onInterceptTouchEvent(MotionEvent ev)
返回是否拦截此事件来自己整

onTouchEvent(MotionEvent ev)
在dispatchTouchEvent方法中调用用来处理点击事件，返回是否消耗此事件。

dispatchTouchEvent(MotionEvent ev)
用来进行事件的分发，如果事件能被传递到当前View，则一定会被调用。
返回受当前View的onTouchEvent方法和子View的dispatchTouchEvent影响，表示是否消耗当前事件。

换句话说，就是

```
poublic boolean dispatchTouchEvent(MotionEvent ev){
if(onInterceptTouchEvent(ev))
return  onTouchEvent(ev);
else
return childView.dispatchTouchEvent(ev);
}
```

对于一个viewgroup来说如果收到了一个点击事件，先调用dispatchTouchEvent，如果onInterceptTouchEvent返回true，表示拦截此事件，自己处理，他的onTouchEvent就会被调用，如果onInterceptTouchEvent返回false的话，表示不拦截，传递给子view，子view的dispatchTouchEvent方法会被调用。手指放上屏幕到抬起中间的一系列动作，为同一事件，一般来说一个事件只能由一个view来处理。除非你把MotionEvent丢到其他view的onTouchEvent来整。

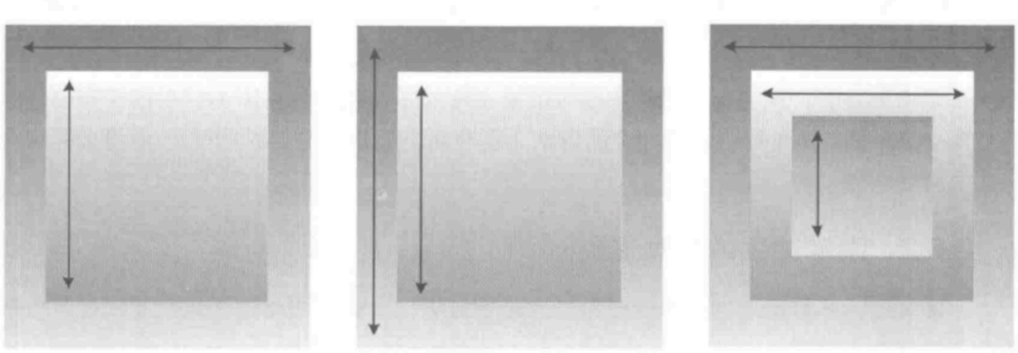
tips：如果设置了onTouchlistener的话，返回false才会调用onTouchEvent,否则不调用onTouchEvent。所以touchListener的优先级比onTouchEvent高。onClickListener的优先级比onTouchEvent还低，处于最底层。

other little tips by《安卓开发艺术》：

(1)viewgroup默认不拦截任何事件，onInterceptTouchEvent返回false。
(2)view没有onInterceptTouchEvent方法，一旦事件给他，他就必须处理。
(3)view的onTouchEvent默认返回true，除非他是不可点击的。
(4)view的enable属性不影响onTouchEvent，只要clickable或者longclickable其中之一为true，onTouchEvent就返回true。
(5)子view可以通过requestDisallowInterceptTouchEvent干预父view的事件分

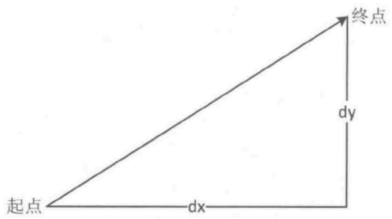
activity先接收到touch事件，然后内部的Window会将其分发至decorView，decorView一般就是当前页面的底部容器，即setContentView的父容器。

滑动冲突常见状态



(1)场景1

针对于场景一的滑动冲突，对见于ViewPager 包装Frangment，这在app首页中是很常见的一种，但是viewpager已经帮我们处理了滑动冲突，如果自己解决，我的思路如下：



用户意欲左右滑动的时候经常会伴随上下的轻微滑动。同理，用户上下滑动的时候，也会伴随轻微的左右滑动。所以，有很多判断方式，例如滑动方向水平竖直的夹角，滑动方向水平竖直位移比较，甚至水平竖直方向的速度比较，来判断用户到底是左右滑动，还是上下滑动。

(2)场景2

针对于场景二的滑动冲突，多见于一个可上下滑动的ViewGroup内包含了一个也可上下滑动的view，例如ScrollView包装ListView，这个时候，从业务角度出发，判断到底是外部view滑动还是内部组件滑动。

(3)场景3

场景三类似于1，2的混合，主要也是从业务出发。区分是谁响应滑动状态

滑动冲突解决思路

(1)外部拦截

```java
@Override
public boolean onInterceptHoverEvent(MotionEvent event) {
    boolean intercept = false;
    float x = event.getX();
    float y = event.getY();
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            intercept = false;
            break;

        case MotionEvent.ACTION_MOVE:
            if (parentNeedThisEvent) { //父布局是否需要该event
                intercept = true;
            } else {
                intercept = false;
            }

        case MotionEvent.ACTION_UP:
            intercept = false;
            break;
    }

    myLastInterceptX = x;
    myLastInterceptY = y;


    return intercept;
}
```

大体代码如下，只需从业务角度判断父布局是否需要该事件

(2)内部拦截

```java
@Override
public boolean dispatchTouchEvent(MotionEvent event) {
    float x = event.getX();
    float y = event.getY();
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            getParent().requestDisallowInterceptTouchEvent(true);
            break;

        case MotionEvent.ACTION_MOVE:
            float deltaX = x - myLastX; //上图dx
            float deltaY = x - myLastY; //上图dy
            if (parentNeedThisEvent) { //父布局是否需要该event
                getParent().requestDisallowInterceptTouchEvent(false);
            }
    }
    myLastX = x;
    myLastY = y;
    return super.dispatchTouchEvent(event);
}
```

内部拦截需要父容器默认拦截除了ACTION_DOWN外所有事件，这样子元素调用getParent().requestDisallowInterceptTouchEvent(true);父容器才能继续拦截事件。如果拦截ACTION_DOWN的话，所有事件都无法传递到子元素