火丁笔记

多研究些问题，少谈些主义。

MVC演化史

发表于**2011-05-02**

Martin Fowler在他所写的《企业应用架构模式》一书中感慨道：MVC已经成为我们最常误用的模式。人们之所以常常误用MVC，很大程度上是因为混淆了不同的MVC变体。

Classic MVC

大概上世纪七十年代，Xerox PARC的Trygve提出了MVC的概念，并应用在Smalltalk系统中，为了和其它类型的MVC加以区分，历史上习惯的称之为Classic MVC。

- Model：封装领域数据及逻辑
- View：查询领域数据并展现给用户
- Conctroller：截获用户请求并改变领域数据

注意：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：

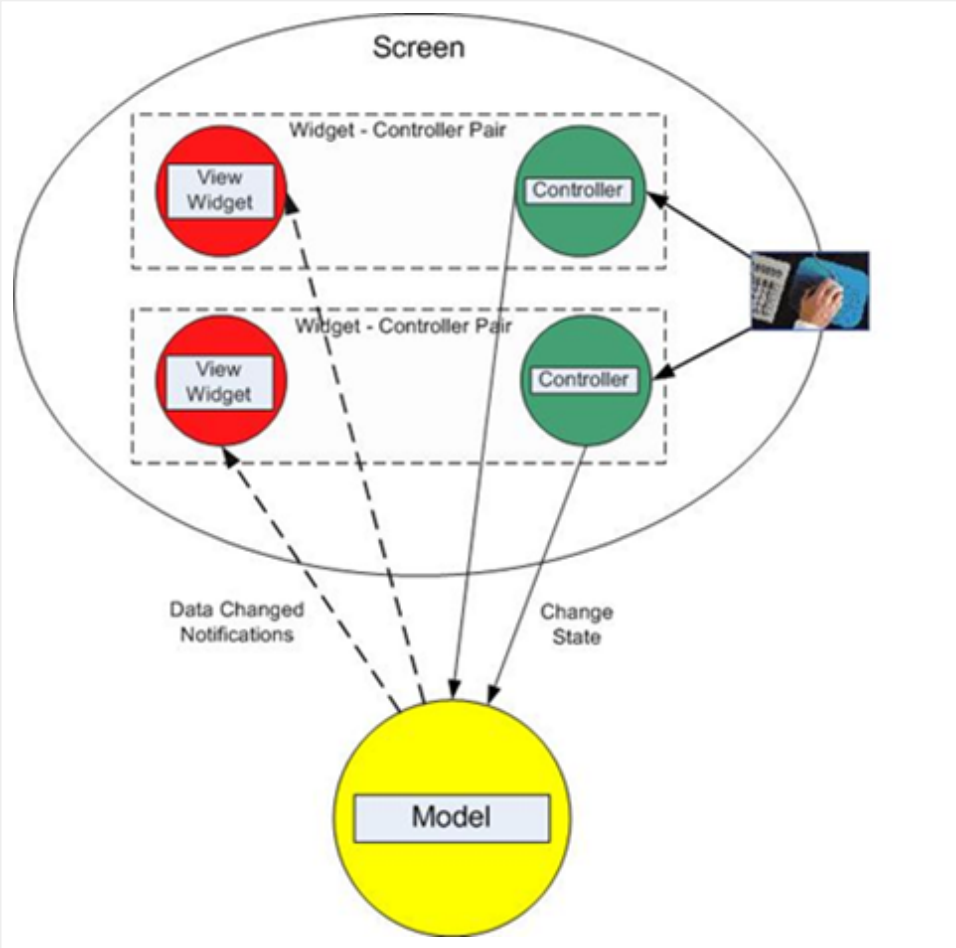
- 从Model中分离View
- 从View中分离Controller

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，如文本、图像。
- 提高可测试性：相对Model而言，View是不容易测试的。

从View中分离Controller就不那么重要了。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)



— Classic MVC

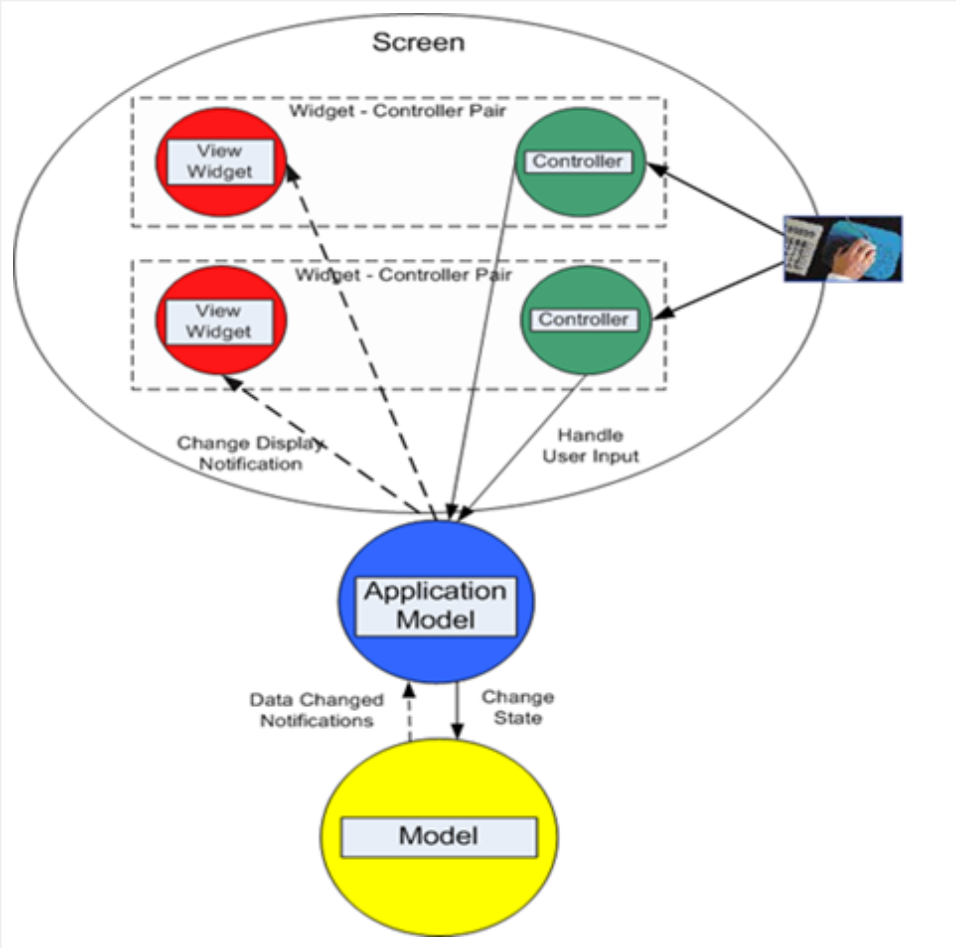
图解：Controller截获用户通过鼠标或键盘发出的请求，然后改变Model的状态，Model通过Observer Synchronization通知View自己的状态发生了变化，View查询Model展现数据。

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，所以这样的味道很坏。
- 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，所以这样的味道同样不好。

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 MVC

图解：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：

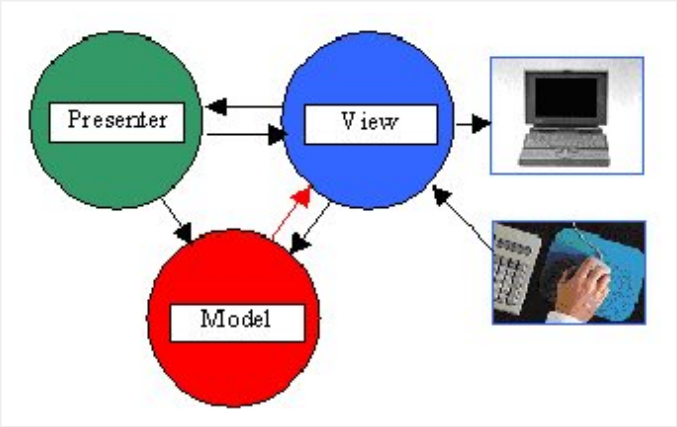
首先随着以微软视窗为主的图形化操作系统的兴起，操作系统本身提供了一套原生的View接口，用来截获用户通过鼠标或键盘发出的请求，结果让Controller显得多余了。

其次由于在Application Model MVC中，View的渲染只能通过事件的方式实现，Application Model不能直接操作View，所以某些情况下不能方便的实现业务逻辑。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。

MVP

大概上世纪九十年代，IBM的Mike Potel提出了MVP的概念。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扭转了一个角度，把其中的VC颠倒了一下顺序：



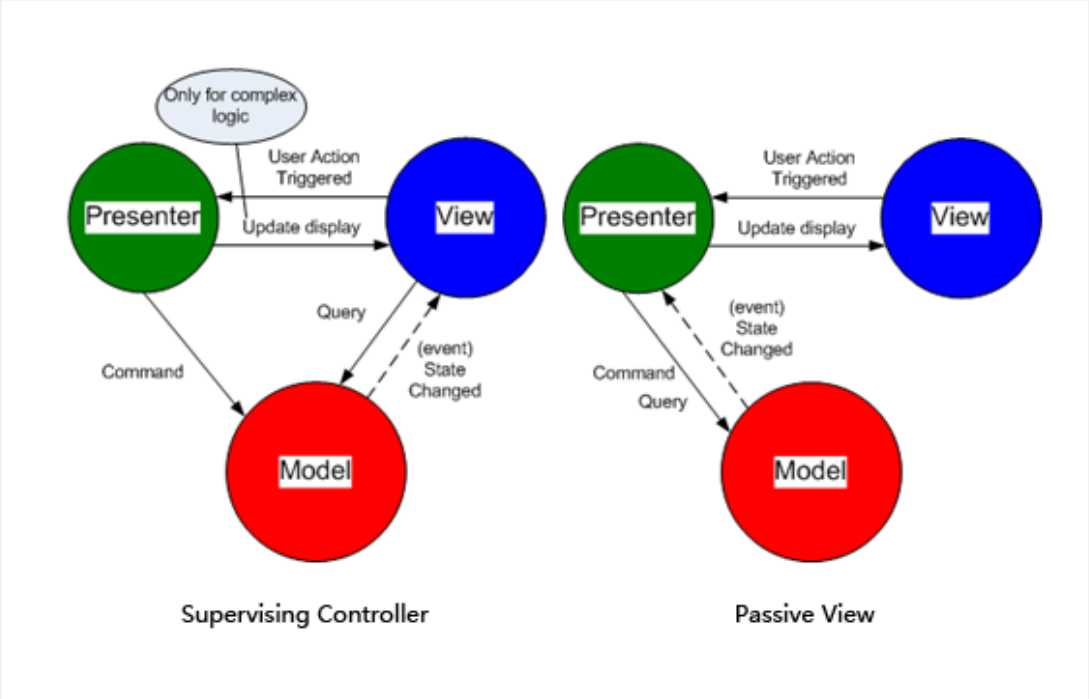
— MVP

图解：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；当用户通过键盘输入音量大小时，View截获请求，并把请求委派给Presenter，如果Presenter发现内容非法，直接操作View实现逻辑。

Martin Fowler分析了MVP的实现方式，分类为Supervising Controller和Passive View。



— Supervising Controller and Passive View

图解：MVP的两种分类：Supervising Controller和Passive View

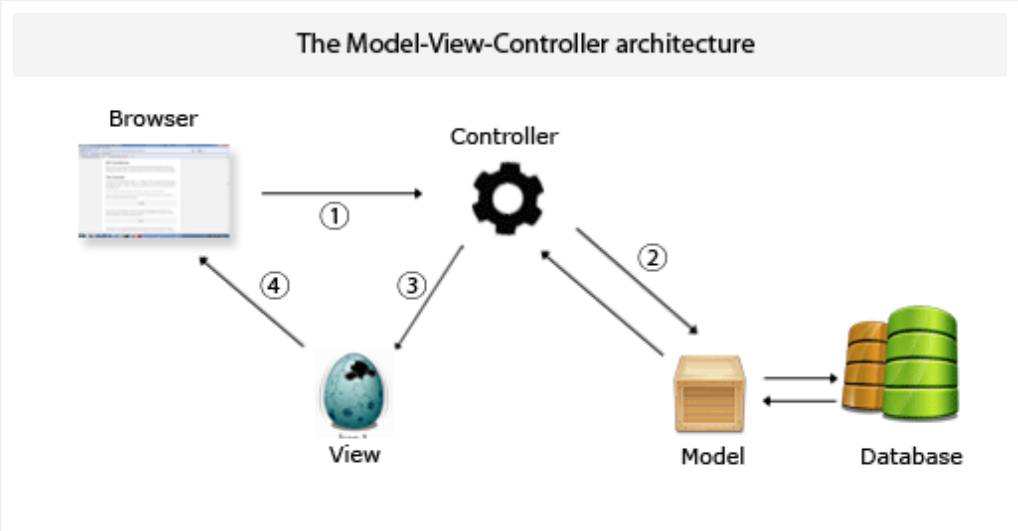
二者的区别在于Model和View是否有联系，在Supervising Controller的实现中，View可以查询Model，Model状态发生变化的话会通知View，而在Passive View的实现中，View不可以查询Model，Model状态发生变化的话会通知Presenter，由Presenter完成View的渲染。比较而言，Passive View的可测试性更好一些，但Presenter的代码量相应大些。

…

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。

Web MVC

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社区的Rails，其大致流程如下图所示：



— Web MVC

图解：一个典型的Web MVC流程

1. Controller截获用户发出的请求
2. Controller调用Model完成状态的读写操作
3. Controller把数据传递给View
4. 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。

花絮：Python社区的Django框架宣称自己使用的是MTV，其实质仍然是Web MVC。

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。WebForms的优点和缺点都很突出，篇幅所限，具体的描述大家可以参考下面链接：

1. 为WebForms说几句话，以及一些ASP.NET开发上的经验（1）
2. 为WebForms说几句话，以及一些ASP.NET开发上的经验（2）
3. 为WebForms说几句话，以及一些ASP.NET开发上的经验（3）

注：微软推出了ASP.NET MVC向Web MVC靠拢，似乎要两手抓两手都要硬。

— ASP.Net MVP vs MVC

图解：微软Web MVP vs Web MVC。注意截获请求的是Controller还是View！

…

以上便是MVC的演化史，我尽量让描述浅显易懂，但文字总是枯燥的，还好有好事者做了一首MVC之歌：Model-View-Controller Song，闲暇无事之时不妨听听。

最后以C2上的一句话结尾：We need SMART Models, THIN Controllers, and DUMB Views

说明：本文使用了下列链接中的内容：

- GUI Architectures
- Interactive Application Architecture Patterns
- Model View Controller: History, theory and usage
- Twisting the MVC Triad – Model View Presenter (MVP) Design Pattern

收工！

此条目由**老王**发表在**Technical**分类目录，并贴了**MVC**、**MVP**标签。将**固定链接[https://blog.huoding.com/2011/05/02/64]** 加入收藏夹。

《MVC演化史》上有26个想法

Liang Shan
在**2011-05-03 10:11:19**说道：

DUMB Views 应该是什么样的呢

> 老王
> 在**2011-05-03 10:35:19**说道：
>
> 因为View是不容易测试的，所以应该尽可能的保持简单。

joshle
在**2011-05-07 10:07:55**说道：

嗯，不错！学习了

Iterse's blog
在**2011-08-23 23:37:27**说道：

学习了！

Pingback引用通告： MVC 演化史& BigPipe « 麵店小弟

Pingback引用通告： MVC演化史 | Chinasb's Blog

frank
在**2011-09-17 15:49:07**说道：

老王真的说得很好，都说到点子上了

blue5tar
在**2011-10-16 21:10:22**说道：

没有地方留言 在这里问您一个关于领域对象的问题

请问 一个优惠券 和一个优惠券使用记录(用两个表存储) 他们应该算是一个领域对象 还是两个领域对象 谢谢

> 老王
> 在**2011-10-17 09:24:21**说道：
>
> 这要看具体情况而定，如果优惠卷使用记录本身不涉及其他业务逻辑，那么就把它和优惠卷放到一个对象里即可，反之就应该独立封装，但不管你设计成一个对象，还是两个对象，都应该确保这两个领域概念是一个整体，领域驱动设计中，有“根”的概念，放到这里，优惠卷应该是根，优惠卷使用记录附属于它，具体点说，它们应该是一个组合关系，外界不能直接访问优惠卷使用记录，必须通过优惠卷来访问，这样的有利于集中控制逻辑。

Pingback引用通告： MVC之患上肥胖症的Controller – 尘埃落定

cainanyang
在**2012-03-06 11:47:22**说道：

很赞，谢了。

redstar
在**2012-09-17 15:10:22**说道：

good