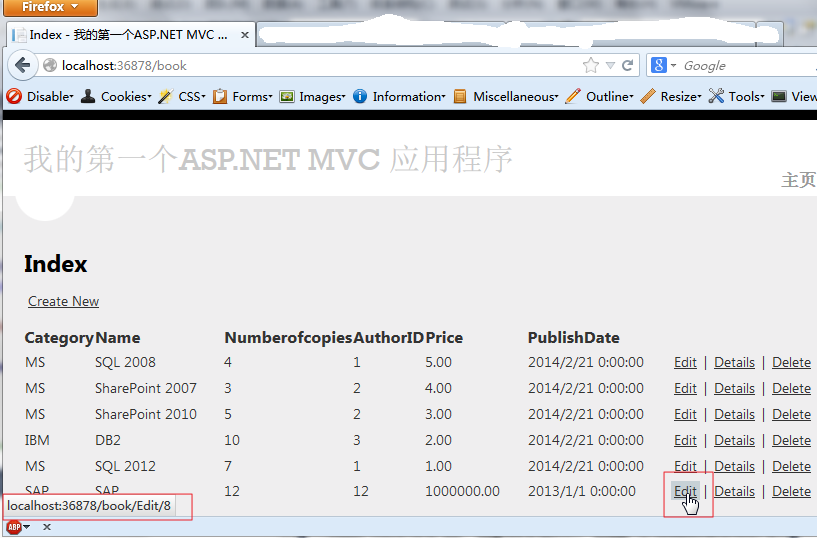
[**学习ASP.NET MVC(六)——我的第一个ASP.NET MVC 编辑页面**](http://www.cnblogs.com/chillsrc/p/3871829.html)

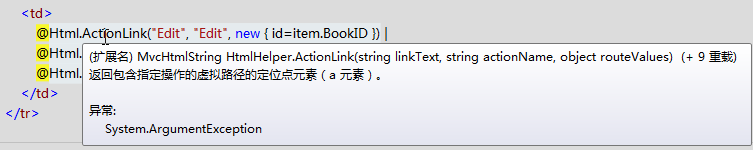
      在上一文章中由Entity Framework(实体框架)去实现了对数据库的CURD操作。在本篇文章中，主要是调试修改自动生成的动作方法和视图，以及调试编辑功能与编辑功能的Book控制器。

      首先，在Visual Studio中运行一下上次的应用程序，通过浏览器访问http://localhost:36878/Book。将鼠标指针移到浏览器中的一个“Edit”链接上，就可以看到这个“Edit”指向的URL。如下图红框所示。



       “Edit” 链接通过Html.ActionLink方法在浏览中生成指向Views\Book\Edit.cshtml 视图的链接。代码如下。

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



       如上图中所示，这个“HTML”对象是通过System.Web.Mvc.WebViewPage基类的属性暴露出来的。这个ActionLink方法可以很容易地动态生成一个链接，这个链接指向控制器的一个动作方法。ActionLink方法的第一个参数是显示链接的文本信息（例如，“Edit”）。第二个参数是指动作方法要调用控制器中的方法名称。最后一个参数是生成的路由数据（如示例中的ID=8）的一个匿名对象。

       在上图中所示的生成的“Edit”链接是http://localhost:36878/Book/Edit/8。默认路由URL模板是{controller}/{action}/{id}（模板代码位于App\_Start\ RouteConfig.cs）。因此，浏览器会发出[http://localhost:36878/Book/Edit/8](http://localhost:36878/Book/Edit/8%E8%BF%99%E4%B8%AAURL%E8%AF%B7%E6%B1%82%E4%B8%8E%E5%B9%B6%E4%BC%A0%E9%80%92%E5%8F%82%E6%95%B0ID=8) 这个URL请求，同时传递参数ID=8给Asp.net MVC的Book控制器的“Edit”方法。下面就是App\_Start\ RouteConfig.cs里面的路由模板设置。

[复制代码](javascript:void(0);)

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",

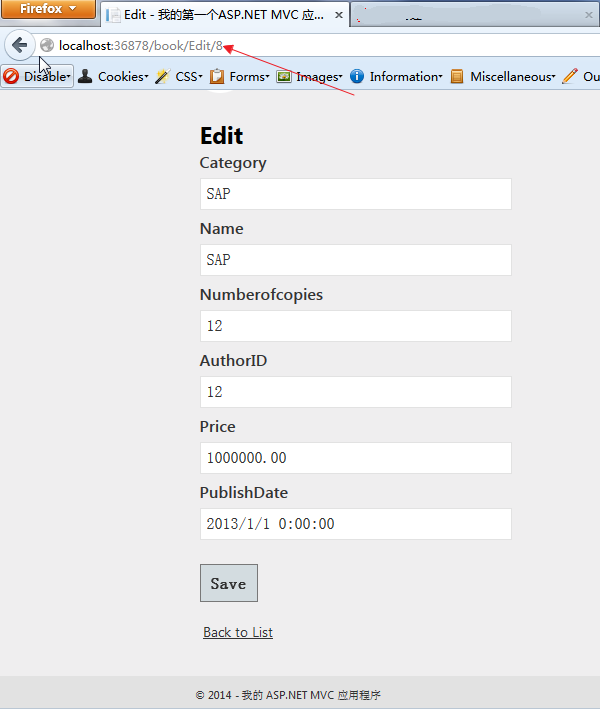
id = UrlParameter.Optional }

);

}

[复制代码](javascript:void(0);)

          您还可以使用action方法后面带上查询参数。例如，URL <http://localhost:36878/Book/Edit?ID=8>，通过ID=8，把参数ID的数据（8）传递给了Book控制器的“Edit”方法。以上两种方式的执行结果如下图。



         其次，我们打开BookController.cs文件，来仔细看看里面的两个“Edit”方法。代码如下所示。

[复制代码](javascript:void(0);)

//

// GET: /Book/Edit/8

public ActionResult Edit(int id = 0)

{

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

if (book == null)

{

return HttpNotFound();

}

return View(book);

}

//

// POST: /Book/Edit/8

[HttpPost]

public ActionResult Edit(Book book)  
 {

if (ModelState.IsValid)

{

db.Entry(book).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(book);

}

[复制代码](javascript:void(0);)

      请注意，第二个“Edit”方法的上面有一个“HttpPost”属性。这个属性指定了此方法只能通过POST请求来调用。也可以给第一个方法上面加上“HTTPGET”属性，但这不是必要的，因为HTTPGET属性是默认的。 （Visual Studio将会给所有没有指明的方法，默认且隐性的分配HTTPGET属性，这种隐性分配了“HTTPGET”属性的方法称为HTTPGET操作方法。）   
        第一个“HTTPGET”的“ Edit”方法是将书籍的ID做为参数，使用Entity Framework查找方法找到指定ID的书籍，并返回找到的书籍数据给编辑视图。 此方法中的ID参数被指定了默认值，如果当调用时不带参数，则ID会使用默认值0。如果书籍记录没有找到，将会返回HttpNotFound。由基架系统创建的编辑视图通过<label>与<input>来呈现Book类中的每个属性。示例代码如下。

[复制代码](javascript:void(0);)

@model MvcApplication1.Models.Book

@{

ViewBag.Title = "Edit";

}

<h2>Edit</h2>

@using (Html.BeginForm()) {

@Html.ValidationSummary(true)

<fieldset>

<legend>Book</legend>

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

<div class="editor-label">

@Html.LabelFor(model => model.Category)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.Category)

@Html.ValidationMessageFor(model => model.Category)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.Name)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.Name)

@Html.ValidationMessageFor(model => model.Name)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.Numberofcopies)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.Numberofcopies)

@Html.ValidationMessageFor(model => model.Numberofcopies)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.AuthorID)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.AuthorID)

@Html.ValidationMessageFor(model => model.AuthorID)

</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.PublishDate)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.PublishDate)

@Html.ValidationMessageFor(model => model.PublishDate)

</div>

<p>

<input type="submit" value="Save" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

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

}

[复制代码](javascript:void(0);)

       请注意，这个视图模板文件的顶部中有一句@model MvcApplication1.Models.Book，这一句代码的意思是在文件中声明了对象的类型定义 ——即指定了本视图中的对象类型是Book。   
       由VS自动生成的基架代码中使用了几个辅助方法，这几个辅助方法用来减少输入HTML标记。

    第一个辅助方法是：Html.LabelFor辅助方法用于显示字段的名称（例如：“Name”，“PublishDate”，“Price”）。

     第二个辅助方法是：Html.EditorFor辅助方法用于自动生成一个HTML<input>元素。

    第三个辅助方法是：Html.ValidationMessageFor辅助方法用于显示与该属性关联的验证消息。

      再次，按F5运行应用程序，并在浏览器中浏览/Book网址，在页面中，使用鼠标左键单击“Edit”链接。浏览器会自动导航到编辑页面，在页面中，单击鼠标右键，在弹出菜单中选择“查看源代码”，你会看到已经生成的HTML表单元素。如下。

[复制代码](javascript:void(0);)

<form action="/book/Edit/8" method="post"> <fieldset>

<legend>Book</legend>

<input data-val="true" data-val-number="字段 BookID 必须是一个数字。" data-val-required="BookID 字段是必需的。" id="BookID"   
 name="BookID" type="hidden" value="8" />

<div class="editor-label">

<label for="Category">Category</label>

</div>

<div class="editor-field">

<input class="text-box single-line" id="Category" name="Category" type="text" value="SAP" />

<span class="field-validation-valid" data-valmsg-for="Category" data-valmsg-replace="true"></span>

</div>

<div class="editor-label">

<label for="Name">Name</label>

</div>

<div class="editor-field">

<input class="text-box single-line" id="Name" name="Name" type="text" value="SAP" />

<span class="field-validation-valid" data-valmsg-for="Name" data-valmsg-replace="true"></span>

</div>

<div class="editor-label">

<label for="Numberofcopies">Numberofcopies</label>

</div>

<div class="editor-field">

<input class="text-box single-line" data-val="true" data-val-number="字段 Numberofcopies 必须是一个数字。"   
 data-val-required="Numberofcopies 字段是必需的。" id="Numberofcopies" name="Numberofcopies" type="number" value="12" />

<span class="field-validation-valid" data-valmsg-for="Numberofcopies" data-valmsg-replace="true"></span>

</div>

<div class="editor-label">

<label for="AuthorID">AuthorID</label>

</div>

<div class="editor-field">

<input class="text-box single-line" data-val="true" data-val-number="字段 AuthorID 必须是一个数字。"   
 data-val-required="AuthorID 字段是必需的。" id="AuthorID" name="AuthorID" type="number" value="12" />

<span class="field-validation-valid" data-valmsg-for="AuthorID" 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="字段 Price 必须是一个数字。"   
data-val-required="Price 字段是必需的。" id="Price" name="Price" type="text" value="1000000.00" />

<span class="field-validation-valid" data-valmsg-for="Price" data-valmsg-replace="true"></span>

</div>

<div class="editor-label">

<label for="PublishDate">PublishDate</label>

</div>

<div class="editor-field">

<input class="text-box single-line" data-val="true" data-val-date="字段 PublishDate 必须是日期。"   
 data-val-required="PublishDate 字段是必需的。" id="PublishDate" name="PublishDate" type="datetime" value="2013/1/1 0:00:00" />

<span class="field-validation-valid" data-valmsg-for="PublishDate" data-valmsg-replace="true"></span>

</div>

<p>

<input type="submit" value="Save" />

</p>

</fieldset>

</form>

[复制代码](javascript:void(0);)

      其中最后一个<input>元素设置为Submit，用于提交HTML <form>中的表格数据到/Book/Edit URL。

**处理POST请求**  
       下面的代码示例，显示了Edit操作方法的HttpPost版本。

[复制代码](javascript:void(0);)

[HttpPost]

public ActionResult Edit(Book book)

{

if (ModelState.IsValid)

{

db.Entry(book).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

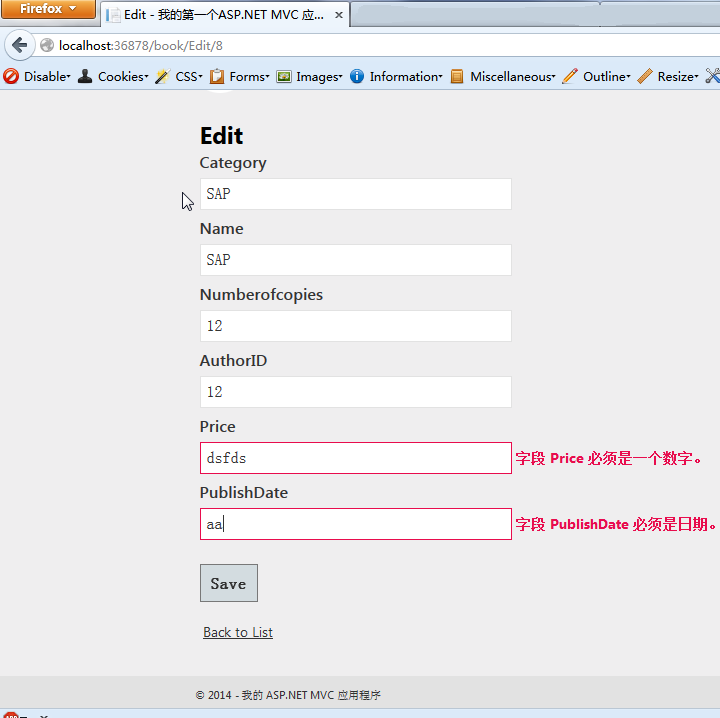
}

return View(book);

}

[复制代码](javascript:void(0);)

        ASP.NET MVC Model Binder简单的说就是控制器中的Action方法需要参数数据；而这些参数数据包含在HTTP请求中，包括表单上的Value和URL中的参数等。通过Model Binder的功能将表单上的Value和URL中的参数换成Book对象，然后将Book对象绑定到Action操作方法中的参数上面。其中ModelState.IsValid方法验证用户提交的表单中的数据是否可以被用于修改（编辑或更新）一个Book对象。如果数据是有效的，Book数据将被保存到数据库的Book集合中（BookDBContext实例）。新的Book数据是通过调用BookDBContext的SaveChanges方法保存到数据库中。保存数据后，代码将用户重定向到BookController类的Index操作，同时在浏览器中显示最新的Book列表。   
        如果提交过来的表单中的值是无效的，这些值将重新显示在浏览器的页面中。在Edit.cshtml视图模板的Html.ValidationMessageFor助手方法显示相应的错误消息。如下图。



          请注意，ASP.NET MVC 4.0 已经支持jQuery的验证方法，在非英语语言环境中使用逗号（“，”）代替小数点，你必须包括globalize.js和你自己所需要的特定语言的globalize.cultures.js文件（从https://github.com/ jQuery/ globalize 下载）和JavaScript中使用Globalize.parseFloat。下面的代码显示了修改Views\Book\ Edit.cshtml文件中显示法语（“fr-FR”）：

[复制代码](javascript:void(0);)

@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>

}

[复制代码](javascript:void(0);)

         小数字段可能需要使用一个逗号，不是小数点。作为一个临时的解决办法，你可以在全球化的元素添加到项目的根目录下的Web.config文件中。下面的代码显示了全球化设置为”en-US”。

[复制代码](javascript:void(0);)

<system.web>

<globalization culture ="en-US" />

<!--elements removed for clarity-->

</system.web>

[复制代码](javascript:void(0);)

**说明：所有的创建，编辑，删除或以其他方式修改数据的方法都是通过HttpPost将数据传给相应控制器的相应方法。**

               HTTPGET方法遵循类似的模式。他们获取一个书籍（Book）对象（或是对象列表，在Index方法情况下） ，同时将对象数据传递给视图，由视图进行呈现。 Create方法传递一个空的书籍对象给Create视图。通过HTTP GET方式修改数据存在一个安全隐患。使用GET方法修改数据也违反了HTTP的最佳实践和REST的架构模式，它指定了GET请求不应该改变你的应用程序的状态。换句话说，执行GET操作应该是一个安全的操作，没有任何副作用，不会修改你的持久化的数据。