

Android事件传递机制解析

android lavnFan 2016年04月21日发布

在Android开发中我们可能会遇到这样的问题，onTouch与onTouchEvent的区别是什么？onTouch与onClick又有什么区别？什么时候需要重写onTouchEvent事件进行使用呢？为什么我写的view没有不论点击还是滑动都没响应呢？等等这一系列的问题都与Android的事件分发机制有关，现在我们来慢慢剖析Android的事件分发原理。

先提两个问题：

1. 事件是怎么传递的？
2. 事件又是如何处理？

Android中与事件有关的API函数是：

```
public boolean dispatchTouchEvent(MotionEvent ev); //分发事件
public boolean onInterceptTouchEvent(MotionEvent ev); //拦截事件
public boolean onTouchEvent(MotionEvent ev); //处理事件
```

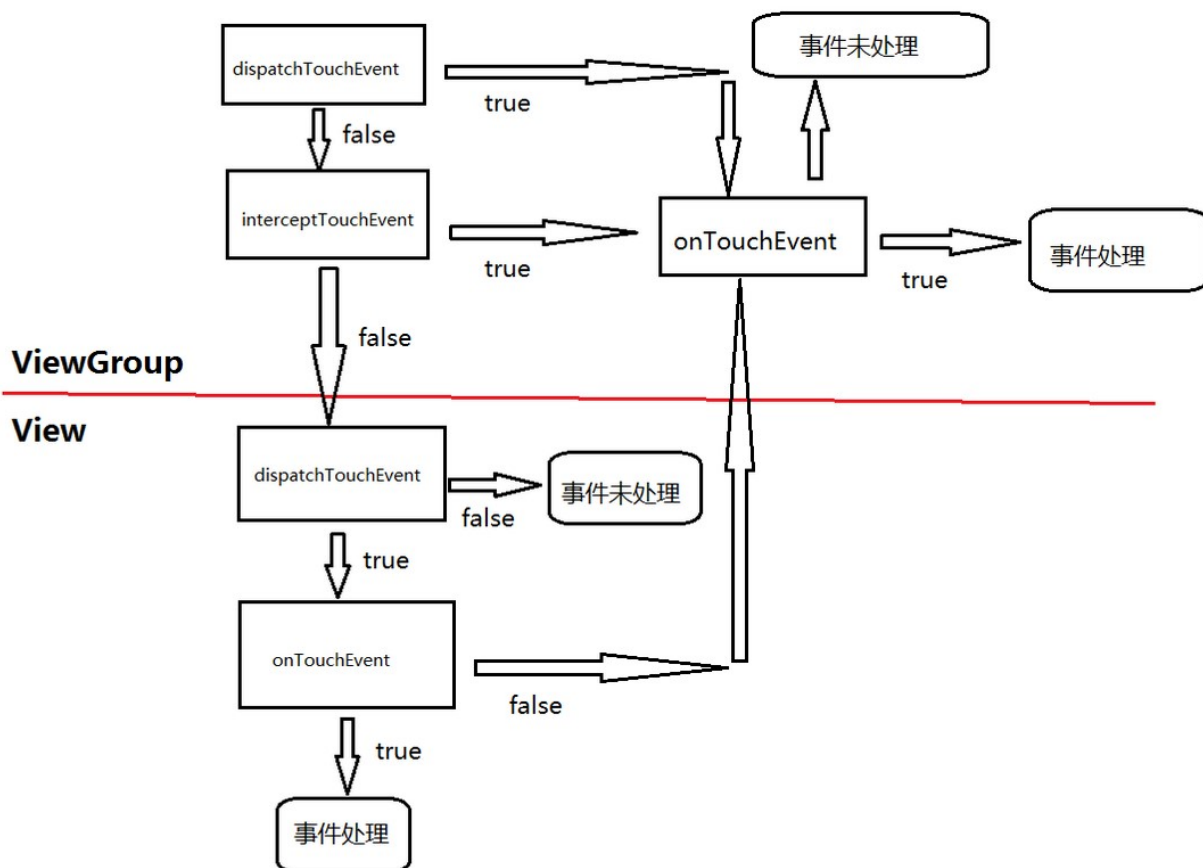
一、事件分发机制

1、事件是如何传递的：

ViewGroup接收到事件后进行事件的分派，如果自己需要处理这个事件，则进行拦截；如果不处理，则传递给子View进行处理，然后由子view进行分派，拦截和处理。可类比于：上级接到任务后进行任务分派，如果上级自己处理这个任务，则自己处理；如果不想处理，则把这个任务丢给下级进行处理...

我们需要注意：

viewGroup中包含的最小子view是不含拦截onInterceptTouchEvent事件的，最小的子view比如Button,TextView...因为他们已在树的最低层，已无法向下传递了。



上图左边部分很清楚形象的描述了事件的传递：

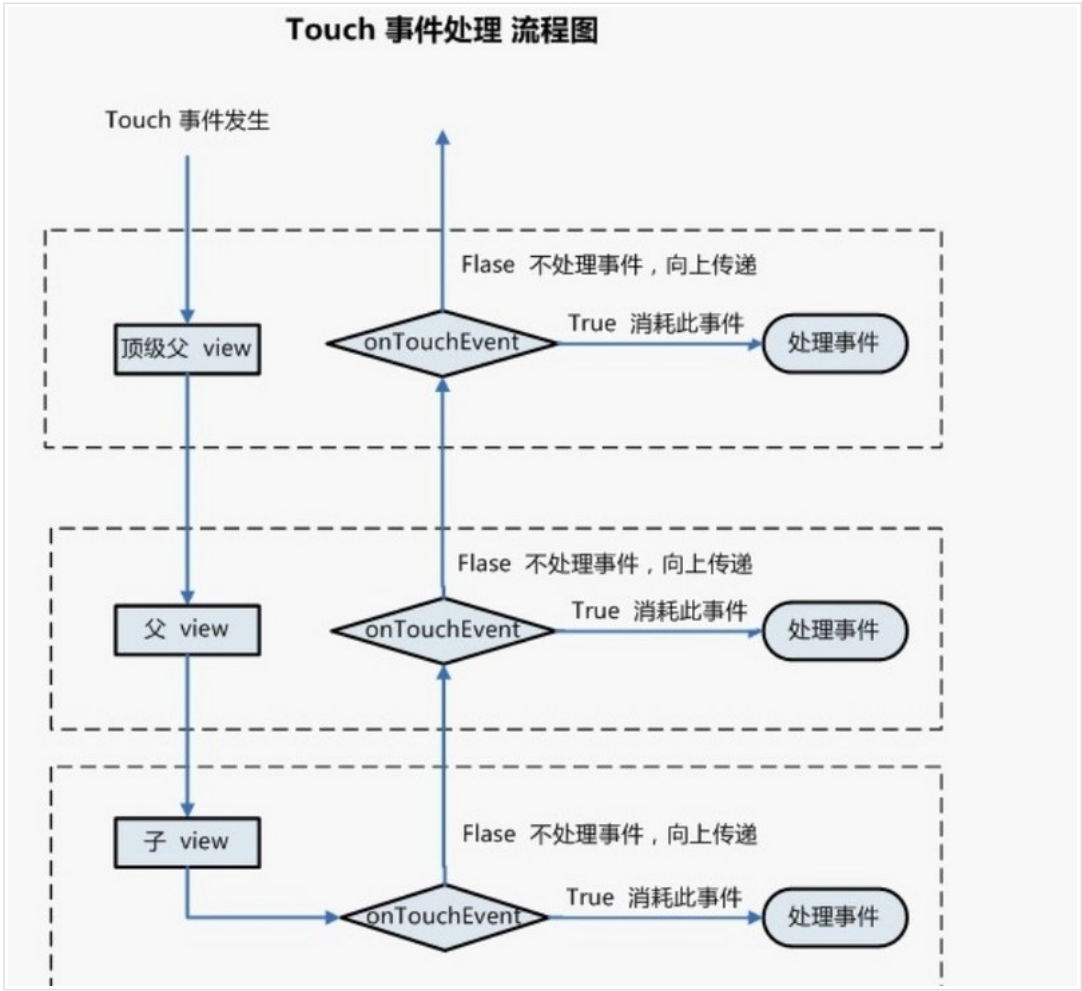
- 对于ViewGroup,接收事件后，进行分发：
 - 如果不进行分发，则dispatchTouchEvent返回true，事件消亡未处理。
 - 如果进行分发该事件，则dispatchTouchEvent返回false，处理或传递该事件：
 - 如果想自己处理该事件,则onInterceptTouchEvent返回true，拦截事件，给自己的onTouchEvent进行处理；
 - 如果不想处理该事件，则onInterceptTouchEvent返回false，把事件传递给子View进行处理。
- 对于最底层的子View，没有onInterceptEvent拦截事件，接收到事件后分发：
 - 如果不进行分发，则dispatchTouchEvent返回false，事件未处理，注意这里的最小子view返回false代表未分发事件；
 - 如果进行分发该事件，则dispatchTouchEvent返回true：
 - 如果处理该事件，则onTouchEvent返回true把该事件消费掉；
 - 如果不想处理该事件，则onTouchEvent返回false，等待上级处理。

上边所讲的是ViewGroup与最底层的子view之间的事件传递，那Android整个事件的传递机制该如何呢？

首先由Activity分发事件(和最底层的子View一样无拦截事件，因为最顶层嘛，拦截了就无法传递了)，先分发给根View,也就是DecorView（DeCorView为整个Window界面的最顶层的View,包含通知栏，标题栏，内容显示栏三块区域,我们经常写的setContentView就是为DecorView中的内容栏进行设置），然后由根View分发到子View。

2、事件是如何处理的：

假若经过分发与拦截后，最终将事件传递到了某个子View，则事件处理如下图所示：



子View的onTouchEvent进行事件处理，如果返回true，则消耗此事件，不会再继续传递；如果返回false，则不处理此事件，把这个事件往上一级的ViewGroup进行传递，由上一级进行处理。可类比如：上级把任务交给你进行处理，但由于能力不够无法处理，则把任务交给上一级进行处理，如果上一级还处理不了，则继续往上传递处理。最终事件是否处理，由子View和ViewGroup的onTouchEvent的返回值决定是否会把事件消耗掉。

二、onTouch事件

1、onTouch与onTouchEvent

想弄清楚这两个的关系，我们可以先来看一下他们在源码中的位置：

```

public boolean dispatchTouchEvent(MotionEvent event) {
    // If the event should be handled by accessibility focus first.
    if (event.isTargetAccessibilityFocus()) {
        // We don't have focus or no virtual descendant has it, do not handle the event.
        if (!isAccessibilityFocusedViewOrHost()) {
            return false;
        }
        // We have focus and got the event, then use normal event dispatch.
        event.setTargetAccessibilityFocus(false);
    }

    boolean result = false;

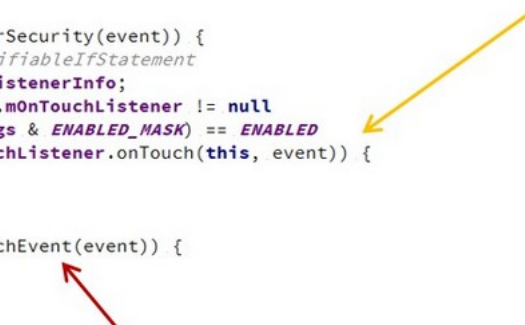
    if (mInputEventConsistencyVerifier != null) {
        mInputEventConsistencyVerifier.onTouchEvent(event, 0);
    }

    final int actionMasked = event.getActionMasked();
    if (actionMasked == MotionEvent.ACTION_DOWN) {
        // Defensive cleanup for new gesture
        stopNestedScroll();
    }

    if (onFilterTouchEventForSecurity(event)) {
        //noinspection SimplifiableIfStatement
        ListenerInfo li = mListenerInfo;
        if (li != null && li.mOnTouchListener != null
            && (mViewFlags & ENABLED_MASK) == ENABLED
            && li.mOnTouchListener.onTouch(this, event)) {
            result = true;
        }

        if (!result && onTouchEvent(event)) {
            result = true;
        }
    }
}

```



- 首先我们要清楚，dispatchTouchEvent是onTouchEvent，onTouch和onInterceptEvent的入口，他们都是在dispatchTouchEvent分发事件中决定是否要拦截和处理的；
- 其次，黄线代表onTouch事件的位置，红线代表onTouchEvent事件的位置，可以得到，**onTouch的执行顺序总是在onTouchEvent事件前的**；而且如果onTouch执行后返回true后，就会使得result为true，不会再去执行onTouchEvent事件了，只有当onTouch事件没有执行或返回为false时onTouchEvent才会得到执行。
- onTouch执行需要两个前提条件：其一是需要设置mOnTouchListener，其二是当前View必须要enable。如果控件为disable，则onTouch事件不管怎么样都不会执行；但如果需要响应触摸事件呢？这时我们不是还有onTouchEvent事件嘛，重写一下onTouchEvent方法来实现Touch的响应就行了。

2、onTouch与onClick

查看源码，我们发现：

```
public boolean onTouchEvent(MotionEvent event) {
    final float x = event.getX();
    final float y = event.getY();
    final int viewFlags = mViewFlags;
    final int action = event.getAction();

    if ((viewFlags & ENABLED_MASK) == DISABLED) {
        if (action == MotionEvent.ACTION_UP && (mPrivateFlags & PFLAG_PRESSED) != 0) {
            setPressed(false);
        }
        // A disabled view that is clickable still consumes the touch
        // events, it just doesn't respond to them.
        return (((viewFlags & CLICKABLE) == CLICKABLE
            || (viewFlags & LONG_CLICKABLE) == LONG_CLICKABLE
            || (viewFlags & CONTEXT_CLICKABLE) == CONTEXT_CLICKABLE);
    }

    if (mTouchDelegate != null) {
        if (mTouchDelegate.onTouchEvent(event)) {
            return true;
        }
    }

    if (((viewFlags & CLICKABLE) == CLICKABLE ||
        (viewFlags & LONG_CLICKABLE) == LONG_CLICKABLE) ||
        (viewFlags & CONTEXT_CLICKABLE) == CONTEXT_CLICKABLE) {
        switch (action) {
            case MotionEvent.ACTION_UP:
                boolean prepressed = (mPrivateFlags & PFLAG_PREPRESSED) != 0;
                if ((mPrivateFlags & PFLAG_PRESSED) != 0 || prepressed) {
                    // take focus if we don't have it already and we should in
                    // touch mode.
                    boolean focusTaken = false;
                    if (isFocusable() && isFocusableInTouchMode() && !isFocused()) {
                        focusTaken = requestFocus();
                    }

                    if (prepressed) {
                        // The button is being released before we actually
                        // showed it as pressed. Make it show the pressed
                        // state now (before scheduling the click) to ensure
                        // the user sees it.
                        setPressed(true, x, y);
                    }

                    if (!mHasPerformedLongPress && !mIgnoreNextUpEvent) {
                        // This is a tap, so remove the Longpress check
                        removeLongPressCallback();

                        // Only perform take click actions if we were in the pressed state
                        if (!focusTaken) {
                            // Use a Runnable and post this rather than calling
                            // performClick directly. This lets other visual state
                            // of the view update before click actions start.
                            if (mPerformClick == null) {
                                mPerformClick = new PerformClick();
                            }
                            if (!post(mPerformClick)) {
                                performClick();
                            }
                        }
                    }
                }
            }
        }
    }
}
```

onClick事件藏在onTouchEvent事件的ACTION_UP中，也就是标示的performClick，这样结合上面onTouch与onTouchEvent事件的关系，可以很容易得到：

- 执行先后不一样。触摸事件先执行（onTouch>onClick）
- 触摸事件返回值影响点击事件（前者影响后者，而后者不影响前者）；onTouch方法的返回值改为true时，只执行onTouch事件，不执行onClick事件，当然也不执行onTouchEvent事件。

参考资料：

- http://www.cnblogs.com/smyhvae/p/4802274.html?utm_source=tuicool&utm_medium=referral
- http://blog.csdn.net/guolin_blog/article/details/9097463

2016年04月21日发布 更多 ▾

1 推荐

收藏

你可能感兴趣的文章

[Android中FrameLayout无法获取OnClick Event问题](#) 2.8k 浏览

[touch事件传递流程分析](#) 9 收藏，859 浏览

[查看Android中的AlarmManager事件](#) 2 收藏，5.2k 浏览

本作品 保留所有权利 。未获得许可人许可前，不允许他人复制、发行、展览和表演作品。不允许他人基于该作品创作演绎作品。

评论 默认排序 ▾



文明社会，理性评论

发布评论



lavrFan

291 声望

关注作者

发布于专栏

个人总结

有关Android开发、数据结构与算法等等总结

5 人关注

关注专栏

系列文章

Android 动画 4 收藏, 721 浏览

相关收藏夹

换一组



珍藏工具

5 个条目 | 1 人关注



安卓开发

41 个条目 | 5 人关注



Android公用库

8 个条目 | 0 人关注

分享扩散：



Copyright © 2011-2017 SegmentFault. 当前呈现版本 17.02.05

浙ICP备 15005796号-2 浙公网安备 33010602002000号

移动版 桌面版