1、事件的种类和手势

- 2、View的体系结构和事件分发的框架
- 3、View和ViewGroup的分发流程
- 4、滑动冲突解决方案
- 5、怎样快速成长为移动架构师?

1.1 单点触摸



根据面向对象思想,事件被封装成 MotionEvent 对象

事件	简介		
ACTION_DOWN	手指 初次接触到屏幕 时触发。		
ACTION_MOVE	手指 在屏幕上滑动 时触发,会会多次触发。		
ACTION_UP	手指 离开屏幕 时触发。		
ACTION_CANC EL	事件 被上层拦截 时触发。		

手指落下(ACTION_DOWN) -> 移动(ACTION_MOVE) -> 离开 (ACTION_UP)



1.2 多点触摸

3

多点触控 (Multitouch, 也称 Multi-touch),即同时接受屏幕上多个点的人机交互操作,多点触控是从 Android 2.0 开始引入的功能

事件	简介			
ACTION_DOWN	第一个 手指 初次接触到屏幕 时触发。			
ACTION_MOVE	手指 在屏幕上滑动 时触发,会多次触发。			
ACTION_UP	最后一个 手指 离开屏幕 时触发。			
ACTION_POINTER_DOWN	有非主要的手指按下 (即按下之前已经有手指在屏幕上) 。			
ACTION_POINTER_UP	有非主要的手指抬起 (即抬起之后仍然有手指在屏幕上) 。			
以下事件类型不推荐使用	以下事件在 2.2 版本以上被标记为废弃			
ACTION_POINTER_1_DOWN	第 2 个手指按下,已废弃,不推荐使用。			
ACTION_POINTER_2_DOWN	第 3 个手指按下,已废弃,不推荐使用。			
ACTION_POINTER_3_DOWN	第 4 个手指按下,已废弃,不推荐使用。			
ACTION_POINTER_1_UP	第 2 个手指抬起,已废弃,不推荐使用。			
ACTION_POINTER_2_UP	第 3 个手指抬起,已废弃,不推荐使用。			
ACTION_POINTER_3_UP	第 4 个手指抬起,已废弃,不推荐使用。			



1.3 手势





1、一个手指



2、两个手指



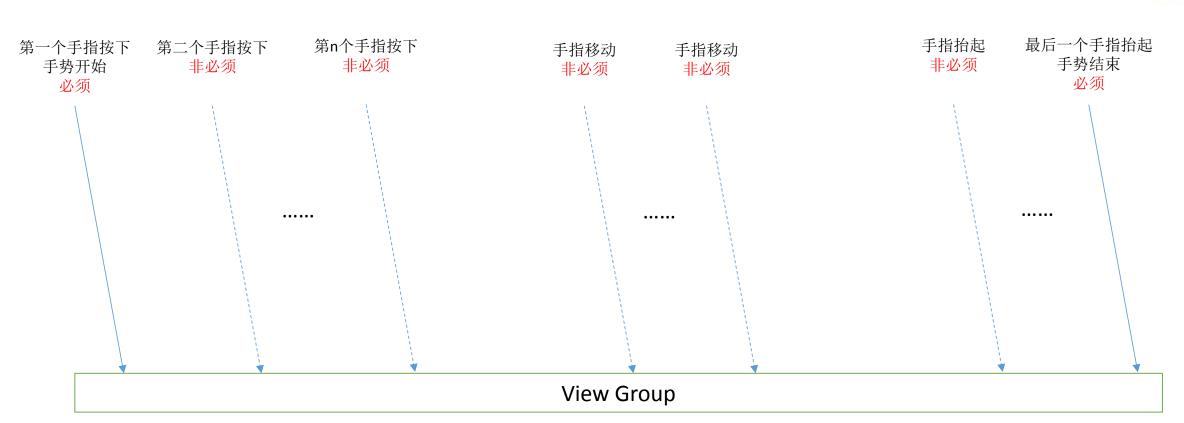
3、手势1



4、手势2

1.4多点手势手指操作流程

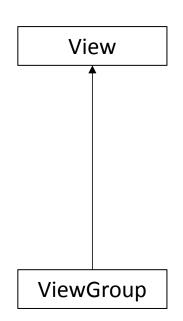


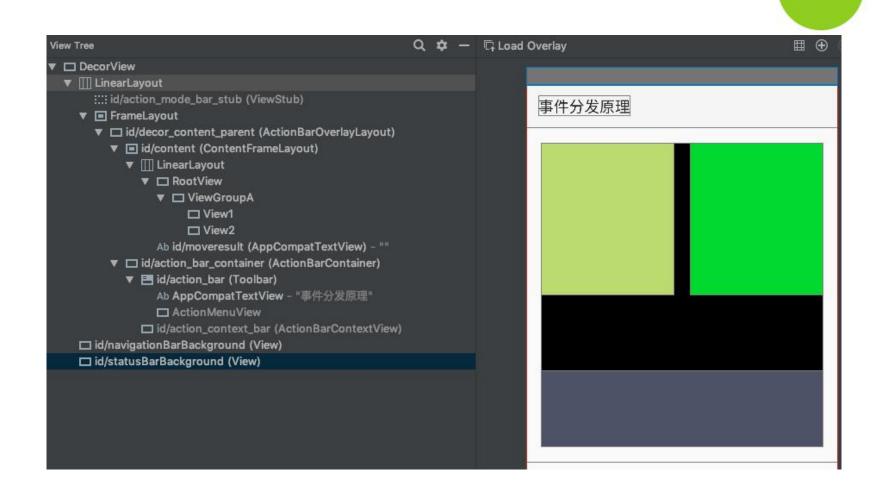


- 1、事件的种类和手势
- 2、View的体系结构和事件分发的框架
- 3、View和ViewGroup的分发流程
- 4、滑动冲突解决方案
- 5、怎样快速成长为移动架构师?

2.1 View和ViewGroup的关系

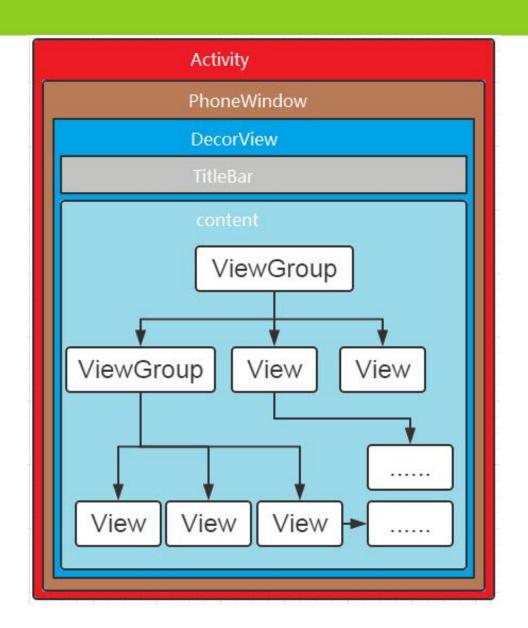


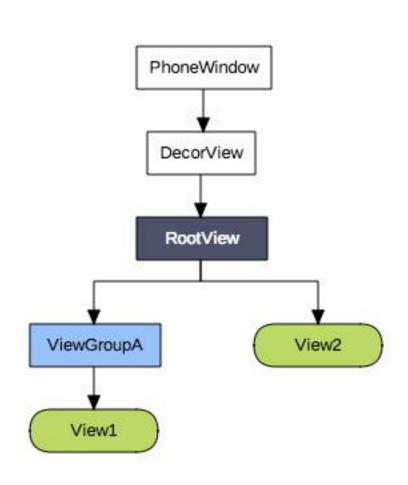




2.2 Android页面View的体系结构 美学







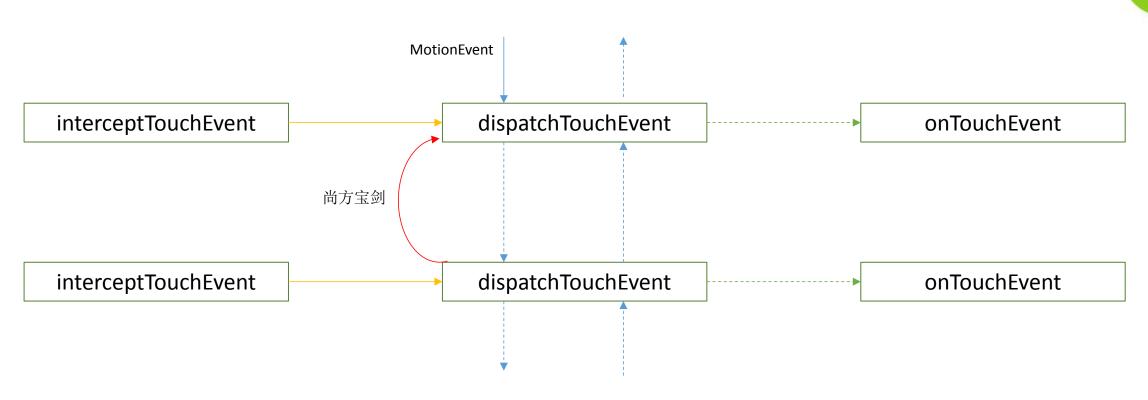
2.3 事件的处理函数



类型	相关方法	Activit y	ViewGroup	View
事件分发	dispatchTouchEven t	V	\checkmark	V
事件拦截	onInterceptTouchE vent	×	√	×
事件消费	onTouchEvent	V	√ X	V

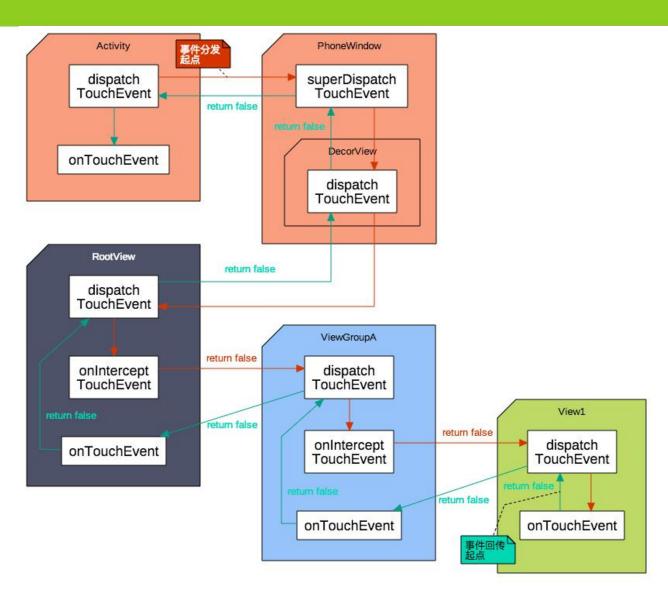
2.4 事件的处理函数的关系





2.5.1 事件分发大流程





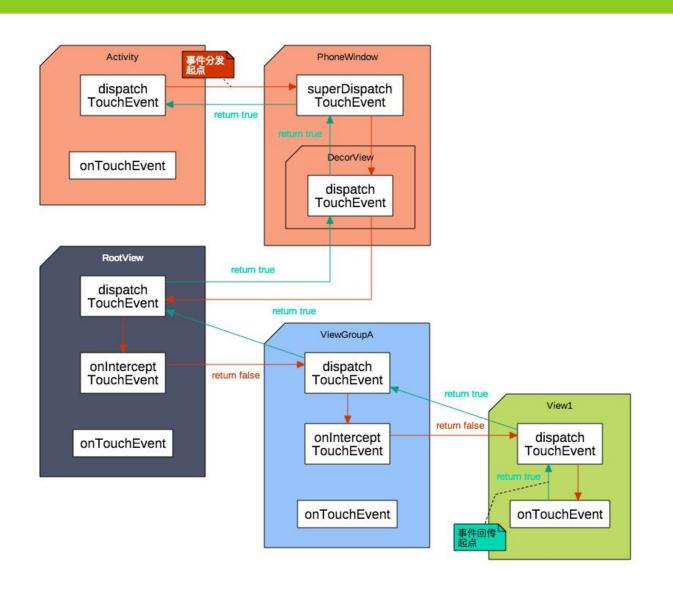
1.事件返回时 dispatchTouchEvent 直接指向了父View的 onTouchEvent 这一部分是不合理的,实际上它仅仅是给了父View的 dispatchTouchEvent 一个 false 返回值,父View根据返回值来调用自身的 onTouchEvent。

2.ViewGroup 是根

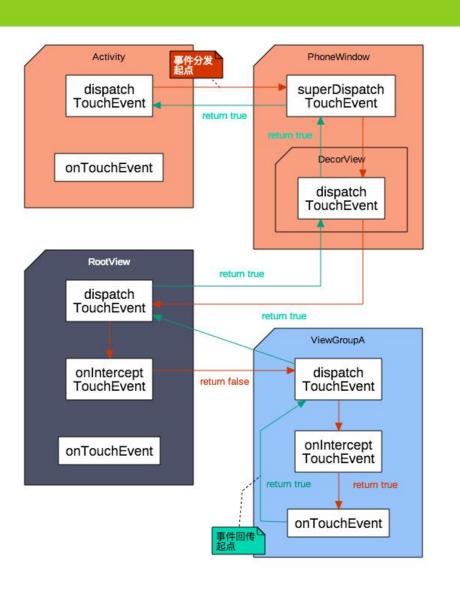
据 onInterceptTouchEvent 的返回值来确定 是调用子View的 dispatchTouchEvent 还是 自身的 onTouchEvent,并没有将调用交

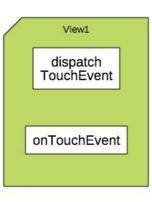
```
給 onInterceptTouchEvent
public boolean dispatchTouchEvent(MotionEvent ev) {
  boolean result = false; // 默认状态为没有消费过
  if (!onInterceptTouchEvent(ev)) {
    // 如果没有拦截交给子View
    result = child.dispatchTouchEvent(ev);
  }
  if (!result) {
    // 如果事件没有被消费,询问自身onTouchEvent
    result = onTouchEvent(ev);
  }
  return result;
}
```

2.5.2 事件分发大流程(View消费了事件)——



2.5.3 事件分发大流程(ViewGroup消费了事件)





- 1、事件的种类和手势
- 2、View的体系结构和事件分发的框架
- 3、View和ViewGroup的分发流程
- 4、滑动冲突解决方案
- 5、怎样快速成长为移动架构师?

3.1 事件分发相关概念







3.2.1 事件分发极简流程



```
public boolean dispatchTouchEvent(MotionEvent ev) {
   boolean result = false; // 默认状态为没有消费过
   if (!onInterceptTouchEvent(ev)) { // 如果没有拦截交给子View
      result = child.dispatchTouchEvent(ev);
   }
   if (!result) { // 如果事件没有被消费,询问自身onTouchEvent
      result = onTouchEvent(ev);
   }
   return result;
}
```

3.2.2 事件分发进阶流程



```
public boolean dispatchTouchEvent(MotionEvent ev) {
 // 默认状态为没有消费过
 boolean result = false;
 //决定是否拦截
 final boolean intercepted = false;
 if (!requestDisallowInterceptTouchEvent()) {
   intercepted = onInterceptTouchEvent(ev);
 //找出最适合接收的孩子
 if (!intercepted && (DOWN || POINTER DOWN || HOVER MOVE)) { // 如果没有拦截交给子View
   for (int i = childrenCount - 1; i >= 0; i--) {
     mFirstTouchTarget = child.dispatchTouchEvent(ev);
 //分发事件
 if (mFirstTouchTarget == null) { // 如果事件没有被消费,询问自身onTouchEvent
   result = onTouchEvent(ev);
 } else {
   for(TouchTarget touchTarget : mFirstTouchTarget) {
     result = touchTarget.child.dispatchTouchEvent(ev);
 return result;
```

3.3.1 View dispatchTouchEvent

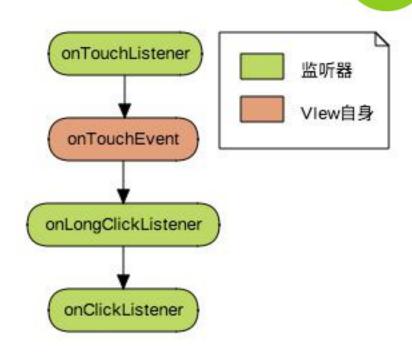


1: 为什么 View 会有 dispatchTouchEvent?

A: 我们知道 View 可以注册很多事件监听器,例如:单击事件(onClick)、长按事件(onLongClick)、触摸事件(onTouch),并且View自身也有 onTouchEvent 方法,那么问题来了,这么多与事件相关的方法应该由谁管理? 毋庸置疑就是 dispatchTouchEvent,所以 View 也会有事件分发。

2: 与 View 事件相关的各个方法调用顺序是怎样的?

- •单击事件(onClickListener)需要两个两个事件(ACTION_DOWN 和 ACTION_UP)才能触发,如果先分配给onClick判断,等它判断完,用户手指已经离开屏幕,黄花菜都凉了,定然造成 View 无法响应其他事件,应该最后调用。(最后)
- •长按事件(onLongClickListener) 同理,也是需要长时间等待才能出结果,肯定不能排到前面,但因为不需要ACTION_UP,应该排在 onClick 前面。(onLongClickListener > onClickListener)
- •触摸事件(onTouchListener) 如果用户注册了触摸事件,说明用户要自己处理触摸事件了,这个应该排在最前面。(最前)
- •View自身处理(onTouchEvent) 提供了一种默认的处理方式,如果用户已经处理好了,也就不需要了,所以应该排在 onTouchListener 后面。(onTouchListener > onTouchEvent)



3.3.2 View和ViewGroup的onTouchEvent

```
public boolean dispatchTouchEvent(MotionEvent event) {
   boolean result = false;
                        // result 为返回值,主要作用是告诉调用者事件是否已经被消费。
   if (onFilterTouchEventForSecurity(event)) {
      ListenerInfo li = mListenerInfo:
       * 如果设置了OnTouchListener, 并且当前 View 可点击, 就调用监听器的 onTouch 方法。
       * 如果 onTouch 方法返回值为 true, 就设置 result 为 true。
      if (li != null && li.mOnTouchListener != null
             && (mViewFlags & ENABLED MASK) == ENABLED
             && li.mOnTouchListener.onTouch(this, event)) {
          result = true;
       * 如果 result 为 false、则调用自身的 onTouchEvent。
       * 如果 onTouchEvent 返回值为 true, 则设置 result 为 true。
       */
      if (!result && onTouchEvent(event)) {
          result = true;
   return result;
 public boolean dispatchTouchEvent(MotionEvent event) {
   if (mOnTouchListener.onTouch(this, event)) {
         return true;
   } else if (onTouchEvent(event)) {
         return true;
    return false;
```

```
public boolean onTouchEvent(MotionEvent event) {
    final int action = event.getAction();
       // 检查各种 clickable
   if (((viewFlags & CLICKABLE) == CLICKABLE ||
            (viewFlags & LONG_CLICKABLE) == LONG_CLICKABLE) ||
            (viewFlags & CONTEXT_CLICKABLE) == CONTEXT_CLICKABLE) {
       switch (action) {
           case MotionEvent.ACTION UP:
               removeLongPressCallback(); // 移除长按
               performClick();
                                          // 检查单击
               break:
           case MotionEvent.ACTION_DOWN:
               checkForLongClick(0);
                                           // 检测长按
               break:
       return true;
                                           // ◀表示事件被消费
    return false;
```

OnClick 和 OnLongClick 去哪里了?

3.4.1 ViewGroup的onInterceptTouchEvent

```
/**
* @param ev The motion event being dispatched down the hierarchy.
* @return Return true to steal motion events from the children and have
* them dispatched to this ViewGroup through onTouchEvent().
* The current target will receive an ACTION_CANCEL event, and no further
* messages will be delivered here.
*/
public boolean onInterceptTouchEvent(MotionEvent ev) {
  if (ev.isFromSource(InputDevice.SOURCE_MOUSE)
      && ev.getAction() == MotionEvent.ACTION_DOWN
      & ev.isButtonPressed(MotionEvent.BUTTON_PRIMARY)
      && isOnScrollbarThumb(ev.getX(), ev.getY())) {
    return true;
  return false;
```

99.99999999999...999%返回为false

3.5.1 ViewGroup的dispatchTouchEvent成员

TouchTarget和mFirstTouchTarget, 这个变量是给手势设置的,跨越事件保留的

```
private static final class TouchTarget {
  private static final int MAX RECYCLED = 32;
  private static final Object sRecycleLock = new Object[0];
  private static TouchTarget sRecycleBin;
  private static int sRecycledCount;
  public static final int ALL POINTER IDS = -1; // all ones
  // The touched child view.
  @UnsupportedAppUsage
  public View child;
 // The combined bit mask of pointer ids for all pointers captured
by the target.
  public int pointerIdBits;
  // The next target in the target list.
  public TouchTarget next;
  @UnsupportedAppUsage
  private TouchTarget() {
```

- 1、mFirstTouchTarget是TouchTarget,是一个链表;
- 2、mFirstTouchTarget记录的是该view的第一个接收该手势的子view;
- 3、mFirstTouchTarget的next的TouchTarget记录的是这个手势的其他手指或者鼠标所在的子view;
- 4、mFirstTouchTarget是一个view group是否有孩子处理该事件的一个标志,为null表明没有孩子处理它,否则表明action down已经分发给他的孩子了;
- 5、mFirstTouchTarget是View group的成员变量,标志着每个view group都有一个这样的变量,如果一个手势被一个view处理,那么他的父亲和祖父们的mFirstTouchTarget都不会为null;

3.5.2局部变量: newTouchTarget & alreadyDispatchedToNewTouchTarget



与action_down, action_pointer_down以及 action_hove_move的view一样,这里先不进行分发,因为mFirstTouchTarget不允许加重复的view,所以现在我们只需要将这个手指id加入到这个view的pointerIdBits就好

- 1、对于这次事件找到了新子view接收处理,设置 newTouchTarget,
- 2、同时设置 alreadyDispatchedToNewTouchTarget为true, 后面不需要再进行分发了

没有找到view愿意接收处理该事件,将这个手指分配给mFirstTouchTarget的最近的一个view,至于这在什么情况下发生,还有待调查和思考?

```
newTouchTarget = getTouchTarget(child);
    (newTouchTarget != null) {
    // thite is atready receiving touch within its bounds.
    // Give it the new pointer in addition to the ones it is handling.
    newTouchTarget.pointerIdBits |= idBitsToAssign;
    break:
if (dispatchTransformedTouchEvent(ev, cancel: false, child, idBitsToAssign)) {
    // Child wants to receive touch within its bounds.
    mLastTouchDownTime = ev.getDownTime();
    if (preorderedList != null) {
       // childIndex points into presorted list, find original index
       for (int j = 0; j < childrenCount; j++) {
          if (children[childIndex] == mChildren[j]) {
             mLastTouchDownIndex = i;
             break;
    } else {
       mLastTouchDownIndex = childIndex:
    mLastTouchDownX = ev.getX();
    mlastTouchDownV = av netV/)
   newTouchTarget = addTouchTarget(child, idBitsToAssign);
   alreadyDispatchedToNewTouchTarget = true;
   (newTouchTarget == null && mFirstTouchTarget != null) {
     // Did not find a child to receive the event.
     // Assign the pointer to the least recently added target
     newTouchTarget = mFirstTouchTarget;
     while (newTouchTarget.next != null) {
         newTouchTarget = newTouchTarget.next;
     newTouchTarget.pointerIdBits |= idBitsToAssign;
```

3.6.1 辅助功能view的分发逻辑



1、如果现在分发的这个事件就是分发给我这个辅助功能获焦的view的,那么我们立即进行正常的分发,同时清除这个MotionEvent事件FLAG_TARGET_ACCESSIBILITY_FOCUS标志, 请注意这个标志是事件的,而不是view的

```
// If the event targets the accessibility focused view and this is it, start
// normal event dispatch. Maybe a descendant is what will handle the click.

if (ev.isTargetAccessibilityFocus() && isAccessibilityFocusedViewOrHost()) {
    ev.setTargetAccessibilityFocus(false);
}

2、如果这个事件被该ViewGroup拦截,或者已经有子view正在处理这个手势,我们清除这个事件的FLAG_TARGET_ACCESSIBILITY_FOCUS标志,进行正常的分发
// If intercepted, start normal event dispatch. Also if there is already
// a view that is handling the gesture, do normal event dispatch.

if (intercepted | | mFirstTouchTarget != null) {
    ev.setTargetAccessibilityFocus(false);
}
```

3.6.2 辅助功能view的分发逻辑

学学

- 3、如果这个事件带有
- FLAG_TARGET_ACCESSIBILITY_FOCUS标志,说明他是来自辅助功能的,那么我们优先分发给辅助功能view,如果它不处理,那么我们就会清除这个标志,照常分发给其他的所有孩子;
- 现在让我们寻找具有辅助功能的宿主,同时避免持有这种状态,因为这些辅助功能的事件是非常少有的。
- 4、有FLAG_TARGET_ACCESSIBILITY_FOCUS标志的事件能通过这个条件的view一定是要能接受pointerevents的,并且落在这个view的范围之内,否则,清除FLAG_TARGET_ACCESSIBILITY_FOCUS,继续。。。
- 5、分发完成后,没有人愿意受理这个辅助功能的事件,所以我们直接清除这个 FLAG TARGET ACCESSIBILITY FOCUS标志位。

优先处理辅助,如果辅助view 没有分发成功,还会分发一次

```
If the event is targeting accessibility focus we give it to the
// view that has accessibility focus and if it does not handle it
// we clear the flag and dispatch the event to all children as usual.
// We are looking up the accessibility focused host to avoid keeping
// state since these events are very rare.
View childWithAccessibilityFocus = ev.isTargetAccessibilityFocus()
       ? findChildWithAccessibilityFocus() : null;
if (actionMasked == MotionEvent.ACTION DOWN
        II (split && actionMasked == MotionEvent.ACTION POINTER DOWN)
        | actionMasked = MotionEvent.ACTION_HOVER_MOVE) {
   final int actionIndex = ev.getActionIndex(); // always 0 for down
   final int idBitsToAssign = split ? 1 << ev.getPointerId(actionIndex)</pre>
           : TouchTarget.ALL POINTER IDS:
   // Clean up earlier touch targets for this pointer id in case they
   // have become out of sync.
   removePointersFromTouchTargets(idBitsToAssign);
   final int childrenCount = mChildrenCount;
   if (newTouchTarget == null && childrenCount != 0) {
       final float x = ev.getX(actionIndex);
       final float y = ev.getY(actionIndex);
       // Find a child that can receive the event.
       // Scan children from front to back.
       final ArrayList<View> preorderedList = buildTouchDispatchChildList();
       final boolean customOrder = preorderedList == null
               && isChildrenDrawingOrderEnabled();
       final View[] children = mChildren;
       for (int i = childrenCount - 1; i >= 0; i--) {
           final int childIndex = getAndVerifyPreorderedIndex(
                   childrenCount, i, customOrder);
           final View child = getAndVerifyPreorderedView(
                   preorderedList, children, childIndex);
           // If there is a view that has accessibility focus we want it
           // to get the event first and if not handled we will perform a
           // normal dispatch. We may do a double iteration but this is
           // safer given the timeframe.
           if (childWithAccessibilityFocus != null) {
               if (childWithAccessibilityFocus != child) {
               childWithAccessibilityFocus = null;
                   || !isTransformedTouchPointInView(x, y, child, outLocalPoint: null)) {
```

- 1、事件的种类和手势
- 2、View的体系结构和事件分发的框架
- 3、View和ViewGroup的分发流程
- 4、滑动冲突解决方案
- 5、怎样快速成长为移动架构师?

4.1 滑动冲突解决方案



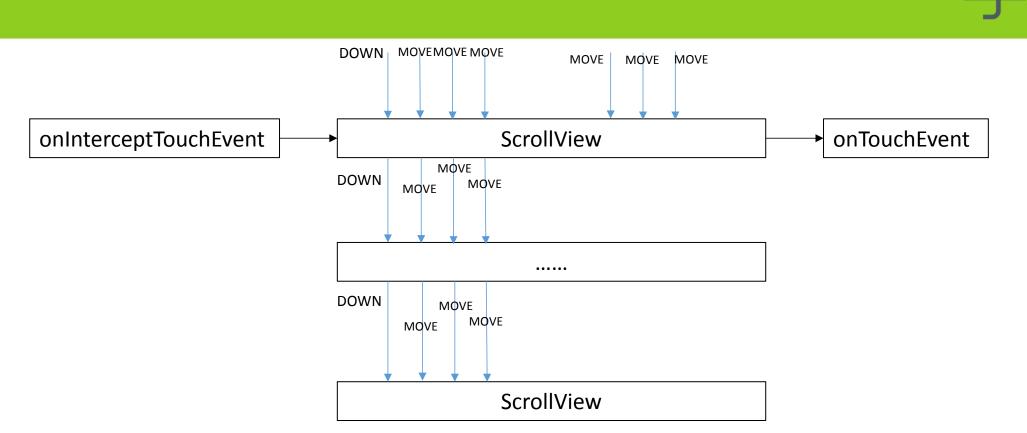


2、内部拦截





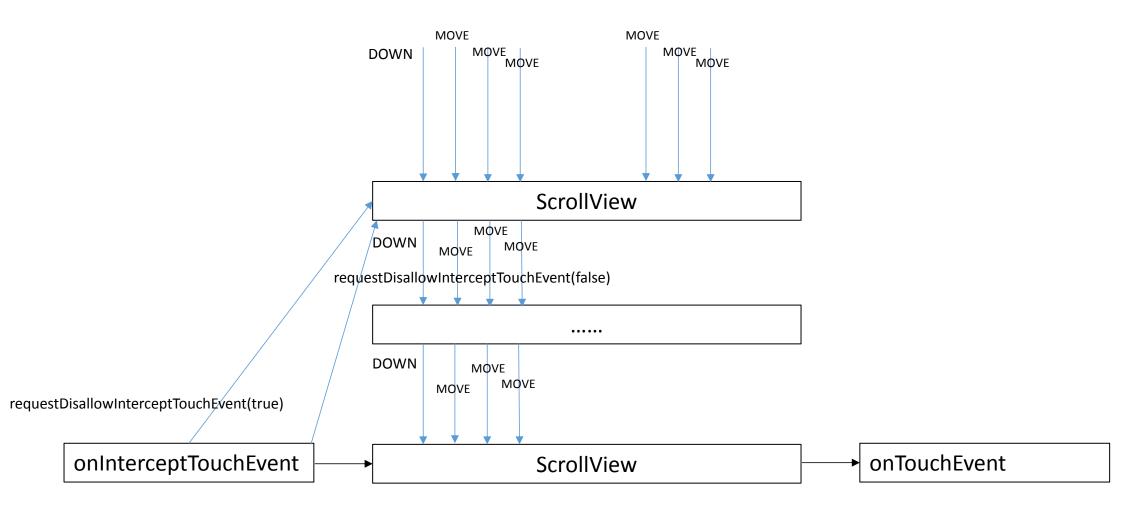
4.2 外部拦截

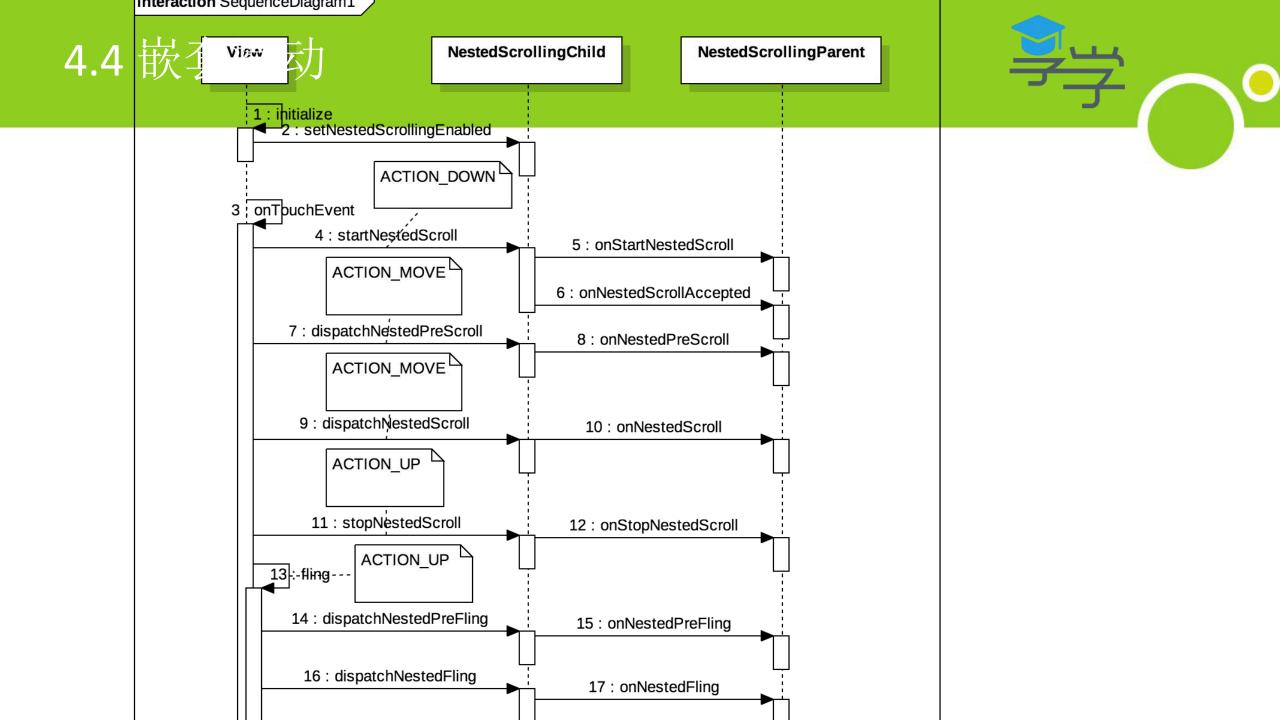


- •当ViewPager接收到DOWN事件,ViewPager默认不拦截DOWN事件,DOWN事件交由ListView处理,由于ListView可以滚动,即可以消费事件,则ViewPager的mFirstTouchTarget会被赋值,即找到处理事件的子View。然后ViewPager接收到MOVE事件,
- •若此事件是ViewPager不需要,则同样会将事件交由ListView去处理,然后ListView处理事件;
- •若此事件ViewGroup需要,因为DOWN事件被ListView处理,mFirstTouchEventTarget会被赋值,也就会调用onInterceptedTouchEvent,此时由于ViewPager对此事件感兴趣
- ,则onInterceptedTouchEvent方法会返回true,表示ViewPager会拦截事件,此时当前的MOVE事件会消失,变为CANCEL事件,往下传递或者自己处理,同时mFirstTouchTarget被重置为null。
- •当MOVE事件再次来到时,由于mFristTouchTarget为null,所以接下来的事件都交给了ViewPager。

4.3 内部拦截







- 1、事件的种类和手势
- 2、View的体系结构和事件分发的框架
- 3、View和ViewGroup的分发流程
- 4、滑动冲突解决方案
- 5、怎样快速成长为移动架构师?

5.1 Android程序员职业发展规划





01 技术序列

- 技术攻坚
- 架构设计
- 专业知识

...

高级技术专家

技术专家

资深工程师

高级技术总监

技术总监

技术经理

高级工程师

工程师

2 管理序列

- 团队管理
- 项目管理
- 沟通协作

谢谢大家

