含む変数をクエリーに渡すときには、次の例のように cfsqltype の値として cf_sql_char を指定し、maxLength 属性を指定します。

```
<cfquery name = "getFirst" dataSource = "cfdocexamples">
    SELECT * FROM employees
    WHERE LastName = <cfqueryparam value = "#LastName#"
                                cfsqltype = "cf_sql_char" maxLength = "17">
</cfquery>
```

この場合、cfqueryparam によって次のことが確認されます。

- LastName が文字列を含んでいることが確認されます。
- 文字列が 17 文字以下であることが確認されます。
- 単一の値としてデータベースに認識されるように、文字列が単一引用符 (') で囲まれてエスケープされます。ハッカーが悪意のある URL を渡した場合でも、次のように処理されます。

  WHERE LastName = 'Smith DELETE FROM MyCustomerTable'.

cfSqlType の使用

次の表に、cfqueryparam タグの value 属性を評価できる、利用可能な SQL タイプをリストします。

BIGINT	BIT	CHAR	DATE
DECIMAL	DOUBLE	FLOAT	IDSTAMP
INTEGER	LONGVARCHAR	MONEY	MONEY4
NUMERIC	REAL	REFCURSOR	SMALLINT
TIME	TIMESTAMP	TINYINT	VARCHAR

注意：cfsqltype 属性を指定すると、DBMS でバインド変数が使用され、これによってパフォーマンスが大幅に向上する場合があります。

データベースの更新

Adobe ColdFusion を使用して、データベースの情報の挿入、更新、および削除を行うことができます。

データベースの更新について

ColdFusion は、最初はデータベースと簡単に対話する手段として開発されました。通常は一組のページからなる ColdFusion フォームを使用すると、データベースの内容の挿入、更新、および削除をすばやく行うことができます。ColdFusion フォームの一方のページにはエンドユーザーが値を入力するフォームが表示され、もう一方のページではアクション (挿入、更新、または削除) が実行されます。

CFML とともに SQL コマンドを使用するかどうかは、データ操作の範囲およびタイプによって異なります。ColdFusion で SQL コマンドを使用するには、SQL に関する最小限の知識が必要です。

データの挿入

データベースにデータを挿入するには通常、次の 2 つのアプリケーションページを使用します。

- 挿入フォーム
- 挿入アクションページ

挿入フォームは、標準 HTML フォームタグまたは cfform タグを使用して作成できます (709 ページの「cfform タグによるカスタムフォームの作成」を参照)。フォームが送信されると、指定のデータソースに挿入操作 (および呼び出されている他のすべての操作) を行う ColdFusion アクションページにフォーム変数が渡されます。挿入アクションページでは、cfinsert タグまたは、SQL の INSERT ステートメントを含んだ cfquery タグを使用できます。挿入アクションページでは、エンドユーザーに確認メッセージを表示する必要もあります。

HTML 挿入フォームの作成

標準 HTML タグを使用してフォームを作成する手順を次に示します。

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>Insert Data Form</title>
</head>

<body>
<h2>Insert Data Form</h2>

<table>
<!--- begin html form;
put action page in the "action" attribute of the form tag. --->
<form action="insert_action.cfm" method="post">
<tr>
    <td>Employee ID:</td>
    <td><input type="text" name="Emp_ID" size="4" maxlength="4"></td>
</tr>
<tr>
    <td>First Name:</td>
    <td><input type="Text" name="FirstName" size="35" maxlength="50"></td>
</tr>
<tr>
    <td>Last Name:</td>
    <td><input type="Text" name="LastName" size="35" maxlength="50"></td>
</tr>
<tr>
    <td>Department Number:</td>
    <td><input type="Text" name="Dept_ID" size="4" maxlength="4"></td>
</tr>
<tr>
    <td>Start Date:</td>
    <td><input type="Text" name="StartDate" size="16" maxlength="16"></td>
```

```
</tr>
<tr>
    <td>Salary:</td>
    <td><input type="Text" name="Salary" size="10" maxlength="10"></td>
</tr>
<tr>
    <td>Contractor:</td>
    <td><input type="checkbox" name="Contract" value="Yes" checked>Yes</td>
</tr>
<tr>
    <td> </td>
    <td><input type="Submit" value="Submit"> <input type="Reset"
value="Clear Form"></td>
</tr>
</form>
<!--- end html form --->
</table>

</body>
</html>
```

2 このファイルに "insert_form.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、Web ブラウザで表示します。

注意：このフォームは、アクションページを作成しないと機能しません。詳細については、422 ページの「データを挿入するアクションページの作成」を参照してください。

データ入力フォームについての注意事項

アクションページで cfinsert タグを使用してデータベースにデータを挿入する場合は、フォームページの作成時に次のルールに従ってください。

- HTML フォームでは、データを挿入するデータベース列に関係するフィールドのみを作成します。
- デフォルトでは、cfinsert タグを使用すると、すべてのフォームフィールドが、同じ名前のデータベース列に挿入されます。たとえば、Form.Emp_ID の値がデータベース列 Emp_ID に挿入されます。データベース列の名前に一致しないフォームフィールドは、すべて無視されます。

注意：cfinsert タグの formfields 属性を使用して、formfields="prod_ID,Emp_ID,status" のように、挿入するフィールドを指定することもできます。

データを挿入するアクションページの作成

cfinsert タグまたは cfquery タグを使用して、データベースにデータを挿入するアクションページを作成できます。

cfinsert による挿入アクションページの作成

cfinsert タグを使用すると、cfform または HTML フォームからの単純なデータ挿入を最も簡単に処理できます。このタグでは、データベースフィールドの名前に一致するすべてのフォームフィールドのデータが挿入されます。

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head> <title>Input form</title> </head>

<body>
<!--- If the Contractor check box is clear,
    set the value of the Form.Contract to "No" --->
<cfif not isdefined("Form.Contract")>
    <cfset Form.Contract = "N">
</cfif>

<!--- Insert the new record --->
<cfinsert datasource="cfdocexamples" tablename="EMPLOYEE">

<h1>Employee Added</h1>
<cfoutput> You have added #Form.FirstName# #Form.Lastname# to the employee database.
</cfoutput>

</body>
</html>
```

2 このページに "insert_action.cfm" という名前を付けて保存します。

3 Web ブラウザで "insert_form.cfm" を表示し、値を入力します。

注意：値を挿入する前後で cfdocexamples データソースの Employees テーブルの表示を比較することをお勧めします。

4 送信ボタンをクリックします。

Employee テーブルに値が挿入され、確認メッセージが表示されます。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfif not isdefined("Form.Contract")> <cfset Form.Contract = "N"> </cfif>`	Form.Contract の値が定義されていない場合は、値を No に設定します。 [Contractor] チェックボックスがチェックされていない場合は、アクションページにデータが渡されませんが、データベースフィールドには何らかの値が必要です。
`<cfinsert datasource="cfdocexamples" tablename="EMPLOYEE">`	cfdocexamples データベースの Employee テーブルに行を作成します。フォームのデータを、フォームフィールドと同じ名前のデータベースフィールドに挿入します。
`<cfoutput>You have added #Form.FirstName# #Form.Lastname# to the employee database.</cfoutput>`	データベースに値が挿入されたことをユーザーに通知します。

注意：cfinsert タグまたは cfupdate タグでフォーム変数を使用した場合、データベースの数値列、日付列、時刻列に送信されるフォームデータは、ColdFusion によって自動的に検証されます。これらのフィールドに非表示フィールド検証関数を使用して、カスタムのエラーメッセージを表示できます。詳細については、692 ページの「データの取得および形式設定の概要」を参照してください。

cfquery による挿入アクションページの作成

フォームデータを使用して複雑な挿入を行うには、cfinsert タグを使用する代わりに、cfquery タグ内で SQL の INSERT ステートメントを使用します。SQL の INSERT ステートメントは柔軟性が高く、情報を選択的に挿入したり、ステートメントの中で関数を使用したりすることができます。

次の手順では、422 ページの「cfinsert による挿入アクションページの作成」に従って "insert_action.cfm" ページが既に作成されていることを前提としています。

1 "insert_action.cfm" の cfinsert タグを、次の cfquery のコード (太字部分) に置き換えます。

```
<html>
<head>
    <title>Input form</title>
</head>

<body>
<!--- If the Contractor check box is clear), set the value of the Form.Contract
    to "No" --->
<cfif not isdefined("Form.Contract")>
    <cfset Form.Contract = "No">
</cfif>

<!--- Insert the new record --->
<cfquery name="AddEmployee" datasource="cfdocexamples">
    INSERT INTO Employee
    VALUES (#Form.Emp_ID#, '#Form.FirstName#',
    '#Form.LastName#', #Form.Dept_ID#,
    '#Form.StartDate#', #Form.Salary#, '#Form.Contract#')
</cfquery>

<h1>Employee Added</h1>
<cfoutput>You have added #Form.FirstName# #Form.Lastname# to the employee database.
</cfoutput>

</body>
</html>
```

2 このページを保存します。

3 Web ブラウザで "insert_form.cfm" を表示し、値を入力します。

4 送信ボタンをクリックします。

Employee テーブルに値が挿入され、確認メッセージが表示されます。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
`<cfquery name="AddEmployee" datasource="cfdocexamples"> INSERT INTO Employee VALUES (#Form.Emp_ID#, '#Form.FirstName#', '#Form.LastName#', #Form.Dept_ID#, '#Form.StartDate#', #Form.Salary#, '#Form.Contract#') </cfquery>`	cfdocexamples データベースの Employee テーブルに新しい行を挿入します。追加するフォームフィールドを指定します。 データベースと同じ左から右の順番ですべてのデータベースフィールドにデータを挿入するので、クエリー内でデータベースフィールドの名前を指定する必要はありません。 #From.Emp_ID#、#Form.Dept_ID# および #Form.Salary# は数値なので、引用符で囲む必要はありません。

特定のフィールドへの挿入

前述の例では、テーブルのすべてのフィールド (Employee テーブルには 7 つのフィールドがあります) にデータを挿入しましたが、すべてのフィールドにデータを追加する必要がない場合もあります。特定のフィールドにデータを挿入するには、cfquery の SQL ステートメントで、INSERT INTO および VALUES の後にフィールド名を指定します。たとえば、次の cfquery では、給与と開始日を更新していません。これらのフィールドのデータベース値は、データベースの設計によって 0 または null になります。

```
<cfquery name="AddEmployee" datasource="cfdocexamples">
    INSERT INTO Employee
        (Emp_ID,FirstName,LastName,
        Dept_ID,Contract)
    VALUES
        (#Form.Emp_ID#,'#Form.FirstName#','#Form.LastName#',
        #Form.Dept_ID#,'#Form.Contract#')
</cfquery>
```

データの更新

データベースのデータを更新するには通常、次の 2 つのアプリケーションページを使用します。

- 更新フォーム
- 更新アクションページ

更新フォームは、cfform タグまたは HTML フォームタグを使用して作成できます。更新フォームでは、更新アクションページが呼び出されます。更新アクションページでは、cfupdate タグ、または SQL の UPDATE ステートメントを含んだ cfquery タグを使用できます。更新アクションページでは、エンドユーザーに確認メッセージを表示する必要もあります。

更新フォームの作成

更新フォームと挿入フォームの主な相違点は次のとおりです。

- 更新フォームでは、更新するレコードのプライマリキーを参照します。

  プライマリキーとは、データベーステーブルでレコードを一意に識別するために使用されているフィールドのことです。たとえば、従業員名と住所のテーブルでは、Emp_ID のみが各レコードに固有のフィールドです。

- 更新フォームでは、既存のレコードデータが表示されます。

更新フォームでプライマリキーを指定する最も簡単な方法は、更新するレコードのプライマリキーの値を持つ非表示入力フィールドを使用することです。この非表示フィールドで、更新するレコードを ColdFusion に指示します。

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>Update Form</title>
</head>

<body>
<cfquery name="GetRecordtoUpdate" datasource="cfdocexamples">
    SELECT * FROM Employee
    WHERE Emp_ID = #URL.Emp_ID#
</cfquery>


<cfoutput query="GetRecordtoUpdate">
<table>
<form action="update_action.cfm" method="Post">
    <input type="Hidden" name="Emp_ID" value="#Emp_ID#"><br>
<tr>
    <td>First Name:</td>
    <td><input type="text" name="FirstName" value="#FirstName#"></td>
</tr>
<tr>
    <td>Last Name:</td>
    <td><input type="text" name="LastName" value="#LastName#"></td>
</tr>
<tr>
```

```
    <td>Department Number:</td>
    <td><input type="text" name="Dept_ID" value="#Dept_ID#"></td>
</tr>
<tr>
    <td>Start Date:</td>
    <td><input type="text" name="StartDate" value="#StartDate#"></td>
</tr>
<tr>
    <td>Salary:</td>
    <td><input type="text" name="Salary" value="#Salary#"></td>
</tr>
<tr>
    <td>Contractor:</td>
    <td><cfif #Contract# IS "Yes">
        <input type="checkbox" name="Contract" checked>Yes
    <cfelse>
        <input type="checkbox" name="Contract">Yes
    </cfif></td>
</tr>
<tr>
    <td> </td>
    <td><input type="Submit" value="Update Information"></td>
</tr>
</form>
</table>
</cfoutput>

</body>
</html>
```

2 このファイルに "update_form.cfm" という名前を付けて保存します。

3 ページ URL および従業員 ID を指定して、"update_form.cfm" を Web ブラウザで表示します。たとえば、**http://localhost/myapps/update_form.cfm?Emp_ID=3** のように入力します。

注意：このページでは従業員の情報を表示できますが、データベースを更新するにはアクションページを作成します。詳細については、427 ページの「データを更新するアクションページの作成」を参照してください。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfquery name="GetRecordtoUpdate" datasource="cfdocexamples"> SELECT * FROM Employee WHERE Emp_ID = #URL.Emp_ID# </cfquery>`	cfdocexamples データソースにクエリーを実行して、このページを呼び出した URL で指定されている従業員 ID に一致するレコードを返します。
`<cfoutput query="GetRecordtoUpdate"> ... </cfoutput>`	GetRecordtoUpdate クエリーの結果を、この後のフォームで変数として使用できるようにします。
`<form action="update_action.cfm" method="Post"> ... </form>`	"update_action.cfm" アクションページに渡す変数を設定するフォームを作成します。

コード	説明
`<input type="Hidden" name="Emp_ID" value="#Emp_ID#"> `	非表示入力フィールドを使用して、Emp_ID (プライマリキー) の値をアクションページに渡します。
`First Name:` `<input type="text" name="FirstName" value="#FirstName#"> ` `Last Name:` `<input type="text" name="LastName" value="#LastName#"> ` `Department Number:` `<input type="text" name="Dept_ID" value="#Dept_ID#"> ` `Start Date:` `<input type="text" name="StartDate" value="#StartDate#"> ` `Salary:` `<input type="text" name="Salary" value="#Salary#"> `	更新フォームのフィールドにデータを設定します。この例では、ColdFusion の形式設定関数を使用していません。したがって、開始日は 1985-03-12 00:00:00 のように表示され、給与には通貨記号やカンマが追加されません。ユーザーは、有効なデータ入力方法を利用して任意のフィールドの情報を変更できます。
`Contracto r:` `< cfif #Contract# IS "Yes">` `<input type="checkbox"  name="C ontract" checked>Yes ` `<cfelse>` `<input type="checkbox" name="Contract"> Yes  ` `</cfif>` ` ` `<input type="Submit" value="Update Information">` `</form>` `</cfoutput>`	[Contract] フィールドではチェックボックスを表示して値を設定するので、特殊な処理が必要です。cfif 構文を使用して、[Contract] フィールドの値が Yes であればチェックボックスにチェックを付けます。Yes でなければ無効のままにします。

データを更新するアクションページの作成

cfupdate タグ、または UPDATE ステートメントを含んだ cfquery タグを使用して、データを更新するためのアクションページを作成できます。

cfupdate による更新アクションページの作成

cfupdate タグは、フロントエンドフォームのデータに基づいて単純な更新を行う場合に便利です。cfupdate タグのシンタックスは、cfinsert タグとほとんど同じです。

cfupdate タグを使用するには、フォームから送信するデータにプライマリキーフィールドを含めます。cfupdate タグによって、更新するテーブルのプライマリキーフィールドが自動的に検出されると、受信したフォームフィールドの中からプライマリキーフィールドが検索され、それに基づいて更新するレコードが選択されます (したがって、プライマリキー自体を更新することはできません)。その後、残りのフォームフィールドを使用して、レコード内の対応するフィールドが更新されます。フォームに必要なフィールドは、変更するデータベースフィールドに対応するフィールドのみです。

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Update Employee</title>
</head>
<body>
<cfif not isdefined("Form.Contract")>
    <cfset form.contract = "N">
<cfelse>
    <cfset form.contract = "Y">
</cfif>

<cfupdate datasource="cfdocexamples" tablename="EMPLOYEE">

<h1>Employee Updated</h1>
<cfoutput>
You have updated the information for #Form.FirstName# #Form.LastName# in the employee
    database.
</cfoutput>

</body>
</html>
```

2 このページに "update_action.cfm" という名前を付けて保存します。

3 ページ URL および従業員 ID を指定して、"update_form.cfm" を Web ブラウザで表示します。たとえば、**http://localhost/myapps/update_form.cfm?Emp_ID=3** のように入力します。

4 任意のフィールドに新しい値を入力して、[Update Information] をクリックします。

Employee テーブルのレコードが新しい値に更新され、確認メッセージが表示されます。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明

`<cfif not isdefined("Form.Contract")>`
`<cfset Form.contract = "N">`
`<cfelse>`
`<cfset form.contract = "Y">`
`</cfif>`
	Form.Contract の値が定義されていない場合は値を No に設定し、定義されていれば Yes に設定します。[Contractor] チェックボックスがチェックされていない場合は、アクションページにデータが渡されませんが、データベースフィールドには何らかの値が必要です。

`<cfupdate datasource="cfdocexamples"` `tablename="EMPLOYEE">`	フォームのプライマリキー (Emp_ID) に一致するデータベースレコードを更新します。フォームコントロールと同じ名前を持つレコードフィールドがすべて更新されます。

`<cfoutput>`
`You have updated the information for #Form.FirstName#`
`#Form.LastName# in the employee database.`
`</cfoutput>`
	変更が行われたことをユーザーに通知します。

cfquery による更新アクションページの作成

複雑な更新を行うには、cfupdate タグを使用する代わりに、cfquery タグ内で SQL の UPDATE ステートメントを使用します。SQL の UPDATE ステートメントでは、複雑な更新をより柔軟に行うことができます。

次の手順では、427 ページの「cfupdate による更新アクションページの作成」に従って "update_action.cfm" ページが既に作成されていることを前提としています。

1 "update_action.cfm" の cfupdate タグを、次の cfquery のコード (太字部分) に置き換えます。

```
<html>
<head>
    <title>Update Employee</title>
</head>
<body>
<cfif not isdefined("Form.Contract")>
    <cfset form.contract = "No">
<cfelse>
    <cfset form.contract = "Yes">
</cfif>

<!--- cfquery requires date formatting when retrieving from
Access. Use the left function when setting StartDate to trim
the ".0" from the date when it first appears from the
Access database --->
    <cfquery name="UpdateEmployee" datasource="cfdocexamples">
    UPDATE Employee
    SET FirstName = '#Form.Firstname#',
        LastName = '#Form.LastName#',
        Dept_ID = #Form.Dept_ID#,
        StartDate = '#left(Form.StartDate,19)#',
        Salary = #Form.Salary#
    WHERE Emp_ID = #Form.Emp_ID#
</cfquery>

<h1>Employee Updated</h1>
<cfoutput>
You have updated the information for
#Form.FirstName# #Form.LastName#
in the employee database.
</cfoutput>
</body>
</html>
```

2 このページを保存します。

3 ページ URL および従業員 ID を指定して、"update_form.cfm" を Web ブラウザで表示します。たとえば、**http://localhost/myapps/update_form.cfm?Emp_ID=3** のように入力します。

4 任意のフィールドに新しい値を入力して、[Update Information] をクリックします。

Employee テーブルのレコードが新しい値に更新され、確認メッセージが表示されます。

cfquery タグによって、Microsoft Access データベースから日付情報が取得されると、次のように日付と 0.1 秒単位までの時刻が表示されます。

データの削除

データベースからデータを削除するには、cfquery タグで SQL の DELETE ステートメントを使用します。ColdFusion に cfdelete タグはありません。

単一のレコードの削除

単一のレコードを削除するには、SQL の DELETE ステートメントの WHERE 条件にテーブルのプライマリキーを指定します。次の手順では、Emp_ID がプライマリキーです。SQL の DELETE ステートメントは次のようになります。

```
DELETE FROM Employee WHERE Emp_ID = #Form.Emp_ID#
```

削除するデータを画面上で確認してから削除したいことがあります。次の手順では、データの挿入や更新に使用したフォームページを再利用して、削除するデータを表示します。送信前にフォームに入力したデータは使用されないので、削除するレコードをテーブルに表示できます。

1 "update_form.cfm" で、タイトルを「Delete Form」に変更し、送信ボタンのテキストを「Delete Record」に変更します。

2 次のように form タグを編集します。

```
<form action="delete_action.cfm" method="Post">
```

3 変更したファイルに "delete_form.cfm" という名前を付けて保存します。

4 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>Delete Employee Record</title>
</head>
<body>

<cfquery name="DeleteEmployee"
    datasource="cfdocexamples">
        DELETE FROM Employee
        WHERE Emp_ID = #Form.Emp_ID#
</cfquery>

<h1>The employee record has been deleted.</h1>
<cfoutput>
You have deleted #Form.FirstName# #Form.LastName# from the employee database.
</cfoutput>
</body>
</html>
```

5 このページに "delete_action.cfm" という名前を付けて保存します。

6 ページ URL および従業員 ID を指定して、"delete_form.cfm" を Web ブラウザで表示します。たとえば、**http://localhost/myapps/delete_form.cfm?Emp_ID=3.Click Delete Record** のように入力します。

Employee テーブルのレコードが削除され、確認メッセージが表示されます。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfquery name="DeleteEmployee" datasource="cfdocexamples"> DELETE FROM Employee WHERE Emp_ID = #Form.Emp_ID# </cfquery>`	フォームの Emp_ID (非表示) フィールドに一致する Emp_ID 列を持つレコードが、データベースから削除されます。Emp_ID はテーブルのプライマリキーなので、1 つのレコードのみが削除されます。
`<cfoutput> You have deleted #Form.FirstName# #Form.LastName# from the employee database. </cfoutput>`	レコードが削除されたことをユーザーに通知します。

複数のレコードの削除

SQL の条件を使用して複数のレコードを削除できます。次の例では、販売部門 (Dept_ID が 4) の全従業員のレコードを Employee テーブルから削除しています。

```
DELETE FROM Employee WHERE Dept_ID = 4
```

Employee テーブルからすべてのレコードを削除するには、次のコードを使用します。

```
DELETE FROM Employee
```

重要：データベースからレコードを削除すると、元に戻すことはできません。DELETE ステートメントは慎重に使用してください。

ColdFusion 10 でのデータベース関連の機能強化

- クライアント情報の追跡
- 新しい SQL 型のサポート
- その他の機能強化（接続の検証および例外処理の強化）

クライアント情報の追跡

データベース操作の実行中にデータベースの監査を行えるように、特定のクライアント情報（アプリケーション名やクライアント ID など）を追跡することができます。サポートされるクライアント情報は、データベースによって異なります。例えば、Oracle の場合、クライアント情報は $v_session という名前の表でサポートされます。その他のデータベースでは、それはローカルキャッシュまたはレジスタに保存されます。

ColdFusion タグの cfquery、cfupdate、cfinsert および cfstoredproc の ClientInfo 属性を使用して、クライアント情報を送信できます。送信される情報は、クエリが実行される前に設定されます。

ClientInfo 属性でサポートされる値は次のとおりです。

- AccountingInfo
- Action：クエリで実行されるアクションです。
- ApplicationName：アプリケーション名です。
- ClientHostName：クエリが実行されるホストです。
- ClientID：クライアント ID です。
- ClientUser：ユーザー ID です。
- ProgramID：プログラム ID です。
- Module：モジュール名です。

サポートされている clientInfo プロパティを確認するには、cfdbinfo タグを使用します。

例

```
<cfscript>
    clientInfo = structNew();
    clientInfo.AccountingInfo = "MyAccount_cfquery";
    clientInfo.Action = "cfstoredproc_cfquery";
    clientInfo.ApplicationName = "testApp_cfquery";
    clientInfo.ClientHostName = "Testserver_cfquery";
    clientInfo.ClientID = "testID_cfquery";
    clientInfo.ClientUser = "cfadmin_cfquery";
    clientInfo.ProgramID = 1234;
    clientInfo.Module = "test_query";
</cfscript>
<cfquery name="qName"  datasource="#regdatasource#" clientInfo="#clientInfo#">
Select * from employees
</cfquery>
```

クライアント情報のメタデータへのアクセス

cfdbinfo タグでは、type 属性の値に ClientInfo がサポートされます。この値を指定した場合、指定したデータソースでサポートされているメタデータが返されます。

```
<CFDBINFO type="clientinfo" datasource="#regdatasource#" name="result">
<cfdump var="#result#">
```

クライアント情報へのアクセス

fetchclientinfo 属性を使用してデータベース操作を実行すると、データベース固有のクライアント情報にアクセスできます。fetchclientinfo 属性がサポートされる ColdFusion タグは、cfquery、cfinsert および cfstoredproc です。true に設定した場合、最後のクエリによって渡された、キーと値のペアを持つ構造体が返されます。

例

```
<cfquery fetchclientinfo=true result="resultQ" datasource="#regdatasource#">
select * from employees
</cfquery>
```

ColdFusion Administrator を使用したクライアント情報の引き渡し

ColdFusion Administrator のデータソースの詳細設定ページを使用すると、データベース固有の情報を設定できます。指定した場合は、クエリが実行される前に、次の情報がデータベースに送信されます。

- クライアントホスト名：クエリが実行されるホスト名です。
- クライアントユーザー名：ユーザーが <cflogin> タグを使用してログインしている場合のユーザー名です。
- アプリケーション名：application.cfc で指定されているアプリケーション名です。
- 接頭辞：指定した場合、値は application.cfc で指定されているアプリケーション名の前に付加されます。

注意：クエリタグで同じクライアント情報プロパティが指定された場合は、その指定がサーバーレベルの設定よりも優先されます。

CFSQLType の新しいデータ型のサポート

cfqueryparam および cfprocparam タグでは、次の SQL 型がサポートされます。

- CF_SQL_NCHAR
- CF_SQL_NVARCHAR
- CF_SQL_LONGNVARCHAR
- CF_SQL_NCLOB
- CF_SQL_SQLXML

その他の機能強化

その他の機能強化は次のとおりです。

クエリ実行前の接続の検証

ColdFusion Administrator のデータソースの詳細設定ページで、接続を検証するオプションを設定できます。このオプションを設定した場合は、クエリが実行される前にデータベース接続が検証されます。このオプションを設定すると、アプリケーションのパフォーマンスに影響することがあります。

必要に応じて、接続を検証するクエリを指定することもできます。指定した場合、それに基づいて接続が検証されます。指定しない場合は、デフォルトのメカニズムを使用して接続が検証されます。

例外処理の改善

<cfcatch> タグの type=database プロパティが改善され、例外処理が向上しました。

#CFCATCH.exceptions# で、詳細が構造体のリストで提供されます。複数の例外が発生した場合は、複数の要素が提供されます。各要素により、クラス、メッセージ、および原因のリスト（存在する場合）のカテゴリの情報が提供されます。

例

```
<cftry>
    <cfquery datasource="badmysql" timeout="2">
          Select * from employees
    </cfquery>
<cfcatch type="database">
    <cfdump var="#cfcatch#">
</cfcatch>
</cftry>
```

クエリーオブクエリーの使用

レコードセットからデータを取得するクエリーを、クエリーオブクエリーと呼びます。クエリーオブクエリーを使用すれば、生成されたレコードセットをデータベーステーブルと見なして、クエリーを実行できます。

レコードセットについて

クエリーオブクエリーは、cfquery タグや他の方法で作成されたレコードセットを操作する場合に使用します。

データベースにクエリーを実行すると、データが含まれたレコードセットが取得されます。このレコードセットは、ユーザーに表示する用途だけでなく、アプリケーションのパフォーマンスを向上させる用途にも利用できます。

レコードセットには行 (レコード) と列 (フィールド) が含まれているので、一種のデータベーステーブルまたはスプレッドシートと考えることができます。たとえば、cfpop タグで取得されるレコードセットでは、メッセージが行であり、宛先、送信元、件名などのメッセージコンポーネントが列であると考えることができます。

レコードセットの作成

クエリーオブクエリーは、レコードセットを生成する次のような任意の ColdFusion タグまたは関数で実行できます。

- cfcollection
- cfdirectory
- cfftp
- cfhttp
- cfindex
- cfldap
- cfmail
- cfpop
- cfprocresult
- cfquery (データベースまたは別のクエリーオブクエリーに対して実行する場合)
- cfsearch
- cfstoredproc
- cfwddx
- QueryNew 関数

QueryNew() 関数によるレコードセットの作成

レコードセットは、cfquery または他の CFML タグを使用する以外に、QueryNew 関数でも作成できます。

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>The queryNew function</title>
</head>
<body>
<h2>QueryNew Example</h2>

<!--- Create a query, specify data types for each column. --->
<cfset qInstruments = queryNew("name, instrument, years_playing",
    "CF_SQL_VARCHAR, CF_SQL_VARCHAR, CF_SQL_INTEGER")>

<!--- Add rows. --->
<cfset newrow = queryaddrow(qInstruments, 3)>

<!--- Set values in cells. --->
<cfset temp = querysetcell(qInstruments, "name", "Thor", 1)>
<cfset temp = querysetcell(qInstruments, "instrument", "hammer", 1)>
<cfset temp = querysetcell(qInstruments, "years_playing", "1000", 1)>

<cfset temp = querysetcell(qInstruments, "name", "Bjorn", 2)>
<cfset temp = querysetcell(qInstruments, "instrument", "sitar", 2)>
<cfset temp = querysetcell(qInstruments, "years_playing", "24", 2)>

<cfset temp = querysetcell(qInstruments, "name", "Raoul", 3)>
<cfset temp = querysetcell(qInstruments, "instrument", "flute", 3)>
<cfset temp = querysetcell(qInstruments, "years_playing", "12", 3)>

<!--- Output the query. --->
<cfoutput query="qInstruments">
    <pre>#name##instrument# #years_playing#</pre>
</cfoutput>

<h3>Individual record retrieval:</h3>
<cfoutput>
<p>#qInstruments.name[2]# has played #qInstruments.instrument[2]# for
    #qInstruments.years_playing[2]# years.</p>
</cfoutput>

</body>
</html>
```

2 このページに "queryNew.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

3 ブラウザで "queryNew.cfm" を表示します。

クエリーオブクエリーについて

タグまたは関数でレコードセットを作成したら、従属クエリーを実行してそのレコードセットからデータを取得できます。レコードセットからデータを取得するクエリーを、クエリーオブクエリーと呼びます。クエリーオブクエリーを使用する状況としては、最初のクエリーでテーブル全体をメモリに読み込んだ後に、そのテーブルデータ (レコードセット) に対してソートクエリーやフィルタクエリーを実行する場合などがあります。つまり、レコードセットをデータベーステーブルと見なしてクエリーを実行します。

注意 : レコードセットは cfquery タグを使用しなくても生成できるので、クエリーオブクエリーはメモリ内クエリーとも呼ばれます。

クエリーオブクエリーの利点

クエリーオブクエリーには、次のような多くの利点があります。

1 同じテーブルに何度もアクセスする必要がある場合、データが既にメモリ (レコードセット) に保管されているので、アクセス時間が大幅に短縮されます。

クエリーオブクエリーは、5,000 ～ 50,000 行のテーブルに適しており、ColdFusion のホストコンピュータのメモリによる制限を受けます。

2 複数のデータソースから得たクエリー結果を結合できます。

たとえば、複数のデータベースから得たクエリー結果を結合 (union) して、メーリングリストの重複データを排除できます。

3 キャッシュされたクエリー結果は、さまざまな方法で効率的に処理できます。データベースに 1 回だけクエリーを実行し、その結果を使用して複数の要約テーブルを作成することができます。

たとえば、支払給与を部門別、技能別、職種別に調べる場合は、データベースに 1 回だけクエリーを実行し、その結果を 3 つの異なるクエリーで使用して、それぞれの要約データを生成できます。

4 詳細データを得るときに、データベースにアクセスしなくても、詳細情報やマスタの詳細情報を得ることができます。

たとえば、部門と従業員に関するデータをクエリーで取得し、その結果をキャッシュして、画面上に従業員名を表示します。ユーザーがアプリケーションで従業員を選択したら、データベースにアクセスせずに、キャッシュに格納されているクエリー結果からその従業員の詳細データを表示できます。

5 レポート定義でクエリーオブクエリーを使用して、サブレポートデータを生成できます。詳細については、1005 ページの「サブレポートの使用」を参照してください。

クエリーオブクエリーの実行

クエリーオブクエリーを実行するには、次の手順を行います。

1 マスタクエリーを実行してレコードセットを生成します。

マスタクエリーは、レコードセットを作成するタグまたは関数を使用して記述できます。詳細については、433 ページの「レコードセットの作成」を参照してください。

2 詳細クエリー (dbtype="query" を指定した cfquery タグ) を記述します。

3 詳細クエリーの中に、関連するレコードを取得する SQL ステートメントを記述します。SQL コードのテーブル名には、既存のクエリーの名前を指定します。datasource 属性は指定しないでください。

4 データベースの内容が頻繁に変更されない場合は、マスタクエリーの cachedwithin 属性を使用して、クエリー結果をキャッシュします。これは、複数のページリクエストにまたがって保持されます。このようにすると、最初のページリクエストでデータベースにアクセスした後は、一定の時間が経過するまでデータベースにはクエリーされなくなります。cachedwithin 属性の値 (日数、時間、分、秒の形式) を指定するには、CreateTimeSpan 関数を使用します。

詳細クエリーによって、新しいクエリー結果セットが生成されます。この結果セットは、詳細クエリーの name 属性の値によって識別されます。次の例では、マスタクエリーと、そのマスタクエリーから情報を抽出する 1 つの詳細クエリーを使用する方法を示します。

クエリーでのクエリー結果の使用

1 次の内容の ColdFusion ページを作成します。

```
 <h1>Employee List</h1>
<!--- LastNameSearch (normally generated interactively) --->
<cfset LastNameSearch="Doe">

<!--- Master Query --->
<cfquery datasource="cfdocexamples" name="master"
    cachedwithin=#CreateTimeSpan(0,1,0,0)#>
    SELECT * from Employee
</cfquery>

<!--- Detail Query (dbtype=query, no data source) --->
<cfquery dbtype="query" name="detail">
    SELECT Emp_ID, FirstName, LastName
    FROM master
    WHERE LastName=<cfqueryparam value="#LastNameSearch#"
cfsqltype="cf_sql_char" maxLength="20"></cfquery>

<!--- output the detail query results --->
<p>Output using a query of query:</p>
<cfoutput query=detail>
    #Emp_ID#: #FirstName# #LastName#<br>
</cfoutput>

<p>Columns in the master query:</p>
<cfoutput>
    #master.columnlist#<br>
</cfoutput>

<p>Columns in the detail query:</p>
<cfoutput>
    #detail.columnlist#<br>
</cfoutput>
```

2 このページに "query_of_query.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

3 ブラウザで "query_of_query.cfm" を表示します。

コードの説明

マスタクエリーによって、cfdocexamples データソースの Employee テーブル全体が取得されます。詳細クエリーによって、特定の姓を持つ従業員に関する 3 つの列のみが表示されます。このコードおよびその機能について、次の表で説明します。

コード	説明
`cfset LastNameSearch="Doe"`	詳細クエリーで選択する姓を設定します。実際のアプリケーションでは、この情報はユーザーの入力から取得します。
`<cfquery datasource="cfdocexamples" name="master" cachedwithin=#CreateTimeSpan(0,1,0,0)#> SELECT * from Employee </cfquery>`	cfdocexamples データソースに対してクエリーを実行し、Employees テーブルのすべてのデータを選択します。クエリーデータはキャッシュされます。これは、このページに対する複数のリクエストにまたがって保持されます。キャッシュされてから 1 時間経過するまで、データベースへのクエリーは実行されなくなります。
`<cfquery dbtype="query" name="detail"> SELECT Emp_ID, FirstName, LastName FROM master WHERE LastName=<cfqueryparam value="#LastNameSearch#" cfsqltype="cf_sql_char" maxLength="20"> </cfquery>`	detail という名前の新しいクエリーで、マスタクエリーをデータソースとして使用します。この新しいクエリーでは、LastNameSearch 変数の値と同じ姓を持つエントリのみを選択しています。また、従業員 ID、名、および姓の 3 つの列のデータのみを選択しています。ここでは、エラーを招くコードや危険なコードが実行されないように、cfqueryparam タグを使用しています。

コード	説明
<cfoutput query=detail> #Emp_ID#: #FirstName# #LastName# </cfoutput>	詳細クエリーで、従業員 ID、名、および姓のリストを表示します。
<cfoutput> #master.columnlist# </cfoutput>	マスタクエリーによって返されたすべての列を表示します。
<cfoutput> #detail.columnlist# </cfoutput>	詳細クエリーによって返されたすべての列を表示します。

レコードセットのデータを分割して表示する方法

データベースが大きい場合は、一度に表示する行数を制限できます。次の例に、クエリーオブクエリーの currentRow クエリー変数を使用して行数を制限する方法を示します。クエリー変数の詳細については、417 ページの「クエリー結果に関する情報の取得」を参照してください。

1 次の内容の ColdFusion ページを作成します。

```
 <html>
<head>
<title>QoQ with incremental row return</title>
</head>

<body>
<h3>QoQ with incremental row return</h3>
<!--- define startrow and maxrows to facilitate 'next N' style browsing --->
<cfparam name = "MaxRows" default = "5">
<cfparam name = "StartRow" default = "1">

<!--- master query: retrieve all info from Employee table --->
<cfquery name = "GetSals" datasource = "cfdocexamples">
    SELECT * FROM Employee
    ORDER BY LastName
</cfquery>

<!--- detail query: select 3 fields from the master query --->
<cfquery name = "GetSals2" dbtype = "query">
    SELECT FirstName, LastName, Salary
    FROM GetSals
    ORDER BY LastName
</cfquery>

<!--- build table to display output --->
<table cellpadding = 1 cellspacing = 1>
    <tr>
    <td bgcolor = f0f0f0>
    <b><i> </i></b>
    </td>

    <td bgcolor = f0f0f0>
    <b><i>FirstName</i></b>
    </td>

    <td bgcolor = f0f0f0>
    <b><i>LastName</i></b>
    </td>

    <td bgcolor = f0f0f0>
    <b><i>Salary</i></b>
    </td>
    </tr>
```

```
<!--- Output the query and define the startrow and maxrows
     parameters. Use the query variable currentRow to
     keep track of the row you are displaying. --->
    <cfoutput query = "GetSals2" startrow = "#StartRow#" maxrows = "#MaxRows#">
    <tr>
    <td valign = top bgcolor = ffffed>
        <b>#GetSals2.currentRow#</b>
    </td>

    <td valign = top>
        <font size = "-1">#FirstName#</font>
    </td>

    <td valign = top>
        <font size = "-1">#LastName#</font>
    </td>

    <td valign = top>
        <font size = "-1">#LSCurrencyFormat(Salary)#</font>
    </td>
    </tr>
    </cfoutput>
<!--- If the total number of records is less than or equal to
    the total number of rows, provide a link to the same page, with the
    StartRow value incremented by MaxRows (5, in this example) --->
    <tr>
    <td colspan = 4>
    <cfif (startrow + maxrows) lte getsals2.recordcount>
    <a href="qoq_next_row.cfm?startrow=<cfoutput>#Evaluate(StartRow +
        MaxRows)#</cfoutput>">See next <cfoutput>#MaxRows#</cfoutput>
        rows</a>
    </cfif>
    </td>
    </tr>
</table>
</body>
</html>
```

2 このページに "qoq_next_row.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

3 ブラウザで "qoq_next_row.cfm" を表示します。

cfdump タグによるクエリー結果の表示

CFML コードをデバッグする場合は、cfdump タグを使用すると、クエリーの内容をすばやく表示できます。このタグの形式は次のとおりです。

```
<cfdump var="#query_name#">
```

cfdump タグの詳細については、『CFML リファレンス』を参照してください。

SQL 以外のレコードセットへのクエリーオブクエリーの使用

クエリーオブクエリーは、cfquery の結果だけでなく、レコードセットを返すすべての CFML タグまたは関数に対して使用できます。cfdirectory タグ、cfsearch タグ、cfldap タグなどの、SQL 以外のレコードセットにもクエリーを実行できます。

クエリーオブクエリーのユーザーガイド

SQL やデータベース操作に慣れている方にはお馴染みのクエリーオブクエリー機能もいくつか含まれています。

ドット表記法の使用

クエリーオブクエリーでは、テーブル名のドット表記法がサポートされています。

例

A という構造体に B というフィールドが含まれており、その中に Products というテーブルが含まれている場合は、次のようなドット表記法でこのテーブルを参照できます。

```
SELECT tape_ID, length
FROM A.B.Products;
```

結合の使用

結合操作では、1 つの SELECT ステートメントを使用して、複数の関連するテーブルから 1 つの結果セットを返します。通常、それらのテーブルは、プライマリキーと外部キーの関係で関連付けられています。結合を実行する SQL 節は 2 つあります。

- **WHERE 節** : クエリーオブクエリーでは、WHERE 節による結合がサポートされています。
- **INNER JOIN および OUTER JOIN**: クエリーオブクエリーでは、INNER JOIN または OUTER JOIN 節による結合はサポートされていません。

注意：クエリーオブクエリーでサポートされているのは、2 つのテーブルの結合のみです。

UNION の使用

UNION 演算子を使用して、複数の SELECT 式の結果を 1 つのレコードセットに結合できます。結合するテーブルの列数が同じであり、対応する列のデータ型が UNION と互換性がある必要があります。次のいずれかの条件に該当する場合、列のデータ型は UNION と互換性があります。

- 同じデータ型である場合。たとえば両方とも Tinyint である場合。
- 両方ともが数値である場合。たとえば、Tinyint、Smallint、Integer、Bigint、Double、Float、Real、Decimal、Numeric など。
- 両方とも文字である場合。たとえば、Char、Varchar、LongVarchar など。
- 両方とも日付である場合。たとえば、Time、TimeStamp、Date など。

注意：クエリーオブクエリーでは、ODBC 形式の日付および時刻はサポートされていません。

シンタックス

```
select_expression = select_expression UNION [ALL] select_expression
```

例

次のテーブルがあるとします。

テーブル 1	
Type(int)	Name(varchar)
1	テニス
2	野球
3	サッカー

テーブル 2	
ID(int)	Sport(varchar)
3	サッカー
4	バレーボール
5	卓球

テーブル 1 とテーブル 2 を結合するには、次のように UNION ステートメントを使用します。

```
SELECT * FROM Table1
UNION
SELECT * FROM Table2
```

この UNION ステートメントによって、次の結果 (UNION) テーブルが作成されます。

結果テーブル	
Type(int)	Name(varchar)
1	テニス
2	野球
3	サッカー
4	バレーボール
5	卓球

列名のエイリアスの使用

UNION テーブルでの列名は、UNION 操作の最初の SELECT ステートメントの結果セットの列名になり、もう一方の SELECT ステートメントの列名は無視されます。結果テーブルの列名を変更するには、次のようにエイリアスを使用します。

```
Select Type as SportType, Name as SportName from Table1
UNION
Select * from Table2
```

行の重複と複数のテーブル

デフォルトでは、UNION 演算子によって結果テーブルから重複している行が削除されます。キーワード ALL を使用すると、重複している行が含まれるようになります。

次の例のように、UNION 演算子を使用すると無制限にテーブルを結合できます。

```
Select * from Table1
UNION
Select * from Table2
UNION
Select * from Table3
...
```

括弧と評価の順序

デフォルトでは、クエリーオブクエリーの SQL エンジンは、UNION 演算子を含んでいるステートメントを左から右へ評価します。評価順序を変更するには、括弧を使用します。たとえば、次の 2 つのステートメントは異なります。

```
 /* First statement. */
SELECT * FROM TableA
UNION ALL
(SELECT * FROM TableB
    UNION
    SELECT * FROM TableC
)

/* Second statement. */
(SELECT * FROM TableA
    UNION ALL
    SELECT * FROM TableB
)
UNION
SELECT * FROM TableC
```

ステートメント 1 では、テーブル B とテーブル C の結合で重複が除外されます。次に、このセットとテーブル A の結合で、キーワード ALL によって重複が保持されます。ステートメント 2 では、テーブル A とテーブル B の結合で重複が保持されますが、後続のテーブル C との結合で重複が除外されます。結果として、キーワード ALL の効果はなくなっています。

UNION での他のキーワードの使用

UNION を実行する場合、個々の SELECT ステートメントで ORDER BY 節または COMPUTE 節を指定することはできません。最後の SELECT ステートメントの後ろに ORDER BY または COMPUTE 節を 1 つだけ入れることができます。この節は、結合された最後の結果セットに適用されます。それぞれの SELECT ステートメントには、GROUP BY および HAVING の式のみを指定できます。

条件演算子の使用

クエリーオブクエリーでは、SQL ステートメントで次の条件演算子を使用できます。

テスト条件演算子

この条件演算子は、ブール式が True、False、または Unknown のいずれであるかをテストします。

シンタックス

```
 cond_test ::= expression [IS [NOT] {TRUE | FALSE | UNKNOWN} ]
```

例

```
 SELECT _isValid FROM Chemicals
WHERE _isValid IS true;
```

null 条件演算子

この条件演算子は、式が null かどうかをテストします。

シンタックス

```
 null_cond ::= expression IS [NOT] NULL
```

例

```
 SELECT bloodVal FROM Standards
WHERE bloodVal IS NOT null;
```

比較条件演算子

この条件演算子は、ある式と、それと同じデータ型 (数値、文字列、日付、またはブール値) を持つ別の式を比較します。レコードセットの必要な行のみを選択的に取得する場合に使用できます。

シンタックス

```
comparison_cond ::= expression [> | >= | <> | != | < | <=] expression
```

例

次の例では、比較条件演算子を使用して、IQ が 150 以上の犬のみを取得します。

```
SELECT dog_name, dog_IQ
FROM Dogs
WHERE dog_IQ >= 150;
```

BETWEEN 条件演算子

この条件演算子は、式と式を比較します。レコードセットの必要な行のみを選択的に取得する場合に使用できます。比較条件演算子と同様に、BETWEEN 条件演算子も比較を行います。ただし、BETWEEN 条件演算子は、特定の範囲の値との比較を行います。したがって、そのシンタックスには、範囲を表す最小値と最大値の 2 つの値が必要です (それらの値も範囲に含まれます)。これらの値はキーワード AND で区切ります。

シンタックス

```
between_cond ::= expression [NOT] BETWEEN expression AND expression
```

例

次の例では、BETWEEN 条件演算子を使用して、IQ が 150 以上 165 以下の犬のみを取得します。

```
SELECT dog_name, dog_IQ
FROM Dogs
WHERE dog_IQ BETWEEN 150 AND 165;
```

IN 条件演算子

この条件演算子を使用すると、一致条件のカンマ区切りのリストを指定できます。これは、OR 条件演算子と似ています。IN 条件演算子は、長いリストを使用する場合にコードが読みやすくなるだけでなく、別の SELECT ステートメントを含めることもできます。

シンタックス

```
in_cond ::= expression [NOT] IN (expression_list)
```

例

次の例では、IN 条件演算子を使用して、Ken's Kennels または Barb's Breeders のいずれかで誕生した犬のみを取得します。

```
SELECT dog_name, dog_IQ, Kennel_ID
FROM Dogs
WHERE kennel_ID IN ('Kens','Barbs');
```

LIKE 条件演算子

この条件演算子を使用すると、検索パターンとデータを比較する、ワイルドカード検索を実行できます。LIKE 条件演算子は、データを部分的に不明な値と比較する点で、BETWEEN、IN などの他の条件演算子とは異なります。

シンタックス

```
like_cond ::= left_string_exp [NOT] LIKE right_string_exp [ESCAPE escape_char]
```

left_string_exp には、特定の文字列、または文字列の列への参照を指定できます。right_string_exp には、文字列の列への参照、または検索パターンを指定できます。検索パターンは、リテラルテキストと少なくとも 1 つのワイルドカード文字から構成される検索条件です。ワイルドカード文字は、検索パターンの不明な部分を表す特殊文字で、次のように解釈されます。

- アンダースコア (_) は任意の 1 文字を表します。
- パーセント記号 (%) は、0 個以上の文字を表します。
- 角括弧 ([ ]) はその範囲内の任意の文字を表します。
- キャレットが入っている角括弧 [^] は、その範囲にない任意の文字を表します。
- 他のすべての文字は、その文字自体を表します。

注意：以前のバージョンの ColdFusion では、括弧で囲まれた範囲はサポートされていません。

例

次の例では、LIKE 条件演算子を使用して、Boston Terrier、Jack Russell Terrier、Scottish Terrier などの、テリア種の犬のみを取得します。

```
SELECT dog_name, dog_IQ, breed
FROM Dogs
WHERE breed LIKE '%Terrier';
```

次の例は、括弧で囲まれた範囲を使用する SELECT ステートメントです。

```
SELECT lname FROM Suspects WHERE lname LIKE 'A[^c]%';
SELECT lname FROM Suspects WHERE lname LIKE '[a-m]%';
SELECT lname FROM Suspects WHERE lname LIKE '%[]';
SELECT lname FROM Suspects WHERE lname LIKE 'A[%]%';
SELECT lname FROM Suspects WHERE lname LIKE 'A[^c-f]%';
```

大文字と小文字の区別

ColdFusion の通常のルールとは異なり、クエリーオブクエリーでは大文字と小文字が区別されます。ただし、クエリーオブクエリーでは 2 つの文字列関数 UPPER() と LOWER() がサポートされているので、これらを使用すれば大文字と小文字を区別せずに検索を行えます。

例

次の例では、'Sylvester' のみが検索されます。

```
SELECT dog_name
FROM Dogs
WHERE dog_name LIKE 'Sylvester';
```

次の例では大文字と小文字が区別されません。LOWER() 関数を使用して、'Sylvester'、'sylvester'、'SYLVESTER' などをすべて小文字に変換した上で、すべて小文字の文字列 'sylvester' と比較します。

```
SELECT dog_name
FROM Dogs
WHERE LOWER(dog_name) LIKE 'sylvester';
```

LIKE 条件の右側に変数を使用している場合に、大文字と小文字を区別せずに比較するには、LCase 関数または UCase 関数を使用して、変数のすべてのテキストを大文字または小文字に揃えます。次に例を示します。

```
WHERE LOWER(dog_name) LIKE '#LCase(FORM.SearchString)#';
```

ワイルドカードのエスケープ

条件演算子の ESCAPE 節を使用して、独自のエスケープ文字を指定できます。

例

次の例では、ESCAPE 節を使用して、通常ワイルドカード文字として解釈されるパーセント記号 (%) を文字として検索できるようにしています。

```
SELECT emp_discount
FROM Benefits
WHERE emp_discount LIKE '10\%'
ESCAPE '\';
```

列のデータ型の管理

クエリーオブクエリーでは、列のデータ型を定義するメタデータが各列に設定されている必要があります。ColdFusion で作成されるすべてのクエリーにはメタデータが設定されます。ただし、QueryNew 関数で 2 番目のパラメータを省略した場合、作成されるクエリーにはメタデータが含まれません。このオプションの 2 番目のパラメータを使用して、クエリーの各列のデータ型を定義します。

QueryNew 関数での列のデータ型の指定

❖ QueryNew 関数を入力します。次の例に示すとおり、最初のパラメータには列名を、2 番目のパラメータにはデータ型を指定します。

```
<cfset qInstruments = queryNew("name, instrument, years_playing", "CF_SQL_VARCHAR, CF_SQL_VARCHAR,
CF_SQL_INTEGER")>
```

注意：クエリーオブクエリーのメタデータを確認するには、GetMetaData 関数を使用します。

QueryAddColumn 関数での列のデータ型の指定

1 パラメータを設定していない QueryNew 関数を指定して、クエリーを作成します。

```
<!--- Make a query. --->
<cfset myQuery = QueryNew("")>
```

2 QueryAddColumn 関数を使用して列を追加します。3 番目のパラメータでデータ型を指定します。

```
<!--- Create an array. --->
<cfset FastFoodArray = ArrayNew(1)>
<cfset FastFoodArray[1] = "French Fries">
<cfset FastFoodArray[2] = "Hot Dogs">
<cfset FastFoodArray[3] = "Fried Clams">
<cfset FastFoodArray[4] = "Thick Shakes">
<!--- Use the array to add a column to the query. --->
<cfset nColumnNumber = QueryAddColumn(myQuery, "FastFood", "CF_SQL_VARCHAR",
    FastFoodArray)>
```

データ型を指定しなかった場合は、条件式の実行時に各列の最初の 50 行がチェックされて、データ型が推定されます。

場合によっては、推定されたデータ型がアプリケーションで適切に機能しないことがあります。特に、WHERE 節や他の条件式でその列を使用している場合は、データ型が互換性を持っている必要があります。互換性がない場合は、CAST 関数を使用して、互換性のあるデータ型にいずれかの列を再キャストします。キャストの詳細については、444 ページの「CAST 関数の使用」を参照してください。データ型の互換性の詳細については、451 ページの「クエリーオブクエリーの処理について」を参照してください。

注意：QueryNew 関数でデータ型を指定すると、互換性に関する問題を回避できます。

CAST 関数の使用

場合によっては、列のデータ型が、実行したい処理に対して互換性のあるデータ型でないことがあります。たとえば、cfhttp タグによって返されるクエリー列は、すべて数値が含まれている場合でも、必ず CF_SQL_VARCHAR 型になります。この場合は、クエリーオブクエリーの CAST 関数を使用して、列の値を適切なデータ型に変換します。

CAST 関数のシンタックスは次のとおりです。

```
CAST ( expression AS castType )
```

castType には、次のいずれかが入ります。

- BINARY
- BIGINIT
- BIT
- DATE
- DECIMAL
- DOUBLE
- INTEGER
- TIME
- TIMESTAMP
- VARCHAR

次に例を示します。

```
<cfhttp url="http://quote.yahoo.com/download/quotes.csv?Symbols=csco,jnpr&format=sc1l1&ext=.csv"
    method="GET"
    name="qStockItems"
    columns="Symbol,Change,LastTradedPrice"
    textqualifier="""
    delimiter=","
    firstrowasheaders="no">
<cfoutput>
    <cfdump var="#qStockItems#">
    <cfdump var="#qStockItems.getColumnNames()#">
</cfoutput>

<cfoutput>
<cfloop index="i" from="1" to="#arrayLen(qStockItems.getColumnNames())#">
    #qStockItems.getMetaData().getColumnTypeName(javaCast("int",i))#<br/>
</cfloop>
</cfoutput>
<cftry>
    <cfquery name="hello" dbtype="query">
        SELECT SUM(CAST(qStockItems.LastTradedPrice as INTEGER))
        AS SUMNOW from qStockItems
    </cfquery>
    <cfcatch>Error in Query of Queries</cfcatch>
</cftry>

<cfoutput>
    <cfdump var="#hello#">
</cfoutput>
```

集計関数の使用

集計関数は、データのセットを処理して 1 つの値を返します。この関数は、テーブル全体を取得してからテーブル全体のレコードセットを処理するのとは対照的に、テーブルから要約情報を取得します。

次の操作を実行する場合は、集計関数の使用を検討してください。

- 列の平均を表示する
- 列の行数を数える
- 列内で最も早い日付を検出する

すべてのリレーショナルデータベース管理システム (RDBMS) ですべての集計関数がサポートされているわけではありません。データベースのマニュアルを参照してください。次の表に、クエリーオブクエリーでサポートされている集計関数をリストします。

関数	説明
AVG()	列の平均を返します。
COUNT()	列の行数を返します。
MAX()	列の最大値を返します。
MIN()	列の最小値を返します。
SUM()	列の合計値を返します。

シンタックス

```
aggregate_func ::= <COUNT>(* | column_name) | AVG | SUM | MIN | MAX)
([ALL | DISTINCT] numeric_exp)
```

例

次の例では、AVG() 関数を使用してすべてのテリアの IQ の平均を取得します。

```
SELECT dog_name, AVG(dog_IQ) AS avg_IQ
FROM Dogs
WHERE breed LIKE '%Terrier';
```

任意の式への集計関数の適用

クエリーオブクエリーでは次のように、任意の式に対して集計関数を適用できます。

```
SELECT lorange, count(lorange+hirange)
FROM roysched
GROUP BY lorange;
```

集計関数を含む式

クエリーオブクエリーでは次のように、集計関数を含む数式がサポートされています。

```
SELECT MIN(lorange) + MAX(hirange)
FROM roysched
GROUP BY lorange;
```

GROUP BY と HAVING 式の使用

クエリーオブクエリーでは、エイリアスを使用して参照すれば、任意の数式を使用できます。

例

次のコードは、正しいコードです。

```
SELECT (lorange + hirange)/2 AS midrange,
COUNT(*)
FROM roysched
GROUP BY midrange;
```

次のコードは、正しいコードです。

```
SELECT (lorange+hirange)/2 AS x,
COUNT(*)
FROM roysched GROUP BY x
HAVING x > 10000;
```

次のコードは、クエリーオブクエリーではサポートされません。

```
SELECT (lorange + hirange)/2 AS midrange,
COUNT(*)
FROM roysched
GROUP BY (lorange + hirange)/2;
```

ORDER BY 節の使用

クエリーオブクエリーでは、ORDER BY 節を使用したソートがサポートされています。これは、SELECT ステートメントの最後に配置する必要があります。複数の列、位置で指定した列、非選択の列を基準にしてソートできます。キーワード DESC を使用すると、降順のソートを指定できます (デフォルトでは、多くの RDBMS ソートは昇順なので、キーワード ASC は不要です)。

シンタックス

```
order_by_column ::= ( <IDENTIFIER> | <INTEGER_LITERAL> ) [<ASC> | <DESC>]
```

例

次の例は、ORDER BY 節を使用した簡単なソートです。

```
SELECT acetylcholine_levels, dopamine_levels
FROM results
ORDER BY dopamine_levels
```

次の例では、より複雑なソートを行っています。まずドーパミンレベルで昇順にソートしてから、アセチルコリンで降順にソートしています。キーワード ASC は不要です。これは、明示したい場合にのみ使用します。

```
SELECT acetylcholine_levels, dopamine_levels
FROM results
ORDER BY 2 ASC, 1 DESC
```

エイリアスの使用

クエリーオブクエリーでは、データベース列のエイリアスが使用できます。エイリアスとは、データベースフィールドまたは値に割り当てる別名です。エイリアスは、同じ SQL ステートメント内で再利用できます。

エイリアスの使用例としては、複数の列を結合して 1 つの値を生成する場合があります。たとえば、名と姓を結合して氏名の値を作成する場合があります。新しい値はデータベース内に存在しないので、エイリアスを割り当てて参照します。SELECT ステートメントでエイリアスを割り当てるには、AS キーワードを使用します。

例

ORDER BY 節、GROUP BY 節、および HAVING 節でエイリアスを使用できます。

注意：テーブル名にエイリアスを割り当てることはできません。

```
SELECT FirstName + ' ' + LastName AS fullname
from Employee;
```

以降の例は、次の 2 つのマスタクエリーに基づいています。

```
<cfquery name="employee" datasource="2pubs">
    SELECT * FROM employee
</cfquery>

<cfquery name="roysched" datasource="2pubs">
    SELECT * FROM roysched
</cfquery>
```

ORDER BY の例

```
<cfquery name="order_by" dbtype="query">
    SELECT (job_id || job_lvl)/2 AS job_value
    FROM employee
    ORDER BY job_value
</cfquery>
```

GROUP BY の例

```
<cfquery name="group_by" dbtype="query">
    SELECT lorange || hirange AS x, count(hirange)
    FROM roysched
    GROUP BY x
</cfquery>
```

HAVING の例

```
<cfquery name="having" dbtype="query">
    SELECT (lorange || hirange)/2 AS x,
    COUNT(*)
    FROM roysched GROUP BY x
    HAVING x > 10000
</cfquery>
```

null 値の処理

条件式はブール論理 (2 値論理) を使用して処理されますが、null 値を正しく処理するには 3 値論理を使用する必要があります。IS [NOT] NULL 節は、クエリーオブクエリーで正しく機能します。ただし、次の式は、breed 列が null の場合には正しく機能しません。

```
WHERE (breed > 'A')
WHERE NOT (breed > 'A')
```

正しく動作するには、いずれの式の結果セットにも null の breed 列が含まれていないことが必要です。この制限を避けるには、条件に明示的なルールを追加して、次のように変更します。

```
WHERE breed IS NOT NULL AND (breed > 'A')
WHERE breed IS NOT NULL AND not (breed > 'A')
```

文字列の連結

クエリーオブクエリーでは、次の例のように、+ および || の 2 つの文字列連結演算子をサポートしています。

```
LASTNAME + ', ' + FIRSTNAME
LASTNAME || ', ' || FIRSTNAME
```

予約語のエスケープ

ColdFusion には予約語があります。その多くは SQL 言語のキーワードであり、通常は列名やテーブル名として使用しません。列名やテーブル名で予約語をエスケープするには括弧で囲みます。

重要：以前のバージョンの ColdFusion では、エスケープしなくても一部の予約語が使用できます。

例

次に示す SELECT ステートメントの例は有効です。

```
SELECT [from] FROM parts;
SELECT [group].firstname FROM [group];
SELECT [group].[from] FROM [group];
```

次の例のようなネストされているエスケープはサポートされていません。

```
SELECT [[from]] FROM T;
```

次の表に、ColdFusion の予約語のリストを示します。

ABSOLUTE	ACTION	ADD	ALL	ALLOCATE
ALTER	AND	ANY	ARE	AS
ASC	ASSERTION	AT	AUTHORIZATION	AVG
BEGIN	BETWEEN	BIT	BIT_LENGTH	BOTH
BY	CASCADE	CASCADED	CASE	CAST
CATALOG	CHAR	CHARACTER	CHARACTER_LENGTH	CHAR_LENGTH
CHECK	CLOSE	COALESCE	COLLATE	COLLATION
COLUMN	COMMIT	CONNECT	CONNECTION	CONSTRAINT
CONSTRAINTS	CONTINUE	CONVERT	CORRESPONDING	COUNT
CREATE	CROSS	CURRENT	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR	DATE	DAY
DEALLOCATE	DEC	DECIMAL	DECLARE	DEFAULT
DEFERRABLE	DEFERRED	DELETE	DESC	DESCRIBE
DESCRIPTOR	DIAGNOSTICS	DISCONNECT	DISTINCT	DOMAIN
DOUBLE	DROP	ELSE	END	END-EXEC
ESCAPE	EXCEPT	EXCEPTION	EXEC	EXECUTE
EXISTS	EXTERNAL	EXTRACT	FALSE	FETCH
FIRST	FLOAT	FOR	FOREIGN	FOUND
FROM	FULL	GET	GLOBAL	GO
GOTO	GRANT	GROUP	HAVING	HOUR
IDENTITY	IMMEDIATE	IN	INDICATOR	INITIALLY
INNER	INPUT	INSENSITIVE	INSERT	INT
INTEGER	INTERSECT	INTERVAL	INTO	IS
ISOLATION	JOIN	KEY	LANGUAGE	LAST
LEADING	LEFT	LEVEL	LIKE	LOCAL
LOWER	MATCH	MAX	MIN	MINUTE
MODULE	MONTH	NAMES	NATIONAL	NATURAL
NCHAR	NEXT	NO	NOT	NULL
NULLIF	NUMERIC	OCTET_LENGTH	OF	ON
ONLY	OPEN	OPTION	OR	ORDER
OUTER	OUTPUT	OVERLAPS	PAD	PARTIAL
POSITION	PRECISION	PREPARE	PRESERVE	PRIMARY

PRIOR	PRIVILEGES	PROCEDURE	PUBLIC	READ
REAL	REFERENCES	RELATIVE	RESTRICT	REVOKE
RIGHT	ROLLBACK	ROWS	SCHEMA	SCROLL
SECOND	SECTION	SELECT	SESSION	SESSION_USER
SET	SMALLINT	SOME	SPACE	
SQL	SQLCODE	SQLERROR	SQLSTATE	SUBSTRING
SUM	SYSTEM_USER	TABLE	TEMPORARY	THEN
TIME	TIMESTAMP	TIMEZONE_HOUR	TIMEZONE_MINUTE	TO
TRAILING	TRANSACTION	TRANSLATE	TRANSLATION	TRIM
TRUE	UNION	UNIQUE	UNKNOWN	UPDATE
UPPER	USAGE	USER	USING	VALUE
VALUES	VARCHAR	VARYING	VIEW	WHEN
WHENEVER	WHERE	WITH	WORK	WRITE
YEAR	ZONE			

日付に対するクエリーオブクエリー

QueryNew 関数を使用してクエリーオブジェクトを作成し、列に日付定数を挿入した場合、その日付は、クエリーオブジェクトにクエリーオブクエリーを適用するまで、クエリーオブジェクトの内部で文字列として保存されます。クエリーオブジェクトにクエリーオブクエリーを適用すると、文字列表現が日付オブジェクトに変換されます。

クエリーオブクエリーでは、SQL および ODBC 形式の次の日付定数がサポートされています。

- **SQL 形式** : 次のいずれかの形式の日付、時刻、またはタイムスタンプ。
  - **日付文字列** : **yyyy-mm-dd**。たとえば、1955-06-13。
  - **時刻文字列** : **hh:mm:ss[.[nnn]]**。たとえば、14:34:30.75。
  - **タイムスタンプ文字列** : **yyyy-mm-dd hh:mm:ss[.[nnn]]**。たとえば、1924-01-14 12:00:00.000。
- **ODBC 形式** : 次のいずれかの形式の日付、時刻、またはタイムスタンプ。
  - **日付文字列** : **{d 'value'}**。たとえば、{d '2004-07-06'}。
  - **時刻文字列** : **{t 'value'}**。たとえば、{t '13:45:30'}。
  - **タイムスタンプ文字列** : **{ts 'value'}**。たとえば、{ts '2004-07-06 13:45:30'}。

日付を元の形式に変換するには、DateFormat 関数を使用して **"mm/dd/yy"** マスクを適用します。

クエリーオブクエリーのパフォーマンスについて

クエリーオブクエリーのパフォーマンスは、データベースから直接アクセスされた単一テーブルのクエリーオブジェクトでは高くなります。これは、データベースからアクセスされたクエリーオブジェクトのメタ情報が ColdFusion で保管されるからです。

SQL 結合が発生するクエリーの場合、クエリーオブクエリーのパフォーマンスは次のようになります。

1 2 つの列参照または定数の間に 1 つの等号のみを使用した単純な結合では、クエリーオブクエリーは効率的です。次に例を示します。

```
SELECT T1.a, b, c, d FROM T1, T2 WHERE T1.a = T2.a
```

2 述部に複数の式が含まれる結合では、クエリーオブクエリーの効率は低下します。次に例を示します。

```
SELECT T1.a, b, c, d FROM T1, T2
WHERE T1.a = T2.a AND T1.b + T1.c = T2.b + T2.c
```

クエリーオブクエリーの処理について

クエリーオブクエリーでは、次の処理が行えます。

- 列の比較
- 参照により渡されるクエリー
- 複合オブジェクト

データ型が異なる列の比較

ColdFusion MX 7 以降では、データ型の異なる列を比較する機能が拡張されています。

オペランドの一方が既知の型を持つ列である場合 (列の型が不明になるのは定数のみです)、クエリーオブクエリーは、型が不明な定数に対して、メタデータを持つオペランドの型を適用しようとします。適用可能な型の組み合わせは次のとおりです。

- バイナリ、文字列
- 日付、文字列
- 数値、bigdecimal
- ブール値、数値

つまり、バイナリと文字列の間で適用することはできますが、日付と文字列の間では適用できません。

両方のオペランドが既知のデータ型である場合、その型は同じであることが必要です。唯一の例外として、integer、float、double の間では適用が行えます。

両方のオペランドが定数の場合は、まず制約の厳しい型を、次に制約の緩やかな型を値に適用しようとします。

- まずバイナリを試し、次に文字列を試す
- まず日付を試し、次に文字列を試す
- まずブール値を試し、次に数値を試す

参照によるクエリーの受け渡し

クエリーオブクエリーは、その関連するクエリーから参照によってコピーされます。つまり、クエリーオブクエリーを作成してもクエリーは作成されません。したがって、クエリーオブクエリーに対して、データの並べ替え、変更、削除などの変更を行うと、ベースとなるクエリーオブジェクトにもその変更が適用されます。

元のクエリーを変更したくない場合は、Duplicate 関数を使用してコピーを作成し、コピーしたクエリーを使用してクエリーオブクエリーを作成します。

複合オブジェクトの管理

配列や構造体などの複合オブジェクトを含むレコードセットでは、クエリーオブクエリーを使用できません。

注意：レコードセットは複合オブジェクトに保存できます。

LDAP ディレクトリの管理

CFML アプリケーションでは、cfldap タグを使用して、LDAP (Lightweight Directory Access Protocol) ディレクトリのアクセスおよび管理を行います。

ここでは、LDAP データベースをクエリーおよび更新する方法について説明します。また、LDAP に馴染みがない読者のために、LDAP ディレクトリおよび LDAP プロトコルの概要についても説明します。ただし、使用している LDAP データベースの構造および属性については既知であると想定しています。したがって、LDAP ディレクトリの作成方法やディレクトリサーバーの管理方法については説明しません。LDAP および LDAP サーバーの詳細については、LDAP サーバーのマニュアルや LDAP に関する書籍を参照してください。

ここでの例では、Netscape および iPlanet Directory Server に付属の Airius サンプル LDAP データベースを使用します。

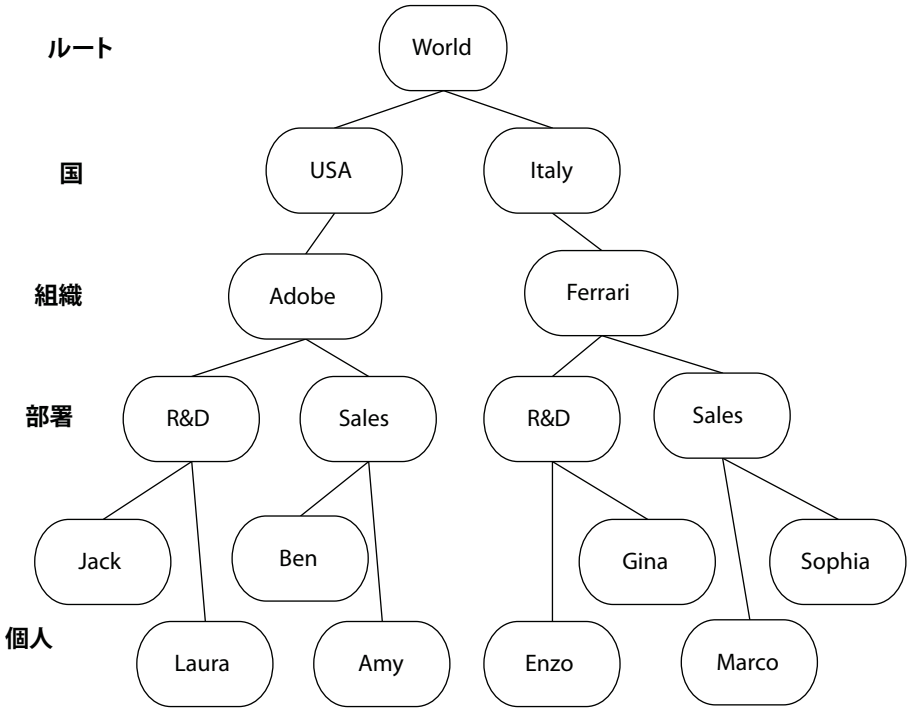
LDAP について

LDAP は、組織がディレクトリ情報を階層化したり、階層化したディレクトリ情報にアクセスしたりするためのプロトコルです。この場合のディレクトリとは、ディスクドライブ上のフォルダに格納されているファイルの集まりという意味ではなく、電話帳のような情報の集まりのことを意味します。

LDAP は、1990 年代の半ばに、処理能力の限られたコンピュータから ISO X.500 ディレクトリにアクセスするための方法として開発されました。それ以来、iPlanet Server などのネイティブ LDAP ディレクトリサーバー製品が開発されてきました。今日では、Novell NDS、Microsoft ADS (Active Directory Services)、Lotus Domino、Oracle などのディレクトリサーバーで、LDAP アクセスがサポートされています。

LDAP ディレクトリは、一般的には階層構造化されたデータベースです。階層内の各レイヤーは通常、組織構造の各レベルに対応しています。

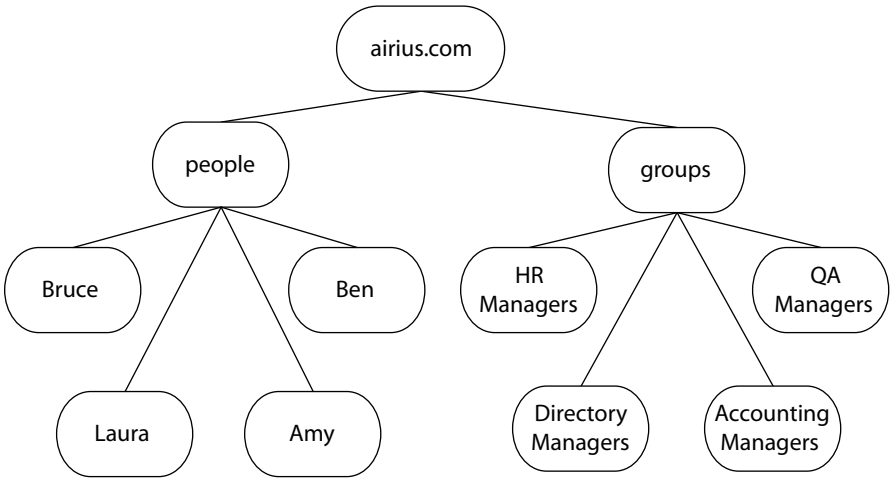
次の図は、簡単なディレクトリ構造を示しています。



この例は、完全に対称であり、各レイヤーのすべてのエントリは同じタイプです。

あるエントリにあるレイヤーが別のエントリにあるレイヤーと異なる情報を含むように LDAP ディレクトリを構築することもできます。

次の図は、非対称のディレクトリを示しています。



このディレクトリ構造では、ツリーの第 2 レベルが、people と groups という 2 つの組織ユニットに分割されています。第 3 レベルには、組織ユニットに固有の情報を持つエントリが存在しています。各個人のエントリには、氏名、電子メールアドレス、および電話番号が含まれています。各グループのエントリには、グループメンバーの氏名が含まれています。

この複雑さや柔軟性が LDAP の便利さの鍵となります。これにより、さまざまな組織構造を表現できます。

読み取りは頻繁に行うが変更はほとんど行わないディレクトリ状の情報にアクセスする場合は、従来のデータベースよりも LDAP を使用するほうが効率的です。

LDAP は、電子メール、住所、電話番号などの組織情報を格納するのによく使用されますが、それ以外の用途にも使用できます。たとえば、ColdFusion の拡張セキュリティ情報を LDAP データベースに保管することができます。

LDAP の情報構造

LDAP の情報構造は、LDAP のいくつかの概念に基づいています。

- エントリ
- 属性
- 識別名 (DN)
- スキーマ (オブジェクトクラスや属性タイプなど)

エントリ

LDAP の基本的な情報オブジェクトはエントリです。エントリは、1 つまたは複数の属性で構成されています。エントリは、ディレクトリスキーマで定義されているコンテンツルールに従って構築されます (454 ページの「スキーマ」を参照)。

LDAP ディレクトリでは、末端にあるノードだけでなく、ディレクトリ内の各ノードが 1 つのエントリになります。前の図では、示されているすべての項目がエントリです。たとえば最初の図では、USA も Ferrari も 1 つのエントリです。USA エントリには Language などの属性があり、Ferrari エントリには CEO などのエントリが含まれています。

属性

LDAP ディレクトリのエントリは、1 つまたは複数の属性で構成されています。属性はタイプと値を持ちます。タイプは、値が表す情報を示します。また、値の処理方法も示します。たとえば、属性が複数の値を持つかどうかはタイプによって決まります。電子メールアドレスの値を持つ mail 属性タイプは多値属性なので、一人に対して複数の電子メールアドレスを保管できます。

よく使用されるいくつかの属性タイプには、短い名前が用意されています。通常、この短い名前は長いタイプ名に対応しており、どちらの名前も同じように使用できます。次の表に、LDAP ディレクトリでよく使用される属性タイプのキーワードを示します。

キーワード	正式名	コメント
c	CountryName	
st	stateOrProvinceName	
l	LocalityName	通常は市を示しますが、その他の地域単位でも可
street	StreetAddress	
o	OrganizationName	
ou	OrganizationalUnitName	
cn	CommonName	通常は氏名
sn	SurName	
dc	domaincomponent	
mail	mail	電子メールアドレス

詳細については、455 ページの「属性タイプ」を参照してください。

識別名 (DN)

エントリの識別名 (DN) は、ディレクトリ内のエントリを一意に識別するためのものです。1 つの DN は、いくつかの相対識別名 (RDN) で構成されます。RDN は、親エントリに含まれている子エントリの中から 1 つを識別するためのものです。たとえば、LDAP についての最初の図では、Ferrari エントリの RDN は "o=Ferrari" です。

1 つのエントリの DN は、そのエントリの RDN の後に、親エントリの DN を付加したものになります。つまり、DN は、そのエントリからディレクトリツリーのルートエントリに至るまでの、各エントリの RDN を順に並べたものになります。それぞれの RDN はカンマで区切ります。オプションでスペースを含めてもかまいません。たとえば、最初の図では、Ferrari エントリの DN は "o=Ferrari, c=Italy" です。

ファイルシステムのパスや URL と同様に、検索を行うには正しい形式の LDAP 名を入力する必要があります。

注意 : RDN はディレクトリエントリの属性です。DN は属性ではありませんが、クエリーの attributes リストで "dn" と指定すると、DN を出力できます。詳細については、『CFML リファレンス』の cfldap を参照してください。Adobe ColdFusion で返される DN は、カンマの後に必ずスペースが付加されています。

多値 **RDN** は、属性と値の複数のペアで構成されます。多値 RDN では、属性と値のペアをプラス記号 (+) で区切ります。前述のサンプルディレクトリでは、各個人に対して、"cn=Robert Boyd + mail=rjboyd@adobe.com" のように、一般名と電子メールアドレスからなる複雑な RDN を使用できます。

スキーマ

スキーマおよびオブジェクトクラスの概念は、LDAP を完全に理解する上で中核となる概念です。ここでは、この概念の詳細な説明はしませんが、cfldap タグを効果的に使用するための情報を提供します。

ディレクトリスキーマは、ディレクトリに保管できるものを規定するルールセットです。このディレクトリスキーマによって、少なくとも次の 2 つのディレクトリ特性が定義されます。

- エントリが所属するオブジェクトクラス
- ディレクトリ属性タイプ

オブジェクトクラス

オブジェクトクラスは、関連する情報をグループ化したものです。多くの場合、1 つのオブジェクトクラスは、国、個人、部屋、ドメインなどの、現実の物や概念に対応しています (country、person、room、domain はすべて標準のオブジェクトタイプ名です)。LDAP ディレクトリの各エントリは、1 つまたは複数のオブジェクトクラスに属しています。

オブジェクトクラスは、次の特性によって定義されます。

- クラス名
- クラスを識別する一意のオブジェクト ID
- そのクラスのエントリに含まれている必要がある属性タイプ
- そのクラスのエントリにオプションで含めることができまる属性タイプ
- (オプション) クラスの派生元の上位クラス

あるクラスから派生したクラスに属しているエントリのオブジェクトクラス属性には、最下位レベルのクラスと、この最下位レベルのクラスを派生させたすべての上位クラスがリストされます。

ディレクトリエントリを追加、変更、または削除する場合は、エントリのオブジェクトクラスが多値属性である可能性を考慮する必要があります。たとえば、新規エントリを追加する場合は、cfldap タグの attributes 属性でオブジェクトクラスを指定します。エントリのオブジェクトクラス名を取得するには、クエリー属性のリストで "objectclass" を指定します。特定のタイプの情報を持つエントリを取得するには、cfldap タグの filter 属性でオブジェクトクラス名を使用します。

属性タイプ

スキーマの属性タイプの指定では、次のプロパティを定義します。

- 属性タイプ名
- 属性タイプを識別する一意のオブジェクト ID
- (オプション) タイプが単一値であるか多値であるか (デフォルトは多値)
- 属性シンタックスおよび一致ルール (大文字と小文字の区別など)

属性タイプの定義では、タイプが表現する値の範囲やサイズを制限したり、アプリケーション固有の使用インジケータを使用したりすることもできます。標準的な属性では、登録済みの数値 ID で、シンタックスおよび一致ルール情報を指定します。属性シンタックスの詳細については、ETF RFC 2252 (http://www.ietf.org/rfc/rfc2252.txt) を参照してください。

creatorsName や modifyTimeStamp などの操作属性は、ディレクトリサービスで管理されるので、ユーザーアプリケーションでは変更できません。

ColdFusion での LDAP の使用

ColdFusion で LDAP ネットワークディレクトリサービスにクエリーを実行するには、cfldap タグを使用します。cfldap タグを使用すると、LDAP を使用して次のようなことが行えます。

- インターネットホワイトページを作成して、個人やリソースを簡単に探したり、情報を取得したりできるようにする。
- ディレクトリエントリの管理や更新を行うフロントエンドを提供する。
- ディレクトリクエリーで取得したデータを利用して処理を行うアプリケーションを構築する。
- 組織または企業の既存のディレクトリサービスをアプリケーションに統合する。

cfldap タグの action 属性では、LDAP ディレクトリに対する次の操作を指定できます。

アクション	説明
query	ディレクトリから属性値を取得します。
add	ディレクトリにエントリを追加します。
delete	ディレクトリからエントリを削除します。
modify	ディレクトリエントリの属性値の追加、削除、または変更を行います。
modifyDN	ディレクトリエントリ名を変更します (識別名を変更します)。

次の表に、各アクションでの必須属性とオプション属性を示します。各属性の詳細については、『CFML リファレンス』の cfldap タグを参照してください。

アクション	必須属性	オプション属性
query	server、name、start、attributes	port、username、password、timeout、secure、rebind、referral、scope、filter、sort、sortControl、startRow、maxRows、separator、delimiter
add	server、dn、attributes	port、username、password、timeout、secure、rebind、referral、separator、delimiter
delete	server、dn	port、username、password、timeout、secure、rebind、referral
modify	server、dn、attributes	port、username、password、timeout、secure、rebind、referral、modifyType、separator、delimiter
modifyDN	server、dn、attributes	port、username、password、timeout、secure、rebind、referral

LDAP ディレクトリへのクエリーの実行

cfldap タグを使用して、LDAP ディレクトリを検索できます。このタグは、結果が含まれた ColdFusion クエリーオブジェクトを返します。これは、通常のクエリー結果と同じように使用できます。LDAP ディレクトリにクエリーを実行するときは、検索を開始するディレクトリエントリと、値を取得する属性を指定します。検索スコープや属性内容のフィルタルールを指定したり、他の属性を使用して検索を細かく制御したりすることもできます。

スコープ

検索スコープは、検索の範囲を制限します。デフォルトのスコープは、start 属性で指定した識別名の 1 つ下のレベルです。このスコープには、start 属性で指定したエントリは含まれません。たとえば、start 属性が "ou=support, o=adobe" である場合は、support の 1 つ下のレベルが検索されます。start エントリのレベルにクエリーを制限したり、start エントリの下にあるサブツリー全体に範囲を拡張することもできます。

検索フィルタ

検索フィルタのシンタックスは、< 属性 > < 演算子 > < 値 > の形式です。デフォルトのフィルタ objectclass=* は、スコープ内のすべてのエントリを返します。

次の表に、フィルタ演算子をリストします。

演算子	例	一致
=*	(mail=*)	mail 属性を含むすべてのエントリ
=	(o=adobe)	組織名が adobe であるエントリ
~=	(sn~=Hansen)	Hansen に近い姓を持つエントリ。近似一致のルールはディレクトリベンダによって異なりますが、検索文字列で指定した単語に " 近い発音 " を持つエントリに一致するのが一般的です。この例では、Hansen や Hanson などの姓を持つエントリが返されます。
>=	(st>=ma)	"ma" という名前、およびアルファベット順の state 属性リストで "ma" の後にある名前
<=	(st<=ma)	"ma" という名前、およびアルファベット順の state 属性リストで "ma" の前にある名前
*	(o=macro*)	"macro" で始まる組織名
	(o=*media)	"media" で終わる組織名
	(o=mac*ia)	"mac" で始まり、"ia" で終わる組織名。"m*ro*dia" のように、文字列内に複数の * 演算子を使用できます。
	(o=*med*)	"med" という文字列を含む組織名。これには "med" の完全一致の文字列も含まれます。
&	(&(o=adobe) (co=usa))	組織名が "adobe" で、国が "usa" であるエントリ
|	(|(o=adobe) (sn=adobe) (cn=adobe))	組織名、姓、または一般名が "adobe" であるエントリ
!	(!(STREET=*))	StreetAddress 属性を含まないエントリ

ブール演算子 & および | は、3 つ以上の属性に対しても使用できます。この演算子は、対象となるすべての属性の前に置きます。フィルタは括弧で囲み、条件全体を括弧で囲んでグループ化します。

一致パターンの中にアスタリスク、左括弧、右括弧、円記号、または null 文字を使用する場合は、その文字の代わりに、次の 3 文字のエスケープシーケンスを使用します。

文字	エスケープシーケンス
*	¥2A
(	¥28
)	¥29
¥	¥5C
NUL	¥00

たとえば、一般名 St*r Industries を検索するには、フィルタ

(cn=St¥2Ar Industries) を使用します。

LDAP v3 では、サーバー固有の一致ルールを許容する拡張可能な一致フィルタがサポートされています。拡張可能な一致フィルタの使用方法の詳細については、LDAP サーバーのマニュアルを参照してください。

検索とソートの注意事項

- 多値属性タイプの複数の値を検索するには、& 演算子を使用して、各属性値の式を組み合わせます。たとえば、cn=Robert Jones および cn=Bobby Jones のエントリを検索するには、次のフィルタを指定します。

```
filter="(&(cn=Robert Jones)(cn=Bobby Jones))"
```

- 検索フィルタ属性にはオブジェクトクラスを使用できます。たとえば、次のような検索フィルタを使用できます。

```
filter="(objectclass=inetorgperson)"
```

- クエリー結果のソート方法を指定するには、ソートする属性を sort フィールドで識別します。デフォルトでは、大文字と小文字を区別した昇順で結果が返されます。降順でソートしたり、大文字と小文字を区別しない場合は、sortControl 属性を使用します。
- ColdFusion は、LDAP サーバーにソートをリクエストします。その際、次のような結果になる可能性があります。
  - ソートの順番が、ColdFusion MX とそれ以前のバージョンで異なる可能性があります。
  - ソートを指定しても、LDAP サーバーでソートがサポートされていない場合は、ColdFusion でエラーが発生します。ソートがサポートされていないサーバーからの結果をソートするには、結果に対してクエリーオブクエリーを使用します。
- フィルタ演算子を使用して高度な検索基準を指定している場合は、cfldap でサポートされている同期検索ルーチンの LDAP サーバーでの処理速度が遅いと、パフォーマンスが低下する可能性があります。cfldap タグの timeout 属性や maxRows 属性を使用して、エントリ数を制限したり、一定時間内にサーバーが応答しない場合にクエリーを終了したりすれば、クエリーを実行するページの体感的なパフォーマンスを改善できます。

エントリのすべての属性の取得

通常、エントリのすべての属性を取得するクエリーは使用しません。このようなクエリーでは、ディレクトリサーバーでのみ使用される属性が返されます。すべての属性を取得するには、クエリーで attributes="*" と指定します。

このクエリーを実行すると、各要素に属性の名前と値のペアが 1 つ含まれている構造体が結果として返されます。このタグではクエリーオブジェクトは返されません。LDAP ディレクトリのエントリは、リレーショナルテーブルの行と異なり、オブジェクトクラスによって構成が異なるからです。

たとえば、次のコードでは Airius ディレクトリの内容が取得されます。

```
<cfldap name="GetList"
    server=#myServer#
    action="query"
    attributes="*"
    scope="subtree"
    start="o=airius.com"
    sort="sn,cn">
```

このタグでは、この組織内のすべての個人のエントリと、すべてのグループのエントリが返されます。グループのエントリと個人のエントリではオブジェクトクラスが異なるので、エントリを構成する属性も異なります。もし、両方のタイプのエントリを 1 つのクエリーオブジェクトで返したとすると、グループに固有の属性値のみを持つ行や、個人に固有の属性値のみを持つ行が存在することになります。実際には、各属性を 1 つのエントリとして持つ構造体が返されます。

例 : LDAP ディレクトリへのクエリーの実行

次の例では、cfldap タグを使用して、Airius 社のサンタクララオフィスに所属する個人の情報を取得します。個人名の全部または一部を入力すると、一致した名前、所属部門、電子メールアドレス、および電話番号のリストを取得できます。

この例では、Netscape Directory Server に付属するサンプルの Airius 社ディレクトリを使用します。このディレクトリへのアクセス権がない場合は、使用している LDAP ディレクトリにクエリーを実行するようにコードを変更してください。

1 次のようなファイルを作成します。

```
<!--- This example shows the use of CFLDAP --->
<html>
<head> <title>cfldap Query Example</title> </head>

<h3>cfldap Query Example</h3>

<body>
<p>This tool queries the Airius.com database to locate all people in the company's
    Santa Clara office whose common names contain the text entered in the form.</p>

<p>Enter a full name, first name, last name, or name fragment.</p>

<form action="cfldap.cfm" method="POST">
    <input type="text" name="name"><br><br>
    <input type="submit" value="Search">
</form>
<!--- make the LDAP query --->
<!-- Note that some search text is required.
    A search filter of cn=** would cause an error -->
<cfif (isdefined("form.name") AND (form.name IS NOT ""))>
    <cfldap
        server="ldap.airius.com"
        action="query"
        name="results"
        start="ou=People, o=Airius.com"
        scope="onelevel"
        filter="(&(cn=*#form.Name#*)(l=Santa Clara))"
        attributes="cn,sn,ou,mail,telephonenumber"
        sort="ou,sn"
        maxrows=100
        timeout=20000
    >

<!--- Display results --->
    <table border=0 cellspacing=2 cellpadding=2>
     <tr>
        <th colspan=4><cfoutput>#results.RecordCount# matches found</cfoutput>
        </th>
     </tr>
     <tr>
        <th>Name</th>
        <th>Department</th>
        <th>E-Mail</th>
        <th>Phone</th>
     </tr>
    <cfoutput query="results">
     <tr>
        <td>#cn#</td>
        <td>#listFirst(ou)#</td>
        <td><a href="mailto:#mail#">#mail#</a></td>
        <td>#telephonenumber#</td>
    </tr>
    </cfoutput>
    </table>
</cfif>

</body>
</html>
```

2 server 属性の ldap.airius.com を、インストールした Airius データベースの名前に変更します。

3 ページに cfldap.cfm という名前を付けて保存し、ブラウザで実行します。

コードの説明

このコードについて、次の表で説明します。

コード	説明
<form action="cfldap.cfm" method="POST"> <input type="text" name="name"> <input type="submit" value="Search"> </form>	フォームを使用して、検索する名前またはその一部を取得します。
<cfif (isdefined("form.name") AND (form.name IS NOT "")) >	ユーザーがフォームを送信したことを確認します。これは、このフォームページがアクションページでもあるので必要となります。ユーザーが検索テキストを入力したことを確認します。
<cfldap server="ldap.airius.com" action="query" name="results" start="ou=People, o=Airius.com" scope="onelevel" filter="(&(cn=*#form.Name#*)(l=Santa Clara))" attributes="cn,sn,ou,mail,telephonenumber" sort="ou,sn" maxrows=100 timeout=20000 >	LDAP サーバー ldap.airius.com に匿名で接続し、ディレクトリをクエリーし、results というクエリーオブジェクトに結果を返します。 ou=People, o=Airius.com という識別名のディレクトリエントリからクエリーを開始し、このエントリの 1 つ下のディレクトリレベルを検索します。 location (l) 属性値が "Santa Clara" であり、入力したテキストが一般名属性に含まれているエントリのレコードをリクエストします。 各エントリの一般名、姓、組織ユニット、電子メールアドレス、および電話番号を取得します。 結果を、組織名でソートしてから姓でソートします。デフォルトのソート順を使用します。 リクエストを 100 エントリに制限します。サーバーが 20 秒以内にデータを返さない場合、LDAP のタイムアウトを示すエラーが発生します。
<table border=0 cellspacing=2 cellpadding=2> <tr> <th colspan=4><cfoutput>#results.RecordCount# matches found</cfoutput> </th> </tr> <tr> <th>Name</th> <th>Department</th> <th>E-Mail</th> <th>Phone</th> </tr> <cfoutput query="results"> <tr> <td>#cn#</td> <td>#listFirst(ou)#</td> <td>#mail#</td> <td>#telephonenumber#</td> </tr> </cfoutput> </table> </cfif>	出力を表示する表を開始します。 返されたレコードの数を表示します。 各エントリの一般名、部門、電子メールアドレス、および電話番号を表示します。 組織ユニット値のリスト内にある最初のエントリのみを表示します。詳細については、表の後にある説明を参照してください。

この検索では、フィルタ内で論理積ステートメントを使用しています。検索で返される surname 属性は、結果をソートするためにのみ使用しています。

このクエリーの ou 属性の値は、2 つの値で構成されるカンマ区切りのリストです。値の 1 つは部門名で、もう 1 つは People です。これは、Airius データベースで、ou 属性タイプが次の 2 か所で使用されているからです。

- 識別名の中 (人、グループ、ディレクトリサーバーなどの組織ユニットを区別する、ディレクトリツリーの第 2 レベル)
- 各個人のエントリ (部門識別子として使用)

属性値は、個人エントリからディレクトリツリールートの順で返されるので、ListFirst 関数を使用して所属部門を抽出します。

LDAP ディレクトリの更新

cfldap タグを使用すると、LDAP ディレクトリエントリに対して次のようなアクションを実行できます。

- 追加
- 削除
- 属性の追加
- 属性の削除
- 属性の置き換え
- DN の変更 (エントリ名の変更)

これらのアクションによって、LDAP ディレクトリの内容をリモートで管理できます。

LDAP ディレクトリを管理するための ColdFusion ページを構築します。このフォームでは、テーブル内にディレクトリエントリを表示し、固有のユーザー ID に基づいてフォームフィールドを設定するボタンを用意します。

この ColdFusion サンプルページでは、エントリ属性の追加や削除、または DN の変更は行いません。これらの操作については、468 ページの「ディレクトリエントリの属性の追加と削除」および 468 ページの「ディレクトリエントリの DN の変更」を参照してください。

この例では、コードを短くするために、本番用のアプリケーションでは不適切な動作も許容しています。特に、次のような動作が行われます。

- 無効なユーザー ID を入力して [更新] または [削除] ボタンをクリックすると、" オブジェクトは存在しません " というエラーが ColdFusion で発生します。これは、更新または削除するディレクトリエントリが存在しないからです。実際のアプリケーションでは、エントリを変更または削除する前に、その ID がディレクトリに存在することを確認してください。
- 空白のフォームに有効な ID を入力して [更新] をクリックすると、そのユーザーのすべての属性が削除されます。実際のアプリケーションでは、ディレクトリを更新する前に、すべての必須属性フィールドに有効なエントリが含まれていることを確認してください。

ディレクトリエントリの追加

LDAP ディレクトリにエントリを追加する場合は、DN、すべての必須属性 (エントリのオブジェクトクラスなど)、およびその他のオプション属性を指定します。次の例に、LDAP ディレクトリにエントリを追加するフォームの構成を示します。

1 次のようなファイルを作成します。

```
<!--- Set the LDAP server ID, user name, and password as variables
    here so they can be changed in only one place. --->
<cfset myServer="ldap.myco.com">
<cfset myUserName="cn=Directory Manager">
<cfset myPassword="password">

<!--- Initialize the values used in form fields to empty strings. --->
<cfparam name="fullNameValue" default="">
<cfparam name="surnameValue" default="">
<cfparam name="emailValue" default="">
<cfparam name="phoneValue" default="">
<cfparam name="uidValue" default="">

<!---When the form is submitted, add the LDAP entry. --->
<cfif isdefined("Form.action") AND Trim(Form.uid) IS NOT "">
    <cfif Form.action is "add">
        <cfif Trim(Form.fullName) is "" OR Trim(Form.surname) is ""
            OR Trim(Form.email) is "" OR Trim(Form.phone) is "">
                <h2>You must enter a value in every field.</h2>
                <cfset fullNameValue=Form.fullName>
                <cfset surnameValue=Form.surname>
                <cfset emailValue=Form.email>
                <cfset phoneValue=Form.phone>
                <cfset uidValue=Form.uid>
        <cfelse>
            <cfset attributelist="objectclass=top, person,
                    organizationalperson, inetOrgPerson;
                cn=#Trim(Form.fullName)#; sn=#Trim(Form.surname)#;
                mail=#Trim(Form.email)#;
                telephonenumber=#Trim(Form.phone)#;
                ou=Human Resources;
                uid=#Trim(Form.uid)#">
            <cfldap action="add"
                attributes="#attributeList#"
                dn="uid=#Trim(Form.uid)#, ou=People, o=Airius.com"
                server=#myServer#
                username=#myUserName#
                password=#myPassword#>
            <cfoutput><h3>Entry for User ID #Form.uid# has been added</h3>
            </cfoutput>
        </cfif>
    </cfif>
</cfif>

<html>
<head>
    <title>Update LDAP Form</title>
</head>
<body>
<h2>Manage LDAP Entries</h2>

<cfform action="update_ldap.cfm" method="post">
    <table>
        <tr><td>Full Name:</td>
            <td><cfinput type="Text"
                 name="fullName"
                 value=#fullNameValue#
                 size="20"
                 maxlength="30"
                 tabindex="1"></td>
        </tr>
        <tr><td>Surname:</td>
```

```
            <td><cfinput type="Text"
                name="surname"
                Value= "#surnameValue#"
                size="20"
                maxlength="20"
                tabindex="2"></td>
        </tr>
        <tr>
            <td>E-mail Address:</td>
            <td><cfinput type="Text"
                name="email"
                value="#emailValue#"
                size="20"
                maxlength="20"
                tabindex="3"></td>
        </tr>
        <tr>
            <td>Telephone Number:</td>
            <td><cfinput type="Text"
                name="phone"
                value="#phoneValue#"
                size="20"
                maxlength="20"
                tabindex="4"></td>
        </tr>
        <tr>
            <td>User ID:</td>
            <td><cfinput type="Text"
                name="uid"
                value="#uidValue#"
                size="20"
                maxlength="20"
                tabindex="5"></td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="Submit"
                 name="action"
                 value="Add"
                 tabindex="8"></td>
        </tr>
    </table>
    <br>
    *All fields are required for Add<br>
</cfform>

<!---Output the user list. --->
<h2>User List for the Human Resources Department</h2>
<cfldap name="GetList"
    server=#myServer#
    action="query"
    attributes="cn,sn,mail,telephonenumber,uid"
    start="o=Airius.com"
    scope="subtree"
    filter="ou=Human Resources"
    sort="sn,cn"
    sortControl="asc, nocase">
```

```
<table border="1">
    <tr>
        <th>Full Name</th>
        <th>Surname</th>
        <th>Mail</th>
        <th>Phone</th>
        <th>UID</th>
    </tr>
    <cfoutput query="GetList">
    <tr>
        <td>#GetList.cn#</td>
        <td>#GetList.sn#</td>
        <td>#GetList.mail#</td>
        <td>#GetList.telephonenumber#</td>
        <td>#GetList.uid#</td>
    </tr>
    </cfoutput>
</table>
</body>
</html>
```

2 ファイルの冒頭に記述されている myServer、myUserName、および myPassword 変数の値を、使用している LDAP サーバーに適した値に変更します。

3 ページに "update_ldap.cfm" という名前を付けて保存し、ブラウザで実行します。

コードの説明

このコードについて、次の表で説明します。

コード	説明

`<cfset myServer="ldap.myco.com">`
`<cfset myUserName="cn=Directory Manager">`
`<cfset myPassword="password">`
	LDAP 接続情報変数を初期化します。すべての接続情報を変数にすることで、それらを 1 か所で変更できるようにしています。

`<cfparam name="fullNameValue" default="">`
`<cfparam name="surnameValue" default="">`
`<cfparam name="emailValue" default="">`
`<cfparam name="phoneValue" default="">`
`<cfparam name="uidValue" default="">`
	フォームフィールド値の変数に、デフォルト値として空の文字列を設定します。データ入力フォームでは、value 属性を持つ cfinput フィールドを使用しています。これによって、フォームの初期値が設定されます。また、送信されたフォームが不完全であるためにページが再表示されても、入力された値が保持されます。

`<cfif isdefined("Form.action") AND Trim(Form.uid) IS NOT "">`	ユーザーがフォームにユーザー ID を入力したことを確認します。
`<cfif Form.action is "add">`	ユーザーが [Add] をクリックした場合は、次に続くコードを処理します。

`<cfif Trim(Form.fullName) is "" OR Trim(Form.surname) is ""`
`OR Trim(Form.email) is "" OR Trim(Form.phone) is "">`
`<h2>You must enter a value in every field.</h2>`
`<cfset fullNameValue=Form.fullName>`
`<cfset surnameValue=Form.surname>`
`<cfset emailValue=Form.email>`
`<cfset phoneValue=Form.phone>`
`<cfset uidValue=Form.uid>`
	送信されたフォームに空白のフィールドがある場合は、メッセージを表示し、ユーザーが送信したデータを表示する別のフォームフィールドを設定します。

`<cfelse>`
`<cfset attributelist="objectclass=top, person,`
`organizationalperson, inetOrgPerson;`
`cn=#Trim(Form.fullName)#; sn=#Trim(Form.surname)#;`
`mail=#Trim(Form.email)#;`
`telephonenumber=#Trim(Form.phone)#; ou=Human Resources;`
`uid=#Trim(Form.uid)#">`
	ユーザーがすべてのフィールドにデータを入力した場合は、attributelist 変数を設定して、オブジェクトクラスや組織ユニット (この例では Human Resources) などのエントリ属性を指定します。
	
	Trim 関数で、ユーザーデータの前後にあるスペースを削除しています。

コード	説明
<cfldap action="add" attributes="#attributeList#" dn="uid=#Trim(Form.uid)#, ou=People, o=Airius.com" server=#myServer# username=#myUserName# password=#myPassword#> <cfoutput> Entry for User ID #Form.uid# has been added </cfoutput> </cfif> </cfif> </cfif>	ディレクトリに新規エントリを追加します。
<cffo rm action="update_ldap.cfm" method="post"> <table> <tr> <td>Full Name:</td> <td><cfinput type="Text" name="fullName" value=#fullNameValue# size="20" maxlength="30" tabindex="1"></td> </tr> . . <tr> <td colspan="2"> <input type="Submit" name="action" value="Add" tabindex="8"></td> </tr> </table> *All fields are required for Add </cfform>	データ入力フォームをテーブルとして出力します。各 cfinput フィールドは常に値を持ちます。これは、ページが呼び出されるときに value 属性によって設定されます。ユーザーが [Add] をクリックしてフォームが再表示されるときには、value 属性によってフォームのコンテンツが更新されます。ユーザーが必要なデータをすべて入力しなかった場合に、この機能を使用します。
<cfldap name="GetList" server=#myServer# action="query" attributes="cn,sn,mail,telephonenumber,uid" start="o=Airius.com" scope="subtree" filter="ou=Human Resources" sort="sn,cn" sortControl="asc, nocase">	ディレクトリをクエリーし、一般名、姓、電子メールアドレス、電話番号、およびユーザー ID を一致するエントリから取得します。 DN が o=Airius.com であるエントリのサブツリーを検索して、組織ユニットが Human Resources であるすべてのエントリを選択します。 結果を姓でソートし、さらに一般名でソートします (これによって、姓、名の順でソートされます)。昇順で大文字と小文字を区別せずにソートします。
<table border="1"> <tr> <th>Full Name</th> <th>Surname</th> <th>Mail</th> <th>Phone</th> <th>UID</th> </tr> <cfoutput query="GetList"> <tr> <td>#GetList.cn#</td> <td>#GetList.sn#</td> <td>#GetList.mail#</td> <td>#GetList.telephonenumber#</td> <td>#GetList.uid#</td> </tr> </cfoutput> </table> </body> </html>	テーブル内にクエリー結果を表示します。

ディレクトリエントリの削除

ディレクトリエントリを削除するには、エントリの DN を指定します。

次の例は、エントリを追加するコードをベースにしています。ここでは、[Retrieve] および [Delete] ボタンを追加しています。[Retrieve] ボタンを使用すると、フォームを削除する前にユーザーの情報を表示できます。

1 461 ページの「ディレクトリエントリの追加」で作成した "update_ldap.cfm" を開きます。

2 1 番目と 2 番目の cfif タグの間に、次のコードを追加します。

```
<cfelseif Form.action is "Retrieve">
    <cfldap name="GetEntry"
        server=#myServer#
        action="query"
        attributes="cn,sn,mail,telephonenumber,uid"
        scope="subtree"
        filter="uid=#Trim(Form.UID)#"
        start="o=Airius.com">
    <cfset fullNameValue = GetEntry.cn[1]>
    <cfset surnameValue = GetEntry.sn[1]>
    <cfset emailValue = GetEntry.mail[1]>
    <cfset phoneValue = GetEntry.telephonenumber[1]>
    <cfset uidValue = GetEntry.uid[1]>
<cfelseif Form.action is "Delete">
    <cfldap action="delete"
        dn="uid=#Trim(Form.UID)#, ou=People, o=Airius.com"
        server=#myServer#
        username=#myUserName#
        password=#myPassword#>
    <cfoutput><h3>Entry for User ID #Form.UID# has been deleted
    </h3></cfoutput>
```

3 [Add] ボタン (フォームの下部にある Value="Add" の input タグ) のコードの最後にある </td> 終了タグを削除します。

4 [Add] ボタンの input タグの後に、次のコードを追加します。

```
     
    <input type="Submit"
         name="action"
         value="Retrieve"
         tabindex="7">
     
    <input type="Submit"
         name="action"
         value="Delete"
         tabindex="8"></td>
```

5 このファイルを保存してブラウザで実行します。

コードの説明

このコードについて、次の表で説明します。

コード	説明
<cfelseif Form.action is "Retrieve"> <cfldap name="GetEntry" server=#myServer# action="query" attributes="cn,sn,mail,telephonenumber,uid" scope="subtree" filter="uid=#Trim(Form.UID)#" start="o=Airius.com"> <cfset fullNameValue = GetEntry.cn[1]> <cfset surnameValue = GetEntry.sn[1]> <cfset emailValue = GetEntry.mail[1]> <cfset phoneValue = GetEntry.telephonenumber[1]> <cfset uidValue = GetEntry.uid[1]>	ユーザーが [Retrieve] をクリックした場合は、ディレクトリをクエリーして、指定されたユーザー ID の情報を取得します。 フォームフィールドの Value 属性を該当するクエリー列に設定します。 この例では、GetEntry クエリーオブジェクトの最初の行を識別するために、配列インデックス [1] を使用しています。クエリーは常に 1 行のみを返すので、このインデックスは省略できます。
<cfelseif Form.action is "Delete"> <cfldap action="delete" dn="uid=#Trim(Form.UID)#, ou=People, o=Airius.com" server=#myServer# username=#myUserName# password=#myPassword#> <cfoutput> Entry for User ID #Form.UID# has been deleted 	ユーザーが [Delete] をクリックした場合は、指定されたユーザー ID を持つエントリを削除し、ユーザーにエントリが削除されたことを通知します。
 <input type="Submit" name="action" value="Retrieve" tabindex="7"> <input type="Submit" name="action" value="Delete" tabindex="8"></td>	取得アクションおよび削除アクション用の送信ボタンを表示します。

ディレクトリエントリの更新

cfldap タグで、エントリの属性値を変更できます。それには、エントリの DN を dn 属性に指定し、変更する属性と新しい値のリストを attributes 属性に指定します。

次の例は、エントリを追加および削除するコードをベースにしています。この例では、1 つのエントリの属性を更新できます。UID は DN の一部であるため、変更できません。

1. 461 ページの「ディレクトリエントリの追加」で作成した "update_ldap.cfm" を開きます。
2. cfelseif ブロックと cfif タグの間に、次のコードを追加します。

```
 <cfelseif Form.action is "Update">
<cfset attributelist="cn=#Trim(form.FullName)#; sn=#Trim(Form.surname)#; mail=#Trim(Form.email)#;
    telephonenumber=#Trim(Form.phone)#">
<cfldap action="modify"
    modifytype="replace"
    attributes="#attributeList#"
    dn="uid=#Trim(Form.UID)#, ou=People, o=Airius.com"
    server=#myServer#
    username=#myUserName#
    password=#myPassword#>
<cfoutput><h3>Entry for User ID #Form.UID# has been updated</h3>
</cfoutput>
```

3. [Delete] ボタン (フォームの下部にある Value="Delete" の input タグ) のコードの最後にある </td> 終了タグを削除します。
4. [Delete] ボタンの input タグの後に、次のコードを追加します。

```
  
<input type="Submit"
    name="action"
    value="Update"
    tabindex="9"></td>
```

5 このファイルを保存してブラウザで実行します。

コードの説明

このコードについて、次の表で説明します。

コード	説明
<cfelseif Form.action is "Update"> <cfset attributelist="cn=#Trim(form.FullName)#; sn=#Trim(Form.surname)#; mail=#Trim(Form.email)#; telephonenumber=#Trim(Form.phone)#"> <cfldap action="modify" modifytype="replace" attributes="#attributeList#" dn="uid=#Trim(Form.UID)#, ou=People, o=Airius.com" server=#myServer# username=#myUserName# password=#myPassword#> <cfoutput> Entry for User ID #Form.UID# has been updated </cfoutput>	ユーザーが [Update] をクリックした場合は、属性リストにフォームフィールド値を設定し、指定された UID を持つエントリの属性を置換します。 エントリが更新されたことを示すメッセージを表示します。 このコードでは、フォーム内のすべての属性を、空白であるかどうかを確認せずに置換に使用します。さらに完全な例では、空白のフィールドを確認し、データの入力を求めるか、または attributes 文字列の該当する属性を除外します。
 <input type="Submit" name="action" value="Update" tabindex="9"></td>	更新アクションのための送信ボタンを定義します。

ディレクトリエントリの属性の追加と削除

次の表に、エントリの LDAP 属性を追加および削除するときに指定する必要のある cfldap タグを示します。

アクション	cfldap のシンタックス
エントリに属性を追加	dn = "entry dn" action = "modify" modifyType = "add" attributes = "attribname=attribValue[;...]"
エントリから属性を削除	dn = "entry dn" action = "modify" modifyType = "delete" attributes = "attribName[;...]"

1 つのステートメントで複数の属性を追加または削除できます。これを行うには、属性文字列の中で属性をセミコロンで区切ります。

次の例では、description および seealso LDAP 属性を指定しています。

```
attributes="description=Senior Technical Writer;seealso=writers"
```

属性文字列で多値属性の値を区切るときに使用する文字は変更できます。また、文字列に複数の属性を含めるときに使用する属性の区切り文字も変更できます。詳細については、469 ページの「カンマまたはセミコロンを含む属性の指定」を参照してください。

属性を追加または削除できるのは、ディレクトリスキーマの該当するエントリのオブジェクトクラスで、その属性がオプションと定義されている場合のみです。

ディレクトリエントリの DN の変更

エントリの DN を変更するには、cfldap タグに次の情報を指定します。

```
dn="original DN"
action="modifyDN"
attributes="dn=new DN"
```

次に例を示します。

```
<cfldap action="modifyDN"
    dn="#old_UID#, ou=People, o=Airius.com"
    attributes="uid=#newUID#"
    server=#myServer#
    username=#myUserName#
    password=#myPassword#>
```

新しい DN およびエントリ属性はディレクトリスキーマに従っている必要があります。したがって、ディレクトリツリー内でエントリを自由に移動することはできません。変更できるのはリーフノードのみです。たとえば、子を持つグループは名前を変更できません。

注意：LDAP v2 では、エントリの DN を変更できません。

高度なトピック

より高度なテクニックを使用すると、LDAP ディレクトリをより効果的に使用できます。

カンマまたはセミコロンを含む属性の指定

LDAP 属性値にカンマが含まれていることがあります。cfldap タグでは通常、カンマは値リストの属性値を区切る文字と解釈されます。同様に、属性にセミコロンが含まれていることがありますが、cfldap では通常、セミコロンは属性リストの属性を区切る文字と解釈されます。デフォルトのセパレータおよび区切り文字を上書きするには、cfldap タグの separator および delimiter 属性を使用します。

たとえば、LDAP エントリに次の属性を追加するとします。

```
cn=Proctor, Goodman, and Jones
description=Friends of the company; Rationalists
```

cfldap タグを次のように使用します。

```
<cfldap action="modify"
    modifyType="add"
    attributes="cn=Proctor, Goodman, and Jones: description=Friends of the company;
        Rationalists"
    dn="uid=goodco, ou=People, o=Airius.com"
    separator="&"
    delimiter=":"
    server=#myServer#
    username=#myUserName#
    password=#myPassword#>
```

cfldap 出力の使用

LDAP データから、検索可能な コレクションを作成できます。

cfldap クエリーで複雑なデータが返される場合は、クエリーからクエリーを生成する機能を使用すると便利です。クエリーオブクエリーの詳細については、433 ページの「クエリーオブクエリーの使用」を参照してください。

ディレクトリスキーマの表示

LDAP v3 のディレクトリスキーマ情報は、ルート DN の特別なエントリに格納されています。この情報にアクセスするには、ディレクトリルートの subschemaSubentry 属性を使用します。

次の ColdFusion クエリーでは、ディレクトリスキーマを取得して表示する方法を示します。これを実行すると、スキーマのオブジェクトクラスや属性タイプの定義情報が表示されます。オブジェクトクラスについては、クラス名、上位クラス、必須属性タイプ、およびオプション属性タイプが表示されます。属性タイプについては、タイプ名、タイプの説明、および単一値か多値かが表示されます。

この例では、スキーマのすべての情報は表示されません。たとえば、一致ルールは表示されません。また、オブジェクトクラス ID、属性タイプ ID、属性タイプシンタックス ID、およびオブジェクトクラスの説明も表示されません。オブジェクトクラスの説明は、すべて "Standard Object Class" という値になります。

注意：LDAP サーバーのスキーマを表示するには、サーバーが LDAP v3 をサポートしている必要があります。

この例は、iPlanet Directory Server 5.0 では実行できません。4.x サーバーでは動作します。

LDAP ディレクトリのスキーマの表示

1 次のようなファイルを作成します。

```
<html>
<head>
    <title>LDAP Schema</title>
</head>

<body>
<!--- Start at Root DSE to get the subschemaSubentry attribute. --->
<cfldap
    name="EntryList"
    server="ldap.mycorp.com"
    action="query"
    attributes="subschemaSubentry"
    scope="base"
    start="">

<!--- Use the DN from the subschemaSubEntry attribute to get the schema. --->
<cfldap
    name="EntryList2"
    server="ldap.mycorp.com"
    action="query"
    attributes="objectclasses, attributetypes"
    scope="base"
    filter="objectclass=*"
    start=#entryList.subschemaSubentry#>

<!--- Only one record is returned, so query loop is not required. --->
<h2>Object Classes</h2>
<table border="1">
    <tr>
        <th>Name</th>
        <th>Superior class</th>
        <th>Must have</th>
        <th>May have</th>
    </tr>
    <cfloop index = "thisElement" list = #Entrylist2.objectclasses#>
        <cfscript>
            thiselement = Trim(thisElement);
            nameloc = Find("NAME", thisElement);
            descloc = Find("DESC", thisElement);
            suploc = Find("SUP", thisElement);
            mustloc = Find("MUST", thisElement);
            mayloc = Find("MAY", thisElement);
            endloc = Len(thisElement);
        </cfscript>
        <tr>
            <td><cfoutput>#Mid(thisElement, nameloc+6, descloc-nameloc-8)#
                </cfoutput></td>
            <cfif #suploc# NEQ 0>
                <td><cfoutput>#Mid(thisElement, suploc+5, mustloc-suploc-7)#
                </cfoutput></td>
```

```
            <cfelse>
                <td>NONE</td>
            </cfif>
            <cfif #mayloc# NEQ 0>
                <td><cfoutput>#Replace(Mid(thisElement, mustloc+6,
                    mayloc-mustloc-9), " $ ", ", ", "all")#</cfoutput></td>
                <td><cfoutput>#Replace(Mid(thisElement, mayloc+5, endloc-mayloc-8),
                    " $ ", ", ", "all")#</cfoutput></td>
            <cfelse>
                <td><cfoutput>#Replace(Mid(thisElement, mustloc+6,
                    endloc-mustloc-9), " $ ", ", ", "all")#</cfoutput></td>
                <td>NONE</td>
            </cfif>
        </tr>
    </cfloop>
</table>
<br><br>

    <h2>Attribute Types</h2>
<table border="1" >
    <tr>
        <th>Name</th>
        <th>Description</th>
        <th>multivalued?</th>
    </tr>
    <cfloop index = "thisElement"
        list = #ReplaceNoCase(EntryList2.attributeTypes, ", alias", "<br> Alias",
        "all")# delimiters = ",">
        <cfscript>
            thiselement = Trim(thisElement);
            nameloc = Find("NAME", thisElement);
            descloc = Find("DESC", thisElement);
            syntaxloc = Find("SYNTAX", thisElement);
            singleloc = Find("SINGLE", thisElement);
            endloc = Len(thisElement);
        </cfscript>
        <tr>
            <td><cfoutput>#Mid(thisElement, nameloc+6, descloc-nameloc-8)#
                </cfoutput></td>
            <td><cfoutput>#Mid(thisElement, descloc+6, syntaxloc-descloc-8)#
                </cfoutput></td>
            <cfif #singleloc# EQ 0>
                 <td><cfoutput>Yes</cfoutput></td>
            <cfelse>
                 <td><cfoutput>No</cfoutput></td>
            </cfif>
        </tr>
    </cfloop>
</table>
</body>
</html>
```

2 サーバー ldap.mycorp.com を、使用している LDAP サーバーに変更します。cfldap タグで、ユーザー ID およびパスワードの指定が必要な場合もあります。

3 このテンプレートに ldapschema.cfm という名前を付けて Web のルートディレクトリの下の myapps に保存し、ブラウザで表示します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
<cfldap name="EntryList" server="ldap.mycorp.com" action="query" attributes="subschemaSubentry" scope="base" start="">	ディレクトリサーバーのルートから subschemaSubentry 属性の値を取得します。値はスキーマの DN です。
<cfldap name="EntryList2" server="ldap.mycorp.com" action="query" attributes="objectclasses, attributetypes" scope="base" filter="objectclass=*" start=#entryList.subschemaSubentry#>	スキーマ DN を使用して、スキーマから objectclasses および attributetypes 属性を取得します。

コード	説明

Object Classes
<table border="1">
<tr>
<th>Name</th>
<th>Superior class</th>
<th>Must have</th>
<th>May have</th>
</tr>
<cfloop index = "thisElement" list = #Entrylist2.objectclasses#>
<cfscript>
thiselement = Trim(thisElement);
nameloc = Find("NAME", thisElement);
descloc = Find("DESC", thisElement);
suploc = Find("SUP", thisElement);
mustloc = Find("MUST", thisElement);
mayloc = Find("MAY", thisElement);
endloc = Len(thisElement);
</cfscript>
	各オブジェクトクラスの名前、上位クラス、必須属性、およびオプション属性をテーブルに表示します。
	スキーマには、すべてのオブジェクトクラスの定義がカンマ区切りのリストとして含まれているので、リストタイプの cfloop タグを使用します。
	thisElement 変数にはオブジェクトクラスの定義が含まれます。前後のスペースをすべて削除したら、Find 関数でクラス定義フィールドのキーワードを使用して、オブジェクトクラス ID を含む必要なフィールドの開始位置を取得します (ID は表示されません)。
	後で計算に使用する thisElement 文字列の長さを取得します。

<tr> <td><cfoutput>#Mid(thisElement, nameloc+6, descloc-nameloc-8)# </cfoutput></td> <cfif #suploc# NEQ 0> <td><cfoutput>#Mid(thisElement, suploc+5, mustloc-suploc-7)# </cfoutput></td> <cfelse> <td>NONE</td> </cfif> <cfif #mayloc# NEQ 0> <td><cfoutput>#Replace(Mid(thisElement, mustloc+6, mayloc-mustloc-9), " $ ", ", ", "all")#</cfoutput></td> <td><cfoutput>#Replace(Mid(thisElement, mayloc+5, endloc-mayloc-8), " $ ", ", ", "all")#</cfoutput></td> <cfelse> <td><cfoutput>#Replace(Mid(thisElement, mustloc+6, endloc-mustloc-9), " $ ", ", ", "all")#</cfoutput></td> <td>NONE</td> </cfif> </tr> </cfloop> </table>	フィールド値を表示します。Mid 関数を使用して、thisElement 文字列から個別のフィールド値を抽出します。 最上位のオブジェクトクラスには、上位オブジェクトクラスのエントリがありません。この場合に対処するために、suploc 位置変数をテストします。値が 0 でない場合は通常どおりに処理し、値が 0 の場合は "NONE" を出力します。 オプション属性がないこともあります。この場合も上位クラスと同様に処理します。必須属性の位置の計算では、オプション属性のフィールドがあればその位置を使用します。ない場合はオブジェクトクラス定義文字列の末尾を使用します。

Attribute Types
<table border="1" >
<tr>
<th>Name</th>
<th>Description</th>
<th>multivalued?</th>
</tr>
<cfloop index = "thisElement" list = #ReplaceNoCase(EntryList2.attributeTypes, ", alias", "
 Alias", "all")# delimiters = ",">
<cfscript>
thiselement = Trim(thisElement);
nameloc = Find("NAME", thisElement);
descloc = Find("DESC", thisElement);
syntaxloc = Find("SYNTAX", thisElement);
singleloc = Find("SINGLE", thisElement);
endloc = Len(thisElement);
</cfscript>
<tr>
<td><cfoutput>#Mid(thisElement, nameloc+6, descloc-nameloc-8)#
</cfoutput></td>
<td><cfoutput>#Mid(thisElement, descloc+6, syntaxloc-descloc-8)#
</cfoutput></td>
<cfif #singleloc# EQ 0>
<td><cfoutput>Yes</cfoutput></td>
<cfelse>
<td><cfoutput>No</cfoutput></td>
</cfif>
</tr>
</cfloop>
	属性タイプでも、オブジェクトクラスと同じような計算を行います。
	属性タイプフィールドには、", alias for...." というテキストが含まれていることがあります。このテキストには、属性エントリの区切り文字としても使用されるカンマが含まれています。REReplaceNoCase 関数を使用して、"alias" という単語の前にあるカンマを、HTML の
	 タグに置き換えます。
	属性定義には、数値シンタックス識別子が含まれています。これは、ページには表示していませんが、他のフィールドの位置を計算するためにその位置を使用しています。

リファラル

LDAP データベースは、複数のサーバーに分散できます。LDAP v3 標準には、現在のサーバー上にリクエストした情報がない場合に、別のサーバーへのリファラルをサーバーからクライアントに返すメカニズムが用意されています。この機能は、LDAP v2 対応のサーバーにも装備されている場合があります。

ColdFusion ではリファラルを自動的に処理できます。cfldap タグの referral 属性にゼロ以外の値を指定すると、リファラルで指定されたサーバーにリクエストが送られます。

referral 属性値には、リクエストで許容するリファラル数を指定します。たとえば、referral 属性が 1 で、サーバー A がサーバー B へのリファラルを送信し、さらにサーバー B がサーバー C へのリファラルを送信すると、ColdFusion はエラーを返します。referral 属性が 2 で、サーバー C に情報があれば、LDAP リクエストは成功します。使用する値は、分散 LDAP ディレクトリのトポロジや、応答速度の重要性、完全なレスポンスの重要性によって異なります。

ColdFusion でリファラルが処理される場合に、新しいサーバーへのリクエストに cfldap タグのログイン情報を使用するかどうかは、rebind 属性によって指定します。デフォルトでは使用しない設定になっており、サーバーには匿名ログインが送信されます。

LDAP セキュリティの管理

LDAP セキュリティの実装を検討する場合は、サーバーセキュリティとアプリケーションセキュリティを検討します。

サーバーセキュリティ

cfldap タグは、SSL (Secure Socket Layer) v2 セキュリティをサポートしています。このセキュリティでは、証明書ベースで LDAP サーバーの検証が行われます。また、ColdFusion サーバーと LDAP サーバー間でやり取りするデータ (ユーザーのパスワードを含む) が暗号化され、サーバー間でやり取りするデータの安全性が確保されます。SSL v2 セキュリティを指定するには、cfladap タグの secure="cfssl_basic" 属性を設定します。

LDAP サーバーのセキュリティについて

ColdFusion は、Java Native Directory Interface (JNDI)、LDAP プロバイダ、および SSL パッケージを使用して、SSL 通信のクライアントサイドを作成します。通信のサーバーサイドは LDAP サーバーです。cfldap タグで SSL を使用して接続する LDAP サーバーは、SSL サーバー証明書を持っています。これは、認証機関によって安全に " 署名 " された、送信者を識別 (認証) するための証明書です。LDAP サーバーは、SSL 接続の最初の段階で、サーバー証明書をクライアントに提示します。クライアントがこの証明書を信頼すると、SSL 接続が確立され、安全な LDAP 通信が開始されます。

サーバーを信頼するかどうかを ColdFusion が判断するときには、ColdFusion が使用している JRE の jre/lib/security/cacerts キーストア内の情報と、サーバーの資格情報が比較されます。ColdFusion のデフォルトの cacerts ファイルには、数多くの認証機関についての情報が含まれています。このファイルに情報を追加して更新する必要がある場合は、ColdFusion の "jre/bin" ディレクトリにある keytool ユーティリティを使用して、X.509 形式の証明書をインポートできます。たとえば、次の行を入力します。

```
keytool -import -keystore cacerts -alias ldap -file ldap.crt -keypass bl19mq
```

keytool ユーティリティの初期キーパスパスワードは、"change it" です。keytool ユーティリティの使用方法の詳細については、Sun JDK のマニュアルを参照してください。

サーバーとの安全な通信が確立されたら、サーバーにログイン資格情報を提供する必要があります。ログイン資格情報は、cfldap タグの username 属性と password 属性で指定します。サーバーがログイン資格情報を有効と判断すると、ColdFusion がディレクトリにアクセスできるようになります。

LDAP セキュリティの使用

セキュリティを使用するには、まず、LDAP サーバーが SSL v2 をサポートしていることを確認します。

cfldap タグの secure 属性を次のように指定します。

```
secure = "cfssl_basic"
```

次に例を示します。

```
<cfldap action="modify"
    modifyType="add"
    atributes="cn=Lizzie"
    dn="uid=lborden, ou=People, o=Airius.com"
    server=#myServer#
    username=#myUserName#
    password=#myPassword#
    secure="cfssl_basic"
    port=636>
```

port 属性に、セキュリティ保護された LDAP 通信に使用するサーバーポートを指定します。このポートは、デフォルトで 636 に設定されています。ポートを指定しないと、デフォルトのセキュリティ保護されていない LDAP ポート 389 が使用されます。

アプリケーションのセキュリティ
アプリケーションのセキュリティ保護では、cfldap タグで使用するパスワードが部外者に盗まれないようにする必要があります。この最善の方法は、username 属性と password 属性に変数を使用することです。変数の設定は、暗号化されたアプリケーションページで行うようにします。アプリケーションのセキュリティの詳細については、337 ページの「アプリケーションの保護」を参照してください。

Solr 検索のサポート

Solr 検索サービスは、Lucene Java 検索ライブラリをベースとした、オープンソースのエンタープライズ向け検索サーバーです。

Solr 検索サービス

Solr は、Lucene をベースとした全文検索エンジンであり、次の特徴があります。

- XML/HTTP インターフェイス
- 型とフィールドを定義するスキーマが柔軟
- Web 管理インターフェイス
- 広範なキャッシュ機能
- インデックス複製
- 拡張性の高いオープンアーキテクチャ
- Java 5 で記述されており、WAR としてデプロイ可能
- 語尾変化のサポート
- MS Office 2007 ファイル形式のサポート

ColdFusion のインストーラを実行すると、Solr Web アプリケーションを含む ColdFusion 9 Solr サービスが自動的に作成されます。UNIX および Linux の場合は、Solr シェルスクリプトの開始と停止を手動で行う必要があります。

Solr の設定に使用する CFML タグ

cfcollection このタグを使用してコレクションを作成するには、engine="solr" とともに action="create"/"list"/"map" を指定します。Solr コレクションを使用する場合、言語の指定は不要です。言語は <cfindex> タグで指定できます。その他の属性はすべて同じままです。

cfindex ColdFusion では、コレクション名に基づいて検索エンジンが検出されます。

サポート対象言語

Solr では次の言語がサポートされています。

- デンマーク語
- オランダ語
- フィンランド語
- フランス語
- ドイツ語
- イタリア語
- ノルウェー語
- スペイン語
- ポルトガル語
- ロシア語
- スウェーデン語
- 中国語
- 日本語
- 韓国語
- チェコ語
- ギリシャ語
- タイ語

Solr では、あらゆる言語のドキュメントがサポートされます。ここに掲載されていない言語 (アラビア語など) がドキュメントに含まれていても、Solr では内容のインデックスを作成できます。ただし、語尾変化には対応できません。このような場合は、cfindex タグで language 属性を指定しないでください。

Solr では語尾変化がサポートされます。つまり、検索語の原形 (語幹) が考慮されます。この機能は、language 属性が指定されている場合にのサポートされます。

Solr 検索の例

次の例は、Solr を使用して検索処理を行う際の検索シンタックスを示しています。

- 単一語検索 : 次の例は、コレクション内で単一の語を検索する方法を示しています。

```
<cfsearch name="qsearch1" collection="solr_complex" criteria="Graphics">
```

- 複合語検索 : 次の例は、"ColdFusion" および "Green" という語を含むドキュメントまたはクエリーを検索する方法を示しています。

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria="+Green +Coldfusion">
```

- 少なくとも 1 つの語と一致する箇所の検索 : 次の例は、少なくとも "Coldfusion" OR (Green OR Blue) に一致するものを検索する方法を示しています。

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" +Coldfusion Green Blue">
```

- 1 つの語のみに一致、他の語には一致しない箇所の検索 : 次の例は、"Green" かつ NOT "Coldfusion" の条件に一致するものを検索する方法を示しています。

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" -Coldfusion +Green">
```

- あいまい検索 : 次の例は、"roam"、"roams"、"foam"、"foams" などの語を検索する方法を示しています。

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" roam~">
```

"roam" のあいまい検索は、次の方法でも実行できます。

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria="roam~">
```

"roam" との類似度が高い語の検索 :

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" roam~0.8" >
```

- ワイルドカード検索 : 次のシンタックスを使用すると、"test"、"text"、"teat" などの語を検索できます。

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" te?t">
```

次の例では、"test"、"text"、"teeeeeext"、"texyzt" などが検索されます。

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" te*t">
```

注意 : 検索の 1 文字目に * および疑問符 (?) を使用することはできません。

- 近接検索 : 近接する 5 つの単語内で "apache" および "jakarta" を検索するには、次の検索を使用します。

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria='"jakarta apache" ~10'>
```

- 範囲検索 : 次のシンタックスを使用すると、タイトルが "fuzzy1.txt" から "text1.txt" までの範囲に入るすべてのドキュメントが検索されます。

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria="title:fuzzy1.txt TO text1.txt">
```

変更日が特定の範囲に入るドキュメントを検索するには :

```
<cfsearch name="qsearch"
collection="solr_srch"
criteia="modified:20080101 TO 20500101">
```

開始項目と終了項目も、その範囲内に含まれます。これらの項目を範囲から除外するには、代わりに中括弧 {} を使用します。

- フィールド検索 : タイトルに "fuzzy1.txt" を含むドキュメントを検索には、次のシンタックスを使用します。

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria="title:fuzzy1.txt">
```

タイトルに "fuzzy1.txt" または "fuzzy2.txt" を含むドキュメントを検索するには :

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria="title:fuzzy?.txt">
```

次のシンタックスを使用して同じ検索を実行することもできます。

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria=' ""title:fuzzy1.txt"" OR ""title:fuzzy2.txt"" '>
```

この他に、次のシンタックスを使用して検索することもできます。

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria="title:(test* +fuzzy1*)">
```

- 文字列検索 :

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria='"Cold Fusiongava" OR "Internet Tools"'>
```

- 同義語の検索 : "MB"、"megabyte"、"gig" などの同義語を含むドキュメントを検索する方法には、次の 2 つがあります。

  1 コレクションをまだ作成していない場合は、次のファイルを参照してください。

  <ColdFusion のホームディレクトリ >/solr/multicore/template/conf/synonyms.txt

  このファイルには、'GB, gig, gigabyte, gigabytes' などのデフォルトのマッピングがいくつか記述されています。独自の同義語マッピングを定義する場合は、それ以降の行に記述します。

  2 既に作成したコレクションの同義語マッピングを追加する場合は、"<collection_location>/conf/synonyms.txt" を開いて、マッピングを定義します。

  マッピングの定義が終わったら、Solr サーバーを再起動します。

検索語の強調表示

次のコードが示すように、Solr のデフォルトの設定では、サマリーコンテンツ内の検索語が強調表示されます。

```
<cfsearch
collection="syn1"
criteria="Services solr"
name="results"
status="r"
suggestions="always"
contextPassages="1">
```

ドキュメント全体の内容を強調表示するには、solrconfig.xml ファイルと schema.xml ファイルを変更します。これらのファイルは次の場所にあります。

- < コレクションディレクトリ >/conf : 将来作成されるすべての Solr コレクションに変更を適用するには、この場所にあるファイルを変更します。
- <Solr のホーム >/multicore/template/conf : 特定のコレクションだけに変更を適用するには、この場所にあるファイルを変更します。

1 Solr サービスを停止します。

2 solrconfig.xml の次のセクションを「置換後 :」の内容で置き換えます。

```
<requestHandler name="standard" class="solr.StandardRequestHandler" default="true">
    <!-- default values for query parameters -->
     <lst name="defaults">
       <str name="echoParams">explicit</str>
       <str name="hl.fl">summary title </str>
       <!-- omp = Only More Popular -->
       <str name="spellcheck.onlyMorePopular">false</str>
       <!-- exr = Extended Results -->
       <str name="spellcheck.extendedResults">false</str>
       <!--  The number of suggestions to return -->
       <str name="spellcheck.count">1</str>

     </lst>
     <arr name="last-components">
      <str>spellcheck</str>
     </arr>
  </requestHandler>
```

置換後 :

```
<requestHandler name="standard" class="solr.StandardRequestHandler" default="true">
    <!-- default values for query parameters -->
     <lst name="defaults">
       <str name="echoParams">explicit</str>
       <str name="hl.fl">contents title </str>
       <!-- omp = Only More Popular -->
       <str name="spellcheck.onlyMorePopular">false</str>
       <!-- exr = Extended Results -->
       <str name="spellcheck.extendedResults">false</str>
       <!--  The number of suggestions to return -->
       <str name="spellcheck.count">1</str>

     </lst>
     <arr name="last-components">
      <str>spellcheck</str>
     </arr>
  </requestHandler>
```

3 schema.xml の次のセクションを「置換後 :」の内容で置き換えます。

```
<field name="contents" type="text" indexed="true"  stored="false" required="false" multiValued="true"
omitNorms="true"/>
```

置換後 :

```
<field name="contents" type="text" indexed="true" stored="true" required="false" multiValued="true"
omitNorms="true"/>
```

4 Solr を再起動します。

5 コレクションのインデックスを再作成します。

注意 : solrconfig.xml と schema.xml を変更するとインデックスのサイズが大きくなります。

Solr メモリの設定

Solr で使用するメモリ量を増やすには、次の手順を実行します。

Windows 以外のプラットフォーム

1 次のコマンドを使用して Solr を停止します。

```
sudo ./cfsolr stop
```

2 cfsolr スクリプトで、VMARGS= から始まる行を探し、デフォルト値の -Xmx256m を適切な値に変更します。たとえば、-Xmx1024m に変更します。

3 次のコマンドを使用して Solr を開始します。

sudo ./cfsolr start

Windows プラットフォーム

1 Solr サービス (ColdFusion 9 Solr Service) を停止します。

2 solr.lax で、lax.nl.java.option.additional= から始まる行を探し、デフォルト値の -Xmx256m を適切な値に変更します。たとえば、-Xmx1024m に変更します。

3 Solr サービスを開始します。

ColdFusion 9.0.1 での Solr の機能強化

全体的なインデックス作成の精度向上とは別に、次の機能が強化されています。

- すべてのドキュメントにつき、正しい MIME タイプが表示されます。
- mp3 や JPEG などのバイナリファイルのメタデータのインデックス作成のサポートが強化されています。
- previousCriteria 属性が (cfsearch タグで) サポートされています。
- cfindex タグと cfsearch タグの両方で、categoryTree 属性がサポートされています。
- ドキュメント全体で用語の強調表示を有効／無効にする新しいセクションが、ColdFusion Administrator に追加されました。これは、Solr がローカルマシンで実行中の場合にのみ適用されます。

  注意：次の手順は、Solr がローカルマシンで実行中の場合にのみ行ってください。Solr をリモートマシンで実行中の場合は、『ColdFusion 9 アプリケーションの開発』の「検索語の強調表示」セクションに記載されている手順に従ってください。

  1 ColdFusion Administrator で、[ColdFusion コレクション] に移動して、Solr コレクションをクリックします。

  2 [有効にする] または [無効にする] ボタンを使用して、検索語の強調表示を有効または無効にします。

  3 コレクションのインデックスを再作成します。

  注意：検索語の強調表示を有効にすると、Solr コレクションのサイズが増加します。そのため、検索語の強調表示を有効にする場合は、Solr に十分なメモリを割り当てるようにしてください。

Solr のアップグレード

アップデータの一部で、Solr がアップグレードされています。

そのため、ColdFusion 9.0.1 で Solr を使用する場合は、Solr をアップグレードしてください。

ローカルインストールの場合は、ColdFusion アップデータを実行すると、自動的に Solr がアップグレードされます。

リモートインストールの場合は、次の手順に従って、Solr を手動でアップグレードしてください。

1 Solr を停止します。

2 **Solr_Home**/multicore にある solr.xml のバックアップを取ります。

3 Solr をアンインストールします。

4 Adobe のダウンロードサイトから入手可能なスタンドアローン版の Solr を再インストールします。

5 Solr を停止します (Solr が自動的に開始された場合)。

6 バックアップした solr.xml のコピーを **Solr_Home**/multicore に戻します。

注意：アップグレード後は、Solr コレクション全体のインデックスを再作成してから、検索サービスを使用してください。

ColdFusion 10 での Solr の機能強化

このリリースでの機能拡張により、次の操作が可能になります。

- データベースのインデックス作成でのデータ読み込みハンドラーの使用
- 動的なカスタムフィールドに基づいたインデックス作成および検索
- 個別のコレクションのリロード
- 検索用の言語の追加
- ColdFusion Administrator を使用した検索システムのセキュリティ保護（データとサービス／ Solr サーバー／詳細設定の表示／ HTTPS 接続を使用）
- インデックスを作成したドキュメントの自動コミット
- 特定のフィールドまたはドキュメント全体の検索結果向上の促進

ファイルの場所とファイル名の変更

- スタンドアローンインストールでは、すべての Solr ファイルは jetty フォルダー ColdFusion Zeus¥cfusion¥jetty に格納されます（以前は ColdFusion Zeus¥cfusion¥solr）。
- Windows では、Solr サービスの名前が ColdFusion Zeus Jetty Service に変更されました（以前は ColdFusion Zeus Solr Service）。
- Windows では、実行可能ファイルの名前が jetty.exe に変更されました（以前は solr.exe）。

データ読み込みハンドラーの使用

ColdFusion 9 では、データベースのインデックス作成は 2 つの手順のプロセスでした（cfquery タグを使用したデータベースのクエリと、cfindex タグを使用したインデックス作成）。ColdFusion 10 では、cfquery を使用してデータを取得する必要はありません。Solr がデータベースと直接通信し、インデックス作成のパフォーマンス向上に役立つデータ読み込みハンドラーを使用してデータを取得します。

要件に応じて、完全な、または部分的なインデックス作成を実行できます。例えば、初めてデータベースのインデックスを作成するときは、完全なインデックス作成を行います。データベースを更新したときは、部分的なインデックス作成を実行してコレクションを更新します。

データ読み込みハンドラーを使用したインデックス作成

次の手順で、データベースのインデックス作成用にデータ読み込みハンドラーを設定できます。

1 次の手順に従います。

- **完全な読み込みの場合：**次の dataconfig.xml を作成し、データベーステーブルの列と Solr のマッピングを定義します。

```
<dataconfig>
    <datasource driver="org.hsqldb.jdbcDriver" url="jdbc:mysql:/temp/example/ex" user="user"
                password="user"/>
    <document name="products">
        <entity name="item" query="select * from item">
            <field column="ID" name="id"/>
            <field column="NAME" name="name"/>
        </entity>
    </document>
</dataconfig>
```

- **差分読み込みの場合：**次の dataconfig.xml を作成します。

```
<dataconfig>
    <dataSource
    driver="com.mysql.jdbc.Driver" "jdbc:mysql:/temp/example/ex" user="user" password="password" />
    <document name="rrr">
    <entity name="item" pk="ID" query="select ID,NAME,PRICE,WEIGHT,last_modified  from item"
            deltaimportquery="select ID,NAME,PRICE,WEIGHT,last_modified from item where
ID='${dataimporter.delta.id}'"
           deltaquery="select id from item where last_modified > '${dataimporter.last index_time}'">
        <field column="ID" name="uid"/>
        <field column="NAME" name="name_t"/>
        <field column="PRICE" name="price_f"/>
        <field column="WEIGHT" name="weight_d"/>
        <entity name="feature" pk="ITEM_ID"
                query="select description as features from feature where item_id='${item.ID}'">
            <field name="features_t" column="features"/>
        </entity>
        <entity name="item_category" pk="ITEM_ID, CATEGORY_ID"
                query="select CATEGORY_ID from item_category where ITEM_ID='${item.ID}'">
            <entity name="category" pk="ID"
                    query="select description as cat from category where id =
'${item_category.CATEGORY_ID}'">
                <field column="cat" n a m
                       e="cat t"/>
            </entity>
        </entity>
    </entity>
</dataconfig>
```

属性について詳しくは、http://wiki.apache.org/solr/DataImportHandler の「Configuration in data-config.xml」セクションの「**Schema for the data config**」を参照してください。

- last_modified がインデックスを作成するテーブルの列名であり、その列がタイムスタンプであることを確認してください。
- この列をマッピングしないと、部分的な読み込みは失敗します。
- 最新のタイムスタンプは、コレクションの場所で使用できる dataimport.properties で作成されます。

2 作成したコレクションの conf ディレクトリにファイルを保存します。

3 solrconfig.xml（conf ディレクトリにあります）で、次のセクションのコメントを外します。

```
<!--
  <requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">
    <lst name="defaults">
      <str name="config">data-config.xml</str>
    </lst>
  </requestHandler>
  -->
```

これでデータ読み込みハンドラーが有効になります。

4 コレクションをリロードします。

5 cfindex アクション fullImport、deltaImport、status または abort のいずれかを使用します。

cfindex タグに対する変更

type 属性の新しい値

データ読み込みハンドラーを使用するには、type=dih を指定します。

新しいアクション

Solr がデータベースから直接データを取得できるように、次の新しいアクションが、cfindex タグに追加されました。

- fullimport：データベースの完全なインデックスを作成します。初めてデータベースのインデックスを作成するときなどが該当します。

  次に例を示します。

  ```
  <cfindex collection="dih1" type="DIH" action="fullimport" status="st">
  <cfsearch collection="dih1" criteria="damaged" name="s" orderby="price_f desc" status="stat">
  ```

- deltaimport：インデックスを部分的に作成します。例えば、データベースを更新したときは、完全な読み込みの代わりに、差分読み込みを実行してコレクションを更新できます。

  次に例を示します。

  ```
  <cfindex collection="dih1" type="DIH" action="deltaimport" status="st">
  <cfsearch collection="dih1" criteria="damaged" name="s" orderby="price_f desc" status="stat">
  ```

- status：インデックス作成のステータスを提供します。処理されたドキュメントの総数、およびアイドルや実行中といったステータスなどです。

  次に例を示します。

  ```
  <cfindex collection="bt" type="DIH" action="status" status="s">
  <cfoutput>
      Rows Indexed : #s.TOTAL_DOCUMENTS_PROCESSED#
      <br>
  </cfoutput>
  <cfoutput>
      Status of Solr Server : #s.status#
      <br>
  </cfoutput>
  ```

- abort：進行中のインデックス作成タスクを中止します。

  ```
  <cfindex collection="bt" type="DIH" action="abort" status="s">
  <cfoutput>
      Status of Solr Server : #s.status#
      <br>
  </cfoutput>
  ```

カスタムデータの保存

インデックスの作成に加えて、動的に定義されるカスタムフィールドを使用してカスタム情報を保存できます。

例えば、PDF のインデックス作成中に、次の例のようにして、作成者や発行日などの情報を保存できます。

```
<cfindex collection="CodeColl"
    action="refresh"
    type="file"
    key="C:\learning_resources\wwwroot\vw_files\gettingstarted.pdf"
    urlpath="http://localhost:8500/vw_files/"
    language="English"
    title="Cfindex Reference page"
    status="info"
    blurb_s=information
    publisher_s=adobe/>
```

カスタムフィールドを指定するには、次のシンタックスを使用します。

```
 <cfindex ...
    datefield_dt=#date1#
    column_i=#secondaryColumn#
    body=#primaryColumn#
    ....../>
```

注意：カスタムフィールドには小文字のみを含めることができます。

このコードで、_i は値を保存してインデックスを作成する整数カスタムデータを示します。_i で終わるすべてのフィールド名は、Solr の整数として扱われます。

同様に、_s は文字列のカスタムデータを示します。

サポートされるすべてのデータ型は、schema.xml に列記されています。

```
<dynamicfield name="*_i" type="sint" indexed="true" stored="true"/>
<dynamicfield name="*_s" type="string" indexed="true" stored="true"/>
<dynamicfield name="*_l" type="slong" indexed="true" stored="true"/>
<dynamicfield name="*_t" type="text" indexed="true" stored="true"/>
<dynamicfield name="*_b" type="boolean" indexed="true" stored="true"/>
<dynamicfield name="*_f" type="sfloat" indexed="true" stored="true"/>
<dynamicfield name="*_d" type="sdouble" indexed="true" stored="true"/>
<dynamicfield name="*_dt" type="date" indexed="true" stored="true"/>
<dynamicfield name="random*" type="random"/>
```

注意：_dt は、ColdFusion でサポートされる日付形式のみをサポートします。

例

```
<cfindex collection="custom1" type="file" action="update" key="#datadir#/text/text1.txt"
         status="s" date_dt=NOW>
<cfsearch collection="custom1" criteria="" name="n" orderby="title">
```

cfsearch の新しい属性 orderBy

新しい属性 orderBy が cfsearch に追加されています。この属性は、カスタムフィールドの列をランク順にソートします。これはオプションの属性であり、デフォルトでは昇順にソートされます。

```
<cfsearch
collection="someCollection"
criteria="someCriteria"
orderby="field1 desc,field2,field3 asc"  ...../>
```

インデックスを作成したドキュメントの自動コミット

cfindex で属性 autoCommit を true に設定すると、変更が検索サーバーに自動的にコミットされます。次に例を示します。

```
<cfindex collection="autocommit_check" action="update" type="file" key="#Expandpath(".")#/_boost1.txt"
first_t="fieldboost" second_t="secondfield" fieldboost="first_t:1,second_t:2" docboost="6"
autocommit="true">
```

false の場合は、cfindex action="commit" を使用して明示的にコミットしない限り、インデックスを作成したドキュメントはコミットされません。デフォルト値は true です。

検索結果のランキングの向上

cfindex の次の属性は、検索結果のランキングの向上に役立ちます。

- fieldBoost：インデックス作成中に特定のフィールドのランクを上げます。

  fieldBoost は、フィールドのスコアを上げることにより、検索結果でのランキングを上げます。カンマ区切りのリストとして値を指定することで、複数のフィールドのランクを上げることができます。

- docBoost：インデックス作成中にドキュメント全体のランクを上げます。

  docBoost は、ドキュメントのスコアを上げることにより、検索結果でのランキングを上げます。

ColdFusion 9 からの変更

- ColdFusion 9 では、カスタムフィールドのサポートは custom1、custom2、custom3、custom4 だけに制限されていました。ColdFusion 10 では、カスタムフィールドは動的です。

- ColdFusion 9 では、すべてのカスタムフィールドが表示されます。ColdFusion 10 では、cfdump はデータがあるフィールドのみを生成します。つまり、custom 1 および custom 2 だけを指定した場合は、これら 2 つのフィールドだけが表示されます。
- 次のようなコードについて考えます。

  ```
  <cfsearch criteria='some_criteria and column_i: [ 10 - 20 ]'...>
  ```

  ここで、some_criteria はフィルター処理を示します。例えば、column_i: [10 - 20] は値が 10 ～ 20 の範囲であるすべての項目を検索します。column_i は、インデックスの作成中にユーザーが指定するカスタムフィールドです。

  このオプションは ColdFusion 9 でも使用できましたが、4 つのカスタムフィールドに限定されていました。ColdFusion 10 では制限はありません。
- ColdFusion 10 では、受け取る検索結果の順序をソートできます。

注意：Solr コレクションで文字列型のフィールドを検索するときは、条件を引用符で囲む必要があります。次に例を示します。

criteria='string_s:"something missing"'

Solr 検索の例 1

```
<cfsearch collection="custom1" criteria="rank_i:[2 TO 4]" name="s1" orderby="value_i"
          status="s">
<cfdump var="#s1#">
```

Solr 検索の例 2：ワイルドカードの使用

```
<!------ Searching with wildcard *--------->
<cfsearch collection="custom1" criteria="text1_t:blue*" name="s1" status="s"
          orderby="cf_i">
<cfoutput>
    Searching Text with wildcard * :#s.FOUND#
    <br>
</cfoutput>
```

検索の制限事項

制限事項：文字列型のカスタムフィールドの検索

文字列は、* 以外のワイルドカードでは検索できません。文字列がトークン化されていないため、文字列内の任意の単語の検索はできません。文字列は全体としてのみ検索でき、個別の単語では検索できません。

例えば、str_s="All work and no play" の場合、この文字列内の play や work は検索できません。文字列全体を使用して検索を実行する必要があります。ただし、検索で文字列をソートすることはできます（orderby 属性を使用）。

制限事項：テキスト型のカスタムフィールドの検索

テキスト型のフィールドはトークン化されているので、テキスト内の任意の単語で検索できます。また、ワイルドカードを使用してテキストを検索することもできます。唯一の制限は、テキスト型は検索中にソートできないことです。テキスト型はトークン化されているので、Solr はテキストをトークンのセットとして扱い、したがってソートはできません。

制限事項：カスタムフィールドの検索では大文字と小文字が区別される

カスタムフィールドは小文字でのみ検索できます。例えば、カスタムフィールドの名前が newDate の場合、newdate を検索する必要があります。

制限事項：orderBy 属性の使用

orderBy 属性は、文字列のようなトークン化されていないフィールドで使用する必要があります。

コレクションのリロード

ColdFusion 9 では、個別のコレクションをリロードするには、Solr を再起動する必要があります。それによって、すべてのコレクションがリロードされます。したがって、言語やフィールドタイプの追加、またはデータ読み込みハンドラーの有効化など、schema.xml を変更したときは常に、変更を有効にするために Solr を再起動する必要があります。

ColdFusion 10 では、リロードを特定のコレクションに制限できるので、パフォーマンスが大きく向上します。

コレクションをリロードするには、

1 ColdFusion Administrator で、データとサービス／ ColdFusion コレクションに移動します。

2 特定のコレクションで、Solr コレクション／アクションのリロードアイコンをクリックします。

追加言語のサポート

ColdFusion では、英語以外に 17 の言語での検索とインデックス作成がサポートされます。ご使用の言語が使用できない場合は、Solr がその言語でのインデックス作成と検索をサポートしているのであれば、言語をリストに追加できます。

サポートされる言語について詳しくは、http://wiki.apache.org/solr/LanguageAnalysis を参照してください。

Solr がサポートしている言語は、次の方法で追加できます。

1 schema.xml で filter クラスを追加します。

- 次のようにフィールドタイプを追加します。

```
<fieldtype name="text_th" class="solr.TextField">
    <analyzer class="org.apache.lucene.analysis.th.ThaiAnalyzer"/>
</fieldtype>
....
<fieldtype name="text_hi" class="solr.TextField">
    <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.IndicNormalizationFilterFactory"/>
        <filter class="solr.HindiNormalizationFilterFactory"/>
        <filter class="solr.HindiStemFilterFactory"/>
    </analyzer>
</fieldtype>
```

- 次のようにフィールド名を追加します。

```
<field name="contents_pt" type="text_pt" indexed="true" stored="false"
       required="false"/>
<field name="contents_hi" type="text_hi" indexed="true" stored="false"
       required="false"/>
```

2 ColdFusion Administrator で、データとサービス／ Solr サーバーに移動します。

3 「インデックス作成言語を設定」セクションで、次の項目を指定します。

- **新規言語：**言語を指定します（Hindi など）。
- **新規言語接尾辞：**言語の接尾辞を指定します（例えば、Hindi の場合は hi）。

ColdFusion 10 でのセキュリティの機能強化

Solr のセキュリティ保護

Solr はドキュメントレベルまたは通信レベルでは実行できません。ただし、Solr が実行しているアプリケーションサーバーをセキュリティで保護することにより、Solr 検索にセキュリティを追加できます。これを行うには、次のようにします。

1 Solr が実行しているアプリケーションサーバーをセキュリティで保護します。デフォルトは jetty です。

2 ColdFusion Administrator で、データとサービス／ Solr サーバーに移動します。

3 「Solr サーバーの設定」で、「詳細設定の表示」をクリックします。

4 「HTTPS 接続を使用」を選択し、「Solr Admin HTTPS ポート」を指定します。

注意： DIH を使用するときに使用することをお勧めします。

認証のサポート

ColdFusion 9 では、すべてのユーザーがインデックス作成対象のドキュメントにアクセスし、追加、更新、削除できます。このリリースでは、jetty で基本的な認証を提供し、コレクションへのアクセスをセキュリティで保護します。

1 jetty サーバーの web.xml を次のように変更します。

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            Solr authenticated application
        </web-resource-name>
        <url-pattern>
            /core1/
        </url-pattern>
        {*}
    </web-resource-collection>
    <auth-constraint>
        <role-name>
            core1-role
        </role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>
        BASIC
    </auth-method>
    <realm-name>
        Test Realm
    </realm-name>
</login-config>
```

2 jetty.xml で次のセクションのコメントを解除します。

```
<set name="UserRealms">
    <array type="org.mortbay.jetty.security.UserRealm">
        <item>
            <new>
                <set name="name">
                    Test Realm
                </set>
                <set name="config">
                    <systemproperty name="jetty.home" default="."/>
                    /etc/realm.properties
                </set>
            </new>
        </item>
    </array>
</set>
```

3 ユーザー名とパスワードを、次のように /etc/example/realm.properties ファイルに追加します。

username:password, core1-role

4 ColdFusion Administrator で、データとサービス／ Solr サーバーに移動し、「Solr サーバーの設定」セクションの「詳細設定の表示」をクリックします。

5 ユーザー名とパスワードを指定し、「送信」をクリックします。

注意：資格情報を指定しないと、認証なしでインデックス作成操作が行われます。

第 8 章：ColdFusion ORM

リレーショナルデータベースは、ほとんどの企業アプリケーションにおいて中心的な役割を担っています。ただし、リレーショナルデータベースをオブジェクトにマッピングするのは難しい課題となります。オブジェクトリレーショナルマッピング (ORM) とは、アプリケーションオブジェクトモデルとリレーショナルデータベースの間のマッピングを定義するためのプログラミングフレームワークです。

オブジェクトモデルの場合、アプリケーションオブジェクトはデータベースの構造を認識していません。オブジェクトには、プロパティと、他のオブジェクトへの参照が含まれます。データベースは、他のテーブルとの関連付けが可能な列を含むテーブルで構成されます。ORM は、リレーショナルデータベースとオブジェクトモデルの間を仲介します。

ORM を使用すると、アプリケーションのオブジェクトモデルのみを利用してデータへのアクセスと更新を行えます。ORM には次のような特徴があります。

- データベースベンダに依存しない
- キャッシュ
- 並行処理
- パフォーマンスの最適化

ColdFusion ORM の概要

ColdFusion のこれまでのリリースでは、次のような方法でデータベースアクセスが実現されていました。

- SQL ステートメントを処理する cfquery、cfinsert、cfupdate などのタグを使用してリレーショナルデータを管理する。
- ColdFusion コンポーネント (CFC) を使用してオブジェクトを管理し、アプリケーション自体を使用してオブジェクトのライフサイクルを管理する。
- 基本的な CRUD (Create、Retrieve、Update、Delete) 操作も含め、CFC ごとに SQL クエリーを記述する。

これらのタスクの管理は、アプリケーションの規模が大きくなるほど複雑になります。

ColdFusion ORM を使用すると、これらのタスクの大部分が自動化され、次のような利点があります。

- アプリケーションコードが簡潔になり、管理が容易になる
- 生産性が向上し、データベースアプリケーションを短期間で開発できる
- ORM の最適化機能が組み込まれているので、高速なアプリケーションを作成できる
- 記述するコードの量を最小限に抑えることができる

ColdFusion ORM は、オブジェクトモデルとリレーショナルデータベースの間のマッピングフレームワークだけでなく、データクエリー機能や検索機能も提供します。

詳細については、www.hibernate.org を参照してください。

ColdFusion ORM の例

ColdFusion ORM では、オブジェクトを介して永続性が管理されます。ORM の世界では、オブジェクトをエンティティと呼ぶこともあります。ColdFusion では、CFC とそのプロパティを介して永続性が管理されます。ColdFusion アプリケーションの永続 CFC はそれぞれ、データベースのテーブルにマッピングされます。永続 CFC のプロパティはそれぞれ、テーブルの列にマッピングされます。

次の例では、ColdFusion ORM に関連するこれらの概念を説明するために、簡単なアプリケーションを作成します。この例では、ColdFusion 9 インストーラのマニュアルオプションの一部として付属する cfartgallery データソースを使用します。cfartgallery データソースには、Artists テーブルと Art テーブルが含まれています。Artists は、Art テーブルとの間に 1 対多の関係を持っていています。

手順 1:

"Application.cfc" ファイルで ORM 設定を指定します。

最低限必要な設定は、次のサンプルコードに示すとおりです。

Application.cfc

```
<cfset this.name = "ArtGalleryApp">
<cfset this.ormenabled = "true">
<cfset this.datasource = "cfartgallery">
```

これらとは別に、ORM の設定ではその他の項目も指定できます。詳細については、494 ページの「ORM の設定」を参照してください。

重要： これらの設定は "Application.cfc" でのみ定義し、"Application.cfm" では定義しないでください。

手順 2:

"ARTISTS.cfc" をデータベースのテーブルにマッピングします。

1. "ARTISTS.cfc" を作成します。
2. この CFC を永続化して、ARTISTS テーブルにマッピングします。

   ARTISTS.cfc を永続化するには、cfcomponent タグで persistent 属性を true に設定します。table 属性ではテーブル名を指定する必要があります。table 属性が指定されていない場合は、CFC 名がテーブル名として使用されます。

   各 CFC には、エンティティ名を割り当てることができます。エンティティ名は、ORM 関連の関数で永続 CFC を操作するときに使用される名前です。エンティティ名は cfcomponent の entityname 属性を使用して指定できます。entityname を指定が指定されていない場合は、CFC 名が entityname として使用されます。
3. 次に、"ARTISTS.cfc" でプロパティを作成し、それらのプロパティをテーブルの列にマッピングします。テーブルの列ごとに 1 つのプロパティを作成する必要があります。プロパティを列にマッピングするには、column 属性の値を対応する列名に設定します。column 属性が指定されていない場合は、プロパティの名前が列名として使用されます。

ORM 固有の属性を設定する方法については、498 ページの「ORM マッピングの定義」を参照してください。

"ARTISTS.cfc" の定義は次のようになります。

```
<cfcomponent persistent="true">
    <cfproperty name="id" column = "ARTISTID" generator="increment">
    <cfproperty name="FIRSTNAME">
    <cfproperty name="LASTNAME">
    <cfproperty name="ADDRESS">
    <cfproperty name="CITY">
    <cfproperty name="STATE">
    <cfproperty name="POSTALCODE">
    <cfproperty name="EMAIL">
    <cfproperty name="PHONE">
    <cfproperty name="FAX">
    <cfproperty name="thepassword">
</cfcomponent>
```

手順 3:

CRUD 操作を実行します。

ARTISTS テーブルからデータを取得するには、EntityLoad() を使用します。

```
    ARTISTS = EntityLoad("ARTISTS")
```

ARTISTS テーブルに含まれるすべてのレコードがオブジェクト配列として取得されます。

新しいアーティストを追加するには、新しいアーティストオブジェクトを作成し、このオブジェクトに対して EntitySave() を呼び出します。

```
<cfscript>
try {
    newArtistObj = EntityNew("artists");
    newArtistObj.setfirstname("John");
    newArtistObj.setlastname("Smith");
    newArtistObj.setaddress("5 Newport lane");
    newArtistObj.setcity("San Francisco");
    newArtistObj.setstate("CA");
    newArtistObj.setPostalCode("90012");
    newArtistObj.setphone("612-832-2343");
    newArtistObj.setfax("612-832-2344");
    newArtistObj.setemail("jsmith@company.com");
    newArtistObj.setThePassword("jsmith");
    EntitySave(newartistobj);
    ormflush();
} catch(Exception ex) {
    WriteOutput("<p>#ex.message#</p>");
}
</cfscript>
```

既存のレコードを更新するには、該当するオブジェクトを読み込んで変更を加えます。このオブジェクトの行を更新する必要があるかどうかは ColdFusion によって自動的に検出され、ORMFlush() が呼び出されたときに更新されます。

注意： デフォルトでは、リクエストの最後に ORMFlush() が呼び出されます。

次のコードでは、newArtistObj は既に ORM によって管理されているので、再度読み込む必要はありません。

```
newArtistObj.setphone("612-832-1111");
ormflush();
```

レコードを削除するには、EntityDelete() を使用します。

```
EntityDelete(newArtistObj);
ormflush();
```

手順 4:

関係の定義

最初に、ART テーブルのマッピングを定義して、作品とアーティストの間の関係を定義します。

"ART.cfc" の定義は次のようになります。

```
<cfcomponent persistent="true">
    <cfproperty name="artid" generator="increment">
    <cfproperty name="artname">
    <cfproperty name="price">
    <cfproperty name="largeimage">
    <cfproperty name="mediaid">
    <cfproperty name="issold">
</cfcomponent>
```

cfartgallery では、ARTISTS テーブルは ART テーブルとの間に 1 対多の関係を持ち、外部キー列 ARTISTID を使用して結合されています。これは、各アーティストに複数の作品があり、複数の作品が 1 人のアーティストによって作成されたことを意味します。この関係をオブジェクトモデルで表すと、各 ARTISTS オブジェクトに ART オブジェクトの配列が含まれることになります。各 ART オブジェクトには、その ARTISTS オブジェクトへの参照が含まれます。これは双方向の関係の例です。

これを実現するには、各 ARTIST の ART オブジェクトの配列を格納する新しいプロパティを ARTISTS.cfc に追加する必要があります。

```
<cfproperty name="art" type="array" fieldtype="one-to-many" cfc="Art" fkcolumn="ARTISTID">
```

fieldtype="one-to-many" では、関係のタイプを指定しています。

CFC="Art" は、「ART」cfc との間に関係があることを示しています。

fkcolumn="artistid" では、外部キーを指定しています。

各作品は 1 人のアーティストによって作成され、各アーティストには複数の作品があるため、ART は ARTISTS テーブルに対して多対 1 の関係にあります。この関係を定義するには、"ARTISTS.cfc" との関係を定義するプロパティを "ART.cfc" に追加します。

```
<cfproperty name="artists" fieldtype="many-to-one" fkcolumn="artistid" cfc="Artists" lazy="true">
```

fieldtype="many-to-one" では、関係のタイプを指定しています。

CFC="ARTISTS" は、「ARTISTS」cfc との間に関係があるあることを示しています。

fkcolumn="ARTISTID" では、外部キーを指定しています。

手順 5 :

関係を使用してレコードを取得します。

```
<cfscript>
    artist = EntityLoad("Artists", 1, true);
    arts = artist.getArts();
    WriteOutput("<b>" & artist.getid() & " " & artist.getfirstname() & " " &
    artist.getlastname() & "</b> has " & ArrayLen(arts) & " arts:<br>");
        if (ArrayLen(arts) > 0)
        {
            for(j = 1; j <= ArrayLen(arts); j ++)
            {
                art = arts[j];
                WriteOutput(art.getartname() & "<br>");
            }
        }
</cfscript>
```

アーキテクチャ

ColdFusion ORM で永続オブジェクトを作成するには、オブジェクトマッピングを定義する必要があります。オブジェクトマッピングには、次のような詳細情報が含まれます。

- オブジェクトのクラスに対応するテーブル名
- オブジェクトの各フィールドに対応する列名
- 関連するオブジェクトの結合条件

ColdFusion では、このマッピングを CFC で指定できます。そのような CFC は永続 CFC と呼ばれます。通常、個々の永続 CFC は、データベース内のテーブルにマッピングされます。CFC の各プロパティは一般にテーブルの列にマッピングされます。また、それ以外のプロパティを使用して、関係やその他のマッピング詳細情報を定義する場合もあります。

ColdFusion でアプリケーションの Hibernate 設定を作成する際には、これらの永続 CFC を使用して Hibernate マッピングファイルが自動的に生成されます。マッピングファイルの拡張子は「.hbmxml」になります。たとえば、"ARTISTS.cfc" という永続 CFC を使用する場合は、自動的に "Artists.hbmxml" が生成されます。Hibernate マッピングファイルには、ColdFusion ORM と連携するために、Hibernate で定義された XML 形式のマッピング情報が含まれます。これらの Hibernate マッピングファイルは、手動で作成できます。

Hibernate マッピングファイルの手動作成の詳細については、521 ページの「高度なマッピング」を参照してください。

ColdFusion アプリケーションで ColdFusion ORM を使用するには、Application.cfc の THIS スコープで ormenabled を true に設定する必要があります。永続 CFC を定義するには、cfcomponent タグで persistent="true" を設定します。cfcomponent および cfproperty には、マッピング情報を指定するための属性の配列があります。

詳細については、498 ページの「ORM マッピングの定義」を参照してください。

アプリケーションが起動され、そのアプリケーションで Hibernate 設定ファイルが指定されている場合、ColdFusion は最初にその設定ファイルをロードします。Hibernate 設定ファイルには、方言、キャッシュ設定、マッピングファイルなど、アプリケーションに必要な各種の設定パラメータが含まれています。設定ファイルが指定されていない場合、ColdFusion ORM はデフォルト設定を使用して Hibernate 設定を作成します。

Hibernate 設定がロードされると、アプリケーションフォルダとそのマッピング先フォルダにあるマッピングファイル (*.hbmxml) がすべてロードされ、設定に追加されます。

その後、ColdFusion は、アプリケーションフォルダとそのマッピング先フォルダで永続 CFC を検索します。どの永続 CFC に対しても Hibernate マッピングファイルが存在しない場合は、Hibernate マッピングファイルが自動的に生成されます。プライマリキー、外部キー、列データ型などのマッピング情報が永続 CFC に含まれていない場合は、ColdFusion が自動的にデータベースを調べてマッピングを判別します。

次に、ColdFusion は DDL を生成する必要があるかどうかを調べます。これは、ORM 設定の dbcreate オプションを使用して設定できます。dbcreate で指定されている設定オプションに基づいてテーブルが作成または更新されます。その後、Hibernate SessionFactory が作成され、アプリケーションが実行されている間は、そのファクトリをアプリケーションで使用できます。SessionFactory は、永続オブジェクトのライフサイクルを管理する Hibernate セッションを作成するために使用されます。

ColdFusion では、最初に CRUD メソッドが呼び出されると Hibernate セッションが開始され、リクエストが終了するか、ORMCloseSession() メソッドが呼び出されるとセッションが終了します。

パフォーマンスを高めるため、Hibernate では、セッション中に行われるすべての作成操作、更新操作、削除操作が 1 つのバッチにまとめられ、セッションがフラッシュされたとき、または必要があるときにのみ、そのバッチが実行されます。セッションのフラッシュは、リクエストが終了したとき、または ORMFlush() メソッドが呼び出されたときに発生します。

トランザクションの場合には、トランザクションが開始されるたびに新しいセッションが作成され、トランザクションが終了するとセッションも終了します。それまでに開かれていたセッションは、トランザクションの開始時にフラッシュされて閉じられます。

Hibernate 設定の作成およびロードは、アプリケーションが起動されない限り実行されません。したがって、永続 CFC や Hibernate マッピングファイルでマッピング設定が変更されても、それらの変更は自動的にはロードされません。このような変更内容をロードするには、アプリケーションを再起動するか、ORMReload() を呼び出します。

アプリケーションを再起動するには、ApplicationStop() を使用してアプリケーションを停止します。その後、このアプリケーションの任意のページにリクエストを送信すると、アプリケーションが自動的に起動します。

ORM の設定

ORM の設定は "application.cfc" で行うため、この設定はアプリケーションごとに固有になります。ColdFusion アプリケーションで ORM を使用するには、次の設定を行う必要があります。

1. アプリケーションに対して ORM を有効にします。これを行うには、application.cfc の THIS スコープで ormenabled プロパティを true に設定します。
2. データソース名を指定します。これを行うには、アプリケーションの THIS スコープでデータソースプロパティを true に設定するか、アプリケーションの ORM 設定でデータソースプロパティを定義します。

   データソースはサーバーで設定する必要があることに注意してください。

ORM 設定を指定するには、ormsettings という構造体を使用します。この構造体は、Application.cfc の THIS スコープで定義されます。次の表で、Application.cfc で定義可能な ORM の設定について説明します。

プロパティ名	説明
ormenabled	ColdFusion アプリケーションで ORM を使用するかどうかを指定します。ORM を使用する場合は値を true に設定します。デフォルトは false です。
datasource	ORM で使用するデータソースを定義します。
ormsettings	すべての ORM 設定を定義する構造体です。詳細については、494 ページの「ORM の設定」を参照してください。

ORM の設定

ColdFusion で ORM を設定するために使用される ormsettings 構造体では、次の設定を指定できます。これらの設定はすべてオプションです。ORM の設定は、値を true または yes に指定すると有効になり、それ以外の場合は無効になります。

プロパティ名	デフォルト	説明
autogenmap	true	永続 CFC のマッピングを自動的に生成するかどうかを指定します。 autogenmap=false の場合は、.HBMXML ファイルを使用してマッピングを定義する必要があります。
automanageSession ColdFusion 9.0.1 で追加されました	true	Hibernate セッションの管理が ColdFusion によって自動的に行われるようにするかどうかを指定できます。 • 有効にした場合 : セッションは、ColdFusion によって完全に管理されます。つまり、セッションが消去されるタイミング、セッションがクリアされるタイミング、セッションが閉じられるタイミングは、ColdFusion によって決められます。 • 無効にした場合 : セッションの消去、クリア、終了の管理をアプリケーションで担当します。トランザクションにおける唯一の例外として、トランザクションがコミットされると、セッションはアプリケーションによって消去されます。 このフラグが有効と無効のいずれであるかにかかわらず、リクエストの終了時には、ORM セッションが閉じられます。
cacheconfig		セカンダリキャッシュプロバイダで使用する設定ファイルの場所を指定します。この設定は、secondarycacheenabled=true を指定した場合にのみ適用されます。 詳細については、544 ページの「セカンダリレベルのキャッシュ」を参照してください。
cacheprovider	ehcache	ORM でセカンダリキャッシュとして使用するキャッシュプロバイダを指定します。指定可能な値は次のとおりです。 • Ehcache • JBossCache • Hashtable • SwarmCache • OSCache その他のキャッシュプロバイダを示す完全修飾クラス名。 この設定は、secondarycacheenabled=true が指定されている場合にのみ使用できます。 詳細については、544 ページの「セカンダリレベルのキャッシュ」を参照してください。

プロパティ名	デフォルト	説明
catalog		ORM で使用するデフォルトのカタログを指定します。
cfclocation		マッピングを生成するために ColdFusion で使用する永続 CFC を検索するディレクトリ (またはディレクトリの配列) を指定します。cfclocation を設定した場合は、そこで指定したパスのみが検索対象になります。これが設定されていない場合は、アプリケーションディレクトリ、そのサブディレクトリ、およびそのマッピング先ディレクトリから永続 CFC が検索されます。
datasource		ORM で使用するデータソースを指定します。ここでデータソースを指定しない場合は、アプリケーションに対して指定されているデータソースが使用されます。データソース名を指定するには、this.datasource="<datasource_name>" という規則を使用します。
dbcreate	none	ColdFusion ORM では、アプリケーションに対して ORM が初期化されるときに、そのアプリケーション用のテーブルをデータベース内に自動的に作成できます。この動作は、ormsettings の dbcreate を使用して有効化できます。dbCreate で指定できる値は次のとおりです。 • update : この値を設定した場合、テーブルが存在しないときは新しいテーブルが作成され、テーブルが存在するときはそのテーブルが更新されます。 • dropcreate : この値を設定した場合、テーブルが存在するときはそのテーブルを削除してから新しいテーブルが作成されます。 • none (デフォルト) : この値を設定した場合、データベーススキーマは何も変更されません。

プロパティ名	デフォルト	説明
dialect		方言を指定します。 ColdFusion では次の方言がサポートされています。 • DB2 • DB2AS400 • DB2OS390 • Derby • PostgreSQL • MySQL • MySQLwithInnoDB • MySQLwithMyISAM • Oracle8i • Oracle9i • Oracle10g • Sybase • SybaseAnywhere • MicrosoftSQLServer • Informix これらの方言だけでなく、完全修飾クラス名を使用することにより、独自の方言を指定することもできます。 **注意：**Microsoft Access を使用する場合、方言を自動的に検出することはできません。その場合は、方言として Microsoft SQL Server を使用してください。
eventHandling	false	ORM イベントのコールバックを行うかどうかを指定します。詳細については、CFC でのイベント処理を参照してください。
flushatrequestend	true	リクエストの終了時に ormflush を自動的に呼び出すかどうかを指定します。flushatrequestend が false の場合は、リクエストが終了しても ormflush は自動的に呼び出されません。 ORM セッション管理を参照してください。
logSQL	false	ORM で実行した SQL クエリーを記録するかどうかを指定します。LogSQL=true の場合は、SQL クエリーの実行履歴がコンソールに表示されます。
namingstrategy		データベース標準およびネーミング規則を定義します。ネーミングストラテジを参照してください。

プロパティ名	デフォルト	説明
ormconfig		Hibernate 設定ファイル。 このファイルには、方言、キャッシュ設定、マッピングファイルなど、アプリケーションに必要な各種の設定パラメータが含まれています。 Hibernate 設定 XML ファイルで定義されている設定よりも、ormsettings で定義されている設定のほうが優先されます。Hibernate 設定 XML ファイルには接続情報が含まれますが、ColdFusion では独自の接続プールが使用されるため、この接続情報は無視されます。 このファイルは、ormsetting で定義できない Hibernate 設定を使用する必要がある場合にのみ、使用する必要があります。
savemapping	false	生成された Hibernate マッピングファイルをディスクに保存するかどうかを指定します。この値を true に設定すると、Hibernate マッピングの XML ファイルは CFC と同じディレクトリに「<CFC 名 >.hbmxml」という名前で保存されます。 savemapping の値が CFC で指定されている場合は、ormsetting で指定されている値よりも優先されます。
schema		ORM で使用するデフォルトのスキーマを指定します。
secondarycacheenabled	false	セカンダリキャッシュを有効にするかどうかを指定します。詳細については、セカンダリキャッシュの使用を参照してください。
skipCFCWithError ColdFusion 9.0.1 で追加されました	false	エラーのある CFC をスキップするかどうかを指定できます。true に設定すると、エラーのある CFC は無視されます。
sqlscript		ORM が初期化された後に実行する SQL スクリプトファイルのパスを指定します。これは dbcreate が dropcreate に設定されている場合に適用されます。ファイルの絶対パスまたはアプリケーションからの相対パスを指定する必要があります。この SQL スクリプトファイルを使用して、アプリケーションへのアクセス前に、テーブルの値を設定できます。
useDBForMapping	true	Hibernate マッピングの生成に必要な情報が不足している場合に、その情報を確認するためにデータベースを検索するかどうかを指定します。列データ型、プライマリキー、および外部キーに関する情報がデータベースから検索されます。

Application.cfc の例

```
<cfset this.name = "ArtGallery">
    <cfset this.ormenabled = "true">
    <cfset this.ormsettings={datasource="cfartgallery", logsql="true"}>
```

ロギング

ORM によって生成されて実行される SQL クエリーを監視することは、トラブルシューティングやパフォーマンスの最適化において重要な作業となります。

クエリーを監視して記録するには、次の方法を使用します。

- ormsettings で logsql を定義する : SQL のロギングをすばやく有効化できます。application.cfc で次のフラグを有効にする必要があります。

  <cfset this.ormsettings.logsql = "true">

  これにより、Hibernate で生成されるすべての SQL クエリーが、コンソールおよびサーバーの出力ログファイルに記録されます。

- log4J.properties を使用する : Hibernate では log4j を使用してロギングが行われるので、log4j.properties を変更することで、SQL を含むロギングを完全に制御できます。このファイルは、<ColdFusion のホームディレクトリ >/lib ディレクトリにあります。

次に log4j.properties ファイルの例を示します。

```
###--------------- Hibernate Log Settings ------
### Set Hibernate log
log4j.logger.org.hibernate=ERROR, HIBERNATECONSOLE

### log just the SQL
#log4j.logger.org.hibernate.SQL=DEBUG, HIBERNATECONSOLE
#log4j.additivity.org.hibernate.SQL=false
### Also log the parameter binding to the prepared statements.
#log4j.logger.org.hibernate.type=DEBUG
### log schema export/update ###
log4j.logger.org.hibernate.tool.hbm2ddl=DEBUG, HIBERNATECONSOLE
### log cache activity ###
log4j.logger.org.hibernate.cache=ERROR, HIBERNATECONSOLE
# HibernateConsole is set to be a ColsoleAppender for Hibernate message  using a PatternLayout.
log4j.appender.HIBERNATECONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.HIBERNATECONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.HIBERNATECONSOLE.layout.ConversionPattern=%d{MM/dd HH:mm:ss} [%t] HIBERNATE %-5p - %m%n%n
#-----------------------------------------------
```

これらの設定は、エンティティ操作に関連して生成される SQL、実行中のステートメントにデータをバインドする方法、DDL に関連して生成される SQL、およびセカンダリキャッシュに対して実行される操作を制御します。すべてのログは、HIBERNATECONSOLE を使用してコンソールに記録されます。HIBERNATECONSOLE は、実際にはコンソールアダプタです。これは簡単に FileAppender に変更でき、変更するとログファイルに記録されるようになります。この設定は、次のものに関するロギングを制御します。

- エンティティ操作に関連して生成される SQL
- 準備済みのステートメントに対するパラメータのバインディング
- DDL に関連して生成される SQL
- セカンダリキャッシュの操作

デフォルトの設定では、すべてのログがコンソールに記録されます。log4j で提供されている FileAppender を使用して、ログファイルに出力することもできます。

Appender の詳細については、log4j を参照してください。

Log4j のプロパティ	説明
log4j.logger.org.hibernate.SQL	SQL のロギングを行う条件と方法を制御します。DEBUG を指定すると、すべての SQL がロギングされます。
log4j.logger.org.hibernate.type	これを指定すると、準備済みステートメントへのパラメータバインディングがロギングされます。
log4j.logger.org.hibernate.tool.hbm2ddl	DDL に関連する SQL (スキーマのエクスポート) をロギングします。
log4j.logger.org.hibernate.cache	セカンダリキャッシュの情報をロギングします。

ORM マッピングの定義

ORM マッピングは、CFC または別個の Hibernate マッピングファイル (.hbmxml) で定義できます。Hibernate マッピングファイルの詳細については、521 ページの「高度なマッピング」を参照してください。ORM マッピングを CFC で定義するには、cfcomponent および cfproperty タグで ORM 固有の属性を使用します。

マッピング情報を含む CFC (ARTIST.cfc) の例を次に示します。

```
<cfcomponent persistent="true" entityname="Artist" table="Artists">
    <cfproperty name="id" column="ARTISTID" generator="increment">
    <cfproperty name="firstname">
    <cfproperty name="lastname">
    <cfproperty name="address">
    <cfproperty name="city">
    <cfproperty name="state">
    <cfproperty name="postalcode">
    <cfproperty name="email">
    <cfproperty name="phone">
    <cfproperty name="fax">
    <cfproperty name="thepassword">
</cfcomponent>
```

ColdFusion コンポーネントのマッピング

cfcomponent タグで永続 CFC を定義するには、persistent="true" を設定します。このタグでは、永続 CFC に関するさまざまなマッピングを指定できます。

永続 CFC に init メソッドがある場合は、引数が存在しないこと、またはすべての引数がオプションであることを確認してください。

シンタックス

```
<cfcomponent
    accessors= "yes|no"
    persistent="true|[false]"
    entityname="entity_name"
    table="table_name"
    schema="schema"
    catalog="catalog"
    dynamicInsert="true|[false]"
    dynamicupdate="true|[false]"
    readonly="true|[false]"
    selectbeforeupdate="true|[false]"
    discriminatorvalue="discriminator_value"
    discriminatorcolumn="discriminator_column"
    joincolumn="join_column"
    cacheuse="read-only|read-write|nonstrict-read-write|transactional|[none]"
    cachename="cache_name"
    batchsize="batchsize"
    optimisticLock="none|[version]|dirty|all"
    lazy="[true]|false"
    savemapping="true|false">
```

属性

属性	必須 / オプション	デフォルト	説明
accessors	true		false に設定すると、暗黙的な getter と setter が生成されなくなります。
batchsize	オプション	1	一度に取得するレコード数を示す整数値。 詳細については、遅延ロードを参照してください。
cachename	オプション		この値は、セカンダリキャッシュの名前を指定する場合に使用します。詳細については、キャッシュを参照してください。

属性	必須 / オプション	デフォルト	説明
cacheuse	オプション	none	この値は、セカンダリキャッシュにこのコンポーネントのデータを格納するときに使用するキャッシュ処理方式を指定する場合に使用します。詳細については、キャッシュを参照してください。
catalog	オプション		データベースカタログ名を指定する場合に使用します。
discriminatorcolumn	オプション		この属性は、継承マッピングに必要な識別子列を定義する場合に使用します。詳細については、継承マッピングを参照してください。
discriminatorvalue	オプション		この属性は、継承マッピングに必要な識別子の値を定義する場合に使用します。詳細については、継承マッピングを参照してください。
dynamicInsert	オプション	false	実行時に INSERT SQL を生成するかどうか。 • true • false この SQL には、null 以外の値を持つ列のみが含まれます。
dynamicupdate	オプション	false	実行時に UPDATE SQL を生成するかどうか。 • true • false この SQL には、変更された値を持つ列のみが含まれます。
entityname	オプション	CFC の名前	エンティティの名前を指定します。エンティティ名は、ORM 関連の関数で永続 CFC を操作するときに使用される名前です。entityname を指定が指定されていない場合は、CFC 名が entityname として使用されます。entityname はアプリケーション内で一意になっている必要があります。
joincolumn	オプション		この属性は、継承マッピングに必要な結合列を定義する場合に使用します。詳細については、継承マッピングを参照してください。
lazy	オプション	true	遅延ロードを使用するかどうか。 • true • false 詳細については、遅延ロードを参照してください。
optimisticLock	オプション	version	ロック方式を指定します。 次の 4 つの値の中から、いずれかを指定できます。 • none • version • dirty • all 詳細については、オプティミスティックロックを参照してください。

属性	必須 / オプション	デフォルト	説明
persistent	必須	false	この CFC が永続 CFC であるかどうか。 • true • false
readonly	オプション	false	テーブルが読み取り専用かどうか。 • true • false これは EntitySave() メソッドによって挿入されます。
savemapping	オプション		生成された Hibernate マッピングファイルをディスクに保存します。アプリケーションレベルで savemapping が設定されている場合は、ここで設定した値が優先されます。
schema	オプション		スキーマ名を指定する場合に使用します。
selectbeforeupdate	オプション	false	オブジェクトが実際に変更されていることが確実でない限り SQL UPDATE を実行しないようにするかどうか。 • true • false update() を使用する新しいセッションに一時的なオブジェクトが関連づけられている場合、Hibernate では、UPDATE が実際に必要かどうかを判別するために別個の SQL SELECT が実行されます。
table	オプション	CFC の名前	この CFC がマッピングされるデータベーステーブルの名前を指定します。

プロパティのマッピング

次のものを定義するには cfproperty タグを使用します。

- プライマリキーまたは複合キー
- 列
- 関係
- バージョン管理

次の表に、これらのマッピングの定義に使用する一般的な属性を示します。

属性	必須 / オプション	デフォルト	説明
default	オプション		オブジェクトが作成されたときのプロパティのデフォルト値を設定します。
fieldtype	オプション	column	プロパティのタイプを指定します。この属性は、次のものを指定する場合に使用します。 • プライマリキー : プロパティをテーブルのプライマリキーにマッピングするには、fieldtype = "id" を指定します。詳細については、プライマリキーを参照してください。 • 列 : プロパティをテーブルの列にマッピングするには、fieldtype = "column" を指定します。詳細については、列を参照してください。 • 関係 : fieldtype = "< 関係のタイプ >" を指定します。< 関係のタイプ > では、one-to-one、one-to-many、many-to-one、many-to-many のいずれかを指定できます。詳細については、関係の定義を参照してください。 • バージョン : バージョン管理されたデータが列に含まれることを示すには、fieldtype = "version" を指定します。詳細については、バージョンを参照してください。 • タイムスタンプ : タイムスタンプ付きのデータが列に含まれることを示すには、fieldtype="timestamp" を指定します。詳細については、タイムスタンプを参照してください。 • コレクション : コレクションのマッピングを定義するには、fieldtype="collection" を指定します。詳細については、コレクションマッピングを参照してください。
name	必須		プロパティの名前を指定します。
type	オプション		プロパティの ColdFusion データ型を指定します。

persistent	オプション	true	このプロパティを永続化するかどうかを指定します。persistent="false" の場合は、すべての ORM 関連属性が無視されます。
			このプロパティを ORM で永続化する必要がない場合は、この属性を false に設定します。次に例を示します。
			`<cfcomponent persistent=true>`
			`  <cfproperty name="a">`
			`  <cfproperty name="b" persistent="true">`
			`  <cfproperty name="c" persistent="false">`
			`</cfcomponent>`
			この CFC のオブジェクトが永続化されると、プロパティ a および b はデータベースで永続化されますが、プロパティ c は永続化されません。

remotingFetch	オプション	true	remotingFetch が false に設定されている場合、その属性の値は Flash Remoting を介して送信されません。この属性はすべてのプロパティに対してデフォルトで true に設定されます。ただし、persistent=true に設定されている ORM CFC の場合は、関係が 1 対 1、1 対多、多対 1、多対多のいずれであっても、remotingFetch 属性の値はデフォルトで false に設定されます。

プライマリキー

単純なプライマリキー

リレーショナルデータベースでは、テーブル内の行を一意に識別するキーとしてプライマリキーを定義します。通常、テーブルには、1 つの情報列に対応するプライマリキーが 1 つあります。

cfproperty をテーブル内のプライマリキーにマッピングするには、fieldtype="id" 属性を設定します。

シンタックス

```
<cfproperty
    name="property_name"
    fieldType="id"
    ormtype="type"
    column="column_name"
    generator="increment|identity
    |sequence|sequence-identity|seqhilo
    |uuid|guid|native|[assigned]|select|foreign"
    params="{key1=val1,key2=val2...}"
    sqltype="sql_type"
    length="column_length"
    unsavedvalue="instantiated_instance">
```

例

assigned 方式で生成されるプライマリキーを定義する例を次に示します。

```
<cfproperty name="artistid" fieldtype="id" column="ARTISTID" generator="assigned">
```

increment 方式のジェネレータを使用して自動生成されるプライマリキーを定義する例を次に示します。

```
<cfproperty name="artid" fieldtype="id" column="ARTID" generator="increment">
```

追加のパラメータを必要とするジェネレータを使用して自動生成されるプライマリキーを定義する例を次に示します。

```
<cfproperty name="id" fieldtype="id" column="ID" generator="sequence" params="{sequence='id_sequence'}>
```

属性

属性	必須 / オプション	デフォルト	説明
column	オプション	name 属性の値	プライマリキーの列名を指定するために使用します。
fieldType	オプション		プライマリキーの場合は「id」に設定する必要があります。fieldtype を指定せずに useDBForMapping=true を指定した場合は、データベースを調べて fieldtype が決定されます。
generator	オプション	assigned	永続 CFC のインスタンスに対して一意の識別子を生成するときに使用するアルゴリズムです。詳細については、ジェネレータ を参照してください。
length	オプション		この属性では、列の長さを指定します。この属性はテーブルを作成する場合にのみ使用します。
name	必須		プロパティの名前
ormtype	オプション	String	プライマリキーのデータ型を指定します。データ型を設定せずに、ORM 設定で useDBForMapping=true を指定した場合は、データベースを調べて ormtype が決定されます。 ColdFusion でサポートされるデータ型については、ORM データ型を参照してください。
params			各種のジェネレータで ID を生成するために必要な追加パラメータです。 params 属性の値は、CF 構造体のシンタックスに準拠している必要があります。例 : params = {key1='value1', key2='value2'} 詳細については、ジェネレータ を参照してください。

属性	必須 / オプション	デフォルト	説明
selectkey	オプション		データベーストリガによって生成されたプライマリキーを取得するために使用する列名を指定します。
sqltype	オプション		この属性では、列で使用する必要がある DB 固有の SQLType を指定します。この属性はテーブルを作成する場合にのみ使用します。 この属性が指定されていない場合は、このプロパティで指定されている ormtype に基づいて SQLType が自動的に決定されます。
unSavedValue	オプション		新しくインスタンス化されたインスタンスがまだデータベースに保存またはロードされていないことを示す識別子プロパティ値です。

複合キー

複数の列で構成されているプライマリキーは、複合キーと呼ばれます。複合キーを指定するには、プライマリキーを構成するすべてのプロパティで fieldtype='id' を使用します。

例

テーブルで列 Order_id と Product_id が複合キーを構成している場合は、次のように定義する必要があります。

```
<cfproperty name="Order_Id" fieldtype="id" column="Order_Id">
<cfproperty name="Product_Id" fieldtype="id" column="Product_Id">
```

複合キーの場合は、ジェネレータを常に「assigned」に設定する必要があります。

ジェネレータ

ジェネレータとは、永続 CFC のインスタンスに対して一意の識別子を生成するために使用されるアルゴリズムです。ジェネレータとして指定できる値は、次のいずれかになります。

- increment : このアルゴリズムでは、ORM によって管理されているカウンタをインクリメントして、long、short、または int 型の識別子が生成されます。テーブルでプライマリキーの自動生成が有効になっていない場合、ORM を使用してプライマリキーを生成するには、一般に、この設定を使用します。テーブルにデータを挿入するプロセスが単一の ColdFusion インスタンスのみである場合は、この設定を使用することをお勧めします。
- identity : ID 列をサポートする DB2、MySQL、Sybase、MS SQL などのデータベースでは、ID キーの生成を使用できます。キーはデータベースによって自動的に生成され、生成されたキーがオブジェクトに追加されます。この方式を使用する場合は、ORM で 2 つの SQL クエリーを実行して新しいオブジェクトを挿入する必要があります。
- sequence : シーケンスをサポートする DB2、Oracle、PostgreSQL、Interbase、McKoi、SAP などのデータベースでは、シーケンススタイルキーの生成を使用できます。キーはデータベースによって自動的に生成され、生成されたキーがオブジェクトに追加されます。この方式を使用する場合は、ORM で 2 つの SQL クエリーを実行して新しいオブジェクトを挿入する必要があります。このジェネレータには、params 属性で指定する必要があるシーケンスパラメータが含まれます。

  次に例を示します。

  ```
  <cfproperty name="id" fieldtype="id" generator="sequence" params="{sequence='id_sequence'}">
  ```

- native : 通常、このアルゴリズムは、プライマリキーを自動生成する場合に使用します。このアルゴリズムでは、使用するデータベースの機能に応じて、ID またはシーケンスが選択されます。
- assigned : このアルゴリズムは、アプリケーションでオブジェクトに独自の ID を割り当てる必要がある場合に使用します。ColdFusion では、このジェネレータがデフォルトで使用されます。
- foreign : これは <one-to-one> の関係でプライマリキーの関連付けを行う場合に使用します。この場合、プライマリキーは関連付けられるオブジェクトのプライマリキーと同一になります。このジェネレータを使用する場合は、params 属性で

プロパティパラメータの設定が必要になる場合があります。param プロパティの値は、関係プロパティの名前に設定する必要があります。

詳細については、1 対 1 関係を参照してください。

- seqhilo: www.hibernate.org/5.html を参照してください。
- uuid: www.hibernate.org/5.html を参照してください。
- guid: www.hibernate.org/5.html を参照してください。
- select: www.hibernate.org/5.html を参照してください。
- sequence-identity: www.hibernate.org/5.html を参照してください。

列

cfproperty をテーブル内の列にマッピングするには、fieldtype="column" を指定します。cfproperty で fieldtype が指定されていない場合は、列プロパティとしてマッピングされます。

シンタックス

```
<cfproperty
    name="Property name"
    fieldtype="column"
    column="column_name"
    persistent="true|false"
    formula="SQL expression"
    ormtype="ormtype"
    update="[true]|false"
    insert="[true]|false"
    optimisticLock="[true]|false"
    generated="true|[false]"
    length="column_length"
    precision="precision"
    scale="scale"
    index="index_name"
    unique = "true|[false]"
    uniquekey="uniquekey_name"
    notnull="true|[false]"
    dbdefault="default_col_value"
    sqltype="sql_type">
```

例

単純なプロパティを指定するには、次のようにします。

```
<cfproperty name="FIRSTNAME"/>
```

列名とは異なる名前のプロパティを指定するには、次のようにします。

```
<cfproperty name="LNAME" column="LASTNAME"/>
```

読み取り専用のプロパティを指定するには、次のようにします。

```
<cfproperty name="password" column="THEPASSWORD" insert="false" update="false">
```

属性

属性	必須 / オプション	デフォルト	説明
column	オプション	プロパティの名前	列の名前です。
dbdefault			スキーマがエクスポートされたときの、テーブルの列のデフォルト値を設定します。
fieldType	オプション	column	列プロパティの場合は「column」に設定する必要があります。
formula	オプション		プロパティの値を定義する SQL 式です。式を指定した場合、列の値は無視されます。 計算プロパティを参照してください。
generated	オプション	never	database {never|always|insert} でこのプロパティの値を実際に生成するかどうかを指定します。 • never : このプロパティの値をデータベースで生成しないことを指定します。 • always : このプロパティの値をデータベースで常に生成することを指定します。 • insert : 挿入時にはこのプロパティの値を生成し、更新時には再生成しないことを指定します。
insert	オプション	true	この列を SQL の UPDATE および INSERT ステートメントに含めるかどうかを指定します。 {true/false} 列を読み取り専用にする必要がある場合は、update=false および insert=false を設定します。
name	必須		プロパティの名前です。この属性が指定されていない場合は、プロパティの名前が列名として使用されます。
optimisticlock	オプション	true	このプロパティを更新するときにテーブル行に対するオプティミスティックロックを取得する必要があるかどうかを指定します。 {true/false}
ormtype	オプション	string	データ型を指定します。 指定した場合は、次のようになります。 • ORM 設定 useDBForMapping が true に設定されていて、テーブルが存在する場合は、テーブルを調べて ormtype が取得されます。 • (それ以外の場合) 型が指定されている場合はその型が ormtype として使用されます。
update	オプション	true	この列を SQL の UPDATE ステートメントに含めるかどうかを指定します。 {true/false} 列を読み取り専用にする必要がある場合は、update=false および insert=false を設定します。

DDL 専用の属性

次の属性は、DDL の生成が必要な場合にのみ使用され、実行時には使用されません。

属性	必須 / オプション	デフォルト	説明
dbdefault	オプション		テーブルの列のデフォルト値を指定します。
index	オプション		マッピング対象の列を使用して作成されるインデックスの名前を指定します。
length	オプション		長さの値を指定します。
notnull	オプション	false	この列に対して notnull 制約を追加する必要があるかどうかを指定するブール値です。
precision	オプション		精度の値を指定します。
scale	オプション		スケールの値を指定します。

sqltype	オプション		この設定は、SQL データ型に対する ormtype のデフォルトマッピングよりも優先されます。sqltype は、テーブルの作成時に、列に適用する DB 固有の SQL 型として使用されます。この属性が指定されていない場合は、このプロパティで指定されている ormtype に基づいて sqltype が自動的に決定されます。
			次に例を示します。
			`<cfProperty name="active" ormtype="char" sqltype="bit">`
			`<cfProperty name="balance" ormtype="float" sqltype="decimal(13,3)">`

unique	オプション		列に対する一意制約が必要かどうかを指定します。
uniquekey	オプション		単一の一意キー制約に列をグループ化します。

計算プロパティ

計算プロパティは、列から値を取得するのではなく、SQL クエリーを使用して計算されるプロパティです。このプロパティの値を取得するための SQL を指定するには、formula 属性を使用します。

```
<cfcomponent persistent="true" table="ARTISTS" schema="APP">
    <cfproperty name="ID" column="ARTISTID" fieldtype="id"/>
    <cfproperty name="FIRSTNAME"/>
    <cfproperty name="LASTNAME"/>
    <cfproperty name="NumberOfArts" formula="select count(*) from Art art where
                        art.ArtistID=ArtistID"/>
</cfcomponent>
```

バージョン管理

バージョン管理は、コンポーネントの同時性制御を実装するための技法です。コンポーネントでは version プロパティまたは timestamp プロパティを指定できます。

詳細については、オプティミスティックロックを参照してください。

注意：1 つのコンポーネントで指定できるバージョン管理プロパティは、timestamp または version のどちらか一方のみです。2 つの timestamp、2 つの version、timestamp と version のように、複数のバージョン管理プロパティを指定すると、エラーが発生します。

バージョン

バージョン管理されたデータが列に含まれることを示すには、version 属性を使用します。version 属性は長期間にわたるトランザクションに便利です。

シンタックス

```
<cfproperty
name="fieldname"
fieldtype="version"
column="column name"
ormtype="type"
generated="true|[false]"
insert="[true]|false">
```

例

単純な version プロパティを作成するには、次のようにします。

```
<cfproperty name="version" fieldtype="version">
```

属性

属性	必須 / オプション	デフォルト	説明
column	オプション		バージョン管理されたデータを含む列の名前です。
fieldtype	必須		プライマリキーの場合は「version」に設定する必要があります。
generated	オプション	never	バージョン管理されたフィールドをデータベースで生成するかどうかを指定します。有効な値は "never" と "always" です。
insert	オプション		バージョン管理されたフィールドを SQL の INSERT ステートメントに含めるかどうかを指定します。
name	必須		プロパティの名前です。
ormtype	オプション	int	データ型として指定できる値は、次のいずれかになります。 integer long short

タイムスタンプ

タイムスタンプ付きのデータが列に含まれることを示すには、timestamp 属性を使用します。timestamp 属性は、version 属性の代わりとして使用します。

シンタックス

```
<cfproperty
name="fieldname"
fieldtype="timestamp"
column="column name"
generated="true|[false]"
source="[vm]|db">
```

属性	必須 / オプション	デフォルト	説明
column	オプション		タイムスタンプ付きデータを含む列の名前です。
fieldtype	必須		フィールドのタイプを指定します。 タイムスタンプ付きフィールドの場合は、フィールドタイプの値を timestamp に設定します。
generated	オプション	false	タイムスタンプフィールドをデータベースで生成するかどうかを指定します。次のいずれかの値を指定できます。 false true
name	必須		プロパティの名前です。
source	オプション	vm	タイムスタンプをどこから取得するかを指定します。次のいずれかの値を指定できます。 db vm

ORM データ型

CFC では、次の ORM データ型を使用できます。

- string
- character
- char
- short
- integer
- int
- long
- big_decimal
- float
- double
- Boolean
- yes_no
- true_false
- text
- date
- timestamp
- binary
- serializable
- blob
- clob

テーブル名および列名での SQL キーワードのエスケープ

ColdFusion では、テーブルまたは列の名前として SQL キーワードが使用されている場合、または名前にスペースが含まれている場合、これらの名前は自動的にエスケープされます。

SQL キーワードのリストは、<ColdFusion のホームディレクトリ >/lib/sqlkeywords.properties ファイルにあります。このファイルには、ANSI の標準 SQL キーワードと、データベース固有キーワードの一部が含まれています。このファイルを修正して、他の SQL キーワードを追加することもできます。このファイルで指定されていないデータベース用の SQL キーワードを追加する場合は、ColdFusion で使用できるように、そのキーワードを「ANSI」リストにも追加する必要があります。

関係の定義

関係は ORM の最も重要な部分です。リレーショナルデータベースでは、外部キーを使用してテーブル間の関係を定義します。それに対して、オブジェクトの場合は、一方のオブジェクトから他方を参照する関連付けを使用して 2 つのオブジェクトの関係を定義します。ORM では、オブジェクトの関係をデータベースの関係にマッピングする方法を定義します。

このトピックでは、関係と関連付けを同じ意味合いで使用します。

関係のマッピングを定義する方法を習得するには、いくつかの重要な概念を理解しておく必要があります。

- ソースオブジェクト : 関連するオブジェクトへの参照を含むオブジェクトを、関係のソースと呼びます。
- ターゲットオブジェクト : 参照されているオブジェクトまたは関連付けられているオブジェクトを、関係のターゲットと呼びます。
- 方向と誘導可能性 : リレーショナルデータベースでは、関係は常に一方向になります。つまり、あるテーブルから別のテーブルに進むことはできますが、逆方向には進めません。それに対し、オブジェクトモデルは一方向でも双方向でもかまいません。一方向の関連付けとは、ソースにはターゲットへの参照があるの対し、ターゲットにはソースに関する情報がないことを意味します。双方向の関連付けとは、両方のオブジェクトに相互の参照があり、どちらのオブジェクトからでも他方のオブジェクトに進むことができることを意味します。つまり、ソースにはターゲットへの参照があり、ターゲットにもソースへの参照があります。これはまた、両方のオブジェクトが同時にソースでもありターゲットでもあることを意味します。

  オブジェクト間の関連付けを設定するには、参照を適切に設定する必要があります。たとえば、人 (Person) と住所 (Address) の関係の場合、1 人に 1 つの住所があるので、次のように住所を人に関連付ける必要があります。

  ```
  person.setAddress(address);
  ```

  この時点で、Person オブジェクトには Address オブジェクトに関する情報がありますが、Address オブジェクトには Person オブジェクトに関する情報がありません。したがって、これは Person と Address の間の一方向の関係です。これを双方向にするには、次のようにして Person を Address に関連付ける必要があります。

  ```
  address.setPerson(person);
  ```

- 多重度 : この用語は、特定のソースが持つことのできるターゲットエンティティの数、および特定のターゲットが持つことのできるソースエンティティの数を意味します。たとえば、1 人のアーティストに複数の作品があるとします。オブジェクトモデルでは、1 つの作品に 1 人のアーティストへの参照があり、1 人のアーティストに複数の作品への参照があることになります。したがって、作品 - アーティストの関係の多重度は多対 1 であり、アーティスト - 作品の関係の多重度は 1 対多です。その他の多重度のタイプとしては、1 対 1 と多対多があります。

  このトピックでは、多重度のことを関係のタイプと呼びます。

データベーステーブル内の関係の結果として、プロパティが 2 つの永続コンポーネント間の関係を定義することを示すには、cfproperty タグの fieldtype で次のいずれかを指定します。

- one-to-one
- one-to-many
- many-to-one

- many-to-many

リンクテーブルを使用して関係を確立することもできます。リンクテーブルには、関係を構成する両方のテーブルの外部キーが含まれます。ORM は、ターゲットテーブルではなくリンクテーブルを使用してマップキー列を検索します。

関係の属性

次の表に、すべての関係タイプに関連する属性の詳細を示します。

「適用対象」列は、その属性を適用できる関係タイプを示しています。「すべて」は、その属性がすべての関係タイプに適用されることを示します。

属性	適用対象	必須 / オプション	デフォルト	説明
batchsize	one-to-many many-to-many	オプション		初期化されていないコレクションを取得するときの「バッチサイズ」を指定する整数値です。詳細については、バッチ取得を参照してください。
cacheuse	one-to-many many-to-many	オプション		この値は、セカンダリキャッシュにこのコンポーネントのデータを格納するときに使用するキャッシュ処理方式を指定する場合に使用します。 詳細については、キャッシュを参照してください。
cachename	one-to-many many-to-many	オプション	<entityname> <relationname>	この値は、セカンダリキャッシュの名前を指定する場合に使用します。 詳細については、キャッシュを参照してください。
cascade	すべて	オプション		詳細については、「cascade オプション」のセクションを参照してください。
cfc	すべて	必須		関連付けられる CFC の名前です。
constrained	one-to-one	オプション	false	他のテーブルのプライマリキーを参照するように、このテーブルのプライマリキー列に制約を設定するかどうかを指定します。 true false 詳細については、1 対 1 関係を参照してください。
fetch	すべて	オプション	select	関連オブジェクトの取得に結合クエリーまたは順次選択クエリーを使用するかどうかを指定します。有効な値は次のとおりです。 join select 詳細については、遅延ロードを参照してください。
fieldtype	すべて	必須	column	次の中から、関係マッピングのタイプを指定します。 one-to-one one-to-many many-to-one many-to-many

属性	適用対象	必須 / オプション	デフォルト	説明
fkcolumn	すべて	オプション		外部キー列を指定します。 リンクテーブルを使用して関係が確立されている場合は、ソースオブジェクトのプライマリキーを参照しているリンクテーブルの外部キー列を指定します。 ソーステーブルの複合キーを参照する複数の外部キー列を使用して関係が確立されている場合は、カンマ区切りの列名を使用する必要があります。また、指定する列名の順序が、定義されている複合キーの順序と一致している必要もあります。値を指定しない場合は、次のようになります。 • テーブルが存在し、制約が定義されている場合は、テーブルから自動的に値が選択されます。 • テーブルが存在しない場合は、自動的に値が生成されます。
foreignkeyname	one-to-one many-to-one、 many-to-many	オプション	autogenerated	外部キー制約の名前を指定します。これは、ORM によってテーブルが作成される場合にのみ使用されます。
index	many-to-one	オプション	false	外部キー列のインデックスの名前を指定します。
insert	many-to-one	オプション	true	この列を SQL の UPDATE および INSERT ステートメントに含めるかどうかを指定します。有効な値は次のとおりです。 true false 列を読み取り専用にする必要がある場合は、update=false および insert=false を設定します。
inverse	one-to-many many-to-many	オプション	false	このオブジェクトを永続化するときに、この関連付けに対して SQL クエリーを実行するかどうかを指定します。有効な値は次のとおりです。 true false 詳細については、「inverse」セクションを参照してください。
inversejoincolumn	すべて	オプション		ターゲットテーブルのプライマリキー列を参照する、結合テーブルの外部キー列を指定します。 複合キーの場合は、列名のカンマ区切りリストを使用できます。 ターゲットテーブルの複合キーを参照する複数の外部キー列が結合テーブルに含まれる場合は、カンマ区切りの列名を使用する必要があります。また、指定する列名の順序が、定義されている複合キーの順序と一致している必要もあります。値を指定しない場合は次のようになります。 • テーブルが存在し、制約が定義されている場合は、テーブルから自動的に値が選択されます。 • テーブルが存在しない場合は、自動的に値が生成されます。
lazy	すべて	オプション	true	関連付けを遅延ロードするかどうかを指定します。 true false extra 詳細については、遅延ロードを参照してください。

属性	適用対象	必須 / オプション	デフォルト	説明
linkcatalog	すべて	オプション		リンクテーブルのカタログです。
linkschema	すべて	オプション		リンクテーブルのスキーマです。
linktable	すべて	必須		リンクテーブルの名前です。
mappedby	すべて	オプション		関係では、プライマリキー以外の一意の列が外部キーで参照されている場合があります。そのような場合は、mappedby を使用して、一意のキー列にマッピングするプロパティの名前を指定します。
missingrowIgnored	many-to-one、 many-to-many、 （ColdFusion 9.0.1）one-to-one	オプション	false	有効な値は次のとおりです。 true false この値が true に設定されていて、外部キーで参照されている行が存在しない場合は、null の関連付けとして扱われます。 デフォルトは false で、その場合は例外が返されます。
name	すべて	必須		フィールドの名前です。
notnull	many-to-one	オプション	false	この属性は、ORM によってテーブルが作成されたときに、外部キー列に null 以外の制約を追加する場合に使用します。
optimisticlock	すべて	オプション	true	このプロパティを更新するときにテーブル行に対するオプティミスティックロックを取得する必要があるかどうかを指定します。有効な値は次のとおりです。 true false 詳細については、オプティミスティックロックを参照してください。
orderby	one-to-many many-to-many	オプション		関連付けられたコレクションのソートに使用する orderby 文字列を指定します。この文字列を指定するには、次の形式を使用します。 `"col1 <asc/desc> (, col2<asc/desc>)" or "col1(, col2)"` 後者の形式では、デフォルトで asc が使用されます。
readonly	one-to-many many-to-many	オプション	false	有効な値は次のとおりです。 true false true に設定すると、コレクションが変更不可になり、キャッシュ可能になります。
`remotingFetch`	すべて	オプション	false	remotingFetch 属性のデフォルト値は、プロパティの関係が 1 対 1、1 対多、多対 1、多対多のいずれであっても、false になります。クライアントサイドのデータを取得するには、この値を true に設定します。
singularname	one-to-many many-to-many	オプション	プロパティ名	生成される関係メソッドのカスタム名を定義します。CFC 間の関係に対して生成されるメソッドを参照してください。
structkeycolumn	one-to-many many-to-many type=struct			コレクションタイプが struct の場合にキーとして使用するターゲットテーブルの列を指定します。

属性	適用対象	必須 / オプション	デフォルト	説明
structkeytype	one-to-many many-to-many type =struct	オプション		type=struct の場合、キーのデータ型を指定します。 データ型の一覧については、データ型のセクションを参照してください。
type	one-to-many many-to-many	オプション		関係プロパティのデータ型を指定します。 array struct
update	many-to-one	オプション	true	この列を SQL の UPDATE ステートメントに含めるかどうかを指定します。 true false 列を読み取り専用にする必要がある場合は、update=false および insert=false を設定します。
unique	many-to-one	オプション	false	この属性は、ORM によってテーブルが作成されたときに、外部キー列に一意制約を追加する場合に使用します。これを使用すると、実質的に関係のタイプを 1 対 1 に変更できます。
uniquekey	many-to-one	オプション		単一の一意キー制約に列をグループ化します。
where	one-to-many many-to-many	オプション		この属性は、取得したコレクションにフィルタを適用するための SQL を指定する場合に使用します。詳細については、「関係に対するフィルタの適用」を参照してください。

cascade オプション

関連付けでは、一方のオブジェクトに対して実行されたアクションを他方のオブジェクトに対しても適用することが難しくなります。たとえば、部署と従業の間の 1 対多の関連付けでは、従業員を追加する場合、部署でも同じ変更を行う必要があります。Hibernate の cascade オプションを使用すると、そのような操作を自動的に実行できます。

cascade 属性では次の値を指定できます。

- all: すべての操作を関連オブジェクトに対して連鎖的に適用できます。
- save-update: 親オブジェクトが保存された場合に、関連オブジェクトも保存します。
- delete: 親オブジェクトに対して削除操作が呼び出された場合に、子オブジェクトを削除します。
- delete-orphan: これは 1 対多の関係のみに適用される特種な cascade オプションです。関連付けが既に削除されている子オブジェクトをすべて削除します。
- all-delete-orphan: すべての操作を子オブジェクトに対して連鎖的に適用し、delete-orphan アクションを実行します。
- refresh: 更新アクションを子オブジェクトに対して連鎖的に適用します。更新アクションは、オブジェクトとそのコレクションのリロードするときに使用されます。

通常、cascade 属性は多対 1 または多対多の関係では使用しません。

cascade 属性では、複数の値をカンマで区切って指定することもできます。1 対 1 または 1 対多の関係の場合、最もよく使用される値は all-delete-orphan です。

親オブジェクトが削除された場合でも子オブジェクトを残しておいてかまわない関連付けの場合は、cascade の値で save-update を使用できます。

関連オブジェクトに対するフィルタの適用

1 対多および多対多の関係では、配列または構造体が取得されます。フィルタを適用すると、一部の関連オブジェクトのみを取得できます。フィルタは where 属性で指定できます。これは、SQL の where 節です。例として、アーティストと作品の 1 対多の関連付けについて考察します。

各 Artist オブジェクトに関連付けられた未売却の作品のみを取得するには、次のようにマッピングを定義します。

```
<cfproperty name="unsoldArts" cfc="Art" fieldtype="one-to-many" fkcolumn="ARTISTID" where="issold=0">
```

inverse

双方向の関係では、関連付けプロパティの inverse 属性を使用して、オブジェクトを永続化するときに関連付けに対して SQL クエリーを実行するかどうかを指定します。

例として、ART CFC と ARTIST CFC について考察します。これらの CFC の間には、双方向の 1 対多関係があります。つまり、各 ART オブジェクトには ARTIST オブジェクトへの参照があり、ARTIST オブジェクトには ART オブジェクトへの参照があります。ARTIST とそれに関連付けられた ART を永続化するときは、オブジェクトの両方の側からデータベース内の関係を確立できます。関係の一方の側で inverse=true を設定すると、ORM で SQL が実行されるときに、そちら側が無視されるようになります。

原則として、双方向の関係では、一方の側で inverse を true に設定する必要があります。1 対多または多対 1 の関係では、「多」の側で inverse を設定する必要があります。たとえば、ARTIST-ART の関係では、ARTIST の「art」プロパティで inverse を true に設定する必要があります。

多対多の関係では、どちらの側でも inverse=true を設定できます。

1 対 1 関係

1 対 1 の関係では、ソースオブジェクトに別のターゲットオブジェクトを 1 つだけ参照する属性があり、ターゲットにも同様の属性があります。この関係は、従業員とオフィスブースの関係に相当します。各社員には 1 つのオフィスブースが割り当てられ、各オフィスブースは 1 人の従業員に属しています。

2 つの永続コンポーネント間の 1 対 1 関係を定義するには、fieldtype の値を one-to-one に設定します。

シンタックス :

```
<cfproperty name="fieldname"
fieldtype="one-to-one"
cfc="Referenced_CFC_Name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns"
cascade="cascade_options"
constrained="true|[false]"
fetch="join|[select]"
lazy="[true]|false">
```

1 対 1 関係には次の 2 つのタイプがあります。

- プライマリキーの関連付け
- 一意の外部キーの関連付け

プライマリキーの関連付け

このタイプの関連付けでは、1 つのテーブルのプライマリキーが、別のテーブルのプライマリキーを参照します。つまり、両方のテーブルが同じプライマリキーを共有します。

次の例は、このマッピングを定義する方法を示しています。

例

例として、EMPLOYEE と OFFICECUBICLE の関係について考察します。両方のテーブルは同じプライマリキーを共有しています。これらのテーブルのマッピングは次のようになります。

- EMPLOYEE.cfc

```
<cfcomponent persistent="true" table="Employee">
    <cfproperty name="id" fieldtype="id" generator="native">
    <cfproperty name="firstname">
    <cfproperty name="lastname">
    <cfproperty name="officecubicle" fieldtype="one-to-one" cfc="OfficeCubicle">
</cfcomponent>
```

- OFFICECUBICLE.cfc

```
<cfcomponent persistent="true" table="OfficeCubicle">
    <cfproperty name="id" fieldtype="id" generator="foreign" params="{property='Employee'}"
ormtype="int">
    <cfproperty name="Employee" fieldtype="one-to-one" cfc="Employee" constrained="true">
    <cfproperty name="Location">
    <cfproperty name="Size">
</cfcomponent>
```

fieldtype=one-to-one は、このプロパティが 1 対 1 のプロパティであることを示しています。

OFFICECUBICLE.cfc の Employee プロパティで constrained=true が設定されているため、OFFICECUBICLE テーブルには、自身の ID として EMPLOYEE テーブルの ID を参照するように制約が設定されています。

EMPLOYEE テーブルの ID は自動生成されます。OFFICECUBICLE テーブルの ID は、Employee テーブルの ID と一致している必要があります。そのためには、generator="foreign" を設定します。foreign ジェネレータは入力として 1 つの 'property' パラメータを取ります。このパラメータは OFFICECUBICLE エンティティの関係プロパティ名 (この例では「EMPLOYEE」) に設定する必要があります。

この時点で、両方のテーブル内で関連付けられている行のプライマリキーの値は一致している必要があります。マッピングされるテーブルに制約が適用されるコンポーネントのマッピングの ID ジェネレータアルゴリズムは、foreign に設定されている必要があります。

一意の外部キーの関連付け

このタイプの関連付けでは、1 つのテーブルの外部キーが別のテーブルのプライマリキーを参照し、外部キー列に一意制約が適用されます。この関係を定義するには、テーブルに外部キー列を含む CFC の関係プロパティで fkcolumn 属性を指定する必要があります。関係の相手側では、mappedby 属性を使用する必要があります。

シンタックス

```
<cfproperty
name="fieldname"
fieldtype="one-to-one"
cfc="Referenced_CFC_Name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns"
mappedby="Mapped_Field_name_in_referenced_CFC"
cascade="none"
fetch="join|[select]"
lazy="[true]|false">
```

注意：mappedby 属性を fkcolumn 属性と組み合わせて指定することはできません。

例

次に示す EMPLOYEE と OFFICECUBICLE の例では、OFFICECUBICLE に EMPLOYEEID という外部キー列があります。この外部キーは、Employee テーブルのプライマリキーを参照しています。OFFICECUBICLE には自動生成されるプライマリキーがありますが、関係の確立には使用されていません。

EMPLOYEE.cfc

```
<cfcomponent persistent="true" table="Employee">
    <cfproperty name="EmployeeID" fieldtype="id" generator="native">
    <cfproperty name="firstname">
    <cfproperty name="lastname">
    <cfproperty name="officecubicle" fieldtype="one-to-one" cfc="officecubicle" mappedby="Employee">
</cfcomponent>
```

OFFICECUBICLE.cfc

```
<cfcomponent persistent="true" table="officecubicle">
    <cfproperty name="id" fieldtype="id" generator="native">
    <cfproperty name="Employee" fieldtype="one-to-one" cfc="Employee" fkcolumn="EmployeeID">
    <cfproperty name="Location">
    <cfproperty name="Size">
</cfcomponent>
```

- OFFICECUBICLE エンティティの fkcolumn="EmployeeID" により、EmployeeID が OFFICECUBICLE テーブルの外部キー列であることが指定されています。
- mappedby="Employee" は、1 対 1 の関係が、OFFICECUBICLE エンティティの外部キープロパティ「EMPLOYEE」との関係であり、プライマリキーとの関係ではないことを示しています。
- Employee エンティティでは、fkcolumn を指定できません。

この例では、OFFICECUBICLE エンティティに自動生成される独立したプライマリキーがあります。

1 対多関係

1 対多の関係では、ソースオブジェクトにターゲットオブジェクトのコレクションを格納するフィールドがあります。これらのターゲットオブジェクトも、ソースオブジェクトに対する逆方向の関係を持つ場合があります。この関係は、ソーステーブルのプライマリキーにマッピングする外部キーをターゲットテーブルに設定することで確立されます。

アーティストと作品の関係は 1 対多関係の例であり、1 人のアーティストに複数の作品があります。

2 つの永続コンポーネント間の 1 対多関係を定義するには、cfproperty タグで fieldtype の値を one-to-many に設定します。ソースオブジェクトには、ターゲットオブジェクトのコレクションが含まれます。ColdFusion では、次のいずれかの型のコレクションを使用できます。

- 配列
- 構造体

このコレクションは、永続化に対応したコレクションです。このコレクションに対する追加操作または削除操作は、自動的にデータベースに永続化されます。

配列

Artist オブジェクトには、Art オブジェクトを配列として格納できます。CFC でこのマッピングを定義するには、次のシンタックスを使用します。

シンタックス

```
<cfproperty
name="field_name"
fieldtype="one-to-many"
cfc="Referenced_CFC_name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns "
type="array"
orderby="order_by_string"
cascade="cascade_options"
lazy="[true]|false|extra"
fetch="join|[select]"
inverse="true|[false]"
batchsize="N"
optimisticlock="[true]|false"
readonly="true|[false]">
```

アーティストと作品の例では、Artist.cfc の関係プロパティが次のように定義されています。

```
<cfproperty name="art" type="array" fieldtype="one-to-many" cfc="Art" fkcolumn="ARTISTID">
```

- type=array では、Artist オブジェクトに Art オブジェクトを配列として格納することを指定しています。
- fkcolumn="ArtistID" では、ARTIST テーブルのプライマリキーを参照している ARTISTID が外部キー列であることを指定しています。

構造体

Artist オブジェクトには、Art オブジェクトを構造体として格納できます。ART テーブルの任意の列をキーとして使用できます (通常はプライマリキーまたは一意のキーを使用します)。値は Art オブジェクトです。このマッピングを定義するには、次のシンタックスを使用します。

シンタックス

```
<cfproperty
name="field_name"
fieldtype="one-to-many"
cfc="Referenced_CFC_name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns"
type="struct"
orderby="order_by_String"
structkeycolumn = "Structure_Key_Column"
structkeytype="ormtype"
cascade="cascade_options"
lazy="[true]|false|extra"
fetch="join|[select]"
inverse="true|[false]"
batchsize="N"
optimisticlock="[true]|false"
readonly="true|[false]">
```

アーティストと作品の例では、関係プロパティを次のように定義できます。

```
<cfproperty name="art" type="struct" fieldtype="one-to-many" cfc="Art" fkcolumn="ARTISTID"
structkeytype="int" structkeycolumn="ArtID">
```

- type=struct では、Artist オブジェクトに Art オブジェクトを構造体として格納することを指定しています。
- structkeycolumn="ArtID" では、構造体のキーが ArtID であることを指定しています。

ARTID が Art テーブルのプライマリキーであることに注意してください。

- structkeytype="int" では、データ型が structkeycolumn であることを指定しています。
- fkcolumn="ArtistID" では、Artist テーブルのプライマリキーを参照している ARTISTID が外部キー列であることを指定しています。

多対 1 関係

多対 1 関係は、1 対多関係の逆です。この関係では、複数のソースオブジェクトが同じターゲットオブジェクトを参照できます。

作品とアーティストの間の関係はこの関係の例であり、複数の作品が同じアーティストに属しています。この関係は、ソーステーブルの外部キーでターゲットテーブルのプライマリキーを参照することで確立されます。

2 つの永続コンポーネント間の多対 1 関係を定義するには、cfproperty タグで fieldtype の値を many-to-one に設定します。

シンタックス

```
<cfproperty
name="fieldname"
fieldtype="many-to-one"
cfc="Referenced_CFC_Name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns"
column="Foreign_Key_Column"
mappedby="Mapped_Field_name_in_referenced_CFC"
cascade="cascade_options"
fetch="join|[select]"
lazy="true|false"
insert="[true]|false"
update="[true]|false"
optimisticlock="[true]|false"
missingrowIgnored="true|[false]">
```

作品とアーティストの例では、ART.cfc の関係プロパティを次のように定義できます。

```
<cfproperty name="artist" fieldtype="many-to-one" fkcolumn="artistid" cfc="Artist">
```

fkcolumn="ARTISTID" では、Art テーブルの外部キー列が、ARTIST テーブルのプライマリキー ARTISTID を参照していることを指定しています。

多対多関係

多対多の関係では、ソースオブジェクトにターゲットオブジェクトのコレクションが含まれ、ターゲットオブジェクトにもソースオブジェクトのコレクションが含まれます。

注文と製品の関係は多対多関係の例であり、1 つの注文には複数の製品が含まれ、1 つの製品には複数の注文が含まれます。

この関係は、「LinkTable」と呼ばれる第 3 のテーブルを使用して確立されます。LinkTable には、関係を構成する両方のテーブルの外部キーが含まれます。ORM は、ターゲットテーブルではなく LinkTable でマップキー列を検索します。

前に示した注文と製品の例では、LinkTable を使用して多対多関係が確立されます。

2 つの永続 CFC 間の多対多関係を定義するには、cfproperty タグで fieldtype="many-to-many" を設定します。

注意：fkcolumn の名前が指定されていない場合は、「#relationName#_ID」という形式で fkcolumn の名前が生成されます。

シンタックス
Order.cfc

```
<cfproperty
name="fieldname"
fieldtype="many-to-many"
cfc="fully qualified name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or a composite key with comma-separated primary key columns"
mappedby="Property in the target component that is referenced by fkcolumn in join table"
type="[array]|struct"
orderby="order by String
structkeycolumn="The structure key column name"
structkeydatatype="datatype".
cascade="cascade options" inverse="true|[false]" lazy = "[true]|false" [optional] fetch="join|[select]"
[optional] batchsize="integer" optimisticlock="[true]|false" readonly="true|[false]"
missingrowIgnored="true|[false]">
```

注文と製品の例では、2 つの外部キー (OrderId と ProductId) を含む第 3 のテーブル「OrderProduct」を使用して、多対多関係が確立されます。OrderId は注文テーブルのプライマリキー orderId を参照し、ProductId は製品テーブルのプライマリキー productId を参照します。この関係は次のように定義できます。

- Order.cfc

  ```
  <cfproperty
  name="products"
  fieldtype="many-to-many"
  CFC="Product"
  linktable="Order_Product"
  FKColumn="orderId"
  inversejoincolumn="productId"
  lazy="true"
  cascade="all"
  orderby="productId">
  ```

- Product.cfc

  ```
  <cfproperty
  name="orders"
  fieldtype="many-to-many"
  CFC="Order"
  linktable="Order_Product"
  FKColumn="productId"
  inversejoincolumn="orderId"
  lazy="true"
  cascade="all"
  orderby="orderId">
  ```

  fkcolumn では、ソーステーブルのプライマリキーを参照するリンクテーブルの外部キーを指定しています。

  InverseJoinColumn では、ターゲットテーブルのプライマリキーを参照するリンクテーブルの外部キーを指定しています。この属性では、inversejoincolumn="field1, field2" のように、複合キーの値も指定できます (この場合、複合キーは field1 と field2 で構成されます)。

高度なマッピング

コレクションマッピング

コレクションマッピングは、1 対多関係のマッピングに似ています。ただし、コレクションマッピングでは、永続的なターゲットオブジェクトのコレクションではなく、値のコレクションを使用します。

例として、Artist テーブルと Art テーブルについて考察します。各 Artist オブジェクトに作品オブジェクトではなく作品名の配列を格納する場合は、コレクションマッピングを使用する必要があります。

CFC でコレクションマッピングを定義するには、cfproperty タグで fieldtype="collection" を使用します。

コレクションは Array でも Struct でもかまいません。

配列

シンタックス

```
name="field_name"
fieldtype="collection"
type="array"
table="table_name"
fkcolumn="foreign_key_column_name"
elementtype="ormtype"
elementColumn="column_name from the link table that should be
used for populating"
orderby="order by string"
lazy = "true|[false]"
readonly="true|[false]"
optimisticlock="[true]|false"
batchsize="batch size">
```

例

各 Artist オブジェクトに作品オブジェクトではなく作品名の配列を格納する場合、このマッピングは Artist.cfc で次のように定義できます。

```
<cfproperty name="artNames" fieldtype="collection" type="array" table="ART" fkcolumn="ARTISTID"
elementcolumn="ARTNAME" elementtype="string">
```

属性

属性	必須 / オプション	デフォルト	説明
batchsize	オプション		初期化されていないコレクションを取得するときの「バッチサイズ」を指定する整数値です。詳細については、遅延ロードを参照してください。
elementColumn	必須		コレクションとして取得したデータを格納する列名を指定します。
elementtype	オプション	String	選択した列のデータ型。詳細については、ORM データ型を参照してください。
fieldtype	必須		"collection" に設定する必要があります。
fkcolumn	オプション		指定したテーブル内の外部キー列。 外部キー列を指定せずに ormsettings で useDBForMapping を true に設定した場合は、データベースを調べて外部キー列が自動的に決定されます。

属性	必須 / オプション	デフォルト	説明
lazy	オプション	true	遅延ロードを行うかどうかを指定します。 true false 詳細については、遅延ロードを参照してください。
name	必須		コレクションの名前。
optimisticlock	オプション	true	ロック方式を指定します。 true false
orderBy	オプション		ソート基準の文字列を指定します。
readonly	オプション	false	true false true に設定すると、コレクションが変更不可になり、キャッシュ可能になります。
table	必須		値を取得するテーブルの名前。
type	オプション	array	コレクションのタイプを指定します。 array struct

構造体

シンタックス

```
<cfproperty
name="field_name"
fieldtype="collection"
type="struct"
table="table_name"
fkcolumn="foreign_key_column_name"
structkeycolumn="column in the target table to be used as key in the struct"
structkeytype="ormtype of the key in the struct"
elementtype="ormtype of the valye in the struct"
elementColumn="column name from the table that should be used in
value of struct"
orderby="order by string"
lazy = "[true]|false"
readonly="true|[false]"
optimisticlock="[true]|false"
batchsize="batch size">
```

属性	必須 / オプション	デフォルト	説明
batchsize	オプション		このコレクションのインスタンスを遅延取得する場合の「バッチサイズ」を指定する整数値。
elementcolumn	必須		コレクションとして取得したデータを格納する列名を指定します。
elementtype	必須		値のデータ型。詳細については、ORM データ型を参照してください。
fieldtype	必須		collection に設定する必要があります。

属性	必須 / オプション	デフォルト	説明
fkcolumn	オプション		テーブル内の外部キー列。 外部キー列を指定せずに ORMSetting で useDBForMapping を true に設定した場合は、データベースを調べて外部キー列が自動的に決定されます。
lazy	オプション	true	遅延ロードを行うかどうかを指定します。 true false 詳細については、遅延ロードを参照してください。
name	必須		コレクションプロパティの名前。
optimisticlock	オプション	true	true false
orderby	オプション		ソート基準の文字列を指定します。
readonly	オプション	false	有効な値は次のとおりです。 true false true に設定すると、コレクションは変更不可になり、キャッシュ可能になります。
structkeycolumn	必須		構造体のキーとして使用するテーブル列の名前。
structkeyType	必須		type=struct の場合、キーのデータ型を指定します。 データ型の一覧については、データ型のセクションを参照してください。
table	必須		コレクションを取得するテーブルの名前。
type	オプション	array	コレクションのタイプを指定します。 array struct

継承マッピング

永続化する必要があるオブジェクトが階層である場合は、その階層全体が永続化されるように、そのオブジェクト階層の CFC をリレーショナルテーブルにマッピングする必要があります。

継承マッピングには、次のような方法があります。

- 階層ごとのテーブル
- サブクラスごとのテーブル (識別子なし)
- サブクラスごとのテーブル (識別子あり)

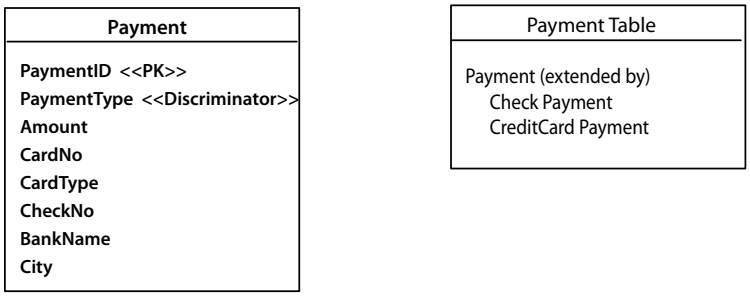
階層ごとのテーブル

このモデルでは、オブジェクト階層を 1 つのテーブルで永続化します。このテーブルには、階層を構成する CFC の全プロパティに対応する列が含まれます。行によって表される具体的なサブクラスは、discriminator 列の値に基づいて識別されます。この方法では、階層を構成するすべての CFC が同じテーブル名を持ちます。

注意：discriminator 列と discriminator 値が指定されていない場合は、デフォルトの discriminator 列名と値が使用されます。

例

次の例は、階層ごとのテーブルを実装する方法を示しています。



階層ごとのテーブルの例

上の図の場合、discriminator 列は PaymentType です。PaymentType の値がクレジットカードであるか小切手であるかに応じて、各行は CreditCardpayment オブジェクトまたは checkPayment オブジェクトとして表されます。

次の例は、階層ごとのテーブルをモデル化する方法を示しています。

Payment.cfc (親クラス)

```
<cfcomponent persistent="true" table="Payment" discriminatorColumn="paymentType">  <cfproperty name="id">
  <cfproperty name="amount">
</cfcomponent>
```

CreditCardPayment.cfc

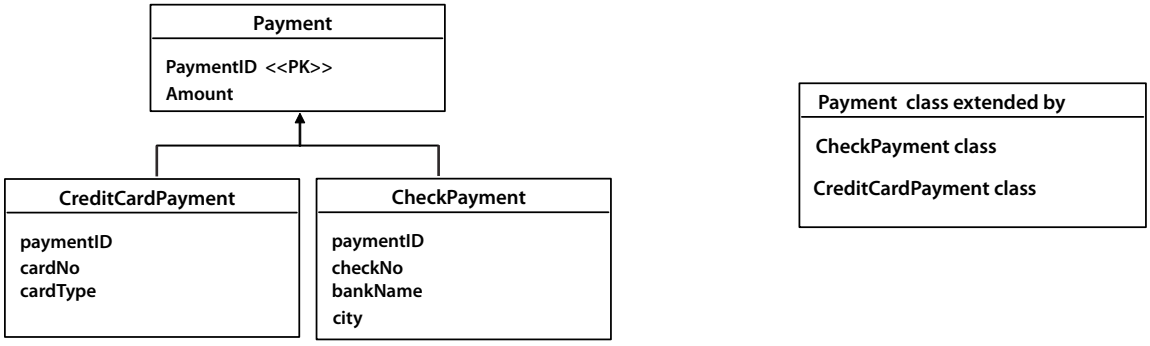
```
<cfcomponent persistent="true" extends="Payment" table="Payment" discriminatorValue="CCard">
    <cfproperty name="cardNo">
    <cfproperty name="cardType">
</cfcomponent>
```

CheckPayment.cfc

```
<cfcomponent persistent="true" extends="Payment" table="Payment"  discriminatorValue="check">
    <cfproperty name="checkNo">
    <cfproperty name="bankName">
    <cfproperty name="city">
</cfcomponent>
```

サブクラスごとのテーブル (識別子なし)

このモデルでは、階層内のクラスごとに個別のテーブルが存在し、これらのテーブルがプライマリキーで結合されます。オブジェクトが永続化されると、親コンポーネントのプロパティは親テーブルに保存され、残りのプロパティは子テーブルに保存されます。



サブクラスごとのテーブル (識別子なし)

上の図の場合、これらのテーブルは paymentId を結合列として結合されます。テーブルをモデル化するには、次のようにします。

Payment.cfc

```
<cfcomponent persistent="true" table="Payment">
  <cfproperty name="paymentId">
  <cfproperty name="amount">
</cfcomponent>
```

CreditCardpayment.cfc

```
<cfcomponent persistent="true" extends="Payment" table="CreditCardPayment"
             joinColumn="paymentId">
    <cfproperty name="cardNo">
  <cfproperty name="cardType">
</cfcomponent>
```

CheckPayment.cfc

```
<cfcomponent persistent="true" extends="Payment" table="CheckPayment" joinColumn="paymentId">
    <cfproperty name="checkNo">
  <cfproperty name="bankName">
  <cfproperty name="city">
</cfcomponent>
```

CreditCardPayment タイプのオブジェクトが永続化されると、プロパティ amount は Payment テーブルに保存され、プロパティ cardNo と cardType は CreditCardPayment テーブルに保存されます。CreditCardPayment のプライマリキーは、Payment テーブルのプライマリキーと同じままです。

サブクラスごとのテーブル (識別子あり)

このモデルは、サブクラスごとのテーブル (識別子なし) のモデルとほぼ同じですが、親テーブルに識別子列がある点が異なります。また、子コンポーネントの cfcomponent タグに disciminatorValue 属性が含まれる点も異なります。

次の例は、discriminator 属性を含むサブクラスごとのテーブルを示しています。

Payment.cfc

```
<cfcomponent persistent="true" table="Payment" discriminatorColumn="paymentType">
    <cfproperty name="paymentId">
    <cfproperty name="amount">
</cfcomponent>
```

CreditCardPayment.cfc

```
<cfcomponent persistent="true" extends="Payment" table="CreditCardPayment" joinColumn="paymentId"
discriminatorValue="CCard">
    <cfproperty name="cardNo">
    <cfproperty name="cardType">
</cfcomponent>
```

CheckPayment.cfc

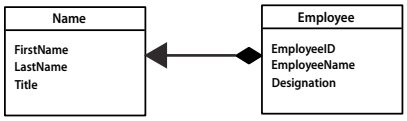
```
<cfcomponent persistent="true" extends="Payment" table="CheckPayment" joinColumn="paymentId"
discriminatorValue="Check">
    <cfproperty name="checkNo">
    <cfproperty name="bankName">
    <cfproperty name="city">
</cfcomponent>
```

CreditCardPayment タイプのオブジェクトが永続化されると、プロパティ amount は Payment テーブルに保存され、プロパティ cardNo と cardType は CreditCardPayment テーブルに保存されます。CreditCardPayment のプライマリキーは、Payment テーブルのプライマリキーと同じままです。PaymentType の値は、各オブジェクトの disciminatorColumn 属性の値です。

埋め込みマッピング

このマッピングは、親のデータとともに永続化する必要がある埋め込みオブジェクトが CFC にある場合に使用します。埋め込みオブジェクトの CFC では、cfcomponent タグの embedded 属性を true に設定する必要があります。

重要： 埋め込みオブジェクトを永続オブジェクトにすることは使用できません。この機能は、Hibernate マッピングファイル (.hbmxml ファイル) で Hibernate マッピングが明示的に定義されている場合にのみサポートされます。



Employee.cfc に埋め込められた Name.cfc

この図は、Employee.cfc の EmployeeName フィールドが Name.cfc のオブジェクトになっていることを示しています。データベースでは、これらのオブジェクトはどちらも、Employee テーブル内の 1 つの行として永続化されます。Name オブジェクト自体に固有の ID はありません。このマッピングは次のようにモデル化できます。

name.cfc

```
<cfcomponent embedded="true">
    <cfproperty name="FirstName">
    <cfproperty name="LastName">
    <cfproperty name=" Title">
</cfcomponent>
```

employee.cfc

```
<cfcomponent persistent="true">
    <cfproperty name="EmployeeID">
    <cfproperty name="EmployeeName">
    <cfproperty name="Designation">
</cfcomponent>
```

employee.hbmxml

```
<hibernate-mapping >
    <class  name="cfc:Employee" table="Employees">
        <id name="EmployeeID" type="integer" column="EmployeeID">
            <generator class="native"/>
        </id>
        <component name="EmployeeName" class="cfc:Name">
        <property name="LastName" type="string" column="LastName"/>
        <property name="FirstName" type="string" column="FirstName"/>
        <property name="Title" type="string" column="Title"/>
        </component>
        <property name="Designation" type="string" column="Designation"/>
    </class>
</hibernate-mapping>
```

永続 CFC に埋め込みオブジェクトのコレクションがある場合は、次の例に示すように、このマッピングを XML でも定義する必要があります。ここでは、employee オブジェクトに IMData オブジェクトのコレクションがあります。IMData オブジェクトが永続オブジェクトでないことに注意してください。

employee.cfc

```
<cfcomponent     persistent="true">
    <cfproperty name="EmployeeID">
    <cfproperty name="EmployeeName">
    <cfproperty name= "IMIDs" type="array">
    <cfproperty name="Designation">
</cfcomponent>
```

IMData.cfc

```
<cfcomponent embedded="true">
    <cfproperty name="type">
    <cfproperty name="ID">
</cfcomponent>
```

employee.hbmxml

```
<hibernate-mapping>
    <class  name="cfc:Employee" table="Employees">
        <id name="EmployeeID" type="integer" column="EmployeeID">
                        <generator class="native"/>
        </id>
        <property name="EmployeeName" type="string" column="EmployeeName"/>
            <bag name="IMIDs" table="IMData" lazy="true">
                <key column="EmployeeID" />
                <composite-element class="cfc:IMData">
                    <property name="type" type="string" column="Type"/>
                    <property name="ID" type="string" column="ID"/>
                </composite-element>
            </bag>
            <property name="Designation" type="string" column="Designation"/>
    </class>
</hibernate-mapping>
```

Emp.cfm

```
<cfscript>
    employee = EntityNew("Employee");
    employee.setEmployeeName("Dan Watson");
    imdata1 = new IMData();
    imdata1.setType("IMClient1");
    imdata1.setID("msngrId1");
    imdata2 = new IMData();
    imdata2.setType("IMClient 2");
    imdata2.setID("msngrId2");
    employee.setIMIDs([imdata1, imdata2]);
    EntitySave(employee);
</cfscript>
```

CFC での結合マッピング

結合マッピングは、1 つの CFC のプロパティを複数のテーブルにマッピングする場合に使用します。この場合は、結合列を使用してテーブルを結合します。たとえば、Employee テーブルと Address テーブルを Employees という 1 つの CFC にマッピングするとします。この場合は、"employee.cfc" の一部のフィールドを Employee テーブルで永続化し、その他のフィールドを Address テーブルで永続化します。Address テーブルで永続化する必要があるフィールドに関しては、joincolumn 属性と table 属性を指定する必要があります。この例では、table 属性を Address に設定し、joinColumn 属性を AddressID に設定します。

注意： Hibernate は、結合の取得にデフォルトで OUTER JOIN を使用します。INNER JOIN の場合は、HQL を使用します。

"employee.cfc" の例を次に示します。

Employee.cfc

```
<cfcomponent persistent="true">
    <cfproperty name="id">
    <cfproperty name="name">
    <cfproperty name="houseno" column="houseno" table="Address" joincolumn="addressId">
    <cfproperty name="street" table="Address" joincolumn="addressId">
    <cfproperty name="city" table="Address" joincolumn="addressId">
    <cfproperty name="country" table="Address" joincolumn="addressId">
</cfcomponent>
```

Hibernate マッピングファイルでの ORM マッピングの定義

ColdFusion では、標準の Hibernate マッピング XML ファイルを使用してオブジェクトとデータベースの間のマッピングを定義することもできます。Hibernate マッピングでは Java クラスと CFC の両方を使用できます。

Hibernate マッピングファイルを使用する際には、次の点に注意してください。

- Hibernate 設定ファイルの拡張子は *.hbmxml です。
- ファイルはアプリケーションフォルダーに配置されます。
- クラス名は「cfc:<CFC の完全修飾名 >」という形式で指定する必要があります。Hibernate マッピングでパッケージを指定する場合は、"cfc:<CFC の名前 >" という形式でクラス名を指定する必要があります。
- entityname 属性はオプションです。この属性を指定しない場合は、デフォルトでコンポーネント名が使用されます。たとえば、artgallery.art というコンポーネントの場合、entityname 属性の値はデフォルトで「Art」になります。
- エンティティ名は 1 つのアプリケーションに対して一意になっている必要があります。同じ名前のコンポーネントが 2 つ存在する場合、それぞれのコンポーネントが別々のパッケージに含まれているとしても、異なるエンティティ名を指定してください。

Hibernate マッピングの例を次に示します。

```
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class lazy="true" name="cfc:artGallery.Art" schema="APP" table="Art">
    <id name="artid" type="int">
        <column length="10" name="ARTID"/>
        <generator class="identity"/>
    </id>
        <property name="artname" type="string">
            <column length="50" name="ARTNAME"/>
        </property>

        <property name="price" type="java.math.BigDecimal">
            <column length="19" name="PRICE"/>
        </property>

        <property name="largeimage" type="string">
            <column length="30" name="LARGEIMAGE"/>
        </property>

        <property name="mediaid" type="int">
            <column length="10" name="MEDIAID"/>
        </property>

        <property name="issold" type="boolean">
            <column length="5" name="ISSOLD"/>
        </property>
        <many-to-one class="cfc:artGallery.Artists"  column="artistid" name="artist"/>
    </class>
</hibernate-mapping>
```

プロパティなしの CFC のマッピング

永続 CFC にプロパティがなく、テーブルが存在し、そのテーブルを指定した場合は、すべてのテーブル列がプロパティとして追加されます。

例

Artist.cfc

```
component persistent="true" table="artists"
{
}
```

index.cfm

```
<cfset artists = entityLoad("Artist")>
<cfdump var="#artists#">
```

列は列プロパティとしてのみ取得されます。関係、タイムスタンプ、バージョンフィールドは追加されません。

制約

- CFC のマッピングファイルが存在する場合、プロパティは自動的には追加されません。
- 継承の場合、プロパティは自動的には CFC に追加されません。

オブジェクトの使用

エンティティオブジェクトに対しては、各種の操作を実行して、そのエンティティで自動生成されたメソッドを呼び出すことができます。

生成されるアクセサ

ORM マッピングの定義で説明されているように、オブジェクトの永続化に関連するフィールドを定義するには、CFC で cfproperty を使用します。ColdFusion では、CFC のプロパティごとに、呼び出し可能なアクセサメソッド (getter と setter) が生成されます。詳細については、暗黙的な Get 関数と Set 関数を参照してください。たとえば、Artist で次のプロパティが定義されているとします。

```
<cfproperty name="firstName" >
```

Artist オブジェクトでは、次の 2 つのメソッドが生成されます。

- setFirstName(firstName)
- getFirstName()

これらのメソッドは、CFC の通常のメソッドと同じように呼び出せます。プロパティの場合、生成された setter を呼び出すと、プロパティの値がオブジェクトの VARIABLES スコープに保存されます。生成された getter を呼び出すと、プロパティの値が VARIABLES スコープから取得されます。ORM では、常に、VARIABLES スコープのプロパティ値が使用されます。つまり、ORM はオブジェクトのデータをテーブルに保存した後、VARIABLES スコープからプロパティの値を取得します。テーブルからの読み取り後にオブジェクトの値が設定されると、ORM はプロパティの値を VARIABLES スコープに格納します。プロパティのアクセサメソッドを独自に定義する場合は、ORM がアクセスできるように、VARIABLES スコープにそのプロパティの値を格納する必要があります。

CFC 間の関係に対して生成されるメソッド

CFC で関係を定義すると、CFC で定義されている関係ごとに、関連オブジェクトの追加と削除や、関連オブジェクトの存在確認を行うためのメソッドが CFC オブジェクト内に生成されます。

関係に対して生成されるメソッドには次のものがあります。

- add< 関係プロパティ名 >()

  このメソッドは、1 対多および多対多の関係に対して生成されます。このメソッドを呼び出すと、コンポーネントの関連付けコレクション (配列または構造体) にオブジェクトが追加されます。双方向の関係では、このメソッドを呼び出しても、相手側の関連付けは設定されません。

  type="array" の場合、このメソッドのシグネチャは次のようになります。

  ```
  add<relationship_property_name>(<associated_object>)
  ```

  type="struct" の場合、このメソッドのシグネチャは次のようになります。

  ```
  add<relationship_property_name>(<key>, <associated_object>)
  ```

- boolean remove< 関係プロパティ名 >()

  このメソッドは、1 対多および多対多の関係に対して生成されます。このメソッドを呼び出すと、コンポーネントの関連コレクション (配列または構造体) からオブジェクトが削除されます。関連オブジェクトがコレクションから正常に削除された場合は、true が返されます。双方向の関係では、このメソッドを呼び出しても、相手側の関連付けは削除されません。

  type="array" の場合、このメソッドのシグネチャは次のようになります。

  ```
  boolean remove<relationship_property_name>(<associated_object>)
  ```

  type="struct" の場合、このメソッドのシグネチャは次のようになります。

  ```
  boolean remove<relationship_property_name>(<key>).
  ```

- boolean Has< 関係プロパティ名 >()

  このメソッドは、すべての関係に対して生成されます。1 対多および多対多の関係の場合、このメソッドを呼び出すと、関連付けコレクションが空であるかどうかが確認されます。関連付けコレクションが空の場合は、true が返されます。1 対 1 および多対 1 の関係の場合、このメソッドを呼び出すと、関連オブジェクトが存在するかどうかが確認されます。

- boolean Has< 関係プロパティ名 >(< 関連オブジェクト >)

  このメソッドは、1 対多および多対多の関係に対して生成されます。このメソッドを呼び出すと、指定された関連オブジェクトが関連付けコレクションに存在するかどうかが確認されます。存在する場合は true が返されます。

  type="array" の場合、このメソッドのシグネチャは次のようになります。

  ```
  boolean has<relationship_property_name>(<associated_object>)
  ```

  type="struct" の場合、このメソッドのシグネチャは次のようになります。

  ```
  boolean has<relationship_property_name>(<key>)
  ```

例

ここでは、アーティスト (ARTISTS テーブル) と作品 (ART テーブル) の関係を例にとって説明します。この例では、アーティストと作品の関係は 1 対多です。

Artist.cfc

```
<cfcomponent persistent="true" schema="APP" table="Artists">
    <cfproperty name="artistid" fieldtype="id"/>
    <cfproperty name="firstname"/>
    <cfproperty name="lastname"/>
    <cfproperty name="state"/>
    <cfproperty name="art" fieldtype="one-to-many" cfc="Art" fkcolumn="ArtistID" >
</cfcomponent>
```

Art.cfc

```
<cfcomponent persistent="true" schema="APP" table="Art">
    <cfproperty name="artid" fieldtype="id"/>
    <cfproperty name="artname"/>
    <cfproperty name="issold"/>
</cfcomponent>
```

この例では、Artist に Art との関係フィールド art があります。Artist オブジェクトには、次のメソッドが暗黙的に追加されます。

- addArts(Art art)
- booleanremoveArts(Art art)
- booleanhasArts()
- booleanhasArts(Art art)

生成される関係メソッドの名前は、singularName 属性を使用して変更できます。たとえば、Artist の関係プロパティが次のように指定されているものとします。

```
<cfproperty name="art" fieldtype="one-to-many" cfc="Art" fkcolumn="ArtistID"  singularName="Art">
```

この場合は、次のメソッドが生成されます。

- addArt(Art art)
- removeArt(Art art)
- hasArt()
- hasArt(Art art)

ORM オブジェクトに対する作成、読み取り、更新、削除操作の実行

データ中心アプリケーションでは、次の操作をデータベースに対して実行できます。

- 挿入 (作成)
- 更新
- 取得
- 削除

ColdFusion アプリケーションでオブジェクトリレーショナルモデルを定義した後は、ColdFusion ORM で提供されるメソッドを使用して、オブジェクトに対して CRUD 操作を直接実行できます。ColdFusion ORM では、データベースでのオブジェクトの永続化が処理されます。

エンティティの作成

EntityNew

指定されたエンティティ名を使用して永続 CFC のオブジェクトが作成されます。これは CreateObject とほぼ同じですが、エンティティの作成時には entityname が使用されるのに対し、CreateObject では CFC 名が使用されます。指定された entityname を持つ CFC がアプリケーションで定義されていない場合は、エラーが発生します。

永続 CFC に init メソッドがある場合、関数 EntityNew は、オブジェクトの作成中に init メソッドを呼び出します。

シンタックス

```
<entity> EntityNew("<entityName>")
```

エンティティの保存

EntitySave

指定されたエンティティとそれに関連するすべてのエンティティのデータがデータベースに保存されます (既存のデータがある場合は更新されます)。

指定されたエンティティに関して新しいレコードを挿入するか、既存のレコードを更新するかは、ColdFusion によって自動的に判断されます。forceinsert=true の場合、ColdFusion は常に新しいレコードとしてエンティティを挿入しようとします。

このメソッドを呼び出しても、挿入または更新の SQL はすぐには実行されない場合があります。さまざまな SQL ステートメントが実行待ちになっているときは、パフォーマンス上の理由により、バッチとして実行される可能性があります。これらの SQL ステートメントは、ORM セッションがフラッシュされると実行されます。

シンタックス

```
EntitySave(entity, [forceinsert])
```

パラメータ	説明
entity	データベースに保存する必要があるエンティティ。
forceinsert	true の場合、ColdFusion は常に新しいレコードとしてエンティティを挿入しようとします。

例：

```
<cfset artist = EntityNew("Artist")>
<cfset artist.setFirstName("Marcia")>
<cfset artist.setlastName("Em")>
<cfset EntitySave(artist)>
```

オブジェクトの更新

EntitySave

オブジェクトを更新する方法は、オブジェクトを保存する方法と同じです。更新する必要があるオブジェクトをロードし、オブジェクトを更新して、オブジェクトを保存します。

シンタックス

```
EntitySave(entity, [forceinsert])
```

例

次の例では、Artist ID 1 を持つアーティストのファーストネームを変更します。

```
<cfset artist1 = EntityLoad("Artist", 1, true)>
<cfset artist1.setFirstName("Garcia")>
<cfset EntitySave(artist1)>
```

オブジェクトの読み取りおよびロード

エンティティをロードするには、EntityLoad メソッドを使用します。すべての EntityLoad メソッドは、入力としてエンティティ名を取ります。

永続 CFC に init メソッドがある場合、このメソッドは、オブジェクトの作成中に init メソッドを呼び出します。

シンタックス

```
EntityLoad (entityname)
EntityLoad (entityname, id [, unique])
EntityLoad (entityname, filtercriteria [,unique]
EntityLoad(entityname, filtercriteria, sortorder [, options])
EntityLoadByExample(sampleentity [, unique])
EntityReload(entity)
```

例

- EntityLoad (entityname)

  指定された名前を持つエンティティの配列をロードして返します。たとえば、「artist」CFC のオブジェクトをすべて取得するには、次のようにします。

  ```
  <cfset artist = EntityLoad('ARTIST')>
  ```

- EntityLoad (entityname, id [, unique])

  プライマリキーの値が id に設定されているエンティティをロードして返します。デフォルトでは、エンティティは配列として返されます。unique が true に設定されている場合は、エンティティが返されます。

  エンティティに複合キーがある場合は、キーと値のペア (ColdFusion 構造体) として id を指定する必要があります。

  例 1:

  この例では、PK 100 を持つ Artist オブジェクトをロードし、その Artist オブジェクトを含む単一要素の配列を返します。

  ```
  <cfset artistArr = EntityLoad('Artist', 100)>
  ```

  例 2:

  この例では、PK 100 を持つ Artist オブジェクトをロードし、その Artist オブジェクトを返します。

  ```
  <cfset artistobj = EntityLoad('Artist', 100, true)>
  ```

  例 3:

  この例では、複合キー OrderID=100 および ProductID=1 を持つ OrderDetail オブジェクトをロードし、その OrderDetail オブジェクトを返します。

  ```
  <cfset orderDetail = EntityLoad('orderdetails', {OrderID=100, ProductID=1}, true)>
  ```

- EntityLoad (entityname, filtercriteria [,unique]

  filtercriteria に一致するエンティティ名を持つエンティティの配列をロードして返します。filtercriteria は、プロパティの名前と値で構成されるキーと値のペア (ColdFusion 構造体) です。filtercriteria にキーと値のペアが複数ある場合は、常に AND 演算子が使用されます。この filtercriteria に一致するレコードが 1 件だけであることがわかっている場合は、unique=true を指定すると、配列ではなく単体のエンティティが返されます。unique=true になっているときに、複数のレコードが返されると、例外が発生します。たとえば、州が「CA」に設定されているすべてのアーティストの詳細を取得するには、次のようにします。

  ```
  <cfset artistsFromCA = EntityLoad('Artist', {state="CA"})>
  ```

  一意のオブジェクトを取得するには、unique= "true" を指定します。条件を満たすオブジェクトが複数ある場合は、例外が発生します。

  この例では、firstName が「Austin」で lastname が「Weber」である Artist オブジェクトをロードします。

  ```
  <cfset artist = EntityLoad('artist', {firstname="Austin", lastname="Weber"}, "true")>
  ```

- EntityLoad(entityname, filtercriteria, sortorder [, options])

filtercriteria を満たすエンティティの配列をロードし、sortorder パラメータで指定されたとおりに並べ替えて返します。

filtercriteria は、プロパティの名前と値で構成されるキーと値のペア (ColdFusion 構造体) です。filtercriteria にキーと値のペアが複数ある場合は、常に AND 演算子が使用されます。

sortorder では、次のシンタックスに従って文字列を指定する必要があります。

"propname1 asc, propname2 desc, ..."

次に sortorder の例を示します。

"firstname asc, lastname desc"

"firstname"

"country, age desc"

例：

州が CA に設定されているアーティストを取得し、City と FirstName を基準に並べ替えるには、次のようにします。

```
<cfset artistsFromCA = EntityLoad('artist', {state="CA"}, "city asc, firstName")>
```

options 引数では、特定の設定オプションを名前と値のペアにして、入力として使用できます。次のオプションを指定すると、エンティティの取得動作を制御できます。

- maxResults: 取得されるオブジェクトの最大数を指定します。
- offset : 結果セットのどのインデックスから取得を開始するかを指定します。
- cacheable: このクエリーの結果がセカンダリキャッシュにキャッシュされるかどうかを指定します。デフォルトは false です。
- cachename: セカンダリキャッシュ内のキャッシュの名前です。
- timeout : クエリーのタイムアウト値 (秒) を指定します。

maxResults と timeout は、ページ割りに使用されます。

例

州が「CA」に設定されているアーティストのうち最初の 5 人をロードし、firstName で並べ替えるには、次のようにします。

```
<cfset artists = EntityLoad("Artist",{state='CA'}, "FirstName", {maxResults=5})>
```

- EntityLoadByExample(sampleentity [, unique])

  sampleentity に一致するオブジェクトの配列をロードして返します。フィルタ条件は、sampleentity で指定された null 以外のプロパティを AND 演算することによって構築されます。たとえば、指定した値に一致するオブジェクトの配列を取得するには、次のようにします。

  ```
  <cfset artist= CreateObject("component", "artist")>
  <cfset artist.setState("CA")>
  <cfset artist.setCity("Berkeley")>
  <cfset artist=EntityLoadByExample(artist)>
  ```

  指定した filtercriteria に一致するレコードが 1 件だけであることがわかっている場合は、unique=true を指定すると、配列ではなく単体のエンティティが返されます。unique=true になっているときに、複数のレコードが返されると、例外が発生します。

- EntityReload(entity)

  このセッションで既にロードされているエンティティのデータをリロードします。このメソッドは、データベースからデータを再取得して、エンティティの内容を再設定します。

オブジェクトの削除

EntityDelete

このメソッドを使用すると、指定したエンティティに対応するレコードがデータベースから削除されます。マッピングで指定されている cascade 属性によっては、そのエンティティに関連付けられているオブジェクトも削除されます。

シンタックス

EntityDelete(entity)

例

たとえば、ArtistID 5 を持つ Artist を削除するには、次のようにします。

```
<cfset artist = EntityLoad('artist', 5, true)>
<cfset EntityDelete(artist)>
```

オブジェクトからクエリーへの変換

EntitytoQuery : このメソッドは、入力として指定したエンティティオブジェクトまたはエンティティオブジェクトの配列を、クエリーオブジェクトに変換します。プロパティの名前がクエリー列の名前として使用されます。継承マッピングの場合は、オプションパラメータ Entity_name を使用して、指定されたエンティティのクエリーを返します。入力配列のオブジェクトはすべて、同じ型である必要があります。関係プロパティは、結果のクエリーには含まれません。

シンタックス

```
EntitytoQuery (orm_object, [entity_name])
EntitytoQuery (orm_object_array, [entity_name])
```

例 1

```
<cfset artists = EntityLoad("Artist")>
<cfset artistQuery = EntityToQuery(artists)>
```

例 2

```
<cfset creditCardPayments = EntityLoad("CreditCardPayment")>
<cfset paymentQuery = EntityToQuery(creditCardPayments, "payment")>
```

エンティティの結合

EntityMerge

現在の ORM セッションにエンティティを追加するには、entitymerge 関数を使用します。指定されたオブジェクトの状態を、同じ識別子を持つ永続オブジェクトにコピーし、その永続オブジェクトを返します。

セッションに現在関連付けられている永続インスタンスがない場合は、インスタンスがロードされます。このインスタンスはセッションに関連付けられていません。このセッションから返されるオブジェクトを使用する必要があります。詳細については、『CFML リファレンス』の EntityMerge を参照してください。

クエリーの使用

ColdFusion では、HQL (Hibernate Query Language) を使用してデータベースに直接クエリーを実行できます。HQL を使い慣れているユーザーであれば、この言語を使用して複雑なクエリーを実行できます。

一般に、HQL は次のような場合に使用します。

- クエリーの対象が特定のオブジェクトではなく、オブジェクト内の一部のフィールドのみである場合。

- オブジェクトをロードせずにオブジェクト内の一部のフィールドのみを取得する場合。
- テーブルの結合を使用する場合。
- min、max、avg、count などの集計関数を使用する場合。
- AND 以外の演算子を使用する必要があるフィルタを指定してエンティティを取得する場合。

HQL メソッドから返される値またはエンティティの配列は、HQL クエリーから返される内容に応じて、1 次元配列または多次元配列になりします。

指定したフィルタ条件に一致するレコードが 1 件だけであることがわかっている場合は、unique=true を指定すると、配列ではなく単体のエンティティが返されます。unique=true を使用すると、クエリーの結果から重複するレコードを除外できます。

注意：HQL で使用する entityname と properties では、大文字と小文字が区別されます。

使用可能な HQL メソッドは、次のとおりです。

ORMExecuteQuery(hql, [params] [,unique])

ORMExecuteQuery(hql, [,unique] [, queryoptions])

ORMExecuteQuery(hql, params [,unique] [,queryOptions])

ORMExecuteQuery (hql, params, boolean unique, Map queryOptions)

ORMExecuteQuery(hql, [,unique] [, queryoptions])

アプリケーションレベルで指定されているデフォルトのデータソースに対して、HQL を実行します。queryoptions を使用すると、取得の動作を制御する次のオプションを指定できます。

- maxResults: 取得されるオブジェクトの最大数を指定します。
- offset : 結果セットのどのインデックスから取得を開始するかを指定します。
- cacheable: このクエリーの結果がセカンダリキャッシュにキャッシュされるかどうかを指定します。デフォルトは false です。
- cachename: セカンダリキャッシュ内のキャッシュの名前です。
- timeout: クエリーのタイムアウト値 (秒単位) を指定します。

maxResults と timeout は、ページ割りに使用されます。

注意：クエリーによってオブジェクトまたはオブジェクトの配列が返された場合、(使用可能な場合は) 永続 CFC の init メソッドが呼び出されます。

例

ART テーブルから作品オブジェクトの配列を取得するには、次のようにします。

```
<cfset art = ORMExecuteQuery("from ART")>
```

価格が 400 ドルを超える作品オブジェクトの配列を取得するには、次のようにします。

```
<cfset art = ORMExecuteQuery("from ART where price > 400")>
```

priceid 100 を持つ作品オブジェクトの配列を取得するには、次のようにします。

```
<cfset artObj = ORMExecuteQuery("from ART where priceid = 100>
```

アーティストのファーストネームを含むオブジェクトの配列を取得するには、次のようにします。

```
<cfset firstNameArray = ORMExecuteQuery("select FirstName from Artist")>
```

作品オブジェクトの数を取得するには、次のようにします。

```
<cfset numberOfArts = ORMExecuteQuery("select count(*) from Art")>
```

artistid 1 を持つオブジェクトの配列を取得するには、次のようにします。

```
<cfset firstName = ORMExecuteQuery("select FirstName from Artist where ARTISTID = 1", true)>
```

クエリー結果の 5 行目から 10 個の artist オブジェクトの配列を取得するには、次のようにします。

```
<cfset artists = ORMExecuteQuery("from Artist", false, {offset=5, maxresults=10, timeout=5})>
```

ORMExecuteQuery(hql, params [,unique] [,queryOptions])

このタイプの ORMExecuteQuery を使用すると、クエリーに名前なしパラメータを渡すことができます。疑問符 '?' をパラメータのプレースホルダーとして使用します。パラメータの値は、配列として params に渡す必要があります。

例 : 名前なしパラメータ

artistid 40 を持つアーティストオブジェクトの配列を取得するには、次のようにします。

```
<cfset artists = ORMExecuteQuery("from ARTIST where artistid > ?", [40])>
```

priceid 1 を持つ作品オブジェクトの配列を取得するには、次のようにします。

```
<cfset artObj = ORMExecuteQuery("from ART where priceid=?", [1], true)>
```

priceid が 40 で価格が 80 ドル未満のオブジェクトの配列を取得するには、次のようにします。

```
<cfset artists = ORMExecuteQuery("from ART where priceid > ? and price < ?", [40, 80])>
```

注意 : 複数のパラメータがある場合は、パラメータの順番に基づいて値が取得されます。たとえば、1 番目のパラメータは 1 番目の値で置き換えられ、2 番目のパラメータは 2 番目の値で置き換えられます。

例 : 名前付きパラメータ

このタイプの ORMExecuteQuery を使用すると、クエリーに名前付きパラメータを渡すことができます。パラメータのプレースホルダーは名前にする必要があり、":age" や ":id" のように、先頭に ":" を付ける必要があります。名前に対する値は、キーと値のペアとして渡す必要があります。

たとえば、住所が USA であり、国籍も USA であるアーティスト全員の詳細を取得するには、次のようなコードを使用します。

```
<cfset USArtists = ORMExecuteQuery("from ARTIST where country=:country and citizenship=:country",
{country='USA'})>
<cfset orderDetail = ORMExecuteQuery("from Orders where OrderID=:orderid and ProductID=:productid",
{orderid=1, productid=901}, true)>
```

注意 : パラメータの大文字と小文字は区別されません。

例 : グループ化

このタイプの ORMExecuteQuery を使用すると、クエリーの集計値またはグループ化された値を取得できます。

たとえば、アーティストの氏名とともに作品の売却状況を取得するには、次のようなクエリーを記述します。

```
<cfset artist = ORMExecuteQuery(
            "SELECT art.Artist.Firstname, art.Artist.Lastname, SUM(art.Price) as Sold FROM  Art as art WHERE
art.IsSold=1 GROUP BY art.Artist.Firstname, art.Artist.Lastname")>
    <cfloop array="#artist#" index="artistItem">
    <cfoutput>
    #artistItem[1]# #artistItem[2]# #artistItem[3]#<br>
    </cfoutput>
</cfloop>
```

注意 : 特定の列名を含む選択クエリーを使用する場合は、getFirstName() や getLastName() などのデータ取得用の組み込みを使用できません。結果は配列オブジェクトとして返され、配列インデックスを使用して値を取得できます。

例 : 並べ替え

このタイプの ORMExecuteQuery を使用すると、order by 節に基づいて並べ替えられたデータをデータソースから取得できます。たとえば、ファーストネームを基準として Artist テーブルのデータを並べ替えるには、次のようなコードを使用します。

```
<cfset artist = ORMExecuteQuery('FROM Artist ORDER BY firstname ASC', false, {maxresults=5} )>
<cfloop array="#artist#" index="artistObj">
    <cfoutput>Name = #artistObj.getFirstName()#
#artistObj.getLastName()#<br></cfoutput>
<br>
</cfloop>
```

例 : 集計関数

このタイプの ORMExecuteQuery では、sum、count、avg などの集計関数を使用してデータを取得できます。

```
<cfset artist = ORMExecuteQuery(
            "SELECT COUNT(*) FROM Art as art WHERE art.Artist.ArtistID=:ArtistID AND art.IsSold=:Sold", {
ArtistID=1, Sold=True }, True )>
<cfoutput>
    #artist#
</cfoutput><br>
```

例 : 式

このタイプの ORMExecuteQuery では、算術演算子、論理演算子、バイナリ比較などの式を使用してデータを取得できます。

たとえば、10000 ドル以上の作品の価格、名前、および説明を取得するには、次のようなコードを使用します。

```
<cfset artArr = ORMExecuteQuery("from Art where price>=10000")>
<cfloop array="#artArr#" index="artObj">
    <cfoutput>
        Art Name = #artObj.getArtName()#<br>
        Description = #artObj.getDescription()#<br>
        Price = #artObj.getPrice()#<br>
    </cfoutput>
<br>
</cfloop>
```

トランザクションと同時性

トランザクションを使用せずに ORM メソッドを呼び出すと、ORM セッションがフラッシュされたときにすべてのデータがデータベースにコミットされます。ORM セッションは、ORMFlush() が呼び出されるとフラッシュされ、自動フラッシュが有効になっている場合はリクエストの終了時にフラッシュされます。

この動作は同時性があまり高くない場合には効果的ですが、ほとんどの現実的なシナリオでは、データベース内のデータの整合性を維持するために、アプリケーションでトランザクションを使用する必要があります。

ColdFusion ORM では、次の 2 つの方法でトランザクションを管理できます。

- **Hibernate トランザクションの使用 :** ユーザーが完全に管理します。ColdFusion は関与しません。アプリケーションで、セッションの消去と終了、およびトランザクションのコミットとロールバックを実行する必要があります。

  トランザクションの詳細については、次の URL を参照してください。

  http://community.jboss.org/wiki/sessionsandtransactions

- **CFTransaction の使用 :** トランザクションは ColdFusion によって管理されます。トランザクションを (別々のデータソースに) 分散することはできないため、トランザクション内で行われた変更は単一の Hibernate セッションにしか影響しないようにアプリケーションで対応する必要があります。つまり、1 つのデータソースしか使用できません。

1 つのトランザクション内でその他のセッション (データソース) のデータを読み込むことはできますが、変更を行うのは 1 つのセッションに限定する必要があります。トランザクションのいかなる時点においても、ダーティセッションが複数存在すると、例外が発生して、トランザクションがロールバックされます。トランザクションが開始する前に、リクエストの既存のセッションはすべて消去されます。前回のセッションが存在する場合は再利用されます。

トランザクションがコミットされると、ダーティセッションは自動的に消去されます。その後でトランザクションがコミットされます。トランザクションがロールバックされると、変更があったセッションは使用できなくなります。これは、変更があったセッションにより、ロールバックされたデータが後になってコミットされてしまう可能性があるためです。そのため、トランザクションがロールバックされると、そのトランザクションに関係するセッションはクリアされます。

このドキュメントでは、トランザクションについては説明しません。トランザクションについて詳しくは、hibernate のマニュアルを参照してください。

トランザクションの内部で ORM メソッドを実行するには、<cftransaction> の中で ORM メソッドを呼び出す必要があります。<cftransaction> で ORM を使用する簡単なコード例を次に示します。

```
<cftransaction>
    <cfset acct1 = EntityLoad("Account", "101")>
    <cfset acct2 = EntityLoad("Account", "102")>
    <cfset acct1.debit(1000)>
    <cfset acct2.credit(1000)>
    <cfset EntitySave(acct1)>
    <cfset EntitySave(acct2)>
</cftransaction>
```

<cftransaction> で明示的にコミットを呼び出していないので、<cftransaction> の終了時にトランザクションが自動的にコミットされます。

<cftransaction> のセマンティクスはすべて、ORM に対しても使用できます。これらのセマンティクスには、セーブポイント、複数ロールバック、複数コミット、ネストされたトランザクションなどがあります。また、1 つの <cftransaction> でクエリーと ORM の両方を使用することもできます。

<cftransaction> が開始されると、既存の ORM セッションがフラッシュされて終了し、新しい ORM セッションが作成されます。<cftransaction> で適切な ColdFusion タグを使用すると、<cftransaction> をコミットまたはロールバックできます。明示的にコミットまたはロールバックされていないトランザクションが終了すると、そのトランザクションが自動的にコミットされた後に、ORM セッションが閉じられます。トランザクション内でエラーが発生し、例外処理が用意されていない場合は、トランザクションがロールバックされます。

<cftransaction> の詳細については、『CFML リファレンス』を参照してください。

注意： <cftransaction> タグ内で ORMFlush() が明示的に呼び出された場合、SQL は実行されますが、トランザクションがコミットされない限りデータはコミットされません。

ColdFusion 9 アップデート 1 における動作の変更

タイミング	ColdFusion 9 の動作	変更後の動作
トランザクションが開始されたとき	既存のセッションは閉じられて、新しいセッションが開始されます。	既存のセッションは消去されて再利用されます。
トランザクションがコミットされたとき	既存のセッションは消去されて閉じられます。	既存のセッションは消去されます。
トランザクションがロールバックされたとき	既存のセッションは消去されないまま閉じられます。	既存のセッションはクリアされます。

オプティミスティックロック

<cftransaction> を使用すると、長時間実行される可能性があるトランザクションのためにデータベースがロックされるので、同時性の高いアプリケーションではスケーラビリティが損なわれる場合があります。また、トランザクションはリクエストの継続時間を超えて実行することはできません。特定のリクエストでロードされたオブジェクトのインスタンスが、別のリクエストまたは別のアプリケーションで更新される場合があります。このようなケースでは、別のリクエストによって行が変更された場合、トランザクションの整合性を維持するために、更新を防止する必要があります。このような動作は、オプティミスティック同時性制御を使用して実現できます。この機能を使用すると、アプリケーションの同時性とスケーラビリティを高めることができます。

オプティミスティック同時性制御では、番号ベースまたはタイムスタンプベースのバージョン管理方式を使用します。番号ベースの方式では、オブジェクトが変更されるたびにバージョン番号がインクリメントされ、タイムスタンプ方式では、タイムスタンプが現在の時刻に設定されます。バージョン番号のインクリメントやタイムスタンプの更新は Hibernate によって管理され、データベースレベルでは管理されないことに注意する必要があります。

- バージョンの使用 : バージョン番号によるオプティミスティック同時性制御を使用するには、fieldtype='version' を含むプロパティを CFC に追加します。

  次に例を示します。

  ```
  /**
  * @persistent
  * @table Users
  */
  component{
      property name="id" fieldtype="id" datatype="int" generator="native";
      property string fname;
      property string lname;
      property name="version" fieldtype="version" datatype="int" ;
  }
  ```

  ユーザーオブジェクトが更新されるたびに、バージョン番号が自動的にインクリメントされます。バージョン番号は SQL の更新ステートメントで使用され、他のリクエストまたは他のアプリケーションによってバージョン番号が変更されていない場合にのみ更新処理が続行されます。

  現在のセッションの外部でバージョン番号が変更されたために更新処理が失敗した場合は、セッションに古いデータが含まれていたことを示すエラーが返されます。

- タイムスタンプの使用 : タイムスタンプによるオプティミスティック同時性制御を使用するには、fieldtype="timestamp" を含むプロパティを CFC に追加します。

  次に例を示します。

  ```
  /**
  * @persistent
  * @table Users
  */
  component{
      property name="id" fieldtype="id" datatype="int" generator="native";
      property string fname;
      property string lname;
      property name="lastModified" fieldtype="timestamp";
  }
  ```

  ユーザーオブジェクトが更新されるたびに、タイムスタンプが自動的に現在の時刻に設定されます。この方式ではユーザーオブジェクトの最終変更時刻もわかるので、バージョン方式より役立つ場合があります。

  現在のセッションの外部でタイムスタンプが変更されたために更新処理が失敗した場合は、セッションに古いデータが含まれていたことを示すエラーが返されます。

オブジェクトにバージョンまたはタイムスタンプのプロパティが含まれていない場合でも、オプティミスティックロックは使用できますが、同じ ORM セッション内で取得されて変更されるオブジェクトに限られます。孤立したオブジェクト (他のリクエストまたは ORM セッションでロードされたオブジェクト) のオプティミスティックロックでは、バージョン番号またはタイムスタンプを使用する必要があります。

バージョン番号またはタイムスタンプを持たないオブジェクトに対してオプティミスティックロックを使用するには、CFC で 'optimistic-lock' 属性を設定する必要があります。この属性では次の値を指定できます。

- all: すべてのプロパティを更新クエリーの where 節に含めることを意味します。
- dirty (デフォルト): 変更されたプロパティだけを更新クエリーの where 節に含めることを意味します。
- version: バージョンフィールドだけを更新クエリーの where 節に含めることを意味します。
- none: どのプロパティも where 節に含めないことを意味します。実質的に、そのコンポーネントに対してオプティミスティック同時性制御を無効にすることを意味します。

  例：

```
/**
* @persistent
* @table Users
* @optimistic-lock all
*/
component{
    property name="id" fieldtype="id" datatype="int" generator="native";
    property string fname;
    property string lname;
}
```

オプティミスティックロック制御は、CFC レベルで定義できるだけでなく、optimisticlock 属性 (true|false: デフォルトは true) を使用してプロパティレベルでも定義できます。

特定のプロパティで optimisticlock=true を指定すると、そのプロパティが更新されるときにオプティミスティックロックを取得できます。この属性を設定すると、プロパティが dirty のときにバージョン番号をインクリメントするかどうかが決定されます。

1 対多または多対多関係の場合は、コレクションの状態が変化すると、対応するエンティティのバージョン番号がインクリメントされます。1 対多関係の場合は、この設定を無効にすることをお勧めします。

パフォーマンスの最適化

遅延ロード

SQL クエリーを最適化すると、データ中心アプリケーションのパフォーマンスが向上します。SQL クエリーの最適化に使用される一般的な手法には次のものがあります。

- データベースへのラウンドトリップを避け、Join を使用した 1 つの SQL クエリーで操作に必要なすべてのデータを取得します。
- 必要なデータだけを取得して、データベースの負荷を軽減します。

SQL クエリーは、ORM エンジンにより生成されて実行されます。したがって、Hibernate には SQL を最適化するためのさまざまな要素が用意されています。取得方式は、最も重要な要素の 1 つです。取得方式では、取得するデータ、データを取得するタイミング、データを取得する方法が定義されています。

オブジェクトとその関連付けをロードする方式は 4 種類あります。

- 即時取得

- 遅延取得
- 一括取得
- バッチ取得

注意：サーバーでメモリ監視機能が有効になっている場合は、オブジェクトのサイズを計算するために、すべてのフィールドへのアクセスが発生します。したがって、遅延取得の対象に設定されているフィールドへのアクセスも発生するため、結果的に、それらのフィールドも即時にロードされることになります。

即時取得

この方式では、指定したエンティティが取得された直後に、個別の SQL クエリーを使用してデータベースから、またはセカンダリキャッシュから、関連するオブジェクトが取得されます。関連するオブジェクトがセカンダリキャッシュにキャッシュされている場合、または Join クエリーより個別のクエリーを使用するほうが効率的である場合を除き、この方式は効率的ではありません。この方式を定義するには、CFC の関係プロパティの定義で lazy="false" および fetch="select" を設定します。

```
<cfproperty name="art" fieldtype="one-to-many" cfc="ART" fkcolumn="ARTISTID" lazy="false" fetch="select">
```

この方式では、artists オブジェクトがロードされた直後に、そのアーティストに関連する art オブジェクトが別の SQL クエリーを使用してロードされます。したがって、この方式は「N+1 選択問題」に対して非常に脆弱です。

遅延取得

この方式では、指定したエンティティに関連するオブジェクトまたはコレクションは必要な場合にのみ取得されます。この方法では、データが必要になるたびに新しいリクエストをデータベースに送信する必要がありますが、取得するデータの量とタイミングを制御できます。この方式はデータベースの負荷を軽減するのに役立ちます。

デフォルトでは、エンティティをロードすると、ColdFusion ORM によってそのエンティティのデータがロードされますが、関係およびマッピングされているコレクションはロードされません。これらは getter メソッドを呼び出さない限りロードされません。したがって、関係とコレクションマッピングは必要な場合にのみロードされます。たとえば、artist オブジェクトをロードしても、そのアーティストに属する作品オブジェクトはロードされず、それらは getarts() を呼び出したときにのみロードされます。

ColdFusion ORM には、関係に対して 3 種類の遅延ロードが用意されています。

- lazy: これは、コレクションマッピング、1 対多関係、および多対多関係に適用されるデフォルトの遅延ロードです。この場合は、コレクションまたは関係のアクセサを呼び出すと、コレクションが完全にロードされます。したがって、特定のアーティストに対して EntityLoad() を呼び出しても、その時点ではそのアーティストに属する作品はロードされません。artist.getarts() を呼び出すと、そのアーティストに属するすべての art オブジェクトがロードされます。これを行うには、CFC の関係プロパティの定義で lazy="true" を設定します。

  例：

  artist.cfc

  ```
  <cfproperty name="art" fieldtype="one-to-many" cfc="ART" fkcolumn="artistId" lazy="true">
  ```

- extra lazy: これは 1 対多関係および多対多関係に適用されます。このタイプの遅延ロードは、lazy タイプよりも高度であり、関係のアクセサを呼び出しても、関連するオブジェクトはロードされません。これらのオブジェクトのプライマリキーだけがロードされ、プロキシオブジェクトが維持されます。ラッパーオブジェクトに対して何らかのメソッドを呼び出すと、そのオブジェクトのデータがデータベースからロードされます。

  たとえば、artist.getarts() を呼び出すと、データベースに対してクエリーが実行され、関連する作品オブジェクトのプライマリキーが取得されて、プロキシとなる作品オブジェクトが作成されます。したがって、すべての作品オブジェクトのデータがメモリにロードされることはありません。特定の作品オブジェクトにアクセスして何らかのメソッドを呼び出すと、データベースに対して別のクエリーが実行されて、その作品オブジェクトがロードされます。これを行うには、CFC の関係プロパティの定義で lazy="extra" を設定します。

例：

artist.cfc

```
<cfproperty name="art" fieldtype="one-to-many" cfc="art" fkcolumn="artistId" lazy="extra" >
```

- proxy: 1 対 1 関係および多対 1 関係に適用されます。指定したオブジェクトがロードされても、それに関連するオブジェクトはデータベースからロードされません。ColdFusion は、関連オブジェクトの代わりとなるプロキシオブジェクトのみを作成し、関連オブジェクトに対して何らかのメソッドが呼び出されると、プロキシオブジェクトのデータがデータベースからロードされて、プロキシオブジェクトにデータが挿入されます。

  たとえば、art-artist テーブルの関係が lazy に設定されている場合、art オブジェクトがロードされても artist オブジェクトはロードされず、art.getartist() を呼び出すと、プロキシオブジェクトのみが取得されます。プロキシオブジェクトに対して何らかのメソッドを呼び出すと、データベースに対するクエリーが実行されて、artist オブジェクトのデータがロードされます。これを行うには、CFC の関係プロパティの定義で lazy="true" を設定します。

  例：

  ART.cfc

  ```
  <cfproperty name="artist" fieldtype="many-to-one" cfc="artist" fkcolumn="artistId" lazy="true">
  ```

  重要：エンティティはリクエスト (Hibernate セッション) の間に 1 回だけロードされ、そのエンティティのコピーの数は常に 1 つだけです。したがって、lazy に設定された作品とアーティストの関係では、アーティストが既にロードされている場合、art.getartist() を呼び出してもプロキシオブジェクトは作成されず、ロード済みのアーティストオブジェクトが返されます。

遅延ロードを無効にするには、CFC の関係プロパティの定義で lazy="false" を設定します。

アプリケーションのパフォーマンスを向上させるには、適切な遅延ロードオプションを選択することが非常に重要です。extra lazy を使用するとデータベースへのトリップ回数は増えますが (データベースへのトリップが発生するたびに多くのリソースが消費されますが)、メモリ内のデータは少なくなります。一方、遅延ロードを使用しないと、巨大なオブジェクトグラフがメモリに格納されることになります。したがって、アプリケーションのニーズに応じてバランスの良い方式を使用する必要があります。

一括取得

この方式では、1 つの SQL Join クエリーを使用して、指定したエンティティと同時に、関連するオブジェクトまたはコレクションが取得されます。この方式を使用するとデータベースへのトリップ回数が減るので、指定したエンティティをロードした直後に必ず関連オブジェクトへアクセスする場合に適した最適化技法です。この方式を定義するには、CFC の関係プロパティの定義で fetch="join" を設定します。

バッチ取得

この方式を使用すると、即時取得または遅延取得の 2 回目の SQL SELECT が Hibernate によって最適化され、1 回のクエリーでオブジェクトまたはコレクションのバッチがロードされます。そのため、現在のリクエストで参照されているプロキシされたオブジェクトまたは初期化されていないコレクションのバッチをロードできます。これは一般に、ネストされたツリーのロードで便利です。これを指定するには、CFC または関係プロパティで batchsize 属性を使用します。

バッチ取得を調整する方法には次の 2 つがあります。

- **CFC レベルでのバッチ取得 :** これを使用するとプロキシではなく実際のオブジェクトのバッチ取得が可能になります。この方法は、1 対 1 関係および多対 1 関係に適用されます。例として、1 回のリクエスト (ORM セッション) で 25 個の art インスタンスがロードされる、作品とアーティストの例について考察します。各作品オブジェクトにはアーティストへの参照があり、関係は lazy に設定されています。したがって、art オブジェクトには、artist のプロキシオブジェクトが含まれます。すべての art オブジェクトに対して繰り返し getartist() を呼び出すと、デフォルトでは、SELECT ステートメントが 25 回実行され、各プロキシオブジェクトに対して実際の artist オブジェクトが 1 つずつ取得されます。これをバッチ処理で実行するには、artist CFC で batchsize 属性を指定します。

```
<cfcomponent table="artist" batchsize="10" ...>
```

最初の art オブジェクトに対して getartist() を呼び出すと、現在のリクエストでプロキシされている 10 個の artist オブジェクトがバッチ取得されます。

したがって、25 個の art オブジェクトがある場合、このタイプのバッチ取得を使用すると、最大で 3 回のクエリー (10 個、10 個、5 個の単位) が Hibernate で実行されます。

- **コレクションレベルでのバッチ取得 :** これを使用すると、初期化されていない値のコレクション、1 対多関係、または多対多関係をバッチ取得できます。例として、25 個の artist オブジェクトがロードされ、各アーティストの作品コレクションが lazy に設定されている artist-art の 1 対多関係について考察します。すべての artist に対して繰り返し getarts() を呼び出すと、デフォルトでは、SELECT ステートメントが合計 25 回実行され (1 つの artist オブジェクトにつき 1 回)、各 artist の art オブジェクトがロードされます。この動作を最適化するには、関係プロパティで "batchsize" を指定してバッチ取得を有効にします。

  例 :

  artist.cfc:

  ```
  <cfproperty name="art" fieldtype="one-to-many" cfc="art" fkcolumn="artistId" lazy="true" batchsize="10">
  ```

  ここで指定している batchsize は、1 人のアーティストに対して 1 回に 10 個の作品をロードするという意味ではないことに注意してください。実際には、10 個の作品コレクション (10 人のアーティストの作品) がまとめてロードされることになります。

  最初のアーティストに対して getarts() を呼び出すと、他の 9 人のアーティストの作品も同時に取得されます。

  batchsize 属性の値は、セッションで予想されるプロキシオブジェクトの数または初期化されていないコレクションの数に基づいて選択する必要があります。

キャッシュ

キャッシュはデータベースアプリケーションを最適化するために幅広く使用されており、これを使用するとデータベースとアプリケーションの間のトラフィック量を効果的な削減できます。

ColdFusion の ORM では、次の 2 つのレベルでキャッシュを使用できます。

- セッションレベル
- セカンダリレベル

セッションレベルのキャッシュ

データベースからオブジェクトがロードされたオブジェクトは、ORM セッションが開かれている間は、常に ORM セッションにキャッシュされます。オブジェクトを取得するためにセッションで EntityLoad が初めて呼び出されると、ORM によりデータベースからデータが取得され、オブジェクトが作成されます。それ以降に、同じセッション内で同じオブジェクトをロードするメソッドが呼び出された場合は、セッションキャッシュからオブジェクトが取得されます。強制的にデータベースからオブジェクトを取得するには、そのオブジェクトに対して EntityReload を呼び出す必要があります。

ORM セッションとそのライフサイクルの詳細については、549 ページの「ORM セッション管理」と 492 ページの「アーキテクチャ」を参照してください。

セカンダリレベルのキャッシュ

ColdFusion には、データベースから取得したデータをセカンダリキャッシュに保存する機能もあります。セカンダリキャッシュの内容はセッションの存続期間が終了した後も残ります。セカンダリキャッシュプロバイダの機能によっては、プロセスの存続期間と同じに設定することも、無期限 (ディスクキャッシュ) に設定することもできます。また、セカンダリキャッシュプロバイダの機能によっては、分散環境でキャッシュを使用することもできます。

セッションレベルのキャッシュとセカンダリレベルのキャッシュの大きな違いは、セッションレベルではオブジェクト全体がキャッシュされるのに対して、セカンダリレベルではデータのみがキャッシュされることです。

セカンダリレベルのキャッシュを利用するには、ColdFusion の ORM を外部キャッシュプロバイダと組み合わせて使用します。一般的なセカンダリキャッシュプロバイダには、Hibernate のプラグインとして使用できる EHCache、JBossCache、OSCache、SwarmCache、Tangosol Coherence Cache などがあります。

ColdFusion では、EHCache がデフォルトのセカンダリキャッシュプロバイダとして使用されます。EHCache はメモリベースのキャッシュとディスクベースのキャッシュを両方サポートする分散型キャッシュソリューションです。EHCache の設定には、設定ファイルを使用します。この設定ファイルでは、複数のキャッシュ領域を定義できます。それぞれのキャッシュ領域には、保存できる要素の数、削除ポリシー、有効期間 (ttl)、アイドル時間などの詳細を指定する、独自の設定があります。

ehcache.xml は <ColdFusion のルートディレクトリ >¥lib¥ にあります。ehcache.xml 内のプロパティの詳細については、次の URL にあるドキュメントを参照してください。

http://ehcache.org/

EHCache 設定ファイル (ehcache.xml) の例を次に示します。

```
<ehcache>
    <diskStore path="java.io.tmpdir"/>
    <defaultCache
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="true"
        diskPersistent="false"
        diskExpiryThreadIntervalSeconds="120"
        memoryStoreEvictionPolicy="LRU"
        />
    <cache name="Artist"
        maxElementsInMemory="20"
        eternal="true"
        overflowToDisk="false"
        />
</ehcache>
```

ColdFusion 9.0.1 での ehcache.xml に対する変更

ehCache.xml には次の設定プロパティが含まれます。

- diskSpoolBufferSizeMB: スプールバッファーに割り当てるディスクストアのサイズです。

  デフォルトのサイズは 30 MB です。各スプールバッファーは、そのキャッシュによってのみ使用されます。

  トレースレベルのロギングをオンにすると、action="put" を使用して作成または更新したキャッシュのバックアップがディスクストアに発生したかどうかが表示されます。

- clearOnFlush: キャッシュが消去されたときにメモリストアがクリアされるかどうかを決定します。デフォルトでは、メモリストアはクリアされます。

- diskExpiryThreadIntervalSeconds: ディスク期限付きスレッドの実行間隔の秒数です。デフォルト値は 120 秒です。

注意： cacheGetProperties および cacheSetProperties 関数を使用すると、これらのプロパティを取得および設定できます。

セカンダリキャッシュの使用

セカンダリキャッシュを使用するには、アプリケーションで次の設定を定義する必要があります。

- ormsettings.secondarycacheenabled

  この設定では、アプリケーションでセカンダリキャッシュを使用するかどうかを定義します。デフォルトでは false に設定されています。

- ormsettings.Cacheprovider

  この設定では、セカンダリキャッシュとして使用するキャッシュプロバイダを定義します。デフォルト値は EHCache です。この設定で指定できる他の値は、JBossCache、OSCache、SwarmCache、および Hashtable です。また、キャッシュプロバイダの完全修飾クラス名を指定することもできます。

- ormsettings.cacheconfig

  この設定では、セカンダリキャッシュプロバイダに必要な設定ファイルを定義します。たとえば、EHCache を使用する場合は、セカンダリキャッシュの設定を含む "EHCache.xml" ファイルを指定する必要があります。この設定では、XML ファイルへのパスを指定します。この設定が定義されていない場合は、キャッシュプロバイダのデフォルト設定が使用されます。

セカンダリキャッシュに保存されたデータは、アプリケーションのすべてのセッションで共有されるので、セカンダリキャッシュを設定した後は、どのオブジェクトをキャッシュする必要があるかを識別することが重要です。一般的に、キャッシュは、次のデータを表現にする CFC に対して有効にすることを推奨します。

- 変更頻度が低いデータ
- そのアプリケーション内でのみ使用され、他のアプリケーションから変更されることがないデータ
- 重要性の低いデータ

また、キャッシュする必要があるオブジェクトのタイプごとに、アクセス方式を決める必要もあります。ORM では、オブジェクトに対して次のキャッシュ方式を使用できます。

- 読み取り専用

  この方式は、読み取り頻度が高く、更新の必要がまったくないデータに適しています。これは最もパフォーマンスの高いキャッシュ方式です。

- 厳格ではない読み取り／書き込み

  この方式は、更新頻度が低いデータに適しています。通常、2 つのトランザクションが同じオブジェクトを同時に更新する可能性はほとんどありません。

- 読み取り / 書き込み

  この方式は、更新を必要とするデータに適している場合があります。この方式は、上の 2 つの方式よりもオーバーヘッドが高くなります。

- トランザクショナル

  この方式では、トランザクショナルキャッシュがサポートされます。この方式は、キャッシュプロバイダでトランザクションが認識される場合にのみ使用できます。

上記の方式が使用できるかどうかは、キャッシュプロバイダに依存します。キャッシュプロバイダによっては、一部のキャッシュ方式がサポートされていない場合もあります。

セカンダリキャッシュには、次の種類のデータを格納できます。

- 永続オブジェクトのデータ
- 永続オブジェクトの関連付け
- クエリーデータ

永続オブジェクトのキャッシュデータ

この場合は、永続オブジェクトのデータがキャッシュされます。関連付けと関連オブジェクトのデータはキャッシュされません。永続 CFC でこのフラグを有効にするには、そのコンポーネントで次の属性を指定します。

- cacheuse : キャッシュ方式を定義します。
- cachename : セカンダリキャッシュプロバイダで使用するキャッシュ領域の名前を定義します。コンポーネント用の領域名を指定しない場合は、コンポーネントのエンティティ名がキャッシュ名と見なされます。設定ファイルで領域が指定されていない場合は、デフォルトの設定を使用して領域が自動的に作成されます。

次に例を示します。

```
<cfcomponent persistent="true" schema="APP" table="Artists" cachename="artist" cacheuse="read-only">
```

永続オブジェクトの関連付けデータのキャッシュ

この場合は、関連オブジェクトのプライマリキーがキャッシュされます。関連付けの一部としてロードされるオブジェクトに対してキャッシュが有効になっていない限り、それらのオブジェクトはキャッシュされません。関連付けをキャッシュするには、関連付けプロパティで次の属性を指定します。

- cacheuse : キャッシュ方式を定義します。
- cachename : セカンダリキャッシュプロバイダで使用するキャッシュ領域の名前を定義します。関連付けプロパティ用の領域名を指定しない場合は、< コンポーネント名 >.< プロパティ名 > がキャッシュ名と見なされます。設定ファイルで領域が指定されていない場合は、デフォルトの設定を使用して領域が自動的に作成されます。

次に例を示します。

```
<cfproperty name="art" fieldtype="one-to-many" cfc="CArt" fkcolumn="ArtID" cachename="ArtistArts"
cacheuse="read-only">
```

クエリーデータのキャッシュ

この場合は、ORMExecuteQuery() メソッドまたは EntityLoad() メソッドで実行されたクエリーの結果がセカンダリキャッシュに格納されます。クエリーデータのキャッシュを有効にするには、cacheable=true と cachename='cachename' という値をメソッドの options 構造体に渡します。cachename を指定しない場合は、デフォルトのクエリーキャッシュにクエリーがキャッシュされます。キャッシュの内容削除を制御できるように、cachename を指定することをお勧めします。

次に例を示します。

```
availableArts = ORMExecuteQuery("from CArt where issold=0", {}, false, {cacheable=true,
cachename="availableArtsQuery"});
```

EHCache を使用したセカンダリキャッシュの例

手順 1 : "Application.cfc" で次の設定を定義します。

```
<cfset this.name="Caching_Example">
<cfset this.datasource="cfartgallery">
<cfset this.ormenabled="true">
<cfset this.ormsettings.secondarycacheEnabled=true>
<cfset this.ormsettings.cacheProvider= "ehcache">
<cfset this.ormsettings.cacheConfig="ehcache.xml">
```

手順 2 : CFC でキャッシュ設定を定義します。

CArtist.cfc

```
<cfcomponent persistent="true" schema="APP" table="Artists" cachename="artist" cacheuse="read-only">
    <cfproperty name="artistid" fieldtype="id"/>
    <cfproperty name="firstname"/>
    <cfproperty name="lastname"/>
    <cfproperty name="state"/>
    <cfproperty name="art" fieldtype="one-to-many" cfc="CArt" fkcolumn="ArtID" cachename="ArtistArts"
cacheuse="read-only">
</cfcomponent>
```

CArt.cfc

```
<cfcomponent persistent="true" schema="APP" table="Art">
    <cfproperty name="artid" generator="identity" fieldtype="id"/>
    <cfproperty name="artname"/>
    <cfproperty name="issold"/>
</cfcomponent>
```

手順 3 :

```
<cfscript>
//This will cache the Artist Component and also the association.  It wouldn't cache the Art objects.
artistObj = EntityLoad("CArtists", 3, true);
//This will cache the query.
availableArts = ORMExecuteQuery("from CArt where issold=0", {}, false, {cacheable=true,
cachename="availableArtsCache"});
</cfscript>
```

セカンダリキャッシュからの内容の削除

ColdFusion には、セカンダリキャッシュから内容を削除できるように、次のメソッドが用意されています。

ORMEvictEntity("<component_name>", [primarykey])

このメソッドは、特定のコンポーネント名に対応する項目をセカンダリキャッシュから削除する場合に使用します。プライマリキーを指定した場合は、そのプライマリキーを持つエンティティのデータが削除されます。単純なプライマリキーの場合は値を使用し、複合的なプライマリキーの場合は構造体を使用する必要があります。

例：

```
<cfset ORMEvictEntity("CArtists")>
```

CArtist エンティティのキャッシュデータをすべて削除します。

```
<cfset ORMEvictEntity("CArtists", 1)>
```

プライマリキーが 1 である CArtists エンティティのキャッシュデータを削除します。

ORMEvictCollection("<component_name>", "<collection_name>", [primarykey])

このメソッドは、特定のコンポーネント名に対応するコレクションデータおよび関連付けデータをセカンダリキャッシュからすべて削除する場合に使用します。プライマリキーを指定した場合は、そのプライマリキーを持つエンティティのコレクションデータまたは関連付けデータが削除されます。

例：

```
<cfset ORMEvictCollection("CArtists", "art")>
```

コンポーネント CArtists に属するコレクション art の関連付けデータまたはコレクションデータをすべて削除します。

```
<cfset ORMEvictCollection("CArtists", "art", 1)>
```

プライマリキー 1 を持つコンポーネント CArtists に属するコレクション art の関連付けデータまたはコレクションデータを削除します。

ORMEvictQueries([cachename])

このメソッドは、デフォルトのクエリーキャッシュからすべてのクエリーのデータを削除する場合に使用します。キャッシュ名を指定した場合は、そのキャッシュ名を持つキャッシュ領域に属するすべてのクエリーのデータが削除されます。
例：

```
<cfset ORMEvictQueries()>
```

デフォルトのクエリーキャッシュからすべてのクエリーのデータを削除します。

```
<cfset ORMEvictQueries("availableArtsCache")>
```

availableArtsCache という名前のキャッシュ領域からすべてのクエリーのデータを削除します。

ColdFusion 9.0.1 でのユーザー定義キャッシュのサポート

ユーザー定義キャッシュは、cacheSetProperties と cacheGetProperties 以外のすべてのキャッシュ関数でサポートされます。

次の例は、ehCache.xml (<ColdFusion のルートディレクトリ >/lib) を編集して、ユーザー定義キャッシュのプロパティを設定しています。

```
<!--- item to put in user-defined cache --->
<cfset currentTime = Now()>
<!--- put item in user-defined cache --->
<cfset timeToLive=createtimespan(0,0,0,30)>
<cfset timeToIdle=createtimespan(0,0,0,30)>
<cfset customCache = "usercache">
<cfset id = "cache1">
<cfset cachePut(id,currentTime,timeToLive,timeToIdle,customCache)>
<!-- list items in the cache --->
List Items in cache:
<cfset cacheIds = cacheGetAllIds(customCache)>
<cfdump var="#cacheIds#"><br>
<!--- print cache data --->
<cfset cachedData = cacheGet(id,customCache)>
<cfoutput>#cachedData#</cfoutput>
<!--- print cache metadata --->
Cache metadata:
<cfset mdata = cacheGetMetadata(id,"object",customCache)>
<cfdump var="#mdata#">
<!--- clear user-defined cache --->
<cfset cacheRemove(ArrayToList(cacheIds),true,customCache)>
```

ORM セッション管理

Hibernate セッションとは、アプリケーションと永続性レイヤーの間の対話を表現するスレッドセーフな一時的オブジェクトです。これは、Hibernate で永続的エンティティに対する CRUD 操作 (Create、Retrieval、Update、Delete) を実行するときに必要になります。ORM を使用する ColdFusion アプリケーションの場合、このセッションは ColdFusion によって自動的に管理されます。Hibernate セッションは第 1 レベルのキャッシュとしても機能し、セッション中に存在する各オブジェクトのコピーは 1 つだけになります。

セッション中に CRUD 操作が実行されても、エンティティのデータはデータベースとすぐには同期されません。つまり、その操作の SQL ステートメントはすぐには発行されず、キューに格納されます。このデータは、セッションがフラッシュされたときに、データベースと同期されます。セッションがフラッシュされると、次の順番で SQL 操作が実行されます。

1. すべてのエンティティ挿入操作 (EntitySave() を使用して対応するオブジェクトが保存された順番と同じ)
2. すべてのエンティティ更新操作
3. すべてのコレクション削除操作
4. すべてのコレクション要素削除、更新、および挿入操作

5 すべてのコレクション挿入操作

6 すべてのエンティティ削除操作 (EntityDelete() を使用して対応するオブジェクトが削除された順番と同じ)

ただし、上記のルールには唯一の例外があり、nativeId を持つオブジェクトは、対応するオブジェクトが保存されるとすぐに挿入されます。

注意：ColdFusion では、アプリケーションスコープで ormenabled が true に設定されている場合に限り、Hibernate セッションが作成されて管理されます。

ColdFusion アプリケーションが起動すると、そのアプリケーションが動作して間、使用可能な Hibernate セッションファクトリが作成されます。このファクトリは、永続オブジェクトのライフサイクルを管理する Hibernate セッションを作成するために使用されます。Hibernate セッションは、最初に CRUD メソッドが呼び出されたときに作成され、リクエストが終了するか、ormclosesession メソッドが呼び出されると閉じられます。ColdFusion と Hibernate の連携の詳細については、492 ページの「アーキテクチャ」を参照してください。

ColdFusion 9 アップデート 1 でサポートされる複数のデータソースを使用する状況では、1 つのリクエストに複数のセッション（各データソースにつき 1 つずつ）が存在します。すべてのエンティティの関数では、対応するセッションが透過的に使用されます。

ColdFusion では、CFML 開発者向けに、Hibernate セッションを直接操作するためのメソッドが何種類か用意されています。ORM セッション関連の関数では、オプションのデータソース引数を使用することもできます。データソースを指定しなかった場合は、ORM に指定されているデフォルトのデータソースが使用されます。これらのメソッドには、次のものがあります。

ORMGetSession()

現在の ORM セッションを返します。これは、アプリケーションで指定されているデータソースに関連付けられた Hibernate セッションです。このメソッドは、API の呼び出しに使用できる基礎の Hibernate セッションを返します。ColdFusion では、これ以外の方法でこの API が公開されることはありません。

ORMCloseSession()

アプリケーションで指定されているデータソースに関連付けられた現在の ORM セッションを閉じます。

ORMFlush()

アプリケーションで指定されたデータソースに関連付けられた現在の ORM セッションをフラッシュします。ORMFlush を呼び出すと、そのリクエストで保留中の CRUD 操作がすべてフラッシュされます。

ORMClearSession()

ORMClearSession を呼び出すと、セッション中にロードまたは作成されたエンティティがすべて削除されます。これにより、第 1 レベルのキャッシュがクリアされ、データベースにまだ保存されていないオブジェクトが削除されます。

ORMGetSessionFactory()

ORMGetSessionFactory は、基礎の Hibernate SessionFactory オブジェクトを返します。

CFC でのイベント処理

ORM では、すべての永続化イベント (Load、Insert、Update、Delete など) のイベントリスナーに対するコールバックを実行できます。これらのイベントは、データの検証や変換、または監査などの汎用的な目的に使用できます。ColdFusion の ORM では、これらのイベントを次の 2 つのレベルで処理できます。

- 永続 CFC で処理する
- イベントハンドラー CFC を使用して処理する

アプリケーションでのイベント処理を有効にするには、ormsettings.eventhandling= "true" という設定を定義します。

デフォルトでは、このフラグは無効になっています。イベントハンドラー CFC が定義されている場合は、このフラグを指定しなくても、フラグは有効であると見なされます。

永続 CFC でのイベント処理

永続 CFC には複数のメソッドが含まれる可能性があり、そのようなメソッドが実際にある場合は、それらのイベントに関するコールバックを CFC に送信できます。その後、CFC 側では、これらのイベントを処理できます。この場合は、システムがエンティティをロード、挿入、更新、または削除したときに発生するエンティティ永続化のイベントが CFC に送信されます。これらのメソッドには、次のものがあります。

- preLoad() : このメソッドは、ロード操作の前、またはデータベースからデータがロードされる前に呼び出されます。
- postLoad() : このメソッドは、ロード操作の完了後に呼び出されます。
- preInsert() : このメソッドは、オブジェクトが挿入される直前に呼び出されます。
- postInsert() : このメソッドは挿入操作の完了後に呼び出されます。
- preUpdate(Struct oldData) : このメソッドは、オブジェクトが更新される直前に呼び出されます。このメソッドには、更新前のエンティティの状態がわかるように、古いデータの構造体が渡されます。
- postUpdate() : このメソッドは、更新操作の完了後に呼び出されます。

  注意： EntityLoad() によってロードされていないオブジェクトに対して EntitySave() メソッドを呼び出した場合、そのオブジェクトは更新されますが、インターセプタの呼び出しは失敗します。この問題は、そのオブジェクトに対して空のマップが作成され、そのマップに以前のデータが関連付けられていないために発生します。
- preDelete() : このメソッドは、オブジェクトが削除される前に呼び出されます。
- postDelete() : このメソッドは、削除操作の完了後に呼び出されます。

イベントハンドラー CFC を使用したイベント処理

アプリケーション全体のイベントハンドラー CFC を定義すると、任意のエンティティが挿入、更新、削除、または取得されたときに、コールバックを処理できます。この CFC は、次に示すように、ORM 設定としてアプリケーションレベルで設定する必要があります。

```
ormsettings.evenHandler="X.Y.EventHandler"
```

イベントハンドラー CFC では、CFIDE.ORM.IEventHandler インターフェイスを実装する必要があります。この CFC は、永続化に関連するすべてのイベントからのコールバックを受け取り、その内容に応じてイベントを処理します。この例では、1 つの CFC がすべての CFC のイベントを処理します。

このインターフェイスには、イベントハンドラー CFC 用の次のメソッドが含まれています。

アプリケーション全体のイベントハンドラ CFC の場合は、他にも引数に加え、コンポーネント名も指定する必要があります。

アプリケーション全体のイベントハンドラのメソッドには、次のものがあります。

- preLoad(entity) : このメソッドは、ロード操作の前、またはデータベースからデータがロードされる前に呼び出されます。
- postLoad(entity) : このメソッドは、ロード操作の完了後に呼び出されます。
- preInsert(entity) : このメソッドは、オブジェクトが挿入される直前に呼び出されます。
- postInsert(entity) : このメソッドは挿入操作の完了後に呼び出されます。
- preUpdate(entity, Struct oldData) : このメソッドは、オブジェクトが更新される直前に呼び出されます。このメソッドには、更新前のエンティティの状態がわかるように、古いデータの構造体が渡されます。

  注意： EntityLoad() によってロードされていないオブジェクトに対して EntitySave() メソッドを呼び出した場合、そのオブジェクトは更新されますが、インターセプタの呼び出しは失敗します。この問題は、そのオブジェクトに対して空のマップが作成され、そのマップに以前のデータが関連付けられていないために発生します。
- postUpdate(entity) : このメソッドは、更新操作の完了後に呼び出されます。
- preDelete(entity) : このメソッドは、オブジェクトが削除される前に呼び出されます。
- postDelete(entity) : このメソッドは、削除操作の完了後に呼び出されます。

注意： 永続 CFC とイベントハンドラー CFC の両方でイベントハンドラーが定義されている場合は、永続 CFC にコールバックが送信された後に、アプリケーション全体のイベントハンドラーが呼び出されます。

データベーススキーマの自動生成

ColdFusion では、ORM がアプリケーションに対して初期化されるときに、テーブルを自動的に作成できます。テーブルを自動生成するには、次のようにします。

Application.cfc の THIS スコープで、ormsettings 構造体の dbCreate プロパティを次のいずれかの値に設定します。

- update : テーブルが存在しない場合は新規のテーブルを作成し、テーブルが存在する場合は既存のテーブルを更新します。
- dropcreate : テーブルが存在する場合は既存のテーブルを削除してから新規のテーブルを作成します。

次に例を示します。

```
<cfset this.ormsettings.dbCreate="update">
```

cfcomponent タグおよび cfproperty タグで定義される特定の属性 (DDL 専用の属性) を使用して、自動生成されるテーブルや列のさまざまな属性を定義できます。DDL 専用の属性は、DDL の生成のみに使用されます。これらの属性の詳細については、505 ページの「列」で DDL 専用の属性の表を参照してください。

例

Application.cfc

```
<cfset this.name = "AG"/>
<cfset this.ormenabled=true/>
<cfset this.datasource = "ORM_DDL">
<cfset this.ormsettings.dbCreate="dropcreate">
<cfset this.ormsettings.sqlscript="mysqlscript.sql">
```

Artists.cfc

```
<cfcomponent persistent="true" table="Artists">
      <cfproperty name="artistid" fieldtype="id" ormtype="integer" length=10>
      <cfproperty name="firstname" ormtype="string" length="20" notnull="true">
      <cfproperty name="lastname" ormtype="string" length="20" notnull="true">
      <cfproperty name="address"  ormtype="string" length="50">
      <cfproperty name="city" ormtype="string" length="20">
      <cfproperty name="state" ormtype="string" length="2">
      <cfproperty name="postalcode" ormtype="string" length="10">
      <cfproperty name="email" ormtype="string" length="50" unique="true">
      <cfproperty name="phone" ormtype="string" length="20">
      <cfproperty name="fax" ormtype="string" length="12">
      <cfproperty name="thepassword" ormtype="string" length="20">
</cfcomponent>
```

art.cfc

```
<cfcomponent persistent="true" table="Art">
      <cfproperty name="artid" generator="identity" fieldtype="id">
      <cfproperty name="artname" ormtype="string" length="50">
      <cfproperty name="price" ormtype="double">
      <cfproperty name="largeimage" ormtype="string" length="30">
      <cfproperty name="mediaid" ormtype="integer" length="10">
      <cfproperty name="issold" ormtype="boolean" dbdefault=1>
      <cfproperty name="artist" fkcolumn="artistid" fieldtype="many-to-one" cfc="CArtists">
</cfcomponent>
```

ネーミングストラテジ

通常、データベース中心アプリケーションを作成する際には、何らかのデータベース標準とネーミング規則に従います。ColdFusion ORM では、「命名方式」を使用することで、この規則をアプリケーションに対して 1 箇所で定義できます。

ネーミングストラテジを使用すると、アプリケーションのコードを全体的に変更する必要がなくなるという利点があります。ネーミングストラテジでは、CFC とそのプロパティに関連付けれたテーブルおよび列の名前をどのように決定するかを指定します。

ネーミングストラテジで、テーブルまたは列の「論理名」を渡すと、使用する必要がある実際のテーブル名または列名が返されます。

- 論理テーブル名 : CFC に対して指定されているテーブル名です。指定されていない場合は、エンティティ名が論理テーブル名として使用されます。エンティティ名も指定されていない場合は、修飾なしの CFC 名 (たとえば、a.b.c.Person の場合は Person) が論理テーブル名として使用されます。
- 論理列名 : CFC プロパティに対して指定されている列名です。指定されていない場合は、プロパティ名が論理列名として使用されます。

ネーミングストラテジをアプリケーションに適用するには、Application.cfc で次のように設定します。

```
<cfset this.ormsettings.namingstrategy="strategy">
```

ストラテジの値は次のいずれかになります。

- default : このストラテジでは、論理テーブル名または論理列名がそのまま使用されます。ColdFusion ORM では、この値がデフォルトのストラテジとして使用されます。
- smart : このストラテジでは、論理テーブル名または論理列名が大文字に変更されます。また、論理テーブル名または論理列名がキャメルケースの場合、このストラテジでは、キャメルケースで表記された名前が分割され、アンダースコアで個々の単語が区切られます。たとえば、CFC の名前が「OrderProduct」である場合、テーブル名は「ORDER_PRODUCT」に変更されます。
- your own cfc : 独自の実装を作成することで、ネーミングストラテジを完全に制御できます。この場合は、ネーミングストラテジの値として、CFC の完全修飾名を指定する必要があります。この CFC では、cfide.orm.INamingStrategy インターフェイスを実装する必要があります。

cfide.orm.INamingStrategy インターフェイスは次のような内容です。

```
 /**
* Strategy to specify the table name for a CFC and column name for a property in the cfc.
* This can be used to specify the application specific table and column naming convention.
* This rule will be applied even if the user has specified the table/column name in the mapping so that
* the name can be changed for any application at one place without changing the names in all the code.
*/
interface
{
/**
* Defines the table name to be used for a specified table name. The specified table name is either
* the table name specified in the mapping or chosen using the entity name.
*/
public string function getTableName(string tableName);

/**
* Defines the column name to be used for a specified column name. The specified column name is either
* the column name specified in the mapping or chosen using the proeprty name.
*/
public string function getColumnName(string columnName);
}
```

このインターフェイスをアプリケーションで指定するには、次のように設定します。

```
this.ormsettings.namingstrategy="com.adobe.UCaseStrategy"
```

注意：ネーミングストラテジは、CFC または cfproperty に関連付けられていないものであっても、リンクテーブルや fkcolumn などのマッピングで使用するすべてのテーブル名および列名に適用されます。

スクリプトを使用したデータベースへのデータ追加

DDL を使用してテーブルを作成した後に、SQL スクリプトを使用してデータベースにデータを追加することもできます。これを行うには、実行する SQL スクリプトファイルのパス (ファイルの絶対パスまたはアプリケーションからの相対パス) を指定します。このスクリプトは、dbcreate が dropcreate に設定されている場合にのみ実行されます。この SQL スクリプトファイルを使用して、アプリケーションへのアクセス前に、テーブルの値を設定できます。各 SQL ステートメントの先頭に改行があり、末尾にセミコロンがあることを確認してください。

例
Mysqlscript.sql

```
insert into Artists(artistid, firstname, lastname, address, city, state, postalcode, email, phone, fax,
thepassword)
values(1, 'Aiden', 'Donolan', '352 Corporate Ave.', 'Denver', 'CO', '80206-4526',
'aiden.donolan@donolan.com', '555-751-8464', '555-751-8463', 'peapod');
insert into Artists(artistid, firstname, lastname, address, city, state, postalcode, email, phone, fax,
thepassword)
values(2, 'Austin', 'Weber', '25463 Main Street, Suite C', 'Berkeley', 'CA', '94707-4513',
'austin@life.com', '555-513-4318', '510-513-4888', 'nopolyes');
insert into Art(artname, price, largeimage, mediaid, issold, artistid)
values('Michael', 13900, 'aiden02.jpg', 1, 0, 1);
insert into Art(artname, price, largeimage, mediaid, issold, artistid)
values('Space', 9800, 'elecia01.jpg', 2, 1, 2);
```

ORM における複数のデータソースのサポート

注意：この機能は、ColdFusion 9 アップデート 1 をインストールしてある場合にのみ適用されます。

概要

ColdFusion アプリケーションでは、ORM に複数のデータソースを使用できます。複数のデータソース設定は、アプリケーションに複数のモジュールが含まれていて、これらのモジュールが相互に通信する場合に便利です。

Hibernate では、本来、Hibernate 設定に対して単一のデータソースをサポートしています。複数のデータソースをサポートするには、ColdFusion で、複数の Hibernate 設定と SessionFactory オブジェクトを作成して管理します。これらは、アプリケーションの各データソースにつき 1 つずつ用意します。

使用例

次の 3 つのモジュールが含まれるアプリケーションを考えます。

- HR
- Finance
- Sales

これらすべてのモジュールには、それぞれ専用のデータベースがあるとします。そのため、データソースは別々のものになります。ただし、アプリケーションレベルでは、3 つのモジュールすべてが互いに通信可能である必要があります。単一のデータソースでは、ORM を使用して完全なアプリケーションを構築することができません。3 つのアプリケーションを別々に構築するのはお勧めできません。アプリケーション間の通信手段が、Web サービスのみになってしまうからです。

ORM に複数のデータソース設定を使用すると、3 つのモジュールすべてを ORM に構築できます。これらのモジュールは 1 つの同じアプリケーションの一部となり、モジュールは互いに通信することができます。

アプリケーションで複数のデータソースを使用するための設定

永続 CFC と、対応するデータソースを示す datasource 属性を設定します。CFC の datasource 属性を指定するには、cfcomponent タグを使用するか、CFC 定義のコンポーネントに注釈を指定します。データソースを指定しなかった場合、その CFC にはデフォルトのデータソースが使用されます。

Hibernate 設定では単一のデータソースを使用するため、(ORM の関係を使用する) すべての関連 CFC に同じデータソースを指定する必要があります。

例

Art.cfc

```
<cfcomponent persistent="true" datasource="artgallery" table="Art">
  ...
</cfcomponent>
```

Author.cfc

```
<cfcomponent persistent="true" datasource="bookclub" table="author">
  …
</cfcomponent>
```

ORM の設定

データソース固有の ORM 設定は、次のとおりです。Application.cfc で、これらの設定に文字列または構造体の値を指定できます。

- schema
- catalog
- dialect

- dbcreate
- sqlscript

複数のデータソースの場合、構造体に、データソース名をキーで、対応する設定を値で指定できます。文字列の値を指定した場合は、ORM のデフォルトのデータソースに適用されます。

例 1

```
<cfset this.ormsettings.dbcreate={artgallery="dropcreate", bookclub="none"}>
```

例 2

```
<cfset this.ormsettings.dbcreate="dropcreate">
```

ORM に複数のデータソースを使用している場合、これらの設定は、デフォルトの ORM データソースに適用されます。

Hibernate マッピングファイルを使用したマッピング

複数のデータソースを使用する場合は、CFC にデータソース情報を記述する必要があります (.hbmxml ファイルには必要ありません)。

また、1 つの .hbmxml で使用されるすべての CFC に、同じデータソースを指定する必要があります。

例

次の例は、異なる 2 つのデータソースを使用する異なる 2 つのエンティティを示しています。この例では、art.cfc と artist.cfc は関連付けられているため、同じデータソースを使用しています。

art.cfc

```
<cfcomponent persistent="true" table="art" datasource="cfartgallery">
<cfproperty name="ArtID" fieldtype="id" generator="native">
<cfproperty name="ArtName">
<cfproperty name="IsSold">
</cfcomponent>
```

artists.cfc

```
<cfcomponent persistent="true" table="artists" datasource="cfartgallery">
<cfproperty name="ArtistID" fieldtype="id">
<cfproperty name="FirstName">
<cfproperty name="LastName">
<cfproperty name="art" fieldtype="one-to-many" cfc="art" fkcolumn="ArtistID">
</cfcomponent>
```

authors.cfc

```
<cfcomponent persistent="true" table=authors datasource="cfbookclub">
<cfproperty name="AuthorID" fieldtype="id">
<cfproperty name="LastName">
<cfproperty name="FirstName">
</cfcomponent>
```

index.cfm

```
<cfoutput>Original Data<br></cfoutput>
<cfset artistObj = EntityLoad("artists", 1, true)>
<cfoutput>#artistObj.getArtistID()# | #artistObj.getFirstName()# |
#artistObj.getLastName()#<br></cfoutput>
<cfset artObj = artistObj.getart()>
<cfoutput>#artObj[1].getartname()# <br></cfoutput>
<cfset authorObj = EntityLoad("authors", 1, true)>
<cfoutput>#authorObj.getFirstName()#</cfoutput>
<cfoutput>#authorObj.getLastName()#</cfoutput>
```

ORM 関数の機能強化

複数のデータソースのサポートにより、影響を受けた ORM 関数は、次のとおりです。

ORMGetSession

説明

リクエストのデータソースに関連付けられている Hibernate セッションを返します。このデータソースが ORM に設定されていない場合は、例外が発生します。データソースが指定されていない場合は、デフォルトのデータソースの Hibernate セッションが返されます。

このセッションオブジェクトを使用して、通常は ColdFusion で公開されていない API を呼び出します。

セッション API については、次の資料を参照してください。

http://docs.jboss.org/hibernate/core/3.3/api/org/hibernate/Session.html

関数のシンタックス

ormgetsession([datasource])

ORMCloseSession

説明

リクエストのデータソースに関連付けられている Hibernate セッションを閉じます。データソースを指定しなかった場合は、デフォルトのデータソースに関連付けられている Hibernate セッションが閉じられます。

関数のシンタックス

ormclosesession([datasource])

ORMCloseAllSessions

説明

リクエストのすべての Hibernate セッションを閉じます。

関数のシンタックス

ormcloseallsessions()

履歴

ColdFusion 9 アップデート 1: この関数が追加されました。

ORMFlush

説明

リクエストのデータソースに関連付けられている Hibernate セッションを消去します。ORMFlush を呼び出すと、そのリクエストで保留中の CRUD 操作がすべて消去されます。現在の ORM セッションでオブジェクトに加えられた変更はすべて、データベースに保存されます。

データソースを指定しなかった場合は、デフォルトのデータソースに関連付けられている Hibernate セッションが消去されます。

関数のシンタックス
ormflush([datasource])

ORMFlushall

説明
リクエストの現在の Hibernate セッションをすべて消去します。

関数のシンタックス
ormflushall()

履歴
ColdFusion 9 アップデート 1: この関数が追加されました。

ORMClearSession

説明
指定したデータソースに関連付けられている Hibernate セッションをクリアします。

この関数は、第 1 レベルのキャッシュをクリアし、データベースにまだ保存されていないオブジェクトを削除します。

データソースを指定しなかった場合は、デフォルトのデータソースに関連付けられている Hibernate セッションがクリアされます。

関数のシンタックス
Ormclearsession([datasource])

ORMGetSessionFactory

説明
データソースに関連付けられている Hibernate セッションファクトリオブジェクトを返します。このデータソースが ORM に設定されていない場合は、エラーが発生します。データソースを指定しなかった場合は、デフォルトのデータソースに関連付けられている Hibernate セッションファクトリオブジェクトが返されます。

Session API の詳細については、次の URL を参照してください。

http://docs.jboss.org/hibernate/core/3.3/api/org/hibernate/SessionFactory.html

関数のシンタックス
Ormgetsessionfactory([datasource])

ORMEvictQueries

説明
このメソッドは、指定したデータソースのデフォルトのクエリーキャッシュからすべてのクエリーのデータを削除する場合に使用します。キャッシュ名を指定した場合は、そのキャッシュ名を持つキャッシュ領域に属するすべてのクエリーのデータが削除されます。

データソースを指定しなかった場合は、デフォルトのデータソースのデフォルトのクエリーキャッシュが削除されます。

シンタックス
ORMEvictQueries([cachename]) ORMEvictQueries([cachename], datasource)

パラメータ	説明
cachename	削除するキャッシュ領域の名前です。
datasource	削除するキャッシュに関連付けられているデータソースの名前です。キャッシュを指定しなかった場合は、デフォルトのクエリーキャッシュが削除されます。

ORMExecuteQuery

説明
Hibernate Query Language (HQL) クエリーを実行します。

デフォルトでは、この関数は、ORM のデフォルトのデータソースに対して実行されます。別のデータソースに対してこの関数を使用する場合は、queryoptions の中でデータソースのキーと値のペアを指定します。

シンタックス
ORMExecuteQuery(hql, [params] [,unique]) ORMExecuteQuery(hql, [,unique] [, queryoptions]) ORMExecuteQuery(hql, params [,unique] [,queryOptions])

パラメータ

パラメータ	説明
hql	実行する HQL クエリーです。
params	エンティティのオブジェクトパラメータです。
unique	オブジェクトパラメータが一意であるかどうかを指定します。
queryoptions	クエリーのオプションのキーと値のペアです。

例
```
<cfset artistArr = ORMExecuteQuery("from Artists where artistid=1", true, {datasource="cfartgallery"})>
<cfset countArray = ORMExecuteQuery("select count(*) from Authors", [], false, {datasource="cfbookclub"})>
```

ColdFusion ORM 検索

ColdFusion 10 の機能拡張では、ColdFusion ORM のインデックス作成および検索機能が提供されます。ColdFusion ORM を使用するアプリケーションを開発するときは、検索機能により全文検索が容易になります。クエリテキストに基づいて、検索条件に一致するすべての永続的エンティティを読み込むことができます。

全文検索は 2 つの手順のプロセスであり、永続的エンティティのインデックス作成と、インデックスを作成された情報に基づく検索で構成されます。

永続的エンティティのインデックス作成

インデックス作成では、永続的エンティティのインデックス作成可能なデータをインデックスファイルに格納します。検索はインデックスを作成されたデータで実行されます。コンポーネントで指定する設定により、インデックス作成可能なデータが決まります。

ORM 検索のインデックスディレクトリの指定

サーバーレベルまたはアプリケーションレベルで、インデックスディレクトリ（アプリケーションのインデックス作成可能なデータのすべての永続的エンティティが保存されるディレクトリ）を指定できます。

サーバーレベル

1 ColdFusion Administrator で、サーバーの設定／設定に移動します。

2 設定ページの「ORM 検索のインデックスファイルを格納するための絶対パスを指定してください」で、ディレクトリの詳細を指定します。

注意：ディレクトリは、指定したパスに、アプリケーションごとにアプリケーション名で作成されます。

アプリケーションレベル

絶対パスまたは相対パスで現在の Application.cfc にインデックスディレクトリのパスを指定します。相対パスを指定した場合は、現在のフォルダーを基準に解決されます。

注意：パスを指定しない場合は、インデックスファイルはデフォルトで **cfroot**/ormindex に格納されます。

アプリケーションレベルでの検索設定

次の表で示すプロパティを使用して、Application.cfc で ColdFusion ORM 検索の設定を指定します。

プロパティ	必須／オプション	デフォルト	説明
searchenabled	ORM 検索を使用する場合は必須	false	yes の場合、ORM 検索が有効になります。
search.autoIndex	オプション	false	yes の場合、自動インデックス作成が有効になります。
search.indexDir	オプション		指定すると、すべてのエンティティのインデックスが保存されるファイルシステムにエンティティ名でフォルダーが作成されます。ColdFusion Administrator で設定するディレクトリは、このために使用されます。 このフォルダーにすべてのインデックス情報が格納されます。 指定しないと、アプリケーションルートディレクトリにフォルダーが作成されます。
search.language	オプション	english	インデックスと検索に使用する言語を指定します。

例

```
component
{
    this.name = "ORM_Search";
    this.ormEnabled = "true";
    this.ormSettings.datasource = "ORM_Dummy";
    this.ormSettings.dbcreate = "dropcreate";
    this.ormsettings.searchenabled = "true";
    this.ormSettings.search.autoindex = "true";
    this.ormSettings.search.indexDir = "C:/ormindex";
    this.ormSettings.search.language = "English";
}
```

コンポーネントレベルでの検索設定

cfcomponent タグで次のことを指定します。この設定は Application.cfc の値より優先されます。

属性	必須 / オプション	デフォルト	説明
indexable	オプション	false	true の場合、コンポーネントのインデックス作成が有効になります。
indexLanguage	オプション	english	インデックスと検索に使用する言語を指定します。 ここ設定する値は、Application.cfc で定義されている値より優先されます。
autoIndex	オプション	true	false の場合、CFC の自動インデックス作成は行われません。つまり、インデックス作成はオフラインモードでのみ行われます。

プロパティレベルでの検索設定

cfproperty タグで次のことを指定します。この設定は Application.cfc および cfcomponent の値より優先されます。

属性	必須 / オプション	デフォルト	説明
indexable	オプション	false	true の場合、列がインデックス作成の対象にマークされます。 PK および複合キー（非 PK プロパティのいずれかが indexed に設定されている場合はインデックスが作成されます）を除き、デフォルト値は false です。
indexTokenized	オプション	true	true の場合、フィールドのテキストはインデックス作成のためにサブキーに分割されます。 indexed が true に設定されている場合にのみ適用されます。
indexFieldName	オプション	属性 name の値	インデックス作成および検索実行の間に検索クエリで使用されるフィールド名を指定します。
indexBoost	オプション		検索結果の優先順位付けに使用します。数値です。 この列の結果は、クエリ結果で他より上に表示されます。値が大きいほど、優先順位が高くなります。
indexStore	オプション	false	値は、true、false および compressed です。 • true：トークン化せずに、元の形式で値を格納します。 • false：値を格納しません。 • compressed：Lucene の実装に基づき、元の値を圧縮形式で格納します。
indexLanguage	オプション	english	インデックスと検索に使用する言語を指定します。 この値は、cfcomponent および Application.cfc で定義されている値より優先されます。

例

Empoyee.cfc

```
component persistent="true" indexable="true" indexlanguage="English" table="Employee"
{
    property name="id" fieldtype="id" ormtype="int" generator="native";
    property name="firstname" indexable="true" indexstore="yes" indexboost="5.0";
    property name="lastname" indexable="true" indextokenize="true" indexfieldname="surname";
    property name="description" indexlanguage="Greek";
    property name="dept" fieldtype="many-to-one" cfc="dept" indexable="true" indexfieldname="department"
notnull="false";
}
```

インデックス作成モード

ColdFusion ORM を使用すると、自動的に、またはオフラインモードでインデックス作成を実行できます。

自動インデックス作成

データベースのエンティティが追加、変更または削除されるたびに、インデックスの作成が実行されます。

自動インデックス作成を有効にするには、Application.cfc で ormsettings.searchenabled を true に設定します。

ORM エンティティが永続化されるたびに、CFC での設定に基づいて、インデックス作成が自動的に行われます。詳しくは、562 ページの「ORM 検索設定の指定」を参照してください。

オフラインでのインデックス作成

インデックスの作成は、ColdFusion の機能を使用して手動で行います。このモードでは、インデックス作成はバッチで実行されます。

次のような状況でのオフラインインデックス作成の実行をお勧めします。

- データベース内の既存データのすべて、または一部のインデックスを作成する場合。
- CPU の負荷を最小にするためにリクエストでのインデックス作成（デフォルトの動作）を回避し、後でバッチ操作としてインデックス作成を実行する場合。

関数 ORMIndex を使用すると、オフラインでのインデックス作成を実行できます。

インデックス作成済み情報に基づく検索

ColdFusion で提供されている関数を使用して、検索を実行できます。次のどちらかで検索できます。

- Lucene クエリ検索
- ColdFusion ORM が Lucene クエリを生成する全文検索

ORM 検索設定の指定

アプリケーションレベル、コンポーネントレベルまたはプロパティレベルで、ORM 検索設定を指定する必要があります。

検索の実行

ORM 検索関数を使用し、オンラインモードおよびオフラインモードで次の検索を実行できます。

- エンティティ
- エンティティ内のインデックス作成可能なフィールド
- エンティティ全体の共通フィールド
- 1 つのエンティティの複数フィールド
- 複数のエンティティの複数フィールド

その他の機能強化

ロギング

ColdFusion Administrator でリクエストのデバッグを有効にすると、HQL クエリのロギングに役に立ちます。これを行うには、次のようにします。

1. ColdFusion Administrator で、デバッグとロギング／デバッグ出力の設定に移動します。
2. 「リクエストのデバッグ出力の有効化」オプションを選択します。

第 9 章：ColdFusion と HTML 5

ColdFusion 10 は HTML 5 の強力な機能のセットを備えています。ColdFusion 10 の HTML 5 機能は、Web での対話と通信の将来的な標準であるため、ユーザーの Web 環境はこれまでより充実していながら簡単なものになります。

ColdFusion WebSocket の使用

WebSocket とは

WebSocket は、HTML 5 準拠ブラウザーの JavaScript インターフェイスによって公開される TCP プロトコルを使用してリモートホストと双方向の通信を行うためのプロトコルです。WebSocket により、両方向同時にホスト間の通信が容易になります。

WebSocket と従来の通信モデルの相違点

WebSocket は HTML 5 のサーバープッシュテクノロジーが基になっています。通信要求は、パブリッシュ／サブスクライブモデルに従うパブリッシャーまたは中央サーバーによって開始されます。

これは、HTTP が依存する、リクエストのタイムアウトやブラウザーの更新といった欠点のあるリクエスト／レスポンスモデルとは異なるものです。

WebSocket の場合、クライアントは様々な情報チャネルをサブスクライブします。チャネルに新しいコンテンツがあると、クライアントにプッシュされます。この方法には次のような長所があります。

- 同時双方向通信
- ブラウザーのリロードが少ない
- リアルタイムデータを処理するアプリケーションの作成が容易

ColdFusion と WebSocket

ColdFusion 10 では、WebSocket プロトコル用のメッセージングレイヤーを提供することで WebSocket を実装しています。これは、CFML と JavaScript を使用して制御できます。

メッセージングレイヤーは、WebSocket を信頼性と拡張性のあるものにする機能を備えています。WebSocket を使用すると、株式、チャート作成、オンラインゲーム、ソーシャルネットワーク、様々な目的のダッシュボード、監視などを目的とするリアルタイムアプリケーションを開発できます。

ブラウザーとフォールバック

ColdFusion は、ブラウザーがサポートする HTML5 のネイティブな WebSocket を利用して、サーバーで制御されたクライアント間通信を提供します。ブラウザーが現時点で HTML5 またはネイティブ WebSocket をサポートしていない場合、ColdFusion はフォールバックオプションとして Flash を使用します。Flash へのフォールバックはスムーズかつ自動的に行われます。

585 ページの「ブラウザーの WebSocket サポート」も参照してください。

次の表に、ColdFusion WebSocket のフォールバックプランを示します。

状況	生じる結果
ネイティブな WebSocket サポートが使用可能である	WebSocket 接続が確立されます。
ネイティブな WebSocket は使用できないが、Flash はインストールされている	自動的に Flash にフォールバックされます。
ネイティブな WebSocket がサポートされておらず、Flash もインストールされていない	接続が成功しなかったことを示すメッセージが送信されます。 続行するには、準拠するブラウザーに移行するか、Flash をインストールしてください。

WebSocket の通信モデル

ColdFusion の WebSocket では、次の 2 つの通信モデルが提供されます。

- **ブロードキャスト：**ホストに複数のクライアントとサーバーが含まれるサブスクライバー／パブリッシャーモデルに従います。パブリッシュは次のどちらかの方法で可能です。
  - クライアントサイドから（サーバーを介して）
  - クライアントの関与なしにサーバーから開始
- **ポイントツーポイント：**特定のクライアントとサーバーの間の双方向通信です。このモデルでは、その他のホストや複数のクライアントは関与しません。

WebSocket を使用したメッセージのブロードキャスト

ブロードキャストのセットアップは次の手順で行います。

ステップ	操作	説明
1	Application.cfc で通信チャネルを定義します。	クライアントは、あらかじめ定義された一連のチャネルに対してサブスクライブしたりパブリッシュすることができます。 チャネルをリスンするには、最初にアプリケーションレベルでそのチャネルを登録します。
2	CFM テンプレートの cfwebsocket タグを使用して、WebSocket オブジェクトを作成します。	必要なすべてのプロパティを持つ WebSocket 接続が自動的に作成され、各種の JavaScript メソッドを呼び出し可能な JavaScript オブジェクトにラップされます。 このタグで、サーバーの IP アドレスとポートを内部的に識別します。
3	WebSocket オブジェクトによって提供される JavaScript 関数を使用して、ビジネスロジックを実装します。	実行時に、サポートされている JavaScript 関数（例えば、subscribe や publish など）を使用して、業務の目的を実現できます。

通信チャネルの定義

通信チャネルを定義するには、Application.cfc で次の設定を指定します。

```
component
{
this.name="testapp";
this.wschannels=[{name = channelName,cfclistener= channel_listener_CFC}];
}
```

説明

- チャネル名を指定します。

  注意：任意の数のサブチャネルを使用できますが、サブチャネルは指定しません。それらは動的に作成されます。

- 必要な場合は、チャネルを作成するコンポーネントである ColdFusion チャネルリスナー CFC を指定します。指定されていない場合は、wwwroot¥CFIDE¥websocket ディレクトリにあるデフォルトの ChannelListener.cfc を呼び出します。

  詳しくは、571 ページの「チャネルリスナー関数の使用」を参照してください。

cfwebsocket を使用した WebSocket オブジェクトの作成

新しい ColdFusion タグの cfwebsocket を使用して、CFM テンプレート内に WebSocket オブジェクトを作成できます。このタグは、クライアントサイドの WebSocket JavaScript オブジェクトに対する参照を作成します。

シンタックス

```
<cfwebsocket
    name="websocketName"
    onMessage="JavaScript function name"
    onOpen="JavaScript function name"
    onClose="JavaScript function name"
    onError="JavaScript function name"
    useCfAuth=true|false
    subscribeTo="channel_list">
```

属性

属性	必須 / オプション	説明
name	必須	WebSocket オブジェクトの名前です。 これは、WebSocket JavaScript 関数を呼び出すために使用される JavaScript オブジェクトへの参照です。
onMessage	必須	WebSocket がサーバーからのメッセージを受信すると呼び出される JavaScript 関数です。この関数には、1 つの引数が必要です。
onOpen	オプション	WebSocket が接続を確立すると呼び出される JavaScript 関数です。 この関数に、引数は必要ありません。
onClose	オプション	WebSocket が接続を閉じると呼び出される JavaScript 関数です。 この関数に、引数は必要ありません。
onError	オプション	WebSocket 接続でのアクションの実行時にエラーが発生した場合に呼び出される JavaScript 関数です。 この関数には、1 つの引数が必要です。
usecfAuth	オプション	true に設定した場合（デフォルト）、WebSocket 接続のための認証は必要はありません（ユーザーがアプリケーションに既にログインしている場合）。これはデフォルト値です。 false の場合は、認証を行うために authenticate の javaScript 関数を使用する必要があります。
subscribeTo	オプション	サブスクライブするチャネルおよびサブチャネルのカンマ区切りリストです。Application.cfc で設定されている任意のチャネルまたはすべてのチャネルを指定できます。

使用方法

name 属性を指定して（例えば、次の例では mycfwebsocketobject と指定）、JavaScript オブジェクトを作成します。

JavaScript オブジェクトを使用して、WebSocket 通信に使用する各種の JavaScript 関数を呼び出すことができます。

例

次に示す例では、

- ユーザーは stocks チャネルを自動的にサブスクライブします。

- チャネル名は Application.cfc で指定してあります。
- デフォルトのチャネルリスナーを使用します（カスタムチャネルリスナーを指定していないため）。
- Index.cfm では、ユーザーが（subscribeTo 属性を使用して）自動的にサブスクライブできるチャネルを指定し、メッセージハンドラーも定義します。

Application.cfc

```
component
{
       this.name="websocketsampleapp1";
       this.wschannels=[{name="stocks"}];
}
```

Index.cfm

```
<script type="text/javascript">
        /* mymessangehandler function recieves all messages that comes via WebSocket. This function
requires one argument.
        */

       function mymessagehandler(messageObject)
       {
            //JavaScript messageobject is converted to a string.
            var message = ColdFusion.JSON.encode(messageObject);
           var txt=document.getElementById("myDiv");
           txt.innerHTML +=message  +"<br>";
       }
</script>
<cfwebsocket name="mycfwebsocketobject"  onmessage="mymessagehandler" subscribeto="stocks" >
<cfdiv id="myDiv"></cfdiv>
```

このコードは、mycfwebsocketobject という名前の JavaScript WebSocket オブジェクトを作成します。サーバーでメッセージを送信する場合は、mymessagehandler 関数を呼び出します。

cfwebsocket タグで subscribeTo が指定されているので、WebSocket オブジェクトによって、ユーザーは自動的に stocks チャネルをサブスクライブします。

WebSocket JavaScript 関数の使用

cfwebsocket タグを使用して JavaScript WebSocket オブジェクトを作成した後は、そのオブジェクトを使用して、次の JavaScript 関数を呼び出すことができます。

関数	説明	シンタックス	パラメータ
authenticate	特定のチャネルに対してユーザーを認証します。	authenticate(*id*, *password*) 次に例を示します。 `mycfwebsocketobject.authenticate("Ben", "password");`	• id：認証するユーザーのユーザー ID です。 • password
subscribe	クライアントが特定のチャネルをサブスクライブできるようにします。 ユーザーの年齢などの追加情報を（custom_header JavaScript オブジェクトの形式で）サーバーに送信できます。この情報を使用して、クライアントのサブスクライブを許可するかどうかを決定できます。	subscribe(channel[, custom_header][, messageHandler]) 次に例を示します。 `mycfwebsocketobject.subscribe(channel);`	• channel：クライアントがサブスクライブを要求するチャネル／サブチャネルの名前です。 • custom_header：オプションです。キーと値のペアで渡します。また、ColdFusion のキーワード selector を使用して、フィルター条件として使用する条件を指定できます（例：mycfwebsocketobject.subscribe(channelname,{age : 25,selector:"symbol eq 'adbe' and value gt 20"})）。 • messageHandler：オプションです。チャネル固有のメッセージハンドラーを指定します。メッセージハンドラーを指定した場合、サーバーから（チャネルに）送信されたデータは、（WebSocket オブジェクトの作成時に指定した）タグの onMessage 属性ではなく、特定のメッセージハンドラーに渡されます。詳しくは、574 ページの「チャネル固有のメッセージハンドラーの使用」を参照してください。

publish	チャネルにメッセージをパブリッシュします。メッセージを使用できない場合は、invokeAndPublish 関数を使用します。
	メッセージは、JavaScript オブジェクトとして任意の JavaScript データ型（配列、文字列、整数、キーと値のペアなど）で渡すことができます。
	追加の情報を送信できます（custom_header の形式で）。この情報を使用して、クライアントにパブリッシュを許可するかどうかを決定できます（例えば、「プラチナメンバーだけがパブリッシュの権限を持つ」など）。
		`publish(channel, message [, custom_header])`
		次に例を示します。
		`mycfwebsocketobject.publish("stocks",msg );`
			• channel：メッセージをパブリッシュする必要があるチャネル／サブチャネルの名前です。
			• message：パブリッシュする必要があるメッセージオブジェクトです。メッセージは、特定のチャネルをサブスクライブするクライアントに対してパブリッシュされます。
			• custom_header：オプションです。キーと値のペアで渡します。また、ColdFusion のキーワード selector を使用して、フィルター条件として使用する条件を指定できます（例：mycfwebsocketobject.publish("mychannel", message, {item:phone, selector:"itemcode eq 'HTC HD'"});）。

関数	説明	シンタックス	パラメータ

invokeAndPublish	CFC ファイル内の指定したメソッドを呼び出して、パブリッシュする必要のあるメッセージを生成します。生データ（加工が必要なデータ）を使用してメッセージを作成する場合に使用します。	`invokeAndPublish(channel, cfcName, functionName [, argumentsArray] [, custom_header])`
		次に例を示します。
		`mycfwebsocketobject.invokeAndPublish(ch, "employee", "getEmployeeDetails", ["eid_1"]);`
			• channel：メッセージをパブリッシュする必要があるチャネル／サブチャネルの名前です。
			• cfcName：メッセージに対して呼び出すメソッドが含まれる CFC です。cfcName は、完全修飾パスとして、またはアプリケーションルートを基準とするパスとして、渡すことができます。
			• functionName：メッセージに対して呼び出す必要がある、CFC 内の関数の名前です。
			• argumentsArray：指定した CFC の関数を呼び出すために渡す必要がある引数の JavaScript 配列です。
			• custom_header：オプションです。キーと値のペアで渡します。また、ColdFusion のキーワード selector を使用して、フィルター条件として使用する条件を指定できます。

getSubscriberCount	特定のチャネルのサブスクライバーの数を提供します。	`getSubscriberCount(channel)`
		次に例を示します。
		`mycfwebsocketobject.getSubscriberCount("stocks");`
			• channel：サブスクライバー数を確認するチャネル／サブチャネルの名前です。
			注意：非同期の応答を受信するインラインコールバック機能をサポートしています。

getSubscriptions	クライアントがサブスクライブしているすべてのチャネルを、JavaScript 配列として提供します。	getSubscription()	**注意：**非同期の応答を受信するインラインコールバック機能をサポートしています。
openConnection	WebSocket 接続を開きます（まだ開かれていない場合）。	`openConnection()`	
isConnectionOpen	WebSocket 接続が開かれているかどうかを確認します。 接続が開かれている場合は、true を返します。	`isConnectionOpen()`	
CloseConnection	WebSocket 接続を閉じます。 正常に完了した場合は、true を返します。	`closeConnection()`	

unsubscribe	指定したチャネルからクライアントをサブスクライブ解除します。クライアントがサブスクライブ解除した後、チャネル内のメッセージはクライアントにパブリッシュされません。	`unsubscribe(channel)`
		次に例を示します。
		`mycfwebsocketobject.unsubscribe(channel);`
			• channel：サブスクライブを解除するチャネルの名前です。

注意：これらの関数を使用する場合、呼び出しのステータスを示すブール値が返されます。ただし、呼び出しの結果はメッセージハンドラーサイドで承認されます。if (token.reqType =="getSubscriberCount) のようにして、リクエストタイプトークンを使用してメッセージハンドラーで応答を分類できます。

例

次の例は、様々な JavaScript 関数を使用する方法を示しています。

Application.cfc

```
component
{
	this.name="websocketsampleapp3";
	this.wschannels=[{name="stocks"}];
}
```

Index.cfm

```
<script>
	//messagehandler recieves all the messages from websocket
	function mycbHandler( messageobj)
	{
		var message = ColdFusion.JSON.encode(messageobj);
	    var txt=document.getElementById("myDiv");
	    txt.innerHTML +=message  +"<br>";
	}

	//openhandler is invoked when socket connection is
	function openHandler()
	{
		var txt=document.getElementById("myDiv");
	    txt.innerHTML +="open Handler invoked <br>";
	}

	function subscribeMe()
	{
		var channelname = document.getElementById("channelname").value;
		mywsobj.subscribe(channelname);
	}
	function getSubscribers()
	{
		var channelname = document.getElementById("channelname").value;
		mywsobj.getSubscriberCount(channelname);
	}
	function unsubscribe_Me()
	{
	    var channelname = document.getElementById("channelname").value;
	    mywsobj.unsubscribe(channelname);
	}
	function publishClient()
	{
		var channelname = document.getElementById("channelname").value;
		var message =  document.getElementById("msg").value;
	mywsobj.publish(channelname,message);
	}
	function get_Subscriptions()
	{
	    mywsobj.getSubscriptions();
	}
	function invokenpublish()
	{
		cfcname = document.getElementById("cfcname").value;
		fnname = document.getElementById("fnname").value;
		channelname = document.getElementById("channelname").value;
		mywsobj.invokeAndPublish(channelname, cfcname, fnname);
	}
	function invokefn()
	{
```

```
                cfcname = document.getElementById("cfcname").value;
                fnname = document.getElementById("fnname").value;
                channelname = document.getElementById("channelname").value;
                mywsobj.invoke(cfcname, fnname);
        }
        function opensocket()
        {
                var txt=document.getElementById("myDiv");
                txt.innerHTML+="opening socket"+"<br >";
                x=mywsobj.openConnection();
        }
        function stopsocket()
        {
                var txt=document.getElementById("myDiv");
                txt.innerHTML+="closing socket"+"<br >";
                x=mywsobj.closeConnection();
        }
        function checksocket()
        {
                var x=mywsobj.isConnectionOpen();
                var txt=document.getElementById("myDiv");
                txt.innerHTML+=x+"<br >";
        }
</script>
<form name="f">
<!---Define JS websocket object name and messagehandler and openhandler  --->
<cfwebsocket name="mywsobj" onMessage="mycbHandler" onOpen="openHandler"/>
<br> Subscribe to:
<input  id="channelname" name="channelname" type="text" value="stocks" >
<input  id="stocksSubscribe" type="button" value="stocksSubscribe" onclick="subscribeMe();">
<input  id="unsubscribeMe" type="button" value="unsubscribeMe" onclick="unsubscribe_Me();">
<input  id="getSubscribersCF" type="button" value="getSubscribersCF" onclick="getSubscribers();">
<input  id="getSubscriptions" type="button" value="getSubscriptions" onclick="get_Subscriptions();">
<br>
Message :<input id="msg" type="text" >
<input  id="publishMe" type="button" value="publishMe" onclick="publishClient();">
<br>
CFC Name: <input  id="cfcname" name="cfcname" type="text" value="invokeandpublish" >
Function Name: <input  id="fnname" name="fnname" type="text" value="publishall" >
<input  id="invoke_publish" type="button" value="invoke_publish" onclick="invokenpublish();">
<input  id="invoke" type="button" value="invoke" onclick="invokefn();">
<br>
<input  id="stop" name ="Close" type="button" value ="stop" onclick="stopsocket()" >
<input  id="open" name ="Open" type="button" value ="open" onclick="opensocket()" >
<input  id="check" name ="Check" type="button" value="check" onclick="checksocket()" >
<br>
<div id="myDiv"></div>
</form>
```

invokeandpublish.cfc

```
component
{

       public function publishall()
       {
              return "All Clients";
       }
}
```

シナリオ：サブスクライブとパブリッシュ

ユーザーが親チャネルをサブスクライブした場合、すべてのサブチャネルも自動的にサブスクライブします。子チャネルをサブスクライブした場合は、親チャネルはサブスクライブしません。

次の表では条件について説明します。

この表では、クライアント 1、クライアント 2、クライアント 3、クライアント 4 の 4 つのクライアントがチャネル Stocks、Stocks.Finance、Stocks.Banks、Stocks.Finance.Adobe をサブスクライブしています。

クライアント	サブスクライブするチャネル	Stocks へのパブリッシュ時にメッセージを受信	Stocks.Finance へのパブリッシュ時にメッセージを受信	Stocks.Banks へのパブリッシュ時にメッセージを受信	Stocks.Finance.Adobe へのパブリッシュ時にメッセージを受信
クライアント 1	Stocks	はい	はい	はい	はい
クライアント 2	Stocks.Finance	いいえ	はい	いいえ	はい
クライアント 3	Stocks.Banks	いいえ	いいえ	はい	いいえ
クライアント 4	Stocks.Finance.Adobe	いいえ	いいえ	いいえ	はい

シナリオ：サブスクライバーリストの取得

次の表では、getSubscribers 関数を使用したときにサブスクライバーリストを受信する条件を説明します。

4 つのクライアントがあるものとします。各クライアントは、Stocks、Stocks.Finance、Stocks.Banks、Stocks.Finance.Adobe のチャネルのうち 1 つだけをサブスクライブします（同じチャネルをサブスクライブしているクライアントはいません）。

getSubscribers 関数が呼び出されたチャネル	サブスクライブしているクライアントの ID が返されるチャネル
Stocks	Stocks
Stocks.Finance	Stocks.Finance、Stocks
Stocks.Banks	Stocks.Banks、Stocks
Stocks.Finance.Adobe	Stocks.Finance.Adobe、Stocks.Finnance、Stocks

チャネルリスナー関数の使用

チャネルリスナーとは、チャネルへのメッセージのフローを決定する関数のセットです。WebSocket オブジェクトの JavaScript 関数が呼び出されると、次にチャネルリスナーの関数が呼び出されます。例えば、subscribe 関数を呼び出すと、この関数はリスナー CFC から allowSubscribe 関数を呼び出します。

次のチャネルリスナーを使用できます。

- **デフォルトのチャネルリスナー：**Application.cfc でのチャネルの定義時にチャネルリスナーが指定されていない場合に使用されます。
- **ユーザー定義のチャネルリスナー：**チャネルの作成時にチャネルリスナー CFC が指定された場合に使用されます。チャネルリスナーファイル内のメソッドを更新して、特定のビジネスロジックを実装します。

  チャネルのサブスクライブやパブリッシュの権限を制限するには、カスタムチャネルリスナーを指定します。また、カスタムチャネルリスナーを使用して、メッセージのフォーマットを編集し、クライアントがメッセージの受信対象であるかどうかを決定することもできます。

ユーザー定義のチャネルリスナーの設定

- （必須）webroot/cfide/websocket/ChannelListener にあるデフォルトのリスナーを、次のコードで示すように拡張します。

  component extends="CFIDE.websocket.ChannelListener"

- ユーザー定義のチャネルリスナー CFC は、アプリケーションと同じディレクトリまたはそのサブディレクトリに置く必要があります。
- ビジネスロジックを実装するために上書きできるのは、デフォルトリスナーの CFC で指定されている関数のみです。カスタム関数はすべて無視されます。

リスナー関数が情報を受信／提供する方法

リスナー内の関数は、次の 3 つの構造体を使用して情報を受信または提供します。これら 3 つの構造体を使用して、カスタム情報を追加できます。

- **clientInfo**：クライアントに関する情報です。値は次のとおりです。
  - channelName
  - clientId：一意のクライアント ID です。

    ロールなどの追加プロパティの設定に使用できます。
  - connectionTime：date オブジェクトでの接続時間です。
- **subscriberInfo**：サブスクライブ要求で渡されるカスタムオプションを含む構造体です。connectionInfo サブキーとして ConnectionInfo オブジェクトも含みます。
- **publisherInfo**：パブリッシュ要求で渡されるカスタムオプションを含む構造体です。connectionInfo サブキーとして ConnectionInfo オブジェクトも含みます。

次の表では、WebSocket JavaScript 関数とそれに対応するチャネルリスナー CFC の関数を示します。

WebSocket オブジェクトの JavaScript 関数	**チャネルリスナー CFC で呼び出される関数**
subscribe	allowSubscribe
unsubscribe	afterUnsubscribe
publish	allowPublish
invokeAndPublish	allowPublish
getSubscribers	なし
getSubscriptions	なし
invoke	なし
openConnection	なし
isConnectionOpen	なし
closeConnection	なし

チャネルリスナー関数

次の表ではリスナーファイルの関数を示します。

関数	説明	シンタックス	パラメータ
allowSubscribe	チャネル／サブチャネルをサブスクライブする前に呼び出されます。 要求されたクライアントによる指定されたチャネルのサブスクライブを許可できるかどうかを調べるために使用されます。true が返された場合、要求されたクライアントのサブスクライブを許可します。 subscriberInfo オブジェクトで定義されているプロパティを、承認の決定に使用できます。	`allowSubscribe(subscriberInfo)`	• subscriberInfo：サブスクライバーの情報の構造体です。関数 subscribe によって custom_header 構造体で渡される情報を含みます。connectionInfo オブジェクトが含まれ、ConnectionInfo キーによってアクセスできます。
allowPublish	チャネル／サブチャネルへのパブリッシュの前に呼び出されます。 要求されたクライアントへの指定されたチャネルでのパブリッシュを許可できるかどうかを調べるために使用されます。 publisherInfo 構造体で定義されているプロパティを使用して、承認を決定できます。 true が返された場合、要求されたクライアントのパブリッシュを許可します。	`allowPublish(publisherInfo)`	• publisherInfo：パブリッシャーの情報の構造体です。関数 publish によって custom_header 構造体で渡される情報を含みます。connectionInfo オブジェクトが含まれ、ConnectionInfo キーによってアクセスできます。
beforePublish	要求されたチャネル／サブチャネルでメッセージをパブリッシュする前に呼び出されます。必要に応じてビジネスロジックを実行し、メッセージをフォーマットするために使用されます。	beforePublish(message, publisherInfo)	• message：クライアントにパブリッシュする必要のあるメッセージです。Any 型を指定できます。 • publisherInfo：パブリッシャーの情報の構造体です。
canSendMessage	サブスクライブしているクライアントにメッセージを送信する前に呼び出されます。 特定のクライアントにメッセージを送信する必要があるかどうかを判定するために使用されます。チャネルをサブスクライブしているすべてのクライアントについて個別に呼び出されます。subscriberInfo および publisherInfo オブジェクトで定義されているプロパティは、メッセージでのクライアントの関心を検出するのに役立ちます。	canSendMessage(message, subscriberInfo, publisherInfo)	• message：クライアントにパブリッシュする必要のあるメッセージです。Any 型を指定できます。 • subscriberInfo：サブスクライバーの情報の構造体です。 • publisherInfo：パブリッシャーの情報の構造体です。
beforeSendMessage	サブスクライブしているクライアントにメッセージを送信する前に呼び出されます。送信前にクライアントの要求に従ってメッセージをフォーマットするために使用できます。この関数は、クライアントごとに実行されます。	beforeSendMessage(message, subscriberInfo)	• message：クライアントにパブリッシュする必要のあるメッセージです。Any 型を指定できます。 • subscriberInfo：サブスクライバーの情報の構造体です。
afterUnsubscribe	チャネル／サブチャネルをサブスクライブ解除した後で呼び出されます。必要に応じて設定をクリアするために使用されます。	afterUnsubscribe (subscriberInfo)	• subscriberInfo：サブスクライバーの情報の構造体です。

チャネル固有のメッセージハンドラーの使用

WebSocket オブジェクトを作成するときに、サーバーからの応答を管理するためのメッセージハンドラーを定義できます。ハンドラーはすべてのチャネルからのメッセージを管理できます。さらに、チャネルをサブスクライブするときに、チャネル固有のメッセージハンドラーを定義できます。

例

Application.cfc

component

```
{
    this.name = "channelspecifichandlerexample";
    this.wschannels = [{name="stocks"}, {name="news"}, {name="products"}];
}
```

index.cfm

```
<script type="text/javascript">
    function stockhandler(stocksmessageobj){
        //write appropriate logic here to handle data coming on stock channel
        if (stocksmessageobj.data != null) {
            var message = stocksmessageobj.data;
            var datadiv = document.getElementById("myDiv");
            datadiv.innerHTML += "Stock Handler:  " + message + "<br />";
        }
    }
    function newshandler(newsmessageobj){
       //write appropriate logic here to handle data coming on news channel
        if (newsmessageobj.data != null) {
            var message = newsmessageobj.data;
            var datadiv = document.getElementById("myDiv");
            datadiv.innerHTML += "News Handler:  " + message + "<br />";
        }
    }
    function productshandler(productsmessageobj){
       //write appropriate logic here to handle data coming on products channel
        if (productsmessageobj.data != null) {
            var message = productsmessageobj.data;
            var datadiv = document.getElementById("myDiv");
            datadiv.innerHTML += "Product Handler:  " + message + "<br />";
        }
    }
    function subscribeMe(){
        var channel = document.getElementById("channeloption").value;
        switch (channel) {
            case "stocks":
                mysock.subscribe("stocks", {}, stockhandler);
                break;
            case "news":
                mysock.subscribe("news", {}, newshandler);
                break;
            case "products":
                mysock.subscribe("products", {}, productshandler);
                break;
        }
    }
    function mycbHandler(messageobj){
        var message = ColdFusion.JSON.encode(messageobj);
        var datadiv = document.getElementById("myDiv");
        datadiv.innerHTML += "Message Handler : " + message + "<br />";
```

```
    }
</script>
<cfwebsocket name="mysock" onmessage="mycbHandler"/>
<form>
    <select id="channeloption">
        <option>
            stocks
        </option>
        <option>
            news
        </option>
        <option>
            products
        </option>
    </select>
    <input id="subscribe" name="subscribe" value="Subscribe" type="button"
           onclick="subscribeMe();">
</form>
<div id="myDiv">
</div>
```

PublishMessage.cfm

```
<cfscript>
    if(isdefined("form.publish"))
        WsPublish(#form.channel#, #form.message#);
</cfscript>
<cfform method="post">
    <cfselect name="channel">
        <option>
            stocks
        </option>
        <option>
            news
        </option>
        <option>
            products
        </option>
    </cfselect>
    Message:
    <input id="message" name="message" type="text">
    <cfinput id="publish" name="publish" value="publish" type="submit">
</cfform>
```

カスタム情報の指定

JavaScript 関数の subscribe、publish、invokeandpublish と、サーバーサイド関数の WSPublish を使用すると、カスタム情報（例えば、年齢やステータスなど）をキーと値のペアの形式で渡すことができます。

これらのカスタムデータは、チャネルリスナーの subscriberInfo で使用できます。

例 1：subscribe を使用したカスタム情報

次に示す例では、

1. subscribe 関数内で、クライアントの年齢を指定します。
2. チャネルリスナーの allowSubscribe メソッド内で、年齢の制限を指定します。

Application.cfc

```
component
{
this.name = "customoptionexample";
this.wschannels = [{name="Testchannel", cfclistener="TestListener"}];
}
```

Index.cfm

```
<script type = "text/javascript">
       function  msgHandler(messageobj)
       {
              var ouputdiv=document.getElementById("myDiv");
              var message = ColdFusion.JSON.encode(messageobj);
              ouputdiv.innerHTML+=message +"<br >" +"<br>";
       }
       function subscribeMe()
       {
              var clientAge = document.getElementById("age").value;
                TestSocket.subscribe("Testchannel", {age: clientAge});       }</script>
<cfwebsocket name="TestSocket" onMessage="msgHandler"/>
<br>
Age <input id="age" name="age" type="text">
<br>
<input id="stocksSubscribe" type="button" value="Subscribe" onclick="subscribeMe();">
<div id="myDiv"></div>
```

TestListener.cfc

```
component extends="CFIDE.websocket.ChannelListener"
{

       public boolean function allowSubscribe(Struct subscriberInfo)
       {
              if(structKeyExists(subscriberInfo, "age"))
                     if((subscriberInfo.age gt 18))                    {
                           return true;
                     }
                     else
                     {
                           return false;
                     }
              else
                     return false;
       }

}
```

例 2：publish を使用したカスタム情報

次の例は、入札の値データを伝達する方法を示しています。

1. publish 関数内で、入札の値をカスタム情報として指定します。
2. チャネルリスナーで、入札の値を現在の入札値と比較して検証する条件を指定します。入札が有効である場合にのみ、メッセージをブロードキャストします。

Application.cfc

```
component
{
    this.name = "customoptionexample1";
    this.wschannels = [{name="bidchannel", cfclistener="bidListener"}];

    boolean function onApplicationStart()
    {
        application.bidvalue = 1;
    }

}
```

Index.cfm

```
<script type="text/javascript">
    function msgHandler(messageobj){
        if (messageobj.data != null) {
            var ouputdiv = document.getElementById("myDiv");
            var message = messageobj.data;
            ouputdiv.innerHTML += message + "<br >" + "<br>";
        }
        if (messageobj.code == -1 && messageobj.reqType == "publish") {
            var ouputdiv = document.getElementById("myDiv");
            var message = "Bid amount is less than current bid value";
            ouputdiv.innerHTML += message + "<br >" + "<br>";
        }
    }
    function publishme(){
        var bidvalue = document.getElementById("amount").value;
        var clname = document.getElementById("name").value;
        var message = "Bid placed by " + clname + " Amount " + bidvalue;
        TestSocket.publish("bidchannel", message, {
            value: bidvalue
        });
    }
</script>
<cfwebsocket name="TestSocket" onmessage="msgHandler" subscribeto="bidchannel"/><br>
Bid Amount:
<input id="amount" name="amount" type="text">
Name:
<input id="name" type="text">
<input id="publishmessage" type="button" value="Publish" onclick="publishme();">
<div id="myDiv">
</div>
```

bidListener.cfc

```
component extends="CFIDE.websocket.ChannelListener"{ public boolean function
    allowPublish(Struct publisherInfo)
    {
    if(structKeyExists(publisherInfo, "value"))
    if((publisherInfo.value gt application.bidvalue))
    {
    application.bidvalue = publisherInfo.value;
    return true;
    }
    else
    { return false;
    }
    else
    return false;
    }
    }
```

セレクターの使用

セレクターは、チャネルをサブスクライブするとき、またはメッセージをパブリッシュするときのフィルターロジックを提供します。

チャネルをサブスクライブするときは、サブスクライバー情報構造体の一部としてセレクターを提供します。例えば、クライアントはセレクター product eq abc を使用してチャネルをサブスクライブします。パブリッシャー情報 product:abc を含むメッセージだけが、サブスクライバーにパブリッシュされます。

同様に、メッセージをパブリッシュするときに、パブリッシャー情報構造体の一部としてセレクターを含めることができます。セレクターで指定されている条件を満たすサブスクライバーに対してのみ、メッセージがパブリッシュされます。

注意：（セレクターを機能させる場合は）カスタムチャネルリスナーの CFC で canSendMessage メソッドを使用できません。

次の例では、セレクターを使用してチャネルをサブスクライブする方法を示します。

1 Application.cfc で selectorchannel というチャネルを作成します。

```
component
{
    this.name = "websocketApp1";
    this.wschannels = [{name="selectorchannel"}];
}
```

2 CFM を作成し、ユーザーがチャネルをサブスクライブできるようにして、セレクターの条件を「値 > 指定された値」に設定します（例えば、特定のしきい値を超える株価の値のみを表示するものとします）。

```
<script type="text/javascript">
    function  msgHandler(messageobj)
       {
              if (messageobj.data != null) {
                     var ouputdiv = document.getElementById("myDiv");
                     var message = messageobj.data;
                     ouputdiv.innerHTML += message + "<br >" + "<br>";
                     }
       }
       function subscribeMe()
       {
              var amt = document.getElementById("amount").value;
              var selectstring="value gt "+ amt;
                 TestSocket.subscribe("selectorchannel", {selector : selectstring});
              document.getElementById("stocksSubscribe").style.display = 'none';
              document.getElementById("l1").style.display = 'none';
              document.getElementById("l2").style.display = 'block';
              document.getElementById("publisme").style.display = 'block';
       }
       function publishme()
       {
              var amt = document.getElementById("amount").value;
              TestSocket.publish("selectorchannel","Value is " +amt,{value: amt})
       }
</script>
<cfwebsocket name="TestSocket" onMessage="msgHandler"/>
<br />
<lablel  id="l1">Stock value above which you want to recieve message</lablel>
<label id="l2" style="display:none;">Value</label>
<input id="amount" name="amount" type="text">
<br>
<input id="stocksSubscribe" type="button" value="Subscribe" onclick="subscribeMe();">
<input id="publisme" type="button" value="Publish" onclick="publishme();" style="display:none;">
<div id="myDiv"></div>
```

WebSocket 認証の詳細の指定

WebSocket 認証の詳細をアプリケーションレベルで指定できます。アプリケーションレベルの認証をサポートするために、OnWSAuthenticate というアプリケーションメソッドが追加されました。

1 Application.cfc で、onWSAuthenticate 関数を定義します。

2 JavaScript 関数の authenticate を呼び出します。この関数により、onWSAuthenticate 関数が呼び出されます。

注意：認証を特定のチャネルに限定することはできません。すべてのチャネルでクライアントを認可しないようにする場合は、チャネルリスナーの allowSubscribe 関数を使用して実行できます。

onWSAuthenticate

説明

ユーザーを認証します。

シンタックス

OnWSAuthenticate(username, password, connectionInfo)

パラメータ

パラメータ	説明
username	認証するユーザーの名前です。
password	ユーザーのパスワードです。
connectionInfo	次のキーが含まれる構造体です。 • Authenticated：YES|NO • ConnectionTime：接続タイムスタンプ。 • clientID：クライアントの一意の ID。 また、カスタムキーもサポートされます。 例えば、ユーザーのロール、ステータスまたは年齢を指定できます。 connectionInfo は、所定の WebSocket クライアントのすべてのチャネルで共有されます。また、変更は、そのクライアントのすべてのサブスクリプションにわたって保持されます。

例

次の例では、onWSAuthenticate 関数を使用して、ユーザーを検証し、ユーザーロールの関連付けを行います。

注意：この例が機能するには、ユーザー定義関数を実装する必要があります。

```
component
{
    this.name="websocketsampleapp23";
    this.wschannels=[{name="stocks",cfclistener="stocksListener"}];

    function  onWSAuthenticate(username, password, connectionInfo)
       {
          //write appropriate logic to fetch user password in funtion checkPassword

          If(checkPassword(username) eq password)
          {
            connectionInfo.authenticated="YES";
        //Role is the custom information that you provide
     connectionInfo.role= "admin";

            return true;
          }
          else{
            connectionInfo.authenticated="NO";
           return false;
          }
          writedump("#connectionInfo#","console");

         }

}
```

WebSocket でのシングルサインオンの有効化

ユーザーは、ColdFusion サーバーで既に認証されている場合、WebSocket チャネルをサブスクライブするときに資格情報を入力する必要はありません。cfwebsocket タグで useCFAuth = "true" のみを指定する必要があります。

次の例では、cflogin タグを使用してログインページを作成しています。認証の後、クライアントはチャネルをサブスクライブします。つまり、クライアントは、サブスクライブ時にログイン資格情報を渡しません。この場合は、onWSAuthenticate 関数を準備したり、JavaScript 関数の authenticate を使用して前述の関数を呼び出す必要はありません。

例

1 cflogin タグを使用してログインページを作成します。

```
<form name="form1" method="post" >
        User: <input name="username" type="text" id="username" value="admin"><br>
        Pass: <input name="password" type="text" id="password"><br>
        <input type="submit" name="Submit" value="Submit">
 </form>
<cfif structKeyExists(form,"username")>
    <cflogout>
    <cfset r = "user">
    <cfif FORM.username is "admin">
    <cfset r = "admin">
    </cfif>
    <cflogin idletimeout="1800">
            <cfloginuser
                 name = "#FORM.username#"
                 password ="#FORM.password#"
                 roles = #r#>
     </cflogin>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput><br>
<cfoutput>Authorized role: #GetUserRoles()#</cfoutput><br>
<cflocation url="index.cfm">
</cfif>
```

2 チャネル shares およびリスナー channelListener を使用して Application.cfc を作成します。

```
component
{
   this.name = "myssoexample";
   this.wschannels = [ {name = "sso", cfclistener="ChannelListener"}];
 }
```

3 ユーザーによるチャネルのサブスクライブおよびメッセージのパブリッシュを許可する CFM を作成します。

```
<script>
var mycbHandler = function(msg)
    {
    if(msg.data)
        {
            var messageObject = msg.data;
            var txt=document.getElementById("myDiv");
        if((messageObject.indexOf("{")!=-1)){
                    var jsonValue = (new Function( "return( " + messageObject + " );" ))();
                if(jsonValue){
                txt.innerHTML+="<br>Authenticated : " + jsonValue.authenticated;
                txt.innerHTML+="<br>UserName : " + jsonValue.username;
                txt.innerHTML+="<br>Role : " + jsonValue.roles;
                }
           }
        }
    }
var openHandler = function()
    {
        mysock.publish("sso","hii");
    }
</script>
<form name="f">
<cfwebsocket name="mysock" onMessage="mycbHandler" subscribeto="sso" onOpen="openHandler"
useCFAuth="yes"/>
<div id="myDiv"></div>
</form>
```

4 パブリッシュ時にサブスクライバー情報を返す ColdFusion リスナーファイルを作成します。

```
component extends="CFIDE.websocket.ChannelListener"
{
public any function beforeSendMessage(any message, Struct subscriberInfo)
  {
    writedump(var=subscriberInfo.connectionInfo, output="console");
     return subscriberInfo.connectionInfo;
  }

}
```

サーバーサイド API

WebSocket のサーバーサイド API を使用すると、CFC でチャネルまたは特定のクライアントと通信できます。例えば、クライアントからのメッセージを受信すると、サーバーは特定のメッセージを送信します。

次の表ではサーバーサイド関数について詳しく説明します。

関数名	シンタックス	説明
WSgetSubscribers	WSgetSubscribers (channel)	clientID および subcriberInfo をキーとして構造体の配列を返します。
WSPublish	WSPublish(channel, message [, filterCriteria])	特定のチャネルにメッセージを送信します。オプションで、フィルター条件の構造体を指定できます。
WSGetAllChannels	WSGetAllChannels ()	Application.cfc で定義されているすべてのチャネルを配列として提供します。

サーバーからのレスポンスの解釈

次は、サーバーから受け取るレスポンスのサンプルです。

```
{"clientid":2077108630,"ns":"coldfusion.websocket.channels","reqType":"welcome","code":0,"type":"response
","msg":"ok"}
{"clientid":2077108630,"ns":"coldfusion.websocket.channels","reqType":"subscribe","code":0,"type":"respon
se","msg":"ok"}
```

次の表で、レスポンスのキーを説明します。

キー	説明
clientid	クライアントに割り当てられている一意の ID です。
ns	ColdFusion WebSocket の名前空間です。
reqType	JavaScript 関数（例えば、publish や invokeandpublish）が表すリクエストのタイプです。
code	code の節を参照してください。
type	response または data のいずれかです。 • response：リクエストが成功した場合に伝達されます。 • data：チャネルが受け取るデータです。
msg	レスポンスに適用されます。成功した場合は ok です。失敗したときは、失敗の原因が返されます。
subscriberCount	サブスクライバーの数を表す整数です。
channels	getSubscriptions に適用され、サブスクライブしているすべてのチャネルのリストです。
data	メッセージは、data によって伝達されます。
publisherID	パブリッシャーのクライアント ID です。WSPublish からのメッセージである場合、ID は 0 です。
channelname	チャネルの名前です。

次は、メッセージハンドラーに送信される subscribeto のレスポンスのサンプルです。

```
{"clientid":2077108664,"ns":"coldfusion.websocket.channels","reqType":"welcome","code":0,"type":"response
","msg":"ok"}
{"clientid":2077108664,"ns":"coldfusion.websocket.channels","channelsnotsubscribedto":"stocks.a","reqType
":"subscribeTo","code":0,"type":"response","msg":"Subscription failed for channel(s)
'stocks.a'.","channelssubscribedto":"stocks"}
```

この場合は、cfwebsocket タグの subsscribeTo 属性に関連するシナリオに適用される channelssubscribedto と channelsnotsubscribedto のキーが含まれています。

ポイントツーポイント通信での WebSocket の使用

クライアントが、複数のクライアントにメッセージを送信する必要がないことを前提とします。つまり、この双方向通信は、ある 1 つのクライアントとサーバーの間だけのものです。

そのため、ブロードキャストモデルとは異なり、通信用のチャネルは必要ないため、Application.cfc でチャネルを定義する必要はありません。

ポイントツーポイント通信を設定するには、次の手順を行います。

1 cfwebsocket タグを使用して WebSocket オブジェクトを作成します。

詳しくは、565 ページの「cfwebsocket を使用した WebSocket オブジェクトの作成」を参照してください。

2 メッセージをサーバーに送信するには、JavaScript のメソッド Invoke を使用します。

3 追加のメッセージを送信するには、関数 WSSendMessage を使用します。

invoke

説明

CFC 内の特定の関数を呼び出します。CFC の関数によって返された値は、メソッドを呼び出したクライアントに返送されます。

シンタックス

invoke(CFCName, functionName [, argumentsArray])

パラメータ

パラメータ	説明
cfcName	（特定の関数を呼び出す）CFC ファイル名です。
functionName	CFC ファイル内の関数名です。
argumentsArray	配列にした関数の引数です。

使用方法

WSSendMessage 関数を使用して、この関数内でクライアントに追加メッセージを返します。

継続的にクライアントにメッセージを送信するには、invoke を使用して呼び出すメソッド内にスレッドを作成する必要があります。

さらに、スレッド内でメッセージを送信し続けることもできます。

例

```
mycfwebsocketobject.invoke("employee","getdept",["eid_2"]);
```

WSSendMessage

説明

メソッドを呼び出した特定のクライアントにメッセージを送信します。これは、invoke WebSocket JavaScript メソッドによって呼び出された関数の一部として含めることができます。

戻り値

なし

シンタックス
WSSendMessage(message)

パラメータ

パラメータ	説明
message	必須。メッセージオブジェクトです。Any 型を指定できます。

例：ポイントツーポイント通信

次の例では、ポイントツーポイント通信を実装する方法を示します。この例では、mycfc.cfc で定義した 3 つの関数を呼び出します。

- メソッド f1 は値を返します。
- メソッド f2 は引数を持ち、文字列を返します。また、クライアントは、f2 内で使用される WSSendMessage メソッドから追加メッセージを受け取ります。
- メソッド f3 内で、特定の間隔でクライアントにメッセージを送信するスレッドを作成します。

1 index.cfm という CFM ページを作成します。

```
<script type="text/javascript">
	function msgHandler(msgobj){
	    var txt = document.getElementById("myDiv");
	    var message = ColdFusion.JSON.encode(msgobj);
	    txt.innerHTML += message + "<br >" + "<br>";
	}
	function invokecfcfn(){
		var fname= document.getElementById("fnname").value;
		if (fname == "f2") {
			alert("f2 selected");
			mysocket.invoke("mycfc", "f2", ["echo"]);
		}
		else
			mysocket.invoke("mycfc", fname);

	}
</script>
<cfwebsocket name="mysocket" onmessage="msgHandler"/>
<form>
	<select id="fnname">
		<option>f1</option>
		<option >f2</option>
		<option>f3</option>
	</select>

	<input id="invokefn" name="invokefn" value="Invoke CFC function " type="button"
onclick="invokecfcfn();">
	<div id="myDiv">
	</div>
</form>
```

2 クライアントページから呼び出される関数を含む CFC を mycfc.cfc という名前で作成します。

```
<cfcomponent>

	<cffunction name="f1" >
		<cfreturn "Message returned from f1">
	</cffunction>

	<cffunction name="f2" returntype="string" >
		<cfargument name="arg1" type="string" required="true"  >
		<cfset msg= "Message from wsssendmessage of f2 which you called with arg " & arg1>
		<cfset wssendMessage(msg)>
		<cfreturn "Message returned from f2">
	</cffunction>

	<cffunction name="f3" >
		<cfthread action="run" name="t1" >
			<cfloop index="i" from="1" to="10">
				<cfset sleep(20000)>
				<cfset wssendMessage("Message #i# from wsssendmessage of f3 #now()#")>
			</cfloop>
		</cfthread>
		<cfreturn "Thread initiated in f3">
	</cffunction>
</cfcomponent>
```

ColdFusion WebSocket のエラー処理

WebSocket のエラーは、次のように分類されます。

カテゴリ	説明	応答コード
チャネルエラー	例えば、チャネルが存在しない場合や、クライアントが既にそのチャネルをサブスクライブしている場合	-1
アプリケーションエラー	CFC を呼び出したときのランタイムエラー	4001
成功	リクエストが正常に完了した場合	0

注意：エラーハンドラーを定義している場合、エラーハンドラーは -1 と 4001 のレスポンスを受け取ります。定義していない場合、レスポンスはメッセージハンドラーに渡されます。

ColdFusion Administrator を使用した WebSocket の設定

ColdFusion Administrator（サーバー設定／ WebSocket）を使用して、次の WebSocket 関連の詳細を指定します。

- WebSocket サーバーがリスンするポート
- ソケットのタイムアウト
- 送受信されるパケットのデータサイズ
- Flash のフォールバックを有効にするかどうか

ブラウザーの WebSocket サポート

次の表では、WebSocket に対するブラウザーサポートの一覧を示します。

ブラウザー	WebSocket のサポート
Internet Explorer 6、7、8、9	Flash Player がインストールされている場合にサポートされます。
Mozilla Firefox 3.x	Flash Player がインストールされている場合にサポートされます。
Mozilla Firefox 6、7、8	サポートされます。
Google Chrome 15.x 以降	サポートされます。
Safari 5.x	サポートされます。
（Android）デフォルトのブラウザー	サポートされます。
（iPad）Safari	サポートされます。

メディアプレーヤーの機能拡張

このリリースでの機能拡張は次をサポートします。

- HTML 5 ビデオ用の再生機能
- Flash Player がインストールされていない場合の HTML 5 ビデオプレーヤーへのフォールバック
- ブラウザーに依存しないビデオコントロール
- Flash ビデオの動的ストリーミング
- メディアプレーヤー用の高度なスキニング
- Flash ビデオの再生リスト
- HTML の track 要素を使用した SRT 形式での字幕の埋め込み
- Open Source Media Framework（OSMF）で構築されたプラグインを使用するメディアプレーヤーの拡張。例として、
  - YouTube サーバーでのビデオの再生
  - リニアモードおよび非リニアモードでのビデオ内での広告の表示による、ステージビデオサポートの使用
- ビデオへのタイトルの追加

注意：メディアプレーヤーの動作は、使用しているデバイスのビデオ機能に依存します。

HTML 5 のビデオ再生機能

Flash ビデオの再生とは別に、HTML 5 準拠ブラウザーでサポートされる任意の形式のビデオを再生できます。

HTML 5 ビデオのソースを、次のいずれかの方法で指定します。

- ビデオソースを 1 つだけフィードしている場合は、次の例で示すようにそれを指定します。

```
<cfmediaplayer name="Player" source="myvideo.mp4" width="200" height="200" />
```

- 複数のビデオをフィードするには、次の例で示すようにソースを指定します。

```
<cfmediaplayer name="Player" width=200 height=200>
<source src="movie.ogg" type="video/ogg" />
<source src="movie.mp4" type="video/mp4" />
<source src="movie.webm" type="video/webm" />
</cfmediaplayer>
```

注意：source タグは、HTML 5 のビデオソースタグです。source タグを指定するときは、type 属性を使用することをお勧めします。

- JavaScript の関数 ColdFusion.MediaPlayer.setSource を使用して、メディアプレーヤーのソースを動的に設定します。

プレーヤーで最善の結果を得るには、次にようにコードで doctype を指定します。

<!DOCTYPE html>

cfmediaplayer のフォールバックプラン

ブラウザーの再生機能によっては、cfmediaplayer が代替の再生プランを提供します。cfmediaplayer は、Flash から HTML と、HTML から Flash の両方のフォールバックをサポートします。

フォールバック機能は次の状況で適用されます。

状況	フォールバック
Flash がインストールされていない	ブラウザーが HTML 5 に準拠し、ビデオソースがブラウザーでサポートされている場合、HTML 5 ビデオ再生にフォールバックします。 そうでない場合は、エラーになり、プレーヤーのコンテナが表示されます。
ブラウザーが HTML 5 のビデオ再生機能を備えていない	Flash Player を確認します。 Flash Player が使用できる場合、Flash Player でコンテンツを再生します（サポートされる形式の場合）。そうでない場合は、エラーになり、プレーヤーのコンテナが表示されます。
Flash がインストールされておらず、ブラウザーが HTML 5 ビデオをサポートしない	エラーになり、プレーヤーのコンテナが表示されます。
Flash と HTML の両方の再生がサポートされており、Flash がデフォルトである	使用できる場合、メディアプレーヤーは特定のビデオソースから Flash ビデオの実行を試みます。または、cfmediaplayer の type 属性の値として html を指定しない場合、HTML 5 ビデオの読み込みを試みます。

再生タイプの指定

次の例で示すように、type 属性に Flash または HTML を指定して、再生の環境設定を設定できます。

- <cfmediaplayer source="myvideo.mp4" type="html"/>
- <cfmediaplayer source="myvideo.mp4" type="flash"/>

再生は Internet Explorer 9 または Google Chrome で行われるものとします。デフォルトでは、値は flash に設定されます。

cfmediaplayer は、ブラウザーが再生環境設定をサポートしている場合にのみ、ユーザーが指定した値を使用します。例えば、ユーザーが type に flash を指定し、ブラウザーに Flash Player がインストールされていない場合、HTML 5 にフォールバックします。

cfmediaplayer タグを使用したビデオポスターイメージ、ループ再生、タイトルの指定

- posterImage：ビデオ再生にポスターイメージを設定するには、この属性を指定します。

  値は URL または相対アドレスです。

- repeat：メディアプレーヤーがビデオの最後に達した後で最初のフレームから最後のフレームまで再生を続けるには、この属性を true に設定します。

  デフォルト値は false です。

- title：メディアプレーヤーにタイトルを設定できます。

  タイトルはメディアプレーヤーの左上隅に表示されます。title が指定され、hideTitle が指定されていない場合、hideTitle は false に設定されます。

title が指定されておらず、hideTitle が false に設定されている場合、ビデオのファイル名がタイトルとして表示されます。

例

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<cfmediaplayer
name="player_html"
title="End Game"
source="/videos/gizmo.ogv"
posterImage="/images/music.jpg"
repeat="true">
</cfmediaplayer>
</body>
</html>
```

メディアプレーヤーの機能の拡張

注意：この機能は、Flash ビデオに対してのみ適用されます。

FlashVars を使用すると、OSMF プラグインと Strobe メディアプレーヤーの両方の機能を使用するように、メディアプレーヤーを拡張できます。

ここで説明するように、OSMF マーケットのプラグインを使用できます。2 つの一般的なプラグインが示されています。

メディアプレーヤーの機能を拡張する一般的な手順は次のとおりです。

1 FlashVars および SWF を（プラグインベンダーから）入手します。

2 cfmediaplayer で、FlashVars を指定して param タグを挿入します。

YouTube サーバーでのビデオの再生

次の例では、cfmediaplayer を使用して YouTube サーバーでホストされているビデオを再生する方法を示します。

```
<!DOCTYPE html >
<html>
<head>
</head>
<body>
<cfmediaplayer
    name="player_youtube"
    source="http://www.youtube.com/watch?v=gv68hDObACk&feature=feedrec_grec_index"
type="flash"
    >
    <param name="flashvars"
value="plugin_YoutubePlugin=/CFIDE/scripts/ajax/resources/cf/assets/YouTubePlugin.swf" />
</cfmediaplayer>
</body>
</html>
```

この場合、ソースは実際の YouTube ビデオリンクです。FlashVars を使用して、YouTube プラグイン SWF を指し示します。

ステージビデオを使用したリニアおよび非リニアの広告の再生

ステージビデオを使用して広告を再生するようにメディアプレーヤーを拡張できます。広告は、ビデオ内でリニアモードおよび非リニアモードで再生します。

例

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function nonLinear()
{
    var player = ColdFusion.MediaPlayer.getPlayer('player');
    player.displayNonLiniarAd (
            "http://gcdn.2mdn.net/MotifFiles/html/1379578/PID_938961_1237818260000_women.flv", {
                right: 10 * Math.random(),
                bottom: 10 * Math.random(),
                scaleMode: "none"});
}
</script>
</head>
<body style="background:#FFFFFF">
<input name = "AdvertisementPlugin" value="Advertisement" type="button" onClick="nonLinear('player')">
<cfmediaplayer
    name="player"
    source="/videos/cathy2_HD.mp4"
    type="flash"
    >
    <param name="flashvars"
value="plugin_ads=/CFIDE/scripts/ajax/resources/cf/assets/AdvertisementPlugin.swf" />
</cfmediaplayer>
</body>
</html>
```

この例では、FlashVars を使用し、広告プラグインを指し示します。リニアモードおよび非リニアモードでビデオを再生するには、プラグインのカスタム JavaScript 関数を使用します。

リファレンス

- サポートされる FlashVars について詳しくは、http://www.osmf.org/downloads/pdf/using_fmp_smp.pdf で提供されているマニュアルの「**Chapter 2: Configuring the Player**」を参照してください。
- メディアプレーヤーで使用できる他のプラグインを探すには、次の URL にアクセスしてください。

  http://www.osmf.com/partner.php
- cfmediaplayer 内でプラグインを使用する方法について詳しくは、次の URL で提供されているドキュメントの「**Chapter 4: Advanced Topics**」の「Using Plug-ins」を参照してください。

  http://www.osmf.org/downloads/pdf/using_fmp_smp.pdf
- Strobe メディア再生について詳しくは、次の URL で提供されているプレゼンテーションを参照してください。

  http://www.osmf.com/downloads/pdf/Strobe_Media_Playback_Presentation.pdf

HTML 5 ビデオへの字幕ファイルの埋め込み

注意：現在、このオプションは Flash ビデオには使用できません。

HTML 5 の track 要素を使用して、字幕を埋め込むことができます。track 要素は、字幕（SRT）ファイルおよび言語タイプとラベルを src で受け取り、それを字幕オプションで表示します。

この機能は、ブラウザーが HTML 5 の track 要素をサポートする場合にのみ動作します。

例

```
<cfmediaplyer source="myvideo.ogv">
<track kind="subtitle" src="myvideo_en.srt" label="English" srclang="en" />
<track kind="subtitle" src="myvideo_fr.srt" label="French" srclang="fr" />
</cfmedialayer>
```

Flash ビデオの再生リストのサポート

注意：このオプションは、HTML 5 ビデオでは使用できません。

再生する必要のあるメディアのリストは、M3U 形式で再生リストファイルに埋め込まれます。

次のようにして再生リストを指定できます。

```
<cfmediaplayer source="playlist.m3u" />
```

スキニング

ColdFusion 9 には、次のスタイルを使用して（style 属性を使用）、Flash Player にスキンを適用するオプションがあります。bgcolor/bgColorTheme、hideborder、titleColorTheme、controlsColorTheme、progressColorTheme、progressbgColor.。

HTML 5 再生の場合、これらの属性を使用できます。

注意：ColdFusion 10 では、style 属性を使用してプログレスバーおよびコントロールバーにスキンを適用できない場合があります。これは、このリリースで現在使用されている新しいメディアプレーヤーがこれらをサポートしていないためです。

Flash Player では、XML を使用して次のように skin 属性のパスを指定することにより、スキンを制御できます。

```
<cfmediaplayer source="myvideo.mp4" skin="./skin/myskin.xml">
```

注意：XML を使用したスタイルの指定は、Flash ビデオに対してのみ適用されます。

リファレンス

メディアプレーヤーのサンプルスキンを参照するには、次の URL にアクセスしてください。

http://www.rblank.com/2010/10/06/sample-skin-for-strobeflash-media-playback/

プレーヤーのコントロール

ColdFusion 9 は、Flash ビデオの次のメディアプレーヤーコントロールをサポートします。再生／一時停止、停止、プログレスバー、現在および合計の再生時間、ボリューム、フルスクリーン。

停止コントロールは今後はサポートされませんが、ColdFusion 10 での拡張により、HTML 5 ビデオにもこれらのコントロールを使用できます。

さらに、Flash Player を使用するときは、次と前のコントロールがサポートされます。これらは、再生リストを再生すると表示されます。現在、このコントロールは、Flash Player に対してのみ使用でき、指定されているソースが再生リストファイル（M3U）のときに表示されます。

動的ストリーミング

注意：この機能は、Flash Player に対してのみ適用されます。

サーバーサイド

この機能を使用すると、様々な再生ストリーム間で代替することにより、ストリーミングビデオを動的に配信できます。

動的ストリーミングを設定するには、次のように、cfmediaplayer の source 属性を使用して動的ストリーミングビデオを指し示します。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="application/javascript">
functi     on getStreamData()
{
    var player = ColdFusion.MediaPlayer.getPlayer("player_ds");
    var dynamicStreams=player.getStreamItems();
    for (var idx = 0; idx < dynamicStreams.length; idx ++)
    {
        var streamLabel = dynamicStreams[idx]['width'] + "x" + dynamicStreams[idx]['height'] + " @ " +
dynamicStreams[idx]['bitrate'] + "kbps";
        var element = document.createElement("option");
        element.innerHTML = streamLabel;
        document.getElementById("streams").appendChild(element);
        if (idx==player.getCurrentDynamicStreamIndex())
        {
            document.getElementById("streams").selectedIndex = idx;
        }
    }
}
function selectionChange()
{
    var index = document.getElementById("streams").selectedIndex;
    var player = ColdFusion.MediaPlayer.getPlayer("player_ds");
    player.setAutoDynamicStreamSwitch(false);
    player.switchDynamicStreamIndex(index);
}
</script>
</head>
<body>
<div  align="center">
<br />
<br />
<br />
<button type="button" onclick="getStreamData()"  value="Stream Details" name= "Stream Details"> Stream
Details</button>
<br />
<br />
<select name="streams" id="streams" onchange="selectionChange()">
</select>
<br />
</div>
<cfmediaplayer
    name="player_ds"
    source="URL to the F4M file" width=800 height=500 type="flash"
    autoplay="true"
    align="center"
    >
</cfmediaplayer>
</body>
</html>
```

クライアントサイド

クライアントサイドでは、HD コントロールを使用して、使用可能な異なるストリーミングビデオを切り替えることができます。

地理位置情報の表示

cfmap で showUser 属性が指定されている場合、地図上でのユーザーの位置を表示します。この機能は、HTML 5 準拠ブラウザーでのみ動作します。

次の節では、ユーザーの位置を表示するための cfmap タグおよび cfmapitem タグに対する変更について説明します。

cfmap

- 新しい属性 showUser が追加されています。デフォルト値は false です。true の場合、HTML 準拠ブラウザーでは地図上にユーザーの位置が表示されます。

  HTML 5 準拠ではないブラウザーの場合は、アドレスはユーザーが centerAddress に指定した値にフォールバックします。値が指定されていない場合は、centerLatitude および centerLongitude に指定されている値にフォールバックします。

  注意：サイトがユーザーの位置を追跡できるようにするには、ユーザーがサイトを認証する必要があります。例えば、Google Chrome の場合、ユーザーは物理的な位置の追跡を許可するように求められます。

cfmapitem

- 新しい属性 showUser が追加されています。デフォルト値は false です。

  true の場合、HTML 準拠ブラウザーでは地図上にユーザーの位置が表示されます。

cfinput

- type 属性は、HTML 5 のすべての入力タイプ（email、range、date など）をサポートするようになっています。詳しくは、http://dev.w3.org/html5/spec/Overview.html#the-input-element を参照してください。
- このタグは、max や min などの新しい属性をサポートします。

クライアントサイドのチャート作成

ColdFusion 10 はクライアントサイドのチャート作成をサポートします。これは、既存のサーバーサイドのチャート作成機能（従来どおり提供されます）に対する追加です。

クライアントサイドのチャート作成は以下をサポートします。

- **動的および対話形式のチャート作成：**チャートの変更、スタイルの追加、新しい系列またはプロットの追加を行います。
- **適切なフォールバック機能を備えた主要なチャート形式：**HTML 5、Flash、SVG、または VML のチャートを使用します。

  ブラウザーが HTML 5 のチャート作成関連機能をサポートしていない場合、チャートは Flash で表示されます。同様に、Flash がサポートされていない場合、チャートは HTML で表示されます。
- **サーバーサイドのチャート作成と同様の機能：**サーバーサイドのチャート作成機能のほとんどを、クライアントサイドのチャート作成で使用できます。
- **従来のチャートと新しいチャート：**現在のチャートタイプに加えて、新しいチャートセットを提供します。
- **サーバーへのトリップが最小：**すべてのユーザー対話について、サーバーレベルでのチャート生成と比較した場合。

サポートされるチャート

折れ線グラフ	面グラフ	棒グラフ	散布図
バブルチャート	横棒グラフ	円グラフ	レーダーチャート
点グラフ	ネスト円グラフ	ピアノチャート	じょうごグラフ
ゲージ	横点グラフ	円錐	3D 折れ線グラフ
3D 面グラフ	3D 円グラフ	3D 横棒グラフ	角錐
円柱			

クライアントサイドのチャート作成の動作

1 cfchart タグを使用し、version 属性の値として 2 を指定します。

デフォルトのバージョンは 1 です（サーバーサイドのチャート作成に該当）。

2 以前のリリースでのサーバーサイドのチャート作成時の指定と同様に、チャートの詳細を指定します。

制約

次のサーバーサイドのチャート作成機能は、クライアントサイドのチャート作成では使用できません。

- URL へのチャートのリンク
- 変数へのチャートの書き込み

チャート作成の例

例 1

クライアントサイドのチャート作成の基本的な例を次に示します。

```
<cfchart format ="html" type="pie">
    <cfchartseries>
        <cfchartdata item="New car sales" value="50000">
        <cfchartdata item="Used car sales" value="25000">
        <cfchartdata item="Leasing" value="30000">
        <cfchartdata item="Service" value="40000">
    </cfchartseries>
</cfchart>
```

例 2

この例では、zcolumn を指定して簡単なバブルチャートを作成する方法を示します。

```
<cfchart format ="html" type="bubble" query="ChartQuery" showlegend="false">
<cfchartseries query="ProdQuery" type="bubble" itemcolumn="title" valuecolumn="total_days"
zcolumn="rem_days" label="Total_Days">
</cfchart>
```

例 3

これは、ラベルを構造体として指定する例です。

```
<cfchart format ="html" type="bubble" que       ry = " ChartQ ue ry " showlegend="false"
labels=#[{"text"="Hello Label","hook":"node:plot=0,index=2,offset-x=-10,offset-y=-90"}]#>
<cfchartseries type="bubble" label="Total_Days">
<cfchartdata item=1 value=10 zvalue=40>
<cfchartdata item=2 value=20 zvalue=30>
<cfchartdata item=3 value=30 zvalue=20>
<cfchartdata item=4 value=20 zvalue=35>
<cfchartdata item=5 value=40 zvalue=10>
</cfchartseries>
</cfchart>
```

第 10 章：ColdFusion での Flex と AIR の統合

Flash Remoting サービスの使用

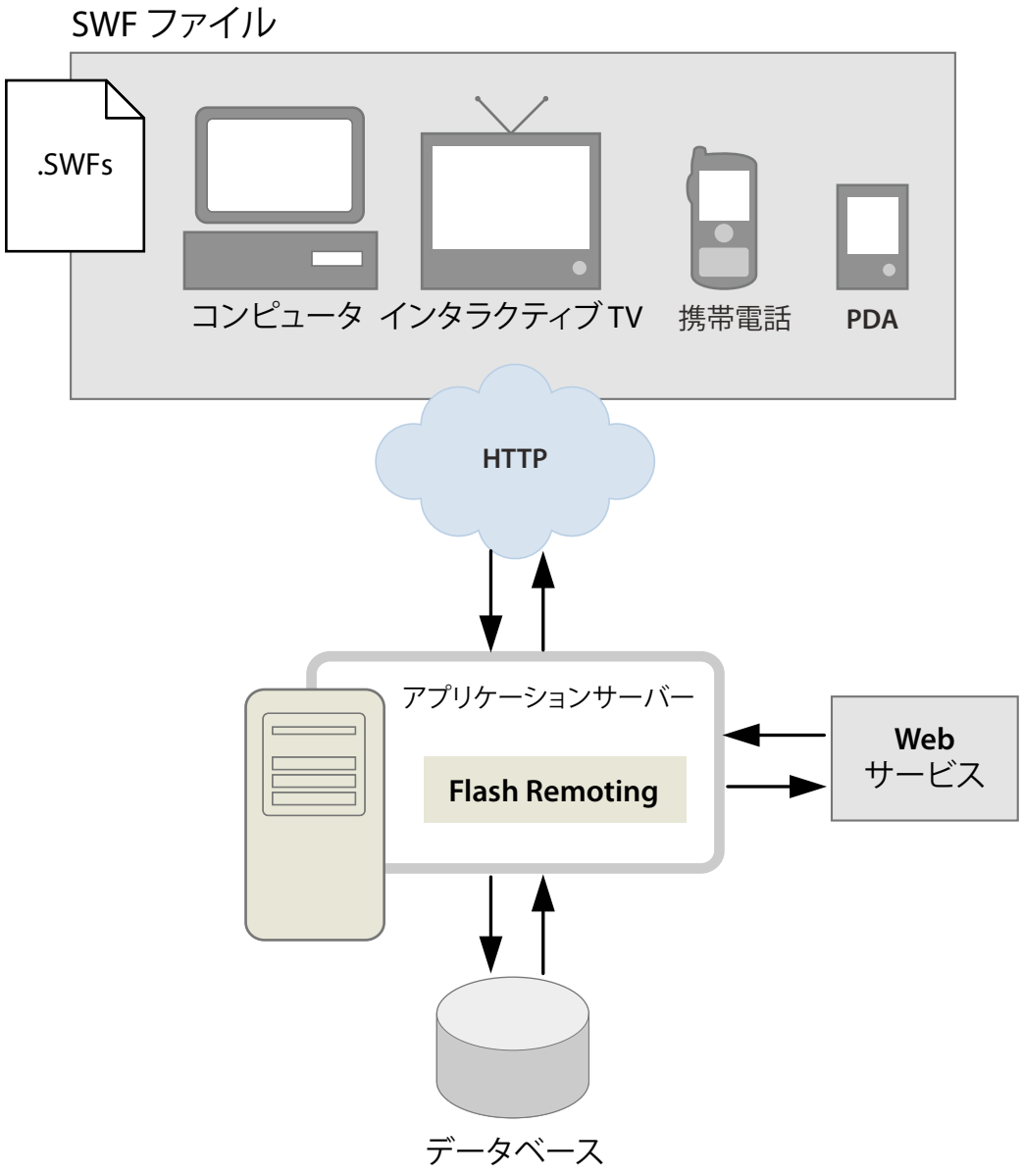
Adobe ColdFusion の Flash Remoting サービスを使用すれば、ColdFusion 開発者と Flash デザイナーが協力して、ColdFusion アプリケーション用のダイナミックな Flash ユーザーインターフェイスを構築できます。

ColdFusion と Flash Remoting の対話を含む、Flash Remoting 機能の詳細については、Flash ヘルプの「**Flash Remoting MX 2004** の使用」および「**Flash Remoting ActionScript Dictionary**」を参照してください。また、Flash Remoting デベロッパーセンター（www.adobe.com/go/learn_cfu_flashremoting_en）では Flash Remoting のマニュアルを参照できます。

ColdFusion での Flash Remoting サービスの使用について

ColdFusion の Flash Remoting サービスを使用すれば、ColdFusion 開発者と Flash MX 2004 デザイナーが協力して、ColdFusion アプリケーション用の Flash ユーザーインターフェイス (UI) を構築できます。Flash UI を構築するには、ユーザーインターフェイスのコードをビジネスロジックコードから分離する必要があります。ユーザーインターフェイスコントロールは Flash で構築し、ビジネスロジックは ColdFusion で作成します。

次の図に、Flash と ColdFusion の関係を簡単に示します。



SWF アプリケーションの計画

Flash UI を使用した ColdFusion アプリケーションの開発を計画するときは、表示コードをビジネスロジックから分離する必要があることを念頭に置いてください。表示コードをビジネスロジックから分離することで、ColdFusion アプリケーションが複数のクライアントタイプ (SWF アプリケーション、Web ブラウザ、Web サービスなど) と対話できるようになります。

複数のクライアントと対話する ColdFusion アプリケーションでは、共通のデータ型 (文字列、整数、クエリーオブジェクト、構造体、配列など) を使用して、ColdFusion ページやコンポーネントから値を返します。結果を受け取るクライアントは、クライアントのタイプに従って、渡されたデータを処理できます。たとえば、Flash ならば ActionScript を、ColdFusion ならば CFML を使用できます。

ColdFusion で Flash Remoting サービスを使用するには、ColdFusion ページやコンポーネントを作成するか、Java オブジェクトをデプロイします。ColdFusion ページでは、Flash 変数スコープを使用して SWF アプリケーションと対話します。CFC (ColdFusion コンポーネント) では、Flash との対話がネイティブでサポートされています。Java オブジェクトのパブリックメソッドも Flash Remoting サービスで使用できます。

Flash Remoting ActionScript API は、ActionScript 2.0 に準拠するように更新されました。ActionScript 2.0 バージョンの API には、次に示す重要な機能が含まれています。

Flash Remoting MX 2004 ActionScript 2.0 API の機能
厳密なデータ型の導入。変数のデータ型を宣言する必要があります。変数に異なるデータ型を代入することはできません。
大文字と小文字の区別の導入。たとえば、myvar と myVar はスペルが異なりますが、ActionScript 1.0 では同じ変数と見なされていました。2.0 では異なる変数と見なされます。
新しい Service クラス。このクラスを使用して、ゲートウェイ接続を作成できると同時に、サービスおよびそのメソッドへの参照を取得できます。このクラスには connection プロパティがあります。これによって接続が返され、リモートサーバーでの承認に使用する資格情報も設定できます。 メモ : NetServices クラスは引き続きサポートされていますが、非推奨です。代わりに、新しい Service および Connection クラスを使用してください。
新しい Connection クラス。これによって、Flash Remoting 接続を作成および使用できます。 メモ : 以前の NetConnection クラスは、Connection クラスに置き換えられました。
新しい PendingCall オブジェクト。Service オブジェクトを使用して実行されるサービスメソッドを呼び出すと返されます。PendingCall オブジェクトには responder プロパティがあります。これを使用して、サービス呼び出しの結果を処理するメソッドを指定します。
新しい RelayResponder クラス。サービス呼び出しの結果または失敗を処理するメソッドを指定します。
新しいプロパティ (columnNames、items、および length)、新しいメソッド (clear()、contains()、editField()、getEditingData()、getIterator()、getLocalLength()、getRemoteLength()、isEmpty()、および sortItems())、および新しい modelChanged イベントを持つ RecordSet オブジェクト。

ActionScript 2.0 Flash Remoting API の詳細については、「Flash Remoting ActionScript Dictionary」ヘルプを参照してください。

Flash Remoting ゲートウェイの設定

ColdFusion の "web.xml" ファイルの次のパラメータは、"gateway-config.xml" ファイルへの Flash Remoting ゲートウェイを参照しています。

```
<init-param>
    <param-name>gateway.configuration.file</param-name>
    <param-value>/WEB-INF/gateway-config.xml</param-value>
        </init-param>
<init-param>
    <param-name>whitelist.configuration.file</param-name>
    <param-value>/WEB-INF/gateway-config.xml</param-value>
</init-param>
<init-param>
    <param-name>whitelist.parent.node</param-name>
    <param-value>gateway-config</param-value>
</init-param>
```

"web.xml" ファイルと "gateway-config.xml" ファイルは両方とも、ColdFusion サーバーの "WEB-INF" ディレクトリに存在します。一般に、この "web.xml" の設定を変更する必要はありません。

ColdFusion MX 7 以降のバージョンの ColdFusion では、以前のリリースの ColdFusion とは異なる方法で Flash ゲートウェイが設定されます。以前のリリースで使用されていたパラメータはサポートされなくなりました。すべての設定パラメータは "gateway-config.xml" ファイルで指定します。また、Flash ゲートウェイでは、ゲートウェイを介してアクセス可

能なリモートソースを指定するホワイトリストがサポートされるようになりました。ホワイトリストを識別する 2 つの "web.xml" エントリで、"gateway-config.xml" ファイルを指定し、gateway-config を親ノードとして指定する必要があります。

"gateway-config.xml" ファイルを編集して、サービスアダプタの設定、ホワイトリストへのサービス名の追加、ロギングレベルの変更、およびゲートウェイでの大文字と小文字の処理方法の指定を行えます。

"gateway-config.xml" ファイルでは、ゲートウェイの次の機能を設定できます。

機能	説明
サービスアダプタ	デフォルトでは、PageableResultSetAdapter、ColdFusionAdapter、CFCAdapter (ColdFusion コンポーネント用)、および CFSSASAdapter (サーバーサイド ActionScript 用) アダプタが ColdFusion で有効になっています。 コメント記号 (<!-- と -->) を削除することにより、JavaBeanAdapter、JavaAdapter、EJBAdapter、ServletAdapter、および CFWSAdapter (Web サービス用) も有効にできます。デフォルトのタグ値として、次のサービスアダプタタグが定義されています。

	`<service-adapters>`
	
	`<adapter>flashgateway.adapter.resultset.PageableResultSetAdapter</adapter>`
	`  <adapter>coldfusion.flash.adapter.ColdFusionAdapter</adapter>`
	`  <adapter>coldfusion.flash.adapter.CFCAdapter</adapter>`
	`  <adapter>coldfusion.flash.adapter.CFSSASAdapter</adapter>`
	`  <!-- <adapter type="stateful-`
	`class">flashgateway.adapter.java.JavaBeanAdapter</adapter> -->`
	`  <!-- <adapter type="stateless-`
	`class">flashgateway.adapter.java.JavaAdapter</adapter> -->`
	`  <!-- <adapter type="ejb">flashgateway.adapter.java.EJBAdapter</adapter> -`
	`->`
	`  <!-- <adapter`
	`type="servlet">flashgateway.adapter.java.ServletAdapter</adapter> -->`
	`  <!-- <adapter>coldfusion.flash.adapter.CFWSAdapter</adapter> -->`
	`</service-adapters>`

セキュリティ	<security> タグの子タグでセキュリティ設定を編集できます。
	特定のアプリケーションサーバーでローカル認証を行うために、<login-command> タグで flashgateway.security.LoginCommand の実装を設定することができます。<login-command> タグは、デフォルトで次の値に設定されています。
	`<login-command>`
	`  <class>flashgateway.security.JRunLoginCommand</class>`
	`  <server-match>JRun</server-match>`
	`</login-command>`
	<show-stacktraces> タグを使用して、スタックトレースを有効にすることができます。スタックトレースは、デバッグや製品サポートに役立ちますが、本番モードではクライアントに送信しないでください。システムに関する内部情報が漏洩する恐れがあります。デフォルトの <show-stacktraces> タグは次のとおりです。
	`<show-stacktraces>false</show-stacktraces>`
	<whitelist> タグでは、ゲートウェイを介してアクセス可能なリモートソースを指定します。すべてに一致することを示すワイルドカードとして、* 文字を使用できます。<whitelist> タグのデフォルト値を次に示します。
	`<whitelist>`
	`  <source>*</source>`
	`</whitelist>`
	アプリケーションをデプロイするときは、必ずこの設定を変更して、アプリケーションを実行するためにゲートウェイからアクセスする必要のあるサービスのみを指定してください。ColdFusion をベースとするサービスの場合、ディレクトリは " パッケージ " として処理されるので、指定するパス (Web ルートから、アクセスを許可するサービスが格納されているディレクトリまたはファイルまでのパス) はピリオドで区切る必要があります。アスタリスク (*) のワイルドカードを使用すれば、特定のディレクトリ内のすべてのサービスにアクセスできるようになります。<source> タグは複数設定できます。
	次のホワイトリストでは、"<Web のルートディレクトリ >/cfdocs/exampleapps/" ディレクトリへのアクセスを許可しています。このディレクトリには、flash1 ～ flash5 の Flash Remoting サンプルディレクトリが含まれています。また、"<Web のルートディレクトリ >/BigApp/remoting" ディレクトリへのアクセスも許可しています。
	`<whitelist>`
	`  <source>cfdocs.exampleapps.*</source>`
	`  <source>BigApp.remoting.*</source>`
	`</whitelist>`

機能	説明
ロガーレベル	ロギングのレベルを、None、Error、Info、Warning、および Debug のいずれかに設定できます。デフォルトのロガーレベルタグは次のとおりです。 `<logger level="Error">coldfusion.flash.ColdFusionLogger</logger>`
リダイレクト URL	<redirect-url> タグを使用して、AMF データで送信されない HTTP リクエストを受け取るための URL を指定できます。デフォルトでは、<redirect-url> タグは {context.root} に設定されています。これは、Web アプリケーションのコンテキストルートです。 `<redirect-url>{context.root}</redirect-url>`
大文字と小文字の区別	<lowercase-keys> タグを使用して、ゲートウェイでの大文字と小文字の処理を指定します。ActionScript 1.0 と ColdFusion では、大文字と小文字を区別しないデータ構造に基づいて、連想配列、オブジェクト、および構造体が保存されます。こうしたデータ型を Java で表す場合は、マッピングを行って大文字と小文字が区別されないようにする必要がありますが、ゲートウェイでは、すべてのキーを小文字に変換することでこのマッピングを実現します。 ActionScript 2.0 では大文字と小文字が区別されるので、<lowercase-keys> タグの値を false に設定する必要があります。 デフォルトの <lowercase-keys> タグは次のとおりです。 `<lowercase-keys>true</lowercase-keys>`

ColdFusion ページでの Flash Remoting サービスの使用

SWF アプリケーションと対話する ColdFusion ページが含まれているディレクトリ名は、ActionScript で呼び出すサービス名になります。このディレクトリに含まれている個別の ColdFusion ページ名は、ActionScript で呼び出すサービス関数名になります。

注意：Flash Remoting では、ColdFusion マッピングを介してアクセスする仮想ディレクトリとは対話できません。

ColdFusion ページでは、Flash 変数スコープを使用して、SWF アプリケーションとやり取りするパラメータにアクセスします。SWF アプリケーションから渡されたパラメータにアクセスするには、パラメータ名の前に Flash スコープ接頭辞を付けるか、Flash.Params 配列を使用します。SWF アプリケーションに値を返すには、Flash.Result 変数を使用します。クエリーオブジェクトのレコードをいくつかに分割して SWF アプリケーションに返す場合は、Flash.Pagesize 変数を使用して、1 分割あたりのレコード数を設定できます。

次の表に、Flash スコープに含まれる変数を示します。

変数	説明	詳細情報
Flash.Params	SWF アプリケーションから渡されたパラメータが含まれている配列。パラメータが渡されなかった場合、Flash.params は存在しますが空になります。	601 ページの「Flash から渡されたパラメータへのアクセス」を参照してください。
Flash.Result	関数を呼び出した SWF アプリケーションに ColdFusion ページから返す変数。 メモ：ActionScript では型の自動変換が行われるので、ColdFusion から Flash にブール値のリテラルを返さないでください。true の場合は 1 を、false の場合は 0 を返してください。	602 ページの「Flash に結果を返す」を参照してください。
Flash.Pagesize	レコードセットをいくつかに分割して SWF アプリケーションに返す場合の、1 分割で返すレコードの数。	603 ページの「レコードを分割して Flash に返す」を参照してください。

次の表に、ColdFusion のデータ型と ActionScript のデータ型の対応関係を示します。

ActionScript のデータ型	ColdFusion のデータ型
Number (プリミティブデータ型)	数値
Boolean (プリミティブデータ型)	ブール値 (0 または 1)
String (プリミティブデータ型)	文字列
ActionScript オブジェクト	構造体
ActionScript オブジェクト (サービス関数に渡される唯一の引数として)	サービス関数の引数。ColdFusion ページ (CFM ファイル): flash 変数スコープ、ColdFusion コンポーネント (CFC ファイル): 名前付き引数
Null	Null (Asc() は 0 を返し、未定義と見なされます)
未定義	Null (Asc() は 0 を返し、未定義と見なされます)
順序配列 **メモ :** ActionScript 配列のインデックスは 0 から始まります (例 : my_ASarray[0])。	配列 **メモ :** ColdFusion 配列のインデックスは 1 から始まります (例 : my_CFarray[1])。
名前付き (連想) 配列	構造体
Date オブジェクト	日付
XML オブジェクト	XML ドキュメント
RecordSet	クエリーオブジェクト (SWF アプリケーションに返す場合のみ。SWF アプリケーションから ColdFusion アプリケーションに RecordSet を渡すことはできません)

また、データ型に関する次の注意事項も検討してください。

- サーバー上の文字列データ型が、ActionScript で有効な数値を表している場合は、Flash によって数値に自動変換されることがあります。
- 複数の独立した値を SWF アプリケーションに返すには、Flash のオブジェクト、配列、または連想配列に変換される複合変数にそれらを格納します。これによって、必要なデータをすべて保持できます。単一の変数を返し、SWF アプリケーションでその要素にアクセスします。

ActionScript における Flash Remoting データの使用方法の詳細については、ヘルプの「Flash Remoting MX 2004 の使用」を参照してください。

Flash から渡されたパラメータへのアクセス

SWF アプリケーションから渡された変数にアクセスするには、パラメータ名に Flash スコープ接頭辞を付けるか、Flash.Params 配列を使用します。配列値にアクセスするには、Flash から渡された値の形式に応じて、順序配列または構造体名のシンタックスを使用します。名前付きのパラメータを渡すのは ActionScript オブジェクトのみです。

たとえば、順序配列として Flash からパラメータが渡された場合は、array[1] で最初の値を参照します。名前付きのパラメータとして渡された場合は、params.name のような標準の構造体名シンタックスを使用します。

ActionScript コレクションに対しては、CFML のほとんどの配列関数や構造体関数を使用できます。ただし、StructCopy CFML 関数は ActionScript コレクションには使用できません。次の表に、ActionScript のさまざまなコレクションと、ColdFusion でそれらのコレクションにアクセスする方法について説明します。

コレクション	ActionScript の例	注意事項

厳密な配列	`var myArray:Array = new Array();`
	`myArray[0] = "zero";`
	`myArray[1] = "one";`
	`myService.myMethod(myArray);`
		Flash Remoting サービスによって、Array 引数が ColdFusion 配列に変換されます。CFML 配列に対する操作はすべて正常に実行できます。
		`<cfset p1=Flash.Params[1][1]>`
		`<cfset p2=Flash.Params[1][2]>`

名前付き配列または連想配列	`var myStruct:Array = new Array();`
	`myStruct["zero"] = "banana";`
	`myStruct["one"] = "orange";`
	`myService.myMethod(myStruct);`
		ActionScript では、名前付き配列のキーに大文字と小文字の区別はありません。
		`<cfset p1=Flash.Params[1].zero>`
		`<cfset p2=Flash.Params[1].one>`

混合配列	`var myMxdArray:Array = new Array();`
	`myMxdArray["one"] = 1;`
	`myMxdArray[2] = true;`
		ColdFusion では、このコレクションを構造体として処理します。ただし、数字で始まるキーは、無効な CFML 変数名です。このデータの取得方法によっては、ColdFusion で例外が発生する場合があります。たとえば、次の CFC メソッドでは例外が発生します。
		`<cfargument name="ca" type="struct">`
		`<cfreturn ca.2>`
		次の CFC メソッドでは例外が発生しません。
		`<cfargument name="ca" type="struct">`
		`<cfreturn ca["2"]>`

ActionScript のオブジェクトイニシャライザを使用した名前付き引数	`myService.myMethod({ x:1, Y:2, z:3 });`	この表記法は、ColdFusion ページに名前付き引数を渡す場合に便利です。ColdFusion ページでは、Flash スコープのメンバーとしてこれらの引数にアクセスできます。
		`<cfset p1 = Flash.x>`
		`<cfset p2 = Flash.y>`
		`<cfset p3 = Flash.z>`
		または、CFC メソッドの標準の名前付き引数としてこれらの引数にアクセスできます。

Flash.Params 配列には、メソッドに渡されたパラメータが順番に保持されています。パラメータを参照するには、次のような標準の構造体名シンタックスを使用します。

```
<cfquery name="flashQuery" datasource="cfdocexamples">
    SELECT ItemName, ItemDescription, ItemCost
    FROM tblItems
    WHERE ItemName EQ '#Flash.paramName#'
</cfquery>
```

この例では、渡された配列の最初のパラメータを参照する Flash.paramName の値によって、クエリー結果をフィルタしています。パラメータが順序配列として SWF アプリケーションから渡された場合は、次のような標準の構造体名シンタックスを使用します。

```
<cfset Flash.Result = "Variable 1:#Flash.Params[1]#, Variable 2: #Flash.Params[2]#">
```

注意：ActionScript の配列インデックスは 0 から始まります。ColdFusion の配列インデックスは 1 から始まります。

Flash に結果を返す

ColdFusion ページから SWF アプリケーションに返されるのは、Flash.Result 変数の値のみです。ColdFusion と Flash との間でサポートされているデータ型の詳細については、600 ページの「ColdFusion ページでの Flash Remoting サービスの使用」のデータ型の表を参照してください。次の手順では、簡単なメッセージが含まれている構造体を SWF アプリケーションに返すサービス関数 helloWorld を作成します。

SWF アプリケーションに構造体を渡す ColdFusion ページの作成

1. Web のルートディレクトリにフォルダを作成し、helloExamples という名前を付けます。
2. ColdFusion ページを作成し、"helloWorld.cfm" という名前を付けて、"helloExamples" ディレクトリに保存します。

3 "helloWorld.cfm" の CFML コードを次のように編集します。

```
<cfset tempStruct = StructNew()>
<cfset tempStruct.timeVar = DateFormat(Now ())>
<cfset tempStruct.helloMessage = "Hello World">
```

この例では、2 つの文字列変数を構造体に追加しています。1 つは形式設定された日付が含まれている変数で、もう 1 つは簡単なメッセージが含まれている変数です。この構造体は、Flash.Result 変数を使用して SWF アプリケーションに返します。

```
<cfset Flash.Result = tempStruct>
```

4 ファイルを保存します。

ディレクトリ名がサービスアドレスになることに注意してください。"helloWorld.cfm" ファイルは、helloExamples という Flash Remoting サービスのメソッドです。次に示す ActionScript の例では、helloWorld ColdFusion ページを呼び出して、返された値を表示します。

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;

// Connect to helloExamples service and create the howdyService service object
    var howdyService:Service = new Service(
            "http://localhost/flashservices/gateway",
            null,
            "helloExamples",
            null,
            null );
// Call the service helloWorld() method
var pc:PendingCall = howdyService.helloWorld();
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "helloWorld_Result", "helloWorld_Fault" );

function helloWorld_Result(re:ResultEvent)
{
    // Display successful result
    messageDisplay.text = re.result.HELLOMESSAGE;
    timeDisplay.text = re.result.TIMEVAR;
}

function helloWorld_Fault(fe:FaultEvent)
{
    // Display fault returned from service
    messageDisplay.text = fe.fault;
}
```

注意： ActionScript では型の変換が自動的に行われるので、ブール値のリテラルを ColdFusion から Flash に返さないでください。true の場合は 1 を、false の場合は 0 を返してください。

レコードを分割して Flash に返す

ColdFusion では、レコードセットの結果をいくつかに分割して Flash に返すことができます。たとえば、クエリーで 20 のレコードが返された場合は、Flash.Pagesize 変数を設定することで、一度に 5 つのレコードを Flash に返すことができます。レコードセットを分割すれば、アプリケーションサーバーのデータが SWF アプリケーションにロードされるまでの待機時間を抑えることができます。

レコードセットを分割して Flash に返す ColdFusion ページの作成

1 ColdFusion ページを作成し、"getData.cfm" という名前を付けて、"helloExamples" ディレクトリに保存します。

2 "getData.cfm" のコードを次のように編集します。

```
<cfparam name="pagesize" default="10">
<cfif IsDefined("Flash.Params")>
    <cfset pagesize = Flash.Params[1]>
</cfif>
<cfquery name="myQuery" datasource="cfdocexamples">
    SELECT *
    FROM tblParks
</cfquery>
<cfset Flash.Pagesize = pagesize>
<cfset Flash.Result = myQuery>
```

SWF アプリケーションからパラメータが渡された場合は、pagesize 変数に Flash.Params[1] 変数の値を代入しています。それ以外の場合、変数の値はデフォルトである 10 に設定されます。次に、cfquery ステートメントでデータベースにクエリーを行っています。クエリーの後、Flash.Pagesize 変数に pagesize 変数を代入しています。最後に、クエリーの結果を Flash.Result 変数に代入して、SWF アプリケーションに返しています。

3 ファイルを保存します。

Flash.Pagesize 変数に値を指定すると、その値よりも多くのレコードを持つレコードセットがページングされ、指定した数のレコードが返されるようになります。たとえば次のコードでは、CFMService の getData() 関数を呼び出し、最初のパラメータを使用してページサイズ 5 を要求します。

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;
// Connect to helloExamples service and create the CFMService service object
    var CFMService:Service = new Service(
            "http://localhost/flashservices/gateway",
            null,
            "helloExamples",
            null,
            null );
// Call the service getData() method
var pc:PendingCall = CFMService.getData(5);
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "getData_Result", "getData_Fault" );

function getData_Result(re:ResultEvent)
{
    // Display successful result
    DataGlue.bindFormatStrings(employeeData, re.result, "#PARKNAME#, #CITY#, #STATE#");
}
function getData_Fault(fe:FaultEvent)
{
    // Display fault returned from service
    trace("Error description from server: " + fe.fault.description);
}
```

この例の employeeData は Flash リストボックスです。結果ハンドラである getData_Result では、employeeData リストボックスの PARKNAME、CITY、および STATE 列を表示しています。レコードの最初の分割が取得された後は、RecordSet ActionScript クラスによって、レコードの取得タスクが処理されます。この場合、ユーザーがリストボックスをスクロールすると、リストボックスによって次の分割が要求されます。

レコードを取得する方法は、クライアントサイドの RecordSet オブジェクトの RecordSet.setDeliveryMode ActionScript 関数で設定できます。

Flash での CFC の使用

SWF アプリケーションで CFC を使用する場合、CFC に変更を加える必要はほとんどありません。cffunction タグでメソッド名を指定し、CFML ロジックを定義します。cfargument タグで引数名を指定します。cfreturn タグで SWF アプリケーションに結果を返します。CFC ファイル (*.cfc) の名前が、ActionScript のサービス名になります。

注意：SWF アプリケーションから CFC メソッドを呼び出せるようにするには、cffunction タグの access 属性を remote に設定します。

次の例では、前に ColdFusion ページとして実装した helloWorld 関数を、CFC で実装します。詳細については、600 ページの「ColdFusion ページでの Flash Remoting サービスの使用」を参照してください。

SWF アプリケーションと対話する CFC の作成

1 CFC を作成し、"flashComponent.cfc" という名前を付けて、"helloExamples" ディレクトリに保存します。

2 "flashComponent.cfc" のコードを次のように編集します。

```
<cfcomponent name="flashComponent">
    <cffunction name="helloWorld" access="remote" returnType="Struct">
        <cfset tempStruct = StructNew()>
        <cfset tempStruct.timeVar = DateFormat(Now ())>
        <cfset tempStruct.helloMessage = "Hello World">
        <cfreturn tempStruct>
    </cffunction>
</cfcomponent>
```

この例では、helloWorld 関数が作成されます。cfreturn タグで SWF アプリケーションに結果を返しています。

3 ファイルを保存します。

helloWorld サービス関数が、flashComponent サービスを介して ActionScript で使用できるようになりました。次の ActionScript の例では、この関数を呼び出します。

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;

// Connect to the Flash component service and create service object
    var CFCService:Service = new Service(
            "http://localhost/flashservices/gateway",
            null,
            "helloExamples.flashComponent",
            null,
            null );
// Call the service helloWorld() method
var pc:PendingCall = CFCService.helloWorld();
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "helloWorld_Result", "helloWorld_Fault" );

function helloWorld_Result(re:ResultEvent)
{
    // Display successful result
    messageDisplay.text = re.result.HELLOMESSAGE;
    timeDisplay.text = re.result.TIMEVAR;
}

function helloWorld_Fault(fe:FaultEvent)
{
    // Display fault returned from service
    messageDisplay.text = fe.fault;
}
```

この例の CFCService オブジェクトは、"helloExamples" ディレクトリの flashComponent コンポーネントを参照しています。この例で helloWorld 関数を呼び出すと、flashComponent で定義された関数が実行されます。

ColdFusion コンポーネントの場合、コンポーネントファイル名 (Web ルートディレクトリからのディレクトリ構造を含む) がサービス名として使用されます。パスディレクトリは、円記号 (¥) ではなくピリオド (.) で区切ります。

ColdFusion Java オブジェクトでの Flash Remoting サービスの使用

ColdFusion では、JavaBeans、Java クラス、Enterprise JavaBeans などの、さまざまな Java オブジェクトを実行できます。ColdFusion Administrator を使用して、クラスパスにディレクトリを追加できます。

ColdFusion クラスパスへのディレクトリの追加

1. ColdFusion Administrator を開きます。
2. [サーバーの設定] メニューの [Java と JVM] リンクをクリックします。
3. [クラスパス] フォームフィールドにディレクトリを追加します。
4. 「変更の送信」をクリックします。
5. ColdFusion を再起動します。

クラスパスに Java ファイルを保存すると、クラスインスタンスのパブリックメソッドが SWF ムービーで使用できるようになります。

たとえば、utils.UIComponents という Java クラスが ColdFusion クラスパスのディレクトリにあるとします。この Java ファイルには次のコードが含まれています。

```
package utils;
public class UIComponents
{
    public UIComponents()
    {
    }
    public String sayHello()
    {
        return "Hello";
    }
}
```

注意：Flash Remoting でコンストラクタを呼び出すことはできません。デフォルトコンストラクタを使用してください。

ActionScript では、次の javaService 呼び出しにより、utils.UIComponents クラスの sayHello パブリックメソッドが実行されます。

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;

// Connect to service and create service object
    var javaService:Service = new Service(
            "http://localhost/flashservices/gateway",
            null,
            utils.UIComponents",
            null,
            null );
// Call the service sayHello() method
var pc:PendingCall = javaService.sayHello();
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "sayHello_Result", "sayHello_Fault" );

function sayHello_Result(re:ResultEvent)
{
    // Display successful result
    trace("Result is: " + re.result);
}

function sayHello_Fault(fe:FaultEvent)
{
    // Display fault returned from service
    trace("Error is: " + fe.fault.description);
}
```

注意：ColdFusion での Java オブジェクトの使用方法の詳細については、1119 ページの「Java オブジェクトの使用」を参照してください。

ColdFusion および Flash でのエラーの処理

デバッグを簡単に行うには、ColdFusion ページまたはコンポーネントで cftry および cfcatch タグを使用して、Flash Player にエラーメッセージを返します。たとえば、ColdFusion ページ "causeError.cfm" に次のコードを設定します。

```
<cftry>
    <cfset dev = Val(0)>
    <cfset Flash.Result = (1 / dev)>
    <cfcatch type = "any">
        <cfthrow message = "An error occurred in this service: #cfcatch.message#">
    </cfcatch>
</cftry>
```

この例の 2 番目の cfset タグで 0 による除算が要求され、エラーが発生します。cfthrow タグの message 属性でエラーの説明を提供しています。この属性は ColdFusion から SWF アプリケーションに返されます。

SWF アプリケーションでエラーを処理するには、causeError_Fault に似た次のエラーハンドラを作成します。

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;

// Connect to service and create service object
    var CFMService:Service = new Service(
            "http://localhost/flashservices/gateway",
            null,
            "helloExamples",
            null,
            null );
// Call the service causeError() method
var pc:PendingCall = CFMService.causeError();
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "causeError_Result", "causeError_Fault" );

function causeError_Result(re:ResultEvent)
{
    // Display successful result
    messageDisplay.text = re.result;
}

function causeError_Fault(fe:FaultEvent)
{
    // Display fault returned from service
    trace("Error message from causeError is: " + fe.fault.description);
}
```

この例により、causeError_Fault 関数から返されたトレースメッセージが、Flash 出力パネルに表示されます。メッセージを構成する fe.fault.description は、"causeError.cfm" ページの #cfcatch.message# に含まれるメッセージに該当します。

注意：Flash と対話する ColdFusion ページを作成する場合は、Flash から利用する前に、ColdFusion ページが正常に動作することを確認してください。

Flash Remoting Update の使用

Flash Remoting Update を使用すると、Adobe Flash Builder や以前のバージョンの Flex Builder に Adobe ColdFusion を組み合わせて、ColdFusion の高度なデータ取得機能 (cfpop、cfldap、cfquery タグなど) を搭載したリッチインターネットアプリケーションを作成できます。また、サーバーコールバックやカスタムユーザーインターフェイスなどの機能を備えた Flash フォームや SWF ファイルも作成できます。

Flash Remoting Update を使用するための前提条件

Flash Remoting Update は、ColdFusion でサポートされているすべてのプラットフォームのどの設定 (サーバー、マルチサーバー、J2EE) の ColdFusion でも利用できます。

Flash Remoting Update を使用するには、次の製品がインストールされている必要があります。

- Flex 2 SDK 以降、Flex Builder 2 以降、Flash Builder 4。
- Flash Player 8.5 以降

Flex のコンパイルの設定

Flash Builder または Flex SDK を使用して、Flex アプリケーションを SWF ファイルにコンパイルできます。Flash Remoting Update を使用するには、MXML をコンパイルするときに、これらのプログラムで ColdFusion の "services-config.xml" ファイルを使用する必要があります。

ColdFusion の設定ファイルを使用するように Flash Builder を設定するか、SDK を使用してアプリケーションをコンパイルするときにファイルを指定する必要があります (アプリケーションのコンパイルおよび実行を参照)。

ColdFusion の設定ファイルを使用する Flex Builder 2 の設定

Flex Builder プロジェクトセットアップウィザードを使用して、サーバータイプとして ColdFusion を選択すると、"services-config.xml" ファイルを使用するように Flex Builder が設定されます。プロジェクトを設定する手順は、次のとおりです。

1. [File]-[New]-[Flex Project] を選択し、新規 Flex プロジェクトウィザードを開きます。次に、[Create a Flex project] ページの最初のセクションで、適切な情報を入力します。
2. 次のいずれかのラジオボタンを選択します。
   - Flex Builder でコンパイルする場合は、[ColdFusion Flash Remoting] を選択します。
   - ColdFusion とともに LiveCycle Data Services をインストール済みで、メッセージングまたはデータ管理を使用する場合は、[Flex Data Services] を選択します。
3. [Flex Data Services] を選択した場合は、アプリケーションを Flex Builder でローカルにコンパイルするか、またはページが表示されているアプリケーションサーバー上でコンパイルするかを選択します。デプロイしたコードはサーバー上でコンパイルしないでください。このオプションは、開発目的でのみ使用します。
4. [Next] をクリックしてプロジェクトの作成を完了し、[Finish] をクリックします。

最初の [Create a Flex Project] ページで [Basic] を選択し、後でアプリケーションをコンパイルして ColdFusion で使用する場合は、次のように Flex Builder を手動で設定します。

1. [Project]-[Properties] を選択します。
2. [Properties] ダイアログボックスの右ペインで、[Flex Complier] を選択します。
3. [Additional Compiler arguments] で、ローカルの ColdFusion にある "services-config.xml" ファイルへの絶対パスの前に -services= を追加します。たとえば、デフォルトのスタンドアローン ColdFusion を使用する Windows システムでは、次のように引数文字列を指定します。

```
-services=C:/ColdFusion9/wwwroot/WEB-INF/flex/services-config.xml
```

ColdFusion の設定ファイルを使用する Flex Builder 3 の設定

Flex Builder プロジェクトセットアップウィザードを使用して、サーバータイプとして ColdFusion を選択すると、"services-config.xml" ファイルを使用するように Flex Builder が設定されます。プロジェクトを設定する手順は、次のとおりです。

1. [File]-[New]-[Flex Project] を選択し、新規 Flex プロジェクトウィザードを開きます。次に、[Create a Flex project] ページの最初のセクションで、適切な情報を入力します。
2. [Create a Flex project] ページの [Server technology] セクションで、アプリケーションサーバータイプとして [ColdFusion] を選択し、[Use remote object access service] を選択します。
3. 次のいずれかのラジオボタンを選択します。
   - Flex Builder でコンパイルする場合は、[ColdFusion Flash Remoting] を選択します。
   - LiveCycle Data Services がインストールされていて、サーバー上でアプリケーションをコンパイルする場合は、Flex Builder 3 で [LiveCycle Data Services] を選択します。

4 [Next] をクリックして [Configure ColdFusion] ページを開き、必要な情報を入力します。手順 3 で [LiveCycle Data Services] を選択した場合は、アプリケーションをローカルにコンパイルするか、サーバー上でコンパイルするかを選択できます。アプリケーションの開発時のみ、サーバー上でコンパイルすることをお勧めします。デプロイしたコードをサーバー上でコンパイルしないでください。これは、ユーザーが要求するまで MXML ページは SWF ファイルにコンパイルされず、コンパイラで HTML ラッパーページが作成されないためです。

5 [Finish] をクリックして設定を完了します。

[Create a Flex project] ページの [Server technology] セクションで [ColdFusion] を選択しなかった場合、後でアプリケーションをコンパイルして ColdFusion で使用するには、次のように Flex Builder を手動で設定します。

1 [Project]-[Properties] を選択します。

2 [Properties] ダイアログボックスの右ペインで、[Flex Complier] を選択します。

3 [Additional Compiler arguments] で、ローカルの ColdFusion にある "services-config.xml" ファイルへの絶対パスの前に -services= を追加します。たとえば、デフォルトのスタンドアローン ColdFusion を使用する Windows システムでは、次のように引数文字列を指定します。

```
-services=C:/ColdFusion8/wwwroot/WEB-INF/flex/services-config.xml
```

CFC の指定

接続先の ColdFusion コンポーネント (CFC) を指定するには、次のいずれかを行います。

- MXML で、Web ルートからの CFC のパスをドット区切りで指定します。
- CFC の名前付きリソースを作成します。このリソースの作成は、データソースの登録に似ています。このリソース名を XML で使用します。

MXML での CFC の指定

MXML で CFC を指定するには、次のようなコードを使用します。

```
<mx:RemotObject
    id="myCfc"
    destination="ColdFusion"
    source="myApplication.components.User"/>
```

ColdFusion 9 では、ColdFusion に対するメッセージングサポートを可能にする BlazeDS がサポートされています。ColdFusion をインストールすると、次のファイルが "/WEB-INF/flex" ディレクトリに追加されます。

- "remoting-config.xml"
- "messaging-config.xml"
- "services-config.xml"
- "proxy-config.xml"

"remoting-config.xml" では、destination 属性にあらかじめ ColdFusion が設定されています。destination 属性に対するデフォルトの source 値は、ワイルドカード * です。ColdFusion 9 における Flash Remoting 関連の変更点については、ColdFusion 9 および ColdFusion 9.0.1 における Flash Remoting 用の XML 設定ファイルの変更点を参照してください。

構成ファイルに他の有効な接続先を設定している場合は、destination 属性に "ColdFusion" を設定する必要はありません。この場合、destination 定義では source 要素の値として * を指定する必要があります。"remoting-config.xml" で source 属性に * 以外の値が指定されている場合は、MXML ファイルの source 属性に指定された値よりもその値が優先されます。

接続先の定義の詳細については、CFC の名前付きリソースの作成を参照してください。

CFC の名前付きリソースの作成

1 "WEB-INF/flex/remoting-config.xml" ファイルを編集し、CFC の接続先エントリを追加します。たとえば、次のように編集します。

```
<service id="remoting-service" class="flex.messaging.services.RemotingService"
messageTypes="flex.messaging.messages.RemotingMessage">
    <adapters>
    <adapter-definition id="cf-object"
    class="coldfusion.flash.messaging.ColdFusionAdapter" default="true" />
    <adapter-definition id="java-object"
    class="flex.messaging.services.remoting.adapters.JavaAdapter" />
    </adapters>

        <default-channels>
        <channel ref="my-cfamf" />
        </default-channels>
            <destination id="ColdFusion">
        <channels>

        <channel ref="my-cfamf" />
        </channels>
        <properties>
        <source>*</source>
        </properties>
    </destination>
</service>
```

source 属性には、Web ルートからの CFC のパス (CFC のクラスパス) をドット表記で指定します。

channel-ref タグは "services-config.xml" ファイルの channel-definition を参照しています。前の例では、次のように、my-cfamf channel-definition が参照されています。

```
<channel-definition id="my-cfamf" class="mx.messaging.channels.AMFChannel"> <endpoint
uri="http://{server.name}:{server.port}{context.root}/flex2gateway/"
class="coldfusion.flash.messaging.CFAMFEndPoint" />
    <properties>
        <polling-enabled>false</polling-enabled>
        <serialization>
            <enable-small-messages>false</enable-small-messages>
        </serialization>
        <coldfusion>
            <access>
                <use-mappings>true</use-mappings>
                <method-access-level>remote</method-access-level>
            </access>
            <use-accessors>true</use-accessors>
            <use-structs>false</use-structs>
            <property-case>
                <force-cfc-lowercase>false</force-cfc-lowercase>
                <force-query-lowercase>false</force-query-lowercase>
                <force-struct-lowercase>false</force-struct-lowercase>
            </property-case>
        </coldfusion>
    </properties>
</channel-definition>
```

2 ColdFusion サーバーを再起動します。

次の表で、"remoting-config.xml" の XML 属性について説明します。

要素	説明
destination id	CFC にアクセスするために MXML の mx:RemoteObject タグで指定する必要がある destination 属性
channels	1 つまたは複数の子 channel 属性のコンテナ。ColdFusion サーバーへのアクセスに使用する AMF チャネルを指定します。
channel-ref	"services-config.xml" ファイルで指定された channel-definition id を参照します。
source	<ColdFusion の Web ルートディレクトリ > からの CFC のドット区切りファイルパス、または、ColdFusion Administrator の [マッピング] ページのエントリ (use-mappings プロパティが true の場合)
access	ColdFusion サーバー上で CFC にアクセスする方法を制御するプロパティ

次の表に、"services-config.xml" の XML 属性を示します。

要素	説明
channel-definition id	チャネル定義
coldfusion	アクセスレベル、CFC を検索するためのマッピング、パブリックメソッドまたはリモートメソッドへのアクセスを設定するタグが含まれます。
access	呼び出す CFC の解決ルールおよびアクセスレベルを定義します。
use-mappings	ColdFusion マッピングを使用して CFC を検索します。デフォルトでは Web ルートディレクトリの下にある CFC ファイルのみを検索できます。
method-access-level	" パブリックおよびリモート " または " リモート " メソッドの呼び出しを許可します。
use-accessors	値オブジェクト CFC が getter と setter を持つかどうかを指定します。値オブジェクト CFC に getter および setter が存在する場合は、use-accessors の値を true に設定します。
use-structs	ActionScript から CFC への変換を行わないようにする場合は、use-structs の値を true に設定します。値が false の場合でも、アセンブラから Flex に構造体を返すことは可能です。デフォルト値は false です。

`force-cfc-lowercase`
`force-query-lowercase`
`force-struct-lowercase`
	ActionScript への変換時に、プロパティ名、クエリー列名、構造体キーを小文字にするかどうかを指定します。クエリー列名は、対応する ActionScript 変数と大文字小文字が完全に一致している必要があります。デフォルト値は false です。

CFC の使用

アプリケーションでの CFC の使用

1 MXML ファイルでは、<mx:RemoteObject> タグを使用して名前付きの CFC リソースに接続します。この接続を使用して、CFC の任意のリモートメソッドを呼び出せます。

2 "remoting-config.xml" ファイルで CFC の宛先を作成した場合は、mx:RemoteObject タグでその宛先名を指定します。次に例を示します。

```
<mx:RemoteObject
    id="a_named_reference_to_use_in_mxml"
    destination="CustomID"
    result="my_CFC_handler(event)"/>
```

CFC の接続先を作成していない場合は、mx:RemoteObbject タグの destination 属性に "ColdFusion" を指定し、CFC パスを指定します。次に例を示します。

```
<mx:RemotObject
    id="myCfc"
    destination="ColdFusion"
    source="myApplication.components.User"/>
```

3 CFC メソッドを呼び出します。次に例を示します。

```
<mx:Button label="reload" click="my_CFC.getUsers()"/>
```

この例では、ユーザーがボタンを押したときに、Click イベントによって CFC メソッド getUsers が呼び出されます。

4 <mx:RemoteObject> タグで指定する、CFC メソッドの結果を処理するためのハンドラを定義します。次に例を示します。

```
private function my_CFC_handler( event:ResultEvent )
{
// Show alert with the value that is returned from the CFC.
mx.controls.Alert.show(ObjectUtil.toString(event.result));
}
```

アプリケーションのコンパイルおよび実行

Flash Builder を使用して、または Flash Builder を使用せずに、アプリケーションをコンパイルおよび実行できます。どちらの方法を使用するかどうかは、LiveCycle Data Services をインストールしたかどうかによって決まります。

Flash Builder を使用したアプリケーションのコンパイルおよび実行

Flash Builder を使用してアプリケーションをコンパイルおよび実行するには、609 ページの「Flex のコンパイルの設定」の説明に従って Flash Builder を設定する必要があります。通常は、アプリケーションをコンパイルして SWF ファイルを作成します。Flash Builder プロジェクトを設定するときは、プロジェクトを格納する場所を指定できます。デフォルトでは、Web ルートの下に SWF ファイルおよび HTML ラッパーページが格納されます。これにより、ブラウザで SWF ファイルの HTML ラッパーをリクエストするなどして、アプリケーションを適切に実行できます。

Flash Builder を使用しないアプリケーションのコンパイルおよび実行

SDK を使用してアプリケーションを直接コンパイルするには、ColdFusion の "services-config.xml" ファイルを使用するように Flex コンパイラを設定します。Flex コンパイラを設定するには、ローカルの ColdFusion にある "services-config.xml" ファイルへの絶対パスの前に mxml コマンドライン "-services=" を追加します。たとえば、デフォルトのスタンドアローン ColdFusion を使用する Windows システムでは、次のように引数文字列を指定します。

```
-services=C:/ColdFusionCentuar/wwwroot/WEB-INF/flex/services-config.xml
```

クライアントとサーバー間での遅延ロード

このリリースでは、ColdFusion ORM をバックエンドで使用し、Flex をフロントエンドで使用する、アプリケーションの関連エンティティの必要に応じた読み込みがサポートされています。アプリケーションでは、メインエンティティを取得し、関連エンティティは取得しないようにすることができます。クライアントアプリケーションが関連エンティティにアクセスを試みたときにのみ、エンティティが読み込まれます。

例

Maria は ColdFusion ORM を使用して、Flex アプリケーションを作成しています。彼女は、会社の全従業員と各従業員が関わっているプロジェクトのリストを作成しようとしています。遅延ロードを使用すると、ユーザーは、最初に従業員の情報と、従業員が携わっているプロジェクトの数を取得できます。その後、特定のプロジェクトをクリックすると、特定の従業員のプロジェクト情報を読み込むことができます。アプリケーションを読み込んだときに最初にすべてのデータを読み込む必要はありません（これは ColdFusion 9 での動作でした）。

遅延ロードの設定

遅延ロード拡張機能を利用するには、クライアントサイドとサーバーサイドで様々な設定を行う必要があります。

サーバーサイドの設定

1 services-config.xml で、channel-definition（使用するチャネル）を指定しているセクションに移動し、次の行を探します。

*<serialize-array-to-arraycollection>**false***

2 この値を true に変更します。

3 （オプション）関連エンティティが読み込まれるときに呼び出されるメソッドの名前を変更する必要がある場合は、メソッドの名前を loadProxy から変更します。

*<proxy-load-method>**loadProxy**</proxy-load-method>*

4 次のサンプルコードで示されているように、loadProxy メソッドをサービスの CFC に追加します。サービス CFC は、ORM コンポーネントと同じディレクトリにあります。

service.cfc

```
component {
remote function loadProxy(any proxyKey, any fullyqualifiedname)
    {
    //writeDump(var="#proxyKey#,,,#fullyqualifiedname#",output="console" );
    writedump(var="#proxykey#",output="console");
    if(fullyqualifiedname contains "cproducts"){
    return EntityLoadByPK("cproducts",proxykey );
    }else{
    return EntityLoadByPK("ccategories",proxyKey );
    }
    }
}
```

loadProxy メソッドは、proxyKey（エンティティインスタンスのプライマリキー）およびエンティティの fullyqualifiedname を引数として取得します。

dphibernate によって送られた fullyqualifiedname が調べられ、必要なオブジェクト（この場合は cproducts または ccategories）が返されます。

5 remotingFetch を lazy に設定します。

lazy は、cfproprety タグの remtoingFetch に（true および false に加えて）追加された新しい値です。

- true：プロパティの値は、Flash Remoting によって Flash に送信されます。
- false：Null が Flash に送信されます。
- lazy：関連エンティティのプロキシオブジェクトが、プライマリキーの値のみと共に送信されます。

  プロキシオブジェクトのいずれかのプロパティがアクセスされたときにのみ、プライマリキーが渡された CFC で定義されている読み込みメソッドに、別のリモート呼び出しが到達します。読み込みメソッドは、すべての値を設定してオブジェクトを返します。

サンプル設定

employee.cfc

```
<cfcomponent persistent="true" table="employees">
    <cfproperty name="employeeID" fieldtype="id" generator="native"/>
    <cfproperty name="personalObj" fieldtype="one-to-one" cfc="cpersonal" cascade="all"
                remotingfetch="lazy"/>
    <cfproperty name="lastName"/>
    <cfproperty name="firstName"/>
</cfcomponent>
```

personal.cfc

```
<cfcomponent persistent="true" table="personal" >
    <cfproperty name="personalID" generator="foreign" params="{property=EmployeeObj}">
    <cfproperty name="employeeObj" fieldtype="one-to-one" cfc="employee constrained="true">
    <cfproperty name="SSN">
    <cfproperty name="fatherName">
</cfcomponent>
```

service.cfc

```
component {
remote function loadProxy(any proxyKey, any fullyqualifiedname)
    {
    //writeDump(var="#proxyKey#,,,#fullyqualifiedname#",output="console" );
    writedump(var="#proxykey#",output="console");
    if(fullyqualifiedname contains "cproducts"){
    return EntityLoadByPK("cproducts",proxykey );
    }else{
    return EntityLoadByPK("ccategories",proxyKey );
    }
    }
}
```

クライアントサイドの設定

1 dpHibernate.swc を Flex プロジェクトに追加します。このファイルは /CFIDE/scripts/ ディレクトリにあります。

2 デフォルトの RemoteObject の代わりに HibernateRemoteObject を使用して、リモート呼び出しを行います。

3 HibernateManaged.defaultHibernateService をリモートオブジェクトインスタンスに設定します。

dpHibernate.swc はこのリモートオブジェクトインスタンスを使用して、遅延ロードの読み込み呼び出しをサーバーに対して行います。

4 次のことを確認します。

   1 ActionScript クラスが、RemoteClass の alias 属性を使用して CFC にマップされています。

   次に例を示します。

   [RemoteClass(alias="orm.employees")]

   2 ActionScript クラスは、org.dphibernate.core.HibernateBean を拡張しています。

   3 管理対象メタデータが ActionScript クラスに追加されています。

4 Flash Remoting を実行してエンティティを取得します。

関連エンティティは、クライアントで明示的にアクセスしたときにのみ読み込まれます。

例

Category.as

```
package model.beans
{
      import mx.collections.ArrayCollection;

      import org.dphibernate.core.HibernateBean;

      [RemoteClass(alias="lazyloading.o2m.ccategories")]
      [Managed]
      public class Category extends org.dphibernate.core.HibernateBean
      {
            public function Category()
            {
            }

            public var categoryID:Number;
            public var categoryName:String;
            public var description:String;
            public var products:ArrayCollection;
      }
}
Product.as
package model.beans
{
      import mx.collections.ArrayCollection;

      import org.dphibernate.core.HibernateBean;
      [RemoteClass(alias="lazyloading.o2m.cproducts")]
      [Managed]
      public class Product extends org.dphibernate.core.HibernateBean
      {
            public function Product()
            {
            }

            public var productID:Number;
            public var productName:String;
            public var categoryID:Category;
      }
}
LazyLoading.MXML
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                     xmlns:s="library://ns.adobe.com/flex/spark"
                     xmlns:mx="library://ns.adobe.com/flex/mx"
                     xmlns:dp="org.dphibernate.rpc.*"
                     applicationComplete="onAppComplete(event)" minWidth="955" minHeight="600"
xmlns:dphibernate="http://www.dphibernate.org/2010/mxml">

      <fx:Declarations>
            <!-- Place non-visual elements (e.g., services, value objects) here -->
            <dp:HibernateRemoteObject destination="ColdFusion"
source="newmanual.apollounit.orm.lazyloading.o2m.service"
                                                endpoint="http://localhost:8500/flex2gateway/lazyloading"
id="BasicService"
                                                  result="BasicService_resultHandler(event)"
fault="BasicService_faultHandler(event)">
            </dp:HibernateRemoteObject>
      </fx:Declarations>
      <fx:Script>
            <![CDATA[
                  import model.beans.*;
```

```
import mx.binding.utils.BindingUtils;
import mx.collections.ArrayCollection;
import mx.rpc.AsyncToken;
import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;

import org.dphibernate.rpc.HibernateManaged;

[Bindable]
public var categories:ArrayCollection;
[Bindable]
public var ProdsList:ArrayCollection;
[Bindable]

public var displaytext:String;
public var category:Category;
public var product:Product;
public var employee:Employee;
public var employee_self123:Employee_self;

protected function BasicService_resultHandler(event:ResultEvent):void
{
      if( event.token.operation == "getCategories"){
            categories = event.result as ArrayCollection;
            var len:Number = categories.length;
            allintext.text = event.result.length;
      }else if(event.token.operation == "getProducts"){
            ProdsList = event.result as ArrayCollection;
      }
}


protected function BasicService_faultHandler(event:FaultEvent):void
{
      allintext.text="There is an error";
}

protected function onAppComplete(event:Event):void
{
      HibernateManaged.defaultHibernateService=BasicService;
      //var token:AsyncToken = BasicService.getCategories();
      //token.operation="getCategories";
}

protected function load_clickHandler(event:MouseEvent):void
{
      var token:AsyncToken = BasicService.getCategories();
```

```
                        token.operation="getCategories";
                  }
            ]]>
      </fx:Script>
      <s:TextArea x="10" y="39" id="allintext" text="hello"/>
      <s:Button x="240" y="10" label="Load Categories" id="load" click="load_clickHandler(event)"/>
      <mx:DataGrid x="240" y="39" id="CategoryList" dataProvider="{categories}">
            <mx:columns>
                  <mx:DataGridColumn headerText="CategoryName" dataField="categoryName"/>
                  <mx:DataGridColumn headerText="CategoryID" dataField="categoryID"/>
            </mx:columns>
      </mx:DataGrid>
      <mx:DataGrid x="239" y="198" id="ProductList"
dataProvider="{Category(CategoryList.selectedItem).products}">
            <mx:columns>
                  <mx:DataGridColumn headerText="ID" dataField="productID"/>
                  <mx:DataGridColumn headerText="Name" dataField="productName"/>
            </mx:columns>
      </mx:DataGrid>
</s:Application>
```

ccategories.cfc

```
<cfcomponent persistent="true" table="categories">
    <cfproperty name="categoryID" fieldtype="id" generator="native"/>
    <cfproperty name="categoryName"/>
    <cfproperty name="description"/>
    <cfproperty name="products" fieldtype="one-to-many" cfc="cproducts" cascade="all"
                fkcolumn="categoryid" lazy="true" remotingfetch="lazy"/>
</cfcomponent>
```

cproducts.cfc

```
<cfcomponent persistent="true" table="products">
    <cfproperty name="productID" fieldtype="id" generator="native"/>
    <cfproperty name="productName"/>
    <cfproperty name="categoryID" fieldtype="many-to-one" cfc="ccategories" lazy="true"
                remotingfetch="lazy"/>
</cfcomponent>
```

FlashBuilder でデバッガー／ネットワークモニターを使用するときの注意事項

デバッグモードでサーバーによって返された結果を調べるとき、デバッガーは関連エンティティを取得します（結果として、遅延ロードの目的は無効になります）。ネットワークモニターを使用するときも同じです。

ただし、アプリケーションを実行するときは、この問題は発生しません。

オフライン AIR アプリケーションのサポート

ColdFusion には、データの永続性や同期などの機能を含め、オフライン状態の AIR アプリケーションをサポートする機能が用意されています。これらの機能を使用すると、ColdFusion サーバー上のデータを表すローカル SQLite データベースを AIR アプリケーションで使用できます。

Flash でビルドされたアプリケーションで、これらの機能を使用することはできません。これらのアプリケーションはブラウザまたは Flash Player 内で稼動します。これらの機能では、ColdFusion データプロバイダに断続的に接続する AIR アプリケーションのみがサポートされます。これらの機能を使用すると、AIR アプリケーションをオフラインで実行し、次回アプリケーションをオンラインで実行するときに、データを ColdFusion アプリケーションと同期できるようになります。

オフライン AIR データアクセスをサポートするには、クライアントサイドで ActionScript 要素をコーディングし、サーバーサイドで CFML をコーディングします。

注意：この後の説明に含まれるコード例の中には、ColdFusion で管理されている従業員データベースを表示して更新する AIR アプリケーションを使用しているものがあります。ただし、すべてのコードがこの例に基づいているわけではないため、これらのコードは完全に一貫したアプリケーションを構成してるわけではありません。

関連項目

638 ページの「ColdFusion 9.0.1 でのオフライン AIR アプリケーションのサポート」

ColdFusion サーバーサイド

ColdFusion アプリケーションは、CFC を使用して、交換または同期するデータを表現します。たとえば、次のような構造を持つ ORM 従業員コンポーネントがあるとします。

```
<cfcomponent persistent="true" displayname="EMP">
    <cfproperty name="id" type="numeric" fieldtype="id" generator="native">
    <cfproperty name="firstName" type="string">
    <cfproperty name="lastName" type="string">
    ...
    <cfproperty name="countryCode" type="string">
</cfcomponent>
```

従来の非 ORM CFC を使用することもできます。この場合は、Fetch メソッドと Sync メソッドで cfquery タグ、および関連するタグと関数を使用して、データベースを操作します。

AIR アプリケーションとの対話を管理し、データの同期を維持するために、ColdFusion アプリケーションは、SyncManager というコンポーネントを使用します。SyncManager は CFIDE.AIR.ISyncManager インターフェイスを実装します。このコンポーネントには、次の 2 つの関数があります。

- fetch 関数 : この関数は、ColdFusion からデータを取得するときに AIR アプリケーションで呼び出します。この関数は ISyncManager インターフェイスの一部ではありませんが、必要です。この関数には任意の名前を付けることができますが、慣例的に fetch と呼ばれます。
- sync 関数 : この関数は、データを更新または変更したときに、ColdFusion および AIR データソースを同期するために AIR アプリケーションで呼び出します。この関数は、次の 3 つのパラメータを取ります。

  operations INSERT、UPDATE、または DELETE を実行する操作の配列。

  clientObjects オブジェクトの配列。配列の各アイテムは、同期するデータのクライアントにおける現在のビューを表します。

  originalObjects オブジェクトの配列。この配列の各アイテムは、ユーザーが現在更新している既存の従業員レコードなど、サーバー上で対応する元データ (存在する場合) を表します。INSERT 操作の場合、このオブジェクトは null になります。DELETE 操作の場合、通常は、このオブジェクトは現在のデータと同じになります。

同期プロセスで競合が発生した場合、sync 関数は AIR クライアントに "CFIDE.AIR.Conflict.cfc" オブジェクトの配列を返します。この Conflict オブジェクトはそれぞれ、単一の serverObject 要素で構成されています。sync 関数を使用すると、競合しているレコードのサーバーコピーと等しくなるように要素が設定されます。その後、クライアントアプリケーションでは、628 ページの「競合の管理」で説明されているように競合を処理できます。

サーバーサイドに関する注意

- sync 関数で DELETE 操作を実行する場合は ClientObject が NULL になるため、プライマリキー ID は Sync メソッドの OriginalObject から取得されます。UPDATE 操作と INSERT 操作の場合は、ClientObject キーの値を使用します。
- 次のコードに示すように、INSERT 操作を実行すると、sync メソッドの OriginalObject パラメータが単純な値であるかどうかが CFC によって確認されます。

```
{NOT IsSimpleValue(OriginalObject)}
```

- INSERT 操作の場合、Sync 関数に渡される OriginalObject は null です。したがって、このオブジェクトのプロパティの取得しようとすると、メソッドが見つからないこと示すエラーが返されます。たとえば、OriginalObject.GetID を使用すると、メソッド GetID() が見つからないことを示すエラーが返されます。したがって、INSERT 操作の場合は、ClientObject を使用して各種のフィールドにアクセスします。
- ColdFusion アプリケーションでは cfquery を使用してデータベースを直接管理できますが、大部分の AIR アプリケーションでは ORM 機能を使用する必要があります。この後の説明では、サーバーサイドのデータ管理に ColdFusion ORM を使用します。
- オフラインの AIR アプリケーションに対して ColdFusion 8 Remoting を使用していて、それらのアプリケーションに ORM の EntitySave() および EntityDelete() メソッドを使用するサーバーサイドの "Sync" メソッドがある場合は、次のようなエラーメッセージが表示されることがあります。

  ```
  Error handling message: flex.messaging.MessageException: Unable to invoke CFC - a different object with
  the same identifier value was already associated with the session: [address#1].
  Root cause:org.hibernate.NonUniqueObjectException: a different object with the same identifier value was
  already associated with the session: [address#1]
  ```

  このエラーは ColdFusion 9 Remoting でも発生する場合がありますが、EntityDelete メソッドを使用した場合にのみ発生します。

  このエラーを解決するには、"Sync" メソッドで、次のように EntitySave および EntityDelete メソッドを呼び出します。

  ```
  <cfif operation eq "INSERT" OR operation eq "UPDATE">
      <cfset obj = ORMGetSession().merge(clientobject)>
      <cfset EntitySave(obj)>
  <cfelseif operation eq "DELETE">
      <cfset obj = ORMGetSession().merge(originalobject)>
      <cfset EntityDelete(obj)>
  </cfif>
  ```

- 競合が発生した場合、sync 関数は "CFIDE.AIR.Conflict" オブジェクトの配列をクライアントに返します。Conflict オブジェクトに含めることができるプロパティには、operation、serverobject、clientobject、originalobject の 4 つがあります。

  Conflict オブジェクトの serverobject プロパティは、サーバーサイドのデータベーステーブルを表す、ユーザー定義の CFC タイプである必要があります。次の例では、employee.cfc タイプの有効な ServerObject プロパティを含む Conflict オブジェクトが生成されます。このタイプは、従業員テーブルを表します。

  ```
  <cfset serverobject = EntityLoadByPK("employee",originalobject.getId())>
  <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
  <cfset conflict.serverobject = serverobject>
  <cfset conflict.clientobject = clientobject>
  <cfset conflict.originalobject = originalobject>
  <cfset conflict.operation = operation>
  <cfset conflicts[conflictcount++] = conflict>
  <cfreturn conflicts>
  ```

  ColdFusion ORM を使用する場合は、上記の例を次のコードに置き換えることができます。

  ```
  <cfset conflict = CreateObject("component","CFIDE.AIR.Conflict")
  <cfset serverobject = EntityLoadByPK("employee",#res.IDENTITYCOL#)>
  <cfset conflict.SetServerobject(serverobject)>
  ```

- 古いデータを使用している AIR クライアントが、既にデータベースから削除されたレコードを更新しようとすると、サーバーから Conflict が返され、KeepAllServerObjects メソッドまたは KeepServerObject メソッドを含むクライアントの競合ハンドルによってサーバーからの変更が受け入れられます。ただし、クライアントメソッドは、サーバー上に既に存在しない古いレコードをクライアントデータベースから削除しません。

  この問題を防止するには、クライアントが更新しようとしているレコードがデータベース上に既に存在しない場合に、サーバーから返す Conflict オブジェクトの serverObject プロパティを null にする必要があります。次に例を示します。

```
<cfset serverobject = EntityLoadByPK("employee",originalobject.getId())>
<!----If the operation is INSERT, serverObject is also NULL.hence NEQ condition---->
<cfif not isdefined('serverobject') and operation NEQ "INSERT" >
<cflog text="CONFLICT::SERVER OBJECT NOT FOUND, RECORD MAY BE DELETED ALREADY">
<cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
<cfset conflict.clientobject = clientobject>
<cfset conflict.originalobject = originalobject>
<cfset conflict.operation = operation>
<cfset conflicts[conflictcount++] = conflict>
<cfcontinue>
</cfif>
```

オフライン AIR アプリケーションのコード構文

ColdFusion と同期する AIR アプリケーションをコーディングするには、AIR プロジェクトに "cfair.swc" ファイルを追加します。このファイルには、AIR と ColdFusion の間の対話をサポートする ColdFusion クライアントサイドコードがすべて含まれています。このファイルは、ColdFusion と同時に "<ColdFusion Web ルートディレクトリ >/CFIDE/scripts/AIR" ディレクトリにインストールされます。Flash Builder で、プロジェクト -[プロパティ]- Flex ビルドパス -[ライブラリパス]-[Add SWC の追加] ダイアログボックスを開き、swc ファイルの場所を指定します。

データオブジェクト

AIR アプリケーションは、ColdFusion サイドのデータ CFC に対応する ActionScript オブジェクト内の管理対象データを表します。たとえば、次の Employee を例に取ると、AIR アプリケーションの "Employee.as" ファイルには、ColdFusion の "employee.cfc" に対応する ActionScript クラス "Employee " が含まれています。

```
package test.basic
    {
        [Bindable]
        [RemoteClass(alias="AIRIntegration.employee")]
        [Entity]
        public class Employee
        {
        /** The user id of the employee **/
        [Id]
        public var id:int;
        public var firstName : String;
        public var lastName : String;
        public var displayName : String;
        ....
        public var countryCode : String = 'US';
        ....
        }
```

注意：SQLite データベースおよびテーブルは自動的に作成されるので、明示的に作成する必要はありません。たとえば、上記のように Employee クラスを定義した場合、最初にこのクラスを呼び出すと、サーバーデータを永続化するために同等の SQLite テーブルが作成されます。

データオブジェクトのメタデータ

データオブジェクトを定義するには、次のメタデータを使用します。

メタデータ要素	用途
[Entity]	このクラスのインスタンスを SQLite データベースに永続化できかどうかを指定します。この要素は必須です。
[Table(name="tableName")]	オブジェクトを保存する SQLite テーブルの名前。デフォルトでは、クラスと同じ名前に設定されます。
[Id]	フィールド定義の前に配置されます。このフィールドがテーブルのプライマリキーであることを示します。複合キーの場合は、すべてのプライマリキーフィールドで Id タグを使用します。
[Transient]	このプロパティまたはフィールドが永続的であるかどうかを指定するブール値。値が True の場合、このフィールドは永続的ではありません。つまり、クライアントサイドの Sqlite テーブルの一部ではありません。
[Column(name="name", columnDefinition="TEXT|INTEGER|FLOAT|BOOLEAN|VARCHACHAR", nullable=true|false, unique=true|false)]),	このフィールドのデータを含む SQLite データベース内の列を指定します。 **name:** 列の名前。指定しない場合は、デフォルトでプロパティと同じ名前に設定されます。 **columnDefinition:** 列で使用する SQL データ型。 **nullable:** フィールドの値を null にできるかどうかを指定します。 **unique:** 各フィールドの値を列内で一意にする必要があるかどうかを指定します。
[RemoteClass]	ColdFusion だけでなく、すべてのリモートオブジェクトで使用されます。リモートサーバー上の対応するクラスを指定するには、alias 属性を使用します。この情報は、ActionScript データ型とリモートデータ型の間のマッピングに使用されます。 サーバーサイドの CFC にマッピングされる ActionScript クラスまたはエンティティの場合は、[RemoteClass] メタデータタグを必ず指定する必要があります。このメタデータタグを指定しないと、ランタイムエラーが発生します。たとえば、Address エンティティのエイリアス名を指定するには、[RemoteClass(alias="myFolder.AIRIntegration.Address")] のように記述します。 エイリアス名は、AIR アプリケーション内で一意になっている必要があります。

注意：クラス内のプライベートプロパティの場合は、次のコードに示すように、プライベートプロパティではなく、アクセサ関数 (getxxx および setxxx) で [Column] メタデータを設定します。

```
private var name:String; // Private property
[Column(name="FNAME",columnDefinition="VARCHAR")]
public function set fname(name:String):void // accessor function
{
        this.name = name;
}
public function get fname():String // accessor function
{
        return name;
}
```

クライアントサイド

ColdFusion 9 では、オフラインアプリケーションサポートをアプリケーションのクライアントサイドにまで拡張するために、ActionScript 要素をクライアントサイドでコーディングできるようになっています。クライアントサイドで交換または同期されるデータは、ローカルデータベースまたはオフラインデータベース上の永続オブジェクトを通じて管理されます。

関係の管理

ActionScript の永続性フレームワークでは、2 つの永続オブジェクトの間で次の関係タイプを定義できます。

- 1 対 1
- 1 対多
- 多対 1

- 多対多

これらの関係が永続性フレームワーク内でどのように処理されるかを理解するために、データベースの Employee オブジェクトと Department オブジェクトを例に取って検討しましょう。

属性値を指定しない場合は、次のデフォルト値が使用されます。

- デフォルトのテーブル名は、クラス名です。
- columnDefinition のデフォルト値は、フィールドの ActionScript 型です。
- referencedColumnName のデフォルト値は、ターゲットエンティティのプライマリキーです。
- targetEntity のデフォルト値は、参照元フィールドの ActionScript 型です。

注意： ORM の CFC を使用する場合は、どの関係に対しても <cfproperty> タグの remotingFetch 属性がデフォルトで false に設定されます。クライアントサイドのデータを取得するには、この属性を true に設定する必要があります。

1 対 1 関係

ここでは、1 人の従業員が 1 つの部門のみに所属している 1 対 1 の関係を検討します。Department オブジェクトと Employee オブジェクトの間で 1 対 1 のマッピングを定義し、DEPTID を外部キー列名として使用するには、次のようなコードを使用します。

```
[Entity]
[Table(name="employee")]
public class Employee
    {
        [Id]
        var id:uint;
        [OneToOne(targetEntity="Department"|fetchType="EAGER(default)|LAZY")]
        [JoinColumn(name="DEPTID", referencedColumnName="DEPT_ID")]
        var dept:Department;
    }
```

[JoinColumn] タグでは、外部キー列と列タグのすべての属性を指定します。関係の両方のエンティティに [JoinColumn] を指定しないでください。たとえば、Department オブジェクトと Employee オブジェクトの間の 1 対 1 の関係では、関係の方向に応じて、一方のエンティティにのみ [JoinColumn] を指定します。

referencedColumnName では、参照するプライマリキー列を指定します。Class は、ターゲットエンティティを示します (この例では、Department です)。

fetchType のデフォルト値は EAGER です。fetchType の詳細については、624 ページの「遅延ロードと取得タイプ」を参照してください。

1 対多関係

ここでは、1 人の従業員が複数の部門に所属している 1 対多の関係を検討します。Department オブジェクトと Employee オブジェクトの間で 1 対多のマッピングを定義するには、次のようなコードを使用します。

```
public class Employee
    {
        [Id]
        var id:uint;
        [OneToMany(targetEntity="Department",mappedBy="department",
        fetchType="EAGER|LAZY(default)")]
        var depts:ArrayCollection;
    }
```

Employee テーブルで指定されている列はありませんが、Employee エンティティを指す Department エンティティ内のフィールドが参照されています。

fetchType のデフォルト値は LAZY です。fetchType の詳細については、624 ページの「遅延ロードと取得タイプ」を参照してください。

多対 1 関係

ここでは、複数の従業員が 1 つの部門に所属している多対 1 の関係を検討します。Department オブジェクトと Employee オブジェクトの間で多対 1 のマッピングを定義するには、次のようなコードを使用します。

```
public class Employee
    {
        [Id]
        var id:uint;
        [ManyToOne(targetEntity="Department",fetchType="EAGER(default)|LAZY")]
        [JoinColumn(name="deptId",referencedColumnName="DEPT_ID")]
        var dept:Department;
    }
```

fetchType のデフォルト値は EAGER です。fetchType の詳細については、624 ページの「遅延ロードと取得タイプ」を参照してください。

多対多関係

ここでは、複数の従業員が複数の部門に所属している多対多の関係を検討します。Department オブジェクトと Employee オブジェクトの間で多対多のマッピングを定義するには、次のようなコードを使用します。

```
public class Employee
    {
        [Id]
        [Column(name="ID")]
        var id:uint;
        [ManyToMany(targetEntity="Department",
        fetchType="EAGER|LAZY(default)")]
        [JoinTable(name="EMP_DEPT")]
        [JoinColumn(name="EMPID", referencedColumnName="ID")]
        [InverseJoinColumn(name="DEPID", referencedColumnName="DEPTID")]
        var depts:ArrayCollection;
    }
```

fetchType のデフォルト値は LAZY です。fetchType の詳細については、624 ページの「遅延ロードと取得タイプ」を参照してください。

多対多関係の場合は、両方ではなく、一方のエンティティにのみ次のようなメタデータを指定します。

```
[JoinTable(name="ORDER_PRODUCT")]
[JoinColumn(name="ORDER_ID",referencedColumnName="oid")]
[InverseJoinColumn(name="PRODUCT_ID",referencedColumnName="pid")]
```

[JoinColumn] タグでは、外部キー列と列タグのすべての属性を指定します。[InverseJoinColumn] タグでは、[JoinTable] タグの結合エンティティへの参照を指定します。この例では、結合テーブル "EMP_DEPT" に "DEPID" という列があり、この列は Department テーブルの "DEPTID" 列を参照しています。

[JoinTable] タグでは、結合列と逆結合列を指定する多対多関係の結合テーブルを定義します。この例では、Offline SQLite DB に "EMP_DEPT" という結合テーブルが作成され、Employee テーブルと Department テーブルの間の多対多関係が定義されます。

遅延ロードと取得タイプ

ActionScript の永続性フレームワークでは遅延ロードがサポートされています。ただし、データベースとの接続が非同期方式なので、この機能はあまり直感的ではない場合があります。

ロードされる関係の取得タイプには EAGER と LAZY の 2 つがあります。取得タイプが EAGER の場合は、最初に呼び出しが行われたときに、関係をロードして、データを取得します。取得タイプが LAZY の場合は、明示的に fetch が呼び出されたときにのみ、関係をロードして、データを取得します。取得タイプのデフォルト値は、EAGER で、ignoreLazyLoad のデフォルト値は false です。

クラス定義レベルで fetchType="EAGER" を指定した場合、loadByPk 関数は、ignoreLazyLoad パラメータで指定された値とは無関係に、関連オブジェクトを常にロードします。

クラス定義レベルで fetchType="LAZY" を指定した場合は、次の 2 つの可能性があります。

- ignoreLazyLoad パラメータを true に設定すると、関連オブジェクトもロードされます。たとえば、Address と Customer という 2 つの関連オブジェクトがある場合、loadByPK(Customer,{id:3},true) を指定すると、Address オブジェクトもロードされます。
- ignoreLazyLoad パラメータの値を指定しない場合は、デフォルト値の false が使用されるので、関連オブジェクトはロードされません。たとえば、Address と Customer という 2 つの関連オブジェクトがある場合、loadByPK(Customer,{id:3}) を指定すると、Address オブジェクトはロードされません。

カスケーディングオプション

カスケーディングを使用すると、指定した操作を、親オブジェクトから関連オブジェクトにまで連鎖的に適用できます。サポートされる操作は、INSERT、UPDATE、および DELETE です。cascadeType 属性では、次のいずれかの値を設定できます。

ALL ソースエンティティが挿入、更新、または削除すると、ターゲットエンティティも挿入、更新、または削除されます。

PERSIST ソースエンティティが挿入または更新されると、ターゲットエンティティも挿入または更新されます。

REMOVE ソースエンティティが削除すると、ターゲットエンティティも削除されます。

カスケーディングでは、1 対 1、1 対多、多対 1、多対多の関係がすべてサポートされます。カスケーディングオプションを指定するには、次のようなコードを使用します。

```
ManyToMany(cascadeType="ALL or PERSIST or REMOVE")
```

cascadeType オプションを指定しない場合は、ソースエンティティのみが永続化または更新されます。

cascadeType='ALL or REMOVE' を指定して親オブジェクトとそれに関連する子オブジェクトを削除する場合は、load***() メソッドを使用して親オブジェクトをロードし、session.remove(parentObj) を使用して親オブジェクトを渡します。このロードメソッドを使用しなかった場合は、親オブジェクトのみが SQLite データベースから削除され、子オブジェクトは残ります。

注意：fetchType='LAZY' をエンティティレベルで指定して遅延ロードを有効にしている場合、load***() メソッドを使用して親オブジェクトをロードしても、子オブジェクトはロードされません。cascadeType='ALL or REMOVE' を指定し、session.remove(parentObj) で親オブジェクトを渡して削除しようとした場合、子オブジェクトは削除されません。この問題を解決するには、ignorelazyloading='true' を指定して load***() メソッドを使用します。

AIR SyncManager クラスを使用したデータの管理

AIR アプリケーションは SyncManager クラスを使用してサーバーからデータを取得し、ローカルデータを ColdFusion データソースと同期します。SyncManager は coldfusion.air.Session オブジェクトを使用してクライアントとローカル SQLite データベースの間のセッションを管理し、ColdFusion 同期マネージャ CFC 内の次のメソッドを呼び出します。

- fetch: リモートシステムからデータを取得するときに呼び出します。
- sync: ローカルデータとリモートデータを同期するときに呼び出します。

ここでは、実装する必要がある基本的な機能について説明します。SyncManager クラスと Session クラスの詳細、および coldfusion.air パッケージに含まれる他のクラスの詳細については、ActionScript 3.0 リファレンスを参照してください。また、ColdFusion Administrator の [情報源] ページの [マニュアル] リンクから、スタンドアローンの『Adobe ColdFusion リファレンスガイド』を参照することもできます。

次のコードに示すように、AIR アプリケーションの init() 関数は SyncManager インスタンスを作成して設定し、ColdFusion から初期データを取得します。

```
private function init():void
{
syncmanager = new SyncManager();
//The ColdFusion server and port. Port without double quotes as it is

//expected to be integer and IP is taken as String.
syncmanager.cfPort = CFServerPort;
syncmanager.cfServer = "CFServerIP";

//The CFC that implements the ColdFusion sync manager. Here
//AIRIntegration is the user defined folder under webroot.
syncmanager.syncCFC = "AIRIntegration.empManager";

//Specify a user-defined CF destination,if not specified, default destination
//'ColdFusion' will be used
syncmanager.destination = 'USerDefinedCFDestination'

//The event listener for conflict events returned byt the CFC
syncmanager.addEventListener(ConflictEvent.CONFLICT, conflictHandler);

//The local database file
var dbFile:File = File.userDirectory.resolvePath("EmpManager.db");

//Create a session object, which handles all interactions between the
//AIR application and the SQLite database.
var sessiontoken:SessionToken = syncmanager.openSession(dbFile, 999);

//Add a responder for handling session connection results.
sessiontoken.addResponder(new mx.rpc.Responder(connectSuccess,
connectFault));
}
```

サーバーからのデータの取得

SyncManager の fetch メソッドを使用して ColdFusion サーバーからデータを取得するには、サーバーデータオブジェクトの fetch メソッドを呼び出します。syncManager.fetch メソッドは、CFC の取得メソッドの名前 (通常は fetch) を第 1 パラメータとして取り、その後 CFC の取得メソッドの任意のパラメータを取ります。

syncManager.fetch メソッドは、データへのアクセスを可能にする AsyncToken オブジェクトを返します。このトークンは同期的に返されます。ColdFusion CFC からのレスポンスは非同期的に返されます。したがって、取得に成功または失敗した場合のレスポンスを処理するレスポンダを指定するには、トークンの addResponder メソッドを呼び出します。

サーバーから初期データを取得するには、Application init() メソッドに次のコードを含めます。

```
// fetch the data.
var token:AsyncToken = syncmanager.fetch("fetch");
//Specify the responder to handle the fetch results.
token.addResponder(new mx.rpc.Responder(fetchSuccess, fetchFault));
```

ローカルデータベースの管理

ローカル SQLite データベースのデータを管理するには、Session オブジェクトを使用します。特定のデータベースファイルとのセッションを開始するには、syncmanager.openSession メソッドを呼び出します。このメソッドは SessionToken トークンを返し、SessionToken.session プロパティはこのセッションへのアクセスを可能にします。セッションオブジェクトへのアクセスを可能にするには、トークンの openSessionSuccess レスポンダイベントハンドラのコードを使用します。したがって、セッションが正常に開かれるまでは、セッションおよびデータベースにはアクセスできません。

次のコードは、前のセッション初期化コードを拡張するものです。また、セッションを使用してリモートデータベースの内容をローカルイメージに保存する openSessionSuccess イベントハンドラを示しています。この例では、サーバーから取得される配列コレクションは users です。

```
var dbFile:File = File.userDirectory.resolvePath("basic.db");
var token:SessionToken = syncmanager.openSession(dbFile, 113);
token.addResponder(new mx.rpc.Responder(openSessionSuccess, openSessionFault));
function openSessionSuccess(event:SessionResultEvent):void
{
        //Initialize a variable for access to the session.
        var session:Session = event.sessionToken.session;
        //Save the remote database data in the local database.
        //users is the array collection fetched from server
        var saveToken:SessionToken = session.saveCache(users);
        //Add responder event handlers for successful and failed saves.
        saveToken.addResponder(new mx.rpc.Responder(saveSuccess, saveFault));
}
```

Begintransaction() 関数に対応する Committransaction() 関数が存在しない場合、AIR サイドの SQLite DB ファイルはロックされます。これを回避するには、イベントフローの最後で次のコードを使用します。

```
var closetoken:SessionToken = session.close();
closetoken.addResponder(new mx.rpc.Responder(closesuccess, closefault));
```

セッションへのアクセスが可能になった後は、セッションオブジェクトメソッドを呼び出して、SQLite データベースからデータを取得 (ロード) したり、データベース上のデータを挿入、削除、または更新することができます。セッションオブジェクトのメソッドについて詳しくは、ActionScript 3.0 リファレンスを参照してください。また、ColdFusion Administrator の [情報源] ページの [マニュアル] リンクから、スタンドアローンの『Adobe ColdFusion リファレンスガイド』を参照することもできます。

注意事項 :

- SQLite データベースでは、テーブルの作成時に列の型が検証されません。列のデータ型として無効な値が指定されている場合でも、その型を持つ列が作成されます。
- アプリケーションで使用されていない一意の整数 ID パラメータを OpenSession メソッドに渡すと、サーバー上でコミットする必要があるクライアント上での変更を追跡する中間データベースファイルが作成されます。1 つのアプリケーションで複数のデータベースを使用する場合は、データベースごとに一意の ID を使用します。1 つの ID を使用することにより、クライアントサイドのトランザクションごとに正しいデータベースを確実に使用できるようになります。
- 取得したデータをローカルデータベースに保存する非同期呼び出し (SaveCache など) の場合、レスポンダが追加される前に、呼び出し完了時のセッショントークンを使用することで、呼び出し結果が使用可能になる場合があります。この状況は、SaveCache 操作で null データを保存しようとした場合に発生します。つまり、取得操作で null データが返された場合です。このような場合、レスポンダは不要です。

  この状況に対処するには、次の 2 つの方法があります。

  1 関数から返されるセッショントークンの result プロパティが null であるかどうかを確認します。

  ```
  if (token.result == null) {
          Add the responder here
  }
  else {
          Directly use the token.result
      }
  ```

  2 SaveCache 関数への入力になる ArrayCollection が null でないことを確認します。レスポンスが null の場合は、取得操作でサーバーから結果が取得されなかったことを示しています。

  ```
  If (ArrayCollection != null) {
          Call SaveCache() function
          Add Responder to the SaveCache Token
      }
      else {
          Handle the condition here.
      }
  ```

- SaveUpdate メソッドを呼び出して、指定したプライマリキーを持つレコードが存在しない場合は、そのレコードが挿入されます。このメソッドで既存のレコードが更新されるのは、クライアントデータベースにプライマリキーが存在する場合のみです。
- サーバーからデータを取得した後は、SaveCache メソッドと SaveCacheUpdate メソッドのみを使用して、取得したデータをクライアントサイドデータベースに保存します。取得したデータの保存に Save 関数を使用した場合、そのデータには、コミット時にサーバーに挿入するためのマークが付けられるので、すぐにサーバーに書き戻されます。この場合、サーバーデータベースのプライマリキーの競合が発生します。クライアントからのプライマリキー ID を無視することによりサーバーサイドのロジックでこの競合を処理し、サーバーで新しい ID を生成できるようにした場合は、レコードが挿入されます。その結果、異なる ID を持つ複数のコピーが作成されます。
- AIR 統合をオフラインでサポートする場合、クライアントサイドの ActionScript クラスの変数をグローバルに宣言せずに、session.saveCache() または session.saveUpdateCache() を使用してサーバーから取得したデータレコードを保存しようとすると、AIR サイドのエラースタックトレースに次のようなメッセージが記録されることがあります。

```
"Error: The object of type Object does not have the Entity metadata tag at
coldfusion.air::EntityCache/addMetadata()[D:\p4\depot\ColdFusion\cf_main\tools\AIRIntegration\OfflineS
upport\src\coldfusion\air\EntityCache.as:228]"
```

サーバーへのデータの送信

クライアントの SyncManager オブジェクトは、ローカルデータとサーバーデータを同期できるように、セッション中に発生したローカルデータの変更をすべて追跡します。また、SyncManager は、既にサーバー上にあるデータがローカルで更新されたときに、古いデータインスタンスを追跡します。

AIR アプリケーションから Session.commit メソッドを呼び出すと、セッション中に発生したすべての変更が CFC の同期関数に渡されます。同期関数は、返された情報の間に競合がないかどうかを確認します。競合がない場合は、サーバーのデータソースが更新します。競合がある場合は、628 ページの「競合の管理」で説明されているとおりにデータが処理されます。

注意： session.commit メソッドを呼び出してサーバーからエラーが返されなかった場合は、commit メソッドから CommitSucces イベントが送信されます。このイベントは、session.commit メソッドが正常に実行されたことを示し、クライアントデータがサーバーの CFC 同期メソッドに渡されたことを意味します。同期メソッドへデータを送信する間にエラーが発生した場合は、クライアントに CommitFault イベントが返されます。したがって、CommitSucces イベントは、サーバーがクライアントデータを保存したことを意味するのではなく、サーバーがデータを受信したことのみを意味します。たとえば、競合がある場合、サーバーはデータを保存しませんが、この場合も CommitSucces イベントは送信されます。競合は、SyncManager にイベントリスナーを追加することで、別個に処理します。イベントフローでは、CommitSucces イベントが先に送信され、その後に ConflictEvent イベントが送信されます。

SyncManager 用のリモート資格情報の設定

ColdFusion サーバーに接続する AIR クライアントを認証するには、リモーティング資格情報を送信します。この資格情報は、サーバーサイドの <cflogin> タグの下で使用できます。これは、通常の Flash Remoting オブジェクトに対してリモーティング資格情報を設定することと同じです。

次のコードには、SyncManager の getRemoteObject() メソッドが含まれています。このメソッドは、通常の Flash Remoting オブジェクトと同じく完全な制御を行えるように、基盤となる Flash Remoting オブジェクトを取得します。

```
syncmanager.getRemoteObject().SetRemoteCredentials("username","password");
```

競合の管理

サーバー上で既に変更されているデータをクライアントが変更すると、オフラインアプリケーションで競合が発生する場合があります。そのような競合を検出するために、session.Commit メソッドは、次のデータを ColdFusion サーバーの同期メソッドに渡します。

operations: INSERT、UPDATE、または DELETE を実行する操作の配列。

clientobjects: 新しいデータ変更の配列。

originalobjects: 変更前にクライアントデータベースに存在していたデータの配列。次の状況では、競合は発生しません。

- レコードを更新するときに、サーバー上のデータが originalobject のデータと一致している場合。変更前のクライアントにはサーバーと同じデータが存在していました。この場合、サーバーは自身のデータソースを更新します。古いクライアントデータがサーバー上のデータと異なる場合は、アプリケーション側で競合を処理する必要があります。
- 新規レコードを挿入する場合。この場合、originalobject の値は存在しませんが、ColdFusion はデータストアにレコードを挿入できます。

競合を検出するには、ColdFusion の ObjectEquals 関数を使用します。この関数に、クライアントからの新しい cfc インスタンスと元のインスタンスを渡すと、それらのインスタンスが等しいかどうかが確認されます。それらのインスタンスが等しい場合は、最新のデータがクライアントで使用されています。それらのインスタンスが等しくない場合は、サーバー上にあるインスタンスが同期メソッドによって返され、サーバーから Conflict が返される可能性があります。この場合、CFIDE.AIR.conflict.cfc のインスタンスが作成されて、serverobject プロパティ (この cfc の唯一のプロパティ) がサーバー上のデータ値に設定され、Conflict オブジェクトの配列が AIR クライアントに返されます。

次のコードは、同期操作に ORM メソッドを使用し、競合も処理する sync メソッドの例です。

```
<cffunction name="sync" returntype="any">
<cfargument name="operations" type="array" required="true">
<cfargument name="clientobjects" type="array" required="true">
<cfargument name="originalobjects" type="array" required="false">
<cfset conclits = ArrayNew(1)>
<cfset conflictcount = 1>

<cfloop index="i" from="1" to="#ArrayLen( operations )#">
    <cfset operation = operations[i]>
    <cfset clientobject = clientobjects[i]>
    <cfset originalobject = originalobjects[i]>
    <cfif operation eq "INSERT">
                <cfset obj = ORMGetSession().merge(clientobject)>
                <cfset EntitySave(obj)>
    <cfelseif listfindnocase("UPDATE,DELETE",operation) neq 0>
        <cfset serverobject = EntityLoadByPK("employee",originalobject.getId())>
        <cfif not isdefined('serverobject') >
            <cflog text="CONFLICT::SERVER OBJECT NOT FOUND, RECORD MAY BE DELETED ALREADY">
                    <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
                    <cfset conflict.clientobject = clientobject>
                    <cfset conflict.originalobject = originalobject>
                    <cfset conflict.operation = operation>
                    <cfset conflicts[conflictcount++] = conflict>
                    <cfcontinue>
            </cfif>
            <cfset isNotConflict = ObjectEquals(originalobject, serverobject)>
            <cfif isNotConflict>
                    <cfif operation eq "UPDATE">
                          <cfset obj = ORMGetSession().merge(clientobject)>
                          <cfset EntitySave(obj)>
```

```
                    <cfelseif operation eq "DELETE">
                          <cfset obj = ORMGetSession().merge(originalobject)>
                          <cfset EntityDelete(obj)>
                    </cfif>
              <cfelse><!----Conflict--->
                    <cflog text = "is a conflict">
                    <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
                    <cfset conflict.serverobject = serverobject>
                    <cfset conflict.clientobject = clientobject>
                    <cfset conflict.originalobject = originalobject>
                    <cfset conflict.operation = operation>
                    <cfset conflicts[conflictcount++] = conflict>
                    <cfcontinue>
              </cfif>
    </cfif>
              </cfloop>
<cfif conflictcount gt 1>
<cfreturn conflicts>
</cfif>
</cffunction>
```

CFC での競合の処理は、アプリケーションごとに異なります。状況によっては、競合を無視して、サーバーデータソースを新しいクライアントデータで上書きするほうが適切である場合もあります。上の例のように、CFC は多くの場合、サーバー上のデータ値を返すことによってクライアントに競合を通知します。

クライアントサイドでは、次のようなコードを使用して、サーバーから返された競合を処理するメソッドを登録します。

```
syncmanager.addEventListener(ConflictEvent.CONFLICT, conflictHandler);
function conflictHandler(event:ConflictEvent):void
{
var conflicts:ArrayCollection = event.result as ArrayCollection;
var token:SessionToken = session.keepAllServerObjects(conflicts);
token.addResponder(new mx.rpc.Responder(conflictSuccess, conflictFault));
}
```

conflictevent オブジェクトには、Conflict オブジェクトの配列が含まれており、Conflict オブジェクトには、clientinstance、originalinstance、および serverinstance が含まれています。次のコードに示すように、アプリケーションはサーバーのデータを受け入れる場合、keepAllServerObjects を呼び出して競合ハンドラに渡された ArrayCollection を受け入れるか、keepServerObject を呼び出して個々の Conflict インスタンスを受け入れます。この競合ハンドラは、返されたサーバーオブジェクトを単純に受け入れます。

```
function conflictHandler(event:ConflictEvent):void
{
var conflicts:ArrayCollection = event.result as ArrayCollection;
var conflict:Conflict = conflicts.getItemAt(0);
var token:SessionToken = session.keepServerObject(conflict);
token.addResponder(new mx.rpc.Responder(conflictSuccess, conflictFault));
}
```

競合は次の状況で発生する場合があります。

- サーバーデータが更新された後にクライアントが更新を実行した場合。この場合、クライアントでは古いデータインスタンスが使用されており、サーバー上の最新データは使用されていません。サーバーサイドでは、同期メソッドで conflict.cfc のインスタンスを作成し、そのインスタンスをサーバー上で設定することにより、クライアントに競合を通知できます。クライアントサイドでは、競合ハンドラで keepServerObject 関数を呼び出して、クライアントデータベースにサーバーインスタンスを上書きすることにより、競合を解決できます。
- サーバー上に既に存在しないレコードに対してクライアントが更新を実行した場合。この場合も、Conflict.cfc のインスタンスを作成して返すことにより、サーバーからクライアントに競合を渡すことができます。挿入操作の場合は、サーバー上のインスタンスが存在しないため、serverobject プロパティを設定する必要はありません。

- クライアントが挿入を実行したが、サーバーデータで自動インクリメントプライマリキーフィールドなどが使用されていた場合。この場合、サーバーでは、クライアントによって挿入されたプライマリキーは使用されません。サーバーは、正しいキーフィールド値をクライアントに通知するために、サーバーインスタンスを含む競合の cfc インスタンスを返します。その後、ActionScript Calling keepServerObject メソッドを使用して、サーバーからの新しいプライマリキー値でローカルデータを上書きできます。

注意：コミット後、または競合解決後は、サーバーに他のクライアントから送信された新しいデータが存在する可能性があるので、クライアントデータベースをサーバーデータソースに同期することをお勧めします。

ActionScript には、いくつかの予約済みキーワードがあります。クラスや SQLite テーブルに名前を付けるときには、予約済みキーワードを使用しないように注意してください。たとえば、Order は、ActionScript の予約済みキーワードです。テーブルやクラスの名前を Order にすると、テーブルの作成に失敗します。このような名前の競合を回避するには、[Table(name="OrderTable")] メタデータタグを使用して、デフォルト名をオーバーライドします。この場合、Order.as クラスのコードは次のようになります。

```
package test
{
[Entity]
[Table(name="OrderTable")]
public class Order
{
public function Order()
{
}
[Id]
public var oid:uint;
public var name:String;
[ManyToMany(targetEntity="test::Product",cascadeType='ALL')]
public var products:Array;
}
}
```

オフライン AIR アプリケーションの例

ここで使用する例では、データベース内の Customer オブジェクトと Address オブジェクトの間に 1 対 1 の関係があるオフライン AIR アプリケーションを作成する方法について説明します。この例は、別の関係タイプのオフライン AIR アプリケーションを作成する場合の基礎としても使用できます。

クライアントサイド (AIR アプリケーション) のコード

AIR プロジェクト内に "onetoone" という名前のフォルダを作成し、次のようなコードを含む ActionScript クラスファイル "Customer.as" および "Address.as" を追加します。

Customer.as

```
package onetoone
 {
        [Bindable]
        [RemoteClass(alias="AIRIntegration.custome r")]
        [Entity]

    public class  Customer
    {
        [Id]
            public var cid:int;
            public var name: String;
            [OneToOne(cascadeType='ALL',fetchType="EAGER")]
            [JoinColumn(name="add_id",referencedColumnName="aid")]

    public var address:Address;
    }
}
```

Address.as

```
package onetoone
{
        [Bindable]
        [RemoteClass(alias="AIRIntegration.address")]
        [Entity]

    public class Address
    {
        [Id]
            public var aid:int;
            public var street:String;
    }
}
```

MainApplication.mxml

データベースに対して CRUD 操作を実行するには、次のようなコードを "MainApplication.mxml" ファイルに追加します。

注意：ActionScript クラス "Customer.as" および "Address.as" については、グローバル変数が既に宣言されています。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplic ation xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" creationComplete="init()">
<mx:Script>

<![CDATA[
import mx.collections.ArrayCollection;
import mx.rpc.AsyncToken;
import mx.controls.Alert;

import coldfusion.air.events.*;
import coldfusion.air.*;

import onetoone.Address;
import onetoone.Customer;

private var session:Session;
private var dbFile:File;

private var cusColl:ArrayCollection;
private var syncmanager:SyncManager;
```

```
private var add:Address;//global variable for address.as
private var cus:Customer; //global variable for customer.as

private function init():void
    {
        // Provide Credentials for Server side Connection and CFC
        syncmanager = new SyncManager();
        syncmanager.cfPort = 80;
        syncmanager.cfServer = "localhost";
        // Path of the Server side CFC from CF webroot
        syncmanager.syncCFC = "AIRIntegration.cusManager";
        // This handler is called when any conflict occurs while
        writing back changes on the server side
        syncmanager.addEventListener(ConflictEvent.CONFLICT, conflictHandler);
        // Fetch Server side DB data onto Client SQLite DB while
        starting the App
        var token:AsyncToken= syncmanager.fetch("fetch");
        token.addResponder(new mx.rpc.Responder(fetchSuccess, fetchFault));
    }

private function conflictHandler(event:ConflictEvent):void
    {
        Alert.show("conflict man!");
        var conflicts:ArrayCollection = event.result as ArrayCollection;
        // Accept Server data and write it to client side SQLite DB
        var token:SessionToken
        = session.keepAllServerObjects(conflicts);
        token.addResponder(new
        mx.rpc.Responder(conflictSuccess, conflictFault));
    }

private function conflictSuccess(event:SessionResultEvent):void
    {
        Alert.show("conflict resolved");
    }

private function fetchSuccess(event:SyncResultEvent):void
    {
        var cus:Array = event.result as Array;
        cusColl = new ArrayCollection(cus);
        // Open a Session for the client side SQLite DB
        dbFile = File.userDirectory.resolvePath("onetoonesync.db");
        var sessiontoken:SessionToken
        =syncmanager.openSession(dbFile,017915);
        sessiontoken.addResponder(new
        mx.rpc.Responder(connectSuccess,connectFault));
    }

private function connectSuccess(event:SessionResultEvent):void
    {
        session = event.sessionToken.session;
        if(cusColl.length > 0)
        {
        // This operation saves it to the AIR SQLite DB
        var savetoken:SessionToken
        = session.saveCache(cusColl);
        savetoken.addResponder(new
        mx.rpc.Responder(saveCacheSuccess, savefault));
    }
    else
        {
            Alert.show("No data available from Server to save on local DB");
```

```
        }
    }

private function saveCacheSuccess(event:SessionResultEvent):void
    {
        Alert.show("Data saved on client Sqlite DB from Server");

        /*
        A new Insert is tried here. Note that this is not a complete user interface
        application. Otherwise, typically, users need to provide inputs to populate
        the Customer/Address Objects
        */
    var cus:Customer = new Customer();
    cus.cid=12;
    cus.name="New Customer";
    var add:Address = new Address();
    add.aid = 14;
    add.street = 'New Address';
    cus.address = add;

    /*
    INSERT the new Records, this is first saved in client side SQLite DB.
    On the Commit operation this new record is saved in the Server side DB
    Notice that although you are saving the Customer object here,
    this operation saves even the binded Address Object also,
    as both the entities are CASCADED inside Customer Class
    */
    var savetoken:SessionToken = session.save(cus);
    savetoken.addResponder(new mx.rpc.Responder(savesuccess, savefault));
    }

private function savesuccess(event:SessionResultEvent):void
    {
    Alert.show("Customer was Inserted Successfully");
    // Load some otehr Customer(ex: id=11) so that we can perform Update on
    that Customer
    var loadtoken:SessionToken =session.loadByPK(Customer,{cid:11});
    loadtoken.addResponder(new mx.rpc.Responder(loadCustomer,loadFault))
    }
private function loadCustomer(event:SessionResultEvent):void
    {
    var cus1:Customer = event.result as Customer;
    cus1.name = "UpdateCustomerName";
    var add1:Address = new Address;
    add1.aid = 22;
    add1.street = 'UpdatedCustomerAddress';
    cus1.address = add1;
    /*
    Let's call update now and save it to Client side SQLite DB
    */
    var savetoken:SessionToken = session.update(cus1);
    savetoken.addResponder(new mx.rpc.Responder(updateSuccess,updatefualt));
    }

private function updateSuccess(event:SessionResultEvent):void
    {
    Alert.show("Customer was updated Successfully");
    /*
    Let's Load another Customer(for example, with id 128) to perform a Delete operation on that
    */
    var loadtoken:SessionToken = session.loadByPK(Customer,{cid:128});
    loadtoken.addResponder(new mx.rpc.Responder(loadCustomerForDelete,loadFault));
```

```
    }

private function loadCustomerForDelete(event:SessionResultEvent):void
    {
    // pass the loaded customer to remove function
    var removetoken:SessionToken = session.remove(event.result);
    removetoken.addResponder(new mx.rpc.Responder(removeSuccess,removeFault));
    }
private function removeSuccess(event:SessionResultEvent):void
    {
    Alert.show("Customer was deleted Successfully");
    }

private function commit():void
    {
    /*
    Until now, you have performed Insert/Update/Delete operation on Customer/Address
    entities on the client side SQLite DB. Now use the Commit opertaion to
    send them to the Server.
    */
    var committoken:SessionToken = session.commit();
    committoken.addResponder(new mx.rpc.Responder(commitSuccess, commitFault));
    }

private function commitSuccess(event:SessionResultEvent):void
    {
    Alert.show("Server has been updated with local changes");
    /*
    Now that you have completed all the operations, you can close the SQLite DB
    connection/session. It is a good practice to Close the session,
    after you complete all the operations.
    */
    var closetoken:SessionToken
    = session.close();
    closetoken.addResponder(new
    mx.rpc.Responder(sessionclosesuccess, sessionclosefault));
    }
private function sessionclosesuccess(event:SessionResultEvent):void
    {
    Alert.show("Session Close Success");
    }

    // Fault Handlers
    private function fetchFault(event:SyncFaultEvent):void
    {
    Alert.show("fetch fault" + event.toString());
    }
    private function conflictFault(event:SessionFaultEvent):void
    {
    Alert.show("conflict not resolved");
    }
    private function connectFault(event:SessionFaultEvent):void
    {
    Alert.show("connect failure" + event.toString());
    }
    private function sessionclosefault(event:SessionFaultEvent):void
    {
    Alert.show("Session Close Failed::"+event.error);
    }
    private function removeFault(event:SessionFaultEvent):void
    {
    Alert.show("Delete Operation Failed::"+event.error);
```

```
    }
    private function commitFault(event:SessionFaultEvent):void
    {
    Alert.show("Commit Failed::"+event.error);
    }
    private function loadFault(event:SessionFaultEvent):void
    {
    Alert.show("Load Failed::"+event.error);
    }
    private function updatefualt(event:SessionFaultEvent):void
    {
    Alert.show("update fault"+event.error);
    }
    private function savefault(event:SessionFaultEvent):void
    {
    Alert.show("Save Fault::"+event.error);
    }
]]>
</mx:Script>
<mx:Button click="commit()" name="commitbutton"
label="Commit/write local data to Server">
</mx:Button>
</mx:WindowedApplication>
```

サーバーサイドのコード

次のようなコードを含む "Application.cfc"、"Customer.cfc"、"Address.cfc"、および "Cusmanager.cfc" という cfc ファイルを作成します。AIR クライアントは "Cusmanager.cfc" ファイルと対話します。このファイルでは、サーバーから取得し、再びサーバーに返して同期するデータを指定します。

Application.cfc

```
<cfcomponent>
        <cfset this.name = "OneTonOneExample">
        <cfset this.datasource="testorm">
        <cfset this.ormenabled="true">
        <cfset this.ormsettings={dialect = "MicrosoftSQLServer"}>
</cfcomponent>
```

Customer.cfc

```
<cfcomponent persistent="true">
        <cfproperty name="cid" fieldtype="id" >
        <cfproperty name="name" >
        <cfproperty name="address" fieldType='one-to-one'
        CFC="address" fkcolumn='aid' cascade='all' >
</cfcomponent>
```

Address.cfc

```
<cfcomponent persistent="true">
        <cfproperty name="aid" fieldtype="id" >
        <cfproperty name="street" >
</cfcomponent>
```

Cusmanager.cfc

```
<cfcomponent implements="CFIDE.AIR.ISyncManager">
<!----Fetch method--->
<cffunction name="fetch" returnType="Array" access="remote">
<cfset cus = ArrayNew(1)>
<cfset cus = EntityLoad("customer")>
<cfreturn cus>
</cffunction>
<!----SYNC method--->
<cffunction name="sync" returntype="any">
<cfargument name="operations" type="array" required="true">
<cfargument name="clientobjects" type="array" required="true">
<cfargument name="originalobjects" type="array" required="false">

<cfset conclits = ArrayNew(1)>
<cfset conflictcount = 1>

<cfloop index="i" from="1" to="#ArrayLen( operations )#">
    <cfset operation = operations[i]>
    <cfset clientobject = clientobjects[i]>
    <cfset originalobject = originalobjects[i]>
        <cfif operation eq "INSERT">
                <cfset obj = ORMGetSession().merge(clientobject)>
                <cfset EntitySave(obj)>
    <cfelseif listfindnocase("UPDATE,DELETE",operation) neq 0>
        <cfset serverobject = EntityLoadByPK("employee",originalobject.getcid())>
        <cfif not isdefined('serverobject') >
        <cflog text="CONFLICT::SERVER OBJECT NOT FOUND, RECORD MAY BE DELETED ALREADY">
                    <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
                    <cfset conflict.clientobject = clientobject>
                    <cfset conflict.originalobject = originalobject>
                    <cfset conflict.operation = operation>
                    <cfset conflicts[conflictcount++] = conflict>
                    <cfcontinue>
            </cfif>

            <cfset isNotConflict = ObjectEquals(originalobject, serverobject)>
            <cfif isNotConflict>

                    <cfif operation eq "UPDATE">
                          <cfset obj = ORMGetSession().merge(clientobject)>
                          <cfset EntitySave(obj)>
                    <cfelseif operation eq "DELETE">
                          <cfset obj = ORMGetSession().merge(originalobject)>
```

```
                        <cfset EntityDelete(obj)>
                    </cfif>

            <cfelse><!----Conflict--->
                    <cflog text = "is a conflict">
                    <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
                    <cfset conflict.serverobject = serverobject>
                    <cfset conflict.clientobject = clientobject>
                    <cfset conflict.originalobject = originalobject>
                    <cfset conflict.operation = operation>
                    <cfset conflicts[conflictcount++] = conflict>
                    <cfcontinue>
            </cfif>
    </cfif>
        </cfloop>
<cfif conflictcount gt 1>
<cfreturn conflicts>
</cfif>
</cffunction>
</cfcomponent>
```

ColdFusion 9.0.1 でのオフライン AIR アプリケーションのサポート

ColdFusion 9 で導入された AIR の統合機能には、Adobe Integrated Runtime (AIR) 内に存在する SQLite データベースでエンティティを永続化するための ActionScript ORM が含まれています。今回のリリースでは、この ActionScript ORM の次の機能が強化されています。

- プライマリキーの自動生成がサポートされています。
- (AIR 1.5 で導入された) 暗号化データベースがサポートされています。
- ActionScript ORM で SQLite データベースの操作を追跡するために使用するキャッシュファイルの場所が、applicationDirectory から applicationStoragedirectory 内に変わりました。キャッシュディレクトリの場所は、SyncManager の openSession API で指定できます。
- 1 対 1、1 対多、および多対多のデータベースの関係において、自己結合の関係がサポートされています。
- Array と ArrayCollection の両方が、データベースの関係のコレクションを表すために ActionScript エンティティで使用できるようにサポートされています。
- ActionScript ORM で、エンティティを SQLite データベースに永続化するために使用したすべての SQL ステートメントがロギングされます。
- 新しい keepClientObject と keepAllClientObjects の API が追加されました。これらの API では、ColdFusion サーバーで競合が発生してもサーバー側の更新が保持されません。
- SessionToken クラスがダイナミックであるため、ORM API から返されたトークンにデータを保存できます。
- autoCommit モードがサポートされています。

プライマリキーの自動生成

今回のリリースでは、GeneratedValue メタデータタグを使用した ActionScript ORM のプライマリキーの生成がサポートされています。

GeneratedValue

説明

このタグを ActionScript のプライマリキーファイルに追加すると、プライマリキーが自動生成されます。

パラメータ

パラメータ	説明
strategy	UUID は、Flash UUID API を使用して ID (string タイプのプライマリキー用) または INCREMENT (int タイプのプライマリキー用) を生成します。
initialValue	INCREMENT の strategy の場合にのみ適用します。プライマリキーの初期値を指定します。デフォルト値は 0 です。
incrementBy	INCREMENT の strategy の場合にのみ適用します。プライマリキーを生成するときにインクリメントされる値の整数を指定します。

オブジェクトにその ID 値が存在しない場合は、その値が生成されてプライマリキー値として割り当てられます。オブジェクトインスタンスにそのキー値が既に存在する場合は、そのキーの生成は無視されます。

整数のプライマリキーの場合、データベーステーブルに存在する既存のプライマリキーがチェックされます。一番大きいキー値が initialValue より大きい場合、次に生成されるキーは、一番大きいキー値をインクリメントしたものになります。たとえば、指定した initialValue が 1 で、データベースに (既に) キー値 5 が存在する場合、次のキーは 6 (5+1。incrementBy が 1 に設定されている場合) の値で生成されます。

例

```
//Integer Primary Keys
===========
package test.apollo.CFSQLiteSupport.INCREMENTPK
{
    [Entity]
    [RemoteClass(alias="Customer")]
    public class Customer
    {
        public function Customer()
        {
        }
        [Id]
        [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2) ]
        public var cid:int;
        public var name:String;

        [OneToOne(mappedBy="customer")]
        public var ord:Order;
    }
}
//String Primary Keys
===========
package test.apollo.CFSQLiteSupport.UUIDPK
{
    [Entity]
    [RemoteClass(alias="Customer")]
    public class Customer
    {
        public function Customer()
        {
        }
        [Id]
        [GeneratedValue(strategy="UUID") ]
        public var cid:String;
        public var name:String;

        [OneToOne(mappedBy="customer")]
        public var ord:Order;
    }
}
```

データベースの暗号化

暗号化キーを使用して、ActionScript ORM で使用するデータベースを保護することができます。

syncmanager.openSession メソッドの ByteArray 暗号化キーを使用して、データベースを暗号化します。(ActionScript ORM で使用する) ユーザー指定のデータベースファイルとキャッシュデータベースファイルの両方が、指定した暗号化キーで暗号化されます。

このキーはオプションです。

例

```
dbFile = File.userDirectory.resolvePath("customerManger.db");
    dbDir = File.applicationStorageDirectory;
    var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey("UserPassword");

    var sessiontoken:SessionToken =syncmanager.openSession(dbFile,179176,encryptionKey,dbDir);
```

EncryptionKeyGenerator の詳細については、『Flex での Adobe AIR 1.5 アプリケーションの開発』の「EncryptionKeyGenerator クラスによる安全な暗号化キーの取得」セクションを参照してください。

キャッシュディレクトリの指定

キャッシュファイルを保存するキャッシュディレクトリを指定するには、syncmanager.openSession メソッドの cacheDirectory (flash.filesystem.File のインスタンス) を使用します。

cacheDirectory はオプションです。

注意：デフォルトでは、ActionScript ORM で使用するキャッシュファイルは、File.applicationStorageDirectory に保存されます。ColdFusion 9 では、File.applicationDirectory に保存されていました。

例については、640 ページの「データベースの暗号化」を参照してください。

自己結合のサポート

データベーステーブルは、外部キーによって自分自身に対して関連付けることができます。一般的な例として、Employee テーブルに管理者の関係が定義され、この定義に管理者 (従業員を管理する人間) の従業員 ID が含まれている場合を考えます。

管理者 ID は、同じテーブルの別の行を参照します。これが、1 対 1 の自己結合の例です。

中間的な結合テーブルを使用した 1 対多の自己結合や多対多の自己結合も可能です。

ColdFusion 9 アップデート 1 では、ActionScript ORM のすべての関係の自己結合をサポートしています。

次の ActionScript の customer エンティティのクラス定義では、すべての自己結合の関係がどのように定義されるかを示しています。

```
    package
{

    [Bindable]
    [RemoteClass(alias="AIRIntegration.customer")]
    [Entity]
    public class Customer
    {
        [Id]
        [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2)]
        public var cid:int;
        public var name:String;

        [OneToOne(cascadeType='ALL',fetchType="EAGER")]
        [JoinColumn(name="add_id",referencedColumnName="aid")]
        public var address:Address;

        // Many-to-One self Join
        [ManyToOne(targetEntity="onetoone::Customer",fetchType="EAGER")]
        [JoinColumn(name="managerId",referencedColumnName="cid")]
        public var manager:Customer;

        // One-to-one Self Join
        [OneToOne(targetEntity="onetoone::Customer",fetchType="EAGER")]
        [JoinColumn(name="spouseId",referencedColumnName="cid",unique="true")]
        public var spouse:Customer;

       // Many-to-Many self Join
       [ManyToMany(targetEntity="onetoone::Customer",fetchType="EAGER")]
       [JoinTable(name="CUSTOMER_PARENTS_MAPPINGS")]
       [JoinColumn(name="CUST_ID",referencedColumnName="cid")]
       [InverseJoinColumn(name="PARENT_ID",referencedColumnName="cid")]
       public var parents:Array;


       // Many-to-Many self Join
       [ManyToMany(targetEntity="onetoone::Customer",fetchType="EAGER")]
       [JoinTable(name="CUSTOMER_CHILDREN_MAPPINGS")]
       [JoinColumn(name="CUST_ID",referencedColumnName="cid")]
       [InverseJoinColumn(name="CHILD_ID",referencedColumnName="cid")]
        public var children:Array;

[OneToMany(targetEntity="onetoone::Order",cascadeType='REMOVE',mappedBy="customer",fetchType="EAGER")]
        public var orders:Array;

    }
}
```

ArrayCollection による複数のエンティティの保持

データベースの関係の複数のエンティティを保持するのに、Array だけでなく、ArrayCollection も使用できるようになりました。また、関連エンティティを表すのに Array を使用するときのように、ActionScript エンティティで ArrayCollection を使用することもできます。

例

```
package
{
    import mx.collections.ArrayCollection;

    [Bindable]
    [RemoteClass(alias="AIRIntegration.customer")]
    [Entity]
    public class Customer
    {
        [Id]
        [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2)]
        public var cid:int;
        public var name:String;

        [OneToOne(cascadeType='ALL',fetchType="EAGER")]
        [JoinColumn(name="add_id",referencedColumnName="aid")]
        public var address:Address;

[OneToMany(targetEntity="onetoone::Order",cascadeType='REMOVE',mappedBy="customer",fetchType="EAGER")]
        public var orders:ArrayCollection;

    }
}
```

サーバーサイドの設定

664 ページの「ColdFusion 9 および ColdFusion 9.0.1 における Flash Remoting 用の XML 設定ファイルの変更点」を参照してください。

SQL ステートメントのロギング

ActionScript ORM では、実行されたすべての SQL ステートメントがロギングされます。

ログの設定は、次のように行います。

❖ AIR アプリケーションのログ対象を追加するには、次の例のように指定します。

```
var logTarget:TraceTarget = new TraceTarget();
logTarget.filters = "*";
logTarget.level = LogEventLevel.ALL;
Log.addTarget(logTarget);
```

ログ対象は、すべての trace ステートメントが表示される TraceTarget です。ログ対象は、Flash API を使用するその他のログに設定することもできます。

ColdFusion ActionScript API

coldfusion.air パッケージの session クラスに、次の 2 つの API が導入されました。

keepAllClientObjects

説明

競合インスタンスの ArrayCollection を取得し、すべての競合インスタンスのクライアントオブジェクトを ArrayCollection で保持します。

戻り値

coldfusion.air.SessionToken (keepAllClientObjects を呼び出すためのトークン) のインスタンス

シンタックス
public function keepAllClientObjects(conflicts:ArrayCollection):SessionToken

パラメータ

パラメータ	説明
mx.collections.ArrayCollection	サーバーによって発生した競合の ArrayCollection

例
```
private function conflictHandler(event:ConflictEvent):void
{
   // Alert.show("Server returned a Conflict !");
    var conflicts:ArrayCollection = event.result as ArrayCollection;
    // Ignore Server data and retain client Data in  SQLite DB
    var token:SessionToken = session.keepAllClientObjects(conflicts);
    token.addResponder(new mx.rpc.Responder(conflictSuccess, conflictFault));
}
```

keepClientObject

説明
(サーバーでデータ競合が発生しても) サーバーのオブジェクトでなく、クライアントオブジェクトが保持されるようにします。

また、この API では、保持されたクライアントオブジェクトは、同期時に new 操作としてサーバーに送信されません。

戻り値
keepClientObject 呼び出しに関連付けられた coldfusion.air.SessionToken インスタンス

シンタックス
public function keepClientObject(conflict:coldfusion.air.Conflict):SessionToken

パラメータ

パラメータ	説明
coldfusion.air.Conflict	サーバーで発生した競合

例
keepAllClientObjects セクションの例を参照してください。keepClientObject で唯一異なる点として、ユーザーは競合の AarrayCollection の各競合につき、自分で繰り返し処理を行う必要があります。

オフライン AIR SQLite API の機能強化
openSession に次の新しいパラメータがサポートされました。

新しいパラメータ	タイプ	必須 / オプション	説明
encryptionKey	ByteArray	オプション	オフライン SQLite データベースを暗号化する場合に使用します。詳細については、640 ページの「データベースの暗号化」を参照してください。
cacehDirectory	File	オプション	カスタムキャッシュディレクトリを指定する場合に使用します。詳細については、640 ページの「キャッシュディレクトリの指定」を参照してください。

SessionToken クラスのダイナミック化

ActionScript でダイナミックなクラスは、クラスのインスタンスに追加のキーと値のペアを追加できます。

今回のリリースでは、sessionToken はダイナミッククラスです。そのため、API の呼び出し元から取得可能な追加情報を、正常時または異常時のハンドラに追加できます。

例

```
private function fetchData():void
{
var token:AsyncToken= syncmanager.fetch("fetch");
token.addResponder(new mx.rpc.Responder(fetchSuccess, fetchFault));

                // Test For SessionToken class to be Dynamic, so that Dynamic Properties could be added
                token.userdefined_key = "value";
          }
        public function fetchSuccess(event:SyncResultEvent):void
            {

              if(event.token.userdefined_key == "value")
                { .... }
             }
```

autoCommit のサポート

SyncManager で、ブール値の autoCommit プロパティがサポートされています。

デフォルト値は false です。

true の場合、save、saveUpdate および remove メソッドが使用されると、ローカルデータベースの変更がサーバーにコミットされます。次に例を示します。

```
private var syncmanager:SyncManager = new SyncManager();
syncmanager.autoCommit = true;
```

この機能は、サーバーとの同期時の競合を最小限に抑えるのに役立ちます。特に、クライアントとサーバーサイドでプライマリキーを自動生成する場合などに有効です。

SessionResultEvent および SessionToken の新しい属性

SessionResultEvent および SessionToken クラスには、新しい autoGeneratedId 属性が含まれます。この属性には、ActionScript ORM で使用される自動生成 ID が挿入されます。autoGeneratedId は、その特定の呼び出しの ActionScript ORM によってキーが生成される場合にのみ挿入されます。

例

```
private function connectSuccess(event:SessionResultEvent):void
{
  // Generate an Order Obect
   .
   .
   .
   .
  // Save the Order
  var savetoken:SessionToken = session.save(ord);
  savetoken.addResponder(new mx.rpc.Responder(savesuccess, savefault));
}
private function savesuccess(event:SessionResultEvent):void
{
   // This is how, you can access autogenrated PK
   RememberINTPK = event.autoGeneratedId.toString();
   var loadtoken:SessionToken = session.loadByPK(Order,{oid:RememberINTPK},true);
   loadtoken.addResponder(new mx.rpc.Responder(loadsuccess,loadfailure))
}
```

注意： たとえば、プライマリキーはサーバーデータベースで生成されるが、クライアント SQLite テーブルにプライマリキーを生成したい場合を考えます (例を参照)。このシナリオでは競合が発生するので、アプリケーションの開発者が解決する必要があります。1 つの選択肢としては、クライアントオブジェクトとサーバーオブジェクト間の競合を最小限に抑えるようにアプリケーションを設計することです。この場合は、クライアントオブジェクトのプライマリキーを null または空の文字列で設定しておき、サーバーサイドの ORM EntitySave 関数を使用してデータをデータベースサーバーに保存します。

ColdFusion サービスのプロキシ ActionScript クラス

AIR および Flash の Flex ベースアプリケーションから ColdFusion のプロキシ ActionScript クラスを使用すると、一部の ColdFusion サービスにアクセスできます。この機能は、Flash および AIR で実行されるすべての Flex ベースアプリケーションで使用できます。ColdFusion には、cfchart、cfdocument、cfimage、cfmail、cfpdf、cfpop の各タグおよびそれらの子タグに対応するサービスが用意されています。ColdFusion を使用して、アプリケーションからサーバーにファイルをアップロードすることもできます。

ColdFusion には、次の Flex プロキシクラスとそれに関連するサポートクラスが用意されています。

- Config (ColdFusion サービスを使用できるようにアプリケーションを設定します)
- Util (ファイルのアップロードをサポートする機能を含みます)
- Chart (cfchart)
- Document (cfdocument)
- Image (cfimage)
- Mail (cfmail)
- PDF (cfpdf)
- Pop (cfpop)

これらのクラスは coldfusion.service.mxml パッケージの一部であり、cfservices.swc ファイルとして配布されています。通常、これらのクラスは、次に示すように、cf 名前空間識別子を使用して MXML タグ形式で使用します。

```
<cf:Image id="image" action="AddBorder" source="Uploaded Image server URL" thickness="5" color="Blue"/>
```

Flex でビルドされたアプリケーションで ColdFusion サービスを使用するには、Config クラスを使用して接続を確立した後に、その他のクラスを使用して ColdFusion サービスにアクセスします。

ColdFusion 9 では、すべてのプロキシタグに加えて、リモーティング宛先も Config クラスで指定できます。

注意：Flex および AIR から ColdFusion サービスを使用するには、「ColdFusion Web サービス」セクションの「ColdFusion サービスの有効化」で説明されているとおりに、サービスへのアクセスを有効にする必要があります。

"cfservices.swc" ファイルについて

Flex または AIR でビルドされたアプリケーションで ColdFusion サービスクラスを使用する手順は次のとおりです。

1. /CFIDE/scripts/AIR/cfservices.swc にある cfservices.swc ファイルをアプリケーションにインクルードします。
2. Flash Builder で、Flex/AIR プロジェクトを右クリックし、[プロパティ]-[Flex ビルドパス]-[ライブラリパス]-[SWC の追加] を選択して、プロジェクトに "cfservices.swc" ファイルを追加します。

すべての ColdFusion サービスクラスと coldfusion.air パッケージに含まれる他のクラスの詳細については、ActionScript 3.0 リファレンスを参照してください。また、ColdFusion Administrator の [情報源] ページの [マニュアル] リンクから、スタンドアローンの『Adobe ColdFusion リファレンスガイド』を参照することもできます。

Mail や Pop など、特定のサービスの属性については、対応する ColdFusion タグおよび関数の属性を参照してください。

Config クラスの使用

Config クラスは、接続の詳細やイベントハンドラなど、ColdFusion サービスの設定パラメータを設定する場合に使用します。したがって、Config クラスは、他のサービスクラスを使用する前に使用する必要があります。Config クラスで設定されたパラメータはグローバルに適用されますが、個々のサービスプロキシクラスでオーバーライドできます。

次の表に Config クラスのパラメータを示します (通常、これらのパラメータは Config タグの属性として使用します)。

属性	説明
serviceUserName	リクエストされている特定のサービスにアクセスするためのアクセス許可を持つ、ColdFusion Administrator で設定されたユーザー名です。
servicePassword	ColdFusion Administrator でユーザー名に対して設定されたパスワードです。
cfServer	CF サーバーのサーバー名または IP アドレスです。
cfPort	CF サーバーが実行しているポートです。
cfContextRoot	コンテキストルートです (CF サーバーにある場合)。
secureHTTP	サービスの実行に HTTP と HTTPS のどちらを使用するかを指定するブール値です。
destination	destination 属性を使用すると、"WEB-INF/flex/remoting-config.xml" でユーザー定義のリモーティング宛先を指定できます。指定しない場合は、ColdFusion のデフォルトの宛先が使用されます。

通常、config クラスを指定するには、次のように MXML タグを使用します。

```
<cf:Config id="conf" cfServer="CF Server IP Address/HostName" CFPort="HTTP Port on Which CF accepts request"
destination="UserDefinedRemotingDestination" >
```

また、次のように、サービスタグ内でサーバー設定を直接オーバーライドすることもできます。

```
<cf:Image id="image" action="AddBorder" cfServer="IP Address" cfPort="Port number" source="Uploaded Image
server URL" thickness="5" color="Blue" destination="UserDefinedRemotingDestination" />
```

Util クラスの使用

通常、Image、PDF、および Mail サービスは、サーバーにアップロードされたファイルに対して実行されます。CF サーバーにファイルをアップロードするには、Util クラスを使用して ColdFusion の Upload サービスを実行します。Util クラスは次の 2 つの要素から構成されます。

- UPLOAD_URL 定数 : ColdFusion サーバー上での Upload サービスの URL (ColdFusion の Web ルートディレクトリからの相対 URL) を含みます。
- extractURLFromUploadResponse() メソッド : Upload サービスから返されたレスポンスを入力として受け取り、ColdFusion サーバーにアップロードされたファイルのパスを返します。

次のワークフローで UPLOAD_URL 定数および extractURLFromUploadResponse() 関数を使用してファイルをアップロードし、そのファイルを ColdFusion サービスで使用します。

ColdFusion サービスのイベントフロー

1 ActionScript の flash.net.FileReference API および Util.UPLOAD_URL 変数を使用して、アクションの対象となるイメージ、PDF、またはメール添付ファイルをサーバーにアップロードします。flash.net.FileReference API に渡すアップロード URL は、アプリケーションの ActionScript 部分で次のように記述できます。

```
uploadURL.url = "http://"+conf.cfServer+":"+conf.cfPort+"/"+conf.contextRoot+"/"+Util.UPLOAD_URL;
var variables:URLVariables = new URLVariables();
    variables.serviceusername = conf.serviceUserName;
    variables.servicepassword = conf.servicePassword;
    uploadURL.data = variables;
    uploadURL.method="POST";
```

ここで、<cf:Config> タグを使用して "conf.cfServer"、"conf.cfPort" および "conf.ContextRoot" を指定します。ColdFusion が J2EE アプリケーションとしてデプロイされている場合は "conf.ContextRoot" のみを指定します。

注意：ActionScript の FileUpload 機能については詳しい説明が不足しているため、この機能の範囲外になりますが、MAIL クラスのセクションに掲載されているコードでその使用例を示しています。FileUpload 機能の詳細については、ActionScript のマニュアルを参照してください。

2 ファイルのアップロードが完了すると、アップロードされたファイルの URL を含む XML レスポンスがサーバーから返されます。Util クラス関数 extractURLFromUploadResponse() を使用して、この XML から URL を抽出します。

3 サービスタグの source 属性でファイルの URL を指定します。

4 必要なサービスタグの属性を設定し、次のメソッドを呼び出してサービスアクションを実行します。

```
serviceObject.execute()
```

5 アクションが成功すると、サーバーから結果が返されます。例外が発生した場合はエラーが返されます。サービスタグで指定したサービス固有の結果ハンドラおよびエラーハンドラ、または <cf:config> で指定したグローバルハンドラで ResultEvent オブジェクトおよび FaultEvent オブジェクトを処理します。ResultEvent オブジェクトには、操作の対象になったファイルの URL が含まれます。各ユーザーは、ローカルコンピュータにファイルをダウンロードするか、アプリケーション内でレンダリングすることにより、このファイルを保存できます。FaultEvent オブジェクトには、操作の実行時にサーバーで発生した例外の詳細が含まれます。

プロキシクラスをさらに細かく制御する場合は、プロキシクラスオブジェクトに対して getRemoteObject() メソッドを使用することで、基盤となる RemoteObject を取得できます。たとえば、<cf:Mail id="mailId"> の場合は、ActionScript で次のコードを使用して取得できます。

```
var mailobject:RemoteObject = mailId.getRemoteObject();
```

Mail クラスの使用

Mail クラスは ColdFusion メールサービスのプロキシであり、cfmail タグの機能を提供します。Mail タグの属性では、cfmail タグとその子タグで必須のアクション属性を指定します。このタグのデフォルトのメールアクションは send です。

次の AIR アプリケーションでは Mail サービスとファイルアップロード機能を使用しています。このアプリケーションは "CFCredentials.mxml" ファイルに含まれるユーザーの資格情報を参照して認証を行います。

この例で使用している "CFCredentials.mxml" ファイルの内容は次のとおりです。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml" currentState="collapsed" toolTip="Double Click to
Collapse/Expand" creationComplete="retrieveCredential()" headerHeight="5" layout="vertical"
mouseDown="check Collapse(event )" resizeEffect="Resize">
<mx:Script>
    <![CDATA[
    import mx.controls.Alert;
    import mx.rpc.http.HTTPService
    import mx.rpc.events.Result Event;
    import mx.rpc.events.FaultEvent;
    import flash.data.EncryptedLocalStore;
    import flash.utils.ByteArray;
    public var service:HTTPService = new HTTPService();
         private function testConnection () :void
        {
               var CF_HTTP URL:String;
               if(cfip.text == null)
               {
                 Alert.show("IP Not provided or Invalid IP Address");
               }
               else if(cfprt.text!= "" && cfcnxtrt.text != "")
               {CF_HTTPURL="http://"+cfip.text+":"+cfprt. text+"/"   +cfcnxtrt.text+"/flex2gateway/";
               }
               else if(cf prt.text == "" && cfcnxtrt.text == "")
               {
                  CF_HTTPURL ="http://"+cfip.text+"/flex2gateway/";
               }
               else if(cf prt.text!= "" && cfcnxtrt.text == "")
               {
                  CF_HTTPURL="http://"+cf ip.text+":"+cfprt.text+"/flex2gateway/";
               }

               if(cfserviceusername.text == "" && cfservicepassword.text == "")
                {
                 Alert.show("CF Service UserName and Password are not required to test
                CF server connectivity but they will be required while using the CF
                services","Note");

                }

                service.url = CF_HTTPURL;
                service.method = "POST";
                service.addEventListener(ResultEvent.RESULT,httpResult);
                service.addEventListener(FaultEvent.FAULT,httpFault);
                service.send();

         }

         public function httpResult(event:ResultEvent):void
        {
        Alert.show("Connection with ColdFusion server Successful","Connection Status");
        }
         public function httpFault(event:FaultEvent):void
```

```
        {
        Alert.show("ColdFusion server could not be reached, Make sure credentials are
        correct and CF server is running","Error");
         }
         private function checkCollapse(event:MouseEvent):void
        {
        if( event.clickCount == 2)
            {
                currentState = currentState == "collapsed" ? "":"collapsed";
            }
        }
        private function rememberCredential():void
        {
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes(cfip.text);
        En cryptedLocalStore.setItem('IP', data );
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes(cfprt.text);
        Encr yptedLocalStore.setItem('PORT', data );
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes(cfcnxtrt.text);
        Encrypt edLocalStore.setItem('CONTEXT', data );
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes(cfserviceusername.text);
        Encr yptedLocalStore.setItem( 'USER', data );
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes(cfservicepassword.text);
        Encrypted LocalStore.setItem('PASS', data );
        }
        private function retrieveCredential():void
        {
        try{
        cfip.text = EncryptedLocalStore.getItem('IP').toString();
        cfprt.text = EncryptedLocalStore.getItem('PORT').toString();
        cfcnxtrt.text = EncryptedLocalStore.getItem('CONTEXT').toString();
        cfserviceusername.text = EncryptedLocalStore.getItem('USER').toString();
        cfservicepassword.text = EncryptedLocalStore.getItem(' PASS').to String();
        }
        catch(e:Error)
        {
        }
}
          private function resetCredential():void
           {
             EncryptedLocalStore.reset();
             cfip.text = "";
             cfprt.text = "";
             cfcnxtrt.text  = "";
             cfserviceusername.text  = ""  ;
             cfservicepassword.text = "";
           }

    ]]>
</mx:Script>
<mx:ControlBar>
```

```
    <mx:Label text="CFServer IP"/>
    <mx: TextInput id="cfip" text="" width="   70"/>
    <mx:Label text="CFServer Port"/>
    <mx:TextInput id="cfprt" text="" width="40"/>
    <mx:Labeltext="CFServer Context Root (if any)"/>
    <mx:TextInput id="cfcnxtrt" text="" width="70"/>
    <mx:Label text="CFService UserName"/>
    <mx:TextInput id= "cfserviceusername" text="" width="70"/>
    <mx:Label text="CFService Password"/>
    <mx:TextInput displayAsPassword="true" id="cfservicepassword" text="" width="70"/>
    <mx:Button id="testconn" label="Test Connection" click="testConnection()"/>
    <mx:Button id="save" label=" Remember"click="rememberCredential()"/>
    <mx:Button id="reset" label ="Reset" click = "resetCred e n tial()"/>
</mx:ControlBar>
<mx:states>
    <mx:State name="collapsed">
        <mx: SetProperty  name="height" value="10"/>
    </mx:State>
</mx:states>
</mx:Panel>
```

AIR アプリケーションの例を次に示します。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:local="com.*" creationComplete="init()" xmlns:cf="coldfusion.service.mxml.*">
<mx:Script>
<![CDATA[
import coldfusion.service.events.*;
import com.CFCredential;
import mx.collections.ArrayCollection;
import mx.binding.utils.BindingUtils;
import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;
import mx.controls.Alert;
import flash.events.*;
import flash.net.FileReference;
import flash.net.FileReferenceList;
import flash.net.URLRequest;
import flash.net.URLVariables;
import coldfusion.service.Util;
    public var fileTypes:Array = new Array();
    public var imageTypes:FileFilter = new FileFilter("Images (*.jpg; *.jpeg; *.gif; *.png)" ,"*.jpg;
*.jpeg; *.gif; *.png");
    public var documentTypes:FileFilter = new FileFilter("Documents (*.pdf), (*.doc), (*.rtf),
(*.txt)",("*.pdf; *.doc; *.rtf, *.txt"));
    [Bindable]
        private var fileslist:ArrayCollection;
        private var filereflist:FileReferenceList
        public var fileRef:FileReference = new FileReference();
    [Bindable]
    public var mailPartArray:Array = [{type:"text",content:"Plain text only"},
{type:"html",content:"<B>bold text man!!</B>"}];
    public var uploadURL:URLRequest = new URLRequest();
    [Bindable]
    public var attachCollection:Array = new Array();
    public var urlcnt:int=0;
    public function init():void
    {
    fileslist = new ArrayCollection();
    filereflist = new FileReferenceList;
    fileRef = new FileReference;
    uploadURL.url = "http://"+conf.cfServer+":"+conf.cfPort+"/"+conf.contextRoot+"/"+Util.UPLOAD_URL;
```

```
    var variables:URLVariables = new URLVariables();
    variables.serviceusername = conf.serviceUserName;
    variables.servicepassword = conf.servicePassword;
    uploadURL.data = variables;
    uploadURL.method = "POST"; //this can also be set to "POST" depending on your needs
            uploadURL.contentType = "multipart/form-data";
            fileTypes.push(imageTypes);
            fileTypes.push(documentTypes);
        //Add Event Listeners to UI
        attachbutton.addEventListener(MouseEvent.CLICK, browseFiles);
        sendbutton.addEventListener(MouseEvent.CLICK,uploadFiles);
        filereflist.addEventListener(Event.SELECT, selectHandler);
        //mailtest.send();
        }

//Browse for files
    private function browseFiles(event:Event):void
        {
            filereflist.browse(fileTypes);
        }

//  called after user selects files form the browse dialogue box.
    private function selectHandler(event:Event):void
        {var i:int;
            for (i=0;i < event.currentTarget.fileList.length; i ++)
            {
                fileslist.addItem(event.currentTarget.fileList[i]);
                attachList.text +=  event.currentTarget.fileList[i].name + ",  ";
            }
        }
    private function uploadFiles(event:Event):void
    {
        if (fileslist.length > 0)
        {
            fileRef = FileReference(fileslist.getItemAt(0));
            fileRef.addEventListener(Event.COMPLETE, completeHandler);
            fileRef.addEventListener(DataEvent.UPLOAD_COMPLETE_DATA,dataHandler);
            fileRef.upload(uploadURL);
        }
        else if (fileslist.length == 0)
            {
                sendmail();
            }
        }

// called after a file has been successfully uploaded | We use this as well to check if there are any files
left to upload and how to handle it
    private function completeHandler(event:Event):void
        {
            // Alert.show("File Uploaded successfully");
            fileslist.removeItemAt(0);
            if (fileslist.length > 0)
            {
                uploadFiles(null);
            }
        }
//called after file upload is done and Data has been returned from Server
    private function dataHandler(event:DataEvent):void
    {
        attachCollection[urlcnt++] = {"file":
        Util.extractURLFromUploadResponse(event.data.toString())};
        if (fileslist.length == 0)
```

```
            sendmail();
    }
    private function sendmail():void
    {
        mailtest.execute();
    }
    public function handleResult(event:ColdFusionServiceResultEvent):void
    {
        from.text="";too.text=""; cc.text="";bcc.text="";
        subject.text="";mailbody.text="";
        attachList.text=""; fileslist.removeAll();
        Alert.show("Mail was delivered Successfully");
    }
    public function handleError(event:ColdFusionServiceFaultEvent):void
    {
        Alert.show("Failure"+ event.toString());
    }
    ]]>
</mx:Script>
<mx:Panel width="100%" height="100%">
    <local:CFCredential id="cred" />
    <mx:ControlBar>
    <mx:Spacer width="100%"/>
    <mx:HBox>
        <mx:Button label="Send Mail" id="sendbutton"/>
        <mx:Button label="Attachment" id="attachbutton"/>
    </mx:HBox>
    </mx:ControlBar>
<mx:VBox width="100%" height="100%">
    <mx:HBox width="100%">
    <mx:Text text="From" width="100" />
    <mx:TextInput width="100%" id="from"/>
    </mx:HBox>
    <mx:HBox width="100%">
    <mx:Text text="To" width="100"/>
    <mx:TextInput width="100%" id="too"/>
    </mx:HBox>
    <mx:HBox width="100%">
    <mx:Text text="CC" width="100"/>
    <mx:TextInput width="100%" id="cc"/>
    </mx:HBox>
    <mx:HBox width="100%">
    <mx:Text text="Bcc" width="100"/>
    <mx:TextInput width="100%" id="bcc"/>
    </mx:HBox >
    <mx:HBox width="100%">
    <mx:Text text="Subject" width="100"/>
    <mx:TextInput width="100%" id="subject"/>
    </mx:HBox>
    <mx:HBox width="100%">
    <mx:Text text="Attachments" width="100"/>
    <mx:TextInput width="100%" id="attachList"  enabled="false"
    backgroundDisabledColor="white"/>
    </mx:HBox>
    <mx:TextArea width="100%" height="100%" id="mailbody"/>
</mx:VBox>
</mx:Panel>

    <!--Provide your CF server credentials here-->
        <cf:Config id="conf"
```

```
        cfServer="{cred.cfip.text}"
        cfPort="{int(cred.cfprt.text)}"
        cfContextRoot="{cred.cfcnxtrt.text}"
        servicePassword="{cred.cfservicepassword.text}"
        serviceUserName="{cred.cfserviceusername.text}"
    />

    <cf:Mail id="mailtest"
        server="xx.xxx.xx.xxx"
        to="{too.text}" bcc="{bcc.text}" cc="{cc.text}"
        failTo="jayesh@adobe.com" replyTo="jayesh@adobe.com"
        subject="{subject.text}" content="{mailbody.text}"
        from="{from.text}"
        attachments="{attachCollection}"
        type="text" charset="utf-8" mailerId="CF" priority="1"
        timeout="60" useTLS="true" wrapText="5"
        result="handleResult(event)"
        fault="handleError(event)"
    />
</mx:WindowedApplication>
```

Image クラスの使用

Image クラスは ColdFusion イメージサービスのプロキシであり、cfimage タグの機能を提供します。Image タグの属性では、cfimage タグで必須のアクション属性を指定します。次に、一般的な使用方法を示します。この例では、境界線を追加しています。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
xmlns:cf="coldfusion.service.mxml.*"  creationComplete="init()">
<mx:Script>
<![CDATA[
import mx.controls.Alert;
import coldfusion.service.events.*;

function init():void
    {
        img.execute();
    }

function handleResult(event:ColdFusionServiceResultEvent):void
    {
        mx.controls.Alert.show("result=" + event.result.toString());
        retImage.source = event.result.toString();
    }

function handleError(event:ColdFusionServiceFaultEvent):void
    {
        mx.controls.Alert.show(event.toString());
    }
]]>
</mx:Script>
```

```
<cf:Config
    id="configid"
    cfServer="127.0.0.1"
    cfPort="80"
    servicePassword="service"
    serviceUserName="service"/>

<!-- Add border-->
<cf:Image
    id="img"
    action="addborder"
    color="red"
    thickness="5"
    source="http://127.0.0.1:80/GetExifmetaData.jpg"
    result="handleResult(event)"
    fault="handleError(event)"/>

<mx:Image id="retImage"/>
</mx:Application>
```

バッチ操作

バッチ操作を使用すると、個々の操作を 1 つずつ実行する代わりに、複数のイメージ操作を一度に実行できます。サーバーにイメージをアップロードし、バッチ操作を使用してそのイメージに対して複数の操作を実行し、変更後のイメージをサーバーから取得できます。このアクションを使用するには、アクションの連想配列と、それに対応する属性を使用します。連想配列は、MXML 内で作成するよりも ActionScript 内で作成するほうが簡単です。

次の例は、アクション情報を含む連想配列を作成してバッチ操作を実行するコードを示しています。

ActionScript 部分 :

```
[Bindable]
public var attributes:Array =
[{AddBorder:{color:"Red",thickness:"50"}},
{Resize:{width:"50%",height:"50%",interpolation:"blackman",blurfactor:"2"}},
{Flip:{transpose:"270"}}]
```

MXML 部分 :

```
<!-- batch operation --
<cf:Image id="img" action="batchoperation"
source="http://localhost:8500/cat.jpg"
attributes="{attributes}"/>
```

PDF クラスの使用

Pdf クラスは ColdFusion PDF サービスのプロキシであり、cfpdf タグの機能を提供します。Pdf タグの属性では、cfpdf タグで必須のアクション属性を指定します。次の例で、サポートされるアクションを示します。

```
<!-- Get Info-->
<cf:Pdf id="pdftest" action="GETINFO"
source="http://localhost:8500/lcds26_devguide_040908.pdf"/>

<!-- Delete Pages-->
<cf:Pdf id="pdftest1" action="deletepages"
source="http://localhost:8500/lcds26_devguide_040908.pdf"
pages="1" resultHandler="handleDeleteResult"
errorHandler="handleDeleteError"/>

<!-- Merge files--><cf:Pdf id="pdftest" action="mergeFiles"
source="http://localhost:8500/lcds26_devguide_040908.pdf,
http://localhost:8500/EC205W_JoelGeraci.pdf"
resultHandler="handleMergeResult"
errorHandler="handleMergeError"/>

<!-- extract pages-->
<cf:Pdf id="pdftest" action="extractpages"
source="http://localhost:8500/lcds26_devguide_040908.pdf"
pages="2" keepBookmark="true"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- addwatermark-->
<cf:Pdf id="pdftest" action="addwatermark"
source="http://localhost:8500/1page.pdf"
image="http://localhost:8500/IMG_8680.JPG"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- removewatermark-->
<cf:Pdf id="pdftest" action="removewatermark"
source="http://localhost:8500/CFFileServlet/
_cfservicelayer/_cf3466030530070122606.pdf"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- protect-->
<cf:Pdf id="pdftest" action="protect"
source="http://localhost:8500/1page.pdf"
newUserPassword="test" permissions="All"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- mergespecificpages-->
<cf:Pdf id="pdftest" action="mergespecificpages"
```

```
pdfParam="{pdfparams}" keepBookmark="true"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- set info-->
<cf:Pdf id="pdftest" action="setinfo"
source="http://localhost:8500/1page.pdf" info="{elements}"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- thumbnail-->
<cf:Pdf id="pdftest" action="thumbnail"
source="http://localhost:8500/EC205W_JoelGeraci.pdf"
resultHandler="handleThumbnailResult"
errorHandler="handleThumbnailError"/>

<!-- ProcessDDX-->
<cf:Pdf id="pdftest" action="processddx"
ddxString="{ddx}" outputFiles="{outputFiles}" result="handleProcessDDXResult"
fault="handleProcessDDXError"/>
```

Chart クラスの使用

Chart クラスは ColdFusion チャートサービスのプロキシであり、cfchart タグとその子タグである chartdata および chartseries タグの機能を提供します。

- チャートオブジェクトのプロパティでは、cfchart タグの属性を指定します。
- チャート系列は、チャートオブジェクトの chartSeries 要素として表されます。chartSeries 要素はオブジェクトの配列であり、それぞれが単一のチャート (chartseries タグ) ドキュメントセクションを表します。これらのオブジェクトには、チャートタイプを指定する type 要素、チャートデータを指定する chartdata 要素、その他の系列属性を指定する要素などが含まれます。
- 各チャートのデータはオブジェクトの配列として表され、各オブジェクトには item 要素と value 要素が含まれます。これらの配列を chartSeries オブジェクトの chartdata 要素として使用します。
- このサービスを実行するには、document object execute() 関数を呼び出します。

次の例で、チャートサービスの使用方法を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" xmlns:cf="coldfusion.service.mxml.*" creationComplete="init()">
<mx:Script>
    <![CDATA[
     import mx.controls.Alert;
     import mx.rpc.events.FaultEvent;
     import mx.rpc.events.ResultEvent;

    [Bindable]
    public var chaSer:Array;
    public var chaDat:Array;
     function init():void
     {
     chaDat =
     [{item:"Facilities",value:"35000"},
     {item:"Facilities1",value:"32000"}];
     chaSer = [{type:"bar",chartdata:chaDat},
     {type:"line",chartdata:chaDat}];chartest.execute();
     }
            function handleResult(event:ResultEvent):void
            {
```

```
            Alert.show("success" + event.result.toString());
            }
            function handleFault(event:FaultEvent):void
            {
            Alert.show("failure" + event.toString());
            }
    ]]>
</mx:Script>
    <cf:Config id="configid" cfServer="localhost"
    cfPort="80" servicePassword="service"
    serviceUserName="service" >
    </cf:Config>
<cf:Chart id="chartest"
        action="generate"
        format="jpg"
        xAxisTitle="Department"
        yAxisTitle="Salary Average"
        chartSeries="{chaSer}"
        result="handleResult(event)"
        fault="handleFault(event)"
        backgroundColor = "Black"
        chartHeight = "500"
        chartWidth = "600"
        dataBackgroundColor = "yellow"
        font = "ariel"
        fontBold = "yes"
        fontItalic = "yes"
        fontSize = "12"
        foregroundColor = "red"
        gridLines = "2"
        labelFormat = "number"
        markerSize = "10"
        showBorder = "yes"
        showLegend = "yes"
        showMarkers = "yes"
        showxGridLines="yes"
        showyGridLines="yes"
        tipBgColor="blue"
        tipStyle = "MouseOver"
        title = "AIR Integration testing"/>
</mx:Application>
```

Document クラスの使用

Document クラスは ColdFusion ドキュメントサービスのプロキシであり、cfdocument タグとその子タグである cfdocumentsection および cfdocumentitem タグの機能を提供します。

- ドキュメントオブジェクトのプロパティでは、cfdocument タグの属性を指定します。どのセクションにも属さないドキュメントコンテンツはドキュメントオブジェクトの content 要素として指定します。
- ドキュメントセクションは、ドキュメントオブジェクトの documentSection 要素として表されます。documentSection 要素はオブジェクトの配列であり、それぞれが単一のドキュメントセクションを表します。これらのオブジェクトには、セクションコンテンツを指定する content 要素、ドキュメントアイテムを指定する documentItem 要素、その他のセクション属性を指定する要素などが含まれます。
- ドキュメントセクション (またはドキュメントオブジェクト) 内のすべてのドキュメントアイテムは、type 要素および content 要素を持つオブジェクトの配列として表されます。配列要素の type フィールドでは、そのアイテムがヘッダ、フッタ、改ページのいずれであるかを指定します。ドキュメントアイテムの配列は、ドキュメントオブジェクトまたは documentSection オブジェクトの documentItem 要素として指定します。

- このサービスを実行するには、document object execute() 関数を呼び出します。

次の例は、セクションおよびアイテムを作成してドキュメントに追加する方法を示す完全なコードからの抜粋です。

```
[Bindable]
var docItem:Array = [{type:"header",content:"<font size='-3'>
<i>Salary Report</i></font>"},{type:"footer",
content:"<font size='-3'>
Page #cfdocument.currentpagenumber#</font>"}];

[Bindable]var docSectionItem:Array = [{content:"<table width='95%'
border='2' cellspacing='2' cellpadding='2' >
<tr><th>Salary</th></tr><tr>
<td><font size='-1'>John</font></td>
<td align='right'><font size='-1'>Guess What</font></td></tr>
<tr><td align='right'><font size='-1'>Total</font></td>
<td align='right'><font size='-1'>Peanuts</font></td></tr>",
documentitem:docItem},{content:"content2",documentitem:docItem}];
.
.
.
cfDoc.documentSection = docSectionItem;
```

次の例は、ドキュメントの一般的な使用方法を示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical" xmlns:cf="coldfusion.service.mxml.*"
creationComplete="init()">
<mx:Script>
    <![CDATA[
    import mx.controls.Alert;
    import mx.rpc.events.ResultEvent;
    import coldfusion.service.PdfParam;

    [Bindable]
    var docItem:Array = [{type:"header",content:"
    <font size='-3'><i>Salary Report</i></font>"},
    {type:"footer",content:"<font size='-3'>
    Page <cfoutput>#cfdocument.currentpagenumber#
    </cfoutput></font>"}];

    [Bindable]
    var docSection:Array =
    [{content:"content1"},{content:"content2"},
    {content:"content3"}];

    [Bindable]
    var docSectionItem:Array =
    [{content:"content1",documentitem:docItem},
    {content:"content2",documentitem:docItem},
    {content:"content3",documentitem:docItem}];
    [Bindable]
    var res:String = new String();
    private function init():void
        {
            doctestnow.execute();
        }
    private function handleResult(event:ResultEvent):void
        {
            res=event.result.toString();
            //Alert.show("httpurl= "+event.result.toString());
        }
```

```
        private function handleError(event:Event):void
        {
            mx.controls.Alert.show(event.toString());
        }
        ]]>
</mx:Script>
<cf:Config id="configid" cfServer="localhost"
cfPort="80" servicePassword="service" serviceUserName="service" />

        <!-- simple case-->
        <cf:Document id="doctestnow" action="generate"
        format="flashpaper" result="handleResult(event)"
        fault="handleError(event)"
        content="&lt;table&gt;&lt;tr&gt;&lt;td&gt;bird&lt;/td&gt;&lt;td&gt;
        1&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;fruit&lt;/td&gt;&lt;
        td&gt;2&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;rose&lt;/td&gt;
        &lt;td&gt;3&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt;"/>
        <!--doc item case -->
        <!--<cf:Document id="doctestnow" action="generate"
        format="flashpaper" result="handleResult(event)"
        fault="handleError(event)" documentItem="{docItem}"/>-->
        <!-- doc section case-->
        <!--<cf:Document id="doctestnow" action="generate"
        format="flashpaper" result="handleResult(event)"
        fault="handleError(event)" documentSection="{docSection}"/>-->

        <!-- doc section and doc item case
        <cf:Document id="doctestnow" action="generate"
        format="flashpaper" result="handleResult(event)"
        fault="handleError(event)" documentSection="{docSectionItem}" />-->
        <mx:SWFLoader source="{res}"/>
</mx:Application>
```

Pop クラスの使用

Pop クラスは、ColdFusion Pop サービスのプロキシであり、cfpop タグの機能を提供します。Pop オブジェクトのプロパティでは cfpop タグで必須のアクション属性を指定します。このサービスを実行するには、object execute() 関数を呼び出します。次の例で、サポートされるユーザーアクションを示します。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:cf="coldfusion.service.mxml.*" creationComplete="init()">
<mx:Script>
<![CDATA[
import mx.rpc.events.ResultEvent;
import mx.rpc.events.FaultEvent;
import mx.controls.Alert;
import coldfusion.service.events.*;
    public function init():void
        {
            poptest.execute();
        }
    public function handleResult(event:ResultEvent):void
        {
            Alert.show("Success" + event.toString());
        }
    public function handleError(event:FaultEvent):void
        {
            Alert.show("Failure");
        }
    public function handleGetAll(event:ResultEvent):void
```

```
        {
            var res:Array = event.result as Array;
            for(var i:uint = 0; i < res.length; i++)
            {
                var key:String;
                for(key in res[i])
                {
                    trace("object key = " + key.toString());
                    if(res[i][key] != null)
                    {
                        trace("object value = " + res[i][key].toString());
                    }
                }
            }
        }
    ]]>
</mx:Script>

<cf:Config id="configid" cfServer="localhost" cfPort="8500"
servicePassword="service" serviceUserName="service"  />

<!--<cf:Pop id="poptest" action="getall" result="handleGetAll(event)"
host="xx.xxx.xx.xxx" userName="failoveruser" password="password"
fault="handleError(event)"/>-->

<!--<cf:Pop id="poptest" action="getheaderonly" result="handleGetAll(event)"
host="xx.xxx.xx.xxx" userName="failoveruser" password="password"
fault="handleError(event)"/>-->

<cf:Pop id="poptest" action="delete" messageNumber="25" host="xx.xxx.xx.xxx"
userName="failoveruser" password="password" result="handleResult(event)"
fault="handleError(event)" />
</mx:Application>
```

LiveCycle Data Services ES アセンブラの使用

Adobe ColdFusion を Adobe Flex アプリケーションのバックエンドデータマネージャとして使用する場合は、Adobe LiveCycle Data Services ES アセンブラを使用します。LiveCycle Data Services ES アセンブラを設定し、そのアセンブラを使用するアプリケーションを作成します。

LiveCycle Data Services ES と ColdFusion を併用するには、ColdFusion コンポーネントに関する知識や、ColdFusion アプリケーションでのデータのアクセスおよび使用に関する知識、LiveCycle Data Services ES の使用に関する知識が必要です。

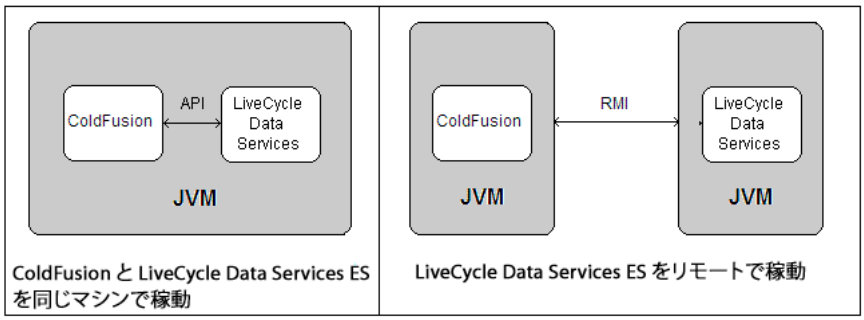
ColdFusion と Flex について

ColdFusion 9 では LiveCycle Data Services 2.6.1 がサポートされていますが、ColdFusion のインストーラには LiveCycle Data Services をインストールするオプションが用意されていません。ColdFusion で LiveCycle Data Services を使用するには、このサービスを手動でインストールする必要があります。

デフォルトでは、ColdFusion と同時に BlazeDS がインストールされます。これにより、ColdFusion でのメッセージングサポートが可能になります。

Flash Remoting の使用に関連する変更点については、ColdFusion 9 および ColdFusion 9.0.1 における Flash Remoting 用の XML 設定ファイルの変更点を参照してください。

LiveCycle Data Services ES アセンブラを使用すると、Data Management Service を使用する Flex アプリケーションのバックエンドデータを、ColdFusion コンポーネント (CFC) で管理できます。LiveCycle Data Services ES は、ColdFusion の一部として実行することも、リモートで実行することもできます。LiveCycle Data Services ES を ColdFusion の一部として実行する場合、LiveCycle Data Services ES と ColdFusion は直接通信を行います。LiveCycle Data Services ES をリモートで実行する場合、LiveCycle Data Services ES と ColdFusion は RMI を使用して通信を行います。次の図に、それぞれの場合の ColdFusion と LiveCycle Data Services ES の通信方法を示します。



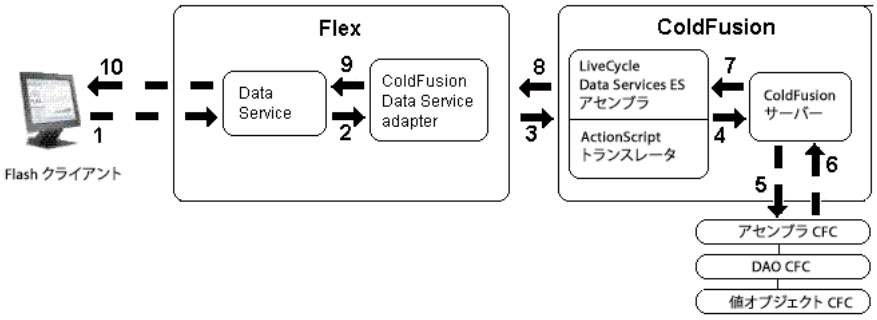
注意：LiveCycle Data Services ES アセンブラを使用するには、Flex アプリケーションを Flex Data Services 2.0.1 または LiveCycle Data Services 2.5 上で実行する必要があります。ただし、Flex Data Services 2.0.1 ではすべての機能はサポートされません。

Flex サーバーには ColdFusion Data Service Adapter が組み込まれています。このアダプタは、データに対する変更を処理して、クライアントのデータとバックエンドのデータを同期します。sync、fill、count、get の実行、競合の検出、LiveCycle Data Services ES への結果の受け渡しを行います。

ColdFusion には LiveCycle Data Services ES アセンブラが組み込まれています。このアセンブラは、ActionScript トランスレータと連携して、必要に応じて入力引数や戻り値を変換します。

注意：LiveCycle Data Services ES をインストールすると、ColdFusion による .SWF ファイルのマッピングが行われません。つまり、すべての SWF ファイルは、Web サーバーでなく、ColdFusion Web アプリケーションを介して処理されます。

次の図に、Flex アプリケーションが ColdFusion コンポーネントのメソッドを呼び出したときに実行される、LiveCycle Data Services ES と ColdFusion のプロセスを示します。



1 Flash クライアントが LiveCycle Data Management Service Adapter にデータを要求します。

2 Flex が、Data Service の fill、sync、get、または count メソッドを呼び出します。

3 LiveCycle Data Services ES をリモートで実行している場合、"lcds2.6_install/resources/lib/flex-*.jar" ファイルを "<ColdFusion のホームディレクトリ >/lib/" ディレクトリ (スタンドアローン設定の場合) または "WEB-INF/cfusion/lib" ディレクトリ (J2EE 設定の場合) にコピーします。

4 ColdFusion Data Service Adapter が、LiveCycle Data Services ES アセンブラにリクエストを送信します。LiveCycle Data Services ES をリモートで実行している場合、アダプタは Java RMI (Remote Method Invocation) を使用してリクエストを送信します。

5 LiveCycle Data Services ES アセンブラと ActionScript トランスレータが、ActionScript 3.0 のデータ型を適切な ColdFusion の値に変換します。

6 ColdFusion サーバーが、アセンブラ CFC の fill、sync、get、または count メソッドを実行します。これによって、DAO CFC の適切なメソッドが実行されます。

7 ColdFusion アプリケーションが、値オブジェクトの配列または適切な戻り値を作成して、ColdFusion サーバーに送信します。

8 ColdFusion サーバーが、LiveCycle Data Services ES アセンブラに結果を送信します。

9 LiveCycle Data Services ES アセンブラと ActionScript トランスレータが、ColdFusion の値を適切な ActionScript 3.0 のデータ型に変換し、アセンブラが結果を ColdFusion Data Service Adapter に送信します。

10 ColdFusion Data Service Adapter が、LiveCycle Data Management Service に結果を送信します。

11 LiveCycle Data Management Service が、Flash クライアントに結果を渡します。

注意： RMI レジストリ (ColdFusion Data Service アセンブラとリモートの LiveCycle Data Management Service の間で通信を行うために必要) では、ポート 1099 が使用されます。これは Java RMI のデフォルトポートです。ポート番号を変更するには、ColdFusion サーバーと Flex サーバーの両方で、Java JVM 引数に -Dcoldfusion.rmiport=1234 を追加します。

アプリケーションの開発およびデプロイのプロセス

ColdFusion Data Service Adapter と LiveCycle Data Services ES アセンブラを使用してバックエンドデータベースを操作する Flex アプリケーションを開発してデプロイするための一般的なプロセスを次に示します。

1 アプリケーションを設計します。

2 Flex アプリケーションを作成し、DataService コンポーネントを MXML または ActionScript で定義します。DataService コンポーネントでは、サーバーサイドの Data Management Service のメソッドを呼び出すことで、さまざまなアクティビティを実行します。たとえば、リモートデータソースからデータを取得してクライアントサイドのデータコレクションに設定したり、クライアントサイドとサーバーサイドでデータを同期したりします。

3 Flex アプリケーションが ColdFusion のバックエンドアプリケーションに接続できるように、ColdFusion Data Service Adapter の宛先を設定します。詳細については、662 ページの「ColdFusion Data Service Adapter の宛先の設定」を参照してください。

4 ColdFusion CFC を記述します。詳細については、670 ページの「ColdFusion CFC の記述」を参照してください。

注意： ColdFusion に付属している Flash Builder 用のウィザードを使用すれば、CFC を簡単に作成できます。詳細については、1322 ページの「Eclipse 用の ColdFusion Extension の使用」を参照してください。

5 Flex を使用してアプリケーションをテストします。

ColdFusion Data Service Adapter の宛先の設定

Flex アプリケーションが ColdFusion のバックエンドアプリケーションに接続するために必要な情報を提供するには、宛先 (destination) を設定します。destination では、ColdFusion Data Service Adapter、宛先との間でメッセージを転送するために使用するチャネル、fill、get、sync、count メソッドを含む CFC などの設定を指定します。

設定情報を指定するには、次のファイルを編集します。

1 "services-config.xml"

このファイルでは、チャネル定義を指定し、Flex コンソールでの ColdFusion 固有のデバッグ出力を有効にします。LiveCycle Data Services ES を使用して複数の ColdFusion インスタンスを実行する場合は、RTMP チャネル用に "services-config.xml" ファイルでポート番号を変更します。

2 "data-management-config.xml"

このファイルは、LiveCycle Data Services 2.6.1 を手動でインストールした場合にのみ追加されます。このファイルでは、アダプタと宛先を指定します。

Flex に LiveCycle Data Services ES アセンブラを認識させ、宛先との間でメッセージを送受信できるようにするには、次の手順を行います。

- ColdFusion 固有のチャネル定義の指定
- ColdFusion Data Service Adapter の指定
- 宛先の指定
- ColdFusion 固有のデバッグ出力の有効化

Flash Remoting の機能強化

ColdFusion 9 では、機能強化された Flash Remoting が導入されています。

- 新しい Flash Remoting では、以前の Flash Remoting (ColdFusion 8) でサポートされていないオブジェクトの循環参照がサポートされています。
- 新しい Flash Remoting は、以前の Flash Remoting よりも大幅に高速化されています。

デフォルトの設定で ColdFusion 9 をインストールした場合は、新しい Flash Remoting が使用されます。"WEB-INF/flex" ディレクトリにある XML 設定ファイルの構造は、新しい Flash Remoting をサポートするように変更されています。

ColdFusion 9 では以前の XML 設定ファイルもサポートされていますが、これらのファイルでは新しい Flash Remoting が使用されません。新しい Flash Remoting を利用するには、以前の XML ファイルを ColdFusion9 に移行する際に、これらのファイルが新規 XML 構造の変更点に準拠していることを確認する必要があります。

ColdFusion 9 で LCDS を使用している場合、Flash Remoting は LCDS 2.6.1 で動作します。新しい Flash Remoting を使用するには、LCDS 2.6.1 を使用していることを確認してください。

ColdFusion 9 では、LCDS2.5 以前のリリースもサポートするため、以前の Flash Remoting との下位互換性も確保されています。

引き続き LCDS 2.5 以前のバージョンを使用する必要がある場合は、ColdFusion 9 で導入された新しい Flash Remoting を使用できません。この場合は、従来の XML 設定ファイルを使用して LCDS2.5 以前のバージョンを引き続き利用できます。

LCDS 2.5 で従来の Flash Remoting を使用するには、最初に "cfusion/lib" から既存の "ColdFusion 9 flex-*.jar" ファイルを削除します。これを行うには、ファイルのバックアップを作成し、"LCDS2.5 flex-*.jar" ファイルを "cfusion¥lib" ディレクトリに配置します。この後、従来の (ColdFusion 8 の) XML 設定ファイルを "WEB-INF¥Flex¥" ディレクトリに配置することで、引き続きこれらのファイルを使用できます。また、"WEB-INF¥flex¥jar¥cfdataserviceadapter.jar" が存在することも確認してください。

LCDS2.5 と ColdFusion の統合に関する詳細な手順については、www.adobe.com/jp/ のテクニカルノートを参照してください。

場合によっては、LCDS 2.6.1 を ColdFusion 9 に統合した後も、従来の Flash Remoting が必要になるケースがあります。このシナリオは実装可能ですが、その場合は新しい Flash Remoting の利点を活用できません。

ColdFusion 9 および ColdFusion 9.0.1 における Flash Remoting 用の XML 設定ファイルの変更点

ColdFusion 9 では、"services-config.xml" ファイルの構造が変更されています。構造上の変更点は次のとおりです。

- 新しい <coldfusion> タグが <channel-definition> の <properties> の下に追加されました。ここでは、<access> タグ、<use-accessor> タグ、<use-structs> タグ、および <property-case> タグが定義されています。

  ColdFusion 8 方式の Flash Remoting の場合、これらのタグは "data-management-config.xml" ファイル内の定義に含まれていました。

- 以前は、<serialization> タグに次のものが含まれていました。

```
<serialization>
    <instantiate-types>false</instantiate-types>
</serialization>
```

  現在は、<instantiate-types> を true に設定するか、"services-config.xml" ファイルからこのタグを削除する必要があります。

- serialization プロパティで <enable-small-messages> フラグを false に設定する必要があります。

  注意：クライアントサイドでカスタムのチャネル定義を作成して、XML ベースのチャネル設定を無効にする場合も、enableSmallMessages フラグを false に設定する必要があります。その場合は、次のようなコードを使用します。

```
<mx:ChannelSet id="myChannelSet">
<mx:channels>
<mx:AMFChannel enableSmallMessages="false" url="http://localhost:8500/flex2gateway/cfamfpolling"
id="cfAMFPolling" pollingEnabled="true" pollingInterval="8"/>
</mx:channels>
</mx:ChannelSet>
```

- ColdFusion 9 では、エンドポイントクラスの名前が ColdFusion 8 から変更されています。次の表に、チャネル定義とそれに対応するエンドポイントクラスを示します。

チャネル定義 ID	ColdFusion 9 のエンドポイントクラス	ColdFusion 8 のエンドポイントクラス
my-cfamf	coldfusion.flash.messaging.CFAMFEndPoint	flex.messaging.endpoints.AMFEndpoint
cf-polling-amf	coldfusion.flash.messaging.CFAMFEndPoint	flex.messaging.endpoints.AMFEndpoint
my-cfamf-secure	coldfusion.flash.messaging.SecureCFAMFEndPoint	flex.messaging.endpoints.SecureAMFEndpoint
cf-rtmp	coldfusion.flash.messaging.CFRTMPEndPoint	flex.messaging.endpoints.RTMPEndpoint

  注意： LiveCycle Data Services ES では、チャネル定義 cf-polling-amf および cf-rtmp が使用されます。

- ColdFusion 9.0.1 は BlazeDS 4 をサポートします。

  次の表に BlazeDS のエンドポイントクラスの詳細を示します。

チャネル定義 ID	エンドポイントクラス
my-streaming-amf	coldfusion.flash.messaging.CFStreamingAMFEndPoint
secure-streaming-amf	coldfusion.flash.messaging.SecureCFStreamingAMFEndPoint

- ColdFusion 9.0.1 は LCDS 3 および LCDS 3.1 をサポートします。

  次の表に LCDS のエンドポイントクラスの詳細を示します。

チャネル定義 ID	エンドポイントクラス
my-nio-amf	coldfusion.flash.messaging.CFNIOAMFEndPoint secure-nio-amf
secure-nio-amf	coldfusion.flash.messaging.SecureCFNIOAMFEndPoint
secure-cf-rtmp	coldfusion.flash.messaging.SecureCFRTMPEndPoint
my-nio-amf-stream	coldfusion.flash.messaging.CFStreamingNIOAMFEndPoint
secure-nio-amf-stream	coldfusion.flash.messaging.SecureCFStreamingNIOAMFEndPoint

- ColdFusion 9.0.1 では、channel-definition 構文が、services-config.xml（**CF_root**/wwroot/WEB-INF/flex/）に serialize-array-to-arraycollection という名前で導入されました。

  この構文により、ColdFusion 配列を ActionScript Array または ArrayCollection にシリアル化するかどうかの決定を、ユーザーがより柔軟に管理できます。

  シリアル化するには、XML の次のコードの値を true に設定します。

  <serialize-array-to-arraycollection>false</serialize-array-to-arraycollection>

  注意：この構文では、ColdFusion 配列を LCDS Flex クライアントに送信する場合は考慮されていません。この場合は、常に ColdFusion 配列が ActionScript ArrayCollection に変換されます。

- 現在は、"services-config.xml" から参照する必要があるその他のファイルがすべて、"services-config.xml" に「インクルード」されています。

ColdFusion 8 では、次のような "services-config.xml" ファイルが使用されていました。

```
<channel-definition id="cf-polling-amf" class="mx.messaging.channels.AMFChannel">
<endpoint url="http://{server.name}:{server.port}{context.root}/
    flex2gateway/cfamfpolling" class="flex.messaging.endpoints.AMFEndpoint"/>
            <properties>
            <polling-enabled>true</polling-enabled>
                <serialization>
                    <instantiate-types>false</instantiate-types>
                </serialization>
            </properties>
</channel-definition>

<channel-definition id="cf-rtmp" class="mx.messaging.channels.RTMPChannel">
<endpoint url="rtmp://{server.name}:2048" class="flex.messaging.endpoints.RTMPEndpoint"/>
<properties>
    <idle-timeout-minutes>20</idle-timeout-minutes>
        <serialization>
            <!-- This must be turned off for any CF channel -->
                <instantiate-types>false</instantiate-types>
            </serialization>
</properties>
</channel-definition>
```

新しい "services-config.xml" ファイルは次のような内容になっています。

```
<channel-definition id="cf-polling-amf "class="mx.messaging.channels.AMFChannel">
<endpoint uri="http://{server.name}:{server.port}{context.root}
/flex2gateway/cfamfpolling" class="coldfusion.flash.messaging.CFAMFEndPoint" />
<properties>
    <polling-enabled>true</polling-enabled>
    <polli ng-interval-seconds>8</polling-interval-seconds>
    <serialization>
        <enable-small-messages>false</enable-small-messages>
    </serialization>
    <coldfusion>
        <access>
            <use-mappings>true</use-mappings>
            <method-access-level>remote</method-access-level>
        </access>
        <use-accessors>true</use-accessors>
        <use-structs>false</use-structs>
        <property-case>
            <force-cfc-lowercase>false</force-cfc-lowercase>
            <force-query-lowercase>false</force-query-lowercase>
            <force-struct-lowercase>false</force-struct-lowercase>
        </property-case>
    </coldfusion>
</properties>
</channel-definition>
<channel-definition id="cf-rtmp" class="mx.messaging.channels.RTMPChannel">
<endpoint uri="rtmp://{server.name}:2048" class="coldfusion.flash.messaging.CFRTMPEndPoint"/>
    <properties>
            <idle-timeout-minutes>20</idle-timeout-minutes>
    <serialization>
        <enable-small-messages>false</enable-small-messages>
    </serialization>
            <coldfusion>
                <access>
                    <use-mappings>true</use-mappings>
                    <method-access-level>remote</method-access-level>
                </access>
                <use-accessors>true</use-accessors>
                <use-structs>false</use-structs>
                <property-case>
                    <force-cfc-lowercase>false</force-cfc-lowercase>
                    <force-query-lowercase>false</force-query-lowercase>
                    <force-struct-lowercase>false</force-struct-lowercase>
                </property-case>
            </coldfusion>
    </properties>
</channel-definition>
```

ColdFusion 固有のチャネル定義の指定

LiveCycle Data Services ES では、Flex メッセージングシステムの一部であるメッセージチャネルを経由して、宛先との間でメッセージが転送されます。宛先を設定するときは、使用するメッセージングチャネルを参照します。ColdFusion のバックエンドアプリケーションに接続するためには、"services-config.xml" ファイルの channels セクションに cf-polling-amf チャネルと cf-rtmp チャネルの定義が含まれていることを確認する必要があります。LiveCycle Data Services ES を ColdFusion で実行している場合、デフォルトでは、"wwwroot¥WEB-INF¥flex" ディレクトリに "services-config.xml" ファイルがあり、このファイルにチャネル定義が含まれています。LiveCycle Data Services ES をリモートで実行していて、LiveCycle Data Services ES をデフォルトの場所にインストールした場合は、"¥WEB-INF¥flex" ディレクトリに "services-config.xml" ファイルがあります。

新しいチャネル定義の内容は、次のとおりです。

```
<channel-definition id="cf-polling-amf" class="mx.messaging.channels.AMFChannel ">
<endpoint uri="http://{server.name}:{server.port}{context.root}
/flex2gateway/cfamfpolling"class="coldfusion.flash.messaging.CFAMFEndPoint" />
<properties>
    <polling-enabled> true </polling-enabled>
    <polling-interval-seconds>8</polling-interval-seconds>
    <serialization>
        <enable-small-messages>false</enable-small-messages>
    </serialization>
    <coldfusion>
        <access>
            <use-mappings>true</use-mappings>
            <method-access-level>remote</method-access-level>
        </access>
        <use-accessors>true</use-accessors>
        <use-structs>false</use-structs>
        <property-case>
            <force-cfc-lowercase>false</force-cfc-lowercase>
            <force-query-lowercase>false</force-query-lowercase>
            <force-struct-lowercase>false</force-struct-lowercase>
        </property-case>
    </coldfusion>
</properties>
</channel-definition>
```

要素	説明
access	呼び出す CFC の解決ルールおよびアクセスレベルを定義します。
use-accessors	値オブジェクト CFC が getter と setter を持つかどうかを指定します。値オブジェクト CFC に getter および setter が存在する場合は、use-accessors の値を true に設定します。ただし、値オブジェクト CFC に getter および setter が存在しない場合に use-accessors を true に設定すると、値オブジェクト CFC のプロパティの値は this スコープで設定されます。CFC に getter および setter が含まれていない場合は、この要素を false に設定すると、ColdFusion でこれらのメソッドを確認する時間が削減されるので、パフォーマンスが向上します。デフォルト値は true です。
use-structs	ActionScript を CFC に変換するかどうかを指定します。ActionScript から CFC への変換を行わないようにする場合は、use-structs の値を true に設定します。値が false の場合でも、アセンブラから Flex に構造体を返すことは可能です。デフォルト値は false です。
force-cfc-lowercase force-query-lowercase force-struct-lowercase	ActionScript への変換時に、プロパティ名、クエリー列名、構造体キーを小文字にするかどうかを指定します。クエリー列名は、対応する ActionScript 変数と大文字小文字が完全に一致している必要があります。デフォルト値は false です。
use-mappings	source 属性に ColdFusion マッピングを基準 (開始) とした相対位置を設定できるかどうかを指定するブール値デフォルト値は true です。
method-access-level	ColdFusion がリクエストに応答するために CFC で必要な access 属性を指定します。使用できる値は次のとおりです。 • remote Flex は、リモートアクセスを指定した関数にのみアクセスできます (デフォルト)。 • public Flex は、リモートアクセスおよびパブリックアクセスの両方を指定した関数にアクセスできます。

ColdFusion Data Service Adapter の指定

Flex には、各種のバックエンドアプリケーションに接続するためのアダプタが用意されています。ColdFusion Data Service Adapter を使用するには、データ管理設定ファイルでそのアダプタを指定します。これを行うには、Flex アプリケーションを実行するサーバー上の "WEB-INF/flex" フォルダにある "data-management-config.xml" ファイルの adapters セクションに、次の adapter-definition をコピーします。LiveCycle Data Services ES を ColdFusion で実行している場合、デフォルトでは、"data-management-config.xml" ファイルにアダプタ定義が含まれます。

アダプタ定義の内容は、次のとおりです。

```
<adapter-definition id="coldfusion-dao" class="coldfusion.flex.CFDataServicesAdapter"/>
```

宛先の指定

宛先とは、アプリケーションから呼び出すサーバーサイドのサービスまたはオブジェクトです。Data Management の宛先は "data-management-config.xml" ファイルで設定します。

宛先には次の要素が含まれます。

要素	説明
destination id	ID は宛先ごとに一意である必要があります。
adapter-ref	使用するアダプタの名前。ColdFusion 固有の宛先には、ColdFusion adapter-ref 要素を使用します。
channels-ref	ColdFusion で設定されたチャネルのうち、instantiate-types フラグが false に設定されているチャネルを使用します。
component	ColdFusion サーバー上の名前またはパス。
scope	スコープ。application、session、request のいずれかを指定できます。値 application では、インスタンスが 1 つのみ作成されます。request では、呼び出すごとに新しい CFC が作成されます。ColdFusion では session はサポートされません。このスコープを、ColdFusion の変数スコープと混同しないでください。両者は無関係です。
hostname	ColdFusion ホストのホスト名または IP アドレス。LiveCycle Data Services を ColdFusion の一部として実行している場合は、ホスト名や IP アドレスを指定しません。LiveCycle Data Services ES をリモートで実行している場合は、ホスト名または IP アドレスを指定します。
identity	ColdFusion Administrator で設定した、ColdFusion Data Management サーバーの ID。 この指定が必要なのは、RMI を使用して ColdFusion サーバーにリモートでアクセスしており、マシン上に ColdFusion のインスタンスが複数存在している場合のみです。
remote-username remote-password	すべてのクライアントのアセンブラ CFC に渡す資格情報。一般には、クライアントサイドで ActionScript の setRemoteCredentials() API を使用する方法をお勧めします。
identity property	データベースのプライマリキーであるプロパティまたはプロパティのリスト。
query-row-type	オプション。アセンブラの fill メソッドでクエリーが返される場合は、ArrayCollection で返されるクエリーの各行に対して、ActionScript 型を定義します。
fill-method	create または update 操作後に fill 操作の結果を更新するかどうかを指定します。
use-fill-contains	オプション。アセンブラが fill-contains メソッドを持っているかどうかを指定します。このメソッドは、fill 操作をリフレッシュするかどうかを決めるために使用されます。指定したメソッドが true を返した場合、create または update 操作の後に、fill 操作が再実行されます。auto-refresh を true に設定する場合にのみ、use-fill-contains を true に設定します。デフォルト値は false です。
auto-refresh	オプション。create または update 操作の後に fill 操作をリフレッシュするかどうかを指定します。デフォルト値は true です。
ordered	オプション。この fill 操作を実行したコレクションで順序が重要であるかどうかを指定します。順序が重要でない場合は、パフォーマンスを最適化できます。デフォルト値は true です。

サンプルの定義を次に示します。

```
<destination id="cfcontact">
<!-- Use the ColdFusion adapter for any CF specific destinations-->
    <adapter ref="coldfusion-dao" />
<channels>
<channel ref="cf-polling-amf" />
</channels>

    <properties>
        <!--The component name or path on the CF server-->
            <component>samples.contact.ContactAssembler</component>
        <!--Either "application" or "request"-->
            <scope>request</scope>
        <!-- The hostname or IP address of the CF host. If Data Services is installed as
            part of CF, you omit this. If Data Services runs outside of CF, you must
            define this. <hostname>localhost</hostname>-->

        <!--This is the ID of the ColdFusion Data Management service as configured in
        the ColdFusion Administrator. Only needed if you have more than one instance of
        CF on a machine and Data Services is not installed as part of CF.
        <identity>default</identity> -->
        <!--Credentials to pass to the assembler CFC for all clients. Generally better
        to use setRemoteCredentials() API on client <remote-username></remote-username>
        <remote-password></remote-password>-->
    <metadata>
        <identity property="contactId" />
        <!--Optional, If the Assembler fill routine returns a query,you must define an
            Actionscript type for the rows.-->
        <query-row-type>samples.contact.Contact</query-row-type>
    </metadata>

    <network>
        <!--  Add network elements here-->
    </network>

    <server>
    <!-- The method declarations are ignored for CFC Assemblers, with the exception of
    the fill-method settings. No parameters are defined here, unlike Java. Any arguments
    provided via the AS call are passed along to the CFC, just use optional arguments
    when defining the CFC.-->
    <fill-method>
    <!--Does the assembler have a "fill-contains" method? This method is used to
    determine whether to refresh the fill. If the specified method returns true the fill
    is re-executed after a create or update. Auto-refresh determines if the fill is
    always refreshed if not specified. May only be used when auto-refresh is true.
    Optional. Default is false.-->
    <use-fill-contains>false</use-fill-contains>
    <!-- Determines whether to refresh the fill on updates or creates. Optional. Default
        value is true.-->
    <auto-refresh>true</auto-refresh>
    <!--Determines whether order is important for this filled collection. Allows for
    performance optimization when order is not important. Optional. Default value is
    true.-->

    <ordered>true</ordered>

    </fill-method>
    </server>
    </properties>
</destination>
```

ColdFusion 固有のデバッグ出力の有効化

ColdFusion 固有のデバッグ出力を Flex コンソールで有効にするには、"services-config.xml" ファイルの <logging> セクションの <filters> タグに、次の <pattern> タグを追加します。

```
<pattern>DataService.coldfusion</pattern>
```

詳細については、Flex のマニュアルセットに含まれている『Flex アプリケーション開発ガイド』の、データサービスの設定に関するセクションを参照してください。

注意：ColdFusion Administrator を使用して、LiveCycle Data Management サポートの有効、無効を切り替えることができます。ColdFusion を複数実行している場合は、一意の ID を使用して、LiveCycle Data Management サポートを有効にする ColdFusion の各インスタンスを指定します。この ID は "data-management-config.xml" ファイルの identity 要素で指定します。

ColdFusion CFC の記述

ColdFusion CFC を作成する場合は、次のいずれかを行います。

- アセンブラ CFC と値オブジェクト CFC を作成します。
- アセンブラ CFC、DAO (Data Access Object) CFC、および値オブジェクト CFC を作成します。

データベース操作機能をアセンブラ CFC のメソッドで直接実装して、値オブジェクト CFC を作成することができます。値オブジェクト CFC は、プロパティ定義と、関連する get および set メソッドを持つ CFC です。

低レベルのデータベース機能を高レベルの Flex アセンブラ操作から分離する場合は、DAO (Data Access Object) CFC を作成して、低レベルのデータベース機能を含めることができます。この方法は Bean/DAO と呼ばれます。この方法を使用する場合、fill、get、sync、count の各メソッドはアセンブラ CFC に記述する必要があります。アセンブラ CFC のメソッドで、DAO CFC のメソッドを呼び出して、レコードの取得などの低レベルのデータベース機能を実行させます。DAO CFC によって、値オブジェクト (値が格納された CFC) を作成します。値オブジェクトは基本的に結果セットの 1 行に相当します。

LiveCycle Data Management Service で認識されるメソッドは、fill、get、sync、count の 4 つです。fill メソッドは、データベースからレコードを取得して配列に格納します。get メソッドは、特定のレコードを取得します。sync メソッドは、変更オブジェクトの配列である変更リストを使用して、同期の競合を記録します。count メソッドは、結果セットに含まれているレコードの数を返します。このようなデータベース操作を実行するために、Flex アプリケーションはアセンブラ CFC の適切な fill、get、sync、count メソッドを呼び出します。また fillContains メソッドを使用して、fill 操作の結果を更新するかどうかを確認することもできます。詳細については、672 ページの「fill 操作の管理」を参照してください。

fill メソッドの作成

fill メソッドは、データベースからレコードを取得して配列に格納します。Bean/DAO の方法を使用する場合は、低レベルの read メソッドを DAO CFC に作成します。

fill メソッドは、read 操作の結果を返します。fill メソッドでは、read の結果を格納するための配列を作成してから、read 操作の結果を返します。fill メソッドの基本構造は次のようになります。

```
<cffunction name="fill" output="no" returntype="samples.contact.Contact[]" access="remote">
    <cfreturn variables.dao.read()>
</cffunction>
```

このメソッドでは、値オブジェクト CFC、クエリー、または CFML 構造体の配列を返すことができます。値オブジェクト CFC よりもクエリーを使用したほうが、パフォーマンスが向上する場合があります。ただし、クエリーを使用する場合、ネストした結果セットは ColdFusion で処理できません。たとえば、fill メソッドから返す CFC プロパティのいずれかに、別の複合型 (CFC 型など) が設定されている場合、ColdFusion でクエリー内の列をカスタムタイプのオブジェクトに自動変換することはできません。このような場合は、CFC の配列を返した後に、DAO CFC の fill メソッドまたは read メソッドで適切なオブジェクトを作成します。

アセンブラ内で現在 ColdFusion コンポーネントを作成している箇所は、すべて構造体に置き換えることができます。ただし、この場合でも、Flex からは CFC 値オブジェクトが渡されます。たとえば、ActionScript 型でリモートエイリアスが定義されている場合、sync メソッドで返される変更オブジェクトには CFC が含まれます。

値オブジェクト CFC は、get メソッドで作成できます。ただし、構造体の機能を使用して、CFC の代わりに構造体を作成して返すこともできます。構造体は CFC と同じように変換されるからです。また、CFC の配列の代わりに構造体の配列を返すこともできます。たとえば、データに対する処理が必要で、CFC の処理速度では不十分である場合などが考えられます。一般に、構造体のほうが CFC よりも処理が高速です。また、結果オブジェクトのメンバーが複合オブジェクトである場合も、構造体を使用します。この場合は、そのキーの値として別の構造体を作成し、これに __type__ key を指定します。

fill メソッドの returntype に値オブジェクト CFC、クエリー、または配列を指定するには、次のように記述します。

1 値オブジェクト :

```
<cffunction name="fill" output="no"
    returntype="samples.contact.Contact[]" access="remote">
```

2 クエリー :

```
<cffunction name="fill" output="no"
    returntype="query" access="remote">
```

3 構造体の配列 :

```
<cffunction name="fill" output="no"
    returntype="array" access="remote">
```

値オブジェクト、クエリー、構造体の配列のいずれを使用するかに応じて fill 関数の returntype を指定するのに加えて、低レベルの read 関数でも次の指定を行います。

- read 関数の returntype に、値オブジェクト CFC、クエリー、または配列を指定します。たとえば次のように記述します。
  - <cffunction name="read" output="false" access="public" returntype="**samples.contact.Contact[]**">
  - <cffunction name="read" output="false" access="public" returntype="**query**">
  - <cffunction name="read" output="false" access="public" returntype="**array**">
- 値オブジェクトを使用する場合 :
  - 次のように、値オブジェクトを格納する配列を作成します。

    ```
    <cfset var ret = ArrayNew(1)>
    ```

  - クエリーをループし、クエリーの各行に対応する値オブジェクトを作成します。たとえば、次のようにします。

    ```
    <cfloop query="qRead">
        <cfscript>
            obj = createObject("component",
                "samples.contact.Contact").init();
            obj.setcontactId(qRead.contactId);
            obj.setfirstName(qRead.firstName);
            obj.setlastName(qRead.lastName);
            obj.setaddress(qRead.address);
            obj.setcity(qRead.city);
            obj.setstate(qRead.state);
            obj.setzip(qRead.zip);
            obj.setphone(qRead.phone);
            ArrayAppend(ret, obj);
        </cfscript>
    </cfloop>
    ```

- クエリーを使用する場合 :
  - クエリーの各行に対応する ActionScript 型を ColdFusion が正しく判断できるように、宛先の行の型が設定されている必要があります。これには、宛先の metadata セクションにある query-row-type 要素を使用します。
  - fill メソッドで次のように指定します。

```
<cffunction name="fill" output="no" returntype="query"
    access="remote">
    <cfargument name="param" type="string" required="no">
        <cfquery name="myQuery" .>
        </cfquery>
        <!--- Return the result --->
    <cfreturn myQuery>
</cffunction>
```

  - DAO CFC を使用する場合は、read メソッドを編集して、CFC の配列の代わりにクエリーを返すようにします。
  - クエリー列名と ActionScript オブジェクトのプロパティで大文字と小文字が一致するようにします。これには、宛先の property-case 設定を使用します。force-query-lowercase 要素を false に設定して、ColdFusion ですべての列名が小文字に変換されるようにします。
- 構造体の配列を使用する場合 :
  - 次のように、値オブジェクトを格納する配列を作成します。

```
<cfset var ret = ArrayNew(1)>
```

  - クエリーをループし、クエリーの結果を格納した構造体を作成します。たとえば、次のように処理します。

```
<cfloop query="qRead">
    <cfscript>
        stContact = structNew();
        stContact["__type__"] = "samples.contact.Contact";
        stContact["contactId"] = qRead.contactId;
        stContact["firstName"] = qRead.firstName;
        stContact["lastName"] = qRead.lastName;
        stContact["address"] = qRead.address;
        stContact["city"] = qRead.city;
        stContact["state"] = qRead.state;
        stContact["zip"] = qRead.zip;
        stContact["phone"] = qRead.phone;
        ArrayAppend(ret, duplicate(stContact));
    </cfscript>
</cfloop>
```

  - __type__ という構造体要素を使用して、型が値オブジェクト CFC であることを指定します。たとえば、次のようにします。

```
stContact["__type_"] = "samples.contact.Contact";
```

  - contact["firstName"] のように連想配列のシンタックスを使用して、ActionScript プロパティと大文字小文字が一致するようにします。他のシンタックスを使用すると (たとえば contact.firstName="Joan" のようにすると)、ColdFusion によってキー名が大文字に変換されます。

fill 操作の管理

項目を作成または更新した後に fill の結果をリフレッシュするかどうかを示すために、アセンブラに fillContains メソッドを記述し、"data-management-config.xml" ファイルの fill-method セクションで use-fill-contains と auto-refresh の両方を true に設定します。fill-method セクションの例を次に示します。

```
<fill-method>
    <use-fill-contains>true</use-fill-contains>
    <auto-refresh>true</auto-refresh>
    <ordered>false</ordered>
</fill-method>
```

この例では、fill の結果がソートされていないので、ordered を false に設定しています。fill 結果がソートされている場合は、ordered を true に設定します。ソート済みの fill 結果で項目を変更した場合は、fill 結果全体をリフレッシュします。

fillContains メソッドは、fill 結果の項目が変更された後に fill 操作の再実行が必要であるかどうかを Flex アプリケーションに示します。fillCcontains メソッドは、変更に対する fill 操作の必要性を示す値を返します。fillContains メソッドが true を返した場合、create または update 操作の後に fill 操作が実行されます。

次の例に、fillContains メソッドのシグネチャを示します。

```
<cffunction name="fillContains" output="no" returnType="boolean" access="remote">
     <cfargument name="fillArgs" type="array" required="yes">
     <cfargument name="item" type="[CFC type object]" required="yes">
     <cfargument name="isCreate" type="boolean" required="yes">
```

fillContains メソッドには、次の引数があります。

- fillArgs は、fill メソッドに渡す引数のリストです。
- item は、結果セットにあるかどうか確認する必要のあるレコードです。
- isCreate は、レコードが新規かどうかを示します。

fillContains メソッドの例を次に示します。ここでは、関数に渡された Contact 項目に fill の引数 (氏名の一部) が含まれているかどうか判断します。

```
<cffunction name="fillContains" output="no" returnType="boolean"access="remote">
    <cfargument name="fillArgs" type="array" required="yes">
    <cfargument name="item" type="samples.contact.Contact" required="yes">
    <cfargument name="isCreate" type="boolean" required="yes">

<cfif ArrayLen(fillArgs) EQ 0>
<!--- This is the everything fill. --->
<cfreturn true>
<cfelseif ArrayLen(fillArgs) EQ 1>
<!--- This is a search fill. --->
    <cfset search = fillArgs[1]>
    <cfset first = item.getFirstName()>
    <cfset last = item.getLastName()>
    <!--- If the first or last name contains the search string, --->
    <cfif (FindNoCase(search, first) NEQ 0) OR (FindNoCase(search, last)
        NEQ 0)>
<!--- this record is in the fill. --->
<cfreturn true>
<cfelse>
<!--- this record is NOT in the fill. --->
<cfreturn false>
</cfif>
</cfif>

<!--- By default, do the fill.--->
<cfreturn true>
</cffunction>
```

LiveCycle Data Services ES をローカルで実行している場合は、fill 操作がリフレッシュによってトリガされたのか、クライアントによってトリガされたのかを判断できます。それには、次のように、ColdFusion アプリケーションで DataServiceTransaction.getCurrentDataServiceTransaction().isRefill() を呼び出します。

```
<cfscript>
dst = CreateObject("java", "flex.data.DataServiceTransaction");
t = dst.getCurrentDataServiceTransaction();
isRefill = t.isRefill();
</cfscript>
```

これは、RMI 経由の場合 (ColdFusion と Flex が同じ Web アプリケーション内に存在しない場合) は機能しません。

get メソッドの作成

get メソッドは、特定のレコードを取得します。get メソッドでは、低レベルの read メソッドを呼び出します。Bean/DAO の方法を使用する場合は、670 ページの「ColdFusion CFC の記述」の説明にあるように、低レベルの read メソッドを DAO CFC に作成します。

get メソッドの基本構造を次の例に示します。

```
<cffunction name="get" output="no" returnType="samples.contact.Contact" access="remote">
    <cfargument name="uid" type="struct" required="yes">
    <cfset key = uid.contactId>
    <cfset ret=variables.dao.read(id=key)>
    <cfreturn ret[1]>
</cffunction>
```

get メソッドの returntype には、次のいずれかを指定できます。

- 値オブジェクト CFC
- Any
- 配列

sync メソッドの作成

sync メソッドは、変更オブジェクトの配列である変更リストを使用して、同期の競合を記録します。sync メソッドでは、変更の配列を受け取り、この配列をループして変更を適用した後、変更オブジェクトを返します。たとえば、次のようにします。

```
<cffunction name="sync" output="no" returnType="array" access="remote">
    <cfargument name="changes" type="array" required="yes">

    <!-- Create the array for the returned changes. -->
    <cfset var newchanges=ArrayNew(1)>

    <!-- Loop over the changes and apply them. --->
    <cfloop from="1" to="#ArrayLen(changes)#" index="i" >
        <cfset co = changes[i]>
        <cfif co.isCreate()>
            <cfset x = doCreate(co)>
        <cfelseif co.isUpdate()>
            <cfset x = doUpdate(co)>
        <cfelseif co.isDelete()>
            <cfset x = doDelete(co)>
        </cfif>
        <cfset ArrayAppend(newchanges, x)>
    </cfloop>

    <!-- Return the change objects, which indicate success or failure. --->
    <cfreturn newchanges>
</cffunction>
```

count メソッドの作成

count メソッドは、結果セットに含まれているレコードの数を返します。Bean/DAO の方法を使用する場合は、670 ページの「ColdFusion CFC の記述」の説明にあるように、低レベルの count メソッドを DAO CFC に作成します。

count メソッドの基本構造は次のようになります。エラー処理は含まれません。

```
<cffunction name="count" output="no" returntype="Numeric" access="remote">
    <cfargument name="param" type="string" required="no">
    <cfreturn variables.dao.count()>
</cffunction>
```

この count メソッドでは、DAO CFC の別の count メソッドを呼び出しています。DAO CFC の count メソッドの基本構造は、次のようになります。エラー処理は含まれません。

```
<cffunction name="count" output="false" access="public" returntype="Numeric">
    <cfargument name="id" required="false">
    <cfargument name="param" required="false">
    <cfset var qRead="">

    <cfquery name="qRead" datasource="FDSCFCONTACT">
        select COUNT(*) as totalRecords
        from Contact
    </cfquery>

    <cfreturn qRead.totalRecords>
</cffunction>
```

データ変更時の Flex アプリケーションへの通知

ColdFusion に用意されている LiveCycle Data Services ES のイベントゲートウェイタイプを使用すれば、宛先で管理しているデータが変更されたときに、ColdFusion アプリケーションから Flex に通知することができます。LiveCycle Data Services ES のイベントゲートウェイを設定し、そのイベントゲートウェイを使用するアプリケーションを作成します。詳細については、1304 ページの「Data Management イベントゲートウェイの使用」を参照してください。

認証

LiveCycle Data Services ES アセンブラを使用する場合、ユーザーを認証するには、DataService オブジェクトで Flex の setRemoteCredentials() メソッドを使用します。FlexSession オブジェクトにある資格情報が ColdFusion アプリケーションに渡されるので、cflogin タグを使用して認証を実行できます。別の方法として、Flex の宛先で資格情報を設定することもできますが、この方法はお勧めできません。

資格情報は次のいずれかの方法で設定できます。

- ActionScript による資格情報の指定
- Flex の宛先による資格情報の指定

ActionScript による資格情報の指定

ActionScript で資格情報を指定するには、次の例のように setRemoteCredentials() メソッドを使用します。

```
ds = new DataService("mydest");
ds.setRemoteCredentials("wilsont", "password");
```

Flex の宛先による資格情報の指定

Flex の宛先で資格情報を指定するには、"data-management-config.xml" ファイルを編集します。このファイルは、Flex アプリケーションを実行するサーバー上の "WEB-INF/flex" フォルダにあります。properties 要素に remote-username 要素と remote-password 要素を次のように追加します。

```
<destination id="cfcontact">
    <adapter ref="coldfusion-dao" />
        <channels>
            <channel ref="cf-dataservice-rtmp" />
        </channels>
    <properties>
        <source>samples.contact.ContactAssembler</source>
        <scope>application</scope>
        <remote-username>wilsont</remote-username>
        <remote-password>password</remote-password>
...
    /properties>
</destination>
```

SSL の有効化

SSL (Secure Sockets Layer) を有効にすることで、ColdFusion と Flex との間の通信を暗号化できます。SSL の有効化は、LiveCycle Data Services ES をリモートで実行している場合にのみ意味があります。SSL を使用するには、キーストアファイルを作成します。キーストアは自己署名証明書です。認証機関が署名した証明書は不要ですが、そのような証明書を使用する場合は、次の手順で示す Flex の設定は不要です。キーストア内の情報は暗号化され、指定したパスワードを使用しないとアクセスできません。キーストアを作成するには、Java keytool ユーティリティを使用します。このユーティリティは Java Runtime Environment (JRE) に付属しています。

SSL を有効にするには、次の手順を行います。

1 キーストアの作成

2 Flex の設定

3 ColdFusion Administrator での SSL の有効化

キーストアの作成

keytool ユーティリティを使用して SSL サーバー (ColdFusion) キーストアファイルを作成します。コマンドの例を次に示します。

```
keytool -genkey -v -alias FlexAssembler -dname "cn=FlexAssembler" -keystore cf.keystore -keypass mypassword
-storepass mypassword
```

次の表で、keytool ユーティリティのパラメータを説明します。

パラメータ	説明
-alias	キーストア項目の名前。常にその名前で参照する限りは、任意の名前が使用できます。
-dname	識別名。サーバーの一般名 (cn) を含みます。
-keystore	キーストアファイルの場所。
-keypass	秘密キー用のパスワード。
-storepass	キーストア用のパスワード。暗号化された storepass が ColdFusion 設定ファイルに記録されます。
-rfc	printable encoding 形式で証明書を作成します。
-file	キーストアファイルの名前。
-v	詳細な証明書情報を出力します。

次に、JVM が信頼する証明書を判断するときに使用するファイルの中に、作成した証明書を格納します。証明書を格納するファイル (通常は cacerts という名前) は JRE の "lib/security" フォルダにあります。

Flex の設定

1 keytool ユーティリティを使用してキーストアを証明書に書き出します。コマンドの例を次に示します。

```
keytool -export -v -alias FlexAssembler -keystore cf.keystore -rfc -file cf.cer
```

2 keytool ユーティリティを使用して、サーバーの JRE cacerts ファイルに証明書を読み込みます。コマンドの例を次に示します。

```
keytool -import -v -alias FlexAssembler -file cf.cer -keystore
C:\fds2\UninstallerData\jre\lib\security\cacerts
```

先の例では、デフォルトの設定でインストールされ JRun と統合された LiveCycle Data Services ES のキーストアの場所を指定しています。別のサーバーを使用している場合は、使用している JRE の cacerts ファイルの場所を指定します。たとえば、JBoss を使用している場合は、キーストアの場所を $JAVA_HOME/jre/lib/security/cacerts のように指定します。

ColdFusion Administrator での SSL の有効化

1 ColdFusion Administrator で、[データとサービス]-[Flex 統合] を選択し、[キーストアの絶対パス] テキストボックスでキーストアファイルを指定します。

2 [キーストアパスワード] テキストボックスでキーストアのパスワードを指定します。

3 [SSL による RMI を Data Management 用に有効化] オプションを選択し、[変更の送信] をクリックします。

無効なキーストアファイルやパスワードを指定した場合、SSL は有効にならず、Flex Data Management サポートは無効になります。

データ変換

次の表に、ColdFusion のデータ型と、それに対応する Adobe Flash または ActionScript のデータ型を示します。

ColdFusion のデータ型	Flash のデータ型
文字列	文字列
配列	[] = 配列
構造体	{} = 型指定なしのオブジェクト
クエリー	ArrayCollection
CFC	クラス = 型指定ありのオブジェクト (対応する ActionScript クラスが存在する場合。それ以外の場合は、ActionScript の型指定なしの汎用オブジェクト (map) になります)
CFC の日付	ActionScript の日付
CFC の文字列	ActionScript の文字列
CFC の数値	ActionScript の数値
ColdFusion XML オブジェクト	ActionScript XML オブジェクト

サーバーサイド ActionScript の使用

サーバー設定の Adobe ColdFusion では、Flash Remoting サービスが提供されています。Adobe Flash の開発者は、このモジュールを使用してサーバーサイド ActionScript を作成できます。これらの ActionScript ファイルでは、CF.query と CF.http という 2 つの新しい ActionScript 関数を使用して、ColdFusion クエリーと HTTP の機能に直接アクセスできます。

サーバーサイド ActionScript について

ColdFusion には、Flash と ColdFusion の対話を仲介する、Flash Remoting サービスと呼ばれるモジュールが用意されています。Flash Remoting では多様なオブジェクトタイプがサポートされており、ColdFusion サーバー上の ActionScript ファイルを参照できます。大量のデータ処理が必要な操作をサーバー上に分離することで、サーバーからクライアントにデータを転送するネットワークトランザクションを抑えることができます。

Flash 開発者は、サーバーサイドの ActionScript ファイルを作成して ColdFusion のリソースにアクセスできるので、CFML (ColdFusion Markup Language) を学ぶ必要はありません。この機能を使用すれば、アプリケーションの Flash プレゼンテーション要素を、ビジネスロジックから論理的に分離することができます。ActionScript ファイルをサーバー上に作成して、この処理をクライアントアプリケーションから分離することができます。

サーバーサイド ActionScript を使用することで、単純なインターフェイスでクエリーを構築できます。これらのクエリーは、同様に単純なインターフェイスで、クライアントサイド ActionScript から実行することができます。

クライアントサイド ActionScript の必要条件

クライアントサイドで必要な処理は、Flash Remoting サービスへの接続の確立と、使用するサーバーサイド ActionScript の参照のみです。

次にコード例を示します (コメントを参照してください)。

```
// This #include is needed to connect to the Flash Remoting service
#include "NetServices.as"

// This line determines where Flash should look for the Flash Remoting service.
// Ordinarily, you enter the URL to your ColdFusion server.
// Port 8500 is the Flash Remoting service default.
NetServices.setDefaultGatewayUrl("http://mycfserver:8500");

// With the Flash Remoting service URL defined, you can create a connection.
gatewayConnnection = NetServices.createGatewayConnection();

// Reference the server-side ActionScript.
// In this case, the stockquotes script file lives in the web root of the
// ColdFusion server identified previously. If it lived in a subdirectory
// of the web root called "mydir," you would reference it
// as "mydir.stockquotes".
stockService = gatewayConnnection.getService("stockquotes", this);

// This line invokes the getQuotes() method defined in the stockquotes
// server-side ActionScript.
stockService.getQuotes("macr");

// Once the record set is returned, you handle the results.
// This part is up to you.
function getQuotes_Result ( result )
{
    // Do something with results
}
```

注意：サーバーサイド ActionScript の新しい関数である CF.query と CF.http の 2 つは、クライアントサイド ActionScript ではサポートされていません。

サーバーサイドの必要条件

サーバー上で ActionScript を実行すれば、ActionScript の知識を活用しながら、ColdFusion クエリーや HTTP の機能に直接アクセスできます。ActionScript 関数の CF.query と CF.http を使用すれば、ColdFusion HTTP やクエリーを操作できます。

注意：サーバーサイドの ActionScript ファイルには、.asr 拡張子を使用します。

たとえば、前に示したクライアントサイドコードに対応するサーバーサイド ActionScript コードは、次のようになります。

```
// Filename: stockquotes.asr
// Here is the getQuotes method invoked in the client-side ActionScript.
// It accepts a single stock quote symbol argument.
function getQuotes(symbol)
{
    // Query some provider for the specified stock quote and return the
    // results. In this case, the getQuotesFromProvider method is
    // defined elsewhere in this ActionScript code.
    data = getQuotesFromProvider(symbol);
    // Return the data to the client.
    // Note: this example does not include any of the error checking
    // logic you would normally use prior to returning the data.
    return data;
}
```

getQuotes 関数では、stockquotes リクエストを処理し、リクエストの結果を RecordSet オブジェクトとしてクライアントに返します。

ソフトウェアの必要条件

サーバーサイド ActionScript ファイルを使用するには、次のソフトウェアをインストールする必要があります。

- Adobe Flash
- ColdFusion
- Flash Remoting コンポーネント

この製品の詳細については、www.adobe.com/jp/ を参照してください。

サーバーサイド ActionScript ファイルの場所

ActionScript ファイル (*.asr) は、Web サーバーのルートディレクトリの下であれば、どこにでも置くことができます。Web ルートまたは仮想ディレクトリのサブディレクトリを指定するには、パッケージのドット表記法 (完全修飾ディレクトリ名でスラッシュではなくドットを使用する方法) を使用します。たとえば、次の代入コードの場合、"stockquotes.asr" ファイルは "/mydir/stock/" ディレクトリにあります。

```
stockService = gatewayConnnection.getService("mydir.stock.stockquotes", this);
```

cfsuite.asr.stock.stockquotes のような仮想マッピングを指定することもできます。この例では、cfsuite は仮想マッピング、asr.stock はそのマッピングのサブディレクトリです。

利点

サーバーサイド ActionScript を使用すれば、ActionScript 開発者がその知識を活用して SWF ファイルのバックエンドコードを作成できるので、高い対話性をユーザーに提供できます。また、自社のビジネスに必要な ActionScript 関数をサーバーサイドに配置することで、SWF ファイルでライブラリとしてそれらを共有できます。

たとえば、すべての SQL クエリーメソッドを定義したライブラリを、サーバーサイド ActionScript ファイルとして作成することができます。サーバーサイドでクエリーメソッドを定義しておけば、Flash デザイナーは目的のクエリー関数を実行するだけで、SWF ムービーにデータを取り込めるようになります。ColdFusion データソースからデータを取得するたびに SQL を記述したり、クエリーを作成したりする必要はありません。これによって、Flash デザインチーム全体で使用できる、再利用可能なクエリーを作成できます。

ColdFusion クエリーや HTTP を操作する ActionScript コードは、非常に簡潔です。CF.query 関数および CF.http 関数を使用すれば、簡単に SQL クエリーや HTTP 操作を実現できます。

たとえば、クエリーデータを返すサーバーサイド ActionScript 関数の一般的な例は次のようになります。

```
// This function shows a basic CF.query operation using only
// arguments for data source name and for SQL.
function basicQuery()
{
    mydata = CF.query({datasource:"customers",
        sql:"SELECT * FROM myTable"});
    return mydata;
}
```

次のステップ

ActionScript に慣れていれば、次の方法を確認するだけで開発を始めることができます。

- クライアントサイド ActionScript を使用して、Flash Remoting サービスとの接続を確立する方法。680 ページの「Flash Remoting サービスへの接続」を参照してください。
- サーバーサイドの ActionScript 関数とメソッドを参照する方法。681 ページの「サーバーサイド ActionScript 関数の使用」を参照してください。
- サーバーサイドの CF.query と CF.http 関数をコーディングする方法。683 ページの「CF.query 関数の使用」および 688 ページの「CF.http 関数の使用」を参照してください。これらの関数については、『CFML リファレンス』のリファレンスページも参照してください。

Flash Remoting の使用方法の詳細については、595 ページの「Flash Remoting サービスの使用」および『Flash Remoting の使用』を参照してください。

Flash Remoting サービスへの接続

サーバーサイド ActionScript ファイルに定義されている関数を使用するには、Adobe SWF ムービーをサーバーサイド Flash Remoting サービスに接続します。

Flash Remoting サービスへの接続の作成

1 サーバーサイド ActionScript 関数を使用する SWF ムービーの最初のフレームで、必要な ActionScript クラスをインクルードします。

   a 次のコマンドを使用して、NetServices クラスをインクルードします。

   ```
   #include "NetServices.as"
   ```

   b (オプション) 次のコマンドを使用して、NetDebug クラスをインクルードします。

   ```
   #include "NetDebug.as"
   ```

   NetDebug および RecordSet クラスの詳細については、『Flash Remoting の使用』を参照してください。

2 サーバーサイド ActionScript 関数は、Flash Remoting サービスを介して呼び出すので、Flash Remoting サービスの URL を NetServices.setDefaultGatewayUrl 関数の引数に渡します。次に例を示します。

   ```
   NetServices.setDefaultGatewayURL("http://localhost:8500/flashservices")
   ```

   サーバーのホスト名を指定します。Flash Remoting サービスのデフォルトのポート番号は 8500 です。

3 NetServices.createGatewayConnection 関数を使用して、ゲートウェイ接続を確立します。たとえば、次のようになります。

   ```
   gatewayConnection = NetServices.createGatewayConnection();
   ```

サーバーサイド ActionScript 関数の使用

Flash Remoting サービスに接続したら、サーバーサイド ActionScript ファイルに定義されている関数を呼び出して、結果を受け取ることができます。

関数の呼び出し

1 getService 関数を使用して、サーバーサイド ActionScript ファイルのインスタンスを作成します。この関数によって、サーバーサイド ActionScript ファイルが、クライアントサイドで使用するオブジェクトとしてインスタンス化されます。次に例を示します。

```
albumService = gatewayConnection.getService("recordsettest", this)
```

recordsettest は、サーバーサイド ActionScript ファイルの名前 (ファイル名拡張子の .asr を除く) です。

2 サーバーサイド ActionScript オブジェクトに定義されている関数を呼び出します。ドット表記法を使用して、オブジェクト名の後に関数名を指定します。次に例を示します。

```
albumService.getAlbum("The Color And The Shape", "1999");
```

albumService はサーバーサイド ActionScript ファイルのインスタンスです。getAlbum は関数であり、"The Color and The Shape" と "1999" の 2 つの引数を渡しています。

注意：引数は、関数宣言で定義されている順序で指定する必要があります。

3 ActionScript で関数の結果を処理します。681 ページの「ActionScript での関数の結果の使用」を参照してください。

ActionScript での関数の結果の使用

サーバーサイド ActionScript から返された結果を使用するには、対応する結果関数を作成します。結果関数では、特殊なネーミング規則を使用することで、サーバーサイド ActionScript を呼び出す関数に関連付けます。たとえば、basicCustomerQuery というクライアントサイド ActionScript 関数を定義した場合、basicCustomerQuery_Result という名前の結果関数も作成する必要があります。

CF.http を使用する場合と CF.query を使用する場合では、サーバーサイド ActionScript 関数から返される結果が多少異なります。

- CF.query 関数はレコードセットを返します。このレコードセットは、RecordSet ActionScript クラスオブジェクトのメソッドで操作できます。「CF.query 関数で返された結果の使用」を参照してください。
- CF.http 関数は、サーバーサイド ActionScript で参照しているプロパティを通じて、単純なテキスト文字列を返します。「CF.http 関数で返された結果の使用」を参照してください。

CF.query 関数で返された結果の使用

CF.query のレコードセットで返されたデータにアクセスするには、RecordSet ActionScript オブジェクトの関数を使用します。たとえば、レコードセット内のレコード数や列名を調べることができます。また、RecordSet 関数でもレコードセットからクエリーデータを取得できます。それには、次の例のように、レコードセットの特定の行番号を参照して、getItemAt RecordSet 関数を使用します。

```
// This function populates a Flash text box with data in the first row
// of the record set under the "email" column name.
function selectData_Result ( result )
{
    stringOutput.text = result.getItemAt(0)["email"];
    _root.employeesView.setDataProvider(result);
}
```

この例では、getItemAt 関数の角括弧 ([]) 内で列名を参照しています。ActionScript では、インデックスは 0 から始まるので、getItemAt(0) は最初の行を返します。

詳細については、683 ページの「CF.query 関数の使用」を参照してください。

CF.http 関数で返された結果の使用

CF.http サーバーサイド ActionScript 関数では、データがテキストとして返されます。サーバーサイド関数では、CF.http 関数で返されたオブジェクトのプロパティを参照します。これらのプロパティには、取得したファイルのコンテンツ、HTTP ステータスコード、返されたファイルの MIME タイプなどが含まれています。クライアントサイドでは、CF.http 関数で返されたデータを処理する関数を作成します。これらの関数でテキストデータを処理します。

詳細については、688 ページの「CF.http 関数の使用」を参照してください。

グローバルスコープオブジェクトとリクエストスコープオブジェクト

グローバルスコープオブジェクトとリクエストスコープオブジェクトは、すべてのサーバーサイド ActionScript で暗黙的に使用できます。次の表で、これらのスコープオブジェクトについて説明します。

スコープ名	タイプ	説明
config	グローバル	サーバーサイド ActionScript アダプタの初期化情報を提供します。 クラス : javax.servlet.ServletConfig
application	グローバル	現在の Web アプリケーションのコンテキスト。このコンテキストには、ログファイルへの書き込みに使用できる MIME タイプなどを提供するメソッドが定義されています。Web アプリケーションごとに 1 つのコンテキストがあります。 クラス : javax.servlet.ServletContext
request	リクエスト	クライアントリクエスト情報を含むオブジェクト。このオブジェクトによって、パラメータの名前と値、属性、入力ストリームなどのデータが提供されます。 クラス : HttpServletRequest (javax.servlet.ServletRequest のサブタイプ)
response	リクエスト	クライアントへのレスポンス送信を補助するオブジェクト。レスポンスを送信する HTTP 固有の機能を提供します。クライアントにデータを返信するために OutputStream または PrintWriter を使用しないでください。 クラス : HttpServletResponse (javax.servlet.ServletResponse のサブタイプ)

これらのスコープオブジェクトの詳細については、http://java.sun.com にある javax.servlet クラスに関する資料を参照してください。

CF.query 関数とデータソースについて

CF.query 関数を使用すると、ColdFusion データソースから取得したデータを SWF ムービー要素に挿入できます。CF.query 関数を使用すると、次のことが行えます。

ColdFusion データソースから SWF ムービーへのデータの取り込み

1. ColdFusion データソースをクエリーするサーバーサイド ActionScript ファイルを作成します。
2. ColdFusion サーバー上の ActionScript ファイル (*.asr) を参照する ActionScript コードを、SWF ムービー内に作成します。

サーバーサイド ActionScript では、クエリーを実行して、レコードセットのデータをクライアント (SWF ムービー) に返します。クライアントでは、RecordSet ActionScript オブジェクトのメソッドを使用してレコードセットのデータを操作し、SWF ムービーでデータを表示します。

注意 : クライアントサイド ActionScript ファイルの拡張子は *.as です。サーバーサイド ActionScript ファイルの拡張子は *.asr です。これは ActionScript remote という意味です。

ダイナミックデータの提供

ダイナミックデータを提供するには、ColdFusion でサーバーサイド ActionScript 機能を使用します。これを行うには、ColdFusion データソースをクエリーするサーバーサイド ActionScript ファイルを作成します。ActionScript を使用する前に、次の方法を理解しておく必要があります。

- CF.query ActionScript 関数を使用して、サーバーサイド ActionScript ファイルでデータベースクエリーを作成する方法。683 ページの「CF.query 関数の使用」を参照してください。
- SWF ムービーでサーバーサイド ActionScript ファイルを参照する方法。680 ページの「Flash Remoting サービスへの接続」を参照してください。

CF.query 関数を使用すると、次のようなタスクを行えます。

- ColdFusion データソースに対してユーザーを検証するユーザーログインインターフェイスを作成する。
- フォーム要素とデータグリッドに、ColdFusion データソースのデータを挿入する。
- データベースのデータ (URL やイメージファイルのパスなど) を取り込んでバナーを作成する。

CF.query 関数では、サポートされているすべての ColdFusion データソースからデータを取得できます (683 ページの「ColdFusion データソースについて」を参照してください)。

ColdFusion データソースについて

ColdFusion の開発者にとって、データソースという言葉は、ローカルやネットワーク上にある、さまざまなタイプの構造化データを意味します。Web サイト、LDAP (Lightweight Directory Access Protocol) サーバー、POP メールサーバー、およびさまざまな形式のドキュメントに対してクエリーを実行することができます。サーバーサイド ActionScript では、データソースという言葉は通常、ColdFusion データベースへのエントリポイントを意味します。

データソースの識別と設定には、ColdFusion Administrator を利用できます。ColdFusion データソースを正しくクエリーする ActionScript ファイルを作成するには、ColdFusion でデータソースがどのように識別されるかを理解する必要があります。また、データベース接続に関係するその他の要因 (ユーザー名やパスワードが接続に必要かどうかなど) も把握する必要があります。

ColdFusion でサーバーサイド ActionScript を使用すれば、ColdFusion データソースのレコードセットデータを Flash クライアントに返すことができます。ColdFusion データソース名と、そのデータソースに対する SQL ステートメントを、サーバーサイド ActionScript の CF.query 関数の引数として指定します。

通常、サーバーサイド ActionScript では、ColdFusion データソースと対話して、Flash Remoting サービス経由で Flash クライアントにレコードセットを返します。

ColdFusion データソースの詳細については、『ColdFusion 設定と管理』を参照してください。

CF.query 関数の使用

サーバーサイド ActionScript の CF.query 関数を使用すると、ColdFusion データソースからデータを取得することができます。この関数を使用して、任意の ColdFusion データソースにクエリーを実行できます。

注意 : CF.query 関数は cfquery CFML タグに対応していますが、現在は cfquery 属性のサブセットがサポートされています。

CF.query 関数を使用すると、次のことが行えます。

- クエリーするデータソースの識別
- データソースへの SQL ステートメントの受け渡し
- データベースへの他のオプションパラメータの受け渡し

CF.query 関数のリファレンス情報については、『CFML リファレンス』の CF.query を参照してください。

CF.query 関数のシンタックスについて

CF.query ActionScript 関数では、名前付き引数と位置引数の両方のスタイルがサポートされています。名前付き引数のスタイルのほうが読みやすくなりますが、コードが長くなります。位置引数のスタイルでは、CF.query 引数のサブセットのみがサポートされますが、コードがコンパクトになるので、CF.query 関数をシンプルに記述したい場合に適しています。

CF.query での名前付き引数のシンタックスの使用

CF.query 関数では、次の名前付き引数を取ります。

```
// CF.query named argument syntax
CF.query
    ({
        datasource:"data source name",
        sql:"SQL stmts",
        username:"username",
        password:"password",
        maxrows:number,
        timeout:milliseconds
    })
```

注意：名前付き引数のスタイルでは、関数の引数を中括弧 ({}) で囲みます。

CF.query での位置引数のシンタックスの使用

位置引数では CF.query 引数のサブセットのみがサポートされており、より効率的にコードを作成できます。位置引数スタイルのシンタックスは、次のとおりです。

```
// CF.query positional argument syntax
CF.query(datasource, sql);
CF.query(datasource, sql, maxrows);
CF.query(datasource, sql, username, password);
CF.query(datasource, sql, username, password, maxrows);
```

注意：位置引数を使用する場合は、中括弧を使用しないでください。

CF.query レコードセットについて

CF.query 関数は、RecordSet オブジェクトを返します。これは、RecordSet クラスのインスタンスです。RecordSet クラスには、レコードセットのデータを操作するさまざまな関数が用意されています。

クライアントサイド ActionScript で RecordSet ActionScript クラスのメソッドを使用して、CF.query レコードセットに返されたデータを変更できます。

RecordSet クラスでは、現在次のメソッドが使用可能です。

メソッド	説明
addItem	指定の RecordSet の末尾にレコードを追加します。
addItemAt	指定のインデックスにレコードを挿入します。
addView	RecordSet オブジェクトのステートの変更通知をリクエストします。
filter	元の RecordSet オブジェクト内の特定のレコードのみを含む RecordSet オブジェクトを作成します。
getColumnNames	RecordSet のすべての列名を返します。
getItemAt	RecordSet オブジェクトからレコードを取得します。
getItemID	レコード固有の ID を取得します。
getLength	RecordSet オブジェクトのレコードの総数を返します。

メソッド	説明
getNumberAvailable	サーバーからダウンロードされたレコード数を返します。
isFullyPopulated	RecordSet オブジェクトの編集や操作が可能かどうかを判定します。
isLocal	RecordSet オブジェクトがローカルか、またはサーバーに関連付けられているかを判定します。
removeAll	RecordSet オブジェクトからすべてのレコードを削除します。
removeItemAt	指定のレコードを削除します。
replaceItemAt	レコードのすべてのコンテンツを置き換えます。
setDeliveryMode	サーバーに関連付けられたレコードセットの配信モードを変更します。
setField	レコードの 1 つのフィールドを、新しい値に置き換えます。
sort	すべてのレコードを、指定の比較関数によってソートします。
sortItemsBy	すべてのレコードを、選択したフィールドによってソートします。

これらの関数は、CF.query 関数によって Flash クライアントに返された、すべての RecordSet オブジェクトに対して使用可能です。これらの関数は、次のようにして実行します。

```
objectName.functionName();
```

たとえば、CF.query 関数によって返されたレコードセットのデータを処理する結果関数では、RecordSet の getColumnNames 関数を使用して、レコードセットに返されたデータベース列の名前を参照できます。

```
function selectData_Result ( result )
{
    //result holds the query data; employeesView is a Flash list box
    stringOutput.text = result.getColumnNames();
    _root.employeesView.setDataProvider(result);
}
```

簡単なアプリケーションの構築

ここでは、簡単なサーバーサイド ActionScript アプリケーションの構築手順を説明します。このサンプルアプリケーションは、社員情報のディレクトリで、NetServices オブジェクトを使用して personneldirectory というサーバーサイド ActionScript に接続します。personneldirectory サーバーサイド ActionScript では、ColdFusion データソースからデータを取得し、RecordSet オブジェクトとして SWF ファイルに結果を返します。

注意：サーバーサイド ActionScript アプリケーションによって、アプリケーションのバックエンドサービスが提供されます。

この例では、次のものが必要になります。

- "personneldirectory.asr" というサーバーサイド ActionScript ファイル。このファイルに、ColdFusion データソースとやり取りを行う関数を実装します。
- NetServices オブジェクトを作成するクライアントサイド SWF ムービー。

アプリケーションの作成

1. データベースにクエリーを行い、Flash Remoting サービスを通じてクライアントにデータを返す、サーバーサイド ActionScript を作成します。
2. SWF ムービーインターフェイスを作成します。686 ページの「SWF ムービーインターフェイスの作成」を参照してください。

3 ユーザーデータを Flash Remoting サービスに送信する検索関数を定義します。686 ページの「Flash Remoting サービスへのユーザーデータの送信」を参照してください。

4 Flash Remoting サービスから返された結果を取得する結果関数を定義します。687 ページの「Flash Remoting サービスの結果の取得」を参照してください。

5 SWF ムービーが Flash Remoting サービスとの接続を確立したことを確認します。687 ページの「Flash Remoting サービス接続の確認」を参照してください。

サーバーサイド ActionScript 関数の作成

ここでは、ColdFusion データソースに対して簡単な検索操作を実行する検索関数を作成します。この関数は、firstName と lastName の 2 つの引数を取り、これらの引数に一致するすべてのレコードを返します。

サーバーサイド ActionScript 関数の作成

❖ 次のコードを含むサーバーサイド ActionScript ファイルを作成します。

```
//search takes firstName lastName arguments
function search(firstName, lastName)
{
    searchdata = CF.query({datasource: "bigDSN",
        sql:"SELECT * from personnel WHERE fname = firstName AND lname = lastName"{};

    if (searchdata)
        return searchdata;
    else
        return null;
}
```

SWF ムービーインターフェイスの作成

ここで作成する SWF ムービーインターフェイスは、各種のテキストボックスと送信ボタンを持つ 1 つのフレームで構成されています。

1 Flash オーサリング環境で、Flash ソースファイルを作成し、"pDirectory.fla" として保存します。

2 2 つの入力テキストボックスを作成します。1 つのテキストボックスの変数名を **lastName** とし、もう 1 つのテキストボックスの変数名を **firstName** とします。

3 ダイナミックテキストボックスを作成し、その変数名を **status** にします。

4 リストボックスコンポーネントを挿入し、**dataView** という名前にします。

5 プッシュボタンコンポーネントを挿入します。

6 作業内容を保存します。

Flash Remoting サービスへのユーザーデータの送信

サーバーサイド ActionScript にデータを送信するには、SWF ムービーからサーバーサイド ActionScript にデータを渡す関数を作成します。フレームレベルで適用される search 関数は、firstName および lastName テキストボックスに入力されたデータを収集して、directoryService オブジェクトの関数に引数として渡します。このオブジェクトは、SWF ムービーが Flash Remoting サービスに接続したときに作成されます。詳細については、687 ページの「Flash Remoting サービス接続の確認」を参照してください。

Flash ActionScript の例を次に示します。

```
#include "NetServices.as"
function search()
{
    // The search() method is defined in the server-side AS file
    directoryService.search(firstName.text, lastName.text);
    dataView.setDataProvider(null);
    status.text = "waiting...";
}
```

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
directoryService.search (firstName.text, lastName.text);	firstName および lastName テキストボックスの内容をサーバーサイド ActionScript に渡します。
dataView.setDataProvider (null);	dataView リストボックスコンポーネントをクリアします。
status.text = "waiting...";	サーバーサイド ActionScript からレコードセットを取得するまでの間、ステータステキストボックスにメッセージを表示します。

Flash Remoting サービスの結果の取得

サーバーサイド ActionScript 関数を呼び出す関数を作成したら、サーバーサイド ActionScript から返されるデータを処理する関数も作成します。その関数名は、呼び出しを行う関数名の後に _Result を付加したものにします。

たとえば、クエリーデータをリクエストする basicQuery という関数を作成したら、返されるデータを処理する basicQuery_Result という結果関数を定義します。

次の例の結果関数 search_Result では、dataView.setDataProvider 関数にレコードセットを提供しています。

```
function search_Result(resultset)
{
    dataView.setDataProvider(resultset);
    status.text = (0+resultset.getLength())+" names found.";
}
```

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
function search_Result (resultset)	_Result という接尾辞を付けることで、Flash Remoting サービスが search 関数の結果をこの関数に返すようになります。
dataView.setDataProvider (resultset);	Flash Remoting サービスから返された結果を、dataView リストボックスに割り当てます。
status.text = (0+resultset. getLength())+" names found.";	Flash Remoting サービスから返されたレコードの数を表示します。

Flash Remoting サービス接続の確認

SWF ムービーが Flash Remoting サービスに接続していることを確認するには、if ステートメントを使用します。次に例を示します。

```
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayUrl("http://localhost:8500/flashservices/
        gateway");
    gateway_conn = NetServices.createGatewayConnection();
    directoryService = gateway_conn.getService(personneldirectory, this);
    status.text = "Type into the text boxes, then click 'Search'";
}
```

この例では、inited 変数の値を評価しています。inited が null の場合 (接続されていない場合) は、ムービーが NetServices オブジェクトを使用して Flash Remoting サービスに接続します。Flash Remoting サービスへの接続の詳細については、680 ページの「Flash Remoting サービスへの接続」を参照してください。

CF.http 関数について

CF.http ActionScript 関数を使用すると、HTTP の Get メソッドおよび Post メソッドを利用して、次のようにリモート HTTP サーバーから情報を取得できます。

- Get メソッドでは、URL の中に情報を直接含めてリモートサーバーに送信します。このメソッドは、オブジェクト (Web ページのコンテンツなど) を取得する一方向トランザクションを CF.http で実行する場合によく使用されます。
- Post メソッドでは、フォームまたは CGI プログラムに変数を渡すことができます。また、HTTP Cookie を作成することもできます。

CF.http 関数の一般的な使用例としては、Get メソッド引数を使用して、指定の URL からページを取得する操作があります。CF.http 関数では、Get メソッドがデフォルトです。

次のサーバーサイドの例では、指定の URL からファイルコンテンツを取得します。

```
function basicGet(url)
{
    // Invoke with just the url argument. This is an HTTP GET.
    result = CF.http(url);
    return result.get("Filecontent");
}
```

クライアントサイドの例は次のようになります。

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://mycfserver:8500");
gatewayConnnection = NetServices.createGatewayConnection();
myHttp = gatewayConnnection.getService("httpFuncs", this);

// This is the server-side function invocation
url = "http://anyserver.com";
myHttp.basicGet(url);

// Create the results function
function basicGet_Result()
{
    url = "http://anyserver.com
    ssasFile.basicGet(url)
}
```

CF.http 関数の使用

CF.http 関数は、プロパティ (属性ともいいます) を持つオブジェクトを返します。これらの属性を参照して、返されたファイルのコンテンツや、ヘッダ情報、HTTP ステータスコードなどにアクセスします。次の表に、利用できるプロパティを示します。

プロパティ	説明
Text	指定した URL 位置に、テキストデータが含まれるかどうかを示すブール値。
Charset	URL で指定されたドキュメントで使用されている文字セット。 通常は、HTTP サーバーによってこの情報が提供されるか、または HTTP プロトコルの Content-Type ヘッダフィールドの charset パラメータに文字セットが指定されています。たとえば、次の HTTP ヘッダは、文字エンコードが EUC-JP であることを示しています。 Content-Type: text/html; charset=EUC-JP
Header	レスポンスヘッダの生データ。次にヘッダの例を示します。 HTTP/1.1 200 OK Date: Mon, 04 Mar 2002 17:27:44 GMT Server: Apache/1.3.22 (Unix) mod_perl/1.26 Set-Cookie: MM_cookie=207.22.48.162.4731015262864476; path=/; expires=Wed, 03-Mar-04 17:27:44 GMT; domain=.adobe.com Connection: close Content-Type: text/html
Filecontent	テキストおよび MIME ファイルの内容。
Mimetype	MIME タイプ。text/html、image/png、image/gif、video/mpeg、text/css、audio/basic などがあります。
responseHeader	レスポンスヘッダ。ヘッダキーが 1 つの場合、値は単純型としてアクセスできます。ヘッダキーが複数ある場合、値は responseHeader 構造体の配列に格納されます。
Statuscode	HTTP エラーコードと、関連するエラー文字列。レスポンスヘッダで返される一般的な HTTP ステータスコードには次のものがあります。 400: Bad Request 401: Unauthorized 403: Forbidden 404: Not Found 405: Method Not Allowed

CF.http 関数での HTTP Post パラメータの参照

HTTP Post パラメータを CF.http 関数に渡すには、オブジェクトの配列を作成し、この配列を params という変数に代入します。次の引数は、CF.http 関数の params 引数で、オブジェクト配列としてのみ渡すことができます。

パラメータ	説明
name	渡すデータの変数名。
type	トランザクションのタイプ。 • URL • FormField • Cookie • CGI • File
value	渡す URL、フォームフィールド、Cookie、ファイル、または CGI 変数の値。

次の例では、オブジェクトの配列に HTTP Post パラメータを含めて CF.http 関数で渡しています。

```
function postWithParamsAndUser()
{
    // Set up the array of Post parameters. These are just like cfhttpparam tags.
    params = new Array();
    params[1] = {name:"arg2", type:"URL", value:"value2"};

    url = "http://localhost:8500/";

    // Invoke with the method, url, params, username, and password
    result = CF.http("post", url, params, "karl", "salsa");
    return result.get("Filecontent");
}
```

CF.http Post メソッドの使用

Cookie、フォームフィールド、CGI、URL、およびファイル変数を、指定の ColdFusion ページや CGI プログラムに送信して処理させるには、Post メソッドを使用します。Post 操作では、送信する変数ごとに params 引数を使用します。Post メソッドから指定の ColdFusion ページや実行可能ファイルにデータが渡されると、送信した変数が解釈され、データが返されます。

たとえば、Post メソッドを使用した HTML フォームを作成する場合は、フォームデータを渡すページの名前を指定します。CF.http 関数でも同様の方法で Post メソッドを使用できます。ただし、CF.http 関数を使用する場合、Post を受け取ったページは何も表示しません。次の例を参照してください。

```
function postWithParams()
{
    // Set up the array of Post parameters. These are just like cfhttpparam tags.
    // This example passes formfield data to a specified URL.
    params = new Array();
    params[1] = {name:"Formfield1", type:"FormField", value:"George"};
    params[2] = [name:"Formfield2", type:"FormField", value:"Brown"};

    url = "http://localhost:8500/";

    // Invoke CF.http with the method, url, and params
    result = CF.http("post", url, params);
    return result.get("Filecontent");
}
```

CF.http Get メソッドの使用

指定のサーバーからテキストファイルやバイナリファイルなどのファイルを取得するには、Get メソッドを使用します。CF.http 関数から返されたオブジェクトのプロパティを参照して、ファイルコンテンツ、ヘッダ情報、MIME タイプなどの情報にアクセスします。

CF.http 関数を使用して、Web からデータを取得する一般的な方法を次に示します。

```
// Returns content of URL defined in url variable
// This example uses positional argument style
function get()
{
    url = "http://www.adobe.com/software/coldfusion/";

    //Invoke with just the url argument. Get is the default.
    result = CF.http(url);
    return result.get("Filecontent");
}
```

CF.http 関数のプロパティの詳細については、『CFML リファレンス』の CF.http を参照してください。

第 11 章：情報のリクエストと表示

データの取得および形式設定の概要

Adobe ColdFusion では、データの取得や形式設定が行えます。フォームを使用してユーザーからデータを取得したり、ダイナミック Web ページで表示するデータを制御することもできます。さらに、クエリー結果をテーブルに挿入したり、ColdFusion 関数を使用してデータの形式設定や操作を行うこともできます。これらの機能を使用するには、HTML フォームに精通している必要があります。

ColdFusion でのフォームの使用

ColdFusion では、さまざまなタイプのフォームを使用できます。プレーン HTML または CFML に加えて、HTML フォーム、Flash フォーム、またはスキン XML フォームも生成できます。

ColdFusion フォームタグ

HTML または CFML のタグを使用して、フォームを定義することができます。ColdFusion には、HTML の相当するタグを機能強化した、次の CFML タグが用意されています。

- cfapplet
- cfform
- cfinput
- cfselect
- cftextarea

これらのタグでは、対応する HTML タグのすべての属性に加えて、ColdFusion 固有の属性および機能もサポートされています。

また、HTML には直接相当するものがない、次のフォームタグも使用できます。

- cfcalendar: Flash の月別カレンダーで日付を選択できます。
- cfgrid: 行と列からなるグリッドでデータを表示したり入力したりできます。クエリーでデータを直接取得できます。
- cfslider: スライドマーカーを移動させてデータを入力できます。
- cftree: グラフィカルな階層ツリー形式でデータを表示します。クエリーで直接データを取得できます。

ColdFusion フォームタグの機能

ColdFusion フォームタグには、次の機能が用意されています。

ビルトイン検証のサポート クライアントブラウザまたはサーバーでデータを検証できます。フィールドが必須であるかどうか、格納できるデータのタイプ、最大長、または値の範囲を指定できます。さらに、データマスキングを使用して、ユーザーの入力を制御することもできます。検証の詳細については、729 ページの「データの検証」を参照してください。

注意：ColdFusion には、HTML フォームコントロールをサーバーで検証する機能も用意されています。

Flash 形式のフォームおよび要素 フォームを Flash として表示できます。Flash を使用すれば、さまざまなプラットフォームで同じ動作を提供できるとともに、HTML にはない高度な表示機能が使用できます。たとえば、複数のタブを持つアコーディオン形式のフォームペインや、要素の自動配置機能などが使用できます。また、cftree、cfgrid、および cfcalendar

フォーム要素を、HTML フォームで Flash アイテムとして表示することもできます。Flash フォームおよびフォーム要素の詳細については、751 ページの「Flash フォームの作成」を参照してください。

XML スキン可能フォーム ColdFusion では、生成した XML フォームに XSLT スキンを適用して、フォームの形式を設定することができます。XML 形式設定フォームを使用すれば、フォームのプレゼンテーションと、フォームのロジックやデータフィールド情報を分離することができます。これにより、カスタムスキンを適用してフォームの外観を細かく制御したり、カスタムコントロールを作成したりすることができます。XML スキン可能フォームの詳細については、768 ページの「スキン可能 XML フォームの作成」を参照してください。

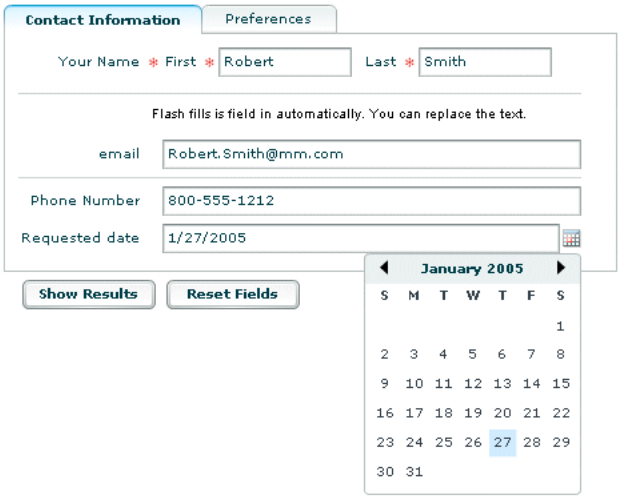
ColdFusion 変数の直接サポート ColdFusion 変数を直接使用して、フォームコントロールに簡単に値を設定できます。たとえば、cfgrid および cftree タグでクエリー結果を指定して、データを設定することができます。

強力で柔軟な機能を備えた CFML フォームタグを使用すれば、機能性の高い魅力的なフォームを簡単に開発できます。

CFML タグには固有の機能が多数用意されていますが、それらの説明や使用方法については、ここではほとんど触れていません。cftree や cfgrid などの、ColdFusion 特有のタグの使用方法の詳細については、709 ページの「cfform タグによるダイナミックフォームの作成」を参照してください。

基本フォームの作成

次の例では、ユーザーがデータを入力できる単純なフォームの作成方法を示します。このフォームでは、基本的な CFML フォームタグを使用します。検証、Flash/XML 形式、特殊な入力コントロールなどの ColdFusion の高度な機能は使用しません。タグ名の冒頭にある "cf" という接頭辞を削除すれば、純粋な HTML フォームとして実際に使用することができます。



次の表に、フォームコントロールタグのシンタックスを示します。

コントロール	コード
テキストコントロール	`<cfinput type="Text" name="ControlName" size="Value" maxlength="Value">`

リスト (選択) ボックス	`<cfselect name="ControlName">`
	`<option value="Value1">DisplayName1`
	`<option value="Value2">DisplayName2`
	`<option value="Value3">DisplayName3`
	`</cfselect>`

ラジオボタン	`<cfinput type="Radio" name="ControlName" value="Value1">DisplayName1`
	`<cfinput type="Radio" name="ControlName" value="Value2">DisplayName2`
	`<cfinput type="Radio" name="ControlName" value="Value3">DisplayName3`

コントロール	コード
チェックボックス	`<cfinput type="Checkbox" name="ControlName" value="Yes|No">Yes`
リセットボタン	`<cfinput type="Reset" name="ControlName" value="DisplayName">`
送信ボタン	`<cfinput type="Submit" name="ControlName" value="DisplayName">`

次に、フォームのソースを詳細に示します。後述の例で、このフォームを使用して入力を行うには、"formpage.cfm" という名前を付けてこのコードを保存します。

```
<html>
<head>
<title>Input form</title>
</head>
<body>
<!--- Specify the action page in the form tag. The form variables will
        pass to this page when the form is submitted. --->

<cfform action="actionpage.cfm" method="post">

<!--- Text box. --->
<p>
First Name: <cfinput type="Text" name="FirstName" size="20"maxlength="35"><br>
Last Name: <cfinput type="Text" name="LastName" size="20" maxlength="35"><br>
Salary: <cfinput type="Text" name="Salary" size="10" maxlength="10">
</p>

<!--- List box. --->
<p>
City
<cfselect name="City">
    <option value="Arlington">Arlington
    <option value="Boston">Boston
    <option value="Cambridge">Cambridge
    <option value="Minneapolis">Minneapolis
    <option value="Seattle">Seattle
</cfselect>
</p>

<!--- Radio buttons. --->
<p>
Department:<br>
<cfinput type="radio" name="Department" value="Training">Training<br>
<cfinput type="radio" name="Department" value="Sales">Sales<br>
<input type="radio" name="Department"
    value="Marketing">Marketing<br>
</p>

<!--- Check box. --->
<p>
Contractor? <cfinput type="checkbox" name="Contractor"
    value="Yes" checked>Yes
</p>

<!--- Reset button. --->
<cfinput type="Reset" name="ResetForm" value="Clear Form">
<!--- submit button --->
<cfinput type="Submit" name="SubmitForm" value="Submit">

</cfform>
</body>
</html>
```

フォームに関するガイドライン

フォームを使用するときは、次のガイドラインに従ってください。

- フォームコントロールの名前を、対応するデータベースフィールドと同じ名前にすると、コードが読みやすくなります。たとえば、データソースの FirstName フィールドに対応するテキストコントロールは、FirstName という名前にします。
- ラジオボタンの選択肢は、3 ～ 5 個にすると使いやすくなります。それ以上の選択肢が必要な場合は、ドロップダウンリストを使用することを検討してください。
- リストボックスでは、ユーザーに多数の選択肢の中から選択させたり、1 つのリストから複数の項目を選択させたりすることができます。
- 選択されていないチェックボックス、ラジオボタン、およびリストボックスは、アクションページにデータが渡されません。存在しない変数をアクションページで参照しようとすると、エラーになります。変数が存在するかどうかをアクションページで判断する方法については、697 ページの「変数の存在のテスト」を参照してください。
- ドロップダウンリストには、クエリーデータを使用してダイナミックに項目を挿入することができます。詳細については、705 ページの「リストボックス項目のダイナミックな設定」を参照してください。

アクションページの操作

ユーザーがフォームを送信すると、cfform または form タグの action 属性で指定されているアクションページが ColdFusion により実行されます。ColdFusion アクションページは、通常のアプリケーションページとほぼ同じですが、フォームから送られてくるフォーム変数を使用できる点が異なります。

アクションページでのフォーム変数の処理

フォームが送信されると、値が設定されているすべてのフォームコントロールのフォーム変数が、アクションページに渡されます。

注意：複数のコントロールが同じ名前を持つ場合は、1 つのフォーム変数が、カンマ区切りの値のリストとしてアクションページに渡されます。

フォーム変数の名前は、フォームページ上のフォームコントロールに割り当てられている名前と同じです。フォーム変数は、アクションページのタグ、関数、式などの中で、名前を使用して参照します。

フォーム変数は、アクションページの Form スコープに含まれています。フォーム変数を参照するときは、"Form." という接頭辞を付けて、フォーム変数を参照していることを明示的に示します。たとえば、次のコードでは、アクションページに出力するために LastName フォーム変数を参照しています。

```
<cfoutput>
    #Form.LastName#
</cfoutput>
```

また、Form スコープには、Form.fieldnames というリスト変数も含まれています。この変数には、アクションページに送信されたすべてのフォーム変数のリストが含まれています。アクションページにフォーム変数が渡されなかった場合、Form.fieldnames リストは作成されません。

フォームデータを使用した SQL ステートメントの生成

前の章で説明したように、次のようなクエリーを作成することで、データベーステーブル内のすべての従業員のレコードを取得できます。

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECTFirstName, LastName, Contract
    FROM Employee
</cfquery>
```

検索条件に一致する従業員の情報を取得したい場合は、SQL SELECT ステートメントで SQL WHERE 節を使用します。WHERE 節で比較条件を指定すると、その結果に基づいてクエリーデータがフィルタ処理されます。

たとえば、姓が Smith という従業員のデータのみを取得するには、次のようなクエリーを作成します。

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT FirstName, LastName, Contract
    FROM Employee
    WHERE LastName = 'Smith'
</cfquery>
```

実際のアプリケーションでは、SQL WHERE 節に LastName の値を直接記述するのではなく、次のように、ユーザーがフォームに入力したテキストを使用できます。

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary
    FROM Employee
    WHERE LastName=<cfqueryparam value="#Form.LastName#"
    CFSQLType="CF_SQL_VARCHAR">
</cfquery>
```

セキュリティを確保するために、この例ではフォーム変数を cfqueryparam タグ内にカプセル化して、ユーザーが LastName に有効な文字列値を渡したことを確認しています。クエリーで cfqueryparam タグを使用する方法や、ダイナミック SQL の詳細については、413 ページの「データのアクセスおよび取得」を参照してください。

アクションページの作成

前に作成したページ "formpage.cfm" のアクションページを作成するには、次の手順を行います。

フォームのアクションページの作成

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>Retrieving Employee Data Based on Criteria from Form</title>
</head>

<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary
    FROM Employee
    WHERE LastName=<cfqueryparam value="#Form.LastName#"
    CFSQLType="CF_SQL_VARCHAR">
</cfquery>
<h4>Employee Data Based on Criteria from Form</h4>
<cfoutput query="GetEmployees">
#FirstName#
#LastName#
#Salary#<br>
</cfoutput>
<br>
<cfoutput>Contractor: #Form.Contractor#</cfoutput>
</body>
</html>
```

2 このページに "actionpage.cfm" という名前を付けて、"myapps" ディレクトリに保存します。

3 "formpage.cfm" ページをブラウザで表示します。

4 データを入力します。たとえば、姓のテキストボックスに「Smith」と入力し、フォームを送信します。

ブラウザには、入力した姓に一致するデータベースエントリの姓名および給与が 1 行ずつ表示され、各行の下に " 契約社員 : Yes" という行が表示されます。

5 ブラウザの [戻る] をクリックしてフォームを再度表示します。

6 チェックボックスを無効にして、フォームを再度送信します。

チェックボックスによって変数がアクションページに渡されないので、エラーになります。"actionpage.cfm" を編集してエラーを修正する方法については、697 ページの「変数の存在のテスト」を参照してください。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
`<cfquery name="GetEmployees"datasource="cfdocexamples">`	クエリーに GetEmployees という名前を付けて、データソース cfdocexamples にクエリーを実行します。

`SELECT FirstName, LastName, Salary`
`FROM Employee`
`WHERE LastName=<cfqueryparam value="#Form.LastName#"`
`CFSQLType="CF_SQL_VARCHAR">`
	Employee テーブルから FirstName、LastName、および Salary フィールドを取得します。ただし、LastName フィールドの値が、"formpage.cfm" フォームの LastName テキストボックスに入力された値に一致している場合にのみ取得します。

`<cfoutput query="GetEmployees">`	GetEmployees クエリーの結果を表示します。
`#FirstName#` `#LastName#` `#Salary# `	各レコードの FirstName、LastName、Salary フィールドの値を、順番に 1 行ずつ表示します。SELECT ステートメントの条件に一致するすべてのレコードが表示されます。各レコードは改行で区切られます。
`</cfoutput>`	cfoutput ブロックを閉じます。
` ` `<cfoutput>Contractor: #Form.Contractor#` `</cfoutput>`	空白行を挿入し、その下に " 契約社員 " というテキストと、フォームの Contractor チェックボックスの値を表示します。 次のさらに詳細な例では、変数の存在を確認し、クエリーでその変数を使用します。

変数の存在のテスト

アプリケーションページで変数の存在を確認するには、ColdFusion の IsDefined 関数を使用します。関数とは、入力を渡すとそれに応じて処理が実行される名前付きのプロシージャのことです。たとえば、IsDefined 関数は変数が存在するかどうかを示します。CFML では、多数の関数が使用できます。詳細については、『CFML リファレンス』を参照してください。

次のコードのように、Contractor フォーム変数の存在を確認してから使用すれば、前の例のようなエラーは発生しません。

```
<cfif IsDefined("Form.Contractor")>
    <cfoutput>Contractor: #Form.Contractor#</cfoutput>
</cfif>
```

IsDefined 関数に渡す引数は、常に二重引用符 (") で囲む必要があります。IsDefined 関数の詳細については、『CFML リファレンス』を参照してください。

定義されていない変数を評価しようとすると、ページが処理できず、エラーメッセージが表示されます。このような問題の原因を特定するには、ColdFusion Administrator のデバッグ機能を有効にします。ColdFusion Administrator のデバッグ情報には、どの変数がアプリケーションページに渡されているかが示されます。

必須入力フィールド

HTML フォームの弱点の 1 つは、入力フィールドを必須フィールドとして定義できないことです。データベースアプリケーションにとって、フィールドの必須化は重要な機能であり、ColdFusion でもその機能が使用できます。フィールドを必須にするには、次のいずれかの手順を行います。

- cfinput、cfselect、cftextarea、および cftree タグの required 属性を使用します。

- 非表示フィールドの name 属性で、目的のフィールド名に接尾辞 _required を付加した値を指定します。この方法は、CFML および HTML フォームタグで使用できます。

たとえば、cfinput タグの "FirstName" フィールドに対して値の入力を義務付けるには、次のシンタックスを使用します。

```
<cfinput type="Text" name="FirstName" size="20" maxlength="35" required="Yes">
```

HTML の input タグの "FirstName" フィールドに対して値の入力を義務付けるには、次のシンタックスを使用します。

```
<input type="Text" name="FirstName" size="20" maxlength="35">
<input type="hidden" name="FirstName_required">
```

いずれの例でも、ユーザーが "FirstName" フィールドを空欄のままにすると、ColdFusion によってフォームの送信が拒否され、そのフィールドが必須であることを知らせるメッセージが返されます。このエラーメッセージの内容はカスタマイズすることができます。

required 属性を使用している場合は、次のように、message 属性を使用してメッセージをカスタマイズします。

```
<cfinput type="Text" name="FirstName" size="20" maxlength="35" required="Yes"
    message="You must enter your first name.">
```

非表示フィールドタグを使用している場合は、次のように、非表示フィールドの value 属性を使用してメッセージをカスタマイズします。

```
<input type="hidden" name="FirstName_required"
    value="You must enter your first name.">
```

フォーム変数に関する注意事項

アクションページでフォーム変数を使用するときは、次のガイドラインに従ってください。

- フォーム変数は、アクションページだけでなく、アクションページでインクルードするページでも使用できます。
- アクションページでフォーム変数を参照するときは、フォーム変数に "Form." という接頭辞を付けます。
- 変数値を出力するには # 記号で囲みます。
- 1 を超える size 属性が設定されたチェックボックス、ラジオボタン、リストボックスの変数は、オプションが選択された場合にのみアクションページに渡されます。テキストボックス、パスワード、およびテキストエリアフィールドは、何も入力されていなくても空の文字列が渡されます。
- アクションページで、渡されていない変数を使用しようとすると、エラーになります。
- 複数のコントロールが同じ名前を持つ場合は、1 つのフォーム変数が、カンマ区切りの値のリストとしてアクションページに渡されます。
- フォーム変数値の検証はクライアントまたはサーバーで行えます。

クエリーとデータの操作

クエリーデータの生成と表示は、ColdFusion の最も重要で柔軟な機能の 1 つです。この説明のいくつかは、クエリー結果だけでなく、あらゆるデータの表示にも使用できます。

HTML テーブルでのクエリー結果の表示

HTML テーブルを使用して、ページ上でのクエリー結果の表示方法を指定できます。クエリー結果を表示するには、テーブルタグの中に cfoutput タグを組み込みます。また、HTML の th タグを使用して、ヘッダ行に列ラベルを付けることができます。クエリー結果の行ごとにテーブル行を作成するには、cfoutput タグの中に tr ブロックを置きます。

また、CFML 関数を使用して、日付や数値などの形式を設定することもできます。

テーブルへのクエリー結果の取り込み

1 エディタで ColdFusion の "actionpage.cfm" ページを開きます。

2 次のようにページを変更します。

```
<html>
<head>
<title>Retrieving Employee Data Based on Criteria from Form</title>
</head>

<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary
    FROM Employee
    WHERE LastName=<cfqueryparam value="#Form.LastName#"
    CFSQLType="CF_SQL_VARCHAR">
</cfquery>
<h4>Employee Data Based on Criteria from Form</h4>
<table>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Salary</th>
</tr>
<cfoutput query="GetEmployees">
<tr>
<td>#FirstName#</td>
<td>#LastName#</td>
<td>#Salary#</td>
</tr>
</cfoutput>
</table>
<br>
<cfif IsDefined("Form.Contractor")>
    <cfoutput>Contractor: #Form.Contractor#</cfoutput>
</cfif>
</body>
</html>
```

3 このページに "actionpage.cfm" という名前を付けて、"myapps" ディレクトリに保存します。

4 "formpage.cfm" ページをブラウザで表示します。

5 LastName テキストボックスに「Smith」と入力し、フォームを送信します。

フォームで指定した条件に一致するレコードがテーブルに表示されます。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
`<table>`	データをテーブルに配置します。

`<tr>`
`<th>First Name</th>`
`<th>Last Name</th>`
`<th>Salary</th>`
`</tr>`
	テーブルの 1 行目に、見出しとなる 3 つの列 "First Name"、"Last Name"、および "Salary" を表示します。

`<cfoutput query="GetEmployees">`	GetEmployees クエリーの結果を表示するように指定します。

コード	説明

`<tr>`
`<td>#FirstName#</td>`
`<td>#LastName#</td>`
`<td>#Salary#</td>`
`</tr>`
	クエリー内の各レコードに対して、テーブル内に行を作成します。1 つの行には、レコードの FirstName、LastName、および Salary フィールドの値を表示する 3 つの列を作成します。

`</cfoutput>`	出力領域を終了します。
`</table>`	テーブルを終了します。

データ項目の形式設定

データ項目は個別に形式設定できます。たとえば、給与データは金額として形式設定できます。給与データをドルの形式に設定するには、CFML 関数の DollarFormat を使用します。

Salary の形式の変更

1 エディタでファイル "actionpage.cfm" を開きます。

2 次の行を、

```
<td>#Salary#</td>
```

次のように変更します。

```
<td>#DollarFormat(Salary)#</td>
```

3 このページを保存します。

フレキシブルな検索インターフェイスの構築

フォームの用途の 1 つが、フォームデータを利用した検索です。たとえば、WHERE 節の中でフォームデータを使用して、データベースクエリーを作成できます。

複数の検索条件を指定できるようにする場合は、WHERE 節内の条件論理を SQL AND 節で囲みます。次のアクションページでは、従業員を部門、姓、またはその両方で検索できます。

より柔軟な検索インターフェイスの構築

1 エディタで ColdFusion の "actionpage.cfm" ページを開きます。

2 次のようにページを変更します。

```
<html>
<head>
<title>Retrieving Employee Data Based on Criteria from Form</title>
</head>
<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT Departmt.Dept_Name,
        Employee.FirstName,
        Employee.LastName,
        Employee.StartDate,
        Employee.Salary
    FROM Departmt, Employee
    WHERE Departmt.Dept_ID = Employee.Dept_ID
    <cfif IsDefined("Form.Department")>
    AND Departmt.Dept_Name=<cfqueryparam value="#Form.Department#"
                CFSQLType="CF_SQL_VARCHAR">
    </cfif>
    <cfif Form.LastName IS NOT "">
        AND Employee.LastName=<cfqueryparam value="#Form.LastName#"
                CFSQLType="CF_SQL_VARCHAR">
```

```
    </cfif>
</cfquery>

<h4>Employee Data Based on Criteria from Form</h4>
<table>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Salary</th>
</tr>
<cfoutput query="GetEmployees">
<tr>
<td>#FirstName#</td>
<td>#LastName#</td>
<td>#Salary#</td>
</tr>
</cfoutput>
</table>
</body>
</html>
```

3 ファイルを保存します。

4 "formpage.cfm" ページをブラウザで表示します。

5 部門を選択し (姓も入力可)、フォームを送信します。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明

`SELECT Departmt.Dept_Name,`
`Employee.FirstName,`
`Employee.LastName,`
`Employee.StartDate,`
`Employee.Salary`
`FROM Departmt, Employee`
`WHERE Departmt.Dept_ID = Employee.Dept_ID`
	Departmt テーブルおよび Employee テーブルを、各テーブルの "Dept_ID" フィールドに基づいて結合し、リストされているフィールドを取得します。

`<cfif IsDefined("Form.Department")>`
`AND Departmt.Dept_Name=<cfqueryparam`
`value="#Form.Department#"`
`CFSQLType="CF_SQL_VARCHAR">`
`</cfif>`
	フォームで特定の部門が指定された場合は、指定された部門名を持つレコードのみを取得します。SQL AND ステートメントでは、Form.Department を ColdFusion 変数として識別させるために # 記号を使用しますが、IsDefined 関数では # 記号を使用しません。

`<cfif Form.LastName IS NOT "">`
`AND Employee.LastName=<cfqueryparam`
`value="#Form.LastName#"`
`CFSQLType="CF_SQL_VARCHAR">`
`</cfif>`
	フォームで姓が指定された場合は、指定された姓を持つレコードのみを取得します。

ユーザーへのクエリー結果の提示

ユーザーに結果を戻すときは、ユーザーのニーズに合った、適切な種類および量の情報を表示するようにしてください。特に、次のような状況を考慮します。

- クエリー結果がない場合
- すべての結果が得られるまでに時間がかかる場合

クエリー結果がない場合の処理

結果を表示するコードでは、クエリーでレコードが返されなかった場合も考慮する必要があります。検索でレコードが取得されたかどうかを判断するには、RecordCount クエリー変数を使用します。この変数を条件論理式の中で使用して、ユーザーに適切な検索結果を表示します。

注意：RecordCount などのクエリー変数の詳細については、413 ページの「データのアクセスおよび取得」を参照してください。

たとえば、GetEmployees クエリーでレコードが検出されなかった場合にそれを通知するには、データを表示するコードの前に次のコードを挿入します。

```
<cfif GetEmployees.RecordCount IS "0">
    No records match your search criteria. <BR>
<cfelse>
```

次の手順に従います。

- RecordCount には、クエリー名の接頭辞を付けます。
- cfif タグの後に、ユーザーにメッセージを表示するコードを記述します。
- cfelse タグの後に、返されたデータの形式を設定するコードを記述します。
- 2 番目のコードの後に終了タグ </cfif> を配置して、条件コードを終了します。

ユーザーに検索結果を返す

1 "actionpage.cfm" ページを編集します。

2 次のようにページを変更します。

```
<html>
<head>
<title>Retrieving Employee Data Based on Criteria from Form</title>
</head>

<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT Departmt.Dept_Name,
        Employee.FirstName,
        Employee.LastName,
        Employee.StartDate,
        Employee.Salary
    FROM Departmt, Employee
    WHERE Departmt.Dept_ID = Employee.Dept_ID
    <cfif isdefined("Form.Department")>
        AND Departmt.Dept_Name = <cfqueryparam value="#Form.Department#"
            CFSQLType="CF_SQL_VARCHAR">
    </cfif>
    <cfif Form.LastName is not "">
        AND Employee.LastName = <cfqueryparam value="#Form.LastName#"
            CFSQLType="CF_SQL_VARCHAR">
    </cfif>
</cfquery>

<cfif GetEmployees.recordcount is "0">
No records match your search criteria. <br>
```

```
Please go back to the form and try again.
<cfelse>
<h4>Employee Data Based on Criteria from Form</h4>
<table>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Salary</th>
</tr>
<cfoutput query="GetEmployees">
<tr>
<td>#FirstName#</td>
<td>#LastName#</td>
<td>#Salary#</td>
</tr>
</cfoutput>
</cfif>
</table>
</body>
</html>
```

3 ファイルを保存します。

4 フォームに戻り、検索条件を入力してフォームを送信します。

5 指定した条件に一致するレコードがない場合は、メッセージが表示されます。

時間のかかる結果を少しずつ返す

cfflush タグを使用すると、処理に時間のかかるリクエストの結果を、ページ全体の処理が完了する前に、少しずつ表示させることができます。このタグによって、リクエストの処理に時間がかかる場合でも、ユーザーに迅速にフィードバックできます。たとえば、結果を返すのに時間がかかる場合に、cfflush タグを使用して " 処理中ですのでお待ちください " というメッセージを表示できます。また、長いリストを、取得したものから少しずつ表示することもできます。

ページの最初の cfflush タグでは、すべての HTML ヘッダおよび利用可能なその他の HTML データがブラウザに送られます。以降の cfflush タグでは、前回のフラッシュの後に生成された出力のみが送信されます。

interval 属性を指定すれば、一定のバイト数のデータが揃うたびに出力をフラッシュさせることができます。このバイト数には、このタグが呼び出された時点で既に利用可能な HTML ヘッダおよびその他のデータは含まれません。cfloop タグで cfflush タグを使用して、取得できたデータから少しずつフラッシュすることができます。これは、データの量が多くてクエリーの反応が遅い場合に役立ちます。

データのフラッシュは、十分な量のデータが揃ってから実行してください。ブラウザによっては、少量のデータには反応しないことがあります。また、interval 属性は数百バイト程度の適切なサイズに設定してください。数千バイトでは大きすぎます。

cfflush タグの制限 : cfflush タグを実行するとブラウザにデータが送信されるので、このタグには次のような制限があります。

- ページ上で cfflush タグの後に cfcontent、cfcookie、cfform、cfheader、cfhtmlhead、cflocation、SetLocale といったタグや関数を使用すると、エラーが起きるか、予期せぬ結果になります。これらのタグや関数は、通常は HTML ヘッダを変更するものですが、cfflush タグによってヘッダが送信されてしまうので、その後にこれらを記述しても正常に処理されません。
- cfflush タグを含むページ上の任意の場所で cfset タグを使用して Cookie を設定しても、ブラウザ内に Cookie は設定されません。
- cfsavecontent、cfqueryparam、カスタムタグなど、いくつかのタグの本文内で cfflush タグを使用すると、エラーが発生することがあります。

- クライアント変数を Cookie として保存する場合、cfflush タグの後で設定したクライアント変数はブラウザ内に保存されません。
- Cookie のエラー以外の cfflush エラーは、cfcatch タグで検出できます。Cookie のエラーは、cfcatch type="Any" タグで検出します。

例 : cfloop タグと Rand 関数の使用

次の例では、cfloop タグと、乱数を生成する Rand 関数を使用して、作為的に表示データの生成を遅らせます。この例では、最初のデータ取得に時間がかかり、追加データの表示も遅いというケースをシミュレーションします。

```
<html>
<head>
    <title>Your Magic numbers</title>
</head>

<body>
<h1>Your Magic numbers</h1>
<P>It will take us a little while to calculate your ten magic numbers.
It takes a lot of work to find numbers that truly fit your personality.
So relax for a minute or so while we do the hard work for you.</P>
<h2>We are sure you will agree it was worth the short wait!</h2>
<cfflush>

<cfflush interval=10>
<!--- Delay Loop to make is seem harder. --->
<cfloop index="randomindex" from="1" to="200000" step="1">
    <cfset random=rand()>
</cfloop>

<!--- Now slowly output 10 random numbers. --->
<cfloop index="Myindex" from="1" to="10" step="1">
    <cfloop index="randomindex" from="1" to="100000" step="1">
        <cfset random=rand()>
    </cfloop>
    <cfoutput>
        Magic number #Myindex# is:  #RandRange(
    100000, 999999)#<br><br>
    </cfoutput>
</cfloop>
</body>
</html>
```

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<h2>We are sure you will agree it was worth the short wait!</h2>` `<cfflush>`	HTML ヘッダと cfflush タグへのすべての HTML 出力をユーザーに送ります。これによって、説明文と H2 タグの内容が表示されます。

コード	説明
`<cfflush interval=10>`	10 バイト以上のデータが得られるたびにユーザーにフラッシュします。

`<cfloop index="randomindex" from="1" to="200000" step="1">`
`<cfset random=Rand()>`
`</cfloop>`
	Rand 関数で多数の乱数を計算させて、作為的に遅延を発生させます。

`<cfloop index="Myindex" from="1" to="10" step="1">` `<cfloop index="randomindex" from="1" to="100000" step="1">` `<cfset random=rand()>` `</cfloop>` `<cfoutput>` `Magic number #Myindex# is:  #RandRange (100000,999999)#  ` `</cfoutput>` `</cfloop>`	10 個の乱数を生成して表示します。このコードでは 2 つのループを使用します。外側のループは、乱数を 10 回表示するためのものです。内側のループは、Rand 関数で多数の (使用しない) 乱数を生成して、遅延を発生させるためのものです。その後で、RandRange 関数を呼び出して 6 桁の乱数を生成し、表示します。

リストボックス項目のダイナミックな設定

693 ページの「基本フォームの作成」のコードでは、フォーム内のリストボックスのオプションをハードコードしました。フォームに情報を直接入力する代わりに、データベースフィールドを使用して、リストボックスの項目をダイナミックに設定できます。このようにすると、データベースで行った変更がフォームページに自動的に反映されます。

リストボックスをダイナミックに設定するには、2 つのタグを使用します。

- cfquery タグを使用して、データベーステーブルから列データを取得します。
- query 属性を指定した cfselect タグを使用して、このフォームコントロールの option をダイナミックに設定します。

リストボックスのダイナミックな設定

1 "formpage.cfm" ページを開きます。

2 次のようにファイルを変更します。

```
<html>
<head>
<title>Input form</title>
</head>
<body>
<cfquery name="GetDepartments" datasource="cfdocexamples">
    SELECT DISTINCT Location
    FROM Departmt
</cfquery>

<!--- Define the action page in the form tag.
    The form variables pass to this page
    when the form is submitted --->

<cfform action="actionpage.cfm" method="post">

<!--- Text box. --->
<p>
First Name: <cfinput type="Text" name="FirstName" size="20" maxlength="35"><br>
Last Name: <cfinput type="Text" name="LastName" size="20" maxlength="35"><br>
Salary: <cfinput type="Text" name="Salary" size="10" maxlength="10">
</p>

<!--- List box. --->
City
<cfset optsize=getDepartments.recordcount + 1>
<cfselect name="City" query="GetDepartments" value="Location" size="#optsize#">
    <option value="">Select All
</cfselect>
```

```
<!--- Radio buttons. --->
<p>
Department:<br>
<cfinput type="radio" name="Department" value="Training">Training<br>
<cfinput type="radio" name="Department" value="Sales">Sales<br>
<cfinput type="radio" name="Department" value="Marketing">Marketing<br>
<cfinput type="radio" name="Department" value="HR">HR<br>
</p>

<!--- Check box. --->
<p>
Contractor? <cfinput type="checkbox" name="Contractor" value="Yes" checked>Yes
</p>

<!--- Reset button. --->
<cfinput type="reset" name="ResetForm" value="Clear Form">

<!--- Submit button. --->
<cfinput type="submit" name="SubmitForm" value="Submit">
</cfform>
</body>
</html>
```

3 このページに "formpage.cfm" という名前を付けて保存します。

4 "formpage.cfm" ページをブラウザで表示します。

フォームに変更が反映されます。

値を送信するにはアクションページが必要です。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明

`<cfquery name="GetDepartments" datasource="cfdocexamples">`
`SELECT DISTINCT Location`
`FROM Departmt`
`</cfquery>`
	Departmt テーブル内のすべての部門の場所を抽出します。DISTINCT 節によって、重複する場所の名前がクエリー結果から削除されます。

`<cfset optsize=getDepartments.recordcount + 1>`	選択リストにダイナミックに追加するエントリ数に 1 (手動で追加する [すべて選択] オプションの分) を加えた数値を optsize 変数に設定します。

`<cfselect name="City" query="GetDepartments" value="Location"`
`size="#optsize#">`
`<option value="">Select All`
`</cfselect>`
	GetDepartments クエリーの Location 列を使用して、市の選択リストを設定します。これによって、クエリーの各行に対して 1 つのオプションが設定されます。
	すべての場所を選択するオプションを追加します。このオプションを選択した場合、フォームの値は空の文字列になります。アクションページでは、空の文字列をチェックして、適切に処理する必要があります。

ダイナミックチェックボックスと複数選択リストボックスの作成

HTML または CFML フォームに、同じ名前を持つチェックボックスが複数ある場合や、複数選択リストボックス (リストから複数の項目を選択できるボックス) がある場合、ユーザーの選択した値はカンマ区切りのリストとして渡されます。これらのリストは、さまざまな状況で役立ちます。

注意：ユーザーがチェックボックスを 1 つも選択しなかった場合、またはリストボックスから何も選択しなかった場合、変数は作成されません。値が存在しない場合、cfinput および cfupdate タグは正常に機能しません。このエラーを防ぐには、フォームフィールドを必須化するか、ダイナミック SQL を使用するか、または cfparam タグを使用してフォームフィールドのデフォルト値を設定します。

チェックボックス

同じ名前を持つチェックボックスをフォームに複数配置した場合、作成される変数は、それらの値をカンマで区切ったリストになります。値は、数値か英数字文字列のいずれかです。この 2 つのデータ型は、処理方法が若干異なります。

数値の処理

1 つまたは複数の部門を、チェックボックスで選択できるようにします。選択された部門に基づいて、データベースに詳細情報をクエリーします。部門選択用のチェックボックスのコードは、次のようになります。

```
<cfinput type="checkbox"
    name="SelectedDepts"
    value="1">
    Training<br>

<cfinput type="checkbox"
    name="SelectedDepts"
    value="2">
    Marketing<br>

<cfinput type="checkbox"
    name="SelectedDepts"
    value="3">
    HR<br>

<cfinput type="checkbox"
    name="SelectedDepts"
    value="4">
    Sales<br>
</html>
```

ユーザーには部門名を表示しますが、各チェックボックスの value 属性には、データベースの部門レコードのプライマリキーに対応する数字を使用します。

ユーザーが [Marketing] と [Sales] を選択した場合、SelectedDepts フォームフィールドの値は 2,4 になります。SelectedDepts の値は、次の SQL ステートメントで使用します。

```
SELECT *
    FROM Departmt
    WHERE Dept_ID IN ( #Form.SelectedDepts# )
```

データベースには次のステートメントが送られます。

```
SELECT *
    FROM Departmt
    WHERE Dept_ID IN ( 2,4 )
```

文字列値の処理

数値ではなく文字列値が含まれているデータベースフィールドを検索するには、checkbox と cfquery のシンタックスを変更し、単一引用符 (') で囲まれた文字列値をデータソースに渡すようにしてください。

最初の例では、Dept_ID という数値プライマリキーに基づいて部門情報を検索しましたが、この例では、文字列値が含まれている Dept_Name というデータベースフィールドがプライマリキーであると想定します。この場合、チェックボックスのコードは次のようになります。

```
<cfinput type="checkbox"
    name="SelectedDepts"
    value="Training">
    Training<br>

<cfinput type="checkbox"
    name="SelectedDepts"
    value="Marketing">
    Marketing<br>

<cfinput type="checkbox"
    name="SelectedDepts"
    value="HR">
    HR<br>

<cfinput type="checkbox"
    name="SelectedDepts"
    value="Sales">
    Sales<br>
```

ユーザーが [Marketing] と [Sales] を選択した場合、SelectedDepts フォームフィールドの値は、Marketing,Sales というリストになります。SQL ステートメントは次のようになります。

```
SELECT *
    FROM Departmt
    WHERE Dept_Name IN
    (#ListQualify(Form.SelectedDepts,"'")#)
```

SQL では、すべての文字列を単一引用符で囲む必要があります。ListQualify 関数は、各項目を指定の修飾文字 (ここでは単一引用符) で囲んだリストを返します。

フォームで 2 番目と 4 番目のチェックボックスを選択した場合は、次のステートメントがデータベースに送られます。

```
SELECT *
    FROM Departmt
    WHERE Dept_Name IN ('Marketing','Sales')
```

複数選択リスト

複数選択リストボックスを定義するには、select または cfselect タグで、multiple または multipe="yes" 属性を指定し、size 属性に 1 を超える値を設定します。複数選択リストボックスで複数の選択肢を選択した場合も、複数のチェックボックスを選択した場合と同じように処理されます。複数選択リストボックスのデータは、選択されたエントリをカンマで区切ったリストとしてページに渡されます。たとえば、リストボックスに [Training]、[Marketing]、[HR]、[Sales] の 4 つのエントリがあり、ユーザーが [Marketing] と [Sales] を選択した場合、フォームフィールドの変数値は Marketing,Sales になります。

チェックボックスの場合と同様に、複数選択リストを使用してデータベースを検索できます。

数値の処理

複数選択リストボックスで部門を選択できるようにします。選択された部門に基づいて、詳細情報をクエリーします。

```
Select one or departments to get more information on:
<cfselect name="SelectDepts" multiple>
    <option value="1">Training
    <option value="2">Marketing
    <option value="3">HR
    <option value="4">Sales
</cfselect>
```

ユーザーが [Marketing] と [Sales] を選択した場合、SelectDepts フォームフィールドの値は 2,4 になります。このパラメータは、次の SQL ステートメントで使用します。

```
SELECT *
    FROM Departmt
    WHERE Dept_ID IN (#form.SelectDepts#)
```

次のステートメントがデータベースに送られます。

```
SELECT *
    FROM Departmt
    WHERE Dept_ID IN (2,4)
```

文字列値の処理

複数選択リストボックスで部門を選択できるようにします。データベースの検索フィールドは、文字列フィールドです。選択された部門に基づいて、詳細情報をクエリーします。

```
<cfselect name="SelectDepts" multiple>
    <option value="Training">Training
    <option value="Marketing">Marketing
    <option value="HR">HR
    <option value="Sales">Sales
</cfselect>
```

ユーザーが [Marketing] と [Sales] を選択した場合、SelectDepts フォームフィールドの値は Marketing,Sales になります。

チェックボックスで文字列値のデータベースフィールドを検索する場合と同様に、複数選択リストボックスでも ColdFusion の ListQualify 関数を使用します。

```
SELECT *
    FROM Departmt
    WHERE Dept_Name IN (#ListQualify(Form.SelectDepts,"'")#)
```

次のステートメントがデータベースに送られます。

```
SELECT *
    FROM Departmt
    WHERE Dept_Name IN ('Marketing','Sales')
```

cfform タグによるダイナミックフォームの作成

cfform タグを使用すれば、Java アプレットや Flash コントロールなどの高度なグラフィカルコントロールを備えた、高機能なダイナミックフォームを作成できます。これらのコントロールは、Java や Flash のコードを記述しなくても作成できます。

cfform タグによるカスタムフォームの作成

cfform タグとその CFML サブタグを使用して、次に示す 3 つの形式のダイナミックフォームを作成できます。

HTML できる限り標準の HTML タグを使用し、グリッド、ツリー、カレンダーなどの複雑なコントロールについてはアプレットまたは Flash を使用します。HTML を使用すると、使い慣れた外観を提供できますが、データと表示を簡単に分離することができません。また、Flash のタブ付きナビゲータやアコーディオン形式のコンテナ、カスタマイズされた XML コントロールなどの複雑な機能を使用できません。

Flash 視覚効果の高い洗練された機能を利用できます。HTML では利用できない、タブ付きナビゲータやアコーディオン形式のコンテナなどのコントロールがサポートされています。Flash フォームは、ブラウザに依存しません。Flash 形式を処理する Flash Player は、Windows および Macintosh のすべての主要ブラウザや、Linux の Netscape および Mozilla で動作します。

XML XSLT (Extensible Stylesheet Language Transformation) スキンを指定することで、XML をスタイル指定付きの HTML 出力に変換できます。Adobe ColdFusion には、使用可能なスキンがいくつか用意されています。また、独自のカスタムスキンを作成してカスタムコントロールをサポートすることもできます。

cfform タグとそのサブタグには、入力データの検証機能も用意されています。たとえば、ブラウザまたはサーバーで検証を行うことができます。データ型を確認したり、データ入力をマスクしたりすることが可能です。

cfform では、サブタグを使用してさまざまなダイナミック機能を追加できます。サブタグの中には、対応する HTML が存在しないものもあります。対応する HTML が存在するタグでも、データソースからダイナミックにコントロールを設定する機能が直接サポートされています。また、cfform タグの preservedata 属性を使用すれば、ユーザーがフォームを送信してフォームが再表示されても、フォームのユーザー入力を保持できます。

ここでは、cfform タグの機能について説明するとともに、対応する HTML が存在しない cfform の子タグの使用方法について説明します。cfform フォームのその他の機能については、次のセクションを参照してください。

- 729 ページの「データの検証」
- 751 ページの「Flash フォームの作成」
- 768 ページの「スキン可能 XML フォームの作成」

cfform コントロール

次の表に、cfform フォームで使用できる ColdFusion コントロールを示します。これらのタグは cfform タグ内でのみ使用できます。特に説明がない限り、これらのコントロールは HTML フォーム、Flash フォーム、および XML スキン可能フォームでサポートされています。

コントロール	説明	詳細情報
cfapplet	カスタム Java アプレットをフォームに埋め込みます。Flash フォームではサポートされていません。	727 ページの「Java アプレットの埋め込み」
cfcalendar	インタラクティブな Flash カレンダーを表示します。HTML フォームと Flash フォームでは使用できますが、XML スキン可能フォームでは無視されます。ユーザーは、このカレンダーで日付を選択して、フォーム変数として送信することができます。	『CFML リファレンス』の cfcalendar タグ
cfform	コンテナコントロールを作成します。この中に複数のフォームコントロールを格納して、編成や形式設定を行います。Flash フォームおよび XML スキン可能フォームの cfform タグの本文で使用します。HTML フォームでは無視されます。	751 ページの「Flash フォームの作成」、768 ページの「スキン可能 XML フォームの作成」
cfformitem	Flash フォームに水平線、垂直線、形式設定されたテキスト、または形式設定されていないテキストを挿入します。Flash フォームおよび XML フォームの cfform または cfformgroup タグの本文で使用されます。HTML フォームでは無視されます。	751 ページの「Flash フォームの作成」、768 ページの「スキン可能 XML フォームの作成」
cfgrid	Java アプレットまたは Flash のデータグリッドを作成します。データの設定は、クエリーから行うか、各セルの内容を個別に定義します。また、グリッドを使用して、データソースのレコードの挿入、更新、削除も行えます。	719 ページの「cfgrid によるデータグリッドの作成」
cfinput	HTML の input タグに似ていますが、入力検証機能が追加されています。	693 ページの「基本フォームの作成」
cfselect	選択ボックスを表示します。HTML の select タグに似ていますが、入力検証機能が追加されています。	718 ページの「ドロップダウンリストボックスの構築」

コントロール	説明	詳細情報
cfslider	スライダを移動してデータを入力できる Java アプレットベースのコントロールを作成します。Flash フォームではサポートされていません。	719 ページの「スライダバーコントロールの構築」
cftextarea	テキスト入力領域を表示します。HTML の textarea タグに似ていますが、入力検証機能が追加されています。	『CFML リファレンス』の cftextarea タグ
cftree	Java アプレットまたは Flash の階層ツリー形式のコントロールを作成します。さまざまな要素のグラフィカルイメージを使用できます。ツリーのデータおよび属性を表す ColdFusion 構造体を生成することもできます。	712 ページの「cftree タグによるツリーコントロールの構築」

preservedata 属性による入力データの保持

cfform の preservedata 属性を設定すると、ユーザーがフォームを送信した後も、フォームに入力されたユーザーデータが引き続き表示されます。データは、cfinput、cfslider、cftextinput、および cftree コントロールや、クエリーによって設定された cfselect コントロールで保持されます。デフォルト値が設定されているコントロールでも、ユーザーの入力したデータが保持されます。

フォームとフォームのアクションコードが同じページに含まれている場合 (フォームがそれ自体にデータを送信する場合) は、フォームにデータを保持できます。また、アクションページにフォームのコピー (同じ名前のコントロール) が存在する場合もデータを保持できます。アクションページのフォームは、元のフォームと同一である必要はありません。フォーム要素の数が元のフォームと異なっていてもかまいません。その場合は、両方のページで同じ名前を持つフォーム要素だけが、データを保持します。

注意：データの保持は、アクションページの preservedata 設定で制御します。

たとえば、次のフォームを "preserve.cfm" という名前で保存すると、フォームの送信後も入力したテキストが引き続き表示されます。

```
<cfform action="preserve.cfm" preservedata="Yes">
    <p>Please enter your name:
    <cfinput type="Text" name="UserName" required="Yes"><p>
    <input type="Submit" name=""> <input type="RESET">
</cfform>
```

preservedata 属性の使用に関する注意事項

preservedata 属性を使用するときは、次のガイドラインに従ってください。

- cftree タグでは、前に選択されていた要素までツリーが展開されます。ツリーが正しく展開されるようにするには、completePath 属性を True に設定します。
- preservedata 属性は、cfgrid タグに影響を与えません。グリッドの情報をクエリーで設定している場合は、グリッドを再表示する前に、cfgridupdate タグなどを使用してデータソースのデータを更新する必要があります。グリッドを表示すれば更新されたデータベース情報が表示されます。

ブラウザに関する注意事項

アプレットベースの cfgrid、cfslider、および cftree フォームでは、その内容を表示するために JavaScript および Java が使用されます。さまざまなブラウザで同じ表示を行うために、これらのアプレットでは Java プラグインが使用されます。したがって、ブラウザの Java サポートレベルには依存しません。

ブラウザのプラグインは、ColdFusion によって必要に応じてダウンロードされ、インストールされます。一部のブラウザでは、プラグインのインストールを確認するためのダイアログボックスが 1 つ表示されます。ブラウザによっては (古いバージョンの Netscape など)、簡単なオプションウィンドウをいくつか設定する必要があります。

このコントロールは、JavaScript を使用して ColdFusion にデータを返すので、ブラウザで JavaScript が無効に設定されていると、これらのコントロールが含まれているフォームは正しく実行されません。この場合、コントロールは表示されますが、データの送信と検証が行われず、JavaScript エラーが発生することがあります。

Java は、直接ブラウザで処理されるのではなくプラグインで処理されるので、ブラウザで Java の実行を無効にしても、コントロールの処理には影響を与えません。何らかの理由で、必要なコントロールをブラウザに表示できない場合は、コントロールの代わりに " サポートされていない " というメッセージが表示されます。

cfform タグの notsupported 属性を使用すると、代替エラーメッセージを指定できます。

Flash 版の cfgrid および cftree コントロールを使用すれば、ブラウザの Java および JavaScript に関する問題を回避することができます。これらのコントロールは、Windows、Mac OS X、および Linux で動作し、Java のサポートに依存しません。Flash 版の cfslider コントロールはありません。また、アプレット版の cfcalendar コントロールはありません。

cftree タグによるツリーコントロールの構築

cftree タグを使用すると、データソースクエリーで取得した階層状の情報を、コンパクトな折り畳み式のツリーのフォームに表示できます。cftree タグでツリーコントロールを作成するには、cftreeitem タグを使用してコントロールを構築します。

次に示す 3 つの形式のツリーを作成することができます。

アプレット Java アプレットを作成します。クライアントでダウンロードする必要があります。アプレットのダウンロードには時間がかかるので、HTML のフォーム要素を使用する場合と比べて、同じ情報を取得するにも若干時間がかかることがあります。また、cftree タグを正しく動作させるには、Java 対応ブラウザを使用する必要があります。

Flash HTML フォームまたは Flash フォームに含めることができる Flash コントロールを生成します。Flash フォームの詳細については、751 ページの「Flash フォームの作成」を参照してください。

オブジェクト ツリーデータと cftree および cftreeitem の多くの属性を表現する、階層状の ColdFusion 構造体を作成します。

サポートされている機能および属性は、形式によって異なります。ここでは、3 つのすべての形式で使用できる一般的な技法について説明するとともに、一部の形式でのみ使用できる技法についても説明します。ここでは、すべての例でアプレット形式を使用します。いくつかの属性はアプレット固有のものです。各形式でサポートされている機能および属性の詳細については、『CFML リファレンス』の cftree を参照してください。

クエリーからのツリーコントロールの作成

1 次の内容の ColdFusion ページを作成します。

```
<cfquery name="engquery" datasource="cfdocexamples">
    SELECT FirstName || ' ' || LastName AS FullName
    FROM Employee
</cfquery>
<cfform name="form1" action="submit.cfm">
<cftree name="tree1"
    required="Yes"
    hscroll="No">
    <cftreeitem value="FullName"
        query="engquery"
        queryasroot="Yes"
        img="folder,document">
</cftree>
</cfform>
```

2 このページに "tree1.cfm" という名前を付けて保存し、ブラウザで表示します。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
<cftree name="tree1"	ツリーを作成し、tree1 という名前を付けます。
required="Yes"	ユーザーがツリーの項目を選択する必要があることを指定します。
hscroll="No"	横方向にスクロールできないようにします。
<cftreeitem value="FullName" query="engquery"	engquery というクエリーの結果を使用してツリー項目を作成します。このタグではクエリーを使用するので、クエリーエントリごとにツリー項目が 1 つ作成されます。
queryasroot="Yes"	クエリー名をツリーコントロールのルートレベルとして指定します。
img="folder,document"	ツリー構造で、ColdFusion に組み込まれているフォルダとドキュメントのイメージを使用します。 cfquery タグのデータを cftree タグで使用する場合、ツリーの各レベルのイメージまたはファイル名をカンマ区切りのリストとして指定できます。

クエリー出力のグループ化

cftree コントロールに表示する各従業員を部門別に分類するには、cftreeitem の value 属性に、列名をカンマで区切って指定します。

ソートしたクエリー結果に基づくツリーの構成

1 次の内容を持つ "tree2.cfm" という名前の ColdFusion ページを作成します。

```
<!--- CFQUERY with an ORDER BY clause. --->
<cfquery name="deptquery" datasource="cfdocexamples">
    SELECT Dept_ID, FirstName || ' ' || LastName
    AS FullName
    FROM Employee
    ORDER BY Dept_ID
</cfquery>

<!--- Build the tree control. --->
<cfform name="form1" action="submit.cfm">

<cftree name="tree1"
    hscroll="No"
    border="Yes"
    height="350"
    required="Yes">

<cftreeitem value="Dept_ID, FullName"
    query="deptquery"
    queryasroot="Dept_ID"
    img="computer,folder,document"
    imgopen="computer,folder"
    expand="yes">

</cftree>
<br>
<br><input type="Submit" value="Submit">
</cfform>
```

2 このページを保存してブラウザで表示します。

コードの説明

次の表で、前述のコード (太字部分) およびその機能について説明します。

コード	説明
ORDER BY Dept_ID	部門別にクエリー結果をソートします。
<cftreeitem value="Dept_ID,FullName"	ツリーに部門 ID を挿入し、各部門の下にその部門の各従業員の名前を挿入します。
queryasroot="Dept_ID"	ルートに "Dept_ID" というラベルを付けます。
img="computer,folder,document" imgopen="computer,folder"	ColdFusion に用意されているコンピュータイメージをルートレベルに、フォルダイメージを部門 ID に、ドキュメントイメージを名前に使用します。レベルが展開されている (開いている) ときも折り畳まれているときも同じイメージを使用します。従業員名は開けないので、imgopen 属性には 2 つの項目のみが含まれます。

cftreeitem のカンマ区切りの value、img、および imgopen 属性は、ツリーのレベル構造に対応しています。アプレット形式で img 属性を省略した場合は、ツリー内のすべてのレベルに対してフォルダイメージが使用されます。imgopen 属性を省略した場合は、ツリー内の展開されたすべてのレベルに対してフォルダイメージが使用されます。Flash 形式の場合、img および imgopen 属性は無視され、子を持つレベルにはフォルダが、子が存在しないノードにはドキュメントが常に使用されます。

cftree フォーム変数

cftree タグで required 属性を Yes に設定すると、ツリーコントロールで項目を選択する必要があることを指定できます。cfform の action 属性で指定したアプリケーションページには、required 属性の設定に関係なく、次の 2 つのフォーム変数が渡されます。

- Form.**treename**.path: ユーザーが選択した絶対パスを、次の形式で返します。**[root]¥node1¥node2¥node_n¥value**
- Form.**treename**.node: ユーザーが選択したノードを返します。

パスのルート部分を含めるには、cftree タグの completepath 属性を Yes に設定します。そうでない場合、パスの値は最初のノードから始まります。queryasroot タグを使用してツリー項目のルート名を指定している場合は、その値がルートとして返されます。ルート名を指定していない場合は、クエリー名がルートとして返されます。クエリー名がない場合は、ツリー名がルートとして返されます。

前の例で、ツリーの "John Allen" という名前を選択すると、次のフォーム変数が返されます。

```
Form.tree1.path = 3\John Allen
Form.tree1.node = John Allen
```

cftree タグで completePath="Yes" と指定していないので、deptquery のルートはパスに含まれません。cftree の delimiter 属性で、パスフォーム変数の各要素を区切る文字を指定できます。デフォルトは円記号 (¥) です。

入力の検証

cftree タグに validate 属性はありませんが、required 属性を使用してツリーコントロールの項目を強制的に選択させることができます。また、検証を実行する JavaScript コードを onValidate 属性で指定できます。

ツリーコントロールの構築

cftree タグを使用すると、複雑なツリーコントロールを構築できます。複数の cftreeitem エントリの関係を指定する方法を知っておくと、複雑な cftree 構文を扱うときに役立ちます。

1 レベルのツリーコントロールの作成

次の例は、1 つのルートと複数の項目から構成されています。

```
<cfquery name="deptquery" datasource="cfdocexamples">
    SELECT Dept_ID, FirstName || ' ' || LastName
    AS FullName
    FROM Employee
    ORDER BY Dept_ID
</cfquery>

<cfform name="form1" action="submit.cfm">
    <cftree name="tree1">
        <cftreeitem value="FullName"
        query="deptquery"
        queryasroot="Department">
        img="folder,document">
    </cftree>
<br>
<cfinput type="submit" value="Submit">
</cfform>
```

複数レベルのツリーコントロールの作成

複数レベルのツリー構造を作成するには、cftree コントロールにツリー項目を挿入するときに、その親を指定します。cftreeitem タグの parent 属性を使用すれば、ツリー項目の関係を示すことができます。

この例では、最上位の Divisions を除くすべての cftreeitem タグで親を指定しています。たとえば、cftreeitem タグでは Divisions を親として指定しています。

次のコードでは、クエリーからではなく直接ツリーを作成します。

```
<cfform name="form2" action="cfform_submit.cfm">
<cftree name="tree1" hscroll="No" vscroll="No"
    border="No">
    <cftreeitem value="Divisions">
    <cftreeitem value="Development"
        parent="Divisions" img="folder">
    <cftreeitem value="Product One"
        parent="Development" img="document">
    <cftreeitem value="Product Two"
        parent="Development">
    <cftreeitem value="GUI"
        parent="Product Two" img="document">
    <cftreeitem value="Kernel"
        parent="Product Two" img="document">
    <cftreeitem value="Product Three"
        parent="Development" img="document">
    <cftreeitem value="QA"
        parent="Divisions" img="folder">
    <cftreeitem value="Product One"
        parent="QA" img="document">
    <cftreeitem value="Product Two"
        parent="QA" img="document">
    <cftreeitem value="Product Three"
        parent="QA" img="document">
    <cftreeitem value="Support"
        parent="Divisions" img="fixed">
    <cftreeitem value="Product Two"
        parent="Support" img="document">
    <cftreeitem value="Sales"
        parent="Divisions" img="computer">
    <cftreeitem value="Marketing"
        parent="Divisions" img="remote">
    <cftreeitem value="Finance"
        parent="Divisions" img="element">
</cftree>
</cfform>
```

cftree タグでのイメージ名

注意：ここでの説明は、アプレット形式のツリーに適用されます。Flash では、ツリーのアイコンを制御することはできません。Flash では、開いたフォルダ、閉じたフォルダ、ドキュメントのアイコンが使用されます。オブジェクト形式では、イメージ情報は、オブジェクト構造体のフィールドに保持されます。

ツリーに表示されるデフォルトのイメージはフォルダです。cftreeitem タグの img 属性を使用すれば、別のイメージを指定できます。

img 属性を使用すると、開いていないツリー項目の横に、指定のイメージが表示されます。imgopen 属性を使用すると、開いている (展開されている) ツリー項目の横に、指定のイメージが表示されます。これらの属性には、ColdFusion のビルトインイメージ名、イメージファイルへのファイルパス、または http://localhost/Myapp/Images/Level3.gif などのイメージの URL を指定できます。Flash 形式ではカスタムイメージを使用できません。通常、カスタムイメージの高さは 20 ピクセル未満にします。

cfquery タグのデータから cftree コントロールを構築する場合は、cftreeitem タグの img 属性で、ツリーの各レベルのイメージ名またはファイル名をカンマ区切りのリストとして指定できます。

ColdFusion のビルトインイメージ名は次のとおりです。

- computer
- document
- element

- folder
- floppy
- fixed
- remote

注意：アプレット形式では、cftree タグの lookAndFeel 属性を使用して、Windows、Motif、または Metal の外観を指定することができます。

cftree タグへの URL の埋め込み

cftreeitem タグで href 属性を使用すると、ツリー項目をリンクとして指定できます。cftree コントロールでこの機能を使用するには、cftreeitem タグの href 属性でリンク先を定義します。リンクの URL は、次の例のように、相対 URL または絶対 URL のいずれかです。

cftree コントロールへのリンクの埋め込み

1 次の内容を持つ "tree3.cfm" という名前の ColdFusion ページを作成します。

```
<cfform action="submit.cfm">
<cftree name="oak"
    highlighthref="Yes"
    height="100"
    width="200"
    hspace="100"
    vspace="6"
    hscroll="No"
    vscroll="No"
    border="No">
    <cftreeitem value="Important Links">
    <cftreeitem value="Adobe Home"
        parent="Important Links"
        img="document"
        href="http://www.adobe.com">
    <cftreeitem value="ColdFusion Developer Center"
        parent="Important Links"
        img="document"
        href="http://www.adobe.com/devnet/coldfusion/">
</cftree>
</cfform>
```

2 このページを保存してブラウザで表示します。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
href="http://www.adobe.com">	ツリーのノードにリンクを設定します。
href="http://www.adobe.com/devnet/mx/coldfusion/">	ツリーのノードにリンクを設定します。 この例には該当しませんが、クエリーからツリー項目を構築している場合は、href 属性でクエリー内の列の名前を参照できます。

URL でのツリー項目の指定

URL にリンクしているツリー項目をクリックすると、選択された値を識別する cftreeItemKey 変数が、次のように URL に追加されます。

```
http://myserver.com?CFTREEITEMKEY=selected_item_value_attribute
```

value 属性にスペースが含まれている場合、スペースはプラス記号 (+) に置換されます。

選択されたツリー項目が URL の一部として自動的に渡されるので、選択に基づいて詳細情報を表示する基本的な " ドリルダウン " アプリケーションを簡単に実装できます。たとえば、URL で他の ColdFusion ページを指定した場合は、変数 URL.CFTREEITEMKEY を使用して選択された値にアクセスできます。

この動作を無効にするには、cftree タグの appendkey 属性を no に設定します。

ドロップダウンリストボックスの構築

cfform タグ内で cfselect タグを使用して作成できるドロップダウンリストボックスは、HTML の select タグに似ていますが、ユーザー入力やエラー処理をより詳細に制御できます。また、最も重要な機能として、クエリーから選択リストを自動的に構築することができます。

ドロップダウンリストボックスを構築するには、クエリーを使用するか、option タグで作成したオプション要素のリストを使用します。cfselect タグで使用する option タグのシンタックスは、HTML の option タグのシンタックスと同じです。

クエリーのデータを使用して cfselect タグを構築するには、クエリーの名前と、表示するリスト要素が含まれているクエリー列の名前を指定します。

クエリーデータからの cfselect ドロップダウンリストボックスの構築

1 次の内容の ColdFusion ページを作成します。

```
<cfquery name="getNames"
    datasource="cfdocexamples">
    SELECT * FROM Employee
</cfquery>

<cfform name="Form1" action="submit.cfm">

    <cfselect name="employees"
        query="getNames"
        value="Emp_ID"
        display="FirstName"
        required="Yes"
        multiple="Yes"
        size="8">
    </cfselect>

    <br><input type="Submit" value="Submit">
</cfform>
```

2 このファイルに "selectbox.cfm" という名前を付けて保存し、ブラウザで表示します。

このタグには multiple 属性があるので、ユーザーはリストボックスで複数のエントリを選択できます。また、value タグで Emp_ID (Employee テーブルのプライマリキー) が指定されているので、cfform の action 属性で指定したアプリケーションページに渡される Form.Employee 変数には、名前ではなく従業員 ID が使用されます。

クエリーを使用して、いずれかのクエリー列でグループ化された、2 つのレベルを持つ階層リストを作成することができます。この例については、『CFML リファレンス』の cfselect を参照してください。

スライダバーコントロールの構築

cfform タグで cfslider コントロールを使用して、スライダコントロールを作成できます。その際には、ラベルのテキスト、ラベルのフォント名、サイズ、ボールド、イタリック、色、スライダの範囲、位置、動作などのさまざまな特性を定義できます。スライダバーには、視覚効果が高く、有効な値しか入力できないという利点があります。cfslider タグは、Flash フォームではサポートされていません。

スライダコントロールの作成

1 次の内容の ColdFusion ページを作成します。

```
<cfform name="Form1" action="submit.cfm">
    <cfslider name="myslider"
        bgcolor="cyan"
        bold="Yes"
        range="0,1000"
        scale="100"
        value="600"
        fontsize="14"
        label="Slider %value%"
        height="60"
        width="400">
</cfform>
```

2 このファイルに "slider.cfm" という名前を付けて保存し、ブラウザで表示します。

アクションページでスライダの値を取得するには、変数 Form.< スライダ名 > を使用します。ここでは、Form.myslider を使用しています。

cfgrid によるデータグリッドの作成

cfgrid タグは、スプレッドシートテーブルに似た cfform のグリッドコントロールを作成します。グリッドのデータは、cfquery タグまたは他のデータソースを使用して設定できます。他の cfform タグと同様に、cfgrid タグにはさまざまな形式設定オプションや、JavaScript 検証スクリプトを使用してユーザーの選択を検証するオプションが用意されています。

また、cfgrid タグでは次のタスクも実行できます。

- グリッド内データの英数字順によるソート
- データの更新、挿入、および削除
- グリッド内へのイメージの埋め込み

注意：Flash 形式のグリッドでは、アプレット形式のグリッドで利用可能な機能の一部がサポートされています。各形式でサポートされている機能の詳細については、『CFML リファレンス』の cfgrid タグを参照してください。

グリッドエントリを昇順にソートするには、列ヘッダをダブルクリックします。再びダブルクリックすると、降順にソートできます。アプレット形式では、グリッドコントロールにソートボタンを追加することもできます。

グリッドデータを選択してフォームを送信すると、cfform の action 属性で指定したアプリケーションページに、選択情報がフォーム変数として渡されます。

cftree タグで cftreeitem タグを使用するのと同様に、cfgrid タグでは cfgridcolumn および cfgridrow タグを使用します。行や列では、名前、データ型、選択オプションなどに加えて、さまざまな形式設定オプションを定義できます。グリッド内の各列を定義したり、クエリー列とグリッド列を関連付けたりするには、cfgridcolumn タグを使用します。

クエリーを使用せずにグリッドの行データを定義する場合は、cfgridrow タグを使用します。query 属性が cfgrid タグで指定されている場合、cfgridrow タグは無視されます。

cfgrid タグには、グリッドの動作および外観を制御する多数の属性があります。ここでは、最も重要な属性のみ説明します。これらの属性の詳細については、『CFML リファレンス』の cfgrid タグを参照してください。

データグリッドの操作およびデータの入力

次の図に、cfgrid タグを使用して作成したアプレット形式のグリッドの例を示します。

次の表に、操作のヒントを示します。

アクション	手順
グリッドの行のソート	列を昇順にソートするには、列ヘッダをダブルクリックします。列を降順にソートするには、再びダブルクリックします。
列の再配置	任意の列見出しをクリックしたまま、新しい位置に列をドラッグします。
編集可能なグリッド領域	編集可能なセルをクリックすると、黄色のボックスで囲まれます。
編集できないグリッド領域	編集できないセル、行、または列をクリックすると、背景色が変わります。デフォルトの色は、サーモンピンクです。
グリッドセルの編集	セルをダブルクリックします。データの入力後は、Enter キーを押します。
行の削除	行の任意のセルをクリックし、[削除] ボタンをクリックします。Flash 形式のグリッドでは利用できません。
行の挿入	[挿入] ボタンをクリックします。空の行がグリッドの下部に表示されます。各セルに値を入力するには、セルをダブルクリックし、値を入力し、Enter キーを押します。Flash 形式のグリッドでは利用できません。

クエリーからのグリッドデータの挿入

1 次の内容を持つ "grid1.cfm" という名前の ColdFusion ページを作成します。

```
<cfquery name="empdata" datasource="cfdocexamples">
    SELECT * FROM Employee
</cfquery>

<cfform name="Form1" action="submit.cfm" >
    <cfgrid name="employee_grid" query="empdata"
            selectmode="single">
        <cfgridcolumn name="Emp_ID">
        <cfgridcolumn name="LastName">
        <cfgridcolumn name="Dept_ID">
    </cfgrid>
    <br>
    <cfinput name="submitit" type="Submit" value="Submit">
</cfform>
```

注意：cfgridcolumn の display="No" 属性を使用すれば、グリッドには含めるがエンドユーザーには表示しない列を非表示にできます。通常、この属性は、cfgrid タグで返される結果に、テーブルのプライマリキーなどの列を保持するために使用します。

2 このファイルを保存してブラウザで表示します。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
<cfgrid name="employee_grid" query="empdata"	employee_grid という名前のグリッドを作成し、クエリー empdata の結果を挿入します。 cfgrid タグに query 属性を定義し、cfgridcolumn で対応する属性を定義しなかった場合、グリッドにはクエリーのすべての列が含まれます。
selectmode="single">	ユーザーが 1 つのセルのみを選択できるようにします。編集は許可しません。他のモードは、row、column、および edit です。
<cfgridcolumn name="Emp_ID">	クエリー結果の Emp_ID 列の内容を、グリッドの最初の列に配置します。
<cfgridcolumn name="LastName">	クエリー結果の LastName 列の内容を、グリッドの 2 番目の列に配置します。
<cfgridcolumn name="Dept_ID">	クエリー結果の Dept_ID 列の内容を、グリッドの 3 番目の列に配置します。

編集可能なグリッドの作成

データを編集可能なグリッドを構築できます。各セルデータの編集だけでなく、行の挿入、更新、削除も行えます。グリッドの編集を有効にするには、cfgrid タグで selectmode="edit" を指定します。

グリッド行の追加や削除を許可するには、cfgrid タグの insert または delete 属性を Yes に設定します。insert 属性や delete 属性を Yes に設定すると、挿入ボタンや削除ボタンがグリッドに表示されます。

グリッドを使用して ColdFusion データソースを変更するには、2 つの方法があります。

- cfgrid のフォーム変数を別のページに渡し、そのページで cfquery を実行して、cfgrid タグから返されたフォーム値に基づいてデータソースレコードを更新します。
- cfgridupdate タグを含むページにグリッドの編集内容を渡します。このタグによって、自動的にフォーム変数値が抽出され、データソースにデータが直接渡されます。

cfquery タグを使用すると、データソースとのやり取りを完全に制御できます。cfgridupdate タグは、cfquery タグのような詳細な制御を必要としない場合に便利です。

セルの内容の制御

次の方法で、cfgrid セルに入力可能なデータを制御できます。

- デフォルトでは、セルは編集できません。セルの内容を編集するには、cfgrid 属性の selectmode="edit" を使用します。
- cfgridcolumn の type 属性を使用して、ソート順序を制御したり、フィールドをチェックボックスにしたり、イメージを表示したりできます。
- cfgridcolumn の values 属性を使用して、選択可能な値のドロップダウンリストを指定できます。valuesDisplay 属性を使用すると、データベースに入力する実際の値とは別に、ドロップダウンリストに表示する項目のリストを指定できます。valuesDelimiter 属性を使用すると、values や valuesDisplay リストで値を区切るために使用する文字を指定できます。
- cfgrid タグに validate 属性はありませんが、onValidate 属性を使用して、検証を実行する JavaScript 関数を指定できます。

セル内容の制御の詳細については、『CFML リファレンス』で cfgridcolumn タグの属性に関する説明を参照してください。

編集データの取得

ユーザーがグリッドで行の挿入や削除を行ったり、行内のセルを変更したりしてグリッドを送信すると、次の配列がフォーム変数として作成されます。

配列名	説明
gridname.colname	挿入、削除、または更新されたセルの新しい値が格納されています。削除されたセルのエントリには空の文字列が含まれます。
gridname.Original.colname	挿入、削除、または更新されたセルの元の値が格納されています。
gridname.RowStatus.Action	グリッド行で実行された変更のタイプが格納されています。削除は D、挿入は I、更新は U です。

注意：これらの名前に含まれているピリオドは、構造体の区切り文字では**ありません**。このピリオドは、配列名のテキストの一部です。

グリッド内の各列に対して、**gridname.colname** 配列と **gridname.Original.colname** 配列が 1 つずつ作成されます。グリッド内で挿入、削除、または変更された各行に対して、各配列に 1 つの行が作成されます。

たとえば、2 つの表示可能な列 col1 および col2 と 1 つの非表示列 col3 から構成される mygrid という cfgrid タグを更新すると、次のような配列が作成されます。

```
Form.mygrid.col1
Form.mygrid.col2
Form.mygrid.col3
Form.mygrid.original.col1
Form.mygrid.original.col2
Form.mygrid.original.col3
Form.mygrid.RowStatus.Action
```

配列インデックスの値は、追加、削除、または変更された行数だけ存在しますが、グリッド行番号を示しているわけではありません。特定の変更が行われた各行は、全配列で同じインデックスを持ちます。変更されていない行は、配列内にエントリを持ちません。

col2 の 1 つのセルを変更した場合は、次の配列要素に、編集操作、編集後のセル値、および元のセル値が含まれます。

```
Form.mygrid.RowStatus.Action[1]
Form.mygrid.col2[1]
Form.mygrid.original.col2[1]
```

ある行の col1 と col3、および別の行の col2 のセル値を変更した場合は、次の配列エントリに元の値と変更後の値が含まれます。

```
Form.mygrid.RowStatus.Action[1]
Form.mygrid.col1[1]
Form.mygrid.original.col1[1]
Form.mygrid.col3[1]
Form.mygrid.original.col3[1]
Form.mygrid.RowStatus.Action[2]
Form.mygrid.col2[2]
Form.mygrid.original.col2[2]
```

配列の残りのセル (たとえば Form.mygrid.col2[1] と Form.mygrid.original.col2[1]) には、変更されていない元の値が含まれます。

例 : グリッド内のデータの編集

次の例では、編集可能グリッドを作成します。この例では、コードを簡潔にするために、Employee テーブルで処理するフィールドを 3 つに限定しています。実際のコードでは、少なくとも 7 つのテーブルフィールドをすべて含めることになります。また、Emp_ID 列の内容を非表示にしたり、部門 ID ではなく (Departmt テーブルから取得した) 部門名を表示するとユーザーにとってわかりやすくなります。

編集可能グリッドの作成

1 次の内容の ColdFusion ページを作成します。

```
<cfquery name="empdata" datasource="cfdocexamples">
    SELECT * FROM Employee
</cfquery>

<cfform name="GridForm"
    action="handle_grid.cfm">

    <cfgrid name="employee_grid"
        height=425
        width=300
        vspace=10
        selectmode="edit"
        query="empdata"
        insert="Yes"
        delete="Yes">

        <cfgridcolumn name="Emp_ID"
            header="Emp ID"
            width=50
            headeralign="center"
            headerbold="Yes"
            select="No">

        <cfgridcolumn name="LastName"
            header="Last Name"
            width=100
            headeralign="center"
            headerbold="Yes">

        <cfgridcolumn name="Dept_ID"
            header="Dept"
            width=35
            headeralign="center"
            headerbold="Yes">

    </cfgrid>
    <br>
    <cfinput name="submitit" type="Submit" value="Submit">
</cfform>
```

2 このファイルに "grid2.cfm" という名前を付けて保存します。

3 この結果をブラウザで表示します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明

`<cfgrid name="employee_grid"`
`height=425`
`width=300`
`vspace=10`
`selectmode="edit"`
`query="empdata"`
`insert="Yes"`
`delete="Yes">`
	empdata クエリーのデータを cfgrid コントロールに挿入します。グリッドセルを選択すると編集できるようになります。行を挿入したり、削除したりすることができます。グリッドは 425 × 300 ピクセルで、上下に 10 ピクセルの間隔があります。

`<cfgridcolumn name="Emp_ID"`
`header="Emp ID"`
`width=50`
`headeralign="center"`
`headerbold="Yes"`
`select="No">`
	データソースの Emp_ID 列のデータを格納するために、幅 50 ピクセルの列を作成します。Emp ID というヘッダを中央揃えにし、太字にします。
	この列のフィールドは、選択して編集できないようにします。このフィールドはテーブルのプライマリキーなので、既存レコードのこのフィールドは、ユーザーが変更できないようにする必要があります。このフィールドは DBMS で自動インクリメント値として生成されるのが普通です。

`<cfgridcolumn name="LastName"`
`header="Last Name"`
`width=100`
`headeralign="center"`
`headerbold="Yes">`
	データソースの LastName 列のデータを格納するために、幅 100 ピクセルの列を作成します。Last Name というヘッダを中央揃えにし、太字にします。

`<cfgridcolumn name="Dept_ID"`
`header="Dept"`
`width=35`
`headeralign="center"`
`headerbold="Yes">`
	データソースの Dept_ID 列のデータを格納するために、幅 35 ピクセルの列を作成します。Dept というヘッダを中央揃えにし、太字にします。

cfgridupdate タグによるデータベースの更新

cfgridupdate タグを使用すれば、レコードの挿入や削除などのデータベース更新作業を簡単に行えます。レコードの追加、更新、および削除は同時に行うことができます。このタグは、さまざまなフォーム変数から cfgrid の変更を自動的に収集し、適切な SQL ステートメントを生成してデータソースを更新します。

ほとんどの場合、データベースの更新には cfgridupdate タグを使用します。ただし、cfquery タグのように SQL を詳細に制御することはできません。特に、cfgridupdate タグには次の特性があります。

- 更新できるテーブルは 1 つのみです。
- まず行を削除し、次に行を挿入し、既存の行を変更します。変更の順序を変えることはできません。
- エラーが発生すると、更新は停止します。変更が途中まで行われた可能性がありますが、それに関する情報は返されません。

cfgridupdate タグによるデータソースの更新

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Update grid values</title>
</head>
<body>

<h3>Updating grid using cfgridupdate tag.</h3>

<cfgridupdate grid="employee_grid"
    datasource="cfdocexamples"
    tablename="Employee">

Click <a href="grid2.cfm">here</a> to display updated grid.

    </body>
</html>
```

2 このファイルに "handle_grid.cfm" という名前を付けて保存します。

3 "grid2.cfm" ページをブラウザで表示し、グリッドを変更して送信します。

注意：グリッドセルを更新するには、セルの内容を変更してから Enter キーを押します。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
<cfgridupdate grid="employee_grid"	Employee_grid グリッドを使用してデータベースを更新します。
datasource="cfdocexamples"	cfdocexamples データソースを更新します。
tablename="Employee"	Employee テーブルを更新します。

cfquery タグによるデータベースの更新

cfquery タグでも、cfgrid の変更に基づいてデータベースを更新できます。このタグを使用すると、更新方法を詳細に制御できるとともに、エラー処理も行えます。

cfquery タグによるデータソースの更新

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Catch submitted grid values</title>
</head>
<body>

<h3>Grid values for Form.employee_grid row updates</h3>

<cfif isdefined("Form.employee_grid.rowstatus.action")>

    <cfloop index = "counter" from = "1" to =
        #arraylen(Form.employee_grid.rowstatus.action)#>

        <cfoutput>
            The row action for #counter# is:
            #Form.employee_grid.rowstatus.action[counter]#
            <br>
        </cfoutput>

        <cfif Form.employee_grid.rowstatus.action[counter] is "D">
            <cfquery name="DeleteExistingEmployee"
                datasource="cfdocexamples">
                DELETE FROM Employee
                WHERE Emp_ID=<cfqueryparam
                            value="#Form.employee_grid.original.Emp_ID[counter]#"
                            CFSQLType="CF_SQL_INTEGER" >
            </cfquery>

        <cfelseif Form.employee_grid.rowstatus.action[counter] is "U">
            <cfquery name="UpdateExistingEmployee"
                datasource="cfdocexamples">
                UPDATE Employee
                SET
                    LastName=<cfqueryparam
                            value="#Form.employee_grid.LastName[counter]#"
                            CFSQLType="CF_SQL_VARCHAR" >,
                    Dept_ID=<cfqueryparam
```

```
                            value="#Form.employee_grid.Dept_ID[counter]#"
                            CFSQLType="CF_SQL_INTEGER" >
                    WHERE Emp_ID=<cfqueryparam
                        value="#Form.employee_grid.original.Emp_ID[counter]#"
                        CFSQLType="CF_SQL_INTEGER">
                </cfquery>

            <cfelseif Form.employee_grid.rowstatus.action[counter] is "I">
                <cfquery name="InsertNewEmployee"
                    datasource="cfdocexamples">
                    INSERT into Employee (LastName, Dept_ID)
                    VALUES (<cfqueryparam
                        value="#Form.employee_grid.LastName[counter]#"
                            CFSQLType="CF_SQL_VARCHAR" >,
                        <cfqueryparam value="#Form.employee_grid.Dept_ID[counter]#"
                            CFSQLType="CF_SQL_INTEGER" >)
                </cfquery>

            </cfif>
        </cfloop>
    </cfif>

    Click <a href="grid2.cfm">here</a> to display updated grid.

    </body>
    </html>
```

2 既存の "handle_grid.cfm" ファイルの名前を "handle_grid2.cfm" に変更して保存し、このファイルに "handle_grid.cfm" という名前を付けて保存します。

3 "grid2.cfm" ページをブラウザで表示し、グリッドを変更して送信します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明

`<cfif isdefined("Form.employee_grid.rowstatus.action")>`
`<cfloop index = "counter" from = "1" to =`
`#arraylen(Form.employee_grid.rowstatus.action)#>`
	編集タイプの配列が存在する場合は、テーブルを変更します。そうでない場合は、何も行いません。変更する行ごとに、残りのコードをループします。counter 変数は、変更する行の情報が含まれている配列で使用する、共通のインデックスです。

`<cfoutput>` `The row action for #counter# is:` `#Form.employee_grid.rowstatus.action[counter]#` ` ` `</cfoutput>`	この行のアクションコードを表示します。更新は U、挿入は I、削除は D です。

`<cfif Form.employee_grid.rowstatus.action[counter] is "D">`
`<cfquery name="DeleteExistingEmployee"`
`datasource="cfdocexamples">`
`DELETE FROM Employee`
`WHERE Emp_ID=<cfqueryparam`
`value="#Form.employee_grid.original.Emp_ID[counter]#"`
`CFSQLType="CF_SQL_INTEGER" >`
`</cfquery>`
	アクションが行の削除である場合は、削除する行の Emp_ID (プライマリキー) を指定する SQL の DELETE クエリーを生成します。

コード	説明
<cfelseif Form.employee_grid.rowstatus.action[counter] is "U"> <cfquery name="UpdateExistingEmployee" datasource="cfdocexamples"> UPDATE Employee SET LastName=<cfqueryparam value="#Form.employee_grid.LastName[counter]#" CFSQLType="CF_SQL_VARCHAR" >, Dept_ID=<cfqueryparam value="#Form.employee_grid.Dept_ID[counter]#" CFSQLType="CF_SQL_INTEGER" > WHERE Emp_ID=<cfqueryparam value="#Form.employee_grid.original.Emp_ID[counter]#" CFSQLType="CF_SQL_INTEGER"> </cfquery>	アクションが行の更新である場合は、SQL の UPDATE クエリーを生成して、Emp_ID プライマリテーブルキーで指定される行の LastName フィールドと Dept_ID フィールドを更新します。
<cfelseif Form.employee_grid.rowstatus.action[counter] is "I"> <cfquery name="InsertNewEmployee" datasource="cfdocexamples"> INSERT into Employee (LastName, Dept_ID) VALUES (<cfqueryparam value="#Form.employee_grid.LastName[counter]#" CFSQLType="CF_SQL_VARCHAR" >, <cfqueryparam value="#Form.employee_grid.Dept_ID[counter]#" CFSQLType="CF_SQL_INTEGER" > </cfquery>	アクションが行の挿入である場合は、SQL の INSERT クエリーを生成して、従業員の姓と部門 ID をグリッド行からデータベースに挿入します。この INSERT ステートメントでは、Emp_ID プライマリキーは DBMS によって自動的にインクリメントされるものと想定しています。UNIX 版の cfdocexamples データベースを使用している場合は、レコードが挿入されても Emp_ID に数値は設定されません。
</cfif> </cfloop> </cfif>	削除、更新、または挿入を選択する cfif タグを閉じます。 各行を変更するループを閉じます。 アクティブコード全体を囲む cfif タグを閉じます。

Java アプレットの埋め込み

cfapplet タグを使用すると、Java アプレットを ColdFusion ページまたは cfform タグに埋め込むことができます。cfapplet タグを使用するには、ColdFusion Administrator の [拡張機能] の下にある [Java アプレット] ページを使用して、Java アプレットを登録します。アプレットのインターフェイスは ColdFusion Administrator で定義し、カプセル化するので、cfapplet タグの呼び出しは簡単です。

フォームの cfapplet タグには、HTML の applet タグにはない、いくつかの利点があります。

- **戻り値** : cfapplet タグには、フォーム変数の名前を指定する name という必須属性があります。したがって、アプレットの戻り値を取得するために追加の JavaScript コードを作成する必要はありません。他の ColdFusion フォーム変数と同様に、Form.< 変数名 > の形式で戻り値を参照できます。
- **使いやすさ** : アプレットのインターフェイスは ColdFusion Administrator で定義するので、ページ内の cfapplet タグでは、アプレット名を参照し、フォーム変数名を指定するだけで済みます。
- **パラメータのデフォルト** : ColdFusion Administrator で定義したパラメータ名と値のペアが使用されます。cfapplet タグでパラメータと値のペアを指定すれば、定義済みの値を上書きできます。

登録したアプレットを使用するには、アプレットソースとフォーム変数名を指定します。

```
<cfapplet appletsource="Calculator"name="calc_value">
```

一方、HTML の applet タグでは、ColdFusion ページで使用するたびにアプレットのパラメータをすべて宣言する必要があります。

Java アプレットの登録

ColdFusion ページで Java アプレットを使用する前に、ColdFusion Administrator でアプレットを登録します。

1 ColdFusion のプログラムグループにある [Administrator] アイコンをクリックし、Administrator のパスワードを入力して、ColdFusion Administrator を開きます。

2 [拡張機能] の下の [Java アプレット] をクリックします。

[Java アプレット] ページが表示されます。

3 [新規アプレットの登録] ボタンをクリックします。

[登録された Java アプレットの追加 / 編集] ページが表示されます。

4 ColdFusion Administrator オンラインヘルプの説明に従って、各アプレット登録フィールドにオプションを入力します。パラメータを追加するには [追加] ボタンをクリックします。

5 送信ボタンをクリックします。

cfapplet タグによるアプレットの埋め込み

登録したアプレットは、cfapplet タグを使用して、ColdFusion ページに配置できます。cfapplet タグには、appletsource と name の 2 つの必須属性があります。アプレットは既に登録されており、デフォルトのアプレットパラメータも定義されているので、アプレットを実行する cfapplet タグは簡潔に記述できます。

```
<cfapplet appletSource="appletname" name="form_variable">
```

行揃えや配置に関する値の上書き

ColdFusion Administrator で定義した値と異なる値をアプレットで使用するには、cfapplet のオプションパラメータを使用してカスタム値を指定します。たとえば、次の cfapplet タグでは、配置と行揃えにカスタム値を指定しています。

```
<cfapplet appletSource="myapplet"
    name="applet1_var"
    height=400
    width=200
    vspace=125
    hspace=125
    align="left">
```

パラメータ値の上書き

任意のパラメータに新しい値を指定すれば、ColdFusion Administrator でアプレットパラメータに割り当てた値を上書きできます。パラメータを上書きするには、該当するパラメータおよびデフォルト値が ColdFusion Administrator の [アプレット] ページで既に定義されている必要があります。次の例では、Param1 と Param2 の 2 つのパラメータに対して、デフォルト値と異なる値を指定しています。

```
<cfapplet appletSource="myapplet"
    name="applet1_var"
    Param1="registered parameter1"
    Param2="registered parameter2">
```

アプレットから渡されるフォーム変数の処理

cfapplet タグの name 属性は、アクションページ内の Form.< アプレット名 > という変数に対応します。この変数には、cfform タグで実行されたアプレットメソッドの戻り値が格納されます。

すべての Java アプレットに戻り値があるとはかぎりません。たとえば、グラフィカルウィジェットは特定の値を返さないことがあります。このタイプのアプレットでは ColdFusion Administrator 内のメソッドフィールドは空になりますが、そのような場合も cfapplet の name 属性を指定する必要があります。

使用できるメソッドは、登録するアプレットごとに 1 つのみです。アプレットに含まれている複数のメソッドにアクセスする場合は、使用するメソッドごとに固有の名前でアプレットを登録します。

アプリケーションページにおける Java アプレットの戻り値の参照

1 ColdFusion Administrator の [登録された Java アプレットの追加 / 編集] ページで、メソッドの名前を指定します。

2 cfapplet タグの name 属性にメソッド名を指定します。

ページでアプレットが実行されると、指定した名前のフォーム変数が作成されます。メソッドを指定しないとフォーム変数は作成されません。

データの検証

Adobe ColdFusion では、フォームデータ、変数データ、関数パラメータなどのデータを検証できます。

ColdFusion の検証について

データの検証では、アプリケーションに入力されたデータが特定のタイプや形式設定ルールに従っているかどうかを確認して、データを制御できます。データの検証によって、次のことが行えます。

- ユーザーにフィードバックを提供して、入力情報をすばやく訂正させることができます。たとえば、電話番号フィールドに名前が入力された場合は、フォーム上で即座にフィードバックを提供することができます。また、正しい形式で番号を入力するように強制することもできます。
- 処理できない無効なデータによってアプリケーションエラーが発生するのを防止できます。たとえば、計算に使用する変数が数値以外のデータを持たないようにすることができます。
- 悪意のあるユーザーが、バッファーオーバーランなどのシステムセキュリティの脆弱性を利用するデータを入力できないようにすることで、セキュリティを向上できます。

ColdFusion には、データを検証するためのさまざまな方法が用意されています。これらの方法には、フォームデータを検証するものや、ColdFusion 変数を検証するものがあります。また、ユーザーが ColdFusion に送信する前にフォームデータを検証するものや、ColdFusion サーバー上で検証するものもあります。

データ検証を設計するときは、次の点を検討します。

検証方法 クライアントのブラウザで検証するかサーバーで検証するかを決定します。また、サーバーサイドまたはクライアントサイドでの具体的な検証方法 (フィールドからフォーカスが移動したとき、ユーザーがフォームを送信したときなど) を決定します。

検証のタイプ データの検証に使用する具体的な方法を決定します。有効な電話番号のテストなど、データの有効性をテストするためのルールなどを決定します。

検証方法

検証方法によって、適用対象となる ColdFusion タグやコーディング環境が異なります。たとえばマスキングは、HTML および Flash 形式の cfinput タグでのみ使用できます。また、実行する場所やタイミングも異なります。たとえば、ユーザーがフォームデータを送信するときに検証するにはクライアントブラウザで、データを処理するときに検証するにはサーバーで実行します。

次の表で、ColdFusion の検証方法について説明します。

検証方法	適用対象	実行場所とタイミング	説明
マスク (mask 属性)	HTML および Flash 形式の cfinput タグ	ユーザーがデータを入力したときにクライアントで実行	JavaScript または ActionScript が ColdFusion によって生成され、ユーザーの入力したデータが指定のパターンに基づいて直接制御されます。たとえば、999-999-9999 というパターンを指定した場合、ユーザーは 10 桁の数字を入力する必要があります。このとき、形式設定された電話番号を作成するために - セパレーターが自動的に挿入されます。 マスクの使用方法の詳細については、736 ページの「無効なデータの処理」を参照してください。
onBlur (validateat="onBlur" 属性)	cfinput および cftextarea タグ	データフィールドのフォーカスが移動したときにクライアントで実行	HTML および XML 形式の場合は、ブラウザで実行する JavaScript が ColdFusion によって生成され、入力データが有効であるかどうかが確認され、入力が無効である場合は即座にフィードバックが提供されます。 Flash 形式では、Flash のビルトイン検証ルーチンが使用されます。
onSubmit (validateat="onSubmit" 属性)	cfinput および cftextarea タグ	ユーザーが送信ボタンをクリックしたときにクライアントで実行	HTML または XML 形式の場合、検証ロジックは onBlur 検証と同じですが、ユーザーがフォームを送信するまでテストは実行されません。 Flash 形式では、この検証タイプは onBlur 検証と同じです。Flash の検証では、この 2 つのイベントは区別されません。
onServer (validateat="onServer" 属性)	cfinput および cftextarea タグ	送信されたフォームを ColdFusion が取得したときにサーバーで実行	送信されたデータの有効性が ColdFusion によって確認され、データが無効である場合は検証エラーページが実行されます。検証エラーページを指定するには、cferror タグを使用します。
非表示フィールド	HTML のみのフォームを含む、すべてのフォーム	送信されたフォームを ColdFusion が取得したときにサーバーで実行	検証ロジックは onServer 検証と同じですが、追加の非表示フィールドを手動で作成する必要があります。この検証方法は、HTML タグまたは CFML タグで使用できます。 非表示フィールドの使用方法の詳細については、741 ページの「非表示フィールドによるフォームデータの検証」を参照してください。
JavaScript (onValidate ="*function*" 属性)	HTML フォームおよび XML フォームの cfgrid、cfinput、cfslider、cftextarea、および cftree タグ	ユーザーが送信ボタンをクリックしたときに、フィールド固有の onSubmit 検証が実行される前に、クライアントで実行	ColdFusion からブラウザに送信される HTML ページに指定の JavaScript 関数が含められ、ブラウザによってこの関数が呼び出されます。 JavaScript を使用した検証方法の詳細については、745 ページの「JavaScript によるフォーム入力の検証およびエラー処理」を参照してください。

検証方法	適用対象	実行場所とタイミング	説明
IsValid 関数	ColdFusion 変数	関数が実行されるときにサーバーで実行	変数が指定の検証ルールに準拠しているかどうかが ColdFusion によってテストされ、true または false が返されます。 IsValid 関数を使用した検証方法の詳細については、748 ページの「IsValid 関数および cfparam タグによるデータの検証」を参照してください。
cfparam タグ	ColdFusion 変数	タグが実行されるときにサーバーで実行	ColdFusion によって、指定された変数が確認されます。値が検証基準に合致しない場合は、式の例外が生成されます。 cfparam タグを使用した検証方法の詳細については、748 ページの「IsValid 関数および cfparam タグによるデータの検証」を参照してください。
cfargument タグ	UDF および CFC 関数の引数	関数が呼び出されたときにサーバーで実行	引数の値が関数に渡されるときに、ColdFusion によってその値が確認されます。値が検証基準に合致しない場合は、アプリケーション例外が生成されます。 cfargument タグの使用方法の詳細については、150 ページの「ユーザー定義関数の作成と呼び出し」を参照してください。

注意：ColdFusion の例外処理の詳細については、275 ページの「エラー処理」を参照してください。

検証方法の選択

検証方法を選択するときは、次の点に注意してください。

- フォームデータを検証する場合、フォームのタイプ (HTML、Flash、XML) によって使用可能な方法が異なることがあります。たとえば、検証の制限はフォームのタイプによって異なります。
- 検証方法によって、適用対象のフォームコントロールやデータ型が異なります。
- 検証方法によって、データ検証を実行できるタイミングや場所が異なります。場所としてはクライアントとサーバーがあります。タイミングとしては、ユーザーがデータを入力するとき、フォームを送信するとき、ColdFusion で変数または関数の引数が処理されるときがあります。
- それぞれの検証方法には、ユーザーフィードバックの形式や機能の制限などの、固有の特徴や注意事項が存在します。
- 環境やアプリケーションで対応する必要があるセキュリティ問題も、検証方法の選択に影響を与えます。

前述の表に、注意事項の一部が示されています (729 ページの「検証方法」を参照してください)。次の表では、検証方法を選択するための追加の注意事項を示します。フォームフィールドに関する追加の注意事項については、735 ページの「検証タイプに関する注意事項」を参照してください。

検証方法	機能	注意事項	セキュリティ問題
マスク (mask 属性)	ユーザーの入力を直接制御します。	cfinput タグに制限されます。比較的簡単なユーザー入力パターンに限られます。	HTML および XML 形式では、JavaScript がブラウザで直接実行されるので、検証が回避される可能性があります。
onBlur (validateat="onBlur" 属性)	ユーザーが無効なデータを入力した場合に、即座にフィードバックを提供します。	cfinput および cftextarea タグに制限されます。HTML または XML 形式では、ブラウザで JavaScript が有効にされている必要があります。	HTML および XML 形式では、JavaScript がブラウザで直接実行されるので、検証が回避される可能性があります。
onSubmit (validateat="onSubmit" 属性)	ユーザーは入力したデータをすべて利用できます。無効なデータのみ再入力する必要があります。	cfinput および cftextarea タグに制限されます。Flash 形式では、onBlur と同じです。HTML または XML 形式では、すべてのフィールドが入力された後に検証されます。ブラウザで JavaScript が有効にされている必要があります。	HTML および XML 形式では、JavaScript がブラウザで直接実行されるので、検証が回避される可能性があります。
onServer (validateat="onServer" 属性)	ブラウザサポートを必要としません。	cfinput および cftextarea タグに制限されます。	フォームとともに検証ルールが送信されるので、検証が回避される可能性があります。
非表示フォームフィールド	ブラウザサポートを必要としません。HTML または CFML フォームの要素に対して使用できます。	フォームに制限されます。	フォームとともに検証ルールが送信されるので、検証が回避される可能性があります。
JavaScript (onValidate = "*function*" 属性)	ブラウザでサポートされているすべてのクライアントサイド処理が実行できます。HTML または CFML フォームの要素に対して使用できます。	特定の ColdFusion フォームタグに制限されます。呼び出す JavaScript 関数は 1 つのみです。JavaScript のサポートレベルは、ブラウザによって異なる場合があります。また、ユーザーがブラウザの JavaScript を無効にしている可能性があります。	JavaScript がブラウザで直接実行されるので、検証が回避される可能性があります。
IsValid 関数	フォームフィールドだけでなく、任意の変数で使用できます。返された結果 (Yes または No) によって、さらなる処理が必要かどうかを判断できます。	フォームフィールドで使用した場合は、データの送信後に実行されます。変数の検証が必要になるたびに使用する必要があります。フォームの検証方法では実行できないデータ型の確認が行えます。	なし
cfparam タグ	フォームフィールドだけでなく、任意の変数で使用できます。このタグは、データの検証に加えて、デフォルト値を設定できます。	フォームフィールドで使用した場合は、データの送信後に実行されます。検証エラーが発生した場合は、エラー処理コードを使用して対応します。	なし
cfargument タグ	cffunction タグを使用して記述された関数の引数に使用されます。	サーバーで関数が呼び出されたときに実行されます。検証エラーが発生した場合は、エラー処理コードを使用して対応します。	なし

セキュリティに関する注意事項

フォーム固有の検証方法では、無効なデータや誤った形式のデータが送信されることは防げますが、悪意のあるデータが HTML フォームから送信されることは防げません。悪意のあるユーザーは、ブラウザ上で実行される JavaScript による検証を回避したり、非表示フィールドから検証ルールが送信されないようにする可能性があります。悪意のあるデータが送信されないようにする必要がある場合は、次の方法を使用することを検討してください。

- Flash フォームの onSubmit または OnBlur 検証。Flash のビルトイン検証が使用されます。
- IsValid 関数、cfparam タグ、および cfargument タグ。CFML コードで変数や引数をテストできます。
- cfquery タグ内の cfqueryparam タグ。悪意のあるクエリー入力からデータベースを保護できます。419 ページの「cfqueryparam によるセキュリティの強化」を参照してください。
- スクリプト保護オプション。クロスサイトスクリプティング攻撃を阻止できます。このオプションは、ColdFusion Administrator の [サーバーの設定]-[設定] ページで設定するか、"Application.cfc" の This.scriptProtect 変数または cfapplication タグの scriptprotect 属性を使用して設定します。クロスサイトスクリプティング攻撃およびこのオプションの詳細については、『CFML リファレンス』の cfapplication タグに関するページを参照してください。

データ検証のタイプ

次の表に、ColdFusion のほとんどの検証方法で検証可能なデータのタイプを示します。ただし、マスク検証は含まれていません。一部の検証タイプは、サポートされている検証方法が限られています。その場合は、表の中にその制限が示されています。Flash フォームの onBlur および onSubmit の検証アルゴリズムは、表に示されているアルゴリズムと異なる場合があります。多くの場合、Flash フォームのアルゴリズムでは利用できる機能が少なくなります。[タイプフィールド] 列のアスタリスク (*) は、そのフィールドが必須であることを示します。onServer 検証アルゴリズムの詳細については、741 ページの「非表示フィールドによるフォームデータの検証」の表を参照してください。

タイプフィールド	説明
date	サーバーで検証する場合は、IsDate 関数で true が返される任意の日付 / 時刻形式 (時刻値を含む) が許可されます。クライアントで検証する場合は、USdate と同じです。
USdate *	mm/dd/yy の形式の US 日付 (月と日は 1 ～ 2 桁、年は 1 ～ 4 桁)。区切り文字は、スラッシュ (/)、ハイフン (-)、ピリオド (.) のいずれかです。
eurodate *	dd/mm/yy の形式の日付 (月と日は 1 ～ 2 桁、年は 1 ～ 4 桁)。区切り文字は、スラッシュ (/)、ハイフン (-)、ピリオド (.) のいずれかです。
time *	サーバーで検証する場合は、IsDate 関数で True が返される任意の日付 / 時刻形式 (日付値を含む) が許可されます。クライアントで検証する場合は、hh:mm[:ss] [A/PM] の形式の時刻が許可されます。
float *	数値。整数を許可します。サーバーでフォームフィールドを検証する場合、整数値は実数に変換されます。
numeric	数値。整数を許可します。サーバーでフォームフィールドを検証する場合、整数値は変更されません。
integer *	整数。
range *	range 属性、または max および min 属性によって指定される数値の範囲。
boolean	ブール値に変換可能な値 (Yes、No、True、False または数値。大文字と小文字は区別されません)。
telephone *	米国の標準の電話番号の形式。最初に国番号 1 を付加することや、最大 5 桁の内線番号を付加することができます。x で始めることもできます。
zipcode *	米国の 5 桁または 9 桁の郵便番号。#####-#### の形式。区切り文字にはハイフン (-) かスペースを使用できます。
creditcard *	空白とダッシュは削除されます。mod10 アルゴリズムを使用して番号が検証されます。番号は 13 ～ 16 桁であることが必要です。
ssn * または social_security_number *	米国の社会保障番号。###-##-#### の形式。区切り文字にはハイフン (-) かスペースを使用できます。

タイプフィールド	説明
email *	有効な電子メールアドレス。名前 @ サーバー . ドメインの形式。形式が正しいかどうかは確認されますが、アクティブで有効な電子メールアドレスであるかどうかは確認されません。
URL *	有効な URL パターン。http、https、ftp ファイル、mailto、および news URL がサポートされています。
guid *	Microsoft/DCE 形式に準拠する一意の識別子 (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx)。x には 16 進数の数字が入ります。
uuid *	ColdFusion の形式 (xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx) に準拠する UUID (Universally Unique Identifier)。x には 16 進数の数字が入ります。
regex * または regular_expression *	pattern 属性で指定された正規表現と値を照合します。HTML および XML 形式でのみ有効です。Flash 形式では無視されます。

注意：onServer 検証および非表示フィールド検証におけるデータの処理方法の詳細については、741 ページの「非表示フィールドによるフォームデータの検証」を参照してください。

次の検証タイプは、cfinput タグでのみ使用できます。

タイプ	説明
maxlength	maxlength 属性で指定した最大文字数に入力を制限します。
noblanks	空白のみで構成されるフィールドを許可しません。この検証は、required 属性が True の場合にのみ使用されます。
SubmitOnce	cfform の submit および image タイプでのみ使用します。次のページがロードされる前にユーザーが同じフォームを複数回送信できないようにします。たとえば、注文確認ページが表示される前に、同じ注文フォームを何度も送信できないようにすれば、注文の重複が防げます。これは、HTML および XML 形式のみで有効です。Flash 形式では無視されます。

次の検証タイプは、cfparam タグ、cfargument タグ、および IsValid 関数のみで使用できます。

タイプ	説明
any	任意のタイプの値
array	値の配列
binary	バイナリ値
query	クエリーオブジェクト
string	文字列値または 1 つの文字
struct	構造体
variableName *	ColdFusion 変数のネーミング規則に準拠した形式を持つ文字列

フォームフィールドの検証

基本的なフォームフィールド検証では、次のことを行います。

- cfinput タグまたは cftextarea タグを使用します。
- 数値などの検証タイプを指定します。複数のタイプを指定することもできます。
- オプションで、エラーメッセージを指定します。
- オプションで、検証方法を指定します。デフォルトでは onSubmit 検証が使用されます。

次の例では、電話番号の onBlur 検証を指定しています。

```
Phone: <cfinput type="text" name="HPhone"
        validateat="onBlur"
        validate="required,telephone"
        message="Please enter a standard U.S. telephone number with an optional
            extension, such as x12345">
```

次のセクションでは、cfinput および cftextarea タグの検証に関する注意事項について説明し、より詳細な例を示します。

検証タイプに関する注意事項

一般的な注意事項: フォームデータの検証方法を決定するときは、次の項目を検討してください。

- onBlur、onSubmit、onServer、または非表示フォームフィールドを使用してフォームデータを検証する場合は、検証する各フィールドに対して 1 つまたは複数の検証タイプを指定することができます。たとえば、フィールドエントリが必須であり、そのタイプが数値であることを指定できます。onSubmit、onBlur、または onServer 検証で複数の検証タイプを指定するには、カンマ区切りのリストでタイプの値を指定します。
- onBlur、onSubmit、または onServer タイプ検証を使用する場合は、検証する各フィールドに対して 1 つのエラーメッセージのみを指定できます。非表示フィールド検証を使用する場合は、各検証ルールに対してカスタムメッセージを作成できます (範囲チェックの場合は例外)。
- cfinput タグのほとんどの検証タイプ属性は、テキストフィールドまたはパスワードフィールドにのみ適用できます。

検証アルゴリズムの相違: フォームデータの検証時に使用される検証コードは、検証方法およびフォームのタイプによって異なる場合があります。したがって、次のように、場合によって使用されるアルゴリズムが異なります。

- onSubmit および OnBlur 検証の日付時刻値に使用される検証アルゴリズムは、すべてのサーバーサイドの検証方法で使用される検証アルゴリズムと異なります。
- Flash の onSubmit および OnBlur 検証に使用されるアルゴリズムは、HTML または XML 形式で使用されるアルゴリズムとは異なる場合があります。一般的に、より単純なルールに準拠します。

サーバーでの検証方法に使用される検証アルゴリズムの詳細については、741 ページの「非表示フィールドによるフォームデータの検証」を参照してください。

XML スキン可能フォームでのデータの検証

XML スキン可能フォームを作成し、Adobe によって提供されるスキン ("basic.xsl" や "silver.xsl" スキンなど) を使用する場合は、HTML フォームで使用できるすべてのフォーム検証方法を使用できます。

カスタムスキン (XSL ファイル) を使用する場合、使用可能な検証方法はスキンによって異なります。"<ColdFusion Web ルートディレクトリ >¥CFIDE¥scripts¥xsl" ディレクトリには "_cfformvalidation.xsl" ファイルが格納されています。このファイルには、ColdFusion の HTML フォーム検証方法がすべて実装されており、onBlur、onSubmit、onServer、および非表示フォームフィールド検証がサポートされています。XML スキンライタでスキン XSLT にこのファイルを含めて、スキンで使用する ColdFusion 検証を実装できます。

例 : 基本的なフォーム検証

次のフォームでは、新規ユーザーの登録時に必要な情報を要求します。このフォームでは、ユーザーが必須情報を入力したかどうかを確認します。電話番号のみがオプションです。また、電話番号および電子メールアドレスの形式と、リマインダ番号が適切な範囲であるかどうかも確認します。この例では、onSubmit 検証を実行します。同じページに値を渡して、結果をダンプします。

```
 <cfif IsDefined("form.fieldnames")>
    <cfdump var="#form#"><br>
</cfif>

<cfform name="myform" preservedata="Yes" >
    First Name: <cfinput type="text" size="15" name="firstname"
        required="yes" message="You must enter a first name."><br>
    Last Name: <cfinput type="text" size="25" name="lastname"
        required="yes" message="You must enter a last name."><br>
    Telephone: <cfinput type="text" size="20" name="telephone"
        validate="telephone" message="You must enter your telephone
        number, for example 617-555-1212 x1234"><br>
    E-mail: <cfinput type="text" size="25" name="email"
        validate="email" required="Yes"
        message="You must enter a valid e-mail address."><br>
    Password:<cfinput type="password" size="12" name="password1"
        required="yes" maxlength="12"
        message="You must enter a password."><br>
    Reenter password:<cfinput type="password" size="12" name="password2"
        required="yes" maxlength="12"
        message="You must enter your password twice."><br>
    We will ask you for the following number, in the range 100-999 if you forget
        your password.<br>
    Number: <cfinput type="text" size="5" name="chalenge"
        validate="range" range="100,999" required="Yes"
        message="You must enter a reminder number in the range 100-999."><br>
    <cfinput type="submit" name="submitit">
</cfform>
```

無効なデータの処理

無効なデータをどのように処理するかは、検証タイプによって異なります。ここでは、検証エラーの処理ルールと注意事項について説明します。無効なデータの処理を含む、ColdFusion のエラー処理の詳細については、275 ページの「エラー処理」を参照してください。

1 onBlur、onSubmit、または onServer 検証では、cfinput タグまたは cftextarea タグの message 属性を使用して、テキストのみのエラーメッセージを指定できます。指定しなかった場合は、無効なフォームフィールドの名前を含むデフォルトのメッセージが使用されます。OnServer 検証の場合は、281 ページの「フォームフィールド検証エラーの処理」の説明にあるように、このメッセージをカスタマイズできます。次の例では、ユーザーが無効な電子メールアドレスを入力したときにエラーメッセージを表示します。

```
E-mail: <cfinput type="text" size="25" name="email"
    validate="email" message="You must enter a valid e-mail address.">
```

2 非表示フォーム検証の場合は、非表示フィールドの value 属性に、テキストのみのエラーメッセージを指定できます。指定しなかった場合は、無効なフォームフィールドの名前を含むデフォルトのメッセージが使用されます。281 ページの「フォームフィールド検証エラーの処理」の説明にあるように、このメッセージはカスタマイズできます。たとえば次の cfinput タグでは、非表示フィールド検証を使用して、ユーザーが無効なアドレスを入力したときにエラーメッセージを表示します。この例では、ユーザーが数値を入力しなかった場合にエラーメッセージを表示するために、onServer 検証も使用しています。

```
Telephone: <cfinput type="text" size="20" name="telephone"
    validateat="onServer" required="Yes"
    message="You must enter a telephone number">
    <cfinput type="hidden" name="telephone_cfformtelephone"
    value="The number you entered is not in the correct format.<br>Use a
    number such as (617) 555-1212, 617-555-1212, or 617-555-1212 x12345">
```

3 HTML および XML フォーム (ColdFusion スキンを使用) の場合は、ほとんどの ColdFusion フォームタグで onError 属性を使用できます。onSubmit エラーが発生した場合に実行する JavaScript 関数をこの属性で指定できます。

4 IsValid 関数では、データが有効な場合のコードパスと、無効な場合のコードパスを作成します。次の例は、簡素化されたケースを示しています。ユーザーが無効な電子メールアドレスを入力した場合は特定のエラーメッセージを、アドレスが有効な場合は別のメッセージを表示します。

```
<cfif IsValid("email", custEmail)>
    Thank you for entering a valid address.
    <!--- More processing would go here. --->
<cfelse>
    You must enter a valid e-mail address.<br>
    Click the Back button and try again.
</cfif>
```

5 cfparam および cfargument タグでは、ColdFusion の標準のエラー処理方法を使用します。try ブロックにこのタグを配置し、catch ブロックを使用してエラーを処理することができます。または、カスタムエラー処理ページを使用できます。次に示すフォームアクションページのコードでは、送信されたフォームに無効な電子メールアドレスが含まれている場合に cfparam タグで発生するエラーを、カスタムエラーページ "expresserr.cfm" で処理します。

```
<cferror type="EXCEPTION" exception="expression" template="expresserr.cfm">
<cfif IsDefined("form.fieldnames")>
    <cfparam name="form.custEmail" type="email">
    <!--- Normal form processing code goes here. --->
</cfif>
```

フォーム入力値のマスキング

cfinput タグの mask 属性を使用すれば、text または datefield 入力フィールドに入力可能なデータの形式を制御できます。cfcalendar タグでも mask 属性を使用できます。1 つのフィールドで、マスキングと検証を組み合わせて使用することもできます。

- HTML および Flash フォームでは、text フィールドに入力可能なデータの形式を、マスクを使用して制御できます。
- cfcalendar タグや、Flash フォームの datefield タイプの cfinput フィールドでは、ユーザーがカレンダーで選択した日付に適用する形式を、マスクを使用して制御できます。

注意：標準の ColdFusion XML スキンでは、マスキングはサポートされていません。

テキスト入力のマスキング

テキストフィールドでは、リテラルマスク文字 (たとえば電話番号の "-" など) が自動的に挿入されます。ユーザーは、フィールドの変化する部分のみを入力します。データのマスキングには、次の文字を使用できます。

マスク文字	アクション
A	大文字または小文字を許可します : A ～ Z および a ～ z。
X	大文字、小文字または数字を許可します : A ～ Z、a ～ z、および 0 ～ 9。
9	数字を許可します : 0 ～ 9。
?	任意の文字を許可します。
その他のすべての文字	リテラル文字を自動挿入します。

次のパターンでは、EB-1234-c1-098765 という形式で部品番号を入力するように規定しています。ユーザーは、最初の数字 (1 など) から入力を開始します。その前の EB という接頭辞やすべてのハイフン (-) は、自動的に挿入されます。ユーザーは、4 桁の数字を入力し、その後に 2 つの英数字と、6 桁の数字を入力する必要があります。

```
<cfinput type="text" name="newPart" mask="EB-9999-XX-999999" />
```

注意：ユーザーに対して、A、X、9、または ? を入力するように強制することはできません。すべての文字を大文字または小文字に揃える必要がある場合は、アクションページで ColdFusion の UCase または LCase 関数を使用します。

cfcalendar および datefield 入力のマスキング

cfcalendar タグおよび Flash 形式の datefield 入力コントロールでは、次のマスクを使用して出力形式を指定します。マスクには大文字または小文字を使用できます。

マスク	パターン
D	1 桁または 2 桁で表される日付 (1 や 28 など)
DD	2 桁で表される日付 (01 や 28 など)
M	1 桁または 2 桁で表される月 (1 や 12 など)
MM	2 桁で表される月 (01 や 12 など)
MMM	月の省略名 (Jan や Dec など)
MMMM	月の名前 (January や December など)
YY	2 桁で表される年 (05 など)
YYYY	4 桁で表される年 (2005 など)
E	1 桁で表される曜日。0 (日曜日) ～ 6 (土曜日)
EEE	曜日の省略名 (Mon や Sun など)
EEEE	曜日名 (Monday や Sunday など)

次のパターンは、Flash フォームの datefield 入力コントロールで選択された日付を、04/29/2004 という形式のテキストとして ColdFusion に送信するように指定します。

```
<cfinput name="startDate" type="datefield" label="date:" mask="mm/dd/yyyy"/>
```

正規表現によるフォームデータの検証

cfinput タグおよび cftextinput タグに入力されたテキストを、正規表現を使用して一致および検証することができます。一致パターンは、標準文字と特殊文字を組み合わせて定義します。ユーザー入力とパターンが一致した場合にのみ、検証は成功します。

正規表現を使用すれば、状況に合った多様な入力パターンを規定して、入力テキストを確認できます。末尾が異なるさまざまな単語を許可するなどの複雑なパターン検証も、簡単な正規表現を使用して実現できます。

正規表現では、ColdFusion の変数や関数を使用できます。ColdFusion サーバーでは、正規表現が評価される前に、変数と関数が評価されます。したがって、他の入力データやデータベース値からダイナミックに検証パターンを生成することができます。

注意：ここで説明しているルールは、JavaScript 正規表現のルールです。これは、cfinput タグと cftextinput タグで使用する正規表現にのみ適用されます。ColdFusion 関数の REFind、REReplace、REFindNoCase、および REReplaceNoCase で使用されるルールとは異なります。ColdFusion 関数で使用する正規表現の詳細については、129 ページの「関数での正規表現の使用」を参照してください。

特殊文字

正規表現では、特殊文字は演算子として扱われます。特殊文字を通常の文字として使用するには、特殊文字の前に円記号 (¥) を付けてエスケープします。たとえば、円記号 (¥) を表すには、円記号を 2 つ (¥¥) 使用します。

1 文字の正規表現

次のルールは、1 文字に一致する正規表現に適用されます。

- 特殊文字 : + * ? . [^ $ ( ) { | ¥
- 特殊文字でない文字や、円記号 (¥) が前に付いてエスケープされた文字は、その文字自体に一致します。
- 円記号 (¥) が前に付いた特殊文字は、そのリテラル文字に一致します。つまり、円記号 (¥) によって、特殊文字がエスケープされます。
- ピリオド (.) は、改行文字以外の任意の文字に一致します。
- 角括弧 ([]) で囲まれた文字セットは、そのセット内の任意の文字に一致する 1 文字の正規表現です。たとえば、"[akm]" は、**a**、**k**、または **m** のいずれかの文字に一致します。角括弧内に右角括弧 (]) を含める場合は、右角括弧を最初の文字にする必要があります。それ以外の場合は、¥] を使用しても、正しく動作しません。
- ハイフン (-) は文字の範囲を表します。たとえば、[a-z] は任意の小文字に一致します。
- 角括弧内の文字セットの前にキャレット (^) を挿入した場合は、その文字セットに含まれていない任意の文字に一致します。空の文字列には一致しません。たとえば、"[^akm]" は、**a**、**k**、**m** 以外の任意の文字に一致します。キャレット (^) は、文字セットの最初の文字として使用した場合にのみ、この特殊な意味を持ちます。
- 個々の文字を文字セットに置き換えて、大文字と小文字を区別しないようにできます。たとえば "[Nn][Ii][Cc][Kk]" は、Nick という名前の大文字と小文字を区別しないパターンを表します (NICK、nick、nIcK など)。
- 次のエスケープシーケンスを使用すれば、特殊な文字や文字クラスに一致させることができます。

エスケープシーケンス	一致	エスケープシーケンス	説明
[¥b]	円記号	¥s	スペース、タブ、フォームフィード、ラインフィードのいずれかの空白文字
¥b	スペースなどの単語の境界	¥S	¥s で一致する空白文字を除いた任意の文字
¥B	単語以外の境界	¥t	タブ
¥cX	制御文字 Ctrl-x。たとえば、¥cv はテキストをペーストするための制御文字 Ctrl-v に一致します。	¥v	垂直方向のタブ
¥d	1 桁の数字。[0-9] と同じ。	¥w	英数字またはアンダースコア。[A-Za-z0-9_] と同じ。
¥D	1 桁の数字を除く任意の文字	¥W	¥w で一致しない任意の文字。[^A-Za-z0-9_] と同じ。
¥f	フォームフィード	¥n	括弧で囲まれた n 番目の式へのバックリファレンス。740 ページの「バックリファレンス」を参照してください。
¥n	ラインフィード	¥ooctal	指定の 8 進数値によって表される、ASCII 文字テーブルの文字
¥r	キャリッジリターン	¥¥x**hex**	指定の 16 進数値によって表される、ASCII 文字テーブルの文字

複数文字の正規表現

複数文字の正規表現を構築するには、次のルールに従います。

- 括弧を使用して、正規表現の一部をサブ式としてグループ化すれば、1 つのユニットとして処理できます。たとえば "(ha)+" は、"**ha**" の 1 回以上の繰り返しに一致します。
- 後ろにアスタリスク (*) が付いている 1 文字の正規表現またはサブ式は、その 0 回以上の繰り返しに一致します。たとえば、"[a-z]*" は 0 文字以上の小文字に一致します。

- 後ろにプラス記号 (+) が付いている 1 文字の正規表現またはサブ式は、その 1 回以上の繰り返しに一致します。たとえば、"[a-z]+" は、1 文字以上の小文字に一致します。
- 後ろに疑問符 (?) が付いている 1 文字の正規表現またはサブ式は、その 0 回または 1 回の繰り返しに一致します。たとえば、"xy?z" は "**xyz**" または "**xz**" に一致します。
- 正規表現の先頭に挿入したキャレット (^) は、フィールドの先頭に一致します。
- 正規表現の末尾に挿入したドル記号 ($) は、フィールドの末尾に一致します。
- 正規表現を連結すると、対応する文字列を連結したものに一致します。たとえば、"[A-Z][a-z]*" は大文字で始まる任意の単語に一致します。
- OR 文字 (|) を使用すると、2 つの正規表現から選択できます。たとえば、"jell(y|ies)" は "**jelly**" または "**jellies**" に一致します。
- 中括弧 ({}) は、正規表現を繰り返す範囲を指定します。"{m, n}" という形式の場合、**m** は範囲の開始を示す 0 以上の整数です。**n** は範囲の終了を示す **m** 以上の整数です。たとえば、"(ba){0,3}" は "**ba**" の 3 回までの繰り返しに一致します。"{m,}" という形式を使用すると、前にある正規表現の **m** 回以上の繰り返しに一致します。"{m}" という形式を使用すると、前にある正規表現の **m** 回の繰り返しに一致します。"{,n}" という形式は使用できません。

バックリファレンス

バックリファレンスを使用すると、以前に一致した括弧内のテキストに一致させることができます。バックリファレンスは、円記号が前に付いた 1 桁の数字 (¥n) で表します。これは、括弧で囲まれた **n** 番目のサブ式を表します。

バックリファレンスの使用例としては、重複語の検索があります。たとえば、テキスト内の "the the" や "is is" という箇所を検索できます。次に、正規表現でバックリファレンスを使用する例を示します。

```
(\b[A-Za-z]+)[ ]+\1
```

このコードは、同じ単語が 2 個連続して出現する箇所に一致します。つまり、ある単語 (単語の境界を表す特殊文字 ¥b と "[A-Za-z]+") の後に 1 つ以上のスペース ("[ ]+") があり、括弧で囲まれた最初のサブ式で一致したテキスト (最初の単語) がその後に続いている箇所に一致します。たとえば、"is is" には一致しますが、"This is" には一致しません。

完全一致と部分一致

ColdFusion の検証では、通常、正規表現パターンに一致する部分が値の中に含まれていれば、その値は有効と見なされます。しかし、値全体がパターンに一致している場合に有効と見なしたい場合は、次のように、フィールドの先頭と末尾に " アンカー " する特殊文字を、パターンに挿入します。

- キャレット (^) をパターンの先頭に挿入すると、そのパターンに一致する文字列で始まっているフィールドのみが有効と見なされます。
- ドル記号 ($) をパターンの末尾に挿入すると、そのパターンに一致する文字列で終わっているフィールドのみが有効と見なされます。
- キャレット (^) とドル記号 ($) をそれぞれパターンの先頭と末尾に挿入すると、そのパターンに正確に一致するフィールドのみが有効と見なされます。

正規表現の例

次の表に、正規表現の例と、それに一致するものを示します。

式	説明
[¥?&]value=	URL パラメータ値が含まれている任意の文字列
^[A-Z]:(¥¥[A-Z0-9_]+)+$	大文字の Windows ディレクトリパスのうち、ドライブのルートではなく、英数字とアンダースコアのみで構成されているパス
^(¥+|-)?[1-9][0-9]*$	0 で始まらない整数 (符号が付いている場合はそれも含む)
^(¥+|-)?[1-9][0-9]*(¥.[0-9]*)?$	実数値
^(¥+|-)?[1-9]¥.[0-9]*E(¥+|-)?[0-9]+$	科学的記数法で表された実数値
a{2,4}	"**a**" の 2 ～ 4 回の繰り返し (aa、aaa、aaaa) が含まれている文字列。たとえば、aardvark には一致しますが automatic には一致しません。
(ba){2,}	"**ba**" の 2 回以上の繰り返しが含まれている文字列。たとえば、Ali baba には一致しますが、Ali Baba には一致しません。

注意：正規表現については、『Mastering Regular Expressions』(Jeffrey E.F. Friedl 著、O'Reilly & Associates, Inc 発行) を参照してください。

非表示フィールドによるフォームデータの検証

ColdFusion では、非表示フォームフィールドを使用して、サーバー上でフォームフィールドを検証できます。非表示フォームフィールドには、検証するフィールドの名前に検証タイプを付加した名前を付けます。非表示フィールド検証では、ColdFusion フォームフィールドの onServer 検証と同じ方法およびアルゴリズムが使用されます。

非表示フィールド検証には次の特徴があります。

- 標準の HTML タグで使用できます。たとえば、HTML input タグのデータを検証できます。これは、ColdFusion MX 7 よりも前のリリースで有用です。それらのリリースでは、cfinput タグで一部の HTML type 属性しかサポートされていませんでした。
- ColdFusion MX 7 よりも前のリリースと後方互換性があります。それらのリリースでは、サーバーで検証を行う唯一の方法が非表示フィールド検証でした。
- 検証タイプごとに別個のタグを使用するので、1 つのフィールドに複数の検証ルールを使用する場合は、各ルールで異なるエラーメッセージを指定できます。
- 非表示フィールド検証は、input、cfinput、textarea、cftextarea だけでなく、データ値を送信する任意のフォームフィールドタイプで使用できます。

非表示フォームフィールド検証の指定

非表示フィールド検証を指定するには、次のことを行います。

- 作成する検証ルールごとに、type="hidden" を指定した HTML input 要素または CFML cfinput タグを作成します。
- 非表示フィールド名の最初の部分として、検証するフィールドの名前を指定します。
- 非表示フィールド名の 2 番目の部分として、アンダースコア (_) 文字の後に検証タイプを指定します。
- 1 つのフォームデータフィールドに、複数のルールを指定できます。たとえば、myValue というフィールドに range 検証と required 検証を指定するには、myValue_cfformrange および myValue_cfformrequired という非表示フィールドを作成します。
- ほとんどの検証タイプで、フィールドの value 属性にエラーメッセージを指定します。
- 範囲、最大長、または正規表現検証では、value 属性に最大長などのルールを指定します。これらの検証タイプでは、カスタムエラーメッセージを指定できません。

次の例では、非表示フィールドを使用して、データフィールドへの入力を必須にし、フィールドに日付を入力するように指定します。この例は、HTML タグのみで構成されます。

```
 <input type="text" name="StartDate" size="16" maxlength="16"><br>
<input type="hidden" name="StartDate_required"
        value="You must enter a start date.">
<input type="hidden" name="StartDate_date"
    value="Please enter a valid date as the start date.">
```

非表示フォームフィールド名の最後に指定する、検証タイプを表す接尾辞を次に示します。このタイプ識別子は、常にアンダースコアから始まります。検証ルールによっては、名前を 2 つ持つものがあります。"_cf" で始まらない名前は、以前のリリースで使用されていたものです。後方互換性を維持するために、それらの名前も引き続き使用できるようにしています。ただし、一貫性の維持と明確化のため、新しいフォームでは "_cf" で始まる名前を使用することをお勧めします。

フィールド名の接尾辞	検証内容
_integer、_cfforminteger	-2,147,483,648 ～ 2,147,483,647 の整数先頭の文字 "$ ¤ ¥£ +" は有効な入力として処理されますが、数値からは削除されます。
_cfformnumeric	任意の数値。先頭の文字 "$ ¤ ¥£ +" は有効な入力として処理され、数値から削除されません。
_float、_cfformfloat	最大有効桁数 12 桁の浮動小数点数として表せる任意の値 (整数を含む)。先頭の文字 "$ ¤ ¥£ +" は有効な入力として処理されますが、数値からは削除されます。 入力データを実数に変換します。たとえば、アクションページで整数値をダンプすると、".0" が表示されます。
_range、_cfformrange	value 属性で指定した範囲内の数値。value 属性では、"min=minvalue max=maxvalue" の形式で範囲を指定します。この検証では、カスタムエラーメッセージを指定できません。
_date、_cfformdate	ColdFusion で認識可能な形式の日付。入力は ODBC 日付形式に変換されます。時刻が含まれていても有効と見なされますが、ODBC 値からは削除されます。
_cfformusdate	m/d/y、m-d-y、または m.d.y の形式の日付。m および d は 1 桁または 2 桁、y は 2 桁または 4 桁です。文字列から ODBC 値への変換は行われません。時刻が含まれている場合は無効と見なされます。
_eurodate、_cfformeurodate	d/m/y、d-m-y、または d.m.y の形式の日付。m および d は 1 桁または 2 桁、y は 2 桁または 4 桁です。入力は ODBC 日付形式に変換されます。時刻が含まれていても有効と見なされますが、ODBC 値からは削除されます。
_time、_cfformtime	時刻。12 時間表記と 24 時間表記が使用できます。hh:mm:ss の形式で秒を含めることができます。am または pm (大文字と小文字は区別されない) を含めることもできます。 入力は ODBC 時刻形式に変換されます。日付が含まれていても有効と見なされますが、ODBC 値からは削除されます。
_cfformcreditcard	空白とハイフンを除いた番号が、mod10 アルゴリズムに準拠していることを検証します。番号は 13 ～ 16 桁であることが必要です。
_cfformSSN、 _cfformsocial_security_number	9 桁の社会保障番号。形式は xxx-xx-xxxx または xxx xx xxxx です。
_cfformtelephone	米国の標準の電話番号の形式。国際電話番号はサポートされていません。 地域コードを括弧で囲むことができます (囲まなくてもかまいません)。また、標準の番号グループをハイフン (-)、スペース、ピリオドで区切ることができます (区切らなくてもかまいません)。最初に国番号 1 を付加することや、最大 5 桁の内線番号を付加することができます。x で始めることもできます。地域コードと局番は、1 ～ 9 の数字で始まる必要があります。
_cfformzipcode	米国の 5 桁または 9 桁の郵便番号。9 桁の郵便番号では、最後の 4 桁の前にハイフン (-) かスペースを使用します。
_cfformemail	有効な電子メールアドレス。アドレスに使用できる有効な文字は、a ～ z、A ～ Z、0 ～ 9、アンダースコア (_)、ハイフン (-)、ピリオド (.) およびセパレータです。1 つのアットマーク (@) が含まれており、@ 文字の後のテキストにはピリオドが含まれている必要があります。たとえば、 my_address@MyCo.com や b-b.jones27@hisco.co.uk は有効です。

フィールド名の接尾辞	検証内容
_cfformURL	有効な URL。http://、https://、ftp://、file://、mailto:、または news: で始まる必要があります。必要に応じて、ユーザー名とパスワード識別子、およびクエリー文字列を指定できます。アドレスの主要部分に使用できる文字は、A ～ Z、a ～ z、0 ～ 9 およびハイフン (-) のみです。
_cfformboolean	ブール値として処理できる値 : Yes、No、True、False、0、1
_cfformUUID	ColdFusion の形式 (xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx) に準拠する UUID (Universally Unique Identifier)。x には 16 進数の数字が入ります。
_cfformGUID	Microsoft/DCE 形式に準拠する一意の識別子 (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx)。x には 16 進数の数字が入ります。
_cfformnoblanks	空白が含まれていないことが必要です。_required 非表示フィールドも指定した場合にのみ、この検証が使用されます。
_cfformmaxlength	文字数が、タグの value 属性で指定した数を超えていないことが必要です。
_cfformregex、 _cfformregular_expression	value 属性で指定した JavaScript 正規表現にデータが一致することが必要です。
_required、_cfformrequired	フォームフィールドでデータが入力または選択される必要があります。

非表示フォームフィールドに関する注意事項

非表示フォームフィールド検証を使用するかどうかや、その使用方法を判断するときは、次のルールおよび注意事項を検討してください。

- 標準の HTML input タグのデータを検証する場合は、非表示フィールド検証を使用します。cfinput および cftextarea タグには、validateAt 属性が含まれています。この属性を使用すると、より簡単にサーバーサイド検証を指定できます。
- 単一のフィールドに複数の検証ルールを指定し、各検証で個別のエラーメッセージを表示する場合は、cfinput タグおよび cftextarea タグの非表示フィールド検証を検討してください。
- 表に示されている接尾辞をフィールド名として使用しないでください。
- フィールドに検証ルールを追加しても、そのフィールドは必須フィールドになりません。ユーザーに確実に入力させるには、個別に _required 非表示フィールドを追加します。

非表示フォームフィールドの例

次の手順では、開始日および給与を入力する簡単なフォームを作成します。非表示フィールドを使用して、データが入力されたことと形式が正しいことを検証します。

この例では、自己送信フォームを使用します。つまり、フォームページとアクションページが同一の ColdFusion ページを使用します。このページでは IsDefined 関数を使用して、フォームデータが送信済みであることを確認しています。これによって、入力データを送信しないかぎり、ページに結果を表示しないようにしています。このフォームでは HTML タグのみを使用します。これらのタグは、対応する CFML タグに置き換えることができます。

```
 <html>
<head>
    <title>Simple Data Form</title>
</head>
<body>
<h2>Simple Data Form</h2>

<!--- Form part. --->
<form action="datatest.cfm" method="Post">
    <input type="hidden"
        name="StartDate_cfformrequired"
        value="You must enter a start date.">
    <input type="hidden"
        name="StartDate_cfformdate"
        value="Enter a valid date as the start date.">
    <input type="hidden"
        name="Salary_cfformrequired"
        value="You must enter a salary.">
    <input type="hidden"
        name="Salary_cfformfloat"
        value="The salary must be a number.">
    Start Date:
    <input type="text"
        name="StartDate" size="16"
        maxlength="16"><br>
    Salary:
    <input type="text"
        name="Salary"
        size="10"
        maxlength="10"><br>
    <input type="reset"
        name="ResetForm"
        value="Clear Form">
    <input type="submit"
        name="SubmitForm"
        value="Insert Data">
</form>
<br>

<!--- Action part. --->
<cfif isdefined("Form.StartDate")>
    <cfoutput>
        Start Date is: #DateFormat(Form.StartDate)#<br>
        Salary is: #DollarFormat(Form.Salary)#
    </cfoutput>
</cfif>
</html>
```

このフォームを送信すると、ColdFusion によってフォームフィールドが調べられ、検証ルールが検索されます。次にそのルールに基づいて、ユーザーの入力が分析されます。入力ルールに違反している場合は、エラーページが表示され、非表示フィールドの value 属性で指定されたエラーメッセージが示されます。ユーザーは、フォームの画面に戻って問題を訂正し、フォームを再送信する必要があります。ColdFusion では、ユーザーが正しく入力するまでフォームの送信が受け付けられません。

_cfforminteger および _cfformfloat ルールが設定されているフィールドでは、数値に含まれることが多いカンマや通貨記号は自動的に削除されます。その後、ColdFusion によってフォームフィールドが検証され、フォームのアクションページにデータが渡されます。ただし、_cfformnumeric ルールが設定されているフィールドでは、これらの文字は削除されません。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<form action="datatest.cfm" method="Post">`	このフォームから情報を収集し、Post メソッドを使用して情報を "dataform.cfm" ページ (このページ) に送信します。
`<input type="hidden" name="StartDate_cfformrequired" value="You must enter a start date."> <input type="hidden" name="StartDate_cfformdate" value="Enter a valid date as the start date.">`	" 開始日 " 入力フィールドへの入力を必須にします。入力がない場合は、"You must enter a start date." というエラーメッセージを表示します。また、有効な日付形式で入力するように指定します。入力が有効でない場合は、"Enter a valid date as the start date." というエラーメッセージを表示します。
`<input type="hidden" name="Salary_cfformrequired" value="You must enter a salary."> <input type="hidden" name="Salary_cfformfloat" value="The salary must be a number.">`	" 給与 " 入力フィールドへの入力を必須にします。入力がない場合は、"You must enter a salary." というエラーメッセージを表示します。また、有効な数値を入力するように指定します。入力が有効でない場合は、"The salary must be a number." というエラーメッセージを表示します。
`Start Date: <input type="text" name="StartDate" size="16" maxlength="16"> `	ユーザーが開始日を入力する StartDate というテキストボックスを作成します。テキストボックスの幅を 16 文字に設定します。
`Salary: <input type="text" name="Salary" size="10" maxlength="10"> `	ユーザーが給与を入力する Salary というテキストボックスを作成します。テキストボックスの幅を 10 文字に設定します。
`<cfif isdefined("Form.StartDate")> <cfoutput> Start Date is: #DateFormat(Form.StartDate)#  Salary is: #DollarFormat(Form.Salary)# </cfoutput> </cfif>`	StartDate が定義されている場合にのみ、StartDate および Salary フォームフィールドの値を表示します。これらはフォームが送信されるまで定義されないので、最初のフォームでは表示されません。DateFormat 関数を使用して、開始日をデフォルトの日付形式で表示します。DollarFormat 関数を使用して、ドル記号とカンマ付きで給与を表示します。

JavaScript によるフォーム入力の検証およびエラー処理

ColdFusion では、JavaScript を使用して独自の検証ルーチンやエラーハンドラを作成できます。

JavaScript による入力の検証

cfinput タグおよび cftextarea タグでは、ネイティブの ColdFusion 入力検証を指定できる validate 属性がサポートされていますが、次のタグでは、cfform 入力を検証する JavaScript 関数を指定できる onValidate 属性がサポートされています。

- cfgrid
- cfinput
- cfslider
- cftextarea
- cftree

onValidate 属性で指定した JavaScript 関数には、次の引数が渡されます。

- フォームの JavaScript DOM オブジェクト
- フォーム要素の name 属性
- 検証するコントロールの値

たとえば、次のように cfinput タグを記述します。

```
<cfinput type="text"
    ...
<!--- Do not include () in JavaScript function name. --->
    onvalidate="handleValidation"
    ...
>
```

次のように JavaScript 関数を定義します。

```
<script>
<!--
function handleValidation(form_object, input_object, object_value) {
...
}
//-->
</script>
```

例 : パスワードの検証

次の例では、パスワードを検証します。パスワードには、大文字、小文字、数字がそれぞれ最低 1 つ含まれている必要があります。長さは 8 ～ 12 文字であることが必要です。パスワードが無効な場合は、ブラウザにメッセージボックスを表示します。パスワードが有効な場合は、ページを再表示して、成功を表す簡単なメッセージを示します。

JavaScript によるフォームデータの検証

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>JavaScript Validation</title>

<!--- JavaScript used for validation. --->
<script>
<!--
    // Regular expressions used for pattern matching.
    var anUpperCase = /[A-Z]/;
    var aLowerCase = /[a-z]/;
    var aNumber = /[0-9]/;
    /* The function specified by the onValidate attribute.
        Tests for existence of at least one uppercase, lowercase, and numeric
        character, and checks the length for a minimum.
        A maximum length test is not needed because of the cfinput maxlength
        attribute. */
    function testpasswd(form, ctrl, value) {
        if (value.length < 8 || value.search(anUpperCase) == -1 ||
            value.search (aLowerCase) == -1 || value.search (aNumber) == -1)
        {
            return (false);
        }
        else
        {
            return (true);
        }
    }
//-->
</script>
</head>

<body>
<h2>JavaScript validation test</h2>
<!--- Form is submitted only if the password is valid. --->
```

```
<cfif IsDefined("Form.passwd1")>
    <p>Your Password if valid.</p>
</cfif>
<p>Please enter your new password:</p>

<cfform name="UpdateForm" preservedata="Yes" >
    <!--- The onValidate attribute specifies the JavaScript validation
              function. The message attribute is the message that appears
              if the validation function returns False. --->
    <cfinput type="password" name="passwd1" required="YES"
        onValidate="testpasswd"
        message="Your password must have 8-12 characters and include uppercase
and lowercase letters and at least one number."
        size="13" maxlength="12">

<input type="Submit" value=" Update... ">
</cfform>

</body>
</html>
```

2 このページに "validjs.cfm" という名前を付けて保存します。

3 "validjs.cfm" ページをブラウザで表示します。

失敗した検証の処理

onError 属性を使用すれば、onValidate、onBlur、または onSubmit 検証が失敗した場合に実行する JavaScript 関数を指定できます。たとえば、入力を検証する JavaScript 関数を onValidate 属性で指定する場合は、失敗した検証を処理する JavaScript 関数を onError 属性で指定できます。この関数は、onValidate で False が返された場合に実行されます。onValidate 属性を使用する場合は、検証エラーを処理する JavaScript 関数を onError 属性で指定することもできます。onError 属性は、次の cfform タグで使用できます。

- cfgrid
- cfinput
- cfselect
- cfslider
- cftextinput
- cftree

onError 属性で指定した関数には、次の JavaScript オブジェクトが渡されます。

- JavaScript フォームオブジェクト
- フォーム要素の name 属性
- 検証に失敗した値
- CFML タグの message 属性で指定されたエラーメッセージテキスト

次の例に、onError 属性を使用したフォームを示します。この属性では、alert メソッドでエラーメッセージを表示する showErrorMessage という JavaScript 関数を呼び出しています。この関数のメッセージでは、無効な値と、cfinput タグの message 属性の内容を示しています。

```
<!--- The JavaScript function to handle errors.
         Puts a message, including the field name and value, in an alert box. --->
<script>
    <!--
    function showErrorMessage(form, ctrl, value, message) {
        alert("The value " + value +" of the " + ctrl + " field " + message);
    }
    //-->
</script>

<!--- The form.
    The cfinput tags use the onError attribute to override the ColdFusion
    default error message mechanism. --->
<cfform>
    <!--- A minimum quantity is required and must be a number. --->
    Minimum Quantity: <cfinput type="Text" name="MinQuantity"
        onError="showErrorMessage" validate="numeric" required="Yes"
        message="is not a number." ><br>
    <!--- A maximum quantity is optional, but must be a number if supplied. --->
    Maximum Quantity: <cfinput type="Text" name="MaxQuantity"
        onError="showErrorMessage" validate="numeric"
            message="is not a number." ><br>
    <cfinput type="submit" name="submitit">
</cfform>
```

IsValid 関数および cfparam タグによるデータの検証

IsValid 関数と cfparam タグは、フォーム変数だけでなく、任意の ColdFusion 変数値を検証できます。この関数およびタグは、ColdFusion サーバーですべての処理が実行されるので、必要な検証手順が確実に実行される安全なメカニズムとして使用できます。ユーザーが、送信するフォームデータを変更して検証を回避することは不可能です。エラー処理コードではすべての CFML シンタックスを使用できるので、ユーザーエラーにも柔軟に対応できます。

この 2 つの検証方法では、次の処理が実行されます。

- IsValid 関数では、ColdFusion 変数の値がテストされます。値が有効な場合は True が返され、無効な場合は False が返されます。
- type 属性を指定した cfparam タグでは、値の有効性がテストされます。値が有効な場合は、何も実行されません。無効な場合は、ColdFusion の式の例外が発生します。

多くの場合、どちらの方法を使用しても同じ目的が達成できます。どちらを選択するかは、コーディングやプログラミングのスタイルによります。また、エラーが発生した場合に表示する情報にもよります。

例 : IsValid 関数による検証

次の例では、ユーザーが数値の ID、有効な電子メールアドレス、および有効な電話番号を送信したかどうかを確認します。送信した値のいずれかが検証テストの基準を満たしていない場合は、ページにエラーメッセージを表示します。

```
<!--- Action code. First make sure the form was submitted. --->
<cfif isDefined("form.saveSubmit")>
    <cfif isValid("integer", form.UserID) and isValid("email", form.emailAddr)
            and isValid("telephone", form.phoneNo)>
        <cfoutput>
            <!--- Application code to update the database goes here --->
            <h3>The e-mail address and phone number for user #Form.UserID#
                have been added</h3>
        </cfoutput>
    <cfelse>
        <H3>Please enter a valid user ID, phone number, and e-mail address.</H2>
    </cfif>
    <cfelse>
</cfif>

<!--- The form. --->
<cfform action="#CGI.SCRIPT_NAME#">
    User ID:<cfinput type="Text" name="UserID"><br>
    Phone: <cfinput type="Text" name="phoneNo"><br>
    E-mail: <cfinput type="Text" name="emailAddr"><br>
    <cfinput type="submit" name="saveSubmit" value="Save Data"><br>
</cfform>
```

例 : cfparam タグによる検証

次に示す 2 つの例では、cfparam タグを使用して、748 ページの「例 : IsValid 関数による検証」と同じテストを行います。この例では、ユーザーが数値の ID、有効な電子メールアドレス、および有効な電話番号を送信したかどうかを確認します。送信した値のいずれかが検証テストの基準を満たしていない場合は、式の例外が発生します。

最初の例では、cferror タグで指定した "exprerr.cfm" ページによってエラーが処理されます。この場合、複数のエラーがあっても、表示されるエラーは 1 つのみです。

2 番目の例では、無効なフィールドは個別の try/catch ブロックで処理されます。この場合は、すべてのエラーに関する情報が表示されます。

エラー処理ページの使用

自己送信フォームとアクションページは次のようになります。

```
<!--- Action part of the page. --->
<!--- If an expression exception occurs, run the expresser.cfm page. --->
<cferror type="EXCEPTION" exception="expression" template="expresserr.cfm">
<!--- Make sure the form was submitted. --->
<cfif isDefined("form.saveSubmit")>
<!--- Use cfparam tags to check the form field data types. --->
        <cfparam name="form.emailAddr" type="email">
        <cfparam name="form.UserID" type="integer">
        <cfparam name="form.phoneNo" type="telephone">
    <!--- Application code to update the database goes here. --->
    <cfoutput>
        <h3>The e-mail address and phone number for user #Form.UserID#
            have been added</h3>
    </cfoutput>
</cfif>

<!--- The form. --->
<cfform action="#CGI.SCRIPT_NAME#">
    User ID:<cfinput type="Text" name="UserID"><br>
    Phone: <cfinput type="Text" name="phoneNo"><br>
    E-mail: <cfinput type="Text" name="emailAddr"><br>
    <cfinput type="submit" name="saveSubmit" value="Save Data"><br>
</cfform>
```

"expresserr.cfm" ページは次のようになります。

```
<cfoutput>
    You entered invalid data.<br>
    Please click the browser Back button and try again<br>
    #cferror.RootCause.detailMessage#
</cfoutput>
```

cftry タグおよび cfcatch タグの使用

自己送信フォームとアクションページは次のようになります。

```
<!--- Use a Boolean variable to indicate whether all fields are good. --->
<cfset goodData="Yes">
<!--- Make sure the form was submitted. --->
<cfif isDefined("form.saveSubmit")>
<!--- The cftry block for testing the User ID value. --->
    <cftry>
<!--- The cfparam tag checks the field data types. --->
        <cfparam name="form.UserID" type="integer">
<!--- If the data is invalid, ColdFusion throws an expression exception. --->
<!--- Catch and handle the exception. --->
        <cfcatch type="expression">
            <!--- Set the data validity indicator to False. --->
            <cfset goodData="No">
            <cfoutput>
                Invalid user ID.<br>
                #cfcatch.detail#<br><br>
            </cfoutput>
        </cfcatch>
    </cftry>
<!--- The cftry block for testing the e-mail address value. --->
    <cftry>
        <cfparam name="form.emailAddr" type="email">
        <cfcatch type="expression">
            <cfset goodData="No">
            <cfoutput>
                Invalid e-mail address.<br>
                #cfcatch.detail#<br><br>
            </cfoutput>
        </cfcatch>
    </cftry>
<!--- The cftry block for testing the telephone number value. --->
    <cftry>
        <cfparam name="form.phoneNo" type="telephone">
        <cfcatch type="expression">
            <cfset goodData="No">
            <cfoutput>
                Invalid telephone number.<br>
                #cfcatch.detail#<br><br>
```

```
            </cfoutput>
        </cfcatch>
    </cftry>
    <!--- Do this only if the validity indicator was not set to False in a
            validation catch block. --->
    <cfif goodData>
        <!--- Application code to update the database goes here. --->
        <cfoutput>
            <h3>The e-mail address and phone number for user #Form.UserID#
                have been added</h3>
        </cfoutput>
    </cfif> <!--- goodData is True--->
</cfif> <!--- Form was submitted. --->

<cfform action="#CGI.SCRIPT_NAME#" preservedata="Yes">
    User ID:<cfinput type="Text" name="UserID"><br>
    Phone: <cfinput type="Text" name="phoneNo"><br>
    E-mail: <cfinput type="Text" name="emailAddr"><br>
    <cfinput type="submit" name="saveSubmit" value="Save Data"><br>
</cfform>
```

Flash フォームの作成

Adobe Flash 形式を使用すれば、効果的なフォームが作成できます。Adobe Flash フォームの表示には、HTML ではなく Flash が使用されます。

Flash フォームについて

ColdFusion では、Flash (SWF ファイル) 形式のフォームをクライアントに提供できます。作成した CFML コードから Flash バイナリが自動的に生成されて、クライアントに表示されます。Flash フォームには、HTML フォームと比べて次のような利点があります。

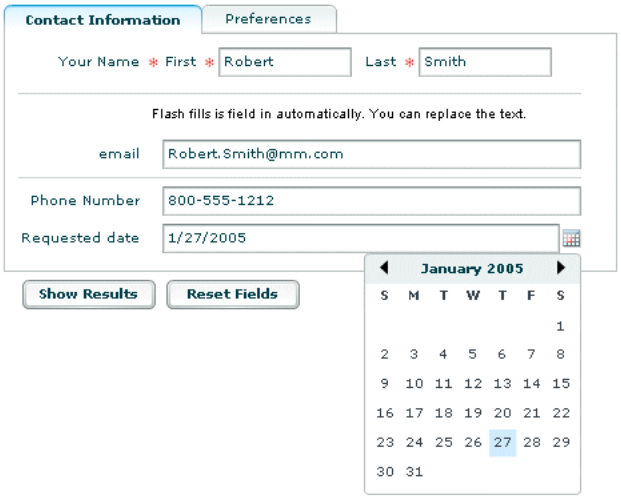
- Flash フォームはブラウザに依存しません。Flash Player は、Windows および Macintosh のすべての主要ブラウザや、Linux の Netscape および Mozilla で動作します。
- 視覚効果の高い洗練された機能をデフォルトで利用できます。また、定義済みのカラースキンを適用したり、CSS (Cascading Style Sheet) に似た方法で外観をカスタマイズしたりすることができます。
- Flash フォームでは、タブ付きナビゲータや、アコーディオンスタイルのダイアログボックスを使用できます。これによって、複数のページを使用することなく、複数の部分からなる複雑なフォームを開発できます。
- さまざまなレイアウト処理が自動的に実行されます。

  注意：Flash フォームのシステム必要条件は、ColdFusion の必要条件とは異なります。たとえば、ColdFusion でサポートされている J2EE サーバーの一部で、Flash フォームが動作しない場合があります。詳細については、『ColdFusion インストール』を参照してください。

Flash フォームの作成に加えて、cfcalendar、cftree、および cfgrid タグで Flash 形式を指定することができます。これらのタグを使用すれば、Flash 形式の日付選択カレンダー、ツリー、およびグリッドを HTML フォームに埋め込むことができるので、Java アプレットを使用する必要がなくなります。ここでは、HTML フォーム内での Flash グリッドおよびツリーの使用方法については説明していませんが、グリッドおよびツリーに関する説明は、それらの要素にも適用されます。

Flash フォームの例

Flash フォームに用意されているさまざまな機能を使用すれば、使いやすく見た目のよい複合フォームをすばやく作成できます。次の図に、多数の機能を使用した、2 つのタブを持つ Flash フォームを示します。



このフォームには次の機能が含まれています。

- 各タブは、フォーム全体を構成する独立したセクションであり、フォームを送信する前に両方のタブでデータを入力できます。この方法を使用すれば、複数の HTML ページにわたって複数のフォームを表示する必要がなくなります。
- 赤色のアスタリスクが付いている姓フィールドおよび名フィールドは必須フィールドです。
- 名前フィールドに入力すると、そのデータが電子メールフィールドに自動的に反映されます。ユーザーはこの情報を上書きできます。
- ユーザーが日付フィールドを選択すると、日付を選択するためのカレンダーが自動的に開きます。

Flash フォームと HTML フォームでの CFML の違い

Flash フォームは SWF ファイル形式でクライアントに送信されるので、Flash フォーム内のすべての要素は Flash によってレンダリングされます。フォームを Flash でレンダリングするためには、次の点に注意する必要があります。

- Flash フォーム本文内のプレーンテキストおよび HTML タグは無視されます。
- すべてのフォームコンテンツは、Flash フォームをサポートする CFML タグの中で指定します。
- ColdFusion には、Flash の機能を利用し HTML と同じタスクを Flash で実現するためのタグが 2 つ用意されています。フォームにテキストブロックおよび水平線や垂直線を追加するには、cfformitem タグを使用します。フォームを編成するには、cfformgroup タグを使用します。
- 標準の ColdFusion フォームタグ (cfinput や cftree など) には、Flash フォームでのみ機能する属性が含まれています。この属性を設定することで、フォームのスタイルや動作を指定することができます。たとえば、skin 属性や、Flash 固有のさまざまな style 属性値を指定して、外観を設定することができます。また bind 属性を使用して、フィールドに他のフィールドのデータ値を適用することができます。

  フォームタグとその機能については、『CFML リファレンス』の各タグのページを参照してください。それらのページでは、Flash フォームで機能する属性や値が示されています。

Flash フォームの構築

Flash フォームの構築には、標準の ColdFusion フォームタグに加えて、cfformgroup タグや cfformitem タグを使用します。これらのタグを使用して、次のようなフォーム要素を作成します。

- cfcalendar、cfgrid、cfinput、cfselect、cftextarea、および cftree タグを使用して、データを表示および入力するためのコントロールを作成します。
- cfformitem タグを使用して、HTML と同じように、テキスト (形式設定も可)、スペーサー、水平線、垂直線を挿入できます。
- cfformgroup タグを使用して、フォームコンテンツをグループ化し格納するコンテナを作成できます。水平方向に整列するボックスや、タブ付きナビゲータなどがあります。

Flash フォームでは、コンテナおよび子による階層構造が形成されます。

1. cfform タグがマスタコンテナであり、その内容が子のコンテナおよびコントロールになります。
2. cfformgroup タグによって、子要素を格納するコンテナが定義されます。
3. その他のすべてのタグは、個別のコントロール (水平線などの表示要素) を作成します。

   たとえば、Flash フォームについての図に示した Flash フォームでは、コンテナおよび子による次のような階層構造が形成されています。このアウトラインでは、図に表示されているページの構造のみを示します。[Preferences] タブの構造は省略します。

```
1   cfform
2       cfformgroup type="tabnavigator" -- Tab navigator container
3           cfformgroup type="page" -- Tabbed page container, child of tabnavigator
4               cfformgroup type="horizontal" -- Aligns its two children horizontally
5                   cfinput type="text" -- First name input control
5                   cfinput type="spacer" -- Space between the name input controls
5                   cfinput type="text" -- Last name input control
4               cfformitem type="hrule" -- Displays a rule
4               cfformitem type="html" -- Displays formatted text
4               cffinput type="text" -- E-mail input control
4               cfformitem type="hrule" -- Displays a rule
4               cfinput type="text" -- Phone number input control
4               cfinput type="spacer" -- Space between the phone and date controls
4               cfinput type="datefield" -- Date input control
3           cfinput type="page" -- Second tabbed page container for preferences
            .
            .
2       cfformgroup type="horizontal" -- Follows the tabnavigator in the form
3       cfinput type="submit" -- Submit button control
3       cfinput type="reset" -- Reset button control
```

cfformitem タグによるテキスト、イメージ、罫線、およびスペースの追加

Flash フォームではインライン HTML がサポートされていないので、フォームにテキストブロック、水平線、垂直線を追加するには cfformitem タグを使用します (cfinput などの Flash フォームコントロールでは、label または value 属性を使用してテキストラベルを指定できます)。また、cfformitem タグを使用して、フォーム要素間にスペーサーを挿入することもできます。

cfformitem type="hrule" および cfformitem type="vrule" タグを使用すると、水平線および垂直線をフォームに挿入できます。罫線の太さ、色、影などの特性は、スタイル仕様で指定できます。罫線のスタイルの指定の詳細については、『CFML リファレンス』の ColdFusion Flash Form Style Reference の Styles for cfformitem with hrule or vrule type attributes を参照してください。

cfformitem type="spacer" タグを使用すると、指定の高さおよび幅の空白をフォームに挿入できます。このタグタイプではスタイルを使用しません。空白は、フォームレイアウトを調整して、フォームの外観を向上させるのに役立ちます。

cfformitem type="text" タグを使用すると、プレーンテキストをフォームに挿入できます。style 属性を使用すれば、テキストに一貫した形式を適用できます。

cfformitem type="html" タグを使用すると、形式設定されたテキストおよびイメージをフォームに挿入できます。このタグの本文に、基本的な HTML スタイルの形式設定タグを含めることで、イメージを追加したり、テキストの形式やスタイルを設定したりすることができます。

cfformitem type="html" タグの本文では、次の形式設定タグと属性を使用できます。

タグ	有効な属性
a	href: リンク先の URL。 target: ウィンドウ名。_blank などの標準の HTML ウィンドウ名を設定できます。
b	なし
br	なし
font	color: #FF00AA のように 16 進数形式で指定する必要があります。# 記号を 1 つ使用します。 face: フォントフェイス名のカンマ区切りリストを設定できます。クライアントシステムで利用可能な最初のフォントが使用されます。 size: ピクセル単位で指定します。+ および - の相対値を使用できます。
i	なし
img	img タグの属性テーブルを参照してください。 メモ : このタグは、/> で閉じるか、終了タグ </img> で閉じる必要があります。
li	なし
p	align。値は left、right、center のいずれかであることが必要です。
textformat	textformat タグの属性テーブルを参照してください。
u	なし

img タグでは、次の属性がサポートされています。

属性	説明
src	(必須) JPEG または SWF ファイルへの URL またはパス。イメージは、完全にダウンロードされるまで表示されません。Flash Player では、プログレッシブ JPEG ファイルはサポートされていません。
width	イメージの幅 (ピクセル単位)。
height	イメージの高さ (ピクセル単位)。
align	テキストフィールド内でのイメージの水平方向の配置。有効な値は、left と right です。デフォルト値は left です。
hspace	イメージの周囲に配置する、テキストを表示しない水平方向のスペース (ピクセル単位)。デフォルト値は 8 です。
vspace	イメージの周囲に配置する、テキストを表示しない垂直方向のスペース (ピクセル単位)。デフォルト値は 8 です。

注意：Flash にはダイナミックなサイズ変更ルールが存在するので、イメージを適切に表示するには、formitem タグの height 属性を指定し、フォームや内包する cfformgroup タグの width および height 属性を指定することが必要になる場合があります。またイメージは常に、テキスト内ではなく、新しい行に表示されます。コード内でイメージの後にテキストが続いている場合は、イメージの右側にある利用可能なスペースに、そのテキストが表示されます。

textformat タグは Flash に固有のタグであり、次の属性を持ちます。

属性	説明
blockindent	ブロックインデント (ポイント単位)
indent	左余白から段落内の最初の文字までのインデント
leading	行間のレディング (垂直方向のスペース)
leftmargin	段落の左余白 (ポイント単位)
rightmargin	段落の右余白 (ポイント単位)
tabstops	正の整数の配列としてのカスタムタブストップ。テキストでタブを指定するには、¥t エスケープ文字を使用します。

これらのタグの詳細については、Flash のマニュアルを参照してください。

次のコードでは、水平線で囲まれた形式設定テキスト領域を表示する、単純な Flash フォームを作成します。

```
<cfform name="myform" height="220" width="400" format="Flash" >
    <!--- Use text formitem tag with style specifications for the heading. --->
    <cfformitem type="text" style="fontWeight:bold; fontSize:14;">
    Flash form with formatted text and rules
    </cfformitem>
    <!--- The spacer adds space between the text and the rule --->
    <cfformitem type="spacer" height="2" />
    <cfformitem type="hrule" />
    <cfformitem type="html">
        <b><font color="#FF0000" size="+4" face="serif">
        This form has formatted text, including:</font></b><br>
        <textformat blockindent="20" leading="2">
            <li>colored text</li>
            <li><i>italic and bold text</i></li>
            <li>a bulleted list in an indented block</li>
        </textformat>
        <p><b>The text is preceded and followed by horizontal rules</b></p>
        It also has a link to a web page.</b><br>
        <a href="http://www.adobe.com/" target="_blank">
        <font color="#0000FF"><u>
        This link displays the Adobe home page in a new browser window
        </u></font></a>
    </cfformitem>
    <cfformitem type="spacer" height="2"/>
    <cfformitem type="hrule"/>
</cfform>
```

cfformgroup タグによるフォームの構築

ColdFusion には、さまざまなタイプのフォームグループコンテナが用意されています。これらを使用して、Flash フォームの基本的な構造を規定します。コンテナのタイプは、cfformgroup タグの type 属性で指定します。次のコンテナタイプを使用して、コントロールのレイアウトやグループを制御します。

タイプ	説明
horizontal	個別のコントロールを水平方向に配置します。コントロールの左にラベルを適用することもできます。cfformitem コントロールなどの、ColdFusion フォームコントロールを配置する場合にのみ使用します。cfformgroup コンテナの配置には使用しないでください。この場合は、代わりに hbox 属性を使用します。 他の cfformgroup タグを horizontal フォームグループ内に配置した場合、含めた cfformgroup タグ内のコントロールが、horizontal グループ内の他のコントロールに合わせて配置されることはありません。
vertical	個別のコントロールを垂直方向に配置します。コントロールの左 (上ではない) にラベルを適用することもできます。cfformitem コントロールなどの、ColdFusion フォームコントロールのグループに対してのみ使用します。cfformgroup コンテナの配置には使用しないでください。この場合は、代わりに vbox 属性を使用します。 他の cfformgroup タグを vertical フォームグループ内に配置した場合、含めた cfformgroup タグ内のコントロールが、vertical グループ内の他のコントロールに合わせて配置されることはありません。
hbox	水平方向にコントロールのグループを配置します。ラベルは適用しません。他の cfformgroup コンテナを配置する場合にこの属性値を使用します。このタグでは、ラベルを持つ子コントロールは配置されません。したがって、個別のコントロールを配置する場合にはこのタグを使用しないでください。
vbox	垂直方向にコントロールのグループを配置します。ラベルは適用しません。他の cfformgroup コンテナを配置する場合にこの属性値を使用します。このタグでは、ラベルを持つ子コントロールは配置されません。したがって、個別のコントロールを配置する場合にはこのタグを使用しないでください。
hdividedbox	複数の子を水平方向に配置し、子の間に区切りハンドルを設定します。ユーザーは、このハンドルをドラッグして、子の相対サイズを変更できます。ラベルは適用しません。hdividedbox コンテナの直接の子は、horizontal または vertical 以外の type 属性を持つ cfformgroup タグである必要があります。
vdividedbox	複数の子を垂直方向に配置し、子の間に区切りハンドルを設定します。ユーザーは、このハンドルをドラッグして、子の相対サイズを変更できます。ラベルは適用しません。vdividedbox コンテナの直接の子は、horizontal または vertical 以外の type 属性を持つ cfformgroup タグである必要があります。
tile	子を行内の最初から順番に長方形のグリッドに配置します。ラベルは適用しません。
panel	label 属性のテキストを含むタイトルバー、境界線、および垂直方向に配置された子を持つコンテンツ領域で構成されます。
accordion	子の各ページを、ラベルバーが設定されたアコーディオンプリーツに配置します。一度に 1 つのプリーツの内容を表示します。ユーザーはラベルをクリックして、プリーツページを展開したり、折り畳んだりすることができます。ラベルは適用しません。
tabnavigator	それぞれの子をタブ付きページに配置します。ユーザーはタブをクリックして、選択したページを表示します。ラベルは適用しません。
page	accordion または tabnavigator コンテナの直下の子。プリーツバーまたはタブ上のラベルを指定し、その子コンテナおよびコントロールを垂直方向に配置します。

cfformgroup のタイプである accordion、tabnavigator、および page の使用方法の詳細については、759 ページの「アコーディオンおよびタブナビゲータコンテナを使用した複合フォームの作成」を参照してください。

例 : cfformgroup タグを使用した構築

次の例では、2 つの vbox コンテナが hdividedbox コンテナの中に含まれているフォームを作成します。左のボックスでは、horizontal コンテナを使用して、2 つのラジオボタンを配置します。右のボックスでは、tile コンテナを使用して、チェックボックスをレイアウトします。区切りハンドルをドラッグして、2 つのボックスのサイズを変更することができます。
フォームを送信すると、Form スコープがダンプされて、送信されたデータが表示されます。

```
<cfif Isdefined("Form.fieldnames")>
<cfdump var="#form#" label="form scope">
<br><br>
</cfif>

<cfform name="myform" height="200" width="460" format="Flash" skin="HaloBlue">
    <cfformitem type="html" height="20">
        <b>Tell us your preferences</b>
    </cfformitem>
    <!--- Put the pet selectors to the left of the fruit selectors. --->
    <cfformgroup type="hdividedbox" >
    <!--- Group the pet selector box contents, aligned vertically. --->
        <cfformgroup type="VBox"height="130">
            <cfformitem type="text" height="20">
                Pets:
            </cfformitem>
            <cfformgroup type="vertical" height="80">
                <cfinput type="Radio" name="pets" label="Dogs" value="Dogs"
                    checked>
                <cfinput type="Radio" name="pets" label="Cats" value="Cats">
            </cfformgroup>
        </cfformgroup>

    <!--- Group the fruit selector box contents, aligned vertically. --->
        <cfformgroup type="VBox" height="130">
            <cfformitem type="text" height="20">
                Fruits:
            </cfformitem>
            <cfformgroup type="tile" height="80" width="190" label="Tile box">
                <--- Flash requires unique names for all controls --->
                <cfinput type = "Checkbox" name="chk1" Label="Apples"
                    value="Apples">
                <cfinput type="Checkbox" name="chk2" Label="Bananas"
                    value="Bananas">
                <cfinput type="Checkbox" name="chk3" Label="Pears"
                    value="Pears">
                <cfinput type="Checkbox" name="chk4" Label="Oranges"
                value="Oranges">
                <cfinput type="Checkbox" name="chk5" Label="Grapes"
                    value="Grapes">
                <cfinput type="Checkbox" name="chk6" Label="Cumquats"
                    value="Cumquats">
            </cfformgroup>
        </cfformgroup>
    </cfformgroup>
    <cfformgroup type="horizontal">
        <cfinput type="submit" name="submit" width="100" value="Show Results">
        <cfinput type="reset" name="reset" width="100" value="Reset Fields">
    </cfformgroup>
</cfform>
```

Flash フォームでのサイズの制御

Flash フォームの要素のサイズ設定は、機能面よりも、視覚面に大きな影響を与えます。height および width 属性を指定しなかった場合は、Flash によって適切なレイアウトが実行されます。ただし、次の点に注意する必要があります。

- cfform タグで height および width 属性を指定しなかった場合は、表示されるブラウザウィンドウ全体 (フォームがテーブル内に配置されていない場合)、またはテーブルセル全体 (フォームがテーブル内に配置されている場合) がそのフォームのために予約されます。これは、フォームコンテンツを表示するためにその大きさが必要ない場合でも同様です。したがって、このフォームの前後に HTML 出力があると、出力ページがブラウザウィンドウのサイズを超えます。

- コントロール (フォームグループを含む) の高さまたは幅を指定しなかった場合は、利用可能なスペースにコントロールが収まるように大きさが調整されます。たとえば、大きさが指定されていない入力ボックスは、多くの場合、それを格納しているコントロールの幅に合わせて拡張されます。

一般に、Flash フォームを設計するときは、次のプロセスを行うことをお勧めします。

Flash フォームとコントロールのサイズの決定

1. 最初にフォームを作成するときは、フォームまたはその子タグで height および width 属性を指定しないでください。フォームを実行し、結果を検証して、使用する高さおよび幅の値を決定します。
2. cfform タグの height および width 属性を指定して、フォームの大きさを設定します。絶対値 (ピクセル単位) またはパーセント値を指定できます。パーセント値は、フォームを格納しているウィンドウのサイズに対する相対値です。
3. 個別のタグに height または width 属性を指定します。この値はピクセル単位である必要があります。
4. 適切な外観になるまで、さまざまなタグに対して手順 3 を繰り返します。必要に応じて手順 2 も行います。

クエリーデータに基づく Flash フォーム要素の繰り返し

cfformgroup の repeater タイプを使用すれば、cfformgroup タグの一連の子コントロールを、クエリーの各行に基づいて作成することができます。子コントロールの各セットでは、子タグの bind 属性を使用して、現在のクエリー行のフィールドにアクセスできます。この cfformgroup タイプを使用すると、ColdFusion で Flash SWF ファイルを再コンパイルしなくても、クエリーに基づいてコントロールの数が変化する Flash フォームを作成できます。この結果、サーバーのパフォーマンスが大幅に向上します。

注意：データのバインドの詳細については、761 ページの「Flash フォームでのデータのバインド」を参照してください。

repeater で使用する開始行と最大行数を、オプションで指定できます。ほとんどの ColdFusion タグとは異なり、repeater のインデックス値は 1 ではなく 0 から始まります。たとえば、クエリーオブジェクトの最初の行から開始し、最大 15 行まで使用する repeater を指定するには、次のようなタグを使用します。

```
<cfformgroup type="repeater" query="q1" startrow="0" maxrows="15">
```

リピーターを使用する例としては、教師が特定のクラスを選択して、生徒の評価を更新できるフォームが挙げられます。各クラスの生徒数は同じとは限りません。したがって、このフォームでは、生徒数に応じて入力行数を変化させる必要があります。別の例としては、注文する商品の名前と数を表示し、ユーザーが注文数を変更できるショッピングカートアプリケーションが挙げられます。

次の例では、repeater の type 属性値を持つ cfformgroup タグを使用して、フォームを作成します。この例では、クエリーを作成し、repeater を使用してクエリーの各行に対応する氏名入力ボックスを作成します。

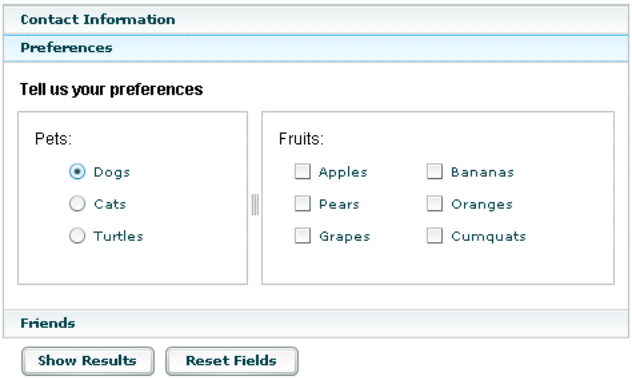
```
<cfif IsDefined("Form.fieldnames")>
    <cfdump var="#form#" label="form scope">
    <br><br>
</cfif>
<cfscript>
    q1 = queryNew("id,firstname,lastname");
    queryAddRow(q1);
    querySetCell(q1, "id", "1");
    querySetCell(q1, "firstname", "Rob");
    querySetCell(q1, "lastname", "Smith");
    queryAddRow(q1);
    querySetCell(q1, "id", "2");
    querySetCell(q1, "firstname", "John");
    querySetCell(q1, "lastname", "Doe");
    queryAddRow(q1);
    querySetCell(q1, "id", "3");
    querySetCell(q1, "firstname", "Jane");
    querySetCell(q1, "lastname", "Doe");
    queryAddRow(q1);
    querySetCell(q1, "id", "4");
    querySetCell(q1, "firstname", "Erik");
    querySetCell(q1, "lastname", "Pramenter");
</cfscript>

<cfform name="form1" format="flash" height="220" width="450">
    <cfselect label="select a teacher" name="sel1" query="q1" value="id"
        display="firstname" width="100" />
    <cfformgroup type="repeater" query="q1">
        <cfformgroup type="horizontal" label="name">
            <cfinput type="Text" name="fname"bind="{q1.currentItem.firstname}">
            <cfinput type="Text" name="lname" bind="{q1.currentItem.lastname}">
        </cfformgroup>
    </cfformgroup>
    <cfinput type="submit" name="submitBtn" value="Send Data" width="100">
</cfform>
```

アコーディオンおよびタブナビゲータコンテナを使用した複合フォームの作成

accordion および tabnavigator タイプの cfformgroup タグを使用すれば、通常は複数の HTML ページで構成される複数のフォームを、1 つの複合フォームとして構築することができます。アコーディオンおよびタブナビゲータコンテナでは、中間フォームを送信することなく、複数のエントリ領域を切り替えることができます。入力したすべてのデータはフォームを送信するまで操作でき、すべてのフォーム要素は一度にロードされます。

アコーディオンコンテナでは、それぞれの論理フォームページが個別のアコーディオンプリーツに配置されます。各プリーツにはラベルバーがあります。ユーザーがこのバーをクリックすると、現在のページが折り畳まれ、選択したページが展開されて、利用可能なフォームスペースが表示されます。次の図に、3 つのプリーツを持つアコーディオンを示します。ここでは、中央の [Preferences] プリーツが開いています。



タブナビゲータコンテナでは、それぞれの論理フォームページが個別のタブ付きフレームに配置されます。ユーザーがタブをクリックすると、以前のページに代わって選択したページが表示されます。Flash フォームについての図は、タブナビゲータコンテナを示しています。

次の例では、ユーザーの連絡先情報と好きなものを設定する 2 つのタブを持つタブナビゲータコンテナを生成します。最初の cfformgroup タグの type 属性を、tabnavigator から accordion に変更すれば、タブナビゲータをアコーディオンコンテナに変更できます。アコーディオンコンテナにスクロールバーが表示されないようにするには、cfform タグの height 属性の値を 310 に増やし、tabnavigator タグの height 属性を 260 に増やす必要があります。

```
<cfif IsDefined("Form.fieldnames")>
    <cfdump var="#form#" label="form scope">
    <br><br>
</cfif>
<br>
<cfform name="myform" height="285" width="480" format="Flash" skin="HaloBlue">
    <cfformgroup type="tabnavigator" height="240" style="marginTop: 0">
        <cfformgroup type="page" label="Contact Information">
            <!--- Align the first and last name fields horizontally. --->
            <cfformgroup type="horizontal" label="Your Name">
                <cfinput type="text" required="Yes" name="firstName" label="First"
                    value="" width="100"/>
                <cfinput type="text" required="Yes" name="lastName" label="Last"
                    value="" width="100"/>
            </cfformgroup>
            <cfformitem type="hrule" />
            <cfformitem type="HTML"><textformat indent="95"><font size="-2">
                Flash fills this field in automatically.
                You can replace the text.
                </font></textformat>
            </cfformitem>
            <!--- The bind attribute gets the field contents from the firstName
                    and lastName fields as they get filled in. --->
            <cfinput type="text" name="email" label="email"
                bind="{firstName.text}.{lastName.text}@mm.com">
            <cfformitem type="spacer" height="3" />
            <cfformitem type="hrule" />
            <cfformitem type="spacer" height="3" />

            <cfinput type="text" name="phone" validate="telephone" required="no"
                label="Phone Number">
```

```
            <cfinput type="datefield" name="mydate1" label="Requested date">
        </cfformgroup>

        <cfformgroup type="page" label="Preferences" style="marginTop: 0">
            <cfformitem type="html" height="20">
                <b>Tell us your preferences</b>
            </cfformitem>
            <!--- Put the pet selectors to the left of the fruit selectors. --->
            <cfformgroup type="hdividedbox" >
            <!--- Group the pet selector box contents, aligned vertically. --->
                <cfformgroup type="VBox"height="130">
                    <cfformitem type="text" height="20">
                        Pets:
                    </cfformitem>
                    <cfformgroup type="vertical" height="80">
                        <cfinput type="Radio" name="pets" label="Dogs" value="Dogs"
                            checked>
                        <cfinput type="Radio" name="pets" label="Cats" value="Cats">
                    </cfformgroup>
                </cfformgroup>

            <!--- Group the fruit selector box contents, aligned vertically. --->
                <cfformgroup type="VBox" height="130">
                    <cfformitem type="text" height="20">
                        Fruits:
                    </cfformitem>
                    <cfformgroup type="tile" height="80" width="190" label="Tile box">
                        <--- Flash requires unique names for all controls. --->
                        <cfinput type = "Checkbox" name="chk1" Label="Apples"
                            value="Apples">
                        <cfinput type="Checkbox" name="chk2" Label="Bananas"
                            value="Bananas">
                        <cfinput type="Checkbox" name="chk3" Label="Pears"
                            value="Pears">
                        <cfinput type="Checkbox" name="chk4" Label="Oranges"
                            value="Oranges">
                        <cfinput type="Checkbox" name="chk5" Label="Grapes"
                            value="Grapes">
                        <cfinput type="Checkbox" name="chk6" Label="Kumquats"
                            value="Cumquats">
                    </cfformgroup>
                </cfformgroup>
            </cfformgroup>
        </cfformgroup>

        <cfformgroup type="horizontal">
            <cfinput type = "submit" name="submit" width="100" value = "Show Results">
            <cfinput type = "reset" name="reset" width="100" value = "Reset Fields">
        </cfformgroup>
    </cfform>
```

Flash フォームでのデータのバインド

bind 属性を使用すれば、あるフィールドの値に、別のフォームフィールドの内容を含めることができます。bind 属性は、cftextarea タグや、値を取る任意の cfinput タイプ (hidden を含む) で使用できます。このデータバインディングは、ユーザーがクライアントシステムで Flash 内にデータを入力したときにダイナミックに実行されます。ユーザーがフォームを送信するまで、Flash から ColdFusion に情報が送信されることはありません。bind 属性を使用してフィールドの値を指定するには、次の形式を使用します。

データソース	bind 属性の形式
cfinputtype="text" または cftextarea のテキスト	bind="{**sourceName**.text}"
cfinput で選択されたラジオボタン	bind="{**sourceName**.selectedData}"
cftree で選択された項目	bind="{**sourceName**.selectedNode.getProperty('data').value}
cfgrid で選択された項目	bind="{**sourceName**.selectedItem.**COLUMNAME**}"
cfselect で選択された項目	bind="{**sourceName**.selectedItem.data}"

注意：bind 属性を使用する場合は、value 属性を使用できません。

バインディング形式には、次のルールやテクニックが適用されます。

- これらの形式の **sourceName** の値には、バインドする要素が含まれているタグの name 属性を指定します。
- ツリーで選択された項目に関する追加情報をバインドすることができます。表示値を取得するには、value の代わりに display を使用します。ツリー内のノードへのパスを取得するには path を使用します。
- cfselect 項目の表示値をバインドするには、data の代わりに label を使用します。
- ユーザーが cfselect コントロールで複数の項目を選択した場合、selectedItem オブジェクトには最後に選択された項目が含まれますが、selectedItems 配列には選択されたすべての項目が含まれます。myTree.selectedItems[1].data のように指定して、配列内の個別の値にアクセスすることができます。selectedItems 配列は、ユーザーが複数の項目を選択した場合にのみ存在します。それ以外の場合は定義されません。
- Flash bind ステートメントで ActionScript 式を使用できます。

  次の例では、firstName および lastName フィールドの値を使用して電子メールアドレスを作成します。ユーザーは、この値を変更したり、エントリを入力して置換したりすることができます。

```
<cfformgroup type="horizontal" label="Your Name">
    <cfinput type="text" required="Yes" name="firstName" label="First"
        value="" width="100"/>
    <cfinput type="text" required="Yes" name="lastName" label="Last"
        value="" width="100"/>
</cfformgroup>
<cfinput type="text" name="email" label="email"
    bind="{firstName.text}.{lastName.text}@mm.com">
```

Flash フォームのスタイルおよびスキンの設定

ColdFusion では、次の方法を使用して、Flash フォームやその要素のスタイルおよび外観を設定できます。

スキン フォームの全体的な外観を簡単に設定できます。1 つのスキンで、フォーム全体を設定します。

スタイル スキンに比べて、細かく外観を設定できます。1 つのスタイルで、単一のコントロールの外観を指定します。多くのスタイルは、コントロールの子にも継承されます。

これらの方法は、組み合わせて使用できます。フォームに対してスキンを指定し、スタイルを使用して個別のコントロールの外観 (入力テキストのフォントなど) を指定できます。

使用可能なスタイルの名前および値の詳細については、『CFML リファレンス』の ColdFusion Flash Form Style Reference を参照してください。

Flash スキンによるフォームの外観の設定

cfform タグの skin 属性を使用して、フォームの全体的な外観を選択できます。スキンによって、強調表示された要素や、選択された要素の表示色が決まります。

次に示す Flash スキンを選択できます。

- haloBlue
- haloGreen (デフォルト)
- haloOrange
- haloSilver

Flash フォームのスタイルについて

ColdFusion Flash フォームタグの style 属性を使用すれば、コントロールの特性を CSS のシンタックスによって指定できます。次のタグで style 属性を指定できます。

- cfform
- cfformgroup
- cfcalendar
- cfformitem の hrule および vrule タイプ
- cfgrid
- cfinput
- cfselect
- cftextarea
- cftree

cfform および cfformgroup の属性は、一般にフォームまたはフォームグループのすべての子に適用されます。

Flash では、標準的な CSS スタイルが多数サポートされています (すべてサポートされているわけではありません)。ColdFusion Flash フォームでは、特定の CSS スタイル仕様を、個別の CFML タグおよび対応する Flash コントロールやグループに適用することのみが可能です。HTML フォームのように、外部スタイルシートを使用したり、ドキュメントレベルのスタイルシートを定義したりすることはできません。

Flash フォームのスタイル仕様のシンタックス

Flash のスタイルを指定するには、次の形式を使用します。

```
style="stylename1: value; stylename2: value; ..."
```

たとえば、次のコードでは、テキスト入力コントロールに 3 つのスタイル値を指定しています。

```
<cfinput type="text" name="text2" label="Last"
    style="borderSyle:inset; fontSize:12; backgroundColor:##FFEEFF">
```

Flash フォームのスタイル値の形式について

スタイルのプロパティには、ブール値、文字列、数値、またはこれらの値の配列を指定できます。

長さの形式

長さや大きさの値を取るスタイル (フォントサイズなど) は、ピクセル単位で指定します。

fontSize スタイルプロパティでは、数値に加えて一連のキーワードも使用できます。fontSize スタイルプロパティの設定では、次のキーワードを使用できます。正確なサイズは、クライアントブラウザによって異なります。

- xx-small
- x-small

- small
- medium
- large
- x-large
- xx-large

次の cfinput タグでは、style 属性に fontSize キーワードを使用して、入力ボックスのテキストサイズを指定しています。

```
<cfinput type="text" name="text1" style="fontSize:X-large" label="Name">
```

時間の形式

時間の値を取るスタイル (たとえば、アコーディオンプリーツの開く速度を指定する openDuration スタイルなど) は、ミリ秒単位で指定します。次の例に示す accordion タグは、アコーディオンプリーツの切り替えに 0.5 秒かかります。

```
<cfformgroup type="accordion" height="260" style="openDuration: 500">
```

色の形式

色の値 (backgroundColor スタイルの値など) は、次の形式で指定します。

形式	説明
16 進数	16 進数で色を表す場合は、2 つのシャープ記号 (##) の後に 6 桁のコードを指定します。# 記号を 2 つ重ねることにより、ColdFusion でこの記号が解釈されないようにします。使用できる値の範囲は、##000000 から ##FFFFFF です。
VGA カラー名	VGA カラー名は、CSS がサポートされているすべてのブラウザで使用できる、16 色の基本カラーのセットです。利用可能なカラー名は、Aqua、Black、Blue、Fuchsia、Gray、Green、Lime、Maroon、Navy、Olive、Purple、Red、Silver、Teal、White、Yellow です。一部のブラウザでは、より多くのカラー名がサポートされています。VGA カラー名では、大文字と小文字は区別されません。

16 進数の色のみがサポートされているスタイルもいくつかあります。

コントロールによっては、複数の色を指定できる場合があります。たとえば、ツリーコントロールの depthColors スタイルプロパティでは、ツリー内の各レベルに対して異なる背景色を使用できます。複数の色を割り当てるには、次の例のように、カンマ区切りリストを使用します。

```
style="depthColors: ##EAEAEA, ##FF22CC, ##FFFFFF"
```

Flash フォームのスタイルの適用範囲と継承について

共通スタイルの中には、指定可能だが何も効果がないタグがいくつか存在しているものもあります。これは、Flash コントロールのスタイルの実装方法に起因します。したがって、『CFML リファレンス』の ColdFusion Flash Form Style Reference の Styles valid for all controls の表に記載されているスタイルは、コントロールで使用してもエラーにはならないが何も効果がない場合があります。

スタイルには、継承できるものと継承できないものがあります。継承できないスタイルは、そのタグにのみ適用され、その子には適用されません。たとえば、hbox フォームグループとその子タグに同じ背景色を適用するには、すべてのタグで色を指定する必要があります。継承できるスタイルは、そのタグだけでなく、子に対しても適用されます。

例 : Flash フォームへのスタイルの適用

次のコードでは、スキンとスタイルを使用して外観を設定する Flash フォームを作成します。

このフォームのコードは次のとおりです。コード内のコメントで、外観に対する形式設定コントロールとスタイルの効果を説明します。

```
<!--- Specify the form height and width, use the HaloBlue skin.
         Note: Flash ignores a backgroundColor style set in cfform. --->
<cfform name="myform" height="390" width="440" format="Flash" skin="HaloBlue">
    <!--- The input area is a panel. Styles to specify panel characteristics.
             Child controls inherit the background color and font settings. --->
    <cfformgroup type="Panel" label="Contact Information"
            style="marginTop:20; marginBottom:20; fontSize:14; fontStyle:italic;
            headerColors:##FFFF00, ##999900; backgroundColor:##FFFFEE;
            headerHeight:35; cornerRadius:12">
        <!--- This vbox sets the font size and style, and spacing between and
                 around its child controls. --->
        <cfformgroup type="vbox" style="fontSize:12; fontStyle:normal;
                verticalGap:18; marginLeft:10; marginRight:10">
            <!--- Use a horizontal group to align the first and last name fields
                     and set a common label. --->
            <cfformgroup type="horizontal" label="Name" >
                <!--- Use text styles to highlight the entered names. --->
                <cfinput type="text" required="Yes" name="firstName" label="First"
                    value="" width="120" style="color:##006090; fontSize:12;
                    fontWeight:bold" />
                <cfinput type="text" required="Yes" name="lastName" label="Last"
                    value="" width="120" style="color:##006090; fontSize:12;
                    fontWeight:bold"/>
            </cfformgroup>
            <!--- Horizontal rules surround the e-mail address.
                     Styles specify the rule characteristics. --->
            <cfformitem type="hrule" style="color:##999900; shadowColor:##DDDD66;
                strokeWidth:4"/>
            <cfformitem type="HTML" Style="marginTop:0; marginBottom:0">
                <textformat indent="57"> <font size="-1">Flash fills this field in
                automatically. You can replace the text.</font></textformat>
            </cfformitem>
            <cfinput type="text" name="email" label="email"
                bind="{firstName.text}.{lastName.text}@mm.com">
            <cfformitem type="hrule" style="color:##999900; shadowColor:##DDDD66;
                strokeWidth:4"/>
            <cfinput type="text" name="phone" validate="telephone" label="Phone">
            <!--- Styles control the colors of the current, selected, and
                rolled-over dates. --->
            <cfinput type="datefield" name="mydate1" label="Date"
                style="rollOverColor:##DDDDFF; selectionColor:##0000FF;
                todayColor:##AAAAFF">
        </cfformgroup> <!--- vbox --->
    </cfformgroup> <!--- panel --->
    <!--- A style centers the buttons at the bottom of the form. --->
    <cfformgroup type="horizontal"style="horizontalAlign:center">
        <cfinput type = "submit" name="submit" width="100" value = "Show Results">
        <cfinput type = "reset" name="reset" width="100" value = "Reset Fields">
    </cfformgroup>
</cfform>
```

Flash フォームでの ActionScript の使用

ActionScript 2.0 は、Flash やその他の関連製品で使用されている、JavaScript に似た強力なスクリプト言語です。Flash フォームでは、ActionScript 2.0 コードのサブセットを使用できます。

ここでは、ActionScript を Flash フォームに挿入する方法、および ColdFusion Flash フォームで ActionScript を使用する場合の制限事項や補足情報を示します。ただし、ActionScript の作成方法については説明しません。ActionScript の詳細および使用方法については、Flash ActionScript 2.0 に関するマニュアルを参照してください。たとえば、www.adobe.com/go/learn_cfu_docs_jp から LiveDocs の Flash セクションおよび Flex セクションにあるドキュメントを参照してください。

CFML での ActionScript コードの使用

ActionScript は、CFML Flash フォームの、次のタグ属性で使用できます。

- フォームイベントおよびコントロールイベント。たとえば、cfform タグの onSubmit 属性、cfinput タグの onChange および onClick 属性などがあります。『CFML リファレンス』の各タグのページで行われている属性に関する説明の中で、これらのイベント属性が列挙されています。
- バインド式。この式を使用してフィールドの値を設定できます。データのバインドの詳細については、761 ページの「Flash フォームでのデータのバインド」を参照してください。

  ActionScript コードをフォーム属性にインラインで含めたり、独自に定義したカスタム関数を呼び出したりすることができます。また、ActionScript の include コマンドを属性で使用して、.as ファイルとして保存されている ActionScript を読み込むこともできます。

  次の例は、華氏から摂氏への単純な変換ツールです。変換はクライアントで直接実行されます。ユーザーが ColdFusion サーバーにフォームを送信する必要はありません。

```
<cfform format="flash" width="200" height="150">
    <cfinput type="text" name="fahrenheit" label="Fahrenheit" width="100"
        value="68">
    <cfinput type="text" name="celsius" label="Celsius" width="100">
    <cfinput type="button" name="convert" value="Convert" width="100"
        onClick="celsius.text = Math.round((farenheit.text-32)/1.8*10)/10">
</cfform>
```

  注意：非表示フィールドにアクセスする場合、text プロパティ (***fieldname***.text) は使用できません。非表示フィールドにアクセスするには、formname.fieldname = 'value' の形式を使用してください。

カスタム ActionScript 関数

カスタム ActionScript 関数は、CFML の UDF と同等の働きをします。ColdFusion で独自の関数を定義するには、cfformitem タグを使用し、type 属性の値を script にします。また、ActionScript (.as) ファイルに独自の関数を定義することもできます。ColdFusion には、Flash フォームのコントロールで使用できるいくつかのカスタム ActionScript 関数があらかじめ定義されています。

すべてのフォームコントロールで、フォームをリセットまたは送信する次のカスタム関数を使用できます。

- resetForm()
- submitForm()

cfgrid タグでは、グリッドの行を挿入および削除する次のカスタム関数を使用できます。

- GridData.insertRow(**gridName**)
- GridData.deleteRow(**gridName**)

次の例では、Flash フォームの行を追加したり削除したりするカスタムボタンを、2 つの GridData 関数を使用して追加します。これらのボタンは、cfgrid タグで insert="yes" および delete="yes" を指定した場合に作成されるボタンと同じですが、これらのボタンでは、独自のボタンテキストおよび配置を指定することができます。この例では、グリッドの下ではなく横にボタンを配置し、標準のボタンラベルよりも長いラベルを使用しています。

```
<cfform format="flash" height="265" width="400">
    <cfformitem type="html">
        You can edit this grid as follows:
        <ul>
        <li>To change an item, click the field and type.</li>
        <li>To add a row, click the Insert Row button and type in the fields
            in the highlighted row.</li>
        <li>To delete a row, click anywhere in the row and click the
            Delete Row button</li>
        </ul>
        <p><b>When you are done, click the submit button.</b></p>
    </cfformitem>
    <!--- The hbox aligns the grid and the button vbox horizontally --->
    <cfformgroup type="hbox" style="verticalAlign:bottom;
            horizontalAlign:center">
        <!--- To make all elements align properly, all of the hbox children must
            be containers, so we must put the cfgrid tag in a vbox tag. --->
        <cfformgroup type="vbox">
            <!-- An editable grid with hard coded data (for simplicity).
                By default, this grid does not have insert or delete buttons. --->
            <cfgrid name="mygrid" height="120" width="250" selectmode="edit">
                <cfgridcolumn name="city">
                <cfgridcolumn name="state">
                <cfgridrow data="Rockville,MD">
                <cfgridrow data="Washington,DC">
                <cfgridrow data="Arlington,VA">
            </cfgrid>
        </cfformgroup>
        <!--- Group the Insert and Delete buttons vertically;
                use a vbox to ensure correct alignment. --->
        <cfformgroup type="vbox" name="buttons"style="verticalAlign:bottom;
                horizontalAlign:center">
            <!--- Use a spacer to position the buttons. --->
            <cfformitem type="spacer" height="18" />
            <!--- Use the insertRow method in the onClick event to add a row. --->
            <cfinput type="button" name="ins" value="Insert a new row" width="125"
                onClick="GridData.insertRow(mygrid);">
            <!--- Use the deleteRow method in the onClick event to delete
                the selected row --->
            <cfinput type="button" name="del" value="Delete selected row"
                    width="125" onClick="GridData.deleteRow(mygrid)">
            <cfinput type="submit" name="f1" value="Submit" width="125">
        </cfformgroup>
    </cfformgroup>
</cfform>

<!--- Dump the form if it has been submitted. --->
<cfif IsDefined("form.fieldnames")>
<cfdump var="#form#"><br>
</cfif>
```

Flash フォームに関する推奨事項

フォームの再コンパイルを最小限に抑える

Flash フォームは、SWF ファイルとしてクライアントに送信されます。SWF ファイルは、ColdFusion によって CFML コードからコンパイルされることで生成されます。次の手法を使用すれば、ColdFusion で Flash フォームが再コンパイルされる頻度を減らすことができます。

- データ以外のものはダイナミックにしないでください。変数名が変更されるたびに、またはフォームの特性 (要素の幅やラベルなど) が変更されるたびに、Flash 出力の再コンパイルが必要になります。データ値が変更されても、出力の再コンパイルは必要になりません。
- 10 以下の要素について 10 回以下のループを実行する場合は、cfformgroup type="repeater" を使用します。このタグを使用すれば、要素の数が変わっても再コンパイルの必要は生じません。ただしこのタグは、ループおよび要素の数が増えるにつれて、処理オーバーヘッドが増大します。したがって、大量のデータセットまたは要素が存在する場合は、一般に repeater を使用しないほうが効率的です。

Flash フォームでのデータのキャッシュ

cfform タグの timeout 属性で、サーバー上に Flash フォームデータを保持する秒数を指定できます。Flash フォームが生成されると、フォームの値がサーバー上のメモリに保管されます。Flash フォームがクライアントにロードされるときに、サーバー上のこれらのフォーム値が要求されます。この属性が 0 (デフォルト) の場合、サーバー上のデータは Flash フォームから要求された後すぐに削除されます。

Flash フォームはリロードされることがあります。たとえば、ユーザーが Flash フォームのページを表示した後に別のページに進み、ブラウザの [戻る] ボタンをクリックして Flash フォームのページに戻ることがあります。これは、検索フォームやログインフォームなどでは一般的な操作です。ユーザーが元のページに戻ったときの動作は次のとおりです。

- timeout 値が 0 の場合、またはタイムアウト時間が経過した場合、そのデータは使用できなくなり、データ期限切れの例外が ColdFusion からブラウザに返されます。ブラウザでは、通常、ページのリロードを促すメッセージが表示されます。
- タイムアウト時間がまだ経過していない場合、ブラウザには元のデータが表示されます。

  クレジットカード番号や社会保障番号などの機密情報がフォームデータに含まれている場合は、タイムアウト値を 0 のままにしてください。0 にできない場合は、小さな値に設定することを検討してください。

クラスタ環境での Flash フォームの使用

Flash フォームをクラスタで使用する場合は、" スティッキー " セッションが必要になります。

スキン可能 XML フォームの作成

ColdFusion では、XML スキン可能フォームを作成できます。XML スキン可能フォームとは、XForms に準拠した XML が生成されるフォームのことで、通常は XSLT (eXtensible Stylesheet Language Transformations) スキンを使用して形式設定します。

XML や XSLT の知識がまったくない場合でも、Adobe ColdFusion に用意されているスキンを利用して、XML スキン可能フォームを使用することができます。ColdFusion での XML の使用方法の詳細については、1028 ページの「XML および WDDX の使用」を参照してください。

XML スキン可能フォームについて

format="XML" 属性が指定された ColdFusion フォームは、**XML** スキン可能フォームです。XML スキン可能フォームが ColdFusion で処理されると、XML 形式のフォームが生成されます。デフォルトでは、この XML に XSLT (XML Stylesheet Language Transform) スキンが適用されて、ユーザーのブラウザに表示するための形式設定された HTML ページが生成されます。必要に応じて、XSLT ファイルを指定することや、ColdFusion ページで XML を処理することもできます。

XML スキン可能フォームを使用すると、ColdFusion で生成および表示されるフォームのタイプと外観を制御できます。ColdFusion には、一連の標準スキンが用意されています。使用するスキンをアプリケーションで指定しなかった場合や、スキンを適用しないことを指定した場合は、標準スキンの 1 つであるデフォルトスキンが使用されます。また、独自の XSLT スキンを作成して ColdFusion で使用されるようにすれば、独自のスタイルや外観をフォームに適用することができます。

ColdFusion フォームと XForms

ColdFusion スキン可能フォームは、W3C XForms の仕様に準拠するとともに、この仕様を拡張しています。この仕様では、フォームの外観に依存しないシンタックスでインタラクティブフォームを定義できる XML シンタックスが定義されています。ColdFusion スキン可能フォームでは、この仕様をサポートすると同時に、XForms モデルでは直接記述できない情報 (外観情報、検証ルール、標準の HTML 属性など) を指定するための属性が ColdFusion のフォームタグに用意されています。ColdFusion スキン可能フォームでは、こうした情報を XForms エクステンションに保持することで、XSL 変換でその値を使用して、外観の決定やその他の処理を実行できるようにしています。

ColdFusion スキン可能フォームの XML 構造の詳細については、773 ページの「ColdFusion XML 形式」を参照してください。

XSLT スキンの役割

XML スキン可能フォームの処理方法や表示方法は、XSLT スキンおよび関連付けられた CSS (Cascading Style Sheet) を使用して制御します。

- XSLT スキンでは、XML の処理方法を制御します。また、多くの場合、XML を変換して表示用の HTML を生成します。スキンでは、出力を形式設定するための CSS スタイルシートを指定します。
- CSS スタイルシートでは、生成出力の外観を決定するスタイル定義を指定します。

XSLT スキンを使用すれば、生成出力を柔軟に制御できます。独自の外観を作成したり、目的に応じて複数の外観を使い分けることもできます。たとえば、イントラネットとインターネットで同じフォームを使用する場合でも、スキンを変更するだけで異なる外観を提供できます (フォームに表示する情報をスキンで選択することもできます)。また、ワイヤレスモバイルデバイスなどのデバイス向けの処理を行うこともできます。

ColdFusion での XML スキン可能フォームの処理

XML 形式と XSLT スキンが指定された cfform タグは、ColdFusion で次のように処理されます。

1. CFML フォームタグが、XForms に準拠した XML テキスト形式に変換され、フォームと同じ名前を持つ変数で参照できるようになります。フォーム内のインラインテキストや HTML タグは無視されます。これらは XML に含められません。ただし、cfselect タグの子である HTML の option タグは処理されます。
2. XML に XSLT スキンが適用されます。これによって、HTML への変換などが行われます。XSLT ファイルでは CSS スタイルシートが指定されています。
3. 生成された、スタイル設定済みのフォームが、クライアント (ユーザーのブラウザなど) に返されます。

cfform タグで skin 属性を設定しなかった場合は、デフォルトのスキンが使用されます。

skin="none" を指定した場合、最初の手順は実行されますが、以降の手順はスキップされます。この場合は、XML バージョンのフォームをアプリケーションで適宜処理する必要があります。この方法を使用すれば、独自の XSL エンジンを指定したり、より大きなフォームにそのフォームを組み込んだりすることが可能です。

ColdFusion XSL スキン

ColdFusion には、次の XSLT スキンが用意されています。

- basic
- basiccss
- basiccss_top
- beige
- blue
- default
- lightgray
- red
- silver

XSLT スキンファイルは、"<ColdFusion Web ルートディレクトリ >¥CFIDE¥scripts¥xsl" ディレクトリにあります。スタイル定義に使用する CSS ファイルは、"<ColdFusion Web ルートディレクトリ >¥CFIDE¥scripts¥css" ディレクトリにあります。

default スキンと basic スキンでは、同一の形式設定が行われます。cfform タグで skin 属性を指定しなかった場合は、default スキンが使用されます。default スキンと basic スキンは、テーブルを使用してフォームコンテンツを配置する単純なスキンです。basic スキンでは、div タグと span タグを使用して要素が配置されます。色の名前を持つスキンは、basic スキンに似ていますが、よりカラフルな外観になります。

例 : 単純なスキン可能フォーム

次の図は、default スキンで形式設定された単純な XML スキン可能フォームです。

このフォームを使用して例および説明を示します。

XML スキン可能フォームの構築

ColdFusion XML スキン可能フォームを構築するには、cfformgroup や cfformitem タグなどの、標準の ColdFusion フォームタグを使用します。それらのタグは、フォームの要素 (ビルディングブロック) になります。

ColdFusion では、次のタグが XML に変換されて、XSLT で処理できるようになります。

標準の ColdFusion フォームデータコントロールタグ cfgrid、cfinput、cfselect、cfslider、cftextarea、および tree タグは、フォームに表示するコントロールを指定します。

cfformitem タグ 個別の項目 (テキストや罫線など) をフォームに追加します。有効なタイプは、スキンによって異なります。

cfformgroup タグ フォームコンテンツをグループ化し、格納します。有効なタイプは、スキンによって異なります。

フォームを開発するときは、これらのタグを使用して、コンテナと子による階層構造を形成します。このモデルでは、cfform タグがマスタコンテナであり、その内容が子のコンテナおよびコントロールになります。それぞれの cfformgroup タグによって、子要素を格納するコンテナが定義されます。

どのタグや属性をフォームで使用するかは、XSLT スキンの機能によって異なります。使用できるのは、スキンでサポートされているタグおよび属性のみです。サードパーティのスキンを使用する場合は、サポートされている属性に関する情報が提供されているか確認してください。

標準の ColdFusion フォームタグの使用

フォームの入力要素を生成するには、標準の CFML フォームと同じように、cfinput タグや cftree タグなどの標準の ColdFusion フォームタグを使用します。これらのタグおよびサブタグ (cfselect タグの option タグなど) のほとんどは、対応する XForms 要素にマッピングされます。アプレットや、Flash 形式の cfgrid および cftree タグは、Java アプレットや Flash オブジェクトを含んだ ColdFusion XML エクステンションにマッピングされます。XML 形式の cfgrid および cftree タグは、ColdFusion XML エクステンションに変換されます。

使用できる属性やその効果は、スキンによって異なります。

ColdFusion スキンを使用する場合 : ColdFusion に用意されているスキンでは、HTML フォームと同じ属性がサポートされています。また、label 属性を使用して、次のタグのラベルを指定することもできます。

- type 属性の値が text、button、password、および file の cfinput
- cfselect
- cfslider
- cftextarea

その他のスキンを使用する場合 : その他のスキンでは、一部の属性がサポートされていなかったり、カスタムの属性がサポートされていたりすることがあります。サポートされている属性に関する情報は、XSLT スキンの開発者から入手してください。

cfformitem タグの使用

インラインテキストや標準の HTML タグは、XML フォームの生成時には無視されます。したがって、形式設定された HTML、プレーンテキストブロック、およびその他の表示要素 (水平線や垂直線など) をフォームに追加したい場合は、cfformitem タグを使用する必要があります。

cfformitem の type 属性の値は、すべて小文字に変換されます。たとえば、type="MyType" と指定した場合は、"mytype" というタイプ名に変換されます。

フォームで使用可能な cfformitem の type 属性にその他の制限はありませんが、項目を表示するためには、XSLT スキンでその属性を処理する必要があります。

ColdFusion スキンを使用する場合 : ColdFusion に用意されているスキンでは、次の cfformitem タイプがサポートされています。

- hrule
- text
- html

hrule タイプは、HTML の hr タグを挿入します。text タイプは、形式設定されていないプレーンテキストを表示します。

html タイプは、HTML の形式が設定されたテキストを表示します。strong、p、ul、li などの標準の HTML テキストマークアップタグとその属性を使用できます。たとえば、例 : 単純なスキン可能フォームで使用されている次のコードでは、cfformitem タグを使用してフォームに説明テキストを挿入しています。

```
<cfformitem type="html">
    <b>We value your input</b>.<br>
    <em>Please tell us a little about yourself and your thoughts.</em>
</cfformitem>
```

その他のスキンを使用する場合 : その他のスキンでサポートされている属性および属性値は、スキンの実装によって異なります。サポートされている属性と属性値に関する情報は、XSLT スキンの開発者から入手してください。

cfformgroup タグの使用

cfformgroup タグを使用すれば、子タグを編成して (たとえば水平方向または垂直方向に配置して) フォームを構造化できます。一部のスキンでは、cfformgroup タグを使用して、より複雑な形式設定 (タブ付きナビゲータやアコーディオンコンテナなど) が使用できます。フォームで使用可能な type 属性に制限はありませんが、生成された XML を XSLT で処理して表示に反映できる必要があります。

ColdFusion スキンを使用する場合 : ColdFusion に用意されているスキンでは、次の type 属性値がサポートされています。

- horizontal
- vertical
- fieldset

horizontal および vertical タイプは、指定の方向に子タグを配置し、グループの左にラベルを配置します。例 : 単純なスキン可能フォームで使用されている次のコードでは、cfformgroup タグを使用して **Name** というラベルを適用し、姓フィールドと名フィールドを水平方向に配置しています。

```
<cfformgroup type="horizontal" label="Name">
    <cfinput type="text" name="firstname" label="First" required="yes">
    <cfinput type="text" name="lastname" label="Last" required="yes">
</cfformgroup>
```

fieldset タイプは、HTML の fieldset タグに対応します。このタイプは、子の周囲にボックスを描画して子をグループ化するとともに、上辺の線の一部を凡例テキストに置換します。凡例を指定するには、label 属性を使用します。ボックスの大きさを指定するには、style 属性の height および width の値を使用します。

次のコードは、3 つのテキストコントロールを持つ単純なフォームグループです。cfformgroup type="vertical" タグで、フォームの全体的な配置を指定しています。cfformgroup type="horizontal" で、姓と名のフィールドを水平方向に配置しています。

```
<cfform name="comments" format="xml" skin="basiccss" width="400"
        preservedata="Yes" >
    <cfformgroup type="fieldset" label="Contact Information">
        <cfformgroup type="vertical">
            <cfformgroup type="horizontal" label="Name">
                <cfinput type="text" size="20" name="firstname" required="yes">
                <cfinput type="text" size="25" name="lastname" required="yes">
            </cfformgroup>
            <cfinput type="text" name="email" label="E-mail" validation="email">
        </cfformgroup>
        </cfformgroup>
</cfform>
```

注意：XML では大文字と小文字が区別されますが、ColdFusion では区別されません。ColdFusion では、cfformgroup および cfformitem 属性はすべて小文字に変換されます。たとえば、cfformgroup type="Random" と指定した場合、ColdFusion で生成される XML では "random" というタイプ名に変換されます。

その他のスキンを使用する場合：その他のスキンでサポートされている属性および属性値は、スキンの実装によって異なります。サポートされている属性と属性値に関する情報は、XSLT スキンの開発者から入手してください。

例：スキン可能 XML フォームの CFML

769 ページの「XML スキン可能フォームについて」の図で示したフォームは、次の CFML コードによって作成されます。このコードは、CFML を使用してフォームを構築する方法を示しています。

```
<cfform name="comments" format="xml" skin="basiccss" width="400" preservedata="Yes" >
    <cfinput type="hidden" name="revision" value="12a">
    <cfformgroup type="fieldset" label="Basic Information">
        <cfformgroup type="vertical">
        <cfformgroup type="horizontal" label="Name">
            <cfinput type="text" size="20" name="firstname" required="yes">
            <cfinput type="text" size="25" name="lastname" required="yes">
        </cfformgroup>
        <cfinput type="text" name="email" label="E-mail" validate="email" maxlength="35">
        <cfselect name="satisfaction" style="width:120px" multiple="false" label="Satisfaction">
            <option selected>very satisfied</option>
            <option>somewhat satisfied</option>
            <option>somewhat dissatisfied</option>
            <option>very dissatisfied</option>
            <option>no opinion</option>
        </cfselect>
        </cfformgroup>
        </cfformgroup>
    <cfformitem name="html1" type="html">
    <p><b>We value your input</b>.<br>
    <em>Please tell us a little about yourself and your thoughts.</em></p>
    </cfformitem>
    <cftextarea name="thoughts" label="Additional Comments" rows="5" cols="66">We really want to hear from
you!</cftextarea>
    <cfformgroup type="horizontal">
        <cfinput type="submit" name="submit" style="width:80" value="Tell Us">
        <cfinput type="reset" name="reset" style="width:80" value="Clear Fields">
    </cfformgroup>
</cfform>
```

ColdFusion XML 形式

ここでは、ColdFusion cfform タグとその子から生成される XML について説明します。この情報は、独自の XSL スキンを作成する際に役立ちます。

XML 名前空間の使用

ColdFusion でフォーム用に生成される XML では、いくつかの XML 名前空間に属する要素や属性が使用されます。名前空間とは、名前のコレクションのことで、XML 名を一意にするために使用されます。多くの場合、名前空間は何らかの Web 標準仕様、特定の DTD (ドキュメントタイプ定義)、またはスキーマに対応しています。XML では、名前空間名とコロン (:) を、その名前空間で定義されているタグ名の前に配置します。たとえば、XForms 名前空間の model タグは、xf:model となります。

ColdFusion では、W3C (World Wid Web Consortium) によって定義される、いくつかの標準 XML 名前空間が使用されています。これらの名前空間は、XHTML、XForms、XML Events などの、標準の XML 方言の仕様に対応します。また、ColdFusion XML フォームでは、スキン可能フォーム XML エクステンションに対するカスタム名前空間も使用されています。次の表に、ColdFusion によって生成される XML で使用される名前空間を示します。

接頭辞	URL	用途
html	http://www.w3.org/1999/xhtml	アクション、高さ、幅、名前などのフォームタグ情報。XHTML に準拠しています。
xf	http://www.w3.org/2002/xforms	cfform タグに対応する XForms モデル (フィールドの初期値を含む) および XForms 要素。
ev	http://www.w3.org/2001/xml-events	システムイベント。cfinput type="reset" に使用されます。
cf		XForms の要素または属性に対応していない属性のパススルーを含む、すべての ColdFusion Extension。

XML の構造

CFML タグの属性と要素値は、XForms の xf:model 要素や、個別のコントロール要素 (XForms の xf:input、xf:secret、xf:group 要素など) として XML に変換されます。

ColdFusion では、次の形式の XForms XML が生成されます。各行の数字は、タグのネストレベルを表します。

```
1   form tag
2       XForms model element
3           XForms instance element
4               cf:data element
3           XForms submission element
3           XForms bind element
3           XForms bind element
3       .
3       .
3       .
2       (end of model element)
2       XForms or ColdFusion extension control element
2       XForms or ColdFusion extension control element
            .
            .
            .
1   (end of form)
```

データモデル

XForms データモデルには、フォームから送信されるデータが指定されています。データモデルには、データを送信可能なそれぞれの表示コントロールに関する情報 (初期値や検証情報など) が含まれています。cfformgroup や cfformitem タグに関する情報は含まれていません。データモデルは、次の要素およびその子で構成されます。

- 1 つの xf:instance 要素
- 1 つの xf:submission 要素
- データを送信可能なフォームコントロールごとに 1 つの xf:bind 要素

xf:instance 要素

XForms の xf:instance 要素には、フォームデータコントロールに関する情報が含まれています。データを送信可能なコントロールには、対応するインスタンス要素が存在しています。コントロールに初期値が設定されている場合は、インスタンス要素にその値が含まれています。

xf:instance 要素には、単一の cf:data 要素が含まれています。cf:data 要素には、それぞれのデータコントロール (cfgrid、ほとんどの cfinput タグタイプ、cfselect、cfslider、cftextarea、および cftree) に対応する要素が 1 つずつ含まれています。それらの要素名は、CFML タグの name 属性に対応しています。アプレットおよび Flash 形式の cfgrid および cftree タグの場合は、ツリーまたはグリッドの Java アプレットオブジェクトの cf_param_name パラメータの値が要素名として使用されます。submit、image、reset、および button タイプの cfinput タグは、データを送信できないので、インスタンスデータを持ちません。

各要素の本文には、CFML タグの value 属性またはそれと同等のもので指定された初期コントロールデータが含まれています。たとえば cfselect タグの場合、xf:instance 要素の本文は、selected 属性を持つすべての option タグの name 属性が列挙されたカンマ区切りのリストになります。submit および image ボタンの本文は、name 属性の値になります。

次の例は、769 ページの「XML スキン可能フォームについて」の図で示したフォームの xf:instance 要素です。

```
<xf:instance>
    <cf:data>
            <firstname/>
            <lastname/>
            <email/>
            <revision>Comment Form revision 12a</revision>
            <satisfaction>very satisfied</satisfaction>
            <thoughts>We really want to hear from you!</thoughts>
    </cf:data>
</xf:instance>
```

xf:submission 要素

xf:submission 要素には、フォーム送信時のアクションが指定されています。この要素には、cfform の action 属性と method 属性の値が含まれています。

次の例は、769 ページの「XML スキン可能フォームについて」の図で示したフォームの XML です。

```
<xf:submission action="/_MyStuff/phase1/forms/XForms/FrameExamples/Figure1.cfm"
    method="post"/>
```

xf:bind 要素

xf:bind 要素は、入力コントロールの動作に関する情報 (コントロールのタイプやデータ検証ルールなど) を提供します。XML には、データを送信可能なインスタンス要素のそれぞれに対して、bind 要素が 1 つずつ記述されています。cfformitem タグや、submit、input、reset、image タイプの cfinput タグなどのコントロールには、bind 要素はありません。各要素には次の属性が含まれています。

属性	説明
id	CFML タグの name 属性の値
nodeset	コントロールのインスタンス要素に対する XML 形式のパスを含む XPath 式
required	CFML タグの required 属性の値

xf:bind 要素には、ColdFusion 固有の情報 (タイプや検証値など) が設定された xf:extension 要素が含まれています。次の表に、ここで使用されている名前空間要素を示します。

要素	説明
cf:attribute name="type"	コントロールのタイプ。次のいずれかです。 CHECKBOX、FILE、IMAGE、PASSWORD、RADIO、SELECT、SUBMIT TEXT、CFSLIDER。 TEXT タイプは、cfinputtype="text" および cftextinput で使用されます。
cf:attribute name="onerror"	コントロールの onError 属性で指定されている JavaScript 関数 (この属性がある場合)。
cfargument name="maxlength"	コントロールの maxlength 属性の値 (この属性がある場合)。
cf:validate type="valiadationtype"	データ検証情報。 検証タイプを示す 1 つの属性 type と、1 つまたは複数の cf:argument および cf:trigger 子要素を持ちます。次のそれぞれに対して cf:validate 要素が生成されます。 • cfinput または cftextareavalidation 属性 • cfinput または cftextarearange 属性 • cfslider : range および message 属性は、cf:validate type="range" 要素によって指定されます。
cf:argument (cf:validate 要素の本文内)	データ検証の仕様。 1 つの属性 name と、本文テキストを持ちます。1 つの cf:validate 要素には、複数の cf:argument 子要素が含まれていることがあります。これらの子要素は、最大長や値の範囲など、検証に関連する CFML タグ属性値に対応します。この要素の本文には、CFML 属性値が含まれます。 有効な name の値は次のとおりです。特に指定がない限り、この名前は、対応する CFML タグ属性の名前に対応します。 • max • message • min • pattern
cf:trigger (cf:validate 要素の本文内)	検証を実行するタイミング。フォーム要素の validateAt 属性の値が指定されます。 1 つの属性 event を持ちます。この値は次のいずれかです。 • onBlur • onSubmit • onServer validateAt 属性で複数の検証トリガが指定されている場合、XML では、リスト内の各エントリに対して 1 つの cf:trigger 要素が設定されます。

次の例は、769 ページの「XML スキン可能フォームについて」の図で示したフォームの xf:bind 要素です。

```
<xf:bind id="firstname"
        nodeset="//xf:model/xf:instance/cf:data/firstname"
        required="true()">
    <xf:extension>
        <cf:attribute name="type">TEXT</cf:attribute>
        <cf:attribute name="onerror">_CF_onError</cf:attribute>
    </xf:extension>
</xf:bind>
<xf:bind id="lastname"
        nodeset="//xf:model/xf:instance/cf:data/lastname"
        required="true()">
    <xf:extension>
        <cf:attribute name="type">TEXT</cf:attribute>
        <cf:attribute name="onerror">_CF_onError</cf:attribute>
    </xf:extension>
</xf:bind>
<xf:bind id="email"
        nodeset="//xf:model/xf:instance/cf:data/email" required="false()">
    <xf:extension>
        <cf:attribute name="type">TEXT</cf:attribute>
        <cf:attribute name="onerror">_CF_onError</cf:attribute>
    </xf:extension>
</xf:bind>
<xf:bind id="satisfaction"
        nodeset="//xf:model/xf:instance/cf:data/satisfaction"
        required="false()">
    <xf:extension>
        <cf:attribute name="type">SELECT</cf:attribute>
        <cf:attribute name="onerror">_CF_onError</cf:attribute>
    </xf:extension>
</xf:bind>
<xf:bind id="thoughts"
        nodeset="//xf:model/xf:instance/cf:data/thoughts" required="false()">
    <xf:extension>
        <cf:attribute name="type">TEXT</cf:attribute>
        <cf:attribute name="onerror">_CF_onError</cf:attribute>
    </xf:extension>
</xf:bind>
```

コントロール要素

xf:bind 要素の後の XML タグでは、フォームのコントロールとそのレイアウトが指定されています。フォームコントロールタグ、cfformitem タグ、cfformgroup タグのそれぞれに対して、要素が 1 つずつ含まれています。

CFML タグと XML タグのマッピング

CFML タグは、次の表に示す XForms 要素および ColdFusion Extension にマッピングされます。

CFML タグ	XML タグ
cfinput type="text"	xf:input
cfinput type="password"	xf:secret
cfinput type="hidden"	なし : インスタンスデータのみ
cfinput type="file"	xf:upload
cfinput type="radio"	xf:select1
cfinput type="checkbox"	xf:select

CFML タグ	XML タグ
cfinput type="button"	xf:trigger
cfinput type="image"	xf:submit
cfinput type="reset"	xf:submit
cfinput type="submit"	xf:submit
cfselect multiple="false"	xf:select1
cfselect multiple="true"	xf:select
cftextarea	xf:textarea
cfslider	xf:range
cfgrid	cf:grid
cftree	cf:tree
cfformitem type="text"	xf:output
cfformitem type="html"	xf:output
cfformitem type="*" (text、html 以外のすべて)	xf:group appearance="*"
cfformgroup type="*"	xf:group appearance="*"

type 属性が text または html の cfformitem タグは、XForms の output 要素に変換され、タグ本文が <![CDATA[セクションに含められます。他のすべての cfformitem タグは XForms の group 要素に変換され、各要素の appearance 属性に cfformitem タグの type 属性の値が設定されます。XSLT ではこれらの要素を処理して、意味のある出力を生成する必要があります。たとえば、ColdFusion の default スキンの変換では、xf:output テキストブロックを表示し、xf:groupappearance="hrule" 要素を処理しますが、その他の xf:group 要素はすべて無視します。

コントロール要素の一般的な構造

標準の XForms コントロール要素で表せるコントロール要素は、次の一般的な構造を持ちます。

```
<xf:tagname bind="bindid" id="bindid">
    <xf:label>label</xf:label>
    <xf:extension>
        <cf:attribute name="type">controltype</cf:attribute>
        <cf:attribute name="attribname>attribvalue</cf:attribute>
        <cf:attribute name="attribname>attribvalue</cf:attribute>
        .
        .
        .
    </xf:extension>
</xf:tagname>
```

次の表で、この構造の変数部分について説明します。

部分	説明
tagname	777 ページの「CFML タグと XML タグのマッピング」の表に示されている、xf または cf 名前空間に属する要素名。
bindid	このコントロールのモデル xf:bind 要素の ID 属性。コントロールの CFML タグの name 属性で指定します。
label	コントロールのラベルテキスト。次のいずれかで指定されます。 • CFML タグの label 属性 • radiobutton、radio、submit、および reset タイプの cfinput タグの value 属性 • cfselect の option サブタグの本文の内容 • cfgrid および cftree タグでは使用されません。
controltype	コントロールのタイプ。次のいずれかです。 • cfinput の type 属性 • select、slider、または textarea。それぞれ cfselect タグ、cfslider タグ、cftextarea タグを意味します。 • cfgrid および cftree タグでは使用されません。
attribname	CFML タグ属性の名前。CFML コードで指定されているが、XML にエントリを持たないそれぞれの属性に対して、cf:attribute タグが 1 つずつ存在します。
attribvalue	CFML タグ属性の値。

タグ固有の要素構造

ここでは、いくつかのタイプの入力タグを取り上げて、XML のタグ固有の機能について説明します。この説明は、すべてを網羅したものではありません。特定の ColdFusion フォームタグの構造については、ColdFusion によってタグから生成される XML を参照してください。

選択タグ

選択に使用される cfselect、cfinputtype="radio"、および cfinputtype="checkbox" タグは、XForms の select 要素および select1 要素に変換されます。これらの要素には、1 つの xf:choices 要素が含まれています。この xf:choices 要素には、ユーザーが選択できるそれぞれの項目に対して xf:item 要素が 1 つずつ含まれています。通常、各項目には xf:label 要素と xf:value 要素があります。チェックボックスは単一の項目を持ち、選択およびラジオボタンコントロールには複数の項目を持ちます。

次の例は、2 つのラジオボタンを含むグループの CFML コードと、生成される XML コントロール要素です。この例では、cfformgroup タグを使用してラジオボタングループを配置およびラベル設定する方法も示されています。

CFML

```
<cfformgroup type="horizontal" label="Accept?">
    <cfinput type = "Radio" name = "YesNo" value = "Yes" checked>
    <cfinput type = "Radio" name = "YesNo" value = "No">
</cfformgroup>
```

XML

```
<xf:group appearance="horizontal">
    <xf:label>Accept?</xf:label>
    <xf:extension/>
    <xf:select1 appearance="full" bind="YesNo" id="YesNo">
        <xf:extension>
            <cf:attribute name="type">radio</cf:attribute>
        </xf:extension>
        <xf:choices>
            <xf:item>
                <xf:label>Yes</xf:label>
                <xf:value>Yes</xf:value>
                <xf:extension>
                    <cf:attribute name="checked">checked</cf:attribute>
                </xf:extension>
            </xf:item>
            <xf:item>
                <xf:label>No</xf:label>
                <xf:value>No</xf:value>
                <xf:extension/>
            </xf:item>
        </xf:choices>
    </xf:select1>
</xf:group>
```

cfgrid タグ

cfgrid タグは、XML の cf:grid タグで表されます。このタグには 4 つの属性があります。format 属性は、Flash、Applet、または XML です。id 属性、name 属性、および bind 属性の値は、すべて cfgrid タグの name 属性の値になります。

アプレットおよび Flash 形式のグリッドの場合は、XML の cf:grid タグ本文の <![CDATA[セクションに、HTML 埋め込みオブジェクトとして cfgrid コントロールが挿入されます。このコントロールは、Java アプレットか SWF ファイル形式のいずれかになります。

XML 形式のグリッドの場合は、CFML が次の形式の XML に変換されます。

```
<cf:grid bind="gridname" name="gridname" format="xml" id="gridname>
    <metadata>
        <cfgridAttribute1>attributeValue</cfgridAttribute1>
        ...
        (There are an entry for attributes with a specified or default value.)
    </metadata>
    <columns>
        <column cfgridcolumnAttribute1="value" ... />
        ...
    </columns>
    <rows>
        <row>
            <column1Name>row1Column1Value</column1Name>
            <column2Name>row1Column2Value</column2Name>
            ...
        </row>
        <row>
            <column1Name>row2Column1Value</column1Name>
            <column2Name>row2Column2Value</column2Name>
        </row>
        ...
    </rows>
</cf:grid>
```

次の例は、2 つのノードを持つ最小のグリッドです。

CFML

```
<cfgrid name="mygrid" Format="xml" selectmode="Edit" width="350">
    <cfgridcolumn name="CorName" header="Course Name" >
    <cfgridcolumn name="Course_ID" header="ID">
    <cfgridrow data="one0,two0">
    <cfgridrow data="one1,two1">
</cfgrid>
```

XML

ほとんどのメタデータ行は、簡略化のために省略しています。

```
<cf:grid bind="mygrid" format="XML" id="mygrid" name="mygrid">
    <metadata>
        <autowidth>false</autowidth>
        <insert>false</insert>
        <delete>false</delete>
        <sort>false</sort>
        <italic>false</italic>
        <bold>false</bold>
        <appendkey>true</appendkey>
        <highlughthref>true</highlughthref>
        <griddatalines>Left</griddatalines>
        <gridlines>true</gridlines>
        <rowheaders>true</rowheaders>
        <rowheaderalign>Left</rowheaderalign>
        <rowheaderitalic>false</rowheaderitalic>
        <rowheaderbold>false</rowheaderbold>
        <colheaders>true</colheaders>
        <colheaderalign>Left</colheaderalign>
        <colheaderitalic>false</colheaderitalic>
        <colheaderbold>false</colheaderbold>
        <selectmode>Edit</selectmode>
        <notsupported>&lt;b&gt; Browser must support Java to view ColdFusion Java
            Applets&lt;/b&gt;</notsupported>
        <picturebar>false</picturebar>
        <insertbutton>insert</insertbutton>
        <deletebutton>delete</deletebutton>
        <sortAscendingButton>SortAsc</sortAscendingButton>
        <sortDescendingButton>SortDesc</sortDescendingButton>
    </metadata>
    <columns>
        <column bold="false" display="true" header="Course Name"
            headerBold="false" headerItalic="false" italic="false"
            name="CorName" select="true"/>
        <column bold="false" display="true" header="ID"
            headerBold="false" headerItalic="false" italic="false"
            name="Course_ID" select="true"/>
    </columns>
    <rows>
        <row>
            <CorName>one0</CorName>
            <Course_ID>two0</Course_ID>
        </row>
        <row>
            <CorName>one1</CorName>
            <Course_ID>two1</Course_ID>
        </row>
    </rows>
</cf:grid>
```

cftree タグ

アプレットおよび Flash 形式のツリーの場合は、XML のタグ本文の <![CDATA[セクションに、HTML 埋め込みオブジェクトとして cftree コントロールが挿入されます。このコントロールは、Java アプレットか Flash SWF ファイル形式のいずれかになります。cf:tree XML タグは、format (Flash または Applet) と id の 2 つの属性を持ちます。

XML 形式のツリーの場合は、CFML が次の形式の XML に変換されます。

```
cf:tree format="XML" id="treename>
    <metadata>
        <cftreeAttribute1>attributeValue</cftreeAttribute1>
        ...
    </metadata>
    <node cfml tree item attributes>
        <node //nested node with no children
            cfml tree item attributes />
        ...
    </node>
    ...
</cf:tree>
```

次の例は、2 つのノードを持つ最小のツリーです。

CFML

```
<cfform name="form2" Format="XML" >
<cftree name="tree1" hscroll="No" vscroll="No" format="xml"
    border="No">
    <cftreeitem value="Divisions">
    <cftreeitem value="Development"
        parent="Divisions" img="folder">
</cftree>
</cfform>
```

XML

次のコードでは、ツリーの外観に関連する XML のみを示します。

```
<cf:tree format="xml" id="tree1">
    <metadata>
        <fontWeight/>
        <align/>
        <lookAndFeel>windows</lookAndFeel>
        <delimiter>\</delimiter>
        <completePath>false</completePath>
        <border>false</border>
        <hScroll>false</hScroll>
        <vScroll>false</vScroll>
        <appendKey>true</appendKey>
        <highlightHref>true</highlightHref>
        <italic>false</italic>
        <bold>false</bold>
    </metadata>
    <node display="Divisions" expand="true" href="" img=""
        imgOpen="" parent="" path="Divisions" queryAsRoot="true"
                    value="Divisions">
        <node display="Development" expand="true" href=""
            img="folder" imgOpen="" parent="Divisions"
            path="Divisions\Development" queryAsRoot="true"
                        value="Development"/>
    </node>
</cf:tree>
```

cfformgroup タグと cfformitem タグ

すべての cfformgroup タグと cfformitem タグ (type="html" および type="text" を除く) は、xf:group 要素になります。この要素の構造は、次のルールによって決定されます。

- CFML タグの type 属性によって、xf:group の appearance 属性が指定されます。
- type 属性の値は、すべて小文字に変換されます。
- cfformgroup タグに限り、CFML label 属性によって、xf:group の label 属性が指定されます。
- その他すべての CFML 属性は、xf:extension 要素の cf:attribute 要素に配置されます。
- cfformitem タグは、<![CDATA[セクションに本文テキストが含まれた xf:output 要素になります。

次の例は、2 つの cformitem タグと、生成される XML です。

CFML

```
<cfformitem name="text1" type="text" style="color:green">
    Please tell us a little about yourself and your thoughts.
</cfformitem>
<cfformitem type="hrule" height="3" width="200" testattribute="testvalue" />
```

XML

```
<xf:output><![CDATA[Please tell us a little about yourself and your
thoughts.]]>
    <xf:extension>
        <cf:attribute name="style">color:green</cf:attribute>
    </xf:extension>
</xf:output>
<xf:group appearance="hrule">
    <xf:extension>
        <cf:attribute name="width">200</cf:attribute>
        <cf:attribute name="height">3</cf:attribute>
        <cf:attribute name="testattribute">testvalue</cf:attribute>
    </xf:extension>
</xf:group>
```

例 : コントロール要素の XML

次のコードは、769 ページの「XML スキン可能フォームについて」の図で示したフォームの、入力コントロールに関する XML です。このコードは、xf:model 要素の直後に配置されます。

```
<xf:group appearance="horizontal">
    <xf:label>name</xf:label>
    <xf:extension/>
    <xf:input bind="firstname" id="firstname">
        <xf:label>First</xf:label>
        <xf:extension>
            <cf:attribute name="type">text</cf:attribute>
            <cf:attribute name="size">20</cf:attribute>
        </xf:extension>
    </xf:input>
    <xf:input bind="lastname" id="lastname">
        <xf:label>Last</xf:label>
        <xf:extension>
            <cf:attribute name="type">text</cf:attribute>
            <cf:attribute name="size">25</cf:attribute>
        </xf:extension>
    </xf:input>
</xf:group>
<xf:input bind="email" id="email">
    <xf:label>Email</xf:label>
    <xf:extension>
        <cf:attribute name="type">text</cf:attribute>
        <cf:attribute name="validation">email</cf:attribute>
    </xf:extension>
</xf:input>
<xf:output><![CDATA[<b>We value your input</b>.<br>
    <em>Please tell us a little about yourself and your thoughts.</em>]]>
    <xf:extension/>
</xf:output>
<xf:group appearance="vertical">
    <xf:extension/>
    <xf:select1 appearance="minimal" bind="satisfaction" id="satisfaction">
        <xf:label>Satisfaction</xf:label>
        <xf:extension>
            <cf:attribute name="type">select</cf:attribute>
            <cf:attribute name="style">width:200</cf:attribute>
        </xf:extension>
        <xf:choices>
            <xf:item>
                <xf:label>very satisfied</xf:label>
                <xf:value>very satisfied</xf:value>
            </xf:item>
            <xf:item>
                <xf:label>somewhat satisfied</xf:label>
                <xf:value>somewhat satisfied</xf:value>
            </xf:item>
            <xf:item>
                <xf:label>somewhat dissatisfied</xf:label>
                <xf:value>somewhat dissatisfied</xf:value>
            </xf:item>
            <xf:item>
                <xf:label>very dissatisfied</xf:label>
                <xf:value>very dissatisfied</xf:value>
            </xf:item>
            <xf:item>
                <xf:label>no opinion</xf:label>
                <xf:value>no opinion</xf:value>
            </xf:item>
        </xf:choices>
    </xf:select1>
    <xf:textarea bind="thoughts" id="thoughts">
        <xf:label>Additional Comments</xf:label>
```

```
        <xf:extension>
            <cf:attribute name="type">textarea</cf:attribute>
            <cf:attribute name="rows">5</cf:attribute>
            <cf:attribute name="cols">40</cf:attribute>
        </xf:extension>
    </xf:textarea>
</xf:group>
<xf:group appearance="horizontal">
    <xf:extension/>
    <xf:submit id="submit" submission="comments">
        <xf:label>Tell Us</xf:label>
        <xf:extension>
            <cf:attribute name="type">submit</cf:attribute>
            <cf:attribute name="name">submit</cf:attribute>
        </xf:extension>
    </xf:submit>
    <xf:submit id="reset">
        <xf:label>Clear Fields</xf:label>
        <reset ev:event="DOMActivate"/>
        <xf:extension>
            <cf:attribute name="name">reset</cf:attribute>
        </xf:extension>
    </xf:submit>
</xf:group>
```

XSLT スキンの作成

ColdFusion で生成された XML を処理する独自の XSLT スキンを作成できます。ここでは、読者が XSLT と CSS プログラミングに習熟していることを前提としています。XSLT 変換や CSS スタイルの作成に関する一般的な説明は行いません。このセクションでは、次の項目について説明します。

- フォーム属性値が XML ファイルに渡される方法
- ColdFusion でテンプレートとして提供されている XSLT スキンを拡張する方法
- "basic.xsl" ファイルを拡張して、cfformgroup および cfformitem タグの追加の type 属性をサポートする基本的なテクニック
- ColdFusion CSS ファイルを拡張して、フォームの外観を向上させる方法

XSLT スキンファイルの場所

XSLT スキンを名前で指定し、.xsl 拡張子を省略すると、ColdFusion によって、cfform のスクリプトソースディレクトリとそのサブディレクトリ内で該当するファイルが検索されます。スクリプトソースディレクトリは、cfform タグの scriptsrc 属性で指定できます。また、デフォルトの場所を ColdFusion Administrator の [設定] ページで設定できます。デフォルトの場所は、ColdFusion のインストール時に "/CFIDE/scripts/" (Web ルートディレクトリからの相対位置) に設定されます。

XSLT スキンの場所は、相対ファイルパス、絶対ファイルパス、または URL で指定できます。相対パスの場合、基準は CFML ページがあるディレクトリとなります。次の形式が有効です。

形式	場所
<cfform format="xml" skin="basic">	デフォルトディレクトリおよびそのサブディレクトリ内で XML/CSS が検索されます。
<cfform format="xml" skin="c:¥foo¥bar¥basic.xsl">	絶対パスが使用されます。

形式	場所
<cfform format="xml" skin="basic.xsl">	現在のディレクトリ内が検索されます。
<cfform format="xml" skin="..¥basic.xsl:">	現在のディレクトリの親ディレクトリ内が検索されます。
<cfform format="xml" skin="http://anywhereOnTheWeb/basic.xsl">	指定した URL が使用されます。

注意：ホスティング業者では、スキンのデフォルトの場所を CFIDE の外に移動することが有用な場合があります。そのようにすれば、cfform で必要とされるファイルにサイト開発者がアクセスできるようにしつつ、CFIDE のセキュリティを確保できます。

属性および値のパススルー

特別な処理を必要としないフォームタグの属性または属性値は、次のように XML に直接渡されます。

- cfformitem および cfformgroup の type 属性は、xf:group 要素の appearance 属性に変換されます。
- 認識または処理されないタグ属性の名前と値は、cf:attribute 要素に渡されます。

このパススルー機能を利用すれば、次の項目のカスタムバージョンを作成して、XSLT で処理することができます。

- cfformitem タイプ (罫線、スペーサー、またはその他の表示要素)
- cfgroup タイプ (分割されたボックスやタブ付きのダイアログボックスなど)
- カスタム cfinput タイプ (カスタムの年選択要素など)
- ColdFusion タグ属性 (検証の制御に使用する属性など)

ColdFusion XSLT スキンの拡張

ColdFusion に用意されている基本的な XSLT 変換は、独自のスキンを作成するためのテンプレートとして使用して、その機能を拡張することができます。各スキンにはベースとなる XSL ファイルが存在しています。このファイルには、いくつかのユーティリティ XSL ファイルが含まれています。ユーティリティファイルは、アンダースコア (_) で始まる名前を持ち、複数のベーススキンによって共有されます。次の表で、"<ColdFusion Web ルートディレクトリ >¥¥x ディレクトリに存在する XSL ファイルについて説明します。

ファイル	説明
default.xsl	XML 形式のフォームで skin 属性を指定しなかった場合に使用されるデフォルトの変換。"basic.xsl" ファイルと同じです。
basic.xsl	テーブルを使用してフォーム要素を配置する基本フォーム形式。
basiccss.xsl	HTML の div および span タグを使用してフォーム要素を配置する基本フォーム形式。
< カラー名 >.xsl	テーブルを使用してフォーム要素を配置し、< カラー名 > で指定されるカラースキームをフォームに適用する基本フォーム形式。"basic.xsl" ファイルがベースになっています。
_cfformvalidation.xsl	ColdFusion 検証ルールを適用します。すべてのスキンによって使用されます。
_formelements.xsl	フォーム要素の変換ルール。cfformgroup タグで定義されるものは除きます。すべてのスキンによって使用されます。
group< タイプ >.xsl _group_< タイプ >_table.xsl _group_< タイプ >_css.xsl	cfformgroup タグの変換ルール。タグの *type* 属性がファイル名に含まれています。ファイル名に **table** が付いているものは、"basic.xsl" およびその派生ファイルによって使用されます。ファイル名に css が付いているものは、"basiccss.xsl" によって使用されます。

すべてのスキンでは、同じ CFML タグおよびタグタイプがサポートされており、XML から HTML への比較的単純な変換が実行されます。たとえば、水平線や垂直線はサポートされていません。

ColdFusion のスキン XSL ファイルには、独自の変換を設計および開発するときに役立つ、次のような特徴があります。

- カスタム変換を実装するための全体的な構造を提供する、初期テンプレートとして使用できます。
- ColdFusion で生成されるさまざまな XML 要素の処理方法が示されています。
- インクルードファイルが使用されており、独自の XSLT コードのテンプレートで利用できます。
- ベース XSL ファイルでは "_cfformvalidation.xsl" ファイルがインクルードされています。このファイルには、ColdFusion の onServer 検証を実行するために必要な非表示フィールドを生成するコードや、ColdFusion の onSubmit および onBlur 検証を実行するための JavaScript が含まれています。このファイルを、編集せずに XSLT テンプレートにインクルードすれば、ColdFusion 検証を実行できるようになります。または、このファイルを編集して、他の検証フォームを追加したり、検証ルールを変更したりすることもできます。
- 子タグのレイアウトとグループへのラベルの適用を行う、いくつかのフォームグループを提供するファイルが用意されており、ベース XSL ファイルでインクルードされています。これらのファイルをテンプレートとして使用して、フォームグループタイプを追加したり、より洗練された水平および垂直フォームグループを提供することができます。
- 追加の XSL ファイルをインクルードして、cfformgroup および cfformitem のカスタム type 属性を追加することができます。

"basic.xsl" の cfformgroup および cfformitem サポートの拡張

"basic.xsl" ファイルを拡張して、追加の cfformgroup および cfformitem タイプをサポートする手順を次に示します。他の xsl ファイルも同様の手順で拡張できます。

"basic.xsl" への cfformgroup および cfformitem タイプのサポートの追加

1. XSL ファイルを作成します。
2. サポートするそれぞれの type 属性に対して、形式設定を行う xsl:template 要素を 1 つずつ作成します。この要素の match 属性は、次の形式に従う必要があります。

   ```
   match="xf:group[@appearance='type_attribute_name']"
   ```

   たとえば、panel タイプの cfformgroup を追加するには、次に示す開始タグを持つ要素を追加します。

   ```
   <xsl:template match="xf:group[@appearance='panel']">
   ```

3. XSL ファイルを "<ColdFusion Web ルートディレクトリ >¥¥x" ディレクトリにデプロイします。
4. "basic.xsl" ファイルの Supported groups セクションの末尾に include ステートメントを追加します。たとえば、panel タイプの cfformgroup をサポートする "my_group_panel.xsl" ファイルを作成した場合は、"basic.xsl" ファイルに次の行を含めます。

   ```
   <!-- include groups that will be supported for this skin-->
   <xsl:include href="_group_vertical_table.xsl" />
   <xsl:include href="_group_horizontal_table.xsl" />
   <xsl:include href="_group_fieldset.xsl"/>
   <xsl:include href="my_group_panel.xsl" />
   ```

ColdFusion CSS ファイルの拡張によるフォームスタイルの設定

ColdFusion スキン可能フォームの XSL ファイルでは、対応する CSS スタイルシートを使用して、フォームのスタイルやレイアウト特性を指定します。次の CSS ファイルは、"<ColdFusion Web ルートディレクトリ >¥CFIDE¥scripts¥css" ディレクトリに存在します。

ファイル	説明
basic_style.css default_style.css	テーブルベースの形式を使用した単純なスタイルを ColdFusion XSL ファイルに適用します。この 2 つのファイルの内容は同じであり、"basic.xsl" および "default.xsl" 変換で使用されます。cfform タグでスキンを指定しなかった場合は、"default_style.css" が使用されます。
basic2_style.css	div ベースで形式設定を行っている XSL ファイルに適用するために、basic_style の配置変更を制限したもの。"basiccss.xsl" 変換で使用されます。
css_layout.css	div ベースで形式設定を行っているフォームをレイアウトするためのスタイル仕様。"basiccss.xsl" 変換で使用されます。
< カラー名 >_style.css	特定の色を使用した ColdFusion スキンで使用されます。"basic_style.css" と同じクラスにプロパティ仕様を追加しています。

ColdFusion XSL ファイルおよびそれに対応する CSS スタイルシートでは、フォームの形式設定のためにクラスを活用しています。たとえば、"basic.xsl" ファイルには、要素スタイルが 1 つだけ存在します。その他のスタイルはすべて、クラスをベースとしています。CSS ファイルには、XSL ファイルで使用されるすべてのクラスの仕様が含まれていますが、そのすべてに形式設定情報が含まれているわけではありません。たとえば、水平フォームグループで使用される "basic_style.css" の水平方向のクラス定義は空です。

ColdFusion に用意されているスタイルシートを拡張すれば、XSL 変換を変更しなくても、XML スキン可能フォームのスタイルを向上させることができます。

Ajax ユーザーインターフェイスコンポーネントおよび機能の使用

Ajax ベースのユーザーインターフェイス機能 (レイアウトコントロールやフォームコントロールなど) を使用して、ダイナミックアプリケーションを作成することができます。

Ajax フレームワークの一般的な機能や、Ajax のデータ機能およびプログラミング機能の使用方法 (フォームデータのバインディングや JavaScript リソースの管理など) については、839 ページの「Ajax データ機能および開発機能の使用」を参照してください。

Ajax および ColdFusion のユーザーインターフェイス機能について

Ajax (Asynchronous JavaScript and XML) とは、インタラクティブな Web アプリケーションを作成するための一連の Web テクノロジーのことです。Ajax アプリケーションは通常、次のテクノロジーから構成されます。

- 情報の形式設定および表示を行う HTML と CSS
- クライアントサイドでダイナミックな処理を実行する JavaScript
- XMLHttpRequest 関数によるサーバーとの非同期通信
- サーバーとクライアントの間でデータのシリアル化や転送を行うための XML または JSON (JavaScript Object Notation)

ColdFusion には、Ajax テクノロジーを使用したダイナミックアプリケーションの作成を支援するツールが多数用意されています。ColdFusion のタグや関数を使用すれば、複雑な Ajax アプリケーションを簡単に作成できます。

ColdFusion の Ajax 機能

ColdFusion では、次の 2 種類の Ajax 機能を使用できます。

- データ機能と開発機能

- ユーザーインターフェイス機能

データ機能と開発機能

ColdFusion のデータ機能と開発機能を使用すれば、ColdFusion でデータをダイナミックに処理する効果的な Ajax アプリケーションを開発できます。データ機能と開発機能には、Spry などの他の Ajax フレームワークで使用できる機能が多数用意されています。

次のデータ機能と開発機能は、フォームタグやレイアウトタグを使用するときに重要になります。

- ColdFusion の多くのタグでは、データバインディングがサポートされています。フォームタグや表示タグでバインディングを使用すれば、フォームの入力に基づいてダイナミックに情報を表示できます。フォームのデータを別のフォームフィールドにそのまま表示するだけでなく、フォームフィールドのデータをパラメータとして CFC 関数、JavaScript 関数、または CFM ページに渡し、その処理結果を使用して表示を制御することもできます。
- cfajaximport タグを使用すると、ColdFusion ページにインポートする JavaScript ファイルや CSS ファイルの場所を指定したり、特定のタグに必要なファイルを選択的にインポートしたりできます。ファイルの場所を自由に変更できるので、さまざまな設定をサポートでき、またアプリケーションごとのスタイル定義などの高度な設定も可能です。

データ機能、開発機能、およびその使用方法の詳細については、839 ページの「Ajax データ機能および開発機能の使用」を参照してください。

ユーザーインターフェイスのタグおよび機能

ColdFusion の一部のユーザーインターフェイス要素には、Ajax の機能が組み込まれています。それらのタグや、タグと属性の組み合わせは、次のカテゴリに分類できます。

- 内容のレイアウトや表示を行うコンテナタグ
- ファイルを処理するファイル管理タグ
- データをダイナミックに表示するフォームタグ
- メニューバーやプルダウンメニューの作成に使用するメニュータグ
- ツールヒントや入力支援などのユーザー補助機能
- 地図、進行状況表示バー、メディアプレーヤー、メッセージボックスを使用するためのその他の 4 種類のタグ

次の表に、Ajax ベースの機能を表示するための基本的なタグと属性を示します。フォーム固有の機能の詳細については、803 ページの「Ajax フォームのコントロールおよび機能の使用」を参照してください。

タグ / 属性	**説明**
コンテナタグ	
cfdiv	HTML の div 領域。バインド式を使用してダイナミックにデータを挿入できます。この領域のフォームは非同期で送信されます。
cflayout	水平方向のボックス、垂直方向のボックス、タブ領域、または境界線で囲まれた一連の領域 (上、下、左、右、中央の領域を使用可能)。
cflayoutarea	cflayout 領域内の個別の領域。たとえば、タブレイアウトでユーザーがタブを選択したときに表示される内容。この領域のフォームは非同期で送信されます。
cfpod	ブラウザウィンドウ内の区画。オプションのタイトルバーと、表示要素を配置する本文で構成されます。この領域のフォームは非同期で送信されます。
cfwindow	ブラウザ内に表示されるポップアップウィンドウ。ポップアップウィンドウは、ColdFusion.Window.createWindow 関数を使用して作成することもできます。この領域のフォームは非同期で送信されます。
ファイル管理タグ	
cffileupload	ユーザーのシステムから複数のファイルをアップロードするためのダイアログボックス。

タグ / 属性	説明
フォームタグ	
cfgrid format="html"	編集およびソートが可能なダイナミックなデータグリッド。
cfinput type="datefield"	ユーザーが日付を入力するためのコントロール。ポップアップカレンダーから日付を選択できます。
cftextarea richtext="yes"	表示テキストを形式設定するための一連のコントロールを備えたテキスト領域。
cftree format="html"	ダイナミックなツリー形式のデータ表現。
cfslider	一定の範囲から数値を選択するために ColdFusion フォーム内に表示されるスライダコントロール。
メニュータグ	
cfmenu	メニューバー、またはドロップダウンメニューのルート。
cfmenuitem	メニューを構成する個別のアイテム、またはサブメニューのルート。
ユーザー補助のタグと属性	
cfinput type="text" autosuggest="*bindexpression*"	選択候補を自動的に表示するドロップダウンボックス。ユーザーが入力したテキストに基づいて、完全な入力値の選択候補がリストに表示されます。
cftooltip タグ、および cfinput、cfselect、cftextarea コントロールの tooltip 属性	コントロールや領域に関するテキスト形式の説明。マウスのカーソルをコントロールまたは領域の上に置くと表示されます。
その他のタグ	
cfprogressbar	ファイルのダウンロードなどの進行状況を示す進行状況表示バー。
cfmap	ColdFusion Web ページ内の地図。
cfmediaplayer	ページに埋め込まれたメディアプレーヤー。
cfmessagebox	ポップアップメッセージを表示するためのコントロール。

タグや属性以外にも、表示の制御や管理を行うための JavaScript 関数が多数用意されています。その多くは、特定のタグの表示を制御するための関数です。たとえば、JavaScript 関数を使用して、ウィンドウの表示と非表示をダイナミックに切り替えることができます。また、ユーティリティ関数も用意されています。たとえば、コントロールの属性値を取得する ColdFusion.getElementValue 関数や、コンテナタグの URL の結果を表示する ColdFusion.navigate 関数などがあります。ColdFusion のすべての Ajax JavaScript 関数のリスト、および関数の詳細な説明については、『CFML リファレンス』の Ajax JavaScript Functions を参照してください。

ColdFusion Ajax のユーザーインターフェイス機能の使用

ColdFusion の Ajax ユーザーインターフェイス機能を使用すれば、データに応じてページをダイナミックに更新できるので、HTML ページを複数用意したり、ページを更新したり、HTML 以外の表示ツール (Flash フォームなど) を使用したりする必要がなくなります。多くのユーザーインターフェイス機能ではデータバインディングを使用して、フォームフィールドの値、フォームコントロールの選択値、Spry データセットの選択値などに基づいてデータをダイナミックに取得できます。

ColdFusion の Ajax ユーザーインターフェイスのコントロールおよび機能は、大きく次の 2 種類に分類できます。

- 表示レイアウト
- データ相関

表示レイアウトコントロールには、cflayout、cfpod、cfwindow コントロールなどがあります。データ相関機能には、HTML の cfgrid コントロール、cfmenu コントロール、テキスト入力コントロールの選択候補リストなどがあります。ほとんどの表示レイアウト機能とデータ相関機能ではデータバインディングを使用できるので、ユーザーとの間でダイナミックな対話処理が行えます。

ColdFusion の Ajax ユーザーインターフェイス機能は、Yahoo User Interface Library と Ext JavaScript Library に基づいています。また、cftextarea リッチテキストエディタは、FCKeditor テキストエディタに基づいています。ほとんどの場合、ColdFusion のタグと関数 (JavaScript 関数を含む) を使用するだけでインターフェイスの作成と管理を行えますが、上級者の場合は、ライブラリコード (特に CSS スタイル) を変更してコントロールの表示をさらにカスタマイズできます。

Ajax ユーザーインターフェイスレイアウトの制御

次のレイアウトタグを使用すると、表示をダイナミックに制御できます。

- cfdiv
- cflayout
- cfpod
- cfwindow

これらのタグを使用してフォームの内容を非同期で送信する方法については、803 ページの「Ajax コンテナによるフォーム送信」を参照してください。

cfdiv タグの使用

cfdiv タグは汎用のコンテナで、バインド式を使用してその内容を指定できます。したがって、バインドイベントが発生したときに、ページの任意の領域をダイナミックに更新することができます。デフォルトでは、HTML の div 領域が生成されますが、本文の内容を持つ他の HTML タグを生成することもできます。このタグでは、ColdFusion の他の Ajax コンテナタグと異なり、任意のバインド式 (CFC 関数、JavaScript 関数、URL、またはバインドパラメータを含む文字列) を使用してコンテナの内容を指定できます。したがって、cfdiv タグを使用すると、ページのダイナミックな生成がより柔軟に行えます。

cfdiv タグは、フォームを非同期で送信する場合にも便利です。これは、タグの生成にバインド式を使用しているかどうかには関係ありません。cfdiv タグ (バインド式から返された HTML も含む) 内のフォームを送信すると、フォームは非同期で送信され、その結果 cfdiv 領域にデータが設定されます。これは、cflayoutarea、cfwindow、および cfpod タグを使用した場合でも同じです。たとえば、アーティスト名を一覧表示し、ユーザーがアーティスト名を追加できるフォームを作成するとします。この場合、フォームを cfdiv タグ内に定義すると、ユーザーがフォームを送信したときに、ページ全体ではなく cfdiv タグ内の領域だけが更新されます。非同期フォームのコンテナコントロールの使用例については、803 ページの「Ajax コンテナによるフォーム送信」を参照してください。

cfdiv タグの使用例としては、cfgrid タグで社員のリストを表示するアプリケーションが考えられます。cfdiv タグのバインド式のパラメータにその cfgrid を含めることで、選択されているグリッド行の詳細を cfdiv タグに表示することができます。ユーザーがグリッド上で社員をクリックするたびに、cfdiv 領域にその社員の詳細が表示されます。

cfdiv タグのバインド式を使用してデータを取得する簡単な例を示します。この例では、バインディングを使用して、テキスト入力フィールドの内容を HTML の div 領域に表示しています。ユーザーが入力ボックスにテキストを入力した後に、Tab キーを押して別の場所に移動するか、アプリケーションの別の領域をクリックすると、入力したテキストが div 領域に表示されます。

メインアプリケーションファイルである "cfdivtag.cfm" ファイルの内容は、次のとおりです。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>cfdiv Example</title>
</head>

<body>
<cfform>
    <cfinput name="tinput1" type="text">
</cfform>

<h3> using a div</h3>
<cfdiv bind="url:divsource.cfm?InputText={tinput1}" ID="theDiv"
        style="background-color:##CCffFF; color:red; height:350"/>
</body>
</html>
```

div 領域の内容を定義する "divsource.cfm" ファイルのコードは、次のとおりです。

```
<h3>Echoing main page input:</h3>
<cfoutput>
    <cfif isdefined("url.InputText")>
        #url.InputText#
    <cfelse>
        No input
    </cfif>
</cfoutput>
```

レイアウトの使用

cflayout タグは、子の cflayoutarea 領域の外観と配置を制御します。cflayoutarea 領域には表示要素が含まれ、次のいずれかの方法で配置できます。

- 水平方向または垂直方向。
- 配置する場所を選択できる、境界線で囲まれたグリッドの中 (パネル状のレイアウト)。最大で 5 つ (上、下、左、右、中央) の領域を使用できます。中央以外の領域は、サイズ変更やクローズを許可するかどうかをオプションで設定できます。中央の領域の大きさは、他の領域で使用されていない領域として自動的に調整されます。各領域の表示と非表示はダイナミックに切り替えることができます。また、領域の折り畳み / 展開やクローズを許可するかどうかも設定できます。
- タブ表示。タブを選択すると表示領域が更新されて、そのタブのレイアウト領域の内容が表示されます。タブの表示と非表示を切り替えたり、タブの有効化と無効化を切り替えたりできます。また、タブのクローズを許可するかどうかもオプションで設定できます。

レイアウト領域のスクロールバーは、常に表示するか、領域内に内容が収まらない場合にのみ表示するか、常に表示しないかを選択できます。また、レイアウト領域からはみ出して内容を表示させることもできます。レイアウト領域の中にレイアウトをネストして、複雑なレイアウトを構築することもできます。

レイアウト領域の内容は cflayoutarea タグの本文で定義できますが、バインド式を使用して内容をダイナミックに取得することもできます。この場合、バインド式では CFC 関数を呼び出すか、CFML ページをリクエストするか、または JavaScript 関数を呼び出します。

ColdFusion にはレイアウトを管理するための JavaScript 関数が多数用意されており、境界線で囲まれた領域の折り畳みと展開、表示と非表示を切り替えたり、タブの作成、有効化、無効化、選択、表示、非表示を行ったりすることができます。すべての JavaScript 関数のリストについては、『CFML リファレンス』の Ajax JavaScript Functions を参照してください。

タブレイアウトの使用例を次に示します。この例では、JavaScript 関数を使用して、タブの有効化と無効化、および表示と非表示を切り替えています。

```
 <html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>

<body>
<!--- The tabheight attribute sets the height of all tab content areas and therefore the
            layout height. The width style controls the layout width. --->
<cflayout type="tab" name="mainTab" tabheight="300px" style="width:400px">

    <!--- Each layoutarea is one tab. --->
    <cflayoutarea title="First Tab" name="tab1">
    <h2>The First Tab</h2>
    <p>
    Here are the contents of the first tab.
    </p>
    </cflayoutarea>

    <cflayoutarea title="Second Tab" name="tab2">
    <h2>The Second Tab</h2>
    <p>
    This is the content of the second tab.
    </p>
    </cflayoutarea>
</cflayout>

<p>
Use these links to test selecting tabs via JavaScript:<br />
<a href="" onClick="ColdFusion.Layout.selectTab('mainTab','tab1');return false;">
    Click here to select tab 1.</a><br />
<a href="" onClick="ColdFusion.Layout.selectTab('mainTab','tab2');return false;">
    Click here to select tab 2.</a><br />
</p>

<p>
Use these links to test disabling/enabling via JavaScript. Notice that you cannot disable
    the currently selected tab.<br />
<a href="" onClick="ColdFusion.Layout.enableTab('mainTab','tab1');return false;">
    Click here to enable tab 1.</a><br />
<a href="" onClick="ColdFusion.Layout.disableTab('mainTab','tab1');return false;">
    Click here to disable tab 1.</a><br />
</p>

</body>
</html>
```

境界線で囲まれたレイアウトに子要素として cfpod を使用する例については、次のセクションを参照してください。タブレイアウトの別の例については、『CFML リファレンス』の cflayoutarea タグを参照してください。境界線で囲まれたレイアウトを、垂直方向のレイアウト領域の中にネストする例については、『CFML リファレンス』の cflayout を参照してください。

レイアウトのスタイル設定

cflayout タグと cflayoutarea タグには style 属性があります。cflayout タグの style 属性を使用すると、レイアウトコンテナのスタイルを制御し、レイアウト領域のほとんどのスタイルに対するデフォルト値を設定できます。たとえば、cflayout タグの文字色および背景色のスタイルによって、レイアウト領域のデフォルトの文字色と背景色が設定されますが、cflayout タグの border スタイルに設定した色はレイアウト全体を囲む境界線にのみ適用され、レイアウト領域の境界線には適用されません。cflayoutarea タグの style 属性を使用すれば、レイアウト領域ごとにスタイルを制御できます。ここで設定した内容は、cflayout タグの設定よりも優先されます。

コントロールが複雑になってくると、レイアウトやレイアウト領域で設定したスタイルが維持できなくなることがあります。したがって、たとえば、height スタイルは cflayout タグで指定せずに、それぞれの cflayoutarea タグで指定するのが一般的です。

次の例は、2 つのレイアウト領域で構成される簡単なタブレイアウトです。レイアウトの背景色は薄いピンクで、レイアウト領域にはピクセル幅の 3 つの赤い境界線が表示されます。

```
<cflayout name="layout1" type="tab" style="background-color:##FFCCCC">
    <cflayoutarea title="area1" style="border:3px solid red">
        Layout area 1
    </cflayoutarea>
    <cflayoutarea title="area1" style="border:3px solid red">
        Layout area 2
    </cflayoutarea>
</cflayout>
```

ポッドの使用

cfpod コントロールを使用すると、内容を表示するための領域を作成できます。この領域にはタイトルバーが表示され、領域全体は境界線で囲まれます。ポッドの内容は cfpod タグの本文で定義できます。または、バインド式を使用して URL からダイナミックに取得することもできます。ポッドは、Web のポータルインターフェイスを構成するポートレットなどの、画面を構成する複数の独立した (多くの場合インタラクティブな) 領域を作成する場合によく使用されます。

各ポッドのヘッダや本文のスタイルを制御するには、headerStyle 属性と bodyStyle 属性に CSS スタイルプロパティを指定します。

次の例では、cflayoutarea タグ内で複数のポッドを使用して、簡単なポータルを作成しています。時間ポッドでは、CFML ページから現在の時刻を取得しています。その他のポッドでは、わかりやすくするために cfpod の本文でその内容を定義しています。それぞれのポットで headerStyle および bodyStyle 属性を使用して、外観を制御しています。

"cfpodExample.cfm" アプリケーションのコードは次のとおりです。

```
<html>
<head>
</head>
<body>
<cflayout name="theLayout" type="border" style="height:300;">
    <cflayoutarea position="left" size="300" style="float:right;">
        <cfpod width="300" name="theNews" title="All the latest news"
                headerstyle="background-color:##DDAADD; font-size:large;
                    font-style:italic; color:black"
                bodyStyle="background-color:##FFCCFF; font-family:sans-serif;
                    font-size:80%">
            Contents of a news feed would go here.
        </cfpod>
    </cflayoutarea>
    <cflayoutarea position="center" align="center" >
        <cfpod name="theSports" width="500"
                title="What's new in your favorite sports"
                headerstyle="background-color:##AADDDD; font-size:large;
                    font-style:italic; color:black"
                bodyStyle="background-color:##CCFFFF; font-family:sans-serif;
                    font-size:90%">
            Contents of a sports feed would go here.
        </cfpod>
```

```
    </cflayoutarea>
    <cflayoutarea position="right" size="302">
        <cfpod width="300" height="20" name="thetime" title="The Weather"
                source="podweather.cfm"
                headerstyle="background-color:##DDAADD; font-style:italic;
                    color:black"
                bodyStyle="background-color:##FFCCFF; font-family:sans-serif;
                    font-size:80%" />
        <cfpod width="300" name="thestocks" title="What's new in business"
                headerstyle="background-color:##DDAADD; font-size:large;
                    color:black; font-style:italic"
                bodyStyle="background-color:##FFCCFF; font-family:sans-serif;
                    font-size:80%">
            Contents of a news feed would go here.
        </cfpod>
    </cflayoutarea>

</cflayout>
</body>
</html>
```

この例では、"podweather.cfm" ページの内容は次の 1 行だけです。より複雑な例では、フィードから天気予報をダイナミックに取得して、表示用に形式を設定することができます。

```
Partly Cloudy, 76 degrees
```

ポップアップウィンドウの使用

ColdFusion の HTML ポップアップウィンドウには次の特徴があります。

- タイトルバーが表示されます。
- ブラウザウィンドウの上に表示されます。ブラウザウィンドウ上の任意の位置に表示できます。
- モーダルにする (ポップアップウィンドウが表示されている間はメインウィンドウを操作できないようにする) ことも、非モーダルにする (どちらのウィンドウも操作できるようにする) こともできます。
- ポップアップウィンドウのドラッグ、クローズ、サイズ変更を許可するかどうかを指定できます。
- 作成したウィンドウは独立して表示できます。ウィンドウを作成したら、JavaScript 関数を使用することで、表示と非表示を何回でも切り替えることができます。表示するためにウィンドウを作成しなおす必要はありません。

ウィンドウの表示と非表示

ウィンドウは次の方法で表示することができます。

- cfwindow タグの initShow 属性の値を指定して、ウィンドウの作成と表示を行う方法。
- cfwindow タグの initShow 属性の値に false を指定し、JavaScript の ColdFusion.Window.show 関数を呼び出してウィンドウを表示する方法。
- JavaScript の ColdFusion.Window.create および ColdFusion.Window.show 関数を使用する方法。

ColdFusion.Window.hide 関数を呼び出して、現在表示しているウィンドウを非表示にすることができます。ColdFusion.Window.onShow および ColdFusion.Window.onhide 関数を使用すれば、ウィンドウが表示または非表示になったときに実行する JavaScript 関数を指定できます。

次の例に、ウィンドウの作成、表示、非表示を行う方法を示します。この例では、設定可能なオプションを使用して、ウィンドウのクローズ、ドラッグ、サイズ変更を許可するかどうかを設定しています。アプリケーションを実行すると、cfwindow タグによってウィンドウ 1 が作成され、表示されます。このウィンドウは、非表示と表示を切り替えることができます。ウィンドウ 2 を表示するには、[Create Window 2] ボタンをクリックし、次に [Show Window 2] ボタンをクリックします。このウィンドウも表示と非表示を切り替えることができます。

アプリケーションのメインページは次のようになります。

```
<html>
<head>
<script>
    <!--
    //Configuration parameters for window 2.
    var config =
    {x:250,y:300,height:300,width:300,modal:false,closable:false,
        draggable:true,resizable:true,initshow:false,minheight:200,minwidth:200
    }
    -->
</script>

</head>
<body>

<!--- Create a window with a title and show it. Don't allow dragging or resizing. --->
<cfwindow name="window1" title="CFML Window" draggable="false"
        resizable="false" initshow="true" height="250" width="250" x=375 y=0>
    <p>
    This content was defined in the cfwindow tag body.
    </p>
</cfwindow>

<form>
<!--- Use the API to show and hide Window 1. --->
    <input type="button" value="Show Window1"
            onClick="ColdFusion.Window.show('window1')">
    <input type="button" value="Hide Window1"
            onClick="ColdFusion.Window.hide('window1')"><br />

<!--- Use the API to create, show, and hide Window 2 --->
    <input type="button" value="Create Window2"
            onClick="ColdFusion.Window.create('window2', 'JavaScript Window',
            'window2.cfm', config)">
    <input type="button" value="Show Window2"
            onClick="ColdFusion.Window.show('window2')">
    <input type="button" value="Hide Window2"
        onClick="ColdFusion.Window.hide('window2')">
</form>

</body>
</html>
```

次のサンプルは、ウィンドウ 2 の内容を記述する "window2.cfm" ファイルの内容です。

```
<cfoutput>
<p>
This content was loaded into window 2 from a URL.<br />
</p>
</cfoutput>
```

ウィンドウの表示イベントおよび非表示イベントの使用

ウィンドウが表示または非表示になったときにトリガされる onShow および onHide イベントを使用して、アプリケーションを制御できます。イベントハンドラを指定するには、ColdFusion.Window.onShow および ColdFusion.Window.onHide 関数を呼び出します。どちらの関数も、パラメータとしてウィンドウ名とイベントハンドラ関数を取ります。イベントハンドラ関数は、唯一のパラメータとしてウィンドウ名を取ることができます。

次の例では、ウィンドウを表示または非表示にすると警告メッセージが表示されます。警告メッセージにはウィンドウ名を表示します。ウィンドウが初めて表示されるときには警告は表示されません。ウィンドウが初めて表示されるときは、cfwindow タグの initShow 属性が使用されるからです。ユーザーが [Toggle Window] または [Close] ボタンをクリックしてウィンドウを非表示にすると、警告メッセージが表示されます。

```
<html>
<head>
    <script language="javascript">
        //Boolean value tacking the window state.
        var shown=true;

        //Functions to display an alert box when
        function onshow(name) {
            alert("window shown = " + name);
        }
        function onhide(name) {
            alert("window hidden = " + name);
        }

        //Initialize the window show/hide behavior.
        function initWindow() {
            ColdFusion.Window.onShow("testWindow", onshow);
            ColdFusion.Window.onHide("testWindow", onhide);
        }

        //Show or hide the window, depending on its current state.
        function toggleWindow() {
            if (shown) {
                ColdFusion.Window.hide("testWindow");
                shown = false;
            }
            else {
                ColdFusion.Window.show("testWindow");
                shown = true;
            }
        }

    </script>
</head>
<!-- The body tag onLoad event calls the window show/hide initializer function. -->
<body onLoad="initWindow()">

<cfwindow name="testWindow" initshow=true title="test window" closable=true> Window contents
</cfwindow>

<cfform>
    <cfinput name="button" value="Toggle Window" onclick="javascript:toggleWindow()" type="button"/>
</cfform>
</body>
</html>
```

コントロールコンテナの内容

ColdFusion では、次に示すように、さまざまな方法でコンテナタグの内容の設定や変更を行えます。

- コンテナタグの source 属性 (cfdiv の場合は bind 属性) でバインド式を使用できます。バインドしたコントロールが変更されるたびに、コンテナがダイナミックに更新されます。
- ColdFuson.navigate 関数を使用すれば、特定のコンテナの本文を、指定した URL から返される内容に変更できます。この関数では、新しい内容をロードした後に追加の処理を行うコールバックハンドラや、エラーハンドラを指定できます。

コールバックハンドラは、navigate 関数の処理が正常に終了したことを通知するときに便利です。たとえば、navigate 関数を呼び出す前に、ロード中であることがわかるようにポッドのタイトルバーをイタリックに設定し、navigate 関数の処理が終了したらコールバックハンドラを使用して通常のタイトルバーに戻すことができます。同様に、ポッドに本のページを表示している場合は、コールバックハンドラーを使用して、別のフィールドに表示しているページ番号をページのロード後に更新できます。

- cfpod および cfwindow コントロールでは、**controlName**_body という特殊な変数を使用して、本文の内容にアクセスして変更することができます。たとえば、**controlName**_body.innerHTML プロパティを使用して、HTML の本文を設定できます。cfpod および cfwindow タグでは、**controlName**_title を使用して、コントロールのタイトルバーの内容を取得したり設定したりできます。

このように複数の方法が用意されているので、コードの記述が柔軟に行えます。たとえば、ColdFuson.navigate 関数を使用しても **controlName**_body 変数を使用しても同じような処理を実行できます。ただし、**controlName**_body 変数を使用する場合は、本文のマークアップを取得するために Ajax を明示的にリクエストする必要があります。また、取得したマークアップに含まれている JavaScript 関数が正しく機能するとは限りません。ColdFusion.navigate 関数を使用すれば、このような問題を回避できます。したがって、**controlName**_body 変数は、簡単な処理でのみ使用することをお勧めします。

次の例では、さまざまな方法でコンテナの内容を変更します。この例は、メインページと、ボタンをクリックしたときにメインページのウィンドウに表示するテキストが含まれている "windowsource.cfm" ページから構成されます。メインページは、1 つの cfpod コントロール、2 つの cfwindow コントロール、および次のボタンから構成されます。

- [単純な navigate] ボタン。このボタンをクリックすると、ColdFusion.navigate 関数が呼び出され、2 番目のウィンドウの内容が変更されます。
- [w2 の本文とタイトルを変更] ボタン。このボタンをクリックすると、2 番目のウィンドウの本文とタイトルの innerHTML 値が特定の文字列で置き換えられます。
- [ポッド本文を変更] ボタン。このボタンをクリックすると、ポッド本文の innerHTML が 2 番目のウィンドウのタイトルの innerHTML に変更されます。

メインページは次のようになります。

```
<html>
<head>
<!--- Callback handler puts text in the window.cfm callback div block. --->
<script language="javascript">
    var mycallBack = function(){
        document.getElementById("callback").innerHTML = "<br><br>
    <b>This is printed by the callback handler.</b>";
    }

<!--- The error handler pops an alert with the error code and message. --->
    var myerrorHandler = function(errorCode,errorMessage){
        alert("[In Error Handler]" + "\n\n" + "Error Code: " + errorCode + "\n\n" +
            "Error Message: " + errorMessage);
    }
</script>
</head>

<body>
<cfpod height="50" width="200" title="The Title" name="theTitle">
    This is a cfpod control.
</cfpod><br>

<!--- Clicking the link runs a ColdFusion.navigate function that replaces the second window's
            contents with windowsource.cfm. The callback handler then updates the window
            contents further. --->
<cfwindow name="w1" title="CF Window 1" initShow=true
        x=10 y=200 width="200">
    This is a cfwindow control.<br><br>
```

```
    <a href="javascript:ColdFusion.navigate('windowsource.cfm','w2',
        mycallBack,myerrorHandler);">Click</a> to navigate Window 2</a>
</cfwindow>

<cfwindow name="w2" title="CF Window 2" initShow=true
        x=250 y=200 width="200">
    This is a second cfwindow control.
</cfwindow>

<cfform>
    <!--- This button only replaces the second window body with the body of the
                windowsrc.cfm page. --->
    <cfinput type="button" name="button" value="Simple navigate"
            onClick="ColdFusion.navigate('windowsource.cfm','w2');">
    <!--- This button replaces the second window body and title content. --->
    <cfinput type="button" name="button2" value="Change w2 body & title"
            onClick="w2_body.innerHTML='New body inner HTML';w2_title.innerHTML=
                'New Title inner HTML'">
    <!--- This button puts the second window title in the pod body. --->
    <cfinput type="button" name="button3" value="Change pod body"
            onClick="theTitle_body.innerHTML=w2_title.innerHTML;">
</cfform>
</body>
</html>
```

"windowsource.cfm" ページは次のようになります。

```
This is markup from "windowsource.cfm"
<!--- The callback handler puts its output in the following div block. --->
<div id="callback"></div>
```

メニューとツールバーの使用

cfmenu および cfmenuitem タグを使用すると、メニューやツールバーを作成できます。

メニューとツールバーの定義

- メニューの全般的な特性は、cfmenu タグで定義します。
- ツールバーまたはメニューを作成するには、cfmenu の type 属性の値に horizontal または vertical を指定します。水平方向 (horizontal) の cfmenu がツールバーで、垂直方向 (vertical) の cfmenu がメニューです。
- cfmenu の下にサブメニューを作成できます。ただし、水平方向にメニューアイテムを並べられるのは、最上位のみです。つまり、ツールバーの子は、すべて垂直方向に並べられます。
- 最初に表示されるのは最上位のメニューです。親メニューに含まれているメニュールートにマウスのカーソルを合わせると、サブメニューが表示されます。
- メニューアイテムを作成するには、cfmenuitem タグを使用します。
- サブメニューを作成するには、cfmenuitem タグをネストします。親タグがサブメニューのルートになります。
- divider を除くすべての cfmenuitem タグには、必ず display 属性を指定します。この属性には、メニューアイテムとして表示するテキストを定義します。また、オプションで image 属性も定義できます。
- ツールバーでは、すべてのアイテムの間に区切り線が表示されます。メニューに区切り線を表示するには、divider 属性を指定した cfmenuitem タグを使用します。
- メニューアイテムをアクティブにするには、ユーザーがメニューアイテムをクリックしたときに呼び出す URL または JavaScript 関数を href 属性に指定します。

次の例では、サブメニューを持つ単純なツールバーを作成します。このサブメニューでは、表示内容を変更する JavaScript を使用しています。ユーザーがメニューの末端にあるアイテムを選択すると、メニューの下にある div ブロックに、選択したメニューのパスが表示されます。

```
 <html>
<head>
</head>
<body>

<!--- The selected function changes the text in the selectedItemLabel div block to show the
        selected item. --->
<script type="text/javascript">
    function selected(item) {
        var el = document.getElementById("selectedItemLabel");
        el.innerHTML = "You selected: " + item;
    }
</script>

<!--- A horizontal menu with nested submenus. Clicking an end item calls the selected
        function. --->
<cfmenu name="hmenu" bgcolor="##9999ff" selectedfontcolor="##0000dd"
        selecteditemcolor="##ddddff">
    <cfmenuitem display="Home" href="javascript:selected('Home');" />
    <cfmenuitem display="File">
        <cfmenuitem display="Open...">
            <cfmenuitem display="Template" href="javascript:selected('File &gt;
                Open... &gt; Template');" />
            <cfmenuitem divider="true" />
            <cfmenuitem display="CSS" href="javascript:selected('File &gt; Open... &gt;
                CSS');" />
        </cfmenuitem>
        <cfmenuitem display="Close" href="javascript:selected('File &gt; Close');" />
    </cfmenuitem>
    <cfmenuitem display="Help">
        <cfmenuitem display="About" href="javascript:selected('Help &gt; About');" />
    </cfmenuitem>
</cfmenu>

<!--- A div with initial text.
        The selected function changes the text by resetting the innerHTML. --->
<div style=" margin-top: 100; margin-left: 10;"><span id="selectedItemLabel">
    Please select an item!</span></div>

</body>
</html>
```

メニューのスタイル設定

cfmenu および cfmenuitem タグには、メニューの外観を簡単に制御できる属性がいくつか用意されています。それらの属性は、基本的な外観を指定する属性と、CSS スタイルを指定する属性の 2 種類に分かれます。cfmenu タグの fontColor 属性などの基本属性では、各メニューの特性を制御できます。CSS スタイル属性では、メニューの全体または一部の CSS スタイル仕様を指定できます。このセクションでは、CSS スタイル仕様がメニュースタイルに与える影響について説明します。スタイルに関連するすべての属性の詳細については、『CFML リファレンス』の cfmenu および cfmenuitem を参照してください。

cfmenu タグと cfmenuitem タグには、一連の CSS スタイル属性が用意されています。それらの属性は階層構造をなしており、それぞれ影響を与える範囲が異なります。次の表で、各属性を階層順に並べて説明します。

属性	説明
cfmenu の属性	
menuStyle	メニューに適用されます。メニューアイテムを囲むすべての部分も対象になります。cfmenu タグで childStyle 属性を指定していない場合や、cfmenuitem タグでスタイル情報を指定していない場合は、この属性によって最上位のアイテムのスタイルが制御されます。
childStyle	最上位のメニューアイテムと、すべての子のメニューアイテム (サブメニューの子を含む) に適用されます。この属性を使用すると、すべてのメニューアイテムに特定のスタイル仕様を適用できます。
cfmenuitem の属性	
style	現在のメニューアイテムにのみ適用されます。childStyle 属性よりも、この属性のほうが優先されます。
menuStyle	このメニューアイテムに含まれているすべてのサブメニューのスタイルを制御します。この属性は、現在のメニューアイテムのサブメニューには適用されますが、サブメニューに含まれている子のサブメニューには適用されません。
childStyle	現在のメニューアイテムに含まれているすべての子のメニューアイテムに適用されます。サブメニューに含まれている子も対象になります。

これらのスタイルの他にも、cfmenu タグで設定する bgcolor などの、スタイルに関連する属性も考慮します。

メニューを設計するときは、次の点に注意してください。

- フォントのサイズは 20 ピクセル以下にします。垂直方向のメニューでこれよりも大きいサイズに設定すると、メニューテキストが領域に収まらない可能性があります。
- スタイル属性間の関係を考慮します。メニューやサブメニューは、それぞれの子アイテムと周囲のメニュー領域から構成されているので、背景色の選択に注意してください。たとえば、cfmenu タグの menuStyle 属性と childStyle 属性で異なる background-color スタイルを指定した場合、どのメニューアイテムも同じ色になりますが、周囲のメニュー領域は違う色になります。

メニューにおけるスタイル属性の効果を示すサンプルについては、『CFML リファレンス』の cfmenuitem タグに掲載されている例を参照してください。

ColdFusion で提供されている属性を使用すれば、重要なスタイルオプションのほとんどを操作できますが、必要に応じて、すべてのメニューの基本的なメニュースタイルを変更することもできます。これを行うには、"yui.css" ファイルを開き、該当する CSS ファイルのセクションでメニュー関連のスタイルを編集します。"yui.css" ファイルは、"<Web のルートディレクトリ >/CFID/scripts/ajax/resources/yui" ディレクトリにあります (デフォルトの場合)。これらのスタイルの詳細については、Yahoo! User Interface Library のメニューに関する資料を参照してください。

ファイルのアップロード

cffileupload タグを使用すると、複数のファイルを選択してサーバーにアップロードできます。

cffileupload の使用

cffileupload タグは、複数のファイルをアップロードするためのダイアログボックスを表示します。次のようなファイルアップロード機能があります。

- コールバックハンドラとエラーハンドラを使用して、アップロードの完了後またはエラー発生時に、ファイルアップロード処理を制御できます。
- ファイルアップロード用のコントロールのスタイルを指定できます。
- エラー発生時にアップロードを停止するか続行するかを選択できます。
- コールバックハンドラまたはエラーハンドラにカスタムレスポンスを送信できます。

コールバックハンドラおよびエラーハンドラへのカスタムレスポンスの送信

次に示すように、サーバー上のアップロード操作を処理するページまたは URL を使用すると、ステータスとメッセージをキーとして含む struct をレスポンスとして送信できます。

```
<cffile action = "upload"
destination = "#Expandpath('./upload')#"
nameconflict="makeunique">
<cfset str.STATUS = 200>
<cfset str.MESSAGE = "File Upload Successful"><cfoutput>#serializeJSON(str)#</cfoutput>
```

次にエラーハンドラの例を示します。

```
<cftry>
<cffile action = "upload"
destination = "#Expandpath('./upload')#">
<cfcatch type="any">
<cfset str.STATUS = 500>
<cfset str.MESSAGE = "Error occurred while uploading the file"><cfoutput>#serializeJSON(str)#</cfoutput>
</cfcatch>
</cftry>
```

アップロード先のディレクトリに既に存在するファイルと同名のファイルをアップロードしようとすると、エラーが発生します。ステータスとメッセージは catch ブロックで指定した値に設定されます。

スタイルの使用

ファイルアップロード用のコントロールのスタイルを指定するには、headercolor、textcolor、bgcolor、titletextalign、titletextcolor、および rollovercolor 属性を使用します。

次の例は、ファイルアップロード用のコントロールのスタイルを指定する方法を示しています。

```
<cffileupload
          url="uploadAll.cfm"
          name="myuploader3"
          align="right"

style="headercolor:silver;textcolor:1569C7;titletextalign:right;titletextcolor:black;bgcolor:74BBFB;"/>
```

次のコードは、onUploadComplete 属性の使用方法を示しています。

```
<!--- upload.cfm --->
<!--- <cffile action = "upload" destination = "#Expandpath('./upload')#" nameconflict="makeunique"> --->
<script language="javascript">
    var uploadCompleteHandler = function(obj){
        var result = "Upload Details:" + "\n\n";
        for(var i=0;i < obj.length; i++){
            result = result + "FILENAME: " + obj[i].FILENAME + "\n" + "STATUS: " + obj[i].STATUS + "\n" +
"MESSAGE: " + obj[i].MESSAGE + "\n\n";
        }
    }
</script>
<br>
<cffileupload
    url="uploadall.cfm"
    name="myuploader"
    onUploadComplete="uploadCompleteHandler"
    maxUploadSize=100
    stopOnError=false
    />
```

デフォルトでの（MIME タイプの検証による）アップロードのセキュリティ保護

cffile タグで upload および uploadAll アクションを使用する際に、次のことを行えるように機能が強化されました。

- accept 属性を使用できます。
- strict=true（デフォルト）を使用して、ファイルの適切な MIME タイプをチェックできます。

注意：マルチファイルアップローダーを使用した場合は、アップローダーでステータスが「完了」と表示されていても、アップロードに失敗したインスタンス（例えば、MIME タイプや拡張子のチェックで失敗したもの）が存在する可能性があります。このようなシナリオの場合は、アップロードページでエラーを捕捉して、シリアル化した構造体の MESSAGE と STATUS のキーにエラーの状態を設定して返す必要があります。

Ajax フォームのコントロールおよび機能の使用

ColdFusion の HTML フォームやコントロールには、Ajax ベースの機能が実装されています。

- cfgrid、cfinput、cfselect、cftextarea、および cftree コントロールではバインディングがサポートされており、コントロールの内容を取得できます。
- ColdFusion 関数では、フォームの非同期送信がサポートされています。この機能を使用すれば、ページ全体を更新せずに済みます。Ajax コンテナコントロールの中にあるフォームは、自動的に非同期で送信されます。また、JavaScript の ColdFusion.Ajax.SubmitForm 関数と Ajax プロキシの setForm 関数を使用すれば、手動で非同期送信を行えます。
- cfgrid および cftree タグを使用すれば、HTML のグリッドとツリーを作成できるので、Java アプレットや Flash を使用する必要はありません。
- cftextarea コントロールでは、リッチテキストエディタのオプションを使用できます。このテキストエディタは設定可能です。
- cfinput タグでは datefield というタイプがサポートされており、Ajax ベースのポップアップカレンダーから日付を選択することができます。
- cfinput タグの text タイプでは autosuggest 属性がサポートされており、ユーザーが入力したテキストに基づいてドロップダウンリストに入力候補を表示することができます。
- cfinput、cfselect、および cftextarea タグでは tooltip 属性がサポートされており、ユーザーがコントロールの上にマウスのカーソルを合わせたときに表示するツールヒントを指定できます。cftooltip タグを使用すれば、フォームコントロールだけでなく、ページの任意の部分でツールヒントを表示できます。

Ajax フォームコントロールの使用

ColdFusion の Ajax ベースのフォームコントロールを使用すれば、ページ全体を更新せずに Ajax フォームを送信できます。

注意：Ajax を使用してフォームをページから非同期で送信する場合は、cfinput または input タグを使用してファイルをアップロードすることはできません。

Ajax コンテナによるフォーム送信

ColdFusion の Ajax コンテナタグである cfdiv、cflayoutarea、cfpod、cfwindow タグでは、その中に含まれているフォームは自動的に非同期で送信されます。フォームが送信されると、アクションページから結果が返されて、コンテナの内容が更新されますが、ページのそれ以外の領域は変化しません。

この処理を行う "submitSimple.cfm" ページの例を次に示します。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
<body>
<cflayout type="vbox" name="layout1">
    <cflayoutarea>
        <h3>This area is not refreshed when the form is submitted.</h3>
        <br />
    </cflayoutarea>

    <cflayoutarea>
        <h3>This form is replaced by the action page</h3>
        <cfform name="myform" format="html" action="showName.cfm">
            <cfinput type = "Text" name = "name">
            <cfinput type = "submit" name = "submit" value = "Enter name">
        </cfform>
    </cflayoutarea>
</cflayout>

</body>
</html>
```

次の例では、テキスト入力に名前を入力して [名前を入力] ボタンをクリックすると、入力したテキストがページのフォームに表示されますが、それ以外の部分は更新されません。"showName.cfm" アクションページは次のようになります。

```
<cfif IsDefined("Form.name")>
    <cfoutput>The Name is : <strong>#Form.name#</strong></cfoutput>
</cfif>
```

cfajaxproxy の SetForm 関数の使用

cfajaxproxy タグによって作成されるプロキシオブジェクトには、SetForm という関数が用意されています。SetForm 関数を呼び出すと、その次に呼び出す CFC 関数に、フォームの値が引数として渡されます。この方法を使用すれば、特定の CFC 関数にフォームの現在のフィールド値を渡して必要な処理を行い、結果を返すことができます。

SetForm 関数を使用する場合、CFC 関数に渡される引数には次のルールが適用されます。

- cfargument タグでフォームフィールドを指定しなくても、フィールド名を介してフィールド値を取得できます。
- フォームフィールドと同じ名前の CFC 引数がある場合、その引数はフォームフィールド値によって上書きされます。
- cfargument タグでフォームフィールドを指定しなかった場合、arguments 構造体で位置引数の表記法 (配列表記法) を使用するときは注意が必要です。明示的に宣言された引数の後にフォームフィールドがあるとは限りません。
- CFC 関数の arguments スコープには、ColdFusion がその動作を制御するために使用する 2 つのフィールドが含まれています。それらは内部的に使用されるフィールドであり、その名前は今後のリリースで変更される可能性があります。現在の名前は次のとおりです。どちらも true に設定されています。
    - _CF_NODEBUG。true の場合、ColdFusion は呼び出しの応答でデバッグ出力を返しません。
    - _CF_NOCACHE。true の場合、ColdFusion は応答でノーキャッシュヘッダを送信します。これによって、ブラウザに応答がキャッシュされなくなるので、Ajax リクエストのたびに必ずネットワークコールが実行されるようになります。

次の例で、SetForm 関数を使用してログインフォームの内容を送信する方法を示します。ユーザーが [Login!] ボタンをクリックすると、doLogin 関数によってプロキシの setForm 関数が呼び出され、その後に AuthenticationSystem.cfc validateCredentials メソッドが呼び出されます。validateCredentials メソッドはユーザーのパスワードを照合し、有効である場合はプロキシに true を返します。doLogin メソッドは戻り値を取得し、値が true であればログインウィンドウを閉じます。プロキシは同期式 (デフォルト) で動作するので、これらの処理が完了するまでユーザーはページの内容にアクセスできません。戻り値が false の場合は、ログインウィンドウのタイトルバーにメッセージを表示します。

"setForm.cfm" アプリケーションは次のようになります。

```
<html>
<head>
        <script type="text/javascript">
            function doLogin() {
                // Create the Ajax proxy instance.
                var auth = new AuthenticationSystem();

                // setForm() implicitly passes the form fields to the CFC function.
                auth.setForm("loginForm");
                //Call the CFC validateCredentials function.
                if (auth.validateCredentials()) {
                    ColdFusion.Window.hide("loginWindow");
                } else {
                    var msg = document.getElementById("loginWindow_title");
                    msg.innerHTML = "Incorrect username/password. Please try again!";
                }
            }
        </script>
</head>

<body>
<cfajaxproxy cfc="AuthenticationSystem" />

<cfif structKeyExists(URL,"logout") and URL.logout>
    <cflogout />
</cfif>

<cflogin>
    <cfwindow name="loginWindow" center="true" closable="false"
                draggable="false" modal="true"
                title="Please login to use this system"
                initshow="true" width="400" height="200">
        <!--- Notice that the form does not have a submit button.
            Submit action is performed by the doLogin function. --->
        <cfform name="loginForm" format="xml">
            <cfinput type="text" name="username" label="username" /><br />
            <cfinput type="password" name="password" label="password" />
            <cfinput type="button" name="login" value="Login!" onclick="doLogin();" />
        </cfform>
    </cfwindow>
</cflogin>

<p>
This page is secured by login.
You can see the window containing the login form.
The window is modal; so the page cannot be accessed until you log in.
<ul>
    <li><a href="setForm.cfm">Continue using the application</a>!</li>
    <li><a href="setForm.cfm?logout=true">Logout</a>!</li>
</ul>
</p>
</body>
</html>
```

"AuthenticationSystem.cfc" ファイルを次の例に示します。

```
<cfcomponent output="false">

    <cffunction name="validateCredentials" access="remote" returntype="boolean"
            output="false">
        <cfargument name="username" type="string"/>
        <cfargument name="password" type="string"/>

        <cfset var validated = false/>
        <!--- Ensure that attempts to authenticate start with new credentials. --->
        <cflogout/>

        <cflogin>
            <cfif arguments.username is "user" and arguments.password is "secret">
                <cfloginuser name="#arguments.username#"
                        password="#arguments.password#" roles="admin"/>
                <cfset validated = true/>
            </cfif>
        </cflogin>

        <cfreturn validated/>
    </cffunction>
</cfcomponent>
```

ColdFusion.Ajax.submitForm 関数の使用

ColdFusion.Ajax.submitForm 関数を使用すると、フォームの内容を CFML ページなどのアクティブページにいつでも送信できます。たとえば、この関数を使用して、入力が完了していないフォームを自動的に保存できます。この関数はフォームへの添付ファイルのアップロードをサポートしていません。

この関数を使用するときは、送信するフォームの名前と、フォームを処理するページの URL を関数に渡します。また、次のオプションのパラメータも指定できます。

- 返される結果を処理するコールバック関数
- エラーハンドラ (HTTP エラーコードとエラーメッセージの 2 つのパラメータを取ります)
- HTTP メソッド (デフォルトでは POST)
- フォームを非同期で送信するかどうか (デフォルトでは true)

次の例では、ColdFusion.Ajax.submitForm 関数を使用して、2 つのフォームフィールドを "asyncFormHandler.cfm" ページに送信します。このページは、フォームの値をそのまま表示します。コールバックハンドラを使用してウィンドウを開き、返された情報を表示します。

```
<html>
<head>
<!--- The cfajaximport tag is required for the submitForm function to work
            because the page does not have any Ajax-based tags. --->
<cfajaximport>

<script>
    function submitForm() {
        ColdFusion.Ajax.submitForm('myform', 'asyncFormHandler.cfm', callback,
            errorHandler);
    }

    function callback(text)
    {
        alert("Callback: " + text);
    }

    function errorHandler(code, msg)
    {
        alert("Error!!! " + code + ": " + msg);
    }
</script>

</head>
<body>

<cfform name="myform">
    <cfinput name="mytext1"><br />
    <cfinput name="mytext2">
</cfform>

<a href="javascript:submitForm()">Submit form</a>
</body>
</html>
```

"asynchFormHandler.cfm" ページの内容は次の 1 行です。

```
<cfoutput>Echo: #form.mytext1# #form.mytext2#</cfoutput>
```

ColdFusion.navigate 関数によるフォーム送信

JavaScript の ColdFusion.navigate 関数を使用すると、URL にフォームを送信し、返された結果を特定のコンテナコントロール (cfdiv、cflayout、cfpod、cfwindow タグなど) に表示できます。ユーザーがデータを送信したフォームが含まれているコントロール以外のコントロールに結果を表示できます。また、この関数を使用すると、ユーザーがフォームの外部で何らかのアクションを行ったとき (たとえばメニューアイテムをクリックしたとき) に、フォームを非同期で送信できます。

この関数の使用例については、『CFML リファレンス』で ColdFusion.navigate 関数を参照してください。

HTML グリッドの使用

ColdFusion には HTML の cfgrid コントロールが用意されており、バインド式を使用してダイナミックにデータを設定できます。バインド式を使用する HTML のグリッドは、ページ形式で生成されます。ユーザーがグリッドのページを切り替えると、移動先のページに必要な分のデータがデータソースから取得されてグリッドに表示されます。ユーザーがフォームの内容を編集するときにもバインド式を使用できます。また、その他の ColdFusion コントロールもグリッドにバインドできます。HTML のグリッドには、グリッドの管理や操作を行うための JavaScript 関数が用意されています。

また、スタティックな HTML のグリッドも作成できます。それには、バインド式を指定せずに cfgrid タグを使用します。スタティックなグリッドでは、すべてのデータがまとめて表示されます。

HTML モードでは、マウスカーソルを列ヘッダ上に置くと、下矢印ボタンが表示されます。このボタンをクリックすると、次のオプションが含まれたリストが表示されます。

- 列の内容に基づいて昇順または降順でグリッドをソートする。
- 表示する列を選択する。
- グリッドに groupfield 属性がある場合 : グループ化の無効と有効を切り替えて列の値でグループ化する。

HTML グリッドで selectMode="edit" を指定すると、グリッドの下部にあるバーに、[挿入]、[保存]、[キャンセル]、[削除] の各ボタンが表示されます。[挿入] ボタンは、編集可能な新しい行を開きます。[保存] ボタンは、バインドソースに変更をコミットします。[キャンセル] ボタンは、保存されていない変更をロールバックします。[削除] ボタンは、選択されている行を削除します。[削除] ボタンをクリックした後に [保存] ボタンをクリックする必要はありません。

フォームデータのダイナミックな設定

HTML グリッドでは、bind 属性にバインド式を指定することで、グリッドデータをダイナミックに設定できます。バインド式では、CFC 関数、JavaScript 関数、または URL を呼び出します。また、バインドパラメータを指定して、グリッドにダイナミックに表示する情報や、他のフォームフィールドの属性値を指定することができます。

バインド式には、次のバインドパラメータを渡します。関数呼び出しまたは URL のパラメータを 1 つでも省略すると、エラーが発生します。これらのパラメータを渡すことにより、グリッドやその状態に関する情報がデータ生成関数に通知されます。パラメータの実際のデータは自動的に供給されます。手動で値を設定することはできません。

パラメータ名	説明
cfgridpage	データを取得するページの番号。
cfgridpagesize	ページに表示するデータの行数。このパラメータの値は、pageSize 属性の値です。
cfgridsortcolumn	グリッドのソート順序の基準となる列の名前。ユーザーが列のヘッダをクリックするまで、この値は設定されません。
cfgridsortdirection	ソートの方向。ASC (昇順) または DESC (降順) のいずれかです。ユーザーが列のヘッダをクリックするまで、この値は設定されません。

注意：ユーザーまたはアプリケーションによってグリッドのソート (グリッドの列ヘッダのクリックなど) がまだ実行されていない場合、cfgridsortcolumn および cfgridsortdirection パラメータは空になります。

バインディングとバインドパラメータの詳細については、『CFML リファレンス』の 839 ページの「Ajax データ機能および開発機能の使用」を参照してください。

オプションのパラメータを使用して、呼び出す関数に追加の情報を渡すことができます。これは、返すデータを決定するために必要なデータを提供する場合に使用します。たとえば、市の名前を返す関数に、都道府県の名前を渡す場合などが考えられます。オプションの関数パラメータは、その一部または全部をバインドパラメータとして指定できます。たとえば、都道府県を選択する cfselect コントロールから都道府県名を取得することができます。

他のコントロールが変更されてもグリッドを自動的に更新したくない場合は、すべてのオプションのバインドパラメータに @none 指定子を使用します。これによって、バインドしているコントロールの値が変更されても、グリッドが自動更新されなくなります。グリッドの内容を明示的に更新するには、JavaScript の ColdFusion.Grid.refresh 関数を使用します。@none 指定子の使用方法と、コントロールを明示的に更新する方法の詳細については、842 ページの「バインドパラメータの指定」を参照してください。

ユーザーがデータをソートできる場合 (sort 属性が true である場合)、バインド式から呼び出された関数では、バインドパラメータの cfgridsortcolumn と cfgridsortdirection の値を参照してソート方法を確認し、適切にソートしたデータを返す必要があります。ユーザーによるソートを許可していない場合でも、これらのパラメータを関数に渡す必要があります。渡さないとエラーが発生します。また、これらのパラメータが空の文字列である場合にも対処できるように、関数またはアクションページを記述する必要があります。グリッドをソートするためにユーザーが列ヘッダを選択するか、JavaScript の ColdFusion.Grid.sort 関数を呼び出すまで、これらの値は設定されないからです。

返すデータの形式は、データの取得方法によって異なります。

バインドのタイプ	戻り値
CFC	ColdFusion の構造体。呼び出し元に合わせて自動的に変換されます。または、JSON 形式の構造体を返すこともできます。
URL	JSON 形式の構造体。本文にその他の内容を含めることはできません。
JavaScript	JavaScript オブジェクト。

bind 属性に CFC を指定した場合は、queryConvertForGrid 関数を使用してクエリーを構造体に直接変換し、CFC の戻り値として使用することができます。

bind 属性に CFML ページを指定した場合は、queryConvertForGrid 関数を使用してクエリーを構造体に変換し、serializeJSON 関数を使用してその構造体を JSON 形式に変換します。

JavaScript オブジェクトまたはその JSON 形式を手動で作成する場合、その最上位には次の 2 つのキーが必要です。

- TOTALROWCOUNT : 返すクエリーデータセットに含まれている行の数。この値は、グリッド内の全ページに含まれているデータ行の数です。現在のページの行数ではありません。
- QUERY : 返すクエリーの内容。QUERY の値は、次の 2 つキーを持つオブジェクトであることが必要です。
  - COLUMNS : 列名の配列。
  - DATA : COLUMNS の各列の値が格納された配列を要素とする 1 次元配列。

注意：戻り構造体を CFC で手動で作成する場合は、QUERY の値に ColdFusion のクエリーオブジェクトを使用できます。これは、ColdFusion によってリモートアクセス用の形式に自動的に変換されます。

たとえば次のオブジェクトは、cfgrid タグにデータを提供するために JavaScript のバインド関数から返すことができます。

```
var myobject =
    {"TOTALROWCOUNT":6,"QUERY":{"COLUMNS":["EMP_ID","FIRSTNAME",
        "EMAIL"],"DATA":[[1,"Carolynn","CPETERSON"],
        [2,"Dave","FHEARTSDALE"], [3,"Linda","LSTEWART"],
        [4,"Aaron","ASMITH"], [5,"Peter","PBARKEN"],
        [6,"Linda","LJENNINGS"],]}};
```

次の例では、バインド式と CFC を使用して、ページ化されたデータグリッドにダイナミックにデータを設定します。CFML ページのフォームは次のとおりです。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>

<body>
<cfform name="form01">
    <cfgrid format="html" name="grid01" pagesize=5 sort=true
            bind="cfc:places.getData({cfgridpage},{cfgridpagesize},
            {cfgridsortcolumn},{cfgridsortdirection})">
        <cfgridcolumn name="Emp_ID" display=true header="eid" />
        <cfgridcolumn name="FirstName" display=true header="Name"/>
        <cfgridcolumn name="Email" display=true header="Email" />
    </cfgrid>
</cfform>
</body>
</html>
```

"places.cfc" ファイルは次のようになります。関数が呼び出されるたびにすべてのデータセットが取得され、QueryConvertForGrid 関数によってページに必要なデータのみが抽出され返されます。

```
<cfcomponent>
    <cffunction name="getData" access="remote" output="false">
        <cfargument name="page">
        <cfargument name="pageSize">
        <cfargument name="gridsortcolumn">
        <cfargument name="gridsortdirection">
        <cfquery name="team" datasource="cfdocexamples">
            SELECT Emp_ID, FirstName, EMail
            FROM Employees
            <cfif gridsortcolumn neq "" or gridsortdirection neq "">
                order by #gridsortcolumn# #gridsortdirection#
            </cfif>
        </cfquery>
        <cfreturn QueryConvertForGrid(team, page, pageSize)>
    </cffunction>
</cfcomponent>
```

次の例は、前の例と同じ処理を行っていますが、メインページで URL バインド式を使用し、データを返すために CFML ページを使用している点が異なります。

メインページのフォームは次のとおりです。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>

<body>
<cfform name="form01">
    <cfgrid format="html" name="grid01" pagesize=5 sort=true
        bind="url:getdata.cfm?page={cfgridpage}&pageSize={cfgridpagesize}
            &sortCol={cfgridsortcolumn}&sortDir={cfgridsortdirection}">
        <cfgridcolumn name="Emp_ID" display=true header="eid" />
        <cfgridcolumn name="FirstName" display=true header="Name"/>
        <cfgridcolumn name="Email" display=true header="Email" />
    </cfgrid>
</cfform>
</body>
</html>
```

"getdata.cfm" ページを次の例に示します。

```
<!--- Empty string; the default end of the query SQL. --->
<cfset queryEnd="">

<cfquery name="team" datasource="cfdocexamples">
    SELECT Emp_ID, FirstName, EMail
    FROM Employees
    <cfif sortcol neq "" or sortdir neq "">
        order by #sortcol# #sortdir#
    </cfif>
</cfquery>

<!--- Format the query so that the bind expression can use it. --->
<cfoutput>#serializeJSON(QueryConvertForGrid(team, page, pageSize))#
</cfoutput>
```

データベースに対して SQL クエリーを実行できる場合は、ページに必要なデータのみをクエリーで取得することによって、効率化を図ることができます。QueryConvertForGrid 関数を使用せずに、戻り構造体を手動で作成して 1 ページ分のデータのみを返します。TotalRowCount フィールドには、データセット全体の行数を設定します。返すページに含めるデータの行数ではないので注意してください。

bindOnLoad 属性の使用

通常は、イベントがトリガされてからコントロールのバインド式が実行されますが、bindOnLoad 属性を使用すると、コントロールがロードされた直後にバインド式が実行されます。これを利用すれば、コントロールに初期値を設定できます。この属性を持つすべての Ajax コントロールでは、この属性がデフォルトで false になっています (ただし、cfdiv および cfgrid は例外で、デフォルトで true になっています)。これらのコントロールで bindOnLoad の値を true に設定すれば、ロード時にコントロールの初期値が設定されます。

bindOnLoad 属性が true であるコントロールが 2 つあり、一方が他方をバインドしている場合は、まず onLoad ページイベントの発生時に両方のコントロールがロードされます。ロードが完了すると、バインドされているコントロールからバインドしているコントロールに変更イベントが送信されるので、バインドしているコントロールが再度ロードされます。開発者としては、どちらのコントロールも 1 回ずつ呼び出して初期値を設定したかったのですが、実際には、バインドされているコントロールがロードを完了すると、バインドしているコントロールが 2 回呼び出されてしまいます。

cfinput、cfselect、および cftextarea コントロールの bindOnLoad 属性はデフォルトで false になっているので、bindOnLoad 属性を明示的に設定しない限り、cfgrid または cfdiv タグでこれらのコントロールをバインドしても問題は発生しません。ただし、これらのコントロールの bindOnLoad 属性を true に設定している場合は、これらのコントロールが返すデータのみを取得するように、cfgrid または cfdiv 属性を false に設定します。

グリッドが Spry データセットにバインドされている場合も、二重ロードが発生することがあります。デフォルトでは、ページがロードされるとグリッドおよびデータセットにデータがロードされますが、最初の行にフォーカスが設定されるとデータセットからグリッドに選択変更イベントが送信されるので、グリッドがデータを再度ロードします。bindOnLoad 属性を false に設定すれば、データセットからグリッドに選択の change イベントが送信されたときにのみグリッドにデータがロードされるようになります。

グリッドの内容のダイナミックな編集

バインド式を使用して cfgrid のデータをダイナミックに取得している場合は、ユーザーがフォームを送信しなくても、入力データを使用してデータソースをダイナミックに更新したり、データソースのデータをダイナミックに削除することができます。cfgrid のデータを編集するには、フィールドの内容を選択してから新しい値を入力します。行を削除するには、行のフィールドを選択し、グリッドの一番下にある削除ボタンをクリックします。

バインド式を使用しているグリッドに新しい行を直接挿入することはできません。行を挿入するには、フォームにデータを入力します。フォームを送信するとグリッドが更新されます。

データをダイナミックに更新または削除するには、次の作業を行います。

- cfgrid タグで selectmode="edit" と指定します。これでユーザーがグリッドを編集できるようになります。
- cfgrid タグで onChange 属性を指定します。この属性のバインド式には、データソースを更新する CFC メソッド、JavaScript 関数、またはページの URL を指定する必要があります。このバインド式の形式は 808 ページの「フォームデータのダイナミックな設定」で説明した形式と同じですが、グリッドから自動的に渡される、次のバインドパラメータを指定する必要があります。これらのパラメータを渡すことにより、グリッドやその状態に関する情報が onChange 関数に通知されます。

パラメータ名	説明
cfgridaction	グリッドで実行するアクション。更新は 'U'、削除は 'D' です。
cfgridrow	構造体または JavaScript オブジェクト。キーは列の名前で、値は更新または削除する行の元の値です。
cfgridchanged	エントリが 1 つの構造体または JavaScript オブジェクト。キーは値を変更する列の名前で、値はフィールドの新しい値です。グリッドのアクションが削除の場合、この構造体は存在しますが空になります。

データをダイナミックに更新する場合、onError 属性を使用すれば、エラーが発生して CFC または URL から HTTP エラーステータスが返されたときに実行する JavaScript 関数の名前を指定できます。このメソッドは、HTTP エラーコードと、エラーの内容を説明するテキストメッセージの 2 つのパラメータをが取ります。次の例では、onError ハンドラ関数の使用方法を示します。

```
<script type="text/javascript">
    function errorhandler(id,message) {
        alert("Error while updating \n Error code: "+id+" \nMessage:
        "+message);}
</script>
```

次の例では、特定の部署に所属する社員を表示し、ユーザーがフィールドのデータを編集できるようにします。フォーカスが編集対象のフィールドから移動すると、onChange イベントがトリガされ、CFC の editData 関数が呼び出されてデータソースが更新されます。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript">
    function errorhandler(id,message) {
        alert("Error while updating\n Error code: "+id+"\n Message: "+message);
    }
</script>
</head>

<body>
<cfform name="form01">
    <cfgrid format="html" name="grid01" pagesize=11
            stripeRows=true stripeRowColor="gray"
            bind="cfc:places.getData({cfgridpage},{cfgridpagesize},
                {cfgridsortcolumn},{cfgridsortdirection})"
            delete="yes" selectmode="edit"
            onchange="cfc:places.editData({cfgridaction},{cfgridrow},{cfgridchanged})">
        <cfgridcolumn name="Emp_ID" display=true header="Employee ID"/>
        <cfgridcolumn name="FirstName" display=true header="Name"/>
        <cfgridcolumn name="Email" display=true header="Email"/>
    </cfgrid>
</cfform>
</body>
</html>
```

getData 関数は、808 ページの「フォームデータのダイナミックな設定」で説明した getData 関数と同じです。CFC の editData 関数は次のようになります。

```
<cffunction name="editData" access="remote" output="false">
    <cfargument name="gridaction">
    <cfargument name="gridrow">
    <cfargument name="gridchanged">

    <cfif isStruct(gridrow) and isStruct(gridchanged)>
        <cfif gridaction eq "U">
            <cfset colname=structkeylist(gridchanged)>
            <cfset value=structfind(gridchanged,#colname#)>
            <cfquery name="team" datasource="cfdocexamples">
                update employees set <cfoutput>#colname#</cfoutput> =
                    '<cfoutput>#value#</cfoutput>'
                where Emp_ID = <cfoutput>#gridrow.Emp_ID#</cfoutput>
            </cfquery>
        <cfelse>
            <cfquery name="team" datasource="cfdocexamples">
                delete from employees where emp_id = <cfoutput>#gridrow.Emp_ID#
                    </cfoutput>
            </cfquery>
        </cfif>
    </cfif>
</cffunction>
```

コントロールへのグリッドの内容のバインディング

グリッドフィールドのデータをフォームコントロールにバインドするには、フォームコントロールの bind 属性の値としてバインドパラメータを指定します。そのシンタックスは次のようになります。

```
<cfinput name="name" type="text" bind="{gridName.columnName}">
```

デフォルトでは、グリッドの選択行が変わるたびにバインドパラメータが再評価されて、選択されたグリッドセルの該当する列の値がコントロールに反映されます。

JavaScript のグリッド関数

次の JavaScript 関数を使用して、HTML 形式のグリッドを管理できます。

関数	説明
ColdFusion.Grid.getGridObject	Ext JS JavaScript ライブラリオブジェクトを取得します。
ColdFusion.Grid.refresh	グリッドの表示を手動で更新します。
ColdFusion.Grid.sort	グリッドをソートします。

詳細については、『CFML リファレンス』で ColdFusion.Grid.getGridObject、ColdFusion.Grid.refresh、および ColdFusion.Grid.sort の各関数を参照してください。

ブール型の列の使用とグループ化

次の例では、ブール型の列の使用方法を示します。このコードは、選択したグリッド列でグリッドのデータをグループ化する方法も示しています。

この例では、すべてのタイプのブール型表現を使用しています。グループ化のために、groupField が active に設定されているので、それに従ってグリッドデータがグループ化されます。

```
<cfset emps = querynew("firstname,department, salary,active")>
<cfset queryaddrow(emps,10)>
<cfset querysetcell(emps,"firstname","Debra",1)>
<cfset querysetcell(emps,"department","Accounting",1)>
<cfset querysetcell(emps,"salary","100000",1)>
<cfset querysetcell(emps,"active","Y",1)>
<cfset querysetcell(emps,"firstname","Doherty",2)>
<cfset querysetcell(emps,"department","Finance",2)>
<cfset querysetcell(emps,"salary","120000",2)>
<cfset querysetcell(emps,"active","Yes",2)>
<cfset querysetcell(emps,"firstname","Ben",3)>
<cfset querysetcell(emps,"department","Law",3)>
<cfset querysetcell(emps,"salary","200000",3)>
<cfset querysetcell(emps,"active","true",3)>
<cfset querysetcell(emps,"firstname","Aaron",4)>
<cfset querysetcell(emps,"department","Accounting",4)>
<cfset querysetcell(emps,"salary","200000",4)>
<cfset querysetcell(emps,"active","1",4)>
<cfset querysetcell(emps,"firstname","Josh",5)>
<cfset querysetcell(emps,"department","CF",5)>
<cfset querysetcell(emps,"salary","400000",5)>
<cfset querysetcell(emps,"active",true,5)>
<cfset querysetcell(emps,"firstname","Peterson",6)>
<cfset querysetcell(emps,"department","Accounting",6)>
<cfset querysetcell(emps,"salary","150000",6)>
<cfset querysetcell(emps,"active","0",6)>
<cfset querysetcell(emps,"firstname","Damon",7)>
<cfset querysetcell(emps,"department","Finance",7)>
<cfset querysetcell(emps,"salary","100000",7)>
<cfset querysetcell(emps,"active","N",7)>
```

```
<cfset querysetcell(emps,"firstname","Tom",8)>
<cfset querysetcell(emps,"department","CF",8)>
<cfset querysetcell(emps,"salary","100000",8)>
<cfset querysetcell(emps,"active","false",8)>
<cfset querysetcell(emps,"firstname","Adam",9)>
<cfset querysetcell(emps,"department","CF",9)>
<cfset querysetcell(emps,"salary","300000",9)>
<cfset querysetcell(emps,"active",false,9)>
<cfset querysetcell(emps,"firstname","Sean",10)>
<cfset querysetcell(emps,"department","CF",10)>
<cfset querysetcell(emps,"salary","250000",10)>
<cfset querysetcell(emps,"active","No",10)>
<cfform name="form01">
    <cfgrid format="html"  insert="yes" insertButton="Add Row"
        name="grid01"
        selectmode="edit"
        width=600
        collapsible="true"
        title="Employees"
        autowidth="yes"
        query="emps"
        sort="yes"
        groupField="active">
            <cfgridcolumn name="FirstName" header="FirstName"/>
            <cfgridcolumn name="Department" header="Department" />
            <cfgridcolumn name="Salary" display=true header="Salary" type="numeric"
values="1000000,1200000" valuesdisplay="1000000,1200000"/>
            <cfgridcolumn name="Active"  display=true header="Contract" type="boolean" />
    </cfgrid>
</cfform>
```

日付列の使用

次の例では、日付列の使用方法を示します。このコードでは、startdate 列の型が date に設定されています。mask には Y/m/d を使用します。Y は 4 桁の年、m は先頭にゼロを付けた月、d は先頭にゼロを付けた日を表します。

```
<!--- using cfgridcolum type="date" --->
<cfset emps = querynew("firstname,department, salary,startdate")>
<cfset queryaddrow(emps,3)>
<cfset querysetcell(emps,"firstname","Debra",1)>
<cfset querysetcell(emps,"department","Accounting",1)>
<cfset querysetcell(emps,"salary","100000",1)>
<cfset querysetcell(emps,"startdate","2009/1/1",1)>
<cfset querysetcell(emps,"firstname","Doherty",2)>
<cfset querysetcell(emps,"department","Finance",2)>
<cfset querysetcell(emps,"salary","120000",2)>
<cfset querysetcell(emps,"startdate","2005/2/21",2)>
<cfset querysetcell(emps,"firstname","Ben",3)>
<cfset querysetcell(emps,"department","Law",3)>
<cfset querysetcell(emps,"salary","200000",3)>
<cfset querysetcell(emps,"startdate","2008/03/03",3)>
<cfform name="form01">
    <cfgrid format="html"  insert="yes" insertButton="Add Row"
        name="grid01"
        selectmode="edit"
        width=600
        collapsible="true"
        title="Employees"
        autowidth="yes"
        query="emps"
        sort="yes"
        groupField="department">
            <cfgridcolumn name="FirstName" header="FirstName"/>
            <cfgridcolumn name="Department" header="Department" />
            <cfgridcolumn name="Salary" display=true header="Salary" type="numeric"
values="1000000,1200000" valuesdisplay="1000000,1200000"/>
            <cfgridcolumn name="StartDate"  display=true header="StartDate" type="date" mask="Y/m/d"/>
    </cfgrid>
</cfform>
```

使用可能なマスクの種類の詳細については、『CFML リファレンス』を参照してください。

ColdFusion 9.0.1 での mask 属性を使用したときの日付処理

HTML グリッドの datefield 列に mask 属性を適用すると、日付は、ColdFusion によって次に示す中間形式に変換されます。

MMMMM, dd yyyy HH:mms

次に例を示します。

January, 19 2005 07:35:42

これは、日付の正しい変換処理に必要で、データがサーバーに送信されたとき (たとえば、onChange グリッドイベントを使用したとき) と、データがサーバーから受信されたとき (たとえば、グリッドの日付フィールドにデータを挿入したとき) の両方のタイミングで適用されます。そのため、場合によっては、データベースの日付列を更新するときに、ユーザーによる日付の形式編集が必要になる場合があります。

注意：NULL の日付値は、フォームの送信時に空の文字列で送信されます。このような場合は、データベースの日付列を更新するときに、明示的に値に NULL を設定します。

HTML のツリーの使用

HTML の cftree タグを使用して、Ajax ベースのツリーデータ表現を作成できます。ツリーデータは、クエリーまたはバインド式を使用して取得できます。クエリーを使用する場合の動作は、アプレットや Flash のツリーの場合と同じです。バインド式を使用する場合は、他のコントロールや Spry データセットの値に基づいてツリーを作成できます。ツリーはダイナミックにロードされ、現在の表示に必要なデータのみが取得されます。

バインド式によるツリーの作成

bind 属性とバインド式を使用すると、ユーザーのツリー操作に合わせて、ツリーデータのロードと表示をダイナミックに行えます。子のツリー項目のデータは親ノードが展開されるまでロードされません。この動作を利用すると、大量のデータをあらかじめツリーに渡す必要がなくなります。また、(項目が展開されるたびに子を取得するように設定することもでき) ツリーの子をダイナミックに変更できるので、アプリケーションの応答性が向上します。

バインディングとバインドパラメータの詳細については、841 ページの「フォームフィールドへのデータのバインディング」を参照してください。

ツリーにおけるバインド式は次のように動作します。

- バインド式を使用する場合、cftree タグで使用できる cftreeitem タグは 1 つのみです。したがって、バインド式から呼び出される関数または URL では、ツリーのすべてのレベルにデータを設定できることが必要です。
- ツリー項目を展開すると、bind 属性で指定されている CFC 関数、JavaScript 関数、またはアクティブページから、その項目の子ノードの値を含む配列が返されます。クライアントのダイナミックツリーコードでは、それらの値を使用して子項目が構築されます。
- ツリーがバインドしているコントロールで、ツリーがリスンしているイベントが発生すると、ツリーが更新されます。たとえば、ツリーのバインド式でバインドパラメータに選択ボックスを指定している場合は、選択ボックスの選択値が変わると、ルートノードまでツリーが折り畳まれます。

バインド式を使用して cftree コントロールにデータを設定する場合は、CFC 関数、JavaScript 関数、または URL を指定し、その関数または URL に、次のバインドパラメータをすべて渡します。渡していないバインドパラメータが 1 つでもあると、エラーが発生します。これらのパラメータを渡すことにより、ツリーやその状態に関する情報がコントロールに自動的に通知されます。

バインドパラメータ	説明
{cftreeitempath}	現在のノード (親ノード) のパスをメソッドに渡します。メソッドはこれに基づいて次のノードを生成します。
{cftreeitemvalue}	現在のツリー項目の値 (通常は value 属性) を渡します。

呼び出された関数または URL が返せるのは単一のレベルに属する項目だけで、ネストされた配列や構造体は返せません。

ルートレベルのツリー項目を生成するために最初に関数または URL が呼び出されるときは、cftreeitemvalue 変数に空の文字列が渡されます。バインド関数では、渡された値が空の文字列かどうかを確認し、空の文字列であればルートレベルのツリーを生成します。

JavaScript の ColdFusion.Tree.refresh 関数を使用してツリーを手動で更新する場合は、@none イベント指定子も役に立ちます。Refresh 関数を呼び出すと、@none が指定されているパラメータを含むすべてのバインドパラメータからデータが取得されます。他のコントロールを指定するすべてのバインドパラメータに @none を指定すると、そのコントロールに変更があってもツリーは自動的に更新されなくなります。ツリーを明示的に更新する ColdFusion.Tree.Referesh 関数を使用すれば、バインドパラメータからデータが取得されます。

バインド式の関数または URL が返すデータの形式は、バインド式の種類によって異なります。

バインドのタイプ	戻り値
CFC	ColdFusion の構造体の配列。構造体は、ColdFusion が呼び出し元に結果を返すときに自動的に JSON 形式に変換されます。または、JSON 形式の構造体を返すこともできます。
JavaScript	JavaScript のオブジェクトの配列。
URL	JSON 形式の構造体の配列。本文にその他の内容を含めることはできません。

配列内の各構造体では、子項目のノードの内容と外観が定義されます。どの構造体にも必ず VALUE フィールドが存在しますが、この他にそれぞれの構造体に応じて次のフィールドが存在します。LEAFNODE を除き、これらの構造体のキーは cftreeitem の属性に対応します。

- DISPLAY
- EXPAND
- HREF
- IMG
- IMGOPEN
- LEAFNODE
- TARGET

注意：CFC が有効なフィールドを返さない場合、エラーが発生しませんが、ツリーは正常に動作しません。

LEAFNODE 構造体要素は、バインド応答構造体でのみ使用されます。これはブール値で、リーフノードであるかどうかを表します。この値が true の場合、ノードの横に +/- アイコンが表示されないので、ユーザーはノードを展開することができません。

バインド式で JavaScript 関数を指定する場合、その関数で使用するフィールド名はすべて大文字にする必要があります。たとえば、VALUE と DISPLAY は使用できますが、value と display は使用できません。ColdFusion では構造体のキー名で大文字のみを使用します。ColdFusion では大文字と小文字が区別されないので、CFC のフィールド名では小文字を使用できますが、JavaScript では大文字と小文字が区別されるので、JavaScript 関数のフィールド名では大文字を適切に使用する必要があります。

URL を使用して CFML ページからツリー項目を取得する場合は、serializeJSON 関数を使用すれば、配列を JSON 形式に変換できます。たとえば、ツリー項目を格納した配列の名前が itemsArray の場合、ページ出力を指定する行は次のようになります。

```
<cfoutput>#serializeJSON(itemsArray)#</cfoutput>
```

例 1 : 単純なツリー

次の例では、レベル数に制限のない、レベルごとに 1 つのノードがある単純な階層ツリーを作成します。display 属性で指定する各ノードのラベルは、ノードのレベルを表します。

CFML ページを次の例に示します。

```
<cfform name="testform">
    <cftree name="t1" format="html">
        <cftreeitem bind="cfc:makeTree.getNodes({cftreeitemvalue},{cftreeitempath})">
    </cftree>
</cfform>
```

ユーザーがノードを展開したときに呼び出される getNodes メソッドを持つ "maketree.cfc" ファイルを次の例に示します。

```
<cfcomponent>
    <cffunction name="getNodes" returnType="array" output="no" access="remote">
        <cfargument name="nodeitemid" required="true">
        <cfargument name="nodeitempath" required="true">
        <!--- The initial value of the top level is the empty string. --->
        <cfif nodeitemid IS "">
            <cfset nodeitemid =0>
        </cfif>
        <!--- Create an array with one element defining the child node. --->
        <cfset nodeArray = ArrayNew(1)>
        <cfset element1 = StructNew()>
        <cfset element1.value = nodeitemid + 1>
        <cfset element1.display = "Node #nodeitemid#">
        <cfset nodeArray[1] = element1>
        <cfreturn nodeArray>
    </cffunction>
</cfcomponent>
```

リーフノードの処理

ツリーのリーフノードに関する情報を返すコードでは、構造体の LEAFNODE フィールドを true に設定してください。これによって、リーフノードのツリーエントリに + のアイコンが表示されないので、そのノードを展開できないことをユーザーが認識できます。LEAFNODE フィールドの使用例を次に示します。

例 2 : リーフノードを処理する複雑なツリー

次のツリーの例では、cfartgallery データベースを使用して、最上位レベルに分野、2 番目のレベルにアーティスト、リーフノードに作品を生成します。ユーザーが作品をクリックすると、その作品のイメージが表示されます。

この例では、ツリーのレベルごとの戻り値や親の値の生成方法を示します。また、戻り構造体の LEAFNODE 要素の使用方法も示します。

このアプリケーションでは、CFC 戻り構造体のキーを小文字にしていますが、自動的に大文字に変換されます。データベースに収録されている分野は絵画、彫刻、写真であるため、これらの分野を表示する最上位レベルのツリーノードにのみ子ノードが表示されます。

アプリケーションのメインページは次のようになります。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!--- The loadimage function displays the image of the selected art.
        It is called when the user clicks the image item. --->
<script>
    function loadImage(img) {
        var imgURL = '<img src="/cfdocs/images/artgallery/'+img+'">';
        var imgDiv = document.getElementById('image');
            imgDiv.innerHTML = imgURL;
    }
</script>

</head>
<body>

<!--- The form uses a table to place the tree and the image. --->
<cfform name="ex1" action="ex1.cfm" method="post">
    <table>
        <tr valign="top">
            <td>
                <cftree name="mytree" format="html">
                    <!--- When you use a bind expression, you must have only one
                            cftreeitem, which populates the tree level. --->
                    <cftreeitem bind="cfc:tree.getItems({cftreeitempath},
                        {cftreeitemvalue})">
                </cftree>
            </td>
            <td>
                <div id="image"></div>
            </td>
        </tr>
    </table>
</cfform>
</body>
</html>
```

"tree.cfc" ファイルを次の例に示します。

```
<cfcomponent output="false">

<cfset variables.dsn = "cfartgallery">

<!--- Function to populate the current level of the tree. --->
<cffunction name="getItems" returnType="array" output="false" access="remote">
    <cfargument name="path" type="string" required="false" default="">
    <cfargument name="value" type="string" required="false" default="">
    <cfset var result = arrayNew(1)>
    <cfset var q = "">
    <cfset var s = "">

    <!--- The cfif statements determine the tree level. --->
    <!--- If there is no value argument, the tree is empty. Get the media types. --->
    <cfif arguments.value is "">
        <cfquery name="q" datasource="#variables.dsn#">
        SELECT mediaid, mediatype
        FROM media
        </cfquery>
        <cfloop query="q">
            <cfset s = structNew()>
            <cfset s.value = mediaid>
            <cfset s.display = mediatype>
            <cfset arrayAppend(result, s)>
        </cfloop>
```

```
    <!--- If the value argument has one list entry, it is a media type. Get the artists for
            the media type.--->
    <cfelseif listLen(arguments.value) is 1>
        <cfquery name="q" datasource="#variables.dsn#">
        SELECT artists.lastname, artists.firstname, artists.artistid
        FROM art, artists
        WHERE art.mediaid = <cfqueryparam cfsqltype="cf_sql_integer"
                value="#arguments.value#">
        AND art.artistid = artists.artistid
        GROUP BY artists.artistid, artists.lastname, artists.firstname
        </cfquery>
        <cfloop query="q">
            <cfset s = structNew()>
            <cfset s.value = arguments.value & "," & artistid>
            <cfset s.display = firstName & " " & lastname>
            <cfset arrayAppend(result, s)>
        </cfloop>

    <!--- We only get here when populating an artist's works. --->
    <cfelse>
        <cfquery name="q" datasource="#variables.dsn#">
        SELECT art.artid, art.artname, art.price, art.description,
                art.largeimage, artists.lastname, artists.firstname
        FROM art, artists
        WHERE art.mediaid = <cfqueryparam cfsqltype="cf_sql_integer"
                value="#listFirst(arguments.value)#">
        AND art.artistid = artists.artistid
        AND artists.artistid = <cfqueryparam cfsqltype="cf_sql_integer"
                value="#listLast(arguments.value)#">
        </cfquery>
        <cfloop query="q">
            <cfset s = structNew()>
            <cfset s.value = arguments.value & "," & artid>
            <cfset s.display = artname & " (" & dollarFormat(price) & ")">
            <cfset s.href = "javaScript:loadImage('#largeimage#');">
            <cfset s.children=arrayNew(1)>
            <!--- leafnode=true prevents node expansion and further calls to the
                bind expression. --->
            <cfset s.leafnode=true>
            <cfset arrayAppend(result, s)>
        </cfloop>

    </cfif>

    <cfreturn result>
</cffunction>

</cfcomponent>
```

ツリーへの他のコントロールのバインディング

バインド式を使用している ColdFusion タグでは、次の形式を使用して、ツリーの選択ノードをバインドできます。

- {[**form**:]**tree**.node} : 選択されているツリーノードの値を取得します。
- {[**form**:]**tree**.path} : 選択されているツリーノードのパスを取得します。completePath 属性の値が true の場合、バインドするパスにはルートノードが含まれます。

ツリーの項目で select イベントが発生するたびに、バインド式が評価されます。バインドパラメータでその他のイベントを指定しても無視されます。

JavaScript のツリー関数

次の JavaScript 関数を使用して、HTML ツリーを管理できます。

関数	説明
ColdFusion.Tree.getTreeObject	Yahoo User Interface Library の TreeView JavaScript オブジェクトを取得します。
ColdFusion.Tree.refresh	ツリーを手動で更新します。

詳細については、『CFML リファレンス』で ColdFusion.Tree.getTreeObject 関数および ColdFusion.Tree.refresh 関数を参照してください。

リッチテキストエディタの使用

ColdFusion のリッチテキストエディタを使用すると、オープンソースの FCKeditor Ajax ウィジェットをベースにしたアイコン形式のインターフェイスを操作しながら、リッチテキスト (HTML) 形式のテキストの入力と形式設定を行えます。このエディタにはさまざまな形式設定用のコントロールと、テキストの検索、印刷、プレビューなどの標準的な操作用のアイコンが装備されています。ただし、ここではテキストエディタのコントロールについては説明しません。リッチテキストエディタのアイコンおよびコントロールの詳細については、http://wiki.fckeditor.net/UsersGuide を参照してください。

注意：ページをロードするときに (テキストボックスのような別のコントロールなどから) リッチテキスト領域にバインドしないように注意してください。

簡単なリッチテキストエディタの例を次に示します。この例では、ユーザーがテキストを入力し [Enter] ボタンをクリックすると、画面が更新されて、エディタ領域の上部に形式設定後のテキストが表示されます。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>

<body>
<!--- Display the text if the form has been submitted with text. --->
<cfif isdefined("form.text01") AND (form.text01 NEQ "")>
    <cfoutput>#form.text01#</cfoutput><br />
</cfif>

<!--- A form with a basic rich text editor and a submit button. --->
<cfform name="form01" >
    <cftextarea richtext=true name="text01" />
    <cfinput type="submit" value="Enter" name="submit01"/>
</cfform>
</body>
</html>
```

注意：ページでリッチテキストエディタを使用する場合、拡張子が .html または .htm のファイルを ColdFusion で処理するように Web サーバーを設定することはできません。これらの拡張子を持つページはデフォルトの HTML プロセッサで処理する必要があります。

リッチテキストエディタの設定

リッチテキストエディタではさまざまなカスタマイズが行えます。cftextarea 属性では、いくつかの基本的なカスタマイズ機能がサポートされています。詳細については、FCKEditor の Web サイト (http://wiki.fckeditor.net/) を参照してください。

カスタムツールバーの定義

ツールバーの外観を制御するには、次の方法があります。

- toolbar 属性でツールバーの名前を指定します。

- "fckconfig.js" ファイルでカスタムツールバーを作成します。

エディタには、アクティブなアイコン、フィールド、区切り線から構成される 1 つのツールバーが表示されます。toolbar 属性では、ツールバー設定を選択できます。属性の値には、"<ColdFusion の Web のルートディレクトリ>/CFIDE/scripts/ajax/FCKEditor/fckconfig.js" ファイルの FCKConfig.ToolbarSets エントリで定義したツールバーセットの名前を指定します。

リッチテキストエディタには 2 つのツールバーセットが用意されています。サポートされているすべての編集コントロールを備えた Default セットと、最小限のコントロールを備えた Basic セットです。デフォルトでは、エディタは Default セットを使用します。テキスト編集コントロールのみで構成される BasicText という名前のカスタムツールバーを作成するには、"fckconfig.js" ファイルで次のエントリを作成し、textarea タグに toolbar="BasicText" と指定します。

```
FCKConfig.ToolbarSets["BasicText"] = [
        ['Source','DocProps','-','NewPage','Preview'],
        ['Cut','Copy','Paste','PasteText','PasteWord','-','Print','SpellCheck'],
        ['Undo','Redo','-','Find','Replace','-','SelectAll','RemoveFormat'],
        ['Bold','Italic','Underline'],
        ['Outdent','Indent'],
        ['JustifyLeft','JustifyCenter','JustifyRight','JustifyFull'],
        '/',
        ['Style','FontFormat','FontName','FontSize'],
        ['TextColor','BGColor'],
        ['FitWindow','-','About']
    ];
```

この設定で定義するツールバーは 2 つの行から構成されており、完全なツールセットに含まれているコントロールのうち基本的なテキスト編集をサポートするコントロールのみを含めています。

ツールバーを定義する場合は、次のルールに従ってください。

- 定義は FCKConfig.ToolbarSets から始めます。
- ツールバー名は二重引用符と角括弧 ([""]) で囲みます。この名前は、cftextarea タグの toolbar 属性で、大文字と小文字の区別を含めて正しく指定する必要があります。
- ツールバー名の後に等号 (=) を続けます。
- 一連の角括弧の中にツールバーコントロールを指定し、定義の最後にセミコロン (;) を付けます。
- 角括弧によってコントロールがグループ化されます。
- 各項目は一重引用符 (') で囲み、カンマ (,) で区切って指定します。
- 区切り線を指定するときは、ハイフン (-) 文字を使用します。
- 改行するときはスラッシュ (/) を使用します。

有効なツールバー項目のすべてのリストについては、"fckconfig.js" の Default 設定を参照してください。

カスタムスタイルの定義

カスタムスタイルを追加すると、ユーザーがスタイルを選択してテキストに適用できるようになります。カスタムスタイルを作成するには、"/CFIDE/scripts/ajax/FCKEditor/fckstyles.xml" に Style 要素を追加します。XML の Style 要素は次のように設定します。

- スタイルとして表示する名前を name 属性に指定します。
- テキストを囲む HTML 要素を element 属性に指定します。
- 子の Attribute 要素で、HTML タグの属性の名前と値を定義します。

次の例では、太字と下線を選択するスタイルを作成します。

```
<Style name="Custom Bold And Underline " element="span">
    <Attribute name="style" value="font-weight: bold; text-decoration: underline;"/>
</Style>
```

"fckstyles.xml" ではなく独自の XML ファイルを使用してスタイルを定義する場合は、stylesXML 属性でファイルパスを指定します。

カスタムテンプレートの定義

エディタには、HTML の書式設定を textarea コントロールに挿入するための基本的なテンプレートが用意されています。たとえば、ImageandTitle テンプレートを使用すれば、領域の左側にイメージのプレースホルダーを設定したり、イメージの右側にタイトルとテキストを設定したりできます。また、イメージ領域を右クリックしてイメージソースなどのプロパティを指定したり、プレースホルダーのタイトルとテキストを置き換えたりすることもできます。

独自のテンプレートを作成するには、"<ColdFusion Web ルートディレクトリ >/CFIDE/scripts/ajax/FCKEditor/fcktemplates.xml" ファイル内でエントリを作成します。それぞれのテンプレート XML のエントリの形式は次のとおりです。

```
    <Template title="template title" image="template image">
        <Description>template description</Description>
        <Html>
            <![CDATA[
                HTML to insert in the text area when the user selects the template.
            ]]>
        </Html>
    </Template>
```

ユーザーがリッチテキストエディタのツールバーにあるテンプレートアイコンをクリックすると、テンプレートダイアログボックスが開き、テンプレートのタイトル、イメージ、および説明が表示されます。

次のテンプレートの例では、タイトルに続けてテキストを定義しています。

```
<Template title="Title and Text" image="template1.gif">
    <Description>A Title followed by text.</Description>
    <Html>
        <![CDATA[
            <h3>Type the title here</h3>
            Type the text here
        ]]>
    </Html>
</Template>
```

[Title and Text] という名前と template1.gif イメージが、テンプレートの選択ダイアログボックスに表示されます。

"fcktemplates.xml" ではなく独自の XML ファイルを使用してテンプレートを定義する場合は、templatesXML 属性でファイルパスを指定します。

カスタムスキンの定義

skin 属性で指定できるカスタムスキンを作成するには、"<ColdFusion Web ルートディレクトリ >/CFIDE/scripts/ajax/FCKeditor/editor/skins" ディレクトリにサブディレクトリを作成します。このサブディレクトリの名前は、skin 属性でスキンを指定するときに使用する名前です。カスタムスキンのディレクトリの下には必ず "images" サブディレクトリを作成し、次のファイルを置きます。

- **fck_editor.css** : ツールバー、ツールバーアイテム (ボタン、パネルなど)、コンテキストメニューなどを含むメインインターフェイスを定義します。
- **fck_dialog.css** : すべてのダイアログボックスの基本的な構造を定義します。
- **fck_strip.gif** : デフォルトのツールバーボタンとコンテキストメニューのアイコンを定義します。このファイルはすべてのアイコンを縦方向に並べたイメージです。各アイコンは 16 × 16 ピクセルのイメージに対応している必要があります。このアイコンの並びにカスタムイメージを追加できます。

- **images/toolbar.buttonarrow.gif** : ツールバーのコンボやパネルボタンに使用する、小さい矢印のイメージを定義します。

スキンで使用されるその他のイメージ (CSS ファイルで指定しているイメージ) は、すべて "images" サブフォルダに置きます。

スキンをカスタマイズする場合、"fck_editor.css" ファイルと "fck_dialog.css" ファイルを変更する方法が最も一般的です。スキンの形式および内容の詳細については、これらのファイルのコメントを参照してください。

日付フィールドの入力コントロールの使用

タイプの値が datefield である HTML の cfinput コントロールでは、ポップアップカレンダーから日付を選択したり、入力ボックスに直接日付を入力したりできます。このコントロールを使用する場合は、次の点に注意してください。

- Internet Explorer と Firefox で、コントロールの横にラベルテキストが正しく表示されるようにするには、ラベルテキストを タグで囲み、各行の間に
 を 3 つ指定します。
- cflayoutarea タグの overflow 属性の値を visible にすれば、ポップアップカレンダーがレイアウト領域の枠からはみ出ても、完全に表示されます。
- mask 属性を使用して日付形式を指定しても、ユーザーはその形式以外の日付も入力することができます。mask 属性は、ユーザーがポップアップカレンダーで日付を選択したときに使用する形式を指定するものです。また、ユーザーがフィールドに日付を入力してからポップアップカレンダーを開いた場合、カレンダーにその日付が表示されるのは、入力した日付がマスクパターンに従っている場合のみです。mask 属性を指定していない場合、ポップアップカレンダーではデフォルトの一致パターンが使用されます。
- ユーザーがポップアップカレンダーを使用せず、月の名前または月の略称を使用してコントロールに日付を入力した場合、ポップアップカレンダーにその日付が表示されるのは、次の条件が両方とも満たされている場合のみです。
  - 月の入力位置と名前の形式が、マスクパターンと一致している。
  - 月の名前が、大文字と小文字の区別も含めて、monthNames 属性で指定した月の名前 (mmm マスクの場合は 3 文字の略称) に一致している。
- 年の日付マスクとして yy を指定した場合、ポップアップカレンダーでは 1951 年～ 2050 年の範囲が使用されます。たとえば、テキストフィールドに「3/3/49」と入力すると、カレンダーには 2049 年 3 月 3 日が表示されます。
- 無効な数字を日付として入力した場合は、その数字をもとに正しい日付が推測されてポップアップカレンダーに表示されます。たとえば、dd/mm/yyyy マスクを指定している場合に「32/13/2007」と入力すると、ポップアップカレンダーには 2008 年 2 月 1 日が表示されます。

次の例では、簡単なタブのレイアウトを示します。それぞれのタブには、複数の日付フィールドコントロールから構成されるフォームが表示されます。

```
<html>
<head>
</head>

<body>
<cflayout type="tab" tabheight="250px" style="width:400px;">
    <cflayoutarea title="test" overflow="visible">
        <br>
        <cfform name="mycfform1">
            <div style="float:left;">Date 1: </div>
            <cfinput type="datefield" name="mydate1"><br><br><br>
            <div style="float:left;">Date 2: </div>
            <cfinput type="datefield" name="mydate2" value="15/1/2007"><br><br><br>
            <div style="float:left;">Date 3: </div>
            <cfinput type="datefield" name="mydate3" required="yes"><br><br><br>
            <div style="float:left;">Date 4: </div>
            <cfinput type="datefield" name="mydate4" required="no"><br><br><br>
        </cfform>
    </cflayoutarea>
    <cflayoutarea title="Mask" overflow="visible">
        <cfform name="mycfform2">
            <br>
            <div style="float:left;">Date 1: </div>
            <cfinput type="datefield" name="mydate5" mask="dd/mm/yyyy">
                (dd/mm/yyyy)<br><br><br>
            <div style="float:left;">Date 2: </div>
            <cfinput type="datefield" name="mydate6" mask="mm/dd/yyyy">
                (mm/dd/yyyy)<br><br><br>
            <div style="float:left;">Date 3: </div>
            <cfinput type="datefield" name="mydate7" mask="d/m/yy">
                (d/m/yy)<br><br><br>
            <div style="float:left;">Date 4: </div>
            <cfinput type="datefield" name="mydate8" mask="m/d/yy">
                (m/d/yy)<br><br><br>
        </cfform>
    </cflayoutarea>
</cflayout>

</body>
</html>
```

注意：Internet Explorer のバージョンが IE 7 よりも前の場合は、ページの最初の 3 つのフィールドに対応するカレンダーが、後続の入力コントロールの背後に表示される可能性があります。

選択候補を表示するテキスト入力フィールドの使用

HTML でテキスト入力 (type="text") を作成するときは、ユーザーの入力内容に応じてフィールドの選択候補を表示するためのスタティックソースまたはダイナミックなソースを autosuggest 属性で指定できます。ユーザーが一定の文字数を入力してから選択候補を表示するには、autosuggestMinLength 属性に文字数を指定します。

注意：autosuggest 属性を使用する cfinput コントロールの横にラベルテキストを配置し、Internet Explorer と Firefox で正しく表示されるようにするには、style="float: left" 属性を指定した HTML の div タグでラベルテキストを囲みます。また、コントロールが複数あり、各コントロールを別の行に表示する場合は、次の例に示すように、入力コントロールの後に
 タグを 3 つ続けます。これを行わないと、ラベルとコントロールが正しく配置されません。

```
<div style="float: left"> Name: </div>
<cfinput name="userName" type="text" autosuggest="Andrew, Jane, Robert"> <br><br><br>
```

選択候補のエントリは、スタティックな値のリストとして提供します。スタティックな選択候補リストを使用するには、autosuggest 属性にリストのエントリを指定します。各エントリは、delimiter 属性で指定した文字 (デフォルトはカンマ) で区切ります。その例を次に示します。

```
<cfinput type="text"
    autosuggest="Alabama\Alaska\Arkansas\Arizona\Maryland\Minnesota\Missouri"
    name="city" delimiter="\">
```

この例では、cfinput コントロールに「**a**」または「A」と入力すると、A で始まる州のリストがドロップダウンリストに表示されます。矢印キーを押して選択対象まで移動し Enter を押すと、そのアイテムが選択されます。

また、選択候補のリストをダイナミックに生成できます。ダイナミックなリストを使用するには、CFC 関数、JavaScript 関数、または URL を autosuggest 属性に指定します。関数を呼び出す最小間隔を指定するには、autosuggestBindDelay 属性を使用します。この時間間隔を指定することで、サーバーに送信されるリクエストの数が制限されます。ダイナミックなリストでは、選択候補を取得中であることを示すアイコンが、入力フィールドの右にアニメーション表示されます。

バインド式を使用するときは、{cfautosuggestvalue} バインドパラメータを関数呼び出しまたは URL のパラメータに指定する必要があります。このパラメータを介して、入力コントロールにユーザーが入力した値が、関数またはページに渡されます。

選択候補を生成する CFC または JavaScript 関数は、1 次元配列またはカンマ区切りリストの形式で選択候補値を返します。

URL から返す HTTP レスポンスの本文は、選択候補値を列挙した JSON 形式の配列またはリストのみで構成されている必要があります。ColdFusion の serializeJSON 関数を使用すれば、配列を JSON 形式に変換できます。たとえば、選択候補を格納した配列の名前が nodeArray である場合、バインド式の URL で呼び出された CFML ページでは、次の行のみを出力します。

```
<cfoutput>#serializeJSON(nodeArray)#</cfoutput>
```

返すデータは、cfautosuggestvalue の内容に一致していなくてもかまいません。ユーザー入力に一致していない値は、クライアントサイドコードによって除外されます。実際、呼び出された関数やページでは、cfautosuggestvalue パラメータの値を使用しなくてもかまいません。ただし、このパラメータを使用しないと大量のデータが返される場合は、使用してください。

次の例では、バインド式を使用して、選択候補リストを表示します。[姓] テキストボックスに姓を入力すると、データベース内の該当する姓がすべて選択候補リストに表示されます。[名] テキストボックスではバインド式を使用して [姓] テキストボックスにバインドし、姓と [名] ボックスに入力されたテキストに該当する名のみを [名] テキストボックスに表示します。データベースクエリーを使用して、選択候補の条件に一致する結果のみを返しているので、選択候補リストには返された結果がすべて表示されます。また、大文字と小文字を含めてデータベースエントリと一致した場合にのみ、選択候補と一致したと見なされます。

cfdocexamples データベースを使用してこの例をテストするには、最初のボックスに「S」と入力し、選択候補リストに [Smith] と [Stewart] が表示されることを確認します。[Smith] を選択してから、[名] ボックスに「A」または「J」と入力すると、名前の選択候補が表示されます。

このアプリケーションは次のようになります。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
<body>

<cfform>
    Last Name:<br />
    <cfinput type="text" name="lastName"
        autosuggest="cfc:suggestcfc.getLNames({cfautosuggestvalue})"><br />
    <br />
    First Name:<br />
    <cfinput type="text" name="firstName"
        autosuggest="cfc:suggestcfc.getFNames({cfautosuggestvalue},{lastName})">
</cfform>
</body>
</html>
```

"suggestcfc.cfc" ファイルを次の例に示します。

```
<cfcomponent>

    <cffunction name="getLNames" access="remote" returntype="array" output="false">
        <cfargument name="suggestvalue" required="true">
        <!--- The function must return suggestions as an array. --->
        <cfset var myarray = ArrayNew(1)>
        <!--- Get all unique last names that match the typed characters. --->
        <cfquery name="getDBNames" datasource="cfdocexamples">
        SELECT DISTINCT LASTNAME FROM Employees
        WHERE LASTNAME LIKE <cfqueryparam value="#suggestvalue#%"
            cfsqltype="cf_sql_varchar">
        </cfquery>
        <!--- Convert the query to an array. --->
        <cfloop query="getDBNames">
            <cfset arrayAppend(myarray, lastname)>
        </cfloop>
        <cfreturn myarray>
    </cffunction>

    <cffunction name="getFNames" access="remote" returntype="array"
            output="false">
        <cfargument name="suggestvalue" required="true">
        <cfargument name="lastName" required="true">
        <cfset var myarray = ArrayNew(1)>
        <cfquery name="getFirstNames" datasource="cfdocexamples">
        <!--- Get the first names that match the last name and the typed characters. --->
        SELECT FIRSTNAME FROM Employees
        WHERE LASTNAME = <cfqueryparam value="#lastName#"
            cfsqltype="cf_sql_varchar">
        AND FIRSTNAME LIKE <cfqueryparam value="#suggestvalue & '%'#"
            cfsqltype="cf_sql_varchar">
        </cfquery>
        <cfloop query="getFirstNames">
            <cfset arrayAppend(myarray, Firstname)>
        </cfloop>
        <cfreturn myarray>
    </cffunction>

    </cfcomponent>
```

数値データに関する問題

先頭にゼロを付けて数値データを返す CFC メソッドの場合（例えば、郵便番号 02674）、returnformat="string" と設定しても、ゼロはバインド式によって 8 進数およびそれに相当する 10 進数（この場合は 1468）と解釈されます。この問題を解決するには、URL バインドまたは JavaScript 関数（例えば cfajaxproxy を使用）によってルーティングされるバインドの場合は、returnformat=plain を設定することで数値を維持できます。また、選択候補を表示するコントロールの選択候補リストからは、先頭のゼロが除去されます。

cfslider タグの使用

cfslider タグを使用すると、一定の範囲から数値を選択するためのスライダコントロールが ColdFusion フォーム内に配置されます。このスライダは、HTML 形式およびアプレット形式のフォームを表す cfform タグ内で使用できます。cfslider は、Flash フォームではサポートされません。

HTML フォームの場合は、最大値、最小値、およびインクリメント値を指定して、複雑な結果をすばやくフィルタ処理できます。使用可能なスライダコントロールには、上下にコントロールを調整できる Vertical、左右にコントロールを調整できる Horizontal、データヒントとして値が表示される Tip、指定したインクリメント値に基づき一定の間隔で移動する Snapping の 4 種類があります。

実行時のスライダの操作

次のコードは、実行時にスライダを操作する方法を示しています。

```
 <script language="javascript">
    //use Coldfusion AJAX functions
    var sliderChange = function(slider,value)
    {
        //get slider name
        slidername = slider.getId();
        //get slider value
        currValue = ColdFusion.Slider.getValue(slidername);
        //set a new slider value
        newValue = parseInt(currValue+10);
        ColdFusion.Slider.setValue(slidername,newValue);
        //hide slider
        if(confirm("Do you want to hide slider?"))
        {
            ColdFusion.Slider.hide(slidername);
        }
        //show slider
        if(confirm("Do you want to show slider?"))
        {
                ColdFusion.Slider.show(slidername);
        }
        //disable slider
        if(confirm("Do you disable the slider?"))
        {
            ColdFusion.Slider.disable(slidername);
        }
        //enable slider
        if(confirm("Do you enable the slider?"))
        {
            ColdFusion.Slider.enable(slidername);
        }
    }
    var sliderDrag = function(slider)
    {
        //get slider name
        slidername = slider.getId();
        document.getElementById('currentSliderValue').innerHTML =  "Current Slider value : <font
color='red'><strong>" + ColdFusion.Slider.getValue(slidername) + "<strong></font>";
    }
</script>
<br>
<cfform name="frm1">
```

```
    <p>
    <span id="currentSliderValue">Current Slider Value: <font
color="red"><strong>50</strong></font></span><br>
    </p>
</cfform>
<p>
<br><b>Volume</b>:
<cfslider
    name="s"
    format="html"
    min=1
    max=100
    value="50"
    tip="yes"
    onChange="sliderChange"
    onDrag = "sliderDrag"
    vertical="no"
    width="200pt"
    >
</p>
```

地図の使用

cfmap タグを使用すると、ColdFusion ページ内に地図を埋め込むことができます。サポートされる地図のタイプは次のとおりです。

- earth
- terrain
- satellite
- hybrid
- map (デフォルト)

マーカーウィンドウの使用

地図内のマーカーアイコンをクリックすると、マーカーウィンドウが開きます。このウィンドウは、地図内の場所に関する情報 (住所、緯度、経度など) を表示する目的に使用できます。マーカーウィンドウには、スタティックコンテンツを表示することも、ダイナミックコンテンツを挿入することもできます。

スタティックコンテンツを使用したデータの設定

マーカーウィンドウのデータを手動で設定するには、markerwindowcontent 属性の値を指定します。

バインド式を使用したダイナミックデータの挿入

データをダイナミックに挿入するには、markerbind 属性で、CFC、JavaScript 関数、または URL を呼び出すバインド式を指定します。バインド式では、バインドパラメータを使用して、ダイナミックな情報や、他のフォームフィールドの属性値を指定します。

バインド式にはバインドパラメータを渡す必要があります。関数呼び出しまたは URL のパラメータを 1 つでも省略すると、エラーが発生します。これらのパラメータを渡すことにより、地図やその状態に関する情報がデータ生成関数に通知されます。パラメータの実際のデータは自動的に供給されます。手動で値を設定することはできません。

次のコードに示すように、データを渡します。

```
<br>
<cfoutput>
<table>
    <tr>
        <td bgcolor='red'><h4><font color='white'>URL Bind Example</font></td>
    </tr>
</table>
Map Name: #cfmapname#<br>
Latitude, Longitude: (#DecimalFormat(cfmaplatitude)#,#DecimalFormat(cfmaplongitude)#)<br>
Address: #cfmapaddress#<br>
</cfoutput>
```

次の表にパラメータの詳細を示します。

パラメータ名	説明
cfmapname	マップの名前です。
cfmaplatitude	場所の緯度の値です (単位 : 度数)。この値がマップの中央として設定されます。
cfmaplongitude	場所の経度の値です (単位 : 度数)。この値がマップの中央として設定されます。
cfmapaddress	マップの中央として設定される、場所の住所です。

返すデータの形式は、データの取得方法によって異なります。

バインドのタイプ	戻り値
CFC	ColdFusion の構造体。呼び出し元に合わせて自動的に変換されます。または、JSON 形式の構造体を返すこともできます。
URL	JSON 形式の構造体。本文にその他の内容を含めることはできません。
JavaScript	JavaScript オブジェクト。

ウィンドウの表示を制御するには、showmarkerwindow 属性を使用します。

次の例は、バインド式と CFC を使用し、CFC バインド式を使用してダイナミックデータを渡す方法を示しています。CFML ページの内容は次のとおりです。

```
<br>
<cfmap
        centeraddress="Hobart, Tasmania, Australia"
        name="map1"
        type="map"
        tip="Hobart, Tasmania, Australia"
        zoomControl="small3d"
        markerbind="cfc:maps.getMapData({cfmapname}, {cfmaplatitude}, {cfmaplongitude}, {cfmapaddress})"
        showmarkerwindow = true>
    <cfmapitem name="m1" address="Taj Mahal, Agra, India" tip="Taj Mahal, Agra, India">
    <cfmapitem name="m2" latitude="40.46" longitude="117.05" showmarkerwindow=true tip="Great Wall of
China, Bejing">
    <cfmapitem name="m3" address="Stonehenge, England" tip="Stonehenge, England" showmarkerwindow = false>
</cfmap>
```

"map.cfc" は次のようになります。

```
<cfcomponent>
    <cffunction name="getMapData" access="remote">
        <cfargument name="cfmapname">
        <cfargument name="cfmaplatitude">
        <cfargument name="cfmaplongitude">
        <cfargument name="cfmapaddress">
        <cfsavecontent variable="markup">
            <br>
            <cfoutput>
            <table>
                <tr>
                    <td bgcolor='red'><h4><font color='white'>CFC Bind Example</font></td>
                </tr>
            </table>
            Map Name: #cfmapname#<br>
            Latitude, Longitude: (#DecimalFormat(cfmaplatitude)#,#DecimalFormat(cfmaplongitude)#)<br>
            Address: #cfmapaddress#<br>
            </cfoutput>
        </cfsavecontent>
        <cfreturn markup>
    </cffunction>
</cfcomponent>
```

次の例は、JavaScript バインド式を使用してダイナミックデータを渡す方法を示しています。

```
<script language="JavaScript">
    var getMapData = function(cfmapname, cfmaplatitude, cfmaplongitude, cfmapaddress){
        var msg = "";
        msg = msg + "Map Name: " + cfmapname + "<br>";
        msg = msg + "Latitude,longitude: " + "("  + cfmaplatitude + "," + cfmaplongitude + ")" + "<br>";
        msg = msg + "Address: " + cfmapaddress + "<br>";
        //alert(msg);
        return "<br><table><tr><td bgcolor='red'><h4><font color='white'>" + "Javascript Bind Example" +
"</font></td></tr></table><hr>" + msg;
    }
</script>
<cfmap
    centeraddress="Hobart, Tasmania, Australia"
    name="map1"
    type="map"
    tip="Hobart, Tasmania, Australia"
    zoomControl="small3d"
    markerbind="javascript:getMapData({cfmapname}, {cfmaplatitude}, {cfmaplongitude}, {cfmapaddress})"
    showmarkerwindow = true>
    <cfmapitem name="m1" address="Taj Mahal, Agra, India" tip="Taj Mahal, Agra, India">
    <cfmapitem name="m2" latitude="40.46" longitude="117.05" showmarkerwindow=true tip="Great Wall of
China, Bejing">
    <cfmapitem name="m3" address="Stonehenge, England" tip="Stonehenge, England" showmarkerwindow = false>
</cfmap>
```

次の例は、URL バインド式を使用してダイナミックデータを渡す方法を示しています。

```
<cfmap
    centeraddress="Hobart, Tasmania, Australia"
    name="map1"
    type="map"
    tip="Hobart, Tasmania, Australia"
    zoomControl="small3d"
markerbind="url:mapdata.cfm?cfmapname={cfmapname}&cfmaplatitude={cfmaplatitude}&cfmaplongitude={cfmaplong
itude}&cfmapaddress={cfmapaddress}"
    showmarkerwindow = true>
    <cfmapitem name="m1" address="Taj Mahal, Agra, India" tip="Taj Mahal, Agra, India">
    <cfmapitem name="m2" latitude="40.46" longitude="117.05" showmarkerwindow=true tip="Great Wall of
China, Bejing">
    <cfmapitem name="m3" address="Stonehenge, England" tip="Stonehenge, England" showmarkerwindow = false>
</cfmap>
```

Google マップキーの指定

Web ページに Google マップを埋め込むには、Google Maps API キーが必要になります。

Google Maps API キーを取得するために登録する方法の詳細については、次の URL を参照してください。

http://code.google.com/intl/ja/apis/maps/signup.html

現在 ColdFusion でサポートされているのは、Google マップの埋め込みのみです。マップを生成するには、有効な Google Maps API キーを入力し、場所の緯度と経度、または住所を指定します。Google MAP API キーは、次の方法で指定できます。

- cfajaximport タグを使用します。次のように、Map API キーを params 属性で指定します。

  <cfajaximport params="#{googlemapkey='Map API Key'}#"

- 次のように Application.cfc を使用します。

  <cfset this.googlemapkey="Map API Key">

- ColdFusion Administrator の [設定] ページを使用します。[Google Map API キー] フィールドに MAP API キーを指定します。また、runtime.cfc 内に MAP API キーを指定することもできます。

マーカーのスタイル指定

次の項目を指定できます。

- カスタムマーカーアイコン : markericon 属性を使用して、アイコンへのパスを指定します。適切なサイズのイメージを指定してください。
- マーカーアイコンの色 : markercolor 属性を使用します。デフォルトのアイコンに対しては任意の色を指定できますが、それ以外のアイコンに対しては指定できません。
- マップのタイトル : title 属性を使用します。

cfprogressbar タグの使用

cfprogressbar タグには次の特徴があります。

- 指定した期間の間、進行状況表示バーを自動的に実行します。
- バインド式を使用して、データをダイナミックにロードします。
- 進行状況表示バーのスタイルを指定できます。
- 処理の完了後、または例外の発生時に、ユーザーが進行状況表示バーを制御できるように、コールバックハンドラやエラーハンドラを使用します。

- JavaScript API を使用して、進行状況表示バーをプログラムで制御できます。

進行状況表示バーのモード

進行状況表示バーは、次の 2 つのモードをサポートしています。

ダイナミックモード

バインド式を使用して、進行状況表示バーに表示するデータを取得します。bind 属性で、インジケータの長さを決定する関数を指定します。

次の CFM コードは、CFC バインド式の使用方法を示しています。

```
<cfajaxproxy cfc="pbar" jsclassname="pbar">
<head>
    <script>
        var utils = new pbar();
        var count = 0;
        var init = function()
        {
            document.getElementById('cfpbLabel').style.display = 'block';
            ColdFusion.ProgressBar.show('pBar');
            ColdFusion.ProgressBar.start('pBar');
        }
        var hideProgessBar = function()
        {
            document.getElementById('cfpbLabel').style.display = 'none';
            ColdFusion.ProgressBar.hide('pBar');
            utils.resetStatus();
        }
    </script>
</head>
<cfform>
    <div id="cfpbLabel" style="display:none">
        Saving File:
    </div>
    <cfprogressbar
        name="pBar"
        autodisplay=false
        bind="cfc:pbar.getProgessData()"
        onComplete="hideProgessBar"
        width="400">
    <cfset ajaxOnLoad('init')>
</cfform>
```

次の "pb.cfc" には、進行状況表示バーのデータを返す関数が含まれています。

```
<cfcomponent>
    <cffunction name="resetStatus" access="remote">
        <!---
        Clear count from session so that next time the progress bar runs from the start time.
        --->
        <cfif session.count gte 10>
            <cfset structdelete(session,"count")>
        </cfif>
    </cffunction>
    <cffunction name="getProgessData" access="remote">
        <!--- use a count to track progress --->
        <cfif not isdefined('session.count')>
            <cfset session.count = 1>
        <cfelse>
            <cfset session.count = session.count + 1 >
        </cfif>
        <!--- struct with status and message components of the progressbar --->
        <cfset data = {status=session.count * 0.1,message=(session.count * 10)  & "%"}>
        <cfreturn data>
    </cffunction>
</cfcomponent>
```

次の CFM コードは、URL バインド式の使用方法を示しています。

```
<head>
    <script>
        var init = function()
        {
            document.getElementById('cfpbLabel').style.display = 'block';
            ColdFusion.ProgressBar.show('pBar');
            ColdFusion.ProgressBar.start('pBar');
        }
        var hideProgessBar = function()
        {
            document.getElementById('cfpbLabel').style.display = 'none';
            ColdFusion.ProgressBar.hide('pBar');
        }
    </script>
</head>
<cfform>
    <div id="cfpbLabel" style="display:none">
        Saving File:
    </div>
    <cfprogressbar
        name="pBar"
        autodisplay=false
        bind="url:progressdata.cfm"
        onComplete="hideProgessBar"
        width="400">
    <cfset ajaxOnLoad('init')>
</cfform>
```

"Progressdata.cfm" の内容は次のとおりです。

```
<!--- use a count to indicate progress --->
<cfif not isdefined('session.count')>
    <cfset session.count = 1>
<cfelse>
    <cfset session.count = session.count + 1 >
</cfif>
<!--- the struct to be sent back; using the populate the status and message components of the progressbar --->
<cfset data = {status=session.count * 0.1,message=(session.count * 10)  & "%"}>
<!--- clear count from session to start afresh the next time the program is run --->
<cfif session.count eq 10>
    <cfset structdelete(session,"count")>
</cfif>
<!--- data sent back via URL binds must use SerializeJSON() --->
<cfoutput>#SerializeJSON(data)#</cfoutput>
```

次の CFM コードは、JavaScript バインド式の使用方法を示しています。

```
<head>
    <script>
        var count = 0;
        var init = function()
        {
            document.getElementById('cfpbLabel').style.display = 'block';
            ColdFusion.ProgressBar.show('pBar');
            ColdFusion.ProgressBar.start('pBar');
        }
        var hideProgessBar = function()
        {
            document.getElementById('cfpbLabel').style.display = 'none';
            ColdFusion.ProgressBar.hide('pBar');
        }
        var getProgessData = function()
        {
            count++;
            if(count > 10)
                return {STATUS:1,MESSAGE:"Done"}
            else
                return {STATUS:count*0.1,MESSAGE:(count * 10) + "%"}
        }
    </script>
</head>
<cfform>
    <div id="cfpbLabel" style="display:none">
        Saving File:
    </div>
    <cfprogressbar
        name="pBar"
        autodisplay=false
        bind="javascript:getProgessData()"
        onComplete="hideProgessBar"
        width="400"
        >
    <cfset ajaxOnLoad('init')>
</cfform>
```

手動モード

手動モードでは、進行状況の表示が完了するまでの時間を指定します。

次の例では、autodisplay が false に設定されているため、ページが最初にロードされた時点では、進行状況表示バーは表示されません。ページがロードされると、init 関数が呼び出されます。この関数により、進行状況表示バーが表示されて実行されます。このモードで使用されるデフォルトの間隔は 1 秒です。

```
<head>
    <script>
        var init = function()
        {
            document.getElementById('cfpbLabel').style.display = 'block';
            ColdFusion.ProgressBar.show('pBar');
            ColdFusion.ProgressBar.start('pBar');
        }
        var hideProgessBar = function(){
            document.getElementById('cfpbLabel').style.display = 'none';
            ColdFusion.ProgressBar.hide('pBar');
        }
    </script>
</head>
<cfform>
    <div id="cfpbLabel" style="display:none">
        Saving File:
    </div>
    <cfprogressbar
        name="pBar"
        duration="10000"
        autodisplay=false
        onComplete="hideProgessBar"
        width="400"
    />
    <cfset ajaxOnLoad('init')>
</cfform>
```

実行時の進行状況表示バーの操作

ここでは、JavaScript API を使用して進行状況表示バーのステータスを更新する方法について説明します。次の CFM コードを実行すると、JavaScript API の ColdFusion.ProgressBar.updatestatus を使用する進行状況表示バーがロードされます。

プログラムがロードされると、init 関数によって進行状況表示バーが表示され、進行状況表示バーを手動で更新できるように JavaScript の getProgressData 関数が呼び出されます。getProgressData 関数は、JavaScript API の update status に渡される status 変数と message 変数に値を代入します。

進行状況表示バーを実行時に操作するときは、duration=***5000*** のように、仮の期間を指定してください。カスタムの JavaScript 関数によって実際の期間が決定される場合でも、duration は必須の属性です。

```
<cfajaxproxy cfc="pbar" jsclassname="pbar">
<head>
    <script>
        var utils = new pbar();
        var init = function()
        {
            document.getElementById('cfpbLabel').style.display = 'block';
            ColdFusion.ProgressBar.show('pBar');
            getProgessData();
        }
        var hideProgessBar = function()
        {
            document.getElementById('cfpbLabel').style.display = 'none';
            ColdFusion.ProgressBar.hide('pBar');
        }
        var getProgessData = function()
        {
            for(i=1;i <= 10;i++)
            {
                var status = parseFloat(i * 0.10);
```

```
                var message = Math.round(status * 100) + "%";
                ColdFusion.ProgressBar.updateStatus('pBar',status,message);
                utils.sleep(1000);
            }
            hideProgessBar();
        }
    </script>
</head>
<cfform>
    <div id="cfpbLabel" style="display:none">
        Saving File:
    </div>
    <cfprogressbar
        name="pBar"
        autodisplay=false
        duration=15000
        onComplete="hideProgessBar"
        width="400">
    <cfset ajaxOnLoad('init')>
</cfform>
```

JavaScript コード内のスリープ機能は、次の CFC に含まれる sleep 関数によって提供されます。

```
<cfcomponent>
<cffunction name="sleep" access="remote">
    <cfargument name="timetosleep" default="1000">
    <cfset sleep(timetosleep)>
</cffunction>
</cfcomponent>
```

進行状況表示バーのスタイル指定

cfprogressbar の style 属性を使用すると、次の要素を設定できます。

- 進行状況表示バーの背景色
- 進行状況メッセージの色
- 進行状況インジケータの色

次のコードでは、style="bgcolor:ADD8E6;progresscolor:6183A6;textcolor:191970" というスタイル指定を行っています。

cfmessagebox タグの使用

cfmessagebox タグを使用すると、ポップアップメッセージを表示するコントロールを定義できます。また、このタグでは、標準的なアラートボックスだけでなく、ボックス内に表示するプロンプトや入力フィールドも定義できます。

確認ダイアログボックスの使用

次のコードは、[Yes] ボタンと [No] ボタンを含んだ確認メッセージボックスを作成する方法を示しています。

```
<cfmessagebox
    type="confirm"
    name="msgbox1"
    title="Confirm Dialog"
    message="Do you want proceed?"
    buttonType="YesNo"
    icon="info"
    labelYes="Click Yes to continue"
    labelNo="No"
    x=100
    y=200>
<!--- This example illustrates usage of the Confirm dialog in "YesNoCancel" mode --->
<cfmessagebox
    type="confirm"
    name="msgbox2"
    title="Save File"
    message="Do you want to save the file?"
    buttonType="YesNoCancel"
    icon="question"
    labelYes="Click Yes to save the file"
    labelNo="No"
    labelCancel="Quit"
    width="400"
    x=500
    y=200>
<br><br>
<input
    type="button"
    name="confirm1"
    onClick="javascript:ColdFusion.MessageBox.show('msgbox1');"
    value="YesNo Confirm"
    >
<input
    type="button"
    name="confirm2"
    onClick="javascript:ColdFusion.MessageBox.show('msgbox2');"
    value="YesNoCancel Confirm"
    >
```

メッセージボックスのスタイル指定

bodyStyle 属性は、メッセージボックス本文の CSS スタイル指定であり、メッセージのスタイルを指定できます。原則として、この属性は色とフォントスタイルを設定する目的に使用します。

bodyStyle 属性の使用例を次に示します。

```
<cfmessagebox
    type="alert"
    name="msgbox1"
    title="Download Status"
    message="File Download Complete"
    icon="info"
    width="400"
    bodyStyle="background-color:white;color:blue"
    x=300
    y=200>
<br><br>
<input
    type="button"
    name="alert"
    onClick="javascript:ColdFusion.MessageBox.show('msgbox1');"
    value="Alert MessageBox"
    >
```

Ajax データ機能および開発機能の使用

Adobe ColdFusion では、Web ページでダイナミックにデータを処理する Ajax 機能がサポートされています。

ColdFusion の Ajax ユーザーインターフェイス機能については、788 ページの「Ajax ユーザーインターフェイスコンポーネントおよび機能の使用」を参照してください。

ColdFusion の Ajax データ機能および開発機能について

Ajax (Asynchronous JavaScript and XML) とは、インタラクティブな Web アプリケーションを作成するための一連の Web テクノロジーのことです。Ajax アプリケーションは通常、次のテクノロジーから構成されます。

- 情報の形式設定および表示を行う HTML と CSS
- クライアントサイドでダイナミックな処理を実行する JavaScript
- XMLHttpRequest 関数によるサーバーとの非同期通信
- サーバーとクライアントの間でデータのシリアル化や転送を行うための XML または JSON (JavaScript Object Notation)

ColdFusion には、Ajax テクノロジーを使用したダイナミックアプリケーションの作成を支援するツールが多数用意されています。ColdFusion のタグや関数を使用すれば、複雑な Ajax アプリケーションを簡単に作成できます。

ColdFusion の Ajax 機能

ColdFusion には、データ管理および開発に使用する Ajax 機能と、ユーザーインターフェイスに使用する Ajax 機能が用意されています。

データ機能と開発機能

ColdFusion のデータ機能と開発機能を使用すれば、ColdFusion でデータをダイナミックに処理する効果的な Ajax アプリケーションを開発できます。データ機能と開発機能には、Spry などの他の Ajax フレームワークで使用できる機能が多数用意されています。

- ColdFusion の多くのタグでは、データバインディングがサポートされています。フォームタグや表示タグ (cfselect や cfwindow など) を使用するアプリケーションでバインディングを使用すれば、フォームの入力に基づいてダイナミックに情報を表示できます。フォームのデータを別のフォームフィールドにそのまま表示するだけでなく、フォームフィールドのデータをパラメータとして CFC 関数、JavaScript 関数、または URL に渡し、その処理結果を使用して表示を制御

することもできます。データバインディングでは、イベントを使用して表示を自動的に更新します。通常、バインドされた入力データの変更時に更新が実行されます。また、JavaScript の ColdFusion.Ajax.submitForm 関数を使用して、バインド可能な要素の現在の値を取得することもできます。

- cfajaxproxy タグは、サーバーの CFC に対応する JavaScript プロキシを作成します。このタグは、クライアントとサーバー間の通信を管理するとともに、通信とその結果の処理を簡素化し管理するための関数をいくつか提供しています。このタグを使用すると、CFC のすべてのリモート関数にアクセスできます。また、このタグを使用することにより、アプリケーション (Dojo や Backbase といった Ajax フレームワークやウィジェットセットを使用するアプリケーションなど) は ColdFusion サーバーからデータに容易にアクセスできます。
- cfsprydataset タグを使用すると、バインド式を使用して、Adobe Spry データセットをダイナミックに作成し更新することができます。ダイナミック領域などの Spry フレームワーク要素を使用するアプリケーションは、このタグを使用することで、ColdFusion コントロールへの入力に基づいて Spry 要素に情報を設定します。この機能により、Spry コントロールと ColdFusion コントロールを容易に併用することができます。
- cfajaximport タグを使用すると、ColdFusion ページにインポートする JavaScript ファイルや CSS ファイルの場所を指定できます。また、Ajax ベースの特定のタグや関数に必要なファイルを選択的にインポートすることもできます。ファイルの場所を自由に変更できるので、さまざまな設定をサポートでき、またアプリケーションごとのスタイル定義などの高度な設定も可能です。ColdFusion は必要なファイルを自動的に判別しインポートしますが、場合によっては手動で情報を指定する必要があります。
- ColdFusion に用意されている CFML 関数を使用すれば、サーバーで JSON 形式のデータを作成して利用したり、HTML の cfgrid タグで使用するデータを作成したりできます。
- フローティングロギングウィンドウを表示して、クライアントサイドのロギング情報とデバッグ情報を示します。ColdFusion の Ajax 機能は、このウィンドウに情報とエラーメッセージを表示します。また、ロギングタグを使用して追加情報 (JavaScript の複合変数の構造など) を表示することもできます。

ユーザーインターフェイス機能

- Ajax ベースの HTML コントロールには、次のものがあります。
  - ツリー
  - グリッド
  - リッチテキストエディタ
  - 日付フィールド
  - 選択候補を表示するテキスト入力
- ポップアップメニューおよびメニューバー
- ボーダーレイアウト、ボックスレイアウト、タブ付きレイアウト、ポップアップウィンドウ、およびポッド領域を提供するコンテナタグ
- HTML の div などの領域での非同期フォーム送信とバインディングを可能にする cfdiv コンテナタグ
- 特定のコントロールおよび HTML 領域に関するツールヒント

ユーザーインターフェイス機能の使用方法の詳細については、788 ページの「Ajax ユーザーインターフェイスコンポーネントおよび機能の使用」を参照してください。

ColdFusion の Ajax タグ

次の表に、ColdFusion の Ajax 関連のタグおよび関数を示します。Ajax ベースの機能がサポートされているタグもすべて含まれます。これらのタグの本文でのみ使用されるサブタグは含まれていません。

データタグ	UI タグ	UI タグ	関数
cfajaximport	cfdiv	cfselect	AjaxLink
cfajaxproxy	cfgrid	cftextarea	AjaxOnLoad
cfsprydataset	cfinput	cftree	DeserializeJSON
	cflayout	cftooltip	IsJSON
	cfmenu	cfwindow	QueryConvertForGrid
	cfpod		SerializeJSON

フォームフィールドへのデータのバインディング

ColdFusion の多くの Ajax 機能ではバインディングがサポートされています。バインディングを使用すれば、ユーザーの入力やデータの変更に基づいてダイナミックに処理を行うことができます。バインディングでは、バインド式が評価され、バインドパラメータで指定したフォームコントロールフィールドで特定のイベント (デフォルトでは onChange) が発生するたびに、新しいデータに基づいて表示が更新されます。ユーザーがフォームデータを入力するなどして情報が変化すると、バインド式を指定したタグの値とその表示がダイナミックに更新されます。バインディングを使用すれば、ページ全体を更新することなく、ページの特定の部分のみを更新することができます。

注意：ウィンドウやタブが表示または選択されていない場合は、それにバインドしているコントロールが変更されても、ウィンドウやタブのコンテンツは更新されません。非表示であるウィンドウやタブにバインドしているコントロールでイベントが発生した場合は、ウィンドウやタブを表示すると更新が行われます。

ColdFusion タグでは、その種類に応じて、バインド式でバインドパラメータ値を直接使用したり、CFC 関数、JavaScript 関数、HTTP リクエストにパラメータとしてバインドパラメータ値を渡し、その関数またはリクエストの応答を使用することで、ページを更新します。バインド式のデータソースには、次のものを使用します。

- ColdFusion フォームコントロールの属性および値。次のコントロールにバインドできます。
  - cfgrid
  - タイプが checkbox、datefield、file、hidden、radio、または text である cfinput
  - cfselect
  - cftextarea
  - cftree
- Spry データセット要素

注意：バインド式を使用して、ダイナミックにロードされる領域内のコントロールをバインドすることはできません。たとえば、cflayoutarea タグで source 属性を使用してコンテンツを指定している場合は、ページ上のコントロールを、そのページのレイアウト領域内のコントロールにバインドすることはできません。ただし、ダイナミックにロードされる領域から、ページ上のコントロールをバインドすることは可能です。したがって、source 属性で指定したファイルでは、cflayoutarea タグが使用されているページ上のコントロールをバインド式に含めることができます。

バインド式の結果によって、タグの値が決定されます。たとえば、cfwindow コントロールの source 属性でバインド式として URL を指定した場合は、その URL で指定したページから返される値が、ウィンドウのコンテンツとして使用されます。

その他の例については、788 ページの「Ajax ユーザーインターフェイスコンポーネントおよび機能の使用」、およびバインディングをサポートするコントロールのリファレンスページを参照してください。

バインド式の使用

bind 式を指定するには、次のいずれかの形式を使用します。

- cfc:**componentPath.functionName(parameters)**

  注意：ColdFusion 9 の場合、コンポーネントパスではマッピングを使用できません。componentPath の値は、**Web** ルートまたは現在のページが含まれているディレクトリからのドット区切りのパスであることが必要です。

- javascript:**functionName(parameters)**
- url:**URL?parameters**
- **URL?parameters**
- いくつかの {**bind parameter**} で構成される文字列 ({firstname}.{lastname}@{domain} など)

1 ～ 4 の形式では、通常、パラメータにバインドパラメータが含まれます。次の表に、バインド式がサポートされているタグ属性と、使用する形式を示します。

属性	タグ	サポートされている形式
autosuggest	cfinput type="text"	1、2、3
bind	cfdiv、cfinput、cftextarea	1、2、3、5
bind	cfajaxproxy、cfgrid、cfselect、cfsprydataset、cftreeitem	1、2、3
onChange	cfgrid	1、2、3
source	cflayoutarea、cfpod、cfwindow	4

使用例を次に示します。

```
bind="cfc:myapp.bookorder.getChoices({book})"
source="/myApp/innerSource/cityWindow.cfm?cityname={inputForm:city}
```

この例の {book} および {inputForm:city} は、book および city コントロールからダイナミックにデータを取得するバインドパラメータです。city コントロールは inputForm フォームにあります。

JavaScript 関数が定義されているページを bind 属性で指定する場合、そのページの関数定義は次の形式であることが必要です。

```
functionName = function(arguments) {function body}
```

次のような関数定義は、正常に動作しない可能性があります。

```
function functionName (arguments) {function body}
```

ただし、カスタム JavaScript すべてを外部の JavaScript ファイルにインクルードして、アプリケーションのメインページにインポートすることをお勧めします。source 属性を使用して取得するコードには、カスタム JavaScript をインラインで記述しないでください。インポートするページには、関数定義に関するこのような制限はありません。

バインドパラメータの指定

バインドパラメータでは、次の例のように、フォームコントロール値やその他の属性を指定します。

```
bind="cfc:myapplication.bookSearch.getStores({form1:bookTitle})"
```

この例では、バインドパラメータは form1:bookTitle であり、form1 フォームの bookTitle フィールドの value 属性を指定しています。

バインドパラメータは、次のいずれかの形式を取ります。

```
{[formName:]controlName[.attributeName][@event]}
{SpryDataSetName.fieldName}
```

角括弧 ([]) はオプションのコンテンツであることを示すものです。パラメータの一部ではありません。

注意：バインド式にリテラルの括弧文字を含めるには、¥{ や ¥} のように円記号 (¥) でエスケープします。

formname の値

formname エントリは、バインドするコントロールが含まれているフォームを識別します。同じ名前のバインドターゲットが複数のフォームに含まれている場合は、フォーム名を指定します。フォーム名を指定するには、フォームの id 属性 (id 属性が未指定の場合は name 属性) を先頭に置き、その後ろにコロン (:) を続けます。たとえば、inputForm というフォームに含まれている book コントロールを指定するには、次の形式を使用します。

```
bind="cfc:myapp.bookorder.getChoices({inputForm:book})"
```

controlName の値

フォームフィールドにコントロールをバインドするには、バインドするフォームコントロールの id 属性か name 属性の値を **controlName** の値に指定する必要があります。コントロールに id 属性と name 属性の両方がある場合は、いずれかの値を使用できます。

cfgrid や cftree など、ColdFusion のフォームコントロールはすべてバインド可能です。cftable などのその他の ColdFusion タグの値はバインドできません。

Spry データセットをバインドするには、バインドパラメータのこの部分にデータセット名を指定します。

複数のラジオボタンやチェックボックスをバインドするには、それらに同一の name 値を指定します。ラジオボタングループ内のすべてのラジオボタンに同じ name 値が指定されている場合、バインドパラメータは選択されたボタンを表します。複数のチェックボックスに同じ name 値が指定されている場合、バインドパラメータは選択されたコントロールの値を列挙したカンマ区切りリストになります。また、各チェックボックスまたはラジオボタンに一意の id 属性を指定する場合、各ボタンまたはチェックボックスに HTML の label タグを指定し、for 属性にその id 値を使用します。この場合、ユーザーはボタンやボックスだけでなく、ラベルをクリックすることによっても項目を選択できます。

cfselect コントロールが複数選択をサポートしている場合、バインド式は選択項目に関する情報のカンマ区切りリストを返します。

バインド可能なコントロールは、バインドが登録されるときに DOM ツリーで利用可能なコントロールのみです。バインドの登録は、バインド式を含むページがブラウザウィンドウまたはコンテナタグにロードされるときに行われます。したがって、2 つの cfdiv、cflayoutarea、cfpod、または cfwindow コンテナで source 属性 (cfdiv タグの場合は bind 属性) を使用して、一方のコンテナ内のコントロールをもう一方のコンテナ内のコントロールにバインドすることはできません。一方のコンテナがロードされるときに、もう一方のコンテナがロード済みであるとは限らないからです。同様に、メインページ上の要素に、ダイナミックにロードされるコンテナ内の要素をバインドすることはできません。この問題を回避するには、source または bind 属性を使用してバインドターゲットを含むマークアップを取得するのではなく、メインページにインラインでバインドターゲットを定義します。つまり、バインド式のソースとして使用するフィールドが含まれている " マスター " フォームはスタティックに (メインページで) ロードし、そのデータを使用する " 子 " コントロールはダイナミックに (source または bind 属性で指定したページで) ロードします。

attributeName の値

フォームコントロールをバインドした場合、デフォルトでは、そのコントロールの value 属性が取得されます。バインドターゲットが cfselect タグの場合は、選択項目の値のカンマ区切りリストが取得されます。

別の属性をバインドするには、コントロールの name または id の後ろに、ピリオド (.) と属性名を指定します。たとえば、チェックボックスである cfinput タグの checked 属性を CFC パラメータとして渡すには、次のような式を使用します。

```
bind="cfc:myapp.bookorder.useStatus({myForm:approved.checked@click})
```

注意：innerHTML という属性名を指定すると、値ではなく、選択ボックスの表示テキストをバインドできます。

注意：チェックボックスをバインドする場合は、@click イベント指定子を使用すると、Internet Explorer でボックスからフォーカスを移動したときではなく、チェックボックスを選択または選択解除したときにバインド式がトリガされるようになります。

グリッドおよびツリーには、デフォルトのバインド属性はありません。

- {**gridID.columnName**} という形式を使用して、グリッドターゲット属性を指定してください。このバインド式によって、選択されている行の指定の列の値が取得されます。
- ツリーの場合は、ツリー内の特定のノードにバインドする必要があります。ノードを指定するには、ノード ID かノードのパスを使用します。

Spry データセット要素または属性をバインドするには、標準の Spry パス表記を使用します。たとえば、要素名を指定します。

event の値

デフォルトでは、バインドパラメータで指定したコントロールで onChange イベントが発生するたびにバインド式の関数が実行されます。他の JavaScript イベントで更新を行うには、バインド式の末尾にアットマーク (@) とイベント名 ("on" 接頭辞を削除したもの) を指定します。たとえば次のコードは、book コントロールの上でマウスボタンが押されるたびに getChoices CFC を実行します。

```
bind="cfc:myapp.bookorder.getChoices({inputForm:book@mousedown})"
```

注意：type 属性が button である cfinput コントロールをバインドする場合は、バインドイベントとして click などを指定します。デフォルトの change イベントのままでは更新は行われません。

Spry データセットをバインドする場合は、イベントを指定しません。データセットの選択行が変更されたり、データセットに新しいデータがリロードされたりすると、式が評価されます。

イベントとして @none を指定すれば、そのバインドパラメータでイベントが発生してもバインド式は再評価されなくなります。これが役立つ場合としては、バインド式で複数のバインドパラメータを使用してさまざまなフォームフィールドをバインドしており、その中の 1 つのフィールドが変更されたときにのみ再評価を行い、他のフィールドが変更されても再評価を行わないようにしたい場合があります。この場合は、他のフィールドに @none を指定すれば、それらのフィールドでイベントが発生してもバインドが行われなくなります。次のコードに、この使用例を示します。

```
bind="cfc:books.getinfo({iForm:book}, {iForm:author@none})"
```

@none イベント指定子は、選択候補を表示するテキスト入力、ツリー、グリッドでも役に立ちます。

- 選択候補を表示するテキスト入力では、ユーザーがテキストを入力するとバインド式が評価され、@none が指定されているパラメータを含むすべてのバインドパラメータからデータが取得されます。選択候補を表示するテキスト入力のすべてのバインドパラメータに @none を指定すれば、パラメータが変更されても更新が行われないようにすることができます。
- ColdFusion.Grid.refresh または ColdFusion.Tree.refresh 関数を呼び出すとバインド式が評価され、@none が指定されているパラメータを含むすべてのバインドパラメータからデータが取得されます。すべてのバインドパラメータに @none を指定すれば、バインドしているコントロールに変更があってもツリーやグリッドは更新されなくなります。明示的に更新を行えばすべてのバインドパラメータからデータが取得されます。

バインド式での CFC 関数の使用

JavaScript 関数と同様に、バインド式の CFC 関数にも位置引数を渡します。位置引数を渡す場合、CFC 関数の定義で使用されている引数名をバインドパラメータ名と同じにする必要はありませんが、CFC 関数の定義と同じ順序で引数を列挙する必要があります。

別の方法として、名前を指定して引数を渡します。名前を指定して渡す場合、CFC 関数の定義と同じ順序で引数を列挙する必要はありませんが、CFC 関数の定義で使用されている引数名をバインド式で指定する必要があります。バインド式で引数名を指定するには、次の形式を使用します。この例では、arg1 および arg2 という 2 つのパラメータを指定しています。

```
bind="cfc:mycfc.myfunction(arg1={myform:myfield1},arg2={myform:myfield2})"
```

Web ルートの外にある CFC のサポート

注意：この機能を使用するには、ColdFusion 9 アップデート 1 をインストールする必要があります。

Web ルートの外のあるコンポーネントにバインド式でアクセスできます。これは、cfajaxproxy などのタグや、グリッド、マップ、プログレスバーなどの Ajax コンポーネントをより効果的な方法で使用できることを意味します。

注意：以前のリリースでは、CFC は、Ajax アプリケーションが関数に Web アクセス可能である必要がありました。

相対パスまたは絶対パスを使用した CFC へのアクセスだけでなく、次のいずれかのメソッドを使用して、CFC にアクセスすることもできます。

- （ColdFusion Administrator で定義されている）論理マッピング
- (Application.cfc で定義されている) アプリケーションごとのマッピング
- (cfimport または import を使用した) インポート

使用方法

次のコードに、この機能強化のアプリケーションごとのマッピングの使用方法を示します。

Application.cfc

```
THIS.mappings["/mycfc"] = "C:¥www¥shared¥components";
```

Test.cfm

```
<cfajaxproxy cfc="mycfc.utils" jsclassname='jsobjname' />
```

例

この例では、"c:¥components" を参照する Application.cfc に、mycfcs という名前のアプリケーションごとのマッピングが既に作成されています。サンプルコードを機能させるには、システムルート（この例の場合は c:¥）に components という名前のフォルダーを作成し、そのフォルダーに Employee.cfc をコピーします。

Application.cfc

```
<cfcomponent>
    <cfset this.name = "cfcoutsidewebroot">
    <cfset this.sessionmanagement = true>
    <Cfset mappingname = "/mycfcs">
    <Cfset mappingpath = "c:\components\">
    <cfset this.mappings[mappingname] = mappingpath>
</cfcomponent>
```

Employee.cfc

```
<cfcomponent>
    <cfscript>
        remote any function getEmployees(page,pageSize,gridsortcolumn="EMP_ID",gridsortdirection="ASC"){
            var startRow = (page-1)*pageSize;
            var endRow = page*pageSize;

            if(!isdefined("arguments.gridsortcolumn") or isdefined("arguments.gridsortcolumn") and
trim(arguments.gridsortcolumn) eq "")
                gridsortcolumn = "EMP_ID";
            if(!isdefined("arguments.gridsortdirection") or isdefined("arguments.gridsortdirection") and
arguments.gridsortdirection eq "")
                gridsortdirection = "ASC";
            var mysql = "SELECT Emp_ID, FirstName,  EMail, Department FROM Employees";
            if(isdefined("arguments.gridsortcolumn") and arguments.gridsortcolumn neq "")
                mysql = mysql & " ORDER BY " & gridsortcolumn;
            if(isdefined("arguments.gridsortdirection") and arguments.gridsortdirection neq "")
                mysql = mysql & " " & gridsortdirection ;
            rs1 = new query(name="team", datasource="cfdocexamples", sql=mysql).execute();
                return QueryConvertForGrid(rs1.getResult(), page, pageSize);
        }


        remote any function editEmployees(gridaction,gridrow,gridchanged){
                writelog("edit employee info");
        }

    </cfscript>
</cfcomponent>
```

Employee.cfm

```
<cfform>
    <cfgrid
            format="html"
            name="grid01"
            pagesize=10
            title="Employee database"

bind="cfc:mycfcs.employee.getEmployees({cfgridpage},{cfgridpagesize},{cfgridsortcolumn},{cfgridsortdirect
ion})"
            onChange="cfc:mycfcs. employee.editEmployees({cfgridaction},{cfgridrow},{cfgridchanged})">
            <cfgridcolumn name="Emp_ID" display=false header="ID" />
            <cfgridcolumn name="FirstName" display=true header="First Name"/>
            <cfgridcolumn name="Email" display=true header="Email"/>
            <cfgridcolumn name="Department" display=true header="Department" />
    </cfgrid>
</cfform>
```

コントロールの属性でのバインディングの使用

直接バインディングを使用する場合は、ColdFusion のフォームコントロールまたは表示コントロールの属性にバインド式を指定します。最も単純なバインディングとしては、次の例のように、あるフォームフィールド (氏名フィールド) を使用して、別のフィールド (電子メールフィールド) を設定する場合があります。名前またはドメインを入力し、Tab キーを押して別のフィールドに移動すると、電子メールフィールドが設定されます。

```
<html>
<head>
</head>
<body>

<cfform name="mycfform">
    First Name: <cfinput type="text" name="firstname" value=""><br>
    Last Name: <cfinput type="text" name="lastname" value=""><br>
    Domain: <cfinput type="text" name="domain" value=""><br>
    E-mail: <cfinput type="text" name="email1" size="30"
        bind="{firstname}.{lastname}@{domain}">
</cfform>
</body>
</html>
```

次の例では、name 属性が同じで id 属性が異なるラジオボタンやチェックボックスをバインドしています。各コントロールでは、label タグの値を id 値に使用しているので、ラベルをクリックすることでコントロールを選択したり選択解除したりできます。

```
<html>
<head>
</head>
<body>

<cfform name="myform">
    Pick one:
    <cfinput id="pickers1" name="pickone" type="radio" value="Apples">
        <label for="pickers1">Apples</label>
    <cfinput id="pickers2" name="pickone" type="radio" value="Oranges">
        <label for="pickers2">Oranges</label>
    <cfinput id="pickers3" name="pickone" type="radio" value="Mangoes">
        <label for="pickers3">Mangoes</label>
    <br>
    <cfinput name="pickone-selected" bind="{pickone}"><br />
    <br />

    Pick as many as you like:
    <cfinput id="pickers4" name="pickmany" type="checkbox" value="Apples">
        <label for="pickers4">Apples</label>
    <cfinput id="pickers5" name="pickmany" type="checkbox" value="Oranges">
        <label for="pickers5">Oranges</label>
    <cfinput id="pickers6" name="pickmany" type="checkbox" value="Mangoes">
        <label for="pickers6">Mangoes</label>
    <br/>
    <cfinput name="pickmany-selected" bind="{pickmany}"><br />

</cfform>
</body>
</html>
```

多くのアプリケーションでは、CFC 関数または JavaScript 関数を呼び出すか、URL を使用して (通常は CFML ページへの) HTTP リクエストを作成し、バインドパラメータを関数または URL パラメータとして渡します。

次の例では、前のセクションの最初の例と同じフォームを使用していますが、バインド式に次の変更が加えられています。

- バインディングをトリガするイベントとして、名前フィールドとドメインフィールドの keyup イベントを指定しています。したがって、これらのいずれかのフィールドに文字が入力されるたびに、電子メールフィールドが更新されます。
- CFC を呼び出しています。この CFC では、電子メールアドレスを生成するために、名の先頭の文字のみを使用し、ドメイン名をすべて小文字にします。

"bindapp.cfm" ページを次の例に示します。

```
<html>
<head>
</head>
<body>
<cfform name="mycfform">
    First Name: <cfinput type="text" name="firstname" value=""><br>
    Last Name: <cfinput type="text" name="lastname" value=""><br>
    Domain: <cfinput type="text" name="domain" value=""><br>
    E-mail: <cfinput type="text" name="email"
        bind="cfc:bindFcns.getEmailId({firstname@keyup},{lastname@keyup},
        {domain@keyup})">
</cfform>
</body>
</html>
```

CFC ファイル "bindFcns.cfc" を次の例に示します。

```
<cfcomponent>
    <cffunction name="getEmailId" access="remote">
        <cfargument name="firstname">
        <cfargument name="lastname">
        <cfargument name="domain">
        <cfreturn "#left(arguments.firstname,1)#.#arguments.lastname#@#lcase(arguments.domain)#">
    </cffunction>
</cfcomponent>
```

ColdFusion の Ajax 機能に関する章には、バインディングを使用した例が数多く掲載されています。その中には、ここで示した例よりも複雑なバインディングを使用しているものもあります。

cfajaxproxy タグによる表示コントロールのバインディング

bind 属性を指定した cfajaxproxy タグを使用することで、ColdFusion の Ajax コントロールをバインドした次の要素を作成することができます。

- 単一の CFC 関数
- 単一の JavaScript 関数
- HTTP リクエスト (CFML ページの URL など)

バインドしたコントロールで特定のイベント (デフォルトでは onChange イベント) が発生するたびに、関数またはリクエストが実行されます。

注意：バインド属性に URL を指定すると、CF_NODEBUG という URL パラメータが HTTP リクエストに挿入されます。この値が true の場合、ColdFusion はデバッグ情報を追加せずに応答を送信します（通常は、応答の最後にデバッグ情報が追加されます）。これによって、Ajax リクエストに対する JSON レスポンスに非 JSON テキスト (デバッグ情報など) が追加されなくなります。

cfajaxproxy タグの次の属性で、関数またはページから返されたデータの処理方法を指定します。

- onError 関数は、HTTP エラーが返されたときの処理コードを指定します。この属性は、URL または CFC バインドで使用します。
- onSuccess 関数では、関数またはページが正常に返されたときの処理を実行します。必要に応じて、返された情報を使用して表示を更新します。

cfajaxproxy タグを使用して関数またはリクエストをバインドすると、サーバーサイドアクションを実行できます。たとえば、特定のコントロールに対するユーザーアクションを検出したら、バインドパラメータ値を使用してデータベースを更新し、サーバーの応答に基づいて何らかのアクションをコントロールで実行することができます。この例では、onSuccess 関数を使用してサーバーからの結果を処理しているので、CFML コントロールの bind パラメータを使用するよりも柔軟な処理が行えます。また、この形式のバインディングを使用すれば、コントロールの bind パラメータで特定のアクションを行うとともに、cfajaxproxy タグで別の処理を行うことが可能になります。

たとえば、編集可能な cfgrid コントロールと、グリッド行を削除するための [削除] ボタンを備えたフォームがあるとします。この場合、アプリケーションは次のように動作する必要があります。

- ユーザーが [削除] ボタンをクリックしたら、次の 2 つの処理を実行する必要があります。
  - CFC の mycfc.deleteButton 関数を呼び出して、データベースから行を削除します。
  - グリッドを更新して、行の削除を反映します。
- ユーザーがグリッドコンテンツを編集したら、グリッドは mycfc.update 関数を呼び出してデータベースを更新する必要があります。

この動作を実装するには、次のようにします。

- cfgrid タグで bind 属性にバインド式を指定して、ユーザーがグリッドコンテンツを変更するたびに mycfc.update 関数が呼び出されるようにします。
- cfajaxproxy タグで bind 属性を指定して CFC の mycfc.deleterow 関数を呼び出すとともに、onSuccess 属性を指定して ColdFusion.Grid.refresh 関数を呼び出し、CFC 関数が正常に返されたときに表示グリッドが更新されるようにします。

そのコードを次に示します。

```
<cfajaxproxybind="cfc:mycfc.deleteRow({deletebutton@click},
        {mygrid.id@none}"onSuccess="ColdFusion.Grid.refresh('mygrid', true)">

...

<cfinput type="button" name="deletebutton">
<cfgrid name="mygrid" bind="mycfc.update({cfgridpage}, {cfgridpagesize},
        {cfgridsortcolumn}, {cfgridsortdirection})>
```

次の例は、cfajaxproxy タグの bind 属性の簡単な使用例です。簡略化のために、バインド式では JavaScript 関数を呼び出しています。したがって、cfajaxproxy タグでは onError 属性を使用できません。

```
<html>
<head>
<script language="javascript">
    function test(x,y){
        return "Hello, " + x + "!";
    }
    function callbackHandler(result){
        alert("Bind expression evaluated. Result: \n" + result);
    }
</script>

<cfajaxproxy bind="javascript:test({input1@none},{button1@click})"
        onSuccess="callbackHandler">
</head>

<body>
<cfform name="mycfform">
        <cfinput type="text" value="" name="input1" size="30">
        <cfinput type="button" name="button1" value="Submit">
</cfform>
</body>
</html>
```

JavaScript でのバインド可能な属性値の取得

JavaScript コードで ColdFusion.Ajax.submitForm 関数を使用すれば、バインド可能なコントロールの任意の属性値を取得できます。この方法は、cfgrid や cftree などの複雑なコントロールの値を取得する場合に役立ちます。詳細については、『CFML リファレンス』の ColdFusion.Ajax.submitForm 関数を参照してください。

クライアントとサーバーの対話の管理

クライアントとサーバーの対話は、次のいくつかの方法で管理します。

- cfajaxproxy タグを使用して、CFC とその関数に対するクライアントサイドの JavaScript プロキシを作成できます。クライアントの JavaScript コードでプロキシ関数を呼び出して、サーバーサイドの CFC 関数にアクセスします。
- cfsprydataset タグを使用すると、URL または CFC から Spry データセットにダイナミックにデータを設定できます。その後、このデータセットを使用して Spry ダイナミック領域にデータを設定します。また、バインド式で Spry データセットを使用することもできます。
- cfajaxproxy タグを使用して、特定の CFC 関数、JavaScript 関数、または HTTP リクエストに ColdFusion の Ajax フォームコントロールのフィールドをパラメータとしてバインドし、成功または失敗した場合の処理を JavaScript 関数として指定できます。関数は、バインド式で指定したイベントが発生するたびに実行されます。
- ColdFusion の Ajax ベースの UI タグ (cftree や cfgrid など) でデータバインディングを行って、CFC または URL から自動的にデータを取得できます。

cfsprydataset タグの使用方法を含む、Spry の操作方法については、854 ページの「ColdFusion での Spry の使用」を参照してください。ColdFusion UI タグと cfajaxproxy タグによるバインディングを含む、バインディングの使用方法の詳細については、841 ページの「フォームフィールドへのデータのバインディング」を参照してください。ColdFusion の Ajax ベース UI タグの使用方法の詳細については、788 ページの「Ajax ユーザーインターフェイスコンポーネントおよび機能の使用」を参照してください。

ColdFusion Ajax CFC プロキシの使用

cfajaxproxy タグを使用して、CFC とその関数に対するクライアントサイドの JavaScript プロキシを作成できます。プロキシオブジェクトには次の特性があります。

- CFC リモート関数に対応する JavaScript 関数が用意されています。クライアントサイドの JavaScript コードでこれらの関数を呼び出すと、サーバー上の CFC 関数がリモートで呼び出されます。
- 通信の制御、非同期の結果およびエラーハンドラコールバックの指定、サーバーへのフォームデータの送信を行うための JavaScript サポート関数が用意されています。これらの関数の詳細については、『CFML リファレンス』の **cfajaxproxy** タグを参照してください。
- クライアントと CFC 間の対話を管理します。たとえば、JavaScript 配列や構造体を Web 上で転送するために必要な、JSON 形式へのシリアル化や JSON 形式からのシリアル化解除を行います。
- CFC 戻り値を自動的にシリアル化 (JSON 形式) またはシリアル化解除します。

ColdFusion の Ajax プロキシを使用すれば、任意の JavaScript コードで間接的に CFC 関数を呼び出すことができます。したがって、ColdFusion の Ajax UI 要素を使用したアプリケーションだけでなく、すべての Ajax アプリケーションで CFC のダイナミックデータを使用できます。また、バインド式では 1 つの関数しか指定できませんが、プロキシを使用すれば CFC のすべての関数にアクセスできます。

JavaScript CFC プロキシの作成

cfc 属性が設定された cfajaxproxy タグを使用すると、Web クライアント上の CFC を表す JavaScript プロキシが生成されます。cfajaxproxy タグを使用する ColdFusion ページは Ajax クライアント Web ページとして使用されるので、通常はページの先頭に cfajaxproxy タグが置かれ、ページの残りの部分は、HTML と、表示の制御やページロジックの実行に必要な JavaScript から構成されます。

注意：JavaScript では大文字と小文字が区別されるので、ColdFusion 構造体のキーやスコープのキーをクライアントに送信するときは、大文字と小文字を正しく使い分ける必要があります。ColdFusion の変数名や構造要素名は、デフォルトでは大文字で記述されます (myStruct["myElement"]="value" のように連想配列表記法で名前を指定すると、構造体の要素名を小文字で作成できます)。たとえば、ColdFusion の SerializeJSON 関数によって生成される、クエリーを表す JSON オブジェクトに含まれている 2 つの配列のキーは、columns と data ではなく、COLUMNS と DATA です。

CFC プロキシの作成方法と使用方法の詳細については、『CFML リファレンス』の cfajaxproxy タグを参照してください。

CFC プロキシの設定

プロキシには、プロキシの動作を制御するための JavaScript 関数が用意されています。

- setAsyncMode および setSyncMode 関数を使用して、呼び出しモードを制御できます。デフォルトでは、リモート CFC 関数の呼び出しはすべて非同期です。これは Ajax アプリケーションで最も一般的な同期モードです。
- setCallbackHandler および setErrorHandler 関数を使用して、非同期呼び出しが成功または失敗した場合の結果を処理する関数を指定できます。

  注意：エラー処理を正常に実行するためには、ColdFusion Administrator の [サーバーの設定]-[設定] ページにある [HTTP ステータスコードの有効化] オプションを選択します。
- setHTTPMethod 関数を使用して、呼び出しに GET HTTP リクエスト (デフォルト) を使用するか POST リクエストを使用するかを制御できます。
- setForm 関数を使用して、すべてのフォームデータをリモート関数に送信するための準備を行えます。この関数を使用すると、各フォームフィールドが個別のパラメータとして CFC 関数に渡されます。
- setReturnFormat 関数を使用して、結果の形式を JSON 形式 (デフォルト)、WDDX 形式、またはプレーンテキスト形式の中から選択できます。setQueryFormat 関数を使用して、JSON 形式のクエリーを返すために使用するオブジェクト (列名と行配列の配列を返すか、WDDX クエリー形式に対応したオブジェクトを返すか) を指定できます。これらの関数は、ColdFusion から返されるデータの形式にのみ影響します。プロキシからサーバーに送信されるデータは、常に JSON 形式です。

CFC へのデータの送信

Ajax CFC プロキシを使用すれば、フォームデータだけでなく、JSON 形式にシリアル化できる任意のクライアントサイドデータを CFC 関数に渡すことができます。ただし、DOM ツリー要素はネイティブコード上のラッパーであるため、シリアル化できません。したがって、Ajax プロキシを使用して呼び出す CFC 関数に DOM ツリー要素を直接渡すことはできません。CFC に渡せるデータ (JSON に正しくシリアル化されるデータ) は、配列、オブジェクト、単純型といった基本的な JavaScript 型のみです。したがって、DOM 要素を直接使用するのではなく、CFC 関数に必要な要素属性のみを (個別に、または配列やオブジェクトに格納して) 渡すことになります。

cfc 属性を使用する場合は、JavaScript で CFC プロキシ関数を呼び出す前に setForm 関数を呼び出して、クライアントページを更新することなくフォームデータを CFC に渡します。setForm 関数を呼び出した後にプロキシ関数を呼び出すと、指定されたフォームのすべてのフィールド値が CFC 関数に渡されます。渡された各フィールドは、フォームコントロールの ID 属性 (この属性が未指定の場合はデフォルトで name 属性) を名前に持ち、コントロールの値を値として持つ引数として、CFC 関数の Arguments スコープに格納されます。

注意：setForm 関数を使用して file フィールドのコンテンツを送信することはできません。

フォームパラメータをプロキシ関数に渡すには、setForm 関数を呼び出した直後にプロキシ関数を呼び出します。その後のプロキシ関数呼び出しでは、フォームパラメータは取得されません。

フォームフィールドに加えて、引数を明示的に渡す必要がある場合は、明示的に渡す引数を表す cfargument タグを、フォームフィールドを表すすべての cfargument タグよりも前に置く必要があります。たとえば、submitForm という次のような JavaScript 関数があるとします。

```
function submitForm() {
    var proxy = new remoteHandler();
    proxy.setCallbackHandler(callbackHandler);
    proxy.setErrorHandler(errorHandler);
    proxy.setForm('myform');
    proxy.setData('loggedIn');
}
```

この例では、"remoteHandler.cfc" の setData 関数を次のように定義します。

```
    <cffunction name="setData" access="remote" output="false">
        <cfargument name="loggedIn">
        <cfargument name="userName">
    ...
```

この例で使用されている userName は、フォームフィールドの名前です。userName の cfargument タグを、loggedIn (明示的に渡す変数) の cfargument タグよりも前に置いた場合、CFC 関数は loggedIn の値を取得できません。CFC 関数では、フォームフィールドの cfargument タグを省略することができます。

例 : 非同期 CFC プロキシの使用

次の例では、リモート CFC メソッドを使用して従業員ドロップダウンリストを設定しています。リストから名前を選択すると、CFC メソッドが呼び出されて従業員に関する情報が取得され、結果が表示されます。

アプリケーションのメインページには、次の行が含まれます。

```
<!--- The cfajaxproxy tag creates a client-side proxy for the emp CFC.
        View the generated page source to see the resulting JavaScript.
        The emp CFC must be in the components subdirectory of the directory
        that contains this page. --->
<cfajaxproxy cfc="components.emp" jsclassname="emp">

<html>
    <head>
        <script type="text/javascript">

            // Function to find the index in an array of the first entry
            // with a specific value.
            // It is used to get the index of a column in the column list.
            Array.prototype.findIdx = function(value){
                for (var i=0; i < this.length; i++) {
                    if (this[i] == value) {
                        return i;
                    }
                }
            }

            // Use an asynchronous call to get the employees for the
            // drop-down employee list from the ColdFusion server.
            var getEmployees = function(){
                // Create an instance of the proxy.
                var e = new emp();
                // If you set a callback handler for the proxy, the proxy's calls
                // are asynchronous.
                e.setCallbackHandler(populateEmployees);
                e.setErrorHandler(myErrorHandler);
            // The proxy getEmployees function represents the CFC
            // getEmployees function.
                e.getEmployees();
            }

            // Callback function to handle the results returned by the
            // getEmployees function and populate the drop-down list.
            var populateEmployees = function(res)
            {
                with(document.simpleAJAX){
                    var option = new Option();
                    option.text='Select Employee';
                    option.value='0';
                    employee.options[0] = option;
                    for(i=0;i<res.DATA.length;i++){
                        var option = new Option();
```

```
            option.text=res.DATA[i][res.COLUMNS.findIdx('FIRSTNAME')]
                + ' ' + res.DATA[i][[res.COLUMNS.findIdx('LASTNAME')]];
            option.value=res.DATA[i][res.COLUMNS.findIdx('EMP_ID')];
            employee.options[i+1] = option;
        }
    }
}

// Use an asynchronous call to get the employee details.
// The function is called when the user selects an employee.
var getEmployeeDetails = function(id){
    var e = new emp();
    e.setCallbackHandler(populateEmployeeDetails);
    e.setErrorHandler(myErrorHandler);
// This time, pass the employee name to the getEmployees CFC
// function.
    e.getEmployees(id);
}
// Callback function to display the results of the getEmployeeDetails
// function.
var populateEmployeeDetails = function(employee)
{
    var eId = employee.DATA[0][0];
    var efname = employee.DATA[0][1];
    var elname = employee.DATA[0][2];
    var eemail = employee.DATA[0][3];
    var ephone = employee.DATA[0][4];
    var edepartment = employee.DATA[0][5];

    with(document.simpleAJAX){
        empData.innerHTML =
        '<span style="width:100px">Employee Id:</span>'
        + '<font color="green"><span align="left">'
        + eId + '</font></span><br>'
        + '<span style="width:100px">First Name:</span>'
        + '<font color="green"><span align="left">'
        + efname + '</font></span><br>'
        + '<span style="width:100px">Last Name:</span>'
        + '<font color="green"><span align="left">'
        + elname + '</font></span><br>'
        + '<span style="width:100px">Email:</span>'
        + '<font color="green"><span align="left">'
        + eemail + '</span></font><br>'
        + '<span style="width:100px">Phone:</span>'
        + '<font color="green"><span align="left">'
        + ephone + '</font></span><br>'
        + '<span style="width:100px">Department:</span>'
        + '<font color="green"><span align="left">'
        + edepartment + '</font></span>';
    }
}

// Error handler for the asynchronous functions.
var myErrorHandler = function(statusCode, statusMsg)
{
    alert('Status: ' + statusCode + ', ' + statusMsg);
```

```
            }

        </script>
    </head>

    <body>
        <!--- The form to display the employee drop-down list and
        employee data. --->
        <form name="simpleAJAX" method="get">
            List of Employees:   
            <select name="employee" onChange="getEmployeeDetails(this.value)">
                <script language="javascript">
                    getEmployees();
                </script>
            </select>
            <br><br>
            <span id="empData"></span>
        </form>
    </body>
</html>
```

データソースからデータを取得する次のコンポーネントは、"application" ディレクトリの "components" サブディレクトリの "emp.cfc" というファイルに含める必要があります。この CFC は、ColdFusion に付属する cfdocexamples データソースを使用します。

```
<cfcomponent>
    <cfset this.dsn = "cfdocexamples">
    <cffunction name="getEmployees" access="remote" returnFormat="json" output="false">
        <cfargument name="empid" required="no" type="string" default="0">
        <Cfquery name="qryEmp" datasource="#this.dsn#">
            select * from Employees
            <cfif empid neq 0>
            where Emp_ID = #empid#
            </cfif>
        </Cfquery>
        <cfreturn qryEmp>
    </cffunction>
</cfcomponent>
```

ColdFusion での Spry の使用

ColdFusion では、1 つのアプリケーションにネイティブの ColdFusion 要素と Spry 要素を混在させることができます。

- ColdFusion タグのバインド式で、Spry データセットを直接使用します。したがって、cfinput などの ColdFusion フォーム要素でダイナミック Spry データセットのフィールドをバインドすることで、データセットが更新されるたびに (たとえば、Spry コントロールやデータセットを読み込んだダイナミック領域でユーザーが項目を選択した場合などに) フォーム要素を更新することができます。

  Spry データセットをバインドするには、データセット名の後に、バインドする要素へのパスを指定します。このパスは、標準の Spry パスシンタックスを使用して記述します。たとえば、dsFilters という Spry データセットに name という列がある場合、name 列の現在の行の値をバインドするには、{dsFilters.name} というバインドパラメータを使用します。バインドパラメータでイベントを指定することはできません。バインド式の再評価は、データセット内の選択行が変更されるたびに実行されます。次の例に、バインドシンタックスを示します。

  ```
  <cfinput name="Input1" type="text"
  bind="CfC:DataManager.getInData(filter={dsFilters.name})
  ```

- Spry データセットでは、データソースとして CFC 関数を使用します。これを行うには、URL を使用してリモート CFC メソッドを呼び出す場合と同様に、Spry.Data.XMLDataSet 関数で CFC の URL を指定します。次の例のように、URL パラメータ method を使用してメソッド名を指定し、追加の URL パラメータで関数にデータを渡します。

```
Spry.Data.XMLDataSet("MyAppMgr.cfc?method=getFilter&filter=scores",
"filters/filter");
```

- cfsprydataset タグを使用すれば、ColdFusion フォームデータに基づいて、Spry XML または JSON データセットをダイナミックに作成および更新できます。作成したデータは、Spry ダイナミック領域などの要素で使用できます。

  次の cfsprydataset タグの例では、cfgrid コントロールで選択されている名前を getData.getProductDetails 関数に渡すことによって、dsProducts という Spry XML データセットを作成しています。データセットは、name 値が変更されるたびに更新されます。

```
<cfsprydataset
    name="dsProducts"
    type="xml"
    bind="CFC:getData.getProductDetails(prodname={myform:mygrid.name})"
    xpath="products/product"
    options="{method: 'POST'}"
    onBindError="errorHandler">
```

ColdFusion の "<Web のルートディレクトリ >/CFIDE/scripts/ajax/spry" ディレクトリには、Spry 1.5 フレームワークリリースが完全収録されています。Spry フレームワークの詳細については、www.adobe.com/go/learn_spry_framework_jp を参照してください。詳細については、『CFML リファレンス』の cfsprydataset タグを参照してください。

Spry データセットの例

この例は、次の処理を行います。

1. CFC 関数を直接使用して、Spry XML データセットに XML ファイルのデータを読み込みます。
2. Spry データの情報を、Spry ダイナミック領域のリストボックスに表示します。
3. Spry データセットの選択項目を使用して、cfgrid コントロールのコンテンツを制御します。cfgrid のバインド式で CFC を呼び出して、Spry XML データセットの選択項目を表すバインドパラメータを渡します。
4. cfsprydataset タグを使用して 2 つ目の Spry XML データセットを作成します。このタグでは、cfgrid コントロールの選択項目をバインドパラメータとして CFC 関数を呼び出します。
5. 2 つ目の Spry データセットの情報を、2 つ目の Spry ダイナミック領域に表示します。

この例では、XML ファイルから読み込んだ Spry リストボックスを使用して、表示する書籍のジャンル (すべて、フィクション、またはノンフィクション) を選択できます。選択されたジャンルに応じて cfgrid コントロールに情報が表示され、選択されたジャンルがテキスト入力コントロールに表示されます。cfgrid コントロールで項目を選択すると、2 つ目の Spry ダイナミック領域に情報が表示されます。

このアプリケーションは、次のファイルから構成されます。

- 表示コントロールと関連ロジックを含む "roundtrip.cfm" ページ
- 次の関数を含む "GridDataManager.cfc" ファイル
  - Spry データセット用の XML を取得する getFilter 関数
  - cfgrid コントロールのコンテンツを取得する getData 関数
  - 選択された書籍の詳細情報を取得する getProduct 関数
- Spry データセット用の XML データが含まれている "Filters.xml" ファイル

イメージを表示するためには、images というサブディレクトリをアプリケーションディレクトリに作成して、cfbookclub データベースの BOOKS テーブルの BOOKIMAGE 列で指定されている名前のイメージを格納します。

"roundtrip.cfm" ページ

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">

<head>
<!--- The screen.css style sheet is provided in the Spry distribution. --->
<link href="screen.css" rel="stylesheet" type="text/css" media="all"/>
<!--- Include the XPath and Spry JavaScript files. --->
<script type="text/javascript"
    src="/CFIDE/scripts/ajax/spry/includes/xpath.js"></script>
<script type="text/javascript"
    src="/CFIDE/scripts/ajax/spry/includes/SpryData.js"></script>

<!--- Create the dsFilters Spry XML data set used to populate the FiltersList dynamic region
    that lists the filters. Call the GridDataManager CFC getFilter method directly from a
    Spry XMLDataSet function because no binding is needed. --->
<script>
    var dsFilters = new
        Spry.Data.XMLDataSet("GridDataManager.cfc?method=getFilter", "filters/filter");
</script>

<!--- Use a cfsprydataset tag with binding to generate a dsProduct Spry data set with details
    about the book grid selection. --->
<cfsprydataset
    name="dsProduct"
    type="xml"
    bind="CFC:GridDataManager.getProductDetails(prodname={bookform:bookgrid.TITLE})"
    xpath="products/product"
    options="{method: 'POST'}"
    onBindError="errorHandler">

<!--- Function to handle bind errors. --->
<script language="javascript">
    errorHandler = function(code,msg){
        alert("Error w/bind occurred. See details below:\n\n" + "Error Code: "
            + code + "\n" + "Error Message: " + msg);
    }
</script>

<!--- Specify the size of the FiltersList Spry dynamic region.
            By default it would be unnecessarily large. --->
<style type="text/css">
<!--
#FiltersList {
    height:100px;
    width: 150px;
}
-->
</style>
</head>

<body>
<!--- A Spry dynamic region containing repeated ListBoxItem controls.
    Each item specifies a filter to use in filling the book list grid.
    The items are populated by the data from the CFC getFilter method. --->
<div id="FiltersList" spry:region="dsFilters" class="SpryHiddenRegion">
    <div spry:repeat="dsFilters" class="ListBoxItemGroup">
        <div class="ListBoxItem"
            onclick="dsFilters.setCurrentRow('{dsFilters::ds_RowID}');"
            spry:selectgroup="feedsList" spry:select="SelectedListBoxItem"
            spry:hover="ListBoxItemHover">
                {dsFilters::description}
        </div>
```

```
    </div>
</div>

<!--- A ColdFusion form with the book list data grid. --->
<cfform name="bookform">
<!--- Create a book list grid.
        Users select the book for which to get details from this grid.
        Populate it with the results of the CFC getData method.
        Pass the method the value of the name field of the selected
        item in the dsfilters Spry dynamic region. --->
    <cfgrid name="bookgrid"
            format="html"
            bind="CfC:GridDataManager.getData(page={cfgridpage},
                pageSize={cfgridpagesize},sortCol={cfgridsortcolumn},
                sortDir={cfgridsortdirection},filter={dsFilters.name})"
            selectMode="browse"
            width=400
            delete="true"
            pageSize=7>
        <cfgridcolumn name="TITLE" header="Name" width=200>
        <cfgridcolumn name="GENRE" header="Type" width=200>
    </cfgrid><br />
    <!--- Show the value of the name field of the selected item in the Spry dynamic region.
    --->
    <cfinput name="filter" bind="{dsFilters.name}">
</cfform>

<hr>

<!--- A Spry dynamic region that uses the dsProduct data set to display information on the
    selected product. --->
<div id="RSSResultsList" spry:detailregion="dsProduct" class="SpryHiddenRegion">
    <strong>{name}</strong><br>
    <img src="images/{bookimage}" alt="product box shot" width="238" height="130"/>
    <div>{desc}</div>
</div>

<hr>

</body>
</html>
```

"gridDataManager.cfc" ファイル

```
<cfcomponent name="GridDataManager">

    <!--- The getFilter function gets the filter XML to populate the dsFilters Spry data set.
        It specifies returnFormat=plain to send XML text. --->
    <cffunction name="getFilter" access="remote" output="false" returnFormat="plain">
        <cffile action="read" file="#ExpandPath('.')#\Filters.xml" variable="filtersxml">
        <cfcontent type="text/xml" reset="yes">
        <cfreturn filtersxml>
    </cffunction>

    <!--- The getData function returns books that match the specified genre, or all books if
        there is no genre. --->
    <cffunction name="getData" access="remote" output="false">
        <cfargument name="page" required="yes">
            <cfargument name="pageSize" required="yes">
        <cfargument name="sortCol" required="yes">
            <cfargument name="sortDir" required="yes">
        <cfargument name="filter" required="no">
        <cfquery name="books" datasource="cfbookclub">
            select TITLE, GENRE from BOOKS
            <cfif isDefined("arguments.filter") AND arguments.filter NEQ "">
                where GENRE = '#arguments.filter#'
            </cfif>
            <cfif arguments.sortCol NEQ "" AND arguments.sortDir NEQ "">
                order by #arguments.sortCol# #arguments.sortDir#
            <cfelse>
                order by TITLE ASC
            </cfif>
        </cfquery>
        <!--- Return the data only for the current page. --->
        <cfreturn QueryConvertForGrid(books, arguments.page,
            arguments.pageSize)>
    </cffunction>

    <!--- The getProductDetails gets data for a single book and converts it to XML for use
        in the dsProduct Spry data set. --->
    <cffunction name="getProductDetails" access="remote" output="false">
        <cfargument name="prodname" default="The Road">
        <!--- Get the information about the book from the database. --->
        <cfquery name="bookDetails" datasource="cfbookclub">
            select TITLE, GENRE, BOOKIMAGE, BOOKDESCRIPTION from BOOKS
                where TITLE = '#arguments.prodname#'
        </cfquery>
        <!--- Convert the query results to XML. --->
```

```
        <cfoutput>
            <cfxml variable="BookDetailsXML" >
                <?xml version="1.0" encoding="iso-8859-1"?>
                <products>
                    <product>
                        <name>#BookDetails.TITLE#</name>
                        <category>#BookDetails.GENRE#</category>
                        <bookimage>#BookDetails.BOOKIMAGE#</bookimage>
                        <desc>#BookDetails.BOOKDESCRIPTION#</desc>
                    </product>
                </products>
            </cfxml>
        </cfoutput>
        <!--- Convert the XML object to an XML string. --->
        <cfset xmldata = xmlparse(BookDetailsXML)>
        <cfcontent type="text/xml" reset="yes">
        <cfreturn xmldata>
    </cffunction>

</cfcomponent>
```

"Filters.xml" ファイル

```
<?xml version="1.0" encoding="iso-8859-1"?>
<filters>

    <filter>
        <filterid>1</filterid>
        <name></name>
        <description>No Filter</description>
    </filter>

    <filter>
        <filterid>2</filterid>
        <name>Fiction</name>
        <description>Look for Fiction</description>
    </filter>

    <filter>
        <filterid>3</filterid>
        <name>Non-fiction</name>
        <description>Look for Nonfiction</description>
    </filter>

</filters>
```

クライアントサイドのサポートファイルの指定

ColdFusion では、デフォルトで次の処理が行われます。

- Ajax ベースの機能に必要なすべてのファイル (クライアントサイド JavaScript や CSS など) が、"<Web のルートディレクトリ >/CFIDE/scripts/ajax" から取得されます。
- 各アプリケーションページで明示的に使用されているタグに必要な JavaScript ファイルのみがインポートされます。

場合によっては、このデフォルトの動作を変更することが必要になります。

カスタムのスクリプトまたは CSS の指定

何らかの理由で、CFIDE ディレクトリのデフォルトの場所を使用できない場合があります。たとえば、ColdFusion Administrator にアクセスできないように、ホストサイトがその場所へのアクセスをブロックしていることがあります。そのような場合は、"CFIDE/scripts" ディレクトリやアプリケーションで使用するサブディレクトリを別の場所に移動します。

また、一部のクライアントサイドファイルをカスタマイズする場合もあります。たとえば、フォームコントロールの外観をカスタマイズした CSS ファイルを特定のアプリケーションにのみ適用することがあります。

どちらの場合も、ColdFusion に新しい場所を通知します。次の一方または両方のディレクトリを指定します。

- ColdFusion の Ajax 機能で必要なすべてのクライアントサイドリソースが含まれているディレクトリ
- ColdFusion の Ajax 機能で必要な CSS ファイルのみが含まれているディレクトリ

クライアントサイドリソースが含まれているディレクトリの指定

ColdFusion の Ajax 機能に必要なクライアントサイドリソースが含まれているディレクトリは、次の任意の方法で指定します。

- すべてのアプリケーションに必要な ColdFusion クライアントサイドファイル (cfform タグで使用するファイルなど) が 1 箇所にある場合は、ColdFusion Administrator の [サーバーの設定]-[設定] ページの [デフォルト ScriptSrc ディレクトリ] フィールドでディレクトリを指定します。指定するディレクトリとそのサブディレクトリは、構造およびコンテンツが "CFIDE/scripts" ディレクトリツリーと同じであることが必要です。
- 特定のページの Ajax 機能に必要なクライアントサイドファイルが 1 箇所にある場合は、cfajaximport タグの scriptsrc 属性を使用してソースディレクトリを指定します。このタグは Administrator の設定よりも優先されますが、標準の cfform 機能で使用されるファイルには影響を与えません。指定するディレクトリには ajax というサブディレクトリが含まれ、その構造とコンテンツが "CFIDE/scripts" ディレクトリツリーと同じであることが必要です。
- 特定のフォームのクライアントサイドソースディレクトリは、cfform タグの scriptsrc 属性で指定します。この設定は、そのフォームとその子コントロールのすべての cfajaximport タグ設定よりも優先されます。指定するディレクトリとそのサブディレクトリは、構造およびコンテンツが "CFIDE/scripts" ディレクトリツリーと同じであることが必要です。

1 つのページで複数のリソースディレクトリを使用する場合は、各 JavaScript ファイルは、初めて必要になったときにページに 1 回のみインポートされることを確認してください。したがって、1 つのページで同じ JavaScript ファイルの別のコピーを使用することはできません。

ColdFusion では、この問題を防ぐために、1 つのページで複数の scriptsrc 属性が指定された場合にはエラーが発生するようになっています。したがって、複数のフォームでカスタムのクライアントサイドリソースファイルが必要な場合は、cfform タグの scriptsrc 属性を使用せず、1 つの cfajaximport タグでそれらの場所を指定します。

CSS ファイルが含まれているディレクトリの指定

cfajaximport タグの cssSrc 属性を使用すると、ColdFusion の Ajax ベースのコントロールに適用する CSS ファイルのみが含まれているディレクトリを指定できます。CSS ファイルが検索されるときは、どの scriptsrc 属性よりもこの属性が優先されます。たとえば、ほとんどのページで scriptsrc ディレクトリツリー内の CSS ファイルを使用する一方、カスタムの外観が必要な一部のアプリケーションページでは cssSrc 属性を指定することができます。

scriptsrc および cssSrc 属性の使用方法の詳細、および指定するディレクトリのコンテンツの要件については、『CFML リファレンス』の cfajaximport タグを参照してください。

タグ固有の JavaScript ファイルのインポート

ColdFusion では、次の場合、Ajax ベースのタグに必要な JavaScript ファイルは自動的にインポートされません。

- cfdiv、cflayoutarea、cfpod、cfwindow などのコンテナタグの source 属性または bind 属性に指定したページで、ColdFusion Ajax ベースのタグを使用している場合。コンテナタグを含むページで cfajaximport タグを指定して、他の

ページにある Ajax 機能タグを tags 属性で指定します。対象のタグが、source 属性を含むページでも使用されている場合は、この処置は必要ありません。

- 必要な ColdFusion Ajax JavaScript 関数のインポートを行っていないページで、ColdFusion.Window.create や ColdFusion.navigate などの ColdFusion の Ajax JavaScript 関数を使用している場合。この場合は、cfajaximport タグを使用して、必要な JavaScript 関数をインポートします。coldFuson.navigate などの、使用するコントロールが限定されていない関数を使用する場合は、属性をすべて省略できます。デフォルトでは、コントロールが限定されないベース関数がインポートされます。ColdFusion.Window.create などの関数を使用する場合は、tags 属性を使用して、関連付けるコントロールを識別します (たとえば、次の行の cfwindow)。

```
<cfajaximport tags="cfwindow">
```

タグ固有の JavaScript ファイルのインポートの詳細については、『CFML リファレンス』の cfajaximport タグを参照してください。

データ交換形式の使用

HTTP 接続を介してやり取りする複雑なデータはすべて、Web 上で転送できる文字列表現にシリアル化する必要があります。Web クライアントアプリケーションで最もよく使用されているのは、XML と JSON です。

通常、ColdFusion の Ajax 機能を使用すると、必要なシリアル化およびシリアル化解除がすべて自動的に行われます。cfajaxproxy タグで作成したプロキシや、CFC 関数を呼び出すバインド式では、自動的に JSON 形式のデータがリクエストされ、JSON データから JavaScript 変数へのシリアル化解除が自動的に行われます。

また、ColdFusion には、Web 交換形式でデータを作成、変換、管理する機能も用意されています。これらの機能は、カスタム Ajax 要素を使用して ColdFusion サーバーからデータを取得する場合などに役立ちます。

ColdFusion のデータシリアル化機能は、HTTP 接続を介して転送される複合データを作成したり利用したりするすべてのアプリケーションで使用します。たとえば、JSON 形式を使用した Web サービスやフィードを作成したい場合、現在公開されている多くの Yahoo! Web サービスでは、単純な URL を使用して JSON 形式のデータを取得できます。

注意：XML データや WDDX データを処理するための ColdFusion タグや関数については、1028 ページの「XML および WDDX の使用」を参照してください。

CFC のリモート戻り値のデータ形式の制御

CFC 関数をリモートから呼び出すと、その戻り値はデフォルトで WDDX 形式に変換されますが、JSON 形式やプレーン文字列データとして返すこともできます。プレーンデータを返す場合、XML オブジェクトは自動的に文字列表現に変換されます。

CFC 関数のデータをリクエストする ColdFusion の Ajax 要素 (バインド式や、cfajaxproxy タグで生成される関数プロキシなど) は、HTTP URL の returnFormat パラメータを自動的に生成して、CFC 関数に JSON 形式のデータをリクエストします。

CFC 関数の戻り値の形式は、次の方法で制御します。

- cffunction タグで returnFormat 属性を使用する
- CFC 関数を呼び出す HTTP リクエストに returnFormat パラメータを設定する
- CFC プロキシの setReturnFormat 関数を使用する (この方法は、クライアントサイドコードで XML や WDDX などの非 JSON 形式データが必要になる場合にのみ使用します)

関数からクエリーを取得する場合、リクエストで戻り値の形式として JSON を指定すると、クエリーは次のいずれかの形式の JSON オブジェクトにシリアル化されます。

- 2 つのエントリを含む JSON オブジェクト。2 つのエントリとは、列名の配列と、列データ配列の配列です。

  これらのエントリが返されるのは次の場合です。

  - デフォルト
  - HTTP URL パラメータとして queryFormat="row" を指定した場合
  - cfajaxproxy タグを使用し、パラメータ値 row を渡してプロキシオブジェクトの setReturnFormat 関数を呼び出した場合

  ColdFusion のクライアントサイドバインディングやプロキシコードでは、このデータが自動的に JavaScript 形式に変換され、HTML グリッドで直接利用されます。

- 3 つのエントリを含む JSON オブジェクト。3 つのエントリとは、行数、列名の配列、キーが列名で値が列データの配列であるオブジェクトです。

  これらのエントリが返されるのは次の場合です。

  - HTTP URL パラメータとして queryFormat="column" を指定した場合
- cfajaxproxy タグを使用し、パラメータ値 column を渡してプロキシオブジェクトの setQueryFormat 関数を呼び出した場合

  ColdFusion のクライアントサイドバインディングやプロキシコードでは、列形式データは HTML グリッドで直接利用される JavaScript 形式に変換されません。cfajaxproxy タグでこの形式を使用すれば、列名を直接使用してデータを参照できます。たとえば、ユーザーデータを含むクエリーを CFC 関数が返す場合は、JavaScript で userData.firstName[0] や userData.lastName[0] のように記述してユーザー名を取得します。

詳細については、『CFML リファレンス』の SerializeJSON 関数を参照してください。

JSON の使用

JSON (JavaScript Object Notation) は、コンピュータシステム間の転送に役立つ JavaScript ベースの軽量のデータ交換形式です。XML や WDDX よりも単純な形式なので、Ajax アプリケーションで使用されるデータをコンパクトかつ効率的に転送できます。ColdFusion の Ajax バインド式で CFC を使用すると、CFC 関数が JSON 形式でデータを返すように、HTTP リクエストに returnformat="json" パラメータが含められます。JSON 形式の結果は自動的に処理されます。

JSON では、オブジェクトは { **key** : **value** , **key** : **value...** } という形式で表されます。配列は、[**value** , **value**...] という標準的な形式で表されます。値には、文字列、数値、オブジェクト、配列、true、false、または null を指定できます。したがって、配列やオブジェクトをネストすることができます。JSON 形式の詳細な仕様については、www.JSON.org を参照してください。

ColdFusion の Ajax ベースのコントロールや cffunction タグで JSON 形式の変換が自動的に行われることを利用すれば、他のアプリケーションで JSON 形式のデータを活用することができます。実際、現在多くのフィードで JSON 形式がサポートされています。たとえば、Yahoo 検索インターフェイスは JSON データセットを返します。del.icio.us の JSON フィードでは、ユーザーのポストやタグを取得できます。Blogger でも JSON 形式のフィードが提供されています。これらのフィードを表示するために Ajax を使用する必要はありません。標準の ColdFusion タグおよび関数を使用すれば結果を表示できます。

サーバーサイドコードでの JSON 形式の使用をサポートする CFML 関数としては、次のものがあります。

- DeserializeJSON
- SerializeJSON
- IsJSON

これらの関数の詳細と例については、『CFML リファレンス』を参照してください。

次の例に、非 Ajax アプリケーションで ColdFusion JSON 関数を使用する方法を示します。この例では、Yahoo で "ColdFusion Ajax" を検索して、次の結果を表示します。

- 見つかった Web ページの数
- 返された結果 (デフォルトでは 10 件) のタイトルと要約タイトルは、Web ページの URL にリンクされています。

```
<!--- Send an http request to the Yahoo Web Search Service. --->
<cfhttp
    url='http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=YahooDemo&query=
    "ColdFusion Ajax"&output=json'>

<!--- The result is a JSON-formatted string that represents a structure.
        Convert it to a ColdFusion structure. --->
<cfset myJSON=DeserializeJSON(#cfhttp.FileContent#)>

<!--- Display the results. --->
<cfoutput>
    <h1>Results of search for "ColdFusion 9"</h1>
    <p>There were #myJSON.ResultSet.totalResultsAvailable# Entries.<br>
    Here are the first #myJSON.ResultSet.totalResultsReturned#.</p>
    <cfloop index="i" from="1" to="#myJSON.ResultSet.totalResultsReturned#">
        <h3><a href="#myJSON.ResultSet.Result[i].URL#">
            #myJSON.ResultSet.Result[i].Title#</a></h3>
        #myJSON.ResultSet.Result[i].Summary#
    </cfloop>
</cfoutput>
```

数値のシリアル化

注意：この機能を使用するには、ColdFusion 9 アップデート 1 をインストールする必要があります。

以前のリリース（ColdFusion 9 を含む）では、serializeJSON を使用して整数をシリアル化すると、数値は double に変換されます。たとえば、SerializeJSON (123) では 123.0 が返されます。

ColdFusion 9.0.1 では、整数は元の形式のままですが、引用符で囲まれます。つまり、SerializeJSON (123) では "123" が返されます。これは、正の整数に対してのみ適用されます。

次の表に、もう少し例を示します。

入力	シリアル化された JSON
10	“10”
012	“012”
10.25	“10.25”
10.25E5	“1025000.0”
10.25E-5	“1.025E-4”
-10	“-10.0”
-10.25	“-10.25”

注意：戻り値の引用符を除去するには、正の整数および引用符で囲まれた整数の場合は、JVM 引数の json.numberasdouble を true に設定します。一方、-10 のような負の整数の場合は、json.numberasdouble=true を設定しても “-10.0” にシリアル化されます。

ColdFusion 9.0.1 での Ajax プラミングの拡張

ORM CFC で、プロパティの remotingfetch 属性がサポートされています。

デフォルトでは、remotingfetch は false に設定されています。

ColdFusion で、ORM CFC のシリアル化が実行されると、remotingfetch プロパティが内部検査されます。プロパティが false の場合、ColdFusion から関係についての情報は一切返されません。

true に設定されている場合は、関係についての情報が表示されます。循環参照が検出された場合は、関係の 1 レベルのみが表示されます。

Ajax アプリケーションのデバッグ

ColdFusion には、ポップアップ表示ウィンドウに情報をロギングする JavaScript 関数が用意されています。クライアントサイドの多数の標準アクティビティもこのウィンドウにロギングされます。

ロギング情報の表示

ロギングウィンドウを表示する手順は、次のとおりです。

1 ロギングウィンドウに情報を送信するように ColdFusion を設定します。

2 メイン CFML ページリクエストでロギングウィンドウ情報をリクエストします。

ロギング出力の有効化

ロギングウィンドウに情報を送信するように ColdFusion を設定する手順は、次のとおりです。

- ColdFusion Administrator の [デバッグとロギング]-[デバッグ出力の設定] ページで [AJAX デバッグログウィンドウの有効化] オプションを選択します。ロギングウィンドウに例外メッセージを表示するには、[デバッグ出力の設定] ページで [Robust 例外情報の有効化] オプションを選択します。
- ColdFusion Administrator の [デバッグとロギング]-[デバッグする IP アドレス] ページで、デバッグを実行するシステムの IP アドレスが指定されていることを確認します。デフォルトでこのリストに含まれるのは、127.0.0.1 のみです。

ページのロギング情報の表示

ブラウザで CFML ページをリクエストしたときにロギングウィンドウを表示するには、次のように、ページをリクエストするときに URL に HTTP パラメータ cfdebug を指定します。

```
http://localhost:8500/myStore/products.cfm?cfdebug
```

表示されたデバッグログウィンドウは、ユーザーがブラウザで新しいページに移動するまで実行されます。ロギングウィンドウに含まれるオプションを使用すると、次のいずれかまたは両方の条件によってメッセージをフィルタリングできます。

- 厳格度
- カテゴリ

デバッグ、情報、エラー、ウィンドウの 4 つの厳格度を任意に組み合わせて、表示するロギング情報を選択できます。メッセージの厳格度は、どのロギング関数を呼び出すかによって決まります。

ロギングウィンドウには、標準カテゴリを使用して出力をフィルタするオプションが常に表示されます。標準カテゴリは、バインド、グローバル、http、LogReader、およびウィジェットです。カテゴリの使用方法の詳細については、865 ページの「標準の ColdFusion ロギングメッセージ」を参照してください。また、ColdFusion のロギング呼び出しで指定したカスタムカテゴリに対するフィルタオプションも表示されます。作成できるカテゴリ数に制限はありませんが、アプリケーションを効率的にデバッグする上で必要なカテゴリのみに限定することをお勧めします。

情報のロギング

ColdFusion には、ロガーに情報を送信する次の JavaScript 関数が用意されています。それらの多くは、厳格度に対応しています。

関数	厳格度	用途
ColdFusion.Log.debug	デバッグ	問題のデバッグに役立つメッセージ。
ColdFusion.Log.dump	デバッグ	単一の変数を cfdump に似た形式で表示します。この関数を使用すれば、JavaScript の配列変数やオブジェクト変数の構造とコンテンツを表示できます。
ColdFusion.Log.error	エラー	エラーに関する情報。この関数は、エラー処理のコードでのみ使用してください。
ColdFusion.Log.info	情報	正常に動作しているコードに関する情報。クライアントサイドコードの実行状況の追跡や分析などに役立ちます。

ウィンドウレベルのメッセージを生成することはできません。このレベルは、ログリーダーウィンドウが生成するメッセージ (ログ関数呼び出しでの JavaScript エラーに関する情報など) のために予約されています。

ロギング関数を呼び出すときは、メッセージとカテゴリを指定します。

- メッセージには、JavaScript 変数や HTML マークアップ (太字や改行など) を含めることができます。
- カテゴリには、わかりやすい簡潔な名前を付けてください。各カテゴリに対して、ロギングウィンドウの出力をフィルタリングするためのチェックボックスオプションが生成されます。このパラメータはオプションです。デフォルト値はグローバルです。標準の ColdFusion カテゴリまたはカスタムカテゴリを指定できます。

ページの情報をロギングするには、そのページに ColdFusion Ajax タグがあるか、cfajaximport タグを使用する必要があります。cfajaximport タグを使用する場合、ロギングを有効にする属性を指定する必要はありません。

次のロギング関数は、Pod A カテゴリに属する error レベルのログメッセージを生成します。

```
ColdFusion.Log.error("<b>Invalid value:</b><br>" + arg.A, "Pod A");
```

標準の ColdFusion ロギングメッセージ

ColdFusion は、次のカテゴリのメッセージを自動的にロギングします。

カテゴリ	説明
グローバル	(デフォルト) ColdFusion Ajax ライブラリ内からロギングされないメッセージ。ロギングインフラストラクチャの初期化など。
http	HTTP 呼び出しとその応答に関する情報。HTTP リクエストのコンテンツや、CFC 呼び出しと応答に関する情報など。
LogReader	ログ表示ウィンドウに関するメッセージ。
バインド	バインド関連のアクション。バインド式の評価など。
ウィジェット	コントロール固有のアクション。ツリーやグリッドの生成など。

Ajax プログラミングのルールとテクニック

次に示すテクニックは、Ajax アプリケーションのエラーを回避し、アプリケーションのセキュリティを強化し、効果的なアプリケーションを開発する上で役立ちます。

エラーの回避

次のルールとテクニックは、アプリケーションのエラーを回避する上で役立ちます。

- コードが正常に動作するように、すべてのページ (ダイナミックにロードされるコンテンツや、ダイナミック領域が含まれているページも含みます) に、有効な html、head、body タグがあり、すべての script タグがページの head 領域に配置されていることを確認します。この確認は、ColdFusion の Ajax タグや script タグを使用しているページでは重要です。これによって、スクリプトコードが正常に処理され、コードが正しい順序で生成されるようになります。また、Internet Explorer などのいくつかのブラウザでの問題も回避されます。

- ダイナミックに読み込むすべてのページ (バインド式や ColdFusion.navigate 関数を使用したり、ColdFusion の Ajax コンテナタグでフォームを送信したりして読み込むすべてのページ) では、JavaScript 関数を定義するときに次のシンタックスを使用する必要があります。

  ```
  functionName = function(arguments) {function body}
  ```

  次のような関数定義は、正常に動作しない可能性があります。

  ```
  function functionName (arguments) {function body}
  ```

  ただし、アドビ システムズ社では、ダイナミックに取得するコードにインラインでカスタム JavaScript を記述するのではなく、外部の JavaScript ファイルにすべてのカスタム JavaScript を記述してアプリケーションのメインページにインポートすることをお勧めします。インポートするページには、関数定義に関するこのような制限はありません。

- CFM ページでは、そのページにバインドされたファイル内にある JavaScript 関数を呼び出す場合、関数の宣言でキーワード var を使用しないでください。var は、関数ローカルスコープの変数を宣言する場合に使用します。したがって、親ページから JavaScript 関数を呼び出すことはできません。

- 通常、コントロールの id 属性、または id 属性が未指定の場合は name 属性が、そのページ (source 属性で指定したすべてのページも含む) で一意である必要があります。このルールには、次のような例外があります。

  - ラジオボタングループ内のオプションに同じ name 属性を使用することができます。バインド式には、選択されているボタンに関する情報が返されます。

  - グループ内のチェックボックスに同じ name 属性を使用することができます。これによって、グループ内で選択されているすべてのコントロールに関する情報を 1 つのバインド式で取得することができます。

  - ページ上に複数の似たフォームがある場合は、各フォームに同じ名前または同じ ID のコントロールが含まれていることがあります。この場合は、バインドパラメータにフォーム名を含めることによって、個々のコントロールを識別します。

- CFC 関数を呼び出してデータを取得する cfajaxproxy タグやバインド式を使用しているアプリケーションでは、Application.cfc の onRequestEnd 関数や onRequestEnd.cfm ページを使用して出力を作成しないでください。通常、ColdFusion の Ajax 機能では、サーバーから返すデータはすべて JSON 形式であることが必要です。onRequestEnd メソッドや onRequestEnd.cfm ページを使用すると、返すデータの末尾に非 JSON 情報が追加されてしまいます。

- ColdFusion の構造要素名は、デフォルトではすべて大文字になります。したがって、cfajaxproxy タグで指定する onSuccess 関数などのクライアントサイドの Ajax コードでは、要素名に明示的に小文字を含めている場合を除き、戻り値オブジェクトの要素名に大文字を使用する必要があります (myStruct["myElement"]="value" のように連想配列表記法で名前を指定すれば、小文字を使用した構造要素名を作成できます)。

- HTML の形式が正しくないためにブラウザの DOM 階層順序でエラーが発生すると、ColdFusion の Ajax コントロールで JavaScript エラーが発生することがあります。形式が正しくない HTML の例としては、テーブルに cfform タグが含まれ、そのタグにテーブル行が含まれている場合があります。この場合は、cfform タグの中にテーブルタグを含めます。

アプリケーションの外観や動作に影響する可能性があるブラウザ固有の問題とその他の問題については、Adobe Web サイトにある ColdFusion のリリースノート www.adobe.com/go/learn_cfu_docs_jp および ColdFusion デベロッパーセンター www.adobe.com/go/prod_techarticles_jp を参照してください。

セキュリティの強化

ColdFusion には、Ajax アプリケーションのセキュリティを強化する上で役立つ機能がいくつか用意されています。また、ColdFusion Administrator では、クライアントサイドのロギングウィンドウへの出力はデフォルトで無効になっています (864 ページの「ロギング出力の有効化」を参照してください)。

- クライアントで実行されるコードにリモート URL を使用することはできません。これは、クロスサイトスクリプティングを防止するためです。たとえば、cfwindow タグの source 属性に http://www.myco.com/mypage.cfm のような URL を使用した場合、リモートページはウィンドウにロードされず、エラーメッセージが表示されます。リモート URL にア

クセスする必要がある場合は、サーバー上で実行される CFML コードでアクセスを行います。たとえば、source 属性で指定されたページで cfhttp タグを使用してアクセスします。

- CFC 関数が JSON 形式でリモートデータを返す場合、デフォルトでは接頭辞やラッパーは追加されませんが、JSON データを利用したクロスサイトスクリプティング攻撃を防ぐために、返されるデータに接頭辞を追加することができます。これを行うにはいくつかの方法があります。次のリストの各項目の値は、その前の項目によって決まります。

  1. Administrator で、[サーバーの設定]-[設定] ページの [シリアル化 JSON への接頭辞付加] オプションを有効にします (デフォルト値は false)。また、この設定を使用して接頭辞の文字も指定できます。デフォルトの接頭辞は // です。これは JavaScript のコメントマーカーであり、返される JSON コードはブラウザでコメントとして処理されます。接頭辞 // を使用することで、戻り値が等価の JavaScript オブジェクトに変換されるのを防止し、クロスサイトスクリプティング攻撃を回避できます。
  2. Application.cfc ファイルの This.secureJSON および This.secureJSONPrefix 変数値を設定するか、cfapplication タグの secureJSON および secureJSONPrefix 属性を設定します。
  3. cffunction タグの secureJSON 属性を設定します。ただし、cffunction タグを指定して接頭辞を設定することはできません。

  通常、クレジットカード番号などの機密データを返す CFC ページや CFML ページでは、このいずれかの方法を使用してください。

  これらの方法を使用してセキュリティ接頭辞を追加しても、CFC 関数を呼び出した ColdFusion Ajax 要素 (バインド式や、cfajaxproxy タグで作成される CFC プロキシなど) では自動的に接頭辞が削除されるので、クライアントサイドコードを変更する必要はありません。

- ColdFusion には、不正なユーザーがサーバーに対してセキュリティ攻撃を行う (パスワードの変更などを試みる) のを防ぐための機能が用意されています。次の方法を使用すれば、CFML ページやリモート CFC 関数へのリクエストが、アプリケーションの有効な構成要素である ColdFusion Ajax 機能 (バインド式や CFC プロキシなど) から送信されたものであることを保証できます。
  - Ajax クライアントにデータを返す CFC 関数の cffunction タグで、verifyClient 属性に値 yes を指定します。
  - ColdFusion Ajax クライアントからリクエストされる CFML ページまたは関数の冒頭で、ColdFusion の VerifyClient 関数を呼び出します。この関数はパラメータを取りません。

  VerifyClient 関数または属性を指定すると、暗号化されたセキュリティトークンが各リクエストで求められるようになります。この関数を使用するには、アプリケーションでクライアント管理またはセッション管理を有効にします。有効にしていない場合、エラーは発生しませんが、ColdFusion でクライアントの検証は行われません。

  クライアント検証を有効にするのは、ColdFusion Ajax クライアントコードに応答するコードのみにしてください。クライアントサイドのサポートコードが含まれているのは ColdFusion Ajax ライブラリのみです。ColdFusion Ajax アプリケーション以外のクライアントでクライアント検証を有効にすると、クライアントアプリケーションが動作しなくなる可能性があります。

  通常は、機密性の高いアクション (パスワードの更新など) をサーバーに対して実行する Ajax リクエストでこの関数を使用します。保護の必要がない検索エンジン Web サービスのパブリック API においては、クライアント検証を有効にしないのが普通です。また、アプリケーションのトップレベルのページではクライアント検証を有効にしないでください。ユーザーがブラウザのアドレスバーに URL を入力するときはセキュリティトークンを使用できません。

効率的なプログラミング

次の推奨事項は、ColdFusion Ajax アプリケーションの強化やカスタマイズに役立ちます。

- AjaxOnLoad 関数を使用すれば、ページロード時に実行する JavaScript 関数を指定して、ページを正しく機能させるために必要な初期化アクションを実行できます。コンテナタグにページがロードされるときは、AjaxOnLoad 関数を使用して関数を呼び出します。たとえば、ページが表示されたときにユーザーがまだログインしていない場合は、この関数を使

用してログインウィンドウを表示できます。AjaxOnLoad 関数で指定する JavaScript 関数では、ログインステータスを判定し、必要に応じてウィンドウを表示します。

- 次の ColdFusion JavaScript 関数を使用すれば、境界線スタイルおよびタブスタイルの cflayout コントロール、cfwindow コントロール、HTML 形式の cfgrid、および cftree コントロールを基盤とする Ext JS または Yahoo YUI JavaScript ライブラリのオブジェクトにアクセスできます。次に、そのオブジェクトを処理して、表示されたコントロールを変更します。
  - ColdFusion.Layout.getBorderLayout
  - ColdFusion.Grid.getGridObject
  - ColdFusion.Layout.getTabLayout
  - ColdFusion.Tree.getTreeObject
  - ColdFusion.Window.getWindowObject

  これらのオブジェクトの詳細と管理方法については、Ext のマニュアル (extjs.com/deploy/ext/docs/) および Yahoo ツールキットマニュアル (developer.yahoo.com/yui/) を参照してください。

第 12 章：Office ファイルとの相互運用性

Adobe ColdFusion には、PDF、Adobe Flash、および Adobe Connect と連携するためのインターフェイスが用意されています。ColdFusion 9 では、サードパーティ製品との統合機能が OpenOffice および Microsoft Office アプリケーション (Excel、PowerPoint、SharePoint など) にまで拡張されています。

Office との相互運用性では、OpenOffice ライブラリと Apache POI ライブラリの両方がサポートされています (Apache POI の詳細については、http://poi.apache.org/ を参照してください)。OpenOffice ライブラリは、Word ドキュメントを含むすべての Office ファイル形式から PDF への変換をサポートします。cfdocument、cfpresentation または cfspreadsheet タグを使用して Office ファイルの変換を試みると、OpenOffice がインストールされているかどうかが最初に確認されます。

OpenOffice のインストールが見つからない場合は、POI ライブラリが使用されます。POI ライブラリは、Word ドキュメントを除くすべての Office ファイルの変換をサポートします。

サポートされている Office 変換形式の完全なリストについては、873 ページの「サポートされている Office 変換形式」を参照してください。

cfdocument の使用

既存の機能に加え、cfdocument タグで、Word ドキュメントおよび PowerPoint プレゼンテーションを PDF に変換できるようになりました。Microsoft Word のすべてのバージョンおよび Microsoft PowerPoint のバージョン 97 から 2003 までがサポートされています。

OpenOffice を使用したドキュメントの操作

OpenOffice は、ワードプロセッサ、表計算、プレゼンテーションなどをサポートするオープンソースのオフィスソフトウェアです。OpenOffice では、国際的オープン標準を使用してデータが保存されます。詳細については、http://www.openoffice.org/ を参照してください。

ColdFusion 9 は OpenOffice をサポートし、cfdocument タグを使用して Word ドキュメント (.doc 形式) を PDF に変換します。

cfdocument を使用してドキュメントファイルの変換を試みると、OpenOffice がインストールされているかどうかが最初に確認されます。OpenOffice のインストールが見つかった場合は、OpenOffice ライブラリを使用してリッチテキスト変換が処理されます。

変換後の PDF ドキュメントを開く際には、cfdocument の userPassword 属性と permissions 属性が使用されます。
OpenOffice のドキュメント変換をサポートする cfdocument の属性の詳細については、『CFML リファレンス』を参照してください。

OpenOffice をインストールするには、http://download.openoffice.org/index.html にアクセスしてください。
OpenOffice のインストールおよび設定の詳細については、『ColdFusion 9 インストール』を参照してください。

例

次の例では、MyDocument.doc ドキュメントを PDF ファイルに変換します。PDF 変換は、format 属性を "pdf" に設定した場合にのみ実行されます。

注意：変換を行う場合は、"c:¥documents¥MyDocument.doc" のように、必ず絶対パスを指定します。

```
<cfdocument
    format="pdf"
    srcfile="C:\documents\MyDocument.doc"
    filename="C:\documents\MyDocument.pdf">
</cfdocument>
```

注意：filename 属性を指定していない場合は、変換後の PDF がブラウザで開かれます。

PowerPoint プレゼンテーションファイルの操作

PowerPoint プレゼンテーション (PPT ファイル) を PDF ドキュメントに変換するには、cfdocument タグを使用します。

例

次の例では、PowerPoint プレゼンテーションを PDF ファイルに変換します。

```
<cfdocument
    format="pdf"
    srcfile="C:\presentations\MyPresentation.ppt"
    filename="C:\presentations\MyPresentation.pdf">
</cfdocument>
```

cfpresentation の使用

cfpresentation タグは、プレゼンテーションのコンテンツを定義する cfpresentationslide タグ、およびスライドのプレゼンタに関する情報を提供する cfpresenter タグの親タグになります。

PowerPoint プレゼンテーションを Adobe Connect プレゼンテーションまたは HTML に変換するには、cfpresentation タグを使用します。Internet Explorer、Mozilla Firefox、Safari などのブラウザは、PPT から Connect プレゼンテーションまたは HTML への変換と互換性があります。

cfpresentation および cfpresentationslide の詳細については、『CFML リファレンス』を参照してください。

例

次の例では、PowerPoint プレゼンテーションを Adobe Connect プレゼンテーションに変換します。

```
<cfpresentation
    title="my presentation"
    directory="C:\presentations\"
    overwrite=true>
    <cfpresentationslide
        src="#ppttemplate#backgrounds.ppt"
        slides="1">
    </cfpresentationslide>
    <cfpresentationslide
        duration="4"
        video="video1.flv">
            Sample slide
    </cfpresentationslide>
</cfpresentation>
```

次の例では、HTML ファイルを PowerPoint プレゼンテーションに変換します。

```
<cfpresentation
    title = "text string"
    format= "ppt"
    destination="#generated#html_to_ppt_01.ppt"
    backgroundColor = "YELLOW"
    overwrite = "yes">
    <cfpresentationslide
        Title="Q1 Sales Figures"
        duration="14">


<h3>Q1 Sales Figures</h3>
<cfchart
    format="png"
    showborder="yes"
    chartheight="250"
    chartwidth="300"
    pieslicestyle="sliced">
    <cfchartseries type="pie">

    <cfchartdata
    item="Europe"
    value="9">

<cfchartdata
    item="Asia"
    value="20">

<cfchartdata
    item="North America"
    value="50">

<cfchartdata
    item="South America"
    value="21">
</cfchartseries>
</cfchart>
</cfpresentationslide>

<cfpresentationslide
    src="cfdocument_pos24.html"
    duration="15" />
</cfpresentation>
```

次の例では、PowerPoint プレゼンテーションを Connect プレゼンテーションに変換します。

```
<cfpresentation
    title="my presentation"
    directory="C:\presentations"
    overwrite=true>
    <cfpresentationslide
        src="#ppttemplate#backgrounds.ppt"
        slides="1-3,5">
    </cfpresentationslide>
    <cfpresentationslide
        duration="4"
        video="video1.flv">
            Sample slide
    </cfpresentationslide>
</cfpresentation>
```

cfspreadsheet の使用

cfspreadsheet タグを使用すると、Excel スプレッドシートを管理できます。このタグを使用すると、次のことを行えます。

- スプレッドシートファイル (XLS ファイル) を読み取り、そのデータを ColdFusion スプレッドシートオブジェクト、クエリー、CSV 文字列、または HTML 文字列に格納する。
- クエリー、ColdFusion スプレッドシートオブジェクト、または CSV 文字列変数から新規の XLS に単一のシートを書き込む。
- 既存の XLS ファイルにシートを追加する。

スプレッドシート内の行と列、およびそのデータを操作するには、スプレッドシート関数を使用します。また、スプレッドシート内のセルのコメント、値、および式を指定または取得することもできます。

Microsoft Office Excel 2007 は、cfspreadsheet と、次のものを除くすべてのスプレッドシート関数でサポートされています。

- SpreadSheetAddSplitPane
- SpreadSheetAddFreezePane

cfspreadsheet とすべてのスプレッドシート関数の詳細については、『CFML リファレンス』を参照してください。

例

次の例では、スプレッドシートファイル (SingleSheet.xls) を読み取り、スプレッドシートのデータを CSV 文字列に格納します。

```
<cfspreadsheet action = "read"
    format="csv"
    src="C:\documents\SingleSheet.xls"
    name="csvvar"
    rows="1-4,5,6,7-8">
    <cfoutput>#csvvar#</cfoutput>
```

次の例では、スプレッドシートファイル (template_02.xls) を読み取り、スプレッドシートのデータをクエリーに格納します。

```
<cfspreadsheet
        action = "read"
        src="C:\dcuments\template_02.xls"
        query="excelquery"
        sheet="1"
        rows="1-3,4-5"
        columns="1,4">
<cfoutput
        query="excelquery"
        startrow="1"
        maxrows="#excelquery.recordcount#">
        #excelquery.col_1#
        #excelquery.col_2#
</cfoutput>
```

次の例では、スプレッドシートファイル (template_08_Charts_Graph.xls) を読み取り、スプレッドシートのデータを HTML 文字列に格納します。

```
<cfspreadsheet
        action = "read"
        format="html"
        src="C:\documents\template_08_Charts_Graph.xls"
        name="report1"
        rows="5-11"
        columns="1-6">
<cfoutput>
        #report1#
</cfoutput>
```

次の例では、クエリーのデータをスプレッドシートファイル (SingleSheet1.xls) 内の単一のシートに書き込みます。

```
<cfquery
        name="excelquery"
        datasource="cfdocexamples">
        SELECT PARKNAME, REGION, STATE FROM Parks WHERE STATE='WI'
        ORDER BY ParkName, State
</cfquery>
<cfspreadsheet
        action = "write"
        filename="C:\SingleSheet1.xls"
        query="excelquery"
        overwrite="true">
```

サポートされている Office 変換形式

次の表に、Office アプリケーションでサポートされている変換形式と、その変換をサポートする CFML タグを示します。また、その変換に OpenOffice のインストールが必須であるかどうかも示します。

Microsoft Word のすべてのバージョンおよび Microsoft PowerPoint のバージョン 97 から 2003 までがサポートされています。また、Microsoft Excel のバージョン 97 から 2007 までのすべてのバージョンがサポートされています。

形式		**CFML タグ**	**OpenOffice インストール**
変換元	**変換後**		
PPT	Connect プレゼンテーション	cfpresentation	オプション
PPT	HTML	cfpresentation	オプション
PPT	PDF	cfdocument	オプション
HTML	PPT	cfpresentation	必須ではありません
Excel	HTML	cfspreadsheet	必須ではありません
Excel	クエリー	cfspreadsheet	必須ではありません
Excel	メモリ内変数	cfspreadsheet	必須ではありません
クエリー	Excel	cfspreadsheet	必須ではありません
メモリ内変数	Excel	cfspreadsheet	必須ではありません
Word	PDF	cfdocument	必須

SharePoint の統合

ColdFusion は、Microsoft Windows SharePoint Services 2.0 または 3.0、および Microsoft Office SharePoint Portal Server 2003 または Microsoft Office SharePoint Server 2007 と組み合わせて使用できます。ColdFusion アプリケーションは、Web サービスのアクションとして公開されている SharePoint 機能と統合できます。

ColdFusion からの SharePoint アクションのロード

cfsharepoint タグを使用すると、Web サービスを直接ロードせずに、公開されている SharePoint 機能にアクセスできます。cfsharepoint タグは、SharePoint サーバーで基本認証が使用されている場合にのみ機能します。ColdFusion はデフォルトで、SharePoint で Web サービスとして公開されている機能の一部をサポートします。ColdFusion でサポートされていない Web サービスを使用する場合は、ロードされる Web サービスの WSDL (Web Services Description Language) の URL を指定します。

サポートされている SharePoint 機能の詳細については、『CFML リファレンス』を参照してください。

cfsharepoint の使用

ColdFusion と Sharepoint を統合すると、ユーザーリスト、ビュー、グループなどの動的な管理や、イメージおよびドキュメントワークスペースの操作が可能になり、検索機能を効果的に利用できます。cfsharepoint タグを使用すると、リストの新規作成、リスト項目の取得、および SharePoint サーバー上のリスト項目の更新を行えます。

次の例は、"getpics" という画像ライブラリリストを作成する方法を示しています。

```
<cfsharepoint
    action="addlist"
    login="#login#"
    params="#{ listname ="getpics",
    description="This a picture library list",
    templateId= "109 " }#"/>
 <! --- Creates a folder within the picture library list>
< cfsharepoint
    action  ="create new folder"
    login= "#login#"
    name="collection1"
    params="#{strListName="getpics",
    strParentFolder=""}#"/>
<!--- Uploads pictures to the folder that you created --->
<cfscript>
    myimage = filereadbinary(expandpath("Bird.jpg"));
    //convert the image into byte array to pass as input for "upload" action.
</cfscript>
<cfsharepoint
    action="upload" login="#login#"
    params="#{strListName="testpics",
    strfolder="Collection1",
    bytes="#myimage#",
    filename="bird.jpg",
    fOverwriteifexist=true}#"/>

<!--- Rotates the picture downloaded from the SharePoint server.--->
<cfsharepoint
    name ="result1"
    action="download"
    login="#login#"
    params="#{strListName="getpics",
    strfolder="New Folder",
```

```
    itemFileNames=["bird.jpg"],type=1,
    fFetchoriginalIfNotAvailable=true}#"/>
<cfimage
    action="rotate"
    source="#result1.file#"
    isbase64="yes"
    angle="45"
    name="temp"
    destination="bird.jpg"
    overwrite="yes"/>
<cfscript>
baseimage = filereadbinary(expandpath("bird.jpg"));
//convert the image into byte array to pass as input for "upload" action.
</cfscript>
<!--- Uploads the rotated image back to the SharePoint server --->
<cfsharepoint
action="upload"
login="#login#"
params="#{strListName="getpics",strfolder="Collection1",
bytes="#baseimage#",filename="bird.jpg",fOverwriteifexist=true}#"/>
```

次のようなコードを使用してリスト項目を取得すると、すべての更新が正しく行われたことを確認できます。

```
<cfsharepoint
    action="getimaginglistitems"
    login="#login#" name="result"
    params="#{strListName="getpics",strFolder="#result3.title#"}#"/>
<cfloop array="#result.Library#" index="n">
<cfif n.ows_FileLeafRef  contains "temppicrotate.jpg">
SUCCESS
<cfbreak>
</cfif>
</cfloop>
```

カスタム Web パーツを使用した SharePoint から ColdFusion へのアクセス

カスタム Web パーツを使用すると SharePoint の内部から ColdFusion にアクセスできます。カスタム Web パーツは、SharePoint Services 2.0 または 3.0、および Microsoft Office SharePoint Portal Server 2003 または Microsoft Office SharePoint Server 2007 にデフォルトで付属する Page Viewer Web Part テンプレートを使用して作成できます。

1. SharePoint Server ページで [Modify Shared Page] をクリックします。
2. [Add Web Part] を選択します。
3. ポップアップメニューで [Browse] をクリックします。Web パーツのリストが表示されます。
4. [Page Viewer Web Part] を選択します。
5. [Add] をクリックします。ページビューア Web パーツがロードされます。
6. [Open the Tools Pane] リンクをクリックします。
7. [URL] テキストフィールドで、ColdFusion アプリケーションの URL を指定します。ColdFusion アプリケーションが Web パーツ内にロードされます。

注意：Macintosh プラットフォームでマルチサーバーインストールを使用している場合、"tools.jar" ファイルが "WEB-INF/cfusion/lib" に存在していると、SharePoint が正しく動作しません。この場合は、"coldfusion.jsp.JavaCompiler's UnknownCompiler: Unable to run the internal Java compiler: java.lang.NoClassDefFoundError: javax/tools/StandardJavaFileManager." というエラーメッセージが表示されます。この問題を解決するには、"tools.jar" ファイルをバックアップディレクトリにコピーし、"WEB-INF/cfusion/lib" から "tools.jar" ファイルを削除します。

Single Sign-On を使用した SharePoint から ColdFusion アプリケーションへのアクセス

SharePoint のカスタム Web パーツを使用すると、シングルサインオン (SSO) を使用して SharePoint サーバーから複数の ColdFusion アプリケーションにアクセスできます。サインイン後、複数の Web パーツから ColdFusion にアクセスすることで、ユーザーは複数のセキュアな ColdFusion アプリケーションにアクセスできます。

注意: シングルサインオン機能を使用するには、Microsoft Office SharePoint Portal Server 2003 または Microsoft Office SharePoint Server 2007 が必要です。

ColdFusion アプリケーションを SharePoint から使用するには、CFSharepoint SSO WebPart テンプレートを使用します。このテンプレートは PageViewer WebPart をカスタマイズしたものです。このテンプレートを使用すると、SSO 資格情報を ColdFusion アプリケーションに渡せるようになります。このテンプレートは、Adobe Web サイトからダウンロードするか、ColdFusion 9 の DVD からコピーできます。

次の点に注意してください。

- Web パーツはネイティブのシングルサインオンソリューションのみをサポートします。他のプラグイン方式のシングルサインオンサービスはサポートされません。
- ColdFusion アプリケーションに渡されるのはシングルサインオンの資格情報のみです。資格情報を取得してアプリケーションにログインするために必要なロジックが ColdFusion アプリケーションに含まれている必要があります。

Microsoft Office SharePoint Server 2007 用の CF9SSOWebPart.wsp Web パーツのデプロイ

Microsoft Office SharePoint Server 2007 を使用するためにシングルサインオンを設定するには、CF9SSOWebPart.wsp ファイルを SharePoint サーバーにデプロイします。

1. Web Server Extensions フォルダ内の BIN フォルダに CF9SSOWebPart.wsp ファイルをコピーします。通常、このフォルダは、SharePoint サーバーの "Program Files¥Common Files¥Microsoft Shared¥Web Server Extensions¥12¥BIN" にあります。
2. SharePoint にソリューションをデプロイするには、コマンドプロンプトを使用して "Program Files¥Common Files¥Microsoft Shared¥Web Server Extensions¥12¥BIN" に移動し、必要に応じて次のコマンドを入力します。

   既存のソリューションを削除するには :

   ```
   STSADM.EXE -o deletesolution -name CF9SSOWebPart.wsp -override
   ```

   SharePoint にソリューションを追加するには :

   ```
   STSADM.EXE -o addsolution -f CF9SSOWebPart.wsp
   ```

   URL を指定して設定済みの Web サイトにソリューションをデプロイするには :

   ```
   STSADM.EXE -o deploysolution -name CF9SSOWebPart.wsp
   -url <virtual server url> -local -allowGacDeployment
   ```

   設定済みのすべての Web サイトにソリューションをデプロイするには :

   ```
   STSADM.EXE -o deploysolution -name CF9SSOWebPart.wsp -allcontenturls -local -allowGacDeployment
   ```

[Web Part] ページへの CF9SSOWebPart.wsp Web パーツのインポート

1. Web パーツへのアクセスを可能にする必要がある SharePoint サーバー上の Web ページに移動します。
2. [Web Part] ページで [Site Actions]-[Site Settings] をクリックします。
3. [Site Settings] ページで [Galleries]-[Web Parts] をクリックします。

4 [Web Part gallery] のツールバーペインで [Upload] をクリックします。

5 CF9SSOWebPart.wsp Web パーツを選択します。

6 ツールバーペインに次の詳細を入力します。

- アクセスする ColdFusion アプリケーションの URL
- ユーザー ID として使用するフォームフィールド名
- パスワードとして使用するフォームフィールド名
- 資格情報が設定されている SSO アプリケーションの名前

デプロイされた後の Web パーツは、([Tools] ペインで入力されたアプリケーション名に基づいて) SharePoint Single Sign-On データベースから資格情報を取得します。これらの資格情報は、指定されたフォームフィールドを含むフォームを使用して ([Tools] ペインで指定された) URL を介して ColdFusion アプリケーションに転送されます。

Microsoft Office SharePoint Portal Server 2003 用の CF9SharepointSSOCab.CAB Web パーツのデプロイ

Microsoft Office SharePoint Portal Server 2003 を使用するためにシングルサインオンを設定するには、CF9SharepointSSOCab.CAB ファイルをデプロイします。

1 Web Server Extensions フォルダ内の BIN フォルダに "CF9SharepointSSOCab.CAB" をコピーします。通常、このフォルダは、SharePoint サーバーの "Program Files¥Common Files¥Microsoft Shared¥Web Server Extensions¥60¥BIN" にあります。

2 Layouts フォルダ内に CFSharePointSSO という名前のフォルダを作成します。通常、Layouts フォルダは、"Program Files¥Common Files¥Microsoft Shared¥Web Server Extensions¥60¥Template¥Layouts" にあります。

3 "CF9SharepointSSOCab.CAB" ファイルから、前の手順で作成した CFSharePointSSO フォルダに、テンプレートファイル "CFSSO.aspx" をコピーします。

4 コマンドプロンプトで "Program Files¥Common Files¥Microsoft Shared¥Web Server Extensions¥60¥bin" に移動し、次のコマンドを入力して CAB ファイルを追加します。

```
stsadm.exe -o addwppack -filename CF9SharepointSSOCab.CAB -globalinstall
```

CAB ファイルが既に存在する場合は、次のコマンドを入力して、既存の CAB ファイルを削除してから、CAB ファイルを追加します。

```
stsadm.EXE -o deletewppack -name CF9SharepointSSOCab.CAB

stsadm.exe -o addwppack -filename CF9SharepointSSOCab.CAB -globalinstall
```

CFSharepointSSO Web パーツの設定

1 [Site Settings] ページで [Manage Security] に移動し、[Additional Settings]-[Manage Web Part Gallery] に移動します。

2 [Web Part Gallery] のツールバーで [New] をクリックします。[New Web Parts List] が表示されます。

3 CFSSOwebpart.dwp Web パーツを選択して、[Populate Gallery] をクリックします。

4 [Virtual Server Gallery] に CFSharepointSSO Web パーツを追加します。[Edit Page]-[Modify Shared Page]-[Add Web Parts]-[Browse]-[Virtual Server Gallery] をクリックします。[Web Parts] リストに Web パーツを追加します。

5 CFSharepointSSO Web パーツを追加したら、[Tools] ペインをクリックして次の詳細を入力します。

- アクセスする ColdFusion アプリケーションの URL

- ユーザー ID として使用するフォームフィールド名
- パスワードとして使用するフォームフィールド名
- 資格情報が設定されている SSO アプリケーションの名前

デプロイされた後の Web パーツは、([Tools] ペインで入力されたアプリケーション名に基づいて) SharePoint Single Sign-On データベースから資格情報を取得します。これらの資格情報は、指定されたフォームフィールドを含むフォームを使用して ([Tools] ペインで指定された) URL を介して ColdFusion アプリケーションに転送されます。

第 13 章：ColdFusion ポートレット

Adobe ColdFusion コンポーネント (CFC) を使用して、独自のポートレットを構築できるようになりました。ColdFusion を使用して作成した独自のポートレットは、次の環境で実行できます。

- JBoss Portal Server
- WebSphere Portal Server 6.1

JBoss Portal Server での ColdFusion ポートレットの実行

ローカルホストであるかリモートホストであるかに関係なく、JBoss Portal Server 上の ColdFusion ポートレットにアクセスして実行できます。

- ローカルホスト : ポータルは、JBoss Portal Server が存在しているのと同じコンピュータ上にあるポートレットにアクセスできます。
- リモートホスト : ポータルは、リモートの ColdFusion サーバーインスタンスにデプロイされたポートレットにアクセスできます。

前提条件

ColdFusion ポートレットの開発を開始する前に、次の作業を行う必要があります。

- JDK 1.5.x をインストールします。
- JSR-168 の場合は、バンドルされているバージョンの JBoss 2.6.7 または 2.6.8 Portal Server と JBoss 4.2.3 Application Server をインストールします。JSR-286 の場合は、バンドルされているバージョンの JBoss 2.7.2 Portal Server と JBoss 4.2.3 Application Server をインストールします。
- JBoss Application Server に ColdFusion をデプロイします。
- ColdFusion Administrator で J2EE セッションを有効にします。

ローカルサーバー用のポートレットの構築

ColdFusion ポートレットを記述するには :

1 CFIDE.portlets.ColdFusionPortlet パッケージを拡張する CFC を作成します。

たとえば、次の HelloPortlet.cfc では、このパッケージを拡張し、doView() および doHelp() メソッドを定義しています。

```
<cfcomponent extends="CFIDE.portlets.ColdFusionPortlet">

    <cffunction name="doView" returntype="void" output="true">
        <cfargument name="renderRequest" type="any" required="true" hint="A
        javax.portlet.RenderRequest java object">
        <cfargument name="renderResponse" type="any" required="true" hint="A
        javax.portlet.RenderResponse java object">
        <cfoutput>
            Hello World ColdFusion Portlet
        </cfoutput>
    </cffunction>
    <cffunction name="doHelp" returntype="void" output="true">
        <cfargument name="renderRequest" type="any" required="true" hint="A
        javax.portlet.RenderRequest java object">
        <cfargument name="renderResponse" type="any" required="true" hint="A
        javax.portlet.RenderResponse java object">
        <h1>ColdFusion Help</h1>
        <p>This is a Help message for the Hello Portlet.</p>
    </cffunction>
</cfcomponent>
```

2 HelloPortlet.cfc を <JBoss サーバーのホームディレクトリ >¥server¥default¥deploy¥cfusion.ear¥cfusion.war¥portlets¥hello に保存します。

3 ColdFusion の Web ルートの WEB-INF フォルダにある portlet.xml で HelloPortlet.cfc を定義します。portlet.xml ファイルの内容は次のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd" version="1.0">
    <portlet>
        <description xml:lang="en">This Portlet is a Hello World CF Portlet</description>
        <portlet-name>ColdFusionPortlet</portlet-name>
        <display-name xml:lang="en">CF Hello Portlet</display-name>
        <portlet-class>coldfusion.portlet.ColdFusionPortlet</portlet-class>
        <init-param>
                <name>cfcName</name>
                <value>portlets.hello.HelloPortlet</value>
        </init-param>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
            <portlet-mode>HELP</portlet-mode>
        </supports>
        <portlet-info>
            <title>ColdFusion Hello World Portlet</title>
        </portlet-info>
    </portlet>
</portlet-app>
```

これでポートレットが定義されてポートレット定義として登録され、cfcName が INIT パラメータとして定義されます。INIT パラメータの値は ColdFusion の Web ルートから指定する必要があります。

4 次のコマンドのいずれかを実行して JBoss サーバーを起動します。

UNIX の場合

<JBoss のホームディレクトリ >/bin/run.sh

Windows の場合

<JBoss のホームディレクトリ >¥bin¥run.bat

デフォルトでは、JBoss はローカルホストのみにバインドされます。JBoss を任意の IP アドレスにバインドするには、bin/run.sh -b 0.0.0.0（UNIX の場合)、または bin¥run.bat -b 0.0.0.0 (Windows の場合) を実行します。

5 JBoss Portal Server を起動します。デフォルトで、JBoss はポート 8080 にバインドされるため、サーバーを起動するには、http://< 一致する IP アドレス >:< ポート >/portal/ という URL を使用します。

例 : http://127.0.0.1:8080/portal

6 右上隅にあるログインリンクをクリックして、ポータルにログインします。デフォルトのユーザー資格情報は admin / admin です。

7 右上隅にある [管理] オプションをクリックします。

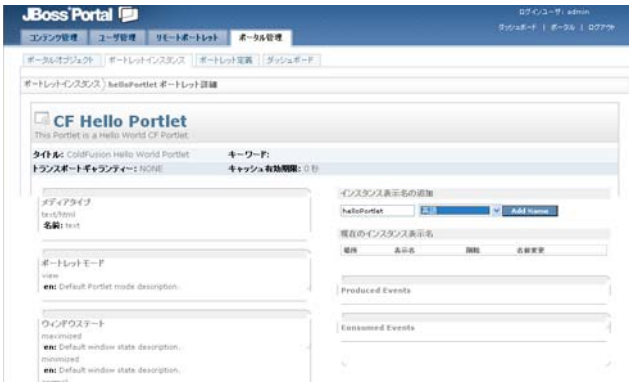
8 [ポートレット定義] タブをクリックします。ここに、"CF HelloPortlet" というポートレット名が表示されます。

CFHelloPortlet が表示されている [ポートレット定義] タブ

9 [アクション] の下にある [インスタンスの作成] をクリックして、このポートレットのインスタンスを作成します。

10 インスタンス名を指定します。

11 インスタンスの表示名を追加します。



CFHelloPortlet のインスタンスの追加

12 [ポータルオブジェクト] タブをクリックします。

13 [新規ポータル作成 名前] ボックスでポータル名を指定して、新しいポータルページを作成します。

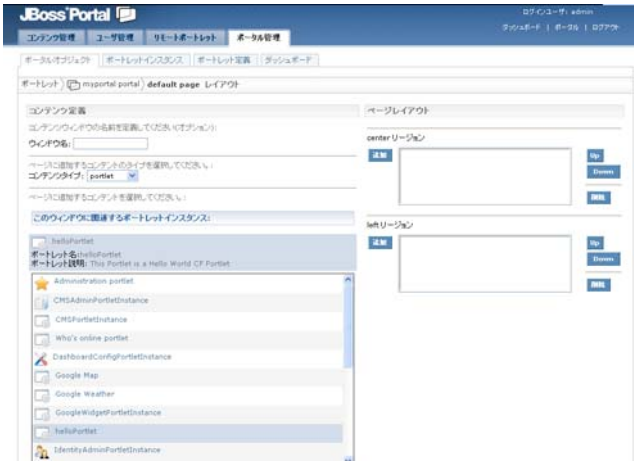
ポータルの作成

14 ポータルのリストから、先ほど作成したポータルページを選択します。

15 [アクション] で [ページレイアウト] を選択します。

作成したポータルの管理

16 [このウィンドウに関連するポートレットインスタンス] リストから、手順 10 で作成したポートレットインスタンスを選択します。

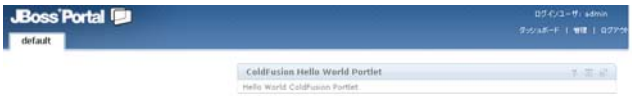


選択されたポータルインスタンス

17 [ページレイアウト] セクションの [center リージョン] または [left リージョン] で [追加] をクリックして、適切な領域にポートレットコンテナを追加します。

18 [ポートレットオブジェクト]-[ポータル] ページに戻り、[デフォルトの作成] オプションを選択して新しいポータルをデフォルトとして設定します。

19 右上隅にある [ポータル] オプションをクリックして、作成したポートレットを含む新しいポータルページを表示します。



CFHelloPortlet が表示されているポータル

リモート ColdFusion ポートレット (WSRP) へのアクセス

リモート ColdFusion ポートレットにアクセスし、Web Service Response Protocol (WSRP) を使用してそれらのポートレットを Web サービスとして公開するには、次の設定を行います。

- WSRP プロデューサ : Web Service Response Protocol (WSRP) を使用して WSDL のプロデューサを作成します。WSRP プロデューサとして使用する ColdFusion インスタンスは、スタンドアローン構成でも、マルチサーバー構成でも、J2EE インスタンスでもかまいません。
- WSRP コンシューマ : ポートレットは Web サービスとして公開され、その後ポータルサーバー (JBoss) により使用されます。

WSRP プロデューサの設定

1 879 ページの「ローカルサーバー用のポートレットの構築」の手順 1 に従って、ColdFusion HelloPortlet.cfc を作成します。

2 HelloPortlet.cfc ファイルを次のディレクトリに保存します。

```
<ColdFusion_webroot>/portlets/hello/
```

3 cf-wsrp-portlet.xml ファイル内で HelloPortlet.cfc を定義します。

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
version="1.0">
    <portlet>
        <description xml:lang="en">This Portlet is a Hello World CF Portlet</description>
        <portlet-name>HelloPortlet</portlet-name>
        <display-name xml:lang="en">Hello Portlet</display-name>
        <portlet-class>portlets.hello.HelloPortlet</portlet-class>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
            <portlet-mode>HELP</portlet-mode>
        </supports>
        <supported-locale>en</supported-locale>
        <portlet-info>
            <title>Hello ColdFusion Portlet</title>
            <short-title>CF Hello</short-title>
            <keywords>hello, portlet</keywords>
        </portlet-info>
    </portlet>
</portlet-app>
```

cf-wsrp-portlet.xml を更新した後、ColdFusion インスタンスを再起動します。

WSRP コンシューマの設定

1 次のコマンドを実行して JBoss サーバーを起動します。

UNIX の場合

bin/run.sh

Windows の場合

bin¥run.bat

デフォルトでは、JBoss はローカルホストのみにバインドされます。JBoss を任意の IP アドレスにバインドするには、bin/run.sh -b 0.0.0.0 (UNIX の場合)、または bin¥run.bat -b 0.0.0.0 (Windows の場合) を実行します。

2 JBoss Portal Server を起動します。デフォルトで、JBoss はポート 8080 にバインドされるため、サーバーを起動するには http://localhost:8080/portal/ という URL を使用します。

3 右上隅にあるログインリンクをクリックして、ポータルにログインします。デフォルトのユーザー資格情報は admin / admin です。

4 右上隅にある [管理] オプションをクリックします。

5 [WSRP] タブをクリックします。

6 [Create a Consumer Named] ボックスで、WSRP コンシューマの名前 (**wsrp-test** など) を指定します。

7 [Create Consumer] をクリックします。

8 次のページで、WSRP プロデューサの WSDL URL を次のように指定します。

http://<WSRP プロデューサの IP>:< ポート >/< コンテキストルート >/WSRPProducer?wsdl

9 [Refresh & Save] をクリックします。成功した場合、次のメッセージが表示されます。

[Refresh was successful.]

注意：WSDL URL として http://wsrp.bea.com:7001/producer/producer?WSDL と入力すると、BEA でホストされているデモ用の WSRP コンシューマを試用できます。[Refresh & Save] をクリックすると、consumerRole 登録プロパティを指定するように要求されます。"public" という文字列を入力して、[Update Properties] をクリックします。BEA WSRP プロデューサが設定されない場合、その原因は WSRP プロデューサではなく、JBoss Portal Server に関係している可能性があります。Java 1.5 JVM を使用していることを確認してください。

10 [Portlet Definitions] タブをクリックします。

11 [View portlets provided by the portlet provider named] ドロップダウンリストから **wsrp-test** を選択します。

12 [Select the WSRP Portlet You Created] ボックスから WSRP ポートレットを選択します。

13 [View Portlets] をクリックします。

14 [Actions] 列から [Create Instance] を選択します。

15 インスタンス名を指定し、[Create instance] をクリックします。

16 必要に応じて、インスタンスの表示名やその他の詳細を指定します。

17 [Portal Objects] に戻り、デフォルト設定を選択します。または、独自のポータルオブジェクトを作成することもできます。

18 [Actions] 列から [Page Layout] を選択します。

19 [Portal Instance Associated to this Window] リストから、先ほど作成した WSRP ポートレットインスタンスを選択します。

20 [Page Layout] セクションで [Add] をクリックし、必要な領域にインスタンスを追加します。

21 [Portal Objects]-[Portal] ページに戻ります。

22 [Make Default] をクリックして、WSRP ポートレットを含むポータルをデフォルトに設定します。

23 右上隅にある [Portal] リンクをクリックしてポータルページを表示します。

portlet.cfc で使用する一般的なメソッド

"HelloPortlet.cfc" のような ColdFusion コンポーネントポートレットを作成する場合に頻繁に使用する代表的メソッドには次のものがあります。

メソッド	説明	シンタックス
doView()	このメソッドはポートレットのコンテンツをレンダリングします。これは、ポートレットの現在の状態に基づいてレスポンスのコンテンツを生成するためにポートレットコンテナによって呼び出されます。	<cffunction name="doView" returntype="void" output="true"> <cfargument name="renderRequest" type="any" required="true" hint="A javax.portlet.RenderRequest java object"> <cfargument name="renderResponse" type="any" required="true" hint="A javax.portlet.RenderResponse java object"> <!--- ここにユーザーコードを記述 --?> </cffunction>
doHelp()	HELP モードを開始するヘルパメソッドです。	<cffunction name="doHelp" returntype="void" output="true"> <cfargument name="renderRequest" type="any" required="true" hint="A javax.portlet.RenderRequest java object"> <cfargument name="renderResponse" type="any" required="true" hint="A javax.portlet.RenderResponse java object"> <!--- ここにユーザーコードを記述 --? </cffunction>
doEdit()	EDIT モードを開始するヘルパメソッドです。	<cffunction name="doEdit" returntype="void" output="true"> <cfargument name="renderRequest" type="any" required="true" hint="A javax.portlet.RenderRequest java object"><cfargument name="renderResponse" type="any" required="true" hint="A javax.portlet.RenderResponse java object"> <!--- ここにユーザーコードを記述 --? </cffunction>
ProcessAction()	ポートレットでアクションリクエストを処理するために、ポートレットコンテナによって呼び出されます。	<cffunction name="processAction" returntype="void" access="public" output="false" hint="Called by the portlet container to allow the portlet to process an action request."> <cfargument name="actionRequest" type="any" required="true" hint="A javax.portlet.ActionRequest java object"> <cfargument name="actionResponse" type="any" required="true" hint="A javax.portlet.ActionResponse java object"> <!--- ここにユーザーコードを記述 --? </cffunction>
Init()	ポートレットに対してサービス状態に移行するように指示するために、ポートレットコンテナによって呼び出されます。	<cffunction name="init" returntype="void" access="public" output="false" hint="Called by the portlet container to indicate to a portlet that the portlet is being placed into service."> <cfargument name="portletConfig" type="any" required="true" hint="A javax.portlet.PortletConfig java object"> <!--- ここにユーザーコードを記述 --? </cffunction>

processEvent	このメソッドは、公開後のイベントの処理に使用します。	`<cffunction name="processEvent" returntype="void" access="public" output="false" hint="Called by the portlet container requesting the portlet to process a specific event.">`
		`<!---user code--->`
		`</cffunction>`

ColdFusion ポートレットコンポーネント

ColdFusion ポートレットコンポーネントを設定するときには、コンポーネントのモード、ウィンドウの状態、タイトル、スコープ、パラメータを定義できます。

ColdFusion ポートレット API の詳細については、JSR-168 の仕様を参照してください。ここには、すべての javax.portlet.* クラスの説明が掲載されています。

現在のところ、ポートレットでサポートされている標準は WSRP 1.0 です。

ポートレットのモード

通常、ポータルサーバーでは、[View]、[Edit]、[Help] の 3 つのポートレットモードを使用できます。

[View] モードは、ポートレットをレンダリングしたときのデフォルトの状態です。ポートレットウィンドウのタイトルバーには、[Help] モードまたは [Edit] モードに変更するためのリンクがあります。

[Help] モードビューを追加するには、doView() 関数と同じシグネチャを使用して doHelp() を追加します。

[Edit] モードをサポートするには、doEdit() を作成します。

関連項目

885 ページの「portlet.cfc で使用する一般的なメソッド」

ポートレットのウィンドウの状態

ほとんどのポータルサーバーは、3 つのウィンドウ状態 (通常、最小化、および最大化) をサポートします。現在のウィンドウ状態を取得するには、ColdFusionPortlet 基本コンポーネントの getWindowState() メソッドを呼び出します。

ポートレットのタイトル

ポートレットのタイトルを設定するには、次のように、getTitle というメソッドを CFC に追加します。

```
<cffunction name="getTitle" returntype="string" output="false" access="public">
        <cfargument name="renderRequest" type="any" required="true" hint="A
        javax.portlet.RenderRequest java object">
        <cfreturn "My ColdFusion Portlet">
</cffunction>
```

ポートレットのスコープ

ColdFusion ポートレットツールキットでは、変数 request.portlet が定義されています。これには次の構造体が含まれています。

```
request.portlet.parameters - Parameters of the Portlet Request
request.portlet.attributes - attributes of the Portlet Request
request.portlet.properties - properties of the Portlet Request
```

これらの変数は便宜上定義されています。

ポートレットのパラメータの作成

ポートレット内に複数のページビューを作成するには、renderURL などのレンダリングパラメータを設定します。

たとえば、renderURL パラメータを設定するには、次のようにします。

1 renderURL パラメータを作成します。

```
<cfset params = StructNew()>
<cfset params.page = "somepage">
<cfoutput><a href="#createRenderURL(params)#">Link to somepage</a>
```

2 ページ内のパラメータを確認し、条件に従ってレンダリングします。

```
<cfparam name="request.portlet.parameters.page" default="">
<cfif request.portlet.parameters.page IS "somepage">
    <cfinclude template="somepage.cfm">
<cfelse>
    <!--- put step 1 here --->
</cfif>
```

フォーム送信を使用したアクションの処理

フォーム送信を処理するには、フォームアクション URL を生成する createActionURL() 関数を使用します。

次に例を示します。

```
<cfoutput>
    <form action="#createActionURL()#" method="post">
        Value:  <input type="text" name="action_value" >
                <input type="submit" value="Process Action" />
    </form>
</cfoutput>
```

フォームが送信されると、CFC 内の processAction() メソッドがポータルコンテナにより呼び出されます。したがって、このメソッドを次のように追加します。

```
<cffunction name="processAction" returntype="void" access="public" output="false" hint="Called by the
portlet container to allow the portlet to process an action request.">
    <cfargument name="actionRequest" type="any" required="true" hint="A
    javax.portlet.ActionRequest java object">
    <cfargument name="actionResponse" type="any" required="true" hint="A
    javax.portlet.ActionResponse java object">
    <cfif IsDefined("request.portlet.parameters.action_value")>
    <!--- do something with this value, such as update your database --->
    </cfif>
</cffunction>
```

例

次の例はポートレットの設定方法を示しています。設定するポートレットがローカルサーバーにあるかリモートサーバーにあるかによって、doView() メソッドに次のコードを追加できます。

- ポータルのユーザー情報を取得するには :

  JSR :

  ```
  <cfoutput>#renderRequest.getRemoteUser()#</cfoutput>
  ```

  WSRP :

  ```
  <cfdump var = #renderRequest.getAttribute("javax.portlet.userinfo")#>
  ```

- PDF を表示するには :

  ```
  <cfdocument format="pdf" src="http://www.google.com" filename="cfdoc1.pd f " overwrite="true">
  </cfdocument>
  <cfset pdfURL = getPortletResponse().encodeURL(getPortletRequest().getContextPath() & "/<path of
  pdf>/cfdoc1.pdf")>
  <cfoutput>
      <object data="#pdfURL#" type="application/pdf" width="600" height="400">
      </object>
  </cfoutput>
  ```

- Ajax コンポーネントを表示するには、ポートレットで使用されているすべての URL がエンコードされている必要があります。

```
CFPOD:
<cfset sourceURL = getPortletResponse().encodeURL(getPortletRequest().getContextPath() & "/<path to
cfm>/expandpath.cfm")>
<cfpod name="pod01" source="#sourceURL#" height="500" width="300" title="Example CFPod"/>
expandpath:
<cfoutput>#ExpandPath("./")#</cfoutput>
CFWINDOW:
<cfset sourceURL = getPortletResponse().encodeURL(getPortletRequest().getContextPath() & "/<path to
cfm>/expandpath.cfm")>
<cfwindow title="Test Window" name="myWindow" width="200" height="200" initShow="true"
source="#sourceURL#">
</cfwindow>
```

JSR-286 のサポート

ColdFusion 9 では、JSR-286 仕様もサポートされています。ポートレットには、アクション、イベント、レンダリングという 3 種類のリクエストがあります。ポートレットでは、アクションリクエストが処理された後に、イベントリクエストが処理され、レンダリングリクエストは最後に処理されます。

JSR-286 では、次のような機能は使用できます。

イベントの公開と処理

イベントを定義するには、"portlet.xml" でイベントを宣言する必要があります。

```
<event-definition>
<qname xmlns:cf="http://adobe.com/coldfusion/portlet/example">cf:HelloEvent
</qname>
    <value-type>java.lang.String</value-type>
</event-definition>
```

このコードは、cf:HelloEvent というイベントを定義しています。cf は名前空間を参照しており、HelloEvent はローカル名です。型は <value-type> タグで定義されています。これらのイベント定義では、qname を使用してイベントを一意に識別する必要があります。

ここで、イベントが公開 (生成) または処理 (利用) する特定のポートレットに、イベントを追加します。この情報も "portlet.xml" に追加します。

<supported-publishing-event> タグは、イベントを公開する場合に使用します。

イベントの公開 (イベントプロデューサ)

```
<portlet>
…
<supported-publishing-event>
<qname xmlns:cf="http://adobe.com/coldfusion/portlet/example">cf:HelloEvent</qname>
</supported-publishing-event>
…
</portlet>
```

イベントの処理 (イベントコンシューマ)

```
<portlet>
….
<supported-processing-event>
<qname xmlns:cf="http://adobe.com/coldfusion/portlet/example">cf:HelloEvent</qname>
</supported-processing-event>
….. .
</portlet>
```

ポートレット定義には、公開タグと処理タグの両方を使用できます。portlet.xml ファイルには、イベント定義、イベント公開タグ、およびイベント処理タグがあり、そのポートレットによってイベントが作成または利用されます。

ポートレットでのイベントの開始 (CFC)

ポートレットコードの processAction() メソッドでイベントを公開するには、ActionResponse オブジェクトに対して setEvent() を呼び出します。この setEvent() メソッドは 2 つのパラメータを取ります。1 つはイベントオブジェクトの QName で、もう 1 つは "portlet.xml" で定義されたオブジェクトの型です。

次に processAction() メソッドの例を示します。

```
<cffunction name="processAction" returntype="void" access="public" output="false" hint="Called by the
portlet container to allow the portlet to process an action request.">
        <cfargument name="actionRequest" type="any" required="true" hint="A javax.portlet.ActionRequest
java object">
        <cfargument name="actionResponse" type="any" required="true" hint="A javax.portlet.ActionResponse
java object">
        <cfset super.processAction(arguments.actionRequest, arguments.actionResponse)>
        <!--- send event notification --->
        <cftry>
            <cfset arguments.actionResponse.setEvent("HelloEvent",
request.portlet.parameters.event_value)>
            <cfcatch type="any">
                <cflog file="simple-event-portlet" type="error" text="processAction() threw exception:
#cfcatch.message#">
            </cfcatch>
        </cftry>
</cffunction>
```

processEvent() メソッドでイベントを取得します。

```
<cffunction name="processEvent" returntype="void" access="public" output="false" hint="Called by the
portlet container requesting the portlet to process a specific event.">
        <cfargument name="eventRequest" type="any" required="true" hint="A javax.portlet.EventRequest java
object">
        <cfargument name="eventResponse" type="any" required="true" hint="A javax.portlet.EventResponse
java object">
    <cfset var e = StructNew()>
    <cftry>
        <cfset e.name = arguments.eventRequest.getEvent().getName()>
        <cfset e.value = arguments.eventRequest.getEvent().getValue()>
    <cfif NOT IsDefined("application.EventReceivingPortletEvents")>
        <cfset application.EventReceivingPortletEvents = ArrayNew(1)>
    </cfif>
        <cfset ArrayAppend(application.EventReceivingPortletEvents,e)>
            <cfcatch type="any">

            </cfcatch>
    </cftry>
</cffunction>
```

フィルタの使用

"portlet.xml" でのフィルタ定義とマッピング

```
<filter>
    <filter-name>Example ColdFusion Filter</filter-name>
    <filter-class>coldfusion.portlet.ColdFusionPortletFilter</filter-class>
    <lifecycle>RENDER_PHASE</lifecycle>
    <lifecycle>EVENT_PHASE</lifecycle>
    <lifecycle>RESOURCE_PHASE</lifecycle>
    <lifecycle>ACTION_PHASE</lifecycle>
    <init-param>
        <name>cfcName</name>
        <value>portlets.filter.ExampleFilter</value>
    </init-param>
</filter>
```

特定のポートレットに適用されるフィルタのフィルタマッピングを追加します。

```
<!-- Applies Example Filter to All Portlets -->
    <filter-mapping>
        <filter-name>Example ColdFusion Filter</filter-name>
        <portlet-name>*</portlet-name>
    </filter-mapping>
```

ExampleFilter.cfc:

"portlet.xml" で参照されている ExampleFilter.cfc は次のとおりです。

```
<cfcomponent extends="CFIDE.portlets.filter.ColdFusionPortletFilter">
        <cffunction name="doRenderFilter" returntype="void">
        <cfargument name="renderRequest">
        <cfargument name="renderResponse">
        <cfargument name="filterChain">
        <cflog file="portlet-filter" type="information" text="doRenderFilter() invoked">
        <!--- call the next filter in the chain --->
        <cfset arguments.filterChain.doFilter(arguments.renderRequest, arguments.renderResponse)>
    </cffunction>
    <cffunction name="doActionFilter" returntype="void">
        <cfargument name="actionRequest">
        <cfargument name="actionResponse">
        <cfargument name="filterChain">
        <cflog file="portlet-filter" type="information" text="doActionFilter() invoked">
        <!--- call the next filter in the chain --->
        <cfset arguments.filterChain.doFilter(arguments.actionRequest, arguments.actionResponse)>
    </cffunction>
    <cffunction name="doResourceFilter" returntype="void">
        <cfargument name="resourceRequest">
        <cfargument name="resourceResponse">
        <cfargument name="filterChain">
        <cflog file="portlet-filter" type="information" text="doResourceFilter() invoked">
        <!--- call the next filter in the chain --->
        <cfset arguments.filterChain.doFilter(arguments.resourceRequest, arguments.resourceResponse)>
    </cffunction>
    <cffunction name="doEventFilter" returntype="void">
        <cfargument name="eventRequest">
        <cfargument name="eventResponse">
        <cfargument name="filterChain">
        <cflog file="portlet-filter" type="information" text="doEventFilter() invoked">
        <!--- call the next filter in the chain --->
        <cfset arguments.filterChain.doFilter(arguments.eventRequest, arguments.eventResponse)>
    </cffunction>
</cfcomponent>
```

WebSphere Portal Server での ColdFusion ポートレットの実行

WebSphere Portal Server 6.1 上の ColdFusion ポートレットにアクセスして実行するには :

1. "cfusion.war" ファイルを作成します。
2. jar -xvf cfusion.war を使用して、このファイルを解凍します。
3. "cfusion.war" の下に "/portlets" ディレクトリを作成します。
4. "/portlets" ディレクトリにポートレットを追加します。次の場所にある "portlet.xml" にポートレットエントリを追加します。

   cfusion.war/WEB-INF/portlet.xml

5. jar cvf cfusion.war を使用して、WAR ファイルを再びパッケージ化します。
6. WebSphere Portal Server の管理コンソールを使用して、このファイルをデプロイします。"portlet.xml" に登録されたポートレットが表示可能な状態になります。
7. ポータルページを作成し、これらのポートレットを追加します。

第 14 章：ドキュメント、チャート、レポートの操作

ColdFusion での PDF フォームの操作

Adobe ColdFusion では、Adobe® Acrobat® Professional および Adobe® LiveCycle™ Designer で作成した PDF フォームを操作できます。

PDF フォームについて

Adobe ColdFusion では、インタラクティブ PDF フォームをアプリケーションに組み込むことができます。PDF フォームから送信されたデータを抽出したり、XML データファイルやデータベースの値をフォームフィールドに設定したり、ColdFusion で作成する PDF ドキュメントに PDF フォームを埋め込むことができます。

ColdFusion では、Adobe Acrobat および LiveCycle で作成されたインタラクティブフォームがサポートされています。Adobe Acrobat 6.0 以前では、インタラクティブ Acroform を作成できます。Adobe Acrobat Professional 7.0 以降で提供されている Adobe LiveCycle Designer では、インタラクティブフォームを生成できます。

フォームの種類が異なると、ColdFusion でのデータの操作方法も異なります。たとえば、Acrobat で作成したフォームから XML データファイルを生成して、LiveCycle で作成したフォームにそのデータを設定したり、その逆を行ったりすることはできません。この 2 種類のフォームでは、XML ファイルの形式が異なるからです。

Acrobat で作成したフォームでは、XFDF (XML Forms Data Format) というファイル形式が使用されます。LiveCycle で作成したフォームでは、Acrobat および Adobe Reader 6 で導入された XFA (XML Forms Architecture) という形式が使用されます。例については、893 ページの「PDF フォームへの XML データの設定」を参照してください。ファイル形式は、フォームのフィールドにあらかじめデータソースの値を設定する方法にも影響を与えます。これは、データ構造体やフィールド名をマッピングするからです。例については、894 ページの「PDF フォームフィールドの事前設定」を参照してください。

JavaScript の使用方法も状況によって異なります。PDF ファイルの JavaScript オブジェクトモデルは、HTML の JavaScript オブジェクトモデルとは異なります。したがって、HTML JavaScript で記述しているスクリプトを PDF ファイルに適用することはできません。また、JavaScript は、Acrobat で作成したフォームと LiveCycle で作成したフォームでも異なるので、一方の形式で記述したスクリプトは、もう一方では機能しません。

ColdFusion 9 には、PDF フォームを操作する次のタグが用意されています。

タグ	説明
cfpdfform	フォームのデータを読み込んでファイルに書き込んだり、データソースのデータをフォームに設定します。
cfpdfformparam	cfpdfform タグまたは cfpdfsubform タグの子タグで、PDF フォームの個々のフィールドに値を設定します。
cfpdfsubform	cfpdfform タグの子タグで、フォームフィールドに正しく値が設定されるように、PDF フォームの階層を作成します。cfpdfsubform タグには、1 つまたは複数の cfpdpformparam タグを記述します。

次の表に、PDF フォームに対して行えるタスクの一部を示します。

タスク	タグおよびアクション
PDF フォームに XML データを設定する	cfpdf タグの populate アクション
PDF フォームの個々のフィールドに、あらかじめデータソースのデータを設定する	cfpdfform タグの populate アクションと、cfpdfsubform および cfpdfparam タグ
PDF フォームの構造を確認する	cfpdfform タグの read アクションと、cfdump タグ
PDF ドキュメントにインタラクティブ PDF フォームを埋め込む	cfdocument タグ内での cfpdfform タグの populate アクション メモ : cfpdfform タグは、cfdocumentsection タグの中ではなく、同じレベルに記述する必要があります。
PDF フォームをブラウザに直接書き込む	cfpdfform タグの populate アクションで destination 属性を指定しない
PDF フォームの出力を XML ファイルに書き込む	cfpdfform タグの read アクション
PDF フォームを ColdFusion から印刷する	cfprint タグ
PDF フォーム送信からデータを抽出する	cfpdfform タグの read アクションで source="#PDF.Content#"
PDF フォーム送信から抽出したデータを PDF ファイルに書き込む	cfpdfform タグの populate アクションで destination 属性を指定して source="#PDF.Content#"
LiveCycle で生成したフォームのデータを XDP ファイルに書き込む	cfpdfform タグの populate アクションで出力ファイルに XDP の拡張子を指定して source="#PDF.Content#"
HTTP Post 送信からデータを抽出する	cfdump タグを使用してフォームデータの構造体を確認し、フォームフィールドと出力フィールドをマッピングする
Acrobat で生成したフォームをフラット化する (LiveCycle で生成したフォームには使用できません)	cfpdf 詳細については、917 ページの「Acrobat で作成したフォームのフラット化」を参照してください。
Acrobat または LiveCycle で生成したフォームと他の PDF ドキュメントを結合する	cfpdf action="merge" 詳細については、916 ページの「PDF ドキュメントの結合」を参照してください。

PDF フォームへの XML データの設定

アプリケーションによっては、PDF フォームのデータを XML データファイルで送信する場合があります。たとえば、LiveCycle で作成したフォームを電子メールで送信する場合は、XML データファイルを生成して、指定の電子メールアドレスに添付ファイルとして送信することができます。XML データファイルは PDF ファイルよりもサイズが小さいので、データの送信やアーカイブを効率的に行えます。ただし、XML ファイルは人間にとって読みやすい形式ではありません。ファイルを元の形式で表示するには、PDF フォームのテンプレートを Acrobat で開いて、XML データファイルをインポートする必要があります。

ColdFusion では、XML データと、そのデータを生成した PDF フォームを再結合する処理を自動化することができます。再結合するには、cfpdfform タグの populate アクションを使用して、ソース (テンプレートとして使用する PDF フォーム) を指定し、ユーザーがフォームから送信した情報が格納されている XML データファイルを指定します。結果を新しいファイルに保存するオプションを使用すれば、フォームデータだけでなく、入力後のフォームを元の形式のまま保存できます。次の例では、"payslipTemplate.pdf" フォームに "formdata.xml" データファイルのデータを設定し、このフォームを "employeeid123.pdf" という新しい PDF ファイルに書き込みます。

```
<cfpdfform source="c:\payslipTemplate.pdf" destination="c:\empPayslips\employeeid123.pdf"
action="populate" XMLdata="c:\formdata.xml"/>
```

LiveCycle で作成したフォームの場合は、PDF ファイルではなく、XDP (XML Data Package) ファイルに出力を書き込むこともできます。詳細については、899 ページの「XDP ファイルへの LiveCycle フォームの出力の書き込み」を参照してください。

注意：populate アクションで宛先を指定しなかった場合は、値が設定された PDF フォームがブラウザウィンドウに表示されます。

フォームに XML データファイルの値を設定する場合は、XML データが適切な形式であることを確認してください。XML データファイルの形式は、データの生成元が Acrobat であるか LiveCycle であるかによって異なります。Acrobat では、XFDF (XML Forms Data Format) のファイル形式で生成されます。次に、XFDF 形式の例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
- <xfdf xmlns="http://ns.adobe.com/xfdf/" xml:space="preserve">
    - <fields>
        - <field name="textname">
                <value>textvalue</value>
            </field>
        - <field name="textname1">
                <value>textvalue1</value>
            </field>
        </fields>
    </xfdf>
```

LiveCycle で作成したフォームの場合は、XFA (XML Forms Architecture) 形式を使用する必要があります。次に、XFA 形式の例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
- <xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
- <form1>
        <SSN>354325426</SSN>
        <fname>coldfusion</fname>
        <num>354325426.00</num>
    - <Subform1>
            <SSN />
        </Subform1>
</form1>
</xfa>
```

PDF フォームフィールドの事前設定

ColdFusion では、個々のフォームフィールドに、データソースから抽出したデータをあらかじめ設定することができます。たとえば、ユーザー名とパスワードに基づいてデータソースをクエリーし、返された顧客情報に基づいて、注文フォームの関連フィールドに値を設定することができます。顧客は、フォームのその他のフィールドを入力して、処理のためにフォームを送信できます。この処理を行うには、PDF フォームのフィールド名やデータ構造体と、データソースのフィールドをマッピングします。

PDF フォームの構造体を確認するには、次の例のように、cfpdfform タグの read アクションを使用します。

```
<cfpdfform source="c:\forms\timesheet.pdf" result="resultStruct" action="read"/>
```

続いて、cfdump タグを使用してこの構造体を表示します。

```
<cfdump var="#resultStruct#">
```

Acrobat フォームで作成したフォームの結果の構造体は、次の例のようになります。

struct	
firstName	[空の文字列]
lastName	[空の文字列]
department	[空の文字列]
...	...

ColdFusion でフィールドにあらかじめ値を設定するには、cfpdfform タグの直後に、各フィールドの cfpdfformparam タグを追加します。

```
<cfpdfform action="populate" source="c:\forms\timsheet.PDF">
    <cfpdfformparam name="firstName" value="Boris">
    <cfpdfformparam name="lastName" value="Pasternak">
    <cfpdfformparam name="department" value="Marketing">
...
</cfpdfform>
```

LiveCycle で標準の空のフォームから作成したフォームには、form1 という名前のサブフォームが含まれています。LiveCycle で作成したフォームの結果の構造体は、次の例のようになります。

struct		
form1	struct	
	txtfirstName	[空の文字列]
	txtlastName	[空の文字列]
	txtdepartment	[空の文字列]
	...	...

ColdFusion でフィールドにあらかじめ値を設定するには、form1 の cfpdfsubform タグを追加し、このタグの直後に、値を設定する各フィールドの cfpdfformparam タグを追加します。

```
<cfpdfform source="c:\forms\timesheetForm.pdf" action="populate">
    <cfpdfsubform name="form1">
        <cfpdfformparam name="txtfirstName" value="Harley">
        <cfpdfformparam name="txtlastName" value="Davidson">
        <cfpdfformparam name="txtDeptName" value="Engineering">
        ...
    </cfpdfsubform>
</cfpdfform>
```

注意: LiveCycle フォームで作成したダイナミックフォーム (LiveCycle Designer でダイナミック PDF フォームファイルとして保存したフォーム) では、レコードが繰り返された回数を記録することができます。そのため、サブフォームのレコードが存在しない場合は、cfpdfform タグの read アクションで返される構造体にサブフォームは含まれません。階層を確認するには、LiveCycle Designer でフォームを表示します。

サブフォームのネスト

Acrobat フォームには、サブフォームは含まれませんが、複合フィールド名が含まれている場合があります。たとえば、Acrobat フォームに、form1.x.f1、form1.x.f2、form1.x.f3 といったフィールドが含まれていることがあります。

cfpdfparam タグでは、ピリオドの含まれたフィールド名は処理されないので、Acrobat で作成したフォームに複合フィールド名が含まれている場合は、LiveCycle で作成したサブフォームと同じように処理されます。複合フィールド名が含まれている Acrobat フォームの結果の構造体は、次の例のようになります。

struct	
form1	struct

struct			
	x	struct	
		f1	[空の文字列]
		f2	[空の文字列]
		...	...

ColdFusion で、Acrobat で作成したフォームのフィールドにあらかじめ値を設定するには、サブフォームと同じようにフィールド名をネストします。

```
<cfpdfform action="populate" source="acrobatForm.pdf">
    <cfpdfsubform name="form1">
        <cfpdfsubform name="x">
            <cfpdfformparam name="f1" value="AGuthrie">
            <cfpdfformparam name="f2" value="123">
            <cfpdfformparam name="f3" value="456">
        </cfpdfsubform>
    </cfpdfsubform>
</cfpdfform>
```

LiveCycle で作成するフォームでは、form1 サブフォームの中にサブフォームを含めることがよくあります。たとえば、次の補助金申請書にはネストしたサブフォームが含まれています。

struct				
form1	struct			
	grantapplication	struct		
		page1	struct	
			orgAddress	[空の文字列]
			orgCity	[空の文字列]
			orgState	[空の文字列]
			...	...
		page2	struct	
			description	[空の文字列]
			pageCount	[空の文字列]
			...	...

ColdFusion でフィールドに値を設定するには、cfpdfsubform タグをネストして、構造体をマッピングします。

```
<cfpdfform source="c:\grantForm.pdf" destination="c:\employeeid123.pdf" action="populate">
    <cfpdfsubform name="form1">
        <cfpdfsubform name="grantapplication">
            <cfpdfsubform name="page1">
                <cfpdfformparam name="orgAddress" value="572 Evergreen Terrace">
                <cfpdfformparam name="orgCity" value="Springfield">
                <cfpdfformparam name="orgState" value="Oregon">
                ...
            </cfpdfsubform>
            <cfpdfsubform name="page2">
                <cfpdfformparam name="description" value="Head Start">
                <cfpdfformparam name="pageCount" value="2">
                ...
            </cfpdfsubform>
        </cfpdfsubform>
    </cfpdfsubform>
</cfpdfform>
```

注意：PDF ファイルに含められるインタラクティブフォームは 1 つのみです。したがって、PDF ファイルにサブフォームが含まれている場合は、送信ボタンを押すと、すべてのサブフォームのデータがまとめて送信されます。

PDF ドキュメントへの PDF フォームの埋め込み

cfdocument タグ内で cfpdfform タグを使用すると、PDF ドキュメントに、既存のインタラクティブ PDF フォームを埋め込むことができます。この方法は、標準のインタラクティブフォームを使用して追加情報を含めたい場合に便利です。たとえば、企業が汎用的な PDF フォームを使用して、従業員の情報を管理しているとします。このフォームをさまざまな状況で再利用して、最新の従業員情報を維持することができます。

スタティックな PDF ページを作成するには、cfdocument タグと cfdocumentsection タグを使用します。その後に、cfdocument タグ内で cfpdfform タグを使用して、PDF ドキュメントにインタラクティブフォームを作成します。ユーザーがフォームを更新して印刷または送信すると、スタティックな PDF ページも含めて、ドキュメント内のすべてのページがフォームと一緒に印刷または送信されます。

注意：PDF ドキュメントに埋め込めるインタラクティブフォームは 1 つのみです。したがって、cfdocument タグの中に記述できる cfpdfform タグは 1 つのみです。ただし、1 つの cfpdfform タグの中には、複数の cfpdfsubform タグおよび cfpdfformparam タグを記述できます。

cfpdfform タグは、最低 1 つの cfdocumentsection タグと併用しますが、cfdocumentsection タグの中に cfpdfform タグを含めないようにしてください。cfpdfform と cfdocumentsection タグは、次の例のように、同じレベルで記述します。

```
<cfdocument format="pdf">
    <cfdocumentitem type="header">
    <font size="+1">This is the Header</font>
    </cfdocumentitem>
    <cfdocumentitem type="footer">
    <font size="+1">This is the Footer</font>
    </cfdocumentitem>

    <cfdocumentsection>
    <p>This is the first document section.</p>
    </cfdocumentsection>

    <cfpdfform source="c:\forms\embed.pdf" action="populate">
        <cfpdfsubform name="form1">
            <cfpdfformparam name="txtManagerName" value="Janis Joplin">
            <cfpdfformparam name="txtDepartment" value="Sales">
        </cfpdfsubform>
    </cfpdfform>

    <cfdocumentsection>
    <p>This is another section</p>
    </cfdocumentsection>
</cfdocument>
```

cfpdfform タグの内容は、新しいページで開始されます。cfdocument タグの直後から cfpdfform タグの前までに存在するテキストまたはコードは、ドキュメントセクションには適用されますが、cfpdfform タグ内のインタラクティブ PDF フォームには適用されません。

埋め込み PDF フォームの一部であるヘッダとフッタは、PDF ドキュメントのその他の部分には適用されず、cfdocument タグで定義されているヘッダとフッタは、インタラクティブフォームには適用されません。cfdocumentitem タグで定義されているヘッダとフッタの情報は、埋め込みフォームの後にあるセクションから再開され、埋め込みフォームページの存在が反映されます。

注意：PDF フォームを埋め込んでいる場合、cfpdfform タグの read アクションは使用できません。また、cfpdfform タグで宛先を指定できません。ただし、cfdocument タグでファイル名を指定して、PDF フォームが含まれている PDF ドキュメントを出力ファイルに書き込むことはできます。ファイル名を指定しなかった場合は、ブラウザ内に PDF ドキュメントのコンテキストで PDF フォームが表示されます。

PDF フォーム送信からのデータの抽出

データの抽出は、PDF フォームの送信方法によって異なります。ColdFusion では、2 種類の PDF フォーム送信がサポートされています。1 つは HTTP Post で、フォームそのものではなく、フォームのデータを送信します。もう 1 つは PDF で、PDF ファイル全体を送信します。

PDF 送信は、アーカイブを行う場合などに使用します。PDF 送信では、データとともにフォームが送信されるので、出力をファイルに書き込むことができます。HTTP Post 送信では、フィールドデータのみが送信されるので処理が高速です。データベースを更新する場合や、フォームから収集した特定のデータを操作する場合に便利ですが、HTTP Post 送信を直接ファイルに書き込むことはできません。

注意：LiveCycle Designer で作成したフォームでは、XDP や XML など、いくつかの送信方法を使用できますが、ColdFusion でデータを抽出できるのは、HTTP Post 送信と PDF 送信のみです。

LiveCycle Designer で、HTTP Post 送信を実行する XML コードは、次の例のようになります。

```
<submit format="formdata" target="http://localhost:8500/pdfforms/pdfreceiver.cfm" textEncoding="UTF-8"/>
```

LiveCycle Designer で、PDF 送信を実行する XML コードは、次の例のようになります。

```
<submit format="pdf" target="http://localhost:8500/pdfforms/pdfreceiver.cfm" textEncoding="UTF-16"
xdpContent="pdf datasets xfdf"/>
```

注意：Acrobat フォームは、XML 形式ではなく、バイナリ形式で送信されます。

PDF 送信からのデータの抽出

次のコードでは、PDF 送信からデータを抽出し、そのデータを fields という構造体に書き込みます。

```
<!--- The following code reads the submitted PDF file and generates a result structure called fields. --->
<cfpdfform source="#PDF.content#" action="read" result="fields"/>
```

データ構造体を表示するには、次のように cfdump タグを使用します。

```
<cfdump var="#fields#">
```

注意：PDF 送信からデータを抽出する場合は、常にソースとして "#PDF.content#" を指定します。

次の例のように、変数にフォームフィールドを設定できます。

```
<cfset empForm="#fields.form1#">
```

出力をファイルに書き込むには、cfpdfform タグの populate アクションを使用します。ソースには "#PDF.content#" を指定します。次の例では、PDF フォームのフィールドから一意のファイル名を生成しています。

```
<cfpdfform action="populate" source="#PDF.content#"
    destination="timesheets\#empForm.txtsheet#.pdf" overwrite="yes"/>
```

XDP ファイルへの LiveCycle フォームの出力の書き込み

Acrobat フォームの場合、出力は PDF ファイルにしか書き込むことはできません。LiveCycle フォームの場合は、出力を XDP ファイルに書き込むこともできます。ファイル形式は、ファイル名拡張子によって決まります。出力を XDP 形式で保存するには、単純に、次の例のように宛先のファイル名で XDP の拡張子を使用します。

```
<cfpdfform action="populate" source="#PDF.content#"
    destination="timesheets\#empForm.txtsheet#.xdp" overwrite="yes"/>
```

XDP ファイルは、PDF ファイルの XML 表現です。LiveCycle Designer の場合は、実行時に LiveCycle フォームを表示するために必要な情報 (構造体、データ、注釈など) が XDP ファイルに含まれます。

ColdFusion の XDP ファイルには、XDP の XML コードと PDF イメージが含まれます。したがって、ファイルサイズは PDF ファイルよりも大きくなります。PDF フォームを XDP ファイルに書き込むのは、LiveCycle サーバーで LiveCycle Designer ワークフローに PDF フォームを組み込む必要がある場合のみです。

XML ファイルへの PDF 出力の書き込み

ColdFusion では、PDF フォームからデータを抽出して、XML データファイルに書き込むことができます。これを行うには、フォームの出力を PDF ファイルとして保存する必要があります。これは、cfpdfform タグのソースが必ず PDF ファイルであるからです。

PDF ファイルの出力を XML ファイルに書き込むには、次の例のように、cfpdfform タグの read アクションを使用します。

```
<cfpdfform action="read" source="#empForm.txtsheet#.pdf"
    XMLdata="timesheets\#empForm.txtsheet#.xml"/>
```

ディスク領域を節約する場合は、PDF ファイルを削除して、XML データファイルを残しておくことができます。テンプレートとして使用した空の PDF フォームを残しておけば、populate アクションを使用して、PDF ファイルを再生成することができます。フォームに値を設定する方法については、893 ページの「PDF フォームへの XML データの設定」を参照してください。

HTTP Post 送信からのデータの抽出

HTTP Post 送信の場合は、次のように cfdump タグとフォーム名の変数を使用して、データ構造体を表示します。

```
<cfdump var="#FORM.form1#">
```

注意：HTTP Post 送信からデータを抽出する場合は、常にソースとしてフォーム名を指定します。たとえば、LiveCycle の標準テンプレートから生成したフォームの場合は、"#FORM.form1#" と指定します。

この構造体は、テンプレートとして使用した (送信前の) PDF ファイルの構造体と同じであるとは限りません。たとえば、送信前のフォームの構造体が、次の例のようになっていたとします。

struct		
form1	struct	
	txtDeptName	[空の文字列]
	txtEMail	[空の文字列]
	txtEmpID	[空の文字列]
	txtFirstName	[空の文字列]
	txtLastName	[空の文字列]
	txtPhoneNum	[空の文字列]

HTTP Post で送信した後の結果の構造体は、次の例のようになる場合があります。

struct				
FORM1	struct			
	SUBFORM	struct		
		HEADER	struct	
			HTTPSUBMITBUTTON1	[空の文字列]
			TXTDEPTNAME	Sales
			TXTFIRSTNAME	Carolynn
			TXTLASTNAME	Peterson
			TXTPHONENUM	(617) 872-9178
		TXTEMPID	1	
		TXTEMAIL	carolynp@company	

注意：cfpdfform タグを使用したデータの抽出が複数のページにまたがる場合は、1 つの構造体ではなく、1 ページにつき 1 つの構造体が返されます。

構造体の相違には、HTTP Post 送信に対して Acrobat で適用される内部ルールが反映されています。

HTTP Post 送信からデータを抽出してデータベースを更新するには、次のコードに示すように、データベース列とフォームフィールドをマッピングします。

```
<cfquery name="updateEmpInfo" datasource="cfdocexamples">
UPDATE EMPLOYEES
    SET FIRSTNAME = "#FORM1.SUBFORM.HEADER.TXTFIRSTNAME#",
        LASTNAME = "#FORM1.SUBFORM.HEADER.TXTLASTNAME#",
        DEPARTMENT = "#FORM1.SUBFORM.HEADER.TXTDEPTNAME#",
        IM_ID = "#FORM1.SUBFORM.TXTEMAIL#",
        PHONE = "#FORM1.SUBFORM.HEADER.TXTPHONENUM#"
    WHERE EMP_ID = <cfqueryparam value="#FORM1.SUBFORM.TXTEMPID#">
</cfquery>
```

次のコードに示すように、変数を設定して、フィールド名へのショートカットを作成することもできます。

```
<cfset fields=#form1.subform.header#>
```

cfoutput タグを使用すると、フォームデータを表示できます。

```
<h3>Employee Information</h3>
<cfoutput>
    <table>
        <tr>
            <td>Name:</td>
            <td>#fields.txtfirstname# #fields.txtlastname#</td>
        </tr>
        <tr>
            <td>Department:</td>
            <td>#fields.txtdeptname#</td>
        </tr>
        <tr>
            <td>E-Mail:</td>
            <td>#fields.txtemail#</td>
        <tr>
            <td>Phone:</td>
            <td>#fields.txtphonenum#</td>
        </tr>
    <table>
</cfoutput>
```

PDF フォームを使用したアプリケーションの例

次の例では、アプリケーションで PDF フォームを使用する方法を示します。

PDF 送信の例

次の例では、LiveCycle Designer で作成した PDF フォームのフィールドに、従業員のログイン情報に基づいて値を設定します。従業員がフォームに入力して PDF の送信ボタンをクリックすると、PDF フォーム全体とそのデータが、2 番目の処理ページに送信されます。このページによって、入力後のフォームがファイルに書き込まれます。

ColdFusion ログインページでは、従業員がユーザー名とパスワードを入力します。

```
<!--- The following code creates a simple form for entering a user name and password.
    The code does not include password verification. --->
<h3>Timesheet Login Form</h3>
<p>Please enter your user name and password.</p>
<cfform name="loginform" action="loginform_proc.cfm" method="post">
<table>
    <tr>
        <td>user name:</td>
        <td><cfinput type="text" name="username" required="yes"
                message="A user name is required."></td>
    </tr>
    <tr>
        <td>password:</td>
        <td><cfinput type="password" name="password" required="yes"
                message="A password is required."></td>
    </tr>
</table>
    <br/>
    <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

最初の処理ページでは、クエリーを使用して、このユーザー名に関するすべての情報を cfdocexamples データベースから取得します。cfpdfform タグを使用して、LiveCycle Designer で作成した対応する PDF フォーム (timesheetForm.pdf) に従業員の名前、電話番号、電子メールアドレス、部門を設定します。ブラウザには情報が設定されたフォームが表示されます。従業員は、このフォームに入力してフォームを送信できます。

```
<!--- The following code retrieves all of the employee information for the user name entered
    on the login page. --->
<cfquery name="getEmpInfo" datasource="cfdocexamples">
    SELECT * FROM EMPLOYEES
    WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<!---
The following code populates the template called "timesheetForm.pdf" with data from the query and displays
the interactive PDF form in the browser. A field in the PDF form contains the name of the output file being
written. It is a combination of the user name and the current date.
--->

<!--- Notice the use of the cfpdfsubform tag. Forms created from templates in LiveCycle Designer include a
subform called form1. Use the cfpdfsubform tag to match the structure of the form in ColdFusion. Likewise,
the field names in the cfpdfformparam tags must match the field names in the PDF form. If the form structures
and field names do not match exactly, ColdFusion does not populate the form fields. --->

<cfpdfform source="c:\forms\timesheetForm.pdf" action="populate">
    <cfpdfsubform name="form1">
        <cfpdfformparam name="txtEmpName" value="#getEmpInfo.FIRSTNAME#
            #getEmpInfo.LASTNAME#">
        <cfpdfformparam name="txtDeptName" value="#getEmpInfo.DEPARTMENT#">
        <cfpdfformparam name="txtEmail" value="#getEmpInfo.IM_ID#">
        <cfpdfformparam name="txtPhoneNum" value="#getEmpInfo.PHONE#">
        <cfpdfformparam name="txtManagerName" value="Randy Nielsen">
        <cfpdfformparam name="txtSheet"
            value="#form.username#_#DateFormat(Now())#">
    </cfpdfsubform>
</cfpdfform>
```

ユーザーがタイムシートフォームへの入力を終えて (期間、プロジェクト、週あたりの時間を入力して)、送信ボタンをクリックすると、Acrobat から 2 番目の ColdFusion 処理ページにバイナリ形式の PDF ファイルが送信されます。

注意：LiveCycle Designer では、PDF フォームで標準の送信ボタンを使用し、このボタンのオブジェクトプロパティに [submit as: PDF] を指定します。また、[Submit to URL] フィールドに、ColdFusion 処理ページの URL を指定します。

cfpdfform タグの read アクションで、PDF の内容を fields という結果の構造体に読み込みます。cfpdfform タグの populate アクションで、入力後のフォームを "timesheets" サブディレクトリ内のファイルに書き込みます。

```
<!--- The following code reads the PDF file submitted in binary format and generates a result structure
called fields. The cfpdfform populate action and the cfoutput tags reference the fields in the structure.
--->

<cfpdfform source="#PDF.content#" action="read" result="fields"/>
<cfset empForm="#fields.form1#">
<cfpdfform action="populate" source="#PDF.content#" destination="timesheets\#empForm.txtsheet#.pdf"
overwrite="yes"/>

<h3>Timesheet Completed</h3>
<p><cfoutput>#empForm.txtempname#</cfoutput>,</p>
<p>Thank you for submitting your timesheet for the week of <cfoutput>#DateFormat(empForm.dtmForPeriodFrom,
"long")#</cfoutput> through <cfoutput>#DateFormat(empForm.dtmForPeriodto, "long")#</cfoutput>. Your
manager, <cfoutput>#empForm.txtManagerName#</cfoutput>, will notify you upon approval.</p>
```

HTTP Post の例

次の例では、HTTP Post で送信された PDF フォームからデータを抽出し、このデータを使用して従業員データベースを更新します。このフォームは、LiveCycle Designer で作成されたものです。

ColdFusion ログインページでは、従業員がユーザー名とパスワードを入力します。

```
<!--- The following code creates a simple form for entering a user name and password. The code does not
include password verification. --->

<h3>Employee Update Login Form</h3>
<p>Please enter your user name and password.</p>
<cfform name="loginform" action="loginform_procHTTP.cfm" method="post">
<table>
    <tr>
        <td>user name:</td>
        <td><cfinput type="text" name="username" required="yes"
                message="A user name is required."></td>
    </tr>
    <tr>
        <td>password:</td>
        <td><cfinput type="password" name="password" required="yes"
                message="A password is required."></td>
    </tr>
</table>
    <br/>
    <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

最初の処理ページでは、クエリーを使用して、このユーザー名に関するすべての情報を cfdocexamples データベースから取得します。cfpdfform タグを使用して、LiveCycle Designer で作成した対応する PDF フォーム (employeeInfoHTTP.pdf) に従業員の名前、電話番号、電子メールアドレス、部門を設定します。また、このフォームには、従業員 ID が非表示フィールドとして含まれています。ブラウザには情報が設定されたフォームが表示されます。従業員は、このフォームの個人情報を変更して、フォームを送信できます。

```
<!--- The following code retrieves all of the employee information for the user name entered on the form
page. --->
<cfquery name="getEmpInfo" datasource="cfdocexamples">
SELECT * FROM EMPLOYEES
WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<!--- The following code populates the template called "employeeInfoHTTP.pdf" with data from the query. As
in the previous example, notice the use of the cfpdfsubform tag. The txtEmpID field is a hidden field on
the PDF form. --->

<cfquery name="getEmpInfo" datasource="cfdocexamples">
SELECT * FROM EMPLOYEES
WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<cfpdfform source="c:\forms\employeeInfoHTTP.pdf" action="populate">
    <cfpdfsubform name="form1">
        <cfpdfformparam name="txtFirstName" value="#getEmpInfo.FIRSTNAME#">
        <cfpdfformparam name="txtLastName" value="#getEmpInfo.LASTNAME#">
        <cfpdfformparam name="txtDeptName" value="#getEmpInfo.DEPARTMENT#">
        <cfpdfformparam name="txtEmail" value="#getEmpInfo.IM_ID#">
        <cfpdfformparam name="txtPhoneNum" value="#getEmpInfo.PHONE#">
        <cfpdfformparam name="txtEmpID" value="#getEmpInfo.Emp_ID#">
    </cfpdfsubform>
</cfpdfform>
```

従業員がフォームの情報を更新して、HTTP Post の送信ボタンをクリックすると、Acrobat から 2 番目の ColdFusion 処理ページに (フォームそのものではなく) フォームデータが送信されます。

注意：LiveCycle Designer では、PDF フォームで HTTP 送信ボタンを使用します。また、ボタンのオブジェクトプロパティの URL フィールドに、ColdFusion 処理ページの URL を指定します。

フォームデータを参照する場合は、フィールド名だけでなく、構造体を再生成します。フォームデータの構造体を確認するには、cfdump タグを使用します。

```
<!--- The following code reads the form data from the PDF form and uses it to update
    corresponding fields in the database. --->

<cfquery name="updateEmpInfo" datasource="cfdocexamples">
UPDATE EMPLOYEES
    SET FIRSTNAME = "#FORM1.SUBFORM.HEADER.TXTFIRSTNAME#",
        LASTNAME = "#FORM1.SUBFORM.HEADER.TXTLASTNAME#",
        DEPARTMENT = "#FORM1.SUBFORM.HEADER.TXTDEPTNAME#",
        IM_ID = "#FORM1.SUBFORM.HEADER.TXTEMAIL#",
        PHONE = "#FORM1.SUBFORM.HEADER.TXTPHONENUM#"
    WHERE EMP_ID = <cfqueryparam value="#FORM1.SUBFORM.TXTEMPID#">
</cfquery>
<h3>Employee Information Updated</h3>
<p><cfoutput>#FORM1.SUBFORM.HEADER.TXTFIRSTNAME#</cfoutput>,</p>
<p>Thank you for updating your employee information in the employee database.</p>
```

埋め込み PDF フォームの例

次の例では、cfdocument タグで作成した PDF ドキュメントに、インタラクティブ PDF フォームを埋め込んでいます。

ログインページでは、従業員がユーザー名とパスワードを入力します。

```
<h3>Employee Login Form</h3>
<p>Please enter your user name and password.</p>
<cfform name="loginform" action="embed2.cfm" method="post">
    <table>
        <tr>
            <td>user name:</td>
            <td><cfinput type="text" name="username" required="yes"
                message="A user name is required."></td>
        </tr>
        <tr>
            <td>password:</td>
            <td><cfinput type="password" name="password" required="yes"
                message="A password is required."></td>
        </tr>
    </table>
    <br/>
    <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

処理ページでは、クエリーを使用して、cfdocexamples データベースの値をインタラクティブ PDF フォームに設定します。インタラクティブ PDF フォームは、cfdocument タグで作成した PDF ドキュメントに埋め込みます。PDF ドキュメントは 3 つのセクションで構成されます。ドキュメントの最初のセクションと最後のセクションは cfdocumentsection タグで定義され、PDF ドキュメントに埋め込む 2 番目のセクションは cfpdfform タグで定義されます。各セクションは、PDF ドキュメントの新しいページから始まります。PDF フォームの [印刷] ボタンでは、インタラクティブ PDF フォームの前後のセクションのページも含め、ドキュメント全体が印刷されます。

```
 <cfquery name="getEmpInfo" datasource="cfdocexamples">
SELECT * FROM EMPLOYEES
WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<!--- The following code creates a PDF document with headers and footers.
--->
<cfdocument format="pdf">
    <cfdocumentitem type="header">
    <font size="-1" align="center"><i>Nondisclosure Agreement</i></font>
    </cfdocumentitem>
    <cfdocumentitem type="footer">
    <font size="-1"><i>Page <cfoutput>#cfdocument.currentpagenumber#
of#cfdocument.totalpagecount#</cfoutput></i></font>
    </cfdocumentitem>

<!--- The following code creates the first section in the PDF document. --->
    <cfdocumentsection>
    <h3>Employee Nondisclosure Agreement</h3>
    <p>Please verify the information in the enclosed form. Make any of the necessary changes
    in the online form and click the <b>Print</b> button. Sign and date the last page. Staple
    the pages together and return the completed form to your manager.</p>
    </cfdocumentsection>

<!--- The following code embeds an interactive PDF form within the PDF document with fields
    populated by the database query. The cfpdpfform tag automatically creates a section in
    the PDF document. Do not embed the cfpdfform within cfdocumentsection tags. --->

    <cfpdfform action="populate" source="c:\forms\embed.pdf">
        <cfpdfsubform name="form1">
            <cfpdfformparam name="txtEmpName"
                value="#getEmpInfo.FIRSTNAME# #getEmpInfo.LASTNAME#">
            <cfpdfformparam name="txtDeptName" value="#getEmpInfo.DEPARTMENT#">
            <cfpdfformparam name="txtEmail" value="#getEmpInfo.IM_ID#">
            <cfpdfformparam name="txtPhoneNum" value="#getEmpInfo.PHONE#">
            <cfpdfformparam name="txtManagerName" value="Randy Nielsen">
        </cfpdfsubform>
    </cfpdfform>

<!--- The following code creates the last document section. Page numbering resumes in this section. --->
    <cfdocumentsection>
    <p>I, <cfoutput>#getEmpInfo.FIRSTNAME# #getEmpInfo.LASTNAME#</cfoutput>, hereby attest
    that the information in this document is accurate and complete.</p>
    <br/><br/>
    <table border="0" cellpadding="20">
        <tr><td width="300">
        <hr/>
        <p><i>Signature</i></p></td>
        <td width="150">
        <hr/>
        <p><i>Today's Date</i></p></td></tr>
    </table>
    </cfdocumentsection>
</cfdocument>
```

更新 PDF フォームの例

次の例は、ColdFusion で既存のデータを保持したまま、PDF フォームを更新する方法を示しています。このアプリケーションでは、ユーザーが、LiveCycle で作成した空のフォームから事務用品の依頼を作成したり、既存の依頼を変更することができます。また、入力後のフォームを電子メールの添付ファイルで送信することもできます。

```
<!--- supplyReq1.cfm --->
<!--- The following code prefills fields in a blank form in LiveCycle and writes the prefilled
    form to a new file called NewRequest.pdf in the supplyReqs directory. --->
<cfpdfform source="SupplyReq.pdf" action="populate" destination="supplyReqs/NewRequest.pdf"
    overwrite="yes">
    <cfpdfsubform name="form1">
        <cfpdfformparam name="txtContactName" value="Constance Gardner">
        <cfpdfformparam name="txtCompanyName" value="Wild Ride Systems">
        <cfpdfformparam name="txtAddress" value="18 Melrose Place">
        <cfpdfformparam name="txtPhone" value="310-654-3298">
        <cfpdfformparam name="txtCity" value="Hollywood">
        <cfpdfformparam name="txtStateProv" value="CA">
        <cfpdfformparam name="txtZipCode" value="90210">
    </cfpdfsubform>
</cfpdfform>

<!--- The following code lets users choose an existing supply request form or create a new
    request from the NewRequest.pdf form. --->
<h3>Office Supply Request Form</h3>
<p>Please choose the office supply request form that you would like to open. Choose <b>New
    Supply Request</b> to create a request.</p>

<!--- The following code populates a list box in a form with files located in the specified
    directory. --->
<cfset thisDir = expandPath(".")>
<cfdirectory directory="#thisDir#/supplyReqs" action="list" name="supplyReqs">

<cfif #supplyReqs.name# is "NewRequest.pdf">
    <cfset #supplyReqs.name# = "---New Supply Request---">
</cfif>

<cfform name="fileList" action="supplyReq2.cfm" method="post">
    <cfselect name="file" query="supplyReqs" value="name" display="name"
        required="yes" size="8" multiple="no"/><br/>
    <cfinput type="submit" name="submit" value="OK">
</cfform>

<!--- supplyReq2.cfm --->
<!--- The following code displays the PDF form that the user selected. --->
<cfif #form.file# is "---New Supply Request---">
    <cfset #form.file# = "NewRequest.pdf">
</cfif>
<cfpdfform source="supplyReqs/#form.file#" action="populate"/>


<!--- supplyReq3.cfm --->
<!--- The following code reads the PDF file content from the submitted PDF form. --->
<cfpdfform source="#PDF.content#" action="read" result="fields"/>

<!--- The following code writes the PDF form to a file and overwrites the file if it exists. --->
<cfpdfform action="populate" source="#PDF.content#"
destination="SupplyReqs/supplyReq_#fields.form1.txtRequestNum#.pdf" overwrite="yes"/>

<!--- The following code customizes the display based on field values extracted from the PDF
    form. --->
<p><cfoutput>#fields.form1.txtRequester#</cfoutput>,</p>
<p>Your changes have been recorded for supply request
<cfoutput>#fields.form1.txtRequestNum#</cfoutput>.</p>
<p>If the form is complete and you would like to submit it to
<cfoutput>#fields.form1.txtContactName#</cfoutput> for processing, click <b>Submit</b>.
```

```
<!--- The following code gives the option to e-mail the submitted form as an attachment or
    return to the home page. --->
<cfform name="send" method="post" action="supplyReq4.cfm">
    <cfinput type="hidden"
value="SupplyReqs/supplyReq_#fields.form1.txtRequestNum#.pdf" name="request">
    <cfinput type="hidden" value="#fields.form1.txtRequester#" name="requester">
    <cfinput type="submit" value="Submit" name="Submit">
</cfform>
<p>If you would like to modify your request or choose another request,
<a href="supplyReq1.cfm">click here</a>.</p>

<!--- supplyReq4.cfm --->
<!--- The following code sends the completed PDF form as an attachment to the person
    responsible for processing the form. --->
<p>Your request has been submitted.</p>
<cfmail from="#form.requester#@wildride.com" to="cgardener@wildride.com"
    subject="see attachment">
    Please review the attached PDF supply request form.
    <cfmailparam file="#form.request#">
</cfmail>
```

PDF ドキュメントの組み立て

Adobe ColdFusion を使用して PDF ドキュメントを組み立てることができます。cfpdf と cfpdfparam タグを使用して、複数のソースファイルや複数のファイルのページを 1 つのドキュメントに結合します。

PDF ドキュメントの組み立てについて

Adobe ColdFusion で PDF ドキュメントを組み立てるには、cfpdf タグを使用します。このタグには、次の表に示すように、複数のソースを結合して出力ファイルを作成するアクションがいくつか用意されています。

アクション	説明
addWatermark	PDF ドキュメントのページに透かしイメージを追加します。
deletePages	PDF ドキュメントからページを削除します。
addheader	PDF ドキュメントにヘッダを追加します。
addfooter	PDF ドキュメントにフッタを追加します。
removeheaderfooter	PDF ドキュメントからヘッダとフッタを削除します。
optimize	イメージのダウンサンプリングと未使用オブジェクトの削除により、PDF ドキュメントの品質を下げます。
extracttext	指定したページまたは PDF ドキュメント全体からテキストを抽出します。
extractimage	指定したページまたは PDF ドキュメント全体からイメージを抽出します。
transform	ページレベルの変換を実行します。
getInfo	PDF ドキュメントに関連付けられている情報 (作成者、タイトル、作成日など) を抽出します。
merge	複数の PDF ソースファイルの PDF ドキュメントまたはページから 1 つの出力ファイルを組み立てます。
processddx	Adobe® LiveCycle™ Assembler の一部の機能を提供することで、cfpdf タグを拡張します。これはデフォルトのアクションです。
protect	PDF ドキュメントにパスワード保護や暗号化を設定します。

アクション	説明
read	ColdFusion 変数に PDF ドキュメントを読み込みます。
removeWatermark	PDF ドキュメントの指定のページから透かしを削除します。
setInfo	PDF ドキュメントのタイトル、サブタイトル、作成者、キーワードを設定します。
thumbnail	PDF ドキュメントの指定のページからサムネールイメージを生成します。
write	PDF 出力をファイルに書き込みます。また、Acrobat で作成したフォームをフラット化する場合や、ドキュメントを線形化する場合にも使用します。

注意：cfpdf タグを使用して、PDF ドキュメントを最初から作成することはできません。HTML コンテンツから PDF ドキュメントを作成するには、cfdocument タグを使用します。また、Report Builder を使用すると、PDF ドキュメントとしてレポートを生成できます。PDF ドキュメントをファイルに書き込むのではなく、生成した PDF 変数を cfpdf タグのソースに指定することもできます。

cfpdf タグのすべてのアクションは、1 つを除き、よく使用されるタスクへのショートカットになっています。たとえば、出力ファイルのページに透かしイメージを追加したり、ディレクトリ内のすべての PDF ドキュメントを 1 つの出力ファイルに結合したり、PDF ドキュメントをパスワード保護したりする操作が、1 行のコードで実行できます。ColdFusion には、cfpdf タグの機能を拡張する方法として、cfpdfparam タグと processddx アクションの 2 つが用意されています。

cfpdfparam タグは、cfpdf タグの merge アクションでのみ使用します。cfpdfparam タグを使用すると、出力ファイルに含めるファイルを詳細に管理できます。たとえば、異なるディレクトリにある複数のファイルのページを結合することができます。

processddx アクションは、Adobe LiveCycle Assembler の一部の機能を提供することで、cfpdf タグを拡張します。processddx アクションを使用すると、DDX (Document Description XML) 命令を処理できます。詳細については、919 ページの「DDX を使用した高度なタスク」を参照してください。DDX 命令を使用すると、コーディングは増えますが、目次の生成や自動ページ番号の追加など、複雑なタスクを実行できます。

また、ColdFusion には、PDF ファイル、DDX ファイル、PDF 変数を検証する次の 3 つの関数があります。

関数	説明
IsDDX	DDX ファイル、パス、および命令が null でなく、有効であるかどうかを判別します。また、DDX 命令に使用されているスキーマが ColdFusion でサポートされているかどうかを検証します。
IsPDFFile	PDF のソースファイル、パス、バージョンが有効で、ColdFusion を実行中のサーバーでサポートされているかどうかを判別します。また、PDF ファイルが破損していないかどうかを検証します。
IsPDFObject	メモリに保管されている PDF オブジェクトが有効であるかどうかを判別します。また、cfdocument および cfpdf タグで生成した PDF 変数の内容を検証します。

次の表に、ColdFusion で行えるドキュメント組み立てタスクの一部を示します。

タスク	アクション
目次を生成して PDF ドキュメントに追加する	cfpdf action="processddx" と DDX の TableOfContents 要素 cfpdf action="extracttext" も使用できます。
PDF ドキュメントに自動ページ番号を追加する	cfpdf action="processddx" と、_PageNumber および _LastPagenumber ビルトインキー。DDX の Header および Footer 要素内でのみ有効です。
PDF ドキュメントにヘッダおよびフッタを追加する	cfpdf action="processddx" と、DDX の Header および Footer 要素 または cfpdf action="addheader" および cfpdf action="addfooter"
透かしを追加または削除する	cfpdf action="processddx" と、DDX の Watermark および Background 要素 cfpdf action="addWatermark" および cfpdf action="removeWatermark"

タスク	アクション
PDF ドキュメントの暗号化アルゴリズムを変更する	cfpdf action="protect" encrypt="*encryption algorithm*"
PDF ドキュメントのユーザー権限を変更する	cfpdf action="protect" newOwnerPassword=*"xxxxx"*permissions=*"comma-separated list"*
PDF ドキュメントからページを削除する	cfpdf action="deletePages"
PDF ドキュメントからテキストを抽出し、XML ファイルにエクスポートする	cfpdf action="processddx" と DDX の DocumentText 要素
Acrobat で作成したフォームをフラット化する (インタラクティブフォームを削除する)	cfpdf action="write" flatten="yes"
PDF ドキュメントのページからサムネールイメージを生成する	cfpdf action="thumbnail"pages="page numbers"
PDF ドキュメントを線形化して Web 表示を高速化する	cfpdf action="write" saveOption="linear"
異なる場所にある複数のドキュメントのページやページ範囲を 1 つの PDF ドキュメントに結合する	cfpdf action="merge" と複数の cfpdfparam タグ
ディレクトリ内の PDF ドキュメントを 1 つの PDF ドキュメントに結合する	cfpdf action="merge" directory="*path*"
PDF ドキュメントをパスワード保護する	cfpdf action="protect" newUserPassword="*xxxx*"
PDF ドキュメントの初期ビューを設定する	cfpdf action="processddx" と DDX の InitialViewProfile 要素
PDF ドキュメントの別のバージョンを作成する	PDF 変数のクローンを作成する Duplicate 関数

一般的なタスクのショートカットの使用

cfpdf タグのアクションを使用して、PDF ドキュメントの一般的な組み立てや操作を行うショートカットを実行できます。

透かしイメージの追加と削除

PDF ドキュメントの透かしを追加および削除するには、addWatermark および removeWatermark アクションを使用します。次のいずれかの方法で、透かしを作成して PDF ドキュメントに適用できます。

- イメージファイルを透かしとして使用する
- イメージファイルが含まれている変数を指定する
- ColdFusion イメージを指定する
- PDF ドキュメントの 1 ページ目を透かしとして使用する

注意：また、DDX の Watermark または Background 要素と、processddx アクションを使用して、テキスト文字列の透かしを作成できます。詳細については、919 ページの「DDX を使用した高度なタスク」を参照してください。

イメージファイルを透かしとして使用

次の例は、イメージファイルを透かしとして指定する方法を示しています。

```
<cfpdf action="addWatermark" source="artBook.pdf"
    image="../cfdocs/images/artgallery/raquel05.jpg" destination="output.pdf"
    overwrite="yes">
```

デフォルトでは、イメージはページの中央に置かれます。イメージの不透明度は 3 に設定され (完全な不透明は 10)、出力ファイルの各ページの背景にイメージが表示されます。次の例では、透かしを前面に表示し、ページの左マージンから 100 ピクセル、下マージンから 100 ピクセルのオフセットを設定しています。不透明度を 1 に設定しているため、ページの内容がイメージで隠れることはありません。

```
<cfpdf action="addWatermark" source="artBook.pdf"
    image="../cfdocs/images/artgallery/raquel05.jpg" destination="output.pdf"
    overwrite="yes" foreground="yes" opacity=1 showOnPrint="no" position="100,100">
```

すべての属性と設定のリストについては、『CFML リファレンス』の cfpdf タグを参照してください。

ColdFusion 9 リリースでは、addWatermark アクションで rgb 形式および argb 形式がサポートされるようになりました。次の例は、新しいイメージのパラメータを rgb または argb に設定して cfpdf action=addwatermark を使用した場合に、ColdFusion でそのアクションが許可されることを示しています。

```
<!---setting the argb format for myImage--->
<cfset myImage = ImageNew("",200,200,"argb","gray")>

<!---adding watermark for myImage--->
<cfpdf action="addwatermark" rotation="45" foreground="true" image="#myImage#" source="RemoveArts.pdf"
destination="dest.pdf"  overwrite="yes">
```

イメージファイルが含まれている変数の使用

イメージが含まれている変数を透かしとして指定することができます。次の例は、ユーザーがイメージを選択できるフォームの作成方法を示しています。

```
<!--- The following code creates a form where you can choose an image to use
    as a watermark. --->
<h3>Choosing a Watermark</h3>
<p>Please choose the image you would like to use as a watermark.</p>

<!--- Create the ColdFusion form to select an image. --->
<table>
<cfform action="addWatermark2.cfm" method="post"
enctype="multipart/form-data">
    <tr>
        <td><img src="../cfdocs/images/artgallery/maxwell01.jpg"/><br/>
        <cfinput type="radio" name="art" value="../cfdocs/images/artgallery/maxwell01.jpg"
            checked="yes">
        Birch Forest</td>
        <td><img src="../cfdocs/images/artgallery/raquel05.jpg"/><br/>
        <cfinput type="radio" name="art" value="../cfdocs/images/artgallery/raquel05.jpg">
            Lounging Woman</td>
        <td><img src="../cfdocs/images/artgallery/jeff01.jpg"/><br/>
        <cfinput type="radio" name="art"
            value="../cfdocs/images/artgallery/jeff01.jpg">Celebration</td>
        <td><img src="../cfdocs/images/artgallery/paul01.jpg"/><br/>
        <cfinput type="radio" name="art"
            value="../cfdocs/images/artgallery/paul01.jpg">Guitarist
        </td>
    </tr>
</table>
<br/>
<cfinput type="Submit" name="submit" value="Submit"></p>
</cfform>
```

処理ページでは、フォームで選択されたイメージを PDF ファイルの透かしとして使用します。

```
<!--- ColdFusion applies the image selected from the form as the watermark in a PDF document
    by using the input variable form.art. --->
<cfpdf action="addwatermark" source="check.pdf" image="#form.art#" destination="output.pdf"
    foreground="yes" overwrite="true">
<p>The watermark has been added to your personalized checks.</p>
```

ColdFusion イメージを透かしとして使用

ColdFusion イメージを透かしとして指定できます。データベースからイメージを抽出し、メモリ内でイメージを操作できますが、操作後のイメージをファイルに書き込む必要はありません。書き込みを実行しなくても、操作後のイメージを PDF ドキュメントに透かしとして適用できます。

次の例では、最初の ColdFusion ページでデータベースからイメージを抽出して、ポップアップメニューにアートワークのタイトルを設定します。

```
<!--- Create a query to extract artwork from the cfartgallery database. --->
<cfquery name="artwork" datasource="cfartgallery">
SELECT ARTID, ARTNAME, LARGEIMAGE
FROM ART
ORDER BY ARTNAME
</cfquery>

<!--- Create a form that lists the artwork titles generated by the query. Set the value to
    LARGEIMAGE so that the image file is passed to the processing page. --->
<cfform action="addWatermarkB.cfm" method="post">
<p>Please choose a title:</p>
<cfselect name="art" query="artwork" display="ARTNAME" value="LARGEIMAGE" required="yes"
    multiple="no" size="8">
</cfselect>
<br/>
<cfinput type="submit" name="submit" value="OK">
</cfform>
```

アクションページでは、cfimage タグを使用して、選択されたファイルから ColdFusion イメージを生成します。ImageScaleToFit 関数を使用して、イメージのサイズを変更し、バイキュービック補間法を適用して解像度を向上させます。操作後のイメージを透かしに使用するには、次の例に示すように、イメージ変数を指定します。

```
<!--- Verify that an image file exists and is in a valid format. --->
<cfif IsImageFile("../cfdocs/images/artgallery/#form.art#")>
<!--- Use the cfimage tag to create a ColdFusion image from the file chosen from the list. --->
<cfimage source="../cfdocs/images/artgallery/#form.art#" action="read" name="myWatermark">

<!--- Use the ImageScaleToFit function to resize the image by using the bicubic interpolation
    method for better resolution. --->
<cfset ImageScaleToFit(myWatermark,450,450,"bicubic")>

<!--- Use the ColdFusion image variable as the watermark in a PDF document. --->
<cfpdf action="addWatermark" source="title.pdf" image="#myWatermark#"
    destination="watermarkTitle.pdf" overwrite="yes">
<cfelse>
    <p>I'm sorry, no image exists for that title. Please click the Back button and try
        again.</p>
</cfif>
```

ColdFusion イメージの詳細については、934 ページの「ColdFusion イメージの作成と操作」を参照してください。

テキストイメージを作成して透かしとして使用

ImageDrawText 関数を使用すると、ColdFusion でテキストイメージを作成し、透かしとして適用できます。たとえば、次のようにします。

```
<!--- Create a blank image that is 500 pixels square. --->
<cfset myImage=ImageNew("",500,500)>
<!--- Set the background color for the image to white. --->
<cfset ImageSetBackgroundColor(myImage,"white")>
<!---Clear the rectangle specified on myImage and apply the background color. --->
<cfset ImageClearRect(myImage,0,0,500,500)>
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(myImage)>

<!--- Draw the text. --->
<cfset attr=StructNew()>
<cfset attr.size=50>
<cfset attr.style="bold">
<cfset attr.font="Verdana">
<cfset ImageSetDrawingColor(myImage,"blue")>
<cfset ImageDrawText(myImage,"PROOF",100,250,attr)>

<!--- Write the text image to a file. --->
<cfimage action="write" source="#myImage#" destination="text.tiff" overwrite ="yes">

<!--- Use the text image as a watermark in the PDF document. --->
<cfpdf action="addwatermark" source="c:/book/1.pdf" image="text.tiff"
    destination="watermarked.pdf" overwrite="yes">
```

ColdFusion イメージの詳細については、934 ページの「ColdFusion イメージの作成と操作」を参照してください。DDX 要素を使用してテキスト文字の透かしを作成する例については、925 ページの「テキスト文字列の透かしの追加」を参照してください。

PDF ページを透かしとして使用

PDF ファイルの 1 ページ目から透かしを作成して別の PDF ドキュメントに適用するには、copyFrom 属性を使用します。次の例では、"image.PDF" の 1 ページ目から透かしを作成して、"artBook.pdf" の 2 ページ目に適用し、この出力を "output.pdf" という新しいファイルに書き込みます。

```
<cfpdf action="addWatermark" copyFrom="image.pdf" source="artBook.pdf" pages="2"
    destination="output.pdf" overwrite="yes">
```

この例では、"artBook.pdf" の 2 ページ目の背景に "image.pdf" が表示されます。ページに収まるように透かしのサイズが変更されることはありません。透かしに使用するページには、テキストのみ、グラフィックのみ、またはその両方を含めることができます。

透かしの削除

PDF ドキュメントページから透かしを削除するには、removeWatermark アクションを使用します。次の例では、PDF ドキュメント全体から透かしを削除し、ドキュメントを新しい出力ファイルに書き込みます。

```
<cfpdf action="removeWatermark" source="artBook.pdf" destination="noWatermark.pdf">
```

次の例では、PDF ドキュメントの最初の 2 ページから透かしを削除し、ソースドキュメントに上書きします。

```
<cfpdf action="removeWatermark" source="artBook.pdf" destination="artBook.pdf"
    overwrite="yes" pages="1-2">
```

ソースと宛先が同じで、overwrite 属性が yes に設定されているので、ソースファイルが出力ファイルで上書きされます。

PDF ドキュメントの最適化

PDF ドキュメントの品質を下げることで、ドキュメントを最適化することができます。PDF ドキュメントの品質を下げるには、イメージをダウンサンプリングするか、未使用オブジェクトをドキュメントから削除します。

イメージをダウンサンプリングするには、algos 属性の値を bilinear、bicubic、または nearest_neighbour に設定します。次のコードでは、イメージのダウンサンプリング後の PDF が生成されます。

```
<cfpdf action = "optimize" algo = "bicubic" source = "..\myBook.pdf" name = #myBook#>
```

コメント、JavaScript、添付ファイル、ブックマーク、メタデータなどの未使用オブジェクトを PDF ドキュメントから破棄するには、optimize アクションとともに次の属性を使用します。

```
<cfpdf action = "optimize"
    noJavaScript
    noThumbnails
    noBookmarks
    noComments
    noMetadata
    noFileAttachments
    noLinks
    nofonts>
```

encodeall を使用したページ数の最適化

新規の encodeall 属性では、ソース内のエンコードされていないストリームがすべてエンコードされます。ただし、LZW のようなダムエンコーディングと、flate のようなエンコーディングとが区別されないため、エンコードされていないストリームのみが flate エンコードされます。

例：

```
<cfpdf action=write  source="./inputFiles/Source.pdf" destination="./outputFiles/Output.pdf"
encodeAll="yes">
```

PDF に対するヘッダおよびフッタの追加と削除

次のコード例で示すように、addheader および addfooter アクションを使用すると、PDF ドキュメントにヘッダおよびフッタを追加できます。

```
<!---addfooter--->
<cfpdf action = "addfooter"
source = "../myBook.pdf"
destination = "../myBookwithfooter.pdf"
image = "adobelogo.JPG" // Use this attribute to add an image in the footer
align = "right"> // By default, the alignemnt is center

<!---addheader--->
    <cfpdf action = "addheader"
    source = "../myBook.pdf"
    destination = "../myBookwithheader.pdf"
    text = "Adobe"
    align = "left">
```

元の PDF ドキュメントが配置されている場所と、ヘッダおよびフッタを追加した新しい PDF ドキュメントの保存先を指定します。

また、フッタに挿入するイメージやテキストに加え、align、bottommargin、leftmargin、numberformat、opacity などのさまざまな属性も指定できます。

PDF ドキュメントからヘッダおよびフッタを削除するには、次のコード例で示すように removeheaderfooter アクションを使用します。

```
<cfpdf action = "removeheaderfooter" source="..\mybook.pdf" destination = "new.pdf">
```

このアクションを使用すると、PDF ドキュメント全体または指定したページからヘッダおよびフッタを削除できます。

イメージおよびテキストの抽出

extracttext および extractimage アクションを使用すると、PDF ドキュメントからテキストおよびイメージを抽出できます。

extracttext アクションは、次のコード例で示すように、PDF ドキュメント内の指定したページからすべての単語を抽出します。

```
<cfpdf action = "extracttext" source = "../myBook.pdf" pages = "5-20, 29, 80" destination
="../adobe/textdoc.txt"
```

extractimage アクションは、次のコード例で示すように、PDF ドキュメント内の指定したページからすべてのイメージを抽出します。

```
<cfpdf action = "extractimage" source = "../myBook.pdf" pages = "1-200" destination = "..\mybookimages"
imageprefix = "mybook">
```

抽出されたイメージは、destination 属性で指定したディレクトリに保存されます。抽出するイメージに追加する接頭辞 (imageprefix) を指定できます。これを指定しない場合は、イメージ名に "cf + ページ番号 " のような接頭辞が自動的に追加されます。特定の形式でイメージを保存するには、format 属性を使用します。

ページレベルの変換の実行

transform アクションを使用すると、PDF ドキュメントのページを拡大 / 縮小したり、ページの位置や回転の値を指定できます。このアクションには、サイズ (hscale、vscale)、位置 (position)、および回転 (rotation) を定義する 4 つの属性があります。次のコード例で、その使用方法を示します。回転の値は、0、90、180、270 の刻みで指定する必要があります。その他の値を指定すると、エラーが発生します。

```
<cfpdf action = "transform"
source = "..\myBook.pdf"
destination = "..\new\myBook.pdf">
hscale = ".5"
vscale = ".15"
position = "8, 10"
rotation = "180">
```

PDF ドキュメントからのページの削除

PDF ドキュメントからページを削除して、結果をファイルに書き込むには、deletepages アクションを使用します。削除するページは、単一のページ、ページ範囲 (たとえば "81–97")、またはページのカンマ区切りリストで指定できます。次に例を示します。

```
<cfpdf action="deletePages" source="myBook.pdf" pages="10-15,21,89"
    destination="abridged.pdf" overwrite="yes">
```

PDF ファイルの保護

protect アクションを使用すると、PDF ドキュメントのパスワード保護、権限設定、および暗号化を行うことができます。

パスワードの設定

ColdFusion では、オーナーパスワードとユーザーパスワードの 2 種類のパスワードがサポートされています。オーナーパスワードは、ドキュメントの権限を変更できるユーザーを限定するために使用します。オーナーパスワードを指定するときは、ドキュメントに権限を設定して、ユーザーが実行できる操作 (ドキュメントの印刷、内容の変更、内容の抽出など) を制限します。次のコードでは、ドキュメントのオーナーパスワードを作成します。

```
<cfpdf action="protect" newOwnerPassword="splunge" source="timesheet.pdf"
    destination="timesheet.pdf" overwrite="yes" permissions="AllowPrinting">
```

ドキュメントをパスワード保護する場合は、ユーザーパスワードを設定します。ユーザーパスワードは、ドキュメントを開けるユーザーを限定するために使用します。ドキュメントにユーザーパスワードを設定した場合、ユーザーがそのファイルを開こうとすると、パスワードの入力が要求されます。次の例では、ドキュメントのユーザーパスワードを設定します。

```
<cfpdf action="protect" newUserPassword="openSesame" source="timesheet.pdf"
    destination="myTimesheet.pdf">
```

この例の場合、ユーザーが正しいパスワードを入力した後は、PDF ドキュメントに対する制限は一切ありません。使用方法の制限に加えてパスワード保護も行うには、ユーザーパスワードとオーナーパスワードを追加します。オーナーパスワードを使用すると、権限を設定することができます。たとえば、次の例のようにします。

```
<cfpdf action="protect" newUserPassword="openSesame" newOwnerPassword="topSecret"
    source="timesheet.pdf" destination="myTimesheet.pdf" overwrite="yes"
    permissions="AllowPrinting">
```

この例の場合、ユーザーパスワード (openSesame) を入力したユーザーは、ドキュメントの印刷のみを実行できます。オーナーパスワード (topSecret) を入力したユーザーはドキュメントの所有者と見なされて、ファイルへのフルアクセスが可能になり、ファイルのユーザー権限を変更できます。

PDF ドキュメントの権限の設定

PDF ドキュメントの権限を設定するには、newOwnerPassword を指定します。逆に、permissions 属性を設定せずに newOwnerPassword を設定することはできません。権限を変更したり、パスワードを追加できるのは、所有者のみです。所有者が PDF ドキュメントに設定できる権限のリストについては、『CFML リファレンス』の cfpdf を参照してください。

all と none 以外では、ドキュメントの権限をカンマ区切りリストで指定できます。次に例を示します。

```
<cfpdf action="protect" permissions="AllowinPrinting,AllowDegradedPrinting,AllowSecure"
    source="timesheet.pdf" newOwnerPassword="private" newUserPassword="openSesame"
    destination="myTimesheet.pdf">
```

この例の場合、ユーザーは PDF フォームを開く前に、パスワード openSesame を入力する必要があります。すべてのユーザーはドキュメントを任意の解像度で印刷できますが、ドキュメントや権限を変更できるのは所有者のみです。

PDF ファイルの暗号化

PDF ファイルに protect アクションを指定すると、デフォルトでは、RC4 128 ビットアルゴリズムでファイルが暗号化されます。ColdFusion サーバーで使用する Acrobat のバージョンによっては、暗号化を設定してドキュメントの内容を保護し、検索エンジンで PDF ファイルのメタデータが取得されないようにすることができます。

暗号化アルゴリズムを変更する場合は、encrypt 属性を使用します。サポートされている暗号化アルゴリズムのリストについては、『CFML リファレンス』の cfpdf を参照してください。

次の例では、パスワードの暗号化アルゴリズムを RC4 40 ビット暗号化に変更します。

```
<cfpdf action="protect" source="confidential.pdf" destination="confidential.pdf"
    overwrite="yes" newOwnerPassword="paSSword1" newUserPassword="openSesame"
    encrypt="RC4_40">
```

ColdFusion で PDF ドキュメントが暗号化されないようにするには、次の例のように、暗号化アルゴリズムに none を設定します。

```
<cfpdf action="protect" source="confidential.pdf" encrypt="none" destination="public.pdf">
```

ファイルを復号するには、オーナーパスワードまたはユーザーパスワードを使用して、出力を別のファイルに書き込みます。次のコードでは、"confidential.pdf" ファイルを復号して、"myDocument.pdf" という新しいファイルに書き込みます。

```
<cfpdf action="write" source="confidential.pdf" password="paSSword1"
    destination="myDocument.pdf">
```

PDF ドキュメント情報の管理

ソース PDF ドキュメントとともに保存されている情報 (作成日、PDF ドキュメントの作成に使用されたアプリケーション、ドキュメントを作成したユーザーの名前など) を取得するには、getInfo アクションを使用します。データ要素のリストについては、『CFML リファレンス』の **PDF file information elements** を参照してください。

出力ファイルに関する情報 (作成者、タイトル、サブタイトル、キーワードなど) を指定するには、setInfo アクションを使用します。この情報は、PDF ドキュメントのアーカイブや検索を行うときに役立ちます。PDF ドキュメントの情報は、ドキュメントとともに表示および印刷されません。

次の例は、税務文書にキーワードを設定する方法を示しています。この情報は、事業形態別 (個人事業主、組合、小企業) の納税申告義務に基づいて、ドキュメントを組み立てる場合に役立ちます。一部の事業形態では、同じフォームやドキュメントを共用します。各ドキュメントに事業形態のキーワードを設定して 1 つのディレクトリに保存しておき、キーワードの値に基づいてドキュメントを検索することができます。次のコードでは、"p535.pdf" 税務文書に 3 つのキーワードを設定します。

```
<cfset taxKeys=StructNew()>
<cfset taxKeys.keywords="Sole Proprietor,Partnership,S Corporation">
<cfpdf action="setInfo" source="taxes\p535.pdf" info="#taxKeys#"
    destination="taxes\p535.pdf" overwrite="yes">
```

setInfo アクションを使用すると、該当する既存のキーと値のペアは上書きされます。たとえば前の例の場合は、"pc535.pdf" ドキュメントに "tax reference" というキーワードが含まれていても、"Sole Proprietor,Partnership,S Corporation" で上書きされます。

税務文書に関するすべての情報を取得するには、次の例のように、cfdump タグと getInfo アクションを使用します。

```
<cfpdf action="getInfo" source="taxes\p535.pdf" name="taxInfo">
<cfdump var="#taxInfo#">
```

PDF ドキュメントのキーワードのみを取得するには、次のコードを使用します。

```
<cfpdf action="getInfo" source="taxes\p535.pdf" name="taxInfo">
<cfoutput>#taxInfo.keywords#</cfoutput>
```

write アクションでの name 属性の使用

<cfpdf action = "write"> と name 属性を同時に使用することで、PDF ドキュメントの変数名を指定できるようになりました。次に例を示します。

```
<cfpdf action="write" source="myBook" name=#myBook# version="1.4">
```

この機能は、cfpdf タグと cfpdfform タグの両方で使用できます。

PDF ドキュメントの結合

ColdFusion では、次の方法で PDF ドキュメントを結合することができます。

- 指定のディレクトリ内のすべての PDF ファイルを結合する。
- カンマ区切りリストの PDF ファイルを結合する。
- 特定の PDF ファイルやそのページを明示的に結合する (各ソースファイルは異なる場所に保存されていてもかまいません)。
- cfdocument タグまたは cfpdf タグで生成した PDF 変数の内容を結合する。
- PDF パッケージを作成します。

ディレクトリの内容を結合するには、次の例に示すように、merge アクションを使用して、ソース PDF ファイルが保存されているディレクトリを指定します。

```
<cfpdf action="merge" directory="c:/BookFiles" destination="myBook.pdf" overwrite="yes">
```

デフォルトでは、タイムスタンプの降順でソースファイルが結合されます。PDF ファイルを追加する順序を制御するには、order と ascending 属性を設定します。次のコードでは、ファイルのタイムスタンプの昇順でファイルを結合します。

```
<cfpdf action="merge" directory="c:/BookFiles" destination="myBook.pdf" order="name"
    ascending="yes" overwrite="yes">
```

デフォルトでは、有効な PDF ドキュメント以外のファイルが指定のディレクトリに見つかっても、結合処理は続行されます。この設定を無効にするには、次の例のように stopOnError 属性を yes に設定します。

```
<cfpdf action="merge" directory="c:/BookFiles" destination="myBook.pdf" order="time"
    ascending="yes" overwrite="yes" stopOnError="yes">
```

カンマ区切りリストの PDF ファイルを結合することもできます。この場合は、次の例に示すように、各ファイルの絶対パスを指定します。

```
<cfpdf action="merge"
    source="c:\coldfusion\wwwroot\lion\Chap1.pdf,c:\coldfusion\wwwroot\lion\Chap2.pdf"
    destination="twoChaps.pdf" overwrite="yes">
```

追加するファイルを詳細に制御する場合は、cfpdfparam タグと cfpdf タグを使用します。cfpdfparam タグを使用すると、異なるディレクトリに保存されているドキュメントやドキュメントのページが 1 つの出力ファイルに結合されます。
cfpdfparam タグを使用した場合、PDF ファイルは、コード内で記述された順に出力ファイルに追加されます。次の例では、表紙、タイトル、著作権のページの後に、概要の先頭 5 ページが続き、その後に、第 1 章のすべてのページ、第 2 章の先頭ページと 80 ～ 95 ページが続きます。

```
<!--- Use the cfdocument tag to create PDF content and write the output to a variable called coverPage.--->
<cfdocument format="PDF" name="coverPage">
<html>
<body>
    <h1>Cover page</h1>
    <p>Please review the enclosed document for technical accuracy and completeness.</p>
</body>
</html>
</cfdocument>

<!--- Use the cfpdf tag to merge the cover page generated in ColdFusion with pages from PDF files in
different locations. --->
<cfpdf action="merge" destination="myBook.pdf" overwrite="yes" keepBookmark="yes">
    <cfpdfparam source="coverPage">
    <cfpdfparam source="title.pdf">
    <cfpdfparam source="e:\legal\copyright.pdf">
    <cfpdfparam source="boilerplate\intro.pdf" pages="1-5">
    <cfpdfparam source="bookfiles\chap1.pdf">
    <cfpdfparam source="bookfiles\chap2.pdf" pages="1,80-95">
</cfpdf>
```

keepbookmark 属性を yes に設定しているので、ソースドキュメントのしおりは出力ファイルに引き継がれます。

注意：cfpdf タグを使用して、PDF ドキュメントにしおりを作成することはできません。

PDF ポートフォリオの作成

package = "true" 属性と merge アクションを同時に使用することで、PDF パッケージを作成できるようになりました。

```
<cfpdf action="merge" package="yes" destination="./myBook/adobetest.pdf" overwrite="yes">
    <cfpdfparam source="./inputFiles/c.zip" >
    <cfpdfparam source="./inputFiles/d.jpg" >
    <cfpdfparam source="./inputFiles/a.pdf" >
    <cfpdfparam source="./inputFiles/z.txt" >
    <cfpdfparam source="./inputFiles/MSTribute.pps" >
    <cfpdfparam source="./inputFiles/Test1.docx" >
    <cfpdfparam source="./inputFiles/NewMovie.mp3" >
    <cfpdfparam source="./inputFiles/testserver.air" >
    <cfpdfparam source="./inputFiles/123.xml" >
    <cfpdfparam source="./inputFiles/New_test_case.xls" >
</cfpdf>
```

Acrobat で作成したフォームのフラット化

フォームをフラット化すると、フォームフィールドのインタラクティブ機能が削除されます。このアクションは、データを変更できないようにしてフォームデータを表示するときに便利です。PDF フォームをフラット化するには、次の例のように write アクションを使用します。

```
<cfpdf action="write" flatten="yes" source="taxForms\f1040.pdf"
    destination="taxforms/flatForm.pdf" overwrite="yes">
    <a href="http://localhost:8500/Lion/taxforms/flatForm.pdf">1040 form</a>
```

注意：Acrobat で作成した、あらかじめ値が設定されているフォームをフラット化すると、フォームフィールドからデータが削除されます。cfpdf タグの merge アクションのソースファイルとして、Acrobat で作成したフォームを指定した場合、このフォームは自動的にフラット化されます。フィールドに値が入力されていた場合は、フォームフィールドからデータが削除されます。ColdFusion では、LiveCycle で作成したフォームのフラット化はサポートされていません。

PDF ドキュメントの線形化による Web 表示の高速化

Web 経由での PDF ファイルへのアクセスを効率化するには、PDF ドキュメントを線形化します。PDF ファイルを線形化すると、ファイル全体が Web サーバーからダウンロードされる前に、PDF ファイルの先頭ページがブラウザに表示されるようになります。したがって、線形化した PDF ドキュメントは、開くのにほとんど時間がかかりません。

PDF ドキュメントを線形化するには、write アクションの saveOption 属性を指定します。次の例では、出力ファイルを線形化形式で保存します。

```
<cfpdf action="write" saveOption="linear" source="myBook.pdf" destination="fastBook.pdf"
    overwrite="yes">
```

注意：処理対象が大きなドキュメントの場合、線形化の処理はパフォーマンスの低下を招くことがあります。

PDF ページからのサムネールイメージの生成

thumbnail アクションを使用すると、PDF ページからサムネールイメージを生成します。thumbnail アクションで source 属性しか指定されていない場合は、CFM ページを基準にした "thumbnails" という相対ディレクトリが自動的に作成され、ドキュメントの各ページから生成された JPEG イメージはこのディレクトリに保存されます。ファイル名は、次の形式になります。

```
PDFdocumentName_page_n.JPG
```

たとえば、次の例のソースファイルは 100 ページで構成されているとします。

```
<cfpdf action="thumbnail" source="myBook.pdf">
```

ColdFusion では次のファイルが生成されて、"thumbnails" ディレクトリに保存されます。

```
myBook_page_1.jpg
myBook_page_2.jpg
myBook_page_3.jpg
...
myBook_page_100.jpg
```

宛先を指定した場合には、"thumbnails" ディレクトリは作成されず、代わりに、指定したディレクトリにファイルが保存されます。次のコードでは、"myBook.pdf" の 1 ページ目から "myBook_page_1.jpg" というサムネールイメージを生成し、"images" というディレクトリに保存します。このディレクトリは、CFM ページを基準にした相対ディレクトリです。

```
<cfpdf action="thumbnail" source="myBook.pdf" pages="1" destination="images">
```

サムネールのファイル名の接頭辞を変更したり、イメージのファイル形式を PNG や TIFF に変更するには、imagePrefix および format 属性を指定します。次のコードでは、"myBook.pdf" の 2 ページ目から "TOC_page_2.PNG" というファイルを生成します。

```
<cfpdf action="thumbnail" source="myBook.pdf" pages="2" imagePrefix="TOC" format="PNG"
    destination="images">
```

次のコードでは、ページ範囲からサムネールを生成し、イメージの背景を透明にします (デフォルトは不透明)。

```
<cfpdf action="thumbnail" source="myBook.pdf" pages="1-10,15,8-16,59" transparent="yes"
    destination="\myBook\subset" imagePrefix="abridged">
```

サムネールイメージを生成して、ソース PDF ページへのリンクを設定する方法の例については、『CFML リファレンス』の cfpdf タグを参照してください。

ColdFusion 9 リリースでは、thumbnail アクションに新規属性がいくつか導入されています。

- hires : この属性を true に設定すると、ページから高解像度のイメージを抽出できます。この属性は、ドキュメントに高解像度のイメージが含まれていて、それらのイメージの解像度を維持したい場合に便利です。

  次に例を示します。

  ```
  <cfpdf action="thumbnail" source="./WORK/myBook.pdf" destination="./WORK/Testing_CFPDF"
  overwrite="true" hires="yes">
  ```

- overridepage: この属性を true に設定すると、PDF のページサイズではなく、そのページ内に存在するイメージのサイズに準拠したサムネールが生成されます。イメージが存在しない場合、サイズはページの最大サイズに設定されます。
- compresstiffs: サムネールイメージのサイズを圧縮するには、この属性を使用します。属性の名前が示唆しているように、この属性は TIFF 形式に対してのみ有効です。次に例を示します。

  ```
  <cfpdf action="thumbnail" source="C:\WORK\myBook.pdf" destination="C:\WORK\Testing_CFPDF"
  overwrite="true" hires="yes" format="tiff" compresstiffs="yes">
  ```

- maxscale: サムネールイメージの最大スケールに整数値を指定するには、この属性を使用します。
- maxlength: サムネールイメージの最大長の整数値を指定するには、この属性を使用します。
- maxbreadth: サムネールイメージの最大幅の整数値を指定するには、この属性を使用します。

  次の例は、maxscale、maxlength、および maxbreadth の使用方法を示しています。

  ```
  <cfpdf action="thumbnail" source="./WORK/myBook.pdf" destination="./WORK/Testing_CFPDF"
  overwrite="true" format="jpg" maxscale="3" maxlength="300" maxbreadth="200" hires="yes" scale="100">
  ```

  注意：通常、maxscale 属性を使用する場合は、scale 属性の値を 100 に設定します。

Duplicate 関数による異なるバージョンの PDF ドキュメントの作成

Duplicate 関数を使用すると、PDF 変数のクローンを作成できます。この方法を使用すると、1 つのソースファイルからいくつかのバージョンの PDF ドキュメントを効率的に作成することができます。たとえば、PDF 変数のクローンを作成し、各クローンで異なるアクションを実行することで、ユーザーごとに PDF 出力をカスタマイズできます。次の例では、PDF ドキュメントのクローンをメモリに作成して、透かしを含むバージョンと、権限を制限したバージョンを作成しています。

```
<cfset filename="coldfusion.pdf">
<!--- This code reads a PDF document into a PDF variable called pdfVar1.
        --->
<cfpdf action="read" source="#filename#" name="pdfVar1">
<!--- This code uses the Duplicate function to create a clone of pdfVar1 called pdfVar2. --->
<cfset pdfVar2=Duplicate(pdfVar1)>
<!--- This code creates a watermarked version of the source PDF document from the pdfVar1
    variable. --->
<cfpdf action="addwatermark" source="pdfVar1" rotation="45" image="watermark.jpg"
    destination="watermark_coldfusion.pdf" overwrite="yes">
<!--- This code creates a protected version of the source PDF document from the pdfVar2
    variable. --->
<cfpdf action=protect source="pdfVar2" encrypt="RC4_128" permissions="none"
    newownerpassword="owner1" destination="restricted_coldfusion.pdf" overwrite="yes">
```

DDX を使用した高度なタスク

LiveCycle Assembler は、DDX を処理するサーバーベースのアプリケーションです。DDX は、PDF 出力ファイルの定義に使用する宣言型のマークアップ言語です。

processddx アクションを使用すると、LiveCycle Assembler をインストールしなくても、DDX 命令を処理できます。cfpdf のその他のアクションで利用できるすべての機能に加えて、DDX 命令を使用すれば、高度なタスクを実行できます。たとえば、目次を生成して PDF ドキュメントに追加したり、自動ページ番号を使用したヘッダやフッタを追加したり、PDF ドキュメントのグループを作成して形式設定を適用することができます。

ColdFusion では、LiveCycle Assembler のすべての機能が使用できるわけではありません。ColdFusion からアクセス可能な DDX 要素のリストについては、『CFML リファレンス』の **Supported DDX elements** を参照してください。

DDX の完全なシンタックスについては、『Adobe LiveCycle Assembler Document Description XML Reference』を参照してください。

ColdFusion での DDX 命令の使用

ColdFusion で DDX 命令を直接入力することもできますが、一般的には、外部の DDX ファイルを使用します。DDX ファイルは、基本的には DDX の拡張子が付いた XML ファイルです (merge.ddx など)。DDX ファイルは、任意のテキストエディタを使用して作成できます。DDX のシンタックスでは、命令を DDX の開始タグと終了タグで囲む必要があります。次の例では、PDF 要素を使用して、2 つの PDF ソースファイル (Doc1 と Doc2) を 1 つの結果ファイル (Out1) に結合するように命令しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
<PDF source="Doc1"/>
<PDF source="Doc2"/>
</PDF>
</DDX>
```

ColdFusion では、IsDDX 関数を使用して、ソース DDX ファイルを検証できます。

```
<!--- The following code verifies that the DDX file exists and the DDX instructions are
    valid. --->
<cfif IsDDX("merge.ddx")>
```

ColdFusion で DDX 命令を実装するには、2 つの構造体を作成します。1 つは入力の構造体で、DDX の入力命令と PDF ソースファイルをマッピングします。もう 1 つは出力の構造体で、DDX の出力命令と PDF 出力ファイルをマッピングします。

次のコードでは、"Chap1.pdf" と "Chap2.pdf" という 2 つのファイルと、DDX ファイルで定義されている Doc1 と Doc2 の source をマッピングします。

```
<!--- This code creates a structure for the input files. --->
<cfset inputStruct=StructNew()>
<cfset inputStruct.Doc1="Chap1.pdf">
<cfset inputStruct.Doc2="Chap2.pdf">
```

次のコードでは、"twoChaps.pdf" という出力ファイルと、DDX ファイルで定義されている Out1 の result 命令をマッピングします。

```
<!--- This code creates a structure for the output file. --->
<cfset outputStruct=StructNew()>
<cfset outputStruct.Out1="twoChaps.pdf">
```

DDX 命令を処理するには、cfpdf タグの processddx アクションを使用します。このタグでは、次の例に示すように、DDX ファイル、入力の構造体、および出力の構造体を参照します。

```
<cfpdf action="processddx" ddxfile="merge.ddx" inputfiles="#inputStruct#"
    outputfiles="#outputStruct#" name="myBook">
```

name 属性で作成される変数を使用して、アクションの成功または失敗を確認できます。アクションが成功した場合は、出力の構造体で指定されている名前と場所に出力ファイルが生成されます。次のコードでは、構造体をダンプして、出力ファイルごとに成功、失敗の理由、失敗のメッセージ (理由が不明な場合) のいずれかを表示します。

```
<cfdump var="#myBook#">
```

これと同じ操作は、次の例のように、ColdFusion の merge アクションを使用して実行できます。

```
<cfpdf action="merge" destination="twoChaps.pdf" overwrite="yes">
    <cfpdfparam source="Chap1.pdf">
    <cfpdfparam source="Chap2.pdf">
</cfpdf>
</cfif>
```

この場合は、merge アクションを使用したほうが簡単で、わかりやすくなります。DDX は、cfpdf タグのその他のアクションで実現できないタスクを実行したり、特定の要素を詳細に制御したい場合に役立ちます。

目次の追加

目次ページを生成して PDF 出力ファイルに追加するには、DDX 命令を使用します。目次は、複数のソースからドキュメントを組み立てる場合に便利です。生成される目次には、組み立てた PDF ドキュメントのページへのアクティブリンクを含めることもできます。次のコードは、2 つのドキュメントを結合して目次を追加する DDX 命令の作成方法を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
    <PDF result="Out1">
            <PDF source="DocumentTitle"/>
            <TableOfContents/>
            <PDF source="Doc1"/>
            <PDF source="Doc2"/>
    </PDF>
</DDX>
```

TableOfContents 要素は、それ以降の PDF source 要素から目次を生成します。目次を生成する場合は、命令を配置する順序が重要になります。この例の場合は、Title の後の Doc 1 と Doc 2 の前に、独立したページとして目次が生成されます。Doc 1 と 2 のエントリは目次に含まれますが、タイトルページのエントリは含まれません。これは、タイトルページが目次よりも前にあるからです。

対応する ColdFusion ページでは、次の例に示すように、TableOfContents 要素を参照する必要はありません。

```
<!--- The following code verifies that the DDX file exists and the DDX instructions are
    valid. --->
<cfif IsDDX("makeBook.ddx")>

<!--- This code creates a structure for the input files. --->
<cfset inputStruct=StructNew()>
<cfset inputStruct.Title="Title.pdf">
<cfset inputStruct.Doc1="Chap1.pdf">
<cfset inputStruct.Doc2="Chap2.pdf">

<!--- This code creates a structure for the output file. --->
<cfset outputStruct=StructNew()>
<cfset outputStruct.Out1="Book.pdf">

<!--- This code processes the DDX instructions and generates the book. --->
<cfpdf action="processddx" ddxfile="makeBook.ddx" inputfiles="#inputStruct#"
    outputfiles="#outputStruct#" name="myBook">
</cfif>
```

DDX 命令によって目次が生成され、DDX ファイルで指定した PDF ドキュメント内の場所に挿入されます。デフォルトでは、結合した PDF ドキュメントの最上位レベルのしおりへのアクティブリンクが目次に設定されます。

デフォルトの TableOfContents 設定は DDX ファイルで変更できます。次に例を示します。

```
<TableOfContents maxBookmarkLevel="infinite" bookmarkTitle="Table of Contents"
    includeInTOC="false"/>
```

maxBookmarkLevel 属性では、目次ページに含めるしおりのレベルを指定します。有効な値は、infinite または整数です。bookmarkTitle 属性では、出力ファイルの目次ページにしおりを追加します。includeInTOC 属性では、目次ページにしおりのタイトルを含めるかどうかを指定します。

注意：DDX のソースとしてキーワードは指定できません。たとえば、<PDF source = "Title"/> と指定して DDX ファイルに <_BookmarkTitle/> タグを追加した場合は、ColdFusion で例外が発生します。これは、_BookmarkTitle タグが TITLE に変換され、DDX で大文字と小文字が区別されるためです。

TableOfContents 要素の詳細については、『Adobe LiveCycle Assembler Document Description XML Reference』を参照してください。

ヘッダおよびフッタの追加

PDF ドキュメントにヘッダおよびフッタを追加するには、DDX ファイルで Header および Footer 要素を指定します。次の例では、Doc2 という PDF ソースにヘッダとフッタを指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
    <PDF result="Out1">
        <PDF source="Title"/>
        <TableOfContents/>
        <PDF source="Doc2" >
            <Header>
                <Right>
                    <StyledText><p>Right-justified header text</p></StyledText>
                </Right>
                <Left>
                    <StyledText><p>Left-justified header text</p></StyledText>
                </Left>
            </Header>
            <Footer>
                <Center>
                    <StyledText><p>Centered Footer</p></StyledText>
                </Center>
            </Footer>
        </PDF>
    </PDF>
</DDX>
```

この例の場合、Header と Footer 要素は、Doc2 の PDF source の開始タグと終了タグの中に記述されているので、Doc2 にのみ適用されます。つまり、この Header および Footer 要素よりも前にある目次やタイトルページには適用されません。

ヘッダおよびフッタの形式設定

DDX 命令を使用して、次のタスクを実行します。

- ヘッダおよびフッタに自動ページ番号を追加する
- スタイルプロファイルを使用する
- PDF 出力ファイルのドキュメントをグループ化する

自動ページ番号の追加

自動ページ番号を追加するには、Header または Footer 要素の中で、ビルトインキー _PageNumber および _LastPagenumber を使用します。次のコードでは、右揃えの自動ページ番号が付いたフッタを作成します。

```
<Footer>
    <Right>
        <StyledText>
            <p>Page <_PageNumber/> of <_LastPageNumber/></p>
        </StyledText>
    </Right>
</Footer>
```

出力ファイルの 1 ページ目には "**1 / n** ページ " と表示され、以降のページでも対応するページ番号が表示されます。

ビルトインキーの詳細については、『Adobe LiveCycle Assembler Document Description XML Reference』を参照してください。

スタイルプロファイルの使用

前の例では、StyledText 要素を使用してインラインテキストに形式設定を適用しました。参照によって適用可能なスタイルを定義するには、StyleProfile 要素を使用します。スタイルプロファイルを使用すると、特定のスタイルを PDF 出力ファイルのさまざまな要素に適用できるようになります。次のコードでは、目次の Header のスタイルプロファイルを定義しています。

```
<StyleProfile name="TOCheaderStyle">
    <Header>
        <Center>
            <StyledText>
            <p> color="red" font-weight="bold" font="Arial">Table of Contents</p>
            </StyledText>
        </Center>
    </Header>
</StyleProfile>
```

このスタイルプロファイルを適用するには、次の例のように、Header 要素の styleReference 属性を使用して、StyleProfile の名前を指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
    <PDF source="Title"/>
    <TableOfContents>
        <Header styleReference="TOCheaderStyle"/>
    </TableOfContents>
        <PDF source="Doc1"/>
        <PDF source="Doc2"/>
        <PDF source="Doc3"/>
        <PDF source="Doc4"/>
    </PDF>

    <StyleProfile name="TOCheaderStyle">
    <Header>
        <Center>
            <StyledText>
            <p> color="red" font-weight="bold" font="Arial">Table of Contents</p>
            </StyledText>
        </Center>
    </Header>
    </StyleProfile>
</DDX>
```

PDF ドキュメントのグループ化

出力 PDF ファイル内のドキュメントグループにスタイルプロファイルを適用するには、PDFGroup 要素を使用します。次の例では、出力ファイルで章のグループを作成し、そのグループ内のすべてのドキュメントの Footer 要素にスタイルプロファイルを適用しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
    <PageLabel prefix="page " format="Decimal"/>
    <PDF source="Title"/>
    <TableOfContents>
        ...
    </TableOfContents>
    <PDFGroup>
        <Footer styleReference="FooterStyle" />
        <PDF source="Doc1"/>
        <PDF source="Doc2"/>
        <PDF source="Doc3"/>
        <PDF source="Doc4"/>
    </PDFGroup>
    </PDF>

<StyleProfile name="FooterStyle">
    <Footer>
        <Left>
            <StyledText>
            <p font-size="9pt"><i>CFML Reference</i></p>
            </StyledText>
        </Left>
        <Right>
            <StyledText>
            <p font-size="9pt">Page <_PageNumber/> of <_LastPageNumber/></p>
            </StyledText>
        </Right>
    </Footer>
    </StyleProfile>
</DDX>
```

完全な例については、929 ページの「DDX 命令によるドキュメントの作成」を参照してください。

PDF ドキュメントの初期ビューの設定

PDF ドキュメントの初期ビューを設定するには、DDX の InitialViewProfile 要素を使用します。初期ビューを設定すると、PDF 出力ファイルが初めて Adobe Reader で開かれたときの表示方法を指定できます。InitialViewProfile を参照するには、次の例に示すように、PDFresult 要素の InitialView 属性を使用します。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1" initialView="firstView">
        ...
<InitialViewProfile name="firstView" show="BookmarksPanel" magnification="FitPage"
    openToPage="2"/>
...
</DDX>
```

この例の場合、PDF ドキュメントが初めて Adobe Reader で表示されると、ドキュメントの 2 ページ目が開かれ、しおりパネルが表示されます。ドキュメントの倍率は、ページが収まるように調整されます。

IntialViewProfile の設定については、『Adobe LiveCycle Assembler Document Description XML Reference』を参照してください。

テキスト文字列の透かしの追加

DDX の Watermark または Background 要素と、processddx アクションを使用して、テキスト文字列の透かしを作成できます。Background 要素は背景 (ページの内容の後ろ) に、Watermark 要素は前面 (ページの内容の前) に表示されます。両要素のシンタックスは同じです。

次の例は、"DRAFT" というテキスト文字列を透かしに使用する場合の DDX ページです。透かしは、出力ファイルの各ページに表示されます。デフォルトでは、透かしはページの中央に表示されます。この例では、透かしを 30 度回転します。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
        <Watermark rotation="30" opacity="65%">
            <StyledText><p font-size="50pt" font-weight="bold" color="lightgray"
                font="Arial">DRAFT</p></StyledText>
        </Watermark>
...
    </PDF>
</DDX>
```

次の例では、1 ページおきに異なる背景を追加しています。verticalAnchor 属性で、背景テキストをページの上部に表示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
        <Background alternation="EvenPages" verticalAnchor="Top">
            <StyledText><p font-size="20pt" font-weight="bold" color="gray"
                font="Arial">DRAFT</p></StyledText>
        </Background>
        <Background alternation="OddPages" verticalAnchor="Top">
        <StyledText><p font-size="20pt" font-weight="bold" color="gray"
            font="Arial"><i>Beta 1</i></p></StyledText>
        </Background>
...
    </PDF>
</DDX>
```

透かしを出力ファイル全体ではなく、個別のソースファイルに適用することもできます。次の例では、ドキュメントの最初の 3 つの章に異なる背景を適用します。第 4 章には背景を適用しません。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
            <PDF source="Doc1">
            <Background>
            <StyledText><p font-size="20pt" font-weight="bold" color="lightgray"
                font="Arial">CHAPTER 1</p></StyledText>
            </Background>
        </PDF>
        <PDF source="Doc2">
            <Background>
                <StyledText><p font-size="20pt" font-weight="bold"
                    color="lightgray" font="Arial">CHAPTER 2</p></StyledText>
            </Background>
        </PDF>
        <PDF source="Doc3">
            <Background>
                <StyledText><p font-size="20pt" font-weight="bold"
                    color="lightgray" font="Arial">CHAPTER 3</p></StyledText>
            </Background>
        </PDF>
        <PDF source="Doc4"/>
    </PDF>
</DDX>
```

DDX 命令を使用して透かしを作成する方法の詳細については、『Adobe LiveCycle Assembler Document Description XML Reference』を参照してください。

PDF ドキュメントからのテキストの抽出

DDX の DocumentText 要素を使用すると、PDF ドキュメント内のテキストが含まれた XML ファイルを返すことができます。PDF 要素と同様に、DocumentText 要素の result 属性を指定し、この開始タグと終了タグの中に PDFsource 要素を記述します。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
    <DocumentText result="Out1">
        <PDF source="doc1"/>
    </DocumentText>
</DDX>
```

次のコードは、この DDX ファイルを呼び出す CFM ページです。出力の書き込み先には PDF ファイルではなく、XML ファイルを指定します。

```
<cfif IsDDX("documentText.ddx">
    <cfset ddxfile = ExpandPath("documentText.ddx")>
    <cfset sourcefile1 = ExpandPath("book1.pdf")>
    <cfset destinationfile = ExpandPath("textDoc.xml")>

    <cffile action="read" variable="myVar" file="#ddxfile#"/>

    <cfset inputStruct=StructNew()>
    <cfset inputStruct.Doc1="#sourcefile1#">

    <cfset outputStruct=StructNew()>
    <cfset outputStruct.Out1="#destinationfile#">

    <cfpdf action="processddx" ddxfile="#myVar#" inputfiles="#inputStruct#" outputfiles="#outputStruct#"
name="ddxVar">

    <!--- Use the cfdump tag to verify that the PDF files processed successfully. --->
    <cfdump var="#ddxVar#">
</cfif>
```

XML ファイルは "doctext.xsd" で定義されているスキーマに準拠します。詳細については、http://ns.adobe.com/DDX/DocText/1.0 を参照してください。

複数のソースドキュメントを指定した場合は、各ページが 1 つのファイルに集約されます。次の例は、2 つのドキュメントの一部のページを 1 つの出力ファイルに結合する DDX コードです。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<DocumentText result="Out1">
    <PDF source="doc1" pages="1-10"/>
    <PDF source="doc2" pages="3-5"/>
</DocumentText>
</DDX>
```

AcroForm での FDF フォーマットのサポート

PDF の Acro Form で、Forms Data Format (FDF) がサポートされるようになりました。read アクションおよび populate アクションを使用して、FDF フォーマットでの Acroform のエクスポート、インポート、およびデータ挿入が可能になりました。cfpdfform タグのアクション属性に fdfdata という新しい属性が追加されました。この属性では、エクスポートまたはインポートする PDF フォームの名前を指定します。

次の例に、read アクションを使用して PDF フォームを FDF フォーマットでエクスポートする方法を示します。

```
<cfpdfform source= "acroform_export.pdf" action="read" fdfdata="abc.fdf" >
</cfpdfform>
```

フォームデータをインポートするには :

```
<cfpdfform source= "write_acrroform.pdf" action="populate" fdfdata="abc.fdf" destination="hello.pdf">
</cfpdfform>
```

populate アクションを使用して、PDF ドキュメントにフォームデータを挿入できるようになりました。populate と組み合わせて新しい fdf 属性を使用すると、FDF フォーマットを内部的に使用できます。次のコード例で、この機能の使用方法を示します。

```
<cfpdfform source="acroform2.pdf" destinat ion="source_result1 7.pdf" action="populate"  overwrite="true"
fdf ="true">
    <cfpdfsubform  name="Text1">
    <cfpdf subform name="0">
    <cfp dformparam name="0" val ue="Test1.0.0">
    <cfpd fformparam name="1" value ="Test1.0.1">
    <cfpdfformparam name="2" value="Test1.0.2">
    </cfpdfsubform>
        <cfpdfsubform name="1">
        <cfpdfformparam name="0" value="Test1.1.0">
        <cfpdfformparam name="1" value="Test1.1.1">
        <cfpdfformparam name="2" value="Test1.1.2">
    </cfpdfsubform>

    </cfpdfsubform>
    <cfpdfsubform name="Text2">
    <cfpdfformparam name="0" value="Test2.0">
    <cfpdfformparam name="1" value="Test2.1">
    <cfpdfformparam name="2" value="Test2.2">
    <cfpdfformparam name="3" value="Test2.3">
    </cfpdfsubform><cfpdfformparam name="Text3" value="Test3">
    <cfpdfformparam name="Text4" value="Test4">
    <cfpdfformparam name="checkbox1" value="Yes">
    <cfpdfformparam name="listbox1" value="item4">
    <cfpdfformparam name="radiobutton1" value="2">
</cfpdfform>
```

アプリケーションの例

次の例では、cfpdf タグを使用して PDF ドキュメントを操作する簡単なアプリケーションを示します。

キーワード検索に基づいたドキュメントの結合

次の例では、getInfo と merge アクションを使用して、事業形態 (個人事業主、組合、小企業) に基づいて複数の税務ファイルから PDF ドキュメントを組み立てます。このアプリケーションでは、ラジオボタンの選択内容に基づいて、税務フォームと情報ドキュメントを組み立てます。一部の税務フォームや文書は、複数の事業形態に適用されます (たとえば、組合と小企業では、いずれも "f8825.pdf" 税務フォームを使用します)。PDF ドキュメントのキーワードを設定する手順については、915 ページの「PDF ドキュメント情報の管理」を参照してください。

この例では、次のタスクを行います。

- getInfo アクションを使用して、ディレクトリ内の PDF ファイルに対してキーワード検索を実行する
- 検索条件に一致したファイルのカンマ区切りリストを作成する
- merge アクションを使用して、カンマ区切りリストの PDF ドキュメントを 1 つの出力ファイルに結合する

最初の CFM ページでは、事業形態に基づいて税務文書を選択するフォームを作成します。

```
<h3>Downloading Federal Tax Documents</h3>
<p>Please choose the type of your business.</p>
<!--- Create the ColdFusion form to determine which PDF documents to merge. --->
<table>
<cfform action="cfpdfMergeActionTest.cfm" method="post">
    <tr><td><cfinput type="radio" name="businessType"
            Value="Sole Proprieter">Sole Proprietor</td></tr>
    <tr><td><cfinput type="radio" name="businessType"
            Value="Partnership">Partnership</td></tr>
    <tr><td><cfinput type="radio" name="businessType" Value="S Corporation">
            S Corporation</td></tr>
    <cfinput type = "hidden" name = "selection required" value = "must make a selection">
    <tr><td><cfinput type="Submit" name="OK" label="OK"></td></tr>
    </tr>
</cfform>
</table>
```

アクションページでは、"taxes" サブディレクトリ内のファイルをループし、getInfo アクションを使用して各ファイルのキーワードを取得します。PDF ファイルに事業形態のキーワード (Sole Proprietor、Partnership、または S Corporation) が含まれている場合は、そのファイルの絶対パスをカンマ区切りリストに追加します。merge アクションで、リスト内のファイルから出力 PDF ファイルを組み立てます。

```
<!--- Create a variable for the business type selected from the form. --->
<cfset bizType=#form.businessType#>
<!--- Create a variable for the path of the current directory. --->
<cfset thisPath=ExpandPath(".")>

<!--- List the files in the taxes subdirectory. --->
<cfdirectory action="list" directory="#thisPath#\taxes" name="filelist">

<!--- The following code loops through the files in the taxes subdirectory. The getInfo
    action to retrieves the keywords for each file and determines whether the business type
    matches one of the keywords in the file. If the file contains the business type keyword,
    ColdFusion adds the file to a comma-separated list. --->
<cfset tempPath="">
<cfloop query="filelist">
    <cfset fPath="#thisPath#\taxes\#filelist.name#">
    <cfpdf action="GetInfo" source="#fPath#" name="kInfo"></cfpdf>
    <cfif #kInfo.keywords# contains "#bizType#">
        <cfset tempPath=#tempPath# & #fPath# & ",">
    </cfif>
</cfloop>

<!--- Merge the files in the comma-separated list into a PDF output file called "taxMerge.pdf". --->
<cfpdf action="merge" source="#tempPath#" destination="taxMerge.pdf" overwrite="yes"/>

<h3>Assembled Tax Document</h3>
<p>Click the following link to view your assembled tax document:</p>
<a href="http://localhost:8500/Lion/taxmerge.pdf">
        <p>Your Assembled Tax Document</a></p>
```

DDX 命令によるドキュメントの作成

次の例では、DDX 命令と processddx アクションを使用してドキュメントを作成する方法について説明します。具体的には、次のタスクを行います。

- いくつかの PDF ドキュメントを 1 つの出力ファイルに結合する。
- 目次ページを生成して追加する。
- ヘッダとフッタを追加する。
- 自動ページ番号を追加する。

- ドキュメントの目次と本文に異なるスタイルを適用する。

次のコードは、DDX ファイルです。

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
    <PDF source="Doc0"/>
<TableOfContents maxBookmarkLevel="3" bookmarkTitle="Table of Contents"
    includeInTOC="false">
        <Header styleReference="TOCheaderStyle"/>
        <Footer styleReference="TOCFooterStyle"/>
    </TableOfContents>
    <PDFGroup>
        <Footer styleReference="FooterStyle"/>
        <PDF source="Doc1"/>
        <PDF source="Doc2"/>
        <PDF source="Doc3"/>
        <PDF source="Doc4"/>
    </PDFGroup>
    </PDF>

    <StyleProfile name="TOCheaderStyle">
    <Header>
        <Center>
            <StyledText>
            <p color="red" font-weight="bold" font="Arial">Table of Contents</p>
            </StyledText>
        </Center>
    </Header>
    </StyleProfile>

    <StyleProfile name="TOCFooterStyle">
    <Footer>
        <Right>
            <StyledText>
            <p font-size="9pt">Page <_PageNumber/> of <_LastPageNumber/></p>
            </StyledText>
        </Right>
    </Footer>
    </StyleProfile>

    <StyleProfile name="FooterStyle">
    <Footer>
        <Left>
            <StyledText>
            <p font-size="9pt"><i>CFML Reference</i></p>
            </StyledText>
        </Left>
        <Right>
            <StyledText>
            <p font-size="9pt">Page <_PageNumber/> of <_LastPageNumber/></p>
            </StyledText>
        </Right>
    </Footer>
    </StyleProfile>

</DDX>
```

次のコードは、DDX 命令を処理する ColdFusion ページです。

```
<cfif IsDDX("Book.ddx")>
    <cfset inputStruct=StructNew()>
    <cfset inputStruct.Doc0="Title.pdf">
    <cfset inputStruct.Doc1="Chap1.pdf">
    <cfset inputStruct.Doc2="Chap2.pdf">
    <cfset inputStruct.Doc3="Chap3.pdf">
    <cfset inputStruct.Doc4="Chap4.pdf">

    <cfset outputStruct=StructNew()>
    <cfset outputStruct.Out1="myBook.pdf">

    <cfpdf action="processddx" ddxfile="book.ddx" inputfiles="#inputStruct#"
        outputfiles="#outputStruct#" name="ddxVar">

    <cfoutput>#ddxVar.Out1#</cfoutput>
</cfif>
```

Acrobat で作成したフォームへの透かしの適用

次の例では、インタラクティブな Acrobat 税務フォームにあらかじめ値を設定し、ユーザーから送信された入力後のフォームにテキスト文字列の透かしを適用します。具体的には、次のタスクを行います。

- cfpdfform と cfpdfformparam タグを使用して、Acrobat で作成したフォームに値を設定する。
- cfpdfform タグを使用して、PDF の Post 送信の出力をファイルに書き込む。
- cfpdf の processddx アクションを使用して、入力後のフォームにテキスト文字列の透かしを適用する。

注意：この例では、cfdocexamples データベースと、1040 および 1040ez という連邦税務フォームを使用します。有効なユーザー名は "cpeterson" です。この例で使用している 1040 および 1040ez の IRS 税務フォームをダウンロードするには、IRS の Web サイトにアクセスしてください。フォームを (LiveCycle Designer ではなく) Acrobat で開いて送信ボタンを追加し、送信ボタンに ColdFusion 処理ページの URL を指定してください。また、非表示フィールドを追加し、入力後の税務フォームに使用する一意のファイル名が含まれる変数を設定してください。

最初の ColdFusion ページでは、ユーザー名と社会保障番号の入力を要求するログインフォームを作成します。

```
<!--- The following code creates a simple form for entering a user name and password. The
    code does not include password verification. --->

<h3>Tax Login Form</h3>
<p>Please enter your user name and your social security number.</p>
<cfform name="loginform" action="TaxFile2.cfm" method="post">
<table>
    <tr>
        <td>User name:</td>
        <td><cfinput type="text" name="username" required="yes"
                message="A user name is required."></td>
    </tr>
    <tr>
        <td>SSN#:</td>
        <td><cfinput type="text" name="SS1" maxLength="3" size="3"
            required="yes" mask="999"> -
        <cfinput type="text" name="SS2" maxLength="2" size="2" required="yes"
            mask="99"> -
        <cfinput type="text" name="SS3" maxLength="4" size="4" required="yes"
            mask="9999"></td>
    </tr>
</table>
    <br/>
    <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

2 番目の ColdFusion ページでは、cfdocexamples データベースからユーザー情報を取得します。また、使用可能な税務フォームをリストしたポップアップメニューを作成します。

```
<!--- The following code retrieves all of the employee information for the
            user name entered on the login page. --->
<cfquery name="getEmpInfo" datasource="cfdocexamples">
SELECT * FROM EMPLOYEES
WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<h3>Choose a tax form</h3>
<p>Hello <cfoutput>#getEmpInfo.firstname#</cfoutput>,</p>
<p>Please choose a tax form from the list:</p>
<!--- Create a pop-up menu with a list of tax forms. --->
<cfset thisPath=ExpandPath(".")>
<!--- Create a variable called filerID that is a combination of the username and the last
    three digits of the Social Security number. --->
<cfset filerID="#form.username#_#form.SS3#">
<cfdirectory action="list" name="taxForms" directory="#thisPath#/taxforms">
<cfform name="taxList" method="post" action="TaxFile3.cfm">
    <cfselect query="taxForms" value="name" size="10" required="yes" multiple="no"
        name="myTaxForm"/>
    <br/><br/>
    <cfinput type="Submit" name="OK" label="OK">
    <!--- Use hidden fields to pass the first name, last name, and the three parts of
        the SSN# to the tax form. Also, create a hidden field for the filerID variable. --->
    <cfinput type="hidden" name="FirstName" value="#getEmpInfo.FirstName#">
    <cfinput type="hidden" name="LastName" value="#getEmpInfo.LastName#">
    <cfinput type="hidden" name="Phone" value="#getEmpInfo.Phone#">
    <cfinput type="hidden" name="SS1" value="#form.SS1#">
    <cfinput type="hidden" name="SS2" value="#form.SS2#">
    <cfinput type="hidden" name="SS3" value="#form.SS3#">
    <cfinput type="hidden" name="taxFiler" value="#filerID#">
</cfform>
```

3 番目の ColdFusion ページでは、cfpdfform と cfpdfformparam タグを使用して、税務フォームにユーザーの情報を設定します。情報が設定された税務フォームがブラウザウィンドウで表示され、ユーザーはその他のフォームフィールドを入力できます。ユーザーが送信ボタンをクリックすると、Acrobat から ColdFusion 処理ページに入力後の PDF フォームが送信されます。

注意：フォームにあらかじめ値を設定するには、PDF フォームの各フィールド名と、cfpdfformparam タグの対応するデータ要素をマッピングしてください。フォームフィールドを確認するには、フォームを Acrobat Professional で開いて、[フォーム]-[Acrobat でフォームを編集] を選択します。フォームにあらかじめ値を設定する方法については、892 ページの「ColdFusion での PDF フォームの操作」を参照してください。

```
<!--- The following code populates the tax form template chosen from the list with
    information from the database query and the login form. Because no destination is
    specified, ColdFusion displays the interactive PDF form in the browser. A hidden field
    in the PDF form contains the name of the output file to write. It is a combination of
    the user name and the last three numerals of the user SSN#. The submit button added to
    the form created in Acrobat contains a URL to the ColdFusion processing page. --->

<cfpdfform source="taxForms/#form.myTaxForm#" action="populate">
    <cfif "taxForms/#form.myTaxForm#" is "taxForms/f1040.pdf">
        <cfpdfformparam name="f1_04(0)" value="#form.Firstname#">
        <cfpdfformparam name="f1_05(0)" value="#form.Lastname#">
        <cfpdfformparam name="f2_115(0)" value="#form.Phone#">
        <cfpdfformparam name="f1_06(0)" value="#form.SS1#">
        <cfpdfformparam name="f1_07(0)" value="#form.SS2#">
        <cfpdfformparam name="f1_08(0)" value="#form.SS3#">
        <cfpdfformparam name="filerID" value="#form.taxFiler#_1040">
    <cfelseif "taxForms/#form.myTaxForm#" is "taxForms/f1040ez.pdf">
        <cfpdfformparam name="f1_001(0)" value="#form.Firstname#">
        <cfpdfformparam name="f1_002(0)" value="#form.Lastname#">
        <cfpdfformparam name="f1_070(0)" value="#form.Phone#">
        <cfpdfformparam name="f1_003(0)" value="#form.SS1#">
        <cfpdfformparam name="f1_004(0)" value="#form.SS2#">
        <cfpdfformparam name="f1_005(0)" value="#form.SS3#">
        <cfpdfformparam name="filerID" value="#form.taxFiler#_1040ez">
    </cfif>
</cfpdfform>
```

4 番目の ColdFusion ページでは、cfpdfform タグを使用して、PDF の Post 送信の処理を実行し、出力ファイルを生成します。ファイル名は、税務フォームの非表示フィールドの値から生成されます。cfpdf タグの processddx アクションで、"watermark.ddx" ファイルの DDX 命令を使用し、テキスト文字列の透かしを生成してフォームに適用します。

次のコードは、"watermark.ddx" ファイルの内容です。

```
 <?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
    <PDF source="Doc1">
        <Watermark rotation="30" opacity="65%">
            <StyledText><p font-size="85pt" font-weight="bold" color="gray"
                font="Arial">FINAL</p></StyledText>
        </Watermark>
    </PDF>
    </PDF>
</DDX>

<!--- The following code reads the PDF file submitted in binary format and generates a result
    structure called fields. The cfpdfform populate action and the cfoutput tags reference
    the fields in the structure. --->
<cfpdfform source="#PDF.content#" action="read" result="fields"/>

<cfpdfform action="populate" source="#PDF.content#"
    destination="FiledForms\#fields.filerID#.pdf" overwrite="yes"/>

<!--- The following code verifies that the DDX file exists and the DDX instructions are
    valid. --->
<cfif IsDDX("watermark.ddx")>
    <!--- The following code uses the processddx action of the cfpdf tag to create a text-
        string watermark. --->
```

```
    <!--- This code creates a structure for the input files. --->
    <cfset inputStruct=StructNew()>
    <cfset inputStruct.Doc1="FiledForms\#fields.filerID#.pdf">

    <!--- This code creates a structure for the output file. --->
    <cfset outputStruct=StructNew()>
    <cfset outputStruct.Out1="FiledForms\#fields.filerID#.pdf">

    <!--- This code processes the DDX instructions and applies the watermark to the form. --->
    <cfpdf action="processddx" ddxfile="watermark.ddx" inputfiles="#inputStruct#"
outputfiles="#outputStruct#" name="Final">
</cfif>

<h3>Tax Form Completed</h3>
<p>Thank you for filing your tax form on line. Copy this URL to view or download your filed
    tax form:</p>
<cfoutput>
<a href="http://localhost:8500/Lion/FiledForms/#fields.filerID#.pdf">
    Link to your completed tax form</a>
</cfoutput>
```

ColdFusion イメージの作成と操作

Adobe ColdFusion では、イメージの作成と操作、データベース内のイメージの取得および保存、インデックス用および検索用のイメージ情報の取得、イメージの形式の変換、イメージのハードディスクへの書き込みを実行できます。

ColdFusion イメージについて

ColdFusion では、イメージを動的に作成して操作することができます。ColdFusion を使用すると、Adobe® Photoshop® などのイメージ作成ソフトウェアパッケージで手動で実行するさまざまなイメージ効果や描画関数を自動化したり、アプリケーションにイメージを組み込んだりすることができます。たとえば、Web サイトへの投稿者は、写真をさまざまな形式でアップロードすることができます。ColdFusion アプリケーションに数行のコードを追加するだけで、イメージを検証して標準のサイズと外観に再フォーマットし、変更後のイメージをデータベースに書き込み、イメージをブラウザに表示することができます。

次の表に、ColdFusion イメージに対して行えるタスクの一部を示します。

タスク	**関数およびアクション**
ColdFusion 変数がイメージを返すかどうか検証する	IsImage 関数
ファイルが有効なイメージであるかどうか検証する	IsImageFile 関数
サムネールイメージを作成する	ImageScaleToFit 関数、ImageResize 関数、または cfimage タグの resize アクション
透かしを作成する	ImageSetDrawingTransparency 関数と、任意の ImageDraw 関数および ImagePaste 関数
イメージに関する情報を取得する (たとえば、サイズ制限を適用する場合など)	ImageGetHeight と ImageGetWidth 関数、または ImageInfo 関数
JPEG イメージの圧縮を適用する	cfimage タグの write アクションの quality 属性、または ImageWrite 関数
イメージのファイル形式を変換する (たとえば、BMP ファイルを JPEG に変換するなど)	cfimage タグ、または ImageRead と ImageWrite 関数
イメージファイルを Base64 文字列に変換する	cfimage タグまたは ImageWriteBase64 関数

タスク	関数およびアクション
Base64 文字列から ColdFusion イメージを作成する	ImageReadBase64 関数
ColdFusion イメージを BLOB (Binary Large Object Bitmap) としてデータベースに挿入する	cfquery ステートメント内の ImageGetBlob 関数
データベース内の BLOB からイメージを作成する	cfimage タグ、または ImageNew 関数と cfquery ステートメント
バイナリオブジェクトからイメージを作成する	cffile タグでイメージファイルをバイナリオブジェクトに変換し、このバイナリオブジェクトを ImageNew 関数に渡す
CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) イメージを作成する	cfimage タグの captcha アクション

ColdFusion イメージ

ColdFusion イメージは、ColdFusion 特有の構造形式です。ColdFusion イメージには、ソースから読み込んだイメージデータが含まれます。ソースには、イメージファイルか、ColdFusion イメージ変数で表される別の ColdFusion イメージを指定できます。ColdFusion イメージ変数を使用すると、メモリ内で情報を動的に操作することができます。オプションで、ColdFusion イメージをファイル、データベース列、またはブラウザに書き込むこともできます。

cfimage タグ

cfimage タグでは、ColdFusion イメージを作成できます。また、イメージのサイズ変更、境界線の追加、ファイル形式の変換などの、よく実行するイメージ操作も行えます。cfimage タグは、単独で使用することも、Image 関数と併用することもあります。cfimage タグで作成した ColdFusion イメージを Image 関数に渡して、複雑なイメージ操作を行うことができます。

次の表に、cfimage タグのアクションを示します。

アクション	説明
border	イメージの周囲に矩形の境界線を作成します。
captcha	CAPTCHA イメージを作成します。
convert	イメージのファイル形式を変換します。
info	イメージに関する情報 (イメージのカラーモデル、高さ、幅、ソースなど) が含まれた ColdFusion 構造体を作成します。
read	指定のローカルファイルパスまたは URL からイメージを読み込みます。アクションを明示的に指定しなかった場合は、read がデフォルト値として使用されます。
resize	イメージの高さおよび幅を変更します。
rotate	イメージを角度単位で回転します。
write	イメージをファイルに書き込みます。write アクションを使用して、低解像度の JPEG ファイルを生成することもできます。また、write アクションを使用して、イメージを別のファイル形式 (PNG や GIF など) に変換することもできます。
writeToBrowser	イメージを直接ブラウザに書き込みます。このアクションを使用すると、イメージをファイルに保存しなくても、単一のイメージの外観を確認したり、複数のイメージをブラウザに書き込むことができます。

詳細については、『CFML リファレンス』の cfimage タグを参照してください。

Image 関数

ColdFusion には、cfimage タグの機能を拡張する 50 以上の Image 関数が用意されています。cfimage タグで作成したイメージを Image 関数に渡したり、ImageNew 関数でイメージを作成したりすることができます。次の表に、Image 関数をカテゴリ別に示します。

カテゴリ	Image 関数
イメージとサポートされているイメージ形式の確認	IsImage、IsImageFile、GetReadableImageFormats、GetWriteableImageFormats
イメージ情報の取得	ImageGetEXIFTag、ImageGetHeight、ImageGetIPTCTag、ImageGetWidth、ImageInfo
イメージの読み込み、書き込み、変換	ImageGetBlob、ImageGetBufferedImage、ImageNew、ImageRead、ImageReadBase64、ImageWrite、ImageWriteBase64
イメージの操作	ImageAddBorder、ImageBlur、ImageCopy、ImageCrop、ImageFlip、ImageGrayscale、ImageNegative、ImageOverlay、ImagePaste、ImageResize、ImageRotate、ImageScaleToFit、ImageSharpen、ImageShear、ImageTranslate
線、シェイプ、テキストの描画	ImageDrawArc、ImageDrawBeveledRect、ImageDrawCubicCurve、ImageDrawLine、ImageDrawLines、ImageDrawOval、ImageDrawPoint、ImageDrawQuadraticCurve、ImageDrawRect、ImageDrawRoundRect、ImageDrawText
描画制御の設定	ImageClearRect、ImageRotateDrawingAxis、ImageSetAntialiasing、ImageSetBackgroundColor、ImageSetDrawingColor、ImageSetDrawingStroke、ImageSetDrawingTransparency、ImageShearDrawingAxis、ImageTranslateDrawingAxis、ImageXORDrawingMode

ColdFusion イメージの作成

ColdFusion イメージは、メモリ内にイメージのデータが保持されています。ColdFusion でイメージを操作するには、まず ColdFusion イメージを作成します。次の表に、ColdFusion イメージを作成する方法を示します。

タスク	関数およびタグ
既存のイメージファイルから ColdFusion イメージを作成する	cfimage タグまたは ImageNew 関数
空の新規イメージを作成する	ImageNew 関数
データベース内の BLOB データから ColdFusion イメージを作成する	ImageNew 関数と cfquery タグ
バイナリオブジェクトから ColdFusion イメージを作成する	cffile タグと ImageNew 関数
Base64 文字列から ColdFusion イメージを作成する	ImageReadBase64 関数と ImageNew 関数、または cfimage タグ
別の ColdFusion イメージから ColdFusion イメージを作成する	ImageCopy 関数と ImageWrite 関数、または Duplicate 関数、またはイメージを ImageNew 関数か cfimage タグに渡す

cfimage タグの使用

cfimage タグで最も簡単に ColdFusion イメージを作成するには、source 属性 (ColdFusion に読み込むイメージファイル) と、name 属性 (メモリ内のイメージを定義する変数) を指定します。

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" name="myImage">
```

read アクションはデフォルトのアクションであるため、指定する必要はありません。read アクションでは name 属性を指定します。この名前を使用して、ColdFusion イメージを格納する変数 (**myImage** など) が作成されます。

myImage 変数は、別の cfimage タグや Image 関数に渡すことができます。次の例では、ソースとして ColdFusion イメージ変数を指定しています。

```
<cfimage source="#myImage#" action="write" destination="test_myImage.jpg">
```

write アクションは、指定の宛先にファイルを書き込みます。宛先は、絶対パスか相対パスで指定できます。次の例では、URL から ColdFusion イメージを作成し、このイメージをローカルストレージドライブのファイルに書き込んでいます。

```
<cfimage source="http://www.google.com/images/logo_sm.gif" action="write"
     destination="c:\images\logo_sm.gif">
```

write アクションでは、宛先を指定します。

同じファイルへの書き込みを複数回行う場合は、destination を指定するときに、overwrite 属性に "yes" を設定します。そうしなかった場合は、エラーが発生します。

```
<cfimage source="#myImage#" action="write" destination="images/jeff01.jpg" overwrite="yes">
```

ImageNew 関数の使用

ImageNew 関数で ColdFusion イメージを作成する方法は、ColdFusion 変数を定義する方法と同じです。次の例では、"jeff01.jpg" ソースファイルから、"myImage" という ColdFusion イメージ変数を作成します。

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
```

このコードで生成される結果は、次のコードの結果と同じです。

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" name="myImage">
```

cfimage タグと同様に、ソースとして絶対パス、相対パス、URL、または別の ColdFusion イメージを指定できます。次の例では、ローカルドライブのファイルを読み込んで ImageWrite 関数に渡し、この関数でイメージを新しいファイルに書き込みます。

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<cfset ImageWrite(myImage,"myImageTest.png")>
```

次のコードでも、同じ結果が生成されます。

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" name="myImage">
<cfimage source="#myImage#" action="write" destination="myImageTest.png">
```

また、空のイメージを作成することもできます。ImageNew 関数でソースを指定しないと、空のイメージが作成されます。ただし、幅と高さは指定できます。次の例では、幅 300 ピクセル、高さ 200 ピクセルの空のキャンバスを作成します。

```
<cfset myImage=ImageNew("",300,200)>
```

オプションで、次の例のようにイメージタイプを指定することもできます。

```
<cfset myImage=ImageNew("",200,300,"rgb")>
```

その他の有効なイメージタイプは、argb と grayscale です。空のイメージは、ColdFusion の描画関数のキャンバスとして使用できます。例については、947 ページの「透かしの作成」を参照してください。

また、ImageNew 関数を使用して、他のソース (Base64 バイト配列、ファイルパス、URL) から ColdFusion イメージを作成することもできます。次の例では、JPEG ファイルから ColdFusion イメージ (x) を作成し、さらにメモリ内のこのイメージから別の ColdFusion イメージ (y) を作成します。

```
<cfset x = ImageNew("c:\abc.jpg")>
<cfset y = ImageNew(x)>
```

ImageNew 関数の詳細については、『CFML リファレンス』を参照してください。

バイナリオブジェクトからのイメージの作成

cffile タグを使用して、イメージファイルを ColdFusion 変数に書き込むことができます。その後に、この変数を ImageNew 関数に渡して、このバイナリオブジェクトから ColdFusion イメージを作成できます。次に例を示します。

```
<!--- Use the cffile tag to read an image file, convert it to binary format, and write the
    result to a variable. --->
<cffile action = "readBinary" file = "jeff05.jpg" variable = "aBinaryObj">
<!--- Use the ImageNew function to create a ColdFusion image from the variable. --->
<cfset myImage=ImageNew(aBinaryObj)>
```

BLOB データからのイメージの作成

多くのデータベースでは、イメージを BLOB データとして保存しています。データベースから BLOB データを抽出するには、cfquery タグを使用してクエリーを作成します。次の例では、BLOB データを抽出し、cfimage タグを使用して PNG 形式のファイルに書き込みます。

```
<!--- Use the cfquery tag to retrieve employee photos and last names from the database. --->
<cfquery
    name="GetBLOBs" datasource="myblobdata">
    SELECT LastName,Image
    FROM Employees
</cfquery>
<cfset i = 0>
<table border=1>
    <cfoutput query="GetBLOBs">
    <tr>
        <td>
        #LastName#
        </td>
        <td>
        <cfset i = i+1>
<!--- Use the cfimage tag to write the images to PNG files. --->
        <cfimage source="#GetBLOBs.Image#" destination="employeeImage#i#.png"
            action="write">
        <img src="employeeImage#i#.png"/>
        </td>
    </tr>
</cfoutput>
</table>
```

次の例では、ImageNew 関数を使用して、BLOB データから ColdFusion イメージを生成しています。

```
<!--- Use the cfquery tag to retrieve all employee photos and employee IDs from a database. --->
<cfquery name="GetBLOBs" datasource="myBlobData">
SELECT EMLPOYEEID, PHOTO FROM Employees
</cfquery>
<!--- Use the ImageNew function to create a ColdFusion images from the BLOB data that was
    retrieved from the database. --->
<cfset myImage = ImageNew(GetBLOBs.PHOTO)>
<!--- Create thumbnail versions of the images by resizing them to fit in a 100-pixel square,
    while maintaining the aspect ratio of the source image. --->
<cfset ImageScaleToFit(myImage,100,100)>
<!--- Convert the images to JPEG format and save them to files in the thumbnails subdirectory,
    using the employee ID as the filename. --->
<cfimage source="#myImage#" action="write"
    destination="images/thumbnails/#GetBLOBs.EMPLOYEID#.jpg">
```

ColdFusion イメージを BLOB に変換する方法については、941 ページの「BLOB でのデータベースへのイメージの挿入」を参照してください。

Base64 文字列からのイメージの作成

Base64 は、バイナリデータを ASCII 文字の文字列として記述する方式です。データベースによっては、イメージを BLOB データではなく、Base64 形式で保存している場合があります。cfimage タグまたは ImageReadBase64 関数を使用して、データベースから直接 Base64 データを読み込むことができます。この方法を使用すると、バイナリのエンコードとデコードを行う中間処理が不要になります。

次の例では、cfimage タグを使用して、Base64 文字列から ColdFusion イメージを作成しています。

```
<!--- This example shows how to create a ColdFusion image from a Base64 string with headers
    (used for display in HTML). --->
<cfimage source="............."
    destination="test_my64.jpeg" action="write" isBase64="yes">

<!--- This example shows how to use the cfimage tag to write a Base64 string without headers. --->
<cfimage source="/9j/4AAQSkZJRgABAQA............." destination="test_my64.jpeg"
    action="write" isBase64="yes">
```

次の例では、ImageReadBase64 関数を使用して、Base64 文字列から ColdFusion イメージを作成しています。

```
<!--- This example shows how to use the ImageReadBase64 function to read a Base64 string
    with headers. --->
<cfset myImage=ImageReadBase64(".............")>

<!--- This example shows how to read a Base64 string without headers. --->
<cfset myImage=ImageReadBase64("/9j/4AAQSkZJRgABAQA.............")>
```

Base64 文字列の詳細については、941 ページの「Base64 文字列へのイメージの変換」を参照してください。

イメージのコピー

ImageCopy 関数を使用すれば、既存のイメージの矩形領域をコピーして、新しい ColdFusion イメージを生成できます。新しい ColdFusion イメージは、別のイメージの上にペーストしたり、ファイルに書き込むことができます。次に例を示します。

```
<!--- Use the cfimage tag to create a ColdFusion image from a JPEG file.
--->
<cfimage source="../cfdocs/images/artgallery/lori05.jpg" name="myImage">
<!--- Turn on antialiasing to improve image quality. --->
<cfset ImageSetAntialiasing(myImage)>
<!--- Copy the rectangular area specified by the coordinates (25,25,50,50) in the image to
    the rectangle beginning at (75,75), and return this copied rectangle as a new ColdFusion
    image. --->
<cfset dupArea = ImageCopy(myImage,25,25,50,50,75,75)>
<!--- Write the new ColdFusion image (dupArea) to a PNG file. --->
<cfimage source="#dupArea#" action="write" destination="test_myImage.png" overwrite="yes">
```

イメージの複製

ColdFusion イメージを作成する別の方法として、イメージの複製があります。イメージを複製すると、クローンが作成されます。クローンはイメージのコピーですが、元のイメージからは独立しているので、元のイメージが変更されても、その変更はクローンには影響しません。その逆についても同様です。この方法は、同じイメージの別バージョンを作成する場合に便利です。イメージを複製すると、処理時間を短縮できます。これは、ColdFusion イメージの作成時に、データベースやファイルからイメージデータを一度しか取得しなくて済むからです。その後、クローンを複数作成したり、メモリ内のクローンを操作してから、ファイルに書き込むことができます。たとえば、サーバーにアップロードされているイメージのサムネール版やグレースケール版、拡大版を作成する場合を考えます。この場合は、cfimage タグまたは ImageNew 関数を使用して、アップロードされているファイルから ColdFusion イメージを作成します。その後、Duplicate 関数を使用して、ColdFusion イメージのクローンを 3 つ作成します。

クローンを作成するには、ColdFusion イメージ変数を Duplicate 関数に渡します。

```
<!--- Use the ImageNew function to create a ColdFusion image from a JPEG file. --->
<cfset myImage=ImageNew("../cfdocs/images/artgallery/paul01.jpg")>
<!--- Turn on antialiasing to improve image quality. --->
<cfset ImageSetAntialiasing(myImage)>
<!--- Use the Duplicate function to create three clones of the ColdFusion image. --->
<cfset cloneA=Duplicate(myImage)>
<cfset cloneB=Duplicate(myImage)>
<cfset cloneC=Duplicate(myImage)>
<!--- Create a grayscale version of the image. --->
<cfset ImageGrayscale(cloneA)>
<!--- Create a thumbnail version of the image. --->
<cfset ImageScaleToFit(cloneB,50,"")>
<!--- Create an enlarged version of the image. --->
<cfset ImageResize(cloneC,"150%","")>
<!--- Write the images to files. --->
<cfset ImageWrite(myImage,"paul01.jpg","yes")>
<cfset ImageWrite(cloneA,"paul01_bw.jpg","yes")>
<cfset ImageWrite(cloneB,"paul01_sm.jpg","yes")>
<cfset ImageWrite(cloneC,"paul01_lg.jpg","yes")>
<!--- Display the images. --->
<img src="paul01.jpg">
<img src="paul01_bw.jpg">
<img src="paul01_sm.jpg">
<img src="paul01_lg.jpg">
```

また、次の例のように、cfimage タグと ImageNew 関数を使用しても、イメージを複製できます。

```
<!--- Use the cfimage tag to create a ColdFusion image (myImage) and make a copy of it (myImageCopy). --->
<cfimage source="../cfdocs/images/artgallery/paul01.jpg" name="myImage">
<cfimage source="#myImage#" name="myImageCopy">
<!-- Use the ImageNew function to make a copy of myImage called myImageCopy2. --->
<cfset myImageCopy2 = ImageNew(myImage)>
```

イメージの変換

ColdFusion では、イメージのファイル形式を簡単に変換できます。また、イメージファイルをバイナリオブジェクト、BLOB データ、または Base64 文字列に変換することもできます。

イメージファイルの変換

イメージのファイル形式は、宛先ファイルの拡張子によって決まります。したがって、イメージを変換するには、単純に宛先ファイルのファイル名拡張子を変更します。次の例では、JPEG ファイルを GIF ファイルに変換しています。

<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" action="write" destination="jeff01.gif">

同様に、ImageWrite 関数と ImageNew 関数を使用することもできます。

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<cfset ImageWrite(myImage,"jeff01.gif")>
```

いずれの例でも、convert アクションが暗黙的に処理されます。

write アクションは ColdFusion イメージを作成するのではなく、単にイメージをファイルに書き込みます。イメージを変換して ColdFusion イメージ変数を生成するには、convert アクションを使用します。

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" action="convert"
    destination="jeff01.gif" name="myImage">
```

ColdFusion では、ほとんどの標準的なイメージ形式の読み取りと書き込みがサポートされています。詳細については、『CFML リファレンス』の「サポートされるイメージファイル形式」を参照してください。

Base64 文字列へのイメージの変換

ColdFusion イメージを Base64 文字列に変換するには、ImageWriteBase64 関数を使用します。次の例では、yes の値によって、HTML での表示に必要なヘッダを出力に含めるように指定しています。

```
<!--- This example shows how convert a BMP file to a Base64 string. --->
<cfset ImageWriteBase64(myImage,"jeffBase64.txt","bmp","yes")>
```

注意：Microsoft Internet Explorer では、Base64 文字列がサポートされていません。

BLOB でのデータベースへのイメージの挿入

多くのデータベースでは、イメージを BLOB データとして保存しています。ColdFusion イメージをデータベースの BLOB 列に挿入するには、次の例のように、cfquery ステートメントで ImageGetBlob 関数を使用します。

```
<!--- This example shows how to add a ColdFusion image to a BLOB column of a database. --->
<!--- Create a ColdFusion image from an existing JPEG file. --->
<cfimage source="aiden01.jpg" name="myImage">
<!--- Use the cfquery tag to insert the ColdFusion image as a BLOB in the database. --->
<cfquery name="InsertBlobImage" datasource="myBlobData">
INSERT into EMPLOYEES (FirstName,LastName,Photo)
VALUES ("Aiden","Quinn",<cfqueryparam value="#ImageGetBlob(myImage)#" cfsqltype="cf_sql_blob">)
</cfquery>
```

イメージの検証

IsImage 関数を使用すると、イメージ変数が有効な ColdFusion イメージを表しているかどうかを確認できます。この関数は、唯一のパラメータとして変数名を取り、ブール値を返します。

注意：IsImage 関数を使用して、ファイルが有効なイメージであるかどうかを検証することはできません。この場合は、IsImageFile 関数を使用します。

また、ColdFusion には、GetReadableImageFormats および GetWriteableImageFormats の 2 つのタグが用意されています。これらを使用すれば、ColdFusion アプリケーションがデプロイされているサーバーでサポートされているイメージファイル形式を確認できます。詳細については、『CFML リファレンス』を参照してください。

サイズの制限

ColdFusion には、イメージに関する情報 (イメージの高さや幅など) を取得する関数がいくつか用意されています。たとえば、ImageGetWidth や ImageGetHeight 関数を使用すると、Web サイトやデータベースにアップロードするのにイメージが大きすぎないかどうかを確認できます。

次の例では、幅 300 ピクセルまたは高さ 300 ピクセルより大きいイメージはアップロードできないようにしています。

```
<!--- Create a ColdFusion image named "myImage" from a file uploaded to the server. --->
<cfimage action="read" source="#fileUpload.serverFile#" name="myImage">
<!--- Determine whether the file is greater than 300 pixels in width or height. --->
<cfif ImageGetHeight(myImage) gt 300 or ImageGetWidth(myImage) gt 300>
    <!--- If the file exceeds the size limits, delete it from the server. --->
    <cffile action="delete" file="#fileUpload.serverDirectory#/#fileUpload.serverFile#">
    <cfoutput>
<!--- Display the following message. --->
    <p>
    The image you uploaded was too big. It must be less than 300 pixels wide and 300 pixels
        high. Your image was #imageGetWidth(myImage)# pixels wide and
        #imageGetHeight(myImage)# pixels high.
    </p>
</cfif>
```

イメージのメタデータを取得する方法の詳細については、『CFML リファレンス』の ImageGetEXIFTag、ImageGetIPTCTag、および ImageInfo 関数を参照してください。

JPEG イメージの圧縮

大きなファイルのサイズを縮小するには、cfimage タグの write アクションを使用して、JPEG ファイルを低画質のイメージに変換します。quality 属性には、0 (低) ～ 1 (高) の値を指定します。次に例を示します。

```
<cfimage source="../cfdocs/images/artgallery/jeff05.jpg" action="write"
    destination="jeff05_lq.jpg" quality="0.5" overwrite="yes">
```

ImageWrite 関数を使用しても、同じ処理を実行できます。

```
<cfset myImage=ImageNew("jeff05.jpg")>
<cfset ImageWrite(myImage,"jeff05_lq.jpg","0.5")>
```

ColdFusion イメージの操作

ColdFusion イメージでは、いくつかの一般的な操作を実行できます。ColdFusion イメージの操作の詳細については、『CFML リファレンス』を参照してください。

イメージへの境界線の追加

単純な境界線を作成するには、cfimage タグを使用します。次の例では、5 ピクセルの青色の境界線が付いた ColdFusion イメージを作成します。

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" action="border" thickness="5"
    color="blue" destination="testMyImage.jpg" overwrite="yes">
<img src="testMyImage.jpg">
```

境界線は、ソースイメージの周囲に追加されます。これによって、イメージの領域が広がります。

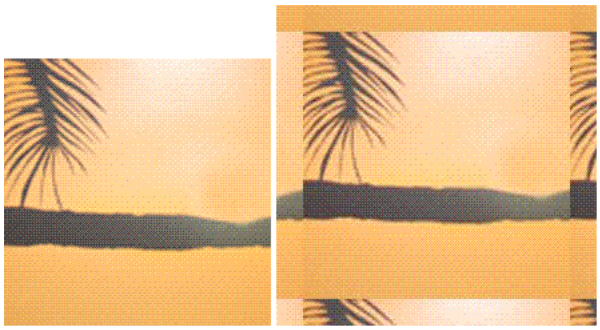
複雑な境界線を作成するには、ImageAddBorder 関数を使用します。次の例では、境界線をネストしています。

```
<!--- Create a ColdFusion image from a JPEG file. --->
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<!--- Add a 5-pixel blue border around the outside edge of the image. --->
<cfset ImageAddBorder(myImage,5,"blue")>
<!--- Add a 10-pixel magenta border around the blue border. --->
<cfset ImageAddBorder(myImage,10,"magenta")>
<!--- Add a 5-pixel green border around the magenta border. --->
<cfset ImageAddBorder(myImage,20,"green")>
<!--- Write the ColdFusion image to a file. --->
<cfset ImageWrite(myImage,"testMyImage.jpg")>
<img src="testMyImage.jpg"/>
```

また、ImageAddBorder 関数では、境界線によるイメージ効果も生成できます。たとえば、wrap パラメータを使用すると、ソースイメージからタイル状の境界線を作成できます。wrap パラメータによって、指定されたピクセル数がイメージの各辺に追加されて、タイル状の境界線が作成されます。その結果、イメージがタイル状に表示されます。

次の例では、ソースイメージの周囲から 20 ピクセルをタイル状にして、境界線を作成します。

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff03.jpg")>
<cfset ImageAddBorder(myImage,20,"","wrap")>
<cfset ImageWrite(myImage,"testMyImage.jpg")>
<img src="testMyImage.jpg"/>
```



その他の境界線タイプの例については、『CFML リファレンス』の ImageAddBorder 関数を参照してください。

テキストイメージの作成

次の 2 種類のテキストイメージを作成できます。

- CAPTCHA イメージ。ColdFusion によって、テキストがランダムに変形されます。
- テキストイメージ。テキストの属性を制御できます。

CAPTCHA イメージの作成

cfimage タグの captcha アクションを使用して、変形したテキストイメージを作成します。このテキストイメージは、人間には判読できますが、機械的には判読できません。CAPTCHA イメージを作成するには、CAPTCHA イメージで表示するテキストを指定します。このテキストが ColdFusion によってランダムに変形されます。テキスト領域の高さと幅 (文字の間隔に影響)、フォントサイズ、CAPTCHA テキストに使用するフォント、難易度 (読みやすさに影響) を指定できます。text 属性で指定するテキスト文字列に、スペースは使用しないでください。ユーザーが CAPTCHA イメージを見ても、スペースが含まれていることは認識できません。

次の例では、CAPTCHA イメージを直接ブラウザに書き込んでいます。

```
<!--- This example shows how to create a CAPTCHA image with the text "rEadMe" and write the
    image directly to the browser. --->
<cfimage action="captcha" fontSize="25" width="162" height="75" text="rEadMe"
    fonts="Verdana,Arial,Courier New,Courier">
```

注意：CAPTCHA イメージを表示するには、width を、"fontSize × text で指定した文字の数 × 1.08" より大きい値に設定する必要があります。この例の場合、width の最小値は 162 です。

ColdFusion 9 では、PNG 形式の CAPTCHA イメージのみサポートされます。

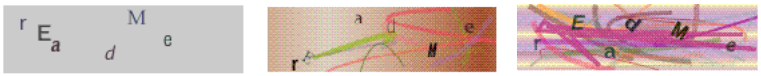
注意：destination 属性を指定して CAPTCHA イメージをファイルに書き込む場合は、複数のユーザーが CAPTCHA イメージにアクセスしてもファイルが上書きされないように、CAPTCHA イメージファイルに一意の名前を使用してください。

次の例では、テキストの変形レベルが高い CAPTCHA イメージを作成しています。

```
<!--- Use the GetTickCount function to generate unique names for the CAPTCHA files. --->
<cfset tc = GetTickCount()>
<!--- Set the difficulty to "high" for a higher level of text distortion. --->
<cfimage action="captcha" fontSize="15" width="180" height="50" text="rEadMe"
    destination="readme#tc#.png" difficulty="high">
```

詳細な例については、952 ページの「CAPTCHA を使用した会員登録の検証」を参照してください。

次の図に、low、medium、high の難易度 (difficulty) の 3 つの CAPTCHA イメージを示します。



ImageDrawText 関数の使用

ImageDrawText 関数を使用してテキストイメージを作成する場合は、テキスト文字列と、その始点の x 座標と y 座標を指定します。次の例のように、テキストを既存のイメージや空のイメージの上に描画することができます。

```
<!--- This example shows how to draw a text string on a blank image. --->
<cfset myImage=ImageNew("",200,100)>
<cfset ImageDrawText(myImage, "Congratulations!",10,50)>
<cfimage source="#myImage#" action="write" destination="myImage.png" overwrite="yes">
<img src="myImage.png">

<!--- This example shows how to draw a text string on an existing image.
--->
<cfset myImage2=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<cfset ImageDrawText(myImage2,"Congratulations!",10,50)>
<cfimage source="#myImage2#" action="write" destination="myImage2.png" overwrite="yes">
<img src="myImage2.png">
```

前の例の場合、テキストは、デフォルトのシステムフォントとフォントサイズで表示されます。テキストの外観を制御するには、次の例のように、テキスト属性のコレクションを指定します。

```
<cfset attr = StructNew()>
<cfset attr.style="bolditalic">
<cfset attr.size=20>
<cfset attr.font="verdana">
<cfset attr.underline="yes">
```

テキスト文字列にテキスト属性を適用するには、ImageDrawText 定義に属性のコレクション名を含めます。次の例では、"attr" テキスト属性コレクションをテキスト文字列 "Congratulations!" に適用しています。

```
...
<cfset ImageDrawText(myImage,"Congratulations!",10,50,attr)>
```

テキストの色を変更するには、ImageSetDrawingColor 関数を使用します。この関数は、それ以降にイメージに描画するすべてのオブジェクトの色を制御します。次の例では、"Congratulations!" と "Gabriella" の 2 行のテキストに、マゼンタの色が継承されます。

```
<!--- This example shows how to draw a text string on a blank image. --->
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<cfset ImageSetDrawingColor(myImage,"magenta")>
<cfset attr = StructNew()>
<cfset attr.style="bolditalic">
<cfset attr.size=20>
<cfset attr.font="verdana">
<cfset attr.underline="yes">
<cfset ImageDrawText(myImage,"Congratulations!",10,50,attr)>
<cfset ImageDrawText(myImage,"Gabriella",50,125,attr)>
<cfimage source="#myImage#" action="write" destination="myImage.jpg" overwrite="yes">
<img src="myImage.jpg"/>
```

有効な色の名前のリストについては、『CFML リファレンス』の cfimage タグを参照してください。

線とシェイプの描画

ColdFusion には、線とシェイプを描画する関数がいくつか用意されています。シェイプの場合、最初の 2 つの値は、シェイプの左上隅の x 座標と y 座標を表します。単純な楕円や矩形の場合、その座標の後の 2 つの数値は、シェイプの幅と高さ (ピクセル単位) を表します。線の場合、値は、線の始点と終点の x 座標と y 座標をそれぞれ表します。塗りつぶしたシェイプを作成する場合は、filled 属性を true に設定します。次の例では、いくつかの描画オブジェクトが含まれたイメージを作成しています。

```
<!--- Create an image that is 200-pixels square. --->
<cfset myImage=ImageNew("",200,200)>
<!--- Draw a circle that is 100 pixels in diameter. --->
<cfset ImageDrawOval(myImage,40,20,100,100)>
<!--- Draw a filled rectangle that is 40 pixels wide and 20 pixels high.
--->
<cfset ImageDrawRect(myImage,70,50,40,20,true)>
<!--- Draw a 100-pixel square. --->
<cfset ImageDrawRect(myImage,40,40,100,100)>
<!--- Draw two lines. --->
<cfset ImageDrawLine(myImage,130,40,100,200)>
<cfset ImageDrawLine(myImage,50,40,100,200)>
<!--- Write the ColdFusion image to a file. --->
<cfimage source="#myImage#" action="write" destination="testMyImage.gif" overwrite="yes">
<img src="testMyImage.gif"/>
```

注意：複数のつながった線を描画するには、ImageDrawLines 関数を使用します。詳細については、『CFML リファレンス』を参照してください。

描画制御の設定

ColdFusion には、描画オブジェクトの外観を制御する関数がいくつか用意されています。ImageDrawText の例で示したように、ImageSetDrawingColor 関数を使用して、イメージ内のテキストの色を定義します。この関数は、線とシェイプの色も制御します。線の属性 (色以外) を制御するには、ImageSetDrawingStroke 関数を使用します。ImageSetDrawingStroke 関数では、コレクションを使用して線の属性を定義します。

描画制御は、イメージ内のそれ以降のすべての描画関数に適用されるので、順序が重要になります。次の例では、描画する線の属性を属性コレクションで定義し、正方形と 2 本の線に適用します。同様に、緑色を矩形と正方形に適用し、赤色を 2 本の線のみに適用します。描画制御はイメージ内で何回でも必要なだけリセットして、必要な効果を得ることができます。

```
<!--- Create an attribute collection for the drawing stroke. --->
<cfset attr=StructNew()>
<cfset attr.width="4">
<cfset attr.endcaps="round">
<cfset attr.dashPattern=ArrayNew(1)>
<cfset dashPattern[1]=8>
<cfset dashPattern[2]=6>
<cfset attr.dashArray=dashPattern>
<cfset myImage=ImageNew("",200,200)>
<cfset ImageDrawOval(myImage,40,20,100,100)>
<!--- Set the drawing color to green for all subsequent drawing functions. --->
<cfset ImageSetDrawingColor(myImage,"green")>
<cfset ImageDrawRect(myImage,70,50,40,20,true)>
<!--- Apply the attribute collection to all subsequent shapes and lines in the image. --->
<cfset ImageSetDrawingStroke(myImage,attr)>
<cfset ImageDrawRect(myImage,40,40,100,100)>
<!--- Set the drawing color to red for all subsequent drawing functions. --->
<cfset ImageSetDrawingColor(myImage,"red")>
<cfset ImageDrawLine(myImage,130,40,100,200)>
<cfset ImageDrawLine(myImage,50,40,100,200)>
<cfimage source="#myImage#" action="write" destination="testMyImage.gif" overwrite="yes">
<img src="testMyImage.gif"/>
```

イメージのサイズ変更

ColdFusion では、簡単にイメージのサイズを変更できます。イメージのサイズを変更してファイルサイズを縮小したり、複数のイメージに均一のサイズを適用したり、サムネールイメージを作成することができます。次の表に、ColdFusion でイメージのサイズを変更する方法を示します。

タスク	関数およびアクション
イメージのサイズを変更する	ImageResize 関数、または cfimage タグの resize アクション
指定の正方形または矩形に収まるようにイメージのサイズを変更し、補間法を制御する	ImageScaleToFit 関数
イメージのサイズを変更し、補間法を制御する	ImageResize 関数

cfimage タグの resize アクションの使用

cfimage タグの resize アクションを使用して、イメージの高さと幅を指定の値に変更します。高さと幅は、ピクセル単位またはイメージの元のサイズのパーセントで指定できます。パーセントで指定する場合は、高さおよび幅の定義でパーセント記号 (%) を付加します。

```
<!--- This example shows how to specify the height and width of an image in pixels. --->
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" action="resize" width="100"
    height="100" destination="jeff01_sm.jpg">
<!--- This example shows how to specify the height and width of an image as percentages. --->
<cfimage source="../cfdocs/images/artgallery/jeff02.jpg" action="resize" width="50%"
    height="50%" destination="jeff02_sm.jpg">

<!--- This example shows how to specify the height of an image in pixels and its width as a
    percentage. Notice that this technique can distort the image. --->
<cfimage source="../cfdocs/images/artgallery/jeff03.jpg" action="resize" width="50%"
    height="100" destination="jeff03_sm.jpg" overwrite="yes">
```

cfimage タグの resize アクションでは、幅と高さの両方を指定する必要があります。

cfimage タグの resize アクションでは、highestQuality 補間方式が使用されます。イメージの画質は最高になりますが、パフォーマンスは低くなります。表示を高速化する場合は、ImageResize 関数または ImageScaleToFit 関数を使用します。

ImageResize 関数の使用

ImageResize 関数は、cfimage タグの resize アクションに似ています。サイズ変更後のイメージで縦横比を維持するには、高さまたは幅の値を指定し、もう一方のサイズには空の値を入力します。

```
<!--- This example shows how to resize an image to 50% of original size and resize it
    proportionately to the new width. The height value is blank. --->
<cfset myImage=ImageNew("http://www.google.com/images/logo_sm.gif")>
<cfset ImageResize(myImage,"50%","")>
<!--- Save the modified image to a file. --->
<cfimage source="#myImage#" action="write" destination="test_myImage.jpeg" overwrite="yes">
<!--- Display the source image and the resized image. --->
<img src="http://www.google.com/images/logo_sm.gif"/>
<img src="test_myImage.jpeg"/>
```

ImageResize 関数では、イメージのサイズ変更に使用する補間法を指定できます。補間法を指定することで、パフォーマンスと画質のバランスを調整できます。デフォルトでは、ImageResize 関数では highestQuality 補間法が使用されます。画質を落としてパフォーマンスを向上させるには、補間法を変更します。また、イメージのぼかし度を設定できます。デフォルト値は 1 (ぼかしなし) です。ぼかし度の最高は 10 (非常にぼかす) です。次の例では、highPerformance の補間法とぼかし度 10 を使用して、イメージのサイズを変更しています。

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/aiden01.jpg")>
<cfset ImageResize(myImage,"","200%","highPerformance", 10)>
<cfimage action="writeToBrowser" source="#myImage#">
```

注意：ぼかし度を高くすると、パフォーマンスは低下します。

すべての補間法のリストについては、『CFML リファレンス』の ImageResize を参照してください。

ImageScaleToFit 関数の使用

サムネールイメージやフォトギャラリーで表示するイメージなど、均一のサイズのイメージを作成するには、ImageScaleToFit 関数を使用します。イメージの領域は、ピクセル単位で指定します。イメージはこの正方形または矩形に収まるようにサイズが変更され、ソースイメージの縦横比は維持されます。ImageResize 関数と同じく、補間法を指定できます。次に例を示します。

```
<!--- This example shows how to resize an image to a 100-pixel square, while maintaining the aspect ratio
of the source image. --->
<cfimage source="../cfdocs/images/artgallery/jeff05.jpg" name="myImage" action="read">
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(myImage)>
<cfset ImageScaleToFit(myImage,100,100,"mediumQuality")>
<!--- Display the modified image in a browser. --->
<cfimage source="#myImage#" action="writeToBrowser">
```

指定の矩形領域にイメージを収めるには、次の例のように矩形の幅と高さを指定します。

```
<!--- This example shows how to resize an image to fit in a rectangle that is 200 pixels
    wide and 100 pixels high, while maintaining the aspect ratio of the source image. --->
<cfimage source="../cfdocs/images/artgallery/jeff05.jpg" name="myImage">
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(myImage)>
<cfset ImageScaleToFit(myImage,200,100)>
<!--- Display the modified image in a browser. --->
<cfimage source="#myImage#" action="writeToBrowser">
```

この例では、処理後のイメージの幅は 200 ピクセル以下、高さは 100 ピクセル以下になります。

また、矩形の高さのみ、または幅のみを指定することもできます。これを行うには、定義しない次元に空の文字列を指定します。次の例を実行すると、イメージの幅は常に 200 ピクセルになり、高さは幅に比例して決められます。

```
<!--- This example shows how to resizes an image so that it is 200 pixels wide, while
    maintaining the aspect ratio of the source image. The interpolation method is set to
    maximize performance (which reduces image quality). --->
<cfimage source="../cfdocs/images/artgallery/jeff05.jpg" name="myImage">
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(myImage)>
<cfset ImageScaleToFit(myImage,200,"","highestPerformance")>
<!--- Display the modified image in a browser. --->
<cfimage source="#myImage#" action="writeToBrowser">
```

詳細については、『CFML リファレンス』の ImageScaleToFit を参照してください。

透かしの作成

透かしは、別のイメージに重ね合わせる半透明のイメージです。透かしの使用例としては、著作権のあるイメージを保護する場合などがあります。ColdFusion で透かしを作成するには、ImageSetDrawingTransparency 関数と ImagePaste 関数を使用します。透かしイメージは、次の 3 つのいずれかの方法で作成できます。

- 既存のイメージファイルから透かしを作成する。たとえば、会社のロゴを透かしとして使用できます。
- ColdFusion でテキストイメージを作成し、透かしとして適用する。たとえば、**Copyright** や **PROOF** などのテキスト文字列を作成し、それをフォトギャラリー内のすべてのイメージに適用できます。
- ColdFusion で描画イメージを作成し、透かしとして使用する。たとえば、描画関数を使用して緑色のチェックマークを作成し、このチェックマークを承認済みのイメージに適用することができます。

イメージファイルからの透かしの作成

次の例では、Web サイトにある既存の GIF イメージから透かしを作成しています。

```
<!--- This example shows how to create a watermark from an existing image. --->
<!--- Create two ColdFusion images from existing JPEG files. --->
<cfimage source="../cfdocs/images/artgallery/raquel05.jpg" name="myImage">
<cfimage source="http://www.google.com/images/logo_sm.gif" name="myImage2">
<cfimage source="#myImage#" action="write" destination="logo.jpg" overwrite="yes">
<cfset ImageSetDrawingTransparency(myImage,50)>
<!--- Paste myImage2 on myImage at the coordinates (0,0). --->
<cfset ImagePaste(myImage,myImage2,0,0)>
<!--- Write the result to a file. --->
<cfimage source="#myImage#" destination="watermark.jpg" action="write" overwrite="yes">
<!--- Display the result. --->
<img src="watermark.jpg"/>
```

テキストイメージからの透かしの作成

次の例では、ColdFusion でテキストイメージを作成し、透かしに使用しています。

```
<!--- Create a ColdFusion image from an existing JPG file. --->
<cfset myImage=ImageNew("../cfdocs/images/artgallery/raquel05.jpg")>
<!--- Scale the image to fit in a 200-pixel square, maintaining the aspect ratio of the
    source image. --->
    <cfset ImageScaleToFit(myImage,200,200)>
<!--- Set the drawing transparency to 75%. --->
    <cfset ImageSetDrawingTransparency(myImage,75)>
<!--- Create a ColdFusion image from scratch. --->
<cfset textImage=ImageNew("",150,140)>
<!--- Set the drawing color to white. --->
    <cfset ImageSetDrawingColor(textImage,"white")>
<!--- Create a collection of text attributes. --->
        <cfset attr=StructNew()>
        <cfset attr.size=40>
        <cfset attr.style="bold">
        <cfset attr.font="Arial">
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(textImage)>
<!--- Draw the text string "PROOF" on the ColdFusion image. Apply the text attributes that
    you specified. --->
<cfset ImageDrawText(textImage,"PROOF",1,75,attr)>
<!--- Rotate the text image by 30 degrees. --->
    <cfset ImageRotate(textImage,30)>
<!--- Scale the image to fit in a 200-pixel square, maintaining the aspect ratio of the
    source image. --->
    <cfset ImageScaleToFit(textImage,200,200)>
<!--- Paste the text image onto myImage. --->
<cfset ImagePaste(myImage,textImage,0,0)>
<!--- Write the combined image to a file. --->
<cfimage source="#myImage#" action="write" destination="test_watermark.jpg" overwrite="yes">
<!--- Display the image. --->
<img src="test_watermark.jpg"/>
```

ColdFusion の描画からの透かしの作成

次の例では、ColdFusion でイメージを描画し、それを透かしとして使用しています。ImageSetDrawingStroke 関数を使用して、描画関数で作成する線とシェイプの属性を定義し、ImageSetDrawingColor 関数を使用して色を定義します。

```
<!--- This example shows how to draw a red circle with a line through it and use it as a
    watermark. --->
<!--- Use the ImageNew function to create a ColdFusion image that is 201x201 pixels. --->
<cfset myImage=ImageNew("",201,201)>
<!--- Set the drawing transparency of the image to 30%. --->
<cfset ImageSetDrawingTransparency(myImage,30)>
<!--- Set the drawing color to red. --->
<cfset ImageSetDrawingColor(myImage,"red")>
<!--- Create an attribute collection that sets the line width to ten pixels. --->
    <cfset attr=StructNew()>
    <cfset attr.width = 10>
<!--- Apply the attribute collection to the ImageSetDrawingStroke function. --->
    <cfset ImageSetDrawingStroke(myImage,attr)>
<!--- Draw a diagonal line starting at (40,40) and ending at (165,165) on myImage. The drawing
    attributes you specified are applied to the line. --->
    <cfset ImageDrawLine(myImage,40,40,165,165)>
<!--- Draw a circle starting at (5,5) and is 190 pixels high and 190 pixels wide. The drawing
    attributes you specified are applied to the oval. --->
<cfset ImageDrawOval(myImage,5,5,190,190)>
<!--- Create a ColdFusion image from a JPEG file. --->
<cfimage source="../cfdocs/images/artgallery/raquel05.jpg" name="myImage2">
<!--- Scale the image to fit in a 200-pixel square, maintaining the aspect ratio of the
    source image. --->
<cfset ImageScaleToFit(myImage2,200,200)>
<!--- Paste the myImage2 directly over the myImage. --->
<cfset ImagePaste(myImage,myImage2,0,0)>
<!--- Save the combined image to a file. --->
<cfimage source="#myImage#" action="write" destination="test_watermark.jpg" overwrite="yes">
<!--- Display the image in a browser. --->
<img src="test_watermark.jpg"/>
```

ブラウザへのイメージの書き込み

cfimage タグの writeToBrowser アクションを使用すると、ファイルにイメージを書き込むことなく、ブラウザに直接表示することができます。この方法は、ColdFusion イメージの外観を確認する場合に便利です。次の例では、イメージに適用した 2 つの効果の結果を確認しています。

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/paul01.jpg")>
<cfset ImageBlur(myImage,5)>
<cfset ImageNegative(myImage)>
<cfimage source="#myImage#" action="writeToBrowser">
```

writeToBrowser アクションでは、イメージが PNG 形式で表示されます。

また、複数のイメージをブラウザに書き込むこともできます。これは、メモリ内のイメージを操作して、ファイルに書き込まずに表示したい場合に役立ちます。たとえば、同じイメージのいくつかのバージョンを作成してブラウザに表示し、その中からファイルに書き込むイメージをユーザーに選択させることができます。または、データベースからイメージを取得し、ブラウザに表示するイメージに " 校正 " や " ドラフト " などの透かしを追加することもできます。この場合も、変更したイメージをファイルに書き込むことなく表示できます。この方法を使用すれば、一連のイメージファイルを保持して、変更後のイメージを直接表示することができます。複数のイメージをブラウザに書き込む例については、951 ページの「透かしの付いたイメージギャラリーの生成」を参照してください。

ColdFusion イメージを使用したアプリケーションの例

ColdFusion イメージを使用して、イメージの処理を自動化する簡単なアプリケーションを作成できます。

サムネールイメージの生成

次の例では、イメージをアップロードするフォームを作成します。サイトの訪問者はこのフォームを使用して、イメージファイルをアップロードし、イメージファイルからサムネールイメージを生成できます。ColdFusion イメージの処理を使用して、次のタスクを行います。

- アップロードされたファイルが有効なイメージであるかどうか検証する。
- イメージの高さと幅が 800 ピクセルを超えていないことを確認する。
- イメージが有効で、サイズ制限内である場合、ソースイメージのサムネール版を生成してファイルに保存する。

フォームページに次のコードを入力します。

```
<!--- This code creates a form with one field where the user enters the image file to upload. --->
<cfform action="makeThumbnail.cfm" method="post" enctype="multipart/form-data">
Please upload an image: <cfinput type="file" name="image">
<cfinput type="submit" value="Send Image" name="Submit">
</cfform>
```

アクションページに次のコードを入力します。

```
<cfset thisDir = expandPath(".")>
<!--- Determine whether the form is uploaded with the image. --->
<cfif structKeyExists(form,"image") and len(trim(form.image))>
    <!--- Use the cffile tag to upload the image file. --->
    <cffile action="upload" fileField="image" destination="#thisDir#" result="fileUpload"
        nameconflict="overwrite">
    <!--- Determine whether the image file is saved. --->
    <cfif fileUpload.fileWasSaved>
    <!--- Determine whether the saved file is a valid image file. --->
        <cfif IsImageFile("#fileUpload.serverfile#")>
    <!--- Read the image file into a variable called myImage. --->
            <cfimage action="read" source="#fileUpload.serverfile#" name="myImage">
            <!--- Determine whether the image file exceeds the size limits. --->
            <cfif ImageGetHeight(myImage) gt 800 or ImageGetWidth(myImage) gt 800>
                <!--- If the file is too large, delete it from the server. --->
                <cffile action="delete"
                    file="#fileUpload.serverDirectory#/#fileUpload.serverFile#">
                <cfoutput>
                <p>
                The image you uploaded was too large. It must be less than 800 pixels wide
                    and 800 pixels high. Your image was #imageGetWidth(myImage)# pixels wide
                    and #imageGetHeight(myImage)# pixels high.
                </p>
                </cfoutput>
                <!--- If the image is valid and does not exceed the size limits,
                    create a thumbnail image from the source file that is 75-pixels
                    square, while maintaining the aspect ratio of the source image.
                    Use the bilinear interpolation method to improve performance.
                    --->
            <cfelse>

    <cfset ImageScaleToFit(myImage,75,75,"bilinear")>
                <!--- Specify the new filename as the source filename with
                    "_thumbnail" appended to it. --->
                <cfset newImageName = fileUpload.serverDirectory & "/" &
                    fileUpload.serverFilename & "_thumbnail." &
                    fileUpload.serverFileExt>
                <!--- Save the thumbnail image to a file with the new filename. --->
                <cfimage source="#myImage#" action="write"
                    destination="#newImageName#" overwrite="yes">
                <cfoutput>
                <p>
```

```
                Thank you for uploading the image. We have created a thumbnail for
                    your picture.
                </p>
                <p>
                <!--- Display the thumbnail image. --->
                <img src="#getFileFromPath(newImageName)#">
                </p>
                </cfoutput>
            </cfif>
        <!--- If it is not a valid image file, delete it from the server. --->
        <cfelse>
            <cffile action="delete"
                file="#fileUpload.serverDirectory#/#fileUpload.serverFile#">
            <cfoutput>
            <p>
            The file you uploaded, #fileUpload.clientFile#, was not a valid image.
            </p>
            </cfoutput>
        </cfif>
    </cfif>
</cfif>
```

透かしの付いたイメージギャラリーの生成

次の例では、cfartgallery データベースからイメージと情報を抽出します。ColdFusion イメージの処理を使用して、次のタスクを行います。

- データベースから返されたレコードのイメージが存在することを確認する。
- 売却済みのイメージの上に、**SOLD!** というテキストを表示する。
- ソースイメージの縦横比を維持したまま、イメージのサイズを 100 ピクセルに変更する。
- イメージに 5 ピクセルの境界線を追加する。
- 変更後のイメージを、ファイルに書き込まず、ブラウザに直接表示します。

例

```
<!--- Create a query to extract artwork and associated information from the cfartgallery database. --->
<cfquery name="artwork" datasource="cfartgallery">
SELECT FIRSTNAME, LASTNAME, ARTNAME, DESCRIPTION, PRICE, LARGEIMAGE, ISSOLD, MEDIATYPE
FROM ARTISTS, ART, MEDIA
WHERE ARTISTS.ARTISTID = ART.ARTISTID
AND ART.MEDIAID = MEDIA.MEDIAID
ORDER BY ARTNAME
</cfquery>
<cfset xctr = 1>
<table border="0" cellpadding="15" cellspacing="0" bgcolor="#FFFFFF">
<cfoutput query="artwork">
    <cfif xctr mod 3 eq 1>
    <tr>
        </cfif>
        <!--- Use the IsImageFile function to verify that the image files extracted
            from thedatabase are valid. Use the ImageNew function to create a
            ColdFusion image fromvalid image files. --->
        <cfif IsImageFile("../cfdocs/images/artgallery/#artwork.largeImage#")>
        <cfset myImage=ImageNew("../cfdocs/images/artgallery/#artwork.largeImage#")>
            <td valign="top" align="center" width="200">
            <cfset xctr = xctr + 1>
            <!--- For artwork that has been sold, display the text string "SOLD!"
                in white on the image. --->
                    <cfif artwork.isSold>
```

```
                        <cfset ImageSetDrawingColor(myImage,"white")>
                            <cfset attr=StructNew()>
                            <cfset attr.size=45>
                            <cfset attr.style="bold">
                            <cfset ImageDrawText(myImage,"SOLD!",35,195, attr)>
                            </cfif>
        <!--- Resize myImage to fit in a 110-pixel square, scaled proportionately. --->
        <cfset ImageScaleToFit(myImage,110,"","bicubic")>
        <!--- Add a 5-pixel black border around the images. (Black is the default color. --->
        <!--- Add a 5-pixel black border to myImage. --->
        <cfset ImageAddBorder(myImage,"5")>
        <!--- Write the images directly to the browser without saving them to the hard drive.
            --->
        <cfimage source="#myImage#" action="writeToBrowser"><br>
            <strong>#artwork.artName#</strong><br>
            Artist: #artwork.firstName# #artwork.lastName#<br>
            Price: #dollarFormat(artwork.price)#<br>
            #artwork.mediaType# - #artwork.description#<br>
            </td>
        </cfif>
        <cfif xctr-1 mod 3 eq 0>
            </tr>
        </cfif>
</cfoutput>
</table>
```

CAPTCHA を使用した会員登録の検証

次の例では、オンラインニュースレターの受信登録をしているのがユーザーであるかどうか (コンピュータで自動生成されたスパムでないかどうか) を検証する簡単なフォームを作成します。フォームページでランダムなテキスト文字列から CAPTCHA イメージを生成し、アクションページで応答を確認します。

例

フォームページに次のコードを入力します。

```
<!--- Set the length of the text string for the CAPTCHA image. --->
<cfset stringLength=6>
<!--- Specify the list of characters used for the random text string. The following list
    limits the confusion between upper- and lowercase letters as well as between numbers and
    letters. --->
<cfset
    stringList="2,3,4,5,6,7,8,9,a,b,d,e,f,g,h,j,n,q,r,t,y,A,B,C,D,E,F,G,H,K,L,M,N,P,Q,R,S,
    T,U,V,W,X,Y,Z">
<cfset rndString="">
<!--- Create a loop that builds the string from the random characters. --->
<cfloop from="1" to="#stringLength#" index="i">
    <cfset rndNum=RandRange(1,listLen(stringList))>
    <cfset rndString=rndString & listGetAt(stringList,rndNum)>
</cfloop>
<!--- Hash the random string. --->
<cfset rndHash=Hash(rndString)>

<!--- Create the user entry form. --->
<cfform action="captcha2.cfm" method="post">
<p>Please enter your first name:</p>
<cfinput type="text" name="firstName" required="yes">
<p>Please enter your last name:</p>
<cfinput type="text" name="lastName" required="yes">
<p>Please enter your e-mail address:</p>
<cfinput type="text" name="mailTo" required="yes" validate="email">
<!--- Use the randomly generated text string for the CAPTCHA image. --->
<p><cfimage action="captcha" fontSize="24" fonts="Times New Roman" width="200" height="50"
    text="#rndString#"></p>
<p>Please type what you see: </p>
<p><cfinput type="text" name="userInput" required="yes" maxlength=6>
<cfinput type="hidden" name="hashVal" value="#rndHash#">
<p><cfinput type="Submit" name ="OK" value="OK"></p>
</cfform>
```

アクションページに次のコードを入力します。

```
<!--- Verify whether the text entered by the user matches the CAPTCHA string. --->
<cfif #form.hashval# eq Hash(#form.userInput#)>
    <cfoutput>
    <p>
    Thank you for registering for our online newsletter, #form.firstName# #form.lastName#.
    </p>
    <p>
    A notification has been sent to your e-mail address: #form.mailTo#.
    </p>
        <cfmail from="newsletter@domain.com" to="#form.mailTo#" subject="Newsletter">
        Thank you for your interest in our Newsletter.
        </cfmail>
    </cfoutput>
<cfelse>
    <p>I'm sorry; please try again.</p>
</cfif>
```

イメージの別バージョンの作成

次の例では、同じイメージの別バージョンを 4 つ生成してフォームに表示し、いずれか 1 つを選択して保存するアプリケーションを作成します。このアプリケーションは、次のタスクを行う 3 つの ColdFusion ページで構成されます。

- データベースクエリーからドロップダウンリストをダイナミックに設定する。

- cfimage タグを使用して、リストで選択されたタイトルに基づいて ColdFusion イメージを作成する。ここでは、ImageNew 関数を使用して、ColdFusion イメージのクローンを 3 つ作成します。ImageSharpen 関数を使用して、各クローンのシャープネス設定を変更します。
- フォームで選択されたファイルを新しい場所に保存する。

例

最初のフォームページでは、cfartgallery データベースからアートワークを抽出するクエリーを作成し、ポップアップメニューにタイトルを表示します。

```
<!--- Create a query to extract artwork from the cfartgallery database. --->
<cfquery name="artwork" datasource="cfartgallery">
SELECT ARTID, ARTNAME, LARGEIMAGE
FROM ART
ORDER BY ARTNAME
</cfquery>

<!--- Create a form that lists the artwork titles generated by the query. Set the value to
    LARGEIMAGE so that the image file is passed to the action page. --->
<cfform action="dupImage2.cfm" method="post">
<p>Please choose a title:</p>
<cfselect name="art" query="artwork" display="ARTNAME" value="LARGEIMAGE" required="yes"
    multiple="no" size="8">
</cfselect>
<br/><cfinput type="submit" name="submit" value="OK">
</cfform>
```

最初のアクションページでは、元のイメージのクローンを 3 回作成し、各クローンのシャープネス設定を変更して結果を表示します。

```
<!--- Determine whether a valid image file exists. --->
<cfif IsImageFile("../cfdocs/images/artgallery/#form.art#")>
    <cfset original=ImageNew("../cfdocs/images/artgallery/#form.art#")>
    <!--- Use the ImageNew function to create a clone of the ColdFusion image. --->
    <cfset clone1=ImageNew(original)>
    <!--- Use the ImageSharpen function to blur the cloned image. --->
    <cfset ImageSharpen(clone1,-1)>
    <!--- Use the ImageNew function to create a second clone of the original image. --->
    <cfset clone2=ImageNew(original)>
    <!--- Use the ImageSharpen function to sharpen the cloned image. --->
    <cfset ImageSharpen(clone2,1)>
    <!--- Use the ImageNew function to create a third clone for the original image. --->
    <cfset clone3=ImageNew(original)>
    <!--- Use the ImageSharpen function to sharpen the cloned image to the maximum setting.
        --->
    <cfset ImageSharpen(clone3,2)>
<!--- Create a form with a radio button for each selection. The value of the hidden field
    is the relative path of the original image file. --->
<p>Please choose an image:</p>
<table>
    <tr>
```

```
    <cfform action="dupImage3.cfm" method="post">
        <td><cfimage source="#original#" action="writeToBrowser"><br />
        <cfinput type="radio" name="foo" value="original">Original Image</td>
        <td><cfimage source="#clone1#" action="writeToBrowser"><br />
        <cfinput type="radio" name="foo" value="blurred">Blurred Image</td>
        <td><cfimage source="#clone2#" action="writeToBrowser"><br />
        <cfinput type="radio" name="foo" value="sharper">Sharper Image</td>
        <td><cfimage source="#clone3#" action="writeToBrowser"><br />
        <cfinput type="radio" name="foo" value="sharpest">Sharpest Image</td>
        <tr><td><cfinput type="Submit" name="OK" value="OK">
        <cfinput type="hidden" name="orig_file"
            value="../cfdocs/images/artgallery/#form.art#">
        </td></tr>
    </cfform>
    </tr>
</table>
<cfelse>
    <p>There is no image associated with the title you selected. Please click the Back button
        and try again.</p>
</cfif>
```

2 番目のアクションページでは、選択されたイメージを C ドライブに保存します。

```
<p>The image you have chosen has been saved.</p>
<cfset img=ImageNew("#form.orig_file#")>
<cfswitch expression=#form.foo#>
    <cfcase value="blurred">
        <cfset ImageSharpen(img,-1)>
    </cfcase>
    <cfcase value="sharper">
        <cfset ImageSharpen(img,1)>
    </cfcase>
    <cfcase value="sharpest">
        <cfset ImageSharpen(img,2)>
    </cfcase>
</cfswitch>

<!--- Use the cfimage tag to write the image selected from the form to a file in the C drive.
    Use the value of the hidden field as the source file for the image. --->
<cfimage source="#img#" action="write" destination="c:/myImage.jpg" overwrite="yes">
<img src="c:/myImage.jpg" />
```

チャートとグラフの作成

cfchart タグを使用して、チャートやグラフを表示できます。

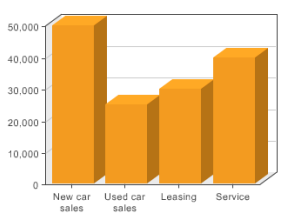
チャートについて

データをチャートやグラフで表現すると、データの持つ意味が明瞭に把握できるようになります。表を作成して数値データを並べるだけでなく、棒グラフ、円グラフ、折れ線グラフなどのチャートを作成して、データをわかりやすく表示できます。これらのチャートでは、カラー、キャプション、データの 2 次元表現や 3 次元表現などが使用できます。

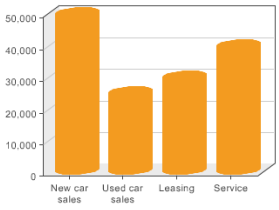
cfchart タグと cfchartseries タグや cfchartdata タグを併用すると、さまざまな種類のチャートを提供できます。これらのタグの属性によって、チャートの外観をカスタマイズできます。

Adobe ColdFusion では、11 種類の 2D または 3D のチャートを作成できます。次の図に、各タイプのチャートのサンプルを示します。

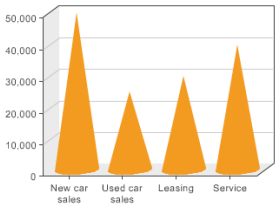
注意：棒グラフと円柱 (シリンダー) グラフは 2D では同じように見えます。円錐グラフと角錐 (ピラミッド) グラフも同様です。



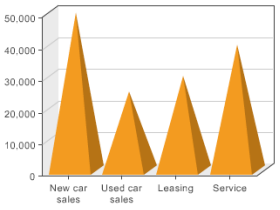
棒



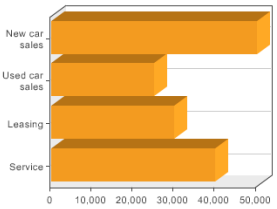
円柱



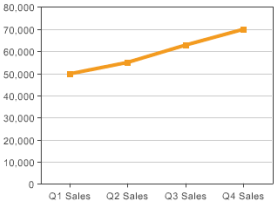
円錐



角錐

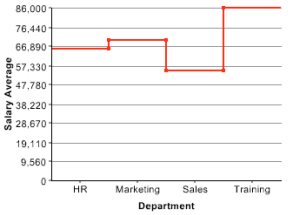


横棒

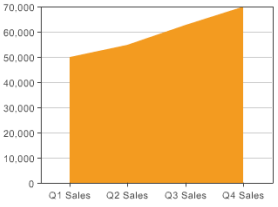


折れ線

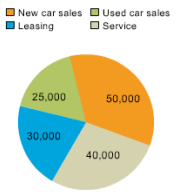
曲線



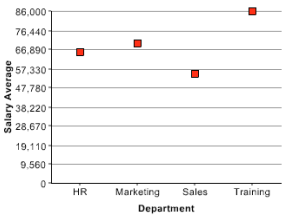
ステップ



面積



円



散布

基本的なチャートの作成

次のいずれかの方法でチャートを作成できます。

- ColdFusion ページで cfchart タグ、cfchartseries タグ、および cfchartdata タグを使用する
- ColdFusion Report Builder のチャートウィザードを使用する。詳細については、979 ページの「印刷用のレポートとドキュメントの作成」を参照してください。

ColdFusion タグによるチャートの作成

ColdFusion タグを使用してチャートを作成するには、cfchart タグと最低 1 つの cfchartseries タグを使用します。必要に応じて、cfchartseries タグ内に cfchartdata タグを記述できます。次の表で、各タグについて説明します。

タグ	説明
cfchart	チャートを表示するコンテナを指定します。このコンテナで、チャートの高さ、幅、背景カラー、ラベル、フォントなどの特性を定義します。 cfchart タグの中には、最低 1 つの cfchartseries タグを記述します。
cfchartseries	チャートにデータを供給するデータベースクエリー、および個々のデータポイントを記述する cfchartdata タグを指定します。チャートタイプ、チャートのカラー、その他の必要な属性を指定します。
cfchartdata	必要に応じて、cfchartseries タグの個々のデータポイントを指定します。

次の例は、チャートを作成する基本的なコードのアウトラインです。

```
<cfchart>
    <cfchartseries type="type">
            <cfchartdata item="something" value="number">
    </cfchartseries>
</chart>
```

次の例では、4 つの値を表すシンプルな円グラフを表示します。

```
<cfchart>
    <cfchartseries type="pie">
        <cfchartdata item="New car sales" value="50000">
        <cfchartdata item="Used car sales" value="25000">
        <cfchartdata item="Leasing" value="30000">
        <cfchartdata item="Service" value="40000">
    </cfchartseries>
</cfchart>
```

注意：Java 呼び出しを使用してチャート作成機能にアクセスする場合は、チャートイメージの上に「Developer 版 - 本稼働用ではありません」という透かしが表示されることがあります。これを防ぐには、サーバーオブジェクトを参照するときに、svr.getDefaultInstance(getPageContext().getServletContext());) ではなく、先頭に svr= を追加して **svr=**svr.getDefaultInstance(getPageContext().getServletContext());) のようなコードを使用します。変更を有効にするためにサーバーを再起動します。

Report Builder ウィザードによるチャートの作成

ColdFusion Report Builder には、チャートを簡単に作成できるウィザードが用意されています。cfchart タグ、cfchartseries タグ、および cfchartdata タグで指定できるすべてのチャート特性を、このウィザードで設定できます。Report Builder チャートウィザードの使用方法の詳細については、979 ページの「印刷用のレポートとドキュメントの作成」を参照してください。

チャートの作成

データからチャートを作成するときの重要なタスクの 1 つが、cfchart タグにデータを供給する方法の選択です。データの供給は次の方法で行うことができます。

- cfchartdata タグを使用して個々のデータポイントを指定する。
- cfchartseries タグを使用してすべてのデータを単独のクエリーで提供する。
- クエリーデータと cfchartdata タグのデータポイントを組み合わせる。
- Report Builder で作成したレポートのすべてのデータを提供する。詳細については、979 ページの「印刷用のレポートとドキュメントの作成」を参照してください。

注意：cfchart タグで作成できるのは、数値データのチャートのみです。したがって、日付、時刻、形式設定済みの通貨 ($3,000.53 など) は、整数か実数に変換してください。

個別のデータポイントによるチャート作成

個別のデータポイントを指定してチャートを作成するには、cfchartseries タグ本文に cfchartdata タグを挿入して、各データポイントを指定します。たとえば次のコードでは、シンプルな円グラフが作成されます。

```
<cfchart>
    <cfchartseries type="pie">
        <cfchartdata item="New Vehicle Sales" value=500000>
        <cfchartdata item="Used Vehicle Sales" value=250000>
        <cfchartdata item="Leasing" value=300000>
        <cfchartdata item="Service" value=400000>
    </cfchartseries>
</cfchart>
```

この円グラフでは、自動車ディーラーの 4 種類の収益が表示されます。それぞれの cfchartdata タグで、各部門の収入と凡例用の説明を記述しています。

注意：2 つのデータポイントに同じ項目名が指定されている場合は、cfchart タグ内で最後に指定されているポイントの値を使用してグラフが作成されます。

cfchartdata タグでは、データポイントに関する次の情報を指定します。

属性	説明
value	チャート作成に使用するデータ値。この属性は必須です。
item	(オプション) このデータポイントの説明。このアイテムは、棒グラフまたは折れ線グラフの水平軸、横棒グラフの垂直軸、および円グラフの凡例に表示されます。

クエリーによるチャート作成

チャートのそれぞれのバー、ドット、線、スライスは、結果セット内の 1 つの行座標および列座標からのデータを表します。関連する一連のデータのことを、チャート系列 (chart series) と呼びます。

1 つの棒、点、線、またはスライスは、2 つの軸の交点を表します。したがって、チャートに表示したときに行と列の値が何らかの意味を持つように、クエリーの結果セットを作成します。それには、多くの場合、クエリーデータを集計する必要があります。通常、クエリーデータを集計するには次のいずれかの方法を使用します。

- SELECT ステートメントで GROUP BY 節を使用し、SQL 集計関数 (SUM、AVG、MAX など) を指定する。
- クエリーオブクエリーを使用する。
- テーブルではなくビューからデータを取得する。

クエリーからチャートを作成する場合は、cfchartseries タグの query 属性を使用してクエリー名を指定します。たとえば、シンプルな棒グラフのコードは次のようになります。

```
<cfchart
        xAxisTitle="Department"
        yAxisTitle="Salary Average"
    >
    <cfchartseries
        type="bar"
        query="DataTable"
        valueColumn="AvgByDept"
        itemColumn="Dept_Name"
        />
</cfchart>
```

この例では、DataTable クエリーの AvgByDept 列の値を表示しています。Dept_Name 列の値はアイテムラベルとして表示しています。

次の表に、クエリーを使用する場合の cfchartseries タグの属性を示します。

属性	説明
query	データが含まれているクエリー。valueColumn と itemColumn も指定します。
valueColumn	チャートに使用する値が含まれているクエリー列。
itemColumn	データポイントの説明が含まれているクエリー列。このアイテムは通常、棒グラフまたは折れ線グラフの水平軸、横棒グラフの垂直軸、および円グラフの凡例に表示されます。

クエリーオブクエリーによるチャート作成

クエリーの結果からだけでなく、クエリーオブクエリーの結果からもチャートを作成できます。クエリーオブクエリーの使用方法の詳細については、433 ページの「クエリーオブクエリーの使用」を参照してください。クエリーオブクエリーは、チャート用のデータを作成するときに非常に役立ちます。たとえば、データベースクエリーに SUM、AVG、GROUP BY などの集計関数を適用して、統計データを含んだクエリーオブクエリーを作成できます。詳細については、433 ページの「クエリーオブクエリーの使用」を参照してください。

また、クエリーデータをダイナミックに参照して必要な修正を加えることもできます。たとえば、クエリー列のエントリ全体をループして、整数のドル値に形式を設定し直すことができます。

次の例では、クエリーオブクエリーを使用して cfdocexamples データベースの給与データを分析し、データを棒グラフで表示しています。

1 次の内容の ColdFusion ページを作成します。

```
<!--- Get the raw data from the database. --->
<cfquery name="GetSalaries" datasource="cfdocexamples">
    SELECT Departmt.Dept_Name,
        Employee.Salary
    FROM Departmt, Employee
    WHERE Departmt.Dept_ID = Employee.Dept_ID
</cfquery>

<!--- Generate a query with statistical data for each department. --->
<cfquery dbtype = "query" name = "DeptSalaries">
    SELECT
        Dept_Name,
        AVG(Salary) AS AvgByDept
    FROM GetSalaries
    GROUP BY Dept_Name
</cfquery>

<!--- Reformat the generated numbers to show only thousands. --->
<cfloop index="i" from="1" to="#DeptSalaries.RecordCount#">
    <cfset DeptSalaries.AvgByDept[i]=Round(DeptSalaries.AvgByDept[i]/1000)*1000>
</cfloop>

<html>
<head>
    <title>Employee Salary Analysis</title>
</head>

<body>
<h1>Employee Salary Analysis</h1>

<!--- Bar chart, from DeptSalaries Query of Queries. --->
<cfchart
        xAxisTitle="Department"
        yAxisTitle="Salary Average"
        font="Arial"
```

```
            gridlines=6
            showXGridlines="yes"
            showYGridlines="yes"
            showborder="yes"
            show3d="yes"
        >

        <cfchartseries
            type="bar"
            query="DeptSalaries"
            valueColumn="AvgByDept"
            itemColumn="Dept_Name"
            seriesColor="olive"
            paintStyle="plain"
        />
    </cfchart>

    <br>
    </body>
    </html>
```

2 このページに "chartdata.cfm" という名前を付けて、Web のルートディレクトリの下の "myapps" ディレクトリに保存します。たとえば、Windows の場合のディレクトリパスは C:¥Inetpub¥wwwroot¥myapps のようになります。

3 ブラウザに戻り、次の URL を入力して "chartdata.cfm" ページを表示します。

http://localhost/myapps/chartdata.cfm

注意：クエリー内の 2 つの行で itemColumn 属性の値が同じである場合は、クエリー内で最後にある行を使用してグラフが作成されます。前述の例で、営業部門の行がクエリー内に **2** つ含まれている場合は、クエリー内で最後にある行の値が使用されます。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明

`<cfquery name="GetSalaries" datasource="cfdocexamples">`
`SELECT Departmt.Dept_Name, Employee.Salary`
`FROM Departmt, Employee`
`WHERE Departmt.Dept_ID = Employee.Dept_ID`
`</cfquery>`
	cfdocexamples データベースをクエリーして、全従業員の部門名 (Dept_Name) と給与 (Salary) を取得します。Dept_Name は Departmt テーブルに、Salary は Employee テーブルにあるので、WHERE 節でテーブルを結合する必要があります。このクエリーの結果データを、ページ内の別の箇所で使用することができます。

コード	説明
<cfquery dbtype = "query" name = "DeptSalaries"> SELECT Dept_Name, AVG(Salary) AS AvgByDept FROM GetSalaries GROUP BY Dept_Name </cfquery>	GetSalaries クエリーから、新しいクエリーを生成します。AVG 集計関数を使用して、従業員の統計データを生成します。部門ごとに 1 行のみが生成されるように、GROUP BY ステートメントを使用します。
<cfloop index="i" from="1" to="#DeptSalaries.RecordCount#"> <cfset DeptSalaries.AvgByDept[i]=Round(DeptSalaries.AvgByDept[i]/1000)*1000> </cfloop>	DeptSalaries クエリーのすべての行をループし、給与データを千の位で丸めます。このループでは、クエリー変数 RecordCount を使用して行数を取得し、クエリーオブジェクトの内容を直接変更しています。
<cfchart xAxisTitle="Department" yAxisTitle="Salary Average" font="Arial" gridlines=6 showXGridlines="yes" showYGridlines="yes" showborder="yes" show3d="yes" > <cfchartseries type="bar" query="DeptSalaries" valueColumn="AvgByDept" itemColumn="Dept_Name" seriesColor="olive" paintStyle="plain" /> </cfchart>	DeptSalaries クエリーの AvgByDept 列のデータを使用して棒グラフを作成します。ラベルには部門名を使用します。

ループを使用して給与データを丸める代わりに、cfchartseries タグ内で cfoutput タグと cfchartdata タグを使用して、次のように書き直すこともできます。

```
<cfchartseries
    type="bar"
    seriesColor="olive"
    paintStyle="plain">

    <cfoutput query="deptSalaries">
        <cfchartdata item="#dept_name#" value=#Round(AvgByDept/1000)*1000#>
    </cfoutput>

</cfchartseries>
```

クエリーとデータポイントの組み合わせ

クエリーとデータ値の両方からチャートを作成するには、cfchartseries タグでクエリー名および関連する属性を指定し、cfchartdata タグを使用して追加のデータポイントを指定します。

cfchartdata タグで指定したチャートデータは、クエリーのデータの前 (たとえば、棒グラフでは左側) に表示されます。cfchart タグの sortXAxis 属性を使用して、X 軸に沿ってデータをアルファベット順にソートすることができます。

クエリーとデータポイントを組み合わせる例としては、データベースにないデータを提供する場合があります。たとえば、ある部門のデータがない場合に、そのデータを追加することができます。次の手順では、前のセクションのクエリーで取得した給与データに Facilities 部門と Documentation 部門のデータを追加しています。

1 エディタで "chartdata.cfm" ファイルを開きます。

2 次のように cfchart タグを編集します。

```
<cfchart chartwidth="600">
    <cfchartseries
        type="bar"
        query="DeptSalaries"
        itemColumn ="Dept_Name"
        valueColumn="AvgByDept"
        >
        <cfchartdata item="Facilities" value="35000">
        <cfchartdata item="Documentation" value="45000">
    </cfchartseries>
</cfchart>
```

3 このページに "chartqueryanddata.cfm" という名前を付けて、Web のルートディレクトリの下の "myapps" ディレクトリに保存します。たとえば、Windows の場合のディレクトリパスは C:¥Inetpub¥wwwroot¥myapps のようになります。

4 ブラウザに戻り、次の URL を入力して "chartqueryanddata.cfm" ページを表示します。

http://localhost/myapps/chartqueryanddata.cfm

複数のデータコレクションのチャート作成

複数のデータ系列を 1 つのチャートで表示したり、2 つのデータ系列を同じチャートで比較したい場合があります。また、同じチャートで異なるチャートタイプを使用したい場合もあります。たとえば、棒グラフと折れ線グラフを組み合わせることがあります。

複数のデータ系列を 1 つのチャートに合成するには、複数の cfchartseries タグを 1 つの cfchart タグ内に記述します。複数のデータ系列をどのように表示するかは、cfchart タグの seriesPlacement 属性で指定します。この属性では、次のオプションを指定できます。

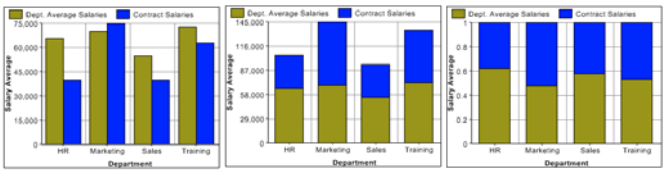
default データを合成する最適な方法が ColdFusion によって決定されます。

cluster 各系列の対応するチャート要素を隣り合わせに配置します。

stacked 各系列の対応する要素を結合します。

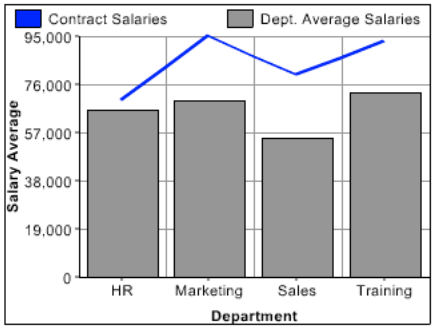
percent 各系列の要素を、対応する全要素の合計のパーセンテージで表します。

次の図は、2 つの棒グラフをさまざまなオプションで合成しています。



cluster　stacked　percent

異なるチャートタイプを組み合わせることもできます。次の図は、棒グラフと折れ線グラフを組み合わせたものです。



他のチャートと組み合わせることができないチャートタイプは、円グラフのみです。あるデータ系列で円グラフを指定した場合、他のチャートは表示されません。

前の図 (棒グラフと折れ線グラフが含まれたチャート) を表示する手順を次の例に示します。この例では、正社員 (棒グラフ) と契約社員 (折れ線グラフ) の給与を対比するチャートを作成します。

注意：複数の系列が重ねられる順序は、cfchartseries タグを記述する順序によって決まります。たとえば、棒グラフを最初に記述し、折れ線グラフを 2 番目に記述した場合は、折れ線グラフの手前に棒グラフが表示されます。

棒グラフと折れ線グラフを組み合わせたチャートを作成するには

1 エディタで "chartdata.cfm" ファイルを開きます。

2 次のように cfchart タグを編集します。

```
<cfchart
        backgroundColor="white"
        xAxisTitle="Department"
        yAxisTitle="Salary Average"
        font="Arial"
        gridlines=6
        showXGridlines="yes"
        showYGridlines="yes"
        showborder="yes"
    >
    <cfchartseries
        type="line"
        seriesColor="blue"
        paintStyle="plain"
        seriesLabel="Contract Salaries"
        >
            <cfchartdata item="HR" value=70000>
            <cfchartdata item="Marketing" value=95000>
            <cfchartdata item="Sales" value=80000>
            <cfchartdata item="Training" value=93000>
        </cfchartseries>
    <cfchartseries
        type="bar"
        query="DeptSalaries"
        valueColumn="AvgByDept"
        itemColumn="Dept_Name"
        seriesColor="gray"
        paintStyle="plain"
        seriesLabel="Dept. Average Salaries"
        />

</cfchart>
```

3 このページに "chart2queries.cfm" という名前を付けて、Web のルートディレクトリの下の "myapps" ディレクトリに保存します。

4 ブラウザに戻り、"chart2queries.cfm" ページを表示します。

チャートの外観の設定

次の方法でチャートの外観を設定できます。

- ColdFusion のデフォルトのチャートスタイルの使用
- cfchart タグおよび cfchartseries タグの属性の使用
- 独自のチャートスタイルの作成

ColdFusion のデフォルトのチャートスタイルの使用

ColdFusion には次のチャートスタイルが用意されています。

- beige
- blue
- default
- red
- silver
- yellow

これらのスタイルを使用するには、cfchart タグの style 属性を使用してスタイルを指定します。次の例では、beige スタイルを使用しています。

```
<cfchart style="beige">
    <cfchartseries type="pie">
        <cfchartdata item="New car sales" value="50000">
        <cfchartdata item="Used car sales" value="25000">
        <cfchartdata item="Leasing" value="30000">
        <cfchartdata item="Service" value="40000">
    </cfchartseries>
</cfchart>
```

cfchart タグおよび cfchartseries タグの属性を使用して、チャートの外観を指定できます。

cfchart タグでは、チャートタイプに応じて、次のオプションの特性を指定できます。

チャートの特性	使用する属性	説明	チャートタイプ
ファイルタイプ	format	チャートを JPEG、PNG、SWF のいずれのファイルとして送信するかを指定します。デフォルトの形式は SWF ファイルです。	すべて
サイズ	chartWidth chartHeight	チャートの幅と高さ (ピクセル単位)。この設定によって、チャート領域全体 (凡例やチャートを囲む背景領域を含む) のサイズが定義されます。 デフォルトでは、高さは 240 ピクセル、幅は 320 ピクセルです。	すべて
カラー	foregroundColor dataBackgroundColor backgroundColor	前面オブジェクトと背景オブジェクトのカラー。 デフォルトでは、前景色は black、背景色は white です。 16 個のカラー名、有効な HTML カラー書式、または 8 桁の 16 進数の値を指定して、RGB 値および透明度を設定することができます。数値の形式を使用する場合は、シャープ記号を使用します。たとえば、##FF33CC のようにします。カラーと透明度を指定するには、##xxFF33CC という形式を使用します。xx は透明度を表します。値 FF は不透明を表し、値 00 は透明を表します。詳細なリストについては、『ColdFusion 設定と管理』を参照してください。	すべて
ラベル	font fontSize fontBold frontItalic labelFormat xAxisTitle yAxisTitle	font 属性は、すべてのテキストのフォントを指定します。デフォルト値は Arial です。UNIX でダブルバイトの文字セットを使用する場合、またはファイルタイプを Flash に設定して Windows でダブルバイトの文字セットを使用する場合は、フォントとして ArialUnicodeMs を指定します。 **メモ :** ColdFusion サーバーにインストールされていないフォントを指定した場合は、利用可能な別のフォントが使用されます。フォントを指定していない場合、日本語、中国語、韓国語などの非 ASCII 文字が正しく表示されない可能性があります。 fontSize は、すべてのテキストのフォントサイズを整数で指定します。デフォルト値は 11 です。 fontBold 属性を yes にすると、すべてのテキストがボールドになります。デフォルト値は no です。 fontItalic 属性を yes にすると、すべてのテキストがイタリックになります。デフォルト値は no です。 labelFormat 属性は、Y 軸のラベルの形式を number (数値)、currency (通貨)、percent (パーセント)、または date (日付) のいずれかで指定します。デフォルト値は number です。 xAxisTitle と yAxisTitle 属性は、X 軸と Y 軸のタイトルを指定します。	すべて
ボーダー	showBorder	showBorder 属性を使用して、チャートを囲むボーダーを描画します。foregroundColor 属性は、ボーダーカラーを指定します。デフォルト値は no です。	すべて
グリッド線	showXGridlines showYGridlines gridLines	showXGridlines および showYGridlines 属性を使用して、x 軸および y 軸のグリッド線を表示します。デフォルトでは、X 軸のグリッド線は no、Y 軸のグリッド線は yes です。 gridLines 属性によって、値軸のグリッド線の総数を指定します。総数には軸そのものも含まれます。各グリッド線の値は、値軸に沿って表示されます。cfchart タグでは、水平グリッド線のみが表示されます。デフォルト値は 0 で、グリッド線は表示されません。	面積 棒 円錐 曲線 円柱 横棒 折れ線 角錐 散布 ステップ

チャートの特性	使用する属性	説明	チャートタイプ
スライスのスタイル	pieSliceStyle	円グラフの表示を solid または sliced で指定します。デフォルト値は sliced です。	円
マーカー	showMarkers markerSize	showMarkers 属性を yes にすると、2D の折れ線グラフ、曲線グラフ、散布グラフのデータポイントにマーカーが表示されます。デフォルト値は yes です。 markerSize 属性は、マーカーのサイズを整数のピクセル値で指定します。デフォルト値は、ColdFusion によって決定されます。	すべて
値軸	scaleFrom scaleTo	データ軸上の最小ポイントと最大ポイントを指定します。 デフォルトでは、最小ポイントは 0 かチャート内の最小の負のデータ値となり、最大ポイントは最大のデータ値となります。 メモ **:** チャートの一部しか表示されないような値を scaleFrom または scaleTo 属性に指定した場合は、チャート全体が表示されるような値が自動的に使用されます。	面積 棒 円錐 曲線 円柱 横棒 折れ線 角錐 散布 ステップ
軸タイプ	XAxisType sortXAxis	X 軸の種類 (数値スケールであるかカテゴリの名前であるか) と、X 軸上での項目のソート方法を指定します。 XAxisType 属性値が scale の場合、X 軸は数値です。すべての cfchartdata で item 属性の値が数値であることが必要です。X 軸は自動的に数値でソートされます。scale 値にすることで、数値間の関係を表すグラフ (年齢別人口など) を作成することができます。 この属性値が category (デフォルト) の場合、X 軸はデータカテゴリを表します。 sortXAxis 属性によって、cfchartdata の item 属性で指定した項目の表示順を決定します。項目の値はテキストとして処理されます。デフォルトでは、最初のチャート系列で記述した順序で項目が表示されます。	面積 棒 円錐 曲線 円柱 横棒 折れ線 角錐 散布 ステップ
3D の外観	show3D xOffset yOffset	show3D 属性を yes にすると、チャートが 3D で表示されます。デフォルト値は no です。 xOffset および yOffset 属性で、水平軸 (xOffset) または垂直軸 (yOffset) に対するチャートの回転率を指定します。値 0 はフラット (回転なし)、-1 と 1 はそれぞれ、左 (-1) または右 (1) への 90 度の回転を表します。デフォルト値は 0.1 です。	すべて
複数の系列	showLegend seriesPlacement	showLegend 属性を yes にすると、複数のデータ系列が含まれる場合にチャートの凡例が表示されます。デフォルト値は Yes です。 seriesPlacement 属性は、各系列の相対的な位置を指定します。デフォルトでは、グラフタイプに基づいて各系列の最適な場所が ColdFusion によって決定されます。	すべて
ヒント	tipStyle tipBGColor	tipStyle 属性は、マウスポインタが指しているチャート要素の情報を示す小さいポップアップウィンドウの表示方法を指定します。オプションは、none、mousedown、または mouseover のいずれかです。デフォルト値は mouseover です。 tipBGColor 属性は、ヒントウィンドウの背景カラーを指定します (Flash 形式のみ)。デフォルト値は white です。	すべて

また、cfchartseries タグを使用して、チャートの外観属性を指定できます。次の表で、各属性について説明します。

チャートの特性	使用する属性	説明	チャートタイプ
複数の系列	seriesLabel seriesColor	seriesLabel 属性は、系列のラベルとして表示するテキストを指定します。 seriesColor 属性は、棒グラフ、折れ線グラフ、角錐グラフなどのシングルカラーを指定します。円グラフの場合は、最初のスライスのカラーになります。後のスライスのカラーは、指定した最初のカラーに基づいて自動的に決まります。または、colorList 属性を使用して指定することもできます。	すべて
ペイント	paintStyle	データ系列にカラーを適用する方法を指定します。単色、浮き上がったボタン、中央が薄く外側が濃い線形グラデーションの塗り、およびより薄いカラーでのグラデーションの塗りを指定できます。デフォルト値は solid です。	すべて
データポイントのカラー	colorList	棒、角錐、面積、横棒、円錐、円柱、ステップ、円グラフの各データポイントで使用するカラーのカンマ区切りリスト。 16 個のカラー名、有効な HTML カラー書式、または 8 桁の 16 進数の値を指定して、RGB 値および透明度を設定することができます。数値の形式を使用する場合は、シャープ記号を使用します。たとえば、##FF33CC のようにします。カラーと透明度を指定するには、##xxFF33CC という形式を使用します。xx は透明度を表します。値 FF は不透明を表し、値 00 は透明を表します。詳細なリストについては、『ColdFusion 設定と管理』を参照してください。 指定したカラー数がデータポイント数より少ない場合、それらのカラーが繰り返し使用されます。指定したカラー数がデータポイント数よりも多い場合、余分なカラーは使用されません。	円
データマーカー	markerStyle	データポイントをマークするためのシェイプを指定します。指定できるシェイプは、circle、diamond、letterx、mcross、rcross、rectangle、snow、および triangle です。2D チャート用にサポートされています。デフォルト値は rectangle です。	曲線 折れ線 散布
ラベル	dataLabelStyle	系列内の項目へのカラーの適用方法を指定します。スタイルには、None、Value、Rowlabel、Columnlabel、および Pattern があります。	すべて

独自のチャートスタイルの作成

次のいずれかの方法で、独自のチャートスタイルを作成できます。

- チャートスタイル XML ファイルの編集
- WebCharts3D によるチャートスタイルの作成

チャートスタイル XML ファイルの編集

ColdFusion で提供されているチャートスタイルを編集して、独自のチャートスタイルを作成できます。スタイル情報は、"<ColdFusion のルートディレクトリ >¥charting¥styles" ディレクトリの XML ファイルに記述されています。ファイルを編集するときは、XML ファイルで指定されている属性のみを変更してください。他の属性を指定する場合は、968 ページの「WebCharts3D によるチャートスタイルの作成」の説明に従ってください。

注意：デフォルトの各チャートスタイルには、2 つの XML ファイルが存在します。たとえば、円グラフの beige スタイルは "beige_pie.xml" ファイルで定義されており、その他のすべてのチャートタイプの beige スタイルは "beige.xml" ファイルで定義されます。

1 編集する XML ファイル ("beige.xml" など) を開きます。

2 ファイルの内容を編集します。

3 このファイルを別名で保存します ("myBeige.xml" など)。

WebCharts3D によるチャートスタイルの作成

ColdFusion MX 7 以降で提供されている WebCharts3D ユーティリティを使用して、チャートスタイルファイルを作成できます。

1. "<ColdFusion のルートディレクトリ >¥charting" ディレクトリの "webcharts.bat" ファイルをダブルクリックして、WebCharts3D を起動します。
2. (オプション) 既存のチャートを開きます。
3. チャートの外観を変更します。

   注意：cfchart タグで使用できるチャートスタイルファイルを作成するためには、この手順の後の表で示されている以外の変更は行わないでください。
4. XML スタイルタブをクリックします。
5. 右下隅にある [保存] ボタンをクリックします。
6. ファイル名を指定します ("mystyle.xml" など)。
7. このチャートスタイルファイルの保存先となるディレクトリを指定します。

   注意：チャートスタイル XML ファイルの検索ルールは、cfinclude タグのインクルードファイルの検索ルールと同じです。詳細については、cfinclude を参照してください。
8. 「保存」をクリックします。

次の表に、cfchart タグおよび cfchartseries タグの属性と、それに対応する WebCharts3D のコマンドを示します。

属性	WebCharts3D のコマンド
chartHeight	ハンドルによりチャートをドラッグ
chartWidth	ハンドルによりチャートをドラッグ
dataBackgroundColor	Background: minColor (タイプは PlainColor である必要がある)
font	font: Family (サポートされるフォントのみを指定)
fontBold	font: Bold をチェック
fontItalic	font: Italic をチェック
fontSize	font: Size
foregroundColor	foreground
gridlines	X-axis: labelcount
labelFormat	Y-axis: LabelFormat: Number | Percent| Currency | Datetime
markerSize	Elements: markerSize
pieSliceStyle	style: solid | slice
rotated	Type Frame chart: Elements: Shape:
scaleFrom	Yaxis: isAbsolute; scaleMin(int)
scaleTo	Yaxis: isAbsolute; scaleMax(int)
seriesPlacement	Elements: place
show3D	is3D
showBorder	Decoration: style (none または simple)?

属性	WebCharts3D のコマンド
showLegend	Legend: isVisible
showMarkers	Elements: showMarkers
showXGridlines	Frame: isVGridVisible
showYGridlines	Frame: isHGridVisible
tipbgColor	Popup: background
tipStyle	Popup: show on MouseOver | show on MouseDown | Disabled
url	Elements: action | Series: action
xAxisTitle	X-axis: TitleStyle: text (テキストを入力)
xAxisType	X-axis: type: (category または scale)
xOffset	Frame: xDepth
yAxisTitle	Y-axis: TitleStyle: text (テキストを入力)
yAxisType	現時点では効果無し
yOffset	Frame: yDepth

次の表に、cfchartseries タグの属性と、それに対応する WebCharts3D コマンドを示します。

属性	WebCharts3D のコマンド
colorlist	Elements: series: Paint: color
markerStyle	Elements: series: Marker type: Rectangle | Triangle | Diamond | Circle | Letter | MCROSS | Snow | RCROSS
paintStyle	Paint: paint: Plain | Shade | Light
seriesColor	Elements: series: Paint: color
seriesLabel	Elements: series:
type	Type: Pie chart | Type Frame chart: Elements: Shape: Bar | Line | Pyramid | Area | Curve | Step | Scatter | Cone | Cylinder | Horizontalbar |

チャートの作成 : 例

棒グラフの作成

次の例では、棒グラフにタイトルを追加し、3D グラフを指定し、グリッド線を追加し、Y 軸の最小値と最大値を設定し、カスタムカラーセットを使用しています。

1 エディタで "chartdata.cfm" ファイルを開きます。

2 次のように cfchart タグを編集します。

```
<!--- Bar chart, from Query of Queries --->
<cfchart
        scaleFrom=40000
        scaleTo=100000
        font="arial"
        fontSize=16
        gridLines=4
        show3D="yes"
        foregroundcolor="##000066"
        databackgroundcolor="##FFFFCC"
        chartwidth="450"
    >

    <cfchartseries
        type="bar"
        query="DeptSalaries"
        valueColumn="AvgByDept"
        itemColumn="Dept_Name"
        seriescolor="##33CC99"
        paintstyle="shade"
        />

</cfchart>
```

3 このファイルに "chartdatastyle1.cfm" という名前を付けて保存します。

4 "chartdatastyle1.cfm" ページをブラウザで表示します。

コードの説明

この例のコードについて、次の表で説明します。

コード	説明
scaleFrom=40000	垂直軸の最小値を 40000 に設定します。
scaleTo=100000	垂直軸の最大値を 100000 に設定します。
font="arial"	Arial フォントを使用してテキストを表示します。
fontSize=16	ラベルのポイントサイズを 16 ポイントに設定します。
gridLines = 4	チャートの上辺から底辺までに 4 本のグリッド線を表示します。
show3D="yes"	チャートを 3D で表示します。
foregroundcolor="##000066"	テキスト、グリッド線、およびラベルのカラーを設定します。
databackgroundcolor="##FFFFCC"	棒の背景カラーを設定します。
seriescolor="##33CC99"	棒のカラーを設定します。
paintstyle="shade"	ペイント表示スタイルを設定します。

円グラフの作成

次の例では、ページに円グラフを追加します。

1 エディタで "chartdata.cfm" ファイルを開きます。

2 DeptSalaries クエリーと cfloop コードを次のように編集します。

```
<!--- A query to get statistical data for each department. --->
<cfquery dbtype = "query" name = "DeptSalaries">
    SELECT
        Dept_Name,
        SUM(Salary) AS SumByDept,
        AVG(Salary) AS AvgByDept
    FROM GetSalaries
    GROUP BY Dept_Name
</cfquery>

<!--- Reformat the generated numbers to show only thousands. --->
<cfloop index="i" from="1" to="#DeptSalaries.RecordCount#">
    <cfset DeptSalaries.SumByDept[i]=Round(DeptSalaries.SumByDept[i]/
    1000)*1000>
    <cfset DeptSalaries.AvgByDept[i]=Round(DeptSalaries.AvgByDept[i]/
    1000)*1000>
</cfloop>
```

3 次の cfchart タグを追加します。

```
<!--- Pie chart, from DeptSalaries Query of Queries. --->
<cfchart
        tipStyle="mousedown"
        font="Times"
        fontsize=14
        fontBold="yes"
        backgroundColor = "##CCFFFF"
        show3D="yes"
        >

    <cfchartseries
        type="pie"
        query="DeptSalaries"
        valueColumn="SumByDept"
        itemColumn="Dept_Name"
        colorlist="##6666FF,##66FF66,##FF6666,##66CCCC"
        />
</cfchart>
<br>
```

4 このファイルに "chartdatapie1.cfm" という名前を付けて保存します。

5 "chartdatapie1.cfm" ページをブラウザで表示します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`SUM(Salary) AS SumByDept,`	DeptSalaries クエリーに SUM 集計関数を追加し、部門ごとに給与の合計を計算します。

コード	説明
<cfset DeptSalaries.SumByDept[i]=Round(DeptSalaries.SumByDept[i]/ 1000)*1000>	cfloop タグで、給与の合計を千の位で丸めます。
<cfchart tipStyle="mousedown" font="Times" fontsize=14 fontBold="yes" backgroundColor = "##CCFFFF" show3D="yes" >	ユーザーがチャートをクリックした場合にのみヒントを表示します。テキストは Times 太字フォント、背景カラーは淡い青、チャートは 3D で表示します。
<cfchartseries type="pie" query="DeptSalaries" valueColumn="SumByDept" itemColumn="Dept_Name" colorlist="##6666FF,##66FF66,##FF6666,##66CCCC" />	DeptSalaries クエリーで得られた給与の合計値である SumByDept を使用して、円グラフを作成します。 チャートの凡例に表示するアイテムラベルには、Dept_Name 列のデータを使用します。 16 進数で表記したカスタムカラーリストで、スライスのカラーを指定します。カラーデータが変数名として解釈されないように、# 記号を 2 つ重ねています。

面積グラフの作成

次の例では、給与分析ページに面積グラフを追加します。このチャートは、開始日別の平均給与を給与分析ページに表示します。ここでは、2 番目のクエリーオブクエリーを使用して、GetSalaries クエリーのデータを新たに分析します。さらに、cfchart の追加属性も使用しています。

1 エディタで "chartdata.cfm" ファイルを開きます。

2 次のように GetSalaries クエリーを編集します。

```
<!-- Get the raw data from the database. -->
<cfquery name="GetSalaries" datasource="cfdocexamples">
    SELECT Departmt.Dept_Name,
        Employee.StartDate,
        Employee.Salary
    FROM Departmt, Employee
    WHERE Departmt.Dept_ID = Employee.Dept_ID
</cfquery>
```

3 次のコードを html タグの前に追加します。

```
<!--- Convert start date to start year. --->
<!--- Convert the date to a number for the query to work --->
<cfloop index="i" from="1" to="#GetSalaries.RecordCount#">
<cfset GetSalaries.StartDate[i]=NumberFormat(DatePart("yyyy", GetSalaries.StartDate[i]) ,9999)>
</cfloop>

<!--- Query of Queries for average salary by start year. --->
<cfquery dbtype = "query" name = "HireSalaries">
    SELECT
        StartDate,
        AVG(Salary) AS AvgByStart
    FROM GetSalaries
    GROUP BY StartDate
</cfquery>

<!--- Round average salaries to thousands. --->
<cfloop index="i" from="1" to="#HireSalaries.RecordCount#">
    <cfset HireSalaries.AvgByStart[i]=Round(HireSalaries.AvgByStart[i]/1000)*1000>
</cfloop>
```

4 次の cfchart タグを、body タグブロックの末尾に追加します。

```
<!--- Area-style Line chart, from HireSalaries Query of Queries. --->
<cfchart
        chartWidth=400
        BackgroundColor="##FFFF00"
        show3D="yes"
    >
    <cfchartseries
        type="area"
        query="HireSalaries"
        valueColumn="AvgByStart"
        itemColumn="StartDate"
     />
</cfchart>
<br>
```

5 このページを保存します。

6 "chartdata.cfm" ページをブラウザで表示します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`Employee.StartDate,`	GetSalaries クエリーのデータに、従業員の開始日を追加します。
`<cfloop index="i" from="1" to="#GetSalaries.RecordCount#"><cfset GetSalaries.StartDate[i]=NumberFormat(DatePart("yyyy", GetSalaries.StartDate[i]) ,9999)></cfloop>`	cfloop タグを使用して採用データから採用年度を抽出し、その結果を 4 桁の数値に変換します。
`<cfquery dbtype = "query" name = "HireSalaries">  SELECT StartDate, AVG(Salary) AS AvgByStart FROM GetSalaries GROUP BY StartDate</cfquery>`	GetSalaries クエリーから、2 番目のクエリーを作成します。このクエリーには、開始年度別の平均給与が含まれます。
`<cfloop index="i" from="1" to="#HireSalaries.RecordCount#"><cfset HireSalaries.AvgByStart[i]=Round(HireSalaries.AvgByStart[i]/1000)*1000></cfloop>`	給与を千の位で丸めます。
`<cfchart chartWidth=400 BackgroundColor="##FFFF00" show3D="yes" ><cfchartseries type="area" query="HireSalaries" valueColumn="AvgByStart" itemColumn="StartDate" /></cfchart>`	HireSalaries クエリーを使用して、面積グラフを作成します。開始日別の平均給与のチャートを作成します。 チャートの幅を 400 ピクセルに制限し、背景カラーを white に設定して、3D でチャートを表示します。

曲線グラフの特性の設定

曲線グラフでは、既に説明した属性を使用します。ただし、曲線グラフは描画に時間がかかります。パフォーマンスを最大にするためには、チャートをオフラインで作成し、ファイルまたは変数に書き込んでから、アプリケーションページで参照します。オフラインでのチャート作成については、974 ページの「変数へのチャートの書き込み」を参照してください。

チャートの管理

チャートの管理には ColdFusion Administrator を使用します。Administrator では、チャートのキャッシュをメモリとディスクのいずれに保存するかを選択できます。また、キャッシュするチャート数、チャートのスレッド数、およびイメージをキャッシュするディスクファイルを指定できます。

作成したチャートは、ColdFusion によってキャッシュされます。したがって、同じチャートが何度もリクエストされても、チャートの描画処理が繰り返されることはなく、キャッシュに保存されているチャートがロードされます。

注意：キャッシュされているチャートを参照するのに特別なコーディングは必要ありません。cfchart タグを使用すると、キャッシュにチャートが保存されていないかどうかが常に確認されます。保存されている場合は、そのチャートがキャッシュからロードされます。

次の表で、ColdFusion のチャートおよびグラフ作成エンジンの設定について説明します。

オプション	説明
[キャッシュタイプ]	キャッシュタイプを設定します。チャートは、メモリまたはディスクにキャッシュできます。メモリにキャッシュするほうが高速ですが、メモリ使用量が増えます。
キャッシュされるイメージの最大数	キャッシュに保管するチャートの最大数を指定します。この制限に達すると、新しいチャートを保存するための領域を作成するために、キャッシュ内の最も古いチャートが削除されます。キャッシュに保管できる最大チャート数は 250 です。
チャートを処理するスレッドの最大数	同時に処理可能な最大チャートリクエスト数を指定します。最小値は 1 で、最大値は 5 です。大きい数値を指定するほど、多くのメモリが必要となります。
ディスクキャッシュの位置	ディスクにキャッシュする場合は、生成されたチャートを保管するディレクトリを指定します。

変数へのチャートの書き込み

場合によっては、スタティックなチャートや、(データ入力の性質上) 描画に時間のかかるチャートを使用することがあります。このような場合は、チャートを作成して変数に書き込むことができます。

変数にチャートを書き込むと、他の ColdFusion ページでその変数を使用してチャートを表示できます。また、その変数をディスクに書き込むことで、ファイルとして保存することもできます。変数をディスクに保存すると、チャートを含むページがリクエストされるたびにチャートが作成されるのを避け、必要な場合にのみ作成や更新を行えば済むようになります。

チャートを変数に書き込むには、cfchart タグの name 属性を使用します。name 属性を指定すると、チャートはブラウザに描画されずに、変数に書き込まれます。

チャートは Adobe Flash SWF ファイル、または JPEG や PNG のイメージファイルとして保存できます。イメージを SWF ファイルとして保存すれば、ColdFusion Flash Remoting を使用して Flash クライアントに変数を渡すことができます。詳細については、595 ページの「Flash Remoting サービスの使用」を参照してください。

注意：JPEG または PNG ファイルにチャートを書き込んで、そのファイルからイメージを再表示した場合は、ポップアップヒントや、データのドリルダウン用にチャートに埋め込まれた URL は機能しません。ただし、イメージを SWF ファイルとして保存すれば、ヒントもドリルダウン URL も機能します。データのドリルダウンの詳細については、975 ページの「チャートのリンク先 URL の設定」を参照してください。

チャートを変数およびファイルに書き込むには

1 次の内容の ColdFusion ページを作成します。

```
<cfchart name="myChart" format="jpg">
    <cfchartseries type="pie">
        <cfchartdata item="New Vehicle Sales" value=500000>
        <cfchartdata item="Used Vehicle Sales" value=250000>
        <cfchartdata item="Leasing" value=300000>
        <cfchartdata item="Service" value=400000>
    </cfchartseries>
</cfchart>

<cffile
    action="WRITE"
    file="c:\inetpub\wwwroot\charts\vehicle.jpg"
    output="#myChart#">
<img src="/charts/vehicle.jpg" height=240 width=320>
```

2 このページに "chartToFile.cfm" という名前を付けて、Web のルートディレクトリの下の "myapps" に保存します。

3 "chartToFile.cfm" ページをブラウザで表示します。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
<cfchart name="myChart" format="jpg">	チャートを定義します。変数 myChart に JPEG 形式で書き込みます。
<cffile action="WRITE" file="c:¥inetpub¥wwwroot¥charts¥vehicle.jpg" output="#myChart#">	cffile タグを使用してチャートをファイルに書き込みます。
<img src="/charts/vehicle.jpg" height=240 width=320>	HTML の img タグを使用してチャートを表示します。

チャートのリンク先 URL の設定

ColdFusion のチャートでは、データのドリルダウン機能が使用できます。この機能を使用すれば、ユーザーがチャートのデータ領域や凡例領域をクリックしたときに、URL をリクエストすることができます。たとえば、円グラフのスライスをユーザーが選択したときに詳細情報を表示したい場合は、この機能を使用できます。

ユーザーがチャートをクリックしたときに開く URL を指定するには、cfchart タグの url 属性を使用します。たとえば、ユーザーがチャートをクリックしたときに "moreinfo.cfm" というページを開くには、次のコードを使用します。

```
<cfchart
    xAxisTitle="Department"
    yAxisTitle="Salary Average"
    url="moreinfo.cfm"
    >

        <cfchartseries
            seriesLabel="Department Salaries"
        ...
        />

</cfchart>
```

次の変数を url 属性で使用すると、ターゲットページに追加情報を渡すことができます。

- $VALUE$: 選択されたアイテムの値または空の文字列
- $ITEMLABEL$: 選択されたアイテムのラベルまたは空の文字列
- $SERIESLABEL$: 選択された系列のラベルまたは空の文字列

たとえば、ユーザーがグラフをクリックしたときに "moreinfo.cfm" ページを開き、3 つの値をすべてこのページに渡すには、次の URL を使用します。

```
url="moreinfo.cfm?Series=$SERIESLABEL$&Item=$ITEMLABEL$&Value=$VALUE$"
```

通常の ColdFusion 変数とは異なり、シャープ記号 (#) で変数を囲みません。変数はドル記号 ($) で囲みます。この url 属性値が使用されているチャートをクリックすると、次のような URL が生成されます。

```
http://localhost:8500/tests/charts/moreinfo.cfm?
    Series=Department%20Salaries&Item=Training&Value=86000
```

また、URL に JavaScript を使用して、クライアントサイドスクリプトを実行することもできます。例については、978 ページの「円グラフから JavaScript へのリンク」を参照してください。

円グラフのダイナミックリンクの設定

次の例では、円グラフのスライスをクリックすると、そのスライスで表される部門の詳細な給与情報が表の中に表示されます。この例は 2 つの部分に分かれています。詳細情報ページを作成する部分と、ダイナミックな円グラフを作成する部分です。

第 1 部 : 詳細情報ページの作成

このページでは、円グラフのスライスがクリックされた場合に、選択された部門の給与情報を表示します。部門名は $ITEMLABEL$ 変数を使用してこのページに渡されます。

1 次の内容のアプリケーションページを作成します。

```
<cfquery name="GetSalaryDetails" datasource="cfdocexamples">
    SELECT Departmt.Dept_Name,
        Employee.FirstName,
        Employee.LastName,
        Employee.StartDate,
        Employee.Salary,
        Employee.Contract
    FROM Departmt, Employee
    WHERE Departmt.Dept_Name = '#URL.Item#'
    AND Departmt.Dept_ID = Employee.Dept_ID
    ORDER BY Employee.LastName, Employee.Firstname
</cfquery>

<html>
<head>
    <title>Employee Salary Details</title>
</head>

<body>
```

```
<h1><cfoutput>#GetSalaryDetails.Dept_Name[1]# Department
    Salary Details</cfoutput></h1>
<table border cellspacing=0 cellpadding=5>
<tr>
    <th>Employee Name</th>
    <th>StartDate</th>
    <th>Salary</th>
    <th>Contract?</th>
</tr>
<cfoutput query="GetSalaryDetails">
<tr>
    <td>#FirstName# #LastName#</td>
    <td>#dateFormat(StartDate, "mm/dd/yyyy")#</td>
    <td>#numberFormat(Salary, "$999,999")#</td>
    <td>#Contract#</td>
</tr>
</cfoutput>
</table>
</body>
</html>
```

2 このページに "Salary_details.cfm" という名前を付けて、Web のルートディレクトリの下の "myapps" ディレクトリに保存します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明

`<cfquery name="GetSalaryDetails" datasource="cfdocexamples">`
`SELECT Departmt.Dept_Name,`
`Employee.FirstName,`
`Employee.LastName,`
`Employee.StartDate,`
`Employee.Salary,`
`Employee.Contract`
`FROM Departmt, Employee`
`WHERE Departmt.Dept_Name = '#URL.Item#'`
`AND Departmt.Dept_ID = Employee.Dept_ID`
`ORDER BY Employee.LastName, Employee.Firstname`
`</cfquery>`
	URL パラメータ文字列で渡された部門の給与データを取得します。データを従業員の氏名でソートします。

`<table border cellspacing=0 cellpadding=5>`
`<tr>`
`<th>Employee Name</th>`
`<th>StartDate</th>`
`<th>Salary</th>`
`<th>Contract?</th>`
`</tr>`
`<cfoutput query="GetSalaryDetails">`
`<tr>`
`<td>#FirstName# #LastName#</td>`
`<td>#dateFormat(StartDate, "mm/dd/yyyy")#</td>`
`<td>#numberFormat(Salary, "$999,999")#</td>`
`<td>#Contract#</td>`
`</tr>`
`</cfoutput>`
`</table>`
	クエリーで取得したデータを表に示します。開始日は標準の " 月 / 日 / 年 " の形式にし、給与はドル記号とカンマが付いた、小数部のない形式にしています。

第 2 部 : ダイナミックなチャートの作成

1 エディタで "chartdata.cfm" を開きます。

2 円グラフの cfchart タグを次のように編集します。

```
<cfchart
        font="Times"
        fontBold="yes"
        backgroundColor="##CCFFFF"
        show3D="yes"
        url="Salary_Details.cfm?Item=$ITEMLABEL$"
        >

    <cfchartseries
        type="pie"
        query="DeptSalaries"
        valueColumn="AvgByDept"
        itemColumn="Dept_Name"
        colorlist="##990066,##660099,##006699,##069666"
    />
</cfchart>
```

3 このファイルに "chartdetail.cfm" という名前を付けて保存します。

4 "chartdata.cfm" ページをブラウザで表示します。

5 円グラフのスライスをクリックして "Salary_details.cfm" ページをリクエストし、クリックしたスライスの部門名を渡します。その部門の給与情報が表示されます。

コードの説明

このコードの太字の部分について、次の表で説明します。

コード	説明
url = "Salary_Details.cfm?Item=$ITEMLABEL$"	ユーザーが円グラフのスライスをクリックすると、現在のディレクトリにある "Salary_details.cfm" ページが呼び出され、選択されたスライスの部門名が含まれる Item というパラメータが渡されます。

円グラフから JavaScript へのリンク

次の例では、円グラフのスライスをクリックすると、ColdFusion で JavaScript が実行されて、そのスライスに関するポップアップウィンドウが表示されます。

JavaScript によるダイナミックチャートの作成

1 次の内容のアプリケーションページを作成します。

```
<script>
function Chart_OnClick(theSeries, theItem, theValue){
alert("Series: " + theSeries + ", Item: " + theItem + ", Value: " + theValue);
    }
</script>

<cfchart
        xAxisTitle="Department"
        yAxisTitle="Salary Average"
        tipstyle=none
        url="javascript:Chart_OnClick('$SERIESLABEL$','$ITEMLABEL$','$VALUE$');"
    >
    <cfchartseries type="bar" seriesLabel="Average Salaries by Department">
        <cfchartData item="Finance" value="75000">
        <cfchartData item="Sales" value="120000">
        <cfchartData item="IT" value="83000">
        <cfchartData item="Facilities" value="45000">
    </cfchartseries>
</cfchart>
```

2 このページに "chartdata_withJS.cfm" という名前を付けて、Web のルートディレクトリの下の "myapps" ディレクトリに保存します。

3 "chartdata_withJS.cfm" ページをブラウザで表示します。

4 円グラフのスライスをクリックして、ポップアップウィンドウを表示します。

印刷用のレポートとドキュメントの作成

Adobe ColdFusion タグ、関数、ツールを使用して、印刷に適したページやレポートを作成できます。

印刷可能出力について

HTML ページはどの Web ブラウザでも印刷できますが、HTML 形式のページは印刷出力用に最適化されていません。たとえば、改行、改ページ、ヘッダ、フッタ、ページ番号などを制御できないという問題は、印刷用のレポートやページを作成する際に起こる問題の一部に過ぎません。

ColdFusion の場合、印刷可能出力とは、次の機能を含むページのことを指します。

- ページ番号
- ヘッダとフッタ
- 改ページ
- オンラインで表示したときにクリック可能なハイパーテキストリンク

ColdFusion には、印刷可能出力を生成するために次のタグが用意されています。

- cfdocument: 印刷可能出力を作成してブラウザに返すか、ファイルに保存します。詳細については、980 ページの「cfdocument タグを使用した PDF および FlashPaper 出力の作成」を参照してください。
- cfreport: 指定されたレポート定義を使用し、印刷可能出力を作成してブラウザに返すか、ファイルに保存します。ColdFusion は、次のツールで作成されたレポート定義をサポートします。
  - **ColdFusion Report Builder**: ColdFusion Report Builder は、ColdFusion に統合されているレポート作成ツールです。ColdFusion Report Builder では、バンド形式のレポートを作成できます。詳細については、986 ページの「Report Builder について」を参照してください。
  - **Crystal Reports**: Crystal Reports は、cfreport タグで利用可能なレポート定義を生成するレポート作成ツールです。詳細については、985 ページの「Crystal Reports を使用したレポートの作成 (Windows のみ)」を参照してください。

ColdFusion の印刷可能レポートは、次の形式で作成できます。

FlashPaper ColdFusion により SWF ファイルが作成されます。Adobe Flash Player の最新バージョンがクライアントにインストールされている必要があります。

Adobe Acrobat ColdFusion により PDF ファイルが作成されます。Adobe Reader がクライアントにインストールされている必要があります。

Microsoft Excel (ColdFusion レポート機能のみ) ColdFusion により Excel スプレッドシートが作成されます。

注意：Excel 形式のレポート出力では、ColdFusion のレポート作成で使用可能な形式設定オプションのサポートが制限されます。イメージとチャートはサポートされず、形式設定 (カンマ、パーセント、通貨記号など) を含む数値データはプレーンテキストとして Excel に表示されます。Excel 形式でレポートを出力する場合は単純なレポートしかサポートされないので、デザインとレイアウトを慎重に検討することをお勧めします。

Crystal Reports (Windows のみ) ColdFusion から Crystal Reports に制御が渡され、Crystal Reports により HTML が作成されます。このオプションは、cfreport タグでのみ使用できます。

cfdocument タグを使用した PDF および FlashPaper 出力の作成

cfdocument タグの開始タグと終了タグの間にあるデータはすべて PDF または FlashPaper 出力形式に変換され、ブラウザに返されるか、ファイルに保存されます。このタグを次の例のように使用すると、HTML を簡単に印刷可能出力に変換できます。

```
<cfdocument format="FlashPaper">
<p>This is a document rendered by the cfdocument tag.</p>
</cfdocument>
```

cfdocument タグはすべての HTML タグと CFML タグをサポートしますが、次のものはサポートしていません。

- cfchart
- Flash Player で表示されるコンテンツを生成するタグ
- form、cfform、cfapplet などのインタラクティブタグ
- 要素または要素の位置をダイナミックに変更する JavaScript

cfdocument タグの開始タグと終了タグの間に存在する HTML は、正しい形式を備えている必要があります。つまり、それぞれの開始タグに対応する終了タグが存在し、ブロックレベルの要素が適切にネストされている必要があります。

注意：<cfdocument> タグと </cfdocument> タグのペアの外側にある HTML や CFML は返されません。

HTML および CFML からの基本的なレポートの作成

HTML ベースのレポートを PDF または FlashPaper 出力に変換するには、cfdocument タグの開始タグと終了タグの間に HTML を記述し、必要に応じて cfdocument 属性を指定して次の項目をカスタマイズします。

- ページサイズ
- ページの向き
- マージン
- 暗号化 (PDF のみ)
- ユーザーパスワードとオーナーパスワード (PDF のみ)
- 権限 (PDF のみ)

これらのオプションの詳細については、『CFML リファレンス』の cfdocument タグに関する説明を参照してください。

注意：レポートにフォントを埋め込むと、ブラウザやプラットフォームの違いによる表示のばらつきを解消できます。フォントの埋め込みに関する詳細については、1007 ページの「簡単なレポートの作成」を参照してください。

次の例では、cfoutput タグを使用してクエリーをループし、従業員リストを表示します。

```
<cfdocument format="flashpaper">
<h1>Employee List</h1>
<!--- Inline query used for example purposes only. --->
<cfquery name="EmpList" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary, Contract
    FROM Employee
</cfquery>
<cfoutput query="EmpList">
#EmpList.FirstName#, #EmpList.LastName#, #LSCurrencyFormat(EmpList.Salary)#,
    #EmpList.Contract#<br>
</cfoutput>
</cfdocument>
```

セクション、ヘッダ、フッタの作成

cfdocument および cfdocumentsection タグを使用すると、次のように印刷可能出力を微調整できます。

- cfdocumentitem: 改ページ、ヘッダ、またはフッタを作成します。
- cfdocumentsection: 出力をいくつかのセクションに分割し、必要に応じてカスタムマージンを指定します。セクションごとに一意のヘッダとフッタを指定するには、セクション内で cfdocumentitem タグを使用します。各ドキュメントセクションは新しいページから開始されます。

cfdocumentitem タグ

ヘッダやフッタを指定したり、改ページを作成するには、cfdocumentitem タグを使用します。cfdocumentsection タグを cfdocumentitem タグと組み合わせて使用する場合と、組み合わせずに使用する場合の結果は次のようになります。

- cfdocumentsection と組み合わせて使用する場合 : cfdocumentitem 属性はそのセクションにのみ適用され、以前に指定されたヘッダとフッタは上書きされます。
- cfdocumentsection と組み合わせずに使用する場合 : cfdocumentitem 属性は次のようにドキュメント全体に適用されます。
  - タグがドキュメントの先頭にある場合は、ドキュメント全体に適用されます。
  - タグがドキュメントの中央にある場合は、ドキュメントの残りの部分に適用されます。
  - タグがドキュメントの終わりにある場合は、何も効果を持ちません。

ドキュメント全体のランニングヘッダを作成するには、次の例のように cfdocumentitem を使用します。

```
 <cfdocument format="PDF">
<!--- Running header --->
<cfdocumentitem type="header">
    <font size="-3"><i>Directory Report</i></font>
</cfdocumentitem>
<h3>cfdirectory Example</h3>
<!--- Use cfdirectory to display directories by name and size --->
<cfdirectory
    directory="#GetDirectoryFromPath(GetTemplatePath())#"
    name="myDirectory" recurse="yes"
    sort="directory ASC, name ASC, size DESC">
<!---- Output the contents of the cfdirectory as a cftable ----->
<cftable query="myDirectory"
    htmltable colheaders>
    <cfcol header="DIRECTORY:" text="#directory#">
    <cfcol header="NAME:" text="#Name#">
    <cfcol header="SIZE:" text="#Size#">
</cftable>
</cfdocument>
```

cfdocumentsection タグ

cfdocumentsection を使用する場合は、ドキュメント内のすべてのテキストを cfdocumentsection タグで囲む必要があります。ColdFusion は、cfdocumentsection タグの外側にある HTML と CFML を無視します。各セクションのマージン属性は、前のセクションまたは親の cfdocument タグで指定されたマージンより優先されます。マージン属性を指定した場合は、親の cfdocument タグの unit 属性によって単位が制御されます。unit 属性のデフォルトはインチです。

セクションごとに一意のヘッダとフッタを指定し、各セクションの前に改ページを挿入するには、次の例のように、セクション内で cfdocumentitem タグを使用します。

```
<cfquery datasource="cfdocexamples" name="empSalary">
SELECT Emp_ID, firstname, lastname, e.dept_id, salary, d.dept_name
FROM employee e, departmt d
WHERE e.dept_id = d.dept_id
ORDER BY d.dept_name
</cfquery>

<cfdocument format="PDF">
    <cfoutput query="empSalary" group="dept_id">
        <cfdocumentsection>
            <cfdocumentitem type="header">
                <font size="-3"><i>Salary Report</i></font>
            </cfdocumentitem>
            <cfdocumentitem type="footer">
            <font size="-3">Page #cfdocument.currentpagenumber#</font>
            </cfdocumentitem>
            <h2>#dept_name#</h2>
            <table width="95%" border="2" cellspacing="2" cellpadding="2" >
            <tr>
                <th>Employee</th>
                <th>Salary</th>
            </tr>
        <cfset deptTotal = 0 >
        <!--- inner cfoutput --->
        <cfoutput>
            <tr>
                <td><font size="-1">
                #empSalary.lastname#, #empSalary.firstname#</font>
                </td>
                <td align="right"><font size="-1">
                #DollarFormat(empSalary.salary)#</font>
                </td>
            </tr>
        <cfset deptTotal = deptTotal + empSalary.salary>
        </cfoutput>
            <tr>
                <td align="right"><font size="-1">Total</font></td>
                <td align="right"><font size="-1">#DollarFormat(deptTotal)#</font></td>
            </tr>
            <cfset deptTotal = 0>
            </table>
        </cfdocumentsection>
    </cfoutput>
</cfdocument>
```

cfdocument スコープの使用

cfdocument タグを使用すると、ColdFusion によって cfdocument という名前のスコープが作成されます。このスコープには、次の変数が含まれます。

currentpagenumber 現在のページ番号を表示します。

totalpagecount 総ページ数を表示します。

currentsectionpagenumber 現在のセクション番号を表示します。

totalsectionpagecount 総セクション数を表示します。

注意：cfdocument スコープの変数は、cfdocumentitem タグ内の式でのみ使用できます。

次の例のように、これらの変数をヘッダまたはフッタで使用すると、現在のページ番号と総ページ数を表示できます。

```
<cfdocumentitem type= "footer> #cfdocument.currentpagenumber# of
    #cfdocument.totalpagecount#</cfdocumentitem>
```

PDF ファイルのしおりの作成

PDF ドキュメント内のセクションごとにしおりを作成するには、次の例のように、cfdocument タグの bookmark 属性を使用します。

```
<cfdocument format="PDF" bookmark="yes">
    <cfdocumentitem type="header">
    <font size="-1" align="center"><i>Building Better Applications</i></font>
    </cfdocumentitem>
    <cfdocumentitem type="footer">
    <font size="-1"><i>Page <cfoutput>#cfdocument.currentpagenumber# of
        #cfdocument.totalpagecount#</cfoutput></i></font>
    </cfdocumentitem>
    <cfdocumentsection name="Introduction">
        <h3>Introduction</h3>
        <p>The introduction goes here.</p>
    </cfdocumentsection>
    <cfdocumentsection name="Chapter 1">
        <h3>Chapter 1: Getting Started</h3>
        <p>Chapter 1 goes here.</p>
    </cfdocumentsection>
    <cfdocumentsection name="Chapter 2">
        <h3>Chapter 2: Building Applications</h3>
        <p>Chapter 2 goes here.</p>
    </cfdocumentsection>
    <cfdocumentsection name="Conclusion">
        <h3>Conclusion</h3>
        <p>The conclusion goes here.</p>
    </cfdocumentsection>
</cfdocument>
```

PDF ドキュメントのしおりパネルにしおりが表示されます。

cfhttp タグを使用した Web ページの表示

Web ページ全体を PDF または FlashPaper 出力形式で表示するには、次の例のように、cfhttp タグを cfdocument タグと組み合わせて使用します。

```
<!--- You can pass a URL in the URL string --->
<cfparam name="url.target_url" default="http://www.boston.com">
<cfoutput>
<cfhttp url="#url.target_url#" resolveurl="yes">

<cfdocument format="FlashPaper">
<cfdocumentitem type="header">
    <cfoutput>#url.target_url#</cfoutput>
</cfdocumentitem>
<cfdocumentitem type="footer">
    <cfoutput>#cfdocument.currentpagenumber# / #cfdocument.totalpagecount#</cfoutput>
</cfdocumentitem>
<!--- Display the page --->
#cfhttp.filecontent#
</cfdocument>
</cfoutput>
```

高度な PDF オプションの使用

cfdocument タグは、次の表に示す Acrobat セキュリティオプションをサポートします。

セキュリティオプション	説明
暗号化	PDF 出力を暗号化するかどうかを指定するには、encryption 属性を使用します。次のいずれかを指定します。 • 128-bit • 40-bit • none
ユーザーパスワード	ドキュメントを表示するためにユーザーが入力する必要があるパスワードを指定するには、userpassword 属性を使用します。
オーナーパスワード	ドキュメントを表示し、必要に応じて変更を加えるためにユーザーが入力する必要があるパスワードを指定するには、ownerpassword 属性を使用します。

また、cfdocument タグの permissions 属性では、次の Acrobat セキュリティ権限がサポートされます。次のいずれかの値を指定します。複数の権限を指定する場合は、値をカンマで区切ります。

権限	説明
印刷	ドキュメントの印刷を許可するには、AllowPrinting 属性を指定します。
変更	ドキュメントの変更を許可するには、AllowModifyContents 属性を指定します (ただし、ドキュメントを変更するには、適切なソフトウェアを所有している必要があります)。
コピー	ドキュメント内のテキストを選択してコピーすることを許可するには、AllowCopy 属性を指定します。
注釈	ドキュメントにコメントを追加することを許可するには、AllowModifyAnnotations を指定します。ユーザーが注釈を追加する場合は、本人が手動で PDF を保存する必要があります。
スクリーンリーダ	スクリーンリーダを使用してドキュメントにアクセスすることを許可するには、AllowScreenReaders を指定します。
入力	フォームフィールドの使用を許可するには、AllowFillIn を使用します。
組み立て	ページの挿入、削除、回転、しおりやサムネールの作成を許可するには、AllowAssembly を指定します。
低品質印刷	低解像度印刷を許可するには、AllowDegradedPrinting を使用します。この形式では、各ページがビットマップとして印刷されるため、印刷時間が長くなる場合があります。

注意：これらのオプションのデフォルト値は、暗号化レベルによって異なります。これらのオプションは、PDF のみに適用されます。詳細については、『CFML リファレンス』の cfdocument に関する説明を参照してください。

次の例では、コピーのみが許可された PDF ドキュメントを作成します。

```
<cfdocument format="PDF" encryption="40-bit"
    ownerpassword="us3rpa$$w0rd" userpassword="us3rpa$$w0rd"
    permissions="AllowCopy" >
<h1>Employee List</h1>
<cfquery name="EmpList" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary
    FROM Employee
</cfquery>
<cfoutput query="EmpList">
    #EmpList.FirstName#, #EmpList.LastName#, #LSCurrencyFormat(EmpList.Salary)#<br>
</cfoutput>
</cfdocument>
```

印刷可能レポートをファイルに保存する方法

生成された PDF または SWF 出力をファイルに保存するのは、次の例のように cfdocument filename 属性を使用します。

```
<!--- The compasstravel database is part of the Getting Started
    tutorial application, found under the cfdocs directory. --->
<cfquery datasource="compasstravel" name="compasstrips">
SELECT tripName, tripDescription, tripLocation, price
FROM trips
ORDER BY price
</cfquery>
<cfdocument format="pdf"
        filename="#GetDirectoryFromPath(GetTemplatePath())#/compasstrips.pdf"
        overwrite="yes">
    <cfdocumentsection>
        <h1 align="center">Compass Travel</h1>
        <h2 align="center">Destination Guide</h2>
        <p align="center"><img src="cfdocs/getting_started/photos/somewhere.jpg"></p>
    </cfdocumentsection>
    <cfdocumentsection>
        <cfdocumentitem type="header">
            <font size="-3"> <i>Compass Travel Trip Descriptions</i></font>
        </cfdocumentitem>
        <cfdocumentitem type="footer">
            <font size="-3">
                <cfoutput>Page #cfdocument.currentpagenumber#</cfoutput>
            </font>
        </cfdocumentitem>
        <cfoutput query="compasstrips">
            <hr>
            <h2>#tripName#</h2>
            <p><b>#tripLocation#</b></p>
            <p>Price: #DollarFormat(price)#</p>
            <p>#tripDescription#</p>
        </cfoutput>
    </cfdocumentsection>
</cfdocument>
```

Crystal Reports を使用したレポートの作成 (Windows のみ)

Windows では、cfreport タグを使用して、Crystal Reports バージョン 9 または 10 で作成されたレポートを実行できます。

注意：Crystal Reports のインストール時に、[HTML へのエクスポートを有効化] オプションと [ディスクへのエクスポートを有効化] オプションを選択します。これらのオプションはデフォルトでは無効になっているので、[Custom Install] オプションを使用する必要があります。

1. Crystal Reports でレポート定義を作成します。
2. CFM ページを作成し、Crystal Reports レポート定義を呼び出す cfreport タグを追加します。次の例に示す cfreport タグは、Crystal Reports レポート定義を呼び出して、フィルタ条件を渡します。

```
<cfreport report = '/reports/monthlysales.rpt'>
    {Departments.Department} = 'International'
</cfreport>
```

3. ブラウザを開き、CFM ページを表示します。

ColdFusion は COM を使用して Crystal Reports バージョン 9 の Craxdrt 9.dll と Crystal Reports バージョン 10 の Craxdrt.dll を呼び出します。cfreport タグで問題が発生した場合は、これらの **DLL** が登録されていることを確認し、登録されていない場合は、**regsvr32** を使用して登録してください。これらの **DLL** のデフォルトの場所は、**C:¥Program Files¥Crystal Decisions¥Report Designer Component¥** です。

Crystal Reports でレポートを定義する方法については、Crystal Reports のマニュアルを参照してください。

Report Builder を使用したレポートの作成

Adobe ColdFusion Report Builder と CFML を使用して統合ビジネスレポートを作成すると、重要なビジネスデータをさらに効果的に活用できます。

Report Builder について

ColdFusion レポート機能は、統合ビジネスレポート機能を ColdFusion に追加することにより、重要なビジネスデータへのアクセスを可能にします。ColdFusion レポート機能は、サーバーサイドのランタイム処理と、Report Builder というグラフィカルユーザーインターフェイス (GUI) で構成されています。

Report Builder のインストールの詳細については、987 ページの「概要」を参照してください。

Report Builder は、バンド形式のレポートを作成するための Windows 専用ツールです。バンド形式のレポートは、複数の水平セクション (バンド) で構成され、印刷されるレポートの各部分にバンドが 1 つずつ割り当てられます。たとえば、レポートヘッダバンドのデータとテキストはレポートの先頭に出力されます。ページヘッダバンドのデータとテキストは各ページの先頭に出力されます。ページフッタバンドのデータとテキストは各ページの最後に出力されます。レポートの中央には詳細バンドがあり、レポートの結果セットまたはデータベースクエリーの各行に対応する行が実行時に 1 行ずつ挿入されます。

Report Builder には、わかりやすいチュートリアルトピックや状況依存型のダイアログボックスヘルプを含む詳細なオンラインヘルプシステムが用意されています。オンラインヘルプを参照するには、F1 キーを押します。

Report Builder と CFR ファイル

Report Builder は、レポート定義を作成するための独立したアプリケーションであり、必要に応じて ColdFusion サーバーと対話します。Report Builder は、レポート定義情報を ColdFusion Report (CFR) ファイルに保存します。このファイルには、フィールド定義、書式、データベース SQL ステートメント、CFML などの情報が含まれています。CFR ファイルを表示するには cfreport タグを使用します。また、レポートに対して CFR が有効になっている場合は、CFR ファイルをブラウザで実行すると、該当するレポートが表示されます。

注意： Report Builder は Windows プラットフォーム上でのみ動作します。ただし、Report Builder で作成した CFR ファイルは、ColdFusion が実行されていて、ColdFusion レポート機能が有効になっているすべてのプラットフォームで動作します。

RDS

RDS (Remote Development Services) は、クエリービルダーやチャートウィザードから ColdFusion データソースを介してデータベースデータにアクセスするために開発された HTTP ベースの独自プロトコルです。Report Builder でこの機能を有効にするには、**RDS** サーバーの設定を定義します。RDS サーバーとは、RDS が有効になっている ColdFusion サーバーの別名です。

詳細については、1001 ページの「レポートでの CFML の使用」を参照してください。

実行時

実行時に CFR ファイルを表示するには、ColdFusion レポート機能が有効になっている ColdFusion サーバーを使用します。CFR ファイルは直接実行して表示することもできますが、cfreport タグを使用して実行することもできます。また、出力をブラウザに返す代わりに、レポートをファイルに保存することもできます。入力パラメータや外部クエリーを渡す必要があるレポートの場合は、cfreport タグを使用してください。cfreport タグで渡されたクエリー属性は、レポート定義に含まれる内部 SQL ステートメントより優先されます。

概要

インストール手順については、『ColdFusion インストール』を参照してください。Report Builder をインストールすると、RDS で使用される Windows DLL も登録されます。これらの DLL が正しく登録されなかった場合は、Report Builder の起動時または RDS の使用時にエラーが発生します。

セットアップウィザード

Report Builder を初めて起動すると、セットアップウィザードが開始されます。セットアップウィザードでは、レポートで使用する ColdFusion サーバーのデフォルト設定を定義します。これらの設定には次のものがあります。

- デフォルトの測定単位 : インチ、センチメートル、またはピクセル。
- ColdFusion サーバー。クエリービルダーおよびチャートウィザードからデータベース内のデータにアクセスするときに使用する RDS サーバー (そのサーバーで RDS が有効にする必要があります)。次の情報をセットアップウィザードで入力する必要があります。
  - ホスト名または IP アドレス。
  - Web サーバーのポート。通常のポート番号は、Web サーバーコネクタを使用する場合は 80、サーバー設定でビルトイン Web サーバーを使用する場合は 8500、マルチサーバー設定で cfusion サーバーとビルトイン Web サーバーを使用する場合は 8300 です。J2EE サーバーを使用する場合は、J2EE サーバー固有の Web サーバーのポート番号を指定します。
  - レポートで使用する ColdFusion サーバーの RDS パスワード。
- ColdFusion サーバーで使用されている Web ルートのディレクトリパス (例 : "C:¥Inetpub¥wwwroot" または "C:¥**ColdFusion**¥wwwroot")。
- ColdFusion サーバーで使用されている Web ルートの URL (例 : "http://localhost" または "http://localhost:8500" など)。

セットアップウィザードでの設定が完了すると、[Report Gallery] ダイアログボックスが表示されます。[レポートウィザードの使用] ラジオボタンをクリックすると、レポートの作成ウィザードが起動します。このウィザードで必要な情報を指定すると、完全なレポート定義が自動的に生成されます。

レポートの作成ウィザードの詳細については、Report Builder オンラインヘルプを参照してください。

RDS の設定

レポートを定義する ColdFusion サーバーごとに、RDS サーバーを設定します。RDS サーバーを設定すると、ColdFusion サーバーで定義されたデータソースにクエリービルダーを使用してアクセスできるようになり、レポートのクエリーフィールドとして使用するデータベース列を選択できるようになります。

RDS サーバーを追加するには

1. メニューバーから [編集]-[環境設定] を選択して [環境設定] ダイアログボックスを開きます。
2. [サーバー接続] をクリックします。
3. ダイアログボックスの左上隅にあるポップアップメニューの横にあるプラス記号 (+) をクリックします。
4. [RDS サーバーの設定] ダイアログボックスで、次の情報を指定して [OK] をクリックします。

   説明 サーバー接続の名前。この名前は、クエリービルダーの左側のポップアップメニューに表示されます。

   ホスト名 ColdFusion を実行するホスト。**localhost** または IP アドレスを入力します。

   ポート Web サーバーのポート番号。デフォルトのポート (80) を使用するか、ColdFusion サーバーのビルトイン Web サーバーのポート番号 (デフォルトのポート番号は 8500) を入力します。

   コンテキストルート（J2EE 設定のみ）ColdFusion Web アプリケーションのコンテキストルート (存在する場合)。

SSL（Secure Socket Layer）を使用して接続（オプション）SSL セキュリティを有効にします。

ユーザー名 ColdFusion RDS の場合は不要。

パスワード RDS パスワード。このパスワードは ColdFusion Administrator で設定します。

ColdFusion Administrator では ColdFusion Administrator パスワードも管理されるので、RDS パスワードと ColdFusion Administrator パスワードを混同しないように注意してください。

パスワードのプロンプト ユーザーがクエリービルダーを使用するたびに RDS パスワードを要求するかどうかを指定します。このオプションを選択する場合は、[ユーザー名] フィールドと [パスワード] フィールドを空白にします。

デフォルトの RDS サーバーを指定するには

1 メニューバーから [編集]-[環境設定] を選択して [環境設定] ダイアログボックスを開きます。

2 [サーバー接続] をクリックします。

3 [優先 RDS サーバー] ポップアップメニューから RDS サーバーを選択し、[OK] をクリックします。

クエリービルダーまたはチャートウィザードを開くと、指定したサーバーに自動的に接続します。

ユーザーインターフェイスの使用方法とヒント

Report Builder のワークスペースには、次の領域があります。

- **ツールボックス**: テキスト、シェイプ、イメージ、サブレポート、グラフなど、レポートに配置される不変要素が含まれます。ツールボックス要素を使用するには、要素をクリックした後、レポートバンド内にドラッグして要素のサイズを定義します。レポートバンドに要素を配置した後は、[プロパティ] パネルを使用して要素の外観と動作を変更できます。
- **整列パネル**: Ctrl キーまたは Shift キーを押しながらレポートバンド内の複数の要素をクリックして選択し、適切な整列アイコンをクリックします。Ctrl + A キーを押すと、レポートバンド内のすべての要素を選択できます。
- **レポートバンド**: レポートバンドに、ツールボックス要素、クエリーフィールド、計算フィールドを配置します。デフォルトのレポートバンドは、レポートヘッダ、ページヘッダ、列ヘッダ、ページフッタ、レポートフッタ、および透かしです。デフォルトでは、ページヘッダ、ページフッタ、透かしは閉じられています。これらを展開するには、隣接するいずれかのスプリッタバーをドラッグします。グループのバンドを追加定義するには、[レポート]-[グループ管理] を選択します。

ワークスペースにデータ要素を配置して書式を設定するために使用するパネルには、次の 3 種類があります。

- **[プロパティ] パネル**: 選択したフィールドの表示特性とレポート特性が含まれます。[プロパティ] パネルを表示するには、メインメニューから [ウィンドウ]-[プロパティインスペクタ] を選択します。プロパティの値を変更するには、新しい値を入力または選択して、Enter キーを押します。プロパティの詳細については、Report Builder オンラインヘルプを参照してください。
- **[フィールドとパラメータ] パネル**: クエリーフィールド、入力パラメータ、計算フィールドに関連する項目が含まれます。[フィールドとパラメータ] パネルを表示するには、メインメニューから [ウィンドウ]-[フィールドとパラメータ] を選択します。追加、編集、および削除の各アイコンを使用して、これらのフィールドを管理します。フィールドを定義した後、レポートバンド内にフィールド名をドラッグすると、フィールドまたはそのラベル (あるいはその両方) を追加できます。
- **[レポートスタイル] パネル**: レポートに対して定義するスタイルが含まれます。[レポートスタイル] パネルを表示するには、メインメニューから [ウィンドウ]-[レポートスタイル] を選択します。レポートスタイルを管理するには、追加、編集、削除アイコンを使用します。スタイルを定義すると、レポート内の要素ごとにフォントやフォントサイズなどを指定する代わりに、複数の要素に対して同じ属性を適用できます。レポートのレイアウト、プラットフォーム、使用可能なフォントなどの条件が変わった場合は、スタイルを変更して、レポート全体に変更を適用できます。また、任意のスタイルをレポートのデフォルトスタイルとして指定できます。特定のスタイルが適用されていない要素にはデフォルトスタイルが適用されます。

[ビュー] メニューを使用すると、ツールボックスやパネルウィンドウを表示するかどうかを制御できます。また、ウィンドウのタイトルをクリックすると、そのウィンドウを切り離して、画面上の他の領域にドラッグできます。たとえば、3 つのパネルをすべてドラッグして、同じウィンドウに固定できます。パネルを切り替えるには、ウィンドウ上部のタブをクリックします。ツールウィンドウまたはパネルを再び固定するには、移動するウィンドウまたはパネルを横または隅にドラッグして、矩形が表示されたらマウスボタンを離します。

詳細については、990 ページの「一般的なレポート作成作業と方法」およびオンラインヘルプを参照してください。

レポート定義のガイドライン

効果的なレポートを作成するには、ColdFusion Report Builder でレポートを定義する前に、次のことを計画してください。

- レポートデザインの問題 :

  読者 このレポートを作成する目的は何か。読者は誰か。

  データ どのようなデータがレポートに必要か。データをどこから入手するか。クエリービルダーを使用する場合でも、レポートにクエリーを渡す場合でも、必要なデータを事前に検討してください。

  グループ化 グループは必要か。必要な場合は、結果セットが適切な順序で返されるように準備し、ソート列に基づいてグループを定義する必要があります。

  計算フィールド 計算を行う必要があるフィールドはあるか。列の合計を計算するには、計算フィールドを使用します。行ごとに合計を計算するには、SQL を使用します。詳細については、990 ページの「一般的なレポート作成作業と方法」を参照してください。

  入力パラメータ レポートに変数入力は必要か。必要な場合は、入力パラメータを定義し、cfreportparam タグを使用して実行時に値をレポートに渡します。詳細については、990 ページの「一般的なレポート作成作業と方法」を参照してください。

- データの取得方法 :

  クエリービルダーと基本的な SQL 標準的な選択条件 (ソートを含む WHERE 節や通常の選択条件セットなど) のみをレポートで使用する場合や、短時間でレポートを作成する必要がある場合は、このオプションを使用します。この方法では、キャッシングなどの cfquery オプションを指定することもできます。

  クエリービルダーと拡張クエリーモード ColdFusion クエリーをレポート定義にカプセル化して使用する場合は、このオプションを使用します。このオプションは、cfdirectory タグ、cfldap タグ、または cfpop タグでクエリーを渡す場合や、クエリーオブクエリーを使用する場合や、QueryNew 関数を使用してダイナミックにクエリーを構築する場合にも便利です。

  cfreport タグと外部クエリー レポートで使用する結果セットをより細かく制御する必要がある場合は、このオプションを使用します。たとえば、クライアント側で選択条件をダイナミックに構築するためのフォームをアプリケーションに組み込む場合に使用します。

- 関連する視覚的情報 :

  チャート 詳細については、1004 ページの「チャートの使用」を参照してください。

  サブレポート 詳細については、1005 ページの「サブレポートの使用」を参照してください。

印刷可能レポートでのフォント管理

レポートの外観は、すべてのクライアントプラットフォームとすべてのブラウザで一貫していることが理想的です。ColdFusion はレポート定義のサイズ指定を使用してグラフィックとイメージを処理することにより、自動的にこの一貫性を実現します。ただし、使用可能なフォントは、ブラウザの種類、バージョン、言語、プラットフォームなどによって異なる場合があるため、レポートのフォント表示に影響する場合があります。さまざまな要因によってレポート表示の一貫性を維持します。

埋め込みフォント

埋め込みフォントを使用すると、レポート表示の一貫性を維持できます。ただし、レポートにフォントを埋め込むと、ファイルサイズが大きくなります。

出力形式

埋め込みフォントの処理方法は、FlashPaper 形式と PDF 形式で異なります。

FlashPaper FlashPaper には常にフォントが埋め込まれるため、レポートは常に適切に表示されます。

PDF PDF レポートには、任意でフォントを埋め込むことができます。レポートにフォントを埋め込まない場合は、必要なフォントがクライアントコンピュータに存在していることを確認する必要があります。

サーバーコンピュータとクライアントコンピュータで使用可能なフォント

レポート内のフォントをレンダリングする際の必要条件は、フォントが存在する場所によって異なります。

サーバーコンピュータ いずれの形式を使用する場合も、レポートで使用されるフォントが ColdFusion を実行するコンピュータ上に存在している必要があります。ColdFusion でレポートを正しくレンダリングするには、これらのフォントが必要です。ColdFusion は、Acrobat のビルトインフォントと、標準のフォントディレクトリ ("Windows¥fonts" ディレクトリなど) に存在するフォントを自動的に検索します。通常以外の場所にフォントがインストールされている場合は、ColdFusion Administrator でそれらのフォントを登録する必要があります。これにより、cfdocument タグと cfreport タグを使用してフォントを検索し、PDF レポートと FlashPaper レポートをレンダリングできるようになります。

クライアントコンピュータ PDF レポートにフォントを埋め込まない場合、レポート表示の一貫性を維持するには、クライアントコンピュータ上にフォントが存在している必要があります。

論理フォントから物理フォントへのマッピング

serif、sans serif、monospaced などの Java 論理フォントを使用する場合、ColdFusion は "<ColdFusion のルートディレクトリ >/lib/cffont.properties" ファイルを使用して、これらのフォントを物理フォントにマッピングします (マルチサーバー設定または J2EE 設定の場合、このファイルは "<ColdFusion Web アプリケーションのルートディレクトリ >/WEB-INF/cfusion/lib" ディレクトリにあります)。これらのマッピングは必要に応じて変更できます。また、英語以外のロケールのオペレーティングシステムを使用している場合は、ファイル名の最後に .<Java ロケールコード > を追加して、ロケール固有のマッピングファイルを作成できます。英語以外のロケールで実行されていることが検出されると、ColdFusion は最初に cffont.properties.<Java ロケールコード > ファイルをチェックします。たとえば、中国語ロケールを使用するコンピュータの場合は、ファイル名を "cffont.properties.cn" に変更します。Java ロケールコードの詳細については、Sun の Web サイトを参照してください。

ColdFusion には、日本語ロケール用の "cffont.properties.ja" ファイルがデフォルトで含まれています。

cfdocument タグと cfreport タグについても同様のことが当てはまります。詳細については、Report Builder オンラインヘルプを参照してください。

一般的なレポート作成作業と方法

Report Builder を使用すると、さまざまな形式のレポートにデータを挿入したり、計算を実行したりすることができます。トラブルシューティングのヒントなどについては、Report Builder オンラインヘルプを参照してください。

グループ化とグループ区切り

情報をグループ化すると、レポートの構成がわかりやくなります。新しいグループごとに個別の見出しを定義したり、グループごとの小計などを各グループ領域の末尾に表示したりすることができます。たとえば、部門、従業員、給与額を表示するレポートを作成するとします。部門別にデータをグループ化すると、各部門の給与の特性を簡単に把握できます。部門

ID が変わると、ColdFusion Report Builder によってグループ区切りが挿入されます。グループ区切りが挿入されると、グループフッタを表示することによって前のグループが終了され、グループヘッダを表示することによって新しいグループが開始されます。

ColdFusion Report Builder によってデータそのものがグループ化されることはありません。結果セットを取得するために使用する SQL が既に適切な順序でグループ化されていることを確認します。通常、グループ化を実装するには、レポートで使用する SQL SELECT ステートメントで "ORDER BY" 節を指定します。たとえば、次のような SQL SELECT ステートメントを使用します。

```
SELECT EmployeeID, LastName, FirstName, Title, City, Region, Country
FROM Employees
ORDER BY Country, City
```

この例では、2 つのグループを定義できます。1 つ目は Country に対応するグループで、2 つ目は City に対応するグループです。複数のグループを定義すると、[グループ管理] ダイアログボックスに上向きの矢印キーと下向きの矢印キーが表示されます。これらを使用して、グループの階層を制御できます。たとえば、都市は国の中に存在するので、国は都市より上のレベルに移動する必要があります。

グループを定義するには

1. メニューバーから [レポート]-[グループ管理] を選択します。
2. 「追加」をクリックします。
3. [名前] フィールドでグループ名を指定します。
4. [グループ] フィールドで、グループ化を制御する値 (グループ式) を指定します。この値の結果が変化すると、実行時にグループ区切りが挿入されます。通常は、クエリーフィールドの名前をこれらの値として指定します。ただし、計算フィールドやその他の種類の式をこの値として指定することもできます。グループ式には、次のものがあります。

   クエリーフィールド 結果セット内の指定された列の値が変わると、グループ区切りが挿入されます。指定するフィールドは query.country のように、結果セットのソート条件の 1 つである必要があります。

   計算フィールド 計算フィールドによって返される値が変わると、グループ区切りが挿入されます。たとえば、calc.FirstLetter という式がクエリー列の最初の文字を返す場合は、レポートをアルファベット順にグループ化できます。

   ブール式 ブール式によって返される値が変わると、グループ区切りが挿入されます。たとえば、passpercentage 列を基準に結果セットをソートする場合は、query.passpercentage LT 50 というブール式を使用します。
5. グループ区切りのオプションを指定します。

   [新規列の開始] グループ区切りの位置で新しい列を強制的に開始します。

   [新規ページの開始] グループ区切りの位置で新しいページを強制的に開始します。

   [ページ番号のリセット] グループ区切りの位置でページ番号を 1 にリセットします。
6. バンドのサイズと印刷情報を指定します。

   [グループの最小高さ] 指定した値以上の高さが現在のページに残っていない場合は、新しいページにグループバンドが印刷されます。

   [ページごとにヘッダを再出力] ページごとにグループヘッダが表示されます。
7. 「OK」をクリックします。

   レポートにグループが追加され、グループのヘッダバンドとフッタバンドが作成されます。
8. もう一度 [OK] をクリックします。
9. グループのヘッダとフッタに、見出し、テキスト、クエリーフィールド、計算フィールドなどの情報を追加します。

グループの小計を作成するには

1 グループ小計を表示するには、計算フィールドを作成します。次の条件を使用する計算フィールドを作成します。

- データ型を数値に設定します。
- [計算] フィールドで [合計] を選択します。
- [計算の対象] フィールドで、合計を算出するフィールドを指定します。たとえば、部門別従業員レポートでは、query.emp_salary で合計を算出します。
- グループが変わったときにフィールドをリセットするように指定します。

2 計算フィールドをレポートに配置します。

計算フィールドの詳細については、Report Builder オンラインヘルプを参照してください。

フィールドと入力パラメータの定義、変更、使用

Report Builder は、クエリーフィールド、入力パラメータ、計算フィールドを通じて、次のように可変データをサポートします。

クエリーフィールド レポートで指定されたデータベース結果セット内の列にマッピングされます。指定されたデータベースクエリーの列ごとに、クエリーフィールドを 1 つずつ定義します。

計算フィールド レポート内の複数の詳細行を分析または合計します。計算フィールドの値はレポートの生成時にダイナミックに計算されます。また、レポート、ページ、列、グループごとに値を再計算することもできます。

入力パラメータ 実行時にレポートに渡すデータフィールドを指定します。これらのフィールドは、cfreportparam タグを使用して渡すことも、メインレポートからサブレポートに渡すこともできます。入力パラメータは、レポートバンドに直接配置することも、計算フィールドへの入力として使用することもできます。

クエリーフィールドを定義するには

1 [ウィンドウ]-[フィールドとパラメータ] を選択します。

2 [クエリーフィールド] をクリックします。

3 タブの上部にあるプラス記号 (+) をクリックします。

4 名前フィールドの値を入力します。この値は、対応する cfquery ステートメントの列名と一致している必要があります。また、ピリオドは使用できません。

5 デフォルトのラベルを入力します。

6 対応するデータベース列のデータ型を次の中から指定します。

オブジェクト	時間	ロング
ブール値	ダブル	ショート
バイト	フロート	Big Decimal
日付	整数	文字列
タイムスタンプ	BLOB	CLOB

7 「OK」をクリックします。

注意：結果セット内のすべてのデータベース列に対応するクエリーフィールドがクエリービルダーによって自動的に定義されます (これは拡張クエリービルダーには当てはまりません)。レポートの作成ウィザードの一部としてクエリービルダーを実行した場合も、ウィザードによってクエリーフィールドがレポートに配置されます。

計算されるフィールドを定義するには

1 [ウィンドウ]-[フィールドとパラメータ] を選択します。

2 [計算されるフィールド] をクリックします。

3 タブの上部にあるプラス記号 (+) をクリックします。

4 名前、デフォルトのラベルテキスト、データ型を指定します。データ型のオプションはクエリーフィールドと同じです。

5 計算オプションを指定します。

[計算] 実行する計算の種類を指定します。有効な値は、[平均]、[カウント]、[DistinctCount]、[First]、[最高]、[最低]、[なし]、[標準偏差]、[合計]、[システム]、および [変化] です。[なし] を指定する場合、通常は [計算の対象] フィールドを使用してダイナミックな式を指定します。[なし] ([計算の対象] フィールドを使用する場合) と [システム] (独自のスクリプトレットクラスを記述する場合) を除き、これらの種類の計算を使用してグループ、ページ、レポートの合計を算出します。

[計算の対象] フィールドまたは式を指定します。[...] ボタンをクリックすると式ビルダーが表示されます。

初期値 計算フィールドの初期値を指定します。

6 次のリセットオプションを指定して、[OK] をクリックします。

[フィールドをリセット] 計算フィールドの値をいつリセットするか指定します。有効な値は、[なし]、[レポート]、[ページ]、[列]、[グループ] です。

[リセットするグループ] [フィールドをリセット] を [グループ] に設定した場合は、このフィールドを使用して、どのグループのグループ区切りが検出されたときにフィールドをリセットするかを指定します。

計算フィールドの詳細については、Report Builder オンラインヘルプを参照してください。

入力パラメータを定義するには

1 [ウィンドウ]-[フィールドとパラメータ] を選択します。

2 [フィールドとパラメータ] パネルで、[入力パラメータ] をクリックします。

3 タブの上部にあるプラス記号 (+) をクリックします。

4 [入力パラメータの追加] ダイアログボックスで、名前フィールドに値を入力します。この値は、レポート定義を使用する cfreport タグに含まれる cfreportparam タグの name 属性などの入力パラメータと一致していなければなりません。

5 デフォルトのラベルテキストを入力します。

6 データ型とデフォルト値を指定し、[OK] をクリックします。データ型のオプションはクエリーフィールドと同じです。

入力パラメータの使用方法の詳細については、1001 ページの「実行時に変数などのデータを渡す入力パラメータの使用」および 1005 ページの「サブレポートの使用」を参照してください。

レポートバンドにクエリーフィールド、計算フィールド、入力パラメータを配置するには

1 [フィールドとパラメータ] パネルのラジオボタンを使用して、ラベルとフィールドのどちらを配置するかを指定します (両方を指定することもできます)。

2 [フィールドとパラメータ] タブから適切なレポートバンドに、クエリーフィールド、計算フィールド、または入力パラメータをドラッグします。

3 適切なバンドに、クエリーフィールド、計算フィールド、または入力パラメータをドラッグします。

4 (オプション) [プロパティ] パネルを使用して、フィールド表示をカスタマイズします。

たとえば、query.emp_salary というクエリーフィールドと、query.emp_salary の合計を算出する計算フィールドがあり、計算フィールドの値をグループごとにリセットするとします。その場合は、query.emp_salary を詳細バンドに配置し、計算フィールドをグループフッタバンドに配置します。

レポートバンドでのツールボックス要素の使用

イメージ、円、正方形、線、ダイナミックフィールド、チャート、サブレポートなどのグラフィック要素やテキスト要素をレポートバンドに追加するには、ツールボックスを使用します。

ツールボックス要素を最も簡単に追加するには、ツールボックス要素をクリックし、適切なレポートバンド内をクリック & ドラッグして領域を定義します。イメージやテキストボックスなどのツールボックス要素を追加した場合は、ダイアログボックスが表示され、必要な情報を入力するよう求められます。また、[プロパティ] シートを使用すると、すべてのツールボックス要素の外観をカスタマイズできます。


ツールボックス要素は [挿入] メニューを使用して追加できます。

チャートの詳細については、1004 ページの「チャートの使用」を参照してください。サブレポートの詳細については、1005 ページの「サブレポートの使用」を参照してください。

テキストボックスを作成するには

1. ツールボックスの [ラベル] アイコン (abc) をクリックします。
2. 適切なバンド内をクリック & ドラッグして、ラベルの領域を定義します。
3. [ラベルテキストの編集] ダイアログボックスに、ラベルテキストを入力します。改行を追加するには、Ctrl + Enter キーを押します。
4. [OK] をクリックするか、Enter キーを押します。

注意：ラベルの前後にある空白は自動的にトリミングされます。前後の空白を残すには、ダイナミックフィールドを定義して、式に空白を含めます。たとえば、" My Title " のように定義します。

イメージファイルをインポートするには

1. ツールボックス内の [イメージ] アイコンをクリックします。
2. 適切なバンド内をクリック & ドラッグして、イメージの領域を定義します。
3. [Image File Name] ダイアログボックスで、イメージファイルが存在するディレクトリを参照し、ファイルを選択して [OK] をクリックします。

データベースの BLOB 列をイメージのソースとして使用するには

1. ツールボックスの [イメージ] アイコンをクリックします (このアイコンには木の絵が描かれています)。
2. 適切なバンド内をクリック & ドラッグして、イメージの領域を定義します。

   [Image File Name] ダイアログボックスが表示されます。


[フィールドとパラメータ] タブからレポートバンドに [BLOB] フィールドをドラッグすることもできます。

1. [キャンセル] をクリックします。

   式ビルダーが表示されます。
2. [イメージ形式] ポップアップメニューをクリックして [File/URL] を [BLOB] に変更します。
3. BLOB 列を含むクエリーフィールドまたは入力パラメータを選択します。

   注意：BLOB 列には、GIF 形式、JPEG 形式、または PNG 形式のバイナリイメージが含まれている必要があります。
4. 「OK」をクリックします。

注意：この手順は、BLOB 列の内容がイメージとしてレンダリング可能であることを前提としています。

矩形、楕円、線を追加するには

1 ツールボックスで矩形、楕円、または線のアイコンをクリックします。

2 適切なバンド内をクリック & ドラッグして、領域または線を定義します。

3 選択した要素のサイズを変更するには、要素の周囲にあるハンドルをドラッグします。

Ctrl キーを押しながら矩形または楕円のサイズを変更すると、要素の形状は常に正方形または正円になります (同じ方法で線のサイズを変更した場合は、常に 45 度の倍数の角度になります)。

ダイナミックフィールドを追加するには

1 ツールボックスの [フィールド] アイコンをクリックします。

2 適切なバンド内をクリック & ドラッグして、ダイナミックフィールドの領域を定義します。

[フィールドの追加] ダイアログボックスが表示されます (クエリーフィールドが定義されていない場合は、式ビルダーが表示されます)。

3 追加するフィールドを選択します。クエリーフィールド、計算フィールド、または入力パラメータを選択した場合は、[フィールドとパラメータ] タブからドラッグした場合と同じ結果になります。

4 (オプション) [手動で入力される式] を選択します。

式ビルダーが表示されます。このオプションは、同じ行で変数を使用した計算を行う場合に便利です。たとえば、注文明細の合計金額を計算するには、次の式を使用します。

```
LSNumberFormat((query.unitprice * query.quantity), ",_.__")
```

5 「OK」をクリックします。

要素の整列

読みやすいレポートを作成するには、要素のレイアウトを整える必要があります。これを行うには、同じバンド内の要素、バンド自体、または別のバンド内の要素を基準として、各バンドの視覚的要素の位置を揃え、間隔を調整し、中央を合わせます。

Report Builder の [パレットの整列] には、次のオプションがあります。

- 左端揃え、中央揃え、右端揃え
- 上端揃え、中央揃え、下端揃え
- 同じ高さ、同じ幅、同じ高さと幅
- 均等配置 (水平方向)
- 均等配置 (垂直方向)

複数のレポート要素の配置、サイズ、または間隔を調整する手順は次のとおりです。

バンド自体を基準にして調整する場合 : [パレットの整列] の下部にある [バンドに揃える] アイコンを使用して、相対位置を制御します。[バンドに揃える] アイコンを有効にすると、アイコンの周囲に矩形が表示され、バンドの高さと幅を基準にして要素が整列され、間隔が設定されます。

要素間で相対的に調整する場合 : [バンドに揃える] を無効にすると、2 つ以上の要素が相互に相対的に整列され、間隔が設定されます。

[パレットの整列] を使用するには

1 Ctrl キーまたは Shift キーを押しながら要素をクリックして選択するか、投げ縄ツールを使用して 2 つ以上の要素を選択します。

2 整列アイコンをクリックするか、メニューバーから [修正]-[整列]-[< 整列オプション >] を選択します。

[パレットの整列] のオプションは、メニューバーの [修正]-[整列] からも選択できます。

要素の外観を微調整する方法の詳細については、Report Builder オンラインヘルプを参照してください。

レポートスタイルの使用

レポートスタイルは Microsoft Word のフォントスタイルに似ています。要素ごとに書式設定を適用するのではなく、スタイルを適用することで複数の要素の書式を制御できます。これにより、レポート全体で書式を制御できるようになります。

また、任意のスタイルをレポートのデフォルトスタイルとして指定できます。特定のフォント属性またはスタイルが適用されていないフィールドには、デフォルトスタイルが適用されます。デフォルトスタイルが定義されている場合、そのスタイルは [レポートスタイル] パネル内に太字で表示されます。

Report Builder では、CSS (Cascading Style Sheet) ファイルからスタイルをインポートしたり、Report Builder で定義したスタイルを CSS ファイルにエクスポートしたりすることもできます。この方法を使用すると、複数のレポートに共通の書式を適用したり、CFM ページを使用して実行時にスタイルを置き換えたりすることができます。詳細については、1015 ページの「CSS (Cascading Style Sheet) の使用」および『CFML リファレンス』を参照してください。

注意：レポートで使用するフォントを選択するときは、ColdFusion を実行するサーバーにそれらのフォントがインストールされていることを確認してください。また、レポートにフォントを埋め込まない場合は、クライアントコンピュータにもそれらのフォントがインストールされていることを確認してください。フォントの詳細については、1007 ページの「簡単なレポートの作成」を参照してください。

スタイルを定義するには

1. [ウィンドウ]-[レポートスタイル] を選択します。
2. [レポートスタイル] タブの上部にある (+) アイコンをクリックします。
3. [名前] フィールドに値を入力します。スタイル名は一意である必要があります。
4. 他のスタイル特性を追加して、[OK] をクリックします。

特定のスタイルをデフォルトとして指定するには

1. テキストスタイルを編集するか、新規のテキストスタイルを作成します。
2. [オブジェクトに対して他のスタイルが選択されていない場合は、これがデフォルトのスタイルです] オプションを選択します。
3. 他のテキストスタイル特性を追加または変更して、[OK] をクリックします。

レポート要素にスタイルを適用するには

1. レポートバンド内の要素を選択します。
2. [ウィンドウ]-[プロパティインスペクタ] を選択します。
3. [スタイル] ポップアップメニューからスタイルを選択します。

詳細については、Report Builder オンラインヘルプを参照してください。

レポートのプレビュー

レポートの作成は反復的な作業であり、通常は作成中のレポートを定期的に表示して、変更を加えた箇所を確認しながら作業を進めます。レポートで内部クエリーが使用されていて、デフォルトの Web ルートが既に設定されている場合は、プレビュー機能が自動的に有効になります。外部クエリーを使用するレポートの場合は、CFM ページを定義して、その CFM ページをレポートに関連付けます。レポートをプレビューすると、そのページが実行されます。

内部クエリーを使用するレポートをプレビューするには

1 (オプション) 次の設定をまだ定義していない場合は、[環境設定] ダイアログボックスを使用して、デフォルトのサーバー接続情報を定義します。

- デフォルトの RDS サーバーの設定 (クエリービルダーとチャートウィザードを使用する場合にのみ必要。レポートプレビューのみを使用する場合は不要)。
- ローカルの Web ルートディレクトリの完全修飾パス (例 : C:¥**ColdFusion**¥wwwroot または C:¥Inetpub¥wwwroot)。
- ローカルの Web ルートの URL (例 : http://localhost:8500 または http://localhost)。

2 (オプション) [レポートプロパティ] ダイアログボックスで出力形式を指定します (デフォルトの形式は FlashPaper)。

3 (オプション) CFM ページが実行される場合は、[レポートプロパティ] ダイアログボックスで CFM ページの URL を指定します。

4 レポートを保存します。

5 メニューバーから [ファイル]-[プレビュー] を選択し、レポートを表示します。

注意 : [プレビュー] ウィンドウではなく [編集 : プレビュー URL] ダイアログボックスが表示された場合は、メニューバーから [編集]-[環境設定] を選択し、[サーバー接続] ペインで Web ルートのファイルと URL の設定が正しいことを確認してください。

6 F12 キーを押してプレビューウィンドウを閉じます。

cfreport タグからクエリーオブジェクトを受け取るレポートの場合は、レポートに URL を関連付けます。必要な場合は Report Builder でレポートをプレビューするときに、この URL を入力するよう求められることがあります。また、[レポートプロパティ] ダイアログボックスを開き、[プレビュー URL] フィールドで CFM ページの URL を指定することもできます。

cfreport タグを使用すると、内部クエリーと外部クエリーのどちらが使用されているかに関係なく、レポートを実行できます。

関連する CFM ファイルを使用してプレビューするには

1 メニューバーから [レポート]-[レポートプロパティ] を選択します。

2 [プレビュー URL] フィールドで、関連付ける CFM ページの URL を指定します。この CFM ページには、template 属性で現在の CFR ファイルが指定されていて、必要に応じて query 属性のクエリーを渡す cfreport タグが含まれている必要があります。

3 レポートを保存します。

4 F12 キーを押します。選択した出力形式に応じて、PDF、FlashPaper、RTF、XML、HTML、または Excel 形式のレポートが [プレビューレポート] ウィンドウに表示されます。

ページ番号の表示

Report Builder には、PAGE_NUMBER という名前の計算フィールドがあらかじめ用意されています。このフィールドをレポートバンドに配置すると、現在のページ番号が表示されます。

ビルトインの計算フィールドを追加するには

1 ツールボックスのフィールドツールをクリックします。

2 ヘッダバンドまたはフッタバンドの中央をクリック & ドラッグして、ページ番号フィールドのサイズを定義します。

[フィールドの追加] ダイアログボックスが表示され、ビルトインの計算フィールドと入力パラメータを含め、そのレポートで定義されているフィールドがすべて表示されます。

3 calc.PAGE_NUMBER を選択して [OK] をクリックします。

フィールドツールを使用すると、任意の種類のフィールド (クエリーフィールド、計算フィールド、入力パラメータ) をレポートに追加できます。

ビルトインの計算フィールドの詳細については、Report Builder オンラインヘルプを参照してください。

レイヤードコントロールの使用

レイヤードコントロールとは、レポートバンドの同じ位置に配置された複数の要素のことです。PrintWhen 式を使用して条件を指定することで、実行時に表示する要素を切り替えることができます。レイヤード要素を使用すると、表示する要素を状況に応じて変更できるので、重要な情報を伝達しやすくなります。

要素を別の要素の上に重ねて配置するには

1. バンド内に要素を配置します。
2. [ウィンドウ]-[プロパティ] を選択し、[プロパティ] パネルを表示します。
3. 各要素の [プロパティ] パネルを使用して次のように PrintWhen 式、表示プロパティ、および配置プロパティを指定します。
4. 各要素の PrintWhen 式を指定します。たとえば、次の式を指定すると、shippeddate (出荷日) が requireddate (期日) よりも後の場合 (つまり遅延の場合) は 2 つ目の要素が表示され、shippeddate が requireddate よりも前の場合は 1 つ目の要素が表示されます。

   1 つ目の要素 query.shippeddate LTE query.requireddate

   2 つ目の要素 query.shippeddate GT query.requireddate
5. 要素ごとに異なる表示特性を指定します。たとえば、注文処理が遅れているときは赤字で表示するように指定します。
6. 各要素の [上]、[左]、[高さ]、[幅] のプロパティを同じ値に設定します。

完全に一致する配置プロパティを指定した場合は、[レイヤードコントロール] メニューを使用して個々の要素にアクセスします。

[レイヤードコントロール] メニューを使用するには

1. 一番上の要素を右クリックします。
2. ポップアップメニューから [レイヤードコントロール]-[< 要素名 >] を選択します。それぞれのレイヤード要素を識別するために、各要素の PrintWhen 式が表示されます。
3. 要素を選択し、[ウィンドウ]-[プロパティインスペクタ] を選択して、要素のプロパティを表示します。

リンクの使用

ハイパーテキストリンクを使用すると、クエリーフィールド、計算フィールド、入力パラメータ、チャート、イメージを、次のものにリンクできます。

- 同じレポート内のアンカーまたはページ
- 別のレポート内のアンカーまたはページ
- HTML ページ。必要に応じてアンカーおよび URL パラメータを指定します。

リンクを使用すると、ドリルダウンレポートを作成できます。ドリルダウンレポートとは、特定の項目をクリックすると詳しい情報が表示されるレポートのことです。たとえば、従業員名をクリックすると、別のページに従業員 ID がパラメータとして渡され、従業員の詳細情報がそのページに表示されます。

アンカーとハイパーテキストリンクを作成する方法については、Report Builder オンラインヘルプを参照してください。

レポート要素のプロパティの定義

レポート自体を含め、レポート内の要素はすべて、一連のプロパティによって定義されます。これらのプロパティは、各要素の外観や動作に影響を与えます。

Report Builder のユーザーインターフェイス要素 (ダイアログボックス、ツールバーアイコン、メニューアイテムなど) を使用すると、ほとんどのプロパティ値を定義できます。たとえば、ツールバーアイコンを使用してテキストラベルのフォントサイズを設定できます。また、[プロパティ] パネルを使用すると、すべてのプロパティ値を設定できます。このパネルには、現在選択されている要素のプロパティがすべて表示されます。

レポート内に配置された要素の間隔が狭いために、個々の要素をマウスで選択することが難しい場合があります。その場合は、[プロパティ] パネルのポップアップメニューから要素を選択すると、簡単に特定の要素を選択できます。

[プロパティ] パネルには、次の 2 つのビューがあります。

[アルファベット順にソート] 現在選択されている要素のすべてのプロパティがアルファベット順に表示されます。

[グループ別にソート] 次のグループごとに、現在選択されている要素に関係するプロパティのみが [プロパティ] パネルに表示されます。

- 詳細設定
- 列
- ページレイアウト
- プリント
- カラーとスタイル
- データ
- フォント
- フォントスタイル
- 書式設定
- ハイパーリンク
- レイアウト
- 出力制御

現在選択されている要素に関係するグループのみが Report Builder に表示されます。

ワークスペース内の要素のプロパティを設定または変更するには

1. 要素を選択します。
2. (オプション) [プロパティ] パネルがまだ表示されていない場合は、[ウィンドウ]-[プロパティインスペクタ] を選択します。

   [プロパティ] パネルに要素のプロパティが表示されます。
3. プロパティを変更します。プロパティの種類に応じて、値を入力するか、ポップアップメニューから値を選択するか、式ビルダーを開いて式を入力します。
4. Enter キーを押します。

色を選択する場合は、色をダブルクリックします。

別の要素を選択するには

ポップアップメニューから要素を選択します。新しい要素を選択する場合は、Report Builder 内でその要素が選択され、その要素のプロパティが表示されます。

[プロパティ] パネルを使用するとすべてのプロパティを設定できますが、通常はダイアログボックスとツールバーアイコンを使用してプロパティを設定します。たとえば、レポート全体の設定を定義するには、[レポートプロパティ] ダイアログボックスを使用します。プロパティを設定する方法の詳細については、Report Builder オンラインヘルプの「プロパティリファレンス」を参照してください。

レポートの表示

アプリケーションからレポートを実行するには、CFR ファイルをブラウザで表示するか、そのレポートを実行する cfreport タグを含む CFM ページを表示します。

cfreport タグを使用してレポートをファイルに保存することもできます。

cfreport タグは、高度な PDF 暗号化オプションをサポートしています。詳細については、『CFML リファレンス』の cfreport を参照してください。

レポートプレビューの詳細については、996 ページの「レポートのプレビュー」を参照してください。

cfreport タグを使用してレポートを表示するには

1 内部クエリーを含む (または含まない) レポートを作成します。

2 CFM ページを作成し、レポートを実行する cfreport タグを追加します。レポートで内部クエリーが使用されていない場合は、query 属性を使用してクエリーを渡します。レポートで内部クエリーが使用されている場合に query 属性を使用すると、query 属性で渡されたクエリーが内部クエリーより優先されます。

```
<cfquery name="northwindemployees" datasource="localnorthwind">
    SELECT EmployeeID, LastName, FirstName, Title, City, Region, Country
    FROM Employees
    ORDER BY Country, City
</cfquery>

<CFREPORT format="PDF" template="EmpReport.cfr"
    query="#northwindemployees#"/>
```

注意：cfreport タグの前後にあるテキストはレンダリングされません。

3 ブラウザを開き、CFM ページを表示します。

ColdFusion によりレポートが生成されます。

HTML レポートを表示すると、レポート内のイメージを表示するための一時ファイルが生成されます。これらの一時ファイルをサーバー上に保存する期間を指定するには、cfreport タグの resourceTimespan 属性を使用します。詳細については、『CFML リファレンス』を参照してください。

CFR ファイルをブラウザで表示するには

1 内部クエリーを使用し、入力パラメータを使用しないレポートを作成します。

2 ブラウザを開き、CFR ファイルを表示します。

レポートをファイルに保存するには

1 内部クエリーを含む (または含まない) レポートを作成します。

2 CFM ページを作成し、レポートを実行する cfreport タグを追加します。必要に応じて、前の手順で説明したとおりに query 属性を渡します。次の例に示すように、作成するファイルの完全修飾名を filename 属性で指定します。

```
<CFREPORT format="PDF" template="emppicture.cfr"
    filename="#GetDirectoryFromPath(GetTemplatePath())#/emppicture.pdf"
    overwrite="yes"/>
```

レポート出力を HTML ファイルに書き込むと、その HTML ファイルを基準とする場所に新しいディレクトリが作成され、レポート内のイメージ (チャートなど) を表示するために生成されたイメージファイルがそのディレクトリに保存されます。詳細については、1018 ページの「HTML でのレポートのエクスポート」を参照してください。

出力ファイルの形式を PDF 形式にする場合は拡張子 .pdf を、FlashPaper 形式にする場合は拡張子 .swf を、XML 形式にする場合は拡張子 .xml を、RTF 形式にする場合は拡張子 .rtf を、HTML 形式にする場合は拡張子 .html を、Excel 形式にする場合は拡張子 .xls を使用します。

3 ブラウザを開き、CFM ページを表示します。ColdFusion によりレポートが生成され、ファイルに保存されます。ブラウザには空のページが表示されます。

ブラウザでの CFR ファイルの表示を無効にするには

1 メニューバーから [レポート]-[レポートプロパティ] を選択して、[レポートプロパティ] ダイアログボックスを開きます。

2 [.cfr ブラウザ呼び出しの許可] オプションのチェックマークをはずして、[OK] をクリックします。

実行時に変数などのデータを渡す入力パラメータの使用

入力パラメータは、実行時にレポートに渡されるデータフィールドです。入力パラメータは、レポートバンドに直接配置することも、計算フィールドへの入力として使用することもできます。

クエリーフィールドと同じ方法で入力パラメータを定義します。対応するパラメータが存在しない場合に使用するデフォルト値を指定できます。入力パラメータを定義する方法の詳細については、992 ページの「フィールドと入力パラメータの定義、変更、使用」を参照してください。

入力パラメータは次の方法で使用できます。

- **cfreportparam タグを使用する** : 入力パラメータの名前と、CFM ページの呼び出しに含まれる cfreportparam タグで指定された名前が一致している必要があります。たとえば、ReportTime という名前の入力パラメータを定義する場合は、次の例に示すように、name 属性を ReportTime に設定した cfreportparam タグを渡します。

```
<cfreport format="PDF" template="FourthReport.cfr" query="#coursedept#">
    <cfreportparam name="ReportTime" value="#DateFormat(Now())#, #TimeFormat(Now())#">
</cfreport>
```

- **サブレポートパラメータを使用する** : メインレポートからサブレポートに情報を渡す必要がある場合は、メインレポートでサブレポートパラメータを定義し、対応する入力パラメータをサブレポートで定義します。詳細については、1005 ページの「サブレポートの使用」を参照してください。

実行時に入力パラメータの内容をダイナミックに決定する方法については、1002 ページの「拡張クエリーモード」を参照してください。

レポートでの CFML の使用

CFML は Report Builder 用のスクリプト言語です。CFML を利用すると、ユーザーのニーズに応じてレポートのデータを選択し、適切な書式を適用することができます。Report Builder では、次の領域で CFML を使用できます。

- 拡張クエリーモード
- レポート関数
- 式

拡張クエリーモード

複雑なクエリーを作成したり、既存のクエリーを再利用したり、追加の CFML 処理をクエリー作成の一環としてカプセル化したりすることが必要になる場合があります。そのような場合は、拡張クエリーモードを使用して、クエリーを返す CFML を作成します。クエリービルダーの上部にある [詳細設定] ボタンをクリックすると、Report Builder にテキスト入力領域が表示されます。ここに、クエリーを生成する CFML を入力できます。ColdFusion はレポート実行時にこのタグを実行し、クエリーの結果セットをレポートに渡します。

注意：拡張クエリーモードを使用する場合、クエリーフィールドは自動的に作成されません。関連するクエリーフィールドを手動で作成してください。

拡張クエリーモードで使用する CFML には、クエリーオブジェクトフィールドを含む変数と名前が一致するクエリーオブジェクトが含まれている必要があります。クエリーオブジェクトを返す任意の CFML タグまたは QueryNew 関数を使用できます。CFML では、複数のクエリーオブジェクトを使用できますが、返すことができるクエリーオブジェクトは 1 つだけです。

注意：空の変数 (たとえば <cfset name="">) を設定すると、Report Builder でレポートデータバインドエラーが発生します。

次に示す CFML は、cfhttp タグを使用してクエリーを取得します。

```
<cfhttp url="http://quote.yahoo.com/download/quotes.csv?Symbols=csco,jnpr&format=sc1l1&ext=.csv"
method="GET"
name="qStockItems"
columns="Symbol,Change,LastTradedPrice"
textqualifier="""""
delimiter=","
firstrowasheaders="no">
```

また、拡張クエリーモードでは、次の例に示すように、URL または FORM スコープにパラメータが存在するかどうかを調べたり、これらのパラメータを使用してデータを取得したりすることができます。

```
<!--- First look for URL param. URL overrides cfreportparam. --->
<cfif isDefined("url.deptidin")>
    <cfset param.deptidin = url.deptidin>
</cfif>

<!-- Then look for FORM param. Overrides URL param. --->
<cfif isDefined("form.deptidin")>
    <cfset param.deptidin = form.deptidin>
</cfif>

<cfquery name="CFReportDataQuery" datasource="cfdocexamples">
SELECTLastName, FirstName, Dept_ID
FROMEmployee
WHERE (Dept_ID = #param.deptidin#)
</cfquery>
```

レポート関数の使用

レポート関数とは、レポート関数エディタを使用してコーディングできるユーザー定義の CFML 関数です。これらの関数は、レポートフィールド内で実行できます。レポート関数を使用すると、データの書式を設定したり (たとえば、完全な住所を表示するために複数のフィールドを連結して書式を設定したり)、データを取得したりすることができます。

InitializeReport、BeforeExport、および FinalizeReport は Report Builder 固有のビルトイン関数です。詳細については、Report Builder オンラインヘルプを参照してください。

Report Builder ビルトイン関数

1 メニューバーから [レポート]-[レポート関数] を選択します。

レポート関数エディタが表示されます。

2 左端にある [デフォルト関数の追加] アイコンをクリックします。

左ペインにビルトイン関数が追加されます。

3 左ペインで関数を選択します。

その関数のコメント付きコードが右ペインに表示されます。

4 コードを修正して、[OK] をクリックします。

レポート関数を作成するには

1 メニューバーから [レポート]-[レポート関数] を選択します。

レポート関数エディタが表示されます。

2 プラス記号をクリックして新しいレポート関数を追加します。

[レポート関数の追加] ダイアログボックスが表示されます。

3 名前を指定して、[OK] をクリックします。

4 レポート関数エディタのテキスト入力領域に、cfreturn タグが配置されます。

5 関数をコーディングして [OK] をクリックします。これは ColdFusion のユーザー定義関数であり、すべての UDF ルールと機能を使用できます。次に示すレポート関数は、住所に関連する複数のフィールドを連結します。

```
 <cfargument name="Name" required="yes"/>
<cfargument name="Address1" required="yes"/>
<cfargument name="Address2" required="yes"/>
<cfargument name="City" required="yes"/>
<cfargument name="State" required="yes"/>
<cfargument name="Zip" required="yes"/>

<cfset variables.CRLF = Chr(13) & Chr(10)>
<cfset variables.ResultVar="">

<cfif Trim(arguments.Name) NEQ "">
    <cfset variables.ResultVar='#arguments.Name#'>
</cfif>
<cfif Trim(arguments.Address1) NEQ "">
    <cfif variables.ResultVar NEQ "">
        <cfset variables.ResultVar='#variables.ResultVar & variables.CRLF#'>
    </cfif>
    <cfset variables.ResultVar='#variables.ResultVar & arguments.Address1#'>
</cfif>
<cfif Trim(arguments.Address2) NEQ "">
    <cfif variables.ResultVar NEQ "">
        <cfset variables.ResultVar='#variables.ResultVar & variables.CRLF#'>
    </cfif>
    <cfset variables.ResultVar='#variables.ResultVar & arguments.Address2#'>
</cfif>
<cfif variables.ResultVar NEQ "">
    <cfset variables.ResultVar='#variables.ResultVar & variables.CRLF#'>
</cfif>

<cfset variables.ResultVar='#variables.ResultVar & arguments.City & ", " & arguments.State & " " &
arguments.Zip#'>

<cfreturn variables.ResultVar>
```

レポート関数を使用するには

1 適切なレポートバンドにダイナミックフィールドを配置します。

[フィールドの追加] ダイアログボックスが表示されます。

2 [手動で入力される式] を指定して、[OK] をクリックします。

式ビルダーが表示されます。

3 "report.< 関数名 >" を指定して、[OK] をクリックします。

式の使用

Report Builder の多くの要素 (クエリーフィールド、計算フィールド、入力パラメータ、イメージ、レポートオブジェクト属性など) は、オペランドを 1 つだけ取る ColdFusion 式です。これらの要素は式なので、CFML 関数を使用して操作できます。

式ビルダーは、Report Builder 要素に CFML 関数をすばやく適用するための GUI です。式ビルダーには、次のような用途があります。

- 多くのレポートオブジェクト属性 (PrintWhen など) では、式を使用して、クエリーパラメータ、入力パラメータ、ColdFusion ページ変数などに属性を関連付けることができます。レポート属性には、実行時のデータやユーザーの設定に基づいて表示する列を関連付けることができます。
- フィールドの連結
- フィールドの書式設定
- 計算フィールド
- ColdFusion ページ変数およびスコープへのアクセスと表示

式ビルダーの使用方法の詳細については、Report Builder オンラインヘルプを参照してください。

式の詳細については、58 ページの「式と # 記号の使用」を参照してください。

チャートの使用

チャートを使用すると、大量のデータや複雑なデータを理解しやすくなります。Report Builder を使用すると、任意のレポートバンドにチャートを配置できます。Report Builder ではさまざまな種類のチャートがサポートされています。

レポートにチャートを追加するには、チャート作成手順を案内するチャートウィザードを使用します。チャートウィザードはクエリーウィザードに完全に統合されています。チャートタイプ、レポートで使用するデータ、書式設定オプションなどをチャートウィザードで定義することにより、データベースと連動するチャートを簡単に作成できます。

チャートウィザードでチャートの特性を定義すると、RDS を使用してチャートイメージがリアルタイムで生成されます。ただし、これらのチャートイメージで使用されるデータは、実際のデータではありません。

チャートウィザードには、次のパネルがあります。

- **[チャートタイプ]** : チャートタイプ (棒グラフなど) とサブタイプ (3D スタックなど) を選択します。
- **[チャート系列]** : 系列のデータを選択します。系列を追加する場合は、系列データをハードコードするか、系列データを取得するためのデータベースクエリーをクエリービルダーで作成できます。
- **[チャートの形式設定]** : タイトルと系列、一般表示、3D 表示、線とマーカー、フォントを指定します。

チャートウィザードで指定するデータは、cfchart タグ、cfchartseries タグ、および cfchartdata タグで指定する属性に対応します。これらのタグの詳細については、『CFML リファレンス』を参照してください。

ColdFusion のチャート作成機能の詳細については、955 ページの「チャートとグラフの作成」を参照してください。
Report Builder を使用したチャート作成方法の詳細については、Report Builder オンラインヘルプを参照してください。

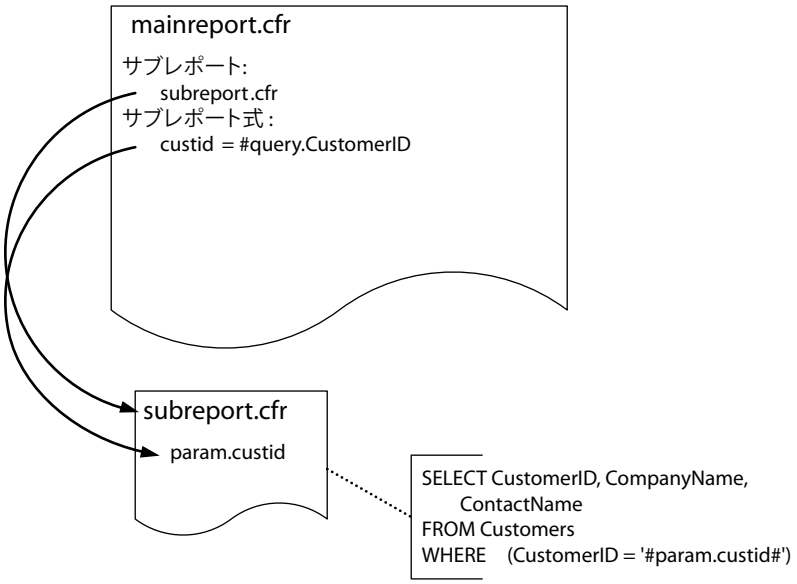
サブレポートの使用

サブレポートを使用すると、レポート内に別のレポートをネストできます。多くの場合、サブレポートのデータはメインレポートのデータに関係しています。メインレポートとサブレポートのデータを関連付けるには、サブレポートにサブレポートパラメータを渡します。ただし、サブレポートのデータは、必ずしもメインレポートのデータに関係していなくてもかまいません。

サブレポートは次のような場合に使用します。

- RIGHT OUTER JOIN などの複雑な SQL を使用したくない場合
- 複数のデータベースからデータを取得する必要がある場合

次の例は、サブレポートパラメータの使用方法と、レポートとサブレポートの関係を示しています。



注意：Report Builder は複数のネストレベルをサポートしますが、表示されるネストレベルは 1 つのみです。

サブレポートの詳細については、Report Builder オンラインヘルプを参照してください。

サブレポートの定義

サブレポートは、メインレポートに挿入する前に定義することもできますが、メインレポートに挿入するときに定義することもできます。

サブレポートには次の特徴があります。

- データは詳細バンド内にのみ表示されます。サブレポートではヘッダバンドとフッタバンドを使用しません。
- サブレポートをメインレポートに関連付けるには、サブレポートに内部クエリーを挿入し、メインレポートの [サブレポート式] プロパティで使用されている入力パラメータの名前を指定する WHERE 節を含んだ SELECT ステートメントを使用する必要があります。

定義済みのサブレポートがある場合は、そのサブレポートをメインレポートに追加し、必要に応じてサブレポートパラメータを定義します。

既存のサブレポートを追加するには

1 新規のメインレポートを定義するか、既存のメインレポートを開きます。

2 ツールボックスの [サブレポート] アイコンをクリックします。

3 適切なレポートバンド内をクリック & ドラッグして、サブレポートの領域を定義します。

4 [既存のレポートから] を選択し、サブレポートを指定して、[次へ] をクリックします。

5 サブレポートのフィールドに対応するメインレポートのフィールドを選択し、[次へ] をクリックします。

6 「終了」をクリックします。

メインレポートにサブレポートが追加され、レポートとサブレポートのマッピングがサブレポートパラメータとして保存されます。

7 サブレポートパラメータの設定を変更するには、サブレポートを選択し、[プロパティ] パネルで [サブレポートパラメータ] をクリックします。

サブレポートで必要になるデータが事前にわかっている場合は、メインレポートに新規のサブレポートを追加して、その場でサブレポートを定義できます。

新規のサブレポートを追加するには

1 新規のメインレポートを定義するか、既存のメインレポートを開きます。

2 ツールボックスの [サブレポート] アイコンをクリックします。

3 レポートバンド内をクリック & ドラッグして、サブレポートの領域を定義します。

4 [新規レポートとして] を選択して、[次へ] をクリックします。

5 [クエリービルダー] をクリックします。

6 サブレポート用のテーブルと列を選択します。

7 [条件] 列と [基準] 列をキー列として使用して、レポートの WHERE 節を指定します。

[条件] で WHERE を指定し、[基準] で ='#CFVariable#' (文字列型の列) または =#CFVariable# (数値型の列) を指定して、CFVariable をサブレポートの入力パラメータの名前に変更します (入力パラメータ名は後で定義します)。

8 [保存] をクリックして、[次へ] をクリックします。

9 必要に応じて、グループ化に関連するフィールドを指定し、[次へ] をクリックします。

10 [Free Form] または [Grid] を指定して、[次へ] をクリックします。

11 [Only Detail Band] を指定して、[次へ] をクリックします。

12 カラースキームを指定して、[次へ] をクリックします。

13 必要に応じて見出しを指定し、[次へ] をクリックします。

14 サブレポートで必要なパラメータごとに、次の項目を指定します。

- パラメータ名
- 関連付けるメインレポートの値 (ポップアップメニューから選択します)
- データ型

15 「次へ」をクリックします。

16 サブレポートの完全修飾ファイル名を指定して、[次へ] をクリックします。

17 「終了」をクリックします。

メインレポートにサブレポートが追加されます。Report Builder を使用すると、サブレポートの名前やメインレポートのサブレポートパラメータを変更できます。

サブレポートの設定を変更するには

1 メインレポート内のサブレポート要素をクリックします。

2 サブレポートを変更するには、[サブレポート式] の値を変更します。

3 サブレポートパラメータを変更するには :

a [サブレポートパラメータ] プロパティをクリックします。

b [...] ボタンをクリックします。

c サブレポートパラメータを追加、変更、または削除して、[OK] をクリックします。

簡単なレポートの作成

次の例では、レポートウィザードを使用して簡単なレポートを作成した後、レポートに変更を加えます。この例では、ColdFusion に付属する cfartgallery データベースを使用します。

この例では、次のタスクを行います。

- レポートウィザードとクエリービルダーを使用してベースレポートを作成する。
- 式ビルダーを使用して、レポート内のデータ表現を変更する。
- 列データの表示テキストを変更する。
- レポートにテキストフィールドを追加し、レポートスタイルを使用してテキスト要素とデータ要素の形式を設定する。
- データベースからイメージファイルとイメージを追加する。
- 計算フィールドを作成して追加し、アーティスト別の総売上高を表示する。
- グループレベルの円グラフとレポートレベルの円グラフを追加し、データベース内の各アーティストと全アーティストに関して、売却済みの作品と未売却の作品の比率を表示する。
- レポートスタイルを CSS (Cascading Style Sheet) ファイルにエクスポートする。

レポートウィザードを使用してレポートを作成するには

1 Report Builder を起動します。

2 レポート作成ウィザードを選択して [OK] をクリックします。

3 [クエリービルダー] ボタンをクリックします。

注意：RDS が既に設定されている場合は、RDS のパスワードを指定します。

a データベースペインのデータソースリストで、cfartgallery データベースを展開します。

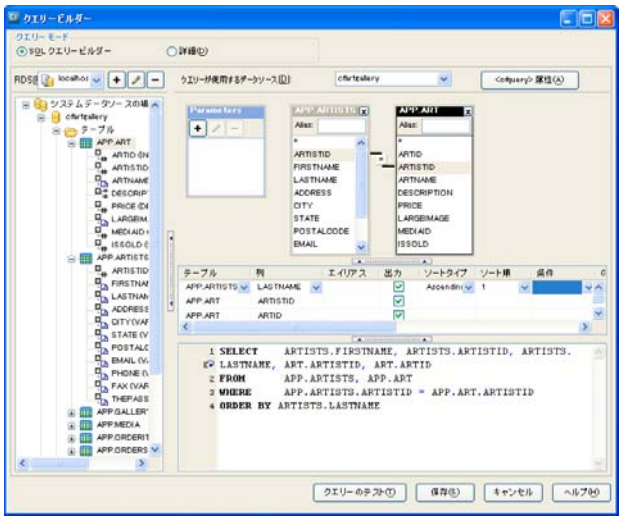
b " テーブル " フォルダを展開します。

c データベースペインで [APP.ART] テーブルをダブルクリックします。テーブルペインに [APP.ART] テーブルが追加されます。

d データベースペインで [APP.ARTISTS] テーブルをダブルクリックします。テーブルペインに [APP.ARTISTS] テーブルが追加されます。[ARTISTID] 列を基準に、これら 2 つのテーブル間の結合が自動的に作成されます。

e [APP.ARTISTS] テーブルで、[FIRSTNAME] 列と [LASTNAME] 列をダブルクリックします。SQL ペイン内の select ステートメントにフィールドが追加されます。

f [ART] テーブルで、[ARTNAME] 列、[DESCRIPTION] 列、[PRICE] 列、[ISSOLD] 列をダブルクリックします。完成したクエリーをクエリービルダーで表示すると、次のようになります。



g [クエリーのテスト] ボタンをクリックして、結果をプレビューします。

h テストクエリーウィンドウを閉じ、クエリービルダーウィンドウで [保存] ボタンをクリックします。

4 [FIRSTNAME] 列をダブルクリックして [出力されないフィールド] ポップアップメニューにその列を追加し、[次へ] ボタンをクリックします。

5 [使用可能なフィールド] リストで [LASTNAME] をダブルクリックし、アーティストの姓ごとにレコードをグループ化します。

6 デフォルト値を変更せずに [次へ] ボタンを 3 回クリックします。

7 [Silver] を選択し、[次へ] ボタンをクリックします。

8 レポートのタイトルを「売上げレポート」に変更して、[終了] ボタンをクリックします。レポート作成ウィザードによりレポートが生成され、Report Builder のワークスペースに表示されます。

9 [ファイル]-[新規保存] を選択し、デフォルトのディレクトリに「ArtSalesReport1」という名前でレポートを保存します。このファイルには、自動的に CFR という拡張子が追加されます。

10 F12 キーを押してレポートをプレビューします。アーティストの姓ごとにグループ化されたレコードが表示されます。

11 終了ボックスをクリックして [プレビューレポート] ウィンドウを閉じ、Report Builder のワークスペースに戻ります。

列見出しラベルの変更

レポートウィザードで作成したレポートの列ヘッダには、デフォルトで列の名前が使用されますが、列見出しのラベルテキストは変更できます。

見出しのラベルテキストを編集するには

1 [列ヘッダ] バンドの [LASTNAME] フィールドをダブルクリックします。

2 列名を「アーティスト名」に変更して、[OK] をクリックします。

3 残りの列ラベルを次のように変更します。

- ARTNAME > タイトル
- DESCRIPTION > 説明

- PRICE > 価格
- ISSOLD > 売却済みか ?

式を使用したデータの書式設定

式ビルダーを使用して、次のタスクを行います。

- ISSOLD の値の表示を yes/no 式に変更する。デフォルトでは、データベースに格納されているデータに基づいて、[ISSOLD] 列には 0 (未売却) または 1 (売却済み) が表示されます。この表示を「yes」または「no」に変更するには、関数を使用します。
- [PRICE] 列の値をドル形式に変更する。
- アーティストの姓と名を連結する。[FirstName] はレポートに出力されないフィールドですが、作成した SQL クエリーの一部なので、式に追加できます。

ブール値を yes/no に変更する

1 詳細バンドで、query.ISSOLD 要素をダブルクリックします。その要素の式ビルダーが表示されます。

2 式ビルダーで、"Functions" フォルダを展開します。

3 [Functions] リストで [Display and Formatting] を選択します。式ビルダーの右ペインに関数のリストが表示されます。

4 関数のリストで [YesNoFormat] をダブルクリックします。式ペインに、次の式が自動的に入力されます。

```
YesNoFormat(query.ISSOLD)
```

5 [OK] をクリックして式ビルダーを閉じ、レポートに戻ります。

6 [ファイル]-[保存] を選択してレポートを保存します。

7 F12 キーを押してレポートをプレビューします。作品が売却済みかどうかに基づき、[売却済みか ?] 列に「yes」または「no」が表示されます。

ドル形式で数値を表示するには

1 詳細バンドの [PRICE] 列で、フィールドをダブルクリックします。

2 式ペインに表示された式を次のテキストに変更します。

```
DollarFormat(query.PRICE)
```

3 [OK] をクリックして式ビルダーを閉じ、レポートに戻ります。

[FIRSTNAME] フィールドと [LASTNAME] フィールドを連結するには

1 [LASTNAME] グループヘッダの [query.LASTNAME] フィールドをダブルクリックします。

2 式ビルダーで、次の式を入力します。

```
query.FIRSTNAME &" "& query.LASTNAME
```

式の入力中に、使用可能なフィールド名が提示されます。

3 式ビルダーで [OK] ボタンをクリックします。

4 Report Builder のメニューバーで [ファイル]-[保存] を選択してレポートを保存します。

5 F12 キーを押してレポートをプレビューします。

各アーティストの姓と名が表示されます。この段階でもアーティストのデータは姓のアルファベット順でグループ化されています。

6 プレビューウィンドウを閉じます。

グループ変更前の改ページの追加

各アーティストの名前がページの先頭に表示されるように改ページを挿入します。

アーティスト名の間に改ページを追加するには

1 メインメニューバーから [レポート]-[グループ管理] を選択します。[LASTNAME] が選択された状態で [グループ管理] ダイアログボックスが表示されます。

2 [編集] ボタンをクリックします。

3 [新規ページの開始] オプションを選択して、[OK] をクリックします。

計算フィールドの追加

アーティストごとの売上額を計算するには

1 [ウィンドウ]-[フィールドとパラメータ] を選択します。

2 [フィールドとパラメータ] パネルが表示されます。

3 計算フィールドのリストを展開します。

4 [計算されるフィールド] を選択して、[フィールドとパラメータ] パネルの上部にある (+) ボタンをクリックします。

5 [計算されるフィールドの追加] ダイアログボックスで、次のようにプロパティを変更します。

   a 計算フィールドの名前を「**Sold**」に変更します。

   b ラベルテキストを「**売却済み**」に変更します。

   c [データタイプ] を [フロート] に変更します。

   d [計算] を [合計] に変更します。

   e [計算の対象] フィールドに、次の式を入力します。

```
Iif(IsBoolean(query.ISSOLD) and query.ISSOLD, query.Price,0)
```

   この式は、各アーティストの作品の価格に売上数を掛けて、アーティストごとの総売上高を算出します。レコードの ISSOLD 値が 1 (売却済み) である場合は、価格に 1 を掛けた値が合計に加算されます。ISSOLD 値が 0 (未売却) である場合は、価格に 0 を掛けた値が合計に加算されます。

   f [フィールドをリセット] の値を [**グループ**] に変更します。

   g [リセットするグループ] の値を「**LASTNAME**」に変更して、[OK] をクリックします。[フィールドとパラメータ] パネルに、計算フィールドの定義が追加されます。

計算フィールドをレポートに追加するには

1 LASTNAME フッタバンドにフィールドを挿入します。

2 [フィールドの追加] ダイアログボックスで、ポップアップメニューから [calc.Sold] を選択します。

3 式ビルダーで、次のコードを入力します。

```
DollarFormat(calc.Sold)
```

4 F12 キーを押してレポートをプレビューします。各アーティストの売却済み作品の総額が表示されます。

フィールドの追加と書式設定

レポートにテキストフィールドを追加して、フィールドのスタイルを定義できます。定義したスタイルは、同じレポート内で再利用することも、エクスポートして他のレポートで使用することもできます。また、レポートスタイルは、cfreport タグと cfreportparam タグを使用して実行時に置き換えることもできます。詳細については、1019 ページの「レポートスタイルの置き換え」を参照してください。

テキストフィールドを追加するには

1. Report Builder ウィンドウの左側にある [コントロール] ツールボックスで、テキストアイコン (abc と表示されているボタン) をクリックして、LASTNAME フッタにある計算フィールドの左側にテキストフィールドを配置します。
2. [ラベルテキストの編集] ダイアログボックスに「**総売上高**」と入力し、[OK] をクリックします。

スタイルを作成するには

1. メインメニューから [ウィンドウ]-[レポートスタイル] を選択します。
2. (+) ボタンをクリックします。
3. [名前] フィールドに、「GroupFooter」と入力します。
4. [カラーとスタイル] タブをクリックして、カラーを **#9999CC** に変更します。
5. [フォント] タブをクリックして、フォントを **MS P ゴシック**に変更し、太字オプションをクリックします。[OK] をクリックします。レポート内で使用可能なスタイルのポップアップメニューに GroupFooter スタイルが追加されます。
6. メニューバーで [ファイル]-[保存] を選択してレポートを保存します。

レポート内のテキスト要素とデータ要素にスタイルを適用するには

1. LASTNAME フッタバンドで [総売上高] テキストボックスを選択します。
2. [ウィンドウ]-[プロパティインスペクタ] を選択します。
3. [スタイル] ポップアップメニューで [GroupFooter] を選択します。
4. 計算フィールド要素を選択して、GroupFooter スタイルを適用します。
5. F12 キーを押してレポートをプレビューします。

イメージの追加

Report Builder でイメージを追加するときには、次のようなタスクを実行できます。

- レポートヘッダにある会社名のテキストボックスを会社のロゴに置き換える。
- クエリービルダーを使用してデータベースからイメージファイルを追加する。
- 表示を高速化するためにレポートを RTF 形式で表示する。

レポートヘッダにロゴを追加するには

1. 売上げレポートの上部にあるヘッダバンドで、[CompanyName] テキストボックスを選択します。
2. [編集]-[カット] を選択してレポートからテキストボックスを削除します。
3. [コントロール] ツールボックスのイメージ追加アイコンをクリックします (このアイコンには、木の絵が描かれています)。
4. [売上げレポート] テキストボックスの上部にあるヘッダバンドで、マウスをドラッグします。マウスボタンを離すと、[Image File Name] ダイアログボックスが表示されます。
5. 次の場所にある Art World ロゴファイルを参照します。

   C:¥ColdFusion9¥wwwroot¥cfdocs¥getting_started¥photos¥somewhere.jpg

6 [開く] をクリックします。選択した領域に Art World ロゴが表示されます。

7 ワークスペースでイメージを選択した状態で、[ウィンドウ]-[プロパティインスペクタ] を選択します。イメージのプロパティインスペクタが表示されます。

   a [カラーとスタイル] で、[透明] の値を [透明] に変更します。

   b [書式設定] で、[イメージのスケール] の値を [元のサイズを維持] に変更します。

8 ヘッダバンドで、Ctrl キーを押しながらワークスペース内のロゴイメージと [売上げレポート] テキストボックスをクリックして選択します。

9 [コントロール] ツールボックスの左端揃えアイコンをクリックします。

10 [ファイル]-[保存] を選択して変更を保存します。

11 F12 キーを押してレポートをプレビューします。

12 プレビューウィンドウを閉じ、必要に応じてイメージのサイズと位置を再調整します。

データベースからイメージを追加するには

1 メニューバーから [レポート]-[レポートクエリー] を選択します。

2 [Art] テーブルで、[LARGEIMAGE] をダブルクリックします。select ステートメントに [LARGEIMAGE] 列が追加されます。

3 [クエリーのテスト] ボタンをクリックします。[ISSOLD] 列の右側に、イメージファイル名のリストが表示されます。

4 [テストクエリー] ウィンドウを閉じ、クエリービルダーで [保存] ボタンをクリックします。

5 [レポート] ウィンドウで、[詳細] バンドの下側にあるスプリッタバーをクリックし、下方にドラッグして [詳細] バンドを拡大します。

6 [コントロール] ツールボックスでイメージ追加アイコンをクリックし、[query.ARTNAME] フィールドの左側にあるレポートの [詳細] バンドでマウスをドラッグします。マウスボタンを離すと、[Image File Name] ダイアログボックスが表示されます。

7 次の場所にある cfartgallery イメージディレクトリを参照します。

   C:¥ColdFusion9¥wwwroot¥cfdocs¥images¥artgallery

8 [ファイル名] フィールドに「**#query.largeimage#**」と入力します。

9 [開く] ボタンをクリックします。レポートの [詳細] バンドに列が追加されます。

10 [詳細] バンドの上部にイメージ列を整列します。

11 [詳細] バンドでイメージ要素を選択した状態で、[ウィンドウ]-[プロパティインスペクタ] を選択します。

12 次のプロパティを変更します。

   a [透明] : [透明]。

   b [イメージのスケール] : [元のサイズを維持]。このオプションを選択すると、縦横比を維持したまま境界ボックス内でイメージのサイズが変更されます。

   c [Error Control] : [No Image]。このオプションを選択すると、データベースに存在しないイメージが指定されている場合に、エラーが表示される代わりに、空のイメージが表示されます。

   d [キャッシュの使用] : [False]。このオプションを選択すると、ブラウザでレポート出力のプレビューが実行されるたびに表示が更新されます。

13 [ファイル]-[保存] を選択して変更を保存します。

レポート出力の形式を変更するには

1 メニューバーから [レポート]-[レポートプロパティ] を選択します。

2 [デフォルトの出力形式] ポップアップメニューで [RTF] を選択します。この形式を使用すると、Web ブラウザでの表示が高速になります。

3 [OK] をクリックして [レポートプロパティ] ダイアログボックスを閉じ、レポートに戻ります。

4 [ファイル]-[保存] を選択して変更を保存します。

5 F12 キーを押してレポートをプレビューします。アーティスト名の下 (作品タイトルの左) にイメージが表示されます。

6 [デフォルトの出力形式] を [HTML] に変更して、結果をプレビューします。

チャートの追加

チャートビルダーを使用してレポートに円グラフを追加できます。最初の円グラフでは、各アーティストの売却済み作品の合計額 (ドル単位) と未売却作品の合計額を表示します。2 つ目の円グラフでは、全アーティストの売却済み作品の合計額と未売却作品の合計額を表示します。

適用範囲を除けば、これら 2 つの円グラフは同じです。各アーティストの売却済み作品と未売却作品の比率を示す円グラフをグループに適用するには、グループのフッタバンドに円グラフを追加します。全アーティストの売却済み作品と未売却作品の比率を示す円グラフをレポートに適用するには、レポートのフッタバンドに円グラフを追加します。

売却済み作品の合計額を算出する計算フィールドは、1010 ページの「計算フィールドの追加」で既に追加されています。この例で使用する円グラフを作成する前に、未売却作品の合計額を算出する 2 つ目の計算フィールドを作成してください。

未売却作品の合計額を算出する計算フィールドを追加するには

1 [ウィンドウ]-[フィールドとパラメータ] を選択します。

2 [フィールドとパラメータ] パネルで、[計算されるフィールド] 見出しを選択します。

3 パネルの一番上にある (+) アイコンをクリックします。

   a [名前] フィールドに、「**Unsold**」と入力します。

   b [デフォルトラベル] フィールドに、「**未売却**」と入力します。

   c [データタイプ] フィールドで、ポップアップメニューから [Big Decimal] を選択します。

   d [計算] フィールドで、ポップアップメニューから [合計] を選択します。

   e [計算の対象] フィールドに、未売却作品の合計額を算出する次の式を入力します。

```
Iif(IsBoolean(query.ISSOLD) and not(query.ISSOLD), query.Price,0)
```

   f [フィールドをリセット] フィールドで、ポップアップメニューから [グループ] を選択します。

   g [リセットするグループ] フィールドで、[LASTNAME] を選択します。

   h [OK] をクリックして [計算されるフィールドの追加] ダイアログボックスを閉じ、レポートに戻ります。

4 メニューバーで [ファイル]-[保存] を選択してレポートを保存します。

グループフッタに円グラフを追加するには

1 LASTNAME フッタバンドを展開します。

2 Report Builder のメニューバーから [挿入]-[チャート] を選択します。

   a [基本チャートタイプ] リストで [円] を選択します。[基本チャートタイプ] の右側に [チャートサブタイプ] が表示されます。

   b 3D チャートを選択します。

3 [次へ] ボタンをクリックします。次に、[追加] ボタンをクリックします。

a [系列ラベル] フィールドに、「**総売上高**」と入力します。

b [ペイントスタイル] フィールドで、[極細] を選択します。

c [データラベル] フィールドで、[値] を選択します。

d [カラーリスト] で、「**Teal,Gray**」と入力します。

e [チャートデータソース] 領域で、[値の固定リストからのデータ] オプションがオンになっていることを確認します。

4 [追加] ボタンをクリックします。

a [ラベル] フィールドに、「**売却済み**」と入力します。

b [値] フィールドで、ポップアップメニューから [#calc.Sold#] を選択します。

c 「OK」をクリックします。

5 [追加] ボタンをもう一度クリックします。

a [ラベル] フィールドに、「**未売却**」と入力します。

b [値] フィールドで、ポップアップメニューから [#calc.Unsold#] を選択します。

c [OK] を 2 回クリックし、[チャート系列] ダイアログボックスに戻ります。

6 [次へ] ボタンをクリックします。[チャートの形式設定] ダイアログボックスで [タイトルと系列] タブをクリックし、次のプロパティを変更します。

a #query.LASTNAME# の [チャートタイトル] フィールドに、「**総売上高**」と入力します。

b [X 軸のタイトル] フィールドに、「**売却済み**」と入力します。

c [Y 軸のタイトル] フィールドに、「**未売却**」と入力します。

d [ラベルの形式] フィールドで、ポップアップメニューから [Currency] を選択します。

e [3D 表示] タブをクリックし、[3D の表示] が選択されていることを確認します。

7 [フォント] タブをクリックし、次のプロパティを変更します。

a [フォント名] を Arial に変更します。

b [フォントサイズ] を 9 に変更します。

8 [終了] ボタンをクリックします。レポートに円グラフ用のプレースホルダーが追加されます。

9 LASTNAME フッタバンド内で円グラフのサイズを変更し、適切な位置に移動します。

10 [ファイル]-[保存] を選択してレポートを保存します。

11 F12 キーを押してレポートをプレビューします。

レポートフッタに円グラフを追加するには

1 レポートフッタの円グラフで使用する 2 つの計算フィールドを作成し、次のパラメータを設定します。

名前	**TotalSold**	**TotalUnsold**
デフォルトラベル :	売却済み総額	未売却総額
データタイプ :	Big Decimal	Big Decimal
計算 :	合計	合計
計算の対象 :	Iif(IsBoolean(query.ISSOLD) and query.ISSOLD, query.Price,0)	Iif(IsBoolean(query.ISSOLD) and not(query.ISSOLD), query.Price,0)

名前	TotalSold	TotalUnsold
初期値 :	0	0
フィールドをリセット :	レポート	レポート
リセットするグループ :	LASTNAME	LASTNAME

2 [ページフッタ] バンドの直下にある [レポートフッタ] バンドを展開します。

3 [グループフッタ] から円グループをコピーし、[レポートフッタ] に貼り付けます。

4 円グラフをダブルクリックし、[次へ] ボタンをクリックします。

5 [Total Sales] をダブルクリックし、[チャート系列の編集] ダイアログボックスを表示します。

6 [系列ラベル] を「アーティストの総売上高」に変更します。

7 チャート系列の値を変更します。

ラベル	値
売却済み	#calc.TotalSold#
未売却	#calc.TotalUnsold#

8 [次へ] ボタンをクリックし、[タイトルと系列] タブをクリックします。

9 [チャートタイトル] を「**アーティストの総売上高**」に変更し、[終了] をクリックします。

10 メニューバーで [ファイル]-[保存] を選択してレポートを保存します。

11 F12 キーを押してレポートをプレビューします。

「アーティストの総売上高」というタイトルの円グラフは、レポートの最後のページにのみ表示されます。計算が正しいことを確認してください。

CSS (Cascading Style Sheet) の使用

レポート作成ウィザードを実行すると、次のスタイルが自動的に作成されてレポートに適用されます。

- ReportTitle
- CompanyName
- PageTitle
- ReportDate
- SubTitle
- DetailData (デフォルトスタイル)
- DetailLabel
- PageFooter
- RectangleStyle
- LineStyle

1011 ページの「フィールドの追加と書式設定」では、GroupFooter というフィールドを追加して GroupFooter バンド内のテキストフィールドとデータフィールドに適用する方法を説明しています。レポートのスタイルは CSS ファイルにエクスポートできます。スタイルに関する CSS コードは Report Builder により自動的に生成されます。複数のレポートで同じス

タイルセットを使用する場合は、この方法を使用すると効率的です。CSS ファイル内のスタイルは任意のテキストエディタを使用して変更することができます。Report Builder で CSS ファイルを直接インポートすることもできますが、実行時にレポート内のスタイルを置き換えることもできます。

レポートのスタイルを CSS ファイルにエクスポートするには

1 [ウィンドウ]-[レポートスタイル] を選択します。

2 エクスポートアイコンをクリックします (このアイコンには、オレンジ色の矢印が描かれています)。

3 [ファイル名] フィールドに、「**artstyles**」と入力します。このファイルには、自動的に CSS という拡張子が追加されます。

4 "artStyles.css" ファイルを参照してダブルクリックし、ファイルを開きます。生成される CSS コードの内容は次のようになります。

```
ReportTitle
{
        color:Black;
        font-size:24pt;
}
CompanyName
{
        color:#6188A5;
        font-weight:bold;
}
PageTitle
{
        color:#333333;
        font-size:14pt;
        font-weight:bold;
}
ReportDate
{
        color:#333333
}
SubTitle
{
        color:#6089A5;
        font-size:12pt;
        font-weight:bold;
}
DetailLabel
{
        color:Black;
        background-color:#E3EDEF;
        font-weight:bold;
}
DetailData
{
        default-style:true;
        color:Black;
        line-size:thin;
}
```

```
PageFooter
{
        color:#2F2F2F;
        font-size:8pt;
}
RectangleStyle
{
        color:#E3EDEF;
        background-color:#E3EDEF;
}
LineStyle
{
        color:#CCCCCC;
        background-color:#CCCCCC;
}
GroupFooter
{
        color:Blue;
        font-weight:bold;
        font-family:Tahoma;
```

5 次のコードに示すように、ReportTitle スタイルの color 属性を **Red** に変更し、font-weight 属性を追加します。

```
ReportTitle
{
        color:Red;
        font-size:24pt;
        font-weight: bold;
}
```

6 CSS ファイルを保存します。

また、ColdFusion からレポートスタイルを置き換えることもできます。詳細については、1019 ページの「レポートスタイルの置き換え」を参照してください。

注意：CSS ファイルにスタイルを追加した場合は、Report Builder で同じ名前のスタイルをレポートに追加してください。また、Report Builder は一部の CSS スタイルをサポートしません。詳細については、『CFML リファレンス』の cfreport タグを参照してください。

CSS ファイルをインポートするには

1 [ウィンドウ]-[レポートスタイル] を選択します。

2 スタイルインポートアイコンをクリックします (このアイコンには、青い矢印が描かれています)。

3 "artStyles.css" ファイルの場所を参照し、[OK] をクリックします。レポートスタイルの定義が自動的に更新され、更新されたスタイルがレポートタイトルに適用されます。

4 F12 キーを押してレポートをプレビューします。

レポート設定を実行時に置き換える方法

ColdFusion で cfreport タグを使用すると、Report Builder レポートのレポート設定を実行時に置き換えることができます。次の例では、1007 ページの「簡単なレポートの作成」で作成した CFR ファイルを使用します。

レポートクエリーの置き換え

この例では、アーティストのログイン ID に基づいてレポート内のデータをフィルタ処理します。アーティストがログオンすると、そのアーティストのデータと円グラフがレポートに表示されます。また、全アーティストのデータを示す円グラフもレポートに表示されます。

次のコードを使用すると、ColdFusion に簡単なログインページが作成されます。このフォームでは、ユーザー ID としてアーティストの姓を使用します (このコードには、パスワードの検証機能は含まれていません)。

```
<h3>Artist Login Form</h3>
<p>Please enter your last name and password.</p>
<cfform name="loginform" action="artSalesReport.cfm" method="post">
<table>
    <tr>
        <td>Last Name:</td>
        <td><cfinput type="text" name="username" required="yes" message="A username is
            required."></td>
    </tr>
    <tr>
        <td>Password:</td>
        <td><cfinput type="password" name="password" required="yes" message="A password is
            required."></td>
    </tr>
</table>
    <br />
    <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

Report Builder レポート内で作成したものと同様のクエリーを処理ページに追加します。ColdFusion クエリーには、少なくとも Report Builder クエリーに含まれるすべての列が含まれている必要がありますが、ColdFusion クエリーにはその他のデータを含めることもできます。

次の例に含まれるクエリーは、アーティストの姓に基づいて ART テーブルと ARTISTS テーブルからすべてのデータを選択します。cfreport タグでは、レポートテンプレートとして CFR ファイルのパス名を使用します。

```
<cfquery name="artsales" datasource="cfartgallery">
SELECT *
FROMAPP.ART, APP.ARTISTS
WHERE APP.ART.ARTISTID = APP.ARTISTS.ARTISTID
         AND APP.ARTISTS.LASTNAME= <cfqueryparam value="#FORM.username#">
ORDER BY ARTISTS.LASTNAME
</cfquery>

<cfreport query="#artsales#" template="ArtSalesReport1.cfr" format="RTF"/>
```

アーティストに関するレポートが RTF 形式で表示されます。CFR ファイルで定義されているデフォルトの出力形式より、format 属性の値が優先されます。

HTML でのレポートのエクスポート

レポートを HTML 形式で生成し、ブラウザで直接表示するには、属性を HTML に変更します。

```
<cfreport template="ArtSalesReport1.cfr" format="HTML"/>
```

一時ディレクトリが自動的に生成され、レポート内のすべてのイメージファイルがここに格納されます (チャートは PNG ファイルとして保存されます)。一時ディレクトリの場所は次のとおりです。

C:¥ColdFusion9¥tmpCache¥CFFileServlet¥_cfreport¥_report[< 一意の識別子 >]

次のように resourceTimespan 属性の値として CreateTimeSpan 関数を使用すると、一時ディレクトリをサーバーから削除する時期を指定できます。

```
<cfreport query="#artsales#" template="ArtSalesReport1.cfr" format="HTML"
resourceTimespan="#CreateTimeSpan(0,1,0,0)#"/>
```

この期間は、日数、時間、分、および秒で指定できます。この例では、1 時間後に一時ディレクトリが削除されます。詳細については、『CFML リファレンス』を参照してください。

レポート出力を HTML ファイルにエクスポートするには、filename 属性を指定します。次のコードは、artSales.html という HTML ファイルにレポート出力を書き込みます。

```
<cfreport template="ArtSalesReport1.cfr" format="HTML" filename="artSales.html" overwrite="yes"/>
```

イメージ用のディレクトリは、HTML 出力ファイルが書き込まれる場所の下に < ファイル名 >_files という形式で作成されます。この例では、レポート内のチャートの PNG ファイルが自動的に生成され、"artSales_files" というディレクトリに保存されます。また、cfartgallery データベースから抽出された JPG イメージのコピーも生成され、"artSales_files" ディレクトリに保存されます。詳細については、『CFML リファレンス』を参照してください。

レポートスタイルの置き換え

レポート内のレポートスタイルを置き換えるには、cfreport タグの style 属性を指定します。値には、有効な CSS シンタックス、CSS ファイルのパス名、または有効な CSS コードをポイントする変数を指定する必要があります。CSS 内のスタイル名は、Report Builder で定義されたレポートスタイル名と一致している必要があります。

次のコードは、ArtSalesReport1.cfr レポートのスタイルを "artStyles.css" ファイルで定義されたスタイルに置き換える方法を示しています。

```
<cfreport template="ArtSalesReport1.cfr" style="artStyles.css" format="PDF"/>
```

次のコードは、style 属性の値として CSS スタイルを適用する方法を示しています。

```
<cfreport template="ArtSalesReport1.cfr" style='ReportTitle {defaultStyle: false;
    font-family:"Tahoma"; color: "lime";}' format="FlashPaper">
</cfreport>
```

次のコードは、**myStyle** という変数を作成して、style 属性の値として使用する方法を示しています。

```
<cfset mystyle='DetailData { defaultStyle: true; font-family: "Tahoma"; color: ##00FFF0;}'>
<cfreport template="ArtSalesReport1.cfr" style="#mystyle#" format="HTML">
</cfreport>
```

詳細については、『CFML リファレンス』の cfreport タグを参照してください。

スライドプレゼンテーションの作成

Adobe ColdFusion では、スライドプレゼンテーションを作成できます。

ColdFusion プレゼンテーションについて

ColdFusion では、ソースファイルや、ColdFusion ページの CFML および HTML コードから、ダイナミックなスライドプレゼンテーションを作成することができます。データベースから抽出したデータを使用して、グラフやチャートなどのスライドの内容を設定できます。また、プレゼンテーションの各スライドに、イメージ、オーディオトラック、ビデオクリップを追加することもできます。ColdFusion には、スライドプレゼンテーションを作成するためのタグが 3 種類あります。

タグ	説明
cfpresentation	プレゼンテーションの外観を定義し、プレゼンテーションをファイルに保存するか、クライアントのブラウザで直接実行するかを決定します。
cfpresentationslide	次のいずれかをスライドのソースとして定義します。 • SWF ファイル • HTML ファイル • HTML コンテンツを返す URL • cfpresentationslide の開始タグと終了タグの間に記述した HTML および CFML コード
cfpresenter	スライドのプレゼンテーションを行う人に関する情報を提供します。1 人のプレゼンタを 1 枚または複数のスライドに割り当てることができます。プレゼンタ情報は、スライドの表示中、コントロールパネルに表示されます。

プレゼンテーションには最低 1 つのスライドを指定し、各プレゼンタを 1 枚または複数のスライドに割り当てることができます。次の例は、4 つの異なるソースを 2 人のプレゼンタが担当するスライドプレゼンテーションです。

```
<cfpresentation title="myPresentation">
    <cfpresenter name="Tuckerman" title="V.P. of Marketing"
            email="tuckerman@company.com">
    <cfpresenter name="Anne" title="V.P. of Sales" email="anne@company.com">
    <cfpresentationslide src="slide1.swf" title="Overview" duration="10"
            presenter="Anne"/>
    <cfpresentationslide src="slide2.htm" title="Q1 Sales" duration="30"
            presenter="Anne"/>
    <cfpresentationslide src="http://www.markettrends.com/index.htm"
            title="Market Trends" duration="30" presenter="Tuckerman"/>
    <cfpresentationslide title="Summary" duration="10">
        <h3>Summary</h3>
            <ul>
                <li>Projected Sales</li>
                <li>Challenges Ahead</li>
                <li>Long Term Goals</li>
            </ul>
    </cfpresentationslide>
</cfpresentation>
```

注意：cfpresentationslide タグには終了タグが必要です。スライドの内容としてソースファイルを指定する場合は、終了タグのショートカットとして終了スラッシュを使用します。

プレゼンテーションを実行すると、ColdFusion ページで記述されている順に、指定された継続時間 (duration) の間、各スライドが表示されます。プレゼンタ情報は、対応するスライドの横のコントロールパネルに表示されます。

スライドプレゼンテーションの作成

cfpresentation タグを使用して、スライドプレゼンテーションの外観をカスタマイズします。たとえば、コントロールを表示する場所や、プレゼンテーションインターフェイスの色などを指定できます。次に例を示します。

```
<cfpresentation title="Sales Presentation" controlLocation="left" primaryColor="##0000FF"
shadowColor="###000033" textColor="##FFFF00" showNotes="yes">
```

タイトルは、コントロールパネルの最上部に表示されます。カラー設定は、プレゼンテーションインターフェイスに反映されますが、プレゼンテーション内のスライドの形式には影響しません。showNotes 属性を yes に設定すると、各スライドに定義されているテキストメモが表示されます。

前の例のように、ディレクトリを指定しなかった場合は、クライアントブラウザでプレゼンテーションが直接実行されます。プレゼンテーションでは、サーバー上の一時ディレクトリに書き込まれたファイルが使用されます。プレゼンテーションを保存するには、絶対パスか、CFM ページを基準とした相対ディレクトリを指定します。ディレクトリは自動的に作成されないので、既存のディレクトリであることが必要です。次の例では、プレゼンテーションファイルをローカルドライブの "salesPresentation" ディレクトリに保存します。

```
<cfpresentation title="Sales Presentation" directory="c:\salesPresenation">
```

プレゼンテーションの実行に必要な次のファイルが自動的に生成されて、指定したディレクトリに保存されます。

- components.swf
- index.htm
- loadflash.js
- viewer.swf

また、"data" という名前のサブディレクトリが作成され、このディレクトリに次のファイルが保存されます。

- srchdata.xml (検索インターフェイスを作成するファイル)

- vconfig.xml
- viewer.xml
- プレゼンテーションの各スライドの SWF ファイル
- プレゼンテーションスライドで参照しているメディアファイルのコピー

  メディアファイルには、JPEG ファイル、FLV および SWF ビデオファイル、MP3 オーディオファイルが含まれます。ファイルに保存したプレゼンテーションを実行するには、"index.htm" ファイルをダブルクリックします。

注意：プレゼンテーションのスライドで参照しているファイルが上書きされることはありません。生成されたプレゼンテーションファイルに変更を加えても、ソースファイルに影響はありません。

プレゼンタの追加

オプションで、cfpresentation タグの下に 1 人または複数のプレゼンタを追加できます。プレゼンタ情報は、その情報が割り当てられている現在のスライドのコントロールパネルに表示されます。スライドにプレゼンタは必須ではありません。

cfpresenter タグを使用して、個人情報を指定します。たとえば、肩書、電子メールアドレス、ロゴ、顔写真などを提供できます。次にコードの例を示します。

```
<cfpresentation title="Sales Presentation">
<cfpresenter name="Anne" title="V.P. of Sales" biography="Anne Taylor has been a top seller at Widgets R Us
for five years." logo="images/logo.jpg" image="images/ataylor_empPhoto.jpg"
email="ataylor@widgetsrus.com">
```

name 属性は必須です。この値を使用して、スライドにプレゼンタを割り当てます。スライドにプレゼンタを割り当てるには、cfpresenter タグの name 属性の値を、cfpresentationslide タグの presenter 属性の値として使用します。次の例では、Tuckerman という名前のプレゼンタを作成して、Overview という名前のスライドに割り当てています。

```
<cfpresentation title="Sales Presentation">
<cfpresenter name="Tuckerman" title="V.P. of Marketing">
<cfpresentationslide title="Overview" src="overview.swf" presenter="Tuckerman" duration="10"/>
...
</cfpresentation>
```

注意：プレゼンタは、明示的にスライドに割り当ててください。1 人のプレゼンタを複数のスライドに割り当てるには、それぞれの cfpresentationslide タグでそのプレゼンタ名を使用します。

スライドにプレゼンタを割り当てると、スライドの継続時間の間、コントロールパネルにそのプレゼンタ情報が表示されます。イメージは、JPEG 形式で、ColdFusion ページを基準にした相対パスの場所に保存されている必要があります。email 属性の値はコントロールパネルの連絡先リンクにマッピングされます。このリンクをクリックすると、ローカルの電子メールアプリケーションの電子メールメッセージが開きます。

次のコードでは、プレゼンテーションに 3 人のプレゼンタを作成し、そのうちの 2 人をスライドに割り当てています。

```
<cfpresentation title="Sales Presentation">
    <cfpresenter name="Hannah" title="V.P. of Marketing" image="hannah.jpg">
    <cfpresenter name="Anne" title="V.P. of Sales" image="Anne.jpg">
    <cfpresenter name="Wilson" title="V.P. of Engineering"
            image="Wilson.jpg">
    <cfpresentationslide title="Overview" presenter="Hannah" duration="30"
            src="slide1.htm"/>
    <cfpresentationslide title="Q1 Sales" presenter="Anne" duration="15"
            src="slide2.htm"/>
    <cfpresentationslide title="Projected Sales" presenter="Anne"
            duration="15" src="slide3.htm" video="promo.flv"/>
    <cfpresentationslide title="Conclusion" src="slide4.htm"/>
</cfpresentation>
```

プレゼンタ Hannah を 1 枚のスライドに割り当て、Anne を 2 枚のスライドに割り当てています。プレゼンテーションの最後のスライドには、プレゼンタを割り当てていません。Wilson はスライドに割り当てていないので、プレゼンテーションで彼の情報は表示されません。2 枚目のスライドでは、Anne の写真がコントロールパネルに表示されます。ただし、3 枚目のスライドでは、Anne の写真の代わりに "promo.flv" というビデオが、スライドの継続時間の間コントロールパネルで再生されます。ビデオはスライドでは再生されません。

注意：ビデオは、SWF または FLV 形式であることが必要です。同じスライドにオーディオとビデオを指定することはできません。

スライドの追加

プレゼンテーションのスライド 1 枚につき、1 つの cfpresentationslide タグを使用します。プレゼンテーションでは、cfpresentation タグの下に記述されている順にスライドが実行されます。次のいずれかの方法で、スライドの内容を作成できます。

ソース	説明	例
SWF または HTML ファイル	このファイルは、ColdFusion が実行されているシステムに保存する必要があります。絶対パス、または ColdFusion ページを基準にした相対パスで指定できます。	<cfpresentationslide title="slide 1" src="presentation/slide1.swf"/> <cfpresentationslide title="slide 2" src="c:/presentation/slide2.htm"/>
URL	HTML コンテンツを返す URL であることが必要です。	<cfpresentationslide title="slide 3" src="http://www.worldview.com/index.htm"/>
ColdFusion ページの HTML および CFML コード	HTML および CFML コードは、cfpresentationslide の開始タグと終了タグの間に記述します。	<cfpresentationslide> Total Sales <cfchart format="jpg" chartwidth="500" show3d="yes"> <cfchartseries type="pie" query="artwork" itemcolumn="issold" valuecolumn="price"/> </cfchart> </cfpresentationslide>

ソースファイルによるスライドの作成

次のコードでは、別々の場所にあるソースファイルを使用して、3 枚のスライドを持つプレゼンテーションを作成します。

```
<cfpresentation title="Garden Mania" directory="gardenPresentation">
    <cfpresentationslide title="Seeds of Change" src="c:\gardening\seeds.html" audio="media\hendrix.mp3"
duration="30"/>
    <cfpresentationslide title="Flower Power" src="shockwave\flowerPower.swf" duration="40"/>
    <cfpresentationslide title="Dig Deep" src="http://www.smartgarden.com/index.htm" duration="15"/>
</cfpresentation>
```

この例の場合、プレゼンテーションの実行に必要なファイルが "gardenPresentation" ディレクトリに生成されます。各スライドの新しい SWF ファイルは "data" サブディレクトリに生成されます。また、"hendrix.mp3" ファイルがコピーされて、"data" サブディレクトリに保存されます。

注意：HTML ファイルから作成したスライド内のリンクは、アクティブになりません。

HTML および CFML コードによるスライドの作成

スライドのソースファイルを指定しない場合は、cfpresentationslide タグ本文に HTML または CFML コードを記述して、スライドの内容を作成します。次のプレゼンテーションには、タイプごとにスライドが次のように含まれます。

- HTML で作成したスライドが 1 枚
- HTML および CFML で作成したスライドが 1 枚

- 外部 Web サイトの HTML ファイルから抽出したスライドが 1 枚

```
<cfpresentation title="The Road Ahead">
<cfpresentationslide title="Yellow Bricks" audio="myaudio1.mp3" duration="10">
    <h3>Yellow Bricks</h3>
    <table cellpadding=10>
        <tr>
            <td>
                <ul>
                    <li>Way to go Dorothy</li>
                    <li>Making tracks</li>
                    <li>No place like home</li>
                </ul>
            </td>
            <td><img src="../cfdocs/images/artgallery/maxwell01.jpg"/>
            </td>
        </tr>
    </table>
    </cfpresentationslide>
<cfpresentationslide title="Wild Ride" duration="5">
    <h3>Wild Ride</h3>
    <cfchart format="jpg" title="Who's Ahead" show3D="yes" chartHeight=500 chartWidth=500>
        <cfchartseries type="pyramid">
            <cfchartdata item="Dorothy" value=10>
            <cfchartdata item="Tin Man" value=30>
            <cfchartdata item="Scarecrow" value=15>
            <cfchartdata item="Lion" value=50>
            <cfchartdata item="Toto" value=5>
        </cfchartseries>
    </cfchart>
</cfpresentationslide>
<cfpresentationslide title="The Golden Age of Ballooning" duration="10"
src="http://www.balloning.com/index.htm"/>
</cfpresentation>
```

注意：cfchart タグの format 属性の値は、JPG か PNG であることが必要です。

スライドに表示できるのは、スタティックなデータに限定されません。データベースやクエリーオブクエリーから情報を抽出してスライドに挿入することもできます。

サンプルプレゼンテーション

このセクションでは、2 つのサンプルプレゼンテーションを紹介します。

例 1

次の例では、cfdocexamples データベースから取得したデータを組み込んだ簡単なプレゼンテーションを作成します。ここでは次のタスクを行います。

- HTML および CFML からスライドを作成する。
- スライドにイメージを追加する。
- データベースから抽出したデータを使用してチャートとテーブルを追加する。
- 個別のスライドにオーディオトラックを追加する。

```
<!--- The following query extracts employee data from the cfdocexamples
            database. --->
<cfquery name="GetSalaryDetails" datasource="cfdocexamples">
    SELECT Departmt.Dept_Name,
        Employee.FirstName,
        Employee.LastName,
        Employee.StartDate,
        Employee.Salary,
        Employee.Contract
    From Departmt, Employee
    Where Departmt.Dept_ID = Employee.Dept_ID
    ORDER BY Employee.LastName, Employee.Firstname
</cfquery>

<!--- The following code creates a presentation with three presenters. --->
<cfpresentation title="Employee Satisfaction" primaryColor="##0000FF" glowColor="##FF00FF"
lightColor="##FFFF00" showoutline="no">
    <cfpresenter name="Jeff" title="CFO" email="jeff@company.com"
            logo="../cfdocs/getting_started/photos/somewhere.jpg"
            image="../cfdocs/images/artgallery/jeff01.jpg">
    <cfpresenter name="Lori" title="VP Marketing" email="lori@company.com"
            logo="../cfdocs/getting_started/photos/somewhere.jpg"
            image="../cfdocs/images/artgallery/lori01.jpg">
    <cfpresenter name="Paul" title="VP Sales" email="paul@company.com"
            logo="../cfdocs/getting_started/photos/somewhere.jpg"
            image="../cfdocs/images/artgallery/paul01.jpg">

<!--- The following code creates the first slide in the presentation
            from HTML. --->
    <cfpresentationslide title="Introduction" presenter="Jeff"
            audio="myAudio1.mp3" duration="5">
    <h3>Introduction</h3>
    <table>
        <tr><td>
            <ul>
                <li>Company Overview</li>
                <li>Salary by Department</li>
                <li>Employee Salary Details</li>
            </ul>
        </td></tr>
    </table>
    </cfpresentationslide>

<!--- The following code creates the second slide in the presentation.
            The chart is populated with data from the database query. --->
    <cfpresentationslide title="Salary by Department" presenter="Lori"
            duration="5" audio="myAudio3.mp3">
        <h3>Salary by Department</h3>
        <cfchart format="jpg" xaxistitle="Department" yaxistitle="Salary">
            <cfchartseries type="bar" query="GetSalaryDetails"
                    itemcolumn="Dept_Name" valuecolumn="salary">
            </cfchartseries>
        </cfchart>
    </cfpresentationslide>

<!--- The following code creates the third slide in the presentation. The table is populated with data from
the query. The table also contains an image located relative to the CFM page on the server. --->
    <cfpresentationslide title="Salary Details" presenter="Paul"
            duration="10" audio="myAudio1.mp3">
        <h3>Employee Salary Details</h3>
        <table border cellspacing=0 cellpadding=5 valign="top">
        <tr>
```

```
        <td>
        <table border cellspacing=0 cellpadding=5 valign="top">
        <tr>
            <th>Employee Name</th>
            <th>Start Date</th>
            <th>Salary</th>
            <th>Department</th>
            <th>Contract?</th>
        </tr>
        <cfoutput query="GetSalaryDetails">
        <tr>
            <td>#FirstName# #LastName#</td>
            <td>#dateFormat(StartDate, "mm/dd/yyyy")#</td>
            <td>#numberFormat(Salary, "$9999,9999")#</td>
            <td>#dept_name#</td>
            <td>#Contract#</td>
            </tr></cfoutput>
            </table>
        </td>
        <td width="200" >
            <img src="images/raquel02.jpg"/>
        </td>
    </table>
    </cfpresentationslide>
</cfpresentation>
```

例 2

次の例では、cfartgallery データベースのデータを使用した簡単な販売プレゼンテーションを作成します。具体的には、次のタスクを行います。

- HTML および CFML からスライドを作成する。
- HTML コンテンツを返す URL からスライドを作成する。
- データベースとクエリーオブクエリーから抽出したデータを使用してチャートを追加する。
- 個別のスライドにビデオとオーディオトラックを追加する。

```
<!--- The following query extracts data from the cfartgallery database. --->
<cfquery name="artwork" datasource="cfartgallery">
SELECT FIRSTNAME || ' '|| LASTNAME AS FULLNAME, ARTISTS.ARTISTID, ARTNAME, PRICE, ISSOLD
FROM ARTISTS, ART
WHERE ARTISTS.ARTISTID = ART.ARTISTID
ORDER BY LASTNAME
</cfquery>

<!--- The following query of queries determines the total dollar amount of
            sales per artist. --->
<cfquery dbtype="query" name="artistname">
SELECT FULLNAME,
SUM(PRICE) AS totalSale
FROM ARTWORK
WHERE ISSOLD = 1
GROUP BY FULLNAME
ORDER BY totalSale
</cfquery>

<!--- The following code determines the look of the slide presentation. ColdFusion displays the slide
presentation directly in the browser because no destination is specified. The title appears above the
presenter information. --->
<cfpresentation title="Art Sales Presentation" primaryColor="##0000FF" glowColor="##FF00FF"
lightColor="##FFFF00" showOutline="yes" showNotes="yes">
```

```
<!--- The following code defines the presenter information. You can assign each presenter to one or more
slides. --->
    <cfpresenter name="Aiden" title="Artist" email="Aiden@artgallery.com"
image="../cfdocs/images/artgallery/aiden01.jpg">
    <cfpresenter name="Raquel" title="Artist" email="raquel@artgallery.com"
image="../cfdocs/images/artgallery/raquel05.jpg">
    <cfpresenter name="Paul" title="Artist" email="paul@artgallery.com"
image="../cfdocs/images/artgallery/paul01.jpg">

<!--- The following code defines the content for the first slide in the presentation. The duration of
the slide determines how long the slide plays before proceeding to the next slide. The audio plays for
the duration of the slide. --->
    <cfpresentationslide title="Introduction" presenter="Aiden" duration="5" audio="myAudio1.mp3">
    <h3>Introduction</h3>
    <table>
        <tr><td>
            <ul>
                <li>Art Sales Overview</li>
                <li>Total Sales</li>
                <li>Total Sales by Artist</li>
                <li>Conclusion</li>
            </ul>
        </td>
        <td><img src="../cfdocs/images/artgallery/maxwell01.jpg"/></td></tr>
    </table>
    </cfpresentationslide>

<!--- The following code generates the second slide in the presentation from an HTML file located on an
external website. --->
    <cfpresentationslide title="Artwork Sales Overview" presenter="Raquel" audio="myAudio2.mp3"
duration="5" src="http://www.louvre.com/index.html"/>

<!--- The following code generates the third slide in the presentation, which contains a pie chart with
data extracted from the initial database query. ColdFusion runs the video defined in the
cfpresentationslide tag in place of the presenter image defined in the cfpresenter tag. --->
    <cfpresentationslide title="Total Artwork Sold" presenter="Aiden"
            duration="5" video="video1.flv">
        <h3>Total Sales</h3>
        <cfchart format="jpg" chartwidth="500" show3d="yes">
            <cfchartseries type="pie" query="artwork"
                    colorlist="##00FFFF,##FF00FF" itemcolumn="issold"
                    valuecolumn="price"/>
            </cfchart>
        </cfpresentationslide>

<!--- The following code generates the fourth slide in the presentation with
                data extracted from the query of queries. --->
    <cfpresentationslide title="Sales by Artist" presenter="Paul"
            duration="5" audio="myAudio3.mp3">
        <h3>Total Sales by Artist</h3>
        <table border cellspacing=10 cellpadding=0>
        <TR>
        <TD>
            <table border cellspacing=0 cellpadding=5>
                <tr>
                    <th>Artist Name</th>
                    <th>Total Sales</th>
                </tr>
                <tr>
                <cfoutput query="artistname">
                <td>#FULLNAME#</td>
```

```
                <td>#dollarFormat(totalSale)#</td>
                </tr>
                </cfoutput>
            </table>
            </td>
            <td>
                <cfchart format="jpg" xaxistitle="Artist" yaxistitle="Total Sales"
                        chartwidth="400">
                    <cfchartseries type="bar" query="artistname"
                            itemcolumn="fullname" valuecolumn="totalSale"/>
                </cfchart>
            </td>
            </tr>
        </table>
    </cfpresentationslide>

<!--- The following code defines the final slide in the presentation. This slide does not have a presenter
assigned to it. --->
    <cfpresentationslide title="Conclusion" duration="1" notes="Special thanks to Lori and Jeff for
contributing their art and expertise.">
    <h1>Great Job Team!</h1>
    <p><img src="../cfdocs/images/artgallery/paul05.jpg"></p>
    </cfpresentationslide>
</cfpresentation>
```

第 15 章：Web 要素および外部オブジェクトの使用

XML および WDDX の使用

Adobe ColdFusion では XML ドキュメントを作成、使用、および操作できます。また、アプリケーション間や、CFML と JavaScript の間でデータをやり取りするなど、XML ベースの WDDX (Web Distributed Data Exchange) を構造化データの送信に使用することもできます。

XML および ColdFusion について

XML は、ドキュメントやデータを Web 上でやり取りするための標準言語として、急速に普及しています。XML ドキュメントは、従来の印刷物やそれに似たドキュメントの枠を超える柔軟性を持っています。たとえば、XML を使用してデータベース情報やディレクトリ情報を表すことができます。また、製品の注文や受領などの取引情報や、在庫記録、従業員データなどの情報も扱うことができます。

XML では、タグを使用したテキスト形式でデータが表現されるので、通常は互換性がないアプリケーション (たとえば受注管理システムと在庫管理システム) の間でも情報を共有できます。どのアプリケーションも、相手のアプリケーションのことを理解する必要はありません。XML の DTD またはスキーマに基づいて構築されたデータさえ理解できれば、それを介して他のアプリケーションと通信できます。たとえば、分散型の注文処理アプリケーションでは、受注コンポーネント、注文調達コンポーネント、在庫管理コンポーネント、および請求書発行コンポーネントの間で、XML 形式を使用して情報を共有できます。これらのコンポーネントは共通の XML DTD を使用することもできますが、コンポーネントごとの異なる DTD を使用しても互いに通信できます。

各アプリケーションでは、XML ドキュメントを解析して得られた情報を、自由に処理できます。たとえば、XML のテーブルデータを ColdFusion レコードセットに変換してクエリーを実行し、得られたデータを XML ドキュメントにエクスポートできます。1049 ページの「例 : ColdFusion アプリケーションでの XML の使用」のコードでは、顧客の注文を XML 形式で受け取ってレコードセットに変換し、クエリーを使用して注文金額を決定し、XML ドキュメントの領収書を作成しています。

ColdFusion には、XML ドキュメントを簡単に作成および使用できる便利なツールが用意されています。ColdFusion では、XML ドキュメントに対して次の操作が行えます。

- XML テキストを ColdFusion XML ドキュメントオブジェクトに変換する。
- ColdFusion XML ドキュメントオブジェクトを新規作成する。
- ColdFusion XML ドキュメントオブジェクトを変更する。
- XML を DTD またはスキーマと照合して検証する。
- XSLT (Extensible Stylesheet Language Transformation) を使用して XML を変換する。
- XPath 式を使用して XML ドキュメントからデータを抽出する。
- ColdFusion XML ドキュメントオブジェクトをテキストに変換し、ファイルに保存する。

ColdFusion では、cfform タグで作成したフォームを XML で表現することができます。生成した XML を XSLT スキンで処理して表示用の出力を生成したり、生成した XML テキストを変数に格納して操作することができます。XML フォームの詳細については、768 ページの「スキン可能 XML フォームの作成」を参照してください。

XML ドキュメントオブジェクト

ColdFusion では、**XML** ドキュメントオブジェクトと呼ばれるオブジェクトで XML ドキュメントが表現されます。これは、標準の ColdFusion 構造体によく似たオブジェクトです。実際、ColdFusion のほとんどの構造体関数 (StructInsert など) は、XML ドキュメントオブジェクトでも使用できます。XML ドキュメントオブジェクトで使用できるすべての ColdFusion 関数のリストについては、1039 ページの「XML オブジェクトの管理に使用できる関数」を参照してください。

XML ドキュメントの全体的な構造は、2 通りの方法で表現できます。1 つは基本ビューで、もう 1 つは DOM (Document Object Model) ベースのノードビューです。基本ビューには、ドキュメント内の情報がすべて含まれていますが、ノードビューのほうがデータがより詳細に分割されています。ColdFusion では、いずれのビューを使用しても XML ドキュメントの内容にアクセスできます。

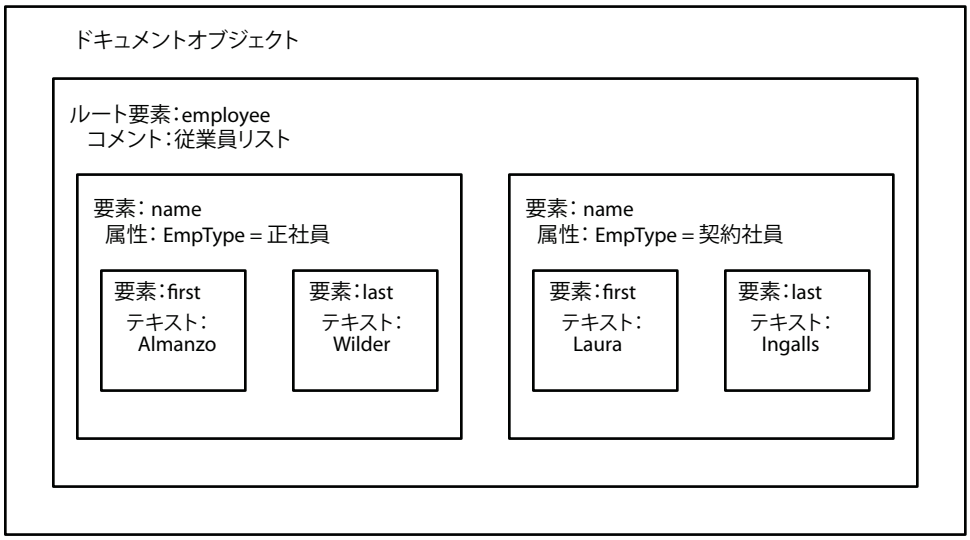
単純な XML ドキュメント

ここでは、次の簡単な XML ドキュメントを例に取って、基本ビューとノードビューについて説明します。このドキュメントは、ColdFusion の XML マニュアルに含まれる多くの例で使用されています。

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
<!-- A list of employees -->
    <name EmpType="Regular">
        <first>Almanzo</first>
        <last>Wilder</last>
    </name>
    <name EmpType="Contract">
        <first>Laura</first>
        <last>Ingalls</last>
    </name>
</employee>
```

基本ビュー

基本ビューでは、ルート要素構造体が 1 つ格納されているコンテナとして、XML ドキュメントオブジェクトが表現されます。ルート要素は、ネストされた要素構造体をいくつでも持つことができます。それぞれの要素構造体は、特定の XML タグ (開始タグと終了タグのセット) と、そのすべての内容を表します。要素構造体の中に別の要素構造体が含まれていることもあります。前に示した単純な XML ドキュメントの基本ビューは、次のようになります。



DOM ノードビュー

DOM ノードビューでは、ドキュメントの XML **DOM (Document Object Model)** と同じ形式を使用して、XML ドキュメントオブジェクトが表現されます。実際、XML ドキュメントオブジェクトは DOM オブジェクトを表現したものです。

DOM は、W3C (World Wide Web Consortium) 勧告 (仕様) の 1 つで、ドキュメントの内容、構造、およびスタイルをダイナミックにアクセスおよび更新可能な、プラットフォームや言語に依存しないインターフェイスです。ColdFusion は、DOM Level 2 Core Specification に準拠しています。この仕様の詳細については、www.w3.org/TR/DOM-Level-2-Core を参照してください。

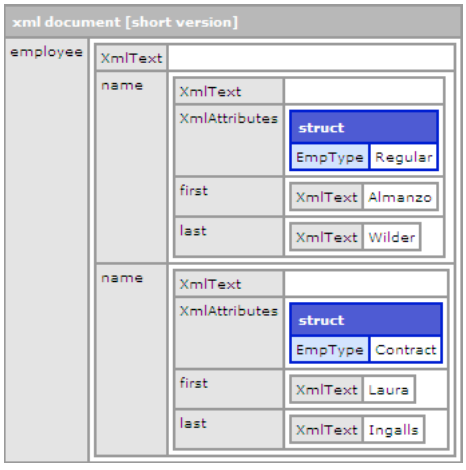
DOM ノードビューでは、ノードの階層ツリーによってドキュメントが表現されます。各ノードは、DOM ノードタイプ、ノード名、およびノード値を持っています。ノードタイプには、要素、コメント、テキストなどがあります。DOM では、ドキュメントオブジェクトとその中の各要素が、さまざまなタイプの複数のノードに構造化され、基本ビューよりも詳細なドキュメント構造が形成されます。たとえば、テキストブロックの中に XML コメントがある場合、DOM ノードビューではテキスト内の XML コメントの位置がわかりますが、基本ビューではわかりません。

ColdFusion では、W3C DOM Level 2 Core 仕様で定義されている DOM オブジェクト、メソッド、およびプロパティを使用して、XML ドキュメントオブジェクトを操作することもできます。

DOM ノードを参照する方法の詳細については、1031 ページの「XML DOM ノード構造体」を参照してください。このドキュメントでは、ノードビューや DOM メソッドおよびプロパティの使用方法については詳しく説明していません。

XML ドキュメントの構造

XML ドキュメントオブジェクトは、ネストした XML 要素構造体がいくつか含まれている 1 つの構造体です。1029 ページの「単純な XML ドキュメント」の XML に対応するドキュメントオブジェクトを cfdump タグで出力すると、次の図のようになります (一部のみ抜粋)。このダンプは、ドキュメントオブジェクトのすべての詳細が含まれる長いバージョンです。初期状態では、基本情報のみが含まれる短いバージョンが表示されます。ダンプヘッダをクリックすると、短いバージョン、長いバージョン、閉じたバージョンを切り替えることができます。



この出力を表示するには、次のコードを使用します。ここでは、前述のコードが Web ルートの下の "C:¥Inetpub¥wwwroot¥testdocs¥employeesimple.xml" に保存されていると仮定しています。

```
<cffile action="read" file="C:\Inetpub\wwwroot\testdocs\employeesimple.xml"
    variable="xmldoc">
<cfset mydoc = XmlParse(xmldoc)>
<cfdump var="#mydoc#">
```

ドキュメントオブジェクトの構造

XML ドキュメントオブジェクトのトップレベルには、次の 3 つのエントリがあります。

エントリ名	型	説明
XmlRoot	要素	ドキュメントのルート要素。
XmlComment	文字列	ドキュメントのすべてのコメント (ドキュメントのプロローグおよびエピローグ内のコメント) を連結した文字列。この文字列には、ドキュメント要素の内部のコメントは含まれません。
XmlDocType	XmlNode	ドキュメントの DocType 属性。このエントリは、ドキュメントで DocType が指定されている場合にのみ存在します。この値は読み取り専用であり、ドキュメントオブジェクトの作成後に設定することはできません。 このエントリは、cfdump タグで XML 要素構造体をダンプしても表示されません。

要素構造体

各 XML 要素には次のエントリがあります。

エントリ名	型	説明
XmlName	文字列	要素の名前。名前空間の接頭辞も含まれます。
XmlNsPrefix	文字列	名前空間の接頭辞。
XmlNsURI	文字列	名前空間の URI。
XmlText または XmlCdata	文字列	要素内のすべてのテキストと CData テキストを連結した文字列。ただし、子要素内のテキストは含まれません。XmlCdata 要素に値を割り当てると、そのテキストは CDATA の中に格納されます。ドキュメントオブジェクトから情報を取得する場合は、どちらの要素名を使用しても同じ値が返されます。
XmlComment	文字列	XML 要素内のすべてのコメントを連結した文字列。ただし、子要素内のテキストは含まれません。
XmlAttributes	構造体	この要素のすべての属性を名前と値のペアとして含む構造体。
XmlChildren	配列	この要素のすべての子要素。
XmlParent	XmlNode	この要素の親 DOM ノード。 このエントリは、cfdump タグで XML 要素構造体をダンプしても表示されません。
XmlNodes	配列	この要素に含まれているすべての XmlNode DOM ノードの配列。 このエントリは、cfdump タグで XML 要素構造体をダンプしても表示されません。

XML DOM ノード構造体

次の表に、XML DOM ノード構造体の内容を示します。

エントリ名	型	説明
XmlName	文字列	ノード名。要素や属性などのノードでは、その要素または属性の名前になります。
XmlType	文字列	要素やテキストなどの、ノードの XML DOM タイプ。
XmlValue	文字列	ノードの値。このエントリは、属性、CDATA、コメント、およびテキストタイプのノードでのみ使用します。

注意：XmlNode 構造体は cfdump タグでは表示されません。XmlNode 構造体を cfdump タグでダンプしようとすると、"Empty Structure" と表示されます。

次の表に、XmlType エントリで有効な各ノードタイプの、XmlName および XmlValue フィールドの内容を示します。ノードタイプは、XML DOM 階層のオブジェクトタイプに対応します。

ノードタイプ	XmlName	xmlValue
CDATA	#cdata-section	CDATA セクションの内容
COMMENT	#comment	コメントの内容
ELEMENT	タグ名	空の文字列
ENTITYREF	参照されるエンティティの名前	空の文字列
PI (処理命令)	処理命令 (PI) の名前	空の文字列
TEXT	#text	テキストノードの内容
ENTITY	エンティティ名	空の文字列
NOTATION	notation 名	空の文字列
DOCUMENT	#document	空の文字列
FRAGMENT	#document-fragment	空の文字列
DOCTYPE	ドキュメントタイプ名	空の文字列

注意：XML 属性は DOM ツリーのノードですが、XML DOM ノードのデータ構造体としてはアクセスできません。要素の属性を表示するには、要素構造体の XMLAttributes 構造体を使用します。

XML ドキュメントオブジェクトとそのすべての要素は、DOM ノード構造体としてアクセスできます。たとえば、1029 ページの「単純な XML ドキュメント」の XML から作成した DOM ツリーのノードを参照する場合は、次の変数名を使用できます。

```
mydoc.XmlName
mydoc.XmlValue
mydoc.XmlRoot.XmlName
mydoc.employee.XmlType
mydoc.employee.XmlNodes[1].XmlType
```

ColdFusion の XML タグおよび関数

次の表に、XML ドキュメントの作成や操作に使用できる ColdFusion のタグと関数を示します。

タグまたは関数	説明
<cfxml variable="objectName" [caseSensitive="Boolean"]>	タグ本文に記述されているマークアップに基づいて ColdFusion XML ドキュメントオブジェクトを作成します。このタグには、XML タグおよび CFML タグを含めることができます。テキストが XML ドキュメントオブジェクトに変換される前に、タグ本文にあるすべての CFML が処理されます。 CaseSensitive="True" 属性を指定すると、ドキュメント内の要素名や属性名の大文字と小文字が区別されます。デフォルト値は False です。 cfxml タグの使用方法の詳細については、1037 ページの「cfxml タグによる XML ドキュメントの作成」を参照してください。
XmlParse (XMLText [[, caseSensitive], validator])	ファイルまたは文字列変数に含まれる XML ドキュメントを、XML ドキュメントオブジェクトに変換します。オプションで、DTD またはスキーマと照合してドキュメントを検証できます。 オプションの 2 番目の引数を True に設定すると、ドキュメント内の要素名や属性名の大文字と小文字が区別されます。デフォルト値は False です。 XmlParse 関数の使用方法の詳細については、1038 ページの「既存の XML による新しい XML ドキュメントの作成」を参照してください。
XmlNew([caseSensitive])	空の XML ドキュメントオブジェクトを返します。 オプションの引数を True に設定すると、ドキュメント内の要素名や属性名の大文字と小文字が区別されます。デフォルト値は False です。 XmlNew 関数の使用方法の詳細については、1038 ページの「XmlNew 関数による XML ドキュメントの作成」を参照してください。
XmlElemNew(objectName[, namespaceURI], elementName)	指定した名前の XML ドキュメントオブジェクト要素を返します。オプションで、指定の名前空間に所属させることもできます。同じオブジェクト内の別の場所で名前空間の接頭辞を既に定義している場合は、namespaceURI パラメータを省略して、接頭辞のみを記述できます。 XmlElemNew 関数の使用方法の詳細については、1043 ページの「要素の追加」を参照してください。
XmlTransform(XMLVar, XSLTStringVar[, parameters])	XSLT (Extensible Stylesheet Language Transformation) を XML ドキュメントに適用します。この XML ドキュメントは、文字列変数でも XML ドキュメントオブジェクトでもかまいません。この関数で生成された XML ドキュメントは、文字列として返されます。 XmlTransform 関数の使用方法の詳細については、1048 ページの「XSLT によるドキュメントの変換」を参照してください。
XmlSearch(objectName, XPathExpression)	XPath 式を使用して XML ドキュメントオブジェクトを検索し、検索基準に一致する XML 要素の配列を返します。 XmlSearch 関数の使用方法の詳細については、1049 ページの「XPath によるデータの抽出」を参照してください。
XmlValidate(xmlDoc[, validator])	DTD (ドキュメントタイプ定義) または XML スキーマを使用して、XML テキストドキュメント (文字列またはファイル内)、または XML ドキュメントオブジェクトを検証します。validator には DTD またはスキーマを使用できます。validator パラメータを省略する場合は、ドキュメント内に DTD またはスキーマが指定されていることが必要です。XmlValidate 関数の使用方法の詳細については、1048 ページの「XML ドキュメントの検証」を参照してください。
XmlChildPos(element, elementName, position)	elementName で指定した要素名を持つ N 番目の子の、XmlChildren 配列内での位置 (インデックス) を返します。たとえば、XmlChildPos(mydoc.employee, "name", 2) では、mydoc.employee.XmlChildren 内での mydoc.employee.name[2] 要素の位置を返します。このインデックスは、ArrayInsertAt 関数と ArrayDeleteAt 関数で使用できます。 XmlChildPos 関数の使用方法の詳細については、1043 ページの「同名子要素の位置の取得」、1043 ページの「要素の追加」、および 1044 ページの「要素の削除」を参照してください。
XmlGetNodeType(xmlNode)	関数から返される XML ドキュメントオブジェクトノードや、要素の XmlNodes 配列に含まれている XML ドキュメントオブジェクトノードの、タイプを識別する文字列を返します。
IsWDDX(String)	文字列が整形式の WDDX パケットかどうかを調べます。
IsXML(String)	文字列が整形式の XML テキストかどうかを調べます。
IsXmlAttribute(variable)	関数パラメータが XML DOM (Document Object Model) 属性ノードかどうかを調べます。

タグまたは関数	説明
IsXmlDoc(objectName)	関数の引数が XML ドキュメントオブジェクトの場合は、True を返します。
IsXmlElem(elementName)	関数の引数が XML ドキュメントオブジェクトの要素の場合は、True を返します。
IsXmlNode(variable)	関数パラメータが XML ドキュメントオブジェクトのノードかどうかを調べます。
IsXmlRoot(elementName)	関数の引数が XML ドキュメントオブジェクトのルート要素の場合は、True を返します。
ToString(objectName)	XML ドキュメントオブジェクトを文字列に変換します。
XmlFormat(string)	文字列を XML 内でテキストとして使用できるように、文字列に含まれる XML の特殊文字をエスケープします。

大文字小文字の区別と XML ドキュメントオブジェクトについて

XML ドキュメントオブジェクトを作成するタグおよび関数では、オブジェクトの大文字と小文字を区別するかどうかを指定できます。大文字と小文字を区別しない場合、XML ドキュメントオブジェクトのコンポーネント識別子 (要素名や属性名など) での大文字と小文字の違いは無視されます。大文字と小文字を区別するように指定すると、大文字と小文字が異なる名前は異なるコンポーネントと見なされます。たとえば、大文字と小文字を区別しない場合、mydoc.employee.name[1] と mydoc.employee.NAME[1] は常に同じ要素と見なされます。大文字と小文字を区別する場合、これらの名前は 2 つの異なる要素と見なされます。大文字と小文字が区別される XML ドキュメントの要素名や属性名を、ドット表記法で参照することはできません。詳細については、1034 ページの「XML オブジェクトの内容の参照」を参照してください。

XML オブジェクトの使用

XML ドキュメントオブジェクトは構造体として表現されるので、XML ドキュメントの内容にアクセスするには、次のいずれかまたは両方の方法を使用できます。

- 要素名を使用する方法 (例 : mydoc.employee.name[1])
- 対応する構造体エントリ名 (XmlChildren 配列エントリ) を使用する方法 (例 : mydoc.employee.XmlChildren[1])

同様に、次のいずれかまたは両方の表記法を使用できます。

- 構造体 (ドット) 表記法 (例 : mydoc.employee)
- 連想配列 (括弧) 表記法 (例 : mydoc["employee"])

XML オブジェクトの内容の参照

代入式の右辺や関数の引数で XML ドキュメントオブジェクトの内容を参照するときは、次のルールに従ってください。

- デフォルトでは、ColdFusion は要素名の大文字と小文字を区別しません。したがって、要素名 MyElement と myELement は同じであると見なされます。要素名の大文字と小文字が区別されるようにするには、ドキュメントオブジェクトの作成時に、cfxml タグで CaseSensitive="True" を指定するか、XmlParse または XmlNew 関数の第 2 引数に True を指定します。
- 大文字と小文字が区別される XML オブジェクトの要素名や属性名は、ドット表記法では参照できません。連想配列 (括弧) 表記法の名前を使用するか、大文字と小文字が区別される名前を使用せずに参照する必要があります。たとえば、次のような名前は使用できません。

  ```
  MyDoc.employee.name[1]

  MyDoc.employee.XmlAttributes.Version
  ```

  代わりに、次のような名前を使用してください。

```
MyDoc.xmlRoot.XmlChildren[1]
MyDoc.xmlRoot["name"][1]
MyDoc.["employee"]["name"][1]

MyDoc.xmlRoot.XmlAttributes["Version"]
MyDoc["employee"].XmlAttributes["Version"]
```

重要：ColdFusion では変数名の大文字と小文字が区別されないので、大文字と小文字が区別される **XML** ドキュメントの要素名や属性名をドット表記法で参照すると、予期しない結果が生じる **(** たとえば、変数名がすべて大文字になる **)** 場合や、例外が発生する場合があります。

- 大文字と小文字が区別される XML オブジェクトの要素名や属性名は、ドット表記法では参照できません。連想配列 (括弧) 表記法の名前を使用するか、大文字と小文字が区別される名前を使用せずに参照する (XmlChildren[1] などのようにする) 必要があります。
- 同じ名前の要素の 1 つを指定するには、配列インデックスを使用します。たとえば、#mydoc.employee.name[1] や #mydoc.employee.name[2] のように指定します。

  要素識別子の最後のコンポーネントで配列インデックスを省略すると、指定の名前を持つすべての要素の配列が返されます。たとえば、mydoc.employee.name は 2 つの name 要素からなる配列を参照します。
- 名前を使用せずに要素を指定するには、XmlChildren 配列に配列インデックスを指定します。たとえば、mydoc.XmlRoot.XmlChildren[1] のように指定します。
- 名前にピリオドやコロンが含まれている要素を指定するには、連想配列 (括弧) 表記法を使用します。たとえば、myotherdoc.XmlRoot["Type1.Case1"] のように指定します。
- 構造体エントリ名の代わりに DOM メソッドを使用できます。

たとえば、次の変数はすべて、1029 ページの「単純な XML ドキュメント」で作成した XML ドキュメントの XmlText の値 "Almanzo" を参照します。

```
mydoc.XmlRoot.XmlChildren[1].XmlChildren[1].XmlText
mydoc.employee.name[1].first.XmlText
mydoc.employee.name[1]["first"].XmlText
mydoc["employee"].name[1]["first"].XmlText
mydoc.XmlRoot.name[1].XmlChildren[1]["XmlText"]
```

次の変数はすべて、1029 ページの「単純な XML ドキュメント」で作成した XML ドキュメントの最初の name 要素の EmpType 属性を参照します。

```
mydoc.employee.name[1].XmlAttributes.EmpType
mydoc.employee.name[1].XmlAttributes["EmpType"]
mydoc.employee.XmlChildren[1].XmlAttributes.EmpType
mydoc.XmlRoot.name[1].XmlAttributes["EmpType"]
mydoc.XmlRoot.XmlChildren[1].XmlAttributes.EmpType
```

これらのリストは、値または属性を参照する可能な組み合わせをすべて列挙したものではありません。

XML オブジェクトへのデータの割り当て

式の左辺で XML オブジェクト参照を使用する場合は、参照文字列の最後の要素までの参照に関して、前述のルールのほとんどが適用されます。

たとえば次の式では、mydoc.employee.name[1].first に 1034 ページの「XML オブジェクトの内容の参照」のルールが適用されます。

```
mydoc.employee.name[1].first.MyNewElement = XmlElemNew(mydoc, NewElement);
```

ただし、次の注意で説明するように、大文字と小文字が区別されるドキュメントオブジェクトでのネーミングルールは、参照文字列全体に適用されます。

重要：ColdFusion では変数名の大文字と小文字が区別されないので、大文字と小文字が区別される **XML** ドキュメントの要素名や属性名をドット表記法で参照すると、予期しない結果が生じる **(** たとえば、変数名がすべて大文字になる **)** 場合や、例外が発生する場合があります。大文字と小文字が区別される **XML** ドキュメントでは、連想配列表記法または **DOM** 表記法の名前を使用してください **(XmlRoot**、**XmlChldren[2]** など **)**。

式の左辺の最後の要素の参照

式の左辺の最後のコンポーネントは、次のルールに従って処理されます。

1 コンポーネント名が要素構造体のキー名 (XmlComment などの XML プロパティ名) の場合は、指定した要素構造体のエントリに式の右辺の値が設定されます。たとえば次の行では、mydoc.employee.name[1].first 要素の XML コメントが "This is a comment" に設定されます。

```
mydoc.employee.name[1].first.XmlComment = "This is a comment";
```

2 コンポーネント名として要素名が指定され、最後に数値インデックスが付いていない場合 (mydoc.employee.name など) は、最初に一致する要素に式の右辺の値が設定されます。

たとえば、mydoc.employee.name[1] と mydoc.employee.name[2] の両方が存在する場合、次の式を実行すると、(name という要素ではなく) mydoc.employee.name[1] が address という新しい要素で置き換えられます。

```
mydoc.employee.name = XmlElemNew(mydoc, "address");
```

mydoc.employee.name[1] と mydoc.employee.name[2] の両方が存在する場合、このコードの実行後は、元の mydoc.employee.name[2] の内容を持つ mydoc.employee.name 要素が 1 つだけ残ります。

3 コンポーネント名が既存の要素に一致していない場合は、式の左辺と右辺で要素名が一致している必要があります。これによって、式の左辺の要素名を持つ要素が作成されます。要素名が一致していない場合はエラーが発生します。

たとえば、mydoc.employee.name.phoneNumber 要素が存在しない場合、次の式を実行すると mydoc.employee.name.phoneNumber 要素が作成されます。

```
mydoc.employee.name.phoneNumber = XmlElemNew(mydoc, "phoneNumber");
```

次の式ではエラーが発生します。

```
mydoc.employee.name.phoneNumber = XmlElemNew(mydoc, "address");
```

4 コンポーネント名が既存の要素に一致しておらず、そのコンポーネントの親も同じく存在していない場合は、左辺に指定されている存在しない親ノードがすべて作成され、前述のルールが最後の要素に適用されます。たとえば、mydoc.employee.phoneNumber 要素が存在しない場合、次の式を実行すると、AreaCode 要素を含んだ phoneNumber 要素が作成されます。

```
mydoc.employee.name.phoneNumber.AreaCode = XmlElemNew(mydoc, "AreaCode");
```

CDATA 値の割り当てと取得

要素テキストを CDATA の開始マーカーと終了マーカーの中に置いて CDATA にするには、テキストを XmlText 要素ではなく XmlCdata 要素に割り当てます。XmlText エントリにテキストを割り当てると、要素テキストに含まれる記号 < および > がエスケープされますが、XmlCdata 要素に割り当てれば、エスケープされずにそのまま記述されます。1 つの要素では、XmlText エントリまたは XmlCdata エントリのいずれか一方にのみ値を代入できます。一方に値を代入するともう一方は上書きされます。

ドキュメントオブジェクトからデータを取得するときには、XmlCdata を参照しても XmlText を参照しても同じ文字列が返されます。

次の例に、ColdFusion での CDATA テキストの処理方法を示します。

```
<cfscript>
    myCDATA = "This is CDATA text";
    MyDoc = XmlNew();
    MyDoc.xmlRoot = XmlElemNew(MyDoc,"myRoot");
    MyDoc.myRoot.XmlChildren[1] = XmlElemNew(MyDoc,"myChildNodeCDATA");
    MyDoc.myRoot.XmlChildren[1].XmlCData = "#myCDATA#";
</cfscript>

<h3>Assigning a value to MyDoc.myRoot.XmlChildren[1].XmlCdata.</h3>
<cfoutput>
    The type of element MyDoc.myRoot.XmlChildren[1] is: #MyDoc.myRoot.XmlChildren[1].XmlType#<br>
    The value when output using XmlCdata is: #MyDoc.myRoot.XmlChildren[1].XmlCData#<br>
    The value when output using XmlText is: #MyDoc.myRoot.XmlChildren[1].XmlText#<br>
</cfoutput>
<br>
The XML text representation of Mydoc is:
<cfoutput><XMP>#tostring(MyDoc)#</XMP></cfoutput>

<h3>Assigning a value to MyDoc.myRoot.XmlChildren[1].XmlText.</h3>
<cfset MyDoc.myRoot.XmlChildren[1].XmlText = "This is XML plain text">
<cfoutput>
    The value when output using XmlCdata is: #MyDoc.myRoot.XmlChildren[1].XmlCData#<br>
    The value when output using XmlText is: #MyDoc.myRoot.XmlChildren[1].XmlText#<br>
</cfoutput>
<br>
The XML text representation of Mydoc is:
<cfoutput><XMP>#tostring(MyDoc)#</XMP></cfoutput>
```

XML ドキュメントオブジェクトの作成と保存

ここでは XML ドキュメントオブジェクトを作成して保存する方法について説明します。使用する方法は、アプリケーションやコーディングスタイルによって異なります。

cfxml タグによる XML ドキュメントの作成

cfxml タグを使用すると、タグ本文に記述されている XML マークアップに基づいて XML ドキュメントオブジェクトを作成できます。タグ本文には、CFML コードを含めることができます。CFML コードは、ColdFusion で処理されてから、その結果が XML に含められます。cfxml タグの簡単な使用例を次に示します。

```
<cfset testVar = True>
<cfxml variable="MyDoc">
    <MyDoc>
        <cfif testVar IS True>
            <cfoutput>The value of testVar is True.</cfoutput>
        <cfelse>
            <cfoutput>The value of testVar is False.</cfoutput>
        </cfif>
        <cfloop index = "LoopCount" from = "1" to = "4">
            <childNode>
                This is Child node <cfoutput>#LoopCount#.</cfoutput>
            </childNode>
        </cfloop>
    </MyDoc>
</cfxml>
<cfdump var=#MyDoc#>
```

この例では、ルート要素 MyDoc を持つドキュメントオブジェクトを作成します。このルート要素には、ColdFusion 変数 testVar の値を表示するテキストが含まれています。MyDoc には 4 つのネストした子要素があります。それらは cfloop タグのインデックスループによって生成されます。cfdump タグで、結果の XML ドキュメントオブジェクトを表示しています。

注意：cfxml タグの本文には <?xml ?> 処理ディレクティブを含めないでください。このディレクティブは不要であり、エラーが発生します。<?xml ?> ディレクティブを含んだ XML テキストを処理する場合は、XmlParse 関数を使用します。

XmlNew 関数による XML ドキュメントの作成

XmlNew 関数は XML ドキュメントオブジェクトを作成します。作成したオブジェクトには内容を設定する必要があります。新しい XML ドキュメントに内容を設定する方法については、1042 ページの「XML 要素の追加、削除、および変更」を参照してください。

注意：XmlNew 関数で作成した XML ドキュメントオブジェクトには、XmlDocType プロパティを設定できません。

次の例では、1037 ページの「cfxml タグによる XML ドキュメントの作成」と同じ ColdFusion ドキュメントオブジェクトを作成して表示します。

```
<cfset testVar = True>
<cfscript>
    MyDoc = XmlNew();
    MyDoc.xmlRoot = XmlElemNew(MyDoc,"MyRoot");
    if (testVar IS TRUE)
        MyDoc.MyRoot.XmlText = "The value of testVar is True.";
    else
        MyDoc.MyRoot.XmlText = "The value of testVar is False.";
    for (i = 1; i LTE 4; i = i + 1)
        {
        MyDoc.MyRoot.XmlChildren[i] = XmlElemNew(MyDoc,"childNode");
        MyDoc.MyRoot.XmlChildren[i].XmlText = "This is Child node " & i &".";
        }
</cfscript>
<cfdump var=#MyDoc#>
```

既存の XML による新しい XML ドキュメントの作成

XmlParse 関数を使用すると、テキストで表現された XML ドキュメントやその断片を、ColdFusion ドキュメントオブジェクトに変換できます。XML が格納された文字列変数を使用することも、テキストが格納されたファイル名または URL を使用することもできます。たとえば、cfhttp action="get" を使用して XML ドキュメントを取得したら、次のコードを使用して XML ドキュメントオブジェクトを作成できます。

```
<cfset myXMLDocument = XmlParse(cfhttp.fileContent)>
```

次の例では、ファイル内の XML テキストドキュメントを XML ドキュメントオブジェクトに変換します。

```
<cfset myXMLDocument=XmlParse("C:\temp\myxmldoc.xml" variable="XMLFileText")>
```

XmlParse 関数にはオプションの第 2 属性があり、ドキュメントオブジェクト内の要素や属性で大文字と小文字の区別を維持するかどうかを指定できます。デフォルトでは、ドキュメントオブジェクト内で大文字と小文字は区別されません。大文字と小文字の区別の詳細については、1034 ページの「XML オブジェクトの内容の参照」を参照してください。

XmlParse 関数では、DTD またはスキーマを指定して XML テキストを検証することもできます。有効な XML でない場合はエラーが発生します。検証用の DTD またはスキーマは、ファイル名または URL で指定することも、それらが格納されている CFML 変数で指定することもできます。また、XML テキスト内で指定されている DTD またはスキーマを使用することもできます。検証を指定する場合は、そのドキュメントで大文字と小文字が区別されるかどうかも指定します。次の例では、URL で指定した DTD を使用して、ファイル内の XML ドキュメントを検証します。

```
myDoc=XMLParse("C:\CFusion\wwwroot\examples\custorder.xml", false,
        "http://localhost:8500/examples/custorder.dtd")>
```

XML ドキュメントオブジェクトの保存とエクスポート

ToString 関数を使用すると、XML ドキュメントオブジェクトをテキスト文字列に変換できます。変換した文字列を変数に格納して、ColdFusion タグや関数で処理できます。

XML ドキュメントをファイルに保存するには、ToString 関数でドキュメントオブジェクトを文字列変数に変換してから、cffile タグでその文字列をファイルに保存します。たとえば次のコードでは、XML ドキュメントである myXMLDocument をファイル "C:¥temp¥myxmldoc.xml" に保存しています。

```
<cfset XMLText=ToString(myXMLDocument)>
<cffile action="write" file="C:\temp\myxmldoc.xml" output="#XMLText#">
```

ColdFusion XML オブジェクトの変更

XML ドキュメントオブジェクトは、通常の ColdFusion 構造化オブジェクトと同様に、さまざまな方法でその内容を変更できます。たとえば、構造体や配列の更新は、多くの場合、代入ステートメントでも関数でも行えます。XML ドキュメントオブジェクトの変更には、配列関数や構造体関数を使用できます。1040 ページの「XML ドキュメントオブジェクトの管理リファレンス」では、XML ドキュメントオブジェクトの内容を変更するための方法を一覧表で示します。その後のセクションでは、それらの変更方法について詳しく説明します。

XML オブジェクトの管理に使用できる関数

次の表に、XML ドキュメントオブジェクトやその機能を管理するときに使用できる ColdFusion 配列関数および構造体関数と、その一般的な用途を示します。操作によっては、配列関数と構造体関数のいずれを使用してもよい場合があります。たとえば、要素のすべての属性やすべての子要素を削除する場合などがあります。

関数	用途
ArrayLen	要素に含まれている子要素の数 (要素の XmlChildren 配列に含まれている要素の数) を調べます。
ArrayIsEmpty	要素の XmlChildren 配列に要素が存在するかどうかを調べます。
StructCount	要素の XmlAttributes 構造体に含まれている属性の数を調べます。
StructIsEmpty	要素の XmlAttributes 構造体に属性が存在するかどうかを調べます。 指定の構造体 (XML ドキュメントオブジェクトや要素も含む) が存在し、それが空であれば、True を返します。
StructKeyArray StructKeyList	要素の XmlAttributes 構造体に含まれているすべての属性の名前を列挙した配列またはリストを取得します。XML 要素の子の名前を返します。
ArrayInsertAt	要素の XmlChildren 配列の指定した場所に、新しい要素を追加します。
ArrayAppend ArrayPrepend	要素の XmlChildren 配列の末尾または先頭に、新しい要素を追加します。
ArraySwap	XmlChildren 配列の指定した場所にある 2 つの子要素を入れ替えます。
ArraySet	XmlChildren 配列の指定した範囲のエントリに、指定した要素構造体を設定します。配列の指定した範囲の各エントリに、構造体がコピーされます。1 つの要素を設定するには、範囲の先頭と末尾に同じインデックスを指定します。
ArrayDeleteAt	要素の XmlChildren 配列から、特定の要素を削除します。
ArrayClear	要素の XmlChildren 配列から、すべての子要素を削除します。
StructDelete	要素の XMLAttributes 構造体から、選択した属性を削除します。 要素の XmlChildren 配列から、特定の要素名を持つ子要素をすべて削除します。 要素のすべての属性を削除します。 要素のすべての子要素を削除します。 選択したプロパティ値を削除します。
StructClear	要素の XMLAttributes 構造体から、すべての属性を削除します。
Duplicate	XML ドキュメントオブジェクト、要素、またはノード構造体をコピーします。

関数	用途
IsArray	XmlChildren 配列を指定した場合は True を返します。mydoc.XmlRoot.name などの要素名を指定した場合は、XmlRoot に name 要素が複数ある場合でも、False を返します。
IsStruct	XML ドキュメントのオブジェクト、要素、およびノードを指定した場合は False を返します。XmlAttributes 構造体を指定した場合は True を返します。
StructGet	XML ドキュメントオブジェクト、要素、ノード、XmlAttributes 構造体などの、指定した構造体を返します。
StructAppend	ドキュメントフラグメントの XML ドキュメントオブジェクトを、別の XML ドキュメントオブジェクトの末尾に付加します。
StructInsert	XmlAttributes 構造体に新しいエントリを追加します。
StructUpdate	ドキュメントオブジェクトのプロパティ (XmlName など) の値や、XmlAttributes 構造体の指定した属性の値を、設定または置き換えます。

注意：この表と次のセクションの表に掲載されていない配列関数および構造体関数は、XML ドキュメントオブジェクト、XML 要素、および XML ノード構造体には**使用しないでください**。

配列による同名要素の操作

XML 要素には、同じ名前の子要素が複数含まれていることがあります。たとえば、多くの XML サンプルで使用されているサンプルドキュメントでは、employee 要素に複数の name 要素が含まれています。多くの場合、同じ名前の子要素は配列として操作できます。たとえば、mydoc.employee の 2 番目の name 要素は、mydoc.employee.name[2] で参照できます。ただし、この表記法を使用する場合は、限られた配列関数しか使用できません。この参照に使用できる配列関数を、次の表に示します。

配列関数	結果
IsArray(elemPath.elemName)	常に False を返します。
ArrayClear(elemPath.elemName)	elemPath 要素から、elemName という名前の要素をすべて削除します。
ArrayLen(elemPath.elemName)	elemPath 要素に含まれる、elemName という名前の要素の数を返します。
ArrayDeleteAt(elemPath.elemName, n)	elemPath 要素から、elemName という名前の n 番目の子要素を削除します。
ArrayIsEmpty(elemPath.elemName)	常に False を返します。
ArrayToList(elemPath.elemName, n)	elemPath 要素に含まれる、elemName という名前のすべての子要素に関するすべての XmlText プロパティをカンマ区切りリストとして返します。

XML ドキュメントオブジェクトの管理リファレンス

次の表は、XML ドキュメントオブジェクトの内容を変更する方法をまとめたクイックリファレンスです。この後のセクションで、XML の変更方法について詳しく説明します。

注意：大文字と小文字が区別される XML オブジェクトの要素名や属性名は、ドット表記法では参照できません。連想配列 (括弧) 表記法の名前を使用するか、大文字と小文字が区別される名前を使用せずに参照する (xmlChildren[1] などのようにする) 必要があります。

要素への情報の追加

新しい情報を要素に追加するには、次の方法を使用します。

タイプ	関数による方法	代入ステートメントによる方法

属性	`StructInsert(xmlElemPath.XmlAttributes, "key", "value")`	`xmlElemPath.XmlAttributes.key ="value"`
		
		`xmlElemPath.XmlAttributes["key"]="value"`

子要素	追加するには :
	`ArrayAppend(xmlElempath.XmlChildren,newElem)`
	挿入するには :
	`ArrayInsertAt(xmlElempath.XmlChildren, position, newElem)`
		追加するには :
		`xmlElemPath.XmlChildren[i] =newElem`
		
		`xmlElemPath.newChildName =newElem`
		(newChildName は newElem.XmlName と同じであることが必要です。また、name[3] のようなインデックス付きの名前は指定できません)

要素からの情報の削除

要素から情報を削除するには、次の方法を使用します。

タイプ	関数による方法	代入ステートメントによる方法
プロパティ	`StructDelete(xmlElemPath, propertyName)`	`xmlElemPath.propertyName=""`

属性	すべての属性 :
	`StructDelete(xmlElemPath, XmlAttributes)`
	
	指定の属性 :
	`StructDelete(xmlElemPath.XmlAttributes,"attributeName")`
		なし

子要素	要素のすべての子 :
	`StructDelete(xmlElemPath, "XmlChildren")`
	または
	`ArrayClear(xmlElemPath.XmlChildren)`
	
	指定の名前を持つすべての子 :
	`StructDelete(xmlElementpath,"elemName")`
	`ArrayClear(xmlElemPath.elemName)`
	
	指定の属性 :
	ArrayDeleteAt(xmlElemPath.XmlChildren,position)
	ArrayDeleteAt(xmlElemPath.elemName,position)
		なし

要素の内容の変更

要素の内容を変更するには、次の方法を使用します。

タイプ	関数による方法	代入ステートメントによる方法

プロパティ	`StructUpdate(xmlElemPath,"propertyName", "value")`	`xmlElemPath.propertyName ="value"`
		
		`xmlElemPath["propertyName"] ="value"`

属性	`StructUpdate(xmlElemPath.XmlAttributes, "attributeName", "value")`	`xmlElemPath.XmlAttributes.attributeName="value"`
		
		`xmlElemPath.XmlAttributes["attributeName"] = "value"`

子要素
(置換)
	`ArraySet(xmlElemPath.XmlChildren, index,index, newElement)`
	
	(1 つの要素を変更するには、両方のインデックスエントリに同じ値を使用します)
		elementName という名前の子要素 (複数ある場合は最初の要素) を置き換える :
		`parentElemPath.elementName =newElement`
		
		`parentElemPath["elementName"]= newElement`
		
		elementName という名前の子要素のうち、指定のものを置き換える :
		`parentElemPath.elementName[index] = newElement`
		または
		`parentElemPath["elementName"][index] = newElement`

XML 要素の追加、削除、および変更

ここでは、XML 要素を追加、削除、および変更する基本的な方法について説明します。これらの説明に含まれるコード例では、1029 ページの「単純な XML ドキュメント」で説明されている XML ドキュメントを使用しています。

子要素の数および位置の確認

XML 要素には、同じ名前の子要素が複数含まれていることがあります。たとえば、前に示した単純な XML ドキュメントでは、employee ルート要素に複数の name 要素が含まれています。

このようなオブジェクトを操作する場合は、同名の子要素がいくつ存在するかを調べたいことがあります。また、同名子要素のうちの特定の要素が XmlChildren 配列のどの位置にあるかを確認したいこともあります。

子要素の数の確認

次のユーザー定義関数を使用すれば、指定の要素に含まれている特定の名前の子要素の数を確認できます。

```
<cfscript>
function NodeCount (xmlElement, nodeName)
{
    nodesFound = 0;
    for (i = 1; i LTE ArrayLen(xmlElement.XmlChildren); i = i+1)
    {
        if (xmlElement.XmlChildren[i].XmlName IS nodeName)
            nodesFound = nodesFound + 1;
    }
    return nodesFound;
}
</cfscript>
```

次の行では、この関数を使用して、mydoc.employee 要素内の "name" というノードの数を表示します。

```
<cfoutput>
Nodes Found: #NodeCount(mydoc.employee, "name")#
</cfoutput>
```

同名子要素の位置の取得

XmlChildPos 関数を使用すれば、同名要素のうちの指定の要素が XmlChildren 配列のどの位置にあるかを確認できます。取得したインデックスを使用すれば、子要素の挿入や削除が行えます。たとえば、mydoc.employee に複数の name 要素がある場合、XmlChildren 配列における name[2] の位置を調べるには、次のコードを使用します。

```
<cfset nameIndex = XmlChildPos(mydoc.employee, "name", 2)>
```

要素の追加

要素を追加する方法には、要素を新たに作成する方法と、既存の要素に基づいて作成する方法があります。

新しい空の要素を作成するには、XmlElemNew 関数を使用します。この関数の形式は次のとおりです。

```
XmlElemNew(docObject, elementName)
```

docObject は、新しい要素を作成する XML ドキュメントオブジェクトの名前です。**elementName** は、新しい要素に付ける名前です。

既存の要素に基づいて新しい要素を作成するには、代入ステートメントを使用して、既存の要素を右辺に指定します。既存の要素に基づいて要素を追加する方法の詳細については、1044 ページの「既存の要素のコピー」を参照してください。

関数による要素の追加

ArrayInsertAt 関数または ArrayAppend 関数を使用すると、XML ドキュメントオブジェクトに要素を追加できます。たとえば次の行では、employee.name[2] の最後の要素の後に phoneNumber 要素を追加します。

```
<cfset ArrayAppend(mydoc.employee.name[2].XmlChildren, XmlElemNew(mydoc,
            "phoneNumber"))>
```

次の行では、employee の最初の要素として新しい department 要素を追加します。name 要素は 2 番目および 3 番目の要素になります。

```
<cfset ArrayInsertAt(mydoc.employee.XmlChildren, 1, XmlElemNew(mydoc,
            "department"))>
```

新しい要素の追加先の配列は、***parentElement*.**XmlChildren の形式で指定します。たとえば、次の行ではエラーが発生します。

```
<cfset ArrayInsertAt(mydoc.employee.name, 2, XmlElemNew(mydoc, "PhoneNumber")>
```

同名の子要素が複数ある場合、特定の位置に新しい要素を挿入するには、XmlChildPos 関数を使用して、XmlChildren 配列における要素の挿入位置を調べます。たとえば次のコードでは、mydoc.employee.name[1] の位置を調べて、新しい name 要素を 2 番目の name 要素として挿入します。

```
<cfscript>
nameIndex = XmlChildPos(mydoc.employee, "name", 1);
ArrayInsertAt(mydoc.employee.XmlChildren, nameIndex + 1, XmlElemNew(mydoc,
                    "name"));
</cfscript>
```

名前空間の使用 : 関数を使用して要素を追加するときは、要素名に名前空間の接頭辞を含めることで、その要素を名前空間に割り当てることができます。名前空間 URI が関連付けられていない接頭辞を使用する場合は、XmlElemNew 関数で名前空間 URI のパラメータも指定します。このパラメータはメソッドの第 **2** パラメータとして指定し、要素名は第 3 パラメータとして指定する必要があります。これによって、その名前空間接頭辞に URI が関連付けられるので、以降の xmlElemNew 関数呼び出しでは URI パラメータを省略できます。

次の例では、supplies ドキュメントルートに、Prod 名前空間に属する要素を 2 つ追加します。最初の XmlElemNew 関数で Prod 名前空間と URI を関連付けているので、2 回目の呼び出しでは要素名に接頭辞を付加するだけで済みます。

```
<cfscript>
    mydoc.supplies.XmlChildren[1] = XmlElemNew(mydoc,
        "http://www.foo.com/Products", "Prod:soap");
    mydoc.supplies.XmlChildren[2] = XmlElemNew(mydoc, "Prod:shampoo");
</cfscript>
```

直接代入による要素の追加

直接代入を使用して、要素配列の末尾に新規要素を追加できます。直接代入で要素を挿入することはできません。

直接代入を使用する場合、左辺の XmlChildren 配列のインデックスには、最後の子要素のインデックスより大きい任意の値を指定できます。たとえば、mydoc.employee に 2 つの要素がある場合は、mydoc.employee.XmlChildren[6] のように、2 よりも大きい数を指定できます。要素は常に、最後 (この例では 3 番目) の子として追加されます。

たとえば次の行では、name 要素が mydoc.employee の子要素の末尾に追加されます。

```
<cfset mydoc.employee.XmlChildren[9] = XmlElemNew(mydoc, "name")>
```

追加する子要素と同じ名前の子要素が親要素に存在していない場合は、代入式の左辺で新規ノードの名前を指定できます。たとえば次の行では、mydoc.employee の最初の name 要素の子に phoneNumber 要素が追加されます。

```
<cfset mydoc.employee.name[1].phoneNumber = XmlElemNew(mydoc, "phoneNumber")>
```

既存の子要素と同じ名前の要素を追加する場合、左辺でノード名を使用することはできません。たとえば、mydoc.employee に 2 つの name ノードがある場合、次の行はエラーになります。

```
<cfset mydoc.employee.name[3] = XmlElemNew(mydoc, "name")>
```

ただし、次の行は正常に実行されます。

```
<cfset mydoc.employee.XmlChilren[3] = XmlElemNew(mydoc, "name")>
```

既存の要素のコピー

既存の要素のコピーを、ドキュメント内の別の場所に追加できます。たとえば、mydoc.employee.name[1].phoneNumber 要素があり、mydoc.employee. name[2].phoneNumber がない場合、次の行を実行すると、元の要素と同じ値を持つ mydoc.employee. name[2].phoneNumber 要素が作成されます。この代入式によって、元の要素がコピーされます。標準の ColdFusion 構造体とは異なり、元の構造体への参照ではなく、完全なコピーが複製されます。コピーを変更しても、元の要素は変更されません。

```
<cfset mydoc.employee.name[2].phoneNumber = mydoc.employee.name[1].phoneNumber>
```

要素をコピーする場合、新しい要素は既存の要素と同じ名前であることが必要です。代入式の左辺で新規要素の名前を指定する場合、その要素名は右辺の要素名と同一であることが必要です。たとえば、次の式ではエラーが発生します。

```
<cfset mydoc.employee.name[2].telephone = mydoc.employee.name[1].phoneNumber>
```

要素の削除

ここでは、個々の要素または複数の要素を削除する方法について説明します。

個々の要素の削除

XML ドキュメントオブジェクトから特定の要素を削除するには、ArrayDeleteAt 関数を使用します。たとえば次の行は、mydoc.employee 要素の 2 番目の子要素を削除します。

```
<cfset ArrayDeleteAt(mydoc.employee.XmlChildren, 2)>
```

また、特定の名前を持つ子要素が 1 つしかない場合は、StructDelete 関数を使用してその子要素を削除できます。たとえば次の行は、2 番目の employee.name 要素の phoneNumber 要素を削除します。

```
<cfset StructDelete(mydoc.employee.name[2], "phoneNumber")>
```

同じ名前の子要素が複数ある場合は、要素の位置を指定します。この位置は、同名要素内での位置で指定することも、すべての子要素内での位置で指定することもできます。たとえば、mydoc.employee の 2 番目の name 要素を削除するには、次の行を使用します。

```
<cfset ArrayDeleteAt(mydoc.employee.name, 2)>
```

また、削除する要素の XmlChildren 配列における位置を調べて、その位置を使用することもできます。これを調べるには XmlChildPos 関数を使用します。たとえば次の行は、mydoc.employee.name[2] の位置を調べ、その要素を削除します。

```
<cfset idx = XmlChildPos(mydoc.employee, "name", 2)>
<cfset ArrayDeleteAt(mydoc.employee.XmlChildren, idx)>
```

複数の要素の削除

同名の子要素が複数ある場合、その名前の子要素をすべて削除するには、StructDelete 関数または ArrayClear 関数で要素名を指定します。たとえば、次のどちらの行でも、employee 構造体からすべての name 要素が削除されます。

```
<cfset StructDelete(mydoc.employee, "name")>
<cfset ArrayClear(mydoc.employee.name)>
```

要素の子要素をすべて削除するには、StructDelete 関数または ArrayClear 関数で XmlChildren を指定します。たとえば、次のどちらの行でも、mydoc.employee.name[2] 要素のすべての子要素が削除されます。

```
<cfset StructDelete(mydoc.employee.name[2], "XmlChildren")>
<cfset ArrayClear(mydoc.employee.name[2].XmlChildren)>
```

要素の属性の追加、変更、および削除

要素の属性を変更する方法は、構造体の内容を変更する方法と同じです。たとえば、次のどちらの行でも、2 番目の mydoc.employee.name 要素に Status 属性が追加されます。

```
<cfset mydoc.employee.name[2].XmlAttributes.Status="Inactive">
<cfset StructInsert(mydoc.employee.name[2].XmlAttributes, "Status", "Inactive")>
```

属性を変更するには、標準の代入ステートメントを使用します。次に例を示します。

```
<cfset mydoc.employee.name[2].XmlAttributes.Status="Active">
```

属性を削除するには、StructDelete を使用します。次に例を示します。

```
<cfset StructDelete(mydoc.employee.name[1].XmlAttributes, "Status")>
```

要素プロパティの変更

要素のプロパティ (テキストやコメントも含む) を変更するには、標準の代入式を使用します。たとえば、mydoc.employee の XML コメントに "MyCompany Documentation Department" を追加するには、次の行を使用します。

```
<cfset mydoc.employee.XmlComment = mydoc.employee.XmlComment & "in the
MyCompany Documentation Department">
```

要素名の変更

XML DOM では、要素名の変更は直接サポートされていません。要素の名前を変更するには、新しい名前で要素を作成し、XML ドキュメントオブジェクトの元の要素の前または後ろに挿入し、元の要素の内容をすべて新規要素にコピーし、元の要素を削除します。

要素のプロパティ値のクリア

要素のプロパティ値をクリアするには、そのプロパティに空の文字列を代入するか、StructDelete 関数を使用します。たとえば、次のどの行でも、mydoc.employee からコメント文字列がクリアされます。

```
<cfset mydoc.employee.XmlComment = "">
<cfset StructDelete(mydoc.employee, "XmlComment")>
```

要素の置換または移動

要素を新しい要素で置換するには、標準の置換式を使用します。たとえば、mydoc.employee.department 要素を organization という新規要素に置換するには、次のいずれかの行を使用します。

```
<cfset mydoc.employee.department = XmlElemNew(mydoc, "Organization")>
<cfset mydoc.employee.XmlChildren[1] = XmlElemNew(mydoc, "Organization")>
```

要素を既存の要素のコピーで置換するには、式の右辺に既存の要素を指定します。たとえば次の行は、mydoc.employee.name[2] の phoneNumber 要素を mydoc.employee.name[1] の phoneNumber 要素に置換します。

```
<cfset mydoc.employee.name[2].phoneNumber=mydoc.employee.name[1].phoneNumber>
```

このコードによって、name[2].phoneNumber に name[1].phoneNumber 要素の完全なコピーが作成されます。

要素を移動するには、その要素を新しい場所に割り当てて、元の場所から削除します。たとえば次の行は、phoneNumber 要素を mydoc.employee.name[1] から mydoc.employee.name[2] に移動します。

```
<cfset mydoc.employee.name[2].phoneNumber=mydoc.employee.name[1].phoneNumber>
<cfset StructDelete(mydoc.employee.name[1], "phoneNumber")>
```

注意：あるドキュメント内の要素を、別のドキュメントにコピーまたは移動することはできません。

XML と ColdFusion クエリーの使用

XML ドキュメントを ColdFusion クエリーオブジェクトに変換して、クエリーオブクエリーで操作することができます。この方法では XPath を使用する必要がなく、ColdFusion プログラマにとって違和感のない方法で XML ドキュメントの検索やデータの抽出ができます。

XML から ColdFusion クエリーへの変換

次の例では、XML ドキュメントを読み取ってクエリーオブジェクトに変換し、そのオブジェクトにクエリーオブクエリーを実行して、選択したデータを取得します。

```
<!--- Read the file and convert it to an XML document object --->
<cffile action="read" file="C:\CFusion\wwwroot\myexamples\employees.xml" variable="myxml">
<cfset mydoc = XmlParse(myxml)>

<!--- get an array of employees --->
<cfset emp = mydoc.employee.XmlChildren>
<cfset size = ArrayLen(emp)>

<cfoutput>
Number of employees = #size#
<br>
</cfoutput>
<br>
<!--- create a query object with the employee data --->
<cfset myquery = QueryNew("fname, lname") >
<cfset temp = QueryAddRow(myquery, #size#)>
<cfloop index="i" from = "1" to = #size#>
    <cfset temp = QuerySetCell(myquery, "fname",
        #mydoc.employee.name[i].first.XmlText#, #i#)>
    <cfset temp = QuerySetCell(myquery, "lname",
        #mydoc.employee.name[i].last.XmlText#, #i#)>
</cfloop>

<!--- Dump the query object --->
Contents of the myquery Query object: <br>
<cfdump var=#myquery#>
<br><br>

<!--- Select entries with the last name starting with A and dump the result --->
<cfquery name="ImqTest" dbType="query">
    SELECT lname, fname
    FROM myquery
    WHERE lname LIKE 'A%'
</cfquery>
<cfdump var=#imqtest#>
```

クエリーオブジェクトから XML への変換

次の例では、クエリーオブジェクトを XML に変換する方法を示します。cfquery を使用して cfdocexamples データベースから従業員リストを取得し、その情報を XML ドキュメントとして保存します。

```
<!--- Query the database and get the names in the employee table --->
<cfquery name="myQuery" datasource="cfdocexamples">
    SELECT FirstName, LastName
    FROM employee
</cfquery>

<!--- Create an XML document object containing the data --->
<cfxml variable="mydoc">
    <employee>
        <cfoutput query="myQuery">
                <name>
                <first>#FirstName#</first>
                <last>#LastName#</last>
            </name>
        </cfoutput>
    </employee>
</cfxml>

<!--- dump the resulting XML document object --->
<cfdump var=#mydoc#>
<!--- Write the XML to a file --->
\\x
        output=#toString(mydoc)#>
```

XML ドキュメントの検証

ColdFusion には、DTD または XML スキーマと照合してドキュメントを検証できる次のメソッドが用意されています。

- XmlParse 関数では、XML テキストを DTD またはスキーマと照合して検証できます。この関数で検証エラーが発生するとエラーが生成され、テキストの解析は中断されます。エラーはないが警告が生成された場合は、ドキュメントが解析されて、その結果が返されます。
- XmlValidate 関数では、XML テキストドキュメントまたは XML ドキュメントオブジェクトを DTD またはスキーマと照合して検証できます。この関数は、検証の詳細情報が含まれたデータ構造体を返します。この構造体には、警告、エラー、致命的エラーの配列と、ドキュメントが有効かどうかを示す Boolean ステータス変数が含まれています。アプリケーションでは、このステータス情報を調べて、以降の処理方法を判断することができます。

XML 検証の例については、『CFML リファレンス』の XmlParse および XmlValidate を参照してください。XmlParse には、DTD を使用したドキュメント検証の例が掲載されています。XmlValidate には、その DTD と同じドキュメント構造を表す XML スキーマを使用したドキュメント検証の例が掲載されています。

XSLT によるドキュメントの変換

XSLT (Extensible Stylesheet Language Transformation) テクノロジーを使用すれば、XML ドキュメントを別の形式や表現方式に変換できます。たとえば、XML ドキュメントを変換して、ブラウザ表示用の HTML を生成する場合に、XSLT がよく使用されます。XSLT はその他の用途にも活用できます。たとえば、受注管理アプリケーションの語彙で記述された XML データを、受注処理アプリケーションの語彙に変換することができます。

XSLT は、XSL (Extensible Stylesheet Language) スタイルシートを適用することで、XML ドキュメントを変換します。XSL スタイルシートをファイルに保管するときは、通常、拡張子 xsl を使用します。ColdFusion には、XML ドキュメントに XSL 変換を適用する XmlTransform 関数が用意されています。この関数に、XML ドキュメント (文字列形式または XML ドキュメントオブジェクト) と XSL スタイルシート (文字列形式) を渡すと、変換されたドキュメントが文字列として返されます。

次のコードでは、次の処理を行っています。

1 simpletransform.xsl スタイルシートファイルを文字列変数に読み込みます。

2 このスタイルシートを使用して mydoc XML ドキュメントオブジェクトを変換します。

3 変換されたドキュメントを別のファイルに保存します。

```
<cffile action="read" file="C:\CFusion\wwwroot\testdocs\simpletransform.xsl"
    variable="xslDoc">
<cfset transformedXML = XmlTransform(mydoc, xslDoc)>
<cffile action="write" file="C:\CFusion\wwwroot\testdocs\transformeddoc.xml"
    output=transformedXML>
```

XSL および XSLT は、W3C (World Wide Web Consortium) によって仕様が定められています。XSL、XSLT、および XSL スタイルシートの詳細については、W3C の Web サイト www.w3.org/Style/XSL/ を参照してください。また、XSL および XSLT の使用方法に関する本も市販されています。

XPath によるデータの抽出

XPath は、XML ドキュメントの特定の部分を参照するための言語です。XSL と同様に、XPath も W3C の仕様です。XPath は、XSL 変換で使用することを主な目的としていますが、より一般的な用途にも活用できます。特に、XPath を使用すれば、複雑なデータセット表現などを使用して、XML ドキュメントからデータを抽出できます。つまり、XPath はデータクエリーツールとして使用できます。

XPath は、XPath 式というパターンを使用して、XML ドキュメントから抽出する情報を指定します。たとえば、簡単な XPath 表現である /employee/name は、employee ルート要素の name 要素を選択します。

XmlSearch 関数では、XPath 式を指定して、XML ドキュメントオブジェクトからデータを抽出します。この関数に、XML ドキュメントオブジェクトと文字列形式の XPath 式を渡すと、XPath 式と XML のマッチングの結果が返されます。結果は、XML オブジェクトノードの配列やブール値など、ColdFusion が表すことのできる XPath の戻り型で返されます。詳細については、『CFML リファレンス』の XmlSearch を参照してください。

次の例では、従業員の姓が含まれている last という要素を "employeesimple.xml" ファイルからすべて抽出し、名前を表示します。

```
<cffile action="read"
    file="C:\inetpub\wwwroot\examples\employeesimple.xml"
    variable="myxml">
<cfscript>
    myxmldoc = XmlParse(myxml);
    selectedElements = XmlSearch(myxmldoc, "/employee/name/last");
    for (i = 1; i LTE ArrayLen(selectedElements); i = i + 1)
        writeoutput(selectedElements[i].XmlText & "<br>");
</cfscript>
```

XPath は W3C によって仕様が定められています。XPath の詳細については、W3C の Web サイト www.w3.org/TR/xpath を参照してください。XSLT に関するほとんどの本では、XPath についても説明されています。

例 : ColdFusion アプリケーションでの XML の使用

次の例では、XML でデータを表現する方法と、アプリケーションで XML データを使用する方法を示します。この例は非常に単純なので、実際にアプリケーションで使用するにはかなりの変更が必要ですが、ColdFusion での XML の一般的な使用方法が示されています。

この例では、XML ドキュメントの形式で注文を受け取り、それを処理して、XML ドキュメントの領収書を生成します。この例では、注文ドキュメントがファイルとして用意されていますが、HTTP リクエストの結果として受信したり、cfpop や cfftp などのメソッドで取得したりすることもできます。ColdFusion ページでは、注文に対して次の処理を行っています。

1 XML ドキュメントからクエリーオブジェクトを生成します。

2 データベーステーブルをクエリーして、注文の割引率を決めます。

3 クエリーオブクエリーを使用して合計額を計算した後、割引価格を計算します。

4 領収書を XML ドキュメントとして生成します。

この例では、処理の各段階で結果を表示して、処理内容を示しています。

XML ドキュメント

"order.xml" ドキュメントは次のような構造になっています。

- ルート要素の名前は order で、1 つの属性 id を持っています。
- 1 つの customer 要素があり、firstname、lastname、および accountnum 属性を持っています。customer 要素には本文がありません。
- 1 つの items 要素があり、その中に複数の item 要素が含まれています。
- 各 item 要素は id 属性を持っており、name、quantity、および unitprice 要素が含まれています。name、quantity、および unitprice 要素では、その値が本文テキストとして含まれています。

次の "order.xml" ドキュメントは cfdocexamples データベース内の情報を使用して正常に動作します。

```
<order id="4323251">
    <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
    <items>
        <item id="43">
            <name>
                Large Hammer
            </name>
            <quantity>
                1
            </quantity>
            <unitprice>
                15.95
            </unitprice>
        </item>
        <item id="54">
            <name>
                Ladder
            </name>
            <quantity>
                2
            </quantity>
            <unitprice>
                40.95
            </unitprice>
        </item>
        <item id="68">
            <name>
                Paint
            </name>
            <quantity>
                10
            </quantity>
            <unitprice>
                18.95
            </unitprice>
        </item>
    </items>
</order>
```

ColdFusion ページ

ColdFusion ページは次のようになります。

```
<!--- Convert file to XML document object --->
<cffile action="read" file="C:\CFusion\wwwroot\examples\order.xml" variable="myxml">
<cfset mydoc = XmlParse(myxml)>

<!--- Extract account number --->
    <cfset accountNum=#mydoc.order.customer.XmlAttributes.accountNum#>
<!--- Display Order Information --->
<cfoutput>
    <b>Name=</b>#mydoc.order.customer.XmlAttributes.firstname#
     #mydoc.order.customer.XmlAttributes.lastname#
    <br>
    <b>Account=</b>#accountNum#
    <br>
    <cfset numItems = ArrayLen(mydoc.order.items.XmlChildren)>
    <b>Number of items ordered=</b> #numItems#
</cfoutput>
<br><br>

<!--- Process the order into a query object --->
<cfset orderquery = QueryNew("item_Id, name, qty, unitPrice") >
<cfset temp = QueryAddRow(orderquery, #numItems#)>
<cfloop index="i" from = "1" to = #numItems#>
    <cfset temp = QuerySetCell(orderquery, "item_Id",
        #mydoc.order.items.item[i].XmlAttributes.id#,#i#)>
    <cfset temp = QuerySetCell(orderquery, "name",
        #mydoc.order.items.item[i].name.XmlText#, #i#)>
    <cfset temp = QuerySetCell(orderquery, "qty",
        #mydoc.order.items.item[i].quantity.XmlText#, #i#)>
    <cfset temp = QuerySetCell(orderquery, "unitPrice",
        #mydoc.order.items.item[i].unitprice.XmlText#, #i#)>
</cfloop>

<!--- Display the order query --->
<cfdump var=#orderquery#>
<br><br>

<!--- Determine the discount --->
<cfquery name="discountQuery" datasource="cfdocexamples">
    SELECT *
    FROM employee
    WHERE Emp_Id = #accountNum#
</cfquery>
<cfset drate = 0>
<cfif #discountQuery.RecordCount# is 1>
    <cfset drate = 10>
</cfif>

<!--- Display the discount rate --->
<cfoutput>
    <b>Discount Rate =</b> #drate#%
</cfoutput>
<br><br>

<!--- Compute the total cost and discount price--->
<cfquery name="priceQuery" dbType="query">
    SELECT SUM(qty*unitPrice)
    AS totalPrice
    FROM orderquery
</cfquery>
<cfset discountPrice = priceQuery.totalPrice * (1 - drate/100)>

<!--- Display the full price and discounted price --->
```

```
<cfoutput>
    <b>Full Price=</b> #priceQuery.totalPrice#<br>
    <b>Discount Price=</b> #discountPrice#
</cfoutput>
<br><br>

<!---Generate an XML Receipt --->
<cfxml variable="receiptxml">
<receipt num = "34">
    <cfoutput>
        <price>#discountPrice#</price>
        <cfif drate GT 0 >
          <discountRate>#drate#</discountRate>
        </cfif>
    </cfoutput>
    <itemsFilled>
        <cfoutput query="orderQuery">
                <name>#name# </name>
                <qty> #qty# </qty>
                <price> #qty*unitPrice# </price>
        </cfoutput>
    </itemsFilled>
</receipt>
</cfxml>

<!--- Display the resulting receipt --->
<cfdump var=#receiptxml#>
```

コードの説明

この CFML コードおよびその機能について、次の表で説明します。簡略化するために、処理結果を表示するコードは除外しています。

コード	説明
<cffile action="read" file="C:\CFusion\wwwroot\examples\order.xml" variable="myxml"> <cfset mydoc = XmlParse(myxml)> <!--- Extract account number ---> <cfset accountNum=#mydoc.order.customer.XmlAttributes.accountNum#>	ファイルから XML を読み取り、XML ドキュメントオブジェクトに変換します。 顧客エントリの accountnum 属性から accountNum 変数を設定します。
<cfset orderquery = QueryNew("item_Id, name, qty, unitPrice") > <cfset temp = QueryAddRow(orderquery, #numItems#)> <cfloop index="i" from = "1" to = #numItems#> <cfset temp = QuerySetCell(orderquery, "item_Id", #mydoc.order.items.item[i].XmlAttributes.id#,#i#)> <cfset temp = QuerySetCell(orderquery, "name", #mydoc.order.items.item[i].name.XmlText#, #i#)> <cfset temp = QuerySetCell(orderquery, "qty", #mydoc.order.items.item[i].quantity.XmlText#, #i#)> <cfset temp = QuerySetCell(orderquery, "unitPrice", #mydoc.order.items.item[i].unitprice.XmlText#, #i#)> </cfloop>	XML ドキュメントオブジェクトをクエリーオブジェクトに変換します。 品目ごとにクエリーを作成し、item_id、name、qty、および unitPrice 値の列を設定します。 mydoc.order.items エントリ内の XML item エントリごとに、品目 id の属性と、name、quantity、および unitprice エントリのテキストをクエリー行に設定します。

コード	説明
<cfquery name="discountQuery" datasource="cfdocexamples"> SELECT * FROM employee WHERE Emp_Id = #accountNum# </cfquery> <cfset drate = 0> <cfif #discountQuery.RecordCount# is 1> <cfset drate = 10> </cfif>	アカウント番号が cfdocexamples データベースの Employee テーブルにある従業員 ID と同じ場合、クエリーは 1 つのレコードを返します。RecordCount は 1 です。この場合、割引率を 10% に設定します。それ以外の場合は、割引率を 0% に設定します。
<cfquery name="priceQuery" dbType="query"> SELECT SUM(qty*unitPrice) AS totalPrice FROM orderquery </cfquery> <cfset discountPrice = priceQuery.totalPrice * (1 - drate/100)>	注文品目の割引率を計算する前に、クエリーオブクエリーと SUM 演算子を使用して合計金額を計算します。その後で、合計に割引を適用します。クエリーの結果は、1 つの値、つまり合計金額です。
<cfxml variable="receiptxml"> <receipt num = "34"> <cfoutput> <price>#discountPrice#</price> <cfif drate GT 0 > <discountRate>#drate#</discountRate> </cfif> </cfoutput> <itemsFilled> <cfoutput query="orderQuery"> <name>#name# </name> <qty> #qty# </qty> <price> #qty*unitPrice# </price> </cfoutput> </itemsFilled> </receipt> </cfxml>	領収書の XML ドキュメントオブジェクトを作成します。この領収書は、receipt というルート要素を持っています。その属性にレシート番号が設定されています。receipt 要素には、注文金額を表す price 要素と、品目ごとに 1 つの itemsFilled 要素が含まれています。

WDDX による Web 上での複合データの移動

WDDX は、配列、連想配列 (ColdFusion 構造体など)、レコードセットなどの複合データ構造体を一般的な方法で記述するための XML 語彙です。これによって、異なるアプリケーションサーバープラットフォーム間や、アプリケーションサーバーとブラウザ間で、HTTP を使用してデータを移動できます。WDDX のターゲットプラットフォームには、ColdFusion、ASP (Active Server Pages)、JavaScript、Perl、Java、Python、COM、Flash、PHP などがあります。

WDDX の XML 語彙は、標準データタイプの構造を記述した DTD (ドキュメントタイプ定義) と、各ターゲットプラットフォームで次の処理を行うためのコンポーネントのセットから構成されています。

- **シリアル化 :** ネイティブ形式で表現されたデータを、WDDX XML ドキュメントまたはドキュメントの一部に変換します。
- **シリアル化解除 :** WDDX XML ドキュメントまたはドキュメントの一部を、ネイティブなデータ表現 (CFML 構造体など) に変換します。

この XML 語彙によって、データ、関連付けられているデータタイプ、および記述子の移動が可能になり、それらのデータを任意のアプリケーションサーバー間のターゲットシステム上で操作することができます。

注意 : WDDX DTD (マニュアルも含む) は、www.openwddx.org/downloads/dtd/wddx_dtd_10.txt で公開されています。

WDDX は、ColdFusion 開発者にとって便利なツールですが、CFML 以外の用途にも活用できます。一般的なプログラミングデータ構造 (配列、レコードセット、構造体など) を WDDX 形式にシリアル化すれば、言語やプラットフォームの範囲を超えて、HTTP でデータを転送することができます。また、複合データをデータベース、ファイル、クライアント変数に保存することもできます。

WDDX には、Web 環境でのデータ転送に適した 2 つの特長があります。

- 軽量です。データのシリアル化およびシリアル化解除に使用する JavaScript (WDDX データをダンプするデバッグ関数を含む) は 22 KB 以下です。

- 従来のクライアント / サーバー型アプローチとは異なり、ソースシステムとターゲットシステムの間で、互いのことを理解する必要はほとんどありません。理解する必要があるのは、転送するデータの構造だけです。

WDDX は 1998 年に開発されましたが、現在では、多くのアプリケーションが WDDX 機能を備えています。WDDX に関する詳細な情報は、www.openwddx.org から入手できます。このサイトでは、WDDX の DTD および SDK を無償でダウンロードできます。また、WDDX FAQ、開発者のフォーラム、WDDX リソースを提供しているサイトへのリンクなどのリソースが提供されています。

WDDX の用途

WDDX は、アプリケーション間で複合データを転送するときに役立ちます。たとえば、CFML アプリケーションと CGI または PHP アプリケーションの間でデータをやり取りする場合に使用できます。また、サーバーとクライアントサイド JavaScript の間でデータを転送する場合にも役立ちます。

アプリケーションサーバー間でのデータ交換

WDDX は、異なるアプリケーションサーバープラットフォーム間で、構造化された複合データをシームレスに転送する場合に役立ちます。たとえば、ColdFusion ベースのアプリケーションでは、cfwddx を使用して発注書を WDDX に変換し、CGI ベースのシステムを使用しているサプライヤに cfhttp で送信することができます。

サプライヤは、WDDX のシリアル化を解除してネイティブのデータ形式に変換し、注文書から情報を抽出して、ASP ベースのアプリケーションを使用している配送業者に渡すことができます。

サーバーとブラウザ間でのデータ転送

WDDX を使用して、サーバーとブラウザの間でデータを転送することができます。サーバーデータを WDDX 形式に変換してブラウザに送信し、ブラウザで JavaScript オブジェクトに変換できます。同様に、ブラウザ上の JavaScript データをアプリケーションページでシリアル化して WDDX 形式に変換し、アプリケーションサーバーに送信できます。サーバーでは WDDX XML のシリアル化を解除して CFML データに変換できます。

サーバー上で WDDX データをシリアル化およびシリアル化解除するには、cfwddx タグを使用します。ブラウザ上で JavaScript データをシリアル化して WDDX に変換するには、JavaScript ユーティリティクラスの WddxSerializer および WddxRecordset を使用します。これらのユーティリティクラスは、サーバー上の "webroot/CFIDE/scripts/wddx.js" にインストールされています。

WDDX と Web サービス

WDDX は Web サービスと競合するものではありません。WDDX は、アプリケーションの統合に関する問題に特化した補完技術であり、Web 上で安価に実用的かつ生産的な方法でデータを共有するための方法を提供します。

WDDX には、次の利点があります。

- ブラウザや Flash Player などの軽量なクライアントで使用できます。
- 複雑なデータ構造をファイルやデータベースに保管できます。

WDDX を利用したアプリケーションでは、Web サービスを使用することになっても、WDDX を使用し続けることができます。また、そうしたアプリケーションを変換して、Web サービス標準のみを使用するようにすることもできます。その場合でも、サービスおよびデータ交換形式は変更が必要ですが、アプリケーションモデルを変更する必要はありません。

WDDX の仕組み

次の例で、WDDX の仕組みを示します。2 つの文字列変数を含んだ単純な構造体をシリアル化して WDDX XML 表現に変換すると、たとえば次のような形式になります。

```
<var name='x'>
    <struct>

        <var name='a'>
            <string>Property a</string>
        </var>
        <var name='b'>
            <string>Property b</string>
        </var>
    </struct>
</var>
```

この XML のシリアル化を解除して CFML または JavaScript に変換すると、次のスクリプトで生成されるのと同じ構造体が作成されます。

JavaScript	CFScript
x = new Object(); x.a = "Property a"; x.b = "Property b";	x = structNew(); x.a = "Property a"; x.b = "Property b";

逆に、これらのスクリプトによって生成される変数をシリアル化して WDDX に変換すると、前述の XML が生成されます。

ColdFusion には、CFML、WDDX、および JavaScript の間で変換を行うためのタグおよび JavaScript オブジェクトが用意されています。他のデータ形式のシリアル化およびシリアル化の解除は、Web 上で提供されています。詳細については、www.openwddx.org を参照してください。

注意：cfwddx タグおよび "wddx.js" の JavaScript 関数では、データ表現に UTF-8 エンコーディングを使用しています。ColdFusion によって生成された WDDX のシリアル化を解除するツールでは、UTF-8 でエンコードされた文字を処理できる必要があります。UTF-8 エンコードは、128 個の標準の "7 ビット " ASCII 文字については、ASCII および ISO 8859 のシングルバイトエンコードと同一です。ただし、最上位ビットが 1 である "high-ASCII" ISO 8859 文字は、UTF-8 では 2 バイト文字で表現されます。

WDDX でのデータ型のサポート

次に、WDDX でサポートされているデータ型について説明します。この情報は、WDDX DTD の説明からの抜粋です。詳細については、www.openwddx.org の DTD を参照してください。

基本データ型

WDDX では、次の基本データ型を表現できます。

データ型	説明
Null	WDDX の Null 値は、数値や文字列などの型に関連付けられません。cfwddx タグでは、WDDX の Null 値は空の文字列に変換されます。
数値	WDDX ドキュメントでは、浮動小数点数によってすべての数値が表現されます。数値の範囲は、±1.7E±308 に制限されています。精度は小数点以下 15 桁です。
日付時刻値	日付時刻値は、ISO8601 の完全な形式に基づいてエンコードされます。たとえば、2002-9-15T09:05:32+4:0 のようになります。
文字列	文字列の長さに制限はありませんが、null を含むことはできません。文字列は、2 バイト文字を使用してエンコードできます。

複合データ型

WDDX では次の複合データ型を表現できます。

データ型	説明
配列	配列は、任意の型のオブジェクトのコレクションで、整数インデックスが付いています。多くの言語の配列インデックスは 0 から始まりますが、CFML の配列インデックスは 1 から始まります。したがって、配列インデックスを使用するとデータの互換性がなくなる可能性があります。
構造体	構造体は、任意の型のオブジェクトのコレクションで、文字列インデックスが付いています。連想配列とも呼ばれます。WDDX でサポートされている言語の中には大文字と小文字が区別されないものもあるので、構造体内の変数名を大文字と小文字の違いのみで区別することはできません。
レコードセット	レコードセットは、ColdFusion のクエリーオブジェクトに対応する、名前付きフィールドのテーブル列です。レコードセットに保管できるのは、単純データ型のみです。WDDX でサポートされている言語の中には大文字と小文字が区別されないものもあるので、レコードセット内のフィールド名を大文字と小文字の違いのみで区別することはできません。フィールド名は、正規表現 [_A-Za-z][_.0-9A-Za-z]* の条件を満たす必要があります。この正規表現に含まれるピリオド (.) は、" 任意の文字 " ではなく、ピリオド文字そのものを意味しています。
バイナリ	バイナリは、バイナリデータの文字列 (blob) を表します。データは MIME base64 形式でエンコードされます。

データ型の比較

次の表に、WDDX の基本データ型と、Web で広く使用されている言語や技術のデータ型の比較を示します。

WDDX	CFML	XML スキーマ	Java	ECMAScript/ JavaScript	COM
null	N/A	N/A	null	null	VT_NULL
boolean	ブール値	boolean	java.lang.Boolean	ブール値	VT_BOOL
number	数値	number	java.lang.Double	数値	VT_R8
dateTime	日付時刻値	dateTime	java.lang.Date	日付	VT_DATE
string	文字列	string	java.lang.String	文字列	VT_BSTR
array	配列	N/A	java.lang.Vector	配列	VT_ARRAY | VT_VARIANT
struct	構造体	N/A	java.lang. Hashtable	オブジェクト	IWDDXStruct
recordset	Query オブジェクト	N/A	coldfusion.runtime. QueryTable	WddxRecordset	IWDDXRecordset
binary	バイナリ	binary	byte []	WddxBinary	V_ARRAY | UI1

タイムゾーンの処理

WDDX パケットの作成者と利用者が地理的に離れた場所にいることもあります。したがって、データのシリアル化やシリアル化解除を行うときは、タイムゾーン情報を使用して、日付時刻値を正しく処理することが重要です。

日付時刻データをシリアル化するときにタイムゾーン情報を使用するかどうかは、cfwddx action=cfml2wddx タグの useTimezoneInfo 属性で指定します。JavaScript 実装の場合、useTimezoneInfo は WddxSerializer オブジェクトのプロパティです。いずれの場合も、useTimezoneInfo のデフォルト値は True です。

WDDX の日付時刻値は、ISO8601 形式のサブセットを使用して表します。タイムゾーン情報は、UTC (Universal Time : 協定世界時) に対するオフセットの時間を、"2002-9-8T12:6:26-4:0" のように表現します。

cfwddx タグで WDDX をシリアル化解除して CFML に変換するときには、利用可能なタイムゾーン情報が自動的に使用されて、ローカルの日付時刻値に変換されます。したがって、タイムゾーン変換の詳細に配慮する必要はありません。

ただし、ColdFusion の JavaScript オブジェクトで WDDX のシリアル化を解除して JavaScript 表現に変換するときは、タイムゾーン情報が使用されません。これは、JavaScript ではブラウザのタイムゾーンを取得するのが困難なためです。

WDDX の使用

ColdFusion には、一般的な用途で WDDX を作成または変換するためのさまざまなツールが用意されています。

cfwddx タグの使用

このタグでは、次の変換が行えます。

変換元	変換後
CFML	WDDX
CFML	JavaScript
WDDX	CFML
WDDX	JavaScript

CFML クエリーオブジェクトを WDDX に変換する cfwddx タグは、一般的に次のようになります。

```
<cfwddx action="cfml2wddx" input="#MyQueryObject#" output="WddxTextVariable">
```

この例の MyQueryObject は、クエリーオブジェクト変数の名前です。WddxTextVariable は、変換された WDDX XML を保管する変数の名前です。

注意：cfwddx タグの詳細については、『CFML リファレンス』を参照してください。

WDDX データの検証

cfwddx タグには、WDDX を CFML または JavaScript に変換するときに使用できる Validate 属性があります。この属性を True に設定すると、WDDX データのシリアル化が解除される前に、WDDX DTD を使用して WDDX データが検証されます。WDDX が無効な場合は、エラーが発生します。デフォルトでは、WDDX データを ColdFusion または JavaScript データに変換するときに検証は行われません。

IsWDDX 関数は、変数が有効な WDDX データパケットである場合に True を返し、それ以外の場合は False を返します。この関数を使用すると、WDDX パケットを別の形式に変換する前に検証できます。たとえば、cfwddx の validate 属性の代わりに使用すれば、エラー処理ロジックではなく条件ロジックで無効な WDDX を処理できます。また、この関数を使用して、JavaScript でシリアル化を解除するデータをブラウザであらかじめ検証することもできます。

JavaScript オブジェクトの使用

ColdFusion には、JavaScript でデータを WDDX に変換するときに使用できる 2 つの JavaScript オブジェクト WddxSerializer object および WddxRecordset object があります。これらは "<Web のルートディレクトリ >/cfide/scripts/wddx.js" ファイルで定義されています。

『CFML リファレンス』では、これらのオブジェクトとそのメソッドについて詳細に説明しています。1058 ページの「ブラウザからサーバーへのデータ転送」の例では、これらのオブジェクトを使用して JavaScript を WDDX にシリアル化する方法を示しています。

CFML データから JavaScript オブジェクトへの変換

次の例では、サーバー上の ColdFusion ページで取得した cfquery レコードセットを、ブラウザで処理する JavaScript オブジェクトに変換します。

このアプリケーションは、次の 4 つのセクションから構成されています。

- データクエリーの実行
- WDDX JavaScript ユーティリティクラスのインクルード

- 変換関数の呼び出し
- HTML でのオブジェクトデータの記述

次の例では、ColdFusion に付属する cfdocexamples データソースを使用します。

```
<!--- Create a simple query --->
<cfquery name = "q" datasource ="cfdocexamples">
    SELECT Message_Id, Thread_id, Username, Posted
    FROM messages
</cfquery>

<!--- Load the wddx.js file, which includes the dump function --->
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>

<script>
    // Use WDDX to move from CFML data to JavaScript
    <cfwddx action="cfml2js" input="#q#" topLevelVariable="qj">

    // Dump the recordset to show that all the data has reached
    // the client successfully.
    document.write(qj.dump(true));
</script>
```

注意：cfwddx Action="cfml2js" の動作を確認するには、このコードを Web ルートディレクトリ内に (たとえば、"wwwroot/myapps/wddxjavascript.cfm" として) 保存し、ブラウザでこのページを実行して、ブラウザの [ソースの表示] コマンドを選択します。

ブラウザからサーバーへのデータ転送

次の例では、フォームフィールドデータをシリアル化し、それをサーバーに転送し、シリアル化を解除して、そのデータを表示します。簡単にするために、データ量は少なくしています。複雑な JavaScript データコレクションを生成するアプリケーションでは、この基本的な方法を効果的に応用できます。この例では、WddxSerializer JavaScript オブジェクトを使用してデータをシリアル化し、cfwddx タグを使用してデータのシリアル化を解除しています。

この例の実行

1. このファイルを Web ルートディレクトリの下、たとえば "wwwroot/myapps/wddxserializedeserialze.cfm" に保存します。
2. ブラウザで http://localhost/myapps/wddxserializedeserialze.cfm を表示します。
3. フォームフィールドに氏名を入力します。
4. 「次へ」をクリックします。

   入力した氏名は、[今までに追加された名前] ボックスに表示されます。
5. 複数の名前を追加するには、手順 3 ～ 4 を繰り返します。
6. [シリアル化] をクリックし、データをシリアル化します。

   WDDX パケットが生成され、[WDDX パケットの表示] ボックスに表示されます。この手順はテスト専用で、実際のアプリケーションには適していません。実際のアプリケーションでは、シリアル化を自動的に行うようにします。
7. 送信ボタンをクリックしてデータを送信します。

   WDDX パケットがサーバーサイドの処理コードに転送され、WDDX パケットのシリアル化が解除され、その情報が表示されます。

```
<!--- load the wddx.js file --->
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>

<!--- Data binding code --->
<script>

    // Generic serialization to a form field
    function serializeData(data, formField) {
        wddxSerializer = new WddxSerializer();
        wddxPacket = wddxSerializer.serialize(data);
        if (wddxPacket != null) {
            formField.value = wddxPacket;
        }
        else {
            alert("Couldn't serialize data");
        }
    }
    // Person info recordset with columns firstName and lastName
    // Make sure the case of field names is preserved
    var personInfo = new WddxRecordset(new Array("firstName", "lastName"), true);

    // Add next record to end of personInfo recordset
    function doNext() {
        // Extract data
        var firstName = document.personForm.firstName.value;
        var lastName = document.personForm.lastName.value;

        // Add names to recordset
        nRows = personInfo.getRowCount();
        personInfo.firstName[nRows] = firstName;
        personInfo.lastName[nRows] = lastName;

        // Clear input fields
        document.personForm.firstName.value = "";
        document.personForm.lastName.value = "";

        // Show added names on list
        // This gets a little tricky because of browser differences
        var newName = firstName + " " + lastName;
        if (navigator.appVersion.indexOf("MSIE") == -1) {
            document.personForm.names[length] =
                new Option(newName, "", false, false);
        }
        else {
            // IE version
            var entry = document.createElement("OPTION");
            entry.text = newName;
            document.personForm.names.add(entry);
        }
    }
</script>

<!--- Data collection form --->
<form action="#cgi.script_name#" method="Post" name="personForm">

    <!--- Input fields --->
    Personal information<br>
    First name: <input type=text name=firstName><br>
    Last name: <input type=text name=lastName><br>
    <br>

    <!--- Navigation & submission bar --->
```

```
    <input type="button" value="Next" onclick="doNext()">
    <input type="button" value="Serialize"
    onclick="serializeData(personInfo, document.personForm.wddxPacket)">
    <input type="submit" value="Submit">
    <br><br>
    Names added so far:<br>
    <select name="names" size="5">
    </select>
    <br>

    <!--- The WDDX packet is stored here.--->
    <!--- In a real application this text area would be a hidden
            input field. --->
    <br>
    WDDX packet display:<br>
    <textarea name="wddxPacket" rows="10" cols="80" wrap="Virtual">
    </textarea>

</form>


<!--- Server-side processing --->
<hr>
<b>Server-side processing</b><br>
<br>
<cfif isdefined("form.wddxPacket")>
    <cfif form.wddxPacket neq "">

        <!--- Deserialize the WDDX data --->
        <cfwddx action="wddx2cfml" input=#form.wddxPacket#
        output="personInfo">

        <!--- Display the query --->
        The submitted personal information is:<br>
        <cfoutput query=personInfo>
            Person #CurrentRow#: #firstName# #lastName#<br>
        </cfoutput>
    <cfelse>
        The client did not send a well-formed WDDX data packet!
    </cfif>
<cfelse>
    No WDDX data to process at this time.
</cfif>
```

文字列への複合データの格納

次に示す簡単な例では、WDDX を使用して、複合データ (配列が保持されているデータ構造体) をクライアント変数に文字列として保存します。シリアル化の前とシリアル化解除の後には、cfdump タグを使用して構造体の内容を表示しています。cfoutput タグ内では、HTMLEditFormat 関数を使用してクライアント変数の内容を表示しています。HTMLEditFormat 関数は、ブラウザが変数内の XML タグを解釈しないようにするために必要です。

```
<!--- Enable client state management --->
<cfapplication name="relatives" clientmanagement="Yes">

<!--- Build a complex data structure --->
<cfscript>
    relatives = structNew();
    relatives.father = "Bob";
    relatives.mother = "Mary";
    relatives.sisters = arrayNew(1);
    arrayAppend(relatives.sisters, "Joan");
    relatives.brothers = arrayNew(1);
    arrayAppend(relatives.brothers, "Tom");
    arrayAppend(relatives.brothers, "Jesse");
</cfscript>

A dump of the original relatives structure:<br>
<br>
<cfdump var="#relatives#"><br>
<br>

<!--- Convert data structure to string form and save it in the client scope --->
<cfwddx action="cfml2wddx" input="#relatives#" output="Client.wddxRelatives">

The contents of the Client.wddxRelatives variable:<br>
<cfoutput>#HtmlEditFormat(Client.wddxRelatives)#</cfoutput><br>

<!--- Now read the data from client scope into a structure --->
<cfwddx action="wddx2cfml" input="#Client.wddxRelatives#" output="sameRelatives">
<br>
A dump of the sameRelatives structure generated from client.wddxRelatives<br>
<cfdump var="#sameRelatives#">
```

Web サービスの使用

Web サービスを使用すると、リモートアプリケーションの機能をインターネットを介して公開したり利用したりできます。Web サービスを利用するには、リモート機能にアクセスしてアプリケーションタスクを実行します。Web サービスを公開するには、リモートユーザーがアプリケーション機能にアクセスできるようにして、ユーザーが開発するアプリケーションに Web サービスを組み込めるようにします。

Web サービス

リモートコンピュータのコンテンツにアクセスできる機能は、当初から変わらないインターネットの基本機能です。このコンテンツには、HTML で作成されたドキュメントなどのスタティックなコンテンツもあれば、ColdFusion ページや CGI スクリプトによって返されるダイナミックなコンテンツもあります。

Web サービスを利用すれば、他のユーザーが作成してリモートコンピュータで公開しているアプリケーション機能にアクセスできます。ユーザーは、リモートアプリケーションにリクエストを行って、アクションを実行させることができます。

たとえば、株価情報のリクエスト、翻訳するテキストの転送、製品カタログの情報のリクエストなどを行うことができます。Web サービスの利点は、作成済みのアプリケーションロジックを利用することで再作成の手間を省き、アプリケーションを短時間で構築できることにあります。

ColdFusion アプリケーションでリモート Web サービスを参照することを、Web サービスの利用と呼びます。Web サービスは、実装テクノロジーに依存しない標準インターフェイスに基づいて構築されているので、ColdFusion アプリケーションだけでなく、.NET アプリケーションや Java アプリケーションからも利用できます。

Web サービスを作成して、他のユーザーがリモートアクセスで利用できるようにすることを、Web サービスの公開と呼びます。Web サービスを利用するアプリケーションは、ColdFusion や、Web サービス標準を認識できる任意のアプリケーションで実装できます。

Web サービスへのアクセス

最も単純な場合、Web サービスへのアクセス方法は、関数呼び出しと同様です。コンピュータ上のライブラリを参照して関数を呼び出すのと同様に、インターネット上のリモート機能を参照します。

Web サービスの特徴の 1 つは自己記述型であることです。Web サービスを公開するユーザーは、Web サービス記述言語 (WSDL) ファイルを使用して、Web サービスの API に関する記述も公開します。

WSDL ファイルは XML 形式のドキュメントで、Web サービスに関する次の情報が含まれています。

- Web サービスで呼び出せるオペレーション
- 各オペレーションに渡す入力パラメータ
- オペレーションから返される値

Web サービスを利用するには、通常、次の 2 つの手順を行います。

1 Web サービスの WSDL ファイルを解析して、インターフェイスを調べます。

Web サービスの WSDL ファイルは、インターネットで公開されています。Web サービスを利用するには、サービスが定義されている WSDL ファイルの URL を知っている必要があります。たとえば、TemperatureService Web サービスの WSDL ファイルは、次の URL からアクセスできます。

www.xmethods.net/sd/2001/TemperatureService.wsdl

WSDL シンタックスの概要については、1063 ページの「WSDL ファイルの操作」を参照してください。

2 Web サービスにリクエストを行います。

次の例では、Temperature Web サービスのオペレーションを実行して、郵便番号 55987 の地域の気温を取得します。

```
<cfinvoke
     webservice="http://www.xmethods.net/sd/2001/TemperatureService.wsdl"
     method="getTemp"
     returnvariable="aTemp">
       <cfinvokeargument name="zipcode" value="55987"/>
</cfinvoke>
<cfoutput>The temperature at ZIP code 55987 is #aTemp#</cfoutput>
```

Web サービスの利用方法の詳細については、1065 ページの「Web サービスの利用」を参照してください。

Web サービスの基本概念

Web サービスの仕組みを完全に理解するには、Web サービスプロバイダの基本アーキテクチャを理解する必要があります。

注意：詳細については、Web サービスに関する書籍を参照してください。

Web サービスのプラットフォームは、次の 3 つの主要コンポーネントで構成されます。

- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)
- UDDI (Universal Description, Discovery, and Integration)

SOAP による Web サービスのサポート

SOAP は、インターネットを介して Web サービスのリクエストやレスポンスを送受信するための標準の XML 構造を規定したものです。通常は HTTP を使用して SOAP メッセージを送信しますが、SMTP などのプロトコルを使用して送信することもできます。ColdFusion では、Apache Axis SOAP エンジンを統合して Web サービスをサポートしています。

ColdFusion の Web サービスエンジンには、Web サービスの WSDL ファイルの生成などの、Web サービスをサポートする基本機能が搭載されています。ColdFusion では、SOAP に精通していなくても、また SOAP オペレーションを実行しなくても、Web サービスの利用および公開を行うことができます。

SOAP に関する補足情報については、www.w3.org/TR/SOAP/ にある W3C の SOAP 1.1 ノートを参照してください。

WSDL による Web サービスの記述

WSDL ドキュメントは、Web サービスの目的、場所、およびアクセス方法を記述した XML ファイルです。呼び出せるオペレーションと、それに関連付けられたデータ型が記述されています。

ColdFusion では、Web サービスから WSDL ドキュメントを生成して、Web 上でクライアントに公開することができます。詳細については、1063 ページの「WSDL ファイルの操作」を参照してください。

UDDI による Web サービスの検索

Web サービスの利用者は、利用できる Web サービスを検索したいことがあります。Web サービスの公開者は、その Web サービスの情報を他のユーザーが検索できるようにしたいことがあります。UDDI (Universal Description, Discovery, and Integration) を使用すれば、特定の機能を提供する Web サービスをダイナミックに検索できます。サービスプロバイダを検索するには、UDDI クエリーを使用します。返される UDDI レスポンスには、連絡先、業務分野、技術的詳細などの情報や、Web サービスを呼び出す方法が含まれています。

ColdFusion では、UDDI は直接サポートされていませんが、IBM UDDI Business Registry などのパブリック UDDI レジストリを使用して、Web サービスを手動で登録または検索できます。

UDDI の補足情報については、www.uddi.org/about.htm を参照してください。

WSDL ファイルの操作

WSDL ファイルには、Web サービスへのインターフェイスが定義されています。Web サービスを利用するには、そのサービスの WSDL ファイルにアクセスして、サービスに関する情報を調べます。アプリケーションロジックを Web サービスとして公開する場合は、その WSDL ファイルを作成します。

WSDL は、W3C でサポートされているドラフト標準です。WSDL の仕様は、www.w3.org/TR/wsdl で参照できます。

WSDL ファイルの作成

Web サービスを公開するには、サービスの機能を実装した後に、サービスを定義する WSDL ファイルを作成します。ColdFusion では、コンポーネントを使用して Web サービスを作成します。Web サービスで使用されているコンポーネントの WSDL ファイルは自動的に生成されます。Web サービスの作成方法の詳細については、1071 ページの「Web サービスの公開」を参照してください。

コンポーネントの詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

Dreamweaver による Web サービスへのアクセス

Dreamweaver の [コンポーネント] タブを使用して、オペレーション名、パラメータ名、パラメータのデータ型などの Web サービスの詳細を表示することができます。

Dreamweaver の [コンポーネント] タブを開いて、Web サービスを追加する

1 [ウィンドウ]-[コンポーネント] を選択するか、あるいは Ctrl+F7 キーを使用して [コンポーネント] パネルを開きます。

2 [コンポーネント] パネルの左上にあるドロップダウンリストから Web サービスを選択します。

3 プラス (+) ボタンをクリックします。

[WSDL を使用して追加] ダイアログボックスが表示されます。

4 WSDL ファイルの URL を指定します。

Dreamweaver で Web サービスを定義したら、ページに Web サービスをドラッグして、その Web サービスを cfinvoke タグで呼び出すことができます。

Dreamweaver の使用方法の詳細については、Dreamweaver のオンラインヘルプシステムを参照してください。

注意：Macintosh で Dreamweaver を実行している場合は、[Web サービス] オプションを利用することはできません。ただし、コードを手入力して Web サービスを使用することは可能です。

WSDL ファイルの理解

WSDL ファイルを理解するには、ある程度の慣れが必要です。WSDL ファイルは、ブラウザで表示したり、WSDL ファイルを読みやすい形式で表示する機能を備えた Dreamweaver などのツールで開くこともできます。

次の例は、TemperatureService Web サービスの WSDL ファイルです。

```
<?xml version="1.0"?>
<definitions name="TemperatureService"
targetNamespace="http://www.xmethods.net/sd/TemperatureService.wsdl"xmlns:tns="http://www.xmethods.net/sd
/TemperatureService.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">
    <message name="getTempRequest">
        <part name="zipcode" type="xsd:string"/>
    </message>
    <message name="getTempResponse">
        <part name="return" type="xsd:float"/>
    </message>
    <portType name="TemperaturePortType">
        <operation name="getTemp">
            <input message="tns:getTempRequest"/>
            <output message="tns:getTempResponse"/>
        </operation>
    </portType>
    <binding name="TemperatureBinding" type="tns:TemperaturePortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getTemp">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="encoded" namespace="urn:xmethods-Temperature"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body use="encoded" namespace="urn:xmethods-Temperature"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
        </operation>
    </binding>
    <service name="TemperatureService">
        <documentation>Returns current temperature in a given U.S. zipcode</documentation>
        <port name="TemperaturePort" binding="tns:TemperatureBinding">
            <soap:address location="http://services.xmethods.net:80/soap/servlet/rpcrouter"/>
        </port>
    </service>
</definitions>
```

次の表に、WSDL ファイルの主要なコンポーネントを示します。

コンポーネント	定義
definitions	WSDL ファイルのルート要素。この領域には、複数の Web サービスで名前が競合するのを防ぐために使用する名前空間が含まれています。
types	(例には示されていません) サービスのメッセージで使用されるデータ型が定義されています。
message	Web サービスのオペレーションによって転送されるデータが定義されています。通常は、入力パラメータおよび戻り値の名前とデータ型です。
port type	Web サービスで提供されるオペレーションが定義されています。
operation	リモートで呼び出せるオペレーションが定義されています。
input	前に定義されたメッセージを使用して、オペレーションへの入力パラメータが指定されています。
output	前に定義したメッセージを使用して、オペレーションからの戻り値が指定されています。
fault	(例には示されていません) オペレーションから返されるエラーメッセージが指定されています。オプションです。
binding	SOAP、HTTP GET、POST、MIME などの Web サービスへのアクセスに使用するプロトコルが指定されています。
service	関連するオペレーションのグループが定義されています。
port	オペレーションと、それに関連する入出力が定義されています。

この WSDL ファイルの内容の詳細については、1065 ページの「Web サービスの利用」を参照してください。

Web サービスの利用

ColdFusion には、Web サービスを利用するためのさまざまな方法が用意されています。どの方法を選ぶかは、ColdFusion のプログラミングスタイルやアプリケーションによって異なります。

それらの方法について次の表で説明します。

メソッド	CFML の演算子	説明
CFScript	()	CFScript ブロック内から Web サービスを利用します。
CFML タグ		CFML のコードブロック内から Web サービスを利用します。
CFML タグ		CFML のコードブロック内から Web サービスを利用します。

どの方法を選んでも、使用される基本テクノロジーは同じであり、パフォーマンスは変わりません。

例について

ここに示す例では、XMethods の TemperatureService Web サービスを利用しています。この Web サービスは、指定した郵便番号の地域の気温を返します。この Web サービスの WSDL ファイルは、1064 ページの「WSDL ファイルの理解」に掲載されています。

TemperatureService Web サービスは、郵便番号を表す文字列を入力パラメータとして取り、指定された郵便番号の地域の気温を表す浮動小数点数を返します。

Web サービスのパラメータの指定

WSDL ファイルの message 要素および operation 要素には、Web サービスのオペレーションを定義するサブ要素と、各オペレーションの入出力パラメータ (各パラメータのデータ型を含む) が含まれています。次の例に、TemperatureService Web サービスの WSDL ファイルの一部を示します。

```
<message name="getTempRequest">
    <part name="zipcode" type="xsd:string"/>
</message>
<message name="getTempResponse">
    <part name="return" type="xsd:float"/>
</message>
<portType name="TemperaturePortType">
    <operation name="getTemp">
        <input message="tns:getTempRequest"/>
        <output message="tns:getTempResponse"/>
    </operation>
</portType>
```

ここに示した例で使用しているオペレーション名は getTemp です。このオペレーションは、getTempRequest 型のメッセージとして定義される 1 つの入力パラメータを取ります。

getTempRequest という名前の message 要素には、1 つの文字列パラメータ zipcode が含まれています。getTemp オペレーションを呼び出すときには、このパラメータを入力として渡します。

Web サービスからの戻り値の処理

多くの場合、Web サービスのオペレーションはアプリケーションに情報を返します。返される情報の名前とデータ型を調べるには、WSDL ファイルの message 要素および operation 要素のサブ要素を調べます。

次の例に、TemperatureService Web サービスの WSDL ファイルの一部を示します。

```
<message name="getTempRequest">
    <part name="zipcode" type="xsd:string"/>
</message>
<message name="getTempResponse">
    <part name="return" type="xsd:float"/>
</message>
<portType name="TemperaturePortType">
    <operation name="getTemp">
        <input message="tns:getTempRequest"/>
        <output message="tns:getTempResponse"/>
    </operation>
</portType>
```

オペレーション getTemp は、getTempResponse 型のメッセージを返します。この WSDL ファイルの message ステートメントでは、getTempResponse メッセージに return という文字列パラメータがあることが定義されています。

cfinvoke による Web サービスの利用

cfinvoke タグを使用する方法では、1 つのタグで WSDL ファイルを参照し、Web サービスのオペレーションを呼び出します。

cfinvoke タグには、WSDL ファイルの URL、呼び出すメソッド、戻り変数、および入力パラメータを指定する属性が含まれています。シンタックスの詳細については、『CFML リファレンス』を参照してください。

注意：Web サービスにパラメータを渡すには、cfinvokeargument タグを使用するか、cfinvoke タグ自体でパラメータ名を指定します。詳細については、191 ページの「cfinvoke タグでのメソッドパラメータの受け渡し」を参照してください。

cfinvoke による Web サービスへのアクセス

1 次の内容の ColdFusion ページを作成します。

```
<cfinvoke
webservice="http://www.xmethods.net/sd/2001/TemperatureService.wsdl"
method="getTemp"
returnvariable="aTemp">
<cfinvokeargument name="zipcode" value="55987"/>
</cfinvoke>
<cfoutput>The temperature at ZIP code 55987 is #aTemp#</cfoutput>
```

2 このページに "wscfc.cfm" という名前を付けて、Web のルートディレクトリに保存します。

3 このページをブラウザで表示します。

omit 属性を "yes" に設定することにより、パラメータを省略することができます。WSDL で、引数が nullable であると指定されている場合、関連付けられている引数は null に設定されます。WSDL で minoccurs=0 と指定されている場合、その引数は WSDL から省略されます。ただし CFC Web サービスでは、すべての引数に対して required="true" と指定する必要があります。

属性コレクションを使用して、パラメータを渡すこともできます。属性コレクションは 1 つの構造体で、各キーにパラメータ名を、キーの値にパラメータ値を指定したものです。次の例では、属性コレクションを使用して Web サービスを呼び出しています。

```
<cfscript>
    stArgs = structNew();
    stArgs.zipcode = "55987";
</cfscript>

<cfinvoke
webservice="http://www.xmethods.net/sd/2001/TemperatureService.wsdl"
method="getTemp"
argumentcollection="#stArgs#"
returnvariable="aTemp">
<cfoutput>The temperature at ZIP code 55987 is #aTemp#</cfoutput>
```

この例では、CFScript ブロックで構造体を作成していますが、任意の ColdFusion メソッドで構造体を作成できます。

CFScript による Web サービスの利用

次の例では、CFScript を使用して Web サービスを利用します。CFScript では、CreateObject 関数を使用して Web サービスに接続します。その後、サービスに対してリクエストを行います。CreateObject のシンタックスの詳細については、『CFML リファレンス』を参照してください。

Web サービスオブジェクトを作成したら、次のようにドット表記法を使用して、Web サービスのオペレーションを呼び出すことができます。

```
webServiceName.operationName(inputVal1, inputVal2, ... );
```

Web サービスからの戻り値は、次の例のように、変数に書き込むことによって処理できます。

```
resultVar = webServiceName.operationName(inputVal1, inputVal2, ... );
```

または、次の例にように、WriteOutput 関数などに戻り値を直接渡すこともできます。

```
writeoutput(webServiceName.operationName(inputVal1, inputVal2, ...) );
```

CFScript からの Web サービスへのアクセス

1 次の内容の ColdFusion ページを作成します。

```
<cfscript>
ws = CreateObject("webservice",
"http://www.xmethods.net/sd/2001/TemperatureService.wsdl");
xlatstring = ws.getTemp("55987");
writeoutput(xlatstring);
</cfscript>
```

2 このページに "wscfscript.cfm" という名前を付けて、Web のルートディレクトリに保存します。

3 このページをブラウザで表示します。

名前付きパラメータを使用して、Web サービスに情報を渡すこともできます。次の例では、前の例と同じオペレーションを実行しますが、名前付きパラメータを使用して Web サービスをリクエストしています。

```
<cfscript>
ws = CreateObject("webservice",
"http://www.xmethods.net/sd/2001/TemperatureService.wsdl");
xlatstring = ws.getTemp(zipcode = "55987");
writeoutput("The temperature at 55987 is " & xlatstring);
</cfscript>
```

ColdFusion によって生成されていない Web サービスの利用

ColdFusion 以外のテクノロジーを使用して実装されている Web サービスを利用する場合は、次のいずれかのオプションが Web サービスで設定されている必要があります。

- SOAP バインドスタイルが rpc で、encodingStyle が encoding である
- SOAP バインドスタイルが document で、encodingStyle が literal である

次の例に、TemperatureService Web サービスの WSDL ファイルの一部を示します。

```
<binding name="TemperatureBinding" type="tns:TemperaturePortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getTemp">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="encoded" namespace="urn:xmethods-Temperature"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
        <output>
            <soap:body use="encoded" namespace="urn:xmethods-Temperature"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
    </operation>
</binding>
```

TemperatureService Web サービスの WSDL ファイルでは、バインディングスタイルが rpc で、encodingStyle が encoding なので、ColdFusion と互換性があります。

Flash クライアントからの Web サービスの呼び出し

Flash Remoting サービスを使用すれば、Flash クライアントから ColdFusion ページを呼び出すことはできますが、Web サービスを直接呼び出すことはできません。Web サービスを Flash クライアントから呼び出すには、Flash Remoting を使用して、Web サービスを呼び出す ColdFusion コンポーネントを呼び出します。Flash クライアントは入力パラメータをコンポーネントに渡し、コンポーネントは Web サービスから返されたデータを Flash クライアントに返すことができます。

詳細については、595 ページの「Flash Remoting サービスの使用」を参照してください。

Web サービス利用時のエラー検出

Web サービスの処理中に SOAP 障害などのエラーが発生する可能性がありますが、それらのエラーはアプリケーションで検出することができます。アプリケーションで検出されなかったエラーは、ブラウザに返されます。

エラーを検出するには、次の例のように ColdFusion cfcatch タグで application エラータイプを指定します。

```
<cftry>
    Place your application code here ...
    <cfcatch type="application">
        <!--- Add exception processing code here ... --->
    </cfcatch>
    ...
    <cfcatch type="Any">
        <!--- Add exception processing code appropriate for all other
    exceptions here ... --->
    </cfcatch>
</cftry>
```

エラー処理の詳細については、275 ページの「エラー処理」を参照してください。

inout パラメータおよび out パラメータの処理

Web サービスによっては、inout パラメータや out パラメータが定義されている場合があります。**out** パラメータは、戻り値のプレースホルダーを Web サービスに渡すために使用します。Web サービスは、結果を out パラメータに書き込むことによってその結果を返します。**inout** パラメータは、Web サービスに値を渡すために使用するとともに、Web サービスがパラメータ値を上書きして結果を返すためにも使用されます。

次の例は、文字列を含む inout パラメータを入力として取り、その文字列に結果を書き込む Web サービスを示しています。

```
<cfset S="foo">
<cfscript>
    ws=createobject("webservice", "URLtoWSDL")
    ws.modifyString("S");
</cfscript>
<cfoutput>#S#</cfoutput>
```

この Web サービスの入力には S の値を渡しますが、これは inout パラメータとして渡すので、シャープ記号 (#) では囲みません。

注意：ColdFusion では、Web サービスを利用するための inout および out パラメータはサポートされていますが、公開する Web サービスを作成するための inout および out パラメータはサポートされていません。

ColdFusion Administrator での Web サービスの設定

Web サービスは ColdFusion Administrator に登録できるので、Web サービスを参照するときに WSDL の URL を個別に指定する必要はありません。

注意：Web サービスを初めて参照すると、その Web サービスが ColdFusion Administrator に自動的に登録されます。

たとえば次のコードでは、TemperatureService の WSDL ファイルの URL を参照しています。

```
<cfscript>
ws = CreateObject("webservice",
"http://www.xmethods.net/sd/2001/TemperatureService.wsdl");
xlatstring = ws.getTemp("55987");
writeoutput(xlatstring);
</cfscript>
```

wsTemp という名前を使用して TemperatureService Web サービスを Administrator に登録すれば、次のようにして Web サービスを参照できます。

```
<cfscript>
     ws = CreateObject("webservice", "wsTemp");
     xlatstring = ws.getTemp("55987");
     writeoutput("wsTemp: " & xlatstring);
</cfscript>
```

Web サービスを Administrator に登録すれば、コードを短くできるだけでなく、Web サービスの URL が変わった場合でもコードを変更しなくて済むようになります。たとえば、TemperatureService Web サービスが別の場所に移動しても、アプリケーションコードを変更せずに、Administrator の設定を更新するだけで済みます。

詳細については、ColdFusion Administrator のオンラインヘルプを参照してください。

ColdFusion のデータ型と WSDL のデータ型の間のデータ変換

WSDL ファイルでは、オペレーションの入力パラメータと戻りパラメータが、データ型も含めて定義されています。たとえば、TemperatureService Web サービスでは、次のように入力パラメータと戻りパラメータが定義されています。

```
<message name="getTempRequest">
    <part name="zipcode" type="xsd:string"/>
</message>
<message name="getTempResponse">
    <part name="return" type="xsd:float"/>
</message>
```

Web サービスを利用するときは、WSDL のデータ型がどのように ColdFusion のデータ型に変換されるかを理解しておく必要があります。次の表に、この変換を示します。

ColdFusion のデータ型	WSDL のデータ型
数値	SOAP-ENC:double
ブール値	SOAP-ENC:boolean
文字列	SOAP-ENC:string
配列	SOAP-ENC:Array
バイナリ	xsd:base64Binary
数値	xsd:float
文字列	xsd:enumeration
日付	xsd:dateTime
void (オペレーションから何も返されない)	
構造体	複合型
クエリー	tns1:QueryBean (CFC によって返されます)

WSDL のデータ型のうち、文字列や数値などの一般的なデータ型の多くは、ColdFusion のデータ型に直接マッピングします。WSDL の複合データ型の場合、マッピングはやや複雑になります。多くの場合、WSDL の複合データ型は ColdFusion の構造体にマッピングします。複合データ型の処理の詳細については、1081 ページの「複合データ型の処理」を参照してください。

ColdFusion Web サービスの利用

ColdFusion で作成した Web サービスをアプリケーションで利用することができます。ColdFusion Web サービスを利用するときは、データマッピングの処理が ColdFusion によって自動的に行われるので、入力パラメータや戻り値に対して特別な処理を行う必要はありません。

たとえば、戻り値としてクエリーを返すか、入力値としてクエリーを取る Web サービスが ColdFusion で公開されている場合、そのサービスの WSDL ファイルでは、そのデータ型が QueryBean として定義されています。しかし、この Web サービスを利用する ColdFusion アプリケーションでは、関数への入力として ColdFusion クエリーを渡したり、返された QueryBean を ColdFusion クエリーオブジェクトに書き込んだりすることができます。

注意：ColdFusion のデータ型から WSDL のデータ型へのマッピングの詳細なリストについては、1070 ページの「ColdFusion のデータ型と WSDL のデータ型の間のデータ変換」を参照してください。

次の例は、入力値としてクエリーを取り、入力されたクエリーをそのまま返す ColdFusion コンポーネントです。

```
<cfcomponent>
    <cffunction name='echoQuery' returnType='query' access='remote'>
        <cfargument name='input' type='query'>
        <cfreturn #arguments.input#>
    </cffunction>
</cfcomponent>
```

"echotypes.cfc" コンポーネントの WSDL ファイルでは、次のように、関数の入力および出力のタイプとして QueryBean が定義されています。

```
<wsdl:message name="echoQueryResponse">
     <wsdl:part name="echoQueryReturn" type="tns1:QueryBean"/>
</wsdl:message>
<wsdl:message name="echoQueryRequest">
     <wsdl:part name="input" type="tns1:QueryBean"/>
</wsdl:message>
```

WSDL の表示方法の詳細については、1073 ページの「WSDL ファイルの作成」を参照してください。

ColdFusion データ型へのマッピングは ColdFusion によって自動的に処理されるので、次のようにこの Web サービスを呼び出すことができます。

```
<head>
<title>Passing queries to web services</title>
</head>
<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary
    FROM Employee
</cfquery>

<cfinvoke
    webservice = "http://localhost/echotypes.cfc?wsdl"
    method = "echoQuery"
    input="#GetEmployees#"
    returnVariable = "returnedQuery">

<cfoutput>
    Is returned result a query? #isQuery(returnedQuery)# <br><br>
</cfoutput>

<cfoutput query="returnedQuery">
    #FirstName#, #LastName#, #Salary#<br>
</cfoutput>
</body>
```

Web サービスの公開

Web サービスを公開してリモートアプリケーションで利用できるようにするには、ColdFusion コンポーネントを使用して Web サービスを作成します。コンポーネントの詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

Web サービスのコンポーネントの作成

ColdFusion コンポーネント (CFC) は、アプリケーションの機能をカプセル化し、クライアントがその機能にアクセスするための標準インターフェイスを提供します。通常、1 つのコンポーネントには、cffunction タグによって定義された関数が含まれています。

たとえば、次のコンポーネントには 1 つの関数が含まれています。

```
<cfcomponent>
    <cffunction name="echoString" returnType="string" output="no">
        <cfargument name="input" type="string">
        <cfreturn #arguments.input#>
    </cffunction>
</cfcomponent>
```

echoString という関数は、渡された文字列をそのまま返します。関数を Web サービスとして公開するには、次のように関数定義を変更して access 属性を追加し、"remote" を指定します。

```
<cffunction name="echoString" returnType="string" output="no" access="remote">
```

関数をリモートとして定義すると、ColdFusion によってその関数が WSDL ファイルに含められます。Web サービスとしてアクセスできるのは、リモートと指定されている関数のみです。

公開する Web サービスを作成するときは、次のことを行う必要があります。

1. cffunction タグの access 属性の値を remote に指定する必要があります。
2. cffunction タグには、戻り値のタイプを指定する returnType 属性を含める必要があります。
3. cffunction タグの output 属性を No に設定する必要があります。すべての出力は XML に変換されてから Web サービスの利用側に返されるからです。
4. cfargument タグの属性設定 required="false" は無視されます。すべてのパラメータは必須と見なされます。

関数の引数と戻り値のデータ型の指定

cffunction タグでは、1 つの戻り値と、関数に渡す入力パラメータ (複数可能) を定義できます。この定義を行うときは、戻り値および入力パラメータのデータ型を指定できます。

次の例に示すコンポーネントでは、文字列型の戻り値、文字列型の 1 つの入力パラメータ、数値型の 1 つの入力パラメータを持つ関数が定義されています。

```
<cfcomponent>
    <cffunction name="trimString" returnType="string" output="no">
        <cfargument name="inString" type="string">
        <cfargument name="trimLength" type="numeric">
    </cffunction>
</cfcomponent>
```

コンポーネントを公開して Web サービスとしてアクセスできるようにする過程で、コンポーネントを定義する WSDL ファイルが ColdFusion によって生成されます。WSDL ファイルには、ColdFusion データ型と WSDL データ型のマッピングの定義が含まれています。次の表に、このマッピングを示します。

ColdFusion のデータ型	公開される WSDL のデータ型
数値	SOAP-ENC:double
ブール値	SOAP-ENC:boolean
文字列	SOAP-ENC:string
配列	SOAP-ENC:Array
バイナリ	xsd:base64Binary

ColdFusion のデータ型	公開される WSDL のデータ型
日付	xsd:dateTime
guid	SOAP-ENC:string
uuid	SOAP-ENC:string
void (オペレーションから何も返されない)	
構造体	Map
クエリー	QueryBean
any	複合型
コンポーネント定義	複合型

ほとんどの場合、ColdFusion Web サービスの利用側では、この表に従って ColdFusion のデータ型を WSDL のデータ型にマッピングすれば、簡単にコンポーネント関数に対してデータを渡したり、結果を受け取ったりすることができます。

注意：document-literal の Web サービスでは、SOAP-ENC データ型ではなく、XML スキーマデータ型が使用されます。詳細については、1077 ページの「document-literal スタイルの Web サービスの公開」を参照してください。

ColdFusion の構造体およびクエリーについては、正しい型にマッピングするためのデータ処理が必要になることがあります。詳細については、1084 ページの「複合データ型を使用する Web サービスの公開」を参照してください。

ColdFusion コンポーネントのデータ型を、別のコンポーネント定義に基づいて定義することもできます。コンポーネントを使用してデータ型を指定する方法の詳細については、1076 ページの「ColdFusion コンポーネントによる Web サービスのデータ型の定義」を参照してください。

WSDL ファイルの作成

ColdFusion では、Web サービスとして参照されるコンポーネントの WSDL ファイルは自動的に作成されます。たとえば、Web のルートディレクトリに "echo.cfc" というコンポーネントがある場合は、次のようにコンポーネントをリクエストすることで、対応する WSDL ファイルを表示できます。

```
http://localhost/echo.cfc?wsdl
```

cfcomponent タグには、ColdFusion で生成される WSDL を制御するためのオプションの属性が用意されています。それらの属性を使用して、次の例のように、意味のある WSDL 属性名を作成することができます。

```
<cfcomponent style="document"
namespace = "http://www.mycompany.com/"
    serviceportname = "RestrictedEmpInfo"
    porttypename = "RestrictedEmpInfo"
    bindingname = "myns:RestrictedEmpInfo"
    displayname = "RestrictedEmpInfo"
    hint = "RestrictedEmpInfo">
```

WSDL を完全に制御したい上級ユーザーは、cfcomponent の wsdlFile 属性を指定して、定義済みの **WSDL** ファイルを使用できます。

次の例では、Web サービスとして呼び出せる ColdFusion コンポーネントを定義しています。

```
<cfcomponent>
    <cffunction
    name = "echoString"
    returnType = "string"
    output = "no"
    access = "remote">
        <cfargument name = "input" type = "string">
        <cfreturn #arguments.input#>
    </cffunction>
</cfcomponent>
```


Dreamweaver で登録したコンポーネントは、[アプリケーション] パネルの [コンポーネント] タブに表示されます。

ブラウザで WSDL ファイルをリクエストすると、次の内容が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://ws"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://ws"
xmlns:intf="http://ws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://rpc.xml.coldfusion"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by ColdFusion -->
<wsdl:types>
<schema targetNamespace="http://rpc.xml.coldfusion"
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<complexType name="CFCInvocationException">
<sequence/>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="CFCInvocationException">
<wsdl:part name="fault" type="tns1:CFCInvocationException"/>
</wsdl:message>
<wsdl:message name="echoStringResponse">
<wsdl:part name="echoStringReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="echoStringRequest">
<wsdl:part name="input" type="xsd:string"/>
    </wsdl:message>
    <wsdl:portType name="echo">
    <wsdl:operation name="echoString" parameterOrder="input">
    <wsdl:input message="impl:echoStringRequest" name="echoStringRequest"/>
    <wsdl:output message="impl:echoStringResponse"
                name="echoStringResponse"/>
    <wsdl:fault message="impl:CFCInvocationException" name="CFCInvocationException"/>
    </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="echo.cfcSoapBinding" type="impl:echo">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/
            http"/>
    <wsdl:operation name="echoString">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="echoStringRequest">
```

```
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                    encoding/" namespace="http://ws" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="echoStringResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                    encoding/" namespace="http://ws" use="encoded"/>
    </wsdl:output>
    <wsdl:fault name="CFCInvocationException">
    <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
                    encoding/" name="CFCInvocationException" namespace=
                    "http://ws" use="encoded"/>
    </wsdl:fault>
    </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="echoService">
    <wsdl:port binding="impl:echo.cfcSoapBinding" name="echo.cfc">
    <wsdlsoap:address location="http://localhost:8500/ws/echo.cfc"/>
    </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Web サービスの公開

1 次の内容の ColdFusion ページを作成します。

```
<cfcomponent output="false">
    <cffunction
                name = "echoString"
                returnType = "string"
                output = "no"
                access = "remote">
        <cfargument name = "input" type = "string">
        <cfreturn #arguments.input#>
    </cffunction>
</cfcomponent>
```

2 このファイルに "echo.cfc" という名前を付けて、Web のルートディレクトリに保存します。

3 次の内容の ColdFusion ページを作成します。

```
<cfinvoke webservice ="http://localhost/echo.cfc?wsdl"
    method ="echoString"
    input = "hello"
    returnVariable="foo">

<cfoutput>#foo#</cfoutput>
```

4 このファイルに "echoclient.cfm" という名前を付けて、Web のルートディレクトリに保存します。

5 ブラウザで "echoclient.cfm" をリクエストします。

次の文字列がブラウザに表示されます。

```
hello
```

次のコードでも Web サービスを呼び出せます。

```
<cfscript>
    ws = CreateObject("webservice", "http://localhost/echo.cfc?wsdl");
    wsresults = ws.echoString("hello");
    writeoutput(wsresults);
</cfscript>
```

ColdFusion コンポーネントによる Web サービスのデータ型の定義

ColdFusion では、プロパティのみが含まれているコンポーネントを定義できます。これを使用して、Web サービスのデータ型を定義することができます。たとえば、住所を表すプロパティを持つコンポーネントが、"address.cfc" ファイルで次のように定義されているとします。

```
<cfcomponent>
    <cfproperty name="AddrNumber" type="numeric">
    <cfproperty name="Street" type="string">
    <cfproperty name="City" type="string">
    <cfproperty name="State" type="string">
    <cfproperty name="Country" type="string">
</cfcomponent>
```

また、名と姓のプロパティを持つコンポーネントが、"filename.cfc" で次のように定義されているとします。

```
<cfcomponent>
    <cfproperty name="Firstname" type="string">
    <cfproperty name="Lastname" type="string">
</cfcomponent>
```

Web サービスを提供する ColdFusion コンポーネントでは、次のように、address と name を使用してデータ型を定義できます。

```
<cfcomponent>
    <cffunction
        name="echoName" returnType="name" access="remote" output="false">
            <cfargument name="input" type="name">
            <cfreturn #arguments.input#>
    </cffunction>

    <cffunction
        name="echoAddress" returnType="address" access="remote" output="false">
            <cfargument name="input" type="address">
            <cfreturn #arguments.input#>
    </cffunction>
</cfcomponent>
```

注意：コンポーネントファイルが Web ルートの下のディレクトリにない場合は、コンポーネントファイルが含まれているディレクトリへの Web サーバーマッピングを作成します。ColdFusion マッピングを使用して Web サービスにアクセスすることはできません。

この Web サービスの WSDL ファイルには、複合型の name と address のデータ定義が含まれています。それぞれの定義では、対応する ColdFusion コンポーネントファイルの指定に基づいて、型が定義されています。たとえば、name の定義は次のようになります。

```
<complexType name="name">
     <sequence>
       <element name="firstname" nillable="true" type="soapenc:string"/>
       <element name="lastname" nillable="true" type="soapenc:string"/>
     </sequence>
</complexType>
```

returnType 属性では、次の例のように、CFC の配列を指定することもできます。

```
<cfcomponent>
    <cffunction
    name="allNames" returnType="name[]" access="remote" output="false">
    <cfset var returnarray = ArrayNew(1)>
    <cfset var temp = "">
    <cfquery name="empinfo" datasource="cfdocexamples">
     SELECT firstname, lastname
     FROM employee
    </cfquery>
    <cfloop query="empinfo" >
          <cfobject component="name" name="tempname">
          <cfset tempname.Firstname = #empinfo.firstname#>
          <cfset tempname.Lastname = #empinfo.lastname#>
          <cfset temp = ArrayAppend(returnarray, tempname)>
        </cfloop>
        <cfreturn returnarray>
    </cffunction>
</cfcomponent>
```

この Web サービスを呼び出すと、CFC の配列が返されます。CFC のプロパティにアクセスするには、次の例のように、ドット表記法を使用します。

```
<cfinvoke webservice ="http://localhost:8500/ws/cfcarray.cfc?wsdl"
    method ="allNames"
    returnVariable="thearray">

<cfif IsArray(thearray)>
<h1>loop through the employees</h1>
<p>thearray has <cfoutput>#ArrayLen(thearray)#</cfoutput> elements.</p>
<cfloop index="i" from="1" to="#ArrayLen(thearray)#">
    <cfoutput>#thearray[i].firstname#, #thearray[i].lastname# </cfoutput><br>
</cfloop>
<cfelse>
     <h1>Error: thearray is not an array</h1>
</cfif>
```

document-literal スタイルの Web サービスの公開

利用側がメソッドと引数を指定する RPC 指向のオペレーションに加えて、document-literal スタイルの Web サービスも公開できます。document-literal スタイルの WSDL は、RPC 呼び出し規則ではなく XML スキーマを使用するようにクライアントに指示します。

公開されている Web サービスは、ほとんどの場合、document-literal スタイルか RPC スタイルのいずれかになります。タイプを確認するには、次の例のように、WSDL ドキュメントを開いて soap:binding 要素を検索し、style 属性を確認します。

```
<wsdl:binding name="WeatherForecastSoap" type="tns:WeatherForecastSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
```

この例では、スタイルは document-literal です。WSDL を調べて、呼び出せるメソッドと、各メソッドのパラメータを確認する必要があります。

クライアントでは、Web サービスを呼び出す cfinvoke タグやその他の ColdFusion メソッドによって、スタイルが自動的に処理されます。ほとんどの場合、変更の必要はありません。同様に、document-literal スタイルの Web サービスとして CFC を公開する場合も、適切な WSDL が ColdFusion によって自動的に作成され管理されます。

document-literal スタイルの Web サービスとして CFC を公開するには、cfcomponent style="document" を指定し、document-literal スタイルの Web サービスで必要な他の属性を指定します。たとえば、次の CFC は document-literal スタイルで公開されます。

```
<cfcomponent style="document" >
    <cffunction
name = "getEmp"
returntype="string"
output = "no"
access = "remote">
    <cfargument name="empid" required="yes" type="numeric">
    <cfset var fullname = "">
    <cfquery name="empinfo" datasource="cfdocexamples">
     SELECT emp_id, firstname, lastname
     FROM employee
     WHERE emp_id = <cfqueryparam cfsqltype="cf_sql_integer"
            value="#arguments.empid#">
    </cfquery>
    <cfif empinfo.recordcount gt 0>
     <cfset fullname = empinfo.lastname & ", " & empinfo.firstname>
    <cfelse>
    <cfset fullname = "not found">
    </cfif>
<cfreturn #fullname#>
</cffunction>
</cfcomponent>
```

Web サービスのセキュリティ保護

公開する Web サービスへのアクセスを制限して、Web サービスを呼び出せるユーザーを管理できます。Web サーバーを使用して、Web サービスが含まれているディレクトリへのアクセスを制御することもできますが、ColdFusion セキュリティを使用して、ColdFusion ページへのアクセスを制御するのと同様に Web サービスへのアクセスを制御できます。

CFC ファイルの HTML 記述をブラウズするには、ブラウザでそのファイルの URL を指定して、そのファイルをリクエストします。デフォルトでは、CFC ファイルを直接参照するすべての URL へのアクセスが保護されており、パスワードの入力を要求するプロンプトがリクエスト時に表示されます。ファイルを表示するには ColdFusion の RDS パスワードを使用します。

CFC ファイルの表示に対するセキュリティを無効にするには、ColdFusion Administrator を使用して RDS パスワードを無効にします。

詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

Web サーバーによるアクセス制御

IIS や Apache などのほとんどの Web サーバーには、HTTP 基本認証メカニズムを使用したディレクトリへのアクセス保護機能が実装されています。クライアントが、保護されたディレクトリにあるリソースにアクセスしようとして、正しく認証されなかった場合、Web サーバーから自動的に認証チャレンジ (通常は HTTP エラー 401 アクセス拒否エラー) が送信されます。

それに応答して、クライアントのブラウザにログインプロンプトが開き、ユーザー名とパスワードのフィールドが表示されます。ユーザーがこの情報を送信すると、ブラウザから Web サーバーにデータが送信されます。認証が完了すると、Web サーバーはディレクトリへのアクセスを許可します。また、ブラウザが開いている間は認証データがキャッシュされるので、以降のリクエストには自動的に認証データが含められます。

Web サービスクライアントは、ユーザー名とパスワードの情報をリクエストの一部として渡すこともできます。cfinvoke タグには username 属性と password 属性が含まれており、HTTP 基本認証を使用して Web サーバーにログイン情報を渡すことができます。次の例のように、Web サービスを呼び出すときにこれらの属性を含めることができます。

```
<cfinvoke
    webservice = "http://some.cfc?wsdl"
    returnVariable = "foo"
    ...
    username="aName"
    password="aPassword">
<cfoutput>#foo#</cfoutput>
```

ユーザー名とパスワードはコロンで区切られ、base64 バイナリ形式でエンコードされた文字列として authorization リクエストヘッダに挿入されます。このユーザー名 / パスワードの受け渡し方法は、Web サーバーによって使用される HTTP 基本認証メカニズムと互換性があります。

ColdFusion Administrator を使用して、Web サービスをあらかじめ定義できます。その際に、Web サービスへのリクエストに挿入するユーザー名とパスワードを指定できます。したがって、cfinvoke タグを使用してこの情報をエンコードする必要はありません。ColdFusion Administrator で Web サービスを定義する方法の詳細については、1069 ページの「ColdFusion Administrator での Web サービスの設定」を参照してください。

ColdFusion によるアクセス制御

Web サーバーを使用して Web サービスへのアクセスを制御する代わりに、独自のセキュリティメカニズムとして "Application.cfc" または "Application.cfm" ファイルでユーザー名 / パスワードの文字列を処理できます。この場合は、次に示す "Application.cfc" ファイルの onRequestStart メソッドのように、cflogin タグを使用して authorization ヘッダからユーザー名 / パスワード情報を取得し、バイナリ文字列をデコードして、ユーザー名とパスワードを抽出します。

```
<cflogin>
    <cfset isAuthorized = false>

    <cfif isDefined("cflogin")
        <!--- Verify user name from cflogin.name and password from
    cflogin.password using your authentication mechanism. --->
        >
        <cfset isAuthorized = true>
    </cfif>
</cflogin>

<cfif not isAuthorized>
    <!--- If the user does not pass a user name/password, return a 401 error.
        The browser then prompts the user for a user name/password. --->
    <cfheader statuscode="401">
    <cfheader name="WWW-Authenticate" value="Basic realm=""Test""">
    <cfabort>
</cfif>
```

この例では、検証方法は示していません。検証の詳細については、337 ページの「アプリケーションの保護」を参照してください。

Web サービスを公開するための推奨事項

ColdFusion の Web サービスには、アプリケーション機能の公開や利用のための強力なメカニズムが備わっていますが、公開する Web サービスを生成する前に、次の推奨事項を考慮してください。

1. 公開する Web サービスでは、クエリーや構造体などの ColdFusion の複合型はできるだけ使用しないでください。利用側は、特に ColdFusion 以外のテクノロジーを使用した Web サービスを利用する場合は、複合型を処理するための特別なデータ構造体を作成する必要があります。
2. Web サービス用に実装した ColdFusion コンポーネントをインターネットで公開する前に、ローカルでテストしてください。

リクエストヘッダとレスポンスヘッダの使用

ColdFusion には、Web サービスでリクエストヘッダやレスポンスヘッダを取得および設定するための関数セットが用意されています。これらの関数を使用して、Web サービスのリクエストからレスポンスヘッダを取得したり、mustUnderstand 属性が True に設定されたリクエストで SOAP ヘッダを作成します。

一般に、Web サービスクライアントと Web サービス CFC では異なる関数を使用します。

クライアントの場合 :

- AddSOAPRequestHeader : SOAP ヘッダを設定するためにリクエストの前に呼び出されます。
- GetSOAPResponseHeader : SOAP ヘッダを取得するためにリクエストの後に呼び出されます。
- GetSOAPResponse : SOAP レスポンスを取得するためにリクエストの後に呼び出されます。

Web サービス CFC の場合 :

- IsSOAPRequest : CFC メソッドが Web サービスとして呼び出されているかどうかを判断するために呼び出されます。
- GetSOAPRequestHeader : クライアントによって設定された SOAP ヘッダを取得するために呼び出されます。
- GetSOAPRequest : クライアントによって送信された SOAP リクエストを取得するために呼び出されます。
- AddSOAPResponseHeader : クライアントに返す SOAP ヘッダを設定するために呼び出されます。

注意 : CFC でこれらの関数を使用できるのは、CFC メソッドが Web サービスとして使用されている場合のみです。IsSOAPRequest 関数を使用して、CFC メソッドが Web サービスとして呼び出されているかどうかを判断してください。

次に示すサンプル CFM ページでは、AddSOAPRequestHeader、getSOAPRequest、および GetSOAPResponse 関数を使用しています。

```
 <cfsavecontent variable="my_xml">
<Request xmlns="http://www.oasis-open.org/asap/0.9/asap.xsd">
<SenderKey>ss</SenderKey>
<ReceiverKey>zz</ReceiverKey>
<ResponseRequired>Yes</ResponseRequired>
<RequestID>id</RequestID>
</Request>
</cfsavecontent>
<cfset xml_obj = xmlparse(my_xml)>

<cfscript>
ws = CreateObject("webservice", "http://localhost:8500/soapexamples/HeaderFuncs.cfc?WSDL");
addSOAPRequestHeader(ws, "http://www.cfdevguide.com/", "testrequestheader", "#xml_obj#");
</cfscript>

<cfscript>
ret=ws.showSOAPHeaders();
inxml = getSOAPRequest(ws);
outxml = getSOAPResponse(ws);
</cfscript>

<cfoutput>
<h2>Return Value</h2>
<!--- This code is XML, so use HTMLCodeFormat. --->
The return value was #ret#
<h2>Complete Request XML</h2>
#htmlcodeformat(inxml)#
<h2>Complete Response XML</h2>
#htmlcodeformat(outxml)#
</cfoutput>
```

次のサンプル CFC では、IsSOAPRequest および AddSOAPResponseHeader 関数を使用しています。

```
<cfcomponent>
<cffunction
    name = "showSOAPHeaders"
    returnType = "string"
    output = "no"
    access = "remote"
     hint="After calling this function, use GetSOAPRequest and GetSOAPResponse to view headers">
    <cfset var xml_obj = "">
    <cfset var ret = "">

<cfif IsSOAPRequest()>
<!--- Define a response header --->
<cfsavecontent variable="response_xml">
    <ThisResponseHeader xmlns="http://www.cfdevguide.com">
    <CreatedDateTime><cfoutput>#now()#</cfoutput></CreatedDateTime>
    <ExpiresInterval>6000</ExpiresInterval>
    </ThisResponseHeader>
</cfsavecontent>
<cfset xml_obj = xmlparse(response_xml)>
<!--- Add the response header --->
<cfscript>
addSOAPResponseHeader("http://www.cfdevguide.com/", "testresponseheader", "#xml_obj#");
ret = "Invoked as a web service. Use GetSOAPRequest and GetSOAPResponse to view headers.";
    </cfscript>
<cfelse>
<cfset ret = "Not invoked as a web service">
</cfif>
<cfreturn ret>
</cffunction>
</cfcomponent>
```

複合データ型の処理

Web サービスにおける複合データ型の処理は、次のいずれかに分類されます。

- Web サービスを利用するために、Web サービスのデータ型を ColdFusion データ型にマッピングする
- Web サービスを公開するために、クライアントが ColdFusion データ型を参照する方法を理解する

複合データ型が使用されている Web サービスの利用

次の表に、WSDL データ型を ColdFusion データ型に変換する方法を示します。

ColdFusion のデータ型	**WSDL のデータ型**
数値	SOAP-ENC:double
ブール値	SOAP-ENC:boolean
文字列	SOAP-ENC:string
配列	SOAP-ENC:Array
数値	SOAP-ENC:float
バイナリ	xsd:base64Binary
日付	xsd:dateTime
void (オペレーションから何も返されない)	
構造体	複合型

この表では、複合データ型が ColdFusion の構造体にマッピングされることが示されています。ColdFusion の構造体は、データを柔軟に表現できます。1 次元配列、多次元配列、別の構造体などを含む構造体を作成できます。

複合型から構造体へのマッピングは、自動的には行われません。構造体としてデータにアクセスするためには、データを何らかの方法で処理する必要があります。次のセクションでは、Web サービスに複合型を渡す方法と、Web サービスから返された複合型を処理する方法について説明します。

複合型の入力パラメータを Web サービスに渡す方法

Web サービスは、複合データ型を入力値として取ることがあります。その場合は、複合データ型をモデル化した ColdFusion の構造体を作成して、その構造体を Web サービスに渡します。

たとえば、次の WSDL ファイルからの抜粋は、Employee という複合型の定義を示しています。

```
<s:complexType name="Employee">
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="fname" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="lname" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="active" type="s:boolean" />
      <s:element minOccurs="1" maxOccurs="1" name="age" type="s:int" />
      <s:element minOccurs="1" maxOccurs="1" name="hiredate" type="s:dateTime" />
      <s:element minOccurs="1" maxOccurs="1" name="number" type="s:double" />
    </s:sequence>
</s:complexType>
```

Employee データ型の定義には 6 つの要素が含まれており、各要素にはデータ型および名前が含まれています。

この WSDL ファイルには、Employee データ型を使用するメッセージ定義が含まれています。このメッセージでは、次のコードのような入力パラメータが定義されています。

```
<message name="updateEmployeeInfoSoapIn">
    <part name="thestruct" type="s0:Employee" />
</message>
```

また、この WSDL ファイルでは、updateEmployeeInfo というオペレーションが定義されています。このオペレーションは、従業員情報に基づいて従業員データベースを更新します。このオペレーションでは、次のコードのように Employee 型のパラメータを入力値として取ります。

```
<operation name="updateEmployeeInfo">
    <input message="s0:updateEmployeeInfoSoapIn" />
</operation>
```

updateEmployeeInfo オペレーションを呼び出すには、次のコードのように、ColdFusion 構造体を作成し、Employee の 6 つの要素に対応する構造体の 6 つのフィールドを初期化して、オペレーションを呼び出します。

```
<!--- Create a structure using CFScript, then call the web service. --->
<cfscript>
    stUser = structNew();
    stUser.active = TRUE;
    stUser.fname = "John";
    stUser.lname = "Smith";
    stUser.age = 23;
    stUser.hiredate = createDate(2002,02,22);
    stUser.number = 123.321;

    ws = createObject("webservice", "http://somehost/EmployeeInfo.asmx?wsdl");
    ws.updateEmployeeInfo(stUser);
</cfscript>
```

入力パラメータを複合型として渡すときには構造体を使用できます。ただし、複合型をモデル化して構造体を構築するためには、Web サービスの WSDL ファイルを調べて複合型のレイアウトを把握する必要があります。この手順にはある程度の慣れが必要です。

複合型の戻り値の処理

Web サービスから複合型が返される場合は、その戻り値を直接 ColdFusion 変数に書き込むことができます。

前のセクションでは、Employee という複合データ型を使用して、オペレーションの入力パラメータが定義されていました。WSDL ファイルでは、次のコードのように、Employee タイプを使用して戻り値を定義することもできます。

```
<message name="updateEmployeeInfoSoapOut">
    <part name="updateEmployeeInfoResult" type="s0:Employee" />
</message>
<operation name="updateEmployeeInfo">
    <input message="s0:updateEmployeeInfoSoapIn" />
    <output message="s0:updateEmployeeInfoSoapOut" />
</operation>
```

この例では、オペレーション updateEmployeeInfo は入力として複合型を取り、出力として複合型を返します。入力パラメータを処理するには、構造体を作成しました。戻り値を処理するには、次の例のように戻り値を ColdFusion 変数に書き込みます。

```
<!--- Create a structure using CFScript, then call the web service. --->
<!--- Write the returned value to a ColdFusion variable. --->
<cfscript>
    stUser = structNew();
    stUser.active = TRUE;
    stUser.fname = "John";
    stUser.lname = "Smith";
    stUser.age = 23;
    stUser.hiredate = createDate(2002,02,22);
    stUser.number = 123.321;

    ws = createObject("webservice", "http://somehost/echosimple.asmx?wsdl");
    myReturnVar = ws.echoStruct(stUser);

</cfscript>

<!--- Output the returned values. --->
<cfoutput>
    <br>
    <br>Name of employee is: #myReturnVar.fname##myReturnVar.lname#
    <br>Active status: #myReturnVar.active#
    <br>Age:#myReturnVar.age#
    <br>Hire Date: #myReturnVar.hiredate#
    <br>Favorite Number: #myReturnVar.number#
</cfoutput>
```

myReturnVar 変数の要素にアクセスするには、構造体のフィールドにアクセスする場合と同様に、ドット表記法を使用します。複合型で要素がネストしている場合は、構造体でフィールドが複数ネストしている場合と同様に、a.b.c.d のようなドット表記法を使用して、必要なネストレベルの要素にアクセスします。

ただし、変数 myReturnVar は ColdFusion の構造体ではありません。この変数は複合型のコンテナですが、ColdFusion 構造体の属性は持っていません。この変数で ColdFusion 関数 isStruct を呼び出すと、False が返されます。

次の例のように、変数の内容を ColdFusion 構造体にコピーできます。

```
<cfscript>
...
    ws = createObject("webservice", "http://somehost/echosimple.asmx?wsdl");
    myReturnVar = ws.echoStruct(stUser);

    realStruct = structNew();
    realStruct.active = #myReturnVar.active#;
    realStruct.fname = "#myReturnVar.fname#";
    realStruct.lname = "#myReturnVar.lname#";
    realStruct.age = #myReturnVar.age#;
    realStruct.hiredate = #myReturnVar.hiredate#;
    realStruct.number = #myReturnVar.number#;

</cfscript>
```

realStruct で IsStruct を呼び出すと True が返され、すべての ColdFusion 構造体関数を使用して処理できます。

この例のように、Web サービスから返された複合型を処理するには、ColdFusion の変数および構造体が役立ちます。ColdFusion 変数に書き込まれた複合型の要素にアクセスするには、Web サービスの WSDL ファイルを調べます。WSDL ファイルには Web サービスの API が定義されており、Web サービスから返されたデータを処理するために必要な情報が提供されます。

複合データ型を使用する Web サービスの公開

ColdFusion の struct および query データ型は、WSDL 固有のデータ型にマッピングされません。struct または query 型のパラメータを使用する ColdFusion Web サービスを公開する場合は、利用側のアプリケーションがそのデータを処理できる必要があります。

注意：ColdFusion Web サービスを別の ColdFusion アプリケーションが利用する場合は、特別な処理は必要ありません。Web サービスの struct および query データ型は、ColdFusion によって正しく利用側にマッピングされます。詳細については、1070 ページの「ColdFusion Web サービスの利用」を参照してください。

構造体の公開

ColdFusion 構造体では、キーと値のペアをいくつでも保持でき、任意の ColdFusion データ型を使用できます。構造体は、データを表すには便利で強力ですが、SOAP 1.1 エンコード方式や XML スキーマ仕様で定義されている XML データ型には直接マッピングできません。したがって、ColdFusion 構造体はカスタム型として扱われます。WSDL 内の複合型の XML スキーマは次のようになります。

```
<complexType name="mapItem">
     <sequence>
       <element name="key" nillable="true" type="xsd:anyType"/>
       <element name="value" nillable="true" type="xsd:anyType"/>
     </sequence>
</complexType>
<complexType name="Map">
     <sequence>
       <element maxOccurs="unbounded" minOccurs="0" name="item" type="apachesoap:mapItem"/>
     </sequence>
</complexType>
```

この複合型は、構造体の表現を定義しています。構造体のキーおよび値には任意の型が使用できます。

ColdFusion 構造体の WSDL マッピングでは、最後のフィールドを除き、構造体のキーと値の各ペアは構造体の次の要素を指しています。最後のフィールドには値が含まれています。キーと値のペアにアクセスするには、ドット表記法を使用します。

クエリーの公開

ColdFusion では、query データ型は WSDL の QueryBean 型として公開されます。次の WSDL ファイルの抜粋のように、QueryBean データ型には 2 つの要素が含まれています。

```
<complexType name="QueryBean">
    <all>
        <element name="data" nillable="true" type="intf:ArrayOf_SOAP-ENC_Array" />
        <element name="ColumnList" nillable="true"
    type="intf:ArrayOf_SOAP-ENC_string" />
    </all>
</complexType>
```

次の表で、QueryBean の要素について説明します。

要素名	説明
ColumnList	列名が含まれている文字列の配列
data	クエリーデータが含まれている 2 次元配列

QueryBean の WSDL ファイルでは、これらの要素が次のように定義されています。

```
<complexType name="ArrayOf_SOAP-ENC_Array">
    <complexContent>
        <restriction base="SOAP-ENC:Array">
            <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="SOAP-ENC:Array[]" />
        </restriction>
    </complexContent>
</complexType>
<complexType name="ArrayOf_SOAP-ENC_string">
    <complexContent>
    <restriction base="SOAP-ENC:Array">
        <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd:string[]" />
    </restriction>
    </complexContent>
</complexType>
```

SOAP リクエストとレスポンスに関するトラブルシューティング

ColdFusion では、次の機能を使用して、SOAP リクエストとレスポンスに関するトラブルシューティングを行えます。

- getSOAPRequest および getSOAPResponse 関数。
- TCP モニタ。

SOAP リクエストとレスポンスの表示

getSOAPRequest および getSOAPResponse 関数を使用すると、Web サービスとの間でやり取りされる XML を取得して表示することができます。上級ユーザーであれば、こうした情報を利用してカスタム機能を実現できますが、通常はデバッグ目的でこれらの関数を使用します。

これらの関数は次の状況で使用します。

- GetSOAPRequest : クライアントでは、Web サービスリクエストの後にこの関数を呼び出します。Web サービス CFC では、Web サービス CFC メソッドでこの関数を呼び出します。
- GetSOAPResponse : クライアントでは、Web サービスリクエストの完了後にこの関数を呼び出します。Web サービス CFC では、このメソッドを使用できません。

次の例では、Web サービスクライアントで GetSOAPRequest と GetSOAPResponse 関数を使用しています。

```
<cfscript>
    ws = CreateObject("webservice", "http://localhost:8500/soapexamples/tester.cfc?WSDL");
    addSOAPRequestHeader(ws, "http://mynamespace/", "username", "randy");
    ret = ws.echo_me("value");
</cfscript>

<cfset soapreq = GetSOAPRequest(ws)>
<h2>SOAP Request</h2>
<cfdump var="#soapreq#">

<cfset soapresp = GetSOAPResponse(ws)>
<h2>SOAP Response</h2>
<cfdump var="#soapresp#">
...
```

次の例では、Web サービス CFC メソッドで GetSOAPRequest 関数を使用しています。

```
<cfcomponent displayName="testerdebug" hint="Test for underscores">

<cffunction access="remote" name="echo_me" output="false" returntype="string"
displayname="Echo Test" hint="Header test">
<cfargument name="in_here" required="true" type="string">
     <cfset var soapreq = "">

<cfif IsSOAPRequest()>
<cfset soapreq = GetSOAPRequest()>
    <cflog text="#soapreq#"
    log="APPLICATION"
    type="Information">
...
```

TCP モニタの使用

TCPMonitor は swing ベースのアプリケーションで、これを使用して HTTP トラフィックのリクエストとレスポンスを監視できます。また、SOAP トラフィックのリクエストおよびレスポンスフローも監視できます。TCPMonitor は、Macromedia JRun で使用されていたスニファサービスの代わりとして機能します。

TCPMonitor の実行

❖ Windows および UNIX プラットフォームでは、"jrun_root/bin" ディレクトリにあるスニファユーティリティを起動することで、TCPMonitor を実行できます。

TCP モニタのメインウィンドウが表示されます。

TCPMonitor は swing ベースのアプリケーションで、これを使用して HTTP トラフィックのリクエストとレスポンスを監視できます。また、SOAP トラフィックのリクエストおよびレスポンスフローも監視できます。

TCPMonitor を実行するには :

1 Windows および UNIX プラットフォームでは、"<ColdFusion のルートディレクトリ >/bin" (サーバー設定の場合) または "<JRE のインストールディレクトリ >/bin" (マルチサーバー設定の場合) ディレクトリにあるスニファユーティリティを起動することで、TCPMonitor を実行できます。

TCP モニタのメインウィンドウが表示されます。

注意 : J2EE 設定の場合は、次のコマンドを使用して、JAR ファイルからユーティリティを直接実行します。

```
java -cp cf_webapp_root/WEB-INF/cfusion/lib/axis.jar java org.apache.axis.utils.tcpmon [listening_port]
[target_host] [target_port]
```

2 次の表の説明に従って、メインウィンドウに値を入力します。

フィールド	説明
Listen Port#	8123 など、受信される接続を監視するローカルポート番号を入力します。サーバーが実行されている通常のポートの代わりに、このポートにリクエストを送信します。TCPMonitor はリクエストを受け取って、ターゲットポートに転送します。
Listener	TCPMonitor を JRun のスニファサービスとして使用する場合は、[Listener] を選択します。
Proxy	TCPMonitor のプロキシサポートを有効にするには、[Proxy] を選択します。
Target Hostname	受信した接続の転送先となるターゲットホストを入力します。 たとえば、ローカルサーバーのサービスを監視している場合、ホスト名は localhost です。
Target Port#	TCPMonitor の接続先となるターゲットマシンのポート番号を入力します。たとえば、サーバー設定のローカル ColdFusion サーバーで実行しているサービスを監視している場合、デフォルトのポート番号は 8500 です。
HTTP Proxy Support	このチェックボックスは、TCPMonitor のプロキシサポートを設定する場合にのみ選択します。

コマンドラインで TCPMonitor を呼び出す場合は、オプションで Listen Port#、Target Hostname、および Target Port# の値を指定できます。TCPMonitor のシンタックスは次のとおりです。

```
java org.apache.axis.utils.tcpmon [listening_port] [target_host] [target_port]
```

3 [Add] をクリックして、このプロファイルを TCPMonitor セッションに追加します。

トンネリングされた新しい接続のためのタブが表示されます。

4 新しいタブを選択します。ポートの競合がある場合は、[Request] パネルに警告が表示されます。

5 この TCPMonitor セッションで定義した Listen Port を使用して、ページを要求します。たとえば、Listen Port に 8123 と入力した場合は、ブラウザに次の URL を入力します。

```
http://localhost:8123/
```

TCPMonitor に、現在のリクエストおよびレスポンス情報が表示されます。

それぞれの接続について、[Request] パネルにリクエストが表示され、[Response] パネルにレスポンスが表示されます。リクエストとレスポンスのペアは、すべてログに記録されます。トップパネルでエントリを選択すると、選択したペアを表示できます。

6 [Save] をクリックすると、後で表示するために結果をファイルに保存できます。保存対象としない古い要求のトップパネルをクリアするには、[Remove Selected] または [Remove All] をクリックします。

7 [Resend] をクリックすると、現在表示中のリクエストを再送して、新しいレスポンスを表示できます。再送する前に [Request] パネルを編集すると、異なるリクエストの効果をテストできます。

8 ポートを変更する場合は、[Stop] をクリックし、ポート番号を変更してから [Start] をクリックします。

9 別のリスナーを追加する場合は、[Admin] タブをクリックして、前の説明に従って値を入力します。

10 [Close] をクリックすると、この TCPMonitor セッションを終了できます。

ColdFusion Web サービスの使用

ColdFusion の多くの機能を、ドキュメントスタイルまたはリテラルスタイルの SOAP Web サービスとして公開できるようになりました。Web サービスを利用すると PHP、.NET、Ruby などの他言語からも ColdFusion の機能を利用できます。次のタグ (およびその子タグ) の機能を SOAP サービスとして利用できます。

- cfchart
- cfdocument

- cfimage
- cfmail
- cfpop
- cfpdf

ColdFusion Web サービスを利用するための手順

ColdFusion Web サービスを使用するためには、まず ColdFusion Administrator を使用して、ColdFusion サービスにアクセスできるようにする必要があります。アクセス権限の設定が完了したら、次の設定を行います。

1 ColdFusion Web サービスを呼び出すファイルをサーバー (PHP など) にアップロードします。

2 HTTP ソース (アップロードしたファイル) の URL、サービスユーザー名、パスワードなどの属性を指定して ColdFusion Web サービスを呼び出します。

3 ColdFusion サーバーからレスポンスが返されます。

ColdFusion サービスの有効化

ColdFusion Administrator では、ユーザーのアクセス権限を設定できますが、ユーザーごとに異なるサービスへのアクセスを許可することもできます。たとえば、社内アプリケーションに対してはすべてのサービスへのアクセスを許可し、それ以外のアプリケーションに対してはイメージサービスとグラフ作成サービスのみへのアクセスに制限することができます。

サービスアクセスを有効にするには :

1 [Administrator Services]-[IP アドレス] ページで、サービスにアクセスできる URL アドレスを指定します。アドレスを 1 つずつ指定することもできますが、10-30 のようにアドレスの範囲を指定することもできます。また、ワイルドカードとして * を使用することもできます。IPv4 アドレスと IPv6 アドレスの両方を指定できます。

   たとえば、次のようなパターンでアドレスを指定できます。

   10.*.*.*

   10*.30-50.20-30

   10:10:10:10:10:10:10-FF:*

2 [Administrator Service]-[ユーザーマネージャ] ページで、サービスへのアクセスを許可するユーザーを指定します。ユーザー名とパスワードを指定する必要があります。また、このページの [公開サービス] セクションで、アクセスを許可するサービスを指定します。

SOAP を使用した ColdFusion サービスへのアクセス

ドキュメントスタイルの Web サービスとして ColdFusion サービスにアクセスするには、SOAP を使用します。使用可能なサービスのリクエストとレスポンスを確認するには、"cfWebRoot¥CFIDE¥services" ディレクトリにあるサービス別の WSDL を参照してください。サービス別の WSDL は、CFC ファイル "chart.cfc"、"document.cfc"、"image.cfc"、"mail.cfc"、"pdf.cfc"、"pop.cfc" から参照できます。

たとえば、イメージサービスを確認するには、次のような URL を使用します。

```
http://<CF server >:<port>/CFIDE/services/image.cfc?wsdl
```

WSDL では、各サービスを通じて公開されるすべてのメソッドと属性を確認できます。属性の説明は、対応する ColdFusion タグまたは関数の属性と同じです。

PHP サーバーから ColdFusion サービスへのアクセス

PHP から ColdFusion サービスにアクセスするには :

1 PHP をインストールします。

2 ライブラリまたはフレームワーク (WSO2 Web services Frame Work for PHP など) を使用して、WSDL を PHP クラスに変換します。

3 必要なパラメータを指定してクラスを呼び出します。

次の例は、イメージに境界線を追加する PHP ページ内のセクションを示しています。

```
$input = new AddBorder();
//Fill in the class fields of $input to match your business logic
$input->serviceusername = "myuser";
$input->servicepassword = "mypassword";
$input->source = "http://myPHPSite/Images/image.jpg";
$input->thickness = "30";
$input->color = "blue";
$input->bordertype = "";
// call the operation
$response = $proxy->AddBorder($input);
```

このコードの内容は次のとおりです。

serviceusername は、リクエストされているサービスへのアクセス権限を持つ、ColdFusion Administrator で設定されたユーザー名です。

servicepassword は、サービスユーザー名に対して設定されたパスワードです。

source は、PHP サーバーにあるイメージのソース URL です。その他の属性は、ColdFusion の ImageAddborder() 関数と同じです。

イメージに対するバッチ操作

次のコードは、アップロードされたイメージに対して複数の操作を実行します。ここでは、batchOperation() を使用して、イメージを切り取り、そのイメージに境界線を追加しています。

```
$input = new batchOperation();
//TODO: fill in the class fields of $input to match your business logic
//Crop
$element1 = new Element();
$element2 = new Element();
$element3 = new Element();
$element4 = new Element();
$elementArray1 = new ArrayOf_xsd_anyType();
$element1->key = 'x';
$element1->value = '10';
$element2->key = 'y';
$element2->value = '10';
$element3->key = 'width';
$element3->value = '200';
$element4->key = 'height';
$element4->value = '200';
$elementArray1->item[0] = $element1;
$elementArray1->item[1] = $element2;
$elementArray1->item[2] = $element3;
$elementArray1->item[3] = $element4;
$ElementcollectionCrop = new Elementcollection();
$ElementcollectionCrop->key = 'Crop';
$ElementcollectionCrop->value = $elementArray1;
//AddBorder
$element5 = new Element();
```

```
$element6 = new Element();
$element7 = new Element();
$elementArray2 = new ArrayOf_xsd_anyType();
$element5->key = 'thickness';$element5->value = '30';
$element6->key = 'color';
$element6->value = 'green';
$element7->key = 'bordertype';
$element7->value = '';
$elementArray2->item[0] = $element5;
$elementArray2->item[1] = $element6;
$elementArray2->item[2] = $element7;
$ElementcollectionAddBorder = new Elementcollection();
$ElementcollectionAddBorder->key = 'AddBorder';
$ElementcollectionAddBorder->value = $elementArray2;
$input->serviceusername = "myuser";
$input->servicepassword = "mypassword";
$input->source = "http:/<php server>:<port>/image.jpg";
$input->attributes =
array($ElementcollectionCrop,$ElementcollectionAddBorder);
// call the operation
$response = $proxy->batchOperation($input);
```

.NET から ColdFusion サービスへのアクセス

Visual Studio で新規の Web サービスプロジェクトを作成し、そのプロジェクトに Web 参照として ColdFusion WSDL を追加します。次に、Web フォームなどの新規アイテムを追加します。

例 : .NET を使用した PDF の作成

次の例は、www.google.com から PDF を作成する .NET コードのサンプルです。

```
Document.DocumentService objWebService = new
Document.DocumentService();
Document.Documentsection[] docsectionArray = { };
Document.Documentitem[] docitemArray = { };
string result = objWebService.generate("myuser", "mypassword", "pdf",
"", "", "", "", "yes", "", "", "", "", "","", "", "", "", "", "", "", "", "",
"", "", "", "http://www.google.com/", "", "", "", "", docsectionArray,
docitemArray);
```

このコードの内容は次のとおりです。

Document は、ドキュメント WSDL に対する Web 参照です。

generate() は、HTML ソースから PDF を生成するメソッドです。

ColdFusion 10 での Web サービスの機能強化

ColdFusion 10 には、Axis 2 Web サービスフレームワークが統合されています。これにより、Web サービスでは WSDL 2 仕様、SOAP 1.2 プロトコル、およびドキュメントリテラルラップスタイルを使用できます。また、機能拡張により、ColdFusion 9 での Web サービスの使用時に発生する可能性があった相互運用性の問題の多くが解決されています。

次の表では統合の効果を示します。

使用機能	公開機能
WSDL 1.1 および WSDL 2.0 仕様をサポートします。	WSDL 1.1 および WSDL 2.0 仕様をサポートします。 • WSDL2 にアクセスするには、URL を http://localhost:8500/<path of cfc>?wsdl2 のように使用します。 • WSDL1 にアクセスするには、URL を http://localhost:8500/<path of cfc>?wsdl のように使用します。
ColdFusion での Axis 2 のサポートにより、次のようなスタイルで WSDL を公開する Web サービスを使用できます。 • RPC エンコード • ドキュメントリテラル • ドキュメントリテラルラップ 使用のために Axis 2 を使用するには、cfinvoke タグで wsversion に 2 を指定します。また、アプリケーションレベルでバージョンを指定することもできます。詳しくは、1092 ページの「アプリケーションレベルの変更」を参照してください。cfinvoke で wsversion に 1 を指定した場合は、Axis 1 が使用されます。ColdFusion から公開された Axis 1 サービスは、Axis 1 によってのみ使用できます。値を指定しない場合、WSDL を RPC エンコードスタイルで公開すると、ColdFusion は Axis 1 を使用して Web サービスを使用し、ドキュメントリテラルまたはドキュメントリテラルラップで公開すると、Axis 2 が使用されます。	ColdFusion 9 では、次の WSDL スタイルのみがサポートされました。 • RPC エンコード • ドキュメントリテラル cfcomponent タグで指定した style 属性により、WSDL の公開に使用されるスタイルが決まりました。ColdFusion 10 では、Axis 2 との統合により、新しいスタイルのドキュメントリテラルラップで WSDL を公開できます。このスタイルで WSDL を公開するには、cfcomponent で style 属性に wrapped を指定します。wsversion に 2 が指定されている場合にのみ、style 属性に wrapped を指定できることに注意してください。

SOAP 1.1 および SOAP 1.2 プロトコルをサポートします。
• cfinvoke を使用して Web サービスにアクセスするときに、次のコードのように serverport を指定します。
`<cfset str = structNew()>`
`<cfset str.serviceport =`
`"cfsuite.webservices.axis2.wscf.basic.cfcHttpSoap12Endpoint">`
`<cfscript>`
`//invoke method`
`ws = createObject("webservice",`
`"http://localhost:8500/cfsuite/webservices/axis2/wscf/basic.cfc?wsdl",str);`
`</cfscript>`
`<cfinvoke webservice="#ws#" method="Echo"`
`returnvariable="foo">`
`<cfoutput>#foo#</cfoutput>`
ここでは、SOAP 1.2 エンドポイントを指定したので、SOAP 1.2 プロトコルが SOAP メッセージの送信に使用されます。
	SOAP 1.1 および SOAP 1.2 をサポートします。
	ColdFusion 9 は SOAP 1.1 エンドポイントのみをサポートします。ColdFusion 10 での Axis 2 との統合により、SOAP 1.1 および SOAP 1.2 プロトコルのサポートが提供されます。
	つまり、WSVersion を 2 に設定して Web サービスを公開する場合、エンドポイント W3C SOAP 1.2 および W3C SOAP 1.1 がサポートされます。一方、値に 1 を指定した場合は、W3C SOAP 1.1 のみがサポートされます。
	WSVersion の指定について詳しくは、設定のセクションを参照してください。
	SOAP について詳しくは、www.w3.org/TR/SOAP/ を参照してください。

Axis 設定の指定

Axis のバージョンは、サーバーレベル、アプリケーションレベルまたはコンポーネントレベルで指定できます。

サーバーレベルの変更

CFusion¥lib ディレクトリにある neo-xmlrpc.xml の次のセクションを変更します。

```
<struct type='coldfusion.server.ConfigMap'>
    <var name='version'>
        <string>2</string></var>
</struct>
```

また、application.cfc で次のタグを変更して、Web サービスのバージョンを指定できます。

```
<cfset this.wssettings.version.publish="1|2">
```

アプリケーションレベルの変更

使用する Axis のバージョンは、アプリケーションレベルで次のように指定できます。

- **公開の場合：**<cfset this.wssettings.version.publish="2">

  注意： ここで指定したバージョンは、サーバーレベルで指定されているバージョンを上書きします。

- **使用の場合：**<cfset this.wssettings.version.consume="2">

アプリケーションレベルでの style 属性の設定

次のように設定します。<cfset this.wssettings.style="rpc/document/wrapped">

Web サービスでの ColdFusion の複合データ型の処理

アプリケーションレベルで includeCFTypesInWSDL 属性を次のように設定します。

<cfset this.wssettings.includeCFTypesInWSDL="true">

デフォルト値は true です。

true に設定すると、複合データ型用のスキーマが生成され、WSDL に組み込まれます。クライアントサイドでは、複合データ型用のスタブが生成されます。ユーザーはそれを引数として渡すことができます。

属性 includeCFTypesInWSDL を送信する必要がある状況を次に説明します。

```
<cffunction name="echoObject" access="remote" returnType="any">
            <cfargument name="argObj" type="any">
            <cfdump var="#argObj#" output="console">
            <cfreturn argObj>
    </cffunction>
```

Axis 2 は、CFC のメソッドのシグネチャを調べて、WSDL を生成します。この関数では、引数の型も戻り値の型も any です。したがって、WSDL で生成されるスキーマには、struct、query、xml などの ColdFusion の複合データ型のスキーマは含まれません。これらのスキーマが WSDL に存在しない場合、クライアントサイドでは、複合データ型のスタブクラスはなく、関数は役に立ちません。

同様に、次の関数では、引数の型も戻り値の型も struct です。この場合も、WSDL には struct のスキーマだけが含まれます。struct 内のドキュメントまたはクエリを渡したい場合、WSDL ではスキーマとして使用できません。

```
<cffunction name="echoComplexStruct" access="remote" returnType="struct">
            <cfargument name="argStr" type="struct">
            <cfdump var="#argStr#" output="console">
            <cfreturn argStr>
</cffunction>
```

コンポーネントレベルの変更

- 新しい属性 wsversion が cfcomponent に追加されています。

  2 を指定すると、CFC は Axis 2 エンジンを使用してデプロイされます。ここで指定する値は、アプリケーションレベルまたはサーバーレベルで指定されている値を上書きします。

- 新しい値 wrapped を属性 style に対して指定できます。wsversion を 2 に設定している場合はデフォルト値は wrapped で、1 の場合はデフォルト値は rpc です。

注意：style をアプリケーションレベルで設定した場合は、どちらの場合も、デフォルト値の代わりに、アプリケーションレベルの値が使用されます。

シンタックスは次のようになります。

```
<cfcomponent
    wsversion = 1|2>
    style = "rpc|document|wrapped"
    ....
    <cffunction ...>
        ...
    </cffunction>

    <cffunction ...>
        ...
    </cffunction>
</cfcomponent>
```

createObject および cfobject に対する変更

- Axis のバージョンを指定するための新しいプロパティ wsVersion が createObject に追加されました。
- Axis のバージョンを指定するための新しい属性 wsVersion が cfobject タグに追加されました。

ColdFusion 9 からの変更

複数の CFC で同じ表示名が使用されている場合、ColdFusion 9 ではすべての CFC が公開されます。一方、ColdFusion 10 では、wsVersion が 2 に設定されていると、複数の CFC の表示名が同じ場合、最初の CFC だけが公開されます。2 番目の CFC にアクセスしようとすると、エラーになります。

cfinvoke、cfobject または createObject を使用して Web サービスにアクセスすると、サービスは自動的に登録されません。ColdFusion Administrator で Web サービスを登録する必要があります（データとサービス／ Web サービス）。

制約

注意：次の制限事項は、ColdFusion 10 の将来のリリースで対処されます。

- Axis 1 とは異なり、Axis 2 では同じ表示名を持つ複数の CFC をサポートできません。
- cfinvoke タグの属性 Serviceaddress および portTypeName は、Axis 2 ではサポートされません。
- JWS ファイルを公開する場合は、Axis 1 のみがサポートされます。

ColdFusion での RESTful Web サービス

ColdFusion 10 では、REST（Representational State Transfer）サービスを作成して公開できます。クライアントは、HTTP/HTTPS リクエストを使用してこのサービスを使用できます。

REST とは

REST の概念については、次の URL で Java のチュートリアルを参照してください。

http://download.oracle.com/javaee/6/tutorial/doc/gijqy.html

REST と ColdFusion

cfcomponent、cffuntion、cfargument の各タグで特定の属性を定義することで REST サービスを作成し、REST リソースとして公開できます。

- **HTTP のリクエストレスポンスモデルを順守：**HTTP を媒体として使用するだけでなく、サービスでは HTTP のすべての基準に従うことができます。REST サービスとして公開されたコンポーネントは、HTTP/HTTPS リクエストを使用して利用できます。

  REST サービスは URI（Uniform Resource Identifier）によって識別され、Web ページからだけでなく、ブラウザーのアドレスバーで URI を指定してアクセスできます。

- **HTTP のすべてのメソッドをサポート：**REST 対応の CFC は、HTTP メソッド GET、POST、PUT、DELETE、HEAD および OPTIONS をサポートします。

- **シリアル化／シリアル化解除の暗黙の処理：**ColdFusion は JSON および XML のシリアル化／シリアル化解除をネイティブにサポートします。したがって、クライアントアプリケーションは HTTP/HTTPS リクエストを発行することによって REST サービスを使用できます。レスポンスは XML または JSON 形式にシリアル化できます。

- **Web サービスを REST サービスおよび WSDL サービスの双方で公開：**同じ ColdFusion コンポーネントを REST サービスおよび WSDL サービスとして作成し、公開できます。

REST Web サービスの作成

- ColdFusion コンポーネントまたはコンポーネント内の関数を REST リソースとして作成し公開できます。
- CFC を REST Web サービスとして作成するには、cfcomponent タグで restPath または rest を指定します。
- cffunction で、REST リソースとして公開する関数の access 属性を remote に設定します。

例

```
<cfcomponent rest="true" restpath="restService">
    <cffunction name="sayHello" access="remote" returntype="String" httpmethod="GET">
        <cfset rest = "Hello World">
        <cfreturn rest>
    </cffunction>
</cfcomponent>
```

REST サービスを使用するアプリケーションの登録

REST 対応にする CFC を作成した後、restInitAplication 関数を使用して、または ColdFusion Administrator で、Web サービスとして登録するためのフォルダーを指定します。

注意：ネストされた REST アプリケーションは登録できません。

ColdFusion Administrator の使用（データとサービス／ REST サービス）

フォルダーを指定すると、rest または restPath を指定した、そのフォルダーまたはサブフォルダー内のすべての CFC が登録されます。

1 ColdFusion が CFC を検索するアプリケーションパスまたはルートフォルダーを参照して選択します。

2 （オプション）「サービスマッピング」セクションで、アプリケーション名の代わりに仮想マッピングを指定します。

  フォルダーに Application.cfc およびアプリケーション名がある場合、サービスはアプリケーション名で識別されます。サービスマッピングを指定することでこれを上書きできます。その場合、サービスは指定したサービスマッピングで識別されます。フォルダーに Applicaiton.cfc がない場合は、サービスマッピングの指定は必須です。

  ColdFusion ルートの外部の REST フォルダーを指定する場合は、次のいずれかのマッピングを追加してフォルダーを登録します。

c:¥restfolder のマッピングを ¥map として追加済みであるものと考えます。

```
<cfset restinit("c:\restfolder","myapp")>
<cfset restinit("\map","myapp")>
```

3 （オプション）デフォルトの REST サービスとしてアプリケーションを指定します。1 つのサーバーインスタンスに対してデフォルトとして設定できるアプリケーションは 1 つだけです。デフォルトアプリケーションはいつでも変更できます。「デフォルトアプリケーションを設定する」を選択し、「サービスを追加」をクリックします。デフォルトとしてのサービスを削除するには、選択を解除します。

4 詳細を指定した後、「サービスを追加」をクリックして登録します。

「アクティブな ColdFusion REST サービス」セクションで、すべての登録済み Web サービスの詳細を指定します。

登録が済むと、すべての CFC は RESTful サービスとして公開されます。登録後に起動すると、登録されたサービスが自動的に公開されます。

注意：アプリケーションの REST 関連コンポーネントを変更したときは常に、アプリケーションを更新します。

Web サービスへのアクセス

- 次の例で示すように、Web サービスにアクセスするには cfhttp タグを使用します。

```
<cfhttp
    url="http://localhost:8500/rest/RestTest/restService"
    method="get"
    port="8500"
    result="res">
</cfhttp>
```

- ブラウザーで URL http://localhost:8500/rest/RestTest/restService を指定してアクセスします。

URL の解釈

例で指定した URL は次のように解釈できます。

URL のコンポーネント	説明
http://localhost:8500	ColdFusion サーバーの IP アドレスとポートを含むベース URL。 ColdFusion を JEE アプリケーションとしてデプロイする場合は、URL にコンテキストルートが含まれます（例：http://localhost:8500/**cfusion**）。
rest	送信されるリクエストが REST リクエストであることを示します。 このデフォルト値の名前を変更するには、cfusion/wwroot/WEB-INF にある web.xml でコンテキストパスを変更し、config¥wsconfig¥1 にある uriworkermap.properties ファイル内の同じマッピングを更新します。
restTest	ColdFusion Administrator でサービスを登録するときに使用したアプリケーション名またはサービスマッピングです。ColdFusion Administrator でサービスマッピングを指定していない場合、アプリケーション名は Application.cfc から取得されます。
restService	サービスで定義した REST パスです。つまり、cfcomponent タグの属性 restPath の値です。

accept ヘッダーの提供

accept ヘッダーを REST URL で提供できます。次にその例を示します。

- http://localhost:8500/rest/RestTest/restService.json という REST URL の場合、accept パラメーターは application¥json に設定されます。
- http://localhost:8500/rest/RestTest/restService.xml という REST URL の場合、accept パラメーターは application¥xml に設定されます。

HTTP コンテンツタイプのネゴシエーション

RESTful Web サービスによって返されるコンテンツタイプは、Accept HTTP ヘッダーに依存します。

クライアントでは、Accept リクエストヘッダーを次の順序で設定できます。

- 優先されるコンテンツタイプのカンマ区切りリスト。
- セミコロン（;）で区切った後に、0 ～ 1 の範囲で q=***[0-1]*** という形式の浮動小数点数。デフォルト値は 1 です。

  最小値の 0 はそのコンテンツタイプが受け付けられないことを示し、最大値の 1 は最高の優先度を示します。

2 つのタイプが同じ優先度の場合は、順番の優先度が考慮されます。

例

次の例では、Accept ヘッダーでの優先度を示す値が指定されていないので、クライアントは XML と JSON の両方の形式を使用できます。

```
GET http://adobe.com/stuff
Accept: application/xml, application/json
```

次の例のリクエストでは、コンテンツタイプを返す優先度が指定されています。

```
GET http://adobe.com/stuff
Accept: text/*;q=0.9, */;q=0.1, audio/mpeg, application/xml;q=0.5
```

コンテンツタイプの優先順位は次のようになります。

1. audio/mpeg（優先度が指定されていないため）
2. text/*（0.9 が最高の値です）
3. application/xml
4. */

次の例では、text/* と audio/mpeg に同じ優先度を指定していますが、順番により text/* が優先されます。

```
GET http://adobe.com/stuff
Accept: text/*;q=0.9, */;q=0.1, audio/mpeg=0.9, application/xml;q=0.5
```

サブリソースの指定

REST サービスで使用できる関数は、リソース関数、サブリソース関数またはサブリソースロケーターです。

リソース関数

関数レベルでは restPath を指定していないが、httpMethod を指定している関数です。この場合、CFC の URL にアクセスすると、リソース関数が呼び出されます。

サブリソース関数

restPath と httpMethod の両方を指定してある関数です。

サブリソース関数は、CFC によって作成されるリソースのサブリソースを作成するために使用します。サブリソースの cffunction に httpMethod 属性がある場合、次の例で示すように、関数は HTTP リクエストを直接処理できます。

例

Employee.cfc

```
<cfcomponent rest="true" restPath="/hello">
       <cffunction name="sayHello" access="remote" returnType="String" httpMethod="GET" restPath="name">
---------
    </cffunction>
</cfcomponent>
```

この例では、httpMethod と restPath が定義されています。***baseurl***/hello/name は、URL ***baseurl***/hello に対するサブリソースです。

サブリソースロケーター

httpMethod 属性を指定せず、restPath を指定した場合は、リクエストを処理するコンポーネントを動的に解決できます。返されるオブジェクトはリソースクラスのインスタンスとして扱われ、リクエストを処理するため、またはリクエストを処理するオブジェクトをさらに解決するために、使用されます。

例

この例では、StudentService.cfc と Student.cfc が REST リソースです。StudentService.cfc では、getStudent 関数がコンポーネントを返します。関数では、Student のオブジェクトが作成され、値 name と age が設定されます。戻り値の型でオブジェクトが指定されていると、そのオブジェクトで定義されている、リクエストされた httpmethod が設定されている関数が呼び出されます。

StudentService.cfc は、restPath が指定されているので、直接呼び出すことができます。StudentService.cfc の関数 getStudent はサブリソースロケーターとして機能し、Student.cfc を返すので、Student.cfc は StudentService.cfc を介してのみ呼び出すことができます。

StudentService.cfc：

```
<cfcomponent rest="true" restpath="/studentService">

    <cffunction name="addStudent" access="remote" returntype="void" httpmethod="POST">
        <cfargument name="name" type="string" required="yes" restargsource="Form"/>
        <cfargument name="age" type="numeric" required="yes" restargsource="Form"/>
            <!--- Adding the student to data base. --->
    </cffunction>
        <cffunction name="getStudent" access="remote" returntype="student" restpath="{name}-{age}">
        <cfargument name="name" type="string" required="yes" restargsource="Path"/>
        <cfargument name="age" type="string" required="yes" restargsource="Path"/>
            <!--- Create a student object and return the object. This object will handle the request now.
--->
        <cfset var myobj = createObject("component", "student")>
        <cfset myobj.name = name>
        <cfset myobj.age = age>
        <cfreturn myobj>
    </cffunction>
</cfcomponent>
```

Student.cfc

```
<cfcomponent>
    <cfproperty name="name" type="string"/>
    <cfproperty name="age" type="numeric"/>
    <!--- Getting the student detail. --->
    <cffunction name="getStudent" access="remote" returntype="String" httpmethod="GET"
produces="text/xml">
        <!--- Retrieve the Student from the DB. --->
        <!--- get student from db where name and age matches --->
        <cfset st.name = "Thomas">
        <cfset st.age = "25">
        <cfset st.surname = "Paul">
        <cfset str = "<student><name>" & st.name & "</name><age>" & st.age & "</age><surname>" & st.surname
& "</surname></student>">
        <cfreturn str>
    </cffunction>
    <!--- Updating the student detail. --->
    <cffunction name="updateStudent" access="remote" returntype="void" httpmethod="PUT">
        <!--- Retrieve the Student from the DB. --->
        <!--- Update the student in DB. --->
        <cfset st.name = name>
        <cfset st.age = age>
    </cffunction>
    <!--- Deleting the student. --->
    <cffunction name="deleteStudent" access="remote" returntype="String" httpmethod="DELETE">
        <!--- Delete the Student from the DB. --->
        <!---<cfset st = deleteStudentFromDB(name)>--->
        <cfreturn "Student deleted">
    </cffunction>
</cfcomponent>
```

Student.cfm

```
<!--- adding the student --->
<cfhttp url="http://localhost:8500/rest/RestTest/studentService" method="post" result="res">
<cfhttpparam type="formfield" name="name" value="Thomas" >
<cfhttpparam type="formfield" name="age" value="25" >
</cfhttp>
<cfoutput>Student Added</cfoutput>
</br>
</br>
<!--- fetching the details --->
<cfhttp url="http://localhost:8500/rest/RestTest/studentService/Thomas-25" method="get" result="res">
</cfhttp>
<cfoutput>#xmlformat(res.filecontent)#</cfoutput>
</br>
</br>
<!--- updating the student details --->
<cfhttp url="http://localhost:8500/rest/RestTest/studentService/Thomas-25" method="put" result="res">
</cfhttp>
<cfoutput>Updated the student</cfoutput>
</br>
</br>
<!--- deleting the student --->
<cfhttp url="http://localhost:8500/rest/RestTest/studentService/Thomas-25" method="delete" result="res">
</cfhttp>
<cfoutput>Student Deleted</cfoutput>
```

HTTP レスポンス

デフォルトでは、ColdFusion は HTTP の成功および失敗のレスポンスを次のようにクライアントに提供します。

成功レスポンス

デフォルトのレスポンス	説明
200 OK	レスポンスに本文がある場合に送信されます。
204 No Content	レスポンスに本文がない場合に送信されます。

エラーレスポンス

デフォルトのレスポンス	説明
404 Not found	リクエストの URL が無効です。
406 Not Acceptable	REST サービスのどの関数も、クライアントによってリクエストされた MIME タイプを生成できません。
415 Unsupported Media Type Error	リソースはクライアントリクエストの MIME タイプを使用できません。
405 Method not allowed	リクエストの HTTP メソッドがバインドされていない有効な URI でクライアントが HTTP メソッドを呼び出した場合です。 クライアントが HTTP の HEAD メソッドおよび OPTIONS メソッドをリクエストした場合は、このエラーは発生しません。 その特定の URI に対する HEAD リクエストに対応できるリソースメソッドはないが、GET を処理できるメソッドは存在する場合は、それが呼び出されて、リクエスト本文なしでそのメソッドからレスポンスが返されます。 OPTIONS を処理できる既存のメソッドがない場合、自動的に生成された意味のあるレスポンスが、Allow ヘッダーを設定されて送信されます。

カスタムレスポンス

ColdFusion で使用できるデフォルトのレスポンスに加えて、カスタムレスポンスを設定できます。

例えば、成功レスポンスの 201 Created を提供する必要があるものとします。このようなデフォルトのレスポンスはありません。送信できるのは 200 OK または 204 No Content だけです。

このような場合は、次のいずれかの方法でカスタムレスポンスを作成できます。

restSetResponse を使用したカスタム成功レスポンスの送信

1 CFC を REST サービスとして定義する場合は、cffunction（カスタムレスポンスを送信する関数）で returnType を void に設定します。

次に例を示します。

```
<cffunction name="create" httpMethod="POST" produces="application/xml" returnType="void">
    <cffunction>
```

2 次の例で示すように、cffunction で送信するカスタムレスポンスの構造体を作成します。

```
<cfset response=structNew()>
<cfset response.status=201>
<cfset response.content="<customer id="&id&"><name>"&name&"</name></customer>">        <cfset
response.headers=structNew()>
<cfset response.headers.location="http://localhost:8500/rest/CustomerService/customers/123">
```

この例では、レスポンスのステータスとして 201、および送信するコンテンツを設定しています。例えば、顧客の詳細および POST リクエストのレスポンスを作成した場所です。

注意：カスタムレスポンスでステータスを指定しないと、レスポンスのステータスとして 500 Internal server error が送信されます。

3 restSetResponse 関数は次のように使用します。

```
restSetResponse( response );
```

cfthrow を使用したカスタムエラーレスポンスの送信

カスタムエラーレスポンスを送信するものとします。例えば、次のような場合を考えてみます。

method.service.cfc

```
<cffunction name="getCustomer" httpMethod="GET" produces="application/xml" restPath="{id}"
return="string">
    <cfargument name="id" type="numeric" argtype="PathParam">
<!--- Getting the customer. ---><cffunction>
```

この場合、顧客データベースがあり、/customers/123 で GET HTTP リクエストを行っています。

しかし、指定された ID 123 の顧客は見つかりません。そこで、クライアントに 404 Resource Not Found レスポンスを返送します。このレスポンスはデフォルトでは不可能です。

この場合、次のように cfthrow を使用してカスタムエラーレスポンスを送信できます。

```
<cfthrow type="RestError" errorcode="404">
```

Application.cfc の変更

REST サポート用に Application.cfc が次のように拡張されました。

変数	説明
this.restsettings.cfclocation	特定の場所においてのみ CFC を公開するため、REST CFC が配置されるディレクトリのカンマ区切りリストを提供します。ディレクトリパスは絶対でも相対でもかまいません。 設定しないと、アプリケーションルートからのすべての CFC が公開されます。
this.restsettings.skipCFCWithError	エラーが発生した場合、例外の原因になった CFC を無視して公開を続けます。 true の場合、エラーのある CFC は無視され、残りの CFC が公開されます。デフォルトでは false です。 false に設定すると、エラーの場合は、アプリケーション自体が公開されません。ただし、登録された他のアプリケーションは公開されます。 アプリケーションの起動中にエラーが発生した場合は、エラーがコンソールに出力されます。 問題を記録するために、アプリケーションごとに個別にログファイルがあります。

REST Web サービスの拡張

RESTful CFC を拡張するときは、次の条件が適用されます。

- 基本 CFC の関数に対して、REST 属性を定義できます。したがって、基本 CFC を拡張するすべての CFC が、REST 属性を継承します。

  次の例では、CustomerResource.cfc が BaseCustomerResource.cfc を拡張しています。

  BaseCustomerResource.cfc

```
<cfcomponent>
    <cffunction name="SayPlainHello" access="remote" produces="text/plain" returntype="string"
httpmethod="POST">
        <cfreturn "BaseCustomerResource Plain">
    </cffunction>
    <cffunction name="SayXMLHello" access="remote" produces="text/xml,application/xml"
returntype="string" httpmethod="POST">
        <cfreturn "BaseCustomerResource XML">
    </cffunction>
</cfcomponent>
```

BaseCustomerResource.cfc には、CFC 内の関数に適用されるすべての REST 属性があります。BaseCustomerResource.cfc を定義した後、BaseCustomerResource.cfc を拡張する CustomerService.cfc を次のように定義できます。

Customerservice.cfc

```
<cfcomponent rest="true" restpath="/customerservice" extends="BaseCustomerResource">

    <cffunction name="SayPlainHello" access="remote" returntype="string">

        <!--- Implement the method. --->
        <cfreturn "CustomerResource Plain">
    </cffunction>

    <cffunction name="SayXMLHello" access="remote" returntype="string">

        <!--- Implement the method. --->
        <cfreturn "CustomerResource XML">
    </cffunction>

</cfcomponent>
```

BaseCustomerResource.cfm

```
<cfhttp url="http://localhost:8500/rest/RestTest/customerservice" method="post" result="res">
<cfhttpparam type="header" name="accept" value="text/plain" >
</cfhttp>
<cfoutput>#res.filecontent#</cfoutput>
</br>
</br>
<cfhttp url="http://localhost:8500/rest/RestTest/customerservice" method="post" result="res">
<cfhttpparam type="header" name="accept" value="application/xml" >
</cfhttp>
<cfoutput>#res.filecontent#</cfoutput>
</br>
</br>
```

rest および restPath 以外の REST 属性は、CustomerService.cfc 内では必要ありません。

- RESTful CFC を継承するときは、拡張する CFC で使用されている属性の上書きを選択できます。

  例えば、次の CFC は BaseCustomerResource.CFC を拡張しますが、関数 SayPlainHello は基本 CFC の関数を上書きしています。

```
<cfcomponent rest="true" restPath="/customerservice" extends="BaseCustomerResource">
    <cffunction name="SayPlainHello" access="remote" produces="text/plain" returntype="string"
httpmethod="PUT">
        <cfargument name="username" type="string" argtype="pathparam">
        <!--- Implement the method. --->
    </cffunction>


    <cffunction name="SayXMLHello" access="remote" returntype="string">
        <cfargument name="username" type="string">
        <!--- Implement the method. --->
    </cffuntion>
</cfcomponent>
```

注意：関数内の 1 つの属性だけを上書きする場合でも、すべての REST 属性を再指定する必要があります。

- REST サービスとして使用するには、継承された CFC で REST 属性（rest ／ restPath）を指定する必要があります。基本 CFC で定義するだけでは機能しません。

REST サービスとデータ交換形式

ColdFusion の REST サービスは、Web 経由で送信できるように、XML または JSON 形式で表すことができます。サービスで使用されている ColdFusion のデータ型の XML または JSON へのシリアル化／シリアル化解除は、ColdFusion が暗黙で行います。

REST 対応の関数は、ColdFusion のデータ型であるクエリ、構造体、CFC 型、配列、CFC の配列、XML、文字列、数値、ブール、日付、バイナリ（XML の場合）を、引数として受け取ることができます。

ColdFusion は、データを XML および JSON の定義済み形式にシリアル化します。同様に、ColdFusion は、本文が ColdFusion によって定義されている形式である場合にのみ、本文をシリアル化解除します。

REST サービスの XML シリアル化

シリアル化の仕様

- リクエストの Accept ヘッダーは、text/xml または application/xml である必要があります。
- 必要な MIME タイプを生成できる関数がサービス内に必要です。
- 関数は、ColdFusion がサポートするいずれかのデータ型を返す必要があります。
- 循環配列はサポートされません。公開されたシリアル化済み文字列を表示できますが、次の例で説明するように意図した出力になりません。

```
<cfset this.arr1 = arrayNew(1)>
<cfset this.arr1[1] = "1">
<cfset this.arr1[2] = this.arr1>
<cfset this.arr1[3] = "3">
```

変数に配列を代入するとき（この場合は <cfset this.arr1[2] = this.arr1>）、arr1 を arr1 の 2 番目のインデックスの項目として代入します。ColdFusion は暗黙で新しい配列を作成し、arr1 から新しく作成した配列にデータをコピーします。新しく作成された配列は、arr1 の 2 番目のインデックスに代入されます。したがって、両方のインスタンスは異なり、循環依存関係が影響を受けます。

シリアル化すると、次のような出力が得られます。

```
<array id="1" size="3">
<item index="1" type="string">1</item>
<item index="2" type="array">
<array id="2" size="1">
<item index="1" type="string">1</item>
</array>
</item>
<item index="3" type="string">3</item>
</array>
```

見るとわかるように、内側の配列は最初の要素の後で切り捨てられています。

シリアル化解除の仕様

- リクエストのコンテンツは、ColdFusion で指定されている定義済みの形式である必要があります（詳しくは、「形式の定義」のセクションを参照）。
- リクエストのコンテンツタイプは、text/xml または application/xml である必要があります。
- リクエストの MIME タイプを使用できる関数がサービス内に必要です。
- cfargument では属性 restArgSource および restArgName を指定できません。つまり、リクエストの本文のデータのみを送信できます。
- cfargument 型は ColdFusion がサポートするデータ型である必要があります。
- restArgSource 属性を指定しない引数は 1 つしか存在できません。リクエストの本文全体が、引数の型にシリアル化解除されます。
- 循環配列はサポートされません。

形式の定義

クエリー

- XML のルート要素は query である必要があります。
- 循環依存関係を処理するには、ID 属性を使用します。
- 有効な属性値は、文字列、日付、ブール、数値、ドキュメント、クエリ、構造体、配列、CFC（値が CFC インスタンスの場合）です。

例

次の例では、ルート query 要素には 2 つの子要素 columnnames と rows があります。columnnames はクエリの列の名前です。rows は複数の row 要素のデータを持つことができます。ここで、行の列の順序は、columnames で定義されている列と一致する必要があります。行内の列ごとに type 属性が必須です。type 属性の値には、ColdFusion の任意のデータ型を設定できます。

```
<query id="1">
    <columnnames>
        <column name="columnnameone"/>
        <column name="columnnametwo"/>
    </columnnames>
    <rows>
        <row>
            <column type="string">value one</column>
            <column type="string">value two</column>
        </row>
        <row>
            <column type="number">444.0</column>
            <column type="string">value four</column>
        </row>
    </rows>
</query>
```

構造体

- XML の root 要素は struct である必要があります。
- 循環依存関係を処理するには、ID 属性を使用します。
- struct は複数の entry 子要素を持つことができます。つまり、シリアル化される Struct インスタンス内のキーと値のペアです。entry 要素には 2 つの必須属性 name（エントリの名前）と type（エントリの値の型）が必要です。type 属性の値には、ColdFusion のいずれかのデータ型を設定できます。

例

```
<struct id="1">
    <entry name="name" type="string">joe</entry>
    <entry name="age" type="string">30</entry>
    <entry name="address" type="string">101 some drive, pleasant town, 90010</entry>
    <entry name="eyecolor" type="string">blue</entry>
    <entry name="income" type="string">50000</entry>
</struct>
```

CFC コンポーネント

- XML のルート要素は component である必要があります。
- 循環依存関係を処理するには、ID 属性を使用します。
- name 属性は、CFC の Web ルートからの完全修飾名を提供する必要があります。
- component 要素は、複数の property 子要素を含むことができます。
- property 要素は、cfcomponent で定義されている cfproperty 値です。
- property の name 属性は、定義されている cfproperty の名前に対応します。
- type 属性は cfproperty の型を指定します。
- コンポーネント内のいずれかのプロパティが null 値を持つ場合、そのプロパティはシリアル化された形式には含まれません。これはシリアル化解除にも適用されます。

例

Student.cfc

```
<component id="1" name="testrest.student">
    <property name="name" type="string">paul</property>
    <property name="age" type="number">444.0</property>
    <property name="dob" type="date">478377000000</property>
</component>
```

次の例で、testrest.student は、CFC Student.cfc が webroot の下の testrest ディレクトリに配置されることを意味します。

配列および cfcomponent の配列の形式

- ルート要素は array である必要があります。
- size 属性は配列の長さを指定します。
- type 属性は、array 要素の場合は必須ではありませんが、item 要素の場合は必須です。
- 配列が cfcomponent 配列である場合、array 要素の type 属性は CFC の完全修飾名である必要があります。
- array 要素には複数の item 子要素が含まれます。配列内の非 null 要素だけがシリアル化されます。
- item 要素には、配列内での要素のインデックスを指定する index 属性と type 属性があります。

例

次の例では、2 つの struct オブジェクトを含む配列を示します。

```
<array id="1" size="2">
    <item index="0" type="struct">
        <struct id="2">
            <entry name="name" type="string">joe</entry>
            <entry name="age" type="string">30</entry>
            <entry name="address" type="string">101 some drive, pleasant town, 90010</entry>
            <entry name="eyecolor" type="string">blue</entry>
            <entry name="income" type="string">50000</entry>
        </struct>
    </item>
    <item index="1" type="struct">
        <struct id="3">
            <entry name="name" type="string">paul</entry>
            <entry name="age" type="string">25</entry>
            <entry name="address" type="string">some other address</entry>
            <entry name="eyecolor" type="string">black</entry>
            <entry name="income" type="string">40000</entry>
        </struct>
    </item>
</array>
```

例：CFC の配列のシリアル化

次の例では、CFC の配列の XML および JSON 形式へのシリアル化を示します。

1 arrayCFCdefinition.cfc の配列を作成します。

```
<cfcomponent>
    <cfproperty name="str" type="string"/>
</cfcomponent>
```

2 arrayCFC.cfc が、必要な配列を次のように生成します。

```
<cfcomponent restpath="arrayOfCFC">
    <cffunction name="func1" access="remote" output="false" returntype="arrayCFCdefinition[]"
                httpmethod="get" produces="text/xml">
        <cfset arrCFC = arraynew(1)>
        <cfloop from=1 to=2 index="i">
            <cfset obj = createObject("component", "arrayCFCdefinition")>
            <cfset obj.str = i>
            <cfset arrayAppend(arrCFC, obj)>
        </cfloop>
        <cfreturn arrCFC>
    </cffunction>
    <cffunction name="func2" access="remote" output="false" returntype="arrayCFCdefinition[]"
                httpmethod="get" produces="text/json">
        <cfset arrCFC = arraynew(1)>
        <cfloop from=1 to=2 index="i">
            <cfset obj = createObject("component", "arrayCFCdefinition")>
            <cfset obj.str = i>
            <cfset arrayAppend(arrCFC, obj)>
        </cfloop>
        <cfreturn arrCFC>
    </cffunction>
</cfcomponent>
```

3 次のようにして、リソースにアクセスします。

- XML の場合

```
<cfhttp url="http://127.0.0.1:8500/rest/RestTest/arrayOfCFC" method="get" result="res1">
    <cfhttpparam type="header" name="accept" value="text/xml">
</cfhttp>
```

- JSON の場合

```
<cfhttp url="http://127.0.0.1:8500/rest/RestTest/arrayOfCFC" method="get" result="res2">
    <cfhttpparam type="header" name="accept" value="text/json">
</cfhttp>
```

4 レスポンスとして次のようなシリアル化された出力を受け取ります。

- XML の場合

```
<array id="1" size="2" type="cfsuite.restservices.restservices.new.ArrayCFCdefinition">
    <item index="1" type="COMPONENT">
        <component id="2" name="cfsuite.restservices.RESTServices.New.arrayCFCdefinition">
            <property name="STR" type="NUMBER">
                1.0
            </property>
        </component>
    </item>
    <item index="2" type="COMPONENT">
        <component id="3" name="cfsuite.restservices.RESTServices.New.arrayCFCdefinition">
            <property name="STR" type="NUMBER">
                2.0
            </property>
        </component>
    </item>
</array>
```

- JSON の場合

```
[{"Str":1},{"Str":2}]
```

例：CFC の配列：シリアル化解除

次の例では、CFC の配列の XML 形式からのシリアル化解除を示します。

注意：CFC の配列のシリアル化解除は、JSON についてはサポートされていません。

1 arrayCFCdefinition.cfc の配列を作成します。

```
<cfcomponent>
    <cfproperty name="str" type="string"/>
    <cffunction name="check" returntype="any">
        <cfreturn this.str>
    </cffunction>
</cfcomponent>
```

2 arrayCFC.cfc が、必要な配列を次のように生成します。

```
<cfcomponent>
    <cffunction name="func3" access="remote" output="false" returntype="string"
                httpmethod="put" produces="text/xml" consumes="text/xml">
        <cfargument name="arg" type="arrayCFCdefinition[]"/>

        <cfif arg[2].check() eq "2">
            <cfreturn "true">
        <cfelse>
            <cfreturn "false">
        </cfif>
    </cffunction>
</cfcomponent>
```

3 次のようにして、XML のリソースにアクセスします。

```
<cfhttp url="http://127.0.0.1:8500/rest/RestTest/arrayOfCFC" method="put" result="res3">
    <cfhttpparam type="header" name="content-type" value="text/xml">
    <cfhttpparam type="header" name="accept" value="text/xml">
    <cfhttpparam type="body" value="<ARRAY ID=""1"" SIZE=""2""
TYPE=""cfsuite.restservices.restservices.new.ArrayCFCdefinition""><ITEM INDEX=""1""
TYPE=""COMPONENT""><COMPONENT ID=""2""
NAME=""cfsuite.restservices.restservices.new.ArrayCFCdefinition""><PROPERTY NAME=""STR""
TYPE=""NUMBER"">1.0</PROPERTY></COMPONENT></ITEM><ITEM INDEX=""2"" TYPE=""COMPONENT""><COMPONENT
ID=""3"" NAME=""cfsuite.restservices.restservices.new.ArrayCFCdefinition""><PROPERTY NAME=""STR""
TYPE=""NUMBER"">2.0</PROPERTY></COMPONENT></ITEM></ARRAY>">
</cfhttp>
```

4 関数を参照します。

配列の 2 番目のインデックスの arrayCFC definition.cfc のプロパティの値を確認します。

文字列、ブール、数値、バイナリ、日付

リクエストの本文で値を直接指定します。

循環依存関係の処理

ColdFusion では、循環依存関係は ID 参照を使用して処理します。すべての ColdFusion 複合データ型には、シリアル化時に一意の ID があります。同じオブジェクトを別の場所でシリアル化する必要がある場合は、再度オブジェクトをシリアル化する代わりに、ID を使用して既にシリアル化されているデータが参照されます。

次の例では、メインのオブジェクトは構造体です。構造体にはオブジェクトの配列が含まれます。配列には 2 つの要素があり、どちらの要素も構造体の同じインスタンスです。

シリアル化の際に、配列の最初の要素はそのままシリアル化されます。シリアル化された構造体の ID は 2 です。このオブジェクトは既にシリアル化されているので、2 番目の要素をシリアル化する代わりに、IDREF 属性を使用して既にシリアル化されている構造体インスタンスを参照します。

```
<struct id="1">
    <entry name="arrayinastruct" type="array">
        <array id="2" size="2">
            <item index="0" type="struct">
                <struct id="3">
                   <entry name="name" type="string">joe</entry>
                    <entry name="age" type="string">30</entry>
                    <entry name="address" type="string">101 some drive, pleasant town, 90010</entry>
                    <entry name="eyecolor" type="string">blue</entry>
                    <entry name="income" type="string">50000</entry>
                </struct>
            </item>
            <item index="1" type="struct">
                <struct idref="3"/>
            </item>
        </array>
    </entry>
</struct>
```

注意：シリアル化解除のときにも ColdFusion によってオブジェクト参照が処理されます。

JSON のシリアル化と REST サービス

シリアル化の仕様

- リクエストの Accept ヘッダーのコンテンツタイプは、text/JSON、application/JSON または text/plain である必要があります。
- REST サービスには、必要な MIME タイプを処理できる関数が必要です。
- 関数は、ColdFusion がサポートするバイナリ以外のいずれかのデータ型を返す必要があります。
- 循環の動作はサポートされていません。ただし、配列の場合は、公開されたシリアル化済み文字列を表示できますが、次の例で説明するように意図した出力になりません。

  ```
  <cfset this.arr1 = arrayNew(1)>
  <cfset this.arr1[1] = "1">
  <cfset this.arr1[2] = this.arr1>
  <cfset this.arr1[3] = "3">
  ```

  変数に配列を代入するとき（この場合は <cfset this.arr1[2] = this.arr1>）、arr1 を arr1 の 2 番目のインデックスの項目として代入します。ColdFusion は暗黙で新しい配列を作成し、arr1 から新しく作成した配列にデータをコピーします。新しく作成された配列は、arr1 の 2 番目のインデックスに代入されます。したがって、両方のインスタンスは異なり、循環依存関係が影響を受けます。

  シリアル化すると、次のような出力が得られます。

  ```
  [1,[1],3]
  ```

  見るとわかるように、内側の配列は最初の要素の後で切り捨てられています。

シリアル化解除の仕様

- リクエストのコンテンツは、ColdFusion によって指定されている定義済みの形式です。
- リクエストのコンテンツタイプは、text/JSON、application/JSON または text/plain です。
- サービスの関数は、リクエストの MIME タイプを使用します。
- cfargument では、属性 restargsource および restargname は指定されていません。
- cfargument の型は、ColdFusion がサポートするバイナリまたは CFC 定義以外のデータ型です。
- restArgSource 属性が指定されない引数は 1 つだけです。リクエストの本文全体が、引数の型にシリアル化解除されます。

- 循環の動作はサポートされていません。ただし、循環配列の場合は、公開されたシリアル化解除済み文字列を表示できますが、意図した出力になりません。

形式の定義

クエリの形式

```
{'COLUMNS':['columnNameOne','columnNameTwo'],'Data':[['value one','value two'],['444.0','value four']]}
```

構造体の形式

```
{'NAME':'joe','AGE':30,'ADDRESS':'101 Some Drive, Pleasant Town, 90010','EYECOLOR':'blue','INCOME':50000}
```

コンポーネントの形式

```
{'NAME':'Paul','AGE':444.0,'DOB':'July, 27 2011 00:00:00'}
```

注意：コンポーネントのシリアル化解除はサポートされません。

配列の形式

```
[{'NAME':'joe','AGE':30,'ADDRESS':'101 Some Drive, Pleasant Town,
90010','EYECOLOR':'blue','INCOME':50000},{'NAME':'paul','AGE':25,'ADDRESS':'Some other
address','EYECOLOR':'black','INCOME':40000}]
```

文字列、ブール、数値、日付

リクエストの本文で値を直接指定します。

CFML アプリケーションへの J2EE および Java 要素の統合

JSP ページやサーブレットなどの J2EE 要素、JSP タグ、EJB (Enterprise JavaBeans) などの Java オブジェクトを、Adobe ColdFusion アプリケーションに統合することができます。

ColdFusion、Java、および J2EE について

ColdFusion は、J2EE 準拠の Java テクノロジープラットフォームで構築されています。この構造により、ColdFusion アプリケーションで J2EE 要素を利用したり、J2EE 要素にアプリケーションを統合することが可能になっています。
ColdFusion ページでは、次のことを行えます。

- JavaScript およびクライアントサイド Java アプレットをページに含める
- JSP タグを使用する
- JSP ページとの相互運用を行う
- Java サーブレットを使用する
- JavaBeans や Enterprise JavaBeans などの Java オブジェクトを使用する

ColdFusion と、クライアントサイドの JavaScript およびアプレットについて

ColdFusion ページには、HTML ページのように、クライアントサイド JavaScript および Java アプレットを組み込むことができます。JavaScript を使用するには、HTML ページの場合と同様に、JavaScript コードを記述します。JavaScript は ColdFusion で無視され、クライアントに送信されます。

Java のクライアントサイドアプレットを使用する場合は、cfapplet タグを利用すると便利です。

ColdFusion ページでのアプレットの使用

1 ColdFusion Administrator の [Java アプレット拡張機能] ページでアプレットの .class ファイルを登録します。アプレットの登録の詳細については、ColdFusion Administrator のオンラインヘルプを参照してください。

2 アプレットを呼び出すには、cfapplet タグを使用します。appletSource 属性には、ColdFusion Administrator に割り当てたアプレット名を指定する必要があります。

たとえば ColdFusion には、あるテキストボックスから他のテキストボックスにテキストをコピーする Copytext というサンプルアプレットが用意されています。このアプレットは、ColdFusion のセットアップ時に自動的に Administrator に登録されます。このアプレットを使用するには、ページにこのアプレットを組み込みます。次に例を示します。

```
<cfform action = "copytext.cfm">
    <cfapplet appletsource = "copytext" name = "copytext">
</cfform>
```

ColdFusion と JSP について

ColdFusion では、次の方法で JSP タグとページがサポートされています。

- JSP ページとの相互運用 : ColdFusion ページは、JSP ページをインクルードすることや、JSP ページに転送することができます。JSP ページは、ColdFusion ページをインクルードすることや、ColdFusion ページに転送することができます。これらのページは、永続スコープでデータを共有できます。
- JSP タグライブラリのインポートおよび使用 : cfimport タグを使用して JSP タグライブラリをインポートし、それらのタグを使用することができます。

ただし、ColdFusion ページは JSP ページではありません。ColdFusion ページでは、ほとんどの JSP シンタックスを使用できません。特に、次の機能は ColdFusion ページでは使用できません。

Include、Taglib、および Page ディレクティブ : 代わりに、タグライブラリをインポートするには CFML の import タグを使用します。また、ページをインクルードするには、ColdFusion の GetPageContext 関数によって返されるページコンテキストオブジェクトの include (または forward) メソッドを使用します。詳細については、1112 ページの「JSP タグおよびタグライブラリの使用」および 1113 ページの「JSP ページおよびサーブレットとの相互運用」を参照してください。

式、宣言、およびスクリプトレット JSP スクリプト要素 : 代わりに、CFML 要素および CFML 式を使用します。

JSP コメント : 代わりに、CFML コメントを使用します。ColdFusion では、JSP コメントは無視されてブラウザに渡されます。

標準 JSP タグ : jsp:plugin などがあります。J2EE サーバーが jar ファイルのこれらのタグへのアクセスを許可している場合を除きます。代わりに、ColdFusion タグおよび PageContext オブジェクトを使用します。

ColdFusion とサーブレットについて

Java サーブレットの一部は JSP ページではなく、Java プログラムです。ColdFusion アプリケーションに JSP サーブレットを組み込むことができます。たとえば、特定のビジネスロジックを実行する既存のサーブレットがある場合、そのサーブレットを使用するには、ColdFusion ページで ColdFusion の GetPageContext 関数を使用してそのサーブレットを指定します。

GetPageContext 関数を使用してサーブレットにアクセスすると、ColdFusion ページとサーブレット間で Request、Session、および Application スコープが共有されるので、これらのスコープを共有データに使用することができます。

ColdFusion ページでは、cfservlet タグを使用してサーブレットにアクセスできます。また、form タグでサーブレット URL を使用したり、servlet タグを使用する SHTML ページにアクセスすることもできます。

注意 : JRun サーバー上のサーブレットにアクセスできる cfservlet タグは、ColdFusion MX 以降は非推奨になっています。

ColdFusion と Java オブジェクトについて

Java オブジェクトには、次のものがあります。

- J2EE API を構成する標準の Java クラスおよびメソッド
- 次のような、カスタマイズされた Java オブジェクト
  - JavaBeans などのカスタムクラス
  - Enterprise JavaBeans

ColdFusion ページでは、cfobject タグを使用して Java オブジェクトにアクセスします。

ColdFusion では、次の順序でオブジェクトが検索されます。

1. ColdFusion の Java の動的な Class ロードディレクトリ
2. "<Web のルートディレクトリ >/WEB-INF/lib" の Java アーカイブ (.jar) ファイル
3. "<Web のルートディレクトリ >/WEB-INF/classes" のクラス (.class) ファイル

ColdFusion は、次のセクション「クラスのロードについて」で説明するように、これらのディレクトリからクラスをリロードします。

1. ColdFusion Administrator の JVM および Java の [設定] ページで指定されたクラスパス
2. デフォルトの JVM クラスパス

クラスのロードについて

"<Web のルートディレクトリ >/WEB-INF/classes" ディレクトリに含まれている .class ファイルであるクラスや、"<Web のルートディレクトリ >/WEB-INF/lib" ディレクトリに含まれている JAR ファイルであるクラスは、動的にロードされます。これらのディレクトリで定義されているクラスは、メモリ内に既にオブジェクトが格納されている場合でも、ファイルのタイムスタンプが確認されます。ファイルがメモリ内のクラスよりも新しい場合は、そのディレクトリからクラスがロードされます。

この機能を使用するには、変更する Java 実装クラスが JVM クラスパスにないことを確認する必要があります。

クラスの自動ロードを無効にするには、クラスを JVM クラスパスに配置します。JVM クラスパスに配置されているクラスは、サーバーの稼動期間に 1 度だけロードされます。これらのクラスをリロードするには、ColdFusion を停止して、再起動します。

GetPageContext および PageContext オブジェクトについて

ColdFusion ページは J2EE サーブレットページなので、すべての ColdFusion ページには、ベースとなる Java の PageContext オブジェクトがあります。CFML には、GetPageContext 関数が含まれており、ColdFusion ページで使用することができます。

PageContext オブジェクトには、J2EE の統合に便利なフィールドやメソッドがいくつか用意されています。特に、標準の JSP タグと同じ機能を提供する include および forward メソッドが含まれています。

PageContext オブジェクトの他の機能の詳細については、Java のマニュアルを参照してください。このクラスの Javadoc は、http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/jsp/PageContext.html で参照できます。

CFML 変数と Java 変数について

ColdFusion 変数では大文字と小文字が区別されず、Java 変数では大文字と小文字が区別されるので、変数名には注意が必要です。ColdFusion と Java コード (Java ページやサーブレットを含む) の間でデータを共有するときは、次のルールとガイドラインに従ってください。

ルール

- 大文字と小文字が混在する変数を使用する場合、すべての変数名は大文字と小文字に関係なく一意であることが必要です。たとえば、MyVariable と MYVARIABLE という 2 つの Java 変数は使用しないでください。ColdFusion では、この 2 つは区別されません。
- CFML ページと JSP ページまたはサーブレットの間で Request スコープ変数を共有する場合、JSP ページまたはサーブレットでは、共有するすべての Request スコープ変数名を小文字で記述することが必要です。大文字と小文字が混在している場合や、すべて大文字である場合は、CFML がこれらの変数を参照すると null ポインタ例外が発生します。
- CFML ページと JSP ページまたはサーブレットの間で Application スコープ変数または Session スコープ変数を共有し、名前の付いた ColdFusion アプリケーションを使用する場合 (通常の場合)、JSP ページまたはサーブレットの変数では大文字と小文字は区別されません。
- CFML ページと JSP ページまたはサーブレットの間で Application スコープ変数または Session スコープ変数を共有し、名前のない ColdFusion アプリケーションを使用する場合、JSP ページまたはサーブレットの変数名はすべて小文字であることが必要です。
- cfobject タグまたは CreateObject 関数でクラス名を指定する場合は、名前の大文字小と文字を正しく指定する必要があります。

ガイドライン

- すべての変数名で小文字のみを使用すれば、これらの問題を防ぐことができます。
- CFML では、Java または JSP と同じ大文字小文字を使用します。これによってアプリケーションの動作が変化することはありませんが、混乱を避けることができます。

JSP タグおよびタグライブラリの使用

JSP タグライブラリの JSP タグを使用できます。たとえば、オープンソースの Apache Jakarta Project Taglibs プロジェクトのタグライブラリ (http://jakarta.apache.org/taglibs/index.html) にあるカスタムタグを使用できます。このプロジェクトではいくつかの JSP カスタムタグライブラリが公開されており、JNDI アクセスからランダムなテキスト文字列の生成まで、さまざまな処理を行うことができます。

ColdFusion ページでの JSP タグの使用

JSP ページでは、jsp:forward、jsp:include などの標準タグセットを使用します。また、JSP アプリケーションにカスタム JSP タグライブラリをインポートすることもできます。ColdFusion ページでは標準 JSP タグとカスタム JSP タグの両方を使用できます。

標準 JSP タグと ColdFusion

ほとんどの標準 JSP タグの機能は、ColdFusion タグに用意されています。たとえば、cfapplet タグには、jsp:plugin タグと同じサービスが用意されています。また cfobject タグでは、jsp:usebean タグと同じように JavaBeans を使用することができます。同様に、ColdFusion ではプロパティを参照すると自動的に取得されるので、jsp:getproperty タグを使用する必要がありません。したがって、ColdFusion では、標準 JSP タグの使用は直接サポートされていません。

ただし、JSP ページや Java サーブレットを呼び出せる forward および include の 2 つの標準 JSP タグは、ColdFusion ページで役立つ機能を備えています。「GetPageContext および PageContext オブジェクトについて」で説明した PageContext オブジェクトには、同じ操作を行う forward および include メソッドがあります。これらのメソッドの使用方法の詳細については、1114 ページの「ColdFusion ページから JSP ページまたはサーブレットへのアクセス」を参照してください。

ColdFusion ページでのカスタム JSP タグの使用

ColdFusion ページでカスタム JSP タグを使用するには、次の手順に従ってください。

カスタムタグの使用

1 タグライブラリ ("<タグライブラリ名>.jar" ファイルや "<タグライブラリ名>.tld" ファイル) を、"<Web のルートディレクトリ >/WEB-INF/lib" ディレクトリに配置します。cfimport タグを使用するには、JSP カスタムタグライブラリをこのディレクトリに配置する必要があります。

2 ColdFusion を再起動します。

3 タグライブラリの JSP タグを使用する ColdFusion ページで、cfimport タグにタグライブラリ名を指定します。次に例を示します。

```
<cfimport taglib="/WEB-INF/lib/random.jar" prefix="random">
```

TLD ファイルが JAR ファイルに含まれていない場合は、.jar 拡張子の代わりに .tld 拡張子を使用します。

注意：cfimport タグは、インポートするタグを使用するページに存在している必要があります。cfimport タグを "**Application.cfm**" に挿入することはできません。

4 カスタムタグを使用するには、< 接頭辞 >:< タグ名 > の形式を使用します。次に例を示します。

```
<random:number id="myNum" range="000000-999999" />
```

注意：cfsavecontent タグを使用してカスタム JSP タグの出力を抑制することはできません。

例 : random タグライブラリの使用

次の例では、Apache Jakarta Taglibs プロジェクトの random タグライブラリを使用して、ライブラリの number タグを呼び出します。number タグを呼び出すと、安全なアルゴリズムを使用して 6 桁の乱数を生成する乱数ジェネレータが初期化されます。変数 randPass.random を参照するたびに、新しい乱数が取得されます。

```
<cfimport taglib="/WEB-INF/lib/taglibs-random.jar" prefix="myrand">
<myrand:number id="randPass" range="000000-999999" algorithm="SHA1PRNG" provider="SUN" />
<cfset myPassword = randPass.random>
<cfoutput>
    Your password is #myPassword#<br>
</cfoutput>
```

Jakarta random タグライブラリの詳細とそのタグの使用方法については、Apache Jakarta Taglibs プロジェクトの Web サイト (http://jakarta.apache.org/taglibs/index.html) の資料を参照してください。Taglibs プロジェクトでは、オープンソースのカスタムタグライブラリが多数公開されています。

JSP ページおよびサーブレットとの相互運用

ColdFusion ページと JSP ページは、さまざまな方法で相互運用できます。

- ColdFusion ページは JSP ページおよびサーブレットを呼び出すことができます。
- JSP ページは ColdFusion ページを呼び出すことができます。
- ColdFusion ページ、JSP ページ、およびサーブレットは 3 つのスコープでデータを共有できます。

ColdFusion アプリケーションへの JSP およびサーブレットの統合

ColdFusion アプリケーションに JSP ページやサーブレットを統合できます。たとえば、アプリケーションページを JSP と CFML の両方で作成することができます。ColdFusion ページから JSP ページにアクセスするには、JSP の include および forward メソッドを使用して JSP ページを呼び出します。通常の Web アプリケーションと同様に、ColdFusion ページで href リンクを使用して JSP ページを開くこともできます。

これらの機能を使用すれば、ColdFusion アプリケーションに既存の JSP ページを組み込むことや、ColdFusion ページを使用して既存の JSP アプリケーションを拡張することができます。

ColdFusion ページを呼び出す JSP ページでは、jsp:forward または jsp:include タグを使用して ColdFusion ページを呼び出します。JSP ページから ColdFusion ページを呼び出す例については、1115 ページの「ColdFusion ページからの JSP ページの呼び出し」を参照してください。

ColdFusion ページから JSP ページまたはサーブレットへのアクセス

ColdFusion ページから JSP ページまたはサーブレットにアクセスするには、GetPageContext 関数と、forward または include メソッドを使用します。たとえば、ColdFusion アプリケーションに "Hello World" の JSP ページを含めるには、次のコードを使用します。

```
GetPageContext().include("hello.jsp");
```

JSP ページにパラメータを渡すには、ページの URL にパラメータを含めます。

たとえば、新しい ColdFusion 注文処理アプリケーションに、既存の JSP 顧客レスポンスコンポーネントを統合します。注文処理アプリケーションでは、注文番号、総額、および出荷予定日を提供し、顧客レスポンスコンポーネントでは、特定の顧客番号のファイルに記載されている電子メールアドレスにレスポンスを送信します。ColdFusion アプリケーションでは、次の CFScript コードを使用してレスポンス JSP ページを呼び出します。

```
urlParams = "UID=#order.uid#&cost=#order.total#&orderNo=#order.orderNo#&shipDate=#order.shipDateNo#"
getPageContext().forward(URLEncodedFormat("/responsegen/responsegen.jsp?#urlParams#"));
```

同じ機能を持つサーブレットにアクセスする場合も、同じコードを使用します (URL は適宜変更します)。たとえば、HelloWorldServlet というサーブレットを実行するには、サーブレットの .java ファイルまたは .class ファイルを "< サーバーのルート >/WEB-INF/classes" ディレクトリに置き、/servlet/HelloWorldServlet という URL でそのサーブレットにアクセスします。

JSP ページでの ColdFusion アプリケーションとセッション変数へのアクセス

ColdFusion は、J2EE アプリケーションサーバー上で J2EE アプリケーションとして動作します。J2EE アプリケーションの ServletContext は、オブジェクトを属性として保管するデータ構造体です。ColdFusion の Application スコープは、Application スコープ名によって名前が指定された属性として表現されます。この属性には、スコープ値がハッシュテーブルとして含まれています。したがって、JSP ページまたはサーブレットで ColdFusion の Application スコープ変数にアクセスするには、次の形式を使用します。

```
((Map)application.getAttribute("CFApplicationName"))).get("appVarName")
```

同様に、ColdFusion の Session スコープは、J2EE セッション内の構造体です。ColdFusion では、アプリケーション名でセッションが識別されるので、セッション構造体は、アプリケーション名で識別される J2EE セッションの属性に含まれています。したがって、ColdFusion のセッション変数には次のようにしてアクセスします。

```
((Map)(session.getAttribute("CFApplicationName"))).get("sessionVarName")
```

ColdFusion ページおよび JSP ページまたはサーブレット間でのデータ共有

アプリケーションに ColdFusion ページおよび JSP ページまたはサーブレットが含まれている場合は、Request、Session、および Application スコープでデータを共有できます。次の表に、スコープデータを共有する JSP ページにアクセスする方法を示します。

スコープ	データの共有に使用するメソッドまたはタグ
Request	forward、include **メモ :** JSP ページまたはサーブレットでは、共有 Request スコープの変数名は、すべて小文字にする必要があります。
Session	href、cfhttp、forward、include
Application	href、cfhttp、forward、include

注意 : ColdFusion ページと JSP ページの間でデータを共有する場合は、データ型の変換の問題に注意してください。詳細については、1123 ページの「Java と ColdFusion のデータ型の変換」を参照してください。

Session 変数を共有するには、ColdFusion Administrator で J2EE セッション管理を指定します。J2EE Session スコープ管理の設定および使用方法の詳細については、310 ページの「ColdFusion セッション管理と J2EE セッション管理」を参照してください。

たとえば、前の例で使用した顧客注文の構造体を Session スコープに格納することができます。その場合、一連のパラメータとして注文の値を渡す必要はありません。代わりに、JSP ページでは Session スコープの変数に直接アクセスできます。ColdFusion ページでは、単純に次のコードで JSP ページを呼び出せます。

```
getPageContext().forward(URLEncodedFormat("/responsegen/responsegen.jsp"));
```

Request、Session、および Application スコープを使用して、ColdFusion ページと JSP ページの間でデータを共有する例 (適切な JSP コードのサンプルを含む) については、1115 ページの「例 : CFML での JSP の使用」を参照してください。

注意 : サーバー設定で実行している ColdFusion では、JSP またはサーブレットを呼び出すときに Form スコープも共有されます。ただし、J2EE 設定で Form スコープを共有できるかどうかは、J2EE アプリケーションサーバーに依存します。たとえば、JRun は Form スコープを共有しますが、IBM WebSphere は共有しません。ColdFusion は常に Request、Session、および Application スコープを共有します。

名前のない ColdFusion Application および Session スコープ

"Application.cfc" 初期化コードの This.name 変数でアプリケーション名を指定しなかった場合、または ColdFusion の cfapplication タグを使用してアプリケーション名を指定しなかった場合、そのアプリケーションは名前を持たず、Application スコープは ColdFusion J2EE サーブレットコンテキストに対応するものになります。したがって、ColdFusion でサポートされる名前のないアプリケーションは 1 つのみです。複数の cfapplication タグおよび "Application.cfc" ファイルでアプリケーション名を指定しなかった場合、これらのアプリケーションのすべてのページは、サーブレットコンテキストを Application スコープとして共有します。

名前のないアプリケーションのすべてのセッションは、J2EE アプリケーションサーバーのセッションオブジェクトに直接対応します。J2EE セッション変数を使用しない場合、J2EE セッションは少なくともセッションがタイムアウトになるまで維持されます。

JSP ページで名前のない ColdFusion アプリケーションの Application スコープ変数にアクセスするには、次の形式を使用します。

```
application.getAttribute("applicationVariableName")
```

名前のない ColdFusion アプリケーションの Session スコープ変数にアクセスするには、次の形式を使用します。

```
session.getAttribute("sessionVariableName")
```

注意 : JSP ページおよびサーブレットで、名前のない ColdFusion アプリケーションのアプリケーション変数やセッション変数を使用する場合は、変数名の大文字と小文字を正しく指定する必要があります。変数名の文字は、ColdFusion で変数を作成したときに使用した名前と大文字小文字が完全に一致している必要があります。名前が指定された ColdFusion アプリケーションの場合は、アプリケーションおよびセッション変数名の大文字と小文字を区別する必要はありません。

例 : CFML での JSP の使用

次の簡単な例では、JSP ページ、サーブレット、および ColdFusion ページの統合方法を示します。また、Request、Application、および Session スコープを使用して、ColdFusion ページ、JSP ページ、およびサーブレットの間でデータを共有する方法も示します。

ColdFusion ページからの JSP ページの呼び出し

次のページは、Request、Session、および Application 変数を設定し、name パラメータを渡して JSP ページを呼び出します。

```
<cfapplication name="myApp" sessionmanagement="yes">
<cfscript>
Request.myVariable = "This";
Session.myVariable = "is a";
Application.myVariable = "test.";
GetPageContext().include("hello.jsp?name=Bobby");
</cfscript>
```

コードの説明

この CFML コードおよびその機能について、次の表で説明します。

コード	説明
`<cfapplication name="myApp" sessionmanagement="yes">`	アプリケーション名を myApp と指定し、セッション管理を有効にします。ほとんどのアプリケーションでは、このタグは "Application.cfm" ページに記述します。
`<cfscript> Request.myVariable = "This"; Session.myVariable = "is a"; Application.myVariable = "test.";`	ColdFusion の Request、Session、および Application スコープ変数を設定します。すべての変数を myVariable という同じ名前にしています。
`GetPageContext().include("hello.jsp?name=Bobby" ); </cfscript>`	GetPageContext 関数を使用して、ColdFusion ページに対する現在のサーブレットのページコンテキストを取得します。ページコンテキストオブジェクトの include メソッドを使用して、"hello.jsp" ページを呼び出します。name パラメータを URL に渡します。

"hello.jsp" ページが ColdFusion ページによって呼び出されます。このページは、ヘッダに name パラメータを表示し、本文に 3 つの変数を表示します。

```
<%@page import="java.util.*" %>
<h2>Hello <%= request.getParameter("name")%>!</h2>

<br>Request.myVariable: <%= request.getAttribute("myVariable")%>
<br>session.myVariable: <%= ((Map)(session.getAttribute("myApp"))).get("myVariable")%>
<br>Application.myVariable: <%= ((Map)(application.getAttribute("myApp"))).get("myVariable")%>
```

この JSP コードおよびその機能について、次の表で説明します (わかりやすくするために改行を追加しています)。

コード	説明
`<%@page import="java.util.*" %>`	java.util パッケージをインポートします。このパッケージには、JSP ページに必要なメソッドが含まれています。
`<h2>Hello <%= request.getParameter ("name")%>!</h2>`	ColdFusion ページから URL パラメータとして渡された名前を表示します。このパラメータ名は大文字と小文字が区別されます。 メモ : getParameter リクエストメソッドは、アプリケーションサーバーによっては、ColdFusion ページの一部のリクエストパラメータ値を取得できないことがあります。たとえば、IBM WebSphere では、getParameter でフォームフィールドを取得することはできません。

コード	説明
` request.myVariable: <%= request.getAttribute("myvariable")%>`	JSP リクエストオブジェクトの getAttribute メソッドを使用して、Request スコープ変数 myVariable の値を表示します。 JSP ページでは、CFML ページと共有するすべての Request スコープ変数を、すべて小文字で参照する必要があります。CFML ページでは大文字も小文字も使用できますが、JSP ページで大文字と小文字が混在する名前やすべて大文字の名前を使用した場合、その変数は ColdFusion ページから設定できません。
` session.myVariable: <%= ((Map)(session.getAttribute("myApp"))).get("myVariable")%>`	JSP セッションオブジェクトの getAttribute メソッドを使用して、myApp オブジェクト (Application スコープ) を取得します。このオブジェクトを Java Map オブジェクトにキャストし、get メソッドを使用して、表示する myVariable 値を取得します。 CFML ページと JSP ページでは、変数名の大文字小文字に関係なく、Session 変数を共有します。JSP ページの変数は、大文字小文字がどのように混在していても、ColdFusion ページから値を受け取れます。たとえば、この行の myVariable の代わりに、MYVARIABLE や myvariable を使用できます。
` Application.myVariable: <%=((Map)(application.getAttribute("myApp"))).get("myVariable")%>`	JSP myApp アプリケーションオブジェクトの getAttribute メソッドを使用して、Application スコープの myVariable の値を取得します。 CFML ページと JSP ページでは、変数名の大文字小文字に関係なく、Application 変数を共有します。JSP ページの変数は、大文字小文字がどのように混在していても、ColdFusion ページから値を受け取れます。たとえば、この行の myVariable の代わりに、MYVARIABLE や myvariable を使用できます。

JSP ページからの ColdFusion ページの呼び出し

次の JSP ページは、Request、Session、および Application 変数を設定し、name パラメータを渡して ColdFusion ページを呼び出します。

```
 <%@page import="java.util.*" %>

<% request.setAttribute("myvariable", "This");%>
<% ((Map)session.getAttribute("myApp")).put("myVariable", "is a");%>
<% ((Map)application.getAttribute("myApp")).put("myVariable", "test.");%>

<jsp:include page="hello.cfm">
    <jsp:param name="name" value="Robert" />
</jsp:include>
```

コードの説明

この JSP コードおよびその機能について、次の表で説明します。

コード	説明
`<%@page import="java.util.*" %>`	java.util パッケージをインポートします。このパッケージには、JSP ページに必要なメソッドが含まれています。
`<% request.setAttribute("myvariable", "This");%>`	JSP リクエストオブジェクトの setAttribute メソッドを使用して、Request スコープ変数 myVariable の値を設定します。 JSP ページでは、CFML ページと共有するすべての Request スコープ変数を、すべて小文字で参照する必要があります。CFML ページでは大文字も小文字も使用できますが、JSP ページで大文字と小文字が混在する名前やすべて大文字の名前を使用した場合、その変数は JSP ページと ColdFusion ページで共有されません。
`<% ((Map)session.getAttribute("myApp")).put("myVariable", "is a");%>`	JSP セッションオブジェクトの getAttribute メソッドを使用して、myApp オブジェクト (Application スコープ) を取得します。このオブジェクトを Java Map オブジェクトにキャストし、set メソッドを使用して myVariable 値を設定します。 CFML ページと JSP ページでは、変数名の大文字小文字に関係なく、Session 変数を共有します。JSP ページの変数は、大文字小文字がどのように混在していても、ColdFusion ページと値を共有できます。たとえば、この行の myVariable の代わりに、MYVARIABLE や myvariable を使用できます。
`<% ((Map)application.getAttribute("myApp")).put("myVariable","test.");%>`	JSP アプリケーションオブジェクトの getAttribute メソッドを使用して、myApp オブジェクト (Application スコープ) を取得し、それを Map オブジェクトにキャストします。次に、myApp アプリケーションスコープオブジェクトの myVariable の値を設定します。 CFML ページと JSP ページでは、変数名の大文字小文字に関係なく、Application 変数を共有します。JSP ページの変数は、大文字小文字がどのように混在していても、ColdFusion ページと値を共有できます。たとえば、この行の myVariable の代わりに、MYVARIABLE や myvariable を使用できます。
`<jsp:include page="hello.cfm"> <jsp:param name="name" value="Robert"/> </jsp:include>`	name パラメータを Robert に設定して、ColdFusion ページ "hello.cfm" を呼び出します。

次の "hello.cfm" ページが JSP ページよって呼び出されます。ヘッダに name パラメータが表示され、本文に 3 つの変数が表示されます。

```
 <cfapplication name="myApp" sessionmanagement="yes">
<cfoutput>
<h2>Hello #URL.name#!</h2>
Request.myVariable: #Request.myVariable#<br>
Session.myVariable: #Session.myVariable#<br>
Application.myVariable: #Application.myVariable#<br>

</cfoutput>
```

この CFML コードおよびその機能について、次の表で説明します。

コード	説明
`<cfapplication name="myApp" sessionmanagement="yes">`	アプリケーション名を myApp と指定し、セッション管理を有効にします。ほとんどのアプリケーションでは、このタグは "Application.cfm" ページに記述します。
`<cfoutput><h2>Hello #URL.name#!</h2>`	JSP ページの jsp:param タグから渡された名前を表示します。このパラメータ名は大文字と小文字が区別されません。
`Request.myVariable: #Request.myVariable# ` `Session.myVariable: #Session.myVariable# ` `Application.myVariable:` `#Application.myVariable# ` `</cfoutput>`	Request.myVariable、Session.myVariable、および Application.myVariable の値を表示します。CFML ページの変数名はすべて大文字と小文字が区別されません。

Java オブジェクトの使用

Java オブジェクトのインスタンスを作成するには、cfobject タグを使用します。また、オブジェクトのプロパティ (属性) やメソッド (オペレーション) を呼び出すには、cfset や cfoutput などの ColdFusion タグか、または CFScript を使用します。

メソッドの引数および戻り値は、単純な配列やオブジェクトなどの有効な Java 型であれば、どのような型でもかまいません。ColdFusion では、文字列を引数として渡した場合は適切に変換されますが、文字列を戻り値として受け取った場合は適切に変換されません。型変換の問題の詳細については、1123 ページの「Java と ColdFusion のデータ型の変換」を参照してください。

この説明の例では、cfobject タグの name 属性に値 "obj" が指定されており、このオブジェクトに Property というプロパティがあり、"Method1"、"Method2"、および "Method3" というメソッドがあるものとしています。

注意：cfdump タグを使用すると、オブジェクトのパブリックメソッドおよびデータを表示できます。

オブジェクトに関する基本テクニックの使用

ColdFusion では、Java オブジェクトを呼び出して、オブジェクトのプロパティやメソッドにアクセスできます。

オブジェクトの呼び出し

ColdFusion で Java オブジェクトを利用するには、cfobject タグを使用します。これを使用すれば、JVM クラスパスにある Java クラスや、次のいずれかの場所にある Java クラスにアクセスできます。

- "<Web のルートディレクトリ >/WEB-INF/lib" 内の Java アーカイブ (.jar) ファイル
- "<Web のルートディレクトリ >/WEB-INF/classes" 内のクラス (.class) ファイル

次に例を示します。

```
<cfobject type="Java" class="MyClass" name="myObj">
```

cfobject タグを使用するとクラスがロードされますが、インスタンスオブジェクトは作成**されません**。cfobject を呼び出した直後にアクセスできるのは、スタティックメソッドおよびフィールドのみです。

init メソッドを呼び出す前に、オブジェクトのスタティックでないパブリックメソッドを呼び出すと、デフォルトコンストラクタが暗黙的に呼び出されます。

オブジェクトコンストラクタを明示的に呼び出すには、cfobject タグを使用した後に、適切な引数を指定して ColdFusion の特別な init メソッドを呼び出します。たとえば、次のようにします。

```
<cfobject type="Java" class="MyClass" name="myObj">
<cfset ret=myObj.init(arg1, arg2)>
```

注意：この init メソッドはオブジェクトのメソッドではなく、クラスコンストラクタに対して new 関数を呼び出すための ColdFusion 識別子です。したがって、Java オブジェクトに init メソッドがある場合は、名前が競合するので、そのオブジェクトの init メソッドは呼び出せません。

オブジェクトに継続的にアクセスする場合は、init 関数を使用します。init 関数はオブジェクトのインスタンスへの参照を返しますが、cfobject は返さないためです。

cfobject を使用して作成したオブジェクトや、他のオブジェクトによって返されたオブジェクトは、ColdFusion ページの実行終了時に暗黙的に解放されます。

プロパティの使用

オブジェクトが次のいずれかの条件を満たしている場合は、次のコーディングシンタックスを使用してプロパティにアクセスできます。

- プロパティがパブリックプロパティとして公開されている。

- プロパティ自体はパブリックではないが、オブジェクトが JavaBean であり、get**PropertyName**() や set**PropertyName**(**value**) の形式の getter メソッドや setter メソッドがパブリックとして公開されている。詳細については、1120 ページの「JavaBean の get および set メソッドの呼び出し」を参照してください。
- プロパティを設定するには : <cfset obj.property = "somevalue">
- プロパティを取得するには : <cfset value = obj.property>

注意：ColdFusion では、プロパティおよびメソッド名を大文字に統一する必要はありません。ただし、一貫性を確保するために、ColdFusion でも Java と同じ大文字小文字を使用することを推奨します。

メソッドの呼び出し

オブジェクトメソッドは、通常 0 個以上の引数を取ります。メソッドには、値を返すものと返さないものがあります。メソッドを呼び出すには、次の方法を使用します。

1 メソッドに引数がない場合は、次の cfset タグのように、メソッド名に空の括弧を続けます。

```
<cfset retVal = obj.Method1()>
```

2 メソッドに 1 つ以上の引数がある場合は、引数をカンマで区切って括弧に入れます。次の例では、整数引数と文字列引数を 1 つずつ指定しています。

```
<cfset x = 23>
<cfset retVal = obj.Method1(x, "a string literal")>
```

注意：Java メソッドを呼び出す場合は、使用するデータの型が重要になります。詳細については、1123 ページの「Java と ColdFusion のデータ型の変換」を参照してください。

JavaBean の get および set メソッドの呼び出し

Java クラスが JavaBeans パターンに準拠している場合、ColdFusion は、get**PropertyName**() および set**PropertyName**(**value**) メソッドを自動的に呼び出すことができます。したがって、メソッドを明示的に呼び出さなくても、プロパティを直接参照することでプロパティの設定および取得を行えます。

たとえば、myFishTank クラスが JavaBean である場合は、次のコードを実行すると、myFish オブジェクトの getTotalFish() メソッドを呼び出した結果が返されます。

```
<cfoutput>
    There are currently #myFish.TotalFish# fish in the tank.
</cfoutput>
```

次の例では、setGuppyCount(int **number**) メソッドを暗黙的に呼び出して、myFish オブジェクトにグッピーを追加します。

```
<cfset myFish.GuppyCount = myFish.GuppyCount + 1>
```

注意：JavaBeans パターンに完全に準拠していないクラスでも、getProperty および setProperty メソッドを持っていれば、直接参照の方法を使用して値の取得や設定が行える場合があります。ただし、この方法は、getProperty および setProperty メソッドを持つすべてのクラスで使用できるわけではありません。たとえば、標準 Java クラス (Date、Boolean、Short、Integer、Long、Float、Double、Char、Byte、String、List、Array) またはその派生クラスは直接参照できません。

ネストされたオブジェクトの呼び出し

ColdFusion では、ネストされた (スコープが指定されている) オブジェクトの呼び出しがサポートされています。たとえば、オブジェクトメソッドの戻り値であるオブジェクトのプロパティやメソッドを呼び出す場合は、次のシンタックスを使用できます。

```
<cfset prop = myObj.X.Property>.
```

同様に、次のような CFScript コードを使用できます。

GetPageContext().include("hello.jsp?name=Bobby");

このコードでは、ColdFusion の GetPageContext 関数で返される Java PageContext オブジェクトの include メソッドを呼び出しています。

簡単な Java クラスの作成と使用

データ型の指定を行わない ColdFusion とは異なり、Java は厳密に型を指定する言語です。したがって、Java メソッドの呼び出しには、注意するべき点がいくつかあります。

Employee クラス

Employee クラスには、4 つのデータメンバーがあります。FirstName および LastName はパブリックであり、Salary および JobGrade はプライベートです。また、オーバーロードされた 3 つのコンストラクタと、オーバーロードされた 1 つの SetJobGrade メソッドがあります。

次の Java ソースコードをファイル "Employee.java" に保存してコンパイルし、生成された "Employee.class" ファイルをクラスパスで指定されたディレクトリに配置します。

```
public class Employee {

public String FirstName;
public String LastName;
private float Salary;
private int JobGrade;

public Employee() {
    FirstName ="";
    LastName ="";
    Salary = 0.0f;
    JobGrade = 0;
}

public Employee(String First, String Last) {
    FirstName = First;
    LastName = Last;
    Salary = 0.0f;
    JobGrade = 0;
}

public Employee(String First, String Last, float salary, int grade) {
    FirstName = First;
    LastName = Last;
    Salary = salary;
    JobGrade = grade;
}

public void SetSalary(float Dollars) {
    Salary = Dollars;
}

public float GetSalary() {
    return Salary;
}

public void SetJobGrade(int grade) {
```

```
    JobGrade = grade;
}

public void SetJobGrade(String Grade) {
    if (Grade.equals("CEO")) {
        JobGrade = 3;
    }
    else if (Grade.equals("MANAGER")) {
        JobGrade = 2;
    }
    else if (Grade.equals("DEVELOPER")) {
        JobGrade = 1;
    }
}

public int GetJobGrade() {
    return JobGrade;
}

}
```

Employee クラスを使用する CFML ページ

次のテキストを "JEmployee.cfm" として保存します。

```
<html>
<body>
<cfobject action="create" type="java" class="Employee" name="emp">
<!--- <cfset emp.init()> --->
<cfset emp.firstname="john">
<cfset emp.lastname="doe">
<cfset firstname=emp.firstname>
<cfset lastname=emp.lastname>
</body>

<cfoutput>
    Employee name is #firstname# #lastname#
</cfoutput>
</html>
```

このページをブラウザで表示すると、次の出力が表示されます。

Employee name is john doe

コードの説明

この CFML コードおよびその機能について、次の表で説明します。

コード	説明
`<cfobject action=create type=java class=Employee name=emp>`	Employee Java クラスをロードし、これに emp というオブジェクト名を付けます。
`<!--- <cfset emp.init()> --->`	コンストラクタは呼び出しません。クラスを最初に使用するとき (この例では、次の行を処理するとき) に、ColdFusion によってデフォルトコンストラクタが呼び出されます。
`<cfset emp.firstname="john">` `<cfset emp.lastname="doe">`	emp オブジェクトのパブリックフィールドに値を設定します。
`<cfset firstname=emp.firstname>` `<cfset lastname=emp.lastname>`	emp オブジェクトからフィールドの値を取得します。

`<cfoutput>`
`Employee name is #firstname# #lastname#`
`</cfoutput>`
	取得した値を表示します。

Java に関する注意事項

Java クラスオブジェクトを使用して ColdFusion ページを記述する場合は、次の点に注意してください。

- Java クラス名では大文字と小文字が区別されます。Java コードと CFML コードの両方で、クラス名として Employee が使用されていることを確認する必要があります。
- Java メソッド名およびフィールド名では大文字と小文字が区別されますが、ColdFusion 変数では区別されません。大文字と小文字の変換は ColdFusion によって必要に応じて行われます。したがって、CFML で emp.firstname や emp.lastname を使用しても、サンプルコードは機能します (Java ソースコードでは、これらのフィールドは FirstName および LastName と定義されています)。
- コンストラクタを呼び出していない場合 (または、この例のようにコメントアウトしている場合) は、クラスを最初に使用したときに、ColdFusion によってデフォルトコンストラクタが実行されます。

代替コンストラクタの使用

次の ColdFusion ページでは、Employee オブジェクトの代替コンストラクタの 1 つを明示的に呼び出しています。

```
 <html>
<body>

<cfobject action="create" type="java" class="Employee" name="emp">
<cfset emp.init("John", "Doe", 100000.00, 10)>
<cfset firstname=emp.firstname>
<cfset lastname=emp.lastname>
<cfset salary=emp.GetSalary()>
<cfset grade=emp.GetJobGrade()>

<cfoutput>
    Employee name is #firstname# #lastname#<br>
    Employee salary #DollarFormat(Salary)#<br>
    Employee Job Grade #grade#
</cfoutput>

</body>
</html>
```

この例では、コンストラクタは 4 つの引数を取ります。最初の 2 つは文字列、3 つ目は浮動小数点数、そして 4 つ目は整数です。

Java と ColdFusion のデータ型の変換

Java では変数の型を厳密に指定します。ColdFusion では変数の型を明示しませんが、ColdFusion データの表現には、ベースとなる Java の型が使用されます。

ほとんどの場合、メソッド名があいまいでなければ、Java オブジェクトに必要なデータ型が ColdFusion によって判断され、多くの場合、ColdFusion データが必要な型に変換されます。たとえば、ColdFusion のテキスト文字列は、Java の String 型に暗黙的に変換されます。同様に、Java オブジェクトに int 型のパラメータを取る doIt メソッドが含まれている場合は、整数値が含まれている CFML 変数 x を使用して doIt を呼び出すと、ColdFusion によって変数 x が Java int 型に変換されます。ただし、Java メソッドがオーバーロードされている場合 (パラメータの型のみが異なる同名メソッドがクラスに複数実装されている場合) は、あいまいな状況になります。

デフォルトのデータ型変換

ColdFusion では、Java の型と ColdFusion の型のマッチングが必要に応じて自動的に行われます。

ColdFusion で引数を渡したときに、ColdFusion のデータ値がどの Java データ型に変換されるかを、次の表に示します。左の列は、データを表すために ColdFusion で使用されているベース表現を示しています。右の列は、そのデータが ColdFusion で自動変換されたときに使用される可能性がある Java データ型を示しています。

CFML	Java
整数	short、int、long (short および int は、精度が低くなる場合があります)
実数	float double(float は、精度が低くなる場合があります)
ブール値	boolean
日付時刻値	java.util.Date
文字列 (リストを含む)	String CFML 文字列が数値または日付を表現する場合は、short、int、long、float、double、java.util.Date。 Yes、No、True、および False という値 (大文字と小文字は区別されません) を含む文字列の場合は、boolean。
配列	java.util.Vector (ColdFusion の配列は、内部では java.util.Vector オブジェクトのインスタンスを使用して表現されています)。 また、Java 配列のデータ型に変換できる同じ型のデータが含まれている CFML 配列は、byte[]、char[]、boolean[]、int[]、long[]、float[]、double[]、String[]、または Object[] にマッピングすることができます。CFML 配列に異なる型のデータが含まれていると、単純な配列型への変換は失敗することがあります。
構造体	java.util.Map
Query オブジェクト	java.util.Map
XML ドキュメントオブジェクト	サポートされていません。
ColdFusion コンポーネント	適用外

Java メソッドから返されたデータが ColdFusion によってどの ColdFusion データ型に変換されるかを、次の表に示します。

Java	CFML
boolean/Boolean	ブール値
byte/Byte	文字列
char/Char	文字列
short/Short	整数
int/Integer	整数
long/Long	整数
float/Float	実数
double/Double	実数
String	文字列
java.util.Date	日付時刻値
java.util.List	カンマ区切りリスト
byte []	配列
char[]	配列
boolean[]	配列

Java	CFML
String[]	配列
java.util.Vector	配列
java.util.Map	構造体

JavaCast 関数によるあいまいなデータ型の解決

Java メソッドはオーバーロードできるので、同じ名前のメソッドがクラスに複数含まれていることがあります。どのメソッドを使用するかは、呼び出しで渡されたパラメータとその型に基づいて、実行時に JVM によって判断されます。

The Employee class に示した Employee クラスは、SetJobGrade メソッドの実装を 2 つ持っています。1 つのメソッドは文字列値を取り、もう 1 つは整数を取ります。次に示すようなコードでは、いずれの実装を使用するのかがあいまいです。

```
<cfset emp.SetJobGrade("1")>
```

"1" は文字列としても数値としても解釈できるので、どの実装を使用すればよいのかあいまいです。ColdFusion でこのようなあいまいな状況が生じた場合は、ユーザー例外が発生します。

ColdFusion の JavaCast 関数を使用して、次のように変数の Java 型を指定すれば、この問題を解決できます。

```
<cfset emp.SetJobGrade(JavaCast("int", "1"))>
```

JavaCast 関数は、Java データ型を示す文字列と、その型を設定する変数の 2 つのパラメータを取ります。指定できる Java データ型は、boolean、int、long、float、double、および String です。

JavaCast 関数の詳細については、『CFML リファレンス』を参照してください。

Java 例外の処理

Java 例外は、標準の ColdFusion 例外と同じように、cftry および cfcatch タグを使用して処理します。例外を処理する cfcatch タグで、例外クラスの名前を指定します。たとえば、Java オブジェクトが myException という例外を投げる場合は、cfcatch タグで myException を指定します。

注意： Java オブジェクトで生成されるすべての例外を検出するには、cfcatch の type 属性で java.lang.Exception を指定します。すべての Throwable エラーを検出するには、cfcatch タグの type 属性で java.lang.Throwable を指定します。

ColdFusion で例外を処理する方法については、275 ページの「エラー処理」を参照してください。

例 : 例外が発生するクラス

次の Java コードでは、サンプル例外を投げる testException クラスを定義しています。また、Java のビルトイン Exception クラスを拡張して、エラーメッセージを取得するためのメソッドを含めた、myException クラスも定義しています。

myException クラスのコードは次のとおりです。渡されたメッセージを使用して例外を投げるか、引数が渡されなかった場合はデフォルトの例外を投げます。

```
//class myException
public class myException extends Exception
{
    public myException(String msg) {
        super(msg);
    }
    public myException() {
        super("Error Message from myException");
    }
}
```

testException クラスには、次のように、エラーメッセージを指定して myException エラーを投げるメソッドが含まれています。

```
public class testException {
    public testException ()
    {
    }
    public void doException() throws myException {
        throw new myException("Throwing an exception from testException class");
    }
}
```

例 : CFML Java 例外処理コード

次の CFML コードでは、testException クラスの doException メソッドを呼び出し、cfcatch ブロックで例外を処理しています。

```
<cfobject action=create type=java class=testException name=Obj>
<cftry>
    <cfset Obj.doException() >
    <cfcatch type="myException">
        <cfoutput>
            <br>The exception message is: #cfcatch.Message#<br>
        </cfoutput>
    </cfcatch>
</cftry>
```

例 : CFML での Java の使用

次に、CFML で Java オブジェクトを使用する例をいくつか示します。これらの例では、カスタム Java クラス、ユーザー定義関数での標準 Java API クラス、JavaBean、および EJB (Enterprise JavaBeans) の使用方法を示します。

UDF での Java API の使用

次の例に示すユーザー定義関数 (UDF) は、Common Function Library プロジェクト (www.cflib.org) にある UDF NetLib ライブラリの GetHostAddress 関数と同じ機能を実現します。この関数では、標準 Java 2 java.net パッケージの InetAddress クラスを使用して、指定されたホストのインターネットアドレスを取得します。

```
<cfscript>
function GetHostAddress(host) {
    // Define the function local variables.
    var iaddrClass="";
    var address="";
    // Initialize the Java class.
    iaddrClass=CreateObject("java", "java.net.InetAddress");
    // Get the address object.
    address=iaddrClass.getByName(host);
    // Return the address
    return address.getHostAddress();
}
</cfscript>
<cfoutput>#gethostaddress("adobe.com")#</cfoutput>
```

EJB の使用

ColdFusion では、JRun 4.0 サーバーで提供されている EJB を使用できます。JRun サーバーの "jrun.jar" ファイルのバージョンは、ColdFusion の "jrun.jar" ファイルのバージョンと同じであることが必要です。

EJB を呼び出すには、cfobject タグを使用して、目的のオブジェクトを作成して呼び出します。EJB を使用するには、その前に、次の作業を行う必要があります。

1 正しくデプロイした EJB を J2EE サーバー上で実行します。この bean は JNDI サーバーに登録されている必要があります。

2 次の情報を用意します。

- EJB サーバーの名前
- EJB サーバーの JNDI ネーミングサービスのポート番号
- ネーミングサービスに登録されている EJB の名前

3 ColdFusion Web サーバー上に、EJB ホームおよびコンポーネントインターフェイスのコンパイル済みクラスをインストールします。具体的には、"<Web のルートディレクトリ >/WEB-INF/classes" ディレクトリ内にクラスファイルとしてインストールするか、または、"<Web のルートディレクトリ >/WEB-INF/lib" ディレクトリ内に JAR ファイルとしてパッケージしてインストールします。

注意： JRun サーバーで提供されている EJB を使用するには、ColdFusion の "jrun.jar" ファイルと、EJB を提供する JRun サーバーの "jrun.jar" ファイルのバージョンが同じであることが必要です。"jrun.jar" ファイルは、ColdFusion の "<ColdFusion のルートディレクトリ >¥runtime¥lib" ディレクトリにあります。

EJB を使用するための具体的な手順は、EJB サーバーおよび使用する EJB によって異なりますが、通常は次の手順に従います。

1 cfobject タグを使用して、JNDI ネーミングコンテキストクラス (javax.naming.Context) のオブジェクトを作成します。このクラスのフィールドを使用して、EJB の検索に使用する情報を定義します。フィールドのみを使用するので、オブジェクトは初期化しません。

2 cfobject タグを使用して、コンテキストオブジェクトプロパティを保存する java.util.Properties クラスオブジェクトを作成します。

3 init メソッドを呼び出して Properties オブジェクトを初期化します。

4 Properties オブジェクトを設定して、初期 JNDI ネーミングコンテキストの作成に必要なプロパティを含めます。必要なプロパティには、INITIAL_CONTEXT_FACTORY および PROVIDER_URL プロパティがあります。ネーミングコンテキストに安全にアクセスするには、SECURITY_PRINCIPAL および SECURITY_CREDENTIALS 値も指定する必要があります。これらのプロパティの詳細については、JNDI のマニュアルを参照してください。

5 cfobject タグを使用して、JNDI InitialContext (javax.naming.InitialContext) オブジェクトを作成します。

6 Properties オブジェクト値を使用して InitialContext オブジェクトの init メソッドを呼び出して、オブジェクトを初期化します。

7 InitialContextext オブジェクトの lookup メソッドを呼び出して、bean のホームインターフェイスへの参照を取得します。bean の JNDI 名を lookup 引数として指定します。

8 bean のホームオブジェクトの create メソッドを呼び出して、bean のインスタンスを作成します。エンティティ bean を使用するときは、一般に finder メソッドを代わりに使用します。finder メソッドは、既存のエンティティ bean を検索します。

9 アプリケーションで、必要に応じて bean のメソッドを使用できるようになります。

10 処理を終えたら、コンテキストオブジェクトの close メソッドを呼び出して、オブジェクトを閉じます。

次のコードでは、JRun 4.0 サーバー上の単純な Java エンティティ bean を使用して、この手順を実行しています。このコードでは、bean の getMessage メソッドを呼び出して、メッセージを取得しています。

```
 <html>
<head>
    <title>cfobject Test</title>
</head>

<body>
<H1>cfobject Test</H1>
<!--- Create the Context object to get at the static fields. --->
<CFOBJECT
    action=create
    name=ctx
    type="JAVA"
    class="javax.naming.Context">

<!--- Create the Properties object and call an explicit constructor--->
<CFOBJECT
    action=create
    name=prop
    type="JAVA"
    class="java.util.Properties">

<!--- Call the init method (provided by cfobject)
        to invoke the Properties object constructor. --->
<cfset prop.init()>

<!--- Specify the properties These are required for a remote server only --->
<cfset prop.put(ctx.INITIAL_CONTEXT_FACTORY, "jrun.naming.JRunContextFactory")>
<cfset prop.put(ctx.PROVIDER_URL, "localhost:2908")>
<!--- <cfset prop.put(ctx.SECURITY_PRINCIPAL, "admin")>
        <cfset prop.put(ctx.SECURITY_CREDENTIALS, "admin")>
    --->
<!--- Create the InitialContext --->
<CFOBJECT
    action=create
    name=initContext
    type="JAVA"
    class="javax.naming.InitialContext">

<!--- Call the init method (provided through cfobject)
        to pass the properties to the InitialContext constructor. --->
<cfset initContext.init(prop)>

<!--- Get reference to home object. --->
<cfset home = initContext.lookup("SimpleBean")>

<!--- Create new instance of entity bean.
        (hard-wired account number). Alternatively,
        you would use a find method to locate an existing entity bean. --->
<cfset mySimple = home.create()>

<!--- Call a method in the entity bean. --->
<cfset myMessage = mySimple.getMessage()>

<cfoutput>
    #myMessage#<br>
</cfoutput>

<!--- Close the context. --->
<cfset initContext.close()>

</body>
</html>
```

カスタム Java クラスの使用

次のコードは、1121 ページの「簡単な Java クラスの作成と使用」の例よりも複雑なカスタムクラスです。Example クラスでは、整数、浮動小数点数、配列、ブール値、および Example オブジェクトを操作できます。

Example クラス

次の Java コードは Example クラスの定義です。Java クラス Example には、パブリックの整数メンバー mPublicInt があります。このコンストラクタは、mPublicInt を 0 か、整数の引数に初期化します。Example クラスには、次のパブリックメソッドがあります。

メソッド	説明
ReverseString	文字列を逆順にします。
ReverseStringArray	文字列配列の要素を逆順にします。
Add	オーバーロードされています。2 つの整数または浮動小数点数を加算して返すか、または 2 つの Example クラスオブジェクトの mPublicInt メンバーを加算して Example クラスオブジェクトを返します。
SumArray	整数配列内の要素の合計を返します。
SumObjArray	Example クラスオブジェクトの配列の mPublicInt メンバーの値を加算して、Example クラスオブジェクトを返します。
ReverseArray	整数配列を逆順にします。
Flip	ブール値を切り替えます。

```
public class Example {
    public int mPublicInt;

    public Example() {
        mPublicInt = 0;
    }

    public Example(int IntVal) {
        mPublicInt = IntVal;
    }

    public String ReverseString(String s) {
        StringBuffer buffer = new StringBuffer(s);
        return new String(buffer.reverse());
    }

    public String[] ReverseStringArray(String [] arr) {
        String[] ret = new String[arr.length];
        for (int i=0; i < arr.length; i++) {
            ret[arr.length-i-1]=arr[i];
        }
        return ret;
    }

    public int Add(int a, int b) {
        return (a+b);
    }

    public float Add(float a, float b) {
        return (a+b);
    }

    public Example Add(Example a, Example b) {
```

```
        return new Example(a.mPublicInt + b.mPublicInt);
    }

    static public int SumArray(int[] arr) {
        int sum=0;
        for (int i=0; i < arr.length; i++) {
            sum += arr[i];
        }
        return sum;
    }

    static public Example SumObjArray(Example[] arr) {
        Example sum= new Example();
        for (int i=0; i < arr.length; i++) {
            sum.mPublicInt += arr[i].mPublicInt;
        }
        return sum;
    }

    static public int[] ReverseArray(int[] arr) {
        int[] ret = new int[arr.length];
        for (int i=0; i < arr.length; i++) {
            ret[arr.length-i-1]=arr[i];
        }
        return ret;
    }

    static public boolean Flip(boolean val) {
        System.out.println("calling flipboolean");
        return val?false:true;
    }
}
```

useExample ColdFusion ページ

次の "useExample.cfm" ページでは、Example クラスを使用して数値、文字列、ブール値、および Example オブジェクトを操作します。CFML JavaCast 関数で、CFML 変数を適切な Java データ型に変換しています。

```
 <html>
<head>
    <title>CFOBJECT and Java Example</title>
</head>
<body>

<!--- Create a reference to an Example object --->
<cfobject action=create type=java class=Example name=obj>
<!--- Create the object and initialize its public member to 5 --->
<cfset x=obj.init(JavaCast("int",5))>

<!--- Create an array and populate it with string values,
        then use the Java object to reverse them. --->
<cfset myarray=ArrayNew(1)>
<cfset myarray[1]="First">
<cfset myarray[2]="Second">
<cfset myarray[3]="Third">
<cfset ra=obj.ReverseStringArray(myarray)>

<!--- Display the results --->
<cfoutput>
    <br>
    original array element 1: #myarray[1]#<br>
    original array element 2: #myarray[2]#<br>
    original array element 3: #myarray[3]#<br>
```

```
    after reverseelement 1: #ra[1]#<br>
    after reverseelement 2: #ra[2]#<br>
    after reverseelement 3: #ra[3]#<br>
    <br>
</cfoutput>


<!--- Use the Java object to flip a Boolean value, reverse a string,
        add two integers, and add two float numbers --->
<cfset c=obj.Flip(true)>
<cfset StringVal=obj.ReverseString("This is a test")>
<cfset IntVal=obj.Add(JavaCast("int",20),JavaCast("int",30))>
<cfset FloatVal=obj.Add(JavaCast("float",2.56),JavaCast("float",3.51))>

<!--- Display the results --->
<cfoutput>
    <br>
    StringVal: #StringVal#<br>
    IntVal: #IntVal#<br>
    FloatVal: #FloatVal#<br>
    <br>
</cfoutput>

<!--- Create a two-element array, sum its values,
        and reverse its elements --->
<cfset intarray=ArrayNew(1)>
<cfset intarray[1]=1>
<cfset intarray[2]=2>
<cfset IntVal=obj.sumarray(intarray)>
<cfset reversedarray=obj.ReverseArray(intarray)>

<!--- Display the results --->
<cfoutput>
    <br>
    IntVal1 :#IntVal#<br>
    array1: #reversedarray[1]#<br>
    array2: #reversedarray[2]#<br>
    <br>
</cfoutput><br>

<!--- Create a ColdFusion array containing two Example objects.
        Use the SumObjArray method to add the objects in the array
        Get the public member of the resulting object--->
<cfset oa=ArrayNew(1)>
<cfobject action=create type=java class=Example name=obj1>
<cfset VOID=obj1.init(JavaCast("int",5))>
<cfobject action=create type=java class=Example name=obj2>
<cfset VOID=obj2.init(JavaCast("int",10))>
<cfset oa[1] = obj1>
<cfset oa[2] = obj2>
<cfset result = obj.SumObjArray(oa)>
<cfset intval = result.mPublicInt>

<!--- Display the results --->
<cfoutput>
    <br>
    intval1: #intval#<br>
    <br>
</cfoutput><br>
</body>
</html>
```

ColdFusion 10 での拡張された Java 統合

Java ライブラリの統合

ColdFusion 10 Beta では、ユーザーが指定するカスタムパスから Java ライブラリを読み込むことができます。以前のバージョンでは、ColdFusion の lib ディレクトリに置かれた Java ライブラリを使用します。このようなライブラリはアプリケーション固有ではなく、Java ライブラリの追加や既存ライブラリの更新は容易ではありません。また、ColdFusion を再起動する必要があります。

ColdFusion 10 Beta では、アプリケーション用の Java ライブラリをユーザーが選択したディレクトリに配置できます。このディレクトリのパスは Application.cfc で指定します。その後、Java 型の cfobject を作成して、アプリケーションでライブラリを使用します。

動的な読み込みを使用しない Application.cfc でのカスタム Java ライブラリパスの指定

Java ライブラリを読み込むカスタムパスをプロジェクトの Application.cfc で指定します。

この場合、ファイルを更新すると、ColdFusion を再起動して更新したファイルを読み込む必要があります。

次のエントリをこのファイルに追加します。

```
THIS.javaSettings = {LoadPaths = ["**.\java_lib\",".\java\myjar.jar**"], loadColdFusionClassPath = true,
reloadOnChange = false}
```

パラメータ

パラメータ	説明
loadPaths	Java クラスまたは JAR ファイルを含むディレクトリへのパスの配列。 JAR またはクラスへのパスを指定することもできます。パスを解決できない場合は、エラーが発生します。
loadColdFusionClassPath	ColdFusion の lib ディレクトリからクラスを読み込むかどうかを示します。 デフォルト値は false です。
reloadOnChange	ColdFusion を再起動せず、更新されたクラスおよび JAR を動的にリロードするかどうかを示します。デフォルト値は false です。

動的な読み込みを使用する Application.cfc でのカスタム Java ライブラリパスの指定

Java ライブラリを読み込むカスタムパスをプロジェクトの Application.cfc で指定します。

この場合、ファイルを更新しても、ColdFusion を再起動して更新したファイルを読み込む必要はありません。

次のエントリをこのファイルに追加します。

```
THIS.javaSettings = {LoadPaths = ["**.\java_lib\",".\java\myjar.jar**"], loadColdFusionClassPath = true,
reloadOnChange= true, watchInterval = 100, watchExtensions = "**jar,class,xml**"}
```

パラメータ

パラメータ	説明
loadPaths	Java クラスまたは JAR ファイルを含むディレクトリへのパスの配列。 JAR またはクラスへのパスを指定することもできます。パスを解決できない場合は、エラーが発生します。
loadColdFusionClassPath	ColdFusion の lib ディレクトリからクラスを読み込むかどうかを示します。 デフォルト値は false です。

パラメータ	説明
reloadOnChange	ColdFusion を再起動せず、更新されたクラスおよび JAR を動的にリロードするかどうかを示します。デフォルト値は false です。
watchInterval	クラスファイルまたは JAR ファイルの変更を調べる間隔（秒単位）を指定します。 この属性は、reloadOnChange 属性を true に設定している場合にのみ適用されます。デフォルト値は 60 秒です。
watchExtensions	変更を監視するファイルの拡張子を指定します。デフォルトでは、.class ファイルと .jar ファイルだけが監視されます。

Java クラスの使用

Java クラスファイルまたは JAR を Application.cfc で指定したディレクトリに保存します。その後、Java 型の cfobject を作成して、JAR またはクラスファイルのメソッドにアクセスします。

この例では、Java クラスを作成し、サンプルアプリケーション内の Java クラスのメソッドにアクセスします。

1 Java ファイル Test.java を作成します。

```
public class Test
    {
        public String testJava()
        {
        return "Hello Java!!";
        }
    }
```

2 Java コンパイラーを使用して Java ファイルをコンパイルします。

3 次のエントリをプロジェクトの Application.cfc に追加します。

```
<cfset THIS.javaSettings = {LoadPaths = ["/myJava/lib"],reloadOnChange=true,watchInterval=30}/>
```

4 アプリケーションフォルダーに次のディレクトリ構造を作成します。

```
/myJava/lib
```

5 Test.class ファイルを /myJava/lib フォルダーにコピーします。

6 次の内容の CFM ファイルを作成します。

```
<cfobject type="java" class="Test" name="myObj">
    <cfset y = myObj.init()>
        <cfoutput >
        #y.testJava()#
        </cfoutput>
```

7 アプリケーションをデプロイして CFM ファイルにアクセスします。

CFC プロキシの使用

CFC プロキシを使用すると、Java クラスから ColdFusion コンポーネントにアクセスできます。CFC を呼び出すには、ColdFusion クラスローダーが現在のクラスローダーである必要があります。例えば、次のコードは指定したファイルの場所から CFC プロキシを作成します。

```
CFCProxy(String fully-qualified-path-of-CFC-file)
```

同様に、次のコードは指定したファイルの場所から CFC プロキシを作成します。また、CFC の This スコープを名前と値のペアで初期化します。

```
CFCProxy(String fully-qualified-path-of-CFC-file name-value-pairs)
```

ColdFusion 10 Beta では、CFProxy クラスに新しい引数 directInvoke が導入されました。

この引数を true に設定した場合、リクエストは ColdFusion のフィルターチェーンを通過しません。その結果、パフォーマンスが向上します。

次の例では、CFC プロキシを使用して ColdFusion コンポーネントにアクセスする Java クラスを作成します。

1 次の Java クラスを作成します。

```
import coldfusion.cfc.CFCProxy;
public class CFCInvoker
{
    public String directInvoke(String cfcPath)
    {
        String myMessage = "";
        try
            {
                CFCProxy myCFC = new CFCProxy(cfcPath, true);
                Object[] myArgs = { "Hello" };
                myMessage = (String)myCFC.invoke("getData", myArgs);
            }
        catch (Throwable e)
            {
                e.printStackTrace();
            }
                return myMessage;
    }
}
```

2 ファイルをコンパイルし、プロジェクトフォルダーの lib ディレクトリに配置します。

3 次のように ColdFusion コンポーネントを作成します。

```
<cfcomponent>
<cffunction name="getData" returntype="string">
    <cfargument name="msg" required="Yes">
        <cfreturn msg & "Java" />
    </cffunction>
</cfcomponent>
```

4 application.cfc で、次のエントリを追加します。

```
THIS.javaSettings = {LoadPaths = ["/lib"],reloadOnChange= true,loadColdFusionClassPath=true};
```

5 CFM ファイルを作成して出力を表示します。

```
<cfobject action="create" type="java" class="CFCInvoker" name="x">
<cfset cfcPath = "C:\ColdFusion10\cfusion\wwwroot\Project\name.cfc"/>
<cfset y = x.directInvoke2(cfcPath)>
<cfoutput>#y#</cfoutput>
```

6 CFM ファイルにアクセスします。

createDynamicProxy 関数の使用

関数 createDynamicProxy は、Java ライブラリに渡される ColdFusion コンポーネントの動的なプロキシを作成します。動的なプロキシを使用すると、ColdFusion コンポーネントを Java オブジェクトに渡すことができます。Java オブジェクトは、ColdFusion コンポーネントと、それがネイティブな Java オブジェクトであるかのようにシームレスに連携できます。

動的なプロキシを作成するには、ColdFusion コンポーネントの名前と Java インターフェイスの配列を指定します。コンポーネントは、指定したインターフェイスを実装しているかのように扱われます。

次のように動的なプロキシを作成します。

```
createDynamicProxy("fullyQualifiedNameOfCFC", ["interfaceName"]);
```

次のパラメーターを指定します。

- ColdFusion コンポーネントまたは CFC インスタンスの完全修飾名。
- 動的なプロキシを作成する Java インターフェイスの配列。

動的なプロキシの作成

次の例では、Java インターフェイスの作成方法を示します。インターフェイスはメソッドを定義します。ColdFusion コンポーネントは、メソッドを ColdFusion 関数として実装します。ColdFusion コンポーネントの動的なプロキシは、インターフェイスのオブジェクトを渡すことで Java クラスを呼び出します。その後、ネイティブな Java クラスであるかのように ColdFusion のメソッドを呼び出します。

1 Java インターフェイス MyInterface を作成します。

```
public interface MyInterface
{
    public String sayHello();
}
```

2 Java ファイルをコンパイルし、ディレクトリ lib に配置します。

3 インターフェイスのインスタンスを使用して ColdFusion オブジェクトを呼び出す Java クラス InvokeHelloProxy を作成します。

コンストラクター InvokeHelloProxy は MyInterface のオブジェクトを作成します。invokeHello() メソッドは、オブジェクトを使用して sayHello() メソッドを呼び出します。

```
public class InvokeHelloProxy
{
    private MyInterface myInterface;
        public InvokeHelloProxy(MyInterface x)
        {
            this.myInterface = x;
        }
        public String invokeHello()
        {
            return myInterface.sayHello();
        }
}
```

4 Java ファイルをコンパイルし、ディレクトリ lib に配置します。

5 インターフェイスで定義されているメソッドを実装する CFC ファイル HelloWorld.cfc を作成し、ディレクトリ cfc に保存します。

```
<cfcomponent>
    <cffunction name="sayHello" returntype="string">
        <cfreturn "Hello World!!!!">
    </cffunction>
</cfcomponent>
```

6 次のコードを Application.cfc に追加します。

```
<cfset THIS.javaSettings = {LoadPaths = ["/lib"],reloadOnChange=true,watchInterval=10}/>
```

7 インターフェイス MyInterface として HelloWorld 用の動的なプロキシを作成する CFM ファイルを追加します。InvokeHelloProxy クラスのオブジェクトを作成し、クラスを初期化します。

このコードは MyInterface の動的なプロキシを作成します。

```
<cfset dynInstnace = createDynamicProxy("cfc.HelloWorld", ["MyInterface"])>
<cfset x = createObject("java","InvokeHelloProxy").init(dynInstnace)>
<cfset y = x.invokeHello()>
<cfoutput>#y#</cfoutput>
```

例：CFC インスタンスの使用

```
<cfset instance=new cfc.helloWorld()>
<cfset dynInstnace = createDynamicProxy(instance, ["MyInterface"])>
<cfset x = createObject("java","InvokeHelloProxy").init(dynInstnace)>
<cfset y = x.invokeHello()>
<cfoutput>#y#</cfoutput>
```

8 アプリケーションをデプロイして CFM ファイルにアクセスします。

Microsoft .NET アセンブリの使用

Adobe ColdFusion では、ローカルまたはリモートにある Microsoft .NET アセンブリのクラスメソッドを呼び出したり、アセンブリフィールドにアクセスすることができます。本マニュアルでは、ColdFusion .NET Extension ソフトウェアを設定して実行する方法について説明します。また、ColdFusion コードで .NET クラスにアクセスして使用する方法についても説明します。.NET テクノロジーの詳細と、.NET アプリケーションの開発方法については、Microsoft .NET のマニュアルを参照してください。

ColdFusion と .NET について

ColdFusion では、Microsoft .NET アセンブリクラスに CFML オブジェクトとしてアクセスして使用することができます。CFML アプリケーションで .NET アセンブリを使用すると、次のことが行えます。

- Word、Excel、PowerPoint などの Microsoft 製品に直接アクセスして制御できます。
- 既存の .NET コンポーネントを使用できます。
- .NET アセンブリを使用すれば、ColdFusion や Java では実現が困難または不可能な機能を利用できます。ただし、ColdFusion は J2EE アプリケーションなので、CFML コードで実現できない機能を作成する場合は、.NET よりも Java で作成するほうが効率的です。

アプリケーションで使用する .NET クラスは、ローカルにある必要はなく、リモートシステムにある .NET コンポーネントにもアクセスできます。これは、ファイアウォールの外にあってもかまいません。また、リモートの .NET コンポーネントを使用するために、.NET ランタイムソフトウェアを ColdFusion システムにインストールする必要はないので、UNIX、Linux、Solaris、OS-X システム上の ColdFusion で .NET アセンブリにアクセスして使用することができます。

オブジェクトタイプとして .NET または dotnet を指定することで、cfobject タグまたは CreateObject 関数を使用して、.NET クラスオブジェクトへの参照を作成できます。この参照を使用して、.NET クラスフィールドへのアクセスや、.NET クラスメソッドの呼び出しが行えます。この方法を使用すれば、ColdFusion から .NET クラスへのステートフルな密結合アクセスが効果的に実現できます。.NET アプリケーションのクラスメソッドを Web サービスとして公開する方法もありますが、信頼性やパフォーマンスの面で劣り、拡張性も制限されるので、.NET クラスの ColdFusion オブジェクトを使用するほうが優れています。

注意：.NET アプリケーションは、ColdFusion コンポーネントの関数に直接アクセスできません。リモートアクセスを指定して、関数を Web サービスとして公開することができます。ColdFusion Web サービスの作成方法の詳細については、1061 ページの「Web サービスの使用」を参照してください。

.NET アセンブリクラスは、通常の ColdFusion オブジェクトと同じ方法で使用するので、.NET テクノロジーの詳細を理解する必要はありません。アクセスする .NET クラスの使用方法を把握していれば十分です。.NET メソッドを使用するコードは、次のように単純です。

```
<cfobject type = ".NET" name = "mathInstance" class = "mathClass"
        assembly = "C:/Net/Assemblies/math.dll">
<cfset myVar = mathInstance.multiply(1,2)>
```

ColdFusion .NET アクセスには、さらに次のような機能があります。

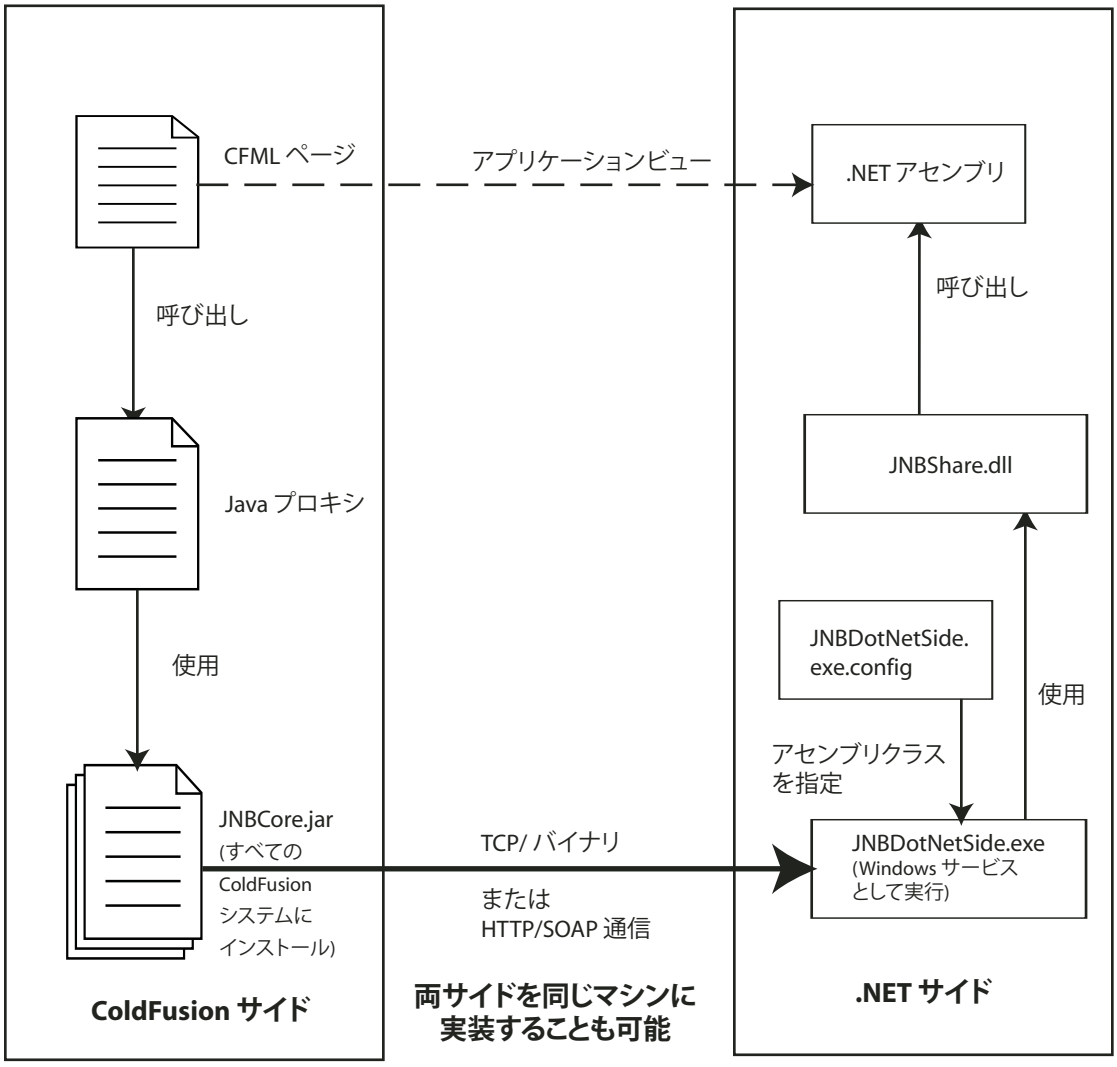
- .NET アセンブリを変更すると、変更が行われたことが ColdFusion によって自動的に認識され、次回の呼び出しには変更されたバージョンが使用されます。
- アプリケーションから、複数のマシンで実行されている .NET アセンブリにアクセスできます。
- ColdFusion と .NET の間の通信は、SSL を使用して保護することができます。
- プリミティブデータ型は、ColdFusion と .NET との間で自動的にマッピングされます。

.NET アクセスの仕組み

ColdFusion が .NET アセンブリにアクセスできるようにするには、アセンブリが存在するシステムで ColdFusion .NET Extension ソフトウェアを実行する必要があります。リモートのアセンブリにのみアクセスする ColdFusion システムには、.NET Extension は必要ありません。.NET Extension ソフトウェアは、.NET アセンブリへのアクセスを可能にする .NET サイドの接続機能を提供します。このソフトウェアには、ColdFusion システムからのリクエストをリスンして処理する .NET サイドのエージェントが含まれています (通常、ColdFusion .NET サービスとして実行します)。

ColdFusion システムの ColdFusion オブジェクトでは、.NET クラスのローカルインターフェイスとして機能する Java プロキシが使用されます。このプロキシはバイナリ TCP または SOAP ベースの HTTP 通信を使用して .NET サイドのエージェントにアクセスします。エージェントは DLL を使用して .NET アセンブリクラスを呼び出します。この通信は、ColdFusion と .NET アセンブリが同じシステムにある場合も含め、常に必要になります。

次の図に、CFML から .NET へのアクセスの仕組みを示します。



.NET アセンブリがローカルシステムにある場合は、必要なプロキシおよび設定情報が ColdFusion によってすべて自動的に作成され管理されます。開発者が行う必要があるのは、.NET Extension をシステムにインストールして、ColdFusion .NET サービスを実行することだけです。この作業を行えば、cfobject タグや CreateObject 関数を使用してアセンブリにアクセスできるようになります。

アセンブリがリモートシステムにある場合は、ColdFusion .NET Extension ソフトウェアを .NET システムにインストールし、これを使用して .NET クラスの Java プロキシを作成し、それを ColdFusion システムに移動またはコピーします。また、リモートシステムの "JNBDotNetSide.exe.config" ファイルを編集して、使用する .NET クラスを指定します。.NET システムには、次の .NET Extension ソフトウェアが必要です。

- "JNBDotNetSide.exe"。ColdFusion システムと通信を行う .NET サイドのエージェント。通常は、ColdFusion .NET サービスとして実行します。
- "JNBDotNetSide.exe.config"。ColdFusion からアクセス可能な .NET アセンブリを識別する設定ファイル。
- "jnbproxy.exe" と "jnbproxyGui.exe"。.NET アセンブリを表す Java プロキシを生成するコマンドラインおよび GUI ベースのプログラム。

- その他のサポートファイル (.NET アセンブリクラスを呼び出す "JNBShare.dll" など)。

ColdFusion .NET Extension のインストールの詳細については、『ColdFusion インストール』ガイドを参照してください。

注意 : .NET の新しいバージョンをインストールするときは、ColdFusion .NET Extension を再インストールする必要があります。

.NET アセンブリへのアクセス

ColdFusion では、2 通りの方法で .NET アセンブリにアクセスできます。

- ColdFusion システムにインストールされている .NET オブジェクトにローカルアクセスを行う。
- 他のシステム上にある .NET オブジェクトにリモートアクセスを行う。

どちらの方法を使用する場合も、アセンブリが存在するシステムに ColdFusion .NET Extension をインストールして、ColdFusion .NET サービスを実行する必要があります。リモートのアセンブリにのみアクセスする ColdFusion システムでは、ColdFusion .NET Extension のインストールやサービスの実行は必要ありません。ColdFusion .NET Extension のインストールの詳細については、『ColdFusion インストール』ガイドを参照してください。

ローカルアセンブリへのアクセス

ローカルアクセスの場合は、cfobject タグまたは CreateObject 関数を初めて使用するときに、.NET アセンブリに対応するプロキシが ColdFusion によって自動的に生成されて使用されます。このプロキシは次の使用に備えてキャッシュされるので、アセンブリプロキシが毎回生成されることはありません。

ローカルの .NET アセンブリにアクセスする場合、通常はデフォルトの通信設定をそのまま使用できますが、場合によっては設定の変更が必要になることもあります。たとえば、システム上の他のソフトウェアがデフォルトの 6086 ポートを使用している場合は、"jnbridge¥DotNetSide.exe.config" ファイルのポート番号の指定を変更し、cfobject タグまたは CreateObject タグで変更したポート番号を指定します。ポート番号の変更の詳細については、1142 ページの「.NET サイドのシステムの設定」を参照してください。

ローカルアクセスを行う場合は、cfobject タグまたは CreateObject 関数を使用して、プロキシを作成してアクセスします。作成した ColdFusion オブジェクトを使用して .NET オブジェクトを作成し、.NET オブジェクトのメソッドを呼び出したり、フィールドにアクセスすることができます。.NET クラスの使用方法の詳細については、1143 ページの「.NET クラスの使用」を参照してください。

リモートアセンブリへのアクセス

リモートアクセスの場合は、TCP または HTTP を使用して、リモートシステム上の .NET サイドエージェントと通信することで、.NET アセンブリにアクセスします。ローカルアクセスの場合と同様に、プロキシのインスタンスを作成してアセンブリメソッドを呼び出しますが、その前に、リモートの .NET サイドエージェントを設定します。また、ほとんどの場合、リモートの .NET クラスを表すプロキシクラスも設定します。

リモート .NET アクセスの設定

1. リモートシステムに ColdFusion .NET 統合ソフトウェアをインストールして、.NET サイドエージェントを実行します (『ColdFusion インストール』ガイドを参照してください)。
2. .NET アセンブリがリモートシステムにのみ存在する場合は、アセンブリを表すプロキシ JAR ファイルをそのシステムで生成します (1140 ページの「Java プロキシクラスの生成」を参照してください)。そのプロキシファイルを、ローカルシステムにコピーまたは移動します。同一の .NET アセンブリがローカルシステムにも存在する場合は、この手順を省略できます。
3. リモートアクセス用に .NET サイドのシステムを設定します (1142 ページの「.NET サイドのシステムの設定」を参照してください)。

Java プロキシクラスの生成

Java プロキシ生成コードでプロキシクラスを生成するには、生成コードが .NET アセンブリに直接アクセスする必要があります。したがって、ColdFusion アプリケーションを実行するシステムにアセンブリがインストールされていない場合は、.NET サイドのシステムでツールを実行して Java プロキシを作成します。.NET サービスをインストールすると、"jnbridge" ディレクトリに "jnbproxyGui.exe" と "jnbproxy.exe" の 2 つのプロキシ生成プログラムがインストールされます。"jnbproxyGui.exe" プログラムは Windows ユーザーインターフェイスアプリケーションで、"jnbproxy.exe" プログラムはコマンドラインアプリケーションです。いずれのプログラムも機能は同じです。

注意：ColdFusion アプリケーションを実行するシステムにアセンブリがインストールされている場合は、(設定が異なるなどの理由で) リモートのアセンブリにアクセスする必要があっても、手動でプロキシクラスを生成する必要はなく、この手順は省略できます。代わりに、cfobject タグ (または CreateObject 関数) の assembly 属性でローカルの .exe または .dll ファイルのパスを指定し、server 属性でリモートサーバーを指定します。ただし、アクセスできるようにリモートシステムを設定する必要があります。

ColdFusion システムでは、jnbproxyGui および jnbproxy プログラムは "<ColdFusion のルートディレクトリ >¥jnbridge" ディレクトリにあります。スタンドアローンのインストーラは、"< インストールディレクトリ >¥jnbridge" ディレクトリにあります。

本マニュアルでは、jnbproxyGui ツールを使用してプロキシ JAR ファイルを生成するために必要な基本情報について説明します。詳細な情報については、次の場所を参照してください。

- "jnbridge" ディレクトリにある "jnbproxy.chm" Windows ヘルプファイルには、ColdFusion の .NET 機能を拡張する JNBridge テクノロジーに関する詳細な説明があります。また、jnbproxyGui と jnbproxy プログラムの詳細情報も含まれています。
- "jnbridge¥docs" サブディレクトリには、ヘルプファイルの PDF 版である "users guide.pdf" などの追加のマニュアルがあります。

注意：JNBridge のマニュアルでは、ColdFusion でサポートされていない機能に関する情報が提供されています。ColdFusion では、.NET アセンブリから ColdFusion へのアクセスや、メモリのみの通信などはサポートされていません。

jnbproxyGui ツールの使用

jnbproxyGui プログラムを使用してプロキシ JAR ファイルを生成します。

プロキシ JAR の生成とインストール

1. "JNBProxyGui.exe" を起動します。
2. 最初にこのプログラムを実行すると、[Enter Java Options] ダイアログボックスが表示されます。オプションを選択して、[OK] をクリックします。

   この設定は、後で [Project]-[Java Options] を選択して変更することができます。

   ColdFusion が存在するシステムの場合 : このシステムで現在 ColdFusion を実行している場合は、JNBProxy の [Enter Java Options] ダイアログボックス ([Project]-[Java Options]) の右側にある [Start Java Automatically] オプションがオフになっていることを確認します。その他の設定はデフォルト値のままにします。

   既存のプロジェクトを開くと、[Restart Java Side] というポップアップウィンドウが開いて、クラスパスの変更を有効にするには Java サイドを停止して再起動する必要があるというメッセージが表示されることがあります。このメッセージは無視して、[OK] をクリックして続行できます。

   このプログラムを起動すると、[Java Options] ダイアログボックスが表示されることがあります。ここで変更を行う必要はありません。[OK] または [キャンセル] をクリックすると、[Launch JNBProxy] ダイアログボックスが開きます。

   場合によっては、[Start Java Automatically] オプションが選択されていないと、JNBProxyGui で次の動作が行われることがあります。

ColdFusion が存在しないシステムの場合 : このシステムで現在 ColdFusion を実行していない場合は、インターフェイスの右側にある次のオプションが設定されていることを確認します。その他の設定はデフォルト値のままにします。

- [Start Java Automatically] オプションが選択されていることを確認します。
- JAR ファイルのコンパイルに使用する "java.exe" ファイルを指定します。このファイルには、Java 1.4 または 1.5 (J2SE 5.0) バージョンを使用できます。
- "jnbcore.jar" ファイルを指定します。ColdFusion サーバーのインストーラーを使用した場合は、**cfroot**¥lib ディレクトリに保存されています。J2EE インストーラを使用した場合は、"<ColdFusion Web アプリケーションのルートディレクトリ >¥WEB-INF¥cfusion¥lib" ディレクトリに保存されています。
- "bcel.jar" ファイルを指定します。ColdFusion サーバーのインストーラを使用した場合は、"<ColdFusion のルートディレクトリ >¥lib" ディレクトリに保存されています。J2EE インストーラを使用した場合は、"<ColdFusion Web アプリケーションのルートディレクトリ >¥WEB-INF¥cfusion¥lib" ディレクトリに保存されています。

3 [Launch JNBProxy] ダイアログボックスで、[Create New Java]-[.NET Project] を選択し、[OK] をクリックします。

4 メインの Java プロキシ生成インターフェイスで、プロジェクトの設定と構築を行います。

a まだ行っていない場合は、アセンブリファイルが格納されているディレクトリを JNBProxy のプロジェクトに追加します。[Project]-[Edit Assembly List] を選択します。[Assembly List] ダイアログボックスで [Add] ボタンをクリックします。[New Assembly List Element] ダイアログボックスで、アセンブリが格納されているディレクトリに移動します。ツリーでディレクトリ (複数も可) を選択して、[OK] をクリックします。[Edit Assembly List] ダイアログボックスが表示されたら、[OK] をクリックします。

b [Locate Assembly File] ダイアログボックスを開き ([Project]-[Add Classes From Assembly File])、手順 a でアセンブリリストに追加したディレクトリに移動します。プロキシを必要とするクラスが含まれているアセンブリファイル (複数も可) を選択して、[OK] をクリックします。

c 選択したファイル内のクラスと、そのクラスが依存しているその他の .NET コアクラスが、[Environment] ペインに表示されます。JAR ファイル内のプロキシを必要とするすべてのクラスを選択し、[Add+] ボタンをクリックして、選択したクラスとすべてのサポートクラスを追加します。

d [Exposed Proxies] リストで、JAR ファイルに含めるクラスを選択します。通常は、表示されたすべてのクラスを選択し、必要なクラスが確実に含まれるようにします。

e メインメニューから [Project]-[Build] を選択します。[Save Generated Proxies] ダイアログボックスで、生成するプロキシの保存場所と JAR ファイルを指定して、[Save] をクリックします。

f プロジェクトが構築されたら、[File]-[Save Project] を選択し、プロジェクトを保存するファイルを指定します。

次回 jnbproxyGui プログラムを実行するときは、このプロジェクトを選択することで、以前の設定 (アセンブリリストなど) を再利用できます。

5 JAR ファイルを ColdFusion システムのディレクトリにコピーします。このパスを cfobject タグの assembly 属性で指定します。

サポートクラス

JNBProxy は、明示的に指定された .NET クラスだけでなく、サポートクラスに対してもプロキシを生成することができます。.NET クラスのサポートクラスとは、その .NET クラスを使用することで直接的または間接的に必要になる可能性があるすべてのクラスのことです。.NET クラスのサポートクラスには、次のすべてのクラスが含まれます。

- そのクラス自体。
- そのクラスのスーパークラスまたはスーパーインターフェイス (存在する場合) と、そのすべてのサポートクラス。
- そのクラスで実装されているインターフェイス (存在する場合) と、そのすべてのサポートクラス。
- そのクラスの各フィールドに対して :
  - そのフィールドのクラスと、そのすべてのサポートクラス。

    - そのフィールドの各インデックスパラメータに対して、そのパラメータのクラスと、そのすべてのサポートクラス。
- そのクラスの各メソッドに対して :
    - そのメソッドの戻り値のクラス (存在する場合) と、そのすべてのサポートクラス。
    - そのメソッドの各パラメータに対して、そのパラメータのクラスと、そのすべてのサポートクラス。
- そのクラスの各コンストラクタの各パラメータに対して、そのパラメータのクラスと、そのすべてのサポートクラス。

Java では、メソッドが投げる例外もサポートクラスに含まれますが、.NET の例外は事前に定義されないので、サポートクラスには含まれません。

サポートクラスの数は、明示的に指定するクラスによって異なりますが、多くの場合 200 ～ 250 クラスです。通常は、すべてのサポートクラスを生成します。ただし、時間や容量を節約するために、サポートクラスを除外して、明示的に指定したクラスのみを生成できます。

プロキシが生成されていないサポートクラスに対してプロキシが必要になった場合は、必要なクラスに最も近いスーパークラスのプロキシが代わりに使用されます。そのプロキシも生成されていない場合は、そのスーパークラスのさらにスーパークラスのプロキシが生成されていれば使用され、以下同様に、System.Object のプロキシ (必ず生成されています) まで続きます。このように、プロキシのセットが不完全であってもコードは実行されますが、機能やその他の情報が失われる可能性があります。

jnbproxyGui ツールでは、[Add] ボタンをクリックすると、明示的に指定したクラスのみがリストに含まれます。[Add+] ボタンをクリックすると、サポートクラスもリストに含まれます。jnbproxy コマンドラインプログラムでは、デフォルトのコマンドでサポートクラスのプロキシが生成されます。このデフォルトを上書きするには、/ns オプションを使用します。

.NET サイドのシステムの設定

.NET サイドのシステムを設定するには、次のように "jnbridge¥JNBDotNetSide.exe.config" ファイルを編集します。

- ローカルアセンブリの場合は、デフォルト以外のポートを使用するときか、SSL セキュリティを使用するときにのみ、このファイルを編集します。
- リモートマシンに .NET アセンブリがある場合は、このファイルにアセンブリを登録して、ColdFusion からアクセスできるようにします。

設定ファイルの編集

1 <configuration> セクションの <configSections> サブセクションに、次の行があることを確認します。

```
<jnbridge>
    <javaToDotNetConfig scheme="Protocol" port="local port number"
        useSSL="true|false" certificateLocation="server certificate path"/>
</jnbridge>
```

    - scheme 属性では通信プロトコルを指定します。jtcp または http であることが必要です。
    - ポート番号は、.NET サイドエージェントのポートです。通常は 6086 です。
    - useSSL 属性では、SSL を使用して通信を保護するかどうかを指定します。この属性はオプションです。デフォルトでは SSL を使用しません。
    - certificateLocation 属性では、サーバーの SSL 証明書の場所を指定します。これが必要なのは、useSSL 属性が true の場合のみです。

    これらの設定は、cfobject タグの対応する属性と同じである必要があります。

2 リモートシステムに .NET アセンブリがある場合は、<jnbridge> セクションに次の要素を追加して、ColdFusion からアクセスするアセンブリを指定します。

```
<assemblyList>
    <assembly file="path to assembly or fully qualified name"/>
    ...
</assemblyList>
```

3 .NET サイドエージェントが実行されている場合は、停止して再起動します。たとえば、ColdFusion システムで、ColdFusion .NET サービスを再起動します。これで、設定した .NET クラスに ColdFusion アプリケーションからアクセスすることができます。

次の例は、単純な "JNBDotNetSide.exe.config" ファイルで、.NET サイドの TCP サーバー設定を指定しています。サーバーは TCP バイナリモードを使用して通信し、ポート 6086 でリスンします。Java クライアントは ¥¥x にアクセスできます。

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <sectionGroup name="jnbridge">
        <section name="dotNetToJavaConfig"
                type="System.Configuration.SingleTagSectionHandler"/>
        <section name="javaToDotNetConfig"
                type="System.Configuration.SingleTagSectionHandler"/>
        <section name="tcpNoDelay"
                type="System.Configuration.SingleTagSectionHandler"/>
        <section name="javaSideDeclarations"
                type="System.Configuration.NameValueSectionHandler"/>
        <section name="assemblyList"
                type="com.jnbridge.jnbcore.AssemblyListHandler, JNBShare"/>
    </sectionGroup>
    <jnbridge>
        <javaToDotNetConfig scheme="jtcp" port="6086"/>
        <assemblyList>
            \\x
            <assembly file="System.Windows.Forms, Version=1.0.5000.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
        </assemblyList>
    </jnbridge>
</configuration>
```

.NET クラスの使用

.NET アセンブリクラスの使用方法は、cfobject タグや CreateObject 関数を使用して作成する Java やその他のオブジェクトと同じです。最も単純なケースでは、次のようなコードでローカルの .NET クラスメソッドをロードできます。

```
<cfobject type = ".NET" name = "mathInstance" class = "mathClass"
        assembly = "C:/Net/Assemblies/math.dll">
<cfset myVar = mathInstance.multiply(1,2)>
```

CFScript と CreateObject 関数を使用する場合は、次のようにします。

```
<cfscript>
    mathInstance = CreateObject(".NET", "mathClass",
        "C:/Net/Assemblies/math.dll");
    myVar = mathInstance.multiply(1,2);
</cfscript>
```

注意：同じ完全修飾名を持つ 2 つの DLL をロードすることはできません。サーバーが再起動するまで、最初にアクセスされた DLL のみが使用されます。たとえば、"page1.cfm" で "c:¥dev¥a.dll" を使用し、"page2.cfm" で "c:¥dev2¥a.dll" を使用し、いずれの DLL も完全修飾名が同じである場合は、最初にロードされた DLL ファイルが両方の CFML ページで使用され続けます。

オブジェクトを作成してクラスメソッドやフィールドにアクセスし、ColdFusion と .NET の間でデータ型を変換する場合は、次の注意事項と制限を考慮してください。

- 1144 ページの「.NET と ColdFusion の間でのデータ型の変換」で説明されている、データ型の変換に関する注意事項
- 『CFML リファレンス』の cfobject: .NET object の「制約」セクションに記載されている制約

オブジェクトのインスタンス化とクラスコンストラクタの呼び出し

cfobject タグを使用して .NET オブジェクトを作成しても、オブジェクトのインスタンスは作成されません。オブジェクトインスタンスは次のいずれかの方法で作成されます。

- クラスにデフォルトコンストラクタがある場合は、オブジェクトのスタティックでないメソッドを最初に呼び出したときに、そのコンストラクタが自動的に呼び出されます。
- クラスにデフォルトコンストラクタがない場合や、クラスに複数のコンストラクタがありデフォルト以外のコンストラクタを使用する場合は、ColdFusion オブジェクトの特別な init メソッドを呼び出します。すべてのクラスコンストラクタに対応する init メソッドが cfobject タグによって自動的に作成されます。init メソッドを実行すると、パラメータの数と型が一致するクラスコンストラクタが呼び出されます。たとえば次のタグでは、2 つの整数パラメータを取る MyClass コンストラクタが呼び出されます。

```
<cfobject type=".NET" name="myObj" class="com.myCo.MyClass"
        assembly="c:\assemblies\myLib.dll">
<cfset myObj.init(10, 5)>
```

注意：スタティックメソッドのみを使用する場合、オブジェクトのインスタンスは作成されません。

メソッドの呼び出し

.NET メソッドは、通常の ColdFusion オブジェクトのメソッドと同じ方法で使用します。たとえば、MyClass クラスに、数値 ID を受け取って名前を返す getName メソッドがある場合は、次のようにして呼び出します。

```
<cfset theID="2343">
<cfset userName=mObj.getName(theID)>
```

フィールドの取得と設定

.NET クラスのパブリックフィールドへのアクセスや変更を行うには、次のメソッドを呼び出します。

```
Get_fieldName()
Set_fieldName(value)
```

たとえば、.NET クラスに accountID というパブリックフィールドがある場合は、次のように、Get_accountID() と Set_accountID() メソッドを使用して、その値へのアクセスや変更を行うことができます。

```
<cfobject type=".NET" class="com.myCo.MyClass"
        assembly="c:\assemblies\myLib.dll" name="myObj">
<cfset theAccount=myObj.Get_accountID()>
<cfset myObj.Set_accountID(theAccount + 1)>
```

final フィールドは、アクセスのみが可能で変更は行えません。したがって、呼び出せるのは Get_**fieldName**() のみになります。

.NET と ColdFusion の間でのデータ型の変換

.NET クラスにアクセスするには、ColdFusion システムに Java プロキシがあり、ターゲットシステムに .NET コードがあることが必要です。したがって、ColdFusion、Java、.NET (厳密には Microsoft Intermediate Language、略して MSIL) の間でデータ型を変換する必要があります。データ型の変換は ColdFusion によって自動的に行われます。通常、正しい変換が行われるように特別な処理を行う必要はありません。ただし、変換にはいくつかの制限があるので、.NET プロキシオブジェクトのメソッドを呼び出すときに、データ型を明示的に指定することが必要になる場合もあります。

次のセクションでは、データ変換の問題とその対処方法について説明します。ColdFusion データ、Java データ型、.NET データ型の間で行われる変換の詳細については、『CFML リファレンス』の cfobject: .NET object を参照してください。

データ型変換のルールとテクニック

ColdFusion は、ColdFusion、Java、CLR の各データ型のデータを自動的に変換します。次の表は、.NET Common Language Runtime (CLR) のプリミティブおよび標準データ型、CLR データ型を表すためにプロキシで使用される Java データ型、および CFML アプリケーションの ColdFusion データ型の間の変換方法を示したものです。

.NET 型	Java 型	ColdFusion 型
sbyte	byte	整数
byte	short	整数
short	short	整数
ushort	int	整数
int	int	整数
uint	long	数値
char	char	整数または文字列
long	long	数値
ulong	float	数値
float	float	数値
double	double	数値 返される数値は、ColdFusion で通常表示されるより高い精度を維持しています。double 型の完全な精度の戻り値を取得して表示するには、PrecisionEvaluate 関数を使用します。double の完全な精度の値を .NET のメソッドに渡すこともできます。
bool	boolean	ブール値
enum		変換されません。ただし、列挙子の要素には、Enumerator_variable.enumerator の形式 (たとえば MyColor.Red) を使用して直接アクセスできます。
array	array	配列
string	String	文字列
System.Collections.ArrayList	java.util.ArrayList	配列 **メモ :** ColdFusion は、.NET 型から ColdFusion 型への変換のみを行います。ColdFusion の配列から .NET の ArrayList への変換は行いません。
System.Collections.Hashtable	java.util.Hashtable	構造体 **メモ :** ColdFusion は、.NET 型から ColdFusion 型への変換のみを行います。ColdFusion の構造体から .NET の Hashtable への変換は行いません。
System.Data.DataTable		クエリー **メモ :** ColdFusion は、.NET 型から ColdFusion 型への変換のみを行います。ColdFusion のクエリーから .NET の DataTable への変換は行いません。

.NET 型	Java 型	ColdFusion 型
System.DateTime	java.util.Date	日付時刻値
decimal System.Decimal	java.math.BigDecimal	10 進数値の文字列表現 10 進数値の使用方法の詳細については、1146 ページの「10 進数の使用」を参照してください。
System.Object		.NET の引数が System.Object 型の場合、ColdFusion の文字列は直接変換されます。それ以外の型の場合は、JavaCast 関数を使用する必要があります。 ColdFusion は .NET メソッドによって返される System.object インスタンスを ColdFusion の型には変換できませんが、Object メソッドを使用するとインスタンスにアクセスできます。 詳細については、1146 ページの「System.Object 型へのデータの変換」を参照してください。

10 進数の使用

JavaCast 関数を使用して ColdFusion のデータを BigDecimal 形式に変換してから、.NET 関数に値を渡す必要があります。次に例を示します。

```
<cfset netObj.netFunc(javacast("bigdecimal","439732984732048"))>
```

ColdFusion は自動的に、返される 10 進数および System.Decimal の値を ColdFusion の文字列表現に変換します。

10 進数および日付時刻値の確実な変換

ColdFusion は、System.Decimal 用のプロキシが値タイプのプロキシの場合にのみ、.NET の 10 進値または System.Decimal 型を変換します。同様に、System.DateTime 用のプロキシが値タイプのプロキシの場合にのみ、.NET の System.DateTime 値を ColdFusion の日付時刻値に変換します。ColdFusion サーバーは、これらのプロキシを生成するときは常に値プロキシを使用します。ただし、JNBProxyGUI.exe ツールを使用してプロキシを生成する場合は、System.Decimal に対するプロキシを値タイプとして生成するように注意してください。

System.Object 型へのデータの変換

.NET メソッドで引数の型として (System.Boolean などの特定の Object サブクラスではなく) System.Object が指定されていて、そのようなメソッドに引数としてプリミティブ値を渡したい場合は、javacast 関数を使用してデータ変換を指定します。ColdFusion は、データ型を認識すると、適切な .NET 型に自動的に変換します。次の表は、ColdFusion 型から .NET 型への変換ルールです。

.NET 型	javacast で使用する型
bool / System.Boolean	boolean
bool[] / System.Boolean[]	boolean[]
char / System.Char	char
char[] / System.Char[]	char[]
double / System.Double	double
double[] / System.Double[]	double[]
float / System.Single	float
float[] / System.Single[]	float[]
int / System.Int32	int
int[] / System.Int32[]	int[]

long / System.Int64	long
long[] / System.Int64[]	long[]
sbyte / System.Sbyte	byte
sbyte []/ System.Sbyte[]	byte []
short / System.Int16	short
short[] / System.Int16[]	short[]
System.Decimal	bigdecimal
System.String	String

注意：ColdFusion の文字列変数を変換するのに、JavaCast 関数を使用する必要はありません。自動的に .NET の System.String に変換されます。

byte (unsigned byte)、ushort (unsigned short)、uint (unsigned int)、ulong (unsigned long) などの .NET プリミティブの符号なしデータ型の場合は、対応する Java 型がないので、特殊なオブジェクトを作成する必要があります。次の表は、.NET のプリミティブ型と、それに対応するものとして使用する必要のあるクラスの一覧です。

.NET 型	cfobject/createObject で使用するクラス
byte / System.Byte	System.BoxedByte
ushort / System.UInt16	System.BoxedUShort
uint / System.UInt32	System.BoxedUInt
ulong / System.UInt64	System.BoxedULong

これらの特殊オブジェクトを代入ステートメントで使用するには、その前に、他の .NET クラスを作成する場合と同じように、createObject 関数または cfobject タグを使用してオブジェクトを作成します。たとえば次の行では、値 100 の ushort 表現が作成されます。

```
<cfset boxedUShort = createObject(".NET". "System.BoxedUShort").init(100)>
```

次の例では、System.Hashtable オブジェクトを作成し、それに全種類のプリミティブの例を設定します。

```
<!--- create a .NET Hashtable --->
<cfset table = createObject(".NET", "System.Collections.Hashtable")>

<!--- call HashTable.add(Object, Object) method for all primitives --->
<cfset table.add("shortVar", javacast("short", 10))>
<cfset table.add("sbyteVar", javacast("byte", 20))>
<cfset table.add("intVar", javacast("int", 123))>
<cfset table.add("longVar", javacast("long", 1234))>
<cfset table.add("floatVar", javacast("float", 123.4))>
<cfset table.add("doubleVar", javacast("double", 123.4))>
<cfset table.add("charVar", javacast("char", 'c'))>
<cfset table.add("booleanVar", javacast("boolean", "yes"))>
<cfset table.add("StringVar", "Hello World")>
<cfset table.add("decimalVar", javacast("bigdecimal", 123234234.505))>

<!--- call HashTable.add(Object, Object) for unsigned primitive types. --->
<cfset boxedByte = createObject(".NET", "System.BoxedByte").init(10)>
<cfset table.add("byteVar", boxedByte)>

<cfset boxedUShort = createObject(".NET", "System.BoxedUShort").init(100)>
<cfset table.add("ushortVar", boxedUShort)>

<cfset boxedUInt = createObject(".NET", "System.BoxedUInt").init(123)>
<cfset table.add("uintVar", boxedUInt)>

<cfset boxedULong = createObject(".NET", "System.BoxedULong").init(123123)>
<cfset table.add("ulongVar", boxedULong)>

<cfdump var="#DotNetToCFType(table)#">
```

他の .NET オブジェクトは、そのまま渡すことができます。

あいまいな型変換の解決

パラメータ数が同じでパラメータのデータ型のみが異なる複数のシグネチャが 1 つのメソッドで定義されている場合、ColdFusion は正しいデータ型変換を判断できません。この場合は、JavaCast メソッドを使用して、.NET 型に対応する Java 型に ColdFusion データを変換します。

たとえば、.NET クラスに myFunc(ulong) および myFunc(int) というメソッドがある場合は、次のように JavaCast メソッドを使用して、ColdFusion 変数を Java float または int データ型に変換します。

```
myFunc(JavaCast(int, MyVar));
```

同様に、.NET クラスに myFunc(int) および myFunc(String) というメソッドがある場合は、次のように JavaCast メソッドを使用して、ColdFusion 変数を Java int または String データ型に変換します。

```
myFunc(JavaCast(String, "123");
```

場合によっては、1 つの Java 型が複数の .NET 型に対応しているために、JavaCast 関数であいまいさを解決できないことがあります。この場合は、Java 型に直接対応する .NET データ型が使用されているメソッドのみを持つプロキシが作成されます。

たとえば、.NET クラスに myFunc(ulong) および myFunc(float) というメソッドがある場合、生成されるプロキシは 1 つのメソッドのみを持ちます。ColdFusion の浮動小数点数の処理に使用される Java float 型に直接対応するのは .NET float 型なので、このメソッドは myFunc(float) を呼び出します。この場合、.NET myFunc(ulong) メソッドを呼び出すことはできません。

複合 .NET データ型の処理

Hashtable、ArrayList、DataTable などの複合 .NET データを使用するときは、通常、ColdFusion が自動的に、構造体、配列、クエリーなどの対応する ColdFusion データ型にデータを変換します。このようなデータを使用するときは、次に示すような特別なアクションを実行して、データの適切なアクセスと変換を有効にします。

- ColdFusion から .NET Hashtable データに適切にアクセスするには、連想配列表記法を使用します。
- Hashtable、ArrayList、または DataTable を入力として受け取るパラメータで ColdFusion の変数を直接使用することはできません。
- 複合 .NET データから ColdFusion 型への自動的な変換は無効にできます。
- 複合 .NET データは、ColdFusion 型に手動で変換できます。

ColdFusion での Hashtable データの使用

.NET Hashtable は大文字と小文字を区別しますが、ColdFusion 構造体アクセスのほとんどのメソッドは、大文字と小文字を区別しません。**structName**["**keyName**"] の形式の連想配列表記法だけが、大文字と小文字を区別します。.NET Hashtable が CF の構造体に変換されるときは、要素のキーの大文字と小文字が異なるだけであっても、データセット全体が変換されます。したがって、大文字と小文字が異なるだけのキーの値を取得するには、連想配列表記法を使用します。

次の例はこの問題を示しています。キー値の大文字と小文字が異なるだけの 3 つのエントリを持つ Hashtable オブジェクトを作成します。この例では、ドット区切りの構造体表記法を使用した出力は、すべてが大文字のキーに対応して、常に同じ値を返しますが、連想配列表記法は正しい結果を返します。

```
<!--- Create a Hashtable and convert it to a ColdFusion structure. --->
<cfset table = createObject(".NET", "System.Collections.Hashtable")>
<cfset table.add("Key", "Value1")>
<cfset table.add("KEY", "Value2")>
<cfset table.add("key", "Value3")>
<cfset cftable = DotNetToCFType(table)>

<cfdump var="#cftable#">

<h3>Using dot notation</h3>
Key : <cfoutput>#cftable.Key#</cfoutput><br>
KEY : <cfoutput>#cftable.KEY#</cfoutput><br>
key : <cfoutput>#cftable.key#</cfoutput><br>
<p>

<h3>Using associative array notation</h3>
Key : <cfoutput>#cftable["Key"]#</cfoutput><br>
KEY : <cfoutput>#cftable["KEY"]#</cfoutput><br>
key : <cfoutput>#cftable["key"]#</cfoutput><br>
```

ColdFusion での .Net ArrayList の使用

ColdFusion は System.Collections.ArrayList オブジェクトを ColdFusion の配列に変換し、すべての標準 ColdFusion 配列操作をこれに対して実行できます。この場合の例を次に示します。

.Net のコード :

```
public ArrayList getList(){
    ArrayList myAL = new ArrayList();
    myAL.Add("Hello");
    myAL.Add(1);
    myAL.add(true);
    Return AL;
}
```

ColdFusion のコード :

```
<cfset cflist = netObject.getList()>
<cfloop array="#cflist#" index="item">
    <cfoutput>#item#</cfoutput><br>
</cfloop>

<cfif cflist[3]>
        <cfoutput>Third element in the list is true</cfoutput>
</cfif>
```

ColdFusion での ADO.Net DataTable の使用

ColdFusion は System.Data.DataTable オブジェクトを ColdFusion のクエリーオブジェクトに変換し、すべての標準 ColdFusion クエリー操作をこれに対して実行できます。この場合の例を次に示します。

.Net のコード :

```
public DataTable datasetMethod()
{
    //conn string
    string connectionString = "...";

    //connection
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        SqlCommand cmd = new SqlCommand(@"SELECT * FROM [tblEmployees]", connection);
        connection.Open();
        SqlDataReader reader = cmd.ExecuteReader();
        DataTable dt = new DataTable();
        dt.Load(reader);
        return dt;
}
}
```

ColdFusion のコード :

```
<cfset query1 = netObject.datasetMethod()>
<cfoutput query="query1">
    Query1.CurrentRow = #query1.CurrentRow#<br>
</cfoutput>
```

.NET 入力パラメータでの ColdFusion 複合型の使用

.NET のメソッドが ArrayList、Hashtable、または DataTable を返すと、ColdFusion はそれを ColdFusion の配列、構造体、またはクエリーに自動的に変換します。一方、ColdFusion のデータ型から .NET 型への変換は自動的には行われません (ColdFusion の配列から .Net の配列型への変換は自動的に行われます)。したがって、.NET の System.Collection.ArrayList 型、System.Collection.Hashtable 型、または System.Data.DataTable 型を必要とする .NET オブジェクトのインスタンスのメソッドに対する入力パラメータとして、ColdFusion の変数を直接使用することはできません。代わりに、これらの .NET 型のインスタンスを作成し、必要なデータを設定した後、それを .NET メソッドに渡します。System.Collection.Hashtable オブジェクトを作成して設定する例については、「System.Object 型へのデータの変換」セクションの最後の例を参照してください。

複合 .NET データの自動変換の無効化

.NET の System.Collections.Hashtable オブジェクト、System.Collections.ArrayList オブジェクト、または System.Data.DataTable オブジェクトから対応する ColdFusion の構造体、配列、またはクエリーオブジェクトへの自動変換を無効にできます。次のような場合には変換を無効にすることを推奨します。

- .NET のメソッドによって返されるコレクションまたは DataTable が大きく、そのデータのごく一部だけが必要な場合。自動変換が有効になっていると、ColdFusion はすべてのオブジェクトのフィールドを含むデータ構造体を作成します。構造体を作成すると、各フィールドを取得するために ColdFusion が .NET のメソッドを内部的に呼び出す必要があり、

多くの時間とリソースが必要です。自動変換を無効にすると、他のオブジェクトと同じように NET オブジェクトからフィールドやデータを取得できます。

- 複合変数を返す .NET メソッドを呼び出し、次にその変数を引数として別の .NET メソッドに渡す場合。自動変換が有効になっている場合は、最初のメソッドからの Hashtable オブジェクトを次のメソッドに直接渡すことはできません。

自動変換を無効にするには、JVM の coldfusion.dotnet.disableautoconversion システムプロパティを true に設定します。たとえば、ColdFusion スタンドアローンサーバーで、または JRun を J2EE サーバーとして使用している場合は、次の設定を "JVM.config" ファイルに追加します。

```
-Dcoldfusion.dotnet.disableautoconversion=true
```

複合 .NET オブジェクトの手動変換

次のいずれかの場合に、System.Collections.Hashtable、System.Collections.ArrayList、または System.Data.DataTable オブジェクトを ColdFusion の構造体、配列、またはクエリーに変換するには、DotNetToCFType 関数を使用します。

- coldfusion.dotnet.disableautoconversion システムプロパティを true に設定してある。
- 自動変換が有効であり、createObject 関数または cfobject タグを使用して作成した複合 .NET オブジェクトを、対応する ColdFusion 表現に変換したい。

関数の使用方法の例については、『CFML リファレンス』の DotNetToCFType を参照してください。

.NET オブジェクトの使用

クラス型の .NET フィールドや戻り値は、ColdFusion では .NET オブジェクトとして使用できます。オブジェクトのメソッドを使用してオブジェクトのデータにアクセスし、サポートされているデータ型を使用して、ColdFusion で使用できるように処理します。

次の例では、システムのドライブに関する情報を取得します。System.IO.DriveInfo.GetDrives() メソッドを呼び出して、ドライブごとに 1 つの System.IO.DriveInfo オブジェクトが格納された配列を取得します。次に、オブジェクトメソッドを呼び出して、ドライブに関する特定の情報を取得し、情報を表示します。この例では、cfdump タグを使用して、コードを単純化しています。

注意：System.IO.DriveInfo は .NET 1.x フレームワークには含まれていません。.NET 2.0 以降のフレームワークに含まれています。.NET フレームワークの選択の詳細については、1158 ページの「.NET のバージョンの選択と変更」を参照してください。

```
<!--- Create a query for the drive information results. --->
<cfset result=QueryNew("name,type,isready,format,label,totalsize,freespace"
                    ,"varchar,varchar,bit,varchar,varchar,double,double")>
<!--- Create a .NET System.IO.DriveInfo object. --->
<cfobject type=".NET" name="sidiClass" class="System.IO.DriveInfo">
<!--- Get the drives. --->
<cfset drives=sidiClass.GetDrives()>
<!--- Loop through drives. --->
<cfloop from="1" to="#ArrayLen(drives)#" index="i">
    <!--- Add a row to the query.--->
    <cfset QueryAddRow(result)>
    <!--- Get the drive name, type, and ready flag. --->
    <cfset QuerySetCell(result, "name", drives[i].Get_Name())>
    <cfset QuerySetCell(result, "type",
                drives[i].Get_DriveType().ToString())>
    <cfset QuerySetCell(result, "isready", drives[i].Get_IsReady())>
    <!--- Get extra details ONLY if the drive is ready. --->
    <cfif drives[i].Get_IsReady()>
        <cfset QuerySetCell(result, "format", drives[i].Get_DriveFormat())>
        <cfset QuerySetCell(result, "label", drives[i].Get_VolumeLabel())>
        <cfset QuerySetCell(result, "totalsize", drives[i].Get_TotalSize())>
        <cfset QuerySetCell(result, "freespace",
                    drives[i].Get_AvailableFreeSpace())>
    </cfif>
</cfloop>
<cfdump var="#result#">
```

.NET の相互運用性の制限

ColdFusion .NET の相互運用性には、次のような制限があります。

- 引数または戻り値の型としてポインタを使用してメソッドを呼び出すことはできません。
- Out パラメータを取るメソッドを呼び出すことはできません。
- ColdFusion は、System.Data.DataTable、System.Collection.Hashtable、および System.Collection.ArrayList から ColdFusion のデータ型への変換のみを行うことができます。ColdFusion のクエリー、構造体、および配列を System データ型に変換することはできません。ただし、ColdFusion の配列を CLR の配列型に変換することはできます。したがって、構造体またはクエリーを .NET メソッドに直接渡すことはできません。
- .NET ユーザーインターフェイスコンポーネントにはアクセスできません。
- .NET 側からコールバック (イベントと委譲) を使用することはできません。
- パラメータ数が同じでパラメータのデータ型のみが異なる複数のシグネチャが 1 つのメソッドで定義されている場合、ColdFusion は正しいデータ型変換を判断できません。この場合は、JavaCast メソッドを使用して、.NET 型に対応する Java 型に ColdFusion データを変換します。
- 1 つの Java 型が複数の .NET 型に対応しているために、同じ数のパラメータを持つ関数の間のあいまいさを JavaCast 関数が解決できない場合は、Java 型に直接対応する .NET データ型を使用する 1 つのプロキシが作成されます。

  あいまいな型変換の処理方法の詳細については、1144 ページの「.NET と ColdFusion の間でのデータ型の変換」を参照してください。
- "DotNetSide.exe.config" ファイルに登録されるアセンブリは、クラス名が一意である必要があります。複数のアセンブリに同じクラス名があると、メソッドの呼び出しがエラーになったり、正しくない結果になります。たとえば、a.dll と b.dll という 2 つの DLL で、同じクラス名 nam1.name2.MyClass を持つことはできません。ColdFusion と .NET が同じマシン上に存在する場合に、ある DLL を使用した後で最初の DLL と同じクラス名を含む別の DLL を使用したい場合は、ColdFusion .NET サービスを再起動する必要があります。異なるマシンに存在する場合は、最初の DLL のエントリを "DotNetSide.exe.config" ファイルから削除した後、.NET サービスをホストする Windows マシンで ColdFusion .NET サービスを再起動します。

サンプルアプリケーション

最初のサンプルアプリケーションでは、Microsoft .NET システムのクラスメソッドを直接使用します。2 番目のサンプルアプリケーションでは、カスタム C# クラスを使用して、Microsoft Word にアクセスします。

例 : .NET クラスを直接使用

次の例では、Microsoft .NET の System.Net.NetworkInformation.Ping クラスメソッドを直接使用して、サーバーに ping を行います。このクラスは、.NET バージョン 2.0 以降でサポートされています。

```
 <!--- This function pings the specified host. --->
<cffunction name="Ping" returntype="string" output="false">
    <cfargument name="host" type="string" required="yes">
    <!--- Local variables --->
    <cfset var pingClass="">
    <cfset var pingReply="">
    <!--- Get Ping class --->
    <cfobject type=".NET" name="pingClass"
            class="System.Net.NetworkInformation.Ping">
    <!--- Perform synchronous ping (using defaults) --->
    <cfset pingReply=pingClass.Send(Arguments.host)>
    <!--- Return result --->
    <cfreturn pingReply.Get_Status().ToString()>
</cffunction>

<h3>Ping Test</h3>
<cfoutput>
    127.0.0.1: #Ping("127.0.0.1")#<br>
    www.adobe.com: #Ping("www.adobe.com")#<br>
</cfoutput>
```

例 : カスタムクラスによる Microsoft Word へのアクセス

次の ColdFusion アプリケーションでは、カスタム C# WordCreator クラスと、Microsoft Office および Word DLL のサポートクラスを使用して、Word ドキュメントを作成します。Microsoft Word を開き、someText 変数に指定したテキストのコピーを 5 つ書き込み、filename 変数で指定したファイルにドキュメントを保存します。Word のインスタンスは開いたままにします。

注意 : .NET System クラスを直接使用し、その他の .NET コードを必要としないサンプルについては、『CFML リファレンス』の cfobject: .NET object の「制約」セクションを参照してください。

2 番目に示されているコードは、WordCreator C# のソースコードです。このアプリケーションをローカルで実行するには、このクラスと "Microsoft Interop.Word.dll" ファイルをコンパイルして、"C:¥dotnet" ディレクトリに置きます。または、任意の場所に置いて、cfobject の assembly 属性でパス指定を変更する方法もあります。インストールしてある Microsoft Office のバージョンによっては、追加の Microsoft DLL ファイルや別の DLL ファイルが必要になる場合もあります。

ColdFusion アプリケーションのコードは、次のとおりです。

```
 <cfset filename="C:\dotNet\demo.doc">
<cfif fileexists(filename)>
    <cffile action="delete" file="#filename#">
</cfif>
<cfobject type=".NET" assembly="C:\dotNetApp\WordApp.dll,C:\dotNet\Interop.Office.dll" name="wordCreator"
class="WordApp.WordCreator">
<cfset wordCreator.init("#filename#")>
<cfdump label="WordCreator Class Dump" var="#wordCreator#">

<cfset someText = "ColdFusion created this sample document using Windows .NET class methods. The text is
long enough to appear in the Word file on multiple lines.">

<cfloop from=1 to=5 index =i>
    <cfset wordCreator.addText(someText)>
    <cfset wordCreator.newParagraph()>
    <cfset wordCreator.newParagraph()>
    <cfset wordCreator.addText("Starting a new paragraph. It starts a
            a new line.")>
    <cfset wordCreator.newParagraph()>
    <cfset wordCreator.newParagraph()>
</cfloop>
<cfset wordCreator.save()>
```

WordCreator クラスの C# ソースは次のとおりです。

```
 using System;
using System.IO;
using System.Collections.Generic;
using System.Text;

// The commented-out lines could be needed on some systems in place of,
// or in addition to, the line that follows them.
// using Microsoft.Office.Core;
// using Word;
// using Microsoft.Office;
// using Microsoft.Office.Interop;
using Microsoft.Office.Interop.Word;
namespace WordApp {
    public class WordCreator {
        object readOnly = false;
        object isVisible = true;
        object missing = System.Reflection.Missing.Value;
        object docType = 0;
        object newTemplate = false;
        object template = "normal.dot";
        object format = WdSaveFormat.wdFormatDocument;
        ApplicationClass app = new ApplicationClass();
        private object fileName;
        private Document doc;
        private bool isNewDoc = false;

            public WordCreator(String fileName) {
            this.fileName = fileName;
            app.Visible = true;
            if (File.Exists(fileName))
                doc = app.Documents.Open(ref this.fileName, ref missing, ref
                    readOnly, ref missing, ref missing, ref missing, ref missing,
                    ref missing, ref missing, ref missing, ref missing, ref
                    isVisible, ref missing, ref missing, ref missing, ref missing);
            else {
                doc = app.Documents.Add(ref template, ref newTemplate,
                    ref docType, ref isVisible);
                isNewDoc = true;
```

```
            }
            doc.Activate();
        }

        public void addText(String text) {
            app.Selection.TypeText(text);
        }

        public void newParagraph() {
            app.Selection.TypeParagraph();
        }

        public void save() {
            if(!isNewDoc)
                doc.Save();
            else doc.SaveAs(ref fileName, ref format, ref missing, ref missing,
                ref missing, ref missing, ref missing, ref missing, ref missing,
                ref missing, ref missing, ref missing, ref missing, ref missing,
                ref missing, ref missing);
        }
    }
}
```

高度なツール

ワークフローによっては、プロキシの生成と .NET Extension ソフトウェアの実行を行う追加のツールが役立つ場合があります。

jnbproxy コマンドの使用

jnbproxyGui プログラムの代わりに jnbproxy コマンドラインツールを使用して、Java プロキシを生成することもできます。詳細については、1140 ページの「Java プロキシクラスの生成」を参照してください。

たとえば、バッチファイルで jnbproxy コマンドを使用して、1 回の操作で複数のプロキシ JAR ファイルを生成できます。

jnbproxy コマンドは次の形式で指定します。

```
jnbproxy options... classes...
```

次に例を示します。

```
jnbproxy /al C:\dotNet\netdll\PrimitiveTypes.dll /d C:\dotNet\MyJavajars
    /host localhost /n PrimitiveTypes /nj /pd j2n /port 6085 /pro b
    /pp C:\ColdFusion8\lib CSharpDatatypes.PrimitiveTypes
```

オプション

次の表に、使用できるオプションを示します。ColdFusion を実行しているシステムでプロキシを作成するには、/nj オプションを使用し、/bp、/java、/jp オプションは指定しません。

オプション	必須 / オプション	デフォルト	説明
/al **assemblylist**	必須		必要な .NET クラスが含まれている .NET アセンブリ (DLL と EXE ファイル) のパスを、セミコロンで区切って指定します。
/bp **bcelpath**	オプション	CLASSPATH 環境変数を使用してファイルを探します。	"bcel.jar" ファイルが含まれているフォルダのパスを指定します。 /nj オプションを使用する場合は無視されます。
/cf	必須		ColdFusion のソフトウェアライセンスを使用します。このオプションを含めないと、プロキシは 30 日間のトライアル期間に制限されます。

オプション	必須 / オプション	デフォルト	説明
/d **directory**	オプション	現在の実行ディレクトリ。	生成したプロキシを含む JAR ファイルを書き込むディレクトリを指定します。
/f **classfile**	オプション		コマンドラインからではなく、指定したテキストファイルからクラスを読み取ります。 詳細については、JNBridge のマニュアルを参照してください。
/h	オプション		オプションと使用方法をリストします。オプションや引数を指定せずに jnbproxy コマンドを入力しても、同じ情報を取得できます。
/host **hostname**	必須		.NET コードがあるホストを指定します。このオプションは、ホスト名と IP アドレスのいずれでも指定できます。通常は、localhost を指定します。
/java **javapath**	オプション	システムの PATH 環境変数を使用して最初に見つかった "java.exe" ファイルを使用します。	Java が自動的に開始されたときに使用する "java.exe" プログラムが含まれているディレクトリのパスを指定します。 /nj オプションを使用する場合は無視されます。
/jp **jnbcorepath**	オプション	CLASSPATH 環境変数を使用します。	"jnbcore.jar" ファイルが含まれているフォルダのパスを指定します。 /nj オプションを使用する場合は無視されます。
/ls	オプション	プロキシを生成してリストします。	指定のクラスをサポートするために生成されるすべてのクラス (1141 ページの「サポートクラス」を参照してください) をリストしますが、プロキシは生成しません。
/n **name**	オプション	"jnbproxies.jar" というファイルを作成します。	プロキシを含める JAR ファイルの名前を指定します。.jar 拡張子はツールによって自動的に追加されるので、指定しないでください。
/nj	オプション	Java を自動的に開始します。	Java を自動的に開始しません。このオプションを使用する場合は、Java が実行されている必要があります。/bp、/java、/jp、/wd オプションは指定しても無視されます。
/ns	オプション	すべてのサポートクラスのプロキシを生成します。	すべてのサポートクラスではなく、コマンドライン (またはクラスファイル) で指定されているクラスに対してのみプロキシを生成します。
/pd	必須		プロキシの方向を指定します。j2n に設定する必要があります。
/port **portNum**	必須		プロキシを生成するときに .NET サイドでリスンするポートを指定します。整数であることが必要です。通常、この値は 6085 です。
/pro **protocol**	必須		.NET サイドと Java サイドの間の通信メカニズムを指定します。使用できる値は次のとおりです。 • b : TCP/ バイナリ • h : HTTP/SOAP
/wd **dir**	オプション	システムのデフォルトの作業ディレクトリ。	JVM の作業ディレクトリを指定します。 /nj オプションが指定されている場合は無視されます。

クラス
プロキシを生成する .NET クラスの完全修飾名 (たとえば CSharpDatatypes.PrimitiveTypes) をスペースで区切ったもの。System.Object と System.Type に対するプロキシは、クラスリストに含まれていなくても、常に生成されます。

データの参照渡しと値渡し
プロキシ生成ツールでは、パラメータや戻り値を参照渡しにするか値渡しにするかを指定できます。

参照渡しと値渡しについて

参照渡しの場合、Java プロキシと .NET サイドの間で送信される情報は、元の .NET オブジェクトへの論理ポインタであり、送信後も .NET サイドに存在しています。値渡しの場合、送信される情報は .NET オブジェクトの内容のコピーであり、関数呼び出しの後も .NET サイドに存在できます。参照渡しと値渡しには、それぞれ異なる利点があります。

参照渡しの場合、Java プロキシと .NET オブジェクトとの間で直接渡されるのは、変更された値のみです。他のすべての情報は、対応するオブジェクト内の情報への参照として渡されます。一般に、参照は実際のオブジェクトよりも小さくなるので、多くの場合、参照渡しは高速です。また、.NET オブジェクトへの参照ポイントは .NET サイドに存在し続けるので、その .NET オブジェクトが更新された場合は、Java サイドのプロキシオブジェクトから更新内容に直ちにアクセスできます。参照プロキシの短所は、元のオブジェクトのデータ (フィールドやメソッドなど) にアクセスする場合に、Java サイドから .NET サイド (情報が存在する場所) にアクセスして Java サイドに戻るという往復が必要になる点です。

値渡しの場合は、データのコピーが .NET と Java との間で渡されます。一般に、データオブジェクトは参照よりも大きいので、オブジェクトの値渡しは参照渡しよりも時間がかかります。また、渡される値は、その時点でのオブジェクトのスナップショットです。渡されたオブジェクトと元の .NET オブジェクトの間には関連付けがないので、オブジェクトが渡された後に元の .NET オブジェクトが更新されても、渡された値が更新されることはありません。値プロキシの利点は、オブジェクト内のすべてのデータが Java サイドのローカルに存在しているので、データを取得するために .NET サイドにアクセスして戻ってくるという往復が発生せず、フィールドへのアクセスが高速である点です。

参照プロキシと値プロキシのどちらを使用するかは、プロキシのセマンティクスとパフォーマンスによって異なります。

- 一般的には、参照プロキシを使用します (デフォルト)。これは、Java と C# の通常のパラメータ受け渡しセマンティクスを保持するためです。
- 次のような場合は、一般的に値プロキシを使用します。
  - クラス関数が、パラメータ値を受け取って値を返すという往復を必ず行う場合。
  - クラスオブジェクトにほとんどデータが含まれていない場合。
  - オブジェクトデータが頻繁に変更される場合で、オブジェクトが比較的小さいか、またはデータのアクセス間隔のほうがオブジェクトの転送にかかる時間よりも短い場合。

データの受け渡し方法の指定

JNBProxy.gui ツールを使用してプロキシを生成すると、参照渡しのプロキシと値渡しのプロキシを指定できます。デフォルトのプロキシタイプは参照です。

クラスのデータ受け渡し方法を指定するには、[Exposed Proxies] ペインでクラスを右クリックします。表示されるリストから受け渡し方法を選択します。受け渡し方法を選択すると、タイプを表すプロキシクラスの色が変更されます。参照は黒、値は青です (パブリックフィールド / プロパティのスタイル)。

複数のプロキシクラスの受け渡し方法を同時に設定します。

1. メニューバーから [Project]-[Pass By Reference / Value] を選択します。
2. [Pass By Reference / Value] ダイアログボックスに、[Exposed Proxies] ペインのすべてのプロキシクラスが表示されます。受け渡す値を設定するクラスを選択します。
3. [Reference] または [Value] ボタンをクリックして (パブリックフィールド / プロパティ)、選択したクラスにタイプを関連付けます。
4. 手順 2 と 3 を繰り返して、複数のクラスのグループを選択して、受け渡し方法を設定します。
5. 「OK」をクリックします。

.NET のバージョンの選択と変更

アプリケーションで .NET オブジェクトを使用しているときにエラーが発生する場合は、バージョンに問題があることがあります。たとえば、System.IO.DriveInfo や System.Net.NetworkInformation.Ping などの、Microsoft システムのクラスの多くは、.NET バージョン 2.0 で追加されたものです。アプリケーションにおけるこれらのクラスの例については、1151 ページの「.NET オブジェクトの使用」および 1153 ページの「例 : .NET クラスを直接使用」を参照してください。

現在の .NET バージョンは、次の関数を使用して取得できます。

```
<cffunction name="GetDotNetVersion" returntype="string">
    <cfset var seClass="">
    <cfobject type=".NET" name="seClass" class="System.Environment">
    <cfreturn seClass.Get_Version().ToString()>
</cffunction>
```

この関数によって、アクティブなバージョンが必要なバージョンでないことが判明した場合は、ColdFusion を実行しているシステムに、正しいバージョンの .NET フレームワーク再配布可能パッケージをインストールまたは再インストールします。その後で、ColdFusion .NET Extension を再インストールして、正しい .NET バージョンが使用されるようにします。

アプリケーションとしての .NET Extension エージェントの実行

.NET サイドの Extension エージェントは、ColdFusion .NET サービスとして自動的に実行されるように、ColdFusion .NET Extension インストーラによって設定されますが、アプリケーションとして実行することもできます。

.NET Extension エージェントをアプリケーションとして実行するには :

1. ColdFusion .NET サービスが実行されている場合は停止します。
2. コマンドプロンプトウィンドウを開いて、"jnbridge" ディレクトリに移動します。スタンドアローン ColdFusion サーバー設定の場合、このディレクトリは "< インストールディレクトリ >¥jnbridge" です。スタンドアローン .NET Extension がインストールされているシステム、または J2EE 構成か複数サーバー構成の場合、このディレクトリは "<.NET のインストールディレクトリ >¥jnbridge" ディレクトリです。デフォルトのインストールディレクトリは "C:¥ColdFusonDotNetExtension" です。
3. 次のコマンドを入力します。

```
JNBDotNetSide
```

CFML アプリケーションでの COM および CORBA オブジェクトの統合

COM (Component Object Model) または DCOM (Distributed Component Object Model) および CORBA (Common Object Request Broker) オブジェクトを呼び出すことができます。

COM と CORBA について

Adobe ColdFusion では、多くのアプリケーションで使用できる COM および CORBA オブジェクトへのアクセスがサポートされています。

オブジェクトについて

COM と CORBA は、ColdFusion でサポートされているオブジェクトテクノロジーの一部です。その他のオブジェクトテクノロジーには、Java および ColdFusion コンポーネントがあります。ColdFusion コンポーネントの詳細については、172 ページの「ColdFusion コンポーネントの構築と使用」を参照してください。

オブジェクトとは、データとその処理が内蔵されているモジュールのことです。いくつかのオブジェクトをビルディングブロックとして組み合わせ、ColdFusion コードに統合することで、1 つのアプリケーションを作成できます。

オブジェクトはハンドル (名前) によって表されます。オブジェクトには、情報を格納するためのプロパティがあります。また、オブジェクトを操作したり、オブジェクトからデータを取得したりするためのメソッドもあります。オブジェクトに関する用語やルールは、オブジェクトテクノロジーによって異なります。

オブジェクトのインスタンスを作成するには、cfobject タグまたは CreateObject 関数を使用します。作成したオブジェクトやそのメソッドは、ColdFusion のタグ、関数、および式で使用できます。オブジェクトを使用するための ColdFusion のシンタックスについては、1159 ページの「オブジェクトの作成と使用」を参照してください。

COM と DCOM について

COM (Component Object Model) は、Microsoft が定めた、コンポーネントの移植性、再利用性、およびバージョン管理機能を実現するサービスの仕様です。DCOM (Distributed Component Object Model) は、分散型サービスに COM を実装したものであり、ネットワーク上のコンポーネントへのアクセスが可能になります。

COM オブジェクトは、ローカルに配置することも、ネットワークノード上に配置することもできます。COM は、Microsoft Windows プラットフォームでサポートされています。

COM の詳細については、Microsoft の Web サイトの COM に関するページ (www.microsoft.com/com) を参照してください。

CORBA について

CORBA (Common Object Request Broker Architecture) は、OMG (Object Management Group) が定めた、オブジェクト指向アプリケーションのための分散コンピューティングモデルです。このモデルでは、オブジェクトはカプセル化されたエンティティであり、そのサービスにアクセスするには規定のインターフェイスを使用する必要があります。各オブジェクトの場所と実装は、サービスをリクエストするクライアントからは隠されています。ColdFusion では、Windows と UNIX 上での CORBA 2.3 の使用がサポートされています。

CORBA では、ORB (Object Request Broker) を使用して、あるシステム上のアプリケーションから、別のシステム上で実行されているオブジェクトにリクエストを送信できます。ORB を介して通信を行うことで、コンピュータプラットフォームや実装言語の違いを意識することなく、分散環境を構築できます。たとえば、あるシステム上の ColdFusion アプリケーションが、別のシステム上の C++ で実装されているオブジェクトと通信することができます。

CORBA は、クライアントサーバーモデルに従っています。クライアントはサーバーで管理されているオブジェクトのオペレーションを呼び出し、サーバーはそのリクエストに応答します。ORB は、IIOP (Internet Inter-ORB Protocol) を使用して、クライアントとサーバーの間の通信を管理します。

CORBA オブジェクトは、IDL (CORBA Interface Definition Language) で定義されたインターフェイスを持ちます。CORBA IDL では、オブジェクトで実行できるオペレーションと、オペレーションのパラメータが記述されています。クライアントは、インターフェイスがどのように実装されているかをまったく知らなくてもリクエストを行うことができます。

クライアントアプリケーションでサーバーのサービスをリクエストするには、オブジェクトへのハンドルを ORB から取得する必要があります。クライアントアプリケーションは、取得したハンドルを使用して、IDL インターフェイス定義で指定されているメソッドを呼び出します。リクエストは ORB からサーバーに渡され、サーバーはそのリクエストを処理して結果をクライアントに返します。

CORBA の詳細については、CORBA に関する情報の主な Web レポジトリである OMG の Web サイト www.omg.com を参照してください。

オブジェクトの作成と使用

オブジェクトの名前付きインスタンスを作成するには、cfobject タグまたは CreateObject 関数を使用します。オブジェクトのプロパティやメソッドを呼び出すには、cfset や cfoutput などの ColdFusion タグを使用します。

オブジェクトの作成と使用に関するテクニックの多くは、COM オブジェクトと CORBA オブジェクトの両方に共通します。この例では、"obj" というサンプルオブジェクトが存在し、このオブジェクトに "Property" というプロパティと、"Method1"、"Method2"、"Method3" というメソッドがあるものと仮定しています。

オブジェクトの作成

ColdFusion でオブジェクトをインスタンス化 (名前の付いたインスタンスを作成) するには、cfobject タグまたは CreateObject 関数を使用します。指定する属性やパラメータは、使用するオブジェクトのタイプによって異なります。詳細については、1162 ページの「COM オブジェクトの作成と使用」および 1171 ページの「CORBA オブジェクトの作成」を参照してください。次の例では、cfobject タグを使用して COM オブジェクトを作成し、CreateObject 関数を使用して CORBA オブジェクトを作成しています。

```
<cfobject type="COM" action="Create" name="obj" class="sample.MyObject">
obj = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "Visibroker")
```

cfobject または CreateObject で作成したオブジェクトや、他のオブジェクトによって返されたオブジェクトは、ColdFusion ページの実行が終了するときに解放されます。

プロパティの使用

プロパティにアクセスするには、次のように、標準の ColdFusion ステートメントを使用します。

1 プロパティを設定するには、ステートメントを使用するか、次のように cfset タグを使用します。

```
<cfset obj.property = "somevalue">
```

2 プロパティを取得するには、ステートメントを使用するか、次のように cfset タグを使用します。

```
<cfset value = obj.property>
```

この例に示すように、プロパティの値を取得するときは、式の右辺で括弧を使用しません。

メソッドの呼び出し

オブジェクトメソッドは、通常 0 個以上の引数を取ります。呼び出し側に値が返されない In 引数は、値渡しです。呼び出し側に値が返される Out または In,Out 引数は、参照渡しです。参照渡しの引数は、通常、オブジェクトによってその値が変更されます。メソッドには、値を返すものと返さないものがあります。

メソッドを呼び出すには、次の方法を使用します。

- メソッドに引数がない場合は、次の cfset タグのように、メソッド名に空の括弧を続けます。

  ```
  <cfset retVal = obj.Method1()>
  ```

- メソッドに 1 つ以上の引数がある場合は、引数をカンマで区切って括弧に入れます。次の例では、整数引数と文字列引数を 1 つずつ指定しています。

  ```
  <cfset x = 23>
  <cfset retVal = obj.Method1(x, "a string literal")>
  ```

- メソッドにリファレンス (Out または In,Out) 引数がある場合は、次の例の変数 x のように、その引数に使用する変数の名前を二重引用符 (") で囲みます。

  ```
  <cfset x = 23>
  <cfset retVal = obj.Method2("x","a string literal")>
  <cfoutput> #x#</cfoutput>
  ```

  この例では、オブジェクトによって x の値が変更された場合は、23 以外の値が表示されます。

ネストされたオブジェクトの呼び出し

ColdFusion では、ネストされた (スコープが指定されている) オブジェクトの呼び出しがサポートされています。たとえば、オブジェクトメソッドの戻り値であるオブジェクトのプロパティやメソッドを呼び出す場合は、次の両方のシンタックスを使用できます。

```
<cfset prop = myObj.X.Property>
```

または

```
<cfset objX = myObj.X>
<cfset prop = objX.Property>
```

COM と DCOM の概要

ColdFusion は、オートメーション (実行時バインディング) COM クライアントです。したがって、COM オブジェクトでは、IDispatch インターフェイスがサポートされている必要があります。また、メソッドの引数とプロパティは、標準のオートメーション型である必要があります。ColdFusion では型の宣言が行われないので、呼び出し時の引数のセットアップには、オブジェクトの型情報が使用されます。オブジェクトのデータ型があいまいな場合は、予期しない動作が発生することがあります。

ColdFusion では、グラフィカルユーザーインターフェイスを持たない、サーバーサイド COM オブジェクトのみを使用してください。グラフィカルユーザーインターフェイスを持つオブジェクトを ColdFusion アプリケーションで呼び出すと、そのコンポーネントは、クライアントのデスクトップではなく、Web サーバーのデスクトップに表示されます。この動作によって、ColdFusion サーバースレッドが占有され、Web サーバーでリクエストが処理できなくなる場合があります。

ColdFusion では、Inproc COM オブジェクト、ローカル COM オブジェクト、またはリモート COM オブジェクトを呼び出せます。どのタイプのオブジェクトを呼び出すかは、cfobject タグの属性で指定します。

COM の必要条件

ColdFusion アプリケーションで COM コンポーネントを利用するには、少なくとも次のものが必要です。

- ColdFusion アプリケーションページで使用する COM オブジェクト (通常、DLL ファイルまたは EXE ファイル)。これらのコンポーネントでは IDispatch インターフェイスが実装されるため、実行時バインディングを許可する必要があります。
- Microsoft OLE/COM オブジェクトビューア。Microsoft から入手できます。このツールを使用すると、登録された COM オブジェクトを表示できます。

  オブジェクトビューアを使用すると、オブジェクトのクラス情報を表示できるので、cfobject タグの class 属性を指定するときに役立ちます。また、オブジェクトでサポートされているインターフェイスを表示できるので、オブジェクトの (IDispatch インターフェイスの) プロパティおよびメソッドを確認できます。

オブジェクトの登録

オブジェクトを取得したら、ColdFusion (や他のプログラム) がそのオブジェクトを検出できるように、Windows に登録します。オブジェクトには、自動的に登録を行うセットアッププログラムを持つものもあれば、手動での登録が必要なものもあります。

Inproc オブジェクトサーバー (.dll または .ocx ファイル) を手動で登録するには、次のように、"regsvr32.exe" ユーティリティを使用します。

```
regsvr32 c:\path\servername.dll
```

ローカルサーバー (.exe ファイル) を登録するには、それらのサーバーを起動するか、次のようなコマンドラインパラメータを指定します。

```
C:\pathname\servername.exe -register
```

コンポーネントの ProgID とメソッドの確認

多くの場合、コンポーネントのメソッドや ProgID に関する資料は、COM オブジェクトのサプライヤから入手できます。マニュアルがない場合は、ColdFusion の cfdump タグか OLE/COM オブジェクトビューアを使用して、コンポーネントのインターフェイスを確認できます。

cfdump タグによる COM オブジェクトインターフェイスの表示

ColdFusion 上では、ColdFusion cfdump タグを使用して、COM オブジェクトに関する次の情報を表示できます。

- パブリックメソッド
- put プロパティ
- get プロパティ

メソッドやプロパティの情報では、パラメータやプロパティの型と、その値が in、out、optional、retval のいずれであるかが示されます。cfdump タグの出力には、オブジェクトの ProgID は含まれません。

注意：ダンプヘッダーには、ColdFusion オブジェクトクラス (coldfusion.runtime.com.ComProxy) と、COM オブジェクトの CLSID が示されます。

OLE/COM オブジェクトビューアの使用

OLE/COM オブジェクトビューアをインストールすると、デフォルトで "¥mstools¥bin¥oleview.exe" という実行可能ファイルがインストールされます。オブジェクトビューアを使用すると、COM オブジェクトの ProgID や、メソッドおよびプロパティを確認できます。

オブジェクトビューアにオブジェクトを表示するには、そのオブジェクトを登録しておく必要があります。詳細については、1161 ページの「オブジェクトの登録」を参照してください。オブジェクトビューアでは、レジストリに登録されている COM オブジェクトおよびコントロールがすべて取得され、グループ別にソートされたシンプルな形式で見やすく表示されます。

カテゴリを選択してコンポーネントを選択すれば、COM オブジェクトの ProgID を確認できます。また、オブジェクトのオペレーションのオプションも確認できます。

オブジェクトのプロパティを表示するには :

1. オブジェクトビューアを開き、ウィンドウをスクロールして目的のオブジェクトを探します。
2. オブジェクトビューアの左ペインで、オブジェクトを選択して展開します。
3. オブジェクトを右クリックして表示します。TypeInfo も表示できます。

   TypeInfo を表示すると、オブジェクトのメソッドとプロパティを確認できます。TypeInfo 領域にアクセスできないオブジェクトもあります。TypeInfo 領域にアクセスできるかどうかは、オブジェクト作成時の設定と使用された言語によって決まります。

COM オブジェクトの作成と使用

アプリケーションページで COM オブジェクト (コンポーネント) のメソッドを呼び出したりプロパティを設定したりするには、その前に、ColdFusion で cfobject タグまたは CreateObject 関数を使用して、コンポーネントのインスタンスを作成します。

たとえば次のコードでは、電子メールを送信するために使用する Windows CDO (Collaborative Data Object) for NTS の NewMail オブジェクトを、cfobject タグで作成しています。

```
<cfobject type="COM"
        action="Create"
        name="Mailer"
        class="CDONTS.NewMail">
```

CFScript で同じ処理を行うには、次のコードのように CreateObject 関数を使用します。

```
Mailer = CreateObject("COM", "CDONTS.NewMail");
```

本マニュアルのいくつかの例では、このオブジェクトを使用しています。

注意：CDO は、Microsoft SMTP サーバーがインストールされているすべての Windows NT および 2000 オペレーティングシステムにデフォルトでインストールされています。Windows NT Server 環境では、Option Pack 4 セットアップに SMTP サーバーが含まれています。Windows 2000 Server および Workstation 環境では、オペレーティングシステムにバンドルされています。

メモ : CDO for NTS の NewMail コンポーネントには、さまざまなメール処理タスクを行うためのメソッドやプロパティが用意されています。OLE/COM オブジェクトビューアでは、メソッドとプロパティが同じレベルで列挙されていることがあり、最初は判別が困難な場合があります。

CDO for NTS の NewMail オブジェクトには、次のプロパティがあります。

```
Body [ String ]
Cc[ String ]
From[ String ]
Importance[ Long ]
Subject[ String ]
To[ String ]
```

これらのプロパティを使用して、メールメッセージの各要素を定義できます。CDO for NTS の NewMail オブジェクトには、メッセージを送信するためのオプション引数を持つ send メソッドもあります。

COM オブジェクトへの接続

cfobject タグの action 属性では、COM オブジェクトへの接続方法として、次のいずれかを指定できます。

Create メソッド (cfobject action="Create") COM オブジェクト (通常は DLL) をインスタンス化して、メソッドの実行やプロパティの設定を行えるようにします。

Connect メソッド (cfobject action="Connect") サーバー上で既に実行されているオブジェクト (通常は実行可能ファイル) にリンクします。

オプションの cfobjectcontext 属性を使用して、オブジェクトのコンテキストを指定することもできます。コンテキストを指定しない場合は、レジストリの設定が使用されます。次の表で、context 属性値について説明します。

属性値	説明
InProc	ColdFusion などの、呼び出しプロセスと同じプロセス領域で実行されている in-Process サーバーオブジェクト (通常は DLL)。
local	ColdFusion プロセス領域の外で実行されているが、同じサーバー上でローカルに実行されている out-of-process サーバーオブジェクト (通常は EXE ファイル)。
remote	ネットワーク上でリモートに実行されている out-of-process サーバープロジェクト (通常は EXE ファイル)。remote を指定する場合は、オブジェクトの場所を識別する server 属性も使用します。

プロパティの設定とメソッドの実行

次の例では、サンプルの Mailer COM オブジェクトを使用して、メールメッセージにプロパティを割り当てる方法を示します。さらに、コンポーネントのメソッドを実行してメールメッセージを処理する方法を示します。

この例では、メソッドパラメータやプロパティ (受信者の名前や電子メールアドレスなど) がフォーム変数に保存されているものとしています。

```
<!--- First, create the object --->
<cfobject type="COM"
    action="Create"
    name="Mailer"
    class="CDONTS.NewMail">

<!--- Second, use the form variables from the user entry form to populate a number
        of properties necessary to create and send the message. --->
<cfset Mailer.From = "#Form.fromName#">
<cfset Mailer.To = "#Form.to#">
<cfset Mailer.Subject = "#Form.subject#">
<cfset Mailer.Importance = 2>
<cfset Mailer.Body = "#Form.body#">
<cfset Mailer.Cc = "#Form.cc#">

<!--- Last, use the Send() method to send the message.
    Invoking the Send() method destroys the object.--->
<cfset Mailer.Send()>
```

注意：COM オブジェクトで発生する例外を処理するには、cftry および cfcatch タグを使用します。例外処理の詳細については、287 ページの「ColdFusion タグでのランタイム例外の処理」を参照してください。

COM オブジェクトの解放

デフォルトでは、Java ガベージコレクタが COM オブジェクトをクリーニングすると、COM オブジェクトのリソースが解放されます。オブジェクトが不要になった場合は、ReleaseCOMObject 関数を使用してすぐにリソースを解放できます。

ReleaseCOMObject 関数を使用すると、外部プロセスとして起動された Microsoft Excel などの COM オブジェクトを解放できます。ガベージコレクタでは、プロセスのクリーニングが実行されるまでに時間がかかることがあります。その結果、複数の外部プロセスが存在して、システムリソースが浪費されることがあります。

COM オブジェクトに、プログラムを終了するメソッドがある場合は、ReleaseComObject 関数を呼び出す前にそのメソッドを呼び出します。使用中のオブジェクトに対して ReleaseComObject 関数を使用すると、正常に完了する前にオブジェクトが解放されて、アプリケーションで例外が発生します。

例

次の例では、Microsoft Excel アプリケーションオブジェクトを作成して使用した後に、不要になったオブジェクトを解放します。

```
<h3>ReleaseComObject Example</h3>
<cfscript>
obj = CreateObject("Com", "excel.application.9");
//code that uses the object goes here
obj.quit();
ReleaseComObject(obj);
</cfscript>
```

COM オブジェクトに関する一般的な注意事項

COM オブジェクトを使用する場合は、エラーを回避および解決するために、次の事項を考慮する必要があります。

- スレッドへの対応
- in 引数および out 引数の使用
- COM 関連の一般的なエラーメッセージの理解

スレッドへの対応

ColdFusion で COM オブジェクトを使用する場合、スレッドへの対応が不適切であると、重大な問題が発生することがあります。オブジェクトはスレッドセーフであることが必要です。スレッドセーフとは、複数のプログラミングスレッドから同時に呼び出されても、オブジェクトの処理が正常に実行されることをいいます。

Visual Basic の ActiveX DLL は、通常スレッドセーフではありません。このような DLL を ColdFusion で使用する場合は、OLE/COM オブジェクトビューアを使用して、オブジェクトのスレッド処理モデルを Apartment モデルに変更することで、スレッドセーフにすることができます。

COM オブジェクトへの参照を Application、Session、または Server スコープに保管する場合は、Apartment スレッド処理モデルを使用しないでください。Apartment スレッド処理モデルは、単一のリクエストのみを処理するモデルです。これらのスコープにオブジェクトを保管する必要がある場合は、オブジェクトのスレッド処理モデルを Both のままにし、316 ページの「cflock によるコードのロック」の説明に従って、そのオブジェクトにアクセスするコードをすべてロックします。

COM オブジェクトのスレッド処理モデルの変更

1. OLE/COM オブジェクトビューアを起動します。
2. 左ペインの [Object Classes] の下にある [All Objects] を選択します。
3. 対象の COM オブジェクトを探します。左ペインにオブジェクトの名前がリストされます。
4. 対象のオブジェクトを選択します。
5. 右ペインの [Implementation] タブを選択します。
6. [App ID] フィールドの下にある [Inproc Server] タブを選択します。
7. [Threading Model] ドロップダウンリストから、[Apartment] または [Both] のいずれか適切なほうを選択します。

in 引数および out 引数の使用

COM オブジェクトメソッドの in 引数は、値渡しです。COM オブジェクトには変数値のコピーが渡されるので、ColdFusion 変数は引用符で囲まずに指定できます。

COM オブジェクトメソッドの out 引数は、参照渡しです。呼び出しページの変数が COM オブジェクトによって変更されるので、呼び出しページでその値を使用することができます。参照によって変数を渡すには、既存の ColdFusion 変数を引用符で囲みます。引数が数値型である場合は、呼び出しを行う前に、その変数に有効な数値を代入します。次に例を示します。

```
 <cfset inStringArg="Hello Object">
<cfset outNumericArg=0>
<cfset result=myCOMObject.calculate(inStringArg, "outNumericArg")>
```

オブジェクトの calculate メソッドに文字列 "Hello Object" が in 引数として渡され、outNumericArg の値が数値に設定されます。

COM 関連の一般的なエラーメッセージの理解

次の表で、COM オブジェクトの使用時に発生する可能性があるエラーメッセージについて説明します。

エラー	原因
Error Diagnostic Information Error trying to create object specified in the tag. COM error 0x800401F3.Invalid class string.	COM オブジェクトが登録されていないか、または存在しません。
Error Diagnostic Information Error trying to create object specified in the tag. COM error 0x80040154.Class not registered.	COM オブジェクトが登録されていないか、または存在しません。このエラーは通常、以前は存在していたオブジェクトが削除された場合に発生します。
Error Diagnostic Information Failed attempting to find "SOMEMETHOD" property/method on the object COM error 0x80020006. Unknown name.	COM オブジェクトは正常にインスタンス化されましたが、指定されたメソッドが存在しません。

Java プロキシによる複合 COM オブジェクトへのアクセス

ColdFusion では、COM オブジェクトにアクセスするための Java プロキシがサポートされています。Java プロキシが事前に作成されていない場合は、ColdFusion がダイナミックに COM インターフェイスを検出する必要があります。これには、次のような 2 つの短所があります。

- ダイナミックな検出には時間がかかり、複雑な COM オブジェクトを頻繁に使用する場合はサーバーのパフォーマンスが低下することがあります。
- ダイナミックな検出では IDispatcher インターフェイスを使用して COM オブジェクトの機能が認識されるので、複雑な COM インターフェイスは処理されないことがあります。

ColdFusion には、この問題を解決するために、COM オブジェクト用のスタティックな Java スタブプロキシクラスを作成する "com2java.exe" というユーティリティが用意されています。ColdFusion は、この Java スタブを使用することで、ダイナミックにプロキシを作成するよりも効率的に COM オブジェクトにアクセスすることができます。また "com2java.exe" ユーティリティでは、ダイナミックプロキシジェネレータでは認識されないような機能のスタブも作成できます。

ColdFusion には、Windows XP、Windows 2000、および Windows 97 版の Microsoft Excel、Microsoft Word、および Microsoft Access に対応する、生成済みのスタブが用意されています。これらのスタブは自動的に使用されるように設定されています。

COM オブジェクトの Java スタブファイルを作成する場合は、COM オブジェクトを作成する場合と同様に、cfobject タグの type 属性に値 COM を指定するか、CreateObject 関数の最初の引数に COM を指定し、ColdFusion で COM オブジェクトにアクセスする場合と同様にオブジェクトのプロパティやメソッドにアクセスします。

"com2java.exe" ユーティリティを使用するには、次の手順に従います。この手順では、Microsoft Outlook を例として使用しています。

COM オブジェクトの Java スタブファイルを作成するには :

1. システムを次のように構成します。
   a. JDK (Java Development Kit) が正しくインストールされていることを確認します。たとえば、CLASSPATH やコマンドプロンプト PATH 変数が正しく構成されていることを確認します。
   b. CLASSPATH に "<ColdFusion のルートディレクトリ >¥lib¥jintegra.jar" を追加します。
2. 次の例のように、Java スタブファイルのための新しいディレクトリを作成します。

```
mkdir C:\src\outlookXP
```

このディレクトリは一時的なものでかまいません。このディレクトリのファイルは、ColdFusion JAR ファイルに追加されます。

3 コマンドラインまたは Windows の [スタート] メニューから、"<ColdFusion のルートディレクトリ >¥Jintegra¥bin¥com2java.exe" プログラムを実行します。ウィンドウが表示されます。

   a COM クラスに、同じ名前のメソッドを持つ複数のインターフェイスが実装されている場合は、[Options] メニューアイテムを選択して、[Implement interfaces that may conflict] オプションをオフにします。生成された Java スタブクラスが、競合する複数のインターフェイスを実装することはありません。生成された getAs**XXX** メソッドを使用すれば、インターフェイスにアクセスすることができます。Java ファイルに生成されたコメントを確認します。

   b [SELECT] ボタンをクリックします。

   c COM オブジェクトのタイプライブラリまたは DLL を選択します。このファイルは、Windows XP の Microsoft Outlook の場合、通常は "Program Files¥Microsoft Office¥Office10¥MSOUTL.OLB" にあります。

   d com2java ダイアログボックスの [Java package] フィールドに、パッケージ名 (**outlookXP** など) を入力します。このパッケージには、COM オブジェクトの Java スタブに対するすべてのクラスが含まれます。

   注意：Microsoft Excel、Microsoft Word、および Microsoft Access 用の生成済みの Java スタブが含まれているパッケージは、名前が coldfusion.runtime.com.com2java で始まります。たとえば、Microsoft Word XP 用の Java スタブクラスが含まれているパッケージの名前は coldfusion.runtime.com.com2java.wordXP になります。したがって、**"msapps.jar"** ファイルにおける **wordXP** クラスのパスは、このパッケージ名の階層に従って、**coldfusion¥runtime¥com¥com2java¥wordXP¥className.class** になります。このネーミング規則は必須ではありませんが、多くの **COM** オブジェクトを使用する場合は、わかりやすくするために同様のパッケージネーミング規則を使用することを検討してください。

   e [Generate Proxies] ボタンをクリックして、ファイルブラウザを表示します。手順 2 で作成したディレクトリを選択し、ファイルブラウザの [OK] ボタンをクリックしてスタブファイルを生成します。

   f [Close] をクリックして "com2java.exe" ユーティリティを閉じます。

   ディレクトリに生成されるファイルには、次のものがあります。

   - 各 COM インターフェイスの Java インターフェイスとプロキシクラス
   - 各 COM クラスの Java クラス
   - 各 ENUM (一連の定数定義) の Java インターフェイス

4 Java コードをコンパイルします。コマンドプロンプトで、次の作業を行います。

   a Java スタブが含まれているディレクトリ (この例では C:¥src¥outlookXP) を作業ディレクトリにします。

   b 次の行を入力します。

```
javac -J-mx100m -J-ms100m *.java
```

   コンパイラスイッチで、必要なファイルをすべてコンパイルするために十分なメモリがあることを確認します。

   注意：手順 1b で CLASSPATH に "jintegra.jar" を含めなかった場合は、スイッチ -classpath:/<ColdFusion のルートディレクトリ >/lib/jintegra.jar をコマンドに追加します。<ColdFusion のルートディレクトリ > は、ColdFusion がインストールされているディレクトリです。

5 ColdFusion サーバーが停止していることを確認します。ColdFusion サーバーを停止するには、[サービス] コントロールパネルを開き、ColdFusion アプリケーションサーバーを選択して、[停止] をクリックします。

6 次の操作を行って、.class ファイルを ColdFusion Microsoft アプリケーションの Java スタブファイルに追加します。

   a Windows のコマンドプロンプトで、クラスファイルが含まれているディレクトリの親ディレクトリを、作業ディレクトリにします。次の例では、手順 4 のコマンドプロンプトで **cd ..** と入力して、"c:¥src" を作業ディレクトリにします。

   b 次のコマンドを入力します。

```
jar -uvf cf_root\lib\msapps.jar directoryName\*.class
```

<ColdFusion のルートディレクトリ > は、ColdFusion がインストールされているディレクトリです。< ディレクトリ名 > は、クラスファイルが含まれているディレクトリの名前です。OutlookXP の例では、次の行を入力します。

```
jar -uvf C:\CFusion\lib\msapps.jar outlookXP\*.class
```

7 "<ColdFusion のルートディレクトリ >/lib/neo-comobjmap.xml" ファイルのリストにオブジェクト定義を追加します。オブジェクト定義は、次の行で構成されます。

```
<var name="progID">
<string>PackageName.mainClass</string>
</var>
```

これらの行では、次の値を使用します。

ProgID OLE/COM オブジェクトビューアに表示される、COM オブジェクトの ProgID。

PackageName 手順 3c で指定したパッケージ名。

mainClass COM オブジェクトのメインクラス。メインクラスには、呼び出すメソッドが含まれています。多くの Microsoft アプリケーションでは、メインクラスは Application です。一般に、手順 4 で作成される最大のクラスファイルがメインクラスです。

たとえば、outlookXP を "neo-comobjmap.xml" に追加するには、次の太字で示されている行を </struct> 終了タグの上に追加します。

```
<var name="access.application.9">
<string>coldfusion.runtime.com.com2java.access2k.Application</string>
</var>
<var name="outlook.application.10">
<string>outlookXP.Application</string>
</var>
</struct>
```

この例では、**outlook.application.10** が Outlook COM オブジェクトの ProgID、**outlookXP** が手順 3c で指定したパッケージ名、**Application** が COM オブジェクトのメインクラスです。

8 ColdFusion サーバーを再起動します。[サービス] コントロールパネルを開き、ColdFusion アプリケーションサーバーを選択して [開始] ボタンをクリックします。

9 スタブのインストールが終わったら、手順 2 で作成したディレクトリとその内容をすべて削除できます。

Application スコープの使用による COM パフォーマンスの向上

COM オブジェクトインスタンスを作成する Java 呼び出しには、かなりの時間がかかることがあります。そのため、ColdFusion で COM オブジェクトを作成すると、ColdFusion 5 を使用したときよりも大幅に時間がかかる場合があります。たとえば、システムによっては、ColdFusion を使用して Microsoft Word アプリケーションオブジェクトを作成すると 1 秒以上かかり、一方、同じシステムで Word オブジェクトを作成したときのオーバーヘッドは約 200 ミリ秒になることがあります。

したがって、Application スコープの単一の COM オブジェクトをすべてのページで共有すれば、COM のパフォーマンスを向上させることができます。

このテクニックは、次の条件がすべて満たされる場合にのみ使用します。

- リクエストまたはセッションごとに COM オブジェクトを作成する必要がない。セッション固有のオブジェクトの場合は、Application スコープの代わりに、ここで説明するテクニックを Session スコープで使用することを検討してください。
- COM オブジェクトが共有に対応している。

オブジェクトは複数のページやセッションから同時にアクセスされることがあるので、スレッド処理やロックに関する次のような問題を考慮してください。

- 最高のパフォーマンスを得るには、オブジェクトをマルチスレッド対応にします。マルチスレッドに対応していない場合、オブジェクトに同時にアクセスできるリクエストは 1 つのみです。
- 共有データにアクセスするコードや、共有データを変更するコードをロックします。共有オブジェクトのデータを変更するコードのうち、複数のリクエストで共有されていないデータ (書き込み可能なプロパティやファイルの内容などを含む) を変更するコードは、一般にロックしなくてかまいません。ただし、実際にロックが必要かどうかは、COM オブジェクトのセマンティクス、インターフェイス、および実装によって異なります。
- Application スコープのロックを使用している、アプリケーション内のすべての cflock タグは、1 つのロックを共有します。したがって、Application スコープのロックを使用して頻繁にアクセスされる COM オブジェクトがある場合は、そのオブジェクトにアクセスするコードがボトルネックになり、そのオブジェクトを使用するページが多数のユーザーによってリクエストされるとスループットが低下することがあります。COM オブジェクトを使用するコードを名前付きロックの中に入れることで、一部の競合を回避できる場合もあります。オブジェクトを作成するコードは、Application スコープロックの中に入れる必要があります。

注意：1166 ページの「Java プロキシによる複合 COM オブジェクトへのアクセス」の説明のように、Java スタブを作成することで一部の COM オブジェクトのパフォーマンスを向上させることもできます。Java スタブを使用する方法は、COM オブジェクトを共有する方法ほどパフォーマンスは向上しませんが、すべての COM オブジェクトで使用できます。また、COM IDispatcher インターフェイスからは適切に利用できない機能がある複雑な COM オブジェクトに正しくアクセスするには、Java スタブを生成します。したがって、この 2 つの方法を組み合わせれば、パフォーマンスを最大限に向上させ、問題を未然に防ぐことができます。

例 1 : FileSystem オブジェクトの使用

次の例では、Application スコープで Microsoft FileSystem スクリプトオブジェクトを使用します。このコードでは、システム上のすべてのハードドライブの名前と空き容量が列挙された構造体を返すユーザー定義関数を作成します。

```
 <cfapplication name="comtest" clientmanagement="No" Sessionmanagement="yes">

<!--- Uncomment the following line if you must delete the object from the
Application scope during debugging. Then restore the comments.
This technique is faster than stopping and starting the ColdFusion server. --->
    <!--- <cfset structdelete(Application, "fso")> --->

<!--- The getFixedDriveSpace user-defined function returns a structure with
the drive letters as keys and the drive's free space as data for all fixed
drives on a system. The function does not take any arguments --->

<cffunction name="getFixedDriveSpace" returnType="struct" output=True>
    <!--- If the FileSystemObject does not exist in the Application scope,
    create it. --->
    <!--- For information on the use of initialization variables and locking in
        this code, see "Locking application variables efficiently" in Chapter 15,
        "Using Persistent Data and Locking" --->
    <cfset fso_is_initialized = False>
    <cflock scope="application" type="readonly" timeout="120">
        <cfset fso_is_initialized = StructKeyExists(Application, "fso")>
    </cflock>
    <cfif not fso_is_initialized >
        <cflock scope="Application" type="EXCLUSIVE" timeout="120">
            <cfif NOT StructKeyExists(Application, "fso")>
                <cfobject type="COM" action="create" class="Scripting.FileSystemObject"
                    name="Application.fso" server="\\localhost">
            </cfif>
        </cflock>
    </cfif>
```

```
    <!--- Get the drives collection and loop through it to populate the
        structure. --->
    <cfset drives=Application.fso.drives()>
    <cfset driveSpace=StructNew()>
    <cfloop collection="#drives#" item="curDrive">
        <!--- A DriveType of 2 indicates a fixed disk --->
        <cfif curDrive.DriveType IS 2>
        <!--- Use dynamic array notation with the drive letter for the struct key
        --->
            <cfset driveSpace["#curDrive.DriveLetter#"]=curDrive.availablespace>
        </cfif>
    </cfloop>
    <cfreturn driveSpace>
</cffunction>

<!--- Test the function. Get the execution time for running the function --->
<cfset start = getTickCount()>
<cfset DriveInfo=getFixedDriveSpace()>
<h3>Getting fixed drive available space</h3>
<cfoutput>Execution Time: #int(getTickCount()-start)# milliseconds</cfoutput><br><br>
<cfdump label="Drive Free Space" var="#driveInfo#">
```

例 2 : Microsoft Word アプリケーションオブジェクトの使用

次の例では、Application スコープで Microsoft Word アプリケーションの COM オブジェクトを使用して、Word ドキュメントを HTML に変換します。コメントに記述されているように、この例は Word 2000 用です。Word 97 で使用するには、"Val(8)" を "Val(10)" に変更します。

この例では、Application スコープロックを使用して、他のページによってオブジェクトの作成が中断されないようにしています。Word オブジェクトが存在する場合は、名前付きのロックを使用して、変換するファイルが同時にアクセスされないようにしています。

```
 <cfapplication name="comtest" clientmanagement="No" Sessionmanagement="yes">
<!--- Uncomment the following line if you must delete the object from the
Application scope --->
<!--- <cfset structdelete(Application, "MyWordObj")> --->

<!--- use the GetTickCount function to get a current time indicator, used for
        displaying the total processing time. --->
<cfset start = GetTickCount()>
<!--- If necessary, create the Word.application object and place it in the
        Application scope --->
<cfset WordObj_is_initialized = False>
<cflock scope="application" type="readonly" timeout=120>
    <cfset WordObj_is_initialized = StructKeyExists(application, "MyWordObj")>
</cflock>
<cfif not WordObj_is_initialized >
    <cflock scope="Application" type="exclusive" timeout="120">
        <cfif not StructKeyExists(application, "MyWordObj")>

<!--- First try to connect to an existing Word object --->
            <cftry>
                <cfobject type="com"
                    action="connect"
                    class="Word.application"
                    name="Application.MyWordobj"
                    context="local">
                <cfcatch>
<!--- No object exists, create one --->
                    <cfobject type="com"
                        action="Create"
                        class="Word.application"
```

```
                        name="Application.MyWordobj"
                        context="local">
                </cfcatch>
            </cftry>

            <cfset Application.mywordobj.visible = False>
        </cfif>
    </cflock>
</cfif>

<!--- Convert a Word document in temp.doc to an HTML file in temp.htm. --->
<!--- Because this example uses a fixed filename, multiple pages could try
    to use the file simultaneously. The lock ensures that all actions from
    reading the input file through closing the output file are a single "atomic"
    operation, and the next page cannot access the file until the current page
    completes all processing.
    Use a named lock instead of the Application scope lock to reduce lock contention. --->
<cflock name="WordObjLock" type="exclusive" timeout="120">
    <cfset docs = application.mywordobj.documents()>
    <cfset docs.open("c:\CFusion\wwwroot\temp.doc")>
    <cfset converteddoc = application.mywordobj.activedocument>
    <!--- Val(8) works with Word 2000. Use Val(10) for Word 97 --->
    <cfset converteddoc.saveas("c:\CFusion\wwwroot\temp.htm",val(8))>
    <cfset converteddoc.close()>
</cflock>

<cfoutput>
    Conversion of temp.htm Complete<br>
    Execution Time: #int(getTickCount()-start)# milliseconds<br>
</cfoutput>
```

CORBA の概要

ColdFusion の cfobject タグと CreateObject 関数は、DII (Dynamic Invocation Interface) を介して CORBA をサポートします。COM の場合と同様に、オブジェクトのタイプ情報が ColdFusion で利用できる必要があります。したがって、IIOP に準拠したインターフェイスレポジトリ (IR) がネットワーク上で実行され、その IR にオブジェクトの IDL (Interface Definition Language) 仕様が登録されている必要があります。CORBA オブジェクトへの参照をネーミングサービスから取得する場合は、ネーミングサービスもネットワーク上で実行されていることが必要です。

ORB ランタイムライブラリは、ColdFusion の起動時にコネクタを使用してロードされるので、利用できる ORB は特定のベンダーの製品に限定されません。現在、ColdFusion には Borland VisiBroker 4.5 ORB のコネクタが用意されています。他の ORB のコネクタを作成するためのソースは NDA の下で提供されており、各サードパーティや ORB ベンダから選択できます。

ColdFusion で CORBA へのアクセスを設定して有効にするには、いくつかの手順を行う必要があります。詳細については、『ColdFusion インストール』を参照してください。

注意：ColdFusion で CORBA アクセスを有効にするには、IDL ファイルを使用してインターフェイスレポジトリを起動する必要があります。このファイルには、サーバー上の ColdFusion アプリケーションで起動するすべての CORBA オブジェクトの IDL が含まれている必要があります。

CORBA オブジェクトの作成

ColdFusion では、cfobject タグと CreateObject 関数は、リモートサーバーの CORBA オブジェクトのためのスタブ (プロキシオブジェクト) を作成します。このスタブオブジェクトは、リモートオブジェクトの呼び出しに使用します。

次の表で、CORBA オブジェクトを作成するときに cfobject タグで使用する属性について説明します。

属性	説明
type	CORBA であることが必要です。デフォルトは COM です。
context	次のように、CORBA のバインディング方法 (オブジェクトの取得方法) を指定します。 • IOR : オブジェクトに固有の IOR (Interoperable Object Reference) を持つファイルを使用します。 • NameService : ネーミングサービスを使用します。
class	設定したバインディング方法でオブジェクトにアクセスするために必要な情報を指定します。 context 属性に IOR を設定した場合は、IOR の文字列バージョンが含まれているファイルの絶対パスを class 属性で指定する必要があります。ColdFusion では、この IOR ファイルが常に読み取り可能であることが必要です。したがって、このファイルは、サーバーに対してローカルであるか、あるいはネットワーク上のアクセス可能な場所に置く必要があります。 context 属性に NameService を設定した場合は、MyCompany/Department/Dev のようにスラッシュ (/) で区切られた名前を class 属性に指定する必要があります。"adobe.current/Eng.current/CF" のように、class 属性で " 種類 " 識別子をピリオドで区切って指定できます。
name	アプリケーションがオブジェクトのインターフェイスを呼び出すときに使用する名前 (ハンドル) を指定します。
locale	(オプション) コネクタの設定を識別します。ColdFusion Administrator にあるコネクタ設定が 1 つの場合や、コネクタ設定は複数あるが、現在 Administrator で選択されているコネクタ設定を使用する場合は、このオプションを省略できます。この属性には、ColdFusion Administrator で CORBA コネクタを設定するときに [CORBA コネクタ] の [ORB 名] フィールドに指定した、VisiBroker などの ORB 名を指定する必要があります。

たとえば、ORB 名を VisiBroker と設定した場合、"tester.ior" ファイルで指定した CORBA オブジェクトを呼び出すには、次の CFML を使用します。

```
<cfobject action = "create" type = "CORBA" context = "IOR"
    class = "d:\temp\tester.ior" name = "handle" locale = "Visibroker">
```

CreateObject 関数を使用してこの CORBA オブジェクトを呼び出すには、関数の戻り変数として名前を指定し、タイプ、クラス、コンテキスト、およびロケールを引数として指定します。たとえば、次のコードは、前述の cfobject タグによって作成したものと同じオブジェクトを作成します。

```
handle = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "Visibroker")
```

ネーミングサービスの使用

現在 ColdFusion が解決できるのは、CORBA 2.3 準拠のネーミングサービスに登録されているオブジェクトのみです。

ネーミングサービスを使用する場合は、そのネーミングコンテキストが、使用中のコネクタ設定のプロパティファイル (ColdFusion Administrator の [CORBA コネクタ] ページで指定) で指定されているネーミングコンテキストと同一であることが必要です。プロパティファイルには、"SVCnameroot=**name**" という行が必要です。**name** は使用するネーミングコンテキストです。オブジェクトを実装するサーバーをこのコンテキストにバインドし、適切な名前を登録する必要があります。

ColdFusion での CORBA オブジェクトの使用

オブジェクトを作成したら、1159 ページの「オブジェクトの作成と使用」で説明したシンタックスを使用して、オブジェクトの属性やオペレーションを呼び出すことができます。

ColdFusion での CORBA インターフェイスメソッドの使用

cfobject タグまたは CreateObject 関数を使用して CORBA オブジェクトを作成すると、CORBA インターフェイスへのハンドルが作成されます。このハンドルは、cfobject タグの name 属性または CreateObject 関数の戻り変数です。たとえば、次の CFML では myHandle という名前のハンドルが作成されます。

```
<cfobject action = "create" type = "CORBA" context = "IOR"
    class = "d:\temp\tester.ior" name = "myHandle" locale="visibroker">
<cfset myHandle = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "visibroker")
```

次の CFML のように、すべてのインターフェイスメソッドは、このハンドル名を使用して呼び出します。

```
<cfset ret=myHandle.method(foo)>
```

メソッド名の大文字と小文字の区別

IDL のメソッド名では、大文字と小文字が区別されます。ただし、ColdFusion では大文字と小文字が区別されません。IDL で大文字と小文字のみが異なるメソッド名は使用できません。

たとえば、次の IDL メソッド宣言は、異なる 2 つのメソッドを表しています。

```
testCall(in string a); // method #1
TestCall(in string a); // method #2
```

しかし、ColdFusion ではこの 2 つのメソッドを区別できません。したがって、いずれかのメソッドを呼び出したときに、どちらのメソッドが呼び出されるかはわかりません。

パラメータの値渡し (in パラメータ)

CORBA の in パラメータは、常に値渡しです。ColdFusion で変数を使用して CORBA メソッドを呼び出すときは、次の例のように、変数名を引用符なしで指定します。

IDL	void method(in string a);
CFML	<cfset foo="my string"> <cfset ret=handle.method(foo)>

変数の参照渡し (out および inout パラメータ)

CORBA の out および inout パラメータは、常に参照渡しです。したがって、メソッドに渡した変数が CORBA オブジェクトによって変更された場合は、変更後の値が ColdFusion ページのその変数に設定されます。

ColdFusion でパラメータの参照渡しを行うには、CORBA メソッドで変数名を二重引用符 (") で囲んで指定します。次の例の IDL 行では、参照渡しの inout パラメータである文字列変数 b を持つメソッドが定義されています。また、このメソッドを呼び出す CFML も示されています。

IDL	void method(in string a, inout string b);
CFML	<cfset foo = "My Initial String"> <cfset ret=handle.method(bar, "foo")> <cfoutput>#foo#</cfoutput>

この場合、ColdFusion 変数 foo は inout パラメータ b に対応します。この CFML を実行すると、次の処理が行われます。

1. ColdFusion によってメソッドが呼び出され、変数が参照渡しされます。
2. CORBA メソッドによって、渡された値 "My Initial String" が別の値に置き換えられます。変数は参照によって渡されているので、この処理によって ColdFusion 変数の値が変更されます。
3. cfoutput タグによって、foo 変数の新しい値が出力されます。

戻り値を持つメソッドの使用

値を返す CORBA メソッドは、ColdFusion 関数と同じ方法で使用します。次に例を示します。

IDL	double method(out double a);
CFML	<cfset foo=3.1415> <cfset ret=handle.method("foo")> <cfoutput>#ret#</cfoutput>

ColdFusion 変数での IDL 型の使用

ColdFusion では、特定の CORBA データ型がサポートされており、CORBA 型と ColdFusion データの間で変換が行われます。

IDL サポート

次の表では、CORBA IDL のそれぞれの型について、ColdFusion でサポートされているかどうか、およびパラメータまたは戻り値として使用できるかどうかを示します。該当するものがない場合もあります。

CORBA IDL 型	一般的なサポート	パラメータとしての使用	戻り値としての使用
constants	不可	不可	不可
attributes	可 (プロパティ用)	該当なし	該当なし
enum	可 (整数として)	可	可
union	不可	不可	不可
sequence	可	可	可
array	可	可	可
interface	可	可	可
typedef	可	該当なし	該当なし
struct	可	可	可
module	可	該当なし	該当なし
exception	可	該当なし	該当なし
any	不可	不可	不可
boolean	可	可	可
char	可	可	可
wchar	可	可	可
string	可	可	可
wstring	可	可	可
octet	可	可	可
short	可	可	可
long	可	可	可
float	可	可	可
double	可	可	可
unsigned short	可	可	可

CORBA IDL 型	一般的なサポート	パラメータとしての使用	戻り値としての使用
unsigned long	可	可	可
longlong	不可	不可	不可
unsigned longlong	不可	不可	不可
void	可	該当なし	可

データ型の変換

次の表に、IDL のデータ型とそれに対応する ColdFusion のデータ型を示します。

IDL 型	ColdFusion 型
boolean	ブール値
char	1 文字からなる文字列
wchar	1 文字からなる文字列
string	文字列
wstring	文字列
octet	1 文字からなる文字列
short	整数
long	整数
float	実数
double	実数
unsigned short	整数
unsigned long	整数
void	該当なし (空の文字列として返されます)
struct	構造体
enum	整数 (0 が enum 型の最初の列挙子に対応します)
array	配列 (IDL で指定された配列サイズに一致する必要があります)
sequence	配列
interface	オブジェクト参照
module	サポートなし (モジュール名による逆参照は不可能です)
exception	ColdFusion で coldfusion.runtime.corba.CorbaUserException 型の例外が発生します。
attribute	ドット表記法を使用したオブジェクト参照

ブール値データについての注意事項

次の値は、ColdFusion ではすべてブール値として扱われます。

True	"yes"、"true"、または 1
False	"no"、"false"、または 0

次のコードのように、これらのすべての値は、ブール値のパラメータを取る CORBA メソッドで使用できます。

IDL	`module Tester`
	`{`
	`interface TManager`
	`{`
	`void testBoolean(in boolean a);`
	`void testOutBoolean(out boolean a);`
	`void testInoutBoolean(inout boolean a);`
	`boolean returnBoolean();`
	`}`
	`}`

CFML	`<cfset handle = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "") >`
	`<cfset ret = handle.testboolean("yes")>`
	`<cfset mybool = True>`
	`<cfset ret = handle.testoutboolean("mybool")>`
	`<cfoutput>#mybool#</cfoutput>`
	`<cfset mybool = 0>`
	`<cfset ret = handle.testinoutboolean("mybool")>`
	`<cfoutput>#mybool#</cfoutput>`
	`<cfset ret = handle.returnboolean()>`
	`<cfoutput>#ret#</cfoutput>`

Struct データ型についての注意事項

IDL の struct 型には ColdFusion 構造体を使用します。ColdFusion 構造体のキー名と IDL struct のフィールド名で大文字と小文字を揃えることによってエラーを回避できます。

Enum 型についての注意事項

ColdFusion では、IDL の enum 型は 0 から始まるインデックスを持つ整数として扱われます。したがって、最初の列挙子は 0 に対応し、2 番目の列挙子は 1 に対応します。次の例では、IDL の列挙子 a は 0 に、b は 1 に、c は 2 に対応します。

IDL	`module Tester`
	`{`
	`enum EnumType {a, b, c};`
	`interface TManager`
	`{`
	`void testEnum(in EnumType a);`
	`void testOutEnum(out EnumType a);`
	`void testInoutEnum(inout EnumType a);`
	`EnumType returnEnum();`
	`}`
	`}`

CFML	`<cfset handle = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "") >` `<cfset ret = handle.testEnum(1)>`

この例では、CORBA オブジェクトの列挙子の (1 番目ではなく) 2 番目のエントリが呼び出されます。

ダブルバイト文字についての注意事項

CORBA 2.0 以降のバージョンをサポートする ORB を使用している場合は、特別な作業をしなくてもダブルバイト文字がサポートされます。ColdFusion の文字列および文字は、使用時に wstring および wchar 型のデータに適切に変換されます。ただし、CORBA 2.0 の IDL 仕様では wchar および wstring 型がサポートされていないので、8 ビット Latin-1 文字セットを使用して文字列データが表されます。この場合、それらの文字が含まれているパラメータを渡すことはできませんが、ColdFusion 文字列データを使用して char および string 型のパラメータを呼び出すことはできます。

例外処理

リモートサーバーで発生する CORBA オブジェクトメソッドの例外を検出するには、次のように、cftry および cfcatch タグを使用します。

1 CORBA 例外を検出するには、cfcatch タグで type="coldfusion.runtime.corba.CorbaUserException" を指定します。

2 例外オブジェクトの内容を取得するには、cfcatch.getContents メソッドを使用します。

cfcatch.getContents メソッドは、IDL によって指定された例外データが含まれている ColdFusion 構造体を返します。

次のコード例は、PrimitiveException 例外タイプ定義によって定義された例外を投げる CORBA オブジェクトの IDL と、例外を検出してオブジェクトの内容を表示する CFML です。

IDL	`interface myInterface`
	`{`
	`exception PrimitiveException`
	`{`
	`long l;`
	`string s;`
	`float f;`
	`};`
	`void testPrimitiveException() raises (PrimitiveException);`
	`}`

CFML	`<cftry>`
	`<cfset ret0 = handle.testPrimitiveException()>`
	`<cfcatch type=coldfusion.runtime.corba.CorbaUserException>`
	`<cfset exceptStruct= cfcatch.getContents()>`
	`<cfdump var ="#exceptStruct#">`
	`</cfcatch>`
	`</cftry>`

CORBA の例

次のコードは、LoanAnalyzer CORBA オブジェクトの使用例を示しています。この簡単なオブジェクトでは、与えられた情報に基づいて、申請者に対する貸し付けを承認するかどうかを判断します。

LoanAnalyzer CORBA インターフェイスにはメソッドが 1 つあり、このメソッドは次の 2 つの引数を取ります。

- 申請者のアカウントを識別する Account 構造体。これには、アカウント所有者と申請者の年齢および収入を表す Person 構造体が含まれています。
- ユーザーが現在所有しているクレジットカードを表す CreditCards シーケンス。クレジットカードタイプは、CardType 列挙子のメンバーによって表されます。この例では、申請者が同じ種類のカードを複数枚持っていないものとしています。

このオブジェクトは、申請が承認されるか拒否されるかを示すブール値を返します。

CFML では、次の処理を行います。

1. オブジェクトメソッドで使用する ColdFusion 変数を初期化します。実際のコードでは、フォームやクエリーから取得した情報を使用します。

   Person 構造体と Account 構造体のコードは単純です。申請者のクレジットカードを表す cards 変数は複雑です。インターフェイス IDL では、列挙子のシーケンスを使用してカードが表されています。ColdFusion では、IDL シーケンスは配列として表され、列挙子は 0 から始まるインデックスを持つ数値で表されます。この数値は、列挙子タイプ定義の項目の中での選択された項目の位置を示します。

   この例の場合、申請者はマスターカード、Visa カード、およびダイナースカードを持っています。マスターカード (MC) は列挙子タイプ定義の 1 番目のエントリなので、ColdFusion では数値 0 で表されます。Visa は 3 番目のエントリなので、2 で表されます。ダイナースは、5 番目のエントリなので、4 で表されます。このシーケンスを表すために、これらの数値を配列に挿入します。その結果 0、2、4 の 3 つの要素から構成される 1 次元配列が作成されます。
2. CORBA オブジェクトをインスタンス化します。
3. CORBA オブジェクトの approve メソッドを呼び出し、戻り変数 ret に結果を取得します。
4. ret 変数の値 (Yes または No) を表示します。

   IDL

```
struct Person
{
long pid;
string name;
string middle;
string last_name;
}
struct Account
{
Person person;
short age;
double income;
}
double loanAmountl
enum cardType {AMEX, VISA, MC, DISCOVER, DINERS};
typedef sequence<cardType> CreditCards;
interface LoanAnalyzer
{
boolean approve( in Account, in CreditCards);
}
```

CFML

```
<!--- Declare a "person" struct ---->
<cfset p = StructNew()>
<cfif IsStruct(p)>
<cfset p.pid = 1003232>
<cfset p.name = "Eduardo">
<cfset p.middle = "R">
<cfset p.last_name = "Doe">
</cfif>
<!---- Declare an "Account" struct --->
<cfset a = StructNew()>
<cfif IsStruct(a)>
<cfset a.person = p>
<cfset a.age = 34>
<cfset a.income = 150120.50>
</cfif>
<!----- Declare a "CreditCards" sequence --->
<cfset cards = ArrayNew(1)>
<cfset cards[1] = 0> <!--- corresponds to Amex --->
<cfset cards[2] = 2> <!--- corresponds to MC --->
<cfset cards[3] = 4> <!--- corresponds to Diners --->
<!---- Creating a CORBA handle using the Naming Service---->
<cfset handle = CreateObject("CORBA", "FirstBostonBank/MA/Loans",
"NameService") >
<cfset ret=handle.approve(a, cards)>
<cfoutput>Account approval: #ret#</cfoutput>
```

第 16 章：外部リソースの使用

電子メールの送受信

cfmail タグと cfpop タグを使用すれば、Adobe ColdFusion アプリケーションにインタラクティブな電子メール機能を追加できます。メールサーバーへの双方向インターフェイスを完備した ColdFusion の電子メール機能は、ユーザーに対する重要な連絡手段を提供します。

ColdFusion とメールサーバーの併用

ColdFusion アプリケーションに電子メール機能を追加すれば、ユーザーのリクエストに自動的に応答することができます。これによって、次のようなさまざまなことが実現できます。

- ユーザーのリクエストや命令に基づいて、電子メールメッセージを送信する。
- ユーザーが補足情報やドキュメントを電子メールを通じてリクエストし、受け取れるようにする。
- 受注エントリや更新情報に基づいて顧客情報を確認する。
- データベースクエリーから取得した情報を利用して、請求書や督促状を送信する。

電子メール機能をアプリケーションに統合するには、いくつかの方法があります。電子メールを送信するには、通常、SMTP (Simple Mail Transfer Protocol) を使用します。電子メールを受信するには、POP (Post Office Protocol) を使用してメールサーバーから電子メールを取得します。ColdFusion アプリケーションで電子メール送受信機能を使用するには、SMTP サーバーへのアクセス権限または POP アカウント (あるいはその両方) が必要です。

ColdFusion アプリケーションページで電子メールを送受信するには、cfmail タグと cfpop タグを使用します。

ColdFusion でのメール送信の仕組み

ColdFusion では、SMTP メール機能を実装するためにスプールアーキテクチャを利用しています。ColdFusion Administrator の [メール] ページでメールのスプールをオンにしている場合は、アプリケーションページで cfmail タグを処理しても、生成されたメッセージはすぐには送信されません。メッセージはディスクにスプールされ、バックグラウンドで処理されます。このアーキテクチャには、次の 2 つの利点があります。

- SMTP 処理が完了する前に、アプリケーションのエンドユーザーにページを返すことができます。この機能は、多数のメッセージを送信するユーザーアクションに対して特に役立ちます。
- 停電やサーバーのクラッシュなどの予期しない事態が発生しても、cfmail タグを使用して送信したメッセージは確実に配送されます。

スプールされたメールメッセージを ColdFusion が確認する頻度を、ColdFusion Administrator の [メール] ページで設定できます。ただし、ColdFusion がビジーである場合や、大量のメッセージがキューに存在している場合は、スプール間隔を過ぎてから送信されることがあります。

ColdFusion の一部のエディションには、メールの送信処理を詳細に調整できる高度なスプールオプションがあります。詳細については、『ColdFusion 設定と管理』を参照してください。

エラーロギングと未送信メッセージ

SMTP メッセージの処理時に発生したすべてのエラーは、ColdFusion ログディレクトリ内の "mail.log" ファイルに記録されます。ログエントリには、エラーが発生した日時と、発生原因に関する診断情報が含まれています。

エラーが原因でメッセージが送信されなかった場合は、次のディレクトリにその情報が書き込まれます。

- Windows の場合 : ¥<ColdFusion のインストールディレクトリ >¥Mail¥Undelivr
- UNIX の場合 : /opt/<ColdFusion のインストールディレクトリ >/mail/undelivr

未送信メッセージのエラーログエントリには、"UnDelivr" (または "undelivr") ディレクトリに保存されたファイルの名前が含まれています。

注意 : 送信できなかったメッセージを再送信するには、メッセージファイルを "Undelivr" ディレクトリから "Spool" ディレクトリに移動します。

ColdFusion Administrator のメールロギング設定については、『ColdFusion 設定と管理』を参照してください。

電子メールメッセージの送信

ColdFusion で電子メールメッセージの送信設定を行うには、SMTP 電子メールサーバーへのアクセス権を取得している必要があります。また、電子メールサーバーにアクセスするアプリケーションページを実行する前に、SMTP サーバーにアクセスできるように ColdFusion Administrator を設定する必要があります。特定のメッセージで、ColdFusion Administrator の SMTP サーバー設定と異なる設定を使用する必要がある場合は、cfmail タグの server 属性で新しいメールサーバーを指定します。

ColdFusion の電子メール機能の設定

1. ColdFusion Administrator で、[サーバーの設定]-[メール] を選択します。
2. [メールサーバー] ボックスに、SMTP メールサーバーの名前または IP アドレスを入力します。
3. (オプション) [サーバーポート] および [接続タイムアウト] のデフォルト設定を変更します。
4. [メールサーバーの接続の確認] オプションを選択し、ColdFusion がメールサーバーにアクセスできるようにします。
5. メールサーバーのポートが 25 (デフォルト SMTP ポート) でない場合は、[サーバーポート] のデフォルト設定を変更します。
6. ColdFusion の一部のエディションでは、メールの送信処理を設定および最適化できる追加のオプションが Administrator の [メール] ページに用意されています。それらのオプションを必要に応じて選択します。
7. 「変更の送信」をクリックします。

   設定が保存されます。サーバーへの接続に成功したか失敗したかを示すメッセージがページに表示されます。

ColdFusion エンタープライズ版には、メールのスプールと送信に関する追加機能があります。それらの機能の詳細と、ColdFusion Administrator のメール設定の詳細については、『ColdFusion 設定と管理』を参照してください。

cfmail タグによる SMTP 電子メールの送信

cfmail タグでは、ColdFusion アプリケーションから SMTP 電子メールを送信するための機能がサポートされています。cfmail タグは cfoutput タグに似ていますが、cfmail タグを使用した場合は、生成テキストがページではなく SMTP メールメッセージとして出力される点が異なります。query など、cfoutput タグで使用できる属性およびコマンドは、cfmail タグでもすべてサポートされています。次の表で、単純な電子メールメッセージの送信に使用する cfmail タグの基本的な属性について説明します。すべての属性のリストについては、『CFML リファレンス』の cfmail の説明を参照してください。

属性	説明
subject	メッセージの件名
from	送信者の電子メールアドレス

属性	説明
to	受信者の電子メールアドレス。複数の受信者を指定するには、受信者をカンマで区切ります。
cc	(オプション) カーボンコピー受信者の電子メールアドレス。この受信者のアドレスは、送信したメール内に表示されます。複数の cc 受信者を指定するには、受信者をカンマで区切ります。
bcc	(オプション) ブラインドカーボンコピー受信者の電子メールアドレス。この受信者のアドレスは、送信したメール内に表示されません。複数の bcc 受信者を指定するには、受信者をカンマで区切ります。

簡単な電子メールメッセージの送信

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Sending a simple e-mail</title>
</head>
<body>
<h1>Sample e-mail</h1>
<cfmail
    from="Sender@Company.com"
    to="#URL.email#"
    subject="Sample e-mail from ColdFusion">

This is a sample e-mail message to show basic e-mail capability.

</cfmail>
The e-mail was sent.

</body>
</html>
```

2 このファイルに "send_mail.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

3 ブラウザを開き、次の URL を入力します。

http://localhost:8500/myapps/send_mail.cfm?email=myname@mycompany.com

myname@mycompany.com は自分の電子メールアドレスに置き換えてください。

このページをリクエストすると、電子メールメッセージが SMTP サーバーを経由してユーザーに送信されます。

注意：電子メールメッセージを受信できない場合は、ColdFusion から SMTP サーバーへのアクセスが適切に設定されているかどうかを調べてください。詳細については、1180 ページの「電子メールメッセージの送信」を参照してください。

cfmail タグには、メールのカスタマイズや送信方法の制御に使用できる多数のオプションがあります。メールテキストを指定の桁数で折り返すオプション、メールの文字エンコードを指定するオプション、メールサーバー、ユーザー名、パスワードを指定するオプションなど、すべての属性の詳細については、『CFML リファレンス』で cfmail の説明を参照してください。

HTML 形式の電子メールの送信

メールメッセージに含まれる HTML コードを解釈および表示できるアプリケーションをすべての受信者が使用している場合は、cfmail タグを使用して HTML メッセージを送信できます。cfmail タグで type="HTML" 属性を指定すると、処理が必要な HTML タグがメッセージに含まれていることが、受信者の電子メールクライアントに通知されます。HTML メールを送信する例については、1187 ページの「メッセージへのイメージの挿入」を参照してください。

マルチパート仕様のメールメッセージの送信

cfmailpart タグを使用すると、パートごとに MIME タイプや文字セットが異なる、マルチパート仕様のメールメッセージを作成できます。たとえば、すべての受信者が HTML メールメッセージを解釈できるかどうか不明な場合は、テキストのパートと HTML のパートを含むマルチパートのメールとしてメッセージを送信できます。その場合は、次の例のように cfmailpart タグを 2 つ使用して、1 つを HTML バージョンのメッセージ、もう 1 つをプレーンテキストのメッセージにします。この例をテストするには、To 属性の値を有効な電子メールアドレスに置き換え、ページを保存および実行して、指定したアドレスに届く電子メールの内容を確認します。

```
<cfmail from = "peter@domain.com" To = "paul@domain.com"
Subject = "Which version do you see?">
    <cfmailpart
        type="text"
        wraptext="74">
            You are reading this message as plain text, because your mail reader
            does not handle HTML text.
    </cfmailpart>>
    <cfmailpart
        type="html">
            <h3>HTML Mail Message</h3>
            <p>You are reading this message as <strong>HTML</strong>.</p>
            <p>Your mail reader handles HTML text.</p>
    </cfmailpart>
</cfmail>
```

注意：HTML バージョンのメッセージでは、シャープ記号 (色指定に使用するものなど) をすべてエスケープします。エスケープするには、bgcolor="##C5D9E5" のように # 記号を 2 つ続けて記述します。

cfmail タグの使用例

cfmail タグを使用したアプリケーションページでは、タグの設定に基づいて電子メールメッセージをダイナミックに生成できます。cfmail で実行できるタスクの一部を次に示します。

- HTML フォームにユーザーが入力したデータに基づいて、受信者と内容を設定し、メールメッセージを送信する。
- クエリーを使用して、受信者のリストをデータベースから取得し、メールメッセージを送信する。
- クエリーを使用して、受信者のリストをデータベースからダイナミックに取得し、請求書などのカスタムメールメッセージを送信する。

フォームデータを利用した電子メールの送信

次の例では、顧客問い合わせフォームの内容をマーケティング部門に送信します。同じアプリケーションページを使用して、顧客問い合わせフォームの内容をデータベースに挿入することもできます。フォームに次のコードを含めると、ユーザーが情報を入力してフォームを送信したときにそのコードが実行されます。

```
<cfmail
    from="#Form.EMailAddress#"
    to="marketing@MyCompany.com,sales@MyCompany.com"
    subject="Customer Inquiry">

A customer inquiry was posted to our website:

Name: #Form.FirstName# #Form.LastName#
Subject: #Form.Subject#

#Form.InquiryText#
</cfmail>
```

クエリーデータを利用した電子メールの送信

次の例では、クエリー (ProductRequests) を使用して、過去 1 週間に製品について問い合わせのあった顧客のリストを取得します。リストに適切なヘッダとフッタを付けてマーケティング部門に送信します。

```
<cfmail
    query="ProductRequests"
    from="webmaster@MyCompany.com"
    to="marketing@MyCompany.com"
    subject="Widget status report">

Here is a list of people who have inquired about
MyCompany Widgets during the previous seven days:

<cfoutput>
#ProductRequests.FirstName# #ProductRequests.LastName# (#ProductRequests.Company#) -
#ProductRequests.EMailAddress#&###013;
</cfoutput>

Regards,
The webmaster
webmaster@MyCompany.com

</cfmail>
```

コードの説明

このコードについて、次の表で説明します。

コード	説明

`<cfoutput>`
`#ProductRequests.FirstName# #ProductRequests.LastName#`
`(#ProductRequests.Company#) -`
`#ProductRequests.EMailAddress#&###013;`
`</cfoutput>`
	標準的なメッセージの中に、ProductRequests クエリーの各行から生成したダイナミックリストを挿入します。cfmail タグでクエリーを指定しているので、cfoutput タグでは query 属性を使用しません。&###013; は出力レコード間に改行を挿入します。

複数の受信者への電子メールの送信

複数の受信者に電子メールを送信する方法としては、cfmail タグの to 属性に受信者のリストをカンマで区切って指定する方法の他に、cfmail タグの query 属性を使用する方法もあります。ここでは、複数の受信者に同じメッセージを送信する例と、受信者に応じてメッセージをカスタマイズする例を紹介します。

複数の受信者への単純なメッセージの送信

次の例では、クエリー (BetaTesters) を使用して、ColdFusion のベータテストを行っているユーザーのリストを取得します。次に、新リリースが入手可能であることをそれらのユーザーに知らせます。cfmail タグ本文の内容はダイナミックではありません。ダイナミックなのは、メッセージの送信先である電子メールアドレスのリストです。Betas テーブルの TesterEmail 列を参照する変数 #TesterEMail# を to 属性で使用することで、ダイナミックリストを指定しています。

```
<cfquery name="BetaTesters" datasource="myDSN">
    SELECT * FROM BETAS
</cfquery>

<cfmail query="BetaTesters"
    from="beta@MyCompany.com"
    to="#BetaTesters.TesterEMail#"
    subject="Widget Beta Four Available">

To all Widget beta testers:

Widget Beta Four is now available
for downloading from the MyCompany site.
The URL for the download is:

http://beta.mycompany.com

Regards,
Widget Technical Support
beta@MyCompany.com

</cfmail>
```

複数の受信者へのカスタム電子メールの送信

次の例では、クエリー (GetCustomers) を使用して、一連の顧客の連絡先を取得します。次に、連絡先が正しいかどうかを顧客に確認する電子メールを送信します。

```
<cfquery name="GetCustomers" datasource="myDSN">
    SELECT * FROM Customers
</cfquery>

<cfmail query="GetCustomers"
    from="service@MyCompany.com"
    to="#GetCustomers.EMail#"
    subject="Contact Info Verification">

Dear #GetCustomers.FirstName# -

We'd like to verify that our customer
database has the most up-to-date contact
information for your firm. Our current
information is as follows:

Company Name: #GetCustomers.Company#
Contact: #GetCustomers.FirstName# #GetCustomers.LastName#

Address:
    #GetCustomers.Address1#
    #GetCustomers.Address2#
    #GetCustomers.City#, #GetCustomers.State# #GetCustomers.Zip#

Phone: #GetCustomers.Phone#
Fax: #GetCustomers.Fax#
Home Page: #GetCustomers.HomePageURL#

Please let us know if any of this
information has changed, or if we must
get in touch with someone else in your
organization regarding this request.

Thanks,
Customer Service
service@MyCompany.com

</cfmail>
```

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
<cfquery name="GetCustomers" datasource="myDSN"> SELECT * FROM Customers </cfquery>	GetCustomers というクエリーで、Customers テーブルのすべてのデータを取得します。
<cfmail query="GetCustomers" from="service@MyCompany.com" to="#GetCustomers.EMail#" subject="Contact Info Verification">	cfmail タグの to 属性で #GetCustomers.Email# クエリー列を指定することで、クエリーの各行に含まれているアドレスに、メッセージを 1 通ずつ個別に送信します。したがって、メール本文では cfoutput タグを使用しません。
Dear #GetCustomers.FirstName# - ..... Company Name: #GetCustomers.Company# Contact: #GetCustomers.FirstName# #GetCustomers.LastName# Address: #GetCustomers.Address1# #GetCustomers.Address2# #GetCustomers.City#, #GetCustomers.State# #GetCustomers.Zip# Phone: #GetCustomers.Phone# Fax: #GetCustomers.Fax# Home Page: #GetCustomers.HomePageURL#	cfmail セクションでは #GetCustomers.FirstName#、#GetCustomers.LastName# などのクエリー列を使用して、メッセージの内容を受信者ごとにカスタマイズします。

電子メールへの電子署名

メールに電子署名を追加するには、次の例に示すように、sign、keystore、keystorepassword、keyalias および keypassword 属性を指定します。

```
<cfmail from="Sender@Company.com" server="sendmail.myCo.com" sign="true"
keystore="C:\OpenSSL\bin\hello.jks" keystorepassword="digital" to="Recepient@Company.com"
keyalias="crypto" keypassword="signature" subject="Mail with Digital Signature">
```

送信するすべてのメッセージに電子署名を追加するには、タグに属性を追加するのではなく、ColdFusion Administrator の [サーバーの設定]-[設定] ページで設定を指定します。

サポートされるキーストアは JKS および PKCS12 です。

一部の国での輸入規制により、ポリシーファイル (local_policy.jar および US_export_policy.jar) でサポートされる暗号の強度は制限されています。キーの強度が制限を超えている場合、キーストアをロードできないことを示すエラーが表示される可能性があります。強度無制限バージョンのポリシーファイルを使用することが認められている国に在住している場合は、これらのポリシーファイルをダウンロードして、デフォルトの暗号 JAR ファイルと入れ換えることができます。これらのファイルは Java SDK Web サイトで入手できます。

cfmailparam タグの使用

cfmailparam タグを使用すると、メッセージにファイルを含めることや、電子メールメッセージにカスタムヘッダを追加することができます。ファイルは添付ファイルとして送信することも、メッセージ内にインライン表示することもできます。cfmailparam タグは cfmail タグ内にネストします。

メッセージへのファイルの添付

次の例のように、添付するファイルごとに cfmailparam タグを 1 つ使用します。

```
<cfmail from="daniel@MyCompany.com"
    to="jacob@YourCompany.com"
    subject="Requested Files">

Jake,

Here are the files you requested.

Regards,
Dan

<cfmailparam file="c:\widget_launch\photo_01.jpg">
<cfmailparam file="c:\widget_launch\press_release.doc">

</cfmail>
```

cfmailparam の file 属性には完全修飾システムパスを使用します。ファイルは、ブラウザマシンではなく、ColdFusion サーバーマシンのドライブ、またはローカルネットワーク上に配置する必要があります。

メッセージへのイメージの挿入

cfmailparam を使用すると、次のようにして、HTML メッセージの中に他のファイルのイメージを含めることができます。

1. cfmail 開始タグに続いて、個々のイメージごとに cfmailparam タグを記述します。
2. 各 cfmailparam タグ内で、次のことを行います。
   - file 属性でイメージの場所を指定します。
   - disposition="inline" を指定します。
   - contentID 属性で、一意の識別子 (myImage1 など) を指定します。
3. HTML 内のイメージを配置したい場所に、次のようにして img タグを記述します。

```
<img src="cid:myImage1">
```

次の例は、インラインイメージを含んだ単純なメールメッセージです。この場合はイメージを段落と段落の間に配置していますが、テキスト内に直接インラインで含めることもできます。この例をテストするには、cfmail の to パラメータを有効な電子メールアドレスで置き換え、file パラメータを有効なイメージのパスに変更します。

```
<cfmail type="HTML"
        to = "Peter@myCo.com"
        from = "Paul@AnotherCo.com"
        subject = "Sample inline image">
    <cfmailparam file="C:\Inetpub\wwwroot\web.gif"
        disposition="inline"
        contentID="image1">
    <P>There should be an image here</p>
    <img src="cid:image1">
    <p> This text follows the picture</p>
</cfmail>
```

メッセージへのカスタムヘッダの追加

電子メールメッセージの受信者がそのメッセージに返信する場合、デフォルトでは元のメッセージの From フィールドに指定されたアドレスに返信されます。cfmailparam タグを使用して Reply-To 電子メールアドレスを指定し、From フィールドを上書きするようメールクライアントに指示することができます。次の例では、cfmailparam を使用して返信アドレスを widget_master@YourCompany.com に設定しています。

```
<cfmail from="jacob@YourCompany.com"
    to="daniel@MyCompany.com"
    subject="Requested Files">
<cfmailparam name="Reply-To" value="widget_master@YourCompany.com">

Dan,
Thanks very much for the sending the widget press release and graphic.
I'm now the company's Widget Master and am accepting e-mail at
widget_master@YourCompany.com.

See you at Widget World 2002!

Jake
</cfmail>
```

注意：1 つの ColdFusion ページで、2 つの用途の cfmailparam タグを組み合わせて使用することもできます。ヘッダおよび添付ファイルごとに cfmailparam タグを記述します。

電子メールメッセージの受信

電子メールのメッセージ情報を取得するには、POP (Post Office Protocol) サーバーにアクセスする ColdFusion ページを作成します。取得したメッセージやヘッダ情報は、画面に表示したり、データベースに書き込んだりすることができます。

cfpop タグを使用すると、インターネットメールクライアントの機能や、電子メールのデータを利用して処理を行う機能をアプリケーションに追加できます。従来のメールクライアントにも個人メール用に十分なインターフェイス機能が備わっていますが、特定のメールボックスに代替インターフェイスを用意すると有用な場合が多くあります。cfpop タグを使用すれば、さまざまなアプリケーションのニーズに合ったメールクライアントを開発できます。主要な電子メールプロトコルには、POP の他に IMAP (Internet Mail Access Protocol) がありますが、cfpop タグは IMAP では動作しません。

POP メール機能を実装すると有用な場合としては、次の 3 つの例があります。

- 複数のユーザーが利用する汎用的なメールボックス (**sales@yourcompany.com** など) がある場合は、個々のユーザーメールクライアントを補う ColdFusion のメールフロントエンドを構築すると役立ちます。
- 多くのアプリケーションでは、特定の目的で形式設定されているメール (たとえば、リストサーバーへの参加依頼のメールなど) の処理を自動化することができます。
- データベースに電子メールメッセージを保存する場合。

POP サーバーに対して cfpop を実行することは、メールボックスの内容に対してクエリーを実行するのに似ています。action 属性でアクションを設定し (ヘッダを取得する場合は GetHeaderOnly、メッセージ全体を取得する場合は GetAll)、name 属性で名前を指定します。cfoutput タグを使用する場合などに cfpop で返されたレコードセットにアクセスするには、その名前を使用します。POP サーバーにアクセスするには、server、username、および password 属性も定義する必要があります。

注意：cfpop タグでは、メッセージの取得時に発生したエラー (たとえば、電子メールメッセージの形式が不正であるために発生したエラー) は可能な限り無視され、空のフィールドを含んだ結果の構造体と、取得できたすべてのメッセージが返されます。

cfpop タグのシンタックスおよび変数の詳細については、『CFML リファレンス』を参照してください。

cfpop タグの使用

アプリケーションに POP メールを追加するには、次の手順に従います。

アプリケーションでの cfpop タグの実装

1 ColdFusion アプリケーションでアクセスするメールボックスを選択します。

2 メールメッセージのどの部分 (メッセージヘッダ、メッセージ本文、添付ファイルなど) を処理するか決めます。

3 取得したメッセージをデータベースに保管するかどうかを決めます。

4 メッセージを取得した後に、そのメッセージを POP サーバーから削除するかどうかを決めます。

5 アプリケーションに cfpop タグを記述して、メールボックスにアクセスするためのユーザーインターフェイスを作成します。

6 出力を処理するアプリケーションページを構築します。取得したメッセージには、ブラウザで正しく表示できない文字が含まれている場合があります。

ブラウザへの出力を制御するには、cfoutput タグで HTMLCodeFormat 関数や HTMLEditFormat 関数を使用します。これらの関数は、不等号 (<、>) やアンパサンド (&) などの、HTML で特別な意味を持つ文字を、<、>、& などの HTML エスケープ文字に変換します。また HTMLCodeFormat タグは、テキストを pre タグブロックで囲みます。

cfpop のクエリー変数

通常の ColdFusion クエリーと同様に、cfpop クエリーでも、レコードに関する情報を含んだ次の変数が返されます。

RecordCount クエリーによって返されたレコードの総数

ColumnList クエリーによって返された列の見出しのリスト

CurrentRow cfoutput または cfloop のクエリーループで現在処理されているクエリー行

このクエリーには、cfquery タグでは返されない、UID という変数が含まれています。この変数には、電子メールメッセージファイルを表す一意の識別子が含まれています。

cfoutput タグでこれらのプロパティを参照するには、クエリー変数の前に、cfpop の name 属性で指定したクエリー名を付加します。

```
<cfoutput>
This operation returned #Sample.RecordCount# messages.
</cfoutput>
```

POP メールの処理

メールを管理するには、cfpop タグを使用します。このタグでは、アクションの対象とするメッセージを指定できます。また、メッセージヘッダ、メッセージ、添付ファイルを取得したり、メッセージを削除することができます。

メッセージの指定

cfpop のすべてのアクションは、そのアクションをすべてのメッセージに対して行うか、選択したメッセージに対して行うかを指定できます。すべてのメッセージに対して行う (たとえば、すべてのメッセージヘッダを取得する) 場合は、messageNumber 属性と UID 属性を指定しません。特定のメッセージに対して行う (たとえば、選択した 3 つのメッセージを削除する) 場合は、messageNumber 属性または UID 属性で、実行対象のメッセージをカンマで区切って指定します。

メッセージヘッダの取得

メッセージを取得せずにメッセージヘッダのみを取得するには、cfpop タグで action="GetHeaderOnly" と指定します。ヘッダのみを取得する場合も、メッセージ全体を取得する場合も、cfpop で返されるクエリーオブジェクトには、指定した受信箱のメッセージごとに 1 つの行が含まれています。クエリーオブジェクトの名前は、cfpop タグの name 属性で指定します。クエリーには次のフィールドがあります。

- date
- from
- header (すべてのメールヘッダフィールドを含む文字列。クエリーオブジェクト内に別途フィールドがあるエントリも含まれます)

- messageNumber (POP サーバーにおけるメッセージの通し番号。クエリーオブジェクトにおける該当するエントリの行番号と同じです)
- messageID (メールヘッダの Message-ID フィールド)
- replyTo
- subject
- cc
- to
- UID (メールヘッダの X-UID フィールド)

cfpop タグで getHeaderOnly 属性を指定した場合、attachmentPath 属性を指定していれば、すべての添付ファイルが取得されます。指定していなければ取得されず、attachmentfiles 列は空の文字列になります。

メッセージヘッダのみの取得

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>POP Mail Message Header Example</title>
</head>

<body>
<h2>This example retrieves message header information:</h2>

<cfpop server="mail.company.com"
    username=#myusername#
    password=#mypassword#
    action="GetHeaderOnly"
    name="Sample">

<cfoutput query="Sample">
    MessageNumber: #HTMLEditFormat(Sample.messageNumber)# <br>
    To: #HTMLEditFormat(Sample.to)# <br>
    From: #HTMLEditFormat(Sample.from)# <br>
    Subject: #HTMLEditFormat(Sample.subject)# <br>
    Date: #HTMLEditFormat(Sample.date)#<br>
        Cc: #HTMLEditFormat(Sample.cc)# <br>
    ReplyTo: #HTMLEditFormat(Sample.replyTo)# <br><br>
</cfoutput>

</body>
</html>
```

2 コードの次の部分を変更して、有効な POP メールサーバー、ユーザー名、およびパスワードを指定します。

```
<cfpop server="mail.company.com"
    username=#myusername#
    password=#mypassword#
```

3 このファイルに "header_only.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、Web ブラウザで表示します。

このコードでは、メッセージヘッダを取得して、Sample という cfpop レコードセットに保存しています。レコードセットデータの操作方法の詳細については、433 ページの「クエリーオブクエリーの使用」を参照してください。

HTMLCodeFormat 関数を使用すると、詳細な電子メールアドレス情報を囲む不等号 (< や >) などのような、HTML で意味を持つ文字を、< や > などのエスケープ文字に置き換えることができます。

また、cfpop で返される日付は、ParseDateTime 関数を使用して処理できます。この関数は、引数に渡された POP の日付時刻オブジェクトを CFML の日付時刻オブジェクトに変換できます。

cfoutput タグでは、次の例のように、任意の列を参照できます。

```
<cfoutput>
    #ParseDateTime(queryname.date, "POP")#
    #HTMLCodeFormat(queryname.from)#
    #HTMLCodeFormat(queryname.messageNumber)#
</cfoutput>
```

メッセージの取得

cfpop タグで action="GetAll" と指定した場合は、getheaderonly の場合と同じ列に加えて、次の列が返されます。

- attachments (添付ファイル名のタブ区切りリスト)
- attachmentfiles (ローカルサーバーに取得された添付ファイルのパスのタブ区切りリスト。ファイルの取得は、attachmentpath 属性を指定した場合にのみ行われます)
- body
- htmlbody
- textbody

メッセージがマルチパートの場合、htmlbody フィールドと textbody フィールドには HTML パートの内容とプレーンテキストパートの内容がそれぞれ含まれ、body フィールドにはメッセージ内の最初のパートが含まれます。メッセージに含まれるパートが 1 つのみである場合は、body にメッセージの内容が含まれ、メッセージタイプに応じて htmlbody フィールドまたは textbody フィールドのいずれかにメッセージのコピーが含まれます。

メッセージ全体の取得

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head><title>POP Mail Message Body Example</title></head>

<body>
<h2>This example adds retrieval of the message body:</h2>
<cfpop server="mail.company.com"
    username=#myusername#
    password=#mypassword#
    action="GetAll"
    name="Sample">

<cfoutput query="Sample">
    MessageNumber: #HTMLEditFormat(Sample.messageNumber)# <br>
    To: #Sample.to# <br>
    From: #HTMLEditFormat(Sample.from)# <br>
    Subject: #HTMLEditFormat(Sample.subject)# <br>
    Date: #HTMLEditFormat(Sample.date)#<br>
    Cc: #HTMLEditFormat(Sample.cc)# <br>
    ReplyTo: #HTMLEditFormat(Sample.replyTo)# <br>
    <br>
    Body:<br>
    #Sample.body#<br>
    <br>
    Header:<br>
    #HTMLCodeFormat(Sample.header)#<br>
    <hr>
    </cfoutput>

</body>
</html>
```

2 コードの次の部分を変更して、有効な POP メールサーバー、ユーザー名、およびパスワードを指定します。

```
<cfpop server="mail.company.com"
    username=#myusername#
    password=#mypassword#
```

3 このファイルに "header_body.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、Web ブラウザで表示します。

この例では、CFML 関数を使用して本文の内容をエンコードしていません。したがって、ブラウザには、HTML メッセージがサポートされているメールプログラムと同じように、形式設定されたメッセージが表示されます。

メッセージと添付ファイルの取得

cfpop タグで attachmentpath 属性を使用して添付ファイルの格納先ディレクトリを指定すると、POP サーバーから添付ファイルが取得され、指定のディレクトリに保存されます。attachmentfiles フィールドには、添付ファイルの場所を示すタブ区切りリストが設定されます。不要になったテンポラリファイルを削除するには、cffile タグを使用します。指定したディレクトリが存在しない場合は、自動的に作成されます。そのためには、システム上でディレクトリを作成するのに必要な権限が ColdFusion に割り当てられている必要があります。

メッセージに添付ファイルがない場合、attachments 列と attachmentfiles 列には空の文字列が設定されます。

注意：CFML には、cfpop で返されたメール添付ファイルの名前を保存前に変更する手段は用意されていません。ColdFusion が動作しているファイルシステムで使用できない名前が添付ファイルに付けられている場合、その添付ファイルは保存できません。

添付ファイルを含むメッセージ全体の取得

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>POP Mail Message Attachment Example</title>
</head>

<body>
<h2>This example retrieves message header,
body, and all attachments:</h2>

<cfpop server="mail.company.com"
    username=#myusername#
    password=#mypassword#
    action="GetAll"
    attachmentpath="c:\temp\attachments"
    name="Sample">

<cfoutput query="Sample">
    MessageNumber: #HTMLEditFormat(Sample.MessageNumber)# <br>
    To: #HTMLEditFormat(Sample.to)# <br>
    From: #HTMLEditFormat(Sample.from)# <br>
    Subject: #HTMLEditFormat(Sample.subject)# <br>
    Date: #HTMLEditFormat(Sample.date)# <br>
    Cc: #HTMLEditFormat(Sample.cc)# <br>
    ReplyTo: #HTMLEditFormat(Sample.ReplyTo)# <br>
    Attachments: #HTMLEditFormat(Sample.Attachments)# <br>
    Attachment Files: #HTMLEditFormat(Sample.AttachmentFiles)# <br>
    <br>
    Body:<br>
    #Sample.body# <br>
    <br>
    Header:<br>
    HTMLCodeFormat(Sample.header)# <br>
    <hr>
</cfoutput>

</body>
</html>
```

2 コードの次の部分を変更して、有効な POP メールサーバー、ユーザー名、およびパスワードを指定します。

```
<cfpop server="mail.company.com"
    username=#myusername#
    password=#mypassword#
```

3 このファイルに "header_body_att.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、Web ブラウザで表示します。

注意：添付ファイルの保存時にファイル名が重複しないように、cfpop の generateUniqueFilenames 属性を Yes に設定してください。

メッセージの削除

cfpop タグを使用してメッセージを削除すると、そのメッセージはサーバーから完全に削除されます。デフォルトでは、取得したメッセージは POP メールサーバーに保持されます。メッセージを削除するには、cfpop タグの action 属性を Delete に設定します。削除するメッセージを指定するには、messagenumber 属性を使用します。この属性を省略すると、そのユーザーのメッセージがサーバーからすべて削除されます。

注意：メッセージ番号は、削除アクションを含んでいる POP メールサーバー通信が終了するたびに再割り当てされます。たとえば、POP メールサーバーから 4 つのメッセージを取得した場合、メッセージ番号 1、2、3、および 4 が返されます。cfpop タグでメッセージ 1 と 2 を削除すると、メッセージ 3 と 4 にはそれぞれ、メッセージ番号 1 と 2 が割り当てられます。

メッセージの削除

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>POP Mail Message Delete Example</title>
</head>

<body>
<h2>This example deletes messages:</h2>

<cfpop server="mail.company.com"
    username=#username#
    password=#password#
    action="Delete"
    messagenumber="1,2,3">

</body>
</html>
```

2 コードの次の部分を変更して、有効な POP メールサーバー、ユーザー名、およびパスワードを指定します。

```
<cfpop server="mail.company.com"
    username=#myusername#
    password=#mypassword#
```

3 このファイルに "message_delete.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、Web ブラウザで表示します。

4 作成した "header_only.cfm" ページを実行し、メッセージが削除されたことを確認します。

重要：Web ブラウザでこのページを表示すると、メッセージはすぐに **POP** サーバーから削除されます。

Microsoft Exchange サーバーとの対話

Adobe ColdFusion では、Microsoft Exchange サーバーと対話して、メールの送信、取得、および管理を行うことができます。また、カレンダーイベント、接続、およびタスクの作成、取得、および管理を行うこともできます。

ColdFusion での Microsoft Exchange サーバーの使用

ColdFusion は Microsoft Exchange サーバーと対話して、次のアクションを行うことができます。

アイテム	アクション
メールメッセージ	取得、添付ファイルの取得、会議情報の取得、別のフォルダへの移動、削除、添付ファイルの削除、プロパティの設定
カレンダーイベント	作成、取得、添付ファイルの取得、削除、添付ファイルの削除、変更、応答
連絡先	作成、取得、添付ファイルの取得、削除、添付ファイルの削除、変更
タスク	作成、取得、添付ファイルの取得、削除、添付ファイルの削除、変更

これらのアクションを行うには、次の ColdFusion タグを使用します。

タグ	用途
cfexchangeconnection	アプリケーションと Exchange サーバーの間の永続的な接続を開いて閉じます。 受信トレイのサブフォルダの情報を取得します。
cfexchangecalendar	カレンダーイベントを作成、取得、および管理します。
cfexchangecontact	連絡先を作成、取得、および管理します。
cfexchangemail	メールメッセージを取得および管理します。メールの送信は行いません。
cfmail	Exchange サーバーに電子メールを送信します。
cfexchangetask	タスクを作成、取得、および管理します。
cfexchangefilter	特定のアイテムを取得する条件を指定します。get アクションを指定した cfexchangecalendar、cfexchangecontact、cfexchangemail、および cfexchangetask タグの子としてのみ使用されます。

ColdFusion と Exchange サーバーを使用すれば、次のことが行えます。

- Exchange のカスタム Web クライアントインターフェイスを構築する。
- 予定されているタスクに関する情報を表示する。
- 連絡先エントリに基づいてメーリングリストを作成する。
- 新しいバグレポートや顧客連絡先に基づいて、ユーザーのタスクリストに自動的にタスクを追加する。
- 会議や予定のスケジュールを設定する。
- 会議参加者の出席予定を表示し管理する。

Exchange サーバーに対する接続の管理

Exchange サーバーと通信を行うには、サーバーとの接続を確立します。この接続では、HTTP プロトコルまたは HTTPS プロトコルを使用できます。デフォルトでは、ログインユーザー名に属するメールボックスに接続しますが、所有者がそのログインユーザー名にアクセス権を委任している任意のメールボックスにも接続できます。プロキシホストを使用してサーバーにアクセスすることもできます。

注意：接続を確立するには、Exchange サーバーでログインユーザーに Outlook Web アクセスを付与する必要があります。このアクセスを付与する方法については、1196 ページの「Outlook Web アクセスの有効化」を参照してください。

サーバーには永続的または一時的に接続することができます。

- 永続的な接続は、明示的に接続を閉じるまで保持されます。永続的な接続では、複数のタスクを 1 つの接続で実行できるので、Exchange サーバーと対話するたびに個別の接続を開いたり閉じたりする処理オーバーヘッドを減らすことができます。
- 一時的な接続は、Exchange サーバーと対話するタグが存続する間のみ保持されます。一時的な接続は、一連の連絡先を取得するだけの場合など、Exchange サーバーにアクセスする必要のあるタグが 1 つのみである場合に有用です。

Exchange サーバーへのアクセスの有効化

Exchange サーバーへのアクセスを有効にするには、次のことを確認します。

- Exchange サーバー、Exchange アクセス、および WebDav アクセスが IIS で設定されている。
- Exchange サーバーですべてのログインユーザーに対して Outlook Web アクセスが有効になっている。
- HTTPS を使用して Exchange サーバーにログインしている場合は、JRE 証明書ストアに有効なクライアント証明書がある。

IIS で Exchange サーバーへのアクセスが設定されていることの確認

1 Exchange サーバーがインストールされているマシンの [管理ツール] コントロールパネルから IIS マネージャを開きます。

2 左ペインのツリーで Web サイトノードを展開します。そこに Exchange が表示される場合は、Web アプリケーションは Exchange 用に設定されています。表示されない場合は、Microsoft の指示に従って Web サイトで Exchange を設定します。

3 左ペインのツリーで Web Service Extension ノードをクリックします。右側のペインに Web Service Extension とその状態が表示されます。Microsoft Exchange Server と WebDav のエントリが両方とも有効になっていることを確認します。禁止されている場合は、エントリを選択して [許可] ボタンをクリックします。

Outlook Web アクセスの有効化

接続を確立するには、Exchange サーバーでログインユーザーに Outlook Web アクセスを付与する必要があります。

Web アクセスの確認と許可

1 Exchange Administrator で、[Administrative Tools]-[Active Directory Users and Computers]-[< ドメイン名 >]-[users] を開きます。

2 接続を確立するのに使用する ID を持つユーザーを右クリックします。

3 [Exchange Features] タブを選択します。

4 [プロトコル] セクションで、Outlook Web アクセスのエントリが無効である場合は有効にします。

Exchange サーバーへの HTTPS アクセスの有効化

ColdFusion から Exchange サーバーへの HTTPS アクセスを有効にするには、次を行う必要があります。

- Exchange サーバーシステムで SSL を有効にする
- JRE 証明書ストアに有効なクライアント証明書があることを確認する

Exchange サーバーシステムでの SSL の有効化

Exchange サーバーシステムで SSL を有効にする手順は次のとおりです。

1 Exchange サーバーがインストールされているシステムで、[管理ツール] コントロールパネルから IIS マネージャを開きます。

2 左ペインのツリーで Web サイトノードを展開します。

3 展開したリストで Exchange および ExchWeb を右クリックして [Web Site Properties] ダイアログボックスを開き、[ディレクトリセキュリティ] タブをクリックします。

4 [セキュリティで保護された通信] セクションで、[編集] をクリックして [セキュリティで保護された通信] ダイアログボックスを開きます。[セキュリティで保護されたチャネル (SSL) を要求する] オプションを選択し、[OK] をクリックして、[適用] をクリックします。

手順 3 と 4 の代わりに、次のようにしてもかまいません。[既定の Web サイト] を右クリックします。[セキュリティで保護された通信]-[編集] で、[セキュリティで保護されたチャネル (SSL) を要求する] オプションを選択し、[OK] をクリックして、[適用] をクリックします。SSL を有効にするノード (Exchange など) を選択します。

ColdFusion サーバーでの HTTPS アクセスの有効化

HTTPS を使用して Exchange サーバーにアクセスするには、JRE 証明書ストアに有効なクライアント証明書が存在する必要があります。Exchange サーバーの証明書が既知の機関によって発行されたものでない場合は、証明書をインストールします。Java 証明書ストアには、既にいくつかの機関の証明書が格納されています。

システム管理者に依頼して ColdFusion サーバーにインストールする証明書を入手するか、または次を行います。

1 Internet Explorer で Outlook Web Access を開き、[File]-[Properties] に移動します。

2 証明書ボタンをクリックします。

3 [Details] タブをクリックし、タブの [Copy To File] ボタンをクリックします。ウィザードに従って証明書を保存します。

証明書をインストールするには、keytool.exe を使用して次のコマンドを実行します。このツールは、"jre¥bin" フォルダにあります。

```
keytool.exe -importcert -file <path_to_certificate_file> -keystore ..\lib\security\cacerts
```

注意：keytool.exe プログラムを使用するには、パスワードを入力する必要があります。デフォルトのパスワードは changeit です。

永続的な接続の使用

永続的な接続を開くには、cfexchangeconnection タグを使用して、open アクション、サーバーの IP アドレスまたは URL、ユーザー名、および接続名 (後続のタグで接続を指定するのに使用) を指定します。通常はパスワードも指定します。また、プロキシホストや委譲先メールボックス ID などのその他の属性も指定できます。詳細については、『CFML リファレンス』の cfexchangeconnection を参照してください。

永続的な接続では、keepAlive プロパティを true に設定した、HTTP または HTTPS プロトコルを使用します。したがって、HTTP リクエストや ColdFusion ページが終了しても、この接続が自動的に閉じられることはありません。使用した後は、接続を閉じる必要があります。閉じなかった場合は、非アクティブ状態のタイムアウト時間が経過するまで接続が保持され、その後、ColdFusion に接続リソースが返されます。

注意：接続を Application スコープなどの永続スコープに保存して、複数のページで再利用することができます。しかし、これを行う利点はありません。接続は軽量なので、永続スコープを使用しても、大幅にパフォーマンスが向上することはありません。

次の例では、接続を開いた後、spamsource.com から送信されたメールをすべて取得して、Exchange サーバーからそれらのメッセージを削除します。

```
<cfexchangeConnection
    action = "open"
    username = "#user1#"
    password = "#password1#"
    server = "#exchangeServerIP#"
    connection = "conn1">

<cfexchangemail action = "get" name = "spamMail" connection = "conn1">
    <cfexchangefilter name = "fromID" value = "spamsource.com">
</cfexchangemail>

<cfloop query="spamMail">
    <cfexchangemail action = "delete" connection = "conn1"
            uid = "#spamMail.uid#">
</cfloop>

<cfexchangeConnection
    action = "close"
    connection = "conn1">
```

一時的な接続の使用

一時的な接続は、その接続を使用しているタグが処理を完了するまで保持されます。一時的な接続を作成するには、アクションタグ (cfexchangetask など) で接続を直接指定します。その際には、cfexchangeconnection タグで使用したのと同じ属性 (接続名は除く) を使用します。

次の例では、一時的な接続を使用して 1 つのタスクを作成しています。

```
<!--- Create a structure with the task fields. --->
<cfscript>
    stask = StructNew();
    stask.Priority = "high";
    stask.Status = "Not_Started";
    stask.DueDate = "3:00 PM 09/14/2007";
    stask.Subject = "My New Task";
    stask.PercentCompleted = 0;
    Message = "Do this NOW!";
</cfscript>

<!--- Create the task by using a transient connection. --->
<cfexchangetask action = "create"
    username = "#user1#"
    password = "#password1#"
    server = "#exchangeServerIP#"
    task = "#stask#"
    result = "theUID">

<!--- Display the UID to confirm that the action completed. --->
<cfdump var = "#theUID#">
```

委譲されたアカウントのアクセス

Exchange では、あるユーザーが別のユーザーに、そのアカウントに対するアクセス権を許可 (委譲) することができます。ユーザーは、カレンダー、連絡先、受信トレイ、タスクリストの任意の組み合わせに対して、参照者 (読み取り専用)、作成者 (読み取りと書き込み)、または編集者 (読み取りと書き込みと削除) のアクセス権を委譲することができます。

注意：ColdFusion を使用してアクセス権を委譲することはできません。

委譲されたユーザーとして委譲者のアカウントにアクセスするには、次の情報を指定します。

- username と password 属性に、委譲されたユーザーのユーザー名とパスワードを指定します。
- mailboxName 属性に、アクセスするアカウントのメールボックス名を指定します。

このアカウントへのアクセスは、永続的な接続を開く cfexchangeconnection タグでも、一時的な接続を使用する ColdFusion Exchange タグでも行えます。

たとえば、docuser3 のアカウントへのアクセス権が docuser4 に委譲されている場合は、cfexchangeconnection タグを使用して、次の例に示すように、docuser4 の資格情報を使用して docuser3 のアカウントに接続できます。

```
<cfexchangeconnection action="open"
    connection="theConnection"
    server="myexchangeserver.mycompany.com"
    username="docuser4"
    password="secret"
    mailboxName="docuser3">
```

この接続を使用して、docuser3 が docuser4 に委譲したあらゆるアクティビティを行うことができます。たとえば、docuser3 がカレンダーに対する参照者アクセス権のみを委譲している場合は、get または getAttachments 属性を指定した cfexchangecalendar タグのみをこの接続で使用できます。

Exchange アイテムの作成

Exchange イベント、連絡先、またはタスクを作成するには、それぞれ cfexchangecalendar、cfexchangecontact、または cfexchangetask タグを使用し、action 属性の値として create を指定します。メールメッセージを作成するには、cfmail タグを使用してメッセージを送信します。メールの送信の詳細については、1179 ページの「電子メールの送受信」を参照してください。

カレンダーイベント、連絡先、またはタスクを作成する場合は、アクション、接続情報 (永続的な接続の名前か、一時的な接続の属性)、追加する情報が含まれている構造体を指定します。また、作成するエントリの Exchange UID の値を格納する result 変数も指定できます。エントリを変更または削除するタグでエントリを識別するには、この UID を使用します。

エントリ情報の指定に使用する属性の名前は、次のように、使用するタグによって異なります。

タグ	属性
cfexchangecalendar	event
cfexchangecontact	contact
cfexchangetask	task

イベント、連絡先、またはタスクデータの詳細が含まれている変数は、次の例のように、シャープ記号 (#) で囲みます。

```
<cfexchangecalendar action="create" connection="myConn" event="#theEvent#"
    result="resultUID">
```

エントリ情報の構造体の内容は、タグによって異なります。構造体の内容の詳細については、『CFML リファレンス』の cfexchangecalendar、cfexchangecontact、および cfexchangetask を参照してください。

注意：Exchange カレンダーの予定を作成するには、カレンダーイベントを作成し、必須またはオプションのいずれの参加者も指定しません。

この例では、ユーザーがフォームに情報を入力し、その情報を使用して Exchange サーバーに連絡先を作成します。

```
<!--- Create a structure to hold the contact information. --->
<cfset sContact="#StructNew()#">

<!--- A self-submitting form for the contact information --->
<cfform format="flash" width="550" height="460">
    <cfformitem type="html"><b>Name</b></cfformitem>
    <cfformgroup type="horizontal" label="">
        <cfinput type="text" label="First" name="firstName" width="200">
        <cfinput type="text" label="Last" name="lastName" width="200">
    </cfformgroup>
    <cfformgroup type="VBox">
        <cfformitem type="html"><b>Address</b></cfformitem>
        <cfinput type="text" label="Company" name="Company" width="435">
        <cfinput type="text" label="Street" name="street" width="435">
        <cfinput type="text" label="City" name="city" width="200">
        <cfselect name="state" label="State" width="100">
            <option value="CA">CA</option>
            <option value="MA">MA</option>
            <option value="WA">WA</option>
        </cfselect>
        <cfinput type="text" label="Country" name="Country" width="200"
                Value="U.S.A.">
        <cfformitem type="html"><b>Phone</b></cfformitem>
        <cfinput type="text" validate="telephone" label="Business"
                name="businessPhone" width="200">
        <cfinput type="text" validate="telephone" label="Mobile"
                name="cellPhone" width="200">
        <cfinput type="text" validate="telephone" label="Fax" name="fax"
                width="200">
        <cfformitem type="html"><b>Email</b></cfformitem>
        <cfinput type="text" validate="email" name="email" width="200">
    </cfformgroup>

    <cfinput type="Submit" name="submit" value="Submit" >
</cfform>
```

```
<!--- If the form was submitted, fill the contact structure from it. --->
<cfif isDefined("Form.Submit")>
    <cfscript>
        sContact.FirstName=Form.firstName;
        sContact.Company=Form.company;
        sContact.LastName=Form.lastName;
        sContact.BusinessAddress.Street=Form.street;
        sContact.BusinessAddress.City=Form.city;
        sContact.BusinessAddress.State=Form.state;
        sContact.BusinessAddress.Country=Form.country;
        sContact.BusinessPhoneNumber=Form.businessPhone;
        sContact.MobilePhoneNumber=Form.cellPhone;
        sContact.BusinessFax=Form.fax;
        sContact.Email1=Form.email;
    </cfscript>

    <!--- Create the contact in Exchange --->
    <cfexchangecontact action="create"
        username ="#user1#"
        password="#password1#"
        server="#exchangeServerIP#"
        contact="#sContact#"
        result="theUID">

    <!--- Display a confirmation that the contact was added. --->
    <cfif isDefined("theUID")>
        <cfoutput>Contact Added. UID is#theUID#</cfoutput>
    </cfif>
</cfif>
```

アイテム作成の別の例 (タスクの作成) については、1197 ページの「一時的な接続の使用」を参照してください。

Exchange アイテムおよび添付ファイルの取得

Exchange サーバーから、カレンダーイベント、連絡先、メールメッセージ、およびタスクを取得することができます。それらのアイテムの添付ファイルも取得することができます。

Exchange のアイテムとその添付ファイルを取得するには、複数の操作が必要な場合があります。

- 受信トレイに直接入っていないメールを取得するには、メールボックスのルートからメールフォルダへのパスを指定します。1 回の操作でアイテムを取得できるメールフォルダは 1 つのみです。cfexchangeconnection タグを使用すると、メールボックス内のすべてのフォルダの名前、パス、およびサイズを取得できます。その結果を使用して、各フォルダを巡回できます。
- アイテムの添付ファイルを取得するには、最初にアイテムを取得してから、そのアイテムの UID を使用してその添付ファイルを取得します。
- Exchange のアイテムにインラインイメージのあるメッセージが含まれる場合は、イメージは添付ファイルとして使用できます。添付ファイルを取得し、添付ファイルの CID を使用してメッセージ内でのイメージの位置を探し、イメージをインラインで表示できます。

  注意：getattachment アクションでは、イメージなどのインライン添付ファイルを含む HTML メールの CID フィールドが設定されない場合があります。この問題は、添付ファイルがインラインで送信された場合に、一部の Exchange クライアントで CID 値が設定されないことがあるために発生します。

フォルダ名の取得と使用

メールボックス内のフォルダ名を取得したり、特定のフォルダのサブフォルダの名前を取得するには、cfexchangeconnection タグの getSubfolders アクションを使用します。これによって、サブフォルダごとに 1 つの行が含まれたクエリーが返されます。クエリーには次の 3 つの列があります。

- フォルダ名
- メールボックスからフォルダまでのフルパス。受信トレイを含みます。
- フォルダサイズ (バイト単位)

タグでは、サブフォルダを取得するフォルダや、サブフォルダのすべてのレベルを再帰的に取得するかどうかを指定できます。

getSubfolders アクションで取得したフォルダパスを、cfexchangemail タグの folder 属性で使用して、アクションの対象となるメールメッセージが含まれているフォルダを指定できます。cfexchangemail タグでフォルダを指定しなかった場合は、受信トレイの最上位レベルのみが検索されて、アクションが実行されます。

複数のフォルダ内のメールに対して操作 (メールアイテムや添付ファイルの取得を含む) を行うには、次の例のように、getSubfolders アクションで返されたクエリー内の各エントリをループします。この例では、受信トレイとそのすべてのサブフォルダにある会議の辞退メッセージをすべてレポートします。

```
<!--- Create a connection. --->
<cfexchangeConnection
    action="open"
    username ="#user2#"
    password="#password2#"
    server="#exchangeServerIP#"
    connection="conn1">

<!--- Get the names and paths to all subfolders. --->
<cfexchangeconnection action="getSubfolders" connection="conn1"
    name="folderInfo" folder="Inbox" recurse="yes">

<!--- Get the information from the Inbox top level.
            The getSubfolders results do not include an Inbox row. --->
    <cfexchangemail action="get" connection="conn1"
            name="theResponses">
        <cfexchangefilter name="MessageType" value="Meeting_Response">
    </cfexchangemail>

<!--- Use a query of queries to select only the declined meetings. --->
<!--- You cannot use cfexchangefilter to filter for the meeting response type. --->
    <cfquery dbtype="query" name="theResponses">
        SELECT * FROM theResponses
        WHERE MEETINGRESPONSE = 'Decline'
    </cfquery>

<!--- Loop through the subfolders and get the meeting responses from each
    folder. --->
<cfloop query="folderInfo">
    <cfexchangemail action="get" connection="conn1"
            name="#folderinfo.foldername#">
        <cfexchangefilter name="folder" value="#folderinfo.folderpath#">
        <cfexchangefilter name="MessageType" value="Meeting_Response">
    </cfexchangemail>

    <!--- Use the evaluate function to get the name of the folder. --->
    <cfset meetingData=evaluate(folderinfo.foldername)>
    <!--- Use a query of queries with a UNION clause to add this folder's
            results to the theResponses query. --->
```

```
    <cfquery dbtype="query" name="theResponses">
        SELECT * FROM meetingData
        WHERE MEETINGRESPONSE = 'Decline'
        UNION
        SELECT * FROM theResponses
    </cfquery>
</cfloop>

<!--- Close the connection. --->
<cfexchangeConnection
    action="close"
    connection="conn1">

<!--- Display the results. --->
<h3>The Declined Responses:</h3>
<cftable query="theResponses" colheaders="yes" border="yes">
    <cfcol header="From" text="#FROMID#">
    <cfcol header="Subject" text="#SUBJECT#">
    <cfcol header="Message" text="#MESSAGE#">
</cftable>
```

アイテムの取得

Exchange サーバーからイベント、連絡先、メールメッセージ、またはタスクを取得するには、それぞれ cfexchangecalendar、cfexchangecontact、cfexchangemail、または cfexchangetask タグを使用し、action 属性の値として get を指定します。これによって、name 属性で指定した名前のクエリーオブジェクトに、目的のアイテムが返されます。取得するアイテムを指定するには、cfexchange フィルタ子タグで選択条件を指定します。Exchange サーバーからアイテムを取得するコードは、次のようなパターンになります。

```
<cfexchange***
        action="get"
        name="results query object name"
        connection information>
    <cfexchangefilter
            name="filter type"
            value"filter value>
    <cfexchangefilter
            name="data/time filter type"
            from="start date/time"
            to="end date/time">
    .
    .
    .
</cfexchange>
```

アイテムの取得時には、次のルールが適用されます。

- cfexchangefilter タグは 0 個以上使用できます。
    - name 属性で指定した構造体で maxrows フィールドを指定していない場合は、最大 100 アイテムまで取得されます。より多くのアイテムを取得するには、maxrows フィールドに 100 よりも大きい値を指定します。
    - 異なる name 属性を持つ cfexchangefilter タグを複数指定した場合は、指定したすべての条件に一致するアイテムがすべて取得されます。
    - 同一の name 属性を持つ cfexchangefilter タグを複数指定した場合は、同一の name 属性を持つ最後のタグのみが使用され、それに一致するアイテムが取得されます。
- name 属性は、Exchange のアイテムレコード内のフィールド名に対応します。name 属性に有効な値は、取得するアイテムのタイプによって異なります。有効な値の詳細なリストについては、『CFML リファレンス』の対応するタグリファレンスを参照してください。

- name 属性にテキストまたは数値情報を取るフィールドを指定した場合は、value 属性を使用して条件を指定できます。
- name 属性に日付、時刻、または日付時刻を取るフィールドを指定した場合は、from および to 属性を使用して範囲を指定できます。これらの属性の 1 つを省略すれば、たとえば 2007 年 12 月 1 日までのすべての日のように、開始日または終了日に制限がない範囲を指定できます。
- 日付の範囲には、指定した日時も含まれます。to または from で指定した日付を持つアイテムがある場合は、そのアイテムも選択されます。
- value 属性で空の文字列を指定して、空の値を検索することはできません。特定のフィールドの値が空であるエントリを検索するには、すべてのエントリを取得し、クエリーオブクエリーを使用して、そのフィールドが空であるエントリのみが選択されるように結果をフィルタします。
- メッセージや件名のようなテキスト文字列を取るフィールドでは、value 属性で指定したフレーズに正確に一致する文字列が含まれているアイテムが返されます。
- cfexchangemail タグでは、1 つのフォルダ内のアイテムのみが取得されます。フォルダに対するフィルタを含めた場合は、受信トレイに直接入っているアイテムのみが取得され、サブフォルダは検索されません。複数のフォルダから情報を取得する例については、1201 ページの「フォルダ名の取得と使用」を参照してください。

結果が取得されると、name 属性で指定したクエリーオブジェクトが存在していない場合は作成され、各行に 1 つのアイテム (メールメッセージなど) が設定されます。クエリー列はアイテムのタイプによって異なります。たとえば、メールメッセージの場合は FromID および ToID フィールドがあり、連絡先の場合は FirstName および LastName フィールドがあります。返される構造体の詳細については、『CFML リファレンス』で対応するタグを参照してください。

次の 2 つの列は、どのタイプのアイテムをクエリーしても結果に含まれます。

- UID 列。アイテムの一意の ID を持ちます。エントリを削除、変更、または応答 (カレンダーエントリの場合) する場合は、この値を使用して対象のアイテムを指定します。また、UID の値を使用して、アイテムの添付ファイルを取得することもできます。
- HasAttachments 列。アイテムに添付ファイルがあるかどうかを示すブール値を持ちます。このフィールドが true の場合は、getAttachments アクションを使用して添付ファイルを取得できます。

次の例では、adobe.com を含む電子メールアドレスから docuser1 ユーザーに先週送信されたメールメッセージを取得します。コードを短くするために、この例では、cfdump タグを使用して結果を表示しています。

```
<cfset rightNow = Now()>
<cfset lastWeek = DateAdd("d","-7", rightNow)>

<cfexchangemail action="get" name="weeksMail"
        username ="#user1#" password="#password1#"
        server="#exchangeServerIP#">
    <cfexchangefilter name="FromID" value="adobe.com">
    <cfexchangefilter name="TimeSent" from="#lastWeek#" to="#rightNow#">
</cfexchangemail>

<cfdump var="#weeksMail#">
```

アイテムの添付ファイルの取得

Exchange の連絡先、イベント、メッセージ、タスクへの添付ファイルを取得するには、ColdFusion Exchange タグで getAttachments アクションを使用します。また、次の情報も指定します。

- 添付ファイルを含むメッセージの UID。
- 取得した添付ファイルに関する情報を格納するクエリーの名前。タグの処理が完了すると、取得した添付ファイルごとに 1 つのレコードがクエリーオブジェクトに格納されます。このクエリーには、ファイル名、保存されている添付ファイルへの完全なパス、MIME タイプ、ファイルサイズ、CID 値 (または空の文字列)、および添付ファイルがメッセージかどうかを示すインジケータという 6 つの列があります。

- 添付ファイルが保存されている場所のパス。パスを省略すると、添付ファイルは取得されませんが、添付ファイルに関する情報は取得されます。
- オプションで、複数の添付ファイルが同じ名前の場合に、名前に番号を追加することで一意のファイル名を作成するかどうか。デフォルトは一意のファイル名を作成しません。

次の ColdFusion Exchange タグでは、theUID 変数で識別されるメッセージの添付ファイルをすべて取得し、それを "C:/temp/cf_files/attachments" ディレクトリに保存し、添付ファイルに関する情報を attachInfo 構造体に保存します。

```
<cfexchangemail action="getattachments"
    connection="myconn1"
    uid="#theUID#"
    name="#attachInfo#"
    attachmentPath="C:/temp/cf_files/attachments"
    generateUniqueFilenames="true">
```

メッセージの添付ファイルを取得するには、メッセージの UID を取得し、そのメッセージに添付ファイルがあることを知る必要があります。この情報を確認するには、cfexchangemail などの ColdFusion Exchange タグで get アクションを使用します。タグの処理が完了すると、name 属性で指定されたクエリーに、次の列が格納されます。

- HasAttachments フィールド。メッセージに添付ファイルがある場合に true になります。
- UID フィールド。アイテムの Exchange UID が含まれます。正確な UID の形式は、アイテムのタイプ (イベント、連絡先、メッセージ、タスク) によって異なります。

これらのフィールドを使用して、メッセージの添付ファイルを取得するかどうかを決定し、メッセージの UID を確認することができます。

次の例では、docuser2 から先週送信されたすべてのメールメッセージの添付ファイルを取得します。各メッセージの添付ファイルは、メッセージの UID の 16 進部分と同じ名前のディレクトリに保存します。添付ファイルのある各メッセージについて、メッセージの件名と日付を表示し、メッセージの添付ファイルを一覧表に示します。この表には、添付ファイルの名前、MIME タイプ、サイズが含まれます。

メッセージに同じ名前の添付ファイルが複数ある場合、generateUniqueFilenames="true" が指定されていても、添付ファイル情報のクエリーには、常に元の同じ名前で添付ファイルがリストされています。generateUniqueFilenames 属性は、ディスク上のファイル名にのみ影響します。ただし、添付ファイル情報構造体の attachmentFilePath 列には、固有のファイル名が格納されます。

```
<cfset rightNow = Now()>
<cfset lastWeek = DateAdd("d","-7", rightNow)>

<cfexchangeconnection
    action="open"
    username ="#user1#"
    password="#password1#"
    server="#exchangeServerIP#"
    connection="conn1">

<cfexchangemail action="get" folder="Inbox/MailTest" name="weeksMail"
        connection="conn1">
    <cfexchangefilter name="FromID" value="docuser2">
    <cfexchangefilter name="TimeSent" from="#lastWeek#" to="#rightNow#">
</cfexchangemail>

<cfloop query="weeksMail">
    <cfif weeksmail.HasAttachment>
        <!--- The UID is surrounded in <> characters and has an @ character.
        Extract the hexadecimal number part for use as a directory name. --->
        <cfset atpos=Find('@', weeksMail.UID)>
        <cfset shortUID=Mid(weeksMail.UID, 2, atpos-2)>

        <cfexchangemail action="getAttachments"
```

```
            connection="conn1"
            folder="Inbox/MailTest"
            uid="#weeksMail.uid#"
            name="attachData"
            attachmentPath="C:/temp/cf_files/attachments/#shortUID#"
            generateUniqueFilenames="true">

        <cfoutput>
            Directory #shortUID# contains these attachments to the
            following message:<br />
            Subject: #weeksMail.Subject#<br />
            Sent: #dateFormat(weeksmail.TimeSent)#<br />
            <cftable query="attachData" colheaders="true">
                <cfcol header="Filename" text="#attachmentFilename#">
                <cfcol header="Size" text="#size#">
                <cfcol header="MIME type" text="#mimeType#">
            </cftable>
        </cfoutput>

    </cfif>
</cfloop>

<cfexchangeconnection
    action="close"
    connection="conn1">
```

インラインでのイメージの表示

HTML メッセージにインラインイメージが含まれる場合、Exchange サーバーはイメージを添付ファイルとして保存します。取得されたメッセージでイメージを表示するには、次の手順を実行します。

1. cfexchangemail タグの get アクションを使用して、メールメッセージを取得します。
2. cfexchangemail タグの getattachments アクションを使用して、メッセージの添付ファイルを取得します。前の手順で取得したメールメッセージの UID を指定します。また、Web ルートの下にある attachmentPath 属性値を指定して、保存されているファイルに URL を使用してアクセスできるようにします。
3. 手順 1 で取得した HTMLMessage フィールドのテキストを検索し、イメージアイテムを見つけます。各イメージの CID (コンテンツ ID) を取得します。
4. 手順 1 で取得した添付ファイルクエリーを検索します。CID 列の値が手順 3 で取得したものである各行について、対応する attachmentFilePath 列の値を取得します。
5. すべての img タグの src 属性の値を、CID 値に対応する attachmentFilePath フィールドの値に置き換えます。
6. 結果の HTML を表示します。

次の例では、添付ファイルからイメージを取得することで、インラインイメージを含むメッセージを表示する方法を示します。

```
<!--- Open the connection to the Exchange server. --->
<cfexchangeconnection
    action="open"
    username = "#user1#"
    password = "#password1#"
    server = "#exchangeServerIP#"
    connection = "testconn">

<!--- Get the mail message. --->
<cfexchangeMail action="get" connection ="testconn" name="getMail">
    <cfexchangeFilter name="Subject" value="sample inline image">
</cfexchangeMail>

<cfdump var="#getMail#">

<!--- The following code assumes we found only one matching message. --->
<cfoutput query="getMail">
    <cfset theUID = #getMail.UID#>
    <cfset htmlmessage = getMail.htmlmessage>
</cfoutput>

<!--- Get the message attachments. --->
<CFExchangeMail action="getAttachments" UID ="#theUID#" connection="testconn" name="attachments"
attachmentPath="C:\ColdFusion8\wwwroot\My_Stuff\cfexchange\Book\attachments"
generateuniquefilenames="no">

<!--- Extract the image names from the mail message --->
<!--- Initialize the index into the message used in finding --->
<cfset findstart = 1>
<!--- Use an index loop to find all image source entries in the message --->
<!--- This example supports up to 25 inline images --->
<cfloop index="i" from="1" to="25">
    <!--- find a cid: entry --->
    <cfset stringStart[i] = Find('"cid:', htmlmessage, findstart)>

    <!--- Exit the loop if no match was found --->
    <cfif (stringstart[i] EQ 0)>
        <cfbreak>
    </cfif>
    <!--- Increment the string index used in finding images. --->
    <cfset findstart = stringstart[i] +5 >
    <!--- Get text to the right of âBADCHAR˜cid:.âBADCHAR™
        Using a string length of 30 should get more than the image name. --->
    <cfset rightpart[i]=Mid(htmlmessage, findstart, 30)>
```

```
    <!--- use the ListFirst function to remove all the text starting
        at the quotation mark. --->
    <cfset imagename[i]=ListFirst(rightpart[i], '"')>

    <!--- Loop over the attachments query and find the CID. --->
    <cfloop query="attachments">
        <!--- Replace the image name with the contents of the attachment --->
        <cfif attachments.CID EQ imagename[i]>
            <cfset htmlmessage = Replace(htmlmessage,"cid:#imagename[i]#",
            "attachments/#attachments.ATTACHMENTFILENAME#")>
        </cfif>
    </cfloop>
</cfloop>

<h3>The full mail message</h3>
<cfoutput>#htmlmessage#</cfoutput>

<cfexchangeconnection
    action="close"
    connection = "testconn">
```

Exchange アイテムの変更

ColdFusion で設定できるカレンダー、連絡先、タスクアイテムのあらゆる要素を変更することができます。メールメッセージについては、Importance、Sensitivity、IsRead の各フラグを変更し、メールメッセージをフォルダ間で移動できます。

注意：アイテムに添付ファイルがあり、アイテムの変更時に添付ファイルを指定した場合は、前の添付ファイルに新しい添付ファイルが追加されます。前の添付ファイルが置換されることはありません。廃止された添付ファイルや変更された添付ファイルを削除するには、deleteAttachments アクションを使用します。

カレンダー、連絡先、およびタスクアイテムの変更

カレンダー、連絡先、またはタスクアイテムを変更するには、cfexchangecalendar、cfexchangecontact、または cfexchangetask タグを使用し、action 属性の値として modify を指定します。contact、event、または task 属性で、変更するアイテムプロパティとその新しい値を含んだ構造体を指定します。変更しないプロパティの値は指定しなくてかまいません。たとえば、カレンダータスクの終了時間を変更するには、EndTime フィールドのみを持つ構造体を event 属性で指定します。

次の例では、カレンダーイベントを作成してから変更します。最初にフォームを送信すると、カレンダーイベントが作成され、入力したデータとともにフォームが再表示されます。フォームを変更して再送信するには、イベントを承認します。2 回目にフォームを送信するときには、変更された情報が送信されます。イベントの承認の詳細については、1211 ページの「会議および予定の操作」を参照してください。

次の例では、(例を短くするために) すべてのイベントデータを再送信していますが、変更されたデータだけが送信されるように例を変更することができます。この例では、コードを簡単にするための繰り返し情報も省略しています。

```
<!--- Initialize the form.eventID to 0, to indicate a new event. --->
<!--- The EventID field is a hidden field managed by this application. --->
<cfparam name="form.eventID" default="0">

<!--- If the form was submitted, populate an event structure from it. --->
<cfif isDefined("Form.Submit")>
    <cfscript>
        sEvent=StructNew();
        sEvent.AllDayEvent="false";
        sEvent.Subject=Form.subject;
        if (IsDefined("Form.allDay")) {
            sEvent.AllDayEvent="true";
            sEvent.StartTime=createDateTime(Year(Form.date), Month(Form.date),
                Day(Form.date), 8, 0, 0);
        }
        else {
            sEvent.StartTime=createDateTime(Year(Form.date), Month(Form.date),
                Day(Form.date), Hour(Form.startTime), Minute(Form.startTime), 0);
            sEvent.EndTime=createDateTime(Year(Form.date), Month(Form.date),
                Day(Form.date), Hour(Form.endTime), Minute(Form.endTime), 0);
        }
        sEvent.Location=Form.location;
        sEvent.RequiredAttendees=Form.requiredAttendees;
        sEvent.OptionalAttendees=Form.optionalAttendees;
        //sEvent.Resources=Form.resources;
        if (Form.reminder NEQ "") {
            sEvent.Reminder=Form.reminder;
        }
        else {
            sEvent.Reminder=0;
        }
        sEvent.Importance=Form.importance;
        sEvent.Sensitivity=Form.sensitivity;
        sEvent.message=Form.Message;
    </cfscript>

    <!--- If the form is being submitted for the first time,
            create an event. --->
    <cfif form.eventID EQ 0>
    <!--- Create the event in Exchange --->
        <cfexchangecalendar action="create"
            username ="#user1#"
            password="#password1#"
            server="#exchangeServerIP#"
            event="#sEvent#"
            result="theUID">
        <!--- Display the new event UID and set form.eventID to it. --->
        <cfif isDefined("theUID")>
            <cfoutput>Event Added. UID is #theUID#</cfoutput>
            <cfset Form.eventID = theUID >
        </cfif>

    <cfelse>
    <!--- The form is being resubmitted with new data; update the event. --->
        <cfexchangecalendar action="modify"
            username ="#user1#"
            password="#password1#"
            server="#exchangeServerIP#"
            event="#sEvent#"
            uid="#Form.eventID#">
        <cfoutput>Event ID #Form.eventID# Updated.</cfoutput>
```

```
    </cfif>
</cfif>

<!--- A self-submitting form for the event information --->
<cfform format="xml" preservedata="true" style="width:500" height="600">
    <cfinput type="text" label="Subject" name="subject" style="width:435">
    <br />
    <cfinput type="checkbox" label="All Day Event" name="allDay">
    <cfinput type="datefield" label="Date" name="date" validate="date"
        style="width:100">
    <cfinput type="text" label="Start Time" name="startTime" validate="time"
        style="width:100">
    <cfinput type="text" label="End Time" name="endTime" validate="time"
        style="width:100"><br />
    <cfinput type="text" label="Location" name="location"
        style="width:435"><br />
    <cfinput type="text" label="Required Attendees" name="requiredAttendees"
        style="width:435"><br />
    <cfinput type="text" label="Optional Attendees" name="optionalAttendees"
        style="width:435"><br />
    <cfinput type="text" label="Resources" name="resources"
        style="width:435"><br />
    <cfinput type="text" label="Reminder (minutes)" validate="integer"
        name="reminder" style="width:200">
    <cfselect name="importance" label="Importance" style="width:100">
        <option value="normal">Normal</option>
        <option value="high">High</option>
        <option value="low">Low</option>
    </cfselect>
    <cfselect name="sensitivity" label="Sensitivity" style="width:100">
        <option value="normal">Normal</option>
        <option value="company-confidential">Confidential</option>
        <option value="personal">Personal</option>
        <option value="private">Private</option>
    </cfselect>
    <cfinput type="textarea" label="Message" name="message" style="width:435;
        height:100">
    <cfinput type="hidden" name="eventID" value="#Form.EventID#">
    <cfinput type="Submit" name="submit" value="Submit" >
</cfform>
```

メール属性の設定

メールメッセージの Importance、Sensitivity、IsRead フラグを設定するには、cfexchangemail タグを使用し、action 属性の値として set を指定します。変更するフラグのみを message 属性で指定します。

次のスニペット例では、受信トレイに直接入っている 2 週間以上経過しているメールメッセージに対して、IsRead フラグを true に変更することで、キャッチアップ操作を行います。この例では、受信トレイのフォルダ内のメッセージのフラグは変更しません。これを行うには、各フォルダに対し、個別に cfexchangemail タグを使用します。複数のフォルダへのアクセスと使用の詳細については、1201 ページの「フォルダ名の取得と使用」を参照してください。

```
<!--- Create a structure with a true IsRead field --->
<cfset changeValues.IsRead="true">

<!--- Open the connection. --->
<cfexchangeConnection
    action="open"
    username ="#user1#"
    password="#password1#"
    server="#exchangeServerIP#"
    connection="conn1">

<!--- Get the mail in the Inbox that is at least two weeks old. --->
<cfexchangemail action="get" name="oldMail" connection="conn1">
    <cfexchangefilter name="timeSent" from="01/01/2000"
        to="#DateAdd("d","-14", Now())#">
</cfexchangemail>

<!--- Loop through the resulting oldMail query and set the IsRead indicator
            to true. --->
<cfloop query="oldMail">
    <cfexchangemail action="set"
        connection="conn1"
        message="#changeValues#"
        uid="#oldMail.uid#">
</cfloop>

<!--- Close the connection. --->
<cfexchangeConnection
    action="close"
    connection="conn1">
```

フォルダ間のメールの移動

メールメッセージをフォルダ間で移動するには、次のコード例で示すように、cfexchangemail タグの move アクションを使用します。この例では、件名が "Rams and Ewes" であるすべてのメッセージを、受信トレイの "Unread" フォルダから "Sheep" フォルダに移動します。

```
<cfexchangemail action="move" connection="con1" folder="Inbox/Unread"
        destinationfolder="Inbox/Sheep">
    <cfexchangefilter name="subject" value="Rams and Ewes">
</cfexchangemail>
```

Exchange アイテムおよび添付ファイルの削除

Exchange アイテムを削除するには、ColdFusion Exchange タグで action 属性に delete を指定し、アイテムの UID を指定します。Exchange アイテムを削除すると、添付ファイルもすべて削除されます。

Exchange アイテムの添付ファイルだけを削除するには、ColdFusion Exchange タグで action 属性に deleteAttachments を指定し、アイテムの UID を指定します。

この例では、終了済みの会議の会議出席依頼を受信トレイからすべて削除します。ただし、受信トレイのフォルダ内のリクエストは削除しません。受信トレイのフォルダ内の出席依頼を削除するには、各フォルダに対し、個別に cfexchangemail タグを使用します。複数のフォルダへのアクセスと使用の詳細については、1201 ページの「フォルダ名の取得と使用」を参照してください。

```
<cfexchangeconnection
    action="open"
    username ="#user#"
    password="#password#"
    server="#exchangeServerIP#"
    connection="conn1">

<!--- Get all meeting notifications from the Inbox. --->
<cfexchangemail action="get" name="requests" connection="conn1">
    <cfexchangefilter name="MessageType" value="Meeting">
</cfexchangemail>

<!--- Get the meeting request data and put it in an array. --->
<cfset i=1>
<cfset meetingData=ArrayNew(1)>
<cfloop query="requests">
    <cfexchangemail action="getmeetinginfo" connection="conn1"
        name="meeting" meetinguid="#MeetingUID#" mailUID="#UID#">
    <cfset meetingData[i]=meeting>
    <cfset i=i+1>
</cfloop>

<!--- Loop through the request data array and delete any outdated
meeting messages from the Inbox. --->
<cfloop index="i" from="1" to="#ArrayLen(meetingData)#" >
    <cfif meetingData[i].StartTime LTE now()>
        <cfexchangemail action="delete" connection="conn1"
            UID="#meetingData[i].UID#">
    </cfif>
</cfloop>

<cfexchangeconnection
    action="close"
    connection="conn1">
```

既知の迷惑メールアドレスからのメールをすべて削除する例については、1197 ページの「永続的な接続の使用」を参照してください。

会議および予定の操作

ここでは、カレンダーイベントと、受信トレイで受信する会議通知に関する次の方法について説明します。

- 会議出席依頼、取り消し通知、出席依頼への応答に関する詳細情報を取得する方法
- 定期的なイベントを指定する方法

会議通知および出席依頼の操作

次の場合、自分のメールボックスに会議通知が届きます。

- 他のユーザーが自分に会議出席依頼を送信した場合
- 自分のカレンダーに含まれている会議を、他のユーザーが取り消した場合
- 通知を有効にして送信した会議出席依頼に、他のユーザーが応答した場合

cfexchangemail タグの get アクションで取得できる情報には、会議に関する詳細な情報は含まれていません。会議関連の情報として含まれているのは、次の項目のみです。

- イベント UID
- メッセージのタイプ (会議出席依頼、応答、取り消し)

- (メッセージが会議出席依頼への応答である場合) 会議への応答が承諾、辞退、仮承諾のいずれであるか

また、会議出席依頼は、承諾するまでカレンダーには表示されません。したがって、cfexchangecalendar タグを使用しても詳細情報は取得できません。

会議メッセージに関する詳細情報を取得するには、cfexchangemail タグの getMeetingInfo アクションを使用します。情報を取得したら、それに応じて適切な処理を行えます。たとえば、cfexchangecalendar タグの response アクションを実行して、会議出席依頼を承諾または辞退できます。

会議メッセージの詳細の取得と、会議出席依頼への応答

1. 会議通知を含むメールメッセージを取得します。それには、cfexchangemail タグを使用し、action 属性の値として get を指定します。また、cfexchangefilter 子タグを使用して、次の属性を指定します。
   - name 属性 : MessageType を値として指定します。
   - value 属性 : Meeting、Meeting_Request、Meeting_Response、Meeting_Cancel のいずれかを値として指定します。Meeting を指定した場合は、すべての会議通知が取得されます。

   cfexchangefilter タグを追加して、取得するメッセージをさらに限定することができます。

   cfexchangemail タグが処理されると、cfexchangemail タグの name 属性で指定した構造体の MeetingUID 列に、会議の UID が格納されます。
2. 各会議に関する情報を取得します。それには、cfexchangemail タグを使用して、次の属性を指定します。
   - action 属性 : getMeetingInfo を値として指定します。
   - meetingUID 属性 : cfexchangemail タグの name 属性で指定した構造体の MeetingUID 列の値を指定します。
   - (オプション) mailuid 属性 : 会議通知が含まれているメッセージの UID を指定します。1 つの会議に関するメッセージが受信トレイに複数ある場合は、この属性を使用してメッセージを特定します。
3. 手順 2 で取得した情報をアプリケーションロジックで処理して、必要なメッセージやアクションを決定します。たとえば、そのユーザーのすべての会議出席依頼をフォームに表示して、各メッセージに応答できるようにします。
4. 会議出席依頼に応答するには、cfexchangecalendar タグで action 属性に値 respond を指定し、次の属性を指定します。
   - uid 属性 : 手順 2 で取得した Meeting UID を指定します。Message UID は使用しないでください。
   - responseType 属性 : accept、decline、または tentative を値として指定します。
   - (オプション) notify 属性 : イベントの所有者が会議の応答メッセージを受け取るかどうかを、true (デフォルト) または false で指定します。
   - 所有者が通知を受け取る場合は、message 属性を使用して、応答内に含めるテキストメッセージを指定できます。

このプロセスを次の例に示します。この例では、受信トレイ内の会議出席依頼をすべて表示し、出席依頼ごとに応答メッセージを送信できるようにしています。

```
<cfexchangeconnection
    action="open"
    username ="#user2#"
    password="#password2#"
    server="#exchangeServerIP#"
    connection="conn1">

<cfif isDefined("Form.Submit")>

<!--- When the form has been submitted, send the responses. --->
    <cfloop index="k" from="1" to="#Form.responses#">
        <cfset resp = Form["response" & k] >
        <cfset msg = Form["respMessage" & k] >
        <cfset msguid = Form["UID" & k] >
        <cfexchangecalendar action="respond" connection="conn1"
            uid="#msguid#" responseType="#resp#" message="#msg#">
        <cfoutput><h4>Response #k# sent!</h4></cfoutput>
    </cfloop>

<cfelse>
    <!--- Get all messages with meeting Requests. --->
    <cfexchangemail action="get" name="requests" connection="conn1">
        <cfexchangefilter name="MessageType" value="Meeting_Request">
    </cfexchangemail>

    <!--- Get the meeting request data. --->
    <cfloop query="requests">
        <cfexchangemail action="getmeetinginfo" connection="conn1"
            name="meeting" meetinguid="#MeetingUID#">
        <cfset meetingData[requests.currentrow]=meeting>
    </cfloop>

    <!--- Display the invitation data in a form. --->
    <cfform name="bar">
        <cfloop index="j" from="1" to="#ArrayLen(meetingData)#">
            <cfoutput>
                <h3>Meeting Request #j#</h3>
                Subject: #meetingData[j].Subject# <br />
                Sensitivity: #meetingData[j].Sensitivity# <br />
                Organizer: #meetingData[j].Organizer# <br />
                All Day?: #meetingData[j].AllDayEvent# <br />
                Day: #DateFormat(meetingData[j].StartTime)#  
                Starts: #TimeFormat(meetingData[j].StartTime)#  
                Ends: #TimeFormat(meetingData[j].EndTime)# <br />
                Duration: #meetingData[j].Duration# <br />
                Location: #meetingData[j].Location# <br />
                Message: #meetingData[j].Message# <br />
            </cfoutput>

            <!--- Specify the response to this invitation. --->
            <h4>response:</h4>
            <cfinput type="radio" checked name="response#j#" value="accept">
```

```
                Accept
            <cfinput type="radio" name="response#j#" value="decline">Decline
            <cfinput type="radio" name="response#j#" value="tentative">Tentative
            <br />
            <cftextarea name="respMessage#j#" label="Message (optional)"
                width="300" height="200" />
            <cfinput type="hidden" name="UID#j#"
                value="#meetingData[j].MeetingUID#">
            <hr />
        </cfloop>
        <cfinput type="hidden" name="responses"
            value="#ArrayLen(meetingData)#">
        <cfinput type="Submit" name="submit" value="Submit">
    </cfform>

</cfif>

<cfexchangeconnection
    action="close"
    connection="conn1">
```

受信トレイとそのすべてのサブフォルダにあるすべての会議辞退メッセージに関する情報を取得する例については、1201 ページの「フォルダ名の取得と使用」を参照してください。

定期的なカレンダーイベントの設定

定期的に繰り返すイベントを作成するには、event 属性構造体の次のフィールドを設定します。

- IsRecurring フィールドを true に設定します。
- RecurrenceType フィールドに、DAILY、WEELY、MONTHLY、YEARLY のいずれかを値として指定します。
- (オプション) RecurrenceCount、RecurrenceEndDate、RecurrenceNoEndDate のいずれかのフィールドを設定します。

  注意：この 3 つのフィールドをすべて省略すると、終了日のないイベントが作成されます。回数または終了日を指定すると、RecurrenceNoEndDate の値は自動的に false に設定されます。したがって、RecurrenceNoEndDate フィールドを指定するのは、繰り返し回数または終了日が設定されている既存のイベントを、終了日のないイベントに変更する場合のみです。

- 繰り返しのタイプに応じて、追加のフィールドに繰り返しの詳細を指定します。

イベントの繰り返し設定 (イベントを繰り返すかどうかも含む) を変更するには、値を変更するフィールドのみを指定します。イベントの繰り返しを停止するには、IsRecurring フィールドを false に設定します。定期的でないイベントを定期的なイベントに変更するには、IsRecurring フィールドを true に設定し、必要な繰り返しフィールドをすべて設定します。

次のセクションでは、それぞれの繰り返しタイプの指定方法について説明します。使用するフィールドの詳細については、『CFML リファレンス』の cfexchangecalendar を参照してください。

注意：設定した開始日と競合する繰り返しルールを指定した場合、イベントが最初に発生するのは、開始日ではなく、開始日以降でルールに一致する最初の日になります。たとえば、第 2 火曜日にイベントをスケジュールし、開始日を 2007 年 6 月 2 日に指定した場合、最初にそのイベントが発生するのは、2007 年 6 月 12 日になります。

日単位の繰り返しの指定

日単位で繰り返しを設定するには、次のいずれかを行います。

- RecurrenceFrequency フィールドを定義して、イベントの頻度を日単位で指定します。たとえば、3 日ごとに会議をスケジュールするには、RecurrenceFrequency="3" と指定します。
- RecurEveryWeekDay="true" と指定すると、週に 5 日開催する会議が設定されます。

日単位の繰り返しでは、平日にのみ複数回繰り返すイベントはスケジュールできません。そのようなイベントをスケジュールするには、週単位の繰り返しを使用して、繰り返しの曜日を複数指定します。

次の CFScript コードサンプルでは、3 日ごとの繰り返しを設定し、イベントが 20 回発生するように設定しています。

```
IsRecurring="true";
RecurrenceType="DAILY";
RecurrenceCount="20";
RecurrenceFrequency="3";
```

週単位の繰り返しの指定

毎週特定の曜日に発生するイベントや、毎週または何週かごとに発生するイベントを作成するには、RecurrenceType="WEEKLY" を使用します。次のフィールドを使用して、頻度を制御します。

- RecurrenceFrequency フィールドを定義して、イベントの頻度を週単位で指定します。このフィールドを省略すると、イベントは毎週発生します。たとえば、4 週ごとに会議をスケジュールするには、RecurrenceFrequency="4" と指定します。
- RecurrenceDays フィールドで、文字列 MON、TUE、WED、THUR、FRI、SAT、SUN を指定します。複数指定する場合はカンマで区切ります。この属性を省略すると、startTime フィールドで指定した日付の曜日にイベントが繰り返されます。

次の CFScript コードサンプルでは、2007 年 12 月 3 日まで、火曜日と木曜日に隔週でイベントが発生するように設定しています。

```
IsRecurring="true";
RecurrenceType="WEEKLY";
RecurrenceEndDate="12/13/2007";
RecurrenceFrequency="2";
RecurrenceDays="TUE,THU;
```

月単位の繰り返しの指定

毎月または何か月かごとに発生するイベントを作成するには、RecurrenceType="MONTHLY" を使用します。次の 2 つのタイプのイベントをスケジュールできます。

- 月の特定の日に発生するイベント。たとえば、3 か月ごとの 10 日に発生するイベントなど。
- 月の特定の週の特定の曜日に発生するイベント。たとえば、毎月第 2 火曜日、または 6 か月ごとの最後の金曜日に発生するイベントなど。

毎月特定の日に発生するイベントを指定する場合は、繰り返しタイプを指定するだけで済みます。繰り返しが毎月ではない場合は、頻度も指定します。これによって、startTime フィールド値で指定した日にイベントが発生するようにスケジュールされます。4 か月ごとの開始日に会議が発生するようにスケジュールするには、次の繰り返しフィールドを指定します。

```
IsRecurring="true";
RecurrenceType="MONTHLY";
RecurrenceFrequency="4";
```

特定の曜日に発生するイベントを指定する場合は、RecurrenceType に加えて、次のフィールドも指定します。

フィールド	説明
RecurrenceFrequency	イベントの頻度を月単位で示します。このフィールドを省略すると、イベントは毎月発生します。
RecurrenceWeek	月の何週目にイベントを発生させるかを示します。有効な値は、first、second、third、fourth、last です。
RecurrenceDay	何曜日にイベントを発生させるかを示します。有効な値は、SUN、MON、TUE、WED、THU、FRI、SAT です。

次の CFScript コードサンプルでは、3 か月ごとの第 3 木曜日にイベントが発生するように設定しています。

```
IsRecurring="true";
RecurrenceType="Monthly";
RecurrenceFrequency="3";
RecurrenceWeek="third";
RecurrenceDay="THU";
```

年単位の繰り返しの指定

毎年または何年かごとに発生するイベントを作成するには、RecurrenceType="YEARLY" を使用します。次の 2 つのタイプのイベントをスケジュールできます。

- 年の特定の日に発生するイベント。たとえば、毎年 8 月 10 日に発生するイベントなど。
- 特定の月、週、および曜日に発生するイベント、たとえば、8 月の第 2 木曜日に発生するイベントなど。

毎年特定の日に発生するイベントを指定する場合は、繰り返しタイプを指定するだけで済みます。これによって、startTime フィールド値で指定した日に毎年イベントが発生するようにスケジュールされます。毎年、開始日に会議が発生するようにスケジュールするには、次の繰り返しフィールドを指定します。

```
IsRecurring="true";
RecurrenceType="YEARLY";
```

毎年特定の曜日に発生するイベントを指定する場合は、RecurrenceType に加えて、次のフィールドも指定します。

フィールド	説明
RecurrenceMonth	何月にイベントを発生させるかを示します。有効な値は、JAN、FEB、MAR、APR、MAY、JUN、JUL、AUG、SEP、OCT、NOV、DEC です。
RecurrenceWeek	月の何週目にイベントを発生させるかを示します。有効な値は、first、second、third、fourth、last です。
RecurrenceDay	何曜日にイベントを発生させるかを示します。有効な値は、SUN、MON、TUE、WED、THU、FRI、SAT です。

次の CFScript コードサンプルでは、8 月の第 3 木曜日にイベントが発生するように設定しています。

```
IsRecurring="true";
RecurrenceType="YEARLY";
RecurrenceMonth="AUG";
RecurrenceWeek="third";
RecurrenceDay="THU";
```

例 : 定期的なカレンダーイベントの設定

次の例では、すべての繰り返しタイプのイベントを作成します。コードを短くするため、イベントの作成に使用するフィールドの組み合わせが無効でないことは確認していません。フォームを送信してイベントが作成されると、フォームが再表示され、イベントの作成に使用されたフィールド値のダンプとイベント UID が表示されます。フォームを再送信してイベントを変更することはできませんが、フォーム内の値を変更して、イベントを作成することはできます。

```
<!--- Create a structure to hold the event information. --->
<cfparam name="form.eventID" default="0">

<!--- If the form was submitted, populate the event structure from it. --->
<cfif isDefined("Form.Submit")>
    <cfscript>
        sEvent.AllDayEvent="false";
        sEvent=StructNew();
        sEvent.Subject=Form.subject;
        if (IsDefined("Form.allDay")) {
            sEvent.AllDayEvent="true";
            sEvent.StartTime=createDateTime(Year(Form.date), Month(Form.date),
                Day(Form.date), 8, 0, 0);
        }
        else {
            sEvent.StartTime=createDateTime(Year(Form.date), Month(Form.date),
                Day(Form.date), Hour(Form.startTime), Minute(Form.startTime), 0);
            sEvent.EndTime=createDateTime(Year(Form.date), Month(Form.date),
                Day(Form.date), Hour(Form.endTime), Minute(Form.endTime), 0);
        }
        sEvent.Location=Form.location;
        sEvent.RequiredAttendees=Form.requiredAttendees;
        sEvent.OptionalAttendees=Form.optionalAttendees;
        //sEvent.Resources=Form.resources;
        if (Form.reminder NEQ "") {
            sEvent.Reminder=Form.reminder;
        }
        else {
            sEvent.Reminder=0;
        }
        sEvent.Importance=Form.importance;
        sEvent.Sensitivity=Form.sensitivity;
        //Recurrence Fields
        if (IsDefined("Form.isRecurring")) {
            sEvent.IsRecurring="true";}
        if (IsDefined("Form.recurrenceNoEndDate")) {
            sEvent.RecurrenceNoEndDate="true";}
        if (Form.recurrenceCount NEQ "") {
        sEvent.RecurrenceCount=Form.recurrenceCount;}
        if (Form.recurrenceEndDate NEQ "") {
        sEvent.RecurrenceEndDate=createDateTime(Year(Form.recurrenceEndDate),
            Month(Form.recurrenceEndDate), Day(Form.recurrenceEndDate), 0, 0,
            0);}
        sEvent.RecurrenceType=Form.recurrenceType;
        if (Form.recurrenceFrequency NEQ "") {
        sEvent.recurrenceFrequency=Form.recurrenceFrequency;}
        if (IsDefined("Form.recurEveryWeekDay")) {
            sEvent.RecurEveryWeekDay="true";}
        if (Form.recurrenceDays NEQ "") {
        sEvent.RecurrenceDays=Form.recurrenceDays;}
        if (Form.recurrenceDay NEQ "") {
        sEvent.RecurrenceDay=Form.recurrenceDay;}
        if (Form.recurrenceWeek NEQ "") {
        sEvent.RecurrenceWeek=Form.recurrenceWeek;}
        if (Form.recurrenceMonth NEQ "") {
        sEvent.RecurrenceMonth=Form.recurrenceMonth;}

        sEvent.message=Form.Message;
    </cfscript>

<cfdump var="#sEvent#">
```

```
<!--- Create the event in Exchange. --->
    <cfexchangecalendar action="create"
        username ="#user1#"
        password="#password1#"
        server="#exchangeServerIP#"
        event="#sEvent#"
        result="theUID">
    <!--- Output the UID of the new event --->
    <cfif isDefined("theUID")>
        <cfoutput>Event Added. UID is#theUID#</cfoutput>
        <cfset Form.eventID = theUID >
    </cfif>
</cfif>

<cfform format="xml" preservedata="true" style="width:500" height="700">
    <cfinput type="text" label="Subject" name="subject" style="width:435">
            <br />
    <cfinput type="checkbox" label="All Day Event" name="allDay">
    <cfinput type="datefield" label="Date" name="date" validate="date"
        style="width:100">
    <cfinput type="text" label="Start Time" name="startTime" validate="time"
        style="width:100">
    <cfinput type="text" label="End Time" name="endTime" validate="time"
        style="width:100"><br />
    <cfinput type="text" label="Location" name="location"
        style="width:435"><br />
    <cfinput type="text" label="Required Attendees" name="requiredAttendees"
        style="width:435"><br />
    <cfinput type="text" label="Optional Attendees" name="optionalAttendees"
        style="width:435"><br />
    <cfinput type="text" label="Resources" name="resources"
        style="width:435"><br />
    <cfinput type="text" label="Reminder (minutes)" validate="integer"
        name="reminder" style="width:200">
    <cfselect name="importance" label="Importance" style="width:100">
        <option value="normal">Normal</option>
        <option value="high">High</option>
        <option value="low">Low</option>
    </cfselect>
    <cfselect name="sensitivity" label="Sensitivity" style="width:100">
        <option value="normal">Normal</option>
        <option value="company-confidential">Confidential</option>
        <option value="personal">Personal</option>
        <option value="private">Private</option>
    </cfselect>
    <hr />
    <!--- Recurrence Information --->
    <cfinput type="checkbox" label="IsRecurring" name="isRecurring">
    <cfinput type="checkbox" label="RecurrenceNoEndDate" name="noEndDate">
    <cfinput type="text" label="RecurrenceCount" validate="integer"
        required="false" name="recurrenceCount">
    <cfinput type="text" label="RecurrenceEndDate" validate="date"
        required="false" name="recurrenceEndDate">
    <cfselect name="RecurrenceType" label="Recurrence Type"
            style="width:100">
        <option value="DAILY">Daily</option>
        <option value="WEEKLY">Weekly</option>
        <option value="MONTHLY">Monthly</option>
        <option value="YEARLY">Yearly</option>
    </cfselect>
    <cfinput type="text" label="RecurrenceFrequency" validate="integer"
```

```
        name="recurrenceFrequency">
    <cfinput type="checkbox" label="RecurEveryWeekDay"
        name="recurEveryWeekDay">
    <cfinput type="text" label="RecurrenceDays" name="recurrenceDays">
    <cfinput type="text" label="RecurrenceDay" name="recurrenceDay">
    <cfselect name="RecurrenceWeek" label="RecurrenceWeek" style="width:100">
        <option value=""></option>
        <option value="first">First</option>
        <option value="second">Second</option>
        <option value="third">Third</option>
        <option value="fourth">Fourth</option>
        <option value="last">Last</option>
    <cfinput type="text" label="RecurrenceMonth" name="recurrenceMonth">
    </cfselect>
    <hr />
    <cfinput type="textarea" label="Message" name="message" style="width:300;
            height:100">
    <cfinput type="Submit" name="submit" value="Submit" >
</cfform>
```

Microsoft Exchange Server 2010 への接続

Microsoft Exchange Server 2010 との統合のサポート

Adobe ColdFusion は、Microsoft Exchange Server 2010 SP1 と対話できます。拡張機能は、次の操作での効率性をもたらす Microsoft Exchange Web Services（EWS）のサポートを提供します。

- 作成、変更、削除などのフォルダー操作。
- Exchange 組織の会議室および会議室リストの取得。
- 効率的なスケジュールに役立つユーザー空き時間情報。
- 会話の詳細の検索、コピー、移動、会話が読まれているかどうかのステータスなどの会話操作。

注意：ColdFusion 9 では、プロトコルのサポートは WEBDAV に限られました。

注意：ColdFusion 10 の J2EE 構成をインストールする場合は、ews フォルダー（cfusion/lib）のすべての JAR がデプロイ時にシステムクラスパスにある必要があります。

注意：この Beta リリースでは、Microsoft Exchange Server 2010 に対するフォームベースの認証はサポートされません。

Microsoft Exchange Server のバージョンの設定

アプリケーションレベル

次のように変数 exchangeServerVersion に値を指定することにより、アプリケーションレベルで Microsoft Exchange Server のバージョンを指定できます。

<cfset this.exchangeserverversion="version">

これは、cfapplication タグの同じ名前の属性に対応します。

cfapplication

新しい属性 exchangeServerVersion が cfapplication に追加されています。

シンタックス

```
<cfapplication name="app_name" exchangeServerVersion="2010">
...
</cfapplication>
```

属性

属性	必須 / オプション	デフォルト	説明
exchangeServerVersion	オプション	2007	Microsoft Exchange Server のバージョンを指定します。 有効な値は次のとおりです。 • 2003 • 2007 • 2010 詳細を指定しない場合は、デフォルトで 2007 が使用されます。

使用方法

ColdFusion が対話する Microsoft Exchange Server のバージョンを指定するには、この属性を使用します。

Application.cfc でバージョンを設定することもできます。

cfexchange タグのタグレベル

新しい属性 serverVersion が次のタグに追加されました。

- cfexchangeconnection
- cfexchangemail
- cfexchangecalendar
- cfexchangetask
- cfexchangecontact

属性	必須 / オプション	デフォルト	説明
serverVersion	オプション	2007	Microsoft Exchange Server のバージョンを指定します。 有効な値は次のとおりです。 • 2003 • 2007 • 2010 詳細を指定しない場合は、デフォルトで 2007 が使用されます。 ここで指定する値は、アプリケーションレベルで指定されている値を上書きします。

cfexchangemail に追加された新しい属性 folderID

新しい属性 folderID は、アクション get、move および set をサポートします。これは、フォルダーを一意に識別する、大文字と小文字が区別される Exchange UID 値です。指定しないと、folder が使用されます。folder も folderID も指定しないと、操作を実行するためのデフォルトのフォルダーとして受信トレイが使用されます。

cfexchangecalendar タグに追加された新しいアクション

次の 3 つのアクションが追加されました。

- **getUserAvailability**：効率よく会議をスケジュールし、ユーザーの空き時間を検索するため。
- getRoomsList：組織内の会議室のリストを検索するため。
- getRooms：会議室リスト内の会議室を検索するため。

cfexchangecalendar タグに対する変更

cfexchangecalendar のすべてのアクションに対する uid 属性の値は次のとおりです。

- exchangeServerVersion を 2003 または 2007 に設定した場合：uid は開催者のメールボックス内の予定の ID を示します。
- exchangeServerVersion を 2010 に設定した場合：uid は出席者のメールボックス内の受信した予定の ID を示します。

Microsoft Exchange Server 2003 または 2007 との対話の場合、予定が作成されるときは、出席者は応答、削除、添付ファイルの取得などの操作に、開催者の UID を使用できます。Microsoft Exchange Server 2010 の場合は動作が異なります。出席者は、予定関連のアクションを実行する必要がある場合は、最初に自分のメールボックスで予定を検索した後、その予定の UID を使用する必要があります。

リモートサーバーとの対話

Adobe ColdFusion は、複雑な HTTP (Hypertext Transfer Protocol) および FTP (File Transfer Protocol) 通信を簡単なタグシンタックスで実現し、サイトのコンテンツを Web 上で配布できるようにします。

リモートサーバーとの対話について

転送プロトコルとは、ソースから宛先にファイルや情報を移動するために使用するメカニズムのことです。広く使用されているプロトコルには、HTTP (Hypertext Transfer Protocol) および FTP (File Transfer Protocol) の 2 つがあります。ColdFusion の cfhttp および cfftp タグを使用すれば、これらのプロトコルを使用してリモートサーバーと対話することができます。

Web ブラウザが HTTP を使用して Web ページを転送するのと同様に、cfhttp タグを使用して Web ページまたは Web ベースのファイルを受信できます。Web ブラウザに URL を入力すると、Web サーバーに HTTP リクエストが送信されます。cfhttp タグを使用すると、Web ページの表示、ColdFusion または CGI アプリケーションへの変数の送信、Web ページからの専用コンテンツの取得、テキストファイルからの ColdFusion クエリーの作成などが行えます。Get または Post メソッドを使用すれば、リモートサーバーと対話することができます。

cfftp タグは FTP の主要な目的であるファイルの転送に役立ちます。HTTP とは異なり、FTP はデータの処理および受け渡しを目的として他のサーバーと対話するようには設計されていません。cfhttp タグを使用して FTP 接続を確立すると、FTP を使用してファイルやディレクトリのアップロード、ダウンロード、および管理が行えます。

cfhttp タグによる Web との対話

リモートサーバーから情報を取得できる cfhttp タグは、CFML タグセットの中でも非常に強力なタグの 1 つです。cfhttp タグを使用すると、Get または Post メソッドを使用してリモートサーバーと対話できます。

- Get メソッドを使用する場合、リモートサーバーに渡す情報はすべて URL の中に含められます。このメソッドは、cfhttp タグでオブジェクトを取得する一方向のトランザクションでよく使用されます。
- Post メソッドを使用すると、ColdFusion ページや CGI プログラムに変数を渡すことができます。渡した変数は ColdFusion ページや CGI プログラムで処理され、呼び出しページにデータが返されます。呼び出しページは、受信したデータを表示するか、またはさらに処理を行います。たとえば、cfhttp タグを使用して別の ColdFusion ページに

Post した場合、その ColdFusion ページは表示されません。リクエストの処理結果が元の ColdFusion ページに返されると、元の ColdFusion ページが必要に応じてその情報を処理します。

cfhttp の Get メソッドの使用

指定したサーバーからテキストファイルやバイナリファイルなどのファイルを取得するには、Get を使用します。取得した情報は、cfhttp.fileContent という特別な変数に保存されます。次の例は、一般的な Get 操作を示しています。

ファイルの取得と変数への保存

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Use Get Method</title>
</head>
<body>
<cfhttp
    method="Get"
    url="http://www.adobe.com"
    resolveurl="Yes">
<cfoutput>
    #cfhttp.FileContent# <br>
</cfoutput>

</body>
</html>
```

2 (オプション) url 属性の値を他の URL に置き換えます。

3 このファイルに "get_webpage.cfm" という名前を付けて、**Web** のルートディレクトリの下の "myapps" ディレクトリに保存し、Web ブラウザで表示します。

ブラウザに、url 属性で指定されている Web ページがロードされます。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfhttp method="Get" url="http://www.adobe.com" resolveurl="Yes">`	URL に指定されたページを取得し、そのリンクが正しく表示されるように、相対リンクではなく絶対リンクを指定します。
`<cfoutput> #cfhttp.FileContent#   </cfoutput>`	変数 cfhttp.fileContent に保存されているページをブラウザで表示します。

Web ページの取得とファイルへの保存

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Use Get Method</title>
</head>
<body>

<cfhttp
    method = "Get"
    url="http://www.adobe.com/software"
    path="c:\temp"
    file="adobe_software.htm">
</body>
</html>
```

2 (オプション) url 属性の値を他の URL に置き換え、ファイル名を変更します。

3 (オプション) パスを C:¥temp からハードドライブの適切なパスに変更します。

4 このページに "save_webpage.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

5 指定したパスに移動し、指定したファイルをテキストエディタで表示します (手順 1 の値を使用している場合は、C:¥temp¥macr_software.htm にあります)。

保存したファイルは、ブラウザに正しく表示されません。これは、Get 操作では、指定した Web ページの HTML しか保存されないためです。ページに含まれているフレーム、イメージ、ファイルは保存されません。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明

`<cfhttp`
`method = "Get"`
`url="http://www.adobe.com/software"`
`path="c:\temp"`
`file="macr_software.htm">`
	URL で指定したページを取得して、path および file 属性で指定したファイルに保存します。
	path 属性および file 属性を使用する場合、resolveurl 属性は無視されます。したがって、保存したページを表示しても、フレームやその他の含まれているファイルは表示されません。

バイナリファイルの取得と保存

1 次の内容の ColdFusion ページを作成します。

```
<cfhttp
    method="Get"
    url="http://www.adobe.com/adobe/accessibility/images/spotlight.jpg"
    path="c:\temp"
    file="My_SavedBinary.jpg">
<cfoutput>
    #cfhttp.MimeType#
</cfoutput>
```

2 (オプション) url 属性の値を、ダウンロードするバイナリファイルの URL に置き換えます。

3 (オプション) パスを C:¥temp からハードドライブの適切なパスに変更します。

4 このファイルに "save_binary.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。それを Web ブラウザで開いて MIME タイプを確認します。

5 (オプション) path 属性で指定した場所にあるバイナリファイルを確認します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
<cfhttp method="Get" url="http://www.adobe.com/adobe/accessibility/images/spotlight.jpg" path="c:\temp" file="My_SavedBinary.jpg">	バイナリファイルを取得し、path および file 属性で指定した場所に保存します。
<cfoutput> #cfhttp.MimeType# </cfoutput>	ファイルの MIME タイプを表示します。

テキストファイルからのクエリーオブジェクトの作成

cfhttp タグを使用し、method="Get" および name 属性を指定すると、区切られたテキストファイルからクエリーオブジェクトを作成できます。このテクニックは、テキストファイルを処理して操作するための強力な方法です。クエリーオブジェクトを作成すれば、クエリー内の列を参照して、そのデータに対してさまざまな操作を簡単に行えるようになります。

ColdFusion では、次の方法でテキストファイルが処理されます。

- フィールドの区切り文字は delimiter 属性で指定します。デフォルトはカンマです。
- フィールドのデータに区切り文字が含まれている場合は、フィールド全体をテキスト修飾子の文字で囲みます。テキスト修飾子は、textqualifier 属性で指定します。デフォルトのテキスト修飾子は二重引用符 (") です。
- textqualifier="" はテキスト修飾子がないことを意味します。textqualifier="""" (4 つ続いた二重引用符) を使用すると、二重引用符をテキスト修飾子として明示的に指定します。
- テキスト修飾子がある場合は、すべてのフィールド値をテキスト修飾子の文字で囲みます。
- フィールドにテキスト修飾子文字を含める場合は、その文字を 2 つ重ねて使用します。たとえば、テキスト修飾子が " である場合、フィールドに二重引用符を含めるには "" を使用します。
- テキストの 1 行目は常に列見出しと解釈され、スキップされます。columns 属性に別の名前のセットを指定して、ファイルの列見出しを上書きすることができます。名前は列ごとに指定します。それらの新しい名前は CFML コードで使用できます。ただし、ColdFusion ではファイルの 1 行目はデータとして解釈されません。
- 同じ列見出し名がある場合は、重複する列見出し名にアンダースコアを付けて一意の名前にします。たとえば、2 つの CustomerID 列が検出された場合は、2 番目の名前が "CustomerID_" に変更されます。

テキストファイルからのクエリーの作成

1 次の内容のテキストファイルを作成します。

```
OrderID,OrderNum,OrderDate,ShipDate,ShipName,ShipAddress
001,001,01/01/01,01/11/01,Mr. Shipper,123 Main Street
002,002,01/01/01,01/28/01,Shipper Skipper,128 Maine Street
```

2 このファイルに "text.txt" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

3 次の内容の ColdFusion ページを作成します。

```
<cfhttp method="Get"
    url="http://127.0.0.1/myapps/text.txt"
    name="juneorders"
    textqualifier="">

<cfoutput query="juneorders">
    OrderID: #OrderID#<br>
    Order Number: #OrderNum#<br>
    Order Date: #OrderDate#<br>
</cfoutput>
<!--- Now substitute different column names --->
<!--- by using the columns attribute --->
<hr>
Now using replacement column names<br>

<cfhttp method="Get"
    url="http://127.0.0.1/myapps/text.txt"
    name="juneorders"
    columns="ID,Number,ODate,SDate,Name,Address"
    textqualifier="">

<cfoutput query="juneorders">
    Order ID: #ID#<br>
    Order Number: #Number#<br>
    Order Date: #SDate#<br>
</cfoutput>
```

4 このファイルに "query_textfile.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、Web ブラウザで表示します。

cfhttp の Post メソッドの使用

Cookie、フォームフィールド、CGI、URL、およびファイル変数を、指定の ColdFusion ページまたは CGI プログラムに送信して処理させるには、Post メソッドを使用します。Post 操作を行う場合は、Post する変数ごとに cfhttpparam タグを使用します。Post メソッドで渡したデータは、指定した ColdFusion ページまたは実行可能ファイルで解釈されて、データが返されます。

たとえば、Post メソッドを使用した HTML フォームを作成する場合は、フォームデータを渡すページの名前を指定します。cfhttp の Post メソッドも同様に使用できます。ただし、cfhttp タグを使用する場合、Post を受け取ったページ自体は何も表示しません。

ColdFusion ページへの変数の引き渡し

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>HTTP Post Test</title>
</head>
<body>
<h1>HTTP Post Test</h1>
<cfhttp method="Post"
    url="http://127.0.0.1:8500/myapps/post_test_server.cfm">
    <cfhttpparam type="Cookie"
        value="cookiemonster"
        name="mycookie6">
    <cfhttpparam type="CGI"
        value="cgivar "
        name="mycgi">
    <cfhttpparam type="URL"
        value="theurl"
        name="myurl">
    <cfhttpparam type="Formfield"
        value="twriter@adobe.com"
        name="emailaddress">
    <cfhttpparam type="File"
        name="myfile"
        file="c:\pix\trees.gif">
</cfhttp>
<cfoutput>
File Content:<br>
    #cfhttp.filecontent#<br>
Mime Type:#cfhttp.MimeType#<br>
</cfoutput>
</body>
</html>
```

2 cfhttp の閉じタグの直前にある GIF ファイルへのパスを、サーバー上の適切なパスに置き換えます。

3 このファイルに "post_test.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

注意：次の手順に従って、変数を表示するためのページを作成してください。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfhttp method="Post" url="http://127.0.0.1:8500/myapps/post_test_server.cfm">`	指定したページに HTTP リクエストを Post します。
`<cfhttpparam type="Cookie" value="cookiemonster" name="mycookie6">`	リクエストで Cookie を送信します。
`<cfhttpparam type="CGI" value="cgivar " name="mycgi">`	リクエストで CGI 変数を送信します。
`<cfhttpparam type="URL" value="theurl" name="myurl">`	リクエストで URL を送信します。
`<cfhttpparam type="Formfield" value="twriter@adobe.com" name="emailaddress">`	リクエストでフォームフィールドを送信します。

コード	説明
`<cfhttpparam type="File" name="myfile" file="c"\pix\trees.gif">`	リクエストでファイルを送信します。 </> タグにより、http リクエストを終了します。
`<cfoutput> File Content:  #cfhttp.filecontent# `	Post 先のページがリクエストを処理して作成したファイルの内容を表示します。この場合は、"server.cfm" 内の cfoutput タグによる出力です。
`Mime Type: #cfhttp.MimeType#  </cfoutput>`	作成されたファイルの MIME タイプを表示します。

変数の表示

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head><title>HTTP Post Test</title> </head>
<body>
<h1>HTTP Post Test</h1>
<cffile destination="C:\temp\"
    nameconflict="Overwrite"
    filefield="Form.myfile"
    action="Upload"
    attributes="Normal">
<cfoutput>
    The URL variable is: #URL.myurl# <br>
    The Cookie variable is: #Cookie.mycookie6# <br>
    The CGI variable is: #CGI.mycgi#. <br>
    The Formfield variable is: #Form.emailaddress#. <br>
    The file was uploaded to #File.ServerDirectory#\#File.ServerFile#.
</cfoutput>
</body>
</html>
```

2 C:¥temp¥ をハードドライブ上の適切なディレクトリパスに置き換えます。

3 このファイルに "post_test_server.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

4 ブラウザで "post_test.cfm" を表示し、C:¥temp¥ (または置き換えたパス) からそのファイルを探します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cffile destination="C:\temp\" nameconflict="Overwrite" filefield="Form.myfile" action="Upload" attributes="Normal">`	サーバー上のファイルに、送信されたドキュメントを書き込みます。type="File" 属性を使用してファイルを送信しても、受信するページでは、ファイル変数ではなく、フォーム変数としてそのファイルを受け取ります。cffile タグによって、ファイル変数が作成されます。
`<cfoutput>`	情報を出力します。結果の表示はこのページでは行われません。結果は cfhttp.filecontent 変数に保存されて、Post 元のページに返されます。
`The URL variable is: #URL.myurl#  `	HTTP リクエストで送信した URL 変数の値を出力します。
`The Cookie variable is: #Cookie.mycookie#  `	HTTP リクエストで送信した Cookie 変数の値を出力します。

コード	説明
`The CGI variable is: #CGI.mycgi#  `	HTTP リクエストで送信した CGI 変数の値を出力します。
`The Form variable is: #Form.emailaddress#.  `	HTTP リクエストで送信したフォーム変数を出力します。type="formField" 属性を使用して変数を送信しても、受信するページでは、フォーム変数として受け取ります。
`The file was uploaded to #File.ServerDirectory#\#File.ServerFile#. </cfoutput>`	このページに cffile タグの結果を出力します。この場合、変数はファイル変数です。

CGI プログラムの結果を返す

次のコードでは、Web サイト上の CGI プログラム "search.exe" を実行して、MIME タイプやレスポンスの長さなどの結果を表示します。この "search.exe" プログラムは、"search" パラメータを取ることが必要です。

```
<cfhttp method="Post"
        url="http://www.my_favorite_site.com/search.exe"
        resolveurl="Yes">
    <cfhttpparam type="Formfield"
        name="search"
        value="ColdFusion">
</cfhttp>
<cfoutput>
    Response Mime Type: #cfhttp.MimeType#<br>
    Response Length: #len(cfhttp.filecontent)# <br>
    Response Content: <br>
        #htmlcodeformat(cfhttp.filecontent)#<br>
</cfoutput>
```

cfftp タグによるファイルの操作

cfftp タグを使用すると、リモートサーバーに対して FTP による操作が行えます。ファイルをアップロードまたはダウンロードするときに、cfftp タグを使用して接続をキャッシュし、ファイルのバッチ転送を行うことができます。

注意：cfftp を使用するには、ColdFusion Administrator の [セキュリティ] 領域にある [サンドボックスセキュリティ] ページで、[ColdFusion セキュリティの有効化] オプションを選択します。スタンダード版では、[セキュリティ]-[基本セキュリティ] になります。

サーバーとブラウザ間で操作を行う場合は、cffile、cfcontent、および cfdirectory タグを使用します。

cfftp タグで主に行う操作は、接続とファイル転送の 2 つです。また、ディレクトリをリストするなどのさまざまな操作を行うコマンドも使用できます。FTP 操作をサポートするすべての属性のリストや、cfftp タグの使用方法については、『CFML リファレンス』を参照してください。

FTP 接続の確立とファイルリストの取得

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>FTP Test</title>
</head>

<body>
<h1>FTP Test</h1>
<!--- Open ftp connection --->
<cfftp connection="Myftp"
    server="MyServer"
    username="MyUserName"
    password="MyPassword"
    action="Open"
    stoponerror="Yes">

<!--- Get the current directory name. --->
<cfftp connection=Myftp
    action="GetCurrentDir"
    stoponerror="Yes">

<!--- output directory name --->
<cfoutput>
    The current directory is:#cfftp.returnvalue#<p>
</cfoutput>

<!--- Get a listing of the directory. --->
<cfftp connection=Myftp
    action="listdir"
    directory="#cfftp.returnvalue#"
    name="dirlist"
    stoponerror="Yes">
<!--- Close the connection.--->
<cfftp action="close" connection="Myftp">
<p>Did the connection close successfully?
    <cfoutput>#cfftp.succeeded#</cfoutput></p>

<!--- output dirlist results --->
<hr>
<p>FTP Directory Listing:</p>

<cftable query="dirlist" colheaders="yes" htmltable>
    <cfcol header="<B>Name</b>" TEXT="#name#">
    <cfcol header="<B>Path</b>" TEXT="#path#">
    <cfcol header="<B>URL</b>" TEXT="#url#">
    <cfcol header="<B>Length</b>" TEXT="#length#">
    <cfcol header="<B>LastModified</b>"
    TEXT="#DateFormat(lastmodified)#">
    <cfcol header="<B>IsDirectory</b>"
        TEXT="#isdirectory#">
</cftable>
```

2 MyServer を、FTP でアクセス可能なサーバーの名前に変更します。

3 MyUserName と MyPassword を有効なユーザー名とパスワードに変更します。

匿名接続を確立するには、ユーザー名に「anonymous」と入力し、慣例としてパスワードに電子メールアドレスを入力します。

4 このファイルに "ftp_connect.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、Web ブラウザで表示します。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cfftp connection="Myftp" server="MyServer" username="MyUserName" password="MyPassword" action="Open" stoponerror="Yes">`	MyServer サーバーへの FTP 接続を開き、MyUserName としてログオンします。エラーが発生した場合は、処理を停止し、エラーを表示します。Myftp 接続を指定することで、この接続を他の cfftp タグで使用できます。

`<cfftp connection=Myftp action="GetCurrentDir" stoponerror="Yes">`
`<cfoutput>`
`The current directory is: #cfftp.returnvalue#<p>`
`</cfoutput>`
	Myftp 接続を使用して現在のディレクトリの名前を取得し、エラーが発生した場合は、処理を停止します。
	現在のディレクトリを表示します。

`<cfftp connection=Myftp action="ListDir" directory="#cfftp.returnvalue#" name="dirlist" stoponerror="Yes">`	Myftp 接続を使用してディレクトリリストを取得します。前回の cfftp 呼び出しで返された値 (接続の現在のディレクトリ) を使用して、リストするディレクトリを指定します。結果を変数 dirlist (クエリーオブジェクト) に保存します。エラーが発生した場合は、処理を停止します。

`<cfftp action="close" connection="Myftp">`
`<p>Did the connection close successfully?`
`<cfoutput>#cfftp.succeeded#</cfoutput></p>`
	接続を閉じます。結果を引き続き使用できるように、操作が失敗しても処理を停止しません。代わりに cfftp.succeeded 変数の値を表示します。この値は、接続が閉じた場合は Yes、操作が失敗した場合は No です。

`<cftable query="dirlist" colheaders="yes" htmltable>`
`<cfcol header="<B>Name</b>" TEXT="#name#">`
`<cfcol header="<B>Path</b>" TEXT="#path#">`
`<cfcol header="<B>URL</b>" TEXT="#url#">`
`<cfcol header="<B>Length</b>" TEXT="#length#">`
`<cfcol header="<B>LastModified</b>"`
`TEXT="#DateFormat(lastmodified)#">`
`<cfcol header="<B>IsDirectory</b>" TEXT="#isdirectory#">`
`</cftable>`
	ListDir FTP コマンドの結果を表形式で表示します。

cfftp タグで接続を確立すると、アプリケーションまたはサーバーが接続を閉じるまで、その接続を再利用して別の FTP 操作を実行できます。既にアクティブな FTP 接続を使用するときは、ユーザー名、パスワード、またはサーバーを再び指定する必要はありません。複数のフレームを使用している場合、特定の接続オブジェクトを使用できるフレームは 1 つのみです。

注意： GetFile や PutFile などの単純な FTP 操作を 1 回のみ行う場合は、接続を確立する必要はありません。1 つの cfftp リクエストに、サーバー、ログイン、パスワードなどの必要なログイン情報をすべて指定できます。

複数のページで使用できる接続キャッシュ

Application スコープまたは Session スコープの変数に接続を明示的に割り当てないかぎり、cfftp タグによって確立された FTP 接続が有効なのは、現在のページのみです。

アプリケーション変数に cfftp 接続を割り当てると、複数のユーザーが同じ接続オブジェクトに同時にアクセスすることがあるので、問題が発生する可能性があります。セッション変数に cfftp 接続を割り当てれば、単一のクライアントのみが接続を使用でき、セッションの間のみ接続が保持されるので、この方法をお勧めします。

例 : 接続のキャッシュ

```
<cflock scope="Session" timeout=10>
<cfftp action="Open"
    username="anonymous"
    password="me@home.com"
    server="ftp.eclipse.com"
    connection="Session.myconnection">
</cflock>
```

この例では、現在のセッション内の他のページでも接続キャッシュを引き続き使用できます。この方法を使用する場合は、アプリケーションのセッション変数を有効にし、セッション変数を使用するコードをロックします。ロックの詳細については、300 ページの「永続データとロックの使用」を参照してください。

注意： retrycount 値や timeout 値などの接続特性を変更すると、接続の再確立が必要になる場合があります。

接続のアクションと属性

次の表に、使用可能な cfftp アクションと、名前付きの (キャッシュされた) 接続を使用する場合に指定する必要がある属性を示します。既存の接続名を指定しない場合は、username 属性、password 属性、および server 属性を指定する必要があります。

アクション	属性	アクション	属性
Open	none	Rename	existing new
Close	none	Remove	server item
ChangeDir	directory	GetCurrentDir	none
CreateDir	directory	GetCurrentURL	none
ListDir	name directory	ExistsDir	directory
RemoveDir	directory	ExistsFile	remotefile
GetFile	localfile remotefile	Exists	item
PutFile	localfile remotefile		

サーバー上のファイルの管理

cffile、cfdirectory、および cfcontent タグを使用すれば、ブラウザとサーバーの間でファイル管理タスクを行うことができます。たとえば、クライアントから Web サーバーへのファイルのアップロード、ディレクトリ情報の表示、Web ブラウザに送信するコンテンツタイプの変更などが行えます。サーバーとサーバーの間で操作を行う場合は、1228 ページの「cfftp タグによるファイルの操作」で説明している cfftp タグを使用します。

ファイル管理について

Adobe ColdFusion では、ColdFusion サーバー上のファイルやディレクトリにアクセスして、それらを管理できます。cffile タグには、ファイルの移動、コピー、削除、名前の変更などを行う属性が用意されています。cfdirectory タグを使用すると、ディレクトリの一覧表示、作成、削除、および名前の変更を行えます。cfcontent タグでは、Web ブラウザに返す MIME (Multipurpose Internet Mail Extensions) コンテンツタイプを定義できます。

cffile タグの使用

cffile タグを使用すると、サーバー上のファイルに対して、次のようなさまざまな操作を行えます。

- HTML フォームを使用してクライアントから Web サーバーにファイルをアップロードする。
- サーバー上のファイルの移動、名前の変更、コピー、または削除を行う。
- サーバー上のテキストファイルを読み書きしたり、テキストを追加したりする。

ファイルアクションを指定するには、action 属性を使用します。指定できるアクションは、upload、move、rename、copy、delete、read、readBinary、write、および append です。必要な属性は、指定する action 属性によって異なります。たとえば、action="write" と指定した場合は、テキストファイルを書き込むための属性が必要です。

注意：サーバー上のディレクトリにユーザーがアクセスできるようにする前に、そのセキュリティや論理構造を確認してください。cffile タグは ColdFusion Administrator で無効にできます。また、ローカルの ColdFusion システム上にないファイルにアクセスするには、リモートのファイルやディレクトリにアクセス可能なアカウントを使用して ColdFusion サービスを実行する必要があります。

ファイルのアップロード

ファイルをアップロードするには、次の 2 つのファイルを作成します。

- ファイルのアップロード情報を指定するための HTML フォーム
- ファイルをアップロードするコードを記述したアクションページ

次の手順で、これらのファイルの作成方法を示します。

ファイルのアップロード情報を指定するための HTML ファイルの作成

1 次の内容の ColdFusion ページを作成します。

```
<head><title>Specify File to Upload</title></head>
<body>
<h2>Specify File to Upload</h2>
<!--- the action attribute is the name of the action page --->
<form action="uploadfileaction.cfm"
        enctype="multipart/form-data"
        method="post">
    <p>Enter the complete path and filename of the file to upload:
    <input type="file"
            name="FiletoUpload"
            size="45">
    </p>
    <input type="submit"
            value="Upload">
</form>
</body>
```

2 このファイルに "uploadfileform.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、ブラウザで表示します。

注意：このフォームは、アクションページを作成するまで動作しません (次の手順を参照)。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<form action="uploadfileaction.cfm" enctype="multipart/form-data" method="post">`	ユーザーがファイルをアップロードするためのファイル選択フィールドを備えたフォームを作成します。action 属性値では、フォームを処理する ColdFusion テンプレートを指定します。enctype 属性値では、フォームにアップロードファイルが含まれていることを示します。ColdFusion フォームを送信するために、method 属性を post に設定します。
`<input type="file" name="FiletoUpload" size="45">`	アップロードするファイルをユーザーが指定できるようにします。file タイプを設定することで、ユーザーのシステムからファイルを読み取ってサーバーに転送できるようにします。これによって、[参照] ボタンが自動的に表示されるので、パスおよびファイル名を入力しなくても該当のファイルを参照できます。

ユーザーは、ファイルパスを入力するかシステムを参照して、送信するファイルを選択できます。

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head> <title>Upload File</title> </head>
<body>
<h2>Upload File</h2>

<cffile action="upload"
        destination="c:\temp\"
        nameConflict="overwrite"
        fileField="Form.FiletoUpload">

<cfoutput>
You uploaded #cffile.ClientFileName#.#cffile.ClientFileExt#
            successfully to #cffile.ServerDirectory#.
</cfoutput>

</body>
</html>
```

2 次の行を変更して、サーバー上の適切な場所を指定します。

```
destination="c:\temp\"
```

注意：このディレクトリはサーバー上に存在する必要があります。

3 このファイルに "uploadfileaction.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

4 "uploadfileform.cfm" をブラウザで表示し、アップロードするファイルを入力してフォームを送信します。

指定したファイルがアップロードされます。

コードの説明

このコードおよびその機能について、次の表で説明します。

コード	説明
`<cffile action="upload"`	クライアントマシン上のアップロードファイルの名前と場所を出力します。
`destination="c:\temp\"`	ファイルの保存先を指定します。
`nameConflict="overwrite"`	同じ名前のファイルが存在する場合は上書きします。
`fileField="Form.FiletoUpload">`	アップロードするファイルの名前を指定します。変数を # 記号で囲まないでください。
`You uploaded #cffile.ClientFileName#.#cffile.ClientFileExt# successfully to #cffile.ServerDirectory#.`	アップロードしたファイルとその保存先を表示します。スコープ変数の詳細については、1235 ページの「ファイルアップロードの結果の評価」を参照してください。

注意：この例では、エラーの確認やセキュリティ対策は行っていません。ファイルをアップロードするアプリケーションをデプロイする場合は、エラー処理とセキュリティの両方を組み込んでください。詳細については、337 ページの「アプリケーションの保護」および 275 ページの「エラー処理」を参照してください。

競合するファイル名の解決

サーバーにファイルを保存する場合、同じ名前のファイルが既に存在している可能性があります。この問題を解決するには、cffile タグの nameConflict 属性に、次のいずれかの値を割り当てます。

Error（デフォルト）ページの処理が停止され、エラーが返されます。ファイルは保存されません。

Skip ファイルのプロパティに基づいて独自の処理が行えるようにします。ファイルは保存されず、エラーも返されません。

Overwrite アップロードするファイルと同じ名前のファイルを上書きします。

MakeUnique アップロードするファイルに一意の名前を付けます。この名前は、ファイルオブジェクト変数 serverFile および serverFileName に保管されます。この変数を使用して、ファイルの保存時に使用された名前を記録することができます。一意の名前は、意図した名前と異なる場合があります。ファイルアップロードステータス変数の詳細については、1235 ページの「ファイルアップロードの結果の評価」を参照してください。

アップロードするファイルタイプの制御

アプリケーションによっては、アップロードできるファイルのタイプを制限したい場合があります。たとえば、ドキュメントライブラリにグラフィックファイルがアップロードされないようにしたいことがあります。

アップロードできるファイルのタイプを制限するには、accept 属性を使用します。accept 属性を指定すると、ファイルの MIME コンテンツタイプが指定の条件に一致している場合にのみファイルがアップロードされます。一致していない場合はエラーが発生します。accept 属性には、MIME データ名をカンマで区切ってリストします。MIME データ名には、ワイルドカード文字を含めることができます。

ファイルの MIME タイプはブラウザによって判断されます。image/gif や text/plain などの一般的なタイプは、ブラウザに登録されています。

注意：最新バージョンの Microsoft Internet Explorer および Netscape では、MIME タイプの関連付けがサポートされています。他のブラウザおよび旧バージョンでは、これらの関連付けが無視される場合があります。

accept 属性を省略するか、または "*/*" を指定すると、アップロードするすべてのファイルが保存されます。ファイルタイプは、次の例のように制限できます。

次の cffile タグは、GIF 形式のイメージファイルのみを保存します。

```
<cffile action="Upload"
    fileField="Form.FiletoUpload"
    destination="c:\uploads\"
    nameConflict="Overwrite"
    accept="image/gif">
```

次の cffile タグは、GIF 形式または JPEG 形式のイメージファイルのみを保存します。

```
<cffile action="Upload"
    fileField="Form.FiletoUpload"
    destination="c:\uploads\"
    nameConflict="Overwrite"
    accept="image/gif, image/jpeg">
```

注意：" アップロードされたファイル (image/jpeg) の MIME タイプを、サーバーが受け入れませんでした。" というエラーメッセージが表示された場合は、「accept="image/jpeg"」と入力して JPEG ファイルを受け入れてください。

次の cffile タグでは、形式にかかわらずイメージファイルが保存されます。

```
<cffile action="Upload"
    fileField="Form.FiletoUpload"
    destination="c:\uploads\"
    nameConflict="Overwrite"
    accept="image/*">
```

ファイル属性とディレクトリ属性の設定

Windows では、cffile タグの attributes 属性を使用してファイル属性を指定します。UNIX では、cffile または cfdirectory タグの mode 属性を使用してファイルやディレクトリへのアクセス許可を指定します。

Windows

Windows では、次のファイル属性を設定できます。

- Hidden

- Normal
- ReadOnly

CFML で複数の属性を指定するには、attributes="ReadOnly,Hidden" のように、attributes 属性でカンマ区切りリストを使用します。attributes 属性を使用しない場合は、ファイルの既存の属性が保持されます。Normal とともに他の属性を指定すると、Normal 設定は他の属性によって上書きされます。

UNIX

UNIX では、所有者、グループ、他のユーザーの 3 タイプのユーザーに対して、ファイルやディレクトリへのアクセス許可を個別に設定できます。それには、各ユーザータイプに対して数値を割り当てます。この数値は、許可するアクセス許可の数値を合計したものです。mode 属性の値は、UNIX の chmod コマンドの 8 進数値に対応します。

- 4 = 読み取り
- 2 = 書き込み
- 1 = 実行

各タイプのユーザーについて、所有者、グループ、他のユーザーの順序で、mode 属性にアクセス許可の値を入力します。たとえば、どのタイプのユーザーにも読み取り権限を割り当てるには、次のコードを使用します。

```
mode=444
```

ファイルまたはディレクトリの所有者に読み取り / 書き込み / 実行の許可を与え、それ以外のユーザーには読み取り許可のみを与えるには、次のコードを使用します。

```
mode=744
```

ファイルアップロードの結果の評価

ファイルのアップロードが完了したら、ファイルアップロードステータス変数を使用してステータス情報を取得できます。このステータス情報には、ファイルの名前や保存されているディレクトリなどのファイル関連データが含まれます。

ファイルアップロードステータス変数にアクセスするには、ドット表記法を使用して file.varname または cffile.varname の形式で指定します。接頭辞には File と cffile のいずれも使用できますが、cffile 接頭辞を使用することをお勧めします。たとえば、cffile.ClientDirectory のようにします。File 接頭辞は、以前のバージョンとの互換性を保つために残されています。

注意：ファイルステータス変数は読み取り専用です。これらの変数には、最新の cffile 操作の結果が設定されます。2 つの cffile タグを実行すると、最初のタグの実行結果は、次の cffile 操作によって上書きされます。

次の表で、アップロード後に使用できるファイルアップロードステータス変数について説明します。

変数	説明
attemptedServerFile	ファイルを保存するために最初に使用が試みられた名前。たとえば、myfile.txt など (1233 ページの「競合するファイル名の解決」を参照)。
clientDirectory	アップロード元のファイルが格納されているクライアントシステム上のディレクトリ。
clientFile	クライアントシステム上にあるソースファイルの完全な名前 (ファイル名拡張子あり)。たとえば、myfile.txt など。
clientFileExt	クライアントシステム上にあるソースファイルの拡張子 (ピリオドなし)。たとえば、txt など (.txt ではない)。
clientFileName	クライアントシステム上にあるソースファイルの名前 (拡張子なし)。たとえば、myfile など。
contentType	保存されたファイルの MIME コンテンツタイプ。たとえば、image/gif の image など。
contentSubType	保存されたファイルの MIME コンテンツサブタイプ。たとえば、image/gif の gif など。

変数	説明
dateLastAccessed	アップロードされたファイルが最後にアクセスされた日付。
fileExisted	同じパスを持つファイルが存在しているかどうかを Yes または No で示します。
fileSize	アップロードされたファイルのサイズ。
fileWasAppended	アップロードされたファイルが既存のファイルに追加されたかどうかを Yes または No で示します。
fileWasOverwritten	ファイルが上書きされたかどうかを Yes または No で示します。
fileWasRenamed	名前の競合を避けるためにアップロードファイルの名前が変更されたかどうかを Yes または No で示します。
fileWasSaved	アップロードされたファイルが保存されたかどうかを Yes または No で示します。
oldFileSize	ファイルのアップロード操作によって上書きされたファイルのサイズ。上書きされたファイルがない場合は空です。
serverDirectory	ファイルが保存されたサーバー上のディレクトリ。
serverFile	サーバーに保存されたファイルの完全な名前 (ファイル名拡張子あり)。たとえば、myfile.txt など。
serverFileExt	サーバーにアップロードされたファイルの拡張子 (ピリオドなし)。たとえば、txt など (.txt ではない)。
serverFileName	サーバーに保存されたファイルの名前 (拡張子なし)。たとえば、myfile など。
timeCreated	アップロードされたファイルが作成された日時。
timeLastModified	アップロードされたファイルが最後に変更された日時。

サーバーファイルの移動、名前の変更、コピー、および削除

cffile タグを使用すると、Web サーバー上のファイルを管理するアプリケーションページを作成できます。このタグを使用して、ディレクトリ間のファイルの移動、ファイル名の変更、ファイルのコピーや削除を行うことができます。

次の表の例では、多くの属性でスタティック値を使用していますが、cffile タグの属性値にはダイナミックパラメータを使用できます。

アクション	コードの例
ファイルの移動	`<cffile action="move" source="c:\files\upload\KeyMemo.doc" destination="c:\files\memo\">`
ファイル名の変更	`<cffile action="rename" source="c:\files\memo\KeyMemo.doc" destination="c:\files\memo\OldMemo.doc">`
ファイルのコピー	`<cffile action="copy" source="c:\files\upload\KeyMemo.doc" destination="c:\files\backup\">`
ファイルの削除	`<cffile action="delete" file="c:\files\upload\oldfile.txt">`

次の例では、アップロードされたファイルに ReadOnly フラグビットを設定します。

```
<cffile action="Copy"
    source="c:\files\upload\keymemo.doc"
    destination="c:\files\backup\"
    attributes="ReadOnly">
```

注意：destination ディレクトリの末尾には円記号 (¥) を使用してください。使用しなかった場合は、パスの最後の要素がファイル名と解釈されます。このルールは、コピーアクションにのみ適用されます。

テキストファイルの読み書きおよびテキストの追加

cffile タグを使用すると、サーバー上のファイルの管理に加えて、テキストファイルの読み取り、作成、および変更を行うことができます。たとえば、次のことが行えます。

- ログファイルを作成できます。ログファイルの作成と書き込みは、cflog タグでも行えます。
- スタティックな HTML ドキュメントを生成できます。
- テキストファイルを使用して、Web ページに挿入する情報を保管できます。

テキストファイルの読み取り

cffile タグを使用すると、既存のテキストファイルを読み取ることができます。既存のファイルをローカル変数に格納して、アプリケーションページの任意の場所で使用できます。たとえば、テキストファイルを読み取って、その内容をデータベースに挿入したり、文字列置換関数を使用してその内容を変更したりできます。

テキストファイルの読み取り

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Read a Text File</title>
</head>

<body>
Ready to read the file:<br>
<cffile action="read"
    file="C:\inetpub\wwwroot\mine\message.txt"
    variable="Message">

<cfoutput>
    #Message#
</cfoutput>
</body>
</html>
```

2 C:¥inetpub¥wwwroot¥mine¥message.txt を、サーバー上の適切なテキストファイルの場所と名前に置き換えます。

3 このファイルに "readtext.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、ブラウザで表示します。

サーバーへのテキストファイルの書き込み

cffile タグを使用すると、ダイナミックコンテンツに基づいてテキストファイルを書き込むことができます。たとえば、テキストファイル内にスタティックな HTML ファイルやログアクションを作成できます。

テキストファイル用のデータを取り込むフォームの作成

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Put Information into a Text File</title>
</head>

<body>
<h2>Put Information into a Text File</h2>

<form action="writetextfileaction.cfm" method="Post">
    <p>Enter your name: <input type="text" name="Name" size="25"></p>
    <p>Enter the name of the file: <input type="text" name="FileName" size="25">.txt</p>
    <p>Enter your message:
    <textarea name="message"cols=45 rows=6></textarea>
    </p>
    <input type="submit" name="submit" value="Submit">
</form>
</body>
</html>
```

2 このファイルに "writetextfileform.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

注意：このフォームは、アクションページを作成するまで動作しません (次の手順を参照)。

テキストファイルの書き込み

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>Write a Text File</title>
</head>
<body>
<cffile action="write"
    file="C:\inetpub\wwwroot\mine\#Form.FileName#.txt"
    output="Created By: #Form.Name#
#Form.Message# ">
</body>
</html>
```

2 パス C:¥inetpub¥wwwroot¥mine¥ を、サーバー上の適切なパスに変更します。

3 このファイルに "writetextfileaction.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

4 ブラウザで "writetextfileform.cfm" ファイルを表示し、フォームに値を入力して送信します。

指定した場所にテキストファイルが書き込まれます。同じ名前のファイルが存在する場合、そのファイルは上書きされます。

テキストファイルへのテキストの追加

cffile タグを使用すると、テキストファイルの末尾にテキストを追加できます。これは、ログファイルの作成時などに役立ちます。

テキストファイルへのテキストの追加

1 "writetextfileaction.cfm" ファイルを開きます。

2 action 属性の値を write から append に変更して、次のようにします。

```
<html>
<head>
    <title>Append a Text File</title>
</head>
<body>
<cffile action="append"
    file="C:\inetpub\wwwroot\mine\message.txt"
    output="Appended By: #Form.Name#">
</body>
</html>
```

3 このファイルに「writetextfileaction.cfm」という名前を付けて、**web_root** の下の myapps ディレクトリに保存します。

4 ブラウザでこのファイルを表示し、フォームに値を入力して送信します。

追加した情報はテキストファイルの末尾に表示されます。

cfdirectory の使用

指定のディレクトリからファイル情報を取得したり、ディレクトリの作成、削除、名前の変更を行ったりするには、cfdirectory タグを使用します。ファイルのリストを取得する場合と、ディレクトリを削除する場合は、オプションの recurse 属性を使用して、すべてのサブディレクトリを対象に取得や削除を行うことができます。

cffile タグと同様に、ColdFusion Administrator で cfdirectory 処理を無効にできます。このタグのシンタックスの詳細については、『CFML リファレンス』を参照してください。

ファイル情報の取得

cfdirectory タグで action="list" と指定すると、name 属性で指定した名前のクエリーオブジェクトが返されます。name 属性は、action="list" と指定した場合は必須です。このクエリーオブジェクトの結果列は、name 属性で指定した値を使用して、cfoutput タグの中で参照できます。

name ディレクトリエントリ名

directory エントリを含むディレクトリ

size ディレクトリエントリのサイズ

type ファイルタイプ (File または Dir)

dateLastModified 最後にエントリが変更された日付

attributes (Windows のみ) ファイルの属性 (存在する場合)

mode (UNIX のみ) 指定のディレクトリへのアクセス許可の設定値を表す 8 進数値

注意：cfdirectory で attributes 属性をソートする場合、サポートされる値は ReadOnly と Hidden です。

サーバーが UNIX システムにあるか Windows システムにあるかに応じて、Attributes 列または Mode 列が空白になります。また、filter 属性でファイル名を指定して、単一のファイルの情報を取得することもできます。

次の手順では、ディレクトリ情報を表示する ColdFusion ページの作成方法について説明します。

ディレクトリ情報の表示

1 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
    <title>List Directory Information</title>
</head>

<body>
<h3>List Directory Information</h3>
<cfdirectory
    directory="c:\inetpub\wwwroot\mine"
    name="mydirectory"
    sort="size ASC, name DESC, datelastmodified">

<table cellspacing=1 cellpadding=10>
<tr>
    <th>Name</th>
    <th>Size</th>
    <th>Type</th>
    <th>Modified</th>
    <th>Attributes</th>
    <th>Mode</th>
</tr>
<cfoutput query="mydirectory">
<tr>
    <td>#mydirectory.name#</td>
    <td>#mydirectory.size#</td>
    <td>#mydirectory.type#</td>
    <td>#mydirectory.dateLastModified#</td>
    <td>#mydirectory.attributes#</td>
    <td>#mydirectory.mode#</td>
</tr>
</cfoutput>
</table>

</body>
</html>
```

2 パス C:¥inetpub¥wwwroot¥mine¥ を、サーバー上の適切なパスに変更します。

3 このファイルに "directoryinfo.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、ブラウザで表示します。

cfcontent の使用

cfcontent タグを使用すると、サーバーからクライアントにファイルをダウンロードできます。このタグでは、ColdFusion ページから返されるコンテンツの MIME タイプを設定できます。また、オプションで、現在のページでダウンロードするファイルの名前も定義できます。デフォルトでは、MIME コンテンツタイプとして text/html が返され、テンプレートテキストが Web ページとしてブラウザに表示されます。

cffile タグおよび cfdirectory タグと同様に、ColdFusion Administrator で処理を無効にできます。

MIME タイプについて

MIME タイプとは、ファイルのコンテンツを識別するラベルです。ブラウザは、指定されている MIME タイプに従ってファイルを処理します。たとえば、MIME コンテンツタイプでスプレッドシートファイルと指定されているファイルがブラウザで検出されると、スプレッドシートプログラムが起動します。

MIME コンテンツタイプは、"type/subtype" の形式で指定します。一般的な MIME コンテンツタイプの例としては、次のものがあります。

- text/html

- image/gif
- application/pdf

cfcontent による MIME コンテンツタイプの変更

ColdFusion ページで生成されたコンテンツとともにブラウザに返す MIME コンテンツタイプを変更するには、cfcontent タグを使用します。

cfcontent タグには必須属性 type があり、これは現在のページによって返される MIME コンテンツタイプを定義します。

cfcontent による MIME コンテンツタイプの変更

1 次の内容の HTML ページを作成します。

```
<h1>cfcontent_message.htm</h1>

<p>This is a <em>test message</em> written in HTML.</p>
<p>This is the <em>second paragraph</em> of the test message.
As you might expect, it is also written in HTML.</p>
```

2 このファイルに "cfcontent_message.htm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存します。

この HTML ファイルは、手順 3 ～ 7 で作成する ColdFusion ファイルで呼び出します。

3 次の内容の ColdFusion ページを作成します。

```
<html>
<head>
<title>cfcontent Example</title>
</head>

<body>
<h3>cfcontent Example</h3>

<cfcontent
    type = "text/html"
    file = "C:\CFusion\wwwroot\myapps\cfcontent_message.htm"
    deleteFile = "No">
</body>
</html>
```

4 必要に応じて file = 行を編集して、"myapps" ディレクトリを指定します。

5 このファイルに "cfcontent.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、ブラウザで表示します。

呼び出したファイル (cfcontent_message.htm) のテキストは、通常の HTML ファイルとして表示されます。

6 "cfcontent.cfm" で、type = "text/html" を type = "text/plain" に変更します。

7 ファイルを保存し、ブラウザで表示します (必要に応じてブラウザを更新します)。

テキストは形式設定されていないテキストとして表示されます。この場合、HTML のタグはテキストとして表示されます。

次の例では、cfcontent タグを使用して Excel スプレッドシートを作成する方法を示します。

cfcontent タグを使用した Excel スプレッドシートの作成

1 次の内容の ColdFusion ページを作成します。

```
<!--- Use cfsetting to block output of HTML
outside cfoutput tags. --->
<cfsetting enablecfoutputonly="Yes">

<!--- Get employee info. --->
<cfquery name="GetEmps" datasource="cfdocexamples">
    SELECT * FROM Employee
</cfquery>

<!--- Set content type. --->
<cfcontent type="application/msexcel">

<!--- Suggest default name for XLS file. --->
<!--- "Content-Disposition" in cfheader also ensures
relatively correct Internet Explorer behavior. --->
<cfheader name="Content-Disposition" value="filename=Employees.xls">

<!--- Format data using cfoutput and a table.
        Excel converts the table to a spreadsheet.
        The cfoutput tags around the table tags force output of the HTML when
        using cfsetting enablecfoutputonly="Yes" --->
<cfoutput>
    <table cols="4">
        <cfloop query="GetEmps">
            <tr>
                <td>#Emp_ID#</td>
                <td>#FirstName#</td>
                <td>#LastName#</td>
            </tr>
        </cfloop>
    </table>
</cfoutput>
```

2 このファイルに "employees_to_excel.cfm" という名前を付けて、<Web のルートディレクトリ > の下の "myapps" ディレクトリに保存し、ブラウザで表示します。

Excel スプレッドシートにデータが表示されます。

イベントゲートウェイの使用

Adobe ColdFusion が提供するイベントゲートウェイを使用して、アプリケーションを作成できます。アプリケーション用にイベントゲートウェイを設定し、アプリケーションをデプロイします。

イベントゲートウェイを使用するには、CFC (ColdFusion コンポーネント) などの ColdFusion の開発に必要な概念や方法を既に習得している必要があります。ColdFusion に同梱されていないカスタムゲートウェイを使用するアプリケーションを作成するには、イベントゲートウェイの要件など、使用するイベントゲートウェイの詳細も理解している必要があります。

イベントゲートウェイについて

ColdFusion イベントゲートウェイは、ColdFusion が非同期で外部のイベントやメッセージに応答したり、イベントやメッセージを生成したりするために使用する ColdFusion 要素です。ColdFusion アプリケーションでイベントゲートウェイを使用すれば、HTTP リクエスト以外の方法で情報を受け取って処理することができます。たとえばイベントゲートウェイを使用して、インスタントメッセージ、モバイルデバイスからのショートメッセージ、TCP/IP ポートに送信されたメッセージなどを処理することができます。

イベントゲートウェイのメカニズムの主な特徴は次のとおりです。

- ColdFusion イベントゲートウェイは HTTP リクエストを必要としません。ColdFusion の開発者は、CFM ページを使用しなくても (CFC を使用するだけで) ColdFusion ゲートウェイアプリケーションを作成できます。
- ColdFusion CFC では、イベントゲートウェイを使用して、外部イベントを直接リスンし、応答することができます。
- イベントゲートウェイは非同期で処理を行います。一般に、メッセージを受け取ったゲートウェイは、レスポンスを求めたり待機したりすることなく、処理を実行するためにメッセージを送信します。
- ColdFusion の開発者は、イベントゲートウェイを作成することで、Java アプリケーションで受信可能な任意のイベントを処理できます。

ColdFusion には、製品レベルのイベントゲートウェイがいくつか同梱されています。たとえば、XMPP (Extensible Messaging and Presence Protocol) というインスタントメッセージングプロトコル用のゲートウェイなどがあります。またアドビ システムズ社では、汎用的なソケットゲートウェイなど、ニーズに応じてカスタマイズできるいくつかのサンプルゲートウェイのソースも提供しています。さらに、Java で独自のゲートウェイを作成して、Java ランタイムやサードパーティプロバイダでサポートされているその他のイベントやメッセージングテクノロジー (たとえば、他のインスタントメッセージングプロトコル、特定の ERP システム、NNTP などのその他のプロトコル) を処理することもできます。

イベントゲートウェイの使用

イベントゲートウェイは、汎用的な非同期メッセージングメカニズムであり、さまざまなイベントやメッセージングリソースを処理できます。たとえば、ColdFusion には、ColdFusion アプリケーションと次のリソースの間で通信を行うためのゲートウェイ (製品品質または軽量サンプルのいずれか) が用意されています。

- SMS (Short Messaging Services) をサポートする携帯電話またはその他のデバイス
- XMPP または IBM Sametime インスタントメッセージクライアント
- Java ソケット (このソケットを使用すると、Telnet 端末クライアントなどの TCP/IP ベースのデバイスやプログラムと通信できます)
- 店頭販売注文処理システムなどの JMS (Java Messaging Service) リソース

イベントゲートウェイは、通信プロトコルを使用した情報の送受信以外にも使用できます。たとえば ColdFusion には、ディレクトリに対する変更を監視し、ディレクトリが変更されると CFC メソッドを呼び出すサンプルイベントゲートウェイが同梱されています。また、CFML アプリケーションが CFC を非同期で呼び出して、CFC からの応答を受けずに処理を続行できるイベントゲートウェイも同梱されています。

さまざまなイベントテクノロジーやメッセージングテクノロジー用のイベントゲートウェイを作成できると同時に、こうしたゲートウェイを使用する多数のアプリケーションを作成できます。実現可能なゲートウェイの例を次に示します。

サーバーからクライアントへのプッシュの例

- 注文書の承認、応答の取得、および注文書に対する承認または却下のマーキングを行う担当者に IM (インスタントメッセージ) または SMS テキストメッセージを送信するアプリケーション
- 監視リストにある株価が上昇したときに、あらかじめ設定されているメッセージング方法 (携帯電話、インスタントメッセージング、または電子メール) を使用してユーザーに通知し、即座に株の購入または売却を実行できるロボットプログラム
- 処理を実行するためにブラウザに入力する必要があるコードが記載された SMS メッセージを Web ユーザーに送信することにより、ユーザーを認証するアプリケーション

クライアントからサーバーへの例

- ユーザーが複数の Web サービスデータプロバイダから情報を取得できるようにするメニューベースの SMS アプリケーション。ColdFusion には、"gateways/cfc" ディレクトリにメニューベースのサンプル SMS アプリケーションが含まれています。

- ユーザーからのメッセージをテクニカルサポートに渡し、対応可能なサポートスタッフのメンバーに割り当てて送信するインスタントメッセージングアプリケーション。このアプリケーションは、ユーザー ID とセッションも記録できます。したがって、ColdFusion を使用して利用状況レポートを生成することも可能となります。
- 従業員の名前の他に電話番号またはバディ ID を含んだメッセージに応答するディレクトリ検索ロボット IM " バディ "

サーバーからサーバーへの例

- ビジネスインテリジェンスシステムで使用するステータス更新を提供する JMS サブシステム
- Web サイトからのダウンロードイベントを監視および提供するシステム

イベントゲートウェイの用語と概念

本マニュアルでイベントゲートウェイについて説明するときは、次の用語を使用します。

イベント ColdFusion が外部ソースから受け取ることができるトリガ。ColdFusion イベントゲートウェイはイベントを受け取ります。

メッセージ イベントによって提供される情報。ColdFusion におけるメッセージとは、イベントがトリガされたときにイベントゲートウェイが受け取るデータ構造体のことです。

イベントゲートウェイ イベントを受け取って、ColdFusion アプリケーションコードとの間で送受信を行う Java コード。本マニュアルでは、Coldfusion イベントゲートウェイの一般的な概念を表す場合に、イベントゲートウェイという用語を (" タイプ " や " インスタンス " という語句を付けずに) 使用します。文脈から意味が明らかである場合は、イベントゲートウェイタイプやイベントゲートウェイインスタンスのことを、単にイベントゲートウェイと呼ぶこともあります。

イベントゲートウェイタイプ Java クラスによって表される、イベントゲートウェイの特定の実装。各イベントゲートウェイタイプは、SMS、インスタントメッセージングプロトコル、ソケットなど、特定の通信方法またはプロトコルに属するメッセージを処理します。一般的に、1 つの通信プロトコルにつき 1 つのイベントゲートウェイタイプを設定します。各イベントゲートウェイタイプは、ColdFusion Administrator の [イベントゲートウェイ] 領域の [ゲートウェイタイプ] ページで設定します。

イベントゲートウェイインスタンス イベントゲートウェイタイプクラスの特定のインスタンス。イベントゲートウェイインスタンスを設定するには、[ゲートウェイインスタンス] ページで、イベントゲートウェイタイプ、ID、このインスタンスを使用するイベントゲートウェイアプリケーション CFC へのパス、および設定ファイル (選択したイベントゲートウェイタイプで必要な場合) を指定します。1 つのイベントゲートウェイタイプに対して、複数のイベントゲートウェイインスタンスを設定することができます。たとえば、複数のイベントゲートウェイアプリケーション用に、複数のインスタンスを設定できます。

イベントゲートウェイアプリケーション イベントゲートウェイインスタンスからのイベントを処理し、イベントゲートウェイインスタンスを使用してメッセージを送信する CFC およびそれをサポートする CFM ページ。イベントゲートウェイアプリケーションは、イベントゲートウェイインスタンスの一部ではなく、インスタンスとの間でやり取りされるイベントメッセージを処理するコードです。

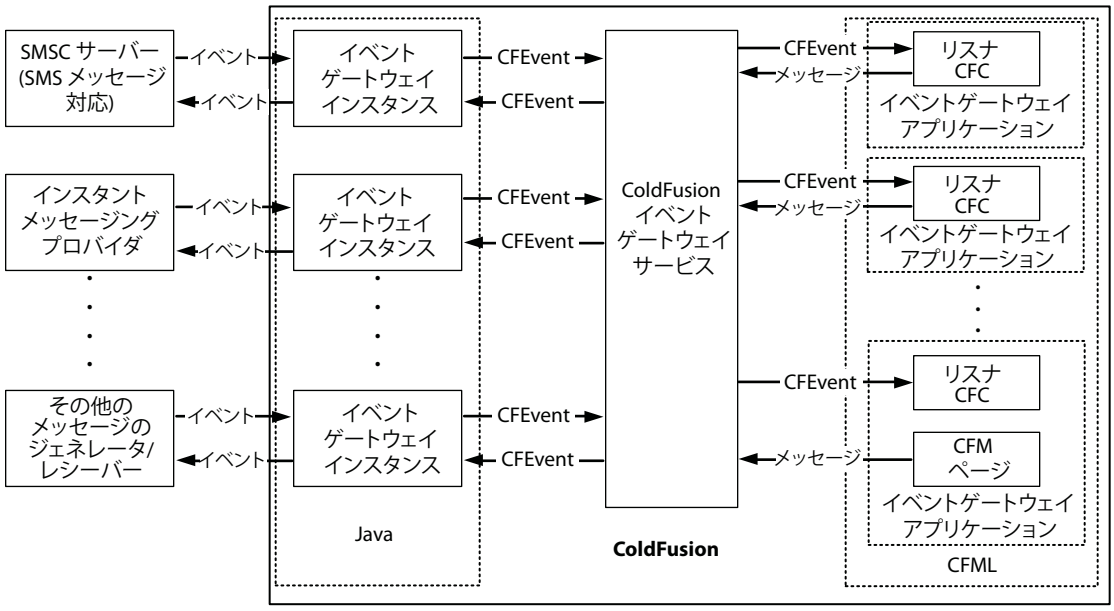
イベントゲートウェイリスナー イベントソースからイベントを受け取り、CFML のリスナー **CFC** に提供するために ColdFusion ゲートウェイサービスにそのイベントを渡す、イベントゲートウェイ内のコード。

リスナー CFC イベントゲートウェイインスタンスからのメッセージに応答するためのメソッドが用意されている CFC。イベントゲートウェイアプリケーションの一部です。

ColdFusion ゲートウェイサービス イベントゲートウェイインスタンスとリスナー **CFC** の仲介など、イベントゲートウェイに対する基本的なサポートを提供する ColdFusion の部分。

イベントゲートウェイアプリケーションの仕組み

次の図に、ColdFusion イベントゲートウェイアプリケーションのアーキテクチャを示します。



イベントゲートウェイの対話の仕組み

Java オブジェクトである ColdFusion イベントゲートウェイインスタンスは、通常、外部プロバイダから送られてくるイベントをリスンします。たとえば、一般的なソケットイベントゲートウェイは、IP ソケットでメッセージをリスンします。また SMS イベントゲートウェイは、SMSC サーバーからのメッセージを受信します。

イベントゲートウェイインスタンスは、ColdFusion イベントゲートウェイサービスを介してリスナー CFC と通信します。リスナー CFC は、メッセージを含んだ CFEvent オブジェクトインスタンスを受け取って処理し、イベントゲートウェイに応答を返します。応答を受け取ったイベントゲートウェイは、外部リソースにメッセージを送信します。

また、ColdFusion アプリケーションからメッセージを発信することもできます。それには、イベントゲートウェイにメッセージを送信する ColdFusion 関数を呼び出します。メッセージを受け取ったイベントゲートウェイは、外部リソース (インスタントメッセージングサーバーなど) にそれを転送します。アプリケーションの CFC は、送信したメッセージに対する応答をリスンします。

一方向のイベントゲートウェイも存在します。こうしたイベントゲートウェイは、特定のイベントをリスンして CFC に送信するか、または ColdFusion 関数からメッセージを取得してそれを発信します。この両方を行うことはできません。たとえば、1260 ページの「サンプルイベントゲートウェイ」で説明されている **DirectoryWatcherGateway** はイベントのリスンのみを行います。また、非同期 CFML イベントゲートウェイは CFML からのメッセージの受信のみを行います。ディレクトリウォッチャゲートウェイのように、イベントのリスンさえ行わないものもあります。このゲートウェイは、定期的にディレクトリの状態を確認することにより、内部で独自のイベントを作成しています。非同期 CFML イベントゲートウェイの詳細については、1258 ページの「CFML イベントゲートウェイによる非同期 CFC へのアクセス」を参照してください。

イベントゲートウェイの構造

ColdFusion イベントゲートウェイを開発するには、coldfusion.eventgateway.Gateway インターフェイスを実装した Java クラスを作成します。ColdFusion イベントゲートウェイは通常、ソケットや SMSC サーバーなどのイベントプロバイダからのイベントをリスンする 1 つまたは複数のスレッドで構成されます。イベントゲートウェイは、ColdFusion イベントゲートウェイサービスのメッセージキューにイベントメッセージを送信します。また、イベントゲートウェイアプリケーション CFC または CFM ページが発信メッセージを送信するときに呼び出すメソッドを提供します。

イベントゲートウェイクラスでは、次のことも行えます。

- バディリスト管理や接続管理などのイベントゲートウェイ固有のサービスを提供するヘルパクラスに、ColdFusion アプリケーションがアクセスできるようにします。
- IP アドレスとポート、パスワード、その他の ID 情報、内部タイムアウト値などの設定情報を指定するファイルを使用できます。

イベントゲートウェイアプリケーションの開発について

ColdFusion アプリケーションの開発者は、イベントゲートウェイを使用するアプリケーションを記述します。イベントゲートウェイの提供者は、ゲートウェイ固有の情報を ColdFusion 開発者に公開します。公開する情報としては、ColdFusion アプリケーションとイベントゲートウェイの間で送受信するメッセージの構造および内容や、ColdFusion アプリケーションで使用できる設定ファイルやヘルパメソッドに関する情報などがあります。

ColdFusion 開発者は、メッセージをリスンする CFC を記述します。多くのイベントゲートウェイタイプは、リスナー CFC の onIncomingMessage というメソッドにメッセージを送信します。最も単純なイベントゲートウェイアプリケーションでは、このメソッドのみを実装します。より複雑なイベントゲートウェイタイプでは、複数の CFC リスナーメソッドが必要になることもあります。たとえば、ColdFusion XMPP IM イベントゲートウェイは、ユーザーメッセージを onIncomingMessage メソッドに送信しますが、バディを追加するリクエストは onAddBuddyRequest メソッドに送信します。

イベントゲートウェイやアプリケーションのタイプによっては、メッセージを発信するための CFM ページや CFC メソッドがイベントゲートウェイアプリケーションに含まれている場合があります。また、アプリケーションがイベントゲートウェイ固有の GatewayHelper オブジェクトを使用して、IM アプリケーションのバディリストの取得や、メッセージングサーバーのステータスの取得などのタスクを実行できる場合もあります。

ColdFusion アプリケーションの開発者は、ColdFusion Administrator でイベントゲートウェイインスタンスの設定を行います。また、必要に応じて設定ファイルも使用します。ColdFusion Administrator の設定情報では、イベントゲートウェイからのメッセージを処理するリスナー CFC や、イベントゲートウェイに関するその他の標準的な設定を指定します。設定ファイルでは、必要に応じてイベントゲートウェイタイプ固有の設定情報を指定します。

イベントゲートウェイの機能とツール

ColdFusion には、イベント処理アプリケーションの開発およびデプロイに役立つ、次のような機能やツールが用意されています。

- 標準のイベントゲートウェイ
- 開発ツールとサンプルコード
- カスタムイベントゲートウェイやイベントゲートウェイアプリケーションで使用するためのゲートウェイディレクトリ構造。このディレクトリには、サンプルコードも含まれています。
- イベントゲートウェイ固有のログファイル
- イベントゲートウェイを管理するための 3 つの ColdFusion Administrator ページ

標準のイベントゲートウェイ

ColdFusion には、さまざまなイベントゲートウェイが同梱されています。それらのイベントゲートウェイでは、次のメッセージングプロトコルがサポートされています。

SMS (Short Message Service) 携帯電話やポケットベルなどのワイヤレスデバイスでショートメッセージ (主にテキスト) を交換するためのシステム。SMS イベントゲートウェイの使用方法の詳細については、1280 ページの「SMS イベントゲートウェイの使用」を参照してください。

XMPP (Extensible Messaging and Presence Protocol) インスタントメッセージングに使用される XML ベースのオープンプロトコル。XMPP イベントゲートウェイの使用方法の詳細については、1265 ページの「インスタントメッセージングイベントゲートウェイの使用」を参照してください。

IBM Lotus Instant Messaging（通常、Lotus Sametime と呼ばれる）リアルタイムのコラボレーションを実現する IBM 製品。Lotus Sametime イベントゲートウェイの使用方法の詳細については、1265 ページの「インスタントメッセージングイベントゲートウェイの使用」を参照してください。

ColdFusion には、CFML 非同期イベントゲートウェイも用意されています。CFML アプリケーションでは、このイベントゲートウェイを使用して、CFC メソッドを非同期に呼び出せます。このイベントゲートウェイで採用されているモデルは、ColdFusion の外部にあるリソースとメッセージを交換するメカニズムではなく、一方向のパスです。アプリケーションは CFC の呼び出しのみを行います。CFC からの戻り値は必要としません (実際には戻り値を受け取ることができません)。CFML 非同期イベントゲートウェイの使用方法の詳細については、1258 ページの「CFML イベントゲートウェイによる非同期 CFC へのアクセス」を参照してください。

開発ツールとサンプルコード

独自のイベントゲートウェイやイベントゲートウェイアプリケーションを開発するときは、ColdFusion に用意されている次のツールやサンプルコードが役立ちます。

- SMS クライアント (電話シミュレータ) および SMSC (Short Message Service Center) サーバーシミュレータ。外部 SMS プロバイダが利用できない場合でも、これを使用して SMS アプリケーションを開発できます。
- ソースコードを含む 4 つのサンプルイベントゲートウェイ
  - 独自のイベントゲートウェイを構築するための基本構造が含まれている空のイベントゲートウェイ用のテンプレート
  - TCP/IP ポートをリスンする TCP/IP ソケットイベントゲートウェイ
  - ディレクトリの内容に対する変更を監視するディレクトリウォッチャイベントゲートウェイ
  - JMS (Java Messaging Service) コンシューマまたはプロデューサとして機能する JMS ゲートウェイ
- 次のものを含むサンプルアプリケーション
  - 問い合わせ - 応答ドリルダウンメニューを使用して、天気予報や株価情報などのサービスを提供するメニューアプリケーション
  - 受信したメッセージをそのまま返す単純なエコーアプリケーション
  - 温度変換ツール (非同期での記録アプリケーション)
  - 従業員の電話番号などの情報を返すアプリケーション

  この章の説明では、これらのサンプルアプリケーションを使用します。
- ゲートウェイの作成に使用する Java インターフェイスやクラスの Javadoc

これらのサンプルの詳細については、1260 ページの「サンプルのイベントゲートウェイおよびゲートウェイアプリケーションの使用」を参照してください。

ColdFusion ゲートウェイディレクトリ

ColdFusion のインストールには、"<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥gateway" (J2EE 設定の場合)、または "<ColdFusion のルートディレクトリ >¥gateway" (サーバー設定の場合) というディレクトリが含まれています。このディレクトリには、ColdFusion のサンプルイベントゲートウェイおよびサンプルイベントゲートウェイアプリケーションのすべてのコードや、標準の ColdFusion イベントゲートウェイで使用するサンプル設定ファイルが含まれています。イベントゲートウェイ、イベントゲートウェイアプリケーション CFC、またはイベントゲートウェイ設定ファイルをこのディレクトリに配置する必要はありませんが、ColdFusion は、このディレクトリに配置されたイベントゲートウェイおよび CFC を検索するように設定されています。

次の表に、イベントゲートウェイディレクトリのサブディレクトリと、その用途および初期の内容を示します。サンプルイベントゲートウェイおよびアプリケーションの使用方法の詳細については、1260 ページの「サンプルのイベントゲートウェイおよびゲートウェイアプリケーションの使用」を参照してください。

ディレクトリ	用途
cfc	イベントゲートウェイアプリケーション CFC。ColdFusion をインストールすると、論理パス ¥gateway とこの "cfc" ディレクトリの間の Administrator マッピングが設定されます。
cfc¥examples	ColdFusion サンプルアプリケーションのコード。
config	すべての ColdFusion イベントゲートウェイの設定ファイル。SMS などの標準の ColdFusion イベントゲートウェイや、ディレクトリウォッチャイベントゲートウェイなどのサンプルイベントゲートウェイの設定ファイルが含まれています。
doc¥api	ゲートウェイ開発者がゲートウェイを記述するときに役立つ、Gateway インターフェイス、GatewayHelper インターフェイス、CFEvent クラス、GatewayServices クラス、GenericGateway クラスの Javadoc。この説明は、『CFML リファレンス』の Gateway development interfaces and classes に記載されている情報のサブセットです。
lib	サンプルのイベントゲートウェイクラスおよびユーザーが開発したイベントゲートウェイクラスの実行可能コード。ColdFusion クラスローダーは、このディレクトリをクラスパスに含め、そのディレクトリ内にあるすべての JAR ファイルをクラスパスに含めます。このディレクトリ内の "examples.jar" ファイルには、DirectoryWatcherGateway、EmptyGateway、および SocketGateway クラスで使用するクラスファイルが含まれています。
src¥examples	サンプルイベントゲートウェイクラスのソースコード。"EmptyGateway.java" ファイルと次のサブディレクトリが含まれています。 • socket : ソケットゲートウェイのソースファイル • watcher : ディレクトリウォッチャゲートウェイのソースファイル • JMS : JMS ゲートウェイのソースファイル

"eventgateway.log" ファイル

ColdFusion に同梱されているイベントゲートウェイは、イベントゲートウェイのエラーやイベントをログファイルに記録します。このファイルは、J2EE 設定の場合は "<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥logs¥eventgateway.log"、サーバー設定の場合は "<ColdFusion のルートディレクトリ >¥logs¥eventgateway.log" です。ColdFusion には、すべてのイベントゲートウェイがこのファイルを使用できるようにするメソッドが含まれています。このログファイルは、イベントゲートウェイやイベントゲートウェイアプリケーションをデバッグするときに役立ちます。

ColdFusion Administrator イベントゲートウェイページ

ColdFusion Administrator には、イベントゲートウェイを管理するための 3 つのページで構成される [イベントゲートウェイ] セクションがあります。

- 設定
- ゲートウェイタイプ
- イベントゲートウェイ

[設定] ページでは、イベントゲートウェイのサポートの有効化や無効化、処理中のイベントに使用できるスレッド数の指定、イベントキューで保持できるイベントの最大数の指定、および SMS テストサーバーの起動を行えます。

[ゲートウェイタイプ] ページでは、名前、Java クラス、およびスタートアップタイムアウトを指定して、イベントゲートウェイを追加、削除、および設定できます。

注意 : ColdFusion Administrator で指定したゲートウェイタイプ名が、ゲートウェイ Java コードおよび CFEvent データ構造体で使用されるゲートウェイタイプと同じである必要はありませんが、一貫性を保つため、両方とも同じ名前を使用してください。

[ゲートウェイインスタンス] ページでは、個々のイベントゲートウェイインスタンスの追加、削除、設定、起動、および停止を行えます。イベントゲートウェイインスタンスを設定するには、一意の ID、ゲートウェイタイプ、リスナー CFC のパス、設定ファイル (一部のゲートウェイタイプでは不要)、スタートアップモード (手動、自動、または無効) を指定します。

イベントゲートウェイアプリケーションの構造

イベントゲートウェイアプリケーションを開発するには、次の要素を必要に応じて作成し、使用します。

- 着信メッセージを処理し、必要な応答を送信するリスナー CFC
- 発信メッセージを直接生成する ColdFusion ページ (必要な場合)
- ColdFusion Administrator でのイベントゲートウェイインスタンスの設定。イベントゲートウェイ設定ファイルが別途必要になる場合があります。
- プロトコルやテクノロジーの追加機能 (たとえば、インスタントメッセージングのバディリストの管理機能など) へのアクセスを提供する GatewayHelper オブジェクト (必要な場合)

リスナー CFC の役割

すべての着信イベントメッセージは、リスナー CFC によって処理される必要があります。リスナー CFC の指定は、ColdFusion Administrator でイベントゲートウェイを設定するときに行います。少なくとも 1 つの CFC を Administrator で指定します。ゲートウェイタイプによっては、複数の CFC を使用できることがあります。ColdFusion イベントゲートウェイサービスがイベントを送信するときは、デフォルトで、CFC の onIncomingMessage メソッドが呼び出されます。

イベントゲートウェイの開発者は、イベントゲートウェイアプリケーションの開発者に対して、リスナー CFC が実装する必要のあるメソッド (onIncomingMessage メソッドのみの場合もあります) や、CFC が処理する必要のあるイベントメッセージデータ (CFEvent インスタンスとして渡される) の構造と内容を公開する必要があります。発信メッセージも、着信メッセージと同じイベントメッセージデータ構造を持ちます。

多くのゲートウェイでは、リスナー CFC が cfreturn 関数を呼び出して応答を送信できますが、ColdFusion は戻り値を必要としません。リスナー CFC は SendGatewayMessage 関数も使用できます。この関数は、cfreturn タグに比べて柔軟性に優れています。

ColdFusion ページの役割

ColdFusion CFM ページでは、イベントメッセージは受信できませんが、イベントゲートウェイを使用してメッセージを送信することは可能です。したがって、メッセージを発信するイベントゲートウェイアプリケーションでは、SendGatewayMessage 関数を使用してメッセージを送信できます。たとえば、パッケージが提供されたことを SMS メッセージでユーザーに通知するアプリケーションでは、SendGatewayMessage 関数を使用して通知を送信できます。

ColdFusion Administrator の役割

ColdFusion Administrator の [ゲートウェイインスタンス] ページでは、特定のイベントゲートウェイインスタンスに対して、そのイベントゲートウェイからのメッセージを処理するリスナー CFC を関連付けます。これによって、そのイベントゲートウェイで受信したメッセージが、ColdFusion イベントゲートウェイサービスによって、指定のリスナー CFC に送信されます。また、このページでは、イベントゲートウェイインスタンスの設定ファイルを指定したり、ColdFusion の起動時にそのイベントゲートウェイインスタンスを起動する (その結果、応答アプリケーションも起動する) かどうかを設定することもできます。Administrator の使用方法の詳細については、ColdFusion Administrator のオンラインヘルプを参照してください。

GatewayHelper オブジェクトの役割

ColdFusion イベントゲートウェイは、情報をやり取りするための経路を提供します。その最も基本的な機能は、イベントメッセージを送受信する機能です。ただし、場合によっては、追加機能の提供が必要になることもあります。たとえば、インスタントメッセージングイベントゲートウェイでは、バディの管理機能やステータス情報の取得機能などのサービスも提供する必要があります。イベントゲートウェイでは、GatewayHelper オブジェクトにアクセスできるようにすることで、これらの機能を提供できます。イベントゲートウェイの開発者は、必要なユーティリティルーチンを Java メソッドとして実装した Java クラスを作成し、ColdFusion アプリケーションの開発者は、CFML GetGatewayHelper メソッドを呼び出してその Java クラスのインスタンスを取得できます。アプリケーションで GatewayHelper オブジェクトのメソッドを呼び出すには、通常の ColdFusion オブジェクトと同じ方法でアクセスします。ColdFusion インスタントメッセージングイベントゲートウェイや、サンプルソケットイベントゲートウェイでは、GatewayHelper オブジェクトが提供されています。

イベントゲートウェイインスタンスの設定

イベントゲートウェイアプリケーションを開発またはデプロイする前に、ColdFusion Administrator を使用して、イベントメッセージを処理するイベントゲートウェイインスタンスを設定します。次の情報を指定します。

- イベントゲートウェイインスタンスを識別するためのイベントゲートウェイ ID。この値は、CFML の GetGatewayHelper 関数および SendGatewayMessage 関数で使用します。
- イベントゲートウェイタイプ。利用可能なイベントゲートウェイタイプ (SMS や Socket など) から選択します。
- 着信メッセージを処理するリスナー CFC の絶対パス。複数のリスナー CFC を指定する場合は、パスをカンマで区切って入力します。ColdFusion の "gateway¥cfc" ディレクトリに配置されている CFC であっても、絶対ファイルパスで指定する必要があります。
- 設定ファイル (選択したイベントゲートウェイタイプまたはインスタンスで必要な場合)
- イベントゲートウェイのスタートアップステータス。次のいずれかです。

  自動 ColdFusion の起動時にイベントゲートウェイを起動します。

  手動 ColdFusion の起動時にイベントゲートウェイを起動しません。ColdFusion Administrator に表示されるイベントゲートウェイのリストから起動することができます。

  無効 イベントゲートウェイを起動できないようにします。

イベントゲートウェイアプリケーションの開発

イベントゲートウェイアプリケーションでは、情報の処理を行います。イベントメッセージのやり取りを行うとともに、必要に応じて他のリソースとさまざまな種類の情報を交換します。イベントゲートウェイアプリケーションでは、イベントゲートウェイに送られたイベントを処理するために、リスナー CFC を用意する必要があります。イベントゲートウェイアプリケーションでは、次のコード要素も使用できます。

- SendGatewayMessage CFML 関数 : リスナー CFC の外部から (または、オプションで CFC から) メッセージを送信できます。
- GatewayHelper オブジェクト
- eventgateway ログファイル

イベントゲートウェイアプリケーションのモデル

イベントゲートウェイアプリケーションは、次に示すモデルの一方または両方の役割を持ちます。

応答アプリケーション 外部ソースからイベントメッセージを受け取ると、ColdFusion リスナー CFC によって応答が開始されます。

発信アプリケーション ColdFusion アプリケーションが、イベントゲートウェイを使用して送信するイベントメッセージを生成します。

通常の ColdFusion アプリケーションと異なり、応答アプリケーションはリクエストフリーです。応答アプリケーションは CFM ページを持たず、CFC のみを持ちます。また、HTTP リクエストに応答しません。ColdFusion イベントゲートウェイサービスからリスナー CFC にイベントメッセージが直接送信され、CFC リスナーメソッドからイベントゲートウェイサービスに応答が直接返されます。携帯電話のユーザーにニュースを提供したり、テキストメッセージを表示したり、その他の情報を提示したりするアプリケーションは、このモデルに該当します。

発信アプリケーションは、ほとんどの ColdFusion アプリケーションに似ています。特定の時点で、リクエストに応答して ColdFusion で CFM ページが実行されます。リクエストを発信するには、ColdFusion Administrator の [スケジュールされたタスク] ページを使用します。アプリケーションが CFML の SendGatewayMessage 関数を呼び出すと、ColdFusion によってイベントゲートウェイにメッセージが送信されます。注文商品が出荷されたことを SMS で顧客に通知するアプリケーションは、このモデルに該当します。

イベントゲートウェイへの情報の送信

ColdFusion アプリケーションからイベントゲートウェイにメッセージを送信するには、次のいずれかの方法を使用します。

- リスナー CFC のリスナーメソッドの中で cfreturn タグを使用する
- ColdFusion の SendGatewayMessage 関数を呼び出す

最初の方法は、着信メッセージに自動応答する場合に便利です。着信メッセージに応答する複雑なアプリケーションでは、戻り値の代わりに、または戻り値に加えて、SendGatewayMessage 関数を使用することもできます。

イベントゲートウェイタイプによっては、GatewayHelper オブジェクトを使用して外部リソースに情報を送信することもあります。たとえば、ColdFusion XMPP および Lotus Sametime インスタントメッセージングイベントゲートウェイで提供されている GatewayHelper オブジェクトを使用すれば、バディリストを管理したり、インスタントメッセージングサーバーの設定情報やステータス情報を指定することができます。GatewayHelper オブジェクトの詳細については、1256 ページの「GatewayHelper オブジェクトの使用」を参照してください。インスタントメッセージングの GatewayHelper オブジェクトの詳細については、1270 ページの「IM メッセージ処理アプリケーションのサンプル」を参照してください。

1254 ページの「イベントゲートウェイ CFC の例」のサンプルコードでは、リスナーの戻り値の使用方法や、イベントゲートウェイに応じて必要なデータを構造体に設定してメッセージとして返す方法が示されています。

イベントゲートウェイリスナー CFC の開発

リスナー CFC は、イベントゲートウェイのメッセージに応答します。リスナー CFC では、次に示す基本的なソフトウェア要素が必ず使用されます。

- リスナーメソッド
- メッセージを含む CFEvent 構造体

リスナー CFC では、ColdFusion の永続スコープを使用することで、複数の CFC 呼び出しにまたがって使用するデータや、他の CFML 要素と共有するデータを保存できます。

リスナーメソッド

ColdFusion イベントゲートウェイサービスは、着信メッセージを処理するために、CFC のリスナーメソッドを呼び出します。実装する必要があるリスナーメソッドの数と名前は、イベントゲートウェイによって異なります。たとえば、ColdFusion の SMS イベントゲートウェイでは、一般に onIncomingMessage という単一のリスナーメソッドが必要になります。SMS イベントゲートウェイリスナーメソッドの名前は、イベントゲートウェイの設定ファイルで変更できます。ColdFusion の XMPP IM イベントゲートウェイでは、onIncomingMessage、onAddBuddyRequest、onAddBuddyResponse、onBuddyStatus、onIMServerMessage の 5 つのメソッドをリスナー CFC に実装する必要があります。イベントゲートウェイ

でメソッド名が指定されていない場合は、デフォルトでリスナー CFC の onIncomingMessage メソッドが呼び出されます。一貫性を保つため、単一のリスナーメソッドを使用するイベントゲートウェイでは、常に onIncomingMessage メソッドを使用することをお勧めします。

リスナーメソッドでは、次の処理を行います。

1 単一のパラメータである CFEvent 構造体を取ります。

2 必要に応じてインスタンスの内容を処理します。

3 オプションで、cfreturn タグを使用してイベントゲートウェイに発信メッセージを返します。また、ColdFusion の SendGatewayMessage 関数を呼び出すことで、イベントゲートウェイにメッセージを返すこともできます。

次のコードは、受信したメッセージをそのまま Socket イベントゲートウェイに返す onIncomingMessage メソッドが実装されているリスナー CFC です。この CFC には、着信メッセージを処理し、ソケットゲートウェイを使用して送信元に応答するために必要な最低限のコードしか含まれていません。

```
<cfcomponent displayname="echo" hint="echo messages from the event gateway">
    <cffunction name="onIncomingMessage" output="no">
        <cfargument name="CFEvent" type="struct" required="yes">
        <!--- Create a return structure that contains the message. --->
        <cfset retValue = structNew()>
        <cfset retValue.DestinationID = arguments.CFEvent.OriginatorID>
        <cfset retValue.MESSAGE = "Echo: " & arguments.CFEvent.Data.MESSAGE>
        <!--- Send the return message back. --->
        <cfreturn retValue>
    </cffunction>
</cfcomponent>
```

戻り構造体で必要なフィールドは、イベントゲートウェイによって異なります。たとえば、SMS イベントゲートウェイを使用してメッセージをエコーするには、次の行を使用して戻り値を指定します。

```
<cfset retValue.command = "submit">
<cfset retValue.sourceAddress = arguments.CFEVENT.gatewayid>
<cfset retValue.destAddress = arguments.CFEVENT.originatorid>
<cfset retValue.ShortMessage = "Echo: " & arguments.CFEvent.Data.MESSAGE>
```

CFEvent 構造体

ColdFusion イベントゲートウェイサービスは、メッセージイベントに関する情報を含む CFEvent 構造体をリスナーメソッドに渡します。次の表で、この構造体のフィールドについて説明します。

フィールド	説明
GatewayID	イベントを送信したイベントゲートウェイ。この値には、ColdFusion Administrator の [ゲートウェイインスタンス] ページで設定したイベントゲートウェイインスタンスの ID が使用されます。アプリケーションで SendGatewayMessage 関数を呼び出してイベントゲートウェイに応答する場合は、この ID を関数の最初のパラメータとして使用します。
Data	メッセージなどのイベントデータを含む構造体。Data 構造体の内容は、イベントゲートウェイタイプによって異なります。
OriginatorID	メッセージの発信元。この値は、プロトコルやイベントゲートウェイタイプによって異なります。多くのイベントゲートウェイでは、応答の宛先を識別するために応答メッセージでこの値を必要とします。メッセージの送信者を識別します。
GatewayType	SMS などの、イベントゲートウェイのタイプ。複数のイベントゲートウェイタイプからのメッセージを処理するアプリケーションでは、このフィールドを使用できます。この値は、イベント Gateway クラスによって指定されるゲートウェイタイプ名です。ColdFusion Administrator のゲートウェイタイプ名と一致しているとは限りません。

フィールド	説明
CFCPath	リスナー CFC の場所です。リスナー CFC はこのフィールドを使用する必要はありません。
CFCMethod	ColdFusion がイベントを処理するために呼び出すリスナーメソッドです。リスナー CFC はこのフィールドを使用する必要はありません。
CFCTimeout	リスナー CFC がイベントリクエストを処理するためのタイムアウト時間 (秒単位)。リスナー CFC はこのフィールドを使用する必要はありません。

ColdFusion アプリケーションがイベントゲートウェイメッセージに応答するときや、メッセージを発信するときは、CFEvent 構造体を使用しません。ただし、ColdFusion イベントゲートウェイサービスは、イベントゲートウェイの outgoingMessage メソッドを呼び出す前に、メッセージデータが設定された Java CFEvent インスタンスを作成します。

リスナー CFC での永続スコープの使用

ColdFusion リスナー CFC では、Application、Client、および Session 永続スコープを使用できます。

着信イベントゲートウェイメッセージには HTTP リクエストが関連付けられていないので、このようなイベントによって開始されるやり取りには、次に示すように、CFM ページをリクエストしたときとは異なるセッション ID やクライアント ID が使用されます。

識別子	構造体
セッション ID	**gatewayType_gatewayID_originatorID**
cfid	**originatorID**
cftoken	**gatewayType_gatewayID**

gatewayID の値は、ColdFusion Administrator で設定したイベントゲートウェイ ID です。**gatewayType** と **originatorID** は、着信メッセージに対してイベントゲートウェイが CFEvent インスタンスに設定した値です。

Application スコープ

Application スコープでは、同じアプリケーション名を使用しているすべての ColdFusion ページまたは CFC の間でデータが共有されます。リスナー CFC で Application スコープを使用すれば、メッセージの送信を行う CFML ページとデータを共有できます。また、複数のリスナー CFC を単一のディレクトリに配置することで、"Application.cfc" または "Application.cfm" ファイルとアプリケーション名をそれらのリスナー CFC で共有することができます。

アプリケーション名を設定するには、通常の ColdFusion コードと同様に、"Application.cfc" の This.name 変数または cfapplication タグを使用します。リスナー CFC で "Application.cfc" または "Application.cfm" ファイルを使用できるのは、CFC が次のディレクトリまたはそのサブディレクトリに配置されている場合です。

- ColdFusion の Web ルート
- ColdFusion Administrator の [マッピング] リストに含まれているディレクトリ

ColdFusion をインストールすると、"gateway¥cfc" ディレクトリに対する ColdFusion Administrator マッピングが自動的に作成されます。

Client スコープ

Client スコープには、メッセージ送信者の ID に関連付けられる、長期的な情報を格納できます。たとえば、特定の IM バディに関する情報を格納できます。

複数の接続にまたがって Client 変数を使用するには、特定のクライアントとのすべての通信で、ゲートウェイタイプが同じクライアント ID を使用する必要があります。これは、IM や SMS ゲートウェイなどの多くのテクノロジーやゲートウェイでは問題になりません。

注意： ゲートウェイで Client スコープ変数を使用するには、Client スコープ変数をデータソースまたはレジストリに格納する必要があります。ゲートウェイが Cookie を使用することはないので、この変数を Cookie に保存することはできません。

Session スコープ

Session スコープには、複数のやり取りにまたがって必要な情報を格納できます。たとえば、ドリルダウンメニューを使用してサービスを選択する対話型の IM アプリケーションや SMS アプリケーションでは、メニューの選択に関する情報を Session スコープに格納できます。

イベントゲートウェイのセッションは、タイムアウトになると終了します。イベントのセッション識別子やクライアント識別子は、リクエストのセッション識別子やクライアント識別子とは異なります。したがって、メッセージを発信する標準の CFM ページと、そのメッセージの着信応答を処理するリスナー CFC の間で、同じ Session スコープや Client スコープを使用することはできません。

Session スコープの使用例については、"gateway¥cfc¥examples¥menu" ディレクトリにあるサンプルメニューアプリケーションを参照してください。

注意: 発信アプリケーションで SendGatewayMessage メソッドを使用してクライアント (SMS ユーザーなど) との対話を開始した場合は、セッションが作成されません。この場合は、送信したメッセージとその宛先を (データベースなどを使用して) 追跡する必要があります。応答イベントを受け取ったら、originatorID を調べて、発信メッセージに対する応答であるかどうかを判断します。

イベントゲートウェイ CFC のデバッグ

イベントゲートウェイ CFC でイベントに応答するときは、CFM ページの場合とは異なり、デバッグ情報を応答ページに表示することはできません。したがって、cfdump タグなどの一般的な ColdFusion デバッグテクニックの多くは使用できません。イベントゲートウェイ CFC を開発するときは、次のデバッグテクニックを検討してください。

- Application スコープにトレース変数を設定します。この変数は永続的であり、CFC のアプリケーション名を指定することができます (1253 ページの「Application スコープ」を参照)。トレース変数などの Application スコープの内容は、CFC と同じアプリケーション名を使用しているすべての CFML ページから参照できます。
- cflog タグを使用して、重要なイベントをファイルに記録し、エラーを追跡します。また、ColdFusion で管理されている "eventgateway.log" ファイルや "exceptions.log" ファイルも調べます。"eventgateway.log" ファイルの使用方法の詳細については、1248 ページの「"eventgateway.log" ファイル」を参照してください。
- CFM ページで SendGatewayMessage 関数を使用すれば、CFC からイベントゲートウェイへの応答をシミュレートできます。この関数の **message** パラメータには、CFC が戻り変数に設定する情報を指定します。
- コマンドラインから ColdFusion を実行する場合は、次のコードのように Java の System.out.println メソッドを使用して、コンソールウィンドウにメッセージを出力できます。

```
<cfscript>
    sys = createObject("java", "java.lang.System");
    sys.out.println("Debugging message goes here");
</cfscript>
```

注意： CFC を変更しても、イベントゲートウェイインスタンスを再起動する必要はありません。次のイベントが発生したときに、更新された CFC が自動的に使用されます。

イベントゲートウェイ CFC の例

次のコードは、複数のイベントゲートウェイ (SMS、XMPP、Lotus Sametime、およびサンプル Socket イベントゲートウェイ) に対応可能な温度単位変換ツールです。ユーザーは、温度単位 (F、Fahrenheit、C、Celsius)、カンマ、および温度で構成された文字列をデバイスに入力します。CFC は摂氏を華氏に、または華氏を摂氏に変換し、結果を返します。

この例では、応答イベントゲートウェイアプリケーションの動作の仕組みと、イベントゲートウェイタイプによって発信メッセージの形式が異なることが示されています。

```
<cfcomponent displayname="tempconverter" hint="Convert temperatures between
    Celsius and Fahrenheit">

<cffunction name="onIncomingMessage" output="no">
    <cfargument name="CFEvent" type="struct" required="yes">
    <!--- Standard error message giving the correct input format. --->
    <cfset var errormsg = "Please enter scale, integer where scale is F or C,
        for example:F, 32">

    <!--- Get the message. --->
    <cfset data=cfevent.DATA>
    <cfset message="#data.message#">
    <!--- Where did it come from? --->
    <cfset orig="#CFEvent.originatorID#">

    <!--- Process the input, generate a message with the new temperature. --->
    <!--- Input format is: degrees, temperature. --->
    <cfif listlen(message) eq 2>
        <cfif (listgetat(message,1) IS "F") OR
                (listgetat(message,1) IS "Fahrenheit") OR
                (listgetat(message,1) IS "C") OR
                (listgetat(message,1) IS "Celsius")>
            <cfset scale=listgetat(message,1)>
            <cfif isNumeric(listgetat(message,2))>
                <cfset temperature=listgetat(message,2)>
                <cfswitch expression="#scale#">
                    <cfcase value="F, Fahrenheit">
                        <cfset retmsg = temperature & " degrees Fahrenheit is "
                            & (temperature-32.0) * (5.0/9.0) & " degrees Celsius">
                    </cfcase>
                    <cfcase value="C, Celsius">
                        <cfset retmsg = temperature & " degrees Celsius is "
                            &(temperature * 10.0/5.0) + 32 & " degrees Fahrenheit">
                    </cfcase>
                </cfswitch>
            <cfelse>
                <cfset retmsg=errormsg>
            </cfif>
        <cfelse>
            <cfset retmsg=errormsg>
        </cfif>
    <cfelse>
        <cfset retmsg=errormsg>
    </cfif>

    <!--- Fill the return value as required for the event gateway type. --->
    <cfif arguments.CFEVENT.GatewayType is "Socket">
        <cfset retValue = structNew()>
        <cfset retValue.MESSAGE = retmsg>
```

```
        <cfset retValue.originatorID = orig>
    <cfelseif (arguments.CFEVENT.GatewayType is "Sametime") OR
             (arguments.CFEVENT.GatewayType is "XMPP")>
        <cfset retValue = structNew()>
        <cfset retValue.MESSAGE = retmsg>
        <cfset retValue.BuddyID = arguments.CFEVENT.DATA.SENDER>
        <cfset retValue.originatorID = orig>
    <cfelseif arguments.CFEVENT.GatewayType is "SMS">
        <cfset retValue = structNew()>
        <cfset retValue.command = "submit">
        <cfset retValue.sourceAddress = arguments.CFEVENT.gatewayid>
        <cfset retValue.destAddress = arguments.CFEVENT.originatorid>
        <cfset retValue.shortMessage = retmsg>
    </cfif>

    <!--- Send the return message back. --->
    <cfreturn retValue>

</cffunction>
</cfcomponent>
```

SendGatewayMessage 関数によるメッセージの送信

SendGatewayMessage 関数の形式は次のとおりです。

```
SendGatewayMessage(gatewayID, messageStruct)
```

- **gatewayID** パラメータには、ColdFusion Administrator で設定した、メッセージの送信先となるイベントゲートウェイインスタンスのゲートウェイ ID を指定する必要があります。
- **messageStruct** パラメータは構造体です。その内容は、イベントゲートウェイの outgoingMessage メソッドの要件によって異なります。また、受信アプリケーションによって異なる場合もあります。たとえば、メッセージの他に、宛先の識別子も指定する場合があります。

イベントゲートウェイに渡される CFEvent インスタンスでは、GatewayID フィールドと Data フィールドにこの 2 つのパラメータが含まれています。残りのフィールドは空です。

次のサンプルコードでは、ファイルに情報を記録する CFC にメッセージを送信しています。SendGatewayMessage 関数が "OK" を返した場合は、メッセージを表示します。このコードでは、Asynch Logger という非同期 CFML イベントゲートウェイのインスタンスを使用しています。**messageStruct** パラメータで使用されている props 変数には、宛先ファイルおよび記録するメッセージの 2 つのエントリが含まれています。

```
<cfscript>
    status = "No";
    props = structNew();
    props.Message = "Replace me with a variable with data to log";
    status = SendGatewayMessage("Asynch Logger", props);
    if (status IS "OK") WriteOutput("Event Message ""#props.Message#"" has been sent.");
</cfscript>
```

注意：情報を記録する CFC のコードについては、1258 ページの「CFML イベントゲートウェイによる非同期 CFC へのアクセス」を参照してください。

GatewayHelper オブジェクトの使用

ColdFusion の GetGatewayHelper 関数を呼び出すと、イベントゲートウェイ固有のヘルパメソッドおよびプロパティを備えた Java の GatewayHelper オブジェクトが作成および初期化されます。この関数を使用するには、イベントゲートウェイが GatewayHelper クラスを実装している必要があります。たとえば、インスタントメッセージングイベントゲートウェイの GatewayHelper オブジェクトには、バディリストを管理するためのメソッドを用意できます。

ColdFusion の GetGatewayHelper 関数は、ヘルパを持つイベントゲートウェイインスタンスの ID を唯一のパラメータとして取り、Java の GatewayHelper オブジェクトを返します。パラメータ値には、ColdFusion Administrator で設定したインスタンスゲートウェイ ID を使用する必要があります。ID 値をアプリケーションでハードコードしない場合 (たとえば、リスナー CFC が複数のイベントゲートウェイインスタンスに対応している場合) は、最初の着信メッセージの CFEvent 構造体からゲートウェイ ID を取得します。

GatewayHelper オブジェクトのメソッドやプロパティにアクセスするには、ColdFusion での Java オブジェクトへの標準的なアクセス方法を使用します (1109 ページの「CFML アプリケーションへの J2EE および Java 要素の統合」を参照)。たとえば、イベントゲートウェイの GatewayHelper クラスに、単一の String パラメータを取る addBuddy メソッドがある場合は、次のコードを使用して ColdFusion XMPP または Sametime ゲートウェイの GatewayHelper オブジェクトを取得し、バディをバディリストに追加することができます。

```
<cfscript>
    myHelper = GetGatewayHelper(myGatewayID);
    status = myHelper.addBuddy("jsmith23", "Jim Smith", "support");
</cfscript>
```

イベントゲートウェイエラーログファイルの使用

標準の ColdFusion イベントゲートウェイで、処理の続行を妨げないエラーが発生した場合は、ColdFusion ログディレクトリにある "eventgateway.log" ファイルにこのエラーが記録されます。他のイベントゲートウェイも、このファイルや、ログディレクトリ内のアプリケーション固有のファイルに情報を記録できます。

標準の ColdFusion イベントゲートウェイは、メッセージングサーバーとのやり取りにおけるエラー、ColdFusion アプリケーションによって送信されたメッセージのエラー、およびイベントゲートウェイの処理中に発生した修復可能なエラーを記録します。また、イベントゲートウェイの初期化や再起動など、重要な通常イベントに関する情報ステータスメッセージも記録します。

"eventgateway.log" ファイルに記録される ColdFusion イベントゲートウェイメッセージの形式は、一般に次のようになります。

```
gatewayType (gatewayID) message body
```

イベントゲートウェイアプリケーションを開発するときは、ColdFusion ログビューアを使用して "eventgateway.log" ファイルを確認できます。ログビューアでは、ゲートウェイタイプやゲートウェイ ID をキーワードに使用して、表示をフィルタ処理できます。異なる厳格度を選択することで、アプリケーションおよびイベントゲートウェイの処理におけるエラーや、起こりうる不具合などを確認することができます。

イベントゲートウェイとアプリケーションのデプロイ

ColdFusion サーバーにイベントゲートウェイアプリケーションをデプロイするには、リスナー CFC をインストールして、この CFC を使用するゲートウェイインスタンスを設定します。

イベントゲートウェイアプリケーションのデプロイ

1. 必要なイベントゲートウェイタイプが ColdFusion Administrator で設定されていることを確認します。設定されていない場合は、イベントゲートウェイをデプロイします (1321 ページの「イベントゲートウェイのデプロイ」を参照)。
2. イベントゲートウェイタイプが設定ファイルを必要とする場合は、"gateway¥config" ディレクトリに有効なファイルが存在することを確認します。設定ファイルは、複数のイベントゲートウェイインスタンスで共有している場合もあれば、イベントゲートウェイインスタンスごとに個別のファイルが必要な場合もあります。
3. イベントゲートウェイアプリケーションのリスナー CFC および他のアプリケーションコンポーネントをインストールします。ColdFusion では、CFC を配置する場所として、"<ColdFusion のルートディレクトリ >¥gateways¥cfc" ディレクトリが用意されており、ColdFusion Administrator のページでこのディレクトリに対するマッピングが設定されています。ただし、これ以外のディレクトリにリスナー CFC をインストールすることも可能です。

4 ColdFusion Administrator の [イベントゲートウェイ] セクションの [ゲートウェイインスタンス] ページで、イベントゲートウェイインスタンスを設定します (1250 ページの「イベントゲートウェイインスタンスの設定」を参照)。

CFML イベントゲートウェイによる非同期 CFC へのアクセス

ColdFusion の CFML イベントゲートウェイを使用すれば、CFML コードから CFC メソッドに非同期でメッセージを送信できます。したがって、CFC メソッドが完了したり、値を返したりするのを待つことなく、処理を続行することができます。このイベントゲートウェイを使用してアクセスする非同期 CFC では、次のようなことを行います。

- アプリケーションの処理を遅延させることなく、新しい情報を使用して Solr コレクションのインデックスを再作成する (例えば、ユーザーが新しいファイルをアップロードしたときなど)
- 情報を記録する (特に、大量のデータを記録する必要がある場合)
- 完了までに長時間を要する可能性があるバッチ処理を実行する

非同期 CFC は、リクエストから独立した状態で実行されるので、ユーザーに対してフィードバックを提供しません。すべての結果やエラー情報は、ファイル、データソース、または他の外部リソースに保存します。

ColdFusion はデフォルトで、onIncomingMessage という CFC メソッドにメッセージを送信します。ただし、SendGatewayMessage メソッドの **data** パラメータで任意のメソッド名を指定できます。

CFML イベントゲートウェイのデータ構造体

CFML の SendGatewayMessage 関数で使用する構造体には、次の 2 種類のフィールドを設定できます。

- CFC で使用される任意の内容を含むフィールドをいくつでも設定できます。
- いくつかのオプションフィールドで、ゲートウェイから CFC への情報の送信方法を設定できます。

CFML ゲートウェイは次のオプションフィールドを検索し、存在する場合はそのフィールドを使用して、メッセージの送信方法を決定します。CFC メソッドに送信するデータに対して、これらのフィールド名を使用しないでください。

フィールド	用途
cfcpath	ColdFusion Administrator で指定されている CFC パスを上書きします。このフィールドを使用すれば、複数の CFC に対して、ColdFusion Administrator で単一のゲートウェイ設定を使用することができます。
method	CFC で呼び出すメソッド名を設定します。デフォルト値は onIncomingMessage です。このフィールドを使用すれば、複数のメソッドを持つ CFC に対して、ColdFusion Administrator で単一のゲートウェイ設定を使用することができます。
originatorID	ColdFusion から CFC に送信する CFEvent オブジェクトの originatorID フィールドを設定します。デフォルト値は CFMLGateway です。
timeout	タイムアウトの値を秒単位で設定します。ここで設定した時間内に、リスナー CFC がイベントリクエストを処理して結果を返さなかった場合、ColdFusion ゲートウェイサービスはリクエストを終了します。デフォルト値は、ColdFusion Administrator の [サーバーの設定] ページで設定したリクエストタイムアウトの値です。リクエストの処理に非常に時間がかかる場合など、デフォルトのタイムアウト値よりもリクエストの処理に時間がかかる場合に、この値を設定します。

CFML ゲートウェイの使用

単一の onIncomingMessage メソッドを持つ非同期 CFC の使用方法を、次の手順に示します。

非同期 CFC の使用

1 onIncomingMessage メソッドを持つ CFC を作成します。アプリケーションの適切なディレクトリに CFC を配置します。たとえば、J2EE 設定の場合は、"<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥gateway¥cfc" ディレクトリまたはそのサブディレクトリに CFC を配置します。サーバー設定の場合は、"<ColdFusion のルートディレクトリ

>¥gateway¥cfc" ディレクトリまたはそのサブディレクトリに CFC を配置します。これらの cfc ゲートウェイディレクトリへのマッピングは、ColdFusion のインストール時に自動的に設定されます。

onIncomingMessage メソッドは、Data フィールドに入力情報が含まれている CFEvent 構造体を取り、必要に応じて Data フィールドの内容を処理します。

2 ColdFusion Administrator の [ゲートウェイインスタンス] ページを使用して、CFML イベントゲートウェイタイプのインスタンスを追加します。次の情報を指定します。

- 一意のゲートウェイ ID。
- 手順 1 で作成した CFC へのパス。
- スタートアップモード。ColdFusion が起動したときにイベントゲートウェイを起動させるには、[自動] を選択します。
- 設定ファイルは指定しないでください。

3 イベントゲートウェイインスタンスを起動します。

4 手順 2 で指定したイベントゲートウェイインスタンス ID に構造体のメッセージを送信する SendGatewayMessage 関数を、CFML コードで記述します。SendGatewayMessage 関数は、ゲートウェイによってメッセージが ColdFusion ゲートウェイサービスのキューに正常に追加された場合は true を返し、それ以外の場合は false を返します。この関数は、CFC がメッセージを受信することや処理することは保証しません。

5 CFML アプリケーションを実行します。

例 : メッセージのロギング

次の非同期 CFML イベントゲートウェイ CFC は、cflog タグを使用して、ColdFusion ログディレクトリ内のファイルにメッセージを記録します。この CFC には、次のフィールドを持つメッセージを渡します。

- file : メッセージを記録するファイルの名前。デフォルト値は defaultEventLog です。
- type : 使用する cflog タイプ属性。デフォルト値は info です。
- message : メッセージテキスト。

```
<cfcomponent>
    <cffunction name="onIncomingMessage" output="no">
        <cfargument name="CFEvent" type="struct" required="yes">
        <cfscript>
            if (NOT IsDefined("CFEvent.Data.file")) {
                    CFEvent.Data.file="defaultEventLog"; }
            if (NOT IsDefined("CFEvent.Data.type")) {
                CFEvent.Data.type="info"; }
        </cfscript>
        <cflog text="#CFEvent.Data.message#"
                file="#CFEvent.Data.file#"
            type="#CFEvent.Data.type#"
            thread="yes"
            date="yes"
            time="yes"
            application="yes">
    </cffunction>
</cfcomponent>
```

このイベントゲートウェイは、次に示す最小構成の CFML ページでテストできます。

```
Sending an event to the CFML event gateway that is registered in the
    ColdFusion Administrator as Asynch Logger.<br>
<cfscript>
    status = false;
    props = structNew();
    props.Message = "Replace me with a variable with data to log";
    status = SendGatewayMessage("Asynch Logger", props);
    if (status IS True) WriteOutput('Event Message "#props.Message#" has been sent.');
</cfscript>
```

サンプルのイベントゲートウェイおよびゲートウェイアプリケーションの使用

ColdFusion には、いくつかのサンプルイベントゲートウェイおよびアプリケーションが同梱されています。J2EE 設定の場合は "<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥gateway" ディレクトリに、サーバー設定の場合は "<ColdFusion のルートディレクトリ >¥gateway" ディレクトリに格納されています。これらのゲートウェイのサンプルコードは、独自のゲートウェイを開発するときに参考にしたり、流用したりすることができます。これらのゲートウェイはサンプルとしてのみ提供されており、完全な製品品質を実装するものではありません。

サンプルイベントゲートウェイ

"gateway¥src¥examples" ディレクトリおよびそのサブディレクトリには、サンプルイベントゲートウェイのソースが含まれています。コンパイル済みのバージョンは、"gateway¥lib¥examples.jar" ファイルに収められています。本マニュアルでは、これらのイベントゲートウェイおよびその機能について簡単に説明します。詳細については、ファイル内のコードおよびコメントを参照してください。

EmptyGateway

"EmptyGateway.java" ファイルには、独自のイベントゲートウェイを作成するときに使用可能なイベントゲートウェイテンプレートが含まれています。このクラスや、新しいイベントゲートウェイの作成方法の詳細については、1308 ページの「カスタムイベントゲートウェイの作成」を参照してください。

SocketGateway

SocketGateway イベントゲートウェイは TCP/IP ポートをリスンします。したがって、Telnet などの TCP/IP ベースのプロトコルを使用してメッセージを送受信するアプリケーションや、ソケット間でメッセージを送信するアプリケーションに対してこのゲートウェイを使用できます。例として、Telnet プロトコルをフルサポートせずに、Telnet 端末クライアントからのメッセージに応答する単純なゲートウェイアプリケーションが挙げられます。

注意：ColdFusion Administrator では、SocketGateway クラスのゲートウェイタイプ名として Socket が使用されます。

"SocketGateway.java" ファイルには、イベントゲートウェイである SocketGateway と、GatewayHelper クラスである SocketHelper の 2 つのクラスが定義されています。ソースファイルは "gateway¥src¥examples¥socket" ディレクトリにあります。

SocketGateway　TCP/IP ポートをリスンします。このイベントゲートウェイはマルチスレッドに対応しており、複数のクライアントを同時に処理できます。既存のクライアントに発信メッセージを送信できますが、リンク自体を確立することはできません。

SocketGateway クラスはデフォルトでポート 4445 をリスンします。このポート番号は設定ファイルで指定できます。設定ファイルでは、次に示す形式の単一の行を設定する必要があります。

```
port=portNumber
```

SocketHelper　次のメソッドを持つ GatewayHelper クラス

getSocketIDs() : 開いているすべての Java ソケットのソケット ID を含む配列を返します。イベントゲートウェイは、各リモートクライアントのソケットを開きます。

killSocket(String **socketid**) : 指定したソケットを削除します。成功かどうかを示すブール値のインジケータを返します。

DirectoryWatcherGateway

DirectoryWatcherGateway イベントゲートウェイは、ディレクトリ内でファイルが作成、削除、または編集されたときに、リスナー CFC にイベントを送信します。ウォッチャはスレッドで実行され、設定ファイルで指定された期間スリープ状態になります。その期間が経過すると、前回実行時からの変更点が確認されます。ウォッチャは、ファイルの追加、削除、変更を検出すると、リスナー CFC にメッセージを送信します。追加、削除、変更の各イベントに対して個別の CFC を設定したり、すべてのイベントに対して 1 つの CFC を使用したりすることができます。このイベントゲートウェイのソースは、"gateway/src/examples/watcher" ディレクトリにあります。

注意：ColdFusion Administrator では、DirectoryWatcherGateway クラスのゲートウェイタイプ名として DirectoryWatcher が使用されます。

設定ファイル

このイベントゲートウェイには、次の形式の行が構成された設定ファイルが必要です。

```
directory=C:/temp
```

注意：Windows ファイルパスでディレクトリの区切り文字として円記号 (¥) を使用する場合は、C:¥¥temp のように、円記号を 2 つ並べてエスケープする必要があります。Windows を含むすべてのオペレーティングシステムで、スラッシュ (/) をディレクトリの区切り文字として使用できます。

注意：Directory Watcher の設定ファイルでファイル名拡張子を指定する際にはピリオドを含めず、"***doc,txt***" のようにカンマを使用します。

設定ファイルでシャープ記号 (#) を前に配置することにより、コメント行を設定することができます。プロパティを省略したり、コメントアウトしたりした場合は、デフォルト値が使用されます。値のないプロパティを指定した場合、空のプロパティが設定されます。設定ファイルには次の値を定義できます。

プロパティ	必須 / オプション	説明
directory	必須	監視するディレクトリのパス
recurse	オプション	サブディレクトリを確認するかどうか。デフォルト値は no です。
extensions	オプション	監視する拡張子のカンマ区切りのリスト。イベントゲートウェイは、この拡張子が付いたファイルの変更のみを記録します。アスタリスク (*) はすべてのファイルを表します。デフォルト値はすべてのファイルです。
interval	オプション	イベントゲートウェイがディレクトリを確認する間隔 (ミリ秒単位)。デフォルト値は 60 秒です。
addFunction	オプション	ファイルが追加されたときに呼び出す関数の名前。デフォルト値は onAdd です。
changeFunction	オプション	ファイルが変更されたときに呼び出す関数の名前。デフォルト値は onChange です。
deleteFunction	オプション	ファイルが削除されたときに呼び出す関数の名前。デフォルト値は onDelete です。

サンプルの設定ファイルは、"gateway¥config¥directory-watcher.cfg" ファイルにあります。

CFC メソッド

ディレクトリの内容が変更されると、設定ファイルで名前を変更していない限り、イベントゲートウェイは次のいずれかの CFC リスナーメソッドを呼び出します。

- onAdd
- onChange
- onDelete

リスナーメソッドに送信される CFEvent.Data フィールドには、次のフィールドが含まれています。

フィールド	説明
TYPE	イベントのタイプ。ADD、CHANGE、DELETE のいずれか。
FILENAME	システムディレクトリルートから、追加、削除、または変更されたファイルへの絶対パス。
LASTMODIFIED	ファイルが作成または変更された日時。ファイルが削除された場合は、このフィールドは含まれません。

このイベントゲートウェイは、複数のリスナー CFC をサポートしており、イベントメッセージをすべてのリスナーに送信します。このイベントゲートウェイは一方向です。イベントを監視して、イベント情報を CFC に送信しますが、CFC からの戻り値や SendGatewayMessage 関数からの入力は受け付けません。

ディレクトリウォッチャは、エラーを ColdFusion ログディレクトリ内の "watcher.log" ファイルに記録します。

ディレクトリウォッチャアプリケーションの例

次のコードは、単純なディレクトリウォッチャアプリケーションの例です。設定ファイルで指定したディレクトリでファイルが追加、削除、または変更されるたびに、ログファイルに記録します。このファイルには、ログエントリが実行された日時が含められます。ただし、ディレクトリウォッチャが頻繁に変更を監視しない場合 (たとえば、1 分以上の間隔で監視している場合)、ログエントリの時刻は、実際にファイルが追加または変更された時刻と異なることがあります。したがって、この情報には、これらの操作の (日付ではなく) 時刻が含まれています。

```
<cfcomponent>
<cffunction name="onAdd" output="no">
    <cfargument name="CFEvent" type="struct" required="yes">
    <cfset data=CFEvent.data>
    <cflog file="MydirWatcher" application="No"
        text="ACTION: #data.type#;FILE: #data.filename#;
        TIME: #timeFormat(data.lastmodified)#">
</cffunction>

<cffunction name="onDelete" output="no">
<cfargument name="CFEvent" type="struct" required="yes">
<cfset data=CFEvent.data>
<cflog file="MydirWatcher" application="No"
    text=" ACTION: #data.type#;FILE: #data.filename#">
</cffunction>

<cffunction name="onChange" output="no">
<cfargument name="CFEvent" type="struct" required="yes">
<cfset data=CFEvent.data>
<cflog file="MydirWatcher" application="No"
        text=" ACTION: #data.type#;FILE: #data.filename#;
        TIME: #timeFormat(data.lastmodified)#">
</cffunction>
</cfcomponent>
```

JMSGateway

JMSGateway クラスは、Java Messaging Service コンシューマまたはプロデューサとして機能します。このイベントゲートウェイのソースは "gateway¥src¥examples¥JMS" にあります。このゲートウェイは設定ファイルを必要とします。設定ファイルは "gateway¥config¥jmsgateway.cfg" にあります。設定オプションの詳細については、設定ファイルを参照してください。ColdFusion Administrator の [ゲートウェイタイプ] ページに、コンパイル済みゲートウェイ ("gateway¥lib¥examples.jar" ファイルに含まれている) の一覧が表示されます。

注意：ColdFusion Administrator では、JMSGateway クラスのゲートウェイタイプ名として JMS が使用されます。

JMS ゲートウェイをコンシューマとして使用

JMSGateway クラスは、設定ファイルで指定されているトピックのサブスクライバを作成します。このゲートウェイは、次のタイプのメッセージを利用します。

- TextMessage
- UTF-8 テキストを含む BytesMessage

このゲートウェイは、メッセージの内容を次のイベント構造体に含めて、設定済み CFC に渡します。

フィールド	内容
data.id	メッセージ相関 ID
data.msg	メッセージのテキスト
gatewayType	ゲートウェイのタイプ : JMS
originatorID	メッセージが利用されたトピック名

リスナー CFC メソッドには、onIncomingMessage という名前を付ける必要があります。この CFC メソッドが応答でメッセージを送信しない場合は、OK または EXCEPTION という値が設定されたステータスフィールドを持つ構造体を返す必要があります。この場合、ゲートウェイは返されたステータスフィールドを確認しますが、この戻り値をさらに処理することはありません。この CFC メソッドがメッセージを送信するには、次のセクションで説明しているとおり、構造体を返す必要があります。

JMS ゲートウェイをプロデューサとして使用

JMS メッセージを送信するには、CFC メソッドの戻り値、または SendGatewayMessage 関数の 2 番目のパラメータ (**messageStruct**) に、次のフィールドを持つ構造体を使用します。

フィールド	内容
status	SEND であることが必要です。
topic	メッセージの提供先となるトピックの名前
id	(オプション) メッセージに関連付ける JMS 相関 ID。デフォルトは null です。
message	提供するメッセージのテキスト
asBytes	(オプション) メッセージを提供する方法。 • 省略した場合や、no か false を設定した場合は、テキストとしてメッセージを送信します。 • 他の値の場合は、メッセージをバイトエンコードされた UTF-8 として送信します。

SendGatewayMessage 関数でメッセージを送信した場合、この関数は、ゲートウェイがメッセージを送信した場合は OK を、メッセージの送信に失敗した場合は EXCEPTION を返します。

ActiveMQ JMS イベントゲートウェイ

Apache ActiveMQ は、JMS を実装するメッセージブローカーです。このイベントゲートウェイのソースは gateway/src/examples/ActiveMQ にあります。ActiveMQ JMS イベントゲートウェイの使用方法の詳細については、"gateway¥docs¥ActiveMQDeveloperGuide.pdf" ファイルを参照してください。

メニューベースのサンプルアプリケーション

ColdFusion には、メニューベースの応答アプリケーションが同梱されています。このメニューアプリケーションは、ColdFusion の任意の標準イベントゲートウェイ (SMS、XMPP、および Sametime)、および Socket サンプルイベントゲートウェイに対応しています。また次のように、SMS を使用するこのアプリケーションのインスタンスが ColdFusion にあらかじめ設定されています。

- ColdFusion Administrator の [ゲートウェイインスタンス] ページには、SMS ゲートウェイタイプを使用するこのアプリケーションのゲートウェイインスタンスが含まれています。
- "gateway/cfc/examples/menu" ディレクトリとそのサブディレクトリには、このアプリケーションで使用する CFML が含まれています。
- "gateway/config/sms-test.cfg" ファイルでは、ColdFusion に同梱されている SMS クライアント (電話シミュレーター) および SMSC (Short Message Service Center) サーバーシミュレーターに対してこのアプリケーションを使用するための設定が含まれています。

このアプリケーションでは、天気予報、株価情報、ステータス / 設定情報、辞書などの言語ツールといった、ユーザーが使用可能なツールのドリルダウンメニューが提供されます。

このアプリケーションのコードは、比較的複雑であり、13 ファイルに分けて配布されています。次の説明で、その仕組みを簡単に示します。アプリケーションの仕組みの詳細については、ソースコードを参照してください。

- トップレベル ("menu") ディレクトリには、"Application.cfm" と "main.cfc" の 2 つのファイルが含まれています。
- "Application.cfm" ファイルは、セッション管理を有効にし、アプリケーション名を指定する単一の cfapplication タグで構成されています。アクティブなメニューなど、セッションの状態に関する情報はセッション変数に保持しています。
- "main.cfc" ファイルにはマスタ CFC が含まれています。ColdFusion Administrator のイベントゲートウェイ設定は、このマスタ CFC をリスナー CFC として使用します。メイン CFC ファイルは、イベントゲートウェイからの CFEvent 構造体を処理します。行う処理は次のとおりです。
  1. gatewayType フィールドを確認し、構造体の残りの内容を判別します。イベントゲートウェイによって、メッセージが配置されているフィールドが異なるので、この確認が必要になります。
  2. Session.menu 変数が存在しない場合は、メニューシステムを初期化します。そのために、他の 2 つの CFC(menu と menunode) のメソッドを呼び出します。この 2 つの CFC には、メニューシステムのコードが含まれています。
  3. session.menu.process メソッドを呼び出して、ユーザーの入力を処理します。このメソッドは、必要に応じて、個別のアプリケーションにメッセージを送信して処理させることができます。
- "apps" ディレクトリには、複数の CFC が含まれています。各ファイルには、天気予報 ("weather.cfc") や辞書の検索 ("definition.cfc") などの、個別のアプリケーションのコードが含まれています。

Socket イベントゲートウェイでのメニューアプリケーションの使用

1. ColdFusion Administrator の [イベントゲートウェイ]-[設定] ページの [SMS テストサーバーの起動] ボタンをクリックします。
2. ColdFusion Administrator の [ゲートウェイインスタンス] ページで、緑色のボタン ([アクション] 列の左から 3 番目のボタン) をクリックして、SMS Menu App - 5551212 イベントゲートウェイを起動します。数秒後にステータスが起動中にならない場合は、[更新] をクリックして、サーバーが起動していることを確認します。
3. J2EE 設定の場合は "<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥bin" ディレクトリ、サーバー設定の場合は "<ColdFusion のルートディレクトリ >¥bin" ディレクトリにある "SMSClient.bat" ファイル (Windows の場合) または "SMSClient.sh" ファイル (UNIX または Linux の場合) を実行して、SMS 電話シミュレーターを起動します。このシミュレーターは、デフォルトの SMS イベントゲートウェイ設定に " コール " するように、デフォルトで設定されています。
4. キーボードを使用するか、シミュレーターのキーパッドを使用して文字を入力し、キーボードの Enter キーを押すか、シミュレーターの [Send] をクリックします。

5 メニューアプリケーションが、トップレベルメニューを使用して応答します。辞書やシソーラスなどの言語ツールの場合は「L」を、株価情報や天気予報を入手する場合は「S」を、サーバーに関する情報を取得する場合は「C」を入力します。キーボードの Enter キーを押すか、シミュレーターの [Send] をクリックします。

6 サブメニューが表示されます。たとえば、手順 5 で「S」を入力した場合は、Q (株価情報)、W (天気予報)、B (前のメニューに戻る) のオプションが表示されます。選択肢を入力します。

7 天気予報の場合は郵便番号、株価の場合は株式シンボル、辞書の場合は単語などの情報の入力が要求されます。必要な情報を入力して送信します (または「B」と入力して、メニューに戻ります)。

8 要求した情報が取得されて表示されます。アプリケーションによっては、「M」と入力して詳細情報を入手できることもあります。「M」(詳細情報を利用できる場合) または別の文字を入力します。前のメニューに戻るには「B」を入力します。

9 メニューアイテムや詳細情報のリクエストを何度か行います。

10 終了するには、メニューバーで [File]-[Exit] の順に選択します。

インスタントメッセージングイベントゲートウェイの使用

Adobe ColdFusion では、付属する 2 つのタイプのインスタントメッセージ (IM) イベントゲートウェイ (IBM Lotus Sametime ゲートウェイと XMPP (Extensible Messaging and Presence Protocol) ゲートウェイ) のいずれかを使用するアプリケーションを開発できます。

ただし、IM イベントゲートウェイを使用するには、ColdFusion イベントゲートウェイの原理やプログラミングテクニックに慣れておく必要があります (1242 ページの「イベントゲートウェイの使用」を参照)。

ColdFusion とインスタントメッセージについて

ColdFusion には、2 つのインスタントメッセージングゲートウェイタイプが用意されています。1 つは、XMPP プロトコルを使用したメッセージング用のゲートウェイタイプです。もう 1 つは、IBM Lotus Instant Messaging (Sametime) 用のゲートウェイタイプです。この 2 つのゲートウェイタイプは、同じインターフェイスを使用して、メッセージの送受信や、IM プレゼンス情報およびインフラストラクチャの管理を行います。したがって本マニュアルでは、2 つのゲートウェイタイプを IM ゲートウェイと総称し、違いがある場合にのみそれぞれのタイプについて説明します。

ColdFusion IM ゲートウェイは IM クライアントとして機能し、次の処理を行えます。

- インスタントメッセージの送受信
- バディまたは友人間でのメッセージの送受信、およびバディ / 友人情報の管理
- ステータス等の情報の設定および取得
- IM サーバーからのメッセージの受信および処理

XMPP について

XMPP (Extensible Messaging and Presence Protocol) は、インスタントメッセージングに使用される XML ベースのオープンプロトコルです。XMPP は、Jabber Software Foundation によって開発された Jabber Instant Messaging and Presence テクノロジーのコアプロトコルです。2004 年 11 月の時点で、XMPP は、IETF (Internet Engineering Task Force) の仕様 (RFC) 3920-3923 により定義されています。RFC 3920 は XMPP コアを、3921 はインスタントメッセージングとプレゼンスをそれぞれ規定しています。現在、多数の XMPP サーバーとクライアントが提供されており、ColdFusion でも IETF XMPP プロトコルがサポートされています。

次の Web サイトで、XMPP プロトコルに関する詳細情報が提供されています。

- Jabber Software Foundation : www.jabber.org/。このサイトには、利用可能な XMPP サーバーおよびクライアントに関する情報が記載されています。
- IETF にある XMPP のインターネット規格のコピー : www.ietf.org/rfc.html
- xmpp.org の Web サイト。2004 年 12 月の時点で開発中ですが、いくつかの有用なリンクが設定されており、関連する仕様にジャンプすることができます : www.xmpp.org/

IBM Lotus Instant Messaging (Sametime) について

IBM Lotus Instant Messaging (通常、Lotus Sametime と呼ばれる) は、リアルタイムのコラボレーションを実現する IBM 製品です。この製品の詳細については、www.lotus.com/sametime を参照してください。

注意 : エンタープライズ版で Lotus Sametime イベントゲートウェイを使用するには、次の行を ColdFusion Administrator で JVM 引数に追加して、FIPS-140 Compliant Strong Cryptography を無効にします。

-Dcoldfusion.disablejsafe=false

IM アプリケーションの開発とデプロイについて

ここでは、ColdFusion IM アプリケーションの開発ツールと開発プロセスについて説明し、IM メッセージングプロバイダについて説明します。

ColdFusion IM ゲートウェイクラス

ColdFusion には、次のインスタントメッセージングゲートウェイクラスが用意されています。

XMPPGateway XMPP イベントゲートウェイタイプ用のクラス

SAMETIMEGateway IBM Lotus インスタントメッセージングイベントゲートウェイ用のクラス

IM アプリケーションを実装するには、いずれかのゲートウェイクラスを使用するゲートウェイインスタンスを ColdFusion Administrator で設定し、このインスタンスを使用してインスタントメッセージングサーバーと通信する ColdFusion アプリケーションを作成します。

アプリケーションの開発およびデプロイのプロセス

次に、IM アプリケーションを開発およびデプロイするための一般的なプロセスを示します。

1. アプリケーションを設計します。
2. 利用可能な XMPP または Lotus Sametime サーバーを使用するよう IM イベントゲートウェイインスタンスを設定します。
3. CFC、CFM ページ、およびその他のアプリケーション要素を記述します。
4. XMPP または Lotus Sametime サーバーおよび適切なクライアントアプリケーションを使用して、アプリケーションをテストします。
5. アプリケーションをデプロイします (1257 ページの「イベントゲートウェイとアプリケーションのデプロイ」を参照)。

IM イベントゲートウェイとプロバイダ間の通信の仕組み

それぞれの IM イベントゲートウェイインスタンスには、1 つのインスタントメッセージング ID が割り当てられています。サーバー固有のツール (標準的なインスタントメッセージングクライアントなど) を使用して、ID とそれに関連するパスワードを IM サーバーで設定します。また ColdFusion で、ゲートウェイ設定ファイルに ID、パスワード、その他のゲートウェイ固有情報を設定して、このファイルを使用するゲートウェイインスタンスを作成します。

ゲートウェイを起動すると、ゲートウェイはその ID とパスワードを使用して IM サーバーにログオンし、この ID に対するメッセージを送受信します。ゲートウェイは、IM サーバーからの着信メッセージを、ColdFusion Administrator でゲートウェイインスタンスを設定するときに指定した CFC に送信します。同様に、CFC や他の CFML コードからの発信メッセージを IM サーバーに送信します。

IM イベントゲートウェイには、ゲートウェイや設定情報を管理するためのヘルパメソッドも用意されています。

着信メッセージの処理

次の ColdFusion CFC メソッドを記述して、IM イベントゲートウェイからの着信メッセージおよびリクエストを処理します。CFC は、IM サーバーからメッセージを受信し、戻り値を返すことでメッセージに応答できます。

CFC メソッド	メッセージタイプ
onIncomingMessage	IM ユーザーからの標準メッセージ
onAddBuddyRequest	「あなたのゲートウェイ ID を自分のバディリストに追加したい」という、他のユーザーからのリクエスト
onAddBuddyResponse	「あなたを自分のバディリストに追加したい」というリクエストを他のユーザーに送信した結果、そのユーザーから返された応答。「自分をあなたのバディリストから削除したい」というバディからのリクエストにも使用されます。
onBuddyStatus	他のユーザーからのプレゼンスステータスメッセージ
onIMServerMessage	IM サーバーからのエラーおよびステータスに関するメッセージ

これらのメソッドの詳細については、1269 ページの「着信メッセージの処理」を参照してください。

発信メッセージの処理

アプリケーションは、CFML の SendGatewayMessage メソッドを使用して発信インスタントメッセージを送信します。着信メッセージを処理する CFC メソッドでもメッセージを送信できます。たとえば、「あなたの ColdFusion ゲートウェイ ID を自分のバディリストに追加したい」という他のユーザーからのリクエストに対して、応答を送信できます。メッセージ送信の詳細については、1269 ページの「発信メッセージの送信」を参照してください。

IMGatewayHelper メソッド

ColdFusion IM ゲートウェイには、CFML の GetGatewayHelper 関数を呼び出すことでアクセス可能な IMGatewayHelper というゲートウェイヘルパクラスが用意されています。IMGatewayHelper クラスのメソッドを使用すると、次のことが行えます。

- ゲートウェイ設定情報の取得および設定や、ゲートウェイ統計情報の取得
- ゲートウェイのオンラインプレゼンスステータスの取得および設定
- ゲートウェイのバディリストの管理
- 他のユーザーに割り当てるゲートウェイステータス情報の取得権限の管理

GatewayHelper のメソッドの使用方法については、1275 ページの「GatewayHelper オブジェクトの使用」を参照してください。

IM イベントゲートウェイの設定

IM 固有の設定情報は、設定ファイルを使用して IM イベントゲートウェイに提供します。ColdFusion Administrator で IM イベントゲートウェイインスタンスを設定するときに、設定ファイルの場所を指定します。ColdFusion には、XMPP および Lotus Sametime イベントゲートウェイのサンプル設定ファイルが用意されています。この設定ファイルは、J2EE 設

定の場合は "<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥gateway¥config" ディレクトリに、サーバー設定の場合は "<ColdFusion のルートディレクトリ >¥gateway¥config" ディレクトリにあります。設定ファイルでは次の情報を指定できます。

注意：次の表のデフォルト値は、設定ファイルでプロパティが省略された場合にゲートウェイが使用する値であり、デフォルト設定ファイルの値ではありません。

プロパティ	デフォルト値	説明
userID	なし	(必須) IM サーバーへの接続に使用する IM ユーザー ID
password	なし	(必須) ユーザーのパスワード
secureprotocol	なし	XMPP のみ。 securerequirement を true に設定した場合は必須です。 セキュリティ保護された通信で使用するプロトコル。使用できる値は次のとおりです。 • TSL • SSL
securerequirement	false	XMPP のみ。 セキュリティ保護された通信を行う必要があるかどうかを指定します。使用できる値は次のとおりです。 • true • false この値が true の場合は、secureprotocol の値を指定する必要があります。セキュリティ保護された接続が確立した場合にのみ、接続することができます。
serverip	XMPP: jabber.org Sametime: stdemo3.dfw.ibm.com	メッセージの送信先となる XMPP または Lotus Sametime サーバーのアドレス。サーバー名または IP アドレスで指定できます。
serverport	XMPP: 5222 Sametime:1533	メッセージの送信先となるサーバーのポート。 XMPP の secureprotocol パラメータを SSL に設定した場合は、5223 を指定します。
retries	-1	ゲートウェイの起動時またはゲートウェイが切断された場合に、IM サーバーへの再接続を試行する回数を表す整数値。 0 = 再試行しない -1 = 永久に再試行する
retryinterval	5	接続の試行間隔 (秒単位) を表す実数。最低値は 1 秒です。
onIncomingMessageFunction	onIncomingMessage	着信メッセージを処理するために呼び出す CFC メソッドの名前。 "onIncomingMessageFunction=" のように、値を指定せずにプロパティを記述した場合、ゲートウェイはこのイベントを CFC に送信しません。
onAddBuddyRequestFunction	onAddBuddyRequest	着信バディリクエストを処理するために呼び出す CFC メソッドの名前。値を指定せずにプロパティを記述した場合、ゲートウェイはこのイベントを CFC に送信しません。
onAddBuddyResponseFunction	onAddBuddyResponse	ColdFusion から送信したバディリクエストに対する着信応答を処理するために呼び出す CFC メソッドの名前。値を指定せずにプロパティを記述した場合、ゲートウェイはこのイベントを CFC に送信しません。
onBuddyStatusFunction	onBuddyStatus	着信バディステータスメッセージを処理するために呼び出す CFC メソッドの名前。値を指定せずにプロパティを記述した場合、ゲートウェイはこのイベントを CFC に送信しません。
onIMServerMessageFunction	onIMServerMessage	着信メッセージメソッドを処理するために呼び出す CFC メソッドの名前。値を指定せずにプロパティを記述した場合、ゲートウェイはこのイベントを CFC に送信しません。

注意：特定のイベントタイプを処理する CFC メソッドが存在しない場合は、該当するプロパティを、値を指定せずに記述します。たとえば、IMServerMessage イベントを処理するメソッドが存在しない場合は、設定ファイルに onIMServerMessageFunction= と記述する必要があります。

着信メッセージの処理

IM イベントゲートウェイは、5 つのタイプのメッセージを処理します。したがって、CFC には、各メッセージタイプに対応するリスナーメソッドを実装する必要があります。次の表に、メッセージを処理する CFC メソッドと、そのメソッドが処理するメッセージを示します。この表に示されている CFC メソッドは、デフォルトの名前です。この名前はゲートウェイ設定ファイルで変更できます。

CFC メソッド	説明
onIncomingMessage	IM ユーザーからの標準メッセージ。アプリケーションは、メッセージの本文を適切に処理します。たとえば、メッセージをインターフェイスウィンドウに表示します。 このメソッドは、送信者に応答メッセージを返すことができます。
onAddBuddyRequest	「あなたのアプリケーションの IM ID を自分のバディリストに追加したい」という、他の IM ユーザーからのリクエスト。CFC は、このリクエストを承認または拒否するか、または何も実行しないことを決定する必要があります。何も実行しないのが適切である場合としては、リクエストをオフラインで検討してから承認し、後で応答を送信したい場合などがあります。 CFC は、決定した結果をメッセージの command フィールドに指定して、そのメッセージを返します。オプションで、理由を表すテキストメッセージも指定できます。リクエストを承認した場合は、ゲートウェイのステータス情報を取得できる ID のリストに、このリクエストを実行したユーザーが自動的に追加されます。何も実行しないことを選択した場合、ColdFusion は応答を行いません。
onAddBuddyResponse	「あなたを自分のバディリストに追加したい」というリクエストを他の IM ユーザーに送信した結果、そのユーザーから返された応答。応答メッセージは、承認か拒否のいずれかになります。 アプリケーションは、必要に応じてこの応答を処理できます。たとえば、メッセージ受信者のリストに ID を追加したり、このリストから ID を削除したりすることができます。 このメソッドは値を返しません。
onBuddyStatus	ゲートウェイバディのステータスを表すメッセージ。オフラインからオンラインに変更されたときなど、バディのステータスが変更されたときに受信します。 このメソッドは値を返しません。
onIMServerMessage	警告やエラーメッセージなど、IM サーバーからのステータスメッセージ。受信するメッセージは、IM サーバーによって異なります。サーバーメッセージの詳細については、ゲートウェイインスタンスで使用している IM サーバーのマニュアルを参照してください。 このメソッドは値を返しません。

使用方法の例など、各メソッドの詳細については、『CFML リファレンス』の IM Gateway CFC incoming message methods を参照してください。これらの関数の使用例については、1270 ページの「IM メッセージ処理アプリケーションのサンプル」を参照してください。

発信メッセージの送信

SendGatewayMessage CFML 関数または CFC リスナーメソッドの戻り値を使用して、発信メッセージを送信します。ColdFusion IM ゲートウェイでは、次の発信メッセージコマンドを使用できます。

コマンド	説明
submit	(デフォルト) 標準メッセージを他の IM ユーザーに送信します。
accept	バディの追加リクエストを承認します。プレゼンス情報の取得を許可する ID のリストにそのバディを追加して、承認メッセージをそのバディ ID に送信します。
decline	バディの追加リクエストを拒否し、拒否メッセージをそのバディ ID に送信します。
noact	ゲートウェイに対して、何も実行しないことを伝えます。ゲートウェイは、何も実行しなかったことを示すメッセージをログに記録します。このメッセージには、ゲートウェイのタイプ、ゲートウェイ ID、およびバディ ID が含まれます。

ゲートウェイリスナー CFC 関数で返すメッセージや、CFML の SendGatewayMessage 関数の 2 番目のパラメータとして使用するメッセージは、次のフィールドを持つメッセージ構造体です。次の表に、この構造体のフィールド、そのフィールドを使用するコマンド、フィールドの使用方法を示します。

フィールド	コマンド	説明
buddyID	すべて	宛先となるユーザーの ID
command	すべて	コマンド。省略した場合は、submit コマンドとして処理されます。
message	submit	宛先ユーザーに送信するテキストメッセージ
reason	accept、decline	バディ追加をリクエストしたユーザーに送信する、処理の理由を示すテキスト、またはその他のメッセージ

一般的に ColdFusion アプリケーションは、onAddBuddyRequest メソッドの戻り値に accept、decline、および noact コマンドを使用し、SendGatewayMessage CFML 関数で、および onIncomingMessage CFC メソッドの戻り値で submit コマンド (または、submit がデフォルトコマンドなのでコマンドを指定しない) を使用します。

CFML の SendGatewayMessage 関数は、任意のコマンドを送信できます。また、承認メッセージや拒否メッセージの送信にも使用できます。たとえば、すべてのバディリクエストを責任者が確認してから承認する場合は、SendGatewayMessage 関数で承認メッセージや拒否メッセージを送信します。この場合は、最初に CFC の onAddBuddyRequest メソッドの戻り値で noact コマンドを送信し、リクエスト情報をデータベースに保存します。責任者は、別の ColdFusion アプリケーションを使用して、リクエスト情報を確認します。このアプリケーションでは、SendGatewayMessage 関数を使用して accept または decline コマンドを送信することで、リクエストしたユーザーに通知します。

次に示すリスナー CFC の onIncomingMessage メソッドの例では、着信した IM メッセージをそのまま発信元に返します。

```
<cffunction name="onIncomingMessage" output="no">
    <cfargument name="CFEvent" type="struct" required="yes">
    <cfset retValue.MESSAGE = "echoing: " & CFEvent.DATA.message>
    <cfset retValue.BuddyID = arguments.CFEVENT.DATA.SENDER>
    <cfreturn retValue>
</cffunction>
```

IM メッセージ処理アプリケーションのサンプル

ここで説明するアプリケーションは、2 つの CFC で構成されています。1 つは、onIncomingMessage イベントに応答して従業員電話番号ディレクトリを検索する CFC です。もう 1 つは、他のすべてのイベントに応答するゲートウェイ管理 CFC です。この例では、アプリケーションがイベントに応答して、発信メッセージを送信する方法を示しています。

ColdFusion Administrator の [ゲートウェイインスタンス] ページに表示される [ColdFusion ゲートウェイインスタンスの追加 / 編集] フォームの [CFC パス] フィールドに、2 つの CFC へのパスをカンマで区切って入力することで、両方の CFC を使用するようにゲートウェイを設定できます。

電話番号ディレクトリ検索 CFC

次の CFC では、単純な従業員電話番号ディレクトリ検索アプリケーションを実装します。ユーザーは、検索する名前の一部を含んだインスタントメッセージを送信します (スペースの場合、すべての名前を要求します)。onIncomingMessage の応答は、一致件数によって異なります。

- 一致する項目がない場合、onIncomingMessage 関数は、一致する項目がないことを示すメッセージを返します。
- 一致件数が 1 件の場合は、名前、部門、および電話番号を返します。
- 一致件数が 10 件以下の場合は、番号が振られた名前リストを返します。ユーザーは、この番号を入力して詳細情報を取得できます。
- 一致件数が 10 件を超える場合は、最初の 10 件の名前リストのみを返します。より複雑なアプリケーションでは、複数のリストを取得してすべての名前にアクセスできるようにすることも可能です。
- 複数一致リストを取得した後にユーザーが番号を入力すると、その番号に対応する名前に関する情報を返します。

次にこの CFC のコードを示します。

```
<cfcomponent>
    <cffunction name="onIncomingMessage">
        <cfargument name="CFEvent" type="struct" required="YES">
        <!--- Remove any extra white space from the message. --->
        <cfset message =Trim(arguments.CFEvent.data.MESSAGE)>
        <!--- If the message is numeric, a previous search probably returned a
            list of names. Get the name to search for from the name list stored in
            the Session scope. --->
        <cfif isNumeric(message)>
            <cfscript>
                if (structKeyExists(session.users, val(message))) {
                    message = session.users[val(message)];
                }
            </cfscript>
        </cfif>

        <!--- Search the database for the requested name. --->
        <cfquery name="employees" datasource="cfdocexamples">
            select FirstName, LastName, Department, Phone
            from Employees
            where 0 = 0
            <!--- A space indicates the user entered a first and last name. --->
            <cfif listlen(message, " ") eq 2>
                and FirstName like '#listFirst(message, " ")#%'
                and LastName like '#listlast(message, " ")#%'
            <!--- No space: the user entered a first or a last name. --->
            <cfelse>
                and (FirstName like '#listFirst(message, " ")#%'
                or LastName like '#listFirst(message, " ")#%')
            </cfif>
        </cfquery>

        <!--- Generate andreturn the message.--->
        <cfscript>
            retrunVal = structNew();
            retrunVal.command = "submit";
            retrunVal.buddyID = arguments.CFEvent.data.SENDER;

            //No records were found.
            if (employees.recordCount eq 0) {
                retrunVal.message = "No records found for '#message#'";
            }
            //One record was found.
            else if (employees.recordCount eq 1) {
```

```
            // Whitespace in the message text results in bad formatting,
            // so the source cannot be indented.
                retrunVal.message = "Requested information:
#employees.firstName# #employees.lastName#
#employees.Department#
#employees.Phone#";
            }
            //Multiple possibilities were found.
            else if (employees.recordCount gt 1) {
                //If more than ten were found, return only the first ten.
                if (employees.recordCount gt 10)
                {
                    retrunVal.message = "First 10 of #employees.recordCount# records";
                }else{
                    retrunVal.message = "Records found: #employees.recordCount#";
                }
                // The session.users structure contains the found names.
                // The record key is a number that is also returned in front of the
                // name in the message.
                session.users = structNew();
                for(i=1; i lte min(10, employees.recordCount); i=i+1)
                {
                    // These two lines are formatted to prevent extra white space.
                    retrunVal.message = retrunVal.message & "
#i# - #employees.firstName[i]# #employees.lastName[i]#";
                    // The following two lines must be a single line in the source
                    session.users[i]="#employees.firstName[i]#
                        #employees.lastName[i]#";
                }
            }
            return retrunVal;
        </cfscript>
    </cffunction>
</cfcomponent>
```

ステータスおよびリクエストを処理する CFC

次に示す CFC は、onIncomingMessage 以外のすべての IM イベントを処理します。この CFC では、ゲートウェイのバディに関する情報を含んだ buddyStatus 構造体を Application スコープに保持しています。これによって、バディやステータスに関する情報を取得するために IM サーバーとやり取りせずに済むようにしています。また、IM サーバーから受信した重要なイベント (バディ追加リクエストなど) やエラーメッセージをログに記録します。特に、次のことを行います。

- onBuddyStatus 関数は、バディのステータスが変更されたことを示すイベントメッセージを取得したときに、Application スコープのバディステータス構造体を更新します。
- onAddBuddyRequest 関数は、リクエストされたバディの名前をデータソースで検索します。その名前が 1 件検出された場合は、バディを追加し、Application スコープの buddyStatus 構造体のステータスを更新します。その名前が検出されなかった場合は、バディリクエストを拒否します。その名前が複数検出された場合は、何も実行しないようにゲートウェイに指示します。また、この関数はすべての動作を記録します。
- onAddBuddyResponse 関数は、バディリクエストが承認された場合に、バディを Application スコープのバディステータス構造体に追加し、現在のステータスを設定します。この関数はすべての応答を記録します。
- onIMServerMessage 関数は、受け取ったすべてのメッセージを記録します。

  この例では、ColdFusion に同梱されている cfdocexamples データベースの Employees テーブルの IM_ID 列を使用しています。この列のエントリは、"company" という XMPP サーバーを使用していることを前提としています。したがって、このサンプルを実行するには、この名前を持つ XMPP サーバーを設定し、IM_ID 列に含まれている名前を持つク

ライアントを設定する必要があります。または、IM サーバーのクライアント名に一致するように、IM_ID 列のエントリを変更する必要があります。また、このサーバーを使用するゲートウェイインスタンスを、ColdFusion Administrator で設定する必要もあります。

次にこの CFC のコードを示します。

```
<cfcomponent>

<cffunction name="onBuddyStatus">
    <cfargument name="CFEvent" type="struct" required="YES">
    <cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
        <cfscript>
        // Create the status structures if they don't exist.
            if (NOT StructKeyExists(Application, "buddyStatus")) {
                Application.buddyStatus=StructNew();
            }
            if (NOT StructKeyExists(Application.buddyStatus, CFEvent.Data.BUDDYNAME)) {
                Application.buddyStatus[#CFEvent.Data.BUDDYNAME#]=StructNew();
            }
            // Save the buddy status and timestamp.
            Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].status=CFEvent.Data.BUDDYSTATUS;
            Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].timeStamp=CFEvent.Data.TIMESTAMP;
        </cfscript>
    </cflock>
</cffunction>

<cffunction name="onAddBuddyRequest">
    <cfargument name="CFEvent" type="struct" required="YES">
    <cfquery name="buddysearch" datasource="cfdocexamples">
        select IM_ID
        from Employees
        where IM_ID = '#CFEvent.Data.SENDER#'
    </cfquery>
    <cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
        <cfscript>
            // If the name is in the DB once, accept; if it is missing, decline.
            // If it is in the DB multiple times, take no action.
            if (buddysearch.RecordCount IS 0) {
                action="decline";
                reason="Invalid ID";
            }
            else if (buddysearch.RecordCount IS 1) {
                action="accept";
                reason="Valid ID";
                //Add the buddy to the buddy status structure only if accepted.
                if (NOT StructKeyExists(Application,
                        "buddyStatus")) {
                    Application.buddyStatus=StructNew();
                }
                if (NOT StructKeyExists(Application.buddyStatus,
                        CFEvent.Data.SENDER)) {
                    Application.buddyStatus[#CFEvent.Data.SENDER#]=StructNew();
                }
                Application.buddyStatus[#CFEvent.Data.SENDER#].status=
                    "Accepted Buddy Request";
                Application.buddyStatus[#CFEvent.Data.SENDER#].timeStamp=
                    CFEvent.Data.TIMESTAMP;
                Application.buddyStatus[#CFEvent.Data.SENDER#].message=
                    CFEvent.Data.MESSAGE;
            }
            else {
                action="noact";
```

```
                reason="Duplicate ID";
            }
        </cfscript>
    </cflock>
    <!--- Log the request and decision information. --->
    <cflog file="#CFEvent.GatewayID#Status"
            text="onAddBuddyRequest; SENDER: #CFEvent.Data.SENDER# MESSAGE: #CFEvent.Data.MESSAGE#
TIMESTAMP: #CFEvent.Data.TIMESTAMP# ACTION: #action#">
    <!--- Return the action decision. --->
    <cfset retValue = structNew()>
    <cfset retValue.command = action>
    <cfset retValue.BuddyID = CFEvent.DATA.SENDER>
    <cfset retValue.Reason = reason>
    <cfreturn retValue>
</cffunction>


<cffunction name="onAddBuddyResponse">
    <cfargument name="CFEvent" type="struct" required="YES">
    <cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
        <cfscript>
            //Do the following only if the buddy accepted the request.
            if (NOT StructKeyExists(Application, "buddyStatus")) {
                Application.buddyStatus=StructNew();
                }
            if (#CFEVENT.Data.MESSAGE# IS "accept") {
                //Create a new entry in the buddyStatus record for the buddy.
                if (NOT StructKeyExists(Application.buddyStatus,
                        CFEvent.Data.SENDER)) {
                    Application.buddyStatus[#CFEvent.Data.SENDER#]=StructNew();
                }
                //Set the buddy status information to indicate buddy was added.
                Application.buddyStatus[#CFEvent.Data.SENDER#].status=
                    "Buddy accepted us";
                Application.buddyStatus[#CFEvent.Data.SENDER#].timeStamp=
                    CFEvent.Data.TIMESTAMP;
                Application.buddyStatus[#CFEvent.Data.SENDER#].message=
                    CFEvent.Data.MESSAGE;
            }
        </cfscript>
    </cflock>
    <!--- Log the information for all responses. --->
    <cflog file="#CFEvent.GatewayID#Status"
        text="onAddBuddyResponse; BUDDY: #CFEvent.Data.SENDER# RESPONSE: #CFEvent.Data.MESSAGE# TIMESTAMP:
#CFEvent.Data.TIMESTAMP#">
</cffunction>


<cffunction name="onIMServerMessage">
    <!--- This function just logs the message. --->
    <cfargument name="CFEvent" type="struct" required="YES">
    <cflog file="#CFEvent.GatewayID#Status"
        text="onIMServerMEssage; SENDER: #CFEvent.OriginatorID# MESSAGE: #CFEvent.Data.MESSAGE# TIMESTAMP:
#CFEvent.Data.TIMESTAMP#">
</cffunction>
</cfcomponent>
```

GatewayHelper オブジェクトの使用

CFML の GetGatewayHelper 関数は、ゲートウェイやバディリストを管理するためのメソッドが用意されている GatewayHelper オブジェクトを返します。GatewayHelper のメソッドを使用すると、次のことが行えます。

- ゲートウェイ設定情報の取得および設定や、ゲートウェイ統計情報の取得
- ゲートウェイオンラインステータスの取得および設定
- ゲートウェイのバディリストの管理
- 他のユーザーに割り当てるゲートウェイステータス情報の取得権限の管理

次のセクションで、クラスメソッドについて簡単に説明します。各メソッドの詳細については、『CFML リファレンス』の IM Gateway GatewayHelper class methods を参照してください。

ゲートウェイの設定情報や統計情報を操作するメソッド

次の表に、ゲートウェイ設定情報の取得および設定や、ゲートウェイ統計情報の取得に使用できるメソッドを示します。

メソッド	説明
getName	ゲートウェイのユーザー名を返します。
getNickName	ゲートウェイの表示名 (ニックネーム) を返します。
getProtocolName	インスタントメッセージングプロトコルの名前 (JABBER for XMPP または SAMETIME) を返します。
numberOfMessagesReceived	起動してから、ゲートウェイが受け取ったメッセージの数を返します。
numberOfMessagesSent	起動してから、ゲートウェイが送信したメッセージの数を返します。
setNickName	ゲートウェイの表示名 (ニックネーム) を設定します。

ゲートウェイのオンラインステータスを操作するメソッド

次の表に、ゲートウェイのオンライン可用性ステータス (プレゼンス情報) の取得および設定に使用できるメソッドを示します。

メソッド	説明
getCustomAwayMessage	setStatus メソッドで不在と設定されている場合に、ゲートウェイのカスタム不在メッセージを返します。
getStatusAsString	ゲートウェイのオンラインステータスを返します。
getStatusTimeStamp	ゲートウェイがオンラインステータスを変更した日時を返します。
isOnline	ゲートウェイが IM サーバーに接続している場合は true を、接続していない場合は false を返します。
setStatus	ゲートウェイのオンラインステータス (不在、アイドルなど) を変更します。

ゲートウェイバディを管理するメソッド

次の表に、ゲートウェイバディリストの管理に使用できるメソッドを示します。

メソッド	説明
addBuddy	ゲートウェイのバディリストにバディを追加し、このバディのオンラインステータスを含むゲートウェイメッセージを送信するように IM サーバーに指示します。

メソッド	説明
getBuddyInfo	バディリスト、アクセス拒否リスト、およびアクセス許可リストから、指定したユーザーに関する情報を取得します。
getBuddyList	ゲートウェイのバディリストを返します。
removeBuddy	ゲートウェイのバディリストから特定のユーザー名を削除し、このユーザーのオンラインステータスを含むゲートウェイメッセージの送信を停止するように IM サーバーに指示します。

ゲートウェイのアクセス許可を管理するメソッド

IM ゲートウェイは、ゲートウェイのオンラインステータスを他のユーザーが取得できるかどうかに関する情報を管理できます。

注意: XMPP 1.0 の仕様には XMPP アクセス許可管理が含まれていますが、ColdFusion のリリース時点で提供されている XMPP サーバーの中には、アクセス許可管理がサポートされていないものもあります。

次の表に、ゲートウェイアクセス許可の管理に使用できるメソッドを示します。

メソッド	説明
addDeny	指定したユーザーをゲートウェイのアクセス拒否リストに追加するように IM サーバーに指示します。ゲートウェイの permitMode が DENY_SOME に設定されている場合、ここで指定したユーザーは、このゲートウェイのステータスメッセージを受け取れなくなります。
addPermit	指定したユーザーをサーバーのアクセス許可リストに追加するように IM サーバーに指示します。ゲートウェィの permitMode が PERMIT_SOME に設定されている場合、ここで指定したユーザーは、このゲートウェイのステータスメッセージを受け取れるようになります。
getDenyList	ステータス情報の送信が許可されていないユーザーのリストを返します。
getPermitList	ステータス情報の送信が許可されているユーザーのリストを返します。
getPermitMode	IM サーバーからゲートウェイのアクセス許可モードを取得します。このアクセス許可モードにより、すべてのユーザーがゲートウェイのオンラインステータス情報を取得できるかどうか、サーバーが、ステータス情報を取得するユーザーを制御するのにアクセス許可リストまたはアクセス拒否リストを使用するのかどうかが決定されます。
removeDeny	ユーザーをゲートウェイのアクセス拒否リストから削除します。
removePermit	ユーザーをゲートウェイのアクセス許可リストから削除します。
setPermitMode	IM サーバーでゲートウェイのアクセス許可モードを設定します。

GatewayHelper の例

この例では、XMPP または SameTime の GatewayHelper クラスを使用して、ステータスなどの情報を取得および設定できます (バディリストの管理やアクセス許可リストの表示などが行えます)。

```
<cfapplication name="gateway_tool" sessionmanagement="yes">

<!--- Set the gateway buddy name to default values.--->
<cfparam name="session.gwid" default="XMPP Buddy Manager">
<cfparam name="session.buddyid" default="hlichtin2@mousemail">

<!--- Reset gateway and buddy ID if form was submitted. --->
<cfif isdefined("form.submitbuddy")>
    <cfset session.buddyid=form.buddyid>
    <cfset session.gwid=form.gwid>
</cfif>

<!--- Display the current gateway and buddy ID. --->
<h3>Using the GatewayHelper</h3>
<!--- Form to display and reset gateway and Buddy ID. --->
<cfform action="#cgi.script_name#" method="post" name="changeIDs">
    Current buddy ID: <cfinput type="text" name="buddyid" value="#session.buddyid#"><br>
    Current gateway ID: <cfinput type="text" name="gwid" value="#session.gwid#"><br>
    <cfinput name="submitbuddy" value="Change gateway/buddy" type="submit">
</cfform>

<!--- When a buddy is set, display the links and forms to get and set
        information, and so on, Where form input is required, the form uses a GET method
        so a url.cmd variable represents each selection. --->

<cfoutput>
<h3>Select one of the following to get or set.</h3>
<ul>
    <li><a href="#cgi.script_name#?cmd=buddyinfo">buddyinfo</a>
    <li>LIST: <a href="#cgi.script_name#?cmd=buddylist">buddylist</a> |
        <a href="#cgi.script_name#?cmd=permitlist">permitlist</a> |
        <a href="#cgi.script_name#?cmd=denylist">denylist</a>
    <li>ADD: <a href="#cgi.script_name#?cmd=addbuddy">addbuddy</a> |
        <a href="#cgi.script_name#?cmd=addpermit">addpermit</a> |
        <a href="#cgi.script_name#?cmd=adddeny">adddeny</a>
    <li>REMOVE: <a href="#cgi.script_name#?cmd=removebuddy">removebuddy</a> |
        <a href="#cgi.script_name#?cmd=removepermit">removepermit</a> |
            <a href="#cgi.script_name#?cmd=removedeny">removedeny</a>
    <!--- NOTE: This list does not include OFFLINE because the gateway resets itself to online. --->
    <li>setStatus (XMPP):
        <cfloop list="ONLINE,AWAY,DND,NA,FREE_TO_CHAT" index="e">
            <a href="#cgi.script_name#?cmd=setstatus&status=#e#">#e#</a> |
        </cfloop>
    <li>setStatus (Sametime):
        <cfloop list="ONLINE,AWAY,DND,IDLE" index="e">
            <a href="#cgi.script_name#?cmd=setstatus&status=#e#">#e#</a> |
        </cfloop>
    <li>
        <form action="#cgi.script_name#" method="get">
            setStatus with CustomAwayMessage:
            <input type="hidden" name="cmd" value="setstatus2">
            <select name="status">
                <cfloop
list="ONLINE,OFFLINE,AWAY,DND,IDLE,INVISIBLE,NA,OCCUPIED,FREE_TO_CHAT,ONPHONE,ATLUNCH,BUSY,NOT_AT_HOME,NO
T_AT_DESK,NOT_IN_OFFICE,ON_VACATION,STEPPED_OUT,CUSTOM_AWAY" index="e">
                    <option value="#e#">#e#</option>
                </cfloop>
            </select>
            <input type="text" name="custommsg" value="(custom away massage)" size="30"/>
            <input type="submit"/>
        </form>
    <li>
```

```
        <form action="#cgi.script_name#" method="get">
            setNickName:
            <input type="hidden" name="cmd" value="setnickname">
            <input type="text" name="nickname" value="(enter nickname)">
            <input type="submit">
        </form>
--->
    <li>INFO: <a href="#cgi.script_name#?cmd=getname">getname</a> |
        <a href="#cgi.script_name#?cmd=getnickname">getnickname</a> |
        <a href="#cgi.script_name#?cmd=getcustomawaymessage">getcustomawaymessage</a> |
        <a href="#cgi.script_name#?cmd=getprotocolname">getprotocolname</a> |
        <a href="#cgi.script_name#?cmd=getstatusasstring">getstatusasstring</a> |
        <a href="#cgi.script_name#?cmd=isonline">isonline</a>
    <li>MESSAGE COUNT:
        <a href="#cgi.script_name#?cmd=numberofmessagesreceived">numberofmessagesreceived</a> |
        <a href="#cgi.script_name#?cmd=numberofmessagessent">numberofmessagessent</a>
    <li>RUNNING TIME: <a href="#cgi.script_name#?cmd=getsignontimestamp">getsignontimestamp</a> |
        <a href="#cgi.script_name#?cmd=getstatustimestamp">getstatustimestamp</a>
    <li>setPermitMode:
        <cfloop list="PERMIT_ALL,DENY_ALL,PERMIT_SOME,DENY_SOME,IGNORE_IN_LIST,IGNORE_NOT_IN_LIST"
                index="e"><a href="#cgi.script_name#?cmd=setpermitmode&mode=#e#">#e#</a> |
        </cfloop> <span class="note">doesn't work for XMPP</span>
    <li><a href="#cgi.script_name#?cmd=getpermitmode">getpermitmode</a>
    <li>setPlainTextMode:
        <cfloop list="PLAIN_TEXT,RICH_TEXT" index="e">
            <a href="#cgi.script_name#?cmd=setplaintextmode&mode=#e#">#e#</a> |
        </cfloop>
    <li><a href="#cgi.script_name#?cmd=getplaintextmode">getplaintextmode</a>
</ul>
</cfoutput>

<!--- The url.cmd value exists if one of the previous links or forms has been submitted, and identifies the
type of request. --->
<cfoutput>
<cfif isdefined("url.cmd")>
    <!--- Get the GatewayHelper for the gateway. --->
    <cfset helper = getGatewayHelper(session.gwid)>
    <!--- Get the buddy list if the list or full buddy information was requested. --->
    <cfswitch expression="#LCase(url.cmd)#">
        <cfcase value="buddylist,buddyinfo">
            <cfset ret=helper.getBuddyList()>
        </cfcase>
        <cfcase value="denylist">
            <cfset ret=helper.getDenyList()>
        </cfcase>
        <cfcase value="permitlist">
            <cfset ret=helper.getPermitList()>
        </cfcase>
        <cfcase value="addbuddy">
            <cfset ret=helper.addBuddy("#session.buddyid#",
                "#session.buddyid#", "")>
        </cfcase>
        <cfcase value="addpermit">
            <cfset ret=helper.addPermit("#session.buddyid#",
                "#session.buddyid#", "")>
        </cfcase>
        <cfcase value="adddeny">
            <cfset ret=helper.addDeny("#session.buddyid#",
                "#session.buddyid#", "")>
        </cfcase>
        <cfcase value="removebuddy">
            <cfset ret=helper.removeBuddy("#session.buddyid#", "")>
```

```
</cfcase>
<cfcase value="removepermit">
    <cfset ret=helper.removePermit("#session.buddyid#", "")>
</cfcase>
<cfcase value="removedeny">
    <cfset ret=helper.removeDeny("#session.buddyid#", "")>
</cfcase>
<cfcase value="setstatus">
    <cfset ret=helper.setStatus(url.status, "")>
</cfcase>
<cfcase value="setstatus2">
    <cfset ret=helper.setStatus(url.status, url.custommsg)>
</cfcase>
<cfcase value="getcustomawaymessage">
    <cfset ret=helper.getCustomAwayMessage()>
</cfcase>
<cfcase value="getname">
    <cfset ret=helper.getName()>
</cfcase>
<cfcase value="getnickname">
    <cfset ret=helper.getNickname()>
</cfcase>
<cfcase value="getprotocolname">
    <cfset ret=helper.getProtocolName()>
</cfcase>
<cfcase value="getsignontimestamp">
    <cfset ret=helper.getSignOnTimeStamp()>
</cfcase>
<cfcase value="getstatusasstring">
    <cfset ret=helper.getStatusAsString()>
</cfcase>
<cfcase value="getstatustimestamp">
    <cfset ret=helper.getStatusTimeStamp()>
</cfcase>
<cfcase value="isonline">
    <cfset ret=helper.isOnline()>
</cfcase>
<cfcase value="numberofmessagesreceived">
    <cfset ret=helper.numberOfMessagesReceived()>
</cfcase>
<cfcase value="numberofmessagessent">
    <cfset ret=helper.numberOfMessagesSent()>
</cfcase>
<cfcase value="setnickname">
    <cfset ret=helper.setNickName(url.nickname)>
</cfcase>
<cfcase value="setpermitmode">
    <cfset ret=helper.setPermitMode(url.mode)>
</cfcase>
<cfcase value="getpermitmode">
    <cfset ret=helper.getPermitMode()>
</cfcase>
<cfcase value="setplaintextmode">
    <cfset ret=helper.setPlainTextMode(url.mode)>
</cfcase>
<cfcase value="getplaintextmode">
```

```
            <cfset ret=helper.getPlainTextMode()>
        </cfcase>
        <cfdefaultcase>
            <cfset ret[1]="Error; Invalid command. You shouldn't get this.">
        </cfdefaultcase>
    </cfswitch>
    <br>
    <!--- Display the results returned by the called GatewayHelper method. --->
    <strong>#url.cmd#</strong><br>
    <cfdump var="#ret#">
    <br>
    <!--- If buddy information was requested, loop through buddy list to get
        information for each buddy and display it. --->
    <cfif comparenocase(url.cmd, "buddyinfo") is 0 and arraylen(ret) gt 0>
    <b>Buddy info for all buddies</b><br>
        <cfloop index="i" from="1" to="#arraylen(ret)#">
            <cfdump var="#helper.getBuddyInfo(ret[i])#" label="#ret[i]#"></cfloop>
    </cfif>
</cfif>
</cfoutput>
```

SMS イベントゲートウェイの使用

Adobe ColdFusion に用意されている SMS (Short Message Service) イベントゲートウェイタイプを使用するアプリケーションを開発できます。ColdFusion には、SMS アプリケーション開発ツールが用意されています。

ただし、SMS イベントゲートウェイを使用するには、ColdFusion イベントゲートウェイの原理やプログラミングテクニックに慣れておく必要があります (1242 ページの「イベントゲートウェイの使用」を参照)。必須ではありませんが、SMS の基本的な知識も役に立ちます。

SMS と ColdFusion について

SMS (Short Message Service) は、携帯電話やポケットベルなどのワイヤレスデバイスの間で、テキストなどのショートメッセージを交換するために設計されたシステムです。SMS は、ヨーロッパとアジアで広く普及しており、米国やその他の国々でも一般的になりつつあります。SMS の使用例としては、次のものがあります。

- 銀行取引の処理
- Web リソースへのアクセスなどに使用する認証コードの送信
- リアリティ番組の人気投票などの投票処理
- 処理 (サーバーの再起動など) の開始と応答の取得
- パッケージの出荷やレストランの空席状況などのイベントの通知や、株価や天気に関するアラート情報の提供
- 個人間でのテキストメッセージの送信
- 携帯電話でのテキストベースのインタラクティブメニューの提供
- ロゴの直接ダウンロードなどの、携帯電話のアップデートの提供
- 自動販売機、車両追跡システム、情報処理可能なガスポンプなどの、テレマティックスおよびモバイル / ワイヤレスデバイスアプリケーションの提供

SMS プロトコルは、次のものを始めとする数多くの機能や特徴を備えています。

- 組み込みの認証機能
- 通信のセキュリティ保護機能

- ほぼリアルタイムで実行されるストアアンドフォワード通信
- セッション対応の双方向通信機能
- 携帯電話などのモバイルデバイスでは SMS が既にサポートされているので、クライアントにソフトウェアをインストールする必要がない

SMS について

ここでは、SMS のテクノロジーについて簡単に説明し、ColdFusion アプリケーションにおける一般的な使用方法を示します。SMS の詳細については、SMS の解説書などの文献を参照してください。

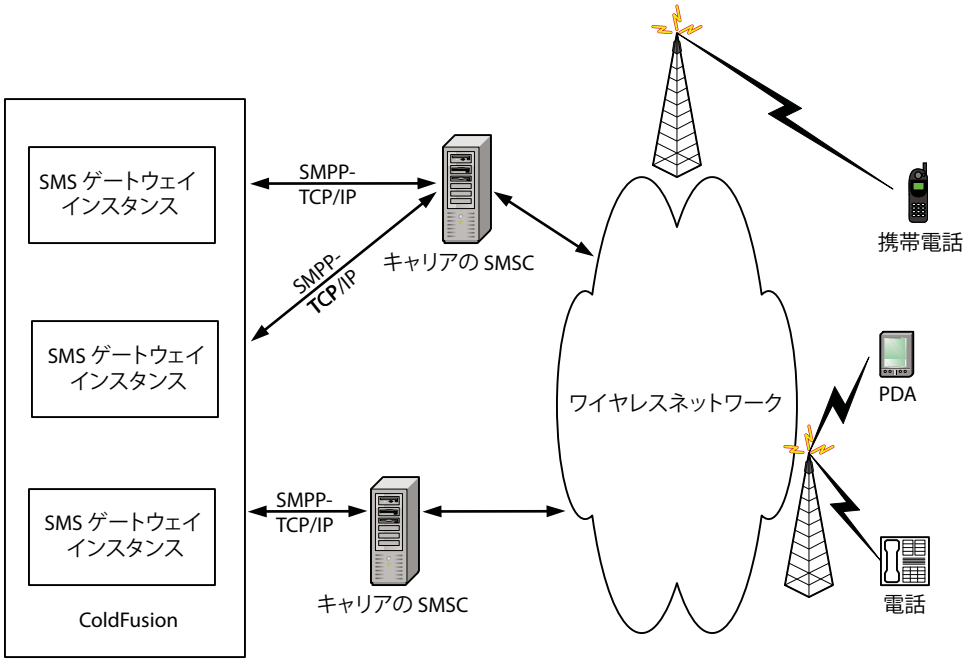
携帯電話などのモバイルデバイスは、SMSC (Short Message Service Center) などのメッセージセンターを介して、ColdFusion の SMS アプリケーションと通信を行います。たとえば、携帯電話のユーザーは、ColdFusion アプリケーションの電話番号 (これは SMS プロバイダによって割り当てられます) に電話をかけます。その番号に送信したメッセージは、SMSC に送られます。SMSC は、メッセージ全体をいったん保存してから転送 (ストアアンドフォワード) します。ColdFusion アプリケーションは、ワイヤレスデバイスからの着信メッセージに応答したり、ワイヤレスデバイスにメッセージを発信したりすることができます。

SMSC は、SMPP (Short Message Peer-to-peer Protocol) over TCP/IP を使用して、ColdFusion SMS イベントゲートウェイと通信を行います。情報の転送は、PDU (Protocol Data Unit) の交換によって行われます。PDU の構造は、トランザクションのタイプによって異なります。たとえば、標準メッセージ送信、バイナリデータ送信、複数の受信者へのメッセージ送信などがあります。

SMSC はストアアンドフォワード方式のサーバーなので、即座に配信できないメッセージを保存して、受信デバイスが利用可能になったときに配信を試行することができます。未配信のメッセージがサーバーに保存される期間は、SMSC プロバイダによって設定されます。たとえば、AT&T Wireless は 72 時間メッセージを保存します。72 時間が経過すると、未配信のメッセージは削除されます。メッセージの ValidityPeriod フィールドを指定することで、異なるタイムアウトを設定することができます。また、メッセージの registeredDelivery フィールドを使用して、メッセージが配信されたかどうかや、いつ配信されたかを通知するように SMSC に指示することもできます。

SMS の通信はセキュリティ保護することができます。音声 / データ通信 (SMSC とモバイルデバイス間の SMS メッセージトラフィックも含む) は、GSM 規格に基づいて暗号化されます。暗号化された通信セッションが開始される前に、モバイルユーザーの ID 認証も SMSC によって行われます。ColdFusion と SMSC の通信もセキュリティ保護する必要があります。一般には、安全なハードウェアまたはソフトウェアの VPN 接続を介して SMPP 接続を行います。

次の図に、モバイルデバイスと ColdFusion ゲートウェイの間の SMS パスを示します。



SMS イベントゲートウェイは、通信キャリアや SMPP アカウントプロバイダの SMSC と ColdFusion の間に双方向 (トランシーバー) 接続を確立します。イベントゲートウェイインスタンスには、SMPP プロバイダから割り当てられたアドレスを設定します。通常、このアドレスは電話番号ですが、ほとんどのキャリアでは、10 桁の数字を必要としない " 短縮コード " のアドレスがサポートされています。また、ゲートウェイインスタンスでは、システム ID とパスワードを使用してプロバイダの特定の TCP/IP アドレスと通信するための設定も行います。

注意：ColdFusion の SMS イベントゲートウェイは、SMPP 3.4 仕様に準拠しています。この仕様は、www.smsforum.net/ の SMS Forum からダウンロードできます。

ColdFusion アプリケーションから SMS 対応デバイスにメッセージを発信するには、宛先となるモバイルデバイスの電話番号を指定します。モバイルデバイスから ColdFusion のリスナー CFC にメッセージを送信するには、ゲートウェイインスタンスの電話番号を使用します。着信メッセージには、送信者の番号が含まれています。したがって、リスナー CFC は、モバイルデバイスから送信されたメッセージに応答することができます。

SMS アプリケーションの開発とデプロイについて

SMS ゲートウェイアプリケーションを開発するには、ColdFusion の SMS アプリケーション開発ツールと開発プロセスを使用して、SMS メッセージングプロバイダを操作します。

ColdFusion SMS アプリケーションツール

ColdFusion には、次に示す SMS アプリケーション開発ツールが用意されています。

SMSGateway SMS イベントゲートウェイタイプ用のクラス

SMS テストサーバー 軽量の SMSC シミュレーター

SMS クライアントシミュレーター SMS テストサーバーを使用して SMS メッセージを送受信するためのグラフィカルインターフェイス

SMS アプリケーションを実装するには、SMSGateway クラスのインスタンスを使用して SMSC と通信する ColdFusion アプリケーションを作成します。SMS テストサーバーとクライアントシミュレーターを使用すれば、外部の SMS サービスプロバイダを利用せずにアプリケーションをテストできます。

アプリケーションの開発およびデプロイのプロセス

SMS アプリケーションを開発およびデプロイする一般的なプロセスは、次のようになります。

1. アプリケーションを設計します。
2. ColdFusion SMS テストサーバーを使用するように SMS イベントゲートウェイインスタンスを設定します。
3. ColdFusion の CFC、CFM ページ、およびその他のアプリケーション要素を記述します。
4. テストサーバーとクライアントシミュレーターを使用してアプリケーションをテストします。
5. 通信プロバイダとの間で SMPP アカウントを確立します。
6. 通信プロバイダの SMSC を使用するようにイベントゲートウェイを再設定するか、または、通信プロバイダの SMSC を使用するイベントゲートウェイインスタンスを作成します。プロバイダから提供された情報を使用して、ゲートウェイを設定します。
7. 通信プロバイダの SMSC と、対象となるモバイルデバイスを使用して、アプリケーションをテストします。
8. アプリケーションを一般に提供します。

SMS プロバイダについて

SMS アプリケーションをデプロイする前に、SMPP 3.4 over TCP/IP をサポートしているプロバイダとの間でアカウントを確立します。プロバイダには次の 2 つのタイプがあります。

- 国内全域をカバーする携帯電話プロバイダなどの通信キャリア
- サードパーティの SMPP アグリゲーター

どのタイプのどのプロバイダを利用するかは、自社のニーズや、プロバイダが提供するサービスや料金体系などに基づいて判断することになります。一般に、低料金のプロバイダでは応答速度が遅くなります。反対に、高料金の通信キャリアではスループットが高く、SMPP の応答速度が速くなります。

SMS イベントゲートウェイとプロバイダ SMSC 間の通信の仕組み

ここでは、ColdFusion SMS イベントゲートウェイと SMPP プロバイダの SMSC との間の通信について簡単に説明します。SMPP 通信の基本事項について解説するとともに、ゲートウェイ設定やメッセージ処理に関連する用語について説明します。詳細については、SMPP の仕様を参照してください。この仕様は、www.smsforum.net/ で参照できます。

SMSC と ColdFusion SMS イベントゲートウェイインスタンス間の通信は、一般に、モバイルデバイスと ColdFusion イベントゲートウェイインスタンス (およびイベントゲートウェイアプリケーション) などの 2 つのエンティティ間で送信されるメッセージ (PDU) で構成されます。

ゲートウェイのバインディング

イベントゲートウェイを使用して通信を行うには、イベントゲートウェイを SMSC にバインドする必要があります。SMS イベントゲートウェイインスタンスは、bind_transceiver PDU を SMSC に送信することでバインディングを開始します。この PDU にはゲートウェイの ID とパスワードが含まれています。最初のバインドリクエストが失敗した場合、ゲートウェイは、ゲートウェイ設定ファイルで指定された再試行間隔の値に従ってバインドを再試行します。この再試行は、バインドが成功するか、ゲートウェイが設定ファイルで指定された最大試行回数に達するまで実行されます。バインド処理が失敗した場合は、"eventgateway.log" ファイルにエラーが記録されます。この場合、接続を確立するには、ゲートウェイインスタンスを ColdFusion Administrator で再起動します。

注意：一部の SMSC は、バインドリクエストに応答して禁止ステータスを送信できます。禁止ステータス応答を受け取ったゲートウェイは、再試行間隔を 1 分に、最大試行回数を 15 回に設定します。SMS ゲートウェイは、SMPP 5.0 に準拠する、AT&T 禁止ステータス応答を検出します。

バインドリクエストを承認した SMSC は、bind_transceiver_resp PDU を返します。このバインディングは、ゲートウェイインスタンスがシャットダウンして SMSC に unbind PDU を送信するまで有効です。ゲートウェイはトランシーバーとしてバインドしているので、ゲートウェイが SMSC にメッセージを発信することも、その逆も可能です。

着信 PDU の処理

ColdFusion SMS イベントゲートウェイが SMSC から Unbind PDU を取得すると、unbind_resp PDU を SMSC に送信して再起動を行い、SMSC へのバインディングを再試行します。

イベントゲートウェイが、EnquireLink またはその他のリクエスト PDU を SMSC から受け取ると、デフォルトの応答を SMSC に送信します。

ゲートウェイは、deliver _sm PDU で SMSC から着信メッセージを受け取りますが、data_sm PDU は処理しません。deliver_sm PDU には、ユーザーまたはアプリケーションが生成したメッセージや、ゲートウェイが送信したメッセージに対する処理応答が含まれています。ゲートウェイは、PDU からショートメッセージフィールドとソース / 宛先アドレスを抽出して、CFEvent オブジェクトに格納し、このオブジェクトを ColdFusion イベントゲートウェイサービスに送信して、リスナー CFC に配信します。CFML アプリケーションにおける着信メッセージの処理方法の詳細については、1286 ページの「着信メッセージの処理」を参照してください。

発信メッセージの処理

ゲートウェイでは、ColdFusion アプリケーションからの 3 種類の発信メッセージがサポートされています。CFML の sendGatewayMessage 関数や、リスナー CFC メソッドの cfreturn タグでは、次のコマンドを指定できます。

submit SMSC に対して、メッセージとともに submit_sm PDU を送信します。この PDU は単一の宛先にメッセージを送信します。

submitMulti SMSC に対して、メッセージとともに submit_multi PDU を送信します。この PDU は、複数の宛先にメッセージを送信します。

data SMSC に対して、メッセージとともに data_sm PDU を送信します。これは submit コマンドの代替コマンドであり、WAP (Wireless Application Protocol) フレームワークなどを介して提供される対話型アプリケーションでよく使用されます。

SMS ゲートウェイでは、これらの PDU のすべてのフィールドの内容を管理できます。個々のコマンドの詳細については、1288 ページの「発信メッセージの送信」を参照してください。

メッセージを受信した SMSC から、メッセージが拒否されたことや送信されなかったことを示すステータスが返された場合は、失敗に関する情報が "eventgateway.log" ファイルに記録されます。サービスタイプが利用できないことを示すステータス (SMPP v5 の ESME_RSERTYPUNAVAIL ステータスまたは AT&T のサービス拒否ステータス) が SMSC から返された場合、ゲートウェイ設定ファイルの transient-retry の値が yes に設定されていれば、ゲートウェイはメッセージの再送信も試行します。

発信メッセージの同期と通知

ゲートウェイと SMSC は非同期で通信を行います。つまり、ゲートウェイは、送信したメッセージに対する SMSC からの応答を待たずに、次のメッセージを送信します。ただし、ゲートウェイインスタンスを設定することで、CFML の sendGatewayMessage 関数が非同期または同期モードで動作するように指定できます。

- 非同期モードの場合は、ColdFusion ゲートウェイサービスのキューにメッセージが追加されると、関数から処理が返されます。
- 同期モードの場合は、SMSC がメッセージを受け取ってメッセージ ID を返すか、エラーが発生するまで、この関数は待機します。

メッセージの同期設定や同期送信の詳細については、1291 ページの「SMS メッセージの送信および応答の制御」を参照してください。

SMS イベントゲートウェイの設定

SMS 固有の設定情報は、設定ファイルを使用して SMS イベントゲートウェイに提供します。ColdFusion Administrator で SMS イベントゲートウェイインスタンスを設定するときに、設定ファイルの場所を指定します。ColdFusion には、SMS イベントゲートウェイのサンプル設定ファイルが用意されています。この設定ファイルは、J2EE 設定の場合は "<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥gateway¥config¥sms-test.cfg" に、サーバー設定の場合は "<ColdFusion のルートディレクトリ >¥gateway¥config¥sms-test.cfg" にあります。次の表で、この設定ファイルの内容について説明します。

注意：次の設定情報では設定フィールドについて説明しますが、SMPP 固有の用語の詳細な説明や、SMPP 仕様で定義されている有効なプロパティ値の一覧、適切な SMPP 値の選択方法については説明していません。詳細については、www.smsforum.net/ の SMPP 3.4 プロトコルに関する資料や、その他の一般に提供されている資料を参照してください。通常、これらの設定値の一部については、SMS サービスプロバイダが要件を指定しています。プロバイダが提供するマニュアルを参照してください。

プロパティ	デフォルト	説明
ip-address		SMPP プロバイダから指定される SMSC の IP アドレス。ColdFusion SMS テストサーバーの場合、通常は 127.0.0.1 を使用します。
port	0	SMSC にバインドするポート番号。ColdFusion SMS テストサーバーはポート 7901 を使用します。
system-id		SMPP プロバイダと接続を確立する場合に、SMSC がこのイベントゲートウェイを識別するための名前。ColdFusion SMS テストサーバーに接続する場合は、cf という system-id を指定する必要があります。
password		SMSC でイベントゲートウェイを認証するためのパスワード。ColdFusion SMS テストサーバーに接続する場合は、cf という password を指定する必要があります。
source-ton	1	SMPP 仕様で指定されている、ソースアドレスの TON (Type of Number)。ソースアドレスとは、発信メッセージに対してイベントゲートウェイが使用するアドレスのことです。値には 0 (不明)、1 (国際番号)、2 (国内番号) を使用できます。
source-npi	1	SMPP 仕様で指定されている、ソースアドレスの NPI (Numeric Plan Indicator)。値には、0 (不明)、1 (ISDN) を使用できます。
source-address	空の文字列	イベントゲートウェイのアドレス (通常は電話番号)。発信メッセージの送信者を SMSC に通知します。
addr-ton	1	このイベントゲートウェイが提供する着信アドレスの TON。
addr-npi	1	このイベントゲートウェイが提供する着信アドレスの NPI。
address-range		リモートデバイスがイベントゲートウェイインスタンスにメッセージを送信する場合に使用できる着信アドレス (電話番号) の範囲。ほとんどの場合、source-address と同じです。
message-rate	100	ゲートウェイが 1 秒間にプロバイダに送信できるメッセージ数を指定する整数値または 10 進数値。0 は無制限です。
mode	synchronous	メッセージ伝送モード。 • synchronous : ゲートウェイは、メッセージを送信すると、サーバーからの応答を待ちます。このモードでは、CFML の SendGatewayMessage 関数はメッセージの SMS messageID を返すか、エラーが発生した場合は空の文字列を返します。 • asynchronous : ゲートウェイは応答を待ちません。このモードでは、CFML の SendGatewayMessage 関数は常に空の文字列を返します。

プロパティ	デフォルト	説明
network-retry	no	メッセージの送信時にネットワークエラーが発生した場合のゲートウェイの動作。 • yes : ゲートウェイはメッセージをキューに追加し、SMSC への再バインドが可能になったらメッセージを配信します。ゲートウェイが非同期モードの場合は、再試行するように設定するのが便利です。非同期モードの場合、CFML の SendGatewayMessage 関数はエラーを返しません。 • no : ゲートウェイはメッセージの送信を再試行しません。
transient-retry	no	一時的な原因によるエラーを SMSC が返した場合のゲートウェイの動作。この場合、時間が経過すれば SMSC がメッセージを受信できるようになることがあります。 • yes : ゲートウェイはメッセージの送信を再試行します。ゲートウェイが非同期モードの場合は、再試行するように設定するのが便利です。非同期モードの場合、CFML の SendGatewayMessage 関数はエラーを返しません。 • no : ゲートウェイはメッセージの送信を再試行しません。
cfc-method	onIncomingMessage	ゲートウェイが着信メッセージを取得したときに ColdFusion が呼び出すリスナー CFC メソッド。
destination-ton	1	発信メッセージのアドレスのデフォルトの TON。
destination-npi	1	発信メッセージのアドレスのデフォルトの NPI。
service-type	空の文字列	メッセージングサービスのタイプ。空の文字列か、CMT、CPT、VMN、VMA、WAP、または USSD を指定できます。
system-type	空の文字列	SMSC へのバインディング時に使用するシステムのタイプ (External Short Message Entity、略して ESME)。SMSC によっては、特定の ESME タイプに固有の応答を送信できることがあります。通常は SMPP に設定します。
receive-timeout	-1 (タイムアウトしない)	接続の確立後、SMSC からメッセージを受信するために待機するタイムアウト (秒単位)。メッセージを受信するまで無制限に待機する場合は、receive-timeout を -1 に設定します。
ping-interval	60	接続が確立されていることを検証するためにイベントゲートウェイがサーバーに送信する EnquireLink メッセージの送信間隔の秒数。
retries	-1 (永久に再試行)	ゲートウェイが切断状態に移行する前に、メッセージを送信するために SMSC への接続を再試行する回数。ゲートウェイが切断状態の場合、getStatus メソッドは FAILED を返します。また ColdFusion Administrator でゲートウェイのステータスが失敗と表示されます。このゲートウェイを使用するには、再起動する必要があります。
retry-interval	10	接続を再試行する間隔 (秒単位)。

発信メッセージでは、source-ton、source-npi、source-address、destination-ton、destination-npi、および service-type の値も設定できます。メッセージフィールド名は、設定ファイルのプロパティ名とは異なります。

着信メッセージの処理

SMS イベントゲートウェイは、deliver_sm PDU に含まれるメッセージを処理します。この PDU は、ゲートウェイに対して、次のいずれかのタイプのメッセージを配信するよう要求します。

- ユーザーまたはアプリケーションによって生成されたテキストメッセージ
- メッセージ処理応答

注意：SMS イベントゲートウェイは、data_sm PDU に含まれるメッセージを処理しません。

このオブジェクトは、イベントゲートウェイからイベントゲートウェイサービスに送信され、さらにイベントゲートウェイサービスからリスナー CFC に配信されます。リスナー CFC が受け取る CFEvent オブジェクトには、次のフィールドが存在します。

注意：イベントゲートウェイハンドラから渡される SMS メッセージなどのデータは、悪意のあるデータである可能性を考慮する必要があります。たとえば、SMS データを SQL データベースにアーカイブしている場合、攻撃者が悪意のあるメッセージを送信して、データを変更または削除しようとしたり、SQL サーバーを乗っ取ろうとしたりすることがあります。したがって、Web フォームの入力を検証する場合と同様に、イベントゲートウェイの入力も検証してください。

フィールド	値
CfcMethod	リスナー CFC のメソッド名
Data.dataCoding	メッセージ内容の文字セットまたは文字以外のデータ型
Data.destAddress	メッセージの送信先アドレス
Data.esmClass	メッセージタイプ
Data.MESSAGE	メッセージの内容
Data.messageLength	MESSAGE フィールドの長さ
Data.priority	メッセージの優先度レベル。0-3 の範囲で指定します。
Data.protocol	GSM プロトコル。他のネットワークでは使用しません。
Data.registeredDelivery	要求された配信受信または配信承認のタイプ (存在する場合)
Data.sourceAddress	このメッセージを送信したデバイスのアドレス
GatewayType	常に SMS
OriginatorID	メッセージを送信したデバイスのアドレス

各フィールドの詳細については、『CFML リファレンス』の SMS Gateway incoming message CFEvent structure を参照してください。

CFC のリスナーメソッドでは、Arguments.CFEvent.Data.MESSAGE フィールドからメッセージを取得して、適切にメッセージを処理します。必要に応じて、次の 2 つのフィールドを使用して、必要なアクションを判断します。

- CFEvent.Data.esmClass は、MESSAGE フィールドの情報のタイプを表します。
- CFEvent.Data.registeredDelivery は、配信受信または配信承認を送信者が要求したかどうかを表します。

CFEvent.Data.esmClass フィールド

CFEvent.Data.esmClass フィールドは、CFEvent.Data.Message フィールドにメッセージが含まれているのか、次のタイプのメッセージ処理応答が含まれているのかどうかを識別します。応答が含まれている場合、CFEvent オブジェクトの Data.MESSAGE フィールドには承認または受信情報が含まれます。

SMSC 配信受信 SMSC によって送信された、メッセージの最終ステータスを示す通知。ショートメッセージテキストには、元のメッセージのメッセージ ID、送信および配信されたメッセージ (配布リストに送信されたメッセージ) の数、メッセージが送信 / 配信または処理された日時、メッセージの処理、ネットワーク固有のエラーコード (エラーの場合)、およびメッセージの最初の 20 バイトが含まれています。SMSC 配信受信メッセージ構造体の詳細については、SMS 3.4 仕様の「Appendix B」を参照してください。

SME 配信承認 ユーザーがショートメッセージを読んだことを示す受信デバイスからの通知。TDMA および CDMA ワイヤレスネットワークでのみサポートされています。

SME 手動 / ユーザー承認 アプリケーションリクエストメッセージに応答して送信された、アプリケーション生成の返信メッセージ。TDMA および CDMA ワイヤレスネットワークでのみサポートされています。

中間配信通知 まだ配信されていないメッセージのステータスに関するプロバイダ固有の通知。このメッセージの SMSC 再試行ライフタイム中に送信されます。中間通知がサポートされているかどうかは、SMSC の実装と SMSC サービスプロバイダに依存します。詳細については、プロバイダのマニュアルを参照してください。

メッセージを送信するときは、発信メッセージの registered_delivery パラメータで、メッセージ処理応答をリクエストできます。アプリケーションで応答をリクエストする場合は、リスナー CFC でそれらのメッセージを適切に処理できる必要があります。

CFEvent.Data.registeredDelivery フィールド

CFEvent.Data.registeredDelivery フィールドは、メッセージ送信者が受信または承認をリクエストしたかどうかを示します。アプリケーションは、SME 配信承認または SME 手動 / ユーザー承認のリクエストに応答することができます。その他の通知タイプは、SMSC によってのみ送信されます。これらの通知タイプの詳細については、SMS 3.4 の仕様を参照してください。「Appendix B」に、返信する承認メッセージの shortMessage フィールドに設定する必要のある情報が記載されています。

着信メッセージの処理例

次のサンプルコードは、ColdFusion の "gateway/cfc/examples" ディレクトリに用意されている、SMS でのみ使用可能な "echo.cfc" サンプルです。このサンプルには、SMS メッセージの受信および応答に必要な最低限のコードが含まれています。

```
<cfcomponent displayname="echo" hint="Process events from the test gateway and return echo">
<cffunction name="onIncomingMessage" output="no">
    <cfargument name="CFEvent" type="struct" required="yes">
    <!--- Get the message --->
    <cfset data=cfevent.DATA>
    <cfset message="#data.message#">
    <!--- where did it come from? --->
    <cfset orig="#CFEvent.originatorID#">
    <cfset retValue = structNew()>
    <cfset retValue.command = "submit">
    <cfset retValue.sourceAddress = arguments.CFEVENT.gatewayid>
    <cfset retValue.destAddress = arguments.CFEVENT.originatorid>
    <cfset retValue.shortMessage = "echo: " & message>
    <!--- send the return message back --->
    <cfreturn retValue>
</cffunction>
</cfcomponent>
```

発信メッセージの送信

ColdFusion アプリケーションは、発信メッセージで submit、submitMulti、および data コマンドをイベントゲートウェイに送信できます。

submit コマンド

SMPP SUBMIT_SM PDU で単一の宛先アドレスにメッセージを送信する場合、SendGatewayMessage 関数の **Data** パラメータや CFC リスナーメソッドの戻り変数で使用する構造体には、一般的に次のフィールドを含めます。

フィールド	内容
command	存在する場合、この値を submit に設定する必要があります。このフィールドを省略した場合、イベントゲートウェイは submit メッセージを送信します。

フィールド	内容
shortMessage または messagePayload	メッセージの内容。いずれかのフィールドを指定します。両方は指定できません。SMPP の仕様では、shortMessage フィールドに対し 254 バイトの最大サイズの制限が設定されています。一部のキャリアは、今後このサイズをさらに制限する方向にあります。messagePayload フィールドには、最大で 64K バイトのデータを設定できます。最初に 0x0424 を指定し、データの長さを示す 2 バイトのデータを設定した後に、メッセージの内容を続けます。
destAddress	メッセージの送信先アドレス (必須)
sourceAddress	このアプリケーションのアドレス。設定ファイルで指定した場合は、このフィールドを省略できます。

この構造体には、配信受信をリクエストするフィールドなどのオプションのフィールドも設定できます。すべてのフィールドのリストについては、『CFML リファレンス』の submit command を参照してください。これらのフィールドの詳細については、SMPP 3.4 仕様の SUBMIT_MULTI PDU に関する資料を参照してください。この資料は、www.smsforum.net/ の SMS Forum からダウンロードできます。

注意 : 長いメッセージを送信する場合は、メッセージを複数の部分に分割し、submit コマンドを使用して各部分を個別に送信することができます。この場合、CFC では、cfreturn 関数ではなく、SendGatewayMessage 関数を複数使用します。

例 : sendGatewayMessage 関数での submit コマンドの使用

次に示す CFM ページの抜粋では、フォームに入力した SMS メッセージを、submit コマンドを使用して CFML の sendGatewyMessage 関数で送信します。この例では、ColdFusion で設定された SMS ゲートウェイを使用して、SMS クライアントシミュレーターにメッセージを送信します。

```
<h3>Sending SMS From a Web Page Example</h3>
<cfif IsDefined("form.oncethrough") is "Yes">
<cfif IsDefined("form.SMSMessage") is True AND form.SMSMessage is not "">
    <h3>Sending Text Message: </h3>
        <cfoutput>#form.SMSMessage#</cfoutput><br>
        <cfscript>
        /* Create a structure that contains the message. */
        msg = structNew();
        msg.command = "submit";
        msg.destAddress = "5551234";
        msg.shortMessage = form.SMSMessage;
        ret = sendGatewayMessage("SMS Menu App - 5551212", msg);
        </cfscript>
</cfif>
<hr noshade>
</cfif>
<!--- begin by calling the cfform tag --->
<cfform action="command.cfm" method="POST">
    SMS Text Message: <cfinput type="Text" name="SMSMessage" value="Sample text Message" required="No"
maxlength="160">
<p><input type = "submit" name = "submit" value = "Submit">
<input type = "hidden" name = "oncethrough" value = "Yes">
</cfform>
</body>
</html>
```

submit コマンドを使用して、着信 SMS メッセージをメッセージの発信元にエコーする単純なリスナー CFC の例については、1288 ページの「着信メッセージの処理例」を参照してください。

submitMulti コマンド

SMPP SUBMIT_MULTI PDU を使用して複数の受信者に同じテキストメッセージを送信する場合、SendGatewayMessage 関数の **Data** パラメータや CFC リスナーメソッドの戻り変数には、一般的に次のフィールドを含めます。

フィールド	内容
command	submitMulti を指定する必要があります。
shortMessage または messagePayload	メッセージの内容です。いずれかのフィールドを指定します。両方は指定できません。SMPP の仕様では、shortMessage フィールドに対し 254 バイトの最大サイズの制限が設定されています。一部のキャリアは、今後このサイズをさらに制限する方向にあります。messagePayload フィールドには、最大で 64K バイトのデータを設定できます。最初に 0x0424 を指定し、データの長さを示す 2 バイトのデータを設定した後に、メッセージの内容を続けます。
destAddress	宛先アドレスの ColdFusion 配列 (必須)。 これらのアドレスに対して個別の TON および NPI の値を指定することはできません。すべてのアドレスで同じ設定を使用する必要があります。
sourceAddress	このアプリケーションのアドレス。設定ファイルで指定した場合は、このフィールドを省略できます。

この構造体には、配信受信をリクエストするフィールドなどのオプションのフィールドも設定できます。すべてのフィールドのリストについては、『CFML リファレンス』の submitMulti command を参照してください。これらのフィールドの詳細については、SMPP 3.4 仕様の SUBMIT_MULTI PDU に関する資料を参照してください。この資料は、www.smsforum.net/ の SMS Forum からダウンロードできます。

例 : onIncomingMessage メソッドでの submitMulti コマンドの使用

次に示す onIncomingMessage メソッドの例では、着信メッセージを発信元アドレスにエコーする応答を送信し、その応答のコピーを 2 番目のアドレスに送信します。このサンプルをテストするには、ColdFusion SMS クライアントアプリケーションのインスタンスを 2 つ実行します。最初のインスタンスにはデフォルトの電話番号 5551212 を使用し、2 番目のインスタンスには 555-1235 の電話番号を設定します。2 番目のインスタンスの電話番号にはハイフン (-) が必要となることに注意してください。最初のシミュレーターからメッセージを送信します。すると応答が両方のウィンドウに表示されます。

```
<cffunction name="onIncomingMessage" output="no">
    <cfargument name="CFEvent" type="struct" required="yes">
    <!--- Get the message. --->
<cfset data=CFEvent.DATA>
<cfset message="#data.message#">
    <!--- Create the return structure. --->
    <cfset retValue = structNew()>
    <cfset retValue.command = "submitmulti">
    <cfset retValue.sourceAddress = arguments.CFEVENT.gatewayid>
    <cfset retValue.destAddresses=arraynew(1)>
    <!--- One destination is incoming message originator;
                    get the address from CFEvent originator ID. --->
    <cfset retValue.destAddresses[1] = arguments.CFEvent.originatorid>
    <cfset retValue.destAddresses[2] = "555-1235">
    <cfset retValue.shortMessage = "echo: " & message>
<cfreturn retValue>
</cffunction>
</cffunction>
```

data コマンド

SMPP DATA_SM PDU を使用して単一の宛先アドレスにバイナリデータを送信する場合、SendGatewayMessage 関数の **Data** パラメータや CFC リスナーメソッドの戻り変数には、次のフィールドを設定する必要があります。

フィールド	内容
command	data を指定する必要があります。

フィールド	内容
messagePayload	メッセージデータ。データをバイナリ形式に変換するには、ColdFusion の toBinary 関数を使用します。
destAddress	メッセージの送信先アドレス
sourceAddress	このアプリケーションのアドレス。設定ファイルで指定した場合は省略できます。

この構造体には、配信受信をリクエストするフィールドなどのオプションのフィールドも設定できます。すべてのフィールドのリストについては、『CFML リファレンス』の data command を参照してください。これらのフィールドの詳細については、SMPP 3.4 仕様の SUBMIT_MULTI PDU に関する資料を参照してください。この資料は、www.smsforum.net/ の SMS Forum からダウンロードできます。

例 : data コマンドの使用

次の onIncomingMessage メソッドの例は、着信メッセージをバイナリデータに変換し、メッセージのバイナリバージョンを発信元アドレスに送り返します。

```
<cffunction name="onIncomingMessage" output="no">
    <cfargument name="CFEvent" type="struct" required="yes">
    <!--- Get the message. --->
<cfset data=CFEvent.DATA>
<cfset message="#data.message#">
    <!--- Create the return structure. --->
    <cfset retValue = structNew()>
    <cfset retValue.command = "data">
    <!--- Sending to incoming message originator; get value from CFEvent. --->
    <cfset retValue.destAddress = arguments.CFEvent.originatorid>
    <cfset retValue.messagePayload = tobinary(tobase64("echo: " & message))>
    <cfreturn retValue>
</cffunction>
```

SMS メッセージの送信および応答の制御

本マニュアルでは、メッセージの送信に使用できる一般的なオプションをいくつか説明するとともに、それらのオプションがアプリケーションに与える影響についても説明します。発信メッセージを設定するための他の方法については、SMPP 仕様を参照してください。

同期モード

ゲートウェイ設定ファイルで、非同期または同期メッセージモードを指定することができます。

- 非同期モードを指定した場合、sendGatewayMessage 関数は、SMSC に送信するためにゲートウェイによってメッセージがサービスコードに送信されると、空の文字列を返します。この時点以降でエラーが発生した場合は、ログに記録されます。たとえば、ゲートウェイによって SMSC に送信されたメッセージがタイムアウトになった場合や、ゲートウェイがエラー応答を取得した場合などはログに記録されます。アプリケーションがエラーについて通知を受けることはありません。
- 同期モード (デフォルト) を指定した場合、sendGatewayMessage 関数は、ゲートウェイが SMSC から応答を取得するまで、または通信試行がタイムアウトになるまで、処理を返しません。メッセージが正常に送信された場合、この関数は SMPP メッセージ ID 文字列を返します。エラーが発生した場合、関数はエラー文字列を返します。

同期モードは、メッセージが SMSC に届いたかどうかを確認する必要がある場合に使用します。また、配信受信をリクエストする場合にも、同期モードを使用します。

注意 : 同期モードを使用している場合、SMSC から返された 16 進文字列の messgeID は、ColdFusion によって自動的に 10 進数の値に変換されます。

次の例は、1288 ページの「submit コマンド」で説明されている「例 : sendGatewayMessage 関数での submit コマンドの使用」の内容を拡張したものです。この例では、空以外の戻り値を確認し、SMS によって返されるメッセージ番号を表示します。また、この例では、ColdFusion で設定された SMS ゲートウェイを使用します。SendGatewayMessage 関数で指定したゲートウェイを変更する場合は、ゲートウェイの設定ファイルで同期モードが指定されていることを確認してください。

```
<h3>Sending SMS From a Web Page Example</h3>

<cfif IsDefined("form.oncethrough") is "Yes">
    <cfif IsDefined("form.SMSMessage") is True AND form.SMSMessage is not "">
         <h3>Sending a Text Message: </h3>
            <cfoutput>#form.SMSMessage#</cfoutput><br><br>

        <cfscript>
            /* Create a structure that contains the message. */
            msg = structNew();
            msg.command = "submit";
            msg.destAddress = "5551234";
            msg.shortMessage = form.SMSMessage;
            ret = sendGatewayMessage("SMS Menu App - 5551212", msg);
        </cfscript>
    </cfif>
    <cfif isDefined("ret") AND ret NEQ "">
        <h3>Text message sent</h3>
        <cfoutput>The Message Id is: #ret#</cfoutput>
        <br><br>
    </cfif>
    <hr noshade>
</cfif>

<!--- begin by calling the cfform tag --->
<cfform>
    SMS Text Message: <cfinput type="Text" name="SMSMessage"
        value="Sample text Message" required="No" maxlength="160">
    <p><input type = "submit" name = "submit" value = "Submit">
    <input type = "hidden" name = "oncethrough" value = "Yes">
</cfform>
```

SMS 発信に関連するオプションパラメータ

ベンダー固有のオプションパラメータは、ColdFusion SMS ゲートウェイ経由で送信できます。

オプションパラメータを設定するには、optionalparameter 属性を使用します。

ゲートウェイで受信したメッセージにオプションパラメータが含まれていた場合、それらのパラメータはリスナー CFC メソッド onIncomingMessage に返されるデータ構造体の optionalParameters キーに格納されます。

次のコードは、オプションパラメータを追加する方法を示しています。

```
params=StructNew();
params["parameter"]=BinaryDecode("string","binaryencoding");
params["parameter"]=CharsetDecode("string, encoding");
outgoingSMS.optionalParameters=params;
```

- parameter : ベンダー固有のオプションパラメータです。
- BinaryDecode :『CFML リファレンス』の BinaryDecode を参照してください。
- CharsetDecode :『CFML リファレンス』の CharsetDecode を参照してください。

オプションパラメータが 1 つのみの場合は、代わりに次のコードを使用できます。

```
out.optionalParameter=parameter;
out.optionalParameterValue="value";
```

注意 : Java Short.decode(String) 関数でキーを解析できること、または値がバイトであることを確認してください。

メッセージ処理通知

メッセージがどのように処理されたかを示すメッセージ処理応答を返すように、SMSC にリクエストすることができます。配信受信をリクエストする場合は、SendGatewayMessage 関数の **Data** パラメータや CFC リスナーメソッドの戻り変数で、RegisteredDelivery フィールドを設定します。このフィールドには次の値を設定できます。

値	説明
0	(デフォルト) 配信情報を返しません。
1	タイムアウトになる前にメッセージが配信されなかった場合に、配信受信を返します。
2	メッセージが配信された場合、または配信されなかった場合に配信受信を返します。

一部のプロバイダでは、中間配信通知もサポートされています。詳細については、プロバイダのマニュアルを参照してください。

配信通知を使用するには、同期モードを使用してメッセージを送信してメッセージ ID を取得します。着信メッセージのルーチンが配信受信を処理できる必要があります (1286 ページの「着信メッセージの処理」を参照)。

有効期間

SMSC がメッセージを保持して配信を試行する期間を変更できます。通常、デフォルト値は 72 時間です。たとえば、緊急時の作業者に送信するメッセージの場合、短い有効期間 (15 分など) が適切です。この値を変更するには、SendGatewayMessage 関数の **Data** パラメータや CFC リスナーメソッドの戻り変数に、validityPeriod フィールドを設定します。期間を指定するには、**YYMMDDhhmmsst**00R という形式を使用します。この形式の **t** は 10 分の 1 秒を示し、00R は、これが日時の値ではなく相対的な期間であることを指定します。たとえば、000001063000000R では、0 年 0 か月 1 日 6 時間 30 分の有効期間が指定されます。

ColdFusion の SMS 開発ツール

ColdFusion には、次に示す SMS アプリケーション開発ツールが用意されています。

- SMS テストサーバー
- SMS クライアントシミュレーター

SMS テストサーバー

ColdFusion SMS テストサーバーは、軽量の SMSC シミュレーターです。TCP/IP ポート 7901 で、ColdFusion SMS ゲートウェイや SMS クライアントシミュレーターなどの他の SMS リソースからの SMPP 接続リクエストをリスンします。これらのリソースからは、ユーザー名、パスワード、および電話番号 (アドレス) が提供されます。このユーザー名とパスワードは、シミュレーターの設定ファイルで指定されている名前とパスワードに一致する必要があります (後述します)。

SMS テストサーバーは、接続を確立した後、着信メッセージをリスンします。そして、宛先アドレスが既存の SMPP 接続に一致している場合に、そのメッセージを指定の宛先アドレスに転送します。

SMS テストサーバーを使用すれば、通信プロバイダなどの外部 SMSC サプライヤが利用できない場合でも、SMS アプリケーションを開発できます。SMS テストサーバーでは、ColdFusion SMS ゲートウェイの submit および submitMulti コマンドがサポートされています。SMS ゲートウェイの data コマンドを使用して送信されたメッセージも受け付けますが、このメッセージは配信しません。このサーバーには、ストアアンドフォワード機能は搭載されていません。

SMS テストサーバーを起動するには、ColdFusion Administrator の [イベントゲートウェイ] 領域の [設定] ページの [SMS テストサーバーの起動] ボタンをクリックします。

注意：ColdFusion を再起動しても、SMS テストサーバーは自動的に再起動しません。ColdFusion を再起動した場合は、手動でテストサーバーを再起動します。

SMS テストサーバーは、J2EE 設定の場合は "<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥lib¥sms-test-users.txt" ファイルを、サーバー設定の場合は "<ColdFusion のルートディレクトリ >¥lib¥sms-test-users.txt" ファイルを読み込んで、有効なユーザー名とパスワードを取得します。ColdFusion で用意されているこのファイルには、名前とパスワードの組み合わせがいくつか設定されています。その組み合わせの 1 つとして、ユーザー名 cf とパスワード cf があります。このファイルを編集して、エントリの追加や削除が行えます。このファイルには、テストサーバーに接続する各ユーザーの名前とパスワードのエントリを設定する必要があります。それぞれのユーザーエントリは、次のように、空の行で区切ります。

```
name=cf
password=cf

name=user1
password=user1
```

SMS クライアントシミュレーター

ColdFusion SMS クライアントシミュレーターは、(機能が限定された) 携帯電話をシミュレートする単純な ESME (External Short Message Entity) です。SMS テストサーバーに接続して、メッセージを交換することができます。

注意：UNIX および Linux システムでは、クライアントシミュレーターは X-Window を必要とします。

シミュレーターを使用するには、次の手順を行います。

SMS シミュレーターの使用

1. SMS テストサーバーが起動しており、ColdFusion Administrator で SMS イベントゲートウェイインスタンスが設定および起動されていることを確認します。
2. Windows の場合は "SMSClient.bat" を、UNIX または Linux の場合は "SMSClient.sh" を実行します。これらのファイルは、J2EE 設定では "<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥bin" ディレクトリに、サーバー設定の場合は "<ColdFusion のルートディレクトリ >¥bin" ディレクトリにあります。

   たとえば Apple Mac OS X システムに、ColdFusion の純粋な Java バージョンをインストールした場合は、次のコマンドを入力してシミュレーターを起動します。

   ```
   java -jar cf_root/WEB-INF/cfusion/lib/smpp.jar
   ```

3. このデバイスで使用するサーバー、ポート、ユーザー名、パスワード、および電話番号の入力を求めるダイアログボックスが表示されます。シミュレーターはこの電話番号をソースアドレスとして送信します。そして、宛先アドレスとしてこの番号が使用されている、SMSC サーバーから送信された SMS メッセージを受け付けます。

   。また、"<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥lib¥sms-test-users.cfg" ファイル、または "<ColdFusion のルートディレクトリ >¥lib¥sms-test-users.cfg" ファイルに設定されている任意のユーザー名とパスワードの組み合わせも指定できます。

4. [Connect] をクリックします。
5. SMS デバイスシミュレータークライアントが表示されます。SMS イベントゲートウェイの設定ファイルのアドレス範囲プロパティで指定されている電話番号のうち、メッセージを送信したい電話番号を、[Send SMS To] フィールドに入力します。
6. メッセージフィールド ([Send] ボタンの左側) に、メッセージを直接入力するか、シミュレーターキーパッドを使用してメッセージを入力します。
7. [Send] ボタンをクリックします。

クライアントシミュレーターの [Connection] メニューには、SMSC サーバーに接続するためのオプション、SMSC サーバーから切断するためのオプション、および接続を確認するためのオプションが用意されています。接続情報は、クライアントの一番下にあるステータス行に表示されます。

サンプル SMS アプリケーション

次の CFC では、単純な従業員電話番号ディレクトリ検索アプリケーションを実装します。ユーザーは、検索する名前の一部を含んだメッセージを送信します (スペースの場合、すべての名前を要求します)。onIncomingMessage の応答は、一致件数によって異なります。

- 一致する項目がない場合、onIncomingMessage 関数は、一致する項目がないことを示すメッセージを返します。
- 一致件数が 1 件の場合は、名前、部門、および電話番号を返します。
- 一致件数が 10 件以下の場合は、番号が振られた名前リストを返します。ユーザーは、この番号を入力して詳細情報を取得できます。
- 一致件数が 10 件を超える場合は、最初の 10 件の名前リストのみを返します。より複雑なアプリケーションでは、複数のリストを取得してすべての名前にアクセスできるようにすることも可能です。
- 複数一致リストを取得した後にユーザーが番号を入力すると、その番号に対応する名前に関する情報を返します。

次にこの CFC のコードを示します。

```
<cfcomponent>
    <cffunction name="onIncomingMessage">
        <cfargument name="CFEvent" type="struct" required="YES">
        <!--- Remove any extra white space from the message. --->
        <cfset message =Trim(arguments.CFEvent.data.MESSAGE)>
        <!--- If the message is numeric, a previous search probably returned a
            list of names. Get the name to search for from the name list stored in
            the Session scope. --->
        <cfif isNumeric(message)>
            <cfscript>
                if (structKeyExists(session.users, val(message))) {
                    message = session.users[val(message)];
                }
            </cfscript>
        </cfif>

        <!--- Search the database for the requested name. --->
        <cfquery name="employees" datasource="cfdocexamples">
            select FirstName, LastName, Department, Phone
            from Employees
            where 0 = 0
            <!--- A space indicates the user entered a first and last name. --->
            <cfif listlen(message, " ") eq 2>
                and FirstName like '#listFirst(message, " ")#%'
                and LastName like '#listlast(message, " ")#%'
            <!--- No space: the user entered a first or a last name. --->
            <cfelse>
                and (FirstName like '#listFirst(message, " ")#%'
                or LastName like '#listFirst(message, " ")#%')
            </cfif>
        </cfquery>

        <!--- Generate andreturn the message.--->
        <cfscript>
            returnVal = structNew();
            returnVal.command = "submit";
            returnVal.sourceAddress = arguments.CFEVENT.gatewayid;
            returnVal.destAddress = arguments.CFEVENT.originatorid;

            //No records were found.
            if (employees.recordCount eq 0) {
                returnVal.shortMessage = "No records found for '#message#'";
            }
```

```
            //One record was found.
            else if (employees.recordCount eq 1) {
            // Whitespace in the message text results in bad formatting,
            // so the source cannot be indented.
                returnVal.shortMessage = "Requested information:
#employees.firstName# #employees.lastName#
#employees.Department#
#employees.Phone#";
            }
            //Multiple possibilities were found.
            else if (employees.recordCount gt 1) {
                //If more than ten were found, return only the first ten.
                if (employees.recordCount gt 10)
                {
                    returnVal.shortMessage = "First 10 of #employees.recordCount# records";
                }else{
                    returnVal.shortMessage = "Records found: #employees.recordCount#";
                }
                // The session.users structure contains the found names.
                // The record key is a number that is also returned in front of the
                // name in the message.
                session.users = structNew();
                for(i=1; i lte min(10, employees.recordCount); i=i+1)
                {
                    // These two lines are formatted to prevent extra white space.
                    returnVal.shortMessage = returnVal.shortMessage & "
#i# - #employees.firstName[i]# #employees.lastName[i]#";
                    // The following two lines must be a single line in the source
                    session.users[i]="#employees.firstName[i]# #employees.lastName[i]#";
                }
            }
            return returnVal;
        </cfscript>
    </cffunction>
</cfcomponent>
```

FMS イベントゲートウェイの使用

FMS イベントゲートウェイは、Flash Media Server 2 と Adobe ColdFusion サーバーとの間のインターフェイスを提供します。したがって、ColdFusion アプリケーションと Adobe Flash クライアントは同じデータを共有できます。

ただし、このゲートウェイを使用するには、ColdFusion イベントゲートウェイの原理やプログラミングテクニックに慣れておく必要があります (1242 ページの「イベントゲートウェイの使用」を参照)。また、Flash Media Server の基本的な知識があると役立ちます。

Flash Media Server について

Flash Media Server 2 は Flash Communication Server の最新バージョンです。Flash Media Server 2 に用意されている従来のストリーミングメディア機能と柔軟な開発環境を使用すれば、革新的な対話型メディアアプリケーションを作成して配布できます。Flash Media Server では、次のメディアエクスペリエンスを作成して配布できます。

- ビデオオンデマンド
- ライブ Web イベントブロードキャスト
- MP3 ストリーミング
- ビデオブログ

- ビデオメッセージング
- マルチメディアチャット環境

Flash Media Server の詳細とダウンロードについては、Adobe の次の Web サイトを参照してください。
www.adobe.com/go/learn_cfu_flashmediaserver_jp

FMS ゲートウェイを介した ColdFusion と Flash Media Server 間の通信の仕組み

FMS イベントゲートウェイを使うと、ColdFusion アプリケーションまたは Flash クライアントを通してデータを変更し、その変更を Flash Media Server 共有オブジェクトに反映できます。FMS イベントゲートウェイは共有オブジェクトをリスンし、他のクライアントによって共有オブジェクトが変更されたら ColdFusion に通知します。また、ColdFusion が FMS イベントゲートウェイを使って共有オブジェクトを変更することもできます。

ColdFusion には、次に示す FMS アプリケーション開発ツールが用意されています。

FCSj.jar Flash Media Server と通信するための Java API が実装されている JAR ファイル。

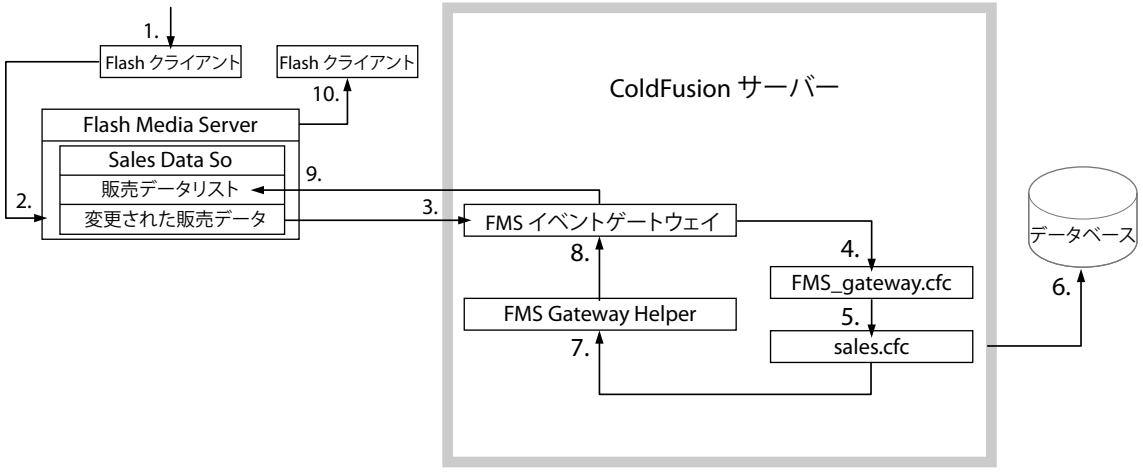
FMSGateway FMS イベントゲートウェイタイプ用のクラス。FMS アプリケーションを実装するには、FMSGateway クラスのインスタンスを使用して Flash Media Server と通信する ColdFusion アプリケーションを作成します。

Flash クライアントでのデータの変更

FMS イベントゲートウェイは Flash Media Server 共有オブジェクトをリスンし、Flash クライアントが共有オブジェクトを変更すると ColdFusion に通知します。Flash クライアントで Flash Media Server 共有オブジェクトを変更すると、次の処理が行われます。

1. ユーザーが Flash クライアントでデータを変更します。
2. Flash Media Server が適切な共有オブジェクトを更新します。
3. Flash Media Server が FMS イベントゲートウェイに通知します。
4. FMS イベントゲートウェイが、ColdFusion アプリケーションの CFC の適切なメソッドを呼び出します。これらの CFC によって必要なアクションがすべて実行されます。このアクションには、共有オブジェクトを更新するように FMS Gateway Helper に通知することも含まれます。
5. FMS Gateway Helper が、共有オブジェクトを更新するためのメッセージを FMS イベントゲートウェイに送信します。
6. FMS イベントゲートウェイが共有オブジェクトを更新します。
7. Flash Media Server が、共有オブジェクトを変更したことをすべての Flash クライアントに通知します。その結果、Flash クライアントに変更が反映されます。

次の図は、Flash Media Server、FMS イベントゲートウェイ、ColdFusion アプリケーション間の通信を示しています。



ColdFusion アプリケーションでのデータの変更

FMS イベントゲートウェイを介して、ColdFusion アプリケーションで Flash Media Server 共有オブジェクトを変更できます。共有オブジェクトに影響するデータを ColdFusion アプリケーションで変更すると、次の処理が行われます。

1. ユーザーが ColdFusion ページを使用して、変更するデータを含むフォームを送信します。
2. ColdFusion ページが、データベースを更新するメソッドが含まれている適切な CFC を呼び出します。
3. CFC のメソッドがデータベースを更新し、FMS Gateway Helper のメソッドを呼び出します。
4. FMS Gateway Helper が FMS イベントゲートウェイを呼び出して、適切な共有オブジェクトを更新するように通知します。
5. Flash Media Server が適切な共有オブジェクトを更新します。
6. Flash Media Server が、共有オブジェクトを変更したことを Flash クライアントに通知します。
7. 必要に応じて、Flash クライアントがその内容を変更します。

アプリケーションの開発およびデプロイのプロセス

FMS イベントゲートウェイを使用したアプリケーションを開発およびデプロイする一般的なプロセスは、次のようになります。

1. アプリケーションを設計します。
2. Flash Media Server を使用するように FMS イベントゲートウェイインスタンスを設定します。
3. ColdFusion の CFC、CFM ページ、およびその他のアプリケーション要素を記述します。
4. ColdFusion アプリケーションのテストに使用する、Flash Media Server 共有オブジェクトを操作する Flash クライアントを作成または識別します。
5. Flash Media Server と Flash クライアントを使用してアプリケーションをテストします。
6. アプリケーションを一般に提供します。

FMS イベントゲートウェイの設定

FMS イベントゲートウェイ固有の設定情報は、設定ファイルを使用して FMS イベントゲートウェイに提供します。ColdFusion Administrator で FMS イベントゲートウェイインスタンスを設定するときに、設定ファイルの場所を指定します。設定ファイルでは、Flash Media Server アプリケーションの URL と、Flash Media Server 共有オブジェクトの名前を指定します。設定ファイルの例を次に示します。

```
#
# FMS event gateway configuration
#

# This is the URL to the Flash Media Server application.
rtmpurl=rtmp://localhost/SalesDataApp

# This is the shared object you would like this gateway to connect and listen to.
sharedobject=SalesDataSO
```

FMS イベントゲートウェイの GatewayHelper クラスのメソッド

次の表に、FMS イベントゲートウェイの GatewayHelper クラスのメソッドを示します。

メソッド	説明
setProperty	Flash Media Server 共有オブジェクトのプロパティを設定します。使用できるパラメータは次のとおりです。 name : 共有オブジェクトの名前を含む文字列。 value : 共有オブジェクト。
getProperty	Flash Media Server 共有オブジェクトのプロパティを取得します。使用できるパラメータは次のとおりです。 name : 共有オブジェクトの名前を含む文字列。

データ変換

ColdFusion と Flash Media Server では異なるデータ型が使用されているので、データの受け渡しにはデータの変換が必要です。Flash Media Server には、数値、文字列、ブール値といった基本的なデータ型に加えて、ColdFusion クエリー、構造体、配列も渡すことができます。ColdFusion クエリーオブジェクトは、java.util.HashMap の配列として Flash Media Server に渡します。配列内のそれぞれの HashMap オブジェクトには、クエリーの行ごとに、列の名前と値を表すキーと値のペアが含まれています。ColdFusion 配列を Flash Media Server に渡すと、その配列が FMS イベントゲートウェイによってオブジェクトの Java 配列に変換されます。ColdFusion 構造体を渡す場合は、変換は必要ありません。

FMS イベントゲートウェイで、共有オブジェクトでの CFC の受け渡しはサポートされていません。

Data Services Messaging イベントゲートウェイの使用

Adobe ColdFusion に用意されている Data Services Messaging イベントゲートウェイタイプを使用すれば、LiveCycle Data Services ES との間でメッセージを送受信するアプリケーションを作成できます。Data Services Messaging ゲートウェイを設定し、そのイベントゲートウェイを使用するアプリケーションを作成およびテストします。

ただし、Data Services Messaging ゲートウェイを使用するには、ColdFusion イベントゲートウェイの原理やプログラミングテクニックに慣れておく必要があります (1242 ページの「イベントゲートウェイの使用」を参照)。また、Adobe LiveCycle Data Services ES にも精通している必要があります。

Flex と ColdFusion について

ColdFusion には、Data Services Messaging イベントゲートウェイが用意されています。これは、ColdFusion Event Gateway Adapter を使用して LiveCycle Data Services ES との間でメッセージの送受信を行います。これを使用すれば、ColdFusion アプリケーションと Flex アプリケーションが同じイベントキューからイベントの公開や利用ができます。

注意： Data Services Messaging イベントゲートウェイを使用して Flex アプリケーションとやり取りするには、Flex アプリケーションを LiveCycle Data Services ES 上で実行する必要があります。

ColdFusion と Flex 間の通信の仕組み

Data Services Messaging イベントゲートウェイを介して、ColdFusion アプリケーションから Flex アプリケーションにメッセージを送信できます。逆に、Flex アプリケーションから ColdFusion アプリケーションにメッセージを送信することもできます。

メッセージの送信は、ColdFusion アプリケーションと Flex アプリケーションのどちらからでも開始できます。

1. Flex アプリケーションがメッセージを生成します。
2. Flex Message Service がメッセージを ColdFusion Event Gateway Adapter に渡します。
3. ColdFusion Event Gateway Adapter が Java RMI (Remote Method Invocation) を使用してメッセージを Data Services Messaging イベントゲートウェイに送信します。
4. Data Services Messaging イベントゲートウェイと ActionScript トランスレータが ActionScript 3.0 のデータ型を適切な ColdFusion の値に変換し、メッセージをイベントゲートウェイキューに追加します。
5. ColdFusion サーバーが Data Services Messaging イベントゲートウェイリスナー CFC の onIncomingMessage メソッドを呼び出します。
6. ColdFusion アプリケーションがメッセージを生成し、ColdFusion サーバーに送信します。
7. ColdFusion サーバーがメッセージを Data Services Messaging イベントゲートウェイに送信します。
8. Data Services Messaging イベントゲートウェイと ActionScript トランスレータが ColdFusion の値を適切な ActionScript 3.0 のデータ型に変換し、ゲートウェイがメッセージを ColdFusion Event Gateway Adapter に送信します。
9. ColdFusion Event Gateway Adapter がメッセージを Flex Message Service に送信します。
10. Flex Message Service がメッセージを Flex アプリケーションに渡します。

注意： RMI レジストリは、ColdFusion Event Gateway Adapter と Data Services Messaging イベントゲートウェイとの間の通信を実現するもので、ポート 1099 を使用します。これは Java RMI のデフォルトポートです。このポート番号は変更できません。RMI レジストリが LiveCycle Data Services と ColdFusion の両方に確実にレジストリサービスを提供できるようにするには、最初に LiveCycle Data Services を起動してから ColdFusion を起動します。Flex を停止した場合は、LiveCycle Data Services を再起動してからゲートウェイを再起動します。

アプリケーションの開発およびデプロイのプロセス

Data Services Messaging イベントゲートウェイを介して Flex アプリケーションと通信する ColdFusion アプリケーションを開発およびデプロイする一般的なプロセスは、次のとおりです。

1. アプリケーションを設計します。
2. Data Services Messaging イベントゲートウェイインスタンスを設定します。
3. ColdFusion の CFC、CFM ページ、およびその他のアプリケーション要素を記述します。
4. Flex Enterprise Services 2 を使用してアプリケーションをテストします。
5. アプリケーションを一般に提供します。

Data Services Messaging イベントゲートウェイの設定

Data Services Messaging イベントゲートウェイのインスタンスの設定は、設定ファイルを作成し、イベントゲートウェイインスタンスの作成時にそのファイルを設定ファイルとして指定することで行えます。また、Flex アプリケーションから送信するメッセージの中で設定情報を渡すこともできます。設定ファイルでは、次のいずれかを行うように、Data Services Messaging イベントゲートウェイに設定情報を渡します。

- Data Services Messaging イベントゲートウェイから別のコンピュータ上の Flex Enterprise Services 2 にメッセージを送信する
- Data Services Messaging イベントゲートウェイを特定の Flex 宛先で使用し、メッセージで指定されている宛先はすべて無視する

Data Services Messaging イベントゲートウェイの設定ファイルは、単純な Java プロパティファイルで、次のプロパティが含まれます。

プロパティ	説明
destination	固定した宛先。この値を指定した場合、メッセージ内の宛先情報は無視されます。
host	Flex Enterprise Services 2 サーバーのホスト名または IP アドレス。

設定ファイルの例を次に示します。

```
#
# Flex event gateway configuration
#

# This is the destination of the messages.
destination=Gateway1

# host name or IP address of the Flex Enterprise Server
host=127.0.0.1
```

設定ファイルを作成する場合は、拡張子 .cfg を使用して、"<ColdFusion のルートディレクトリ >/gateway/config/" ディレクトリに保存します。

発信メッセージの送信

ColdFusion アプリケーションが Flex アプリケーションにメッセージを送信するときには、次の処理が行われます。

1 ColdFusion アプリケーションが、リスナー CFC のリスナーメソッドで cfreturn タグを使用するか、または ColdFusion の SendGatewayMessage 関数を呼び出して、発信メッセージを送信します。

2 発信メッセージが送信されると、Data Services Messaging ゲートウェイのメソッドが呼び出されます。

CFML から送信される発信メッセージでは、次の構造体メンバーが Flex メッセージに変換されます。

名前	内容
body	メッセージの本文。必須。
CorrelationID	メッセージの相関 ID。
Destination	メッセージの Flex 宛先。設定ファイルで指定していない場合は必須です。
Headers	メッセージにヘッダがある場合、CFML 構造体にはキーとしてのヘッダ名とヘッダの値が格納されます。
LowercaseKeys	値が yes に設定されている場合、構造体のキーは ActionScript の型の作成時に小文字に変換されます。
TimeToLive	このメッセージが有効な期間をミリ秒単位で指定します。

また、Data Services Messaging イベントゲートウェイは次の Flex メッセージフィールドに値を自動的に設定します。

名前	内容
MessageID	メッセージを識別する UUID。
Timestamp	メッセージの送信時刻。
ClientID	Data Services Messaging イベントゲートウェイインスタンスの ID。

注意：Data Services Messaging イベントゲートウェイは、ColdFusion Event Gateway Adapter に登録された任意の宛先にメッセージを送信できます。ただし、Data Services Messaging ゲートウェイの設定ファイルで宛先が設定されている場合、メッセージの宛先は無視されます。

発信メッセージの送信例

次の CFM ページからの抜粋では、メッセージを含む構造体を作成しています。destination は、"flex-services.xml" ファイルで指定されている、メッセージの送信先となる Data Services Messaging イベントゲートウェイの宛先 ID です。body はメッセージ本文です。CFML の sendGatewyMessage 関数はメッセージをゲートウェイに送信します。

```
<cfset success = StructNew()>
<cfset success.msg = "E-mail was sent at " & Now()>
<cfset success.Destination = "gateway1">
<cfset ret = SendGatewayMessage("Flex2CF2", success)>
```

プロパティが大文字小文字を正しく保持するよう、Flex 関連の情報は次のように定義します。

```
myStruct['mySensitiveProp']['myOtherSensitiveProp']
```

次に示すのは、ヘッダーを使用して宛先の特定のサブトピックに送信する例です。

```
<cfset var msg = structnew()>
<cfset msg.destiNation = 'ColdFusionGateway'>
<cfset msg.body = 'somebody'>
<cfset msg['headers']['DSSubtopic'] = 'somesubtopic'>
<cfset sendgatewaymessage('CF2FLEX2' , msg)>
```

着信メッセージの処理

Flex アプリケーションがメッセージを ColdFusion アプリケーションに送信すると、Data Services Messaging イベントゲートウェイは設定された CFC の onIncomingMessage 関数に CFEvent 構造体を送信し、次の情報がイベントのデータに関連付けられます。

名前	内容
body	メッセージの本文。
ClientID	メッセージを送信したクライアントの ID。
CorrelationID	メッセージの相関 ID。
Destination	メッセージの Flex 宛先。
Headers	メッセージにヘッダがある場合、CFML 構造体にはキーとしてのヘッダ名とヘッダの値が格納されます。
Timestamp	メッセージのタイムスタンプ。

着信メッセージのデータ構造体には Flex メッセージの messageID と timeToLive の値も格納されています。

着信メッセージの処理例

次の例では、Flex アプリケーションのメッセージ本文に含まれているデータを構造体に格納します。続いて、この構造体の内容を使用して電子メールを生成します。

```
<cfcomponent displayname="SendEmail" hint="Handles incoming message from Flex">
    <cffunction name="onIncomingMessage" returntype="any">
    <cfargument name="event" type="struct" required="true">

        <!--- Create a structure to hold the message object sent from Flex--->
        <cfset messagebody = event.data.body>

        <!--- Populate the structure. --->
        <cfset mailfrom="#messagebody.emailfrom#">
        <cfset mailto="#messagebody.emailto#">
        <cfset mailsubject="#messagebody.emailsubject#">
        <cfset mailmessage ="#messagebody.emailmessage#">

        <!--- Send e-mail with values from the structure. --->
        <cfmail from="#mailfrom#"
            to="#mailto#"
            subject="#mailsubject#">
            <cfoutput>#mailmessage#</cfoutput>
        </cfmail>
    </cffunction>
</cfcomponent>
```

Flex アプリケーションが本文ではなくヘッダを使用してメッセージを送信する場合は、次の例のように構造体を作成して値を設定します。

```
<cfset messageheader = StructNew()>
<cfset messageheader.sendto = event.data.headers.emailto>
<cfset messageheader.sentfrom = event.data.headers.emailfrom>
<cfset messageheader.subject = event.data.headers.emailsubject>
<cfset messageheader.mailmsg = event.data.headers.emailmessage>

<cfset mailfrom="#messageheader.sentfrom#">
<cfset mailto="#messageheader.sendto#">
<cfset mailsubject="#messageheader.subject#">
<cfset mailmessage ="#messageheader.mailmsg#">
```

ColdFusion 9.0.1 で導入された新しいメソッド

ColdFusion Messaging Gateway CFC に、次の新しいメソッドが導入されました。

- allowSend
- allowSubscribe

  いずれのメソッドも、subtopic のパラメータを持っています。

  これらのメソッドは、特定のサブトピックへのデータのサブスクライブおよび送信を制御するのに便利です。

注意：各ゲートウェイ CFC でこれらのメソッドを呼び出すには、messaging-config.xml (Web_INF/Flex) のメッセージ送信先の下にゲートウェイ ID を指定します。デフォルトでは、値は * です。

データ変換

次の表に、ColdFusion のデータ型と、それに対応する Flash または ActionScript のデータ型を示します。

ColdFusion のデータ型	Flash のデータ型
文字列	String
配列	[] = 配列
構造体	{} = 型指定なしのオブジェクト
クエリー	ArrayCollection
CFC	クラス = 型指定ありのオブジェクト (対応する ActionScript クラスが存在する場合。それ以外の場合は、ActionScript の型指定なしの汎用オブジェクト (map) になります)
CFC の日付	ActionScript の日付
CFC の文字列	ActionScript の文字列
CFC の数値	ActionScript の数値
ColdFusion XML オブジェクト	ActionScript XML オブジェクト

Data Management イベントゲートウェイの使用

Adobe ColdFusion に用意されている Data Management イベントゲートウェイタイプを使用すれば、宛先で管理しているデータが変更されたときに、ColdFusion アプリケーションから Adobe Flex アプリケーションに通知されるようにすることができます。Data Management イベントゲートウェイを設定し、そのイベントゲートウェイを使用するアプリケーションを作成します。

ただし、Data Management イベントゲートウェイを使用するには、ColdFusion イベントゲートウェイの原理やプログラミングテクニックに慣れておく必要があります (1242 ページの「イベントゲートウェイの使用」を参照)。また、LiveCycle Data Services ES にも精通している必要があります。

ColdFusion と Flex について

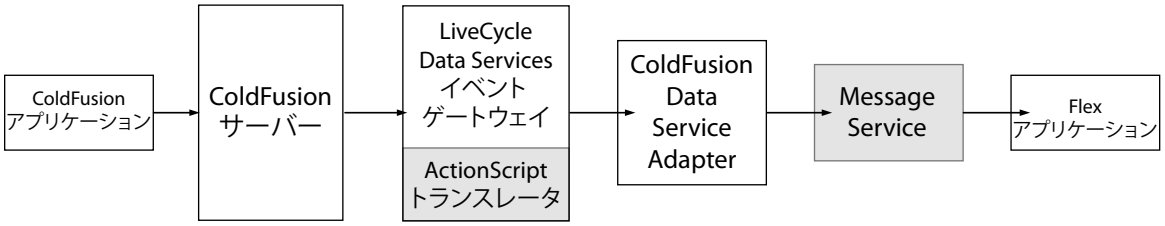
ColdFusion には、Data Management イベントゲートウェイが用意されています。このイベントゲートウェイは、ColdFusion Data Service Adapter を使用して LiveCycle Data Services ES との間でメッセージの送受信を行います。これを使用すれば、宛先によって管理されているデータの変更を、ColdFusion アプリケーションから Flex アプリケーションに通知することができます。

注意：Data Management イベントゲートウェイを使用して Flex アプリケーションにメッセージを送信するには、Flex アプリケーションを LiveCycle Data Services ES 上で実行する必要があります。

ColdFusion と Flex 間の通信の仕組み

Data Management イベントゲートウェイを介して、ColdFusion アプリケーションから Flex アプリケーションにメッセージを送信できます。このゲートウェイタイプでは、ColdFusion アプリケーションから Flex アプリケーションへのメッセージ送信のみが可能です。

次の図は、ColdFusion アプリケーションから Flex アプリケーションにメッセージを送信するプロセスを示しています。



1. ColdFusion アプリケーションがメッセージを生成し、ColdFusion サーバーに送信します。
2. ColdFusion サーバーがメッセージを Data Management イベントゲートウェイに送信します。
3. Data Management イベントゲートウェイと ActionScript トランスレータが ColdFusion の値を適切な ActionScript 3.0 のデータ型に変換し、ゲートウェイがメッセージを ColdFusion Data Service Adapter に送信します。
4. ColdFusion Data Service Adapter がメッセージを LiveCycle Data Services Message Service に送信します。
5. Message Service がメッセージを Flex アプリケーションに渡します。

   LiveCycle Data Services ES を ColdFusion サーバーで実行する場合、LiveCycle Data Services ES と ColdFusion の間の通信に RMI は使用されません。

   LiveCycle Data Services ES をリモートで実行する場合、RMI レジストリが LiveCycle Data Services ES と ColdFusion の両方に確実にレジストリサービスを提供できるようにするには、最初に LiveCycle Data Services ES を起動してから ColdFusion を起動します。LiveCycle Data Services ES を停止した場合は、LiveCycle Data Services ES を再起動してからゲートウェイを再起動します。

   LiveCycle Data Services ES をリモートで実行している場合、RMI レジストリによる ColdFusion Data Service Adapter と Data Management イベントゲートウェイとの間の通信でポート 1099 が使用されます。このポートは Java RMI のデフォルトポートです。ポート番号を変更するには、ColdFusion サーバーと Flex サーバーの両方で、Java JVM 引数に -Dcoldfusion.rmiport=1234 を追加します。1234 は適切なポート番号に置き換えます。

アプリケーションの開発およびデプロイのプロセス

Data Management イベントゲートウェイを介して Flex アプリケーションと通信する ColdFusion アプリケーションを開発およびデプロイする一般的なプロセスは、次のとおりです。

1. アプリケーションを設計します。
2. Data Management イベントゲートウェイインスタンスを設定します。
3. ColdFusion の CFC、CFM ページ、およびその他のアプリケーション要素を記述します。
4. LiveCycle Data Services ES を使用してアプリケーションをテストします。
5. アプリケーションを一般に提供します。

Data Management イベントゲートウェイの設定

Data Management イベントゲートウェイのインスタンスの設定は、設定ファイルを作成し、イベントゲートウェイインスタンスの作成時にそのファイルを設定ファイルとして指定することで行えます。また、メッセージの中で設定情報を渡すこともできます。設定ファイルでは、次のいずれかを行うように、Data Management イベントゲートウェイに設定情報を渡します。

- Data Management イベントゲートウェイから別のコンピュータ上の LiveCycle Data Services ES にメッセージを送信する

- Data Management イベントゲートウェイを特定の Flex 宛先で使用し、メッセージで指定されている宛先はすべて無視する

Data Management イベントゲートウェイの設定ファイルは、単純な Java プロパティファイルで、次のプロパティが含まれます。

プロパティ	説明
destination	固定した宛先。この値を指定した場合、メッセージ内の宛先情報は無視されます。
host	LiveCycle Data Services ES サーバーのホスト名または IP アドレス。LiveCycle Data Services ES を ColdFusion の一部として実行している場合は、ホスト名を省略します。

設定ファイルの例を次に示します。

```
#
# Data Management event gateway configuration
#
# This is the destination where messages are sent.
destination=myDestination

# Host name or IP address of the LiveCycle Data Services ES Server
host=127.0.0.1
```

設定ファイルを作成する場合は、拡張子 .cfg を使用して、"<ColdFusion のルートディレクトリ >/gateway/config/" ディレクトリに保存します。

注意：Data Management イベントゲートウェイは、ColdFusion Data Service Adapter に登録された任意の宛先にメッセージを送信できます。ただし、Data Management イベントゲートウェイの設定ファイルで宛先が設定されている場合、メッセージの宛先は無視されます。

メッセージの送信

ColdFusion アプリケーションは ColdFusion の SendGatewayMessage 関数を呼び出して、Flex アプリケーションにメッセージを送信します。CFML から送信されるメッセージでは、次の構造体メンバーが Flex メッセージに変換されます。

名前	内容
destination	メッセージの宛先。このエントリは、設定ファイルで指定していない場合は必須です。
action	必須。実行している通知アクション (create、delete、deleteID、refreshfill、または update)。
item	action="create" または action="delete" のときに必須。追加または削除されたレコード。
identity	action="deleteID" のときに必須。削除されたレコードの ID プロパティ (プライマリキー) を含む構造体。
fillparameters	オプション。更新する fill 操作を指定する fill パラメータを含む配列。
newversion	action="update" のときに必須。更新されたレコード。
previousversion	オプション。更新される前のレコード。このエントリは競合の解決のために使用されます。
changes	action="update" のときはオプション。レコード内で変更されたプロパティ名のカンマ区切りリストまたは配列。このエントリを省略すると、すべてのプロパティが変更されたと見なされます。多数のレコードを変更する場合、プロパティ名を指定するとパフォーマンスが向上することがあります。 action="batch" のときに必須。変更を含む構造体の配列。1 回の通知で、複数の変更をバッチ処理して送信できます。5 つの update、1 つの delete、2 つの create などの、さまざまなタイプの変更を行えます。どのイベント構造体にも、アクションが含まれている必要があります。

例

次の例では、各イベントタイプの構造体を作成します。その後で、メッセージを含む構造体を作成します。メッセージ構造体には、Flex に送信するイベント構造体の配列が含まれます。destination は、"flex-services.xml" ファイルで指定されている、メッセージの送信先となる Data Management イベントゲートウェイの宛先 ID です。body はメッセージ本文です。CFML の sendGatewyMessage 関数はメッセージをゲートウェイに送信します。

```
<cfscript>
// Create event
createEvent = StructNew();
createEvent.action = "create";
createEvent.item = newContact;

// Create update notification
updateEvent = StructNew();
updateEvent.action="update";
updateEvent.previousversion = oldContact;
updateEvent.newversion = updatedContact;

// Create delete notification
identity = StructNew();
identity["contactId"] = newId;
deleteEvent = StructNew();
deleteEvent.action = "deleteID";
deleteEvent.identity = identity;

// Send a batch notification
all = StructNew();
all.destination="cfcontact";
all.action="batch";
all.changes = ArrayNew(1);
all.changes[1] = createEvent;
all.changes[2] = updateEvent;
all.changes[3] = deleteEvent;
r = sendGatewayMessage("LCDS", all);
</cfscript>
```

データ変換

次の表に、ColdFusion のデータ型と、それに対応する Adobe Flash または ActionScript のデータ型を示します。

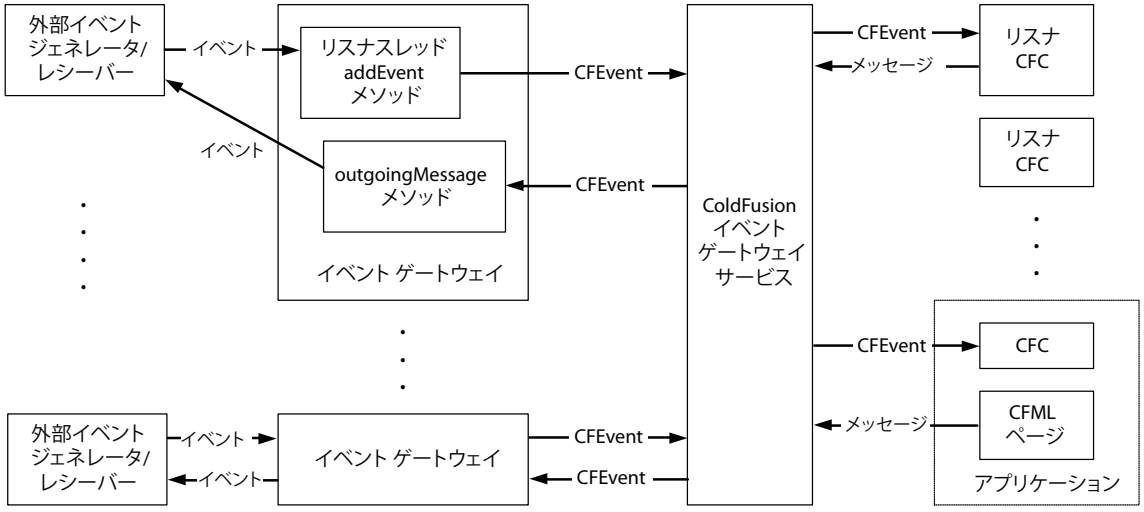
ColdFusion のデータ型	Flash のデータ型
文字列	String
配列	[] = 配列
構造体	{} = 型指定なしのオブジェクト
クエリー	ArrayCollection
CFC	クラス = 型指定ありのオブジェクト (対応する ActionScript クラスが存在する場合。それ以外の場合は、ActionScript の型指定なしの汎用オブジェクト (map) になります)
CFC の日付	ActionScript の日付
CFC の文字列	ActionScript の文字列
CFC の数値	ActionScript の数値
ColdFusion XML オブジェクト	ActionScript XML オブジェクト

カスタムイベントゲートウェイの作成

Adobe ColdFusion では、イベントゲートウェイを作成できます。ゲートウェイを作成するには、Java のイベント処理やスレッド処理などの Java プログラミングに関する詳細な知識を持っており、処理するメッセージのタイプなどのゲートウェイを提供するテクノロジーについて理解している必要があります。また、本マニュアルでは、CFC (ColdFusion コンポーネント) などの ColdFusion の開発に必要な概念や方法を既に習得されていることも前提としています。

イベントゲートウェイのアーキテクチャ

ColdFusion イベントゲートウェイはイベントをリスンし、アプリケーションのリスナー CFC で処理するために ColdFusion に渡します。イベントゲートウェイは、coldfusion.eventgateway.Gateway インターフェイスを実装し、ColdFusion の GatewayServices クラスを使用する必要があります。基本的なイベント処理アーキテクチャの図を拡張した次の図に、ColdFusion イベントゲートウェイの仕組みを示します。



メッセージの受信 : イベントゲートウェイのリスナースレッドが、ソケットや SMSC サーバーなどの外部イベントソースからイベントを受け取り、GatewayServices の addEvent メソッドを呼び出して、CFEvent のインスタンスを ColdFusion に送信します。

メッセージの送信 : ColdFusion イベントゲートウェイサービスが、イベントゲートウェイの outgoingMessage メソッドを呼び出して、宛先情報やメッセージ情報が含まれた CFEvent インスタンスを渡します。イベントゲートウェイは、必要に応じてこのメッセージを外部レシーバーに転送します。

イベントゲートウェイアーキテクチャは、SMS デバイスや IM クライアントなどの外部ソースからのメッセージを処理する場合だけでなく、ローカルシステムや ColdFusion アプリケーションにおける内部イベントを処理する場合にも使用できます。また、ゲートウェイで実装する通信は双方向でなくてもかまいません。

一方向の内部ゲートウェイの例としては、ColdFusion に用意されているサンプルディレクトリウォッチャゲートウェイがあります。このゲートウェイは単一のスレッドを持ち、ローカルディレクトリを定期的に確認して、ディレクトリのコンテンツが変更されたら CFC にメッセージを送信します。このゲートウェイは、発信メッセージをサポートしていません。このゲートウェイのコードは "gateway/src/examples/watcher" ディレクトリにあります。

ColdFusion 製品に同梱されている非同期 CFML ゲートウェイも、内部ゲートウェイです。通常のゲートウェイと異なり、このゲートウェイはリスナースレッドを持ちません。このゲートウェイの outgoingMessage メソッドは、CFML の SendGatewayMessage 関数からメッセージを取得し、メッセージを処理するために CFC の onIncomingMessage メソッドに送

信します。このゲートウェイを使用すれば、ColdFusion でリクエストフリーの非同期処理をサポートできます。このゲートウェイの使用方法の詳細については、1258 ページの「CFML イベントゲートウェイによる非同期 CFC へのアクセス」を参照してください。

イベントゲートウェイの要素

ゲートウェイを作成および設定するには、次の要素を使用します。

- 1309 ページの「ゲートウェイインターフェイス」
- 1310 ページの「GatewayServices クラス」
- 1310 ページの「CFEvent クラス」
- 1311 ページの「GatewayHelper クラス」
- 1312 ページの「ゲートウェイ設定ファイル」
- 1312 ページの「ゲートウェイ開発用のクラス」

注意：ゲートウェイのインターフェイスとクラス (GenericGateway クラスは除く) の詳細については、『CFML リファレンス』の Gateway development interfaces and classes を参照してください。ColdFusion の "gateways¥docs" ディレクトリにある JavaDoc でも、このリストに含まれるすべてのインターフェイスとクラス (GenericGateway クラスを含む) が説明されていますが、『CFML リファレンス』のほうが詳細に記述されています。JavaDoc では例が示されておらず、『CFML リファレンス』に記載されているような使用方法に関する詳細な説明もありません。

ゲートウェイインターフェイス

ColdFusion イベントゲートウェイは、coldfusion.eventservice.Gateway インターフェイスを実装する必要があります。次の表に、インターフェイスメソッドのシグネチャを示します。

注意：各メソッドの実装方法の詳細については、1313 ページの「イベントゲートウェイの作成」を参照してください。これらのメソッドのリファレンス情報については、『CFML リファレンス』の Gateway interface を参照してください。

シグネチャ	説明
void setGatewayID(String id)	ゲートウェイインスタンスを一意に識別するゲートウェイ ID を設定します。ColdFusion は、ゲートウェイクラスコンストラクタがゲートウェイ ID を設定している場合でも、イベントゲートウェイを起動する前にこのメソッドを呼び出します。
void setCFCListeners(String[] **listeners**)	イベントゲートウェイからの着信メッセージをリスンする CFC を識別します。この配列には、リスナー CFC へのパスが含まれます。ColdFusion は、イベントゲートウェイを起動する前、および実行中のイベントゲートウェイの設定が変更された場合に、このメソッドを呼び出します。
GatewayHelper getHelper()	coldfusion.eventgateway.GatewayHelper クラスインスタンスまたは null を返します。この GatewayHelper クラスは、イベントゲートウェイ固有のユーティリティメソッドを CFML アプリケーションに提供します。ColdFusion は、ColdFusion アプリケーションが GetGatewayHelper 関数を呼び出したときにこのメソッドを呼び出します。
String getGatewayID()	ゲートウェイ ID を返します。
int getStatus()	イベントゲートウェイのステータスを取得します。インターフェイスでは、ステータス定数として STARTING、RUNNING、STOPPING、STOPPED、FAILED が定義されています。
void start()	イベントゲートウェイを起動します。着信メッセージを処理するために最低 1 つのスレッドを開始します。ColdFusion は、イベントゲートウェイを起動するときにこのメソッドを呼び出します。

シグネチャ	説明
void stop()	イベントゲートウェイを停止します。スレッドを停止し、リソースを廃棄します。ColdFusion は、イベントゲートウェイを停止するときにこのメソッドを呼び出します。
void restart()	実行中のイベントゲートウェイを再起動します。ColdFusion は、実行中のイベントゲートウェイが ColdFusion Administrator によって再起動されたときにこのメソッドを呼び出します。
String outgoingMessage (coldfusion.eventgateway.CFEvent cfmessage)	ColdFusion によって送信されたメッセージを処理し、ゲートウェイタイプで必要な場合にこのメッセージを処理して送信します。リスナー CFC のリスナーメソッドが CFEvent を返すとき、または ColdFusion アプリケーションが SendGatewayMessage 関数を呼び出すときに、ColdFusion はこのメソッドを呼び出します。CFML の SendGatewayMessage 関数は、返された文字列をその戻り値として取得します。

GatewayServices クラス

Gateway クラスは coldfusion.eventgateway.GatewayServices クラスを使用して、ColdFusion イベントゲートウェイサービスを操作します。このクラスには、次のメソッドがあります。

シグネチャ	説明
GatewayServices getGatewayServices	GatewayServices オブジェクトを返すスタティックメソッド。ゲートウェイコードは、必要に応じていつでもこのメソッドを呼び出すことができます。
boolean addEvent**msg**)	リスナー CFC に送信するために、CFEvent インスタンスを ColdFusion に送信します。イベントゲートウェイは、このメソッドを使用して、すべての着信メッセージをアプリケーションに送信して処理させます。イベントがキューに追加されない場合は、False を返します。
int getQueueSize	ColdFusion イベントキューの現在のサイズを返します。このキューは、すべてのゲートウェイに対するすべてのメッセージを処理します。
int getMaxQueueSize	ColdFusion Administrator で設定されている、ColdFusion イベントキューの最大サイズを返します。
Logger getLogger Logger getLoggerlogfile)	イベントゲートウェイが "eventgateway.log" ログファイル (デフォルト) または指定のログファイルに情報を記録するために使用する ColdFusion Logger オブジェクトを返します。 **logfile** 属性では、ファイル名拡張子を付けずにファイル名を指定する必要があります。たとえば mylogifile のように指定します。ログファイルは ColdFusion ログディレクトリに保存され、指定したファイル名の後に .log が付加されます。 ロガーオブジェクトの使用方法の詳細については、1320 ページの「イベントの記録とログファイルの使用」を参照してください。

CFEvent クラス

Gateway クラスは、CFEvent インスタンスを送受信して、ColdFusion リスナー CFC またはアプリケーションとの間で通信を行います。ゲートウェイは、GatewayServices.addEvent メソッドで CFEvent インスタンスを送信することにより、ColdFusion にメッセージを通知します。同様にゲートウェイは、ColdFusion がゲートウェイの outgoingMessage メソッドを呼び出したときに CFEvent インスタンスを受信します。

CFEvent は java.util.Hashtable クラスを拡張したクラスで、次に示すコンストラクタと、フィールドの設定および取得メソッドがあります。CFML では、CFEvent インスタンスを構造体として扱います。

メソッド	説明
CFEventgatewayID)	CFEvent のコンストラクタ。**gatewayID** パラメータには、ゲートウェイコンストラクタに渡した値か、Gateway の setGatewayID メソッドで設定した値を指定する必要があります。
void setGatewayTypetype) String getGatewayType	イベントゲートウェイのタイプ (SMS など) を識別します。一貫性を保つため、ColdFusion Administrator の [ゲートウェイタイプ] ページでイベントゲートウェイタイプを追加するときは、[タイプ名] フィールドにこの名前を使用してください。
void setDatadata) Map getData	イベントデータ。ColdFusion との間でやり取りするメッセージを含んでいます。このフィールドの内容は、イベントゲートウェイタイプに応じて異なります。Map キーは文字列である必要があります。 ColdFusion では大文字と小文字が区別されないので、setData メソッドに渡した Map は大文字と小文字が区別されない Map に変換されます。したがって、大文字小文字のみが異なる名前を持つエントリをデータに作成しないでください。
void setOriginatorIDid) String getOriginatorID	着信メッセージの発信元または発信メッセージの宛先を識別します。この値は、プロトコルやイベントゲートウェイタイプによって異なります。
void setCFCPathpath) String getCFCPath	イベントを処理するアプリケーションリスナー CFC の絶対パス。デフォルトでは、ColdFusion Administrator の [イベントゲートウェイ] ページでイベントゲートウェイインスタンス用に設定されている最初のパスが使用されます。
void setCFCMethodmethod) String getCFCMethod	ColdFusion がこのイベントを処理するために呼び出すリスナー CFC のメソッド。デフォルトでは、onIncomingMessage メソッドが呼び出されます。一貫性を保つため、単一のリスナーを持つすべてのイベントゲートウェイではこのデフォルトの設定を変更しないことをお勧めします。ColdFusion XMPP ゲートウェイなどの、メッセージタイプに応じて異なるリスナーメソッドを使用するゲートウェイでは、このメソッドを使用して目的のメソッドを識別します。
void setCFCTimeoutseconds) String getCFCTimeout	リスナー CFC がイベントリクエストを処理するためのタイムアウト時間 (秒単位)。ColdFusion がイベントを処理するためにリスナー CFC を呼び出したが、指定のタイムアウト期間内に CFC がイベントを処理しなかった場合、ColdFusion はリクエストを終了し、"application.log" ファイルにエラーを記録します。デフォルトでは、ColdFusion Administrator の [サーバーの設定] ページで設定された [リクエストタイムアウト] の値が使用されます。
String getGatewayID	イベントを処理するイベントゲートウェイインスタンス。CFEvent のコンストラクタで設定されたゲートウェイ ID を返します。

GatewayHelper クラス

ColdFusion には、coldfusion.eventgateway.GatewayHelper という Java マーカーインターフェイスが含まれています。このインターフェイスを実装することで、ゲートウェイ固有のユーティリティメソッドを ColdFusion アプリケーションやリスナー CFC に提供するクラスを定義します。たとえば、インスタントメッセージングイベントゲートウェイは、ヘルパクラスを使用して、バディリスト管理メソッドをアプリケーションに提供することができます。

Gateway クラスは、ヘルパクラスまたは null (ゲートウェイがヘルパクラスを必要としない場合) を返す getHelper メソッドを実装する必要があります。

ColdFusion アプリケーションは、CFML の GetGatewayHelper 関数を呼び出します。この関数は、ゲートウェイの getHelper メソッドを呼び出して、ヘルパクラスのインスタンスを取得します。アプリケーションでは、ColdFusion オブジェクトのドット表記法を使用して、ヘルパクラスメソッドを呼び出せます。

次のコードは、SocketGateway クラスのゲートウェイヘルパである SocketHelper クラスを定義します。このコードには、空のコンストラクタと 2 つのパブリックメソッドがあります。1 つはソケット ID を返し、もう 1 つは指定されたソケットを閉じます。アプリケーションは、このクラスを使用して、セッション接続を監視および終了することができます。

```
public class SocketHelper implements GatewayHelper {
    public SocketHelper() {
    }
    public coldfusion.runtime.Array getSocketIDs () {
        coldfusion.runtime.Array a = new coldfusion.runtime.Array();
        Enumeration e = socketRegistry.elements();
        while (e.hasMoreElements()) {
            a.add(((SocketServerThread)e.nextElement()).getName());
         }
         return a;
     }
    public boolean killSocket (String socketid) {
        try
        {
            ((SocketServerThread)socketRegistry.get(socketid)).socket.close();
            ((SocketServerThread)socketRegistry.get(socketid)).socket = null;
            socketRegistry.remove(socketid);
            return true;
        }
        catch (IOException e) {
            return false;
        }
    }
}
```

ゲートウェイ設定ファイル

ゲートウェイでは、頻繁に変更されることのない情報を、設定ファイルを使用して指定できます。たとえば ColdFusion SMS イベントゲートウェイの設定ファイルには、IP アドレス、ポート番号、システム ID、パスワードなどの値が含まれています。

ColdFusion Administrator で、各イベントゲートウェイインスタンスの設定ファイルのパスを指定することができます。ColdFusion は、イベントゲートウェイをインスタンス化するときに、設定ファイルのパスをゲートウェイコンストラクタに渡します。設定ファイルの形式や内容は、ゲートウェイによって異なります。ファイルの内容を解析して、適切に使用することは、ゲートウェイクラスの役割です。

設定データにアクセスして取得する方法の例としては、java.util.Properties クラスを使用する方法があります。このクラスは ISO8859-1 の形式の入力ストリームを取り、1 行につき 1 つのプロパティが設定されます。次の例に示すとおり、等号 (=) またはコロン (:) を使用してプロパティ名と値を区切る必要があります。

```
ip-address=127.0.0.1
port=4445
```

たとえば、SocketGateway イベントゲートウェイではこの方法を使用して、設定ファイルからオプションのポート番号を取得しています。プロパティファイルを読み取って、そのデータを使用する例については、1313 ページの「クラスコンストラクタ」のコードを参照してください。

ゲートウェイ開発用のクラス

ColdFusion には、イベントゲートウェイクラスを開発するためのビルディングブロックとして使用可能な 2 つのクラスが用意されています。このクラスはそれぞれ、異なる開発手法に対応しています。

- coldfusion.eventgateway.GenericGateway クラスは、作成するゲートウェイクラスのスーパークラスとして使用できる抽象クラスです。
- "gateway¥src¥examples" ディレクトリにある EmptyGateway クラスは、ゲートウェイクラスの作成に使用できるテンプレートゲートウェイです。

GenericGateway クラス

ColdFusion には、ColdFusion Gateway インターフェイスの多くのメソッドを実装し、ゲートウェイクラスで使用できる便利なメソッドを提供する coldfusion.eventgateway.GenericGateway 抽象クラスが含まれています。

このクラスからゲートウェイクラスを派生させることができます。これにより、getGatewayID や SetCFCListeners メソッドなどの、ゲートウェイの実装に関する基本的な構造が提供されます。派生させたクラスは、最低でも次のメソッドを実装する必要があります。

- startGateway (start ではない)
- stopGateway (stop ではない)
- outgoingMessage

派生させたゲートウェイクラスでは、次のものも実装する必要があります。

- 設定ファイルをサポートする場合は、設定ファイルを取るコンストラクタ、および設定をロードするルーチン
- gatewayHelper クラスを使用する場合は、getHelper メソッド
- イベントソースのステータスがゲートウェイとは非同期で変更される場合は、getStatus メソッド

たとえば、JMS ゲートウェイは、汎用ゲートウェイクラスから派生しています。このクラスに関する説明は、"gateway¥docs" ディレクトリにあるゲートウェイクラスの JavaDoc で提供されています。『CFML リファレンス』ではこのクラスについて説明していません。

EmptyGateway クラス

"gateway¥src¥examples¥EmptyGateway.java" ファイルには、独自のイベントゲートウェイを作成するときに使用可能なイベントゲートウェイテンプレートが含まれています。このゲートウェイディレクトリは、サーバー設定の場合は <ColdFusion のルートディレクトリ > に、J2EE 設定の場合は "<ColdFusion のルートディレクトリ >¥WEB-INF-cfusion" ディレクトリにあります。このファイルには、coldfusion.eventgateway.Gateway インターフェイスのすべてのメソッドの最低限の実装が含まれています。このファイルは、リスナースレッドの基本構造を定義し、一般的に使用される Gateway のプロパティを初期化します。EmptyGateway のソースコードには、イベントゲートウェイクラスを作成するために必要な追加情報を説明するコメントが含まれています。

イベントゲートウェイの作成

Gateway クラスを作成するには、テンプレートとして "EmptyGateway.java" ファイルを使用することができます。このファイルは、サーバー設定の場合は "<ColdFusion のルートディレクトリ >/gateway/src/examples/" ディレクトリに、J2EE 設定の場合は "<ColdFusion のルートディレクトリ >/WEB-INF/cfusion/gateway/src/examples/" ディレクトリにあります。このファイルに定義されているイベントゲートウェイは機能しませんが、すべての Gateway クラスメソッドの基本構造となるコードが含まれています。

本マニュアルでは、可能な限り、サンプル Socket イベントゲートウェイをベースとするコードを使用して、イベントゲートウェイの機能を実装する方法を示しています。このファイルは、サーバー設定の場合は "<ColdFusion のルートディレクトリ >/gateway/src/examples/socket/SocketGateway.java"、J2EE 設定の場合は "<ColdFusion のルートディレクトリ >/WEB-INF/cfusion/gateway/src/examples/socket/SocketGateway.java" にあります。

クラスコンストラクタ

イベントゲートウェイでは、次に示す任意のコンストラクタを実装できます。

- MyGateway(String gatewayID, String configurationFile)
- MyGateway(String gatewayID)
- MyGateway()

ColdFusion を起動すると、ColdFusion で設定した各イベントゲートウェイインスタンスのコンストラクタが呼び出されます。また、イベントゲートウェイがインスタンス化されると、ゲートウェイの Start メソッドが呼び出されます。ColdFusion では、まず 2 つのパラメータを取るコンストラクタを使用しようとします。

それぞれのイベントゲートウェイインスタンスは一意の ID を持つ必要があるので、ColdFusion では ID の提供に関して冗長性のあるサポートが提供されています。イベントゲートウェイでデフォルトのコンストラクタのみが実装されている場合、ColdFusion はイベントゲートウェイの setGatewayID メソッドを呼び出して ID を提供します。

イベントゲートウェイで 2 つのパラメータを取るコンストラクタが実装されていない場合、ColdFusion から設定ファイルの情報が取得されません。

通常、コンストラクタはスタティックな GatewayServices.getGatewayServices メソッドを呼び出して、ColdFusion イベントゲートウェイサービスにアクセスします。この処理を呼び出す必要はありませんが、実行するのがよいコーディングスタイルです。

ゲートウェイ ID のみを取る最低限のコンストラクタは、次のようになります。

```
public MyGateway(String gatewayID) {
    this.gatewayID = gatewayID;
    this.gatewayService = GatewayServices.getGatewayServices();
}
```

このゲートウェイコンストラクタで、処理できないエラーが発生した場合は、coldfusion.server.ServiceRuntimeException 例外を投げる必要があります。たとえば、設定ファイルを必要とするイベントゲートウェイで、設定ファイルの内容が読み取れない場合は、この例外を投げる必要があります。

ゲートウェイで設定ファイルを使用する場合は、Start メソッドで設定ファイルをロードしている場合でも、コンストラクタで設定ファイルをロードする必要があります。コンストラクタは独立したスレッドでは実行されないので、このファイルをロードする必要があります。ファイルをロードできない場合は、ColdFusion Administrator にエラーを表示することができます。独立したスレッドで実行される Start メソッドでファイルをロードできない場合、エラーはログに記録されますが、Administrator に即座のフィードバックを提供することはできません。

サンプルの Socket イベントゲートウェイには、2 つのパラメータを取る単一のコンストラクタがあります。このコンストラクタは、設定ファイルをロードします。ColdFusion Administrator で設定ファイルを指定していない場合、またはファイルのパスが無効である場合、コンストラクタは IO 例外を取得します。次に、デフォルトのポートを使用して、処理内容を示すメッセージを記録します。次の例は、Gateway コンストラクタコード、およびこのコードが使用する loadProperties メソッドです。

```
public SocketGateway(String id, String configpath)
{
    gatewayID = id;
    gatewayService = GatewayServices.getGatewayServices();
    // log things to socket-gateway.log in the CF log directory
    log = gatewayService.getLogger("socket-gateway");
    propsFilePath=configpath;
    try
    {
        FileInputStream propsFile = new FileInputStream(propsFilePath);
        properties.load(propsFile);
        propsFile.close();
        this.loadProperties();
     }
     catch (IOException e)
    {
        // Use default value for port and log the status.
        log.warn("SocketGateway(" + gatewayID + ") Unable to read configuration
            file '" + propsFilePath + "': " + e.toString() + ".Using default port
            " + port + ".", e);
    }
}

private void loadProperties() {
    String tmp = properties.getProperty("port");
    port = Integer.parseInt(tmp);
}
```

Gateway クラスのサービスおよび情報ルーチンの提供

ゲートウェイメソッドの中には、イベントゲートウェイの設定やイベントゲートウェイ情報の取得を行うメソッドがあります。ColdFusion イベントゲートウェイサービスは、こうしたメソッドの多くを呼び出して、ColdFusion Administrator によって保存された情報に基づいてイベントゲートウェイを設定したり、イベントゲートウェイサービスやアプリケーションで必要なリソースや情報にアクセスしたりします。これらのメソッドのいくつかは、イベントゲートウェイのコードでも役立ちます。これらのサービスや情報は、次のメソッドによって提供されます。

- setCFCListeners
- setGatewayID
- getHelper
- getGatewayID
- getStatus

ゲートウェイを起動すると、ColdFusion Administrator で指定した CFC に対して setCFCListeners メソッドが呼び出されます。また、設定情報が変更されたときにも、実行中のイベントゲートウェイのこのメソッドが呼び出されます。したがって、このメソッドは、こうした変更を処理するように記述する必要があります。setCFCListeners メソッドは、ゲートウェイサービスに着信メッセージを送信するゲートウェイコードが setCFCPath メソッドでリスナー CFC を指定できるように、リスナー情報を保存する必要があります。

ゲートウェイが起動すると、setGatewayID メソッドが呼び出されます。getGatewayID メソッドは、このメソッドで設定した値を返す必要があります。

アプリケーションが CFML の GetGatewayHelper 関数を呼び出すと、getHelper メソッドが呼び出されます。

次のコードは、SocketGateway クラスにおけるこれらのメソッドの定義を示します。ゲートウェイを作成するには、getHelper の定義を変更して、適切なクラスを返すか、ゲートウェイヘルパクラスが存在しない場合は null を返すようにする必要があります。ほとんどのゲートウェイでは、その他のメソッドの定義は変更しなくても問題ありません。

```
public void setCFCListeners(String[] listeners) {
    this.listeners = listeners;
}
public GatewayHelper getHelper() {
    // SocketHelper class implements the GatewayHelper interface.
    return new SocketHelper();
}
public void setGatewayID(String id) {
    gatewayID = id;
}
public String getGatewayID() {
    return gatewayID;
}
public int getStatus() {
    return status;
}
```

イベントゲートウェイの起動、停止、再起動

イベントゲートウェイは最低でも 1 つのリスナースレッドを使用するので、スレッドを制御するための start、stop、および restart メソッドが必要になります。また、これらのメソッドでは、Gateway クラスの getStatus メソッドによって確認されるステータス変数を管理し、必要に応じてその値を STARTING、RUNNING、STOPPING、STOPPED、および FAILED のいずれかに変更する必要があります。

start メソッド

start メソッドは、イベントゲートウェイを初期化します。このメソッドは、イベントソースの監視や、ソースから受け取ったメッセージへの応答を行うリスナースレッドを起動します。

start メソッドは、ColdFusion Administrator でイベントゲートウェイタイプに設定したタイムアウト時間内に処理を返す必要があります。ColdFusion Administrator では、各ゲートウェイタイプに対して [スタートアップタイムアウト時に停止] オプションが設定できます。このオプションが有効である場合にタイムアウトが発生すると、ColdFusion スタータースレッドは、ゲートウェイスレッドを終了するためにこのスレッドに対する中断を呼び出してから、終了します。

注意：start メソッドがリスナー (たとえば、単一スレッドのゲートウェイ) である場合、このメソッドはゲートウェイが停止するまで処理を返しません。そのようなゲートウェイに対しては、ColdFusion Administrator で [スタートアップタイムアウト時に停止] オプションを設定しないでください。

ゲートウェイが設定ファイルを使用する場合、設定ファイルから設定をロードします。これによって、ColdFusion を再起動することなく、設定ファイルを変更してゲートウェイを再起動することができます。また、コンストラクタで設定ファイルをロードします。詳細については、1313 ページの「クラスコンストラクタ」を参照してください。

SocketGateway クラスでは、start メソッドは初期スレッドを起動します。単一スレッドのゲートウェイでは、このスレッドが唯一のスレッドになります。スレッドが起動すると、socketServer メソッドが呼び出されます。このメソッドは、Java ServerSocket クラスを使用して、マルチスレッドソケットリスナーとメッセージディスパッチャを実装します。リスナーの詳細については、1318 ページの「着信メッセージへの応答」を参照してください。

```
public void start()
{
    status = STARTING;
    listening=true;
    // Start up event generator thread
    Runnable r = new Runnable()
    {
        public void run()
        {
            socketServer();
        }
    };
    Thread t = new Thread(r);
    t.start();
    status = RUNNING;
}
```

stop メソッド

stop メソッドは、リスナースレッドのシャットダウンやリソースの開放など、イベントゲートウェイのシャットダウンタスクを実行します。次に、SocketGateway の stop メソッドを示します。

```
public void stop()
{
    // Set the status variable to indicate that the server is stopping.
    status = STOPPING;
    // The listening variable is used as a switch to stop listener activity.
    listening=false;
    // Close the listener thread sockets.
    Enumeration e = socketRegistry.elements();
    while (e.hasMoreElements()) {
        try
        {
            ((SocketServerThread)e.nextElement()).socket.close();
        }
        catch (IOException e1)
        {
            // We don't care if a close failed.
            //log.error(e1);
        }
    }
    // Close and release the serverSocket instance that gets requests from the
    // network.
    if (serverSocket != null) {
        try
        {
            serverSocket.close();
        }
        catch (IOException e1)
        {
        }
        //Release the serverSocket.
        serverSocket = null;
    }
    // Shutdown succeeded; set the status variable.
    status = STOPPED;
}
```

restart メソッド

ほとんどの場合、restart メソッドの実装では、stop メソッドを呼び出した後に start メソッドを呼び出しますが、サービスによってはこのプロセスを最適化できる場合があります。次のコードは、SocketGateway クラスの restart メソッドを示しています。

```
public void restart() {
    stop();
    start();
}
```

着信メッセージへの応答

着信メッセージ (イベント) にはリスナースレッドが応答します。このスレッドには、次に示すように、ColdFusion イベントゲートウェイサービスにメッセージを送信するコードが含まれている必要があります。

1. CFEvent インスタンスを作成します。
2. メッセージやその他のイベントゲートウェイ固有の情報を含む Map インスタンスを作成し、CFEvent の setData メソッドに渡します。
3. CFEvent の setOriginator メソッドを呼び出して、メッセージのソースを指定します。この呼び出しは、ColdFusion アプリケーションが応答を送信する場合に必要です。
4. CFEvent の setGateWayType メソッドを呼び出して、イベントゲートウェイタイプを指定します。
5. デフォルトの動作が適切でない場合には、他の CFEvent フィールドを設定します。たとえば、setCFCPath メソッドを呼び出して、デフォルトのリスナー CFC を置き換えます。デフォルトの CFEvent フィールドの詳細については、1310 ページの「CFEvent クラス」を参照してください。
6. gatewayService.addEvent メソッドを呼び出して、CFEvent インスタンスを ColdFusion に送信します。
7. イベントがイベントゲートウェイサービスキューに追加されない (addEvent メソッドが False を返す) 場合の処理を行います。

複数のリスナー CFC にメッセージを送信する場合は、CFEvent インスタンスを作成および設定し、gatewayService.addEvent メソッドを呼び出して、それぞれのリスナー CFC にメッセージを送信する必要があります。ゲートウェイの setCFCListeners メソッドで、CFEvent インスタンスを設定するための CFC パスをゲートウェイに提供する必要があります。

ColdFusion サーバーが大量のイベントゲートウェイメッセージを送信する場合、ColdFusion イベントゲートウェイサービスのイベントキューが、ColdFusion Administrator で設定された最大値に達することがあります。このキューが最大値に達すると、gatewayService.addEvent メソッドは False を返して失敗します。コードは、次のいずれかの処理を行えます。

- メッセージが受信されなかったことを示すメッセージを送信者に返します。
- GatewayService の getQueueSize メソッドおよび getMaxQueueSize メソッドによって返される値を定期的に比較して、キューが利用可能になるまで待機し、キューのサイズが最大値を下回ったら addEvent メソッドを再試行します。
- GatewayService の getLogger メソッドによって返されるロガーを使用して、状況を記録します。詳細については、1320 ページの「イベントの記録とログファイルの使用」を参照してください。

SocketGateway クラスでは、java.net.ServerSocket クラスオブジェクトおよび SocketServerThread リスナースレッドを使用してリスナーを実装しています。SocketServerThread のコードについては、SocketGateway のソースを参照してください。リスナースレッドは、TCP/IP ソケットからメッセージを取得すると、次の processInput メソッドを呼び出して、メッセージを ColdFusion に送信します。このメソッドは、CFEvent のすべての必須およびオプションのフィールドを明示的に設定し、イベントを ColdFusion に送信します。addEvent の呼び出しが失敗した場合、エラーが記録されます。

注意：processInput メソッドのコードの大半は、複数のリスナー CFC をサポートするためのものです。単一のリスナー CFC のみを使用するゲートウェイの場合、必要になるのはこのメソッドの後半のコードのみです。

```
private void processInput(String theInput, String theKey)
{
    // Convert listeners list to a local array
    // Protect ourselves if the list changes while we are running
    String[] listeners;
    int size = cfcListeners.size();
    if (size > 0)
    {
        // Capture the listeners list
        synchronized (cfcListeners)
        {
            listeners = new String[size];
            cfcListeners.toArray(listeners);
        }
    }
    else
    {
        // Create a dummy list
        listeners = new String[1];
        listeners[0] = null;
    }
    // Broadcast to all the CFC listeners
    // Send one message at a time with different CFC address on them
    for (int i = 0; i < listeners.length; i++)
    {
        String path = listeners[i];
        CFEvent event = new CFEvent(gatewayID);
        Hashtable mydata = new Hashtable();
        mydata.put("MESSAGE", theInput);
        event.setData(mydata);
        event.setGatewayType("SocketGateway");
        event.setOriginatorID(theKey);
        event.setCfcMethod(cfcEntryPoint);
        event.setCfcTimeOut(10);
        if (path != null)
            event.setCfcPath(path);
            boolean sent = gatewayService.addEvent(event);
        if (!sent)
            log.error("SocketGateway(" + gatewayID + ") Unable to place message on
                event queue. Message not sent from " + gatewayID + ", thread " + theKey
                + ".Message was " + theInput);
    }
}
```

ColdFusion 関数またはリスナー CFC への応答

イベントゲートウェイアプリケーションのリスナー CFC のリスナーメソッドがメッセージを返したり、CFML コードが SendGatewayMessage 関数を呼び出してメッセージを生成すると、ColdFusion イベントゲートウェイサービスは、メッセージを処理するためにイベントゲートウェイの outgoingMessage メソッドを呼び出します。このメソッドは、適切な外部リソースにメッセージを送信する必要があります。

outgoingMessage メソッドのパラメータは、送信するメッセージに関する情報を含んだ CFEvent インスタンスです。CFEvent の getData メソッドは、メッセージに関するイベントゲートウェイ固有の情報 (メッセージテキストなど) を含んだ Map オブジェクトを返します。outgoingMessage が受け取るすべての CFEvent インスタンスの Data フィールドおよび GatewayID フィールドには、情報が含まれています。

リスナー CFC の onIncomingMessage メソッドから返される CFEvent インスタンスには、着信メッセージの発信元 ID とその他の情報が含まれています。しかし、ColdFusion の SendGatewayMessage 関数からのメッセージを処理するゲートウェイでは、この情報が常に提供されるとは限りません。したがって、すべての発信メッセージでデータ Map の宛先 ID を必須にすることをお勧めします。

outgoingMessage メソッドは文字列の値を返します。CFML の sendGatewayMessage 関数は、この値を ColdFusion アプリケーションに返します。返される文字列は、メッセージのステータスを表している必要があります。エラーが発生せず、返す必要のある追加情報 (メッセージ ID など) も存在しない場合、ColdFusion イベントゲートウェイの outgoingMessage メソッドは慣例として "OK" を返します。

イベントメッセージは非同期であるため、戻り値が肯定的なものであっても、一般に、それはメッセージが正常に配信されることを示すのではなく、単に outgoingMessage メソッドでメッセージが正常に処理されたことを示すに過ぎません。ただし、場合によっては、outgoingMessage メソッドを少なくとも部分的に同期させることが可能です。たとえば、SMS ゲートウェイには、次に示す 2 つの outgoingMessage モードが用意されています。

非同期モード outgoingMessage メソッドは、メッセージプロバイダの SMSC (Short Message Service Center) に送信するためにメッセージが内部的にキューに追加されると処理を返します。

同期モード メソッドは、メッセージが SMSC に送信されるか、エラーが発生するまで処理を返しません。

このようにして SMS アプリケーションは、後でメッセージの配信受信の比較などに使用するメッセージ ID を取得することができます。

outgoingMessage メソッドの例

次の outgoingMessage メソッドは、SocketGateway クラスのメソッドに似ています。行う処理は次のとおりです。

1. CFEvent クラスの getData メソッドによって返されるデータ Map の MESSAGE フィールドの内容を取得します。
2. データ Map の outDestID フィールドから宛先を取得します。
3. 宛先のソケットサーバースレッドを使用して、メッセージを書き込みます。

```
public String outgoingMessage(coldfusion.eventgateway.CFEvent cfmsg) {
    String retcode="ok";
    // Get the table of data returned from the event handler
    Map data = cfmsg.getData();
    String message = (String) data.get("MESSAGE");
    // Find the right socket to write to from the socketRegistry hash table
    // and call the writeoutput method of the socket thread.
    // (Get the destination ID from the data map.)
    if (data.get("outDestID") != null)
        ((SocketServerThread)socketRegistry.get(data.get("outDestID"))).
                writeOutput(message);
    else {
        System.out.println("cannot send outgoing message. OriginatorID is not
            available.");
        retcode="failed";
        }
    return retcode;
}
```

イベントの記録とログファイルの使用

GatewayServices.getLogger メソッドは、coldfusion.eventgateway.Logger クラスのインスタンスを返します。これを使用して、ColdFusion ログディレクトリ内のファイルにメッセージを記録できます。このディレクトリは、ColdFusion Administrator の [ロギングの設定] ページで設定します。このメソッドは、パラメータを取らないか、1 つのパラメータを取ります。

- デフォルトの GatewayServices.getLogger メソッドは "eventgateway.log" ファイルを使用します。
- オプションで、ログファイルの名前を指定できます。このとき、.log 拡張子やディレクトリパスは指定しません。

次の例では、ColdFusion ログディレクトリの "mygateway.log" ファイルにゲートウェイからのメッセージを記録します。

```
coldfusion.eventgateway.Logger log =getGatewayServices().getLogger("mygateway");
```

Logger クラスには次のメソッドがあります。これらはすべて、メッセージ文字列を取ります。使用するメソッドによって、ログメッセージに設定される厳格度が決まります。

- info
- warn
- error
- fatal

これらのメソッドの 2 番目のパラメータとして、例外インスタンスを渡すこともできます。例外を渡すと、ColdFusion ログディレクトリ内の "exception.log" ファイルに例外情報が保存されます。

これらのメソッドを使用することで、適切なメッセージを記録できます。ただし、デフォルトの "eventgateway.log" ファイルを使用する場合は、このファイルがすべての ColdFusion 標準ゲートウェイによって使用され、他のゲートウェイでも使用される可能性がある点に注意してください。したがって、このファイルに定常的に記録するメッセージは、あまり頻繁に発生しない通常の現象 (ゲートウェイの起動やシャットダウンなど) や、データを保持する必要があるエラーに限定します。

メッセージテキストには次の形式が使用されます。アプリケーションではこのパターンに対応する必要があります。

```
GatewayType (Gateway ID) Message
```

たとえば SMS イベントゲートウェイには、次に示す例外検出コードが含まれています。このコードは、一般的な例外メッセージと例外名を "eventgateway.log" ファイルに記録し、"exceptions.log" ファイルにも自動的に例外を記録します。

```
catch(Exception e)
{
    logger.error("SMSGateway (" + gatewayID + ") Exception while processing
    incoming event: " + e, e);
}
```

注意：イベントゲートウェイアプリケーションを開発するときは、ColdFusion ログビューアを使用して、アプリケーションのログファイルや標準の ColdFusion ログファイル ("eventgateway.log" および "exception.log" ファイルを含む) を確認できます。ビューアでは、厳格度を選択したりキーワードを検索したりして、表示をフィルタ処理できます。

イベントゲートウェイのデプロイ

イベントゲートウェイをデプロイするには、イベントゲートウェイタイプをデプロイして、イベントゲートウェイインスタンスを設定します。

ColdFusion でのイベントゲートウェイタイプのデプロイ

1. Gateway クラスをコンパイルして、他の必要なクラスとともに JAR ファイル内に保存します。

   注意：ColdFusion クラスローダーは、ゲートウェイの **"¥lib"** ディレクトリをクラスパスに含め、そのディレクトリ内にあるすべての **JAR** ファイルをクラスパスに含めます。

2. JAR ファイルを、J2EE 設定の場合は "<ColdFusion のルートディレクトリ >¥WEB-INF¥cfusion¥gateway¥lib" ディレクトリに、サーバー設定の場合は "<ColdFusion のルートディレクトリ >¥gateway¥lib" ディレクトリに保存します。このディレクトリは ColdFusion クラスパス上に存在します。

3. ColdFusion イベントゲートウェイサービスが、ColdFusion Administrator の [イベントゲートウェイ]-[設定] ページで有効になっていることを確認します。

4. ColdFusion Administrator の [データとサービス]-[ゲートウェイタイプ] で、[ゲートウェイタイプの管理] ボタンをクリックして [ゲートウェイタイプ] ページを表示します。

5. [ColdFusion ゲートウェイタイプの追加 / 編集] フォームに、タイプ名 (たとえば、SMS イベントゲートウェイの場合は SMS)、説明、および完全 Java クラス名 (たとえば、SMS イベントゲートウェイの場合は

coldfusion.eventgateway.sms.SMSGateway) を入力します。必要に応じて、[スタートアップタイムアウト] 設定をデフォルトの値から変更します。

6 [タイプの追加] ボタンをクリックして、ColdFusion にイベントゲートウェイタイプをデプロイします。

次の手順で、ゲートウェイタイプを使用するイベントゲートウェイインスタンスの設定方法を説明します。

イベントゲートウェイインスタンスの設定

1 イベントゲートウェイタイプのデプロイを完了した場合は、[ゲートウェイインスタンスの管理] ボタンをクリックします。そうでない場合は、ColdFusion Administrator で [イベントゲートウェイ]-[ゲートウェイインスタンス] を選択します。

2 [ColdFusion ゲートウェイインスタンスの追加 / 編集] フォームで、次の操作を行います。

- [ゲートウェイ ID] フィールドにインスタンス名を入力します。
- 追加したイベントゲートウェイタイプを [ゲートウェイタイプ] メニューから選択します。
- メッセージを処理するリスナー CFC へのパスを指定します。
- イベントゲートウェイに設定ファイルが必要な場合は、[設定ファイル] フィールドにファイルへのパスを入力します。
- ColdFusion の起動時にゲートウェイを自動的に起動させない場合は、[スタートアップモード] を [手動] または [無効] に変更します。

3 [ゲートウェイインスタンスの追加] ボタンをクリックします。

4 設定済みインスタンスのリストで、起動するゲートウェイインスタンスのエントリの起動ボタン (緑色の三角形) をクリックします。

Eclipse 用の ColdFusion Extension の使用

Eclipse 用の Adobe ColdFusion Extension には、頻繁に行われるコーディング作業を効率化するウィザード群や、Adobe Flash Builder および Eclipse からリモートサーバーに接続するための拡張機能が用意されています。

Eclipse 用の ColdFusion Extension を使用するには、ColdFusion コンポーネントに関する知識や、ColdFusion アプリケーションでのデータのアクセスおよび使用に関する知識が必要です。また、Eclipse や Adobe Flash Builder に習熟していることも前提となります。

Eclipse 用の ColdFusion Extension について

Eclipse 用の ColdFusion Extension には、頻繁に行われるコーディング作業を効率化する次の機能が用意されています。

- Eclipse RDS Support プラグイン。ColdFusion サーバー上のファイルやデータソースにアクセスできるようになります。
- ColdFusion/Flex アプリケーションウィザード。アプリケーション内にマスタページと詳細ページを作成し、これらのページでデータベース内のレコードを作成、読み取り、更新および削除 (CRUD) できます。
- ColdFusion/Ajax アプリケーションウィザード。アプリケーション内に Ajax 要素を使用するマスタページと詳細ページを作成し、これらのページでデータベース内のレコードを作成、読み取り、更新および削除 (CRUD) できます。
- RDS CRUD ウィザード。ColdFusion サーバーの ColdFusion Administrator で登録されているテーブルに基づいて、ColdFusion コンポーネント (CFC) をダイナミックに作成できます。
- ActionScript to CFC ウィザード。ActionScript クラスファイルに基づいて CFC を作成できます。

- CFC to ActionScript ウィザード。CFC のバリューオブジェクトに基づいて ActionScript ファイルを作成できます。
- サービスブラウザ。CFC の表示、Web サービスのリストの管理、Web サービスを呼び出す CFML コードの生成を行うことができます。

Eclipse 用の ColdFusion 拡張機能のインストールの詳細については、『ColdFusion インストール』ガイドを参照してください。

Eclipse RDS Support

RDS (Remote Development Services) を使用すると、ColdFusion サーバーの ColdFusion Administrator で登録されているファイルやデータソースにアクセスできます。Eclipse RDS Support を使用するには、ColdFusion のインストール時に RDS を有効にしておく必要があります。Eclipse RDS Support により、Flash Builder または CFEclipse を IDE として使用して、ColdFusion ファイルにリモートからアクセスできるようになります。

Eclipse RDS Support は、すべての ColdFusion サーバープラットフォームでサポートされています。

Eclipse RDS Support をインストールするには、次の製品がインストールされている必要があります。

- Eclipse 3.1 以降、Flex Builder 2 以降、または Flash Builder
- ColdFusion MX 7.0.1 以降

RDS の設定

RDS を使用する前に、ColdFusion サーバーを設定する必要があります。

RDS を使用して接続する ColdFusion サーバーの設定

1 Flash Builder または Eclipse で、[ウィンドウ]-[設定]-[ColdFusion]-[RDS 構成] を選択します。

2 デフォルトの localhost サーバーを設定するには、localhost を選択して次の情報を指定します。
   - 説明
   - ホスト名 (127.0.0.1)
   - ポート番号 (ビルトイン Web サーバーを使用している場合は 8500)
   - コンテキストルート (必要に応じて。コンテキストルートの詳細については、『ColdFusion インストール』ガイドを参照してください)
   - パスワード (RDS パスワード)

3 他にもサーバーを指定する場合は、[New] をクリックして次の情報を指定します。
   - 説明 (任意の名前)
   - ホスト名 (IP アドレスまたはコンピュータ名)
   - ポート番号 (ビルトイン Web サーバーを使用している場合は 8500)
   - 必要な場合は、コンテキストルート

     コンテキストルートの詳細については、『ColdFusion インストール』ガイドを参照してください。
   - パスワード (RDS パスワード)

4 定義したサーバーを削除するには、目的のサーバーを選択して [Remove] をクリックします。

5 接続をテストするには、目的のサーバーを選択して [Test Connection] をクリックします。

注意：ColdFusion MX 7 またはそれ以前のリリースを使用している場合は、"RDS サーバーに正しく通信できましたが、セキュリティ証明情報が正しくありませんでした " というメッセージが表示されます。パスワードが検証されなかったという意味ですが、このメッセージはパスワードが正しい場合でも表示されます。[OK] をクリックしてメッセージボックスを閉じてかまいません。

CF サーバーへの RDS 接続の設定が終了すると、RDS サーバー上のファイル、フォルダおよびデータソースを表示できるようになります。各 RDS サーバーは、RDS ファイルビューや RDS データビューにノードとして表示され、RDS サーバーの設定時に指定した名前が表示されます。

ファイル、フォルダまたはデータソースの表示

1. Flash Builder では、[ウィンドウ]-[その他のビュー] を選択します。Eclipse では、[ウィンドウ]-[ビューの表示]-[その他] を選択します。
2. [RDS] を選択します。
3. RDS サーバー上のファイルシステムにアクセスするには、[RDS ファイルビュー] を選択します。
4. RDS サーバー上のデータソースにアクセスするには、[RDS データビュー] を選択します。

RDS ファイルビューの使用

RDS ファイルビューには、RDS サーバー上のフォルダとファイルがすべて表示されます。次の表に示すボタンを使用して、各種の操作を行えます。

ボタン	アクション
	アクティブな RDS サーバーを更新して最新の状態を表示します。
	選択されているフォルダ内にファイルを作成します。
	選択されているファイルを削除します。
	選択されているフォルダ内にフォルダを新規作成します。
	選択されているフォルダを削除します。

注意：RDS Eclipse Support では、ファイルのコピー & ペースト、ドラッグ & ドロップ、ファイル属性の変更などのファイル操作は実行できません。ただし、ファイルの削除、保存、別名保存、名前変更は行えます。また、ColdFusion 5 以降の ColdFusion サーバーでは、最終更新日フィールドが表示されません。

フォルダまたはファイルの名前を変更するには、そのフォルダ名またはファイル名を右クリックします。

RDS データビューの使用

RDS データビューには、RDS サーバー上のデータソースがすべて表示されます。次の表に示すボタンを使用して、各種の操作を行えます。

ボタン	名前	説明
	更新	選択されているアイテムを更新して最新の状態を表示します。
	RDS クエリービューア	RDS クエリービューアを開きます。

RDS クエリービューアまたは Visual Query Builder を使用してクエリを作成できます。RDS Visual Query Builder は、ColdFusion Report Builder のクエリビルダや、HomeSite のクエリビルダに似ています。

RDS クエリービューアを使用したクエリの作成と実行

1 [RDS データビュー] タブの [RDS クエリービューア] アイコンをクリックします。

RDS クエリービューアが独自のタブで開きます。他のドキュメントが開いている場合は、RDS クエリービューアにフォーカスが移動します。

2 次のいずれかの操作を行います。

- SQL を入力し、適切なフィールド名とテーブル名をダブルクリックします。
- [Visual Query Builder] ボタンをクリックします。

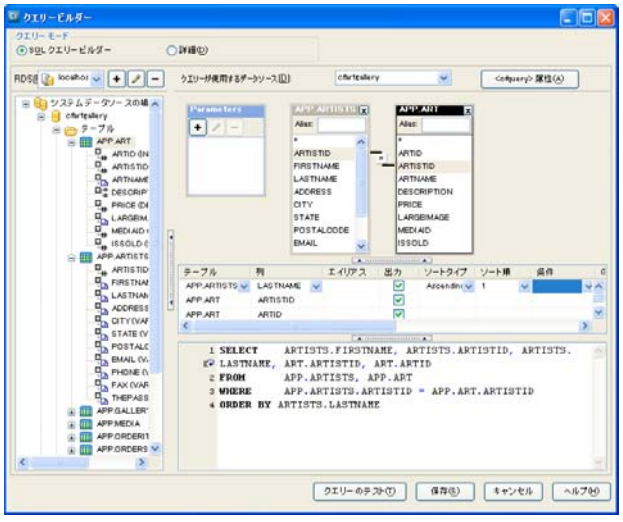
Visual Query Builder の使用方法の詳細については、1325 ページの「Visual Query Builder の使用」を参照してください。

3 クエリをテストするには、[クエリーの実行] をクリックします。

クエリ結果の最初の 50 件が表示されます。

Visual Query Builder の使用

Visual Query Builder を使用して SQL ステートメントを作成します。次の図は、Visual Query Builder のユーザーインターフェイスです。



テーブルペインとプロパティパネルを使用した SQL ステートメントの作成

1 データソースを展開します。

2 SELECT ステートメント内で名前を付ける列をダブルクリックします。

ユーザーが列を選択すると、クエリービルダーはペインの下端にある領域に SELECT ステートメントを作成します。

3 複数のテーブルの列を選択する場合は、それらのテーブルの結合に使用する列を、1 つのテーブルから別のテーブル内の関連する列にドラッグする方法で指定する必要があります。

4 (オプション) 次の手順でソート順を指定します。

a [プロパティ] パネル内の列を指定します。

b ソート基準にする列の [ソートタイプ] セル内をクリックします。

c [Ascending] (昇順) または [Descending] (降順) を指定します。

d (オプション) 複数のソート列を指定する場合は、[ソート順] セルを使用して優先度を指定します。

5 (オプション) 次の手順で選択条件を指定します。

a [プロパティ] パネル内の列を指定します。

b [条件] セルをクリックします。

c [WHERE] をクリックします。

d [基準] セルで WHERE 節の条件を指定します。

注意：選択条件を指定すると、対応する WHERE 節が自動的に作成されます。INNER JOIN またはその他の詳細な選択条件を使用するには、SQL を手動でコーディングする必要があります。

6 (オプション) 集計関数、GROUP BY または式を指定するには、次の操作を行います。

a [プロパティ] パネル内の列を指定します。

b [条件] セルをクリックします。

c Group By または集計関数 (COUNT など) を選択します。

7 (オプション) SQL を手動で指定するには、SQL ペインに目的の SQL ステートメントを入力します。

注意：WHERE 節の代わりに INNER JOIN を使用する場合や、OUTER JOIN やデータベースストアドプロシージャを使用する場合は、SQL を手動で指定します。

8 (オプション) クエリパラメータのデータ型を指定するには、次の操作を行います。

a [Parameters] の下にある [+] ボタンをクリックします。

b パラメータの名前を入力します。

c データ型を選択します。

9 [SQL] ペインに表示される SELECT ステートメントを確認し、必要に応じて [テーブル] ペインと [プロパティ] パネルを使用して修正します。

10 (オプション) [クエリーのテスト] をクリックします。

11「保存」をクリックします。

ColdFusion/Flex アプリケーションウィザード

ColdFusion/Flex アプリケーションウィザードを使用して、CRUD (Create/Read/Update/Delete) アプリケーション用の ColdFusion ファイルと Flex ファイルを作成できます。アプリケーションに含めるマスタページ、詳細ページ、マスタ / 詳細ページと、それらのアプリケーションページ間の関係を指定します。このウィザードでは、Visual Query Builder を使用して SQL ステートメントを作成できます。Visual Query Builder の使用方法の詳細については、1325 ページの「Visual Query Builder の使用」を参照してください。

アプリケーションの設計

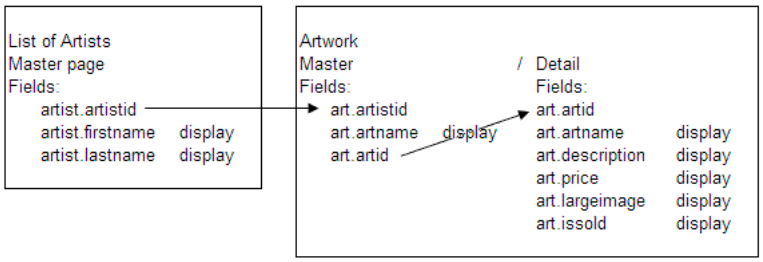
ColdFusion/Flex アプリケーションウィザードを起動する前に、アプリケーションを構成するページについて、次の点も含めて検討する必要があります。

- 各ページをマスタページ、詳細ページ、マスタ / 詳細ページのいずれにするか
- 各ページを表示するフィールド
- ページ間を接続するフィールド

次に紹介する例では、アートギャラリー用のアプリケーションを作成します。最初のページでは、ギャラリーの所属アーティスト全員の名前を一覧に表示します。ユーザーがアーティスト名を選択すると、そのアーティストの全作品を掲載したページが表示されます。ユーザーが作品を選択すると、その作品について詳説するページが表示されます。この例のアプリケーションには、次のページが必要です。

- アーティスト名を一覧表示するマスタページ。
- マスタ / 詳細ページ。マスタページ部には、アーティスト名の一覧用マスタページで選択されたアーティストの作品が一覧表示されます。詳細ページ部には、作品マスタページ部で選択された作品に関する詳細情報が表示されます。

アプリケーションを構成するテーブルやフィールド、そのうちのどのフィールドをアプリケーションに実際に表示するかなどを、次に示すような図にまとめてみると参考になります。



ColdFusion/Flex アプリケーションウィザードの起動

1. RDS サーバーを設定します。詳細については、1323 ページの「RDS の設定」を参照してください。
2. Eclipse または Flash Builder で、[ファイル]-[新規]-[その他] を選択します。
3. [ColdFusion Wizards] の下の [ColdFusion/Flex Application Wizard] を選択して [Next] をクリックします。
4. 概要ページの内容を読んで [Next] をクリックします。
5. ColdFusion/Flex アプリケーションウィザードで以前に作成したアプリケーションから設定を読み込むには、該当する設定ファイル名を選択して [Load ColdFusion/Flex Application Wizard Settings] をクリックします。
6. 「次へ」をクリックします。
7. アプリケーションを配置する RDS サーバーを選択します。
8. 使用するデータソースを指定します。データソースは、ColdFusion Administrator で設定しておきます。

   ここではデフォルトのデータソースを 1 つしか指定しませんが、アプリケーションからは他のデータソースのデータにもアクセスできます。
9. 「次へ」をクリックします。

フォームレイアウトの指定

[Form Layout] ダイアログボックスで、アプリケーションで使用するページを指定できます。マスタ、詳細、マスタ / 詳細ページのいずれかを作成できます。アプリケーション内で、マスタページ、詳細ページおよびマスタ / 詳細ページを次のように関連付けることができます。

ページタイプ	関連付け可能なページ
マスタ	マスタ マスタ / 詳細 詳細 マスタおよび詳細 マスタおよびマスタ / 詳細
マスタ / 詳細	マスタ マスタ / 詳細

ページの作成

1 プラス記号 (+) をクリックします。

2 [Name] テキストボックスにページの名前を入力します。

3 [Page Type] でページタイプを選択します ([Master]、[Detail] または [Master/Detail])。

4 作成するページのタイプに応じて、[Edit Master Page]、[Edit Detail Page]、または [Edit Master Section] をクリックします。

Visual Query Builder が起動します。

5 Visual Query Builder を使用して、ページに含めるデータソース、テーブルおよびフィールドを指定し、[Save] をクリックしてクエリを保存します。Visual Query Builder の使用方法の詳細については、1325 ページの「Visual Query Builder の使用」を参照してください。

6 アプリケーションに含めるページごとに手順 1 から 5 を繰り返します。

7 右矢印ボタンと左矢印ボタンを使用して、アプリケーション内のページの関係を指定します。たとえば、ナビゲーションツリーパネルで、詳細ページは関連するマスタページの直下にインデント表示する必要があります。ツリー構造内では、アイテムをドラッグ & ドロップして移動できます。

8 「次へ」をクリックします。

[Project Information] ページが表示されます。

9 次の情報を指定します。

- コンテキストルート (必要な場合)
- アプリケーションにログインページを含めるかどうか
- プロジェクトで使用する "services-config.xml" 設定ファイルの場所
- Web ルートの URL
- 既存または新規のいずれの Flash Builder または Eclipse プロジェクトを使用するか
- 新規プロジェクトの場合はその名前と場所

10 「終了」をクリックします。

アプリケーションを構成する ColdFusion ファイルと Flex ファイルが自動的に作成されます。このアプリケーションをテストするには、Flash Builder または Eclipse の [Run Main] ボタンをクリックするか、main アプリケーションページをブラウザで表示します。main アプリケーションページの URL は、http://< サーバー名 >:< ポート番号 >/< プロジェクト名 >/bin/main.html です。必要に応じてアプリケーションファイルを適宜修正することもできます。

ColdFusion/Flex アプリケーションウィザードでのアプリケーション作成のヒント

ColdFusion/Flex アプリケーションウィザードを使用すると、CRUD アプリケーションを効率的に作成できるようになりますが、アプリケーションを設計どおりに作成するためには、次の情報を考慮する必要があります。

- ユーザーインターフェイス要素を調整するには、Flash Builder または Eclipse のデザインモードで MXML ファイルを開きます。
- 以前に作成したプロジェクトと同名のプロジェクトを作成すると、以前のプロジェクトのファイルを含むバックアップフォルダが作成されます。
- プライマリキーが定義されていないテーブルを使用してマスタページと詳細ページを作成すると、データベース内の最初のフィールドが行のプライマリキー値として自動的に選択されます。
- マスタページでは、フィールドを [Parameters] ボックスにリンクすると、cfqueryparam タグを使用したタイプ検証がクエリに追加されます。この操作は必須ではありません。
- マスタページでは、プライマリキー列を選択する必要があります。デフォルトでは、キーが自動的に選択されます。マスタページを作成する際、ID プロパティへのリンクを指定しないと、このマスタページをサイトツリー内で別のマスタページの下位に追加することはできません。
- アプリケーションで使用するフィールドのうち、アプリケーション上に実際には表示しないフィールドについては、[Display] 列の選択を解除します。
- ページ上のデータのソート基準とするフィールドについてはソート順を指定し、その他の選択条件も適宜指定します。
- フィールドのラベルを変更するには、[Label] 列で目的のフィールド名をクリックし、新しいフィールド名を入力します。
- 詳細ページでは、ダイナミックデータを挿入するためのコンボボックスを作成します。その場合は、コンボボックスに値を挿入するフィールドの [Input Control] 列を [ComboBox] に変更します。次に、そのフィールドの [Input Lookup Query (sub-select)] 列をクリックし、使用するデータを Visual Query Builder で指定します。
- 詳細ページを作成する際は、プライマリキーの [Display] 列は自動的に選択解除されます。
- 詳細ページを作成する際は、[Input Control] 列にデフォルトで値が割り当てられます。これらのデフォルト値は変更できます。デフォルト値は次のように割り当てられます。
  - ブール値およびビット値は、チェックボックスとして表示されます。
  - メモ値および CLOB 値は、テキスト領域として表示されます。
  - その他のすべての値は、テキスト入力コントロールとして表示されます。

ColdFusion/Ajax アプリケーションウィザード

ColdFusion/Ajax アプリケーションウィザードを使用して、Ajax 要素を含む ColdFusion CRUD (Create/Read/Update/Delete) アプリケーションを作成できます。ウィザードの使用方法の詳細については、1326 ページの「ColdFusion/Flex アプリケーションウィザード」を参照してください。

このウィザードの起動方法は、ColdFusion/Flex アプリケーションウィザードの起動方法とほぼ同じです。手順の中で ColdFusion/Ajax アプリケーションウィザードを選択する点のみが異なります。ColdFusion/Flex アプリケーションウィザードとは異なり、ColdFusion/Ajax アプリケーションウィザードはログイン画面を生成しません。

ActionScript to CFC ウィザード

ActionScript to CFC ウィザードを使用すると、ActionScript クラスファイルに基づいて ColdFusion コンポーネント (CFC) を作成できます。

ActionScript to CFC ウィザードの使用

1 Flash Builder または Eclipse で、プロジェクトナビゲータに移動します。

2 ActionScript クラスファイルを右クリックします。

3 [ColdFusion Wizards]-[Create CFC] を選択します。

4 [Save to Project Folder] テキストボックスに CFC ファイルの保存場所のパスを入力するか、[Browse] ボタンをクリックして場所を選択します。

5 [CFC Filename] テキストボックスに CFC のファイル名を入力します。

6 既存のファイルを置換するには、[Overwrite file if it already exists] チェックボックスをオンにします。

7 CFC 内にプロパティ定義に加えて get メソッドと set メソッドを作成するには、[Generate Get/Set Methods] チェックボックスをオンにします。

8 プロパティのスコープを指定するには、[Property scope] で [public] または [private] を選択します。

9 「Finish」をクリックします。

CFC to ActionScript ウィザード

CFC to ActionScript ウィザードを使用すると、ColdFusion コンポーネント (CFC) のバリューオブジェクトに基づいて ActionScript ファイルを作成できます。

CFC to ActionScript ウィザードの使用

1 Flash Builder または Eclipse で、プロジェクトナビゲータに移動します。

2 CFC バリューオブジェクトファイルを右クリックします。

3 [ColdFusion Wizards]-[Create AS class] を選択します。

4 [Save to Project Folder] テキストボックスに ActionScript ファイルの保存場所のパスを入力するか、[Browse] ボタンをクリックして場所を選択します。

5 [AS Class Package] テキストボックスにクラスパッケージ名を入力します。

6 [AS Class Name] テキストボックスに ActionScript クラスファイルの名前を入力します。

7 既存のファイルを置換するには、[Overwrite file if it already exists] チェックボックスをオンにします。

8 [Path to CFC] テキストボックスに CFC のパスを入力します。

9 ActionScript クラスファイル内に get メソッドと set メソッドを作成するには、[Generate Get/Set Methods] チェックボックスをオンにします。

10 「Finish」をクリックします。

RDS CRUD ウィザード

RDS (Remote Development Services) CRUD ウィザードを使用すると、ColdFusion サーバーの ColdFusion Administrator に登録されたテーブルに基づいて ColdFusion コンポーネント (CFC) をダイナミックに作成できます。RDS CRUD ウィザードを使用するには、Eclipse RDS Support プラグインをインストールする必要があります。Eclipse RDS Support プラグインは、ColdFusion ウィザード群のインストール時に一緒にインストールされます。

RDS CRUD ウィザードを使用して、次のタイプの CFC を作成できます。

- ActiveRecord スタイルの CRUD CFC。プロパティ、get メソッドと set メソッド、SQL メソッドがすべて 1 つの CFC に含まれています。この CFC には、次のメソッドが含まます。

  init() または init(プライマリキー値)

  load(プライマリキー値)

  save()

  delete()

- Bean/DAO スタイルの CRUD CFC。関連する次の 2 つの CFC が作成されます。
  - プロパティ定義および get メソッドと set メソッドを含む Bean CFC (バリューオブジェクトとも呼ばれます)。
  - 次のメソッドを含む DAO CFC。

    read(プライマリキー値)

    create(**cfc** インスタンス)

    update(**cfc** インスタンス)

    delete(**cfc** インスタンス)

- Data Service アセンブラ CFC。Bean (バリューオブジェクト)、DAO CFC およびアセンブラ CFC が含まれています。アセンブラ CFC は、Flex Data Services 機能を利用する場合に必要です。

RDS CRUD ウィザードの使用

1 Flash Builder または Eclipse で、次の手順で RDS データビューに移動します。

   a [ウィンドウ]-[ビューの表示]-[その他] を選択します。

   b [RDS] を選択します。

   c [RDS データビュー] を選択します。

2 テーブル名を右クリックします。

3 [ColdFusion Wizards]-[Create CFC] を選択します。

4 [CFC Folder] テキストボックスに、CFC を保存するプロジェクトフォルダのパスを入力します。

5 [CFC Package Name] テキストボックスに CFC パッケージ名を入力します。

6 (オプション) データベースでプライマリキーが定義されていない場合は、[Primary Key] でプライマリキー列を選択します。

7 (オプション) CFC に指定される他の値に加えてプライマリキー列を指定する場合は、[The primary key is controlled by the user] チェックボックスをオンにします。プライマリキーがデータベースで自動的に生成される場合は (ID フィールド)、このチェックボックスはオフにしてください。

8 既存のファイルを置換するには、[Overwrite files if they already exist] チェックボックスをオンにします。

9 [CFC Type] で次のいずれかを選択します。
   - Active Record CFC
   - Bean CFC & DAO CFC
   - Flex Data Service Assembler CFCs

10 CFC の名前を適切なテキストボックスに入力します。

11 ActionScript バリューオブジェクトを作成するには :

a [Create an ActionScript value object in addition to the CFC(s)] チェックボックスをオンにします。

b [AS Folder] テキストボックスに ActionScript バリューオブジェクトの保存場所のパスを入力するか、[Browse] ボタンをクリックして場所を選択します。

c ActionScript クラスファイル内に get メソッドと set メソッドを作成するには、[Generate Get/Set Methods] チェックボックスをオンにします。

12「Finish」をクリックします。

サービスブラウザー

ColdFusion サービスブラウザを使用して、コンピュータ上の CFC と Web サービスをすべて表示できます。

サービスブラウザの使用

1 Flash Builder または Eclipse で、[ウィンドウ]-[ビューの表示]-[その他] を選択します。

2 [ColdFusion]-[Services Browser] を選択します。

サービスブラウザでは、次の操作を行えます。

- コンポーネントを表示する
- Web サービスを管理する

コンポーネントの表示

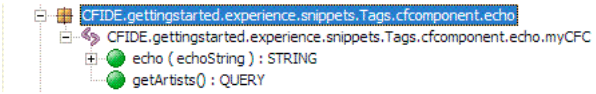
サービスブラウザには、次のコンポーネントが一覧表示されます。

- ColdFusion コンポーネントブラウザに一覧表示されるコンポーネント

  ColdFusion コンポーネントブラウザの場所は、"<ColdFusion のルートディレクトリ >¥wwwroot¥CFIDE¥componentutils¥componentdoc.cfm" です。
- ColdFusion Administrator のマッピングページで指定されたディレクトリに配置されているコンポーネント
- ColdFusion Administrator のカスタムタグのパスページで指定されたディレクトリに配置されているコンポーネント

CFC 内の関数がリモート関数、パブリック関数、プライベート関数のいずれであるかに応じて、一覧に表示する CFC を絞り込むことができます。

次に、一覧に表示される要素の例を示します。



一覧の先頭行にパスが表示されています。2 行目に表示されているのは CFC の名前です。次の 2 行に、CFC 内の関数の名前が表示されています。関数名に続いて、引数、コロン、戻り値のデータ型が表示されます。echo(echoString):STRING という行からは、echo 関数に echoString という引数が指定されており、この関数が文字列を返すことがわかります。この例の myCFC CFC は次のように記述されています。

```
<cfcomponent>
    <cffunction name="echo" output="No" returntype="string">
        <cfargument name="echoString" required="Yes">
        <cfreturn "echo: #arguments[1]#">
    </cffunction>

    <cffunction name="getArtists" returntype="query" hint="query the database and return the
            results.">
        <cfquery name="artists" datasource="cfcodeexplorer">
            select *
            from artists
        </cfquery>
        <cfreturn artists>
    </cffunction>
</cfcomponent>
```

Web サービスの管理

サービスブラウザでは、Web サービスの一覧に対して WSDL URL の追加や削除を行い、この Web サービス一覧を管理できます。また、ColdFusion ファイルを編集する際にサービスブラウザを使用して、Web サービスの呼び出しや Web サービスオブジェクトの作成を行う CFML コードを生成できます。同様に、ActionScript ファイルを編集する際にも、サービスブラウザーを使用して ActionScript を生成できます。

Web サービスの一覧を表示するには、[Services Browser] ビューの右上隅にある [Web サービスの表示] ボタンをクリックします。

一覧への Web サービスの追加

1 [Services Browser] ビュー内を右クリックします。

2 [WSDL の追加] を選択します。

3 有効な WSDL URL を入力します。

4 「OK」をクリックします。

一覧からの Web サービスの削除

1 [Services Browser] ビュー内を右クリックします。

2 [WSDL の削除] を選択します。

ColdFusion ファイル内での Web サービスの呼び出し

1 コードを挿入する位置にマウスポインタを配置します。

2 Web サービスの一覧を表示します。

3 Web サービスまたは Web サービス内のメソッドを強調表示して右クリックします。

4 [CFInvoke の挿入] を選択します。

自動的に生成されたコードが ColdFusion ファイルに表示されます。自動生成されるコードの例を次に示します。

```
<cfinvoke
     webservice="http://arcweb.esri.com/services/v2/MapImage.wsdl"
     method="convertMapCoordToPixelCoord"
     returnVariable="convertMapCoordToPixelCoord" >
    <cfinvokeargument name="mapCoord" type="" />
    <cfinvokeargument name="viewExtent" type="" />
    <cfinvokeargument name="mapImageSize" type="" />
</cfinvoke>
```

ColdFusion ファイル内での Web サービスオブジェクトの作成

1 コードを挿入する位置にマウスポインタを配置します。

2 Web サービスの一覧を表示します。

3 Web サービスまたは Web サービス内のメソッドを強調表示して右クリックします。

4 [CFInvoke の挿入] を選択します。

自動的に生成されたコードが ColdFusion ファイルに表示されます。自動生成されるコードの例を次に示します。

```
createObject("webservice",
"http://arcweb.esri.com/services/v2/MapImage.wsdl").convertMapCoordToPixelCoord(mapCoord, viewExtent,
mapImageSize);
```