Android不同层次的手势监听

第一：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：

1、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，比如对于Down事件，不可以屏蔽，否则连点击事件都无法下发。

2、设定一个距离阈值mDistanceGat，用于标记手势是否有效。

3、如果底层存在点击Item，为了防止滑动过程中变色，可以适时地屏蔽触摸事件：构造Cancle事件主动下发，这是为了兼容最基本的点击效果，不过，满足点击的手势判定前， Move事件要正常下发。具体实现如下：

```
   @Override
    public boolean dispatchTouchEvent(MotionEvent event) {   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);
                        super.dispatchTouchEvent(e);
                    } else {
                        super.dispatchTouchEvent(event);//不符合条件正常下发
                  }
```

4、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 d
istanceY) {

                if (!slideReturnFlag && distanceX > 5) {
                    slideReturnFlag = true;
                }}
```

5、如何处理Up事件：dispatch是否往下派发。具体的做法是，根据手势是否有效，如果手势无效，那么Up肯定是需要往下派发的。如果有效，根据后续操作进行。因为有时候为了防止子View获取到不必要的点击事件。具体实现如下

```
 @Override
    public boolean dispatchTouchEvent(MotionEvent event) {  case MotionEvent.ACTION_UP:
                    if (mGestureListener != null && !slideReturnFlag
                            && flagDirection == MotionDirection.HORIZION) {
                        if (stateMotion == CurrentMotionState.SlideRight) {
                            mGestureListener.onSlideRight();
                        }
                    } else {  super.dispatchTouchEvent(event); //无效的手势 }
                    flagDirection = MotionDirection.NONE;
                    stateMotion = CurrentMotionState.NONE;
```

```
                    slideReturnFlag=false;
                    break;
```

6、在DisPatch中最好记录down_X、down_Y ，为了后面的处理与判断，因为Dispatch中最能保证你获取到该事件。同时要保证Dispatch事件的下发，

第二：父容器级别的手势监听，

注意事项：容器级别的监听至少要使得当前容器强制获取手势的焦点，至于如何获取焦点，可以自己编写onTouch事件，并且reture true。不过我们把判断处理放在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。

1、父容器监听的使用场景

- 容器中，子View是否存在交互事件，是否存在滑动
- 上层容器是否存在拦截事件的可能，比如SrollView

2、实现

- 子View不存在交互事件：

这类容器可以采用Dispatch来实现，不过需要强制获取焦点，同时也要适时的释放焦点。具体实现如下：

如何保证本层一定接收到Down后续事件。dispatch的Down事件能够返回True即可。

如何保证本层不被偶然的屏蔽，使用 getParent().requestDisallowInterceptTouchEvent(true);即可，当然，有强制获取也要适时的释放，当手势判定为无效的时候就要释放，具体实现如下：

```
 @Override
    public boolean dispatchTouchEvent(MotionEvent ev) {  getParent().requestDisallowIntercep
tTouchEvent(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_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);
    }
```

- 子View存在交互事件：子View存在交互事件，就要通过dispatch与onTouch的配合使用，dispatch为了判断手势的有效性，同时既然从容器层开始，强制获取焦点是必须的，底层如何强制获取焦点，不关心。这里如果没有消费Down，则说明底层View消费了。同时要兼容无效手势强制焦点获取的释放，防止上传滚动View，具体实现如下：

```
@Override
   public boolean dispatchTouchEvent(MotionEvent ev) {

       mGestureDetector.onTouchEvent(ev);


       switch (ev.getActionMasked()) {
getParent().requestDisallowInterceptTouchEvent(true);


case MotionEvent.ACTION_DOWN:
               down_X = ev.getX();
               down_Y = ev.getY();
               slideReturnFlag = false;
               break;
           default:
               break;
       }
       return  super.dispatchTouchEvent(ev);
   }
   // 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()) && !slideRet
urnFlag
                       && 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();
                       }
                   }
               }
               return true;
           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);
               }
               return true;
           default:
               break;
       }
```

```
        return super.onTouchEvent(ev);
    }
```

3、父容器手势的拦截，有些时候，子View具有点击事件，点击变颜色。给予一定看空间，之后，强制拦截事件。拦截后如何，dispatch返回true保证事件下传，不必担心

```
 @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);
    }
```

第四：HorizontalScrollView边缘状态下，滑动手势的监听，具体实现如下，主要是边缘处的手势判断。

```
 @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);
    }
```

第五：防止垂直滚动的ScrollView过早的屏蔽事件：重写拦截函数即可：

```
 @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {
        if (Math.abs(ev.getY() - down_Y) < getResources().getDimensionPixelSize(R.dimen.slid
e_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;
```

第六：边缘Viewpager 的滑动手势；

1、防止过早拦击

```
 @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_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);
                }
                break;
            default:
```

```
                break;
        }

        return super.dispatchTouchEvent(ev);
    }
```

2、防止往回滑动等

```
 /*
     * 触摸事件的处理，要判断是否是ViewPager不可滑动的时候
     */
    @Override
    public boolean onTouchEvent(MotionEvent arg0) {

        // 防止跳动
        boolean ret = super.onTouchEvent(arg0);

        switch (arg0.getActionMasked()) {
            case MotionEvent.ACTION_DOWN:
                Log.v("lishang", "down");
                break;
            case MotionEvent.ACTION_CANCEL:
            case MotionEvent.ACTION_UP:

                Log.v("lishang", "up");
                if (slideDirection == SlideDirection.RIGHT) {

                    if (slideReturnFlag || getCurrentItem() != 0 || arg0.getX() - down_X < m
DistanceGate || 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 {}
```

第七：getParent().requestDisallowInterceptTouchEvent

这个函数的的作用仅仅能够保证事件不被屏蔽，但是倘若本层dispatch在down的时候返回false，那么事件的处理就无效了，就算强制获取焦点

TIPs：

- 使用dispatch 还是使用onTouch，关键看看需要考虑底层么？如果底层可以屏蔽上层的那么久采用哦onTouch，因为底层处理了，上层不用处理了。
- 如果拦截Move，说明本身的Down已经在Dispatch的时候返回true了，所以不存在手势派发不到的问题，既然Down返回True了，不论拦截的Move返回什么，事件可以保证持续下发
-