1. 安卓事件的分发

安卓的事件分发大概会经历 Activity -> PhoneWindow -> DecorView -> ViewGroup -> View 的 dispatchTouchEvent。

// 伪代码

public boolean dispatchTouchEvent() {

boolean res = false;

// 是否不允许拦截事件

// 如果设置了 FLAG\_DISALLOW\_INTERCEPT，不会拦截事件，所以在 child 里可以通过 requestDisallowInterceptTouchEvent 控制父 View 是否来拦截事件

final boolean disallowIntercept = (mGroupFlags & FLAG\_DISALLOW\_INTERCEPT) != 0;

if (!disallowIntercept && onInterceptTouchEvent()) { // View 不调用这里，直接执行下面的 touchlistener 判断

if (touchlistener && touchlistener.onTouch()) {

return true;

}

res = onTouchEvent(); // 里面会处理点击事件 -> performClick() -> clicklistener.onClick()

} else if (DOWN) { // 如果是 DOWN 事件，则遍历子 View 进行事件分发

// 循环子 View 处理事件

for (childs) {

res = child.dispatchTouchEvent();

}

} else {

// 事件分发给 target 去处理，这里的 target 就是上一步处理 DOWN 事件的 View

target.child.dispatchTouchEvent();

}

return res;

}

1. 事件是如何到达 Activity 的

用户点击设备, linux 内核接受中断, 中断加工成输入事件数据写入对应的设备节点中, InputReader 会监控 /dev/input/ 下的所有设备节点, 当某个节点有数据可以读时，通过 EventHub 将原始事件取出来并翻译加工成输入事件，交给 InputDispatcher，InputDispatcher 根据 WMS 提供的窗口信息把事件交给合适的窗口,窗口 ViewRootImpl 派发事件，通过Activity的dispatchTouchEvent传递给DecorView的dispatchTouchEvent，从而传递给整个View树。

其中主要有几个阶段：

硬件中断：操作系统对硬件事件的接收是通过中断来进行的。

内核启动的时候会在中断描述符表中对中断类型以及对应的处理方法的地址进行注册。

当有中断的时候，就会调用对应的处理方法，把对应的事件写入到设备节点里。

InputManagerService：InputManagerService 是用来处理 Input 事件的，Java 侧的 InputManagerService 就是 C++ 代码的一个封装，以及提供了一些 callback 用来传递事件到 Java 层。初始化 EventHub，EventHub 的作用是用来监控设备节点是否有更新。 初始化 InputManager，InputManager 里初始化了 InputReaderThread 和 InputDispatcherThread 两个线程，一个用来读取事件，一个用来派发事件。

InputReaderThread：InputReaderThread 里做了三件事情：

从 EventHub 获取事件

处理事件，这里事件有不同的类型，会做不同的处理和封装

把事件发送给 InputDispatcher

InputDispatcherThread：其中会找到当前合适的 Window，然后调用 InputChannel 去发送事件。

这里的 InputChannel 对应的是 ViewRootImpl 里的 InputChannel。

WindowInputEventReceiver：在 ViewRootImpl 里有一个 WindowInputEventReceiver 用来接受事件并进行分发。

InputChannel 发送的事件最终都是通过 WindowInputEventReceiver 进行接受。

WindowInputEventReceiver 是在 ViewRootImpl.setView 里面初始化的，setView 的调用是在 ActivityThread.handleResumeActivity -> WindowManagerGlobal.addView。最终就调用到 Activity.dispatchTouchEvent 里，也就是开始的流程了。

1. CANCEL 事件什么时候会触发

这个如果仔细看 dispatchTouchEvent 的代码的话，可以看到一些时机：

1. View 收到 ACTION\_DOWN 事件以后，上一个事件还没有结束（可能因为 APP 的切换、ANR 等导致系统扔掉了后续的事件），这个时候会先执行一次 ACTION\_CANCEL
2. 子 View 之前拦截了事件，但是后面父 View 重新拦截了事件，这个时候会给子 View 发送 ACTION\_CANCEL 事件
3. 如何解决滑动冲突

1.通过重写父类的 onInterceptTouchEvent 来拦截滑动事件

2.通过在子类中调用 parent.requestDisallowInterceptTouchEvent 来通知父类是否要拦截事件，requestDisallowInterceptTouchEvent 会设置 FLAG\_DISALLOW\_INTERCEPT 标志