第 17 章　ASP.NET MVC 基础

在 ASP.NET 应用程序开发中，开发人员很难将 ASP.NET 应用程序进行良好分层并使相应的页面进行相应的输出，例如页面代码只进行页面布局和样式的输出而代码页面只负责进行逻辑的处理。为了解决这个问题，微软开发了 MVC 开发模式方便开发人员进行分层开发。

17.1　了解 MVC

MVC 是一个设计模式，MVC 能够将 ASP.NET 应用程序的视图、模型和控制器进行分开，开发人员能够在不同的层次中进行应用程序层次的开发，例如开发人员能够在视图中进行页面视图的开发，而在控制器中进行代码的实现。

17.1.1　MVC 和 Web Form

在 ASP.NET Web Form 的开发当中，用户能够方便的使用微软提供的服务器控件进行应用程序的开发，从而提高开发效率。虽然 ASP.NET Web Form 提高了开发速度、维护效率和代码的复用性，但是 ASP.NET 现有的编程模型抛弃了传统的网页编程模型，在很多应用问题的解决上反而需要通过复杂的实现完成。

在 ASP.NET MVC 模型中，ASP.NET MVC 模型给开发人员的感觉仿佛又回到了传统的网页编程模型中（如 ASP 编程模型），但是 ASP.NET MVC 模型与传统的 ASP 同样是不同的编程模型，因为 ASP.NET MVC 模型同样是基于面向对象的思想进行应用程序的开发。

相比之下，ASP.NET MVC 模型是一种思想，而不是一个框架，所以 ASP.NET MVC 模型与 ASP.NET Web Form 并不具有可比性。同样 ASP.NET MVC 模型也不是 ASP.NET Web Form 4.0，这两个开发模型就好比一个是汽车一个是飞机，而两者都能够达到同样的目的。

ASP.NET MVC 模型是另一种 Web 开发的实现思路，其实现的过程并不像传统的 ASP.NET 应用程序一样。当用户通过浏览器请求服务器中的某个页面时，其实是实现了 ASP.NET MVC 模型中的一个方法，而不是具体的页面，这在另一种程度上实现了 URL 伪静态。当用户通过浏览器请求服务器中的某一个路径时，ASP.NET MVC 应用程序会拦截相应的地址并进行路由解析，通过应用程序中编程实现展现一个页面给用户，这种页面展现手法同传统的 ASP.NET Web From 应用程序与其他的如 ASP，PHP 等应用程序都不相同。

同时，随着互联网的发展，搜索引擎在 Web 开发中起着重要的作用，这就对页面请求的地址有了更加严格的要求。例如百度、谷歌等搜索引擎会对目录形式的页面路径和静态形式的页面路径收录的更好，而对于动态的如 abc.aspx?id=1&action=add&t=3 这种样式的页面路径不甚友好。

另外，所有引擎又在一定程度上决定了 Web 应用的热度，例如当在百度中搜索“鞋”这个关键字时，如果搜索的结果中客户的网站在搜索结果的后几页，用户通常不会进行翻页查询，相比之下用户更喜欢在搜索结果中查看前几页的内容。

ASP.NET MVC 开发模型在用户进行页面请求时会进行 URL 拦截并通过相应的编程实现访问路径

和页面的呈现，这样就能够更加方便的实现目录形式的页面路径和静态形式，对于 Web 应用动态的地址如 abc.aspx?id=1&action=add&t=3 可以以 abc/action/id/add 的形式呈现，这样就更加容易的被搜索引擎所搜录。

注意：ASP.NET MVC 模型和 ASP.NET Web Form 并不具备可比性，因为 ASP.NET MVC 模型和 ASP.NET Web Form 是不同的开发模型，而 ASP.NET MVC 模型和 ASP.NET Web Form 在各自的应用上都有有点和缺点，并没有哪个开发模型比另一个模型好之说。

17.1.2 ASP.NET MVC 的运行结构

在 ASP.NET MVC 开发模型中，页面的请求并不是像传统的 Web 应用开发中的请求一样是对某个文件进行访问，初学者可能会在一开始觉得非常的不适应。例如当用户访问/home/abc.aspx 时，在服务器的系统目录中一定会存在 abc.aspx 这个页面，而对于传统的页面请求的过程也非常容易理解，因为在服务器上只有存在了 home 文件夹，在 home 文件夹下一定存在 abc.aspx 页面才能够进行相应的页面访问。

对于 ASP.NET MVC 开发模型而言，当请求 URL 路径为“/home/abc.aspx”时，也许在服务器中并不存在相应的 abc.aspx 页面，而可能是服务器中某个方法。在 ASP.NET MVC 应用程序中，页面请求的地址不能够按照传统的概念进行分析，要了解 ASP.NET MVC 应用程序的页面请求地址就需要了解 ASP.NET MVC 开发模型的运行结构。ASP.NET MVC 开发模型的运行结构如图 17-1 所示。

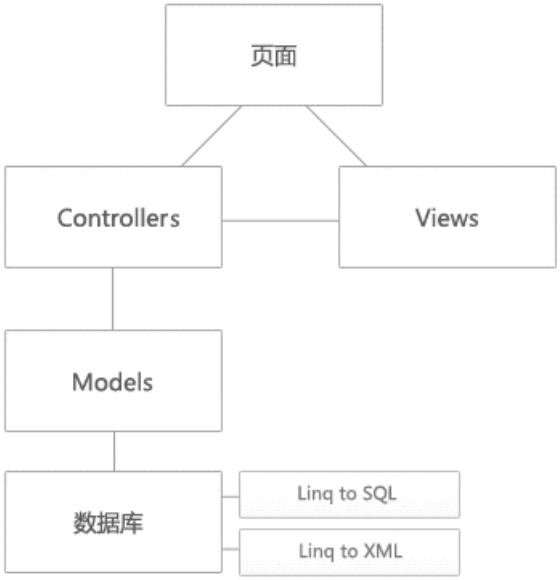

图 17-1 ASP.NET MVC 开发模型

正如图 17-1 所示，ASP.NET MVC 开发模型包括三个模块，这三个模块分别为 MVC 的 M、V、C，其中 M 为 Models（模型）、V 为 Views（视图）、C 为 Controllers（控制器），在 ASP.NET MVC 开发模型中，这三个模块的作用分别如下所示。

- Models：Models 负责与数据库进行交互，在 ASP.NET MVC 框架中，使用 LINQ 进行数据库连接和操作。
- Views：Views 负责页面的页面呈现，包括样式控制，数据的格式化输出等。
- Controllers：Controllers 负责处理页面的请求，用户呈现相应的页面。

与传统的页面请求和页面运行方式不同的是，ASP.NET MVC 开发模型中的页面请求首先会发送到 Controllers 中，Controllers 再通过 Models 进行变量声明和数据读取。Controller 通过页面请求和路由设置呈现相应的 View 给浏览器，用户就能够在浏览器中看到相应的页面。这里讲解 ASP.NET MVC 开发模型的工作流程可能会让读者感到困惑，具体 ASP.NET MVC 开发模型的工作流程会在后面详细讲解。

17.2　ASP.NET MVC 基础

ASP.NET MVC 开发模型和 ASP.NET Web From 开发模型并不相同，ASP.NET MVC 为 ASP.NET Web 开发进行了良好的分层，ASP.NET MVC 开发模型和 ASP.NET Web From 开发模型在请求处理和应用上都不尽相同，只有了解 ASP.NET Web From 开发模型的基础才能够高效的开发 MVC 应用程序。

17.2.1　安装 ASP.NET MVC

ASP.NET MVC 是微软推出的最新的 ASP.NET Web 开发模型，开发人员可以在微软的官方网站上下载 ASP.NET MVC 安装程序，也能够使用光盘中附属的 ASP.NET MVC 安装程序进行安装，光盘中附带的是 ASP.NET MVC beta 版本，正式版同 beta 版本基本上没有任何区别，开发人员可以在官方网站下载最新的安装程序。单击下载或附录中的 AspNetMVCBeta-setup.msi 进行 ASP.NET MVC 开发模型的安装和相应示例的安装，如图 17-2 所示。

用户单击 ASP.NET MVC 安装界面中的【Next】按钮进入 ASP.NET MVC 安装的用户条款界面，单击【I accept the terms int the License Agreement】复选框同意 ASP.NET MVC 用户条款，如图 17-3 所示。同意后单击【Next】按钮进入 ASP.NET MVC 安装准备界面，进入安装界面后单击【Install】按钮进行安装。

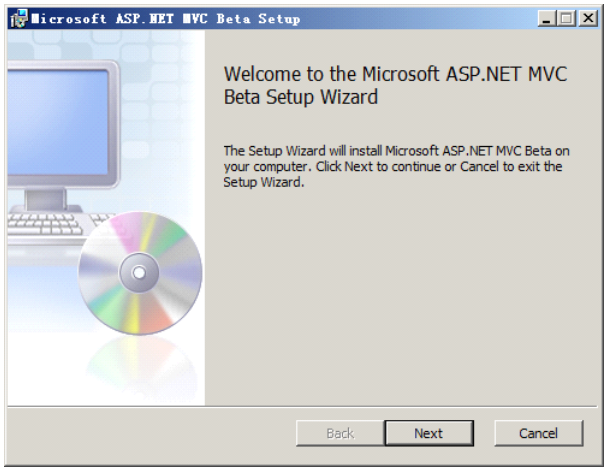

图 17-2　ASP.NET MVC 安装界面

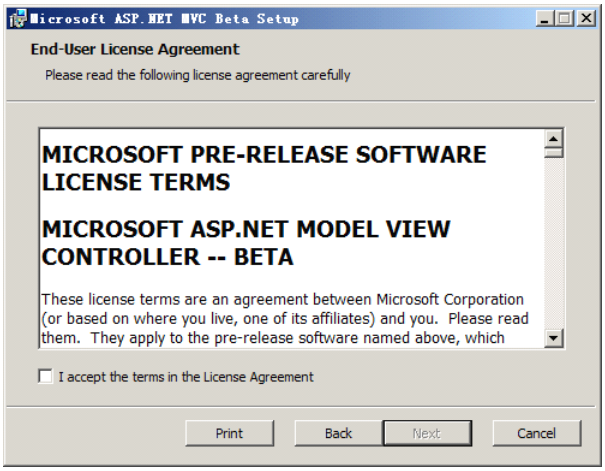

图 17-3　ASP.NET MVC 用户条款

注意：在安装 ASP.NET MVC 前必须安装 Visual Studio 2008 进行 ASP.NET MVC 应用程序的开发，安装完成 ASP.NET MVC 应用程序后就能够在 Visual Studio 2008 进行创建 ASP.NET MVC 应用程序。

单击【Install】按钮应用程序，系统就会在计算机中安装 ASP.NET MVC 开发模型和 Visual Studio 2008 中进行 ASP.NET MVC 程序开发所需要的必备组件以便在 Visual Studio 2008 为开发人员提供原生的 ASP.NET MVC 开发环境。安装完毕后，安装程序会提示 ASP.NET MVC 安装程序已经安装完毕，安装

完毕后开发人员就能够使用 Visual Studio 2008 开发 ASP.NET MVC 应用程序。安装过程如图 17-4 和 17-5 所示。

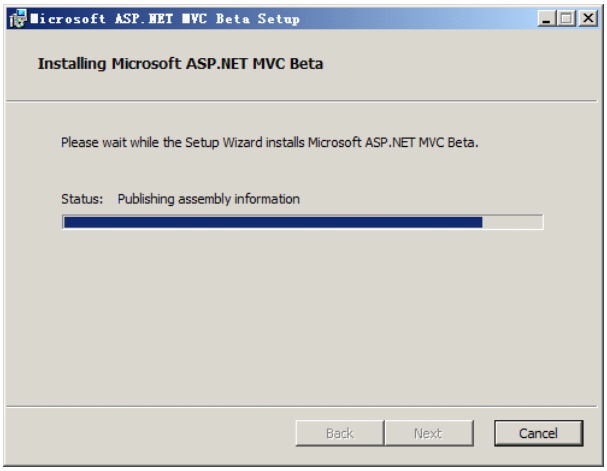

图 17-4　ASP.NET MVC 安装

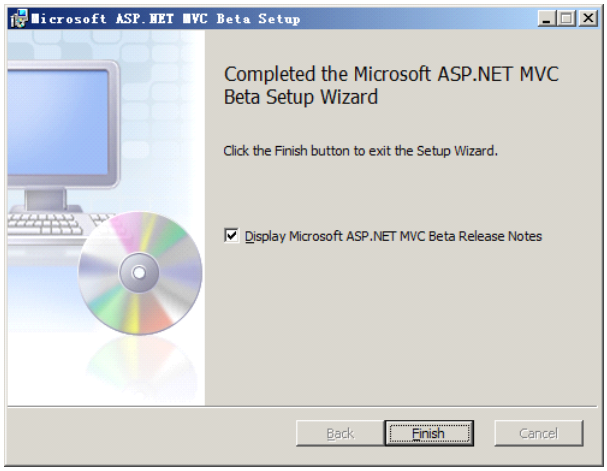

图 17-5　ASP.NET MVC 安装完毕

17.2.2　新建一个 MVC 应用程序

安装完成 ASP.NET MVC 开发模型后就能够在 Visual Studio 2008 中创建 ASP.NET MVC 应用程序进行 ASP.NET MVC 应用程序的开发，安装 ASP.NET MVC 开发模型后，Visual Studio 2008 就能够为 ASP.NET MVC 提供原生的开发环境。在菜单栏中选择【文件】选项，单击【文件】选项在下拉菜单中选择【新建项目】就能够创建 ASP.NET MVC 应用程序，如图 17-6 所示。

单击【确定】按钮后就能够创建 ASP.NET MVC 应用程序。Visual Studio 2008 为 ASP.NET MVC 提供了原生的开发环境，以及智能提示，开发人员在进行 ASP.NET MVC 应用程序开发中，Visual Studio 2008 同样能够为 ASP.NET MVC 应用程序提供关键字自动补完、智能解析等功能以便开发人员高效的进行 ASP.NET MVC 应用程序的开发。创建 ASP.NET MVC 应用程序后，系统会自动创建若干文件夹和文件，如图 17-7 所示。

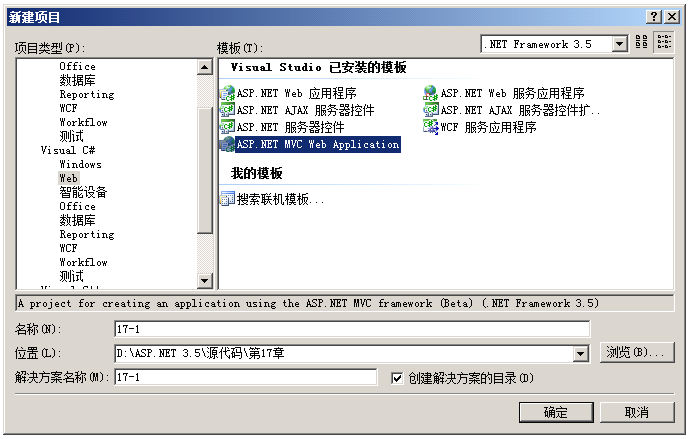

图 17-6　创建 ASP.NET MVC 应用程序

图 17-7　自动创建的文件

在自动创建的文件中，这些文件包括 ASP.NET MVC 应用程序中最重要的文件夹用于分层开发，这些文件夹分别为 Models、Views 和 Controllers，分别对应 ASP.NET MVC 开发模型的 Models（模型）、

Views（视图）、Controller（控制器），开发人员能够在相应的文件夹中创建文件进行 ASP.NET MVC 应用程序的开发。

17.2.3 ASP.NET MVC 应用程序的结构

在创建完成 ASP.NET MVC 应用程序，系统会默认创建一些文件夹，这些文件夹不仅包括对应 ASP.NET MVC 开发模型的 Models、Views 和 Controllers 文件夹，还包括配置文件 Web.config、Global.aspx 和 Default.aspx。

1．Default.aspx：页面驱动

Default.aspx 用于 ASP.NET MVC 应用程序程序的驱动，当用户执行相应的请求时，Default.aspx 能够驱动 ASP.NET MVC 应用程序页面的处理和生成，Default.aspx 页面代码如下所示。

```
<%@ Page
Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="_17_1._Default" %>
```

Default.aspx 页面代码同传统的 ASP.NET Web Form 基本相同，但 Default.aspx 只是用于 MVC 应用程序的驱动。Default.aspx 使用 IHttpHandler 类获取和发送 HTTP 请求，Default.aspx.cs 页面代码如下所示。

```
using System.Web;
using System.Web.Mvc;                                                    //使用 Mvc 命名空间
using System.Web.UI;
namespace _17_1
{
    public partial class _Default : Page
    {
        public void Page_Load(object sender, System.EventArgs e)
        {
            HttpContext.Current.RewritePath(Request.ApplicationPath);    //拦截虚拟目录根路径
            IHttpHandler httpHandler = new MvcHttpHandler();
            httpHandler.ProcessRequest(HttpContext.Current);
        }
    }
}
```

上述代码用于 ASP.NET MVC 应用程序的驱动。在 ASP.NET MVC 应用程序被运行时，会拦截虚拟目录的根路径将请求发送到 Controllers 实现。

2．Global.asax：全局配置文件

Global.asax 是全局配置文件，在 ASP.NET MVC 应用程序中的应用程序路径是通过 Global.asax 文件进行配置和实现的，Global.asax 页面代码如下所示。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;                                                    //使用 Mvc 命名空间
using System.Web.Routing;                                                //使用 Mvc 命名空间
namespace _17_1
{
    // Note: For instructions on enabling IIS6 or IIS7 classic mode,
```

```
    // visit http://go.microsoft.com/?LinkId=9394801
    public class MvcApplication : System.Web.HttpApplication
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.MapRoute(
                "Default",                                              //配置路由名称
                "{controller}/{action}/{id}",                          //配置访问规则
                new { controller = "Home", action = "Index", id = "" } //为访问规则配置默认值
            );                                                         //配置 URL 路由
        }
        protected void Application_Start()
        {
            RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

上述代码在应用程序运行后能够实现相应的 URL 映射，当用户请求一个页面时，该页面会在运行时启动并指定 ASP.NET MVC 应用程序中 URL 的映射以便将请求提交到 Controllers 进行相应的编程处理和页面呈现。

Global.asax 实现了伪静态的 URL 配置，例如当用户访问/home/guestbook/number 服务器路径时，Global.asax 通过 URLRouting 够实现服务器路径/home/guestbook/number 到 number.aspx 的映射。有关 URLRouting 的知识会在后面的小结中讲解。

注意：在 ASP.NET MVC 开发模型中，浏览器地址栏的 URL 并不能够被称为是伪静态，为了方便读者的理解可以暂时称为伪静态，但是最主要的是要理解访问的路径并不像传统的 Web 开发中那样是访问真实的某个文件。

3．Models、Views 和 Controllers 三层结构

Models、Views 和 Controllers 文件夹是 ASP.NET MVC 开发模型中最为重要的文件夹，虽然这里以文件夹的形式呈现在解决方案管理器中，其实并不能看作传统的文件夹。Models、Views 和 Controllers 分别用于存放 ASP.NET MVC 应用程序中 Models、Views 和 Controllers 的开发文件。在创建 ASP.NET MVC 应用程序后，系统会自行创建相应的文件，这里也包括 ASP.NET MVC 应用程序样例，如图 17-8 和图 17-9 所示。

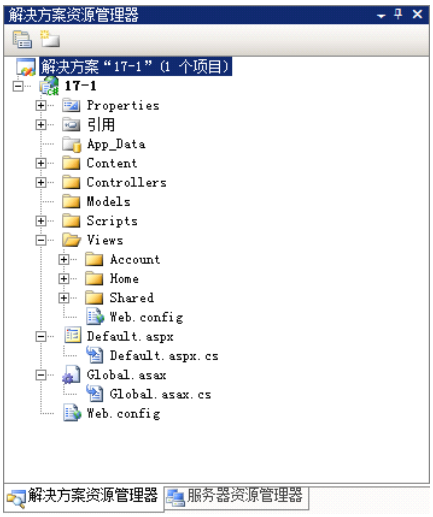

图 17-8　Views 视图文件夹

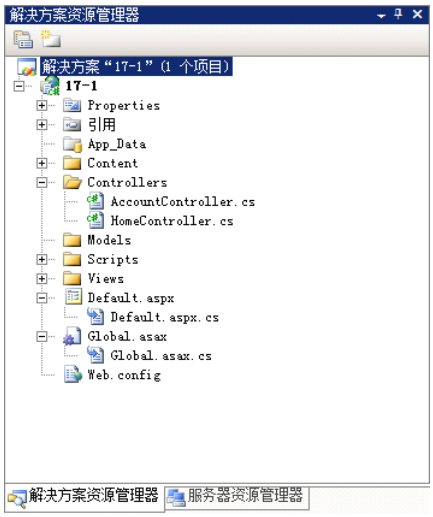

图 17-9　Controllers 控制器文件夹

正如图 17-8、17-9 所示，在样例中分别创建了若干 Controllers 控制器文件，以及 Views 页面文件。运行 ASP.NET MVC 应用程序后，用户的请求会发送到 Controllers 控制器中，Controllers 控制器接受用户的请求并通过编程实现 Views 页面文件的映射。

17.2.4　运行 ASP.NET MVC 应用程序

创建 ASP.NET MVC 应用程序后就能够直接运行 ASP.NET MVC 应用程序，默认的 ASP.NET MVC 应用程序已经提供了样例方便开发人员进行编程学习，单击【F5】运行 ASP.NET MVC 应用程序，运行后如图 17-10 所示。

在创建 ASP.NET MVC 应用程序后系统会创建样例，图 17-10 显式的就是 ASP.NET MVC 默认运行界面，单击旁边的【About Us】连接页面跳转到相应的页面，如图 17-11 所示。

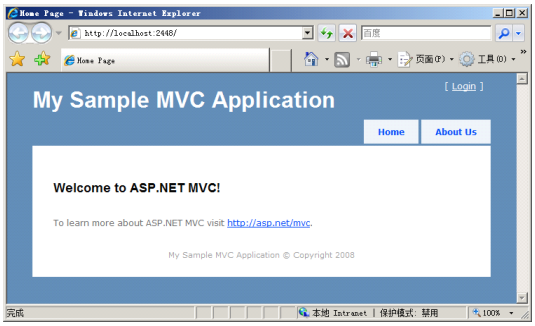

图 17-10　ASP.NET MVC 应用程序初始界面

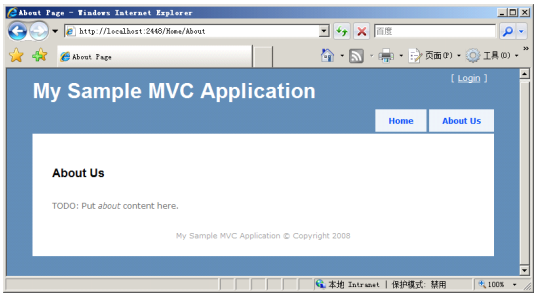

图 17-11　About 页面

当单击【About Us】链接后，页面会跳转到关于页面，页面 URL 为 http://localhost:2448/Home/About。在 ASP.NET MVC 应用程序中，URL 路径的请求方式与传统的 ASP.NET Web Form 应用程序不同，开发人员可以发现，在服务器文件中并没有/Home/About/index.aspx 文件也没有/Home/About/这个目录。

注意：在 ASP.NET MVC 应用程序中，这里再三强调，其 URL 并不是服务器中的某个文件而是一种地址映射。

在服务器中没有/Home/About/index.aspx 文件也没有/Home/About/这个目录，因为/Home/About 中所呈现的页面是通过 Controller 控制器和 Global.ascx 进行相应的文件的路径的映射的，关于地址映射的内

容会在后面的小结中详细讲解。

17.3 ASP.NET MVC 原理

运行了 ASP.NET MVC 应用程序后，就能够通过相应的地址访问不同的页面。在 ASP.NET MVC 应用程序中，应用程序中页面的 URL 并不是在服务器中实际存在的页面或目录而是访问了相应的方法，ASP.NET MVC 应用程序通过 Global.ascx 和 Controllers 实现了 URL 映射。

17.3.1 ASP.NET MVC 运行流程

在运行 ASP.NET MVC 应用程序后，会发现访问不同的 ASP.NET MVC 应用程序页面时，其 URL 路径并不会呈现相应的.aspx 后缀。同样当访问相应的 ASP.NET MVC 应用程序页面，在服务器中并不存在对应的页面。为了了解如何实现页面映射，就需要了解 ASP.NET MVC 应用程序的运行流程。

在 ASP.NET MVC 程序中，应用程序通过 Global.ascx 和 Controllers 实现了 URL 映射。当用户进行 ASP.NET MVC 程序的页面请求时，该请求首先会会被发送到 Controllers 控制器中，开发人员能够在控制器 Controllers 中创建相应的变量并将请求发送到 Views 视图中，Views 视图会使用在 Controllers 控制器中通过编程方式创建相应的变量并呈现页面在浏览器中。当用户在浏览器中对 Web 应用进行不同的页面请求时，该运行过程将会循环反复。

对于 Models 而言，Controller 通常情况下使用 Models 读取数据库。在 Models 中，Models 能够将传统的关系型数据库映射成面向对象的开发模型，开发人员能够使用面向对象的思想进行数据库的数据存取。Controllers 从 Model 中读取数据并存储在相应的变量中，如图 17-12 所示。

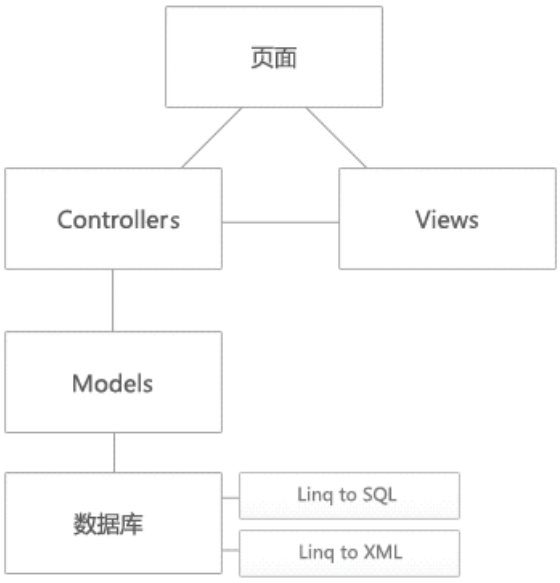

图 17-12 ASP.NET MVC 运行流程

正如图 17-12 所示，在用户进行页面请求时，首先这个请求会发送到 Controllers 中，Controllers 从 Models 中读取相应的数据并填充 Controllers 中的变量，Controllers 接受相应请求再将请求发送到 Views 中，Views 通过获取 Controllers 中的变量的值进行整合并生成相应的页面到用户浏览器中。

在 Models 中需要将数据库抽象成面向对象中的一个对象，开发人员能够使用 LINQ 进行数据库的抽象，这样就能够方便的将数据库中的数据抽象成相应的对象并通过对象的方法进行数据的存取和更新。

17.3.2 ASP.NET MVC 工作原理

正如上一节中讲解的 ASP.NET MVC 工作流程，在 ASP.NET MVC 应用程序中，系统默认创建了相应的文件夹进行不同层次的开发，在 ASP.NET MVC 应用程序的运行过程中，同样请求会发送到 Controllers 中，这样就对应了 ASP.NET MVC 应用程序中的 Controllers 文件夹，Controllers 只负责数据的读取和页面逻辑的处理。在 Controllers 读取数据时，需要通过 Models 中的 LINQ to SQL 从数据中读取相应的信息，读取数据完毕后，Controllers 再将数据和 Controller 整合并提交到 Views 视图中，整合后的页面将通过浏览器呈现在用户面前。

当用户访问 http://localhost:2448/Home/About 页面时，首先这个请求会发送到 Controllers 中，Controllers 通过 Global.ascx 文件中的路由设置进行相应的 URL 映射，Global.ascx 文件相应代码如下所示。

```
public static void RegisterRoutes(RouteCollection routes)                    //注册路由
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        "Default",
        "{controller}/{action}/{id}",
        new { controller = "Home", action = "Index", id = "" }               //配置路由
    );
}
```

上述代码中实现了映射操作，具体是如何实现可以先无需关心，首先需要看看 Controllers 文件夹内的文件，以及 Views 文件夹的文件，如图 17-13 所示。

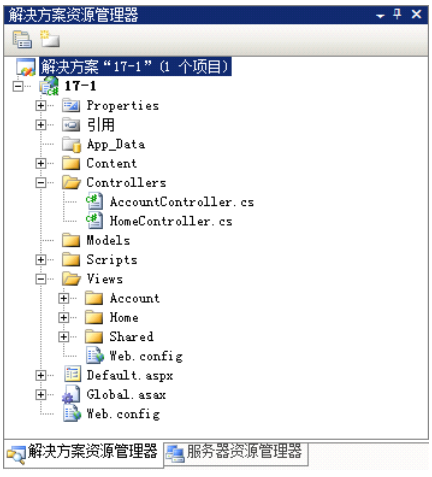

图 17-13 Controller 文件夹和 Views 文件夹

从图 17-13 中可以看出，在 Views 中包含 Home 文件夹，在 Home 文件夹中存在 About.aspx 和 Index.aspx 文件，而同样在 Controllers 文件夹中包含与 Home 文件夹同名的 HomeController.cs 文件。当用户访问 http://localhost:2448/Home/About 路径时，首先该路径请求会传送到 Controller 中。

注意：在 Controllers 文件夹中创建 HomeController.cs 文件同 Home 是同名文件，在 Controllers 中创建的文件，其文件名后的 Controller.cs 是不能更改的，所以 HomeController.cs 文件也可以看做是 Home 文件夹的同名文件。

在 Controller 中，Controller 通过 Global.ascx 文件和相应的编程实现路径的映射，示例代码如下所示。

```
[HandleError]
public class HomeController : Controller
{
    public ActionResult About()                                    //实现 About 页面
    {
        ViewData["Title"] = "About Page";
        return View();                                             //返回视图
    }
}
```

上述代码实现了 About 页面的页面呈现，在运行相应的方法后会返回一个 View，这里默认返回的是与 Home 的 About 方法同名的页面，这里是 about.aspx，about.aspx 页面代码如下所示。

```
<%@ Page
    Language="C#"
    MasterPageFile="~/Views/Shared/Site.Master"
    AutoEventWireup="true" CodeBehind="About.aspx.cs" Inherits="_17_1.Views.Home.About" %>
    <asp:Content ID="aboutContent" ContentPlaceHolderID="MainContent" runat="server">
        <h2>About Us</h2>
        <p>
            TODO: Put <em>about</em> content here.
        </p>
    </asp:Content>
```

将 about.aspx 页面中的文字进行相应的更改，示例代码如下所示。

```
<asp:Content ID="aboutContent" ContentPlaceHolderID="MainContent" runat="server">
    <h2>About Us</h2>
    <p>
        <span style=”color:red”>这是一个关于页面</span>
    </p>
</asp:Content>
```

运行 about.aspx 页面，运行后如图 17-14 所示。

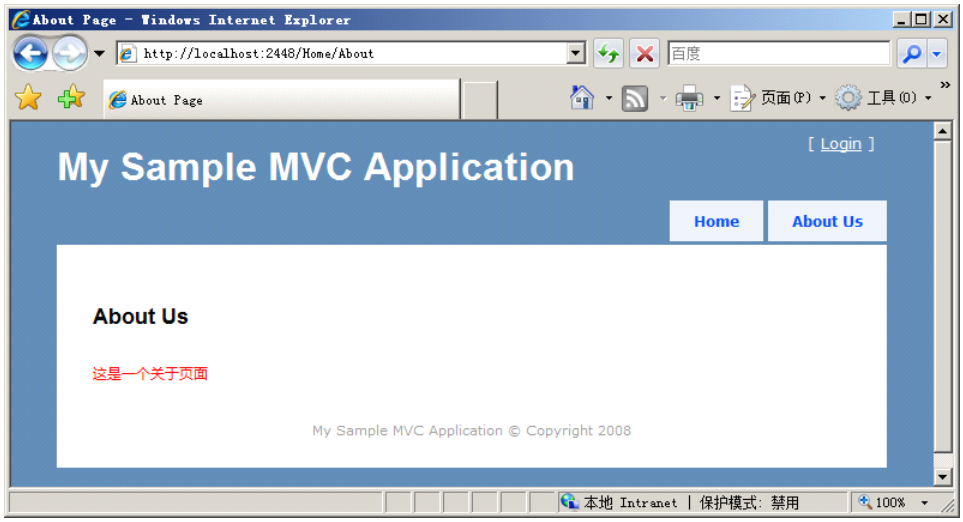

图 17-14　修改后的 About Us 页面

从上述代码可以看出，Controllers 与 Global.ascx 用于 URL 的映射，而 Views 用于页面的呈现。从这里可以看出，当用户访问 http://localhost:2448/Home/About 页面时，访问的并不是服务器中的 /Home/About 页面，而访问的是 Controllers 中的 HomeControllers 的 About 方法。

注意：ASP.NET MVC 应用程序中的 URL 路径访问的并不是一个页面，而是一个方法，例如访问 /Home/About 页面就是访问的是 HomeControllers 中的 About 方法，而访问/Account/Login 页面就是访问的是 AccountControllers 中的 Login 方法。

在 ASP.NET MVC 应用程序中，ASP.NET MVC 应用程序的对应关系如图 17-15 所示。

图 17-15　ASP.NET MVC 应用程序关系图

在 ASP.NET MVC 应用程序中，HomeController.cs 对应 Views 的 Home 文件夹，而其中的 Index 方法和 About 方法对应 Index.aspx 文件和 About.aspx 文件。

注意：在命名时，默认情况下 XXXController.cs 对应 Views 的 XXX 文件夹，而其中 XXXController.cs 中的 YYY()方法对应 XXX 文件夹中的 YYY.aspx，而访问路径为 XXX/YYY 是访问的是 XXXController.cs 中的 YYY()方法。

实现相应的 URL 映射需要通过修改 Global.ascx 文件进行实现，如何通过修改 Global.ascx 文件进行不同的 URL 映射将在后面的小结中讲解。

17.4　ASP.NET MVC 开发

在了解了 ASP.NET MVC 工作原理和工作流程，以及 ASP.NET MVC 中的 URL 映射基础原理，就能够进行 ASP.NET MVC 应用程序的开发，在进行 ASP.NET MVC 应用程序开发的过程中可以深入的了解 ASP.NET MVC 应用程序模型和 URL 映射原理。

17.4.1　创建 ASP.NET MVC 页面

ASP.NET MVC 应用程序包括 MVC 三个部分，其中 Models 是用于进行数据库抽象，Views 是用于进行视图的呈现而 Controllers 是用于控制器和逻辑处理，在创建 ASP.NET MVC 应用程序时，可以为 ASP.NET MVC 应用程序分别创建相应的文件。首先在 Views 文件夹中创建一个文件夹，这里创建一个 Beta 文件夹。创建文件夹后单击 Beta 文件夹，右击文件夹，在下拉菜单中选择【添加】选项，在【添加】选项中单击【新建项】选项，单击后系统会弹出对话框用于 View 文件的创建，如图 17-16 所示。

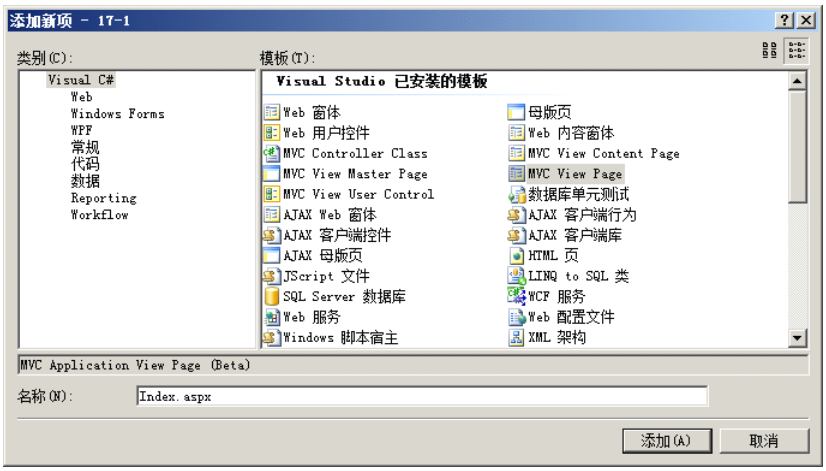

图 17-16　创建 View 文件

在 Views 中可以创建 MVC View Page 用于 Views 文件的创建，从而用于在 ASP.NET MVC 应用程序中呈现相应页的视图，在 Index.aspx 中可以编写相应的代码用于视图的呈现，Index.aspx 页面代码如下所示。

```
<%@ Page
Language="C#"
AutoEventWireup="true" CodeBehind="Beta.aspx.cs" Inherits="_17_1.Views.Beta.Beta" %>
    <h2>About Us</h2>
        <p>
        <span style="color:red">这是一个测试页面</span>
        </p>
```

Index.aspx 页面用于视图的呈现，在一个传统的 ASP.NET 应用程序窗体中，ASP.NET 应用程序窗体是派生自 System.Web.UI.Page 的，而 ASP.NET MVC 应用程序页面代码需要派生自 ViewPage，Index.aspx 的 cs 文件代码在创建时与传统的 ASP.NET 应用程序窗体不同，示例页面代码如下所示。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;                                          //使用 MVC 命名空间
namespace _17_1.Views.Beta
{
    public partial class Index : ViewPage                      //派生自 ViewPage
    {
    }
}
```

在完成 Beta.aspx 的创建后，在 ASP.NET MVC 应用程序开发模型中还需要创建 Controllers 用于接受用户请求和 Beta.aspx 页面同名的方法实现。单击 Controllers 文件夹，右击 Controllers 文件夹，在下拉菜单中选择【添加】选项，在【添加】选项中单击【新建项】选项。这里可以创建一个同名的类文件，如图 17-17 所示。

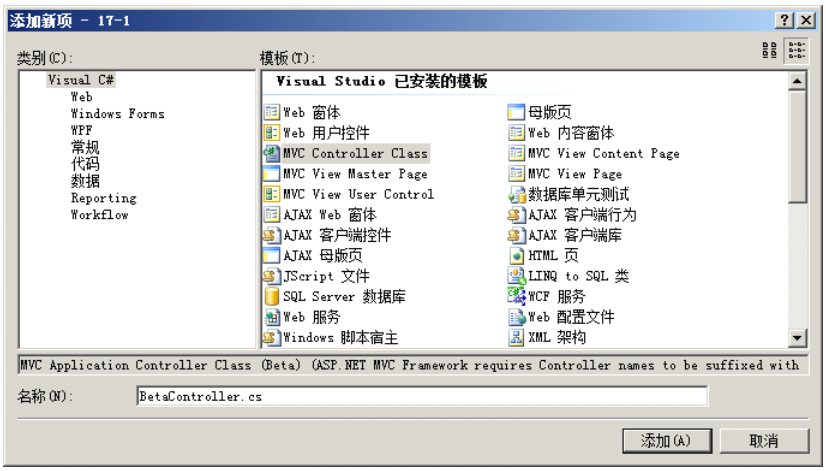

图 17-17　创建 Controllers 文件

创建 Controllers 类文件时，创面的类文件的名称必须为 Views 文件夹中相应的视图文件夹的名称加上 Controllers.cs，正如图 17-17 所示，如创建的是“Beta”文件夹，在创建 Controllers 时必须创建 BetaControllers.cs，在创建相应的类文件后才能够拦截相应的 URL 并进行地址映射，创建后的 Controllers 类文件代码如下所示。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;                                          //使用 MVC 命名空间
using System.Web.Mvc.Ajax;                                     //使用 MVC 命名空间
namespace _17_1.Controllers
{
    [HandleError]
    public class BetaController : Controller
    {
        public ActionResult Index()                            //实现 Index 方法
        {
            return View();                                     //返回 Index 视图
        }
    }
}
```

这里值得注意的是，仅仅创建一个 Index.aspx 页面并不能够在浏览器中浏览 Index.aspx 页面，必须在相应的 Controllers 类文件中实现与 Index.aspx 页面文件同名的方法 Index()才能够实现 Index.aspx 页面的访问。Views 中的 Index.aspx 页面能够使用 Controllers 类文件中的 Index 方法中的变量进行数据呈现。单击【F5】运行页面，运行后如图 17-18 所示。

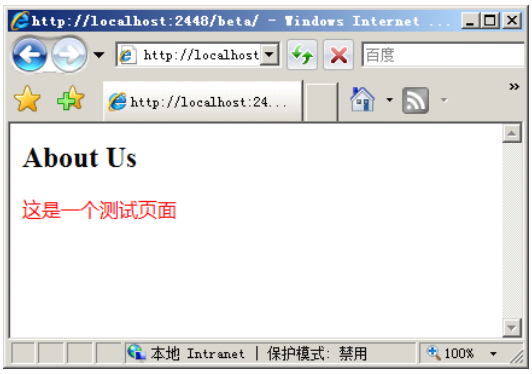

图 17-18　MVC 页面运行

这里讲解了如何手动创建 ASP.NET MVC 页面。在某些版本的 Visual Studio 中，安装了 ASP.NET MVC 开发包应用程序后，可能不会存在 MVC 文件的创建，这时只能通过创建 ASP.NET Web Form 再通过编码实现。

如果希望能够创建 ASP.NET MVC 模板而不使用手动创建可以在 C:\Program Files\Microsoft ASP.NET\ASP.NET MVC Beta\Temp 目录下将压缩包拷贝到相应的 Visual Studio 安装目录 X:\Microsoft Visual Studio 9.0\Common7\IDE\ItemTemplates\CSharp\Web\2052\中，拷贝后在开始菜单中选择“运行”，在窗口中输入 cmd，就会弹出一个黑色的命令行窗口，在命令行输入 cd X:\Microsoft Visual Studio 9.0\Common7\IDE\ItemTemplates\CSharp\Web\2052\进入目录，输入 devenv.exe /setup 进行模板的安装，安装完成后就能够在添加新项中选择 MVC 应用程序模板。

17.4.2　ASP.NET MVC 数据呈现（ViewData）

在 ASP.NET MVC 应用程序中，Controllers 负责数据的读取而 Views 负责界面的呈现，在界面的呈现中 Views 通常不进行数据的读取和逻辑运算，数据的读取和逻辑运算都交付给 Controllers 负责。为了能够方便的将 Controllers 与 Views 进行整合并在 Views 中呈现 Controllers 中的变量，可以使用 ViewData 整合 Controllers 与 Views 从而进行数据读取和显示。

在 ASP.NET MVC 应用程序的 Views 中，其值并不是固定的，而是通过 Controllers 传递过来的，在 Controllers 类文件中的页面实现代码中，可以需要使用 ViewData 进行值的传递，BetaControllers.cs 中 Index.aspx 实现的 Index()的方法示例代码如下所示。

```
[HandleError]
public class BetaController : Controller
{
    public ActionResult Index()                                    //实现 Index 方法
    {
        ViewData["beta"] = "这是一个 ViewData 字符串";              //使用 ViewData
        return View();                                             //返回视图
    }
}
```

上述代码使用 ViewData 存储数据，ViewData 的声明和赋值方式与 Session 对象相同，直接通过编写 ViewData[键值的名称]=XXX 进行相应的键值的赋值。如果需要在页面中进行相应的值的呈现，只需要输出 ViewData[键值的名称]即可。

在 ASP.NET MVC 应用程序中，字符输出都需要呈现在 Views 视图中，在 Controllers 中进行 ViewData

变量的赋值，就需要在 Views 中输出相应的变量，BetaControllers.cs 中的 Index()方法实现的是 Index.aspx 页面，在 Index.aspx 可以使用 ViewData["beta"]变量，示例代码如下所示。

```
<h2>About Us</h2>
<p>
    <span style="color:Red">这是一个测试页面</span><br/>
    <span style="color:Green"><%=ViewData["beta"] %></span>
</p>
```

上述代码中在运行后会输出 ViewData["beta"]变量中存储的值，运行后如图 17-19 所示。

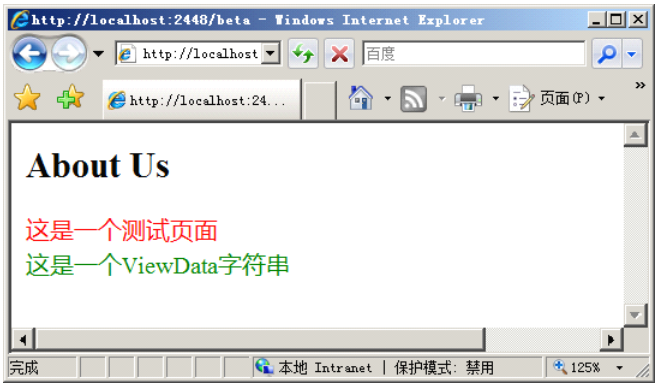

图 17-19　输出 ViewData

ViewData 不仅可以为某个具体的值，ViewData 还可以是一个泛型变量，示例代码如下所示。

```
[HandleError]
public class BetaController : Controller
{
    public ActionResult Index()
    {
        List<string> str = new List<string>();                    //创建泛型变量
        str.Add("str 字符串 1<hr/>");                              //添加成员
        str.Add("str 字符串 2<hr/>");                              //添加成员
        str.Add("str 字符串 3<hr/>");                              //添加成员
        str.Add("str 字符串 4<hr/>");                              //添加成员
        ViewData["beta"] = str;                                   //赋值 ViewData
        return View();                                            //返回视图
    }
}
```

在为 ViewData 赋值泛型变量后，在相应的 View 页面中也可以输出 ViewData 的值，示例代码如下所示。

```
<h2>About Us</h2>
<p>
    <span style="color:Red">这是一个测试页面</span><br/>
    <% foreach(string str in ViewData["beta"] as List<string>) %>
    <% = str%>
</p>
```

上述代码通过使用 foreach 进行 ViewData 变量中相应健值的值的遍历，运行后如图 17-20 所示。

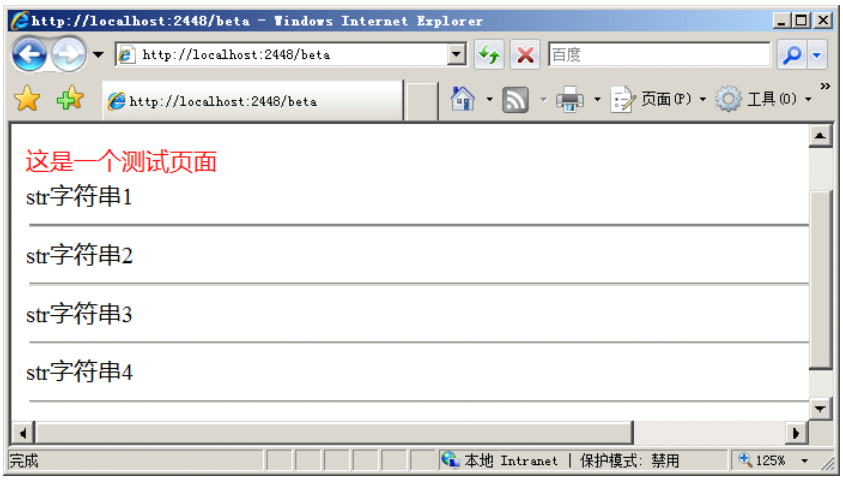

图 17-20　遍历 ViewData 变量的值

使用 List 类能够将数据库中的数据存放在泛型列表类中，开发人员能够将数据库中的数据遍历并存放在 Controllers 类文件中的页面实现的类的 ViewData 变量中，当需要遍历数据进行呈现时，例如新闻列表或者是论坛列表等，可以通过遍历 ViewData 变量的值遍历输出数据库中的数据。

17.4.3　ASP.NET MVC 跨页数据呈现（TempData）

ASP.NET MVC TempData 同 ASP.NET MVC ViewData 一样，是在 Controllers 中声明的变量以便在 Views 中进行调用，示例代码如下所示。

```
[HandleError]
public class BetaController : Controller
{
    public ActionResult Index()
    {
        TempData["beta"] = "TempData 字符串";
        return View();
    }
}
```

上述代码在 Controllers 中声明了 TempData，在 Views 中的相应页面可以使用此 TempData 进行变量的输出，示例代码如下所示。

```
<%@ Page
Language="C#"
AutoEventWireup="true" CodeBehind="Beta.aspx.cs" Inherits="_17_1.Views.Beta.Index" %>
    <h2>About Us</h2>
    <p>
        <%=TempData["beta"] %>
    </p>
```

上述代码呈现了 TempData 变量的值，运行后如图 17-21 所示。

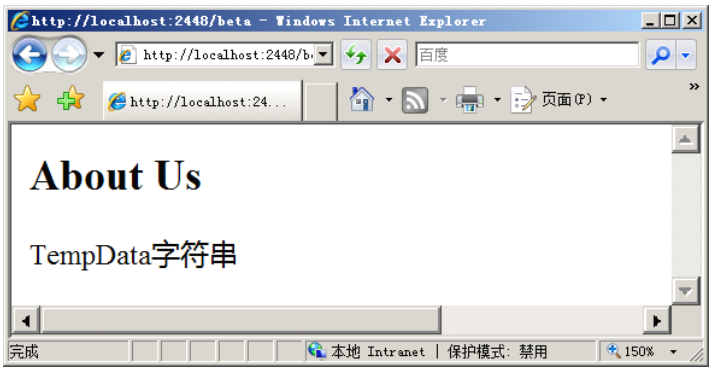

图 17-21　显示 TempData 变量

在数据呈现上，TempData 变量同 ASP.NET MVC ViewData 基本相同，但是 TempData 能够在跳转中保存值。当用户访问一个页面时，该页面的 Controllers 中包含 TempData 变量。当这个页面通过 Redirect 跳转到另一个页面时，另一个页面能够使用跳转页面的 TempData 变量。在跳转页面中，在跳转前可以编写 TempData 变量保存相应的值，示例代码如下所示。

```
[HandleError]
public class BetaController : Controller
{
    public ActionResult Index()
    {
        TempData["Index"] = "这是跳转页面的字符串哦..";                    //编写 TempData
        Response.Redirect("/Beta/Get");                                    //页面跳转
        return View();                                                     //返回视图
    }
}
```

上述忽代码编写了一个 TempData 变量并将页面跳转到 Get.aspx，这里在 Beta 文件夹下创建一个 Get.aspx 页面读取相应的 TempData 变量的值。创建完成后，编写 HTML 代码如下所示。

```
<%@ Page
Language="C#"
AutoEventWireup="true" CodeBehind="Get.aspx.cs" Inherits="_17_1.Views.Beta.Get" %>
    <h2>接受传递的参数</h2>
        <p>
            <%=TempData["Index"] %>
        </p>
```

编写了页面代码后还不能对页面进行访问，由于 MVC 编程模型中路径是访问的 Controller 中的方法，所以还需要在 Controller 中实现 Get 页面的方法，示例代码如下所示。

```
        public ActionResult Get()
        {
            return View();                                                 //返回默认视图
        }
```

上述代码返回默认视图，当不给 View 方法进行参数传递，将会返回与方法同名的.aspx 页面文件。这里没有对 View 方法传递参数，则返回的是 Get.aspx 页面视图，当用户访问/Beta/路径时，代码会创建一个 TempData 变量并跳转到/Beta/Get 路径，在/Beta/Get 路径相应的文件中可以获取跳转前的 TempData 变量的值，运行后如图 17-22 所示。

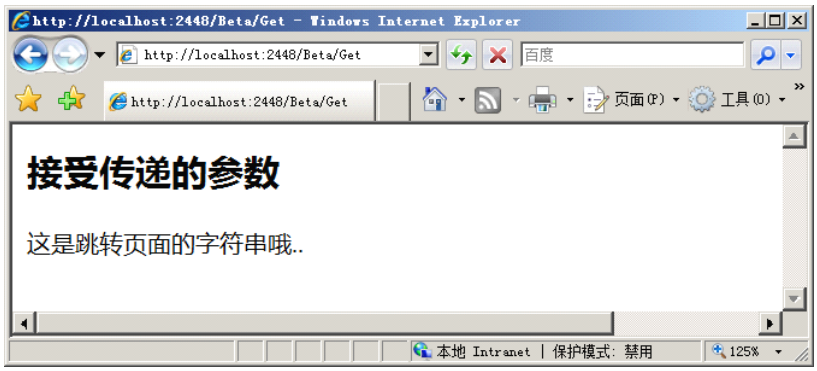

图 17-22　接受 TempData 变量的值

在 Get.aspx 页面相应的实现代码中并没有声明任何的 TempData 变量，而是在跳转前的页面中声明的 TempData 变量，与 ViewData 相比跳转后的页面能够获取 TempData 变量的值而不能获取 ViewData 的值，这在某些应用场合如抛出异常等情况下非常适用。

注意：TempData 变量在跳转中能够跨页面进行数据读取，但是跨页面跳转后 TempData 变量只能呈现一次。简单的说就是跳转的第一次能过获取跳转前页面的 TempData 变量的值，而再次操作时就无法使用跳转前页面的 TempData 变量值。

17.4.4　ASP.NET MVC 页面重定向

在 ASP.NET Web Form 中，可以通过 Response.Redirect(页面)的方法进行页面的重定向，在 ASP.NET MVC 编程模型中，也可以通过 Response.Redirect(页面)的方法进行页面重定向。不仅如此，ASP.NET MVC 编程模型还支持多种页面重定向方法，传统的页面重定向可以使用 Response.Redirect(页面)方法，示例代码如下所示。

```
public ActionResult Index()
{
    Response.Redirect("/Beta/Get");                    // Response.Redirect(页面)
    return View();                                     //返回视图
}
```

在 MVC 应用程序框架中，开发人员不仅能够使用传统的 Response.Redirect(页面)的方法进行页面重定向，MVC 还支持直接返回重定向参数进行重定向，示例代码如下所示。

```
public ActionResult Index()
{
    return Redirect("/Beta/Get");                      //返回重定向参数
}
```

上述代码通过使用重定向参数进行页面重定向。由于 MVC 编程模型是通过 Controllers 进行页面的呈现，而 MVC 编程模型同样是基于面向对象的，当用户访问某个路径实际上是访问相应的 Controllers 的方法。对于相同的页面而言，开发人员能够使用 MVC 编程模型中提供的 RedirectToAction 进行页面重定向，示例代码如下所示。

```
public ActionResult Index()
{
    return RedirectToAction("Get");                    //通过方法重定向页面
}
public ActionResult Get()
{
```

```
        return View();                                          //返回默认视图
    }
```

上述代码只能使用在同一个 Controllers 中，当需要跳转到不同的 Controllers 中时，同样可以通过此方法进行页面重定向，示例代码如下所示。

```
    public ActionResult Index()
    {
        return RedirectToAction("Login","Account");             //跨 Controllers 跳转
    }
```

上述代码同样使用 RedirectToAction 方法进行重定向，重载的 RedirectToAction 方法前一个参数是相应的页面的方法，而后一个参数是相应的页面所在的 Controllers。

17.4.5 ASP.NET MVC URL 路由（URLRouting）

在 ASP.NET MVC 编程模型中，除了 M、V、C 三个模块，MVC 编程模型中最为重要的就是 ASP.NET MVC URLRouting 的概念。运行 ASP.NET MVC 应用程序，其 URL 如图 17-23 所示。

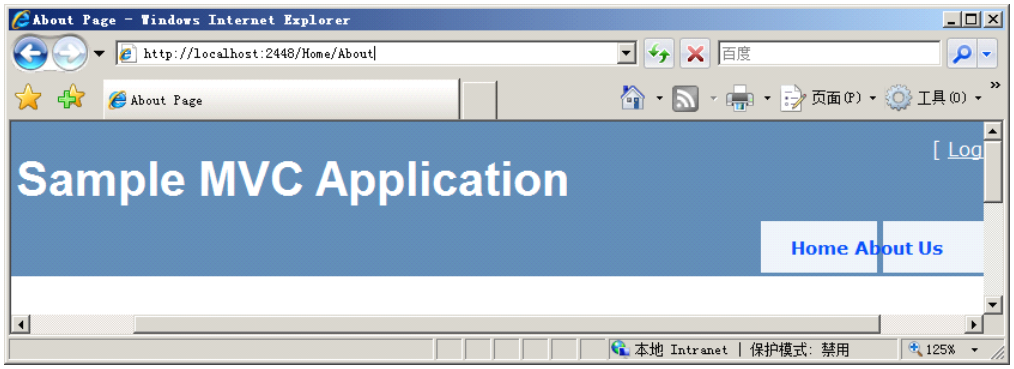

图 17-23　ASP.NET MVC 应用程序 URL 路径

从途中可以看出 URL 路径为 http://localhost:2448/Home/About，从前面的小结中可以知道，当访问了该路径时，实际上是访问了 HomeControllers.cs 中的 About 方法，而 About 方法通过 About.aspx 页面进行视图呈现，视图中所使用的数据是在 About 方法中声明的 ViewData，这样才组成了一个 MVC 应用程序页面。

ASP.NET MVC 应用程序中实现上述过程，即将/Home/About 映射到相应的 Controllers 的相应方法中就必须使用到 ASP.NET MVC URLRouting，ASP.NET MVC URLRouting 定义在 Global.ascx 文件中，Global.ascx 文件代码如下所示。

```
namespace _17_1
{
    public class MvcApplication : System.Web.HttpApplication
    {
        public static void RegisterRoutes(RouteCollection routes)          //注册路由
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");              //注册路径
            routes.MapRoute(
                "Default",                                                 //设置默认名称
                "{controller}/{action}/{id}",                              //设置路由规则
                new { controller = "Home", action = "Index", id = "" }     //实现默认规则
            );
```

```
        }
        protected void Application_Start()                              //应用程序执行
        {
            RegisterRoutes(RouteTable.Routes);                          //实现方法
        }
    }
}
```

上述代码通过 URLRouting 实现了 URL 地址的映射，在 Global.ascx 中，最为重要的是 RegisterRoutes 方法，该方法实现了相应映射规则，示例代码如下所示。

```
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");           //注册路径
            routes.MapRoute(
                "Default",                                              //设置默认名称
                "{controller}/{action}/{id}",                           //设置路由规则
                new { controller = "Home", action = "Index", id = "" }  //实现路由规则
            );
        }
```

上述代码中使用了 RouteCollection 对象的 MapRoute 进行地址配置，其中为 ASP.NET MVC 应用程序 URL 的配置的规则为"{controller}/{action}/{id}"，这也就是说其映射会按照 controller 名称、方法名称和 ID 进行映射，所以/Home/About 路径就映射到了 HomeControllers.cs 中的 About 方法。在了解了基本的 URLRouting 实现 URL 地址映射后，开发人员能够修改相应的映射规则进行更改，更改后的规则如下所示。

```
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.MapRoute(
                "Default",                                              //设置默认名称
                "{controller}/{action}.html/{id}",                      //修改规则
                new { controller = "Home", action = "Index", id = "" }  //实现默认规则
            );
        }
```

上述代码在相应的规则中进行了修改，修改规则是访问相应的 Controllers 中的方法名称加.html 进行页面访问，这样 http://localhost:2448/Home/About 就不能够进行访问，因为 URL Routing 规则进行了更改。如果要访问相应的页面，则必须访问 http://localhost:2448/Home/About.html 进行页面访问，运行后如图 17-24 所示。

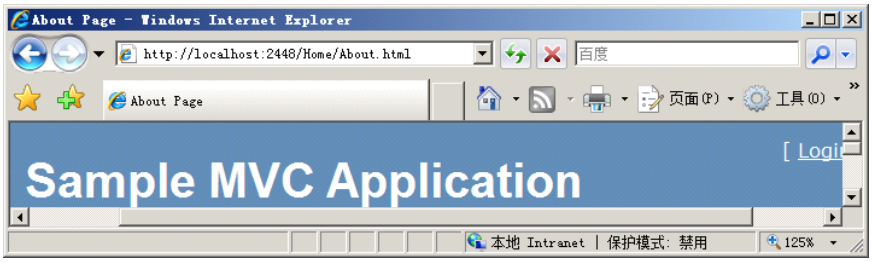

图 17-24　修改 URL Routing

正如图 17-24 所示，在访问页面时，原有的页面再不能够进行访问，而必须通过访问/Home/About.html

进行页面访问。

17.4.6 ASP.NET MVC 控件辅助工具（Helper）

在 ASP.NET MVC 开发模型中，由于将页面进行分层开发和呈现，开发人员在视图开发中通常是不推荐使用服务器控件的，因为在 ASP.NET MVC 页面是派生自 ViewPage 而 ASP.NET WebForm 是派生自 System.Web.UI.Page 的，同样为了规范 ASP.NET MVC 开发模型中页面的呈现和运行，使用服务器控件也不是最好的选择。为了能够方便的呈现控件和进行URL操作，ASP.NET MVC 开发模型提供了 Helper 进行控件的呈现和 URL 操作，Helper 包括 HtmlHelper 和 UrlHelper。

1．HTML 辅助工具（HtmlHelper）

由于在 ASP.NET MVC 开发模型中不推荐使用服务器控件，这就会提高 ASP.NET 页面编程的复杂性，使用 HtmlHelper 能够减少相应的编程复杂性。使用 HtmlHelper 能够创建 HTML 控件并进行控件编程，在 MVC 编程模型中，其执行过程很像传统的 ASP 的执行过程。使用 HtmlHelper 创建 HTML 控件的代码如下所示。

```
<h2>HtmlHelper</h2>
<p>
    请输入用户名:<% =Html.TextBox("Name") %>                    //使用 TextBox
<br/>
    请输入密码:<% =Html.Password("Name") %>                     //使用 Password
<br/>
    <input id="Submit1" type="submit" value="submit" />
</p>
```

上述代码通过 HtmlHelper 创建了 HTML 控件，HtmlHelper 方法创建控件只能够在 Views 中使用而不能在 Controllers 中使用。上述代码运行后如图 17-25 所示。

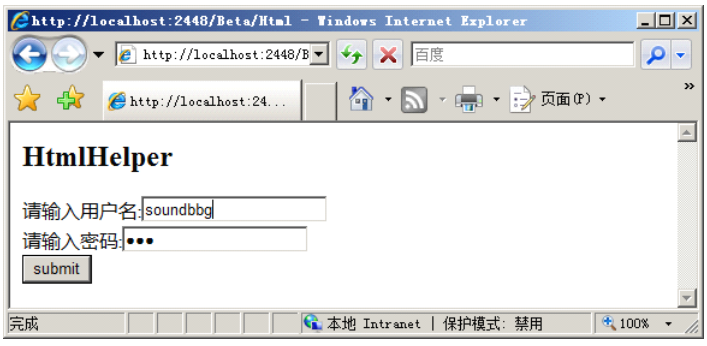

图 17-25　HtmlHelper 创建的 HTML 控件

注意：这里的 TextBox 控件和 Password 控件并不是 ASP.NET 控件，TextBox 控件和 Password 控件分别生成的是 HTML 控件。

2．URL 辅助工具（UrlHelper）

UrlHelper 在 MVC 开发框架中比较简单，UrlHelper 是用来呈现相应的 URL 路径的，UrlHelper 使用的示例代码如下所示。

```
<h2>HtmlHelper</h2>
 <p>
    <%=Url.Action("Index","Beta") %>
 </p>
```

上述代码通过使用 UrlHelper 的 Action 方法进行 URL 的呈现，在 Action 方法中，其参数分别为方法名和 Controller，上述代码中用于显示 BetaController 中的 Index 页面 URL，运行后如图 17-26 所示。

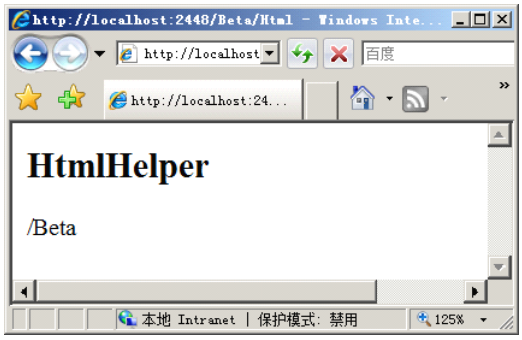

图 17-26　UrlHelper

17.4.7　ASP.NET MVC 表单传值

ASP.NET 的运行模型很像传统的 ASP，在 ASP.NET MVC 开发模型中，由于无法使用 runat="server" 进行表单传值，开发人员只能够自己编写表单进行传值。进行表单传值有两种方法，一种是编写 from 进行表单传值，一种是通过 HtmlHelper 进行表单生成和传值。编写 form 的表单传值方法示例代码如下所示。

```
<h2>HtmlHelper</h2>
<p>
<form id="form1" method="post" action="<%=Html.AttributeEncode(Url.Action("Html","Beta")) %>">
    请输入用户名:<% =Html.TextBox("Name") %>
    <br/>
    请输入密码:<% =Html.Password("Name") %>
    <br/>
    <input id="Submit1" type="submit" value="submit" />
    <%=Url.Action("Index","Beta") %>
    <%= ViewData["p"] %>
</form></p>
```

上述代码通过传统的方法在 HTML 中编写 from 标签，而 form 标签的属性 action 需要通过使用 HtmlHelper 进行指定，这里指定的是 BetaControllers 中的 Html 方法进行参数传递处理。在 Html 方法中，可以通过编程实现相应的表单传值处理，示例代码如下所示。

```
public ActionResult Html()
{
    if (Request.HttpMethod != "POST")                    //判断传递方法是否为 Post
    {
        ViewData["p"] = "表单还没被传递";                  //填充相应的 ViewData
    }
    else
    {
        ViewData["p"] = "表单已被传递";                    //填充相应的 ViewData
    }
    return View();                                       //返回视图
}
```

上述代码首先会判断传递的方法是否为 POST，如果为 POST，说明表单已经传递，否则表单还没

有传递，上述代码运行后如图 17-27 和图 17-28 所示。

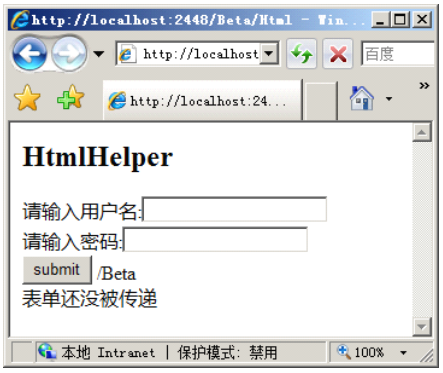

图 17-27　表单没传递

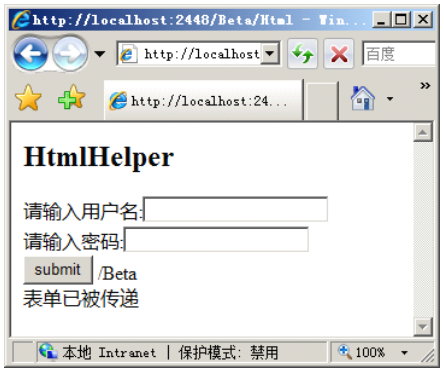

图 17-28　表单已传递

当用户单击按钮控件时，就会进行表单的传递。在 Html 方法中，通过编程进行了表单传递后的操作。在 MVC 编程中，可以通过 HttpMethod 判断表单是否传递，如果传递表单，则可以通过编程的方法进行数据判断和数据操作等其他操作。在 ASP.NET MVC 中，还可以使用 HttpHelper 进行表单传值，示例代码如下所示。

```
<h2>HtmlHelper</h2>
<p>
<% using (Html.BeginForm("Html", "Beta"))
    { %>
     请输入用户名:<% =Html.TextBox("Name")%>
     <br/>
     请输入密码:<% =Html.Password("Name")%>
     <br/>
     <input id="Submit1" type="submit" value="submit" />
     <%=Url.Action("Index", "Beta")%>
     <br/>
     <%= ViewData["p"]%>
<%} %>
</p>
```

在表单创建时，HttpHelper 的 BeginForm 方法能够创建一个 Form 进行表单生成，BeginForm 方法包括两个参数，第一个参数为方法，第二个参数为 Controllers。当运行该页面时，BeginForm 方法能够创建一个 Form 进行表单传值。

注意：使用 BeginForm 方法创建表单，而表单的结束并不是使用 EndForm，使用 BeginForm 方法创建需要使用 using{}的方法进行 Form 中内容容器。

当 ASP.NET MVC 应用程序能够进行表单传值后，就能够通过 HttpMethod 进行判断，如果用户进行了提交，开发人员能够通过编程实现数据更新的插入和删除等操作。在 ASP.NET MVC 应用程序中，其数据通常使用 LINQ 进行描述和操作，关于 LINQ 的知识会在后面专门讲解。

17.5　小结

本章讲解了 ASP.NET MVC 开发模型，以及工作原理，在创建 ASP.NET MVC 应用程序时，系统会自行创建若干文件和文件夹。ASP.NET MVC 开发模型和 ASP.NET Web Form 极不相同，所以创建的文

件夹和文件也不相同，要了解 ASP.NET MVC 开发模型就首先需要了解这些文件和文件夹的作用。本章还讲解了 ASP.NET MVC 的工作原理和工作流程，包括 ASP.NET MVC 中的 Controllers、Model 以及 Views 是如何形成一个完整的页面呈现在客户端浏览器中。本章还包括：

- 安装 ASP.NET MVC：讲解了如何在 Visual Studio 2008 中安装 ASP.NET MVC 应用程序开发包进行 ASP.NET MVC 应用程序开发。
- 新建一个 MVC 应用程序：讲解了如何创建一个新的 ASP.NET MVC 进行应用程序开发。
- ASP.NET MVC 应用程序的结构：讲解了 ASP.NET MVC 应用程序的基本结构，以及 ASP.NET MVC 中 M、V、C 的概念。
- 创建 ASP.NET MVC 页面：讲解了如何创建 ASP.NET MVC 页面。
- ASP.NET MVC ViewData：讲解了 ASP.NET MVC 中 ViewData 的作用和使用方法。
- ASP.NET MVC TempData：讲解了 ASP.NET MVC 中 TempData 的作用和使用方法。
- ASP.NET MVC 页面重定向：讲解了 ASP.NET MVC 中页面重定向的方法。
- ASP.NET MVC URLRouting：讲解了什么是 ASP.NET MVC URLRouting，以及 URLRouting 基本原理。
- ASP.NET MVC Helper：讲解了基本的 ASP.NET MVC 中 HtmlHelper，以及 UrlHelper 的作用。
- ASP.NET MVC 表单传值：讲解了如何使用 ASP.NET MVC 应用程序的 HtmlHelper，以及传统的 HTML 方式进行表单传值。

在 ASP.NET MVC 应用程序中，使用数据操作通常是使用 LINQ 进行数据操作的，当用户提交了表单，开发人员能够通过判断进行数据的相应操作，开发人员能够使用 LINQ 进行数据操作，有关 LINQ 的知识将在后面的章节讲解。