# MVP in Android



Model： 封装的各种数据的实体类，以及提供对数据操作的接口，比如本地查询数据或者网上加载数据等操作。完全独立，只涉及数据层，不依赖ui。

View：xml布局文件以及Activity或Fragment

Xml布局文件编写静态ui界面，Activity或Fragment提供可交互的ui操作，比如弹窗或点击事件等。

Activity或Fragment里通常拥有Presenter的引用，各种业务操作可先定义为接口，由Presenter来实现，然后通过Presenter的引用来调用。

Presenter：业务层操作

通常拥有Activity或Fragment的引用，也拥有Model层的引用，先通过Model层的引用获取数据后再通过Activity的引用来通知Activity该做出的反应。

**代码示例**

# RxJava

抛物线： <http://gank.io/post/560e15be2dca930e00da1083>

Github： <https://github.com/ReactiveX/RxJava>    
<https://github.com/ReactiveX/RxAndroid>

RxJava是什么？------异步，一个在JAVA VM上使用可观测的序列来组成异步、基于事件的程序的库

RxJava 有四个基本概念：Observable (可观察者，即被观察者)、 Observer (观察者)、 subscribe (订阅)、事件。Observable 和 Observer 通过 subscribe() 方法实现订阅关系，从而 Observable 可以在需要的时候发出事件来通知 Observer。

与传统观察者模式不同， RxJava 的事件回调方法除了普通事件 onNext() （相当于 onClick() / onEvent()）之外，还定义了两个特殊的事件：onCompleted() 和 onError()。



使用方式：（被观察者.订阅（观察者），虽然与现实相反，但这样支持链式调用）

代码示例：



# 观察者模式（发布/订阅模式）

**概念**

现实概念上的观察者模式：A对象（观察者）需要时时关注B对象（被观察者）的某种状态是否发生改变，在发生变化的一瞬间做出反应。

程序里的观察者模式：B对象（被观察者）的某种状态发生变化时，需要主动通知A对象（观察者），让A对象（观察者）做出反应。所以也称作发布/订阅模式，A对象（观察者）订阅B对象（被观察者）的某一状态，当B对象的被订阅的状态发生变化时，就发布通知，告知所有订阅这一状态的观察者需要做出相应的反应了。

两者的区别有点类似于《计算机组成原理》里的程序查询方式和中断方式，一种是主动查询，一种是被动等待通知。

**Android源码里的例子**

OnClickListener()

-按钮的点击事件

Button-----被观察者 onClickListener-----观察者（观察Button的onClick()状态）

两者通过Button.setOnClickListener(onClickListener)来建立订阅关系



**代码示例**

# VerticalGridView

# PropertyValuesHolder

用途：可以用来实现view的抖动

使用方法：

第一步：生成Keyframe对象；  
第二步：利用PropertyValuesHolder.ofKeyframe()生成PropertyValuesHolder对象  
第三步：ObjectAnimator.ofPropertyValuesHolder()生成对应的Animator

示例代码：

Keyframe frame0 = Keyframe.ofFloat(0f, 0);

Keyframe frame1 = Keyframe.ofFloat(0.1f, -20f);

Keyframe frame2 = Keyframe.ofFloat(1, 0);

PropertyValuesHolder frameHolder = PropertyValuesHolder.ofKeyframe("rotation",frame0,frame1,frame2);

 Animator animator = ObjectAnimator.ofPropertyValuesHolder(mImage,frameHolder);

animator.setDuration(1000);

animator.start();

Keyframe.ofFloat(0, 0)表示动画进度为0时，动画所在的数值位置为0；Keyframe.ofFloat(0.25f, -20f)表示动画进度为25%时，动画所在的数值位置为-20；Keyframe.ofFloat(1f,0)表示动画结束时，动画所在的数值位置为0；

# ViewStub

在开发应用程序的时候，经常会遇到这样的情况，会在运行时动态根据条件来决定显示哪个View或某个布局。那么最通常的想法就是把可能用到的View都写在上面，先把它们的可见性都设为View.GONE，然后在代码中动态的更改它的可见性。这样的做法的优点是逻辑简单而且控制起来比较灵活。但是它的缺点就是，耗费资源。虽然把View的初始可见View.GONE但是在Inflate布局的时候View仍然会被Inflate，也就是说仍然会创建对象，会被实例化，会被设置属性。也就是说，会耗费内存等资源。

      推荐的做法是使用[***android***](http://lib.csdn.net/base/15).view.ViewStub，ViewStub 是一个轻量级的View，它一个看不见的，不占布局位置，占用资源非常小的控件。可以为ViewStub指定一个布局，在Inflate布局的时候，只有 ViewStub会被初始化，然后当ViewStub被设置为可见的时候，或是调用了ViewStub.inflate()的时候，ViewStub所向 的布局就会被Inflate和实例化，然后ViewStub的布局属性都会传给它所指向的布局。这样，就可以使用ViewStub来方便的在运行时，要还 是不要显示某个布局。

      但ViewStub也不是万能的，下面总结下ViewStub能做的事儿和什么时候该用ViewStub，什么时候该用可见性的控制。

     首先来说说ViewStub的一些特点：

         1. ViewStub只能Inflate一次，之后ViewStub对象会被置为空。按句话说，某个被ViewStub指定的布局被Inflate后，就不会够再通过ViewStub来控制它了。

         2. ViewStub只能用来Inflate一个布局文件，而不是某个具体的View，当然也可以把View写在某个布局文件中。

     基于以上的特点，那么可以考虑使用ViewStub的情况有：

         1. 在程序的运行期间，某个布局在Inflate后，就不会有变化，除非重新启动。

              因为ViewStub只能Inflate一次，之后会被置空，所以无法指望后面接着使用ViewStub来控制布局。所以当需要在运行时不止一次的显示和 隐藏某个布局，那么ViewStub是做不到的。这时就只能使用View的可见性来控制了。

         2. 想要控制显示与隐藏的是一个布局文件，而非某个View。

              因为设置给ViewStub的只能是某个布局文件的Id，所以无法让它来控制某个View。