RecycleView滑动监听

获取RecyclerView滑动时显示的Item的相关信息

1.可显示de第一个item位置

2.可完全显示de第一个item位置

3.可完全显示de第一个item位置

4.可显示de最后item位置

```
mRecyclerView.addOnScrollListener(new RecyclerView.OnScrollListener()
{
    @Override
    public void onScrolled(final RecyclerView recyclerView, final int dx, final int dy)
    {
        //可显示de第一个item位置
        int FirstVisibleItemPosition = ((LinearLayoutManager) recyclerView.getLayoutManager()).findFirstVisibleItemPosition();
        Log.i("zw", "FirstVisibleItemPosition：" + FirstVisibleItemPosition + "位置");

        //可完全显示de第一个item位置
        int FirstCompletelyVisibleItemPosition = ((LinearLayoutManager) recyclerView.getLayoutManager()).findFirstCompletelyVisibleItemPosition();
        Log.i("zw", "FirstCompletelyVisibleItemPosition：" + FirstCompletelyVisibleItemPosition + "位置");

        //可完全显示de第一个item位置
        int LastCompletelyVisibleItemPosition = ((LinearLayoutManager) recyclerView.getLayoutManager()).findLastCompletelyVisibleItemPosition();
        Log.i("zw", "LastCompletelyVisibleItemPosition：" + LastCompletelyVisibleItemPosition + "位置");

        //可显示de最后item位置
        int LastVisibleItemPosition = ((LinearLayoutManager) recyclerView.getLayoutManager()).findLastVisibleItemPosition();
        Log.i("zw", "LastVisibleItemPosition：" + LastVisibleItemPosition + "位置");

    }
});
```

滑动监听的工具类RecycleViewScrollHelper

```
package com.taro.headerrecycle.helper;

import android.support.annotation.IntRange;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.View;

/**
 * Created by taro on 16/5/10.
 */
public class RecycleViewScrollHelper extends RecyclerView.OnScrollListener {
    private RecyclerView mRvScroll = null;
    private OnScrollDirectionChangedListener mScrollDirectionChangedListener = null;
    //滑动位置变动的监听事件
    private OnScrollPositionChangedListener mScrollPositionChangedListener = null;
    //是否同时检测滑动到顶部及底部
    private boolean mIsCheckTopBottomTogether = false;
    //检测滑动顶部/底部的优先顺序,默认先检测滑动到底部
    private boolean mIsCheckTopFirstBottomAfter = false;
    //检测底部滑动时是否检测满屏状态
    private boolean mIsCheckBottomFullRecycle = false;
    //检测顶部滑动时是否检测满屏状态
    private boolean mIsCheckTopFullRecycle = false;
    //顶部满屏检测时允许的容差值
    private int mTopOffsetFaultTolerance = 0;
    //底部满屏检测时允许的容差值
    private int mBottomOffsetFaultTolerance = 0;


    private int mScrollDx = 0;
    private int mScrollDy = 0;

    /**
     * recycleView的滑动监听事件,用于检测是否滑动到顶部或者滑动到底部.
     *
     * @param listener {@link OnScrollPositionChangedListener}滑动位置变动监听事件
```

```
     */
    public RecycleViewScrollHelper(OnScrollPositionChangedListener listener) {
        mScrollPositionChangedListener = listener;
    }

    @Override
    public void onScrollStateChanged(RecyclerView recyclerView,
                                     int newState) {
        if (mScrollPositionChangedListener == null || recyclerView.getAdapter() == null || recyclerView.getChildCount() <= 0) {
            return;
        }
        RecyclerView.LayoutManager layoutManager = recyclerView.getLayoutManager();
        if (layoutManager instanceof LinearLayoutManager) {
            LinearLayoutManager linearManager = (LinearLayoutManager) layoutManager;
            int lastItemPosition = linearManager.findLastVisibleItemPosition();
            int firstItemPosition = linearManager.findFirstVisibleItemPosition();
            RecyclerView.Adapter adapter = recyclerView.getAdapter();
            if (newState == RecyclerView.SCROLL_STATE_IDLE) {
                //判断顶部/底部检测的优先顺序
                if (!mIsCheckTopFirstBottomAfter) {
                    //先检测底部
                    if (this.checkIfScrollToBottom(recyclerView, lastItemPosition, adapter.getItemCount())) {
                        //若检测滑动到底部时,判断是否需要同时检测滑动到顶部
                        if (mIsCheckTopBottomTogether) {
                            //检测是否滑动到顶部
                            this.checkIfScrollToTop(recyclerView, firstItemPosition);
                            //不管是否滑动到顶部,已经触发了滑动到底部,所以直接返回,否则会调用滑动到未知位置的
                            return;
                        } else {
                            //若不需要同时检测,直接返回
                            return;
                        }
                    } else if (this.checkIfScrollToTop(recyclerView, firstItemPosition)) {
                        //当未检测滑动到底部时,再检测是否滑动到顶部
                        return;
                    }
                } else {
                    //先检测是否滑动到顶部
                    if (this.checkIfScrollToTop(recyclerView, firstItemPosition)) {
                        if (mIsCheckTopBottomTogether) {
                            //检测是否滑动到底部
                            this.checkIfScrollToBottom(recyclerView, lastItemPosition, adapter.getItemCount());
                            return;
                        } else {
                            //若不需要同时检测,直接返回
                            return;
                        }
                    } else if (this.checkIfScrollToBottom(recyclerView, lastItemPosition, adapter.getItemCount())) {
                        //当未检测滑动到底部时,再检测是否滑动到底部
                        return;
                    }
                }
            }
        }
        //其它任何情况
        mScrollPositionChangedListener.onScrollToUnknown(false, false);
    }

    /**
     * 检测是否滑动到了顶部item并回调事件
     *
     * @param recyclerView
     * @param firstItemPosition 第一个可见itemView的position
     * @return
     */
    private boolean checkIfScrollToTop(RecyclerView recyclerView, int firstItemPosition) {
        if (firstItemPosition == 0) {
            if (mIsCheckTopFullRecycle) {
                int childCount = recyclerView.getChildCount();
                View firstChild = recyclerView.getChildAt(0);
                View lastChild = recyclerView.getChildAt(childCount - 1);
                int top = firstChild.getTop();
                int bottom = lastChild.getBottom();
                //recycleView显示itemView的有效区域的top坐标Y
                int topEdge = recyclerView.getPaddingTop() - mTopOffsetFaultTolerance;
```

```
            //recycleView显示itemView的有效区域的bottom坐标Y
            int bottomEdge = recyclerView.getHeight() - recyclerView.getPaddingBottom() - mBottomOffsetFaultTolerance;
            //第一个view的顶部大于top边界值,说明第一个view已经完全显示在顶部
            //同时最后一个view的底部应该小于bottom边界值,说明最后一个view的底部已经超出显示范围,部分或者完全移出了界面
            if (top >= topEdge && bottom > bottomEdge) {
                mScrollPositionChangedListener.onScrollToTop();
                return true;
            } else {
                mScrollPositionChangedListener.onScrollToUnknown(true, false);
            }
        } else {
            mScrollPositionChangedListener.onScrollToTop();
            return true;
        }
    }
    return false;
}

/**
 * 检测是否滑动到底部item并回调事件
 *
 * @param recyclerView
 * @param lastItemPosition 最后一个可见itemView的position
 * @param itemCount        adapter的itemCount
 * @return
 */
private boolean checkIfScrollToBottom(RecyclerView recyclerView, int lastItemPosition, int itemCount) {
    if (lastItemPosition + 1 == itemCount) {
        //是否进行满屏的判断处理
        //未满屏的情况下将永远不会被回调滑动到低部或者顶部
        if (mIsCheckBottomFullRecycle) {
            int childCount = recyclerView.getChildCount();
            //获取最后一个childView
            View lastChildView = recyclerView.getChildAt(childCount - 1);
            //获取第一个childView
            View firstChildView = recyclerView.getChildAt(0);
            int top = firstChildView.getTop();
            int bottom = lastChildView.getBottom();
            //recycleView显示itemView的有效区域的bottom坐标Y
            int bottomEdge = recyclerView.getHeight() - recyclerView.getPaddingBottom() + mBottomOffsetFaultTolerance;
            //recycleView显示itemView的有效区域的top坐标Y
            int topEdge = recyclerView.getPaddingTop() + mTopOffsetFaultTolerance;
            //第一个view的顶部小于top边界值,说明第一个view已经部分或者完全移出了界面
            //最后一个view的底部小于bottom边界值,说明最后一个view已经完全显示在界面
            //若不处理这种情况,可能会存在recycleView高度足够高时,itemView数量很少无法填充一屏,但是滑动到最后一项时依然会发生回调
            //此时其实并不需要任何刷新操作的
            if (bottom <= bottomEdge && top < topEdge) {
                mScrollPositionChangedListener.onScrollToBottom();
                return true;
            } else {
                mScrollPositionChangedListener.onScrollToUnknown(false, true);
            }
        } else {
            mScrollPositionChangedListener.onScrollToBottom();
            return true;
        }
    }
    return false;
}

@Override
public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
    if (mScrollDirectionChangedListener != null) {
        if (dx == 0 && dy == 0) {
            mScrollDirectionChangedListener.onScrollDirectionChanged(0, 0);
        } else if (dx == 0) {
            boolean isUp = dy > 0;
            boolean isBeenUp = mScrollDy > 0;
            if (isUp != isBeenUp) {
                mScrollDx = dx;
                mScrollDy = dy;
                mScrollDirectionChangedListener.onScrollDirectionChanged(dx, dy);
            }
        } else if (dy == 0) {
            boolean isLeft = dx > 0;
```

```
            boolean isBeenLeft = mScrollDx > 0;
            if (isLeft != isBeenLeft) {
                mScrollDx = dx;
                mScrollDy = dy;
                mScrollDirectionChangedListener.onScrollDirectionChanged(dx, dy);
            }
        }
    }
}

//重置数据
private void reset() {
    mScrollDx = 0;
    mScrollDy = 0;
}

/**
 * 关联recycleView,当关联新的recycleView时,会自动移除上一个关联recycleView
 *
 * @param recyclerView
 */
public void attachToRecycleView(RecyclerView recyclerView) {
    if (recyclerView != mRvScroll) {
        unAttachToRecycleView();
        mRvScroll = recyclerView;
        if (recyclerView != null) {
            recyclerView.addOnScrollListener(this);
        }
    }
}

/**
 * 移除与recycleView的绑定
 */
public void unAttachToRecycleView() {
    if (mRvScroll != null) {
        mRvScroll.removeOnScrollListener(this);
    }
    this.reset();
}

/**
 * 设置滑动方向改变时的回调接口
 *
 * @param listener
 */
public void setScrollDirectionChangedListener(OnScrollDirectionChangedListener listener) {
    mScrollDirectionChangedListener = listener;
}

/**
 * 设置顶部允许偏移的容差值,此值仅在允许检测满屏时有效,当{@link #setCheckIfItemViewFullRecycleViewForTop(boolean)}设置为true 或者{@link
#setCheckIfItemViewFullRecycleViewForBottom(boolean)}设置为true 时有效.<br>
 * 在检测底部滑动时,对顶部的检测会添加此容差值(更容易判断当前第一项childView已超出recycleView的显示范围),用于协助判断是否滑动到底部.
 * 在检测顶部滑动时,对顶部的检测会添加此容差值(更容易判断为滑动到了顶部)
 *
 * @param offset 容差值,此值必须为0或正数
 */
public void setTopOffsetFaultTolerance(@IntRange(from = 0) int offset) {
    mTopOffsetFaultTolerance = offset;
}

/**
 * 设置顶部允许偏移的容差值,此值仅在允许检测满屏时有效,当{@link #setCheckIfItemViewFullRecycleViewForTop(boolean)}设置为true 或者{@link
#setCheckIfItemViewFullRecycleViewForBottom(boolean)}设置为true 时有效.<br>
 * 在检测底部滑动时,对底部的检测会添加此容差值(更容易判断当前最后一项childView已超出recycleView的显示范围),用于协助判断是否滑动到顶部.
 * 在检测顶部滑动时,对底部的检测会添加此容差值(更容易判断为滑动到了底部)
 *
 * @param offset 容差值,此值必须为0或正数
 */
public void setBottomFaultTolerance(@IntRange(from = 0) int offset) {
    mBottomOffsetFaultTolerance = offset;
}
```

```
/**
 * 设置是否需要检测recycleView是否为满屏的itemView时才回调事件.<br>
 * <p>
 * 当RecycleView的childView数量很少时,有可能RecycleView已经显示出所有的itemView,此时不存在向上滑动的可能.<br>
 * 若设置当前值为true时,只有在RecycleView无法完全显示所有的itemView时,才会回调滑动到顶部的事件;否则将不处理;<br>
 * 若设置为false则反之,不管任何时候只要滑动并顶部item显示时都会回调滑动事件
 *
 * @param isNeedToCheck true为当检测是否满屏显示;false不检测,直接回调事件
 */
public void setCheckIfItemViewFullRecycleViewForTop(boolean isNeedToCheck) {
    mIsCheckTopFullRecycle = isNeedToCheck;
}

/**
 * 设置是否需要检测recycleView是否为满屏的itemView时才回调事件.<br>
 * <p>
 * 当RecycleView的childView数量很少时,有可能RecycleView已经显示o出所有的itemView,此时不存在向下滑动的可能.
 * 若设置当前值为true时,只有在RecycleView无法完全显示所有的itemView时,才会回调滑动到底部的事件;否则将不处理;
 * 若设置为false则反之,不管任何时候只要滑动到底部都会回调滑动事件
 *
 * @param isNeedToCheck true为当检测是否满屏显示;false不检测,直接回调事件
 */
public void setCheckIfItemViewFullRecycleViewForBottom(boolean isNeedToCheck) {
    mIsCheckBottomFullRecycle = isNeedToCheck;
}

/**
 * 设置是否先检测滑动到哪里.默认为false,先检测滑动到底部
 *
 * @param isTopFirst true为先检测滑动到顶部再检测滑动到底部;false为先检测滑动到底部再滑动到顶部
 */
public void setCheckScrollToTopFirstBottomAfter(boolean isTopFirst) {
    mIsCheckTopFirstBottomAfter = isTopFirst;
}

/**
 * 设置是否同时检测滑动到顶部及底部,默认为false,先检测到任何一个状态都会直接返回,不会再继续检测其它状态
 *
 * @param isCheckTogether true为两种状态都检测,即使已经检测到其中某种状态了.false为先检测到任何一种状态时将不再检测另一种状态
 */
public void setCheckScrollToTopBottomTogether(boolean isCheckTogether) {
    mIsCheckTopBottomTogether = isCheckTogether;
}

/**
 * 滑动位置改变监听事件,滑动到顶部,底部或者非以上两个位置时
 */
public interface OnScrollPositionChangedListener {
    /**
     * 滑动到顶部的回调事件
     */
    public void onScrollToTop();

    /**
     * 滑动到底部的回调事件
     */
    public void onScrollToBottom();

    /**
     * 滑动到未知位置的回调事件
     *
     * @param isTopViewVisible    当前位置顶部第一个itemView是否可见,这里是指adapter中的最后一个itemView
     * @param isBottomViewVisible 当前位置底部最后一个itemView是否可见,这里是指adapter中的最后一个itemView
     */
    public void onScrollToUnknown(boolean isTopViewVisible, boolean isBottomViewVisible);
}

/**
 * 滑动方向改变时监听事件
 */
public interface OnScrollDirectionChangedListener {
    /**
     * 滑动方向改变时监听事件,当两个参数值都为0时,数据变动重新layout
     *
```

```
        *@param scrollVertical    竖直方向的滑动方向,向上&lt;0,向下&gt;0,不动(水平滑动时)=0
        *@param scrollHorizontal  水平方向的滑动方向,向左&lt;0,向右&gt;0,不动(竖直滑动时)=0
        */
        public void onScrollDirectionChanged(int scrollHorizontal, int scrollVertical);
    }
}
```