Asp.net MVC 学习

1. C#知识点提要

1.1 特性（类似于java的注解）

例如 当使用nunit 进行测试的时候，测试的类上要用

[Testfixture]

Public class test{

//测试方法上要使用[Test]

[Test]

Public void test(){}

}

1.2 自动属性

在java中写bean的时候，写get,set方法要具体实现，而在c#中，可以直接通过

Public string Name{get;set;} 直接完成对属性的读写方法。

但是，记住，不能只写get或者set方法，必须要一起写，且get,set前面可以加权限，如 private set

Public class userInfo｛

public string name{set;get;}

public string password{set;get;}

｝

1.3 对象和集合的初始化器

对于上面那个model，我们采用对象初始化器写法

UserInfo user=new UserInfo{

Name=”zk”;

Password=”123”

}; //创建并初始化了一个实例

List<UserInfo> list=new List<UserInfo> {

new UserInfo{name=”zz”,password=”123”}

new UserInfo{name=”zzk”,password=”123”}

}

1.4 c#中委托的介绍

委托是一个类，它定义了方法的类型，使得可以将方法当作另一个方法的参数

只有当方法的签名，也就是方法的返回类型、参数类型等和委托类型一致时，才能对它进行封装。

来进行传递。

一个委托可以进行多个方法的绑定！

|  |
| --- |
| GreetingDelegate delegate1 = new GreetingDelegate(EnglishGreeting); delegate1 += ChineseGreeting;   // 给此委托变量再绑定一个方法 |

|  |
| --- |
| 例如：1.建立一个委托。  delegate int GuangChaoshi(int a);  2.定义一个方法  public static int JieZhang(int a)  {  return a + 10;  }  3.使用  static void Main(string[] args)  {  把这个方法传给委托，为什么可以这样做？因为委托的参数为int ，这个方法的返回值也是int  GuangChaoshi gwl = JieZhang;  Console.WriteLine(gwl(10) + ""); //打印20，委托的应用  Console.ReadKey();  } |

委托的特点：

\* 委托类似于c++中的函数指针，但是类型安全的

\* 委托允许将方法作为参数传递。

委托的声明：

\* delegate: 常用的一种声明

Delegate至少0个参数，至多32个参数，可以无返回值，也可以指定返回值类型

　　Func可以接受0个至16个传入参数，必须具有返回值 （Func<参数1,参数2,….参数16>）最后一个参数为Func的返回值。

　　Action可以接受0个至16个传入参数，无返回值

　　Predicate只能接受一个传入参数，返回值为bool类型

1.5 Lambda 表达式

Lambda表达式的格式如下：(实际就是一个匿名函数)

Lambda很典型的作用是和委托的结合使用，因为委托就是用来接收方法的，而Lambda就是创建一个匿名方法的

例子：

|  |
| --- |
| 1. **delegate** **int** GuangChaoshi(**int** a); 2. **static** **void** Main(**string**[] args) 3. { 4. // GuangChaoshi gwl = JieZhang; 5. GuangChaoshi gwl = p => p + 10;   //委托 lambda 两者的结合 6. Console.WriteLine(gwl(10) + "");   //打印20，表达式的应用 7. Console.ReadKey(); 8. } |

（参数列表）=> 表达式或者语句块

示例代码：

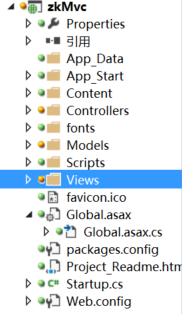
1. (x, y) => x \* y         //多参数，隐式类型=> 表达式
2. x => x \* 5              //单参数， 隐式类型=>表达式
3. x => { return x \* 5; }      //单参数，隐式类型=>语句块
4. (int x) => x \* 5            //单参数，显式类型=>表达式
5. (int x) => { return x \* 5; }      //单参数，显式类型=>语句块
6. () => Console.WriteLine()   //无参数

1.6 linq 语言集成查询

类似于 流运算 和sql

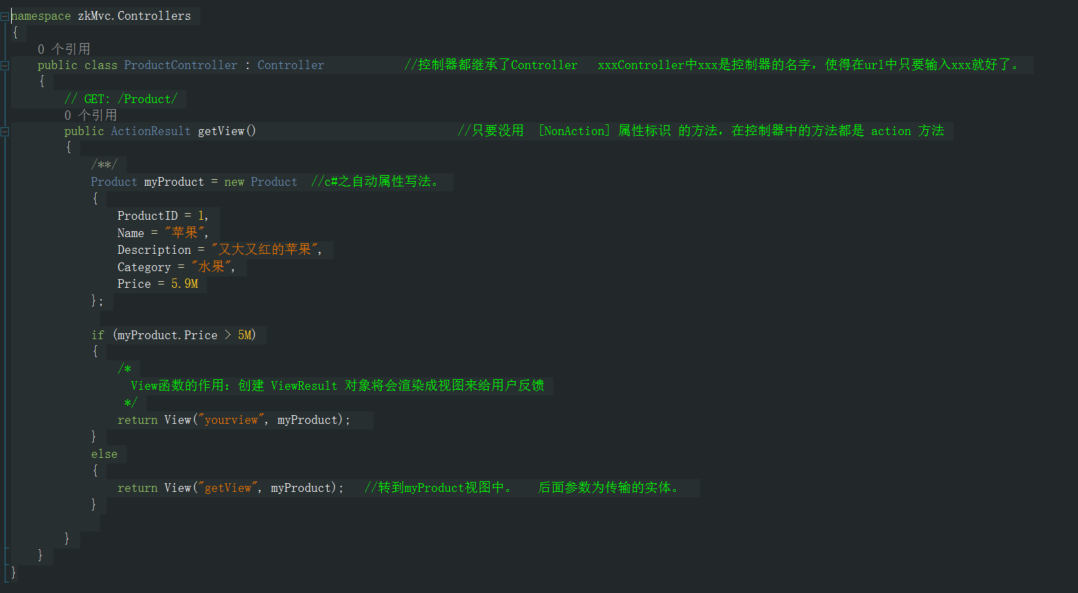
二． 基础MVC学习

2.1 controller 控制器 和 view视图 以及 model传递。



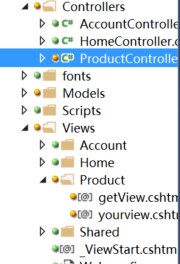
在创建一个mvc空项目后，可以看到上述目录。有一个Controller文件夹，把所有xxxcontroller.cs都放在这个文件夹里。xxx为控制器名字。在相应的控制器中创建视图文件view，就会在Views中创建相应的xxx文件夹，然后在其中添加相应的视图文件。

如



相应的文件排布如图：

上述错误：不是转到myProduct视图，而是转到getView视图

可以看到views文件夹中存在一个Product文件夹，里面有上述代码。

Q:那么，是否 视图view 就只能给相应的控制器呢？

A:如果放在相应的文件夹内，那么就只能给以文件夹名字为控制器名的控制器对应。但如果多个控制器要使用一个视图，那么就可以把视图放在Shared文件夹中。

对于一个action方法中，可能根据条件返回不同的视图，那么相应视图的

浏览url也需要改动成： 控制器名/action方法名

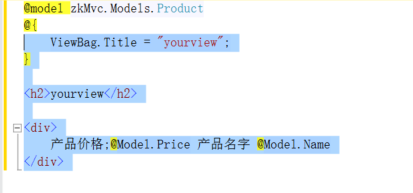


从上面代码可以看到

return View(“视图名”,对象实体);

Q：那么视图view cshtml确实的获取到了你传递的对象实体，可是它不能确定你传递的对象的类型？那怎么办呢？

A：通过@model zkMvc.Models.Product 来明确该试图的model类型。

 [使用方法就是@Model.属性](mailto:使用方法就是@Model.属性)

2.2 razor符号 即： @

虽然网上说了一大堆，但是我觉得，就是在 html里写c#代码前需要加的一个标志符。

具体用法：

1.引入css文件或者js文件，或者其他文件

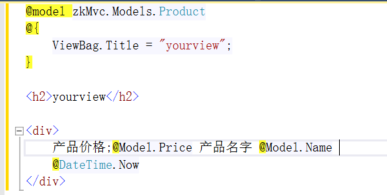
1. @Styles.Render("~/Css")
2. @Scripts.Render("~/bundles/modernizr")
3. @Scripts.Render("~/bundles/jquery")

第一条的作用是引入了一个捆绑的css文件，第二个引入一个捆绑的modernizr文件，modernizr是一个javascript文件库，它主要作用是为了兼容各类浏览器之间的差异。第三条语句引入一个捆绑的jquery文件。第四条语句@renderSection的作用是在模板页中占用一个位置，这个位置的名字就叫做scripts，然后让用户在子页面中去呈现这个叫做scripts的Section。false的意思是在子页面中并不要求一定要实现这个Section的呈现。

但实际上，你是找不到这些文件的，因为它只是一个虚拟路径，通过App\_Start中的BundleConfig.cs配置的

|  |
| --- |
| public static void RegisterBundles(BundleCollection bundles)  {  bundles.Add(new ScriptBundle("~/bundles/jquery").Include(  "~/Scripts/jquery-{version}.js"));  bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(  "~/Scripts/jquery.validate\*"));  // 使用要用于开发和学习的 Modernizr 的开发版本。然后，当你做好  // 生产准备时，请使用 http://modernizr.com 上的生成工具来仅选择所需的测试。  bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(  "~/Scripts/modernizr-\*"));  bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(  "~/Scripts/bootstrap.js",  "~/Scripts/respond.js"));  bundles.Add(new StyleBundle("~/Content/css").Include(  "~/Content/bootstrap.css",  "~/Content/site.css"));  } |

2. 调用model对象或者其他有权向可以调用的对象



第一行是通过@model 来告诉视图其中的类型。

@Model.Name 是调用model对象。

@DateTime 是调用其他对像。

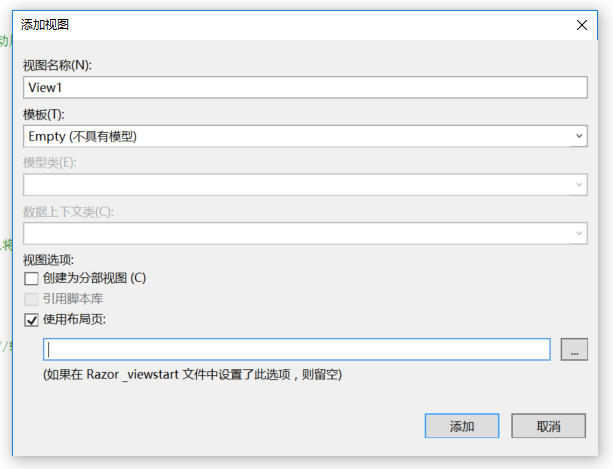
2.3 母版视图

一个网站，多个页面总有一些视图是重复的，比如公司的logo之类的：

屏幕剪辑

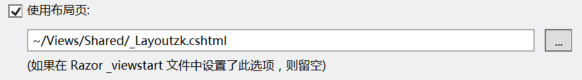
Q:那么原理是什么呢？

A:在创建视图的时候,如果使用布局页，意思就是，使用母版视图：

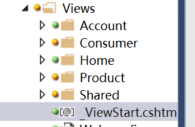


这里如果什么都不填，就会使用默认母版视图。

当然也可以使用自己自定义的布局页：



那么如果需要使得默认视图页为自己设置的。那么只要在：

 views中的\_ViewStart.cshtml改变

里面只有这一行代码：

|  |
| --- |
| @{  Layout = "~/Views/Shared/\_Layout.cshtml";  } |

只要把地址改成自己的布局页就可以了。

原理是：

项目加载的时候，会最先加载\_ViewStart.cshtml 因此不设置的话，那么所有的view的Layout都会等于"~/Views/Shared/\_Layout.cshtml" 。

2.4 数据传输以及分页视图的思考

Q：我们可以看到前面action方法返回的时候，除了带视图，还带了一个对象。

然后在对应视图通过@model 来确定传进来的对象的类型。很明显，我们并没有看到前面有提到，return View(视图名，对象名)中参数可以扩展的，即，只能传递一个对象，作为视图相关联对象，那么如果一个视图中要展示数据库中取出的多个类型的model，怎么办？

A：实际上，ASP MVC中规定了，一张数据表对应一个model，一个视图对应一个viewModel。但是并没有说，一个viewModel就是一个model啊。来举个实际例子感受一下：

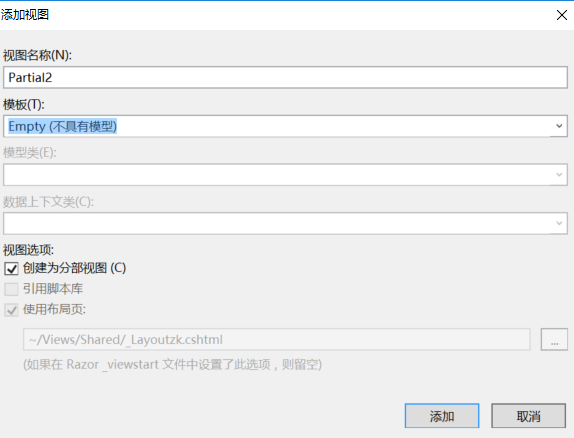
假设，数据库中有一个user表和artic表，都需要在一个视图上显示出来，那么我们可以创建一个viewModel类

IndexData 用来包含 user 和 artic 就可以解决了。

|  |
| --- |
| public class IndexData  {  public IEnumerable<Information> User { get; set; }  public IEnumerable<Article> Article { get; set; }  public IndexData()  {  Entities db = new Entities();  this.Information = db.User.ToList();  this.Article = db.Article.ToList();  }  } |

Q：在html页面上，有的地区，需要被多个views使用，但是又不是所有的views使用，因此不能用母版视图来搞定。那要怎么办呢？

A：可以创建一个分页布局



创建后用下面代码在主视图合适的地方引用：

@Html.Partial("\_LoginPartial")，

三. MVC控件Html学习

传统的Html元素不能和服务端数据进行绑定 HtmlHelper类提供了一系列的方法来生成Html元素 并可以实现与数据绑定在一起 然后生成Html

3.1 Html.BeginForm

**Html.BeginForm（actionName , controllerName , FormMethod , htmlAttribute）**

创建一个表单

**actionName 和 controllerName**

这两个参数表示表单要提交到哪个controllerName下的哪个Action方法中

**FormMethod**

此参数是一个枚举 表示表单提交方式  GET or POST

**htmlAttribute**

表示form元素的Html属性 是一个object对象 使用new {html属性名字="值"} class也是html属性 但同时它是C#关键字 只能这样指定：new {@class="formstyle"}

**Html.EndForm（）**

表示表单结束 如

1 @Html.BeginForm("index", "default", FormMethod.Post, new {id="form1"})

2 <input type="text" />

3 @{Html.EndForm();}

BeginForm方法返回System.Web.Mvc.Html.MvcForm类型 如果不用中括号括起来 则表示要输出该方法返回的值 EndForm无返回值 所以也需要使用中括号括起来 最后一个参数通过objecr来指定它的Html属性

3.2 Html.Raw()

**Html.Raw()**

@Html.Raw() 方法输出带有html标签的字符串，如：  
@Html.Raw("<div style='color:red'>输出字符串</div>")  
结果：**输出字符串**

3.3 Html.RadioButton

**Html.RadioButton（name,value,Ischecked,htmlAttribute）**

创建单选按钮

**name**

单选按钮的name

**value**

单选按钮的值

**Ischecked**

是否是选中状态

1 @{Html.BeginForm("index", "default", FormMethod.Post);}

2 男人 @Html.RadioButton("radiobtn","man",true,new{@class="radioStyle"})

3 女人 @Html.RadioButton("radiobtn","woman",false,new{@class="radioStyle"})

4 @{Html.EndForm();}

**Html.RadioButtonFor（expression,value,htmlAttribute）**

同样是创建单选按钮的方法 但+For后缀的方法可以使用强类型作为参数expression的参数 推导出类型的属性 它可以将类型的属性名字作为表单元素的name的值

**expression**

类型为System.Linq.Express.Expression.<Fun<dynamic,Tproperty>>的表达式  如

1 @model Course.Models.Employee

2 @{Html.BeginForm("index", "default", FormMethod.Post);}

3 北京 @Html.RadioButtonFor(n=>n.Address,"北京",new{@class="radioStyle",@checked="checked"})

4 上海 @Html.RadioButtonFor(n=>n.Address,"上海",new{@class="radioStyle"})

5 @{Html.EndForm();}

**~~Html.CheckBox（）~~**

创建复选框

不建议使用此方法来创建复选框 服务端不好获取值 请直接使用input 注意每个复选框需要有value值 否则获取的选中的复选框的值永远是on

1 <input type="checkbox" class="Book" value="1" name="Books" checked="checked"/>民谣<br/>

2 <input type="checkbox" class="Book" value="2" name="Books" checked="checked"/>电子<br/>

3 <input type="checkbox" class="Book" value="3" name="Books" />低保真<br/>

在Action中这样获取

1 public ActionResult Editor(int[] Books)

2 {

3 foreach (var item in Books)

4 {

5 }

6 }

3.4 HtmlDropDownList

**Html.DropDownList（name , selectList , defaultSelected,htmlAttribute）**

创建下拉选项

**name**

下拉选项的name

**selectList**

一个IEnumerable<SelectListItem>集合 集合中的每个选项是SelectListItem类型的 我们可以在Action中创建实现了IEnumerable<selectListItem>接口的集合 然后将集合作为此方法的第二个参数 如

复制代码

1 public ActionResult Index()

2 {

3 List<SelectListItem> itemList = new List<SelectListItem>

4 {

5 new SelectListItem{Text="荔枝",Value="荔枝", Selected=false},

6 new SelectListItem{Text="番茄",Value="番茄",Selected=false},

7 new SelectListItem{Text="芒果",Value="芒果",Selected=false}

8 };

9 ViewData["choose"] = itemList;

10 return View();

11 }

复制代码

在视图中

1 @{Html.BeginForm("index", "default", FormMethod.Post);}

2 @Html.DropDownList("choose",ViewData["choose"] as IEnumerable<SelectListItem>,new{@class="selectStyle"})

3 @{Html.EndForm();}

**defaultSelected**

一个文本 表示默认选择的项 可选

1 @Html.DropDownList("choose",ViewData["choose"] as IEnumerable<SelectListItem>,"**请选择**",new{@class="selectStyle"})

使用Linq创建下拉选项集合 如

复制代码

1 public ActionResult Index()

2 {

3 List<Employee> empList = new List<Employee>

4 {

5 new Employee{ ID=1, Name="sam"},

6 new Employee{ ID=2, Name="leo"},

7 new Employee{ ID=3, Name="korn"}

8 };

9 List<SelectListItem> itemList = (from n in empList.ToList()

10 select new SelectListItem { Text = n.Name, Value = n.ID.ToString(), Selected = false }).ToList();

11

12 ViewData["choose"] = itemList;

13 return View();

14 }

复制代码

还可以直接创建SelectList对象来得到同样的结果 这种方式更简洁 如

复制代码

1 public ActionResult Index()

2 {

3 List<Employee> empList = new List<Employee>

4 {

5 new Employee{ ID=1, Name="sam"},

6 new Employee{ ID=2, Name="leo"},

7 new Employee{ ID=3, Name="korn"}

8 };

9 SelectList selecyList = new SelectList(empList, "ID", "Name");

10 ViewData["choose"] = selecyList;

11 return View();

12 }

复制代码

在视图中只需两个参数 参数1为choose 引用的是ViewData["choose"]中的键

1 @Html.DropDownList("choose","请选择")

**Html.DropDownListFor（expression,selectList,htmlAttribute）**

同样是创建下拉选项的方法 但+For后缀的方法可以使用强类型作为参数expression的参数 推导出类型的属性 它可以将类型的属性名字作为下拉选项元素的name的值

**expression**

类型为System.Linq.Express.Expression.<Fun<dynamic,Tproperty>>的表达式

**selectList**

类型为SelectLlist的对象

复制代码

1 public ActionResult Index()

2 {

3 List<Employee> empList = new List<Employee>

4 {

5 new Employee{ ID=1, Name="sam"},

6 new Employee{ ID=2, Name="leo"},

7 new Employee{ ID=3, Name="korn"}

8 };

9 SelectList selectList = new SelectList(empList, "ID", "Name");

10 ViewData["choose"] = selectList;

11 return View();

12 }

复制代码

**在视图中**

1 @model Course.Models.Employee

2 @{Html.BeginForm("index", "default", FormMethod.Post);}

3 @Html.DropDownListFor(n=>n.Name,ViewData["choose"] as SelectList,new{@class="selectStyle"})

4 @{Html.EndForm();}

3.5 Html.ActionLink

**Html.ActionLink(linkText , actionName , controlName , routeValues , htmlAttribut）**

**//var url = "Url.Action("SearchResult")" + "?name=" + keyword;**

创建超链接

**linkText**

超链接文本

**actionName**

提交到哪个Action处理

**controlName**

提交到哪个控制器 可选 默认就是当前视图所属的Action所在的控制器

**routeValues**

object对象 设置超链接查询字符 跟设置html属性是一样的 如new {id=1,name="sam"}

1 @Html.ActionLink("详细","Detail",new { id=1})

生成的超链接为

1 生成对应的超链接为：

2 <a href="default/Detail/1">详细</a>

提供多个查询字符

1 Html.ActionLink("详细","Detail",new { id=1,name="sam"})

2 生成对应的超链接为：

3 <a href="default/Detail?id=1&name=1">详细</a>

3.6 Html.Partial

**Html.Partial（）**

将分布视图渲染到当前视图页面 该方法具有多个重载版 所有参数介绍如下

partialName

参数为分布视图名称

viewData

一个ViewDataDictionary类型的对象

model

分布视图需要使用的强类型对象

分布视图可以手动在当前视图的目录中创建 扩展名还是cshtml 只不过里面没有<html>/<head>/<body>标签 你可以在分布视图中编写body以下级别的html元素 也可以写Js和服务端脚本 除了没有几个主体标签 其它和视图都是一样的 可以右击Action - 添加视图 选择分布视图 如

[](http://images.cnitblog.com/blog/406079/201309/06221250-d3cf0472d57d4ca589990c2c88d79318.png)

打开TestPartial文件 输入以下代码做个测试

<div style="background:#010067;width:200px;height:100px;padding-top:50px;text-align:center;color:white;">

<label style="color:red;font-size:20px;font-weight: bolder;vertical-align:middle;">+</label>

<label style="vertical-align:middle;">这个框里的内容是分布视图TestPartial</label>

</div>

接着打开Index视图 调用此方法来加载分布视图TestPartial

<body>

<div style="background:#ffd800;width:200px;padding:10px;">

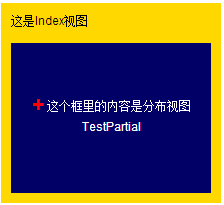
这是Index视图<br /><br />

@Html.Partial("TestPartial")

</div>

</body>

运行<http://localhost:8559/default/index> 结果如图：

[](http://images.cnitblog.com/blog/406079/201309/06221726-39873b64d3dd48e4aa87464daac3a43e.png)

viewData参数和model参数的用法如下

复制代码

<body>

<div style="background:#ffd800;width:200px;padding:10px;">

这是Index视图<br /><br />

@Html.Partial("TestPartial", new Course.Models.Employee{ ID=1, Name="sam"}, new ViewDataDictionary { {"music1","Free Jazz"},{"music2","BossaNova"}})

</div>

</body>

复制代码

在分布视图中可获取viewData和model model必须声明一下

复制代码

@model Course.Models.Employee

<div style="background:#010067;width:200px;height:100px;padding-top:50px;text-align:center;color:white;">

<label style="color:red;font-size:20px;font-weight: bolder;vertical-align:middle;">+</label>

<label style="vertical-align:middle;">@Model.Name</label>

<p><label style="vertical-align:middle;">@ViewData["music1"]</label></p>

<p><label style="vertical-align:middle;">@ViewData["music2"]</label></p>

</div>

复制代码

运行<http://localhost:8559/default/index> 结果如图：

[](http://images.cnitblog.com/blog/406079/201309/06221251-fcf4bcf5a5a34e5b848efd4400eed94d.png)

**Html.RenderPartial（）**

与Partial方法有类似的行为 区别在于Partial是将分布视图作为字符串加入主视图 而RenderPartial则是将分布式图写入响应输出流 在性能上RenderPartial要优于前者 但以往内此方法不返回值 所以必须使用中括号括起来

@{Html.RenderPartial("TestPartial");}

@{Html.RenderPartial("TestPartial", new Course.Models.Employee{ ID=1, Name="sam"}, new ViewDataDictionary { {"music1","Free Jazz"},{"music2","BossaNova"}});}

3.7 Html.Action

**Html.Action（）**

调用一个子操作（Action） 并以Html形式返回结果

actionName

Action的名称

controllerName

控制器的名称

routeValues

路由参数 格式:new {id=xx,code=xxx}

此方法与Partial类似 区别在于 Partial方法不经过Action处理 它直接加载一个分部视图页面 而Action方法会先经过Action处理再加载分布视图 其用法如下

public ActionResult Menu()

{

return PartialView("Menu");

}

右击Menu方法 创建一个分部视图Menu 打开分部视图 输入如下代码

复制代码

<div style="background: #010067; text-align: center; color: white;">

<p>这是分布视图</p>

<ul>

<li>坂本龍一 </li>

<li>Ian Brown</li>

<li>Ataraxia </li>

</ul>

</div>

复制代码

在Index页面调用此方法

<body>

<div style="background:#ffd800;width:200px;padding:10px;">

这是Index视图<br /><br />

@Html.Action("Menu")

</div>

</body>

结果

[](http://images.cnitblog.com/blog/406079/201309/06221251-cd7ed90c528e4faa9deadc1981d46c49.png)

**Html.RenderAction（）**

与Action方法有类似的行为 区别在于Action方法是将分布视图作为字符串加入主视图 而RenderAction则是将分布式图写入响应输出流 在性能上RenderAction要优于前者 但以往内此方法不返回值 所以必须使用中括号括起来

类似的Html辅助方法还有Html.Hidden（）、Html.PassWord（）、Html.TextBox（） 略 ……

UrlHelper类提供了方法用于生成**URL**

3.8 其他辅助

**Content（）**

**将一个虚拟的、相对的URL转换为应用程序的绝对URL 如**

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

**Encode（）**

**对URL地址进行加密**

1 @Url.Encode("http://www.cnblogs.com/")

生成http%3a%2f%2fwww.cnblogs.com%2f

四. 请求—>处理-->回应的原理

4.1 登录请求

主要流程没有什么好说的，主要通过一个简单的例子来说明一些问题。

首先，我们要创建一个User对象

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations; //引用这个才可以使用注解[Required]等  using System.Linq;  using System.Web;  namespace zkMvc.Areas.Member.Models  {  public class User  {  [Required(ErrorMessage="必填")]  [StringLength(20, MinimumLength = 4, ErrorMessage = "{2}到{1}个字符")]  [Display(Name = "用户名")]  public string UserName { get; set; }  [Required(ErrorMessage = "必填")]  [Display(Name = "密码")]  [StringLength(20, MinimumLength = 6, ErrorMessage = "{2}到{1}个字符")]  [DataType(DataType.Password)]  public string Password { get; set; }  public string email { get; set; }  } |

然后我们创建一个TestController

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.Mvc;  using zkMvc.Areas.Member.Models;  namespace zkMvc.Areas.Member.Controllers  {  public class TestController : Controller  {  //  // GET: /Member/Test/  public ActionResult Register() //用来进入注册页面的代码。  {  return View("Register");  }  [HttpPost] //添加这个注解，说明这个方法就是用来处理post提交的方法  [ValidateAntiForgeryToken] //添加安全保护。  public ActionResult rgisterAfter(User register) //点击提交后，进入的逻辑。  {  return Content(register.UserName);  }  }  } |

再根据需要创立一个Register 视图

|  |
| --- |
| model zkMvc.Areas.Member.Models.User  @{  ViewBag.Title = "注册";  }  <h2>Register</h2>  @using (Html.BeginForm("registerAfter","Test",FormMethod.Post))  { //这里表示创建一个表单，submit的url为 Test/registerAfter  @Html.AntiForgeryToken()  <div class="form-horizontal">  <h4>用户注册</h4>  <hr />  @Html.ValidationSummary(true)  <div class="form-group">  @Html.LabelFor(model => model.UserName, new { @class = "control-label col-md-2" })  <div class="col-md-10">  @Html.EditorFor(model => model.UserName)  @Html.ValidationMessageFor(model => model.UserName)  </div>  <!--添加 @Html.ValidationMessageFor(model => model.UserName) 表示展示错误信息 -->  </div>  <div class="form-group">  @Html.LabelFor(model => model.Password, new { @class = "control-label col-md-2" })  <div class="col-md-10">  @Html.EditorFor(model => model.Password) //这表示input  @Html.ValidationMessageFor(model => model.Password) //表示错误信息  </div>  </div>  <div class="form-group">  <div class="col-md-offset-2 col-md-10">  <input type="submit" value="注册" class="btn btn-default" />  </div>  </div>  </div>  }  @section Scripts {  @Scripts.Render("~/bundles/jqueryval")  } |

这里值得着重讲解一下：

Q：如果Html.BeginForm("registerAfter","Test",FormMethod.Post)改成Html.BeginForm().且submit按钮也没有什么触发事件。那是不是就无法提交呢？

A：不是的，只要把TestController中的registerAfter(User user)方法改成

Register方法的重载 Register(User user),就可以了。

|  |
| --- |
| public ActionResult Register(User register) //点击提交后，进入的逻辑。  {  return Content(register.UserName);  } |

**这是因为点击提交后，如果没有指定url，那么仍旧会使用 Test/Register 这个url，且带了一个绑定了model的对象。所以就会转到重载的Register方法中去。**

asp.mvc中的url主要是进入控制器中的，在控制器中再转入到相应的页面。

\* return Content(register.UserName);这一行代码只是为了测试。实际上，这里应该会从web层转到service层，且转到Dao层去获取数据，验证数据库中是否有用户，如果没有就把新用户数据插入到数据库中。并根据返回值，返回相应的视图中去。

\* 可以看到User中定义了三个属性，但是我们往controller中只传了两个属性，这样也是可以形成对象的。

4.2 view和Controller中的传值方式

1: ViewData传值方式  
 ViewData的生命周期和View相同, 只对当前View有效.  
   ViewData["zd"] = dfdfd   
2:TempData传值方式  
   可以跨Action传递  
   TempData的数据至多只能经过一次Controller传递, 并且每个元素至多只能被访问一次，  
   例如一个用法为，抛出一个异常。跳转到error页面

|  |
| --- |
| public ActionResult Index3() {       TempData["tempIndex"] = "出错了!";       Response.Redirect("/home/error");       return View(); } |

3:QueryString传值  
1)也可以使用new{}来为form的action增加querystring  
2)在controler里使用Request.QueryString["word"]获取值  
例如：

|  |
| --- |
| <li>                 <%= Html.ActionLink("Browse", "Browse", "User", new { word = "word1" }})%></li>  Controler页面： public ActionResult Browse(string word)         {             ViewData["word"] = Request.QueryString["word"];             ViewData["word2"] = word;             return View();         } |

4 Post传值  
 4.1所讲就是Post传值

4.3 代码分层问题  
 …………………………………………………………………….

**by zk 2017/6/12**