第 5 章

视　图

本章导读：

在 ASP.NET MVC 中，只需要处理两种主要类型的组件：一种是控制器，它负责执行请求并为原始输入生成原始结果；另一种是视图引擎，它负责生成基于由控制器计算出的结果的任何预期的 HTML 响应。本章将介绍 ASP.NET MVC 中能见度最高、最靠近终端用户的层面，即 View。View 通过应用程序在 Action 中返回 ViewResult 或 PartialViewResult，在运行阶段内部调用 ExecuteResult() 方法而产生模板转换(transformation)，将运行阶段运算后产生的 Model 经由模板引擎进行转换，从而生成 HTML 页面代码，输出到浏览器中。因为 View 用来生成网页，因此可以说它是 ASP.NET 所有内建的 ActionResult 中使用率最高的一种类型。

在一般的应用程序开发上，View 几乎服务了 ASP.NET MVC 程序一切的界面设计(或称接口设计)与运算结果呈现，在隐性的应用中还包含数据验证等 API 的行为，View 是在构建网络应用产品或服务时最不可或缺的技术。

本章要点：

视图引擎(View Engine)负责处理 ASP.NET 内容，并查找有关指令，这些指令典型的是将动态内容插入发送给浏览器的输出，而 Razor 是 MVC 框架视图引擎的名称，因此，首先介绍视图的作用和类型，然后简要地论述视图引擎的内部结构，并对如何为引擎提供视图模板和数据进行实际的考量。

5.1　视图的作用

视图的职责是向用户提供用户界面，不像基于文件的 Web 框架，如 ASP.NET Web Forms 和 PHP，视图本身不会被直接访问，浏览器不能直接指向一个视图并渲染它。相反，视图总是被控制器渲染，因为控制器为它提供了要渲染的数据。

在标准 ASP.NET MVC 项目结构中，View 的位置被设计存储在应用程序根目录中名为 Views 的子目录下，在一些简单的情况中，视图不需要或需要很少控制器提供的信息。更常见的情况是控制器需要向视图提供一些信息，所以它会传递一个数据转移对象，叫作模型。视图将这个模型转换为一种适合显示给用户的格式。在 ASP.NET MVC 中，完成这一过程有两部分操作：一个是检查由控制器提交的模型对象；另一个是将其内容转换为 HTML 格式。

当创建新的项目模板时，将会注意到，项目以一种非常具体的方式包含了一个结构化

的 Views 目录。按照约定，每个控制器在 Views 目录下都有一个对应的文件夹，其名称和控制器一样，只是没有 Controller 后缀名。例如，控制器 HomeController 在 Views 目录下就会对应一个名为 Home 的文件夹。在每个控制器的 View 文件夹中，每个操作方法都有一个同名的视图文件与其对应，这就提供了视图与操作方法关联的基础。

ASP.NET MVC 的 Razor 视图引擎具有非常好的扩展性，使用 Razor C＃语法的 ASP.NET 网页的扩展名为 cshtml，使用 Razor VB 语法的 ASP.NET 文件的扩展名为 vbhtml，Razor 视图引擎将编写视图模板所需的代码量降至最少。例如，控制器中的操作方法通过 View()方法返回 ViewResult 对象，代码如下所示。

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "Asp.NET MVC application";
        return View();
    }
}
```

注意，这个控制器操作没有指定视图的名称，当不指定视图名称时，操作方法返回的 ViewResult 对象将按照约定确定视图。它会在目录“/View/ControllerName”(这里的 ControllerName 不带 Controller 后缀)下查找与 Action 名称相同的视图，在这种情况下选择的视图便是“/Views/Home/Index.cshtml”。

到目前为止介绍的控制器操作简单地调用 return View()渲染视图，还不需要指定视图的文件名。可以这么做，是因为它们利用了 ASP.NET MVC 框架的一些隐式约定，这些约定定义了视图选择逻辑。与 ASP.NET MVC 中大部分约定的设置一样，这一约定是可以重写的。如果想让 Index()操作方法渲染一个不同的视图，可以向其提供一个不同的视图名称，代码如下所示。

```
public ActionResult Index()
{
    ViewBag.Message = "Asp.NET MVC application";
    return View("Welcome");
}
```

这样编码后，虽然操作的方法仍然在“/Views/Home”目录中查找视图，但是选择的不再是 Index.cshtml，而是 Welcome.cshtml。在其他一些应用中，可能还需要定位到位于完全不同目录结构中的视图。针对这种情况，可以使用带有～符号的语法提供视图的完整路径，代码如下。

```
public ActionResult Index()
{
    ViewBag.Message = "Asp.NET MVC application";
```

```
    return View("~/Views/Manage/Index.cshtml");
}
```

注意，为了在查找视图时避开视图引擎的内部查找机制，使用这种语法时必须提供视图的文件扩展名。

5.2 视图类型

View 是扩展名为.cshtml(或.vbhtml)的程序代码，广义上说，一个 View 可解释为代表着一个页面内容，但实际上 View 可细分为多种功能，赋予不一样的责任。

视图可以分为以下 3 种类型。

常规视图：常存放于对应 Controller 名称的目录下。

分部视图：放置在 Controller 名称的目录下或集中放置在 Shared 目录中。

布局页：也叫母版页，是指可被其他页面作为模板引用的特殊网页。

接下来进一步说明和解析上述 3 种 View 类型。

5.2.1 常规视图

虽然可以手动创建视图文件，把它添加到 Views 目录下，但是 Visual Studio 中的 ASP.NET MVC 工具的 Add View 对话框使得创建视图非常容易。可以在 Views 文件夹或者子文件夹上右击，从弹出的快捷菜单中选择“新建”→“视图”命令，也可以在控制器方法上右击，从弹出的快捷菜单中选择“添加视图”命令。

下面通过新建学生说明如何创建视图。首先在 Models 文件夹下新建一个 Student 模型，然后在 HomeController 中添加 Create 动作。Student 模型和 Create 动作的代码如下所示。

```
public class Student
{
    public string Name { get; set; }
    public string Number { get; set; }
    public string Sex { get; set; }
    public int Age { get; set; }
}
public ActionResult Create()
{
    return View();
}
```

然后在操作方法 Create()上右击，从弹出的快捷菜单中选择“新建视图”菜单项，打开“添加视图”对话框，如图 5.1 所示。

下面对每个菜单项进行详细描述。

(1) 视图名称：如果在操作方法上打开这个对话框时，视图的名称默认被填充为操

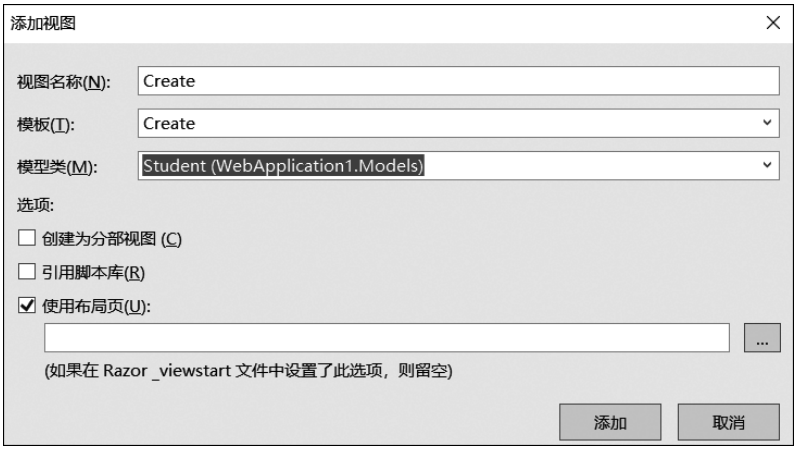

图 5.1 “添加视图”对话框

作方法的名称，则视图的名称是必填项。

(2) 模板：一旦选择一个模型类型，就可以选择一个基架模板。这些模板利用 Visual Studio 模板系统生成基于选择模型类型的视图。

视图模板包括以下类型。

- Create：创建一个视图，其中带有创建模型新实例的表单，并为模型类型的每个属性生成一个标签和输入框。
- Delete：创建一个视图，其中带有删除现有模型实例的表单，并为模型的每个属性显示一个标签，以及当前该属性的值。
- Details：创建一个视图，它显示了模型类型的每个属性的标签及其相应值。
- Edit：创建一个视图，其中带有编辑现有模型实例的表单，并为模型类型的每个属性生成一个标签和输入框。
- Empty：创建一个空视图，使用@model 语法指定模型类型。
- Empty(不具有模型)：与 Empty 基架一样，创建一个空视图。但是，由于这个基架没有模型，因此在选择此基架时不需要选择模型类型，这是唯一不需要选择模型类型的一个基架类型。
- List：创建一个带有模型实例表的视图。为模型类型的每个属性生成一列。确保操作方法向视图传递的是 Enumerable<ModelType>类型，同时，为了执行创建、编辑、删除操作，视图中还包含指向操作的链接。

(3) 模型类：除了选用“Empty(不具有模型)”类型的模板外，其他模板都需要指定与视图关联的模型类。

(4) 引用脚本库：这个选项用来指示要创建的视图是否应该包含指向 JavaScript 库(如果对视图有意义)的引用。默认情况下，Layout.cshtml 文件既不引用 jQuery Validation 库，也不引用 Unobtrusive jQuery Validation 库，只引用主 jQuery 库。

当创建一个包含数据条目表单的视图(如 Edit 视图或 Create 视图)时，选择这个选项会添加对 jqueryval(在 BundleConfig.cs 中，主要用来压缩 JavaScript 和 CSS)捆绑的脚本

引用。如果要实现客户端验证，那么这些库就是必需的。除这种情况外，完全可以忽略这个复选框。

(5) 创建为分部视图：选择这个选项意味着要创建的视图不是一个完整的视图，因此，Layout 选项是不可用的。生成的分部视图除在其顶部没有<html>标签和<head>标签外，很像一个常规的视图。

(6) 使用布局页：这个选项决定了要创建的视图是否引用布局，还是成为一个完全独立的视图。如果选择使用默认布局，就没必要指定一个布局了，因为在_ViewStart.cshtml 文件中已经指定了布局，这个选项是用来重写默认布局文件的。

单击“添加”按钮，会创建出基于 Student 模型的 Create 视图，该视图也称为强类型视图，代码如下所示。

```
@model WebApplication1.Models.Student

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Student</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Name, htmlAttributes: new { @class
= "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Name, "", new { @
class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Number, htmlAttributes: new { @class
= "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Number, new { htmlAttributes =
```

```
new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Number, "", new { @
class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Sex, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Sex, new { htmlAttributes = new {
@class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Sex, "", new { @class
= "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Age, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Age, new { htmlAttributes = new {
@class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Age, "", new { @class
= "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

第一行@model定义了该视图对应的模型为Student，当使用模型创建视图时，会使视图创建非常简单，通过模板基架会自动创建出合适的视图直接使用。

5.2.2 分部视图

何为“分部视图”？在Web Forms开发中经常用到用户自定义控件，其作用是提高代

码的复用性，减少代码的冗余，使程序更加模块化。在 ASP.NET MVC 中，对应地引入基于 Razor 结构的分布页，其作用与 Web Forms 开发中的用户自定义控件差不多。

分部视图(Partial View)是指可应用于 View 中以作为其组织部分的 View 的部分(片段)，分部视图可以像类一样，编写一次，然后在其他 View 中被多次反复应用。由于分部视图需要被多个不同的 View 所引用，所以分部视图一般放在 Views/Shared 文件夹中以共享。

Partial View 与 View 几乎完全相同，细微的差别是 Action 返回 PartialViewResult 时，不会引用_ViewStart.cshtml 中默认的指定，在 Partial View 中除了自行指定使用哪份 Layout，不会引用前述_ViewStart.cshtml 中的默认值。Action 执行结束时，Controller 类型中的 PartialView()方法成员返回一个 PartialViewResult 给 ActionInvoker，以便后续 Partial View 的执行。

```
public ActionResult GetData (  )
{
    return PartialView("_GetDataPartial");
}
```

在命名习惯上，Partial View 的文件名部分会以 Partial 为结束，如_GetTimePartial.cshtml，虽然这不是一定要遵守的命名规则，但是这个命名方式的确有助于在解决方案资源管理器中寻找 View 文件。

在视图中要想渲染一个分部视图，可以使用 Html.Partial 和 Html.RenderPartial 辅助方法，在学习 HTML 辅助方法时将会详细介绍。

5.2.3 布局页

当创建一个默认的 ASP.NET MVC 项目时，在 Views 目录下会自动添加一个 Razor 布局使应用程序中的多个视图保持一致的外观。如果熟悉 Web Forms，其中母版页和布局的作用是相同的，但是布局提供了更简捷的语法和更大的灵活性。可使用布局为网站定义公共模板(或只是其中的一部分)。公共模板包含一个或多个占位符，应用程序中的其他视图为它(们)提供内容。从某些角度看，布局很像视图的抽象基类。

指定母版视图是比较容易的，可以使用视图引擎所支持的规则，也可以在控制器中选择下一个视图时将母版视图的名称作为参数传递给 View()方法。注意，与普通视图相比，布局页可能会遵循不同的规则。例如，ASPX 视图引擎要求母版模板的扩展名为.master，且需要放在 Shared 文件夹中。而 Razor 视图引擎则要求添加.cshtml 扩展名，并需要在 Views 文件夹根目录下的一个专用的_ViewStart.cshtml 文件中指定路径。_ViewStart.cshtml 文件代码如下所示。

```
@{
    Layout="~/Views/Shared/_Layout.cshtml";
}
```

_ViewStart.cshtml 页面可用来消除多个视图使用统一布局的冗余，这个文件中的代

码先于同目录下任何视图代码的执行，可以递归地应用到子目录下的任何视图，所以子视图可以重写 Layout 属性的默认值，从而重新选择一个不同的布局。如果一组视图拥有共同的设置，那么使用_ViewStart.cshtml 文件对共同的视图配置进行统一设置。如果有视图需要覆盖统一的设置，那么只修改对应视图的属性值即可。

下面看 Shared 文件夹下项目生成的默认布局_Layout.cshtml 文件代码。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - 我的 ASP.NET 应用程序</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle=
"collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("应用程序名称", "Index", "Home", new { area = "" },
new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <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>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - 我的 ASP.NET 应用程序</p>
        </footer>
```

```
        </div>

        @Scripts.Render("~/bundles/jquery")
        @Scripts.Render("~/bundles/bootstrap")
        @RenderSection("scripts", required: false)
    </body>
    </html>
```

该视图看起来像一个标准的 Razor 视图，但需要注意以下几个问题。

1. RenderBody

@RenderBody 是布局页中比较重要的元素，这是一个占位符，用于在布局页的＜body＞与＜/body＞之间的某个位置定义视图页或视图的位置。

呈现引用此布局页的视图或视图页时，MVC 会自动将视图或视图页的内容合并到布局页中调用 RenderBody()方法的位置处，多个 Razor 视图可以利用这个布局显示一致的外观。

2. RenderSection

@RenderSection 用于在布局页指定在视图中(注意不是指分部视图)用 Section 定义的占位符。将视图中定义的 Section 嵌入布局页中指定的位置后，当呈现引用此布局页的视图时，MVC 会自动将所定义的 Section 的内容合并到布局页中调用 RenderSection()方法的位置处。

@RenderSection 有一个必要参数作为区段名称，并且有一个选择性参数 required，省略 required 参数或者设置 required：true 用来指定套用这份 Layout 的 View 是否必须满足这个区段，如果没有提供该区段，则会导致执行时弹出错误，用以下程序代码说明。

如果在布局页中添加下面的语句：

```
@RenderSection("SectionOne", false)
```

或

```
@RenderSection("SectionOne", required :false)
```

在视图中就可以用下面的办法定义：

```
@section SectionOne{
    …
}
```

如果在布局页中通过 Razor 语法设置了 required ：false 的 RenderSection()方法，但是又希望当所有没有实现 sectionName 的视图呈现的相关内容有默认值，则可通过 IsSectionDefined 实现。IsSectionDefined (sectionName)用于判断视图页中是否已经定义了用 sectionName 指定的名称，如果视图页中没有定义该名称，就在布局页中定义呈现的内容。

在布局页中添加下面的语句：

```
@if(IsSectionDefined("SectionOne"))
{
    @RenderSection("SectionOne")
}
else
{
    <p>SectionOne Section is not defined!</p>
}
```

在需要设置的视图中就可以用下面的办法定义：

```
@section SectionOne{
    …
}
```

若没有设置名为 SectionOne 的 RenderSection 视图，则显示的内容可写在布局页的 else()方法体中。

3. Layout 与 View 的执行顺序

在执行顺序上，View 会被优先执行，然后被 View 引用的 Layout 执行，这一点是经常容易被误解为相反的情况，因此产生不正确的程序编写结果。例如，在程序中以 ViewBag、ViewData 在 View 与 Layout 之间进行变量的传递，就会有执行顺序的问题需要考虑，因此一开始需要特别留意概念的建立。

下面在目录“～/Views/Shared”下定义一个名为_MyLayout 的布局页面，并修改布局页代码如下。

```
<!DOCTYPE html>
<html>
      <head><title>@ViewBag.Title</title></head>
      <body>
          <h1>@ViewBag.Title</h1>
          <div id="main-content">@RenderBody()</div>
      </body>
</html>
```

5.3 ASP.NET 视图引擎

View Engine 是隐藏在 Controller/Action 和 View 之间的黏合剂，当 Action 执行结束并且返回 ViewResult(或 PartialViewResult)后，ActionInvoker 调用 ActionResult 中定义的 ExecuteResult()方法启动了系统中 View Engine 的工作。View Engine 的职责是根据 ActionInvoker 提供的 context 获取合适的 View，将从 ViewResult 得到的数据给模板程序执行，并将结果输出为网页。因历史关系，现在的 ASP.NET MVC 内建了两份 View Engine 的实现，即 Controller 中已经概略提到的 Web Forms View Engine 与 Razor