ГЛАВА 6

Шаблоны серверных элементов управления

Начиная с ASP.NET 2.0, добавление шаблонов и привязка данных к серверным элементам управления требует намного меньше кода, чем предыдущие версии ASP.NET. Примеры этой главы используют преимущества этих усовершенствований.

Шаблоны позволяют настроить отображение данных серверными элементами управления или элементами HTML. Шаблоны и привязка данных обычно идут рука об руку в наиболее развитых элементах управления, таких как серверные элементы ASP.NET `GridView` или `Repeater`, делая Web-разработку, ориентированную на базы данных, быстрой и простой. Извлеките часть данных из базы через ADO.NET, привяжите к серверному элементу управления, сконфигурируйте его стилевые свойства и шаблоны, и Web-разработчики смогут создавать очень привлекательные HTML-изображения, которые выглядят и работают подобно формам Visual Basic с выравниванием данных, выделениями цветом и т.д. Серверный элемент выполняет всю трудную работу, которая обычно требует значительного объема ручного кодирования в простом старом ASP. В следующей главе мы раскроем тему привязки данных на основе шаблонов.

В этой же главе сосредоточим внимание на добавлении поддержки шаблонов к серверным элементам управления. Начнем с примеров с шаблонным элементом управления по имени `TemplateMenu`; этот пример демонстрирует, как строить серверный элемент, который позволит применять шаблоны к гиперссылкам. В следующих примерах элемента будет показано, как трактовать дескрипторы в качестве данных для построения содержимого элемента управления.

Специализированное содержимое элемента управления

HTML — это комбинация содержимого и внешнего вида. Предыдущая глава показала, как серверные элементы управления настраивают внешнее представление содержимого посредством использования атрибутов стиля, модифицирующих такие средства, как шрифт, цвет, размер или даже размещение HTML. В этой главе мы поговорим о том, как модифицировать центральное содержимое Web-формы посредством невероятно удобной техники — шаблонов и привязки данных.

Шаблоны позволяют Web-разработчикам специфицировать HTML-элементы и серверные элементы управления, которые визуализируются как часть основного вывода серверного элемента управления. Серверный элемент получает шаблоны, являющиеся местом размещения содержимого, как показано на рис. 6.1. Шаблоны позволяют на-

страивать то, как визуализируется элемент управления, просто редактируя страницу .aspx. Разработчики серверных элементов управления должны добавлять шаблонную функциональность к специальным элементам управления, чтобы обеспечить этот уровень гибкости, когда имеет смысл пользователям-разработчикам позволять модификацию UI серверного элемента управления.

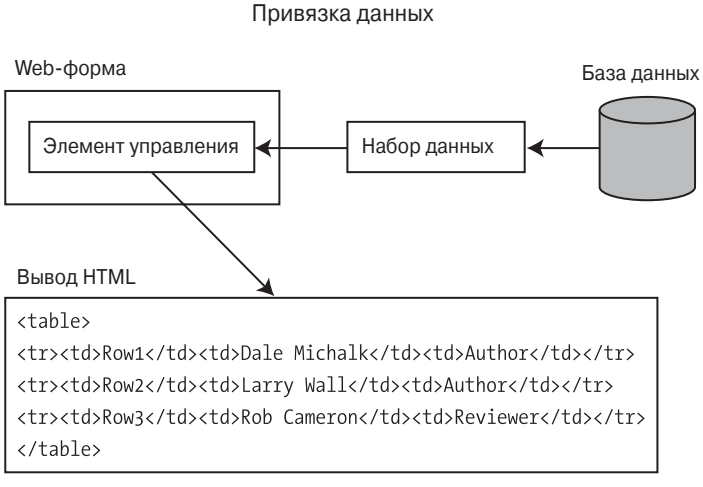

Рис. 6.1. Шаблоны с серверными элементами управления

Использование шаблонов элементов управления

Шаблон ASP.NET — это смесь HTML-элементов и элементов управления ASP.NET, представляющих компоновку определенной области элемента управления. Шаблоны повышают уровень гибкости в элементе управления и позволяют разработчику/пользователю настраивать графическое представление содержимого элемента. Следующий фрагмент кода представляет гипотетический шаблон MyTemplate:

```
<MyTemplate>
  <span>Raw HTML</span>
  <asp:Label id="Label1" runat="server" Text="My Server Control Label" />
</MyTemplate>
```

Шаблон имеет начальный и конечный дескрипторы с некоторым содержимым, состоящим из смеси "сырого" HTML с серверными элементами управления. Обычно вы специфицируете содержимое декларативно, но можете также и загрузить его из файла или создать его экземпляр из предварительно построенного шаблонного класса во время выполнения для динамического использования шаблона, как мы покажем в главе 7, когда будет тестировать нашу версию элемента управления Repeater.

В некоторых отношениях поддержка шаблонов ASP.NET частично избавляет разработчика элементов управления от забот об их внешности и поведении. Разработчик элемента управления может сосредоточиться на внутреннем его устройстве, в то время как пользователь элемента управления или графический дизайнер, работающий совместно с пользователем элемента, могут строить симпатичные шаблоны, которые располагаются внутри элемента управления. Конечно, разработчик элемента управления может предоставлять специальные дизайнеры, которые могут помочь в создании шаблонов, хотя это и не обязательно (функциональность времени проектирования мы обсудим в главе 11).

Атрибут ParseChildren

Элемент управления, который обеспечивает поддержку шаблонов, должен указывать анализатору страниц ASP.NET, что он желает управлять дочерним содержимым, добавляя атрибут ParseChildren к объявлению его класса.

Этот атрибут инструктирует анализатор страниц о необходимости обрабатывать дочерние элементы как свойства элементов управления. Для элементов, унаследованных от System.Web.UI.WebControls.WebControl или производных классов, эта функциональность достается бесплатно через наследование. Вы должны добавить атрибут вручную к элементам, основанным на System.Web.UI.Control.

Атрибут ParseChildren имеет свойство по имени ChildrenAsProperties, которое конфигурирует поведение анализатора. WebControl устанавливает свойство ChildrenAsProperies атрибута ParseChildren в true по умолчанию, как показано в строке кода ниже. Если вы хотите использовать другое значение для серверного элемента управления на основе WebControl, не забудьте установить атрибут явно, чтобы переопределить поведение по умолчанию.

```
ParseChildrenAttribute(ChildrenAsProperties = true)
```

ParseChildrenAttribute также имеет сокращенную версию для установки свойства ChildrenAsProperties:

```
ParseChildren(true)
```

Присутствие атрибута ParseChildren, установленного в true для ChildrenAsProperties, заставляет анализатор страниц ASP.NET отображать элементы XML верхнего уровня или дескрипторы под серверным элементом управления непосредственно на его свойства. Серверный элемент управления отвечает за обеспечение соответствующего отображения, как показано на рис. 6.2.

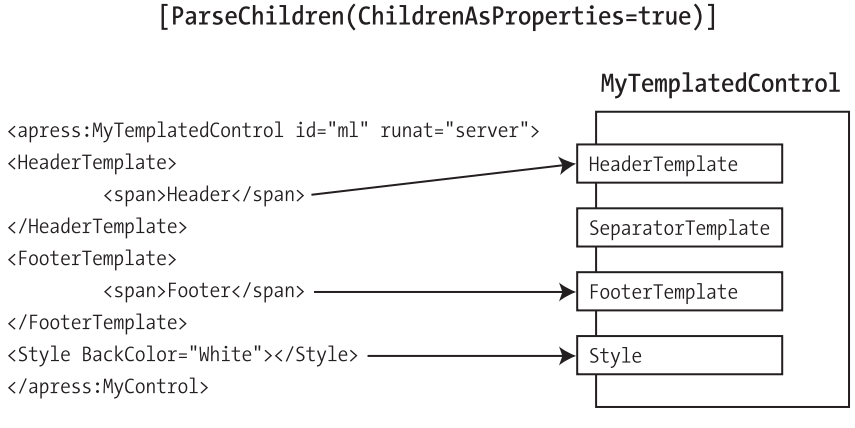

Рис. 6.2. Атрибут ParseChildren с ChildrenAsProperties=true

Если значение свойства ChildrenAsProperties устанавливается в false вместо true, ASP.NET пытается обработать дочернее содержимое внешних дескрипторов элемента управления как серверные элементы, что показано на рис. 6.3. Реализация по умолчанию интерфейса IParserAccessor и его единственного метода AddParsedSubObject добавляет эти дочерние серверные элементы управления к коллекции Controls родительского элемента управления. Литеральный текст между дескрипторами становится экземпляром серверного элемента управления LiteralControl.

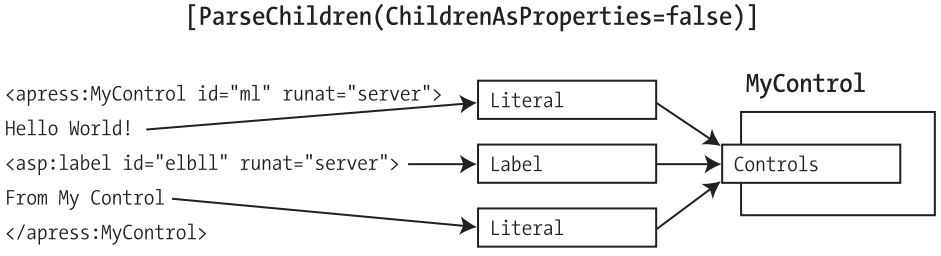

Рис. 6.3. Атрибут `ParseChildren` с `ChildrenAsProperties=false`

Элемент управления — меню с шаблонами

Элемент управления — меню, реализованный в главе 2, отображал простой список гиперссылок HTML. Мы пересмотрим его дизайн, чтобы проиллюстрировать применение шаблонов UI. Гиперссылки останутся, пока мы не предоставим несколько специальных серверных элементов управления для визуализации вывода.

Мы построим элемент управления `TemplateMenu` как составной элемент с тремя шаблонами: `HeaderTemplate`, `FooterTemplate` и `SeparatorTemplate` (рис. 6.4). Как можно было бы предположить, `HeaderTemplate` и `FooterTemplate` позволяют разработчикам видоизменять верхнюю и нижнюю части элемента — меню. `SeparatorTemplate` обеспечивает видоизменение содержимого между гиперссылками. Для простоты элемент управления использует внутренний источник данных для обеспечения содержимого для визуализации каждой гиперссылки в меню.

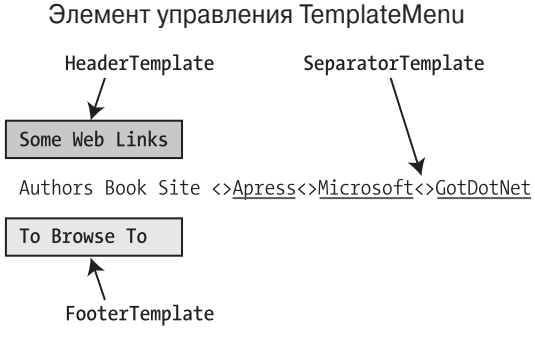

Рис. 6.4. Шаблоны элемента управления `TemplateMenu`

Первый шаг в построении нашего составного элемента управления — наследование его от `CompositeControl`. Это позволит использовать включающий дескриптор `<DIV>` вместе с реализацией `INamingContainer`, что обязательно для шаблонных элементов управления, поскольку вы будете встраивать элементы управления в шаблоны, которые могут вступить в конфликт с их значениями идентификаторов.

`CompositeControl` — это один из новых базовых классов, перечисленных в главе 2, который предоставляет дополнительную полезную функциональность разработчику элемента управления. Первичная выгода от наследования от `CompositeControl` — ассоциированный дизайнер `CompositeControlDesigner`, который обеспечивает базовую поддержку времени проектирования для разработчиков составных элементов управления.

В следующем коде показано объявление нашего серверного элемента управления:

```
[ToolboxData("<{0}:templatemenu
runat=server></{0}:templatemenu>"),Designer(typeof(TemplateMenuDesigner))]
public class TemplateMenu : CompositeControl
   private ArrayList menuData;
  public TemplateMenu() : base(HtmlTextWriterTag.Div)
  {
    menuData = new ArrayList();
    menuData.Add(new MenuItemData("Authors Book Site","","",""));
    menuData.Add(new MenuItemData("Apress","http://www.apress.com","",""));
    menuData.Add(new MenuItemData("Microsoft","http://www.microsoft.com","",""));
    menuData.Add(new MenuItemData("GotDotNet","http://asp.net","",""));
  }
...
}
```

Обратите внимание на атрибуты, примененные к классу `TemplateMenu`. Атрибут `ToolBoxData` предоставляет средства для настройки начального HTML, такие как начальное значение и значение по умолчанию, когда элемент управления перетаскивается из панели инструментов Visual Studio Toolbox и помещается на поверхность проектирования.

Поскольку мы хотим иметь возможность шаблоны с использованием среды проектирования Visual Studio, переопределим `CompositeControlDesigner` по умолчанию и добавим наш собственный специальный дизайнер через атрибут `Designer`. Создание специальных классов дизайнеров мы опишем в главе 11, но для демонстрации функциональности воспользуемся им уже здесь.

Конструктор для нашего элемента управления наполняет приватную коллекцию `ArrayList`, которая содержит заголовок и URL каждой гиперссылки в меню. Жесткое кодирование данных в элементе управления заменяется в последующих примерах, где связанные данные представлены через вложенные дочерние дескрипторы.

`MenuItemData` — это тип, содержащийся в приватном `ArrayList`. Он имеет свойства для заголовка гиперссылки, URL, на который перенаправляется браузер, URL для отображения графического изображения вместо текста, и свойство `Target` для направления определенного фрейма для загрузки адреса из гиперссылки. Мы используем его для хранения данных гиперссылки во всех наших примерах меню.

Свойства шаблона

Поскольку `WebControl` разбирает дочернее содержимое на свойства, мы должны дать анализатору страницы ASP.NET цель, которая соответствует свойствам нашего элемента управления, когда он встречает дочерние дескрипторы `HeaderTemplate`, `FooterTemplate` и `SeparatorTemplate` на странице `.aspx`. Элемент управления делает это, представляя свойства типа `ITemplate` именно с этими именами:

```
private ITemplate headerTemplate;
[Browsable(false),Description("The header template"),
PersistenceMode(PersistenceMode.InnerProperty),
TemplateContainer(typeof(BasicTemplateContainer))]
public ITemplate HeaderTemplate
{
  get
  {
    return headerTemplate;
  }
  set
  {
    headerTemplate = value;
  }
}
```

Предыдущий фрагмент представляет часть элемента управления, реализующую свойство HeaderTemplate и его хранилище. Свойство HeaderTemplate выполняет get и set приватного поля типа ITemplate по имени headerTemplate. Среда ASP.NET достаточно интеллектуальна, чтобы запросить тип свойства и распознать, что он работает с ITemplate. Затем он выполняет ряд шагов для создания экземпляра класса Template и присваивания его свойству. Код для свойств FooterTemplate и SeparatorTemplate идентичен коду свойства HeaderTemplate.

И последнее, что необходимо свойству шаблона — это атрибут TemplateContainer, сообщающий ASP.NET тип элемента управления, который будет содержать шаблон. Этот шаг привязывает содержимое элемента управления внутри шаблона к его внешнему контейнеру и позволяет работать ценным дополнительным средствам наподобие привязки данных.

BasicTemplateContainer — очень простая оболочка WebControl, которая визуализируется в дескриптор <SPAN>. Полное определение класса выглядит так:

```
public class BasicTemplateContainer : WebControl, INamingContainer
{
  public BasicTemplateContainer() : base(HtmlTextWriterTag.Span)
  {
    this.BorderWidth = 2;
    this.BorderStyle = BorderStyle.Outset;
  }
}
```

Сам шаблонный контейнер может быть настроен, как показано ранее, посредством модификации атрибутов BorderWidth и BorderStyle.

Шаблон-разделитель выглядел бы неважно с модифицированными атрибутами границ, поэтому он использует SeparatorTemplateContainer, который визуализируется как пустой HTML-дескриптор Span:

```
public class SeparatorTemplateContainer : WebControl, INamingContainer
{
  public SeparatorTemplateContainer() : base(HtmlTextWriterTag.Span)
  {
  }
}
```

Создание раздела Header

Элемент управления TemplateMenu — это составной элемент, так что ему нужно переопределить метод CreateChildControls, чтобы добавить дочерние элементы в свою коллекцию Controls. Элемент управления абстрагирует этот процесс, используя вспомогательный метод CreateControlHierarchy, чтобы выполнить работу по созданию дочерних элементов. CreateControlHierarchy содержит код для добавления шаблонов и гиперссылок как дочерних элементов управления:

```
override protected void CreateChildControls()
{
  Controls.Clear();
  CreateControlHierarchy();
}
```

CreateControlHierarchy начинается с работы с шаблоном HeaderTemplate. Первый момент, который всегда должен проверяться — имеет ли свойство значение. Это определяется проверкой свойства шаблона на равенство значению null:

```
if (HeaderTemplate != null)
{
  BasicTemplateContainer header = new BasicTemplateContainer();
  HeaderTemplate.InstantiateIn(header);
  Controls.Add(header);
  Controls.Add(new LiteralControl("<br>"));
}
```

Если свойство-шаблон равно `null`, то в большинстве случаев серверный элемент управления должен визуализировать обобщенный шаблон HTML по умолчанию, чтобы вывод был согласованным. Элемент управления ASP.NET `DataGrid` делает это посредством визуализации простой таблицы HTML, когда он привязывается к источнику данных с шаблонами. Для шаблона `HeaderTemplate` мы игнорируем шаблон и не отображаем ничего, если он равен `null`.

После того, как код проверит свойство шаблона на равенство `null`, он создает экземпляр контейнера, который будет служить вместилищем для содержимого шаблона. Элемент управления `TemplateMenu` оборачивает все шаблоны в HTML-элемент `<SPAN>` посредством использования специального элемента управления `BasicTemplateContainer`, основанного на `System.Web.UI.WebControls.WebControl`. Чтобы загрузить содержимое шаблона в элемент управления `BasicTemplateContainer`, интерфейс `ITemplate` предоставляет метод `InstantiateIn`, который принимает контейнер в параметре, как показано на рис. 6.5.

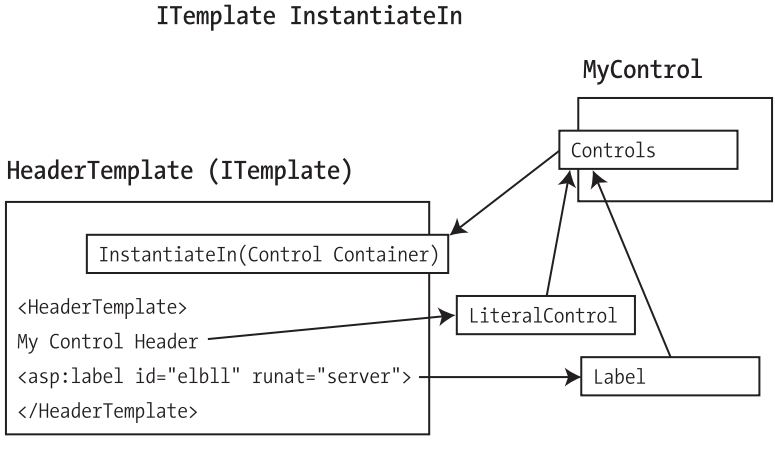

Рис. 6.5. `ITemplate` и `InstantiateIn`

Использование `InstantiateIn` завершает работу, необходимую для контейнера `header` элемента. Затем код элемента управления добавляет `header` в свою коллекцию `Controls`. Мы также добавляем объект `LiteralControl`, который визуализирует дескриптор `<BR>`, чтобы сделать отделение заголовка от гиперссылок обязательным средством визуализации UI элемента управления `TemplateMenu`.

Создание раздела `Footer`

Код для `FooterTemplate` в конце `CreateControlHierarchy` почти идентичен тому, что мы видели в `HeaderTemplate`. Единственная реальная разница между различными шаблонными свойствами заключается в добавлении дескриптора `<BR>` перед добавлением содержимого шаблона.

```
if (FooterTemplate != null)
{
  Controls.Add(new LiteralControl("<br>"));
  BasicTemplateContainer footer = new BasicTemplateContainer();
  FooterTemplate.InstantiateIn(footer);
  Controls.Add(footer);
}
```

Создание раздела Hyperlink

Середина кода для метода `CreateControlHierarchy` добавляет пункты в меню, используя элемент управления ASP.NET `HyperLink`. Данные для обработки представлены коллекцией `ArrayList` из приватного поля `menuData`, экземпляр которого мы создаем в конструкторе элемента управления.

Первая задача при построении каждой гиперссылки — итерация по `ArrayList` с именем `menuData`. Мы используем цикл и счетчик для отслеживания того, когда нам нужно применить шаблон `MenuSeparatorTemplate` для разделения гиперссылок. Цикл управляет извлечением экземпляров класса `MenuItemData` из коллекции и выполнением вспомогательного метода `CreateMenuItem`.

```
int count = menuData.Count;
for (int index = 0; index < count; index++)
{
  MenuItemData itemdata = (MenuItemData) menuData[index];
  CreateMenuItem(itemdata.Title, itemdata.Url,itemdata.ImageUrl, itemdata.Target);
  if (index != count-1)
  {
    if (SeparatorTemplate != null)
    {
      SeparatorTemplateContainer separator = new SeparatorTemplateContainer ();
      SeparatorTemplate.InstantiateIn(separator);
      Controls.Add(separator);
    }
    else
    {
      Controls.Add(new LiteralControl(" | "));
    }
  }
}
```

`CreateMenuItem` создает элемент управления ASP.NET `HyperLink` и добавляет его к дочерним элементам элемента `TemplateMenu`:

```
private void CreateMenuItem(string title, string url,
                            string target, string imageurl)
{
  HyperLink link = new HyperLink();
  link.Text = title;
  link.NavigateUrl = url;
  link.ImageUrl = imageurl;
  link.Target = target;
  Controls.Add(link);
}
```

`SeparatorTemplate` использует другой шаблонный контейнер — `SeparatorTemplateContainer`, но иначе: код следует за ведущими шаблонами `FooterTemplate` и `HeaderTemplate`. Что уникально в этом фрагменте — это добавление кода для визуализации удобного разделителя через `LiteralControl` на случай, если пользователь решит не привязывать значение `SeparatorTemplate` на странице `.aspx`.

Когда вы строите серверный элемент управления, который поддерживает шаблоны, рекомендуем убедиться, что элемент управления работает правильно, или, по крайней мере, изящно упрощается, с базовым шаблоном по умолчанию, если шаблоны не специфицированы.

Полная версия элемента управления `TemplateMenu` показана в листинге 6.1.

Листинг 6.1. Элемент управления `TemplateMenu`

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Collections;
using System.ComponentModel;
using ControlsBook2Lib.Ch11.Design;
namespace ControlsBook2Lib.Ch06
{
  [ToolboxData("<{0}:templatemenu runat=server></{0}:templatemenu>"),
  Designer(typeof(TemplateMenuDesigner))]
  public class TemplateMenu : CompositeControl
  {
    private ArrayList menuData;
    public TemplateMenu()
      : base()
    {
      menuData = new ArrayList()
      // Используется новое средство инициализации объекта и коллекции C# 3.0
        {
          new MenuItemData{Title="Apress", Url="http://www.apress.com"},
          new MenuItemData{Title="Microsoft", Url="http://www.microsoft.com"},
          new MenuItemData{Title="ASP.Net", Url="http://asp.net"}
        };
    }
    private ITemplate headerTemplate;
    [Browsable(false), Description("The header template"),
    PersistenceMode(PersistenceMode.InnerProperty),
    TemplateContainer(typeof(BasicTemplateContainer))]
    public ITemplate HeaderTemplate
    {
      get
      {
        return headerTemplate;
      }
      set
      {
        headerTemplate = value;
      }
    }
    private ITemplate footerTemplate;
    [Browsable(false), Description("The footer template"),
    PersistenceMode(PersistenceMode.InnerProperty),
    TemplateContainer(typeof(BasicTemplateContainer))]
    public ITemplate FooterTemplate
    {
      get
      {
        return footerTemplate;
      }
```

```
  set
  {
    footerTemplate = value;
  }
}
private ITemplate separatorTemplate;
[Browsable(false), Description("The separator template"),
PersistenceMode(PersistenceMode.InnerProperty),
TemplateContainer(typeof(SeparatorTemplateContainer))]
public ITemplate SeparatorTemplate
{
  get
  {
    return separatorTemplate;
  }
  set
  {
    separatorTemplate = value;
  }
}
private void CreateControlHierarchy()
{
  if (HeaderTemplate != null)
  {
    BasicTemplateContainer header = new BasicTemplateContainer();
    HeaderTemplate.InstantiateIn(header);
    Controls.Add(header);
  }
  int count = menuData.Count;
  for (int index = 0; index < count; index++)
  {
    MenuItemData itemdata = (MenuItemData)menuData[index];
    HyperLink link = new HyperLink() { Text = itemdata.Title,
    NavigateUrl = itemdata.Url, ImageUrl = itemdata.ImageUrl,
    Target = itemdata.Target };
    Controls.Add(link);
    if (index != count - 1)
    {
      if (SeparatorTemplate != null)
      {
        SeparatorTemplateContainer separator = new SeparatorTemplateContainer();
        SeparatorTemplate.InstantiateIn(separator);
        Controls.Add(separator);
      }
      else
      {
        Controls.Add(new LiteralControl(" | "));
      }
    }
  }
  if (FooterTemplate != null)
  {
    BasicTemplateContainer footer = new BasicTemplateContainer();
    FooterTemplate.InstantiateIn(footer);
    Controls.Add(footer);
  }
}
```

```
    override protected void CreateChildControls()
    {
      Controls.Clear();
      CreateControlHierarchy();
    }
    public override ControlCollection Controls
    {
      get
      {
        EnsureChildControls();
        return base.Controls;
      }
    }
    public override void DataBind()
    {
      CreateChildControls();
      ChildControlsCreated = true;
      base.DataBind();
    }
  }
}
```

Новые средства C# 3.0 инициализации объектов и инициализации коллекций вступают в дело при создании экземпляра `menuArray` в листинге 6.1. Исходный файл для шаблонов — `TemplateContainers.cs` — показан в листинге 6.2. Листинг 6.3 содержит класс данных `MenuItemData`, используемый для наполнения меню гиперссылок.

Листинг 6.2. Файл кода `TemplateContainers.cs`

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace ControlsBook2Lib.Ch06
{
  public class BasicTemplateContainer : WebControl, INamingContainer
  {
    public BasicTemplateContainer() : base(HtmlTextWriterTag.Span)
    {
      this.BorderWidth = 2;
      this.BorderStyle = BorderStyle.Outset;
    }
  }
  public class SeparatorTemplateContainer : WebControl, INamingContainer
  {
    public SeparatorTemplateContainer() : base(HtmlTextWriterTag.Span)
    {
    }
  }
}
```

Листинг 6.3. Класс данных `MenuItemData`

```
using System;
using System.ComponentModel;
namespace ControlsBook2Lib.Ch06
{
  [TypeConverter(typeof(ExpandableObjectConverter))]
```

```
  public class MenuItemData
  {
    public MenuItemData()
    {
    }
    // Переопределите этот метод для отображения только
    // MenuItemData вместо полностью квалифицированного типа
    // в специальном редакторе коллекции
    public override string ToString()
    {
      return "MenuItemData";
    }
    [NotifyParentProperty(true)]
    public string Title {get; set; }

    [NotifyParentProperty(true)]
    public string Url { get; set; }

    [NotifyParentProperty(true)]
    public string ImageUrl {get; set; }

    [NotifyParentProperty(true)]
    public string Target { get; set; }
  }
}
```

Просмотр элемента управления `TemplateMenu`

Web-форма, созданная для просмотра элемента управления `TemplateMenu`, вряд ли может быть проще. Она состоит из единственного элемента управления в форме, и все. Следующие шаблоны используют средства настройки UI элемента управления:

```
<apress:TemplateMenu id="menu1" runat="server" height="43px" width="224px">
  <SeparatorTemplate> | </SeparatorTemplate>
  <HeaderTemplate>
    <span style="FONT-WEIGHT: bold; COLOR: white;
BACKGROUND-COLOR: blue">Please follow the link of interest</span>
  </HeaderTemplate>
  <FooterTemplate>
    <span style="FONT-WEIGHT: bold; COLOR: white;
BACKGROUND-COLOR: red">Thanks for visiting this site</span>
  </FooterTemplate>
</apress:TemplateMenu><br/><br/>
```

Мы можем редактировать шаблоны для Web-формы щелчком на вкладке HTML в Visual Studio. .NET Framework также представляет возможность визуального редактирования шаблонов, а атрибут `Designer(typeof(TemplateMenuDesigner))]`, примененный к классу `TemplateMenu`, предоставляет эту поддержку для серверного элемента управления `TemplateMenu`.

Этот специальный дизайнер, который мы детально опишем в главе 11, добавляет команду меню Edit Templates (Редактировать шаблоны) к списку задач для элемента управления. Щелкните на стрелке задачи при выбранном элементе управления в дизайнере, щелкните на Edit Template, затем выберите шаблон, который вы хотите редактировать. Это откроет визуальный графический интерфейс для шаблона, где вы сможете перетаскивать другие элементы управления ASP.NET, такие как элемент — изображение для шаблона сепаратора, и редактировать стиль шаблона в инструментальном окне

Properties (Свойства). На рис. 6.6 показано, как будет выглядеть наш элемент управления после настройки его шаблонов. Листинги 6.4 и 6.5 содержат, соответственно, файл `.aspx` и класс отделенного кода для Web-формы `TemplateMenu`.

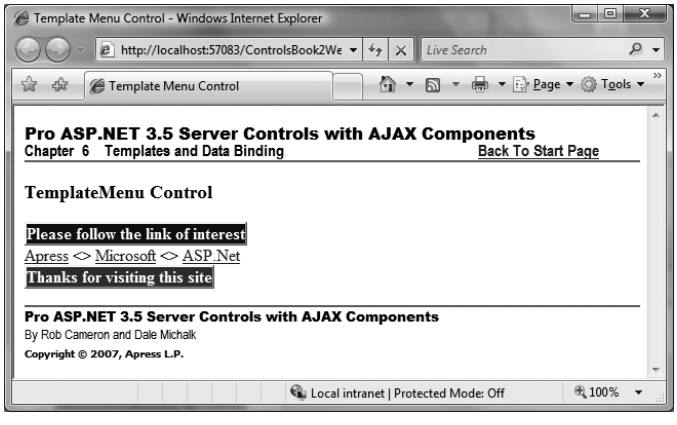

Рис. 6.6. Web-форма `TemplateMenu`, отображаемая в браузере

Листинг 6.4. Файл `.aspx` Web-формы `TemplateMenu`

```
<%@ Page Language="C#"
  MasterPageFile="~/MasterPage/ControlsBook2MasterPage.Master"
  AutoEventWireup="true" CodeBehind="TemplateMenu.aspx.cs"
  Inherits="ControlsBook2Web.Ch06.TemplateMenu"
  Title="Template Menu Control Demo" %>
<%@ Register TagPrefix="apress" Namespace="ControlsBook2Lib.Ch06"
  Assembly="ControlsBook2Lib" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ChapterNumAndTitle" runat="server">
  <asp:Label ID="ChapterNumberLabel" runat="server"
    Width="14px">6</asp:Label>  <asp:Label
    ID="ChapterTitleLabel" runat="server" Width="360px">
    Server Control Templates</asp:Label>
    </asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="PrimaryContent" runat="server">
  <h3>
    TemplateMenu Control</h3>
  <apress:TemplateMenu ID="menu1" runat="server" Height="43px" Width="224px">
    <SeparatorTemplate>
      &lt;&gt;
    </SeparatorTemplate>
    <HeaderTemplate>
      <div style="font-weight: bold; color: white; background-color: blue">
        Please follow the link of interest</div>
    </HeaderTemplate>
    <FooterTemplate>
      <div style="font-weight: bold; color: white; background-color: red">
        Thanks for visiting this site</div>
    </FooterTemplate>
  </apress:TemplateMenu>
  <br />
  <br />
</asp:Content>
```

Листинг 6.5. Файл класса отделенного кода `TemplateMenu`

```
using System;
namespace ControlsBook2Web.Ch06
{
  public partial class TemplateMenu : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
    }
  }
}
```

Просмотр визуализированного HTML

HTML, визуализированный элементом управления `TemplateMenu`, доказывает, что элемент управления успешно включил шаблон в свой вывод:

```
<div id="ctl00_ControlsBookContent_menu1" style="height:43px;width:224px;">
  <span style="display:inline-block;border-width:2px;border-style:Outset;">
    <span style="FONT-WEIGHT: bold; COLOR: white; BACKGROUND-COLOR: blue">
    Please follow the link of interest
    </span>
  </span><br>
  <a href="http://www.apress.com">Apress</a><span> &lt;&gt; </span>
  <a href="http://www.microsoft.com">Microsoft</a><span> &lt;&gt; </span>
  <a href="http://asp.net">ASP.Net</a><br>
  <span style="display:inline-block;border-width:2px;border-style:Outset;">
    <span style="FONT-WEIGHT: bold; COLOR: white; BACKGROUND-COLOR: red">
     Thanks for visiting this site
    </span>
  </span>
</div>
```

Шаблоны `HeaderTemplate` и `FooterTemplate` поступают в финальный HTML дословно. Код, строящий гиперссылки, также корректно вставляет шаблона `SeparatorTemplate` с символами <>. В следующем разделе мы поговорим, как сохранять данные в виде дочерних дескрипторов, являющихся частью серверного элемента управления.

Разбор данных из дескрипторов элемента управления

Элемент управления `TemplateMenu` имеет одно существенное ограничение. Данные, которые он использует для отображения гиперссылок, жестко закодированы в его конструкторе. Пользователь этого элемента должен иметь исходный код, чтобы модифицировать то, что отображается. Это не лучший способ построения серверных элементов управления, которые должны быть гибкими и адаптируемыми. Предоставление Web-разработчикам возможности передавать необходимые данные — намного лучший подход, и именно его мы рассмотрим в последующих разделах.

Подход, которым мы воспользуемся для добавления настраиваемых данных к элементу, заключается в применении дочерних дескрипторов, передаваемых в данных элементу управления. Это подобно методу, используемому списочными элементами управления ASP.NET, такими как `DropDownList` и `CheckBoxList`, которые поддерживают использование дескриптора `asp:listitem` для декларативной передачи данных.

Серверный элемент управления, который мы построим в следующем разделе, использует атрибут ParseChildren. В следующем разделе мы построим серверный элемент управления, который продемонстрирует, как можно дальше настраивать этот процесс с применением класса ControlBuilder.

Элемент управления TagDataMenu

Элемент управления TagDataMenu — это пример элемента, который мы создадим в этом разделе, и который читает значения своих дочерних дескрипторов для построения коллекции данных для гиперссылок. Он делает это, используя преимущества атрибута ParseChildren, который указывает анализатору страниц ASP.NET, что его содержимое следует трактовать как элементы, подлежащие добавлению к коллекции.

Следующий атрибут показывает свойство ChildrenAsProperties, устанавливаемое в атрибут ParseChildren, наряду со свойством DefaultProperty:

```
[ParseChildren(ChildrenAsProperties=true, DefaultProperty="MenuItems")]
```

Следующий сокращенный код делает то же самое:

```
[ParseChildren(true, "MenuItems")]
```

Установка свойства DefaultProperty в атрибуте ParseChildren заставляет анализатор страниц ASP.NET искать все дочернее содержимое XML серверного элемента управления как члены коллекции. Эти члены создаются анализатором страниц и добавляются к коллекции, специфицированной свойством MenuItems. Тип элемента коллекции определяется именем дочернего дескриптора. ASP.NET ищет имя типа в проекте и создает объект для него, заполняя его свойства согласно атрибутам дескриптора.

TagDataMenu конфигурирует себя для использования коллекции MenuItems, устанавливая свой атрибут ParseChildren соответствующим образом:

```
ParseChildren attribute accordingly:
[ParseChildren(true, "MenuItems")]
[ToolboxData("<{0}:TagDataMenu runat=server></{0}:TagDataMenu>")]
public class TagDataMenu : CompositeControl
{
  public TagDataMenu() : base(HtmlTextWriterTag.Div)
  {
  }
...
}
```

На рис. 6.7 показаны отношения между элементом управления TagDataMenu и его коллекцией MenuItems.

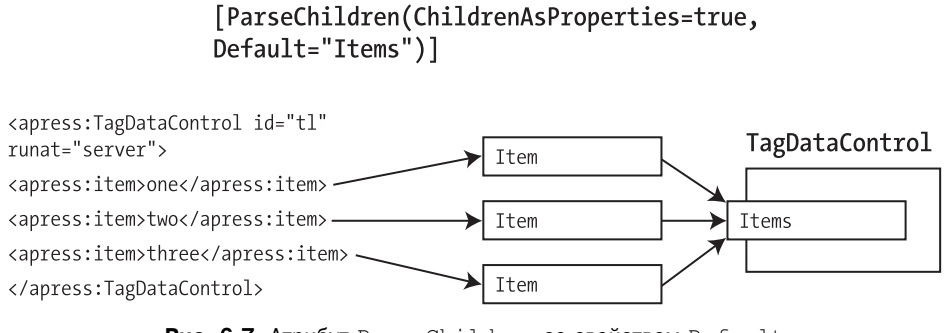

Рис. 6.7. Атрибут ParseChildren со свойством Default

Свойство MenuItems, представленное элементом управления, использует преимущества обобщений, реализуя List как List<MenuItemData>. Эта коллекция наполняется ASP.NET на основе данных дочернего дескриптора элемента управления.

```
private MenuItemDataCollection menuData;
[DesignerSerializationVisibility(DesignerSerializationVisibility.Content),
Description("Collection of MenuItemData objects for display"),
PersistenceMode(PersistenceMode.InnerDefaultProperty),NotifyParentProperty(true)]
public MenuItemDataCollection MenuItems
{
  get
  {
    if (menuData == null)
    {
      menuData = new MenuItemDataCollection();
    }
    return menuData;
  }
}
```

Допущение вхождений данных в дочерние дескрипторы связано с одним компромиссом: недостатком поддержки шаблонов. ASP.NET предполагает, что все дочерние дескрипторы попадают в коллекцию DefaultProperty, и не разбирает их, если видит шаблоны. Для таких простых элементов управления, как TagDataMenu, это оправданный компромисс. Более развитые элементы управления, вроде Repeater, который мы построим в главе 7, будут представлять свойства, позволяющие Web-разработчикам устанавливать источник данных программно.

Облегченный метод CreateControlHierarchy не должен заботиться о построении шаблонов. Мы можем повторно использовать CreateMenuItem из элемента управления TemplateMenu. Для разделения ссылок меню мы применяем символ канала (|):

```
override protected void CreateChildControls()
{
  Controls.Clear();
  CreateControlHierarchy();
}
private void CreateControlHierarchy()
{
  int count = menuData.Count;
  for (int index = 0; index < count; index++)
  {
    MenuItemData itemdata = (MenuItemData) menuData[index];
    CreateMenuItem(itemdata.Title, itemdata.Url,
      itemdata.ImageUrl, itemdata.Target);
    if ((count > 1) && (index < count -1))
    {
      Controls.Add(new LiteralControl(" | "));
    }
  }
}
```

Листинг 6.6. Файл класса элемента управления TagDataMenu

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Collections.Generic;
using System.ComponentModel;
using ControlsBook2Lib.Ch11.Design;
```

```
namespace ControlsBook2Lib.Ch06
{
  // Атрибут PaseChildren заставляет анализатор страниц ASP.NET трактовать
  // дочернее содержимое как элементы, подлежащие добавлению в коллекцию
  [ParseChildren(true, "MenuItems")]
  [ToolboxData("<{0}:tagdatamenu runat=server></{0}:tagdatamenu>")]
  public class TagDataMenu : CompositeControl
  {
    public TagDataMenu()
      : base()
    {
    }
    private List<MenuItemData> menuData = new List<MenuItemData>();
    // Эта коллекция автоматически наполняется ASP.NET
    // из-за атрибута класса ParseChildren
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Content),
    Description("Collection of MenuItemData objects for display"),
    PersistenceMode(PersistenceMode.InnerDefaultProperty),
    NotifyParentProperty(true)]
    public List<MenuItemData> MenuItems
    {
      get
      {
        if (menuData == null)
        {
          menuData = new List<MenuItemData>();
        }
        return menuData;
      }
    }
    private void CreateMenuItem(string title, string url, string
    target, string imageUrl)
    {
      HyperLink link = new HyperLink();
      link.Text = title;
      link.NavigateUrl = url;
      link.ImageUrl = imageUrl;
      link.Target = target;
      Controls.Add(link);
    }
    override protected void CreateChildControls()
    {
      Controls.Clear();
      CreateControlHierarchy();
    }
    private void CreateControlHierarchy()
    {
      int count = MenuItems.Count;
      for (int index = 0; index < count; index++)
      {
        MenuItemData itemdata = (MenuItemData)MenuItems[index];
        CreateMenuItem(itemdata.Title, itemdata.Url,
        itemdata.ImageUrl, itemdata.Target);
        if ((count > 1) && (index < count - 1))
        {
          Controls.Add(new LiteralControl(" | "));
        }
      }
    }
```

```
    public override ControlCollection Controls
    {
      get
      {
        EnsureChildControls();
        return base.Controls;
      }
    }
  }
}
```

Мы упоминали ранее о том, что свойство `MenuItems` использует обобщения и объявлено как `List`. Это исключает необходимость в создании специальной коллекции. Этот тип обеспечивает возможность добавить редактор коллекций во время проектирования, чтобы позволить редактировать данные пунктов меню через диалоговое окно, почти таким же образом, как встроенные серверные элементы управления ASP.NET, обладающие постоянством `InnerDefaultProperty`.

По умолчанию ASP.NET предлагает редактор типа UI для редактирования во время проектирования `MenuItems`. Между тем, для свойств, которые мы не хотим делать видимыми в инструментальном окне Properties в Visual Studio, мы добавляем следующий атрибут:

```
[Browsable(false)]
```

Для свойств коллекции, которые мы не хотим сериализовать в виде данных, необходимо добавить такой атрибут:

```
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
```

Атрибут `DesignerSerializationVisibility` указывает на то, является ли значение свойства видимым (`Visible`) и должно сохраняться в коде инициализации; или же оно скрыто (`Hidden`) и не должно сохраняться в коде инициализации; или же оно состоит из содержимого (`Content`), которое должно иметь код инициализации, сгенерированный для каждого общедоступного, не скрытого свойства объекта, присвоенного свойству. Значением по умолчанию, если атрибут отсутствует, является `Visible`, и Visual Studio Designer во время проектирования попытается сериализовать свойство в соответствии с его типом. Свойство `MenuItems` класса `TagDataMenu` — это пример свойства с видимостью, установленной в `Content`:

```
[DesignerSerializationVisibility(DesignerSerializationVisibility.Content),
PersistenceMode(PersistenceMode.InnerDefaultProperty),NotifyParentProperty(true)]
public MenuItemDataCollection MenuItems
  {
  }
...
}
```

Теперь мы перейдем к дискуссии о том, как настроить процесс разбора (анализа), используя класс `ControlBuilder`. Эта опция предоставлена для полной настройки процесса разбора, в чем вы убедитесь в следующем разделе.

Элемент управления `BuilderMenu`

`BuilderMenu` демонстрирует второй прием чтения дочерних дескрипторов и создания данных меню. Он использует средство ASP.NET, которое позволяет полностью перенастроить процесс разбора элемента управления за счет реализации специального класса `ControlBuilder`. Реализация элемента управления `BuilderMenu` будет иден-

тична TagDataMenu, за исключением возможности управления процессом разбора дескрипторов.

Нормальный ControlBuilder по умолчанию привязан к классам, унаследованным от базового System.Web.UI.Control, и решает перечисленные ниже задачи.

- Разбирает дочернее XML-содержимое по типам элементов управления.
- Создает дочерний элемент управления.
- Вызывает метод AddParsedSubObject интерфейса IParserAccessor, чтобы добавить дочерний элемент управления к коллекции Controls серверного элемента, как показано на рис. 6.8.

```
[ParseChildren(false),
ControlBuilder(typeof(MyControlBuilder)]
```

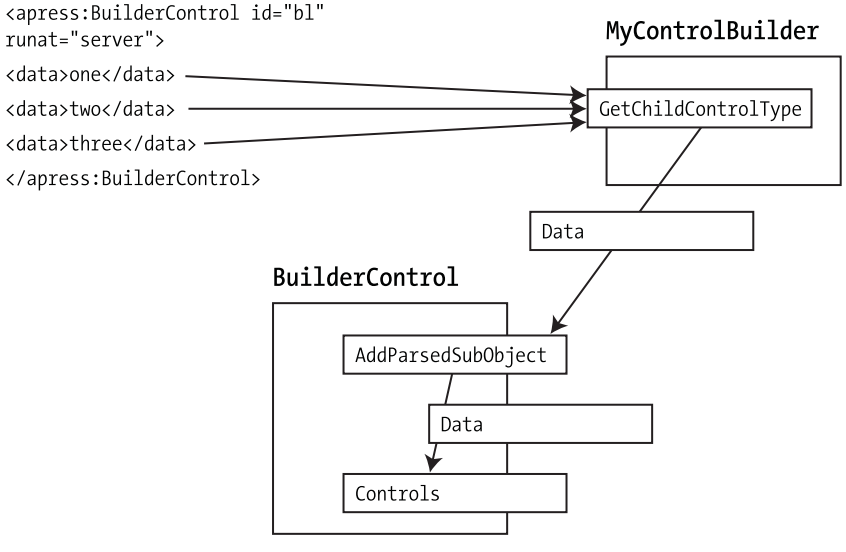

Рис. 6.8. Установки по умолчанию ControlBuilder и IParserAccessor.AddParsedSubObject

Специальный ControlBuilder получает возможность полностью перенастроить способ разбора дочернего содержимого анализатором ASP.NET. Он конфигурируется добавлением атрибута ControlBuilder. Часть объявления класса элемента управления BuilderMenu выглядит так:

```
[ParseChildren(false)]
[ControlBuilder(typeof(MenuControlBuilder))]
[ToolboxData("<{0}:BuilderMenu runat=server></{0}:BuilderMenu>")]
public class BuilderMenu : CompositeControl
{
  public BuilderMenu() : base()
  {
  }
...
}
```

Первое, что здесь нужно отметить — это установка в `false` атрибута `ParseChildren`. Это означает, что мы хотим, чтобы `ControlBuilder` принимал решение о том, как следует обработать дочерние дескрипторы XML. Атрибут, представляющий интерес — это `ControlBuilder`, который получает ссылку `System.Type` класса `MenuControlBuilder`.

`MenuControlBuilder` унаследован от класса `System.Web.UI.ControlBuilder` и переопределяет два метода для настройки процесса разбора. Наиболее частая причина для создания вашего собственного `ControlBuilder` заключается в необходимости переопределить метод `GetChildControlType`, чтобы элемент управления, к которому применен построитель (builder), мог определить, как выполнять отображение между дочерним дескриптором и классом, который должен быть создан для представление дочернего дескриптора во время обработки на стороне сервера.

```
public class MenuControlBuilder : ControlBuilder
{
public override Type GetChildControlType(String tagName, Dictionary attributes)
{
  if (String.Compare(tagName, "data", true) == 0)
  {
    return typeof(MenuItemData);
  }
  return null;
}
```

`MenuControlBuilder` ищет дочерние дескрипторы по имени `"data"`. Если он находит соответствие, то возвращает тип `MenuItemData` обратно элементу `BuilderMenu`, к которому он привязан. Это предполагает, что дескрипторы данных имеют соответствующие атрибуты, которые отображаются на свойства типа `MenuItemData`.

Другой метод, переопределяемый классом `MenuControlBuilder` — это `AppendLiteralString`. Мы предоставляем пустую реализацию этого метода, чтобы проигнорировать литеральное содержимое, находящееся между дескрипторами, содержащими данные.

```
public override void AppendLiteralString(string s)
{
   // Игнорирует литералы между дескрипторами
}
```

Есть и другие средства разбора `ControlBuilder`, которые мы опустили в этой демонстрации. Например, `ControlBuilder` может разбирать “сырое” содержимое строки между родительскими дескрипторами серверного элемента управления и обеспечивать поддержку для вложенных `ControlBuilder` для обработки содержимого дочернего элемента управления. За дополнительной информацией обращайтесь к документации .NET Framework.

Возвращаясь к элементу управления `BuilderMenu`, рассмотрим реализацию `IParserAccessor` и метода `AddParsedSubObject`. Следующий код просто добавляет объект данных, переданный в его внутреннюю коллекцию `ArrayList`, представленную полем `menuData`. Единственная проверка, которую он выполняет — это проверка типа для обеспечения передачи правильного экземпляра типа от `ControlBuilder`, ассоциированного с элементом управления.

```
protected override void AddParsedSubObject(Object obj)
{
  if (obj is MenuItemData)
  {
    menuData.Add(obj);
  }
}
```

В листингах 6.7 и 6.8 показан финальный код `BuilderMenu` и его вспомогательного `MenuControlBuilder`.

Листинг 6.7. Файл класса элемента управления `BuilderMenu`

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Collections;
using System.ComponentModel;
using ControlsBook2Lib.Ch11.Design;
namespace ControlsBook2Lib.Ch06
{
  [ParseChildren(false)]
  [ControlBuilder(typeof(MenuControlBuilder))]
  [ToolboxData("<{0}:buildermenu runat=server></{0}:buildermenu>")]
  public class BuilderMenu : CompositeControl
  {
    public BuilderMenu() : base()
    {
    }
    private ArrayList menuData = new ArrayList();
    public ArrayList MenuItems
    {
      get
      {
        return menuData;
      }
    }
    protected override void AddParsedSubObject(Object obj)
    {
      if (obj is MenuItemData)
      {
        menuData.Add(obj);
      }
    }
    private void CreateMenuItem(string title, string url,
                                string target, string imageUrl)
    {
      HyperLink link = new HyperLink();
      link.Text = title;
      link.NavigateUrl = url;
      link.ImageUrl = imageUrl;
      link.Target = target;
      Controls.Add(link);
    }
    private void CreateControlHierarchy()
    {
      int count = menuData.Count;
      for (int index = 0; index < count; index++)
      {
        MenuItemData itemdata = (MenuItemData)menuData[index];
        CreateMenuItem(itemdata.Title, itemdata.Url,
        itemdata.ImageUrl, itemdata.Target);
        if ((count > 1) && (index < count - 1))
        {
          Controls.Add(new LiteralControl(" | "));
        }
      }
    }
```

```
    override protected void CreateChildControls()
    {
      CreateControlHierarchy();
    }
    public override ControlCollection Controls
    {
      get
      {
        EnsureChildControls();
        return base.Controls;
      }
    }
  }
}
```

Листинг 6.8. Файл класса построителя элемента управления `MenuControlBuilder`

```
using System;
using System.Web;
using System.Web.UI;
using System.Collections;
namespace ControlsBook2Lib.Ch06
{
  public class MenuControlBuilder : ControlBuilder
  {
    public override Type GetChildControlType(String tagName, IDictionary attributes)
    {
      if (String.Compare(tagName, "data", true) == 0)
      {
        return typeof(MenuItemData);
      }
      return null;
    }
    public override void AppendLiteralString(string s)
    {
      s.Trim();
      // Игнорирует литералы между дескрипторами
    }
  }
}
```

Просмотр Web-формы элементов управления Tag Parsing Menu

Web-форма Tag Parsing Menu демонстрирует оба наши декларативно загружаемые элементы управления меню в действии. В Visual Studio, если вы щелкаете элемент управления `TagDataMenu` и выбираете свойство `MenuItems`, то видите кнопку, щелчок на которой вызовет UI редактора коллекции для `MenuItemData`. Для элемента управления `TagDataMenu` дочерние элементы принимают имя класса данных гиперссылок `MenuItemData`:

```
<apress:tagdatamenu id="menu1" runat="server">
  <apress:MenuItemData title="Apress" url="http://www.apress.com" imageurl=""
                       target="" />
  <apress:MenuItemData title="Microsoft" url="http://www.microsoft.com"
                       imageurl="" target="" />
  <apress:MenuItemData title="GotDotNet" url="http://www.gotdotnet.com"
                       imageurl="" target="" />
</apress:tagdatamenu>
```

Элемент управления `BuilderMenu` нуждается в дескрипторах с именами `"data"`, так что `MenuControlBuilder` захватывает каждый пункт и передает его элементу управления:

```
<apress:buildermenu id="menu2" runat="server">
  <data title="Apress" url="http://www.apress.com" imageurl="" target="" />
  <data title="Microsoft" url="http://www.microsoft.com" imageurl="" target="" />
  <data title="GotDotNet" url="http://www.gotdotnet.com" imageurl="" target="" />
</apress:buildermenu>
```

На рис. 6.9 можно видеть, что конечный результат для двух элементов управления в браузере после их визуализации идентичен.

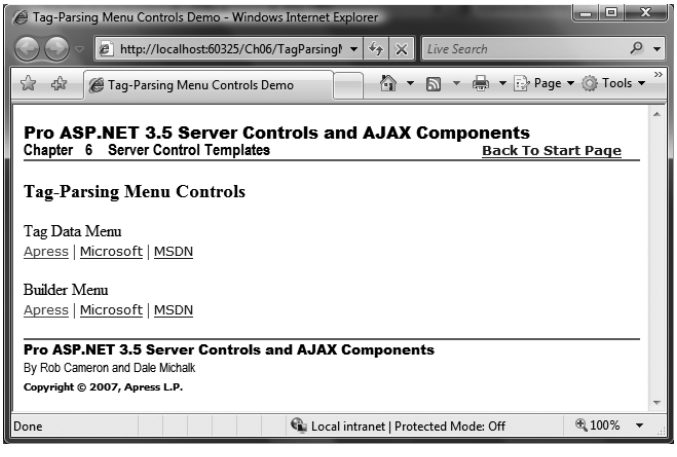

Рис. 6.9. Web-форма Tag Parsing Menu, отображенная в браузере

Листинг 6.9. Файл `.aspx` Web-формы Tag Parsing Menu

```
<%@ Page Language="C#"
  MasterPageFile="~/MasterPage/ControlsBook2MasterPage.Master"
  AutoEventWireup="true" CodeBehind="TagParsingMenu.aspx.cs"
  Inherits="ControlsBook2Web.Ch06.TagParsingMenu"
  Title="Tag-Parsing Menu Controls Demo" %>
<%@ Register TagPrefix="apress" Namespace="ControlsBook2Lib.Ch06"
  Assembly="ControlsBook2Lib" %>
<asp:Content ID="Content1" ContentPlaceHolderID=
      "ChapterNumAndTitle" runat="server">
  <asp:Label ID="ChapterNumberLabel" runat="server"
    Width="14px">6</asp:Label>  <asp:Label
    ID="ChapterTitleLabel" runat="server" Width="360px">
    Server Control Templates</asp:Label></asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID=
    "PrimaryContent" runat="server">
  <h3>
    Tag-Parsing Menu Controls</h3>
  <div id="TagDataMenu">
    Tag Data Menu<br />
    <apress:TagDataMenu ID="TagDataMenu1" runat="server">
      <apress:MenuItemData Title="Apress" ImageUrl=""
                           Url="http://www.apress.com" Target="">
      </apress:MenuItemData>
```

```
      <apress:MenuItemData Title="Microsoft" ImageUrl=""
              Url="http://www.microsoft.com" Target=""></apress:MenuItemData>
      <apress:MenuItemData Title="MSDN" ImageUrl=""
              Url="http://msdn.microsoft.com" Target="">
      </apress:MenuItemData>
    </apress:TagDataMenu>
  </div>
  <br />
  <div id="Builder Menu">
    Builder Menu<br />
    <apress:BuilderMenu ID="BuilderMenu1" runat="server">
    <data title="Apress" url="http://www.apress.com" imageurl="" target="" />
    <data title="Microsoft" url=http://www.microsoft.com imageurl="" target="" />
    <data title="MSDN" url="http://msdn.microsoft.com" imageurl="" target="" />
    </apress:BuilderMenu>
  </div>
  <br />
</asp:Content>
```

Листинг 6.10. Файл класса отделенного кода Web-формы Tag Parsing Menu

```
using System;
namespace ControlsBook2Web.Ch06
{
  public partial class TagParsingMenu : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
    }
  }
}
```

Резюме

Шаблоны — это один из двух первичных способов модификации графического содержимого Web-формы ASP.NET. Шаблоны предоставляют разработчикам способ декларативно вставлять "сырой" HTML через серверные элементы управления в вывод встроенного элемента управления. Шаблоны могут загружаться динамически через `Page.LoadTemplate` или создавать экземпляры посредством классов, реализующих интерфейс `ITemplate`.

Атрибут `ParseChildren` определяет, должен ли ASP.NET разбирать внутреннее содержимое дескриптора серверного элемента управления на странице `.aspx` как дочерние элементы или дочерние свойства. Атрибут `ControlBuilder` позволяет элементу перенаправлять процесс разбора дочерних элементов специализированному классу `ControlBuilder`.

В следующей главе мы продолжим нашу дискуссию об элементах управления на основе шаблонов, добавив в коктейль привязку данных.