链滴

浅谈 RecyclerView 的性能优化

作者：xuexiangjys

原文链接：https://ld246.com/article/1668224927552

来源网站：链滴

RecyclerView的性能优化

在我们谈RecyclerView的性能优化之前，先让我们回顾一下RecyclerView的缓存机制。

RecyclerView缓存机制

众所周知，RecyclerView拥有四级缓存，它们分别是：

- Scrap缓存：包括mAttachedScrap和mChangedScrap，又称屏内缓存，不参与滑动时的回收复，只是用作临时保存的变量。
    - mAttachedScrap：只保存重新布局时从RecyclerView分离的item的无效、未移除、未更新的holder。
    - mChangedScrap：只会负责保存重新布局时发生变化的item的无效、未移除的holder。
- CacheView缓存：mCachedViews又称离屏缓存，用于保存最新被移除(remove)的ViewHolder，经和RecyclerView分离的视图，这一级的缓存是有容量限制的，默认最大数量为2。
- ViewCacheExtension：mViewCacheExtension又称拓展缓存，为开发者预留的缓存池，开发者以自己拓展回收池，一般不会用到。
- RecycledViewPool：终极的回收缓存池，真正存放着被标识废弃(其他池都不愿意回收)的ViewHolder的缓存池。这里的ViewHolder是已经被抹除数据的，没有任何绑定的痕迹，需要重新绑定数据。

RecyclerView的回收原理

（1）如果是RecyclerView不滚动情况下缓存(比如删除item)、重新布局时。

- 把屏幕上的ViewHolder与屏幕分离下来，存放到Scrap中，即发生改变的ViewHolder缓存到mChagedScrap中，不发生改变的ViewHolder存放到mAttachedScrap中。
- 剩下ViewHolder会按照 mCachedViews > RecycledViewPool的优先级缓存到mCachedViews者RecycledViewPool中。

（2）如果是RecyclerView滚动情况下缓存(比如滑动列表)，在滑动时填充布局。

- 先移除滑出屏幕的item，第一级缓存mCachedViews优先缓存这些ViewHolder。
- 由于mCachedViews最大容量为2，当mCachedViews满了以后，会利用先进先出原则，把旧的ViewHolder存放到RecycledViewPool中后移除掉，腾出空间，再将新的ViewHolder添加到mCachedViews中。
- 最后剩下的ViewHolder都会缓存到终极回收池RecycledViewPool中，它是根据itemType来缓存同类型的ArrayList<ViewHolder>，最大容量为5。

RecyclerView的复用原理

当RecyclerView要拿一个复用的ViewHolder时：

- 如果是预加载，则会先去mChangedScrap中精准查找(分别根据position和id)对应的ViewHolder。
- 如果没有就再去mAttachedScrap和mCachedViews中精确查找(先position后id)是不是原来的ViewHolder。
- 如果还没有，则最终去mRecyclerPool找，如果itemType类型匹配对应的ViewHolder，那么返回

例，让它 重新绑定数据。

- 如果mRecyclerPool也没有返回ViewHolder才会调用 createViewHolder()重新去创建一个。

这里有几点需要注意：

- 在mChangedScrap、mAttachedScrap、mCachedViews中拿到的ViewHolder都是精准匹配。
- mAttachedScrap和mCachedViews没有发生变化，是直接使用的。
- mChangedScrap由于发生了变化，mRecyclerPool由于数据已被抹去，所以都需要调用 onBindViewHolder()重新绑定数据才能使用。

缓存机制总结

- RecyclerView最多可以缓存 N（屏幕最多可显示的item数【Scrap缓存】） + 2 (屏幕外的缓存【CacheView缓存】) + 5*M (M代表M个ViewType，缓存池的缓存【RecycledViewPool】)。
- RecyclerView实际只有两层缓存可供使用和优化。因为Scrap缓存池不参与滚动的回收复用，所以CacheView缓存池被称为一级缓存，又因为ViewCacheExtension缓存池是给开发者定义的缓存池，一不用到，所以RecycledViewPool缓存池被称为二级缓存。

如果想深入了解RecyclerView缓存机制的同学，可以参考《RecyclerView的回收复用缓存机制详解》这篇文章。

性能优化方案

根据上面我们对缓存机制的了解，我们可以简单得到以下几个大方向：

- 1.提高ViewHolder的复用，减少ViewHolder的创建和数据绑定工作。【最重要】
- 2.优化 onBindViewHolder方法，减少ViewHolder绑定的时间。由于ViewHolder可能会进行多绑定，所以在 onBindViewHolder()尽量只做简单的工作。
- 3.优化 onCreateViewHolder方法，减少ViewHolder创建的时间。

提高ViewHolder的复用

1.多使用Scrap进行局部更新。

- (1) 使用 notifyItemChange、notifyItemInserted、notifyItemMoved和 notifyItemRemoved方法替代 notifyDataSetChanged方法。
- (2) 使用 notifyItemChanged(int position, @Nullable Object payload)方法，传入需要刷新的容进行局部增量刷新。这个方法一般很少有人知道，具体做法如下：
    - 首先在notify的时候，在payload中传入需要刷新的数据，一般使用Bundle作为数据的载体。
    - 然后重写 RecyclerView.Adapter的 onBindViewHolder(@NonNull RecyclerViewHolder holder, int position, @NonNull List<Object> payloads)方法

```
@Override
public void onBindViewHolder(@NonNull RecyclerViewHolder holder, int position, @NonNull List<Object> payloads) {
    if (CollectionUtils.isEmpty(payloads)) {
        Logger.e("正在进行全量刷新:" + position);
```

```
            onBindViewHolder(holder, position);
            return;
        }
        // payloads为非空的情况，进行局部刷新
        //取出我们在getChangePayload（）方法返回的bundle
        Bundle payload = WidgetUtils.getChangePayload(payloads);
        if (payload == null) {
            return;
        }
        Logger.e("正在进行增量刷新:" + position);
        for (String key : payload.keySet()) {
            if (KEY_SELECT_STATUS.equals(key)) {
                holder.checked(R.id.scb_select, payload.getBoolean(key));
            }
        }
    }
```

详细使用方法可参考XUI中的RecyclerView局部增量刷新 中的代码。

● (3) 使用 DiffUtil、SortedList进行局部增量刷新，提高刷新效率。和上面讲的传入 payload原理样，这两个是Android默认提供给我们使用的两个封装类。这里我以 DiffUtil举例说明该如何使用。

- 首先需要实现 DiffUtil.Callback的5个抽象方法，具体可参考DiffUtilCallback.java
- 然后调用 DiffUtil.calculateDiff方法返回比较的结果 DiffUtil.DiffResult。
- 最后调用 DiffUtil.DiffResult的 dispatchUpdatesTo方法，传入RecyclerView.Adapter进行据刷新。

详细使用方法可参考XUI中的DiffUtil局部刷新 和 XUI中的SortedList自动数据排序刷新 中的代码。

2.合理设置RecyclerViewPool的大小。如果一屏的item较多，那么RecyclerViewPool的大小就不能使用默认的5，可适度增大Pool池的大小。如果存在RecyclerView中嵌套RecyclerView的情况，可以虑复用RecyclerViewPool缓存池，减少开销。

3.为RecyclerView设置 setHasStableIds为true，并同时重写RecyclerView.Adapter的 getItemId方来给每个Item一个唯一的ID，提高缓存的复用率。

4.视情况使用 setItemViewCacheSize(size)来加大CacheView缓存数目，用空间换取时间提高流畅。对于可能来回滑动的RecyclerView，把CacheViews的缓存数量设置大一些，可以省去ViewHolde绑定的时间，加快布局显示。

5.当两个数据源大部分相似时，使用 swapAdapter代替 setAdapter。这是因为 setAdapter会直接空RecyclerView上的所有缓存，但是 swapAdapter会将RecyclerView上的ViewHolder保存到pool，这样当数据源相似时，就可以提高缓存的复用率。

优化onBindViewHolder方法

1.在onBindViewHolder方法中，去除冗余的setOnItemClick等事件。因为直接在onBindViewHolde方法中创建匿名内部类的方式来实现setOnItemClick，会导致在RecyclerView快速滑动时创建很多象。应当把事件的绑定在ViewHolder创建的时候和对应的rootView进行绑定。

2.数据处理与视图绑定分离，去除onBindViewHolder方法里面的耗时操作，只做纯粹的数据绑定操。当程序走到onBindViewHolder方法时，数据应当是准备完备的，禁止在onBindViewHolder方法

面进行数据获取的操作。

3.有大量图片时，滚动时停止加载图片，停止后再去加载图片。

4.对于固定尺寸的item，可以使用 setHasFixedSize避免 requestLayout。

优化onCreateViewHolder方法

1.降低item的布局层级，可以减少界面创建的渲染时间。

2.Prefetch预取。如果你使用的是嵌套的RecyclerView，或者你自己写LayoutManager，则需要自实现Prefetch，重写 collectAdjacentPrefetchPositions方法。

其他

以上都是针对RecyclerView的缓存机制展开的优化方案，其实还有几种方案可供参考。

1.取消不需要的item动画。具体的做法是：

```
((SimpleItemAnimator) recyclerView.getItemAnimator()).setSupportsChangeAnimations(false);
```

2.使用 getExtraLayoutSpace为LayoutManager设置更多的预留空间。当RecyclerView的元素比较，一屏只能显示一个元素的时候，第一次滑动到第二个元素会卡顿，这个时候就需要预留的额外空间让RecyclerView预加载可重用的缓存。

最后

以上就是RecyclerView性能优化的全部内容，俗话说：百闻不如一见，百见不如一干，大家还是赶紧手尝试着开始进行优化吧!