**[打造属于你的LayoutManager](http://blog.csdn.net/huachao1001/article/details/51594004)**

# 1 RecyclerView机制

RecyclerView内部有个Recycler，它其实就是一个垃圾回收再利用的工具，我们定义LayoutManager时，我们需要将不用的View回收掉；在需要获取新的View时直接申请，即通过getViewForPosition()方法，返回的View可能是之前回收的垃圾View，也可能是new出来的新View，这些都是RecyclerView帮我们做的。那么RecyclerView内部的垃圾View缓存是什么样子的呢？我们接下来看看~

## 1.1RecyclerView的二级缓存

在RecyclerView中，有两个缓存：Scrap和Recycle。Scrap中文就是废料的意思，Recycle对应是回收的意思。这两个缓存有什么作用呢？首先Scrap缓存是指里面缓存的View是接下来需要用到的，即里面的绑定的数据无需更改，可以直接拿来用的，是一个轻量级的缓存集合；而Recycle的缓存的View为里面的数据需要重新绑定，即需要通过Adapter重新绑定数据。关于这两个缓存的使用场景，下一节详细介绍。

当我们去获取一个新的View时，RecyclerView首先去检查Scrap缓存是否有对应的position的View，如果有，则直接拿出来可以直接用，不用去重新绑定数据；如果没有，则从Recycle缓存中取，并且会回调Adapter的onBindViewHolder方法（当然了，如果Recycle缓存为空，还会调用onCreateViewHolder方法），最后再将绑定好新数据的View返回。

## 1.2 将View缓存的两种方式

前面我们了解到，RecyclerView中有二级缓存，我们可以自己选择将View缓存到哪里。我们有两种选择的方式：Detach和Remove。Detach的View放在Scrap缓存中，Remove掉的View放在Recycle缓存中；那我们应该如何去选择呢？

在什么样的场景中使用Detach呢？主要是在我们的代码执行结束之前，我们需要反复去将View移除并且马上又要添加进去时，选择Datach方式，比如：**当我们对View进行重新排序的时候，可以选择Detach，因为屏幕上显示的就是这些position对应的View，我们并不需要重新去绑定数据，这明显可以提高效率**。使用Detach方式可以通过函数detachAndScrapView()实现。

而使用Remove的方式，是当View不在屏幕中有任何显示的时候，你需要将它Remove掉，以备后面循环利用。可以通过函数removeAndRecycleView()实现。

# 2 开始自定义LayoutManager

首先，得将我们自定义的LayoutManager继承RecyclerView.LayoutManager，而RecyclerView.LayoutManager是一个抽象类，但是抽象方法只有一个generateDefaultLayoutParams也就是说，我们只需要重新这一个方法就可以自定义我们自己的LayoutManager啦~，让我们happy地去自定义吧~

@Overridepublic RecyclerView.LayoutParams generateDefaultLayoutParams() {

return new RecyclerView.LayoutParams(

RecyclerView.LayoutParams.WRAP\_CONTENT,

RecyclerView.LayoutParams.WRAP\_CONTENT);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2
* 3
* 4
* 5
* 6

然后在在MainActivity中默默的加了以下两行代码：

MyLayoutManager layoutManager = new MyLayoutManager();

recyclerView.setLayoutManager(layoutManager);

* 1
* 2
* 1
* 2

很兴奋的运行看效果~~~：哇擦！啥都没有~。哈哈，被耍的感觉有木有！其实，学过自定义ViewGroup都知道，我们需要对子View进行布局，不了解的可以参考我的另一篇博文[《自定义View，有这一篇就够了 》](http://blog.csdn.net/huachao1001/article/details/51577291)，即需要重写onLayout()函数，并且在函数体里面需要对子View进行布局。我们自定义的LayoutManager主要工作就是对子View布局，那更需要我们重新类似onLayout的函数了。正如你所想的那样，LayoutManager有个函数onLayoutChildren()就是负责对子View布局的。

如果我们需要实现一个垂直方向的LinearLayout，我们可以这么写：

@Overridepublic void onLayoutChildren(RecyclerView.Recycler recycler, RecyclerView.State state) {

//在布局之前，将所有的子View先Detach掉，放入到Scrap缓存中

detachAndScrapAttachedViews(recycler);

//定义竖直方向的偏移量

int offsetY = 0;

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

//这里就是从缓存里面取出

View view = recycler.getViewForPosition(i);

//将View加入到RecyclerView中

addView(view);

//对子View进行测量

measureChildWithMargins(view, 0, 0);

//把宽高拿到，宽高都是包含ItemDecorate的尺寸

int width = getDecoratedMeasuredWidth(view);

int height = getDecoratedMeasuredHeight(view);

//最后，将View布局

layoutDecorated(view, 0, offsetY, width, offsetY + height);

//将竖直方向偏移量增大height

offsetY += height;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23

注意到，我们在最开始先执行了detachAndScrapAttachedViews(recycler)，即将所有的子View先Detach掉，放入到Scrap缓存中，为什么要这样做呢？主要是考虑到，屏幕上可能还有一些ItemView是继续要留在屏幕上的，我们不直接Remove，而是选择Detach。

看一下MainActivity的完整代码吧~

package com.hc.customlayoutmanager;

import android.os.Bundle;import android.support.v7.app.AppCompatActivity;import android.support.v7.widget.RecyclerView;import android.view.LayoutInflater;import android.view.View;import android.view.ViewGroup;import android.widget.TextView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

private RecyclerView recyclerView;

private ArrayList<MyEntity> myData;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

recyclerView = (RecyclerView) findViewById(R.id.recyclerView);

initData();

MyLayoutManager layoutManager = new MyLayoutManager();

MyAdapter adapter = new MyAdapter();

recyclerView.setLayoutManager(layoutManager);

recyclerView.setAdapter(adapter);

}

//初始化数据

private void initData() {

int size = 30;

myData = new ArrayList<>(size);

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

MyEntity e = new MyEntity();

e.setStr("str:" + i);

myData.add(e);

}

}

//自定义Adapter

class MyAdapter extends RecyclerView.Adapter<MyViewHolder> {

@Override

public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

View v = LayoutInflater.from(MainActivity.this).inflate(R.layout.recycler\_view\_item, parent, false);

MyViewHolder viewHolder = new MyViewHolder(v);

return viewHolder;

}

@Override

public void onBindViewHolder(MyViewHolder holder, int position) {

MyEntity myEntity = myData.get(position);

holder.setStr(myEntity.getStr());

}

@Override

public int getItemCount() {

return myData.size();

}

}

//自定义Holder

static class MyViewHolder extends RecyclerView.ViewHolder {

private TextView strTv;

public MyViewHolder(View itemView) {

super(itemView);

strTv = (TextView) itemView.findViewById(R.id.str);

}

public void setStr(String str) {

strTv.setText(str);

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84

其他非关键代码这里就不贴出来了，后面会附上源码~

# 3 添加滑动

现在我们实现了简单的layout，但是还不能像ListView那样滑动，那么如何设置滚动呢？首先，你得重写canScrollVertically()函数，并返回true。同理，如果实现水平方向的滑动，则重写canScrollHorizontally()并返回true。

@Overridepublic boolean canScrollVertically() {

return true;

}

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

然后重写scrollVerticallyBy()函数，用于实现垂直方向滑动，同理，如果你想要实现水平方向的滑动那么重写scrollHorizontallyBy()函数。

@Overridepublic int scrollVerticallyBy(int dy, RecyclerView.Recycler recycler, RecyclerView.State state) {

//实际要滑动的距离

int travel = dy;

//如果滑动到最顶部

if (verticalScrollOffset + dy < 0) {

travel = -verticalScrollOffset;

} else if (verticalScrollOffset + dy > totalHeight - getVerticalSpace()) {//如果滑动到最底部

travel = totalHeight - getVerticalSpace() - verticalScrollOffset;

}

//将竖直方向的偏移量+travel

verticalScrollOffset += travel;

// 平移容器内的item

offsetChildrenVertical(-travel);

return travel;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21

其中getVerticalSpace()函数是用于获取RecyclerView在垂直方向上的可用空间，即去除了padding后的高度：

private int getVerticalSpace() {

return getHeight() - getPaddingBottom() - getPaddingTop();

}

* 1
* 2
* 3
* 1
* 2
* 3

另外就是，我们要获取所有的ItemView的高度之和totalHeight，以及竖直方向的滑动偏移量verticalScrollOffset，verticalScrollOffset的起始值为0，而totalHeight可用通过遍历子View来获取，在scrollVerticallyBy()函数中可用获取这两个数据：

private int verticalScrollOffset = 0;private int totalHeight = 0;

@Overridepublic void onLayoutChildren(RecyclerView.Recycler recycler, RecyclerView.State state) {

//如果没有item，直接返回

if (getItemCount() <= 0) return;

// 跳过preLayout，preLayout主要用于支持动画

if (state.isPreLayout()) {

return;

}

//在布局之前，将所有的子View先Detach掉，放入到Scrap缓存中

detachAndScrapAttachedViews(recycler);

//定义竖直方向的偏移量

int offsetY = 0;

totalHeight = 0;

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

//这里就是从缓存里面取出

View view = recycler.getViewForPosition(i);

//将View加入到RecyclerView中

addView(view);

measureChildWithMargins(view, 0, 0);

int width = getDecoratedMeasuredWidth(view);

int height = getDecoratedMeasuredHeight(view);

//最后，将View布局

layoutDecorated(view, 0, offsetY, width, offsetY + height);

//将竖直方向偏移量增大height

offsetY += height;

//

totalHeight += height;

}

//如果所有子View的高度和没有填满RecyclerView的高度，

// 则将高度设置为RecyclerView的高度

totalHeight = Math.max(totalHeight, getVerticalSpace());

}

# 4 回收子View

当你觉得一切都非常完美的时候，却忽略了一个很关键的点！那就是回收~。我们知道，RecyclerView强大就强大在View的循环回收利用上，而一个View是否需要回收，是由我们的LayoutManager来管理的~还记得我们前面说的Remove吗？也就是将View放到Recycle缓存中去~

我们前面自定义的LayoutManager并没有回收子View，接下来我们去看看如何循环利用子View吧~。首先，我们应该将所有的item的上下左右的偏移量记录下来，并且要记录哪些Item需要被回收：

//保存所有的Item的上下左右的偏移量信息private SparseArray<Rect> allItemFrames = new SparseArray<>();//记录Item是否出现过屏幕且还没有回收。true表示出现过屏幕上，并且还没被回收private SparseBooleanArray hasAttachedItems = new SparseBooleanArray();

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

接下来就是初始化这两个变量，在onLayoutChildren()函数中对上面定义的两个变量进行初始化：

@Override

public void onLayoutChildren(RecyclerView.Recycler recycler, RecyclerView.State state) {

//如果没有item，直接返回

if (getItemCount() <= 0) return;

// 跳过preLayout，preLayout主要用于支持动画

if (state.isPreLayout()) {

return;

}

//在布局之前，将所有的子View先Detach掉，放入到Scrap缓存中

detachAndScrapAttachedViews(recycler);

//定义竖直方向的偏移量

int offsetY = 0;

totalHeight = 0;

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

//这里就是从缓存里面取出

View view = recycler.getViewForPosition(i);

//将View加入到RecyclerView中

addView(view);

measureChildWithMargins(view, 0, 0);

int width = getDecoratedMeasuredWidth(view);

int height = getDecoratedMeasuredHeight(view);

totalHeight += height;

Rect frame = allItemFrames.get(i);

if (frame == null) {

frame = new Rect();

}

frame.set(0, offsetY, width, offsetY + height);

// 将当前的Item的Rect边界数据保存

allItemFrames.put(i, frame);

// 由于已经调用了detachAndScrapAttachedViews，因此需要将当前的Item设置为未出现过

hasAttachedItems.put(i, false);

//将竖直方向偏移量增大height

offsetY += height;

}

//如果所有子View的高度和没有填满RecyclerView的高度，

// 则将高度设置为RecyclerView的高度

totalHeight = Math.max(totalHeight, getVerticalSpace());

recycleAndFillItems(recycler, state);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42

注意，上面的for循环里面并没有再调用layoutDecorated()函数，而是在最后调用了recycleAndFillItems()函数，这个函数是先将不需要的Item进行回收，然后在从缓存中取出需要的Item，代码如下：

/\*\*

\* 回收不需要的Item，并且将需要显示的Item从缓存中取出

\*/private void recycleAndFillItems(RecyclerView.Recycler recycler, RecyclerView.State state) {

if (state.isPreLayout()) { // 跳过preLayout，preLayout主要用于支持动画

return;

}

// 当前scroll offset状态下的显示区域

Rect displayFrame = new Rect(0, verticalScrollOffset, getHorizontalSpace(), verticalScrollOffset + getVerticalSpace());

/\*\*

\* 将滑出屏幕的Items回收到Recycle缓存中

\*/

Rect childFrame = new Rect();

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

View child = getChildAt(i);

childFrame.left = getDecoratedLeft(child);

childFrame.top = getDecoratedTop(child);

childFrame.right = getDecoratedRight(child);

childFrame.bottom = getDecoratedBottom(child);

//如果Item没有在显示区域，就说明需要回收

if (!Rect.intersects(displayFrame, childFrame)) {

//回收掉滑出屏幕的View

removeAndRecycleView(child, recycler);

}

}

//重新显示需要出现在屏幕的子View

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

if (Rect.intersects(displayFrame, allItemFrames.get(i))) {

View scrap = recycler.getViewForPosition(i);

measureChildWithMargins(scrap, 0, 0);

addView(scrap);

Rect frame = allItemFrames.get(i);

//将这个item布局出来

layoutDecorated(scrap,

frame.left,

frame.top - verticalScrollOffset,

frame.right,

frame.bottom - verticalScrollOffset);

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49

最后不要忘了在scrollVerticallyBy中添加recycleAndFillItems(recycler, state);，因为在滑动过程中，需要重新对Item进行布局，即从缓存中取出Item进行数据绑定后放在新出现的Item的位置上。并且，还需要在scrollVerticallyBy最开始调用detachAndScrapAttachedViews(recycler);代码如下：

@Overridepublic int scrollVerticallyBy(int dy, RecyclerView.Recycler recycler, RecyclerView.State state) {

//先detach掉所有的子View

detachAndScrapAttachedViews(recycler);

//实际要滑动的距离

int travel = dy;

//如果滑动到最顶部

if (verticalScrollOffset + dy < 0) {

travel = -verticalScrollOffset;

} else if (verticalScrollOffset + dy > totalHeight - getVerticalSpace()) {//如果滑动到最底部

travel = totalHeight - getVerticalSpace() - verticalScrollOffset;

}

//将竖直方向的偏移量+travel

verticalScrollOffset += travel;

// 平移容器内的item

offsetChildrenVertical(-travel);

recycleAndFillItems(recycler, state);

Log.d("--->", " childView count:" + getChildCount());

return travel;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

看看我们的效果，主要关注打印的日志信息里面的子View的数量，确保确实是循环利用了子View