## mvc,webform,Layers的区别介绍及reflector下载

一．学习mvc5之前需要下载的东西

1.vs2013(mvc5) or vs2015(mvc6)

2.reflect插件,vs插件【用于调试代码】 ILSpy没办法调试Web代码

在工具联机中在线更新中下载reflector

二．简诉mvc的概念

1.MVC => Model + View + Controller

<1>.Model=>实体 【三层Model】(状态存储+业务逻辑)

<2>.Controller 就是我们的页面逻辑处理,用来处理Http发过来的请求,Model中获取数据然后转发给view,生成view.

<3>.View就是我们的页面展示【Razor模板】

三．和WebForm直接的区别

一个page贯穿一个.cs代码 1=>1 耦合

Controller 将behind和page做了分离 多=>多 松耦合

WebForm的缺点: ViewState状态(当年js不被重视,用户体验也不是那么重视)

为了隐藏http的无状态性,ViewState功不可没,封装太厉害没有js的用武之地。

四. Mvc和三层的区别

1.mvc只是三层中view层的功能细分

所以说mvc和三层没有直接关系

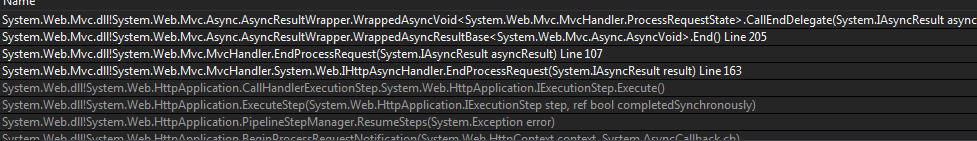
五.演示reflector的调试

1.生成PDB 把所有System.web.xxx都勾选上.

2.使用vs的调用堆栈我们就可以看出代码的调用流程

从调用堆栈中我们大概看出一个流程

PipelineRuntime=>HttpRuntime=>HttpApplication=>MvcHandler=>Controller=>HomeController.(MVC 管道的大致流程)



## 从ashx的角度理解asp.net中httpPipleline管道流

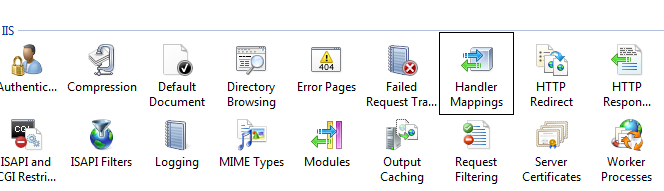
1. httpPipleline管道

1.一个http请求过来我们的iis和websit如何进行处理,然后到我们的处理代码中,这么一个整体流程。

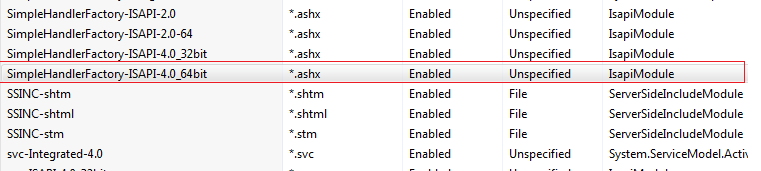
2.通过代码调试窥探管道流。

二．不管是mvc还是webform都是基于aspx.net这个框架。也就是说管道流都是差不多的,只是mvc在管道流中做了一个截取,通过module的方法。Webform得益于iis的处理程序映射.

三.为了方便我们从一般ashx处理程序入手。

处理程序映射

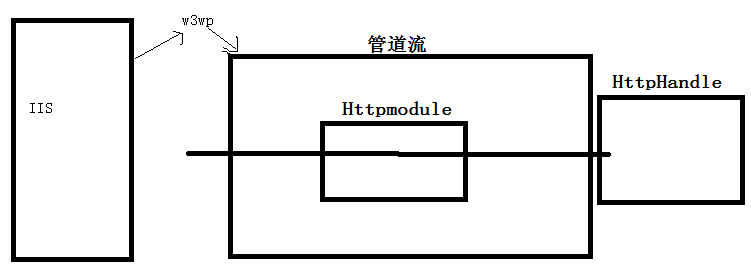
1. 很多人在不用mvc的时候,习惯用js+ashx.( System.Web.UI.SimpleHandlerFactory-ISAPI-4.0\_64bit)来处理.



四.Httpmodule , HttpHandle概念

1.httpmodule就是一个管道流的拦截器,我们可以在module中对http进行监控或者修改。

2.httphandle就是我们的处理程序。SimpleHandlerFactory产生handle



五.实战调试

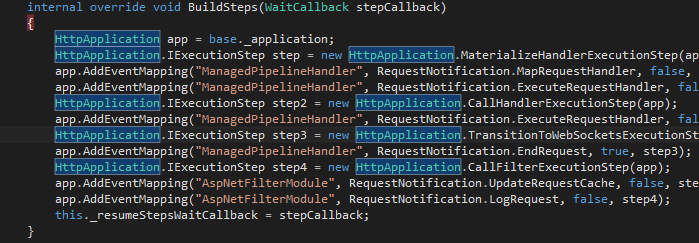
1. ProcessRequestNotification =>

InitializeRequestContext(nativeRequestContext, flags, out wr, out httpContext);构建一个httpcontext的上下文.

applicationInstance = HttpApplicationFactory.GetApplicationInstance(context); =>创建httpapplication(Globle)

InitInternal(HttpContext context, HttpApplicationState state, MethodInfo[] handlers);=> InitIntegratedModules()=>创建Module.

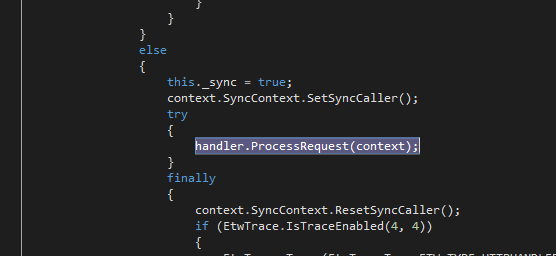
=> BuildSteps()=>创建我们的httphandle.(ManagedPipelineHandler)



注意获取handler的类,

MaterializeHandlerExecutionStep => gethandler

CallHandlerExecutionStep =>PR函数 handler.ProcessRequest(context);



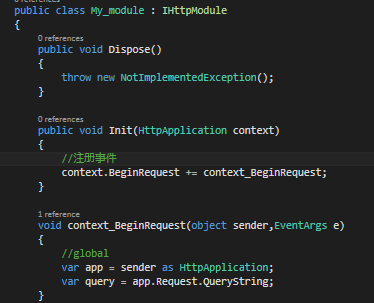
## 自定义module和handler加深对管道流的理解

1. 自定义Httpmodule,到现在为止我们应该知道httpmodule就是来拦截我们的请求,这拦截是通过事件注册的方式.其实在httpApplication 上有很多事件.
2. BeginRequest
3. EndRequest
4. PostResolveRequestCache 被MVC注册
5. ..等等

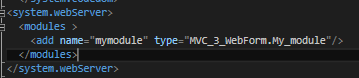
二.检测参数中是否有异常字符【防止sql注册】

<1>.要继承IHttpModule这个接口

<2>.在Init方法中注册一下刚刚看到的BeginRequest.



<3>.在webconfig中注册我们的mymodule节点



三.总结mudule就是通过注册application事件的形式进行拦截.【因为MVC就是这样做的】.

自定义HttpHandle.

我们在iis的”程序映射”中发现每一个扩展名都有一个”类”进行处理。

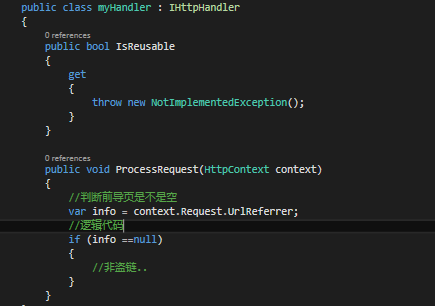
现在我们可以自己定义一个扩展名,然后自己配置一个【处理类】

例子：图片防盗链

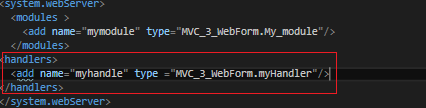
\*.JPG 默认就是静态文件,就是被StaticFileModule,DefaultDocumentModule,DirectoryListingModule处理的

\*.JGP=>自定义一个程序来处理.

1.定义一个继承于IHttpHandler的类



2.配置webconfig中的httphandler节点



1. 在ISS中处理程序中添加映射

## 从MCV的角度讲解如何通过UrlRoutingModule截获asp.net管道流

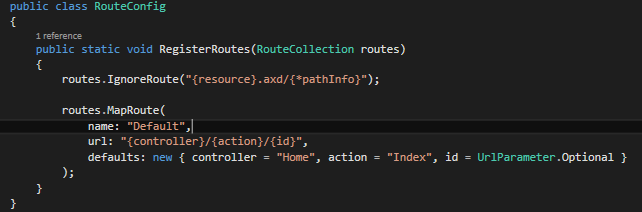
1. Mvc管道
2. mvc中到底有没有moudle和handler

非MVC中都是一个扩展名对应一个”handler”

XXXX.COM/HOME/index/1 =>并没有扩展名,所以没有办法去解析

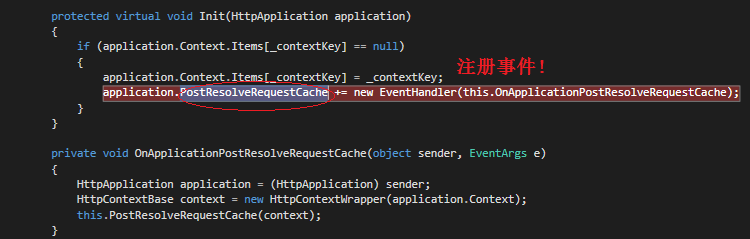
为了能解析,mvc中只能通过两种手段一种是module一种是handler(需要注册事件)

1. Route:定义路由模板

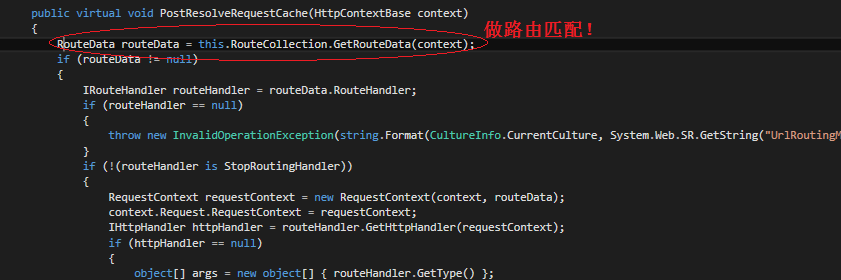


这个就是匹配的路由模板。

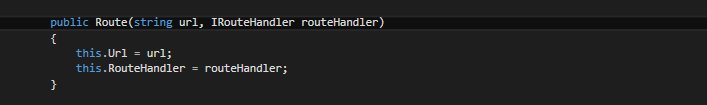
1. 既然我们定义了路由模板,那么必定会module截获. (UrlRoutingModule)



做路由匹配找到正确的路由。



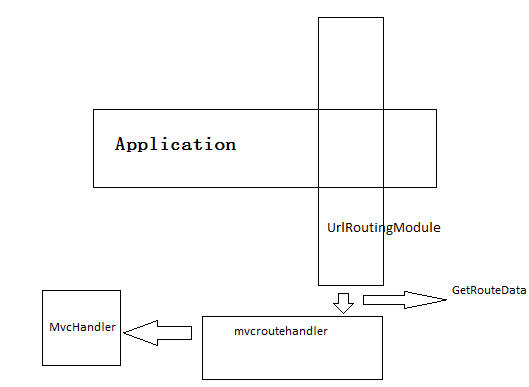
RouteHandler赋值：



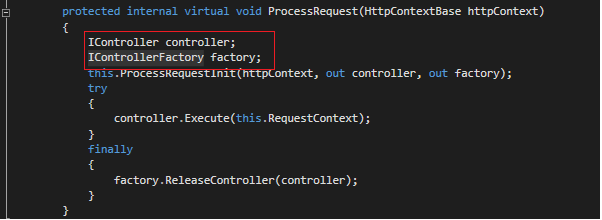
=>IRouteHandler routeHandler = routeData.RouteHandler=>【mvcroutehandler】

=>mvcRouteHandler = > MvcHandler(requestContext)

截获流程如下:

在UrlRoutingModule类中注册app事件。

获取到了MvcHandler过后,我们将得到它的PR函数,里面获取controller。



## 从源码角度讲解MVC的Route以及如何自定义IRouteConstraint约束

1. Router的讲解
2. 路由模板的匹配

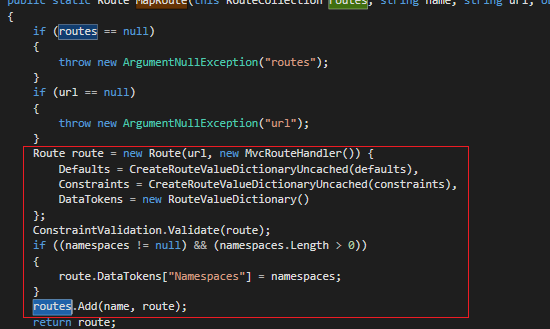
添加路由我们用MapRoute,剔除的路由我们用IgnoreRoute.

1. 简单看下MapRoute和IgnoreRoute的源码

MapRoute源码如下[mvcRouteHandler]

=>RouteTable[RouteCollection]

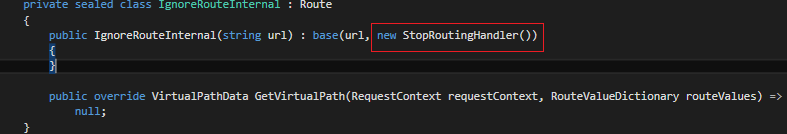
=>Route【就是我们构建的路由模板】



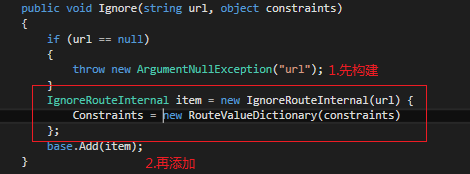
Routes.Add(name,route)加入到我们的collection中去。

IgnoreRoute的源码:

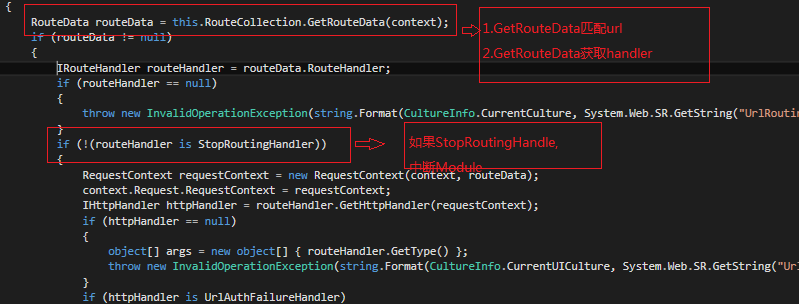
其它的handler为StopRoutingHandler



同样,它也是先构建再添加:



(1).通过源码我们发现,Ignore和我们的map都是构建route然后添加routecollection中去,唯一不一样的是maproute我们加入的handler是mvcroutehandler,Ignore我们的handler是StopRoutingHandler.

http=>foreach会找Route,如果Route被匹配,同时还应该去找该Route的handler是什么类型，如果是StopRoutingHandler我们直接跳出.GetRouteDate. 

1. MapRoute中的参数constraints怎么用？

其本质对url模板的参数进行限定.

比如说:

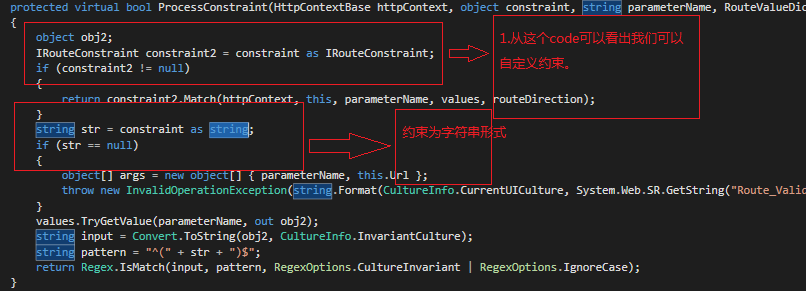
defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

这里的模板我们可以对id,action做限定。还可以对controller进行限定.

1. 比如我想做到输入的url模板中id只能是整数。



1. 如上图所示,我们给id定于了约束,字符串约束。



return Regex.IsMatch(input, pattern, RegexOptions.CultureInvariant | RegexOptions.IgnoreCase); 根据我们定义的条件返回false。

自定义约束如下:



## 对众多AcrionResult的深入源码分析【上】

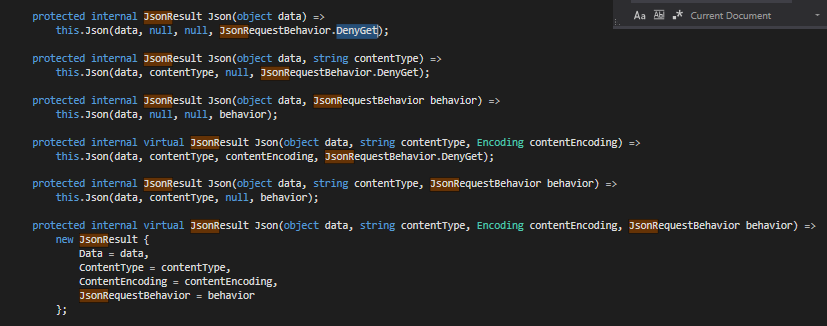
1. ActionResulit
2. 到底mvc中有多少Result

二.非ViewResult的其他Result

介绍一些常用的result

1. JosonResult

重下面看默认是拒绝Get请求的.我们可以通过Ajax去访问。



public ActionResult Index()

{

return Json(new { name="Yw",action="mvc"}, JsonRequestBehavior.AllowGet);

// return View();

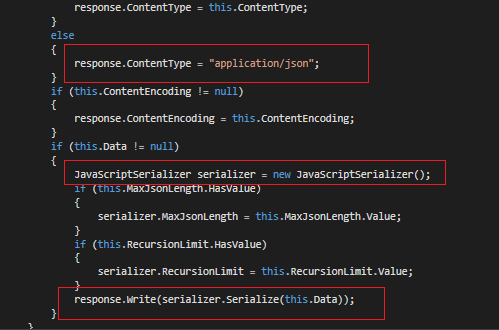
}

研究其源码:

1.构造JsonResult对象

2.调用其ExecuteResult方法

代码如下:



1. JavaScriptResult

例子代码如下:

public ActionResult Index()

{

return JavaScript("var i = 10;alert(i);");

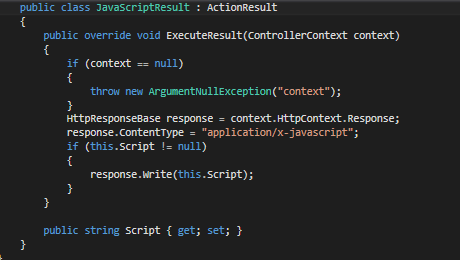
}

分析其源码:

1. new 了一个JavaScript对象



1. 执行JavaScriptResult对象中的Excute方法

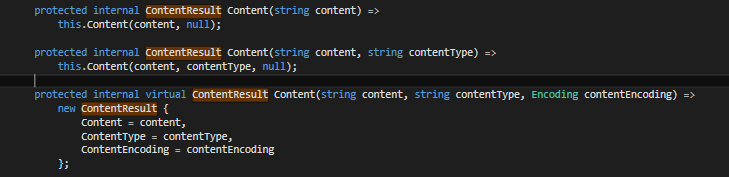


1. ContentResult

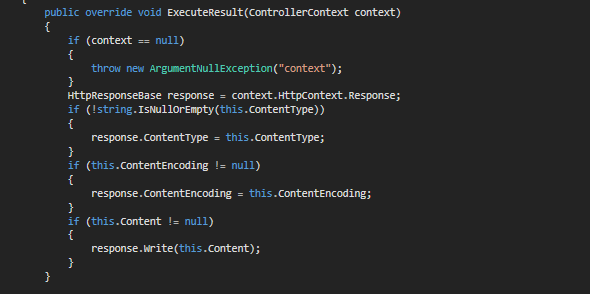
例子代码如下:

return Content("你好!Mvc");

1.构造Content对象

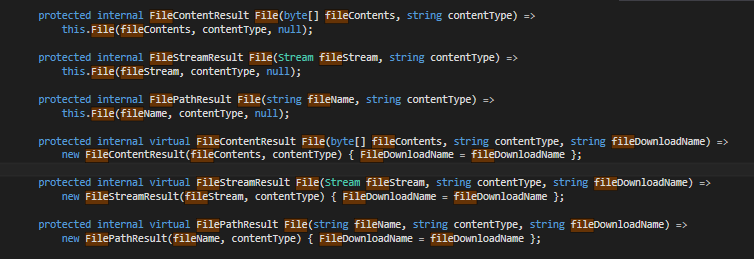


2.执行Excute方法

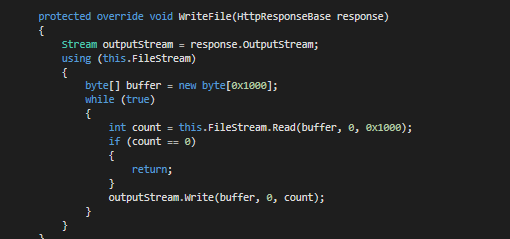


1. EmptyResult
2. HttpStateResult
3. HttpNotFoundResult
4. FileResult

代码如下:



调用：



1. Redirect

最终执行:context.HttpContext.Response.Redirect(text,false);

## 对众多AcrionResult的深入源码分析【下】

1. ViewResult

这个是Result家族中最复杂的。

因为这个设计到了Razor模板

Code + Razor模板最后生成 🡺html 【output】

二.Razor模板最后生成了什么?

C#中万事万物皆为对象(类)

Index.cshtml最后会转换成Index\_html.cs

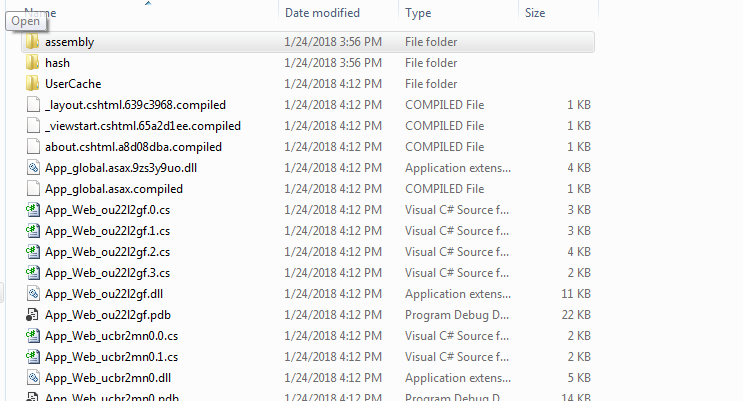
我们发现,其实我们的asax会生成一个dll…

ViewStart.cshtml也生成一个dll

Home下面的每个action都生成了一个类。Dll

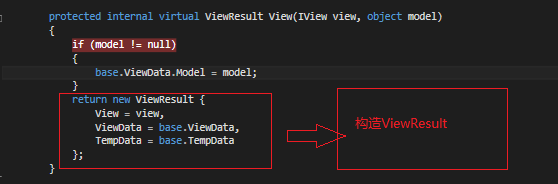
Shard文件夹也生产了一个DLL,每个cshtml变成一个类。

路径: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Temporary ASP.NET Files\root\cb9c5e39\74e5ef94

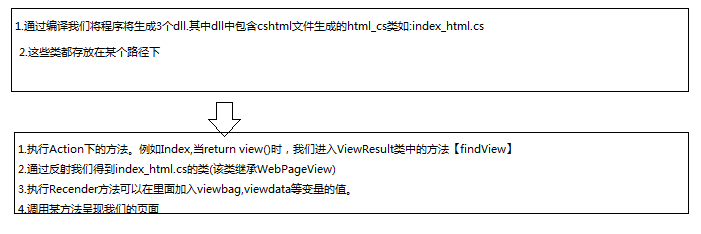


我们发现我们每个cshtml都会生成一个继承WebViewPage<object>的这个类。

最后通过excute生成我们的页面。

源码分析我们的ViewResult

总结如下:



## 第八章Razor视图基础结构以及ViewStart解析

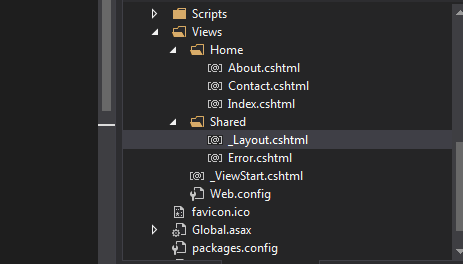
1. View的基本结构

1. Share文件夹

<1>. \_layout.cshtml 这个模板页面。相当于我们webform中的master.

<2>. 局部页面\_loginpariter.cshtml相当于我们webform的ascx用户控件

<3>.Error错误页面..webform中的错误页面



Razor ===》 aspx引擎

2.\_ViewStart.cshtml 先于我们的页面执行

<1>有什么好处呢？统一设置。

<2>坏处？如果在view的根目录下设置会导致其他文件夹受到影响。

3.viewstart也是一个类文件.

4.Razor的语法

[1.@{}](mailto:1.@%7b%7d)

[2.@.Html](mailto:2.@.Html)(调用类)

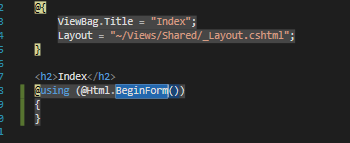
1. @DateTime.Now.Year

## 第九章RenderAction和Action以及Partial 和RenderPartial的认识

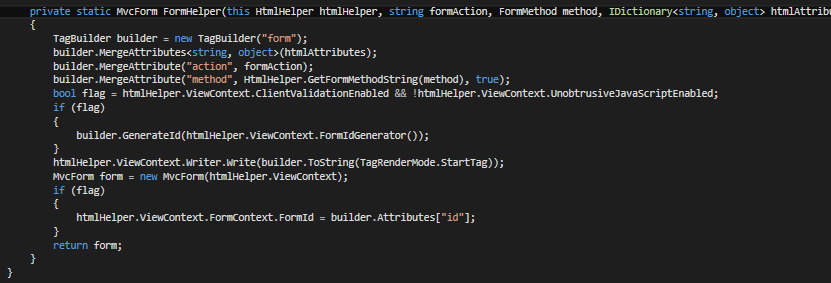
一.HtmlHelper

1.我们常用的方法

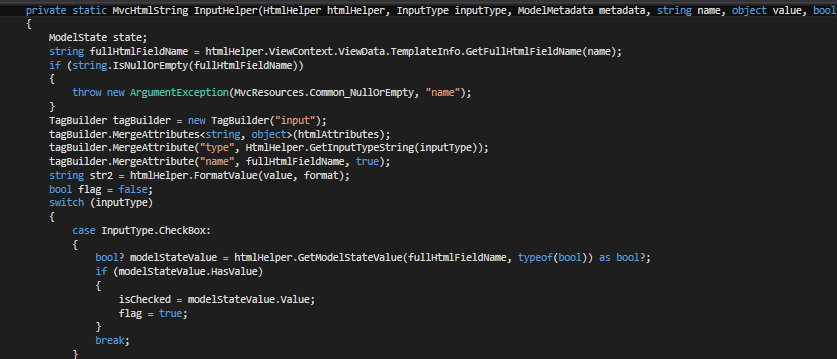
<1>@Html.BeginForm生成form表单



其后面的源代码为:



<2>@Html.TextBox("username")



<3> @Html.TextBoxFor(m => m.Name);强类型

其实都是些静态的target标记

二.Action 和 RenderAction(Action打交道,适合加上自己的业务逻辑)

都叫用户控件。

1. 一个有返回值(将流变成字符串)
2. 一个没有返回值(走了一遍管道流)

我们是否能通过url直接访问用户控件？？我们最好不让外界直接访问用户控件。

可以加特性控制[ChildActionOnly]

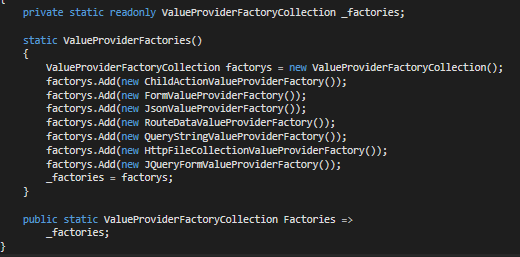
Partial 和 RenderPartial区别(View打交道,不适合加业务逻辑)

一个是流的形式一个字符串形式。

## 第十章 mvc中ValueProvider源码分析

1. ValueProvider是一个字典集合

ValueProviderFactoryCollection=》灌入了下面的内容



ValueProvider就是MVC中获取外界数据的一个大一统接口.

在以前的webform中我们获取外界的数据:

Form, Querystring, files..

二. 在MVC中获取外界数据的几种途径.

1. QueryStringValueProvider

2. FormValueProvider

3. HttpFileCollectionValueProvider

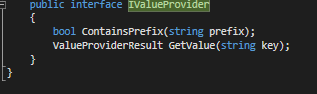
4. JQueryFormValueProvider

5. RouteDataValueProvider

6. JsonValueProvider

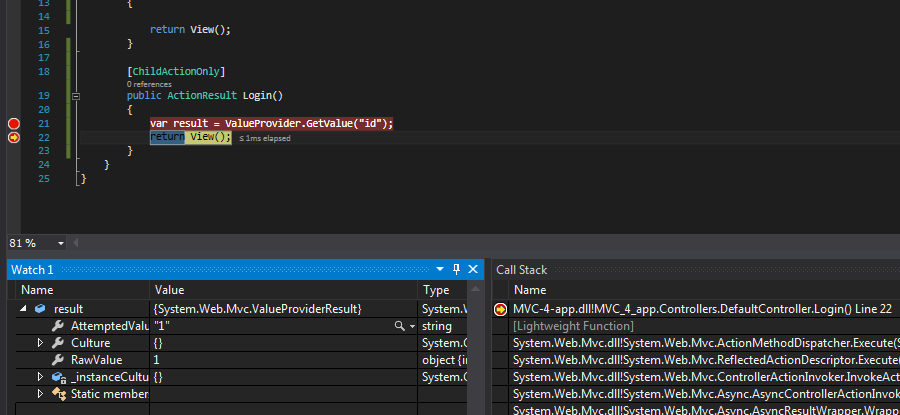
7. ChildActionValueProvider

1. 针对这么多的获取数据方式,这个时候我们需要有一种统一的管理方式那就是ValueProvider 🡺IValueProvider



ValueProvider实现了一个IValueProvider接口,里面有一个GetValue方法。

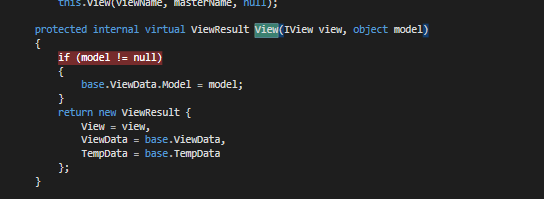
它的返回值是ValureProviderResult我们可以通过Convert进行转换。

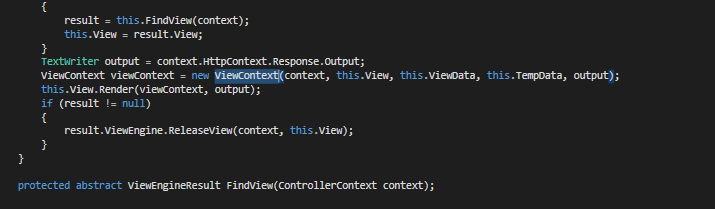


其实ValueProvider不是给我们用的,在源代码defaultmodulebinder中大量被用到。

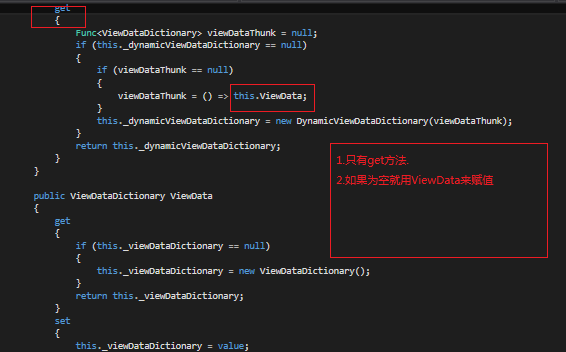
## 第十一章 从源码的角度对ViewData,ViewBag以及TempData原理分析

1. MVC中常用的三个集合 ViewData,ViewBag,TempData
2. ViewData (核心代码如下)





1. ViewBag(其实和ViewData是一个东西)



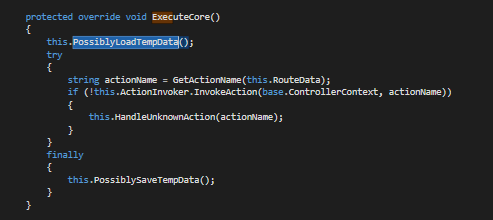
1. TempData

起到了临时变量的作用

可以存储一次,只能读取一次…(第二次读取将不会有数据)

<1>是一个string object的字符

<2>action执行前后,都会对temp进行操作…(其实是用session来进行存储的)

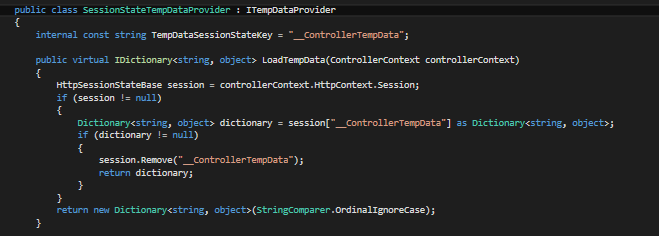


## 第十二章 跨Session实现一个自定义的TempData源码

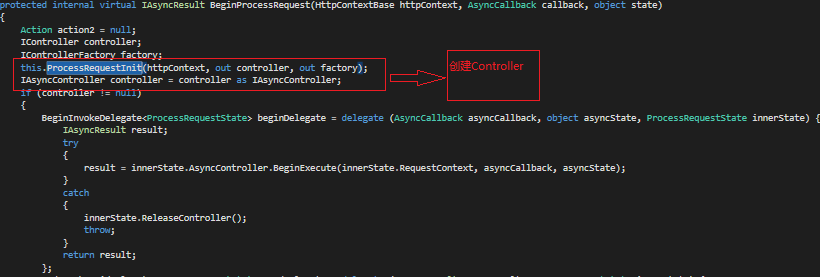
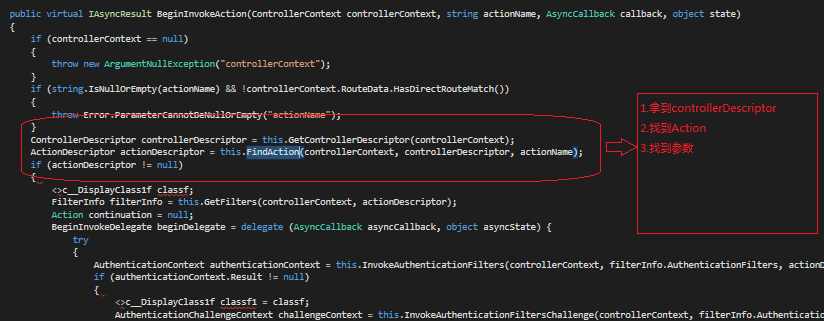
1. TempData是用Session来实现的,既然是Session默认是进程或者线程方式的。这样的Session是没办法实现跨平台的。Mvc在多台机器中部署,那么是怎么做到session在多台机器中共存?

涉及到了分布式的存储,sqlserver/mysql关系型数据库.也可以用nosql/mongodb,redis来存储.

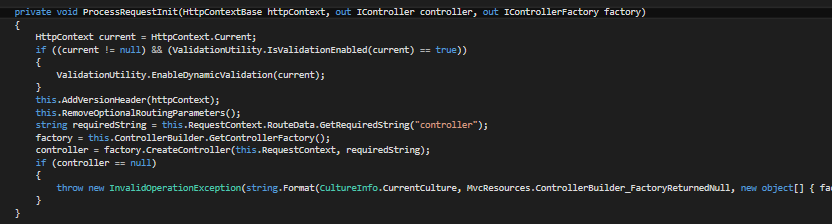
为了掩饰我们用本地的cach来替换。



实现需要继承ITempDataProvider.



核心代码,通过MVChandler拿到Controller



管道流:

Iis=>urlroutingmodule=>urlroutinghandler=>mvchandler=>controllerfactory=>controller=>actioninvoke=>action=>parmeter=>method=>actioninvoke=>output browser…..

## 第十三章ReflectedController-Action-ParameterDescriptor角色扮演分析

一. ControllerDescriptor , ActionDescriptor , ParameterDescriptor

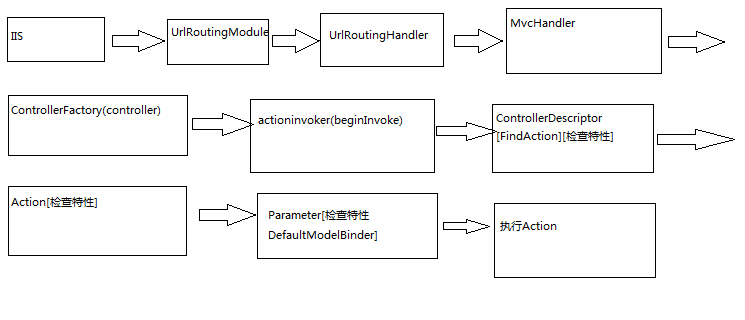
因为这个三个Descriptor在MVC的管道及激活Controller的激活中扮演这很重要的角色.这三个类对Controller,Action,Parameter进行了反射的描述.

Controller上面可能有Attribute… 【Authrozie,HandlerError,Route】

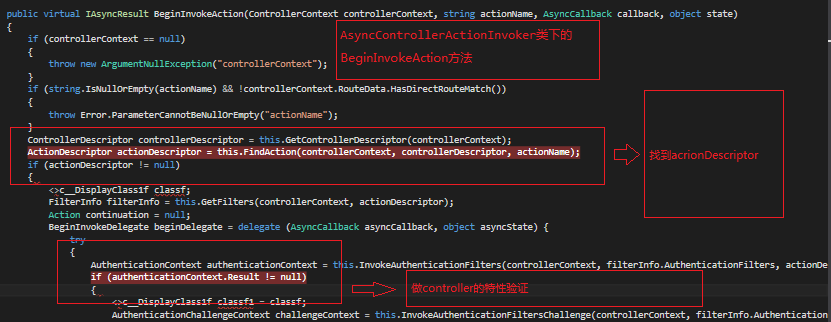
Action 上面可能有Attribute…

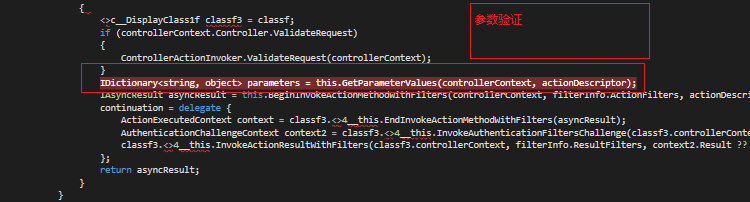
Paramater 上面可能有Attribute…

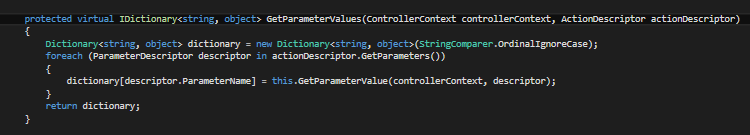
可以通过三个类来获取他们的特性,来判断是否可以执行响应的方法。



二．通过源代码加深理解。







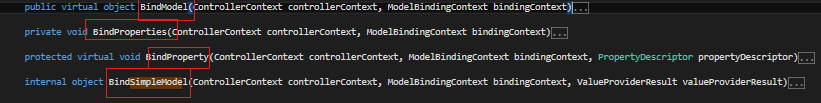
## 第十四章 模型绑定器ModuleBinder之简单和复杂类型

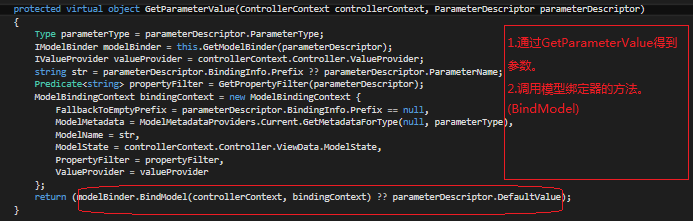
1. DefaultModelBinder

Mvc中就是通过它进行模型的绑定,将ValueProvider里面的数据映射成类.【复杂类型】,int/string[简单类型],array/list【集合类型】.

ValueProvider mvc 外界的数据都是通过它来获取的。

ValueProvider==》通过DefaultModelBinder===🡺将他映射成类或者集合。



二．怎么调到用BindModel 

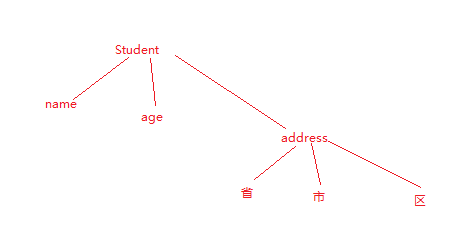
三.绑定机制.

1.简单类型

直接从ValueProvider中直接获取。

2.复杂类型

Student => [name,age,address[省,市,区]]（递归调用）



BindComplexModel=> BindSimpleModel递归调用（BindSimpleModel）

有几个属性就绑定几次SimpleModel.

## 第十五章BindAttribute对Parameterde 的限制及源码分析

1. BindAttribute和自定义一下DefaultModelbinder…

既然有这个,就有他的用处。。。。

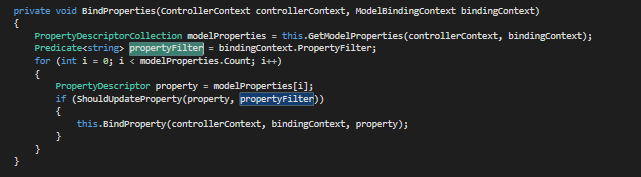
BindAttribute应用到参数级别就是用来控制参数的.

Public ActionResult Index([Bind(Exclude=”Age”)] Student stu)

不希望Age参数传上来.

BindAttribute 三个属性进行修饰:

1. Exclude
2. Include
3. Prefix



## 第十六章 实体验证之DataAnnotationsModel介绍

一.数据批注:

DataAnnotationsModel

1. Required
2. StringLength
3. Compare 新密码和老密码之前的对比
4. RegularExpression
5. Phone
6. Range
7. MaxLength

二.实现方式:

举例: public class RequiredAttribute : ValidationAttribute

首先应该是实现ValidationAttribute接口

[\_\_DynamicallyInvokable]

public override bool IsValid(object value)

{

if (value == null)

{

return false;

}

string str = value as string;

if ((str != null) && !this.AllowEmptyStrings)

{

return (str.Trim().Length > 0);

}

return true;

}

这个代码就是判断的核心代码。

1. 这些特性的场景

用于客户端和服务器端的验证机制….

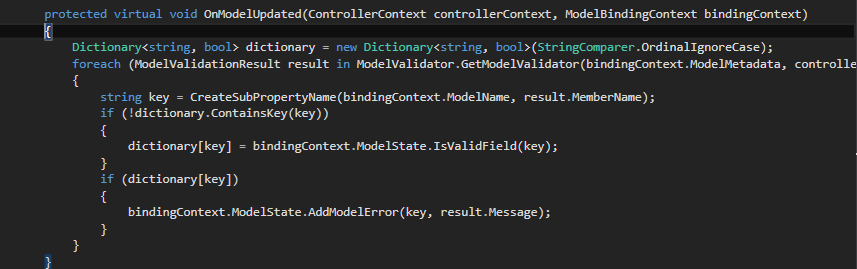
Mvc将页面的业务逻辑全部用”特性”的形式加以标记。。。

这样的话,可以灵活的控制.我们不倾向将批注用在客户端验证。

客户端一般有自己的验证框架。但是可以用批注做服务器端验证。

@Html.ValidationMessageFor做客户端验证信息.

var state = ModelState.IsValid;做服务器端验证。

做验证的核心代码。

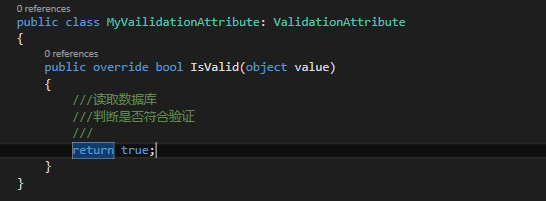
我们发现一个缺陷,目前我们验证的大多都是基于属性本身,而做不到类的范围。

<<1>>我希望对属性的验证是自定义逻辑,我们想对model中的几个值进行综合验证,而不是针对某个值进行验证。

## 第十七章 自定义验证之VaildationAttribute及Model级验证VaildationObject

1. 解决属性的自定义逻辑验证。

既然都继承了VaildationAttribute,并且实现了isValid方法。



二.如何在自定逻辑中获取当前类的中其他字段。

因为我的验证需要更具当面model中的字段进行综合判断.

实现起来很简单,需要我们的model继承接口就ok了。

IValidatableObject接口……Va

## 第十八章 IActionFilter过滤器

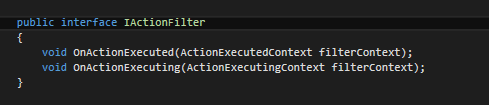
其实在,MVC中所有的action设计器都是采用aop的方式进行拦截的

需求，我们想在action的执行前后进行参数的拦截…【记日志】

其实我们的Controller已经实现了ActionFilter…..

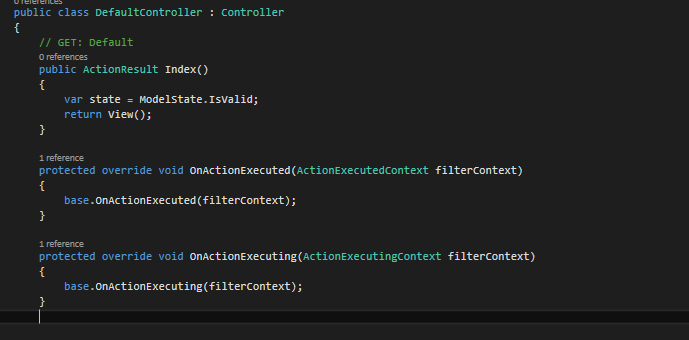
public abstract class Controller : ControllerBase, IActionFilter, IAuthenticationFilter, IAuthorizationFilter, IDisposable, IExceptionFilter, IResultFilter, IAsyncController, IController, IAsyncManagerContainer

二. IActionFilter

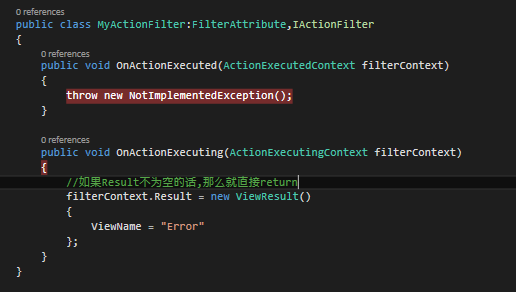


中间应该是ActionInvoke.

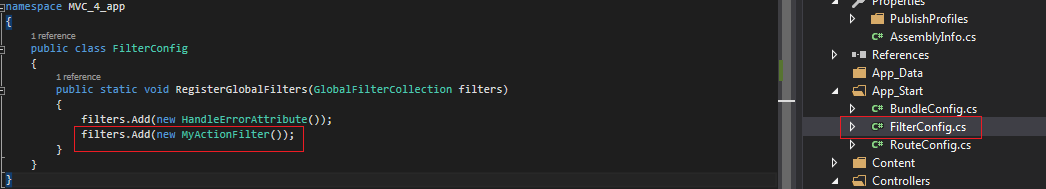
1. 因为Controller继承了IActionFilter所以可以重写其接口的方法：



1. 做成原子方法.



1. 我们可以在全局上面做ActionFilter(全局注入)



## 第十九章 Controller之IResultFilter,IExceptionFilter,HandleError拦截器分析

1. IResultFilter,IExceptionFilter

通用也是采用了aop的方式进行了切面编程。

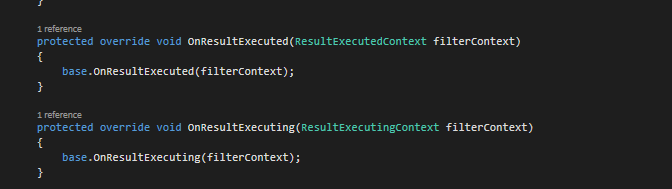
IResultFilter就是我们的view执行的前后。

ActionResult=🡺ViewResult,JsonResult,ContentResult…..

=>ExcuteResult方法（在这个方法的前和后进行执行）

public abstract class Controller : ControllerBase, IActionFilter, IAuthenticationFilter, IAuthorizationFilter, IDisposable, IExceptionFilter, IResultFilter, IAsyncController, IController, IAsyncManagerContainer

IResultFilter(接口)



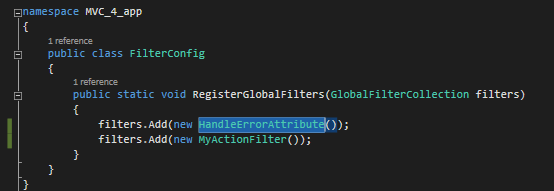
IResultFilter和IActionFilter是一样的。

（1）（2）（3）

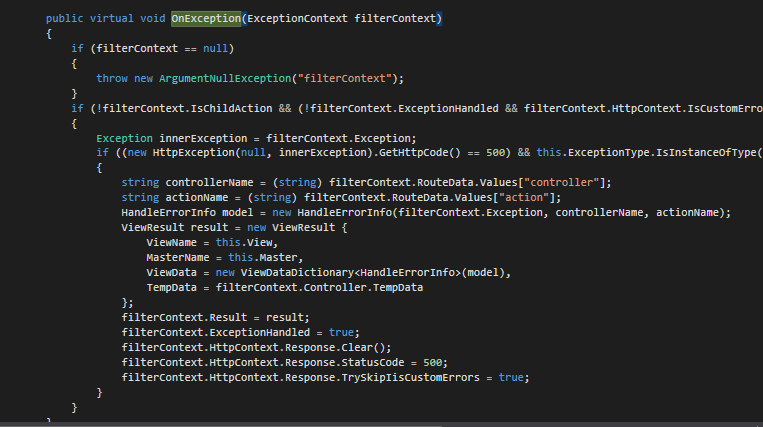
IExceptionFilter

public abstract class Controller : ControllerBase, IActionFilter, IAuthenticationFilter, IAuthorizationFilter, IDisposable, IExceptionFilter, IResultFilter, IAsyncController, IController, IAsyncManagerContainer

在MVC中有个全局的默认的IExceptionFilter那就是HandleError



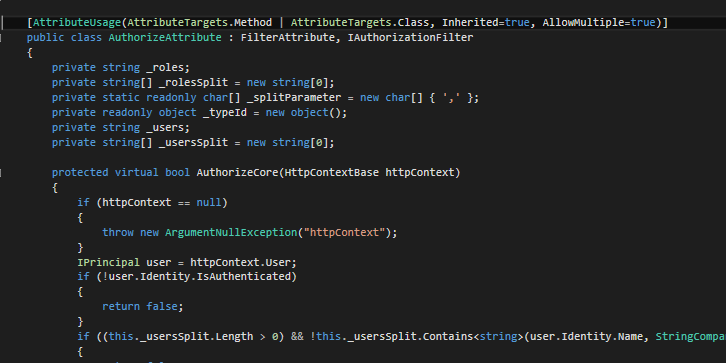
实现类如下:MVC已经提供好



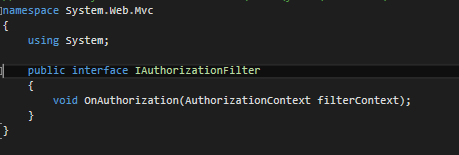
## 第二十章 form验证之Authorize及高度定制AuthorizeAttribute

1. Authorize特性。

这个标记在WebForm中是有的。并不是MVC专有的。也就是说是一个很古老的东西。

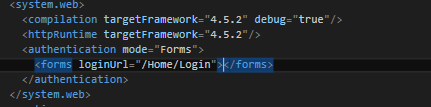


需实现:

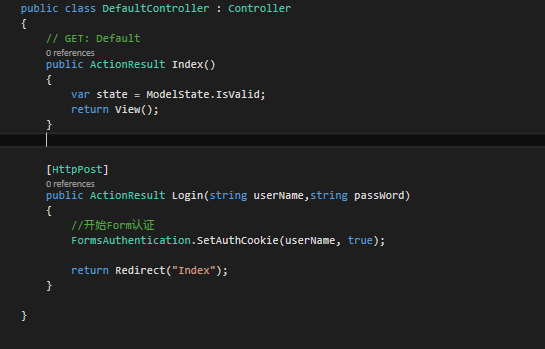


二.简单的实现

1.定义认证:



2.写一个Login的方法



这个默认的表单验证太low…因为我们的web服务器是多台。

现在基本上是多台website….怎么共享我们的cookie用户信息。

这个就是我们所谓的单点登录.

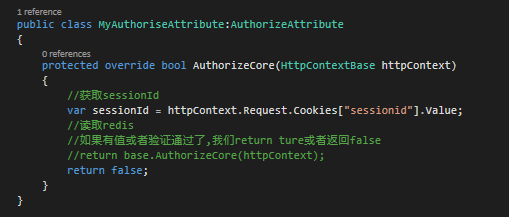
Niginx🡺web1,web2,web3

【redis,mongodb】

SessionId,或者我们分配一个OpenId给客户端。

用户通过这个OpenId来访问我们的Web这样就不能使用form验证了

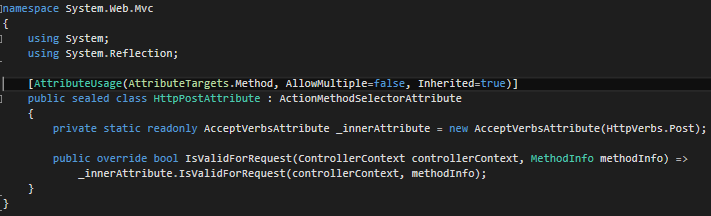


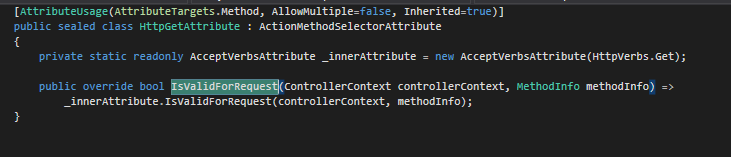


## 第二十一章 Action筛选器之ActionMethodSelectorAttribute介绍

1. ActionMethodSelectorAttribute
2. httppost
3. httpget
4. actionName
5. NonAction

等等。。。





1. Httpget的实现原理:

public sealed class HttpGetAttribute : ActionMethodSelectorAttribute

{

private static readonly AcceptVerbsAttribute \_innerAttribute = new AcceptVerbsAttribute(HttpVerbs.Get);

public override bool IsValidForRequest(ControllerContext controllerContext, MethodInfo methodInfo) =>

\_innerAttribute.IsValidForRequest(controllerContext, methodInfo);

}

}

从代码的角度上来看我们HttpGet其实是调用AcceptVerbsAttribute(HttpVerbs.Get);

