

# Android View事件机制 21问21答

2016-04-11 安卓应用频道

(点击上方公众号，可快速关注)

来源：希尔瓦娜斯女神

链接：<http://www.cnblogs.com/punkisnotdead/p/5179115.html>

## Android View事件机制 21问21答

### 1.View的坐标参数 主要有哪些？分别有什么注意的要点？

答：Left，Right，top, Bottom 注意这4个值其实就是 view 和他的父控件的 相对坐标值。并非距离屏幕左上角的绝对值，这点要注意。

此外，X和Y 其实也是相对于父控件的坐标值。 TranslationX, TranslationY 这2个值 默认都为0，是相对于父控件的左上角的偏移量。

换算关系：

$x = \text{left} + \text{tranX}, y = \text{top} + \text{tranY}.$

很多人不理解，为什么事这样，其实就是View 如果有移动的话，比如平移这种，你们就要注意了，top和left 这种值 是不会变化的。

无论你把view怎么拖动，但是 x,y, tranX, tranY 的值是随着拖动平移 而变化的。想明白这点 就行了。

### 2.onTouchEvent和GestureDetector 在什么时候用哪个比较好？

答：只有滑动需求的时候 就用前者，如果有双击等这种行为的时候 就用后者。

### 3.Scroller 用来解决什么问题？

答：view的scrollTo和scrollBy 滑动效果太差了，是瞬间完成。而scroller可以配合view的computeScroll 来完成 渐变的滑动效果。体验更好。

#### 4.ScrollTo和ScrollBy 有什么需要注意的？

答：前者是绝对滑动，后者是相对滑动。滑动的是view的内容 而不是view本身。这很重要。比如textView 调用这2个方法 滑动的就是显示出来的字的内容。

一般而言 我们用scrollBy会比较多一些。传值的话 其实 记住几个法则就可以了。右-左 x为正 否则x为负 上-下 y为负，否则y为正。

可以稍微看一下 这2个的源码：

```
public void scrollTo(int x, int y) {
    if (mScrollX != x || mScrollY != y) {
        int oldX = mScrollX;
        int oldY = mScrollY;
        mScrollX = x;
        mScrollY = y;
        invalidateParentCaches();
        onScrollChanged(mScrollX, mScrollY, oldX, oldY);
        if (!awakenScrollBars()) {
            postInvalidateOnAnimation();
        }
    }
}

public void scrollBy(int x, int y) {
    scrollTo(mScrollX + x, mScrollY + y);
}
```

看到里面有2个变量 mScrollX 和mScrollY 这2个东西没，这2个单位的 值是像素，前者代表view的左边缘和view内容左边缘的距离。后者代表 view上边缘和view内容上边缘的距离。

#### 5.使用动画来实现view的滑动 有什么后果？

答：实际上view动画 是对view的表面ui 也就是给用户呈现出的视觉效果 来做的移动，动画本身并不能移动view的真正位置。属性动画除外。动画播放结束以后，view最终还是会回到自己的位置的，。当然了你可以设置fillafter 属性 来让动画播放结束以后 view表象停留在变化以后的位置。所以这会带来一个很严重的后果。比如你的button在屏幕的左边，你现在用个动画 并且设置了fillafter属性让他去了右边。你会发现 点击右边的button 没有click事件触发，但是点击左边的 却可以触发，原因就是右边的button 只是view的表象，真正的button

还在左边没有动过。你一定要这么做的话 可以提前在右边button移动后的位置放一个新的button，当你动画执行结束以后 把右边的enable 左边的让他gone就可以了。

这么做就可以规避上述问题。

## 6.让view滑动总共有几种方式，分别要注意什么？都适用于那些场景？

答：总共有三种：

a：scrollto，scrollby。这种是最简单的，但是只能滑动view的内容 不可以滑动view本身。

b：动画。动画可以滑动view内容，但是注意非属性动画 就如我们问题5说的内容 会影响到交互，使用的时候要多注意。不过多数复杂的滑动效果都是属性动画来完成的，属于大杀器级别、

c：改变布局参数。这种最好理解了，无非是动态的通过java代码来修改 margin等view的参数罢了。不过用的比较少。我本人不怎么用这种方法。

## 7.Scroller是干嘛的？原理是什么？

答：Scroller就是用于 让view有滑动渐变效果的。用法如下：

```
package com.example.administrator.motioneventtest;

import android.content.Context;
import android.util.AttributeSet;
import android.widget.Scroller;
import android.widget.TextView;

/**
 * Created by Administrator on 2016/2/2.
 */
public class CustomTextView extends TextView{

    private Scroller mScroller;

    public CustomTextView(Context context) {
        super(context);
```

```

        mScroller=new Scroller(context);
    }

```

```

public CustomTextView(Context context, AttributeSet attrs) {
    super(context, attrs);
    mScroller=new Scroller(context);

}

```

```

public CustomTextView(Context context, AttributeSet attrs, int defStyleAttr) {
    super(context, attrs, defStyleAttr);
    mScroller=new Scroller(context);

}

```

//调用此方法滚动到目标位置

```

public void smoothScrollTo(int fx, int fy) {
    int dx = fx - mScroller.getFinalX();
    int dy = fy - mScroller.getFinalY();
    smoothScrollBy(dx, dy);
}

```

//调用此方法设置滚动的相对偏移

```

public void smoothScrollBy(int dx, int dy) {

```

//设置mScroller的滚动偏移量

```

    mScroller.startScroll(mScroller.getFinalX(), mScroller.getFinalY(), dx, dy,4000);

```

```

    invalidate();//这里必须调用invalidate()才能保证computeScroll()会被调用，否则不一定会刷新界面，看不到滚动效果
}

```

//使用scroller最重要不要遗漏这个方法

@Override

```

public void computeScroll() {
    if (mScroller.computeScrollOffset())
    {
        scrollTo(mScroller.getCurrX(),mScroller.getCurrY());
        //这个方法不要忘记调用。
        postInvalidate();
    }
}

```

```

    }
    super.computeScroll();
}
}

```

其实上述代码 很多人应该都能搜到。我们这里主要讲一下 他的原理。

```

//参数很好理解 前面滑动起始点 中间滑动距离 最后一个是 渐变时间
//而且我们看到startScroll 这个方法就是设置了一下参数 并没有什么滑动的代码在
//回到前面的demo能看到我们通常调用完这个方法以后 都会马上调用invalidate()方法

```

```

public void startScroll(int startX, int startY, int dx, int dy, int duration) {
    mMode = SCROLL_MODE;
    mFinished = false;
    mDuration = duration;
    mStartTime = AnimationUtils.currentAnimationTimeMillis();
    mStartX = startX;
    mStartY = startY;
    mFinalX = startX + dx;
    mFinalY = startY + dy;
    mDeltaX = dx;
    mDeltaY = dy;
    mDurationReciprocal = 1.0f / (float) mDuration;
}

```

```

//我们都知道invalidate 会触发view的 draw方法
//我们跟进去看 会发现draw方法里 会调用下面的代码：
//也就是说会调用 computeScroll方法 而view本身这个方法
//是空的所以会留给我们自己实现

```

```

int sx = 0;
int sy = 0;
if (!drawingWithRenderNode) {
    computeScroll();
    sx = mScrollX;
    sy = mScrollY;
}

```

```

//然后回到我们的customtextview 可以看到我们实现的 computeScroll方法如下：

```

//你看在这个方法里 我们调用了scrollTo方法 来实现滑动，滑动结束以后再次触发view的重绘

//然后又会再次触发computeScroll 实现一个循环。

```
public void computeScroll() {
    if (mScroller.computeScrollOffset())
    {
        scrollTo(mScroller.getCurrX(),mScroller.getCurrY());

        //这个方法不要忘记调用。
        postInvalidate();
    }
    super.computeScroll();
}
```

//返回true就代表滑动还没结束 false就是结束了

//其实这个方法 就跟属性动画里的插值器一样 你在使用startScroll方法的时候 会传一个事件的值，

//这个方法就是根据这个事件的值来计算你每一次scrollx和scrolly的值

```
public boolean computeScrollOffset() {
    if (mFinished) {
        return false;
    }

    int timePassed = (int)(AnimationUtils.currentAnimationTimeMillis() - mStartTime);

    if (timePassed < mDuration) {
        switch (mMode) {
            case SCROLL_MODE:
                final float x = mInterpolator.getInterpolation(timePassed * mDurationReciprocal);
                mCurrX = mStartX + Math.round(x * mDeltaX);
                mCurrY = mStartY + Math.round(x * mDeltaY);
                break;
            case FLING_MODE:
                final float t = (float) timePassed / mDuration;
                final int index = (int) (NB_SAMPLES * t);
                float distanceCoef = 1.f;
                float velocityCoef = 0.f;
                if (index < NB_SAMPLES) {
                    final float t_inf = (float) index / NB_SAMPLES;
                    final float t_sup = (float) (index + 1) / NB_SAMPLES;
                    final float d_inf = SPLINE_POSITION[index];
```

```

        final float d_sup = SPLINE_POSITION[index + 1];
        velocityCoef = (d_sup - d_inf) / (t_sup - t_inf);
        distanceCoef = d_inf + (t - t_inf) * velocityCoef;
    }

    mCurrVelocity = velocityCoef * mDistance / mDuration * 1000.0f;

    mCurrX = mStartX + Math.round(distanceCoef * (mFinalX - mStartX));
    // Pin to mMinX <= mCurrX <= mMaxX
    mCurrX = Math.min(mCurrX, mMaxX);
    mCurrX = Math.max(mCurrX, mMinX);

    mCurrY = mStartY + Math.round(distanceCoef * (mFinalY - mStartY));
    // Pin to mMinY <= mCurrY <= mMaxY
    mCurrY = Math.min(mCurrY, mMaxY);
    mCurrY = Math.max(mCurrY, mMinY);

    if (mCurrX == mFinalX && mCurrY == mFinalY) {
        mFinished = true;
    }

    break;
}
}
else {
    mCurrX = mFinalX;
    mCurrY = mFinalY;
    mFinished = true;
}
return true;
}

```

## 8.view的滑动渐变效果总共有几种方法？

答：三种，第一种是scroller 也是使用最多的。问题7里有解释。还有一种就是动画，动画我就不多说了，不属于本文范畴。最后一种也是我们经常使用的就是用handler，每隔一个时间间隔来更新view的状态。

代码不写了很简单。自行体会。

## 9.view的事件传递机制 如何用伪代码来表示？

答：

```
/**
 * 对于一个root viewgroup来说，如果接受了一个点击事件，那么首先会调用他的dispatchTouchEvent方法。
 * 如果这个viewgroup的onInterceptTouchEvent 返回true，那就代表要拦截这个事件。接下来这个事件就
 * 给viewgroup自己处理了，从而viewgroup的onTouchEvent方法就会被调用。如果如果这个viewgroup的
onInterceptTouchEvent
 * 返回false就代表我不拦截这个事件，然后就把这个事件传递给自己的子元素，然后子元素的dispatchTouchEvent
 * 就会被调用，就是这样一个循环直到 事件被处理。
 *
 */
public boolean dispatchTouchEvent(MotionEvent ev)
{
    boolean consume=false;
    if (onInterceptTouchEvent(ev)) {
        consume=onTouchEvent(ev);
    }else
    {
        consume=child.dispatchTouchEvent(ev);
    }
    return consume;
}
```

## 10.view的onTouchEvent，OnClickListener和OnTouchListener的onTouch方法 三者优先级如何？

答：onTouchListener优先级最高，如果onTouch方法返回 false，那onTouchEvent就被调用了，返回true 就不会被调用。至于onClick 优先级最低。

## 11.点击事件的传递顺序如何？

答：Activity-Window-View。从上到下依次传递，当然了如果你最低的那个view onTouchEvent返回false 那就说明他不想处理 那就再往上抛，都不处理的话 最终就还是让Activity自己处理了。举个例子，pm下发一个任务给leader，leader自己不做 给架构师a，小a也不做 给程序员b，b如果做了那就结束了这个任务。

b如果发现自己搞不定，那就找a做，a要是也搞不定 就会不断向上发起请求，最终可能还是

pm做。

//activity的dispatchTouchEvent 方法 一开始就是交给window去处理的  
 //win的superDispatchTouchEvent 返回true 那就直接结束了 这个函数了。返回false就意味  
 //这事件没人处理，最终还是给activity的onTouchEvent 自己处理 这里的getWindow 其实就是phonewindow

```
public boolean dispatchTouchEvent(MotionEvent ev) {
    if (ev.getAction() == MotionEvent.ACTION_DOWN) {
        onUserInteraction();
    }
    if (getWindow().superDispatchTouchEvent(ev)) {
        return true;
    }
    return onTouchEvent(ev);
}
```

//来看phonewindow的这个函数 直接把事件传递给了mDecor

```
@Override
public boolean superDispatchTouchEvent(MotionEvent event) {
    return mDecor.superDispatchTouchEvent(event);
}
```

//devorview就是 我们的rootview了 就是那个framelayout 我们的setContentView里面传递的那个layout  
 //就是这个decorview的 子view了

```
@Override
public final View getDecorView() {
    if (mDecor == null) {
        installDecor();
    }
    return mDecor;
}
```

## 12.事件分为几个步骤？

答：down事件开头，up事件结尾，中间可能会有数目不定的move事件。

## 13.ViewGroup如何对点击事件分发？

答：

viewgroup就是在actionMasked == MotionEvent.ACTION\_DOWN 和 mFirstTouchTarget != null 这两种情况来判断是否会进入拦截事件的流程

看代码可以知道 如果是ACTION\_DOWN事件 那就肯定进入 是否要拦截事件的流程

如果不是ACTION\_DOWN事件 那就要看mFirstTouchTarget != null 这个条件是否成立

这个地方有点绕但是也好理解，其实就是 对于一个事件序列来说 down是事件的开头 所以肯定进入了这个事件是否拦截的流程 也就是if 括号内。

mFirstTouchTarget其实是一个单链表结构他指向的是 成功处理事件的子元素。

也就是说 如果有子元素成功处理了 事件，那这个值就不为NULL。反过来说

只要viewgroup拦截了事件，mFirstTouchTarget就不为NULL，所以括号内就不会执行，也就侧面说明了一个结论：

某个view 一旦决定拦截事件，那么这个事件所属的事件序列 都只能由他来执行。并且onInterceptTouchEvent 这个方法不会被调用了

```
final boolean intercepted;
if (actionMasked == MotionEvent.ACTION_DOWN
    || mFirstTouchTarget != null) {
    final boolean disallowIntercept = (mGroupFlags & FLAG_DISALLOW_INTERCEPT) != 0;
    if (!disallowIntercept) {
        intercepted = onInterceptTouchEvent(ev);
        ev.setAction(action); // restore action in case it was changed
    } else {
        intercepted = false;
    }
} else {
    // There are no touch targets and this action is not an initial down
    // so this view group continues to intercept touches.
    intercepted = true;
}
```

#### 14.如果某个view 处理事件的时候 没有消耗down事件 会有什么结果？

答：假如一个view，在down事件来的时候 他的onTouchEvent返回false，那么这个down事件 所属的事件序列 就是他后续的move 和up 都不会给他处理了，全部都给他的父view处理。

### 15.如果view 不消耗move或者up事件 会有什么结果？

答：那这个事件所属的事件序列就消失了，父view也不会处理的，最终都给activity 去处理了。

### 16.ViewGroup 默认拦截事件吗？

答：默认不拦截任何事件，onInterceptTouchEvent返回的是false。

### 17.一旦有事件传递给view，view的onTouchEvent一定会被调用吗？

答：是的，因为view 本身没有onInterceptTouchEvent方法，所以只要事件来到view这里 就一定会走onTouchEvent方法。

并且默认都是消耗掉，返回true的。除非这个view是不可点击的，所谓不可点击就是clickable和longgclikable同时为false

Button的clickable就是true 但是textview是false。

### 18.enable是否影响view的onTouchEvent返回值？

答：不影响，只要clickable和longClickable有一个为真，那么onTouchEvent就返回true。

### 19.requestDisallowInterceptTouchEvent 可以在子元素中干扰父元素的事件分发吗？如果可以，是全部都可以干扰吗？

答：肯定可以，但是down事件干扰不了。

### 20.dispatchTouchEvent每次都会被调用吗？

答：是的，onInterceptTouchEvent则不会。

### 21.滑动冲突问题如何解决 思路是什么？

答。要解决滑动冲突 其实最主要的就是有一个核心思想。你到底想在一个事件序列中，让哪个view 来响应你的滑动？比如 从上到下滑，是哪个view来处理这个事件，从左到右呢？

用业务需求 来想明白以后 剩下的 其实就很好做了。核心的方法 就是2个 外部拦截也就是父亲拦截，另外就是内部拦截，也就是子view拦截法。学会这2种 基本上所有的滑动冲突

都是这2种的变种，而且核心代码思想都一样。

外部拦截法：思路就是重写父容器的onInterceptTouchEvent即可。子元素一般不需要管。可以很容易理解，因为这和android自身的事件处理机制 逻辑是一模一样的

@Override

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

```
    boolean intercepted = false;
```

```
    int x = (int) ev.getX();
```

```
    int y = (int) ev.getY();
```

```
    switch (ev.getAction()) {
```

```
        //down事件肯定不能拦截 拦截了后面的就收不到了
```

```
        case MotionEvent.ACTION_DOWN:
```

```
            intercepted = false;
```

```
            break;
```

```
        case MotionEvent.ACTION_MOVE:
```

```
            if (你的业务需求) {
```

```
                //如果确定拦截了 就去自己的onTouchEvent里 处理拦截之后的操作和效果 即可了
```

```
                intercepted = true;
```

```
            } else {
```

```
                intercepted = false;
```

```
            }
```

```
            break;
```

```
        case MotionEvent.ACTION_UP:
```

```
            //up事件 我们一般都是返回false的 一般父容器都不会拦截他。因为up是事件的最后一步。这里返回true也没啥意义
```

```
            //唯一的意义就是因为 父元素 up被拦截。导致子元素 收不到up事件，那子元素 就肯定没有onClick事件触发了，这里的
```

```
            //小细节 要想明白
```

```
            intercepted = false;
```

```
            break;
```

```

        default:
            break;
    }
    return intercepted;
}

```

内部拦截法：内部拦截法稍微复杂一点，就是事件到来的时候，父容器不管，让子元素自己来决定是否处理。如果消耗了 就最好，没消耗 自然就转给父容器处理了。

子元素代码：

```

@Override
public boolean dispatchTouchEvent(MotionEvent event) {
    int x = (int) event.getX();
    int y = (int) event.getY();
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            getParent().requestDisallowInterceptTouchEvent(true);
            break;
        case MotionEvent.ACTION_MOVE:
            if (如果父容器需要这个点击事件) {
                getParent().requestDisallowInterceptTouchEvent(false);
            } // 否则的话 就交给自己本身view的onTouchEvent自动处理了
            break;
        case MotionEvent.ACTION_UP:
            break;
        default:
            break;
    }
    return super.dispatchTouchEvent(event);
}

```

父亲容器代码也要修改一下，其实就是保证父亲别拦截down：

```

@Override
public boolean onInterceptTouchEvent(MotionEvent ev) {

    if (ev.getAction() == MotionEvent.ACTION_DOWN) {
        return false;
    }
}

```

```
}  
return true;  
}
```

---

## 安卓应用频道

专注分享安卓应用相关内容



微信号: AndroidPD



长按识别二维码关注

---

伯乐在线 旗下微信公众号

商务合作QQ: 2302462408