



基础
学JAVA



(/zxt0601) zxt0601 (/zxt0601)

● 博客专家

【Android】掌握自定义LayoutManager(二) 实现流式布局

发表于2016/10/28 17:58:17 12414人阅读

分类：Android RecyclerView家族 自定义LayoutManager

本篇文章已授权微信公众号 guolin_blog（郭霖）独家发布

转载请标明出处：

<http://blog.csdn.net/zxt0601/article/details/52956504> (<http://blog.csdn.net/zxt0601/article/details/52956504>)

本文出自：【张旭童的博客】 (<http://blog.csdn.net/zxt0601>)

本系列文章相关代码传送门：

自定义LayoutManager实现的流式布局 (<https://github.com/mcxtzhang/FlowLayoutManager>)

欢迎star，pr，issue。

本系列文章目录：

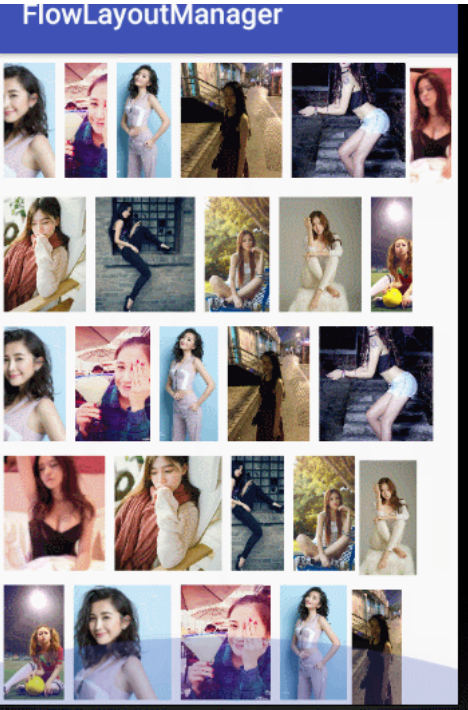
掌握自定义LayoutManager(一) 系列开篇 常见误区、问题、注意事项，常用API。 (<http://blog.csdn.net/zxt0601/article/details/52948009>)

掌握自定义LayoutManager(二) 实现流式布局 (<http://blog.csdn.net/zxt0601/article/details/52956504>)

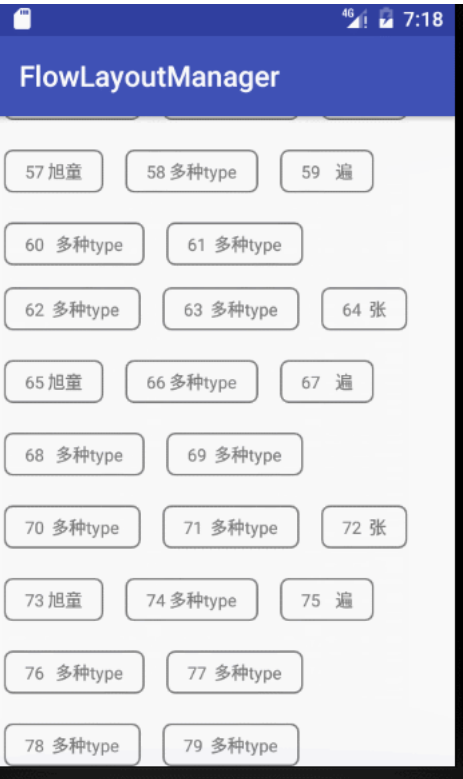


一 概述

在开始之前，我想说，如果需求是每个Item宽高一样，实现起来复杂度比每个Item宽高不一样的，要小10+倍。
然而我们今天要实现的流式布局，恰巧就是至少每个Item的宽度不一样，所以在计算坐标的时候算的我死去活来。先看一下效果图：



艾玛，换成妹子图后貌似好看了许多，我都不认识它了，好吧，项目里它一般长下面这样：



往常这种效果，我们一般使用自定义ViewGroup实现，我以前也写了一个。自定义VG实

现流式布局 (<http://blog.csdn.net/zxt0601/article/details/50533658>)

这不最近再研究自定义LayoutManager么，想来想去也没有好的创意，就先拿它开第一刀吧。

（后话：流式布局Item宽度不一，不知不觉给自己挖了个大坑，造成拓展一些功能难度倍增，观之网上的DEMO，99%Item的大小都是一样的，so，这个系列的下一篇我计划实现一个Item大小一样的酷炫LayoutManager。但是最终做成啥样的效果还没想好，有朋友看到酷炫的效果可以告诉我，我去高仿一个。）

自定义LayoutManager的步骤：

以本文的流式布局为例，需求是一个垂直滚动的布局，子View以流式排列。先总结一下步骤：

- 一 实现 generateDefaultLayoutParams()
- 二 实现 onLayoutChildren()
- 三 竖直滚动需要 重写 canScrollVertically() 和 scrollVerticallyBy()

下面我们就一步一步来吧。

二 实现generateDefaultLayoutParams()

如果没有特殊需求，大部分情况下，我们只需要如下重写该方法即可。

```
1  @Override
2  public RecyclerView.LayoutParams generateDefaultLayoutParams() {
3      return new RecyclerView.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT);
4  }
```

RecyclerView.LayoutParams 是继承自 android.view.ViewGroup.MarginLayoutParams 的，所以可以方便的使用各种margin。

这个方法最终会在 recycler.getViewForPosition(i) 时调用到，在该方法浩长源码的最下方：



```
1      final ViewGroup.LayoutParams lp = holder.itemView.getLayoutParams();
2      final LayoutParams rvLayoutParams;
3      if (lp == null) {
4          //这里会调用mLayout.generateDefaultLayoutParams()为每个ItemView设置LayoutParams
5          rvLayoutParams = (LayoutParams) generateDefaultLayoutParams();
6          holder.itemView.setLayoutParams(rvLayoutParams);
7      } else if (!checkLayoutParams(lp)) {
8          rvLayoutParams = (LayoutParams) generateLayoutParams(lp);
9          holder.itemView.setLayoutParams(rvLayoutParams);
10     } else {
11         rvLayoutParams = (LayoutParams) lp;
12     }
13     rvLayoutParams.mViewHolder = holder;
14     rvLayoutParams.mPendingInvalidate = fromScrap && bound;
15     return holder.itemView;
```

重写完这个方法就能编译通过了，只不过然并卵，界面上是一片空白，下面我们就走进 `onLayoutChildren()` 方法，为界面添加Item。

注：99%用不到的情况：如果需要存储一些额外的东西在 `LayoutParams` 里，这里返回你自己定义的 `LayoutParams` 即可。

当然，你自定义的 `LayoutParams` 需要继承自 `RecyclerView.LayoutParams`。

三 onLayoutChildren()

该方法是 `LayoutManager` 的入口。它会在如下情况下被调用：

- 1 在 `RecyclerView` 初始化时，会被调用两次。
- 2 在调用 `adapter.notifyDataSetChanged()` 时，会被调用。
- 3 在调用 `setAdapter` 替换 `Adapter` 时，会被调用。
- 4 在 `RecyclerView` 执行动画时，它也会被调用。

即 `RecyclerView` **初始化**、**数据源改变时** 都会被调用。

(关于初始化时为什么会被调用两次，我在系列第一篇文章里已经分析过。(http://blog.csdn.net/zxt0601/article/details/52948009))

在系列开篇我已经提到，它相当于 `ViewGroup` 的 `onLayout()` 方法，所以我们需要在里面 `layout` 当前屏幕可见的所有子 `View`，**千万不要layout出所有的子View**。本文如下编写：



```
1 private int mVerticalOffset;//竖直偏移量 每次换行时，要根据这个offset判断
2 private int mFirstVisiPos;//屏幕可见的第一个View的Position
3 private int mLastVisiPos;//屏幕可见的最后一个View的Position
4 @Override
5 public void onLayoutChildren(RecyclerView.Recycler recycler, RecyclerView.State state) {
6     if (getItemCount() == 0) { //没有Item，界面空着吧
7         detachAndScrapAttachedViews(recycler);
8         return;
9     }
10    if (getChildCount() == 0 && state.isPreLayout()) { //state.isPreLayout()是支持动画的
11        return;
12    }
13    //onLayoutChildren方法在RecyclerView 初始化时 会执行两遍
14    detachAndScrapAttachedViews(recycler);
15    //初始化
16    mVerticalOffset = 0;
17    mFirstVisiPos = 0;
18    mLastVisiPos = getItemCount();
19
20    //初始化时调用 填充childView
21    fill(recycler, state);
22 }
```

这个 `fill(recycler, state)` 方法将是你自定义LayoutManager之旅一生的敌人，简单的说它承担了以下任务：

在考虑滑动位移的情况下：

1 回收所有屏幕不可见的子View

2 layout所有可见的子View

在这一节，我们先看一下它的简单版本，不考虑滑动位移，不考虑滑动方向等，只考虑初始化时，**从头至尾**，layout所有可见的子View，在下一节我会配合滑动事件放出它的完整版。



```

1      int topOffset = getPaddingTop(); //布局时的上偏移
2      int leftOffset = getPaddingLeft(); //布局时的左偏移
3      int lineMaxHeight = 0; //每一行最大的高度
4      int minPos = mFirstVisiPos; //初始化时，我们不清楚究竟要layout多少个子View，所以就假设从0~itemcount-1
5      mLastVisiPos = getItemCount() - 1;
6      //顺序addChildView
7      for (int i = minPos; i <= mLastVisiPos; i++) {
8          //找recycler要一个childItemView,我们不管它是从scrap里取，还是从RecyclerViewPool里取，亦或是onCreateViewHolder里拿。
9          View child = recycler.getViewForPosition(i);
10         addView(child);
11         measureChildWithMargins(child, 0, 0);
12         //计算宽度 包括margin
13         if (leftOffset + getDecoratedMeasurementHorizontal(child) <= getHorizontalSpace()) { //当前行还排列的下
14             layoutDecoratedWithMargins(child, leftOffset, topOffset, leftOffset + getDecoratedMeasurementHorizontal(child), topOffset + getDecorate
15
16             //改变 left lineHeight
17             leftOffset += getDecoratedMeasurementHorizontal(child);
18             lineMaxHeight = Math.max(lineMaxHeight, getDecoratedMeasurementVertical(child));
19         } else { //当前行排列不下
20             //改变top left lineHeight
21             leftOffset = getPaddingLeft();
22             topOffset += lineMaxHeight;
23             lineMaxHeight = 0;
24
25             //新起一行的时候要判断一下边界
26             if (topOffset - dy > getHeight() - getPaddingBottom()) {
27                 //越界了 就回收
28                 removeAndRecycleView(child, recycler);
29                 mLastVisiPos = i - 1;
30             } else {
31                 layoutDecoratedWithMargins(child, leftOffset, topOffset, leftOffset + getDecoratedMeasurementHorizontal(child), topOffset + getDeco
32
33                 //改变 left lineHeight
34                 leftOffset += getDecoratedMeasurementHorizontal(child);
35                 lineMaxHeight = Math.max(lineMaxHeight, getDecoratedMeasurementVertical(child));
36             }
37         }
38     }

```

用到的一些工具函数（在系列开篇已介绍过）：



```
1 //模仿LLM Horizontal 源码
2
3 /**
4  * 获取某个childView在水平方向所占的空间
5  *
6  * @param view
7  * @return
8  */
9 public int getDecoratedMeasurementHorizontal(View view) {
10     final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams)
11         view.getLayoutParams();
12     return getDecoratedMeasuredWidth(view) + params.leftMargin
13         + params.rightMargin;
14 }
15
16 /**
17  * 获取某个childView在竖直方向所占的空间
18  *
19  * @param view
20  * @return
21  */
22 public int getDecoratedMeasurementVertical(View view) {
23     final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams)
24         view.getLayoutParams();
25     return getDecoratedMeasuredHeight(view) + params.topMargin
26         + params.bottomMargin;
27 }
28
29 public int getVerticalSpace() {
30     return getHeight() - getPaddingTop() - getPaddingBottom();
31 }
32
33 public int getHorizontalSpace() {
34     return getWidth() - getPaddingLeft() - getPaddingRight();
35 }
```

如上编写一个超级简单的 `fill()` 方法，运行，你的程序应该就能看到流式布局的效果出现了。

可是千万别开心，因为痛苦的计算远没到来。

如果这些都看不懂，那么我建议：

一，直接下载完整代码，配合后面的章节看，看到后面也许前面的就好理解了= =。

二，去学习一下自定义ViewGroup的知识。

此时虽然界面上已经展示了流式布局的效果，可是它并不能滑动，下一节我们让它动起来。

四，动起来



想让我们自定义的LayoutManager动起来，最简单的写法如下：

```
1      @Override
2      public boolean canScrollVertically() {
3          return true;
4      }
5
6      @Override
7      public int scrollVerticallyBy(int dy, RecyclerView.Recycler recycler, RecyclerView.State state) {
8          int realOffset = dy;//实际滑动的距离，可能会在边界处被修复
9
10         offsetChildrenVertical(-realOffset);
11
12         return realOffset;
13     }
```

`offsetChildrenVertical(-realOffset);` 这句话移动所有的childView.

返回值会被RecyclerView用来判断是否达到边界，如果返回值！=传入的dy，则会有一个边缘的发光效果，表示到达了边界。而且返回值还会被RecyclerView用于计算fling效果。

写完编译，哇塞，真的跟随手指滑动了，只不过能动的总共就我们在上一节layout的那些Item，Item并没有回收，也没有新的Item出现。

好了，下面开始正经的写它吧，




```
1      @Override
2      public int scrollVerticallyBy(int dy, RecyclerView.Recycler recycler, RecyclerView.State state) {
3          //位移0、没有子View 当然不移动
4          if (dy == 0 || getChildCount() == 0) {
5              return 0;
6          }
7
8          int realOffset = dy; //实际滑动的距离，可能会在边界处被修复
9          //边界修复代码
10         if (mVerticalOffset + realOffset < 0) { //上边界
11             realOffset = -mVerticalOffset;
12         } else if (realOffset > 0) { //下边界
13             //利用最后一个子View比较修正
14             View lastChild = getChildAt(getChildCount() - 1);
15             if (getPosition(lastChild) == getItemCount() - 1) {
16                 int gap = getHeight() - getPaddingBottom() - getDecoratedBottom(lastChild);
17                 if (gap > 0) {
18                     realOffset = -gap;
19                 } else if (gap == 0) {
20                     realOffset = 0;
21                 } else {
22                     realOffset = Math.min(realOffset, -gap);
23                 }
24             }
25         }
26     }
27
28     realOffset = fill(recycler, state, realOffset); //先填充，再位移。
29
30     mVerticalOffset += realOffset; //累加实际滑动距离
31
32     offsetChildrenVertical(-realOffset); //滑动
33
34     return realOffset;
35 }
```

这里用 `realOffset` 变量保存实际的位移，也是 `return` 回去的值。大部分情况下它=`dy`。在边界处，为了防止越界，做了一些处理，`realOffset` 可能不等于`dy`。
和别的文章不同的是，我参考了 `LinearLayoutManager` 的源码，**先考虑滑动位移**进行View的回收、填充(`fill()` 函数)，**然后再真正的位移**这些子Item。

在 `fill()` 的过程中

流程：

- 一 会先**考虑到dy**，回收界面上不可见的Item。
- 二 **填充**布局子View
- 三 判断是否将`dy`都消费掉了，如果消费不掉：例如滑动距离太多，**屏幕上的View已经填充完了，仍有空白**，那么就要修正`dy`给`realOffset`。



注意事项一：考虑滑动的方向

在填充布局子View的时候，还要考虑滑动的方向，即填充的顺序，是从头至尾填充，还是从尾至头部填充。

如果是向底部滑动，那么是顺序填充，显示底端position更大的Item。（ $dy > 0$ ）

如果是向顶部滑动，那么是逆序填充，显示顶端position更小的Item。（ $dy < 0$ ）

注意事项二：流式布局 逆序布局子View的问题

再啰嗦最后一点,我们想象一下这个逆序填充的过程：

正序过程可以**自上而下，自左向右layout 子View**，每次layout之前判断当前这一行宽度+子View宽度，是否超过父控件宽度，如果超过了就另起一行。

逆序时，有两种方案：

1 利用Rect保存子View边界

**正序排列时，保存每个子View的Rect，
逆序时，直接拿出来，layout。**

2 逆序化

自右向左layout子View，每次layout之前判断当前这一行宽度+子View宽度，是否超过父控件宽度，

如果超过了就另起一行。并且判断最后一个子View距离父控件左边的offset，平移这一行的所有子View，较复杂，采用方案1.

（我个人认为这两个方案都不太好，希望有朋友能提出更好的方案。）

下面上码：



```

1 private SparseArray<Rect> mItemRects;//key 是View的position，保存View的bounds，
2 /**
3  * 填充childView的核心方法,应该先填充，再移动。
4  * 在填充时，预先计算dy的在内，如果View越界，回收掉。
5  * 一般情况是返回dy，如果出现View数量不足，则返回修正后的dy。
6  *
7  * @param recycler
8  * @param state
9  * @param dy RecyclerView给我们的位移量,+,显示底端，-,显示头部
10 * @return 修正以后真正的dy（可能剩余空间不够移动那么多了 所以return <|dy|）
11 */
12 private int fill(RecyclerView.Recycler recycler, RecyclerView.State state, int dy) {
13
14     int topOffset = getPaddingTop();
15
16     //回收越界子View
17     if (getChildCount() > 0) { //滑动时进来的
18         for (int i = getChildCount() - 1; i >= 0; i--) {
19             View child = getChildAt(i);
20             if (dy > 0) { //需要回收当前屏幕，上越界的View
21                 if (getDecoratedBottom(child) - dy < topOffset) {
22                     removeAndRecycleView(child, recycler);
23                     mFirstVisiPos++;
24                     continue;
25                 }
26             } else if (dy < 0) { //回收当前屏幕，下越界的View
27                 if (getDecoratedTop(child) - dy > getHeight() - getPaddingBottom()) {
28                     removeAndRecycleView(child, recycler);
29                     mLastVisiPos--;
30                     continue;
31                 }
32             }
33         }
34     }
35     //detachAndScrapAttachedViews(recycler);
36 }
37
38 int leftOffset = getPaddingLeft();
39 int lineMaxHeight = 0;
40 //布局子View阶段
41 if (dy >= 0) {
42     int minPos = mFirstVisiPos;
43     mLastVisiPos = getItemCount() - 1;
44     if (getChildCount() > 0) {
45         View lastView = getChildAt(getChildCount() - 1);
46         minPos = getPosition(lastView) + 1; //从最后一个View+1开始吧
47         topOffset = getDecoratedTop(lastView);
48         leftOffset = getDecoratedRight(lastView);
49         lineMaxHeight = Math.max(lineMaxHeight, getDecoratedMeasurementVertical(lastView));
50     }
51     //顺序addChildView
52     for (int i = minPos; i <= mLastVisiPos; i++) {
53         //找recycler要一个childItemView,我们不管它是从scrap里取，还是从RecyclerViewPool里取，亦或是onCreateViewHolder里拿。
54         View child = recycler.getViewForPosition(i);
55         addView(child);
56         measureChildWithMargins(child, 0, 0);
57         //计算宽度 包括margin
58         if (leftOffset + getDecoratedMeasurementHorizontal(child) <= getHorizontalSpace()) { //当前行还排列的下
59             layoutDecoratedWithMargins(child, leftOffset, topOffset, leftOffset + getDecoratedMeasurementHorizontal(child), topOffset + getDecorate
60
61             //保存Rect供逆序layout用
62             Rect rect = new Rect(leftOffset, topOffset + mVerticalOffset, leftOffset + getDecoratedMeasurementHorizontal(child), topOffset + getDec
63

```



```

64         mItemRects.put(i, rect);
65
66         //改变 left lineHeight
67         leftOffset += getDecoratedMeasurementHorizontal(child);
68         lineMaxHeight = Math.max(lineMaxHeight, getDecoratedMeasurementVertical(child));
69     } else { //当前行排列不下
70         //改变top left lineHeight
71         leftOffset = getPaddingLeft();
72         topOffset += lineMaxHeight;
73         lineMaxHeight = 0;
74
75         //新起一行的时候要判断一下边界
76         if (topOffset - dy > getHeight() - getPaddingBottom()) {
77             //越界了 就回收
78             removeAndRecycleView(child, recycler);
79             mLastVisiPos = i - 1;
80         } else {
81             layoutDecoratedWithMargins(child, leftOffset, topOffset, leftOffset + getDecoratedMeasurementHorizontal(child), topOffset + getDecoratedMeasurementVertical(child));
82
83             //保存Rect供逆序layout用
84             Rect rect = new Rect(leftOffset, topOffset + mVerticalOffset, leftOffset + getDecoratedMeasurementHorizontal(child), topOffset + getDecoratedMeasurementVertical(child));
85             mItemRects.put(i, rect);
86
87             //改变 left lineHeight
88             leftOffset += getDecoratedMeasurementHorizontal(child);
89             lineMaxHeight = Math.max(lineMaxHeight, getDecoratedMeasurementVertical(child));
90         }
91     }
92 }
93
94 //添加完后，判断是否已经没有更多的ItemView，并且此时屏幕仍有空白，则需要修正dy
95 View lastChild = getChildAt(getChildCount() - 1);
96 if (getPosition(lastChild) == getItemCount() - 1) {
97     int gap = getHeight() - getPaddingBottom() - getDecoratedBottom(lastChild);
98     if (gap > 0) {
99         dy -= gap;
100     }
101 }
102
103 } else {
104     /**
105     * ## 利用Rect保存子View边界
106     正序排列时，保存每个子View的Rect，逆序时，直接拿出来layout。
107     */
108     int maxPos = getItemCount() - 1;
109     mFirstVisiPos = 0;
110     if (getChildCount() > 0) {
111         View firstView = getChildAt(0);
112         maxPos = getPosition(firstView) - 1;
113     }
114     for (int i = maxPos; i >= mFirstVisiPos; i--) {
115         Rect rect = mItemRects.get(i);
116
117         if (rect.bottom - mVerticalOffset - dy < getPaddingTop()) {
118             mFirstVisiPos = i + 1;
119             break;
120         } else {
121             View child = recycler.getViewForPosition(i);
122             addView(child, 0); //将View添加至RecyclerView中，childIndex为1，但是View的位置还是由layout的位置决定
123             measureChildWithMargins(child, 0, 0);
124         }
125     }
126 }

```



```
127         layoutDecoratedWithMargins(child, rect.left, rect.top - mVerticalOffset, rect.right, rect.bottom - mVerticalOffset);
128     }
129 }
130 }
131
132
Log.d("TAG", "count= [" + getChildCount() + "]" + ",[recycler.getScrapList().size():" + recycler.getScrapList().size() + ", dy:" + dy + ", mVerticalOffset" + mV

return dy;
}
```

思路已经在前面讲解过，代码里也配上了注释，计算坐标等都是数学问题，略饶人，需要用笔在纸上写一写，或者运行调试调试。没啥好办法。

值得一提的是，可以通过 `getChildCount()` 和 `recycler.getScrapList().size()` 查看当前屏幕上的Item数量和 `scrapCache`缓存区域的Item数量，合格的LayoutManager，`childCount`数量不应大于屏幕上显示的Item数量，而`scrapCache`缓存区域的Item数量应该是0。

官方的LayoutManager都是达标的，本例也是达标的，网上大部分文章的Demo，都是不合格的。。

原因在系列开篇也提过，不再赘述。（<http://blog.csdn.net/zxt0601/article/details/52948009>）

至此我们的自定义LayoutManager已经可以用了，使用的效果就和文首的两张图一模一样。

下面再提及一些其他注意点和适配事项：

五 适配notifyDataSetChanged()

此时会回调`onLayoutChildren()`函数。因为我们流式布局的特殊性，每个Item的宽度不一致，所以化简处理，每次这里归零。

```
1 //初始化区域
2 mVerticalOffset = 0;
3 mFirstVisiPos = 0;
4 mLastVisiPos = getItemCount();
```

如果每个Item的大小都一样，逆序顺序`layoutChild`都比较好处理，则应该在此判断，`getChildCount()`,大于0说明是`DatasetChanged()`操作，（初始化的第二次也会`childCount>0`）。根据当前记录的`position`和位移信息去fill视图即可。

六 适配 Adapter的替换。

我根据24.2.1源码，发现网上的资料对这里的处理其实是不必要的。

一 资料中的做法如下：

当对RecyclerView设置一个新的Adapter时，onAdapterChanged() 方法会被回调，一般的做法是在这里remove掉所有的View。此时 onLayoutChildren() 方法会被再次调用，一个新的轮回开始。

```
1      @Override
2      public void onAdapterChanged(final RecyclerView.Adapter oldAdapter, final RecyclerView.Adapter newAdapter) {
3          removeAllViews();
4      }
```

二 我的新观点：

通过查看源码+打断点跟踪分析，调用RecyclerView.setAdapter后，调用顺序依次为

1 RecyclerView.setAdapter():

```
1      public void setAdapter(Adapter adapter) {
2          // bail out if layout is frozen
3          setLayoutFrozen(false);
4          setAdapterInternal(adapter, false, true); //张旭童注：注意第三个参数是true
5          requestLayout();
6      }
```

那么我们查看 setAdapterInternal() 方法：



```
1 private void setAdapterInternal(Adapter adapter, boolean compatibleWithPrevious,
2     boolean removeAndRecycleViews) {
3     ...
4     //张旭童注: removeAndRecycleViews 参数此时为ture
5     if (!compatibleWithPrevious || removeAndRecycleViews) {
6         ...
7         if (mLayout != null) {
8             //张旭童注: 所以如果我们更换Adapter时, mLayout不为空, 会先执行如下操作,
9             mLayout.removeAndRecycleAllViews(mRecycler);
10            mLayout.removeAndRecycleScrapInt(mRecycler);
11        }
12        // we should clear it here before adapters are swapped to ensure correct callbacks.
13        //张旭童注: 而且还会清空Recycler的缓存
14        mRecycler.clear();
15    }
16    ...
17    if (mLayout != null) {
18        //张旭童注: 这里才调用的LayoutManager的方法
19        mLayout.onAdapterChanged(oldAdapter, mAdapter);
20    }
21    //张旭童注: 这里调用Recycler的方法
22    mRecycler.onAdapterChanged(oldAdapter, mAdapter, compatibleWithPrevious);
23    ...
24 }
```

也就是说 **更换Adapter一开始** , 还没有执行到 `LayoutManager.onAdapterChanged()` , **界面上的View都已经被remove掉了** , 我们的操作属于**多余的**。

2 LayoutManager.onAdapterChanged()

空实现：**也没必要实现了**

```
1 public void onAdapterChanged(Adapter oldAdapter, Adapter newAdapter) {
2 }
```

3 Recycler.onAdapterChanged() :

该方法先清空scapCache区域（貌似也是多余，一开始被清空过了），然后调用 `Recycler ViewPool.onAdapterChanged()`

。



```
1      void onAdapterChanged(Adapter oldAdapter, Adapter newAdapter,
2          boolean compatibleWithPrevious) {
3          clear();
4          getRecycledViewPool().onAdapterChanged(oldAdapter, newAdapter, compatibleWithPrevious);
5      }
6
7      public void clear() {
8          mAttachedScrap.clear();
9          recycleAndClearCachedViews();
10     }
11
```

4 RecyclerViewPool.onAdapterChanged()

如果没有别的Adapter在用这个RecyclerViewPool，会清空RecyclerViewPool的缓存。

```
1      void onAdapterChanged(Adapter oldAdapter, Adapter newAdapter,
2          boolean compatibleWithPrevious) {
3          if (oldAdapter != null) {
4              detach();
5          }
6          if (!compatibleWithPrevious && mAttachCount == 0) {
7              clear();
8          }
9          if (newAdapter != null) {
10             attach(newAdapter);
11         }
12     }
```


5 LayoutManager.onLayoutChildren()

新的布局开始。

七 总结：

引用一段话

They are also extremely complex, and hard to get right. For every amount of effort RecyclerView requires of you, it is doing 10x more behind the scenes.

本文Demo仍有很大完善空间，有些需要完善的细节非常复杂，需要经过多次试验才能得到正确的结果（这里我更加敬佩Google提供的三个LM）。每一个我们想要实现的需求可能要花费比我们想象的时间*10倍的时间。
上篇也提及到的，不要过度优化，达成需求就好。

可以通过 `getChildCount()` 和 `recycler.getScrapList().size()` 查看当前屏幕上的Item数量和 `scrapCache`缓存区域的Item数量，合格的LayoutManager，`childCount`数量不应大于屏幕上显示的Item数量，而`scrapCache`缓存区域的Item数量应该是0。
官方的LayoutManager都是达标的，本例也是达标的，网上大部分文章的Demo，都是不合格的。。

感兴趣的同学可以对网上的各个Demo打印他们`onCreateViewHolder`执行的次数，以及上述两个参数的值，和官方的LayoutManager比较，这三个参数先达标，才算是**及格**的LayoutManager，但后续优化之路仍很长。

本系列文章相关代码传送门：
自定义LayoutManager实现的流式布局 (<https://github.com/mcxtzhang/FlowLayoutManager>)
欢迎star，pr，issue。

👍 16

🗨 1


上一篇 (/zxt0601/article/details/52948009)

下一篇 (/zxt0601/article/details/53040506)

正规装修公司装修报价高在哪

电动按摩椅价格

评论 (28)




(/blog/index?username=DT235201314)DT235201314

学习了，谢分享


2017-09-25 20:17

18楼



回复


(/comm

 (/blog/index?username=tp_waiwai)tp_waiwai

17楼

当recycleView嵌套recycleView时，作为嵌套的recycleView的LayoutManager时，显示有bug
2017-08-23 14:14

回复

 (/blog/index?username=a591193965)a591193965

16楼

有一个小问题，当RecyclerView.Adapter调用notifyDataChange时，recyclerview会滚动到顶部。
2017-07-20 11:25

回复

[查看全部评论 \(/comment/alllist?id=52956504\)](/comment/alllist?id=52956504) [发表评论 \(/comment/post?id=52956504\)](/comment/post?id=52956504)

1 电热棒	5 博客搬家
2 学国画	6 搬家公司司
3 搬家 物流	7 按摩椅
4 附近的物流公	8 书法班

相关博文

- Android 自定义ViewGroup 实战篇 -> 实现FlowLayout (<http://blog.csdn.net/lmj623565791/article/details/7024...>)
- 100行Android代码自定义一个流式布局 - FlowLayout (<http://blog.csdn.net/FeeLang/article/details/438...>)
- Android自定义控件--流式布局(FlowLayout)--自动适配 (<http://blog.csdn.net/MyLoveyaqiong/article/>) 
- Android 实现FlowLayout流式布局（类似热门标签） (<http://blog.csdn.net/wjr1949/article/details/7024...>)

Android 流式布局FlowLayout 实现关键字标签 (http://blog.csdn.net/kong_gu_you_lan/article/details/5...)

android流式布局：FlexboxLayout用法探析(一) (<http://blog.csdn.net/tabolt/article/details/51799226>)

Android流式布局的简单实现 (<http://blog.csdn.net/zhoujiadick/article/details/47722039>)

Android流式布局FlowLayout (<http://blog.csdn.net/cdecde111/article/details/53019894>)

Android从零开搞系列：自定义View（10）流式布局 (<http://blog.csdn.net/wjzj000/article/details/65936...>)

Android中常见的热门标签的流式布局的实现 (<http://blog.csdn.net/jdsjlzx/article/details/45042081>)

1 博客搬家	5 安卓大会
2 平安自定义车	6 三室一厅装
3 按摩椅厂家	7 胖虎
4 书法	8 搬家公司

我的热门文章

【Android】详解7.0带来的新工具类：DiffUtil (</zxt0601/article/details/52562770>)

【Android】ListView、RecyclerView、ScrollView里嵌套ListView 相对优雅的解决方案:NestFullListVi...

【Android 仿微信通讯录 导航分组列表-上】使用ItemDecoration为RecyclerView打造带悬停头部的分...

五行代码实现 炫动滑动 卡片层叠布局，仿探探、人人影视订阅界面 简单&优雅：LayoutManager+Ite...

【Android】RecyclerView、ListView实现单选列表的优雅之路. (</zxt0601/article/details/52703280>)

