

Asp.net Mvc Framework 系列

做为设计模式的王者,MVC在众多领域都成为良好的模型的代名词,前日我们只能靠 Monorail来实现Asp.net的Mvc的而且确Asp.netMvc已经成为现实
本文只想让大家更直观地认知Asp.net Mvc,如果语言有所不当,还望先贤海涵,当然,如果文中有所纰漏还希望大家指出
尽量本着对初学者负责的态度来写,但期间的恒心与毅力相信过来的人更加明白,所以如果书写有误希望大家谅解.

Asp.net Mvc是ASP.NET 3.5 Extensions Preview 的一个部分.

最新的是 [ASP.NET MVC Preview on CodePlex.com](#)如果后续文章的书写中版本变化,笔者将在后面文章中进行补充说明.

说明:

1. 本文的前提环境为.net 3.5,但笔者会尽力写在.net2.0 下兼容的程序
2. 文本中所使用的 IDE 都为 Visual Studio 2008 RTM(中文) 语言基本为 C#不过为了方便大家理解 ,也可能会有些 Visual Basic
3. 笔者计算机操作系统为 Windows 2003 std
4. 笔者将在文中插入少许广告性例程,希望大家不要反感
5. 其它约定笔者将会后续补充

以下文章属于 **Asp.net Mvc CodePlex Preview 4**

- [Asp.net MVC Preview 4 中自定义Jquery的HtmlHelper扩展](#)
- [Asp.net MVC Preview 4 中使用RenderComponent](#)
- [Asp.net Mvc Pv4 中使用AjaxHelper](#)
- [ASP.NET MVC CodePlex Preview 4 Installer + Source + Changed](#)

以下文章是属于 **Asp.net MVC preview 3**

- [Asp.net MVC Fckeditor的扩展\(支持PV3 及自动绑定\)](#)
- [Asp.net MVC Render及Redirect的扩展](#)

以下文章是属于 **Asp.net MVC preview 2**

- [Asp.net Mvc Framework 一 \(安装并建立示例程序\)](#)
- [Asp.net Mvc Framework 二 \(URL Routing初解\)](#)
- [Asp.net Mvc Framework 三 \(Controller与View\)](#)
- [Asp.net Mvc Framework 四 \(在.net2.0 下运行\)](#)
- [Asp.net Mvc Framework 五 \(向View传值以及Redirect\)](#)

- [Asp.net Mvc Framework 六 \(更多的View传值及显示方式\)](#)
- [Asp.net Mvc Framework 七 \(Filter及其执行顺序\)](#)
- [Asp.net Mvc Framework 八 \(Helper\)](#)
- [Asp.net Mvc Framework 九 \(View与Controller交互\)](#)
- [Asp.net Mvc Framework 十\(测试方法及Filter的示例\)](#)
- [Asp.net Mvc Framework 十一 \(自定义Helper在MVC中的使用\)](#)
- [Asp.net Mvc Framework 十二 Castle扩展](#)
- [Asp.net MVC P2 中无法正确获取 CheckBox值的bug的解决方案](#)
- [Asp.net Mvc中MVContrib中无法使用Castle的发解决方案](#)

相关站点:

- [ASP.NET 开发者中心](#)
- [MVC Framework 下载](#)
- [MVC Framework 官方论坛](#)
- [MVC Framework 文档](#)
- [MVC Contrib 提供MsMVC的扩展](#)

Asp.net Mvc Framework 一 (安装并建立示例程序)

下载

下载地址为:

<http://www.microsoft.com/downloads/details.aspx?FamilyId=38CC4CF1-773A-47E1-8125-BA3369BF54A3&displaylang=en>

下载后将会获得一个AspNetMVCPreview2-setup.msi文件

双击安装

打开Visual Studio 2008(下文中简称vs)

点击菜单中的 文件>新建>项目

在项目类型中选择 C#>Web (上方的Framework选择.net Framework 3.5)则右边会出现

Asp.Net Mvc Web Application 选择即可新建项目

如果没有出现项目模板,可以在命令行中执行 `X:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\devenv.exe /setup`

X为安装Vs的盘符

在新建项目时会出现一个 "Create Test Project"窗口来询问是否创建测试工程,一般情况下默认即可

确定后就会出现新建好的两个工程

Asp.net MVC工程MvcApplication1

Asp.net MVC 测试工程MvcApplication1Tests

我们主要是讲解MvcApplication1

默认情况下执行,即会得到一个"My Sample MVC Application"站点

下面我们讲一下程序中的结构



也许你对这里讲的概念有点模糊,下面让我们来看看这个简单程序里是怎么写的,我们打开 Controller/HomeController.cs

代码如下

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
//请按 1.2.3.4 编号顺序看
namespace MvcApplication2.Controllers
{
    /// <summary>
    /// 1.HomeController 对应 Views 中的 Home 文件夹
    /// </summary>
    public class HomeController : Controller
    {
```

```

/// <summary>
/// 2.这个东西叫 Action 这个 Action 的名字(name)是 Index
/// 默认情况下对应的是 Views 中 此 Controller(HomeController)对
/// 应文件夹(Views/Home)下的 同名 Aspx 文件 (Views/Home/Index.aspx)
/// </summary>
public void Index() {
    //4.这个是要显示的 View 的名字.前面说的只是默认情况下
    //对应的是同名文件,所以当然你也可以自己指定
    RenderView("Index");
}
/// <summary>
/// 3.同上所述,这个 Action 叫 About
/// 默认情况下对应的是(Views/Home/About.aspx)
/// </summary>
public void About() {
    RenderView("About");
}
}
}
}

```

上面讲解我想已经将 Controller/Action 与 Views 中的关系讲得很明了了

这里要注意一点,在这个示例中网站使用了母板页.

即 Views/Shared/Site.Master 这个文件为其它文件提供共同母板

那么我们怎么去访问这些网页呢

如果您是一位 Web 开发者,您会说当然是

<http://localhost/Views/Home/Index.aspx> 和

<http://localhost/Views/Home/About.aspx> 了

其实不然,且听下节

Asp.net Mvc Framework 二 (URL Routing初解)

什么是 URLRouting 呢?

你可以使用 URL routing 来配置一些 URL 的映射,使用户可以按你的规则来访问网站.

使用 URL routing,一定要规定 URL 模式,它包括一个位置标识,它将在你请求网页时按这个规则返回给你内容.当然,这个创建的规则完全是由你自己定义的.

上回说道:

http://localhost/Views/Home/Index.aspx 和

http://localhost/Views/Home/About.aspx 并无法访问

Views/Home/Index.aspx

与

Views/Home/About.aspx

这是怎么回事呢,那我们要怎样才能访问呢

答案是:

http://localhost/Home 和

http://localhost/Home/About

于是可能你会问了:为什么呢?(MS 很春很晚)

原因是因为页面 URL 的请求规则在 Global.asax.cs 中定义的规则所决定

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;
namespace MvcApplication2
{
    /// <summary>
    /// 还是老规矩,按序号看
    /// </summary>
    public class GlobalApplication : System.Web.HttpApplication
    {
        public static void RegisterRoutes(RouteCollection routes) {
            // 4.注意: 将 URL 规则更改为 "{controller}.mvc/{action}/{id}" 即可
            // 自行支持 IIS6 and IIS7 两种模式
            // 笔者注:一般的虚拟主机不支持.mvc,.aspx 也要检查文件存在
            // 所以你可以将.mvc 换成.ashx 或.asbx

            //1.因为 MVC 与传统 Aspx 的最大不同就是访问是访问的 Controller/Action
            //而不是 aspx 文件,要展现给用户哪一个 aspx 文件是由 Controller 决定的
            //这个是文件默认自带的 URLRouting 规则,是将 Controller/Action/id 的访问
            //模式指向那个 Controller
            routes.Add(new Route("{controller}/{action}/{id}", new MvcRoute
```

```

Handler()
{
    Defaults = new RouteValueDictionary(new { action = "Index", id
= "" });
});
//2.这个 URL Routing 是为了解决直接访问域名时,会出现找不到文件的情况
//所以要采用这个方法将主页 Routing 到 Home/Index 上
routes.Add(new Route("Default.aspx", new MvcRouteHandler())
{
    Defaults = new RouteValueDictionary(new { controller = "Home"
, action = "Index", id = "" });
});
}
protected void Application_Start(object sender, EventArgs e) {
    //3.这个没什么好讲了,就是在应用程序启动时初始化它
    RegisterRoutes(RouteTable.Routes);
}
}
}
}

```

注意这一点

URL 只与 Controller 有关

URLRouting 是解决传统的

post.aspx?year=1999&month=3&day=8 的参数 URL 变为
/post/1999/3/8/这样的简短漂亮且有意义的 URL

以示例中的 Global.asax.cs 中定义的{controller}/{action}/{id}规则为列
/Home/About/12 其实就是访问

Controller="Home" Action="About" 它的参数为 id="12"

那么我们要怎么利用 Controller 写自己想要的页面呢

还是老话,下回分解

Asp.net Mvc Framework 三 (Controller与View)

这节我们让 Asp.netMVC 真正的跑起来

我们自己新建一个新的 Controller

开始行动:

在 Controllers 中新建一个 MVC Controller Class,个人宣传一下.就叫 EiceController

默认生成的代码如下:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace MvcApplication2.Controllers
{
    /// <summary>
    /// 记不记得前面讲过的,所有 Controller 都要继承于
    /// Controller 类
    /// </summary>
    public class EiceController : Controller
    {
        public void Index(string id) {
        }
    }
}
```

当然,除了 Controller 我们还要建个 View

先在 Views 中建个 Eice 文件夹

然后我们要建个 Index.aspx

注意了:要建 MVC View (Content) Page,如果你要使用母板页就选用 Content Page,反之选用一般 Page 即可

MVC 的 Aspx 文件与传统的 WebForm 的 Aspx 文件有所不同

我们将 EiceController 的 Index 写为

```
public void Index(string id) {
    ViewData["qs"] = id;
    RenderView("Index");
}
```


在 View 即/Views/Eice/Index.aspx 中写内容

```
<asp:Content ID="Content1" ContentPlaceHolderID="MainContentPlaceHolder" runat="server">
  <%=ViewData["qs"] %>

</asp:Content>
```

接下来我们访问

/eice/index/helloeice

也许你会发现,在页面上出现了 helloeice

由上面两段程序可以看出

string id 用于接收 QueryString["id"] 其实 Action 中的参数除了能接收 QueryString 以外也是可以接收 Forms 的

这里不做过多说明了,在后文中会有介绍

ViewData 是一个页面间的 IDictionary 用于 Controller 向 View 传递数据

这样 View 与 Controller 就可以协作完成显示页面与逻辑处理的工作了

Asp.net Mvc Framework 四 (在.net2.0 下运行)

这是一个题外话,如果您仅是要学习或有自己的服务器的话可以不用管这一节
但如果您使用的是虚拟主机或目标主机只允许.net2.0 的话应该这么做

所有版本为 3.5 或 3.0 的程序集引用属性 复制本地设为 True

另外改 Web.Config 如下

```
<?xml version="1.0"?>
<configuration>
  <configSections /><!--Asp.net Ajax 程序集的节点配置,如果不用,可以去掉-->
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <compilation debug="false">
      <!--
      <assemblies>
        <add assembly="System.Core, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
        <add assembly="System.Web.Abstractions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
        <add assembly="System.Web.Routing, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
        <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
        <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
        <add assembly="System.Data.Linq, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />
      </assemblies>-->
    </compilation>
    <authentication mode="Windows" />
    <pages>
      <namespaces>
        <!--
        <add namespace="System.Web.Mvc"/>
        <add namespace="System.Web.Routing"/>
        <add namespace="System.Linq"/>
        <add namespace="System.Collections.Generic"/>
        -->
      </namespaces>
    </pages>
    <httpModules>
```

```

    <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, System.Web.Routing, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
  </httpModules>
</system.web>
<!--
编译器设置 2.0 下 3.5 的编译器是不起作用的所以省去
<system.codedom>
  <compilers>
    <compiler language="c#;cs;csharp" extension=".cs" warningLevel="4"
      type="Microsoft.CSharp.CSharpCodeProvider, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
      <providerOption name="CompilerVersion" value="v3.5"/>
      <providerOption name="WarnAsError" value="false"/>
    </compiler>

    <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb" warningLevel="4"
      type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
      <providerOption name="CompilerVersion" value="v3.5"/>
      <providerOption name="OptionInfer" value="true"/>
      <providerOption name="WarnAsError" value="false"/>
    </compiler>
  </compilers>
</system.codedom>-->
<system.webServer>
  <validation validateIntegratedModeConfiguration="false"/>

  <modules runAllManagedModulesForAllRequests="true">
    <remove name="UrlRoutingModule" />
    <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, System.Web.Routing, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
  </modules>

  <handlers>
    <!--以下为 IIS7 管道设置 如为 IIS6 也可去掉-->
    <remove name="WebServiceHandlerFactory-Integrated"/>
    <add name="MvcScriptMap" preCondition="classicMode,bitness32" verb="*" path="*.mvc" modules="IsapiModule" scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll" />
    <add name="MvcScriptMap64" preCondition="classicMode,bitness64" verb="*" path="*.mvc" modules="IsapiModule" scriptProcessor="%windir%\Microsof

```

```
t.NET\Framework64\v2.0.50727\aspnet_isapi.dll" />  
    <add name="UrlRoutingHandler" preCondition="integratedMode" verb="*"   
path="UrlRouting.axd" type="System.Web.Routing.UrlRoutingHandler, System.W  
eb.Routing, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364  
E35" />  
    </handlers>  
    </system.webServer>  
</configuration>
```

Asp.net Mvc Framework 五 (向View传值以及Redirect)

ViewData 与 TempData 属性来向 View 页传递对象

上文中已经提到,使用 ViewData 可以将数据由 Controller 传递到 View

在前文中我们建立了 EiceController 类

在本文的示例中我们将这个 Controller 改一下

```
namespace MvcApplication2.Controllers
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Web;
    using System.Web.Mvc;

    /// <summary>
    /// 记不记得前面讲过的,所有 Controller 都要继承于
    /// Controller 类...当然 Controller 有很多种,我们慢慢讲
    /// </summary>
    public class EiceController : Controller
    {
        public void Index() {
            ViewData["ViewData"] = "在这里显示 ViewData";
            TempData["TempData"] = "在这里显示 TempData";
            RenderView("Index");
        }
        public void Index2() {
            RenderView("Index");
        }
    }
}
```

我们将 Index 的参数移除,并提供了 ViewData 和 TempData 的赋值

在 Views/Eice/Index.aspx 这个 View 中我们写以下代码

```
1: <%=ViewData["ViewData"]%><br />
2: <%=TempData["TempData"]%>
```

注意上面的 1.2 不是行号...

接下来我们运行工程

访问 <http://localhost/Eice/Index>

可以看到运行得到以下

1. 在这里显示 ViewData
2. 在这里显示 TempData

再访问 <http://localhost/Eice/Index2>

显示结果为

1.

2. 在这里显示 TempData

这里 1 显示的是 ViewData 中的内容, 2 为 TempData 传递的内容

我们可以看到

ViewData 只能在本 Action 中有效

但是 TempData 可以类似于 Session 一样到其它页面仍然存在, 但只限一页的访问(类似于 Monorail 中的 Flash)

TempData 一般用于临时的缓存内容或抛出错误页面时传递错误信息

Redirect 方法跳转页面到其它的 Controller/Action

```
RedirectToAction(Action 名);
```

```
RedirectToAction(Action 名, Controller 名);
```

```
RedirectToAction(RouteValueDictionary);
```

在这里前两种都没有什么好说的 RedirectToAction("About", "Home"); 就是一种写法

主要是第三种重载

用户可以这样写

```
System.Web.Routing.RouteData routeData = new System.Web.Routing.RouteData();
routeData.Values.Add("Action", "About");
routeData.Values.Add("Controller", "Home");
RedirectToAction(routeData.Values);
```

这样就可以完成页面跳转

当然, 也可以使用传统的 Response.Redirect 来完成页面的跳转

Asp.net Mvc Framework 六 (更多的View传值及显示方式)

我们前面都使用 `RenderView("Index");` 这种方式来显示
`RenderView` 的重载:

```
RenderView(string viewName);  
RenderView(string viewName, object viewData);  
RenderView(string viewName, string masterName);  
RenderView(string viewName, string masterName, object viewData);
```

我们常用的当然就是第一种

第二种 `RenderView(string viewName, object viewData);` 是在显示 view 时附加一个 `ViewData`

如:

```
RenderView("Index", new  
{  
    name = "重典",  
    sex = true  
});
```

我们就可以在相应的 View(即 `Index.aspx`) 中调用 `<%=ViewData["name"]%>` 来得到它的值

`RenderView(string viewName, string masterName);`

则是除了 `Viewname` 之外还指定了母板页

如果程序写为

```
RenderView("index", "layoutpage");
```

则是显示 `index` 这个 View, 但是母板页使用 `/Views/Shared/layoutpage.master`

最后一个重载是前两者的结合, 这里不多说了

更多的 View 传值方式

前面说了向 View 传值可以用 `ViewData` 或者是 `TempData`

这里我们介绍 `ViewData` 传值的另一种的方式

与 Models 绑定

这是我们第一次介绍 Models, 其实 Models 就是一个数据模型, 比如, 用户, 人, 访问记录等
我们建立一个 `Person` 类

```
namespace MvcApplication2.Models  
{  
    public class Person  
    {  
        string _name;  
  
        public string Name {  
            get { return _name; }  
        }  
    }  
}
```

```

    set { _name = value; }
}
bool _sex;

public bool Sex {
    get { return _sex; }
    set { _sex = value; }
}
}
}
}

```

类中有 Name 与 Sex 两个属性

我们仍然用 Eice 的 Index 这个 Action 来书写示例

首先我们实例化一个 Person 并将之传给 View

```

Person p = new Person()
{
    Name = "邹健",
    Sex = true
};
RenderView("Index", p);

```

接下来我们更改 View 中 Eive/Index.aspx.cs

将它的更改如下

```

public partial class Index : ViewPage<Person>
{
}

```

也就是原来 Index 是继承 ViewPage 的而现在继承了 ViewPage<Person> 这个泛型

接下来我们可以在 Index.aspx 中写类似以下语句了

```

<%=ViewData.Name %>
<%=ViewData.Sex %>

```

当然您也可以不写 ViewPage<Person> 而还像从前一样继承于 ViewPage

那么访问方法就是过去的

```

<%=ViewData["Name"] %>
<%=ViewData["Sex"] %>

```

回首望去 MS 讲的内容与标题反了...Eat 去了...每天都为吃发愁...

Asp.net Mvc Framework 七 (Filter及其执行顺序)

应用于 Action 的 Filter

在 Asp.netMvc 中当你有以下及类似以下需求时你可以使用 Filter 功能
判断登录与否或用户权限,决策输出缓存,防盗链,防蜘蛛,本地化设置,实现动态 Action
filter 是一种声明式编程方式,在 Asp.net MVC 中它只能应用在 Action 上
Filter 要继承于 ActionFilterAttribute 抽象类,并可以覆写 void
OnActionExecuting(FilterExecutingContext)和
void OnActionExecuted(FilterExecutedContext)这两个方法
OnActionExecuting 是 Action 执行前的操作,OnActionExecuted 则是 Action 执行后的
操作

下面我给大家一个示例,来看看它的的执行顺序

首先我们先建立 一个 Filter,名字叫做 TestFilter

```
using System.Web.Mvc;

namespace MvcApplication2.Controllers
{
    public class TestFilter : ActionFilterAttribute
    {
        public override void OnActionExecuting(FilterExecutingContext
            filterContext) {
            filterContext.HttpContext.Session["temp"] += "OnActionExecuting<
br/>";
        }

        public override void OnActionExecuted(FilterExecutedContext
            filterContext) {
            filterContext.HttpContext.Session["temp"] += "OnActionExecuted<
br/>";
        }
    }
}
```

在这里我们在 Session["temp"]上标记执行的顺序
我们在 Controller 中的 Action 中写以下代码

```
[TestFilter]
public void Index() {
    this.HttpContext.Session["temp"] += "Action<br/>";
    RenderView("Index");
}
```

在这个 Action 执行的时候,我们也为 Session["temp"]打上了标记.

最后是 View 的内容
很简单我们只写

```
<%=Session["temp"] %>
```

这样我们就可以执行这个页面了
在第一次执行完成后,页面显示

OnActionExecuting

Action

这证明是先执行了 OnActionExecuting 然后执行了 Action,我们再刷新一下页面
则得到

OnActionExecuting

Action

OnActionExecuted

OnActionExecuting

Action

这是因为 OnActionExecuted 是在第一次页面 显示后才执行,所以要第二次访问页面时
才能看到

Controller 的 Filter

Monorail 中的 Filter 是可以使用在 Controller 中的,这给编程者带来了很多方便,那么在
Asp.net MVC 中可不可以使用 Controller 级的 Filter 呢.不言自喻.

实现方法如下 Controller 本身也有 OnActionExecuting 与 OnActionExecuted 方法 ,
将之重写即可,见代码

```
namespace MvcApplication2.Controllers
{
    using System.Web.Mvc;
    public class EiceController : Controller
    {
        public void Index() { this.HttpContext.Session["temp"] += "Action<br/>";
        |
        |     RenderView("Index");
        | }
        public void Index2() {
        |
        |     RenderView("Index");
        | }
        protected override void OnActionExecuting(FilterExecutingContext
        | filterContext) {
        |     filterContext.HttpContext.Session["temp"] += "OnActionExecuting<br/>";
        | }
    }
}
```

```

|
|
|   protected override void OnActionExecuted(FilterExecutedContext
|   filterContext) {
|       filterContext.HttpContext.Session["temp"] += "OnActionExecuted<
br/>";
|   }
|   }
| }

```

这里要注意一点,这两个方法 一定要 **protected**
要是想多个 Controller 使用这个 Filter 怎么办?继承呗.

Filter 的具体生存周期

这是官方站的一数据.

1. 来自 controller 虚方法 的 OnActionExecuting .
2. 应用于当前 Controller 的 Filter 中的 OnActionExecuting:

先执行基类的,后执派生类的

3. 执行应用于 Action 的 Filter 的 OnActionExecuting 顺序:

先执行基类的,后执派生类的

4. Action 方法
5. 应用于 Action 的 Filter 的 OnActionExecuted 的执行顺序

先执行派生类的,后执行基类的

6. 应用于当前 Controller 的 Filter 中的 OnActionExecuted 方法

先执行派生类的,后执行基类的

7. Controller 中的虚方法 OnActionExecuted

示例可见 <http://quickstarts.asp.net/3-5-extensions/mvc/ActionFiltering.aspx>

Asp.net Mvc Framework 八 (Helper)

本人已经疯了...快写完了而关掉浏览器丢失数据之事在此文章上发生了两次,所以本人倍加珍惜

这节讲一下 Asp.netMVC 中的 Helper

何谓 Helper,其实就是在 View 中为了实现一些灵活功能而写的方法组

其实 Asp.net MVC 的 View 是 Aspx 的页面,本身可以声明定义方法,那为什么要有 Helper 呢

其实无非是将界面与逻辑分离,而且 Asp.net MVC 也并不只支持 Aspx 一种 View,在扩展包中,也有 Castle 的 NVelocity 引擎和 Boo 所以,如果在 Aspx 中定义方法的话会影响其扩展性和可移植性.而且代码也不太好看.

UrlHelper 的 Action 方法 用于生成一个超级链接,它的重载为

```
public string Action(string actionName);
public string Action(string actionName, object values);
public string Action(string actionName, RouteValueDictionary valuesDictionary);
public string Action(string actionName, string controllerName);
public string Action(string actionName, string controllerName, object values);
public string Action(string actionName, string controllerName, RouteValueDictionary valuesDictionary);
```

例如我在 View 中写 `Url.Action("Index","Home")`,运行后则会生成 `/Home/Index` 这个地址

如果你的系统中的 URL Routing 规则总是变化的话这个 Helper 则是你必备之选.

```
public string Encode(string url);
```

这也是 UrlHelper 的一个方法 使用方法 如 `<%=Url.Encode("中文")%>` 功能与 `Server.UrlEncode` 相同,这里不多说了

如果你有特殊需要可以用 3.0 新特性,扩展方法来为 UrlHelper 来增加新的功能

HtmlHelper 则是另一个常用之 Helper

它是来生成 HTML 代码用的

eg.

```
<%=Html.ActionLink("首页","index","Home")%>
```

则生成 `首页`

重载方法有:

```
public string ActionLink(string linkText, string actionName);
public string ActionLink(string linkText, string actionName, object values);
public string ActionLink(string linkText, string actionName, RouteValueDictionary valuesDictionary);
```

```
public string ActionLink(string linkText, string actionName, string controllerName);
public string ActionLink(string linkText, string actionName, string controllerName, object values);
public string ActionLink(string linkText, string actionName, string controllerName, RouteValueDictionary valuesDictionary);
```

当然 HTMLHelper 的种类就比 UrlHelper 多得多了
比如有 Button

```
<%=Html.Button("name","value","onclick") %>
```

生成

```
<button onclick="onclick" name="name" id="name">value</button>
```

CheckBox:

```
<%=Html.CheckBox("name","text") %>
```

生成

```
<input value="text" name="name" type="checkbox">&nbsp; text
```

Form:

```
<%=Html.Form("Home","Index",FormMethod.Post) %>
```

生成

```
<form action="/Home" method="post">System.Web.Mvc.SimpleForm
</form>
```

当然还有类似于SubmitButton,Image这些方法,这里就不多讲了

注意一点Preview2 中Html.Mailto方法有些Bug请尽量避免使用这个方法

吾生也有涯，而知也无涯，以有涯随无涯，殆已

附:

功能介绍还有几篇就写完了,争求意见,下面可以讲示例也可以讲对Asp.net MVC进行扩展,不知道大家想看什么,有兴趣的朋友可以回复一下,我做个参考

Asp.net Mvc Framework 系列

Asp.net Mvc Framework 系列

供大家学习参考,转文章随意--重典

Tag标签: CHSNS#,Microsoft,Asp.net MVC,.net,Helper

posted @ 2008-03-13 17:12 重典 阅读(2555) 评论(25) 编辑 收藏 所属分类: Microsoft MVC

发表评论

[回复](#) [引用](#) [查看](#)

[#1 楼](#) 2008-03-13 17:27 | [cslar](#)

哈哈 楼主不要急

是不是在等我占沙发啊?

[回复](#) [引用](#) [查看](#)

[#2 楼](#) [楼主]2008-03-13 17:31 | [重典](#)

@cslar

兄弟这个真是水啊...爆湿的回复,呵呵

[回复](#) [引用](#) [查看](#)

[#3 楼](#) 2008-03-13 17:36 | [永春](#)

感觉和 MonoRail 太像了-_-

有时间的话来点示例吧,看起来快,也直观一点的

[回复](#) [引用](#) [查看](#)

[#4 楼](#) [楼主]2008-03-13 17:39 | [重典](#)

@永春

和 Monorail 那不是一般的像,简直是同卵双生....

不过现在还没有 Monorail 那么方便

我刚刚回了您的帖子,一回来就看到您回了我的,真是...缘份啊

[回复](#) [引用](#)

[#5 楼](#) 2008-03-13 22:10 | [zsj](#) [未注册用户]

很好很不错!

建议先将几个实例,然后深入展开。

对我个人而言,前面的内容基本 ok 了,但是上一节的 fliter 部分不是很清楚,特别是实际应用,迫切希望看到这方面的例子:) 谢谢楼主。

关注 ing...

[回复](#) [引用](#) [查看](#)

[#6 楼](#) [楼主]2008-03-13 23:06 | [重典](#)

@zsj

好的,我下篇文章讲交互,然后给个 Filter 的示例讲解

[回复](#) [引用](#)

[#7 楼](#) 2008-03-14 07:15 | [inhesoftvvv](#) [未注册用户]

Monorail 好象没有方法过滤器(Filter),只有 Controller 的过滤器,不知如何在 MR 中建立单独的方法过滤器,各位前辈能否指教?

[回复](#) [引用](#) [查看](#)

[#8 楼](#) [楼主]2008-03-14 08:42 | [重典](#)

@inhesoftvvv

你说的对,Monorail 中只有 Controller 上可以加过滤器

但方法上可以指定 Filter 不生效,如

```
[Filter(ExecuteEnum.BeforeAction, typeof(AdminFilter))]  
[Filter(ExecuteEnum.BeforeAction, typeof(LoginedFilter))]  
public class UserController : BaseController  
{  
[SkipFilter(typeof(LoginedFilter))]  
public void index(){};  
}
```

[回复](#) [引用](#) [查看](#)

#9 楼 2008-03-14 09:30 | [Shawn Ji](#)

示例好些。。但太简单的示例只能配合理论讲解而已:)

另有问题请教：在两个 Controller 中传值，如下：

HomeController

```
public void CheckIn()  
{
```

```
{
```

```
UserItem UserItem = new UserItem();
```

```
// TODO .....
```

```
RedirectToAction(new RouteValueDictionary(new {
```

```
controller = "Chatroom",
```

```
action = "ChatroomIndex",
```

```
nickName = UserItem.NickName,
```

```
email = UserItem.Email }));
```

```
}
```

ChatroomController

```
public void ChatroomIndex(string nickName, string email)
```

```
{
```

```
// TODO.....
```

```
}
```

在 Global.asax 中进行了多种配置，仍然无法实现，RedirectToAction 出错，

其实我只需要能够使用 RedirectToAction 传递多个参数的方法，不知有无简单方法解决。

谢先~

[回复](#) [引用](#) [查看](#)

#10 楼 [楼主]2008-03-14 09:34 | [重典](#)

@Shawn Ji

使用 TempData 即可

```
public void CheckIn()  
{
```

```
{
```

```
UserItem UserItem = new UserItem();
```

```
// TODO .....
```

```
TempData["UserItem"]=UserItem;
RedirectToAction( "ChatroomIndex","Chatroom");
}
```

```
ChatroomController
public void ChatroomIndex()
{
// TODO.....
//这样在这个的 View 时只要 TempData["UserItem"] as UserItem 就可以使用了
}
```

[回复](#) [引用](#) [查看](#)

[#11 楼](#) 2008-03-14 09:50 | [Shawn Ji](#)

Thank you for your help!

但是看到 TempData 就让我想到了 WebForm 中使用 ViewState 进行存值，当然这并不相同。

有时候并不需要将 UserItem 中所有的值进行传递使用。
难道就没有用参数的形式进行多 Value 的显式传值的方法么。。

[回复](#) [引用](#) [查看](#)

[#12 楼](#) [楼主]2008-03-14 10:09 | [重典](#)

@Shawn Ji

你的写法并无问题,可以正常传值
可能是你的 UriRouting 没有配置正确

新加一个

```
routes.Add(new Route("{controller}/{action}/{nickName}/{email}", new
MvcRouteHandler())
{
Defaults = new RouteValueDictionary(
new { controller = "Home", action = "Index", nickName= "", email= "" }),
});
```

应该就可以了

[回复](#) [引用](#) [查看](#)

[#13 楼](#) 2008-03-14 10:17 | [miao~](#)

强烈要求做个小的例子..如 BOLG 留言板之类...
看了这么多..不知道该怎么用..用在哪..一头雾水啊..呵呵..~~最好是和 LINQ 结合的例子...

[回复](#) [引用](#) [查看](#)

[#14 楼](#) 2008-03-14 10:22 | [Shawn Ji](#)

@重典

这种方法我试过了，当 email 有值的时候会出现 404 的错误，无值的时候正常
因为

```
Requested URL: /Chatroom/ChatroomIndex/aaaaaa/aaaaa@dddf.com
(nickName: aaaaaa, email: aaaaa@dddf.com)
```


我也见到有的例子中使用

("{controller}/{action}/{aaaa}/{bbbb}", 这种方式, 却没搞清楚是怎样实现的~

如果要写例子的话, 也请增加各种传值方式, 以及分层的方法, 跟楼上不同, 我在使用 ADO.NET Entity Framework, 而不是 LINQ, 个人觉得 ADO.NET Entity Model 更有潜力。

[回复](#) [引用](#) [查看](#)

[#15 楼](#) [楼主]2008-03-14 10:26 | [重典](#)

@Shawn Ji

可能是@在传送时是 Url 不允许的

你可以将之 UriEncode 试下看

[回复](#) [引用](#) [查看](#)

[#16 楼](#) [楼主]2008-03-14 10:29 | [重典](#)

@miao~

谢谢支持

今天事情很多,我正在写提交表单

我会写个简单的例子的...

来个留言版吧...呵呵(我可真是懒),最迟明天 Post 出来

[回复](#) [引用](#) [查看](#)

[#17 楼](#) 2008-03-14 10:34 | [Shawn Ji](#)

@重典

:~), 是.的问题, 光想着传值了, 这种问题也给忽略了。

再次感谢。

[回复](#) [引用](#) [查看](#)

[#18 楼](#) [楼主]2008-03-14 12:48 | [重典](#)

@Shawn Ji

^^

[回复](#) [引用](#) [查看](#)

[#19 楼](#) 2008-04-30 16:34 | [土星的狗狗](#)

实例自己来吧~呵呵

[回复](#) [引用](#)

[#20 楼](#) 2008-07-10 12:25 | [wfa](#) [未注册用户]

Html.ActionLink 用处不大, 我想给生成的 a 标签加 target 属性就没法做, 只能自己扩展了。

[回复](#) [引用](#) [查看](#)

[#21 楼](#) [楼主]2008-07-10 20:19 | [重典](#)

@wfa

Html.ActionLink(action,controller,new{ target="_blank"})

[回复](#) [引用](#) [查看](#)

[#22 楼](#) 2008-07-16 15:27 | [云の世界](#)

除了 url helper,别的 helper 有什么意义呢
能少用则少用吧，对网页设计人员和美工很不友好，
第三方编辑器支持也不好。

Asp.net Mvc Framework 九 (View与Controller交互)

这一回为避免写第八节时那种情况再次出现,我改用 Wps 写了,如果复制过去格式有问题请大家见谅

—邹健

本节所有示例都是讲解登录提交表的过程

为了本课能更好讲解我们先建立一个账号的 Model

```
namespace MvcApplication4.Models
```

```
{  
    public class Account  
    {  
        string _username;  
        public string Username {  
            get { return _username; }  
            set { _username = value ; }  
        }  
        string _password;  
        public string Password {  
            get { return _password; }  
            set { _password = value ; }  
        }  
    }  
}
```

一. 使用我们 Asp 时代的老朋友

我们建立一个 HomeController,之后在其中加一个 Index 的 Action

```
public void Index() {  
    RenderView( "Index" );  
}
```

是用于显示 Index 这个 View 的(Index.aspx):

这是提交表单的

```
< form method ="post" action ="<% = Url.Action("save") %>" >  
<% = Html.TextBox( "username" ) %>  
<% = Html.TextBox( "password" ) %>  
<% = Html.SubmitButton( "登录" ) %>  
</ form >
```

大家可以看到在这个 View 中 form 的 Action 为 save

于是我们还需要一个叫 save 的 Action 用于接收表单:

```
public void save(){  
    ViewData[ "username" ] = Request.Form[ "username" ];  
    ViewData[ "password" ] = Request.Form[ "password" ];  
    RenderView( "Result" );  
}
```

最后我们将结果显示在 Result.aspx 这个 View 中:

这是用于接收结果的

```
<% = ViewData [ "username" ] %>
```

```
<% = ViewData [ "password" ] %>
```

大家都看到了,我们在 Save 中使用了我们的老朋友 Request.Form 来接收了表单

二. 其实 Asp.netMVC 为我们提供了更好的方法

是什么方法呢?

那就是我觉得不错的一方法 ReadFromRequest 它可以接收 Form 和 QueryString

看以下 save 这个 Action 的代码

```
public void save(){
    ViewData[ "username" ] = this .ReadFromRequest( "username" );
    ViewData[ "password" ] = this .ReadFromRequest( "password" );
    RenderView( "Result" );
}
```

可以实现上面一样的功能

三. 我们可以用“参数”

前面我们提到了参数来接收的功能

```
public void save ( string username, string password){
    ViewData[ "username" ] = username;
    ViewData[ "password" ] = password;
    RenderView( "Result" );
}
```

一切 OK

四. 到激动人心的地方了我们使用绑定功能

在 Monorail 中我们有这样的功能

public void save([DataBind("account")] Account account) ;可以实现将一个对象与表单绑定

当然 Asp.net MVC 中也应该有

我们将 View 改成:

这是提交表单的

```
< form method ="post" action =" <% = Url.Action("save") %> ">
<% = Html.TextBox( " Account .username" ) %>
<% = Html.TextBox( " Account .password" ) %>
<% = Html.SubmitButton( "登录" ) %>
</ form >
```

而 Save 这个 Action 中写如下代码:

```
public void save() {
    Account user = new Account ();
    BindingHelperExtensions .UpdateFrom(user, Request.Form, " Account " );
    ViewData[ "username" ] = user.Username;
    ViewData[ "password" ] = user.Password;
    RenderView( "Result" );
}
```

呵呵, 仍然可以输

BindingHelperExtensions .UpdateFrom 让我们将从表单中以对象方式传递着数据

当然这个对象如果绑定的为 Dlinq 的对象就更加方便应用了
也可以通过这种方式绑定

View:

这是提交表单的

```
< form method="post" action = " <% = Url.Action("save") %> ">
<% = Html.TextBox( "username" ) %>
<% = Html.TextBox( "password" ) %>
<% = Html.SubmitButton( "登录" ) %>
</ form >
```

Action:

```
public void save() {
    Account user = new Account ();
    BindingHelperExtensions .UpdateFrom(user, Request.Form);
    ViewData[ "username" ] = user.Username;
    ViewData[ "password" ] = user.Password;
    RenderView( "Result" );
}
```

也可以实现

这里用了 `BindingHelperExtensions .UpdateFrom(user, Request.Form);` 来实现绑定
这个用于表单中只有一个对象的情况, 前一个则可适用于多个对象在同一表彰中的情况

本节示例MVC9.rar

Asp.net Mvc Framework 十(测试方法及Filter的示例)

示例下载:

<http://files.cnblogs.com/chsword/MyTestMvc.rar>

顺便说一下建立测试的方法

本身Asp.netMvc是提供有测试功能的

在新建Asp.netMVCApplication时,点击确定,会跳出一个Create Project Test的询问

如果选Yes就自动建立一个 "工程名+Test"的测试工程

测试工程要引用Rhino.Mock(最新版本 3.4)这一开源项目(下载地址

<http://www.ayende.com/projects/rhino-mocks/downloads.aspx>)

测试代码没有什么好说的了,看了示例自然了解,Scott的MockHelpers使测试更加方便

可以利用类似以下代码进行测试

```
[TestMethod]
public void Index() {
    HomeController home = new HomeController();
    var viewengin = new FakeViewEngine();
    home.ViewEngine = viewengin;
    MockRepository mock = new MockRepository();
    using (mock.Record()) {
        mock.SetFakeControllerContext(home);
    }
    using (mock.Playback()) {
        home.Index();
        Assert.AreEqual("Index", viewengin.ViewContext.ViewName);
    }
}
```

示例说明

示例中有个登录功能

有两个页面

/Post/Index 页面只有登录后才能查看

/Post/Post 页面只有登录后的 admin 用户才能查看

否则就会 Error

在验证登录时我写了一个扩展方法

```
public static class ControllerExtension
{
    public static bool IsPost(this Controller controller) {
        return controller.Request.Form.Count > 0;
    }
}
```

登录时我使用 Session 来保存用户登录信息

所有用户信息我在 Global 中进行初始化

```
RegisterRoutes(RouteTable.Routes);
//将 Application 代替数据库用
Application["Posts"] = new List<Post>(); //一个账号集合
Application["Accounts"] = new List<Account>(); //一个帖子集合
//初始化两个账号
List<Account> la = Application["Accounts"] as List<Account>;
la.Add(new Account()//这个是管理员
{
    Username = "admin",
    Password = "admin"
});
la.Add(new Account()//这个是普通用户
{
    Username = "user",
    Password = "user"
});
```

Asp.net Mvc Framework 十一 (自定义Helper在MVC中的使用)

Monorail 中的 Helper 是绑定在 Controller 上的

形如:

```
[Helper(typeof(ChHelper))]  
public abstract class BaseBlockController : SmartDispatcherController {  
}
```

但本身 Helper 是应用在 View 中的,所以 Monorail 这种定义方式 略微违背了分离之道

那么 Asp.net MVC 中是如何绑定自定义的 Helper 的呢?

其实 Asp.net MVC 的 View 也有代码文件即类似 index.aspx.cs

代码文件继承于 ViewPage

下面用一个实例来讲

我首先自定义一个 Helper

实现了对字符串的一些小扩展

```
public class StringHelper  
{  
    public string Red(string str) {  
        return string.Format(@"<span style=""color:red"">{0}</span>",str);  
    }  
    public string Green(string str) {  
        return string.Format(@"<span style=""color:green"">{0}</span>", str);  
    }  
}
```

那么我们怎样才能能在 View 使用它呢?

我们可以在 View 的 cs 中写以下代码

```
public StringHelper Str {  
    get;  
    set;  
}
```

即为

```
public partial class Index : ViewPage  
{  
    public StringHelper Str {  
        get;  
        set;  
    }  
}
```


这样我们就可以在 View 中使用

```
<%=Str.Red(ViewData["title"])%>
```

这样的代码了

如果这个 Helper 你想不仅在此一个文件中使用

那么可以通过将属性 写在一个 BaseViewPage 的基类中 然后所有的 View 都继承于自定义的 BaseViewPage 来实现

同样的,Master 文件中也可以通过类似的方法来完成使用自定义的 Helper