Глава 5

Введение в серверные элементы управления ASP.NET AJAX

В трех первых главах этой книги был дан обзор ASP.NET AJAX и использования этой технологии для создания веб-приложений. AJAX позволяет обойтись без ненужных перезагрузок страниц и излишней обработки, повышая таким образом производительность и внешнюю привлекательность веб-приложений. В главах 3 и 4 рассматривались представленные в ASP.NET AJAX клиентские элементы управления, а также примеры их использования в JavaScript и новом, основанном на XML сценарии, который называется ASP.NET AJAX Library.

Кроме того, в предыдущих главах рассматривались некоторые более сложные аспекты платформы для программирования сценариев: действия (actions), которые представляют собой составные команды, связанные с событием или входным сигналом в элементе управления; поведение (behaviors) — автоматические функциональные блоки, которые могут быть связаны с элементом управления и позволяют осуществлять такие действия, как перетаскивание элемента управления на веб-странице; привязка данных (data binding), которая позволяет привязывать элементы управления друг к другу или к самим себе для передачи данных.

В данной главе рассматривается другая сторона системы — сервер и начинается изучение различных серверных элементов управления, которые можно использовать при создании AJAX-приложений. Выше уже описывался один из таких элементов управления `ScriptManager`. В этой главе `ScriptManager` и другие серверные элементы управления ASP.NET AJAX рассматриваются более подробно. В главе 6 подробно описывается работа этих элементов управления на примере приложения, в котором они фактически используются.

Использование серверных элементов управления ASP.NET AJAX в Visual Studio 2005

Среда Visual Studio 2005 и ASP.NET предоставляют разработчику ряд великолепных инструментов, которые согласуются с представленными в ASP.NET AJAX концепциями и позволяют визуально конструировать страницы. Разработчики могут размещать элементы управления на странице, а эти элементы генерируют JavaScript-код, который необходим для реализации AJAX-функциональности. В следующих разделах рассматривается использование этих элементов управления в интегрированной среде разработки (integrated development environment — IDE).

В главе 3 описывалось создание на базе ASP.NET нового сайта с AJAX-функциональностью. Если вы установили добавочный модуль Web Application Project (проект типа веб-приложение в Visual Studio 2005) или пакет исправлений SP1 для Visual Studio 2005, то создать на базе ASP.NET веб-приложение с AJAX почти так же просто. В любом случае при создании нового проекта на панели инструментов (Toolbox) будет доступен новый раздел — AJAX Extensions (AJAX-расширения), рис. 5.1.

Элементы управления можно перемещать с панели на веб-формы. В оставшейся части этой главы описываются эти элементы и их объектные модели. В следующей главе начинается их использование в практических примерах. На момент написания этой книги в первый выпуск ASP.NET AJAX было включено пять серверных элементов управления: `Timer`, `ScriptManager`, `ScriptManagerProxy`, `UpdateProgress` и `UpdatePanel`. В настоящее время дополнительные элементы управления включены в сборку ASP.NET AJAX под названием Futures CTP, которая должна появиться в следующих выпусках ASP.NET AJAX.

Использование серверных элементов управления ASP.NET AJAX — это самый простой и быстрый способ реализовать AJAX-функциональность в ASP.NET-приложении. Они также идеально подходят, когда речь идет о минимальных изменениях существующего ASP.NET-проекта, которые предполагают широкое использование серверных элементов управления.

Примечание
Если планируется разрабатывать в Visual Studio 2005 веб-приложения AJAX (т.е. проекты, которые придерживаются модели веб-приложений, а не веб-сайтов ASP.NET), то ASP.NET AJAX следует устанавливать после инсталляции SP1 для Visual Studio 2005.

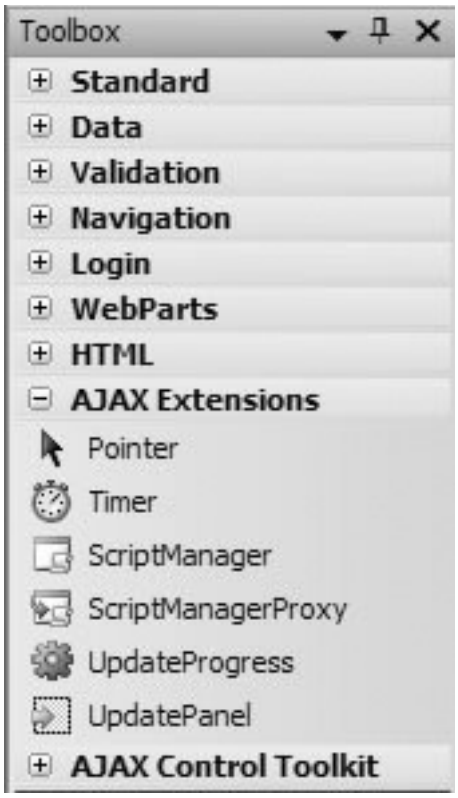

Рис. 5.1. Раздел панели инструментов, содержащий серверные элементы управления AJAX

Введение в `ScriptManager`

Элемент управления `ScriptManager` находится в самом сердце ASP.NET AJAX и является ее главным компонентом. Этот элемент управления, как можно догадаться из названия, управляет развертыванием различных JavaScript-библиотек, которые во время выполнения реализуют клиентскую функциональность ASP.NET AJAX. `ScriptManager` также широко используется другими серверными элементами управления для обеспечения частичной визуализации страницы и управления файлами сценариев.

Использование `ScriptManager`

Выше уже было показано использование `ScriptManager` для создания ссылки на клиентскую сторону с ASP.NET AJAX Library. Для добавления элемен-

Рис. 5.2. Элемент управления ScriptManager

та ScriptManager на страницу его следует просто перетащить на ASP.NET-страницу, как показано на рис. 5.2.

Теперь, если посмотреть на код этой страницы, то можно заметить, что добавление элемента ScriptManager привело к появлению следующего кода:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
```

Если запустить страницу, а затем выбрать пункт меню View➪TRA Source (Вид➪Исходный код) в браузере, станет ясно, что предыдущий однострочный сценарий сгенерировал следующий код.

```
<script src="/Ajax/WebResource.axd?d=
HQhspev9RtnoVp5Ca4MubA2&amp;
t=633008366579531250" type="text/javascript">
</script>
<script src="/Ajax/ScriptResource.axd?d=rbfRw_
fjV44N4zFu5uugvXCg0fpE5bOdbRFvvkMhZEO1
-ghFYTQ7i9aLWWp9hO2901tgv-pDZFxuTtMikT21d-q8lo-
xXLBcAYv3xq0hiRM1&amp;t=
633051881703906250" type="text/javascript">
</script>
<script src="/Ajax3/ScriptResource.axd?d=rbfRw_
fjV44N4zFu5uugvXCg0fpE5bOdbRFvvkMhZEO
1-ghFYTQ7i9aLWWp9hO2901tgv-pDZFxuTtMikT21d3JhQBwnJ44PsSIlv
SkVAgc1&amp;t=633051881703906250" type="text/javascript">
```

```
</script>
<script type="text/javascript">
//<![CDATA[
Sys.WebForms.PageRequestManager._initialize('ScriptManager1',
      document.getElementById('form1'));
Sys.WebForms.PageRequestManager.getInstance()._updateControls(
[], [], [], 90);
//]]>
</script>
```

Примечание
Поскольку клиентские сценарии генерируются автоматически, ваши результаты могут слегка отличаться от приведенного выше блока кода.

Файлы *ScriptResource.axd* и *WebResource.axd* фактически представляют собой ASP.NET HTTP-обработчики, которые создают клиентский JavaScript-код для обеспечения AJAX-функциональности на веб-странице. Закодированные данные после строки запроса содержат информацию о соответствующих ресурсах. Последний блок содержит клиентские сценарии для любых компонентов на странице. В конфигурационном файле *Web.config* ASP.NET AJAX-проекта имеется следующий блок, который регистрирует упомянутые HTTP-обработчики для использования в проекте.

```
<httpHandlers>
   <add verb="*" path="*_AppService.axd" validate="false"
      type="System.Web.Script.Services.ScriptHandlerFactory,
      System.Web.Extensions, Version=1.0.61025.0,
      Culture=neutral,
      PublicKeyToken=31bf3856ad364e35"/>
   <add verb="GET,HEAD" path="ScriptResource.axd"
      type="System.Web.Handlers.ScriptResourceHandler,
      System.Web.Extensions, Version=1.0.61025.0,
      Culture=neutral,
      PublicKeyToken=31bf3856ad364e35"
   validate="false"/>
</httpHandlers>
```

Программирование с использованием `ScriptManager`

Как основной компонент ASP.NET AJAX, элемент управления `ScriptManager` обладает развитой функциональностью, включая способность обмениваться данными со службами аутентификации ASP.NET, подключаться к веб-службам, визуализировать информацию, специфичную для данной местности (национальную информацию), осуществлять сложное управление сценариями, частично

визуализировать страницу и многие другие возможности. Он наследует эту функциональность от класса Control (в пространстве имен System.Web.UI) и в дополнение к членам этого класса также имеет методы, перечисленные в табл. 5.1.

Таблица 5.1. Методы элемента управления ScriptManager

Имя метода	Функция
GetCurrent	(Статический метод) Получает экземпляр ScriptManager для объекта Page
RegisterAsyncPostBackControl	Регистрирует элемент управления для асинхронных postback-отправок данных
RegisterDataItem	Отправляет данные элементу управления во время частичной визуализации страницы
RegisterDispose	Регистрирует сценарий, который можно использовать для правильного расположения элемента управления внутри UpdatePanel. Этот сценарий выполняется во время расположения элемента UpdatePanel
RegisterExtenderControl	Регистрирует расширяющий элемент управления с существующим экземпляром ScriptManager
RegisterPostBackControl	Регистрирует элемент управления для postback-обмена данными. Этот метод можно использовать для элементов управления, существующих внутри UpdatePanel, для которых необходимы полные postback-отправки данных
RegisterScriptControl	Регистрирует элемент управления сценария с существующим экземпляром ScriptManager
SetFocus	Устанавливает фокус браузера на заданный элемент управления

В табл. 5.2 перечислены свойства элемента управления ScriptManager, за исключением свойств, которые наследуются от классов Control и Object.

Осуществление частичной визуализации

Свойство EnablePartialRendering рассматриваемого элемента управления определяет поведение страницы при обновлениях. Значение false (по умолчанию) приводит к полному обновлению страницы при обмене данными с сервером. Значение true подавляет postback-обмен данными и полное обновление страницы и заменяет их нацеленными и частичными обновлениями. Вме-

Таблица 5.2. Свойства элемента управления `ScriptManager`

Имя свойства	Функция
`AllowCustomErrorsRedirect`	Булево значение, указывающее на то, будет ли использоваться раздел нестандартных ошибок в файле *Web.config* для обработки ошибок в асинхронных postback-отправках
`AsyncPostBackErrorMessage`	Сообщение об ошибке, которое отправляется клиенту при возникновении неперехватываемого серверного исключения во время асинхронного postback-обмена данными
`AsyncPostBackSourceElementID`	Уникальный идентификатор элемента управления, который вызывал асинхронный postback-обмен данными
`AsyncPostBackTimeout`	Указывает таймаут асинхронного postback-обмена данными, если ответ не получен
`AuthenticationService`	Возвращает объект `AuthenticationServiceManager`, который связан с текущим экземпляром `ScriptManager`
`EnablePageMethods`	Булево значение, указывающее на то, можно ли вызывать статические методы страницы из клиентского сценария на странице ASP.NET
`EnablePartialRendering`	Булево значение, которое разрешает частичную визуализацию страницы
`EnableScriptGlobalization`	Булево значение, указывающее, будет ли элемент управления `ScriptManager` визуализировать в браузере сценарий для поддержки преобразования и форматирования информации в национальном формате
`EnableScriptLocalization`	Булево значение, указывающее на то, будет ли элемент управления `ScriptManager` загружать локализованные версии файлов сценариев
`IsDebuggingEnabled`	Булево значение, указывающее на то, будут ли визуализироваться отладочные версии библиотек клиентских сценариев
`IsInAsyncPostBack`	Булево значение, указывающее на то, будет ли текущий postback-обмен данными выполняться в режиме частичной визуализации страницы

Окончание табл. 5.2

Имя свойства	Функция
LoadScriptsBeforeUI	Булево значение, указывающее на то, когда будут загружаться сценарии — перед загрузкой разметки для пользовательского интерфейса страницы или после нее
ScriptMode	Определяет то, какие версии библиотек клиентских сценариев будут отображаться — отладочные или окончательные
ScriptPath	Путь к каталогу, который используется для построения путей к расширениям ASP.NET AJAX, а также к другим файлам сценариев
Scripts	Возвращает объект ScriptReferenceCollection, который содержит объекты ScriptReference, декларативно или программно зарегистрированные с элементом управления ScriptManager
Services	Возвращает объект ServiceReferenceCollection, содержащий объект ServiceReference для каждой веб-службы, которую расширения ASP.NET AJAX предоставляют клиентам
SupportsPartialRendering	Булево значение, которое указывает на то, поддерживает ли клиент частичную визуализацию страницы

сто выполнения полного postback-обмена данными приложение имитирует его, используя XMLHttpRequest-объект (что и ожидается от AJAX-приложения).

На серверной стороне страница обрабатывается обычным способом, отвечая на запросы любых элементов управления, использующих вызов _doPostBack(). Существующие серверные postback-события продолжают генерироваться, а обработчики событий продолжают работать, как обычно. Все сконструировано так, чтобы AJAX-приложения как можно меньше изменяли существующие ASP.NET-приложения.

Мощь элемента управления ScriptManager проявляется во время визуализации, если частичная визуализация включена. С помощью ScriptManager приложение определяет, какая часть страницы изменилась. Элемент управления UpdatePanel, который подробнее рассматривается далее в этой главе, определяет обновляемые области на странице. Например, если существует страница, содержащая несколько чат-комнат, и нужно обновить только одну из них, следует поместить эту область страницы в элемент управления UpdatePanel.

Элемент управления ScriptManager переопределяет визуализацию страницы и отправляет HTML-код XMLHttpRequest-объекту для каждого элемента UpdatePanel (эти элементы обсуждаются позже) на странице.

Указание дополнительных компонентов сценариев с помощью тега ScriptReference

Элемент управления ScriptReference имеет дочерний тег <Scripts>, в котором указываются дополнительные сценарии для загрузки в браузер. Этот тег может содержать один или несколько тегов <asp:ScriptReference>, определяющих пути к сценариям. После регистрации файла сценария с помощью этого объекта появляется возможность вызывать на странице его методы. Объект ScriptReference способен использовать сценарии, которые хранятся либо как ресурсы, встроенные в сборку, либо как файлы на веб-сервере.

Для регистрации встроенного сценария сначала необходимо записать в свойство Name тега ScriptReference имя физического файла, в котором хранится сценарий, а затем установить свойство Assembly равным имени сборки, содержащей этот сценарий; см. пример ниже.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
   <Scripts>
      <asp:ScriptReference Assembly="MyAssembly"
         Name="MyAssembly.MyScript.js" />
   </Scripts>
</asp:ScriptManager>
```

Если вызвать в браузере страницу, содержащую предыдущий фрагмент кода, и посмотреть на ее исходный код, то можно заметить, что добавился новый блок кода.

```
<script src="MyScript.js" type="text/javascript"></script>
```

В табл. 5.3 перечислены свойства объекта ScriptReference.

Определение служб

В главе 2 было показано, как службу можно непосредственно задействовать в клиентском приложении, используя JavaScript-прокси. Для указания ссылки на службу можно использовать тег <Services> элемента управления ScriptManager. Этот тег может содержать один или несколько тегов <asp:ServiceReference>, определяющих службу, к которой нужно подключиться.

Этот тег также имеет два атрибута.

- Path: путь к службе. В главе 2 было вкратце сказано о том, что, добавляя в конец URI строку /js, можно автоматически генерировать JavaScript-

Таблица 5.3. Свойства тега ScriptReference

Имя свойства	Функция
Assembly	Фактическое имя сборки, которая содержит файл клиентского сценария как встроенный ресурс
IgnoreScriptPath	Указывает на то, включается ли свойство ScriptPath в URL при регистрации файла сценария из ресурса
Name	Имя встроенного ресурса, который содержит файл клиентского сценария
NotifyScriptLoaded	Указывает на то, следует ли добавлять в файл сценария дополнительный код для уведомления метода ScriptLoaded класса Sys.Application
Path	Путь, по которому элемент ScriptManager может найти автономный файл сценария для загрузки
ResourceUICultures	Разделенный запятыми список культур, которые поддерживаются свойством Path
ScriptMode	Режим целевого сценария (debug (отладка), release (окончательная версия) и т.п.)

прокси к веб-службам на сайтах ASP.NET AJAX. Например, веб-служба на странице *wstest.asmx* возвратила бы JavaScript-прокси <asp:ServiceReference>, который можно было бы вызывать по адресу *wstest.asmx/js*. Использование этого тега для указания службы позволяет с помощью ScriptManager автоматически выполнить бóльшую часть работы на клиентской стороне. Ниже приведен соответствующий фрагмент кода.

```
<Services>
   <asp:ServiceReference Path="wstest.asmx"/>
</Services>
```

- InlineScript: булево значение (true или false), которое определяет способ подключения сценария для генерирования прокси — внутренний блок сценария на самой странице или сценарий, полученный отдельным запросом. Значение по умолчанию false. Если вызвать в браузере страницу, в которой это свойство установлено в true и используется тег <Services> элемента управления ScriptManager, то в код на клиентской стороне будет добавлена следующая строка.

```
<script src="wstest.asmx/js"
   type="text/javascript"></script>
```

Обработка ошибок в элементе управления ScriptManager

Элемент управления ScriptManager предоставляет механизм обработки ошибок, в котором можно задать сообщение об ошибке или реализовать более развитую логику в случае возникновения ошибки. Это особенно полезно для клиентской части приложения, поскольку помогает пользователям изящно справляться с ошибками, которые возникают в содержимом ScriptManager.

Двумя самыми простыми способами реализовать обработку ошибок в ScriptManager являются использование события AsyncPostBackError и установка свойства AsyncPostBackErrorMessage для тега ScriptManager. Ниже приведен пример использования свойства AsyncPostBackErrorMessage.

```
<asp:ScriptManager ID="ScriptManager1" runat="server"
      AsyncPostBackErrorMessage=
"В теге ScriptManger возникла ошибка." />
```

Однако для более развитой обработки ошибок придется использовать обработчик события AsyncPostBackError. Например, можно перехватить сообщение об исключительной ситуации и динамически записать его в свойство AsyncPostBackErrorMessage, а затем выполнить другие необходимые действия для обработки ошибки.

```
protected void ScriptManager1_AsyncPostBackError(object
sender, AsyncPostBackErrorEventArgs e)
{
   ScriptManager1.AsyncPostBackErrorMessage =
         e.Exception.Message;
   //Реализуем дальнейшую логику обработки ошибок
}
```

На этом описание элемента управления ScriptManager можно закончить. В следующей главе элемент управления ScriptManager будет снова использоваться в нескольких примерах. В оставшейся части этой главы рассматриваются другие серверные элементы управления, предоставляемые платформой ASP.NET AJAX.

Введение в элемент управления ScriptManagerProxy

Элемент управления ScriptManagerProxy доступен в виде дополнительного менеджера сценариев для страницы. Он также позволяет задействовать пользовательские службы аутентификации (с помощью своего свойства Authenti-

cationService) и службы профилей (с помощью свойства ProfileServiceManager). На ASP.NET-странице допускается использование только одного элемента ScriptManager. Поэтому если используются главная страница и страница содержимого, поместить дополнительные элементы ScriptManager на страницы содержимого невозможно. Элемент управления ScriptManagerProxy позволяет размещать сценарии и/или службы на страницах содержимого. Прежде чем изучать этот элемент управления более подробно, рассмотрим кратко его свойства для поддерживаемых дочерних тегов, которые перечислены в табл. 5.4.

Таблица 5.4. Дочерние теги элемента управления ScriptManagerProxy

Имя свойства	Функция
AuthenticationService	Возвращает объект AuthenticationServiceManager (для пользовательской службы аутентификации), который связан с текущим экземпляром ScriptManagerProxy
ProfileService	Возвращает объект ProfileServiceManager, который связан с текущим экземпляром ScriptManagerProxy
Scripts	Возвращает объект ScriptReferenceCollection, содержащий объект ScriptReference для каждого файла сценария, зарегистрированного с ScriptManagerProxy
Services	Возвращает объект ServiceReferenceCollection, содержащий объект ScriptReference для каждой службы, зарегистрированной с ScriptManagerProxy

Как уже было сказано, элемент управления ScriptManagerProxy идеально подходит для использования на страницах содержимого, если ScriptManager уже был определен на соответствующей главной странице. Чтобы лучше проиллюстрировать этот момент, рассмотрим такую главную страницу, *MasterPage.aspx*.

```
<%@ Master Language="C#" AutoEventWireup="true"
   CodeBehind="MasterPage.master.cs" Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Пример главной страницы</title>
</head>
<body>
   <form id="form1" runat="server">
```

```
   <div>
      <asp:ScriptManager ID="ScriptManager1" runat="server" />
      Это главная страница <br />
      Она содержит элемент ScriptManager: <br />
      <br />
      <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
            runat="server">
      <br />
      </asp:ContentPlaceHolder>
   </div>
   </form>
</body>
</html>
```

На ее основе создадим новую страницу содержимого, которая называется *ContentPage.aspx* и содержит следующий код.

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.Master"
      AutoEventWireup="true" CodeBehind="ContentPage.aspx.cs"
      Inherits="ContentPage" Title="Пример страницы" %>
<asp:Content ID="Content1" ContentPlaceHolderID=
"ContentPlaceHolder1" runat="server">
</asp:Content>
```

Теперь можно вызвать страницу *ContentPage.aspx* в браузере и посмотреть ее исходный код. Как и ожидается, общий код будет такой же, как сгенерированный элементом `ScriptManager` из главной страницы. Этот код состоит из трех блоков главных сценариев (среди прочего), указывающих на *WebResource.axd* и *ScriptResource.axd*, как показано ниже.

```
<script
src="/Ajax/WebResource.axd?d=HQhspev9RtnoVp5Ca4MubA2&amp;
t=633008366579531250" type="text/javascript"></script>
<script src="/Ajax/ScriptResource.axd?d=
rbfRw_fjV44N4zFu5uugvXCg0fpE5bOdbRFvvkMhZEO1
-ghFYTQ7i9aLWWp9hO2901tgv-pDZFxuTtMikT21d-q8lo-xXLBcAYv3xq0hiRM1
&amp;t=633051881703906250" type="text/javascript">
</script>
<script src="/Ajax/ScriptResource.axd?d=
rbfRw_fjV44N4zFu5uugvXCg0fpE5bOdbRFvvkMhZEO1
-ghFYTQ7i9aLWWp9hO2901tgv-pDZFxuTtMikT21d3JhQBwnJ44PsSIlvSkVAgc1
&amp;t=633051881703906250" type="text/javascript"></script>
```

Но предположим, что на странице содержимого должна быть дополнительная AJAX-функциональность. Например, требуется задействовать один из многих великолепных элементов управления, которые доступны в инструментарии ASP.NET AJAX Control Toolkit (этот инструментарий подробно описывается в главах 7 и 8).

Эти элементы управления требуют дополнительных сценариев, которые не включены в главную страницу. Такая ситуация является идеальной для применения элемента управления ScriptManagerProxy.

Не рассматривая подробно инструментарий ASP.NET AJAX Control Toolkit, с помощью ScriptManagerProxy добавим один из входящих в инструментарий элементов, DragPanelExtender, на страницу содержимого. Для этого перетащим ScriptManagerProxy, который сопровождается элементом управления Label (надпись), и элемент DragPanelExtender (из AJAX Control Toolkit) на страницу. Установим свойство text надписи, например, так: "Эту надпись можно перемещать по странице". С этого момента страница должна выглядеть примерно так, как показано на рис. 5.3, и содержать следующий код.

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.Master"
      AutoEventWireup="true" CodeBehind="ContentPage.aspx.cs"
      Inherits="Ajax.ContentPage"
      Title="Демонстрация DragPanelExtender" %>

<%@ Register Assembly="AjaxControlToolkit"
Namespace="AjaxControlToolkit" TagPrefix="cc1" %>
<asp:Content ID="Content1"
ContentPlaceHolderID="ContentPlaceHolder1"
runat="server">
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
</asp:ScriptManagerProxy>
<cc1:DragPanelExtender ID="DragPanelExtender1" runat="server"
EnableViewState="False" TargetControlID="Label1">
</cc1:DragPanelExtender>
 
<asp:Label ID="Label1" runat="server" Text="Эту надпись можно
перемещать по странице">
</asp:Label>
</asp:Content>
```

Последнее, что нужно сделать перед запуском страницы, это установить свойство TargetControlID объекта DragPanelExtender равным имени элемента Label (в данном случае Label1). Если вызвать эту страницу в браузере, то можно будет перемещать надпись по странице, как показано на рис. 5.4.

Исходный код этой страницы в браузере содержит еще шесть дополнительных блоков сценариев.

```
<script src="/Ajax3/ScriptResource.axd?d=
IV7jia2nXbc7sCg1SWf3RbWQWNeQtdO8PGyfXw5p
BCt7QucJL9oE4uI487xHlPYvLbUfoMzAx0Dl7veKacOLUw2&amp;
t=633083898300000000"
type="text/javascript">
</script>
```

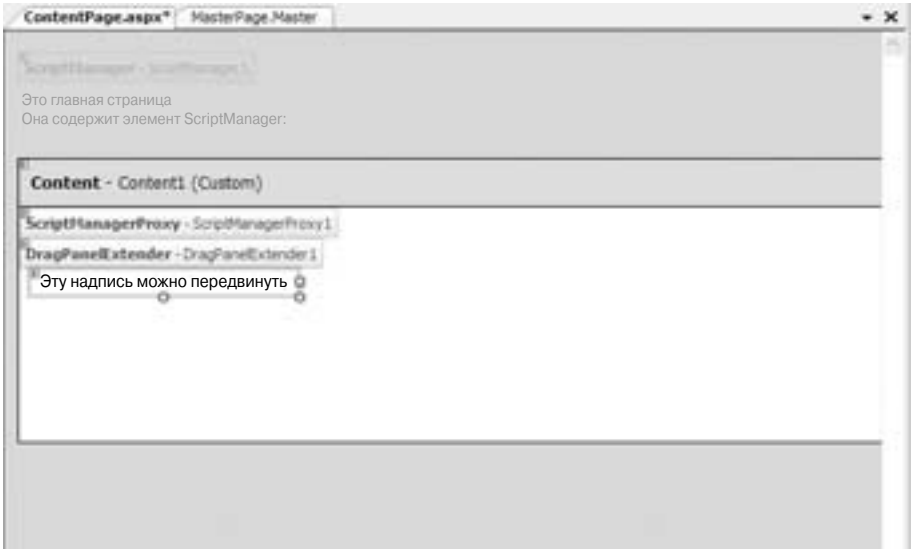

Рис. 5.3. Добавление элемента управления `ScriptManagerProxy` на страницу содержимого

```
<script src="/Ajax3/ScriptResource.axd?d=
IV7jia2nXbc7sCg1SWf3RbWQWNeQtdO8PGyfXw5pBC
t6I1vaIgM5kgWHpAx-XLCYADQCoaENHDXR_fmnXYiB5Q2&amp;
t=633083898300000000"
type="text/javascript">
</script>
<script src="/Ajax3/ScriptResource.axd?d=
IV7jia2nXbc7sCg1SWf3RbWQWNeQtdO8PGyfXw5p
BCuGnLpMX4aibann483UFkP4UcDhmgCv77Gz2BPJzb0sGQ2&amp;
t=633083898300000000"
type="text/javascript">
</script>
<script src="/Ajax3/ScriptResource.axd?d=
IV7jia2nXbc7sCg1SWf3RbWQWNeQtdO8PGyfXw5p
BCtjSYyc7zoec8BYAEgCq7Xfw8Q1uMQbSmJ5pFsFPdNdi53i7U-TPJGeX-
iRo2uSjUM1&amp;t=633083898300000000" type="text/javascript">
</script>
<script src="/Ajax3/ScriptResource.axd?d=
IV7jia2nXbc7sCg1SWf3RbWQWNeQtdO8PGyfXw5p
BCsJ5ep-dAHOns9-VxfadeqV_
ZfemVlTAoDxNenJwjPNYSz3EWAPxxWj3tXQmN4a7DI1&amp;
t=633083898300000000" type="text/javascript">
</script>
<script src="/Ajax3/ScriptResource.axd?d=
```

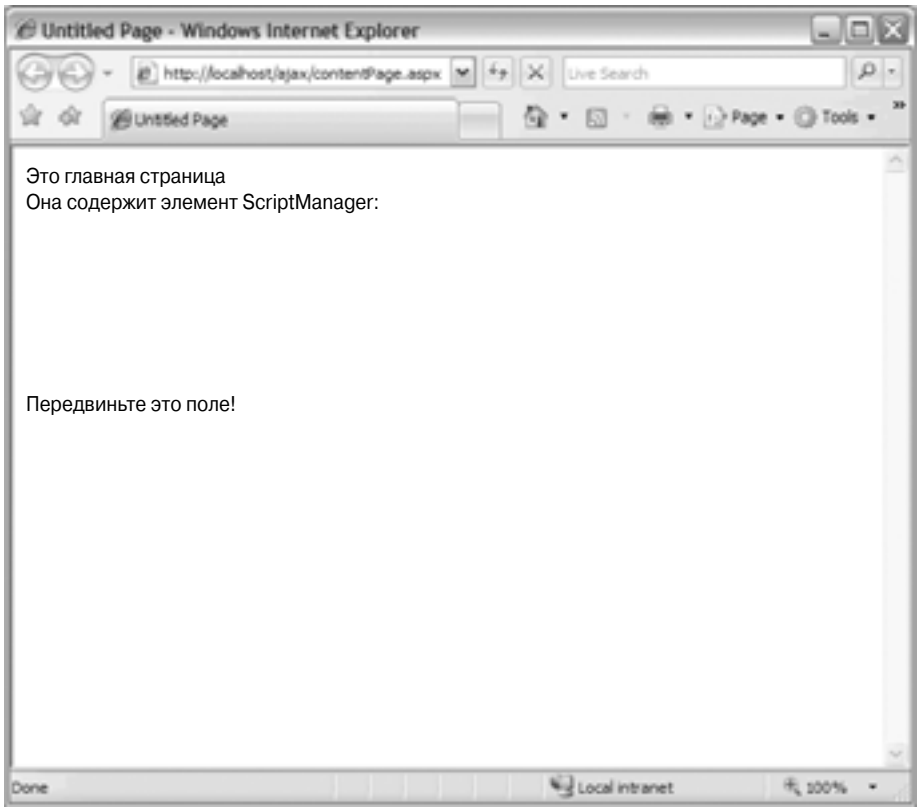

Рис. 5.4. Размещение элемента `UpdatePanel` на веб-форме

```
IV7jia2nXbc7sCg1SWf3RbWQWNeQtdO8PGyfXw5p
BCtKZKNEWlh60o9nJAvWs0ew_AfDKm3BP43z3sXqwMBrtQT-
xZwKhUv0ddRO4WY6Is41&amp;t=633083898300000000"
type="text/javascript">
</script>
```

Эти добавочные блоки сценариев содержат дополнительную логику, необходимую для работы элементов, входящих в состав AJAX Control Toolkit, которые были динамически вставлены на страницу с помощью `ScriptManagerProxy`. Без элемента `ScriptManagerProxy` невозможно было бы автоматически обработать необходимые сценарии, поскольку все это происходит на странице содержимого.

Введение в элемент управления UpdatePanel

В типичных приложениях ASP.NET 2.0 при выполнении обратной отправки страницы на сервер (postback) вся страница визуализируется заново. Это приводит к "мельканию" в окне браузера или другой клиентской программы. Сервер обнаруживает postback-обмен данными и запускает новый жизненный цикл страницы. В конце концов выполняется код заданного обработчика события для элемента, который вызвал postback-событие, и он вызывается после обработчика события страницы.

Использование элементов UpdatePanel и ScriptManager позволяет устранить необходимость в полном обновлении страницы. Элемент UpdatePanel похож на элемент ContentPanel тем, что он отмечает на веб-странице область, которая будет автоматически обновляться при возникновении postback-события (но без упомянутого выше postback-поведения в клиенте). Обмен данными в таком случае происходит через XMLHttpRequest-канал — т.е. действительно в стиле AJAX. Страница на сервере все равно, как обычно, обрабатывает postback-событие, вызывает обработчики событий и так далее, но окончательная визуализация страницы означает, что создаются только заданные в UpdatePanel области. Кроме того, в отличие от ScriptManager на одной странице можно иметь несколько элементов UpdatePanel и даже вкладывать элементы UpdatePanel друг в друга.

Использование элемента управления UpdatePanel

Чтобы использовать элемент управления UpdatePanel, необходимо просто перетащить его на веб-форму в окне конструктора (рис. 5.5).

Однако UpdatePanel не может функционировать без элемента ScriptManager на странице. Кроме того, элемент ScriptManager должен быть размещен на странице до элемента UpdatePanel. Иначе говоря, если просматривать исходный код страницы сверху вниз, ссылка на ScriptManager должна появиться раньше ссылки на UpdatePanel. Использование "помощника" (Tasks Assistant) гарантирует, что элементы размещаются правильно. Если элемент ScriptManager отсутствует или размещен на странице неправильно, то при попытке открыть страницу в браузере возникает ошибка (рис. 5.6).

В элементе управления UpdatePanel имеется область, в которую можно поместить HTML-код. После postback-события будет обновляться только этот код, если ScriptManager настроен на частичное обновление страницы. На рис. 5.7 показана страница с несколькими текстовыми полями и кнопкой. В данном случае два текстовых поля, две надписи и кнопка расположены *за пределами* элемента UpdatePanel, еще одна надпись находится *внутри* UpdatePanel. Последняя надпись называется lblResult. Код кнопки выглядит следующим образом.

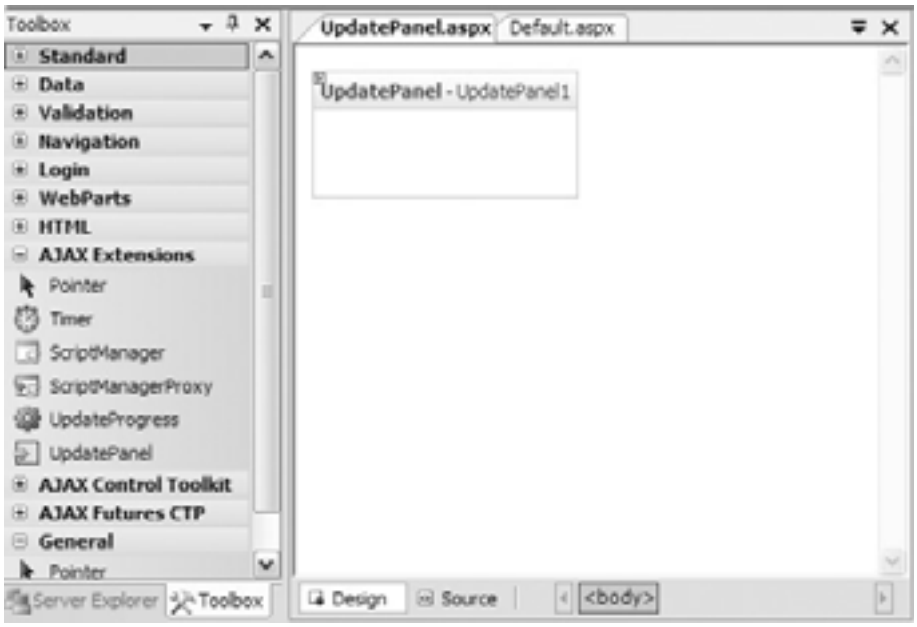

Рис. 5.5. Размещение элемента управления UpdatePanel на веб-форме

```
int x = Convert.ToInt16(txt1.Text);
int y = Convert.ToInt16(txt2.Text);
int z = x+y;
lblResult.Text = z.ToString();
```

Очевидно, что при обновлении в последнюю надпись записывается сумма значений текстовых полей. Так как надпись lblResult расположена в пределах UpdatePanel, а ScriptManager настроен на частичную визуализацию страницы, при щелчке на кнопке будет обновляться только текст внутри элемента UpdatePanel. В главе 6 рассматриваются другие подобные примеры.

Программирование с UpdatePanel

Разметка для элемента UpdatePanel из предыдущего примера выглядит так.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
   <ContentTemplate>
   <asp:Label ID="lblResult" runat="server"
         Text="Label"></asp:Label>
   </ContentTemplate>
</asp:UpdatePanel>
```

Тег <asp:UpdatePanel> поддерживает два дочерних тега: <ContentTemplate> и <Triggers>. Прежде чем обсуждать эти теги, рассмотрим свойства

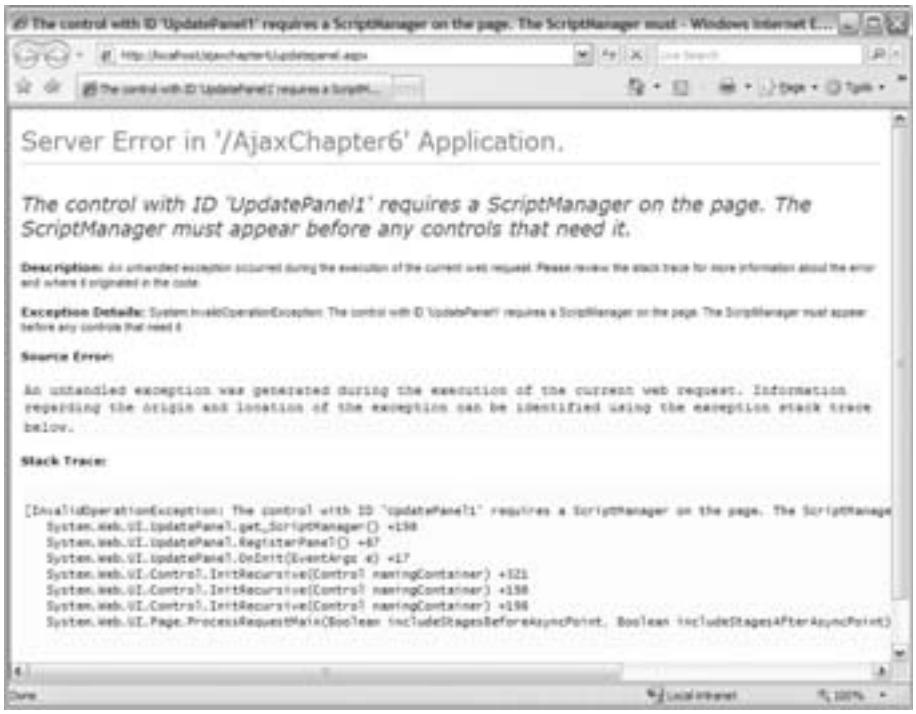

Рис. 5.6. Страница с сообщением об ошибке при неправильной конфигурации `UpdatePanel` и `ScriptManager`

элемента `UpdatePanel` за исключением тех, которые наследуются от класса `Control` (табл. 5.5).

Использование тега `ContentTemplate`

Тег `<ContentTemplate>` определяет HTML- или ASP.NET-элемент, который будет обновляться элементом управления `UpdatePanel`. Для создания этого HTML-кода можно использовать конструктор Visual Studio 2005. Например, если перетащить элемент управления `Calendar` в область шаблона `UpdatePanel` (рис. 5.8), он будет определен внутри тега `<ContentTemplate>`.

Разметка, созданная путем добавления календаря в конструкторе, выглядит так.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
   <ContentTemplate>
   <asp:Label ID="lblResult" runat="server"
         Text="Label"></asp:Label>
   <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
   </ContentTemplate>
</asp:UpdatePanel>
```

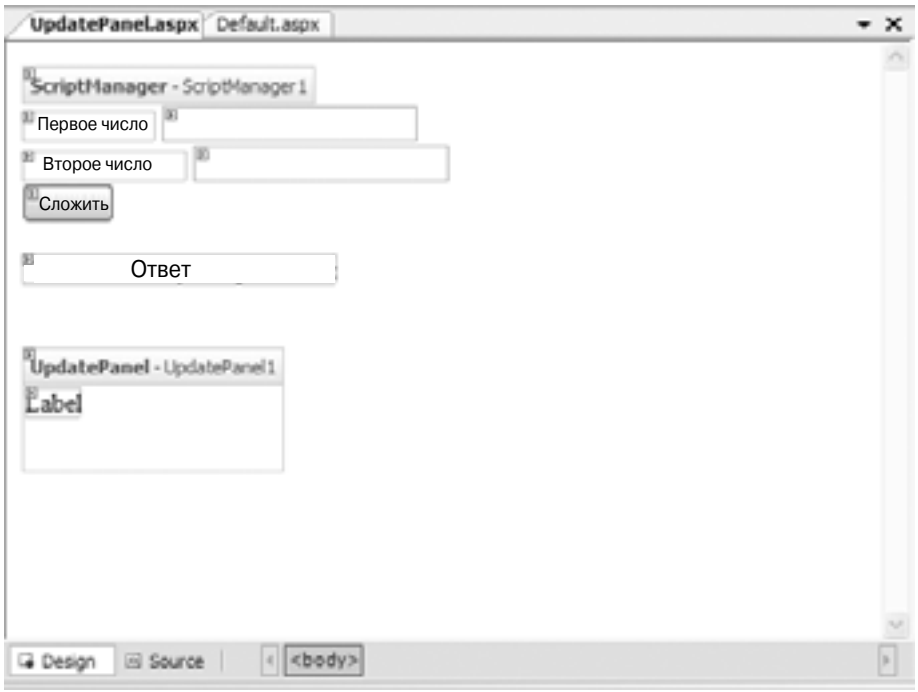

Рис. 5.7. Простое приложение, в котором используется элемент управления `UpdatePanel`

Использование триггеров

Еще одним дочерним тегом для `UpdatePanel` является тег `<Triggers>`. Он позволяет определять триггеры для обновления. Как показано в предыдущей таблице, элемент `UpdatePanel` имеет свойство `calledUpdateMode`. Если установить его значение равным `Conditional` (по умолчанию `Always`), то обновление разметки будет происходить только при срабатывании триггера. Тег `Triggers` содержит семейство определений триггеров. В Visual Studio 2005 существует редактор семейства триггеров (Trigger Collections Editor), который можно использовать для просмотра и редактирования триггеров внутри `UpdatePanel`, как показано на рис. 5.9. Редактор можно вызвать, выбрав свойство Trigger Collections в окне свойств элемента `UpdatePanel`.

Внутри тега `<Triggers>` поддерживаются два типа триггеров: `AsyncPostBackTrigger` и `PostBackTrigger`. Эти триггеры можно использовать для элементов управления, которые находятся за пределами `UpdatePanel`. Эти два тега отличаются лишь тем, что первый из них, как следует из его названия, может обрабатывать асинхронные postback-события при срабатывании триггера. Он также имеет дополнительное свойство `EventName`, которое позволяет указывать

Таблица 5.5. Свойства элемента управления `UpdatePanel`

Имя свойства	Функция
`ChildrenAsTriggers`	Булево значение, которое указывает на то, будут ли postback-события из непосредственно дочерних элементов `UpdatePanel` обновлять содержимое панели
`ContentTemplateContainer`	Возвращает объект элемента управления, который впоследствии можно использовать для добавления к нему дочерних элементов управления
`Controls`	Возвращает объект `ControlCollection`, который содержит дочерние элементы управления для `UpdatePanel`
`IsInPartialRendering`	Указывает на то, будет ли `UpdatePanel` обновляться в результате асинхронных postback-событий
`RenderMode`	Указывает на то, будет ли содержимое `UpdatePanel` заключено в HTML-дескриптор `<div>` или `<span>`
`RequiresUpdate`	Указывает на то, будет ли обновляться содержимое элемента управления `UpdatePanel`
`UpdateMode`	Определяет периодичность обновления содержимого `UpdatePanel`. По умолчанию “always” (всегда)

имя события целевого элемента управления, отвечающего за инициирование обновления.

Триггер `AsyncPostBackTrigger` следует определять вместе со связанным элементом управления (задается с помощью `ControlID`) и именем события (которое указывается в свойстве `EventName`). При возникновении этого события срабатывает триггер, и элемент управления `UpdatePanel` обновляется. Триггер `PostBackTrigger` указывается с помощью тега `<asp:PostBackTrigger>`, а триггер `AsyncPostBackTrigger` — с помощью тега `<asp:AsyncPostBackTrigger>`. Продолжим предыдущий пример, модифицировав его.

```
<asp:Button ID="btnAdd" runat="server" Text="Add"
OnClick="btnAdd_Click" />
<br />
   <asp:UpdatePanel ID="UpdatePanel1" runat="server"
   UpdateMode=Conditional >
   <ContentTemplate>
   <asp:Label ID="lblResult" runat="server"
   Text="Label"></asp:Label>
```

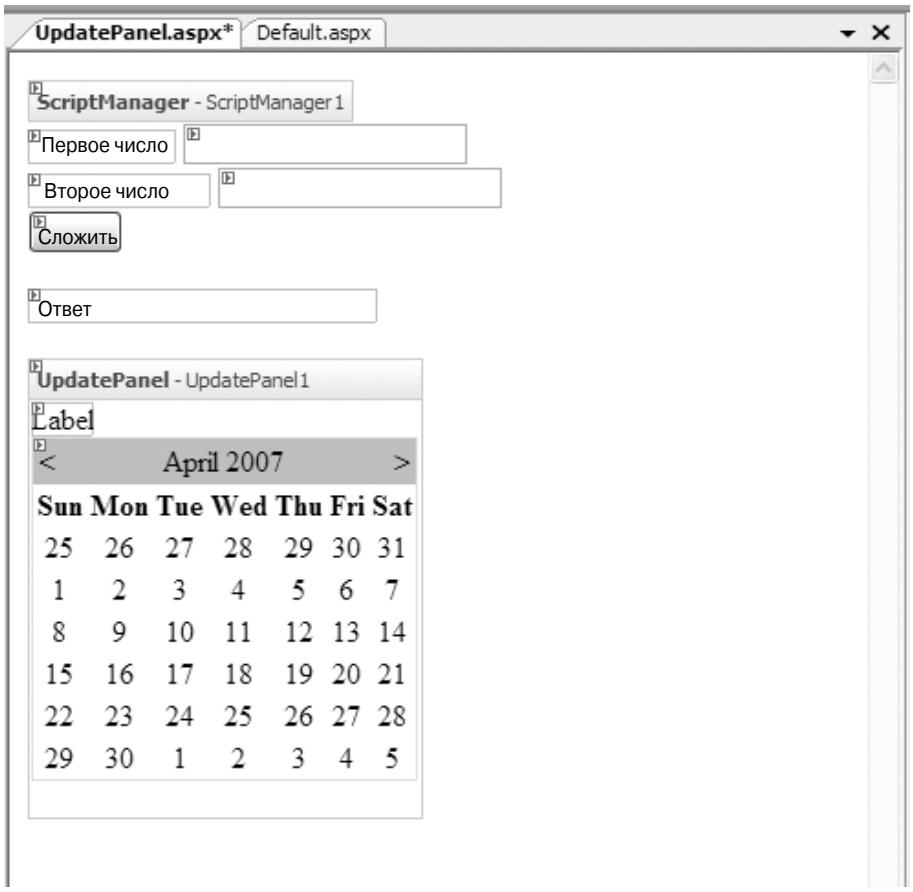

Рис. 5.8. Добавление элементов управления в шаблон содержимого UpdatePanel

```
<asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
</ContentTemplate>
<Triggers>
<asp:AsyncPostBackTrigger ControlID="btnAdd"
EventName="Click" />
</Triggers>
</asp:UpdatePanel>
```

Здесь AsyncPostBackTrigger указывает, что ключом для триггера является кнопка с именем btnAdd, событие Click которой и запускает триггер. Таким образом, при щелчке на кнопке срабатывает триггер AsyncPostBackTrigger и выполняется частичное обновление. Обратите внимание на то, что объявление кнопки btnAdd находится вне блока UpdatePanel.

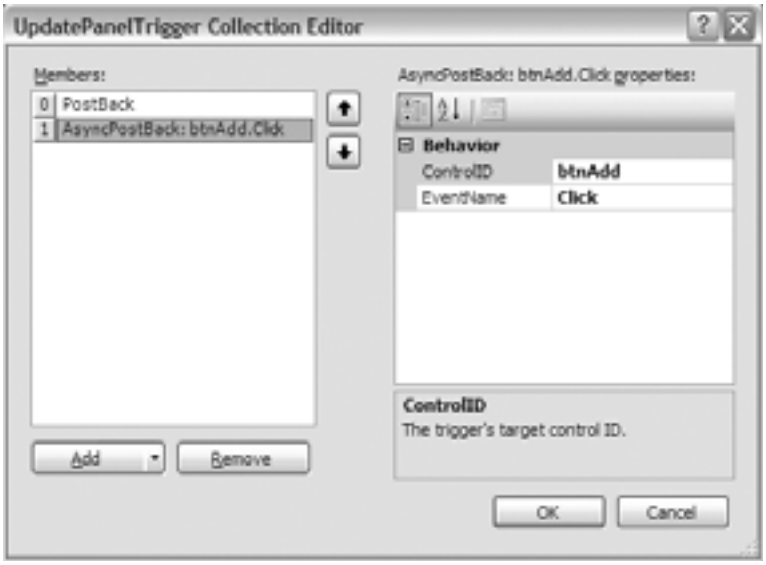

Рис. 5.9. Редактор семейства UpdatePanelTrigger в Visual Studio 2005

Введение в элемент управления `UpdateProgress`

Еще одним серверным элементом управления в ASP.NET AJAX выступает `UpdateProgress`. Он является индикатором выполнения текущей асинхронной операции. Обычно в качестве индикатора активности служит строка состояния браузера. При частичном обновлении и модели асинхронных postback-событий AJAX строка состояния браузера в этом качестве не применима. Поэтому элемент управления `UpdateProgress` идеально подходит для этой роли, а кроме того, он является более дружественным к пользователю.

Использование элемента `UpdateProgress`

Можно использовать на странице один `UpdateProgress` для нескольких элементов `UpdatePanel` или применять множество `UpdateProgress` с различными UI для разных `UpdatePanel`, если для каждого раздела страницы нужен свой индикатор выполнения. По умолчанию, если не установлено свойство `AssociatedUpdatePanelID` элемента `UpdateProgress`, он будет срабатывать, откликаясь на события во всех `UpdatePanel` на странице (если их несколько). Чтобы использовать `UpdateProgress`, необходимо перетащить его на страницу, в результате чего будет создан тег `<asp:UpdateProgress>`:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" />
```

Фактическая разметка, после того как состоится обращение к элементу, определяется тегом <ProgressTemplate>.

Когда приложение осуществляет обращение к серверу, отображается HTML-код, определенный в теге <ProgressTemplate>. Именно здесь можно использовать GIF-изображение или какое-либо специальное сообщение, чтобы проинформировать пользователя о ходе выполнения процесса.

Программирование с использованием UpdateProgress

Прежде чем переходить к примеру использования элемента UpdateProgress, рассмотрим его свойства, перечисленные в табл. 5.6.

Таблица 5.6. Свойства элемента управления UpdatePanel

Имя свойства	Функция
AssociatedUpdatePanelID	ID элемента UpdatePanel, для которого UpdateProgress отображает состояние
DisplayAfter	Время в миллисекундах до отображения элемента UpdateProgress
DynamicLayout	Значение, определяющее, будет ли шаблон индикатора активности визуализироваться динамически. Если значение равно false, он будет занимать необходимое пространство всегда, даже если содержимое индикатора не отображается

В настоящее время во многих крупных приложениях нередко используются длительные операции с данными. В таких случаях для уведомления пользователя о состоянии приложения полезно использовать элемент управления UpdateProgress. Для простоты создадим страницу, имитирующую выполнение длительного процесса путем приостановки потока выполнения на несколько секунд.

Чтобы сделать это, поместим на новую WebForm-страницу элементы управления ScriptManager, UpdatePanel и UpdateProgress. После этого внутри элемента UpdatePanel создадим новую кнопку (Button). В режиме редактирования исходного кода *.aspx*-страницы внутри тега UpdatePanel создадим новый тег <ProgressTemplate> со следующей разметкой.

```
<ProgressTemplate>
   Выполняется расчет...
</ProgressTemplate>
```

Страница должна выглядеть примерно так, как показано на рис. 5.10.

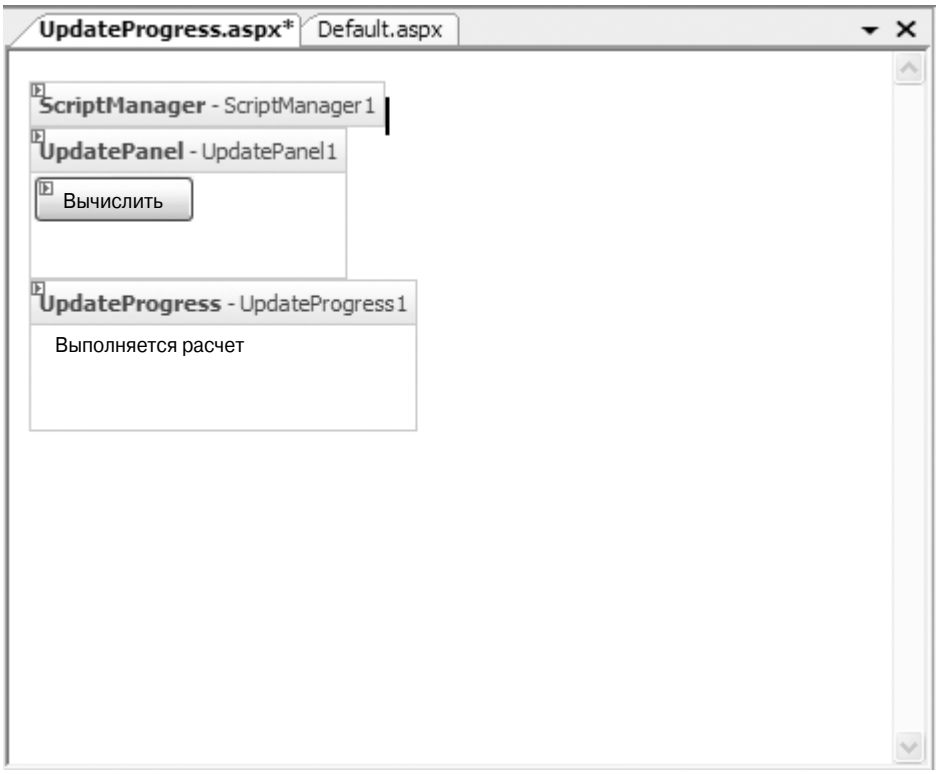

Рис. 5.10. Использование элемента управления `UpdateProgress`

Теперь можно сымитировать выполнение длительного процесса после щелчка на кнопке. Для этого в обработчике события кнопки можно приостановить поток выполнения на четыре секунды.

```
protected void Button1_Click(object sender, EventArgs e)
{
   System.Threading.Thread.Sleep(4000);
}
```

Если открыть страницу в браузере и выбрать кнопку Вычислить, на странице на несколько секунд появится, а затем исчезнет строка “Выполняется расчет” (рис. 5.11).

В исходном коде загруженной в браузер страницы будет новый тег `<div>`, а также дополнительный JavaScript-код, который был создан элементом `UpdateProgress`.

```
<div id="UpdateProgress1" style="display:none;">
   Выполняется расчет...
</div>
```

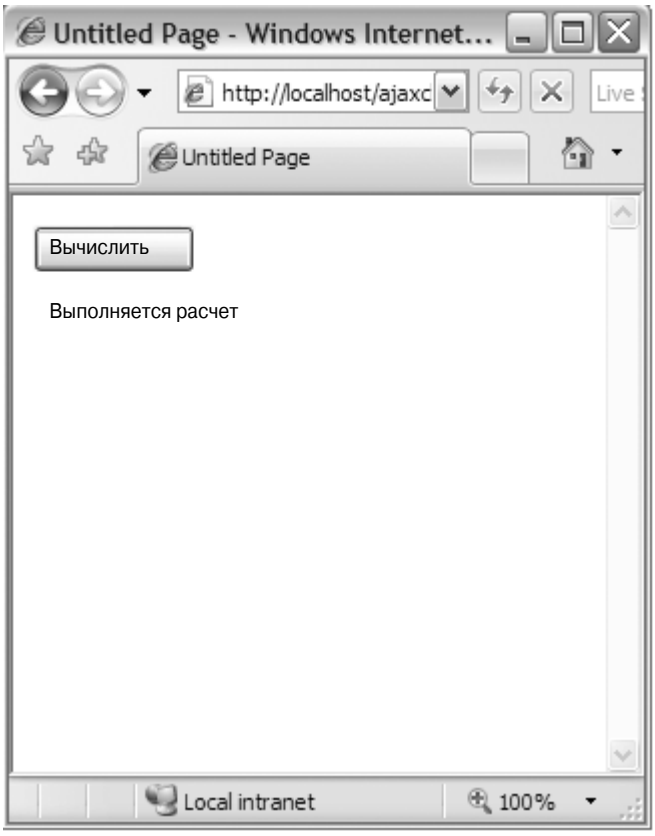

Рис. 5.11. Демонстрация работы элемента `UpdateProgress`

Новые сценарии, которые были динамически введены в страницу, включают свойство `style` для дескриптора `<div>`, хотя никакого дополнительного программирования для этого не потребовалось.

Введение в элемент управления `Timer`

Самым простым серверным элементом управления ASP.NET AJAX, вероятно, является `Timer`. Он во многом похож на одноименный элемент управления, который какое-то время использовался для Winform-приложений. `Timer` предоставляет простую в использовании функциональность таймера, которую можно настроить на многократное выполнение операций в зависимости от истекшего времени. Можно запускать определенные операции с регулярным интервалом синхронно или асинхронно, не обновляя страницу вручную. Эта возможность может оказаться полезной во многих веб-приложениях.

Представьте, например, что страница содержит критически важную динамически обновляемую информацию, такую как биржевые сводки или сведения о прибытии авиалайнеров. Эта информация должна обновляться регулярно. В такой ситуации можно применить на странице элемент `Timer` для инициирования обновлений `UpdatePanel`, которые будут выполняться без полной перезагрузки страницы.

Использование элемента `Timer`

Чтобы использовать `Timer`, на странице, конечно, должен присутствовать элемент `ScriptManager`. Добавить элемент `Timer` можно путем его перетаскивания в окне конструктора. Таймеры применяются для обновления содержимого элемента `UpdatePanel` по истечении определенного времени.

Чтобы увидеть элемент `Timer` в работе, можно добавить на пустую страницу элементы `UpdatePanel` и `ScriptManager`. Затем на страницу следует установить элемент `Timer`. Кроме этого, внутри элемента `UpdatePanel` следует поместить надпись (`Label`). В окне конструктора страница должна выглядеть примерно так, как показано на рис. 5.12.

Рис. 5.12. Использование элемента управления `Timer` в окне конструктора

Наконец следует дважды щелкнуть мышью на элементе `Timer` для того, чтобы Visual Studio 2005 сгенерировала код обработчика события `OnTick` таймера. В разметке страницы после этого появится тег `<asp:Timer>`. Ниже приведен

пример таймера Timer1 с интервалом 4000 мс (4 с) и обработчиком события Timer1_Tick.

```
<asp:Timer ID="Timer1" runat="server" Interval="4000"
      OnTick="Timer1_Tick">
</asp:Timer>
```

Теперь внутри метода Timer1_Tick в базовом классе страницы можно выполнять какую-либо операцию каждый раз при срабатывании таймера, например, обновление показаний часов. Триггер AsyncPostBackTrigger используется внутри UpdatePanel для инициирования обновлений по событию Tick таймера. Как выглядит разметка для этого, показано ниже.

```
<div>
   <asp:ScriptManager ID="ScriptManager1" runat="server">
   </asp:ScriptManager>
</div>
<asp:Timer ID="Timer1" runat="server" Interval="4000"
      OnTick="Timer1_Tick">
</asp:Timer>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
   <Triggers>
      <asp:AsyncPostBackTrigger ControlID="Timer1"
            EventName="Tick" />
   </Triggers>
   <ContentTemplate>
      <asp:Label ID="Label1" runat="server" Text=""></asp:Label>
   </ContentTemplate>
</asp:UpdatePanel>
```

В браузере в надписи Label1 каждые четыре секунды будет появляться новое время. Это будет происходить без полного обновления страницы (рис. 5.13).

Резюме

В этой главе представлено введение в серверные элементы управления, доступные в ASP.NET AJAX. В частности, здесь рассматривался элемент ScriptManager, который является сердцем платформы ASP.NET AJAX. Он берет на себя управление рабочим циклом ASP.NET AJAX, а также управление связанными сценариями. Кроме него, рассматривался элемент UpdatePanel, который позволяет включать AJAX-функциональность в существующих страницах ASP.NET, используя частичные обновления страниц.

В этой главе был представлен общий обзор главных серверных элементов управления ASP.NET AJAX и их работы. В ASP.NET AJAX существует еще одна группа серверных элементов управления, которые называются *расширениями*

Рис. 5.13. Демонстрация элемента управления `Timer`

элементов управления. Они поставляются в составе инструментария ASP.NET AJAX Control Toolkit. В данной главе эти элементы лишь кратко упоминаются, для их подробного рассмотрения отведены главы 7 и 8. В следующей главе вниманию читателя представлены использующие эту функциональность приложения и примеры, анализ которых позволит выяснить возможности написания вами собственных приложений в ASP.NET AJAX.