谈谈UI架构设计的演化

2015年2月11日 00:22


寒冬winter
ali-mfe-hire@list.alib…

经典MVC

在1979年，经典MVC模式被提出。

在当时，人们一直试图将纯粹描述思维中的对象与跟计算机环境打交道的代码隔离开来，而Trygve Reenskaug在跟一些人的讨论中，逐渐剥离出一系列的概念，最初是Thing、Model、View、Editor。后来经过讨论定为Model、View和Controller。作者自言“最难搞的就是给这些架构组件起名字”。

因为当时的软件环境跟现在有很大不同，所以经典MVC中的概念很难被现在的工程师理解。比如经典MVC中说：“view永远不应该知道用户输入，比如鼠标操作和按键。”对一个现代的软件工程师来说，这听上去相当不可思议：难道监听事件不需要类似这样的代码吗？

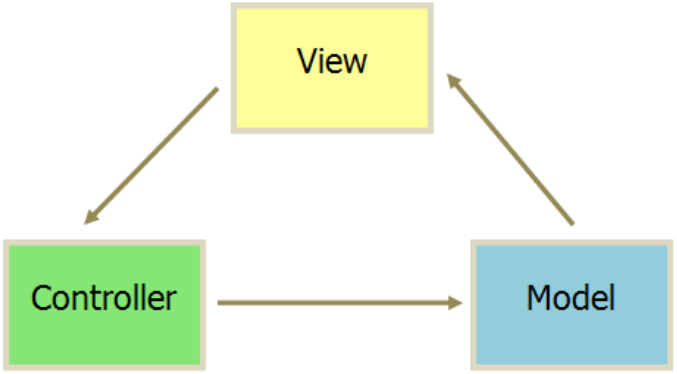
```
view.onclick = ……
```

但是想想在70年代末，80年代初，我们并没有操作系统和消息循环，甚至鼠标的光标都需要我们的UI系统来自行绘制，所以我们面对的应该是类似下面的局面：

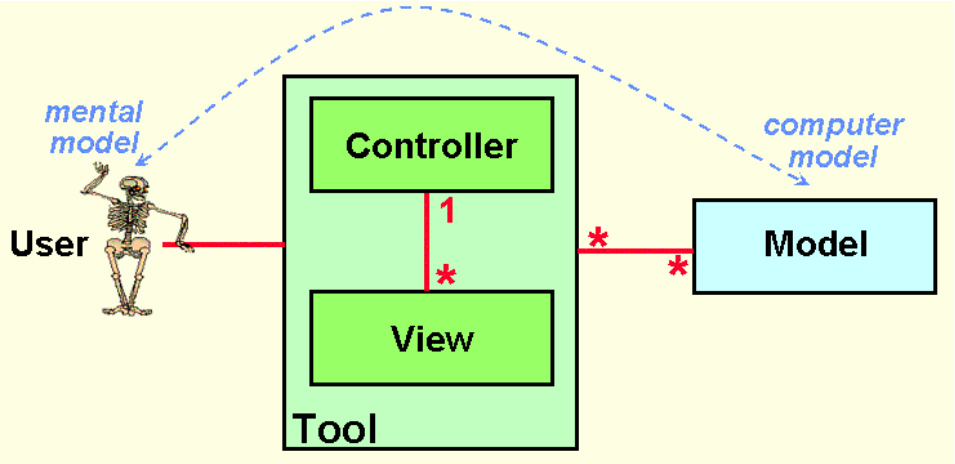
```
mouse.onclick = ……mouse.onmove = ……
```

当鼠标点击事件发生后，我们需要通过view的信息将点击事件派发到正确的view来处理。假如我们面对的是鼠标、键盘驱动这样的底层环境，我们就需要一定的机制和系统来统一处理用户输入并且分配给正确的view或者model来处理。这样也就不难理解为什么经典MVC中称"controller是用户和系统之间的链接"。

因为现在的多数环境和UI系统设计思路已经跟1979年完全不同，所以现代一些喜好生搬硬套的"MVC"实现者常常会认为controller的输入来自view，以至于画出model、view、controller之间很奇葩的依赖关系：



我们来看看Trygve Reenskaug自己画的图（这恶趣味的骷髅啊……）：



值得一提的是，其实MVC的论文中，还提到了"editor"这个概念。因为没有出现在标题中，所以editor声名不著。MVC论文中推荐controller想要根据输入修改view时，从view中获取一个叫做editor的临时对象，它也是一种特殊的controller，它会完成对view和view相关的model的修改操作。

控件系统

MVC是一种非常有价值的架构思路，然而时代在变迁，随着以windows系为代表的WIMP（window、icon、menu、pointer）风格的应用逐渐成为主流，人们发现，view和controller某些部件之间的局部性实际上强于controller内部的局部性。于是一种叫做控件（control）的预制组件开始出现了。

控件本身带有一定的交互功能，从MVC的视角来看，它既包含view，又包含controller，并且它通过"属性"，来把用户输入暴露给model。

controller的输入分配功能，则被操作系统提供的各种机制取代：

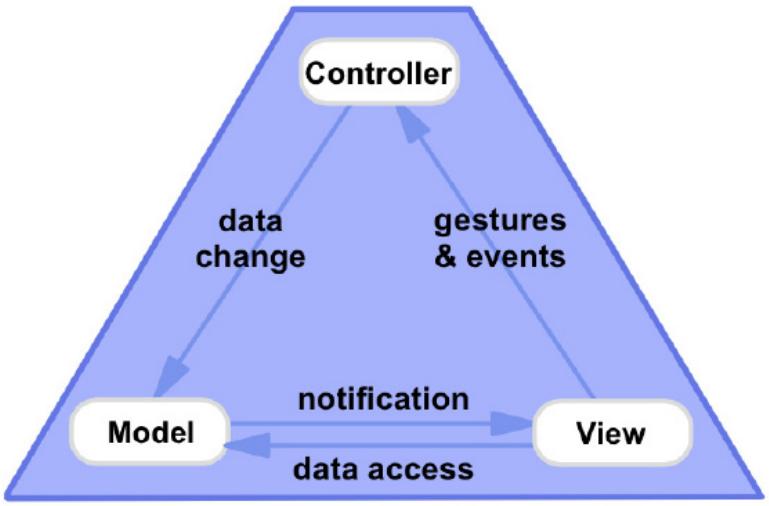
- 指针系统：少数DOS时代过来的程序员应该记得，20年前的程序中的“鼠标箭头”实际上是由各个应用自己绘制的，以MVC的视角来看，这应当属于一个"PointerView"的职责范畴。但是20世纪以后，这样的工作基本由操作系统的底层UI系统来实现了。
- 文本系统：今天我们几乎不需要再去关心文本编辑、选中、拖拽等逻辑，对web程序员可以尝试自己用canvas写一个文本编辑框来体验一下上个时代程序员编写程序的感受。你会发现，选中、插入/覆盖模式切换、换行、退格、双击、拖拽等逻辑异常复杂，经典MVC模式中通常使用TextView和TextEditor配合来完成这样的工作，但是今天几乎找不到需要我们自己处理这些逻辑的场景。
- 焦点系统：焦点系统通过响应鼠标、tab键等消息来使得控件获得操作系统级唯一的焦点状态，所有的键盘事件通常仅仅会由拥有焦点的控件来响应。在没有焦点系统的时代，操作系统通常是单任务的，但是即使是单一应用，仍然要自己管理多个controller之间的优先权和覆盖逻辑，焦点系统不但从技术上，也从交互设计的角度规范化了UI的输入响应，而最妙的是，焦点系统是对视觉障碍人士友好的，现在颇多盲人用读屏软件都是强依赖焦点系统的。

所以时至今日，MVC，尤其是其中controller的功能已经意义不大，若是在控件系统中，再令所有用户输入流经一个**controller**则可谓不伦不类、本末倒置。MVVM的提出者，微软架构师John Gossman曾言：“我倾向于认为它(指controller)只是隐藏到后台了，它仍然存在，但是我们不需要像是1979年那样考虑那么多事情了”

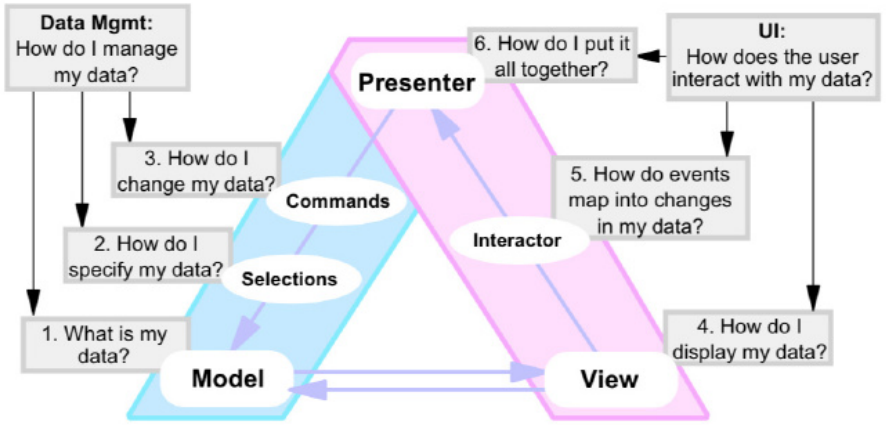
MVP

1996年，Taligent公司的CTO，Mike Potel在一篇论文中提出Model-View-Presenter的概念。

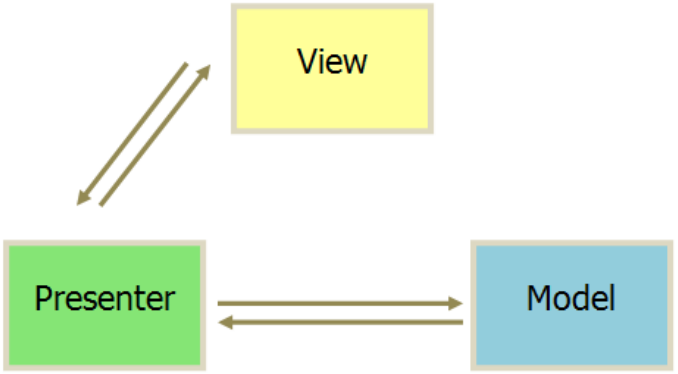
在这个时期，主流的view的概念跟经典MVC中的那个“永远不应该知道用户输入”的view有了很大的差别，它通常指本文中所述的控件，此时在Mike眼中，输入已经是由view获得的了：



Model-View-Presenter是在MVC的基础上，进一步规定了Controller中的一些概念而成的：



对，所以，不论你按照Mike还是Trygve的理解方式，**MVP**和**MVC**的依赖关系图应该是一！模！一！样！的！因为Mike的论文里说了“we refer to this kind（指应用程序全局且使用interactor, command以及selection概念的） of controller as a presenter”。presenter它就是一种controller啊！



把依赖关系画成这样也是醉了啊！不管你信不信我反正是不信啊！

标记语言和**MVVM**

随着20世纪初web的崛起，HTML跟JS这样标记语言+程序语言的组合模式开始变得令人注目

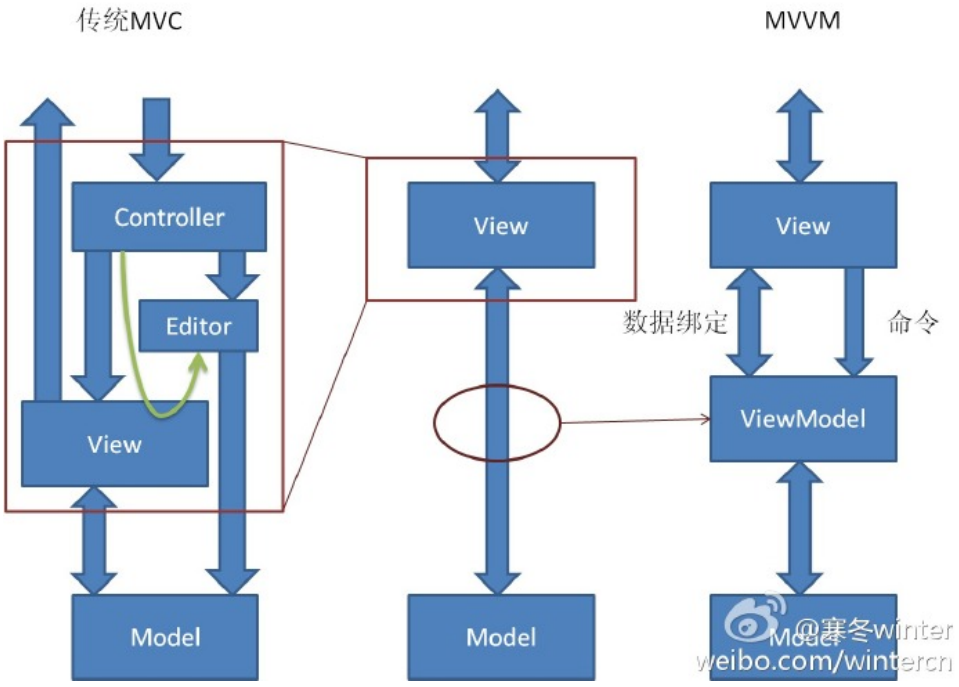
。逐渐推出的Flex、Sliverlight、QT、WPF、JSF、Cocoa等UI系统不约而同地选择了标记语言来描述界面。

在这样的架构中，view（或者说叫控件，不但是从依赖关系上跟程序的其他部件解耦，而且从语言上跟其它部分隔离开来。

标记语言的好处是，它可以由非专业的程序员产生，通过工具或者经过简单培训，一些设计师可以直接产生用标记语言描述的UI。想要突破这个限制使得view跟其它部分异常耦合可能性也更低。

然而这样的系统架构中，MVC和MVP模式已经不能很好地适用了。微软架构师John Gossman在WPF的XAML模式推出的同时，提出了MVVM的概念。

WPF得MVVM正式说明了它的view的概念跟MVC中的view的概念的区别。这里简单画了一下：



在MVVM模式中，数据绑定是最重要的概念，在MVC和MVP中的view和model的互相通讯，被以双向绑定的方式替代，这进一步把逻辑代码变成了声明模式。

结语

从经典MVC到MVVM，UI架构经过数次重大变迁，一些概念也在不断变化，架构和底层环境互相影响、适配，我认为时至今日，经典MVC已经不再是UI架构的正常选项。

更糟糕的是，今天无数经过演绎的MVC实现（如backbone）和科普文，要么是原本作者概念已经很混乱，掺杂私货，要么为了适配现代的标记语言和控件模式，自己修改了经典MVC中的一些概念和耦合关系。实际上今天MVC已经没法作为一种交流的标准词汇了。

写此文，希望大家能了解些历史上的发展历程，莫被不严谨的文章误导。其实本文的相当多观点也是经过演绎的，所以我附上所有原始文献链接，希望大家看了以后能有自己的判断：）也欢迎大家据此指出我理解的错误之处。

参考资料

- MVC论文
- MVC论文翻译
- MVP论文

- MVVM博文
- MVVM博文翻译

他们也赞了


美丽心…

Wend…

小_-柒

hellodon

微笑向…

绞杀巡回

imyaway


mcdon…

J_Y_Ng

Fooving

评论 | 赞过的人

发布

还没有人发表看法，赶快抢个沙发