**《购物网站开发案例学习系列》**

# 任务1 初识ASP.NET MVC项目开发

这个案例结合了黄保翕书的电子商务案例和微软官方的音乐商店案例，介绍和展示了如何使用ASP.NET MVC和Visual Studio进行Web开发的示例应用程序，我们将从基础知识开始，一步一步地进行介绍，因此，初学者也可以去很好的学习体验。

这个案例视图使用Razor引擎，数据库访问使用EF Code First。

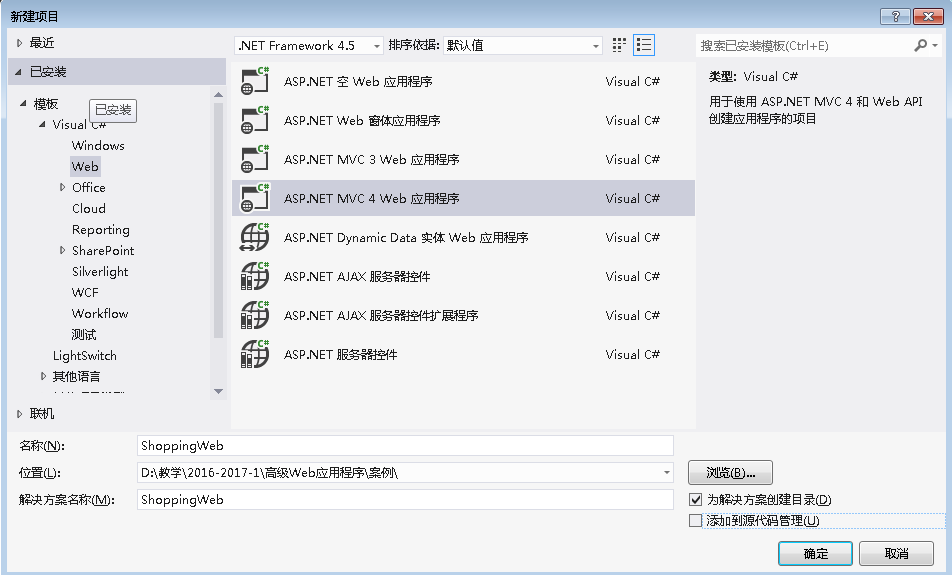
## 1.1 示例项目概述

我们打算创建一个拥有基本功能的电子商务网站，其功能包括会员注册、登录注销、商品浏览、添加购物车、购物车列表、订单完成结账等最常见的电子商务功能。

## 1.2 利用项目模板创建ASP.NET MVC项目

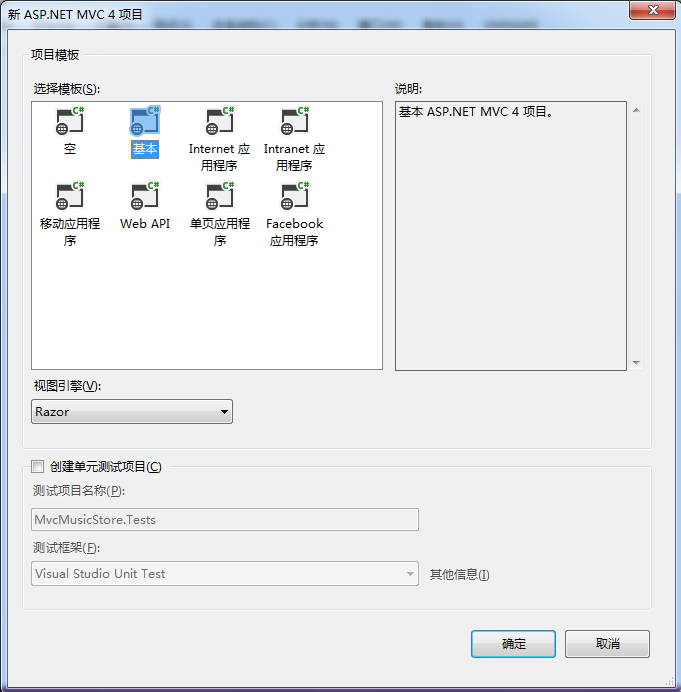
1.启动VS2012，点击“文件|新建|项目”。

2.在新建项目的窗口中，选择ASP.NET MVC 4应用程序。



ShoppingWeb

3.在新ASP.NET MVC 4项目窗口中的“选择模板”列表中选择“基本”模板，在“视图引擎”下拉列表框中选择“Razor”，不用勾选“创建单元测试项目”，最后单击“确定”按钮。



4.此时，ASP.NET MVC项目已经创建完成，此时启动网站可能会出错，因为目前还没有任何控制器。

## 1.3创建控制器

在典型的 Web 应用中，用户请求的 URL 地址通常映射到保存在网站中的文件上，例如，当用户请求 /Products.aspx 的时候，或者 /Products.php 的时候，很可能是在通过处理 Products.aspx 或者 Products.php 文件来完成任务。

ASP.NET MVC 的处理方式则不同，它没有映射到文件上，相反，将这些 URL 地址映射到类的方法上，这些类被称为“控制器”，控制器用来接受 HTTP 请求，处理用户的输入，获取或者保存数据，其中的处理方法称为 Action，然后将回应发送到客户端，可能是显示了一个 HTML 的网页，下载一个文件，重定向到另外一个地址等等。

### 1.3.1 默认的路由配置

在新创建的项目中，打开“App\_Start | RouteConfig.cs” 文件，可以看到如下的代码。

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

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

routes.MapRoute(

name: "Default", //路由名称

url: "{controller}/{action}/{id}", //带有参数的URL

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional } //参数默认值

);

}

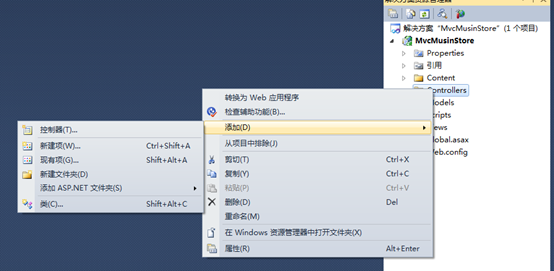
}

  其中 RegisterRoutes 方法注册了默认的路由配置，在方法中的 routes.MapRoute 语句中，请求地址将会被看成三个部分组成，{controller}/{action}/{id}，第一部分称为控制器，如果没有提供的话，默认为 **Home**, 第二部分称为Action 方法，如果没有提供的话，默认为 **Index**, 第三部分称为id, 通常用来提供数据的标识，没有默认值。这样，当请求 / 地址的时候，系统将会把请求映射到名为 Home 的控制器进行处理，调用其中名为 Index 的方法处理请求。

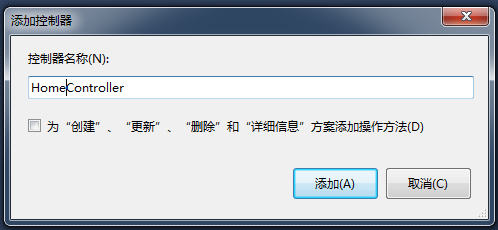
### 1.3.2 增加 HomeController

我们的应用商店从增加一个首页的控制器开始，使用默认的命名约定，控制器的名称应该以 Controller 作为后缀，我们将这个控制器命名为 HomeController。

  在 Controller 文件夹上右击，然后选择“添加”，“控制器(T)…”



在弹出的对话框中，输入控制器的名字 HomeController，按下“添加”按钮。



这样，将会创建一个名为 HomeController.cs 的文件，其中的代码如下：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ShoppingWeb.Controllers

{

public class HomeController : Controller

{

//

// GET: /Home/

public ActionResult Index()

{

return View();

}

}

}

为了尽可能地简单，我们让 Index 方法简单地返回一个字符串，这个字符串将作为回应内容直接返回浏览器，我们做下面的两个简单修改。

* 将方法的返回类型修改为string
* 将返回语句修改为 return "Hello form Home";

这样，我们的方法将会变成如下的内容。

public string Index()

{

return "Hello form Home";

}

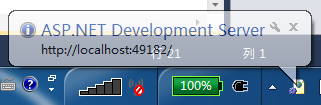
### ****1.3.3 运行程序****

现在，可以运行一下了，我们可以启动服务器，使用下面的任何一种方法都可以。

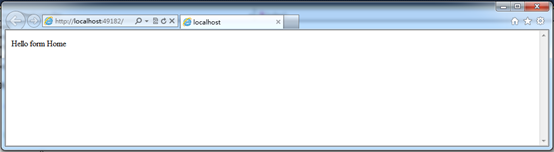
* 选择“调试”，“启动调试”菜单
* 点击工具栏上的绿色的箭头按钮
* 使用键盘快捷方式，F5



使用上面的任何一种方法都会导致编译项目，启动 Visual Studio 中内建的 ASP.NET 开发服务器.



在屏幕的右下角会弹出一个启动 ASP.NET 开发服务器的提示，Visual Studio 将自动打开一个浏览器窗口，其中的地址指向我们的 Web 服务器，这使得我们可以快速使用我们的程序。



Ok, 非常的简单，我们已经创建了一个 Web 站点，写了一个三行的函数，在浏览器中得到了这行文字， 不是很震撼，但这仅仅只是开始。

### 1.3.4 增加 StoreController

  我们已经为站点增加一个简单的 HomeController 作为首页，现在，增加另外一个控制器，可以用来浏览我们的购物网站，我们的网站控制器将要支持三个场景：

* 列出商店中的商品分类
* 浏览商店中某个分类中的商品列表
* 显示特定商品的详细信息

我们从增加一个新的 StoreController 开始，先停止刚才的程序，可以直接关闭浏览器，或者也可以在调试菜单中，选择停止调试。

现在，增加一个新的 StoreController，像在 HomeController 中做的一样，我们在 Controllers 文件夹上右健，选择“添加”，“控制器”。

  新的 StoreController 控制器中已经包含了 Index 方法，我们使用这个方法来实现列出所有分类的列表，我们还会再增加两个额外的方法来实现其他的场景：浏览和明细。

  这些包含在控制器中的方法，我们称为控制器中的 Action，像你在前面看到的，HomeController 中的 Index 方法就是一个 Action，这些 Action 的作用就是处理请求，然后返回对请求的处理结果。

对于我们的 StoreController ，首先让 Index 这个 Action 返回一个“Hello” 串，然后，增加两个方法：Browse() 和 Detials()

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ShoppingWeb.Controllers

{

public class StoreController : Controller

{

//

// GET: /Store/

public string Index()

{

return "Hello from Store.Index()";

}

public string Browse()

{

return "Hello from Store.Browse()";

}

public string Details()

{

return "Hello from Store.Details()";

}

}

}

重新运行程序，现在已经可以访问这些地址了。

* /Store
* /Store/Browse
* /Store/Details

  太棒了，但是现在仅仅能够返回一些常量的字符串，让我们将它们变成动态的，我们从 URL 中获取一些信息，然后把他们显示在返回的页面中。

  首先，修改我们的 Browse 这个 Action，使得它可以从 URL 地址中获取查询信息，为方法增加一个名为“category” 的字符串类型参数，当我们这样做的时候，ASP.NET MVC 就会自动把任何名为category的请求参数的值赋予这个参数。

// GET: /Store/Browse?category=?Disco

public string Browse(string category)

{

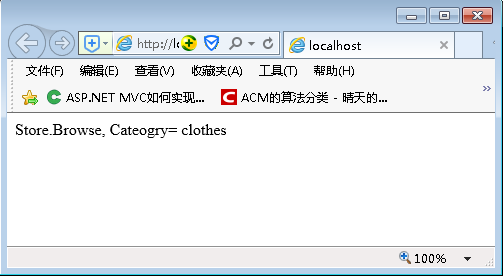
string message = HttpUtility.HtmlEncode("Store.Browse, Category = " + category);

return message;

}

注意：我们使用了HttpUtility.HtmlEncode 方法来处理用户的输入，这样可以防止用户的脚本注入攻击。例如：/Store/Browse?category=<script>window.location=’http://hackersite.com’</script>。

现在，我们在浏览器中访问一下：/Store/Browse?category=clothes。



  下一步，我们处理 Details 这个 Action，使它能够处理名为 ID 的整数类型参数。这次，我们不再在请求参数中传递这个整数，而是嵌在请求的 URL 地址中。例如：/Store/Details/5。

  在 ASP.NET MVC 中，我们可以轻易地完成这个任务而不需要配置任何东西，ASP.NET MVC 默认的路由约定会将跟在 Action 方法之后的部分看作名为 ID 的参数的值，如果你的 Action 方法有一个名为 ID 的参数，那么，ASP.NET MVC 就会自动将这部分作为参数传送给 Action 方法，需要注意的是，MVC 可以帮助你完成数据类型之间的转换，所以，地址的第三部分一定要可以转换为整数。

// GET: /Store/Details/5

public string Details(int id)

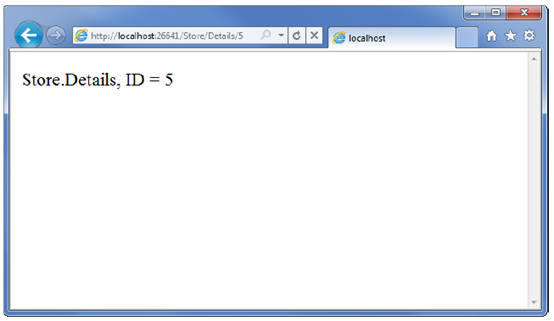
{

string message = "Store.Details, ID = " + id;

return message;

}

再次运行程序访问/Store/Details/5。



总结一下我们完成的任务：

* 创建了一个 ASP.NET MVC 项目
* 讨论了基本的项目文件夹
* 学习了如何运行开发服务器
* 创建了两个控制器 HomeController 和 StoreController
* 为控制器增加了 Action 方法。
* 理解路由
  + /home/index
  + /store/browse
  + /store/browse?genre=disco
  + /store/index
  + /store/details
  + /store/details/5

## 1.4 创建视图与模型

上一节中有同学提到为什么不使用视图，而使用字符串，这一节我们就开始使用视图来处理。

  我们已经可以从控制器的Action中返回一个字符串，这可以帮助我们更好地理解Controller是如何工作的。但是对于创建一个Web程序来说还是不够的。下面我们使用更好的方法来生成 HTML，主要是通过模板来生成需要的HTML，这就是视图所要做的。

### 1.4.1 增加视图模板

为了使用视图模板，我们需要将HomeController 中的 Index 这个 Action 的返回类型修改为 ActionResult，然后，让它像下面一样返回一个视图。

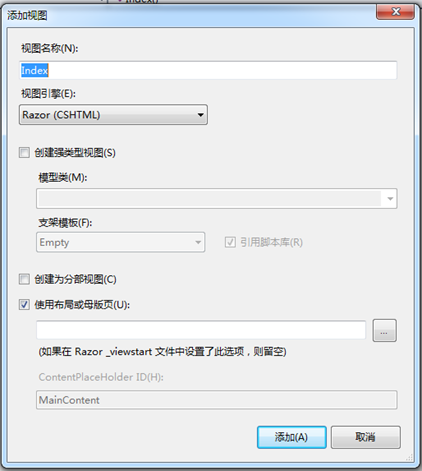
public class HomeController : Controller   
{  
 //  
 // GET: /Home/  
 public ActionResult Index()  
 {  
 return View();  
 }  
}

上面的修改表示我们将使用视图来替换掉原来的字符串，以便生成返回的结果。

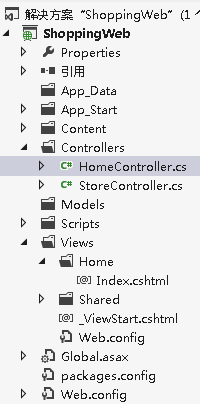
现在为我们的项目增加一个视图，为达到这个目的，我们将光标移到 Index 方法内，然后，点击鼠标的右键，在右键菜单中选择“添加视图(D)…”，这样将会弹出增加视图的对话框。



添加视图的对话框允许我们快速，简单地创建一个视图模板，默认情况下，视图的名称使用当前Action的名字。因为我们是在Index这个Aciton上添加模板，所以添加视图对话框中，视图的名字就是 Index，我们不需要修改这个名字，点击添加。



在点击添加之后，Visual Studio 将会创建一个名为 Index.cshtml的视图模板，放置在 \Views\Home 目录中，如果没有这个目录，MVC 将会自动创建它。



Index.cshtml 所在文件夹的名称和位置是很重要的，它是根据ASP.NET MVC的约定来指定的。目录名称\Views\Home，匹配的控制器就是HomeController，视图模板的名字Index，匹配将要使用这个视图的Action方法的名字。

当使用默认的约定的时候，ASP.NET MVC 允许我们不用显式设置这些名字和位置，当我们的代码如下所示的时候，将会默认寻找 \Views\Home\Index.cshtml。

public class HomeController : Controller   
{  
 //  
 // GET: /Home/  
 public ActionResult Index()  
 {  
 return View();  
 }  
}

Visutal Studio 创建并打开了Index.cshtml 视图模板，其中的内容如下：

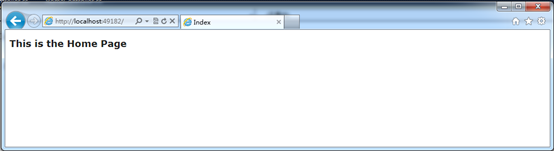
@{   
 ViewBag.Title = "Index";   
}   
   
<h2>Index</h2>

视图使用了Razor语法，这比Web Form视图引擎的语法更加简单。

前三行使用 ViewBag.Title 设置了页面的标题，我们马上就可以看到这样做的效果，但是，首先，我们我们替换一下网页的内容，将 <h1> 标记中的内容修改为 This is the Home Page 。

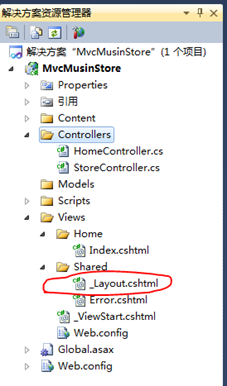
@{   
 ViewBag.Title = "Index";   
}   
   
<h2>This is the Home Page</h2>

现在，我们的首页应该变成下面的样子。



### 1.4.2 为页面的公共内容使用布局

  大多数的网站在页面之间有许多共享的内容：导航，页首，页脚，公司的Logo，样式表等等。Razor 引擎默认使用名为 \_Layout.cshtml 的布局来自动化管理，它保存在 /Views/Shared 文件夹中。



打开之后，可以看到下列内容：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title</title>

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

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

</head>

<body>

@RenderBody()

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

@RenderSection("scripts", required: false)

</body>

</html>

来自内容视图中的内容，将会被通过**@RenderBody()** 来显示，任何出现在网页中的公共内容就加入到 \_Layout.cshtml中，我们希望我们的购物网站有一个公共的页首，其中含有链接到我们的首页和商店区域的链接，所以，我们将这些内容直接添加到这个布局中。

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title</title>

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

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

</head>

<body>

<div id="header">

<h1>

YY购物网站</h1>

<ul id="navlist">

<li class="first"><a href="/" id="current">首页</a></li>

<li><a href="/Store/">商店</a></li>

<li><a href="/About">关于我们</a></li>

</ul>

</div>

@RenderBody()

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

@RenderSection("scripts", required: false)

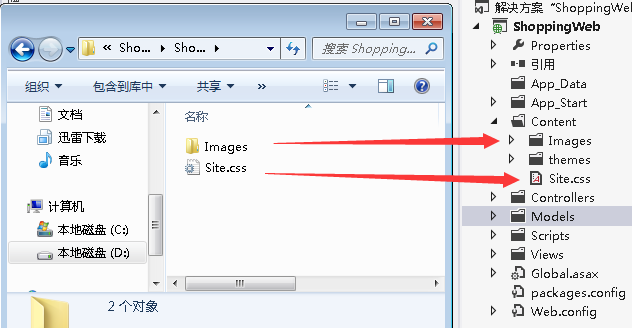
</body>

</html>

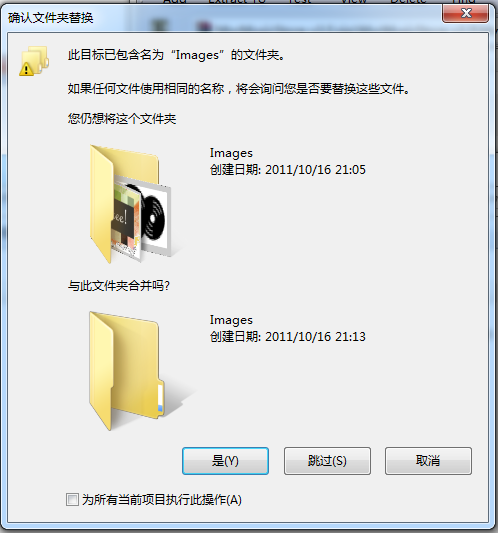
### 1.4.3 更新样式表

在创建项目使用的空项目模板中，仅仅包含很简单的用来显示验证信息的样式。我们的设计师提供了一些额外的CSS样式和图片来改进网站的观感，现在我们就使用它们。

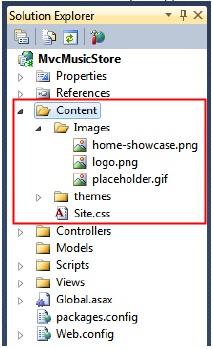
首先，[下载](http://pan.baidu.com/s/1sjuZOpZ)已经做好的Site.css和Images，将这个文件夹的Content文件夹的内容复制到项目的 Content文件夹中。



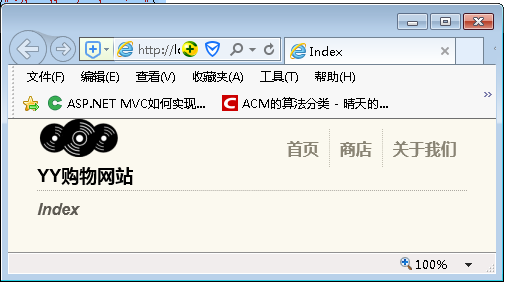
你会被询问是否需要覆盖存在的文件，选择是。



现在，网站的 Content 文件夹中的内容如下所示：



重新运行程序，现在的页面变成了这样。



我们回顾一下，什么发生了变化：HomeController的 Index 的Action方法寻找并通过 \Views\Home\Index.cshtml 模板生成输出结果，代码中是通过return View()实现的，因为默认的命名约定，Index这个Action方法将会默认使用Index视图输出。

而Index视图使用了我们的\_Layout.cshtml模板，所以，欢迎信息被包含在标准的HTML布局中。

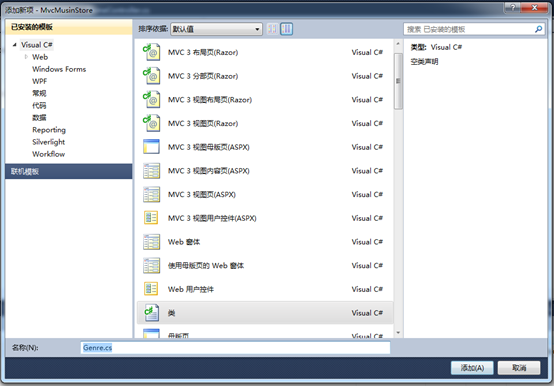
### 1.4.4 使用模型为视图传递信息

仅仅使用硬编码的HTML不能创建令人感兴趣的网站，创建动态网站，我们需要从控制器的Action传送信息给视图模板。

在MVC模式中，术语Model表示应用表现的数据，通常，模型对象用来表示数据库中保存在表中的数据，也不一定如此。

控制器的Action方法通过返回的ActionResult可以传送模型对象给视图。这就允许控制器可以将所有生成回应需要的数据打包，然会传送给视图模板，以便生成适当的HTML回应，在 Action 方法中可以很容易理解，让我们开始吧。

首先，我们将创建一些模型类来表示商店中的商品分类和商品，从创建类型Category类开始，在项目中，右击模型Models文件夹，然后选择增加类选项，然后命名为Category.cs。



在新创建的类中增加一个属性。

public class Category   
{   
 public string Name { get; set; }   
}

注意：这里的{ get; set; }是 C# 的自动属性特性，这使得我们不需要在创建属性的时候，先创建一个成员字段。

现在，用同样的方法创建商品类Product，它有两个属性：Name和Category。

public class Product   
{   
 public string Name { get; set; }   
 public Category Category { get; set; }   
}

现在，我们修改StoreController通过模型来使视图显示动态信息，为了演示方便，我们定义专辑基于一个唯一的标识Id, 我们将在视图中显示这个标识。

我们从修改Details这个Action使得可以显示单个的专辑开始，在StoreController.cs的开始部分增加一些using语句来包含ShoppingWeb.Models命名空间，这使得我们不用总是输入这个命名空间。

using System;   
using System.Collections.Generic;   
using System.Linq;   
using System.Web;   
using System.Web.Mvc;   
  
using ShoppingWeb.Models;

然后，我们更新Details Action，使得返回ActionResult类型的结果而不是字符串，就像在 HomeController中的Index方法中做得一样。

public ActionResult Details(int id)

现在，修改方法的处理逻辑，返回一个专辑对象到视图中，在这个项目最后，显示的数据将会来自数据库，现在我们仅仅填充一些数据而已。

public ActionResult Details(int id)

{

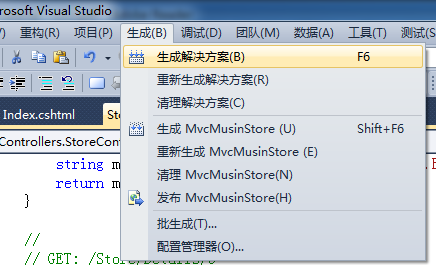
var product = new Product { Name = "Product " + id };

return View(product);

}

如果你对c#不太熟悉，可能你会认为使用var定义变量使用了迟绑定，这是不正确的，C#编译器使用赋予变量的值来推定变量的类型，所以，实际上变量的类型就是Product类型，因此不仅在编译时， Visual Studio的代码编辑器中也会有类型支持。

下面创建一个使用专辑来生成 HTML 的模板，在这样做之前，我们需要编译项目，以便增加视图的对话框知道我们新创建的专辑类型。你可以通过菜单“生成”的“生成解决方案”来完成。

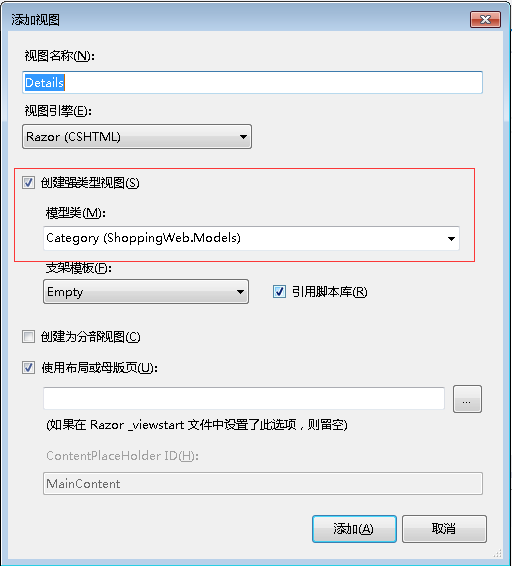


另外，也可以通过热键F6来编译项目。

已经可以创建视图模板了，在 Details 方法中右键选择“增加视图…”。



像以前一样，我们看到创建视图的对话框，不一样的是，我们要选中“创建强类型视图”，然后在下面的列表中选择“Product”类，这样视图将会期望得到一个Category类型的对象。



在点击增加之后，我们的视图模板 \Views\Store\Details.cshtml 被创建了，其中包含的如下的代码：

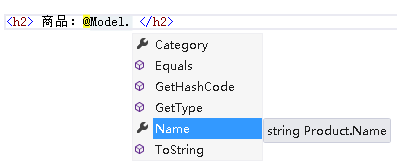
@model ShoppingWeb.Models.Product   
  
@{   
 ViewBag.Title = "Details";   
}   
   
<h2>Details</h2>

注意第一行，表示视图使用强类型的Product类。Razor视图引擎理解传送来的Product对象，所以我们可以容易地访问模型的属性，在Visual Studio中得到智能感知的帮助。

更新 <h2> 标记，使得可以显示专辑的 Title 属性。

@model ShoppingWeb.Models.Category   
  
@{   
 ViewBag.Title = "Details";   
}   
   
<h2>商品: @Model.Name</h2>

注意，智能感知使得可以提示 Album 的属性和方法。



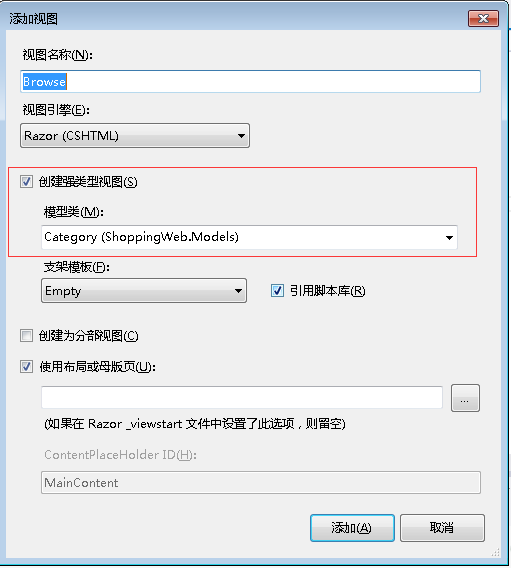
再次运行并访问 /Store/Details/5，可以得到下面的结果。



现在，我们继续修改 Browse 方法，更新方法返回 ActionResult 类型的结果，修改方法的处理，返回一个Category类型的对象实例。

public ActionResult Browse(string category)   
{   
 var categoryModel = new Category { Name = category };   
 return View(categoryModel);   
}

在方法上右击，选择“增加视图…”，增加一个强类型的视图。



修改<h2>标记显示Category的信息。

@model ShoppingWeb.Models.Category   
@{   
 ViewBag.Title = "Browse";   
}   
   
<h2>您正在浏览商品类型: @Model.Name</h2>

重新运行，访问/Store/Browse?Category=食品，可以看到如下的显示。如果出现中文乱码，可以网站根目录下的Web.Config文件中<System.Web>节，加入下面代码。

……

<globalization requestEncoding="gb2312" responseEncoding="gb2312" />

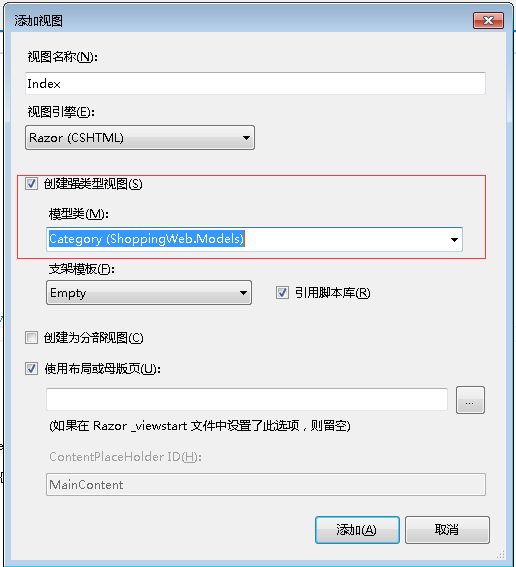
</system.web>



最后，将Index也修改为强类型的视图，显示所有唱片的类别，我们使用Genre的一个列表，而不是单个的Genre对象。

public ActionResult Index()   
{   
 var categories = new List<Category>   
 {   
 new Category { Name = "食品"},   
 new Category { Name = "服装"},   
 new Category { Name = "玩具"}   
 };   
 return View(genres);   
}

创建一个强类型的视图。



首先，我们将期望得到多个Category对象而不是一个，将第一行修改为如下内容。

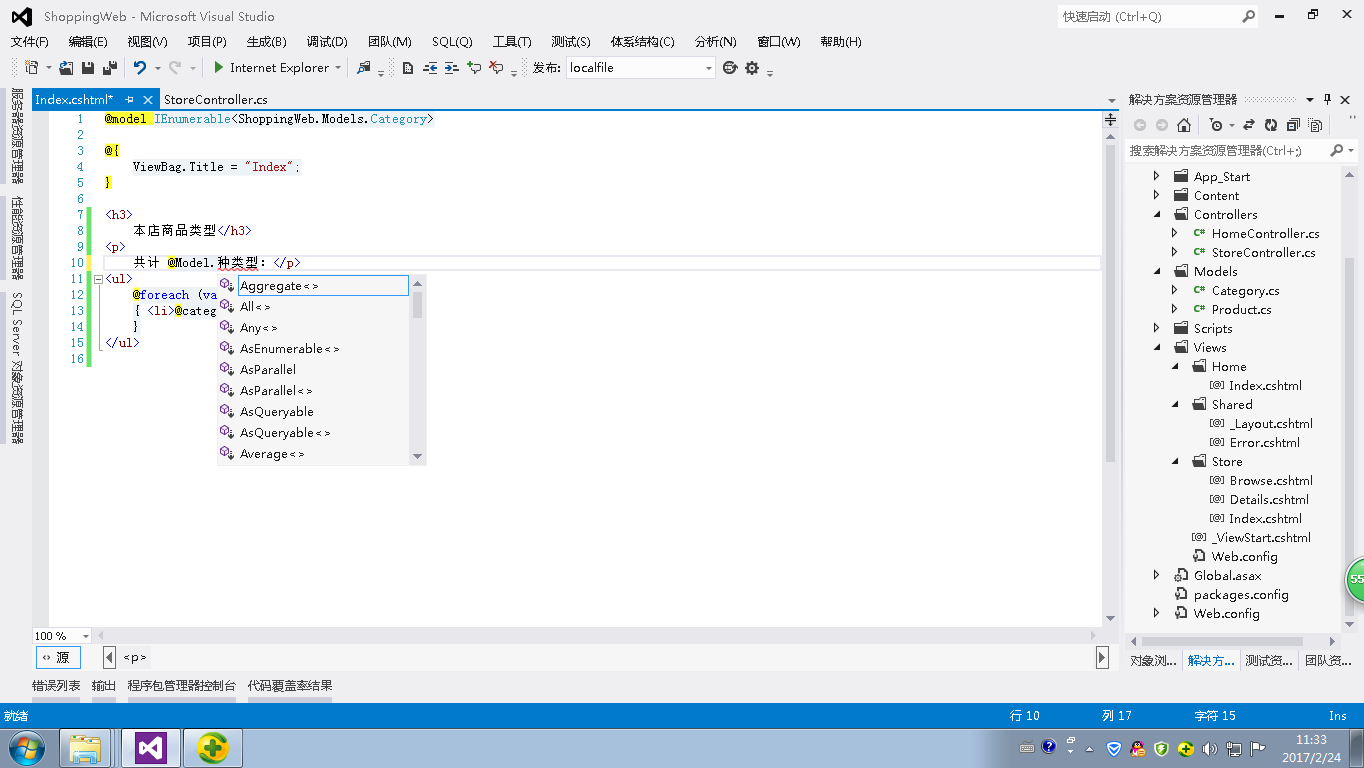
@model IEnumerable<ShoppingWeb.Models.Category>

这告诉视图引擎模式是一个包含多个Category对象的集合，我们使用 IEnumerable<Category> 而不是 List<Category>，因为这样更通用，可以允许我们在以后改变集合为任何实现 IEnumerable 接口的集合。

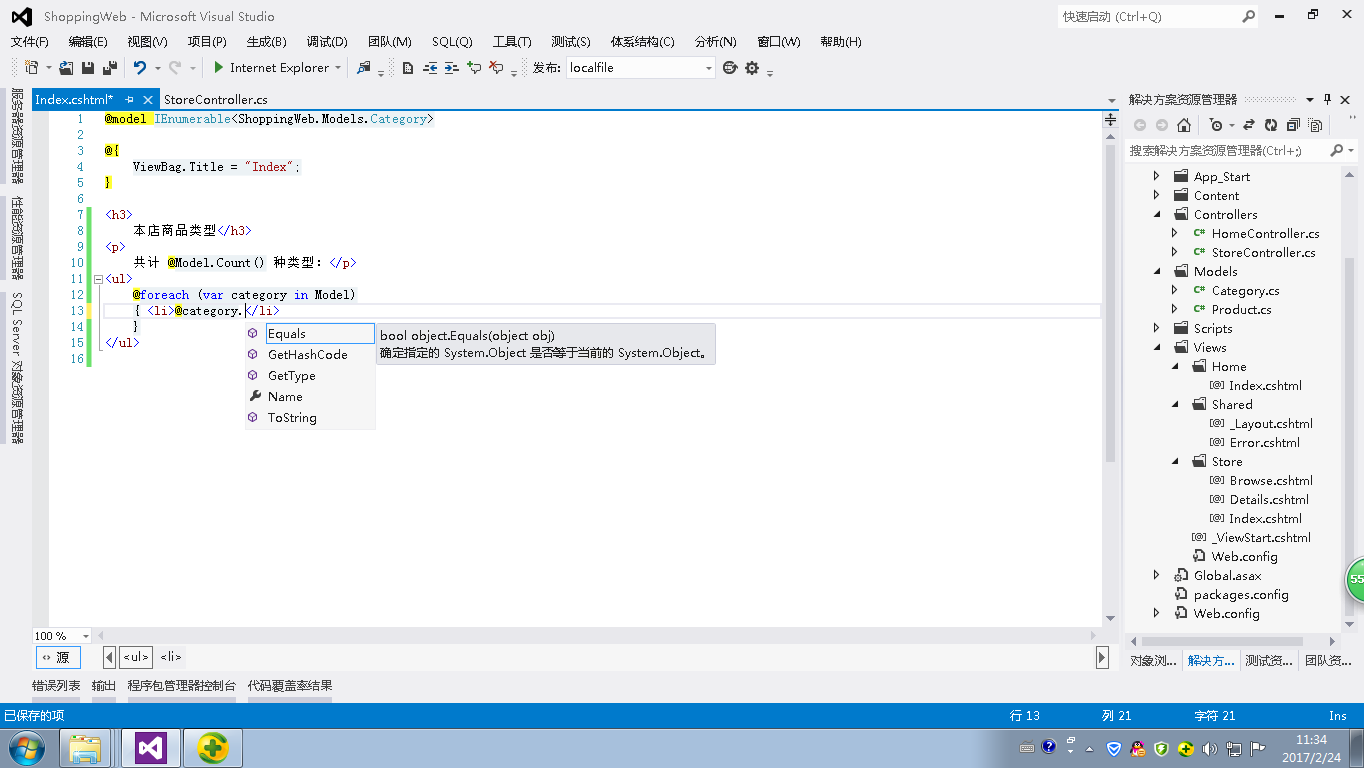
现在，我们遍历集合中的Category对象进行处理。

@model IEnumerable<ShoppingWeb.Models.Category>   
@{   
 ViewBag.Title = "Store";   
}   
<h3>   
 本店商品类型</h3>   
<p>   
 共计@Model.Count()种类型：</p>   
<ul>   
 @foreach (var category in Model)   
 { <li>@category.Name</li>   
 }   
</ul>

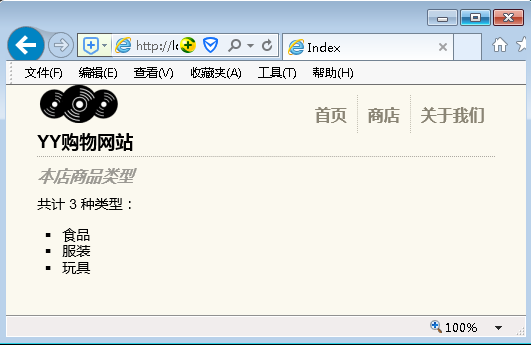
注意，此时有完全的智能提示。



在foreach循环中，也同样有提示。



再次运行程序，我们可以看到如下的结果。



### 1.4.5 增加页面之间的链接

现在，我们的/Store可以使用纯文本列出当前的分类名称，下一步，我们将这些纯文本替换成可以链接到浏览分类的链接/Store/Browse上，这样，当用户点击商品类型“玩具”将会被导航到 /Store/Browse?category=toy 的 URL 地址上。我们再次更新\Views\Store\index.cshtml 视图页面，先看一下，一会我们还会再次改进。

<ul>   
 @foreach (var genre in Model)   
 {   
 <li><a href="/Store/Browse?category=@category.Name">@Category.Name</a></li>   
 }   
</ul>

这样就可以工作了，但是这里使用了硬编码的字符串，如果我们希望修改控制器的名称，那么，我们就要找到所有这样的位置进行修改

更好的处理方式是使用HTMLHelper的助手方法，ASP.NET MVC包含了一个HTML的助手类，其中的方法专门用于在视图模板中完成多种常见的任务，其中的Html.ActionLink()助手方法就是常用的一个，这使得可以容易地创建<a>，包括关于链接的一些细节处理，像地址需要进行URL编码之类。

Html.ActionLink()有多个重载用于多种情况，在简单的情况下，你只需要提供提示的文本，以及指向的Action方法即可，在客户端，举个例子，我们希望链接到/Store的 Index 方法，提示文本为Go to the Store Index，那么下面的代码就可以。

@Html.ActionLink("Go to the Store Index", "Index")

注意：在这个例子中，我们不需要再特别指定控制器的名称，因为我们在使用同一个控制器的不同 Action 方法。

我们的链接还需要一些参数，我们可以使用另外一种重载来传递三个参数。

1. 链接的提示文本，这里显示分类的名称

2. 动作的名称，Browse

3. 路由参数，提供名字category和值，category的名字

合在一起，下面就是需要写在视图模板中的内容

<ul> @foreach (var category in Model)  
{   
 <li>@Html.ActionLink(category.Name, "Browse", new { category = category.Name })</li>  
}  
</ul>

现在，当我们运行程序，访问/Store的时候，将会看到一个分类的列表，每一个分类都是一个超级链接，当点击链接的时候，将会被导航到/Store/Browse?category=[category]的地址

页面中生成的分类链接如下：

<ul>

<li><a href="/store/Browse?category=%E9%A3%9F%E5%93%81">食品</a></li>

<li><a href="/store/Browse?category=%E6%9C%8D%E8%A3%85">服装</a></li>

<li><a href="/store/Browse?category=%E7%8E%A9%E5%85%B7">玩具</a></li>

</ul>