(/zxt0601) zxt0601 (/zxt0601) ● 博客专家

【Android】掌握自定义LayoutManager(一) 系列开篇 常见误区、问题、注意事项，常用API。

发表于2016/10/27 23:29:14 8087人阅读
分类： Android RecyclerView家族 自定义LayoutManager

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：

概述

这篇文章是深入掌握自定义LayoutManager系列的开篇，是一份总结报告。部分内容不属于引言、过于深入，用作系列后续文章的参考，以及浏览完后的复习之用。

本文内容涉及RecyclerView、LayoutManager、RecyclerViewPool、Recycler。

注：
1 以下问题，初学者如有不理解的，可以不用太在意，等学习完自定义LayoutManager相关知识，写几个Demo再回来看更好理解。
2 在RecyclerView中，ItemView和ViewHolder其实是一一绑定的，所以提到的View = ViewHolder。

一 常见误区、问题、注意事项：

在自定义LayoutManager文章开始之前，我总结了一些我在学习以及阅读别人的文章、编码的过程中，遇到的一些疑惑问题，并附上**我个人的理解与答案。欢迎拍砖讨论。**

因网上有大量半吊子写的LayoutManager相关的**中文**文章。（包括我也是半吊子），所以很多文章看完了，心中都有N个疑问，如，作者好牛逼啊，但是为什么我独立写还是写不出来。 自定义一个LayoutManager就自动复用了吗？…等等，下面逐个来讲讲。

Q1 看完了，但是我独立写还是不知道怎么写。

A1： 自定义LayoutManager是一项**颇有难度**的工程，你很难仅仅阅读一两篇文章，花两三个小时就能学习完。
里面涉及到子View的布局，坐标的计算，偏移量的计算，在滑动时、在合适的时机回收屏幕上不再显示的View，如何判断这些View是在屏幕上不可见，以及View究竟是暂时detach掉，还是recycle回收掉…等大量问题
。老实说，也许我水平有限，这是我在学习Android过程中，耗时最久的几个知识点之一。（十几个小时才写出第一个及格的作品）
但是它值得你学习。所以独立写不出来别灰心，先仿照一个Demo写一写，如果用心理解，第二遍第二遍应该就可以独立完成了。

Q2 学习自定义LayoutManager需要的铺垫知识

一 ：熟练掌握自定义ViewGroup。

（在自定义LayoutManager过程的第一步，onLayoutChildren()方法里，就类似于自定义ViewGroup的onLayout()方法。）
但与自定义LayoutManager相比，自定义ViewGroup是一种静态的layout 子View的过程，因为ViewGroup内部不支持滑动，所以只需要无脑layout出所有的View，便不用再操心剩下的事。
而自定义LayoutManager与之不同，在第一步layout时，**千万不要layout出所有的子View**，这里也是网上一些文章里的错误做法，他们带着老思想，在第一步就layout出了所有的childView，这会导致一个很严重的问题：**你的自定义LayoutManager = 自定义ViewGroup。**即，他们**没有View复用机制。**
why？这里简单证明结论，在Q5的回答里会说明为什么。
在Adapter的onCreateViewHolder()方法里增加打印语句，如果你的数据源有100000条数据，那么在RecyclerView第一次显示在屏幕上时，onCreateViewHolder()会执行100000次，你就可以尽情的欣赏ANR了。
反观使用官方提供的三种LayoutManager，开始时屏幕上有n少个ItemView，一般就执行n次onCreateViewHolder()，（也有可能多执行1次），在后续滑动时，大部分情况都只是执行onBindViewHolder()方法，不会再执行onCreateViewHolder()。

二：熟练使用RecyclerView。这个不用多说，毕竟RecyclerView是LayoutManager的宿主。

其实会以上两点就可以开始我们的学习之旅了，不过如果能对RecyclerView的Adapter、RecyclerViewPool、ItemDecoration也有一定的了解那是最好。

Q3 自定义LayoutManager的实战场景多吗？

A3：实战场景还是相当**有限**的。系统自带的三个LayoutManager已经很够用，满足绝大部分需求。

我个人从学习自定义LayoutManager至今的收获 ，大部分是**对RecyclerView机制的理解进一步加深**，也会伴随一定量的源码阅读经验提升。随没有我想象中的提升巨大生产力的赶脚，因为很多时候，产品设计要求的布局，现有方案已经可以很好解决。

但是它值得学习。

Q4 自定义一个LayoutManager就自动复用ItemView了吗？

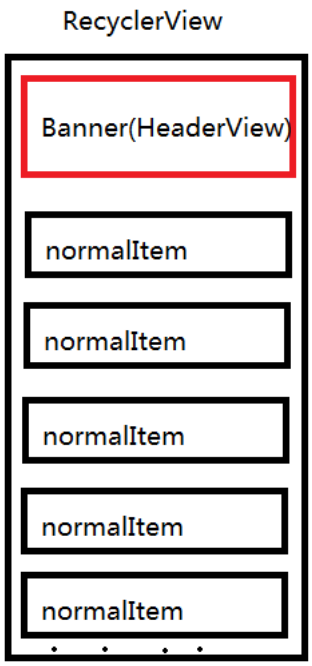
A4：**不是**，实际上这是自定义LayoutManager的重头戏之一，要做到在**合适的时机回收**不可见的旧子View ，**复用**子View layout 新的子View，以及Q2提及的在LayoutManager的**初始化时合理布局可见数量**的子View等，才算是复用了ItemView。
注意，这里的回收是recycle，而不是detach。
如果你只detach了ItemView，并没有recycle它们，它们会一直被保存在Recycler的`mAttachedScrap`里，它是一个ArrayList，保存了被detach但还没有recycle的ViewHolder。

```
1    public final class Recycler {
2        final ArrayList<ViewHolder> mAttachedScrap = new ArrayList<>();
```

（实际上Recycler内部的缓存机制远不止一个mAttachedScrap 。 ）

Q5 用RecyclerView就等于ItemView复用？

A5：显然也**不是**。除了Q4的因素外，这里还有一个很大的误区：很多人认为使用了RecyclerView，ItemView就都回收复用了。
这里出个题：基本上APP都有个TopBanner在，它放在RecyclerView里作为**HeaderView（通过特殊的ItemViewType实现）**，剩下都是**普通的ItemView**，那么列表滚动，当Banner早已不可见时，它的**View(ViewHolder)会被回收**、**被其他ItemView复用**吗？
如下图：



答案：Banner的ViewHolder **会被回收**，但该ViewHolder的内存空间 **不会被释放** ， **不会被其他的ItemView复用**。

回收都好理解，在屏幕上不可见时，LayoutManager会把它回收至RecyclerViewPool里。
然而却**不会给normalItem复用，因为它们的ItemViewType不同。**
所以它的内存空间不会被释放，将一直被RecyclerViewPool持有着，等待着需求相同ItemViewType的ViewHolder的请求到来。
即，当页面滚动回顶部，显示Banner时，这个View会被复用。
先说为什么，再说如何去验证。

为什么？

这涉及到Recycler、RecyclerViewPool的知识，（小安利，我在http://blog.csdn.net/zxt0601/article/details/52267325 (http://blog.csdn.net/zxt0601/article/details/52267325) 这篇文章的第四节里对RecyclerViewPool的源码进行过全解，不过大家也可以自己去查看，源码很短。）
在LayoutManager里，获取childView是通过如下方法得到：

```
1        View child = recycler.getViewForPosition(i);
```

该方法内部，先通过position去获取是否有detach掉的scrapView（ViewHolder），

```
1        holder = getScrapViewForPosition(position, INVALID_TYPE, dryRun);
```

如果没有则根据position去获取itemViewType，

```
1        final int type = mAdapter.getItemViewType(offsetPosition);
```

根据itemViewType获取在RecyclerViewPool里是否有该ViewHolder，

```
1        holder = getRecycledViewPool().getRecycledView(type);
```

这里由于我们的Banner的viewType和normalItem的**viewType不一样，即使Banner被回收进了RecyclerViewPool，但是由于itemViewtype和普通的ItemView不同，它也无法被取出、从而复用**，（发散一下，另外一点，**它也无法被释放**，被强引用在内存里，http://blog.csdn.net/zxt0601/article/details/52267325 (http://blog.csdn.net/zxt0601/article/details/52267325) 这篇文章有详细分析）。
再往下由于holder还是空的，最终便会调用Adapter的onCreateViewHolder()方法crea[illegible]个新的ViewHolder。

```
`holder = mAdapter.createViewHolder(RecyclerView.this, type);`
```

验证:

感兴趣的人去重写任意Adapter的`getItemViewType()`方法：

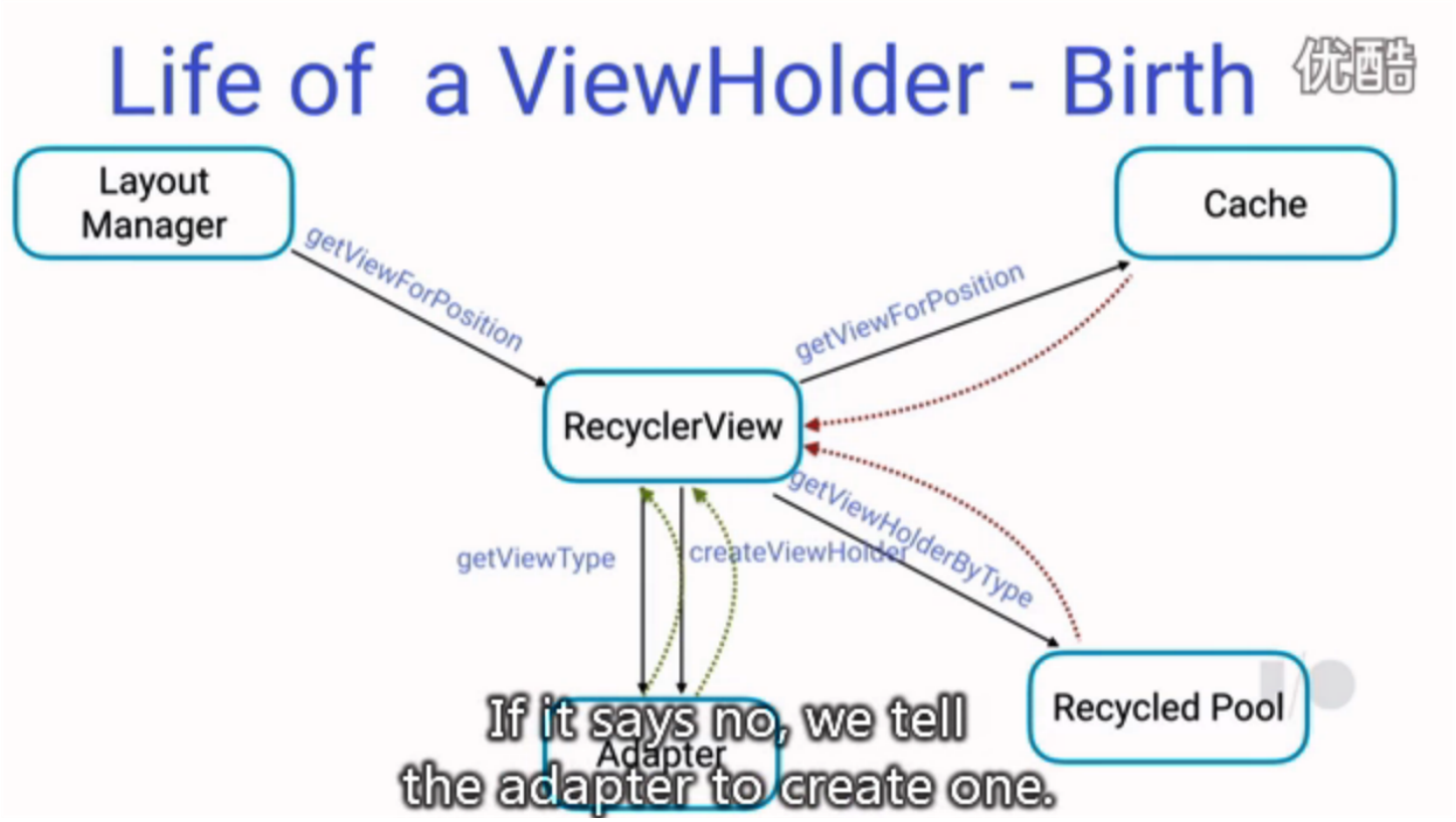
```
@Override
public int getItemViewType(int position) {
    return position;
}
```

这样每一个ItemViewType都不一样，RecyclerView不会有任何的复用，因为每一个ItemView在RecyclerViewPool里都找不到可以复用的holder，ItemView有n个，onCreateViewHolder方法会执行n次。

看到这里就能回答Q2一的问题：
因为在初始化时，**Recycler(scrapCache)和RecyclerViewPool里的缓存都是空的**，所以此时得到的ViewHolder都是**通过onCreateViewHolder(),new 出的ViewHolder**。如果此时get了**整个itemCount数量**的View，那么也会new出itemCount数量的ViewHolder，此时这些ViewHolder都存在内存里，和普通ViewGroup毫无分别，也更容易OOM。

Q6 RecyclerView的缓存机制简述

A6： 上面BB了这么多，涉及到Recycler、RecyclerViewPool以及scrap，detach，remove，recycle等概念。



这张图摘自（http://kymjs.com/code/2016/07/10/01 (http://kymjs.com/code/2016/07/10/01)），源头应该是Google官方的视频里。
我理解图上的cache是被detach掉的ViewHolder存放的区域，即scrapCache区域。
这个区域由

```
1               final ArrayList<ViewHolder> mAttachedScrap = new ArrayList<>();
2               ArrayList<ViewHolder> mChangedScrap = null;
3
4               final ArrayList<ViewHolder> mCachedViews = new ArrayList<ViewHolder>();
```

这三个ArrayList组成。
而被remove掉的ViewHolder会按照ViewType分组被存放在RecyclerViewPool里，默认最大缓存每组（ViewType）5个。

```
1               private SparseArray<ArrayList<ViewHolder>> mScrap =
2                       new SparseArray<ArrayList<ViewHolder>>();
```

Q7 detach 和recycle的时机。

一个View只是**暂时被清除掉，稍后立刻就要用到**，使用detach。它会被缓存进scrapCache的区域。
一个View **不再显示在屏幕上，需要被清除掉，并且下次再显示它的时机目前未知** ，使用remove。它会被以viewType分组，缓存进RecyclerViewPool里。
注意：一个View只被detach，没有被recycle的话，不会放进RecyclerViewPool里，会一直存在recycler的scrap 中。网上有人的Demo就是如此，因此View也没有被复用，有多少ItemCount，就会new出多少个ViewHolder。

Q8 初始化时，onLayoutChildren()为什么会执行两次？

答 ：参看RecyclerView源码，onLayoutChildren 会执行两次，一次RecyclerView的onMeasure() 一次onLayout()。

李菊福：RecyclerView的onMeasure(),会调用`dispatchLayoutStep2()`方法，该方法内部会调用 `mLayout.onLayoutChildren(mRecycler, mState);` ,这是第一次。如下：

```
@Override
    protected void onMeasure(int widthSpec, int heightSpec) {
            ......
            dispatchLayoutStep2();
            ......
    }
```

```
    /**
     * The second layout step where we do the actual layout of the views for the final state.
     * This step might be run multiple times if necessary (e.g. measure).
     */
    private void dispatchLayoutStep2() {
        .....
        mLayout.onLayoutChildren(mRecycler, mState);
        .....
    }
```


onLayout()方法会调用`dispatchLayout();`,该方法内部又调用了`dispatchLayoutStep2();`，这是第二次。

Q9 基于上个问题，我们要注意什么？

答：即使是在写onLayoutChildren()方法时，也要考虑将屏幕上的View（如果有），detach掉，否则屏幕初始化时，同一个position的ViewHolder，也会onCreateViewHolder两次。因此childCount也会翻倍。

最后也是最重要的

LayoutManager API 支持强大且复杂的布局回收，正因为它API强大，所以我们需要实现大量的代码才能完成功能。不要**过度封装、过度优化**你的代码，只要能完成你的需求即可。（**当然最基本的要求：ViewHolder复用 要满足**）
原话如下：

> *Before going any further, a warning is in order. The LayoutManager API allows powerful and complex layout recycling because it doesn't do much for you; these implementations involve a fair amount of code you have to write yourself. As with any project involving custom views, don't get caught in a trap of over-optimizing or over-generalizing your code. Build the features you need for the application use case you're concerned with.*

文章链接：http://wiresareobsolete.com/2014/09/building-a-recyclerview-layoutmanager-part-1/ (http://wiresareobsolete.com/2014/09/building-a-recyclerview-layoutmanager-part-1/)
该文章是我见过学习自定义LayoutManager最好的资料。

二 常用API：

布局API:

```
1  //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。
2  View view = recycler.getViewForPosition(xxx);  //获取postion为xxx的View
```

```
1  addView(view);//将View添加至RecyclerView中，
2  addView(child, 0);//将View添加至RecyclerView中，childIndex为0，但是View的位置还是由layout的位置决定，该方法在逆序layout子View时有大用
```

```
1  measureChildWithMargins(scrap, 0, 0);//测量View,这个方法会考虑到View的ItemDecoration以及Margin
```

```
1  //将ViewLayout出来，显示在屏幕上，内部会自动追加上该View的ItemDecoration和Margin。此时我们的View已经可见了
2  layoutDecoratedWithMargins(view, leftOffset, topOffset,
3                  leftOffset + getDecoratedMeasuredWidth(view),
4                  topOffset + getDecoratedMeasuredHeight(view));
```

回收API：

```
1  detachAndScrapAttachedViews(recycler);//detach轻量回收所有View
2  detachAndScrapView(view, recycler);//detach轻量回收指定View
3
4  // recycle真的回收一个View ，该View再次回来需要执行onBindViewHolder方法
5  removeAndRecycleView(View child, Recycler recycler)
6  removeAndRecycleAllViews(Recycler recycler);
```

```
detachView(view);//超级轻量回收一个View,马上就要添加回来
attachView(view);//将上个方法detach的View attach回来
recycler.recycleView(viewCache.valueAt(i));//detachView 后 没有attachView的话 就要真的回收掉他们
```

移动子ViewAPI:

```
offsetChildrenVertical(-dy); // 竖直平移容器内的item
offsetChildrenHorizontal(-dx);//水平平移容器内的item
```

工具API：

```
public int getPosition(View view)//获取某个view 的 layoutPosition，很有用的方法，却鲜(没)有文章提及，是我翻看源码找到的。
```

```
//以下方法会我们考虑ItemDecoration的存在，但部分函数没有考虑margin的存在
getDecoratedLeft(view)=view.getLeft()
getDecoratedTop(view)=view.getTop()
getDecoratedRight(view)=view.getRight()
getDecoratedBottom(view)=view.getBottom()
getDecoratedMeasuredHeight(view)=view.getMeasuredWidth()
getDecoratedMeasuredHeight(view)=view.getMeasuredHeight()
```

```
//由于上述方法没有考虑margin的存在，所以我参考LinearLayoutManager的源码：
    /**
   * 获取某个childView在水平方向所占的空间
   *
   * @param view
   * @return
   */
    public int getDecoratedMeasurementHorizontal(View view) {
        final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams)
                view.getLayoutParams();
        return getDecoratedMeasuredWidth(view) + params.leftMargin
                + params.rightMargin;
    }

    /**
   * 获取某个childView在竖直方向所占的空间
   *
   * @param view
   * @return
   */
    public int getDecoratedMeasurementVertical(View view) {
        final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams)
                view.getLayoutParams();
        return getDecoratedMeasuredHeight(view) + params.topMargin
                + params.bottomMargin;
    }
```

19 0

上一篇 (/zxt0601/article/details/52848004) 下一篇 (/zxt0601/article/details/52956504)

评论（8）

(/blog/index?username=nullnulln)nullnulln 5楼

写的狠详细，感谢大神

2017-09-04 12:27 回复

(/blog/index?username=freelander_j)freelander_j 4楼

感谢博主分享

2016-12-21 16:58 回复

(/blog/index?username=u014163726)u014163726 3楼

写的真的很好，能看出来非常用心，回头看了看自己写的，很汗颜

2016-11-01 15:08 回复

查看全部评论 (/comment/alllist?id=52948009) 发表评论 (/comment/post?id=52948009)

相关博文

自定义LayoutManager的详解及其使用 (http://blog.csdn.net/lylodyf/article/details/52846602)

Android 掌握自定义LayoutManager(二) 实现流式布局 (http://blog.csdn.net/qq_27489007/article/detai…

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

Android RecyclerView 使用完全解析 体验艺术般的控件 (http://blog.csdn.net/lmj623565791/article/det…

RecyclerView自定义LayoutManager,打造不规则布局 (http://blog.csdn.net/qibin0506/article/details/52…

【Android】掌握自定义LayoutManager(二) 实现流式布局 (http://blog.csdn.net/zxt0601/article/details/…

Android 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…

RecyclerView——实现自定义LayoutManager (http://blog.csdn.net/qq_31370269/article/details/52932…

打造属于你的LayoutManager (http://blog.csdn.net/u014768339/article/details/51643798)

针对RecyclerView打造属于你的LayoutManager (http://blog.csdn.net/scott2017/article/details/516016…

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)