

本章导读

控制器是 ASP.NET MVC 的框架核心,负责控制客户端与服务端的交互,协调 Model 与 View 之间数据的传递。本章将学习如何根据需求选择恰当的模板创建控制器、如何选择控制器中的动作属性、如何确定控制器中动作的返回值等 ASP.NET MVC 应用程序开发的核心内容。

本章要点

- 控制器创建
- 控制器模板
- 控制器的动作选择器
- 控制器的 ActionResult

5.1 控制器简介

控制器(Controller)作为 ASP.NET MVC 的框架核心,主要扮演中转和中介两大角色。 中转作用体现在控制器实现承上启下的作用,根据用户输入执行响应(Action),同时在行为 中调用模型的业务逻辑,返回给用户视图(View)。中介角色体现在分离视图和模型,让视 图和模型各司其职,由控制器实现二者的交互。

控制器在 MVC 架构中的作用如图 5.1 所示。



从入门到实战-微课视频版

SP.NET MVC网站开发

5.2 控制器的基本使用

5.2.1 控制器的基本内容



本节使用第1章创建的 MVC 5 网站,详细讲解控制器的基本结构,所有的控制器都存放 在网站根目录的 Controllers 文件夹中,并且以"控制器名称+Controller"命名,如图 5.2 所示。

视频讲解



图 5.2 控制器存放目录

控制器是继承自 System.Web.Mvc.Controller 的 C # 类,Controller 是内置的控制器基 类。控制器中的每个公有方法都称为一个动作,可以通过对应的 URL 从 Web 调用其来执 行,所有的 Controller 都需要满足如下基本约束。

- (1) Controller 类必须为公有类型。
- (2) 控制器名称必须以 Controller 结尾。

(3) 必须继承自 ASP.NET MVC 内置的 Controller 基类,或实现 IController 接口。

打开 Home 控制器对应的 HomeController.cs 文件,代码如下。

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "修改此模板以快速启动你的 ASP.NET MVC 应用程序。";
        return View();
```

```
}
public ActionResult About()
{
    ViewBag.Message = "你的应用程序说明页。";
    return View();
}
public ActionResult Contact()
{
    ViewBag.Message = "你的联系方式页。";
    return View();
}
```

HomeController 类中包含三个公有方法(Public Method),即三个动作(Action),通过 方法可以接收客户端传来的要求,响应视图(View),方法的返回值类型(ActionResult)在后 续章节中介绍。控制器中所有非公有的方法,如 private 或 protected 类型的方法都不会被 视为动作。

5.2.2 控制器的创建

5.2.1 节针对以模板项目创建的控制器进行了说明,除此以外也可以根据需要新建控制器。接下来讲解如何在 Controller 文件内添加新的控制器。



视频讲解

【例 5-1】 在 D 盘 ASP.NET MVC 应用程序目录中创建 chapter5 子目录,将其作为网 站根 目录,创建一个名为 example5-1 的 MVC 项目,在 Controllers 文件夹中新建 HelloController 控制器,添加基本视图,练习控制器的简单应用。

步骤 1: 在"解决方案资源管理器"中右击 Controllers 文件夹,选择"添加"→"控制器" 选项,如图 5.3 所示 。

			9	在浏览器中查看(Internet Explorer)(B)	
			>	任务运行程序资源管理器	
				配置外部工具	
				使用以下工具浏览(H)	
0	控制器(T)			添加(D)	×
*ם	新建项(W)	Ctrl+Shift+A		限定为此范围(S)	
*o	现有项(G)	Shift+Alt+A	Ē	新建解决方案资源管理器视图(N)	
	新搭建基架的项目(F)			从项目中排除(J)	
*-	新建文件夹(D)		ж	剪切(T)	Ctrl+X
	添加 ASP.NET 文件夹(S)	•	ŋ	复制(Y)	Ctrl+C
3	从 Cookiecutter(C)		£	米占见占(P)	Ctrl+V
	REST API 客户端		X	删除(D)	Del
	容器业务流程协调程序支持		X	重命名(M)	
₽	添加客户端库		ç	在文件资源管理器中打开文件夹(X)	
	新建 Azure WebJob 项目		۶	属性(R)	Alt+Enter
	将现有项目作为 Azure WebJob				
	Web API 控制器类(v2.1)				
**	类(C)	Shift+Alt+C			

图 5.3 添加控制器

从入门到实战-微课视频版

步骤 2: 在"添加基架"对话框中选择"MVC 5 控制器-空"模板,单击"添加"按钮,如图 5.4 所示。其他模板将在下一小节介绍。

添加基架			×
▲ 已安装			
▶ 公用 控制器	▲ MVC 5 控制器 - 空	MVC 5 控制器 - 空 依据 Microsoft	
	▲ Web API 2 控制器 - 空	v5.0.0.0 容 MVC 按制器	
	セ含操作的 Web API 2 控制器(使用 Entity Framework)	ID: MvcControllerEmptyScaffolder	
	● 包含读/写操作的 MVC 5 控制器		
	● 包含读/写操作的 Web API 2 控制器		
	eloa 人間的 MVC 5 控制器(使用 Entity Framework)		
	● 具有操作且使用实体框架的 Web API 2 OData v3 控制器		
	◆↓ 具有读/写操作的 Web API 2 OData v3 控制器		
	单击此处可联机并查找更多基架扩展。		
		添加取消	

图 5.4 添加控制器

步骤 3: 在"添加控制器"对话框中修改控制器名称为 HelloController,单击"添加"按钮,如图 5.5 所示。

添加控制器			×
控制器名称(C):			
		添加	取消

图 5.5 输入控制器名称

步骤 4: 在"解决方案资源管理器"的 Controllers 文件夹下,新增 HelloController.cs 文件, Views 文件夹下新增空文件夹 Hello,用于存放 Hello 控制器中各 Action, 对应的界面如 图 5.6 所示。

步骤 5: 打开 HelloController.cs 文件,代码如下。

```
public class HelloController : Controller
{
    // GET: Hello
    public ActionResult Index()
    {
        return View();
    }
}
```



图 5.6 控制器

使用"MVC 5 控制器-空"模板创建的 Hello 控制器初始时只包含一个默认的无参 Index 动作。

步骤 6:为 Index 创建对应视图,在 Index 方法上右击,内容菜单中选择"添加视图"选项,如图 5.7 所示。

Enamespace WebApplication3.Controllers						
public class HelloController : { // GET: Hello	Con					
public ActionResult Index	0	添加视图(D)				
return View();	0	转到视图(V)	Ctrl+M, Ctrl+G			
3	Ŷ	快速操作和重构	Ctrl+.			
	χ	重命名(R)	F2			
		对 Using 进行删除和排序(E)	Ctrl+R, Ctrl+G			
	Ξ	速览定义	Alt+F12			
	•	转到定义(G)	F12			
		转到实现	Ctrl+F12			
		查找所有引用(A)	Ctrl+K, R			
	Ζ	查看调用层次结构(H)	Ctrl+K, Ctrl+T			

图 5.7 控制器添加视图

步骤 7:在"添加视图"对话框中修改"视图名称"为 Index,"模板"选择 Empty,单击"添加"按钮,如图 5.8 所示。更多视图模板将在第 6 章中介绍。

步骤 8:在 Views/Hello 文件夹内,新增了 Index.cshtml 视图文件,如图 5.9 所示。

步骤 9: 打开 Index.cshtml 文件,修改代码如下。

从入门到实战-微课视频版

添加视图	×
视图名称(N):	Index
模板(工):	Empty (不具有模型) ×
模型类(<u>M</u>):	v
数据上下文类(D):	×
选项:	
🗌 创建为分部视图	Q
✓ 引用脚本库(R)	
✓ 使用布局页(U):	
(如果在 Razor_v	newstart 文件中设置了此选项,则留空)
	添加取消

图 5.8 视图选择模板

M→ 解决方案"WebApplication3"(1 个项目)					
WebApplication3					
Connected Services					
Properties					
▶ ••■ 引用					
App_Data					
App_Start					
Content					
Controllers					
fonts					
Models					
Scripts					
 Views 					
🔺 🛋 Hello					
@ Index.cshtml					
Home					
Shared					
@]_ViewStart.cshtml					
🖓 Web.config					
🗈 favicon.ico					
▶ 🗗 Global.asax					
🛍 packages.config					
Web.config					
4					

图 5.9 视图文件

```
@ {
    ViewBag.Title = "Index";
}
```

<h2>Hello Index</h2>

步骤 10: 运行网站,输入网址"http://localhost:XXXX/Hello/Index",网站运行效果 如图 5.10 所示。

注意:

http://localhost:XXXX/Hello/Index 网址中,XXXX 代表端口号,请自行替换为读者 计算机实际端口号。

→) ④ 🕀 - 1 📄 http://localhost:50547/Hello/Index 😧 P - C 📔 Index - 我的 ASP.NE ×	-	口 命 ☆	× © @
应用程序名称 主页 关于 联系方式			
Hello Index			
© 2019 - 我的 ASP.NET 应用程序			

图 5.10 网站运行页面

5.2.3 控制器的读写模板

在例 5-1 第 2 步的"添加基架"对话框中,可以为待创建的控制器进行模板选择,示例中 使用的是"MVC 5 控制器-空"模板,除此以外,ASP.NET MVC 5 中还支持"包含读/写操作 的 MVC 5 控制器""包视图的 MVC 5 控制器(使用 Entity Framework)"等模板,恰当的模 板选择可以极大地提高后续开发效率。接下来对最常用的"包含读/写操作的 MVC 5 控制 器"模板进行简单介绍。

在例 5-1 的第 2 步选择"包含读/写操作的 MVC 5 控制器"模板,则创建的控制器中除 了 Index 动作外,还会包含 Details、Create、Edit、Delete 等动作。其中,Create、Edit、Delete 都包含[HttpGet]和[HttpPost]修饰的两个动作,在这些方法上适当添加代码就可以实现 读写等相关的操作。

初始默认代码如下。

```
namespace WebApplication3.Controllers
{
    public class HelloController : Controller
    {
        // GET: Hello
        public ActionResult Index()
        {
            return View();
        }
}
```

从入门到实战-微课视频版

```
}
// GET: Hello/Details/5
public ActionResult Details(int id)
{
   return View();
}
// GET: Hello/Create
public ActionResult Create()
{
   return View();
}
// POST: Default/Create
[HttpPost]
public ActionResult Create(FormCollection collection)
{
   try
    {
       // TODO: Add insert logic here
      return RedirectToAction("Index");
   }
   catch
   {
      return View();
   }
}
// GET: Hello/Edit/5
public ActionResult Edit(int id)
{
   return View();
}
// POST: Hello/Edit/5
[HttpPost]
public ActionResult Edit(int id, FormCollection collection)
{
   try
    {
       // TODO: Add update logic here
       return RedirectToAction("Index");
   }
   catch
   {
      return View();
   }
```

```
}
   // GET: Hello/Delete/5
   public ActionResult Delete(int id)
    {
       return View();
   }
   // POST: Hello/Delete/5
    [HttpPost]
   public ActionResult Delete(int id, FormCollection collection)
   {
       try
       {
           // TODO: Add delete logic here
           return RedirectToAction("Index");
       }
       catch
       {
           return View();
       }
   }
}
```

5.3 动作选择器

动作选择器也称为动作方法选择器(Action Method Selector),是应用于动作方法上的 属性,用于响应控制器对方法的调用,通过路由引擎选择正确的操作方法来处理特定的请 求。动作方法选择器使用较多的是动作名称(ActionName)、无为动作(NonAction)和动作 方法限定(ActionVerbs)三种属性。

5.3.1 动作名称属性

当 ActionInvoker 选取 Controller 中的 Action 时,默认会应用反射机制找到相同名字的方法,这个过程就是动作名称选择器运行的过程。除此也可以使用"动作名称"的属性,通过[ActionName]属性设置动作方法的别名。选择器将根据修改后的名称来决定方法的调用,选择适当的 Action。



视频讲解

[ActionName]基本语法如下。

```
[ActionName("newActionName")]
```

newActionName 是开发人员为方法设置的别名,选择时不区分动作名称的大小写。

从入门到实战-微课视频版

【例 5-2】 在 chapter5 目录中创建一个名为 example5-2 的项目,创建 MVC 页面,在控制器中为 Action 添加 ActionName 属性,在页面中测试该动作名称属性。

步骤 1:在 Home 控制器中添加 GetDateTimeView 方法,编辑代码如下。

```
[ActionName("GetDate")]
public string GetDateTimeView()
{
   return DateTime.Now.ToLongDateString();
}
```

步骤 2: 创建对应的 GetDateTimeView.cshtml 页面。

步骤 3: 使用 http://localhost:55566/Home/GetDate 访问页面,如图 5.11 所示。



图 5.11 ActionName 属性访问实例

方法中添加了[ActionName("GetDate")]属性,所以访问 GetDateTimeView 动作,需要使用路由 http://localhost:55566/Home/GetDate,此时 ASP.NET MVC 会去寻找/ Views/Home/ GetDateTimeView.cshtml 视图页面来运行。

一个 Action 只可以包含一个 ActionName 属性,不允许多个方法对应同一个 Action 名称,否则在运行请求对应 Action 时会引发异常。

5.3.2 无为动作属性



视频讲解

NonAction 是 Action 的另一个内置属性,将 NonAction 属性应用在 Controller 中的某 个 Action 上,则 ActionInvoker 将不会运行该 Action。这个特性主要用来保护 Controller 中的某些特殊的公开方法不发布到 Web 上,或是隐藏某些尚未开发完成而又不想删除的 Action,套用这个特性就可以不对外公开该功能。

【例 5-3】 创建 MVC 页面,在控制器中为某一 Action 添加 NonAction 属性,在页面中 测试该无为动作属性。

步骤1:编写 Action 代码如下。

```
[NonAction]
public ActionResult NonAction 测试()
{
    return View();
}
```

步骤 2: 创建对应的"NonAction 测试.cshtml"页面。