Android 中的事件分发和处理

原创 2016-07-24 伯乐专栏/邵辉 安卓应用频道

> 来源：伯乐在线专栏作者 - 邵辉|CRR
> 链接：http://android.jobbole.com/83826/
> 点击 → 了解如何加入专栏作者

上次跟大家分享了一下自定义View的一下要点，这次跟大家聊一下View的事件分发及处理，为什么主题都是View，因为作为一名初级应用层Android工程师，跟我打交道最多的莫过于各种各样的View，只有详细了解他们各自的习性，才能更好地跟他们沟通交流，做出自己想要的效果。

基础储备 View、MotionEvent

我们都能详细地说出Android的四大组件：Activity，Service，ContentProvider和BoardcastReceiver，但是四大组件之外，我们用到也很多的是什么，其中肯定包括View，View是用户跟程序沟通的入口，也是程序展现给用户信息的窗口。关于View，一些基础属性还是要了解的，left，top，right，bottom，分别代表了view的左上角和右下角分别相对x轴，y轴的坐标，而且view的getWidth和getHeight的值都是通过这四个值算得，而且在Android3.0中还增加了x，y，translationX和translationY这几个属性，便于我们对view的平移操作，x、y代表了当前view左上角的xy坐标，而translationX和translationY代表了view相对它的父容器的偏移量，默认值是0。

MotionEvent表示用户的触摸事件，用户的一次点击、触摸或者滑动都会产生一系列的MotionEvent：

- MotionEvent.ACTION_DOWN 表示用户的手指刚接触到屏幕
- MotionEvent.ACTION_MOVE 表示用户的手指正在移动
- MotionEvent.ACTION_UP 表示用户的手指从屏幕上抬起

所以一次用户触摸屏幕可能会产生这些事件:

- 点击屏幕然后松开，Down->Up
- 点击屏幕，然后滑动一段距离，松开屏幕 ，Down->Move->...->Move->Up
  了解了这些基本知识以后，我们就来学习一下具体怎么分发这些事件

ViewGroup 分发-> 拦截 -> 处理

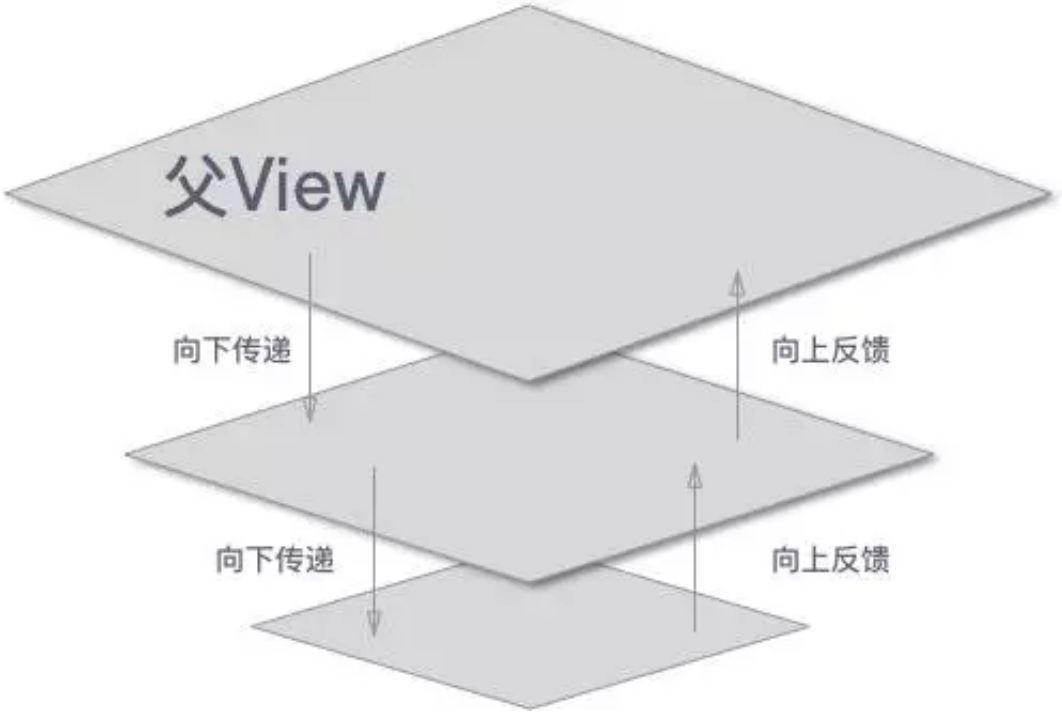
首先说一点，虽然ViewGroup也是继承View而来，但是因为在事件拦截上，ViewGroup分析起来更加方便理解，所以先说ViewGroup，下面也会简单介绍一下View的事件处理。

在事件分发的过程中，主要涉及到三个方法:

- dispatchTouchEvent(MotionEvent event);
- onInterceptTouchEvent(MotionEvent event);
- onTouchEvent();

初看这三个方法就有蒙圈，如果这时候在蒙头钻进源码里，就更是糊涂，我在这里借用任玉刚大大的一段伪代码解释一下这三者之间的关系：

```
public boolean dispatchTouchEvent(MotionEvent event) {
    boolean consume = false;
    if (onInterceptTouchEvent(event)) {
        consume = onTouchEvent(event);
    } else {
        consume = child.dispatchTouchEvent(event);
    }

    return consume;
}
```



- 从这段伪代码中，我们可以看出来，在dispatchTouchEvent中，先调用ViewGroup自身的onInterceptTouchEvent方法，判断自己是否要拦截，如果这时候自己拦截，那就调用自己的onTouchEvent方法，如果onTouchEvent方法返回了True，那么这次的事件就算消耗了，事件传递到此为止，如果返回了False，证明这次没有消耗这次MotionEvent，那么这次的事件就会往上返回，由上一级继续处理；如果当前ViewGroup的onInterceptTouchEvent返回了False，那就会调用它的子view的dispatchTouchEvent方法，这样这个事件就传递下去了，如果它的子View处理不了，那么还会回来调用ViewGroup的onTouchEvent方法，当然这一点是没有在这一段伪代码里体现的，用一段通俗的例子解释：

> 领导收到一份任务(有可能是上级给的)，自己看了一眼，然后决定好好休息，今天不工作，就把这个任务交给了手下的小王，小王的默认属性是只要来任务就接，而且就干，能干不能干一样接，如果这是一个简单的任务，那么小王就解决了，这个任务也就完成了，不幸的是，这次任务小王没有解决掉，然后向领导反馈，领导没办法，手下没人能解决，只好自己干了，就开始解决问题，然后解决掉，任务也完成了。

这就是ViewGroup层的事件分发，当然不是这么简单，这只不过是通过简单的方式去理解，其实在真实的事件分发中，有很多问题需要注意：

- 一个完成的事件序列以Down开始，中间可能包含若干个Move，然后以Up结束
- 一个view一旦拦截一个某个事件，当前事件所在的完整事件序列将都会由这个view去处理，反应在真实的代码中，就是一旦view拦截了down事件，那么此后的move和up事件都将不调用onInterceptTouchEvent，而直接由它处理，这就也意味着在

onInterceptTouchEvent处理事件是不合适的，因为有可能来了事件，却直接跳过onInterceptTouchEvent方法。这个也意味着，一旦一个ViewGroup没有拦截ACTION_DOWN，那么这个事件序列的其他Action，它都将收不到，所以在处理ACTION_DOWN的时候，尤其需要谨慎。

- onTouchEvent中是要判断MotionEvent的Action，因为一次点击操作就会调用两次onTouchEvent方法，一次是ACTION_DOWN，一次是ACTION_UP，如果手滑一下，还会有若干个ACTION_MOVE
- ViewGroup默认不拦截任何事件，源码中ViewGroup的onInterceptTouchEvent方法默认返回的是false

> 整个事件分发，看起来都是由外向内传递的，父View将事件传递给子View，理论上来看，子View是没有办法影响到父View的事件处理的，但是有一个标示位，requestDisallowInterceptTouchEvent方法，通过这个方法 ，子View能够影响父view的事件处理，这个可以用于解决父view和子view的滑动冲突，具体想了解的可以搜索它的相关用法，这里将不进行展开。

View 只有默默的承受

View不同于ViewGroup的是，View中没有onInterceptTouchEvent方法，因为View作为事件处理的最后一级，不需要判断是否要拦截，是一定要拦截，不管能不能处理，都要试一下，所以在View中调用流程是：

> dispatchTouchEvent -> onTouchEvent

而且，最后onTouchEvent的返回值默认都是True，也就是说事件传递下去一般都会被消耗掉的，只是看中途是否有人拦截，这个时候读者可能会有疑问：TextView的onTouchEvent的返回值也是True吗？答案就是:是的，那为什么点在TextView上面还是能触发它的父视图的onTouchEvent，理论上不应该是，TextView消耗掉这次的事件，不回传。理论上确实是这样，但是因为TextView的clickable和longClickable属性都是false，当这两个属性都为false的时候，是不会消耗事件的，所以TextView不会消耗事件，这也就可以解释为什么把一个TextView放在一个Button上面，然后点击TextView还是能触发Button的点击事件

> 在这里可能需要提醒一下大家，算是一个我之前踩到的一个坑，我把一个view的enable状态设成了false，然后又给它增加了onClickListener，这时候我本以为，它的点击事件不会被触发，结果它还是可以被点击，后来才了解到，view的enable状态和onTouchEvent是没有关系的，只有clickable状态是对onTouchEvent有影响的，还有一点 ，设置 view的enable为false确实也会把view的clickable设成false，但是设置view的onclickListener就又把view的clickable变成了true，所以最后的解决方案就是把那两行代码换下先后顺序，问题就迎刃而解了。

详解处理GesutureDetector

费劲千辛万苦，终于把事件拦截下来了，然后我们需要总得做点什么吧，不然都对不起自己浪费这么多口舌，说到对事件的处理，我们首先想到的就是setOnClickListener，殊不知onClickListener的优先级是最低的，下一节里面会对优先级进行说明，而这里，我们将主要想着如果处理事件，当我们兴奋地拿到一连串的事件，但又不知如何下手，甚至于连最简单的点击事件都要自己进行一番处理，更别提做成平移、旋转、缩放这样的操作，但是官方提供的GestureDetector给我们提供了可能。

官方提供的GestureDetector是一个手势辅助检测类，默认能够检测多种手势：

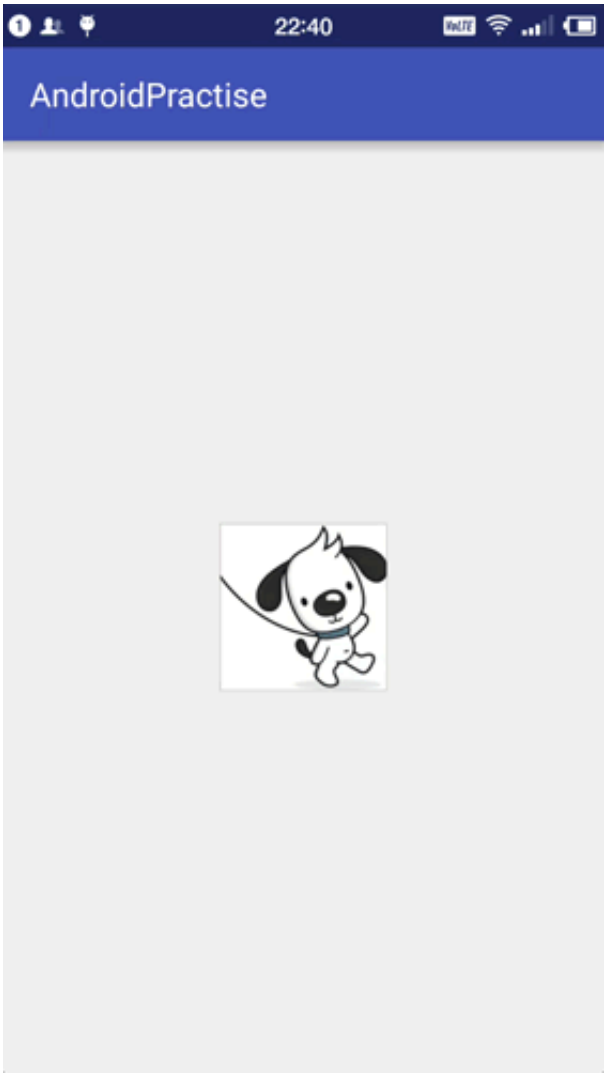
```
class SimpleGestureListener implements GestureDetector.OnGestureListener {
    @Override
    public boolean onDown(MotionEvent e) {
        return false;
    }

    @Override
    public void onShowPress(MotionEvent e) {

    }

    @Override
    public boolean onSingleTapUp(MotionEvent e) {
        return false;
    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {
        return false;
    }

    @Override
    public void onLongPress(MotionEvent e) {

    }

    @Override
```

```
    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
        return false;
    }
}
```

通过这个类，我们可以很方便的处理除了单击和长按之外，还有滑动，双击等各种手势，并对其分别进行处理，如果这些还是不能满足你的好奇心，那还有一个官方提供的ScaleGestureDetector，从名字就可以判断出来这是一个检测缩放手势的辅助类，而且还有大牛仿照ScaleGestureDetector思路做出了平移以及旋转的辅助类，然后我们就可以根据这些辅助类，几乎为所欲为了，下面我写了一个支持平移，缩放，旋转的小Demo。

```
private void init() {

    scaleGesture = new ScaleGestureDetector(getContext(), new ScaleListener());
    moveGesture = new MoveGestureDetector(getContext(), new MovingListener());
    rotateGesture = new RotateGestureDetector(getContext(), new RotateListener());

}

@Override
public boolean onTouchEvent(MotionEvent event) {

    scaleGesture.onTouchEvent(event);
    moveGesture.onTouchEvent(event);
    rotateGesture.onTouchEvent(event);

    return true;
}


private class ScaleListener implements ScaleGestureDetector.OnScaleGestureListener {

    @Override
    public boolean onScale(ScaleGestureDetector detector) {

        setScaleX(detector.getScaleFactor() * getScaleX());
        setScaleY(detector.getScaleFactor() * getScaleY());

        return true;
```

```
    }

    @Override
    public boolean onScaleBegin(ScaleGestureDetector detector) {
        return true;
    }

    @Override
    public void onScaleEnd(ScaleGestureDetector detector) {

    }
}
```



只贴了部分的代码，而且貌似旋转好像还有点问题，以后时间再修正，有用到的读者可以在详细了解下，完整代码，我会在文章的最后给出链接，同时感谢Android multitouch gesture detectors的作者，提供了这么方便的手势操作类

onTouchListener OnTouchEvent OnClickListener

我们在知道onTouchEvent之前肯定都知道onClickListener和onTouchListener，而他们都是事件的消费者，onTouchListener是在onTouch方法中生效，而且onTouch要先于onTouchEvent，就是说一旦设置了onTouchListener并且最后onTouch方法返回了True，那onTouchEvent将不会再被执行，而onClickListener和onTouchEvent有些关系，onTouchEvent的默认实现里会调用onClickListener的onClick方法，如果重写了onTouchEvent，因为onClickListener接受不到ACTION_DOWN和ACTION_UP，那么再设置onClickListener也就不会再生效了，这个时候的单击或者长按处理只能在onTouchEvent中自己处理。

The End

聊了这么多，主要是想对View的拦截机制了解的更多一点，因为最近在拜读任玉刚大大的《Android开发艺术探究》，文中也引用了很多书中的观点，有想了解更多的小伙伴可以去买一本读一下，受益匪浅。

阅读原文