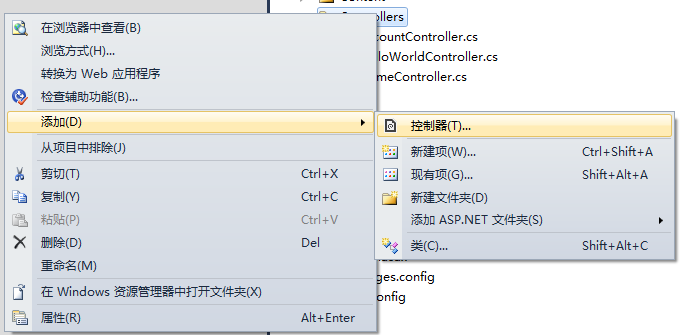
**[学习ASP.NET MVC(五)——我的第一个ASP.NET MVC CURD页面](http://www.cnblogs.com/chillsrc/p/3805067.html)**

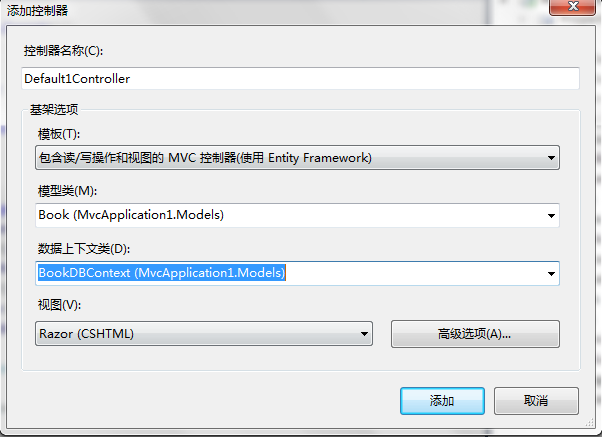
       在上一篇文章中我们已经创建了实体类，在这一篇文章中，我将创建一个新的控制器类——BookController，使用BookController对Books表中的数据进行CURD操作的方法，并使用视图模板在浏览器中显示所查询到的书籍数据信息。

**一、添加控制器类**

      在Visual Studio 的“解决方案资源管理器”中右键单击Controllers文件夹，并创建一个新的BookController控制器类。如下图。



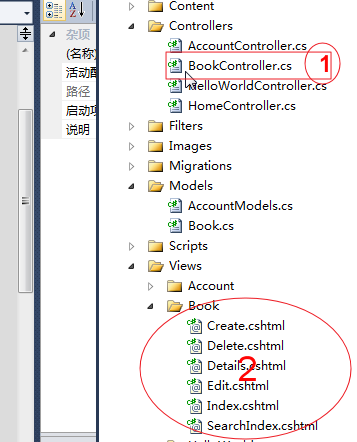
　　Visual Studio 会弹出一个“添加控制器”对话框（如下图）。

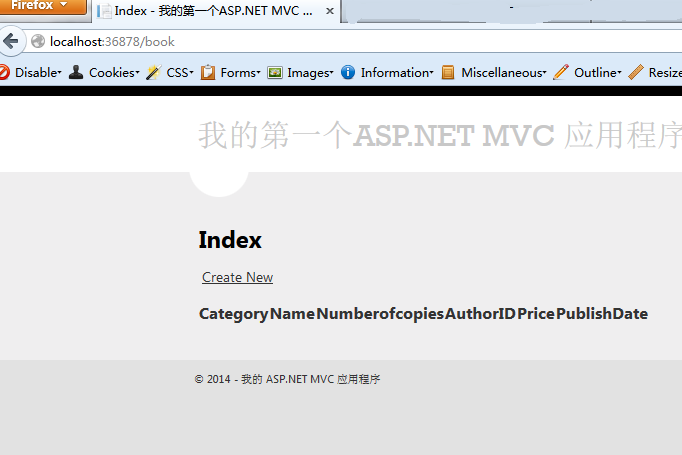


  在对话框中填写如下数据

•控制器名称：BookController。 （可以进行修改，如下图是默认的Default1Controller，修改为BookController。）   
•模板：包含读/写操作和视图MVC控制器（使用Entity Framework）。   
•模型类：Book（MvcApplication1.Models）。   
•数据上下文类：BookDBContext（MvcApplication1.Models）。   
•视图：Razor（CSHTML）。 （默认设置。）

 在填写完以上数据之后，用鼠标左键单击“添加”按钮。Visual Studio会快速创建下列文件和文件夹：  
1)在项目中的Controllers文件夹中创建一个BookController.cs文件。如下图中的1处。  
2)在项目的Views文件夹中创建一个Book文件夹。同时在Book文件夹中创建以下文件 Create.cshtml ， Delete.cshtml ， Details.cshtml ， Edit.cshtml和Index.cshtml。如下图中的2处。

  
3) ASP.NET MVC 4会自动创建对于数据库的CRUD（创建，读取，更新和删除）操作方法和视图。由Visual Studio自动创建CRUD操作方法和视图被称为基架。这样我们不需要修改什么，就有了一个全功能的Web应用程序，可以让你创建，查询，修改和删除书籍数据。  
4）在Visual Studio中运行该应用程序。，浏览器请求http://localhost:36878/Book被路由到书籍控制器的默认Index操作方法。换句话说，浏览器请求http://localhost: 36878/Book实际与请求http://localhost: 36878/Book/Index是一样的。浏览器中却是什么数据也没有显示，因为现在数据库中的Books表是一张空表，没有任何数据。如下图。



**二、创建书籍**  
1) 用鼠标左键，点击“Create New”链接，浏览器会导航到新建页面，在此页面中输入有关书籍的一些数据，然后点击“Create”按钮。如下图1，与图2.

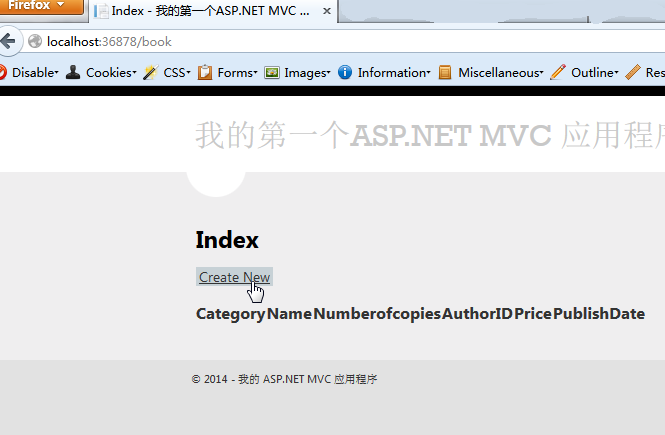


图1

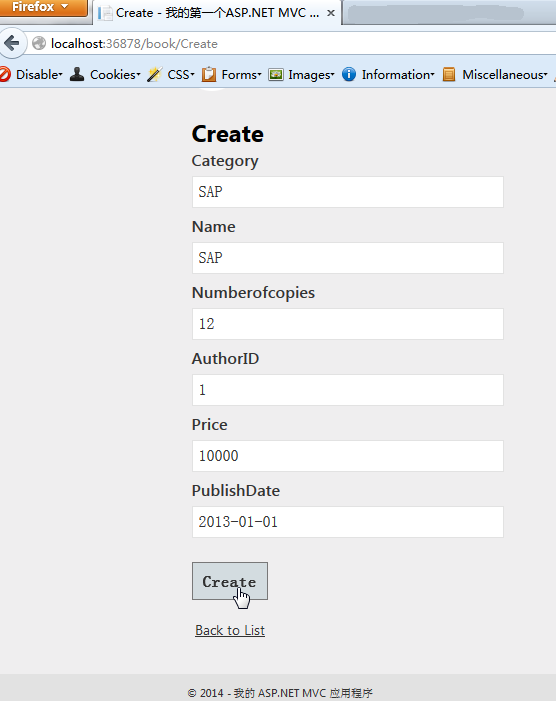
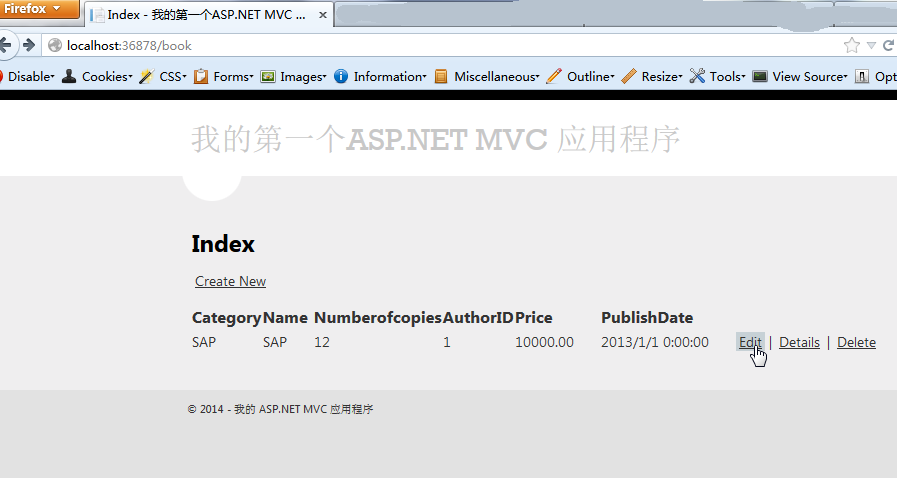


图2

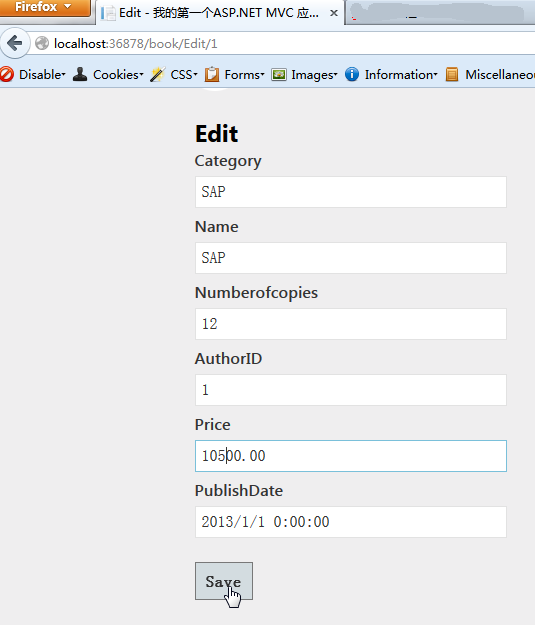
2) 当使用鼠标左键点击“Create”按钮时，浏览器会把表单数据发送到服务器上，服务器程序会把刚才填写的书籍信息保存到数据库中。然后浏览器会自动重定向到书籍列表的网址（<http://localhost:36878/book>），在那个页面就可以看到新创建的书籍数据，已经存在了。如下图。



**三、编辑书籍**

对刚才创建的书籍数据进行修改。在一条书籍记录的最后面有三个链接，分别是Edit、Details、Deltet。 如下图。

1) 用鼠标左键点击“Edit”链接。会浏览器会自动转到编辑界面。在编辑页面中修改完数据，点击“Save”按钮，即把修改的数据保存到数据库。如下图。



2) 2用鼠标左键点击“Details”链接。会浏览器会自动转到详细信息界面。如下图。图1，图2。

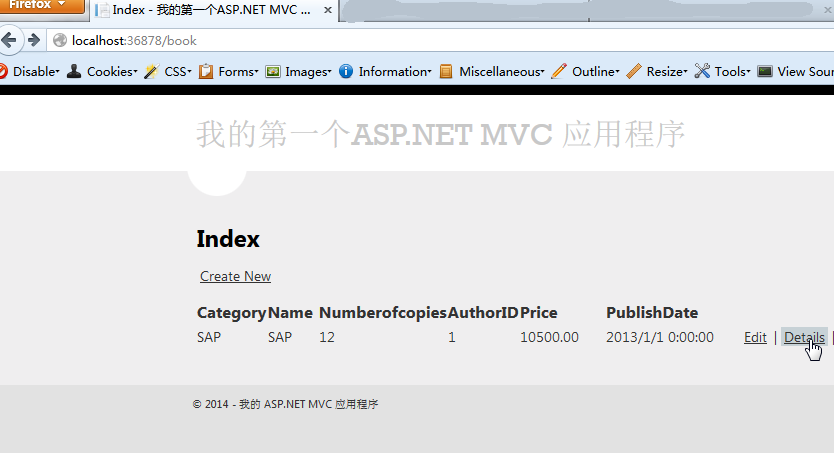


图1

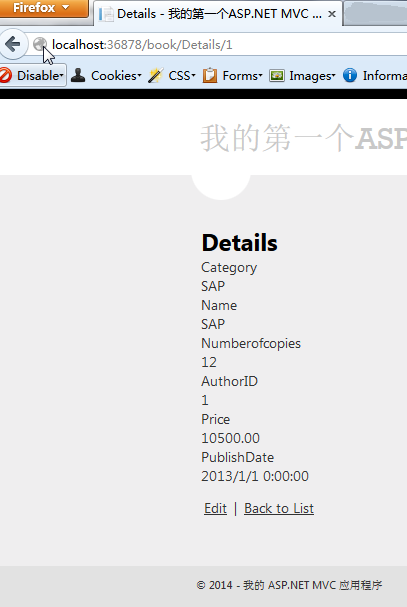


图2

**四、对ASP.NET MVC自动生成代码的解释**  
在Visual Studio的“解决方案资料管理器”中打开Controllers\ BookController.cs文件，查看由Visual Studio自动生成的Index方法。代码如下所示。

[复制代码](javascript:void(0);)

public class BookController : Controller

{

private BookDBContext db = new BookDBContext();

//

// GET: //

public ActionResult Index()

{

return View(db.Books.ToList());

}

}

[复制代码](javascript:void(0);)

注意这一行代码。

private BookDBContext db = new BookDBContext();

这行代码是实例化一个BookDBContext类对象。如前面所述。您可以使用数据库中的Books表中的数据进行查询，修改和删除。

当浏览器发出一个请求到Book控制器之后，由控制器中的默认方法（Index）返回了数据库中Books表中数据的所有行，并将结果传递给Index视图，然后呈现在浏览器，让用户看到。

**五、强类型对象和@model关键字**  
在本文的上面部分，讲了如何通过一个控制器传递数据或对象给使用ViewBag对象的视图模板。该ViewBag是一个动态对象，它提供了将信息传递到一个视图，在后期可以进行方便绑定的方式。  
ASP.NET MVC还提供了强类型数据或将对象实例传递给视图模板的能力。这种强类型的对象能够更方便Visual Studio在进行代码编译时进行代码检查，并在Visual Studio编辑器中有更丰富的智能感知。在Visual Studio的基架机制就是使用这种方法与BookController类和视图模板。  
    在Controllers\ BookController.cs文件中查看自动生成的Details方法。Details方法的代码如下所示。

[复制代码](javascript:void(0);)

public ActionResult Details(int id = 0)

{

Book book = db.Books.Find(id);

if (book == null)

{

return HttpNotFound();

}

return View(book);

}

[复制代码](javascript:void(0);)

如果想找到一本书籍，那么Book实体的一个实例会传递给Details视图。请查看Views\Book\Details.cshtml文件。   
    通过包括在视图模板文件的顶部@model语句，你可以指定视图期望得到的对象类型。当您创建BookController控制器类时，Visual Studio会自动包含在Details.cshtml文件的顶部以下@model的语句：

@model MvcApplication1.Models.Book

这个@ model指令允许你通过强类型访问控制器传递给视图的Model对象的Book类。例如，在Details.cshtml模板，代码通过每本书籍领域的DisplayNameFor和DisplayFor HTML辅助与强类型的Model对象。创建和编辑方法和视图模板也能通过书籍的模型对象进行访问。   
    查看Index.cshtml视图模板，并在BookController.cs文件找到Index方法。请注意代码是如何创建一个List对象时，使用View进行了封装，通过这个封装最终把书籍列表从控制器传给了视图。代码如下。

[复制代码](javascript:void(0);)

public ActionResult Index()

{

return View(db.Books.ToList());

}

[复制代码](javascript:void(0);)

当您创建BookController控制器时，Visual Studio会自动在Index.cshtml文件的顶部添加以下@model的语句：

@model IEnumerable<MvcApplication1.Models.Book>

这个@ model指令允许您访问控制器传递给视图通过使用的强类型的Model对象的书籍名单。例如，在Index.cshtml模板，如下面的代码在循环中通过强类型访问Model对象。代码如下。

[复制代码](javascript:void(0);)

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Category)

</td>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Numberofcopies)

</td>

<td>

@Html.DisplayFor(modelItem => item.AuthorID)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.PublishDate)

</td>

<td>

@Html.DisplayFor(model => item.Rating)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.BookID }) |

@Html.ActionLink("Details", "Details", new { id=item.BookID }) |

@Html.ActionLink("Delete", "Delete", new { id=item.BookID })

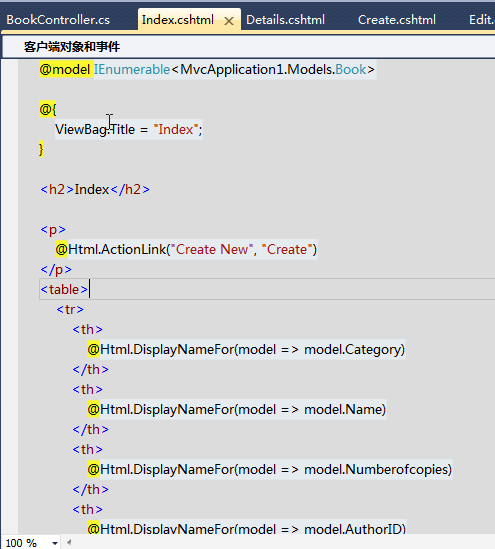
</td>

</tr>

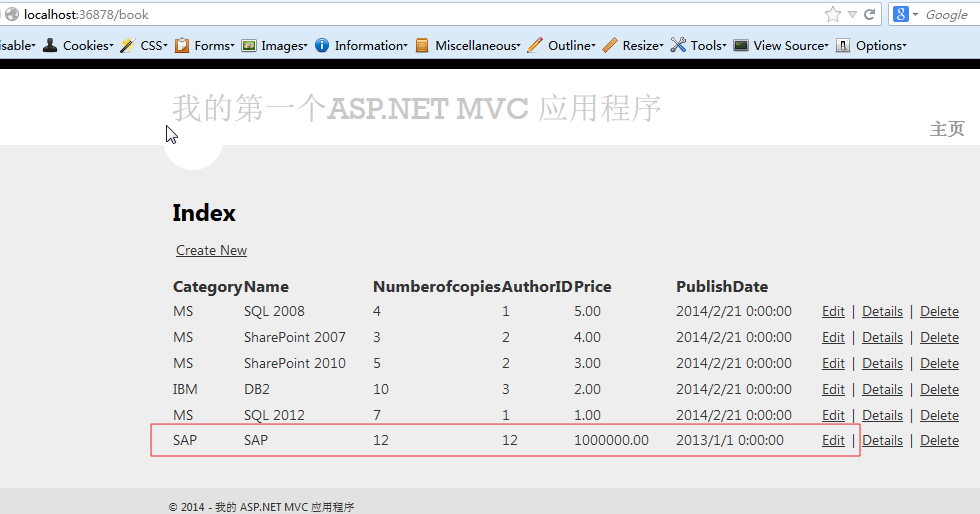
}

[复制代码](javascript:void(0);)

因为模型对象是强类型（如IEnumerable <Book>），则在循环中每个item对象的类型为Book。除了其他好处，这意味着你会得到编译时会进行代码检查功能和完整的IntelliSense支持。如下图。



运行程序，然后你会看到如下图的数据。



小记：最近工作有点忙，所以更新的有点迟了。尽量会写完的。