android中Touch是常用的事件，尤其在自定义的控件中，android中主要有三个地方可以处理Touch事件：

**（1）在View中，有两个回调函数**

Public Boolean dispatchTouchEvent(MotionEvent ev)

Public Boolean onTouchEvent(MotionEvent ev)

**（2）在ViewGroup中，有三个回调函数**

Public Boolean dispatchTouchEvent(MotionEvent ev)

Public Boolean onInterceptTouchEvent(MotionEvent ev)

Public Boolean onTouchEvent(MotionEvent ev)

**（3）在Activity中有三个回调函数**

Public Boolean dispatchTouchEvent(MotionEvent ev)

Public Boolean onTouchEvent(MotionEvent ev)

在这三个不同的地方，它们对Touch事件的处理大致一样，但也有不同的地方，下面对View对Touch的处理过程，首先看看源码：

**public** **boolean** dispatchTouchEvent(MotionEvent event) {

**if** (mInputEventConsistencyVerifier **!= null**) {

mInputEventConsistencyVerifier.onTouchEvent(event, 0);

}

**if** (onFilterTouchEventForSecurity(event)) {

//noinspection SimplifiableIfStatement

**if** (mOnTouchListener != **null** && (mViewFlags & ENABLED\_MASK) == ENABLED && mOnTouchListener.onTouch(**this**, event)) {

**return** **true**;

}

**if** (onTouchEvent(event)) {

**return** **true**;

}

}

**if** (mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onUnhandledEvent(event, 0);

}

**return** **false**;

}

从源码中可以看出，对于View的处理，首先交由onTouchListener进行处理，如果设置了onTouchListener，且onTouchListener的onTouch事件返回true，则该View的onTouchEvent不执行，否则执行该View的onTouchEvent事件

**例子测试：**

**下面函数包含于DrawView（继承于View）中**

@Override

**public** **boolean** onTouchEvent(MotionEvent event) {

Log.*v*("test", "mydrawView touch down");

**return** **super**.onTouchEvent(event);

}

在Activity中实现一个Touch监听器：

DrawView drawView = **new** DrawView(**this**);

drawView.setOnTouchListener(**new** View.OnTouchListener() {

@Override

**public** **boolean** onTouch(View arg0, MotionEvent arg1) {

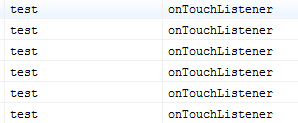
Log.*v*("test", "onTouchListener");

**return** **true**;

}

});

在监听器的onTouch函数中如果返回true，则执行结果为：



在监听器的onTouch函数中如果返回false，则执行结果为：



如果重写了dispatchTouchEvent函数，且监听器的onTouch函数返回false

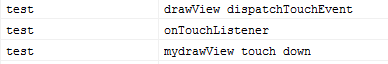
@Override

**public** **boolean** dispatchTouchEvent(MotionEvent event) {

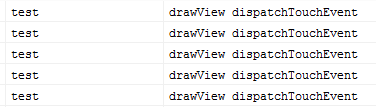
Log.*v*("test", "drawView dispatchTouchEvent");

**return** **super**.dispatchTouchEvent(event);

}



不设置监听器， 同时将dispatchTouchEvent返回true



修改dispatchTouchEvent函数的log为：

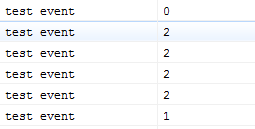
**public** **boolean** dispatchTouchEvent(MotionEvent event) {

Log.*v*("test event", "" + event.getAction());

**return** **true**;

}

运行结果为：



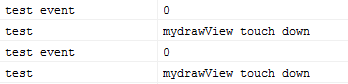
修改dispatchTouchEvent为：

**public** **boolean** dispatchTouchEvent(MotionEvent event) {

Log.*v*("test event", "" + event.getAction());

**return** **super**.dispatchTouchEvent(event);

}



将onTouchEvent函数改为：

**public** **boolean** onTouchEvent(MotionEvent event) {

Log.*v*("test", "mydrawView touch down");

**switch**(event.getAction()){

**case** MotionEvent.*ACTION\_DOWN*:

**return** **true**;

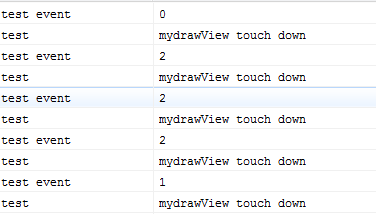
**default**:

**break**;

}

**return** **super**.onTouchEvent(event);

}



可以看到如果我们在当前的View接受处理了ACTION\_DOWN消息，则可以接受以后的一系列消息，如果没有处理ACTION\_DOWN消息，则以后的一系列消息都不接受，实际上每个触屏事件都是以ACTION\_DOWN开头，后面跟一系列的ACTION\_MOVE，最后再有一个ACTION\_UP组成的，也就是说如果没有处理ACTION\_DOWN消息，则后面的ACTION\_MOVE与ACTION\_UP都不会接收到。

**重要概述：**

Touch事件传递过程中经过的元素都是一个View，但是事件处理的最外层的元素却不是View，除了跟Window有关的事件，当一个Touch事件发生的时候，会首先调用当前Activity的dispatchTouchEvent函数，然后才将事件传递到下层的View元素中。当dispatchTouchEvent经过一个View往下传递时，如果这个View是一个ViewGroup，则会调用其onInterceptTouchEvent函数，这个函数表示是否拦截Touch事件，如果这个函数返回true，表示这个ViewGroup拦截了事件的传递，Touch事件不会再往下传递给它的子View，而是由它处理，所以会调用它的onTouchEvent函数，如果在传递过程中没有ViewGroup拦截事件，即经过的所有ViewGroup都返回false，那么事件最终会传递到最内层的View，一般是一个onTouchEvent（默认返回true），如果最后事件传递到一ViewGroup（无子View），会调用它的onTouchEvent，默认返回false

**关于Activity与View进行混合使用的情况：**

在Activity中：

@Override

**public** **boolean** dispatchTouchEvent(MotionEvent ev) {

Log.*v*("test", "activity dispatchTouchEvent");

**return** **super**.dispatchTouchEvent(ev);

}

@Override

**public** **boolean** onTouchEvent(MotionEvent event) {

Log.*v*("test", "activity touch down");

**return** **super**.onTouchEvent(event);

}

在DrawView中：

@Override

**public** **boolean** dispatchTouchEvent(MotionEvent event) {

Log.*v*("test", "mydrawView dispatchTouchEvent");

**return** **super**.dispatchTouchEvent(event);

}

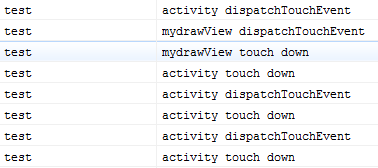
@Override

**public** **boolean** onTouchEvent(MotionEvent event) {

Log.*v*("test", "mydrawView touch down");

**return** **super**.onTouchEvent(event);

}



可以看出，在Touch事件中，首先调用的是Activity中的dispatchTouchEvent函数，在该函数中，调用窗口View的dispatchTouchEvent函数，在View的dispatchTouchEvent函数中调用onTouchEventListener与onTouchEvent函数，在上面的例子中，由于没有绑定onTouchEventListener监听器，同时onTouchEvent函数也返回false，那么接下来调用Activity的onTouchEvent函数

附上Activity的dispatchTouchEvent函数的代码：

**public** **boolean** dispatchTouchEvent(MotionEvent ev) {

**if** (ev.getAction() == MotionEvent.ACTION\_DOWN) {

onUserInteraction();

}

**if** (getWindow().superDispatchTouchEvent(ev)) {

**return** **true**;

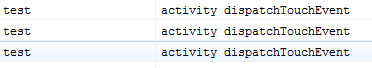
}

**return** onTouchEvent(ev);

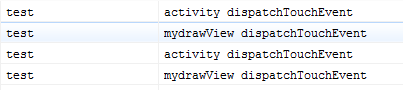
}

一点小测试（证明我们的想法）：

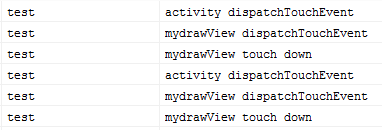
在Activity中的dispatchTouchEvent函数中返回true



在DrawView中的dispatchTouchEvent函数中返回true



在DrawView中的onTouchEvent函数中返回true



**关于Activity与ViewGroup进行混合使用的情况：**

myLinearLayout类：

@Override

**public** **boolean** dispatchTouchEvent(MotionEvent ev) {

Log.*v*("test", "myLinearLayout dispatchTouchEvent");

**return** **super**.dispatchTouchEvent(ev);

}

@Override

**public** **boolean** onInterceptTouchEvent(MotionEvent ev) {

Log.*v*("test", "myLinearLayout onInterceptTouchEvent");

**return** **super**.onInterceptTouchEvent(ev);

}

@Override

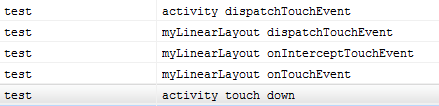
**public** **boolean** onTouchEvent(MotionEvent event) {

Log.*v*("test", "myLinearLayout onTouchEvent");

**return** **super**.onTouchEvent(event);

}

运行结果为：



在GroupView中：

**public** **boolean** onInterceptTouchEvent(MotionEvent ev) {

**return** **false**;

}

**关于Activity与View、ViewGroup进行混合使用的情况：**

新添加一个myTextView，并且该TextView位于myLinearLayout中：

@Override

**public** **boolean** dispatchTouchEvent(MotionEvent event) {

Log.*v*("test", "view int viewgroup dispatchTouchEvent");

**return** **super**.dispatchTouchEvent(event);

}

@Override

**public** **boolean** onTouchEvent(MotionEvent event) {

Log.*v*("test", "view int viewgroup touch down");

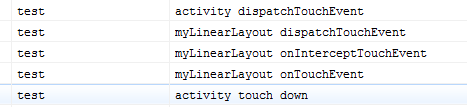
**return** **super**.onTouchEvent(event);

}

当myLinearLayout中的onInterceptTouchEvent返回false



当myLinearLayout中的onInterceptTouchEvent返回true



也就是说如果onInterceptTouchEvent返回true时，那么ViewGroup下的所有View控件都不会受到touch事件，事件只会传递到ViewGroup的onTouchEvent函数， 如果返回false，那么touch事件会先传递到该函数，然后再传递到子View的事件中去

但是对于自己写的监听器而言，尽管监听器中没有处理ACTION\_DOWN事件，仍然会处理ACTION\_MOVE与ACTION\_UP事件

注意：click事件也是在ontouch事件中进行处理的，如果我们在ontouch事件中返回true，则表示以后的消息都在onTouch函数中响应，也就是所有的MOVE、UP事件都要在OnTouchListener中进行处理，Onclick就得不到响应

**关于去掉andorid应用程序的标题与状态栏**

**代码实现：**

//去掉标题栏（应用程序的名字）

**this**.requestWindowFeature(Window.*FEATURE\_NO\_TITLE*);

//去掉隐藏状态栏（电池等图标一切修饰部分） **this**.getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*, WindowManager.LayoutParams.*FLAG\_FULLSCREEN*);/

**XML文本实现：**

//去掉标题栏（应用程序的名字）

android:theme=*"@android:style/Theme.NoTitleBar "*

//去掉标题栏与状态栏

android:theme=*"@android:style/Theme.NoTitleBar.Fullscreen"*

**关于View的invalidate()与postInvalidate()函数**

Android中实现View的更新有两种方法，一种是invalidate，另一种是postInvalidate，其中前者是在UI线程自身中使用，而后者在非UI线程中使用