## 对于单View

- 重写onTouchEvent(),在方法内部定制触摸反馈算法
- 是否消费事件取决于ACTION_DOWN事件或POINTER_DOWN事件是否返回true

## 事件传递

- 所有事件都不是独立的,都是以一个系列存在的
- onTouchEvent返回true表示要消费这一系列事件,它的父View的onTouchEvent也不会被调用(事件及后续事件都由这个View消费,都不会向下传)
- 返回值只和DOWN相关,其他值返回没有意义
- 只用getActionMasked(),不用getAction就行
- getActionMasked中包含多点触控信息而已

## onTouchEvent源码

```
    public boolean onTouchEvent(MotionEvent event) {
        // 获取一些基本信息,源码中没有考虑多点触控的问题,如果要实现需要自己写
        final float x = event.getX();
        final float y = event.getY();
        final int viewFlags = mViewFlags;
        final int action = event.getAction();
        // 判断View是不是可点击的
        final boolean clickable = ((viewFlags & CLICKABLE) ==
CLICKABLE
                || (viewFlags & LONG_CLICKABLE) == LONG_CLICKABLE)
                || (viewFlags & CONTEXT_CLICKABLE) ==
CONTEXT_CLICKABLE;
        // 返回clickable是确保当View设置成Disabled时候,点击View事件不会传
递给他的父View进行消费(该阻拦事件传递还是要阻拦事件传递)
        if ((viewFlags & ENABLED_MASK) == DISABLED) {
            if (action == MotionEvent.ACTION_UP && (mPrivateFlags &
PFLAG_PRESSED) != 0) {
                setPressed(false);
            }
            mPrivateFlags3 &= ~PFLAG3_FINGER_DOWN;
            // A disabled view that is clickable still consumes the
touch
            // events, it just doesn't respond to them.
            return clickable;
```

```
            return clickable;
        }
        // 点击代理消费,mTouchDelegate用于增大点击范围
        if (mTouchDelegate != null) {
            if (mTouchDelegate.onTouchEvent(event)) {
                return true;
            }
        }
        // 可能会用就进入,不过不可用就不进去(clickable)
        // TOOLTIP 解释型文字,长按时弹出的提示性文字
        if (clickable || (viewFlags & TOOLTIP) == TOOLTIP) {
            switch (action) {
                case MotionEvent.ACTION_UP:
                    mPrivateFlags3 &= ~PFLAG3_FINGER_DOWN;
                    if ((viewFlags & TOOLTIP) == TOOLTIP) {
                        //显示提示型文字
                        handleTooltipUp();
                    }
                    if (!clickable) {//如果是不可点击的做各种状态的移除,
如果只显示提示型文字,执行到这里就可以了,下面就没必要执行了
                        removeTapCallback();
                        removeLongPressCallback();
                        mInContextButtonPress = false;
                        mHasPerformedLongPress = false;
                        mIgnoreNextUpEvent = false;
                        break;
                    }
                    boolean prepressed = (mPrivateFlags &
PFLAG_PREPRESSED) != 0;
                    if ((mPrivateFlags & PFLAG_PRESSED) != 0 ||
prepressed) {
                        // 按下或预按下状态
                        boolean focusTaken = false;
                        if (isFocusable() &&
isFocusableInTouchMode() && !isFocused()) {//isFocusable()可获取焦点
                        //!isFocused()当前没有获取焦点
                        //isFocusableInTouchMode()实体按键时选中状态
                        //满足以上三种情况就获取焦点
                            focusTaken = requestFocus();
                        }

                        if (prepressed) {
                            // 如果是预按下,松手时就设置成按下
                            setPressed(true, x, y);
                        }

                        if (!mHasPerformedLongPress &&
!mIgnoreNextUpEvent) {//处理点击事件(点击事件是立即生效,抬起状态需要延迟一
```

```
会)
                            // 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 Perfo
rmClick();
                                }
                                if (!post(mPerformClick)) {
                                    performClickInternal();
                                }
                            }
                        }

                        if (mUnsetPressedState == null) {
                            mUnsetPressedState = new
UnsetPressedState();
                        }

                        if (prepressed) {
                            //如果是预按下,加一个延时,出发抬起显示事件,要不
人看不到
                            postDelayed(mUnsetPressedState,
ViewConfiguration.getPressedStateDuration());
                        } else if (!post(mUnsetPressedState)) {
                            // If the post failed, unpress right
now
                            mUnsetPressedState.run();
                        }

                        removeTapCallback();
                    }
                    mIgnoreNextUpEvent = false;
                    break;

                case MotionEvent.ACTION_DOWN:
                    // 是不是摸到屏幕了(抛出实体按键情况)
```

```
                    if (event.getSource() ==
InputDevice.SOURCE_TOUCHSCREEN) {
                        mPrivateFlags3 |= PFLAG3_FINGER_DOWN;
                    }
                    mHasPerformedLongPress = false;
                    // 如果不是点击,设置一个长按的等待器,等待时间到了显示上
面的解释型文字
                    if (!clickable) {
                        checkForLongClick(0, x, y);
                        break;
                    }
                    // 检测鼠标右键点击
                    if (performButtonActionOnTouchDown(event)) {
                        break;
                    }

                    // 是否在滑动控件里(通过检测父View或父View的父View
等shouldDelayCHildPressedState()方法返回true)
                    boolean isInScrollingContainer =
isInScrollingContainer();

                    //
                    if (isInScrollingContainer) {
                        // 如果在滑动控件中,状态就置为预按下状态(在滑动的父
容器中,不知道它是想滑动还是点击,所以先记录下来)
                        // 因为shouldDelayCHildPressedState()默认返回
的是true,所以一般自定义View时应该重写,返回false,否则点击事件会延迟一会(100ms)
                        mPrivateFlags |= PFLAG_PREPRESSED;
                        if (mPendingCheckForTap == null) {
                            // 点击的等待器,是一个runnable
                            mPendingCheckForTap = new
CheckForTap();
                        }
                        mPendingCheckForTap.x = event.getX();
                        mPendingCheckForTap.y = event.getY();
                        postDelayed(mPendingCheckForTap,
ViewConfiguration.getTapTimeout());
                    } else {
                        // 不再滑动控件中置成按下状态
                        setPressed(true, x, y);
                        // 设置一个长按的等待器
                        checkForLongClick(0, x, y);
                        //上面俩个的意思是如果按下立刻弹起来给一个点击,如果
长时间抬起,就给一个长按事件
                    }
                    break;
```

```
                case MotionEvent.ACTION_CANCEL:
                    if (clickable) {

                        setPressed(false);
                    }
                    removeTapCallback();
                    removeLongPressCallback();
                    mInContextButtonPress = false;
                    mHasPerformedLongPress = false;
                    mIgnoreNextUpEvent = false;
                    mPrivateFlags3 &= ~PFLAG3_FINGER_DOWN;
                    break;

                case MotionEvent.ACTION_MOVE:
                    if (clickable) {
                        // 移动时按钮的波纹效果移动
                        drawableHotspotChanged(x, y);
                    }

                    // Be lenient about moving outside of buttons
                    if (!pointInView(x, y, mTouchSlop)) {
                        // 手指移动到View外边,就结束了
                        removeTapCallback();//移除点击状态监听
                        removeLongPressCallback();//移除长按监听
                        if ((mPrivateFlags & PFLAG_PRESSED) != 0)
{//置为未按下状态
                            setPressed(false);
                        }
                        mPrivateFlags3 &= ~PFLAG3_FINGER_DOWN;
                    }
                    break;
            }

            return true;
        }

        return false;
    }
```

## ViewGroup的触摸反馈

- 需要处理父View和子View之间的关系

### onInterceptTouchEvent()

- 只有父View有,子View没有
- 同一个事件序列,不能给俩个View处理
- 父View调用onInterceptTouchEvent,如果返回true,调用自己的onTouchEvent,否则,遍历调用子View的onTouchEvent
- 当ViewGroup处理起来后,就不止一个View可以消费了,ViewGroup可以在一定时刻拦截过来,ViewGroup和View俩个View都参与消费事件(onInterceptTouchEvent开始返回false,滑动一定距离后,返回true,这时候就调用ViewGroup的onTouchEvent了,以后所有事件都有ViewGroup处理了,无法给View了,onInterceptTouchEvent返回false,就给子View,返回true,就调用自己的onTouchEvent,以后就不会出发onInterceptTouchEvent了)
- 事件先有子View处理,在拦截给父ViewGroup的onTouchEvent处理,这时候父View接收不到之前的事件,也就是父View不会接收到Down事件,只会接收到move事件,所以需要在onInterceptTouchEvent中的Down中做一些初始的数据记录(一般是记录按下时的位置)

## 触摸反馈的流程

- Activity.dispatchTouchEvent()
- 递归ViewGroup(View).dispatchTouchEvent()
- ViewGroup.onInterceptTouchEvent()
- child.DispatchTouchEvent()
- super.dispatchTouchEvent()
- View.onTouchEvent()
- Activity.onTouchEvent()
- View也有dispatchTouchEvent方法,dispatchTouchEvent中调用的onTouchEvent,ViewGroup的dispatchTouchEvent调用子View的dispatchTouchEvent
- View的disptachTouchEvent

```
public boolean dispatchTouchEvent(MotionEvent event){
    return onTouchEvent();
}
```

- ViewGroup的dispatchTouchEvent()

```
public boolean dispatchTouchEvent(MotionEvent event){
    if (interceptTouchEvent()){
        return onTouchEvent();
    }else{
        return 调用子View的dispatchTouchEvent();
    }
}
```

## View.dispatchTouchEvent()

- 如果设置了OnTouchLisenter,调用OnTouchLisenter.onTouch()
- 如果OnTouchLisenter消费了事件,返回true
- 如果OnTouchLisenter没有消费事件,继续调用自己的onTouchEvent,并返回onTouchEvent()的结果
- 如果没有设置OnTouchListener,同上

## ViewGroup.dispatchEvent

- 如果用户时初次按下(ACTION_DOWN),清空TouchTargets和DISALLOW_INTERCEPT标记(都是清除之前的标记)

```
            // Handle an initial down.
            if (actionMasked == MotionEvent.ACTION_DOWN) {
                // Throw away all previous state when starting a
new touch gesture.
                // The framework may have dropped the up or cancel
event for the previous gesture
                // due to an app switch, ANR, or some other state
change.
                cancelAndClearTouchTargets(ev);
                resetTouchState();
            }
```

- 拦截处理(onInterceptTouchEvent)

```
            final boolean intercepted;

            if (actionMasked == MotionEvent.ACTION_DOWN
                    || mFirstTouchTarget != null)
{//mFirstTouchTarget == null表示所有的子View都放弃事件,也就不需要调
用onInterceptTouchEvent了
                //子View是否通知过不要拦截,子View是否调用过
requestDisallowInterceptTouchEvent
                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;
            }
```

- 如果不拦截并且不是CANCLE事件,并且时DOWN或者POINTER_DOWN,尝试把pointer通过TouchTarget分配给子View;并且如果分配给了新的子View,调用child.dispatchTouchEvent()把事件传给子View
- parent.requestDisallowInterceptTouchEvent()父View在当下事件序列内,不会对子View进行拦截

## TouchTarget

- 记录每个子View是被哪些pointer按下的
- 结构是单链表

## 如何调用子View的dispatchTouchEvent