|  |
| --- |
| 触摸反馈的本质：用户操作屏幕点击、滑动、长按等 --> 一序列触摸事件 --> 触摸算法（解读出是什么操作，给出对应的反馈。）  ACTION\_DOWN、ACTION\_MOVE、ACTION\_UP、ACTION\_CANCEL。  通过View的onTouchEvent(MptionEvent event)来处理事件，如果消费该事件就返回true。  事件拦截机制：为解决view叠加对事件的响应的，在每一事件到达view的onTouchEvent()之前，有一个过程：从activity的根view一级一级去询问你要不要拦截这组事件？调用 ViewGroup.onInterceptTouchEvent()。  onInterceptTouchEvent();这个方法可以整个事件流中，对每个事件进行监听（这点和onTouchEvent()是不同的）。 |

1、触摸反馈，也就是点击、长按、滑动这些操作它们的本质原理，以及如何去实现一个自己的触摸反馈算法。

触摸反馈的本质就是把一系列的触摸事件解读为对应的操作。比如，按下抬起就是点击，按下移动一段距离就是滑动；然后你根据解读出来的操作给出相应的反馈，这个就是触摸反馈的本质。

触摸事件并不是相互独立的，而是成序列的，成组的。

每一组事件全部都以按下事件DOWN为开头，以抬起事件或取消事件为结束。

这个取消事件cancel 它是一种特殊的事件。它对应的是事件序列的非人为的提前结束。



1. 每一个触摸事件都会交给view的onTouchEvent(MotionEvent event)方法来处理。

在用户触摸屏幕产生触摸操作的时候，该方法就会被调用。

MotionEvent代表一个什么类型的事件：是按下、移动还是抬起，以及这个事件的坐标、时间等等信息。

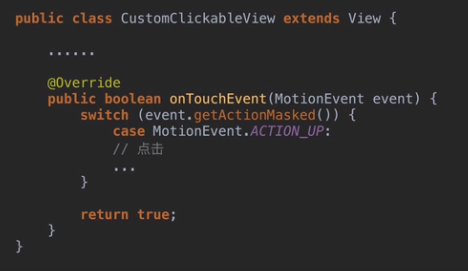
一系列事件不断产生，这个方法就会不断调用；从而对事件做相应的判断及反馈。

比如，点击button，会产生两个事件，按下事件Down，调用一次；然后抬起事件Up，调用一次；接着，控件就会判断自己被点击。如果设置了onClickListener的话，会去调用它的onClick()方法。

实际上，在这两个事件之间还会有move事件，用户的手指的轻微晃动，但是这些move事件被忽略了，不影响判断为点击事件。

1. 如果你想写自己的触摸算法

重写 onTouchEvent()，在里面写上你的触摸反馈算法，并返回 true（关键是 ACTION\_DOWN 事件时返回 true）。



1. 触摸事件分发，其实就是一个为解决触摸事件冲突而设置的机制。

当一个事件组的第一个事件down事件发生的时候，android会从触摸点上最近的那个view开始，一个一个去调用每个view的onTouchEvent()。

如果当前view的onTouchEvent()没有响应，就会继续向下，直到有第一个view做出响应，这个遍历的过程才会结束。这个时候，这个view就成了这组事件的接受者（TODO:是不是在这里做了存储操作，避免后续的遍历，也是考虑move事件，不然这么表示move呢？）。这个down事件的后续事件都会直接发送给它，直到这组事件结束，即后续的Up事件或者cancel事件出现。

这里的响应，就是这个方法onTouchEvent()的返回值。

在Down事件，这个返回值就表明是否响应了或者消费了这个事件。或者说，返回True表明view希望处理以这个down事件为起始点的这组事件流，你把后续的事件都交给我。

其实只有down事件的返回值需要是true，后续的事件的返回值是什么是没有影响的。

1. 然后，我们看看listView滑动事件是怎么做到的呢。

listview的上面还有item，显然是item先调用onTouchEvent()，那listview是怎么判断为滑动操作的呢？

这个依靠的是事件拦截机制。

就是在每一事件到达view的onTouchEvent()之前，有一个过程：从activity的根view一级一级去询问你要不要拦截这组事件？

拦截就是说，我不想交给子view处理，要自己来处理了。

调用 ViewGroup.onInterceptTouchEvent();

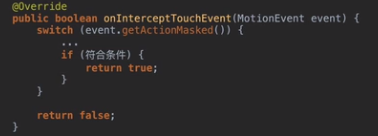
返回FALSE，不拦截，就会继续询问它的子viewGroup;

如果全都问了一遍，都返回false，就开始onTouchEvent()，从下往上。

如果这个view想拦截事件，就返回true。

onInterceptTouchEvent();这个方法可以整个事件流中，对每个事件进行监听（这点和onTouchEvent()是不同的）。你可以选择先观望，给子view一个处理事件的机会，一旦发展到了你的触发条件。比如，你发现用户是在滑动了，你返回true，立刻就可以实现对这个事件流的接管。值得注意的是，这个时候除了接管事件流，还会给它的子View发送一个额外的Cancel事件，这样子view就会恢复到初始状态，因为它不能处理后续的事件了。

如果是会发生触摸冲突的 ViewGroup，还需要重写 onInterceptTouchEvent()，在事件流开始时返回 false，并在确认接管事件流时返回一次 true，以实现对事件的拦截。



特殊地，子view有一个特殊的方法：当子 View 临时需要阻止父 View 拦截事件流时，可以调用父 View 的 requestDisallowInterceptTouchEvent() ，通知父 View 在当前事件流中不再尝试通过onInterceptTouchEvent() 来拦截。