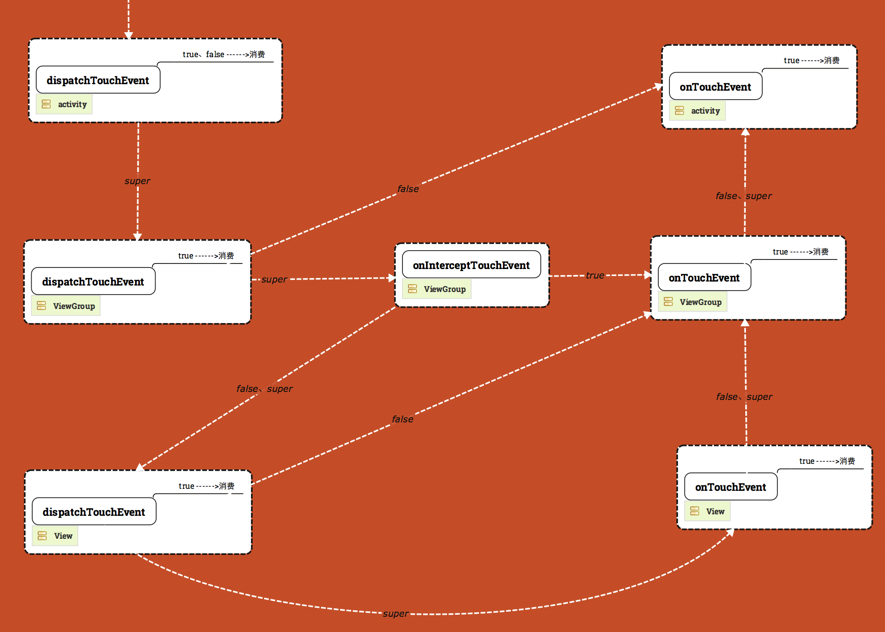
**android 事件分发机制**

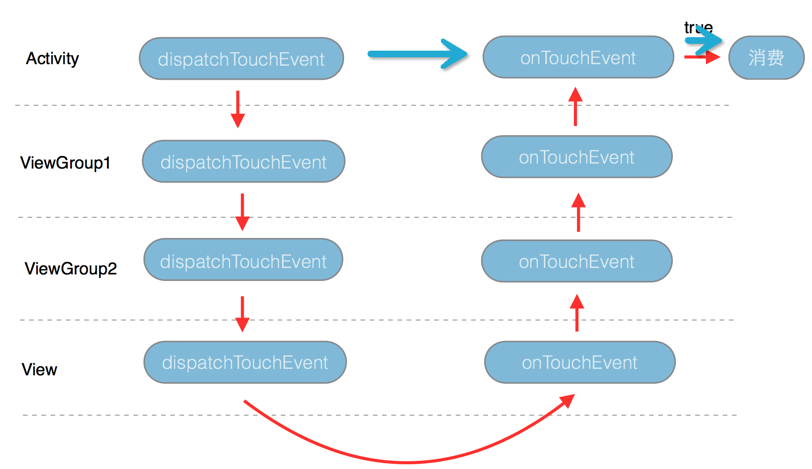
参考：<http://www.jianshu.com/p/e99b5e8bd67b>

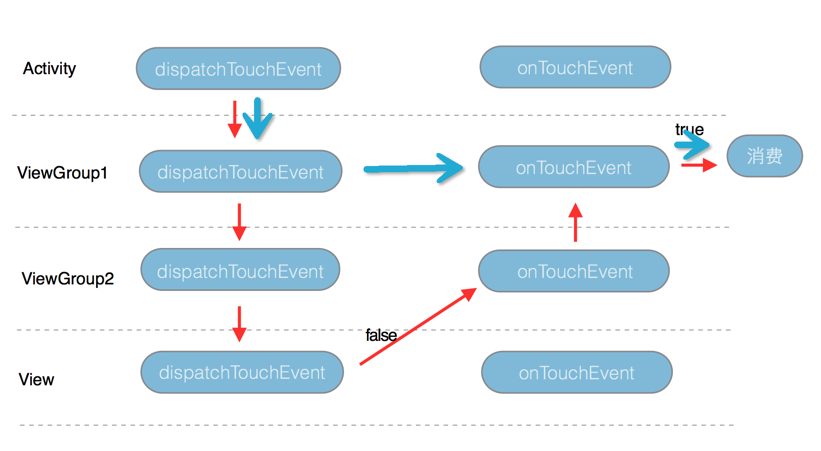
1. 事件分发流程图：



* + 1. 从上往下依次是Activity、ViewGroup、View；
    2. dispatchTouchEvent和onTouchEvent返回true时，都表示该事件已经消费；onTouchEvent返回false时，表示不消费事件，让事件从下往上向父控件传递。
    3. ViewGroup、View的dispatchTouchEvent返回false时，表示**事件停止往子View传递和分发同时开始往父控件回溯，回溯到上一层onTouchEvent**；Activity的dispatchTouchEvent返回false也表示该事件已被消费；
    4. onInterceptTouchEvent返回true，表示事件拦截，后续事件不往下传递；返回false和super. onInterceptTouchEvent表示事件不拦截，允许往下传递。

1. 并不是收到了ACTION\_DOWN事件的函数，就会收到 ACTION\_MOVE 等后续的事件。





**注**：红色箭头表示ACTION\_DOWN事件，蓝色箭头表示ACTION\_MOVE和ACTION\_UP事件。

1. 当ViewGroup子view消费了ACTION\_DOWN事件时，该ViewGroup的onInterceptTouchEvent将接收到ACTION\_MOVE和ACTION\_UP事件；当ViewGroup的onTouchEvent消费了ACTION\_DOWN事件时，ACTION\_MOVE和ACTION\_UP事件直接由该ViewGroup的dispatchTouchEvent事件分发至该ViewGroup的onTouchEvent，而不经过该ViewGroup的onInterceptTouchEvent。
2. onInterceptTouchEvent是ViewGroup提供的方法，用来判断是否拦截某个事件，如果当前View拦截了某个事件，那么在同一个事件序列当中，此方法不会被再次调用，返回结果表示是否拦截当前事件。默认返回false，返回true表示拦截。
3. 某个View一旦开始处理事件，如果它不消耗ACTION\_DOWN事件，那么同一事件序列的其他事情都不会再交给它来处理，并且事件将重新交给它的上一级去处理（调用父容器/activity的onTouchEvent方法）；如果它消耗ACTION\_DOWN事件，但是不消耗其他类型事件，那么这个点击事件会消失，父容器的onTouchEvent方法不会被调用，当前view依然可以收到后续的事件，但是这些事件最后都会传递给Activity处理。
4. 如果ViewGroup拦截了一个半路的事件（比如，MOVE），这个事件将会被系统变成一个CANCEL事件，并传递给之前处理该手势（gesture）的子View，而且不会再传递（无论是被拦截的MOVE还是系统生成的CANCEL）给ViewGroup的onTouchEvent方法。
5. onInterceptTouchEvent方法一旦返回一次true，就再也不会被调用了。
6. 事件分发流程：
7. activity的dispatchTouchEvent()将事件交给window；
8. window交给ViewGroup的dispatchTouchEvent()方法；
9. ViewGroup的dispatchTouchEvent()调用本身的onInterceptTouchEvent()方法；
10. onInterceptTouchEvent()返回false，遍历全部子view；找到能接收事件的子view，然后通过dispatchTransformedTouchEvent()方法将事件传给改子view的dispatchTouchEvent()。如果子view的dispatchTouchEvent()返回false，则通过dispatchTransformedTouchEvent()调用ViewGroup的dispatchTouchEvent()方法，从而执行onTouchEvent()方法（此时的ViewGroup变成普通的View，与View事件执行流程一致）。
11. onInterceptTouchEvent()返回true，通过dispatchTransformedTouchEvent()方法调用该ViewGroup的dispatchTouchEvent()方法，从而执行onTouchEvent()方法（此时的ViewGroup变成普通的View，与View事件执行流程一致）
12. 子view的dispatchTouchEvent()接收到事件后，判断View的OnTouchListener是否为空，当前View是否enable
13. 当前View不为enable， 或OnTouchListener为null，或OnTouchListener不为null 且OnTouchListener.onTouch()返回false时，执行View.onTouchEvent()方法。
14. 当前View为enable 、OnTouchListener不为null、OnTouchListener.onTouch()返回true，三个同时满足时，则不会执行View.onTouchEvent()方法；
15. View.onTouchEvent()中当事件为MotionEvent.ACTION\_UP时会调用performClick()，若OnClickListener不为null，则执行OnClickListener.onClick()方法。
16. 当View.dispatchTouchEvent在进行事件分发的时候，只有前一个action返回true（dispatchTouchEvent返回true或者onTouchEvent返回true），才会触发下一个action（也就是说dispatchTouchEvent返回true才会进行下一次action派发）。

if (onFilterTouchEventForSecurity(event)) {  
 if ((mViewFlags & *ENABLED\_MASK*) == *ENABLED* && handleScrollBarDragging(event)) {

result = true;

}

//noinspection SimplifiableIfStatement

ListenerInfo li = mListenerInfo;

if (li != null && li.mOnTouchListener != null  
 && (mViewFlags & *ENABLED\_MASK*) == *ENABLED* && li.mvvvv.onTouch(this, event)) {  
 result = true;

}

if (!result && onTouchEvent(event)) {

result = true;

}

}

if (!result && mInputEventConsistencyVerifier != null) {

// result为false时，忽略后续事件

mInputEventConsistencyVerifier.onUnhandledEvent(event, 0);

}

1. 子View调用getParent().requestDisallowInterceptTouchEvent(true)可以阻止ViewGroup对其MOVE或者UP事件进行拦截；  
   参考：<http://blog.csdn.net/lmj623565791/article/details/39102591>