**一步步打造一个简单的 MVC 电商网站 – BooksStore**

**http://www.cnblogs.com/liqingwen/p/6640861.html**

## 简介

　　主要功能与知识点如下：

　　　　分类、产品浏览、购物车、结算、CRUD（增删改查） 管理、发邮件、分页、模型绑定、认证过滤器和单元测试等（预计四篇、周五、下周一和周二）。

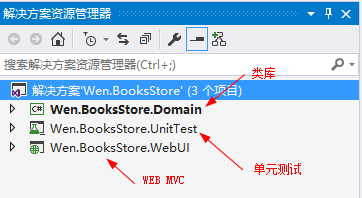
     【备注】项目使用 VS2015 + C#6 进行开发，有问题请发表在留言区哦，还有，页面长得比较丑，请见谅。

## 目录

* 创建项目架构
* 创建域模型实体
* 创建单元测试
* 创建控制器与视图
* 创建分页
* 加入样式

## 一、创建项目架构

　　1.新建一个解决方案“BooksStore”，并添加以下项目：

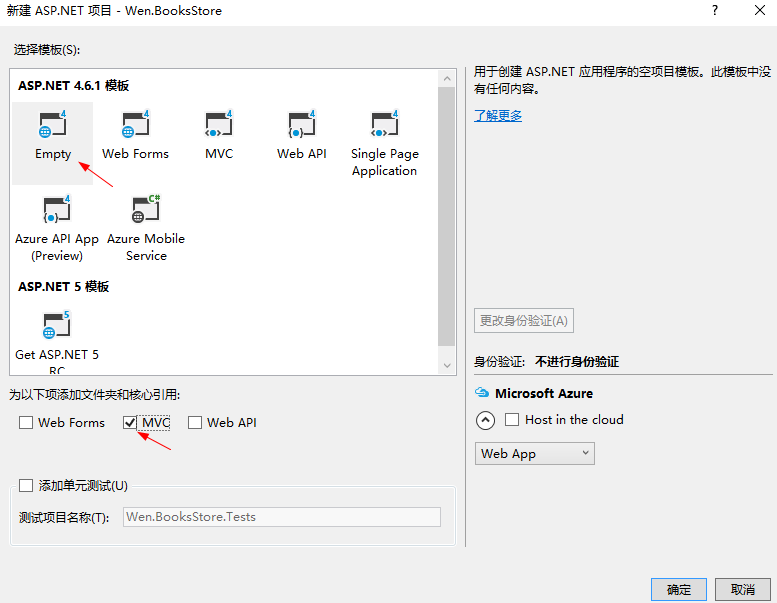


     BooksStore.Domain：类库，存放域模型和逻辑，使用 EF；

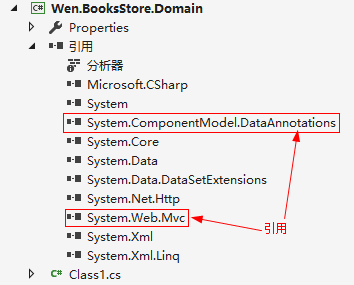
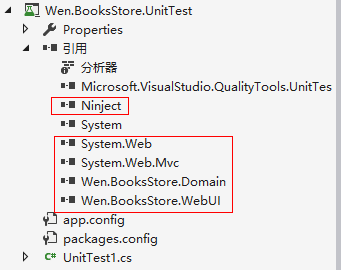
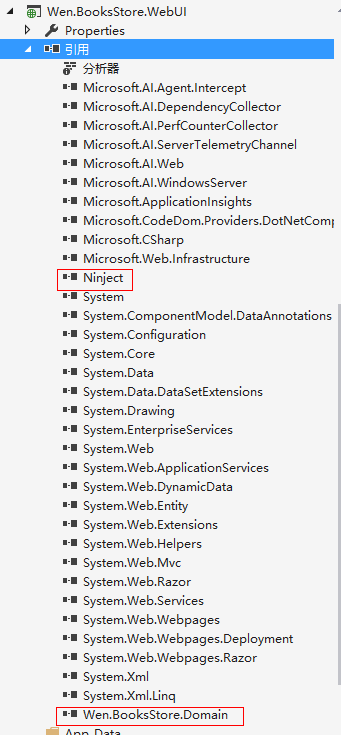
     BooksStore.WebUI：Web MVC 应用程序，存放视图和控制器，充当显示层，使用了 Ninject 作为 DI 容器；

     BoosStore.UnitTest：单元测试，对上述两个项目进行测试。

　　Web MVC 为一个空的 MVC 项目：



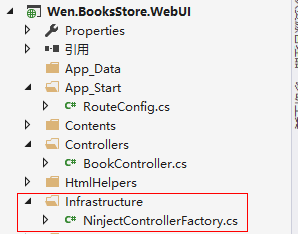
　　2.添加项目引用（需要使用 NuGet）：

　　这是不同项目需要引用的类库和项目

　　3.设置 DI 容器

     我们通过 Ninject ，创建一个自定义的工厂，一个名为 NinjectControllerFactory 的类继承 DefaultControllerFactory（默认的控制器工厂）。你也可以在里面添加自定义的代码，改变 MVC 框架的默认行为。



 　　AddBindings() 添加绑定方法，先留空。

[复制代码](javascript:void(0);)

public class NinjectControllerFactory : DefaultControllerFactory

{

private readonly IKernel \_kernel;

public NinjectControllerFactory()

{

\_kernel = new StandardKernel();

AddBindings();

}

protected override IController GetControllerInstance(RequestContext requestContext, Type controllerType)

{

return controllerType == null

? null

: (IController) \_kernel.Get(controllerType);

}

/// <summary>

/// 添加绑定

/// </summary>

private void AddBindings()

{

}

}

[复制代码](javascript:void(0);)

　　4.并且在 Global.asax 中加入一行代码，告诉 MVC 用新建的类来创建控制器对象。

ControllerBuilder.Current.SetControllerFactory(new NinjectControllerFactory());

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

RouteConfig.RegisterRoutes(RouteTable.Routes);

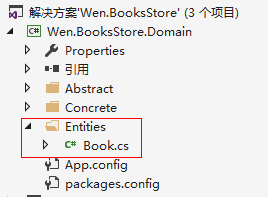
ControllerBuilder.Current.SetControllerFactory(new NinjectControllerFactory());

}

}

[复制代码](javascript:void(0);)

## 二、创建域模型实体



     1.在图中位置创建一个名为 Book 的实体类。

[复制代码](javascript:void(0);)

public class Book

{

/// <summary>

/// 标识

/// </summary>

public int Id { get; set; }

/// <summary>

/// 名称

/// </summary>

public string Name { get; set; }

/// <summary>

/// 描述

/// </summary>

public string Description { get; set; }

/// <summary>

/// 价格

/// </summary>

public decimal Price { get; set; }

/// <summary>

/// 分类

/// </summary>

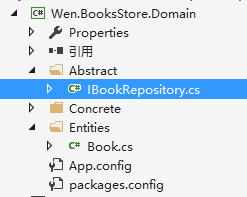
public string Category { get; set; }

}

[复制代码](javascript:void(0);)

　　有了实体之后，我们应该创建一个“库”对该实体进行操作，而这种持久化逻辑操作也应该和域模型是进行隔离的。

　　2.先定义一个接口 IbookRepository，在根目录创建一个名为 Abstract 的文件夹，顾名思义就是应该放置一些抽象的类，如接口。



public interface IBookRepository

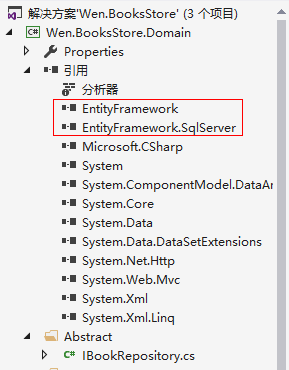
{

IQueryable<Book> Books { get; }

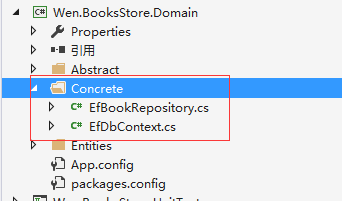
}

　　我们通过该接口就可以得到对应类的相关信息，而不需要去管该数据如何存储，以及存储的位置，这就是存储库模式的本质。

　　3.接下来，我们就需要对数据库进行操作了，我们使用简单的 EF（ORM 对象关系模型） 去对数据库进行操作，所以需要自己通过 Nuget 下载 EF 的类库。



　　4.因为之前定义了接口类，接下来就应该定义实现该接口的类了



　　安装完之后，再次建立一个名为 Concrete 的文件夹，存放实例。

　　在里面创建一个 EfDbContext 的类，派生于 DbContext，该类会为用户要使用的数据库中的每个表自动的定义一个属性。该属性名为 Books，指定了表名，DbSet<Book> 表示为 Book 实体的表模型，Book 对象相当于 Books 表中的行（记录）。

public class EfDbContext : DbContext

{

public DbSet<Book> Books { get; set; }

}

　　再创建一个 EfBookRepository 存储库类，它实现 IBookRepository 接口，使用了上文创建的 EfDbContext 上下文对象，包含了具体的方法定义。

[复制代码](javascript:void(0);)

public class EfBookRepository : IBookRepository

{

private readonly EfDbContext \_context = new EfDbContext();

public IQueryable<Book> Books => \_context.Books;

}

[复制代码](javascript:void(0);)

 　　5.现在只差在数据库新建一张表了。

[复制代码](javascript:void(0);)

CREATE TABLE Book

(

Id INT IDENTITY PRIMARY KEY,

Name NVARCHAR(**100**),

Description NVARCHAR(MAX),

Price DECIMAL,

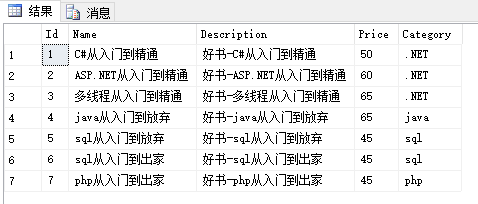
Category NVARCHAR(**50**)

)

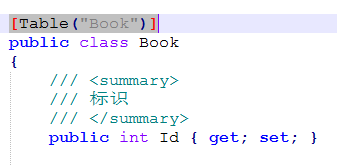
[复制代码](javascript:void(0);)

　　 并插入测试数据：

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 测试数据

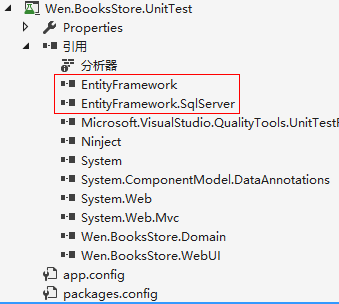


　　因为我希望表名为 Book，而不是 Books，所以我在之前的 Book 类上加上特性 [Table("Book")] ：



## 三、创建单元测试

　　1.做完预热操作后，你可能想立即以界面的的方式进行显示，别急，先用单元测试检查一下我们对数据库的操作是否正常，通过对数据进行简单的读取，检查下连接是否成功。



　　2.单元测试也需要引入 ef 类库（Nuget）。

　　3.安装完之后会生成一个 app.config 配置文件，需要额外添加一行连接字符串（在后续的 Web UI 项目里，也需要加上这条信息，不然会提示对应的错误信息）。

<connectionStrings>

<add name="EfDbContext" connectionString="server=.;database=TestDb;uid=sa;pwd=**123**" providerName="System.Data.SqlClient"/>

</connectionStrings>

　　 4.当所有前置工作都准备好了的时候，就应该填写测试方法了，因为我插入了 7 条数据，这里我就判断一下从数据库读取出的行数是否为 7 ：

[复制代码](javascript:void(0);)

[TestMethod]

public void BooksCountTest()

{

var bookRepository=new EfBookRepository();

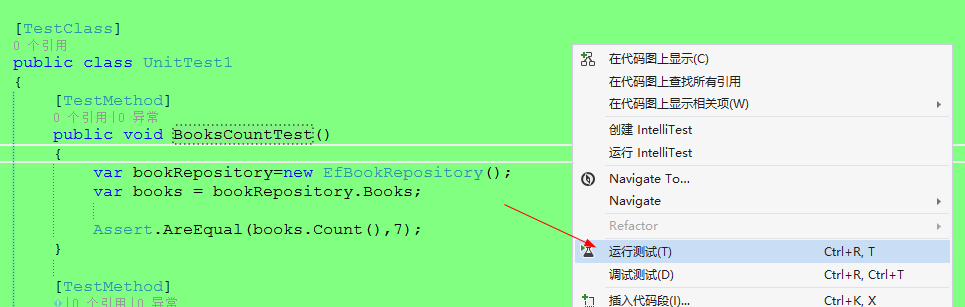
var books = bookRepository.Books;

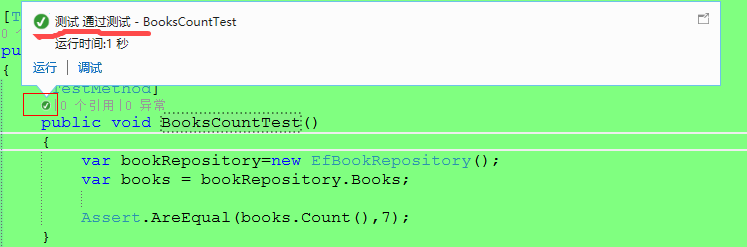
Assert.AreEqual(books.Count(),**7**);

}

[复制代码](javascript:void(0);)

　　5.在该方法体的内部单击右键，你可以看到一个“运行测试”的选项，这时你可以尝试单击它：



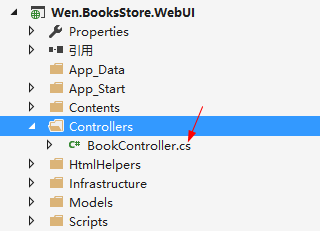


　　从这个符号可以看到，是成功了！

　　接下来，我们要正式从页面显示我们想要的信息了。

## 四、创建控制器与视图

　　1.先新建一个空的控制器：BookController：



　　2.需要我们自定义一个 Details 方法，用于后续与界面进行交互。

[复制代码](javascript:void(0);)

public class BookController : Controller

{

private readonly IBookRepository \_bookRepository;

public BookController(IBookRepository bookRepository)

{

\_bookRepository = bookRepository;

}

/// <summary>

/// 详情

/// </summary>

/// <returns></returns>

public ActionResult Details()

{

return View(\_bookRepository.Books);

}

}

[复制代码](javascript:void(0);)

　　3.接下来，要创建一个视图 View 了。



　　4.将 Details.cshtml 的内容替换为下面的：

[复制代码](javascript:void(0);)

@model IEnumerable<Wen.BooksStore.Domain.Entities.Book>

@{

ViewBag.Title = "Books";

}

@foreach (var item in Model)

{

<div>

<h3>@item.Name</h3>

@item.Description

<h4>@item.Price.ToString("C")</h4>

<br />

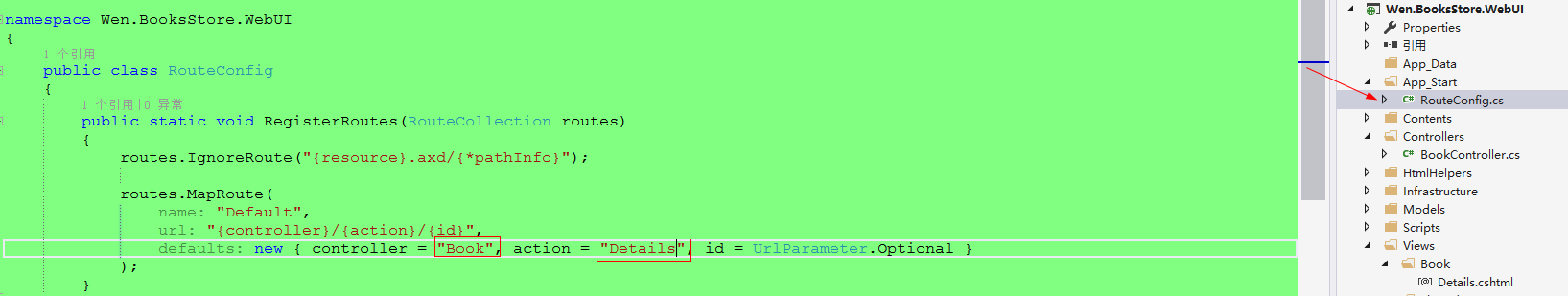
<hr />

</div>

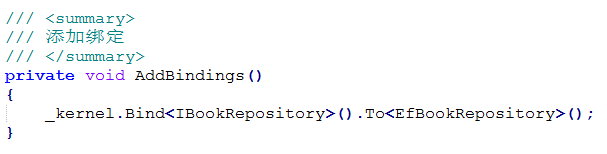
}

[复制代码](javascript:void(0);)

　　5.改下默认的路由机制，让他默认跳转到该页面。



　　6.还有一点需要注意的是，因为我们使用了 Ninject 容器，并且需要对控制器中的构造函数中的参数 IBookRepository 进行解析，告诉他将使用哪个对象对该接口进行服务，也就是需要修改之前的 AddBindings 方法：

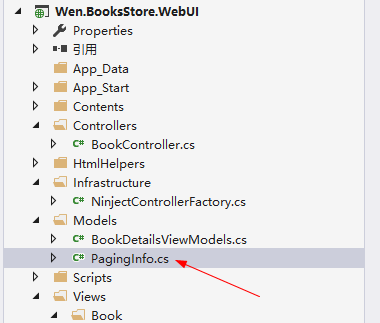


　　7.运行的效果大致如下（因为加了点 CSS 样式，所以显示的效果可能有些许不同），结果是一致的。



## 五、创建分页

　　1.在 Models 文件夹新增一个 PagingInfo.cs 分页信息类。



[复制代码](javascript:void(0);)

/// <summary>

/// 分页信息

/// </summary>

public class PagingInfo

{

/// <summary>

/// 总数

/// </summary>

public int TotalItems { get; set; }

/// <summary>

/// 页容量

/// </summary>

public int PageSize { get; set; }

/// <summary>

/// 当前页

/// </summary>

public int PageIndex { get; set; }

/// <summary>

/// 总页数

/// </summary>

public int TotalPages => (int)Math.Ceiling((decimal)TotalItems / PageSize);

}

[复制代码](javascript:void(0);)

　　2.新增一个 HtmlHelpers 文件夹存放一个基于 Html 帮助类的扩展方法：



[复制代码](javascript:void(0);)

public static class PagingHelper

{

/// <summary>

/// 分页

/// </summary>

/// <param name="helper"></param>

/// <param name="pagingInfo"></param>

/// <param name="func"></param>

/// <returns></returns>

public static MvcHtmlString PageLinks(this HtmlHelper helper, PagingInfo pagingInfo, Func<int, string> func)

{

var sb = new StringBuilder();

for (var i = 1; i <= pagingInfo.TotalPages; i++)

{

//创建 <a> 标签

var tagBuilder = new TagBuilder("a");

//添加特性

tagBuilder.MergeAttribute("href", func(i));

//添加值

tagBuilder.InnerHtml = i.ToString();

if (i == pagingInfo.PageIndex)

{

tagBuilder.AddCssClass("selected");

}

sb.Append(tagBuilder);

}

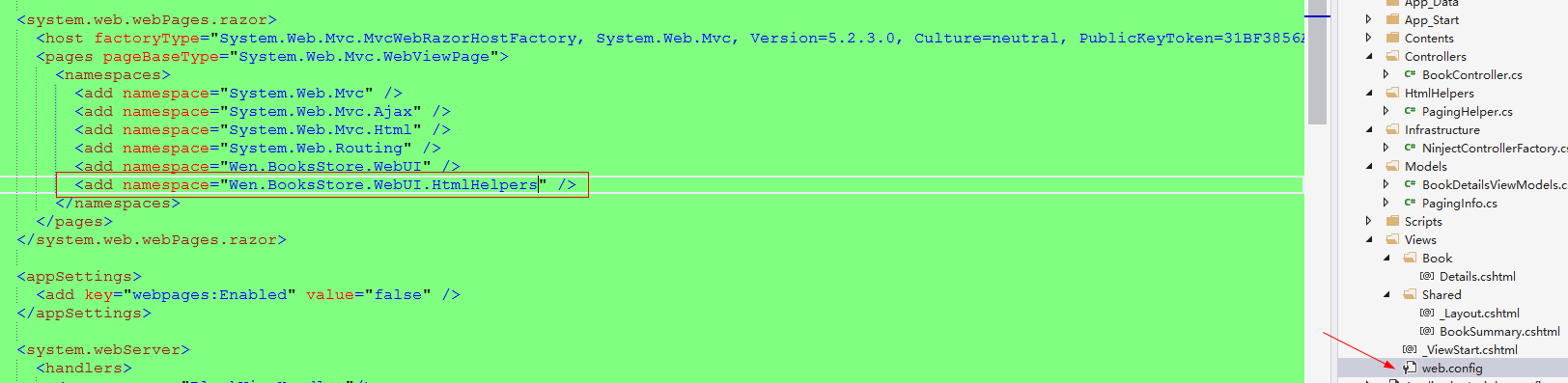
return MvcHtmlString.Create(sb.ToString());

}

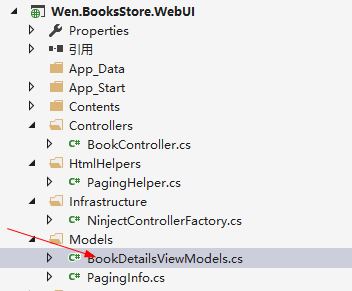
}

[复制代码](javascript:void(0);)

 　　3.添加完毕后需要在配置文件内加入该命名空间



　　4.现在要重新修改 BookController.cs 控制器内的的代码，并添加新的视图模型类 BookDetailsViewModels.cs，让它继承之前的分页类。



public class BookDetailsViewModels : PagingInfo

{

public IEnumerable<Book> Books { get; set; }

}

　　修改后的控制器代码：

[复制代码](javascript:void(0);)

public class BookController : Controller

{

private readonly IBookRepository \_bookRepository;

public int PageSize = 5;

public BookController(IBookRepository bookRepository)

{

\_bookRepository = bookRepository;

}

/// <summary>

/// 详情

/// </summary>

/// <param name="pageIndex"></param>

/// <returns></returns>

public ActionResult Details(int pageIndex = 1)

{

var model = new BookDetailsViewModels()

{

Books = \_bookRepository.Books.OrderBy(x => x.Id).Skip((pageIndex - 1) \* PageSize).Take(PageSize),

PageSize = PageSize,

PageIndex = pageIndex,

TotalItems = \_bookRepository.Books.Count()

};

return View(model);

}

}

[复制代码](javascript:void(0);)

　　5.修改视图模型后，对应的视图页也需要修改

[复制代码](javascript:void(0);)

@model Wen.BooksStore.WebUI.Models.BookDetailsViewModels

@{

ViewBag.Title = "Books";

}

@foreach (var item in Model.Books)

{

<div>

<h3>@item.Name</h3>

@item.Description

<h4>@item.Price.ToString("C")</h4>

<br />

<hr />

</div>

}

<div class="pager">

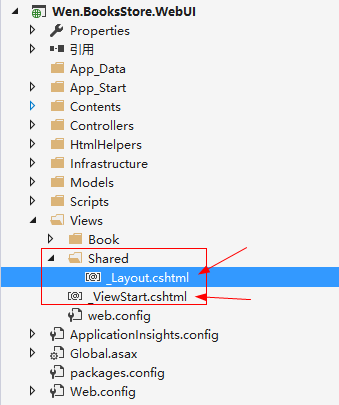
@Html.PageLinks(Model, x => Url.Action("Details", new { pageIndex = x }))

</div>

[复制代码](javascript:void(0);)

## 六、加入样式

　　1.页面的样式简单的设计为 3 大板块，顶部为标题，左侧边栏为分类，主模块将显示具体内容。

　　我们现在要在 Views 文件夹下创建一个文件 \_ViewStart.cshtml，再创建一个 Shared 的文件夹和文件 \_Layout.cshtml。  


　　2.\_Layout.cshtml 这是布局页，当代码执行到 @RenderBody() 时，就会负责将之前 Details.cshtml 的内容进行渲染：

[复制代码](javascript:void(0);)

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

<link href="~/Contents/Site.css" rel="stylesheet" />

</head>

<body>

<div id="header">

<div class="title">图书商城</div>

</div>

<div id="sideBar">分类</div>

<div id="content">

@RenderBody()

</div>

</body>

</html>

[复制代码](javascript:void(0);)

　　 \_ViewStart.cshtml 该文件表示默认的布局页为该视图文件：

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

 　　3.网站的根目录下也要添加一个名为 Contents 的文件夹，用于存放 CSS。

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

body {

}

#header, #content, #sideBar {

display: block;

}

#header {

background-color: green;

border-bottom: 2px solid #111;

color: White;

}

#header, .title {

font-size: 1.5em;

padding: .5em;

}

#sideBar {

float: left;

width: 8em;

padding: .3em;

}

#content {

border-left: 2px solid gray;

margin-left: 10em;

padding: 1em;

}

.pager {

text-align: right;

padding: .5em 0 0 0;

margin-top: 1em;

}

.pager A {

font-size: 1.1em;

color: #666;

padding: 0 .4em 0 .4em;

}

.pager A:hover {

background-color: Silver;

}

.pager A.selected {

background-color: #353535;

color: White;

}

[复制代码](javascript:void(0);)



　　现在，分页也已经有了效果，基本界面就出来了。

 　　本系列的 GitHub地址：<https://github.com/liqingwen2015/Wen.BooksStore>

## 错误修正

　　感谢 [韩之一](http://www.cnblogs.com/lost0/) ：BooksCountTest() 这个方法里 Assert.AreEqual(books.Count(), 7); 参数写反了, 第一个是期望值, 第二个是真实值。

上一次我们尝试了：创建项目架构、创建域模型实体、创建单元测试、创建控制器与视图、创建分页和加入样式，而这一节我们会完成两个功能，分类导航与购物车。

　　主要功能与知识点如下：

　　　　分类、产品浏览、购物车、结算、CRUD（增删改查） 管理、发邮件、分页、模型绑定、认证过滤器和单元测试等（预计剩余两篇，预计明天（因为周六不放假）和周三（因为周二不上班）发布）。

     【备注】项目使用 VS2015 + C#6 进行开发，有问题请发表在留言区哦，还有，页面长得比较丑，请见谅。

## 目录

* 添加分类导航
* 加入购物车
* 创建一个分部视图 Partial View

## 一、添加分类导航

　　上一次我们把网页划分成了三个模块，其中左侧栏的部分尚未完成，左侧栏拥有将书籍分类展示的功能。



图 1

　　1.回到之前的 BookDetailsViewModels 视图模型，我们额外再添加一个新的属性用作分类（CurrentCategory）：

[复制代码](javascript:void(0);)

/// <summary>

/// 书籍详情视图模型

/// </summary>

public class BookDetailsViewModels : PagingInfo

{

public IEnumerable<Book> Books { get; set; }

/// <summary>

/// 当前分类

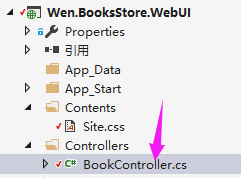
/// </summary>

public string CurrentCategory { get; set; }

}

[复制代码](javascript:void(0);)

　　2.修改完视图模型，现在就应该修改对应的 BookController 中的 Details 方法



[复制代码](javascript:void(0);)

/// <summary>

/// 详情

/// </summary>

/// <param name="category">分类</param>

/// <param name="pageIndex">页码</param>

/// <returns></returns>

public ActionResult Details(string category, int pageIndex = 1)

{

var model = new BookDetailsViewModels

{

Books =

\_bookRepository.Books.Where(x => category == null || x.Category == category)

.OrderBy(x => x.Id)

.Skip((pageIndex - 1) \* PageSize)

.Take(PageSize),

CurrentCategory = category,

PageSize = PageSize,

PageIndex = pageIndex,

TotalItems = \_bookRepository.Books.Count(x => category == null || x.Category == category)

};

return View(model);

}

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif BookController.cs

　　参数增加了一个 category，用于获取分类的字符串，对应 Books 中的属性的赋值语句改为 \_bookRepository.Books.Where(x => category == null || x.Category == category)，这里的 Lambda 表达式 x => category == null || x.Category == category 的意思是，分类字符串为空就取库中所有的 Book 实体，不为空时根据分类进行对集合进行筛选过滤。

　　还要对属性 CurrentCategory 进行赋值。

　　别忘了，因为分页是根据 TotalItems 属性进行的，所以还要修改地方 \_bookRepository.Books.Count(x => category == null || x.Category == category)，通过 LINQ 统计不同分类情况的个数。

　　3.该控制器对应的 Details.cshtml 中的分页辅助器也需要修改，添加新的路由参数：

<div class="pager">

@Html.PageLinks(Model, x => Url.Action("Details", new { pageIndex = x, category = Model.CurrentCategory }))

</div>

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 @model Wen.BooksStore.WebUI.Models.BookDetailsViewModels

2

3 @{

4 ViewBag.Title = "Books";

5 }

6

7 @foreach (var item in Model.Books)

8 {

9 <div class="item">

10 <h3>@item.Name</h3>

11 @item.Description

12 <h4>@item.Price.ToString("C")</h4>

13 <br />

14 <hr />

15 </div>

16 }

17

18 <div class="pager">

19 @Html.PageLinks(Model, x => Url.Action("Details", new { pageIndex = x, category = Model.CurrentCategory }))

20 </div>

[复制代码](javascript:void(0);)

　　4.路由区域也应当修改一下

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[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}",

defaults: new { controller = "Book", action = "Details" }

);

routes.MapRoute(

name: null,

url: "{controller}/{action}/{category}",

defaults: new { controller = "Book", action = "Details" }

);

routes.MapRoute(

name: null,

url: "{controller}/{action}/{category}/{pageIndex}",

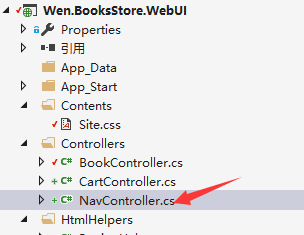
defaults: new { controller = "Book", action = "Details", pageIndex = UrlParameter.Optional }

);

}

[复制代码](javascript:void(0);)

　　5.现在新建一个名为 NavController 的控制器，并添加一个名为 Sidebar 的方法，专门用于渲染左侧边栏。



　　不过返回的 View 视图类型变成 PartialView 分部视图类型：

public PartialViewResult Sidebar(string category = null)

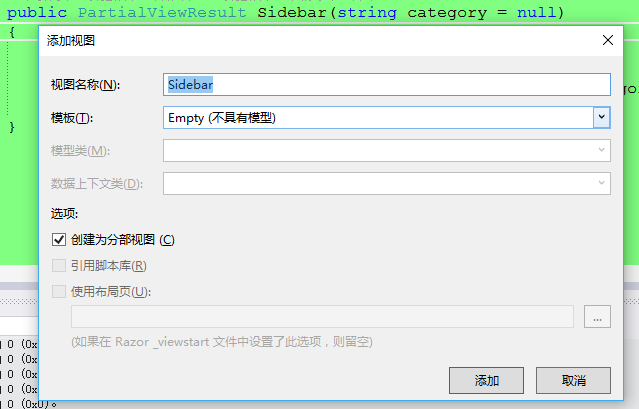
{

var categories = \_bookRepository.Books.Select(x => x.Category).Distinct().OrderBy(x => x);

return PartialView(categories);

}

　　在方法体在右键，添加一个视图，勾上创建分部视图。



　　Sidebar.cshtml 修改为：

[复制代码](javascript:void(0);)

@model IEnumerable<string>

<ul>

<li>@Html.ActionLink("所有分类", "Details", "Book")</li>

@foreach (var item in Model)

{

<li>@Html.RouteLink(item, new { controller = "Book", action = "Details", category = item, pageIndex = 1 }, new { @class = item == ViewBag.CurrentCategory ? "selected" : null })</li>

}

</ul>

[复制代码](javascript:void(0);)

　　MVC 框架具有一种叫作“子动作（Child Action）”的概念，可以适用于重用导航控件之类的东西，使用类似 RenderAction() 的方法，在当前的视图中输出指定的动作方法。

　　因为需要在父视图中呈现另一个 Action 中的分部视图，所以原来的 \_Layout.cshtml 布局页修改如下：



　　现在，启动的结果应该和图 1 是一样的，尝试点击左侧边栏的分类，观察主区域的变化情况。

## 二、加入购物车

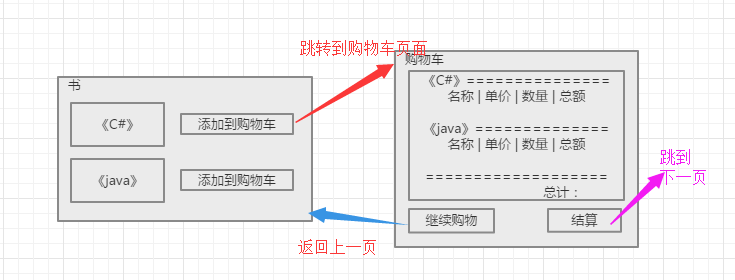
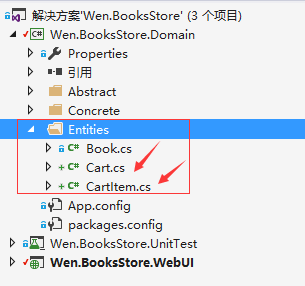


 图 2

　　界面的大体功能如图 2，在每本图书的区域新增一个链接（添加到购物车），会跳转到一个新的页面，显示购物车的详细信息 - 购物清单，也可以通过“结算”链接跳转到一个新的页面。

　　购物车是应用程序业务域的一部分，因此，购物车实体应该为域模型。



　　1.添加两个类：

　　Cart.cs 有添加、移除、清空和统计功能：

[复制代码](javascript:void(0);)

/// <summary>

/// 购物车

/// </summary>

public class Cart

{

private readonly List<CartItem> \_cartItems = new List<CartItem>();

/// <summary>

/// 获取购物车的所有项目

/// </summary>

public IList<CartItem> GetCartItems => \_cartItems;

/// <summary>

/// 添加书模型

/// </summary>

/// <param name="book"></param>

/// <param name="quantity"></param>

public void AddBook(Book book, int quantity)

{

if (\_cartItems.Count == 0)

{

\_cartItems.Add(new CartItem() { Book = book, Quantity = quantity });

return;

}

var model = \_cartItems.FirstOrDefault(x => x.Book.Id == book.Id);

if (model == null)

{

\_cartItems.Add(new CartItem() { Book = book, Quantity = quantity });

return;

}

model.Quantity += quantity;

}

/// <summary>

/// 移除书模型

/// </summary>

/// <param name="book"></param>

public void RemoveBook(Book book)

{

var model = \_cartItems.FirstOrDefault(x => x.Book.Id == book.Id);

if (model == null)

{

return;

}

\_cartItems.RemoveAll(x => x.Book.Id == book.Id);

}

/// <summary>

/// 清空购物车

/// </summary>

public void Clear()

{

\_cartItems.Clear();

}

/// <summary>

/// 统计总额

/// </summary>

/// <returns></returns>

public decimal ComputeTotalValue()

{

return \_cartItems.Sum(x => x.Book.Price \* x.Quantity);

}

}

[复制代码](javascript:void(0);)

　　CartItem.cs 表示购物车中的每一项：

[复制代码](javascript:void(0);)

/// <summary>

/// 购物车项

/// </summary>

public class CartItem

{

/// <summary>

/// 书

/// </summary>

public Book Book { get; set; }

/// <summary>

/// 数量

/// </summary>

public int Quantity { get; set; }

}

[复制代码](javascript:void(0);)

　　2.修改一下之前的 Details.cshtml，增加“添加到购物车”的按钮：

[复制代码](javascript:void(0);)

@model Wen.BooksStore.WebUI.Models.BookDetailsViewModels

@{

ViewBag.Title = "Books";

}

@foreach (var item in Model.Books)

{

<div class="item">

<h3>@item.Name</h3>

@item.Description

<h4>@item.Price.ToString("C")</h4>

@using (Html.BeginForm("AddToCart", "Cart"))

{

var id = item.Id;

@Html.HiddenFor(x => id);

@Html.Hidden("returnUrl", Request.Url.PathAndQuery)

<input type="submit" value="+ 添加到购物车" />

}

<br />

<hr />

</div>

}

<div class="pager">

@Html.PageLinks(Model, x => Url.Action("Details", new { pageIndex = x, category = Model.CurrentCategory }))

</div>

[复制代码](javascript:void(0);)

　　【备注】@Html.BeginForm() 方法默认会创建一个 Post 请求方法的表单，为什么不直接使用 Get 请求呢，HTTP 规范要求，会引起数据变化时不要使用 Get 请求，将产品添加到一个购物车明显会出现新的数据变化，所以，这种情形不应该使用 Get 请求，直接显示页面或者列表数据，这种请求才应该使用 Get。

　　3.先修改下 css 中的样式

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

body {

}

#header, #content, #sideBar {

display: block;

}

#header {

background-color: green;

border-bottom: 2px solid #111;

color: White;

}

#header, .title {

font-size: 1.5em;

padding: .5em;

}

#sideBar {

float: left;

width: 8em;

padding: .3em;

}

#content {

border-left: 2px solid gray;

margin-left: 10em;

padding: 1em;

}

.pager {

text-align: right;

padding: .5em 0 0 0;

margin-top: 1em;

}

.pager A {

font-size: 1.1em;

color: #666;

padding: 0 .4em 0 .4em;

}

.pager A:hover {

background-color: Silver;

}

.pager A.selected {

background-color: #353535;

color: White;

}

.item input {

float: right;

color: White;

background-color: green;

}

.table {

width: 100%;

padding: 0;

margin: 0;

}

.table th {

font: bold 12px "Trebuchet MS", Verdana, Arial, Helvetica, sans-serif;

color: #4f6b72;

border-right: 1px solid #C1DAD7;

border-bottom: 1px solid #C1DAD7;

border-top: 1px solid #C1DAD7;

letter-spacing: 2px;

text-transform: uppercase;

text-align: left;

padding: 6px 6px 6px 12px;

background: #CAE8EA no-repeat;

}

.table td {

border-right: 1px solid #C1DAD7;

border-bottom: 1px solid #C1DAD7;

background: #fff;

font-size: 14px;

padding: 6px 6px 6px 12px;

color: #4f6b72;

}

.table td.alt {

background: #F5FAFA;

color: #797268;

}

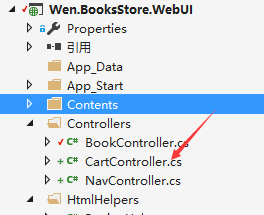
.table th.spec, td.spec {

border-left: 1px solid #C1DAD7;

}

[复制代码](javascript:void(0);)

　　4.再添加一个 CartController



[复制代码](javascript:void(0);)

/// <summary>

/// 购物车

/// </summary>

public class CartController : Controller

{

private readonly IBookRepository \_bookRepository;

public CartController(IBookRepository bookRepository)

{

\_bookRepository = bookRepository;

}

/// <summary>

/// 首页

/// </summary>

/// <param name="returnUrl"></param>

/// <returns></returns>

public ViewResult Index(string returnUrl)

{

return View(new CartIndexViewModel()

{

Cart = GetCart(),

ReturnUrl = returnUrl

});

}

/// <summary>

/// 添加到购物车

/// </summary>

/// <param name="id"></param>

/// <param name="returnUrl"></param>

/// <returns></returns>

public RedirectToRouteResult AddToCart(int id, string returnUrl)

{

var book = \_bookRepository.Books.FirstOrDefault(x => x.Id == id);

if (book != null)

{

GetCart().AddBook(book, 1);

}

return RedirectToAction("Index", new { returnUrl });

}

/// <summary>

/// 从购物车移除

/// </summary>

/// <param name="id"></param>

/// <param name="returnUrl"></param>

/// <returns></returns>

public RedirectToRouteResult RemoveFromCart(int id, string returnUrl)

{

var book = \_bookRepository.Books.FirstOrDefault(x => x.Id == id);

if (book != null)

{

GetCart().RemoveBook(book);

}

return RedirectToAction("Index", new { returnUrl });

}

/// <summary>

/// 获取购物车

/// </summary>

/// <returns></returns>

private Cart GetCart()

{

var cart = (Cart)Session["Cart"];

if (cart != null) return cart;

cart = new Cart();

Session["Cart"] = cart;

return cart;

}

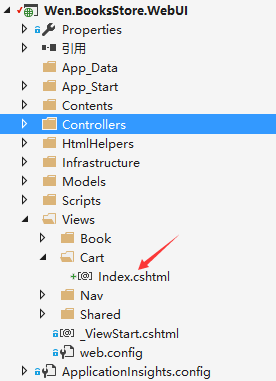
}

[复制代码](javascript:void(0);)

　　【备注】这里的购物车是通过 Session 会话状态进行保存用户的 Cart 对象。当会话过期（典型的情况是用户很长时间没有对服务器发起任何请求），与该会话关联的数据就会被删除，这就意味着不需要对 Cart 对象进行生命周期的管理。

　　【备注】RedirectToAction() 方法：将一个 HTTP 重定向的指令发给客户端浏览器，要求浏览器请求一个新的 Url。

　　5.在 Index 方法中选择右键新建视图，专门用于显示购物清单：



　　Index.cshtml 中的代码：

[复制代码](javascript:void(0);)

@model Wen.BooksStore.WebUI.Models.CartIndexViewModel

<h2>我的购物车</h2>

<table class="table">

<thead>

<tr>

<th>书名</th>

<th>价格</th>

<th>数量</th>

<th>总计</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Cart.GetCartItems)

{

<tr>

<td>@item.Book.Name</td>

<td>@item.Book.Price</td>

<td>@item.Quantity</td>

<td>@((item.Book.Price \* item.Quantity).ToString("C"))</td>

</tr>

}

<tr>

<td> </td>

<td> </td>

<td>总计：</td>

<td>@Model.Cart.ComputeTotalValue().ToString("C")</td>

</tr>

</tbody>

</table>

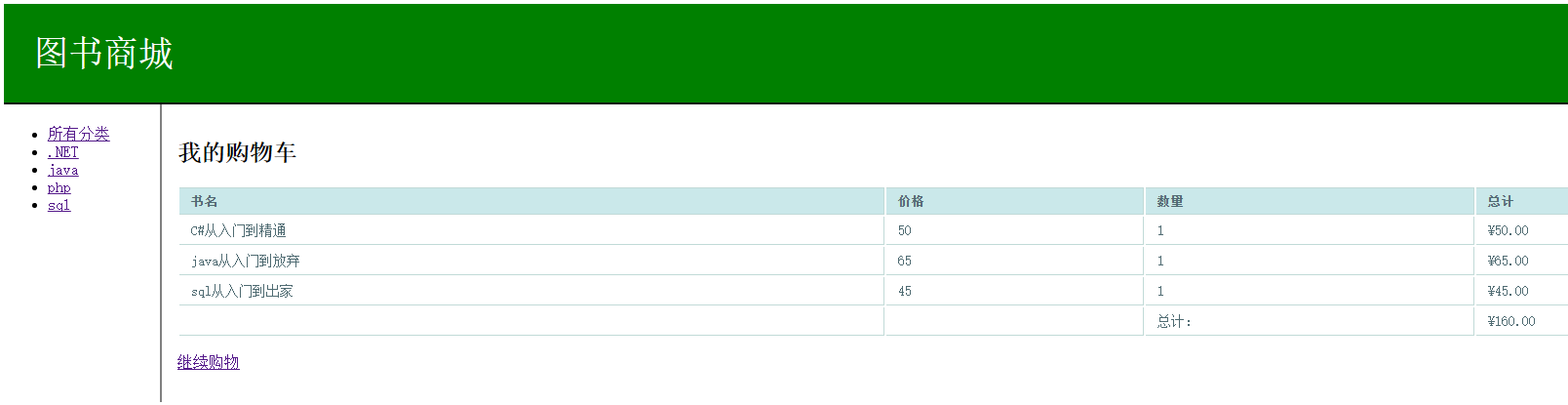
<p>

<a href="@Model.ReturnUrl">继续购物</a>

</p>

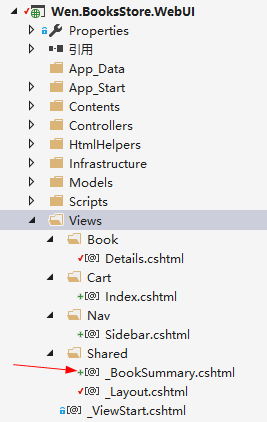
[复制代码](javascript:void(0);)

　　我想，这一定是一个令人激动的时刻，因为我们已经完成了这个基本的添加到购物车的功能。



## 三、创建一个分部视图 Partial View

　　分部视图，是嵌入在另一个视图中的一个内容片段，并且可以跨视图重用，这有助于减少重复，尤其需要在多个地方需要重复使用相同的数据时。



　　在 Shared 内部新建一个名为 \_BookSummary.cshtml 的视图，并且把之前 Details.cshtml 的代码进行整理。



　　修改后的两个视图：

　　Details.cshtml

[复制代码](javascript:void(0);)

@model Wen.BooksStore.WebUI.Models.BookDetailsViewModels

@{

ViewBag.Title = "Books";

}

@foreach (var item in Model.Books)

{

Html.RenderPartial("\_BookSummary", item);

}

<div class="pager">

@Html.PageLinks(Model, x => Url.Action("Details", new { pageIndex = x, category = Model.CurrentCategory }))

</div>

[复制代码](javascript:void(0);)

　　\_BookSummary.cshtml

[复制代码](javascript:void(0);)

@model Wen.BooksStore.Domain.Entities.Book

<div class="item">

<h3>@Model.Name</h3>

@Model.Description

<h4>@Model.Price.ToString("C")</h4>

@using (Html.BeginForm("AddToCart", "Cart"))

{

var id = Model.Id;

@Html.HiddenFor(x => id);

@Html.Hidden("returnUrl", Request.Url.PathAndQuery)

<input type="submit" value="+ 添加到购物车" />

}

<br />

<hr />

</div>

[复制代码](javascript:void(0);)

【博主】反骨仔

【原文】<http://www.cnblogs.com/liqingwen/p/6647538.html>

【参考】《精通 ASP.NET MVC ...》

上一节我们完成了两个主要功能：添加到购物车和分类导航，这一节我们会完成整个购物车的流程，以及订单处理。

　　该系列主要功能与知识点如下：

　　　　分类、产品浏览、购物车、结算、CRUD（增删改查） 管理、发邮件、分页、模型绑定、认证过滤器和单元测试等（预计剩余两篇，周三（因为周二不上班）先发布一篇）。

     【备注】项目使用 VS2015 + C#6 进行开发，有问题请发表在留言区哦，还有，页面长得比较丑，请见谅。

## 目录

* 完成购物车
* 订单结算

## 一、完成购物车

　　上一节其实已经完成了移除购物车和清空购物车的方法，只是尚未将可供用户操作的按钮放在页面区域。除了增加这两个按钮，也会在页面顶部的位置增加购物车的摘要（用于显示用户的购物总额）。

　　下面是上一节已经写好的 CartController 代码。

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

/// <summary>

/// 购物车

/// </summary>

public class CartController : Controller

{

private readonly IBookRepository \_bookRepository;

public CartController(IBookRepository bookRepository)

{

\_bookRepository = bookRepository;

}

/// <summary>

/// 首页

/// </summary>

/// <param name="returnUrl"></param>

/// <returns></returns>

public ViewResult Index(string returnUrl)

{

return View(new CartIndexViewModel()

{

Cart = GetCart(),

ReturnUrl = returnUrl

});

}

/// <summary>

/// 添加到购物车

/// </summary>

/// <param name="id"></param>

/// <param name="returnUrl"></param>

/// <returns></returns>

public RedirectToRouteResult AddToCart(int id, string returnUrl)

{

var book = \_bookRepository.Books.FirstOrDefault(x => x.Id == id);

if (book != null)

{

GetCart().AddBook(book, 1);

}

return RedirectToAction("Index", new { returnUrl });

}

/// <summary>

/// 从购物车移除

/// </summary>

/// <param name="id"></param>

/// <param name="returnUrl"></param>

/// <returns></returns>

public RedirectToRouteResult RemoveFromCart(int id, string returnUrl)

{

var book = \_bookRepository.Books.FirstOrDefault(x => x.Id == id);

if (book != null)

{

GetCart().RemoveBook(book);

}

return RedirectToAction("Index", new { returnUrl });

}

/// <summary>

/// 获取购物车

/// </summary>

/// <returns></returns>

private Cart GetCart()

{

var cart = (Cart)Session["Cart"];

if (cart != null) return cart;

cart = new Cart();

Session["Cart"] = cart;

return cart;

}

}

[复制代码](javascript:void(0);)

　　1.加入移除书籍和清空购物车的功能



http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

@model Wen.BooksStore.WebUI.Models.CartIndexViewModel

<h2>我的购物车</h2>

<table class="table">

<thead>

<tr>

<th>书名</th>

<th>价格</th>

<th>数量</th>

<th>总计</th>

<th> </th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Cart.GetCartItems)

{

<tr>

<td>@item.Book.Name</td>

<td>@item.Book.Price</td>

<td>@item.Quantity</td>

<td>@((item.Book.Price \* item.Quantity).ToString("C"))</td>

<td>

@using (Html.BeginForm("RemoveFromCart", "Cart"))

{

@Html.Hidden("id", item.Book.Id)

@Html.HiddenFor(x => x.ReturnUrl)

<input type="submit" value="- 移除" />

}

</td>

</tr>

}

<tr>

<td> </td>

<td> </td>

<td>总计：</td>

<td>@Model.Cart.ComputeTotalValue().ToString("C")</td>

<td>

@using (Html.BeginForm("Clear", "Cart"))

{

@Html.HiddenFor(x => x.ReturnUrl)

<input type="submit" value="清空购物车" />

}

</td>

</tr>

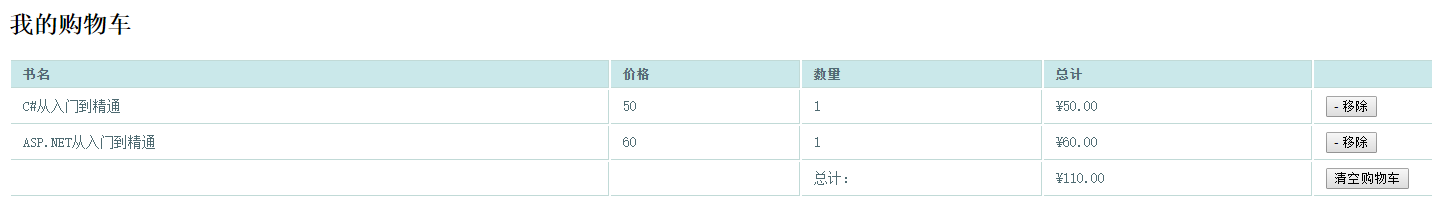
</tbody>

</table>

[复制代码](javascript:void(0);)

　　【备注】@Html.Hidden("id", item.Book.Id) 是用于生成隐藏的字段，如果直接使用 @Html.HiddenFor()，生成的 name 将会是 item.Book.Id ，将和 CartController 中 RemoveFromCart(int id, string return) 的参数不匹配。

　　显示的效果如下：



　　2.添加摘要：我们在购物车存放了许多东西，通过摘要，可以显示购物总额的缩略图，我们选择的位置在顶部右上角的一个比较显眼的位置进行显示它，当然，还需要有点击的跳转按钮方便显示所有的购物清单页面。

　　继续在 CartController 下新增一个 Action，名为 Summary，返回值是一个分部视图：

[复制代码](javascript:void(0);)

/// <summary>

/// 摘要

/// </summary>

/// <returns></returns>

public PartialViewResult Summary()

{

return PartialView(GetCart());

}

[复制代码](javascript:void(0);)

　　对应的 Summary.cshtml

[复制代码](javascript:void(0);)

@model Wen.BooksStore.Domain.Entities.Cart

<div class="bookSummary">

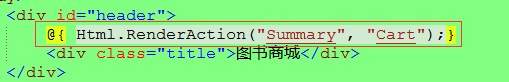
你的购物车：@Model.ComputeTotalValue()

<span>@Html.ActionLink("结算", "Checkout", "Cart", new { retunUrl = Request.Url.PathAndQuery }, null)</span>

</div>

[复制代码](javascript:void(0);)

　　对应的布局页 \_Layout.cshtml 修改的地方为：



http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

<link href="~/Contents/Site.css" rel="stylesheet" />

</head>

<body>

<div id="header">

@{ Html.RenderAction("Summary", "Cart");}

<div class="title">图书商城</div>

</div>

<div id="sideBar">

@{ Html.RenderAction("Sidebar", "Nav"); }

</div>

<div id="content">

@RenderBody()

</div>

</body>

</html>

[复制代码](javascript:void(0);)

　　添加了新的东西，css 也要进行修改：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

body {

}

#header, #content, #sideBar {

display: block;

}

#header {

background-color: green;

border-bottom: 2px solid #111;

color: White;

}

#header, .title {

font-size: 1.5em;

padding: .5em;

}

#sideBar {

float: left;

width: 8em;

padding: .3em;

}

#content {

border-left: 2px solid gray;

margin-left: 10em;

padding: 1em;

}

.pager {

text-align: right;

padding: .5em 0 0 0;

margin-top: 1em;

}

.pager A {

font-size: 1.1em;

color: #666;

padding: 0 .4em 0 .4em;

}

.pager A:hover {

background-color: Silver;

}

.pager A.selected {

background-color: #353535;

color: White;

}

.item input {

float: right;

color: White;

background-color: green;

}

.table {

width: 100%;

padding: 0;

margin: 0;

}

.table th {

font: bold 12px "Trebuchet MS", Verdana, Arial, Helvetica, sans-serif;

color: #4f6b72;

border-right: 1px solid #C1DAD7;

border-bottom: 1px solid #C1DAD7;

border-top: 1px solid #C1DAD7;

letter-spacing: 2px;

text-transform: uppercase;

text-align: left;

padding: 6px 6px 6px 12px;

background: #CAE8EA no-repeat;

}

.table td {

border-right: 1px solid #C1DAD7;

border-bottom: 1px solid #C1DAD7;

background: #fff;

font-size: 14px;

padding: 6px 6px 6px 12px;

color: #4f6b72;

}

.table td.alt {

background: #F5FAFA;

color: #797268;

}

.table th.spec, td.spec {

border-left: 1px solid #C1DAD7;

}

.bookSummary {

width: 15%;

float: right;

margin-top: 1.5%;

}

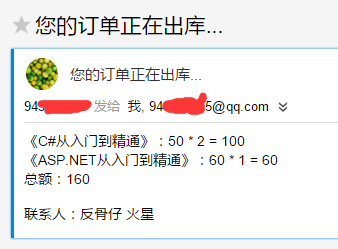
[复制代码](javascript:void(0);)



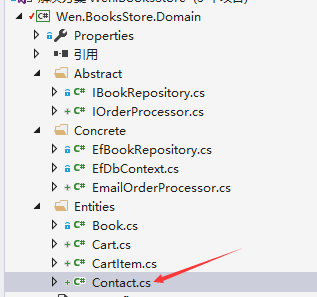
## 二、订单结算

　　购物完毕就是结算页面了，这里的订单结算并不涉及支付接口的调用，只是使用邮件的形式进行通知而已。

　　这里，我设计结算的时候需要要求用户输入一些信息，如姓名、地址和邮箱等信息，在点击确定时我再将这些输入的信息与购物清单的信息从系统的邮箱发到你所输入的邮箱当中。一个比较直观的图：



　　1.在 Entities 中添加一个域模型 Contact.cs 表示联系人的信息。



[复制代码](javascript:void(0);)

/// <summary>

/// 联系信息

/// </summary>

public class Contact

{

[Required(ErrorMessage = "姓名不能为空")]

public string Name { get; set; }

[Required(ErrorMessage = "地址不能为空")]

public string Address { get; set; }

[Required(ErrorMessage = "邮箱不能为空")]

[RegularExpression(@"(\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*\w\w)", ErrorMessage = "输入的邮箱地址不合法")]

public string Email { get; set; }

}

[复制代码](javascript:void(0);)

　　CartController.cs 添加一个用于结算的 Action:

[复制代码](javascript:void(0);)

/// <summary>

/// 结算

/// </summary>

/// <returns></returns>

public ViewResult Checkout()

{

return View(new Contact());

}

[复制代码](javascript:void(0);)

　　Checkout.cshtml 中的：

[复制代码](javascript:void(0);)

@model Wen.BooksStore.Domain.Entities.Contact

<div>

@using (Html.BeginForm())

{

<div class="error">@Html.ValidationSummary()</div>

<div>姓名： @Html.TextBoxFor(x => x.Name)</div>

<div>地址： @Html.TextBoxFor(x => x.Address)</div>

<div>邮箱： @Html.TextBoxFor(x => x.Email)</div>

<div><input type="submit" value="提交" /></div>

}

</div>

[复制代码](javascript:void(0);)

　　这里使用的是模型校验，\_Layout.cshtml 布局页需要引入 js：

<script src="~/Scripts/jquery-1.10.2.js"></script>

<script src="~/Scripts/jquery.validate.js"></script>

<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

<link href="~/Contents/Site.css" rel="stylesheet" />

<script src="~/Scripts/jquery-1.10.2.js"></script>

<script src="~/Scripts/jquery.validate.js"></script>

<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>

</head>

<body>

<div id="header">

@{ Html.RenderAction("Summary", "Cart");}

<div class="title">图书商城</div>

</div>

<div id="sideBar">

@{ Html.RenderAction("Sidebar", "Nav"); }

</div>

<div id="content">

@RenderBody()

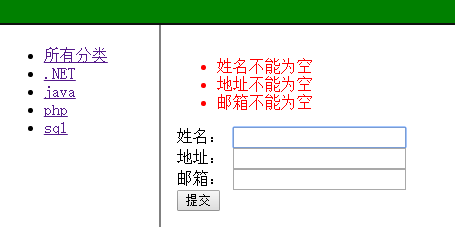
</div>

</body>

</html>

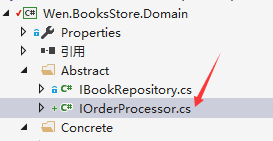
[复制代码](javascript:void(0);)

　　尝试运行，会出现以下页面，如果信息不填的话会出现相关的错误提示：



　　2.接下来，要进入“提交”后的流程了。

　　现在还需要一个组件用于处理订单，创建一个用于订单处理的接口，和一个该接口的实现，再通过 Ninject 进行两者的绑定：



[复制代码](javascript:void(0);)

/// <summary>

/// 订单处理

/// </summary>

public interface IOrderProcessor

{

/// <summary>

/// 处理订单

/// </summary>

/// <param name="cart"></param>

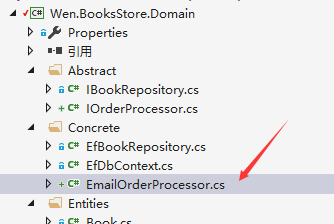
/// <param name="contact"></param>

void ProcessOrder(Cart cart, Contact contact);

}

[复制代码](javascript:void(0);)

　　建立一个实现该接口用于处理订单的实体类，这里并不是调用支付接口，而是简单通过 BCL 中的进行邮件的发送。



　　EmailOrderProcessor.cs：

[复制代码](javascript:void(0);)

/// <summary>

/// 邮件订单处理器

/// </summary>

public class EmailOrderProcessor : IOrderProcessor

{

/// <summary>

/// 发送人

/// </summary>

public static class Sender

{

/// <summary>

/// 账号

/// </summary>

public static string Account = "你的@qq.com";

/// <summary>

/// 密码

/// </summary>

public static string Password = "xxx";

}

/// <summary>

/// 处理订单

/// </summary>

/// <param name="cart"></param>

/// <param name="contact"></param>

public void ProcessOrder(Cart cart, Contact contact)

{

if (string.IsNullOrEmpty(contact.Email))

{

throw new Exception("Email 不能为空！");

}

var sb = new StringBuilder();

foreach (var item in cart.GetCartItems)

{

sb.AppendLine($"《{item.Book.Name}》：{item.Book.Price} \* {item.Quantity} = {item.Book.Price \* item.Quantity}");

}

sb.AppendLine($"总额：{cart.GetCartItems.Sum(x => x.Quantity \* x.Book.Price)}");

sb.AppendLine();

sb.AppendLine($"联系人：{contact.Name} {contact.Address}");

//设置发件人,发件人需要与设置的邮件发送服务器的邮箱一致

var fromAddr = new MailAddress(Sender.Account);

var message = new MailMessage { From = fromAddr };

//设置收件人,可添加多个,添加方法与下面的一样

message.To.Add(contact.Email);

//设置抄送人

message.CC.Add(Sender.Account);

//设置邮件标题

message.Subject = "您的订单正在出库...";

//设置邮件内容

message.Body = sb.ToString();

//设置邮件发送服务器,服务器根据你使用的邮箱而不同,可以到相应的 邮箱管理后台查看,下面是QQ的

var client = new SmtpClient("smtp.qq.com", 25)

{

Credentials = new NetworkCredential(Sender.Account, Sender.Password),

EnableSsl = true

};

//设置发送人的邮箱账号和密码

//启用ssl,也就是安全发送

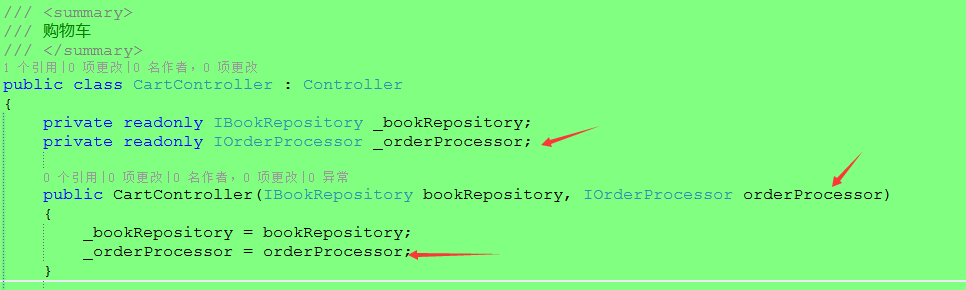
//发送邮件

client.Send(message);

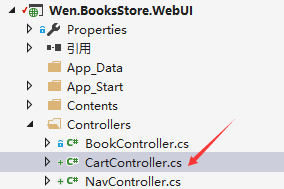
}

[复制代码](javascript:void(0);)

　　CartController 也需要稍作调整：



　　还要在 CartController 中额外添加一个带 [HttPost] 特性的名为 Checkout 方法：



[复制代码](javascript:void(0);)

/// <summary>

/// 结算

/// </summary>

/// <param name="contact"></param>

/// <returns></returns>

[HttpPost]

public ViewResult Checkout(Contact contact)

{

if (!ModelState.IsValid)

return View(contact);

var cart = GetCart();

\_orderProcessor.ProcessOrder(cart, contact);

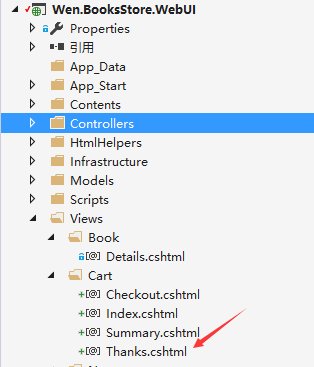
cart.Clear();

return View("Thanks");

}

[复制代码](javascript:void(0);)

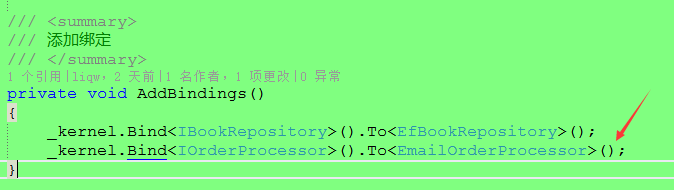
　　当校验成功时，会调用接口发一条信息，并且清空已有的购物车，然后跳转到指定的一个新视图页：



　　新建 Thanks.cshtml，内容如下：

Thanks

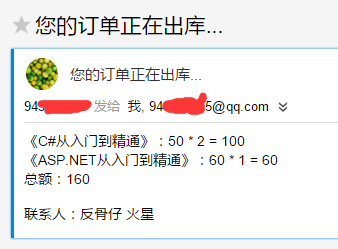
　　别忘了添加绑定哦，使用 DI 容器将两者进行绑定：



　　启动页面，试试效果吧：



 　　看来，好像成功了哦：



【博主】反骨仔

【出处】<http://www.cnblogs.com/liqingwen/p/6652564.html>

【参考】《精通 ASP.NET MVC ...》

上一节我们完成了两个主要功能：完成了整个购物车的流程，以及订单处理（发邮件进行通知），今天我们来学习一下最基本的增删改查，以及登录认证过滤器，加入防 CSRF 攻击，本系列已完结。

　　该系列主要功能与知识点如下：

　　　　分类、产品浏览、购物车、结算、CRUD（增删改查） 管理、发邮件、分页、模型绑定、认证过滤器和单元测试等。

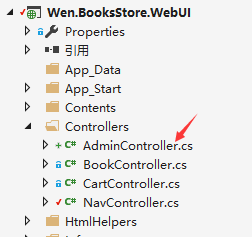
     【备注】项目使用 VS2015 + C#6 进行开发，有问题请发表在留言区哦，还有，页面长得比较丑，请见谅。

## 目录

* 基本的增删改查 CRUD
* 登录授权认证过滤

## 基本的增删改查 CRUD

　　我们创建一个新的控制器进行增删改查功能，AdminController，并添加一个显示所有数据的方法：



[复制代码](javascript:void(0);)

/// <summary>

/// 后台管理控制器

/// </summary>

public class AdminController : Controller

{

private readonly IBookRepository \_bookRepository;

public AdminController(IBookRepository bookRepository)

{

\_bookRepository = bookRepository;

}

/// <summary>

/// 首页

/// </summary>

/// <returns></returns>

public ActionResult Index()

{

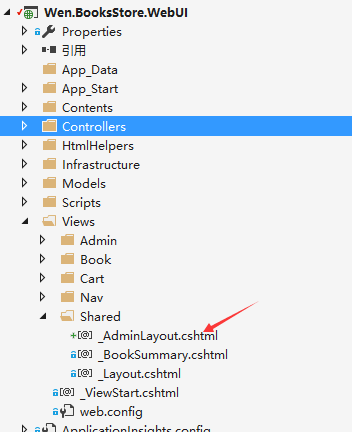
return View(\_bookRepository.Books);

}

}

[复制代码](javascript:void(0);)

　　不在沿用之前的布局页了，创建一个新的布局页 \_AdmindLayout.cshtml：



[复制代码](javascript:void(0);)

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

<link href="~/Contents/admin/Site.css" rel="stylesheet" />

</head>

<body>

<div>

@RenderBody()

</div>

</body>

</html>

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

.table {

width: 100%;

padding: 0;

margin: 0;

}

.table th {

font: bold 12px "Trebuchet MS", Verdana, Arial, Helvetica, sans-serif;

color: #4f6b72;

border-right: 1px solid #C1DAD7;

border-bottom: 1px solid #C1DAD7;

border-top: 1px solid #C1DAD7;

letter-spacing: 2px;

text-transform: uppercase;

text-align: left;

padding: 6px 6px 6px 12px;

background: #CAE8EA no-repeat;

}

.table td {

border-right: 1px solid #C1DAD7;

border-bottom: 1px solid #C1DAD7;

background: #fff;

font-size: 14px;

padding: 6px 6px 6px 12px;

color: #4f6b72;

}

.table td.alt {

background: #F5FAFA;

color: #797268;

}

.table th.spec, td.spec {

border-left: 1px solid #C1DAD7;

}

[复制代码](javascript:void(0);)

　　对应的 Index.cshtml：

[复制代码](javascript:void(0);)

@model IEnumerable<Wen.BooksStore.Domain.Entities.Book>

@{

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<p>

@Html.ActionLink("新增", "Edit")

</p>

<table class="table">

<tr>

<th>

名称

</th>

<th>

描述

</th>

<th>

价格

</th>

<th>

分类

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

@using (Html.BeginForm("Delete", "Admin", FormMethod.Post, new { style = "display:inline;" }))

{

@Html.Hidden("id", item.Id)

<input type="submit" value="删除" />

}

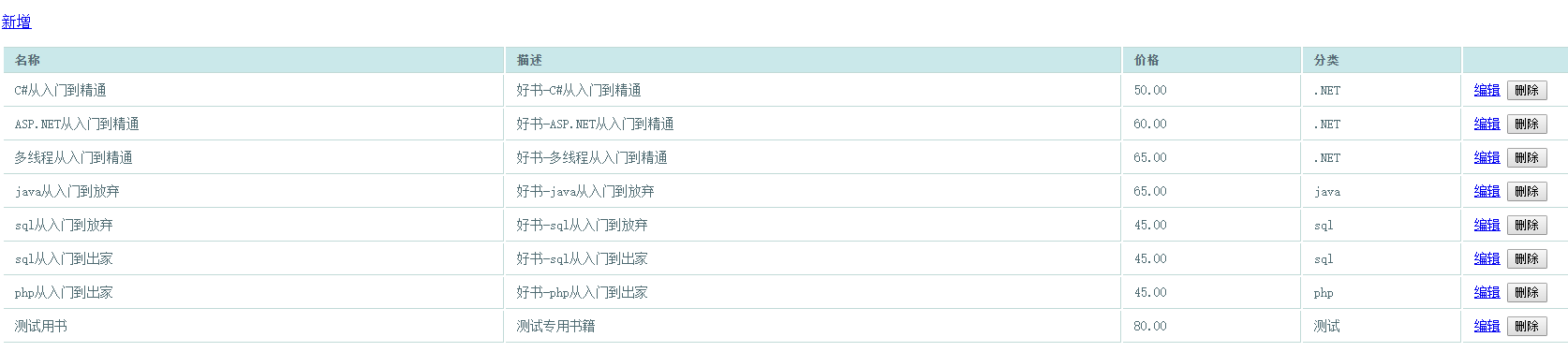
</td>

</tr>

}

</table>

[复制代码](javascript:void(0);)



　　 编辑，我把新增和编辑的位置放在一块，使用 id 进行区分，如果 id = 0 就表示新增的信息。

　　在 AdminCtroller 中添加关于编辑的方法

[复制代码](javascript:void(0);)

/// <summary>

/// 编辑

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public ActionResult Edit(int id = 0)

{

if (id == 0)

{

return View(new Book());

}

var model = \_bookRepository.Books.FirstOrDefault(x => x.Id == id);

return View(model);

}

/// <summary>

/// 编辑

/// </summary>

/// <param name="book"></param>

/// <returns></returns>

[HttpPost]

public ActionResult Edit(Book book)

{

if (!ModelState.IsValid)

{

return View(book);

}

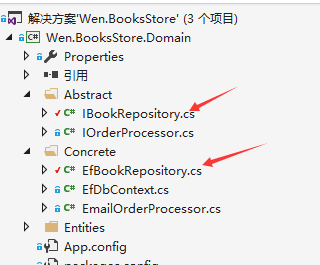
\_bookRepository.SaveBook(book);

return RedirectToAction("Index");

}

[复制代码](javascript:void(0);)

　　更新存储库中的方法：



　　IBookRepository.cs

[复制代码](javascript:void(0);)

/// <summary>

/// 书存储库接口

/// </summary>

public interface IBookRepository

{

/// <summary>

/// 书模型集合

/// </summary>

IQueryable<Book> Books { get; }

/// <summary>

/// 保存书

/// </summary>

/// <param name="book"></param>

/// <returns></returns>

int SaveBook(Book book);

/// <summary>

/// 删除书

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

Book DeleteBook(int id);

}

[复制代码](javascript:void(0);)

　　EfBookRepository.cs

[复制代码](javascript:void(0);)

/// <summary>

/// 书存储库

/// </summary>

public class EfBookRepository : IBookRepository

{

private readonly EfDbContext \_context = new EfDbContext();

/// <summary>

/// 书模型集合

/// </summary>

public IQueryable<Book> Books => \_context.Books;

/// <summary>

/// 保存书

/// </summary>

/// <param name="book"></param>

/// <returns></returns>

public int SaveBook(Book book)

{

if (book.Id == 0)

{

\_context.Books.Add(book);

}

else

{

var model = \_context.Books.Find(book.Id);

if (model==null)

{

return 0;

}

model.Category = book.Category;

model.Description = book.Description;

model.Name = book.Name;

model.Price = book.Price;

}

return \_context.SaveChanges();

}

/// <summary>

/// 删除书

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public Book DeleteBook(int id)

{

var model = \_context.Books.Find(id);

if (model == null)

{

return null;

}

\_context.Books.Remove(model);

\_context.SaveChanges();

return model;

}

}

[复制代码](javascript:void(0);)

　　需要对 Book 模型加上验证用的特性：

[复制代码](javascript:void(0);)

[Table("Book")]

public class Book

{

/// <summary>

/// 标识

/// </summary>

public int Id { get; set; }

/// <summary>

/// 名称

/// </summary>

[Required(ErrorMessage = "名称不能为空")]

public string Name { get; set; }

/// <summary>

/// 描述

/// </summary>

[Required(ErrorMessage = "描述不能为空")]

public string Description { get; set; }

/// <summary>

/// 价格

/// </summary>

[Required(ErrorMessage = "价格不能为空")]

[Range(0.01, double.MaxValue, ErrorMessage = "请填写合适的价格")]

public decimal Price { get; set; }

/// <summary>

/// 分类

/// </summary>

[Required(ErrorMessage = "分类不能为空")]

public string Category { get; set; }

}

[复制代码](javascript:void(0);)

　　\_AdminLayout.cshtml 需要引入验证用的 js（客户端验证）：

<script src="~/Scripts/jquery-1.10.2.js"></script>

<script src="~/Scripts/jquery.validate.js"></script>

<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>

　　Edit.cshtml

[复制代码](javascript:void(0);)

@model Wen.BooksStore.Domain.Entities.Book

@{

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>编辑</h2>

<div>

@Html.ValidationSummary()

<div>

@using (Html.BeginForm())

{

@Html.HiddenFor(x => x.Id)

<table>

<tr>

<td>名称</td>

<td>@Html.TextBoxFor(x => x.Name)</td>

</tr>

<tr>

<td>价格</td>

<td>@Html.TextBoxFor(x => x.Price)</td>

</tr>

<tr>

<td>分类</td>

<td>@Html.TextBoxFor(x => x.Category)</td>

</tr>

<tr>

<td>描述</td>

<td>@Html.TextAreaFor(x => x.Description)</td>

</tr>

</table>

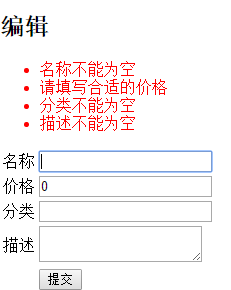
<input type="submit" value="提交" />

}

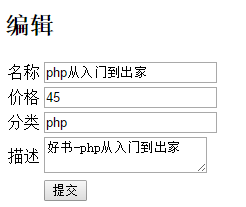
</div>

</div>

[复制代码](javascript:void(0);)



图：错误提示



　　删除

[复制代码](javascript:void(0);)

/// <summary>

/// 删除

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

[HttpPost]

public ActionResult Delete(int id)

{

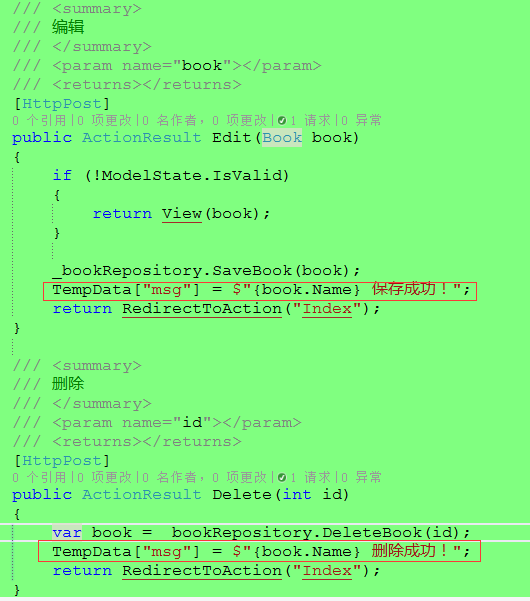
\_bookRepository.DeleteBook(id);

return RedirectToAction("Index");

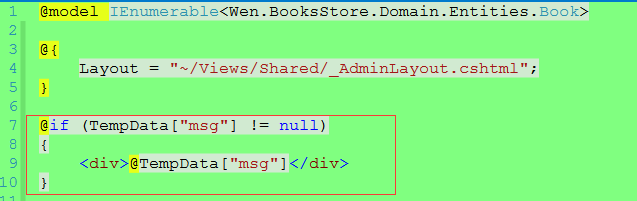
}

[复制代码](javascript:void(0);)

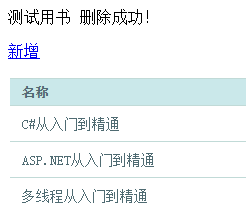
　　加入提示，我们在新增、编辑和删除时应该加入必要的提示信息，使用 TempData。



　　/Admin/Index.cshtml 下的也要添加：



　　执行效果：



　　【备注】TempData 临时数据保存了一条信息，是一个“键/值”字典，类似会话 Session 和 ViewBag，它和 Session 的差别是，在 HTTP 请求结束后会被删除。因为这里使用了 RedirectToAction ，一条重定向指令，会告诉浏览器重定向请求到一个新地址，这时就不能使用 ViewBag，ViewBag 用于在控制器与视图之间传递数据，但它保持数据的时间不能比当前的 HTTP 请求长，重定向意味着用户是跨请求的，ViewBag 不能用于跨请求时传递数据。

## 登录授权认证过滤

　　上面是一个 Admin 的后台管理操作，不是每一个用户都能够进入管理的，所以现在加入登录授权认证功能，只有成功后，才能进入管理界面。

　　先在配置文件 WebConfig.cs 中加入

[复制代码](javascript:void(0);)

<authentication mode="Forms">

<forms loginUrl="~/Account/Login" timeout="2880">

<credentials passwordFormat="Clear">

<user name="admin" password="123"/>

</credentials>

</forms>

</authentication>

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="utf-8"?>

<!--

For more information on how to configure your ASP.NET application, please visit

http://go.microsoft.com/fwlink/?LinkId=301880

-->

<configuration>

<connectionStrings>

<add name="EfDbContext" connectionString="server=.;database=TestDb;uid=sa;pwd=123" providerName="System.Data.SqlClient"/>

</connectionStrings>

<appSettings>

<add key="webpages:Version" value="3.0.0.0"/>

<add key="webpages:Enabled" value="false"/>

<add key="ClientValidationEnabled" value="true"/>

<add key="UnobtrusiveJavaScriptEnabled" value="true"/>

<add key="SendEmailName" value="943239005@qq.com"/>

</appSettings>

<system.web>

<authentication mode="Forms">

<forms loginUrl="~/Account/Login" timeout="2880">

<credentials passwordFormat="Clear">

<user name="admin" password="123"/>

</credentials>

</forms>

</authentication>

<compilation debug="true" targetFramework="4.6.1"/>

<httpRuntime targetFramework="4.6.1"/>

<httpModules>

<add name="ApplicationInsightsWebTracking" type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule, Microsoft.AI.Web"/>

</httpModules>

</system.web>

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="System.Web.Helpers" publicKeyToken="31bf3856ad364e35"/>

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.WebPages" publicKeyToken="31bf3856ad364e35"/>

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Mvc" publicKeyToken="31bf3856ad364e35"/>

<bindingRedirect oldVersion="1.0.0.0-5.2.3.0" newVersion="5.2.3.0"/>

</dependentAssembly>

</assemblyBinding>

</runtime>

<system.codedom>

<compilers>

<compiler language="c#;cs;csharp" extension=".cs"

type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"

warningLevel="4" compilerOptions="/langversion:6 /nowarn:1659;1699;1701"/>

<compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"

type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"

warningLevel="4" compilerOptions="/langversion:14 /nowarn:41008 /define:\_MYTYPE=\&quot;Web\&quot; /optionInfer+"/>

</compilers>

</system.codedom>

<system.webServer>

<validation validateIntegratedModeConfiguration="false"/>

<modules>

<remove name="ApplicationInsightsWebTracking"/>

<add name="ApplicationInsightsWebTracking" type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule, Microsoft.AI.Web"

preCondition="managedHandler"/>

</modules>

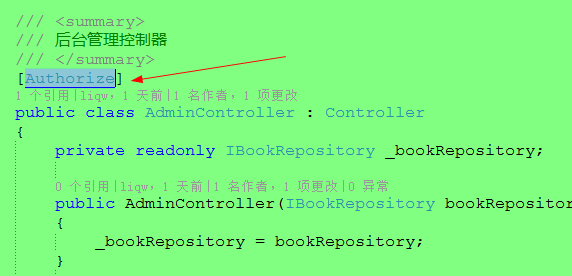
</system.webServer>

</configuration>

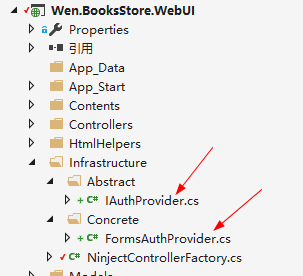
[复制代码](javascript:void(0);)

　　在这里使用的授权认证模式为表单认证，为了简化与数据库的交互操作，采取的是硬编码的形式。如果尚未得到认证，会跳转到 Account/Login 的地址让管理员先进行登录，timeout 表示登录（即认证）成功的保持时长为 2880 分钟（即 48 小时），而 name 表示的就是用户名， password 表示的就是登录密码。

　　这里采用的是授权认证过滤器，我们需要对要认证后才能进入的控制器添加一个特性[Authorize]，即对 AdminController 添加该特性。



　　新建表单认证提供器，一个接口和一个实现：



　　IAuthProvider.cs：

[复制代码](javascript:void(0);)

public interface IAuthProvider

{

/// <summary>

/// 认证

/// </summary>

/// <param name="userName"></param>

/// <param name="password"></param>

/// <returns></returns>

bool Auth(string userName, string password);

}

[复制代码](javascript:void(0);)

　　FormsAuthProvider.cs：

[复制代码](javascript:void(0);)

/// <summary>

/// 表单认证提供者

/// </summary>

public class FormsAuthProvider:IAuthProvider

{

/// <summary>

/// 认证

/// </summary>

/// <param name="userName"></param>

/// <param name="password"></param>

/// <returns></returns>

public bool Auth(string userName, string password)

{

var result = FormsAuthentication.Authenticate(userName, password);

if (result)

{

//设置认证 Cookie

FormsAuthentication.SetAuthCookie(userName, false);

}

return result;

}

}

[复制代码](javascript:void(0);)

　　AddBindings() 方法中注册：

[复制代码](javascript:void(0);)

/// <summary>

/// 添加绑定

/// </summary>

private void AddBindings()

{

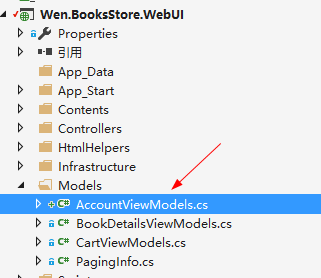
\_kernel.Bind<IBookRepository>().To<EfBookRepository>();

\_kernel.Bind<IOrderProcessor>().To<EmailOrderProcessor>();

\_kernel.Bind<IAuthProvider>().To<FormsAuthProvider>();

}

[复制代码](javascript:void(0);)



[复制代码](javascript:void(0);)

/// <summary>

/// 登录视图模型

/// </summary>

public class LoginViewModel

{

[Required(ErrorMessage = "用户名不能为空")]

public string UserName { get; set; }

[Required(ErrorMessage = "密码不能为空")]

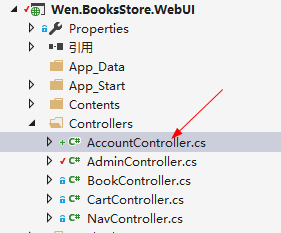
[DataType(DataType.Password)]

public string Password { get; set; }

}

[复制代码](javascript:void(0);)

　　新建 AccountController



[复制代码](javascript:void(0);)

public class AccountController : Controller

{

private readonly IAuthProvider \_authProvider;

public AccountController(IAuthProvider authProvider)

{

\_authProvider = authProvider;

}

/// <summary>

/// 登录

/// </summary>

/// <returns></returns>

public ActionResult Login()

{

return View();

}

/// <summary>

/// 登录

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Login(LoginViewModel model)

{

if (!ModelState.IsValid)

{

return View(new LoginViewModel());

}

var result = \_authProvider.Auth(model.UserName, model.Password);

if (result) return RedirectToAction("Index", "Admin");

ModelState.AddModelError("", "账号或用户名有误");

return View(new LoginViewModel());

}

}

[复制代码](javascript:void(0);)

　　Login.cshtml 登录页面：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

@model Wen.BooksStore.WebUI.Models.LoginViewModel

@{

Layout = null;

}

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>登录</title>

@\*<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css">\*@

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">

@\*<link href="~/Contents/Login/css/htmleaf-demo.css" rel="stylesheet" />\*@

<style type="text/css">

@@import url(https://fonts.googleapis.com/css?family=Roboto:300);

.login-page {

margin: auto;

padding: 8% 0 0;

width: 360px;

}

.form {

background: #FFFFFF;

box-shadow: 0 0 20px 0 rgba(0, 0, 0, 0.2), 0 5px 5px 0 rgba(0, 0, 0, 0.24);

margin: 0 auto 100px;

max-width: 360px;

padding: 45px;

position: relative;

text-align: center;

z-index: 1;

}

.form input {

background: #f2f2f2;

border: 0;

box-sizing: border-box;

font-family: "Roboto", sans-serif;

font-size: 14px;

margin: 0 0 15px;

outline: 0;

padding: 15px;

width: 100%;

}

.form button {

-webkit-transition: all 0.3 ease;

background: #4CAF50;

border: 0;

color: #FFFFFF;

cursor: pointer;

font-family: "Microsoft YaHei", "Roboto", sans-serif;

font-size: 14px;

outline: 0;

padding: 15px;

text-transform: uppercase;

transition: all 0.3 ease;

width: 100%;

}

.form button:hover, .form button:active, .form button:focus { background: #43A047; }

.form .message {

color: #b3b3b3;

font-size: 12px;

margin: 15px 0 0;

}

.form .message a {

color: #4CAF50;

text-decoration: none;

}

.form .register-form { display: none; }

.container {

margin: 0 auto;

max-width: 300px;

position: relative;

z-index: 1;

}

.container:before, .container:after {

clear: both;

content: "";

display: block;

}

.container .info {

margin: 50px auto;

text-align: center;

}

.container .info h1 {

color: #1a1a1a;

font-size: 36px;

font-weight: 300;

margin: 0 0 15px;

padding: 0;

}

.container .info span {

color: #4d4d4d;

font-size: 12px;

}

.container .info span a {

color: #000000;

text-decoration: none;

}

.container .info span .fa { color: #EF3B3A; }

body {

-moz-osx-font-smoothing: grayscale;

-webkit-font-smoothing: antialiased;

background: #76b852; /\* fallback for old browsers \*/

background: -webkit-linear-gradient(right, #76b852, #8DC26F);

background: -moz-linear-gradient(right, #76b852, #8DC26F);

background: -o-linear-gradient(right, #76b852, #8DC26F);

background: linear-gradient(to left, #76b852, #8DC26F);

font-family: "Roboto", sans-serif;

}

</style>

<!--[if IE]>

<script src="http://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.min.js"></script>

<![endif]-->

<script src="~/Scripts/jquery-1.10.2.js"></script>

<script src="~/Scripts/jquery.validate.js"></script>

<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>

</head>

<body>

<div id="wrapper" class="login-page">

<div id="login\_form" class="form">

@using (Html.BeginForm("Login", "Account", FormMethod.Post, new { @class = "login-form" }))

{

<span style="float: left; color: red;">@Html.ValidationSummary()</span>

@Html.AntiForgeryToken()

@Html.TextBoxFor(x => x.UserName, new { placeholder = "用户名" })

@Html.EditorFor(x => x.Password, new { placeholder = "密码", })

<input type="submit" value="登 录" />

}

</div>

</div>

</body>

</html>

[复制代码](javascript:void(0);)





　　【备注】ValidateAntiForgeryToken 特性用于防止跨站请求伪造（CSRF）攻击。

【博主】反骨仔

【原文】<http://www.cnblogs.com/liqingwen/p/6658975.html>

【改编与参考】《精通 ASP.NET MVC 5》