Asp.Net MVC4 入门指南

目录

入门介绍

译者注：

本系列共 9 篇文章，翻译自 Asp.Net MVC4 官方教程，由于本系列文章言简意赅，篇幅适中，从一个示例开始讲解，全文最终完成了一个管理影片的小系统，非常适合新手入门 Asp.Net MVC4，并由此开始开发工作。9 篇文章为：

1. Asp.Net MVC4 入门指南

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/intro-to-aspnet-mvc-4

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2012/11/01/2749906.html

2. 添加一个控制器

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-a-controller

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2012/11/02/2751015.html

3. 添加一个视图

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-a-view

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2012/11/06/2756711.html

4. 添加一个模型

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-a-model

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2012/12/17/2821495.html

5. 从控制器访问数据模型

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/accessing-your-models-data-from-a-controller

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2013/01/11/2855935.html

6. 验证编辑方法和编辑视图

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/examining-the-edit-methods-and-edit-view

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2013/01/24/2874622.html

7. 给电影表和模型添加新字段

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-a-new-field-to-the-movie-model-and-table

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2013/02/26/2933105.html

8. 给数据模型添加校验器

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-validation-to-the-model

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2013/03/05/2944030.html

9. 查询详细信息和删除记录

原文地址：http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/examining-the-details-and-delete-methods

译文地址：http://www.cnblogs.com/powertoolsteam/archive/2013/03/07/2948000.html

10. 第三方控件 ComponentOne Studio for ASP.NET Wijmo 在 MVC4 下的应用

原文地址：http://www.cnblogs.com/powertoolsteam/archive/2013/05/09/3068699.html

---

本教程将为您讲解使用微软的 Visual Studio Express 2012 或 Visual Web Developer 2010 Express Service Pack 1 来建立一个 ASP.NET MVC4 Web 应用程序所需要的基础知识。建

议您使用 Visual Studio 2012，你将不再需要安装任何组件，来完成此教程。如果您使用的是 Visual Studio 2010，您必须安装下面的组件。您可以通过点击下面的链接，来安装所需的所有组件：

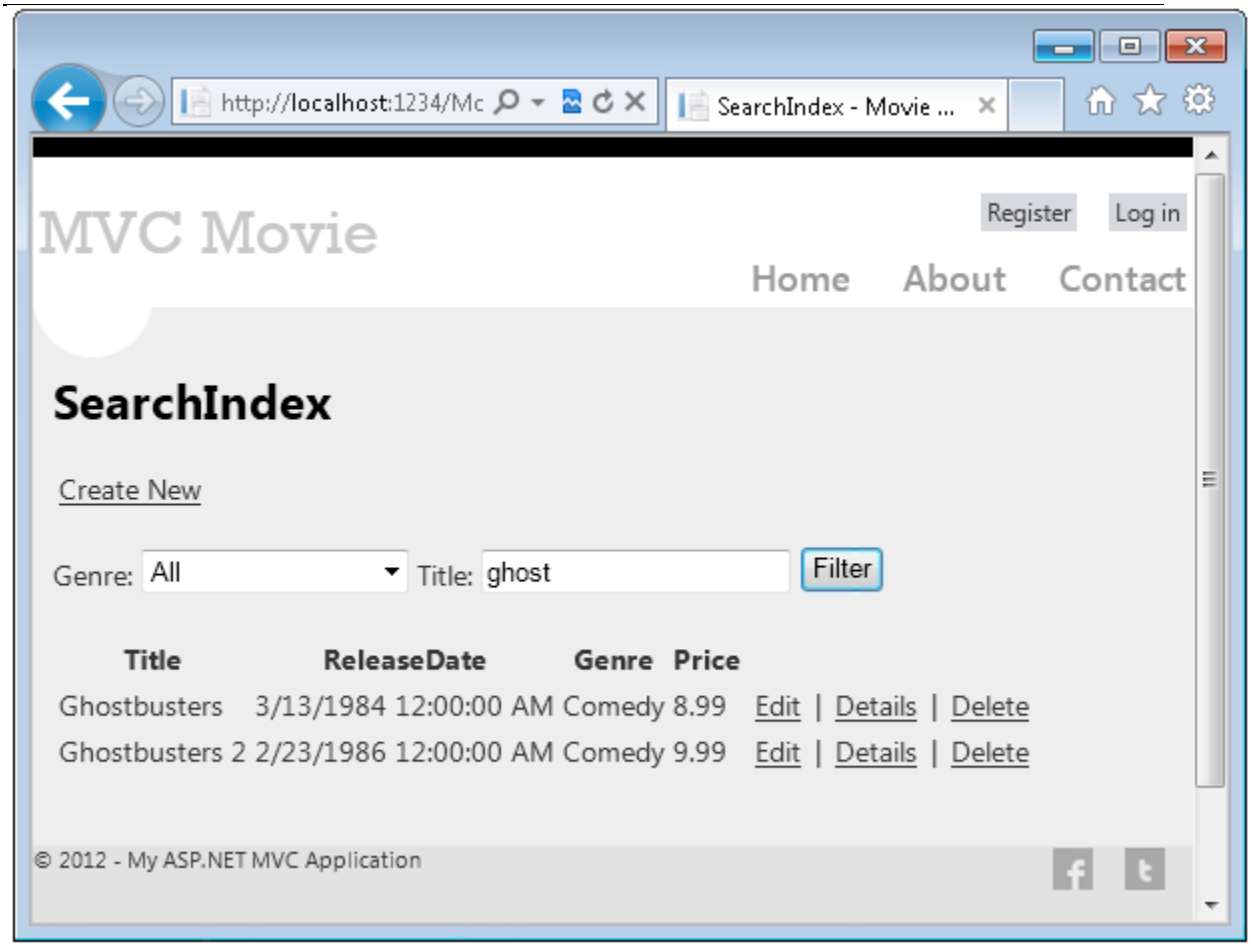
- Visual Studio Web Developer Express SP1 prerequisites
- WPI installer for ASP.NET MVC 4
- LocalDB
- SSDT

如果您使用的是 Visual Studio 2010 而不是 Visual Web Developer 2010，需要安装 WPI installer for ASP.NET MVC 4 和 Visual Studio 2010 prerequisites

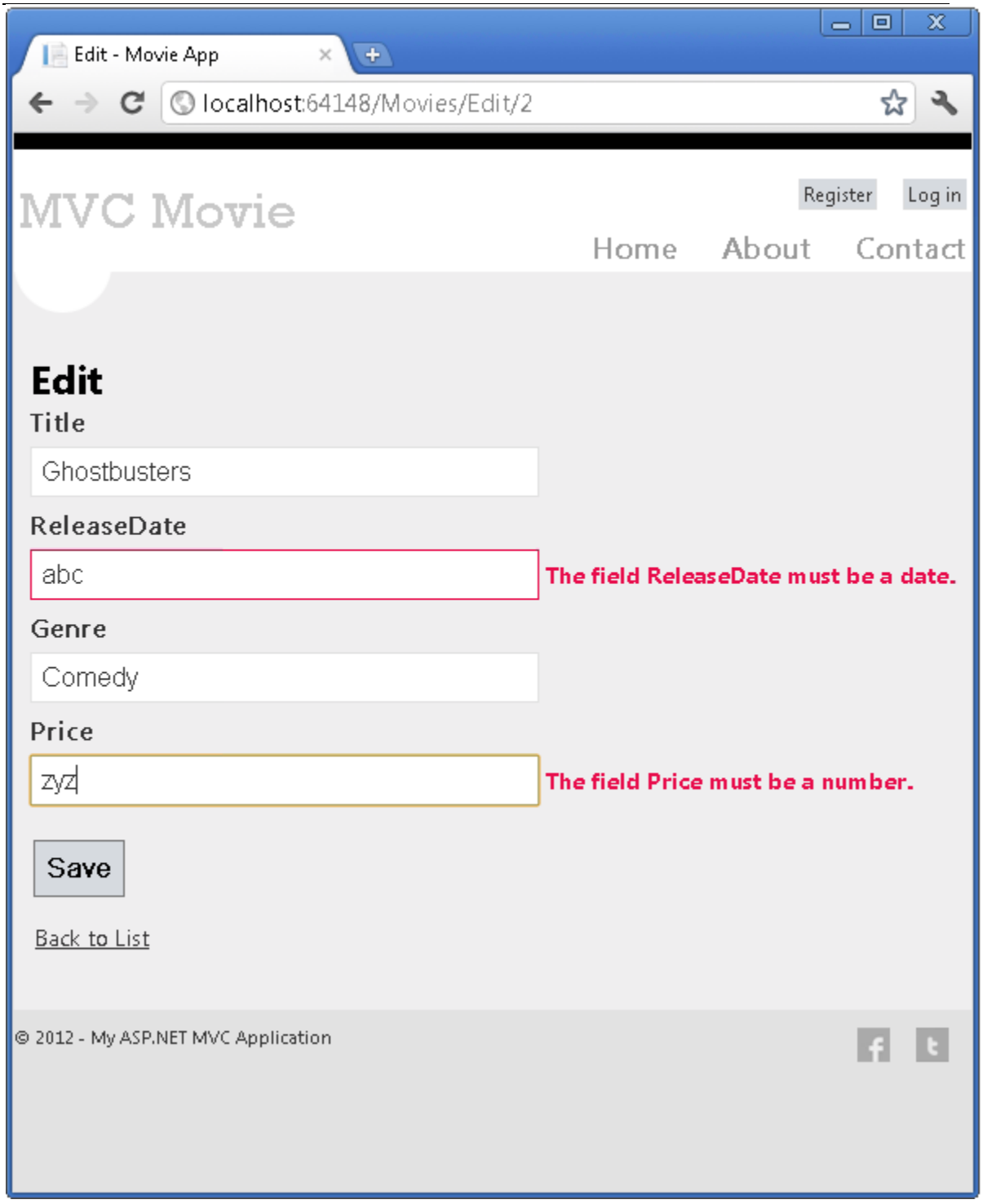
本文的 C#示例源代码，是一个 Visual Web Developer Project： 下载本文 C＃示例源代码。

本示例将构建什么样的应用程序？

您将实现一个简单的电影列表应用程序，此程序将支持创建、 编辑、 搜索和从数据库中选取出电影列表的功能。下面是您将构建的应用程序的两个截屏。它包括显示选取自数据库的电影列表页面：



该应用程序还允许您添加、编辑和删除电影，以及显示单个记录的详细信息。所有的用户数据输入场景都包含了数据验证逻辑，以确保存储在数据库中的数据都是正确的。

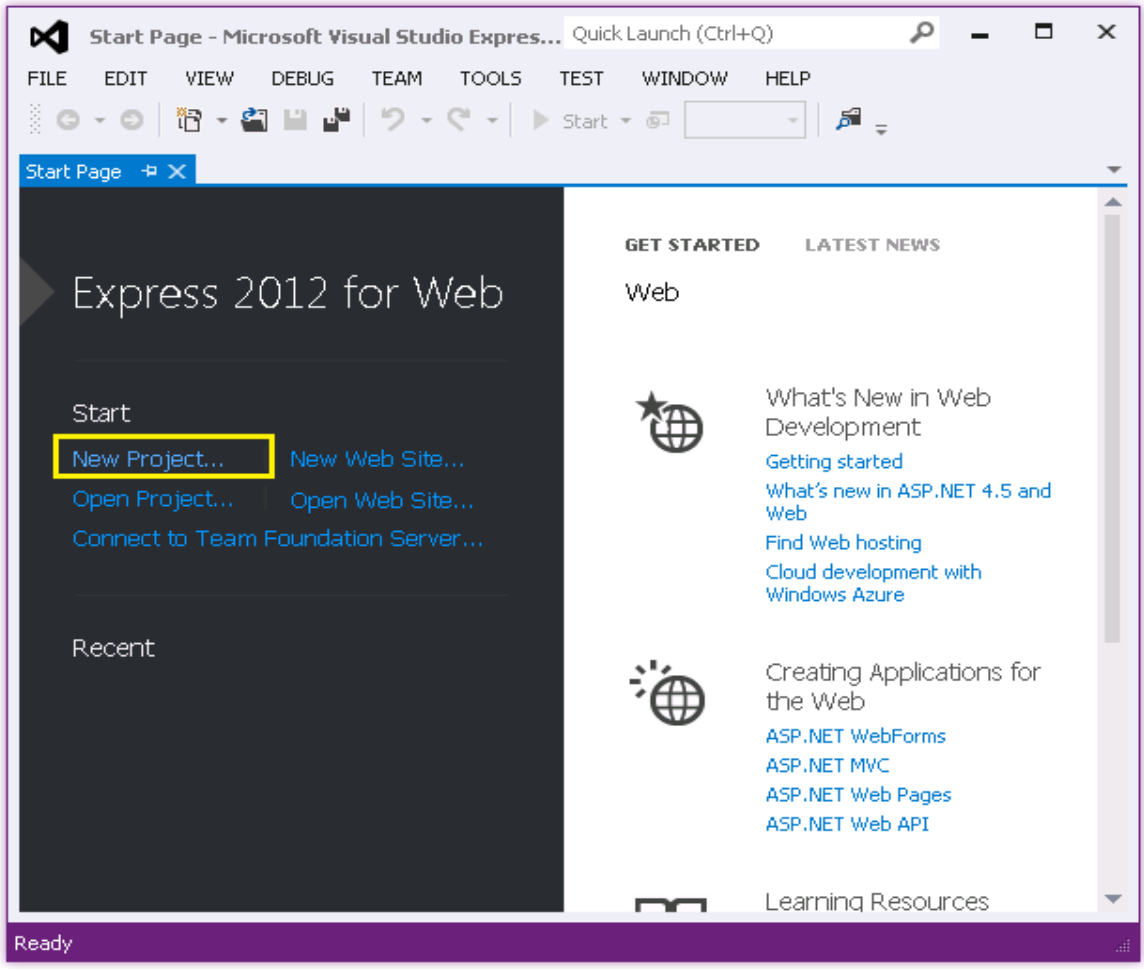


入门

运行 Visual Studio Express 2012 或 Visual Web Developer 2010 Express 来开始这个示例，在这个系列中大多都使用了 Visual Studio Express 2012 的屏幕截图，同时你也可以使用 Visual

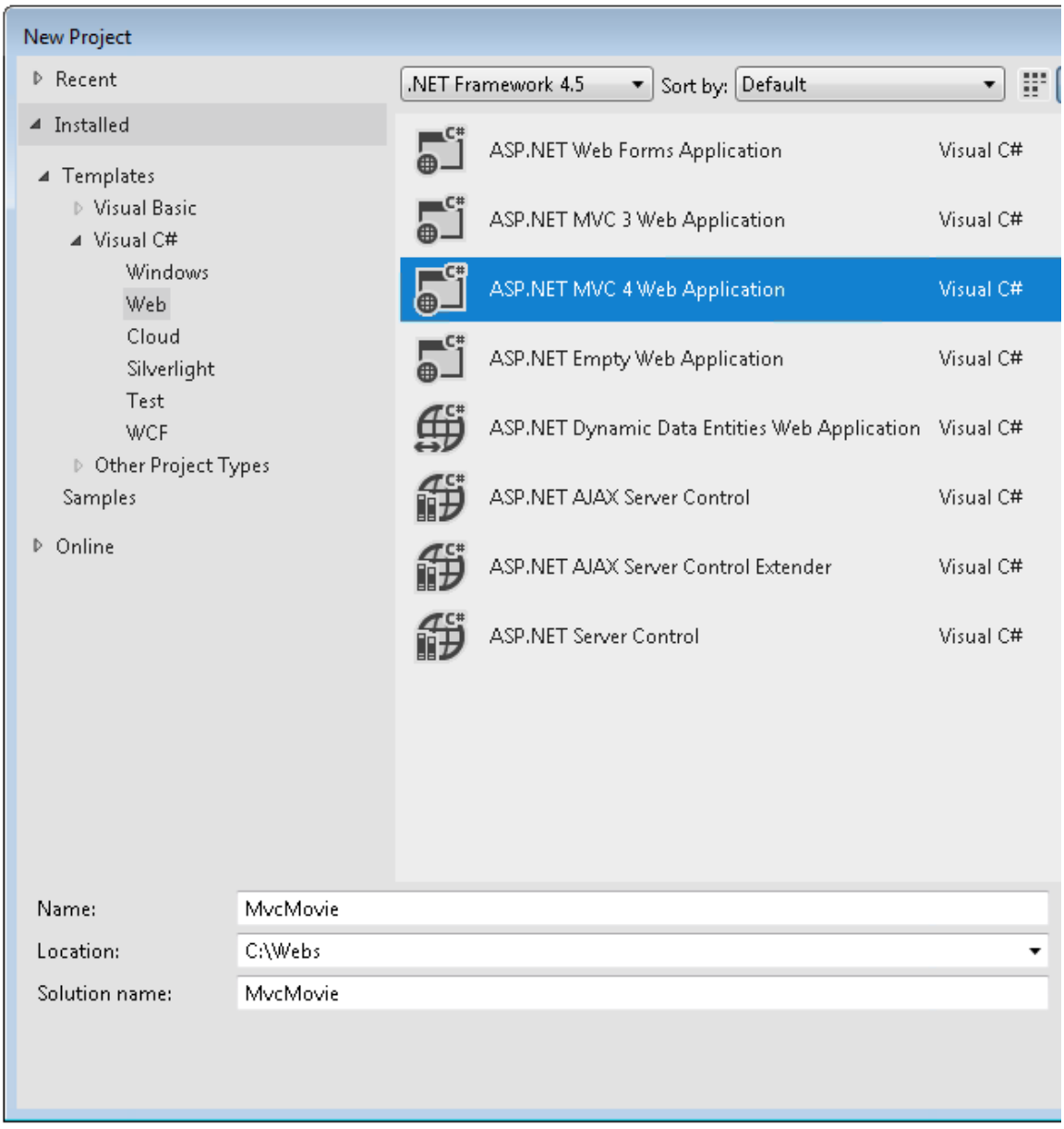
Studio 2010/SP1，Visual Studio 2012, Visual Studio Express 2012 或 Visual Web Developer 2010 Express 来学习完成这个教程。从“开始”页面中，选择“新建项目”。

Visual Studio 是一个 IDE 集成开发环境。就像您使用 Microsoft Word 来编写文档，你可以使用集成开发环境（IDE）来创建一个应用程序。在 Visual Studio 中的一个顶部工具栏中显示了各种不同的选项来供您使用。在 IDE 中还有一个菜单，提供了另一种方式来执行任务。（例如，您可以不从“开始”页面中，选择“新建项目”，您可以使用该菜单，然后选择“文件”>“新建项目”）

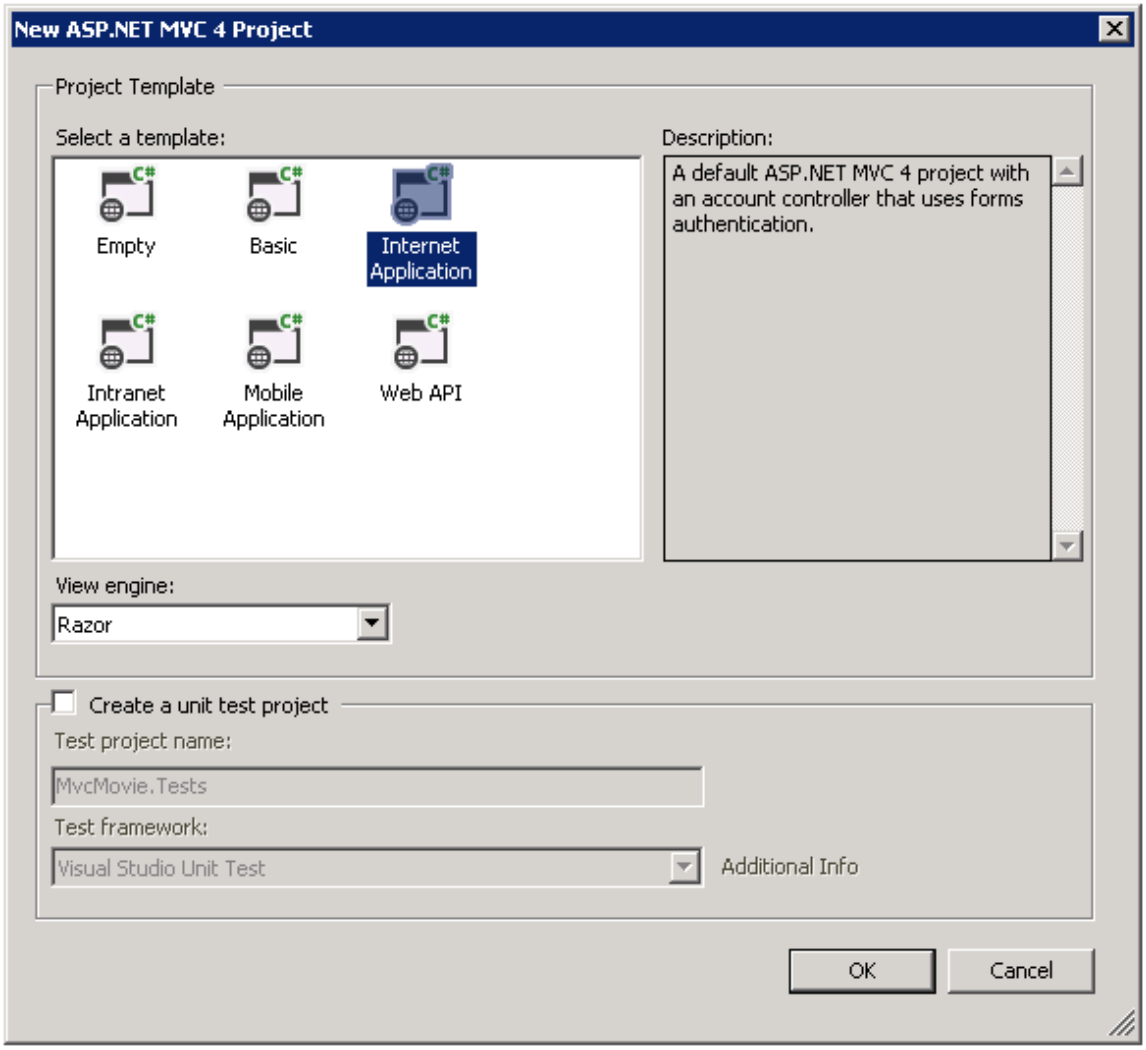


创建您的第一个应用程序

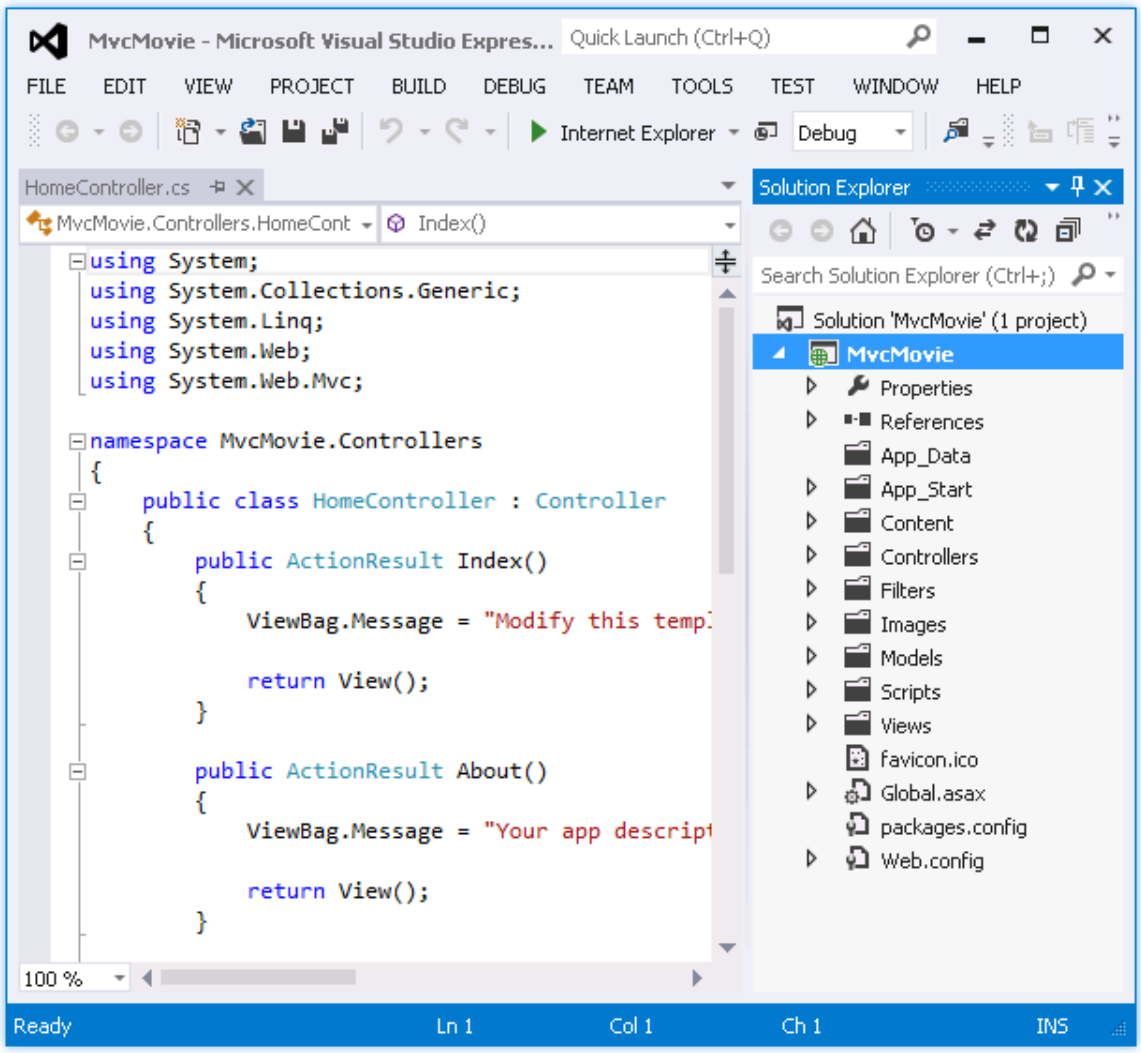
您可以使用 Visual Basic 或 C# 作为编程语言来创建您的应用程序。请在左侧选择 Visual C# ，然后选择 **ASP.NET MVC 4 Web 应用程序。**命名您的工程为"MvcMovie"，然后单击**确定**.



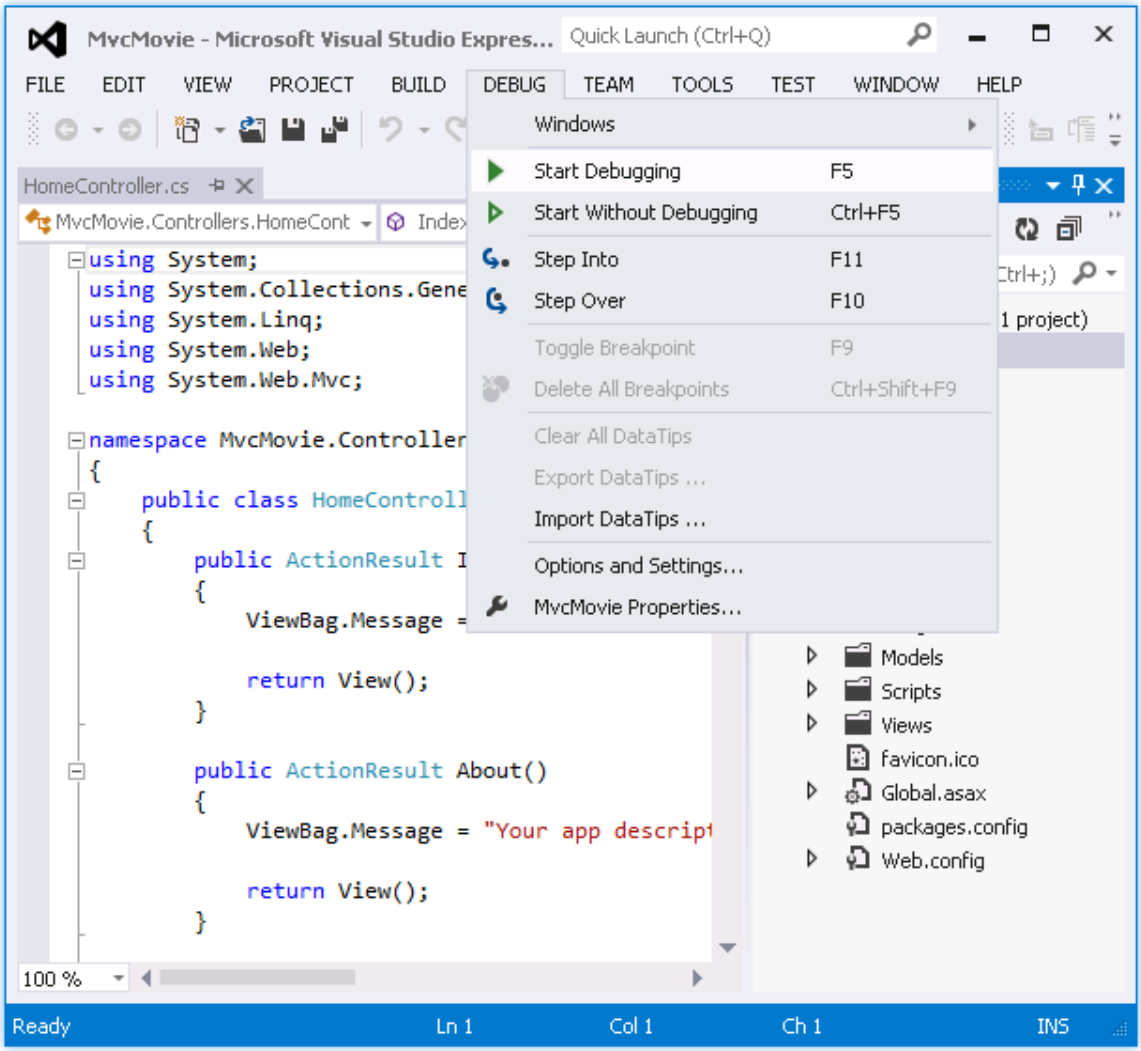
在新的 ASP.NET MVC 4 项目对话框中，选择互联网应用程序。使用 Razor 作为默认视图引擎。



单击**确定**。Visual Studio 刚刚创建的 ASP.NET MVC 项目使用了默认的模板，所以在当前的工程中您不需要做任何事情！这是一个简单的"Hello World ！"工程，并且这也是您开始“MvcMovie”工程的好地方。



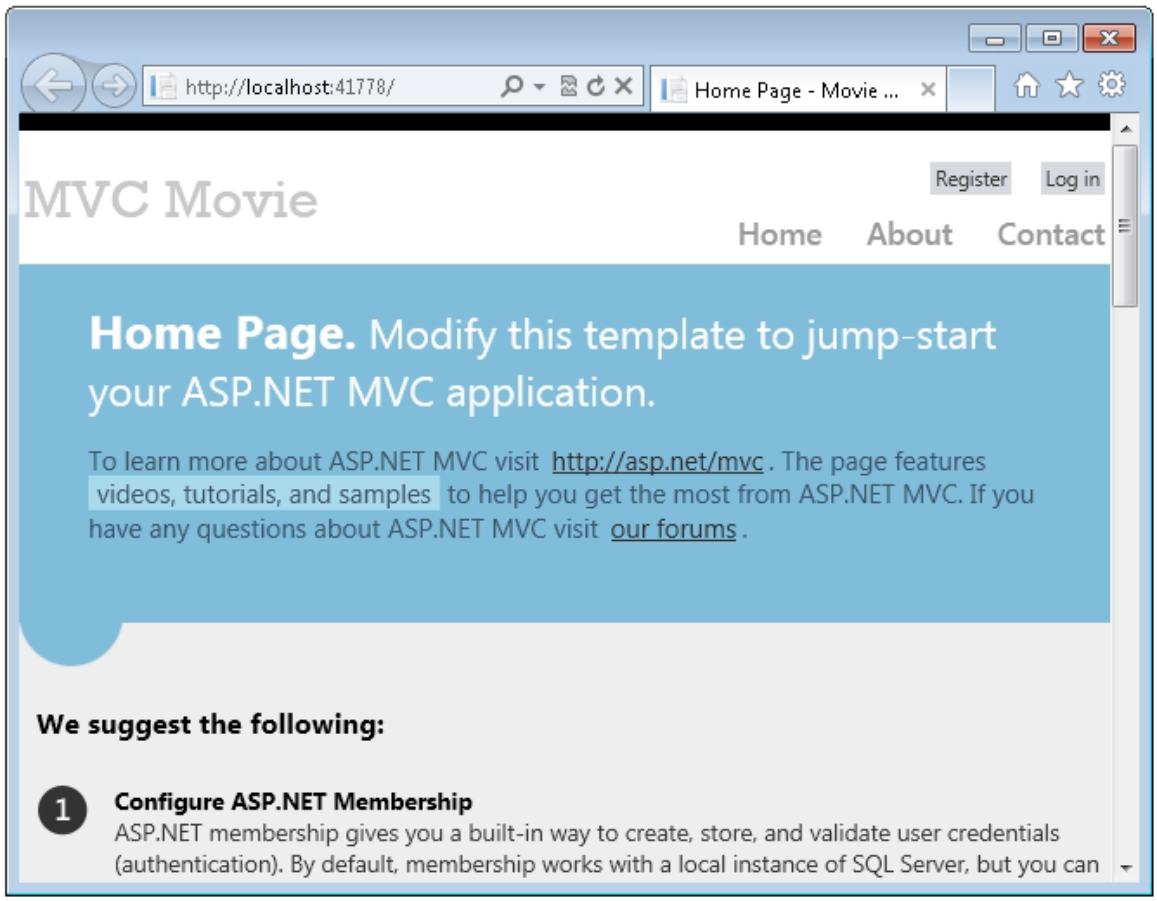
从**调试**菜单中，选择**启动调试**.



请注意您也可以使用键盘的快捷键 F5 来启动调试。

F5 使 Visual Studio 启动 IIS Express 并运行 Web 应用程序。然后 Visual Studio 会启动浏览器并打开应用程序的主页面。请注意，在浏览器的地址栏中会显示 localhost 而不是像 example.com 这样的地址。这是因为 localhost 总是会被解析为您自己的本地计算机，在这种情况下，这正是您你刚刚建立的应用程序。当 Visual Studio 运行一个 Web 工程时，会使用一个随机端口的 Web 服务。在下面的图片中，端口号是 41788。当您运行该应用程序时，

您可能会看到一个不同的端口号。



在默认模板页面的右边，为您提供了“主页(Home)”，“关于(About)”和“联系(Contact)”页面。它还提供了注册和登录功能，并提供了 Facebook 和 Twitter 的链接。接下来的一步是修改此默认应用程序，并了解一些关于 ASP.NET MVC 的知识。关闭浏览器，让我们修改一些源代码吧。

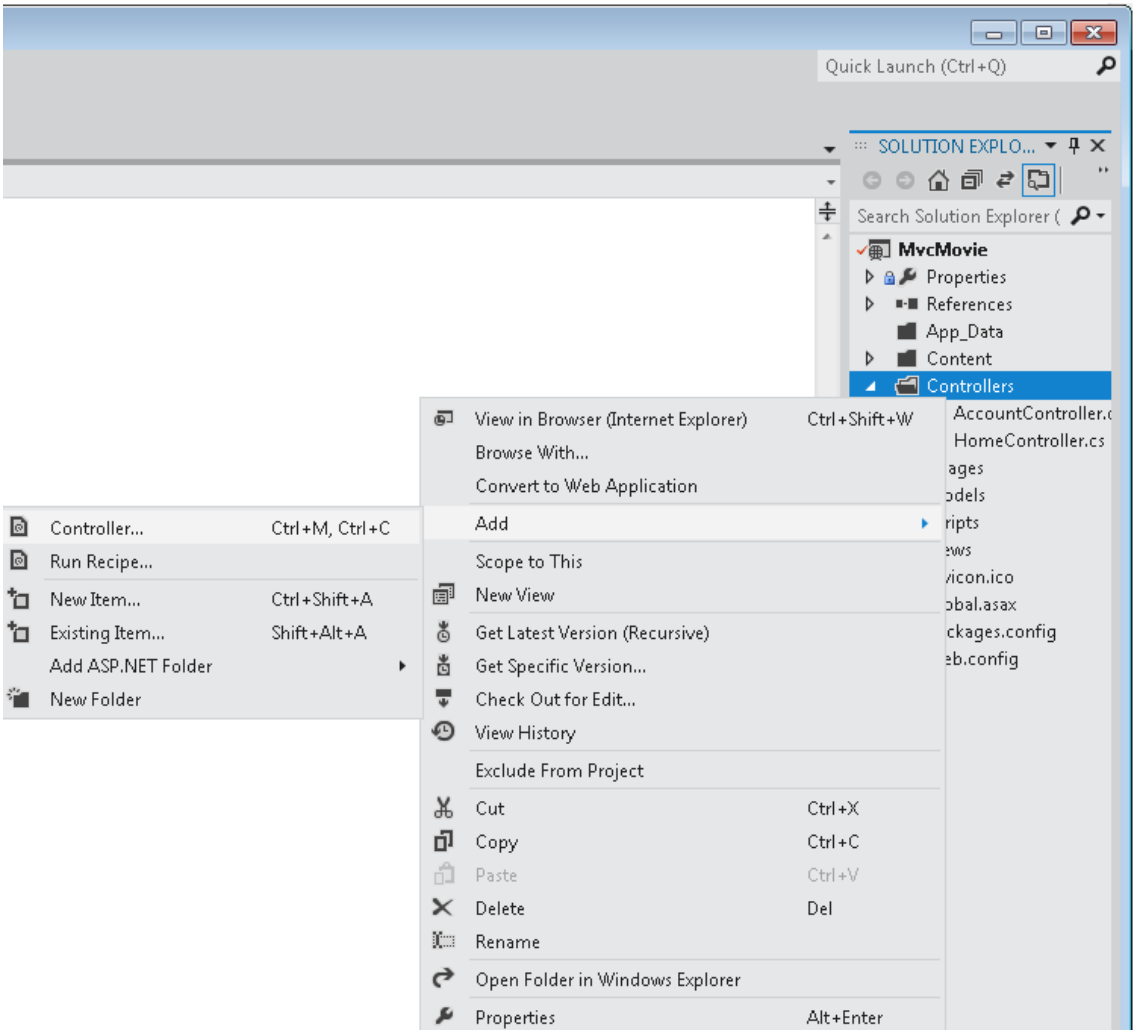
添加一个控制器

MVC 代表: 模型-视图-控制器 。MVC 是一个架构良好并且易于测试和易于维护的开发模式。基于 MVC 模式的应用程序包含：

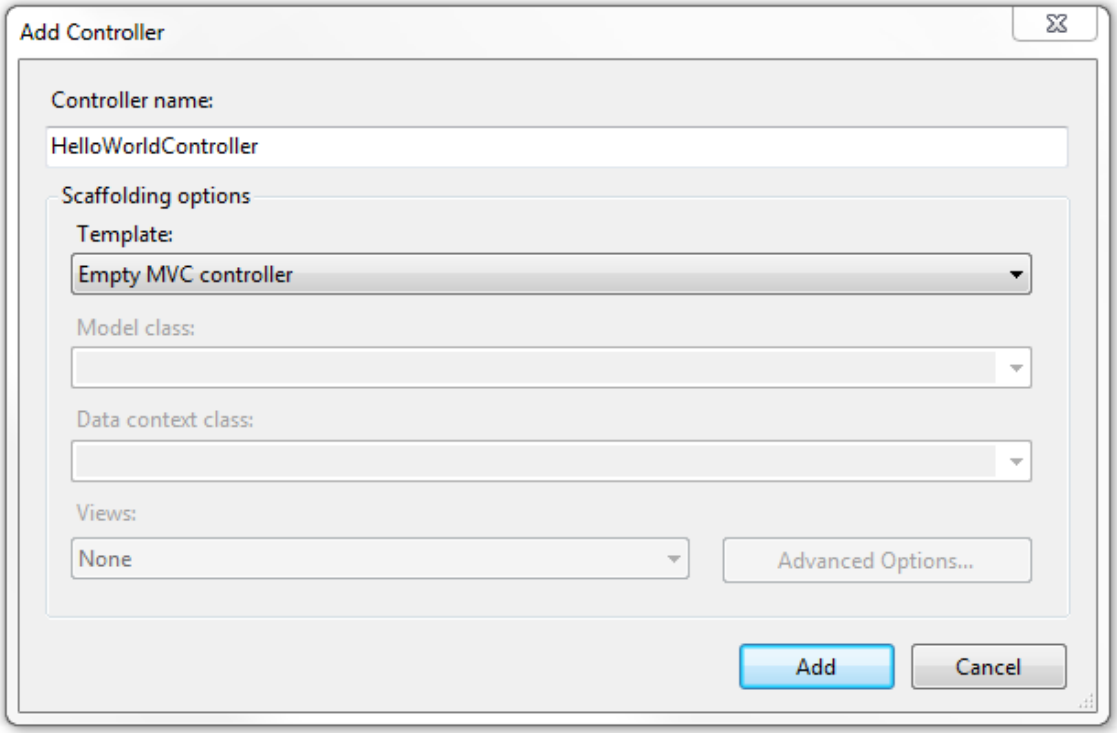
- **M**odels： 表示该应用程序的数据并使用验证逻辑来强制实施业务规则的数据类。
- **V**iews： 应用程序动态生成 HTML 所使用的模板文件。
- **C**ontrollers： 处理浏览器的请求，取得数据模型，然后指定要响应浏览器请求的视图模板。

本系列教程，我们将覆盖所有这些概念，并告诉您如何使用它们来构建应用程序。

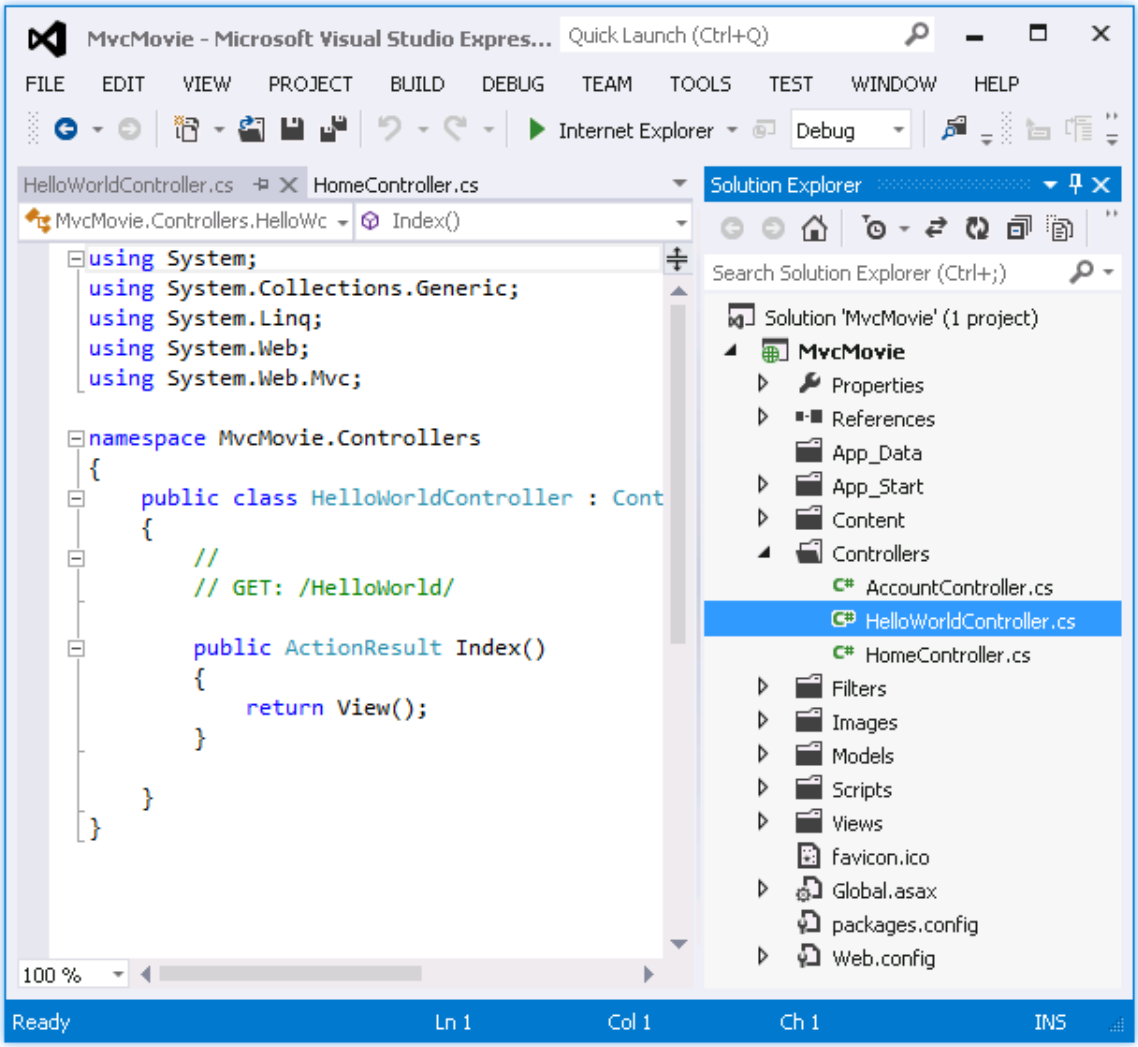
首先，让我们创建一个控制器类。在**解决方案资源管理器中**，用鼠标右键单击控制器文件夹，然后选择“**添加控制器**“。



命名新的控制器为“HelloWorldController”。保留默认的模板为“Empty MVC controller”，并单击“添加“。



请注意，在**解决方案资源管理器中**会创建一个名为 HelloWorldController.cs 的新文件。该文件会被 IDE 默认打开。

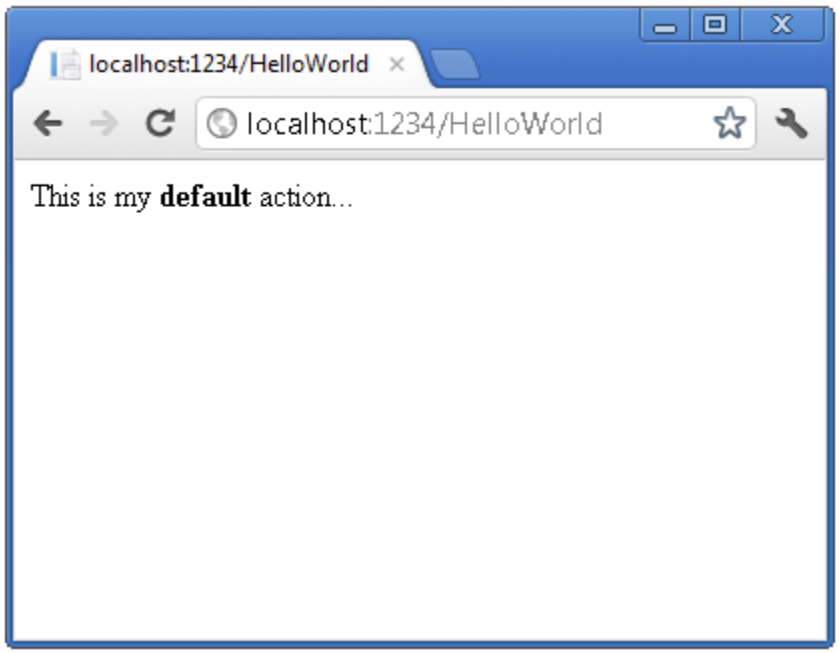


用下面的代码替换该文件中的内容。

```
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/

        public string Index()
        {
            return "This is my <b>default</b> action...";
        }
```

```
        //
        // GET: /HelloWorld/Welcome/

        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

在这个例子中控制器方法将返回一个字符串的 HTML。本控制器被命名 `HelloWorldController` 代码中的第一种方法被命名为 `Index`。让我们从浏览器中调用它。运行应用程序（按 F5 或 CTRL + F5）。在浏览器的地址栏中输入路径“HelloWorld“。（例如，在下面的示例中: http://localhost:1234/HelloWorld）页面在浏览器中的表现如下面的截图。在上面的方法中，代码直接返回了一个字符串。你告诉系统只返回一些 HTML，系统确实这样做了！

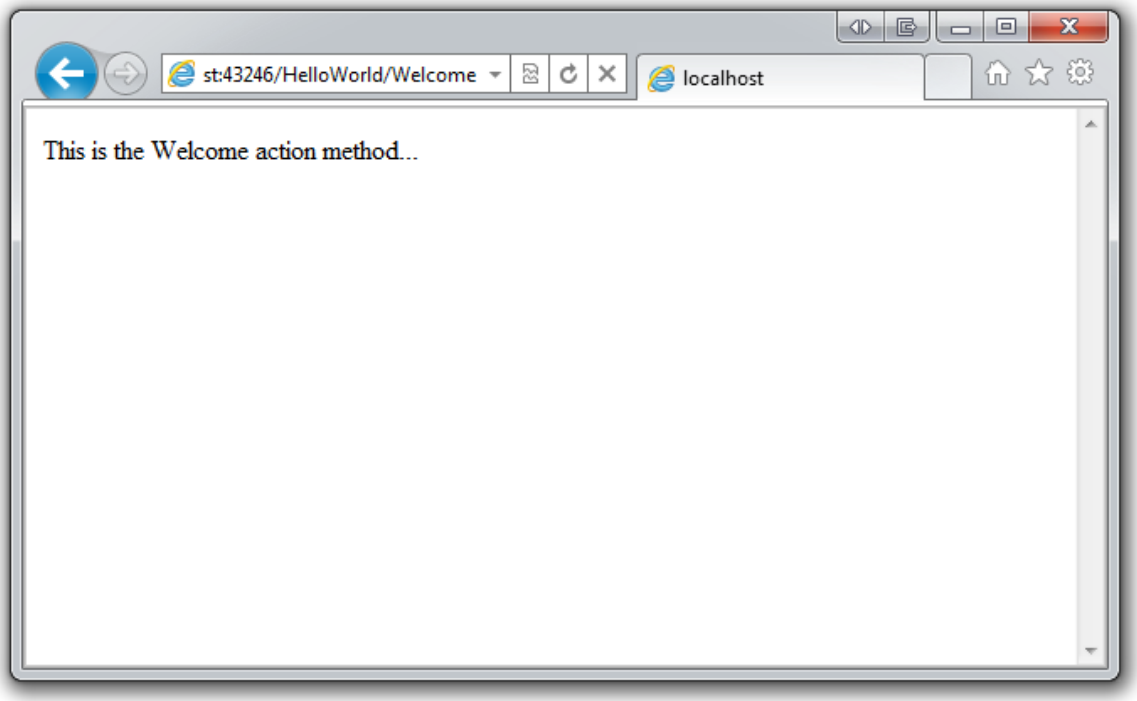


根据传入的 URL，ASP.NET MVC 调用不同的控制器类（和它们之中不同的操作方法）。使用 ASP.NET MVC 默认的 URL 路由逻辑格式，以确定哪些代码会被调用:

/[Controller]/[ActionName]/[Parameters]

第一部分的 URL 确定那个控制器类会被执行。因此 */HelloWorld* 映射到 HelloWorldController 控制器类。第二部分的 URL 确定要执行控制器类中的那个操作方法。因此 */HelloWorld/Index*，会使得 HelloWorldController 控制器类的 Index 方法被执行。请注意，我们只需要浏览 */HelloWorld* 路径，默认情况下会调用 Index 方法。如果没有明确的指定操作方法，Index 方法会默认的被控制器类调用。

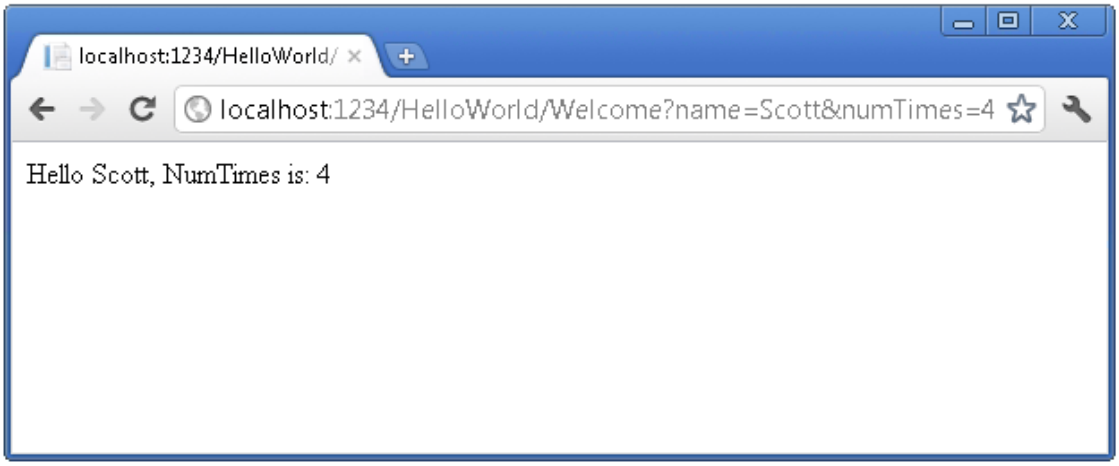
浏览 *http ://localhost:xxxx/HelloWorld/Welcome*。Welcome 方法会被运行并返回字符串 :"This is the Welcome action method...”。 默认的 MVC 映射为 /[Controller]/[ActionName]/[Parameters] 对于这个 URL，控制器类是 HelloWorld ，操作方法是 Welcome，您还没有使用过 URL 的[Parameters] 部分。



让我们稍微修改一下这个例子，以便可以使用 URL 传递一些参数信息给控制器类（例如 ， */HelloWorld/Welcome?name=Scott&numtimes=4* ）。改变您的 Welcome 方法来包含两个参数，如下所示。需要注意的是，示例代码使用了 C＃语言的可选参数功能，numTimes 参数在不传值时，默认值为 1。

```
public string Welcome(string name, int numTimes = 1) {
     return HttpUtility.HtmlEncode("Hello " + name + ", NumTimes is: " +
numTimes);
}
```

运行您的应用程序并浏览此 URL（*http://localhost:xxxx/HelloWorld/Welcome?name=Scott&numtimes=4* ）。你可以对参数 `name` 和 `numtimes` 尝试不同的值。 ASP.NET MVC model binding system 会自动将地址栏中 URL 里的 query string 映射到您方法中的参数。



在这两个例子中，控制器一直在做着 MVC 中“VC”部分的职能。 也就是视图和控制器的工作。该控制器直接返回 HTML 内容。通常情况下，您不会让控制器直接返回 HTML，因为这样代码会变得非常的繁琐。相反，我们通常会使用一个单独的视图模板文件来帮助生成返回的 HTML。让我们来看看下面我们如何能做到这一点吧。

添加一个视图

在本节中，您需要修改 HelloWorldController 类，从而使用视图模板文件，干净优雅的封装生成返回到客户端浏览器 HTML 的过程。

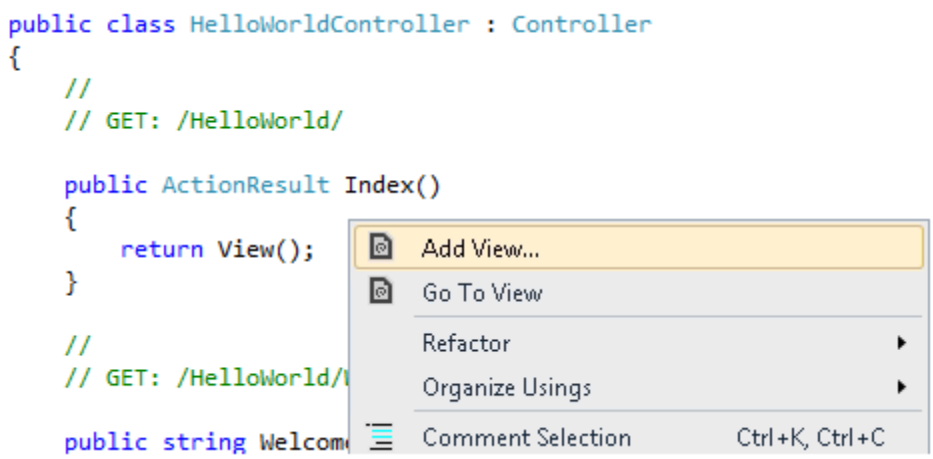
您将创建一个视图模板文件，其中使用了 ASP.NET MVC 3 所引入的 Razor 视图引擎。Razor 视图模板文件使用.cshtml 文件扩展名，并提供了一个优雅的方式来使用 C＃语言创建所要输出的 HTML。用 Razor 编写一个视图模板文件时，将所需的字符和键盘敲击数量降到了最低，并实现了快速，流畅的编码工作流程。

当前在控制器类中的 Index 方法返回了一个硬编码的字符串。更改 Index 方法返回一个 View 对象，如下面的示例代码：

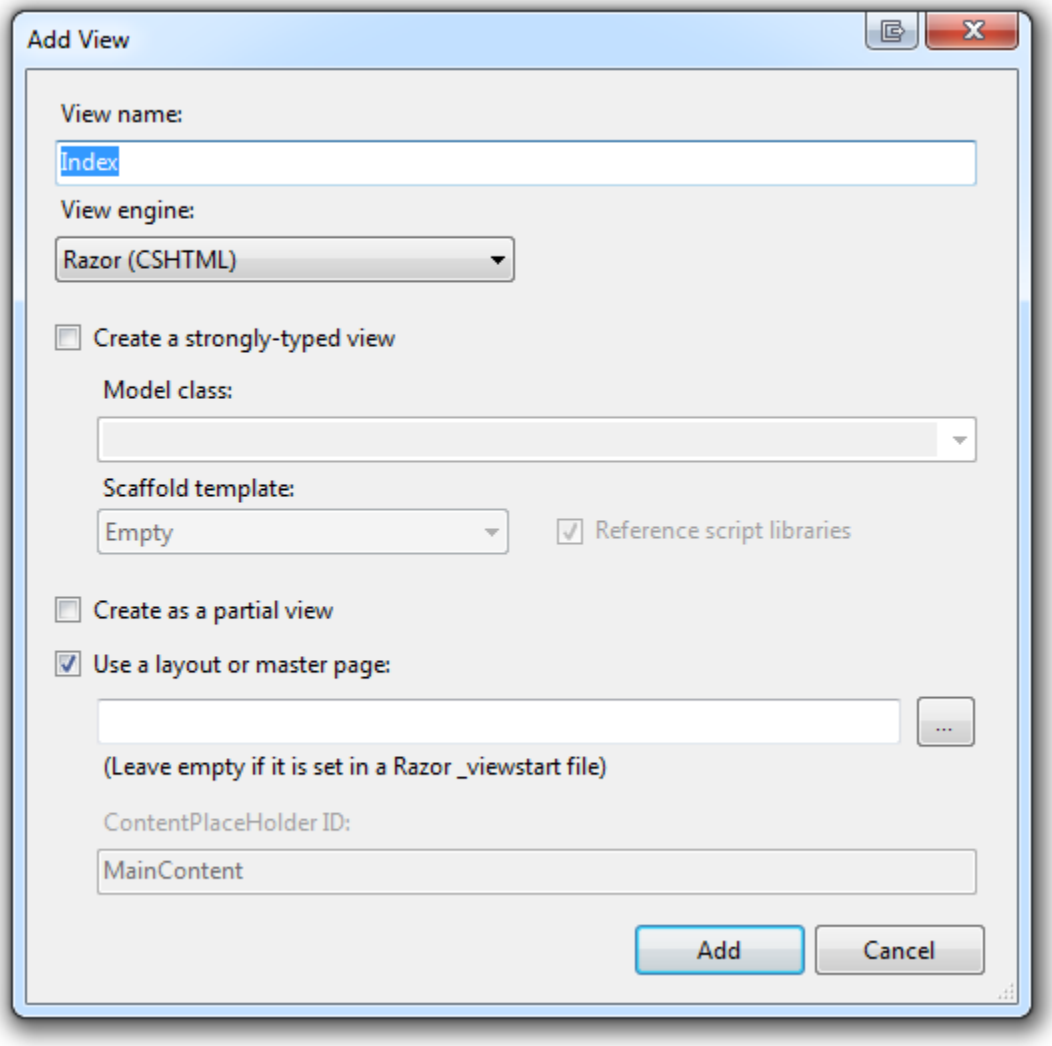
```
public ActionResult Index()
{
    return View();
}
```

上面的 Index 方法使用一个视图模板来生成一个 HTML 返回给浏览器。控制器的方法（也被称为 action method(操作方法) ），如上面的 Index 方法，一般返回一个 ActionResult（或从 ActionResult 所继承的类型），而不是原始的类型，如字符串。

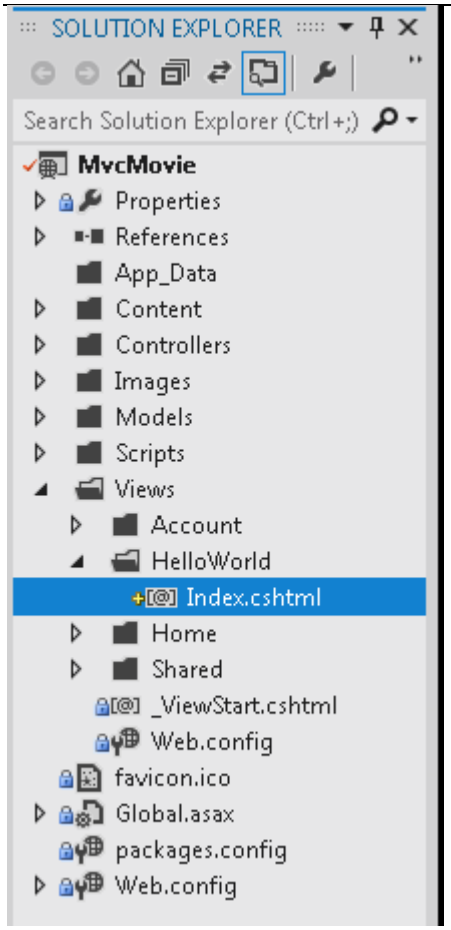
在该项目中，您可以使用的 Index 方法来添加一个视图模板。要做到这一点，在 Index 方法中单击鼠标右键，然后单击“**添加视图**“。



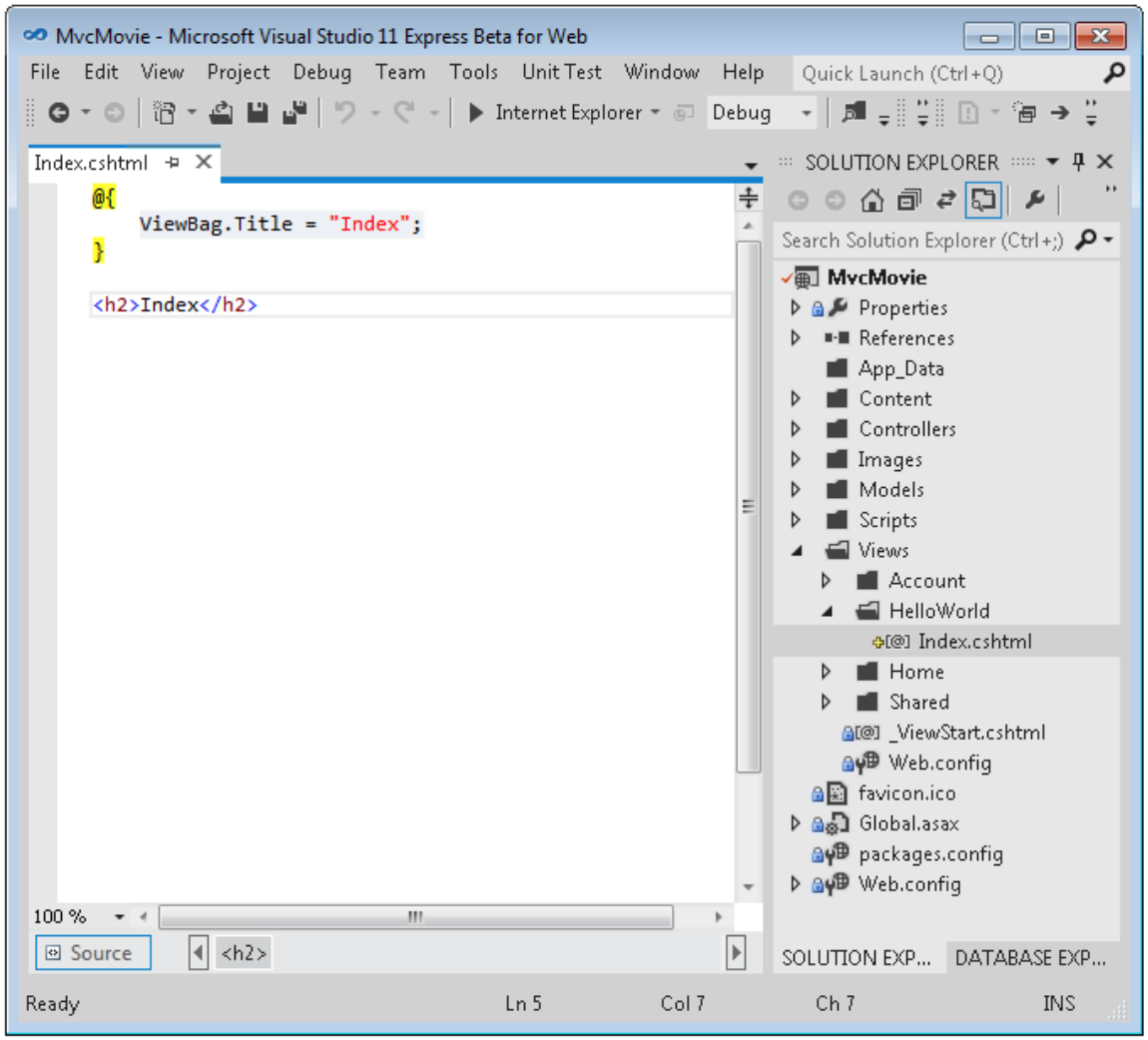
出现**添加视图**对话框。保留缺省值，并单击**添加**按钮：



您可以在**解决方案资源管理器**中看到 MvcMovie\HelloWorld 文件夹和已被创建的 MvcMovie\View\HelloWorld\Index.cshtml 文件：



下图显示了已被创建的 Index.cshtml 文件：



在<h2>标签后面添加以下 HTML。

```
<p>Hello from our View Template!</p>
```

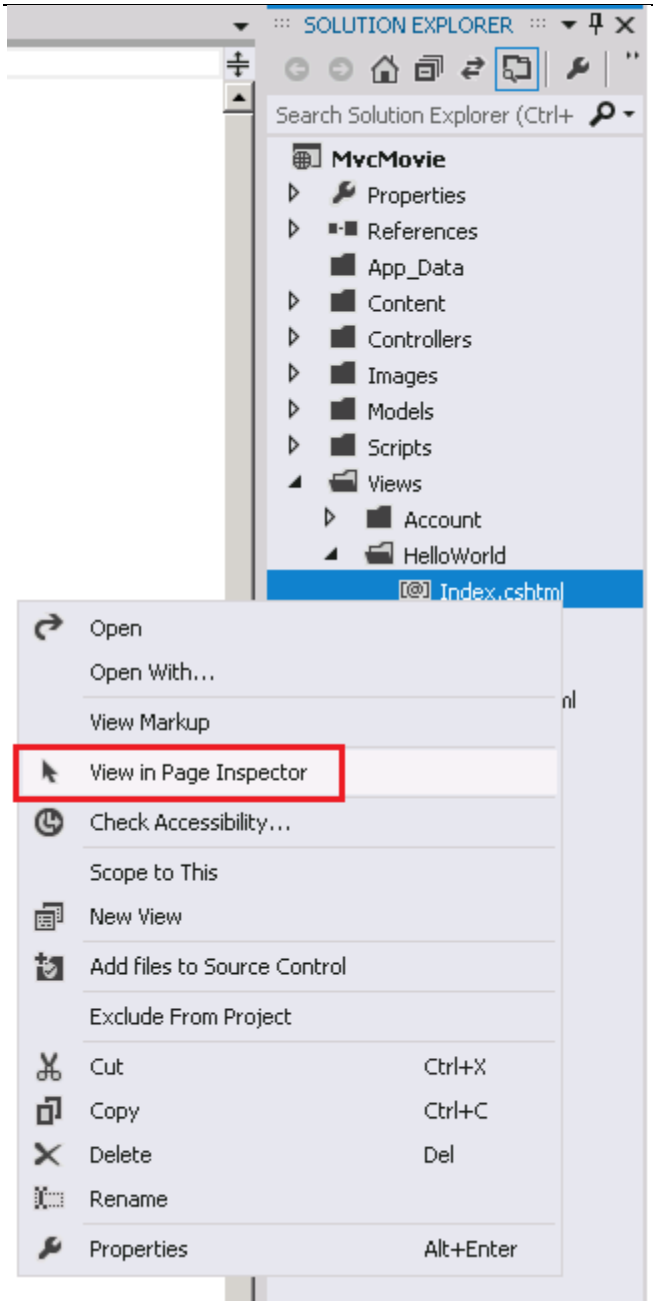
完整的 MvcMovie\HelloWorld\Index.cshtml 文件如下所示。

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

<h2>Index</h2>
```

```
<p>Hello from our View Template!</p>
```

注：如果您使用的是 Internet Explorer 9，您将看不到在上面用黄色高亮标记的`<p>Hello from our View Template!</p>`，单击"兼容性视图"按钮 ，在 IE 浏览器中，图标会从 变为纯色的 图标。另外，您也可以在 Firefox 或 Chrome 查看本教程。

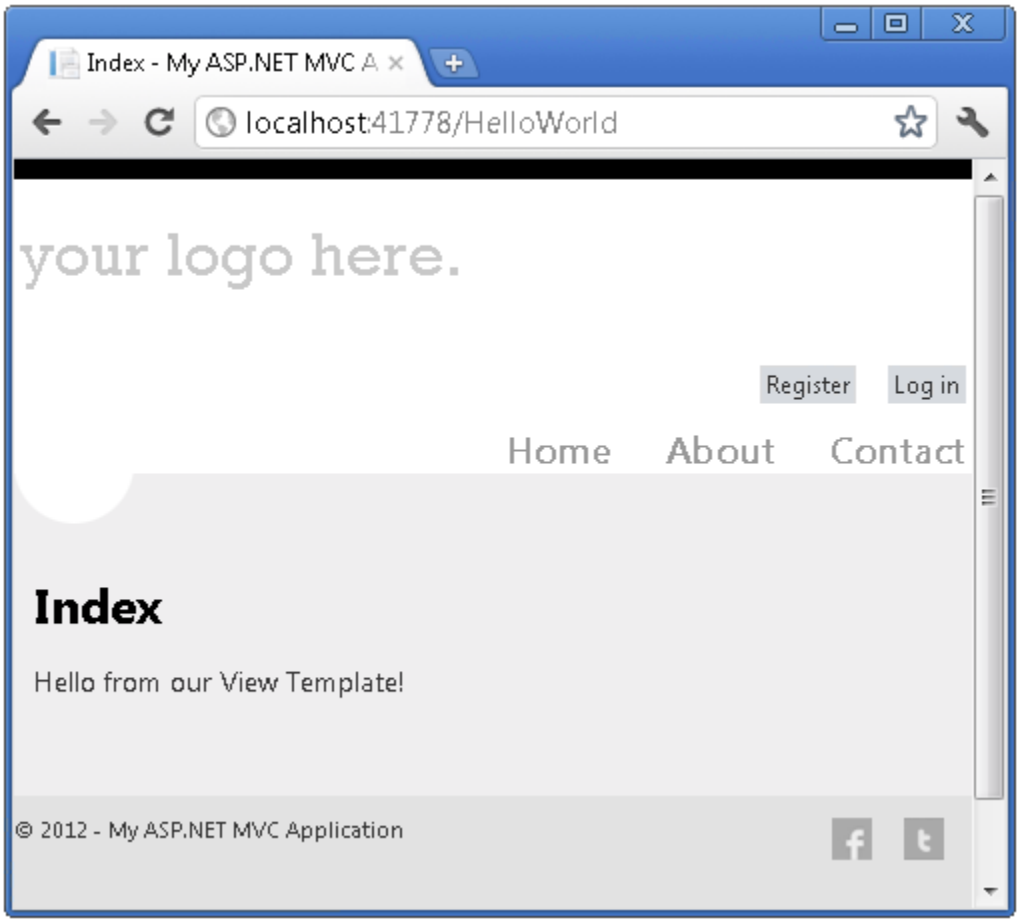
如果您正在使用 Visual Studio 2012，在解决方案资源管理器中，右键单击 *Index.cshtml* 文件，并选择 " **在页面检查器中查看** "。



页面检查器教程中会有更多的信息介绍这个工具。

同时，运行应用程序并在浏览器中浏览：`HelloWorld` 控制器（*http://localhost:xxxx/HelloWorld*“）。在您控制器的 `Index` 方法中并没有做太多的工作，它只是执行了 `return View()`，这个方法指定使用一个视图模板文件来 Render 返回给浏览器的 HTML。因为您没有明确指定使用那个视图模板文件，ASP.NET MVC 会默认使用

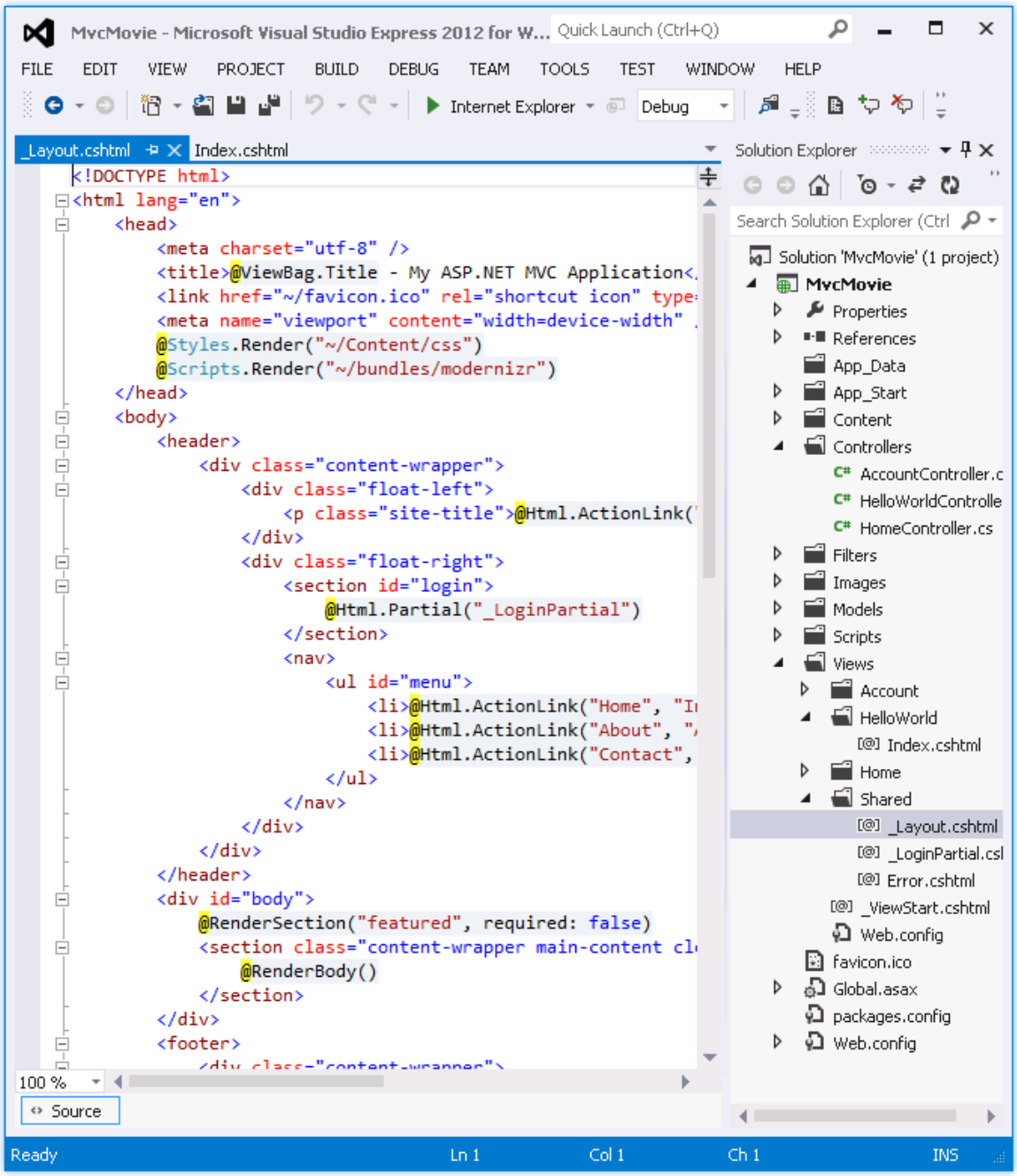
\Views\HelloWorld 文件夹下的 Index.cshtml 视图文件。下图显示了在视图文件中硬编码的字符串 "Hello from our View Template!"



看起来很不错吧。但是，请注意，浏览器的标题栏会显示为"Index My ASP.NET A" 并且在页面顶部的大链接会显示为 "your logo here."，右边的链接为注册(Register)和登录(Log in)，下面的链接为首页（Home），简介（About）和联系（Contact）。让我们来改变一些吧。

修改视图和布局页

首先，您想要修改在页面顶部的标题 "your logo here."。这段文字是每个页面的公用文字。即使这段文字出现在每个页面上，但是实际上它仅保存在工程里的一个地方。在**解决方案资源管理器**里找到*/Views/Shared*文件夹，打开*_Layout.cshtml*文件。此文件被称为*布局页面*（Layout page），并且其它所有的子页面，都共享使用这个布局页面。



布局模版允许您在一个位置放置占位所需的 HTML 容器，然后将其应用到您网站中所有的网页布局。 查找 `@RenderBody()`. 您所创建的所有视图页面都被"包装"在布局页面中来显示，RenderBody 只是个占位符。例如，如果您点击“关于(About)”链接，*Views\Home\About.cshtml* 视图会在 RenderBody 方法内进行 Render。

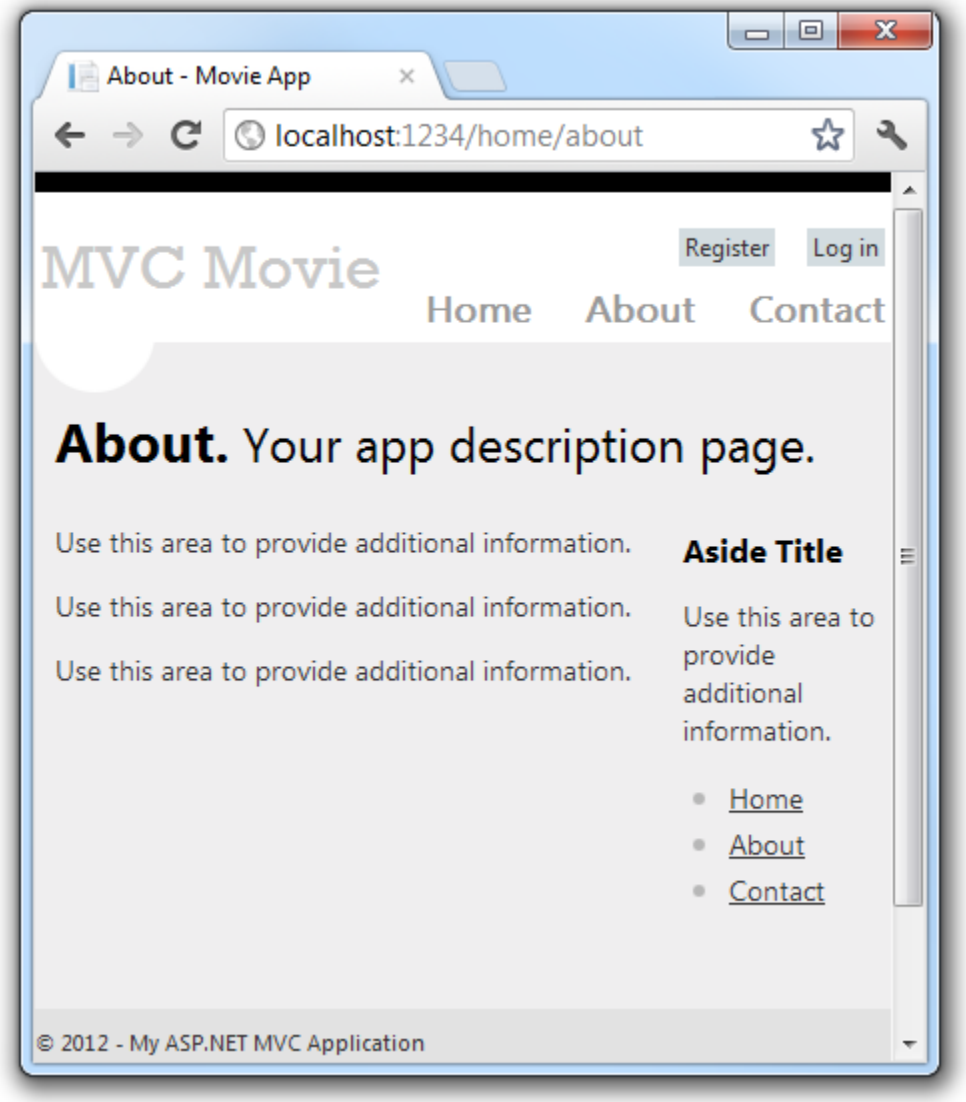
在布局模板页面内把网站标题从"your logo here" 修改为 "MVC Movie”。

```
<div class="float-left">
    <p class="site-title">@Html.ActionLink("MVC Movie", "Index", "Home")</p>
</div>
```

替换 title 元素的内容为以下内容：

```
<title>@ViewBag.Title - Movie App</title>
```

运行应用程序，您会看到 "MVC Movie "。单击“ **关于(About)** “链接，您可以看到该页面也会显示为"MVC Movie "。我们可以在布局模版里再修改一次，使得网站里所有网页的标题都同时被修改掉。



现在，让我们来修改 Index 视图的标题。

打开 MvcMovie\Views\HelloWorld\Index.cshtml 文件，有两个地方需要进行修改：第一，浏览器上的标题文字，其次，二级标题文字 (<h2>元素)。让它们稍有不同，这样就可以看出到底程序里那部分的代码被修改了。

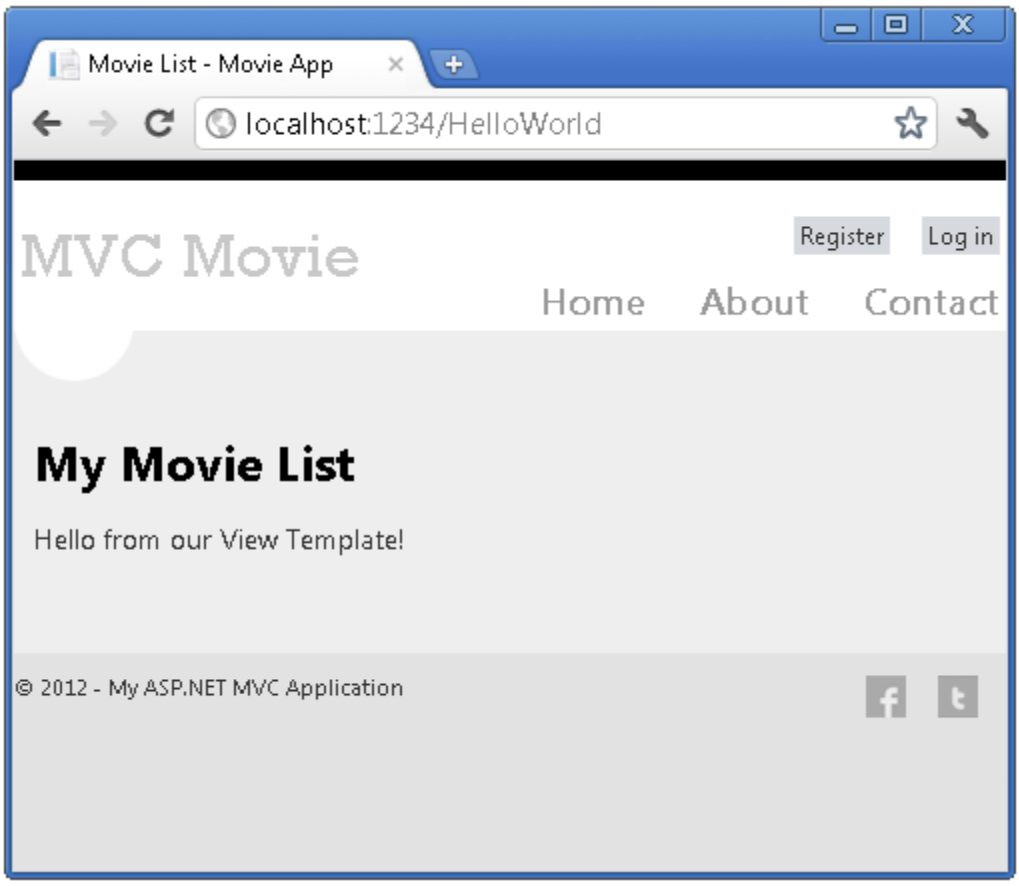
```
@{
    ViewBag.Title = "Movie List";
}

<h2>My Movie List</h2>
```

```
<p>Hello from our View Template!</p>
```

如果要指定 HTML 的 title 元素，上面的代码设置了 `ViewBag` 对象（在 *Index.cshtml* 视图模板中）的 `Title` 属性。如果您回去看看布局模板的源代码，您会发现该模板会输出此值倒`<title>`元素中，从而作为我们之前修改过的 HTML `<head>`里的一部分。使用此 `ViewBag` 方法，您可以轻松地从视图模板传递其它参数给布局模板页面。

运行应用程序，浏览 *http://localhost:xx/HelloWorld*。浏览器的标题、 主标题和二级标题都已经被修改了。（如果您在浏览器中看不到修改，有可能是页面被缓存了。按 Ctrl + F5 强制浏览器重新请求并加载服务器返回的 HTML。在 *Index.cshtml* 视图模版中设置的 `ViewBag.Title` 输出了浏览器的标题，附加的"- Movie App"是在布局模板文件中添加的。

此外还要注意 *Index.cshtml* 视图模板中的内容是如何合并到*_Layout.cshtml* 模板，从而形成一个完整的 HTML 返回到客户端浏览器的。使用布局模板页面，可以很容易进行一个修改并应用到所有页面。



我们这一点（在本例中的"Hello from our View Template!"字符串） 的"数据"只是一段硬编码。这个 MVC 应用程序有了一个"V"（视图），也有了一个"C"（控制器），但还没有"M"(模型)。不过稍后，我们将介绍如何创建一个数据库并检索数据模型。

将数据从控制器传递给视图

在我们讨论数据库和数据模型之前，让我们先讨论一下如何将数据从控制器传递给视图。控制器类将响应请求来的 URL。控制器类是给您写代码来处理传入请求的地方，并从数据库中检索数据，并最终决定什么类型的返回结果会发送回浏览器。视图模板可以被控制器用来产生格式化过的 HTML 从而返回给浏览器。

控制器负责给任何数据或者对象提供一个必需的视图模板，用这个视图模板来 Render 返回给浏览器的 HTML。最佳做法是：**一个视图模板应该永远不会执行业务逻辑或者直接和数据库**

进行交互。相应的，一个视图模板应该只和控制器所提供的数据进行交互。维持这种"隔离关系"可以帮助，保持代码的干净、测试性和更易维护。

当前， `HelloWorldController` 类中 `Welcome` 操作方法需要一个 `name` 和一个 `numTimes` 参数，然后直接输出给浏览器。相比只返回一个字符串，让我们来改变控制器，来使用视图模板吧。视图模板将生成动态的 HTML，这意味着您需要通过适当的方式把数据从控制器传递给视图，从而才能生成动态的 HTML。您可以把视图模板需要的动态数据（参数）在控制器中放入到一个 `ViewBag` 对象中，然后视图模板可以访问这个对象。

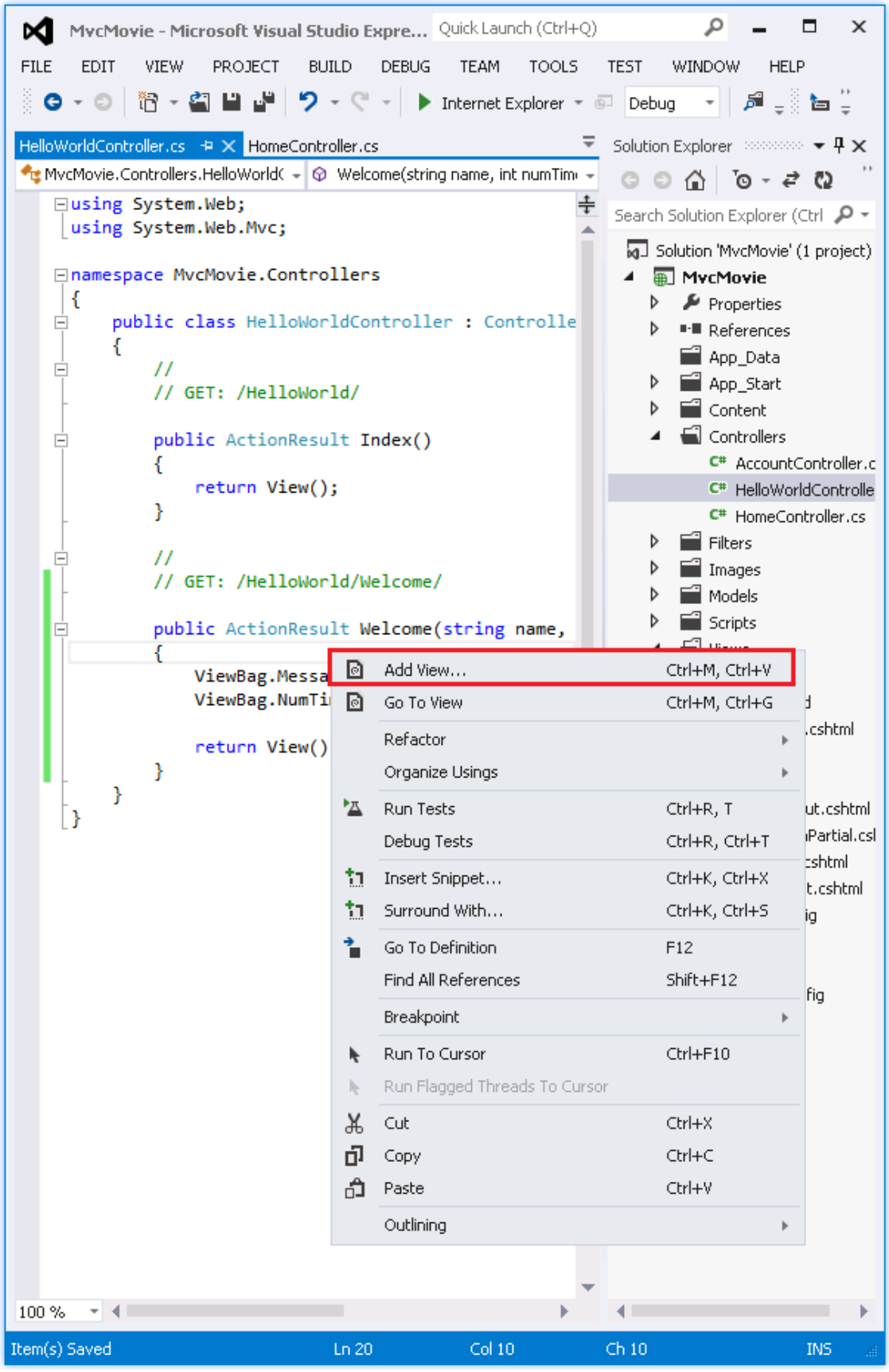
打开 *HelloWorldController.cs* 文件，更改 `Welcome` 方法，将 `Message` 和 `NumTimes` 的值添加到 `ViewBag` 对象里。`ViewBag` 是一个动态的对象，这意味着在您没有给 `ViewBag` 放置属性时，它没有任何属性，您可以把任何您想放置的对象放入到 `ViewBag` 对象中。 ASP.NET MVC model binding system 会自动将地址栏中 URL 里的 query string 映射到您方法中的参数（`name` 和 `numTimes`）。完整的 *HelloWorldController.cs* 文件如下所示：

```
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Welcome(string name, int numTimes = 1)
        {
            ViewBag.Message = "Hello " + name;
            ViewBag.NumTimes = numTimes;

            return View();
        }
    }
}
```

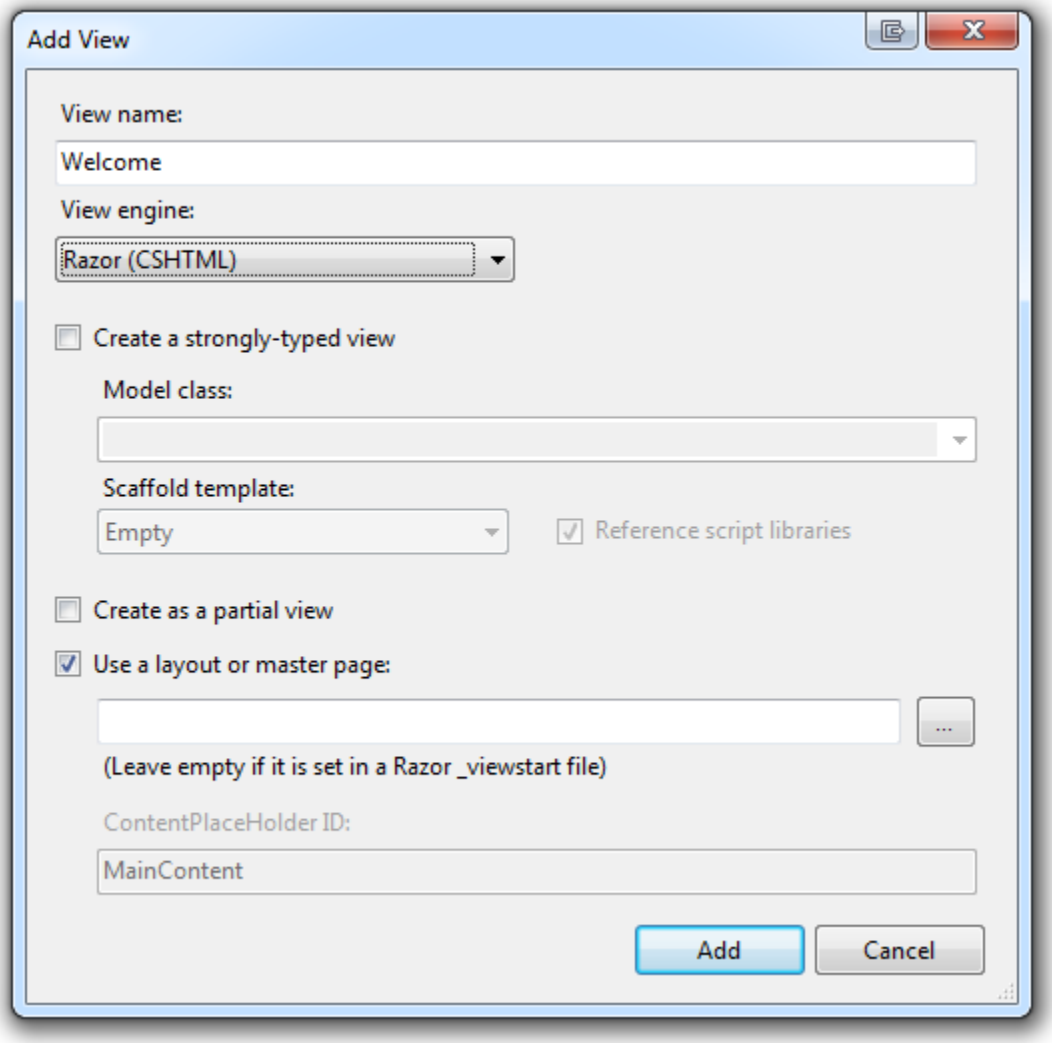
现在 `ViewBag` 对象包含了数据，并将自动传递给视图模板。

接下来，您需要一个欢迎视图模板 ！在**生成**菜单中，选择**生成 MvcMovie** ，以确保项目编译成功。

在 `Welcome` 方法内单击右键，然后单击**添加视图**.


MvcMovie - Microsoft Visual Studio Expre...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Internet Explorer
Debug
HelloWorldController.cs
HomeController.cs
MvcMovie.Controllers.HelloWorld(
Welcome(string name, int numTim
using System.Web;
using System.Web.Mvc;
namespace MvcMovie.Controllers
{
public class HelloWorldController : Controlle
{
//
// GET: /HelloWorld/
public ActionResult Index()
{
return View();
}
//
// GET: /HelloWorld/Welcome/
public ActionResult Welcome(string name,
{
ViewBag.Messa
ViewBag.NumTi
return View()
}
}
}
Solution Explorer
Search Solution Explorer (Ctrl
Solution 'MvcMovie' (1 project)
MvcMovie
Properties
References
App_Data
App_Start
Content
Controllers
AccountController.c
HelloWorldControlle
HomeController.cs
Filters
Images
Models
Scripts
Add View... Ctrl+M, Ctrl+V
Go To View Ctrl+M, Ctrl+G
Refactor
Organize Usings
Run Tests Ctrl+R, T
Debug Tests Ctrl+R, Ctrl+T
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Go To Definition F12
Find All References Shift+F12
Breakpoint
Run To Cursor Ctrl+F10
Run Flagged Threads To Cursor
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Outlining
100 %
Item(s) Saved
Ln 20
Col 10
Ch 10
INS

下面是**添加视图**对话框：



单击**添加**，然后在新加的 *Welcome.cshtml* 文件中，<h2>元素后面，添加以下代码 。您将创建一个循环，NumTimes 将决定输出多少个"Hello"。下面显示了完整的 *Welcome.cshtml* 文件。

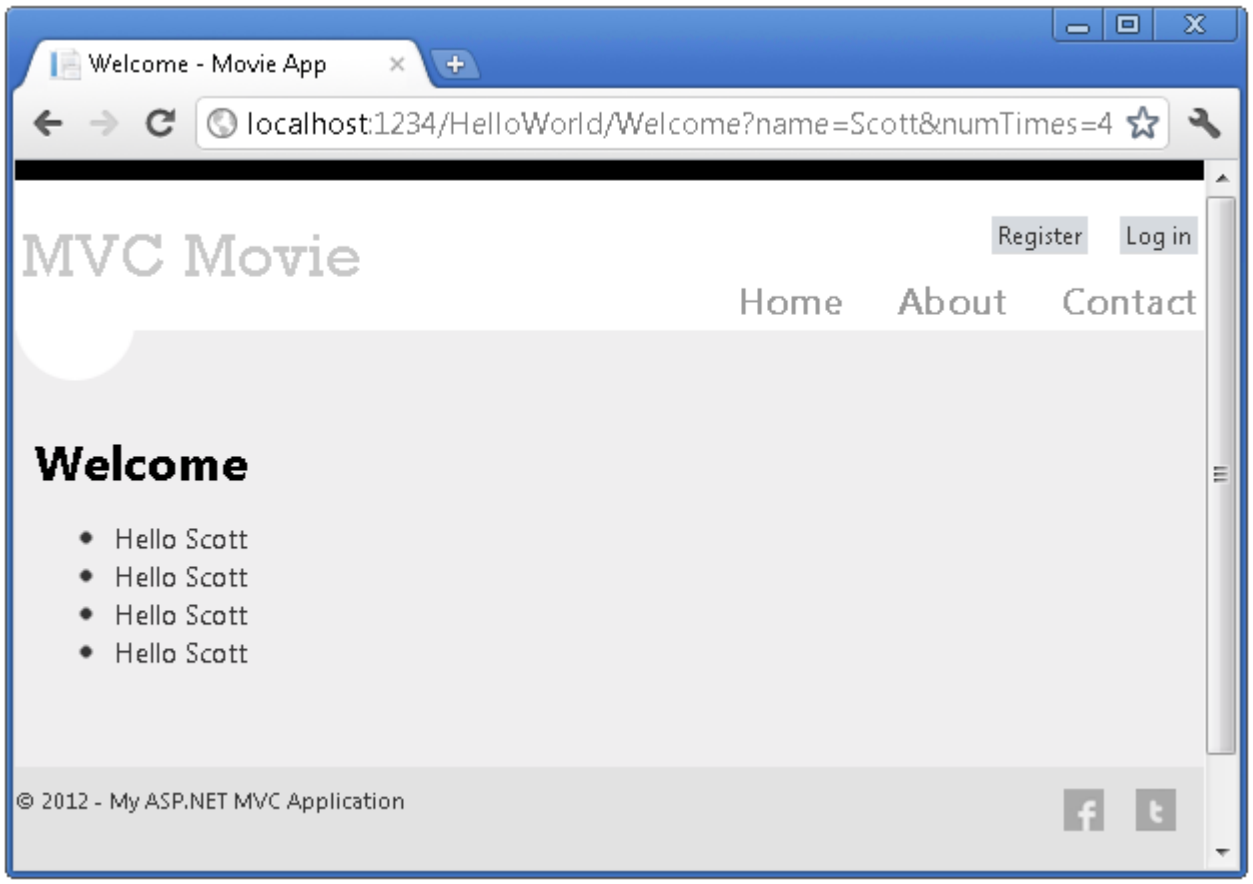
```
@{
    ViewBag.Title = "Welcome";
}

<h2>Welcome</h2>

<ul>
```

```
    @for (int i=0; i < ViewBag.NumTimes; i++) {
        <li>@ViewBag.Message</li>
    }
</ul>
```

运行应用程序，并浏览下面的 URL：

http://localhost:xx/HelloWorld/Welcome?name=Scott&numtimes=4

现在，模型绑定使得数据从 URL 传递给控制器。控制器将数据装入到 `ViewBag` 对象中，通过该对象传递给视图。然后视图为用户生成显示所需的 HTML。



在上面的示例中，我们使用了 `ViewBag` 对象把数据从控制器传递给了视图。在本系列教程后面的文章中，我们将使用视图模型来将数据从一个控制器传递到视图中。用视图模型来传递数据，这一般是首选的办法。Blog Dynamic V Strongly Typed Views 有更加详细的介绍。

到这里，这是一种"M"模型，但不是数据库的那种“M”模型。让我们来创建一个电影数据库吧。

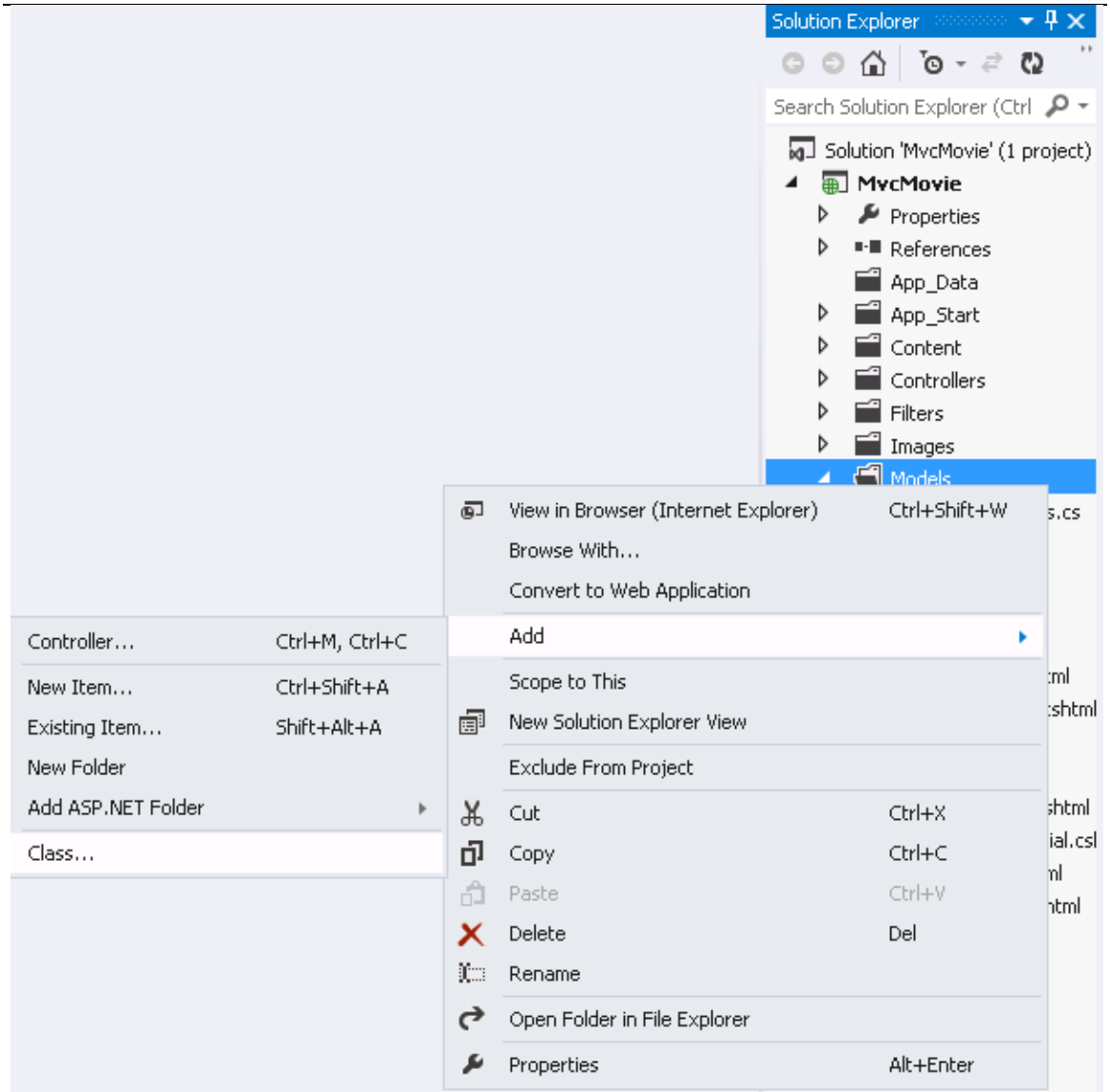
添加一个模型

在本节中，您将添加一些类，这些类用于管理数据库中的电影。这些类是 ASP.NET MVC 应用程序中的"模型(Model)"。

您将使用.NET Framework 数据访问技术 Entity Framework，来定义和使用这些模型类。Entity Framework（通常称为 EF) 是支持代码优先的开发模式。代码优先允许您通过编写简单的类来创建对象模型。（相对于"原始的 CLR objects"，这也被称为 POCO 类）然后可以从您的类创建数据库，这是一个非常干净快速的开发工作流程。

添加模型类

在**解决方案资源管理器中**，右键单击***模型***文件夹，选择**添加**，然后选择**类**.



输入 *Class* 名 "Movie"。

将下列五个属性添加到 `Movie` 类：

```
public class Movie
{
    public int ID { get; set; }
    public string Title { get; set; }
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }
    public decimal Price { get; set; }
}
```

我们将使用 Movie 类来表示数据库中的电影。Movie 对象的每个实例将对应数据库表的一行， Movie 类的每个属性将对应表的一列。

在同一文件中，添加下面的 MovieDBContext 类：

```
 public class MovieDBContext : DbContext
{
    public DbSet<Movie> Movies { get; set; }
}
```

MovieDBContext 类代表 Entity Framework 的电影数据库类，这个类负责在数据库中获取，存储，更新，处理 Movie 类的实例。MovieDBContext 继承自 Entity Framework 的 DbContext 基类。

为了能够引用 DbContext 和 DbSet，您需要在文件的顶部添加以下 using 语句：

```
using System.Data.Entity;
```

下面显示了完整的 *Movie.cs* 文件。（一些不用的 using 语句已经被删除了）

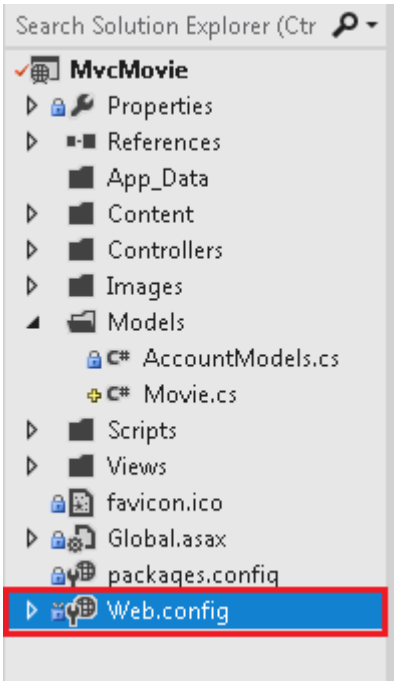
```
using System;
using System.Data.Entity;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }

    public class MovieDBContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
```

```
    }
}
```

创建连接字符串并使用 SQL Server LocalDB

您刚创建的 `MovieDBContext` 类用来连接数据库，并将 `Movie` 对象映射到数据库表记录。你可能会问一个问题，如何指定它将连接到那个数据库。通过在应用程序的 *Web.config* 文件中添加数据库连接信息来指定连接到那个数据库。

打开应用程序根目录的 *Web.config* 文件。（不是 *View* 文件夹下的 *Web.config* 文件。）打开红色高亮标记的 *Web.config* 文件。



在 *Web.config* 文件中的`<connectionStrings>`内添加下面的连接字符串。

```
<add name="MovieDBContext"
   connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\Movies.mdf;Integrated
Security=True"
   providerName="System.Data.SqlClient"
/>
```

下面的例子里显示了部分 Web.config 文件中所新添加的连接字符串：

```
<connectionStrings>
  <add name="DefaultConnection"
       connectionString="Data Source=(LocalDb)\v11.0;Initial Catalog=aspnet-
MvcMovie-2012213181139;Integrated Security=true"
       providerName="System.Data.SqlClient"
  />
  <add name="MovieDBContext"
       connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\Movies.mdf;Integrated
Security=True"
       providerName="System.Data.SqlClient"
  />
</connectionStrings>
```

为了表示和存储电影数据到数据库中，上面少量的代码和 XML 是你所需要的一切。

接下来，您将创建一个新的 MoviesController 类，您可以用它来展示电影数据，并允许用户创建新的影片列表。

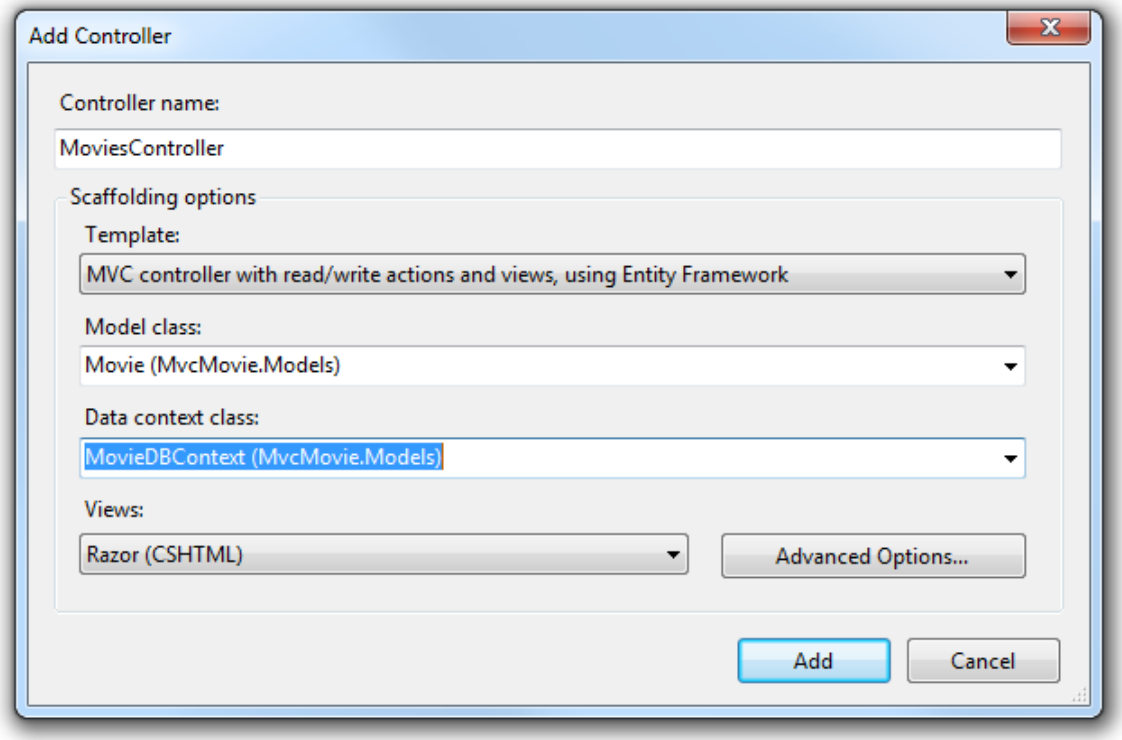
从控制器访问数据模型

在本节中，您将创建一个新的 `MoviesController` 类，并在这个 Controller 类里编写代码来取得电影数据，并使用视图模板将数据展示在浏览器里。

在开始下一步前，先 Build 一下应用程序(**生成应用程序**)(确保应用程序编译没有问题)

用鼠标右键单击 Controller 文件夹，并创建一个新的 `MoviesController` 控制器。当 Build 成功后，会出现下面的选项。设定以下选项：

- 控制器名称： **MoviesController**.（这是默认值）。
- 模板： **MVC Controller with read/write actions and views, using Entity Framework**.
- 模型类：**Movie (MvcMovie.Models)**.
- 数据上下文类： **MovieDBContext (MvcMovie.Models)**.
- 意见：**Razor (CSHTML).**（默认值）。

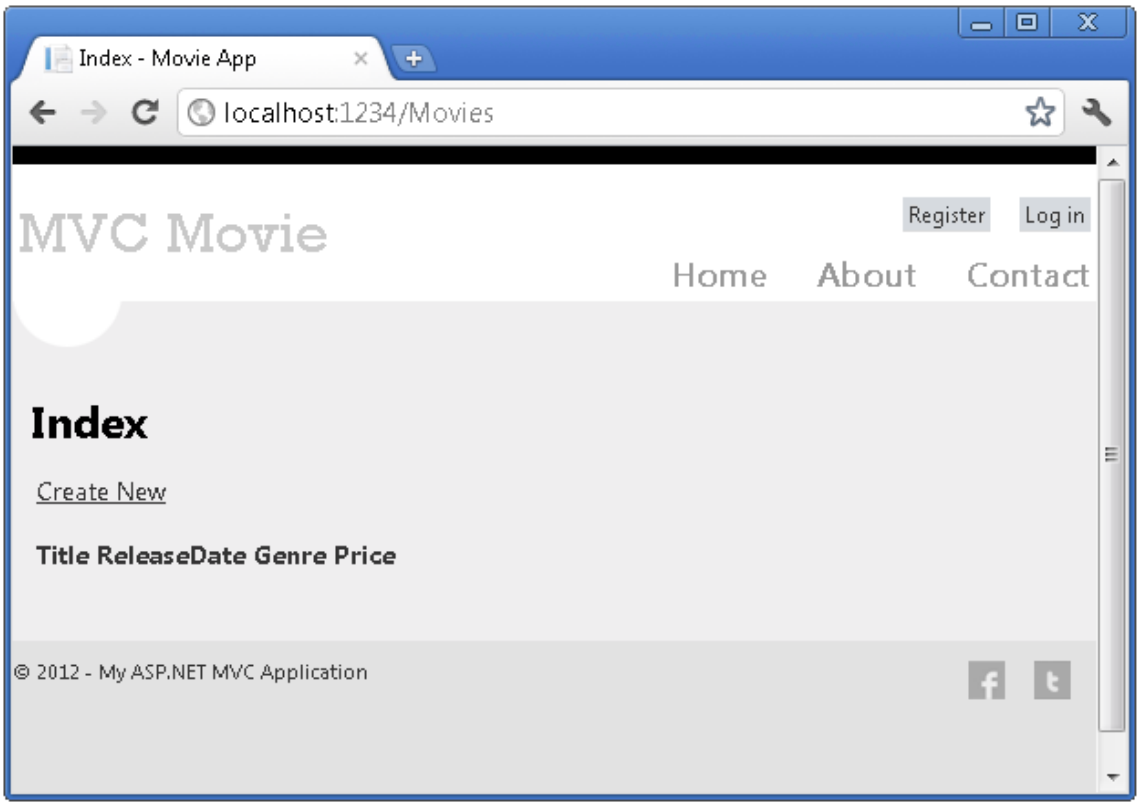


单击**添加**。Visual Studio Express 会创建以下文件和文件夹：

- 项目控制器文件夹中的 MoviesController.cs 文件。
- 项目视图文件夹下的 Movie 文件夹。
- 在新的 Views\Movies 文件夹中创建 Create.cshtml、 Delete.cshtml、 Details.cshtml、 Edit.cshtml 和 Index.cshtml 文件。

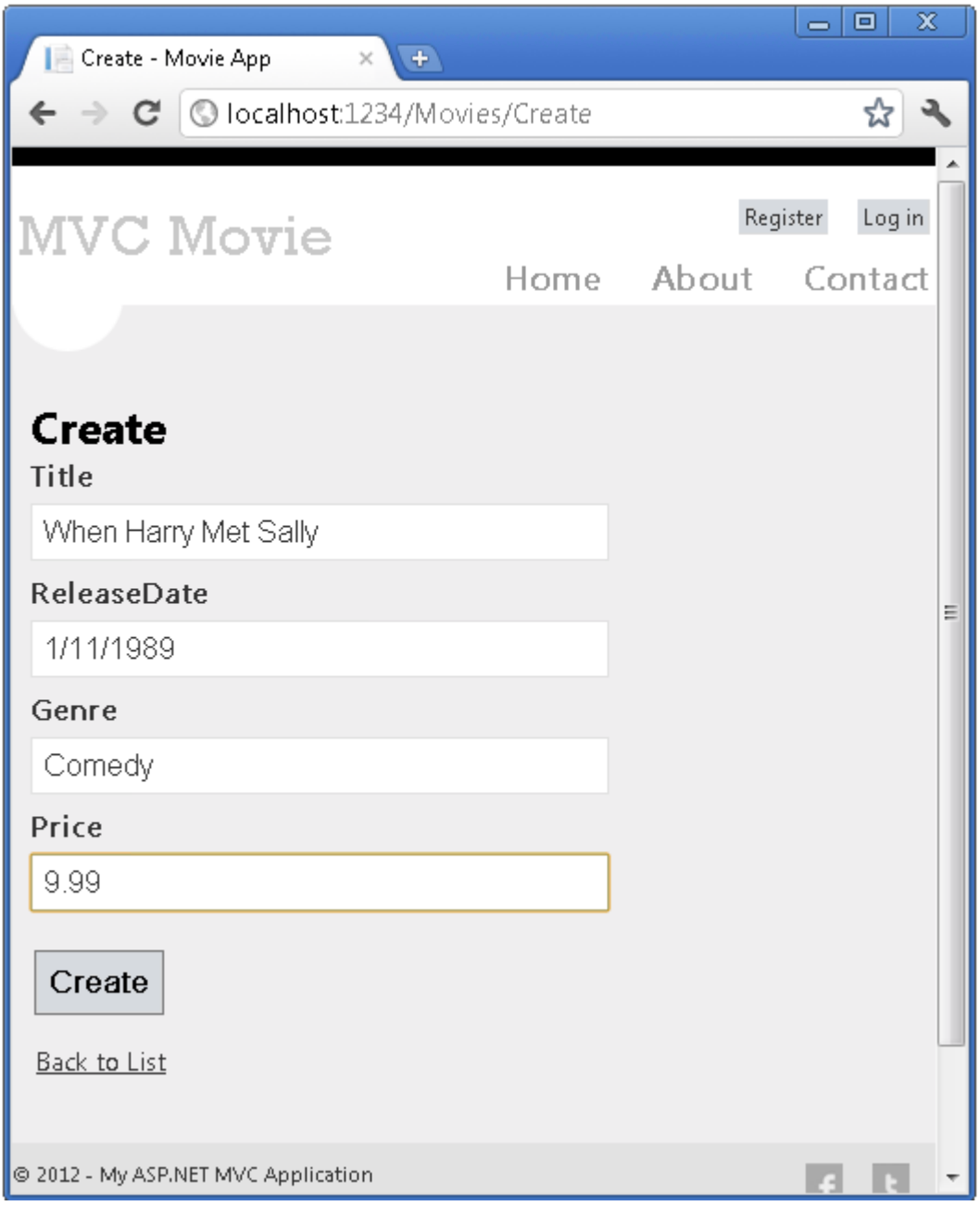
ASP.NET MVC 4 自动创建 CRUD （创建、 读取、 更新和删除）操作方法，和相关的视图文件(CRUD 自动创建的操作方法和视图文件被称为基础结构文件)。现在您有了可以创建，列表、 编辑和删电影 Entity 所有的 Web 功能了。

运行应用程序，通过将/Movies 追加到浏览器地址栏 URL 的后面，从而浏览 Movies 控制器。因为应用程序依赖于默认路由 （ *Global.asax* 文件中的定义），浏览器请求 *http://localhost:xxxxx/Movies* 将被路由到 `Movies` 控制器默认的 `Index` 操作方法。换句话说，浏览器请求 *http://localhost:xxxxx/Movies* 等同于浏览器请求 *http://localhost:xxxxx/Movies/Index*。因为您还没有添加任何内容，所以结果是一个空的电影列表。

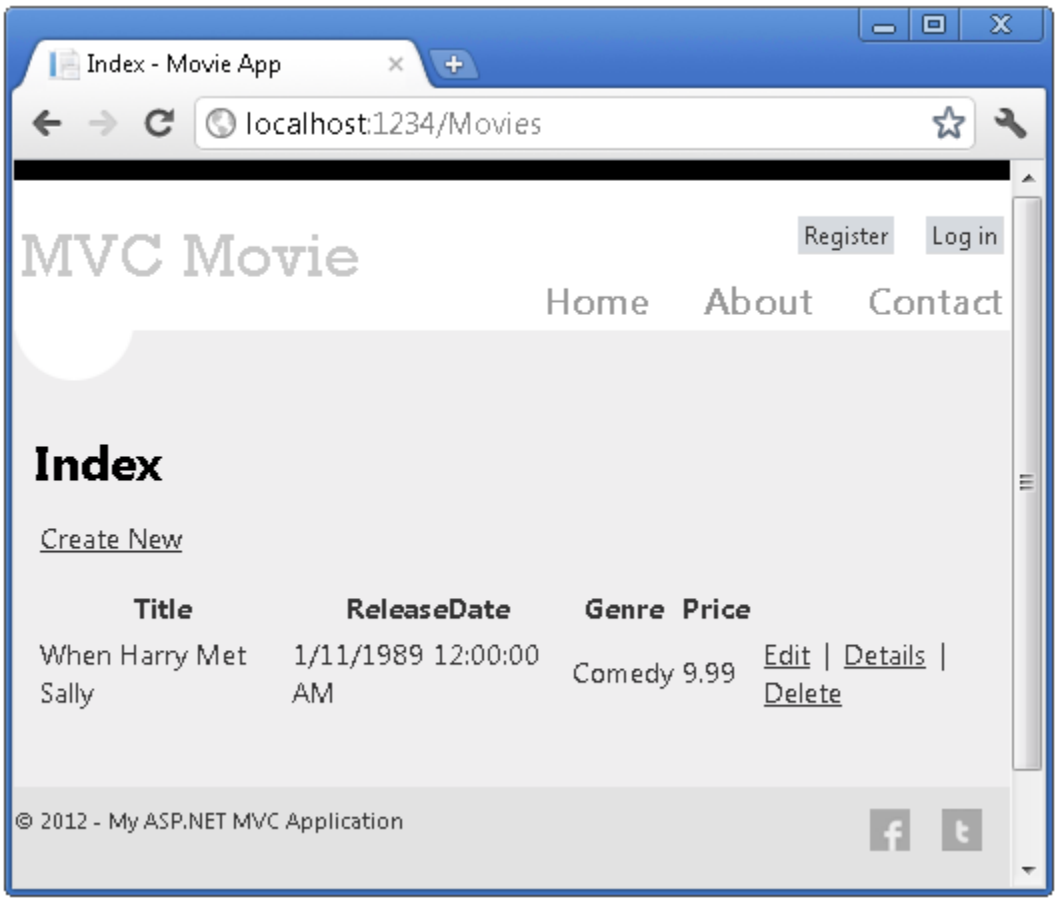


创建电影

点击 **Create New** 链接。输入有关电影的一些详细信息，然后单击 **Create** 按钮。



单击 **Create** 按钮将使得窗体提交至服务器，同时电影信息也会保存到数据库里，然后您会被重定向到 URL/Movies，您可以在列表中看到您刚刚创建的新电影。



创建一些更多的电影数据。同时也可以尝试点击**编辑**、**详细信息**和**删除**功能的链接。

看一下生成的代码

打开 *Controllers\MoviesController.cs* 文件，并找到生成的 `Index` 方法。一本部分电影控制器和 `Index` 方法如下所示。

```
public class MoviesController : Controller
{
    private MovieDBContext db = new MovieDBContext();

    //
    // GET: /Movies/

    public ActionResult Index()
    {
        return View(db.Movies.ToList());
    }
```

下面是 MoviesController 类中实例化电影数据库上下文实例，如前面所述。电影数据库上下文实例可用于查询、 编辑和删除的电影。

```
private MovieDBContext db = new MovieDBContext();
```

向 Movies 控制器请求，从而返回 Movies 电影数据库表中的所有记录，然后将结果传递给 Index 视图。

强类型模型和 @model 关键字

在本系列之前的教程中，您看到了使用 ViewBag 对象，从控制器传递数据或对象给视图模板。ViewBag 是一个动态的对象，提供了方便的后期绑定方法将信息传递给视图。

ASP.NET MVC 还提供了传递强类型数据或对象到视图模板的能力。这种强类型使得更好的在编译时检查您的代码并在 Visual Studio 编辑器中提供更加丰富的智能感知。当创建操作方法和视图时， Visual Studio 中的基础结构机制使用了 MoviesController 类和视图模板。

在 *Controllers\MoviesController.cs* 文件中看一下生成的 Details 方法。电影控制器里的 Details 方法如下所示。

```
public ActionResult Details(int id = 0)
{
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
```

如果查找到了一个 Movie，Movie 模型的实例会传递给 Detail 视图。看一下 *Views\Movies\Details.cshtml* 文件里的内容。

通过引入视图模板文件顶部的@model 语句，您可以指定该视图期望的对象类型。当您创建电影控制器时，Visual Studio 会将@model 声明自动包含到 *Details.cshtml* 文件的顶部：

```
@model MvcMovie.Models.Movie
```

此@model 声明使得控制器可以将强类型的 Model 对象传递给 View 视图，从而您可以在视图里访问传递过来的强类型电影 Model。例如，在 *Details.cshtml* 模板中，DisplayNameFor 和 DisplayFor HTML Helper 通过强类型的 Model 对象传递了电影的每个字段。创建和编辑方法还有视图模板都在传递电影的强类型模型对象。

看一下 *Index.cshtml* 视图模版和 *MoviesController.cs* 中的 Index 方法。请注意这些代码是如何在 Index 操作方法中，创建 List 对象，并调用 View 方法的。

此代码在控制器中传递 Movies 列表给视图：

```
public ActionResult Index()
{
    return View(db.Movies.ToList());
}
```

当您创建电影控制器时，Visual Studio Express 会自动包含 @model 语句到 *Index.cshtml* 文件的顶部：

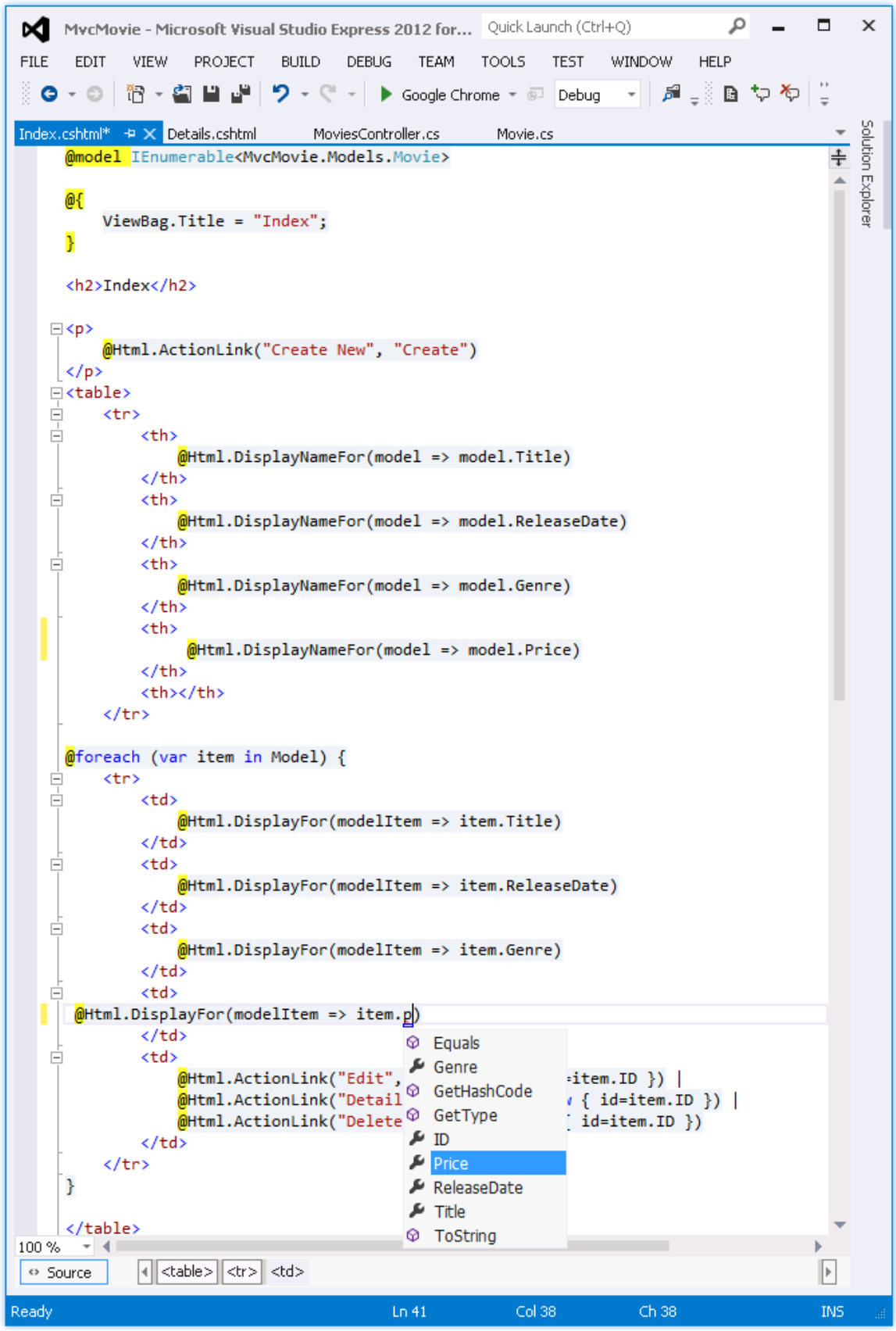
```
@model IEnumerable<MvcMovie.Models.Movie>
```

此@model 声明使得控制器可以将强类型的电影列表 Model 对象传递给 View 视图。例如，在 *Index.cshtml* 模板中，在强类型的 Model 对象上使用 foreach 语句循环遍历电影列表：

```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
```

```
        <td>
            @Html.DisplayFor(modelItem => item.Genre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
         <th>
            @Html.DisplayFor(modelItem => item.Rating)
        </th>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", { id=item.ID })  |
            @Html.ActionLink("Delete", "Delete", { id=item.ID })
        </td>
    </tr>
}
```

因为 `Model` 对象是强类型的（是 `IEnumerable<Movie>`对象），所以在循环中的每个 `item` 对象的类型是 `Movie` 类型。好处之一是，这意味着您可以在代码编译时进行检查，同时在代码编辑器中支持更加全面的智能感知：

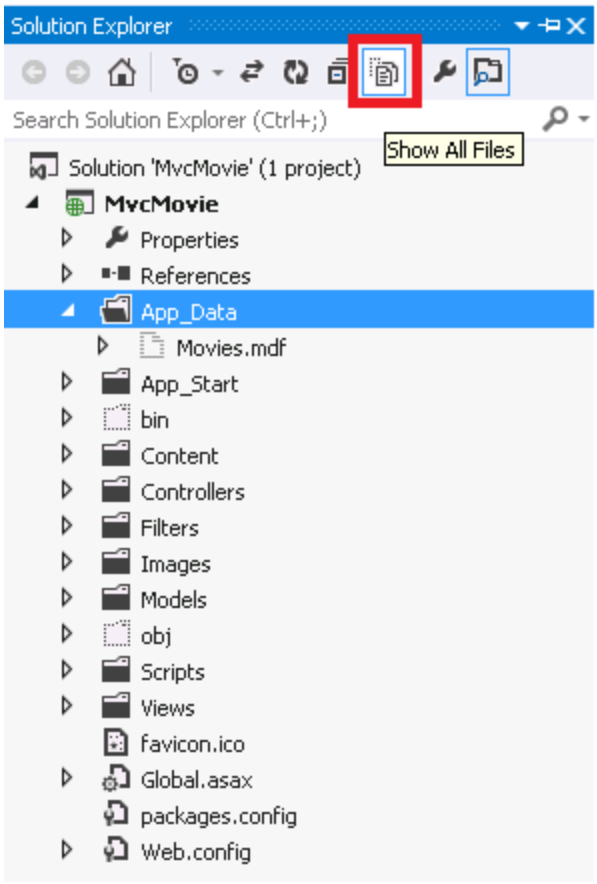

MvcMovie - Microsoft Visual Studio Express 2012 for...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Google Chrome
Debug
Index.cshtml*
Details.cshtml
MoviesController.cs
Movie.cs
Solution Explorer
@model IEnumerable<MvcMovie.Models.Movie>
@{
ViewBag.Title = "Index";
}
<h2>Index</h2>

@Html.ActionLink("Create New", "Create")

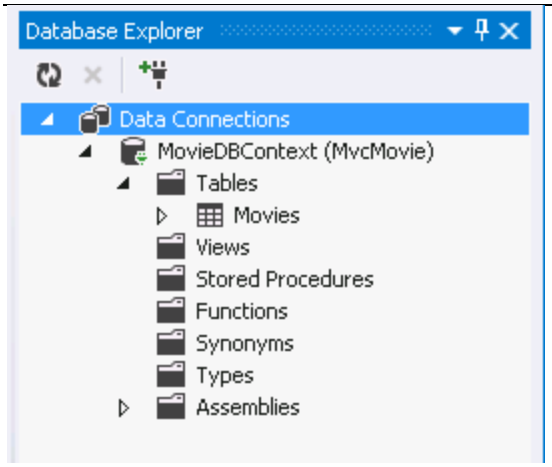
<table>
<tr>
<th>
@Html.DisplayNameFor(model => model.Title)
</th>
<th>
@Html.DisplayNameFor(model => model.ReleaseDate)
</th>
<th>
@Html.DisplayNameFor(model => model.Genre)
</th>
<th>
@Html.DisplayNameFor(model => model.Price)
</th>
<th></th>
</tr>
@foreach (var item in Model) {
<tr>
<td>
@Html.DisplayFor(modelItem => item.Title)
</td>
<td>
@Html.DisplayFor(modelItem => item.ReleaseDate)
</td>
<td>
@Html.DisplayFor(modelItem => item.Genre)
</td>
<td>
@Html.DisplayFor(modelItem => item.p)
</td>
<td>
@Html.ActionLink("Edit",
@Html.ActionLink("Detail
@Html.ActionLink("Delete
</td>
</tr>
}
</table>
Equals
Genre
GetHashCode
GetType
ID
Price
ReleaseDate
Title
ToString
100 %
Source
<table> <tr> <td>
Ready
Ln 41
Col 38
Ch 38
INS

使用 SQL Server LocalDB

Entity Framework Code First 代码优先，如果检测到不存在一个数据库连接字符串指向了 `Movies` 数据库，会自动的创建数据库。在 App_Data 文件夹中找一下，您可以验证它已经被创建了。如果您看不到 *Movies.mdf* 文件，请在**解决方案资源管理器**工具栏上，单击**显示所有文件**按钮，单击**刷新**按钮，然后展开 App_Data 文件夹。



双击 *Movies.mdf* 打开**数据库资源管理器**，然后展开**表**文件夹以查看电影表。



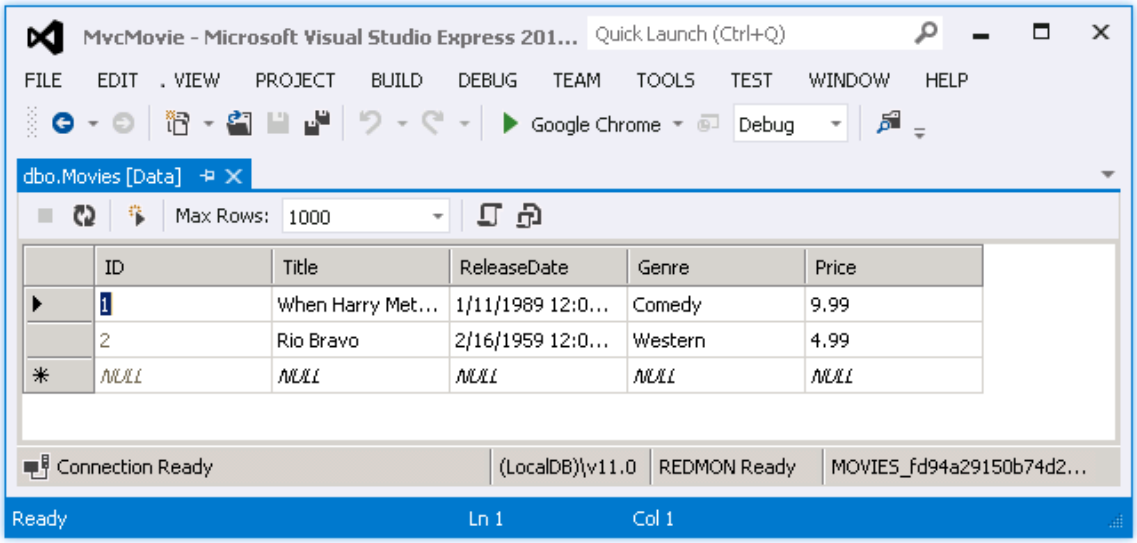
注： 如果没有显示数据库资源管理器，可以从**工具**菜单中，选择**连接到数据库**，然后关闭**选择数据源**对话框。这样将强制打开数据库资源管理器。

注： 如果您使用的 VWD 或 Visual Studio 2010 可能会看到类似下面的错误信息：

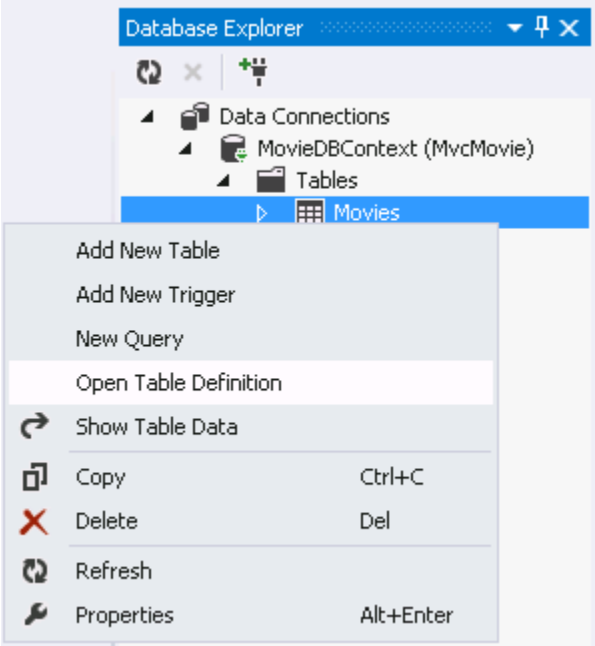
- 因为数据库 ' C:\Webs\MVC4\MVCMOVIE\MVCMOVIE\APP_DATA\MOVIES.MDF ' 是 706 版本的，所以无法打开。本服务器支持 655 和更早版本的数据库。无法降级支持。
- "InvalidOperation Exception was unhandled by user code" 所提供的 SqlConnection 没有指定初始数据库。

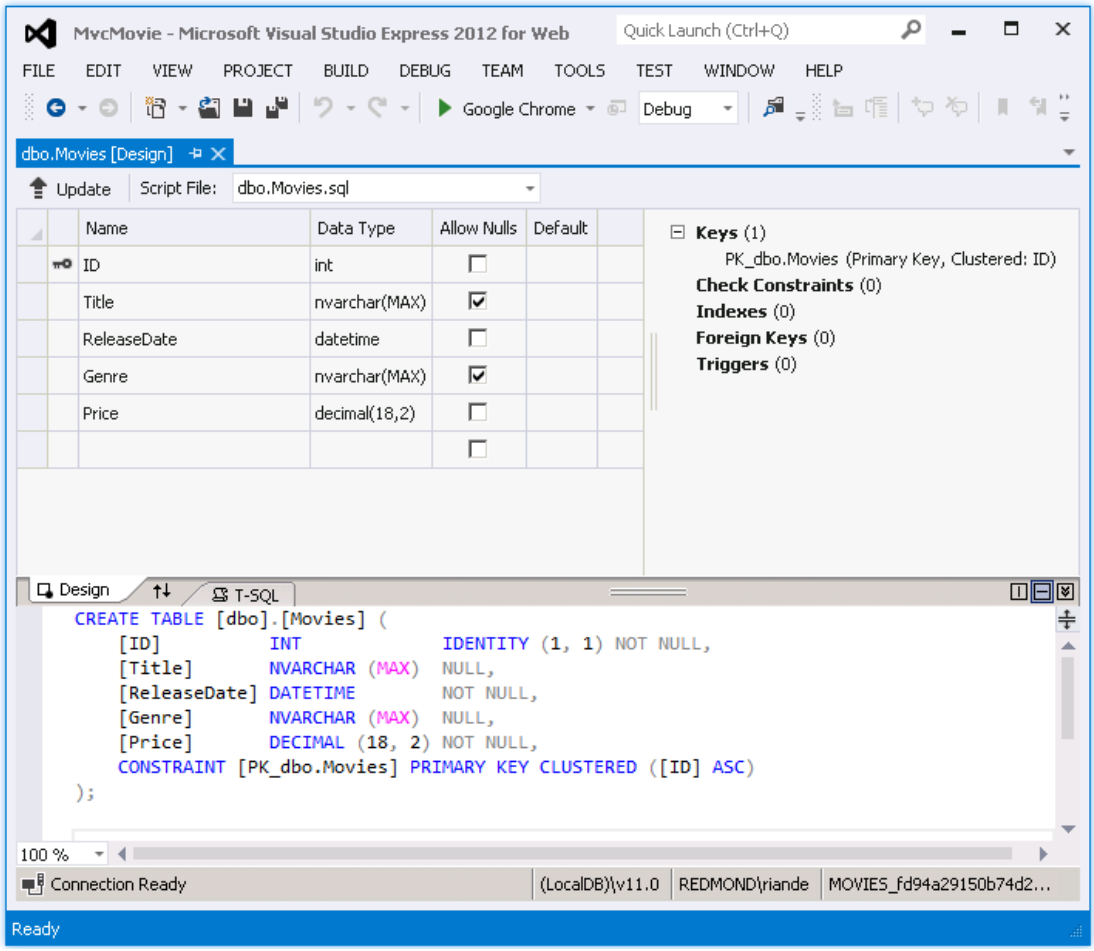
您需要安装 SQL Server 数据工具和 LocalDB。并验证在前面所指定的 `MovieDBContext` 连接字符串。

右键单击 `Movies` 表并选择**显示表数据**以查看您所创建的数据。



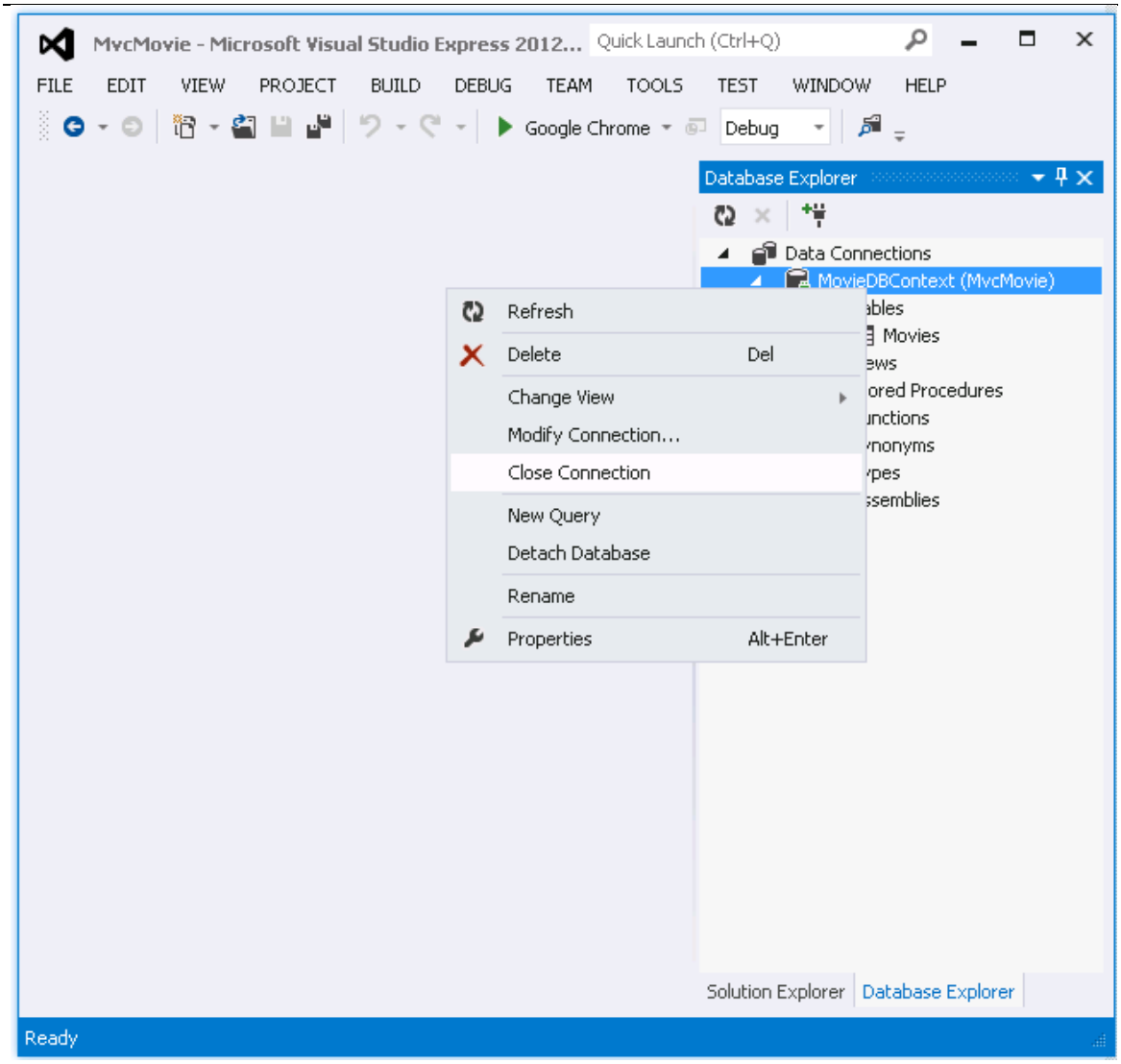
右键单击 `Movies` 表，选择**打开表定义**查看 Entity Framework 代码优先所创建表的表结构。





请注意，如何将 `Movies` 表的表结构映射到您早些时候所创建的 `Movie` 类？Entity Framework 代码优先为您自动创建了基于 `Movie` 类的表结构。

当您完成操作后，通过右键单击 *MovieDBContext* ，选择**关闭连接**关闭该数据库连接。（如果您没有关闭连接，当您下次运行该项目时，可能会出现错误）。

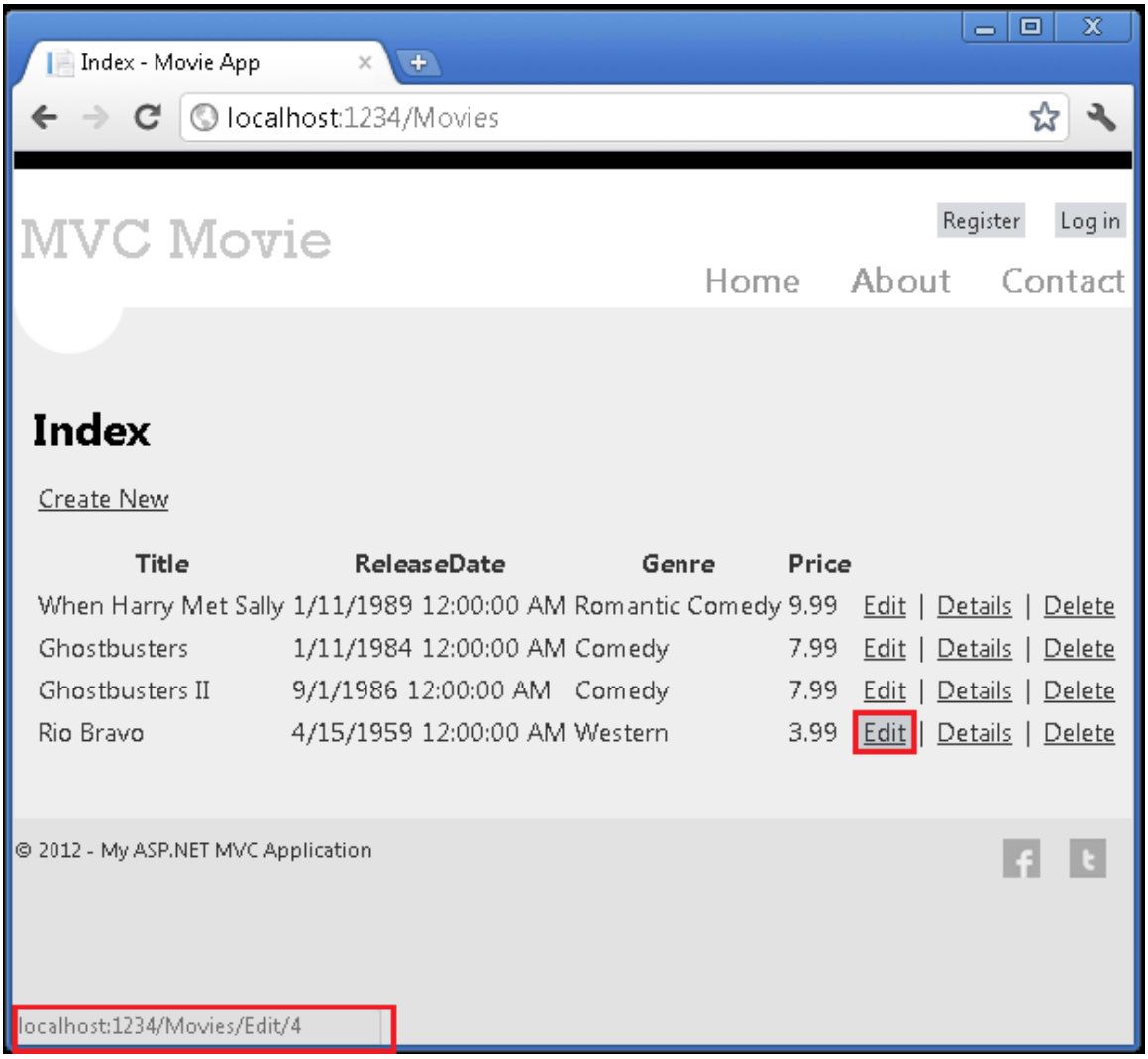


现在，您可以在简单列表页面里，来显示数据库里的数据了。在下一次的教程中，我们会继续看看框架自动生成的其它代码。并添加一个 `SearchIndex` 方法和 `SearchIndex` 视图，使您可以在数据库中搜索电影了。

验证编辑方法和编辑视图

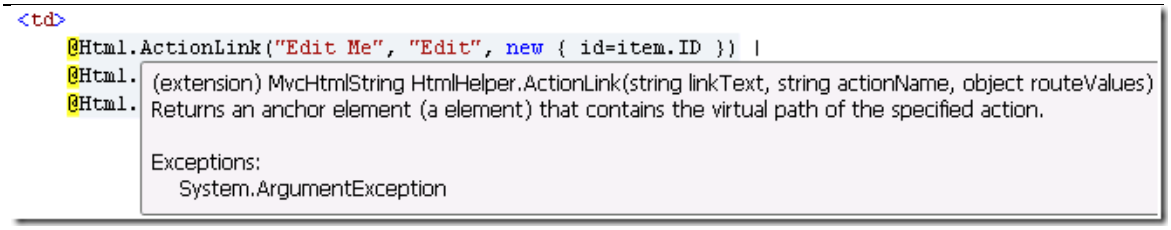
在本节中，您将开始修改为电影控制器所新加的操作方法和视图。然后，您将添加一个自定义的搜索页。

在浏览器地址栏里追加/Movies， 浏览到 Movies 页面。并进入编辑(Edit)页面。



Edit（编辑）链接是由 *Views\Movies\Index.cshtml* 视图中的 `Html.ActionLink` 方法所生成的：

```
@Html.ActionLink("Edit", "Edit", new { id=item.ID })
```

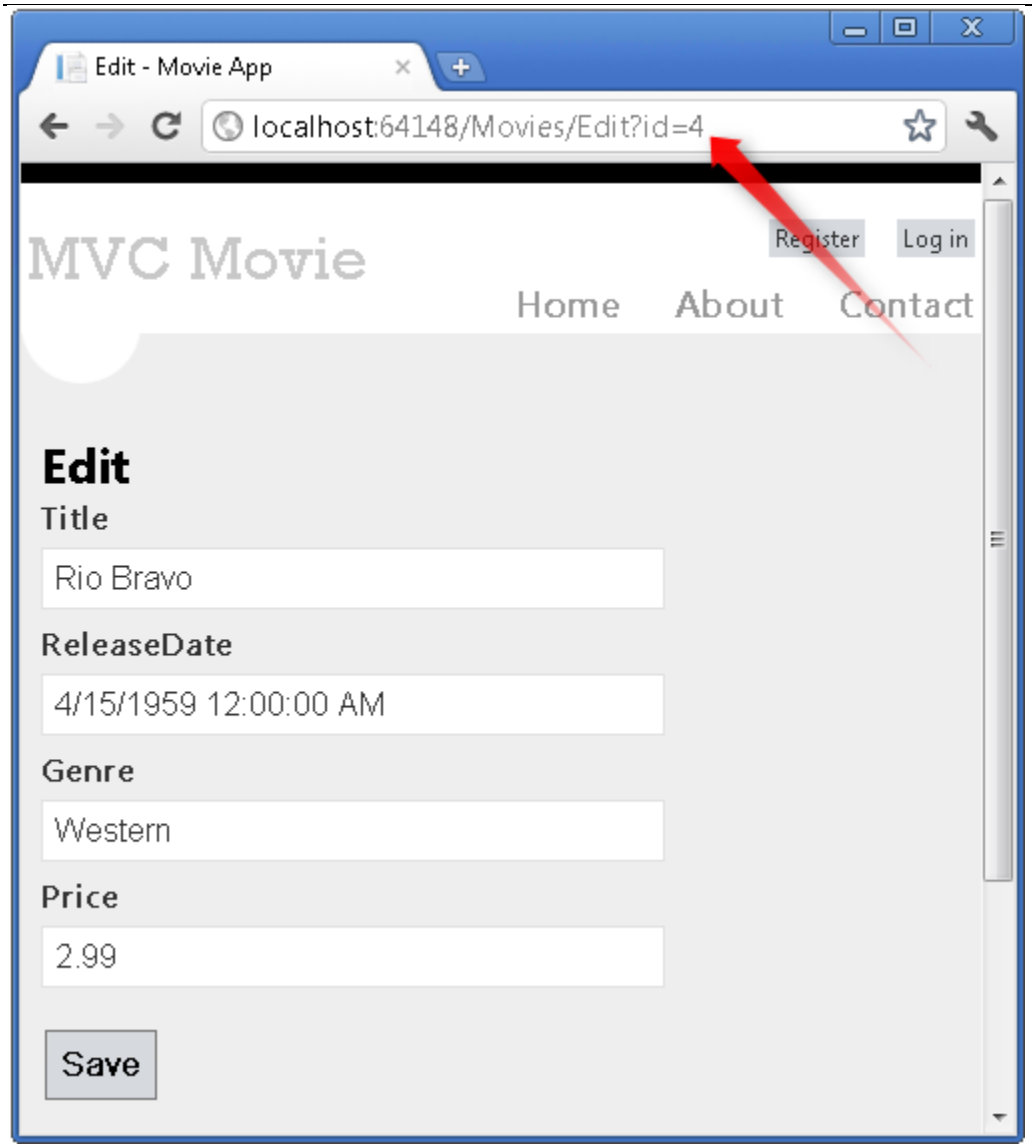


Html 对象是一个 Helper, 以属性的形式, 在 System.Web.Mvc.WebViewPage 基类上公开。 ActionLink 是一个帮助方法，便于动态生成指向 Controller 中操作方法 的 HTML 超链接链接。ActionLink 方法的第一个参数是想要呈现的链接文本（例如，<a>Edit Me)。第二个参数是要调用的操作方法的名称。最后一个参数是一个匿名对象，用来生成路由数据（在本例中，ID 为 4 的）。

在上图中所生成的链接是 *http://localhost:xxxxx/Movies/Edit/4* 默认的路由（在 *App_Start\RouteConfig.cs* 中设定）使用的 URL 匹配模式为: {controller}/{action}/{id}。因此，ASP.NET 将 *http://localhost:xxxxx/Movies/Edit/4* 转化到 Movies 控制器中 Edit 操作方法，参数 ID 等于 4 的请求。查看 *App_Start\RouteConfig.cs* 文件中的以下代码。

```
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 }
    );
}
```

您还可以使用 QueryString 来传递操作方法的参数。例如，URL: *http://localhost:xxxxx/Movies/Edit?ID=4* 还会将参数 ID 为 4 的请求传递给 Movies 控制器的 Edit 操作方法。



打开 `Movies` 控制器。如下所示的两个 `Edit` 操作方法。

```
//
// GET: /Movies/Edit/5

public ActionResult Edit(int id = 0)
{
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
```

```
}

//
// POST: /Movies/Edit/5

[HttpPost]
public ActionResult Edit(Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Entry(movie).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

注意，第二个 Edit 操作方法的上面有 HttpPost 属性。此属性指定了 Edit 方法的重载，此方法仅被 POST 请求所调用。您可以将 HttpGet 属性应用于第一个编辑方法，但这是不必要的，因为它是默认的属性。（操作方法会被隐式的指定为 HttpGet 属性，从而作为 HttpGet 方法。）

HttpGet Edit 方法会获取电影 ID 参数、 查找影片使用 Entity Framework 的 Find 方法，并返回到选定影片的编辑视图。如果不带参数调用 Edit 方法，ID 参数被指定为默认值 零。如果找不到一部电影，则返回 HttpNotFound 。当 VS 自动创建编辑视图时，它会查看 Movie 类并为类的每个属性创建用于 Render 的<label>和<input>的元素。下面的示例为自动创建的编辑视图：

```
@model MvcMovie.Models.Movie

@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>Movie</legend>

        @Html.HiddenFor(model => model.ID)

        <div class="editor-label">
```

```
            @Html.LabelFor(model => model.Title)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Title)
            @Html.ValidationMessageFor(model => model.Title)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.ReleaseDate)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.ReleaseDate)
            @Html.ValidationMessageFor(model => model.ReleaseDate)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Genre)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Genre)
            @Html.ValidationMessageFor(model => model.Genre)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Price)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Price)
            @Html.ValidationMessageFor(model => model.Price)
        </div>

        <p>
            <input type="submit" value="Save" />
        </p>
    </fieldset>
}

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

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

注意，视图模板在文件的顶部有 `@model MvcMovie.Models.Movie` 的声明，这将指定视图期望的模型类型为 `Movie`。

自动生成的代码，使用了 Helper 方法的几种简化的 HTML 标记。`Html.LabelFor` 用来显示字段的名称（"Title"、"ReleaseDate"、"Genre"或"Price"）。`Html.EditorFor` 用来呈现 HTML `<input>`元素。`Html.ValidationMessageFor` 用来显示与该属性相关联的任何验证消息。

运行该应用程序，然后浏览 URL，/Movies。单击 **Edit** 链接。在浏览器中查看页面源代码。HTML Form 中的元素如下所示：

```
<form action="/Movies/Edit/4" method="post">    <fieldset>
        <legend>Movie</legend>

        <input data-val="true" data-val-number="The field ID must be a
number." data-val-required="The ID field is required." id="ID" name="ID"
type="hidden" value="4" />

        <div class="editor-label">
            <label for="Title">Title</label>
        </div>
        <div class="editor-field">
            <input class="text-box single-line" id="Title" name="Title"
type="text" value="Rio Bravo" />
            <span class="field-validation-valid" data-valmsg-for="Title" data-
valmsg-replace="true"></span>
        </div>

        <div class="editor-label">
            <label for="ReleaseDate">ReleaseDate</label>
        </div>
        <div class="editor-field">
            <input class="text-box single-line" data-val="true" data-val-
date="The field ReleaseDate must be a date." data-val-required="The
ReleaseDate field is required." id="ReleaseDate" name="ReleaseDate"
type="text" value="4/15/1959 12:00:00 AM" />
            <span class="field-validation-valid" data-valmsg-for="ReleaseDate"
data-valmsg-replace="true"></span>
        </div>
```

```
        <div class="editor-label">
            <label for="Genre">Genre</label>
        </div>
        <div class="editor-field">
            <input class="text-box single-line" id="Genre" name="Genre"
type="text" value="Western" />
            <span class="field-validation-valid" data-valmsg-for="Genre" data-
valmsg-replace="true"></span>
        </div>

        <div class="editor-label">
            <label for="Price">Price</label>
        </div>
        <div class="editor-field">
            <input class="text-box single-line" data-val="true" data-val-
number="The field Price must be a number." data-val-required="The Price field
is required." id="Price" name="Price" type="text" value="2.99" />
            <span class="field-validation-valid" data-valmsg-for="Price" data-
valmsg-replace="true"></span>
        </div>

        <p>
            <input type="submit" value="Save" />
        </p>
    </fieldset>
</form>
```

被<form> HTML 元素所包括的 <input> 元素会被发送到，form 的 action 属性所设置的 URL: /Movies/Edit。单击 **Edit** 按钮时，from 数据将会被发送到服务器。

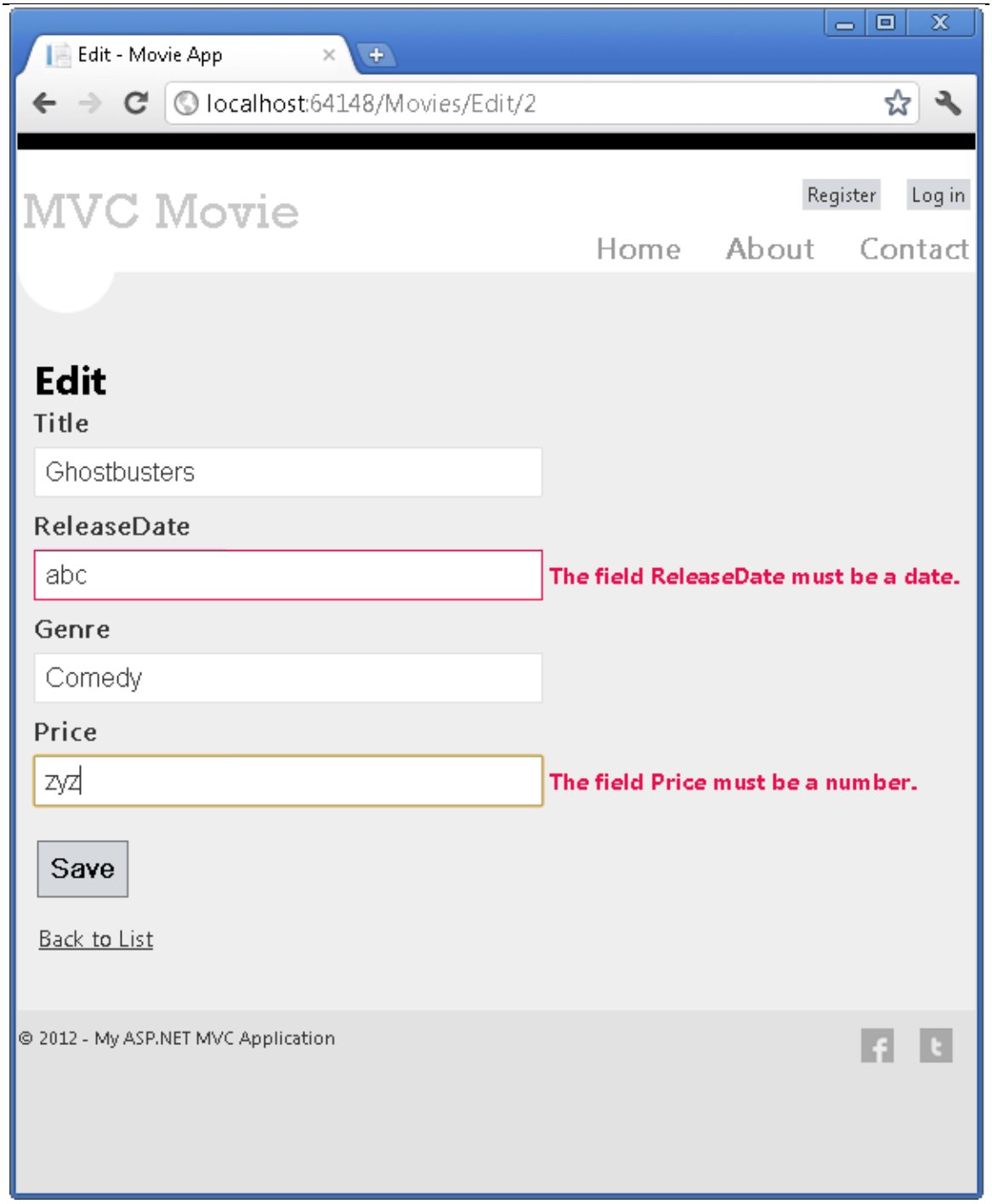
处理 POST 请求

下面的代码显示了 Edit 操作方法的 HttpPost 处理：

```
[HttpPost]
public ActionResult Edit(Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Entry(movie).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

ASP.NET MVC 模型绑定 接收 form 所 post 的数据，并转换所接收的 `movie` 请求数据从而创建一个 `Movie` 对象。`ModelState.IsValid` 方法用于验证提交的表单数据是否可用于修改（编辑或更新）一个 `Movie` 对象。如果数据是有效的电影数据，将保存到数据库的 `Movies` 集合(`MovieDBContext` instance)。通过调用 `MovieDBContext` 的 `SaveChanges` 方法，新的电影数据会被保存到数据库。数据保存之后，代码会把用户重定向到 `MoviesController` 类的 `Index` 操作方法，页面将显示电影列表，同时包括刚刚所做的更新。

如果 form 发送的值不是有效的值，它们将重新显示在 form 中。*Edit.cshtml* 视图模板中的 `Html.ValidationMessageFor` Helper 将用来显示相应的错误消息。



注意，为了使 jQuery 支持使用逗号的非英语区域的验证 ，需要设置逗号（"，"）来表示小数点，你需要引入 *globalize.js* 并且你还需要具体的指定 *cultures/globalize.cultures.js* 文件（地址在 https://github.com/jquery/globalize) 在 JavaScript 中可以使用 `Globalize.parseFloat`。下面的代码展示了在"FR-FR" Culture 下的 Views\Movies\Edit.cshtml 视图：

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
    <script src="~/Scripts/globalize.js"></script>
    <script src="~/Scripts/globalize.culture.fr-FR.js"></script>
    <script>
        $.validator.methods.number = function (value, element) {
            return this.optional(element) ||
                !isNaN(Globalize.parseFloat(value));
        }
        $(document).ready(function () {
            Globalize.culture('fr-FR');
        });
    </script>
    <script>
        jQuery.extend(jQuery.validator.methods, {
            range: function (value, element, param) {
                //Use the Globalization plugin to parse the value
                var val = $.global.parseFloat(value);
                return this.optional(element) || (
                    val >= param[0] && val <= param[1]);
            }
        });

    </script>
}
```

十进制字段可能需要逗号，而不是小数点。作为临时的修复，您可以向项目根 web.config 文件添加的全球化设置。下面的代码演示设置为美国英语的全球化文化设置。

```
  <system.web>
    <globalization culture ="en-US" />
    <!--elements removed for clarity-->
  </system.web>
```

所有 `HttpGet` 方法都遵循类似的模式。它们获取影片对象（或对象集合，如 `Index` 里的对象集合），并将模型传递给视图。`Create` 方法将一个空的 Movie 对象传递给创建视图。创建、 编辑、 删除或以其它方式修改数据的方法都是 `HttpPost` 方法。使用 HTTP GET 方法来修改数据是存在安全风险，在 ASP.NET MVC Tip #46 – Don't use Delete Links because they create Security Holes 的 Blog 中有完整的叙述。在 GET 方法中修改数据还违反了 HTTP 的最佳做法和 Rest 架构模式， GET 请求不应更改应用程序的状态。换句话说，执行 GET 操作，应该是一种安全的操作，没有任何副作用，不会修改您持久化的数据。

添加一个搜索方法和搜索视图

在本节中，您将添加一个搜索电影流派或名称的 SearchIndex 操作方法。这将可使用 */Movies/SearchIndex* URL。该请求将显示一个 HTML 表单，其中包含输入的元素，用户可以输入一部要搜索的电影。当用户提交窗体时，操作方法将获取用户输入的搜索条件并在数据库中搜索。

显示 SearchIndex 窗体

通过将 SearchIndex 操作方法添加到现有的 MoviesController 类开始。该方法将返回一个视图包含一个 HTML 表单。如下代码：

```
public ActionResult SearchIndex(string searchString)
{
    var movies = from m in db.Movies
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(movies);
}
```

SearchIndex 方法的第一行创建以下的 LINQ 查询，以选择看电影：

```
var movies = from m in db.Movies
                 select m;
```

查询在这一点上，只是定义，并还没有执行到数据上。

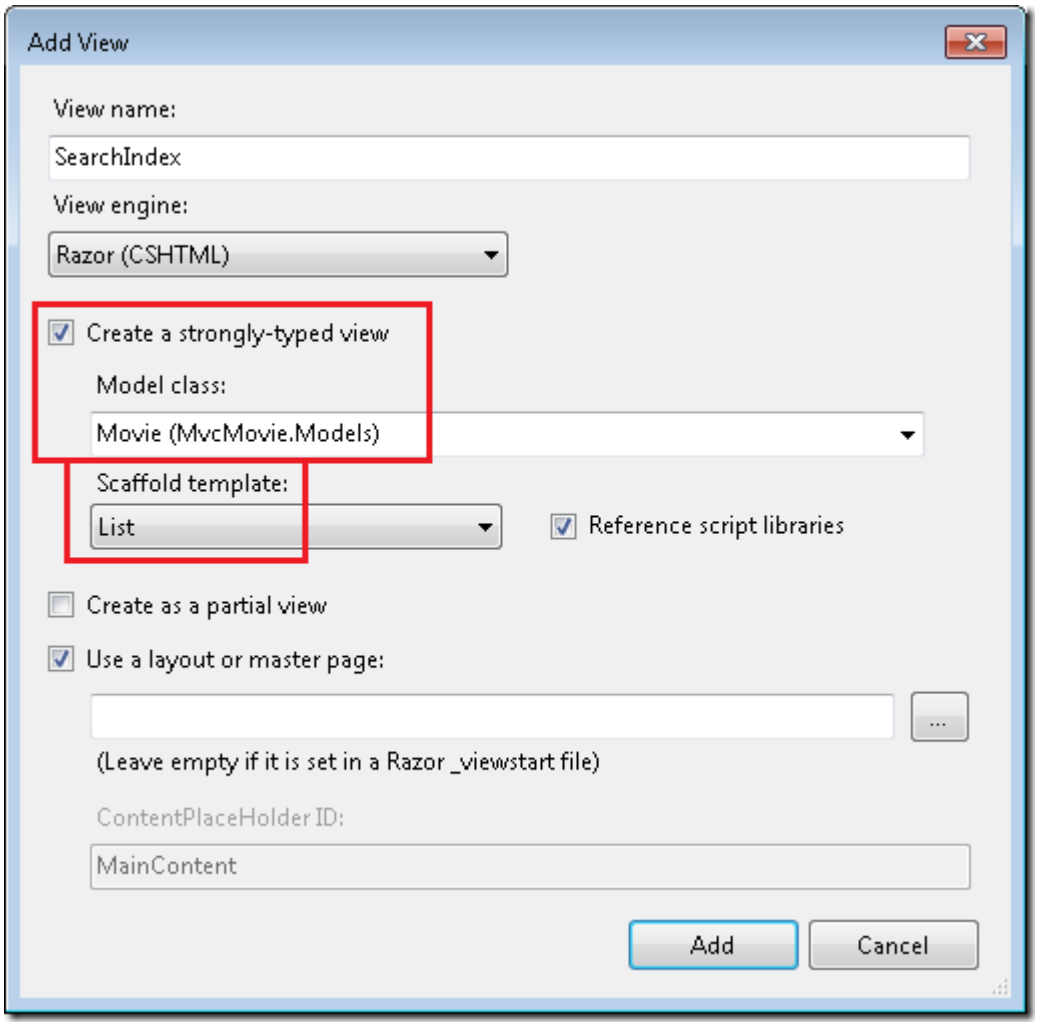
如果 searchString 参数包含一个字符串，可以使用下面的代码，修改电影查询要筛选的搜索字符串：

```
    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }
```

上面 `s => s.Title` 代码是一个 Lambda 表达式。Lambda 是基于方法的 LINQ 查询，（例如上面的 where 查询）在上面的代码中使用了标准查询参数运算符的方法。当定义

LINQ 查询或修改查询条件时（如调用 `Where` 或 `OrderBy` 方法时，不会执行 LINQ 查询。相反，查询执行会被延迟，这意味着表达式的计算延迟，直到取得实际的值或调用 `ToList` 方法。在 `SearchIndex` 示例中，SearchIndex 视图中执行查询。有关延迟的查询执行的详细信息，请参阅 Query Execution.

现在，您可以实现 `SearchIndex` 视图并将其显示给用户。在 `SearchIndex` 方法内单击右键，然后单击**添加视图**。在**添加视图**对话框中，指定你要将 `Movie` 对象传递给视图模板作为其模型类。在**框架模板**列表中，选择**列表**，然后单击**添加**.



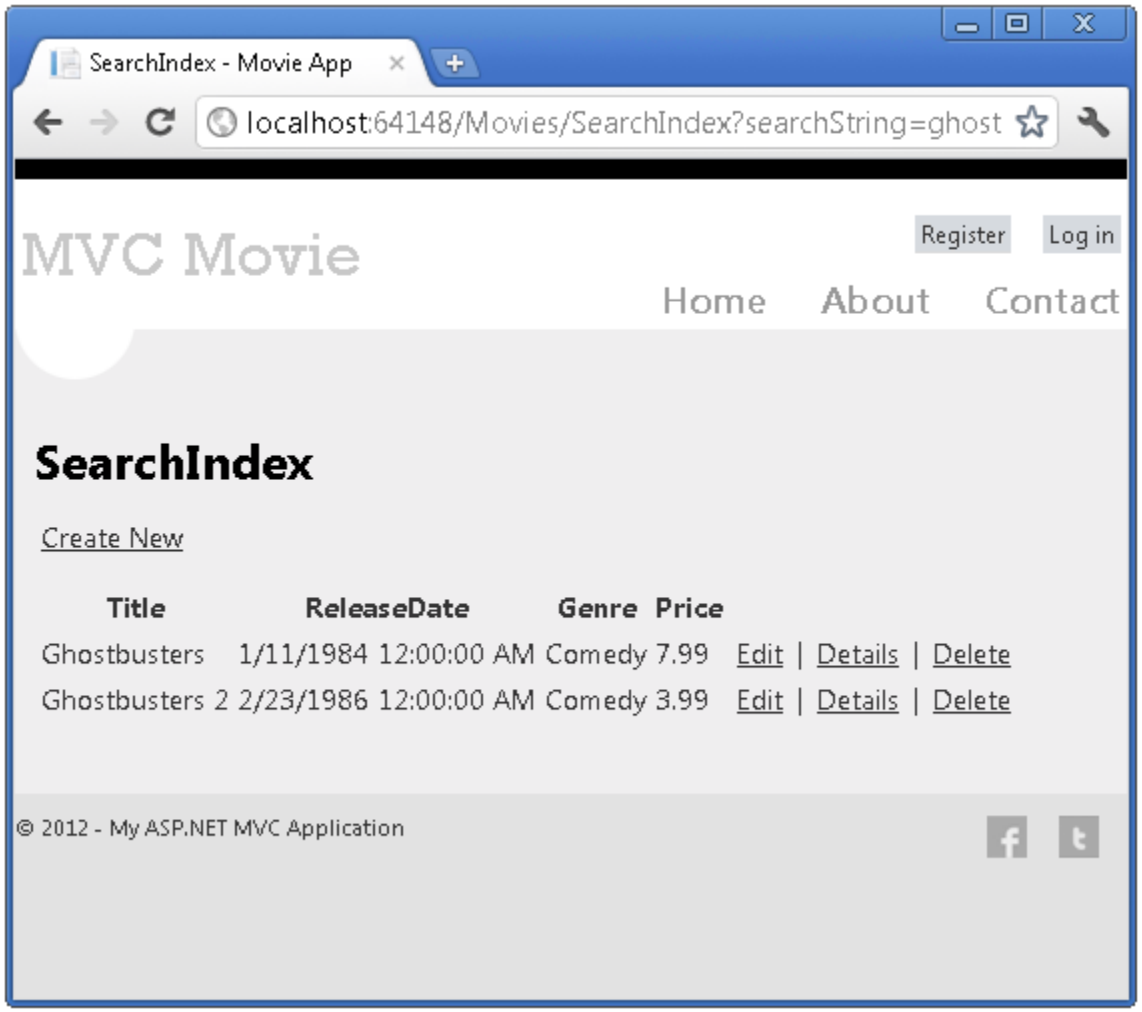
当您单击**添加**按钮时，创建了 *Views\Movies\SearchIndex.cshtml* 视图模板。因为你选中了**框架模板**的列表，Visual Studio 将自动生成**列表**视图中的某些默认标记。框架模版创建了

HTML 表单。它会检查 Movie 类，并为类的每个属性创建用来展示的<label>元素。下面是生成的视图：

```
@model IEnumerable<MvcMovie.Models.Movie>

@{
    ViewBag.Title = "SearchIndex";
}

<h2>SearchIndex</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table>
    <tr>
        <th>
            Title
        </th>
        <th>
            ReleaseDate
        </th>
        <th>
            Genre
        </th>
        <th>
            Price
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Genre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
```

```
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", new { id=item.ID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.ID })
        </td>
    </tr>
}

</table>
```

运行该应用程序，然后转到 */Movies/SearchIndex*。追加查询字符串到 URL 如 `?searchString=ghost`。显示已筛选的电影。



如果您更改 SearchIndex 方法的签名，改为参数 id，在 *Global.asax* 文件中设置的默认路由将使得: id 参数将匹配{id}占位符。

```
{controller}/{action}/{id}
```

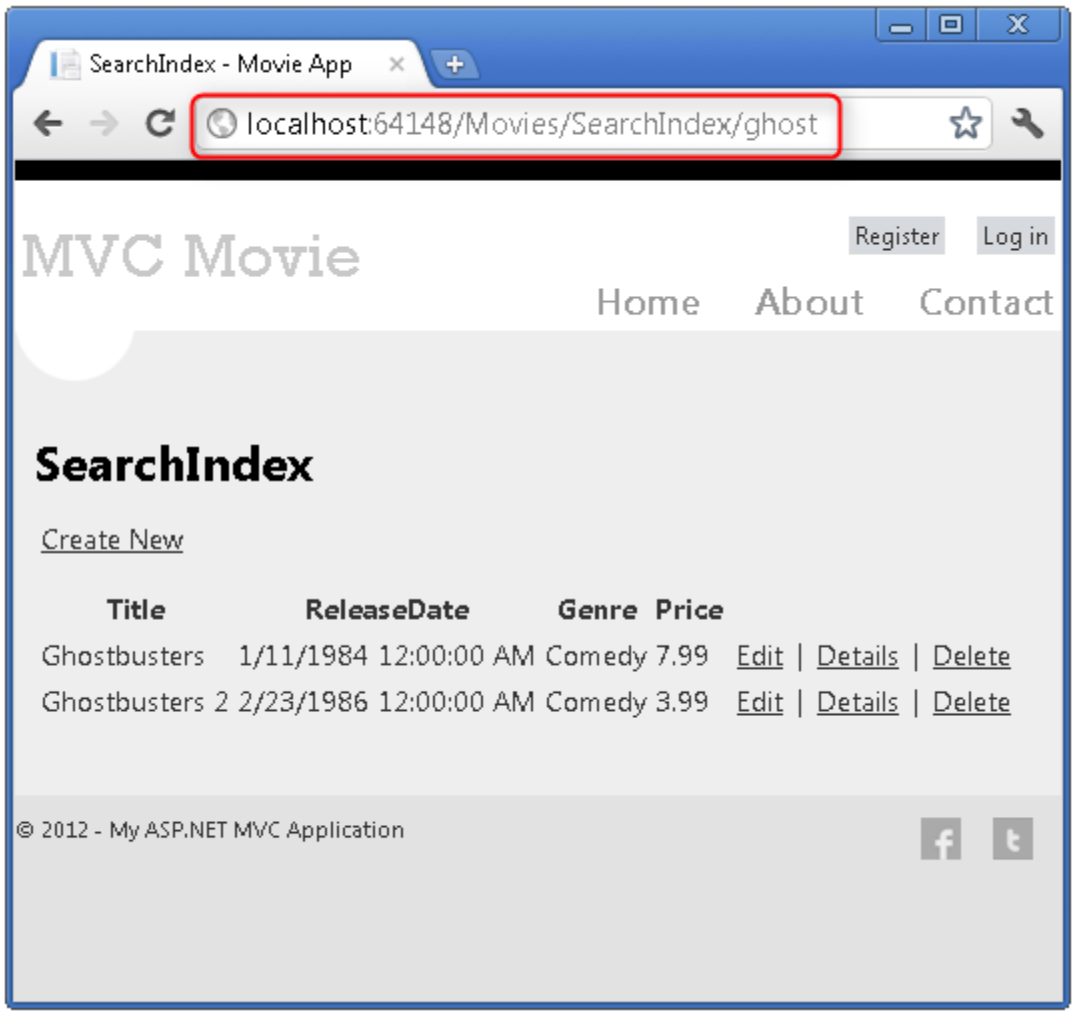
原来的 SearchIndex 方法看起来是这样的:

```
public ActionResult SearchIndex(string searchString)
{
    var movies = from m in db.Movies
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(movies);
}
```

修改后的 SearchIndex 方法将如下所示：

```
public ActionResult SearchIndex(string id)
{
    string searchString = id;
    var movies = from m in db.Movies
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(movies);
}
```

您现在可以将搜索标题作为路由数据（部分 URL）来替代 QueryString。



但是，每次用户想要搜索一部电影时， 你不能指望用户去修改 URL。所以，现在您将添加 UI 页面，以帮助他们去筛选电影。如果您更改了的 `SearchIndex` 方法来测试如何传递路由绑定的 ID 参数，更改它，以便您的 `SearchIndex` 方法采用字符串 `searchString` 参数:

```
public ActionResult SearchIndex(string searchString)
{
     var movies = from m in db.Movies
                  select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(movies);
}
```

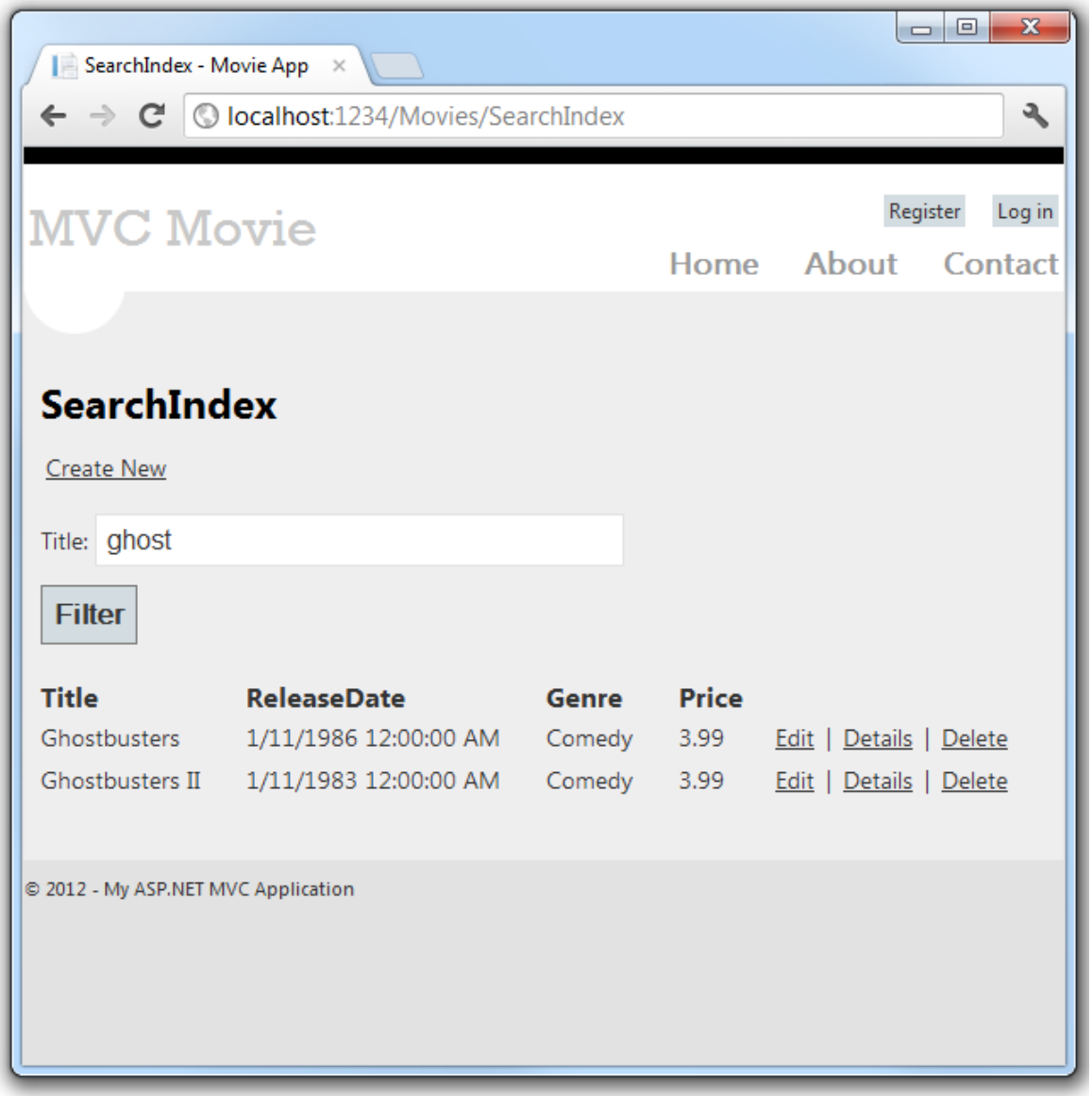
打开 *Views\Movies\SearchIndex.cshtml* 文件，并在 `@Html.ActionLink("Create New", "Create")`后面，添加以下内容：

```
@using (Html.BeginForm()){
         <p> Title: @Html.TextBox("SearchString")<br />
         <input type="submit" value="Filter" /></p>
        }
```

下面的示例展示了添加后， *Views\Movies\SearchIndex.cshtml* 文件的一部分:

```
@model IEnumerable<MvcMovie.Models.Movie>

@{
    ViewBag.Title = "SearchIndex";
}

<h2>SearchIndex</h2>

<p>
    @Html.ActionLink("Create New", "Create")

     @using (Html.BeginForm()){
         <p> Title: @Html.TextBox("SearchString") <br />
         <input type="submit" value="Filter" /></p>
        }
</p>
```

`Html.BeginForm Helper` 创建开放`<form>`标记。`Html.BeginForm` Helper 将使得, 在用户通过单击**筛选**按钮提交窗体时,窗体 Post 本 Url。运行该应用程序，请尝试搜索一部电影。

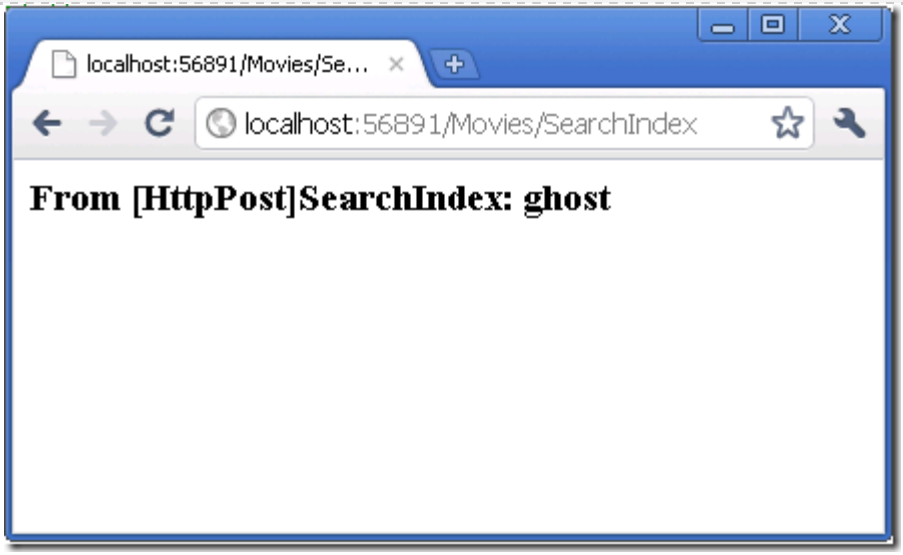


SearchIndex 没有 HttpPost 的重载方法。你并不需要它，因为该方法并不更改应用程序数据的状态，只是筛选数据。

您可以添加如下的 HttpPost SearchIndex 方法。在这种情况下，请求将进入 HttpPost SearchIndex 方法， HttpPost SearchIndex 方法将返回如下图的内容。

```
[HttpPost]
public string SearchIndex(FormCollection fc, string searchString)
{
```

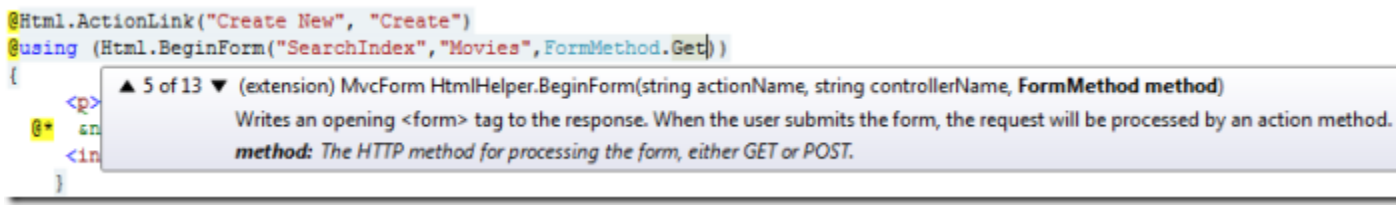
```
    return "<h3> From [HttpPost]SearchIndex: " + searchString + "</h3>";
}
```



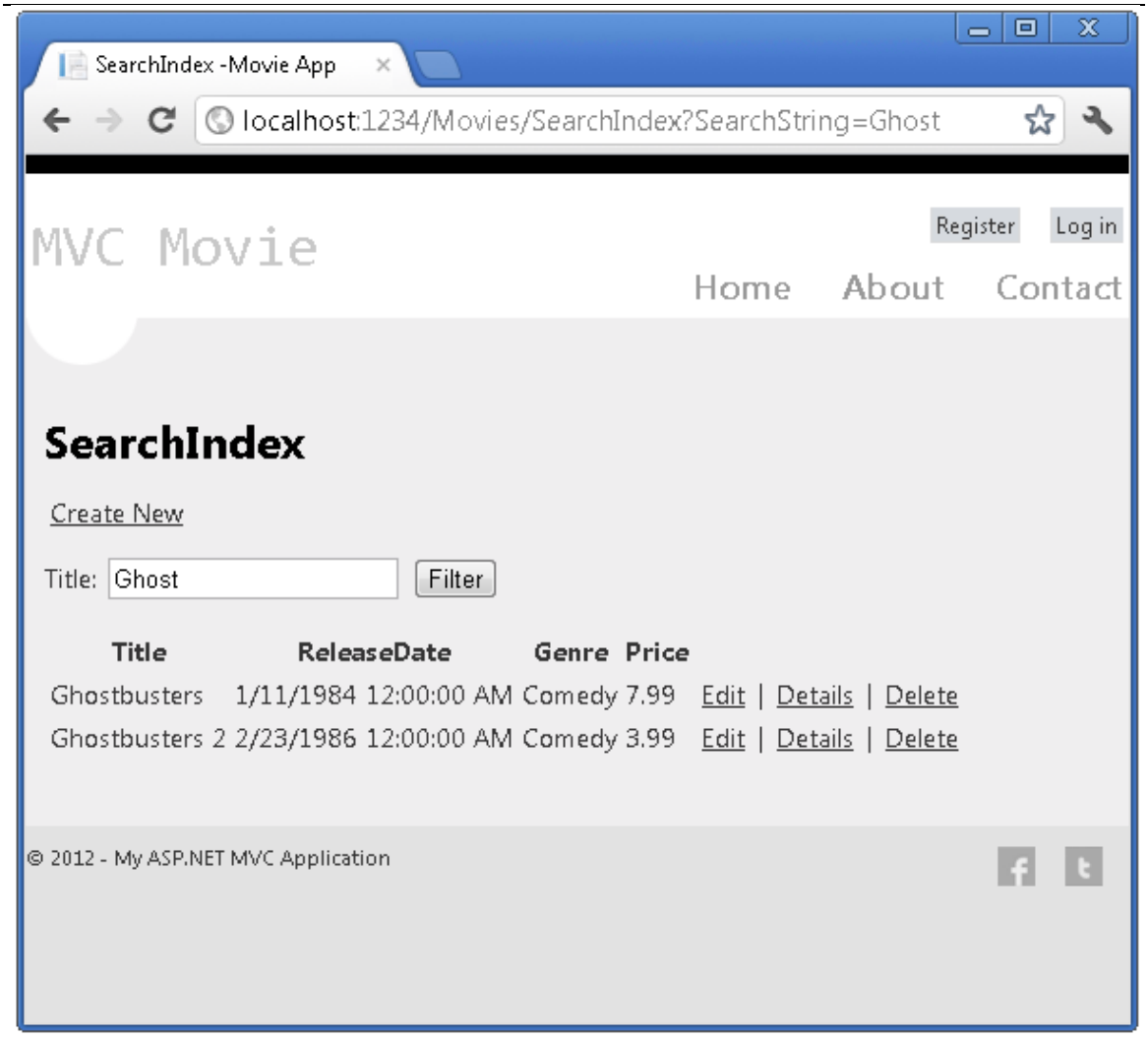
但是，即使您添加此 `HttpPost SearchIndex` 方法，这一实现其实是有局限的。想象一下您想要添加书签给特定的搜索，或者您想要把搜索链接发送给朋友们，他们可以通过单击看到一样的电影搜索列表。请注意 HTTP POST 请求的 URL 和 GET 请求的 URL 是相同的（localhost:xxxxx/电影/SearchIndex）—— 在 URL 中没有搜索信息。现在，搜索字符串信息作为窗体字段值，发送到服务器。这意味着您不能在 URL 中捕获此搜索信息，以添加书签或发送给朋友。

解决方法是使用重载的 `BeginForm` ，它指定 POST 请求应添加到 URL 的搜索信息，并应该路由到 HttpGet `SearchIndex` 方法。将现有的无参数 `BeginForm` 方法，修改为以下内容：

```
@using (Html.BeginForm("SearchIndex","Movies",FormMethod.Get))
```



现在当您提交搜索，该 URL 将包含搜索的查询字符串。搜索还会请求到 `HttpGet SearchIndex` 操作方法，即使您也有一个 `HttpPost SearchIndex` 方法。



按照电影流派添加搜索

如果您添加了 `HttpPost` 的 `SearchIndex` 方法，请立即删除它。

接下来，您将添加功能可以让用户按流派搜索电影。将 `SearchIndex` 方法替换成下面的代码：

```
public ActionResult SearchIndex(string movieGenre, string searchString)
{
    var GenreLst = new List<string>();

    var GenreQry = from d in db.Movies
                   orderby d.Genre
                   select d.Genre;
```

```
    GenreLst.AddRange(GenreQry.Distinct());
    ViewBag.movieGenre = new SelectList(GenreLst);

    var movies = from m in db.Movies
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    if (string.IsNullOrEmpty(movieGenre))
        return View(movies);
    else
    {
        return View(movies.Where(x => x.Genre == movieGenre));
    }

}
```

这版的 SearchIndex 方法将接受一个附加的 movieGenre 参数。前几行的代码会创建一个 List 对象来保存数据库中的电影流派。

下面的代码是从数据库中检索所有流派的 LINQ 查询。

```
var GenreQry = from d in db.Movies
                   orderby d.Genre
                   select d.Genre;
```

该代码使用泛型 List 集合的 AddRange 方法将所有不同的流派，添加到集合中的。(使用 Distinct 修饰符，不会添加重复的流派 -- 例如，在我们的示例中添加了两次喜剧)。该代码然后在 ViewBag 对象中存储了流派的数据列表。

下面的代码演示如何检查 movieGenre 参数。如果它不是空的，代码进一步指定了所查询的电影流派。

```
 if (string.IsNullOrEmpty(movieGenre))
        return View(movies);
    else
```

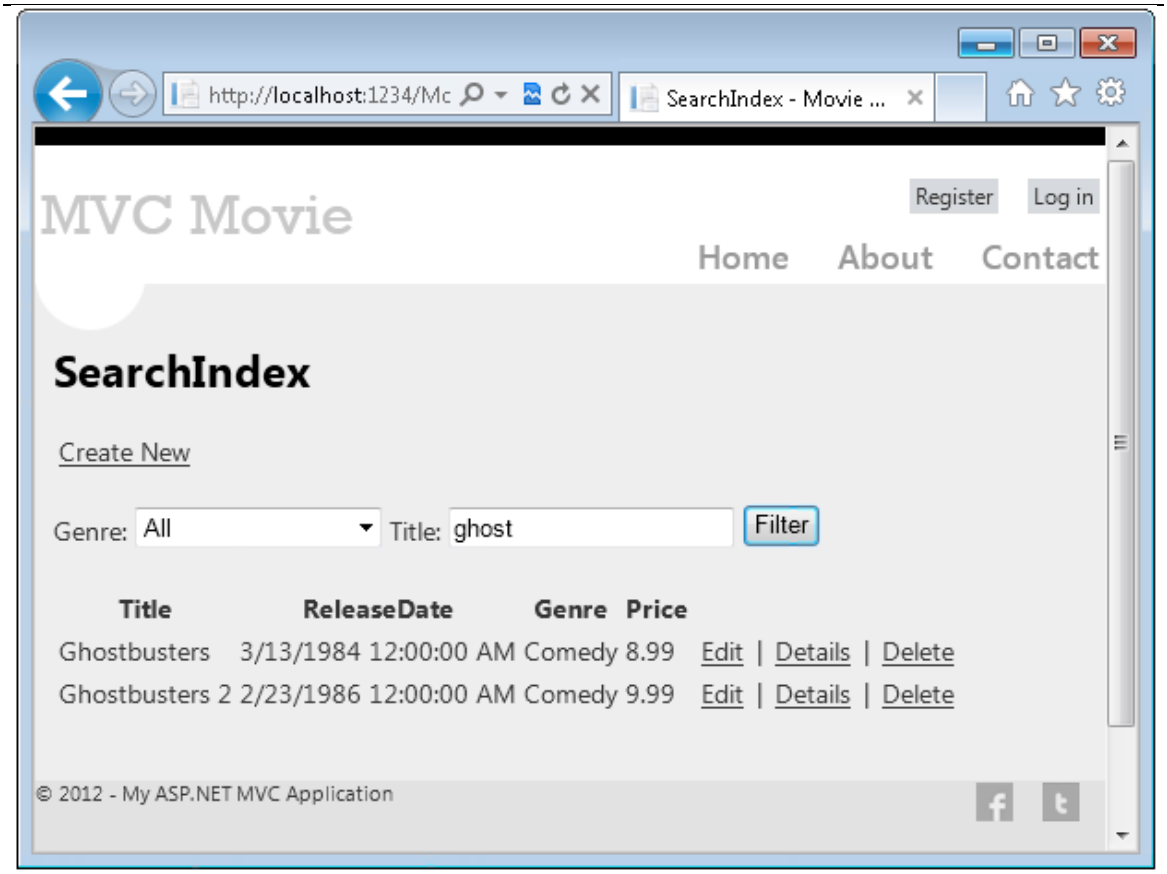
```
    {
        return View(movies.Where(x => x.Genre == movieGenre));
    }
```

在 SearchIndex 视图中添加选择框支持按流派搜索

在 `TextBox` Helper 之前添加 `Html.DropDownList` Helper 到 *Views\Movies\SearchIndex.cshtml* 文件中。添加完成后，如下面所示：

```
<p>
    @Html.ActionLink("Create New", "Create")
    @using (Html.BeginForm("SearchIndex","Movies",FormMethod.Get)){
         <p>Genre: @Html.DropDownList("movieGenre", "All")
           Title: @Html.TextBox("SearchString")
         <input type="submit" value="Filter" /></p>
        }
</p>
```

运行该应用程序并浏览 */Movies/SearchIndex*。按流派、 按电影名，或者同时这两者，来尝试搜索。



在这一节中您修改了 CRUD 操作方法和框架所生成的视图。您创建了一个搜索操作方法和视图，让用户可以搜索电影标题和流派。在下一节中，您将看到如何将属性添加到 `Movie` 模型，以及如何添加一个初始设定并自动创建一个测试数据库。

给电影表和模型添加新字段

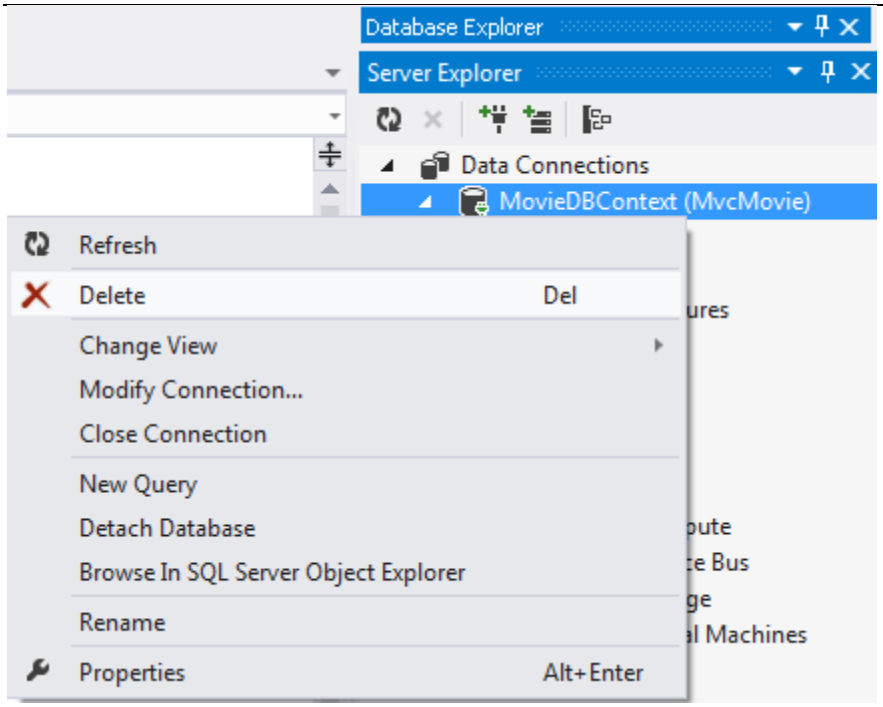
在本节中，您将使用 Entity Framework Code First 来实现模型类上的操作。从而使得这些操作和变更，可以应用到数据库中。

默认情况下，就像您在之前的教程中所作的那样，使用 Entity Framework Code First 自动创建一个数据库，Code First 为数据库所添加的表，将帮助您跟踪数据库是否和从它生成的模型类是同步的。如果他们不是同步的，Entity Framework 将抛出一个错误。这非常方便的在开发时就可以发现错误，否则您可能会在运行时才发现这个问题。（由一个晦涩的错误信息，才发现这个问题。）

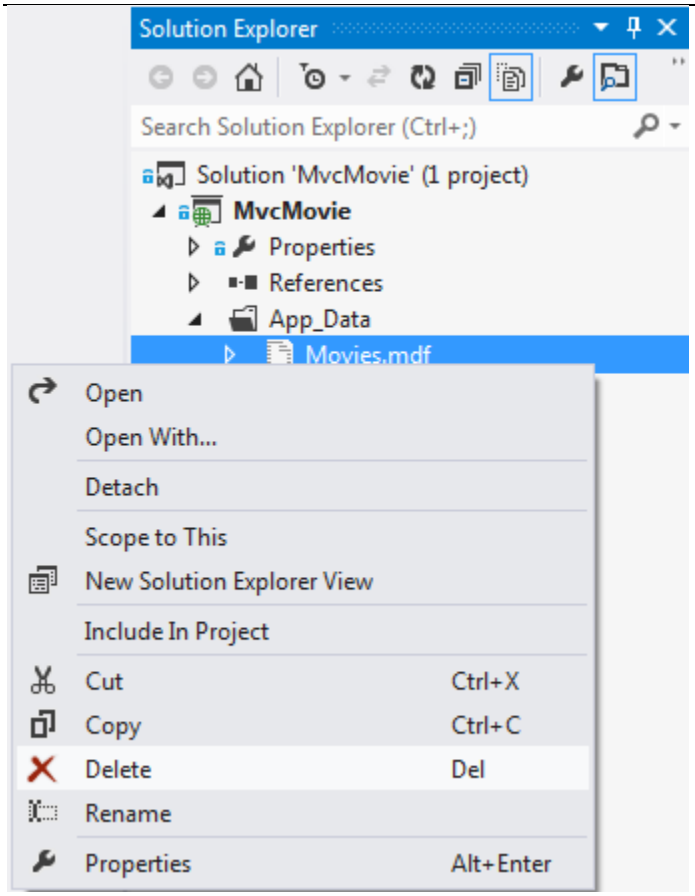
为对象模型的变更设置 Code First Migrations

如果您使用的是 Visual Studio 2012，从解决方案资源管理器中双击 *Movies.mdf*，打开数据库工具。Visual Studio Express for Web 将显示数据库资源管理器，Visual Studio 2012 将显示服务器资源管理器。如果您使用的是 Visual Studio 2010，请使用 SQL Server 对象资源管理器。

在数据库工具（数据库资源管理器、 服务器资源管理器或 SQL Server 对象资源管理器），右键单击 `MovieDBContext` ，并选择**删除**以删除电影数据库。

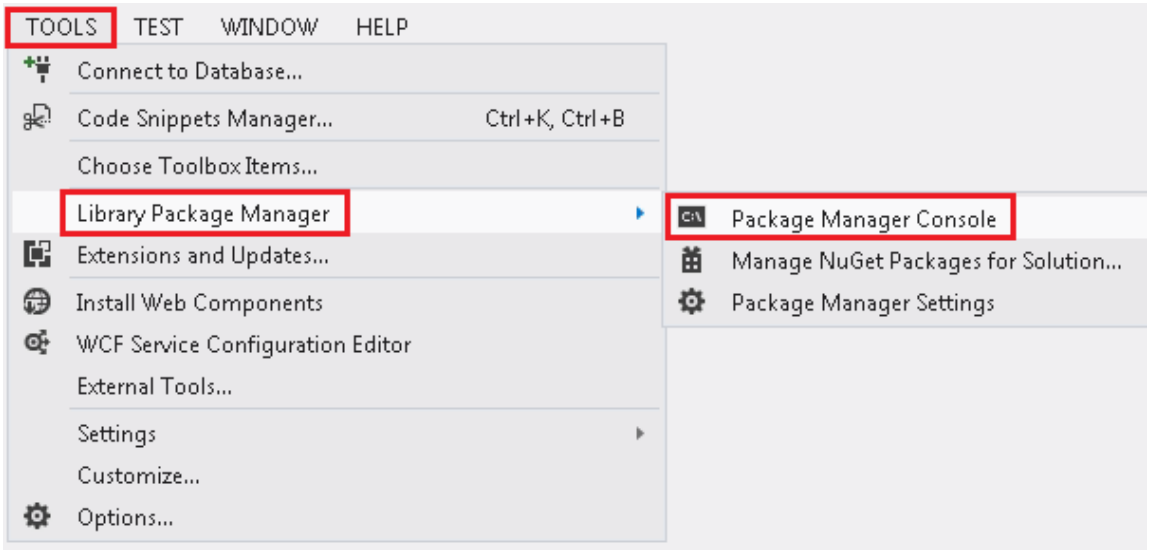


返回到解决方案资源管理器。在 Movies.mdf 文件上右键单击，并选择删除以删除电影数据库。

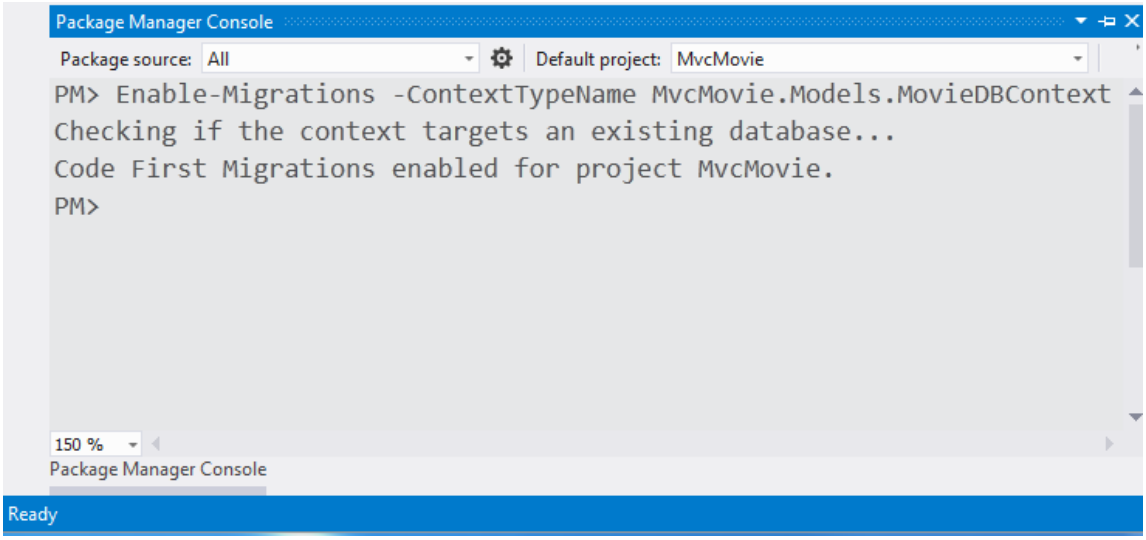


Build 应用程序，以确保没有任何编译错误。

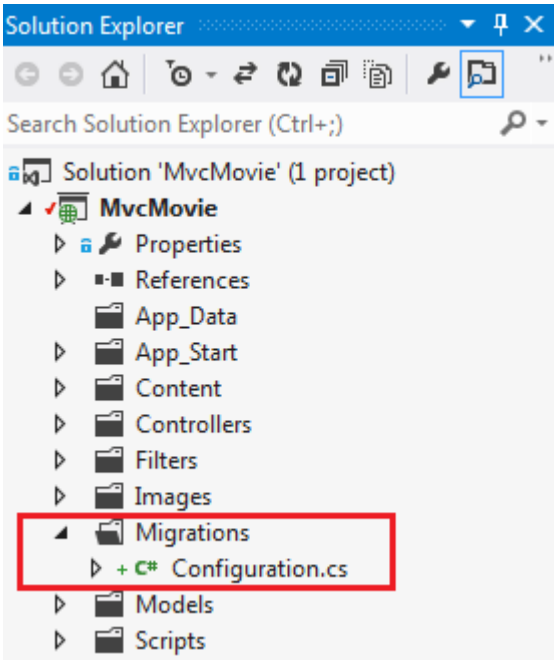
从**工具**菜单上，单击**库包管理器**，然后点击**程序包管理器控制台**.



在 **软件包管理器控制台** 窗口中 PM> 提示符下输入"Enable-Migrations –ContextTypeName MvcMovie.Models.MovieDBContext"。



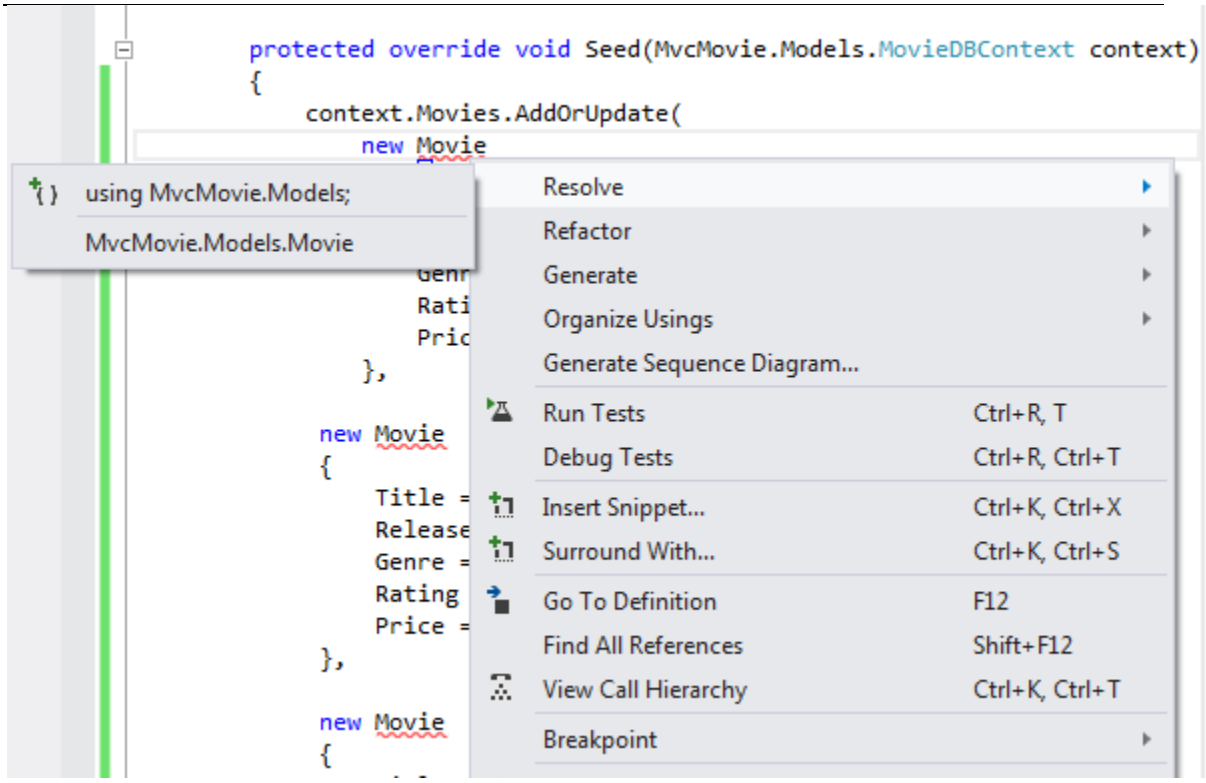
（如上所示）**Enable-Migrations** 命令会在 Migrations 文件夹中创建一个 Configuration.cs 文件。



在 Visual Studio 中打开 Configuration.cs 文件。把 Configuration.cs 文件中的 Seed 方法，替换为下面的代码：

```
protected override void Seed(MvcMovie.Models.MovieDBContext context)
{
    context.Movies.AddOrUpdate( i => i.Title,
        new Movie
        {
            Title = "When Harry Met Sally",
            ReleaseDate = DateTime.Parse("1989-1-11"),
            Genre = "Romantic Comedy",
            Price = 7.99M
        },

         new Movie
         {
             Title = "Ghostbusters ",
             ReleaseDate = DateTime.Parse("1984-3-13"),
             Genre = "Comedy",
             Price = 8.99M
         },

         new Movie
         {
             Title = "Ghostbusters 2",
             ReleaseDate = DateTime.Parse("1986-2-23"),
             Genre = "Comedy",
             Price = 9.99M
         },

       new Movie
       {
           Title = "Rio Bravo",
           ReleaseDate = DateTime.Parse("1959-4-15"),
           Genre = "Western",
           Price = 3.99M
       }
   );

}
```

在 Movie 下面出现的红色波浪线上右键单击，并选择 Resolve 然后点击 using MvcMovie.Models;



这样做之后，将添加以下的 using 语句:

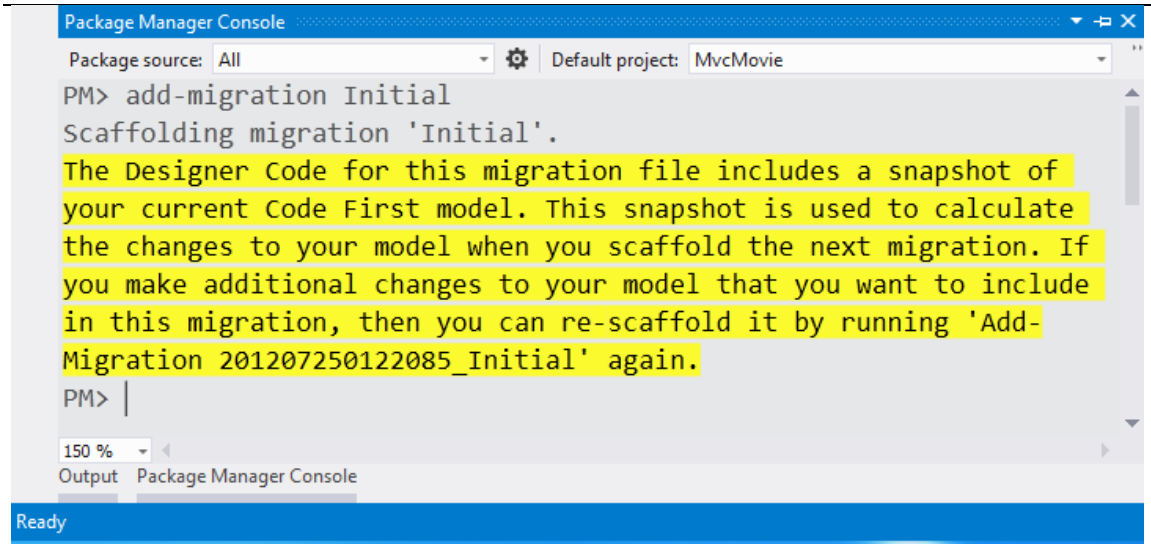
```
using MvcMovie.Models;
```

每次 Code First Migrations 会调用 Seed 方法（即，在程序包管理器控制台中调用 **update-database**），并且此次调用会更新行：更新已经插入的行，或把不存在的行也插入。

按 CTRL-SHIFT-B 来 Build 工程。（如果此次 Build 不成功，以下的步骤将会失败。）

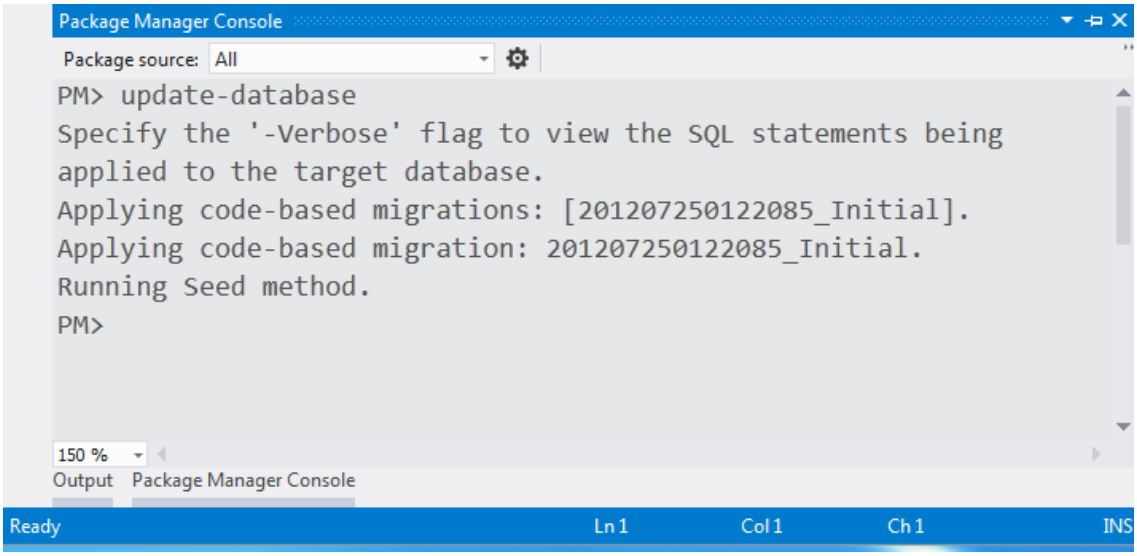
下一步是创建一个 `DbMigration` 类，用于初始化数据库迁移。此迁移类将创建新的数据库，这也就是为什么在之前的步骤中你要删除 `movie.mdf` 文件。

在**软件包管理器控制台**窗口中，输入"add-migration Initial"命令来创建初始迁移。" Initial" 的名称是任意，是用于创建迁移文件的名称。



Code First Migrations 将会在 Migrations 文件夹中创建另一个类文件（文件名为: *{DateStamp}_Initial.cs*），此类中包含的代码将创建数据库的 Schema。迁移文件名使用时间戳作为前缀，以帮助用来排序和查找。查看*{DateStamp}_Initial.cs* 文件，它包含了为电影数据库创建电影表的说明。当您更新数据库时，*{DateStamp}_Initial.cs* 文件将会被运行并创建 DB 的 Schema。然后 **Seed** 方法将运行，用来填充 DB 的测试数据。

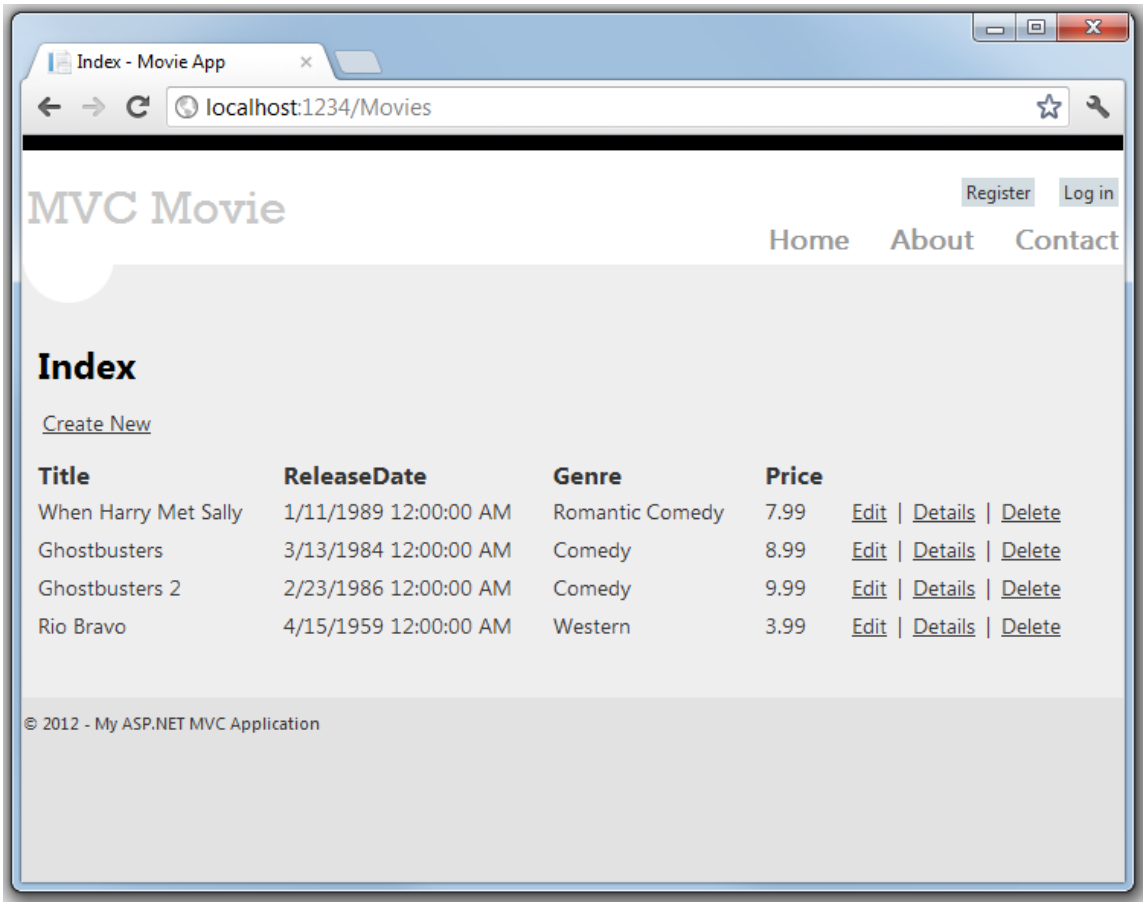
在**软件包管理器控制台**中，输入命令" update-database "，创建数据库并运行 **Seed** 方法。



如果您收到表已经存在并且无法创建的错误，可能是因为您已经删除了数据库，并且在执行 `update-database` 之前，您运行了应用程序。在这种情况下，再次删除 Movies.mdf 文

件，然后重试 `update-database` 命令。如果您仍遇到错误，删除 Migration 文件夹及其内容，然后从头开始重做。（即删除 Movies.mdf 文件，然后再进行 Enable-Migrations）

运行该应用程序，然后浏览 URL /Movies Seed 数据显示如下:



为影片模型添加评级属性

给现有的 `Movie` 类，添加新的 `Rating` 属性。打开 Models\Movie.cs 文件并添加如下 `Rating` 属性：

```
public string Rating { get; set; }
```

完整的 `Movie` 类如下：

```
    public class Movie
    {
        public int ID { get; set; }
```

```
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
        public string Rating { get; set; }
    }
```

Build 应用程序 **Build**>**Build Move** 或 CTRL-SHIFT-B.

现在，您已经更新了 `Model` 类，您还需要更新*\Views\Movies\Index.cshtml* 和 *\Views\Movies\Create.cshtml* 视图模板，以便能在浏览器中显示新的 `Rating` 属性。

打开*\Views\Movies\Index.cshtml* 文件，在 **Price** 列后面添加`<th>Rating</th>`的列头。然后添加一个`<td>`列来显示`@item.Rating` 的值。下面是更新的 *Index.cshtml* 视图模板：

```
@model IEnumerable<MvcMovie.Models.Movie>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Title)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.ReleaseDate)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Genre)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Price)
        </th>
         <th>
```

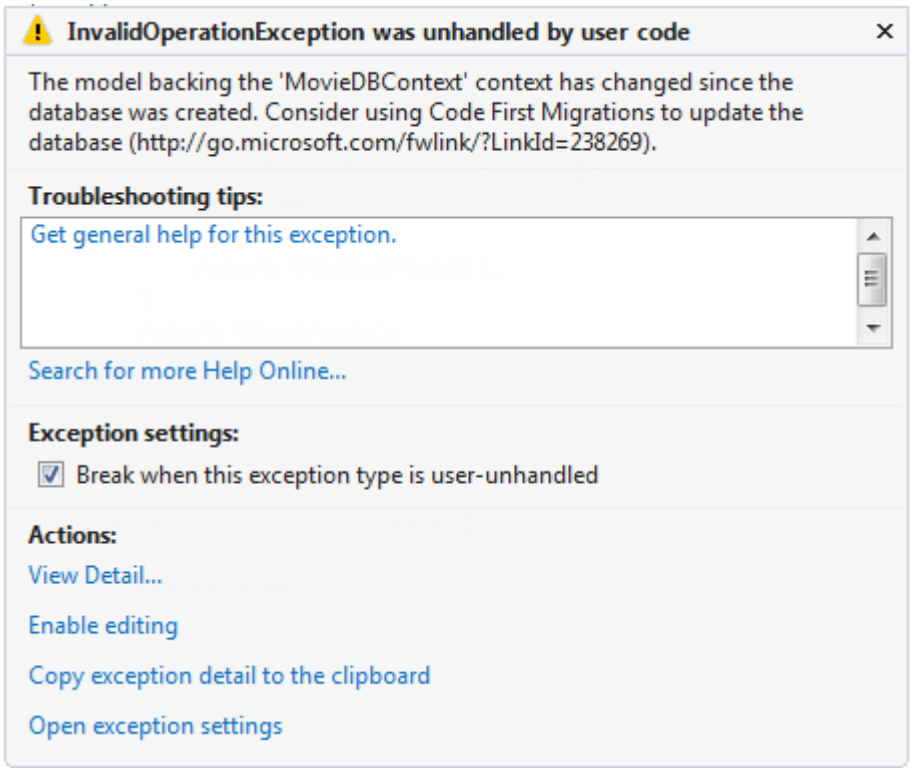
```
            @Html.DisplayNameFor(model => model.Rating)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Genre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Rating)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", new { id=item.ID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.ID })
        </td>
    </tr>
}

</table>
```

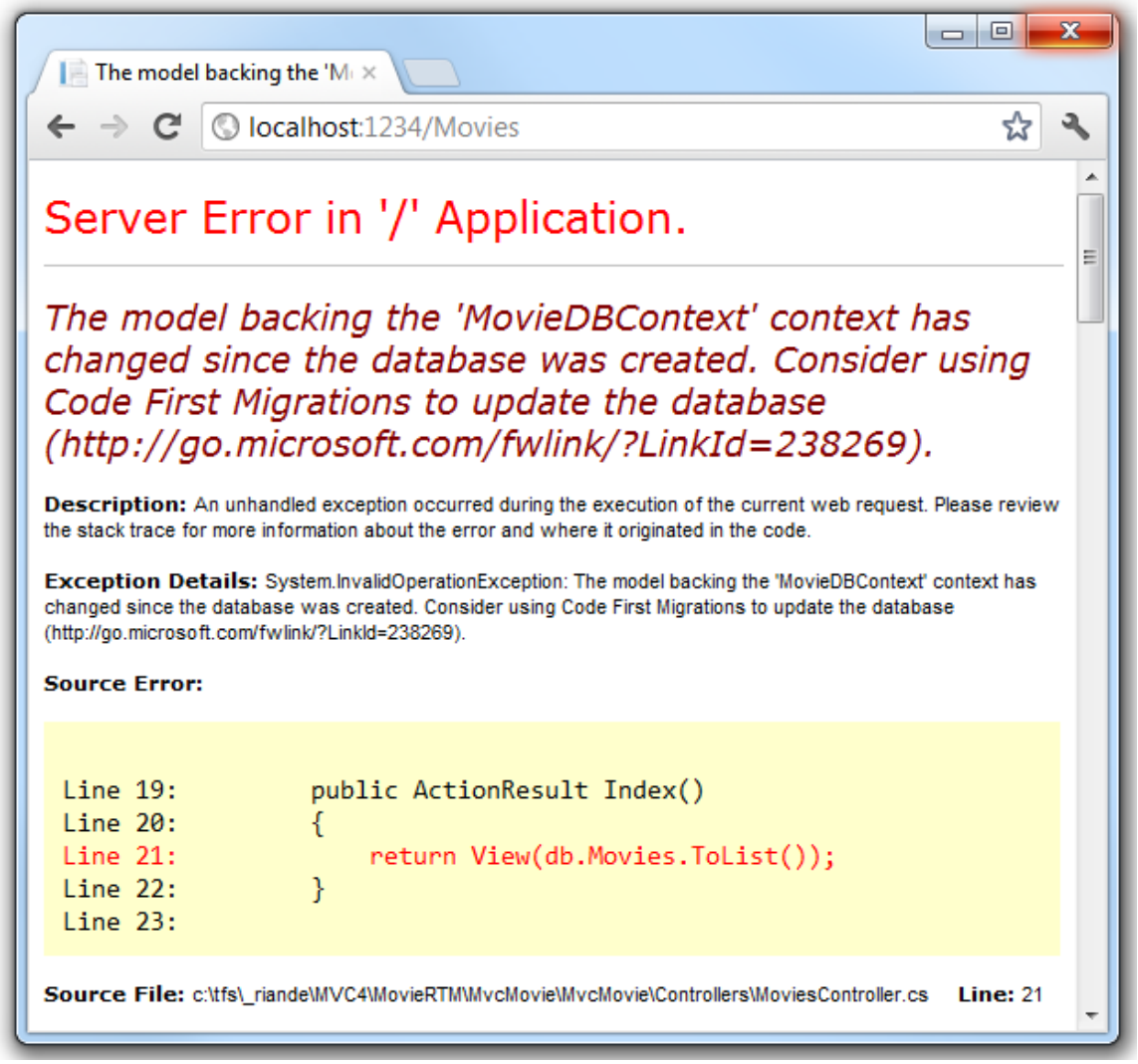
下一步，打开*\Views\Movies\Create.cshtml* 文件，并在 form 标签结束处的附近添加如下代码。您可以在创建新的电影时指定一个电影等级。

```
<div class="editor-label">
    @Html.LabelFor(model => model.Rating)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Rating)
    @Html.ValidationMessageFor(model => model.Rating)
</div>
```

现在，您已经更新应用程序代码以支持了新的 `Rating` 属性。

现在运行该应用程序，然后浏览 */Movies* 的 URL。然而，当您这样做时，您将看到以下之一的错误信息：





你现在看到此错误，因为在应用程序中，最新的 `Movie` 模型类和现有的数据库 `Movie` 表的 Schema 不同。（数据库表中，没有 `Rating` 列。）

我们将使用 Code First Migrations 来解决这一问题。

更新 Seed 方法，以便它能为新的列提供一个值。打开 Migrations\Configuration.cs 文件，并将 Rating 字段添加到影片的每个对象。

```
new Movie
        {
            Title = "When Harry Met Sally",
            ReleaseDate = DateTime.Parse("1989-1-11"),
            Genre = "Romantic Comedy",
```

```
            Rating = "G",
            Price = 7.99M
        },
```

Build 解决方案，然后打开 **软件包管理器控制台** 窗口，并输入以下命令：

```
add-migration AddRatingMig
```

`add-migration` 命令告诉 migration framework，来检查当前电影模型与当前的影片 DB Schema 并创建必要的代码以将数据库迁移到新的模型。AddRatingMig 是一个任意的文件名参数，用于命名 migration 文件。它将有助于使得迁移步骤成为一个有意义的名字。

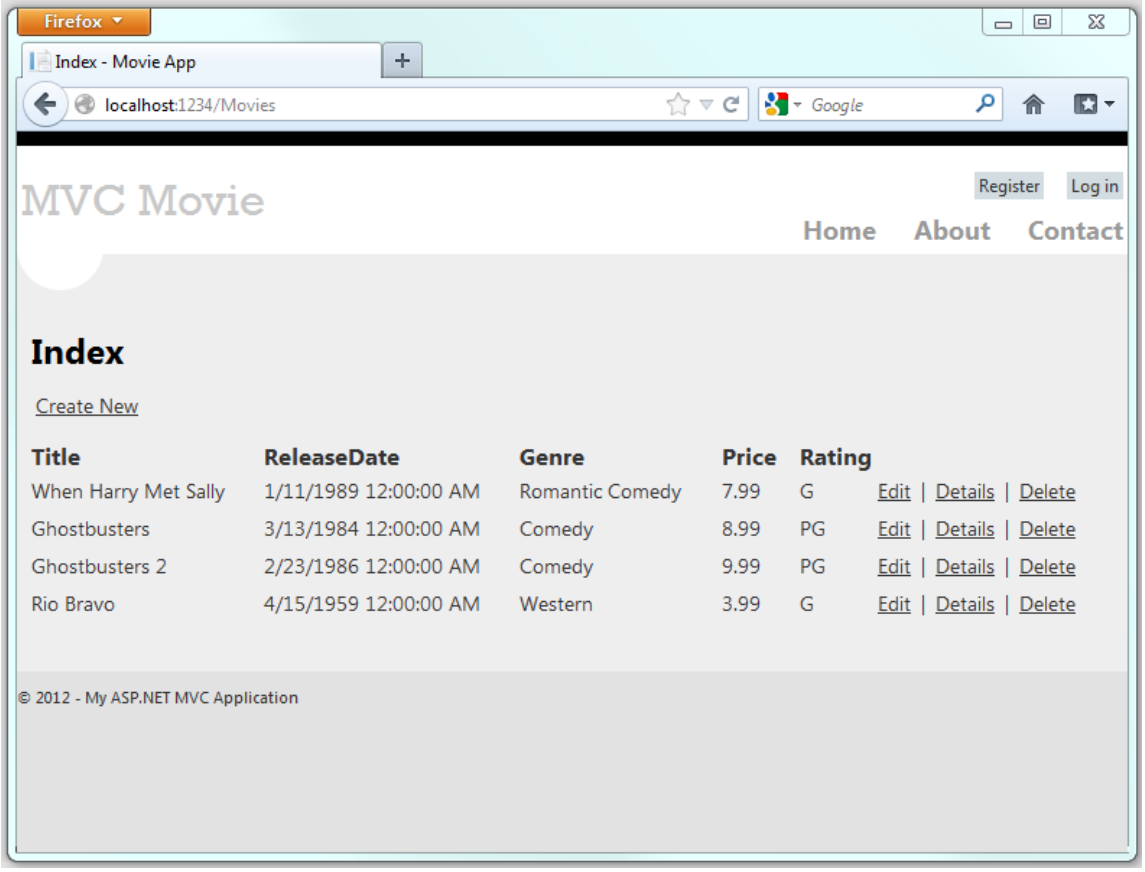
当命令完成后，用 Visual Studio 打开类文件，新继承自 `DbMIgration` 类的定义，并在 `Up` 方法中，您可以看到创建新列的代码：

```
public partial class AddRatingMig : DbMigration
{
    public override void Up()
    {
        AddColumn("dbo.Movies", "Rating", c => c.String());
    }

    public override void Down()
    {
        DropColumn("dbo.Movies", "Rating");
    }
}
```

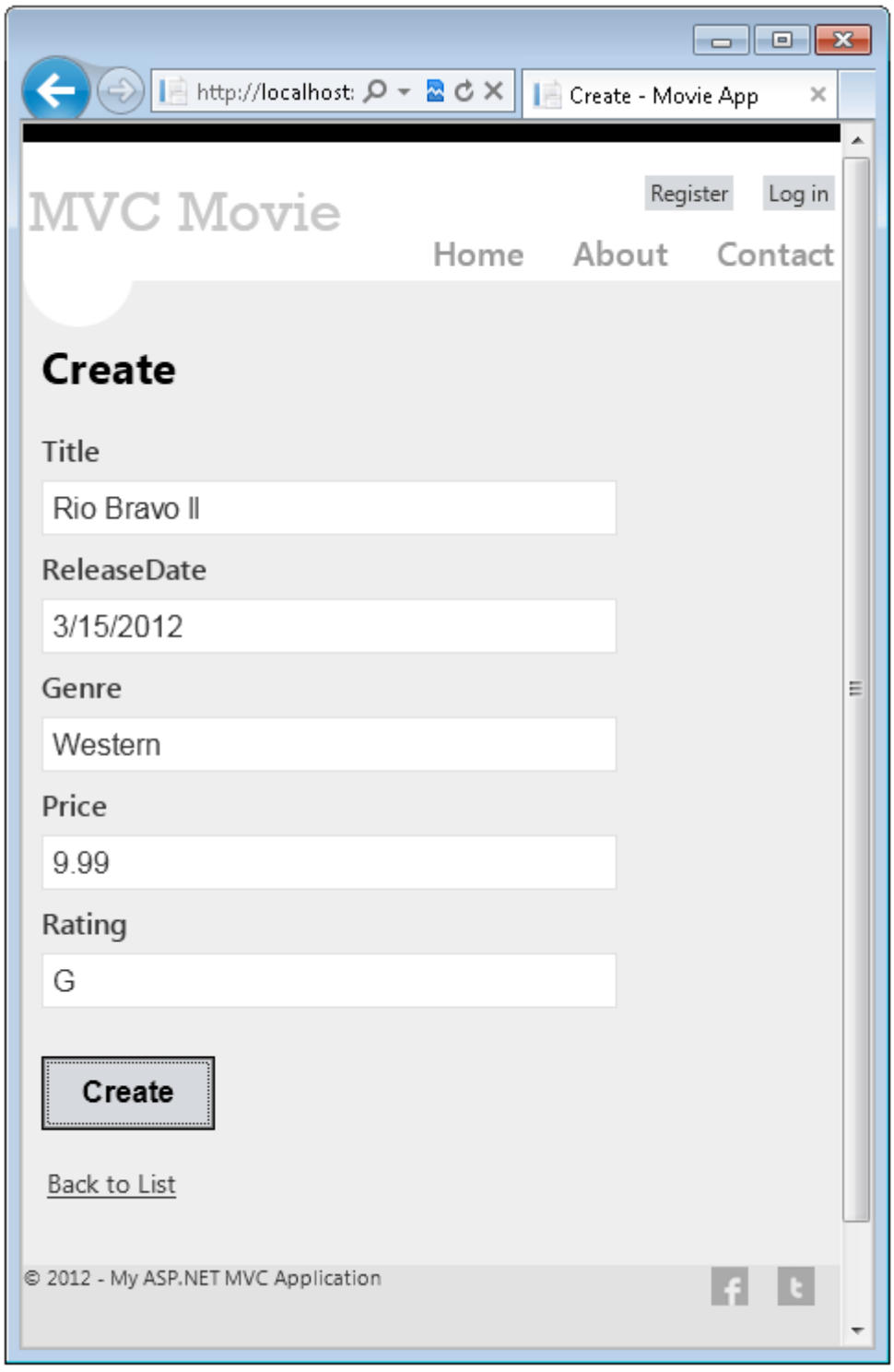
Build 解决方案，然后在 **程序包管理器控制台** 窗口中输入"update-database"命令。
下面的图片显示了 **程序包管理器控制台** 窗口的输出（AddRatingMig 的前缀时间戳将有所不同）。

```
PM> update-database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying code-based migrations: [201207260218501_AddRatingMig].
Applying code-based migration: 201207260218501_AddRatingMig.
Running Seed method.
PM>
```

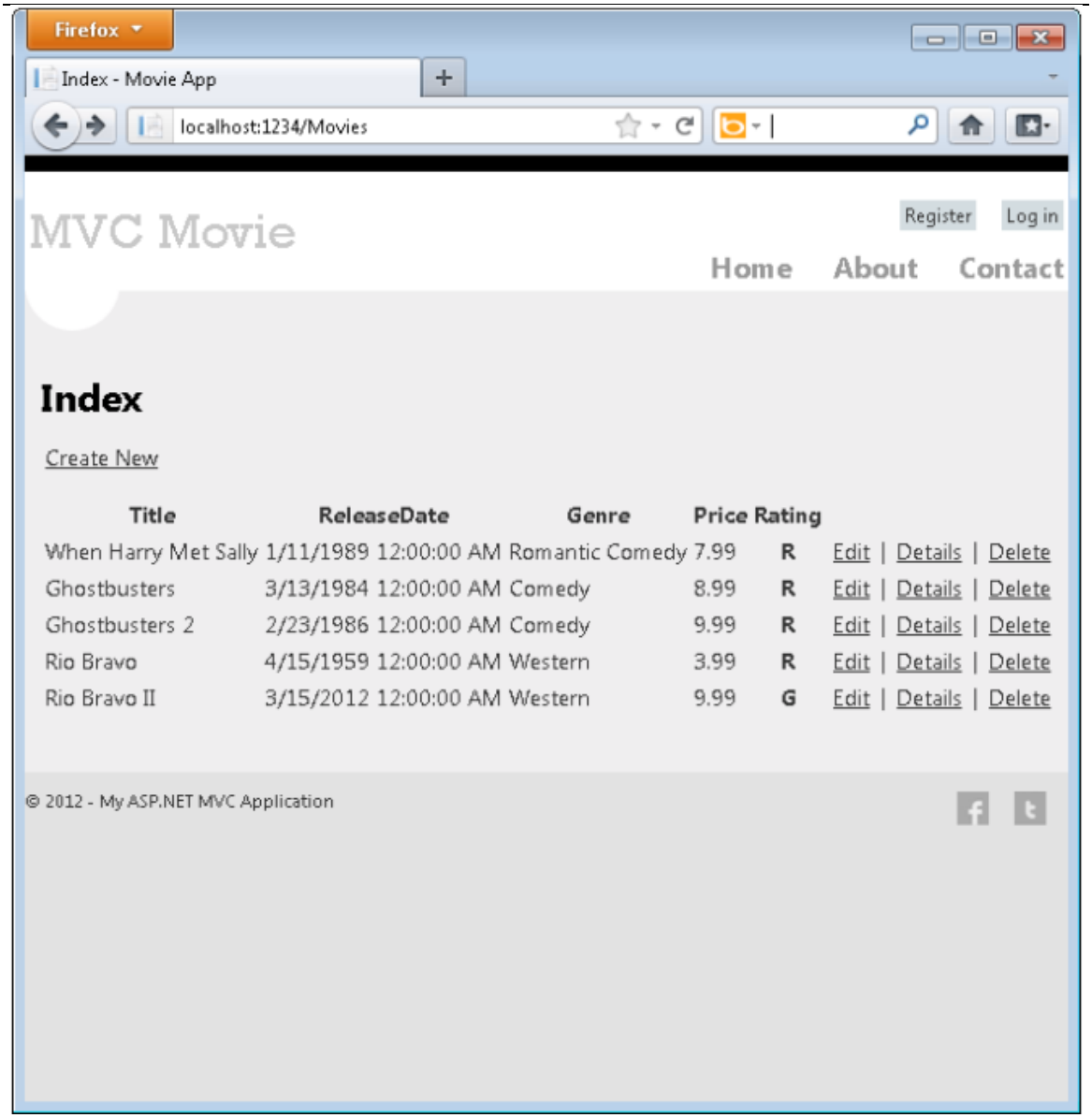
重新运行应用程序，然后浏览 /Movies 的 URL。您可以看到新的评级字段。



单击 **CreateNew** 链接来添加一部新电影。注意，请您可以为电影添加评级。



单击 Create。新的电影，包括评级，将显示在电影列表中：



此外您也应该把 `Rating` 字段添加到编辑、 详细信息和 SearchIndex 的视图模板中。

您可以再次在 **程序包管理器控制台** 窗口中输入"update-database"命令，将不会有任何新的变化，因为数据库 Schema 和模型类现在是匹配的。

在本节中，您看到了如何修改模型对象并始终保持其和数据库 Schema 的同步。您还学习了使用填充示例数据来创建新数据库的例子，您可以反复尝试。接下来，让我们看看如何将丰富的验证逻辑添加到模型类，并对模型类执行一些强制的业务规则验证。

给数据模型添加校验器

在本节中将会给 `Movie` 模型添加验证逻辑。并且确保这些验证规则在用户创建或编辑电影时被执行。

保持事情 DRY

ASP.NET MVC 的核心设计信条之一是 DRY: "不要重复自己（Don't Repeat Yourself）"。ASP.NET MVC 鼓励您指定功能或者行为，只做一次，然后将它应用到应用程序的各个地方。这可以减少您需要编写的代码量，并减少代码出错率，易于代码维护。

给 ASP.NET MVC 和 Entity Framework Code First 提供验证支持是 DRY 信条的一次伟大实践。您可以在一个地方（模型类）中以声明的方式指定验证规则，这个规则会在应用程序中的任何地方执行。

让我们看看您如何在本电影应用程序中，使用此验证支持。

给电影模型添加验证规则

您将首先向 `Movie` 类添加一些验证逻辑。

打开 *Movie.cs* 文件。在文件的顶部添加 `using` 语句，从而引用 `System.ComponentModel.DataAnnotations` 命名空间：

```
using System.ComponentModel.DataAnnotations;
```

注意，该命名空间不包含 `System.Web`。DataAnnotations 提供了一组内置的验证特性，您可以以声明的方式，应用于任何类或属性。

更新 Movie 类，以利用内置的 Required、 StringLength 和 Range 验证属性。以下面的代码为例，以应用验证属性。

```
public class Movie {
    public int ID { get; set; }

    [Required]
    public string Title { get; set; }

    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Required]
    public string Genre { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [StringLength(5)]
    public string Rating { get; set; }
}
```

运行该应用程序，您会再次得到了以下的运行时错误：

The model backing the 'MovieDBContext' context has changed since the database was created. Consider using Code First Migrations to update the database (http://go.microsoft.com/fwlink/?LinkId=238269).

我们将使用 Migrations 来更新 Schema。生成解决方案，然后打开**软件包管理器控制台**窗口，并输入以下命令：

```
add-migration AddDataAnnotationsMig
update-database
```

当此命令完后，Visual Studio 会打开指定名称（*AddDataAnnotationsMig*）的文件，其中定义了派生自 DbMIgration 的新类，并在 Up 方法中，您可以看到代码更新的 Schema 和约束条件。Title 和 Genre 字段不再可以为 null（即，您必须输入一个值）并且 Rating 字段具有最大长度是 5。

验证属性将指定一个验证行为，这样您可以指定模型中的那个属性需要被强制验证。Required 属性指示该属性必须有一个值 ，在此示例中，一部电影必须要有 Title 、 ReleaseDate、Genre 和 Price 属性的值，这样才有效。Range 属性限制了一个指定范围内的值。StringLength 属性允许您设置一个字符串属性的最大长度和其最小长度（可选）。内部类型 （例如 decimal, int, float, DateTime ）默认是必须的，所以不需要 Required 属性。

Code First 确保您在模型类上所指定的验证规则，会在应用程序修改数据库之前执行。例如，下面的代码在调用 SaveChanges 方法时，将引发异常，因为缺失几个必需的 Movie 属性值，并且价格为零 （这在有效范围之外）。

```
MovieDBContext db = new MovieDBContext();

Movie movie = new Movie();
movie.Title = "Gone with the Wind";
movie.Price = 0.0M;

db.Movies.Add(movie);
db.SaveChanges();        // <= Will throw server side validation exception
```

验证规则会自动被 .NET Framework 执行，这将有助于使您的应用程序更加的可靠。它还确保你不会因为忘了验证，无意中使得坏的数据也写入到了数据库。

下面是更新后的 Movie.cs 文件的完整代码清单：

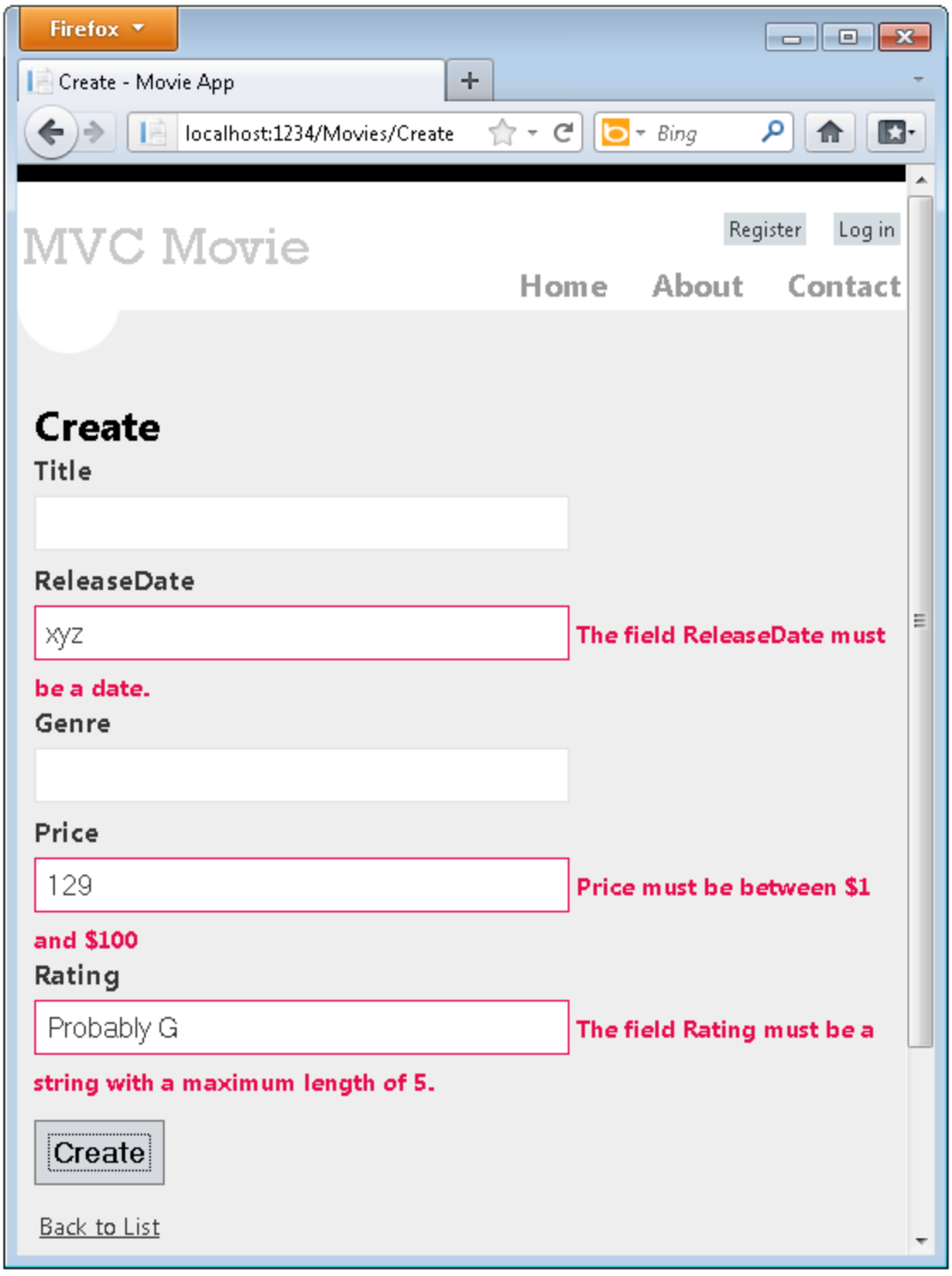
```
using System;
using System.Data.Entity;
using System.ComponentModel.DataAnnotations;

namespace MvcMovie.Models {
    public class Movie {
        public int ID { get; set; }

        [Required]
        public string Title { get; set; }
```

```
        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }

        [Required]
        public string Genre { get; set; }

        [Range(1, 100)]
        [DataType(DataType.Currency)]
        public decimal Price { get; set; }

        [StringLength(5)]
        public string Rating { get; set; }
    }

    public class MovieDBContext : DbContext {
        public DbSet<Movie> Movies { get; set; }
    }
}
```

ASP.NET MVC 的验证错误 UI

重新运行应用程序，浏览 */Movies* 的 URL。

单击 **Create New** 链接，来添加一部新电影。在窗体中填写一些无效值，然后单击 **Create** 按钮。



注意，为了使 jQuery 支持使用逗号的非英语区域的验证 ，需要设置逗号（","）来表示小数点，你需要引入 *globalize.js* 并且你还需要具体的指定 *cultures/globalize.cultures.js* 文件（地址在 https://github.com/jquery/globalize) 在 JavaScript 中可以使用 `Globalize.parseFloat`。下面的代码展示了在"FR-FR" Culture 下的 Views\Movies\Edit.cshtml 视图：

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
    <script src="~/Scripts/globalize.js"></script>
    <script src="~/Scripts/globalize.culture.fr-FR.js"></script>
    <script>
        $.validator.methods.number = function (value, element) {
            return this.optional(element) ||
                !isNaN(Globalize.parseFloat(value));
        }
        $(document).ready(function () {
            Globalize.culture('fr-FR');
        });
    </script>
    <script>
        jQuery.extend(jQuery.validator.methods, {
            range: function (value, element, param) {
                //Use the Globalization plugin to parse the value
                var val = $.global.parseFloat(value);
                return this.optional(element) || (
                    val >= param[0] && val <= param[1]);
            }
        });

    </script>
}
```

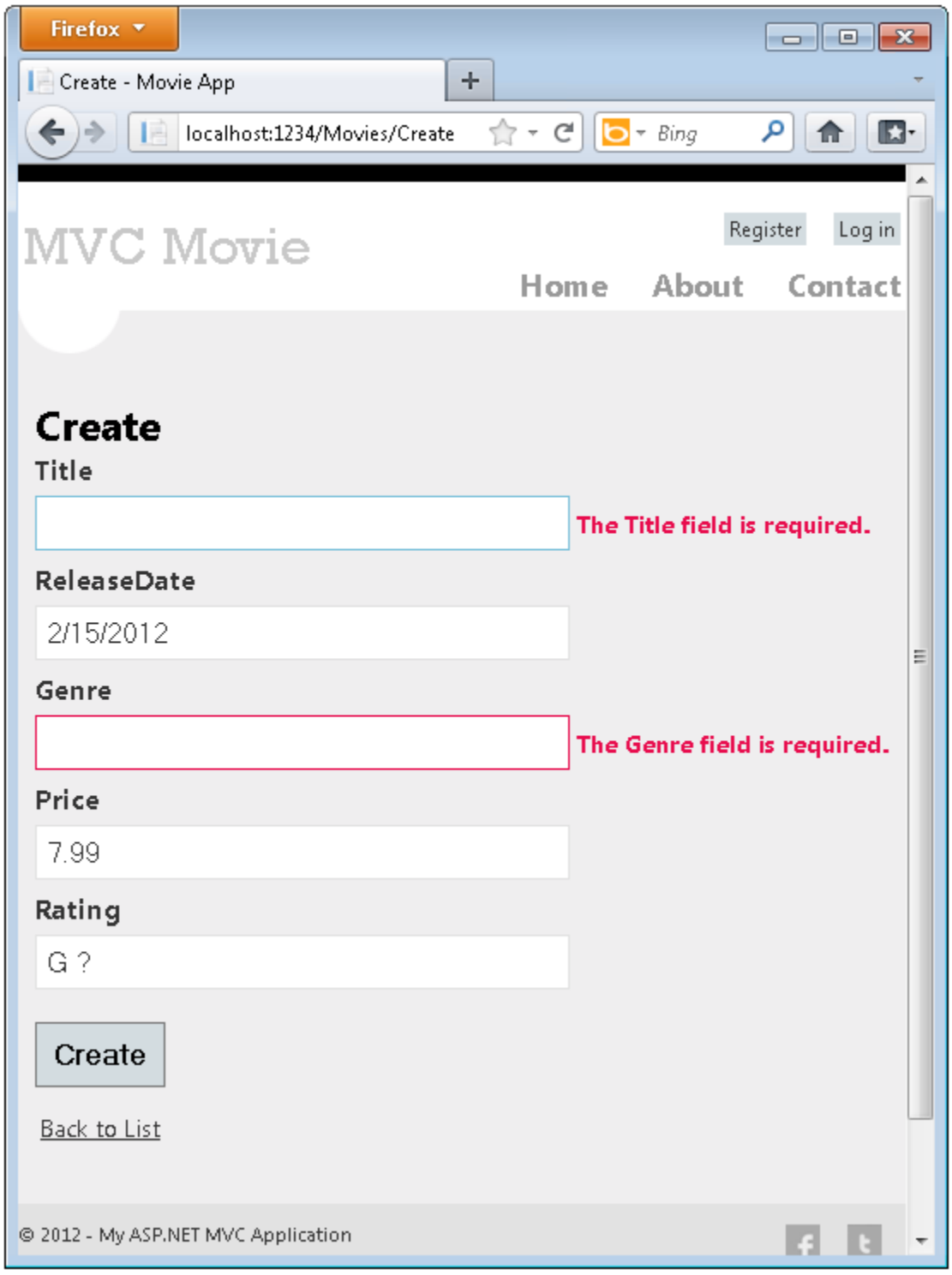
为了使用这种用户验证界面，真正的好处是，您不需要修改 `MoviesController` 类或 *Create.cshtml* 视图中的任何一行代码。在本教程之前所生成的控制器和视图中，`Movie` 模型类的属性上所指定的验证规则一样可以自动适用。

您可能已经注意到了 `Title` 和 `Genre` 属性，在字段中输入文本或者删除文本，是不会执行所需的验证属性的，直到您提交表单 (点 **Create** 按钮)时才执行。对于字段是最初为空（如创建视图中的字段）和只有 Required 属性并没有其它验证属性的字段，您可以执行以下操作来触发验证：

1. Tab into the field.
2. Enter some text.
3. Tab out.
4. Tab back into the field.

5. Remove the text.
6. Tab out.

上面的顺序将触发必需的验证，而并不需要点击提交按钮。在不输入任何字段的情况下，直接点击提交按钮，将触发客户端验证。直到没有客户端验证错误的情况下，表单数据才会发送到服务器。您可以在服务器端 HTTP Post 方法上加上断点来测试一下，或者使用 Fiddler tool 或 IE 9 F12 Developer tools.


Firefox
Create - Movie App
localhost:1234/Movies/Create
Bing
MVC Movie
Register
Log in
Home
About
Contact
Create
Title
The Title field is required.
ReleaseDate
2/15/2012
Genre
The Genre field is required.
Price
7.99
Rating
G ?
Create
Back to List
© 2012 - My ASP.NET MVC Application

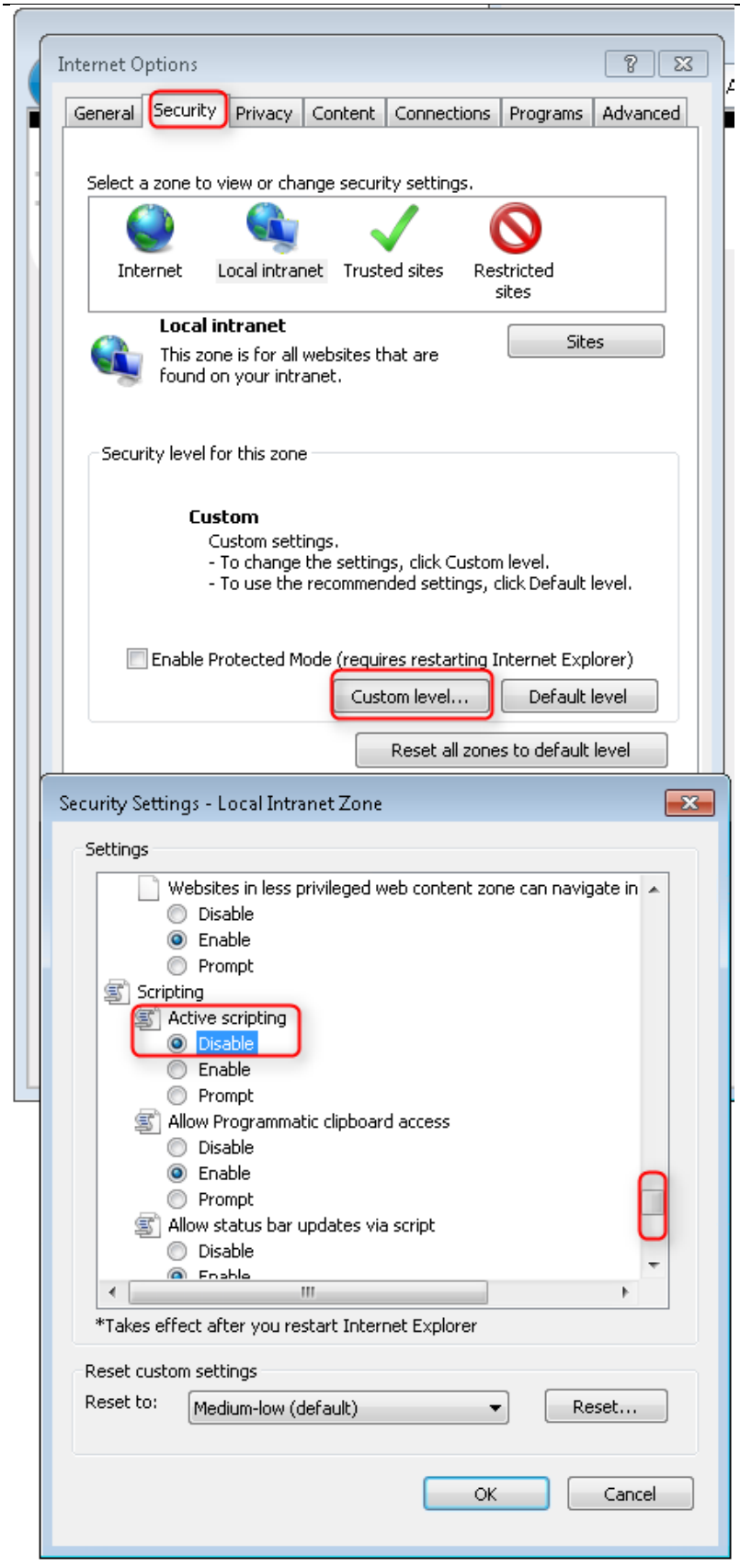
如何验证创建视图和创建方法

您可能很想知道验证用户界面在没有更新控制器或视图代码的情况下是如何生成的。下面列出了 MovieController 类中的 Create 方法。它们是之前教程中自动生成的，并没有修改。

```
//
// GET: /Movies/Create

public ActionResult Create()
{
    return View();
}

//
// POST: /Movies/Create

[HttpPost]
public ActionResult Create(Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Movies.Add(movie);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(movie);
}
```

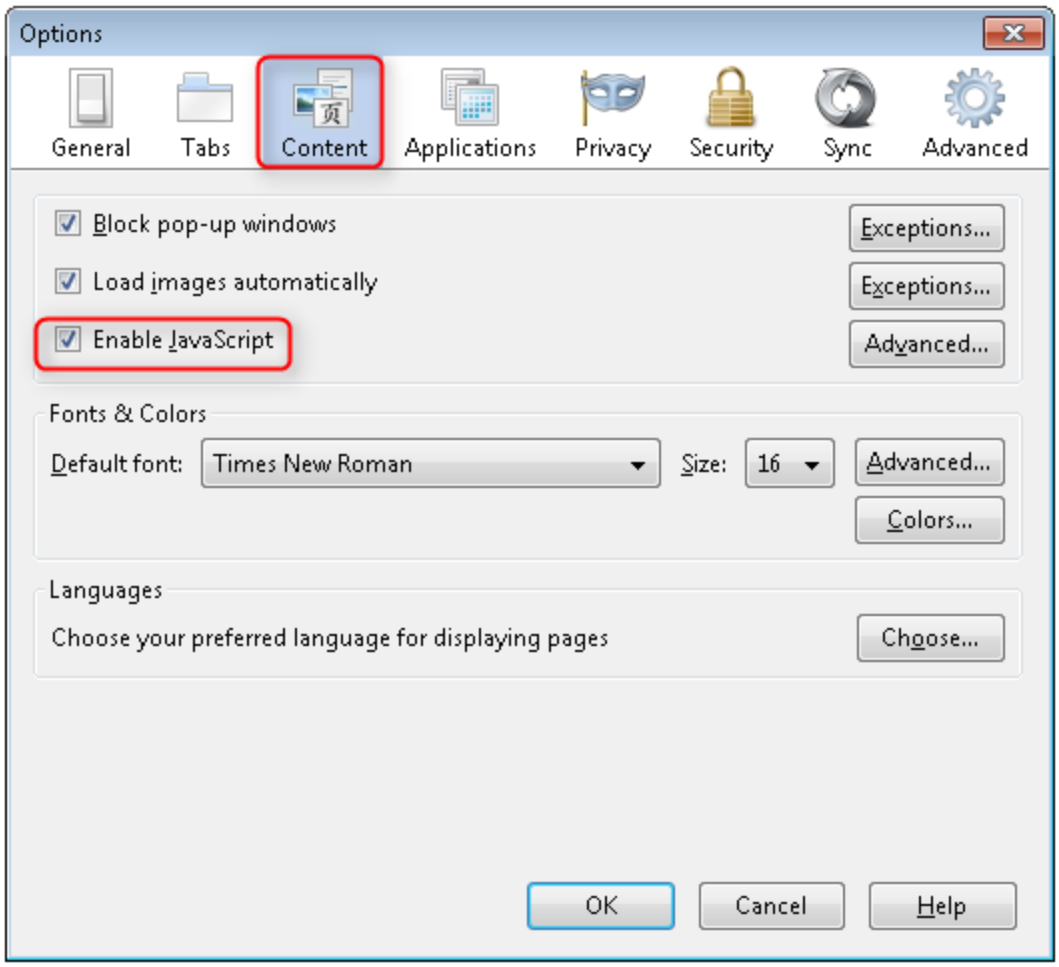
第一种（HTTP GET）Create 方法用来显示初始的创建 form。第二个（[HttpPost]）方法处理 form 的请求。第二种 Create 方法（HttpPost 版本）调用 ModelState.IsValid 来检查是否有的任何的 Movie 验证错误。调用此方法将验证对象上所有应用了验证约束的属性。如果对象含有验证错误，则 Create 方法会重新显示初始的 form。如果没有任何错误，方法将保存信息到数据库。在我们的电影示例中，我们使用了验证，**当客户端检测到错误时，form 不会被 post 到服务器；所以第二个 Create 方法永远不会被调用**。如果您在浏览器中禁用了 JavaScript，客户端验证也会被禁用，HTTP POST Create 方法会调用 ModelState.IsValid 来检查影片是否含有任何验证错误。

您可以在 `HttpPost Create` 方法中设置一个断点，当客户端验证检测到错误时，不会 post form 数据，所以永远不会调用该方法。如果您在浏览器中禁用 JavaScript，然后提交具有错误信息的 form，断点将会命中。您仍然得到充分的验证，即使在没有 JavaScript 的情况下。下图显示了如何禁用 Internet Explorer 中的 JavaScript。

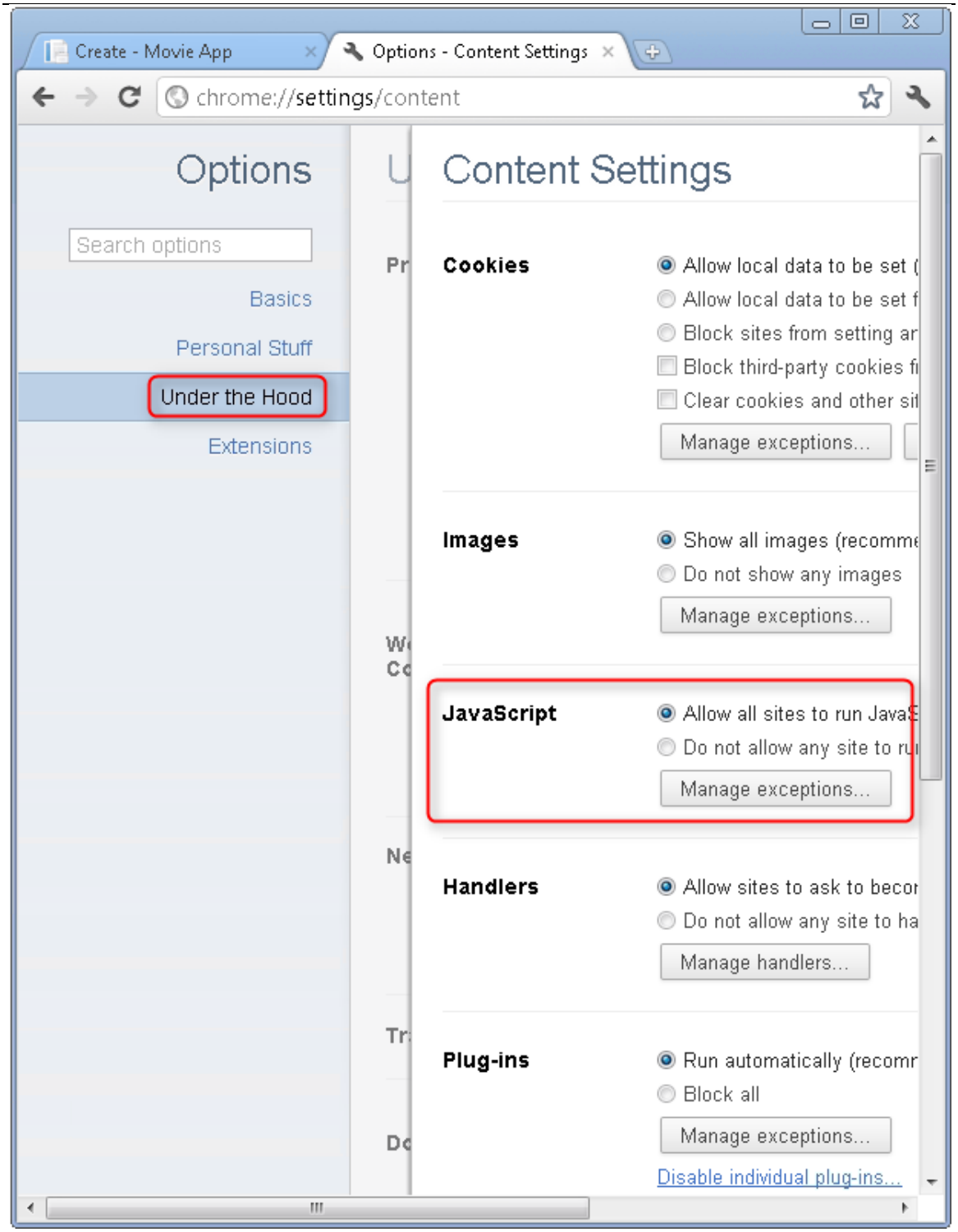

Internet Options
General
Security
Privacy
Content
Connections
Programs
Advanced
Select a zone to view or change security settings.
Internet
Local intranet
Trusted sites
Restricted sites
Local intranet
This zone is for all websites that are found on your intranet.
Sites
Security level for this zone
Custom
Custom settings.
- To change the settings, click Custom level.
- To use the recommended settings, click Default level.
Enable Protected Mode (requires restarting Internet Explorer)
Custom level...
Default level
Reset all zones to default level
Security Settings - Local Intranet Zone
Settings
Websites in less privileged web content zone can navigate in
Disable
Enable
Prompt
Scripting
Active scripting
Disable
Enable
Prompt
Allow Programmatic clipboard access
Disable
Enable
Prompt
Allow status bar updates via script
Disable
*Takes effect after you restart Internet Explorer
Reset custom settings
Reset to:
Medium-low (default)
Reset...
OK
Cancel

```
[HttpPost]
public ActionResult Create(Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Movies.Add(movie);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(movie);
}
```

下图显示了如何在火狐浏览器中禁用 JavaScript。



下图显示了如何在 Chrome 浏览器中禁用 JavaScript。



下面是框架代码在之前的教程中生成的 *Create.cshtml* 视图模板。它用来为以上两个操作方法来显示初始的 form，同时在验证出错时来重新显示视图。

```
@model MvcMovie.Models.Movie

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")"></script>
<script
src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"></script>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>Movie</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Title)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Title)
            @Html.ValidationMessageFor(model => model.Title)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.ReleaseDate)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.ReleaseDate)
            @Html.ValidationMessageFor(model => model.ReleaseDate)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Genre)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Genre)
            @Html.ValidationMessageFor(model => model.Genre)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Price)
        </div>
        <div class="editor-field">
```

```
            @Html.EditorFor(model => model.Price)
            @Html.ValidationMessageFor(model => model.Price)
        </div>
        <div class="editor-label">
    @Html.LabelFor(model => model.Rating)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Rating)
    @Html.ValidationMessageFor(model => model.Rating)
</div>
        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

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

请注意，代码如何使用 `Html.EditorFor` helper 输出为 `Movie` 中的每个属性的`<input>`元素。此 Helper 旁边是对 `Html.ValidationMessageFor` 方法的调用。这两个 Helper 方法将处理由控制器传递到视图的模型对象（在这里是，`Movie` 对象）。它们会自动查找模型中指定的验证属性，并显示适当的错误消息。

如果您想要在后面更改验证逻辑，您可以做在一个地方，将验证信息添加到模型上。（此示例中，是 `movie` 类）。您不必担心不符合规则 ，验证逻辑会在应用程序的不同部分执行——在一个地方定义验证逻辑将会被使用到各个地方。这使代码非常干净，并使它易于维护和扩展。它意味着您会完全遵守 DRY 原则。

给影片模型添加 Formatting

打开 *Movie.cs* 文件并查看 `Movie` 类。 `System.ComponentModel.DataAnnotations` 命名空间提供了内置的验证特性集的格式属性。我们已经为发布日期和价格字段应用了 `DataType` 枚举值。下面的代码示例了 `ReleaseDate` 和 `Price` 属性与相应的 `DisplayFormat` 属性。

```
        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }
```

```
        [DataType(DataType.Currency)]
        public decimal Price { get; set; }
```

DataType 属性不是验证特性，它们用来告诉视图引擎如何 Render HTML 。在上面的示例中，DataType.Date 属性将影片日期显示为日期，例如，下面的 DataType 属性不会验证数据的格式：

```
[DataType(DataType.EmailAddress)]
[DataType(DataType.PhoneNumber)]
[DataType(DataType.Url)]
```

上面列出的属性只提供视图引擎来显示数据的格式（如：<a> 为 URL ，< href="mailto:EmailAddress.com"> 为电子邮件。您可以使用正则表达式属性来验证数据的格式。 ）

另一种使用 DataType 属性的方式，您可以显式设置 DataFormatString。下面的代码示例了具有一个日期格式字符串的 Release Date 属性 （即"d")。

```
        [DisplayFormat(DataFormatString = "{0:d}")]
        public DateTime ReleaseDate { get; set; }
```

下面的代码设置 Price 属性为货币格式。

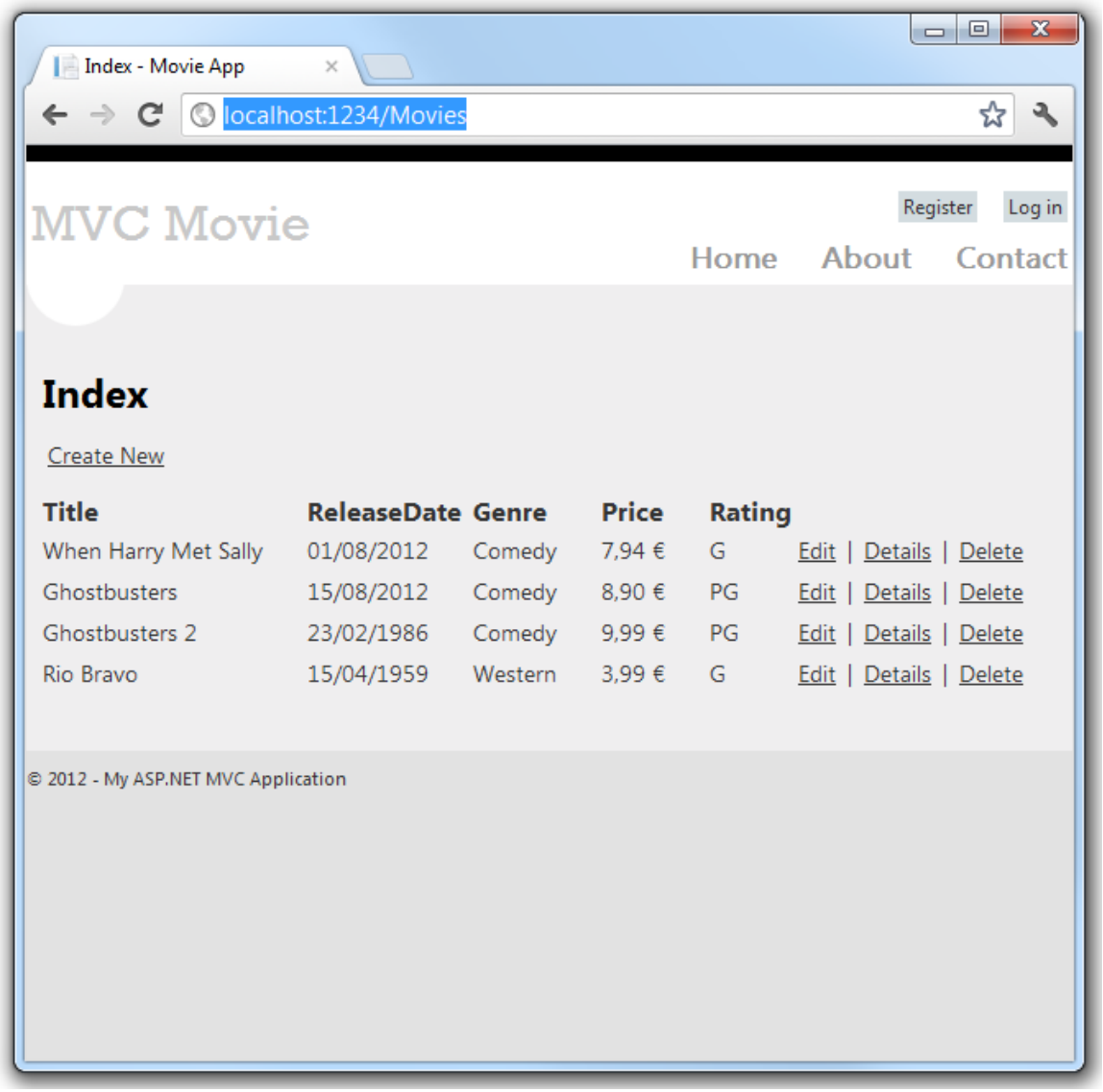
```
        [DisplayFormat(DataFormatString = "{0:c}")]
        public decimal Price { get; set; }
```

完整的 Movie 类如下所示。

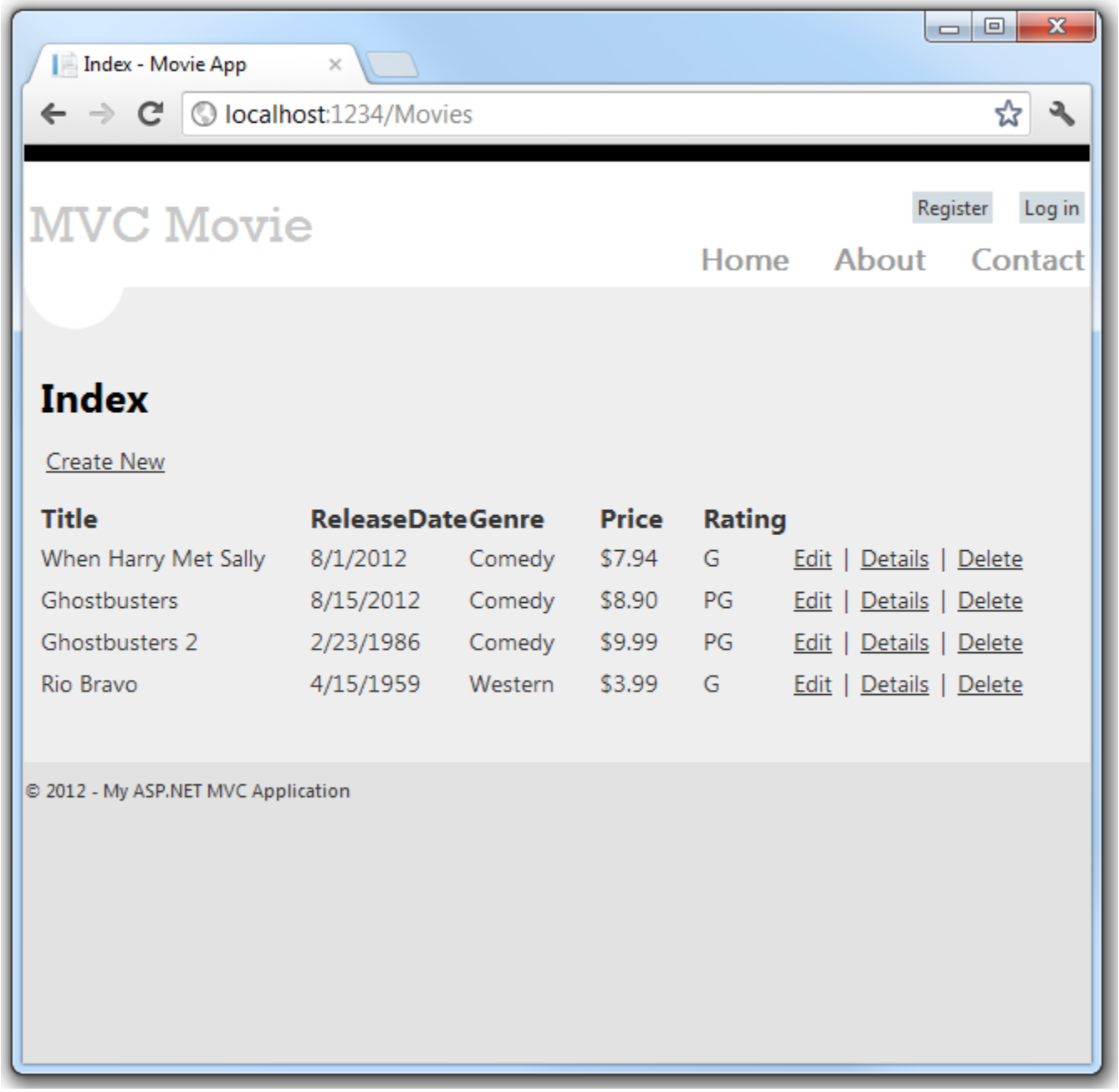
```
public class Movie {
    public int ID { get; set; }

    [Required]
    public string Title { get; set; }

    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }
```

```
    [Required]
    public string Genre { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [StringLength(5)]
    public string Rating { get; set; }
}
```

运行该应用程序并浏览到 `Movies` 控制器。很好的格式化了发布日期和价格。下图显示了 Release Date 和使用 "FR-FR" Culture 的 Price。



下图为默认 Culture 的显示（English US）。



在下一部分，我们先会看看代码，然后再改进一下自动生成的 `Details` 和 `Delete` 方法。

查询详细信息和删除记录

打开 Movie 控制器并查看 Details 方法。

```
public ActionResult Details(int id = 0)
{
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
```

Code First 使得您可以轻松的使用 `Find` 方法来搜索数据。一个重要的安全功能内置到了方法中。方法首先验证 `Find` 方法已经找到了一部电影，然后再执行其它代码。例如，黑客可以通过更改 *http://localhost:xxxx/Movies/Details/1* 到 *http://localhost:xxxx/Movies/Details/12345* （或某些其它值，不代表实际影片的值）从而使得链接 URL 出现错误。如果您没有检测是否找到了 Movie， null Movie 会导致出现数据错误。

查看 `Delete` 和 `DeleteConfirmed` 方法。

```
 // GET: /Movies/Delete/5

public ActionResult Delete(int id = 0)
{
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}

//
// POST: /Movies/Delete/5

[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id = 0)
```

```
{
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    db.Movies.Remove(movie);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

请注意，Delete 的 HTTP Get 方法不会删除指定的电影，它返回删除电影的视图，您可以在此视图中提交（HttpPost）删除电影。如果使用 GET 请求执行删除操作（或者执行编辑操作，创建操作或者更改数据的任何其它操作）开辟了一个安全漏洞。对此的详细信息，请参阅斯蒂芬·瓦尔特的博客 ASP.NET MVC Tip #46 — Don't use Delete Links because they create Security Holes.

将删除数据的 HttpPost 方法命名为唯一签名或名称的 DeleteConfirmed 方法。这两个方法的签名如下所示：

```
// GET: /Movies/Delete/5
public ActionResult Delete(int id = 0)

//
// POST: /Movies/Delete/5
[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id = 0)
```

公共语言运行时 (CLR)重载方法时，需要方法具有独特唯一的签名（方法名称相同但不同的参数列表）。但是，在这里您需要两种删除方法 — — 一个 GET 方法和一个 POST 方法它们都具有相同的签名。（他们都需要接受一个整数作为参数）。

要解决这一点，可以有几种办法。一是使用不同的方法名称。这是框架代码在前面的示例中所使用的方法。然而，这就带来了一个小问题： ASP.NET 将部分的 URL 按名称映射到操作方法，如果您重命名了方法，通常 Routing 将无法找到该方法。解决方法是您在示例中看到的，将 ActionName("Delete")属性添加到 DeleteConfirmed 方法。这会有效的执

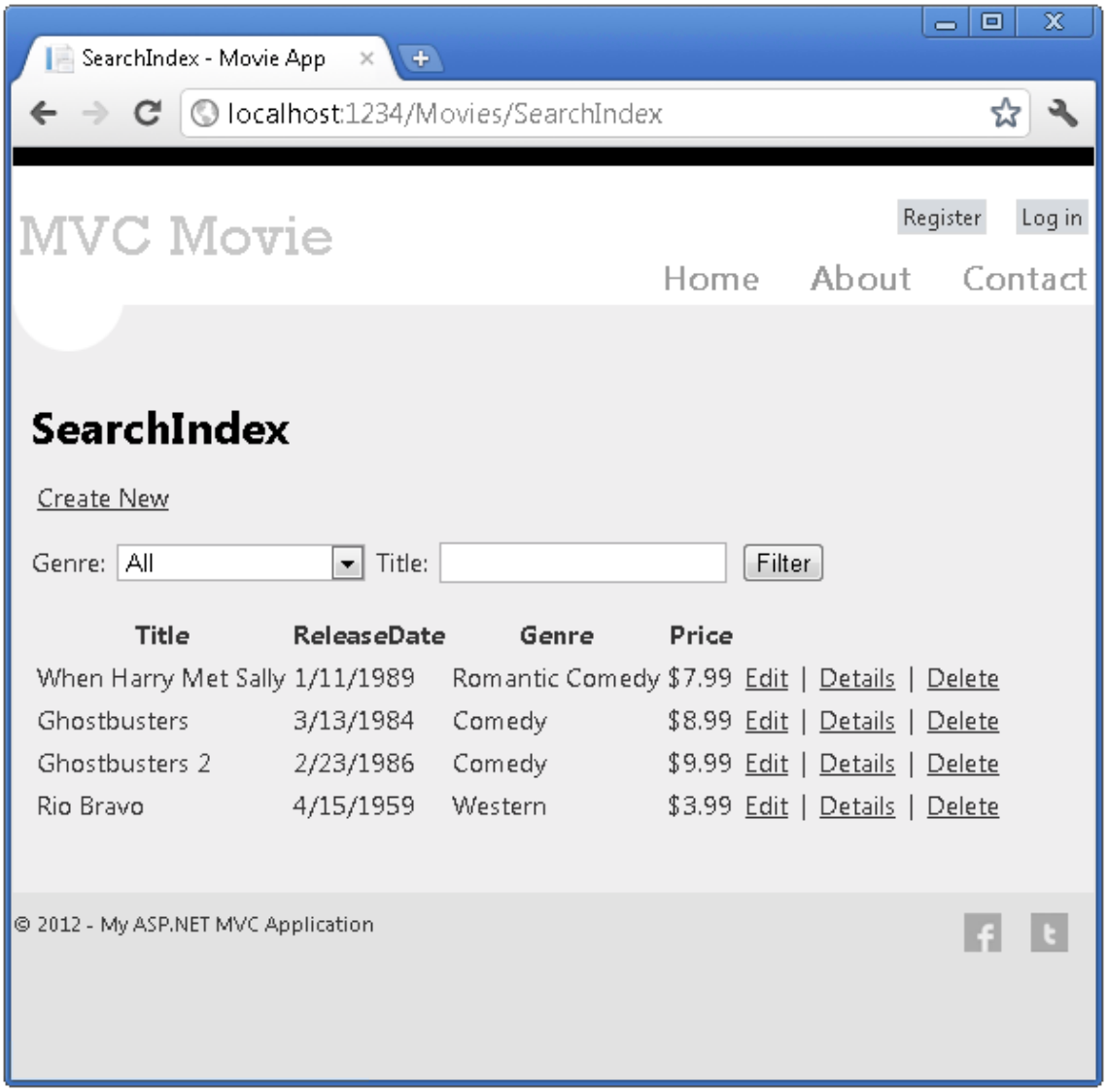
行 Routing 系统的 Url 映射，这样一个包含*/Delete/*的 POST 请求的 URL 将找到 `DeleteConfirmed` 方法。

另一个常见的方法，来避免具有相同名称和签名的方法，是人为地改变 POST 方法，包括未使用参数的签名。例如，有些开发人员添加参数类型 `FormCollection`，`FormCollection` 是会传递给 POST 方法的，然后根本不使用此参数：

```
public ActionResult Delete(FormCollection fcNotUsed, int id = 0)
{
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    db.Movies.Remove(movie);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

小结

您现在有一个完整的 ASP.NET MVC 应用程序并在本地的 DB 数据库中存储数据。您可以创建、 读取、 更新、 删除和搜索电影。



如果您想要部署应用程序，最好先在您本地的 IIS 7 服务器上测试一下您的应用程序。您可以使用此 Web Platform Installer 链接启用 IIS 服务器的 ASP.NET 应用程序的设置。请参阅下面的部署链接：

- Test your ASP.NET MVC or WebForms Application on IIS 7 in 30 seconds
- ASP.NET Deployment Content Map

- Enabling IIS 7.x
- Web Application Projects Deployment

现在鼓励您开始学习中级内容 Creating an Entity Framework Data Model for an ASP.NET MVC Application 和 MVC Music Store 教程, 浏览 ASP.NET articles on MSDN,的文章，再看看很多的视频和资源：http://asp.net/mvc 来了解更多关于 ASP.NET MVC 的信息 ！ ASP.NET MVC forums 论坛是一个好地方，可以用来问您想要知道的问题。

第三方控件 ComponentOne Studio for ASP.NET Wijmo 在 MVC4 下的应用

ComponentOne Studio for ASP.NET Wijmo 最新版本 2013V1 支持 MVC4，其中包括：

- **新增 MVC 4 工程模板 (C# & VB)** 开箱即用的 MVC 4 工程模板基于 Microsoft 内置模板创建，我们仅优化了标记和 CSS 样式为 Wijmo 风格，熟悉的模板布局和界面风格，无疑将缩短您的学习过程、节省开发时间及提高开发效率。
- **新增国际化主题（Metro）**
    - **MVC4 模板自动增强 Wijmo** MVC Scaffolding 模板，将会为您应用程序中的增删改查(CRUD)操作生成默认的模板文件，这些生成的文件为您的工程构建了起始的工程文件目录结构，当然你也可以修改它，Scaffolding 模板的优美之处在于生成后您可以按照您的意愿来扩展它。
    - **Wijmo-增强编辑器模板** 该模板使您可以通过日期选择器、数值输入框和滑动条快速的定制应用。您甚至可以添加其他自定义的模板。

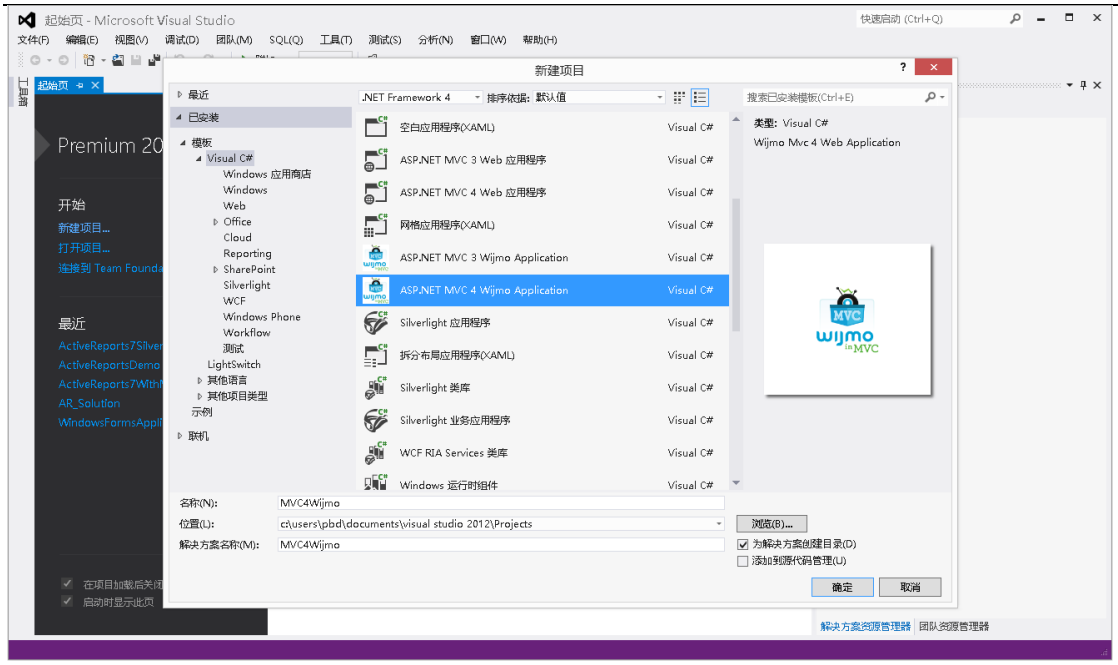
开始使用

使用 ComponentOne Studio for ASP.NET Wijmo 制作 MVC4 应用程序，首先要做的是安装 Studio for ASP.NET Wijmo 。

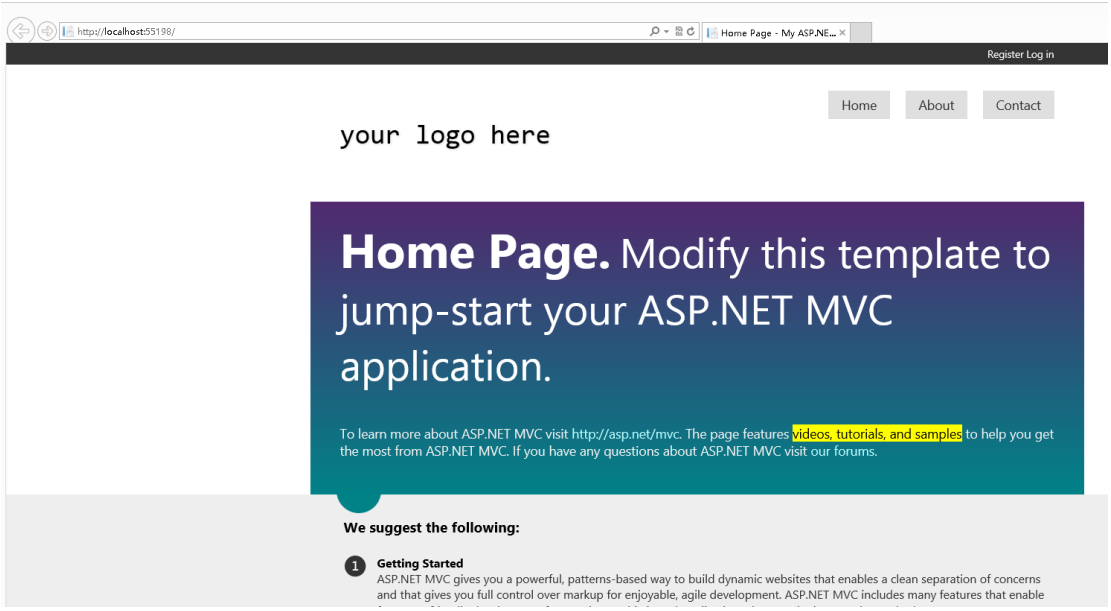
测试环境 VS2012、 MVC4、 Framework4.5、 IE10、 Studio for ASP.NET Wijmo2013V1

文件-新建项目

在安装了 Studio for ASP.NET Wijmo2013V1 之后，在 VS2012 中选择新建项目。在 Web 选项卡中，您可以发现 Studio for ASP.NET Wijmo2013V1。



好了，现在让我们运行程序看看初始效果。您可能对这个界面很熟悉。因为 Wijmo MVC 4 工程模板是基于 Microsoft 内置模板创建。我们仅优化了标记和 CSS 样式为 Wijmo 风格。



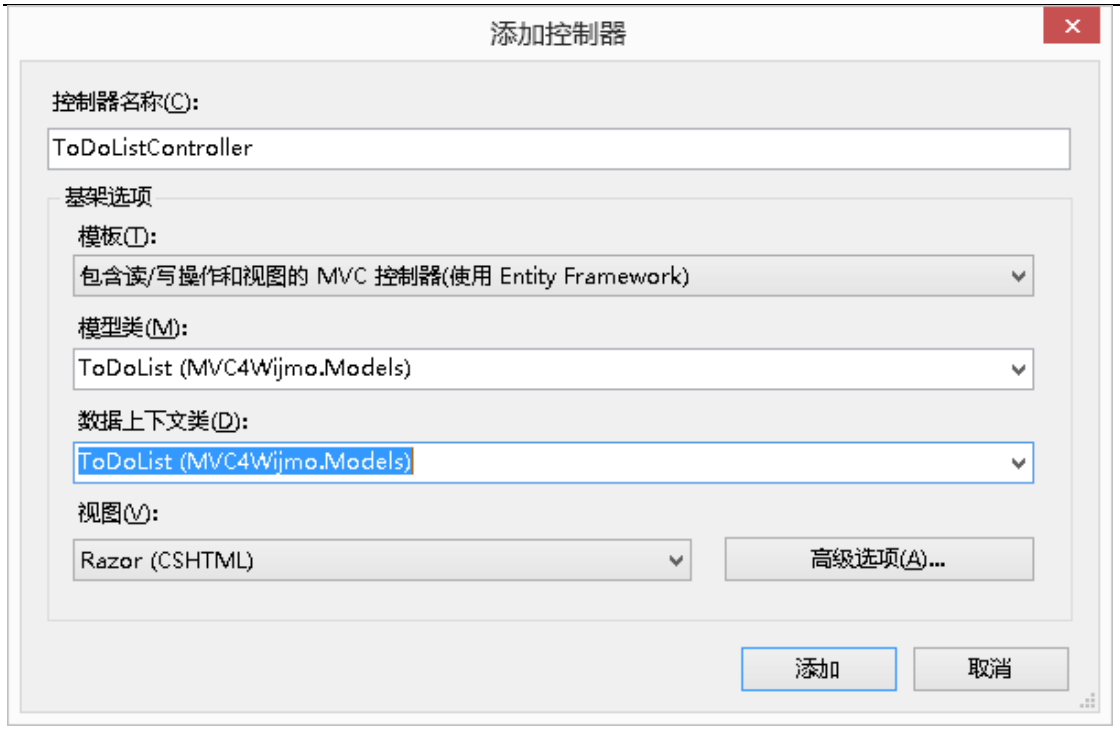
添加模型

下面，让我们使用 Wijmo MVC Scaffolding 模板创建一个简易的“ToDoList”。首先我们来添加模型。需要添加以下代码：

```
namespace MVC4Wijmo.Models
{
   public class ToDoList
   {
      [Editable(false)]
      public int Id { get; set; }
      [Required]
      public string Title { get; set; }
      [Display(Name = "Date Created")]
      public DateTime? CreatedAt { get; set; }
      [Range(0, 5), UIHint("IntSlider")]
      public int Priority { get; set; }
      [Range(0, 1000000)]
      public decimal Cost { get; set; }
      [DataType(DataType.MultilineText)]
      public string Summary { get; set; }
      public bool Done { get; set; }
      [Display(Name = "Date Completed")]
      public DateTime? DoneAt { get; set; }
      public ICollection<TahDoItem> TahDoItems { get; set; }
   }

   public class TahDoItem
   {
      [Editable(false)]
      public int Id { get; set; }
      [Required]
      public string Title { get; set; }
      [Display(Name = "Date Created")]
      public DateTime? CreatedAt { get; set; }
      [Range(0, 5), UIHint("IntSlider")]
      public int Priority { get; set; }
      [DataType(DataType.MultilineText)]
      public string Note { get; set; }
      public int ToDoListId { get; set; }
      public ToDoList ToDoList { get; set; }
      public bool Done { get; set; }
      [Display(Name = "Date Completed")]
      public DateTime? DoneAt { get; set; }
   }
}
```

创建控制器和视图

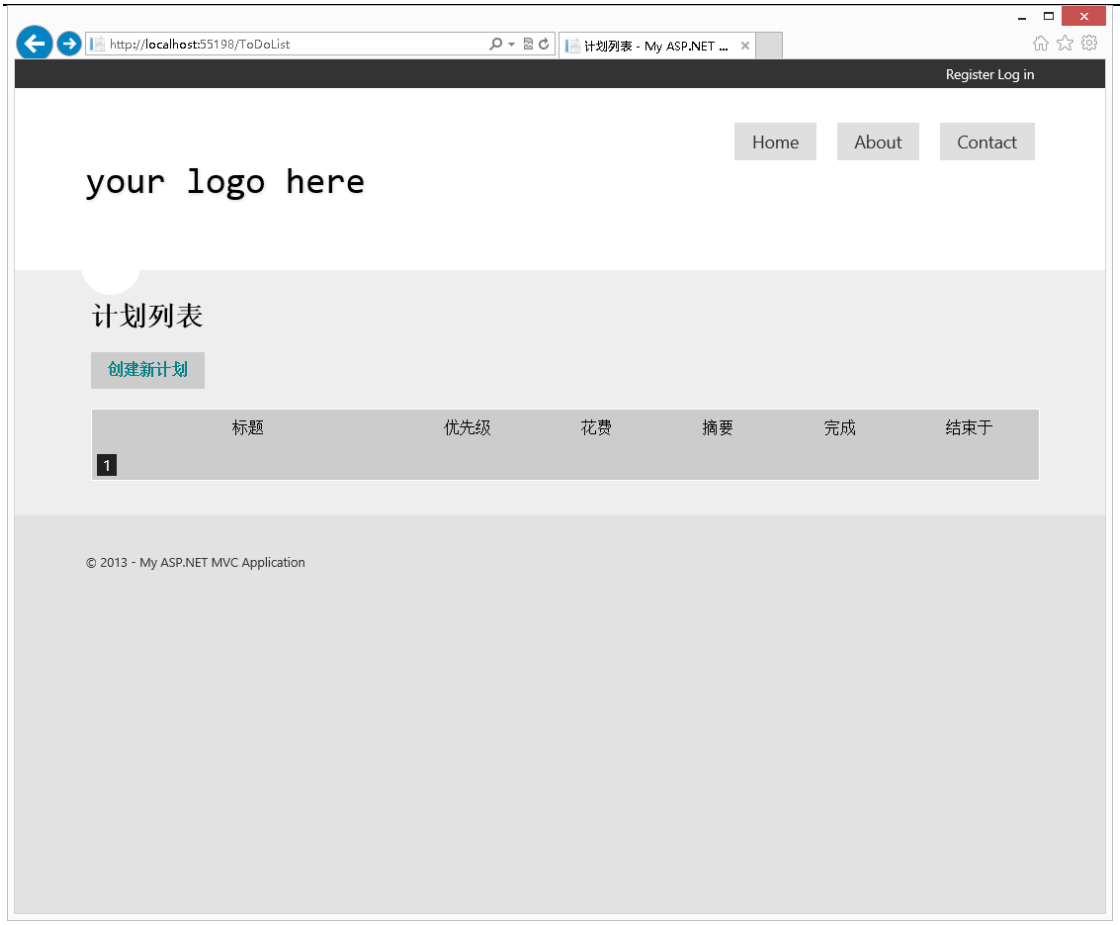
在添加控制器和视图之前，编译项目。这将使 Scaffolding 模板识别新增的模型。现在，邮件点击 Controllers 文件夹，选择“添加控制器”，选择一下选项点击“添加”。



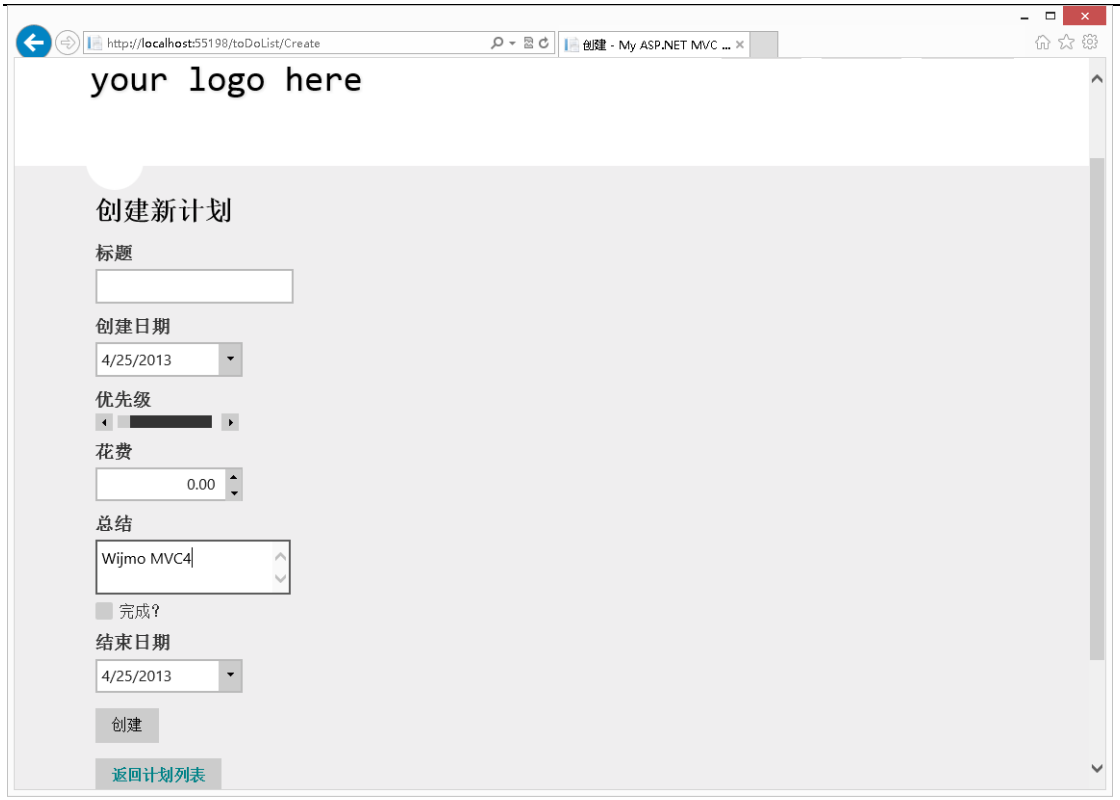
Scaffolding 将会自动生成控制器和增删改查应用程序所需要的所有视图。最大的亮点是这些生成的文件为您的工程构建了起始的工程文件目录结构，当然你也可以修改它，Scaffolding 模板的优美之处在于生成后您可以按照您的意愿来扩展它。

运行

仅仅通过以上步骤，我们就实现了简易的 ToDoList。切换到 ToDoList 页面，应用程序会给模型创建数据源，首先展示给我们的是一张空表格。我们可以通过“创建新计划”按钮添加计划。



在创建视图中您会发现展现在眼前的是标准的 EditorFor Helpers。然而我们已经在工程中添加了自定义编辑模板。所以如果使用日期或数值等类型时，Scaffolding 模板会自动生成编辑器。下面自定义编辑器视图截图：



现在我们就完成了具有增删改查功能的 MVC4 应用程序。这些生成的文件为您的工程构建了起始的工程文件目录结构，当然你也可以修改它，Scaffolding 模板的优美之处在于生成后您可以按照您的意愿来扩展它。

Demo 源码下载：TahDoMvc4.zip

工具下载链接：Studio for ASP.NET Wijmo