使用 ADOBE® FLEX® 4.6
访问数据

法律声明

目录

第 1 章 : 访问数据服务概述

Flex 与其它数据访问技术的比较

Flex 使用数据源和数据的方式与将 HTML 用于用户界面的应用程序是不同的。

客户端处理和服务器端处理

与借助于 JSP 和 servlet、 ASP、 PHP 或 CFML 创建的 HTML 模板集不同， Flex 区分客户端代码与服务器代码，它将应用程序用户界面编译为二进制 SWF 文件，并发送到客户端。

当应用程序向数据服务发出请求时，不会重新编译 SWF 文件，也无需刷新页面。远程服务仅返回数据，随后 Flex 会将所返回的数据绑定到客户端应用程序中的用户界面组件。

例如，在 Flex 中，当用户在应用程序中单击 Button 控件时，客户端代码会调用 Web 服务。从 Web 服务中获得的结果数据将返回到二进制 SWF 文件中，而无需刷新页面。这样，结果数据即可作为动态内容应用于应用程序。

```
<?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)"/>
```

将这个 Flex 示例与以下示例进行比较。以下示例显示了一段使用 JSP 自定义标签调用 Web 服务的 JSP 代码。当用户请求 JSP 时，将在服务器 （而不是客户端）上发出 Web 服务请求，然后使用结果生成 HTML 页中的内容。应用程序服务器会重新生成整个 HTML 页，然后再将其发回到用户的 Web 浏览器。

```
<%@ 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") %>
```

数据源访问

Flex 与其它 Web 应用程序技术之间的另一个区别是，在 Flex 中，您从不直接与数据源通信，使用数据访问组件可连接远程服务，以及与服务器端数据源交互。

以下示例显示直接访问数据源的 ColdFusion 页面：

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

要在 Flex 中获得类似的功能，可使用 HTTPService、 Web 服务或者 RemoteObject 组件来调用从数据源返回结果的服务器端对象。

事件、服务调用和数据绑定

Flex 是一种事件驱动型技术，通过用户操作或程序事件，可触发对服务的访问。例如，用户单击按钮就是一个可以触发服务调用的用户操作事件。程序事件的一个示例是，当应用程序创建完用户界面组件 （如 DataGrid）后，可以使用 DataGrid 的 creationComplete 事件调用远程服务以填充 DataGrid。

Flex 中的服务调用是异步的。客户端应用程序无需等待数据返回。在检索或更新大型数据集时，异步服务调用很有用。因为在等待检索或更新数据时，不会阻止客户端应用程序。

从服务调用返回的数据存储在与服务调用相关联的 CallResponder 中。用户界面组件随后使用数据绑定功能从 CallResponder 检索返回的数据。

使用 Flex 中的数据绑定功能，可以使用数据源动态更新用户界面组件。例如，Flex 组件可以将其 text 属性与 CallResponder 的 lastResult 属性相关联。 CallResponder 中的数据改变时， Flex 组件会自动更新。

Flex 还可以实现双向数据绑定。借助双向数据绑定，无论是 Flex 组件还是数据源中的数据发生变化，相应的数据源或 Flex 组件都会自动更新。在将用户输入的远程数据更新到 Form 组件或 Flex 数据组件时，双向数据绑定非常有用。

更多帮助主题

第 6 页的 “ 使用 Flash Builder 构建以数据为中心的应用程序 ”

使用 Flash Builder 访问数据服务

在 Flex Builder 3 中，可使用 Flex 数据访问组件来实现对数据服务的远程过程调用。但 Flash Builder 将此过程进行了简化。

Flash Builder 提供的各种向导和其它工具可以：

- 帮助访问数据服务
- 配置数据服务返回的数据
- 协助对从服务返回的数据进行分页
- 协助数据管理功能 （该功能使服务器数据的多个更新同步）
- 生成用于访问数据服务的客户端代码
- 将从服务返回的数据绑定到用户界面组件

使用 Flash Builder 访问服务工作流

可以按照下述工作流，使用 Flash Builder 创建用于访问数据服务的应用程序。

1 您可以先连接数据服务，也可以先构建用户界面，具体取决于您的环境。

连接远程服务。如果先连接了远程服务，则随后再构建用户界面。

构建用户界面。如果先构建了用户界面，则随后再连接远程服务。

注： 可以根据个人喜好决定先执行哪个操作。例如，如果您已经规划了一个用户界面设计，则可以先构建用户界面。否则，您可以先连接数据，然后借助 Flash Builder 生成应用程序组件。

2 将数据操作绑定到应用程序组件。

3 （可选）管理数据检索和更新。

使用 Flash Builder 工具，可以实现对返回数据的分页，并协调对多组数据的更新。

在返回大量数据记录时，通常需要执行分页，以便“ 按需 ”检索记录集。

对于更新多个记录的应用程序，可以执行数据管理功能。数据管理功能包括：

- 提交功能，用于同时更新更改的记录
- 撤消机制，用于在将所做的更改写入到服务器之前还原所做的更改
- 代码生成，用于在添加、删除或更改记录时自动更新用户界面组件

4 运行应用程序并监视数据流。

在应用程序完成之后，请运行该应用程序，以便查看其运行过程。使用 Flash Builder 网络监视器查看在应用程序和服务之间传递的数据。网络监视器对于诊断错误和分析性能非常有用。

Flash Builder 还提供了可靠的调试和概要分析环境。网络监视器和 Flash 概要分析器在 Flash Builder Premium 中提供。

更多帮助主题

第 6 页的“ 使用 Flash Builder 构建以数据为中心的应用程序 ”

扩展 Flash Builder 支持的服务

Flash Builder 向导和工具支持访问的服务实现类型如下：

- PHP 服务
- ColdFusion 服务

- BlazeDS
- LiveCycle Data Services
- HTTP （REST 样式）服务
- Web 服务 (SOAP)
- 静态 XML 文件

如果需要对其它服务类型 （如 Ruby on Rails）的工具支持，则可以扩展 Flash Builder 实现。请参阅 Flash Builder Extensibility Reference。

数据访问组件

通过数据访问组件，客户端应用程序可以通过网络调用操作和服务。数据访问组件使用远程过程调用与服务器环境进行交互。数据访问组件有以下三种：RemoteObject 组件、 HTTPService 组件和 WebService 组件。

数据访问组件是针对客户端应用程序设计的，其中的调用和响应模型是访问外部数据的最佳选择。通过这些组件，客户端可以对处理请求的远程服务发出异步请求，然后将数据返回到应用程序。

数据访问组件调用远程服务。随后将从服务获得的响应数据存储到 ActionScript 对象中，或存储为服务返回的任何其它格式。在客户端应用程序中使用数据访问组件可处理以下三种服务类型：

- 远程对象服务 (RemoteObject)
- 基于 SOAP 的 Web 服务 (WebService)
- HTTP 服务，包括基于 REST 的 Web 服务 (HTTPService)

Adobe® Flash® Builder™ 提供了各种向导和工具，用于将数据访问组件的实现包装到服务包装器中。服务包装器封装数据访问组件的功能，为您针对许多低级别实现提供保护。从而您可以更专注于实现服务以及构建用于访问该服务的客户端应用程序。有关使用 Flash Builder 访问数据服务的更多信息，请参阅第 6 页的 “ 使用 Flash Builder 构建以数据为中心的应用程序 ”。

提供对服务的访问

默认情况下，Adobe Flash Player 会阻止对加载应用程序所用主机以外的任何其它主机进行访问。如果未使用服务器端应用程序（例如 LiveCycle Data Services 或 BlazeDS）来代理请求，则 HTTP 服务或 Web 服务必须位于承载着您的应用程序的服务器上，否则，承载着 HTTP 服务或 Web 服务的远程服务器必须定义 crossdomain.xml 文件。通过 crossdomain.xml 文件，服务器可以指示特定域或所有域中提供的 SWF 文件是否可以使用其数据和文档。crossdomain.xml 文件必须位于应用程序正在连接的服务器的 Web 根文件夹下。

HTTPService 组件

在客户端应用程序中，可以使用 HTTPService 组件发送 HTTP GET 或 POST 请求，以及包含从 HTTP 响应获得的数据。如果使用 Flex 来构建桌面应用程序 （在 Adobe AIR® 中运行），则支持 HTTP PUT 和 DELETE。

如果使用了 LiveCycle Data Services 或 BlazeDS，则可以使用 HTTPProxyService，它允许您使用其他 HTTP 方法。通过 HTTPProxyService，可以发送 GET、 POST、 HEAD、 OPTIONS、 PUT、 TRACE 或 DELETE 请求。

HTTP 服务可以是接受 HTTP 请求并发送响应的任何 HTTP URI。此服务类型通常又称为 REST 样式的 Web 服务。REST 表示代表性状态传输 (Representational State Transfer)，是一种针对分布式超媒体系统的体系结构样式。

无法公开与 SOAP Web 服务或远程对象服务相同的功能时，使用 HTTPService 组件是最佳选择。例如，可以使用 HTTPService 组件与未以 Web 服务或远程服务目标形式提供的 JavaServer Pages (JSP)、 Servlet 和 ASP 页进行交互。

调用 HTTPService 对象的 send() 方法时，该方法将对指定的 URI 发出 HTTP 请求，并返回 HTTP 响应。可以选择将参数传递给指定的 URI。

Flash Builder 提供了用于以交互方式连接 HTTP 服务的工作流。有关更多信息，请参阅第 10 页的 “ 访问 HTTP 服务 ”。

更多帮助主题

第 10 页的 “ 访问 HTTP 服务 ”

学位论文：Representational State Transfer (REST)，作者 Roy Thomas Fielding

WebService 组件

WebService 组件用于访问 SOAP Web 服务，此类服务是带有方法的软件模块。Web 服务方法通常称为 “ 操作 ”。Web 服务接口通过 Web 服务描述语言 (WSDL) 进行定义。通过 Web 服务提供的标准相容方式，在不同平台上运行的软件模块可以相互交互。有关 Web 服务的详细信息，请访问万维网联合会网站的 Web 服务部分，网址为 www.w3.org/2002/ws/。

客户端应用程序可以与在 Web 服务描述语言 (WSDL) 文档 （以 URL 形式提供）中定义其接口的 Web 服务进行交互。WSDL 是一种标准格式，用于描述能够由 Web 服务理解的消息、Web 服务对于这些消息的响应格式、Web 服务支持的协议以及将这些消息发送到何处。

Flex 支持 WSDL 1.1，有关说明，请访问 www.w3.org/TR/wsdl。 Flex 支持 RPC-encoded 和 document-literal Web 服务。

Flex 支持格式设置为 SOAP 消息且通过 HTTP 传输的 Web 服务请求和结果。 SOAP 提供基于 XML 格式的定义，用于在 Web 服务客户端 （如使用 Flex 构建的应用程序）和 Web 服务之间交换结构化和类型化信息。

如果在您的环境中 Web 服务是公认标准，则可以使用 WebService 组件连接 SOAP 相容的 Web 服务。对于企业环境中的对象， WebService 组件也很有用，但无需在 Web 应用程序的源路径上提供。

Flash Builder 提供了用于以交互方式连接 Web 服务的工作流。有关更多信息，请参阅第 13 页的 “ 访问 Web 服务 ”。

RemoteObject 组件

与 REST 样式服务或 Web 服务一样，使用远程对象服务可以直接以本机格式访问业务逻辑，而无需转化为 XML 格式，从而节约将现有逻辑公开为 XML 所需的时间。远程对象服务的另一个好处是通过有线网络进行通信的速度较快。虽然数据交换仍通过 HTTP 或 HTTPS 进行，但数据自身会序列化为二进制表示形式。使用 RemoteObject 组件将减少通过有线网络传输的数据量，降低客户端内存使用率，从而缩短处理时间。

在访问服务器上的数据时，ColdFusion、PHP、BlazeDS 和 LiveCycle Data Services 可以使用服务器端类型化。客户端应用程序可直接通过远程调用指定对象上的方法，来访问 Java 对象、ColdFusion 组件（本质上是一个 Java 对象）或 PHP 类。服务器上的对象使用本机数据类型作为参数，使用这些参数查询数据库，并返回属于其本机数据类型的值。

当服务器端类型化不可用时，可以使用 Flash Builder 提供的工具实现客户端类型化。使用 Flash Builder 可配置和定义从服务返回的数据的类型。通过客户端类型化，客户端应用程序可以查询数据库，并检索已正确类型化的数据。对于任何未定义服务返回的数据类型的服务来说，客户端类型化是必需的。

Flash Builder 提供了用于以交互方式连接远程对象服务的工作流。有关更多信息，请参阅第 7 页的 “ 连接数据服务 ”。

第 2 章：使用 Flash Builder 构建以数据为中心的应用程序

可以使用 Flash Builder 工具创建访问数据服务的应用程序。首先为应用程序创建一个 Flex 项目。然后连接数据服务，配置从服务对数据的访问，并为应用程序构建用户界面。在某些情况下，也可以先创建用户界面，然后访问数据服务。

创建 Flex 项目以访问数据服务

Flex 将数据服务作为远程对象、 HTTP （REST 样式）服务或 SOAP Web 服务进行访问。

使用远程对象可以访问的数据服务类型如下：

- ColdFusion 服务
- 基于 AMF 的 PHP 服务
- BlazeDS
- LiveCycle Data Services

有关使用 LiveCycle Service Discovery 向导的详细信息，请参阅 Using LiveCycle Discovery。

对于作为远程对象访问的服务，可创建为相应的应用程序服务器类型配置的 Flex 项目。新建 Flex 项目向导将引导您为以下列出的应用程序服务器类型配置项目：

服务器类型	支持的远程对象服务
PHP	• 基于 AMF 的 PHP 服务
ColdFusion	• ColdFusion Flash Remoting • BlazeDS • LiveCycle Data Services
J2EE	• BlazeDS • LiveCycle Data Services

可以通过任何 Flex 项目配置 （包括未指定服务器技术的项目）连接 HTTP （REST 样式）服务和 SOAP Web 服务。

配置为访问远程对象的项目仅可以访问为其配置的远程对象服务。例如，无法通过为 ColdFusion 配置的项目访问基于 AMF 的 PHP 服务。但是，如果作为 Web 服务或 HTTP 服务连接 PHP 服务，则可以通过此项目进行连接。

更多帮助主题

第 1 页的 “ 访问数据服务概述 ”

更改项目的服务器类型

如果您试图访问未配置 Flex 项目的服务，则会收到 Flash Builder 的通知。如果 Flex 项目未指定正确的服务器配置，则 Flash Builder 会提供一个指向 “ 项目属性 ” 对话框的链接。在 “ 项目属性 ” 对话框中，可以对项目进行配置，以访问数据服务。例如，如果项目未指定服务器配置，则在您尝试访问该项目中的基于 AMF 的 PHP 服务时， Flash Builder 将发出警告。

如果先前已将 Flex 项目配置为访问其它类型的服务，请配置新的 Flex 项目或更改当前项目的配置。如果更改了项目的服务器配置，则不再能够访问先前配置的任何服务。例如，如果将项目配置从 ColdFusion 更改为 PHP，则在项目中访问的所有 ColdFusion 服务将不再可用。

如果需要从同一项目访问不同类型的服务，则可以作为 HTTP 服务或 Web 服务访问服务。

跨域策略文件

如果要访问的服务与应用程序的 SWF 文件位于不同的域中，则必须使用跨域策略文件。基于 AMF 的服务通常不需要跨域策略文件，因为这些服务与应用程序位于同一域中。

连接数据服务

可以通过 Flash Builder 服务向导连接数据服务。

对于远程对象服务，通常会使用相应的应用程序服务器类型创建 Flex 项目。 Flash Builder 会内部检查服务，并可为服务返回的数据配置返回类型。

远程对象服务包括在 ColdFusion、 PHP、 BlazeDS 和 LiveCycle Data Services 中实现的数据服务。

有关使用 LiveCycle Service Discovery 向导的详细信息，请参阅 Using LiveCycle Discovery。

更多帮助主题

第 6 页的 “ 创建 Flex 项目以访问数据服务 ”

访问 ColdFusion 服务

使用 Flash Builder 服务向导可访问已作为 ColdFusion 组件 (CFC) 实现的 ColdFusion 数据服务。 Flex 将这些服务作为远程对象进行访问。

使用将 ColdFusion 指定为应用程序服务器类型的 Flex 项目。创建 Flex 项目时，需指定 “ 使用远程对象访问服务 ” 和 “ 使用 ColdFusion Flash Remoting”。

连接 ColdFusion 数据服务

此过程假定已实现 ColdFusion 服务，并已创建用于访问 ColdFusion 服务的 Flex 项目。

1 在 Flash Builder“ 数据 ” 菜单中，选择 “ 连接 ColdFusion”，打开服务窗口。

2 在 “ 配置 ColdFusion 服务 ” 对话框中，浏览到实现该服务的 CFC 的位置。

注: 如果尚未实现 ColdFusion 服务，则 Flash Builder 可以从单个数据库表生成示例服务。可以参考生成的示例，了解如何访问数据服务。请参阅第 8 页的 “ 从数据库表生成示例 ColdFusion 服务 ”。

3 （可选）修改服务详细信息。

服务名称	指定服务名称。 Flash Builder 会基于服务的文件名，为服务生成一个名称。 服务可以使用的名称受到限制。请参阅第 16 页的 “ 数据服务命名 ”。
服务包	为包含生成的 ActionScript 文件的包指定名称，这些文件用于访问服务。 Flash Builder 根据服务名称生成包，并将其放置在 services 包中。
数据类型包	为包含生成的 ActionScript 类文件的包指定名称，这些文件用于定义从服务检索到的数据类型。 默认情况下， Flash Builder 会创建 valueObjects 包。

4 （可选）单击 “ 下一步 ” 查看服务操作。

5 单击 “ 完成 ” 生成访问该服务的 ActionScript 文件。

注： 在连接服务之后，可以修改服务属性。在 “ 数据 / 服务 ” 视图中选择服务。然后在上下文菜单中，选择 “ 属性 ”。

下一步：第 22 页的 “ 为数据服务操作配置数据类型 ”。

从数据库表生成示例 ColdFusion 服务

可以使用 Flash Builder 生成的示例 ColdFusion 服务作为自创服务的原型。示例服务可访问单个数据库表，且包含用于创建、读取、更新和删除的方法。

Flash Builder 可为生成的服务配置返回数据类型，并可启用数据访问功能 （如分页和数据管理）。

重要说明： 生成的服务仅可在受信任的开发环境中使用。因为通过生成的代码，任何能够通过网络访问服务器者，都可以对数据库表中的数据执行访问、修改或删除操作。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全服务的信息，请参阅保护数据服务。

以下过程假定您已创建用于访问 ColdFusion 服务的 Flex 项目，并具有可用的 ColdFusion 数据源。

1 在 Flash Builder“ 数据 ” 菜单中，选择 “ 连接 ColdFusion”，打开服务向导。

2 在 “ 配置 ColdFusion 服务 ” 对话框中，单击用于生成示例服务的链接。

3 选择 “ 从 RDS 数据源生成 ”，并指定 ColdFusion 数据源和表。

如果该表未定义主键，请为该表选择一个主键。

注： 如果没有可用的 ColdFusion 数据源，请选择 “ 从模板生成 ”。 Flash Builder 编写了一个具有典型服务操作的示例 ColdFusion 组件 (CFC)。取消对 CFC 中特定函数的注释，并修改操作就可以创建一个可用作原型的示例服务。

4 使用默认位置或指定新位置。单击 “ 确定 ”。

Flash Builder 即生成示例服务。可以修改服务名称和包位置，以覆盖默认值。

5 （可选）单击 “ 下一步 ” 查看服务中的操作。

6 单击 “ 完成 ”。

Flash Builder 将生成访问示例服务的 ActionScript 文件。 Flash Builder 还会在系统上的编辑器中打开示例服务，该编辑器必须已经过注册用于编辑 ColdFusion CFC 文件。

访问 PHP 服务

可以通过 Flash Builder 服务向导连接使用 PHP 实现的数据服务。Flex 使用 Action Message Format (AMF) 来序列化客户端应用程序和数据服务之间的数据。Flash Builder 安装 Zend AMF 框架，以便访问使用 PHP 实现的服务。请参阅第 17 页的 “ 安装 Zend Framework”。

通过将 PHP 指定为应用程序服务器类型的 Flex 项目访问 PHP 数据服务。配置 PHP 的项目时，指定的 Web 根目录下必须提供数据服务。将服务放置在如下所列的 services 目录中：

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

更多帮助主题

第 6 页的 “ 创建 Flex 项目以访问数据服务 ”

连接 PHP 数据服务

此过程假定已实现 PHP 服务，并已创建用于访问 PHP 服务的 Flex 项目。

1 在 Flash Builder“ 数据 ” 菜单中，选择 “ 连接 PHP”，打开服务向导。

2 在 “ 配置 PHP 服务 ” 对话框中，浏览到实现服务的 PHP 文件。

注： 如果尚未实现 PHP 服务，则 Flash Builder 可以从单个数据库表生成一个示例服务。可以参考生成的示例，了解如何访问数据服务。请参阅第 9 页的 “ 从数据库表生成示例 PHP 服务 ”。

3 （可选）修改服务详细信息。

服务名称	指定服务名称。 Flash Builder 会基于服务的文件名，为服务生成一个名称。 服务可以使用的名称受到限制。请参阅第 16 页的 “ 数据服务命名 ”。
服务包	为包含生成的 ActionScript 文件的包指定名称，这些文件用于访问服务。 Flash Builder 根据服务名称生成包，并将其放置在 services 包中。
数据类型包	为包含生成的 ActionScript 类文件的包指定名称，这些文件用于定义从服务检索到的数据类型。 默认情况下， Flash Builder 会创建 valueObjects 包。

4 单击 “ 下一步 ” 查看服务操作。

如果没有安装访问 PHP 服务所需的 Zend Framework 的支持版本，Flash Builder 将提示您安装 Zend Framework 的最低版本。请参阅第 17 页的 “ 安装 Zend Framework”。

5 单击 “ 完成 ”。

Flash Builder 即生成访问该服务的 ActionScript 文件。

注： 在连接服务之后，可以修改服务属性。在 “ 数据 / 服务 ” 视图中选择服务。然后在上下文菜单中，选择 “ 属性 ”。

下一步：第 22 页的 “ 为数据服务操作配置数据类型 ”。

从数据库表生成示例 PHP 服务

可以使用 Flash Builder 生成的示例 PHP 服务作为自创服务的原型。示例服务可访问单个 MySQL 数据库表，且包含用于创建、读取、更新和删除的方法。

Flash Builder 可为生成的服务配置返回数据类型，并可启用数据访问功能 （如分页和数据管理）。

重要说明： 生成的服务仅可在受信任的开发环境中使用。因为通过生成的代码，任何能够通过网络访问服务器者，都可以对数据库表中的数据执行访问、修改或删除操作。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全服务的信息，请参阅保护数据服务。

以下过程假定您已创建用于访问 PHP 服务的 Flex 项目，并具有可用的 MySQL 数据源。

1 在 Flash Builder“ 数据 ” 菜单中，选择 “ 连接 PHP”，打开服务向导。

2 在 “ 配置 PHP 服务 ” 对话框中，单击用于生成示例服务的链接。

3 选择“从数据库生成”，并指定用于连接数据库的信息。单击“连接数据库”。

注: 如果没有可用的 PHP 数据源，请选择“从模板生成”。Flash Builder 编写了一个具有典型服务操作的示例项目。取消对项目特定区域的注释，并修改操作就可以创建一个可用作原型的示例服务。

4 选择数据库中的一个表，并指定主键。

5 使用默认位置或指定新位置。单击“确定”。

如果没有安装访问 PHP 服务所需的 Zend Framework 的支持版本，Flash Builder 将提示您安装 Zend Framework 的最低版本。请参阅第 17 页的“安装 Zend Framework”。

Flash Builder 即生成示例服务。可以修改服务名称和包位置，以覆盖默认值。

6 （可选）单击“下一步”查看服务中的操作。

7 单击“完成”。

Flash Builder 将生成访问示例服务的 ActionScript 文件。Flash Builder 还会使用系统编辑器打开示例服务，该编辑器必须已经过注册用于编辑 PHP 文件。

访问 HTTP 服务

可以通过 Flash Builder 服务向导连接基于 REST 的 HTTP 服务。可以通过任何 Flex 项目连接 HTTP 服务。而不必为项目指定服务器技术。

如果要访问的服务与客户端应用程序的 SWF 文件位于不同的域中，则必须使用跨域策略文件。请参阅 Using cross-domain policy files。

配置 HTTP 服务

访问基于 REST 的 HTTP 服务时，可以通过多种方式配置对此服务的访问。“配置 HTTP 服务”向导支持以下内容：

- 使用基本 URL 作为前缀

  如果要访问一个服务的多个操作，使用基本 URL 作为前缀将会很方便。如果为服务指定了基本 URL，则只需为每个操作指定 HTTP 操作的相对路径。

  如果要访问多个服务，则无法使用基本 URL。

- 带有查询参数的 URL

  为操作指定 URL 时，可以在服务操作的 URL 中包含查询参数。“配置 HTTP 服务”向导将操作 URL 中包含的每一个参数填入“参数”表。

- 带有分隔参数的 RESTful 服务

  Flash Builder 支持访问 RESTful 服务，该服务使用分隔参数而不是 GET 查询参数。例如，假设您使用以下 URL 访问 RESTful 服务：

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

  请使用花括号 ({}) 指定操作 URL 中的参数。例如：

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

  随后，“配置 HTTP 服务”向导填充“参数”表：

名称	数据类型	参数类型
items	String	URL
itemID	String	URL

指定 RESTful 服务参数时，始终分别将 “ 数据类型 ” 和 “ 参数类型 ” 配置为 “String” 和 “URL”。

注： 为操作指定 URL 时，可以混合使用 RESTful 服务参数和查询参数。

- 操作 URL 的本地文件的路径

  可以为操作 URL 指定实现 HTTP 服务的本地文件的路径。例如，可以为操作 URL 指定以下路径：

  ```
  c:/MyHttpServices/MyHttpService.xml
  ```

- 添加 GET 和 POST 操作

  配置 HTTP 服务时，可以添加其它操作。在 “ 操作 ” 表中单击 “ 添加 ” 以添加操作。

  指定 GET 或 POST 作为操作方法。

- 向操作中添加参数

  可以向在 “ 操作 ” 表中选定的操作添加参数。在 “ 操作 ” 表中选择操作，然后在 “ 参数 ” 表中单击 “ 添加 ”。

  为添加的参数指定名称和 “ 数据类型 ”。“ 参数类型 ” （GET 或 POST）对应于操作方法。

- POST 操作的内容类型

  可以为 POST 操作指定内容类型。内容类型可以是 application/x-www-form-urlencoded 或 application/xml。

  如果选择 applicaton/xml 作为内容类型，Flash Builder 将生成不可编辑的查询参数。strXML 为默认名称。可以在运行时指定实际参数。

名称	数据类型	参数类型
strXML	String	POST

无法为 application/xml 内容类型添加其它参数。

连接 HTTP 服务

1 在 Flash Builder“ 数据 ” 菜单中，选择 “ 连接 HTTP”，打开服务向导。

2 （可选）指定一个基本 URL 用作所有操作的前缀。

3 在 “ 操作 ” 下，为要访问的每个操作指定以下各项：

   - 指定操作方法 （GET 或 POST）
   - 服务操作的 URL

     请将操作的所有参数包含在该 URL 中。请使用花括号 ({}) 指定 REST 样式服务参数。

     Flash Builder 支持访问以下协议：

     http://

     https://

     标准绝对路径，如 C:/ 或 /Applications/
   - 操作的名称

4 对于选定 URL 中的每个操作参数，指定参数的名称和数据类型。

5 （可选）单击 “ 添加 ” 或 “ 删除 ”，向选定操作添加参数或从中删除参数。

6 （可选）修改服务详细信息。

服务名称	指定服务名称。 Flash Builder 会基于服务的文件名，为服务生成一个名称。 服务可以使用的名称受到限制。请参阅第 16 页的 “ 数据服务命名 ”。
服务包	为包含生成的 ActionScript 文件的包指定名称，这些文件用于访问服务。 Flash Builder 根据服务名称生成包，并将其放置在 services 包中。
数据类型包	为包含生成的 ActionScript 类文件的包指定名称，这些文件用于定义从服务检索到的数据类型。 默认情况下， Flash Builder 会创建 valueObjects 包。

7 （可选）修改为该服务生成的包名。

8 单击 “ 完成 ”。

Flash Builder 即生成访问该服务的 ActionScript 文件。

连接 HTTP 服务后，可配置服务操作的返回类型。配置返回类型时，还可配置操作的参数的类型。请参阅第 22 页的 “ 为数据服务操作配置数据类型 ”。

注： 在连接服务之后，可以修改服务属性。在 “ 数据 / 服务 ” 视图中选择服务。然后在上下文菜单中，选择 “ 属性 ”。

下一步：第 22 页的 “ 为数据服务操作配置数据类型 ”。

访问实现 HTTP 服务的 XML 文件

可以访问实现 HTTP 服务的静态 XML 文件。静态 XML 文件可以是本地文件或者以 URL 形式提供。

服务使用返回 XML 响应的 GET 方法。此功能有助于了解 Flex 中的 HTTP 服务以及制作客户端应用程序中的 mock 数据原型。

访问服务时，可以指定返回 XML 响应的节点。Flash Builder 使用此节点自动配置数据的返回类型。连接服务后，可以将服务操作绑定到用户界面组件。

连接 XML 服务文件

1 在 Flash Builder“ 数据 ” 菜单中，选择 “HTTP”，打开服务向导。

2 指定 “ 本地文件 ” 或 “URL”，然后浏览到文件。

3 在包含所需响应的文件中选择一个节点。

指明响应是否为 “ 数组 ”。

Flash Builder 将为所选节点配置返回类型。

4 修改服务详细信息。

服务名称	指定服务名称。 Flash Builder 会基于服务的文件名，为服务生成一个名称。 服务可以使用的名称受到限制。请参阅第 16 页的 “ 数据服务命名 ”。
服务包	为包含生成的 ActionScript 文件的包指定名称，这些文件用于访问服务。 Flash Builder 根据服务名称生成包，并将其放置在 services 包中。
数据类型包	为包含生成的 ActionScript 类文件的包指定名称，这些文件用于定义从服务检索到的数据类型。 默认情况下， Flash Builder 会创建 valueObjects 包。

5 （可选）修改为该服务生成的包名。

6 单击 “ 完成 ”。

Flash Builder 即生成访问该服务的 ActionScript 文件。

注： 在连接服务之后，可以修改服务属性。在 “ 数据 / 服务 ” 视图中选择服务。然后在上下文菜单中，选择 “ 属性 ”。

访问 Web 服务

可以通过 Flash Builder 服务向导连接 Web 服务 (SOAP)。您可以通过任何 Flex 项目连接 Web 服务。而不必为项目指定服务器技术。

如果要访问的服务与客户端应用程序的 SWF 文件位于不同的域中，则必须使用跨域策略文件。

更多帮助主题

Using cross-domain policy files

连接 Web 服务

1 在 Flash Builder“ 数据 ” 菜单中，选择 “Web 服务 ”，打开服务向导。

2 (BlazeDS/Data Services) 如果您已安装了 LiveCycle Data Services 或 BlazeDS，则可以通过代理访问 Web 服务。

选择 “ 通过 Blazeds/Data Services 代理的目标位置 ”。

指定目标位置。单击 “ 下一步 ”，并继续进行到步骤 5。

注： 仅当 Flex 项目指定 J2EE 作为应用程序服务器类型时，才会支持通过代理访问 Web 服务。

3 输入 SOAP 服务的 URI。

4 （可选）修改服务详细信息。

服务名称	指定服务名称。 Flash Builder 会根据 WSDL URI 为服务生成一个名称。 服务可以使用的名称受到限制。请参阅第 16 页的 “ 数据服务命名 ”。
服务包	为包含生成的 ActionScript 文件的包指定名称，这些文件用于访问服务。 Flash Builder 根据服务名称生成包，并将其放置在 services 包中。
数据类型包	为包含生成的 ActionScript 类文件的包指定名称，这些文件用于定义从服务检索到的数据类型。 默认情况下， Flash Builder 会创建 dataValues 包。

5 （可选）配置服务的代码生成：

服务 端口	从可用的服务中选择服务。 Flash Builder 会根据 WSDL URI 为服务生成一个名称。
操作列表	从您希望在客户端应用程序中访问的服务中，选择操作。

6 单击 “ 完成 ”。

Flash Builder 即生成访问该服务的 ActionScript 文件。

注： 在连接服务之后，可以修改服务属性。在 “ 数据 / 服务 ” 视图中选择服务。然后在上下文菜单中，选择 “ 属性 ”。

在连接 Web 服务之后，可配置服务操作的返回类型。有关详细信息，请参阅第 22 页的 “ 为数据服务操作配置数据类型 ”。

访问 BlazeDS

仅当已安装 Adobe® BlazeDS，并已配置 Remote Development Services (RDS) 服务器时，才可访问 BlazeDS 服务。有关安装和配置 BlazeDS 的信息，请参阅 LiveCycle Data Services ES 文档。

通常可从将 J2EE 配置为应用程序服务器类型的 Flex 项目访问 BlazeDS 数据服务。

更多帮助主题

第 6 页的 “ 创建 Flex 项目以访问数据服务 ”

连接 BlazeDS 服务

此过程假定您已安装 BlazeDS，已配置 Remote Development Server，并已创建用于访问 BlazeDS 服务的 Flex 项目。

1 在 Flash Builder“ 数据 ” 菜单中，选择 “ 连接 BlazeDS”，打开服务向导。

2 选择要导入的目标。

3 （可选）修改服务详细信息。

服务名称	指定服务名称。 Flash Builder 会根据目标为服务生成一个名称。 服务可以使用的名称受到限制。请参阅第 16 页的 “ 数据服务命名 ”。
服务包	为包含生成的 ActionScript 文件的包指定名称，这些文件用于访问服务。 Flash Builder 根据服务名称生成包，并将其放置在 services 包中。
数据类型包	为包含生成的 ActionScript 类文件的包指定名称，这些文件用于定义从服务检索到的数据类型。 默认情况下， Flash Builder 会创建 valueObjects 包。

4 单击 “ 完成 ”。

Flash Builder 即生成访问该服务的 ActionScript 文件。

注： 在连接服务之后，可以修改服务属性。在 “ 数据 / 服务 ” 视图中选择服务。然后在上下文菜单中，选择 “ 属性 ”。

访问 LiveCycle Data Services

仅当已安装了 LiveCycle Data Services，并且已配置了 Remote Development Services (RDS) 服务器时，才可访问 LiveCycle Data Services 提供的服务。有关信息，请参阅 LiveCycle Data Services 文档。

可以从将 J2EE 或 ColdFusion 配置为应用程序服务器类型的 Flex 项目访问 LiveCycle Data Services。

LiveCycle Data Services 的服务类型

连接到 LiveCycle Data Services 时，有下列类型的数据服务可以作为目标：

- 远程服务

  远程服务是使用 AMF 类型化实现的。这些服务不提供服务器端数据管理。可以使用 Flash Builder 工具配置客户端数据管理。请参阅第 27 页的 “ 启用数据管理 ”。

- 数据服务

  数据服务是实现服务器端数据管理的服务。有关更多信息，请参阅 LiveCycle Data Services 文档。

- Web 服务

可通过配置为 LiveCycle Data Services 目标的 LiveCycle Data Services 代理访问的 Web 服务。连接 Web 服务时通常不提供服务器端类型化。

数据类型配置和数据管理

Flash Builder 提供了用于配置和管理客户端数据的工具。可用的 Flash Builder 工具取决于 LiveCycle Data Services 目标的类型：

- 远程服务

  远程服务用于在服务上实现 AMF 类型化。无需为远程服务目标配置返回数据类型。

  但可以使用 Flash Builder 生成用于客户端数据管理的代码。请参阅第 27 页的 “ 启用数据管理 ”。

- 数据服务

  数据服务用于实现服务器端数据类型。无需为数据服务目标配置返回数据类型。

  数据服务目标还提供服务器端数据管理。请不要在数据服务目标中使用客户端数据管理。

- Web 服务

  通过 LiveCycle Data Services 代理获得的 Web 服务目标通常不实现服务器端类型化。可以使用 Flash Builder 工具为 Web 服务操作配置返回类型。请参阅第 22 页的 “ 为数据服务操作配置数据类型 ”。

  可以使用 Flash Builder 生成用于客户端数据管理的代码。请参阅第 27 页的 “ 启用数据管理 ”。

连接到 LiveCycle Data Services 目标 （数据服务和远程服务目标）

此过程假定您已安装了 LiveCycle Data Services，已配置了 Remote Development Server，并已创建了用于访问 LCDS 服务的 Flex 项目。

1 在 Flash Builder“ 数据 ” 菜单中，选择 “ 连接数据 / 服务 ” 打开服务向导。

2 在 “ 选择服务类型 ” 对话框中，选择 “LCDS”。单击 “ 下一步 ”。

3 必要时请提供登录凭据。

4 （可选）修改服务详细信息。

服务名称	如果您不提供服务名称， Flesh Builder 会生成一个服务名称。 Flash Builder 会根据目标为服务生成一个名称。
服务包	为包含生成的 ActionScript 文件的包指定名称，这些文件用于访问服务。 Flash Builder 根据服务名称生成包，并将其放置在 services 包中。
目标	指定可从 LiveCycle Data Services 服务器中获得的一个或多个目标。
数据类型包	指定数据类型包的名称。此包中包含生成的 ActionScript 类文件，这些文件用于定义从服务检索到的数据类型。 默认情况下， Flash Builder 会创建 valueObjects 包。

5 单击 “ 完成 ”。

Flash Builder 即生成访问该服务的 ActionScript 文件。

注： 在连接服务之后，可以修改服务属性。在 “ 数据 / 服务 ” 视图中选择服务。然后在上下文菜单中，选择 “ 属性 ”。

连接到 LiveCycle Data Services 目标 （Web 服务目标）

此过程假定您已安装了 LiveCycle Data Services，已配置了 Remote Development Server，并已创建了用于访问 DS 服务的 Flex 项目。

1 在 Flash Builder“ 数据 ” 菜单中，选择 “ 连接数据 / 服务 ” 打开服务向导。

2 在 “ 选择服务类型 ” 对话框中，选择 “Web 服务 ”。单击 “ 下一步 ”。

3 选择 “ 通过 LCDS/BlazeDS 代理的目标位置 ”。

4 必要时请提供登录凭据。

5 选择目标。

6 （可选）修改服务详细信息。单击 “ 下一步 ”。

服务名称	指定服务名称。 Flash Builder 会根据目标名称为服务生成一个名称。 服务可以使用的名称受到限制。请参阅第 16 页的 “ 数据服务命名 ”。
服务包	为包含生成的 ActionScript 文件的包指定名称，这些文件用于访问服务。 Flash Builder 根据服务名称生成包，并将其放置在 services 包中。
数据类型包	为包含生成的 ActionScript 类文件的包指定名称，这些文件用于定义从服务检索到的数据类型。 默认情况下， Flash Builder 会创建 dataValues 包。

7 （可选）配置服务的代码生成：

服务 端口	从可用的服务和端口选择服务和端口。
操作列表	从您希望在客户端应用程序中访问的服务中，选择操作。

8 单击 “ 完成 ”。

Flash Builder 即生成访问该服务的 ActionScript 文件。

注： 在连接服务之后，可以修改服务属性。在 “ 数据 / 服务 ” 视图中选择服务。然后在上下文菜单中，选择 “ 属性 ”。

更多帮助主题

第 6 页的 “ 创建 Flex 项目以访问数据服务 ”

数据服务命名

对于从 Flash Builder 访问的数据服务，服务可以使用的名称受到限制。但在编译应用程序之前，其中一些限制是不明显的。

服务的命名规则如下：

- 服务的首字母不能为数字。
- 服务名称不能是 ActionScript 关键字。
- 不要使用任何 ActionScript 类名 （包括自定义类）作为服务名称。
- （仅 PHP）如果服务名称包含下划线，则 Flash Builder 无法导入该服务。

注： 最佳选择是，使用与 MXML 文件名称不同的服务名称。

安装 Zend Framework

首次访问 PHP 服务时， Flash Builder 会判断是否安装了 Zend Framework 的支持版本。如果找不到 Zend Framework 的支持版本，Flash Builder 将提示您确认安装 Zend Framework。如果接受，Flash Builder 将安装 Zend Framework 的最低版本。如果拒绝，则需在访问 PHP 服务时手动安装 Zend Framework。

默认 Flash Builder 安装

Flash Builder 将 Zend Framework 安装到 Web 服务器根目录下的 ZendFramework 文件夹中。

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

对于访问 PHP 服务的 Flex 项目， Flash Builder 在项目输出文件夹中创建以下配置文件：

- amf_config.ini
- gateway.php

生产服务器

对于生产服务器， Adobe 建议将 ZendFramework 文件夹移出 Web 根文件夹。更新在 amf_config.ini 中定义的 zend_path 变量。

如果 zend_path 变量已被注释掉，将取消对 zend_path 变量的注释。指定 Zend Framework 的安装位置。

手动安装 Zend Framework

可以选择手动安装 Zend Framework。

1 下载 Zend Framework 的最新版本。

可以安装最小包或完整包。 Flash Builder 将安装最小包。

2 将下载的版本提取到系统上的某个位置。

3 在访问 PHP 服务的 Flex 项目的文件夹中，更新在 amf_config.ini 中定义的 zend_path 变量。

如果 zend_path 变量已被注释掉，将取消对 zend_path 变量的注释。指定 Zend Framework 安装位置的绝对路径。

Zend Framework 安装疑难解答

如果连接 Zend Framework 时出错，请按照以下提示解决错误。

手动安装 Zend Framework

如果是手动安装 Zend Framework，请检查 amf_config.ini 中的 zend_path 变量。

amf_config.ini 位于项目输出文件夹中。

请检查以下内容：

- 确保 zend_path 未被注释。
- 为 Zend Framework 安装指定的路径正确无误：
  - 此路径为指向本地文件系统上目标位置的绝对路径。不能指定指向映射网络资源的路径。
  - 此路径指向 Zend Framework 安装的库文件夹。库文件夹通常位于以下位置：

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

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

Flash Builder 安装 Zend Framework

如果由 Flash Builder 安装 Zend Framework，请检查以下内容：

- Web 根文件夹的位置

  Flash Builder 将 Zend Framework 安装在项目的 Web 根文件夹中。检查 Web 根文件夹的位置。选择 “ 项目 ”>“ 属性 ”>“Flex 服务器 ”。

- 确保将 Web 服务器配置为使用 PHP。
- 检查 amf_config.ini 中的 zend_path 变量。

  amf_config.ini 位于项目输出文件夹中。

  请检查以下内容：

  - zend_path 未被注释。
  - 指定的路径指向位于 Web 根文件夹中的 Zend Framework 安装。
  - 此路径为指向本地文件系统上目标位置的绝对路径。不能指定指向映射网络资源的路径。

使用单个服务实例

连接数据服务之后，项目中的每个应用程序都可以访问该服务。默认情况下，访问服务器时每个应用程序都会创建它自己的服务实例。

可以修改此行为，使项目中仅存在一个服务实例。因此，项目中的每个应用程序都会访问同一个服务实例。通常需要协调从多个应用程序访问数据时，可创建单个服务实例。

可以在某个项目内指定访问单个服务实例，也可以将其指定为所有项目的首选参数。

访问项目的单个服务实例

1 选择 “ 项目 ”>“ 属性 ”>“ 数据 / 服务 ”

2 启用用于使用单个服务实例的复选框。单击 “ 确定 ”。

将单个服务实例指定为首选参数

1 打开 “ 首选参数 ” 对话框。

2 选择 “Flash Builder”>“ 数据 / 服务 ”

3 启用用于使用单个服务实例的复选框。单击 “ 确定 ”。

构建客户端应用程序

可以使用 MXML 代码编辑器创建用户界面。

在使用代码编辑器定义了应用程序的组件后，您可以将从服务返回的数据绑定到用户界面组件。根据需要生成事件处理函数以便用户与应用程序进行交互。

也可以根据 “ 数据 / 服务 ” 视图中可用的服务操作生成表单。

将服务操作绑定到控件

使用“绑定到数据”对话框将服务操作绑定到用户界面组件。

可以从“数据 / 服务”视图中的工具栏上的“数据”菜单访问“绑定到数据”对话框。

在将服务操作绑定到组件时，Flash Builder 会生成 MXML 和 ActionScript 代码，用以从客户端应用程序访问服务操作。

服务操作的返回类型

将服务操作绑定到控件后，Flash Builder 将使用操作返回的数据的数据类型。通常情况下，在将服务操作绑定到组件之前，配置服务操作的返回类型。

如果尚未配置服务操作的返回类型，则“绑定到数据”对话框会提示您完成此步骤。

请参阅第 22 页的“为数据服务操作配置数据类型”。

将 DataGrid 控件绑定到服务操作 （通过“绑定到数据”对话框）

此过程假定您已连接数据服务。

1 在大纲视图中，选择 DataGrid 控件。或者，在 MXML 编辑器中，将光标置于 <s:DataGrid> 标签中。

2 在选中 DataGrid 的情况下，通过从 Flash Builder“数据”菜单选择“绑定到数据”打开“绑定到数据”对话框。

3 选择“新服务调用”，然后选择所需的服务和操作。

如果先前已将服务操作绑定到组件，则可以使用这些结果。在这种情况下，可指定“现有调用结果”，并选择要使用的操作。

4 （可选）选择“更改返回类型”：

如果要重新配置服务操作的返回类型，请选择“更改返回类型”。

如果以前未配置操作的返回类型，请选择“配置返回类型”。

请参阅第 22 页的“为数据服务操作配置数据类型”。

5 单击“确定”。

DataGrid 组件会更改为显示从数据库中检索到的字段。

请参阅配置 DataGrid 和 AdvancedDataGrid 组件。

6 保存并运行该应用程序。

生成对操作的服务调用

Flash Builder 可生成调用服务操作的 ActionScript 方法。此方法未绑定到用户界面组件，但可在应用程序代码中使用。

除了生成 ActionScript 方法之外，Flash Builder 还会创建 CallResponder，用于访问从服务调用返回的数据。请参阅 第 31 页的“CallResponder”。

生成对操作的服务调用

此过程假定您已连接数据服务。

1 在“数据 / 服务”视图中，选择操作。

2 在操作的上下文菜单中，选择“生成服务调用”。

Flash Builder 会生成用于调用操作的方法，并在 MXML 编辑器的源代码模式下显示生成的方法。Flash Builder 会创建一个 CallResponder，用于放置服务调用的结果。

此选项也可从“数据 / 服务”工具栏进行访问。

为应用程序生成表单

表单是 Web 应用程序用来从用户收集信息最常见的方法之一。通过 Flash Builder，可以为从服务调用中检索到的数据生成表单，也可以为用于访问远程数据的自定义数据类型生成表单。

生成表单时， Flash Builder 会创建一个表单布局容器，并添加组件以显示或编辑从服务中检索到的特定数据。

Flash Builder 生成以下类型的表单。

表单	描述
数据类型	包含表示数据类型的字段的组件。
Master-detail 表单	“master” 组件通常是列出从服务中检索的数据的数据控件。 “detail” 表单表示在 master 组件中选中的各个项目。
服务调用	创建两个表单。一个表单用于指定操作的输入。另一个表单用于显示返回的数据。

生成表单时，请指定要包含哪些字段，用于表示每个字段的用户界面控件的类型，以及表单是否可编辑。

生成表单

此过程说明如何为服务调用生成表单。其他类型的表单的生成过程与此相似。

1 要运行 “ 生成表单 ” 向导，请从 “ 数据 / 服务 ” 视图中选择一个操作。然后，执行以下操作之一：

- 在操作的上下文菜单中，选择 “ 生成表单 ”。
- 在 Flash Builder 的 “ 数据 ” 菜单中，选择 “ 生成表单 ”。

2 在 “ 生成表单 ” 向导中，选择 “ 为服务调用生成表单 ”。

3 选择 “ 新服务调用 ” 或 “ 现有调用结果 ”。

指定 “ 现有调用结果 ” 以使用先前为服务调用生成的代码。

否则，请指定 “ 新服务调用 ”，并为表单选择服务和操作。

4 （可选）根据操作的不同，对于生成的表单，有以下多种选项。

如果操作接受参数，则可以选择包含一个用于这些参数的表单。

如果操作返回某个值，则可以选择包含一个用于该返回值的表单。

可以选择表单为可编辑还是只读。

5 （可选）配置输入类型或返回类型。

如果选择的操作有输入参数或者返回某个值，则可以配置输入类型或返回类型。

配置操作的输入类型和返回类型后，才能生成表单。如果先前已配置这些类型，则可以选择再次配置它们。

请参阅第 22 页的 “ 为数据服务操作配置数据类型 ”。

6 单击 “ 下一步 ”。在 “ 属性控件映射 ” 对话框中，选择要在表单中包含的字段，以及用于表示数据的控件类型。

7 单击 “ 完成 ”。

Flash Builder 生成表单时，一个表单可能会放在另一个表单之上。要重新排列生成的表单，请确保您选择并移动的是表单，而不是表单中的一个组件。

如果 Flash Builder 将一个表单放在另一个表单之上，则如何选择表单可能会成为一个难题。在代码编辑器中，选中其中一个表单的标签。

生成 master-detail 表单

要创建 master-detail 表单，首先要将数据控件组件添加到应用程序，并将操作的结果绑定到控件。

例如，添加 DataGrid 组件，并将操作 （如 getItems_paged()）的结果绑定到 DataGrid。

1 在大纲视图中，选择一个数据控件，例如 DataGrid。

2 在“ 数据 ”菜单中，选择“ 生成详细信息表单 ”。

3 按“ 生成表单 ”中所述，继续生成表单。

为数据类型生成表单

要使用表示某个自定义数据类型的字段的组件生成表单，请首先配置该数据类型。请参阅第 22 页的“ 为数据服务操作配置数据类型 ”。

1 在“ 数据 / 服务 ”视图中，选择自定义数据类型。

2 在上下文菜单中，选择“ 生成表单 ”。

3 确保选中“ 为数据类型生成表单 ”，然后选择一种数据类型。

4 （可选）可以选择表单是否可编辑。

5 单击“ 完成 ”。

生成事件处理函数以检索远程数据

将数据服务操作绑定到组件之后， Flash Builder 会生成一个事件处理函数，用于从服务检索数据以填充组件。

例如，如果将某一操作 （如 getAllItems()）绑定到 DataGrid，则 Flash Builder 会生成 creationComplete 事件处理函数。DataGrid 将引用生成的事件处理函数。因此，调用的结果将成为 DataGrid 的数据提供程序。

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

在创建 DataGrid 后，事件处理函数会在运行应用程序时使用从服务中检索的数据填充 DataGrid。

生成事件处理函数时，可以根据需要接受生成的处理函数，也可以使用其它事件处理函数替换它们。例如，可以将 DataGrid 的 creationComplete 事件处理函数替换为 Application 的 creationComplete 处理函数。

也可以为接受用户输入的控件 （如 Buttons 或 Text）生成或创建事件处理函数。

为用户界面组件生成事件处理函数

1 创建包含用户界面组件 （如 DataGrid 或 Button）的应用程序。

2 Flash Builder 提供了“内容辅助”来帮助您生成事件处理函数。按 Control+Space 或 Cmd+Space (Mac) 并选择“生成事件处理函数 ”。

3 Flash Builder 将为事件处理函数生成唯一名称，并将事件处理函数放在 Script 块中。

Flash Builder 在代码编辑器中会为事件处理函数加亮生成的存根。为事件处理函数填写剩余代码。使用“ 内容辅助 ”来帮助您为事件处理函数编写代码。

为数据服务操作配置数据类型

在连接数据服务时，Flash Builder 需要了解服务操作所返回数据的数据类型。它支持能够被 AMF 识别的数据类型，以便与数据服务或远程服务交换数据。

许多数据服务都在服务器端定义所返回数据的类型 （服务器端类型化）。但是，如果未在服务器端定义所返回数据的类型，则客户端应用程序必须为返回的数据配置类型 （客户端类型化）。

指定参数的服务操作必须同时指定与在服务上访问的数据相对应的类型。使用客户端类型化，可为输入参数配置类型。

为客户端类型化配置类型时， Flash Builder 仅识别 AMF 数据类型。这些类型也可以是自定义数据类型 （表示复杂数据）或者 void （指示该操作不返回任何数据）。

可以为返回复杂数据的服务操作配置用户定义的类型。例如，如果要从员工数据库中检索记录，则可将复杂数据返回定义为 “Employee”。在这种情况下， Employee 的自定义数据类型会包含数据库记录中每个字段的所有条目。

客户端类型化的数据类型

数据类型	描述
ActionScript 类型	Boolean Boolean[] ByteArray ByteArray[] Date Date[] int int[] Number Number[] Object Object[] String String[]
不返回数据	void
用户定义的类型	CustomType CustomType[]

用户定义的类型 (Employee)

字段	数据类型
emp_no	Number
first_name	String

字段	数据类型
last_name	String
hire_date	Date
birth_date	Date

验证对服务的访问权限

通常情况下，用户必须经过身份验证才能访问数据服务。使用 HTTP 协议提供访问的 PHP、BlazeDS 和 ColdFusion 服务可能还需要其它身份验证。在某些情况下，这些类型的服务同时需要 HTTP 身份验证和远程身份验证。

在执行以下操作时， Flash Builder 会提供一个用于服务身份验证的选项：

- 为操作配置返回类型

  请参阅第 23 页的 “ 配置数据从操作返回时的类型 ”。

- 使用 “ 测试操作 ” 界面

  请参阅第 25 页的 “ 测试服务操作 ”。

指定 “ 需要身份验证 ” 时， Flash Builder 会打开 “ 服务身份验证 ” 对话框。根据访问的服务类型，可以指定 “ 基本身份验证 ” 或 “ 远程身份验证 ”。

基本身份验证

基本身份验证用于访问 HTTP 服务和 Web 服务。要访问这些服务，必须提供用户名和密码。

如果需要 Flash Builder 在整个会话中使用指定的凭据，请指定 “ 记住用户名和密码 ”。

远程身份验证

远程身份验证用于访问远程对象服务。远程对象服务是使用 ColdFusion、PHP、BlazeDS 或 LiveCycle Data Services 作为远程对象实现的服务。

Flash Builder 不会为未实现远程对象服务的项目提供远程身份验证登录界面。

要访问这些远程对象服务，必须提供用户名和密码。

如果需要 Flash Builder 在整个会话中使用指定的凭据，请指定 “ 记住用户名和密码 ”。

配置操作的输入参数

对于客户端类型化，需要配置从数据服务中获得的操作的输入参数。

以下过程假定在 Flash Builder 中已连接数据服务，且数据服务具有需要可配置输入参数的操作。

1 在 “ 数据 / 服务 ” 视图中，选择包含可配置输入参数的操作。在操作的上下文菜单中，选择 “ 配置输入类型 ”。

2 对于操作的每个参数，在 “ 配置输入类型 ” 对话框中，从可用类型列表中选择一种数据类型。单击 “ 确定 ”。

  如果先前已为服务定义自定义返回数据类型，则可以选择这些类型。

对于服务器端类型化，服务指定输入参数的数据类型。

配置数据从操作返回时的类型

定义操作返回的数据类型的服务提供服务器端类型化。如果服务未定义操作返回的数据类型，则 Flash Builder 会使用客户端类型化来定义返回的数据类型。

Flash Builder 会对从服务操作返回的数据进行内部检查，以确定数据类型。您可以通过以下两种方式配置数据从操作返回时的类型：

- 自动检测示例数据的返回类型

  如果服务实现服务器端类型化，则 Flash Builder 会检测服务所定义的数据类型。

  如果服务未实现服务器端类型化， Flash Builder 会为客户端应用程序创建自定义类型。对于客户端类型化，您需要为自定义类型提供一个名称。通常使用该名称描述返回的数据。例如，如果操作从数据库表返回一个书籍数组，则您会将该类型命名为 “Book”。

- 使用现有类型

  现有类型可以是由服务所定义的类型、 ActionScript 类型或先前定义的自定义类型。

数据服务的类型不同， Flash Builder 用于对数据进行内部检查的过程会稍有差异。例如，对 HTTP 服务的返回类型进行内部检查和配置的过程会与针对 PHP 或 ColdFusion 服务的过程不同。

合并和更改数据类型

在对服务器数据进行内部检查的过程中，可以合并来自其它数据类型的字段，也可以基于现有数据类型创建数据类型。可以通过以下方式修改自定义数据类型：

- 对现有数据类型使用新名称

  如果要在客户端应用程序中以不同的方式使用返回的数据，请使用新名称。

  例如，在客户端应用程序中检索可以在员工摘要和员工详细信息表中使用的员工数据。

- 合并字段

  可以将返回的字段添加到现有数据类型。当关联来自多个源的数据时，添加附加的字段很有用。例如，用于返回从多个数据库表中检索的数据的 JOIN 操作。

  另一个示例是从不同的服务收到的数据。例如，合并从 HTTP 服务和 ColdFusion 服务收到的 Book 数据。

配置自定义数据类型 （PHP 或 ColdFusion 服务）

此过程假定已连接使用 PHP 或 ColdFusion 实现的数据服务。

1 在 “ 数据 / 服务 ” 视图中，从操作的上下文菜单中选择 “ 配置返回类型 ”。

2 如果操作包含参数，请输入参数值。并为参数指定正确的数据类型。

3 （新自定义类型或修改过的自定义类型）选择自动检测此操作返回的数据类型。

  如果服务需要进行身份验证，请选择 “ 需要身份验证 ”，并在必要时提供凭据。请参阅第 23 页的 “ 验证对服务的访问权限 ”。

  Flash Builder 会对操作进行内部检查，并构建一种自定义数据类型。

  为自定义数据类型指定名称。

  如果先前已定义自定义数据类型，则可以选择将返回的字段添加到现有自定义数据类型的定义中。

4 （使用现有类型）使用此选项可指定 ActionScript 类型或先前配置的某个类型。

5 单击 “ 完成 ”。

配置自定义数据类型 （HTTP 服务）

此过程假设您已连接 HTTP 服务。

1 在 “ 数据 / 服务 ” 视图中，从操作的上下文菜单中选择 “ 配置返回类型 ”。

2 （新自定义类型）选择自动检测此操作返回的数据类型。

如果服务需要进行身份验证，请选择“需要身份验证”，并在必要时提供凭据。

Flash Builder 会对操作进行内部检查，并构建一种自定义数据类型。选择 Flash Builder 为操作传递参数值一种方法，然后单击“下一步”：

- （输入参数值）分别为每个参数指定一个值。

  还可以指定参数的数据类型。 Flash Builder 会自动选择默认数据类型。

- （输入服务 URL）输入 HTML 服务的 URL，包括 URL 中的参数和值。例如：

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

- （输入 XML/JSON 响应）将 XML/JSON 响应复制到文本框。

  脱机时，或者 HTTP 服务仍处于开发中但您知道响应来自服务器时，使用此选项。

3 （新自定义类型，续）为自定义数据类型指定名称，或从返回的数据选择一个节点。

如果从返回的数据选择了一个节点，则 Flash Builder 会为针对该节点返回的数据创建自定义数据类型。

指示返回的数据是否是以数组形式返回的。

如果服务返回 XML 文件，“选择根节点”下拉列表将处于启用状态。从 XML 文件中选择节点以指定数据类型。

4 （使用现有类型）使用此选项可指定 ActionScript 类型或先前配置的某个类型。

5 单击“完成”。

测试服务操作

可以使用 Flash Builder 测试服务操作，并查看从操作返回的数据。此功能对于验证服务的行为很有用。

重要说明： 某些操作 （如更新和删除）会修改服务器上的数据。

测试服务操作

此过程假设您已连接数据服务。

1 在“数据 / 服务”视图中，选择服务中的某个操作。在上下文菜单中，选择“测试操作”。

“测试操作”视图会打开，显示选定的操作。如果操作需要输入参数，则“测试操作”视图会列出这些参数。

2 对于所有必需的输入参数，单击“输入值”字段，并为参数指定值。

如果参数需要复杂类型，请单击该字段中的省略号按钮以打开输入参数编辑器。在编辑器中指定值。

输入参数编辑器接受 JSON 表示法，用于表示 ActionScript 中的复杂类型。

3 如果需要从服务器进行身份验证，请选择“需要身份验证”。单击“测试”。

必要时提供身份验证凭据。请参阅第 23 页的“验证对服务的访问权限”。

Flash Builder 将显示从服务返回的数据。

4 （可选）在“测试操作”视图中，选择要测试的其它服务和操作。

管理对服务器中的数据的访问

分页 分页是指对来自远程服务的大型数据集进行增量检索。

例如，假设您希望访问某个包含 10,000 条记录的数据库，并希望在包含 20 行的 DataGrid 中显示这些数据。您可以执行分页操作，以 20 条为单位以增量方式访存这些行。当用户请求更多数据时 （在 DataGrid 中滚动），将访存和显示下一页记录。

数据管理 在 Flash Builder 中，数据管理是指将更新数据从客户端应用程序同步到服务器。通过使用数据管理功能，您可以修改客户端应用程序中的一个或多个项目，而不必对服务器进行任何更新。然后，通过一个操作将所有更改提交到服务器中。您还可以还原所做的修改，而不更新任何数据。

数据管理涉及到协调多个操作 （例如创建、获取、更新和删除），以响应在客户端应用程序中发生的事件 （例如更新员工记录）。

如果在 Flash Builder 中启用了数据管理，则 Flash Builder 还生成可自动更新用户界面组件的代码。例如， Flash Builder 生成代码以使 DataGrid 与服务器上的数据同步。

启用分页

可以为使用以下签名实现分页函数的数据服务启用分页：

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

函数名称	可以使用任何有效的函数名称。
startIndex	要检索的数据的初始行。 在客户端操作中，将 startIndex 的数据类型定义为 Number。
numItems	每页要检索的数据行数。 在客户端操作中，将 numItems 的数据类型定义为 Number。
myDataType	数据服务返回的数据类型。

在服务中实现分页时，还可以实现 count() 操作。count() 操作返回从服务中返回的项目数。Flash Builder 要求 count() 操作实现以下签名：

```
count(): Number
```

函数名称	可以使用任何有效的函数名称。
Number	从该操作检索的记录数。

Flex 使用 count 操作正确地显示用来检索大型数据集的用户界面组件。例如， count() 操作可帮助确定 DataGrid 滚动条的滑块大小。

某些远程服务不提供 count() 操作。分页功能仍然有效，但是用来显示已分页数据的控件无法正确地显示数据集的大小。

对已过滤的查询启用分页操作

可以为从数据库过滤结果的查询启用分页。在查询中过滤结果时，请将以下签名用于分页和计数函数。

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

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

filterParam1	可选过滤器参数。此参数在 getItems_PagedFiltered() 和 countFiltered() 中相同。
filterParam2	可选过滤器参数。此参数在 getItems_PagedFiltered() 和 countFiltered() 中相同。

以下是使用 PHP 实现的 getItems_pagedFiltered() 函数的一个代码片断，用于访问 MySQL 数据库。该代码片断显示如何使用可选过滤器参数。

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

为操作启用分页

此过程假设您已在远程服务中对 getItems_paged() 和 count() 操作进行了编码。同时假设您已按照第 22 页的 “ 为数据服务操作配置数据类型 ” 中所述，为操作配置了返回数据类型。

1 在 “ 数据 / 服务 ” 视图中，从 getItems_paged() 操作的上下文菜单中，选择 “ 启用分页 ”。

2 如果先前没有为数据类型标识唯一关键字，请指定唯一标识此数据类型的实例的属性。单击 “ 下一步 ”。

此属性通常是主关键字。

3 （可选）指定要访存的项目数以定义自定义页面大小。

如果未指定自定义页面大小，则会在服务级别设置默认页面大小。默认页面大小为每页 20 条记录。

4 （可选）指定 count() 操作。单击 “ 完成 ”。

通过使用 count() 操作， Flash Builder 可正确显示用户界面元素，如滚动条的滑块大小。

现在即为该操作启用了分页。

在 “ 数据 / 服务 ” 视图中，用来实现分页的函数的签名不再包括 startIndex 和 numItems 参数。 Flash Builder 现在以动态方式添加这些值。 Flash Builder 将根据您提供的自定义页面大小或默认页面大小 （每页 20 条记录）确定这些值。

启用数据管理

要为服务启用数据管理，服务需要实现下列一个或多个函数：数据管理功能使用以下函数来同步远程服务器上的数据更新：

- 添加 (createItem)

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

createItem() 的返回类型为数据库主键的类型。

- 获取所有属性 (getItem)

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

- 更新 (updateItem)

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

- 删除 (deleteItem)

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

Flash Builder 要求上述函数具有以下签名：

函数名称	可以使用任何有效的函数名称。
item **originalItem**	数据服务返回的数据类型项。
itemID	项的唯一标识符，通常为数据库中的主关键字。
changes[]	与指定项中的字段相对应的数组。此参数只在 updateItem() 的一个版本中使用。
myDataType	可从数据服务中获得的项的数据类型。通常，在从服务检索数据时，需要定义一个自定义数据类型。

autoCommit 标志

使用数据管理可同步对服务器上的数据的更新。在调用 ***service***.commit() 方法之前，不会在服务器上更新在客户端应用程序中对数据所做的更改。

但是，如果要禁用此功能，请将 autoCommit 标志设置为 true。如果 autoCommit 为 true，则不会缓存对服务器数据的更新，而会立即进行更新。请参阅第 33 页的 “ 为服务启用数据管理 ”。

deleteItemOnRemoveFromFill 标志

使用已启用的数据管理删除各个项时，请使用 deleteItemOnRemoveFromFill 标志。默认情况下，该标志设置为 true。删除某一项时，该项立即从数据库中删除。

如果将 deleteItemOnRemoveFromFill 设置为 false，则会延迟删除，直到调用 commit() 方法为止。以下示例显示了针对 DataGrid 的 creationComplete 事件处理函数的代码：如果用户删除 DataGrid 中的某一选定 Employee 项，则在调用 commit() 方法之前，该选定项不会从数据库中删除。

```
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();
}
```

为操作启用数据管理

此过程假设您已经在远程服务中实现了所需的操作。同时假设您已经为使用自定义数据类型的操作配置了返回数据类型。请参阅第 22 页的 “ 为数据服务操作配置数据类型 ”。

1 在 “ 数据 / 服务 ” 视图中，展开 “ 数据类型 ” 节点。

2 在数据类型的上下文菜单中，选择 “ 启用数据管理 ”。

3 如果先前没有为数据类型标识唯一关键字，请指定唯一标识此数据类型的实例的属性。单击 “ 下一步 ”。

   此属性通常是主关键字。

4 指定已实现的添加、获取所有属性、更新和删除操作。单击 “ 完成 ”。

   注：您不必实现所有这些函数。仅需实现应用程序所需的那些函数。

现在即为操作启用了数据管理。

Flash Builder 为客户端应用程序生成代码

Flash Builder 可生成用于访问远程服务操作的客户端代码。 Flash Builder 在下列环境中生成代码：

- 连接到数据服务时
- 在“ 数据 / 服务 ”视图中刷新数据服务时
- 为操作配置返回类型时
- 将服务操作绑定到用户界面控件
- 为服务操作启用分页
- 为服务启用数据管理
- 生成事件处理函数或服务调用时

服务类

通过服务向导连接数据服务。连接服务时， Flash Builder 会生成一个 ActionScript 类文件，用于访问服务操作。

对于访问 RemoteObject 的服务，生成的类会扩展 RemoteObjectServiceWrapper 类。使用 PHP、 ColdFusion、BlazeDS 和 LiveCycle Data Services 实现的服务通常会访问 RemoteObject。

对于 HTTP 服务，生成的类会扩展 HTTPServiceWrapper 类。

对于 Web 服务，生成的类会扩展 WebServiceWrapper 类。

Flash Builder 基于在服务向导中为服务提供的名称，为生成的类文件创建一个名称。默认情况下， Flash Builder 将此类放置在项目的主源文件夹中。通常此文件夹为 src。包的名称基于服务名称。例如， Flash Builder 为 EmployeeService 类生成下列 ActionScript 类。

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

超类包含 EmployeeService 类的实现。

请勿对超类进行编辑，因为它是生成的类。对超类所做的修改可能会被覆盖。对该实现所做的任何更改都可能会导致发生未定义的行为。

在此示例中，将使用 EmployeeService.as 扩展生成的超类，并添加您的实现。

更多帮助主题

第 7 页的“ 连接数据服务 ”

自定义数据类型的类

许多远程数据服务提供服务器端类型化。这些服务将复杂数据返回为自定义数据类型。

对于不返回已设置类型的数据的远程数据服务，Flash Builder 提供客户端类型化。使用客户端类型化，可通过 Flash Builder 连接向导，为服务返回的复杂数据定义和配置数据类型。例如，对于返回员工数据库记录的服务，可以定义和配置一个 Employee 数据类型。

Flash Builder 将为服务返回的每个自定义数据类型的实现生成一个 ActionScript 类。 Flash Builder 使用此类创建 value 对象，然后使用这些对象访问远程服务数据。

例如， Flash Builder 为包含 Employee 数据类型的 EmployeeService 类生成下列 ActionScript 类:

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

这些超类分别包含 EmployeeService 和 Employee 数据类型的实现。

请勿对生成的超类进行编辑。对超类所做的修改可能会被覆盖。对该实现所做的任何更改都可能会导致发生未定义的行为。

在此示例中，将使用 EmployeeService.as 和 Employee.as 扩展生成的超类，并添加您的实现。

将服务操作绑定到用户界面控件

第 19 页的 “ 将服务操作绑定到控件 ” 介绍了如何将从服务操作返回的数据绑定到用户界面控件。将服务操作绑定到控件时，Flash Builder 会生成下列代码:

- 包含 CallResponder 和服务标签的声明标签
- 用于调用服务调用的事件处理函数
- 控件与从操作返回的数据之间的数据绑定

声明标签

声明标签是一个 MXML 元素，用于声明当前类的非默认、非可视属性。将服务操作绑定到用户界面时，Flash Builder 会生成包含 CallResponder 和服务标签的声明标签。 CallResponder 和生成的服务类是容器元素的属性，该元素通常是 Application 标签。

以下示例显示了一个声明标签，该标签可提供对远程 EmployeeService 的访问:

```
<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 管理对服务调用的结果。它包含的标记属性可设置为由服务调用返回的 Async 标记。 CallResponder 包含的另一个 lastResult 属性可设置为来自服务调用的最后一个成功的结果。将事件处理函数添加到 CallResponder，以便访问通过 lastResult 属性返回的数据。

当 Flash Builder 生成 CallResponder 时，会基于它所绑定到的服务操作的名称生成 ID 属性。以下代码示例显示了 EmployeeService 的两个操作的 CallResponder。 getAllItems() 操作已绑定到 DataGrid 的 creationComplete 事件处理函数。 delete 操作已绑定到 DataGrid 中的选定项目。 DataGrid 会在创建 getAllItems() 服务调用后，立即显示从该服务调用检索到的项目。 Delete Item Button 控件可从数据库删除 DataGrid 中的选定记录。

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

事件处理函数

将服务操作绑定到用户界面组件时， Flash Builder 会为 CallResponder 生成一个事件处理函数。该事件处理函数管理操作的结果。还可以在 ActionScript 代码块中创建事件处理函数，并从用户界面组件的属性引用该事件处理函数。

通常，使用从服务返回的数据填充诸如 List 和 DataGrid 等控件。默认情况下， Flash Builder 会为创建之后立即激发的控件生成 creationComplete 事件处理函数。对于其它控件， Flash Builder 会为控件的默认事件生成处理函数。例如，对于 Button， Flash Builder 会为 Button 的 click 事件生成处理函数。

该控件的事件属性会设置为生成的事件处理函数。以下示例显示了为 DataGrid 生成的 creationComplete 事件处理函数:

```
<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>
```

可以为响应用户事件的控件 （如 Button）生成事件处理函数。以下示例显示了为填充 DataGrid 的 Button 生成的事件处理函数。

```
<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)"/>
```

数据绑定

构建用户界面时，将服务操作绑定到控件。请参阅第 19 页的 “ 将服务操作绑定到控件 ”。

Flash Builder 可生成代码，用于将从服务操作返回的数据绑定到显示数据的用户界面控件。

以下示例显示了 Flash Builder 生成的代码，该代码用于填充 DataGrid 控件。对于自定义数据类型 Employee，getAllItems() 操作将返回一组员工记录。

DataGrid 的 dataProvider 属性被绑定到存储在 CallResponder 中的结果 getAllItemsResult。 Flash Builder 使用与为 Employee 记录返回的每个字段相对应的 DataGridColumn 自动更新 DataGrid。每一列的 headerText 和 dataField 属性是根据在 Employee 记录中返回的数据设置的。

```
<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>
```

为服务操作启用分页

启用分页时， Flash Builder 可修改生成的服务的实现。使用分页的数据填充数据控件 （如 DataGrid 或 List）时， Flash Builder 将确定在数据控件中可见的记录数以及数据库中的记录总数。Flash Builder 将这些值作为参数提供给用于实现分页的服务操作。

启用分页之后，您不必修改任何客户端应用程序代码。

有关更多信息，请参阅第 26 页的 “ 启用分页 ”。

为服务启用数据管理

在 Flash Builder 中，数据管理是指同步对服务器上数据的一组更新。可以为从服务返回的自定义数据类型启用数据管理。启用数据管理后，可以修改客户端应用程序中的一个或多个项目，而不必对服务器进行任何更新。随后即可通过一个操作将所有更改提交到服务器中。还可以还原所做的修改，而不更新服务器上的任何数据。第 27 页的 “ 启用数据管理 ” 介绍了如何实现此功能。

启用数据管理时， Flash Builder 可修改生成的服务类和为自定义数据类型生成的类的实现。 Flash Builder 会创建一个 DataManager 来实现此功能。

同步服务器数据的更新

为托管数据类型调用服务操作时，所做更改会反映在客户端应用程序中。但可以指定在调用 DataManager 的 commit() 方法之前，不更新服务器上的数据。

如果为服务启用了数据管理，则服务会具有 autoCommit 标志。默认情况下， autoCommit 为 false。

autoCommit 标志确定是立即提交更改，还是等待 ***service***.commit() 被调用。

如果 autoCommit 为 false，则会在调用 ***service***.commit() 之前，缓存客户端应用程序中的所有服务更新，您可以调用服务的 revertChanges() 方法来废弃更改。

如果 autoCommit 为 true，则更新会立即发送到服务器，此时，不能通过调用 revertChanges() 来废弃更改。

deleteItemOnRemoveFromFill 标志确定删除的项是否立即从数据库中删除。如果设置为 true，则在调用 ***service***.commit() 之前不会删除该项。

以下代码禁止对服务器数据更新进行数据管理同步。对托管类型数据所做的更改会在服务器上立即更新。

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

下列代码支持对服务器数据更新进行数据管理同步。直到针对该服务调用了 commit() 之后，才更新对托管类型数据所做的更改。另外，直到调用了 commit() 之后，删除的项才从数据库中删除。

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

还原更改

DataManager 提供了一种 revertChanges() 方法。通过 revertChanges() 方法，可将客户端应用程序中显示的数据还原为最后一次 commit 调用之前从服务器检索到的值。

请在调用 commit() 之前调用 revertChanges()，从而还原对客户端应用程序中托管数据类型的更改：

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

要提交对托管数据类型所做的更改，请调用 commit() 方法。

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

也可以直接从 bookService 实例调用 commit() 方法。直接从服务实例调用 commit 方法会提交对所有托管数据类型所做的所有更改。

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

注： 无法直接从服务实例调用 revertChanges() 来还原对所有托管数据类型的更改，仅可以为特定托管数据类型调用它。

如果要覆盖数据管理的默认行为，并禁用还原更改的功能，请将 autoCommit 设置为 true。例如，如果有一个实例 bookService，并已为 “Book” 数据类型启用了数据管理，请将 autoCommit 设置为 true：

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

此时，对托管类型数据所做的更改会在服务器上立即更新。

部署访问数据服务的应用程序

将应用程序从开发环境移动到部署环境时，需要考虑许多因素。部署应用程序的过程取决于应用程序、应用程序要求和部署环境。

例如，在仅供公司员工访问的内部网站上部署应用程序的过程，与在公共网站上部署相同应用程序的过程是不同的。

Deploying applications 中概述了考虑事项及 Deployment checklist。该清单讨论一些常见的系统配置问题，这些问题都是客户在生产环境中部署应用程序时遇到的问题。该文档还包含用于诊断常见部署问题的疑难解答提示。

编写访问服务的代码的最佳做法

使用 Flash Builder 工具，可以生成客户端代码来访问数据库中的数据。PHP 和 ColdFusion 都可以使用此功能。但此代码仅可用作原型。不要将此代码用作用于编写安全应用程序的模板。

默认情况下，通过此代码，任何能够通过网络访问服务器者，都可以在数据库表中执行插入、选择、更新或删除操作。在修改生成的代码或编写访问服务的代码时，以下做法比较好。有关更多信息，请参阅保护数据服务。

删除未使用的函数

删除或注释掉不打算在应用程序内使用的所有函数。

添加身份验证

添加用户身份验证以确保仅受信任的用户可以访问数据库信息。

添加授权检查

如果需要进行身份验证，请添加授权检查。即使已对应用程序用户进行了身份验证，也仍需要确保这些用户有权进行特定查询。

例如，可以授权所有人具有选择权限，但限制哪些用户具有删除权限。

另一个示例是授权用户 A 使用 select 查询检索自己的信息。但禁止用户 A 使用 select 查询来访问用户 B 的信息。

数据验证

请务必添加数据验证。例如，验证提供给任何 insert 语句的数据，以确保数据库不会接受错误数据或恶意数据。

客户端验证无法防止有人向您的 Web 服务器发送手动请求。数据验证可防止黑客进入，并确保存储的信息的质量。

限制所检索的数据量

使用 select 方法可以选择表中的所有内容。某些情况下，这样做会导致通过网络传输的信息过多。仅检索所需的数据。

例如，用户表中的 SELECT * 可以通过网络返回用户名和密码。

对敏感数据使用 SSL

使用安全协议可确保所发送信息的保密性。

导出含应用程序发行版的源文件

导出应用程序发行版时，Flash Builder 提供“启用查看源代码”选项。通过此选项，用户可以查看实现应用程序的源文件。服务器项目的源文件包括 services 文件夹，此文件夹中包含用于访问服务实现的文件。

重要说明：借助“查看源代码”选项包括服务文件时需谨慎。服务文件包含有关访问数据库的详细信息，也可能包括用户名和密码等敏感信息。如果在“查看源代码”选项中包括服务，则能够访问已启动应用程序的任何用户都可以访问敏感数据。

更多帮助主题

Flex 安全性

编写安全服务

Adobe 文档中的示例（包括教程和使用 Flash Builder 代码生成创建的应用程序）本身具有指导性，它们说明如何从客户端应用程序访问数据服务。但这些示例通常是为了确保能够清晰地说明问题，并不是对数据进行安全访问的最佳做法。

Flash Builder 文档包括使用生成的代码创建的应用程序，这些示例将部署在受信任的开发环境中。受信任的开发环境可以是本地计算机或受防火墙保护的位置。如果没有采取附加安全措施，任何具有网络访问权限的人都可以访问您的数据库。

编写服务时，最佳做法包括：

- 调用服务上的任何方法之前，对用户进行身份验证。
- 使用服务身份验证以仅允许特定用户执行特定操作。

  例如，假定您有一个应用程序，可通过 RemoteObject 调用修改员工数据。这种情况下，使用 RemoteObject 身份验证可确保仅经理可以更改员工数据。
- 使用编程安全限制对服务的访问权限。
- 向整个服务应用声明性安全约束。
- 访问 Web 服务 (<mx:WebService>) 或 HTTP 服务 (<mx:HTTPService>) 时，必须满足以下条件之一：
  - 服务是在与调用它的应用程序相同的域中实现的。
  - 服务的主机系统具有一个 crossdomain.xml 文件，该文件显式允许从应用程序的域进行访问。

更多帮助主题

Flex 安全性

保护数据服务

编写安全应用程序

Adobe® Flash® Player 可运行使用 Flash 构建的应用程序。其内容是作为一系列指令使用二进制格式交付给 Flash Player 的，使用精确描述的 SWF 文件格式通过 Web 协议进行交付。SWF 文件本身通常位于服务器上，在请求时下载到客户端计算机并在其上显示。大多数内容由二进制 ActionScript 指令组成。 ActionScript 是 Flash 使用的基于 ECMA 标准的脚本语言。ActionScript 提供 API，用于创建和操纵客户端用户界面元素和处理数据。

Flex 的安全模型同时保护客户端和服务器。为实现安全性，通常需要考虑以下两个方面：

- 用户访问服务器资源时进行授权和身份验证
- 在客户端安装沙箱来运行 Flash Player

Flex 支持使用任何 J2EE 应用程序服务器的 Web 应用程序安全性。此外， Flex 中预编译的应用程序可以与任何底层服务器技术的身份验证和授权方案相集成，来阻止用户访问您的应用程序。 Flex Framework 还包含若干个内置的安全机制，使您能够控制对 Web 服务、 HTTP 服务和基于服务器的资源 （如 EJB）的访问权限。

Flash Player 在一个安全沙箱内运行，可防止恶意应用程序代码攻击客户端。

注：在 Adobe AIR 中运行的 SWF 内容与在浏览器中运行的内容遵循不同的安全性规则。有关详细信息，请参阅 AIR 文档中的 Air 安全主题。

有关指向各个安全主题的链接，请参阅位于 Adobe Developer Connection 的安全主题中心。

更多帮助主题

Flex 安全性

第 3 章：为以数据为中心的应用程序实现服务

Action Message Format (AMF)

Flex 使用远程对象服务和 AMF 访问在 ColdFusion、PHP、BlazeDS 和 LiveCycle Data Services 中实现的服务。AMF 为在数据库和客户端应用程序之间交换数据提供消息传递功能。

ColdFusion、BlazeDS 和 LiveCycle Data Services 都为远程对象服务提供了一个 AMF 框架。对于使用 PHP 实现的服务，Flash Builder 使用 Zend AMF 框架。

ColdFusion 和 PHP 服务可以提供服务器端类型化。在服务器端类型化中，服务定义返回的数据类型。但是，如果服务实现没有定义返回数据类型，Flash Builder 就会提供客户端类型化。Flash Builder 可对来自该服务的数据进行抽样，以便您可以在客户端应用程序中配置返回类型。

客户端和服务器端类型化

在 Flex 中，客户端应用程序在访问和显示数据的方法中使用从服务调用返回的数据的数据类型。

但是，如下所列的服务返回无类型数据。

- 第 37 页的 “ 实现 ColdFusion 服务 ”
- 第 43 页的 “ 实现 PHP 服务 ”
- 第 54 页的 “ 从多个源实现服务的示例 ”

例如，对于 getAllEmployees() 操作，服务返回无类型对象数组，用于表示数据库中的记录。Flash Builder 提供支持客户端类型化的工具。通过使用 Flash Builder 工具，可对返回的数据进行内部检查，并为该数据定义一种自定义数据类型。

对于员工记录的返回对象，可以定义自定义数据类型 Employee。记录的每一列成为 Employee 数据类型的一个属性。

通过使用 Employee 自定义数据类型，客户端应用程序可以访问返回的数据，并在客户端应用程序中正确显示该数据。

Flash Builder 也可以访问用于实现服务器端类型化的服务。有关服务器端类型化的示例，请参阅 Flash Builder 服务器端类型示例。

实现 ColdFusion 服务

在 ColdFusion 中实现服务时，请将服务作为 ColdFusion 组件 (CFC) 文件实现。每个 CFC 都包含提供服务操作的函数。

可以在任何 IDE（如 Adobe ColdFusion® Builder™）中创建 ColdFusion 服务。Flash Builder 不提供为编辑 ColdFusion 文件而优化的编辑器。但是，如果在 Flash Builder 中打开 ColdFusion 文件，Flash Builder 会启动系统上与 ColdFusion 文件相关联的应用程序。

ColdFusion 服务示例

可以通过创建包含服务操作的函数的 ColdFusion 组件 (CFC) 实现基本 ColdFusion 服务。以下示例 (employeeService.cfc) 说明实现两个函数的 EmployeeService。getAllIEmployees() 函数检索数据库中的所有员工记录。getEmployees() 函数基于函数的 emp_no 参数返回单个员工记录。

此示例说明客户端类型化。服务返回未设置类型的数据。Flash Builder 使用客户端类型化来内部检查返回的数据以及定义数据类型。

后续的示例说明如何实现用于分页和数据管理的服务。

也可以使用 Flash Builder 来访问用于实现服务器端类型化的服务。请参阅第 37 页的 “ 客户端和服务器端类型化 ”。

用于说明服务器端类型化的示例在完成该文档后不可用。有关服务器端类型化示例，请参阅 Flash Builder 服务器端类型示例。

实现基本服务的 ColdFusion 示例

此示例说明如何在 ColdFusion 中实现基本服务。该示例基于访问数据库表时 Flash Builder 生成的代码。请参阅第 8 页的 “ 从数据库表生成示例 ColdFusion 服务 ”。

此示例实现客户端类型化。请参阅第 37 页的 “ 客户端和服务器端类型化 ”。

有关服务器端类型化的示例，请参阅 Flash Builder 服务器端类型示例。

重要说明： 示例服务仅用作原型。仅可在受信任的开发环境中使用示例服务。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全 ColdFusion 服务的信息，请参阅 ColdFusion 文档 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>
```

EmployeeService 的要点：

- 连接员工数据库，并访问数据库中的员工表。
- 返回对象数组。

  使用 Flex 框架编程时，服务仅返回数据。客户端应用程序处理数据的格式设置和表示。此模型与传统的 ColdFusion CFM 应用程序不同，后者返回使用 HTML 模板设置了格式的数据。

  Flex 将返回的记录集作为一个对象数组处理。每行表示从数据库检索到的一个记录。每列数据库记录成为返回对象的一个属性。客户端应用程序现在可以将返回的数据作为具有一组属性的对象进行访问。

  为返回的对象配置数据类型。请参阅第 37 页的 “ 客户端和服务器端类型化 ”。
- ColdFusion 服务器提供错误处理。

  ColdFusion 所提供的错误处理在调试服务时很有用。以 ColdFusion 管理员身份修改调试和日志记录设置，以提供可靠的调试信息。

  Flash Builder“ 测试操作 ” 界面显示 ColdFusion 服务器返回的信息。

  有关测试服务的详细信息，请参阅第 52 页的 “ 调试远程服务 ”。
- 使用 cfqueryparam 构造数据库查询。

  cfqueryparam 有助于防御 SQL injection 语句进入服务器调用。有关更多信息，请参阅 ColdFusion 文档中的 Enhancing security with cfqueryparam。
- 在授权访问此服务中的函数之前，先对用户进行身份验证。

  该示例代码并不说明如何对用户进行身份验证。请参阅 ColdFusion 文档 About User Security。

更多帮助主题

第 22 页的 “ 为数据服务操作配置数据类型 ”

第 7 页的 “ 访问 ColdFusion 服务 ”

第 8 页的 “ 从数据库表生成示例 ColdFusion 服务 ”

实现分页的 ColdFusion 示例

使用 Flash Builder 工具，可以实现对从远程服务检索到的数据的分页。分页是对大型数据集的增量检索。

Flash Builder 需要特定的函数签名才能实现分页。以下代码示例提供了为 ColdFusion 服务实现数据分页的一种方法。

EmployeeServicePaged 示例基于访问数据库表时由 Flash Builder 生成的代码。请参阅第 8 页的 “ 从数据库表生成示例 ColdFusion 服务 ”。

重要说明： 示例服务仅用作原型。仅可在受信任的开发环境中使用示例服务。在该示例中，能够通过网络访问服务器的任何用户都可以对数据库表中的数据执行访问、修改或删除操作。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全 ColdFusion 服务的信息，请参阅 ColdFusion 文档 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>
```

EmployeeServicePaged 服务返回未设置类型的数据。使用 Flash Builder 工具可为 getEmployees_Paged() 配置返回类型。在配置返回类型之后，可对 getEmployees_Paged() 操作启用分页。

更多帮助主题

第 37 页的 “ColdFusion 服务示例 ”

第 22 页的 “ 为数据服务操作配置数据类型 ”

第 26 页的 “ 管理对服务器中的数据的访问 ”

实现数据管理操作的 ColdFusion 示例

使用 Flash Builder 工具，可以实现远程服务的数据管理功能。数据管理是指将更新数据从客户端应用程序同步到服务器。

Flash Builder 需要一个特定的函数签名组合才能实现数据管理。以下代码示例提供了为 ColdFusion 服务实现数据管理的一种方法。

下面的 EmployeeServiceDM 示例基于访问数据库表时由 Flash Builder 生成的代码。请参阅第 8 页的 “ 从数据库表生成示例 ColdFusion 服务 ”。

重要说明：示例服务仅用作原型。仅可在受信任的开发环境中使用示例服务。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全 ColdFusion 服务的信息，请参阅 ColdFusion 文档 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>
```

EmployeeServiceDM 服务返回未设置类型的数据。使用 Flash Builder 工具可为 getAllEmployeess() 和 getEmployees() 配置返回类型。使用 Employee 作为这些操作返回的自定义数据类型。

配置返回类型后，可对 Employee 数据类型启用数据管理。

更多帮助主题

第 37 页的 “ColdFusion 服务示例 ”

第 22 页的 “ 为数据服务操作配置数据类型 ”

第 26 页的 “ 管理对服务器中的数据的访问 ”

使用 Adobe ColdFusion Builder 生成 CFC

Adobe® ColdFusion® Builder™ 提供 Adobe CFC 生成器。使用 CFC 生成器可以从一组数据库表生成 ORM CFC 或传统的 CFC。然后，ColdFusion Builder 生成的 CFC 可以在 Flash Builder 中用作数据服务。Adobe CFC 生成器可以创建实现服务器端类型化的服务。

有关详细信息，请参阅 Using Adobe CFC Generator。

注： ColdFusion 对象关系映射 (ORM) 使用对象模型定义映射策略，以在关系数据库中存储和检索数据。请参阅 ColdFusion ORM。

实现 PHP 服务

使用 PHP 实现服务时，服务通常作为 PHP 类来实现。 PHP 中的类不必是面向对象的类。更确切地讲，每个类都可以是提供服务操作的函数的库。

可以在任何编辑环境（如 DreamWeaver 或 Zend Studio）中创建 PHP 服务。 Flash Builder 不提供为编辑 PHP 文件而优化的编辑器。但是，如果在 Flash Builder 中打开 PHP 文件，Flash Builder 会启动系统上与 PHP 文件相关联的应用程序。为了方便起见， Flash Builder 还提供一个用于编辑 PHP 文件的纯文本编辑器。

使用 AMF 访问使用 PHP 实现的服务

可使用 Action Message Format (AMF) 提供 PHP 数据服务。 AMF 在 Flash 客户端和 Web 服务器之间提供消息传递。Flash Builder 使用 Zend AMF 框架来为 PHP 数据服务实现 AMF 消息传递。

有关 Zend AMF 的详细信息，请参阅 Zend Framework Programmer's Reference。

有关 Zend Framework 的安装信息，请参阅第 17 页的 “ 安装 Zend Framework”。

有关将 Zend 和 PHP Flash Builder 组合使用的信息，请参阅 Zend 网站。

注 尽管 Flash Builder 使用 Zend AMF 框架，但创建 PHP 服务时，无需使用 Zend 组件。尽管 Zend 组件与 Flash Builder 兼容良好，但您也可以使用任何 PHP 开发环境来创建服务。

PHP 服务示例

可以通过创建包含服务操作的函数的 PHP 类文件实现基本 PHP 服务。以下示例说明实现两个函数的 EmployeeService。

- getAllIEmployees()

  检索数据库中的所有员工记录。

- getEmployeeByID($itemID)

  返回单个员工记录。

此示例说明客户端类型化。服务返回未设置类型的数据。Flash Builder 使用客户端类型化来内部检查返回的数据以及定义数据类型。

后续的示例说明如何实现用于分页和数据管理的服务。

也可以使用 Flash Builder 来访问用于实现服务器端类型化的服务。请参阅第 37 页的 “ 客户端和服务器端类型化 ”。

用于说明服务器端类型化的示例在完成该文档后不可用。有关服务器端类型化示例，请参阅 Flash Builder 服务器端类型示例。

PHP 基本服务示例

此示例说明如何在 PHP 中实现基本服务。该示例基于访问数据库表时 Flash Builder 生成的代码。请参阅第 9 页的 “ 从数据库表生成示例 PHP 服务 ”。

此示例说明客户端类型化。请参阅第 37 页的 “ 客户端和服务器端类型化 ”。

重要说明： 示例服务仅用作原型。仅可在受信任的开发环境中使用示例服务。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全服务的信息，请参阅第 34 页的 “ 部署访问数据服务的应用程序 ”。

```
<?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);
        }
    }
}


?>
```

EmployeeService 的要点：

- 连接通过 localhost 的端口 3306 上访问的员工数据库。访问数据库中的员工表。
- 为连接服务和访问数据库中的表提供类变量。

  这些变量可以在类中的函数中使用。

  使用您的系统的值替换这些变量的值。
- 将对象数组返回到客户端应用程序。

使用 Flex 框架编程时，服务仅返回数据。客户端应用程序处理数据的格式设置和表示。

此模型与传统的 PHP 服务不同，后者返回使用 HTML 模板设置了格式的数据。

- getEmployeesByID($itemID) 函数将输入参数绑定到数据类型。

  变量的数目和字符串类型的长度必须与语句中的参数匹配。语句中的 “?” 是参数的占位符。

  mysqli 可识别以下类型：

  - Integer (i)
  - Double (d)
  - String (s)
  - Blob (b)

- 绑定结果，从而创建一个对象数组 ($row[])。

  Flex 将记录集作为一个对象数组处理。每个对象表示从数据库检索到的一个记录。每列数据库记录成为返回对象的一个属性。客户端应用程序现在可以将返回的数据作为具有一组属性的对象进行访问。

  因为该服务器不为返回的数据定义数据类型，所以您需要为返回的对象配置数据类型。请参阅第 37 页的 “ 客户端和服务器端类型化 ”。

- 提供用于连接初始化到数据库的构造函数。
- 使用 mysqli prepare 语句构造数据库查询。

  使用 prepare 语句有助于防御 SQL injection 语句进入服务器调用。此语句仅在准备好之后才会在服务器上执行。

- 在授权访问此服务中的函数之前，先对用户进行身份验证。

  该示例代码并不说明如何对用户进行身份验证。请参阅 ColdFusion 文档 About User Security。此 ColdFusion 文档中用户身份验证和授权的安全原则适用于 PHP 服务。

- 出错时引发异常。

  提供的异常信息在调试服务实现时很有用。 Flash Builder“ 测试操作 ” 界面显示由异常返回的信息。

  有关测试服务的详细信息，请参阅第 52 页的 “ 调试远程服务 ”。

- 文件名 EmployeeService.php 与服务的 PHP 类名匹配。

  如果文件名与类名不匹配，则在访问服务时会发生错误。

更多帮助主题

第 22 页的 “ 为数据服务操作配置数据类型 ”

第 8 页的 “ 访问 PHP 服务 ”

第 9 页的 “ 从数据库表生成示例 PHP 服务 ”

实现分页的 PHP 示例

使用 Flash Builder 工具，可以实现对从远程服务检索到的数据的分页。分页是对大型数据集的增量检索。

Flash Builder 需要特定的函数签名才能实现分页。以下代码示例提供了为 PHP 服务实现数据分页的一种方法。

此示例基于访问数据库表时由 Flash Builder 生成的代码。请参阅第 9 页的 “ 从数据库表生成示例 PHP 服务 ”。

重要说明： 示例服务仅用作原型。仅可在受信任的开发环境中使用示例服务。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全服务的信息，请参阅第 34 页的 “ 部署访问数据服务的应用程序 ”。

```
<?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);
        }
    }
}
?>
```

EmployeeServicePaged 服务返回未设置类型的数据。使用 Flash Builder 工具可为 getEmployees_Paged() 配置返回类型。在配置返回类型之后，可对 getEmployees_Paged() 操作启用分页。

更多帮助主题

第 43 页的 “PHP 服务示例 ”

第 22 页的 “ 为数据服务操作配置数据类型 ”

第 26 页的 “ 管理对服务器中的数据的访问 ”

实现数据管理的 PHP 示例

使用 Flash Builder 工具，可以实现远程服务的数据管理功能。数据管理是指将更新数据从客户端应用程序同步到服务器。

Flash Builder 需要一个特定的函数签名组合才能实现数据管理。以下代码示例提供了为 PHP 服务实现数据管理的一种方法。

此示例基于访问数据库表时由 Flash Builder 生成的代码。请参阅第 9 页的 “ 从数据库表生成示例 PHP 服务 ”。

重要说明： 示例服务仅用作原型。仅可在受信任的开发环境中使用示例服务。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全服务的信息，请参阅第 34 页的 “ 部署访问数据服务的应用程序 ”。

```
<?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, 'issssi', $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);
        }
    }
}
?>
```

EmployeeServiceDM 服务返回未设置类型的数据。使用 Flash Builder 工具可为 getAllEmployeess() 和 getEmployeesByID() 配置返回类型。使用 Employee 作为这些操作返回的自定义数据类型。

配置返回类型后，可对 Employee 数据类型启用数据管理。

更多帮助主题

第 43 页的 “PHP 服务示例 ”

第 22 页的 “ 为数据服务操作配置数据类型 ”

第 26 页的 “ 管理对服务器中的数据的访问 ”

调试远程服务

可以使用以下方法调试访问远程服务的应用程序。

- Flash Builder“ 测试操作 ” 视图

  使用 Flash Builder“ 测试操作 ” 视图可调用服务操作以及查看返回的数据。“ 测试操作 ” 视图显示该服务所显示的所有错误消息。

- 服务器端脚本

  为了对服务进行额外调试，您可以编写脚本，用于测试服务器代码以及将输出流信息写入到日志文件中。

- Flash Builder 网络监视器

  在使用 Flash Builder 构建访问服务的应用程序之后，使用网络监视器。使用网络监视器可查看在服务器和客户端之间发送的数据。

Flash Builder“ 测试操作 ” 视图

使用 Flash Builder“ 测试操作 ” 视图可调用来自某一服务的操作以及查看操作的结果。其结果包括从服务返回的所有错误消息。

使用 “ 测试操作 ” 视图可查看从操作返回的数据，这些操作可以针对您编写的服务执行，也可以针对从 HTTP 或 Web 服务获得的服务执行。

测试服务操作

此过程假定您已编写正在测试的服务，或具有对 HTTP 或 Web 服务的访问权限。

1 在 Flash Builder“ 数据 / 服务 ” 视图中，导航到需要测试的服务操作。

2 在服务操作的上下文菜单中，选择 “ 测试操作 ”。

3 （可选）在 “ 测试操作 ” 视图中，选择 “ 需要身份验证 ” 以向服务提供登录凭据。

4 如果操作采用参数，请单击 “ 输入值 ” 字段为参数提供值。

如果参数需要复杂类型，请单击 “ 输入值 ” 字段中的省略号按钮以打开接受 JSON 表示法的编辑器。使用 JSON 表示法为参数提供值。

5 单击 “ 测试 ” 查看操作的结果。

用于测试服务器代码的脚本

尝试在 Flash Builder 中连接到服务器之前，请使用测试脚本查看并调试服务器代码。测试脚本具有下列优点：

- 您可以从 Web 浏览器查看测试结果。

  修改代码时，只需刷新浏览器页面即可查看结果。

- 您可以回显结果或将结果打印到输出流，这是无法直接通过 AMF 实现的。
- 显示错误时采用的格式很好，并且这样捕获的错误通常比使用 AMF 捕获的错误更完整。

ColdFusion 脚本

使用下面的脚本 tester.cfm 将调用转储为函数。

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

在 tester2.cfm 中，指定要在 URL 中调用的方法和参数。

```
<!--- 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>
```

例如，使用以下 URL 调用 EmployeeService 中的 getItemID() 方法：

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

tester3.cfm 编写用于记录对操作的调用的日志，并使用 cfdump 转储输入参数。

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

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

PHP 脚本

使用下面的脚本 tester.php 将调用转储为函数。

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

将下列代码添加到 PHP 服务以在代码执行期间记录消息。

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

将以下代码添加到 PHP 服务以启用对日志文件的转储：

```
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);
```

网络监视器

在 Flash Builder 中可通过 Flex 调试透视图访问网络监视器。启用监视器后即可监视数据。有关启用和使用网络监视器的详细信息，请参阅监视访问数据服务的应用程序。

从多个源实现服务的示例

应用程序通常会访问来自不同源的数据，并在应用程序中显示数据关联结果。此示例说明如何关联来自员工数据库中以下三个表的数据：

- Departments

  每个记录都包含以下字段：department number 和 department name。

- Dept_emp

  每个记录都包含以下字段：emp_no、 dept_no、 from_date、 to_date

- Employees

  每个记录都包含以下字段：emp_no、 birth_date、 first_name、 last_name、 gender、 hire_date

该示例应用程序有两个 DataGrid，一个用于 Departments，另一个用于 Employees。

Departments 列出所有部门。选中某个部门时， Employees DataGrid 会列出该部门中的所有员工。

在 Employees DataGrid 中选中某个员工时，会填充一个表单，供您更新员工记录。

创建服务

对于此示例，请创建单个服务。该服务包含以下操作：

- getAllDepartments()
- getEmployeesByDept()
- getEmployeeByID()
- updateEmployee()

EmployeeService (PHP)

EmployeeService.php 实现包含单个函数的服务。GetEmployeesByID() 接受部门 ID 作为一个参数，从而返回给定部门中的所有员工。此函数还返回员工加入和离开该部门的日期。 GetEmployeesByDept() 执行以下 SQL 查询：

```
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
```

重要说明： 示例服务仅用作原型。仅可在受信任的开发环境中使用示例服务。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全服务的信息，请参阅第 34 页的 “ 部署访问数据服务的应用程序 ”。

```
<?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 实现包含单个函数的服务。 GetEmployeesByID() 接受部门 ID 作为一个参数，从而返回给定部门中的所有员工。此函数还返回员工加入和离开该部门的日期。 GetEmployeesByDept() 执行以下 SQL 查询：

```
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
```

重要说明： 示例服务仅用作原型。仅可在受信任的开发环境中使用示例服务。因此，在部署此服务之前，请务必提高其安全性，并适当地限制访问权限。有关编写安全 ColdFusion 服务的信息，请参阅 ColdFusion 文档 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>?>
```

将服务导入到服务器项目中

1 在 Flash Builder 中，创建名为“Associations”的 Flex 项目：

(PHP) 创建项目时，指定 PHP 作为“应用程序服务器类型”。

(PHP) 创建项目后，Flash Builder 会在 PHP 配置的 Web 根文件夹中创建一个输出文件夹。PHP_Associations 项目的默认名称是 PHP_Associations-debug。

(ColdFusion) 创建项目时，指定 ColdFusion 作为“应用程序服务器类型”。然后指定 ColdFusion Flash Remoting。

2 (PHP) 在 PHP_Associations-debug 内，创建一个名为 services 的文件夹。将 EmployeeService.php 复制到 services 文件夹中。

3 (ColdFusion) 在 ColdFusion 配置的 Web 根文件夹中创建一个名为 Associations 的文件夹。将 EmployeeService.chc 复制到 Associations 文件夹中。

4 将 EmployeeService 导入到项目中：

确保在 Flash Builder 中 PHP_Associations 为活动项目。

选择“数据”>“连接 PHP”。要指定 PHP 类，请浏览到 services 文件夹，并选中 EmployeeService.php。单击“完成”。

有关更多信息，请参阅第 9 页的“连接 PHP 数据服务”。

5 配置 EmployeeService 中的操作的返回类型。

- DepartmentService

  在 getAllDepartments() 的上下文菜单中，选择 “ 配置返回类型 ”。

  单击 “ 下一步 ” 自动检测返回类型。

  指定 Department 作为自定义返回类型。单击 “ 完成 ”。

- EmployeeService

  对于 getEmployeesByDept()，选择 “ 配置返回类型 ”。

  单击 “ 下一步 ” 自动检测返回类型。

  指定 d007 作为参数的值。单击 “ 下一步 ”。

  指定 Employee 作为自定义返回类型。单击 “ 完成 ”。

有关更多信息，请参阅第 22 页的 “ 为数据服务操作配置数据类型 ”。

第 4 章：访问服务器端数据

Adobe® Flex® 数据访问组件使用远程过程调用与服务器环境（如 PHP、Adobe ColdFusion 和 Microsoft ASP.NET）进行交互。这些组件给使用 Adobe Flex 框架构建的客户端应用程序提供数据，以及将数据发送到后端数据源。有关数据访问组件的简介，请参阅第 4 页的“ 数据访问组件 ”。

使用 HTTPService 组件

可以将 HTTPService 组件与以下任何种类的服务器端技术一起使用：PHP 页、 ColdFusion 页、 JavaServer Pages (JSP)、Java Servlet、 Ruby on Rails 和 Microsoft ASP 页。另外，还可以使用 HTTPService 来访问基于 REST 的 Web 服务。

有关 HTTPService 组件的 API 参考信息，请参阅 mx.rpc.http.mxml.HTTPService。

使用 PHP 和 SQL 数据

可以将 HTTPService 组件与 PHP 和 SQL 数据库管理系统一起使用，以便在应用程序中显示数据库查询结果。也可以使用该组件在数据库中插入、更新和删除数据。可以使用 GET 或 POST 调用 PHP 页来执行数据库查询。随后，将查询结果数据的格式设置为 XML 结构，并将该 XML 结构以 HTTP 响应形式返回到应用程序。在将结果返回到应用程序之后，可以将其显示在一个或多个用户界面控件中。

MXML 代码

以下示例中的应用程序用 POST 方法调用一个 PHP 页。该 PHP 页查询一个名为 users 的 MySQL 数据库表。它将查询结果设置为 XML 格式，并将该 XML 返回到应用程序，在该应用程序中，查询结果将绑定到 DataGrid 控件的 dataProvider 属性并显示在 DataGrid 控件中。应用程序还将新用户的用户名和电子邮件地址发送到 PHP 页，后者将其插入到用户数据库表中。

```
<?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>
```

HTTPService 的 send() 方法对 PHP 页进行调用。此调用是在 MXML 文件的 Script 块中使用 send_data() 方法进行的。

HTTPService 组件的 resultFormat 属性设置为 object，因此数据作为一个 ActionScript 对象图发回到应用程序。这是 resultFormat 属性的默认值。或者，也可以将 resultFormat 设置为 e4x，这会将数据返回为一个 XMLList 对象，在该对象上可以执行 ECMAScript for XML (E4X) 操作。将 resultFormat 属性切换为 e4x 需要对 MXML 代码进行少量更改，如下所示。

注：如果结果采用 e4x 格式，则在绑定到 DataGrid 时，不需要在点标记中包括 XML 结构的根节点。

本示例中返回的 XML 不包含命名空间信息。有关处理包含命名空间的 XML 的信息，请参阅第 105 页的 “ 使用 e4x 结果格式将结果作为 XML 处理 ”。

```
...
<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}">
...
```

在使用 e4x 结果格式时，可以选择将 lastResult 属性绑定到 XMLListCollection 对象，随后将该对象绑定到 DataGrid.dataProvider 属性，如下面的代码片断中所示：

```
<fx:Declarations>
...
    <mx:XMLListCollection id="xc"
         source="{userRequest.lastResult.user}"/>
...
</fx:Declarations>
...
    <mx:DataGrid id="dgUserRequest" x="22" y="128" dataProvider="{xc}">
...
```

MySQL 数据库脚本

此应用程序的 PHP 代码使用名为 sample 的 MySQL 数据库中的 users 数据库表。可以使用下面的 MySQL 脚本来创建此表

```
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 ;
```

PHP 代码

此应用程序调用下面的 PHP 页。此 PHP 代码执行 SQL 数据库插入和查询操作，并将查询结果以 XML 结构返回到应用程序。

```
<?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)
?>
```

使用 ColdFusion 和 SQL 数据

可以将 HTTPService 组件与 ColdFusion 页和 SQL 数据库管理系统一起使用，以便在应用程序中显示数据库查询结果。也可以使用该组件在数据库中插入、更新和删除数据。通过用 GET 或 POST 调用 ColdFusion 页来执行数据库查询。随后，将查询结果数据的格式设置为 XML 结构，并将该 XML 结构以 HTTP 响应形式返回到应用程序。在将结果返回到应用程序之后，可以将其显示在一个或多个用户界面控件中。

MXML 代码

以下示例中的应用程序用 POST 方法调用一个 ColdFusion 页。ColdFusion 页查询一个名为 users 的 MySQL 数据库表。它将查询结果设置为 XML 格式，并将该 XML 返回到应用程序，在该应用程序中，查询结果将绑定到 DataGrid 控件的 dataProvider 属性并显示在 DataGrid 控件中。应用程序还将新用户的用户名和电子邮件地址发送到 ColdFusion 页，后者将其插入到用户数据库表中。

```
<?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>
```

HTTPService 的 send() 方法对 ColdFusion 页进行调用。此调用是在 MXML 文件的 Script 块中使用 send_data() 方法进行的。

HTTPService 组件的 resultFormat 属性设置为 object，因此数据作为一个 ActionScript 对象图发回到应用程序。这是 resultFormat 属性的默认值。或者，也可以使用 e4x 结果格式，这会将数据返回为一个 XMLList 对象，在该对象上可以执行 ECMAScript for XML (E4X) 操作。将 resultFormat 属性切换为 e4x 需要对 MXML 代码进行少量更改，如下所示。

注：如果结果采用 e4x 格式，则在绑定到 DataGrid 时，不需要在点标记中包括 XML 结构的根节点。

本示例中返回的 XML 不包含命名空间信息。有关处理包含命名空间的 XML 的信息，请参阅第 105 页的 “ 使用 e4x 结果格式将结果作为 XML 处理 ”。

```
...
<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}">
...
```

在使用 e4x 结果格式时，可以选择将 lastResult 属性绑定到 XMLListCollection 对象，随后将该对象绑定到 DataGrid 的 dataProvider 属性，如下面的代码片断中所示：

```
<fx:Declarations>
...
    <mx:XMLListCollection id="xc"
         source="{userRequest.lastResult.user}"/>
...
</fx:Declarations>
...
    <mx:DataGrid id="dgUserRequest" x="22" y="128" dataProvider="{xc}">
...
```

SQL 脚本

此应用程序的 ColdFusion 代码使用名为 sample 的 MySQL 数据库中名为 users 的数据库表。可以使用下面的 MySQL 脚本来创建此表：

```
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 ;
```

ColdFusion 代码

在 “ 使用 ColdFusion 和 SQL 数据 ” 部分中列出的应用程序可调用以下 ColdFusion 应用程序：returncfxml.cfm。此 ColdFusion 代码执行 SQL 数据库插入和查询操作，并将查询结果返回到应用程序。 ColdFusion 页使用 cfquery 标签在数据库中插入数据并查询数据库，使用 cfxml 标签将查询结果的格式设置为 XML 结构。

```
<!--- 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>
```

使用 Javaserver Pages

可以将 Flex HTTPService 组件与 JSP 页和 SQL 数据库管理系统一起使用，以便在应用程序中显示数据库查询结果。也可以使用该组件在数据库中插入、更新和删除数据。使用 GET 或 POST 调用 JSP 页来执行数据库查询。随后，将查询结果数据的格式设置为 XML 结构，并将该 XML 结构以 HTTP 响应形式返回到应用程序。在将结果返回到应用程序之后，可以将其显示在一个或多个用户界面控件中。

MXML 代码

以下示例中的应用程序调用一个从 SQL 数据库检索数据的 JSP 页。它将数据库查询结果的格式设置为 XML，并将该 XML 返回到应用程序，在该应用程序中，查询结果将绑定到 DataGrid 控件的 dataProvider 属性并显示在 DataGrid 控件中。

```
<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>
```

HTTPService 的 send() 方法对 JSP 页进行调用。此调用是在 MXML 文件中 Button 的 click 事件中调用的。

HTTPService 组件的 resultFormat 属性设置为 object，因此数据作为一个 ActionScript 对象图发回到应用程序。这是 resultFormat 属性的默认值。或者，也可以使用 e4x 结果格式，这会将数据返回为一个 XMLList 对象，在该对象上可以执行 ECMAScript for XML (E4X) 操作。将 resultFormat 属性切换为 e4x 需要对 MXML 代码进行少量更改，如下所示。

注：如果结果采用 e4x 格式，则在绑定到 DataGrid 时，不需要在点标记中包括 XML 结构的根节点。

本示例中返回的 XML 不包含命名空间信息。有关处理包含命名空间的 XML 的信息，请参阅第 105 页的 “ 使用 e4x 结果格式将结果作为 XML 处理 ”。

```
...
    <s:HTTPService id="srv" url="catalog.jsp" resultFormat="e4x"/>
...
    <mx:DataGrid dataProvider="{srv.lastResult.product}" width="100%" height="100%"/>
```

在使用 e4x 结果格式时，可以选择将 lastResult 属性绑定到 XMLListCollection 对象，随后将该对象绑定到 DataGrid.dataProvider 属性：

```
<fx:Declarations>
...
    <mx:XMLListCollection id="xc"
         source="{userRequest.lastResult.user}"/>
...
</fx:Declarations>
...
    <mx:DataGrid id="dgUserRequest" x="22" y="128" dataProvider="{xc}">
...
```

JSP 代码

下面的示例显示此应用程序中使用的 JSP 页。此 JSP 页不直接调用数据库。它从名为 ProductService 的 Java 类获取数据，该类又使用名为 Product 的 Java 类表示各个产品。

```
<%@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>
```

在 ActionScript 中调用 HTTP 服务

下面的示例显示了 ActionScript 脚本块中的 HTTP 服务调用。调用 useHTTPService() 方法即会声明服务、设置目标、设置 result 和 fault 事件侦听器并调用服务的 send() 方法。

```
<?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>
```

使用 WebService 组件

使用 Flex 框架创建的应用程序可与在 Web 服务描述语言 1.1 (WSDL 1.1) 文档 （以 URL 形式提供）中定义其接口的基于 SOAP 的 Web 服务进行交互。WSDL 是一种标准格式，用于描述能够由 Web 服务理解的消息、Web 服务对于这些消息的响应格式、Web 服务支持的协议以及将这些消息发送到何处。Flex 的 Web 服务 API 通常支持简单对象访问协议 (SOAP) 1.1、XML Schema 1.0（版本 1999、2000 和 2001）以及 WSDL 1.1 RPC-encoded、RPC-literal 和 document-literal（bare 和 wrapped 参数）。最常见的两种 Web 服务类型使用 RPC-encoded 或 document-literal SOAP 绑定；术语 encoded 和 literal 表示服务所使用的 WSDL 到 SOAP 映射的类型。

Flex 支持 SOAP 消息格式的 Web 服务请求和结果。 SOAP 提供基于 XML 格式的定义，用于在 Web 服务客户端 （如使用 Flex 构建的应用程序）和 Web 服务之间交换结构化和类型化信息。

Adobe® Flash® Player 在安全沙箱内运行，此沙箱限制使用 Flex 构建的应用程序和其它使用 Flash 构建的应用程序通过 HTTP 可以访问的内容。仅允许使用 Flash 构建的应用程序通过提供给它们的协议对相同域上的资源进行 HTTP 访问。因为 Web 服务通常是从远程位置访问的，所以这会引发一个 Web 服务问题。LiveCycle Data Services 和 BlazeDS 中可用的代理服务截取对远程 Web 服务的请求，并重定向请求，然后将响应返回到客户端。

如果未使用 LiveCycle Data Services 或 BlazeDS，则您可以访问与您的应用程序位于同一个域中的 Web 服务；否则必须在承载着 RPC 服务的 Web 服务器上安装 crossdomain.xml （跨域策略）文件 （该文件允许从您的应用程序的域进行访问）。

有关 WebService 组件的 API 参考信息，请参阅 mx.rpc.soap.mxml.WebService。

示例 **WebService** 应用程序

下面的示例代码来自一个使用 WebService 组件调用 Web 服务操作的应用程序。

MXML 代码

以下示例中的 应用程序调用一个 Web 服务，该 Web 服务查询名为 users 的 SQL 数据库表并将数据返回到应用程序。在这个 Flex 应用程序中，返回的数据绑定到 DataGrid 控件的 dataProvider 属性并显示在 DataGrid 控件中。应用程序还将新用户的用户名和电子邮件地址发送到 Web 服务，该 Web 服务会在用户数据库表中执行插入操作。 Web 服务的后端实现是一个 ColdFusion 组件；同一个 ColdFusion 组件在第 84 页的 “ 使用 RemoteObject 组件 ” 中作为远程对象来访问。

```
<?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>
```

WSDL 文档

下面的示例显示了一个用来定义 Web 服务 API 的 WSDL 文档:

```
<?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>
```

在 ActionScript 中调用 Web 服务

下面的示例显示了一个 ActionScript 脚本块中的 Web 服务调用。调用 useWebService() 方法即会声明该服务、设置目标、提取 WSDL 文档并调用该服务的 echoArgs() 方法。

注：在 ActionScript 中声明 WebService 组件时，请调用 WebService.loadWSDL() 方法。

```
<?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>
```

保留的操作名称

只需按服务变量来命名 WebService 操作，即可使它们可访问。但是，如果操作名称恰巧与针对服务定义的方法一样，就会发生命名冲突。可以针对 WebService 组件使用以下 ActionScript 方法来返回给定名称的操作：

```
public function getOperation(name:String):Operation
```

读取 WSDL 文档

可以在 Web 浏览器、简单文本编辑器、 XML 编辑器或者开发环境 （如 Adobe Dreamweaver，其中包含了一个内置实用程序，可以用一种易读格式显示 WSDL 文档的）中查看 WSDL 文档。

WSDL 文档中包含下表中描述的标签。

标签	描述
<binding>	指定客户端 （例如使用 Flex 构建的应用程序）用来与 Web 服务进行通信的协议。绑定包括 SOAP、 HTTP GET、 HTTP POST 和 MIME 绑定。 Flex 仅支持 SOAP 绑定。
<fault>	指定因在处理消息时出现问题而返回的错误值。
<input>	指定客户端 （例如使用 Flex 构建的应用程序）发送给 Web 服务的消息。
<message>	定义 WebService 操作传输的数据。
<operation>	定义 <input>、 <output> 和 <fault> 标签的组合。
<output>	指定 Web 服务发送给 Web 服务客户端 （例如使用 Flex 构建的应用程序）的消息。
<port>	指定 Web 服务端点，以指定绑定和网络地址之间的关联。
<portType>	定义由 Web 服务提供的一个或多个操作。
<service>	定义 <port> 标签集合。每个服务都映射到一个 <portType> 标签并指定用来访问该 <portType> 标签中各种操作的不同方法。
<types>	定义 Web 服务的消息所使用的数据类型。

面向 RPC 的操作和面向文档的操作

WSDL 文件可以指定面向 RPC 的操作，也可以指定面向文档的 (document-literal) 操作。 Flex 支持这两种操作样式。

在调用面向 RPC 的操作时，应用程序会发送一条指定操作及其参数的 SOAP 消息。在调用面向文档的操作时，应用程序会发送一条包含 XML 文档的 SOAP 消息。

在 WSDL 文档中，每个 <port> 标签都有一个 binding 属性来指定特定 <soap:binding> 标签的名称，如以下示例所示：

```
<binding name="InstantMessageAlertSoap" type="s0:InstantMessageAlertSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document"/>
```

相关 <soap:binding> 标签的 style 属性用来确定操作样式。在本例中，采用 document 样式。

服务中的任何操作都可以指定相同的样式，也可以覆盖为服务相关端口指定的样式，如以下示例所示：

```
<operation name="SendMSN">
    <soap:operation soapAction="http://www.bindingpoint.com/ws/imalert/
        SendMSN" style="document"/>
```

有状态的 Web 服务

Flex 使用 Java 服务器会话来保持使用 Cookie 存储会话信息的 Web 服务端点的状态。此功能充当应用程序和 Web 服务之间的媒介。它将端点的标识添加到端点要传递到应用程序的任何内容中。如果端点发送会话信息，则应用程序会接收该信息。此功能不需要任何配置，但是对于使用代理服务时使用 RTMP 通道的目标，不支持此功能。

使用 SOAP 头

SOAP 头是 SOAP 封套中的可选标签，通常包含特定于应用程序的信息 （如身份验证信息）。

向 **Web** 服务请求中添加 **SOAP** 头

某些 Web 服务要求您在调用操作时传递 SOAP 头。

可以通过在事件侦听器函数中调用 WebService 或操作对象的 addHeader() 或 addSimpleHeader() 方法来向所有的 Web 服务操作或单个操作添加 SOAP 头。

在使用 addHeader() 方法时，必须首先分别创建 SOAPHeader 和 QName 对象。 addHeader() 方法具有如下签名：

```
addHeader(header:mx.rpc.soap.SOAPHeader):void
```

要创建 SOAPHeader 对象，请使用以下构造函数：

```
SOAPHeader(qname:QName, content:Object)
```

要在 SOAPHeader() 方法的第一个参数中创建 QName 对象，请使用下面的构造函数：

```
QName(uri:String, localName:String)
```

SOAPHeader() 构造函数的 content 参数是一组基于以下格式的名称 / 值对：

```
{name1:value1, name2:value2}
```

addSimpleHeader() 方法是单个名称 / 值 SOAP 头的快捷方式。在使用 addSimpleHeader() 方法时，可以在该方法的参数中创建 SOAPHeader 和 QName 对象。 addSimpleHeader() 方法具有如下签名：

```
addSimpleHeader(qnameLocal:String, qnameNamespace:String, headerName:String,
    headerValue:Object):void
```

addSimpleHeader() 方法采用如下参数：

- ***qnameLocal*** 是头的 QName 的本地名称。
- ***qnameNamespace*** 是头的 QName 的命名空间。
- ***headerName*** 是头的名称。
- ***headerValue*** 是头的值。这个值可以是字符串 （如果它是一个简单值）、将进行基本 XML 编码的对象或者 XML （如果您要亲自指定头的 XML）。

  以下示例中的代码显示如何使用 addHeader() 和 addSimpleHeader() 方法来添加 SOAP 头。这些方法是在一个名为 headers 的事件侦听器函数中调用的，该事件侦听器是在 <mx:WebService> 标签的 load 属性中指定的。

```
<?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>
```

清除 SOAP 头

使用 WebService 或操作对象的 clearHeaders() 方法可以删除已添加到对象中的 SOAP 头，如下面针对一个 WebService 对象的示例所示。必须在添加头时所在的级别 （WebService 或操作级别）调用 clearHeaders()。

```
<?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>
```

将 Web 服务重定向到其它 URL

某些 Web 服务要求在处理 WSDL 并对 Web 服务发出初始调用之后更改到其它端点 URL。例如，假设您希望使用一个要求您传递安全凭据的 Web 服务。当您调用该 Web 服务以发送登录凭据时，该服务会接受这些凭据，并返回使用该服务的业务操作所需的实际端点 URL。在调用业务操作之前，您必须更改 WebService 组件的 endpointURI 属性。

下面的示例显示一个结果事件侦听器，该侦听器将 Web 服务返回的端点 URL 存储在变量中，然后将该变量传递到一个函数以更改后续请求的端点 URL:

```
...
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;

}
...
```

要求您传递安全凭据的 Web 服务还可能会返回一个标识符，您必须在后续请求的 SOAP 头中附加该标识符。有关更多信息，请参阅第 74 页的 “使用 SOAP 头”。

序列化 Web 服务数据

对 ActionScript 数据进行编码

下表显示了 ActionScript 3.0 类型与 XML 架构复杂类型的编码映射。

XML 架构定义	支持的 ActionScript 3.0 类型	注释
顶级元素		
xsd:element nillable == true	Object	如果输入值为 null，则编码输出用 xsi:nil 属性设置。
xsd:element fixed != null	Object	将忽略输入值而使用 fixed 值。
xsd:element default != null	Object	如果输入值为 null，则使用这个默认值。
本地元素		
xsd:element maxOccurs == 0	Object	将在编码输出中省略输入值。
xsd:element maxOccurs == 1	Object	输入值作为单个实体来处理。如果关联类型是 SOAP 编码数组，则数组和 mx.collection.IList 实现在传递过程中将保持不变，从而由该类型的 SOAP 编码器作为一个特例来处理。
xsd:element maxOccurs > 1	Object	输入值应该可迭代 （例如数组或 mx.collections.IList 实现），尽管不可迭代的值会在处理之前进行包装。各个项目将按照定义编码为不同的实体。
xsd:element minOccurs == 0	Object	如果未定义输入值或者输入值为 null，则将省略编码输出。

下表显示了 ActionScript 3.0 类型与 XML 架构内置类型的编码映射。

XML 架构类型	支持的 ActionScript 3.0 类型	注释
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	使用 mx.utils.Base64Encoder （不换行）。
xsd:boolean	Boolean Number Object	总是编码为 true 或 false。 Number == 1 则为 true，否则为 false。 Object.toString() ==“true” 或 “1”，则为 true，否则为 false。
xsd:byte xsd:unsignedByte	Number String	String 首先转换为 Number。

XML 架构类型	支持的 ActionScript 3.0 类型	注释
xsd:date	Date Number String	使用 Date UTC 访问器方法。 使用 Number 设置 Date.time。 假设按原样对 String 预先设置格式和编码。
xsd:dateTime	Date Number String	使用 Date UTC 访问器方法。 使用 Number 设置 Date.time。 假设按原样对 String 预先设置格式和编码。
xsd:decimal	Number String	使用 Number.toString()。 Infinity、 -Infinity 和 NaN 对于此类型无效。 String 首先转换为 Number。
xsd:double	Number String	限制在 Number 的范围。 String 首先转换为 Number。
xsd:duration	Object	调用 Object.toString()。
xsd:float	Number String	限制在 Number 的范围。 String 首先转换为 Number。
xsd:gDay	Date Number String	使用 Date.getUTCDate()。 Number 直接用于日期。 String 分析为 Number，再用于日期。
xsd:gMonth	Date Number String	使用 Date.getUTCMonth()。 Number 直接用于月份。 String 分析为 Number，再用于月份。
xsd:gMonthDay	Date String	使用 Date.getUTCMonth() 和 Date.getUTCDate()。 String 分析为月份和日期部分。
xsd:gYear	Date Number String	使用 Date.getUTCFullYear()。 Number 直接用于年份。 String 分析为 Number，再用于年份。
xsd:gYearMonth	Date String	使用 Date.getUTCFullYear() 和 Date.getUTCMonth()。 String 分析为年份和月份部分。
xsd:hexBinary	flash.utils.ByteArray	使用 mx.utils.HexEncoder。
xsd:integer 及其派生类型： xsd:negativeInteger xsd:nonNegativeInteger xsd:positiveInteger xsd:nonPositiveInteger	Number String	限制在 Number 的范围。 String 首先转换为 Number。

XML 架构类型	支持的 ActionScript 3.0 类型	注释
xsd:int xsd:unsignedInt	Number String	String 首先转换为 Number。
xsd:long xsd:unsignedLong	Number String	String 首先转换为 Number。
xsd:short xsd:unsignedShort	Number String	String 首先转换为 Number。
xsd:string 及其派生类型： 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	调用 Object.toString()。
xsd:time	Date Number String	使用 Date UTC 访问器方法。 使用 Number 设置 Date.time。 假设按原样对 String 预先设置格式和编码。
xsi:nil	null	如果相应的 XML 架构元素定义中包含 minOccurs > 0，则使用 xsi:nil 编码 null 值，否则将省略整个元素。

下表显示了 ActionScript 3.0 类型与 SOAP 编码类型的映射。

SOAPENC 类型	支持的 ActionScript 3.0 类型	注释
soapenc:Array	Array mx.collections.IList	RPC-encoded 的数组是特例，只有 RPC-encoded 样式 Web 服务支持。
soapenc:base64	flash.utils.ByteArray	按照与 xsd:base64Binary 相同的方式进行编码。
soapenc:*	Object	任何其它 SOAP 编码类型都将基于其 QName 的 localName 进行处理，就好像该类型位于 XSD 命名空间中一样。

将 XML 架构和 SOAP 解码为 ActionScript 3.0

下表显示了 XML 架构内置类型与 ActionScript 3.0 类型的解码映射。

XML 架构类型	解码的 ActionScript 3.0 类型	注释
xsd:anyType xsd:anySimpleType	String Boolean Number	如果内容为空，则使用 xsd:string。 如果内容强制转换为 Number，而且值为 NaN；或者 如果内容以 “0” 或 “-0” 开头，或者 如果内容以 “E” 结尾： 则在内容为 “true” 或 “false” 时使用 xsd:boolean 否则为 xsd:string。 其它情况下内容是有效的 Number，则使用 xsd:double。
xsd:base64Binary	flash.utils.ByteArray	使用 mx.utils.Base64Decoder。
xsd:boolean	Boolean	如果内容为 “true” 或 “1”，则为 true，否则为 false。
xsd:date	Date	如果未提供时区信息，则假设使用本地时间。
xsd:dateTime	Date	如果未提供时区信息，则假设使用本地时间。
xsd:decimal	Number	内容通过 Number(content) 来创建，因此将限制在 Number 的范围内。
xsd:double	Number	内容通过 Number(content) 来创建，因此将限制在 Number 的范围内。
xsd:duration	String	在返回内容时折叠空格。
xsd:float	Number	内容通过 Number(content) 进行转换，因此将限制在 Number 的范围内。
xsd:gDay	uint	内容通过 uint(content) 进行转换。
xsd:gMonth	uint	内容通过 uint(content) 进行转换。
xsd:gMonthDay	String	在返回内容时折叠空格。
xsd:gYear	uint	内容通过 uint(content) 进行转换。
xsd:gYearMonth	String	在返回内容时折叠空格。
xsd:hexBinary	flash.utils.ByteArray	使用 mx.utils.HexDecoder。

XML 架构类型	解码的 ActionScript 3.0 类型	注释
xsd:integer 及其派生类型： 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	内容通过 parseInt() 进行解码。
xsd:string 及其派生类型： 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	原始内容只是以字符串形式返回。
xsd:time	Date	如果未提供时区信息，则假设使用本地时间。
xsi:nil	null	

下表显示了 SOAP 编码类型与 ActionScript 3.0 类型的解码映射。

SOAPENC 类型	解码的 ActionScript 类型	注释
soapenc:Array	Array mx.collections.ArrayCollection	SOAP 编码数组是特例。如果 makeObjectsBindable 为 true，则结果将包装在 ArrayCollection 中，否则将返回一个简单的数组。
soapenc:base64	flash.utils.ByteArray	按照与 xsd:base64Binary 相同的方式进行解码。
soapenc:*	Object	任何其它 SOAP 编码类型都将基于其 QName 的 localName 进行处理，就好像该类型位于 XSD 命名空间中一样。

下表显示了自定义数据类型与 ActionScript 3.0 数据类型的解码映射。

自定义类型	解码的 ActionScript 3.0 类型	注释
Apache Map http://xml.apache.org/xml-soap:Map	Object	java.util.Map 的 SOAP 表示形式。Keys 必须能够以字符串表示。
Apache Rowset http://xml.apache.org/xml-soap:Rowset	对象数组	
ColdFusion QueryBean http://rpc.xml.coldfusion:QueryBean	对象数组 对象的 mx.collections.ArrayCollection	如果 makeObjectsBindable 为 true，则得到的数组将包装在 ArrayCollection 中。

XML 架构元素支持

XML 架构的以下结构或结构属性在 Flex 4 中仅实现了一部分：

```
<choice>
<all>
<union
```

XML 架构的以下结构或结构属性在 Flex 4 中不受支持而且将被忽略：

```
<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>
```

自定义 Web 服务类型映射

在使用来自 Web 服务调用的数据时，Flex 通常会创建无类型匿名 ActionScript 对象，在 SOAP 消息的主体内模拟 XML 结构。如果您希望 Flex 创建特定类的实例，可以使用 mx.rpc.xml.SchemaTypeRegistry 对象并使用相应的 ActionScript 类注册 QName 对象。

例如，假设您在名为 User.as 的文件中包含如下定义：

```
package
{
     public class User
     {
       public function User() {}

       public var firstName:String;
       public var lastName:String;
     }
}
```

接下来，您希望针对 Web 服务调用可返回下列 XML 的 getUser 操作：

```
<tns:getUserResponse xmlns:tns="http://example.uri">
     <tns:firstName>Ivan</tns:firstName>
     <tns:lastName>Petrov</tns:lastName>
</tns:getUserResponse>
```

请确保在调用 getUser 操作时会获得 User 类的实例 （而不是泛型 Object），此时需要在应用程序中的方法内使用下面的 ActionScript 代码：

```
SchemaTypeRegistry.getInstance().registerClass(new QName("http://example.uri", "getUserResponse"), User);
```

SchemaTypeRegistry.getInstance() 是一种静态方法，它返回类型注册表的默认实例。在多数情况下，您只需要这些。但是，这会向应用程序内所有 Web 服务操作中的同一个 ActionScript 类注册给定的 QName。如果您希望为不同的操作注册不同的类，则需要在应用程序的方法中使用下面的代码：

```
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;
```

使用自定义的 Web 服务序列化

可通过两种方法来完全控制如何将 ActionScript 对象序列化为 XML 以及如何反序列化 XML 响应消息。建议的方法是直接使用 E4X。

如果将 XML 的实例作为唯一参数传递给 Web 服务操作，则它将在经过序列化的请求中作为 <SOAP:Body> 节点的子级传递，并保持不变。在需要完全控制 SOAP 消息时，可以使用此策略。同样，在反序列化 Web 服务响应时，可以将操作的 resultFormat 属性设置为 e4x。这会在响应消息中返回一个作为 <SOAP:Body> 节点子级的 XMLList 对象。自此以后，您就可以实现必要的自定义逻辑来创建相应的 ActionScript 对象。

第二种方法是提供自己的 mx.rpc.soap.ISOAPDecoder 和 mx.rpc.soap.ISOAPEncoder 实现，此方法较为繁琐。例如，如果您编写了一个名为 MyDecoder 的类来实现 ISOAPDecoder，就可以在应用程序的方法中使用以下代码：

```
myWS.someOperation.decoder = new MyDecoder();
```

在调用 someOperation 时， Flex 会调用 MyDecoder 类的 decodeResponse() 方法。自此以后，将由该自定义实现来处理整个 SOAP 消息并生成预期的 ActionScript 对象。

使用 RemoteObject 组件

可以使用 Flex RemoteObject 组件来针对 ColdFusion 组件或 Java 类调用方法。

还可以将包含 PHP 和 .NET 对象的 RemoteObject 组件与第三方软件（例如开放源代码项目 AMFPHP 和 SabreAMF 以及 Midnight Coders WebORB）结合使用。有关详细信息，请访问以下网站：

- Zend Framework http://framework.zend.com/
- AMFPHP http://amfphp.sourceforge.net/
- SabreAMF http://www.osflash.org/sabreamf
- Midnight Coders WebORB http://www.themidnightcoders.com/

RemoteObject 组件使用 AMF 协议发送和接收数据，而 WebService 和 HTTPService 组件使用 HTTP 协议。AMF 显著快于 HTTP，但服务器端编码和配置通常更复杂。

Flash Builder for PHP 是一种与 Zend Technologies 合作创建的开发工具，其中包含了集成的 Zend Studio 副本。有关更多信息，请参阅 Adobe 网站。

与 HTTPService 和 WebService 组件一样，您可以使用 RemoteObject 组件在应用程序中显示数据库查询结果。也可以使用该组件在数据库中插入、更新和删除数据。在将查询结果返回到应用程序之后，可以将其显示在一个或多个用户界面控件中。

有关 RemoteObject 组件的 API 参考信息，请参阅 mx.rpc.remoting.mxml.RemoteObject。

示例 RemoteObject 应用程序

MXML 代码

以下示例中的应用程序使用 RemoteObject 组件调用 ColdFusion 组件。ColdFusion 组件查询一个名为 users 的 MySQL 数据库表。它将查询结果返回到应用程序，在该应用程序中，查询结果将绑定到 DataGrid 控件的 dataProvider 属性并显示在 DataGrid 控件中。应用程序还将新用户的用户名和电子邮件地址发送到 ColdFusion 组件， ColdFusion 组件会在用户数据库表中执行插入操作。

```
<?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>
```

在此应用程序中，RemoteObject 组件的 destination 属性设置为 Coldfusion，source 属性设置为 ColdFusion 组件的完全限定名称。

与之相反，当使用 LiveCycle Data Services 或 BlazeDS 时，需要在配置文件 （默认情况下为 remoting-config.xml 文件）内为远程服务目标的 source 属性指定一个完全限定的类名。并在 RemoteObject 组件的 destination 属性中指定目标的名称。目标类还必须具有不带任何参数的构造函数。在使用 ColdFusion 时，您可以选择按这种方式配置目标，而不是使用 RemoteObject 组件的 source 属性。

ColdFusion 组件

应用程序调用下面的 ColdFusion 组件。此 ColdFusion 代码执行 SQL 数据库插入和查询，并将查询结果返回到应用程序。ColdFusion 页使用 cfquery 标签在数据库中插入数据并查询数据库，使用 cfreturn 标签将查询结果的格式设置为 ColdFusion 查询对象。

```
<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>
```

在 ActionScript 中调用 RemoteObject 组件

在下面的 ActionScript 示例中，调用 useRemoteObject() 方法即可以声明服务、设置目标、设置 result 和 fault 事件侦听器并调用服务的 getList() 方法。

```
<?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>
```

访问源路径中的 Java 对象

使用 RemoteObject 组件可以访问位于 LiveCycle Data Services、BlazeDS 或 ColdFusion Web 应用程序的源路径中的无状态和有状态的 Java 对象。可以将独立类文件放在 Web 应用程序的 WEB-INF/classes 目录中，以便将它们添加到源路径中。可以将 Java Archive (JAR) 文件中所包含的类放在 Web 应用程序的 WEB-INF/lib 目录中，以便将它们添加到源路径中。在 LiveCycle Data Services、BlazeDS 或 ColdFusion 的 services-config.xml 文件中，或者在它通过引用而包括的文件（如 remoting-config.xml 文件）中，在远程服务目标的 source 属性中指定一个完全限定的类名。该类还必须具有不带任何参数的构造函数。对于 ColdFusion，可以选择将 RemoteObject 组件的 destination 属性设置为 Coldfusion，将 source 属性设置为 ColdFusion 组件或 Java 类的完全限定名称。

在配置远程服务目标以访问无状态的对象（请求作用域）时，Flex 会为每个方法调用创建一个不同的对象，而不是对同一个对象调用多个方法。您可以将对象的作用域设置为请求作用域（默认值）、应用程序作用域或会话作用域。应用程序作用域中的对象可供包含该对象的 Web 应用程序使用。会话作用域中的对象可用于整个客户端会话。

在配置远程对象目标以访问有状态的对象时，Flex 会在服务器上创建一次该对象，并在方法调用之间保持该对象的状态。如果将对象存储在应用程序作用域或会话作用域中会导致内存问题，请使用请求作用域。

访问 JNDI 中的 EJB 和其它对象

可以访问 Java 命名和目录接口 (JNDI) 中存储的 Enterprise JavaBeans (EJB) 和其它对象，方法是对目标调用方法，该方法是在 JNDI 中查找对象并调用其方法的服务 facade 类。

可以使用无状态对象或有状态对象来调用 Enterprise JavaBeans 和其它使用 JNDI 的对象的方法。对于 EJB，可以调用服务 facade 类，该类返回 JNDI 中的 EJB 对象并针对该 EJB 调用某个方法。

在 Java 类中，可以使用标准的 Java 编码模式，在该模式下，可以创建初始上下文并执行 JNDI 查找。对于 EJB，也可以使用标准编码模式，在该模式下，您所创建的类中可以包含调用 EJB 主对象的 create() 方法和所得到 EJB 的业务方法。

以下示例在 facade 类目标上使用了一个名为 getHelloData() 的方法：

```
<mx:RemoteObject id="Hello" destination="roDest">
    <mx:method name="getHelloData"/>
</mx:RemoteObject>
```

在 Java 端，getHelloData() 方法可以封装针对 EJB 调用业务方法所必需的全部内容。以下示例中的 Java 方法会执行如下操作：

- 新建一个用来调用 EJB 的初始上下文
- 执行一个用来获取 EJB 主对象的 JNDI 查找
- 调用 EJB 主对象的 create() 方法
- 调用 EJB 的 sayHello() 方法

```
...
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);
    }
...
```

保留的方法名称

Flex 远程库使用下面的方法名称；请不要将这些方法名称用于您自己的方法：

```
addHeader()
addProperty()
deleteHeader()
hasOwnProperty()
isPropertyEnumerable()
isPrototypeOf()
registerClass()
toLocaleString()
toString()
unwatch()
valueOf()
watch()
```

另外，方法名称不得以下划线 (_) 字符开头。

只需按服务变量来命名 RemoteObject 方法 （操作），即可使它们可访问。但是，如果操作名称恰巧与针对服务定义的方法一样，就会发生命名冲突。可以针对 RemoteObject 组件使用以下 ActionScript 方法来返回给定名称的操作：

```
public function getOperation(name:String):Operation
```

ActionScript 和 Java 之间的序列化

LiveCycle Data Services 和 BlazeDS 在两个方向上对 ActionScript (AMF 3) 与 Java 和 ColdFusion 数据类型之间的数据进行序列化。有关 ColdFusion 数据类型的信息，请参阅 ColdFusion 文档集。

将数据从 **ActionScript** 转换为 **Java**

当方法参数将数据从应用程序发送到 Java 对象时，数据会自动从 ActionScript 数据类型转换为 Java 数据类型。当 LiveCycle Data Services 或 BlazeDS 搜索 Java 对象的合适的方法时，它会进一步使用更宽松的转换来查找匹配项。

客户端上的简单数据类型（如 Boolean 和 String 值）通常与远程 API 完全匹配。但是， Flex 在搜索 Java 对象的合适的方法时，会尝试一些简单的转换。

ActionScript Array 可以通过两种方法来编制条目索引。严格 Array 中的所有索引均为 Number 类型。关联 Array 中至少有一个索引是基于 String 类型的。一定要了解向服务器发送的 Array 的类型，因为 Array 类型会更改用来针对 Java 对象调用方法的参数的数据类型。密集 Array 中所有的数值索引都是连续的 （即没有间隔），从 0（零）开始。稀疏 Array 的数值索引之间有间隔；这种 Array 作为对象来处理，数值索引会变为属性，这些属性会序列化为 java.util.Map 对象，以免发送许多空条目。

下表列出了对于简单数据类型所支持的 ActionScript (AMF 3) 到 Java 转换。

ActionScript 类型 (AMF 3)	**反序列化为 Java**	**支持的 Java 类型绑定**
Array （密集）	java.util.List	java.util.Collection, Object[ ] (本机数组) 如果类型是一个接口，则会映射到下面的接口实现： • List 变为 ArrayList • SortedSet 变为 TreeSet • Set 变为 HashSet • Collection 变为 ArrayList 自定义 Collection 实现的新实例会绑定到该类型。
Array （稀疏）	java.util.Map	java.util.Map
Boolean 字符串 "true" 或 "false"	java.lang.Boolean	Boolean、 boolean 和 String
flash.utils.ByteArray	byte []	
flash.utils.IExternalizable	java.io.Externalizable	
Date	java.util.Date （已设置为协调世界时 (UTC) 格式）	java.util.Date、 java.util.Calendar、 java.sql.Timestamp、 java.sql.Time 和 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，以及基元类型 double、 long、 float、 int、 short 和 byte
null	null	基元
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（零） 如果发送了 null，则为基元类型 double、 long、 float、 int、 short 和 byte

ActionScript 类型 (AMF 3)	反序列化为 Java	支持的 Java 类型绑定
Object （泛型）	java.util.Map	如果指定了 Map 接口，请为 java.util.Map 创建一个 java.util.HashMap，并为 java.util.SortedMap 创建一个新的 java.util.TreeMap。
String	java.lang.String	java.lang.String、java.lang.Boolean、java.lang.Number、java.math.BigInteger、java.math.BigDecimal、char[]、以及任何基元数字类型
有类型 Object	有类型 Object 在使用 [RemoteClass] 元数据标签指定远程类名称时。 Bean 类型必须具有公共的无参数构造函数。	有类型 Object
undefined	null	null （对于对象）和默认值 （对于基元）
XML	org.w3c.dom.Document	org.w3c.dom.Document
XMLDocument （旧 XML 类型）	org.w3c.dom.Document	org.w3c.dom.Document 可以针对在 services-config.xml 文件中定义的任何通道启用对于 XMLDocument 类型的旧 XML 支持。此设置仅在将数据从服务器发回到客户端时很重要，它控制 org.w3c.dom.Document 实例如何发送到 ActionScript。有关更多信息，请参阅为通道配置 AMF 序列化。

在 Java 中，基元值不能设置为 null。在将 Boolean 和 Number 值从客户端传递到 Java 对象时，Flex 将 null 值解释为基元类型的默认值，例如，0（对于 double、float、long、int、short 和 byte）、\u0000（对于 char）和 false（对于 Boolean）。只有基元 Java 类型才能获得默认值。

LiveCycle Data Services 和 BlazeDS 会像处理任何其他设置了类型的对象那样处理 java.lang.Throwable 对象。处理 java.lang.Throwable 对象时遵循的规则是查找公共字段和 bean 属性，并将有类型 Object 返回到客户端。这些规则与普通的 bean 规则相似，区别在于这些规则会检查 getter 是否具有只读属性。这允许您从 Java 异常获取更多信息。如果需要 Throwable 对象的旧行为，则可以将通道的 legacy-throwable 属性设置为 true ；有关更多信息，请参阅为通道配置 AMF 序列化。

可以将严格 Array 以参数形式传递给要求 java.util.Collection 实现或本机 Java Array API 的方法。

Java Collection 可以包含任意数量的 Object 类型，而 Java Array 要求各个条目具有相同的类型（例如，java.lang.Object[ ] 和 int[ ]）。

LiveCycle Data Services 和 BlazeDS 还将 ActionScript 严格数组转换为常见的 Collection API 接口的适当实现。例如，如果将 ActionScript 严格数组发送到 Java 对象方法 public void addProducts(java.util.Set products)，则 LiveCycle Data Services 和 BlazeDS 会先将其转换为 java.util.HashSet 实例，然后再以参数形式传递它，这是因为 HashSet 是 java.util.Set 接口的适当实现。同样，LiveCycle Data Services 和 BlazeDS 会将 java.util.TreeSet 实例传递到用 java.util.SortedSet 接口设置类型的参数。

LiveCycle Data Services 和 BlazeDS 会将 java.util.ArrayList 实例传递到使用 java.util.List 接口以及任何其他扩展了 java.util.Collection 的接口来设置类型的参数。这些类型随后将作为 mx.collections.ArrayCollection 实例发回到客户端。如果需要将标准 ActionScript Array 发送回客户端，则必须在通道定义的属性的 serialization 部分中将 legacy-collection 元素设置为 true。有关更多信息，请参阅为通道配置 AMF 序列化。

显式映射 **ActionScript** 和 **Java** 对象

对于 LiveCycle Data Services 和 BlazeDS 不会隐式处理的 Java 对象，会将通过 get/set 方法在公共 bean 属性中找到的值以及公共变量作为对象的属性发送到客户端。私有属性、常量、静态属性和只读属性等将不会进行序列化。对于 ActionScript 对象，用 get/set 访问器和公共变量定义的公共属性将发送到服务器。

LiveCycle Data Services 和 BlazeDS 使用标准 Java 类 (java.beans.Introspector) 获取 JavaBean 类的属性描述符。还使用反射功能收集类的公共字段，并且优先使用 bean 属性 （而不是字段）。 Java 属性名与 ActionScript 属性名应该匹配。本机 Flash Player 代码用来确定 ActionScript 类如何在客户端上进行内部检查。

在 ActionScript 类中，可以使用 [RemoteClass(alias=" ")] 元数据标签来创建直接映射到 Java 对象的 ActionScript 对象。数据所转换到的 ActionScript 类必须在 MXML 文件中使用或引用，才能链接到 SWF 文件并在运行时可用。实现上述操作的一个好办法是对结果对象进行强制转换，如下面的示例所示：

```
var result:MyClass = MyClass(event.result);
```

类本身应该使用强类型引用，以便它的依赖项也可以被链接。

下面的示例显示了一个使用 [RemoteClass(alias=" ")] 元数据标签的 ActionScript 类的源代码：

```
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;
    }
}
```

如果您不映射到服务器上的 Java 对象，但需要将对象类型从服务器发回，则可以使用没有别名的 [RemoteClass] 元数据标签。在将 ActionScript 对象发送到服务器时，它会序列化为一个特殊的 Map 对象，但是从服务器返回到客户端的对象还是最初的 ActionScript 类型。

为了禁止将特定属性从 ActionScript 类发送到服务器，需要在 ActionScript 类中该属性声明的上方使用 [Transient] 元数据标签。

将数据从 **Java** 转换为 **ActionScript**

从 Java 方法返回的对象将从 Java 转换为 ActionScript。LiveCycle Data Services 和 BlazeDS 还处理在对象中找到的对象。LiveCycle Data Services 隐式处理下表中的 Java 数据类型。

Java 类型	**ActionScript 类型 (AMF 3)**
java.lang.String	String
java.lang.Boolean, boolean	Boolean
java.lang.Integer, int	int 如果值小于 0xF0000000 且大于 0x0FFFFFFF，则会按照 AMF 编码要求将值提升为 Number。
java.lang.Short, short	int 如果 i 小于 0xF0000000 且大于 0x0FFFFFFF，则会将值提升为 Number。
java.lang.Byte, byte[]	int 如果 i 小于 0xF0000000 且大于 0x0FFFFFFF，则会将值提升为 Number。

Java 类型	ActionScript 类型 (AMF 3)
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 日期按照协调世界时 (UTC) 时区的时间进行发送。客户端和服务器必须根据时区相应地调整时间。
java.util.Date	Date 日期按照 UTC 时区的时间进行发送。客户端和服务器必须根据时区相应地调整时间。
java.util.Collection （例如，java.util.ArrayList）	mx.collections.ArrayCollection
java.lang.Object[]	Array
java.util.Map	Object （无类型）。例如，将 java.util.Map[] 转换为对象的 Array。
java.util.Dictionary	Object （无类型）
org.w3c.dom.Document	XML 对象
null	null
java.lang.Object （以前列出的类型除外）	有类型 Object 通过使用 JavaBean 内部检查规则将对象进行序列化，并且对象包括公共字段。不包括静态字段、瞬态字段、非公共字段，以及非公共 bean 属性或静态 bean 属性。

为通道配置 **AMF** 序列化

您可以支持早期版本的 Flex 中所使用的旧 AMF 类型序列化，并在 services-config.xml 文件的通道定义中配置其它序列化属性。

下表描述了可以在通道定义的 <serialization> 元素中设置的属性：

属性	描述
`<ignore-property-errors>true</ignore-property-errors>`	默认值为 true。确定在传入的客户端对象具有无法在服务器对象上设置的意外属性时，端点是否应该引发错误。
`<log-property-errors>false</log-property-errors>`	默认值为 false。如果为 true，则会记录意外的属性错误。
`<legacy-collection>false</legacy-collection>`	默认值为 false。如果为 true，则 java.util.Collection 实例将以 ActionScript Array 形式返回。如果为 false，则 java.util.Collection 的实例将以 mx.collections.ArrayCollection 形式返回。
`<legacy-map>false</legacy-map>`	默认值为 false。如果为 true，则 java.util.Map 实例将序列化为 ECMA Array 或关联 Array，而不是匿名 Object。

属性	描述
<legacy-xml>false</legacy-xml>	默认值为 false。如果为 true，则 org.w3c.dom.Document 实例将序列化为 flash.xml.XMLDocument 实例，而不是内部 XML（支持 E4X）实例。
<legacy-throwable>false</legacy-throwable>	默认值为 false。如果为 true，则 java.lang.Throwable 实例将序列化为 AMF 状态信息对象 （而不是普通的 bean 序列化，包括只读属性）。
<type-marshaller>className</type-marshaller>	指定 flex.messaging.io.TypeMarshaller 的一个实现，该实现将对象转换为所需类的实例。在调用 Java 方法或者填充 Java 实例，如果反序列化中的输入对象的类型 （例如， ActionScript 匿名 Object 始终反序列化为 java.util.HashMap）与目标 API （例如， java.util.SortedMap）不匹配，使用该属性。因此，该类型可以封送成所需的类型。
<restore-references>false</restore-references>	默认值为 false。一个高级开关，在必须进行类型转换时，让反序列化程序跟踪对象引用；例如，在为 java.util.SortedMap 类型的属性发送匿名 Object 时，该 Object 将首先像平常那样反序列化为 java.util.Map，然后转换为合适的 SortedMap 实现 （如 java.util.TreeMap）。如果其它对象指向对象图中的同一个匿名 Object，则该设置将复原这些引用，而不是到处创建 SortedMap 实现。请注意，如果将此属性设置为 true，在存在大量数据时，将会大大降低性能。
<instantiate-types>true</instantiate-types>	默认值为 true。一个高级开关，如果设置为 false，会禁止反序列化程序创建强类型对象的实例，而是保留类型信息并反序列化 Map 实现 （即 flex.messaging.io.ASObject）中的原始属性。请注意 flex.* 包下面的任何类总是会进行实例化。

使用自定义序列化

如果用来在客户端上的 ActionScript 和服务器上的 Java 之间序列化和反序列化数据的标准机制无法满足您的需要，您可以编写自己的序列化方案。您可以在客户端实现基于 ActionScript 的 flash.utils.IExternalizable 接口，在服务器上实现基于 Java 的相应 java.io.Externalizable 接口。

使用自定义序列化的一个典型原因是，避免在整个网络层传递某个对象的客户端或服务器端表示的所有属性。在实现自定义序列化时，可以对类进行编码，从而使仅在客户端或仅在服务器端具有的特定属性不通过网络传递。在使用标准的序列化方案时，所有的公共属性都在客户端和服务器之间来回传递。

在客户端，用来实现 flash.utils.IExternalizable 接口的类的标识是在序列化流中编写的。该类序列化和重新构造其实例的状态。该类实现 IExternalizable 接口的 writeExternal() 和 readExternal() 方法，以控制对象及其父类型的序列化流的内容和格式，而不是类名或类型。这些方法取代了本机的 AMF 序列化行为。这些方法必须与其远程对应部分对称，才能保存类的状态。

在服务器端，用来实现 java.io.Externalizable 接口的 Java 类与用来实现 flash.utils.IExternalizable 接口的 ActionScript 类执行相似的功能。

注：如果需要使用精确的按引用序列化，请不要使用通过 HTTPChannel 实现 IExternalizable 接口的类型。如果这样做，重复出现的对象之间的引用将丢失而且似乎在端点处克隆。

下面的示例显示一个客户端 (ActionScript) 版本的 Product 类的完整源代码，该类映射到服务器端的基于 Java 的 Product 类。客户端 Product 实现 IExternalizable 接口，而服务器端 Product 实现 Externalizable 接口。

```
// 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);
    }
}
}
```

客户端 Product 使用两种序列化：与 java.io.Externalizable 接口兼容的标准序列化； AMF 3 序列化。下面的示例显示客户端 Product 的 writeExternal() 方法。该方法同时使用这两种类型的序列化：

```
public function writeExternal(output:IDataOutput):void {
    output.writeObject(name);
    output.writeObject(properties);
    output.writeFloat(price);
}
```

如下面的示例所示，服务器端 Product 的 writeExternal() 方法与该方法的客户端版本几乎完全相同：

```
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeObject(name);
    out.writeObject(properties);
    out.writeFloat(price);
}
```

在客户端 Product 的 writeExternal() 方法中，flash.utils.IDataOutput.writeFloat() 方法是标准序列化方法的一个示例，能够满足用于处理基元类型的 Java java.io.DataInput.readFloat() 方法的规范。此方法向服务器端 Product 发送 price 属性 （是一个 Float 值）。

在客户端 Product 的 writeExternal() 方法中，使用 AMF 3 序列化的示例是对 flash.utils.IDataOutput.writeObject() 方法的调用，该方法映射到服务器端 Product 的 readExternal() 方法中的 java.io.ObjectInput.readObject() 方法调用。flash.utils.IDataOutput.writeObject() 方法向服务器端 Product 发送 properties 属性 （该属性是一个 Object）和 name 属性（该属性是一个 String）。之所以能够实现这一点，是因为 AMFChannel 端点中实现了一个 java.io.ObjectInput 接口，该接口获得 writeObject() 方法发送的数据以将其设置为 AMF 3 格式。

反过来，当在服务器端 Product 的 readExternal() 方法中调用 readObject() 方法时，将使用 AMF 3 反序列化；这正是将 ActionScript 版本的 properties 值假定为 Map 类型，而将 name 的类型假定为 String 的原因。

下面的示例显示了服务器端 Product 类的完整源代码：

```
// 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;
        }
}
```

下面的示例显示了服务器端 Product 的 readExternal() 方法：

```
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));
    }
```

在序列化期间，客户端 Product 的 writeExternal() 方法不向服务器发送 id 属性，因为该属性对于服务器版本的 Product 对象没有用处。同样，服务器端 Product 的 writeExternal() 方法不向客户端发送 inventoryId 属性，因为该属性是一个特定于服务器的属性。

请注意，在序列化过程中，在任何一个方向都不发送 Product 的属性名。由于类的状态是固定而且可管理的，因此属性将按照具有完善定义的顺序发送而且发送时不带名称， readExternal() 方法会按照相应的顺序读取它们。

显式参数传递和参数绑定

可通过两种不同的方法调用 HTTPService、 WebService 和 RemoteObject 组件：显式参数传递和参数绑定。在使用显式参数传递时，会将输入内容以 ActionScript 函数的参数形式提供给服务。这种调用服务的方式与 Java 中的方法调用极其相似。不能将 Flex 数据验证程序自动与显式参数传递结合使用。

使用参数绑定，可以将数据从用户界面控件或模型复制到请求参数。参数绑定仅适用于在 MXML 中声明的数据访问组件。在向服务提交请求之前，可以向参数值应用验证程序。有关数据绑定和数据模型的更多信息，请参阅 Data binding 和 Storing data。有关数据验证的更多信息，请参阅 Validating Data。

在使用参数绑定时，可以采用以下方式：在 <mx:method> 标签下的 <mx:arguments> 标签中声明 RemoteObject 方法参数标签；在 <mx:request> 标签内声明 HTTPService 参数标签；在 <mx:operation> 标签下的 <mx:request> 标签内声明 WebService 操作参数标签。可以使用 send() 方法发送请求。

对 RemoteObject 和 WebService 组件使用显式参数传递

对 RemoteObject 和 WebService 组件使用显式参数传递的方法非常相似。以下示例显示的 MXML 代码声明了一个 RemoteObject 组件，并通过在 Button 控件的 click 事件侦听器中使用显式参数传递来调用服务。ComboBox 控件向服务提供数据。简单的事件侦听器处理服务级别的 result 和 fault 事件。

```
<?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>
```

对 HTTPService 组件使用显式参数传递

对 HTTPService 组件使用显式参数传递与对 RemoteObject 和 WebService 组件使用显式参数传递不同。总是使用 HTTPService 组件的 send() 方法来调用服务，这不同于 RemoteObject 和 WebService 组件。针对 RemoteObject 和 WebService 组件调用的方法是客户端版本的 RPC 服务方法或操作。

在使用显式参数传递时，可以指定一个包含名称 / 值对的对象作为 send() 方法参数。 send() 方法参数必须是简单的基本类型；不能使用复杂的嵌套类型，因为无法使用通用的方法将它们转换为名称 / 值对。

如果没有为 send() 方法指定参数，则 HTTPService 组件会使用在 <mx:request> 标签中指定的任何查询参数。

以下两个示例显示了两种使用带参数的 send() 方法来调用 HTTP 服务的方式。第二个示例还显示了如何调用 cancel() 方法以取消 HTTP 服务调用。

```
<?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>
```

对 RemoteObject 组件使用参数绑定

对 RemoteObject 组件使用参数绑定时，始终会在 RemoteObject 组件的 <mx:method> 标签中声明方法。

<mx:method> 标签中可以包含一个 <mx:arguments> 标签，后者可以包含方法参数的子标签。<mx:method> 标签的 name 属性必须与服务的某个方法名称相匹配。参数标签的顺序必须与服务的方法参数的顺序相匹配。您可以对参数标签进行命名，使其与相应方法参数的实际名称尽可能密切匹配，但这没有必要。

注：如果 <mx:arguments> 标签内部有同名的参数标签，而且远程方法不将数组作为唯一的输入源，则服务调用会失败。在编译应用程序时，不会就此发出警告。

可以将数据绑定到 RemoteObject 组件的方法参数。可以引用参数的标签名称进行数据绑定和验证。

下面示例中显示的方法将两个参数绑定到 TextInput 控件的文本属性。 PhoneNumberValidator 验证程序被指定给 arg1（即第一个参数标签的名称）。

```
<?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 按照 MXML 标签所指定的顺序将参数标签发送给方法。

下面的示例在 RemoteObject 组件的 <mx:method> 标签中使用参数绑定，当用户单击 Button 按钮时，会将选定 ComboBox 项的数据绑定到 employeeRO.getList 操作。在使用参数绑定时，可以使用不带参数的 send() 方法来调用服务。

```
<?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>
```

如果不确信服务调用的结果是包含数组还是单个对象，您可以使用 mx.utils.ArrayUtil 类的 toArray() 方法将其转换为数组，如本例所示。如果将 toArray() 方法传递给单个对象，则该方法会返回一个数组，而该对象是唯一的 Array 元素。如果向方法传递一个数组，则该方法将返回同一个数组。有关处理 ArrayCollection 对象的信息，请参阅 Data providers and collections。

对 HTTPService 组件使用参数绑定

当 HTTP 服务需要查询参数时，可以将这些参数声明为 <mx:request> 标签的子标签。标签的名称必须与服务所需要的查询参数的名称相匹配。

下面的示例在 HTTPService 组件的 <mx:request> 标签中使用参数绑定，当用户单击 Button 按钮时，会将选定 ComboBox 项的数据绑定到 employeeSrv 请求。在使用参数绑定时，可以使用不带参数的 send() 方法来调用服务。此示例显示 HTTPService 组件的 url 属性，但是，无论是直接连接服务还是通过目标连接，调用服务的方式都是相同的。

```
<?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>
```

如果不确信服务调用的结果是包含数组还是单个对象，您可以使用 mx.utils.ArrayUtil 类的 toArray() 方法将其转换为数组，如前一个示例所示。如果将 toArray() 方法传递给单个对象，则该方法会返回一个数组，而该对象是唯一的 Array 元素。如果向方法传递一个数组，则该方法将返回同一个数组。有关处理 ArrayCollection 对象的信息，请参阅 Data providers and collections。

对 WebService 组件使用参数绑定

对 WebService 组件使用参数绑定时，始终会在 WebService 组件的 <mx:operation> 标签中声明操作。<mx:operation> 标签中可以包含一个 <mx:request> 标签，后者可以包含操作所期望的 XML 节点。<mx:operation> 标签的 name 属性必须与 Web 服务的某个操作名称相匹配。

可以将数据绑定到 Web 服务操作的参数。可以引用参数的标签名称进行数据绑定和验证。

下面的示例在 WebService 组件的 <mx:operation> 标签中使用参数绑定，当用户单击 Button 按钮时，会将选定 ComboBox 项的数据绑定到 employeeWS.getList 操作。<deptId> 标签与 getList 操作的 deptId 参数直接对应。在使用参数绑定时，可以使用不带参数的 send() 方法来调用服务。此示例显示 WebService 组件的 destination 属性，但是，无论是直接连接服务还是通过目标连接，调用服务的方法都是相同的。

```
<?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>
```

还可以在 XML 中手动指定整个 SOAP 请求主体，其中包含在 <mx:request> 标签中定义的所有正确的命名空间信息。为此，请将 <mx:request> 标签的 format 属性值设置为 xml，如下面的示例所示：

```
<?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>
```

处理服务结果

在 RPC 组件调用服务之后，服务返回的数据将放置在 lastResult 对象中。默认情况下，HTTPService 组件和 WebService 组件操作的 resultFormat 属性值为 object，而返回的数据以 ActionScript 对象的简单树形式来表示。 Flex 解释 Web 服务或 HTTP 服务返回的 XML 数据，以便相应地表示基本类型（如 String、 Number、 Boolean 和 Date）。要使用强类型对象，请使用 Flex 所创建的对象树来填充这些对象。

WebService 和 HTTPService 组件均返回作为复杂类型的匿名 Object 和 Array。如果 makeObjectsBindable 为 true （默认值），则 Object 将包装在 mx.utils.ObjectProxy 实例中，而数组将包装在 mx.collections.ArrayCollection 实例中。

注：ColdFusion 不区分大小写，因此它在内部将其所有数据都设置为大写。在使用 ColdFusion Web 服务时，请记住这一点。

处理 result 和 fault 事件

完成服务调用之后， RemoteObject 方法、WebService 操作或 HTTPService 组件发送 result 事件或 fault 事件。result 事件指示是否有可用结果。fault 事件指示出现了错误。result 事件充当触发器，用于更新绑定到 lastResult 的属性。通过将事件侦听器添加到 RemoteObject 方法或 WebService 操作，可以显式处理 fault 和 result 事件。对于 HTTPService 组件，在该组件本身中指定 result 和 fault 事件侦听器，原因是 HTTPService 组件没有多个操作和方法。

当在 RemoteObject 方法 或 WebService 操作中没有为 result 或 fault 事件指定事件侦听器时，这些事件传递到组件级；您可以指定组件级 result 和 fault 事件侦听器。

在以下 MXML 示例中， WebService 操作的 result 和 fault 事件指定事件侦听器； WebService 组件的 fault 事件也指定事件侦听器：

```
<?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>
```

在以下 ActionScript 示例中，将 result 事件侦听器添加到 WebService 操作中；将 fault 事件侦听器添加到 WebService 组件中：

```
<?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>
```

也可以使用 mx.rpc.events.InvokeEvent 事件指明何时已调用数据访问组件请求。如果操作会排队并且在稍后调用，则此将很有用。

使用 e4x 结果格式将结果作为 XML 处理

可以将 HTTPService 组件和 WebService 操作的 resultFormat 属性值设置为 e4x 以创建类型为 XML 的 lastResult 属性。通过使用 ECMAScript for XML (E4X) 表达式，可以访问 lastResult 属性。在绑定表达式中使用 E4X XML 对象时，不需要在点标记中包括 XML 结构的根节点。这不同于设置为 object 的 lastResult 属性的语法 （其需要在点标记中包括 XML 结构的根节点）。例如，如果 lastResult 属性设置为 e4x，应该使用 {srv.lastResult.product} ；如果 lastResult 属性设置为 object，应该使用 {srv.lastResult.products.product}。

使用 e4x 的结果格式是直接使用 XML 的首选方式，但是，也可以将 resultFormat 属性设置为 xml 以创建类型为 flash.xml.XMLNode 的 lastResult 对象，这是使用 XML 的旧对象。另外，还可以将 HTTPService 组件的 resultFormat 属性设置为 flashvars 以创建包含名称 / 值对的 ActionScript 对象形式的结果，或者设置为 text 以创建原始文本形式的结果。

注：要针对服务结果使用 E4X 语法，请将 HTTPService 或 WebService 组件的 resultFormat 属性设置为 e4x。默认值为 object。

如果将 HTTPService 组件或 WebService 操作的 resultFormat 属性设置为 e4x，则可能需要处理 XML 中包含的返回的命名空间信息。对于 WebService 组件， Web 服务返回的命名空间信息包含在 SOAP 封套的主体中。以下示例显示了一个包含命名空间信息的 SOAP 主体的一部分。此数据由用来获取股票报价的 Web 服务返回。命名空间信息以粗体文本显示。

```
...
<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>
...
```

由于此 soap:Body 包含命名空间信息，因此，如果将 WebService 操作的 resultFormat 属性设置为 e4x，则会为 http://ws.invesbot.com/namespace 创建命名空间对象。以下示例显示了一个执行该操作的应用程序：

```
<?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>
```

您也可以选择为命名空间创建一个变量，并在服务结果绑定中访问该变量，如以下示例所示：

```
<?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>
```

可以使用 E4X 语法来访问在 lastResult 对象中返回的 XML 的元素和属性。可以根据在 XML 中是否声明了命名空间来使用不同的语法。

无命名空间

以下示例显示在未针对元素或属性指定命名空间时，如何获得元素值或属性值：

```
var attributes:XMLList = XML(event.result).Description.value;
```

上面的代码返回以下 XML 文档的 xxx：

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <Description>
        <value>xxx</value>
    </Description>
</RDF>
```

任何命名空间

以下示例显示在针对元素或属性指定了命名空间时，如何获得元素值或属性值：

```
var attributes:XMLList = XML(event.result).*::Description.*::value;
```

上面的代码返回下列任一 XML 文档的 xxx：

XML 文档 1：

```
<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>
```

XML 文档 2：

```
<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>
```

特定命名空间

以下示例显示在针对元素或属性指定了所声明的 rdf 命名空间时，如何获得元素值或属性值：

```
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;
```

上面的代码返回以下 XML 文档的 xxx：

```
<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>
```

下面的示例显示在针对元素或属性指定了所声明的 rdf 命名空间时，获得元素值或属性值的另一种方法：

```
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;
```

上面的代码也是返回以下 XML 文档的 xxx：

```
<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>
```

处理包含 .NET DataSet 或 DataTable 的 Web 服务结果

用 Microsoft .NET Framework 编写的 Web 服务可以向客户端返回特殊的 .NET DataSet 或 DataTable 对象。.NET Web 服务提供基本的 WSDL 文档，此文档不包含关于该服务所操纵的数据类型的信息。当 Web 服务返回 DataSet 或 DataTable 时，会在 SOAP 消息中的 XML 架构元素中嵌入数据类型信息，该信息指定应该如何处理 SOAP 消息的剩余部分。为了更好地处理来自这种类型的 Web 服务的结果，可以将 Flex WebService 操作的 resultFormat 属性设置为 object。您也可以选择将 WebService 操作的 resultFormat 属性设置为 e4x，但是 XML 和 e4x 格式不便于使用，原因在于，例如，如果您希望将数据绑定到 DataGrid 控件，则必须在不常见的响应结构中导航并实现解决方法。

将 Flex WebService 操作的 resultFormat 属性设置为 object 时，从 .NET Web 服务返回的 DataTable 或 DataSet 会自动转换为具有 Tables 属性的对象，Tables 属性中包含一个或多个 DataTable 对象的映射。每个来自 Tables 映射的 DataTable 对象都包含两个属性: Columns 和 Rows。Rows 属性中包含数据。event.result 对象获得与 .NET 中的 DataSet 和 DataTable 属性相对应的以下属性。 DataSet 或 DataTable 数组的结构与此处描述的结构相同，但嵌套在 result 对象的顶层数组中。

属性	描述
`result.Tables`	表名到包含表数据的对象的映射。
`result.Tables["someTable"].Columns`	列名的数组，采用表的 DataSet 或 DataTable 架构中指定的顺序。
`result.Tables["someTable"].Rows`	表示每个表行中数据的对象数组。例如， {columnName1:value, columnName2:value, columnName3:value}。

下面的 MXML 应用程序使用从 .NET Web 服务返回的 DataTable 数据来填充 DataGrid 控件。

```
<?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>
```

下面的示例显示一个 .NET C# 类，该类是一种由应用程序调用的后端 Web 服务实现，它使用 Microsoft SQL Server Northwind 示例数据库：

```
<%@ 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;
    }
}
```