ASP.NET MVC 项目开发教程

开发环境：Visual Studio 2013

项目一：Hello World

任务一：ASP.NET MVC 项目的创建

一、任务实施

1. 创建MVC项目

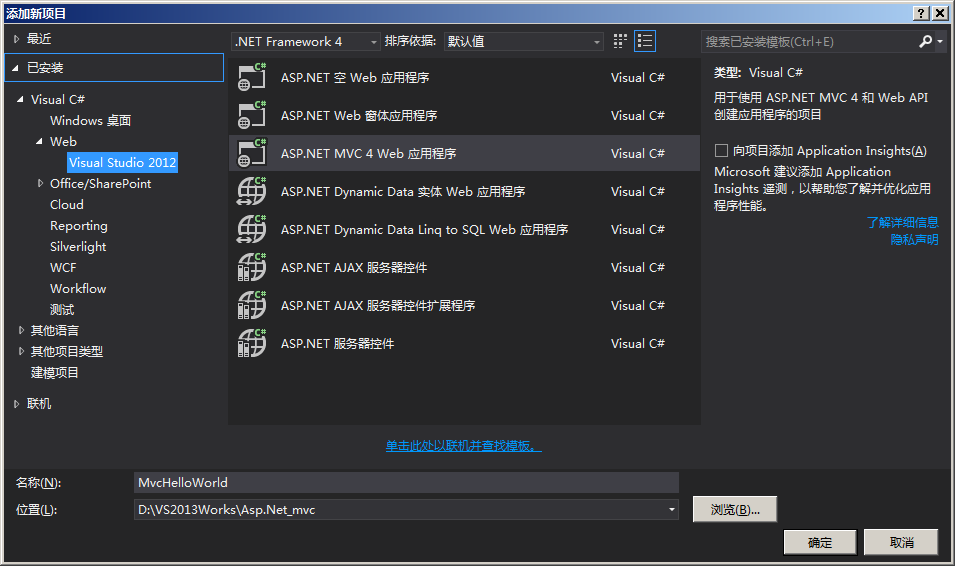


图1.1

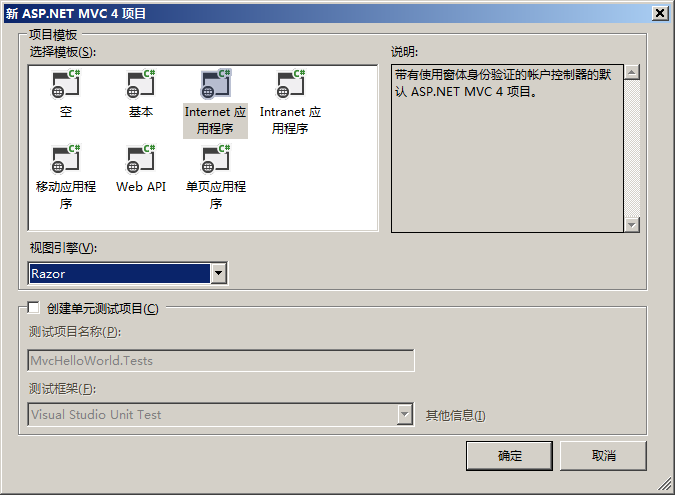


图1.2

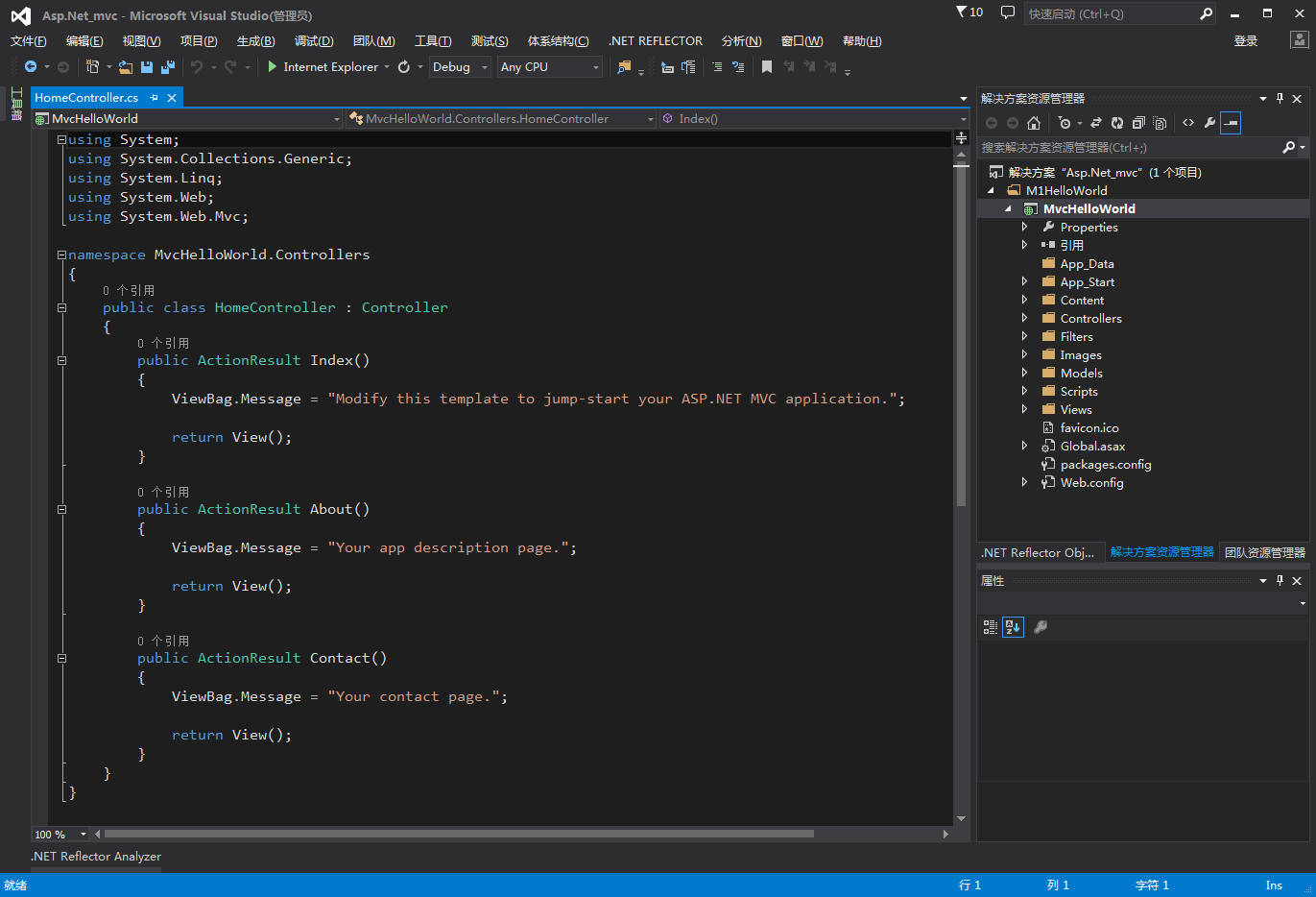


图1.3

2. 运行 Web 应用 程序

从“调试”菜单中选择“启动调试”，或者直接按“F5”键。

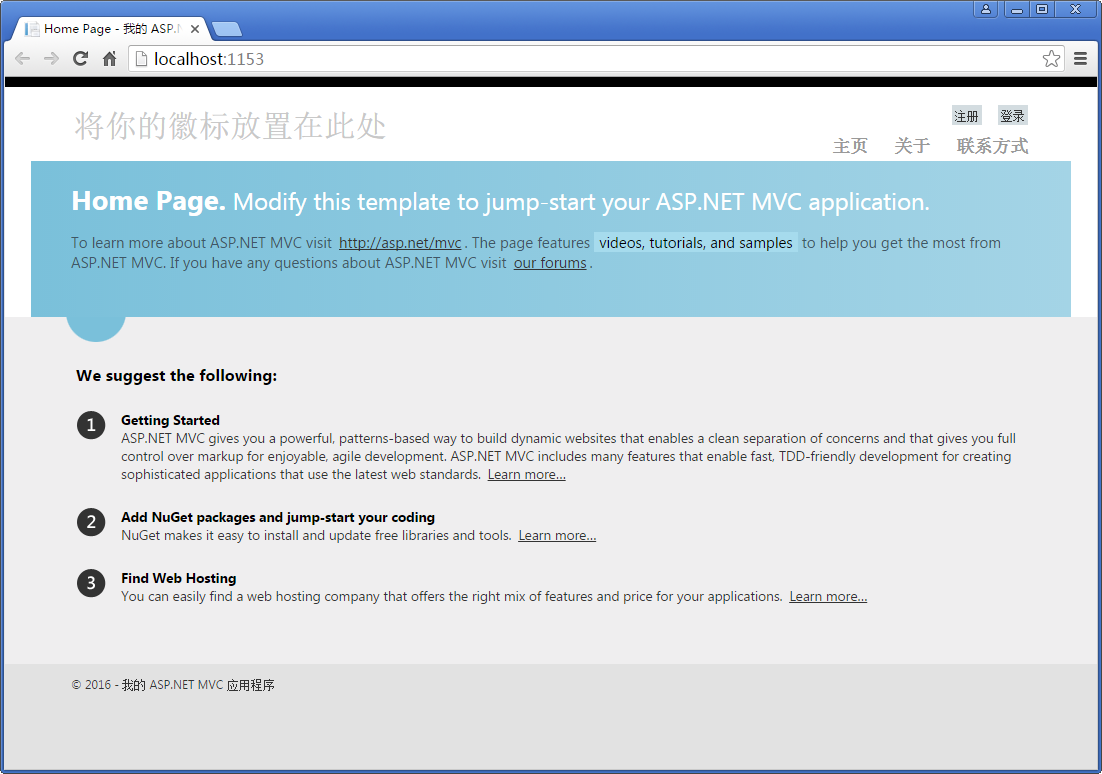


图1.4

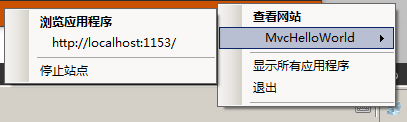


图1.5

二、相关知识

1. MVC 模式

应用程序所做的事：输入、输出、数据处理、数据存储。

MVC（Model View Controller）是模型（Model）－视图（View）－控制器（Controller）的缩写。

它强制性地使应用程序的输入、处理和输出分开。

图1.6

视图：用户看到并与这交互的界面。对Web应用来说，它指由HTML元素组成的界面。开发中表现为一些动态生成HTML页面的模板。

模型：企业数据和业务规则，在MVC的3个部件中拥有最多的处理任务。开发中表现为描述总是空间的一些类，类中封装了数据及与操作这些数据和执行特定业务逻辑相关的代码。模型就像是一个使用了某个工具的数据访问层（Data Access Layer）。

控制器：接收用户的输入并调用模型和视图去完成用户的需求。当单击Web页面中的超链接和发送HTML表单时，请求首先被控制器捕获。控制器本身不输出任何信息和做任何业务处理，它只是接收请求并决定调用哪个模型部件去处理请求，然后再确定用哪个视图来显示返回的数据。开发中的“控制器”是一个协调视图和模型间关系的特殊类，这个类文件的名字通常以Controller结尾。

ASP.NET 支持3种不同的开发模式：Web Page（Web页面）、MVC（模型－视图－控制器）、Web Forms（Web窗体）。

2. URL 路由

在传统的Web应用程序中，URL表示一个磁盘上的文件，当文件不存在时浏览器会收到服务器返回的404错误。

在ASP.NET MVC Web应用程序中，URL被映射为对一个类的方法调用，而不是服务器磁盘文件。被映射的类为控制器（Controller）类，被调用的方法称为操作（Action）方法。

每个ASP.NET MVC Web应用程序至少需要一个路由来说明URL如何映射到Controller类的Action方法。路由存储在路由集（RouteCollection）中。

在创建MVC项目后，在Global.asax.cs文件的Application\_Start方法中通过调用RegisterRoutes方法定义了一个默认路由并将其加入到路由集中，部分代码如下所示（在App\_Start下的RouteConfig.cs中）：

代码清单1.1

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

}

其中的【"{controller}/{action}/{id}"】为 URL 路由模式，它是一种模式匹配规则，用于决定该路由是否适用于传入的URL请求。该规则被“/”分隔为3段，每段都包含一个由一对“花括号”定义的URL参数，因此它适用于任何带有3段的URL请求，URL中从前向后的3段分别对应其中的{controller}、{action}和{id}。为保证程序正确运行，ASP.NET MVC 框架要求参数中必须包含{controller}和{action}参数。

其后的【new { controller = "Home", action = "Index", id = UrlParameter.Optional }】定义的路由的默认值，它与前面的URL模式共同决定的如何匹配URL请求。

表1.1 URL模式匹配举例

|  |  |  |
| --- | --- | --- |
| URL模式 | 传入的URL | 匹配值 |
| {controller}/{action}/{id} | / | Controller = HomeController  Action = Index  Id = null |
| /Hello | Controller = *HelloController*  Action = Index  Id = null |
| /Hello/Welcome | Controller = *HelloController*  Action = *Welcome*  Id = null |
| /Hello/Welcome/tom | Controller = *HelloController*  Action = *Welcome*  Id = *tom* |

以“http://localhost:1153/”这样的地址为例，除去协议、地址及端口号URL中无任何段，则适用于表1.1中的第1种情况。

需要注意的是：缺省的顺序只能是从右到左。

每个URL请求必须要能解析出相应的{controller}和{action}，{Controller}参数的值用来实例化一个Controller类以处理请求。其约定的规则是{controller}参数名后加“Controller”来确定控制器类的名称（如前面的 HomeController）。如果找不到则返回404错误。

{action}参数的值用来确定调用控制器类的哪个操作方法。如果类或方法不存在也会返回404错误。

根据前面的URL路由规则，“http://localhost:1153/”请求等价于“http://localhost:1153/Home/Index”。

public class HomeController : Controller

{

public ActionResult Index()

{

ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";

return View();

}

}

3. ASP.NET MVC 应用程序结构

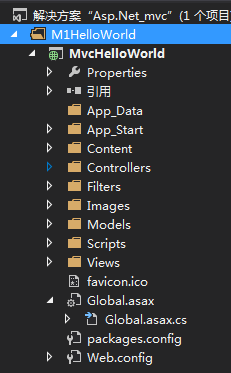


图1.7

默认的ASP.NET MVC项目有6个顶层目录：

* /Controllers目录：Visual Studio默认向其中添加了两个Controller类：HomeController和AccountController类。
* /Views目录：其中有3个子目录（Home、Account和Shared）以及一下些模板文件。
* /Content和/Scripts目录：其中有一个Site.css文件和JavaScript库文件，是站点的静态资源

表1.2 MVC项目默认基层目录

|  |  |
| --- | --- |
| 目录 | 说明 |
| /Controllers | 存放处理URL请求的控制器类 |
| /Models | 存放处理业务逻辑和数据的类 |
| /Views | 存放视图模板文件 |
| /Scripts | 存放JavaScript文件 |
| /Content | 存放CSS和图片文件以及其他静态资源 |
| /App\_Data | 存放数据库文件 |

4. 命名约定

ASP.NET MVC 项目在很大程度上依赖于约定，这可以大减少开发者花在配置方面的时间。

* 每个控制器类的名称都以Controller结尾。
* 所有视图模板文件都放在Views目录中。
* 控制器所使用的视图模板文件都放在Views目录下的以控制器名命名的目录中。
* 所有共享视图文件放在/Views/Shared目录中。

任务二：控制器的创建

一、任务实施

1. 新建控制器

在“解决方案资源管理器”中右击【Controllers】文件夹，然后【添加】→【控制器】

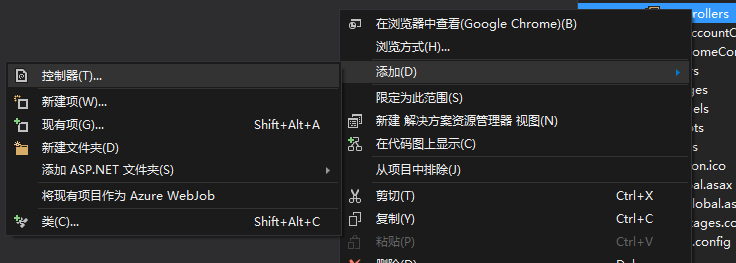


图1.8

注意：调试运行时无法对项目进行修改。

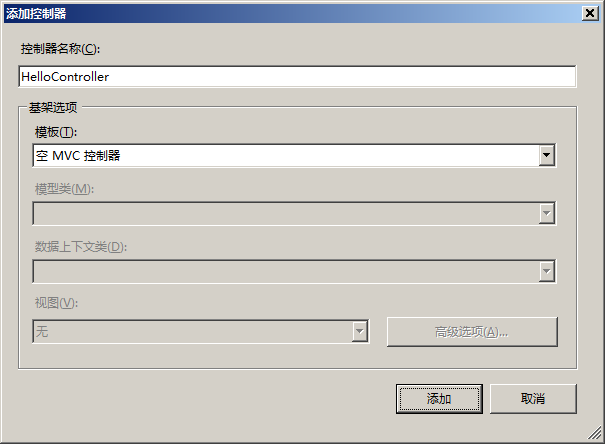


图1.9

2. 修改控制器代码

代码清单1.2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MvcHelloWorld.Controllers

{

public class HelloController : Controller

{

public string Index()

{

return "这是我的<b>默认</b>操作";

}

public string Welcome(string name, int times = 1)

{

string result = "";

result += "这是 Welcome 操作方法 ...<br/>";

result += "名字：" + name + "<br/>";

result += "次数：" + times;

return result;

}

}

}

调试运行该程序（F5键），在地址栏输入“http://localhost:1153/hello”将看到：

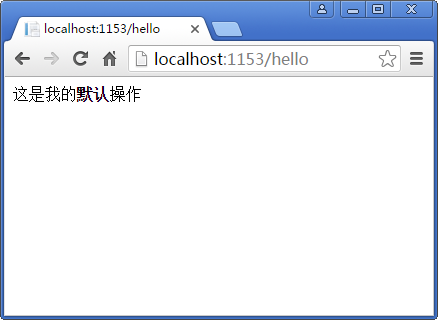


图1.10

在地址栏输入“http://localhost:1153/hello/welcome?name=Lifeng&times=12”将看到：

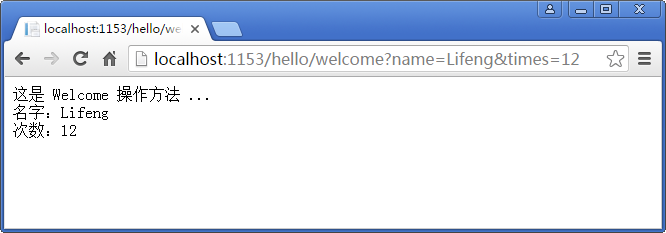


图1.11

二、相关知识

1. 控制器与操作方法

MVC模式中的控制器（Controller）主要负责响应浏览器的HTTP请求，在必要时也会调用模型（Model）来获取或修改数据。

控制器关注的是应用程序的控制流、输入数据的处理和为相关视图（View）的输出提供数据，其工作过程如下。

客户端

Controller（控制器）

Model（模型）

View（视图）

数据库

浏览器

HTTP请求

1、URL请求直接来到Controller中

2、Controller从Model中取数据

Model可能会访问数据库

3、Controller根据业务逻辑选择合适的View呈现给客户端

4、View返回给客户端

图1.12

新建的项目中已经包含了两个控制器：

* HomeController：负责响应对站点根目录的请求。
* AccountController：负责账户相关的请求，如登录和注册账户。

其中，HomeController是一个继承自Controller基类的简单类，其代码清单如下。

代码清单1.3

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MvcHelloWorld.Controllers

{

public class HomeController : Controller

{

public ActionResult Index()

{

ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";

return View();

}

public ActionResult About()

{

ViewBag.Message = "Your app description page.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

}

}

其中的Index和About方法称为操作方法（action methods），控制器可以根据需要定义任意数量的操作方法。操作方法的调用是由用户的交互行为触发的。交互行为可以是：

* 在地址栏输入URL
* 单击一个网页链接
* 提交一个表单
* ……

任何一种行为都会导致浏览器向服务器发送一个URL请求，而在请求中必须包含能够让ASP.NET MVC路由系统解析出来的控制器和操作方法。例如地址“http://localhost:1153/Home/About”会使路由系统调用HomeController控制器的About操作方法来响应用户的请求，About方法则调用View()方法将视图返回给浏览器。

2. 操作方法的返回类型

Index和About操作方法的返回类型是ActionResult类型，ActionResult类是所有操作结果类型的基类。请注意操作方法的最后通过调用控制器的View()方法返回了一个ViewResult类型，ViewResult是ActionResult类型的子类型，正如图1.12所示一样，控制器会选择一个视图来响应请求。

实际上，可以定义一个返回任何对象类型的操作方法，如string类型或integer类型（如控制器HelloController中的Index和Welcome操作方法），这意味着操作方法返回的字符串将直接返回给浏览器显示。

3. 操作参数

操作方法中的形式参数称为操作参数。默认情况下，操作参数会自动从请求数据中获取，这些数据包括表单数据、查询字符串数据和cookie数据。

如果一个操作方法带有参数，MVC框架会在请求数据中查找与操作参数“同名”的数据值，若找到则将该值传递给操作参数，否则将null传递给操作参数。控制器HelloController中的Welcome操作方法即是例证，只不过其times操作参数设置了默认值，在未找到times的同名参数时其值为1。

任务三：Hello控制器Index视图的创建

一、任务实施

1. 添加Index视图

修改代码清单1.2，先给HelloController控制器类的Index操作方法使用视图模板，修改后如代码清单1.4所示，注意其返回类型的变化。

代码清单1.4

public **ActionResult** Index()

{

return **View()**;

}

代码分析

return View() 语句指定使用一个视图模板来呈现对浏览器的响应。ASP.NET MVC 3引入了一种新Razor视图引擎来创建视图模板，基于Razor视图引擎的模板文件的扩展名为cshtml。因未明确指定视图模板文件的名称，ASP.NET MVC 根据约定会默认使用与操作方法同名的视图模板文件，本例中为“~\Views\Hello”文件夹国的Index.cshtml视图模板文件（这里的“~”符号表示项目的根目录）。

右击Index方法然后选择“添加视图”。

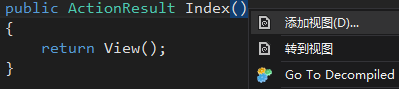


图1.13

在“添加视图”对话框中使用默认值并点击“添加”按钮。

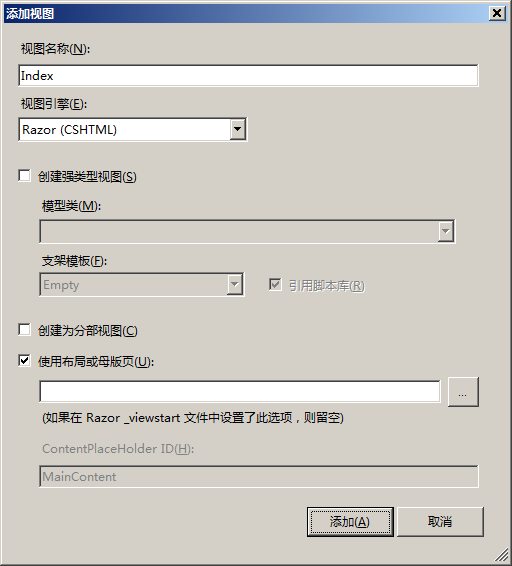


图1.14

完成后在“解决方案管理器”面板的“~\Views\Hello”下面可以找到视图模板文件Index.cshtml。

2. 修改 Index.cshtml 视图模板文件

代码清单1.5

@{

ViewBag.Title = "Hello World";

}

<h2>Hello World! 我的第一个 ASP.NET MVC 应用程序</h2>

<p>这是我<b>默认</b>的操作方法</p>

代码分析

第2行代码通过ViewBag对象的Title属性将字符串“Hello World”传递给布局页面，由布局页面将字符串输出在网页的标题位置。

重新调试运行该程序，在地址栏中输入“http://localhost:1153/hello”，运行效果如图1.15所示。



图1.15

3. 修改页面布局

首先修改的是页面顶部的“我的 MVC 应用程序”标题。这个标题在每个页面都可以看到，它存在于“~\Views\Shared”目录中，文件名为“\_Layout.cshtml”，该文件是其他页面共享的“壳”。

代码清单1.6

<!DOCTYPE html>

<html lang="zh">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

<meta charset="utf-8" />

<title>@ViewBag.Title</title>

<link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />

<meta name="viewport" content="width=device-width" />

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<header>

<div class="content-wrapper">

<div class="float-left">

<p class="site-title">@Html.ActionLink("Hello World 应用程序", "Index", "Home")</p>

</div>

<div class="float-right">

<section id="login">

@Html.Partial("\_LoginPartial")

</section>

<nav>

<ul id="menu">

<li>@Html.ActionLink("主页", "Index", "Home")</li>

<li>@Html.ActionLink("关于", "About", "Home")</li>

<li>@Html.ActionLink("联系方式", "Contact", "Home")</li>

</ul>

</nav>

</div>

</div>

</header>

<div id="body">

@RenderSection("featured", required: false)

<section class="content-wrapper main-content clear-fix">

@RenderBody()

</section>

</div>

<footer>

<div class="content-wrapper">

<div class="float-left">

<p>&copy; @DateTime.Now.Year - Hello World 应用程序</p>

</div>

</div>

</footer>

@Scripts.Render("~/bundles/jquery")

@RenderSection("scripts", required: false)

</body>

</html>

知识解析

布局页面允许在一个地方指定HTML的布局容器，然后将其应用到所有视图。注意该布局页面底部附近的“@RenderBody()”代码，它是一个视图的点位符，视图渲染的结果将插入到该代码处。

重新运行效果如图1.16所示。



图1.16

4. 添加导航链接

在页面布局中增加一个导航链接，以方便访问HelloController类的Index方法。修改“~\Views\Shared\\_Layout.cshtml”中的相关代码。

代码清单1.7

<nav>

<ul id="menu">

<li>@Html.ActionLink("主页", "Index", "Home")</li>

**<li>@Html.ActionLink("欢迎", "Index", "Hello")</li>**

<li>@Html.ActionLink("关于", "About", "Home")</li>

<li>@Html.ActionLink("联系方式", "Contact", "Home")</li>

</ul>

</nav>

重新运行后主页面中的变化如图1.17所示。



图1.17

二、相关知识

1. 视图

用户直接与视图交互，根本感觉不到控制器和模型的存在。视图负责向用户用户界面，其所需输出的数据由控制器提供。

在ASP.NET MVC 中，视图也称为视图模板。每个控制器在Views目录中都有对应的文件夹，其名称与控制器名称相同（不含Controller后缀）。

控制器的每个操作方法都“可能”有一个同名的视图文件与这对应。操作方法通过“return View();”语句的调用来选择一个输出视图，当View()方法的参数为空时表示使用与操作方法同名的视图文件。如果希望使用不同的视图，则需要为View()调用指定一个名称参数（如下面代码中的“NotIndex”）。

public **ActionResult** Index()

{

return **View("NotIndex")**;

}

以上的Index操作方法依然会在与其控制器对应的Views视图文件夹中查找视图，但会选择NotIndex.cshtml作为视图。如果需要使用其他控制器的视图，则需要使用带“~”符号的语法提供视图的完整路径。

public **ActionResult** Index()

{

return **View("~/Views/Home/About.cshtml")**;

}

2. Razor视图引擎

ASP.NET MVC 3之后提供了两种不同的视图引擎：新的Razor视图引擎和原来的Web Form下的ASPX视图引擎。Razor视图引擎相比ASPX视图引擎，是一种干净的、轻量级的、简单的视图引擎。

Razor引擎支持“.cshtml”和“.vbhtml”两种文件类型，前者服务器代码的类别是C#，后者则是VB.NET。

下面是一个简单的视图模板文件内容：

@{

string name = "Tom";

}

<h1>Razor 视图示例</h1>

<p>你好 @name，今年是 @DateTime.Now.Year 年。</p>

从代码可以看出，Razor其实是一种服务器代码和HTML代码混写的代码模板，类似于没有后置代码的.aspx文件。这样的代码需要在服务器端转换为标准的HTML内容后才能发送给浏览器，服务器端的这个过程称为“渲染（Render）”。

3. 布局页

布局页可以使应用程序中的多个视图保持一致的外观，其与Web Form编程中的母板页作用相同，但布局页的语法更简洁、使用更灵活。

可以使用布局页为网站定义公共模板。公共模板中饮食一个或多个占位符，应用程序中的其他视图为它提供内容。

一个项目中可以定义一个或多个布局页，ASP.NET MVC 项目会默认创建一个布局页“\_Layout.cshtml”，位于“~/Views/Shared”目录中。

下面是一个简单的布局页SiteLayout.cshtml的内容。

<!DOCTYPE html>

<html>

<head><title>@ViewBag.Title - 自定义布局页</title></head>

<body>

<h1>@ViewBag.Title - 自定义布局页</h1>

<div>@RenderBody()</div>

</body>

</html>

布局页中的@RenderBody()调用一个占位符，用来表示使用这个布局页的视图将渲染的结果插入到这个位置。下面代码说明的在视图中使用Layout属性来指定布局页的方法。

@{

Layout = "~/Views/Shared/SiteLayout.cshtml";

ViewBag.Title = "Hello World";

}

Layout属性用来指定视图所使用的布局页，如果省略则表示使用默认布局页“\_Layout.cshtml”，等价于如下的描述：

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

ViewBag.Title = "Hello World";

}

指定默认布局页相关描述在Views目录下的“\_ViewStart.cshtml”中，代码如下。

@{

Layout = "~/Views/Shared/\_Layout.cshtml"; //修改这里可以改变默认布局页

}

若视图不想使用则应将Layout设置为null。

@{

Layout = null; //修改这里可以改变默认布局页

}

视图引擎的渲染过程，就是将视图中的实体信息段逐个与布局页中的需求项匹配，从而生成最终的标准HTML内容。这有点类似于带参数的方法调用过程，用下表说明。

|  |  |
| --- | --- |
| 视图 | 渲染结果 |
| @{  Layout = "~/Views/Shared/SiteLayout.cshtml";  ViewBag.Title = "Hello World";  }  <h2>Hello World! 我的第一个 ASP.NET MVC 应用程序</h2>  <p>这是我<b>默认</b>的操作方法</p> | <!DOCTYPE html>  <html>  <head><title>Hello World - 自定义布局页</title></head>  <body>  <h1>Hello World - 自定义布局页</h1>  <div>  <h2>Hello World! 我的第一个 ASP.NET MVC 应用程序</h2>  <p>这是我<b>默认</b>的操作方法</p>  </div>  </body>  </html> |
|  |
| 布局页 |
| <!DOCTYPE html>  <html>  <head><title>@ViewBag.Title - 自定义布局页</title></head>  <body>  <h1>@ViewBag.Title - 自定义布局页</h1>  <div>@RenderBody()</div>  </body>  </html> |

任务四：Hello控制器Welcome视图的创建

一、任务实施

1. 修改Welcome的操作方法

代码清单1.8

public ActionResult Welcome(string name, int times = 1)

{

ViewBag.Message = "你好，" + name + "！";

ViewBag.Times = times;

return View();

}

代码分析：

操作参数中的name和times的值来自于URL的查询字符串。注意，参数的值也可以来自于浏览器的POST请求或cookie。

通过ViewBag对象的Message和Times属性将内容传递给视图，其类型分别为“字符串”和“整数”。

2. 添加Welcome视图

为Welcome方法添加视图，代码如下。

代码清单1.9

@{

ViewBag.Title = "Welcome";

}

<h2>@ViewBag.Message</h2>

@for(int i=0; i < ViewBag.Times; i++)

{

<p>这是 Welcome 视图。</p>

}

代码分析：视图接收由控制器传递来的数据，完成信息的显示及循环的控制。

调试运行程序并在地址栏输入“http://localhost:1153/Hello/welcome?name=Tom&times=3”，运行结果如下：



二、相关知识

1. ViewBag 对象

在控制器的操作方法中，可以使用ViewBag对象向视图传递任何信息，传递的信息可以视图中随意使用。也就是说ViewBag对象是控制器与视图间传递信息的通道。

ViewBag的属性是动态的，它语法简单，可以通过ViewBag属性访问相同的数据。它是一个高效利用了C# 4中danymic关键字的封装器，其中封装了ViewData。这样就可以使用类似属性的访问的语法来检索ViewData字典中的值，也就是说ViewBag.Message等同于ViewBag["Message"]。但如果关键字不是有效的C#标识符，就只能用ViewData来访问。如ViewData["My Name"]因关键字中有“空格”，就不能用ViewBag来访问。

ViewBag还可以在视图与其布局页之间传递数据，这在上一个任务的最后已经使用过了。

2. Razor语法

ASP.NET MVC 3以上版本支持两种视图语法：Razor和ASPX。

Razor语法简单而智能，其核心转换字符是“@”，可以通过理解标记的结构来将代码和HTML识别开来。以如下的SampleView.cshtml为例加以说明。

@{

var name = new string[]{"Jack", "Tom", "Rose"};

}

<html>

<head><title>视图示例</title></head>

<body>

<h1>列出 @name.Length 个名字。</h1>

<ul>

@foreach (var item in name)

{

<li>这个名字是：@item</li>

}

</ul>

</body>

</html>

代码说明：

1）代码表达式

使用“@”符号可以表达两种代码结构：代码表达式和代码块。

代码表达式用于求出表达式的值并在代码处输出，如上述代码中的“@name.Length”，实际输出的结果为“3”。代码的结束位置通过智能扫描完成，遇到无效的字符即刻结束，这称为“隐式”代码表达式求解。相比之下，ASPX语法只支持“显式”代码表达式，即必须告知表达式的结束位置，将上述表达式写成ASPX的表达式的形式则是：

“<% = name.Length %>”

Razor也有可能不能正确理解表达式。

<li>这个名字是：@item.</li>

这个没问题，输出@item的内容，不会将“.”作为语法的一部分。

@{

string serviceName = "mail";

}

<p>@serviceName.sina.com</p >

这可就会出现错误，要解决问题应当这样写：<p>@(serviceName).sina.com</p >，用一对括号括起表达式，括号外为HTML的部分。

<p>serviceName@sina.com</p >

这种写法不会被当作代码，而是当作普通的HTML内容。

@{

ViewBag.title = "search";

}

<p>product\_@ViewBag.title</p >

这回想在“@ViewBag.title”处显示“search”却出错，那就加括号写成：<p>product\_@(ViewBag.title)</p >

需要利用Razor语法输出一个“@”那就要写成“@@”的形式了。

2）代码块

上面的视图文件中有两个代码块：

代码块1：

@{

var name = new string[]{"Jack", "Tom", "Rose"};

}

对应ASPX的书写格式：

<% var name = new string[]{"Jack", "Tom", "Rose"};%>

代码块2：

@foreach (var item in name)

{

<li>这个名字是：@item</li>

}

对应ASPX的书写格式：

<%foreach (var item in name) { %>

<li>这个名字是：<% = item %></li>

<% } %>

注意：代码块中不能写纯文本，必须使用HTML标签，就像前面的<li>……</li>标签。

任务五：源代码管理

一、任务实施

1. 连接到团队项目