Доступ к данным с помощью ADOBE® FLEX® 4.6

Юридическая информация

Последнее обновление 6.12.2011.

Содержание

Глава 1. Обзор доступа к службам данных

Доступ к данным в Flex по сравнению с другими технологиями

Flex обрабатывает источники данных и сами данные иначе, чем приложения, использующие HTML в своих пользовательских интерфейсах.

Обработка на стороне клиента и на стороне сервера

В отличие от набора HTML-шаблонов, создаваемых с использованием JSP и сервлетов, ASP, PHP или CFML, в среде Flex код клиента отделяется от кода сервера. Пользовательский интерфейс приложения компилируется в двоичный файл SWF, отправляемый клиенту.

При отправке приложением запроса в службу данных повторная компиляция файла SWF не производится, также не требуется обновление страницы. Удаленная служба возвращает только данные. Flex обеспечивает привязку возвращенных данных к компонентам пользовательского интерфейса в клиентском приложении.

Например, если в Flex пользователь нажимает элемент управления Button в приложении, код на стороне клиента обеспечит вызов web-службы. Полученные из web-службы данные возвращаются в двоичный файл SWF без обновления страницы. После этого полученные данные будут доступны для использования в приложениях в качестве динамического содержимого.

```
<?xm l version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo" minWidth="1024" minHeight="768"
xmlns:employeesservice="services.employeesservice.*" xmlns:valueObjects="valueObjects.*">

    <fx:Declarations>
        <s:WebService
            id="RestaurantSvc"
        wsdl="http://examples.adobe.com/flex3app/restaurant_ws/RestaurantWS.xml?wsdl" />
        <s:CallResponder id="getRestaurantsResult"
            result="restaurants = getRestaurantsResult.lastResult as Restaurant"/>
    </fx:Declarations>

    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;

            protected function b1_clickHandler(event:MouseEvent):void {
                getRestaurantsResult.token = RestaurantWS.getRestaurantss();
            }
        ]]>
    </fx:Script>
. . .
    <s:Button id="b1" label="GetRestaurants" click="button_clickHandler(event)"/>
```

Сравните данный пример Flex со следующим примером, содержащим код JSP для вызова web-службы с использованием пользовательского тега JSP. При запросе JSP запрос web-службы реализуется на сервере, а не на стороне клиента. Результат используется для создания содержимого HTML-страницы. Сервер приложений обеспечивает повторное создание всей HTML-страницы перед ее отправкой в web-браузер пользователя.

```
<%@ taglib prefix="web" uri="webservicetag" %>

<% String str1="BRL";
String str2="USD";%>

<!-- Call the web service. -->
<web:invoke
    url="http://www.itfinity.net:8008/soap/exrates/default.asp"
    namespace="http://www.itfinity.net/soap/exrates/exrates.xsd"
    operation="GetRate"
    resulttype="double"
    result="myresult">
    <web:param name="fromCurr" value="<%=str1%>"/>
    <web:param name="ToCurr" value="<%=str2%>"/>
</web:invoke>

<!-- Display the web service result. -->
<%= pageContext.getAttribute("myresult") %>
```

Доступ к источникам данных

Еще одно различие между технологиями Flex и других web-приложений состоит в отсутствии непосредственного взаимодействия с источником данных в среде Flex. Компонент доступа к данным используется для подключения к удаленной службе и взаимодействия с источником данных на стороне сервера.

В следующем примере показана страница ColdFusion с прямым доступом к источнику данных:

```
...
<CFQUERY DATASOURCE="Dsn"
    NAME="myQuery">
    SELECT * FROM table
</CFQUERY>
...
```

Для получения подобных функциональных возможностей в среде Flex используется служба HTTPService, web-служба или компонент RemoteObject, которые инициируют вызов объекта на стороне сервера, возвращающего результаты из источника данных.

События, вызовы служб и привязка данных

Технология Flex основана на обработке программных событий. Действие пользователя или программное событие могут инициировать обращение к службе. Например, когда пользователь нажимает кнопку, создается событие действия пользователя, которое может использоваться в качестве триггера для вызова функции. Примером события программы может являться завершение создания компонента пользовательского интерфейса в приложении, например, DataGrid. Событие creationComplete для DataGrid может использоваться при вызове удаленной службы для заполнения DataGrid.

Вызовы служб в среде Flex являются асинхронными. Ожидание возвращаемых данных клиентским приложением не требуется. Асинхронные вызовы служб обеспечивают преимущества при извлечении или обновлении больших наборов данных. Клиентское приложение не блокируется при ожидании данных, которые необходимо извлечь или обновить.

Данные, возвращаемые после вызова службы, сохраняются в свойстве CallResponder, связанном с вызовом службы. После этого в компонентах пользовательского интерфейса выполняется привязка данных для извлечения возвращенных данных из свойства CallResponder.

Привязка данных в среде Flex позволяет осуществлять динамическое обновление компонента пользовательского интерфейса вместе с источником данных. Например, компонент Flex может обеспечивать связывание собственного текстового атрибута с атрибутом lastResult свойства CallResponder. Компонент Flex автоматически обновляется одновременно с изменением данных в CallResponder.

В среде Flex также реализована двунаправленная привязка данных. Если используется двунаправленная привязка данных, при внесении изменений в данные в компоненте Flex или в источнике данных осуществляется автоматическое обновление соответствующего источника данных или компонента Flex. Применение двунаправленной привязки данных рекомендуется при обновлении удаленных данных в соответствии с данными, вводимыми пользователем в компонент Form или в компонент данных Flex.

Дополнительные разделы справки

«Создание ориентированных на данные приложений с помощью Flash Builder» на странице 8

Использование Flash Builder для доступа к службам данных

В версии Flex Builder 3 реализация удаленных вызовов процедур для служб данных осуществляется с использованием компонентов доступа к данным Flex. С помощью Flash Builder этот процесс упрощается.

Flash Builder предоставляет мастеры и другие инструменты для:

- обеспечения доступа к службам данных;
- настройки данных, возвращаемых службой данных;
- помощи в подкачке страниц для данных, возвращаемых службой;
- помощи в управлении данными, которое синхронизирует многочисленные обновления данных сервера;
- создания клиентского кода для доступа к службам данных;
- привязки данных, возвращаемых службой компонентам пользовательского интерфейса.

Рабочий процесс Flash Builder для доступа к службам

Ниже представлен рабочий процесс, который используется при создании приложения, обеспечивающего доступ к службам данных, в Flash Builder.

1 В зависимости от обстоятельств процесс начинается с подключения к службе данных или с создания пользовательского интерфейса.

Соединение с удаленной службой: если процесс начинается с соединения с удаленной службой, на следующем этапе необходимо создать пользовательский интерфейс.

Создание пользовательского интерфейса: если процесс начинается с создания пользовательского интерфейса, на следующем этапе необходимо выполнить соединение с удаленной службой.

Примечание. *Выбор первоначальных действий зависит от личных предпочтений. Например, при наличии запланированного проекта интерфейса пользователя можно начать с создания пользовательского интерфейса. И наоборот, можно сначала подключиться к данным, а затем с помощью Flash Builder создать компоненты приложения.*

2 Привязка операций с данными к компонентам приложения

В Flash Builder реализовано несколько способов, обеспечивающих поддержку пользователя при работе с операциями по привязке данных к компонентам приложения. Flash Builder обеспечивает:

- создание различных форм для данных, возвращаемых операциями службы;
- выбор операций службы для привязки к компонентам пользовательского интерфейса;
- создание формы для представления сложных данных, возвращаемых службой.

3 (Необязательно) Управление поиском и обновлением данных

Инструменты Flash Builder позволяют реализовывать подкачку страниц для возвращаемых данных и координировать обновление наборов данных.

При возвращении большого количества записей данных подкачка страниц обычно реализуется в целях извлечения набора записей по мере необходимости.

Для приложений, обеспечивающих обновление нескольких записей, могут быть реализованы функции управления данными. Возможности управления данными включают в себя:

- функциональные возможности для одновременного обновления измененных записей;
- механизм отмены изменений перед их отправкой на сервер;
- создание кода для автоматического обновления компонентов пользовательского интерфейса при добавлении, удалении или изменении записей.

4 Запуск приложения и контроль потока данных.

Созданное приложение запускается и проверяется в действии. Монитор сети Flash Builder может использоваться для просмотра данных, передаваемых между приложением и службой. Применение монитора сети рекомендуется для диагностирования ошибок и анализа производительности.

Flash Builder также обеспечивает надежные среды отладки и профилирования. Монитор сети и профилировщик Flash доступны в версии Flash Builder Premium.

Дополнительные разделы справки

«Создание ориентированных на данные приложений с помощью Flash Builder» на странице 8

Расширение служб, поддерживаемых Flash Builder

Мастеры и инструментальные средства Flash Builder поддерживают доступ к реализациям служб следующих типов:

- службы PHP
- службы ColdFusion
- BlazeDS
- ADEP Data Services (прежнее название: LiveCycle Data Services)
- службы HTTP (вида REST)
- web-службы (SOAP)
- статичные файлы XML.

Если требуется инструментальная поддержка дополнительных типов служб, таких как Ruby on Rails, можно расширить реализацию Flash Builder. См. документ Flash Builder Extensibility Reference.

Компоненты доступа к данным

Компоненты доступа к данным позволяют клиентским приложениям выполнять вызовы операций и служб в сети. Компоненты доступа к данным используют удаленные вызовы процедур для взаимодействия со средами сервера. Существует три компонента доступа к данным – RemoteObject, HTTPService и WebService.

Компоненты доступа к данным разработаны для клиентских приложений, в которых при доступе к внешним данным целесообразно использовать модель вызова и ответа. Эти компоненты позволяют клиентам выполнять асинхронные запросы к удаленным службам, обрабатывающим запросы, а затем возвращать данные в приложение.

Компонент доступа к данным вызывает удаленную службу. Затем он сохраняет данные ответа службы в объекте ActionScript или в любом другом формате, возвращаемом службой. Компоненты доступа к данным в клиентском приложении используются для работы с тремя типами служб:

- службы удаленных объектов (RemoteObject);
- web-службы на основе SOAP (WebService);
- службы HTTP, включая web-службы на основе REST (HTTPService).

Adobe® Flash® Builder™ предоставляет мастеры и инструменты для включения реализации компонента доступа к данным в оболочку службы. Оболочка службы инкапсулирует функциональные возможности компонента доступа к данным, предоставляя защиту от большого числа реализаций данных нижнего уровня. Это позволяет сконцентрировать внимание на реализации служб и создании клиентских приложений для доступа к службам. Для получения более подробной информации по использованию Flash Builder для доступа к службам данных см. раздел «Создание ориентированных на данные приложений с помощью Flash Builder» на странице 8.

Обеспечение доступа к службам

Adobe Flash Player по умолчанию блокирует доступ к любому хосту, неидентичному тому, что был использован для загрузки приложения. Если приложение на стороне сервера, например ADEP Data Services или BlazeDS, не используется для запросов прокси, то на сервере, где расположено приложение, должна присутствовать служба HTTP или web-служба, либо удаленный сервер, на котором расположена служба HTTP или web-служба, должен определить файл `crossdomain.xml`. Файл `crossdomain.xml` предоставляет серверу путь для указания доступности данных и документов для файлов SWF, обслуживаемых из некоторых или всех доменов. Файл `crossdomain.xml` должен находиться в корневом web-каталоге сервера, к которому приложение устанавливает связь.

Компоненты HTTPService

Компоненты HTTPService используются для отправки запросов HTTP GET или POST и включают данные из ответов HTTP в клиентское приложение. При создании в Flex настольных приложений (выполняемых в Adobe AIR®) поддерживаются HTTP PUT и DELETE.

При наличии ADEP Data Services или BlazeDS может использоваться служба HTTPProxyService, которая позволяет работать с дополнительными методами HTTP. Посредством HTTPProxyService отправляются запросы GET, POST, HEAD, OPTIONS, PUT, TRACE или DELETE.

Служба HTTP может быть любым URI HTTP, принимающим запросы HTTP и отправляющим ответы. Другим общеупотребительным именем службы этого типа является web-служба вида REST. REST расшифровывается как Representational State Transfer («Передача состояния представления») и является стилем архитектуры для распределенных гипермедийных систем.

Компоненты HTTPService являются оптимальным выбором, когда невозможно использовать те же функциональные возможности, что предоставляются web-службой SOAP или удаленной службой объектов. Например, можно использовать компоненты HTTPService для взаимодействия с JavaServer Pages (JSP), сервлетами, а также ASP-страницами, не доступными в качестве web-служб или адресов назначения удаленных служб.

При вызове метода объекта HTTPService `send()` выполняется запрос HTTP к указанному URI и возвращается ответ HTTP. Дополнительно на указанный URI можно передать аргументы.

Flash Builder обеспечивает рабочие процессы, устанавливающие интерактивную связь со службами HTTP. Для получения дополнительной информации см. раздел «Доступ к службам HTTP» на странице 13.

Дополнительные разделы справки

«Доступ к службам HTTP» на странице 13

Диссертация: Representational State Transfer (REST), автор: Roy Thomas Fielding

Компоненты WebService

Компоненты WebService обеспечивают доступ к web-службам SOAP, являющимся модулями программного обеспечения с методами. Методы web-служб обычно называются *операциями*. Интерфейсы web-служб определяются с помощью языка описания web-служб (Web Services Description Language, WSDL). С помощью web-служб модули программного обеспечения, выполняющиеся на различных платформах, могут взаимодействовать друг с другом в соответствии со стандартами. Дополнительную информацию о web-службах см. в соответствующем разделе сайта консорциума World Wide Web по адресу www.w3.org/2002/ws/.

Клиентские приложения могут взаимодействовать с web-службами, определяющими интерфейсы в документе на языке описания web-служб (WSDL), доступном в виде URL-адреса. WSDL является стандартным форматом описания сообщений, распознаваемых web-службой, форматом ее ответов на эти сообщения, протоколов, поддерживаемых web-службой, и адресов для отправки сообщений.

Flex поддерживает WSDL 1.1, описание которого приведено по адресу www.w3.org/TR/wsdl. Flex поддерживает как web-службы с кодировкой RPC, так и web-службы документов-литералов.

Flex поддерживает запросы и результаты web-службы, отформатированные в виде сообщений SOAP и передающиеся через HTTP. Протокол SOAP обеспечивает определение формата на основе XML, который может применяться для обмена структурированной и типизированной информацией между клиентом web-службы, например, приложением, созданным в Flex, и web-службой.

Компонент WebService можно использовать для соединения с web-службой, совместимой с SOAP, если в данной среде web-службы являются установленным стандартом. Компоненты WebService также используются для объектов, которые находятся в корпоративной среде, однако не обязательно содержатся в исходном пути web-приложения.

Flash Builder обеспечивает рабочие процессы, устанавливающие интерактивную связь с web-службами. Подробную информацию см. в разделе «Доступ к web-службам» на странице 16.

Компоненты RemoteObject

Службы удаленных объектов позволяют осуществлять непосредственный доступ к бизнес-логике в ее собственном формате вместо форматирования логики в XML, что происходит при применении служб вида REST или web-служб. Это экономит время, которое бы потребовалось на представление существующей логики в XML-формате. Другим преимуществом служб удаленных объектов является скорость связи с использованием проводных средств. Обмен данными по-прежнему осуществляется по протоколу HTTP или https, однако сами данные приводятся в двоичное представление. В результате использования компонентов RemoteObject по проводам передается меньший объем данных, сокращается объем памяти, используемой на стороне клиента, и уменьшается время обработки.

При обращении к данным сервера в ColdFusion, PHP, BlazeDS и ADEP Data Services может применяться типизация данных на стороне сервера. Клиентское приложение выполняет доступ к объекту Java, компоненту ColdFusion (который является объектом Java на внутреннем уровне) или классу PHP непосредственно путем удаленного вызова метода для обозначенного объекта. Объект на сервере использует собственные типы данных в качестве аргументов, выполняет запросы базы данных с этими аргументами и возвращает значения в собственных типах данных.

Если типизация данных на стороне сервера невозможна, следует применять инструментальные средства Flash Builder для реализации типизации на стороне клиента. Flash Builder используется для определения и настройки типов данных, возвращаемых службой. Типизация на стороне клиента позволяет клиентскому приложению осуществлять запрос в базу данных и извлекать соответственно типизированные данные. Типизация данных на стороне клиента требуется для любой службы, не обеспечивающей определение типа данных, возвращаемых службой.

Flash Builder обеспечивает рабочие процессы, устанавливающие интерактивную связь с удаленными службами объектов. Для получения дополнительной информации см. раздел «Подключение к службам данных» на странице 9.

Глава 2. Создание ориентированных на данные приложений с помощью Flash Builder

С помощью инструментов Flash Builder можно создавать приложения, выполняющие доступ к службам данных. Сначала необходимо создать проект Flex для пользовательских приложений. Затем следует соединиться со службой данных, настроить доступ к данным из службы и создать пользовательский интерфейс для приложения. Иногда сначала требуется создать пользовательский интерфейс и затем выполнить доступ к службе данных.

Создание проекта Flex для доступа к службам данных

Flex выполняет доступ к службам данных как удаленный объект, служба HTTP (вида REST) или web-служба SOAP.

Удаленный объект используется для доступа к следующим типам служб данных:

- службы ColdFusion
- PHP-службы на базе AMF
- BlazeDS
- ADEP Data Services (прежнее название: LiveCycle Data Services)

Для получения информации по использованию мастера LiveCycle Service Discovery см. раздел Использование LiveCycle Discovery.

Для каждой службы, доступ к которой выполняется как удаленный объект, создается проект Flex, настроенный для соответствующего типа сервера приложения. Мастер новых проектов Flex помогает настроить проект для перечисленных ниже типов сервера приложения:

Тип сервера	**Поддерживаемые службы удаленных объектов**
PHP	• PHP-службы на базе AMF
ColdFusion	• ColdFusion Flash Remoting • BlazeDS • ADEP Data Services
J2EE	• BlazeDS • ADEP Data Services

Соединение со службами HTTP (вида REST) и web-службами SOAP можно выполнить из любой конфигурации проекта Flex, в том числе из проектов, для которых не указана технология сервера.

Проект, конфигурация которого предполагает доступ к удаленному объекту, может выполнить доступ только к той службе удаленных объектов, для которой он был настроен. Например, доступ к службе PHP на основе AMF невозможно выполнить из проекта, сконфигурированного для ColdFusion. Однако этот проект может соединиться со службой PHP, если данное соединение выполняется как к web-службе или службе HTTP.

Дополнительные разделы справки

«Обзор доступа к службам данных» на странице 1

Изменение типа сервера проекта

Flash Builder выводит уведомление при попытке доступа к службе, для которой не настроен проект Flex. Если для проекта Flex не указана правильная конфигурация сервера, Flash Builder выводит ссылку на диалоговое окно «Свойства проекта». В диалоговом окне «Свойства проекта» выполняется настройка проекта для доступа к службе данных. Так, Flash Builder выводит предупреждение при попытке пользователя выполнить доступ к службе PHP на основе AMF из проекта, для которого не указана конфигурация сервера.

Если проект Flex был настроен для доступа к другому типу службы, настройте новый проект Flex или измените конфигурацию текущего проекта. При изменении конфигурации сервера проекта доступ к настроенным ранее службам будет невозможен. Например, при изменении конфигурации проекта с ColdFusion на PHP ни одна из служб ColdFusion больше не будет доступна.

Если для данного проекта требуется доступ к другим типам служб, то используются такие службы, как службы HTTP или web-службы.

Файл междоменной политики

При осуществлении доступа к службам, находящимся в другом домене по отношению к файлу SWF, для приложения необходимо использовать файл междоменной политики. Как правило, для служб на базе AMF не требуется файл междоменной политики, поскольку эти службы располагаются в одном домене с приложением.

Подключение к службам данных

Для подключения к службам данных используется мастер служб Flash Builder.

Для служб удаленных объектов обычно создается проект Flex с соответствующим типом сервера приложения. В Flash Builder выполняется внутренний анализ службы и настраиваются типы данных, возвращаемых службой.

Службы удаленных объектов включают в себя службы данных, реализованные в ColdFusion, PHP, BlazeDS и ADEP Data Services (прежнее название: LiveCycle Data Services).

Для получения информации по использованию мастера LiveCycle Service Discovery см. раздел Использование LiveCycle Discovery.

Дополнительные разделы справки

«Создание проекта Flex для доступа к службам данных» на странице 8

Доступ к службам ColdFusion

Мастер службы Flash Builder используется для доступа к службе данных ColdFusion, реализованной как компонент ColdFusion (CFC). Flex выполняет доступ к этим службам как к удаленным объектам.

Используйте проект Flex, указывающий ColdFusion в качестве типа сервера приложения. При создании проекта Flex укажите «Использовать службу доступа к удаленным объектам» и выберите «ColdFusion Flash Remoting».

Подключение к службам данных ColdFusion

Для данной процедуры требуется реализованная служба ColdFusion и проект Flex, созданный для доступа к службам ColdFusion.

1 Откройте окно службы, выбрав в меню данных Flash Builder пункт «Соединить с ColdFusion».

2 В диалоговом окне «Настроить службу ColdFusion» перейдите к местоположению компонента CFC, реализующего эту службу.

Примечание. *Если служба ColdFusion не реализована, Flash Builder может создать типовую службу из одной таблицы базы данных. Используйте созданный типовой файл в качестве примера доступа к службам данных. См. раздел «Создание типовой службы ColdFusion из таблицы базы данных» на странице 10.*

3 (Дополнительно) Измените параметры службы.

Имя службы	Введите имя службы. Flash Builder создает имя для службы на основе имени файла данной службы. К именам, используемым для службы, применяются ограничения. См. раздел «Именование служб данных» на странице 20.
Пакет службы	Укажите имя пакета, в котором содержатся файлы ActionScript для доступа к службе. Flash Builder создает пакет на основе имени службы и помещает его в пакет `services`.
Пакет типов данных	Укажите имя пакета, в котором содержатся созданные файлы класса ActionScript, определяющие типы данных, извлеченных из службы. По умолчанию Flash Builder создает пакет `valueObjects`.

4 (Дополнительно) Нажмите «Далее» для просмотра операций службы.

5 Нажмите «Готово» для создания файлов ActionScript, обеспечивающих доступ к службе.

Примечание. *После подключения к службе можно изменить свойства этой службы. Выберите службу в представлении «Данные/службы». В контекстном меню выберите «Свойства».*

Следующий шаг: «Настройка типов данных для операций службы данных» на странице 28.

Создание типовой службы ColdFusion из таблицы базы данных

В Flash Builder можно создать типовую службу ColdFusion, используемую в качестве прототипа для пользовательских служб. Типовая служба выполняет доступ к одной таблице базы данных и располагает методами для создания, чтения, обновления и удаления.

Flash Builder настраивает типы возвращаемых данных для созданных служб и включает функции доступа к данным, такие как подкачка страниц и управление данными.

Важная информация. *Используйте созданную службу только в доверенной среде разработки. Посредством созданного кода любой пользователь с сетевым доступом к вашему серверу получит возможность вызывать, изменять или удалять информацию из таблицы базы данных. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. См. документ Securing Data Services для получения информации о создании безопасных служб.*

Для данной процедуры требуется проект Flex, созданный для доступа к службам ColdFusion, и доступные источники данных ColdFusion.

1 Откройте окно службы, выбрав в меню данных Flash Builder пункт «Соединить с ColdFusion».

2 Для создания типовой службы щелкните по ссылке в диалоговом окне «Настроить службу ColdFusion».

3 Выберите «Создать из источника данных RDS» и укажите источник данных и таблицу для ColdFusion.

Если первичный ключ в таблице не определен, его следует выбрать для таблицы.

Примечание. *При отсутствии доступного источника данных ColdFusion выберите «Создать из шаблона». Flash Builder создает типовой компонент ColdFusion (CFC) со стандартными операциями служб. Раскомментируйте особые функции в CFC и измените операции, чтобы создать типовую службу, которая будет использоваться в качестве прототипа.*

4 Используйте местоположение по умолчанию или укажите новое местоположение. Нажмите кнопку «OK».

В Flash Builder будет создана типовая служба. Укажите другое имя службы и местоположение пакетов, если требуется изменить значения, заданные по умолчанию.

5 (Дополнительно) Нажмите «Далее» для просмотра операций в службе.

6 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к типовой службе. Flash Builder откроет типовую службу в установленном на компьютере пользователя редакторе, который зарегистрирован для редактирования файлов ColdFusion CFC.

Доступ к службам PHP

Для подключения к службам данных, реализованным в PHP, используется мастер служб Flash Builder. Flex использует формат Action Message Format (AMF) для сериализации данных между клиентским приложением и службой данных. Для предоставления доступа к службам, реализованным в PHP, приложение Flash Builder устанавливает инфраструктуру Zend AMF. См. раздел «Установка Zend Framework» на странице 21.

Доступ к службам данных PHP выполняется из проекта Flex, для которого указан PHP в качестве типа сервера приложения. Служба данных должна находиться в корневом web-каталоге, указанном при настройке проекта для PHP. Поместите службу в каталог служб, как показано ниже:

```
<webroot>/MyServiceFolder/services
```

Дополнительные разделы справки

«Создание проекта Flex для доступа к службам данных» на странице 8

Подключение к службам данных PHP

Для данной процедуры требуется реализованная служба PHP и проект Flex, созданный для доступа к службам PHP.

1 Откройте мастер службы, выбрав в меню данных Flash Builder пункт «Соединить с PHP».

2 В диалоговом окне «Настроить службу PHP» перейдите к файлу PHP, реализующему эту службу.

Примечание. *При отсутствии реализованной службы PHP в Flash Builder можно создать типовую службу из одной таблицы базы данных. Используйте созданный типовой файл в качестве примера доступа к службам данных. См. раздел «Создание типовой службы PHP из таблицы базы данных» на странице 12.*

3 (Дополнительно) Измените параметры службы.

Имя службы	Введите имя службы. Flash Builder создает имя для службы на основе имени файла данной службы. К именам, используемым для службы, применяются ограничения. См. раздел «Именование служб данных» на странице 20.
Пакет службы	Укажите имя пакета, в котором содержатся файлы ActionScript для доступа к службе. Flash Builder создает пакет на основе имени службы и помещает его в пакет `services`.
Пакет типов данных	Укажите имя пакета, в котором содержатся созданные файлы класса ActionScript, определяющие типы данных, извлеченных из службы. По умолчанию Flash Builder создает пакет `valueObjects`.

4 Нажмите «Далее» для просмотра операций службы.

При отсутствии поддерживаемой версии Zend Framework для выполнения доступа к службам PHP программа Flash Builder предложит пользователю установить минимальную версию Zend Framework. См. раздел «Установка Zend Framework» на странице 21.

5 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к службе.

Примечание. *После подключения к службе можно изменить свойства этой службы. Выберите службу в представлении «Данные/службы». В контекстном меню выберите «Свойства».*

Следующий шаг: «Настройка типов данных для операций службы данных» на странице 28.

Создание типовой службы PHP из таблицы базы данных

В Flash Builder можно создать типовую службу PHP, используемую в качестве прототипа для пользовательских служб. Типовая служба выполняет доступ к одной таблице базы данных MySQL и располагает методами для создания, чтения, обновления и удаления.

Flash Builder настраивает типы возвращаемых данных для созданных служб и включает функции доступа к данным, такие как подкачка страниц и управление данными.

Важная информация. *Используйте созданную службу только в доверенной среде разработки. Посредством созданного кода любой пользователь с сетевым доступом к вашему серверу получит возможность вызывать, изменять или удалять информацию из таблицы базы данных. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. См. документ Securing Data Services для получения информации о создании безопасных служб.*

Для данной процедуры требуется проект Flex, созданный для доступа к службам PHP, и доступный источник данных MySQL.

1 Откройте мастер службы, выбрав в меню данных Flash Builder пункт «Соединить с PHP».

2 Для создания типовой службы щелкните по ссылке в диалоговом окне «Настроить службу PHP».

3 Выберите «Создать из шаблона» и укажите сведения для соединения с базой данных. Нажмите «Соединить с базой данных».

Примечание. *При отсутствии доступного источника данных PHP выберите «Создать из шаблона». Flash Builder создает типовой проект со стандартными операциями служб. Раскомментируйте особые области в проекте и измените операции, чтобы создать типовую службу, которая будет использоваться в качестве прототипа.*

4 Выберите таблицу в базе данных и укажите первичный ключ.

5 Используйте местоположение по умолчанию или укажите новое местоположение. Нажмите кнопку «OK».

При отсутствии поддерживаемой версии Zend Framework для выполнения доступа к службам PHP программа Flash Builder предложит пользователю установить минимальную версию Zend Framework. См. раздел «Установка Zend Framework» на странице 21.

В Flash Builder будет создана типовая служба. Укажите другое имя службы и местоположение пакетов, если требуется изменить значения, заданные по умолчанию.

6 (Дополнительно) Нажмите «Далее» для просмотра операций в службе.

7 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к типовой службе. Flash Builder откроет типовую службу в установленном на компьютере пользователя редакторе, который зарегистрирован для редактирования файлов PHP.

Доступ к службам HTTP

Для подключения к службам HTTP на основе REST используется мастер служб Flash Builder. Подключение к службам HTTP возможно из любого проекта Flex. Указывать технологию сервера для проекта не требуется.

При осуществлении доступа к службам, находящимся в другом домене по отношению к файлу SWF, для клиентского приложения необходимо использовать файл междоменной политики. См. раздел Использование файлов междоменной политики.

Настройка служб HTTP

Доступ к службам HTTP на основе REST можно настроить различными способами. Мастер «Настроить службу HTTP» поддерживает следующие варианты:

- Основной URL-адрес в качестве префикса

  Использование основного URL-адреса в качестве префикса удобно при доступе к нескольким операциям из одной службы. Если указан основной URL-адрес для службы, то для каждой операции определяется только относительный путь к операциям HTTP.

  Основной URL-адрес не может использоваться для получения доступа к нескольким службам.

- URL-адреса с параметрами запроса

  При указании URL-адреса для операции можно включить параметры запроса для операций службы. Мастер «Настроить службу HTTP» заполняет таблицу параметров данными каждого параметра, включенного в URL-адрес для операции.

- Службы RESTful с разделенными параметрами

  В Flash Builder поддерживается доступ к службам RESTful, использующим разделенные параметры вместо параметра запроса GET. Например, предположим, что следующий URL-адрес используется для получения доступа к службе RESTful:

  ```
  http://restfulService/items/itemID
  ```

В фигурных скобках ({}) указываются параметры в URL-адресе операции. Например:

`http://restfulService/{items}/{itemID}`

Затем с помощью мастера «Настроить службу HTTP» заполняется таблица параметров:

Name	Тип данных	Тип параметра
items	String	URL-адрес
itemID	String	URL-адрес

При определении параметров службы RESTful значения «Тип данных» и «Тип параметра» всегда сконфигурированы как String и URL-адрес соответственно.

Примечание. *Параметры службы RESTful можно сочетать с параметрами запроса при указании URL-адреса для операции.*

- Путь к локальному файлу для URL-адреса операции

  Для URL-адреса операции можно указать путь к локальному файлу, реализующему службы HTTP. Например, укажите следующий URL-адрес для операции:

  `c:/MyHttpServices/MyHttpService.xml`

- Добавление операций GET и POST

  При конфигурации службы HTTP можно добавить дополнительные операции. Для добавления операции нажмите кнопку «Добавить» в таблице операций.

  Укажите GET или POST в качестве метода операции.

- Добавление параметров к операции

  В таблице операций можно добавлять параметры к выбранным операциям. Выберите операцию в таблице операций и нажмите кнопку «Добавить» в таблице параметров.

  Укажите имя и тип данных для добавляемого параметра. Тип параметра - GET или POST - соответствует методу операции.

- Тип содержимого для операций POST

  Для операций POST можно указать тип содержимого. Типом содержимого может быть `application/x-www-form-urlencoded` или `application/xml`.

  Если в качестве типа содержимого выбирается `applicaton/xml`, то в Flash Builder создается параметр запроса, который не подлежит редактированию. strXML является именем по умолчанию. Действительный параметр указывается во время выполнения.

Name	Тип данных	Тип параметра
strXML	String	POST

Пользователь не может добавлять дополнительные параметры для типа содержимого `application/xml`.

Подключение к службам HTTP

1 Откройте мастер службы, выбрав в меню данных Flash Builder пункт «Соединить с HTTP».

2 (Дополнительно) Укажите основной URL-адрес, который будет использоваться в качестве префикса для всех операций.

3 В разделе «Операции» укажите следующие сведения для каждой операции, к которой будет выполняться доступ:

- Укажите метод операции (GET или POST).
- Укажите URL-адрес операции службы.

  Включите любые параметры для операции в URL-адрес. Используйте фигурные скобки ({}) для указания параметров службы вида REST.

  Flash Builder поддерживает доступ к следующим протоколам:

  `http://`

  `https://`

  стандартные абсолютные пути, такие как `C:/` или `/Applications/`
- Введите имя операции.

4 Для каждого параметра операции в выбранном URL-адресе укажите имя и тип данных параметра.

5 (Дополнительно) Нажмите «Добавить» или «Удалить» для добавления или удаления параметров выбранной операции.

6 (Дополнительно) Измените параметры службы.

Имя службы	Введите имя службы. Flash Builder создает имя для службы на основе имени файла данной службы. К именам, используемым для службы, применяются ограничения. См. раздел «Именование служб данных» на странице 20
Пакет службы	Укажите имя пакета, в котором содержатся файлы ActionScript для доступа к службе. Flash Builder создает пакет на основе имени службы и помещает его в пакет `services`.
Пакет типов данных	Укажите имя пакета, в котором содержатся созданные файлы класса ActionScript, определяющие типы данных, извлеченных из службы. По умолчанию Flash Builder создает пакет `valueObjects`.

7 (Необязательно) Измените имя пакета, созданного для службы.

8 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к службе.

После подключения к службе HTTP настройте тип возвращаемых данных для операций службы. Во время настройки типа возвращаемых данных также можно настроить тип параметров для операции. См. раздел «Настройка типов данных для операций службы данных» на странице 28.

Примечание. *После подключения к службе можно изменить свойства этой службы. Выберите службу в представлении «Данные/службы». В контекстном меню выберите «Свойства».*

Следующий шаг: «Настройка типов данных для операций службы данных» на странице 28.

Доступ к файлу XML, реализующему службы HTTP

Пользователь может выполнить доступ к статическому файлу XML, реализующему службу HTTP. Этот статический файл XML может быть локальным файлом или быть доступным через URL-адрес.

В службе используется метод GET, возвращающий ответ XML. Эта функция используется при знакомстве со службами HTTP в Flex и при создании прототипов подстановочных данных в клиентских приложениях.

При выполнении доступа к службе укажите узел, возвращающий ответ XML. Этот узел используется в Flash Builder для автоматической настройки типа возвращаемых данных. После подключения к службе операции службы можно привязать к компонентам пользовательского интерфейса.

Подключение к файлу службы XML

1 Откройте мастер службы, выбрав в меню данных Flash Builder пункт «Соединить с HTTP».

2 Укажите локальный файл или URL-адрес и перейдите к файлу.

3 Выберите в файле узел, содержащий требуемый ответ.

Укажите, является ли ответ массивом.

После этого в Flash Builder будет настроен тип возвращаемых данных для выбранного узла.

4 Измените параметры службы.

Имя службы	Введите имя службы. Flash Builder создает имя для службы на основе имени файла данной службы. К именам, используемым для службы, применяются ограничения. См. раздел «Именование служб данных» на странице 20
Пакет службы	Укажите имя пакета, в котором содержатся файлы ActionScript для доступа к службе. Flash Builder создает пакет на основе имени службы и помещает его в пакет `services`.
Пакет типов данных	Укажите имя пакета, в котором содержатся созданные файлы класса ActionScript, определяющие типы данных, извлеченных из службы. По умолчанию Flash Builder создает пакет `valueObjects`.

5 (Необязательно) Измените имя пакета, созданного для службы.

6 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к службе.

Примечание. *После подключения к службе можно изменить свойства этой службы. Выберите службу в представлении «Данные/службы». В контекстном меню выберите «Свойства».*

Доступ к web-службам

Для подключения к web-службам (SOAP) используется мастер служб Flash Builder. Подключение к web-службам возможно из любого проекта Flex. Указывать технологию сервера для проекта не требуется.

При осуществлении доступа к службам, находящимся в другом домене по отношению к файлу SWF, для клиентского приложения необходимо использовать файл междоменной политики.

Дополнительные разделы справки

Использование файлов междоменной политики

Подключение к web-службам

1 Откройте мастер службы, выбрав в меню данных Flash Builder пункт «Web-служба».

2 (ADEP Data Services/BlazeDS) При установленной службе ADEP Data Services или BlazeDS доступ к web-службе можно выполнить через прокси.

Выберите пункт «Через адрес назначения прокси LCDS/BlazeDS».

Укажите адрес назначения. Нажмите кнопку «Далее» и перейдите к шагу 5.

Примечание. *Доступ к web-службам через прокси активирован только в том случае, если для проекта Flex указан J2EE в качестве типа сервера приложения.*

3 Введите URI для службы SOAP.

4 (Дополнительно) Измените параметры службы.

Имя службы	Введите имя службы. Flash Builder создает имя для службы на основе WSDL URI. К именам, используемым для службы, применяются ограничения. См. раздел «Именование служб данных» на странице 20
Пакет службы	Укажите имя пакета, в котором содержатся файлы ActionScript для доступа к службе. Flash Builder создает пакет на основе имени службы и помещает его в пакет `services`.
Пакет типов данных	Укажите имя пакета, в котором содержатся созданные файлы класса ActionScript, определяющие типы данных, извлеченных из службы. По умолчанию Flash Builder создает пакет `dataValues`.

5 (Дополнительно) Настройте создание кода для службы:

Служба **Порт**	Выберите службу из доступных служб. Flash Builder создает имя для службы на основе WSDL URI.
Список операций	Выберите операции из службы, к которой требуется получить доступ в клиентском приложении.

6 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к службе.

Примечание. *После подключения к службе можно изменить свойства этой службы. Выберите службу в представлении «Данные/службы». В контекстном меню выберите «Свойства».*

‘После подключения к web-службе настройте типы возвращаемых данных для операций службы. Подробную информацию см. в разделе «Настройка типов данных для операций службы данных» на странице 28.

Доступ к BlazeDS

Доступ к службам BlazeDS возможен только в том случае, если установлена служба Adobe® BlazeDS и настроен сервер удаленной разработки (Remote Development Services, RDS). Для получения сведений по установке и настройке BlazeDS см. документацию к ADEP Data Services ES.

Как правило, доступ к службам данных BlazeDS выполняется из проекта Flex, для которого указан J2EE в качестве типа сервера приложения.

Дополнительные разделы справки

«Создание проекта Flex для доступа к службам данных» на странице 8

Подключение к службам BlazeDS

Для данной процедуры требуется установленная служба BlazeDS, настроенный сервер удаленной разработки и проект Flex, созданный для доступа к службам BlazeDS.

1 Откройте мастер службы, выбрав в меню данных Flash Builder пункт «Соединить с BlazeDS».

2 Выберите адрес назначения для импорта.

3 (Дополнительно) Измените параметры службы.

Имя службы	Введите имя службы. Flash Builder создает имя для службы на основе адреса назначения. К именам, используемым для службы, применяются ограничения. См. раздел «Именование служб данных» на странице 20
Пакет службы	Укажите имя пакета, в котором содержатся файлы ActionScript для доступа к службе. Flash Builder создает пакет на основе имени службы и помещает его в пакет `services`.
Пакет типов данных	Укажите имя пакета, в котором содержатся созданные файлы класса ActionScript, определяющие типы данных, извлеченных из службы. По умолчанию Flash Builder создает пакет `valueObjects`.

4 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к службе.

Примечание. *После подключения к службе можно изменить свойства этой службы. Выберите службу в представлении «Данные/службы». В контекстном меню выберите «Свойства».*

Доступ к ADEP Data Services

Доступ к службам, предоставляемым ADEP Data Services (прежнее название: LiveCycle Data Services), возможен только в том случае, если установлены службы ADEP Data Services и настроен сервер удаленной разработки (Remote Development Services, RDS). Для получения сведений см. документацию к Adobe ADEP Data Services.

Доступ к службам ADEP Data Services выполняется из проекта Flex, для которого указан J2EE или ColdFusion в качестве типа сервера приложения.

Типы служб для ADEP Data Services

При подключении к ADEP Data Services в качестве адресов назначения доступны следующие типы служб данных:

- Удаленная служба

  Удаленные службы реализуются посредством типизации AMF. Для этих служб не предоставляется управление данными на стороне сервера. Для настройки управления данными на стороне клиента используются инструменты Flash Builder. См. раздел «Включение функции управления данными» на странице 36.

- Служба данных

  Службы данных - это службы, которые реализуют управление данными на стороне сервера. Подробную информацию см. в документации по ADEP Data Services.

- Web-служба

  Web-службы доступны через прокси ADEP Data Services, который настраивается как адрес назначения ADEP Data Services. Как правило, при подключении к web-службе типизация данных на стороне сервера не предоставляется.

Настройка типов данных и управление данными

В Flash Builder включены инструменты для настройки и управления данными на стороне клиента. Доступность инструментов Flash Builder зависит от типа адреса назначения ADEP Data Services:

- Удаленная служба

  Удаленные службы реализуют типизацию AMF для службы. Типы возвращаемых данных не настраиваются для адресов назначения удаленных служб.

  Тем не менее, с помощью Flash Builder можно создать код для управления данными на стороне клиента. См. раздел «Включение функции управления данными» на странице 36.

- Служба данных

  Службы данных реализуют типы данных на стороне сервера. Типы возвращаемых данных не настраиваются для адресов назначения служб данных.

  Для адресов назначения служб данных предоставляется управление данными на стороне сервера. Управление данными на стороне клиента не используется для адресов назначения служб данных.

- Web-служба

  Адреса назначения web-служб, доступные через прокси ADEP Data Service, обычно не реализуют типизацию на стороне сервера. Для настройки типов возвращаемых данных для операций web-служб используются инструменты Flash Builder. См. раздел «Настройка типов данных для операций службы данных» на странице 28.

  С помощью Flash Builder можно создать код для управления данными на стороне клиента. См. раздел «Включение функции управления данными» на странице 36.

Подключение к адресам назначения ADEP Data Service (адреса назначения службы данных и удаленной службы)

Для данной процедуры требуется установленная служба ADEP Data Services, настроенный сервер удаленной разработки и проект Flex, созданный для доступа к службам LCDS.

1 Откройте мастер службы, выбрав в меню данных Flash Builder пункт «Соединить с данными/службой».

2 В диалоговом окне «Выбрать тип службы» выберите LCDS. Нажмите кнопку «Далее».

3 При необходимости введите данные для входа в систему.

4 (Дополнительно) Измените параметры службы.

Имя службы	Имя службы не определяется пользователем. Flash Builder создает имя службы. Flash Builder создает имя для службы на основе адреса назначения.
Пакет службы	Укажите имя пакета, в котором содержатся файлы ActionScript для доступа к службе. Flash Builder создает пакет на основе имени службы и помещает его в пакет `services`.
Адреса назначения	Укажите один или несколько адресов назначения, доступных на сервере ADEP Data Services.
Пакет типов данных	Укажите имя для пакета типов данных. В пакете содержатся созданные файлы класса ActionScript, определяющие типы данных, извлеченных из службы. По умолчанию Flash Builder создает пакет `valueObjects`.

5 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к службе.

Примечание. *После подключения к службе можно изменить свойства этой службы. Выберите службу в представлении «Данные/службы». В контекстном меню выберите «Свойства».*

Подключение к адресам назначения ADEP Data Services (адреса назначения web-службы)

Для данной процедуры требуется установленная служба ADEP Data Services, настроенный сервер удаленной разработки и проект Flex, созданный для доступа к службам DS.

1 Откройте мастер службы, выбрав в меню данных Flash Builder пункт «Соединить с данными/службой».

2 В диалоговом окне «Выбрать тип службы» выберите «Web-служба». Нажмите кнопку «Далее».

3 Выберите пункт «Через адрес назначения прокси LCDS/BlazeDS».

4 При необходимости введите данные для входа в систему.

5 Выберите адрес назначения.

6 (Дополнительно) Измените параметры службы. Нажмите кнопку «Далее».

Имя службы	Введите имя службы. Flash Builder создает имя для службы на основе имени адреса назначения. К именам, используемым для службы, применяются ограничения. См. раздел «Именование служб данных» на странице 20
Пакет службы	Укажите имя пакета, в котором содержатся файлы ActionScript для доступа к службе. Flash Builder создает пакет на основе имени службы и помещает его в пакет `services`.
Пакет типов данных	Укажите имя пакета, в котором содержатся созданные файлы класса ActionScript, определяющие типы данных, извлеченных из службы. По умолчанию Flash Builder создает пакет `dataValues`.

7 (Дополнительно) Настройте создание кода для службы:

Служба **Порт**	Выберите службу и порт из доступных служб и портов.
Список операций	Выберите операции из службы, к которой требуется получить доступ в клиентском приложении.

8 Нажмите кнопку «Готово».

В Flash Builder будут созданы файлы ActionScript для доступа к службе.

Примечание. *После подключения к службе можно изменить свойства этой службы. Выберите службу в представлении «Данные/службы». В контекстном меню выберите «Свойства».*

Дополнительные разделы справки

«Создание проекта Flex для доступа к службам данных» на странице 8

Именование служб данных

В службах данных, к которым выполняется доступ из Flash Builder, применяются ограничения для имен, разрешенных для данной службы. Некоторые из этих ограничений остаются неявными до компиляции приложения.

Рекомендуются следующие правила именования служб:

- Начальный символ имени службы не может быть цифрой.
- Имена служб не могут быть ключевыми словами ActionScript.
- Не используйте имена классов ActionScript, в том числе пользовательских классов, в качестве имен служб.
- (Только для PHP) В Flash Builder не выполняется импорт служб, в имени которых содержатся подчеркивания.

Примечание. *Рекомендуется использовать имена служб, не совпадающие с именами файлов MXML.*

Установка Zend Framework

При первом доступе к службам PHP Flash Builder определяет наличие установки поддерживаемой версии Zend Framework. Если поддерживаемая версия Zend Framework не обнаружена, Flash Builder предлагает пользователю подтвердить установку Zend Framework. Если пользователь подтверждает установку, то Flash Builder устанавливает минимальную версию Zend Framework. При отказе от установки Zend Framework необходимо установить вручную для выполнения доступа к службам PHP.

Установка Flash Builder по умолчанию

Flash Builder устанавливает Zend Framework в папку `ZendFramework` в корневом каталоге web-сервера.

```
<web root>/ZendFramework/
```

Для проектов Flex, выполняющих доступ к службам PHP, Flash Builder создает следующие файлы конфигурации в папке вывода проекта:

- `amf_config.ini`
- `gateway.php`

Рабочие серверы

Adobe рекомендует переместить папку `ZendFramework` из корневого web-каталога при использовании рабочих серверов. Обновите переменную `zend_path`, определенную в `amf_config.ini`.

Если переменная `zend_path` закомментирована, снимите комментарий с этой переменной `zend_path`. Укажите папку установки Zend Framework.

Установка Zend Framework вручную

Инфраструктура Zend Framework может быть установлена вручную.

1 Загрузите последнюю версию Zend Framework.

   При установке можно выбрать минимальный или полный пакет. Flash Builder устанавливает минимальный пакет.

2 Распакуйте загруженную версию в папку на компьютере.

3 В папке проекта Flex с доступом к службам PHP обновите переменную `zend_path`, определенную в `amf_config.ini`.

Если переменная `zend_path` закомментирована, снимите комментарий с этой переменной `zend_path`. Укажите абсолютный путь к папке установки Zend Framework.

Устранение неполадок при установке Zend Framework

Ниже предлагаются рекомендации по устранению неполадок, которые могут возникнуть при подключении к Zend Framework.

Установка Zend Framework вручную

Если Zend Framework установлена вручную, проверьте переменную zend_path в файле `amf_config.ini`.

`amf_config.ini` находится в папке вывода проекта.

Убедитесь, что выполняются следующие условия:

- У `zend_path` отсутствуют комментарии.
- Указан правильный путь к установке Zend Framework:
  - Этот путь является абсолютным путем к адресу назначения в локальной файловой системе. Пользователь не может указать путь к подключенному сетевому ресурсу.
  - Путь указывает на папку библиотеки установки Zend Framework. Как правило, папка библиотеки находится по адресу:

    (Windows) `C:\apache\PHPFrameworks/ZendFramework/library`

    (Mac OS) `/user/apache/PHP/frameworks/ZendFramework/library`

Установка Zend Framework, выполняемая Flash Builder

Если установка Zend Framework выполнялась с помощью Flash Builder, проверьте следующие данные:

- Местоположение папки корневого web-каталога

  Flash Builder устанавливает Zend Framework в папку корневого web-каталога проекта. Проверьте местоположение папки корневого web-каталога. Выберите «Проект» > «Свойства» > «Сервер Flex».
- Убедитесь, что в настройках веб-сервера указано использование PHP.
- Проверьте правильность переменной zend_path в файле `amf_config.ini`.

  `amf_config.ini` находится в папке вывода проекта.

  Убедитесь, что выполняются следующие условия:

  - У `zend_path` отсутствуют комментарии.
  - Указан путь к установке Zend Framework в корневом web-каталоге проекта.
  - Этот путь является абсолютным путем к адресу назначения в локальной файловой системе. Пользователь не может указать путь к подключенному сетевому ресурсу.

Использование одиночного экземпляра сервера

После подключения к службе данных каждое приложение в проекте может иметь доступ к этой службе. По умолчанию каждое приложение создает собственный экземпляр службы при доступе к серверу.

Этот вариант поведения можно изменить таким образом, чтобы в проекте содержался только один экземпляр службы. Каждое приложение в проекте будет выполнять доступ к одному и тому же экземпляру службы. Как правило, одиночный экземпляр службы создается при необходимости координации доступа к данным из нескольких приложений.

Пользователь может выбрать доступ к одиночному экземпляру службы на основе проекта или в качестве параметра для всех проектов.

Доступ к одиночному экземпляру сервера для проекта

1. Выберите «Проект» > «Свойства» > «Данные/службы».
2. Установите флажок для использования одиночного экземпляра сервера. Нажмите кнопку «OK».

Выбор требуемого одиночного экземпляра сервера

1. Откройте диалоговое окно «Параметры».
2. Выберите «Flash Builder» > «Данные/службы».
3. Установите флажок для использования одиночного экземпляра сервера. Нажмите кнопку «OK».

Создание клиентского приложения

Создайте пользовательский интерфейс в редакторе MXML. Редактор может использоваться в режиме «Дизайн» либо в режиме «Код».

После размещения компонентов приложения следует привязать данные, возвращаемые службой, к компонентам пользовательского интерфейса. Создайте обработчики событий, необходимые для взаимодействия пользователя с приложением.

Кроме того, можно создать форму из операций службы в представлении «Данные/службы».

Создание приложения в режиме «Дизайн»

Flex располагает широким ассортиментом контейнеров и элементов управления для создания пользовательского интерфейса. Контейнер обеспечивает иерархическую структуру для упорядочения и размещения элементов пользовательского интерфейса. В контейнерах размещаются другие контейнеры, навигаторы, элементы управления и другие компоненты.

Базовыми элементами управления являются такие компоненты пользовательского интерфейса, как Button, TextArea и CheckBox. Управляемые данными элементы, такие как компоненты DataGrid и List, идеально подходят для отображения данных, извлеченных из службы. Навигаторы представляют собой контейнеры, управляющие перемещением пользователя по нижестоящим контейнерам, таким как набор табуляторов.

В режиме «Дизайн» редактора MXML перетащите контейнеры и элементы управления из представления «Компоненты» в область дизайна. Упорядочьте эти компоненты и настройте их свойства. После разработки макета приложения следует привязать данные, возвращаемые службой данных, к компонентам.

Привязка операций служб к элементам управления

Существует несколько способов привязки операций службы к компоненту пользовательского интерфейса. Операцию службы можно перетащить из представления «Данные/службы» на компонент в области дизайна. Кроме того, в целях выбора операции для привязки к компоненту можно открыть диалоговое окно «Привязать к данным».

Диалоговое окно «Привязать к данным» доступно из панели инструментов представления «Данные/службы». Также это окно можно открыть в режиме «Дизайн» редактора MXML при выборе компонента, принимающего данные, такие как DataGrid. После выбора компонента откройте диалоговое окно «Привязать к данным» из контекстного меню компонента. Помимо этого, диалоговое окно «Привязать к данным» открывается из поля «Поставщик данных» в представлении «Свойства».

При выполнении пользователем привязки операции службы данных к компоненту Flash Builder создает код MXML и ActionScript для доступа к операции службы из клиентского приложения.

Типы возвращаемых данных для операций служб

При привязке операции службы к элементу управления программе Flash Builder необходима информация о типе данных, возвращаемых операцией. Часто пользователи настраивают тип возвращаемых данных для операции службы перед привязкой службы к компоненту.

Если тип возвращаемых данных для операции службы не был предварительно настроен, в диалоговом окне «Привязать к данным» отображается запрос на завершение этого шага.

См. раздел «Настройка типов данных для операций службы данных» на странице 28.

Привязка операции службы к элементу управления DataGrid (путем перетаскивания)

Для этой процедуры необходимо подключение к службе данных.

1 В режиме «Дизайн» редактора MXML перетащите компонент DataGrid из представления «Компоненты» в область редактирования.

2 Перетащите операцию из представления «Данные/службы» на компонент DataGrid.

Если тип возвращаемых данных для операции службы предварительно настроен, Flash Builder привяжет операцию к компоненту DataGrid. Компонент DataGrid изменяется в целях отображения полей, извлеченных из базы данных.

Если тип возвращаемых данных для операции не был предварительно настроен, Flash Builder откроет диалоговое окно «Привязать к данным». См. раздел «Привязка элемента управления DataGrid к операции службы (диалоговое окно «Привязать к данным»)» на странице 24.

3 Настройте отображение DataGrid.

См. раздел Настройка компонентов DataGrid и AdvancedDataGrid.

4 Сохраните и запустите приложение.

Привязка элемента управления DataGrid к операции службы (диалоговое окно «Привязать к данным»)

Для этой процедуры необходимо подключение к службе данных.

1 В режиме «Дизайн» редактора MXML перетащите компонент DataGrid из представления «Компоненты» в область редактирования.

2 После выбора компонента DataGrid откройте диалоговое окно «Привязать к данным» с помощью одного из следующих способов:

- Выберите «Привязать к данным» из меню Flash Builder «Данные», контекстного меню компонента DataGrid или из панели инструментов представления «Данные/службы».
- В представлении «Свойства» нажмите кнопку «Привязать к данным» (расположенную рядом с полем «Поставщик данных»).

3 Выберите «Новый вызов службы», а затем выберите службу и операцию.

Если операция службы уже привязана к компоненту, можно использовать эти результаты. В этом случае выберите «Существующий результат вызова» и укажите операцию для использования.

4 (Необязательно) Выберите «Настроить тип возвращаемых данных»:

Если требуется заново настроить тип возвращаемых данных для операции службы, выберите «Настроить тип возвращаемых данных».

Если тип возвращаемых данных для операции не был предварительно настроен, выберите «Настроить тип возвращаемых данных».

См. раздел «Настройка типов данных для операций службы данных» на странице 28.

5 Нажмите кнопку «OK».

Компонент DataGrid изменяется в целях отображения полей, извлеченных из базы данных.

См. раздел Настройка компонентов DataGrid и AdvancedDataGrid.

6 Сохраните и запустите приложение.

Создание вызова службы для операции

В Flash Builder можно создать метод ActionScript для вызова операции службы. Этот метод не привязан к компоненту пользовательского интерфейса, но доступен для использования в коде пользовательского интерфейса.

Помимо создания метода ActionScript, Flash Builder создает CallResponder, обеспечивающий доступ к данным, возвращенным из вызова службы. См. раздел «CallResponder» на странице 40.

Создание вызова службы для операции

Для этой процедуры необходимо подключение к службе данных.

1 Выберите операцию в представлении «Данные/службы».

2 В контекстном меню для операции выберите пункт «Создать вызов службы».

Flash Builder создает метод для вызова операции и отображает созданный метод в режиме «Код» редактора MXML. Flash Builder создает CallResponder, который содержит результаты вызова службы.

Этот параметр можно также выбрать в панели инструментов «Данные/службы».

Создание формы для приложения

Формы являются одним из самых распространенных методов, используемых в web-приложениях для сбора информации от пользователей. В Flash Builder могут создаваться формы для данных, извлекаемых из вызовов служб, или для пользовательских типов данных, обеспечивающих доступ к удаленным данным.

При создании формы в Flash Builder создается контейнер макета Form и добавляются компоненты для отображения или редактирования определенных данных, извлеченных из службы.

В Flash Builder могут создаваться нижеприведенные типы форм.

Форма	**Описание**
Тип данных	Содержит компоненты, представляющие поля типов данных.
Форма «Основной-подробности»	Форма типа «Основной» обычно является элементом управления данных, в котором перечислены данные, извлеченные из службы. Форма «Подробности» представляет отдельные элементы, выбранные в компоненте типа «Основной».
Вызов службы	Создает две формы. В одной форме указаны входные данные для операции. В другой форме отображаются возвращаемые данные.

При создании формы пользователь указывает поля для включения и тип элемента управления пользовательского интерфейса для представления каждого поля, а также определяет, будет ли форма редактируемой.

Создание формы

Ниже представлена процедура создания формы для вызова службы. Процедуры для создания других типов форм аналогичны данной процедуре. Процедура выполняется в режиме «Дизайн» редактора MXML.

1 Мастер создания форм можно запустить несколькими способами. Выберите операцию в представлении «Данные/службы»:

- В контекстном меню операции выберите пункт «Создать форму».
- В меню Flash Builder «Данные» выберите «Создать форму».
- Перетащите операцию из представления «Данные/службы» на компонент Form в области дизайна.

2 В мастере создания форм выберите «Создать форму для вызова службы».

3 Выберите «Новый вызов службы» или «Существующий результат вызова».

Выберите «Существующий результат вызова», чтобы использовать код, предварительно созданный для вызова службы.

В ином случае выберите «Новый вызов службы» и укажите службу и операцию для формы.

4 (Дополнительно) В зависимости от операции создаваемая форма может иметь различные параметры.

Если операция принимает параметры, то пользователь может включить форму для этих параметров.

Если операция возвращает значение, то пользователь может включить форму для возвращаемого значения.

По желанию пользователя данная форма может быть редактируемой или только для чтения.

5 (Дополнительно) Настройте входные типы или типы возвращаемых данных.

Если выбранная операция использует входные параметры или возвращает значение, то пользователь может настроить входные типы или типы возвращаемых данных.

Входные типы и типы возвращаемых данных для операции необходимо настроить до создания формы. Если эти типы уже были предварительно настроены, пользователь может настроить их снова.

См. раздел «Настройка типов данных для операций службы данных» на странице 28.

6 Нажмите кнопку «Далее». В диалоговом окне «Отображение элемента управления свойства» выберите поля, которые необходимо включить в форму, и тип элемента управления для представления данных.

7 Нажмите кнопку «Готово».

8 Перегруппируйте созданные формы в области дизайна.

При создании форм в Flash Builder формы могут располагаться поверх друг друга. Выберите одну из форм и перетащите ее в требуемое местоположение.

Выполняйте это действие в режиме «Код», чтобы убедиться в том, что выбирается и перемещается сама форма, а не компонент этой формы. Так как в Flash Builder формы располагаются поверх друг друга, выбор необходимой формы может вызвать затруднения. Выберите тег одной из форм в режиме «Код». Эта форма будет выделена в режиме «Дизайн».

Создание формы «Основной-подробности»

Для создания формы «Основной-подробности» необходимо сначала добавить компонент управления данными в приложение и привязать результаты операции к этому элементу управления.

Например, добавьте компонент DataGrid и привяжите результаты операции, например, `getItems_paged()`, к DataGrid.

1 В режиме «Дизайн» выберите элемент управления данными, такой как DataGrid.

2 В меню «Данные» выберите «Создать форму подробной информации».

3 Продолжайте создавать форму, как описано в разделе «Создание формы».

Создание формы для типа данных

Для создания формы с компонентами, представляющими поля пользовательского типа данных, необходимо сначала настроить этот тип данных. См. раздел «Настройка типов данных для операций службы данных» на странице 28.

1 Выберите пользовательский тип данных в представлении «Данные/службы».

2 В контекстном меню выберите «Создать форму».

3 Убедитесь, что выбран параметр «Создание формы для типа данных», и выберите «Тип данных».

4 (Дополнительно) Пользователь может указать, будет ли данная форма редактируемой.

5 Нажмите кнопку «Готово».

Создание обработчиков событий для извлечения удаленных данных

При выполнении пользователем привязки операции службы данных к компоненту Flash Builder создает обработчик событий, получающий данные от службы для заполнения компонента.

Например, если пользователь привязывает операцию `getAllItems()` к компоненту DataGrid, Flash Builder создает обработчик событий `creationComplete`. DataGrid формирует ссылку на созданный обработчик событий. Результаты вызова становятся поставщиком данных для DataGrid.

```
. . .
protected function dataGrid_creationCompleteHandler(event:FlexEvent):void
{
    getAllItemsResult.token = productService.getAllItems();
}
. . .
<mx:DataGrid creationComplete="dataGrid_creationCompleteHandler(event)"
dataProvider="{getAllItemsResult.lastResult}">
. . .
</mx:DataGrid>
. . .
```

При запуске приложения с созданным компонентом DataGrid обработчик событий заполняет DataGrid данными, извлеченными из службы.

При создании обработчиков событий можно применять уже созданные обработчики или заменить их другими по желанию. Например, может потребоваться замена обработчика события `creationComplete` в компоненте DataGrid на обработчик `creationComplete` в приложении.

Также обработчики событий могут быть созданы для элементов управления, принимающих вводимые пользователем данные, например, «Кнопки» или «Текст».

Создание обработчика событий для компонента пользовательского интерфейса

1. Создайте приложение, содержащее компонент пользовательского интерфейса, такой как DataGrid или Button.
2. Перейдите в режим «Дизайн» редактора MXML.

   Выполните одно из следующего:

   - В представлении «Данные/службы» перетащите операцию на компонент пользовательского интерфейса.
   - Выберите компонент пользовательского интерфейса и щелкните по значку создания события в представлении «Свойства». Выберите «Создать обработчик событий».

     Flash Builder создает для данного компонента обработчик, предназначенный для событий по умолчанию. Например, для компонента «Кнопка» этим событием будет событие Click.

     Flash Builder откроет режим представления исходного кода редактора и выделит цветом созданную заглушку для обработчика события.

     Заполните обработчик событий оставшимся кодом. Кодирование обработчика событий выполняется с помощью предусмотренного в Flash Builder мастера содержимого.

Настройка типов данных для операций службы данных

При подключении к службе данных приложению Flash Builder необходима информация о типе данных, возвращаемых операцией службы. Поддерживаемые типы данных – это типы, распознаваемые AMF при обмене данными со службой данных или удаленной службой.

Многие службы данных определяют тип возвращаемых данных на сервере (типизация данных со стороны сервера). Однако если тип данных на сервере не определен, необходимо настроить тип возвращаемых данных в клиентском приложении (типизация данных со стороны клиента).

Операции служб, определяющие параметры, должны также указывать тип, соответствующий данным, к которым осуществляется доступ в службе. С помощью типизации данных со стороны клиента выполняется настройка типа для входных параметров.

При настройке типов для типизации данных со стороны клиента Flash Builder распознает только типы данных AMF. В качестве типа может использоваться пользовательский тип данных, представляющий комплексные данные, или void для указания того, что операция никакие данные не возвращает.

Для операций служб, возвращающих комплексные данные, можно настроить типы, определенные пользователем. Например, если производится извлечение записей из базы данных сотрудников, то тип возвращаемых комплексных данных следует указывать как Employee. В этом случае пользовательский тип данных для Employee должен содержать записи для каждого поля в записи базы данных.

Типы данных для типизации со стороны клиента

Тип данных	**Описание**
Типы ActionScript	Boolean Boolean[] ByteArray ByteArray[] Date Date[] int int[] Number Number[] Object Object[] String String[]
Возврат данных не выполняется	void
Определяемый пользователем тип	*CustomType* *CustomType*[]

Определенный пользователем тип (Employee)

Поле	**Тип данных**
emp_no	Number
first_name	String
last_name	String
hire_date	Date
birth_date	Date

Идентификация доступа к службам

Для доступа к службам данных обычно требуется идентификация пользователя. Для служб PHP, BlazeDS и ColdFusion, обеспечивающих доступ по протоколу HTTP, может потребоваться дополнительная идентификация. В некоторых случаях для этих служб необходима как идентификация HTTP, так и удаленная идентификация.

В Flash Builder включена возможность идентификации службы при выполнении пользователем следующих действий:

- Настройка типов возвращаемых данных для операции

  См. раздел «Настройка типа данных, возвращаемых из операции» на странице 31.

- Использование интерфейса тестирования операции

  См. раздел «Тестирование операций службы» на странице 33.

При выборе параметра «Требуется аутентификация» в Flash Builder открывается диалоговое окно идентификации службы. В зависимости от типа службы, к которой выполняется доступ, пользователь может выбрать параметр «Базовая идентификация» или «Удаленная идентификация».

Базовая идентификация

Базовая идентификация обеспечивает доступ к службе HTTP и web-службе. Для доступа к этим службам необходимо ввести имя пользователя и пароль.

Выберите параметр «Запомнить имя пользователя и пароль», если требуется, чтобы в Flash Builder использовались указанные данные пользователя на протяжении сеанса.

Удаленная идентификация

Удаленная идентификация обеспечивает доступ к службам удаленных объектов. К службам удаленных объектов относятся службы, реализованные как удаленные объекты с использованием ColdFusion, PHP, BlazeDS или ADEP Data Services (прежнее название: LiveCycle Data Services).

Flash Builder не предоставляет интерфейс входа в систему с удаленной идентификацией для проектов, которые не реализуют службы удаленных объектов.

Для доступа к службам удаленных объектов необходимо ввести имя пользователя и пароль.

Выберите параметр «Запомнить имя пользователя и пароль», если требуется, чтобы в Flash Builder использовались указанные данные пользователя на протяжении сеанса.

Настройка входных параметров операции

При типизации данных на стороне клиента пользователь настраивает входные параметры для операций, доступных из службы данных.

Для выполнения следующей процедуры необходимо наличие подключения к службе данных в Flash Builder и операций службы данных, для которых требуются настраиваемые входные параметры.

1. Выберите операцию с настраиваемыми входными параметрами в представлении «Данные/службы». В контекстном меню операции выберите пункт «Настроить входные типы».
2. Выберите тип данных из списка доступных типов для каждого аргумента операции в диалоговом окне «Настроить входные типы». Нажмите кнопку «OK».

Если для службы были предварительно определены пользовательские типы возвращаемых данных, то эти типы данных доступны для выбора.

При типизации на стороне сервера служба определяет тип данных для входных параметров.

Настройка типа данных, возвращаемых из операции

Служба, определяющая типы данных, возвращаемых операцией, обеспечивает типизацию на стороне сервера. Если служба не определяет типы данных, возвращаемых операцией, то Flash Builder использует типизацию на стороне клиента для определения типа возвращаемых данных.

В Flash Builder проводится внутренний анализ данных, возвращаемых из операции службы, для определения типа данных. При настройке типа данных, возвращаемых операцией, можно использовать два параметра:

- «Автоматически определить тип возвращаемых данных из типовых данных»

  Если служба реализует типизацию на стороне сервера, Flash Builder принимает тип данных, определенных службой.

  Если служба не реализует типизацию на стороне сервера, Flash Builder создает пользовательский тип для клиентского приложения. Для типизации данных на стороне клиента необходимо указать имя для пользовательского типа данных. Как правило, это имя описывает возвращаемые данные. Например, если операция возвращает массив книг из таблицы базы данных, именем для типа данных может стать слово «Book».

- «Использовать существующий тип»

  Существующим типом может быть тип, определенный службой, тип ActionScript или предварительно определенный пользовательский тип.

Процедуры, используемые в Flash Builder для внутреннего анализа данных, могут различаться в зависимости от типа службы данных. Например, процедура внутреннего анализа и настройки типа возвращаемых данных для службы HTTP отличается от процедуры, используемой для служб PHP или ColdFusion.

Объединение и изменение типов данных

Во время внутреннего анализа данных сервера можно объединить поля из другого типа данных или создать тип на основе существующего типа данных. Ниже представлены некоторые способы изменения пользовательского типа данных:

- Создайте новое имя для существующего типа данных

  Новое имя требуется в том случае, если планируется различное использование возвращаемых данных в клиентском приложении.

  Например, извлечение информации о сотрудниках, которая может использоваться в сводке сведений о сотрудниках и таблицах подробной информации о сотрудниках в клиентском приложении.

- Объедините поля

  Поля возвращаемых данных можно добавить к существующему типу данных. Добавление дополнительных полей используется при связывании данных из различных источников. Например, для операции JOIN, которая возвращает данные, извлеченные из различных таблиц базы данных.

  Другим примером являются данные, полученные от различных служб. Например, объединение данных Book, полученных от служб HTTP и ColdFusion.

Настройка пользовательского типа данных (службы PHP или ColdFusion)

Для этой процедуры необходимо подключение к службе данных, реализованной с PHP или ColdFusion.

1 Выберите «Настроить тип возвращаемых данных» из контекстного меню операции в представлении «Данные/службы».

2 Если имеются аргументы для операции, введите значения аргументов. Укажите правильный тип данных для аргумента.

3 (Новый или измененный пользовательский тип) Выберите «Автоматически определить тип данных, возвращаемых из этой операции».

   Если для службы требуется аутентификация, выберите «Требуется аутентификация» и введите необходимую информацию. См. раздел «Идентификация доступа к службам» на странице 30.

   Flash Builder выполняет внутренний анализ операции и создает пользовательский тип данных.

   Укажите имя для пользовательского типа данных.

   Если пользовательский тип данных уже был указан, поля возвращаемых данных могут быть добавлены к определению существующего пользовательского типа данных.

4 (Использовать существующий тип) Используйте этот параметр для указания типа ActionScript или предварительно настроенного типа.

5 Нажмите кнопку «Готово».

Настройка пользовательского типа данных (служба HTTP)

Для этой процедуры необходимо подключение к службе HTTP.

1 Выберите «Настроить тип возвращаемых данных» из контекстного меню операции в представлении «Данные/службы».

2 (Новый пользовательский тип) Выберите «Автоматически определить тип данных, возвращаемых из этой операции».

   Если для службы требуется аутентификация, выберите «Требуется аутентификация» и введите необходимую информацию.

   Flash Builder выполняет внутренний анализ операции и создает пользовательский тип данных. Выберите метод, с помощью которого Flash Builder будет передавать значения параметров для операции и нажмите «Далее».

   - (Введите значения параметров) Укажите значение для каждого параметра.

     Также можно указать тип данных для параметра. Flash Builder автоматически выбирает тип данных по умолчанию.

   - (Введите URL-адрес службы) Введите URL-адрес для службы HTML, в том числе параметры и значения в URL-адресе. Например:

     `http://httpserviceaddress/service_operation?param1=94105`

   - (Введите ответ XML/JSON) Скопируйте ответ XML/JSON в текстовое поле.

     Используйте эту возможность при автономной работе или если служба HTTP находится в разработке, но ответ сервера известен.

3 (Новый пользовательский тип, продолжение) Укажите имя для пользовательского типа данных или выберите узел из возвращаемых данных.

При выборе узла для возвращаемых данных Flash Builder создает пользовательский тип для данных, возвращаемых для этого узла.

Укажите, являются ли возвращаемые данные массивом.

Если служба возвращает файл XML, активируется раскрывающийся список выбора корневого каталога. Для определения типа данных выберите узел в файле XML.

4 (Использовать существующий тип) Используйте этот параметр для указания типа ActionScript или предварительно настроенного типа.

5 Нажмите кнопку «Готово».

Тестирование операций службы

С помощью Flash Builder выполняется тестирование операций службы и просмотр данных, возвращенных из операции. Эта возможность используется для проверки поведения служб.

Важная информация. *Некоторые операции, такие как обновление или удаление, изменяют данные на сервере.*

Тестирование операции службы

Для этой процедуры необходимо подключение к службе данных.

1 Выберите операцию службы в представлении «Данные/службы». В контекстном меню выберите «Тестовая операция».

Выбранная операция будет отображена в открывшемся представлении «Тестовая операция». Если для операции требуются входные параметры, то они будут перечислены в представлении «Тестовая операция».

2 Щелкните в поле «Ввести значение» и укажите значение для любого требуемого входного параметра.

Если для параметра необходим комплексный тип, нажмите кнопку с многоточием в поле для открытия редактора входных аргументов. Укажите значение в редакторе.

Редактор входных аргументов принимает нотацию JSON для представления комплексных типов в ActionScript.

3 Если для сервера необходима аутентификация, выберите параметр «Требуется аутентификация». Нажмите кнопку «Тестировать».

Введите требуемые данные аутентификации. См. раздел «Идентификация доступа к службам» на странице 30.

Flash Builder отображает данные, возвращенные из службы.

4 (Дополнительно) В представлении «Тестовая операция» выберите дополнительные службы и операции, доступные для тестирования.

Управление доступом к данным с сервера

Подкачка страниц. С помощью этой функции из удаленной службы поэтапно извлекаются большие наборы данных.

Предположим, что требуется получить доступ к базе данных, содержащей 10 000 записей, а затем вывести на экран данные в DataGrid, состоящем из 20 строк. Для этого можно выполнить операцию подкачки страниц для получения строк с шагом по 20. При запросе пользователем дополнительных данных (при прокручивании DataGrid) будет извлечена и выведена следующая страница записей.

Управление данными. В приложении Flash Builder под управлением данными подразумевается синхронизация обновлений данных на сервере в соответствии с данными клиентского приложения. Использование управления данными позволяет изменять один или несколько элементов в клиентском приложении без выполнения обновлений на сервере. Затем все изменения передаются на сервер посредством одной операции. Кроме того, можно восстановить измененные значения без обновления каких-либо данных.

Управление данными включает в себя координирование нескольких операций («создать», «получить», «обновить», «удалить») для ответа на события из клиентского приложения, например, при обновлении данных сотрудника.

При включении управления данными в Flash Builder также создается код, который автоматически обновляет компоненты пользовательского интерфейса. Например, Flash Builder создает код для обеспечения синхронизации компонентов DataGrid с данными на сервере.

Включение функции подкачки страниц

Пользователь может включить подкачку страниц для службы данных, которая реализует функцию подкачки со следующей подписью:

```
getItems_paged(startIndex:Number, numItems:Number): myDataType
```

Имя функции	Для функции можно использовать любое допустимое имя.
startIndex	Начальная строка извлекаемых данных. Укажите тип данных для startIndex как Number в клиентском приложении.
numItems	Число строк с извлекаемыми данными на каждую страницу. Укажите тип данных для numItems как Number в клиентском приложении.
myDataType	Тип данных, возвращаемых службой данных.

При реализации подкачки страниц из службы можно также реализовать операцию `count()`. Операция `count()` возвращает количество элементов, возвращенных из службы. В Flash Builder требуется, чтобы для операции `count()` была реализована следующая подпись:

```
count(): Number
```

Имя функции	Для функции можно использовать любое допустимое имя.
Number	Количество записей, извлекаемых из операции.

Операция count в приложении Flex обеспечивает правильное отображение компонентов пользовательского интерфейса, используемых для извлечения больших наборов данных. Например, при помощи операции `count()` можно определить размер ползунка для полосы прокрутки компонента DataGrid.

Операция `count()` не предоставляется в некоторых удаленных службах. Подкачку страниц можно по-прежнему использовать, но в этом случае элемент управления, отображающий данные, получаемые при подкачке, не сможет правильно представлять размер набора данных.

Операции подкачки страниц для отфильтрованных запросов

Подкачка страниц может быть включена для запросов, выполняющих фильтрацию результатов из базы данных. При фильтрации результатов в запросе для подкачки страниц и функций count применяются подписи, представленные ниже:

```
getItems_pagedFiltered(filterParam1:String, filterParam2:String, startIndex:Number,
numItems:Number): myDataType

countFiltered(filterParam1:String, filterParam2:String)
```

filterParam1	Дополнительный параметр фильтрации. Этот параметр идентичен в getItems_PagedFiltered() и countFiltered().
filterParam2	Дополнительный параметр фильтрации. Этот параметр идентичен в getItems_PagedFiltered() и countFiltered().

Ниже представлен фрагмент кода функции `getItems_pagedFiltered()`, реализованной в PHP для доступа к базе данных MySQL. Данный фрагмент кода иллюстрирует использование дополнительного параметра фильтрации.

```
get_Items_paged($expression, $startIndex, $numItems) {
    . . .
    SELECT * from employees where name LIKE $expression LIMIT $startIndex, $numItems;
    . . .
}
```

Включение функции подкачки страниц для операции

Для этой процедуры необходимо, чтобы операции `getItems_paged()` и `count()` были включены в код в удаленной службе. Также предполагается, что для данной операции настроен тип возвращаемых данных в соответствии с описанием в разделе «Настройка типов данных для операций службы данных» на странице 28.

1. В представлении «Данные/службы» в контекстном меню операции `getItems_paged()` выберите «Включить подкачку страниц».
2. Если уникальный ключ для типа данных предварительно не определен, укажите атрибуты, однозначно идентифицирующие экземпляр этого типа данных. Нажмите кнопку «Далее».

   Как правило, этим атрибутом является первичный ключ.
3. (Дополнительно) Укажите количество элементов, которое требуется получить для определения пользовательского размера страниц.

   Если пользовательский размер страниц не указан, на уровне службы будет установлен размер страниц по умолчанию. По умолчанию размер страницы установлен на отображение 20 записей на странице.
4. (Дополнительно) Укажите операцию `count()`. Нажмите кнопку «Готово».

   С помощью операции `count()` элементы пользовательского интерфейса, такие как размер ползунка для полосы прокрутки, правильно отображаются в Flash Builder.

   Функция подкачки страниц для этой операции теперь включена.

   В представлении «Данные/службы» подпись функции, реализующей подкачку страниц, больше не включает параметры startIndex и numItems. В Flash Builder эти значения добавляются динамически. Flash Builder определяет эти значения на основе указанного пользовательского размера страницы или размера страницы, заданного по умолчанию для отображения 20 записей на странице.

Включение функции управления данными

Для включения управления данными для службы службе необходимо реализовать одну или несколько функций, перечисленных ниже. Эти функции используются в управлении данными для синхронизации обновлений данных на удаленном сервере:

- «Добавить» (createItem);

  ```
  createItem(item: myDatatype):int
  createItem(item: myDatatype):String
  createItem(item: myDatatype):myDataType
  ```

  Типом возвращаемых данных для createItem() является тип первичного ключа в базе данных.

- «Получить все свойства» (getItem);

  ```
  getItem(itemID:Number): myDatatype
  ```

- «Обновить» (updateItem);

  ```
  updateItem((item: myDataType):void
  updateItem((item: myDataType, originalItem: myDataType):void
  updateItem((item: myDataType, originalItem: myDataType, changes: String[]):void
  ```

- «Удалить» (deleteItem).

  ```
  deleteItem(itemID:Number):void
  ```

В Flash Builder необходимо, чтобы эти функции имели следующие подписи:

Имя функции	Для функции можно использовать любое допустимое имя.
item **originalItem**	Элемент типа данных, возвращаемый службой данных.
itemID	Уникальный идентификатор элемента; как правило, является первичным ключом в базе данных.
changes[]	Массив, соответствующий полям в указанном элементе. Этот аргумент используется только в одной версии `updateItem()`.
myDataType	Тип данных элемента, доступного из службы данных. Как правило, пользовательский тип данных определяется при извлечении данных из службы.

Флажок autoCommit

Посредством управления данными выполняется синхронизация обновлений данных на сервере. Изменения данных в клиентском приложении не обновляются на сервере, пока не вызван метод `service.commit()`.

Если требуется отключить эту функцию, установите для флажка autoCommit значение true. Если autoCommit имеет значение true, то обновления данных сервера не кэшируются, а выполняются немедленно. См. раздел «Включение функции управления данными для службы» на странице 42.

Флажок deleteItemOnRemoveFromFill

Флажок deleteItemOnRemoveFromFill используется при удалении элементов с включенной функцией управления данными. По умолчанию для этого флажка установлено значение true. При удалении элемента этот элемент немедленно удаляется и из базы данных.

Установите для флажка deleteItemOnRemoveFromFill значение false, чтобы задержать удаление до вызова метода commit(). В примере показан код для обработчика событий creation complete компонента DataGrid. Если пользователь удаляет выбранный элемент Employee в компоненте DataGrid, то этот выбранный элемент не удаляется из базы данных до вызова метода commit().

```
protected function dg_creationCompleteHandler(event:FlexEvent):void
{
    employeeService.     getDataManager(employeeService.DATA_MANAGER_EMPLOYEE).autoCommit=false;
    employeeService.getDataManager(  e m p l
oyeeService.DATA_MANAGER_EMPLOYEE).deleteItemOnRemoveFromFill=      true;
    getAllEmployeesResult.token = employeeService.getAllEmployees();
}
```

Включение функции управления данными для операции

Эта процедура предполагает, что требуемые операции уже реализованы в удаленной службе. Кроме того, предполагается, что настроен тип возвращаемых данных для операций, использующих пользовательский тип данных. См. раздел «Настройка типов данных для операций службы данных» на странице 28.

1 В представлении «Данные/службы» разверните узел «Типы данных».

2 Из контекстного меню для типа данных выберите «Активировать управление данными».

3 Если уникальный ключ для типа данных предварительно не определен, укажите атрибуты, однозначно идентифицирующие экземпляр этого типа данных. Нажмите кнопку «Далее».

   Как правило, этим атрибутом является первичный ключ.

4 Укажите реализованные операции «Добавить», «Получить все свойства», «Обновить» и «Удалить». Нажмите кнопку «Готово».

   Примечание. *Реализация всех этих функций не требуется. Выполните реализацию только тех функций, которые необходимы для приложения.*

Функция управления данными для этой операции теперь включена.

Создание кода Flash Builder для клиентских приложений

Flash Builder создает код клиента, обеспечивающий доступ к операциям удаленных служб. Flash Builder создает код при:

- подключении к службе данных;
- обновлении службы данных в представлении «Данные/службы»;
- настройке типа возвращаемых данных для операции;
- привязке операции службы к элементу управления пользовательского интерфейса;
- включении функции подкачки страниц для операции служб;
- включении функции управления данными для операции;
- создании обработчика событий или вызова службы.

Классы служб

Для подключения к службам данных используется мастер служб. При подключении к службе Flash Builder создает файл класса ActionScript, обеспечивающий доступ к операциям служб.

Для служб с доступом к RemoteObject созданный класс расширяет класс RemoteObjectServiceWrapper. Службы, реализованные с PHP, ColdFusion, BlazeDS и ADEP Data Services, обычно имеют доступ к RemoteObject.

Для служб HTTP созданный класс расширяет класс HTTPServiceWrapper.

Для web-служб созданный класс расширяет класс WebServiceWrapper.

Flash Builder определяет имя созданного файла класса на основе имени, указанного для службы в мастере служб. В Flash Builder этот класс по умолчанию помещается в основной исходной папке проекта. Как правило, это папка `src`. Имя пакета создается на основе имени службы. Например, Flash Builder создает следующие классы ActionScript для класса EmployeeService.

```
- project
    |
    - src
        |
        + (default package)
        |
        + services
        |   |
        |   +employeeservice
        |       |
        |       + _Super_EmployeeService.as
        |       |
        |       + EmployeeService.as
```

Суперкласс содержит реализацию для класса EmployeeService.

Суперкласс, который является созданным классом, не следует редактировать. Вносимые в суперкласс изменения будут перезаписаны. Все вносимые в реализацию изменения могут привести к неопределенному поведению.

Например, `EmployeeService.as` используется для расширения созданного суперкласса и добавления пользовательской реализации.

Дополнительные разделы справки

«Подключение к службам данных» на странице 9

Классы для пользовательских типов данных

Многие удаленные службы данных обеспечивают типизацию данных на стороне сервера. В качестве пользовательских типов данных эти службы возвращают комплексные данные.

Для служб, не возвращающих типизированные данные, Flash Builder обеспечивает типизацию данных на стороне клиента. С помощью типизации данных на стороне клиента мастер подключения Flash Builder используется для определения и настройки комплексных данных, возвращаемых службой. Например, для службы, возвращающей записи из базы данных сотрудников, необходимо определить и настроить тип данных Employee.

Flash Builder создает класс ActionScript для реализации каждого пользовательского типа данных, возвращаемого службой. Flash Builder использует этот класс для создания объектов значений, которые в дальнейшем используются для доступа к данным удаленной службы.

Так, Flash Builder создает следующие классы ActionScript для класса EmployeeService, в котором содержится тип данных Employee:

```
- project
    |
    - src
        |
        + (default package)
        |
        + services
        |   |
        |   +employeeservice
        |       |
        |       + _Super_EmployeeService.as
        |       |
        |       + EmployeeService.as
        |
        + valueObjects
            |
            + _Super_Employee.as
            |
            + Employee.as
```

В суперклассах содержится реализация EmployeeService и типа данных Employee соответственно.

Созданный суперкласс не следует редактировать. Вносимые в суперкласс изменения будут перезаписаны. Все вносимые в реализацию изменения могут привести к неопределенному поведению.

В этом примере `EmployeeService.as` и `Employee.as` расширяют созданный суперкласс и добавляют реализацию.

привязке операции службы к элементу управления пользовательского интерфейса;

Раздел «Привязка операций служб к элементам управления на странице 24» иллюстрирует способ привязки данных, возвращаемых из операций служб к элементу управления пользовательского интерфейса. При привязке операции службы к элементу управления в Flash Builder создается следующий код:

- тег Declarations, содержащий тег службы и CallResponder;
- обработчик событий для вызова службы;
- привязка элемента управления к данным, возвращаемым из операции.

Тег Declarations

Тег Declarations является элементом MXML, объявляющим невизуальные свойства, отличные от свойств по умолчанию текущего класса. При привязке операции службы к пользовательскому интерфейсу Flash Builder создает тег Declarations, содержащий тег службы и CallResponder. CallResponder и созданный класс службы являются свойствами элемента контейнера, как правило, являющегося тегом Application.

В примере представлен тег Declarations, обеспечивающий доступ к удаленной службе EmployeeService:

```
<fx:Declarations>
    <s:CallResponder id="getAllEmployeesResult"/>
    <employeesservice:EmployeesService id="employeesService"
        fault="Alert.show(event.fault.faultString + '\n'
        + event.fault.faultDetail)" showBusyCursor="true"/>
</fx:Declarations>
```

CallResponder

CallResponder управляет результатами для вызовов служб. Он содержит свойство маркера, установленное на маркер Async, возвращаемый вызовом службы. Кроме того, CallResponder содержит свойство lastResult, установленное на последний успешный результат из вызова службы. Обработчики событий добавляются к CallResponder для обеспечения доступа к данным, возвращаемым свойством lastResult.

При создании CallResponder в Flash Builder также создается свойство id на базе имени операции службы, к которой оно привязано. В следующем примере кода представлены свойства CallResponder для двух операций EmployeeService. Операция `getAllItems()` привязана к обработчику событий creationComplete для DataGrid. Операция delete привязана к выбранной позиции DataGrid. Сразу после создания в DataGrid отображаются элементы, полученные из вызова службы `getAllItems()`. Элемент управления Button «Удалить элемент» удаляет в DataGrid выбранную запись из базы данных.

```
<fx:Script>
    <![CDATA[
        import mx.events.FlexEvent;
        import mx.controls.Alert;

        protected function dg_creationCompleteHandler(event:FlexEvent):void
        {
            getAllItemsResult.token = employeesService.getAllItems();
        }

        protected function button_clickHandler(event:MouseEvent):void
        {
            deleteItemResult.token =
                employeesService.deleteItem(dg.selectedItem.emp_no);
        }
    ]]>
</fx:Script>

<fx:Declarations>
    <s:CallResponder id="getAllItemsResult"/>
    <employeesservice:EmployeesService id="employeesService"
        fault="Alert.show(event.fault.faultString + '\n'
        + event.fault.faultDetail)" showBusyCursor="true"/>
    <s:CallResponder id="deleteItemResult"/>
</fx:Declarations>
<mx:DataGrid id="dg" editable="true"

creationComplete="dg_creationCompleteHandler(event)"dataProvider="{getAllItemsResult.lastRes
ult}">
    <mx:columns>
        <mx:DataGridColumn headerText="emp_no" dataField="emp_no"/>
        <mx:DataGridColumn headerText="last_name" dataField="last_name"/>
        <mx:DataGridColumn headerText="hire_date" dataField="hire_date"/>
    </mx:columns>
</mx:DataGrid>
<s:Button label="Delete Item" id="button" click="button_clickHandler(event)"/>
```

Обработчики событий

При привязке операции службы к компоненту пользовательского интерфейса Flash Builder создает обработчик событий для CallResponder. Обработчик события управляет результатами операции. Кроме того, можно создать обработчик события в блоке кода ActionScript, а также ссылку на этот обработчик события из свойства компонента пользовательского интерфейса.

Обычно элементы управления, такие как List и DataGrid, заполняются данными, возвращаемыми из службы. По умолчанию Flash Builder создает обработчик событий creationComplete для элемента управления, который запускается сразу после создания элемента управления. Для других элементов управления Flash Builder создает обработчик для события элемента управления по умолчанию. Например, для элемента управления Button Flash Builder создает событие для события click элемента управления Button.

Свойство события элемента управления установлено на созданный обработчик событий. В примере показан созданный обработчик событий creation complete для DataGrid:

```
<fx:Script>
    <![CDATA[
        import mx.events.FlexEvent;
        import mx.controls.Alert;

        protected function dg_creationCompleteHandler(event:FlexEvent):void
        {
            getAllItemsResult.token = employeesService.getAllItems();
        }
    ]]>
</fx:Script>
. . .

<mx:DataGrid id="dg" editable="true"
        creationComplete="dg_creationCompleteHandler(event)"
        dataProvider="{getAllItemsResult.lastResult}">
    <mx:columns>
        <mx:DataGridColumn headerText="emp_no" dataField="emp_no"/>
        <mx:DataGridColumn headerText="last_name" dataField="last_name"/>
        <mx:DataGridColumn headerText="hire_date" dataField="hire_date"/>
    </mx:columns>
</mx:DataGrid>
```

Обработчики событий можно создать для элементов управления, отвечающих на события пользователя, например, Button. В примере показан созданный обработчик событий для Button, заполняющий DataGrid:

```
<fx:Script>
    <![CDATA[
        import mx.events.FlexEvent;
        import mx.controls.Alert;

        protected function button_clickHandler(event:MouseEvent):void
        {
            deleteItemResult.token =
                employeesService.deleteItem(dg.selectedItem.emp_no);
        }
    ]]>
</fx:Script>
. . .

<s:Button label="Delete Item" id="button" click="button_clickHandler(event)"/>
```

Привязка данных

При создании пользовательского интерфейса операции службы привязываются к элементам управления. См. раздел «Привязка операций служб к элементам управления» на странице 24.

Flash Builder создает код, привязывающий данные, которые возвращаются из операции службы, к элементу пользовательского интерфейса, в котором отображаются данные.

В примере ниже представлен код, создаваемый Flash Builder для заполнения элемента управления DataGrid. Операция `getAllItems()` возвращает набор записей сотрудников для пользовательского типа данных Employee.

Свойство dataProvider для DataGrid привязано к результатам, хранящимся в CallResponder, getAllItemsResult. Flash Builder выполняет автоматическое обновление DataGrid с помощью DataGridColumns в соответствии с каждым полем, возвращаемым для записи Employee. Свойства headerText и dataField для каждого столбца установлены согласно данным, возвращаемым в записи Employee.

```
<mx:DataGrid creationComplete="datagrid1_creationCompleteHandler(event)"
    dataProvider="{getAllItemsResult.lastResult}" editable="true">
        <mx:columns>
            <mx:DataGridColumn headerText="gender" dataField="gender"/>
            <mx:DataGridColumn headerText="emp_no" dataField="emp_no"/>
            <mx:DataGridColumn headerText="birth_date" dataField="birth_date"/>
            <mx:DataGridColumn headerText="last_name" dataField="last_name"/>
            <mx:DataGridColumn headerText="hire_date" dataField="hire_date"/>
            <mx:DataGridColumn headerText="first_name" dataField="first_name"/>
        </mx:columns>
    </mx:DataGrid>
```

Включение функции подкачки страниц для операции служб

При активации подкачки страниц Flash Builder вносит изменения в реализацию созданной службы. При заполнении элемента управления данных (например, DataGrid или List) с помощью данных с подкачкой страниц Flash Builder определяет число записей, видимых в элементе управления, а также общее число записей в базе данных. Flash Builder предоставляет эти значения в виде аргументов для операции службы, используемых для реализации функции подкачки страниц.

После включения подкачки страниц не требуется изменять код клиентского приложения.

Для получения дополнительной информации см. раздел «Включение функции подкачки страниц» на странице 34.

Включение функции управления данными для службы

В Flash Builder функция управления данными является синхронизацией набора обновлений данных на сервере. Пользователь может активировать управление данными для пользовательских типов данных, возвращаемых из службы. Посредством активации функции управления данными можно изменить один или более элементов в клиентском приложении без необходимости обновления сервера. Затем все изменения передаются на сервер посредством одной операции. Кроме того, можно восстановить измененные значения без обновления каких-либо данных на сервере. Раздел «Включение функции управления данными» на странице 36 иллюстрирует способ реализации этой функции.

При активации управления данными Flash Builder вносит изменения в реализацию созданного класса службы и созданного класса для пользовательских типов данных. Flash Builder создает DataManager для реализации этой функции.

Синхронизация обновлений для данных сервера

При вызове операций служб для функции управления типом данных изменения отражаются в клиентском приложении. Однако данные на сервере не обновляются, пока не вызван метод `commit()` DataManager.

Если для службы включено управление данными, то служба имеет флажок autoCommit. Значением по умолчанию для autoCommit является false.

Флажок autoCommit определяет, будут ли изменения внесены немедленно или после вызова службы *service*.commit().

Если autoCommit имеет значение false, то все обновления для службы в клиентском приложении кэшируются до вызова *service*.commit(). Для отмены изменений можно вызвать метод revertChanges() службы.

Если autoCommit имеет значение true, то обновления отправляются на сервер немедленно. При этом метод revertChanges() нельзя вызвать для отмены изменений.

Флажок deleteItemOnRemoveFromFill определяет, будет ли удаленный элемент немедленно удален из базы данных. При значении true элемент не удаляется до вызова *service*.commit().

Приведенный ниже код отключает синхронизацию управления данными для обновлений данных сервера. Изменения в данных управляемого типа немедленно передаются на сервер.

```
bookService.getDataManager(bookService.DATA_MANAGER__BOOK).autoCommit = true;
```

В следующем коде представлено включение синхронизации управления данными для обновлений данных на сервере. Изменения данных для управляемого типа не обновляются, пока для службы не будет вызван commit(). Также удаленные элементы не удаляются из базы данных до вызова commit().

```
bookService.getDataManager(bookService.DATA_MANAGER__BOOK).autoCommit = false;
bookService.getDataManager(bookService.DATA_MANAGER__BOOK).deleteItemOnRemoveFromFill= true;
```

Отмена изменений

DataManager включает метод revertChanges(). Метод revertChanges() восстанавливает данные, отображаемые в клиентском приложении для значений, полученных с сервера до последнего вызова метода commit.

Вызовите revertChanges() перед вызовом commit(), чтобы отменить изменения для управляемого типа данных в клиентском приложении:

```
bookService.getDataManager (bookService.DATA_MANAGER_BOOK).revertChanges();
```

Для внесения изменений в управляемый тип данных вызовите метод commit().

```
bookService.getDataManager (employeeService.DATA_MANAGER_EMPLOYEE).commit();
```

Кроме того, метод commit() может быть вызван напрямую из экземпляра bookService. Посредством вызова метода commit напрямую из экземпляра службы вносятся все изменения для типов управляемых данных.

```
bookService.commit();
```

Примечание. *Метод revertChanges() нельзя вызвать напрямую из экземпляра службы, чтобы отменить изменения для всех управляемых типов данных. Этот метод может вызываться только для специфического управляемого типа данных.*

Если требуется изменить заданный по умолчанию вариант поведения для управления данными и отключить возможность отмены изменений, установите для autoCommit значение true. Например, при наличии экземпляра bookService и включенном управлении данными для типа данных Book установите autoCommit на true:

```
bookService.getDataManager(bookService.DATA_MANAGER__BOOK).autoCommit = true;
```

При этом изменения в данных управляемого типа немедленно передаются на сервер.

Развертывание приложений, обеспечивающих доступ к службам данных

При переводе приложения из среды разработки в среду развертывания следует учитывать ряд факторов. Процесс развертывания приложения зависит от самого приложения, требований к данному приложению и используемой среды развертывания.

Например, процесс развертывания приложения на внутреннем web-сайте, который доступен только для сотрудников компании, будет отличаться от процесса развертывания того же приложения на общественном web-сайте.

В разделе Deploying applications представлен обзор факторов, которые необходимо принимать во внимание, и контрольный список развертывания. В контрольном списке перечислены наиболее распространенные проблемы конфигурации системы, с которыми сталкиваются пользователи при развертывании приложений для производства. В документацию также включены советы по устранению неполадок и проведению диагностики распространенных проблем развертывания.

Рекомендации по кодированию доступа к службам

С помощью инструментов Flash Builder можно создавать клиентские коды для доступа к данным в базе данных. Эта функция может использоваться как для PHP, так и для ColdFusion. Однако данный код используется только для работы с прототипами. Не следует использовать данный код в качестве шаблона для создания безопасных приложений.

По умолчанию установлено, что посредством этого кода любой пользователь с сетевым доступом к вашему серверу получит возможность вставлять, выбирать, обновлять или удалять информацию из таблицы базы данных. Ниже представлены рекомендации по изменению созданного кода или созданию любого кода, обеспечивающего доступ к службам. Для получения дополнительной информации см. документ Securing Data Services.

Удаление неиспользуемых функций

Удалите или обозначьте комментариями любые функции, для которых не планируется использование в данном приложении.

Добавление аутентификации

Добавьте аутентификацию пользователя для обеспечения доступа к информации в базе данных только для надежных пользователей.

Добавление проверок авторизации

При необходимости аутентификации добавьте проверки авторизации. Даже если пользователи авторизованы для данного приложения, иногда необходимо убедиться, что у них есть права на выполнение определенных запросов.

Например, функция выбора доступна для любого пользователя, но для функции удаления может потребоваться ограничение прав пользователей.

Другой пример: пользователь А авторизован для извлечения собственной информации с помощью запроса select. Но пользователь А не может использовать запрос select для доступа к информации пользователя Б.

Проверка данных

Убедитесь, что проверка данных добавлена. Например, проверка данных, передаваемых в любую инструкцию insert, необходима, чтобы убедиться в том, что база данных не принимает никаких поврежденных или зловредных данных.

Проверка на стороне клиента не способна защитить от действий злоумышленника, вручную отправляющего запросы на web-сервер. Проверка данных защищает против действий компьютерных пиратов и обеспечивает качество сохраняемой информации.

Ограничение количества извлекаемых данных

С помощью методов select можно выбирать любые данные в таблице. В некоторых случаях это может привести к слишком большому объему информации, передаваемой по сети. Следует извлекать только необходимые данные.

Например, `SELECT *` из таблицы пользователей может возвратить имя пользователя и пароль по сети.

Использование протокола SSL для конфиденциальных данных

Использование безопасного протокола обеспечивает конфиденциальность отправляемой информации.

Экспорт исходных файлов с версией выпуска приложения

При экспорте сборки выпуска приложения в Flash Builder предоставляется функция «Отобразить код». С помощью этой функции пользователи могут просмотреть исходные файлы, реализующие приложение. В проектах сервера исходные файлы включают папку `services`, в которой содержатся файлы, обеспечивающие доступ к реализации пользовательской службы.

Важная информация. *При включении файлов службы с помощью функции «Отобразить код» следует соблюдать осторожность. Файлы службы могут содержать данные доступа к пользовательской базе данных и такие конфиденциальные сведения, как имена пользователей и пароль. Если службы включены в функцию «Отобразить код», любой пользователь с доступом к запущенному приложению может просмотреть эти конфиденциальные сведения.*

Дополнительные разделы справки

Flex Security

Создание безопасных служб

В документации Adobe представлены наглядные примеры, в том числе руководства и приложения, спроектированные посредством создания кода в программе Flash Builder. Они иллюстрируют процесс доступа к службам данных из клиентского приложения. Но поскольку основной целью примеров является ясность изложения, в них не учитываются рекомендации по обеспечению безопасного доступа к данным.

В документации Flash Builder содержатся примеры, в том числе приложения, спроектированные из созданного кода. Эти примеры необходимо развернуть в доверенной среде разработки. Такая доверенная среда разработки может находиться на локальном компьютере или в местоположении внутри брандмауэра. При отсутствии дополнительных мер безопасности любой пользователь с сетевым подключением может получить доступ к пользовательской базе данных.

Ниже представлены рекомендации по созданию служб:

- Проведите идентификацию пользователя до вызова любого метода для службы.
- Используйте аутентификацию службы, которая позволяет только некоторым пользователям выполнять определенные действия.

  Предположим, что некое приложение разрешает изменение информации о сотрудниках посредством вызова RemoteObject. В этом случае необходима аутентификация RemoteObject с той целью, чтобы только менеджеры получили разрешение на изменение информации о сотрудниках.
- Используйте программные средства безопасности для ограничения доступа к службам.
- Применяйте декларативные ограничения безопасности ко всем службам.
- При выполнении доступа к web-службе (`<mx:WebService>`) или службе HTTP (`<mx:HTTPService>`) одно из следующих условий должно быть верным:
  - Служба реализуется в том же домене, что и приложение, которое ее вызывает.
  - В системе хоста для службы находится файл `crossdomain.xml`, который явным образом разрешает доступ из домена приложения.

Дополнительные разделы справки

Flex Security

Securing Data Services

Writing secure applications

В Adobe® Flash® Player выполняются приложения, созданные в Flash. Содержимое передается в Flash Player в виде последовательных инструкций в двоичном формате по web-протоколам в точно описанном формате файла SWF. Файлы SWF, как правило, находятся на сервере и затем загружаются при необходимости на компьютер клиента для отображения. Основная часть содержимого состоит из двоичных инструкций ActionScript. ActionScript, используемый в Flash, является языком сценариев на основе стандартов ECMA. В ActionScript включены интерфейсы API, предназначенные для создания элементов пользовательского интерфейса на стороне клиента, управления этими элементами и работы с данными.

Модель безопасности Flex предоставляет защиту как для клиента, так и для сервера. Ниже представлены два основных аспекта безопасности:

- авторизация и аутентификация пользователей, использующих доступ к ресурсам сервера;
- функционирование Flash Player в изолированной среде безопасности для клиента.

Flex поддерживает работу с безопасностью web-приложений любого сервера приложений J2EE. Кроме того, приложения, предварительно скомпилированные в Flex, могут использоваться совместно со схемой аутентификации и авторизации, созданной по любой базовой серверной технологии, для предотвращения доступа пользователей к приложениям. Инфраструктура Flex также включает несколько встроенных механизмов безопасности, позволяющих управлять доступом к web-службам, службам HTTP и ресурсам на основе сервера, таким как EJB.

Flash Player выполняется в изолированной среде обеспечения безопасности, что предотвращает атаки злоумышленников против клиента путем внедрения кода.

Примечание. *К содержимому SWF, выполняющемуся в Adobe® AIR®, применяются другие правила безопасности, чем к выполнению содержимого в браузере. Для получения подробной информации см. тему «Безопасность Air» в документации по AIR.*

Ссылки на различные темы по вопросу безопасности см. на странице Security Topic Center сайта Adobe Developer Connection.

Дополнительные разделы справки

Flex Security

Глава 3. Реализация служб для ориентированных на данные приложений

Формат Action Message Format (AMF)

В Flex используются службы удаленных объектов и AMF для выполнения доступа к службам, реализация которых выполняется в ColdFusion, PHP, BlazeDS и ADEP Data Services AMF обеспечивает передачу сообщений для обмена данными между базой данных и клиентским приложением.

ColdFusion, BlazeDS и ADEP Data Services обеспечивают инфраструктуру AMF для служб удаленных объектов. Flash Builder использует инфраструктуру Zend AMF для служб, реализуемых в PHP.

Службы ColdFusion и PHP обеспечивают типизацию данных на стороне сервера. При типизации на стороне сервера служба определяет тип возвращаемых данных. Однако если реализация службы не может определить тип возвращаемых данных, Flash Builder обеспечивает типизацию данных на стороне клиента. Flash Builder производит выборку данных из службы, позволяющую настройку типа возвращаемых данных в клиентском приложении.

Типизация данных на стороне клиента и на стороне сервера

В Flex клиентское приложение использует тип данных, возвращенных из вызова службы, в методах, которые выполняют доступ к данным и отображают эти данные.

Однако в примерах ниже службы возвращают нетипизированные данные.

- «Реализация служб ColdFusion» на странице 49
- «Реализация служб PHP» на странице 55
- «Пример реализации служб из нескольких источников» на странице 70

Например, для операции `getAllEmployees()` служба возвращает массив нетипизированных объектов, представляющих записи в базе данных. Flash Builder обеспечивает инструменты, включающие типизацию данных на стороне клиента. С помощью инструментов Flash Builder выполняется внутренний анализ возвращаемых данных и определение пользовательских типов для данных.

Для возвращаемого объекта данных сотрудников определяется пользовательский тип данных Employee. Каждый столбец становится свойством типа данных Employee.

С помощью типа данных Employee клиентское приложение может выполнять доступ к возвращаемым данным и правильно отображать их в клиентском приложении.

С помощью Flash Builder также выполняется доступ к службам, реализующим типизацию на стороне сервера. См. примеры типизации данных на стороне сервера в документе Flash Builder server-side type examples.

Реализация служб ColdFusion

В ColdFusion службы реализуются как файлы компонентов ColdFusion (CFC). Все файлы CFC содержат функции для обеспечения операций службы.

Службы ColdFusion можно создать в любой среде IDE, например в Adobe ColdFusion® Builder™. Flash Builder не предоставляет редактор, оптимизированный для изменения файлов ColdFusion. Однако при открытии файла ColdFusion в Flash Builder выполняется запуск приложения в системе, связанной с файлами ColdFusion.

Пример служб ColdFusion

Базовая служба ColdFusion реализуется путем создания компонента ColdFusion (CFC), содержащего функции для операций служб. В примере `employeeService.cfc` показана служба EmployeeService, реализующая две функции. Функция `getAllIEmployees()` извлекает все записи о сотрудниках в базе данных. Функция `getEmployees()` возвращает одну запись о сотруднике на основе параметра `emp_no` функции.

В примере представлена типизация на стороне клиента. Служба возвращает нетипизированные данные. В Flash Builder типизация на стороне клиента используется для проведения внутреннего анализа возвращаемых данных и определения типа данных.

В следующих примерах показана реализация служб для подкачки страниц и управления данными.

С помощью Flash Builder также выполняется доступ к службам, реализующим типизацию на стороне сервера. См. раздел «Типизация данных на стороне клиента и на стороне сервера» на странице 48.

Примеры, иллюстрирующие типизацию на стороне сервера, были недоступны на этапе окончания работы над этим документом. См. примеры типизации данных на стороне сервера в документе Flash Builder server-side type examples.

Пример ColdFusion, реализующий базовую службу

В данном примере показана реализация базовой службы в ColdFusion. В примере используется код, созданный Flash Builder при доступе к таблице базы данных. См. раздел «Создание типовой службы ColdFusion из таблицы базы данных» на странице 10.

В примере представлена типизация на стороне клиента. См. раздел «Типизация данных на стороне клиента и на стороне сервера» на странице 48.

См. примеры типизации данных на стороне сервера в документе Flash Builder server-side type examples.

Важная информация. *Типовые службы предназначены только для работы с прототипами. Используйте типовую службу только в доверенной среде разработки. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. Дополнительную информацию о создании безопасных служб ColdFusion см. в документации по ColdFusion About User Security.*

```
<cfcomponent output="false">

<!---
    This sample service contains functions that illustrate typical service operations.
    This code is for prototyping only.

    Authenticate the user prior to allowing them to call these methods. You can find more
    information at http://www.adobe.com/go/cf9_usersecurity

--->
    <cffunction name="getAllemployees" output="false" access="remote" returntype="any" >

        <!--- Retrieve set of records and return them as a query or array.
                Add authorization or any logical checks for secure access to your data --->

        <cfset var qAllItems="">
        <cfquery name="qAllItems" datasource="employees">
            SELECT * FROM employees
        </cfquery>
        <cfreturn qAllItems>

    </cffunction>

    <cffunction name="getemployees" output="false" access="remote"  returntype="any" >
        <cfargument name="emp_no" type = "numeric" required="true" />

        <!--- Retrieve a single record and return it as a query or array.
                Add authorization or any logical checks for secure access to your data --->

        <cfset var qItem="">
        <cfquery name="qItem" datasource="employees">
            SELECT *
            FROM employees
            WHERE emp_no = <CFQUERYPARAM CFSQLTYPE="CF_SQL_INTEGER"
VALUE="#ARGUMENTS.emp_no#">
        </cfquery>

        <cfreturn qItem>

    </cffunction>

</cfcomponent>
```

Основные аспекты EmployeeService:

- Подключение к базе данных сотрудников и доступ к таблице сотрудников в базе данных
- Возврат массива объектов

  При программировании с использованием инфраструктуры Flex возвращаются только данные. Клиентское приложение выполняет обработку форматирования и представления данных. Эта модель отличается от традиционных приложений ColdFusion CFM, возвращающих данные, отформатированные в шаблоне HTML.

  Flex обрабатывает возвращаемые наборы записей как массив объектов. В каждой строке содержится полученная из базы данных запись. Каждый столбец записи в базе данных становится свойством возвращаемого объекта. Клиентское приложение теперь может выполнить доступ к возвращаемым данным как к объектам с набором свойств.

Настройте тип данных для возвращаемого объекта. См. раздел «Типизация данных на стороне клиента и на стороне сервера» на странице 48.

- Обработка ошибок на сервере ColdFusion

  Обработка ошибок, проводимая ColdFusion, используется при отладке службы. В ColdFusion Administrator измените настройки параметров Debugging и Logging для отображения подробной информации об отладке.

  В интерфейсе тестовой операции Flash Builder отображается информация, возвращаемая сервером ColdFusion.

  Дополнительную информацию о службах тестирования см. в разделе «Отладка удаленных служб» на странице 67.

- Использование `cfqueryparam` для создания запросов базы данных

  `cfqueryparam` представляет собой защиту против инструкций по внедрению кода SQL при обращениях к серверу. Для получения дополнительной информации см. документ Enhancing security with cfqueryparam в документации по ColdFusion.

- Идентификация пользователей до предоставления им доступа к функциям в этой службе

  В типовом коде не показаны способы идентификации пользователей. См. документацию по ColdFusion About User Security.

Дополнительные разделы справки

«Настройка типов данных для операций службы данных» на странице 28

«Доступ к службам ColdFusion» на странице 10

«Создание типовой службы ColdFusion из таблицы базы данных» на странице 10

Пример ColdFusion, реализующий подкачку страниц

С помощью Flash Builder возможна реализация подкачки страниц данных, полученных из удаленной службы. При подкачке страниц выполняется инкрементное извлечение больших наборов данных.

Для реализации функции подкачки страниц для Flash Builder требуются специальные подписи функции. В следующем примере кода показан способ реализации службы ColdFusion для данных с подкачкой страниц.

В примере EmployeeServicePaged используется код, созданный Flash Builder при доступе к таблице базы данных. См. раздел «Создание типовой службы ColdFusion из таблицы базы данных» на странице 10.

Важная информация. *Типовые службы предназначены только для работы с прототипами. Используйте типовую службу только в доверенной среде разработки. Посредством данного примера любой пользователь с сетевым доступом к вашему серверу получит возможность вызывать, изменять или удалять информацию из таблицы базы данных. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. Дополнительную информацию о создании безопасных служб ColdFusion см. в документации по ColdFusion About User Security.*

```
<cfcomponent output="false">

<!---
    This sample service contains functions that illustrate typical service operations.
    This code is for prototyping only.

    Authenticate the user prior to allowing them to call these methods. You can find more
    information at http://www.adobe.com/go/cf9_usersecurity
--->
    <cffunction name="count" output="false" access="remote" returntype="numeric" >

        <!---   Return the number of items in your table.
                Add authorization or any logical checks for secure access to your data --->
        <cfquery name="qread" datasource="employees">
            SELECT COUNT(emp_no) AS itemCount FROM employees
        </cfquery>

        <cfreturn qread.itemCount>

    </cffunction>

    <cffunction name="getemployees_paged" output="false" access="remote" returntype="any" >
        <cfargument name="startIndex" type="numeric" required="true" />
        <cfargument name="numItems" type="numeric" required="true" />

    <!---Return a page of numRows number of records as an array or
        query from the database for this startRow.
        Add authorization or any logical checks for secure access to your data --->
    <!---The LIMIT keyword is valid for mysql database only.
        Modify it for your database --->

        <cfset var qRead="">
        <cfquery name="qRead" datasource="employees">
            SELECT * FROM employees LIMIT #startIndex#, #numItems#
        </cfquery>
        <cfreturn qRead>

    </cffunction>
</cfcomponent>
```

Служба EmployeeServicePaged возвращает нетипизированные данные. Настройка типа возвращаемых данных для `getEmployees_Paged()` выполняется с помощью инструментов Flash Builder. После настройки типа возвращаемых данных необходимо включить подкачку страниц для операции `getEmployees_Paged()`.

Дополнительные разделы справки

«Пример служб ColdFusion» на странице 49

«Настройка типов данных для операций службы данных» на странице 28

«Управление доступом к данным с сервера» на странице 33

Пример ColdFusion, реализующий операции управления данными

С помощью инструментов Flash Builder возможна реализация управления данными для удаленных служб. Под управлением данными подразумевается синхронизация обновлений данных из клиентского приложения на сервере.

Для Flash Builder необходима комбинация специальных подписей функций для реализации функции управления данными. В следующем примере кода показан способ реализации службы ColdFusion для управления данными.

В примере EmployeeServiceDM используется код, созданный Flash Builder при доступе к таблице базы данных. См. раздел «Создание типовой службы ColdFusion из таблицы базы данных» на странице 10.

Важная информация. *Типовые службы предназначены только для работы с прототипами. Используйте типовую службу только в доверенной среде разработки. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. Дополнительную информацию о создании безопасных служб ColdFusion см. в документации по ColdFusion About User Security.*

```
<cfcomponent output="false">

<!---
    This sample service contains functions that illustrate typical service operations.
    This code is for prototyping only.

    Authenticate the user prior to allowing them to call these methods. You can find more
    information at http://www.adobe.com/go/cf9_usersecurity
--->
    <cffunction name="getAllemployees" output="false" access="remote" returntype="any" >

        <!--- Auto-generated method
                Retrieve set of records and return them as a query or array.
                Add authorization or any logical checks for secure access to your data --->

        <cfset var qAllItems="">
        <cfquery name="qAllItems" datasource="employees">
            SELECT * FROM employees
        </cfquery>
        <cfreturn qAllItems>

    </cffunction>

    <cffunction name="getemployees" output="false" access="remote"  returntype="any" >
        <cfargument name="emp_no" type = "numeric" required="true" />

        <!---
                Retrieve a single record and return it as a query or array.
                Add authorization or any logical checks for secure access to your data --->

        <cfset var qItem="">
        <cfquery name="qItem" datasource="employees">
            SELECT *
            FROM employees
            WHERE emp_no = <CFQUERYPARAM CFSQLTYPE="CF_SQL_INTEGER"
VALUE="#ARGUMENTS.emp_no#">
        </cfquery>

        <cfreturn qItem>

    </cffunction>

    <cffunction name="createemployees" output="false" access="remote" returntype="any" >
        <cfargument name="item" required="true" />
```

```
        <!---   Insert a new record in the database.
            Add authorization or any logical checks for secure access to your data --->

        <cfquery name="createItem" datasource="employees" result="result">
            INSERT INTO employees (birth_date, first_name, last_name, gender, hire_date)
            VALUES (<CFQUERYPARAM cfsqltype="CF_SQL_DATE" VALUE="#item.birth_date#">,
                            <CFQUERYPARAM cfsqltype="CF_SQL_VARCHAR"
VALUE="#item.first_name#">,
                            <CFQUERYPARAM cfsqltype="CF_SQL_VARCHAR"
VALUE="#item.last_name#">,
                            <CFQUERYPARAM cfsqltype="CF_SQL_CHAR" VALUE="#item.gender#">,
                            <CFQUERYPARAM cfsqltype="CF_SQL_DATE" VALUE="#item.hire_date#">)

        </cfquery>

        <!--- The GENERATED_KEY is valid for mysql database only, you can modify it for your
database --->
        <cfreturn result.GENERATED_KEY/>

    </cffunction>

    <cffunction name="updateemployees" output="false" access="remote" returntype="void" >
        <cfargument name="item" required="true" />

        <!--- Update an existing record in the database.
              Add authorization or any logical checks for secure access to your data --->

         <cfquery name="updateItem" datasource="employees">
            UPDATE employees SET birth_date = <CFQUERYPARAM cfsqltype=CF_SQL_DATE
VALUE="#item.birth_date#">,
                             first_name = <CFQUERYPARAM cfsqltype=CF_SQL_VARCHAR
VALUE="#item.first_name#">,
                             last_name = <CFQUERYPARAM cfsqltype=CF_SQL_VARCHAR
VALUE="#item.last_name#">,
                             gender = <CFQUERYPARAM cfsqltype=CF_SQL_CHAR
VALUE="#item.gender#">,
                             hire_date = <CFQUERYPARAM cfsqltype=CF_SQL_DATE
VALUE="#item.hire_date#">
```

```
            WHERE emp_no = <CFQUERYPARAM CFSQLTYPE="CF_SQL_INTEGER" VALUE="#item.emp_no#">
        </cfquery>

    </cffunction>

    <cffunction name="deleteemployees" output="false" access="remote" returntype="void" >
        <cfargument name="emp_no" type="numeric" required="true" />

        <!--- Delete a record in the database.
              Add authorization or any logical checks for secure access to your data --->

        <cfquery name="delete" datasource="employees">
            DELETE FROM employees
            WHERE emp_no = <CFQUERYPARAM CFSQLTYPE="CF_SQL_INTEGER"
VALUE="#ARGUMENTS.emp_no#">
        </cfquery>

    </cffunction>
</cfcomponent>
```

Служба EmployeeServiceDM возвращает нетипизированные данные. Для настройки типа возвращаемых данных для `getAllEmployeess()` и `getEmployees()` необходимо использовать инструменты Flash Builder. Для пользовательского типа данных, возвращаемых этими операциями, используется тип Employee.

После настройки типа возвращаемых данных необходимо включить управление данными в типе данных Employee.

Дополнительные разделы справки

«Пример служб ColdFusion» на странице 49

«Настройка типов данных для операций службы данных» на странице 28

«Управление доступом к данным с сервера» на странице 33

Создание CFC с помощью Adobe ColdFusion Builder

В Adobe® ColdFusion® Builder™ включен Adobe CFC Generator. CFC Generator используется для создания ORM CFC или стандартного CFC из набора таблиц базы данных. CFC, созданные с помощью ColdFusion Builder, можно затем использовать в качестве службы данных в Flash Builder. В Adobe CFC Generator создаются службы, реализующие типизацию на стороне сервера.

См. подробные сведения в документе Using Adobe CFC Generator.

Примечание. *При реляционном сопоставлении объектов ColdFusion (object relational mapping, ORM) используется модель объекта для определения стратегии сопоставления при хранении и извлечении данных из реляционной базы данных. См. документ ColdFusion ORM.*

Реализация служб PHP

Обычно службы PHP реализуются как классы PHP. Классы PHP не обязательно должны быть ориентированы на объект. Каждый класс может представлять собой библиотеку функций, обеспечивающих операции служб.

Службы PHP можно создать в любой среде редактирования, например, в Dreamweaver или Zend Studio. Flash Builder не обеспечивает редактор, оптимизированный для изменения файлов PHP. Однако при открытии файла PHP в Flash Builder выполняется запуск приложения в системе, связанной с файлами PHP. Для удобства в Flash Builder включен редактор простого текста, который можно использовать для редактирования файлов PHP.

Использование AMF для доступа к службам, реализованным в PHP

Доступ к службам данных PHP осуществляется с использованием формата инициирующих сообщений (Action Message Format, AMF). AMF обеспечивает передачу сообщений между клиентом Flash и web-сервером. В Flash Builder используется инфраструктура Zend AMF для обеспечения передачи сообщений AMF в службах данных PHP.

Для получения информации по Zend AMF см. Руководство разработчика Zend Framework.

Для получения информации по установке Zend Framework см. раздел «Установка Zend Framework» на странице 21.

Для получения информации об использовании Zend с Flash Builder for PHP посетите веб-сайт Zend.

Примечание. *В Flash Builder используется инфраструктура Zend AMF, но использование компонентов Zend при создании служб PHP не обязательно. Хотя компоненты Zend могут применяться с Flash Builder, для создания служб можно также использовать любую среду разработки PHP.*

Пример служб PHP

Основная служба PHP реализуется путем создания файла класса PHP, содержащего функции для операций служб. В примере показана служба EmployeeService, реализующая две функции:

- `getAllIEmployees()`

  Извлекает записи всех сотрудников из базы данных.

- `getEmployeeByID($itemID)`

  Возвращает запись об одном сотруднике.

В примере представлена типизация на стороне клиента. Служба возвращает нетипизированные данные. В Flash Builder типизация на стороне клиента используется для проведения внутреннего анализа возвращаемых данных и определения типа данных.

В следующих примерах показана реализация служб для подкачки страниц и управления данными.

С помощью Flash Builder также выполняется доступ к службам, реализующим типизацию на стороне сервера. См. раздел «Типизация данных на стороне клиента и на стороне сервера» на странице 48.

Примеры, иллюстрирующие типизацию на стороне сервера, были недоступны на этапе окончания работы над этим документом. См. примеры типизации данных на стороне сервера в документе Flash Builder server-side type examples.

Пример базовой службы PHP

В данном примере показана реализация базовой службы в PHP. В примере используется код, созданный Flash Builder при доступе к таблице базы данных. См. раздел «Создание типовой службы PHP из таблицы базы данных» на странице 12.

В примере представлена типизация на стороне клиента. См. раздел «Типизация данных на стороне клиента и на стороне сервера» на странице 48.

Важная информация. *Типовые службы предназначены только для работы с прототипами. Используйте типовую службу только в доверенной среде разработки. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. Для получения информации о создании безопасных служб см. раздел «Развертывание приложений, обеспечивающих доступ к службам данных» на странице 44.*

```
<?php
/**
 * This sample service contains functions that illustrate typical service operations.
 * This code is for prototyping only.
 *
 *  Authenticate users before allowing them to call these methods.
 */
class EmployeeService {

    var $username = "root";
    var $password = "root";
    var $server = "localhost";
    var $port = "3306";
    var $databasename = "employees";
    var $tablename = "employees";

    var $connection;

    /**
     * The constructor initializes the connection to database. Everytime a request is
     * received by Zend AMF, an instance of the service class is created and then the
     * requested method is called.
     */
    public function __construct() {
        $this->connection = mysqli_connect(
                                $this->server,
                                $this->username,
                                $this->password,
                                $this->databasename,
                                $this->port
                            );

        $this->throwExceptionOnError($this->connection);
    }

    /**
     * Returns all the rows from the table.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     * @return array
     */
    public function getAllEmployees() {

        $stmt = mysqli_prepare($this->connection, "SELECT * FROM $this->tablename");
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();
```

```
    $rows = array();

    mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
        $row->first_name, $row->last_name, $row->gender, $row->hire_date);

    while (mysqli_stmt_fetch($stmt)) {
      $rows[] = $row;
      $row = new stdClass();
      mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
            $row->first_name, $row->last_name, $row->gender, $row->hire_date);
    }

    mysqli_stmt_free_result($stmt);
    mysqli_close($this->connection);

    return $rows;
}

/**
 * Returns the item corresponding to the value specified for the primary key.
 *
 * Add authroization or any logical checks for secure access to your data
 *
 *
 * @return stdClass
 */
public function getEmployeesByID($itemID) {

    $stmt = mysqli_prepare($this->connection, "SELECT * FROM $this->tablename where emp_no=?");
    $this->throwExceptionOnError();

    mysqli_bind_param($stmt, 'i', $itemID);
    $this->throwExceptionOnError();

    mysqli_stmt_execute($stmt);
    $this->throwExceptionOnError();

    mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
            $row->first_name, $row->last_name, $row->gender, $row->hire_date);

    if(mysqli_stmt_fetch($stmt)) {
        return $row;
    } else {
        return null;
```

```
        }
    }

    /**
     * Utitity function to throw an exception if an error occurs
     * while running a mysql command.
     */
    private function throwExceptionOnError($link = null) {
        if($link == null) {
            $link = $this->connection;
        }
        if(mysqli_error($link)) {
            $msg = mysqli_errno($link) . ": " . mysqli_error($link);
            throw new Exception('MySQL Error - '. $msg);
        }
    }
}


?>
```

Основные аспекты EmployeeService:

- установка связи с базой данных сотрудников с доступом к порту 3306 на локальном хосте; доступ к таблице сотрудников в базе данных;
- предоставление переменных класса для подключения к службе и доступа к таблицам в базе данных;

  Эти переменные используются в функциях класса.

  Измените значения этих переменных на значения для текущей системы.
- возврат массива объектов для клиентского приложения;

  При программировании с использованием инфраструктуры Flex возвращаются только данные. Клиентское приложение выполняет обработку форматирования и представления данных.

  Эта модель отличается от традиционных служб PHP, возвращающих данные, отформатированные в шаблоне HTML.
- функция `getEmployeesByID($itemID)` привязывает входной параметр к типам данных;

  Количество переменных и типы длины строк должны соответствовать параметрам в инструкции. Квантификатор «?» в инструкции prepare является заполнителем для параметра.

  mysqli распознает следующие типы:

  - integer (`i`)
  - double (`d`)
  - string (`s`)
  - blob (`b`)
- привязка результатов с созданием массива объектов (`$row[]`);

  Flex обрабатывает наборы записей как массив объектов. Каждый объект представляет полученную из базы данных запись. Каждый столбец записи в базе данных становится свойством возвращаемого объекта. Клиентское приложение теперь может выполнить доступ к возвращаемым данным как к объектам с набором свойств.

Поскольку сервер не определяет тип возвращаемых данных, необходимо настроить тип данных для возвращаемого объекта. См. раздел «Типизация данных на стороне клиента и на стороне сервера» на странице 48.

- предоставление функции конструктора для выполнения подключения к базе данных;
- использование инструкции mysqli prepare для создания запросов базы данных;

  Использование инструкций prepare представляет собой защиту против инструкций по внедрению кода SQL при обращениях к серверу. Инструкция выполняется на сервере только после подготовки.
- идентификация пользователей до предоставления им доступа к функциям в этой службе;

  В типовом коде не показаны способы идентификации пользователей. См. документацию по ColdFusion About User Security. Принципы безопасности при идентификации и авторизации пользователя, изложенные в документации ColdFusion, применимы и к службам PHP.
- вызов исключения при ошибке;

  Информация, содержащаяся в исключениях, используется при отладке реализации службы. В интерфейсе тестовой операции Flash Builder отображается информация, возвращаемая исключениями.

  Дополнительную информацию о службах тестирования см. в разделе «Отладка удаленных служб» на странице 67.
- имя файла `EmployeeService.php` соответствует имени класса PHP для службы.

  При несовпадении имен файла и класса при доступе к службе возникают ошибки.

Дополнительные разделы справки

«Настройка типов данных для операций службы данных» на странице 28

«Доступ к службам PHP» на странице 11

«Создание типовой службы PHP из таблицы базы данных» на странице 12

Пример PHP, реализующий подкачку страниц

С помощью Flash Builder возможна реализация подкачки страниц данных, полученных из удаленной службы. При подкачке страниц выполняется инкрементное извлечение больших наборов данных.

Для реализации функции подкачки страниц для Flash Builder требуются специальные подписи функции. В следующем примере кода показан способ реализации службы PHP для данных с подкачкой страниц.

В примере используется код, созданный Flash Builder при доступе к таблице базы данных. См. раздел «Создание типовой службы PHP из таблицы базы данных» на странице 12.

Важная информация. *Типовые службы предназначены только для работы с прототипами. Используйте типовую службу только в доверенной среде разработки. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. Для получения информации о создании безопасных служб см. раздел «Развертывание приложений, обеспечивающих доступ к службам данных» на странице 44.*

```
<?php

/**
 *  This sample service contains functions that illustrate typical service operations.
 *  This code is for prototyping only.
 *
 *  Authenticate the user prior to allowing them to call these methods.
 *
 */
class EmployeeServicePaged {

    var $username = "root";
    var $password = "root";
    var $server = "localhost";
    var $port = "3306";
    var $databasename = "employees";
    var $tablename = "employees";

    var $connection;

    /**
     * The constructor initializes the connection to database. Everytime a request is
     * received by Zend AMF, an instance of the service class is created and then the
     * requested method is invoked.
     */
    public function __construct() {
        $this->connection = mysqli_connect(
                                $this->server,
                                $this->username,
                                $this->password,
                                $this->databasename,
                                $this->port
                            );

        $this->throwExceptionOnError($this->connection);
    }


    /**
     * Returns the number of rows in the table.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     *
     */
    public function count() {
        $stmt = mysqli_prepare($this->connection, "SELECT COUNT(*) AS COUNT
                                                    FROM $this->tablename");
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        mysqli_stmt_bind_result($stmt, $rec_count);
        $this->throwExceptionOnError();

        mysqli_stmt_fetch($stmt);
```

```
        $this->throwExceptionOnError();

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);

        return $rec_count;
    }


    /**
     * Returns $numItems rows starting from the $startIndex row from the
     * table.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     * @return array
     */
    public function getEmployees_paged($startIndex, $numItems) {

        $stmt = mysqli_prepare($this->connection, "SELECT * FROM
                                    $this->tablename LIMIT ?, ?");
        $this->throwExceptionOnError();

        mysqli_bind_param($stmt, 'ii', $startIndex, $numItems);
        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        $rows = array();

        mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
                                $row->first_name, $row->last_name,
                                $row->gender, $row->hire_date);

        while (mysqli_stmt_fetch($stmt)) {
          $rows[] = $row;
          $row = new stdClass();
          mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
                                    $row->first_name, $row->last_name,
                                    $row->gender, $row->hire_date);
        }

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);
```

```
        return $rows;
    }


    /**
     * Utitity function to throw an exception if an error occurs
     * while running a mysql command.
     */
    private function throwExceptionOnError($link = null) {
        if($link == null) {
            $link = $this->connection;
        }
        if(mysqli_error($link)) {
            $msg = mysqli_errno($link) . ": " . mysqli_error($link);
            throw new Exception('MySQL Error - '. $msg);
        }
    }
}
?>
```

Служба EmployeeServicePaged возвращает нетипизированные данные. Настройка типа возвращаемых данных для `getEmployees_Paged()` выполняется с помощью инструментов Flash Builder. После настройки типа возвращаемых данных необходимо включить подкачку страниц для операции `getEmployees_Paged()`.

Дополнительные разделы справки

«Пример служб PHP» на странице 56

«Настройка типов данных для операций службы данных» на странице 28

«Управление доступом к данным с сервера» на странице 33

Пример PHP, реализующий управление данными

С помощью инструментов Flash Builder возможна реализация управления данными для удаленных служб. Под управлением данными подразумевается синхронизация обновлений данных из клиентского приложения на сервере.

Для Flash Builder необходима комбинация специальных подписей функций для реализации функции управления данными. В следующем примере кода показан способ реализации службы PHP для управления данными.

В примере используется код, созданный Flash Builder при доступе к таблице базы данных. См. раздел «Создание типовой службы PHP из таблицы базы данных» на странице 12.

Важная информация. *Типовые службы предназначены только для работы с прототипами. Используйте типовую службу только в доверенной среде разработки. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. Для получения информации о создании безопасных служб см. раздел «Развертывание приложений, обеспечивающих доступ к службам данных» на странице 44.*

```
<?php

/**
 *  This sample service contains functions that illustrate typical service operations.
 *  This code is for prototyping only.
 *
 *  Authenticate the user prior to allowing them to call these methods.
 */
class EmployeeServiceDM {

    var $username = "root";
    var $password = "root";
    var $server = "localhost";
    var $port = "3306";
    var $databasename = "employees";
    var $tablename = "employees";

    var $connection;

    /**
     * The constructor initializes the connection to database. Everytime a request is
     * received by Zend AMF, an instance of the service class is created and then the
     * requested method is invoked.
     */
    public function __construct() {
        $this->connection = mysqli_connect(
                                $this->server,
                                $this->username,
                                $this->password,
                                $this->databasename,
                                $this->port
                            );

        $this->throwExceptionOnError($this->connection);
    }

    /**
     * Returns all the rows from the table.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     * @return array
     */
    public function getAllEmployees() {

        $stmt = mysqli_prepare($this->connection, "SELECT * FROM $this->tablename");
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        $rows = array();

        mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
                                $row->first_name, $row->last_name,
                                $row->gender, $row->hire_date);
```

```
        while (mysqli_stmt_fetch($stmt)) {
          $rows[] = $row;
          $row = new stdClass();
          mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
                                  $row->first_name, $row->last_name,
                                  $row->gender, $row->hire_date);
        }

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);

        return $rows;
    }

    /**
     * Returns the item corresponding to the value specified for the primary key.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     *
     * @return stdClass
     */
    public function getEmployeesByID($itemID) {

        $stmt = mysqli_prepare($this->connection, "SELECT * FROM
                                        $this->tablename where emp_no=?");
        $this->throwExceptionOnError();

        mysqli_bind_param($stmt, 'i', $itemID);
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
                                $row->first_name, $row->last_name,
                                $row->gender, $row->hire_date);

        if(mysqli_stmt_fetch($stmt)) {
            return $row;
        } else {
            return null;
        }
    }

    /**
     * Returns the item corresponding to the value specified for the primary key.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     *
     * @return stdClass
     */
    public function createEmployees($item) {

        $stmt = mysqli_prepare($this->connection, "INSERT INTO $this->tablename
                                (emp_no, birth_date, first_name, last_name,
```

```
                            gender, hire_date) VALUES (?, ?, ?, ?, ?, ?)");
        $this->throwExceptionOnError();

        mysqli_bind_param($stmt, 'isssss', $item->emp_no, $item->birth_date
                                  $item->first_name, $item->last_name,
                                  $item->gender, $item->hire_date);
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        $autoid = mysqli_stmt_insert_id($stmt);

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);

        return $autoid;
    }

    /**
     * Updates the passed item in the table.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     * @param stdClass $item
     * @return void
     */
    public function updateEmployees($item) {

        $stmt = mysqli_prepare($this->connection, "UPDATE $this->tablename
                               SET emp_no=?, birth_date=?, first_name=?,
                               last_name=?, gender=?, hire_date=?
                               WHERE emp_no=?");
        $this->throwExceptionOnError();

        mysqli_bind_param($stmt, 'isssssi', $item->emp_no, $item->birth_date,
                           $item->first_name, $item->last_name, $item->gender,
                           $item->hire_date, $item->emp_no);
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);
    }

    /**
     * Deletes the item corresponding to the passed primary key value from
     * the table.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     *
     * @return void
     */
    public function deleteEmployees($itemID) {
```

```
        $stmt = mysqli_prepare($this->connection, "DELETE FROM $this->tablename
                                                    WHERE emp_no = ?");
        $this->throwExceptionOnError();

        mysqli_bind_param($stmt, 'i', $itemID);
        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);
    }

    /**
     * Utitity function to throw an exception if an error occurs
     * while running a mysql command.
     */
    private function throwExceptionOnError($link = null) {
        if($link == null) {
            $link = $this->connection;
        }
        if(mysqli_error($link)) {
            $msg = mysqli_errno($link) . ": " . mysqli_error($link);
            throw new Exception('MySQL Error - '. $msg);
        }
    }
}
?>
```

Служба EmployeeServiceDM возвращает нетипизированные данные. Для настройки типа возвращаемых данных для `getAllEmployeess()` и `getEmployeesByID()` необходимо использовать инструменты Flash Builder. Для пользовательского типа данных, возвращаемых этими операциями, используется тип Employee.

После настройки типа возвращаемых данных необходимо включить управление данными в типе данных Employee.

Дополнительные разделы справки

«Пример служб PHP» на странице 56

«Настройка типов данных для операций службы данных» на странице 28

«Управление доступом к данным с сервера» на странице 33

Отладка удаленных служб

Существует несколько способов отладки приложений, обращающихся к удаленным службам:

- представление тестовой операции Flash Builder;

  Представление тестовой операции Flash Builder используется для вызова операций службы и просмотра возвращаемых данных. В представлении тестовой операции отображаются любые сообщения об ошибках, предоставляемые службой.

- сценарии, выполняемые на стороне сервера;

Для дополнительной отладки служб можно создать сценарии, тестирующие код сервера и записывающие информацию выходного потока в файлы журнала.

- монитор сети Flash Builder.

  Монитор сети используется в Flash Builder после компоновки приложения, которое выполняет доступ к службе. С помощью монитора сети можно просматривать данные, передаваемые между сервером и клиентом.

представление тестовой операции Flash Builder;

Представление тестовой операции Flash Builder используется для вызова операций из службы и просмотра результатов данной операции. Результаты включают любые сообщения об ошибках, возвращенные из службы.

Представление тестовой операции используется для просмотра данных, возвращенных из операций с записываемыми службами или службами, доступными из HTTP или web-служб.

Тестирование операции службы

Для этой процедуры необходимо наличие созданной тестируемой службы или доступ к службе HTTP или web-службе.

1. В представлении Flash Builder «Данные/службы» перейдите к операции службы, которую требуется протестировать.
2. В контекстном меню для операции службы выберите «Тестовая операция».
3. (Дополнительно) В представлении «Тестовая операция» выберите параметр «Требуется аутентификация», чтобы предоставить службе данные для входа в систему.
4. Если операция принимает параметры, щелкните в поле «Ввести значение» и укажите значение для параметра.

   Если для параметра необходим комплексный тип, нажмите кнопку с многоточием в поле «Ввести значение» для открытия редактора, принимающего нотацию JSON. Введите значение параметра с помощью нотации JSON.
5. Для просмотра результатов операции нажмите кнопку «Тестировать».

Сценарии для тестирования кода сервера

Перед попыткой связи с сервером в Flash Builder необходимо использовать сценарии для просмотра и отладки кода сервера. Тестовые сценарии обеспечивают следующие преимущества:

- Результаты тестирования можно просматривать в web-браузере.

  После внесения изменений в код обновите страницу браузера для просмотра результатов.
- Можно выполнить повтор или печать результатов в выходной поток, что невозможно осуществить непосредственно из AMF.
- Отображения ошибок тщательно отформатированы и, как правило, являются более полными, чем ошибки, собранные с использованием AMF.

Сценарии ColdFusion

Для осуществления дампа вызова функции используйте следующий сценарий `tester.cfm`.

```
<!--- tester.cfm --->
<cfobject component="EmployeeService" name="o"/>
<cfdump var="#o.getAllItems()#">
```

В `tester2.cfm` указывается метод и аргументы для выполнения вызова по URL-адресу.

```
<!--- tester2.cfm --->
<cfdump var="#url#">

<cfinvoke component="#url.cfc#" method="#url.method#" argumentCollection="#url#"
returnVariable="r">

<p>Result:

<cfif isDefined("r")>
    <cfdump var="#r#">
<cfelse>
    (no result)
</cfif>
```

Например, вызовите метод `getItemID()` в EmployeeService со следующим URL-адресом:

```
http://localhost/tester2.cfm?EmployeeService&method=getItemId&id=12
```

`tester3.cfm` ведет журнал, в котором регистрируются вызовы операций и куда выводятся входные аргументы с использованием `cfdump`.

```
<!--- tester3.cfm --->
<cfsavecontent variable="d"><cfdump var="#arguments#"></cfsavecontent>

<cffile action="append"
file="#getDirectoryFromPath(getCurrentTemplatePath())#MyServiceLog.htm"
output="<p>#now()# operationName #d#">
```

Сценарии PHP

Для осуществления дампа вызова функции используйте следующий сценарий `tester.php`.

```
<pre>
<?php
include('MyService.php');
    $o = new MyService();
    var_dump($o->getAllItems());
?>
</pre>
```

Для регистрации сообщений в журнале в ходе выполнения кода добавьте к службе PHP следующий код:

```
$message = 'updateItem: '.$item["id"];
$log_file = '/Users/me/Desktop/myservice.log';
error_log(date('d/m/Y H:i:s').' '.$message.PHP_EOL, 3, $log_file);
```

Для включения функции дампа данных в файл журнала добавьте к службе PHP следующий код:

```
ob_start();
var_dump($item);
$result = ob_get_contents();
ob_end_clean();

$message = 'updateItem: '.$result;
$log_file = '/Users/me/Desktop/myservice.log';
error_log(date('d/m/Y H:i:s').' '.$message.PHP_EOL, 3, $log_file);
```

Монитор сети

Монитор сети доступен в среде Flash Builder в проекции «Отладка Flex». Для мониторинга данных необходимо включить монитор. Для получения информации о включении и использовании монитора сети см. раздел Мониторинг приложений, обеспечивающих доступ к службам данных.

Пример реализации служб из нескольких источников

Как правило, приложения выполняют доступ к данным из различных источников, при этом в приложении отображаются результаты связывания данных. В примере показано связывание данных из следующих трех таблиц в базе данных сотрудников:

- Departments

  Каждая запись содержит такие поля, как номер отдела и наименование отдела.

- Dept_emp

  Каждая запись содержит такие поля, как emp_no, dept_no, from_date, to_date.

- Employees

  Каждая запись содержит такие поля, как emp_no, birth_date, first_name, last_name, gender, hire_date.

В типовом приложении содержатся два компонента DataGrid: один для Departments и один для Employees.

В Departments перечислены все отделы. После выбора отдела в Employees DataGrid будут перечислены все сотрудники данного отдела.

При выборе сотрудника в Employees DataGrid заполняется форма, с помощью которой можно обновить записи о сотруднике.

Создание служб

Для этого примера требуется создать одну службу. Служба содержит следующие операции:

- getAllDepartments()
- getEmployeesByDept()
- `getEmployeeByID()`
- `updateEmployee()`.

EmployeeService (PHP)

`EmployeeService.php` реализует службу, содержащую одну функцию. `GetEmployeesByID()` принимает идентификатор отдела в качестве аргумента и возвращает список всех сотрудников данного отдела. Эта функция также возвращает даты начала и окончания работы сотрудника в отделе. `GetEmployeesByDept()` выполняет следующий запрос SQL:

```
SELECT
    employees.emp_no,
    employees.birth_date,
    employees.first_name,
    employees.last_name,
    employees.gender,
    employees.hire_date,
    dept_emp.from_date,
    dept_emp.to_date
FROM employees, dept_emp
WHERE dept_emp.emp_no = employees.emp_no and
    dept_emp.dept_no = departments.dept_no
```

Важная информация. *Типовые службы предназначены только для работы с прототипами. Используйте типовую службу только в доверенной среде разработки. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. Для получения информации о создании безопасных служб см. раздел «Развертывание приложений, обеспечивающих доступ к службам данных» на странице 44.*

```
<?php

/**
 * EmployeeService.php
 *
 * This sample service contains functions that illustrate typical service operations.
 * Use these functions as a starting point for creating your own service implementation.
 *
 *  This code is for prototyping only.
 *
 *  Authenticate the user before allowing them to call these methods.
 */
class EmployeeService {

    var $username = "admin2";
    var $password = "Cosmo49";
    var $server = "localhost";
    var $port = "3306";
    var $databasename = "employees";
    var $tablename = "employees";

    var $connection;

    /**
     * The constructor initializes the connection to database. Everytime a request is
     * received by Zend AMF, an instance of the service class is created and then the
     * requested method is called.
     */
    public function __construct() {
        $this->connection = mysqli_connect(
                                $this->server,
```

```
                                $this->username,
                                $this->password,
                                $this->databasename,
                                $this->port
                            );

        $this->throwExceptionOnError($this->connection);
    }


    /**
     * Returns all the rows from the table.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     * @return array
     */
    public function getAllDepartments() {

        $stmt = mysqli_prepare($this->connection, "SELECT * FROM departments");
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        $rows = array();

        mysqli_stmt_bind_result($stmt, $row->dept_no, $row->dept_name);

        while (mysqli_stmt_fetch($stmt)) {
          $rows[] = $row;
          $row = new stdClass();
          mysqli_stmt_bind_result($stmt, $row->dept_no, $row->dept_name);
        }

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);

        return $rows;
    }

    public function getEmployeesByDept($deptId) {
        $stmt = mysqli_prepare($this->connection, "select employees.emp_no,
                                                  employees.first_name,
                                                  employees.last_name,
                                                  employees.gender,
                                                  dept_emp.dept_no
                                           from employees, dept_emp
                                    where dept_emp.emp_no = employees.emp_no
                                        and dept_emp.dept_no = ?
                                    limit 0,30;");
        $this->throwExceptionOnError();

        mysqli_bind_param($stmt, 's', $deptId);
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
```

```
        $this->throwExceptionOnError();

        $rows = array();

        mysqli_stmt_bind_result($stmt, $row->emp_no, $row->first_name,
                            $row->last_name, $row->gender, $row->dept_no);

        while (mysqli_stmt_fetch($stmt)) {
          $rows[] = $row;
          $row = new stdClass();

          mysqli_stmt_bind_result($stmt, $row->emp_no, $row->first_name,
                            $row->last_name, $row->gender, $row->dept_no);
        }

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);

        return $rows;
    }

    /**
     * Returns the item corresponding to the value specified for the primary key.
     *
     * Add authroization or any logical checks for secure access to your data
     *
     *
     * @return stdClass
     */
    public function getEmployeesByID($itemID) {

        $stmt = mysqli_prepare($this->connection, "SELECT * FROM employees
                                                    where emp_no=?");
        $this->throwExceptionOnError();

        mysqli_bind_param($stmt, 'i', $itemID);
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        mysqli_stmt_bind_result($stmt, $row->emp_no, $row->birth_date,
                                $row->first_name, $row->last_name,
                                $row->gender, $row->hire_date);

        if(mysqli_stmt_fetch($stmt)) {
            return $row;
        } else {
            return null;
        }
    }


    /**
     * Updates the passed item in the table.
     *
     * Add authroization or any logical checks for secure access to your data
```

```
     *
     * @param stdClass $item
     * @return void
     */
    public function updateEmployees($item) {

        $stmt = mysqli_prepare($this->connection, "UPDATE employees
                                    SET emp_no=?, birth_date=?, first_name=?,
                                        last_name=?, gender=?, hire_date=?
                                    WHERE emp_no=?");
        $this->throwExceptionOnError();

        mysqli_bind_param($stmt, 'isssssi', $item->emp_no, $item->birth_date,
                          $item->first_name, $item->last_name, $item->gender,
                          $item->hire_date, $item->emp_no);
        $this->throwExceptionOnError();

        mysqli_stmt_execute($stmt);
        $this->throwExceptionOnError();

        mysqli_stmt_free_result($stmt);
        mysqli_close($this->connection);
    }


    /**
     * Utitity function to throw an exception if an error occurs
     * while running a mysql command.
     */
    private function throwExceptionOnError($link = null) {
        if($link == null) {
            $link = $this->connection;
        }
        if(mysqli_error($link)) {
            $msg = mysqli_errno($link) . ": " . mysqli_error($link);
            throw new Exception('MySQL Error - '. $msg);
        }
    }
}
>?>
```

EmployeeService (ColdFusion)

`EmployeeService.cfc` реализует службу, содержащую одну функцию. `GetEmployeesByID()` принимает идентификатор отдела в качестве аргумента и возвращает список всех сотрудников данного отдела. Эта функция также возвращает даты начала и окончания работы сотрудника в отделе. `GetEmployeesByDept()` выполняет следующий запрос SQL:

```
SELECT
    employees.emp_no,
    employees.birth_date,
    employees.first_name,
    employees.last_name,
    employees.gender,
    employees.hire_date,
    dept_emp.from_date,
    dept_emp.to_date
FROM employees, dept_emp
WHERE dept_emp.emp_no = employees.emp_no and dept_emp.dept_no = departments.dept_no
```

Важная информация. *Типовые службы предназначены только для работы с прототипами. Используйте типовую службу только в доверенной среде разработки. Перед развертыванием этой службы необходимо должным образом повысить безопасность и ограничить доступ к этой службе. Дополнительную информацию о создании безопасных служб ColdFusion см. в документации по ColdFusion About User Security.*

```
<cfcomponent output="false">

<!---
    This sample service contains functions that illustrate typical service operations.
    Use these functions as a starting point for creating your own service implementation.

    This code is for prototyping only.

    Authenticate the user before allowing them to call these methods. You can find more
    information at http://www.adobe.com/go/cf9_usersecurity
--->
    <cffunction name="getEmployeesByDept" output="false" access="remote"  returntype="any" >
        <cfargument name="dept_no" type="string" required="true" />

        <cfset var qItem="">
        <cfquery name="qItem" datasource="employees">
        SELECT employees.emp_no,
            employees.birth_date,
            employees.first_name,
            employees.last_name,
            employees.gender,
            employees.hire_date,
            dept_emp.from_date,
            dept_emp.to_date
        FROM employees, dept_emp
        WHERE dept_emp.emp_no = employees.emp_no and
        dept_emp.dept_no = <CFQUERYPARAM CFSQLTYPE="CF_SQL_VARCHAR"
VALUE="#ARGUMENTS.dept_no#">
        </cfquery>

        <cfreturn qItem>

    </cffunction>

</cfcomponent>?>
```

Импорт служб в проект сервера

1 Создайте проект Flex с именем Associations в Flash Builder:

(PHP) При создании проекта укажите PHP в качестве типа сервера приложения.

(PHP) После создания проекта Flash Builder создаст папку вывода в корневом web-каталоге пользовательской конфигурации PHP. Именем по умолчанию для проекта PHP_Associations является `PHP_Associations-debug`.

(ColdFusion) При создании проекта укажите ColdFusion в качестве типа сервера приложения. Затем выберите ColdFusion Flash Remoting.

2 (PHP) В `PHP_Associations-debug` создайте папку с именем `services`. Скопируйте `EmployeeService.php` в папку `services`.

3 (ColdFusion) Создайте папку с именем `Associations` в корневом web-каталоге пользовательской конфигурации ColdFusion. Скопируйте `EmployeeService.chc` в папку `Associations`.

4 Импортируйте EmployeeService в проект:

Убедитесь, что PHP_Associations является активным проектом в Flash Builder.

Выберите «Данные» > «Соединить с PHP». Для указания класса PHP перейдите к папке `services` и выберите `EmployeeService.php`. Нажмите кнопку «Готово».

Для получения дополнительной информации см. раздел «Подключение к службам данных PHP» на странице 11.

5 Настройте тип возвращаемых данных для операций в EmployeeService.

- DepartmentService

  В контекстном меню `getAllDepartments()` выберите «Настроить тип возвращаемых данных».

  Нажмите «Далее» для автоматического определения типа возвращаемых данных.

  Укажите **Department** для пользовательского типа возвращаемых данных. Нажмите кнопку «Готово».

- EmployeeService

  Для `getEmployeesByDept()` выберите «Настроить тип возвращаемых данных».

  Нажмите «Далее» для автоматического определения типа возвращаемых данных.

  Укажите **d007** в качестве значения параметра. Нажмите кнопку «Далее».

  Укажите **Employee** для пользовательского типа возвращаемых данных. Нажмите кнопку «Готово».

Дополнительную информацию см. в разделе «Настройка типов данных для операций службы данных» на странице 28.

Создание пользовательского интерфейса и привязка возвращаемых данных к компонентам DataGrid

1 В режиме «Дизайн» редактора MXML добавьте два компонента DataGrid в область редактирования.

Компоненты DataGrid находятся в папке «Элементы управления» представления «Компоненты». Перетащите компоненты DataGrid в область редактирования.

Укажите **deptDG** для Departments DataGrid. Укажите **empDeptDG** в качестве идентификатора для Employees DataGrid.

2 В представлении «Данные/службы» перетащите операцию getEmployeesByDept() на Employees DataGrid.

Редактор переключится в режим «Код», где параметр для getEmployeesByDept() будет выделен цветом. Удалите созданный обработчик событий.

Перейдите к Employees DataGrid. Удалите ссылку на атрибут обработчика creationComplete для empDeptDG DataGrid.

После удаления ссылки на обработчик событий первая строка кода для DataGrid выглядит примерно так, как показано ниже:

```
<mx:DataGrid x="361" y="27" id="empDeptDG"
dataProvider="{getEmployeesByDeptResult.lastResult}">
```

Примечание. *Обработчик creationComplete не требуется для Employees DataGrid. Заполнение Employees DataGrid выполняется при выборе отдела в Departments DataGrid.*

3 Перейдите к представлению «Дизайн» редактора. В представлении «Данные/службы» перетащите операцию `getAllDepartments()` на Department DataGrid.

4 Создайте обработчик событий для изменений в Departments DataGrid:

Убедитесь, что выбран Departments DataGrid. В представлении «Свойства» щелкните по значку «При изменении» и выберите параметр «Создать обработчик событий».

Редактор переключится в режим «Код», где будет выбрано тело обработчика событий. Укажите следующие данные для обработчика событий:

```
protected function deptDG_changeHandler(event:ListEvent):void {
    getEmployeesByDeptResult.token =
employeeService.getEmployeesByDept(deptDG.selectedItem.dept_no);
}
```

5 Сохраните и запустите приложение.

После щелчка по названию отдела в Departments DataGrid все сотрудники данного отдела будут перечислены в Employees DataGrid.

Закройте приложение.

6 В режиме «Дизайн» редактора MXML выберите Employees DataGrid. В контекстном меню выберите «Создать форму подробной информации» и выполните следующие шаги:

a Для вызова подробной информации выберите «Вызов службы...».

b Для службы выберите EmployeeService.

c Для операции выберите `getEmployeesByID()`.

d Нажмите кнопку «Готово».

7 В созданном обработчике событий укажите следующие сведения для аргумента к `getEmployeesById()`:

```
protected function empDeptDG_changeHandler(event:ListEvent):void
{
    getEmployeesByIDResult.token =
employeeService.getEmployeesByID(empDeptDG.selectedItem.emp_no);
}
```

8 В режиме «Дизайн» редактора MXML перетащите форму под компоненты DataGrid.

9 Добавьте кнопку рядом с формой и внесите следующие изменения в представлении «Свойства»:

- Для идентификатора укажите updateButton.

- Для метки укажите **Update Employee**.
- Щелкните по значку «По щелчку» и выберите параметр «Создать вызов службы».

  Для службы выберите EmployeeService. Для операции выберите `updateEmployees()`. Редактор переключится в режим «Код».
- Измените вызов службы на updateEmployees(), как показано ниже:

```
protected function updateButton_clickHandler(event:MouseEvent):void
{
    var e:Employee = new Employee();
    e.birth_date = birth_dateTextInput.text;
    e.first_name = first_nameTextInput.text;
    e.last_name = last_nameTextInput.text;
    e.hire_date = hire_dateTextInput.text;
    e.gender = genderTextInput.text;
    e.emp_no = employee.emp_no;

    updateEmployeesResult.token = employeeService.updateEmployees(e);
    getEmployeesByDeptResult.token =
        employeeService.getEmployeesByDept(deptDG.selectedItem.dept_no);
}
```

Глава 4. Получение доступа к данным на стороне сервера

В компонентах доступа к данным Adobe® Flex® для взаимодействия с серверными средами, такими как PHP, Adobe ColdFusion и Microsoft ASP.NET, используются удаленные вызовы процедур. Эти компоненты предоставляют данные клиентским приложениям, созданным в инфраструктуре Adobe Flex, и отправляют данные во внутренние источники данных. Вводная информация о компонентах доступа к данным приведена в разделе «Компоненты доступа к данным» на странице 5.

Использование компонентов HTTPService

Компонент HTTPService может использоваться с любой серверной технологией, включая страницы PHP, страницы ColdFusion, страницы JavaServer (JSP), сервлеты Java, Ruby on Rails и страницы Microsoft ASP. Также HTTPService может использоваться для доступа к web-службам на основе REST.

Для получения справочной информации API о компоненте HTTPService см. раздел mx.rpc.http.mxml.HTTPService.

Работа с данными PHP и SQL

Компонент HTTPService может использоваться с PHP и системой управления базой данных SQL для отображения результатов запроса в базу данных в приложении. Этот компонент можно также использовать для вставки, обновления и удаления данных в базе данных. Вызов страницы PHP для выполнения запроса к базе данных осуществляется с помощью методов GET или POST. Затем можно форматировать данные результата запроса в виде структуры XML и возвратить структуру XML в приложение в ответе HTTP. После возвращения результата в приложение можно отобразить его в одном или нескольких элементах управления пользовательского интерфейса.

MXML-код

В следующем примере в приложении выполняется вызов страницы PHP с использованием метода POST. Страница PHP обеспечивает запрос к таблице базы данных MySQL под названием Users. Результаты запроса форматируются в XML, после чего XML-код возвращается в приложение , в котором привязывается к свойству `dataProvider` элемента управления DataGrid и отображается в элементе управления DataGrid. Приложение также осуществляет отправку имен и адресов электронной почты новых пользователей на PHP страницу, обеспечивая их вставку в таблицу пользователей базы данных.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600"
        creationComplete="send_data()">
    <fx:Declarations>
        <s:HTTPService id="userRequest" url="http://myserver/myproj/request_post2.php"
                                                useProxy="false" method="POST">
            <mx:request xmlns="">
                <username>{username.text}</username>
                <emailaddress>{emailaddress.text}</emailaddress>
            </mx:request>
        </s:HTTPService>
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
        private function send_data():void {
        userRequest.send();
        }
        ]]>
    </fx:Script>
    <mx:Form x="20" y="10" width="300">
        <mx:FormItem>
            <s:Label text="Username" />
            <s:TextInput id="username"/>
        </mx:FormItem>
        <mx:FormItem>
            <s:Label text="Email Address" />
            <s:TextInput id="emailaddress"/>
        </mx:FormItem>
        <s:Button label="Submit" click="send_data()"/>
    </mx:Form>
    <mx:DataGrid id="dgUserRequest" x="20" y="160"
        dataProvider="{userRequest.lastResult.users.user}">
        <mx:columns>
            <mx:DataGridColumn headerText="User ID" dataField="userid"/>
            <mx:DataGridColumn headerText="User Name" dataField="username"/>
        </mx:columns>
    </mx:DataGrid>
    <s:TextInput x="20" y="340" id="selectedemailaddress"
text="{dgUserRequest.selectedItem.emailaddress}"/>

</s:Application>
```

Страница PHP вызывается с помощью метода `send()` HTTPService. Этот вызов осуществляется в методе `send_data()` блока сценария файла MXML.

В свойстве `resultFormat` компонента HTTPService установлено значение `object`, поэтому данные отправляются назад в приложение в виде графа объектов ActionScript. Это значение свойства `resultFormat` задано по умолчанию. В качестве альтернативы можно установить значение `e4x` свойства `resultFormat` для возвращения данных в виде объекта XMLList, в котором может выполняться сценарий ECMAScript для операций XML (E4X). Для установки значения `e4x` в свойстве `resultFormat` необходимо внести следующие незначительные изменения в MXML-код.

Примечание. *Если результат имеет формат `e4x`, корневой узел структуры XML не включается в точечную нотацию при привязке к элементу DataGrid.*

Код XML, возвращаемый в этом примере, не содержит информации пространства имен. Информацию о работе с кодом XML, в действительности содержащем пространства имен, см. в разделе «Обработка результатов в XML-формате посредством формата результатов E4X» на странице 132.

```
...
<s:HTTPService id="userRequest" url="http://myserver/myproj/request_post2.php"
                            useProxy="false" method="POST" resultFormat="e4x">
...
<mx:DataGrid id="dgUserRequest" x="22" y="150"
    dataProvider="{userRequest.lastResult.user}">
...
```

При использовании формата результата `e4x` можно дополнительно связать свойство `lastResult` с объектом XMLListCollection, а затем привязать этот объект к свойству `DataGrid.dataProvider`, как показано в следующем фрагменте кода:

```
<fx:Declarations>
...
    <mx:XMLListCollection id="xc"
         source="{userRequest.lastResult.user}"/>
...
</fx:Declarations>
...
    <mx:DataGrid id="dgUserRequest" x="22" y="128" dataProvider="{xc}">
...
```

Сценарий базы данных MySQL

В коде PHP для этого приложения используется таблица базы данных под названием Users, содержащаяся в базе данных MySQL с именем Sample. Следующий сценарий MySQL обеспечивает создание таблицы:

```
CREATE TABLE `users` (
`userid` int(10) unsigned NOT NULL auto_increment,
`username` varchar(255) collate latin1_general_ci NOT NULL,
`emailaddress` varchar(255) collate latin1_general_ci NOT NULL,
PRIMARY KEY (`userid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci AUTO_INCREMENT=3 ;
```

Код PHP

Данное приложение обеспечивает вызов следующей страницы PHP. Этот код PHP обеспечивает вставку данных и запросы в базу данных SQL, а также возвращает результаты запроса в приложение в структуре XML.

```
<?php
define( "DATABASE_SERVER", "servername" );
define( "DATABASE_USERNAME", "username" );
define( "DATABASE_PASSWORD", "password" );
define( "DATABASE_NAME", "sample" );

//connect to the database.
$mysql = mysql_connect(DATABASE_SERVER, DATABASE_USERNAME, DATABASE_PASSWORD);

mysql_select_db( DATABASE_NAME );

// Quote variable to make safe
function quote_smart($value)
{
    // Stripslashes
    if (get_magic_quotes_gpc()) {
    $value = stripslashes($value);
    }
    // Quote if not integer
    if (!is_numeric($value)) {
    $value = "'" . mysql_real_escape_string($value) . "'";
    }
    return $value;
}

if( $_POST["emailaddress"] AND $_POST["username"])
{
//add the user
$Query = sprintf("INSERT INTO users VALUES ('', %s, %s)",
        quote_smart($_POST['username']), quote_smart($_POST['emailaddress']));

$Result = mysql_query( $Query );
}

//return a list of all the users
$Query = "SELECT * from users";
$Result = mysql_query( $Query );

$Return = "<users>";

while ( $User = mysql_fetch_object( $Result ) )
{
$Return .= "<user><userid>".$User->userid."</userid><username>".
            $User->username."</username><emailaddress>".
            $User->emailaddress."</emailaddress></user>";
}
$Return .= "</users>";
mysql_free_result( $Result );
print ($Return)
?>
```

Работа с ColdFusion и данными SQL

Компонент HTTPService может использоваться со страницей ColdFusion и системой управления базой данных SQL для отображения результатов запроса в базу данных в приложении. Этот компонент можно также использовать для вставки, обновления и удаления данных в базе данных. Вызов страницы ColdFusion для выполнения запроса в базу данных осуществляется с помощью методов GET или POST. Затем можно форматировать данные результата запроса в виде структуры XML и возвратить структуру XML в приложение в ответе HTTP. После возвращения результата в приложение можно отобразить его в одном или нескольких элементах управления пользовательского интерфейса.

MXML-код

В следующем примере в приложении страница ColdFusion вызывается с помощью метода POST. Страница ColdFusion обеспечивает отправку запроса в таблицу базы данных MySQL под названием Users. Результаты запроса форматируются в XML, после чего XML-код возвращается в приложение , в котором привязывается к свойству `dataProvider` элемента управления DataGrid и отображается в элементе управления DataGrid. Приложение также осуществляет передачу имен и адресов электронной почты новых пользователей на страницу ColdFusion, обеспечивая их вставку в таблицу пользователей базы данных.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600"
        creationComplete="userRequest.send()">

    <fx:Declarations>
    <s:HTTPService id="userRequest" url="http://server:8500/flexapp/returncfxml.cfm"
        useProxy="false" method="POST">
        <mx:request xmlns="">
            <username>{username.text}</username>
            <emailaddress>{emailaddress.text}</emailaddress>
        </mx:request>
    </s:HTTPService>
    </fx:Declarations>
    <mx:Form x="22" y="10" width="300">
        <mx:FormItem>
            <s:Label text="Username" />
            <s:TextInput id="username"/>
        </mx:FormItem>
        <mx:FormItem>
            <s:Label text="Email Address" />
            <s:TextInput id="emailaddress"/>
        </mx:FormItem>
        <s:Button label="Submit" click="userRequest.send()"/>
    </mx:Form>
    <mx:DataGrid id="dgUserRequest" x="22" y="128"
        dataProvider="{userRequest.lastResult.users.user}">
        <mx:columns>
            <mx:DataGridColumn headerText="User ID" dataField="userid"/>
            <mx:DataGridColumn headerText="User Name" dataField="username"/>
        </mx:columns>
    </mx:DataGrid>
    <s:TextInput x="22" y="300" id="selectedemailaddress"
text="{dgUserRequest.selectedItem.emailaddress}"/>
</s:Application>
```

Страница ColdFusion вызывается с помощью метода `send()` HTTPService. Этот вызов осуществляется в методе `send_data()` блока сценария файла MXML.

В свойстве `resultFormat` компонента HTTPService установлено значение `object`, поэтому данные отправляются назад в приложение в виде графа объектов ActionScript. Это значение свойства `resultFormat` задано по умолчанию. В качестве альтернативы можно использовать формат результата `e4x` для возвращения данных в виде объекта XMLList, в котором можно выполнять сценарий ECMAScript для операций XML (E4X). Для установки значения `e4x` в свойстве `resultFormat` необходимо внести следующие незначительные изменения в MXML-код.

Примечание. *Если результат имеет формат `e4x`, корневой узел структуры XML не включается в точечную нотацию при привязке к элементу DataGrid.*

Код XML, возвращаемый в этом примере, не содержит информации пространства имен. Информацию о работе с кодом XML, в действительности содержащем пространства имен, см. в разделе «Обработка результатов в XML-формате посредством формата результатов E4X» на странице 132.

```
...
<s:HTTPService id="userRequest" url="http://myserver:8500/flexapp/returncfxml.cfm"
                        useProxy="false" method="POST" resultFormat="e4x">
...
<mx:DataGrid id="dgUserRequest" x="22" y="128"
    dataProvider="{userRequest.lastResult.user}">
...
```

При применении формата результата `e4x` можно дополнительно связать свойство `lastResult` с объектом XMLListCollection, а затем привязать этот объект к свойству `dataProvider` элемента DataGrid, как показано в следующем фрагменте кода:

```
<fx:Declarations>
...
    <mx:XMLListCollection id="xc"
         source="{userRequest.lastResult.user}"/>
...
</fx:Declarations>
...
    <mx:DataGrid id="dgUserRequest" x="22" y="128" dataProvider="{xc}">
...
```

Сценарий SQL

В коде ColdFusion для этого приложения используется таблица базы данных с именем Users, содержащаяся в базе данных MySQL под названием Sample. Следующий сценарий MySQL обеспечивает создание таблицы:

```
CREATE TABLE `users` (
`userid` int(10) unsigned NOT NULL auto_increment,
`username` varchar(255) collate latin1_general_ci NOT NULL,
`emailaddress` varchar(255) collate latin1_general_ci NOT NULL,
PRIMARY KEY (`userid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci AUTO_INCREMENT=3 ;
```

Код ColdFusion

Приложение, указанное в разделе «Работа с ColdFusion и данными SQL», вызывает приложение ColdFusion `returncfxml.cfm`. Указанный код ColdFusion обеспечивает вставку данных и отправку запросов к базе данных SQL, а также возвращает результаты запросов в приложение На странице ColdFusion тег `cfquery` используется для вставки данных в базу данных и отправки запросов к базе данных, а тег `cfxml` применяется в целях форматирования результатов запросов в XML-структуру.

```
<!--- returncfxml.cfm --->

<cfprocessingdirective pageencoding = "utf-8" suppressWhiteSpace = "Yes">
<cfif isDefined("username") and isDefined("emailaddress") and username NEQ "">
    <cfquery name="addempinfo" datasource="sample">
    INSERT INTO users (username, emailaddress) VALUES (
    <cfqueryparam value="#username#" cfsqltype="CF_SQL_VARCHAR" maxlength="255">,
        <cfqueryparam value="#emailaddress#" cfsqltype="CF_SQL_VARCHAR" maxlength="255"> )
</cfquery>
</cfif>
<cfquery name="alluserinfo" datasource="sample">
    SELECT userid, username, emailaddress FROM users
</cfquery>
<cfxml variable="userXML">
    <users>
        <cfloop query="alluserinfo">
            <cfoutput>
                <user>
                    <userid>#toString(userid)#</userid>
                    <username>#username#</username>
                    <emailaddress>#emailaddress#</emailaddress>
                </user>
        </cfoutput>
        </cfloop>
    </users>
</cfxml>
<cfoutput>#userXML#</cfoutput>
</cfprocessingdirective>
```

Работа со страницами JavaServer

Компонент HTTPService Flex можно использовать со страницей JSP и системой управления базой данных SQL для отображения результатов запроса в базу данных в приложении. Этот компонент можно также использовать для вставки, обновления и удаления данных в базе данных. Вызов страницы JSP для выполнения запроса в базу данных осуществляется с помощью метода GET или POST. Затем можно форматировать данные результата запроса в виде структуры XML и возвратить структуру XML в приложение в ответе HTTP. После возвращения результата в приложение можно отобразить его в одном или нескольких элементах управления пользовательского интерфейса.

MXML-код

В приложении из следующего примера выполняется вызов страницы JSP, извлекающей данные из базы данных SQL. Результаты запроса к базе данных форматируются в виде XML, после чего код XML возвращается в приложение , в котором привязывается к свойству `dataProvider` элемента управления DataGrid и отображается в элементе управления DataGrid.

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600">

    <fx:Declarations>
        <s:HTTPService id="srv" url="catalog.jsp"/>
    </fx:Declarations>

    <mx:DataGrid dataProvider="{srv.lastResult.catalog.product}"
                                  width="100%" height="100%"/>

    <s:Button label="Get Data" click="srv.send()"/>

</mx:Application>
```

Страница JSP вызывается с помощью метода `send()` HTTPService. Этот вызов осуществляется в событии `click` элемента управления Button в MXML-файле.

В свойстве `resultFormat` компонента HTTPService установлено значение `object`, поэтому данные отправляются назад в приложение в виде графа объектов ActionScript. Это значение свойства `resultFormat` задано по умолчанию. В качестве альтернативы можно использовать формат результата `e4x` для возвращения данных в виде объекта XMLList, в котором можно выполнять сценарий ECMAScript для операций XML (E4X). Для установки значения `e4x` в свойстве `resultFormat` необходимо внести следующие незначительные изменения в MXML-код.

Примечание. *Если результат имеет формат `e4x`, корневой узел структуры XML не включается в точечную нотацию при привязке к элементу DataGrid.*

Код XML, возвращаемый в этом примере, не содержит информации пространства имен. Информацию о работе с кодом XML, в действительности содержащем пространства имен, см. в разделе «Обработка результатов в XML-формате посредством формата результатов E4X» на странице 132.

```
...
    <s:HTTPService id="srv" url="catalog.jsp" resultFormat="e4x"/>
...
    <mx:DataGrid dataProvider="{srv.lastResult.product}" width="100%" height="100%"/>
```

При использовании формата результата e4x можно дополнительно связать свойство `lastResult` с объектом XMLListCollection, а затем привязать этот объект к свойству DataGrid.dataProvider:

```
<fx:Declarations>
...
    <mx:XMLListCollection id="xc"
         source="{userRequest.lastResult.user}"/>
...
</fx:Declarations>
...
    <mx:DataGrid id="dgUserRequest" x="22" y="128" dataProvider="{xc}">
...
```

Код JSP

В следующем примере показана страница JSP, используемая в этом приложении. Непосредственное обращение к базе данных на этой странице JSP отсутствует. Данные поступают из класса Java с именем ProductService, в котором для представления отдельных продуктов, в свою очередь, применяется класс Java с именем Product.

```
<%@page import="flex.samples.product.ProductService,
                flex.samples.product.Product,
                java.util.List"%>
<?xml version="1.0" encoding="utf-8"?>
<catalog>
<%
    ProductService srv = new ProductService();
    List list = null;
    list = srv.getProducts();
    Product product;
    for (int i=0; i<list.size(); i++)
    {
        product = (Product) list.get(i);
%>
<product productId="<%= product.getProductId()%>">
<name><%= product.getName() %></name>
<description><%= product.getDescription() %></description>
<price><%= product.getPrice() %></price>
<image><%= product.getImage() %></image>
<category><%= product.getCategory() %></category>
<qtyInStock><%= product.getQtyInStock() %></qtyInStock>
</product>
<%
    }
%>
</catalog>
```

Вызов служб HTTP в ActionScript

В следующем примере показан вызов службы HTTP в блоке сценария ActionScript. При вызове метода `useHTTPService()` объявляется служба, устанавливается адрес назначения и прослушиватели события ошибки и результата события, а также вызывается метод `send()` службы.

```
<?xml version="1.0"?>
<!-- fds\rpc\HttpServiceInAS.mxml. Compiles -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.rpc.http.HTTPService;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;

            private var service:HTTPService
            public function useHttpService(parameters:Object):void {
                service = new HTTPService();
                service.url = "catalog.jsp";
                service.method = "POST";
                service.addEventListener("result", httpResult);
                service.addEventListener("fault", httpFault);
                service.send(parameters);
            }

            public function httpResult(event:ResultEvent):void {
                var result:Object = event.result;
            //Do something with the result.
            }

            public function httpFault(event:FaultEvent):void {
                var faultstring:String = event.fault.faultString;
                Alert.show(faultstring);
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Использование компонентов WebService

Приложения, созданные в инфраструктуре Flex, могут взаимодействовать с web-службами на основе SOAP, определяющими интерфейсы в документе на языке описания web-служб 1.1 (WSDL 1.1), доступном в виде URL-адреса. WSDL является стандартным форматом описания сообщений, распознаваемых web-службой, форматом ее ответов на эти сообщения, протоколов, поддерживаемых web-службой, и адресов для отправки сообщений. В API web-службы Flex, как правило, поддерживаются протокол Simple Object Access Protocol (SOAP) 1.1, схема XML 1.0 (версии 1999, 2000 и 2001) и кодировка RPC, литерал RPC и литерал документа WSDL 1.1 (пустые и включаемые параметры стиля). В web-службах двух наиболее популярных типов используется привязка к кодировке или литералу удаленного вызова процедуры (RPC) SOAP; условия *encoded* и *literal* указывают на вид сопоставления WSDL-SOAP, применяемый службой.

Flex поддерживает запросы web-службы и результаты, отформатированные в виде сообщений SOAP. Протокол SOAP обеспечивает определение формата на основе XML, который может применяться для обмена структурированной и типизированной информацией между клиентом web-службы, например, приложением, созданным в Flex, и web-службой.

Adobe® Flash® Player применяется в изолированной среде безопасности, для которой установлены ограничения на доступ через HTTP для приложений, созданных в Flex или Flash. Для приложений, созданных с помощью Flash, разрешен доступ только через HTTP к ресурсам на том же домене и через тот же протокол, с которого они обслуживаются. Это представляет собой проблему для web-служб, так как обычно к ним выполняется доступ из удаленных местоположений. Доступная в ADEP Data Services и BlazeDS служба прокси перехватывает запросы к удаленным web-службам, выполняет перенаправление запросов, а также возвращает ответы клиенту.

Если ADEP Data Services или BlazeDS не используются, можно получить доступ к web-службам того же домена, в котором находится приложение, или применить файл crossdomain.xml (файл междоменной политики), позволяющий осуществлять доступ из домена приложения, который должен быть установлен на web-сервере, на котором находится служба RPC.

Справочную информацию API о компоненте WebService см. в разделе mx.rpc.soap.mxml.WebService.

Пример приложения WebService

Следующий пример кода предназначен для приложения, в котором компонент WebService применяется в целях вызова операций web-службы.

MXML-код

В приложении из следующего примера выполняется вызов web-службы, обеспечивающей запрос в таблицу Users базы данных SQL и возвращение данных в приложение , в котором они привязываются к свойству `dataProvider` элемента управления DataGrid и отображаются в элементе управления DataGrid. Приложение также осуществляет передачу имен и адресов электронной почты новых пользователей в web-службу, в которой выполняется вставка данных в таблицу пользователей базы данных. Внутренняя реализация web-службы представляет собой компонент ColdFusion; этот же компонент ColdFusion применяется в качестве удаленного объекта в разделе «Использование компонентов RemoteObject» на странице 107.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600">
    <fx:Declarations>
        <s:WebService
            id="userRequest"
            wsdl="http://localhost:8500/flexapp/returnusers.cfc?wsdl">

            <mx:operation name="returnRecords" resultFormat="object"
                          fault="mx.controls.Alert.show(event.fault.faultString)"
                          result="remotingCFCHandler(event)"/>

            <mx:operation name="insertRecord" result="insertCFCHandler()"
                          fault="mx.controls.Alert.show(event.fault.faultString)"/>
        </s:WebService>
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
            import mx.rpc.events.ResultEvent;

            private function remotingCFCHandler(e:ResultEvent):void
            {
                dgUserRequest.dataProvider = e.result;
            }
```

```
            private function insertCFCHandler():void
            {
                userRequest.returnRecords();
            }
            private function clickHandler():void
            {
                userRequest.insertRecord(username.text, emailaddress.text);
            }
        ]]>
    </fx:Script>

    <mx:Form x="22" y="10" width="300">
        <mx:FormItem>
            <s:Label text="Username" />
            <s:TextInput id="username"/>
        </mx:FormItem>
        <mx:FormItem>
            <s:Label text="Email Address" />
            <s:TextInput id="emailaddress"/>
        </mx:FormItem>
        <s:Button label="Submit" click="clickHandler()"/>
    </mx:Form>

    <mx:DataGrid id="dgUserRequest" x="22" y="160">
        <mx:columns>
            <mx:DataGridColumn headerText="User ID" dataField="USERID"/>
            <mx:DataGridColumn headerText="User Name" dataField="USERNAME"/>
        </mx:columns>
    </mx:DataGrid>

    <s:TextInput x="22" y="320" id="selectedemailaddress"
text="{dgUserRequest.selectedItem.emailaddress}"/>
</s:Application>
```

Документ WSDL

В следующем примере представлен документ WSDL, определяющий API web-службы:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://flexapp"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://flexapp" xmlns:intf="http://flexapp"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns1="http://rpc.xml.coldfusion"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by ColdFusion version 8,0,0,171651-->
    <wsdl:types>
<schema targetNamespace="http://rpc.xml.coldfusion"
xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://flexapp"/>
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    <complexType name="CFCInvocationException">
<sequence/>
    </complexType>

    <complexType name="QueryBean">
<sequence>
    <element name="columnList" nillable="true" type="impl:ArrayOf_xsd_string"/>
    <element name="data" nillable="true" type="impl:ArrayOfArrayOf_xsd_anyType"/>
</sequence>
    </complexType>
</schema>
<schema targetNamespace="http://flexapp" xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://rpc.xml.coldfusion"/>

    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    <complexType name="ArrayOf_xsd_string">
<complexContent>
    <restriction base="soapenc:Array">
<attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"/>
    </restriction>
</complexContent>
    </complexType>
    <complexType name="ArrayOfArrayOf_xsd_anyType">

<complexContent>
    <restriction base="soapenc:Array">
<attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:anyType[][]"/>
    </restriction>
</complexContent>
    </complexType>
</schema>
    </wsdl:types>

    <wsdl:message name="CFCInvocationException">

<wsdl:part name="fault" type="tns1:CFCInvocationException"/>
</wsdl:message>
<wsdl:message name="returnRecordsRequest">
</wsdl:message>
<wsdl:message name="insertRecordResponse">
</wsdl:message>
<wsdl:message name="returnRecordsResponse">
```

```
<wsdl:part name="returnRecordsReturn" type="tns1:QueryBean"/>
</wsdl:message>
<wsdl:message name="insertRecordRequest">
<wsdl:part name="username" type="xsd:string"/>
<wsdl:part name="emailaddress" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="returncfxml">
<wsdl:operation name="insertRecord" parameterOrder="username emailaddress">
<wsdl:input message="impl:insertRecordRequest" name="insertRecordRequest"/>
<wsdl:output message="impl:insertRecordResponse" name="insertRecordResponse"/>
<wsdl:fault message="impl:CFCInvocationException" name="CFCInvocationException"/>
</wsdl:operation>
<wsdl:operation name="returnRecords">
<wsdl:input message="impl:returnRecordsRequest" name="returnRecordsRequest"/>
<wsdl:output message="impl:returnRecordsResponse" name="returnRecordsResponse"/>
<wsdl:fault message="impl:CFCInvocationException" name="CFCInvocationException"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="returncfxml.cfcSoapBinding" type="impl:returncfxml">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="insertRecord">
<wsdlsoap:operation soapAction=""/>
<wsdl:input name="insertRecordRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://flexapp" use="encoded"/>
</wsdl:input>
<wsdl:output name="insertRecordResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://flexapp" use="encoded"/>
</wsdl:output>
<wsdl:fault name="CFCInvocationException">
<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
name="CFCInvocationException" namespace="http://flexapp" use="encoded"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="returnRecords">
<wsdlsoap:operation soapAction=""/>
<wsdl:input name="returnRecordsRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://flexapp" use="encoded"/>
</wsdl:input>
<wsdl:output name="returnRecordsResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://flexapp" use="encoded"/>
</wsdl:output>
<wsdl:fault name="CFCInvocationException">
<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
name="CFCInvocationException" namespace="http://flexapp" use="encoded"/>
</wsdl:fault>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="returncfxmlService">
<wsdl:port binding="impl:returncfxml.cfcSoapBinding" name="returncfxml.cfc">
<wsdlsoap:address location="http://localhost:8500/flexapp/returnusers.cfc"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Вызов web-служб в ActionScript

В следующем примере показан вызов web-службы в блоке сценария ActionScript. При вызове метода `useWebService()` обеспечивается объявление службы, установка адреса назначения, выбор документа WSDL и вызов метода `echoArgs()` службы.

Примечание. *После объявления компонента WebService в ActionScript вызовите метод* `WebService.loadWSDL()`.

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceInAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        import mx.rpc.soap.WebService;
        import mx.rpc.events.ResultEvent;
        import mx.rpc.events.FaultEvent;
        private var ws:WebService;
        public function useWebService(intArg:int, strArg:String):void {
            ws = new WebService();
            ws.wsdl="http://myserver:8500/flexapp/app1.cfc?wsdl";
            ws.echoArgs.addEventListener("result", echoResultHandler);
            ws.addEventListener("fault", faultHandler);
            ws.loadWSDL();
            ws.echoArgs(intArg, strArg);
        }

        public function echoResultHandler(event:ResultEvent):void {
            var retStr:String = event.result.echoStr;
            var retInt:int = event.result.echoInt;
         //Do something.
        }

        public function faultHandler(event:FaultEvent):void {
      //deal with event.fault.faultString, etc
        }
        ]]>
    </mx:Script>
</mx:Application>
```

Зарезервированные имена операций

Для доступа к операциям WebService обычно требуется указать их имена после переменной службы. Однако если имя операции будет совпадать с определенным методом службы, может возникнуть конфликт имен. Для возвращения операции с заданным именем для компонента WebService в ActionScript можно использовать следующий метод:

```
public function getOperation(name:String):Operation
```

Чтение документов WSDL

Документ WSDL может быть просмотрен в web-браузере, редакторе простого текста, редакторе XML или в среде разработки, такой как Adobe Dreamweaver, содержащей встроенную служебную функцию для отображения документов WSDL в предназначенном для чтения формате.

Документ WSDL содержит теги, описанные в следующей таблице.

Тег	Описание
<binding>	Определяет протокол, применяемый клиентами, например приложениями, созданными в Flex, для взаимодействия с веб-службой. Существуют привязки для SOAP, HTTP, GET, HTTP POST и MIME. В среде Flex поддерживаются только привязки SOAP.
<fault>	Указывает значение ошибки, возвращенное в результате проблемы при обработке сообщения.
<input>	Определяет сообщение, отправляемое клиентами, например приложениями, созданными в Flex, в веб-службу.
<message>	Определяет данные, предаваемые операцией WebService.
<operation>	Определяет комбинацию тегов <input>, <output> и <fault>.
<output>	Определяет сообщение, отправляемое веб-службой клиенту веб-службы, например приложению, созданному в Flex.
<port>	Позволяет указать конечную точку web-службы, определяющую связь между привязкой и сетевым адресом.
<portType>	Определяет одну или нескольких операций, предоставляемых web-службой.
<service>	Определяет совокупность тегов <port> . Каждая служба соответствует одному тегу <portType> и определяет различные способы получения доступа к операциям в указанном теге <portType>.
<types>	Определяет типы данных, используемые в сообщениях web-службы.

Операции, ориентированные на RPC и на документ

Файл WSDL может определять как операции, ориентированные на RPC, так и операции, ориентированные на документ (документ-литерал). В среде Flex поддерживаются операции обоих стилей.

При вызове операции, ориентированной на RPC, приложение отправляет сообщение SOAP с указанием операции и ее параметров. При вызове операции, ориентированной на документ, приложение отправляет сообщение SOAP, содержащее документ XML.

В документе WSDL каждый тег <port> обладает свойством binding, указывающим имя определенного тега <soap:binding>, как показано в следующем примере:

```
<binding name="InstantMessageAlertSoap" type="s0:InstantMessageAlertSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document"/>
```

Свойство style связанного тега <soap:binding> позволяет определить стиль операции. В этом примере стилем является document.

Любая операция службы позволяет определить тот же стиль или отменить стиль, указанный для порта, связанного со службой, как показано в следующем примере:

```
<operation name="SendMSN">
    <soap:operation soapAction="http://www.bindingpoint.com/ws/imalert/
        SendMSN"style="document"/>
```

Web-службы, хранящие информацию о состоянии

Flex использует сеансы сервера Java для поддержки состояния конечных точек web-служб, использующих файлы cookie для сохранения информации о сеансе. Эта возможность выступает в роли посредника между приложениями и web-службами. При этом идентификатор конечной точки добавляется к данным, передаваемым конечной точкой в приложение. Приложение получает информацию о сеансе, отправляемую конечной точкой. Эта возможность не требует настройки и не поддерживается для адресов назначения, применяющих канал RTMP при использовании службы прокси.

Работа с заголовками SOAP

Заголовок SOAP является необязательным тегом в оболочке SOAP, содержащим, как правило, специфичную для приложения информацию, например, данные аутентификации.

Добавление заголовков SOAP в запросы web-службы

В некоторых web-службах необходимо передать заголовок SOAP при вызове операции.

Добавление заголовка SOAP ко всем операциям web-службы или отдельным операциям осуществляется путем вызова метода `addHeader()` объекта WebService или объекта операции, либо метода `addSimpleHeader()` в функции прослушивателя события.

При использовании метода `addHeader()` сначала необходимо создать объекты SOAPHeader и QName по отдельности. Метод `addHeader()` имеет следующую подпись:

```
addHeader(header:mx.rpc.soap.SOAPHeader):void
```

Для создания объекта SOAPHeader используйте следующий конструктор:

```
SOAPHeader(qname:QName, content:Object)
```

Для создания объекта QName в первом параметре метода `SOAPHeader()` используйте следующий конструктор:

```
QName(uri:String, localName:String)
```

Параметр `content` конструктора `SOAPHeader()` представляет собой набор пар вида «имя-значение» на основе следующего формата:

```
{name1:value1, name2:value2}
```

Метод `addSimpleHeader()` представляет собой наиболее быстрый метод получения отдельного заголовка SOAP вида «имя-значение». При использовании метода `addSimpleHeader()` в параметрах метода создаются объекты SOAPHeader и QName. Метод `addSimpleHeader()` имеет следующую подпись:

```
addSimpleHeader(qnameLocal:String, qnameNamespace:String, headerName:String,
    headerValue:Object):void
```

Метод `addSimpleHeader()` принимает следующие параметры:

- *`qnameLocal`* является локальным именем для заголовка QName.
- *`qnameNamespace`* является пространством имен для заголовка QName.
- *`headerName`* является именем заголовка.
- *`headerValue`* является значением заголовка. Значение может являться строкой, если это простое значение, объектом с базовым XML-кодированием, или XML, если требуется самостоятельно определить XML заголовка.

  Код в следующем примере показывает, каким образом необходимо использовать метод `addHeader()` и метод `addSimpleHeader()` для добавления заголовка SOAP. Методы вызываются в функции прослушивателя событий под названием `headers`, а прослушиватель событий присваивается в свойстве `load` тега `<mx:WebService>`:

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceAddHeader.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="600">
    <mx:WebService id="ws" wsdl="http://myserver:8500/flexapp/app1.cfc?wsdl"
load="headers();"/>
    <mx:Script>
        <![CDATA[
            import mx.rpc.soap.SOAPHeader;
            private var header1:SOAPHeader;
            private var header2:SOAPHeader;
            public function headers():void {

            // Create QName and SOAPHeader objects.
            var q1:QName=new QName("http://soapinterop.org/xsd", "Header1");
            header1=new SOAPHeader(q1, {string:"bologna",int:"123"});
            header2=new SOAPHeader(q1, {string:"salami",int:"321"});


            // Add the header1 SOAP Header to all web service requests.
            ws.addHeader(header1);

            // Add the header2 SOAP Header to the getSomething operation.
            ws.getSomething.addHeader(header2);

            // Within the addSimpleHeader method,
            // which adds a SOAP header to web
            //service requests, create SOAPHeader and QName objects.
            ws.addSimpleHeader
                ("header3", "http://soapinterop.org/xsd", "foo","bar");
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Очистка заголовков SOAP

Метод `clearHeaders()` объекта WebService или операции применяется для удаления добавленных к объекту заголовков SOAP, как показано в примере ниже для объекта WebService. Метод `clearHeaders()` должен вызываться на том уровне (WebService или операции), на котором был добавлен заголовок.

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceClearHeader.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="600" >

    <!-- The value of the destination property is for demonstration only and is not a real
destination. -->
    <mx:WebService id="ws" wsdl="http://myserver:8500/flexapp/app1.cfc?wsdl"
load="headers();"/>

    <mx:Script>
        <![CDATA[
            import mx.rpc.*;
            import mx.rpc.soap.SOAPHeader;

            private function headers():void {
                // Create QName and SOAPHeader objects.
                var q1:QName=new QName("Header1", "http://soapinterop.org/xsd");
                var header1:SOAPHeader=new SOAPHeader(q1, {string:"bologna",int:"123"});
                var header2:SOAPHeader=new SOAPHeader(q1, {string:"salami",int:"321"});
                    // Add the header1 SOAP Header to all web service request.
                ws.addHeader(header1);
                // Add the header2 SOAP Header to the getSomething operation.
                ws.getSomething.addHeader(header2);

                // Within the addSimpleHeader method, which adds a SOAP header to all
                // web service requests, create SOAPHeader and QName objects.
                ws.addSimpleHeader("header3","http://soapinterop.org/xsd", "foo", "bar");
            }

            // Clear SOAP headers added at the WebService and Operation levels.
            private function clear():void {
                ws.clearHeaders();
                ws.getSomething.clearHeaders();
            }
        ]]>
    </mx:Script>

    <mx:HBox>
            <mx:Button label="Clear headers and run again" click="clear();"/>
    </mx:HBox>

</mx:Application>
```

Перенаправление web-службы на другой URL-адрес

В некоторых web-службах требуется изменение URL-адреса конечной точки после обработки WSDL и осуществление начального вызова web-службы. В качестве примера предположим, что необходимо использовать web-службу, требующую передачи учетных данных безопасности. При вызове web-службы для отправки учетных данных входа в систему, она принимает учетные данные и возвращает фактический URL-адрес конечной точки, который необходим для использования бизнес-операций службы. Перед вызовом бизнес-операций необходимо изменить значение свойства `endpointURI` компонента WebService.

В следующем примере показан прослушиватель результата события, сохраняющий URL-адрес конечной точки, возвращаемый web-службой в составе переменной, и обеспечивающий последующую передачу переменной в функцию с целью изменения URL-адреса конечной точки для последующих запросов:

```
...
public function onLoginResult(event:ResultEvent):void {

//Extract the new service endpoint from the login result.
var newServiceURL = event.result.serverUrl;

// Redirect all service operations to the URL received in the login result.
    serviceName.endpointURI=newServiceURL;

}
...
```

Web-служба, требующая передачи учетных данных безопасности, также должна обеспечивать возвращение идентификатора, который необходимо присоединить к заголовку SOAP для последующих запросов. Для получения дополнительной информации см. раздел «Работа с заголовками SOAP» на странице 95.

Сериализация данных web-службы

Кодирование данных ActionScript

В следующей таблице представлены сопоставления кодирования из типов ActionScript 3.0 в сложные типы XML-схемы.

Определение схемы XML	**Поддерживаемые типы ActionScript 3.0**	**Примечания**
Элементы верхнего уровня		
xsd:element nillable == true	Object	Если вводимым значением является `null`, кодированные выходные данные устанавливаются с атрибутом `xsi:nil`.
xsd:element fixed ! = null	Object	Вводимое значение игнорируется и вместо него применяется фиксированное значение.
xsd:element default ! = null	Object	Если вводимым значением является `null`, вместо него используется это значение по умолчанию.
Локальные элементы		
xsd:element maxOccurs == 0	Object	Вводимое значение игнорируется и опускается в кодированных выходных данных.
xsd:element maxOccurs == 1	Object	Вводимое значение обрабатывается как отдельный объект. Если связанным типом является массив с кодировкой SOAP, то массивы и реализации `mx.collection.IList` передаются целиком и обрабатываются как специальные случаи с использованием кодировщика SOAP для этого типа.
xsd:element maxOccurs > 1	Object	Входное значение должно быть повторяемым (например, массивом или реализацией `mx.collections.IList`), несмотря на включение неповторяющихся значений перед обработкой. Отдельные элементы кодируются в качестве отдельных объектов в соответствии с определением.
xsd:element minOccurs == 0	Object	Если вводимое значение - undefined или `null`, кодированные выходные данные опускаются.

В следующей таблице представлены сопоставления кодирования из типов ActionScript 3.0 во встроенные типы XML-схемы.

Тип схемы XML	**Поддерживаемые типы ActionScript 3.0**	**Примечания**
`xsd:anyType` `xsd:anySimpleType`	Object	Boolean -> `xsd:boolean` ByteArray -> `xsd:base64Binary` Date -> `xsd:dateTime` int -> `xsd:int` Number -> `xsd:double` String -> `xsd:string` uint -> `xsd:unsignedInt`
`xsd:base64Binary`	`flash.utils.ByteArray`	Используется `mx.utils.Base64Encoder` (без включения строки).
`xsd:boolean`	Boolean Number Object	Всегда кодируется как `true` или `false`. Если `Number == 1`, то `true`, в противном случае – `false`. Если `Object.toString() ==` «true» или «1», то `true`, в противном случае – `false`.
`xsd:byte` `xsd:unsignedByte`	Number String	Вначале String преобразуется в Number.
`xsd:date`	Date Number String	Используются методы доступа к данным UTC. Для установки Date.time применяется числовое значение. Предполагается, что строка предварительно отформатирована и кодирована как есть.
`xsd:dateTime`	Date Number String	Используются методы доступа к данным UTC. Для установки Date.time применяется числовое значение. Предполагается, что строка предварительно отформатирована и кодирована как есть.
`xsd:decimal`	Number String	Используется `Number.toString()`. Infinity, -Infinity и NaN недопустимы для этого типа. Вначале строка преобразуется в число.
`xsd:double`	Number String	Ограничено диапазоном числового значения. Вначале String преобразуется в Number.
`xsd:duration`	Object	Вызывается `Object.toString()`.
`xsd:float`	Number String	Ограничено диапазоном числового значения. Вначале String преобразуется в Number.
`xsd:gDay`	Date Number String	Используется `Date.getUTCDate()`. Числовое значение используется для обозначения дня. Строка анализируется как числовое значение дня.

Тип схемы XML	**Поддерживаемые типы ActionScript 3.0**	**Примечания**
`xsd:gMonth`	Date Number String	Используется `Date.getUTCMonth()` . Числовое значение применяется для обозначения месяца. Строка анализируется как числовое значение, обозначающее месяц.
`xsd:gMonthDay`	Date String	Используются `Date.getUTCMonth()` и `Date.getUTCDate()` . Строка анализируется по частям: как обозначение месяца и дня.
`xsd:gYear`	Date Number String	Используется `Date.getUTCFullYear()` . Числовое значение применяется непосредственно для обозначения года. Строка анализируется как числовое значение, обозначающее год.
`xsd:gYearMonth`	Date String	Используются `Date.getUTCFullYear()` и `Date.getUTCMonth()`. Строка анализируется по частям: как обозначение года и месяца.
`xsd:hexBinary`	flash.utils.ByteArray	Используется mx.utils.HexEncoder.

`xsd:integer`
`and derivatives:`
`xsd:negativeInteger`
`xsd:nonNegativeInteger`
`xsd:positiveInteger`
`xsd:nonPositiveInteger`
	Number
	String
		Ограничено диапазоном числового значения.
		Вначале String преобразуется в Number.

`xsd:int` `xsd:unsignedInt`	Number String	Вначале String преобразуется в Number.
`xsd:long` `xsd:unsignedLong`	Number String	Вначале String преобразуется в Number.
`xsd:short` `xsd:unsignedShort`	Number String	Вначале String преобразуется в Number.

Тип схемы XML	Поддерживаемые типы ActionScript 3.0	Примечания

`xsd:string`
`and derivatives:`
`xsd:ID`
`xsd:IDREF`
`xsd:IDREFS`
`xsd:ENTITY`
`xsd:ENTITIES xsd:language`
`xsd:Name`
`xsd:NCName`
`xsd:NMTOKEN`
`xsd:NMTOKENS`
`xsd:normalizedString`
`xsd:token`
	Object	Вызывается `Object.toString()`.

`xsd:time`	Date Number String	Используются методы доступа к данным UTC. Для установки Date.time применяется числовое значение. Предполагается, что строка предварительно отформатирована и кодирована как есть.
`xsi:nil`	null	Если в определении соответствующего элемента схемы XML minOccurs > 0, значение `null` кодируется с использованием xsi:nil; в противном случае, элемент полностью опускается.

В следующей таблице представлены сопоставления из типов ActionScript 3.0 в типы с кодировкой SOAP.

Тип SOAPENC	Поддерживаемые типы ActionScript 3.0	Примечания
soapenc:Array	Array mx.collections.IList	Массивы с кодировкой SOAP представляют собой особые случаи и поддерживаются только в web-службах с кодировкой RPC.
soapenc:base64	flash.utils.ByteArray	Кодируется тем же способом, что и `xsd:base64Binary`.
soapenc:*	Object	Любой другой тип с кодировкой SOAP обрабатывается так, как если бы он находился в пространстве имен XSD на основе `localName` типа QName.

Декодирование схемы XML и SOAP для ActionScript 3.0

В следующей таблице представлены сопоставления декодирования по умолчанию из встроенных типов XML-схемы в типы ActionScript 3.0.

Тип схемы XML	**Декодируемые типы ActionScript 3.0**	**Примечания**
`xsd:anyType` `xsd:anySimpleType`	String Boolean Number	Если содержимое является пустым -> `xsd:string`. Если содержимое приводится к числу и значением является NaN; или, если содержимое начинается с 0 или -0 либо если содержимое заканчивается на E: затем, если содержимое имеет значение true или false -> `xsd:boolean` в противном случае -> `xsd:string`. В противном случае содержимым является допустимое числовое значение и поэтому -> `xsd:double`.
`xsd:base64Binary`	`flash.utils.ByteArray`	Используется `mx.utils.Base64Decoder`.
`xsd:boolean`	Boolean	Если значением является true или 1, то `true`, в противном случае – `false`.
`xsd:date`	Date	При отсутствии информации о часовом поясе применяется местное время.
`xsd:dateTime`	Date	При отсутствии информации о часовом поясе применяется местное время.
`xsd:decimal`	Number	Содержимое создается с помощью `Number(content)` и поэтому ограничено диапазоном числового значения.
`xsd:double`	Number	Содержимое создается с помощью `Number(content)` и поэтому ограничено диапазоном числового значения.
`xsd:duration`	String	Содержимое возвращается со свернутым пустым пространством.
`xsd:float`	Number	Содержимое преобразуется с помощью `Number(content)` и поэтому ограничено диапазоном числа.
`xsd:gDay`	uint	Содержимое преобразуется с помощью `uint(content)`.
`xsd:gMonth`	uint	Содержимое преобразуется с помощью `uint(content)`.
`xsd:gMonthDay`	String	Содержимое возвращается со свернутым пустым пространством.
`xsd:gYear`	uint	Содержимое преобразуется с помощью `uint(content)`.
`xsd:gYearMonth`	String	Содержимое возвращается со свернутым пустым пространством.
`xsd:hexBinary`	`flash.utils.ByteArray`	Используется `mx.utils.HexDecoder`.

Тип схемы XML	**Декодируемые типы ActionScript 3.0**	**Примечания**

`xsd:integer`
`and derivatives:`
`xsd:byte`
`xsd:int`
`xsd:long`
`xsd:negativeInteger`
`xsd:nonNegativeInteger`
`xsd:nonPositiveInteger`
`xsd:positiveInteger`
`xsd:short`
`xsd:unsignedByte`
`xsd:unsignedInt`
`xsd:unsignedLong`
`xsd:unsignedShort`
	Number	Содержимое декодируется с помощью `parseInt()`.

`xsd:string`
и производные:
`xsd:ID`
`xsd:IDREF`
`xsd:IDREFS`
`xsd:ENTITY`
`xsd:ENTITIES xsd:language`
`xsd:Name`
`xsd:NCName`
`xsd:NMTOKEN`
`xsd:NMTOKENS`
`xsd:normalizedString`
`xsd:token`
	String	Необработанное содержимое возвращается в виде строки.

`xsd:time`	Date	При отсутствии информации о часовом поясе применяется местное время.
`xsi:nil`	null	

В следующей таблице представлены отображения декодирования из типов с кодировкой SOAP в типы ActionScript 3.0.

Тип SOAPENC	Декодируемый тип ActionScript	Примечания
`soapenc:Array`	Array `mx.collections.ArrayCollection`	Массивы с кодировкой SOAP представляют собой особые случаи. Если значением `makeObjectsBindable` является `true`, результат включается в ArrayCollection; в противном случае, возвращается простой массив.
`soapenc:base64`	`flash.utils.ByteArray`	Декодируется тем же способом, что и `xsd:base64Binary`.
`soapenc:*`	Object	Любой другой тип с кодировкой SOAP обрабатывается так, как если бы он находился в пространстве имен XSD на основе `localName` типа QName.

В следующей таблице приведены сопоставления декодирования из пользовательских типов данных в типы данных ActionScript 3.0.

Пользовательский тип	Декодированный тип ActionScript 3.0	Примечания
Apache Map `http://xml.apache.org/xml-soap:Map`	Object	Представление SOAP `java.util.Map. Keys` должно быть приведено в форме строк.
Apache Rowset `http://xml.apache.org/xml-soap:Rowset`	Массив объектов	
ColdFusion QueryBean `http://rpc.xml.coldfusion:QueryBean`	Массив объектов Коллекция объектов `mx.collections.ArrayCollection`	Если `makeObjectsBindable` имеет значение true, являющийся результатом массив включается в ArrayCollection.

Поддержка элементов схемы XML

Структуры или атрибуты структуры из следующей схемы XML в версии Flex 4 реализованы лишь частично:

```
<choice>
<all>
<union
```

В версии Flex 4 игнорируются и не поддерживаются следующие структуры и атрибуты структуры схемы XML:

```
<attribute use="required"/>

<element
    substitutionGroup="..."
    unique="..."
    key="..."
    keyref="..."
    field="..."
    selector="..."/>

<simpleType>
    <restriction>
        <minExclusive>
        <minInclusive>
        <maxExclusiv>
        <maxInclusive>
        <totalDigits>
        <fractionDigits>
        <length>
        <minLength>
        <maxLength>
        <enumeration>
        <whiteSpace>
        <pattern>
    </restriction>
</simpleType>

<complexType
    final="..."
    block="..."
    mixed="..."
    abstract="..."/>

<any
processContents="..."/>

<annotation>
```

Настройка сопоставления типов в web-службе

При потреблении данных при вызове web-службы в среде Flex обычно создаются анонимные объекты ActionScript без объявления типов, копирующие структуру XML в теле сообщения SOAP. Если в программе необходимо создать экземпляр определенного класса, можно использовать объект mx.rpc.xml.SchemaTypeRegistry и зарегистрировать объект QName с соответствующим классом ActionScript.

В качестве примера предположим, что в файле User.as содержится следующее определение класса:

```
package
{
     public class User
     {
       public function User() {}

       public var firstName:String;
       public var lastName:String;
     }
}
```

Затем необходимо вызвать операцию `getUser` в web-службе, возвращающий следующий код XML:

```
<tns:getUserResponse xmlns:tns="http://example.uri">
     <tns:firstName>Ivan</tns:firstName>
     <tns:lastName>Petrov</tns:lastName>
</tns:getUserResponse>
```

Убедитесь, что при вызове операции getUser используется экземпляр класса User вместо общего Object; используйте в приложении следующий код ActionScript в методе:

```
SchemaTypeRegistry.getInstance().registerClass(new QName("http://example.uri",
"getUserResponse"), User);
```

Метод `SchemaTypeRegistry.getInstance()` является статическим методом, возвращающим экземпляр реестра типов, заданный по умолчанию. В большинстве случаев этого достаточно. Однако при этом выполняется регистрация данного объекта QName с тем же классом ActionScript во всех операциях web-службы в рамках приложения. Если для других операций должны быть зарегистрированы другие классы, необходимо вставить следующий код в приложение:

```
var qn:QName = new QName("http://the.same", "qname");
var typeReg1:SchemaTypeRegistry = new SchemaTypeRegistry();
var typeReg2:SchemaTypeRegistry = new SchemaTypeRegistry();
typeReg1.registerClass(qn, someClass);
myWS.someOperation.decoder.typeRegistry = typeReg1;

typeReg2.registerClass(qn, anotherClass);
myWS.anotherOperation.decoder.typeRegistry = typeReg2;
```

Использование пользовательской сериализации web-службы

Существуют два подхода к обеспечению полного управления сериализацией объектов ActionScript в XML-форму и к десериализации ответных XML-сообщений. Рекомендуемый подход предусматривает работу непосредственно с E4X.

При передаче экземпляра XML в операцию web-службы в качестве единственного параметра, он переносится в запрос сериализации в виде нижестоящего объекта узла `<SOAP:Body>` без обработки. Рекомендуется использовать эту стратегию при необходимости обеспечения полного управления сообщениями SOAP. Аналогичным образом, при десериализации ответа web-службы в качестве значения свойства `resultFormat` операции можно установить e4x. В результате в составе сообщения ответа будет возвращен объект XMLList с нижестоящими элементами узла `<SOAP:Body>`. На основе этого можно реализовать необходимую пользовательскую логику, позволяющую создать подходящие объекты ActionScript.

При использовании второго и более трудоемкого подхода необходимо предоставить собственные реализации элементов mx.rpc.soap.ISOAPDecoder и mx.rpc.soap.ISOAPEncoder. Например, если ранее был записан класс под названием MyDecoder, обеспечивавший реализацию ISOAPDecoder, в методе приложения содержится следующий код:

```
myWS.someOperation.decoder = new MyDecoder();
```

При вызове `someOperation` Flex вызывает метод `decodeResponse()` класса MyDecoder. После этого он обеспечивает пользовательскую реализацию для обработки всего сообщения SOAP и формирования ожидаемых объектов ActionScript.

Использование компонентов RemoteObject

Компонент RemoteObject Flex применяется для вызова методов в компоненте ColdFusion или классах Java.

Компоненты RemoteObject также могут использоваться с объектами PHP и .NET совместно с программным обеспечением сторонних поставщиков, например проектами с открытым исходным кодом AMFPHP, SabreAMF и WebORB от Midnight Coders. Дополнительную информацию см. на следующих web-сайтах:

- Zend Framework http://framework.zend.com/
- AMFPHP http://amfphp.sourceforge.net/
- SabreAMF http://www.osflash.org/sabreamf
- WebORB от Midnight Coders http://www.themidnightcoders.com/

Для отправки и получения данных компоненты RemoteObject используют протокол AMF, в то время как компоненты WebService и HTTPService используют протокол HTTP. AMF значительно превосходит HTTP по скорости, однако кодирование на стороне сервера и конфигурация обычно более сложны.

Flash Builder for PHP представляет собой инструмент разработки, созданный в партнерстве с компанией Zend Technologies и содержащий встроенную копию Zend Studio. Подробную информацию см. на веб-сайте Adobe.

Аналогично компонентам HTTPService и WebService, компонент RemoteObject используется для отображения результатов запроса в базу данных в приложении. Этот компонент можно также использовать для вставки, обновления и удаления данных в базе данных. После возвращения результата запроса в приложение можно отобразить его в одном или нескольких элементах управления пользовательского интерфейса.

Справочную информацию API о компоненте RemoteObject см. в разделе mx.rpc.remoting.mxml.RemoteObject.

Пример приложения RemoteObject

MXML-код

В приложении , приведенном в следующем примере, компонент RemoteObject применяется для вызова компонента ColdFusion. Компонент ColdFusion обеспечивает запрос к таблице базы данных MySQL под названием Users. Результат запроса возвращается в приложение , в котором привязывается к свойству `dataProvider` элемента управления DataGrid и отображается в элементе управления DataGrid. Приложение также осуществляет отправку имен и адресов электронной почты новых пользователей в компонент ColdFusion, обеспечивающий их добавление в таблицу пользователей базы данных.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600">
    <fx:Declarations>
        <mx:RemoteObject
            id="userRequest"
            destination="ColdFusion"
            source="flexapp.returnusers">

            <mx:method name="returnRecords" result="returnHandler(event)"
                       fault="mx.controls.Alert.show(event.fault.faultString)"/>
            <mx:method name="insertRecord" result="insertHandler()"
                       fault="mx.controls.Alert.show(event.fault.faultString)"/>
        </mx:RemoteObject>
    </fx:Declarations>

    <fx:Script>
        <![CDATA[
            import mx.rpc.events.ResultEvent;

            private function returnHandler(e:ResultEvent):void
            {
                dgUserRequest.dataProvider = e.result;
            }
            private function insertHandler():void
            {
                userRequest.returnRecords();
            }
            private function clickHandler():void
            {
                userRequest.insertRecord(username.text, emailaddress.text);
            }
        ]]>
    </fx:Script>

    <mx:Form x="22" y="10" width="300">
        <mx:FormItem>
            <s:Label text="Username" />
            <s:TextInput id="username"/>
        </mx:FormItem>
        <mx:FormItem>
            <s:Label text="Email Address" />
            <s:TextInput id="emailaddress"/>
        </mx:FormItem>
        <s:Button label="Submit" click="clickHandler()"/>
    </mx:Form>

    <mx:DataGrid id="dgUserRequest" x="22" y="200">
        <mx:columns>
            <mx:DataGridColumn headerText="User ID" dataField="userid"/>
            <mx:DataGridColumn headerText="User Name" dataField="username"/>
        </mx:columns>
    </mx:DataGrid>
</s:Application>
```

В этом приложении свойство `destination` компонента RemoteObject получает значение `Coldfusion`, а в качестве значения свойства `source` устанавливается полное имя компонента ColdFusion.

В отличие от предыдущего примера при работе с ADEP Data Services или BlazeDS полное имя класса определяется в свойстве `source` адреса назначения удаленной службы в конфигурационном файле, которым по умолчанию является файл remoting-config.xml. Имя адреса назначения указывается в свойстве `destination` компонента RemoteObject. Целевой класс также должен содержать конструктор без аргументов. При работе с ColdFusion вместо использования свойства `source` в компоненте RemoteObject адрес назначения можно настроить с помощью этого метода.

Компонент ColdFusion

В приложении вызывается следующий компонент ColdFusion. Приведенный код ColdFusion обеспечивает добавление информации и отправку запросов к базе данных SQL, а также возвращение результатов запросов в приложение На странице ColdFusion тег `cfquery` используется для добавления данных в базу данных и отправки запроса к базе данных, а тег `cfreturn` применяется в целях форматирования результатов запросов в качестве объектов запросов ColdFusion.

```
<cfcomponent name="returnusers">
    <cffunction name="returnRecords" access="remote" returnType="query">

        <cfquery name="alluserinfo" datasource="flexcf">
                SELECT userid, username, emailaddress FROM users
        </cfquery>
        <cfreturn alluserinfo>
    </cffunction>
    <cffunction name="insertRecord" access="remote" returnType="void">

        <cfargument name="username" required="true" type="string">
        <cfargument name="emailaddress" required="true" type="string">
        <cfquery name="addempinfo" datasource="flexcf">
            INSERT INTO users (username, emailaddress) VALUES (
            <cfqueryparam value="#arguments.username#" cfsqltype="CF_SQL_VARCHAR"
maxlength="255">,
            <cfqueryparam value="#arguments.emailaddress#" cfsqltype="CF_SQL_VARCHAR"
maxlength="255"> )
        </cfquery>
        <cfreturn>
    </cffunction>
</cfcomponent>
```

Вызов компонентов RemoteObject в сценарии ActionScript

В следующем примере сценария ActionScript при вызове метода `useRemoteObject()` обеспечивается объявление службы, установка адреса назначения, настройка прослушивателей события ошибки и результата события, а также вызывается метод службы `getList()`.

```
<?xml version="1.0"?>
<!-- fds\rpc\ROInAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.rpc.remoting.RemoteObject;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;

            [Bindable]
            public var empList:Object;
            public var employeeRO:RemoteObject;

            public function useRemoteObject(intArg:int, strArg:String):void {
                employeeRO = new RemoteObject();
                employeeRO.destination = "SalaryManager";
                employeeRO.getList.addEventListener("result", getListResultHandler);
                employeeRO.addEventListener("fault", faultHandler);
                employeeRO.getList(deptComboBox.selectedItem.data);
            }

            public function getListResultHandler(event:ResultEvent):void {
                 // Do something
                empList=event.result;
            }

            public function faultHandler (event:FaultEvent):void {
             // Deal with event.fault.faultString, etc.
                Alert.show(event.fault.faultString, 'Error');
            }
        ]]>
    </mx:Script>
    <mx:ComboBox id="deptComboBox"/>
</mx:Application>
```

Получение доступа к объектам Java в исходном пути

Компонент RemoteObject позволяет получать доступ к объектам Java, хранящим и не хранящим информацию о состоянии, находящимся в исходном пути web-приложений ADEP Data Services, BlazeDS или ColdFusion. Для добавления автономных файлов классов в исходный путь их можно сохранить в каталоге WEB-INF/classes web-приложения. Классы, содержащиеся в файлах Java Archive (JAR) можно поместить в каталог WEB-INF/lib web-приложения, а затем добавить в исходный путь. Полное имя класса указывается в свойстве `source` адреса назначения удаленной службы в конфигурационном файле службы services-config.xml ADEP Data Services, BlazeDS или ColdFusion, либо в файле, включаемом по ссылке, например, в файле remoting-config.xml. Класс также должен содержать конструктор без аргументов. При работе с ColdFusion можно дополнительно установить в свойстве `destination` компонента RemoteObject значение `Coldfusion`, а в свойстве `source` указать полное имя компонента ColdFusion или класса Java.

При настройке адреса назначения удаленной службы для получения доступа к объектам без состояния (область запроса) в программе Flex для каждого вызова метода создается новый объект вместо вызова методов для одного и того же объекта. Область объекта может быть настроена как область запроса (значение по умолчанию), область приложения или область сеанса. Объекты из области приложения доступны для web-приложения, содержащего объект. Объекты из области сеанса доступны для всего сеанса клиента.

При конфигурировании адреса назначения удаленного объекта для получения доступа к объектам, хранящим информацию о состоянии, в среде Flex объект на сервере создается только один раз и его состояние поддерживается в актуальном состоянии между вызовами методов. Если хранение объекта в приложении или области сеанса приводит к низкой производительности памяти, используйте область запроса.

Получение доступа к EJB и другим объектам в JNDI

Доступ к Enterprise JavaBeans (EJB) и другим объектам, сохраненным в интерфейсе для служб каталогов и именования (JNDI), осуществляется путем вызова методов для адреса назначения, являющегося классом фасада службы и обеспечивающего поиск объекта в JNDI и вызов его методов.

Для вызова методов Enterprise JavaBeans и других объектов, использующих JNDI, можно применять объекты, хранящие и не хранящие информацию о состоянии. В EJB возможен вызов класса фасада службы, возвращающего объект EJB из JNDI и вызывающего метод для EJB.

В классе Java применяется стандартный стиль программирования Java, в котором создается первоначальный контекст и выполняется поиск JNDI. Для EJB также используется стандартный стиль программирования, при котором класс содержит методы, обеспечивающие вызов метода `create()` домашнего объекта EJB и результирующих бизнес-методов EJB.

В следующем примере для адреса класса фасада используется метод с именем `getHelloData()`:

```
<mx:RemoteObject id="Hello" destination="roDest">
    <mx:method name="getHelloData"/>
</mx:RemoteObject>
```

На стороне Java метод `getHelloData()` позволяет инкапсулировать все необходимое для вызова бизнес-метода в EJB. Метод Java, приведенный в следующем примере, обеспечивает выполнение следующих действий:

- создание нового первоначального контекста для вызова EJB;
- поиск JNDI, результатом которого является получение домашнего объекта EJB;
- вызов метода `create()` домашнего объекта EJB;
- вызов метода `sayHello()` EJB.

```
...
public void getHelloData() {
    try{
        InitialContext ctx = new InitialContext();
        Object obj = ctx.lookup("/Hello");
        HelloHome ejbHome = (HelloHome)
        PortableRemoteObject.narrow(obj, HelloHome.class);
        HelloObject ejbObject = ejbHome.create();
        String message = ejbObject.sayHello();
        }
    catch (Exception e);
    }
...
```

Зарезервированные имена методов

В удаленной библиотеке Flex применяются следующие имена методов; не используйте их в качестве имен собственных методов:

```
addHeader()
addProperty()
deleteHeader()
hasOwnProperty()
isPropertyEnumerable()
isPrototypeOf()
registerClass()
toLocaleString()
toString()
unwatch()
valueOf()
watch()
```

Имена методов также не должны начинаться с символа подчеркивания (_).

Для доступа к методам RemoteObject (операциям) обычно требуется указать их имена после переменной службы. Однако, если имя операции будет совпадать с определенным методом службы, может возникнуть конфликт имен. Для возвращения операции, соответствующей заданному имени, в сценарии ActionScript в компоненте RemoteObject можно использовать следующий метод:

```
public function getOperation(name:String):Operation
```

Сериализация между ActionScript и Java

ADEP Data Services и BlazeDS сериализуют данные между ActionScript (AMF 3), Java и типами данных ColdFusion в обоих направлениях. Для получения информации о типах данных ColdFusion см. набор документов по ColdFusion.

Преобразование данных из ActionScript в Java

При передаче данных из приложения в объект Java в параметрах метода данные автоматически преобразуются из типа данных ActionScript в тип данных Java. При поиске подходящего метода в объекте Java, выполняемого в ADEP Data Services или BlazeDS, для нахождения соответствия применяются более простые преобразования.

Простые типы данных на стороне клиента, такие как логические и строковые значения, обычно совпадают с удаленным API. Однако при поиске подходящего метода в объекте Java в среде Flex производятся некоторые простые преобразования.

В массиве ActionScript элементы могут индексироваться двумя способами. В *строгом массиве* все индексы являются числами. В *ассоциативном массиве* по крайней мере один индекс основан на значениях типа String. Информация о том, массив какого типа передается на сервер, имеет большое значение, поскольку в результате этого изменяется тип данных параметров, используемых для вызова метода в объекте Java. *Плотным массивом* является массив, в котором все числовые индексы последовательны, начинаются с 0 (нуля) и между ними отсутствуют промежутки. В *разреженном массиве* между числовыми индексами существуют промежутки; массив обрабатывается как объект, а числовые индексы становятся свойствами, подлежащими десериализации в объекте java.util.Map во избежание отправки множества нулевых значений.

В следующей таблице перечислены поддерживаемые преобразования ActionScript (AMF 3) в Java для простых типов данных.

Тип ActionScript (AMF 3)	**Десериализация для Java**	**Поддерживаемая привязка типов в Java**
Array (плотный)	java.util.List	java.util.Collection, *Object*[ ] (собственный массив) Если тип является интерфейсом, он сопоставляется со следующими реализациями : • List становится ArrayList • SortedSet становится TreeSet • Set становится HashSet • Collection становится ArrayList Новый экземпляр пользовательской реализации Collection привязывается к этому типу.
Array (разреженный)	java.util.Map	java.util.Map
Boolean Строка из значений `true` или `false`	java.lang.Boolean	Boolean, boolean, String
flash.utils.ByteArray	byte []	
flash.utils.IExternalizable	java.io.Externalizable	
Date	java.util.Date (отформатирован в соответствии со скоординированным универсальным временем (UTC))	java.util.Date, java.util.Calendar, java.sql.Timestamp, java.sql.Time, java.sql.Date
int/uint	java.lang.Integer	java.lang.Double, java.lang.Long, java.lang.Float, java.lang.Integer, java.lang.Short, java.lang.Byte, java.math.BigDecimal, java.math.BigInteger, String, элементарные типы double, long, float, int, short, byte
null	null	primitives
Number	java.lang.Double	java.lang.Double, java.lang.Long, java.lang.Float, java.lang.Integer, java.lang.Short, java.lang.Byte, java.math.BigDecimal, java.math.BigInteger, String, 0 (нуль) при передаче значения `null`, элементарные типы double, long, float, int, short, byte
Object (общий)	java.util.Map	Если указан интерфейс Map, создает java.util.HashMap для java.util.Map и новый java.util.TreeMap для java.util.SortedMap.
String	java.lang.String	java.lang.String, java.lang.Boolean, java.lang.Number, java.math.BigInteger, java.math.BigDecimal, char[], любой элементарный числовой тип
Object определенного типа	Object определенного типа При использовании тега метаданных `[RemoteClass]`, определяющего имя удаленного класса. Тип Bean должен содержать общедоступный конструктор без аргументов.	Объект определенного типа

Тип ActionScript (AMF 3)	Десериализация для Java	Поддерживаемая привязка типов в Java
undefined	null	`null` для объекта, значения по умолчанию для элементарных типов
XML.	org.w3c.dom.Document	org.w3c.dom.Document
XMLDocument (устаревший XML-тип)	org.w3c.dom.Document	org.w3c.dom.Document Поддержка устаревших типов XML для типа XMLDocument может быть включена для любого канала, определенного в файле services-config.xml. Этот параметр имеет значение только для обратной передачи данных с сервера на сторону клиента; он обеспечивает управление отправкой экземпляров org.w3c.dom.Document в ActionScript. Для получения дополнительной информации см. раздел Настройка сериализации AMF в канале.

Значения элементарных типов в Java не могут равняться `null`. При передаче значений Boolean и Number от клиента в объект Java в программе Flex значения `null` интерпретируются как значения элементарных типов, заданные по умолчанию; например, 0 для типов double, float, long, int, short, byte, \u0000 для char и `false` для Boolean. Значения по умолчанию получают только элементарные типы Java.

Объекты java.lang.Throwable в ADEP Data Services и BlazeDS обрабатываются как любые другие типизированные объекты. Они обрабатываются с использованием правил, обеспечивающих поиск общедоступных полей и свойств bean, а типизированные объекты возвращаются клиенту. Эти правила схожи с обычными правилами bean за исключением того, что они обеспечивают поиск получателей свойств, доступных только для чтения. Это позволяет получать больше информации на основании данных исключения Java. Если для объектов Throwable требуется устаревшее поведение, в свойстве `legacy-throwable` можно установить значение `true` для канала; дополнительную информацию см. в разделе Настройка сериализации AMF в канале.

Строгие массивы могут передаваться в качестве параметров для методов, в которых ожидается реализация API java.util.Collection или собственного массива Java.

Java Collection может содержать любое число типов Object, в то время как в массиве Java Array значения должны иметь один и тот же тип (например, java.lang.Object[ ] и int[ ]).

В ADEP Data Services и BlazeDS также осуществляется преобразование строгих массивов ActionScript в соответствующие реализации общих API-интерфейсов Collection. Например, в случае передачи строгого массива ActionScript в метод `public void addProducts(java.util.Set products)` объекта Java перед его отправкой в качестве параметра в ADEP Data Services и BlazeDS он преобразуется в экземпляр java.util.HashSet, поскольку HashSet является подходящей реализацией интерфейса java.util.Set. Аналогично, ADEP Data Services и BlazeDS обеспечивают передачу экземпляра java.util.TreeSet для параметров, типизированных с помощью интерфейса java.util.SortedSet.

ADEP Data Services и BlazeDS передают экземпляр java.util.ArrayList в параметры, типизированные с помощью интерфейса java.util.List и любого другого интерфейса, расширяющего возможности java.util.Collection. Затем эти типы возвращаются клиенту в виде экземпляров mx.collections.ArrayCollection. Если обычные массивы ActionScript должны быть возвращены клиенту, для элемента `legacy-collection` необходимо установить значение `true` в разделе `serialization` свойств определения канала. Для получения дополнительной информации см. раздел Настройка сериализации AMF в канале.

Явное сопоставление объектов ActionScript и Java

Для объектов Java, неявная обработка которых в ADEP Data Services и BlazeDS не осуществляется, значения, найденные в общедоступных свойствах Bean с помощью методов Get/Set, и общедоступные переменные передаются клиенту в виде свойств для объекта. Частные свойства, константы, статические свойства, свойства, доступные только для чтения, и т.д. не подлежат сериализации. Общедоступные свойства объектов ActionScript, определенные с помощью средств доступа Get/Set и публичных переменных, передаются на сервер.

В ADEP Data Services и BlazeDS используется стандартный класс Java, java.beans.Introspector, позволяющий получить дескрипторы свойства для класса Java Bean. В этой системе также применяется отражения для сбора общедоступных полей в классе. Свойства Bean являются более приоритетными по отношению к полям. Имена свойств в Java и ActionScript должны совпадать. Собственный код Flash Player обеспечивает определения способа, в соответствии с которым на стороне клиента выполняется внутренний анализ классов ActionScript.

В классе ActionScript тег метаданных `[RemoteClass(alias=" ")]` используется для создания объекта ActionScript, сопоставляемого непосредственно с объектом Java. Класс ActionScript, в который преобразуются данные, необходимо использовать или определить в MXML-файле, чтобы обеспечить связь с файлом SWF и доступность во время выполнения. Эффективным способом является приведение объекта результата, что показано в следующем примере:

```
var result:MyClass = MyClass(event.result);
```

В самом классе должны использоваться ссылки со строгим контролем типов, что обеспечит связь его зависимостей.

Исходный код для класса ActionScript, в котором применяется тег метаданных `[RemoteClass(alias=" ")]` показан в следующем примере:

```
package samples.contact {
    [Bindable]
    [RemoteClass(alias="samples.contact.Contact")]
    public class Contact {
        public var contactId:int;

        public var firstName:String;

        public var lastName:String;

        public var address:String;

        public var city:String;

        public var state:String;

        public var zip:String;
    }
}
```

Тег метаданных `[RemoteClass]` может применяться без псевдонима, если не осуществляется сопоставление с объектом Java на стороне сервере, но при этом тип объекта действительно возвращается с сервера. Объект ActionScript подлежит преобразованию в специальный объект Map при его отправке на сервер, однако объектом, возвращенным на сторону клиента сервером, будет являться первоначальный тип ActionScript.

Для предотвращения отправки определенного свойства из класса ActionScript на сервер используйте тег метаданных `[Transient]` выше объявления этого свойства в классе ActionScript.

Преобразование данных из Java в ActionScript

Объект, возвращенный из метода Java, преобразуется из Java в ActionScript. В ADEP Data Services и BlazeDS также выполняется обработка объектов, найденных в составе объектов. В ADEP Data Services неявным образом обрабатываются типы данных Java, приведенные в следующей таблице.

Тип Java	**Тип ActionScript (AMF 3)**
java.lang.String	String
java.lang.Boolean, boolean	Boolean
java.lang.Integer, int	int При значении < 0xF0000000 || value > 0x0FFFFFFF значение становится числовым в соответствии с требованиями кодирования AMF.
java.lang.Short, short	int При i < 0xF0000000 || i > 0x0FFFFFFF значение становится числовым.
java.lang.Byte, byte[]	int При i < 0xF0000000 || i > 0x0FFFFFFF значение становится числовым.
java.lang.Byte[]	flash.utils.ByteArray
java.lang.Double, double	Number
java.lang.Long, long	Number
java.lang.Float, float	Number
java.lang.Character, char	String
java.lang.Character[], char[]	String
java. math.BigInteger	String
java.math.BigDecimal	String
java.util.Calendar	Date Передаваемые даты соответствуют часовому поясу универсального глобального времени. Клиенты и серверы должны обеспечить корректировку времени в соответствии с часовыми поясами.
java.util.Date	Date Передаваемые даты соответствуют часовому поясу UTC. Клиенты и серверы должны обеспечить корректировку времени в соответствии с часовыми поясами.
java.util.Collection (например, java.util.ArrayList)	mx.collections.ArrayCollection
java.lang.Object[]	Array
java.util.Map	Object (без определенного типа). Например, java.util.Map[] преобразуется в массив (из объектов).
java.util.Dictionary	Object (без определенного типа)

Тип Java	**Тип ActionScript (AMF 3)**
org.w3c.dom.Document	XML-объект
null	null
java.lang.Object (отличный от ранее перечисленных типов)	Object определенного типа Сериализация объектов выполняется в соответствии с правилами внутреннего анализа Java Bean, причем объекты также имеют общедоступные поля. Поля, являющиеся статическими, промежуточными или непубличными, а также непубличные или статические свойства bean исключаются.

Настройка сериализации AMF в канале

Пользователи обладают возможностями активировать поддержку сериализации устаревших типов AMF, используемой в более ранних версиях Flex и настраивать другие свойства сериализации в определениях каналов, содержащихся в файле services-config.xml.

В следующей таблице описываются свойства, которые можно установить в элементе `<serialization>` определения канала:

Свойство	**Описание**
`<ignore-property-errors>true</ignore-property-errors>`	Значением по умолчанию является `true`. Определяет, необходимо ли выдавать ошибку в случае, если входящий объект клиента имеет непредвиденные свойства, которые не могут быть установлены в объекте сервера, в конечной точке.
`<log-property-errors>false</log-property-errors>`	Значением по умолчанию является `false`. При значении `true` выполняется регистрация непредвиденных ошибок свойств.
`<legacy-collection>false</legacy-collection>`	Значением по умолчанию является `false`. При значении `true` экземпляры java.util.Collection возвращаются в виде массивов ActionScript. При значении `false` экземпляры java.util.Collection возвращаются в виде mx.collections.ArrayCollection.
`<legacy-map>false</legacy-map>`	Значением по умолчанию является `false`. При значении `true` экземпляры java.util.Map сериализуются как массив ECMA или ассоциативный массив, а не анонимный объект.
`<legacy-xml>false</legacy-xml>`	Значением по умолчанию является `false`. При значении `true`, org.w3c.dom.Document сериализуются как экземпляры flash.xml.XMLDocument, а не встроенные XML-экземпляры (с поддержкой E4X).
`<legacy-throwable>false</legacy-throwable>`	Значением по умолчанию является `false`. При значении `true` экземпляры java.lang.Throwable сериализуются как объекты информации статуса AMF (вместо обычной сериализации bean, включая свойства, доступные только для чтения).

Свойство	Описание
`<type-marshaller>className</type-marshaller>`	Указывает реализацию flex.messaging.io.TypeMarshaller, обеспечивающей преобразование объекта в экземпляр требуемого класса. Используется при вызове метода Java или заполнении экземпляра Java, когда тип входного объекта десериализации (например, десериализация анонимного объекта ActionScript всегда выполняется как java.util.HashMap) не соответствует интерфейсу API адресата (например, java.util.SortedMap). Таким образом, тип можно преобразовать в требуемый тип.
`<restore-references>false</restore-references>`	Значением по умолчанию является `false`. Расширенный переключатель, вызывающий отслеживание средством десериализации ссылок на объект при необходимости выполнения преобразования типа; например, при передаче анонимного объекта для свойства java.util.SortedMap сначала для объекта как обычно выполняется десериализация в java.util.Map, а затем он преобразуется в подходящую реализацию SortedMap (такую как java.util.TreeMap). Если на один и тот же анонимный объект в графе объекта указывают другие объекты, то это свойство позволяет восстановить соответствующие ссылки вместо создания нескольких реализаций SortedMap. Следует отметить, что при указании для этого свойства значения `true` может значительно снизиться производительность в случае большого количества данных.
`<instantiate-types>true</instantiate-types>`	Значением по умолчанию является `true`. Расширенный переключатель при указанном значении `false` прекращает создание средством десериализации экземпляров объектов со строгим контролем типов, вместо этого выполняется сохранение информации о типе и десериализация необработанных свойств в реализации Map, в частности flex.messaging.io.ASObject. Следует отметить, что для любых классов в пакете flex. * экземпляры создаются всегда.

Использование пользовательской сериализации

Если стандартные механизмы сериализации и десериализации данных между ActionScript на стороне клиента и Java на стороне сервера не соответствуют требованиям, можно создать собственную схему сериализации. На стороне клиента реализуется интерфейс flash.utils.IExternalizable на основе ActionScript, и на стороне сервера реализуется соответствующий интерфейс java.io.Externalizable на основе Java.

Основанием для пользовательской сериализации может служить необходимость предотвращения переноса по сетевому уровню всех свойств представления объекта либо на стороне клиента, либо на стороне сервера. При реализации пользовательской сериализации можно таким образом создать код для классов, что определенные свойства только для клиента или только для сервера не будут передаваться по сети. При использовании стандартной схемы сериализации все общедоступные свойства передаются между клиентом и сервером в обоих направлениях.

На стороне клиента идентификатор класса, реализующего интерфейс flash.utils.Iexternalizable, записывается в поток сериализации. Класс выполняет сериализацию и восстанавливает состояние своих экземпляров. Класс реализует методы `writeExternal()` и `readExternal()` интерфейса IExternalizable для обеспечения управления содержимым и форматом потока сериализации, но не именем класса или типом для объекта и его супертипов. Эти методы заменяют исходное поведение сериализации AMF. Для сохранения состояния класса эти методы должны быть симметричны аналогичным удаленным методам.

На стороне сервера класс Java, реализующий интерфейс java.io.Externalizable, предоставляет функциональные возможности, эквивалентные возможностям класса ActionScript, реализующего интерфейс flash.utils.IExternalizable.

Примечание. *Если требуется точная сериализация по ссылке, не используйте типы, реализующие интерфейс IExternalizable с HTTPChannel. При этом ссылки между возвращающимися объектами теряются и, по-видимому, клонируются в конечной точке.*

В следующем примере приведен полный исходный код класса Product для версии клиента (ActionScript), выполняющий отображение в класс Product на основе Java для сервера. Класс Product клиента реализует интерфейс IExternalizable, а класс Product сервера реализует интерфейс Externalizable.

```
// Product.as
package samples.externalizable {

import flash.utils.IExternalizable;
import flash.utils.IDataInput;
import flash.utils.IDataOutput;

[RemoteClass(alias="samples.externalizable.Product")]
public class Product implements IExternalizable {
    public function Product(name:String=null) {
        this.name = name;
    }

    public var id:int;
    public var name:String;
    public var properties:Object;
    public var price:Number;

    public function readExternal(input:IDataInput):void {
        name = input.readObject() as String;
        properties = input.readObject();
        price = input.readFloat();
    }

    public function writeExternal(output:IDataOutput):void {
        output.writeObject(name);
        output.writeObject(properties);
        output.writeFloat(price);
    }
}
}
```

В классе Product клиента используется два вида сериализации: стандартная сериализация, совместимая с интерфейсом java.io.Externalizable и сериализация AMF 3. В следующем примере показан метод `writeExternal()` класса Product клиента, использующий оба типа сериализации:

```
public function writeExternal(output:IDataOutput):void {
    output.writeObject(name);
    output.writeObject(properties);
    output.writeFloat(price);
}
```

Как показано в следующих примерах, метод `writeExternal()` класса Product сервера почти идентичен версии этого метода для клиента:

```
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeObject(name);
    out.writeObject(properties);
    out.writeFloat(price);
}
```

В методе `writeExternal()` класса Product клиента метод `flash.utils.IDataOutput.writeFloat()` является примером стандартного метода сериализации, соответствующего спецификациям методов Java `java.io.DataInput.readFloat()` для работы с примитивными типами. Этот метод отправляет свойство `price` типа Float классу Product сервера.

Примером сериализации AMF 3 в методе `writeExternal()` класса Product клиента является вызов метода `flash.utils.IDataOutput.writeObject()`, соответствующий вызову метода `java.io.ObjectInput.readObject()` в методе `readExternal()` класса Product сервера. Метод `flash.utils.IDataOutput.writeObject()` отправляет классу Product сервера свойство `properties` типа Object и свойство name типа String. Это возможно благодаря реализации в конечной точке AMFChannel интерфейса java.io.ObjectInput, ожидающего, что отправляемые из метода `writeObject()` данные имеют формат AMF 3.

В свою очередь, вызов метода `readObject()` в методе `readExternal()` класса Product сервера использует десериализацию AMF 3, поэтому предполагается, что версия ActionScript значения `properties` имеет тип Map, а значение `name` имеет тип String.

В следующем примере приведен полный исходный код класса Product для сервера:

```
// Product.java
package samples.externalizable;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.Map;

/**
* This Externalizable class requires that clients sending and
* receiving instances of this type adhere to the data format
* required for serialization.
*/
public class Product implements Externalizable {
    private String inventoryId;
    public String name;
    public Map properties;
    public float price;

    public Product()
    {
    }

        /**
        * Local identity used to track third-party inventory. This property is
        * not sent to the client because it is server specific.
        * The identity must start with an 'X'.
        */
        public String getInventoryId() {
            return inventoryId;
        }
```

```
        public void setInventoryId(String inventoryId) {
            if (inventoryId != null && inventoryId.startsWith("X"))
            {
                this.inventoryId = inventoryId;
            }
            else
            {
                throw new IllegalArgumentException("3rd party product
                inventory identities must start with 'X'");
            }
        }

        /**
         * Deserializes the client state of an instance of ThirdPartyProxy
         * by reading in String for the name, a Map of properties
         * for the description, and
         * a floating point integer (single precision) for the price.
         */
        public void readExternal(ObjectInput in) throws IOException,
            ClassNotFoundException {
            // Read in the server properties from the client representation.
            name = (String)in.readObject();
            properties = (Map)in.readObject();
            price = in.readFloat();
            setInventoryId(lookupInventoryId(name, price));
        }

        /**
         * Serializes the server state of an instance of ThirdPartyProxy
         * by sending a String for the name, a Map of properties
         * String for the description, and a floating point
         * integer (single precision) for the price. Notice that the inventory
         * identifier is not sent to external clients.
         */
        public void writeExternal(ObjectOutput out) throws IOException {
            // Write out the client properties from the server representation.
            out.writeObject(name);
            out.writeObject(properties);
            out.writeFloat(price);
        }

        private static String lookupInventoryId(String name, float price) {
            String inventoryId = "X" + name + Math.rint(price);
            return inventoryId;
        }
}
```

В следующем примере показан метод `readExternal()` класса Product сервера:

```
public void readExternal(ObjectInput in) throws IOException,
    ClassNotFoundException {
    // Read in the server properties from the client representation.
    name = (String)in.readObject();
    properties = (Map)in.readObject();
    price = in.readFloat();
    setInventoryId(lookupInventoryId(name, price));
    }
```

Метод `writeExternal()` класса Product клиента в процессе сериализации не отправляет серверу свойство `id`, так как оно не соответствует версии объекта Product на сервере. Аналогично, метод `writeExternal()` класса Product сервера не отправляет клиенту свойство `inventoryId`, поскольку это свойство специфично для сервера.

Следует отметить, что имена свойств класса Product во время сериализации не отправляются ни в одном из направлений. Поскольку состояние класса является фиксированным и управляемым, свойства передаются без указания имен в четко определенном порядке, и считываются методом `readExternal()` в соответствующем порядке.

Явная передача параметров и привязка параметров

Существует два способа вызова компонентов HTTPService, WebService и RemoteObject: явная передача параметров и привязка параметров. При явной передаче параметров входные данные для службы предоставляются в форме параметров функции ActionScript. Этот способ выполнения вызова напоминает способ вызова методов в Java. Совместно с явной передачей параметров невозможно автоматическое использование средств проверки данных Flex.

Привязка параметров позволяет копировать данные в требуемые параметры из элементов управления пользовательского интерфейса или моделей. Привязка параметров доступна только для компонентов доступа к данным, объявленным в MXML. Перед передачей запросов службам можно к значениям параметров применить средства проверки. Дополнительную информацию о привязке данных и моделях данных см. в разделах Привязка данных и Хранение данных. Дополнительную информацию о проверке данных см. в разделе Проверка данных.

При использовании привязки данных объявляются теги параметров метода RemoteObject, вложенные в тег `<mx:arguments>` после тега `<mx:method>`, теги параметра HTTPService, вложенные в тег `<mx:request>`, или теги параметров операции WebService, вложенные в тег `<mx:request>` после тега `<mx:operation>`. Для отправки запроса используется метод `send()`.

Явная передача параметров для компонентов RemoteObject и WebService

Способы явной передачи параметров для компонентов RemoteObject и WebService аналогичны. В следующем примере приведен код MXML для объявления компонента RemoteObject и вызова службы при использовании явной передачи параметров в средстве прослушивания события Click элемента управления Button. Данные службе предоставляются элементом управления ComboBox. События результата и ошибки на уровне службы обрабатываются простыми средствами прослушивания событий.

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCParamPassing.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            [Bindable]
            public var empList:Object;
        ]]>
    </mx:Script>

    <mx:RemoteObject
        id="employeeRO"
        destination="SalaryManager"
        result="empList=event.result"
        fault="Alert.show(event.fault.faultString, 'Error');"/>

    <mx:ComboBox id="dept" width="150">
        <mx:dataProvider>
            <mx:ArrayCollection>
                <mx:source>
                    <mx:Object label="Engineering" data="ENG"/>
                    <mx:Object label="Product Management" data="PM"/>
                    <mx:Object label="Marketing" data="MKT"/>
                </mx:source>
            </mx:ArrayCollection>
        </mx:dataProvider>
    </mx:ComboBox>

   <mx:Button label="Get Employee List" click="employeeRO.getList(dept.selectedItem.data);"/>
</mx:Application>
```

Явная передача параметров для компонентов HTTPService

Явная передача параметров для компонентов HTTPService и компонентов RemoteObject и WebService выполняется различными способами. Для вызова службы всегда используется метод `send()` компонента HTTPService. В этом состоит отличие для компонентов RemoteObject и WebService, в которых вызываются методы, являющиеся версиями методов или операций службы RPC на стороне клиента.

При использовании явной передачи параметров в качестве параметра метода `send()` можно указать объект, содержащий пары «имя-значение». Параметр метода `send()` должен быть простого исходного типа; использование сложных вложенных объектов недопустимо, поскольку отсутствует стандартный способ их преобразования в пары «имя-значение».

Если параметр в вызове метода `send()` не указан, компонент HTTPService использует любой параметр запроса, указанный в теге `<mx:request>`.

В следующих примерах продемонстрированы два способа выполнения вызова службы HTTP при использовании метода `send()` с параметром. Во втором примере также показан вызов метода `cancel()` для отмены вызова службы HTTP.

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCSend.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function callService():void {
                // Cancel all previous pending calls.
                myService.cancel();

                var params:Object = new Object();
                params.param1 = 'val1';
                myService.send(params);
            }
        ]]>
    </mx:Script>

    <mx:HTTPService
        id="myService"
        destination="Dest"
        useProxy="true"/>
<!-- HTTP service call with a send() method that takes a variable as its parameter. The value
of the variable is an Object. -->
    <mx:Button click="myService.send({param1: 'val1'});"/>

<!-- HTTP service call with an object as a send() method parameter that provides query
parameters. -->
    <mx:Button click="callService();"/>
</mx:Application>
```

Привязка параметров для компонентов RemoteObject

При использовании привязки параметров для компонентов RemoteObject методы всегда объявляются в теге `<mx:method>` компонента RemoteObject.

Тег `<mx:method>` может содержать тег `<mx:arguments>`, имеющий в своем составе нижестоящие теги для параметров метода. Свойство имени тега `<mx:method>` должно соответствовать имени одного из методов службы. Порядок тегов аргументов должен соответствовать порядку параметров метода службы. Имена тегов аргументов могут максимально соответствовать фактическим именам соответствующих параметров метода, но это не обязательно.

Примечание. *При наличии одинаковых имен тегов аргументов в теге `<mx:arguments>` вызов службы завершается сбоем, если для удаленного метода массив является непредвиденным единственным входным источником. При компиляции приложения предупреждение об этом отсутствует.*

Данные можно привязывать к параметрам метода компонента RemoteObject. Для привязки и проверки данных указываются имена тегов параметров.

В следующем примере показан метод с двумя параметрами, привязанными к текстовым свойствам элементов управления TextInput. Средство проверки PhoneNumberValidator присваивается имени `arg1`, которое является именем тега первого аргумента.

```
<?xml version="1.0"?>
<!-- fds\rpc\ROParamBind1.mxml. -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:RemoteObject
        id="ro"
        destination="roDest">

        <mx:method name="setData">
            <mx:arguments>
                <arg1>{text1.text}</arg1>
                <arg2>{text2.text}</arg2>
            </mx:arguments>
        </mx:method>
    </mx:RemoteObject>
    <mx:TextInput id="text1"/>
    <mx:TextInput id="text2"/>

    <mx:PhoneNumberValidator source="{ro.setData.arguments}" property="arg1"/>
</mx:Application>
```

Flex отправляет значения тегов аргументов методу в том порядке, в котором теги MXML определены.

В следующем примере используется привязка параметров в теге `<mx:method>` компонента RemoteObject для привязки данных выбранного элемента ComboBox к операции `employeeRO.getList` при нажатии пользователем элемента управления Button. При использовании привязки параметров вызов службы осуществляется с помощью метода `send()` без параметров.

```
<?xml version="1.0"?>
<!-- fds\rpc\ROParamBind2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
                import mx.utils.ArrayUtil;
        ]]>
    </mx:Script>
    <mx:RemoteObject
        id="employeeRO"
        destination="roDest"
        showBusyCursor="true"
        fault="Alert.show(event.fault.faultString, 'Error');">
        <mx:method name="getList">
            <mx:arguments>
                <deptId>{dept.selectedItem.data}</deptId>
            </mx:arguments>
        </mx:method>
    </mx:RemoteObject>
    <mx:ArrayCollection id="employeeAC"
        source="{ArrayUtil.toArray(employeeRO.getList.lastResult)}"/>

    <mx:HBox>
        <mx:Label text="Select a department:"/>
        <mx:ComboBox id="dept" width="150">

            <mx:dataProvider>
```

```
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object label="Engineering" data="ENG"/>
                        <mx:Object label="Product Management" data="PM"/>
                        <mx:Object label="Marketing" data="MKT"/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ComboBox>
            <mx:Button label="Get Employee List"
                click="employeeRO.getList.send()"/>
    </mx:HBox>
    <mx:DataGrid dataProvider="{employeeAC}" width="100%">
        <mx:columns>
            <mx:DataGridColumn dataField="name" headerText="Name"/>
            <mx:DataGridColumn dataField="phone" headerText="Phone"/>
            <mx:DataGridColumn dataField="email" headerText="Email"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

Если неизвестно, содержит ли результат вызова службы массив или отдельный объект, можно использовать метод `toArray()` класса mx.utils.ArrayUtil для преобразования этого результата в массив, как показано в приведенном выше примере. Если методу `toArray()` передается отдельный объект, то метод возвращает массив с данным объектом в качестве единственного элемента. Если методу передается массив, то метод возвращает этот же самый массив. Дополнительную информацию о работе с объектами ArrayCollection см. в разделе Поставщики данных и коллекции.

Привязка параметров для компонентов HTTPService

Когда служба HTTP принимает параметры запроса, их можно объявить как нижестоящие теги тега `<mx:request>`. Имена тегов должны соответствовать именам параметров запроса, ожидаемым службой.

В следующем примере используется привязка параметров в теге `<mx:request>` компонента HTTPService для привязки данных выбранного элемента ComboBox к запросу `employeeSrv` при нажатии пользователем элемента управления Button. При использовании привязки параметров вызов службы осуществляется с помощью метода `send()` без параметров. В этом примере показано свойство `url` в компоненте HTTPService, но способ вызова службы одинаков как при непосредственном подключении к службе, так и при указании адресата.

```
<?xml version="1.0"?>
<!-- fds\rpc\HttpServiceParamBind.mxml. Compiles -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="20">
    <mx:Script>
        <![CDATA[
            import mx.utils.ArrayUtil;
        ]]>
    </mx:Script>

    <mx:HTTPService
        id="employeeSrv"
        url="employees.jsp">
        <mx:request>
            <deptId>{dept.selectedItem.data}</deptId>
        </mx:request>
    </mx:HTTPService>
    <mx:ArrayCollection
        id="employeeAC"
        source=
            "{ArrayUtil.toArray(employeeSrv.lastResult.employees.employee)}"/>
    <mx:HBox>
        <mx:Label text="Select a department:"/>
        <mx:ComboBox id="dept" width="150">
            <mx:dataProvider>
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object label="Engineering" data="ENG"/>
                        <mx:Object label="Product Management" data="PM"/>
                        <mx:Object label="Marketing" data="MKT"/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ComboBox>
        <mx:Button label="Get Employee List" click="employeeSrv.send();"/>
    </mx:HBox>
    <mx:DataGrid dataProvider="{employeeAC}"
        width="100%">
        <mx:columns>
            <mx:DataGridColumn dataField="name" headerText="Name"/>
            <mx:DataGridColumn dataField="phone" headerText="Phone"/>
            <mx:DataGridColumn dataField="email" headerText="Email"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

Если неизвестно, содержит ли результат вызова службы массив или отдельный объект, можно использовать метод `toArray()` класса mx.utils.ArrayUtil для преобразования этого результата в массив, как показано в приведенном выше примере. Если методу `toArray()` передается отдельный объект, то метод возвращает массив с данным объектом в качестве единственного элемента. Если методу передается массив, то метод возвращает этот же самый массив. Дополнительную информацию о работе с объектами ArrayCollection см. в разделе Поставщики данных и коллекции.

Привязка параметров для компонентов WebService

При использовании привязки параметров для компонента WebService операции всегда объявляются в тегах `<mx:operation>` компонента WebService. Тег `<mx:operation>` может содержать тег `<mx:request>`, имеющий в своем составе XML-узлы, ожидаемые операцией. Свойство имени тега `<mx:operation>` должно соответствовать имени одной из операций web-службы.

Данные можно привязывать к параметрам операций web-служб. Для привязки и проверки данных указываются имена тегов параметров.

В следующем примере привязка параметров используется в теге `<mx:operation>` компонента WebService для привязки данных выбранного элемента ComboBox к операции `employeeWS.getList` при нажатии пользователем элемента управления Button. Тег `<deptId>` полностью соответствует параметру `deptId` операции getList. При использовании привязки параметров вызов службы осуществляется с помощью метода `send()` без параметров. В этом примере показано свойство адресата в компоненте WebService, но способ вызова службы идентичен, как при непосредственном подключении к службе, так и при указании адресата.

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceParamBind.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.utils.ArrayUtil;
            import mx.controls.Alert;
        ]]>
    </mx:Script>

    <mx:WebService
        id="employeeWS"
        destination="wsDest"
        showBusyCursor="true"
        fault="Alert.show(event.fault.faultString)">
        <mx:operation name="getList">
            <mx:request>
                <deptId>{dept.selectedItem.data}</deptId>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:ArrayCollection
        id="employeeAC"
        source="{ArrayUtil.toArray(employeeWS.getList.lastResult)}"/>
    <mx:HBox>
        <mx:Label text="Select a department:"/>
        <mx:ComboBox id="dept" width="150">
            <mx:dataProvider>
```

```
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object label="Engineering" data="ENG"/>
                        <mx:Object label="Product Management" data="PM"/>
                        <mx:Object label="Marketing" data="MKT"/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ComboBox>
            <mx:Button label="Get Employee List"
                click="employeeWS.getList.send()"/>
    </mx:HBox>
    <mx:DataGrid dataProvider="{employeeAC}" width="100%">
        <mx:columns>
                <mx:DataGridColumn dataField="name" headerText="Name"/>
                <mx:DataGridColumn dataField="phone" headerText="Phone"/>
                <mx:DataGridColumn dataField=" to email" headerText="Email"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

Можно также вручную указать все тело запроса SOAP в XML-формате с правильной информацией о пространстве имен, определенной в теге `<mx:request>`. Для этого укажите для атрибута `format` тега `<mx:request>` значение `xml`, как показано в следующем примере:

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceSOAPRequest.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:WebService id="ws" wsdl="http://api.google.com/GoogleSearch.wsdl"
        useProxy="true">
        <mx:operation name="doGoogleSearch" resultFormat="xml">
            <mx:request format="xml">
                <ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                    <key xsi:type="xsd:string">XYZ123</key>
                    <q xsi:type="xsd:string">Balloons</q>
                    <start xsi:type="xsd:int">0</start>
                    <maxResults xsi:type="xsd:int">10</maxResults>
                    <filter xsi:type="xsd:boolean">true</filter>
                    <restrict xsi:type="xsd:string"/>
                    <safeSearch xsi:type="xsd:boolean">false</safeSearch>
                    <lr xsi:type="xsd:string" />
                    <ie xsi:type="xsd:string">latin1</ie>
                    <oe xsi:type="xsd:string">latin1</oe>
                </ns1:doGoogleSearch>
            </mx:request>
        </mx:operation>
    </mx:WebService>
</mx:Application>
```

Обработка результатов службы

После вызова службы компонентом RPC данные, возвращаемые службой, помещаются в объект lastResult. По умолчанию значением свойства `resultFormat` компонентов HTTPService и операций компонента WebService является `object`, и возвращаемые данные представлены в виде простого дерева объектов ActionScript. Flex интерпретирует данные XML, возвращаемые web-службой или службой HTTP, в соответствии с представленными базовыми типами, такими как String, Number, Boolean и Date. При работе с объектами со строгим контролем типов необходимо заполнить объекты с помощью дерева объектов, создаваемого Flex.

Компоненты WebService и HTTPService возвращают анонимные объекты и массивы сложных типов. Если `makeObjectsBindable` имеет значение `true`, являющееся значением по умолчанию, то объекты обернуты в экземпляры mx.utils.ObjectProxy, а массивы – в экземпляры mx.collections.ArrayCollection.

Примечание. *В службе ColdFusion не учитывается регистр, поэтому на внутреннем уровне все данные службы записываются в верхнем регистре. При вызове web-службы ColdFusion следует учитывать регистр.*

Обработка событий результата и ошибки

По завершении вызова службы метод RemoteObject, операция WebService или компонент HTTPService передает событие результата или событие ошибки. *Событие результата* указывает на доступность результата. *Событие ошибки* указывает на возникновение ошибки. Событие результата действует в качестве триггера для обновления свойств, связанных с lastResult. События результата и ошибки можно обрабатывать явным образом путем добавления прослушивателей событий к методам RemoteObject или операциям WebService. Для компонента HTTPService необходимо указать прослушиватели событий результата и ошибки непосредственно в компоненте, поскольку компонент HTTPService не имеет нескольких операций или методов.

Если прослушиватели событий результата или ошибки не указываются для метода RemoteObject или операции WebService, то события передаются на уровень компонента, при этом можно указать прослушиватели событий результата и ошибки на уровне компонента.

В следующем примере MXML события `result` и `fault` операции WebService определяют прослушиватели событий; событие `fault` компонента WebService также определяет прослушиватель событий:

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCResultFaultMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.rpc.soap.SOAPFault;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;
            import mx.controls.Alert;
            public function showErrorDialog(event:FaultEvent):void {
                // Handle operation fault.
                Alert.show(event.fault.faultString, "Error");
            }
            public function defaultFault(event:FaultEvent):void {
                // Handle service fault.
                if (event.fault is SOAPFault) {
                     var fault:SOAPFault=event.fault as SOAPFault;
                     var faultElement:XML=fault.element;
                     // You could use E4X to traverse the raw fault element returned in the
SOAP envelope.
                     // ...
```

```
                }
                Alert.show(event.fault.faultString, "Error");
            }
            public function log(event:ResultEvent):void {
                // Handle result.
            }
        ]]>
    </mx:Script>
    <mx:WebService id="WeatherService" wsdl="http://myserver:8500/flexapp/app1.cfc?wsdl"
        fault="defaultFault(event)">
        <mx:operation name="GetWeather"
            fault="showErrorDialog(event)"
            result="log(event)">
            <mx:request>
                <ZipCode>{myZip.text}</ZipCode>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:TextInput id="myZip"/>
</mx:Application>
```

В следующем примере ActionScript прослушиватель события результата добавлен к операции WebService, а прослушиватель события ошибки добавлен к компоненту WebService:

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCResultFaultAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.rpc.soap.WebService;
            import mx.rpc.soap.SOAPFault;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;

            private var ws:WebService;

            public function useWebService(intArg:int, strArg:String):void {
                ws = new WebService();
                ws.destination = "wsDest";
                ws.echoArgs.addEventListener("result", echoResultHandler);
                ws.addEventListener("fault", faultHandler);
                ws.loadWSDL();
                ws.echoArgs(intArg, strArg);
            }
```

```
            public function echoResultHandler(event:ResultEvent):void {
                var retStr:String = event.result.echoStr;
                var retInt:int = event.result.echoInt;
                //do something
        }

            public function faultHandler(event:FaultEvent):void {
            //deal with event.fault.faultString, etc.
                if (event.fault is SOAPFault) {
                     var fault:SOAPFault=event.fault as SOAPFault;
                     var faultElement:XML=fault.element;
                     // You could use E4X to traverse the raw fault element returned in the
SOAP envelope.
                     // ...
                }
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Для указания вызова запроса компонента доступа к данным можно также использовать событие mx.rpc.events.InvokeEvent. Это событие рекомендуется использовать при постановке операций в очередь с последующим вызовом.

Обработка результатов в XML-формате посредством формата результатов E4X

Для свойства `resultFormat` компонентов HTTPService и операций WebService можно установить значение e4x для создания свойства lastResult типа XML. Доступ к свойству `lastResult` выполняется с помощью ECMAScript для выражений XML (E4X). Корневой узел структуры XML не включается в точечную нотацию при использовании объекта XML `E4X` в выражении привязки. В этом заключается отличие от синтаксиса свойства `lastResult`, установленного на объект, для которого корневой узел структуры XML включается в точечную нотацию. Например, если для свойства lastResult установлено значение `e4x`, то следует использовать `{srv.lastResult.product}`; если для свойства lastResult указан объект, то используется `{srv.lastResult.products.product}`.

Использование формата результата `e4x` является предпочтительным способом непосредственной работы с XML, но для свойства `resultFormat` можно также установить значение `xml` для создания объекта `lastResult` типа flash.xml.XMLNode, который является устаревшим объектом при работе с XML. Кроме того, можно установить для свойства `resultFormat` компонентов HTTPService значение `flashvars` или `text` с целью создания результатов в виде объектов ActionScript, содержащих пары «имя-значение» или необработанный текст соответственно.

Примечание. *Если для результатов службы требуется использовать синтаксис E4X, установите для свойства `resultFormat` компонентов HTTPService или WebService значение `e4x`. Значение по умолчанию: `object`.*

При указании для свойства `resultFormat` компонента HTTPService или операции WebService значения `e4x` может потребоваться обработка информации о пространстве имен, содержащейся в возвращенном XML. Для компонента WebService информация о пространстве имен включена в тело оболочки SOAP, возвращаемое web-службой. В следующем примере приведен фрагмент тела SOAP, содержащий информацию о пространстве имен. Эти данные возвращены web-службой, предназначенной для получения биржевых цен. Информация о пространстве имен выделена в тексте жирным шрифтом.

```
...
<soap:Body>
<GetQuoteResponse
xmlns="http://ws.invesbot.com/">
<GetQuoteResult><StockQuote xmlns="">
<Symbol>ADBE</Symbol>
<Company>ADOBE SYSTEMS INC</Company>
<Price>&lt;big&gt;&lt;b&gt;35.90&lt;/b&gt;&lt;/big&gt;</Price>
...
</soap:Body>
...
```

Поскольку в soap:Body содержится информация о пространстве имен, при указании для свойства `resultFormat` операции WebService значения `e4x` необходимо создать объект для пространства имен http://ws.invesbot.com/. Такое приложение показано в следующем примере:

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceE4XResult1.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
    pageTitle="Test" >
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            private namespace invesbot = "http://ws.invesbot.com/";
            use namespace invesbot;
        ]]>
    </mx:Script>
    <mx:WebService
        id="WS"
        destination="stockservice" useProxy="true"
        fault="Alert.show(event.fault.faultString), 'Error'">
        <mx:operation name="GetQuote" resultFormat="e4x">
            <mx:request>
                <symbol>ADBE</symbol>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:HBox>
        <mx:Button label="Get Quote" click="WS.GetQuote.send()"/>
        <mx:Text
            text="{WS.GetQuote.lastResult.GetQuoteResult.StockQuote.Price}"
        />
    </mx:HBox>
</mx:Application>
```

Как показано в следующем примере, в привязке к результату службы дополнительно можно создать переменную для пространства имен и доступ к ней:

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceE4XResult2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
    pageTitle="Test" >
    <mx:Script>
        <![CDATA[
        import mx.controls.Alert;
        public var invesbot:Namespace =
            new Namespace("http://ws.invesbot.com/");
        ]]>
    </mx:Script>
    <mx:WebService
        id="WS"
        destination="stockservice" useProxy="true"
        fault="Alert.show(event.fault.faultString), 'Error'">
        <mx:operation name="GetQuote" resultFormat="e4x">
            <mx:request>
                <symbol>ADBE</symbol>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:HBox>
        <mx:Button label="Get Quote" click="WS.GetQuote.send()"/>
        <mx:Text
          text="{WS.GetQuote.lastResult.invesbot::GetQuoteResult.StockQuote.Price}"
        />
    </mx:HBox>
</mx:Application>
```

Для доступа к элементам и атрибутам XML-данных, возвращаемых в объекте lastResult, используется синтаксис E4X. В зависимости от того, объявлено ли в XML-данных одно или несколько пространств имен, используются различные синтаксисы.

Пространство имен не указано

В следующем примере показано получение значения элемента или атрибута, если в элементе или атрибуте пространство имен не указано:

```
var attributes:XMLList = XML(event.result).Description.value;
```

Предыдущий код возвращает xxx для приведенного ниже XML-документа:

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <Description>
        <value>xxx</value>
    </Description>
</RDF>
```

Указано любое пространство имен

В следующем примере показано получение значения элемента или атрибута, если в элементе или атрибуте указано любое пространство имен:

```
var attributes:XMLList = XML(event.result).*::Description.*::value;
```

Предыдущий код возвращает xxx для любого из следующих XML-документов:

Первый XML-документ:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description>
        <rdf:value>xxx</rdf:value>
    </rdf:Description>
</rdf:RDF>
```

Второй XML-документ:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cm="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <cm:Description>
        <rdf:value>xxx</rdf:value>
    </cm:Description>
</rdf:RDF>
```

Указано определенное пространство имен

В следующем примере показано получение значения элемента или атрибута, если в элементе или атрибуте указано объявленное пространство имен rdf:

```
var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
var attributes:XMLList = XML(event.result).rdf::Description.rdf::value;
```

Предыдущий код возвращает xxx для приведенного ниже XML-документа:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description>
        <rdf:value>xxx</rdf:value>
    </rdf:Description>
</rdf:RDF>
```

В следующем примере показан альтернативный способ получения значения элемента или атрибута, если в элементе или атрибуте указано объявленное пространство имен rdf:

```
namespace rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#";
use namespace rdf;
var attributes:XMLList = XML(event.result).rdf::Description.rdf::value;
```

Предыдущий код также возвращает xxx для приведенного ниже XML-документа:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description>
        <rdf:value>xxx</rdf:value>
    </rdf:Description>
</rdf:RDF>
```

Обработка результатов web-службы, содержащих .NET DataSet или .NET DataTable

Web-службы, созданные с помощью Microsoft .NET Framework, могут возвращать клиенту специальные объекты .NET DataSet или .NET DataTable. Web-служба .NET предоставляет базовый документ WSDL без информации о типе данных, которыми он управляет. При возвращении веб-службой объектов DataSet или DataTable информация о типе данных встраивается в элемент XML-схемы в сообщении SOAP, указывающем способ обработки остальной части сообщения. Для оптимальной обработки результатов web-службы такого

типа следует установить для свойства `resultFormat` операции Flex WebService значение `object`. Дополнительно для свойства `resultFormat` операции WebService можно установить значение `e4x`, но XML-формат и формат e4x неудобны вследствие необходимости обработки нестандартной структуры ответа и реализации методов обхода проблем, например, при требуемой привязке данных к элементу управления DataGrid.

После установки для свойства `resultFormat` операции Flex WebService значения `object` объект DataTable или DataSet, возвращаемый web-службой .NET, автоматически преобразуется в объект со свойством Tables, содержащим схему одного или нескольких объектов DataTable. Каждый объект DataTable из схемы Tables содержит два свойства: Columns и Rows. Свойство Rows содержит данные. Объект event.result получает следующие свойства, соответствующие свойствам DataSet и DataTable в .NET. Массивы DataSet или DataTable имеют одинаковую структуру, представленную в данном документе, но вложены в массив верхнего уровня в объекте результата.

Свойство	**Описание**
`result.Tables`	Схема имен таблиц для объектов, содержащих данные таблиц
`result.Tables["someTable"].Columns`	Массив имен столбцов в порядке, указанном для таблицы в схеме DataSet или DataTable
`result.Tables["someTable"].Rows`	Массив объектов, представляющих данные каждой строки таблицы. Например, {columnName1:value, columnName2:value, columnName3:value}.

Следующее приложение MXML заполняет элемент управления DataGrid данными, возвращенными из web-службы .NET.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns="*" xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">
    <mx:WebService
        id="nwCL"
        wsdl="http://localhost/data/CustomerList.asmx?wsdl"
        result="onResult(event)"
        fault="onFault(event)" />
    <mx:Button label="Get Single DataTable" click="nwCL.getSingleDataTable()"/>
    <mx:Button label="Get MultiTable DataSet" click="nwCL.getMultiTableDataSet()"/>
    <mx:Panel id="dataPanel" width="100%" height="100%" title="Data Tables"/>

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.controls.DataGrid;
            import mx.rpc.events.FaultEvent;
            import mx.rpc.events.ResultEvent;

            private function onResult(event:ResultEvent):void {
                // A DataTable or DataSet returned from a .NET webservice is
                // automatically converted to an object with a "Tables" property,
                // which contains a map of one or more dataTables.
                if (event.result.Tables != null)
                {
                    // clean up panel from previous calls.
                    dataPanel.removeAllChildren();

                    for each (var table:Object in event.result.Tables)
                    {
```

```
                    displayTable(table);
                }

                // Alternatively, if a table's name is known beforehand,
                // it can be accessed using this syntax:
                var namedTable:Object = event.result.Tables.Customers;
                //displayTable(namedTable);
            }
        }

        private function displayTable(tbl:Object):void {
            var dg:DataGrid = new DataGrid();
            dataPanel.addChild(dg);
            // Each table object from the "Tables" map contains two properties:
            // "Columns" and "Rows". "Rows" is where the data is, so we can set
            // that as the dataProvider for a DataGrid.
            dg.dataProvider = tbl.Rows;
        }

        private function onFault(event:FaultEvent):void {
            Alert.show(event.fault.toString());
        }
    ]]>
</mx:Script>

</mx:Application>
```

В следующем примере представлен класс C# .NET, являющийся реализацией серверной части web-сервиса, вызываемого приложением ; этот класс использует демонстрационную базу данных Northwind Microsoft SQL Server:

:

```
<%@ WebService Language="C#" Class="CustomerList" %>
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Data;
using System.Data.SqlClient;
using System;

public class CustomerList : WebService {
    [WebMethod]
    public DataTable getSingleDataTable() {
        string cnStr = "[Your_Database_Connection_String]";
        string query = "SELECT TOP 10 * FROM Customers";
        SqlConnection cn = new SqlConnection(cnStr);
        cn.Open();
        SqlDataAdapter adpt = new SqlDataAdapter(new SqlCommand(query, cn));
        DataTable dt = new DataTable("Customers");

        adpt.Fill(dt);
        return dt;
    }
```

```
    [WebMethod]
    public DataSet getMultiTableDataSet() {
        string cnStr = "[Your_Database_Connection_String]";
        string query1 = "SELECT TOP 10 CustomerID, CompanyName FROM Customers";
        string query2 = "SELECT TOP 10 OrderID, CustomerID, ShipCity,
        ShipCountry FROM Orders";
        SqlConnection cn = new SqlConnection(cnStr);
        cn.Open();

        SqlDataAdapter adpt = new SqlDataAdapter(new SqlCommand(query1, cn));
        DataSet ds = new DataSet("TwoTableDataSet");
        adpt.Fill(ds, "Customers");

        adpt.SelectCommand = new SqlCommand(query2, cn);
        adpt.Fill(ds, "Orders");

        return ds;
    }
}
```