# **RecyclerView使用进阶**

目前的项目中,基本已经使用 RecyclerView 全面替换了ListView,GridView. 使用RecyclerView确实更加灵活,功能也更加强大. RecyclerView的基本套路应该都很熟悉了,这里整理一下一些相对进阶一点的知识点,方便随时复习.

###### **分割线**

虽然和ListView比较, RecyclerView 设置分割线麻烦了很多, 不过也更自由了,可以实现更多的效果.

RecyclerView 默认是没有分割线的,需要通过下面这个方法添加

public void addItemDecoration(ItemDecoration decor) {

addItemDecoration(decor, -1);

}

那么 ItemDecoration 又是什么东西? ItemDecoration是 RecyclerView 的一个内部抽象类,很明显,这个东西是给我们实现的. 当我们实现 ItemDecoration 的时候,只需要关注 3 个方法,说起来麻烦,直接看代码和注释.

public class ItemDivider extends RecyclerView.ItemDecoration {

// 构造方法,可以在这里做一些初始化,比如指定画笔颜色什么的

public ItemDivider() {

}

/\*\*

\* 指定item之间的间距(就是指定分割线的宽度) 回调顺序 1

\* @param outRect Rect to receive the output.

\* @param view The child view to decorate

\* @param parent RecyclerView this ItemDecoration is decorating

\* @param state The current state of RecyclerView.

\*/

@Override

public void getItemOffsets(Rect outRect, View view, RecyclerView parent, RecyclerView.State state) {

super.getItemOffsets(outRect, view, parent, state);

}

/\*\*

\* 在item 绘制之前调用(就是绘制在 item 的底层) 回调顺序 2

\* 一般分割线在这里绘制

\* 看到canvas,对自定义控件有一定了解的话,就能想到为什么说给RecyclerView设置分割线更灵活了

\* @param c Canvas to draw into

\* @param parent RecyclerView this ItemDecoration is drawing into

\* @param state The current state of RecyclerView

\*/

@Override

public void onDraw(Canvas c, RecyclerView parent, RecyclerView.State state) {

super.onDraw(c, parent, state);

}

/\*\*

\* 在item 绘制之后调用(就是绘制在 item 的上层) 回调顺序 3

\* 也可以在这里绘制分割线,和上面的方法 二选一

\* @param c Canvas to draw into

\* @param parent RecyclerView this ItemDecoration is drawing into

\* @param state The current state of RecyclerView

\*/

@Override

public void onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state) {

super.onDrawOver(c, parent, state);

}

}

* getItemOffsets 指定item 之间的间距(默认为0),将来就是在这个间距内绘制分割线
* onDraw 在绘制 item之前执行,也就是说,在这里绘制的图形可能会被item遮盖(所以需要指定item之间的间距)
* onDrawOver 在绘制item之后执行,在这里绘制的图形,可能会遮住item(说以如果要在这里绘制分割线的话,也要找准位置)

PS:在 RecyclerView 25.0.0中,终于有了官方实现的分割线-DividerItemDecoration,可惜只支持 LinearLayoutManager ,感兴趣的可以试试.

下面是我自己的实现,适配 LinearLayoutManager 和 GridLayoutManager

public class ItemDivider extends RecyclerView.ItemDecoration {

private int dividerWith = 1;

private Paint paint;

private RecyclerView.LayoutManager layoutManager;

// 构造方法,可以在这里做一些初始化,比如指定画笔颜色什么的

public ItemDivider() {

initPaint();

paint.setColor(0xffff0000);

}

private void initPaint() {

if (paint == null) {

paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);

paint.setStyle(Paint.Style.FILL);

}

}

public ItemDivider setDividerWith(int dividerWith) {

this.dividerWith = dividerWith;

return this;

}

public ItemDivider setDividerColor(int color) {

initPaint();

paint.setColor(color);

return this;

}

/\*\*

\* 指定item之间的间距(就是指定分割线的宽度) 回调顺序 1

\* @param outRect Rect to receive the output.

\* @param view The child view to decorate

\* @param parent RecyclerView this ItemDecoration is decorating

\* @param state The current state of RecyclerView.

\*/

@Override

public void getItemOffsets(Rect outRect, View view, RecyclerView parent, RecyclerView.State state) {

super.getItemOffsets(outRect, view, parent, state);

if (layoutManager == null) {

layoutManager = parent.getLayoutManager();

}

// 适用 LinearLayoutManager 和 GridLayoutManager

if (layoutManager instanceof LinearLayoutManager) {

int orientation = ((LinearLayoutManager) layoutManager).getOrientation();

if (orientation == LinearLayoutManager.VERTICAL) {

// 水平分割线将绘制在item底部

outRect.bottom = dividerWith;

} else if (orientation == LinearLayoutManager.HORIZONTAL) {

// 垂直分割线将绘制在item右侧

outRect.right = dividerWith;

}

if (layoutManager instanceof GridLayoutManager) {

GridLayoutManager.LayoutParams lp = (GridLayoutManager.LayoutParams) view.getLayoutParams();

// 如果是 GridLayoutManager 则需要绘制另一个方向上的分割线

if (orientation == LinearLayoutManager.VERTICAL && lp != null && lp.getSpanIndex() > 0) {

// 如果列表是垂直方向,则最左边的一列略过

outRect.left = dividerWith;

} else if (orientation == LinearLayoutManager.HORIZONTAL && lp != null && lp.getSpanIndex() > 0) {

// 如果列表是水平方向,则最上边的一列略过

outRect.top = dividerWith;

}

}

}

}

/\*\*

\* 在item 绘制之前调用(就是绘制在 item 的底层) 回调顺序 2

\* 一般分割线在这里绘制

\* 看到canvas,对自定义控件有一定了解的话,就能想到为什么说给RecyclerView设置分割线更灵活了

\* @param c Canvas to draw into

\* @param parent RecyclerView this ItemDecoration is drawing into

\* @param state The current state of RecyclerView

\*/

@Override

public void onDraw(Canvas c, RecyclerView parent, RecyclerView.State state) {

super.onDraw(c, parent, state);

// 这个值是为了补偿横竖方向上分割线交叉处间隙

int offSet = (int) Math.ceil(dividerWith \* 1f / 2);

for (int i = 0; i < parent.getChildCount(); i++) {

View child = parent.getChildAt(i);

RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child.getLayoutParams();

int left1 = child.getRight() + params.rightMargin;

int right1 = left1 + dividerWith;

int top1 = child.getTop() - offSet - params.topMargin;

int bottom1 = child.getBottom() + offSet + params.bottomMargin;

//绘制分割线(矩形)

c.drawRect(left1, top1, right1, bottom1, paint);

int left2 = child.getLeft() - offSet - params.leftMargin;

int right2 = child.getRight() + offSet + params.rightMargin;

int top2 = child.getBottom() + params.bottomMargin;

int bottom2 = top2 + dividerWith;

//绘制分割线(矩形)

c.drawRect(left2, top2, right2, bottom2, paint);

}

}

/\*\*

\* 在item 绘制之后调用(就是绘制在 item 的上层) 回调顺序 3

\* 也可以在这里绘制分割线,和上面的方法 二选一

\* @param c Canvas to draw into

\* @param parent RecyclerView this ItemDecoration is drawing into

\* @param state The current state of RecyclerView

\*/

@Override

public void onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state) {

super.onDrawOver(c, parent, state);

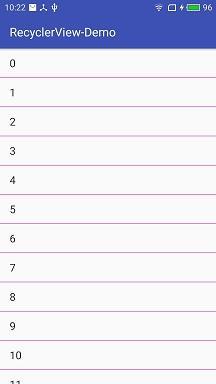
}

}

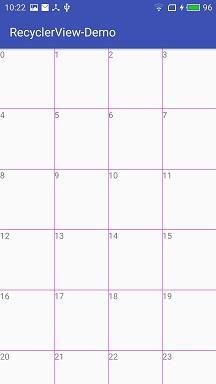
使用方式

recyclerView.addItemDecoration(new ItemDivider().setDividerWith(2).setDividerColor(Color.BLUE));

看看效果



LinearLayoutManager



GridLayoutManager

掌握了分割线的原理,还可以做很多有意思的事.比如像列表分栏,在IOS中很容易做到让当前栏目悬停的效果. 而Android中的常规做法,就是布局嵌套,在屏幕上面单独方一个文本,然后监听列表的滚动.....太麻烦了. 其实借助分割线的原理,可以更简单实现这个效果.

基于组件化的思想,可以将这个功能封装为一个单独的控件

public class StickyRecyclerView extends RecyclerView {

private int lineHeight,titleHeight;

private int lineColor,titleColor,titleTextColor;

public StickyRecyclerView(Context context) {

this(context,null);

}

public StickyRecyclerView(Context context, @Nullable AttributeSet attrs) {

this(context, attrs,0);

}

public StickyRecyclerView(Context context, @Nullable AttributeSet attrs, int defStyle) {

super(context, attrs, defStyle);

TypedArray array = context.obtainStyledAttributes(attrs, R.styleable.StickyRecyclerView);

// 分割线的高度

lineHeight = array.getDimensionPixelOffset(R.styleable.StickyRecyclerView\_dividerHeight,1);

// 分栏的高度

titleHeight = array.getDimensionPixelOffset(R.styleable.StickyRecyclerView\_titleHeight,dip2px(context,35));

// 分割线颜色

lineColor = array.getColor(R.styleable.StickyRecyclerView\_dividerColor,Color.LTGRAY);

// 分栏背景色

titleColor = array.getColor(R.styleable.StickyRecyclerView\_titleColor,Color.LTGRAY);

// 分栏文字颜色

titleTextColor = array.getColor(R.styleable.StickyRecyclerView\_titleTextColor,Color.BLUE);

array.recycle();

// 不用说,肯定是线性布局了,默认就实现

setLayoutManager(new LinearLayoutManager(context));

}

@Deprecated

@Override

public void setAdapter(Adapter adapter) {

super.setAdapter(adapter);

}

// 让 adapter 必须继承 StickyAdapter

public void setAdapter(@NonNull StickyAdapter stickyAdapter){

addItemDecoration(new StickyDivider(stickyAdapter));

super.setAdapter(stickyAdapter);

}

/\*\*

\* 自定义分割线,通过分割线绘制title

\*/

private class StickyDivider extends ItemDecoration{

private StickyAdapter adapter;

private Paint paint;

StickyDivider(@NonNull StickyAdapter adapter) {

super();

this.adapter = adapter;

paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);

paint.setStyle(Paint.Style.FILL);

paint.setTextSize(titleHeight \* 0.5f);

}

/\*\*

\* 计算 item间间隙(是普通分割线 ,还是title)

\*/

@Override

public void getItemOffsets(Rect outRect, View view, RecyclerView parent, State state) {

super.getItemOffsets(outRect, view, parent, state);

if(!adapter.needTitle(((LayoutParams) view.getLayoutParams()).getViewLayoutPosition())){

outRect.top = lineHeight;

}else {

outRect.top = titleHeight;

}

}

/\*\*

\* 底层绘制,绘制分栏title

\*/

@Override

public void onDraw(Canvas c, RecyclerView parent, State state) {

super.onDraw(c, parent, state);

int left = parent.getPaddingLeft();

int right = parent.getMeasuredWidth() - parent.getPaddingRight();

final int childCount = parent.getChildCount();

for (int i = 0; i < childCount; i++) {

final View child = parent.getChildAt(i);

int position = ((LayoutParams) child.getLayoutParams()).getViewLayoutPosition();

int bottom = child.getTop() - ((LayoutParams) child.getLayoutParams()).topMargin;

if(!adapter.needTitle(position)){

// 画分割线

int top = bottom - lineHeight;

paint.setColor(lineColor);

c.drawRect(left, top, right, bottom, paint);

}else {

//画TITLE

int top = bottom - titleHeight;

paint.setColor(titleColor);

c.drawRect(left, top, right, bottom, paint);

drawText(c,adapter.getItemViewTitle(position),left + titleHeight \* 0.25f,bottom - titleHeight \* 0.25f);

}

}

}

/\*\*

\* 上层绘制,绘制顶部悬停title

\*/

@Override

public void onDrawOver(Canvas c, RecyclerView parent, State state) {

super.onDrawOver(c, parent, state);

// 悬停title

int left = parent.getPaddingLeft();

int right = parent.getMeasuredWidth() - parent.getPaddingRight();

int top = parent.getPaddingTop();

int bottom = top + titleHeight;

paint.setColor(titleColor);

c.drawRect(left,top,right,bottom,paint);

int pos = ((LinearLayoutManager)(parent.getLayoutManager())).findFirstVisibleItemPosition();

drawText(c,adapter.getItemViewTitle(pos),left + titleHeight \* 0.25f,bottom - titleHeight \* 0.25f);

}

void drawText(Canvas c, String itemViewTitle, float x, float y){

if(!TextUtils.isEmpty(itemViewTitle)){

paint.setColor(titleTextColor);

//paint.getTextBounds(itemViewTitle, 0, itemViewTitle.length(), mBounds);

c.drawText(itemViewTitle, x,y, paint);

}

}

}

public static abstract class StickyAdapter extends Adapter{

// 获取当前 item 的标题

public abstract String getItemViewTitle(int position);

// 如果标题和前面的item的标题一样,就不需要绘制

boolean needTitle(int position){

return position > -1 && (position == 0 || !getItemViewTitle(position).equals(getItemViewTitle(position - 1)));

}

}

public int dip2px(Context context, float dpValue) {

final float scale = context.getResources().getDisplayMetrics().density;

return (int) (dpValue \* scale + 0.5f);

}

}

大致流程就是通过底层分割线绘制各个分栏,通过顶层分割线绘制顶部悬停的那一栏,具体可以看下注释.  
使用方式和普通RecyclerView 差不多:

stickyRecyclerView.setAdapter(myAdapter);

//关键一:继承关系private class MyAdapter extends StickyRecyclerView.StickyAdapter {

.....

//关键二:重写该方法,返回当前item的标题

@Override

public String getItemViewTitle(int position) {

return String.valueOf(datas.get(position).shuruma.charAt(0));

}

}

其中,分栏背景色,高度,文字颜色,以及分割线颜色和高度都是可以通过自定义属性设置的.

###### **不规则布局**

网格布局很常见,但是不规则的网格布局也不少见.比如要实现下面这个效果



上面是网格,下面又变成列表,在以前的做法可能是给ListView添加一个头部,头部里面放GridView,甚至是ScrollView嵌套等等.做过的同学肯定知道有多少坑在里面. 而使用 RecyclerView ,可以做很大程度的简化,并且很容易就能实现更复杂的布局.

RecyclerView 可以通过 GridLayoutManager 实现网格布局.而要实现上面的效果,关键就在 GridLayoutManager上, GridLayoutManager 可以设置网格的列数,而通过下面的方法,可以指定每一个item占据的列数.

gridLayoutManager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {

@Override

public int getSpanSize(int position) {

// 这里的返回值,表示下标为position的item 占据多少列

return 1;

}

});

通过下面这个例子看起来更加直观:

gridLayoutManager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {

@Override

public int getSpanSize(int position) {

//这里只是一个例子,实际中要根据需求来设置

if(position % 5 == 0){

return 4;

}else if(position % 5 == 1){

return 3;

}else if(position % 5 == 2){

return 1;

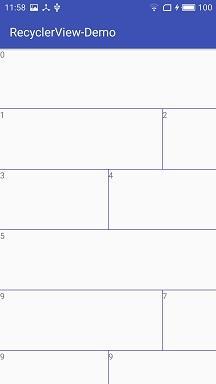
}else{

return 2;

}

}

});



不规则布局

关于不规则布局的内容不多,这里再补充一个例子. ****RecyclerView分页加载,**** Google官方以及一些第三方的下拉刷新控件都不支持分页功能,因为分页功能应该让列表自己去实现. 而目前的列表基本都可以使用RecyclerView完成,所以如果能做个统一封装就方便多了(这里就和 SwipeRefreshLayout封装在一起了,顺便解决 SwipeRefreshLayout 的坑).

public class SuperRefreshLayout extends SwipeRefreshLayout {

private static OnRefreshHandler onRefreshHandler;

private static boolean isRefresh = false;

private Adapter adapter;

private int mTouchSlop;

private float mPrevX;

public SuperRefreshLayout(Context context) {

this(context, null);

}

public SuperRefreshLayout(Context context, AttributeSet attrs) {

super(context, attrs);

setColorSchemeColors(0xff3b93eb);

setProgressBackgroundColorSchemeColor(0xffffffff);

float scale = context.getResources().getDisplayMetrics().density;

setProgressViewEndTarget(true, (int) (64 \* scale + 0.5f));

//refreshLayout.setProgressViewOffset(false,dip2px(this,-40),dip2px(this,64));

mTouchSlop = ViewConfiguration.get(context).getScaledTouchSlop();

}

/\*\*

\* 监听器

\*/

public void setOnRefreshHandler(OnRefreshHandler handler) {

onRefreshHandler = handler;

super.setOnRefreshListener(new OnRefreshCallBack());

}

/\*\*

\* 自动刷新,原生不支持,通过反射修改字段属性

\*/

public void autoRefresh() {

try {

setRefreshing(true);

Field field = SwipeRefreshLayout.class.getDeclaredField("mNotify");

field.setAccessible(true);

field.set(this, true);

} catch (Exception e) {

if(onRefreshHandler != null){

onRefreshHandler.refresh();

}

}

}

@Override

public void setRefreshing(boolean refreshing) {

super.setRefreshing(refreshing);

isRefresh = isRefreshing();

}

/\*\*

\* 加载完毕

\* @param hasMore 是否还有下一页

\*/

public void loadComplete(boolean hasMore){

if(adapter == null){

throw new RuntimeException("must call method setAdapter to bind data");

}

adapter.setState(hasMore ? Adapter.STATE\_MORE : Adapter.STATE\_END);

}

/\*\*

\* 加载出错

\*/

public void loadError(){

if(adapter == null){

throw new RuntimeException("must call method setAdapter to bind data");

}

adapter.setState(Adapter.STATE\_ERROR);

}

/\*\*

\* 只支持 RecyclerView 加载更多,且需要通过此方法设置适配器

\*/

public void setAdapter(@NonNull RecyclerView recyclerView,@NonNull SuperRefreshLayout.Adapter mAdapter) {

adapter = mAdapter;

recyclerView.setAdapter(adapter);

recyclerView.addOnScrollListener(new RecyclerView.OnScrollListener() {

@Override

public void onScrollStateChanged(RecyclerView recyclerView, int newState) {

super.onScrollStateChanged(recyclerView, newState);

if (onRefreshHandler != null

&& !isRefreshing()

&& (adapter.getState() == Adapter.STATE\_MORE || adapter.getState() == Adapter.STATE\_ERROR)

&& newState == RecyclerView.SCROLL\_STATE\_IDLE

&& !ViewCompat.canScrollVertically(recyclerView, 1)

) {

adapter.setState(Adapter.STATE\_LOAIND);

onRefreshHandler.loadMore();

}

}

});

}

/\*\*

\* 如果滑动控件嵌套过深,可通过该方法控制是否可以下拉

\*/

public void setRefreshEnable(boolean enable){

// boolean e = !ViewCompat.canScrollVertically(scrollView,-1);

if(isEnabled() && !enable){

setEnabled(false);

}else if(!isEnabled() && enable){

setEnabled(true);

}

}

/\*\*

\* 解决水平滑动冲突

\*/

@Override

public boolean onInterceptTouchEvent(MotionEvent event) {

switch (event.getAction()) {

case MotionEvent.ACTION\_DOWN:

mPrevX = MotionEvent.obtain(event).getX();

break;

case MotionEvent.ACTION\_MOVE:

final float eventX = event.getX();

float xDiff = Math.abs(eventX - mPrevX);

if (xDiff > mTouchSlop) {

return false;

}

}

return super.onInterceptTouchEvent(event);

}

private class OnRefreshCallBack implements OnRefreshListener {

@Override

public void onRefresh() {

if(adapter != null && adapter.getState() != Adapter.STATE\_MORE){

adapter.setState(Adapter.STATE\_MORE);

}

if(onRefreshHandler != null){

onRefreshHandler.refresh();

}

}

}

public static abstract class OnRefreshHandler{

public abstract void refresh();

public void loadMore() {

}

}

/\*\*

\* 支持加载更多的适配器

\*/

public static abstract class Adapter extends RecyclerView.Adapter {

static final int STATE\_MORE = 0, STATE\_LOAIND = 1, STATE\_END = 2, STATE\_ERROR = 3;

int state = STATE\_MORE;

public void setState(int state) {

if (this.state != state) {

this.state = state;

notifyItemChanged(getItemCount() - 1);

}

}

public int getState() {

return state;

}

@Override

public int getItemViewType(int position) {

if (position == getItemCount() - 1) {

return -99;

}

return getItemType(position);

}

@Override

public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

if (viewType == -99) {

return new RecyclerView.ViewHolder(LayoutInflater.from(parent.getContext()).inflate(R.layout.loadmore\_default\_footer, parent, false)) {};

} else {

return onCreateItemHolder(parent, viewType);

}

}

@Override

public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {

if (getItemViewType(position) == -99) {

ProgressBar progressBar = (ProgressBar) holder.itemView.findViewById(R.id.loadmore\_default\_footer\_progressbar);

TextView textView = (TextView) holder.itemView.findViewById(R.id.loadmore\_default\_footer\_tv);

if (state == STATE\_END) {

progressBar.setVisibility(View.GONE);

textView.setText("没有更多了");

} else if (state == STATE\_MORE) {

progressBar.setVisibility(View.GONE);

textView.setText("点击加载");

} else if (state == STATE\_LOAIND) {

progressBar.setVisibility(View.VISIBLE);

textView.setText("加载中...");

} else if (state == STATE\_ERROR) {

progressBar.setVisibility(View.GONE);

textView.setText("加载失败,点击重新加载");

}

holder.itemView.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View view) {

if (onRefreshHandler != null && !isRefresh && (state == STATE\_MORE || state == STATE\_ERROR)) {

setState(STATE\_LOAIND);

onRefreshHandler.loadMore();

}

}

});

} else {

onBindItemHolder(holder,position);

}

}

@Override

public int getItemCount() {

return getCount() == 0 ? 0 : getCount() + 1;

}

public int getItemType(int position){

return super.getItemViewType(position);

}

public abstract RecyclerView.ViewHolder onCreateItemHolder(ViewGroup parent, int viewType);

public abstract void onBindItemHolder(RecyclerView.ViewHolder holder, int position);

public abstract int getCount();

@Override

public void onViewAttachedToWindow(RecyclerView.ViewHolder holder) {

// 处理瀑布流模式 最后的 item 占整行

if (holder.getLayoutPosition() == getItemCount() - 1) {

LayoutParams lp = holder.itemView.getLayoutParams();

if (lp != null && lp instanceof StaggeredGridLayoutManager.LayoutParams) {

StaggeredGridLayoutManager.LayoutParams p = (StaggeredGridLayoutManager.LayoutParams) lp;

p.setFullSpan(true);

}

}

}

@Override

public void onAttachedToRecyclerView(RecyclerView recyclerView) {

// 处理网格布局模式 最后的 item 占整行

final RecyclerView.LayoutManager layoutManager = recyclerView.getLayoutManager();

if (layoutManager instanceof GridLayoutManager) {

GridLayoutManager gridManager = ((GridLayoutManager) layoutManager);

final GridLayoutManager.SpanSizeLookup spanSizeLookup = gridManager.getSpanSizeLookup();

final int lastSpanCount = gridManager.getSpanCount();

gridManager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {

@Override

public int getSpanSize(int position) {

return position == getItemCount() - 1 ? lastSpanCount :

(spanSizeLookup == null ? 1 : spanSizeLookup.getSpanSize(position));

}

});

}

}

}

}

****整体思路就是给RecyclerView在末尾添加了一个item,并且必要保证这个item占据整行.**** 所以需要处理两种情况:

StaggeredGridLayoutManager

StaggeredGridLayoutManager.LayoutParams p = (StaggeredGridLayoutManager.LayoutParams) lp;

//设置为占满整行

p.setFullSpan(true);

GridLayoutManager

gridManager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {

@Override

public int getSpanSize(int position) {

return position == getItemCount() - 1 ? lastSpanCount :

(spanSizeLookup == null ? 1 : spanSizeLookup.getSpanSize(position));

}

});

所以利用不规则布局就可以让RecyclerView支持分页功能了.

###### **拖动排序和滑动删除**

RecyclerView的拖动拍和滑动删除需要靠 ItemTouchHelper 这个类来支持, ItemTouchHelper 有个内部抽象类 Callback ,实现这个类可以让我们定义相关规则,以及处理回调事件.直接看代码,每个方法都有注释:

public class MyItemTouchHandler extends ItemTouchHelper.Callback {

ItemTouchAdapterImpl adapter;

public MyItemTouchHandler(@NonNull ItemTouchAdapterImpl adapter) {

this.adapter = adapter;

}

/\*\*

\* 设置 允许拖拽和滑动删除的方向

\*/

@Override

public int getMovementFlags(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder) {

// 指定可 拖拽方向 和 滑动消失的方向

int dragFlags,swipeFlags;

RecyclerView.LayoutManager manager = recyclerView.getLayoutManager();

if (manager instanceof GridLayoutManager || manager instanceof StaggeredGridLayoutManager) {

// 上下左右都可以拖动

dragFlags = ItemTouchHelper.UP | ItemTouchHelper.DOWN | ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT;

} else {

// 可以上下拖动

dragFlags = ItemTouchHelper.UP | ItemTouchHelper.DOWN;

}

// 可以左右方向滑动消失

swipeFlags = ItemTouchHelper.START | ItemTouchHelper.END;

// 如果某个值传 0 , 表示不支持该功能

return makeMovementFlags(dragFlags, swipeFlags);

}

/\*\*

\* 拖拽后回调,一般通过接口暴露给adapter, 让adapter去处理数据的交换

\*/

@Override

public boolean onMove(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder, RecyclerView.ViewHolder target) {

// 相同 viewType 之间才能拖动交换

if (viewHolder.getItemViewType() == target.getItemViewType()) {

int fromPosition = viewHolder.getAdapterPosition();

int toPosition = target.getAdapterPosition();

if (fromPosition < toPosition) {

//途中所有的item位置都要移动

for (int i = fromPosition; i < toPosition; i++) {

adapter.onItemMove(i, i + 1);

}

} else {

for (int i = fromPosition; i > toPosition; i--) {

adapter.onItemMove(i, i - 1);

}

}

adapter.notifyItemMoved(fromPosition, toPosition);

return true;

}

return false;

}

/\*\*

\* 滑动删除后回调,一般通过接口暴露给adapter, 让adapter去删除该条数据

\*/

@Override

public void onSwiped(RecyclerView.ViewHolder viewHolder, int direction) {

// 删除数据

adapter.onItemRemove(viewHolder.getAdapterPosition());

// adapter 刷新

adapter.notifyItemRemoved(viewHolder.getAdapterPosition());

}

@Override

public void onChildDraw(Canvas c, RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder, float dX, float dY, int actionState, boolean isCurrentlyActive) {

super.onChildDraw(c, recyclerView, viewHolder, dX, dY, actionState, isCurrentlyActive);

if(actionState == ItemTouchHelper.ACTION\_STATE\_SWIPE) {

//滑动时改变Item的透明度

final float alpha = 1 - Math.abs(dX) / (float)viewHolder.itemView.getWidth();

viewHolder.itemView.setAlpha(alpha);

viewHolder.itemView.setTranslationX(dX);

}

}

/\*\*

\* item被选中(长按)

\* 这里改变了 item的背景色, 也可以通过接口暴露, 让adapter去处理逻辑

\*/

@Override

public void onSelectedChanged(RecyclerView.ViewHolder viewHolder, int actionState) {

if (actionState == ItemTouchHelper.ACTION\_STATE\_DRAG) {

// 拖拽状态

viewHolder.itemView.setBackgroundColor(Color.BLUE);

}else if (actionState == ItemTouchHelper.ACTION\_STATE\_SWIPE) {

// 滑动删除状态

viewHolder.itemView.setBackgroundColor(Color.RED);

}

super.onSelectedChanged(viewHolder, actionState);

}

/\*\*

\* item取消选中(取消长按)

\* 这里改变了 item的背景色, 也可以通过接口暴露, 让adapter去处理逻辑

\*/

@Override

public void clearView(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder) {

viewHolder.itemView.setBackgroundColor(Color.TRANSPARENT);

super.clearView(recyclerView, viewHolder);

}

/\*\*

\* 是否支持长按开始拖拽,默认开启 \* 可以不开启,然后在长按 item 的时候,手动 调用 mItemTouchHelper.startDrag(myHolder) 开启,更加灵活

\*/

@Override

public boolean isLongPressDragEnabled() {

return adapter.autoOpenDrag();

}

/\*\*

\* 是否支持滑动删除,默认开启 \* 可以不开启,然后在长按 item 的时候,手动 调用 mItemTouchHelper.startSwipe(myHolder) 开启,更加灵活

\*/

@Override

public boolean isItemViewSwipeEnabled() {

return adapter.autoOpenSwipe();

}

// 建议让 adapter 实现该接口

public static abstract class ItemTouchAdapterImpl extends RecyclerView.Adapter{

public abstract void onItemMove(int fromPosition, int toPosition);

public abstract void onItemRemove(int position);

// 是否自动开启拖拽

protected boolean autoOpenDrag(){

return true;

}

// 是否自动开启滑动删除

protected boolean autoOpenSwipe(){

return true;

}

}

}

使用方式

new ItemTouchHelper(new MyItemTouchHandler(myAdapter)).attachToRecyclerView(recyclerView);

...

private class MyAdapter extends MyItemTouchHandler.ItemTouchAdapterImpl{

...

@Override

public void onItemMove(int fromPosition, int toPosition) {

// 拖动排序的回调,这里交换集合中数据的位置

Collections.swap(str, fromPosition, toPosition);

}

@Override

public void onItemRemove(int position) {

// 滑动删除的回调,这里删除指定的数据

}

}