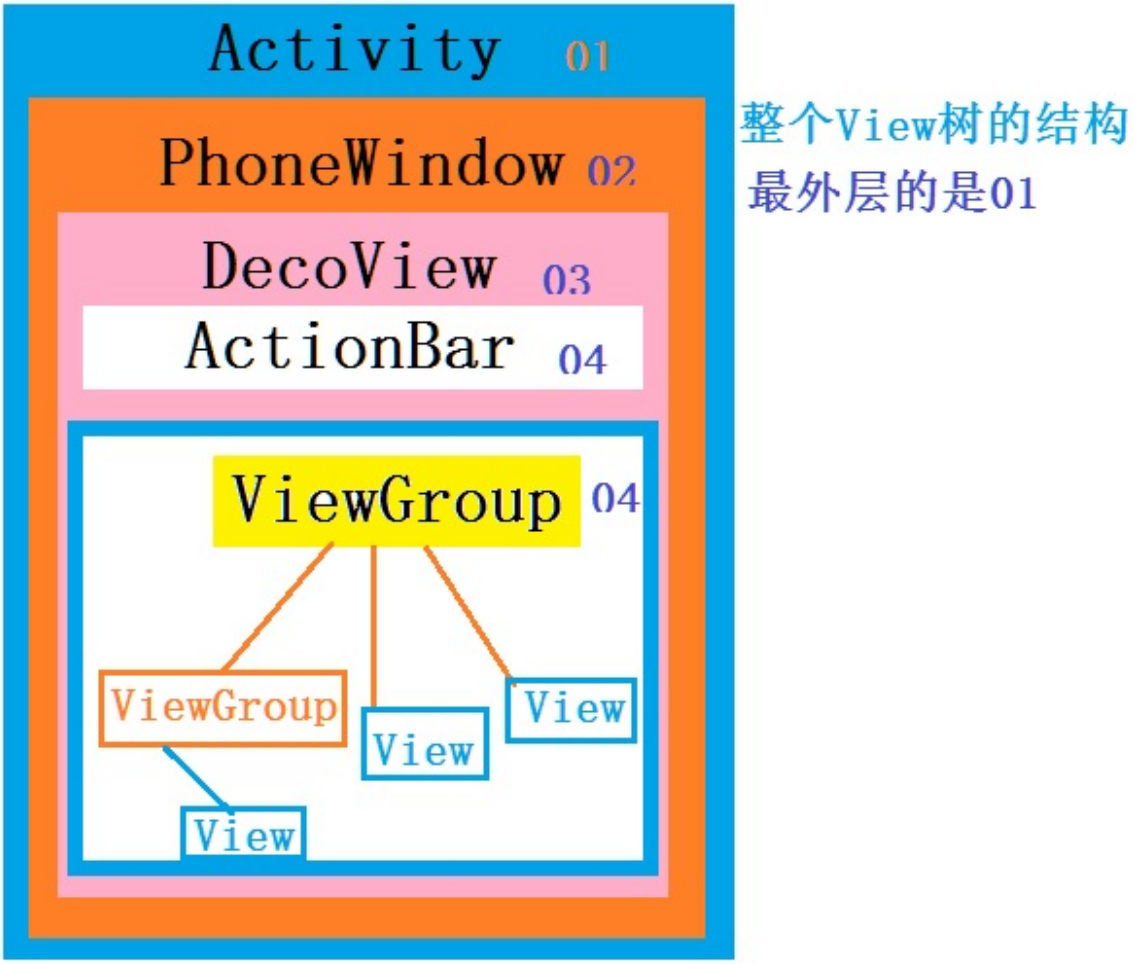
任何一个View树的结构



通过上图可以看出，对每个具体View对象的操作，其实就是个递归的实现

Android中的任何一个布局、任何一个控件其实都是直接或间接继承自View实现的，当然也包括我们后面一步一步引出的自定义控件也不例外，所以说这些View应该都具有相同的绘制流程与机制才能显示到屏幕上（因为他们都具备相同的父类View，可能每个控件的具体绘制逻辑有差异，但是主流程都是一样的）。经过总结发现每一个View的绘制过程都必须经历三个最主要的过程，也就是measure、layout和draw。

对于View的绘制开始调运地方，整个View树的绘图流程是在ViewRootImpl类的performTraversals()方法（这个方法巨长）开始的，该函数做的执行过程主要是根据之前设置的状态，判断是否重新计算视图大小(measure)、是否重新放置视图的位置(layout)、以及是否重绘 (draw)，其核心也就是通过判断来选择顺序执行这三个方法中的哪个

关于ViewRootImpl类　实现类

1、ViewRoot是老版本中的一个类，在Android2.2以后用ViewRootImpl代替ViewRoot，对应于ViewRootImpl.java

2、源码路径在SDK/source/android XX，是一个系统隐藏的类只能在源码中去找

3、一个视图层次结构类,实现所需的视图之间的协议，这在很大程度上是一个内部实现。

4、是链接WindowManager和DecorView的纽带，另外View的绘制也是通过ViewRootImpl来完成的

5、主要作用

A：链接WindowManager和DecorView的纽带，更广一点可以说是Window和View之间的纽带。

B：完成View的绘制过程，包括measure、layout、draw过程。

C：向DecorView分发收到的用户发起的event事件，如按键，触屏等事件。

6、View与WindowManager之间的联系，WindowManager所提供的功能很简单，常用的只有三个方法，即添加View，更新View和删除View，还有其它功能，比如改变Window的位置，WindowManager操作Window的过程更像是在操作Window中的View，这三个方法定义在ViewManager中，而WindowManager继承了ViewManager。 `public interface ViewManager`

7、addView大概一个过程如下： DECOVIEW 得口唯有 ViewRootImpl 唯有肉吃嗯跑

WindowManager——>WindowManagerGobal——>ViewRootImpl——>Session——>WindowManagerService
那么WindowManager又是如何与DecorView相连的呢，最终DecorView肯定是要添加到Window上的，而Window的具体实现类是PhoneWindow，因为DecorView嵌入在Window上

8、在ActivityThread中，当Activity对象被创建完毕后，会将DecorView添加到Window中，同时会创建ViewRootImpl对象，并将ViewRootImpl对象和DecorView建立关联，可以参考一下代码，在ActvityThread中，也就是ViewRootImpl是DecorView的父元素，但是ViewRootImpl并不是View。

在Activity调用setContentView会调用PhoneWindow的setContentView，最后会调用DecorView的addView方法，这也说明了我们添加的View是DecorView的子元素。

PerformTraversals（ ）View的绘制开始调运地方 破佛母 try飞尚四

该方法是在ViewRootImpl.java文件中，一旦触发该操作，就会从decorView开始进行measure，Layout，draw了。
在scheduleTraversals中会调用Choregrapher.postCallback，将它post出去，而postSyncBarrier方法禁止了后续的消息处理，一旦post出去了同步的Barrier之后，所有的非

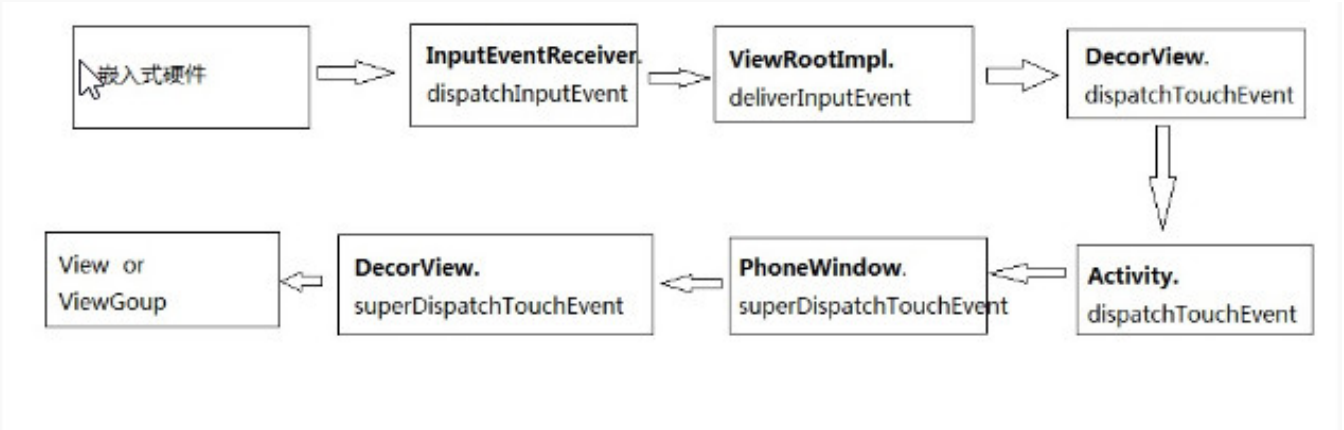
异步调用的消息就会被停止分发。

performTraversals方法会经过measure、layout和draw三个过程才能将一个View绘制出来，所以View的绘制是ViewRootImpl完成的，另外当手动调用invalidate，postInvalidate，requestInvalidate也会最终调用performTraversals，来重新绘制View。其中requestLayout()方法会调用measure过程和layout过程，不会调用draw过程，也不会重新绘制任何View包括该调用者本身。

向DecorView分发事件

包括MotionEvent，还有KeyEvent。我们知道View的时间分发顺序为Activity——>Window——>View，和ViewRootImpl有很大的关系。

首先，事件的根本来源来自于Native层的嵌入式硬件，然后会经过InputEventReceiver接受事件，然后交给ViewRootImpl，将事件传递给DecorView，DecorView再交给PhoneWindow，PhoneWindow再交给Activity。这样看来，整个体系的事件分发顺序为：



InputEvent输入事件，它有2个子类：KeyEvent和MotionEvent，其中KeyEvent表示键盘事件，而MotionEvent表示点击事件，这里InputEventReceiver译为输入事件接收者，顾名思义，就是用于接收输入事件，然后交给ViewRootImpl的dispatchInputEvent方法去分发处理。可以看到mHandler将逻辑切换到UI线程

简单总结：

1、任何一个Viewr的结构都是由Activity------->PhoneWindow--------->DecoView-------->ActionBar--------ViewGroup/View

2、任何一个布局、任何一个控件其实都是直接或间接继承自View实现的

3、任何一个View的绘制流程都必须要经过3个主要的过程，也就是measure、layout和draw。（是否重

新测量，是否重新布局，是否重新绘制）

4、整个View树的绘图流程是在ViewRootImpl类的performTraversals()方法中开始的，这个方法来决定绘制的过程

5、ViewRootImpl类 是系统隐藏的类，主要作用是来：

1.实现所需的视图结构之间的所有协议

2.是Window和View之间的纽带

3.完成View的绘制过程

4.向DecorView分发收到的用户发起的event事件，如按键，触屏等事件。