目录

[1. MVC框架 1](#_Toc495925072)

[2. 简介 1](#_Toc495925073)

[3. MVC 编程模式 1](#_Toc495925074)

[4. 框架内容 2](#_Toc495925075)

[5. 区别 3](#_Toc495925076)

[6. 常见框架 3](#_Toc495925077)

[7. 特点 4](#_Toc495925078)

[8. 外界评价 5](#_Toc495925079)

[9. MVC控件 6](#_Toc495925080)

1. MVC模式

# MVC框架

[MVC](https://baike.baidu.com/item/MVC)全名是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。

# 简介

MVC开始是存在于桌面程序中的，M是指业务模型，V是指[用户界面](https://baike.baidu.com/item/%E7%94%A8%E6%88%B7%E7%95%8C%E9%9D%A2" \t "_blank)，C则是控制器，使用MVC的目的是将M和V的实现代码分离，从而使同一个程序可以使用不同的表现形式。比如一批统计数据可以分别用[柱状图](https://baike.baidu.com/item/%E6%9F%B1%E7%8A%B6%E5%9B%BE" \t "_blank)、[饼图](https://baike.baidu.com/item/%E9%A5%BC%E5%9B%BE)来表示。C存在的目的则是确保M和V的同步，一旦M改变，V应该同步更新。

模型－视图－控制器（[MVC](https://baike.baidu.com/item/MVC" \t "_blank)）是[Xerox PARC](https://baike.baidu.com/item/Xerox%20PARC" \t "_blank)在二十世纪八十年代为编程语言Smalltalk－80发明的一种[软件设计模式](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F" \t "_blank)，已被广泛使用。后来被推荐为[Oracle](https://baike.baidu.com/item/Oracle)旗下[Sun](https://baike.baidu.com/item/Sun" \t "_blank)公司[Java EE](https://baike.baidu.com/item/Java%20EE" \t "_blank)平台的设计模式，并且受到越来越多的使用[ColdFusion](https://baike.baidu.com/item/ColdFusion" \t "_blank)和[PHP](https://baike.baidu.com/item/PHP" \t "_blank)的开发者的欢迎。模型－视图－控制器模式是一个有用的[工具箱](https://baike.baidu.com/item/%E5%B7%A5%E5%85%B7%E7%AE%B1)，它有很多好处，但也有一些缺点。

# MVC 编程模式

MVC 是一种使用 MVC（Model View Controller 模型-视图-控制器）设计创建 Web 应用程序的模式：

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

MVC 模式同时提供了对 HTML、CSS 和 JavaScript 的完全控制。

**Model（模型）**是应用程序中用于处理应用程序数据逻辑的部分。  
　　通常模型对象负责在数据库中存取数据。

**View（视图）**是应用程序中处理数据显示的部分。  
　　通常视图是依据模型数据创建的。

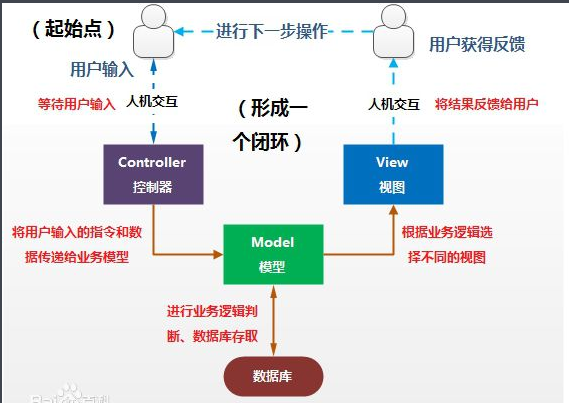
**Controller（控制器）**是应用程序中处理用户交互的部分。  
　　通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

MVC 分层有助于管理复杂的应用程序，因为您可以在一个时间内专门关注一个方面。例如，您可以在不依赖业务逻辑的情况下专注于视图设计。同时也让应用程序的测试更加容易。

MVC 分层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。

# 框架内容

MVC指MVC模式的某种框架，它强制性的使应用程序的输入、处理和[输出](https://baike.baidu.com/item/%E8%BE%93%E5%87%BA" \t "_blank)分开。使用MVC应用程序被分成三个核心部件：模型、[视图](https://baike.baidu.com/item/%E8%A7%86%E5%9B%BE" \t "_blank)、控制器。它们各自处理自己的任务。最典型的MVC就是JSP + [servlet](https://baike.baidu.com/item/servlet" \t "_blank) + [javabean](https://baike.baidu.com/item/javabean" \t "_blank)的模式。



## 4.1视图

视图是用户看到并与之交互的界面。对老式的Web应用程序来说，视图就是由[HTML](https://baike.baidu.com/item/HTML" \t "_blank)元素组成的界面，在新式的Web应用程序中，[HTML](https://baike.baidu.com/item/HTML" \t "_blank)依旧在视图中扮演着重要的角色，但一些新的技术已层出不穷，它们包括[Adobe Flash](https://baike.baidu.com/item/Adobe%20Flash" \t "_blank)和像[XHTML](https://baike.baidu.com/item/XHTML" \t "_blank)，[XML](https://baike.baidu.com/item/XML" \t "_blank)/[XSL](https://baike.baidu.com/item/XSL),[WML](https://baike.baidu.com/item/WML)等一些标识语言和[Web services](https://baike.baidu.com/item/Web%20services" \t "_blank).

MVC好处是它能为应用程序处理很多不同的[视图](https://baike.baidu.com/item/%E8%A7%86%E5%9B%BE" \t "_blank)。在视图中其实没有真正的处理发生，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。

## 4.2模型

模型表示企业数据和业务规则。在MVC的三个部件中，模型拥有最多的处理任务。例如它可能用像[EJB](https://baike.baidu.com/item/EJB)s和ColdFusion Components这样的构件对象来处理数据库，被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据，由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

## 4.3控制器

控制器接受用户的输入并调用模型和视图去完成用户的需求，所以当单击Web页面中的超链接和发送[HTML表单](https://baike.baidu.com/item/HTML%E8%A1%A8%E5%8D%95)时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

# 区别

**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。

框架模式有哪些？

MVC、MTV、[MVP](https://baike.baidu.com/item/MVP/3714550" \t "_blank)、CBD、[ORM](https://baike.baidu.com/item/ORM" \t "_blank)等等；

框架有哪些？

[C++](https://baike.baidu.com/item/C%2B%2B)语言的QT、MFC、gtk，Java语言的[SSH](https://baike.baidu.com/item/SSH" \t "_blank) 、**[SSI](https://baike.baidu.com/item/SSI" \t "_blank)**，[php](https://baike.baidu.com/item/php" \t "_blank)语言的 smarty(MVC模式)，[python](https://baike.baidu.com/item/python" \t "_blank)语言的django(MTV模式)等等

设计模式有哪些？

[工厂模式](https://baike.baidu.com/item/%E5%B7%A5%E5%8E%82%E6%A8%A1%E5%BC%8F)、适配器模式、策略模式等等

简而言之：框架是大智慧，用来对软件设计进行分工；设计模式是小技巧，对具体问题提出解决方案，以提高代码复用率，降低耦合度。

# 常见框架

## 6.1 Struts

[Struts](https://baike.baidu.com/item/Struts)是[Apache](https://baike.baidu.com/item/Apache" \t "_blank)软件基金下Jakarta项目的一部分。Struts框架的主要架构设计和开发者是Craig R.McClanahan。Struts 是Java Web MVC框架中不争的王者。经过长达九年的发展，Struts已经逐渐成长为一个稳定、成熟的框架，并且占有了MVC框架中最大的市场份额。但是Struts某些技术特性上已经落后于新兴的MVC框架。面对[Spring MVC](https://baike.baidu.com/item/Spring%20MVC" \t "_blank)、Webwork2这些设计更精密，扩展性更强的框架，Struts受到了前所未有的挑战。但站在产品开发的角度而言，Struts仍然是最稳妥的选择。

Struts有一组相互协作的类（组件）、[Servlet](https://baike.baidu.com/item/Servlet" \t "_blank)以及jsp tag lib组成。基于struts构架的web应用程序基本上符合JSP Model2的设计标准，可以说是MVC设计模式的一种变化类型。根据上面对framework的描述，很容易理解为什么说Struts是一个web framework，而不仅仅是一些标记库的组合。但 Struts 也包含了丰富的标记库和独立于该框架工作的实用程序类。Struts有其自己的控制器（[Controller](https://baike.baidu.com/item/Controller" \t "_blank)），同时整合了其他的一些技术去实现模型层（[Model](https://baike.baidu.com/item/Model" \t "_blank)）和视图层（View）。在模型层，Struts可以很容易的与数据访问技术相结合，包括[EJB](https://baike.baidu.com/item/EJB)，[JDBC](https://baike.baidu.com/item/JDBC" \t "_blank)和Object Relation Bridge。在视图层，Struts能够与JSP, Velocity Templates,XSL等等这些表示层组件相结合。

## 6.2 Spring

[Spring](https://baike.baidu.com/item/Spring)实际上是Expert One-on-One J2EE Design and Development 一书中所阐述的设计思想的具体实现。在One-on-One 一书中，[Rod Johnson](https://baike.baidu.com/item/Rod%20Johnson" \t "_blank)倡导[J2EE](https://baike.baidu.com/item/J2EE" \t "_blank)实用主义的[设计思想](https://baike.baidu.com/item/%E8%AE%BE%E8%AE%A1%E6%80%9D%E6%83%B3" \t "_blank)，并随书提供了一个初步的开发框架实现（interface21 开发包）。而Spring 正是这一思想的更全面和具体的体现。Rod Johnson 在interface21 开发包的基础之上，进行了进一步的改造和扩充，使其发展为一个更加开放、清晰、全面、高效的开发框架。

Spring是一个开源框架，由Rod Johnson创建并且在他的著作《J2EE设计开发编程指南》里进行了描述。它是为了解决企业应用开发的复杂性而创建的。Spring使使用基本的JavaBeans来完成以前只可能由EJB完成的事情变得可能了。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。

简单来说，Spring是一个轻量的控制反转和面向切面的容

框架。当然，这个描述有点过于简单。但它的确概括出了Spring是做什么的。

## 6.3 ZF

[Zend Framework](https://baike.baidu.com/item/Zend%20Framework)(简写ZF)是由 Zend 公司支持开发的完全基于 PHP5 的开源PHP开发框架，可用于开发 Web 程序和服务，ZF采用 MVC(Model–View-Controller) 架构模式来分离应用程序中不同的部分方便程序的开发和维护。

## 6.4 .NET

.NET MVC是微软官方提供的以MVC模式为基础的.NET Web应用程序(Web Application)框架,它由Castle的[MonoRail](https://baike.baidu.com/item/MonoRail" \t "_blank)而来(Castle的MonoRail是由java而来),目前最新版本是.N 4.5。

# 特点

## 7.1优点

[**耦合性**](https://baike.baidu.com/item/%E8%80%A6%E5%90%88%E6%80%A7)**低**

视图层和业务层分离，这样就允许更改视图层代码而不用重新编译模型和控制器代码，同样，一个应用的业务流程或者业务规则的改变只需要改动MVC的模型层即可。因为模型与控制器和视图相分离，所以很容易改变应用程序的数据层和业务规则。

模型是自包含的，并且与控制器和视图相分离，所以很容易改变应用程序的数据层和业务规则。如果把数据库从MySQL移植到[Oracle](https://baike.baidu.com/item/Oracle" \t "_blank)，或者改变基于RDBMS数据源到[LDAP](https://baike.baidu.com/item/LDAP" \t "_blank)，只需改变模型即可。一旦正确的实现了模型，不管数据来自数据库或是LDAP服务器，视图将会正确的显示它们。由于运用MVC的应用程序的三个部件是相互独立，改变其中一个不会影响其它两个，所以依据这种设计思想能构造良好的[松耦合](https://baike.baidu.com/item/%E6%9D%BE%E8%80%A6%E5%90%88)的构件。[11]

**重用性高**

随着技术的不断进步，需要用越来越多的方式来访问应用程序。[MVC模式](https://baike.baidu.com/item/MVC%E6%A8%A1%E5%BC%8F" \t "_blank)允许使用各种不同样式的视图来访问同一个服务器端的代码，因为多个视图能共享一个模型，它包括任何WEB（HTTP）浏览器或者无线浏览器（wap），比如，用户可以通过电脑也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。例如，很多数据可能用HTML来表示，但是也有可能用WAP来表示，而这些表示所需要的命令是改变视图层的实现方式，而控制层和模型层无需做任何改变。由于已经将数据和业务规则从表示层分开，所以可以最大化的重用代码了。模型也有状态管理和数据持久性处理的功能，例如，基于会话的购物车和电子商务过程也能被Flash网站或者无线联网的应用程序所重用。[11]

[**生命周期**](https://baike.baidu.com/item/%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F)**成本低**

MVC使开发和维护用户[接口](https://baike.baidu.com/item/%E6%8E%A5%E5%8F%A3" \t "_blank)的技术含量降低。

**部署快**

使用MVC模式使开发时间得到相当大的缩减，它使程序员（Java开发人员）集中精力于业务逻辑，界面程序员（HTML和JSP开发人员）集中精力于表现形式上。

**可维护性高**

分离视图层和业务逻辑层也使得WEB应用更易于维护和修改。

**有利软件工程化管理**

由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化管理程序代码。控制器也提供了一个好处，就是可以使用控制器来联接不同的模型和视图去完成用户的需求，这样控制器可以为构造应用程序提供强有力的手段。给定一些可重用的模型和视图，控制器可以根据用户的需求选择模型进行处理，然后选择视图将处理结果显示给用户。

## 7.2缺点

**没有明确的定义**

完全理解MVC并不是很容易。使用MVC需要精心的计划，由于它的内部原理比较复杂，所以需要花费一些时间去思考。同时由于模型和视图要严格的分离，这样也给调试应用程序带来了一定的困难。每个构件在使用之前都需要经过彻底的测试。

**不适合小型，中等规模的应用程序**

花费大量时间将MVC应用到规模并不是很大的应用程序通常会得不偿失。

**增加系统结构和实现的复杂性**

对于简单的界面，严格遵循MVC，使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。

**视图与控制器间的过于紧密的连接**

视图与控制器是相互分离，但却是联系紧密的部件，视图没有控制器的存在，其应用是很有限的，反之亦然，这样就妨碍了他们的独立重用。

**视图对模型数据的低效率访问**

依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。

**一般高级的界面工具或构造器不支持模式**

改造这些工具以适应MVC需要和建立分离的部件的代价是很高的，会造成MVC使用的困难。

# 外界评价

根据天极网资料显示：基于Web的MVC framework在J2EE的世界内已是空前繁荣，[TTS](https://baike.baidu.com/item/TTS" \t "_blank)网站上几乎每隔一两个星期就会有新的MVC框架发布，比较好的MVC,老牌的有Struts、Webwork。新兴的MVC 框架有Spring MVC、Tapestry、JSF等。这些大多是著名团队的作品，另外还有一些边缘团队的作品，也相当出色，如Dinamica、[VRaptor](https://baike.baidu.com/item/VRaptor" \t "_blank)等，这些框架都提供了较好的层次分隔能力，在实现良好的MVC 分隔的基础上，通过提供一些现成的辅助类库，同时也促进了生产效率的提高。

如何选择一个好的框架应用在项目中，将会对项目的效率和可重用是至关重要的。

# MVC控件

在[ASP](https://baike.baidu.com/item/ASP" \t "_blank) [.NET](https://baike.baidu.com/item/.NET) MVC框架中没有了自己的控件，页面显示完全就回到了写[html](https://baike.baidu.com/item/html" \t "_blank)代码的年代。还好在 asp .net mvc框架中也有自带的HtmlHelper和UrlHelper两个帮助类。另外在MvcContrib扩展项目中也有扩展一些帮助类，这样我们就不光只能使用完整的html来编写了需要显示的页面了，就可以使用这些帮助类来完成，但最后运行时都还是要生成html代码的。

## 9.1 HtmlHelper类

HtmlHelper类位于System.Web.MVC.Html命名空间下。主要包括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。

## 9.2 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。

## 9.3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。

在调用的时候要注意下一命名空间：如果扩展方法的命名空间和页面的命名空间不同的就必须引用扩展方法的[命名空间](https://baike.baidu.com/item/%E5%91%BD%E5%90%8D%E7%A9%BA%E9%97%B4)，否则在页面是没有办法调用自定义的方法的。

引用完命名空间之后，就可以在相应的页面调用自定义的扩展方法了。

对于某些项目来说，自定义控件过于复杂和浪费时间。这个时候也可以从技术社区或是源代码站下载适合自己需求的Mvc控件。如一些控件套包，表格控件等。

1. MVP与MVC

MVP 是从经典的模式MVC演变而来，它们的基本思想有相通的地方：Controller/Presenter负责逻辑的处理，Model提供数据，View负责显示。作为一种新的模式，MVP与MVC有着一个重大的区别：在MVP中View并不直接使用Model，它们之间的通信是通过Presenter (MVC中的Controller)来进行的，所有的交互都发生在Presenter内部，而在MVC中View会从直接Model中读取数据而不是通过 Controller。Alex在他的blog中对于这两者之间的比较很直观也比较清楚，原文可以下面的地址找到：http://ameleta.spaces.live.com/blog/cns!5F6316345A821420!163.entry

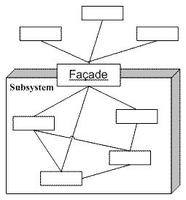
# 简介

在我工作中经常需要处理一些由于开发人员没能很清楚地理解MVC和MVP模式的区别的情况下使用它们而产生的问题。在这篇文章中我将会阐述一下我对两者之间区别的一些理解。

在N层体系结构中MVC/P模式仅仅只是用于表示层(presentation layer)，理解这一点很重要。这两个模式并不是关于怎么构建数据层(data layer)和服务层(service layer)的，而是关于怎么将数据(data)从用户接口(view)中分离出来，以及用户接口如何与数据进行交互的。这些模式的使用让解除你的程序中表示层对对数据和控制逻辑的依赖，从而可以自由的变更表示层。

# 这两种模式的一般性概念

1、模型(Model)表示数据模型和业务逻辑(business logic)。模型并不总是DataSet，DataTable之类的东西，它代表着一类组件(components)或类(class)，这些组件或类可以向外部提供数据，同时也能从外部获取数据并将这些数据存储在某个地方。简单的理解，可以把模型想象成“外观类(facade class)”。译注：这里的外观是指“外观模式”中所说的外观。外观的一般作用是为一个复杂的子系统提供高层次的简单易用的访问接口，可以参看下面的图来理解它的原理：



2、视图(View)将数据层现给用户。一般的视图都只是包含用户界面(UI)，而不包含界面逻辑。比如，Asp.net中包含控件的页面(page)就是一个视图。视图可以从模型中读取数据，但是不能修改或更新模型。

3、层现器(Presenter)/控制器(Controller)包含了根据用户在视图中的行为去更新模型的逻辑。视图仅仅只是将用户的行为告知控制器，而控制器负责从视图中取得数据然后发送给模型。

MVC/P模式的核心是为了将模型从视图/控制器中分离出来，从而使得模型独立于它们，因此模型不包含对视图和控制的引用。

# 什么是MVC(Model View Presenter)模式?

1、为了使得视图接口可以与模型和控制器进行交互，控制器执行一些初始化事件

2、用户通过视图（用户接口）执行一些操作

3、控制器处理用户行为(可以用观察着模式实现)并通知模型进行更新

4、模型引发一些事件，以便将改变发告知视图

5、视图处理模型变更的事件，然后显示新的模型数据

6、用户接口等待用户的进一步操作

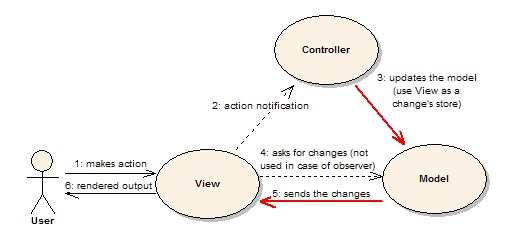
这一模式的有一下几个要点：

1、视图并不使用控制器去更新模型。控制器负责处理从视图发送过来的用户操作并通过与模型的交互进行数据的更新

2、控制器可以和视图融合在一块。Visual Studion中对Windows Forms的默认处理方式就是这样的。【译注：比如我们双击一个Button，然后在它的事件里写处理逻辑，然后将处理的数据写回模型中。这里处理逻辑时间应该是控制器的功能，但是我们并没有专门写一个控制器来做这件事情而是接受了VS的默认处理方式，将它写在Form的代码中，而这里的Form在MVC中它就是一个View。所以这说vs默认的处理方式是将把控制器和视图融合在一起的。】

3、控制器不包含对视图的渲染逻辑(rendering logic)

“主动—MVC”模式，也是通常意义下的MVC模式



【译注：为什么说是主动的？View不是等Controller通知它Model更新了然后才从Model取数据并更新显示，而是自己监视Model的更新(如果用观察者模式)或主动询问Model是否更新。前面那种等待Controller通知的方式是下面所介绍的“被动—MVC”的实现方式。】

**“被动—MVC”模式**

与主动MVC的区别在于：

1、模型对视图和控制器一无所知，它仅仅是被它们使用

2、控制器使用视图，并通知它更新数据显示

3、视图仅仅是在控制器通知它去模型取数据的时候它才这么做(视图并不会订阅或监视模型的更新)

4、控制器负责处理模型数据的变化

5、控制器可以包含对视图的渲染逻辑

# MVP模式

与“被动—MVC模式”很接近，区别在于“视图并不使用模型”。在MVP模式中视图和模型是完全分离的，他们通过Presenter进行交互。

Presenter与控制器非常相似，但是它们也有一些的区别：

1、Presenter处理视图发送过来的用户操作（在MVC中视图自己处理了这些操作）

2、它用更新过的数据去更新模型（在被动MVC中控制器只是通知视图去更新过的模型中去取新的数据，而主动MVC中模型通知视图去更新显示，控制器不需要做工作）

3、检查模型的更新（与被动MVC一样）

4、（与MVC的主要区别）从模型中取数据然后将它们发送到视图中

5、（与MVC的主要区别）将所做的更新告知视图

6、（与MVC的区别）用Presenter渲染视图

# MVP的优势

1、模型与视图完全分离，我们可以修改视图而不影响模型

2、可以更高效地使用模型，因为所以的交互都发生在一个地方——Presenter内部

3、我们可以将一个Presener用于多个视图，而不需要改变Presenter的逻辑。这个特性非常的有用，因为视图的变化总是比模型的变化频繁。

4、如果我们把逻辑放在Presenter中，那么我们就可以脱离用户接口来测试这些逻辑（单元测试）。

# MVP的问题

由于对视图的渲染放在了Presenter中，所以视图和Persenter的交互会过于频繁。

还有一点你需要明白，如果Presenter过多地渲染了视图，往往会使得它与特定的视图的联系过于紧密。一旦视图需要变更，那么Presenter也需要变更了。

如说，原本用来呈现Html的Presenter现在也需要用于呈现Pdf了，那么视图很有可能也需要变更。