Android不同层次的手势监听

## Activity级别的手势监听（以向右滑动返回上层界面为例），

Activity层手势监听的使用场景：一般用于当前页面中没有过多的手势需要处理的时候，至多存在点击事件。

* 注意事项 ：Activity层，dispatch可以抓取所有的事件，至于是否拦截等，一定要看看是否有必要，比如底层存在点击的Item，为了防止滑动过程中变色，可以适时地屏蔽事件
* 设定一个距离阈值mDistanceGat，用于标记手势是否有效。
* 防止手势的往回滑动，最好利用GestureDectetor来判断，如果存在往回滑动，则手势无效，使用方式如下：

mDetector = new GestureDetector(this, new GestureDetector.SimpleOnGestureListener() {

@Override

public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {

if (!slideReturnFlag && distanceX > 5) {

slideReturnFlag = true;

}

 public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {

                if (!slideReturnFlag && distanceX > 5) {

                    slideReturnFlag = true;

                }

                // 锁定方向

                if (getDistance(e2.getX() - e1.getX(), e2.getY() - e1.getY()) > mDistanceGate

                        && flagDirection == MotionDirection.NONE) {

                    if (Math.abs(distanceY) > Math.abs(distanceX)) {

                        flagDirection = MotionDirection.VERTICAL;

                    } else {

                        flagDirection = MotionDirection.HORIZION;

* }  }
* Activity层，dispatch可以抓取所有的事件，至于是否拦截等，一定要看看是否有必要

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | |  |   ，  5、有时候，下面的按钮，或者pressed的View，如果需要处理，可以主动下发Cancle事件，如何保证自定义事件的完整性，要注意处理Cancle事件，被Cancle的事件，Cancle与Up是否相同，比如Button，Cancle是不会当做完整的事件点击事件。  6、getDistance(e2.getX() - e1.getX(), e2.getY() - e1.getY()) > mDistanceGate如何确定滑动方向，必须朝向一个方向滑动，但是既然有了flagDirection == MotionDirection.NONE就不用其他的 flagDirection = MotionDirection.VERTICAL， MotionDirection.HORIZION，Activity应该不存在Cancle的事件。  7、down\_X、down\_Y 在dispatch的时候处理，  mDetector.onTouchEvent(event);也最好在GestureDectetor中处理。   |  | | --- | | public boolean dispatchTouchEvent(MotionEvent event) {          // Activity 监听          if (!getGestureInterfaceState()) {              return super.dispatchTouchEvent(event);          } else {              mDetector.onTouchEvent(event);              switch (event.getActionMasked()) {                  case MotionEvent.ACTION\_DOWN:                      down\_X=event.getX();                      super.dispatchTouchEvent(event); |   至于事件是不是往下派发呢？既然你已经用了Activity，就注意手势页面里面的手势尽量少，对于View变色，因为state改变，   |  | | --- | | 尽量移动一段距离， Math.abs(event.getX() - down\_X) > 10，这是为了兼容最基本的点击效果，那么在满足这个效果前 Move事件也要正常下发。                 case MotionEvent.ACTION\_MOVE:                      if (Math.abs(event.getX() - down\_X) > 10                              && flagDirection == MotionDirection.HORIZION) {                          MotionEvent e = MotionEvent.obtain(event.getEventTime(),                                  event.getEventTime(),                                  MotionEvent.ACTION\_CANCEL,                                  event.getX(),                                  event.getY(), 0);  构造Candle事件。                          super.dispatchTouchEvent(e);                      } |   Up事件的时候，Up事件是否下发呢？如果手势是无效的，就要下发，如果手势有效，要做其他处理，怎没必要下发，甚至说下发反而是错误的。   |  | | --- | | case MotionEvent.ACTION\_UP:                      if (mGestureListener != null && !slideReturnFlag                              && flagDirection == MotionDirection.HORIZION) {                          if (stateMotion == CurrentMotionState.SlideRight) {                              mGestureListener.onSlideRight();                          }                      } else {                          super.dispatchTouchEvent(event);                      }                      flagDirection = MotionDirection.NONE;                      stateMotion = CurrentMotionState.NONE;                      slideReturnFlag=false;                      break; | |

* SlideOffLinearLayoutByDispatch 容器级别的监听

|  |
| --- |
| * 注意事项：容器级别的监听可能在当前容器强制获取手势的焦点。至于如何获取焦点，自己编写onTouch事件，并且reture true，不过我们把处理放在dispatch里面，这样能够保证事件完全获取，因为如果底层消费了事件，onTouch是无法完整获取事件的，但是我们有足够的能力保证dispatch获取完整的事件。无论在本层onTouch消费，还是底层消费，dispatch是用于不会漏掉的。，对于手势的容器，最好用padding，而不采用Magin，为什么呢，因为Margin不在处理容器内部。 |

|  |
| --- |
| * 四要素的配合 * 1、getparent requstdisallowInt， 防止屏蔽 * 2、onTouch return true 消费 * 3、dispatch Move 释放requstdisallowInt  防止过度屏蔽 * 4、GestureDectetor     判断，获取手势信息 |

|  |
| --- |
| Override      public boolean onTouchEvent(MotionEvent event) {          return true;      }      @Override      public boolean dispatchTouchEvent(MotionEvent ev) {          getParent().requestDisallowInterceptTouchEvent(true);          mGestureDetector.onTouchEvent(ev);          switch (ev.getActionMasked()) {              case MotionEvent.ACTION\_DOWN:                  down\_X = ev.getX();                  down\_Y = ev.getY();                  slideReturnFlag = false;                  break;              case MotionEvent.ACTION\_CANCEL:              case MotionEvent.ACTION\_UP:                  if (Math.abs(down\_X - ev.getX()) > Math.abs(down\_Y - ev.getY()) && !slideReturnFlag                          && ev.getX() - down\_X > mDistanceGate) {                      // 返回上个Activity，也有可能是返回上一个Fragment                      FragmentActivity mContext = null;                      if (getContext() instanceof FragmentActivity) {                          mContext = (FragmentActivity)getContext();                          FragmentManager fm = mContext.getSupportFragmentManager();                          if (fm.getBackStackEntryCount() > 0) {                              fm.popBackStack();                          } else {                              mContext.finish();                          }                      }                  }                  break;              case MotionEvent.ACTION\_MOVE:                  if (Math.abs(down\_X - ev.getX()) < Math.abs(down\_Y - ev.getY())                          && Math.abs(ev.getY() - down\_Y) > mDistanceGate / 2) {                      getParent().requestDisallowInterceptTouchEvent(false);                  }              default:                  break;          }          return super.dispatchTouchEvent(ev);      } |

* 最简单的容器监听，只是把滑动时间处理，至于事件如何派发，不关心，主要用于底层没有触摸事件的容器，那么就采用容器的dispatch就行了，如果要兼容MOVE，则注意要合理屏蔽。拦截也要滑动一段距离后拦截。

|  |  |  |
| --- | --- | --- |
| 1、回滑监听  2、注意 在dispatch的时候记录位置   |  | | --- | | case MotionEvent.ACTION\_DOWN:                  down\_X = ev.getX();                  down\_Y = ev.getY(); |   处理方式：dispatch不要屏蔽，注意发给底层，但是，如何保证dispatchTouchEvent发过来呢？，注意onTouch与requestDisallowInterceptTouchEvent的配合使用。   |  | | --- | | @Override      public boolean dispatchTouchEvent(MotionEvent ev) {          mGestureDetector.onTouchEvent(ev);          switch (ev.getActionMasked()) {              case MotionEvent.ACTION\_DOWN:                  down\_X = ev.getX();                  down\_Y = ev.getY();                  slideReturnFlag = false;                  break;              case MotionEvent.ACTION\_CANCEL:              case MotionEvent.ACTION\_UP:                  if (Math.abs(down\_X - ev.getX()) > Math.abs(down\_Y - ev.getY())  && !slideReturnFlag  && ev.getX() - down\_X > mDistanceGate) {                      // 返回上个Activity，也有可能是返回上一个Fragment                      FragmentActivity mContext = null;                      if (getContext() instanceof FragmentActivity) {                          mContext = (FragmentActivity)getContext();                          FragmentManager fm =                                                     mContext.getSupportFragmentManager();                          if (fm.getBackStackEntryCount() > 0) {                              fm.popBackStack();                          } else {                              mContext.finish();                          } } }                  break;              default:                  break; }   return super.dispatchTouchEvent(ev); } |   只要dispatch返回true，事件就会一直下传，dispatch可以算作深度的度量，并且注意必须Down事件的时候，返回的的True才可以，并且如果下层不消费，本层的onTouch就一定会执行，无论消费与否 |

* SlideOffLinearLayout 容器级别的监听，合理的Intercept，注意Candle的处理，被其他View拦截时候，自己要处理，Cancle不要直接等于Up

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * 与上面的不同，如何处理拦截，主要是为了防止底层存在点击事件的拖动时变色的问题。  |  | | --- | | * @Override  public boolean onInterceptTouchEvent(MotionEvent ev) { * if (ev.getActionMasked() == MotionEvent.ACTION\_MOVE && Math.abs(down\_X - ev.getX()) > 20)             return true;         return super.onInterceptTouchEvent(ev);     } |  |  | | --- | | * Cancle的处理 | | * // ACTION\_CANCEL 嵌套如其他scrowView 可能屏蔽     @Override     public boolean onTouchEvent(MotionEvent ev) { * switch (ev.getActionMasked()) {             case MotionEvent.ACTION\_DOWN: * return true;             case MotionEvent.ACTION\_CANCEL:                 return true;             case MotionEvent.ACTION\_UP:                 if (Math.abs(down\_X - ev.getX()) > Math.abs(down\_Y - ev.getY()) && !slideReturnFlag                         && ev.getX() - down\_X > mDistanceGate) |  |  | | --- | |  | |

* 水平滑动的ScrollView

|  |  |
| --- | --- |
| * 1、有时候为了防止上层屏蔽事件，或者说，一旦接触我们自定义的控件，那么就不许屏蔽，不过也要  case MotionEvent.ACTION\_MOVE:在满足条件的时候，允许屏蔽，scrow的相互嵌套。  |  | | --- | | * @Override     public boolean dispatchTouchEvent(MotionEvent ev) { * getParent().requestDisallowInterceptTouchEvent(true);         mGestureDetector.onTouchEvent(ev); * switch (ev.getActionMasked()) {             case MotionEvent.ACTION\_DOWN:                 slideReturnFlag = false;                 down\_X = ev.getX();                 down\_Y = ev.getY();                 oldScrollX = getScrollX();                 break;             case MotionEvent.ACTION\_UP:                 if (Math.abs(down\_X - ev.getX()) > Math.abs(down\_Y - ev.getY())                         && ev.getX() - down\_X > mDistanceGate && !slideReturnFlag                         && oldScrollX == 0) {                     // 返回上个Activity，也有可能是返回上一个Fragment                     FragmentActivity mContext = null;                     if (getContext() instanceof FragmentActivity) {                         mContext = (FragmentActivity)getContext();                         FragmentManager fm = mContext.getSupportFragmentManager(); * if (fm.getBackStackEntryCount() > 0) {                             fm.popBackStack();                         } else {                             mContext.finish();                         }                     }                 }                 break;             case MotionEvent.ACTION\_MOVE:                 if (Math.abs(down\_X - ev.getX()) < Math.abs(down\_Y - ev.getY())                         && Math.abs(ev.getY() - down\_Y) > mDistanceGate / 2) {                     getParent().requestDisallowInterceptTouchEvent(false);                 }             default:                 break;         } * return super.dispatchTouchEvent(ev);     } | |

* 垂直滚动的ScrollView防止过早屏蔽，重写拦截函数onInterceptTouchEvent，

|  |
| --- |
| * /\*\*  \* @author lishang  \* 目的：防止ScrollView过早的屏蔽操作，导致滑动不好用  \*  \*/ public class ScrollViewWithoutInter extends ScrollView { * private float down\_X, down\_Y;          public ScrollViewWithoutInter(Context context) {         super(context);     } * public ScrollViewWithoutInter(Context context, AttributeSet attrs) {         super(context, attrs);     } * public ScrollViewWithoutInter(Context context, AttributeSet attrs, int defStyle) {         super(context, attrs, defStyle);     } * private void init(Context context) {     } * @Override     public boolean onTouchEvent(MotionEvent ev) {         return super.onTouchEvent(ev);     } * @Override     public boolean onInterceptTouchEvent(MotionEvent ev) {         if (Math.abs(ev.getY() - down\_Y) <  getResources().getDimensionPixelSize(R.dimen.slide\_gesture\_vertical\_gate)) {             super.onInterceptTouchEvent(ev);             return false;         }         return super.onInterceptTouchEvent(ev);     } * @Override     public boolean dispatchTouchEvent(MotionEvent ev) { * switch (ev.getAction()) {         case MotionEvent.ACTION\_DOWN:             down\_X = ev.getX();             down\_Y = ev.getY();             break;         case MotionEvent.ACTION\_CANCEL:             break;         case MotionEvent.ACTION\_UP:             break;         default:             break;         }         return super.dispatchTouchEvent(ev);     } * private float getDistance(float distanceX, float distanceY) { * return (float) Math.sqrt(distanceX \* distanceX + distanceY \* distanceY);     }} |

* getParent().requestDisallowInterceptTouchEvent(boolean flag)使用注意事项

|  |  |
| --- | --- |
| 1、及时关闭，及时打开  2、不是万能的，主要用来保证move不被屏蔽   |  | | --- | | @Override  public boolean dispatchTouchEvent(MotionEvent ev) {          getParent().requestDisallowInterceptTouchEvent(true);          boolean ret = super.dispatchTouchEvent(ev);          Log.v("lishang", "hello" + ret);          return ret;          如果这里返回false，那么事件还是传不下来。 | |

* 自定义ViewPager支持滑动关闭。

|  |  |
| --- | --- |
| 利用滑动的百分比，防止一点点往回滑动就关闭。   |  | | --- | | if (slideDirection == SlideDirection.RIGHT) {                      if (slideReturnFlag || getCurrentItem() != 0 || arg0.getX() - down\_X < mDistanceGate || mPercent > 0.01f)                          break;                  } else if (slideDirection == SlideDirection.LEFT) {                      if (getAdapter() != null) {                          if (slideReturnFlag||getCurrentItem() !=  getAdapter().getCount() - 1                        || down\_X - arg0.getX() < mDistanceGate || mPercent > 0.01f)                              break;                      }   } else {                      break; } | |