Android ListView 与 RecyclerView 对比浅析—缓存机制

一，背景

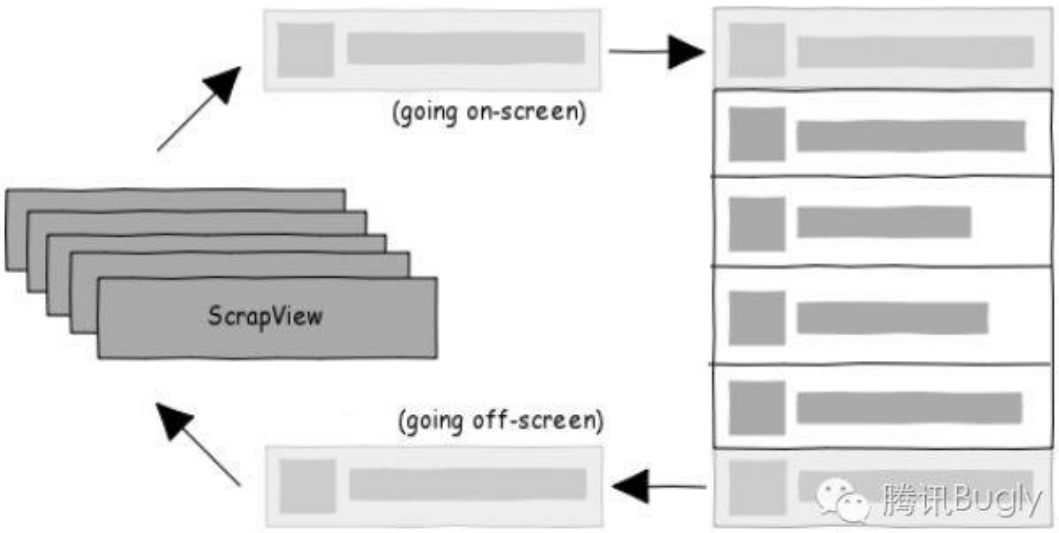
RecyclerView 是谷歌官方出的一个用于大量数据展示的新控件，可以用来代替传统的 ListView，更加强大和灵活。

最近，自己负责的业务，也遇到这样的一个问题，关于是否要将 ListView 替换为 RecyclerView?

秉承着实事求是的作风，弄清楚 RecyclerView 是否有足够的吸引力替换掉 ListView，我从性能这一角度出发，研究 RecyclerView 和 ListView 二者的缓存机制，并得到了一些较有益的”结论”，待我慢慢道来。

同时也希望能通过本文，让大家快速了解 RecyclerView 与 ListView 在缓存机制上的一些区别，在使用上也更加得心应手吧。

PS：相关知识：
ListView 与 RecyclerView 缓存机制原理大致相似，如下图所示：



过程中，离屏的 ItemView 即被回收至缓存，入屏的 ItemView 则会优先从缓存中获取，只是 ListView 与 RecyclerView 的实现细节有差异.（这只是缓存使用的其中一个场景，还有如刷新等）

PPS：本文不贴出详细代码，结合源码食用更佳!

二. 正文

2.1 缓存机制对比

1. 层级不同：

RecyclerView 比 ListView 多两级缓存，支持多个离 ItemView 缓存，支持开发者自定义缓存处理逻辑，支持所有 RecyclerView 共用同一个 RecyclerViewPool(缓存池)。

具体来说：

ListView(两级缓存)：

ListView				
	是否需要回调createView	是否需要回调bindView	生命周期	备注
mActiveViews	否	否	onLayout函数周期内	用于屏幕内ItemView快速重用
mScrapViews	否	是	与mAdapter一致，当mAdapter被更换时，mScrapViews即被清空	

RecyclerView(四级缓存)：

RecyclerView				
	是否需要回调createView	是否需要回调bindView	生命周期	备注
mAttachedScrap	否	否	onLayout函数周期内	用于屏幕内ItemView快速重用
mCacheViews	否	否	与mAdapter一致，当mAdapter被更换时，mCacheViews即被缓存至mRecyclerPool	默认上限为2，即缓存屏幕外2个ItemView
mViewCacheExtension				不直接使用，需要用户在定制，默认不实现
mRecyclerPool	否	是	与自身生命周期一致，不再被引用时即被释放	默认上限为5，技术上可以实现所有RecyclerViewPool共用同一个

ListView 和 RecyclerView 缓存机制基本一致：

1). mActiveViews 和 mAttachedScrap 功能相似，意义在于快速重用屏幕上可见的列表项 ItemView，而不需要重新 createView 和 bindView；

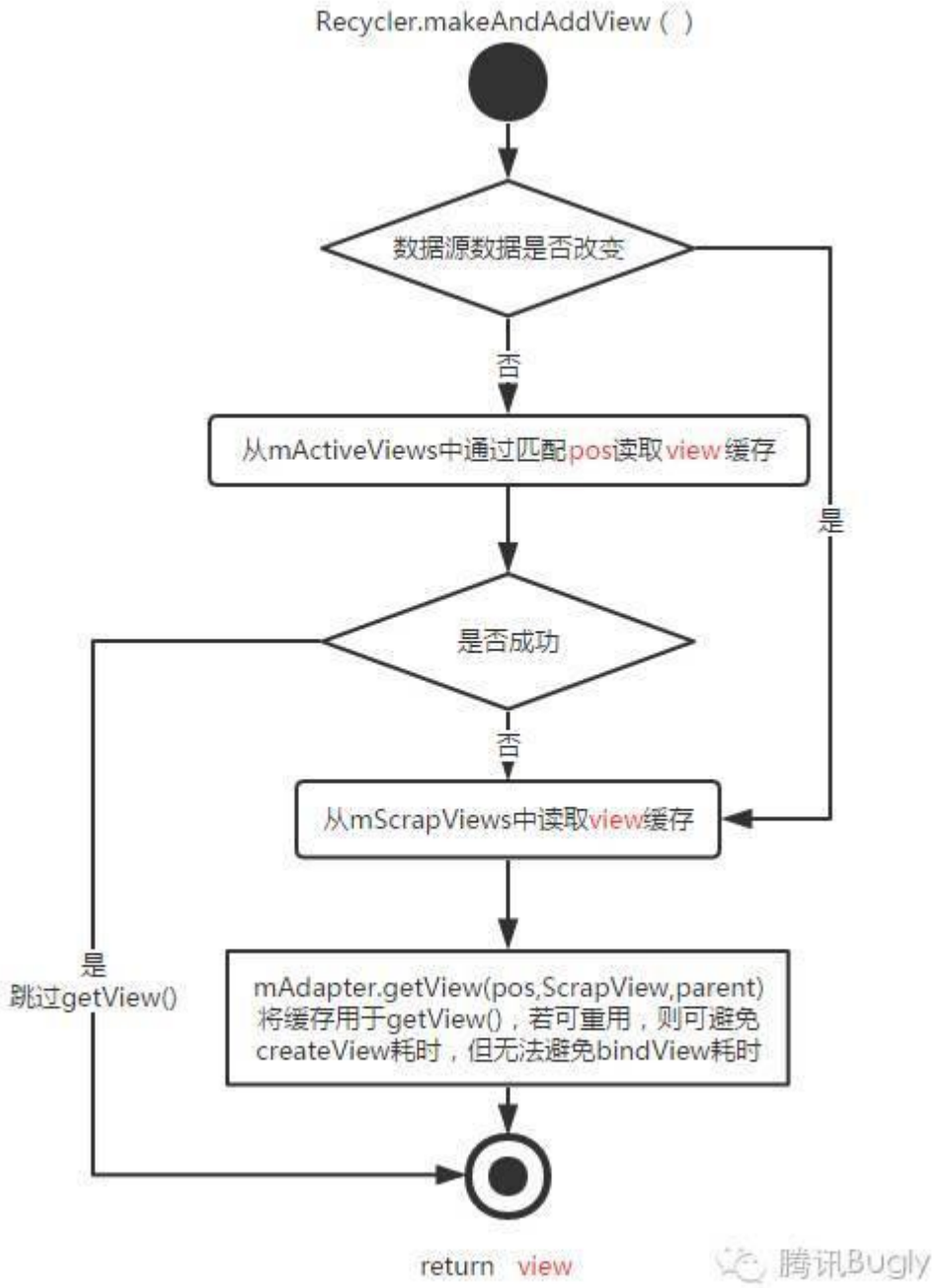
2). mScrapView 和 mCachedViews + mReyclerViewPool 功能相似，意义在于缓存离开屏幕的 ItemView，目的是让即将进入屏幕的 ItemView 重用.

3). RecyclerView 的优势在于 a.mCacheViews 的使用，可以做到屏幕外的列表项 ItemView 进入屏幕内时也无须 bindView 快速重用；b.mRecyclerPool 可以供多个 RecyclerView 共同使用，在特定场景下，如 viewpaper+多个列表页下有优势.客观来说，RecyclerView 在特定场景下对 ListView 的缓存机制做了补强和完善。

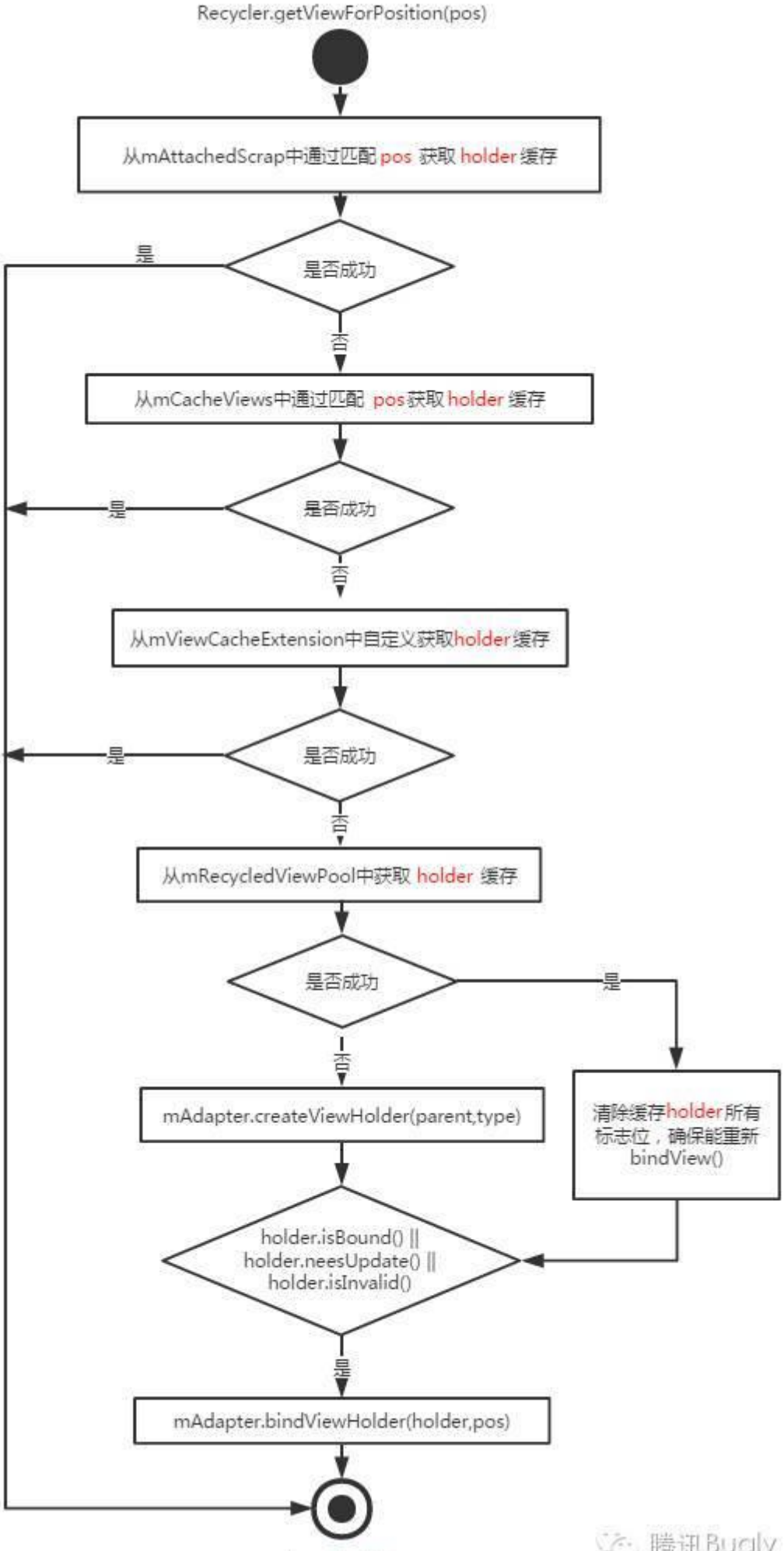
2. 缓存不同：

1). RecyclerView 缓存 RecyclerView.ViewHolder，抽象可理解为：
View + ViewHolder(避免每次 createView 时调用 findViewById) + flag(标识状态)；
2). ListView 缓存 View。

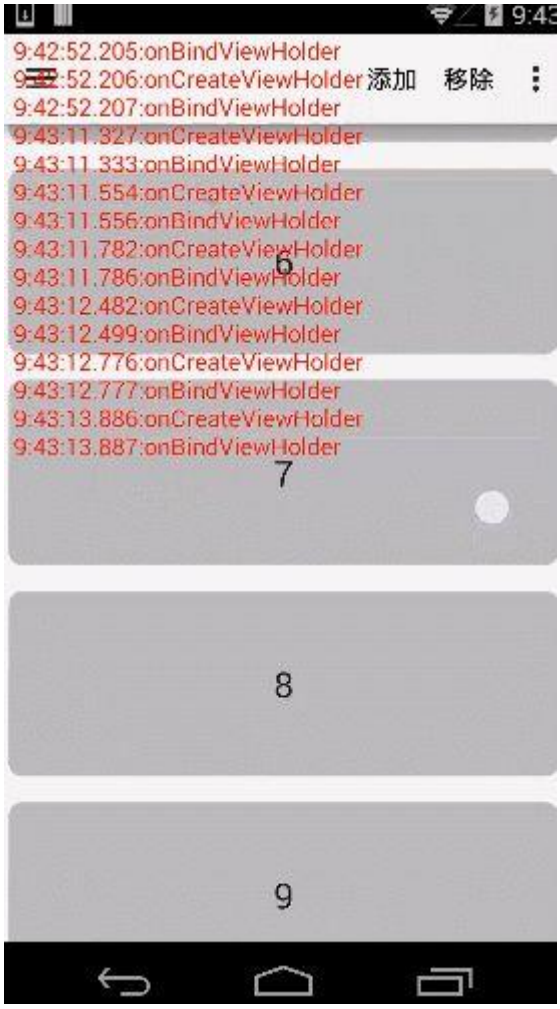
缓存不同，二者在缓存的使用上也略有差别，具体来说：
ListView 获取缓存的流程：



RecyclerView 获取缓存的流程：


Recycler.getViewForPosition(pos)
从mAttachedScrap中通过匹配 pos 获取 holder 缓存
是否成功
是
否
从mCacheViews中通过匹配 pos获取 holder 缓存
是否成功
是
否
从mViewCacheExtension中自定义获取holder缓存
是否成功
是
否
从mRecycledViewPool中获取 holder 缓存
是否成功
是
否
mAdapter.createViewHolder(parent,type)
清除缓存holder所有标志位，确保能重新bindView()
holder.isBound() ||
holder.neesUpdate() ||
holder.isInvalid()
是
mAdapter.bindViewHolder(holder,pos)

1). RecyclerView 中 mCacheViews(屏幕外)获取缓存时，是通过匹配 pos 获取目标位置的缓存，这样做的好处是，当数据源数据不变的情况下，无须重新 bindView：



而同样是离屏缓存，ListView 从 mScrapViews 根据 pos 获取相应的缓存，但是并没有直接使用，而是重新 getView（即必定会重新 bindView），相关代码如下：

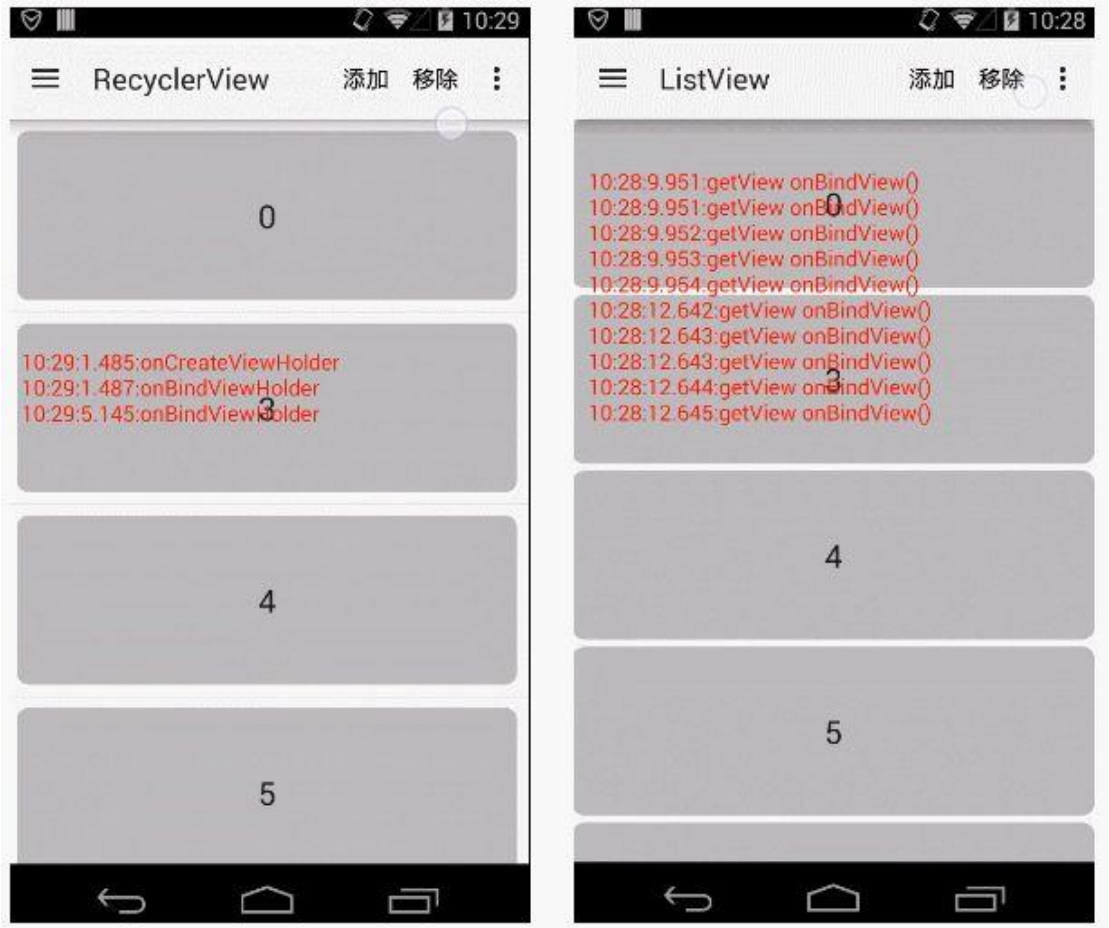
```
// AbsListView 源码：line2345
 //  通过匹配 pos 从 mScrapView 中获取缓存
 final View scrapView = mRecycler.getScrapView(position);
 //  无论是否成功都直接调用 getView,导致必定会调用 createView
 final View child = mAdapter.getView(position, scrapView, this);
 if (scrapView != null) {
        if (child != scrapView) {
              mRecycler.addScrapView(scrapView, position);
        } else {
              ...
        }
 }
```

2). ListView 中通过 pos 获取的是 view，即 pos→view；
RecyclerView 中通过 pos 获取的是 viewholder，即 pos → (view，viewHolder，flag)；

从流程图中可以看出，标志 flag 的作用是判断 view 是否需要重新 bindView，这也是 RecyclerView 实现局部刷新的一个核心.

2.2 局部刷新

由上文可知，RecyclerView 的缓存机制确实更加完善，但还不算质的变化，RecyclerView 更大的亮点在于提供了局部刷新的接口，通过局部刷新，就能避免调用许多无用的 bindView.



(RecyclerView 和 ListView 添加，移除 Item 效果对比)

结合 RecyclerView 的缓存机制，看看局部刷新是如何实现的：

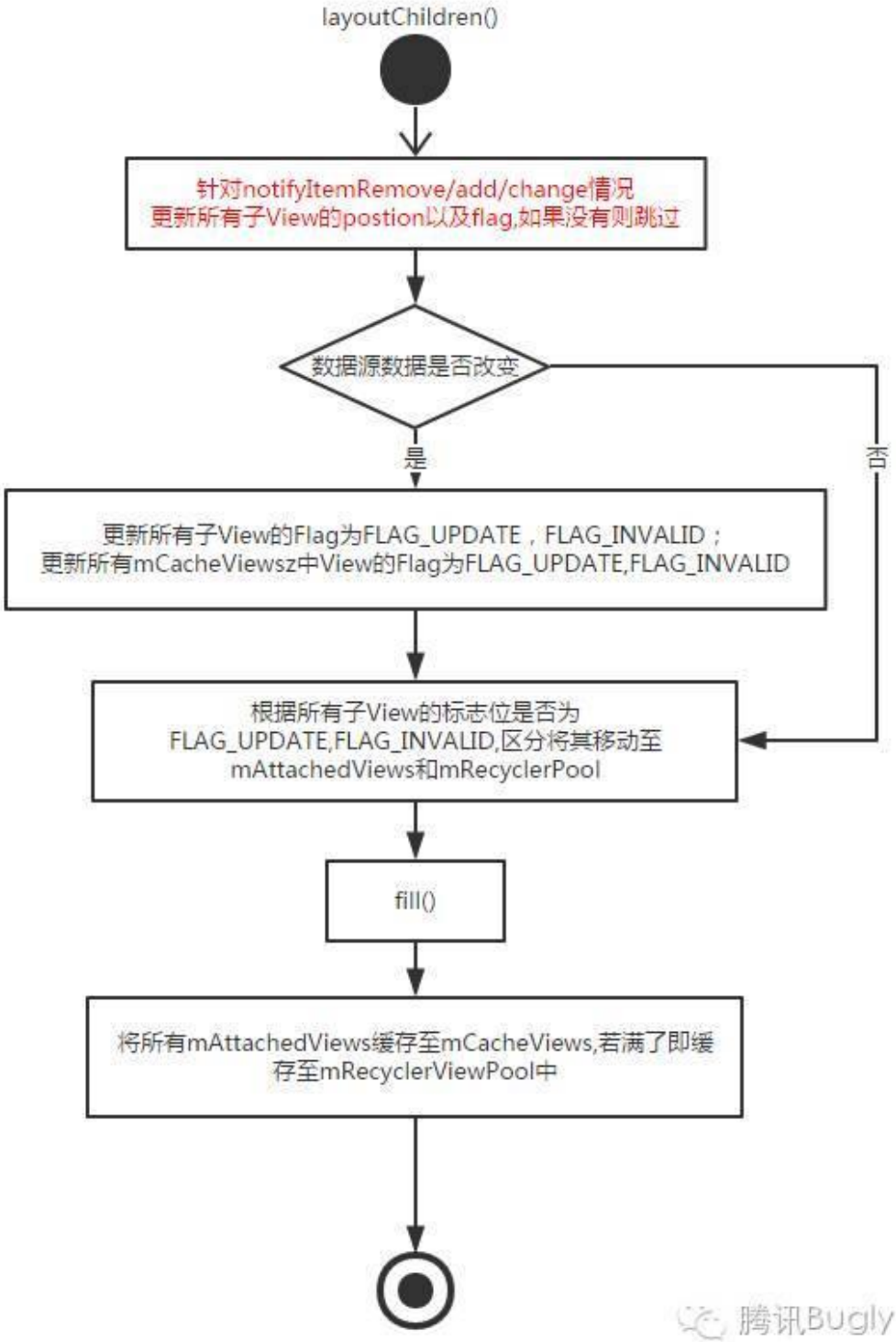
以 RecyclerView 中 notifyItemRemoved(1)为例，最终会调用 requestLayout()，使整个 RecyclerView 重新绘制，过程为：

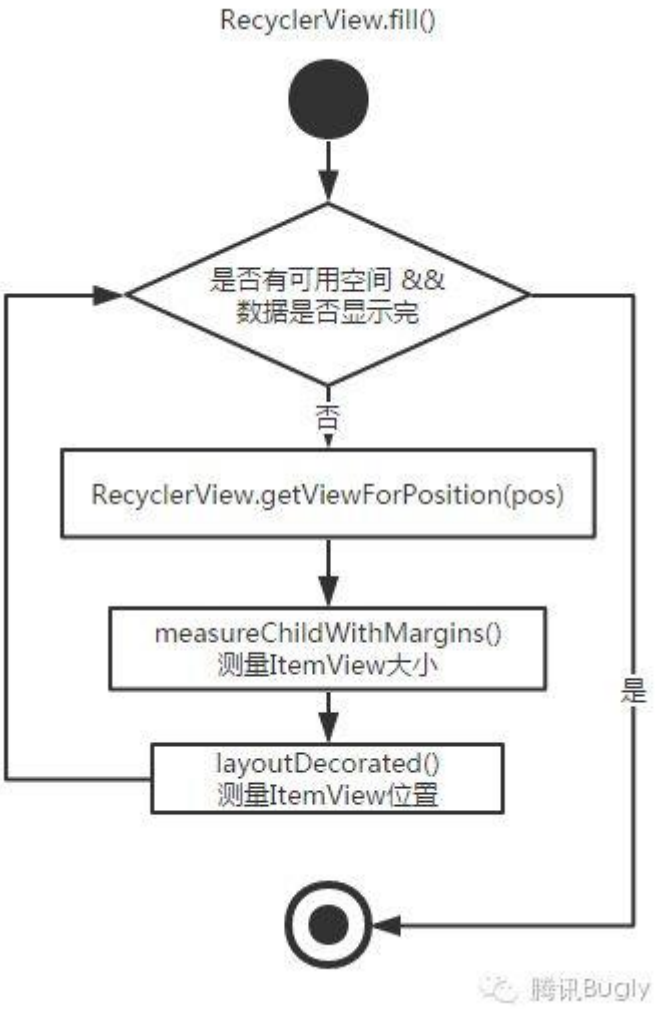
onMeasure()→onLayout()→onDraw()

其中，onLayout()为重点，分为三步：

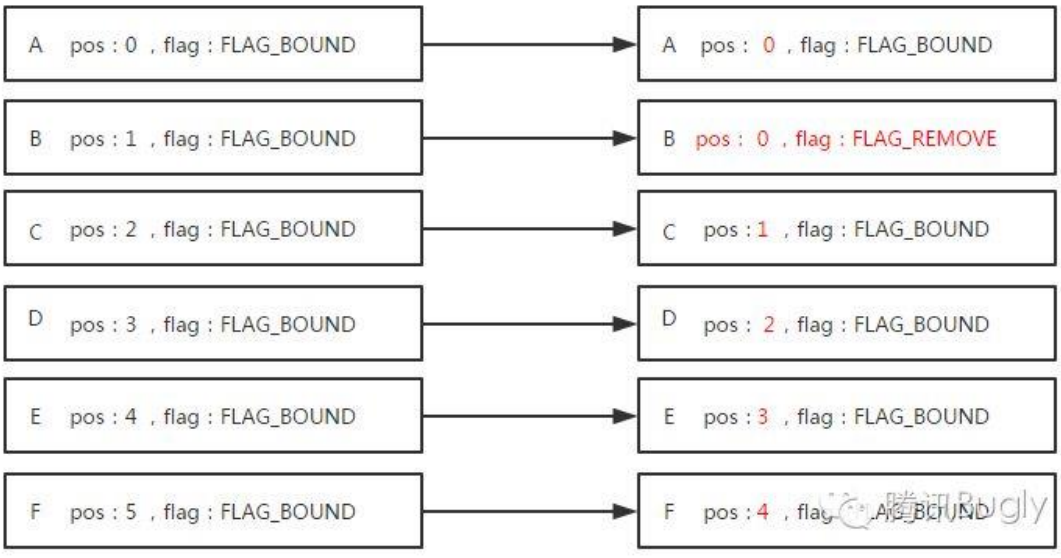
- dispathLayoutStep1()：记录 RecyclerView 刷新前列表项 ItemView 的各种信息，如 Top,Left,Bottom,Right，用于动画的相关计算；
- dispathLayoutStep2()：真正测量布局大小，位置，核心函数为 layoutChildren()；
- dispathLayoutStep3()：计算布局前后各个 ItemView 的状态，如 Remove，Add，Move，Update 等，如有必要执行相应的动画.

其中，layoutChildren()流程图：


layoutChildren()
针对notifyItemRemove/add/change情况
更新所有子View的postion以及flag,如果没有则跳过
数据源数据是否改变
是
否
更新所有子View的Flag为FLAG_UPDATE，FLAG_INVALID；
更新所有mCacheViewsz中View的Flag为FLAG_UPDATE,FLAG_INVALID
根据所有子View的标志位是否为
FLAG_UPDATE,FLAG_INVALID,区分将其移动至
mAttachedViews和mRecyclerPool
fill()
将所有mAttachedViews缓存至mCacheViews,若满了即缓
存至mRecyclerViewPool中



当调用 notifyItemRemoved 时，会对屏幕内 ItemView 做预处理，修改 ItemView 相应的 pos 以及 flag(流程图中红色部分):



当调用 fill()中 RecyclerView.getViewForPosition(pos)时，RecyclerView 通过对 pos 和 flag 的预处理，使得 bindview 只调用一次.

需要指出，ListView 和 RecyclerView 最大的区别在于数据源改变时的缓存的处理逻辑，ListView 是"一锅端"，将所有的 mActiveViews 都移入了二级缓存 mScrapViews，而 RecyclerView 则是更加灵活地对每个 View 修改标志位，区分是否重新 bindView。

三.结论

在一些场景下，如界面初始化，滑动等，ListView 和 RecyclerView 都能很好地工作，两者并没有很大的差异:

文章的开头便抛出了这样一个问题，微信 Android 客户端卡券模块，大部分 UI 都是以列表页的形式展示，实现方式为 ListView，是否有必要将其替换成 RecyclerView 呢?

答案是否定的，从性能上看，RecyclerView 并没有带来显著的提升，不需要频繁更新，暂不支持用动画，意味着 RecyclerView 优势也不太明显，没有太大的吸引力，ListView 已经能很好地满足业务需求。

数据源频繁更新的场景，如弹幕：Recyclerview 实现的弹幕 （旧）等 RecyclerView 的优势

会非常明显。
https://www.jianshu.com/p/2232a63442d6

进一步来讲，结论是：
列表页展示界面，需要支持动画，或者频繁更新，局部刷新，建议使用 RecyclerView，更加强大完善，易扩展；其它情况(如微信卡包列表页)两者都 OK，但 ListView 在使用上会更加方便，快捷。

Ps：仅从一个角度做了对比，盲人摸象，有误跪求指正。

四.参考资料

1. ListView
a. Android-23 源码
b. Android ListView 工作原理解析，带你从源码的角度彻底理解：Android ListView 工作原理完全解析，带你从源码的角度彻底理解
https://blog.csdn.net/guolin_blog/article/details/44996879
c. Android 自己动手写 ListView 学习其原理：
https://blog.csdn.net/androiddevelop/article/details/8734255

2. RecyclerView
a. RecyclerView-v7-23.4.0 源码
b. RecyclerView 剖析：RecyclerView 剖析
https://blog.csdn.net/qq_23012315/article/details/50807224
c. RecyclerView 剖析
https://blog.csdn.net/qq_23012315/article/details/51096696