深入了解MV**模式

> 前言：做客户端开发、前端开发，大致都应该听说过这么几个名词MVC、MVP、MVVM，这些架构的思想大多是为了解决界面应用程序复杂的逻辑问题。同时这些框架的核心目的在于，职责分离，不同的层次要做不同的事情。

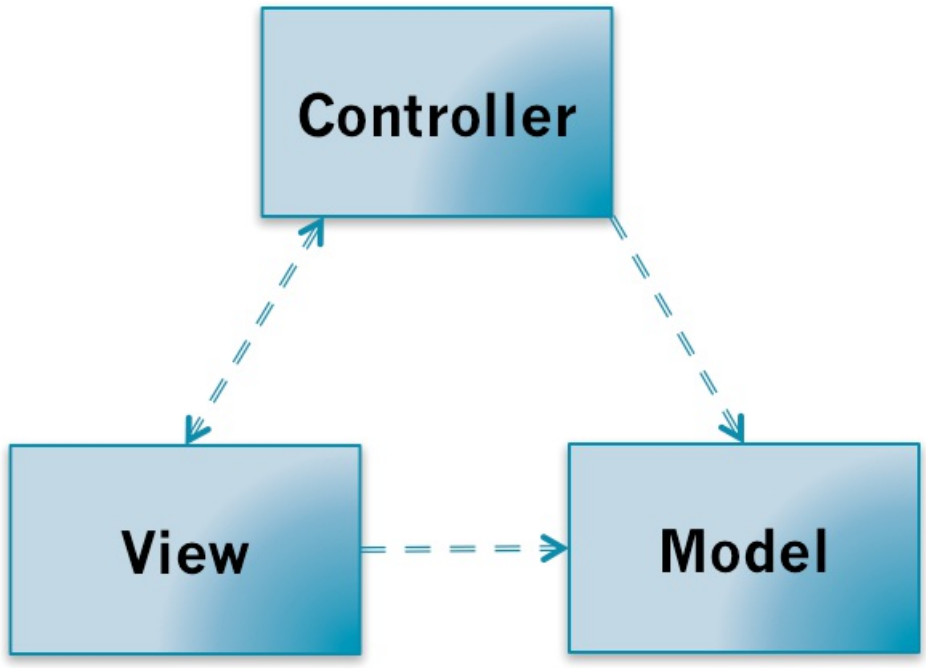
无论是哪种MV**系列，都涉及到了Model和View，如果单纯的只有Model和View，他们是没有办法一起协同工作的，所以就有了各种MV..的设计模式

MVXX模式：

- MVC
- MVP
- MVVM

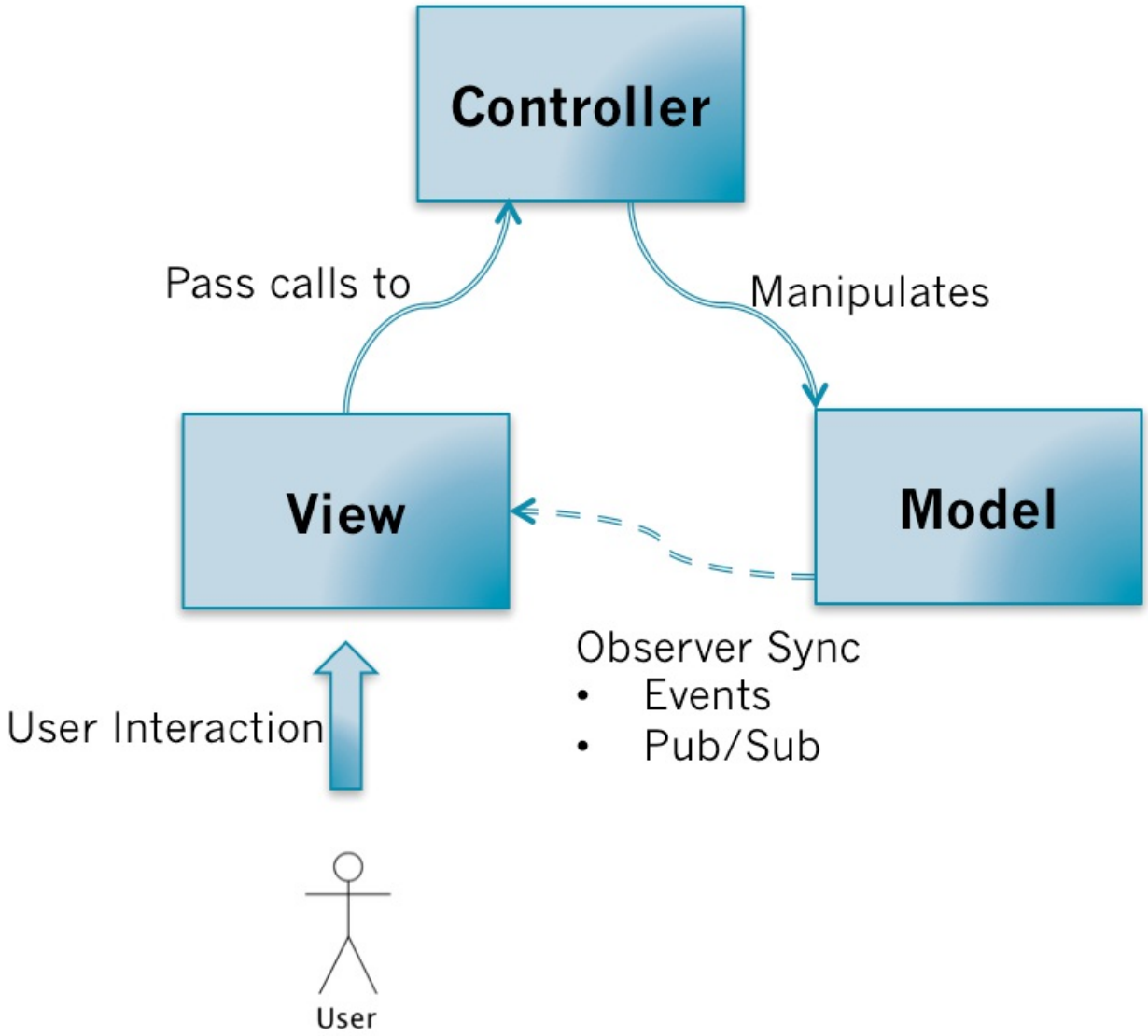
这三种架构模式都是现在比较流行的，在不同的项目中，可能采用不同的架构模式，今天我们就围绕着这三种架构模式的试用场景介绍一下他们的概念，以及异同。

MVC



MVC(Model-View-Controller),Model:逻辑模型，View:视图模型，Controller控制器。简单说这就是一种设计应用程序的思想，目的在于将业务逻辑、数据、界面分离，将业务逻辑聚集到一个部件里面，在改进或者想要定制界面及用户交互时，不需要重写编写业务逻辑。Controller和View都依赖Model层，Controller和View相互依赖。

操作的过程：用户在界面上进行操作(例如手机屏幕)，这个时候View会捕捉到用户的操作，然后将这个操作的处理权利交给Controller，Controller会对来自View的数据进行预处理，决定调用哪个Model的接口，然后由Model执行相应的逻辑，当Model变更之后，再利用观察者模式通知View，View通过观察者模式收到Model的消息之后，向Model请求最新的数据，然后更新页面，如下图：



看似没有什么特别的地方，但是由几个需要特别关注的关键点：

1. View是把控制权交移给Controller，Controller执行应用程序相关的应用逻辑（对来自View数据进行预处理、决定调用哪个Model的接口等等）。
2. Controller操作Model，Model执行业务逻辑对数据进行处理。但不会直接操作View，可以说它是对View无知的。
3. View和Model的同步消息是通过观察者模式进行，而同步操作是由View自己请求Model的数据然后对视图进行更新。

需要特别注意的是MVC模式的精髓在于第三点：Model的更新是通过观察者模式告知View的，具体表现形式可以是Pub/Sub或者是触发Events。而网上很多对于MVC的描述都没有强调这一点。通过观察者模式的好处就是：不同的MVC三角关系可能会有共同的Model，一个MVC三角中的Controller操作了Model以后，两个MVC三角的View都会接受到通知，然后更新自己。保持了依赖同一块Model的不同View显示数据的实时性和准确性。我们每天都在用的观察者模式，在几十年前就已经被大神们整合到MVC的架构当中。

优点

1. 首先最重要的一点是多个视图能共享一个模型，同一个模型可以被不同的视图重用，大大提高了代码的可重用性。
2. 由于MVC的三个模块相互独立，在其中改变一个，其他两个可以保持不变，这样可以把耦合降得很低，这样可以使开发人员可以只关注整个系统中的某一层
3. 控制器提高了应用程序的灵活性和可配置型。控制器可以用来连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

缺点

1. 增加了系统结构的实现的复杂性

2. 视图与控制器之间的联系过于紧密，视图如果没有控制器的存在，能够做的事情少之又少，这样视图就很难独立应用了
3. 视图对模型的数据的访问效率过低，因为之间需要多次的调用
4. View无法组件化，View依赖于特定的Model，如果需要把这个View抽出来用在下一个应用程序复用就比较困难了

MVP

Model(Model View Presenter),Model:逻辑模型，View:视图模型，Presenter:接口。

关于MVP的定义：

```
- MVC的演化版本，让Model和View完全解耦
- 代码清晰，不过增加了很多类
```

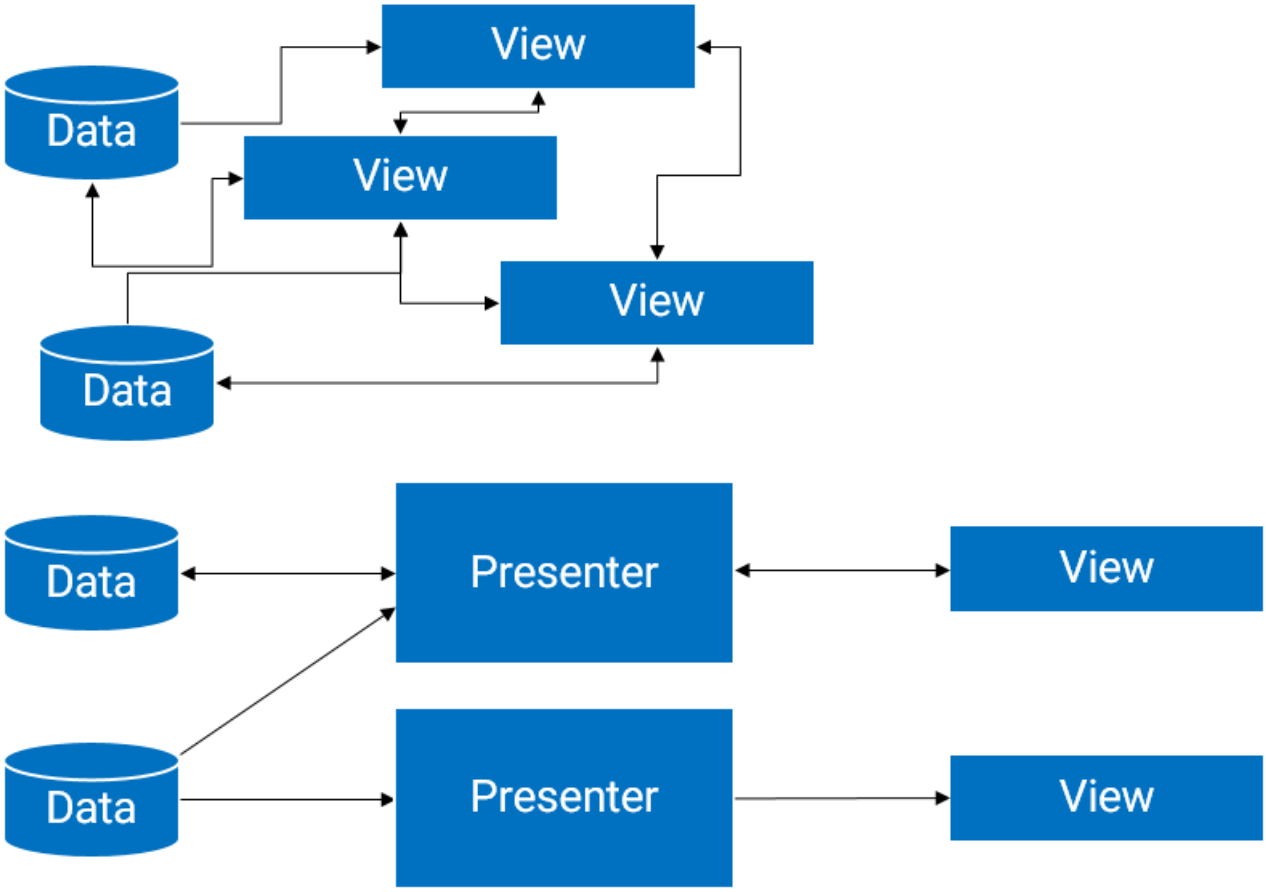
MVP中的P，是Presenter的含义，和MVC比较类似，都是将用户对View的操作交付给Presenter，Presenter会执行相应的逻辑，在这过程中会操作Model，当Model执行完业务逻辑之后，同样是通过观察者模式把自己的结果传递出去，不过不是告诉View，而是告诉Presenter，Presenter得到消息后，通过View的接口更新页面。

在Android中，我们的View可能仅仅就是个布局文件，它能做的事情少之又少，真正与布局文件进行数据绑定操作的事Activity，所以这样做就使Activity即像View又像Controller。

应用了MVP之后：

```
- View:对应Activity，负责View的绘制以及与用户交互
- Model:业务逻辑和实体模型
- Presenter:负责完成View与Model之间的交互
```

应用两张图来说明上述内容:



MVP的优点

1. 便于测试。Presenter对View是通过接口进行，在对Presenter进行不依赖UI环境的单元测试的时候。可以通过Mock一个View对象，这个对象只需要实现了View的接口即可。然后依赖注入到Presenter中，单元测试的时候就可以完整的测试Presenter应用逻辑的正确性。这里根据上面的例子给出了Presenter的单元测试样例。

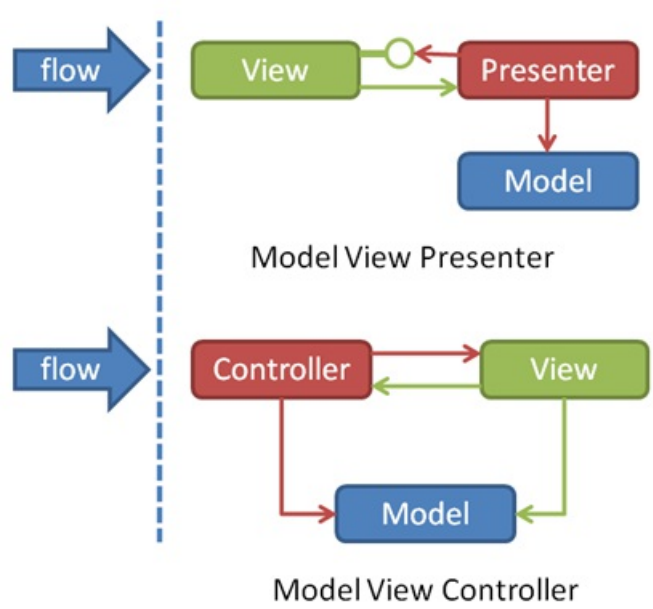
2. View可以进行组件化。在MVP当中，View不依赖Model。这样就可以让View从特定的业务场景中脱离出来，可以说View可以做到对业务完全无知。它只需要提供一系列接口提供给上层操作。这样就可以做到高度可复用的View组件。

MVP的缺点

1. 代码量会一些，实现的难度也会增加一些

MVP与MVC区别

如下图所示：

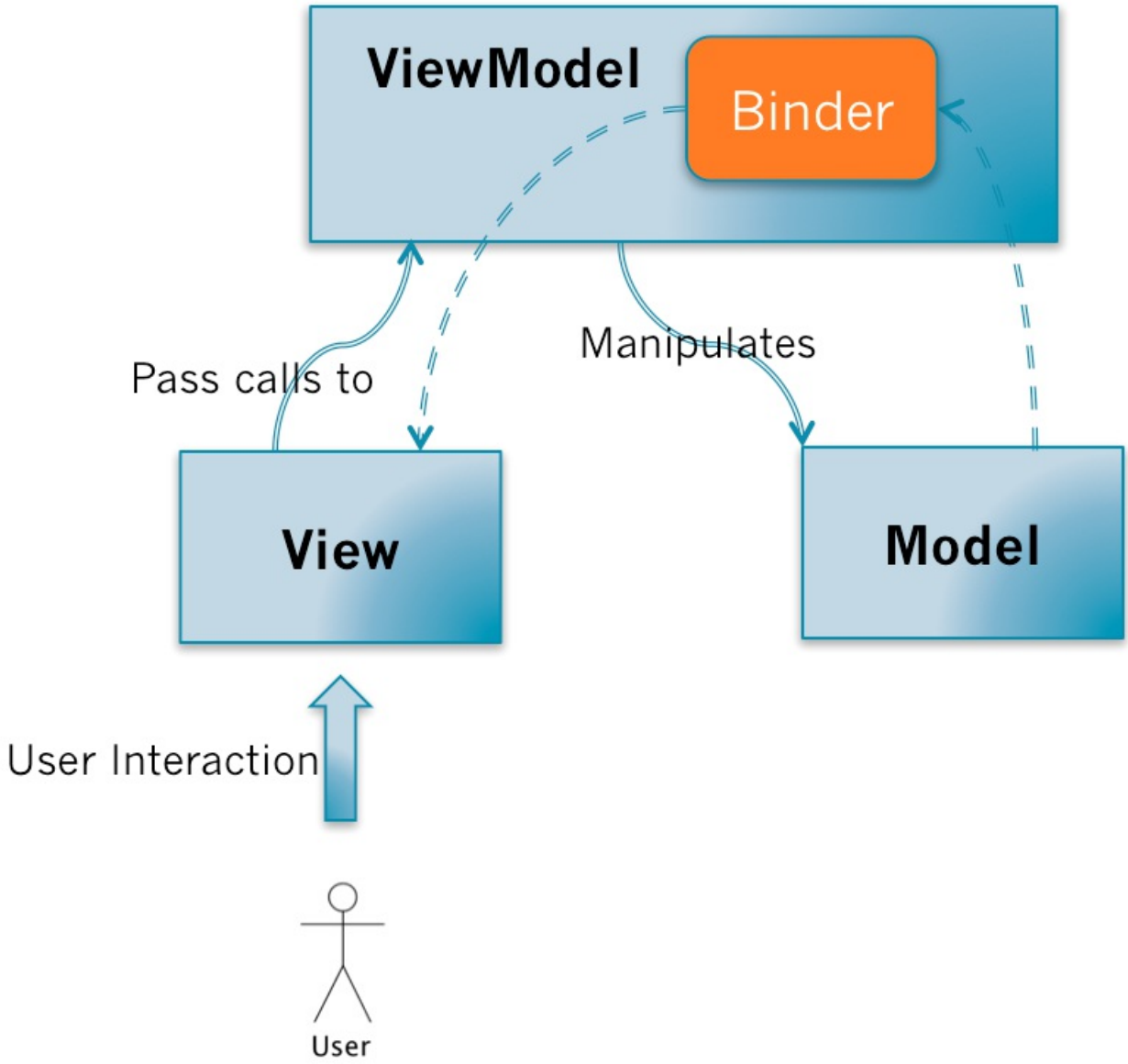


MVVM

MVVM是Model-View-ViewModel的简写，MVVM是思想上的一种变革，也可以看成是MVP的一种变革，它是将"数据模型数据双向绑定"的思想作为核心，因此在View和Model之间便不需要我们写联系了，我们在修改数据的时候视图就可以发生变化，我们在修改视图的时候数据也是会发生变化的，可以在一定程度上提高我们的开发效率的。

调用关系

MVVM的调用关系和MVP一样。但是，在ViewModel当中会有一个叫Binder，或者是Data-binding engine的东西。以前全部由Presenter负责的View和Model之间数据同步操作交由给Binder处理。你只需要在View的模版语法当中，指令式地声明View上的显示的内容是和Model的哪一块数据绑定的。当ViewModel对进行Model更新的时候，Binder会自动把数据更新到View上去，当用户对View进行操作（例如表单输入），Binder也会自动把数据更新到Model上去。这种方式称为：Two-way data-binding，双向数据绑定。可以简单而不恰当地理解为一个模版引擎，但是会根据数据变更实时渲染。



也就是说，MVVM把View和Model的同步逻辑自动化了。以前Presenter负责的View和Model同步不再手动地进行操作，而是交由框架所提供的Binder进行负责。只需要告诉Binder，View显示的数据对应的是Model哪一部分即可。

Android官方推出的MVVM的DataBinding，便是一个双向绑定的库。

优点

1. 提高可维护性。解决了MVP大量的手动View和Model同步的问题，提供双向绑定机制。提高了代码的可维护性。
2. 简化测试。因为同步逻辑是交由Binder做的，View跟着Model同时变更，所以只需要保证Model的正确性，View就正确。大大减少了对View同步更新的测试。

缺点

1. 过于简单的图形界面不适用，或说牛刀杀鸡。
2. 对于大型的图形应用程序，视图状态较多，ViewModel的构建和维护的成本都会比较高。
3. 数据绑定的声明是指令式地写在View的模版当中的，这些内容是没办法去打断点debug的。

相关资料

- 浅谈MVP in Android
- MVC三层框架详细解析
- 还原真实的MV*模式