> 在android开发中会经常遇到滑动冲突（比如ScrollView或是SliddingMenu与ListView的嵌套）的问题，需要我们深入的了解android事件响应机制才能解决，事件响应机制已经是android开发者必不可少的知识。

1.涉及到事件响应的常用方法构成

用户在手指与屏幕接触过程中通过MotionEvent对象产生一系列事件，它有四种状态：

- MotionEvent.ACTION_DOWN ：手指按下屏幕的瞬间（一切事件的开始）
- MotionEvent.ACTION_MOVE ：手指在屏幕上移动
- MotionEvent.ACTION_UP ：手指离开屏幕瞬间
- MotionEvent.ACTION_CANCEL ：取消手势，一般由程序产生，不会由用户产生

Android中的事件onClick, onLongClick，onScroll, onFling等等，都是由许多个Touch事件构成的（一个ACTION_DOWN， n个ACTION_MOVE，1个ACTION_UP）。

android 事件响应机制是先 **分发**（先由外部的View接收，然后依次传递给其内层的最小View）再 **处理** （从最小View单元（事件源）开始依次向外层传递。）的形式实现的。

复杂性表现在：可以控制每层事件是否继续传递（分发和拦截协同实现），以及事件的具体消费（事件分发也具有事件消费能力）。

2 .android事件处理涉及到的三个重要函数

> 事件分发：**public boolean dispatchTouchEvent(MotionEvent ev)**

当有监听到事件时，首先由Activity进行捕获，进入事件分发处理流程。（因为activity没有事件拦截，View和ViewGroup有）会将事件传递给最外层View的dispatchTouchEvent(MotionEvent ev)方法，该方法对事件进行分发。

- return true ：表示该View内部消化掉了所有事件。
- return false ：事件在本层不再继续进行分发，并交由上层控件的onTouchEvent方法进行消费（如果本层控件已经是Activity，那么事件将被系统消费或处理）。
- 如果事件分发返回系统默认的 super.dispatchTouchEvent(ev)，事件将分发给本层的事件拦截onInterceptTouchEvent 方法进行处理

> 事件拦截：**public boolean onInterceptTouchEvent(MotionEvent ev)**

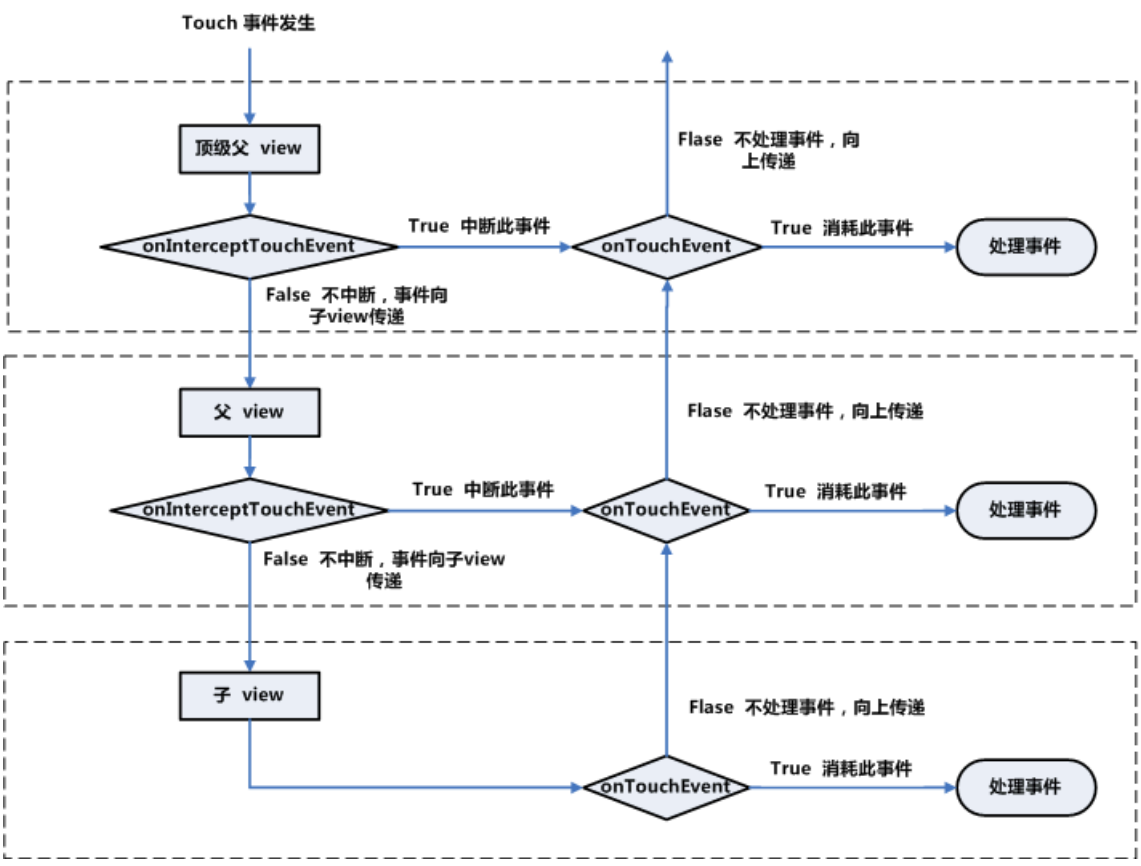
- return true ：表示将事件进行拦截，并将拦截到的事件交由本层控件 的 onTouchEvent 进行处理；
- return false ：则表示不对事件进行拦截，事件得以成功分发到子View。并由子View的dispatchTouchEvent进行处理。
- 如果返回super.onInterceptTouchEvent(ev)，默认表示拦截该事件，并将事件传递给当前View的onTouchEvent方法，和return true一样。

> 事件响应：**public boolean onTouchEvent(MotionEvent ev)**

在dispatchTouchEvent（事件分发）返回super.dispatchTouchEvent(ev)并且onInterceptTouchEvent（事件拦截返回true或super.onInterceptTouchEvent(ev)的情况下，那么事件会传递到onTouchEvent方法，该方法对事件进行响应。

- 如果return true，表示onTouchEvent处理完事件后消费了此次事件。此时事件终结；
- 如果return fasle，则表示不响应事件，那么该事件将会不断向上层View的onTouchEvent方法传递，直到某个View的onTouchEvent方法返回true，如果到了最顶层View还是返回false，那么认为该事件不消耗，则在同一个事件系列中，当前View无法再次接收到事件，该事件会交由Activity的onTouchEvent进行处理；
- 如果return super.dispatchTouchEvent(ev)，则表示不响应事件，结果与return false一样。

> 从以上过程中可以看出，dispatchTouchEvent无论返回true还是false，事件都不再进行分发，只有当其返回super.dispatchTouchEvent(ev)，才表明其具有向下层分发的愿望，但是是否能够分发成功，则需要经过事件拦截onInterceptTouchEvent的审核。事件是否向上传递处理是由onTouchEvent的返回值决定的。



（图来自网络）

3 .View源码分析

Android中ImageView、textView、Button等继承于View但没有重写的dispatchTouchEvent方法，所以都用的View的该方法进行事件分发。　　看View重要函数部分源码：

```
public boolean dispatchTouchEvent(MotionEvent event) {
//返回true,表示该View内部消化掉了所有事件。返回false，表示View内部只处理了ACTION_DOWN事件，事件继续传递，向上级View(

    if (mOnTouchListener != null && (mViewFlags & ENABLED_MASK) == ENABLED &&
            mOnTouchListener.onTouch(this, event)) {
  //此处的onTouch方式就是回调的我们注册OnTouchListener时重写的onTouch()方法
        return true;
    }
    return onTouchEvent(event);
}
```

首先进行三个条件的判断：

（1）查看是否给button设置了OnTouchListener()事件；

（2）控件是否Enable；（控件默认都是enable的）

（3）button里面实现的OnTouchListener监听里的onTouch()方法是否返回true；

如果条件都满足，则该事件被消耗掉，不再进入onTouchEvent中处理。否则将事件将交给onTouchEvent方法处理。

```
public boolean onTouchEvent(MotionEvent event) {
    ...

   /* 当前onTouch的组件必须是可点击的比如Button，ImageButton等等，此处CLICKABLE为true，才会进入if方法，最后返回tr
如果是ImageView、TexitView这些默认为不可点击的View,此处CLICKABLE为false，最后返回false。当然会有特殊情况，如果给这

    if (((viewFlags & CLICKABLE) == CLICKABLE ||
            (viewFlags & LONG_CLICKABLE) == LONG_CLICKABLE)) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_UP:
                ...
                            if (!post(mPerformClick)) {
                                performClick();// 实际就是回调了我们注册的OnClickListener中重新的onClick()方
                            }
                 ...
                break;

            case MotionEvent.ACTION_DOWN:
               ...
                break;

            case MotionEvent.ACTION_CANCEL:
                ...
                break;

            case MotionEvent.ACTION_MOVE:
               ...
                break;
        }
        return true;
    }

    return false;
}
```

```
public boolean performClick() {
    ...
  / /
    if (li != null && li.mOnClickListener != null) {
        ...
        li.mOnClickListener.onClick(this);
        return true;
    }

    return false;
}
```

```
public void setOnClickListener(OnClickListener l) {
    if (!isClickable()) {
        setClickable(true);
    }
    getListenerInfo().mOnClickListener = l;
}
```

> 只有我们注册OnTouchListener时重写的 onTouch()方法中
>
> 返回false —> 执行onTouchEvent方法 —> 导致onClick()回调方法执行
>
> 返回true —> onTouchEvent方法不执行 —> 导致onClick()回调方法不会执行

4 .ViewGroup源码分析

Android中诸如LinearLayout等的五大布局控件，都是继承自ViewGroup，而ViewGroup本身是继承自View，所以ViewGroup的事件处理机制对这些控件都有效。

部分源码：

```
public boolean dispatchTouchEvent(MotionEvent ev) {
        final int action = ev.getAction();
        final float xf = ev.getX();
        final float yf = ev.getY();
        final float scrolledXFloat = xf + mScrollX;
        final float scrolledYFloat = yf + mScrollY;
        final Rect frame = mTempRect;
```

```
final Rect frame = mTempRect;

//这个值默认是false, 然后我们可以通过requestDisallowInterceptTouchEvent(boolean disallowIntercept)方
//来改变disallowIntercept的值
boolean disallowIntercept = (mGroupFlags & FLAG_DISALLOW_INTERCEPT) != 0;

//这里是ACTION_DOWN的处理逻辑
if (action == MotionEvent.ACTION_DOWN) {
 //清除mMotionTarget, 每次ACTION_DOWN都很设置mMotionTarget为null
    if (mMotionTarget != null) {
        mMotionTarget = null;
    }

    //disallowIntercept默认是false, 就看ViewGroup的onInterceptTouchEvent()方法
    if (disallowIntercept || !onInterceptTouchEvent(ev)) {  //第一点
        ev.setAction(MotionEvent.ACTION_DOWN);
        final int scrolledXInt = (int) scrolledXFloat;
        final int scrolledYInt = (int) scrolledYFloat;
        final View[] children = mChildren;
        final int count = mChildrenCount;
        //遍历其子View
        for (int i = count - 1; i >= 0; i--) {  //第二点
            final View child = children[i];

            //如果该子View是VISIBLE或者该子View正在执行动画, 表示该View才
            //可以接受到Touch事件
            if ((child.mViewFlags & VISIBILITY_MASK) == VISIBLE
                    || child.getAnimation() != null) {
             //获取子View的位置范围
                child.getHitRect(frame);

                //如Touch到屏幕上的点在该子View上面
                if (frame.contains(scrolledXInt, scrolledYInt)) {
                    // offset the event to the view's coordinate system
                    final float xc = scrolledXFloat - child.mLeft;
                    final float yc = scrolledYFloat - child.mTop;
                    ev.setLocation(xc, yc);
                    child.mPrivateFlags &= ~CANCEL_NEXT_UP_EVENT;

                    //调用该子View的dispatchTouchEvent()方法
                    if (child.dispatchTouchEvent(ev))  {
                        // 如果child.dispatchTouchEvent(ev)返回true表示
                     //该事件被消费了, 设置mMotionTarget为该子View
                        mMotionTarget = child;
                        //直接返回true
                        return true;
                    }
                    // The event didn't get handled, try the next view.
                    // Don't reset the event's location, it's not
                    // necessary here.
                }
            }
        }
    }
}

//判断是否为ACTION_UP或者ACTION_CANCEL
boolean isUpOrCancel = (action == MotionEvent.ACTION_UP) ||
        (action == MotionEvent.ACTION_CANCEL);

if (isUpOrCancel) {
    //如果是ACTION_UP或者ACTION_CANCEL, 将disallowIntercept设置为默认的false
 //假如我们调用了requestDisallowInterceptTouchEvent()方法来设置disallowIntercept为true
 //当我们抬起手指或者取消Touch事件的时候要将disallowIntercept重置为false
 //所以说上面的disallowIntercept默认在我们每次ACTION_DOWN的时候都是false
    mGroupFlags &= ~FLAG_DISALLOW_INTERCEPT;
}

// The event wasn't an ACTION_DOWN, dispatch it to our target if
// we have one.
final View target = mMotionTarget;
//mMotionTarget为null意味着没有找到消费Touch事件的View, 所以我们需要调用ViewGroup父类的
//dispatchTouchEvent()方法, 也就是View的dispatchTouchEvent()方法
if (target == null) {
    // We don't have a target, this means we're handling the
    // event as a regular view.
    ev.setLocation(xf, yf);
    if ((mPrivateFlags & CANCEL_NEXT_UP_EVENT) != 0) {
        ev.setAction(MotionEvent.ACTION_CANCEL);
        mPrivateFlags &= ~CANCEL_NEXT_UP_EVENT;
    }
    return super.dispatchTouchEvent(ev);
}
```

```
        }

        //这个if里面的代码ACTION_DOWN不会执行，只有ACTION_MOVE
        //ACTION_UP才会走到这里，假如在ACTION_MOVE或者ACTION_UP拦截的
        //Touch事件，将ACTION_CANCEL派发给target，然后直接返回true
        //表示消费了此Touch事件
        if (!disallowIntercept && onInterceptTouchEvent(ev)) {
            final float xc = scrolledXFloat - (float) target.mLeft;
            final float yc = scrolledYFloat - (float) target.mTop;
            mPrivateFlags &= ~CANCEL_NEXT_UP_EVENT;
            ev.setAction(MotionEvent.ACTION_CANCEL);
            ev.setLocation(xc, yc);

            if (!target.dispatchTouchEvent(ev)) {
            }
            // clear the target
            mMotionTarget = null;
            // Don't dispatch this event to our own view, because we already
            // saw it when intercepting; we just want to give the following
            // event to the normal onTouchEvent().
            return true;
        }

        if (isUpOrCancel) {
            mMotionTarget = null;
        }

        // finally offset the event to the target's coordinate system and
        // dispatch the event.
        final float xc = scrolledXFloat - (float) target.mLeft;
        final float yc = scrolledYFloat - (float) target.mTop;
        ev.setLocation(xc, yc);

        if ((target.mPrivateFlags & CANCEL_NEXT_UP_EVENT) != 0) {
            ev.setAction(MotionEvent.ACTION_CANCEL);
            target.mPrivateFlags &= ~CANCEL_NEXT_UP_EVENT;
            mMotionTarget = null;
        }

        //如果没有拦截ACTION_MOVE，ACTION_DOWN的话，直接将Touch事件派发给target
        return target.dispatchTouchEvent(ev);
    }
```

> 1、dispatchTouchEvent作用：决定事件是否由onInterceptTouchEvent来拦截处理。 返回super.dispatchTouchEvent时，由onInterceptTouchEvent来决定事件的流向 返回false时，会继续分发事件，自己内部只处理了ACTION_DOWN 返回true时，不会继续分发事件，自己内部处理了所有事件（ACTION_DOWN,ACTION_MOVE,ACTION_UP）
>
> 2、onInterceptTouchEvent作用：拦截事件，用来决定事件是否传向子View 返回true时，拦截后交给自己的onTouchEvent处理 返回false时，拦截后交给子View来处理
>
> 3、onTouchEvent作用：事件最终到达这个方法 返回true时，内部处理所有的事件，换句话说，后续事件将继续传递给该view的onTouchEvent()处理 返回false时，事件会向上传递，由onTouchEvent来接受，如果最上面View中的onTouchEvent也返回false的话，那么事件就会消失

5 .总结

- 如果ViewGroup找到了能够处理该事件的View，则直接交给子View处理，自己的onTouchEvent不会被触发；
- 可以通过复写onInterceptTouchEvent(ev)方法，拦截子View的事件（即return true），把事件交给自己处理，则会执行自己对应的onTouchEvent方法。
- 子View可以通过调用getParent().requestDisallowInterceptTouchEvent(true); 阻止ViewGroup对其MOVE或者UP事件进行拦截；
- 一个点击事件产生后，它的传递过程如下： Activity->Window->View。顶级View接收到事件之后，就会按相应规则去分发事件。如果一个View的onTouchEvent方法返回false，那么将会交给父容器的onTouchEvent方法进行处理，逐级往上，如果所有的View都不处理该事件，则交由Activity的onTouchEvent进行处理。
- 如果某一个View开始处理事件，如果他不消耗ACTION_DOWN事件（也就是onTouchEvent返回false），则同一事件序列比如接下来进行ACTION_MOVE，则不会再交给该View处理。
- ViewGroup默认不拦截任何事件。

- 诸如TextView、ImageView这些不作为容器的View，一旦接受到事件，就调用onTouchEvent方法，它们本身没有onInterceptTouchEvent方法。正常情况下，它们都会消耗事件（返回true），除非它们是不可点击的（clickable和longClickable都为false），那么就会交由父容器的onTouchEvent处理。
- 点击事件分发过程如下 dispatchTouchEvent—->OnTouchListener的onTouch方法—->onTouchEvent-->OnClickListener的onClick方法。也就是说，我们平时调用的setOnClickListener，优先级是最低的，所以，onTouchEvent或OnTouchListener的onTouch方法如果返回true，则不响应onClick方法...