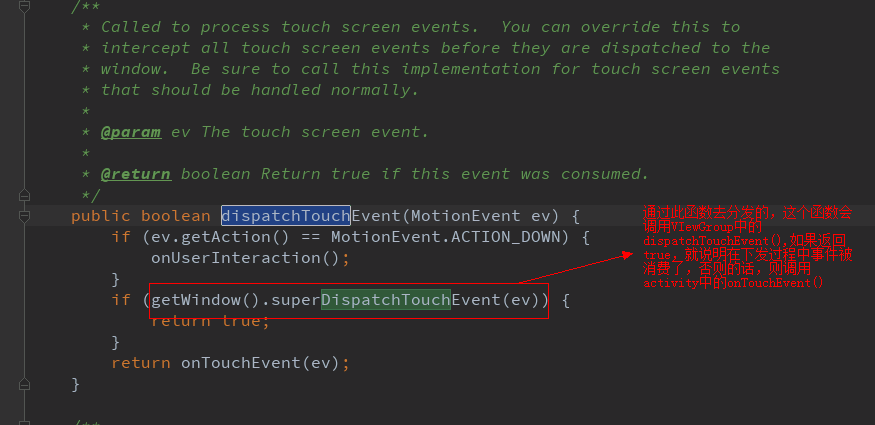
Android事件分发中三个层次：

Activity

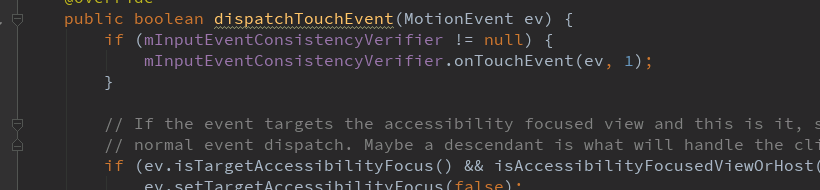
ViewGroup

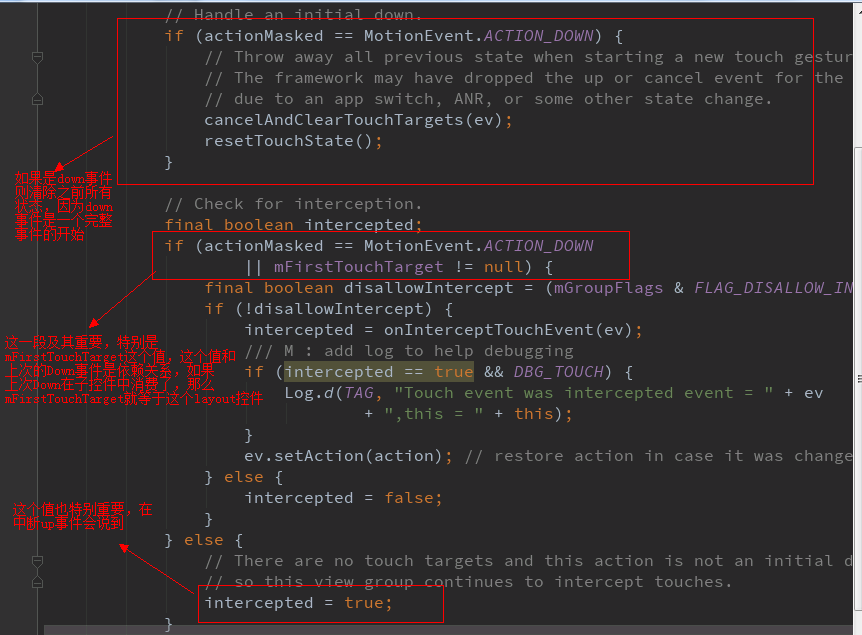
View

Activity.java是通过dispatchTouchEvent函数分发事件，如何分发的呢？

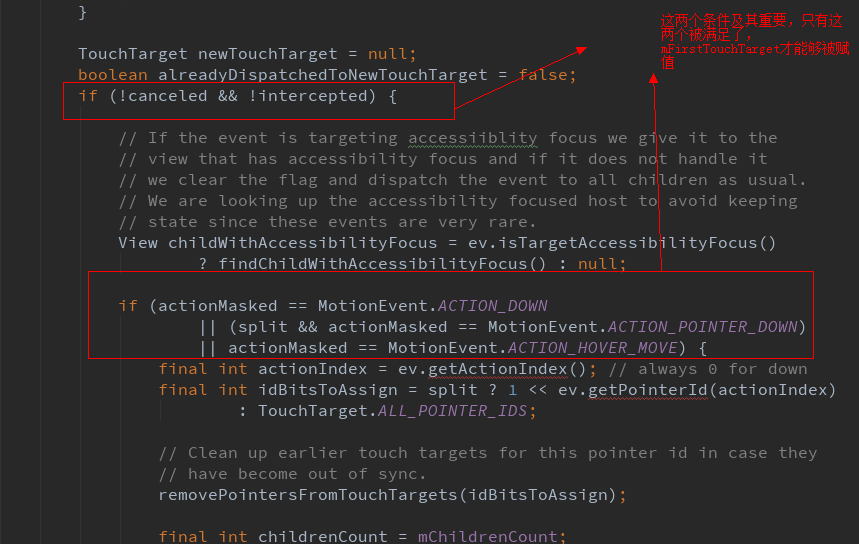


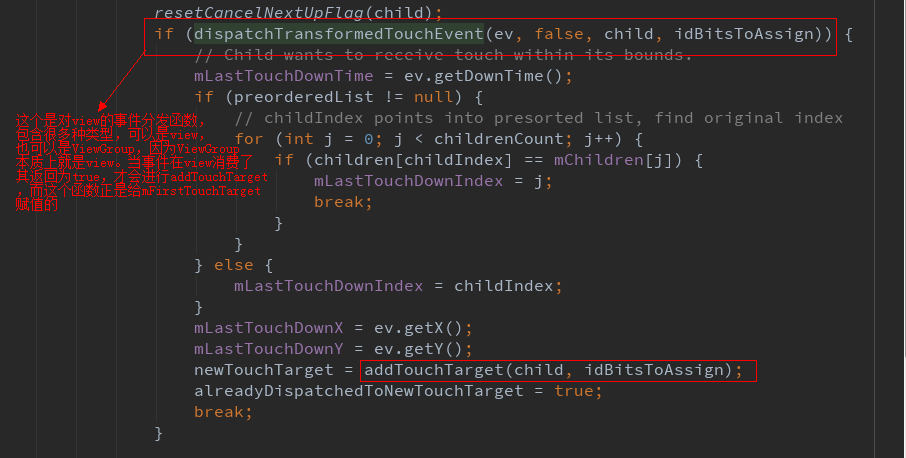
接下来就是ViewGroup.java分析了（注意从activity到我们指定的ViewGroup---我们的布局，这个层次中还有父layout，这个要注意，因为在事件中断的过程中会被用到）





mFirstTouchTarget这个值，如何赋值的呢(必须指明的是，这个是down事件才可以走的流程)，看下图：

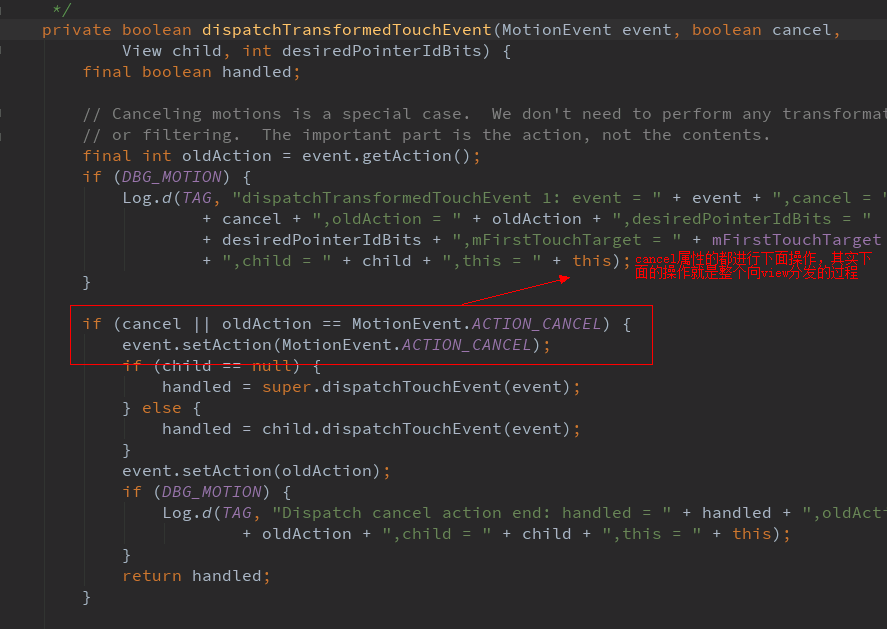




下面这个是所有事件必须走的流程。



然后看下ViewGroup事件如何通过dispatchTransformedTouchEvent（）函数分下下去的。

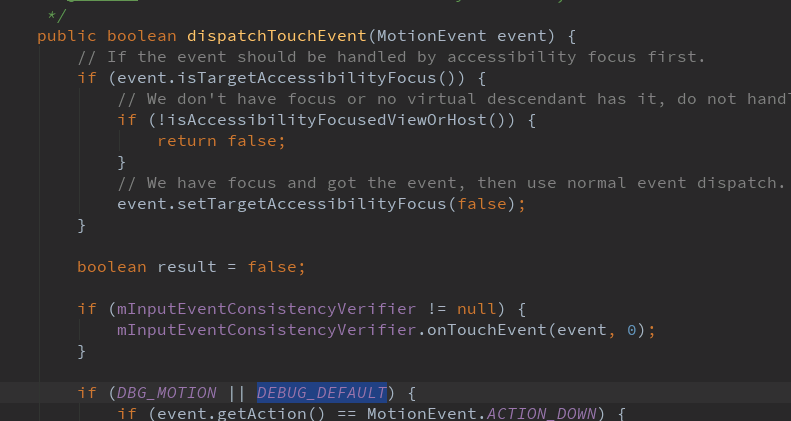


总结上面ViewGroup四个主要的判断地方，

1. 根据是否是Down事件或者mFirstTouchTarget是不为空，来给interacted的值
2. 根据canceled和interacted的值来决定是否进入函数体
3. 在函数体中，根据是否是down事件来进行子控件的遍历
4. 在子控件遍历中，找到对应子控件，然后分发

上面判断完成后，就进入通用代码段了。

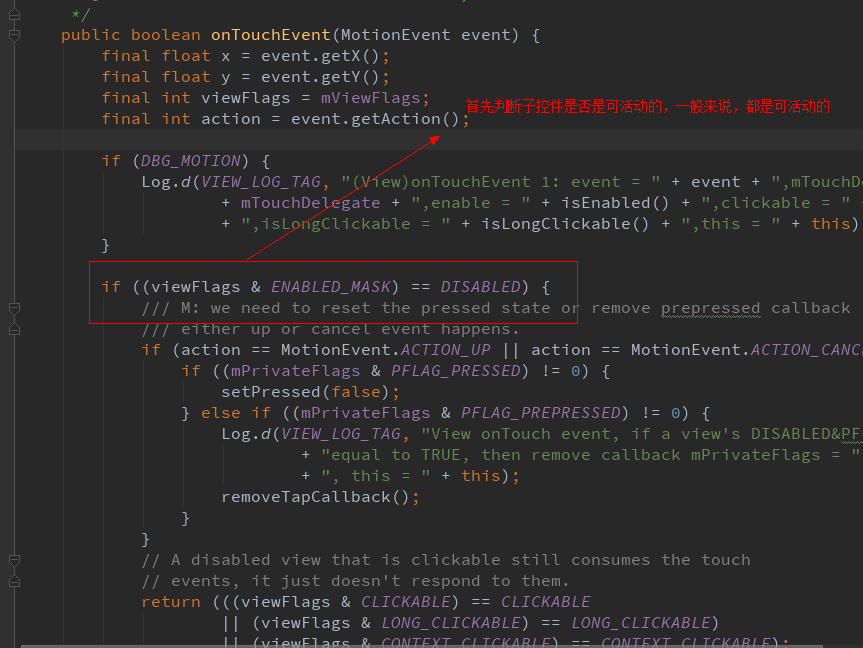
下面就看View.java的分发流程了

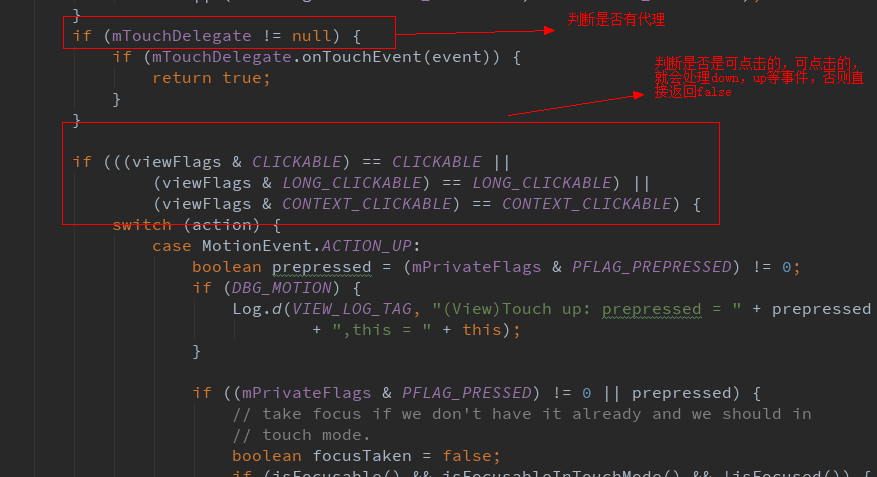


核心部分请看下面，

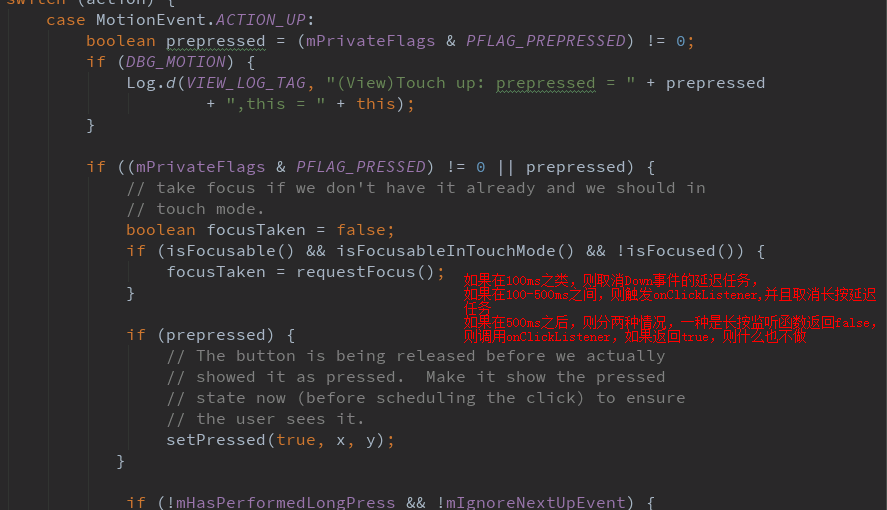
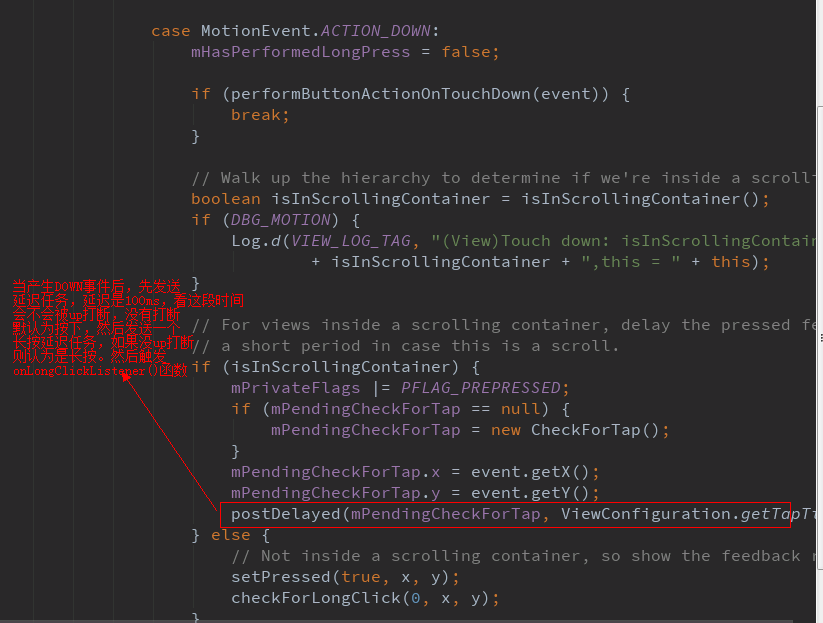


下面具体分析下onTouchEvent（）函数式如何处理这些事件的





下面是Down和up事件的流程（move事件和up一样）



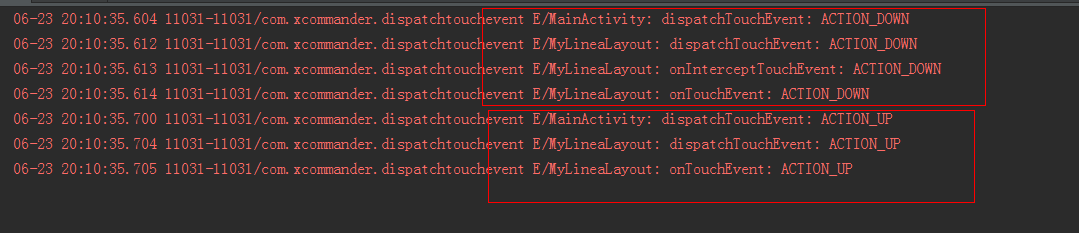
注意的是onTouchEvent函数中最重要的判断是是否是可点击的，是的话，则返回true，表示消费，不是的话，返回false，表示未消费。

下面是关于分发的总结，

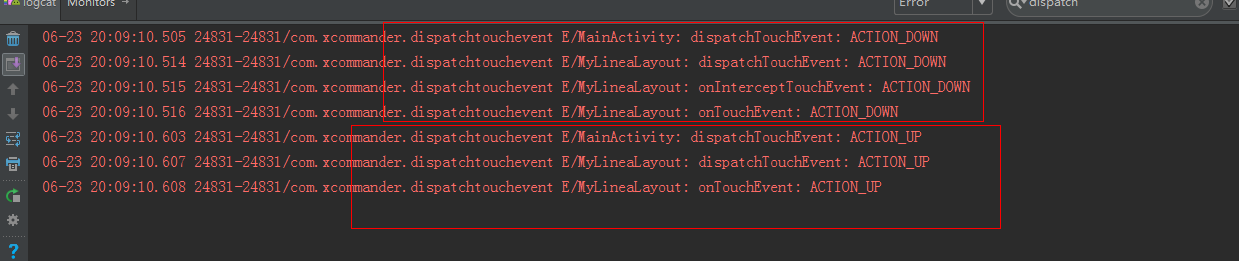
关于中断，

（1）如果down事件在ViewGroup被中断，（我们将ViewGroup中的onTouchEvent返回了true）那么mFirstTouchTarget就会为null，自己分发自己，然后返回true，然后回溯到父layout，mFirstTouchTarget被赋予值，我们应该从mFirstTouchTarget不为空的控件来分析。分两种情况：

Up事件在ViewGroup中没有被设置中断，因为mFirstTouchTarget不为null，所以会继续下面分发，当父layout分发到ViewGroup，就会进入通用代码，然后ViewGroup分发自己，并且返回true。截图如下



Up事件被设置中断了，如上面所说，到了父layout，但是父layout没有被中断，所以父控件分发到ViewGroup中，进入通用代码，然后ViewGroup分发给自己，然而ViewGroup中的mFirstTouchTarget为null（没有进行addOttch方法），所以进入通用代码的第一段，又自己分发自己，并且返回为true，截图如下：



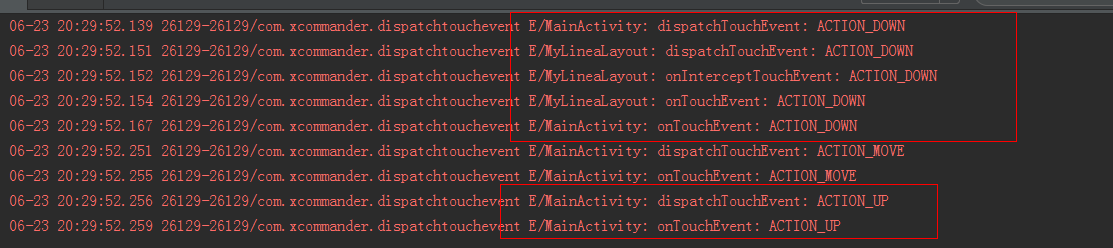
如上面黄色两个部分所说，我们从mFirstTouchTarget不为null的最近节点来分析，不管

ViewGroup设不设置中断，都不会影响的。首先是进入ViewGroup中，mFirstTouchTarget为null，为什么为null，down事件被中断了，导致VIewGroup这段代码中mFirstTouchTarget为null。因为是null，所以导致通用代码中第一段，自己分发自己（把ViewGroup当做view来分发）。所以设置中断和没设置中断一样。

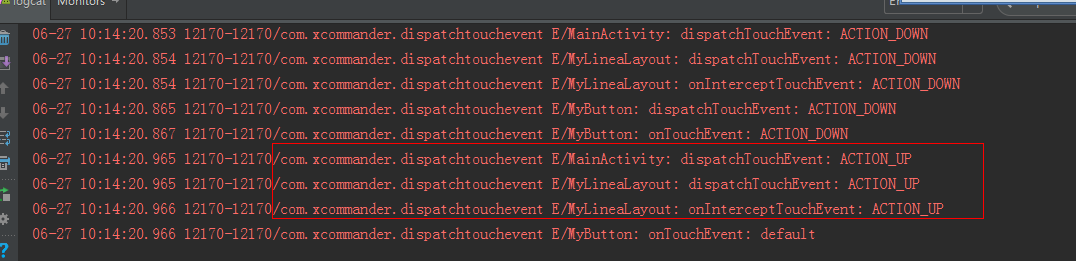
(总结得出如果mFirstTouchTarget为null，则自己分发自己，这个是一种情况，在down事件被中断后，且返回的是true（这个导致上一层的dispatchTransFrom为true，上一层的mFirstTouchTarget不为null，才会有本层ViewGroup的分发），则up事件不管有没有中断，都会从activity dispatch到viewgroup在从viewgroup dispatch时由于mFirstTouchTarget为null，自己分发了自己，直接onTouchEvent，根本不会走)

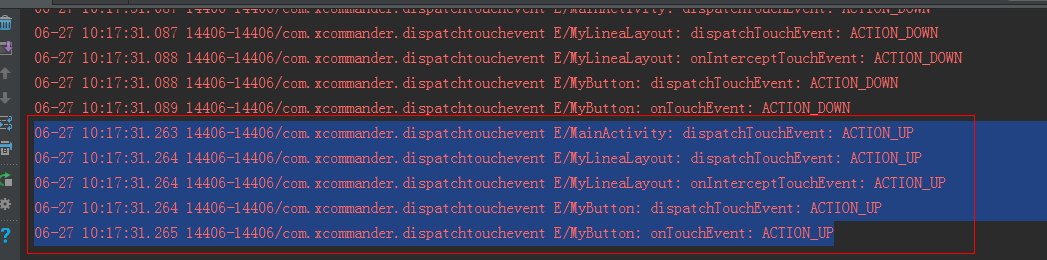
我们将ViewGroup中的onTouchEvent返回了false的时候，down事件和上述一样，由于返回了false，导致上一层的dispatchTransFromEvent为false，然后mFirstTouchTarget为null，不管up事件为不为有没有中断，都不会走viewgroup，因为根本就没分发下来。

以上两种情况。



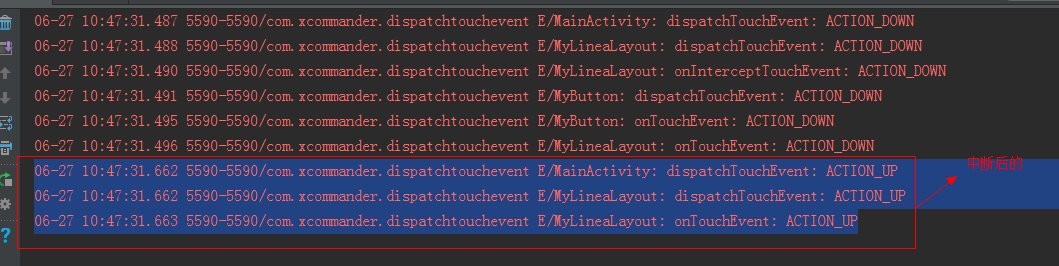
（2）down事件没有被中断，主要看分发到子控件的onTouchEvent返回值，返回为true，则ViewGroup的mFirstTouchTarget不为null，否则为null;mFirstTouchTarget不为null，当up事件下发时，ViewGroup中的mFirstTouchTarget不为null，当up中断时，up动作会变成cancel事件，下发到子控件去。如下图

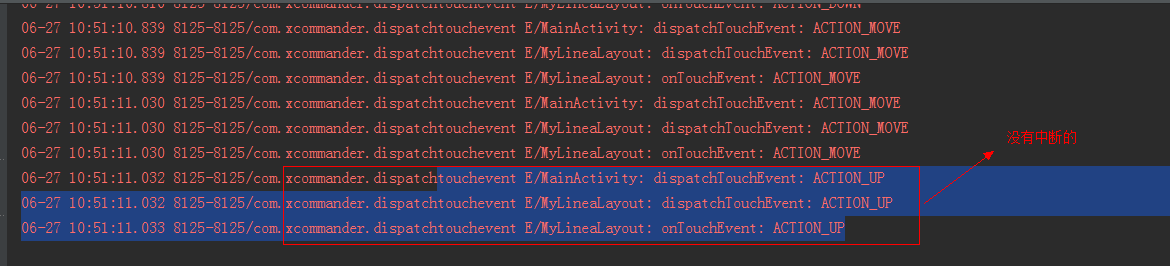
 当up不中断，其就会下发到子view中



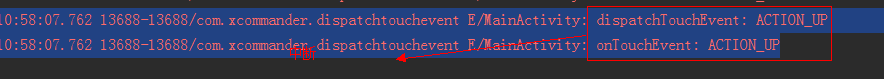
下面就来说，down事件中，子控件onTouchEvent返回false，则mFirstTouchTarget为null，此时ViewGroup必然分发自己，此时看ViewGroup中的onTouchEvent返回true或者false，

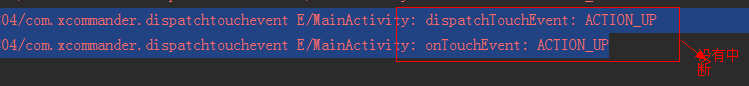
返回true，则父layout中的mFirstTouchTarget不为null，up事件那么就能下发到ViewGroup中来，因为ViewGroup中mFirstTouchTarget为null，up事件不管中不中断都不会影响，都会走自己分发自己。（从两张图可以看出，走的都一样）





当ViewGroup中的onTouchEvent返回为false，那么父layout中mFirstTouchTarget为null，则直接回溯到activity中去了，up事件根本不会下发下来。对比以下两张图就知道了。





关于中断写的太多了，现在把整个流程用流程图画出来（当然包括中断了）

关于down事件和up事件（move事件和up事件一样）的事件消费的大致总结:前提是要保持一样的行为，比如要中断就一起都中断

如果down事件在哪个部分消费，那么up事件就会立刻转向这个部分，来进行消费。返回true就是表示消费。

注意如果事件没分发下来，很可能被中断了（默认是不拦截的），导致子view分发不到，如何不中断，ViewGroup有个requestDisallowInterceptTouchEvent()方法可以设置不中断，这样就可以分发到子控件。

事件分发先告一段落，后续再观察。

记住，java子类继承父类，那么子类就具有父类的任何方法和属性。在使用一个方法的时候，先确定这个方法属于哪个类。