掌握Android Touch事件机制

Dave Smith

@devunwired

# 要点涵盖

* Touch系统概述
* Framwork层的Touch事件
* Touch事件的处理
* 系统提供的Touch Handlers
* 系统提供的Gesture Handlers

# Android系统怎么处理Touch事件？

* 每一个touch事件都被封装成一个MotionEvent对象
* 这个对象描述了用户当前的动作

– ACTION\_DOWN

– ACTION\_UP

– ACTION\_MOVE

– ACTION\_POINTER\_DOWN

– ACTION\_POINTER\_UP

– ACTION\_CANCEL

* 事件元数据包括

– 触摸的位置

– 触摸点的个数（或触摸手指的根数）

– 事件发生的时间

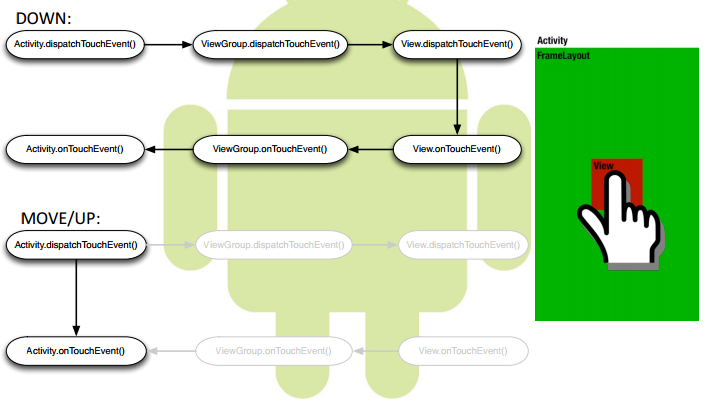
* 一个“手势（gesture）”是以ACTION\_DOWN开始，以ACTION\_UP结束的
* 事件开始于Activity的dispatchTouchEvent()函数（看名字就知道这是对事件进行分发的）
* 事件在views之间依次传递

– 父view（ViewGroups）将事件传递给他的子view

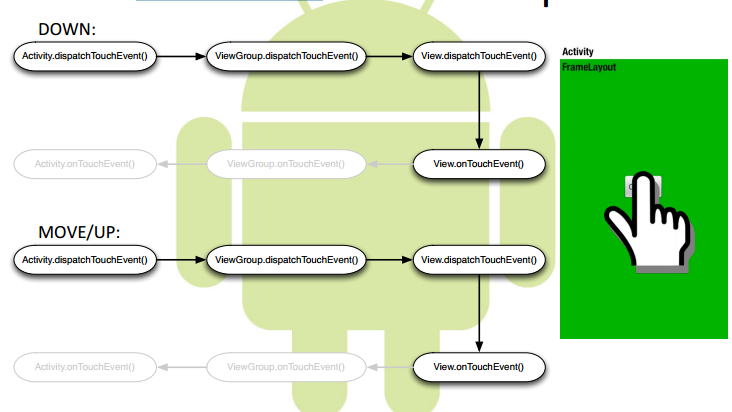
– 事件在传递过程中可以随时被拦截

* 事件会在“传递链”中流动，直到回到Activity中,除非在流动的过程中被消费掉了
* 所有没有被消费的事件最终会回到Activity的onTouchEvent()，并结束传递。
* 外部定义的OnTouchListener能在任何View/ViewGroup中拦截事件，该接口的实现是可选的，即可以实现也可以不实现该接口。
* Activity.dispatchTouchEvent()
* 总是第一个被调用。
* 将事件派送给Windows的根视图（Root View）
* onTouchEvent()
* 如果没有View消费事件，该函数最终会被调用
* 总是最后一个被调用的函数。
* View.dispatchTouchEvent()
* 如果注册了View.OnTouchListener,首先会将事件传递给它处理，调用：
* View.onTouchListener.onTouch()
* 如果没有被消费，View会自己处理，调用：
* View.onTouchEvent()
* ViewGroup.dispatchTouchEvent()
* onInterceptTouchEvent()
* 检查是否要拦截事件，不再传递给子View
* 但是会将ACTION\_CANCEL事件传递给active child
* 一旦消费掉随后而来的所有事件，该函数就会返回true
* 对于每个子View，按照他们被添加的顺序的反序，进行如下操作：
* 如果touch事件是有意义的（即在View内），调用子view的dispatchTouchEvent();
* 如果事件没有被处理，便将事件传递给下一个View。
* 如果没有一个子view处理了事件，便交给ViewGroup的OnTouchListener（如果定义了）处理，调用：
* OnTouchListener.onTouch()
* 如果没有定义OnTouchListener或者OnTouchListener没有处理，调用：
* onTouchEvent()
* 被拦截的事件会跳过子view的处理步骤。
* 一些例子

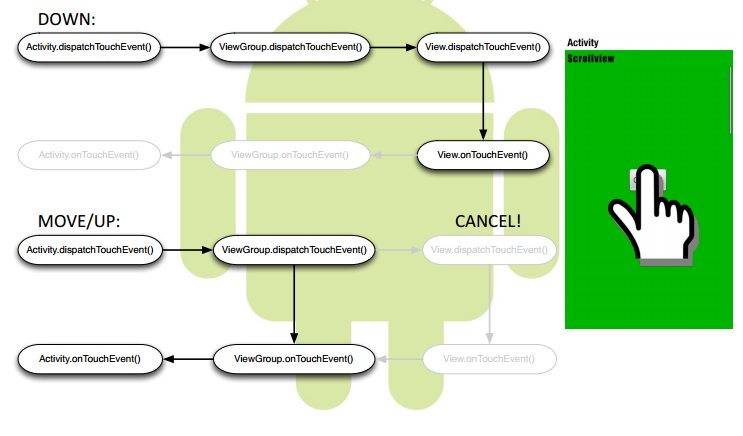
**没有view处理事件时的传递流程**



**有view处理事件时的传递流程**

****

**ViewGroup拦截事件时的传递流程**



**Touch事件处理**

* 如何处理Touch事件

– 子类复写（override）父类的onTouchEvent()函数

– 实现OnTouchListener接口

* 消费事件

– ACTION\_DOWN时返回true，以表示你对该事件感兴趣

* 即使你不对ACTION\_DOWN感兴趣，也返回true

– 对于其他事件，返回true即中断了该事件的继续传递

* ViewConfiguration中一些有用的方法

– getScaledTouchSlop()

表示一段距离，只有当手移动的距离≥该距离以后才会被认为是一个move事件

– getScaledMinimumFlingVelocity()

表示一个速度，当手滑动的数据≥该速度的时候才会被认为是一个fling事件

– getLongPressTimeout()

表示一段时间，当超过点击事件≥该时间后才会被认为是一个long-press事件

– 显示每个设备的设备密度值

* 触发Touch事件的“流动”

– 调用dispatchTouchEvent()

– 不要直接调用onTouchEvent()

类似的，调用requestLayout()而非onLayout()；调用invalidate ()而非onDraw();调用performClick()而非onClick()(译者注);

* 拦截Touch事件（针对ViewGroup）

– 子类复写（override）父类的onInterceptTouchEvent()函数

– 当你想拦截事件时return true

针对当前操作（current gesture）的所有事件都会直接进入你的OnTouchEvent()

针对所有这些事件onInterceptTouchEvent()也不会再被调用

– 当前所有目标（View和Activity）会接到ACTION\_CANCEL事件

* 尽可能调用super.onTouchEvent()

– View.onTouchEvent()做了很多工作来维护状态（pressed、checked等），如果你自己处理了这些事件，你会丢失很多的处理工作。

* 对ACTION\_MOVE事件做边界检查（Protect ACTION\_MOVE with slop checks）

– 因为手指操作毕竟不精细（Fingers are fat and twitchy）

* 总是记得处理ACTION\_CANCEL

– 一些容器类View（例如ScrollView）会拦截事件，因此你需要处理ACTION\_CANCEL来重置状态

– 记住，处理完ACTION\_CANCEL后，不会再有其他事件了。

**多点触控**

* MotionEvent.getPointerCount()

— 返回当前时刻屏幕上有多少触摸点

* 使用ACTION\_POINTER\_DOWN和ACTION\_POINTER\_UP来监听有多少个“二次触摸点”。

– MotionEvent.ACTION\_POINTER\_DOWN:当屏幕上已经有一个点被按住，此时再按下其他点时触发。

– MotionEvent.ACTION\_POINTER\_UP:当屏幕上有多个点被按住，松开其中一个点时触发（即非最后一个点被放开时）。

– MoEonEvent.getAcEonMasked()

– MoEonEvent.getAcEonIndex()

**批处理**

* 为了提高效率，ACTION\_MOVE事件可以放在一个MotionEvent中进行统一处理
* 标准的函数总是返回最近（当前）的事件。

如：getX(),getY(),getEventTime()只返回最近（当前）事件的X坐标，Y坐标和事件发生时间。

* 发生在ACTION\_MOVE和最后一次事件之间的事件可以在历史方法（historical method）中得到

– getHistoricalX(), getHistoricalY(), getHistoricalEventTime()

– getHistoricalSize() 返回进行统一批处理的事件个数

**系统封装好的Touch事件处理对象**

* 常用的有：

• OnClickListener

• OnLongClickListener

• OnTouchListener

— 一个listener能监视一个单一的事件

—一个listene能够消费一个/些事件

* OnScrollListener/View.onScrollChanged()

— 具有滑动功能的view滑动时调用的函数

* 针对更复杂的Touch事件，有如下的对象

**GestureDetector**

– onDown()

– onSingleTapUp()

– onDoubleTap()

– onLongPress()

– onScroll()

– onFling()

**ScaleGestureDetector**

– onScaleBegin()

– onScale(),

– onScaleEnd()

以上的操作本质上都是通过OnTouchListener()和OnTouchEvent()来实现的

**缺点**

– 消费了Up事件，且没有暴露处理CANCEL事件的接口

**委托Touch**

Android 允许使用TouchDelegate来扩展Touch事件的响应区域。

使用方法：

ViewGroup parent;//父View

View child;//期望委托的子View

Rect touchArea;//委托Touch区域

parent.setTouchDelegate( new TouchDelegate(touchArea, child) );

原文链接<http://wugengxin.cn/download/pdf/android/PRE_andevcon_mastering-the-android-touch-system.pdf>