Android触摸事件机制

# 基础介绍

## 事件的介绍

* 何时产生

当用户触摸view或者viewGroup其派生对象的时候，会触发事件。

* 事件类型

主要谈论常用的三个事件类型（其他的暂不考虑在内）。

表1.1 事件类型

|  |  |
| --- | --- |
| 事件类型 | 内容 |
| ACTION\_DOWN | 用户手指按下时进行触发 |
| ACTION\_MOVE | 手指按下到手指松开之间，如果移动距离大于一个数值就触发。 |
| ACTION\_UP | 手指离开屏幕触发 |

更加形象点的话就看如下事件示意图：

Down

Move

Up

多次移动多次触发

图1-1事件机制示意图

* 主要触发方法

在传递事件的过程中，主要涉及如下三个阶段：

1. 分发（dispatch）:

该阶段对应dispatchTouchEvent()方法。

public boolean dispatchTouchEvent(MotionEvent event)

该方法中可以来控制事件是否继续分发到子视图处理。如果返回false，继续传递；否则表示消费掉了，不再传递了。还可以设置：

return super.dispatchTouchEvent(event);

表示继续分发事件。如果是ViewGroup其中会有onInterceptTouchEvent()方法，是否拦截当前事件。

1. 拦截（Intercept）

该阶段对应onInterceptTouchEvent()方法。

public boolean onInterceptTouchEvent(MotionEvent ev)

该方法的可以控制事件是否拦截，返回值如果是false或者super.onInterceptTouchEvent方法表示不拦截，继续传递子视图。

1. 消费（consume）：

该阶段对应onTouchEvent方法。

public boolean onTouchEvent(MotionEvent event)

该方法主要对应该视图处理该事件，返回值true表示事件就这样处理完了，不再传递给上层视图；否则表示当前视图没有处理完，还需要继续传递给上层视图。

* 会传递的类

在Android中，主要还是有三种类可以传递事件：

表1.2 引用传递处理能力的类

|  |  |
| --- | --- |
| 类 | 能力 |
| Activity | 分发和消费 |
| View | 分发和消费 |
| ViewGroup | 分发、拦截和消费 |

* 传递顺序

那肯定是activity->viewGroup->view

注意：viewGroup集成自View，本文所表达的view是那些集成自view的空间，比如说TextView，ImageView等。

表1.3 类型介绍

|  |  |
| --- | --- |
| 类型 | 介绍 |
| Activity | 四大组件之一 |
| View | 所有控件基类 |
| ViewGroup | 多种View的集合 |

简单都说就是从外部开始传递。

# Activity事件分发分析

* 查看dispatchTouchEvent方法：

public boolean dispatchTouchEvent(MotionEvent ev) {

//ACTION\_DOWN事件开始  
 if (ev.getAction() == MotionEvent.*ACTION\_DOWN*) {  
 onUserInteraction();  
 }

//从window对象开始分发  
 if (getWindow().superDispatchTouchEvent(ev)) {  
 return true;  
 }

//Window分发返回true表面之前有处理掉该事件了，就不会调用activity的ontTouchEvent。  
 return onTouchEvent(ev);  
}

* 查看onUserInteraction方法

//按下功能键的时候回触发该方法（back，home等）

public void onUserInteraction() {  
}

* 查看onTouchEvent分发：

public boolean onTouchEvent(MotionEvent event) {

//主要是对于处理边界外点击事件：

//是否是DOWN事件

//是否event的坐标在边界内  
 if (mWindow.shouldCloseOnTouch(this, event)) {  
 finish();  
 return true;

}  
  
 return false;  
}

* 查看getWindow().superDispatchTouchEvent()：

public abstract boolean superDispatchTouchEvent(MotionEvent event);//一个抽象方法啊

在Window对象中只有该抽象方法，那它的实例方法在哪里呢？那就是要找window的子类（PhoneWindow）

* 查看PhoneWindow的superDispatchTouchEvent方法：

@Override  
public boolean superDispatchTouchEvent(MotionEvent event) {  
 //调用的是mDcor（DecorView）

return mDecor.superDispatchTouchEvent(event);  
}

* 查看DecorView的superDispatchTouchEvent方法:

public boolean superDispatchTouchEvent(MotionEvent event) {  
 return super.dispatchTouchEvent(event);//viewGroup分发  
}

终于看到了dispatchTouchEvent方法（用来分发事件）。

DecorView是什么呢？

窗口最顶层的视图

换句话说：DecorView集成FrameLayout,是所有界面的父界面。

在activity界面样式外部包裹的就是DecorView，这里调用的super.dispatchTouchEvent方法开始就是FrameLayout的dispatchTouchEvent，就是ViewGroup的dispatchTouchEvent。

接下来的分析将结合实例结果，不再参考源码，5.0版本以上的分发源码还是很复杂的。如果想看源码解析的小伙伴可以阅读这篇机制详解

（https://www.jianshu.com/p/38015afcdb58）

# 实例分析

下面将进行测试，布局是在activity中布局为EventRelativeLayout，其中放有一个EventTextView。点击EventTextView，查看传递结果。

效果如图：



图2-2 测试图

1）测试1

中间不做任何处理，返回值都是父类该方法。

看如下结果：

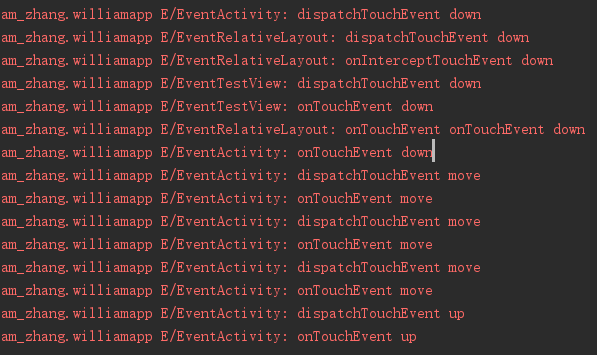


图2-1 结果一

先调用activity的分发方法，到layout的分发方法，再到layout的拦截方法，不进行拦截，到了最后view的分发方法，将调用本身消费方法，传递给上一级调用其消费方法，直到到达acitivity的分发方法中的消费方法。

不进行其他处理返回值都是false。

那如果进行其他处理呢？

2）测试2

textView的消费方法返回true，表示已经消费。

看到如下结果：



图2-3 结果二

TextView消费掉了事件，事件不再继续传递给上层。

3）测试3

Activity分发事件返回true，表示分发已经处理。

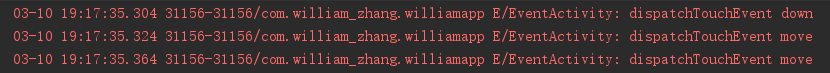


图2-4 结果三

Activity分发事件返回false，表示事件未消费完。

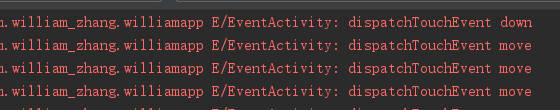


图2-5 结果三

发现activity不管返回false还是true，都不继续分发了，为什么呢？

实际上后面的分发都是通过调用父类分发方法来查找下一个分发目标的。

4）测试4

Layout分发不调用父类分发方法，返回true或者返回false：

返回true：

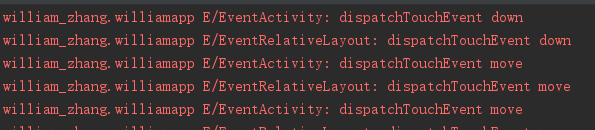


图2-6 结果4

事件被处理，不再继续传递。

返回false:

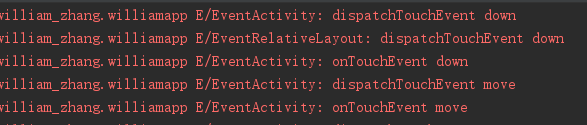


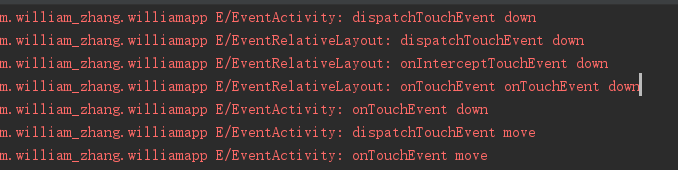
图 2-7 结果4

事件不再向下分发，将会调用上层onTouchEvent时间。

5）测试5

Layout拦截不调用父类分发，返回true或者返回false：

返回true：



返回false：

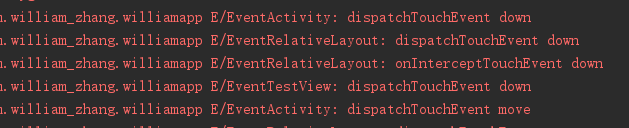


表明拦截时间会拦截下层分发，然后会向上调用onTouchEvent分发消费掉。

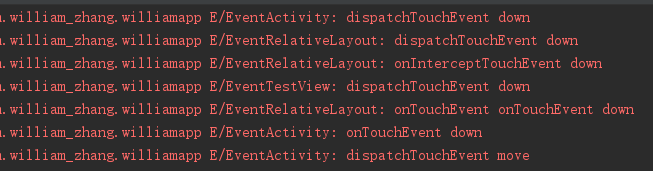
6）测试6

TextView分发不调用父类分发，返回true或者false：

返回true：



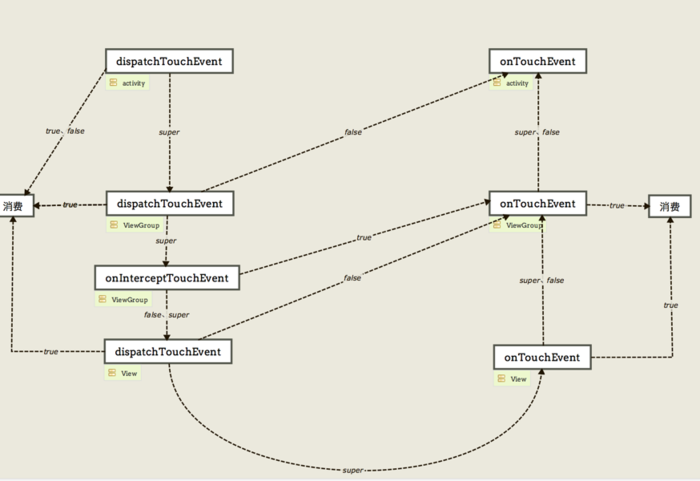
事件不再传递，消费掉了。



事件不再向下层分发，而是向上调用消费方法。

测试完成，可以总结为如下图：

该图来自图解事件分发机制（<https://www.jianshu.com/p/7cf6eb4ce7a9>）；



有兴趣的小伙伴可以结合源码，那样可以更加深刻的掌握事件分发机制。

可以阅读5.0版本以下的三个阶段的源码，那会简单些。

感谢阅读！