


滑动

版本：2018/05/12-1(11:11)

 滑动-思维导图

- 滑动
 - 基础知识
 - View
 - 坐标系
 - MotionEvent
 - 滑动的7种实现方法
 - 弹性滑动
 - Scroller
 - 动画
 - 延时策略
 - 侧滑菜单
 - DrawerLayout
 - SlidingPanelLayout
 - NavigationView
 - ViewDragHelper
 - ViewDragHelper.Callback
 - 扩展实例
 - QQ侧滑菜单
 - GestureDetector
 - OnGestureListener
 - OnDoubleTapListener
 - OnContextClickListener
 - SimpleOnGestureListener
 - 辅助类
 - ViewConfiguration
 - VelocityTracker
 - 参考资料

基础知识

View

1、什么是View

1. View是所有控件的基类
2. View有一个特殊子类ViewGroup， ViewGroup能包含一组View， 但ViewGroup的本身也是View。
3. 由于View和ViewGourp的存在， 意味着View可以是单个控件也可以是一组控件。这种结构形成了View树。

2、View的位置参数： top,left,right,bottom

1. top-左上角的y轴坐标(全部是相对坐标， 相对于父容器)
2. left-左上角的x轴坐标
3. right-右下角的x轴坐标
4. bottom-右下角的y轴坐标
5. 在View中获取这些成员变量的方法， 是getLeft(),getRight(),getTop(),getBottom()即可

3、View从3.0开始新增的参数： x,y,translationX,translationY

1. x,y是View当前左上角的坐标
2. translationX,translationY是在滑动/动画后， View当前位置和View最原始位置的距离。
3. 因此得出等式： $x(\text{View左上角当前位置}) = \text{left}(\text{View左上角初始位置}) + \text{translationX}(\text{View左上角偏移的距离})$
4. View平移时top、 left等参数不变， 改变的是x,y,translationX和translationY

坐标系

4、Android坐标系

1. Android坐标系以 屏幕左上角 为原点，向右X轴为正半轴，向下Y轴为正半轴
2. 触摸事件中getRawX()和getRawY()获得的就是Android坐标系的坐标
3. Android中通过 getLocationOnScreen(int location[]) 能获得当前视图的左上

5、View坐标系

1. View坐标系是以当前视图的 父视图的左上角 作为原点建立的坐标系，方向和Android坐标系一致
2. 触摸事件中getX()和getY()获得的就是视图坐标系中的坐标

MotionEvent

6、MotionEvent的作用

1. MotionEvent 用于 记录移动事件
2. 包括鼠标、手机、traceball、pen的移动事件。

7、MotionEvent包含的手指触摸事件

1. ACTION_DOWN\MOVE\UP对应三个触摸事件。
2. getX()/getY()能获得触摸点的坐标，相当于当前View左上角的(x,y)
3. getRawX()/getRawY(), 获得触摸点相当于手机左上角的(x,y)坐标

滑动的7种实现方法

8、View滑动的7种方法：

1. layout: 对View进行重新布局定位。在onTouchEvent()方法中获得控件滑动前后的偏移。然后通过layout方法重新设置。
2. offsetLeftAndRight和offsetTopAndBottom:系统提供上下/左右同时偏移的API。onTouchEvent()中调用
3. LayoutParams: 更改自身布局参数
4. scrollTo/scrollBy: 本质是移动View的内容，需要通过父容器的该方法来滑动当前View
5. Scroller: 平滑滑动，通过重载 computeScroll()，使用 scrollTo/scrollBy 完成滑动效果。
6. 属性动画: 动画对View进行滑动
7. ViewDragHelper: 谷歌提供的辅助类，用于完成各种拖拽效果。

15、Layout实现滑动

```
/*=====*
 * onTouchEvent-进行偏移计算，之后调用layout
 *=====*/
public boolean onTouchEvent(MotionEvent event) {
    float curX = event.getX(); //手指实时位置的X
    float curY = event.getY(); //Y
    switch(event.getAction()){
        case MotionEvent.ACTION_MOVE:
            int offsetX = (int)(curX - downX); //X偏移
            int offsetY = (int)(curY - downY); //Y偏移
            break;
        case MotionEvent.ACTION_DOWN:
            downX = curX; //按下时的坐标
            downY = curY;
            break;
    }
    return true;
}
```

16、offsetLeftAndRight和offsetTopAndBottom实现滑动

```

/*=====*
 * onTouchEvent-进行偏移计算，直接调用
 *=====*/
public boolean onTouchEvent(MotionEvent event) {
    float curX = event.getX(); //手指实时位置的X
    float curY = event.getY(); //Y
    switch(event.getAction()){
        case MotionEvent.ACTION_MOVE:
            int offsetX = (int)(curX - downX); //X偏移
            int offsetY = (int)(curY - downY); //Y偏移
            /**=====
            * 对left和right, top和bottom同时偏移
            *=====*/
            offsetLeftAndRight(offsetX);
            offsetTopAndBottom(offsetY);
            break;
        case MotionEvent.ACTION_DOWN:
            downX = curX; //按下时的坐标
            downY = curY;
            break;
    }
    return true;
}

```

17、LayoutParams实现滑动：

1. 通过父控件设置View在父控件的位置，但需要指定父布局的类型，不好
2. 用ViewGroup的MarginLayoutParams的方法去设置margin

```

//方法一：通过布局设置在父控件的位置。但是必须要有父控件，而且要指定父布局的类型，不好的方法。
RelativeLayout.LayoutParams layoutParams = (RelativeLayout.LayoutParams) getLayoutParams();
layoutParams.leftMargin = getLeft() + offsetX;
layoutParams.topMargin = getTop() + offsetY;
setLayoutParams(layoutParams);

/*=====
 * 方法二：用ViewGroup的MarginLayoutParams的方法去设置margin
 * 优点：相比于上面方法，就不需要知道父布局的类型。
 * 缺点：滑动到右侧控件会缩小
 *=====*/
ViewGroup.MarginLayoutParams mLayoutParams = (ViewGroup.MarginLayoutParams) getLayoutParams();
mLayoutParams.leftMargin = getLeft() + offsetX;
mLayoutParams.topMargin = getTop() + offsetY;
setLayoutParams(mLayoutParams);

```

18、scrollTo\scrollBy实现滑动

1. 都是View提供的方法。
2. scrollTo-直接到新的x,y坐标处。
3. scrollBy-基于当前位置的相对滑动。
4. scrollBy-内部是调用scrollTo。
5. scrollTo\scrollBy，效果是移动View的内容，因此需要在View的父控件中调用。

```

// 1、移动到目标位置
((View)getParent()).scrollTo(dstX, dstY);
// 2、相对滑动：且scrollBy是父容器进行滑动，因此偏移量需要取负
((View)getParent()).scrollBy(-offsetX, -offsetY);

```

19、scrollTo/By内部的mScrollX和mScrollY的意义

1. mScrollX的值，相当于手机屏幕相对于View左边缘向右移动的距离，手机屏幕向右移动时，mScrollX的值为正；手机屏幕向左移动(等价于View向右移动)，mScrollX的值为负。
2. mScrollY和X的情况相似，手机屏幕向下移动，mScrollY为+正值；手机屏幕向上移动，mScrollY为-负值。
3. mScrollX/Y是根据第一次滑动前的位置来获得的，例如：第一次向左滑动200(等于手机屏幕向右滑动200)，mScrollX = 200；第二次向右滑动50，mScrollX = 200 + (-50) = 150，而不是 (-50)。

20、动画实现滑动的方法

1. 可以通过传统动画或者属性动画的方式实现
2. 传统动画需要通过设置fillAfter为true来保留动画后的状态(但是无法在动画后的位置进行点击操作，这方面还是属性动画好)

3. 属性动画会保留动画后的状态，能够点击。

21、ViewDragHelper

1. 通过 ViewDragHelper 去自定义 ViewGroup 让其子View 具有滑动效果。

弹性滑动

Scroller

1、Scroller的作用

1. 用于 封装滑动
2. 提供了 基于时间的滑动偏移值，但是实际滑动需要我们去负责。

1、Scroller的要点

1. 调用startScroll方法时， Scroller只是单纯的保存参数
2. 之后的invalidate方法导致的View重绘
3. View重绘之后draw方法会调用自己实现的computeScroll()，才真正实现了滑动

1、Scroller的使用

```
// 1、初始化
Scroller mScroller = new Scroller(getContext());

// 2、重写View的方法computeScroll
public void computeScroll() {
    super.computeScroll();
    //判断scroller是否执行完毕。
    if(mScroller.computeScrollOffset()){
        ((View)getParent()).scrollTo(mScroller.getCurrX(), mScroller.getCurrY());
        //通过重绘来不断调用 computeScroll
        invalidate();
    }
}

// 3、开始滑动
case MotionEvent.ACTION_UP:
    View viewGroup = (View) getParent();
    mScroller.startScroll(viewGroup.getScrollX(), viewGroup.getScrollY(),
        -viewGroup.getScrollX(), -viewGroup.getScrollY());
    invalidate();
    break;
```

1、Scroller工作原理

1. Scroller本身不能实现View的滑动，需要配合View的computeScroll方法实现弹性滑动
2. 不断让View重绘，每一次重绘距离滑动的开始时间有一个时间间隔，通过该时间可以得到View当前的滑动距离
3. View的每次重绘都会导致View的小幅滑动，多次小幅滑动就组成了弹性滑动

动画

4、通过动画实现弹性滑动

延时策略

5、通过延时策略实现弹性滑动。

1. 通过handler、View的postDelayed、或者线程的sleep方法。
2. 实现思路：例如将View滑动100像素，通过Handler可以每100ms发送一次消息让其滑动10像素，最终会在1000ms内滑动100像素。

侧滑菜单

DrawerLayout

1、DrawerLayout是什么？

1. Google 推出的 侧滑菜单 。

2、DrawerLayout的使用

1. 侧滑菜单 的布局需要用 `layout_gravity` 属性指定。
2. 主体View 的布局中 宽高 需要为 `match_parent` 且 不能有`layout_gravity`属性

//布局文件

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    android:id="@+id/md_drawerlayout"
    xxx>
    <Button
        android:id="@+id/md_slidemenu_text"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        xxx
        android:layout_gravity="start"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        xxx主体xxx
    </LinearLayout>
</android.support.v4.widget.DrawerLayout>
```

```
DrawerLayout drawerLayout = findViewById(R.id.md_drawerlayout);
Button button = findViewById(R.id.md_slidemenu_text);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        drawerLayout.closeDrawer(button); //关闭侧滑菜单
    }
});
```

3、DrawerLayout中 android:layout_gravity 属性

1. left/start : 菜单位于左侧
2. top/bottom : 菜单位于右侧

4、DrawerLayout的方法

1-打开

```
drawerLayout.openDrawer(button);
```

2-关闭

```
drawerLayout.closeDrawer(button);
```

3-设置监听器(DrawerListener)

```

drawerLayout.setDrawerListener(new DrawerLayout.DrawerListener() {
    //滑动时
    @Override
    public void onDrawerSlide(View drawerView, float slideOffset) {

    }
    //打开时
    @Override
    public void onDrawerOpened(View drawerView) {

    }
    //关闭时
    @Override
    public void onDrawerClosed(View drawerView) {

    }
    //状态改变时: {@link #STATE_IDLE}, {@link #STATE_DRAGGING} or {@link #STATE_SETTLING}.
    @Override
    public void onDrawerStateChanged(int newState) {

    }
});

```

4-设置监听器(SimpleDrawerListener)

```

//可以选择性实现其中的部分回调接口
drawerLayout.setDrawerListener(new DrawerLayout.SimpleDrawerListener() {
    @Override
    public void onDrawerSlide(View drawerView, float slideOffset) {
        super.onDrawerSlide(drawerView, slideOffset);
    }
});

```

SlidingPanelLayout

1、SlidingPanelLayout是什么

1. 提供一种类似于 DrawerLayout 的侧滑菜单效果，“效果并不好”
2. xml 布局中第一个 ChildView 就是 左侧菜单的内容，第二个 ChildView 就是 主体内容

2、SlidingPanelLayout的使用

```

<android.support.v4.widget.SlidingPanelLayout
    android:id="@+id/md_slidingpanelayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/md_slidemenu_text"
        android:layout_width="150dp"
        android:layout_height="match_parent"
        xxx左侧内容xxx/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        xxx主体内容xxx
    </LinearLayout>

</android.support.v4.widget.SlidingPanelLayout>

```

```

SlidingPanelLayout slidingPanelLayout = findViewById(R.id.md_slidingpanelayout);
Button button = findViewById(R.id.md_slidemenu_text);

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //关闭Pane
        slidingPanelLayout.closePane();
    }
});

```

3、SlidingPanelLayout的方法

```
// 1. 打开Pane
slidingPaneLayout.openPane();
// 2. 关闭Pane
slidingPaneLayout.closePane();
// 3. 右侧主体页面缩进去的阴影渐变色
slidingPaneLayout.setSliderFadeColor(Color.BLUE);
// 4. 左侧面板缩进去的阴影渐变色
slidingPaneLayout.setCoveredFadeColor(Color.GRAY);

// 5. 监听器
slidingPaneLayout.setPanelSlideListener(new SlidingPaneLayout.PanelSlideListener() {
    /**
     * 左侧面板在滑动
     * @param panel 被移走的主体View
     * @param slideOffset 滑动的百分比(0~1)
     */
    @Override
    public void onPanelSlide(View panel, float slideOffset) {

    }
    //左侧Pane已经打开
    @Override
    public void onPanelOpened(View panel) {

    }
    //左侧Pane已经关闭
    @Override
    public void onPanelClosed(View panel) {

    }
});
```

NavigationView

NavigationView的作用

1. 配合 DrawerLayout 使用用于实现其中的 左侧菜单效果
2. Google在5.0之后推出NavigationView,
3. 左侧菜单效果 整体上分为两部分, 上面一部分叫做 HeaderLayout , 下面的那些点击项都是 menu 。

ViewDragHelper

1、ViewDragHelper的作用

1. 用于 编写自定义ViewGroup 的 工具类
2. 位于 android.support.v4.widget. 。
3. 提供一系列 操作和状态追踪 用于帮助用户进行 拖拽和定位子View

2、ViewDragHelper的简单实例

实现ChildView可以自由拖拽的ViewGroup

1. 创建 ViewDragHelper
2. 将 ViewGroup 的事件处理交给 ViewDragHelper
3. 自定义 ViewDragHelper.Callback 实现一些触摸回调, 用于实现效果。

```

public class ScrollViewGroup extends LinearLayout{
    private ViewDragHelper mViewDragHelper;

    public ScrollViewGroup(Context context, @Nullable AttributeSet attrs) {
        super(context, attrs);
        /**=====
        * 1、创建ViewDragHelper
        *=====*/
        mViewDragHelper = ViewDragHelper.create(
            this, //ViewGroup
            1f, //设置touchSlop-sensitivity越大,touchslop越小
            new MyViewDragHelperCallback()); //用户触摸事件的回调
    }

    /**=====
    * 2、ViewGroup的事件处理都交给ViewDragHelper
    *=====*/
    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {\
        //转交中断处理权
        return mViewDragHelper.shouldInterceptTouchEvent(ev);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        //交由处理事件，且返回true表示处理后续事件。
        mViewDragHelper.processTouchEvent(event);
        return true;
    }

    /**=====
    * 3、ViewDragHelper.Callback
    *=====*/
    class MyViewDragHelperCallback extends ViewDragHelper.Callback{
        /**
        * 3.1-决定哪些View可以捕获
        * @return true-捕获该child; false-不处理
        */
        @Override
        public boolean tryCaptureView(View child, int pointerId) {
            return true;
        }
        /**
        * 3.2-控制Child在水平方向上的边界
        * @return 范围限制后的新left(当前child的left)
        */
        @Override
        public int clampViewPositionHorizontal(View child, int left, int dx) {
            // left范围为 leftPadding ~ (getWidth() - getPaddingRight() - child.getWidth())
            final int leftMinBound = getPaddingLeft();
            final int leftMaxBound = getWidth() - getPaddingRight() - child.getWidth();
            final int newLeft = Math.min(Math.max(left, leftMinBound), leftMaxBound);
            return newLeft;
        }
        /**
        * 3.3-控制Child在垂直方向上的边界
        * @return 范围限制后的新top
        */
        @Override
        public int clampViewPositionVertical(View child, int top, int dy) {
            final int topMinBound = getPaddingTop();
            final int topMaxBound = getHeight() - getPaddingBottom() - child.getHeight();
            final int newLeft = Math.min(Math.max(top, topMinBound), topMaxBound);
            return newLeft;
        }
    }
}

```

3、ChildView为Button或者 clickable = true 时无法拖动的解决办法

1. 正常流程: 如果子View不消耗事件，那么整个手势（DOWN-MOVE-UP）都是直接进入onTouchEvent，在onTouchEvent的DOWN的时候就确定了captureView。
2. 子View消耗事件: 会先走onInterceptTouchEvent方法，判断是否可以捕获，而在判断的过程中会去判断另外两个回调的方法：getViewHorizontalDragRange和getViewVerticalDragRange，只有这两个方法返回大于0的值才能正常的捕获。


```
/**
 * 返回子View水平滑动范围。
 * return 0: 则该ChildView不会滑动。
 */
@Override
public int getViewHorizontalDragRange(View child)
{
    return getMeasuredWidth()-child.getMeasuredWidth();
}

/**
 * 返回子View垂直滑动范围。
 * return 0: 则该ChildView不会滑动。
 */
@Override
public int getViewVerticalDragRange(View child)
{
    return getMeasuredHeight()-child.getMeasuredHeight();
}
```

ViewDragHelper.Callback

1、ViewDragHelper.Callback的方法和作用

方法	作用
onViewDragStateChanged()	当ViewDragHelper状态发生变化时回调（IDLE，DRAGGING，SETTING-自动滚动时）
onViewPositionChanged()	ChildView位置改变时回调
onViewCaptured()	捕获ChildView时回调
onViewReleased()	松开ChildView时回调
onEdgeTouched()	当触摸到边界时回调
onEdgeLock()	true的时候会锁住当前的边界，false则unLock。
onEdgeDragStarted()	边缘拖拽开始时回调
getOrderedChildIndex()	在同一个坐标（x,y）下应该去获取哪一个View。（mViewDragHelper.findTopChildUnder中需要用到）
getViewHorizontalDragRange()	获取水平方向上的拖拽范围
getViewVerticalDragRange()	获取垂直方向上的拖拽范围
tryCaptureView()	判断是否捕获当前View
clampViewPositionHorizontal()	控制Child在水平方向上的边界
clampViewPositionVertical()	控制Child在垂直方向上的边界

```

/**
 * ChildView不在被拖拽的时候调用。
 *
 * 1. 想要将ChildView 安置到某个位置，需要调用{@link ViewDragHelper#settleCapturedViewAt(int, int)}
 * 2. 想要将ChildView fling到某个位置，需要调用{@link ViewDragHelper#flingCapturedView(int, int, int, int)}
 *
 * 注意：
 * 1. 如果调用这些方法，ViewDragHelper会进入{@link ViewDragHelper#STATE_SETTLING}模式，
 *    此时直到View完全停止，View的捕获都不会停止。
 * 2. 如果不调用这些方法，View会停止且ViewDragHelper会处于{@link ViewDragHelper#STATE_IDLE}模式
 *
 * {@link View#computeScroll()}
 *
 * @param xvel 手指离开屏幕时-X轴速度(像素/秒)
 * @param yvel 手指离开屏幕时-Y轴速度(像素/秒)
 */
@Override
public void onViewReleased(View releasedChild, float xvel, float yvel) {
    super.onViewReleased(releasedChild, xvel, yvel);

    if (releasedChild == mAutoBackView){
        mViewDragHelper.settleCapturedViewAt(mAuthoBackOriginPoint.x, mAuthoBackOriginPoint.y);
        invalidate();
    }
}

/**
 * 触摸到边缘
 */
@Override
public void onEdgeTouched(int edgeFlags, int pointerId) {}

/**
 * true的时候会锁住当前的边界，false则unlock。
 */
@Override
public boolean onEdgeLock(int edgeFlags) {
    return false;
}

/**
 * 边缘拖动的时候回调。能绕过“tryCaptureView”
 *
 * @param edgeFlags A combination of edge flags describing the edge(s) dragged
 * @param pointerId ID of the pointer touching the described edge(s)
 * @see #EDGE_LEFT
 * @see #EDGE_TOP
 * @see #EDGE_RIGHT
 * @see #EDGE_BOTTOM
 */
@Override
public void onEdgeDragStarted(int edgeFlags, int pointerId) {
    super.onEdgeDragStarted(edgeFlags, pointerId);
    //主动通过captureChildView进行捕获
    mViewDragHelper.captureChildView(mEdgeDragView, pointerId);
}

/**
 * 决定ChildView的Z轴上的顺序。(mViewDragHelper.findTopChildUnder(x, y)中需要获取到坐标(x,y)上最上层的子View)
 *
 * @param index 查询的调用位置
 * @return 在调用位置上View的index
 */
@Override
public int getOrderedChildIndex(int index) {
    return index;
}

/**
 * 返回子View水平滑动范围。
 * return 0: 则该ChildView不会滑动。
 */
@Override
public int getViewHorizontalDragRange(View child)
{
    return getMeasuredWidth()-child.getMeasuredWidth();
}

```

```

/**
 * 返回子View垂直滑动范围。
 * return 0: 则该ChildView不会滑动。
 */
@Override
public int getViewVerticalDragRange(View child)
{
    return getMeasuredHeight()-child.getMeasuredHeight();
}

/**
 * 决定哪些View可以捕获
 * @return true-捕获该child; false-不处理
 */
@Override
public boolean tryCaptureView(View child, int pointerId) {
    if(child == mEdgeDragView) return false;
    return true;
}

/**
 * 控制Child在水平方向上的边界
 * @return 范围限制后的新left(当前child的left)
 */
@Override
public int clampViewPositionHorizontal(View child, int left, int dx) {
    // left范围为 leftPadding ~ (getWidth() - getPaddingRight() - child.getWidth())
    final int leftMinBound = getPaddingLeft();
    final int leftMaxBound = getWidth() - getPaddingRight() - child.getWidth();
    final int newLeft = Math.min(Math.max(left, leftMinBound), leftMaxBound);
    return newLeft;
}

/**
 * 控制Child在垂直方向上的边界
 * @return 范围限制后的新top
 */
@Override
public int clampViewPositionVertical(View child, int top, int dy) {
    final int topMinBound = getPaddingTop();
    final int topMaxBound = getHeight() - getPaddingBottom() - child.getHeight();
    final int newLeft = Math.min(Math.max(top, topMinBound), topMaxBound);
    return newLeft;
}

@Override
public void onViewDragStateChanged(int state) {}

@Override
public void onViewPositionChanged(View changedView, int left, int top, int dx, int dy) {}

@Override
public void onViewCaptured(View capturedChild, int activePointerId) {}

```

2、shouldInterceptTouchEvent中方法回调顺序

DOWN:

```

getOrderedChildIndex(findTopChildUnder)
->onEdgeTouched

```

MOVE:

```

getOrderedChildIndex(findTopChildUnder)
->getViewHorizontalDragRange & getViewVerticalDragRange(checkTouchSlop)(MOVE中可能不止一次)
->clampViewPositionHorizontal & clampViewPositionVertical
->onEdgeDragStarted
->tryCaptureView
->onViewCaptured
->onViewDragStateChanged

```

3、processTouchEvent中方法回调顺序

```

DOWN:
    getOrderedChildIndex(findTopChildUnder)
    ->tryCaptureView
    ->onViewCaptured
    ->onViewDragStateChanged
    ->onEdgeTouched

MOVE:
    ->STATE==DRAGGING:dragTo
    ->STATE!=DRAGGING:
        onEdgeDragStarted
        ->getOrderedChildIndex(findTopChildUnder)
        ->getViewHorizontalDragRange & getViewVerticalDragRange(checkTouchSlop)
        ->tryCaptureView
        ->onViewCaptured
        ->onViewDragStateChanged

```

扩展实例

1、ViewDragHelper实例：拖拽返回、边缘拖拽

2、ViewGroup如何去获取子控件

```

View mNormalView;
View mAutoBackView;
View mEdgeDragView;

@Override
protected void onFinishInflate() {
    super.onFinishInflate();

    mNormalView = getChildAt(0);
    mAutoBackView = getChildAt(1);
    mEdgeDragView = getChildAt(2);
}

```

3、ViewGroup如何去获取某ChildView的初始坐标

```

Point mAuthoBackOriginPoint = new Point();

@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
    super.onLayout(changed, l, t, r, b);

    mAuthoBackOriginPoint.x = mAutoBackView.getLeft();
    mAuthoBackOriginPoint.y = mAutoBackView.getTop();
}

```

4、ViewGroup如何进行拖拽返回

1-Callback的 onViewReleased

```

@Override
public void onViewReleased(View releasedChild, float xvel, float yvel) {
    super.onViewReleased(releasedChild, xvel, yvel);
    //1、改变位置
    if (releasedChild == mAutoBackView){
        mViewDragHelper.settleCapturedViewAt(mAuthoBackOriginPoint.x, mAuthoBackOriginPoint.y);
        invalidate();
    }
}

```

2-内部是 mScroller.startScroll 因此需要 computeScroll 配合

```

@Override
public void computeScroll()
{
    if(mViewDragHelper.continueSettling(true))
    {
        invalidate();
    }
}

```

5、ViewDragHelper的边缘拖动

```
/**
 * 屏蔽"目标控件"的滑动效果
 */
@Override
public boolean tryCaptureView(View child, int pointerId) {
    if(child == mEdgeDragView) return false;
    return true;
}
/**
 * 边缘拖动的时候回调。能绕过“tryCaptureView”
 */
@Override
public void onEdgeDragStarted(int edgeFlags, int pointerId) {
    super.onEdgeDragStarted(edgeFlags, pointerId);
    //主动通过captureChildView进行捕获
    mViewDragHelper.captureChildView(mEdgeDragView, pointerId);
}

// 设置边缘追踪
mViewDragHelper.setEdgeTrackingEnabled(ViewDragHelper.EDGE_LEFT | ViewDragHelper.EDGE_TOP);
```

QQ侧滑菜单

```

public class DragViewGroup extends FrameLayout {

    //侧滑类
    private ViewDragHelper mViewDragHelper;
    private View mMenuView,mMainView;
    private int mWidth;

    public DragViewGroup(Context context) {
        super(context);
        initView();
    }

    public DragViewGroup(Context context, AttributeSet attrs) {
        super(context, attrs);
        initView();
    }

    public DragViewGroup(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        initView();
    }

    /**-----
     * 1、初始化数据：调用ViewDragHelper.create方法
     * -----*/
    private void initView() {
        mViewDragHelper = ViewDragHelper.create(this,callback); //需要监听的View和回调callback
    }

    /**-----
     * 2、事件拦截和触摸事件全部交给ViewDragHelper进行处理
     * -----*/
    //事件拦截
    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {

        return mViewDragHelper.shouldInterceptTouchEvent(ev);
    }
    //触摸事件
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        //将触摸事件传递给ViewDragHelper

        mViewDragHelper.processTouchEvent(event);

        return true;
    }

    /**-----
     * 3、也需要重写computeScroll()
     * 内部也是通过scroller来进行平移滑动，这个模板可以照搬
     * -----*/
    @Override
    public void computeScroll() {
        if(mViewDragHelper.continueSettling(true)){
            ViewCompat.postInvalidateOnAnimation(this);
        }
    }

    /**-----
     * 4、处理的回调：侧滑回调
     * -----*/
    private ViewDragHelper.Callback callback = new ViewDragHelper.Callback() {

        /**-----
         * 何时开始触摸：
         * 1.指定哪一个子View可以被移动。
         * 2.如果直接返回true，在该布局之内的所有子View都可以随意划动
         * -----*/
        @Override
        public boolean tryCaptureView(View child, int pointerId) {
            //如果当前触摸的child是mMainView开始检测
            return mMainView == child;
        }

        /**-----

```

```

* 处理水平滑动：
* 1. 返回值默认为0，如果为0则不处理该方向的滑动。
* 2. 一般直接返回left，当需要精准计算padding等值时，可以先对left处理再返回
* -----*/
@Override
public int clampViewPositionHorizontal(View child, int left, int dx) {
    return left;
}

/*-----
* 处理垂直滑动：
* 1. 返回值默认为0，如果为0则不处理该方向的滑动。
* 2. 一般直接返回top，，当需要精准计算padding等值时，可以先对left处理再返回
* -----*/
@Override
public int clampViewPositionVertical(View child, int top, int dy) {
    return 0;
}

/*-----
* 拖动结束后调用，类似ACTION_UP。
* 这里是实现侧滑菜单，一般滑动可以不用这段代码
* -----*/
@Override
public void onViewReleased(View releasedChild, float xvel, float yvel) {
    super.onViewReleased(releasedChild, xvel, yvel);
    //手指抬起后缓慢的移动到指定位置
    if(mMainView.getLeft() < 500){
        //关闭菜单
        mViewDragHelper.smoothSlideViewTo(mMainView,0,0);
        ViewCompat.postInvalidateOnAnimation(DragViewGroup.this);
    }else{
        //打开菜单
        mViewDragHelper.smoothSlideViewTo(mMainView,300,0);
        ViewCompat.postInvalidateOnAnimation(DragViewGroup.this);
    }
}
};

/**-----
* 5、获取子控件用于处理
* 1. 上面完成了滑动功能，这里简单的按照第1、2的顺序指定子控件View的内容
* 2. onSizeChanged能够获得menu等子控件的宽度等信息，有需求可以后续处理
* -----*/
//XML加载组建后回调
@Override
protected void onFinishInflate() {
    super.onFinishInflate();
    mMenuView = getChildAt(0);
    mMainView = getChildAt(1);
}

//组件大小改变时回调
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    super.onSizeChanged(w, h, oldw, oldh);
    mWidth = mMenuView.getMeasuredWidth();
}
}

```

使用(作为父控件，里面依次放menu和main):

```
<com.example.xxxx.DragViewGroup
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@color/colorAccent"/>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@color/colorPrimary"/>

</com.example.xxxx.DragViewGroup>
```

GestureDetector

1、GestureDetector作用和注意点

1. 探测 手势 和 事件，需要通过提供的 MotionEvent
2. 该类仅能用于 touch触摸 提供的 MotionEvent，不能用于 traceball events(追踪球事件)
3. 自定义View 中可以重写 onTouchEvent() 方法并在里面用 GestureDetector 接管。

OnGestureListener

2、OnGestureListener作用

1. 用于在 手势 产生时，去通知监听者。
2. 该 监听器 会监听所有的手势，如果只需要监听一部分可以使用 SimpleOnGestureListener

3、OnGestureListener能监听哪些手势(5种)?


```

public interface OnGestureListener {
    /**
     * 1、按下操作。且其他任何事件之前都会触发该方法。
     * @param e Down MotionEvent
     */
    boolean onDown(MotionEvent e);

    /**
     * 2、按下之后，Move和Up之前。用于提供视觉反馈告诉用户已经捕获了他们的行为。
     * @param e Down MotionEvent
     */
    void onShowPress(MotionEvent e);

    /**
     * 2、抬起操作。
     * @param e Up MotionEvent
     */
    boolean onSingleTapUp(MotionEvent e);

    /**
     * 3、滑动操作(由Down MotionEvent e1触发，当前是Move MotionEvent e2)
     *
     * @param e1 开启滑动的按下操作。
     * @param e2 触发onScroll的滑动操作。
     * @param distanceX 最近一次onScroll和当前onScroll之间的X滑动距离。
     * @param distanceY 最近一次onScroll和当前onScroll之间的Y滑动距离。
     */
    boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY);

    /**
     * 4、长按操作。
     *
     * @param e 开始长按的Down操作。
     */
    void onLongPress(MotionEvent e);

    /**
     * 5、猛扔操作。
     *
     * @param e1 开始fling操作的Down MotionEvent
     * @param e2 触发onFling的Move MotionEvent
     * @param velocityX X轴的速度(pixels / second 像素/每秒)
     * @param velocityY Y轴的速度(pixels / second 像素/每秒)
     */
    boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY);
}

```

上面所有有返回值的方法，return true -消耗该事件；return false -不消耗该事件

4、OnGestureListener的使用方法。

```

/**=====
 * 1、GestureDetector通过context和onGestureListener构造
 *=====*/
GestureDetector gestureDetector = new GestureDetector(getApplicationContext(), new GestureDetector.OnGestureListener() {
    //实现5种回调方法
});
/**=====
 * 2、在View的touch方法中进行拦截。
 *=====*/
imageView.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        //交给GestureDetector进行处理
        return gestureDetector.onTouchEvent(event);
    }
});

```

OnDoubleTapListener

5、OnDoubleTapListener作用

1. 监听“双击操作”
2. 监听“确认的单击操作”---该单击操作之后的操作无法构成一次双击。

6、OnDoubleTapListener能监听哪些手势(3种)?

```
/**
 * 双击或者单击(后续操作无法导致双击)
 */
public interface OnDoubleTapListener {
    /**
     * 1、单击操作。 不会产生双击行为的单击操作才会触发。
     *
     * @param e Down MotionEvent
     * @return true-消耗事件; false-不消耗事件。
     */
    boolean onSingleTapConfirmed(MotionEvent e);

    /**
     * 2、双击操作。
     *
     * @param e 双击操作的第一个按下操作。
     * @return true-消耗事件; false-不消耗事件。
     */
    boolean onDoubleTap(MotionEvent e);

    /**
     * 3、双击操作之间发生了down、move或者up事件。
     *
     * @param e 双击操作期间产生的MotionEvent
     * @return true-消耗事件; false-不消耗事件。
     */
    boolean onDoubleTapEvent(MotionEvent e);
}
```

7、OnDoubleTapListener的使用方法

```
GestureDetector gestureDetector = new GestureDetector(...);
gestureDetector.setOnDoubleTapListener(new GestureDetector.OnDoubleTapListener() {
    // 三种回调方法
});
// 在View的touch方法中进行拦截。
imageView.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        //交给GestureDetector进行处理
        return gestureDetector.onTouchEvent(event);
    }
});
```

OnContextClickListener

8、OnContextClickListener的作用

- 1. 鼠标/触摸板的右击操作

9、OnContextClickListener的方法

```
/**=====
 * 鼠标/触摸板 右键点击
 * 1. 需要确保在View的onGenericMotionEvent中进行拦截
 * 2. 最终交给GestureDetector的onGenericMotionEvent方
 *=====*/
public interface OnContextClickListener {
    boolean onContextClick(MotionEvent e);
}
```

10、OnContextClickListener的使用

需要在View的

```
// 1、设置OnContextClickListener监听器
GestureDetector gestureDetector = new GestureDetector(...);
gestureDetector.setOnContextClickListener(new GestureDetector.OnContextClickListener() {...});

// 2、拦截View的onGenericMotion方法
imageView.setOnGenericMotionListener(new View.OnGenericMotionListener() {
    @Override
    public boolean onGenericMotion(View v, MotionEvent event) {
        return gestureDetector.onGenericMotionEvent(event);
    }
});
```

SimpleOnGestureListener

11、SimpleOnGestureListener的作用

实现了 GestureDetector 的所有监听器，可以选择性实现需要的方法。

12、SimpleOnGestureListener的使用

```
/**=====
 * 可以选择性实现需要的方法
 *=====*/
class MyGestureListener extends GestureDetector.SimpleOnGestureListener{
    @Override
    public boolean onDown(MotionEvent e) {
        return super.onDown(e);
    }
}

// 1、设置OnContextClickListener监听器
GestureDetector gestureDetector = new GestureDetector(getApplicationContext(),
    new MyGestureListener());

// 2、拦截View的Touch方法
imageView.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        return gestureDetector.onTouchEvent(event);
    }
});
```

辅助类

ViewConfiguration

ViewConfiguration的作用

- 1. 定义所有 UI 所需要用的标准常量。
- 2. 包括双击时间间隔、滑动最小距离等等。
- 3. 获取常量需要通过类的静态方法或者成员方法获得。
- 4. 静态方法：与设备无关
- 5. 成员方法：与设备有关

ViewConfiguration的使用方法

```
//类的静态方法
ViewConfiguration.getDoubleTapTimeout(); //构成双击的时间间隔

//类的成员方法
ViewConfiguration configuration = ViewConfiguration.get(getBaseContext());
configuration.getScaledTouchSlop(); //滑动的最小距离
```

ViewConfiguration常量汇总

常量	介绍	作用	类方法or成员方法
----	----	----	-----------

常量	介绍	作用	类方法or成员方法
configuration.getScaledTouchSlop()	滑动的最小距离，低于该值则认为没有滑动。	在两次滑动距离小于该值时可以判断未滑动，以提高用户体验。	成员方法 (该值与设备有关)
configuration.hasPermanentMenuKey()	设备是否具有实体按键(返回按键等)。		成员方法 (该值与设备有关)
ViewConfiguration.getKeyRepeatTimeout()	重复按键的间隔时间。	两次按键小于该事件则表示属于同一次按键	类方法 (该值与设备无关)

VelocityTracker

6、VelocityTracker的作用

- 1. 速度追踪：手指滑动中水平和竖直方向的速度
- 2. 速度是指：在给定时间内手机滑过的像素数，如果从右到左，就是负值(例如1000ms内速度为100，就是在1s内滑过100个像素)
- 3. 使用完毕时需要调用 clear 和 recycle 方法进行清理并回收内存

VelocityTracker的使用

- 1. 在View的onTouchEvent中追踪当前点击事情的速度
- 2. 通过VelocityTracker的computeCurrentVelocity方法先计算速度
- 3. 再获取VelocityTracker的xVelocity/yVelocity获取速度

7、VelocityTracker代码如下

```
//追踪速度
val velocityTracker = VelocityTracker.obtain()
velocityTracker.addMovement(event)

//获取当前速度，但必须在获取前进行速度计算
velocityTracker.computeCurrentVelocity(1000) //时间单位
val xVelocity = velocityTracker.xVelocity
val yVelocity = velocityTracker.yVelocity

velocityTracker.clear()
velocityTracker.recycle()
```

参考资料

- 1. [Android ViewDragHelper完全解析 自定义ViewGroup神器](#)
- 2.