MVC设计模式以及其他

流星
你有我菜吗?

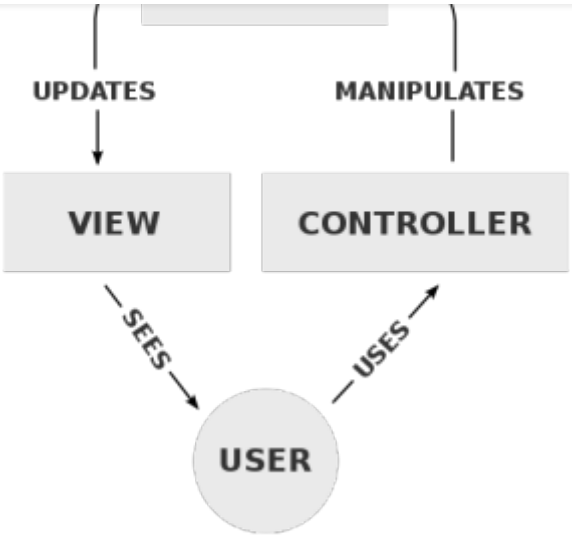
【备注】本文来源转载，文章来源

一、Classic MVC

MVC模式(Model-View-Controller)是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型(Model)、视图(View)和控制器(Controller)。最早由Trygve Reenskaug在1978年提出，并应用在Smalltalk系统中。

Classic Mvc模式：

- Model：封装领域数据及逻辑。用于管理应用程序域的行为和数据，并响应为获取其状态信息(通常来自视图)而发出的请求，还会响应更改状态的指令(通常来自控制器)。
- View：查询领域数据并展现给用户。用于管理信息的显示。
- Conctroller：截获用户请求并改变领域数据。用于解释用户的鼠标和键盘输入，以通知模型和/或视图进行相应的更改。



从依赖关系看，Model不依赖View和Controller，而View和Controller依赖Model。

Classic MVC关注两个分离：

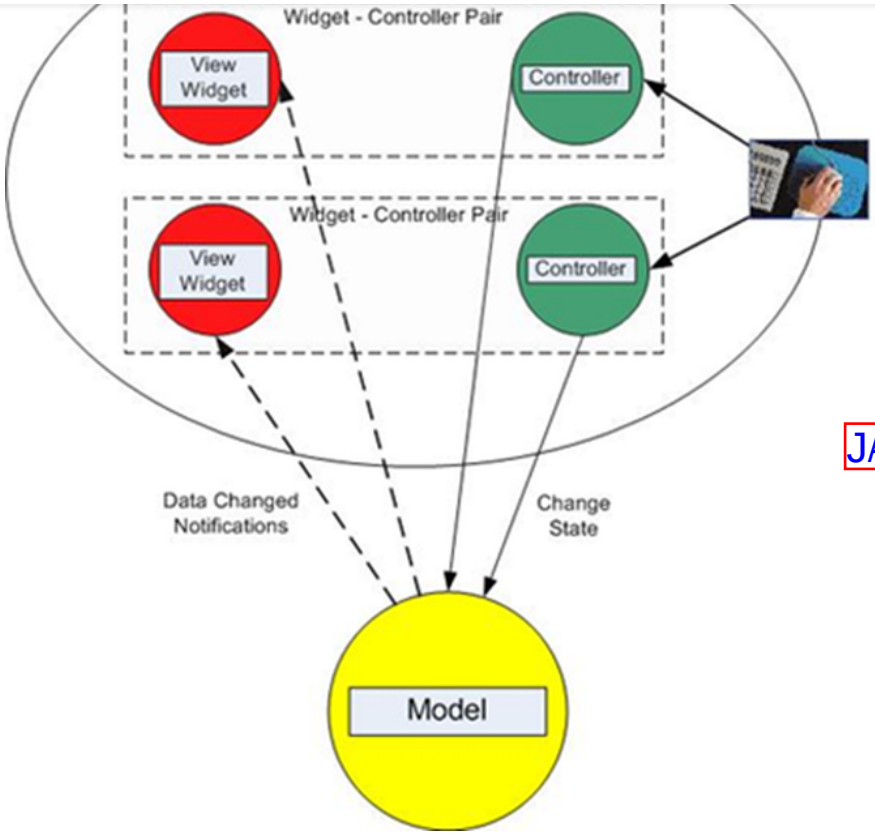
- 从Model中分离View
- 从View中分离Controller

从Model中分离View，主要基于以下几点考虑：

- 不同的关注点：Model关注内在的不可视的逻辑，而View关注外在的可视的逻辑。
- 多种表现形式：同一个Model往往需要多种View表现形式。
- 提高可测试性：相对Model而言，View是不容易测试的。

Desktop软件的时代，View和Controller往往是一一对应的关系，所以常常把他们合并成为UI，事实上，当时多数UI框架都没有实现从View中分离Controller。后来随着Web的兴起，这种分离(模板技术)才开始流行起来。

本质上Classic MVC的结构如下图所示，之所以说本质上，是因为View和Controller其实是彼此关联的，但这种关联和稍后提到的MVP完全不同，更像是一种框架的副产品，为了避免引起混淆，这里省略了它们，具体参阅：How to use Model-View-Controller (MVC)



JAVAFX？

Controller截获用户通过鼠标或键盘发出的请求，然后改变Model的状态，Model通过Observer Synchronization(观察者模式)通知View自己的状态发生了变化。View查询Model展现数据。

Classic MVC并不完美，不适用于复杂的逻辑。举个例子：用户通过鼠标拖动滚动条来调整音量大小，如果音量大于某个数值，背景色变红以示提醒。当使用Classic MVC的时候，如何处理背景色变红的逻辑呢?

有两个选择：

- Model触发一个特殊事件，View收到后完成相关逻辑的处理。但我们前面说过，从依赖关系上看，Model应该完全无视View的存在。
- 在View中判断音量临界值，达到后完成相关逻辑的处理。但我们前面说过，View是不容易测试的，应该尽可能减少逻辑处理。

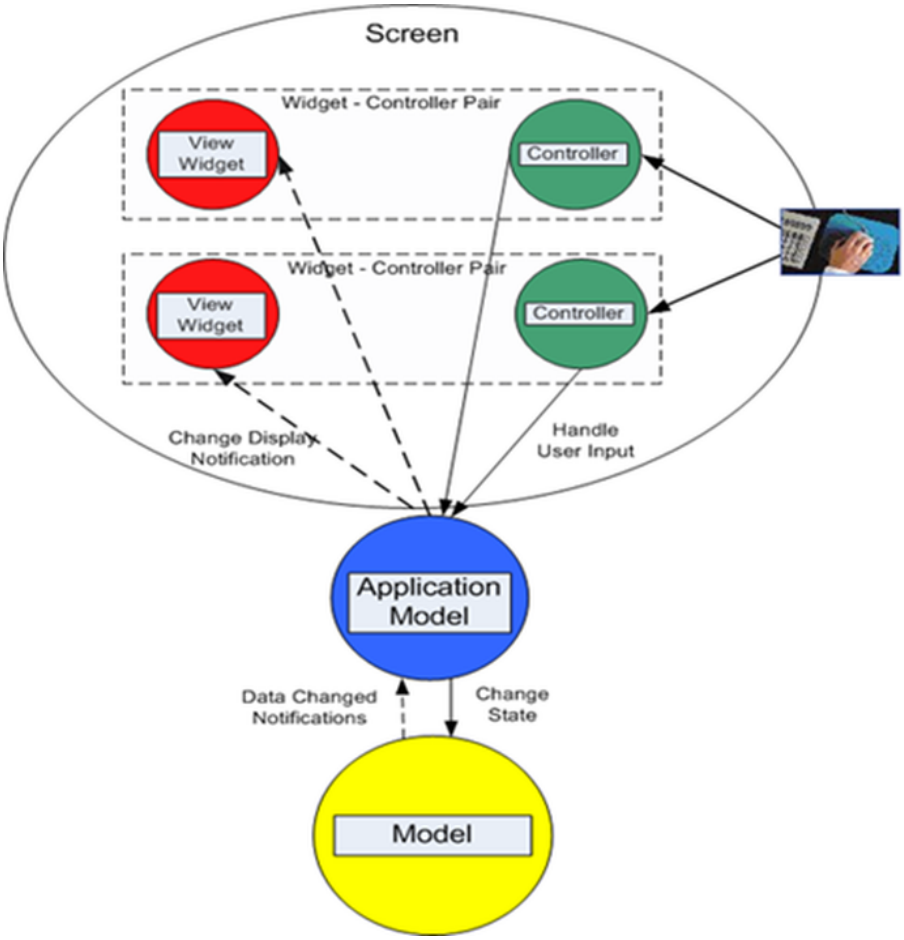
Classic MVC并不完美，不适用于复杂的逻辑。举个例子：用户通过鼠标拖动滚动条来调整音量大小，如果音量大于某个数值，背景色变红以示提醒。当使用Classic MVC的时候，如何处理背景色变红的逻辑呢？有两个选择：

- Model触发一个特殊事件，View收到后完成相关逻辑的处理。但我们前面说过，从依赖关系上看，Model应该完全无视View的存在。
- 在View中判断音量临界值，达到后完成相关逻辑的处理。但我们前面说过，View是不容易

上述只是提出了Classic MVC的一些缺陷，那到底如何解决呢？请继续关注下文~

二、Application Model MVC

大概上世纪八十年代，ParcPlace从Xerox Parc划分出来，负责Smalltalk的研发工作，为了适应更复杂的逻辑，开发了Classic MVC的改进版，也就是Application Model MVC，在原有架构基础上引入了Application Model，如下图所示：



Application Model在Model和View、Controller之间扮演着一个中继者的角色。接着看前面的例子，既然Model和View都不适合放背景色变红的逻辑，那么我们可以尝试把相关逻辑放在Application Model中实现，当用户通过鼠标调整音量大小时，Model触发一个普通事件，Application Model拦截到这个事件，判断音量是否大于临界值，如果是就触发一个特殊事件，View收到后完成相关逻辑的处理。

Application Model MVC虽然看似解决了复杂逻辑的问题，但它仍然存在硬伤：

- 随着以微软视窗为主的图形化操作系统的兴起，操作系统本身提供了一套原生的View接口，用来截获用户通过鼠标或键盘发出的请求，结果让Controller显得多余了。
- 由于在Application Model MVC中，View的渲染只能通过事件的方式实现，Application

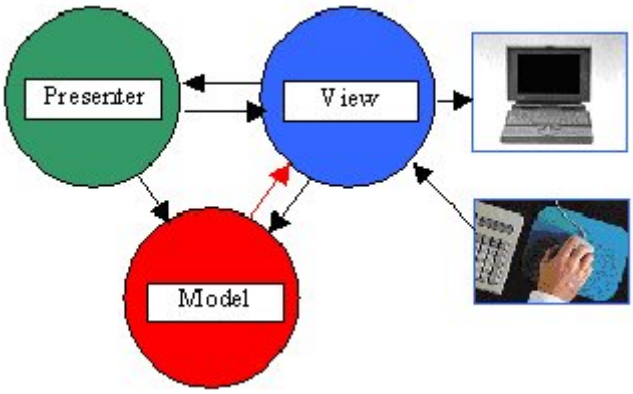
接着前面说的调节音量的例子，这次我们加个新功能，不再通过鼠标拖动滚动条来调整音量大小，而是给出一个文本框，让用户直接通过键盘输入阿拉伯数字表示音量大小，一旦用户输入非法内容(比如说英文字符)，背景色变黄以示警告。问题是如果用户输入非法内容，就不应该改变Model的状态，但不改变Model的状态，View就没有机会收到渲染的事件。

三、MVP

大概上世纪九十年代，IBM的Mike Potel提出了MVP(Model-View-Presenter)的概念。

- **Model** 定义使用者接口所需要被显示的资料模型，一个模型包含着相关的商业逻辑。
- **View** 视图为呈现使用者接口的终端，用以表现来自 Model 的资料，和使用者命令路由再经过 Presenter 对事件处理后的资料。
- **Presenter** 包含着元件的事件处理，负责检索 Model 取得资料，和将取得的资料经过格式转换与 View 进行沟通。

与此同时，Smalltalk团队正在开发新一代框架，当他们看到MVP时，发现它不仅和MVC非常相似，并且很好的解决了复杂逻辑的问题，所以决定使用它，出于复杂度的关系，他们简化了MVP，最终看上去更像是把原本的MVC扭转了60°，把其中的VC颠倒了一下顺序：



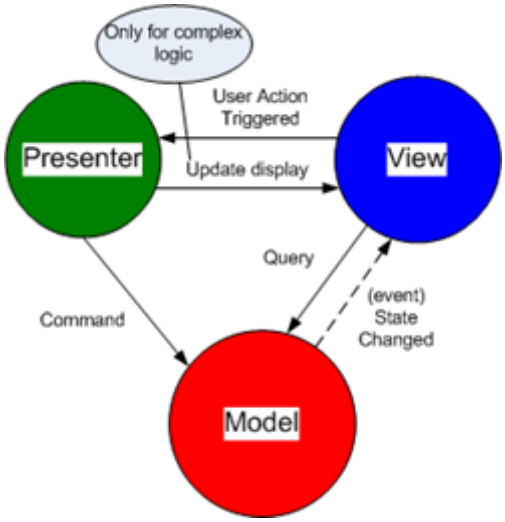
View截获用户请求，然后委派给Presenter，Presenter改变Model的状态，Model通过Observer Synchronization通知View自己的状态发生了变化，View查询Model展现数据。

最重要的是一点是Presenter和View彼此持有对方的引用。虽然View截获用户请求，但它并不处理，而是委派给Presenter处理，保证了可测试性，同时，因为Presenter可以直接操作View，不必受限于观察者模式。

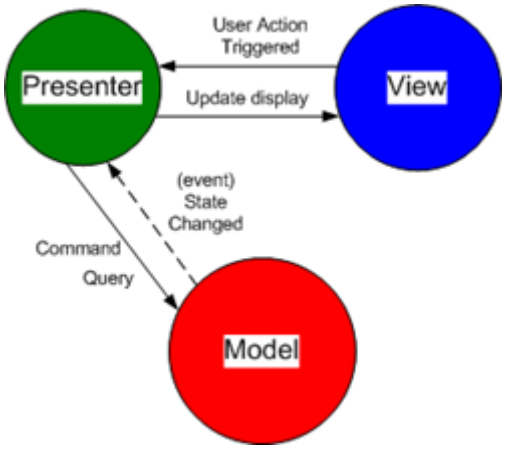
接着前面说的调节音量的例子，当用户通过鼠标拖动滚动条来调整音量大小时，View截获请求，并把请求委派给Presenter，如果Presenter发现音量大于临界值，直接操作View实现逻辑；当用

Martin Fowler分析了MVP的实现方式，分类为

1、Supervising Controller



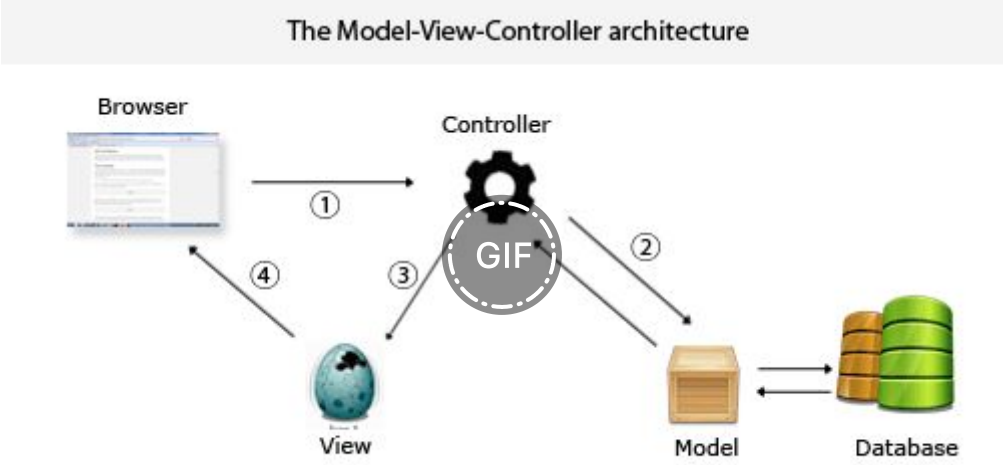
2、Passive View。



二者的区别在于Model和View是否有联系，在Supervising Controller的实现中，View可以查询Model，Model状态发生变化的话会通知View，而在Passive View的实现中，View不可以查询Model，Model状态发生变化的话会通知Presenter，由Presenter完成View的渲染。比较而言，Passive View的可测试性更好一些，但Presenter的代码量相应大些。

前面我们讨论了MVC到MVP的演化史，随着Web的兴起，人们开始把MVC，MVP等知识应用到Web环境下，但Web环境有其特殊性，最重要的一点就是HTTP是无状态的，每次请求都是独立的，所以不可能实现观察者模式。

Java是Web MVC最早的实践者，开发出Model 2，使用JavaBean，JSP，Servlet分别对应MVC中的三个组成部分，紧接着Structs的出现开始让大众注意到Web MVC，不过真正让Web MVC流行起来的却是Ruby on Rails，其大致流程如下图所示：



一个典型的Web MVC流程：

- Controller截获用户发出的请求
- Controller调用Model完成状态的读写操作
- Controller把数据传递给View
- View渲染最终结果并呈献给用户

在Classic MVC中，Controler可以改变Model的状态，View可以查询Model的状态，所以说对Model而言，Controller和View的地位是平等的，不过在Web MVC中，Controller变成了中继者，主要工作是协调Model和View，如此看来，Web MVC中的Controller等同于MVP中的Presenter。那为什么不叫Web MVP，而称之为Web MVC？这是因为截获请求的是Controller而不是View。

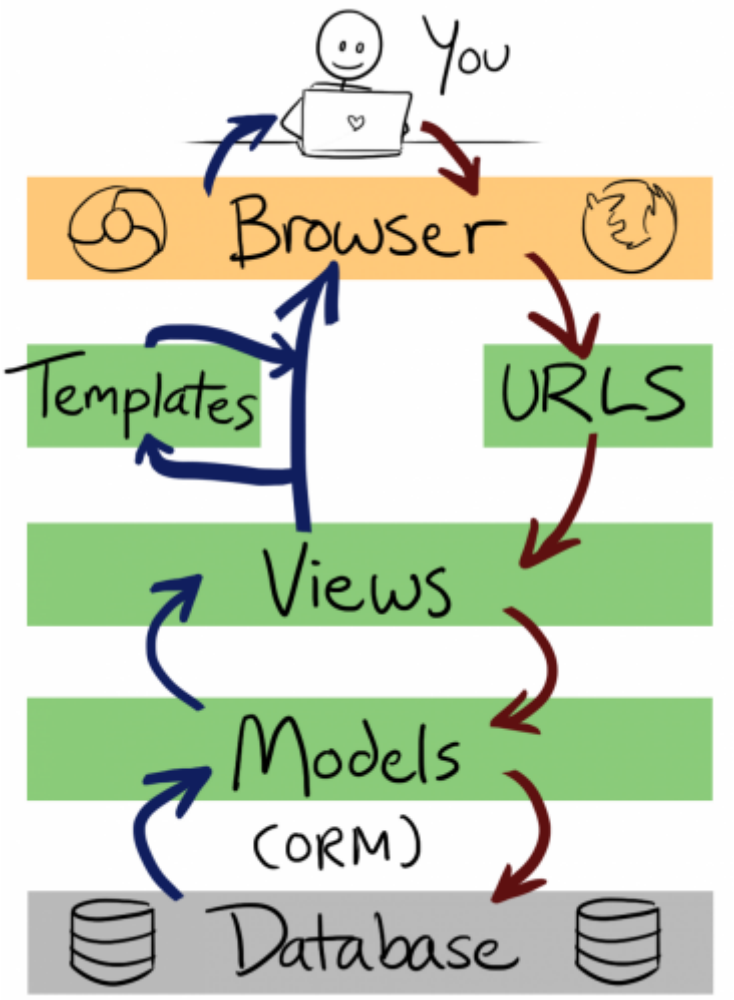
五、MTV

Python的Django框架宣称自己使用的是MTV，其实质仍然是Web MVC。Django将MVC中的视图进一步分解为 Django视图 和 Django模板两个部分，分别决定 “展现哪些数据” 和 “如何展现”，使得Django的模板可以根据需要随时替换，而不仅仅限制于内置的模板。至于MVC控制器部分，由Django框架的URLconf来实现。

Django 里关注的是模型(Model)、模板(Template)和视图(Views)，分别为：

- Model，即数据存取层。 该层处理与数据相关的所有事务： 如何存取、如何验证有效性、

- View，即业务逻辑层。 该层包含存取模型及调取恰当模板的相关逻辑。 你可以把它看作模型与模板之间的桥梁。

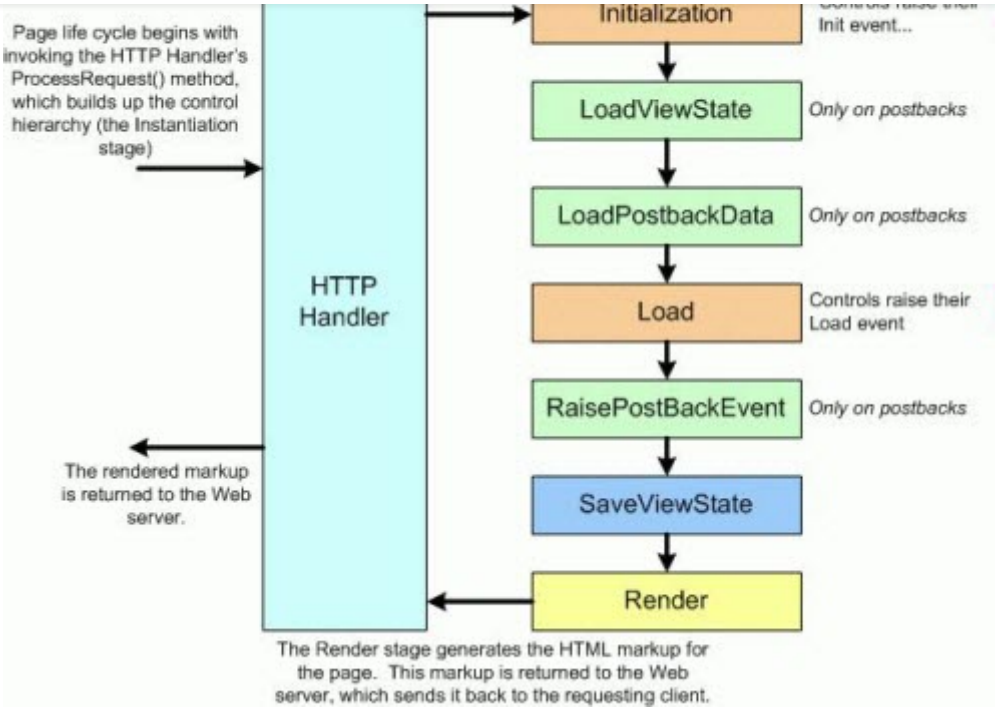


需要注意的是，不能简单的把 Django视图认为是MVC控制器，把 Django 模板认为MVC视图。区别在于：

- Django 视图 不处理用户输入，而仅仅决定要展现哪些数据给用户；
- Django 模板 仅仅决定如何展现Django视图指定的数据。

六、Web MVP

在Desktop的时代，微软通过WinForms实现MVP，把组件化编程发挥到了极致，大大提升了开发效率，随着Web的兴起，微软希望延续这样的编程模式，所以使用WebForms实现了Web MVP，引入了CodeBehind，ViewState等设计概念。

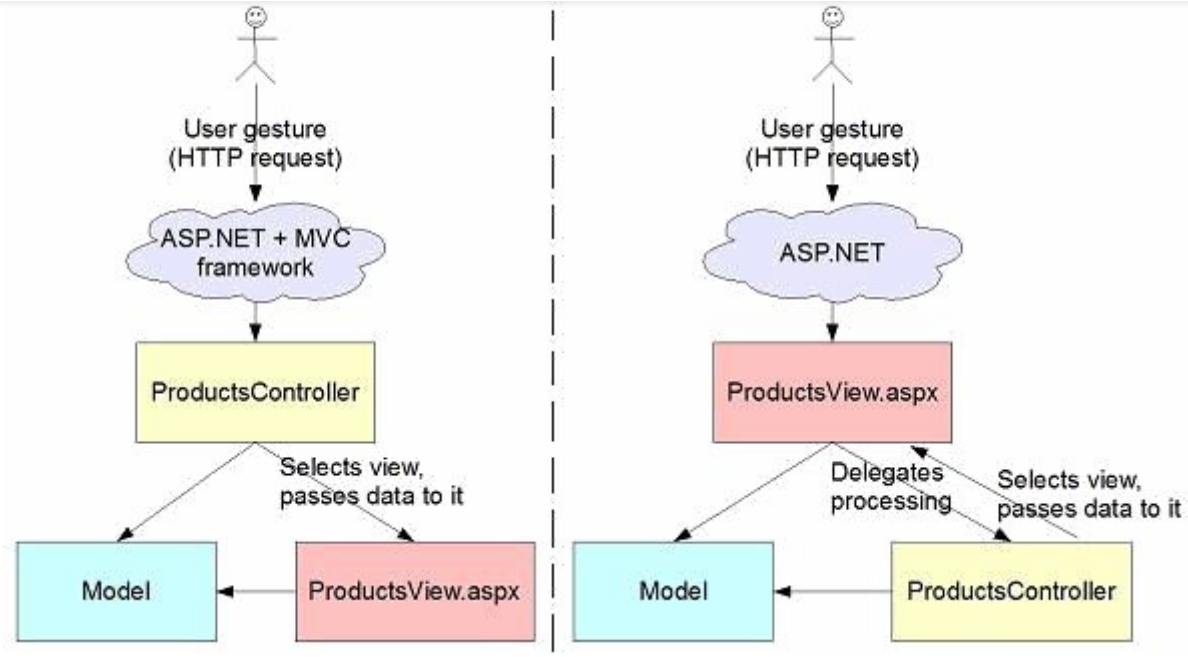


当一个浏览器向服务器请求一个aspx页面时的简体步骤如下：

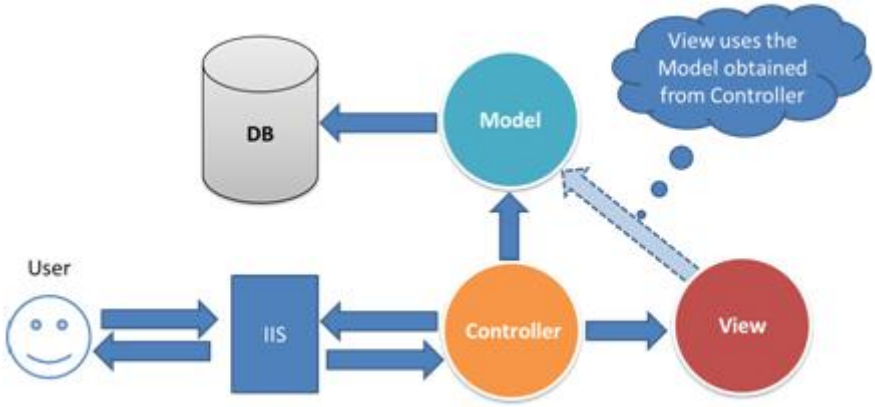
- 服务器会首先创建前台页面aspx类的对象，当子类(aspx类)被创建时，父类(后台页面CS类)也会顺便被创建。
- 接着就会在前台页面类中调用ProcessRequest方法(PR方法不是在前台页面类中定义的，而是在Page类中定义的，
- 因为CS类继承与page类，而aspx类又继承与page类，所以PR方法相当于aspx类的爷爷类中定义的)。
- 在PR方法中调用BuildControlTree方法，把前台页面所有的html控件和runat=server的控件转成对应的控件对象并添加在前台页面
- 类得Controls集合中(这里当前页面即aspx页面类是根节点)，而且runat=server的控件对象会保存在后台CS类中的一个对应类型的变量中。
- 在PR方法中调用后台页面CS类的Page_Load方法，这个方法中的代码是程序员自己写的。
- 最后再PR方法中调用Render_Controls方法，来遍历控件树中每一个节点的Render_Controls方法，生成完整的html代码
- 把完整的html代码返回给浏览器。

七、ASP.NET MVC

微软同时又推出了类似Web MVC的ASP.NET MVC，但是在截获请求部分还是存在着一些差别，具体请看下图：



ASP.NET MVC的具体工作流程为：



当用户从浏览器输入地址，发出页面请求，到返回结果，一般经过以下步骤：

- 当用户输入地址，发出请求时，实际上就是向控制器发出相关命令
- 控制器接收用户指令后，向模型请求获得相关数据
- 模型将对应的数据返回给控制器
- 控制器将有关数据发送到指定视图
- 指定的视图呈现指定的数据

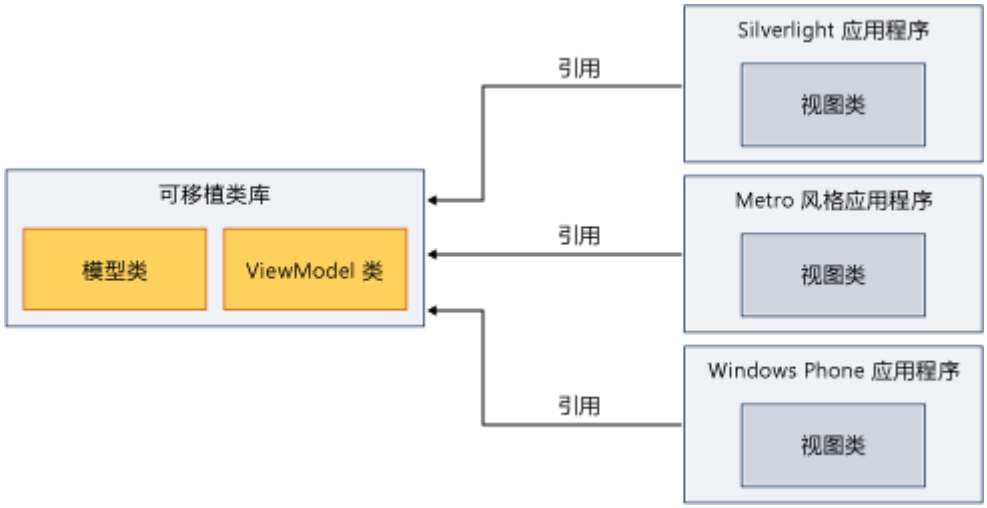
Web Forms构建web相对容易，开发人员只需在一个可视化设计器中拖放控件，设置相关属性即可，通过编写代码来响应事件，使得对于程序的逻辑操作非常直观。但是，开发人员很难了解背后HTML是如何运行的，同时，如果没有合理控制ViewState的话，页面的尺寸将大大超过预期，使得页面打开相当缓慢，随着web应用的复杂化，不容易测试也是开发中面对的一个问题.

计的分离，也提高了程序的可维护性和扩展性，特别是利于应用程序的测试，可以比较容易的实施测试驱动开发。

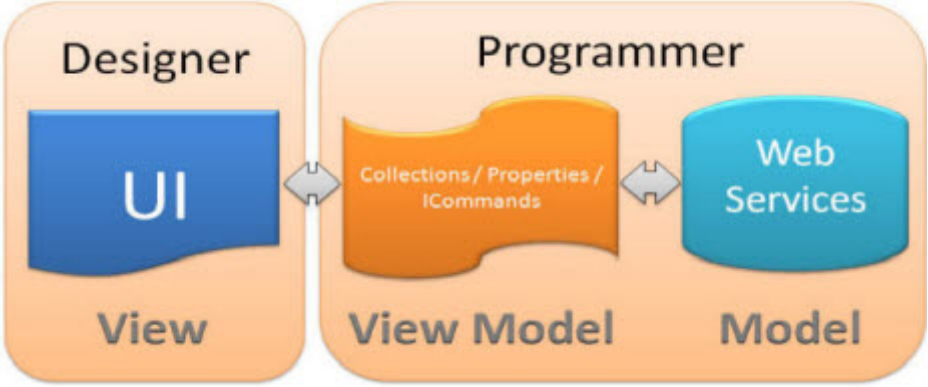
两种开发技术并存。 MVC只是给开发者提供了开发web应用程序的一种选择，而不是替代传统的Web Forms，这两种技术应用于不同的场合具有不同的优缺点。具体两者之间的比较分析，可以查看WebForms vs. MVC。

八、MVVM

Model-View-ViewModel是一种架构模式，主要在WPF、Silverlight和WP7开发里使用，它的目标是从视图层移除几乎所有代码隐藏(code-behind)。MVVM是更加通用的Presentation模式的一个具体实现。MVVM视图模型包含概念模型而不是数据模型，所有业务逻辑和其它操作都是在模型和视图模型里完成的。



在WPF/Silverlight中应用MVVM模式，View主要用于界面呈现，ViewModel用于逻辑实现，Model用于数据的构造，而这三者能够进行通信，最重要的是通过WPF/Silverlight中强大的数据绑定机制，将View和ViewModel有效的联系起来。



使用MVVM架构最大的好处是开发人员在写程序的时候不需要做UI，而设计人员可以使用Microsoft Expression Blend 4+设计全部的UI并且不需要写任何代码。主要的好处如下:

- 设计人员可以用设计工具很容易的设计UI，而且不需要写任何代码
- 你可以更好的设计UI，而且可以让即使不是开发人员使用。
- 可以先设计UI或者与开发同时设计。
- 当UI全部改变时，代码可以不改变。

为了达到以上要求。当你设计UI时，后台不能有任何代码。并且UI与应用程序通过Bindings和Commands相互交互，其中Bindings和Commands在ViewModel中设计。

Model层主要为应用程序提供数据。其主要包含

- Web Services：SilverLight应用程序的特点就是必须通过Web service取得数据，你可以调用Web Service中的方法。
- Rest Services：和Web Services一样
- Generic Collections：任何类型的数据集合

View Model一般有以下三个部分组成

- 属性：一个事物，它的类型可以是一个字符型，也可以是一个对象。实现接口INotifyPropertyChanged,那么任何UI元素绑定到这个属性，不管这个属性什么时候改变都能自动和UI层交互。
- 集合：事物的集合，它的类型一般是ObservableCollection，因此，任何UI元素绑定到它，不管这个集合什么时候改变，都可以自动的与UI交互。
- Commands:一个可以被触发的事件，并且可以传递一个类型为Object的参数。但是前提是要实现接口ICommand。

这一层可以用Expression Blend设计，不用写任何代码。主要有以下三个部分组成

- 把View Model层的属性绑定到 text box, radio button, toggle button, MediaElement, trigger an animation or ViewState change
- 把View Model层的集合绑定到ListBox,TreeView,DataGrid
- Commands

使用InvokeCommandAction实现以下behavior

- A、绑定View Model层的ICommand
- B、指出你需要实现的ICommand(比如Click事件,Selected事件。。。)
- C、传递参数

编辑于 2017-09-16

文章被以下专栏收录

进击的流星
知人者智，自知者明。胜人者有力，自胜者强。知足者富

推荐阅读

MVC设计模式之Web应用剖析

慕课网 发表于猿论

从零学习Sprin 境搭建和MVC

Java后端

1 条评论

切换为时间排序

写下你的评论...

Hugh 2019-08-20

How to use MVC的链接 需要翻墙?

赞