

```
//mvc 通过这个result返回视图
public ActionResult Index()
{
    //return View();//返回一个空视图
    return Content("123");//返回123
}
```

通过model来生成数据

```
//控制器下
//要引用 using myStudy.Models;
{
    public ActionResult showUserInfo()
    {
        //可以通过model,也可以从数据库取数据
        usemodel model = new usemodel();
        model.userName = "小王";
        model.age = 20;
        model.sex = "男";
        return View(model);
    }
}
//model类 usermodel.cs
{
    public class usemodel
    {

        public string userName { get; set; }
        public int age { get; set; }
        public string sex { get; set; }
    }
}
// showuserinfo的视图界面
{
    @model myStudy.Models.usemodel //什么model类 就是上面的model类 小写
    @* 申明 model小写*@

    @{
        ViewBag.Title = "showUserInfo";
    }
    <h2>showUserInfo</h2>
    <div>姓名:@Model.userName</div>    @* 引用 Model大写*@
    <p>@Model.age</p>
    <p>@Model.sex</p>
}
```

注册界面

```
//控制器下两个方法 register(注册 界面)      receive_and_show(注册界面提交表单生成的 展示界面)
{
    public ActionResult register()
    {
        return View();
    }

    //1 参数接收
    public ActionResult receive_and_show(string username,int age,string sex)
    {
        usemodel model = new usemodel();
        model.userName = username;
        model.age = age;
        model.sex = sex;
        return View(model);
    }
}

//register 视图
@{ViewBag.Title = "register";}

<h2>register</h2>

<form action="/useInfo/receive_and_show" method="post"> //提交方式为post    action 为上面的方法    一般不
<div>用户名:<input type="text" name="userName" /></div>
<div>性别:<input type="text" name="sex" /></div>
<div>年龄:<input type="text" name="age" /></div>
<input type="submit" value="提交"/>
</form>

//receive_and_show 视图
@model myStudy.Models.usemodel
@* 申明 model小写*@
@{
    ViewBag.Title = "showUserInfo";
}
<h2>showUserInfo</h2>
<div>姓名:@Model.userName</div>    @* 引用 Model大写*@
<p>@Model.age</p>
<p>@Model.sex</p>
```

图书列表界面

model添加实体数据模型

控制器代码storeController.cs

booklist显示图书列表,购书按钮转到 **order action(get请求)** , **order**界面点击提交会转到**order(post)**界面

```
public class storeController : Controller
{

    //图书列表
    public ActionResult bookList()
    {
        dbEntities dc = new dbEntities();
        List<Books> list = dc.Books.ToList<Books>();
        return View(list);
    }

    //购书
    [HttpGet] //只处理get
    public ActionResult order(int id)//传递id决定购书哪一本
    {
        dbEntities dc = new dbEntities();
        Books entry = dc.Books.SingleOrDefault(a => a.BookId == id);
        return View(entry);
    }

    //下单
    //[HttpPost] //只处理post
    public ActionResult order(int BookId, string address,int ? num)
    {
        //新增
        dbEntities dc = new dbEntities();
        Orders entry = new Orders();
        entry.BookId = BookId;
        entry.Num = num;
        entry.Address = address;
        dc.Orders.Add(entry);
        dc.SaveChanges();
        return RedirectToAction("orderList");//跳转到order这个action
    }

    //订单列表0A
    public ActionResult orderList()
    {
        dbEntities dc = new dbEntities();
        List<sv_Orders> list = dc.sv_Orders.ToList<sv_Orders>(); //sv_Orders 视图
        return View(list);
    }

}
```

视图代码

booklist (list)

```

<td>
    
</td>
<td>
    @*名字 返回的action action返回的参数*@
    @Html.ActionLink("购书", "order", new { id = item.BookId }) |

</td>

```

order (detail)

```

 url可以改成这个

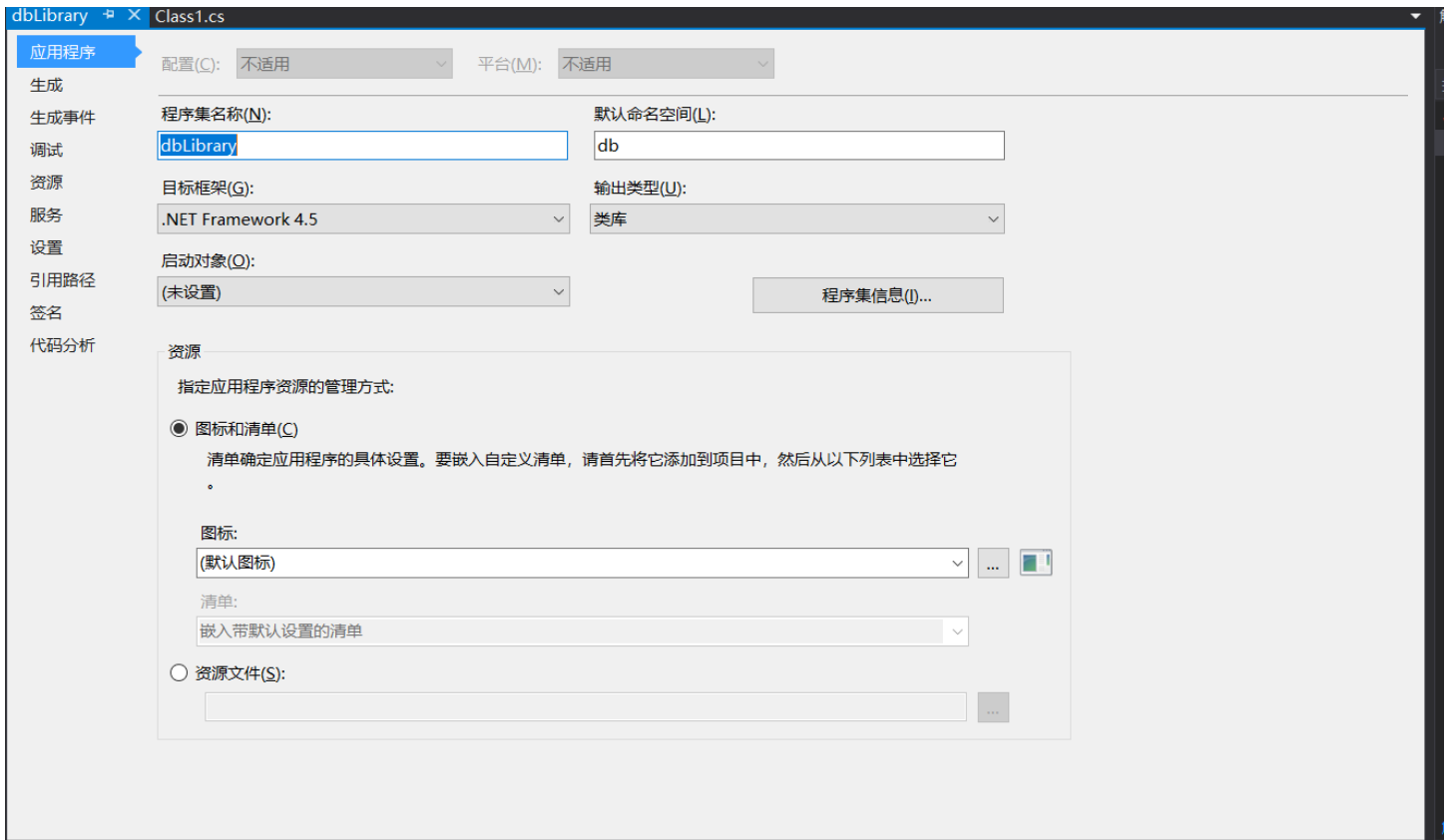
<fieldset>
    <legend>请输入下单信息</legend>
    <form method="post">
        <input type="hidden" name="BookId" value="@Model.BookId"/>  @*隐藏表单*@
        <div>数量:<input type="text" name="num"/></div>
        <div>地址:<textarea name="address"></textarea></div>
        <input type="submit" value="提交"/> //提交会通过[httppost]到 RedirectToAction("ord
    </form>
</fieldset>

```

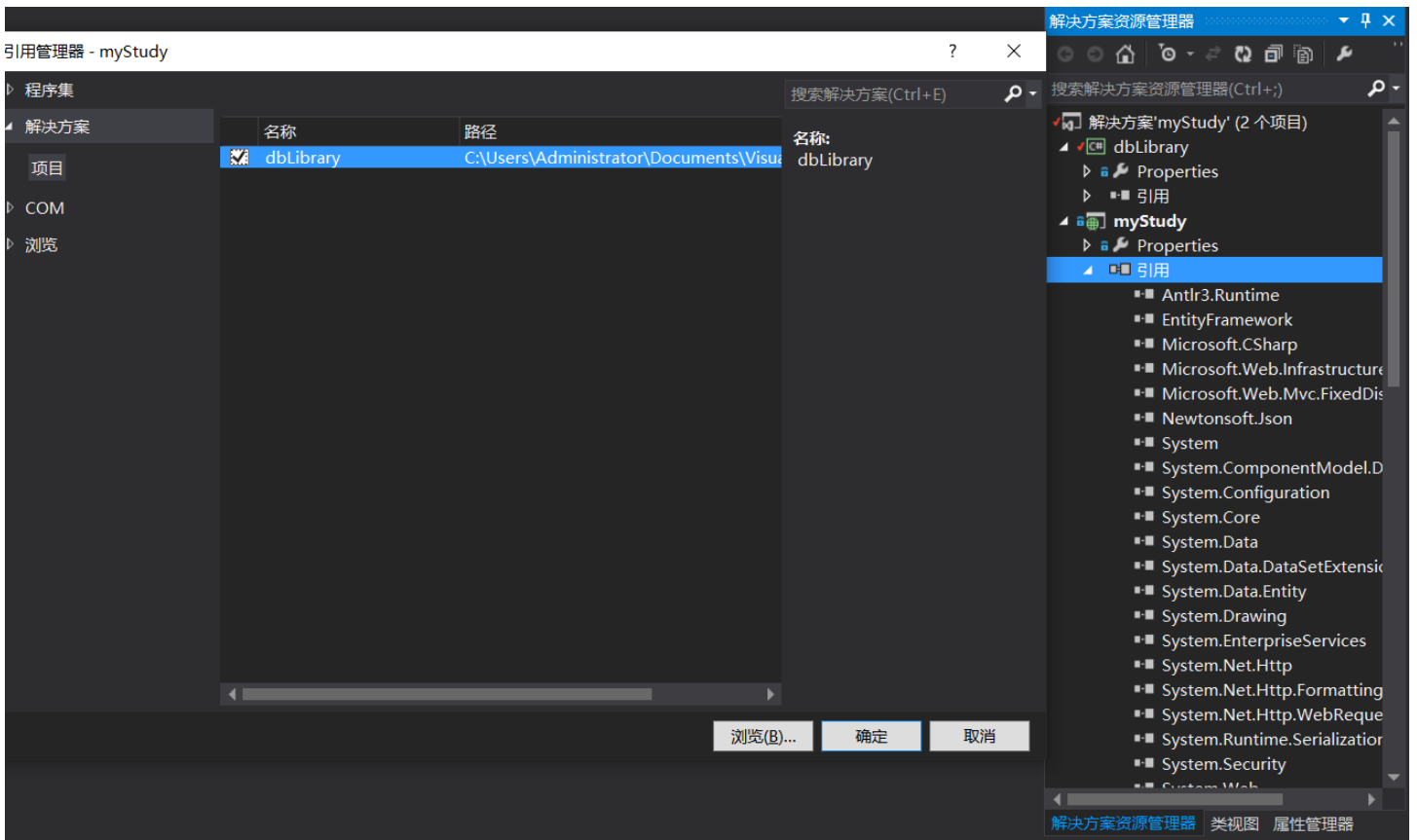
orderlist (list)

类库中EF

解决方案->新建项->类库



在dbstudy添加引用



把appconfig里的连接串替换到webconfig里

原来的view类要改变引用 原先是sv_orders -> db.sv_orders

代码全部重写到方法里,具体看(<https://github.com/codehyz/ASP.NET-MVC/blob/master/mvcmystudy02/mvcmystudy02/Controllers/storeController.cs>)

图书修改

新增

具体的方法写在db.bll.book 模型类里

```
public static void insert1(string AuthorName, string Title, Nullable<decimal> Price)
{
    dbEntities dc = new dbEntities();
    Books entry = new Books();
    entry.AuthorName = AuthorName;
    entry.Title = Title;
    entry.Price = Price;
    dc.Books.Add(entry);
    dc.SaveChanges();
}
```

- index的跳转页面的指向的action 改成Insert1

```
@model IEnumerable<db.Books>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create", "insert1")
</p>

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

- 上面的insert1会跳转到[HttpGet]的insert的action
- 然后提交信息 会跳转的[HttpPost]的insertaction

方式2(主要用这个)

```
public static void insert2(db.Books entry)
{
    dbEntities dc = new dbEntities();
    dc.Books.Add(entry);
    dc.SaveChanges();
}
```

```
public ActionResult insert2()
{
    //强类型视图必须返回对象 ,这里返回一个空对象
    db.Books entry = new db.Books();
    return View(entry);
}

[HttpPost]
0 个引用
public ActionResult insert2(db.Books entry)
{
    db.bll.book.insert2(entry);
    return RedirectToAction("Index");
}
```

修改

基于EntityFramework的数据模型 – 修改

- 基于EF数据模型的修改，方式2
 - 适应：模型对象以参数方式传入(entry)
 - 要求：所有属性都会被传入对象的属性更新
 - 创建上下文对象
dbEntities dc = new dbEntities();
 - 附加对象到上下文对象中并设置为修改状态
dc.Entry(entry).State
=System.Data.Entity.EntityState.Modified;
 - 保存上下文对象
dc.SaveChanges();

删除

基于EntityFramework的数据模型 – 删除

- 基于EF数据模型的删除

- 创建上下文对象

- ✓ **`dbEntities dc = new dbEntities();`**

- 根据主键从模型集合中查询模型对象

- ✓ **`模型类 obj = dc.模型对象集合`**

- `.SingleOrDefault(a=>a.主键==主键值);`**

- 从上下文对象中对应模型对象集合中删除模型对象

- ✓ **`dc.模型对象集合.Remove(模型对象);`**

- 保存上下文对象

- ✓ **`dc.SaveChanges();`**

过滤器

登录过滤

添加Filters文件夹 添加 myAuthAttribute.cs

引入头文件using System.Web.Mvc; //添加类 ,并且在IAuthorizationFilter找到实现的接口

```

myStudy myStudy.Filters.myAuthAttribute OnAuthorization(AuthorizationContext filterCon
4 using System. Web;
5 using System. Web. Mvc; //添加类
6 namespace myStudy. Filters
7 {
8     1 个引用
9     public class myAuthAttribute : FilterAttribute, IAuthorizationFilter
10    {
11        //登录检测
12        0 个引用
13        public void OnAuthorization(AuthorizationContext filterContext)
14        {
15            if(filterContext. HttpContext. Session["islogin"]==null)
16            {
17                //获取登录网址
18                string url = filterContext. HttpContext. Request. RawUrl;
19                //存取在filterContext里
20                filterContext. HttpContext. Session["tourl"] = url;
21                filterContext. Result = new RedirectResult("/login/Index");
22            }
23        }
24    }
25 }
26

```

添加一个登录过滤器loginController.cs

```

// GET: /login/
[HttpGet]
0 个引用
public ActionResult Index()
{
    return View();
}

[HttpPost]
0 个引用
public ActionResult Index(string Username, string passwd)
{
    if(Username=="admin"&&passwd=="123456")
    {
        Session["islogin"] = "isok";
        //判断是否是从别的界面跳转过来的
        if(Session["tourl"]!=null)
        {
            return new RedirectResult(Session["tourl"]. ToString());
        }
        return new RedirectResult("/store/bookList");
    }
    else
    {
        ViewData["msg"]="账号密码不正确";
        return View();
    }
}

```

使用方式

- Action上方加授权过滤器 [过滤器]

```

9 {
10     0 个引用
11     public class storeController : Controller
12     {
13         [myAuth]
14         //图书列表
15         0 个引用
16         public ActionResult bookList()
17         {
18             List<db.Books> list = db.bll.book.getbooks();
19             return View(list);
20         }
21     }
22 }

```

引用之前添加filters路径

- Controller上方加授权过滤器[过滤器]

```

8 namespace myStudy.Controllers
9 {
10     [myAuth]
11     0 个引用
12     public class storeController : Controller
13     {
14         [myAuth]
15         //图书列表
16         0 个引用
17         public ActionResult bookList()
18         {
19             List<db.Books> list = db.bll.book.getbooks();
20             return View(list);
21         }
22     }
23 }

```

- 注册全局过滤器

在filterconfig里添加代码

```

3 using myStudy.Filters;
4 namespace myStudy
5 {
6     1 个引用
7     public class FilterConfig
8     {
9         1 个引用
10        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
11        {
12            filters.Add(new HandleErrorAttribute());
13            filters.Add(new myAuthAttribute()); //直接写会导致 login页面的无限循环,要排除login页面
14        }
15    }
16 }

```

为了避免无限循环

- 方法一

```

// 登录检测
0 个引用
public void OnAuthorization(AuthorizationContext filterContext)
{
    //方法一
    //myStudy.Controllers.storeController
    string controller = filterContext.Controller.ToString();
    //"bookList"
    string actionName = filterContext.ActionDescriptor.ActionName;

    if (controller == "myStudy.Controllers.loginController")
        return; //打破循环

    if (filterContext.HttpContext.Session["islogin"] == null)
    {
        //获取登录网址
    }
}

```

- 方法二

2) 将过滤器应用到代码中 //10-13

- 方法2: 在Action或者Controller上方增加 **[AllowAnonymous]** 特性, 过滤器中判断上方是否有该特性, 如果有则不进行登录检测。

```

//如果在Controller或者Action上加了 AllowAnonymous 特性, 则不进行登录和授权控制
var actionAnonymous = filterContext.ActionDescriptor
    .GetCustomAttributes(typeof(AllowAnonymousAttribute), true)
    as IEnumerable<AllowAnonymousAttribute>;
var controllerAnonymous = filterContext.Controller.GetType()
    .GetCustomAttributes(typeof(AllowAnonymousAttribute), true)
    as IEnumerable<AllowAnonymousAttribute>;
if ((actionAnonymous != null && actionAnonymous.Any())
    || (controllerAnonymous != null && controllerAnonymous.Any()))
{
    return;
}

```

动作结果过滤器

```
namespace myStudy.Filters
{
    1 个引用
    public class myActionAttribute : FilterAttribute, IResultFilter, IActionFilter
    {
        private Stopwatch timer;
        0 个引用
        void IResultFilter.OnResultExecuting(ResultExecutingContext filterContext)
        {

        }
        0 个引用
        void IResultFilter.OnResultExecuted(ResultExecutedContext filterContext)
        {
            timer.Stop();
            filterContext.HttpContext.Response.Write("时间:" + timer.Elapsed.TotalMilliseconds + "ms");
        }

        0 个引用
        public void OnActionExecuted(ActionExecutedContext filterContext)
        {

        }
    }
    0 个引用
}
```

在全局应用

```
public class FilterConfig
{
    1 个引用
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new HandleErrorAttribute());
        filters.Add(new myAuthAttribute()); // 直接写会导致 login 页面的无限循环, 要排除 login 页面
        filters.Add(new myActionAttribute());
    }
}
```

缓存过滤器

结果过滤器 – 页面输出缓存

• 相关属性介绍

- Duration 缓存内容过期时间(秒),必须的
- VaryByParam 通过客户端请求参数的不同值来缓存多份数据
- VaryByHeader 通过RequestHeader的参数的不同取值来缓存多份数据
- CacheProfile 将缓存配置写到配置文件中, 引用配置文件的配置属性。
- SqlDependency 数据库缓存, 缓存依赖的表名

```
// GET: /book/  
[OutputCache(Duration=60)]  
0 个引用  
public ActionResult Index()  
{  
    List<db.Books> list = db.bll.book.getbooks();  
    return View(list);  
}  
  
[HttpPost]
```

```
[OutputCache(Duration=10, VaryByParam="id")]  
0 个引用  
public ActionResult detail(int id)  
{  
    db.Books entry = db.bll.book.detail(id);  
    return View(entry);  
}  
  
|
```

从配置文件获取

webconfig中添加如下代码

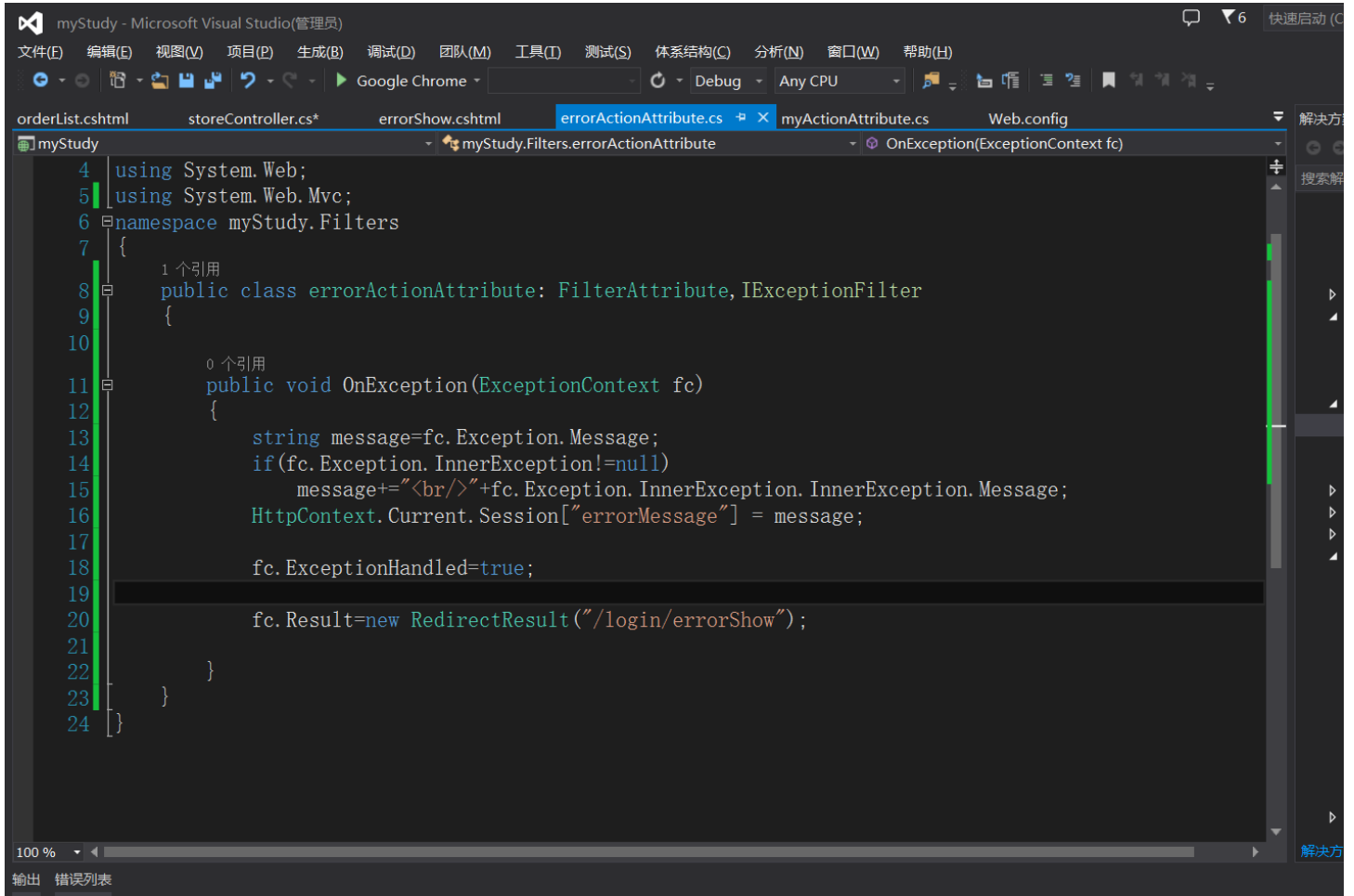
```
</appSettings>
<system.web>
  <compilation>
    <outputCacheSettings>
      <outputCacheProfiles>
        <add name="mycache" duration="20" varyByParam="id"/>
      </outputCacheProfiles>
    </outputCacheSettings>
  </compilation>
</system.web>
```

```
[OutputCache(CacheProfile="mycache")]
0 个引用
public ActionResult detail(int id)
{
    db.Books entry = db.bll.book.detail(id);
    return View(entry);
}
```

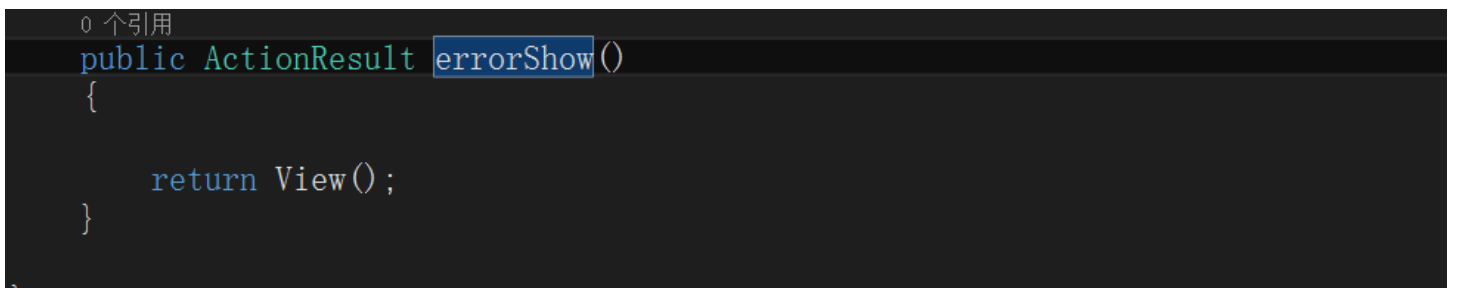
数据库缓存太麻烦了-

异常过滤器

添加一个errorActionAttribute,并且实现里面的接口



因为要跳转到/login/errorShow,所以添加一个在login控制器里添加errorShow的action



添加一个空视图



并且应用在全局


```
1 using System.Web;
2 using System.Web.Mvc;
3 using myStudy.Filters;
4 namespace myStudy
5 {
6     1 个引用
6     public class FilterConfig
7     {
8         1 个引用
8         public static void RegisterGlobalFilters(GlobalFilterCollection filters)
9         {
10             filters.Add(new HandleErrorAttribute());
11             filters.Add(new myAuthAttribute()); //直接写会导致 login页面的无限循环, 要排除login页面
12             filters.Add(new myActionAttribute());
13             filters.Add(new errorActionAttribute());
14         }
15     }
16 }
```

可以随便拿个action试一试

```
//订单列别OA
0 个引用
public ActionResult orderList()
{
    //int a = 3, b=0;
    //a /= b;
    List<db.sv_Orders> list = db.bll.order.getorder();
    return View(list);
}
```

其他过滤器

github.com/codehzy/mvc

view

viewresult

```
public ActionResult Index() //新建一个视图
{
    return View(); //默认跳转到相同名字的视图
}
```

```

        public ActionResult Index()//新建一个index2视图
    {
        //可以和acion名字不同
        return View("Index2");//输入ActionResult/Index返回到index2
    }

```

partitionviewresult

分布视图,要在index2上显示售价20以上的图书和售价20以下的图书

- 控制器写两个partitionviewResult,分别返回两个list,实现代码在业务类里

```

//actionesultcontroller
    //不允许单独运行
    [ChildActionOnly]
    //partication
    public PartialViewResult BookList1()//只会返回html代码不会返回整个网页
    {
        List<db.Books> list = db.bll.book.get_books_greater_20();
        return PartialView(list);
    }
    [ChildActionOnly]
    public PartialViewResult BookList2()
    {
        List<db.Books> list = db.bll.book.get_books_less_20();
        return PartialView(list);
    }

```

```

//book.cs
    public static List<db.Books> get_books_greater_20()
    {
        dbEntities dc = new dbEntities();
        List<db.Books> list = dc.Books.Where(a => a.Price >= 20).ToList();
        return list;
    }

    public static List<db.Books> get_books_less_20()
    {
        dbEntities dc = new dbEntities();
        List<db.Books> list = dc.Books.Where(a => a.Price <= 20).ToList();
        return list;
    }

```

- 为booklist1和booklist2创建两个list视图,记得创建分布式图

添加视图

视图名称(N):

BookList2

视图引擎(E):

Razor (CSHTML)

☒ 创建强类型视图(S)

模型类(M):

Books (db)

支架模板(E):

List

☒ 引用脚本库(R)

☒ 创建为分部视图(C)

☒ 使用布局或母版页(U):

...

(如果在 Razor _viewstart 文件中设置了此选项，则留空)

ContentPlaceHolder ID(H):

MainContent

添加(A)

取消

- index里面引用分布式图

```
//index2.html
@{
    ViewBag.Title = "index2";
    Layout = null;//viewresult 默认套用模板页 可以手动指定不用
}
```

<h2>图书展示</h2>

```
<fieldset>
    <legend>20元以内的书</legend>
    @*嵌入一个action执行结果*@
    @Html.Action("BookList1")
</fieldset>
```

```
<fieldset>
    <legend>20元以外的书</legend>
    @Html.Action("BookList2")
</fieldset>
```

```
<fieldset>
    <legend>返回结果测试</legend>
    <div>
        <a href="/ActionResult/download">下达</a>
    </div>
</fieldset>
```

跳转返回结果

- 用路径跳转
- 用action跳转

```
public ActionResult Index()
{
    //方式一
    return new RedirectResult("/Actionesult/bookList");
    //方式二
    return RedirectToAction("bookList", "ActionResult");
}
```

contentResult

返回文本内容

action写入 content action

```

public ActionResult css(string color)
{
    if (color == "red")
    {
        return Content("body{color:red}", "text/css");
    }
    if (color == "blue")
    {
        return Content("body{color:blue}", "text/css");
    }
    return Content("body{color:black}", "text/css");
}

```

母版页加代码

```

//_Layout.cshtml
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
    <link type="text/css" rel="stylesheet" href="/ActionResult/css?color=blue" />
</head>

```

fileResult

写action

```

public FileResult download()
{
    return File(Server.MapPath("~/test.txt"), "text/plain", "我的文件.txt");
}

```

index加代码

```

<fieldset>
    <legend>返回结果测试</legend>
    <div>
        <a href="/ActionResult/download">下达</a>
    </div>
</fieldset>

```

jsonresult

```
public JsonResult json1()
{
    var person = new
    {
        Name = "小王",
        Age = 12,
        parent = new[]{
            new {Name = "小王父亲",Age=40},
            new {Name = "小王母亲",Age=41},
        }
    };

    return Json(person, JsonRequestBehavior.AllowGet);
}

public JsonResult json2()
{
    List<db.Books> list = db.bll.book.getbooks();

    return Json(list, JsonRequestBehavior.AllowGet);
}
```

直接访问ActionResult 可以看到结果

javascriptResult

控制器写入jsaction

```
public JavaScriptResult js()
{
    return JavaScript("alert('1233oqiwe')");
}
```

模板页加入代码


```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
    @Scripts.Render("~/bundles/jquery")    把jquery提前
    <link type="text/css" rel="stylesheet" href="/ActionResult/css?color=blue" />
    <script type="text/javascript">
        $.getScript("/ActionResult/js");
    </script>
</head>
<body>
    @RenderBody()

    @RenderSection("scripts", required: false)
</body>
</html>

```

注释掉所有的过滤器(计时时间影响输出结果)



```

1 个引用
public class FilterConfig
{
    1 个引用
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new HandleErrorAttribute());
        //filters.Add(new myAuthAttribute()); //直接写会导致 login页面的无限循环, 要排除login页面
        //filters.Add(new myActionAttribute());
        //filters.Add(new errorActionAttribute());
    }
}

```

表单

创建表单

创建viewactioncontroller

一般的form写法:

```
<form action="/Search/SearchAlbum" method="get">  
  <input type="text" name="albumName" />  
  <input type="submit" value="搜索" />  
</form>
```

第二种用代码写表单

用@html.beginform创建一个Form容器

在index.chtml中填写代码

Html辅助方法 - 输入类辅助方法

- @Html.TextBox 生成单行文本框。
name表单名称。
format用来设定文本的展示格式，同
 string.Format的参数
- @Html.TextArea 生成多行文本框。
- @Html.Password 生成密码框
- @Html.Hidden生成隐藏表单
- @Html.RadioButton 生成单选按钮
- @Html.CheckBox 生成复选按钮

