MVC组件设计

[1引言 2](#_Toc399361517)

[1.1背景 2](#_Toc399361518)

[1.2 MVC编程模型 2](#_Toc399361519)

[1.3 框架和设计模式的区别 2](#_Toc399361520)

[2 System.Web.Mvc V 5.2.0.0 3](#_Toc399361521)

[2.1 Routing组件（路由选择） 3](#_Toc399361522)

[2.2 Controller（控制器） 4](#_Toc399361523)

[2.3 控制器可用的ActionResult 4](#_Toc399361524)

[2.4 View（视图） 6](#_Toc399361525)

[2.5 ViewModel（视图模型） 7](#_Toc399361526)

[HtmlHelper帮助类 7](#_Toc399361527)

[UrlHelper帮助类 7](#_Toc399361528)

[自定义控件 8](#_Toc399361529)

[2.6 Filters（过滤器） 8](#_Toc399361530)

[2.7 Action的要求 11](#_Toc399361531)

[3 MVC组件扩展（MvcExtensions） 11](#_Toc399361532)

# 1引言

## 1.1背景

## 1.2 MVC编程模型

MVC 是三个 ASP.NET 开发模型之一。

MVC 是用于构建 web 应用程序的一种框架，使用 MVC (Model View Controller) 设计：

* + Model（模型）表示应用程序核心（比如数据库记录列表）
  + View（视图）对数据（数据库记录）进行显示
  + Controller（控制器）处理输入（写入数据库记录）

MVC 模型同时提供对 HTML、CSS 以及 JavaScript 的完整控制。

MVC 模型通过三个逻辑层来定义 web 应用程序：

* + business layer（业务层、模型逻辑）
  + display layer（显示层、视图逻辑）
  + input control（输入控件、控制器逻辑）

**模型（Model）**

模型（Model）是应用程序中用于处理应用程序数据逻辑的部分。

通常模型对象在数据库中存取数据。

**View（视图）**

View（视图）是应用程序中处理数据显示的部分。

通常从模型数据中创建视图。

**控制器**

控制器是应用程序中处理用户交互的部分。

通常控制器从视图读取数据、控制用户输入，并向模型发送数据数据。

MVC 的这种拆分有助于我们管理复杂的应用程序，因为您能够在同一时间关注一个方面。例如，您可以在不依赖业务逻辑的情况下对视图进行设计。同时对应用程序的设计也更加容易。

MVC 的这种拆分同时也简化了分组开发。不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

## 1.3 框架和设计模式的区别

有很多程序员往往把框架模式和设计模式混淆，认为MVC是一种设计模式。实际上它们完全是不同的概念。

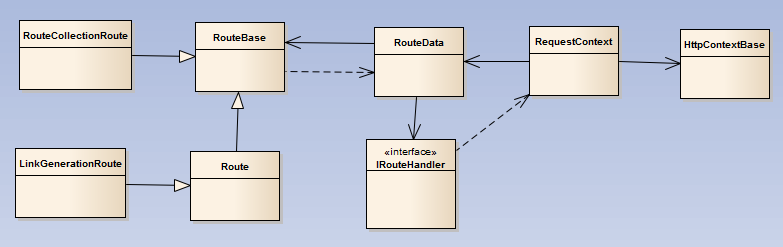
框架、设计模式这两个概念总容易被混淆，其实它们之间还是有区别的。框架通常是代码重用，而设计模式是设计重用，架构则介于两者之间，部分代码重用，部分设计重用，有时分析也可重用。在软件生产中有三种级别的重用：内部重用，即在同一应用中能公共使用的抽象块;代码重用，即将通用模块组合成库或工具集，以便在多个应用和领域都能使用；应用框架的重用，即为专用领域提供通用的或现成的基础结构，以获得最高级别的重用性。

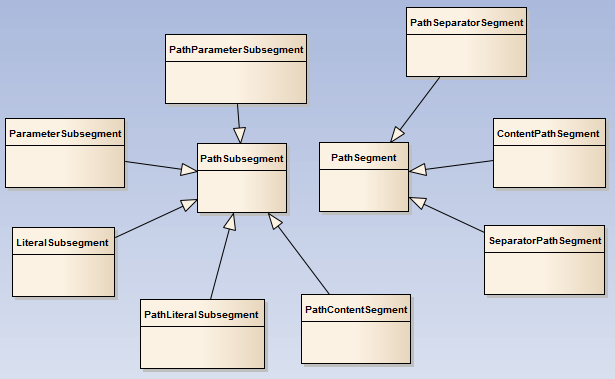
框架与设计模式虽然相似，但却有着根本的不同。设计模式是对在某种环境中反复出现的问题以及解决该问题的方案的描述，它比框架更抽象；框架可以用代码表示，也能直接执行或复用，而对模式而言只有实例才能用代码表示;设计模式是比框架更小的元素，一个框架中往往含有一个或多个设计模式，框架总是针对某一特定应用领域，但同一模式却可适用于各种应用。可以说，框架是软件，而设计模式是软件的知识。

# 2 System.Web.Mvc V 5.2.0.0

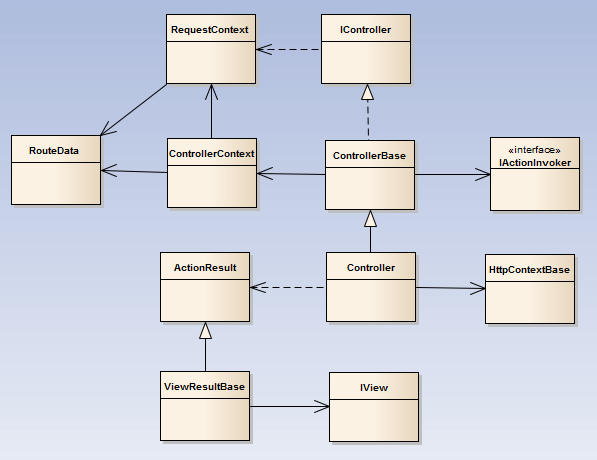
## 2.1 Routing组件（路由选择）

Routing的作用就是负责分析Url





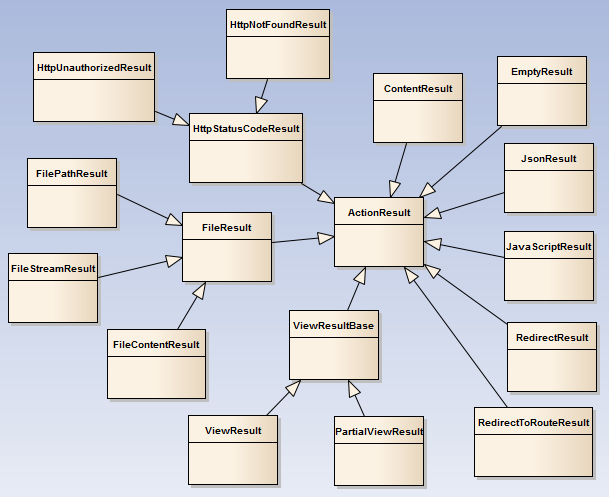
## 2.2 Controller（控制器）



## 2.3 控制器可用的ActionResult

  ActionResult是控制器方法执行后返回的结果类型，控制器方法可以返回一个直接或间接从ActionResult抽象类继承的类型，如果返回的是非ActionResult类型，控制器将会将结果转换为一个ContentResult类型。默认的ControllerActionInvoker调用ActionResult.ExecuteResult方法生成应答结果。

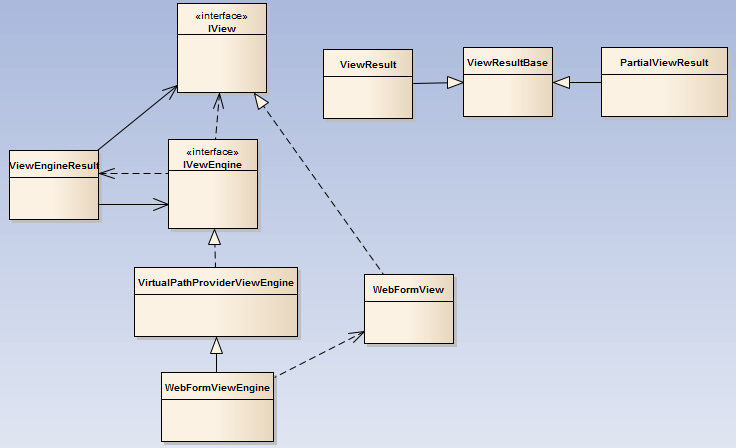
    MVC中实现的默认ActionResult如下：



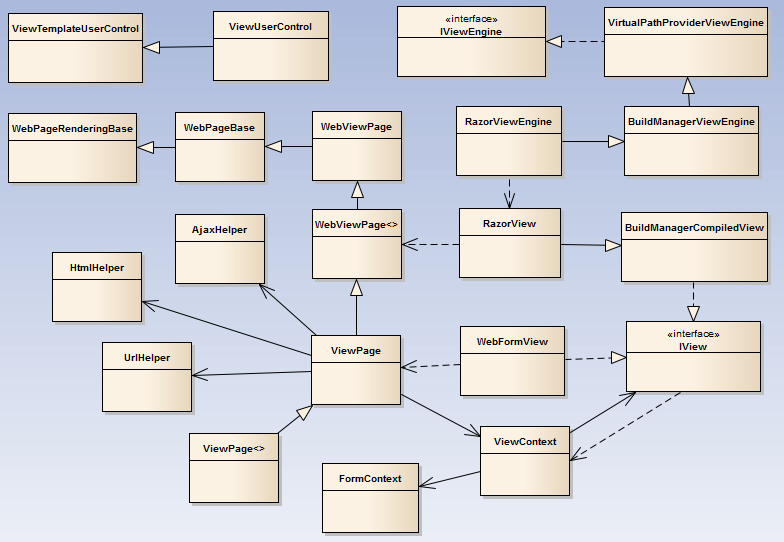
1、**ContentResult**: 返回简单的纯文本内容，可通过ContentType属性指定应答文档类型，通过ContentEncoding属性指定应答文档的字符编码。可通过Controller类中的Content方法便捷地返回ContentResult对象。如果控制器方法返回非ActionResult对象，MVC将简单地以返回对象的ToString()内容为基础产生一个ContentResult对象。  
2、**EmptyResult**: 返回一个空的结果。如果控制器方法返回一个null，MVC将其转换成EmptyResult对象。  
3、**RedirectResult**: 表示一个连接跳转，相当于ASP.NET中的Response.Redirect方法。对应的Controller方法为Redirect。  
4、**RedirectToRouteResult**：同样表示一个调转，MVC会根据我们指定的路由名称或路由信息(RouteValueDictionary)来生成Url地址，然后调用Response.Redirect跳转。对应的Controller方法为RedirectToAction和RedirectToRoute。  
5、**ViewResult**: 表示一个视图结果，它根据视图模板产生应答内容。对应Controller方法为View。  
6、**PartialViewResult**: 表示一个部分视图结果，与ViewResult本质上一致，只是部分视图不支持母版，对应于ASP.NET，ViewResult相当于一个Page，而PartialViewResult则相当于一个UserControl。它对应的Controller方法为PartialView。  
7、**HttpUnauthorizedResult**: 表示一个未经授权访问的错误。MVC会向客户端发送一个401的应答状态。如果在web.config中开启了表单验证(authentication mode="Forms")，则401状态会将Url转向指定的loginUrl链接。  
8、**JavaScriptResult**: 本质上是一个文本内容，只是将Response.ContentType设置为 application/x-javascript，此结果应该和MicrosoftMvcAjax.js脚本配合使用，客户端接收到Ajax应答后，将判断Response.ContentType的值，如果是application/x-javascript,则直接eval执行返回的应答内容。此结果类型对应的Controller方法为JavaScript。  
9、**JsonResult**: 表示一个JSON结果。MVC将Response.ContentType设置为application/json，并通过JavaScriptSerializer类将指定对象序列化为Json表示方式。需要注意，默认情况下，MVC不允许GET请求返回JSON结果，要解除此限制，在生成JsonResult对象时，将其JsonRequestBehavior属性设置为JsonRequestBehavior.AllowGet。此结果对应的Controller方法为Json。  
10、**FilePathResult、FileContentResult、FileStreamResult**: 这三个类继承于FileResult，表示一个文件内容，三者的区别在于，FilePath通过路径传送文件到客户端，FileContent通过二进制数据的方式，而FileStream是通过Stream的方式来传送。Controller为这三个文件结果类型提供了一个名为File的重载方法。

**通过直接或间接地从ActionResult继承，可实现自定义的结果类型.**

## 2.4 View（视图）



## 2.5 ViewModel（视图模型）



**HtmlHelper帮助类**

用法大全

http://www.cnblogs.com/jyan/archive/2012/07/23/2604474.html FormExtensions,InputExtensions,LinkExtensions,SelectExtensions,TextAreaExtensions,ValidationExtensions,RenderPartialExtensions等7个静态类，他们全部是是采用拓展方法来实现的。

**InputExtensions类**：

主要有5种类型的扩展方法，分别用于CheckBox控件、Hidden控件、Pass控件、RadionButton控件和TextBox控件LinkExtensions类：

该类主要用于生成相关链接，主要扩展了ActionLink和RouteLink方法。

**ActionLink：**

扩展方法主要实现一个连接，共有十个重载方法。

**UrlHelper帮助类**

看类名也都知道这个类是用来成URL在 ASP .NET MVC应用程序中。

UrlHelper提供了四个非常常用的四个方法。

1. Action方法通过提供Controller,Action和各种参数生成一个URL，
2. Content方法是将一个虚拟的，相对的路径转换到应用程序的绝对路径，
3. Encode方法是对 URL地址进行加密，与Server.Encode方法一样。
4. RouteUrl方法是提供在当前应用程序中规定的路由规则中匹配出URL。

另外还有两个属性，分别是RequestContext和RouteCollection两个属性，分别指的是包含HTTP上下文和RouteData两个属性，另外，RouteCollection是整个当前应用程序中规定的路由规则。

### 自定义控件

微软提供的HtmlHelper已经是足够大部分开发人员使用了，但是有一些功能要用微软提供的HtmlHelper可能还不满足要求。接下来就谈谈如何自定义的过程。

首先自定义的方法就是对HtmlHelper对象的扩展。

扩展方法实现的三要素：1、静态类　2、静态方法　3、this关键字

1、先定义一个类，例如：MyHtmlHelper：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace MvcApplicationFirstDome.Models {

//静态类

public static class MyHtmlHelper

{

//静态方法

}

}

2、假设要扩展的方式是GetSpan，作用就是当你传入参数时，内部封装了之后返回结果，代码如下。注意在MyHtmlHelper类中要引用using System.Web.Mvc命名空间。

//静态方法

public static string GetSpan(this HtmlHelper htmlHelper,string text)

{

return "<span style='color:Red'>"+text+"</span>";

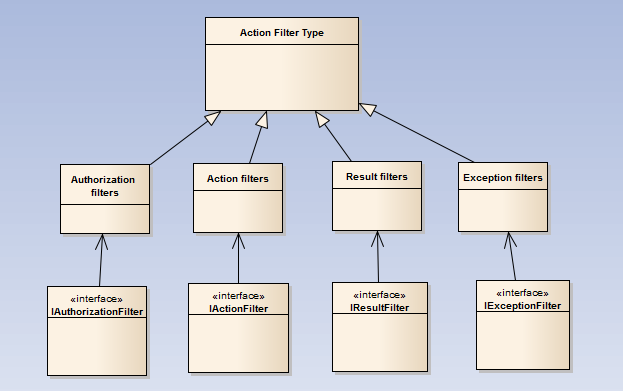
}

经过上面两步之后HtmlHelper的扩展方法GetSpan基本可以使用了，接下来就讲解如何在页面调用了。

## 2.6 Filters（过滤器）

ASP.NET MVC Framework支持四种不同类型的Filter：

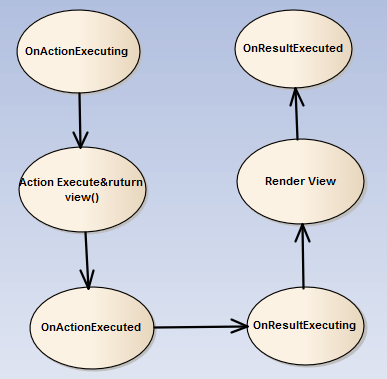
1. Authorization filters – 实现IAuthorizationFilter接口的属性.
2. Action filters – 实现IActionFilter接口的属性.
3. Result filters – 实现IResultFilter接口的属性.
4. Exception filters – 实现IExceptionFilter接口的属性.



ActionFilterAttribute类有以下的方法可以重写：

* OnActionExecuting – 在controller action执行之前调用
* OnActionExecuted – 在controller action执行之后调用
* OnResultExecuting – 在controller action result执行之前调用
* OnResultExecuted – 在controller action result执行之后调用

执行的顺序如下图：



创建 Action filter 必须实现 IActionFilter 接口，它定义了两个方法：

|  |  |  |
| --- | --- | --- |
| 方法名 | 调用时机 | 可进行的操作 |
| OnActionExecuting | Action方法执行之前 | 可以给 filterContext.Result 赋值，以阻止 Action 的执行 可以通过 filterContext.ActionParameters 查看或修改 Action 的参数 |
| OnActionExecuted | Action方法执行之后 | 可以通过 filterContext.Exception 获取 Action 执行时抛出的异常，并能把它标记为“已处理”：filterContext.ExceptionHandled = true。 可以查看通过filterContext.Result查看 Action 的执行结果，但是不能修改。 |

下面就使用 OnActionExecuting 可以在 Action 方法执行前 **能查看并修改参数值的特性** 来完成敏感信息的过滤功能。

**通过直接或间接地从ActionFilterAttribute继承，可实现自定义的结果类型.**

## 2.7 Action的要求

Action的要求  
• 必须是一个公有方法  
• 必须返回ActionResult类型  
• 必须是实例方法  
• 不能是范型方法  
• 没有标注NonActionAttribute  
• 不能被重载（overload）

# 3 MVC组件扩展（MvcExtensions）