Recyclerview 学习系类之 ItemDecoration(一)

作者：angels
原文链接：https://ld246.com/article/1496286816822
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

更多分享请看：http://www.cherylgood.cn

<h4 id="Google官方解释">Google 官方解释</h4>

An ItemDecoration allows the application to add a special drawing and layout offset to specific item views from the adapter's data set. This can be useful for drawing dividers between items, highlights, visual grouping boundaries and more.

All ItemDecorations are drawn in the order they were added, before the item views (in onDraw() and after the items (in onDrawOver(Canvas, RecyclerView, RecyclerView.State).

<h5 id="个人理解-">个人理解：</h5>
大致意思是：

ItemDecoration 允许应用程序从适配器的数据集中为制定的 view 添加制定的图形和布局偏移。该特性一般被用于在两个 item 之间绘制分割线，高亮度以及视觉分组等等。

所有的 ItemDecorations 都按照它们被添加的顺序在 item 被绘制之前（在 onDraw 方法中）在 Items 被绘制之后（在 onDrawOver(Canvas,RecyclerView,RecyclerView.State)）进行绘制。
可以看到，ItemDecoration 是相当强大和灵活的。

<h4 id="method学习-">method 学习：</h4>
<blockquote>
getItemOffsets(Rect outRect, int itemPosition, RecyclerView parent)
This method was deprecated in API level 22.0.0. Use getItemOffsets(Rect, View, RecyclerView, State)
</blockquote>

该方法在 API 22.0.0 之后已被废弃，我们可以看代替的方法

<blockquote>
<a href="https://ld246.com/forward?goto=https%3A%2F%2Fdeveloper.android.com%2Freference%2Fandroid%2Fsupport%2Fv7%2Fwidget%2FRecyclerView.ItemDecoration.html%23

etItemOffsets%28android.graphics.Rect%2C%2520android.view.View%2C%2520android.support.v7.widget.RecyclerView%2C%2520android.support.v7.widget.RecyclerView.State%29" targe="_blank" rel="nofollow ugc">getItemOffsets(Rect outRect, View view, RecyclerView parent RecyclerView.State state)
Retrieve any offsets for the given item.
</blockquote>

我们可以通过该方法中的 outRect 来设置 item 的 padding 值。比如你要在 item 底部添加一条割线，此时为了不影响 item 原来的布局参数，我们一般会返回一个地步 padding 为某个 pd 的 outRct，在 recyclerview 绘制 item 的时候会讲该布局数据加入，我们原来的 item 就会多出一个底部 pading，是不是解耦的很完美呢？

<blockquote>
onDraw(Canvas c, RecyclerView parent)<b>
_This method was deprecated in API level 22.0.0. Override onDraw(Canvas, RecyclerView, RecyclerViewState) _
</blockquote>

该方法也已经过期了，看下面的

<blockquote>
oDraw(Canvas c, RecyclerView parent, <a href="https://ld246.com/forward?oto=https%3A%2F%2Fdeveloper.android.com%2Freference%2Fandroid%2Fsupport%2Fv7%2widget%2FRecyclerView.State.html" target="_blank" rel="nofollow ugc">RecyclerView.State<a> state)
Draw any appropriate decorations into the Canvas supplied to the RecyclerView.

</blockquote>

该方法会在绘制 item 之前调用，也就是说他的层级是在 item 之下的，通过该方法，我们可以绘制 item 之前绘制我们需要的内容。

<blockquote>
onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state)
Draw any appropriate decorations into the Canvas supplied to the RecyclerView.
</blockquote>

该方法已过期，看下面的

<blockquote>
onDrawOver(Canvas c, RecyclerView parent)
_This method was deprecated in API level 22.0.0. Override onDrawOver(Canvas, RecyclerView, ecyclerView.State) _
</blockquote>

该方法于 onDrawOver 类似，在绘制 item 之后会调用该方法。

此时，也许你会疑问，他真的是这样执行的么？为了一探究竟，我们来看下源码吧。
<pre><code class="highlight-chroma"> @Override
public void draw(anvas c) {
    super.draw(c);
    final int count = mItemDecorations.size();
    for (int i = 0; i &t; count; i++) {
        mItemDecora

ions.get(i).onDrawOver(c, this, mState);
    }...}

</code></pre>

从 recyclerview 的源码中我们可以看到，在 draw 方法中后会遍历 recyclerview 里面的 itemDecoration 然后调用 itemdecoration 的 onDrawOver 方法；而 recyclerview 调用了 super.draw(c) 后会先，父类会先调用 recyclerview 的 onDraw 方法；
<pre><code class="highlight-chroma"> @Override
public void onDraw(Canvas c) {
  super.onDraw(c);
  final int count = mItemDecorations.size();
  for (int i = 0; i < count; i++) {
    mItemDecorations.get(i).onDraw(c, this, mState);
  } }

</code></pre>

在 recyclerview 的 onDraw 里又会调用 itemDecoration 的 onDraw 方法,当 recyclerview 的 nDraw 方法执行完之后，recyclerview 的 draw 方法中 super.draw(c)；后面的代码才会继续执行，recyclerview 是在绘制了自己之后才会去绘制 item。

结论：itemDecoration 的 onDraw 方法在 item 绘制之前调用，itemDecoration 的 onDrawOer 方法在绘制 item 之后调用。

接下来我们在看下 getItemOffsets 这个方法。他真的把我们的 outRect 加到 item 的布局参数面了么？预知真相，看源码。
<pre><code class="highlight-chroma"> Rect getItemDecorInsetsForChild(View child) {
    final LayoutParams lp = (LayoutParams) child.getLayoutParams();
    if (!lp.mInsetsDirty) {
      return lp.mDecorInsets;
    }

    if (mState.isPreLayout() && (lp.isItemChanged() || lp.isViewInvalid())) {
      // changed/invalid items should not be updated until they are rebound.
      return lp.mDecorInsets;
    }
    final Rect insets

```
= lp.mDecorInsets;
    insets.set(0, 0, 0, 0);
    final int decorCount = mItemDecorations.size();
    for (int i = 0; i < decorCount; i++) {
        mTempRect.set(0, 0, 0, 0);
        mItemDecorations.get(i).getItemOffsets(mTempRect, child, this, mState);
        insets.left += mTempRect.left;
        insets.top += mTempRect.top;
        insets.right = mTempRect.right;
        insets.bottom += mTempRect.bottom;
    }
    lp.mInsetsDirty = false;
    return insets;
}
```


首先我们可以看到，getItemOffsets 这个方法在 recyclerview 的 getItemDecorInsetsForChild 中被调用，该方法会把所有的 itemDecortion 中的 rect 累加后返回；我们再看下 getItemDecorInsesForChild 在哪被调用的。

```
  public void measureChild(View child, int widthUsed, int heightUsed) {
      final LayoutParams lp = (LayoutParams) child.getLayoutParams();

      final Rect insets = mRecyclerView.getItemDecorInsetsForChild(child);

```


在 measureChild 方法中被调用，也就是 recyclerview 在测量 childView 的时候

```
  public void measureChildWithMargins(View child, int widthUsed, int heightUsed) {
      final LayoutParams lp = (LayoutParams) child.getLayoutParams();

      final Rect insets = mRecyclerView.getItemDecorInsetsForChild(child);

```



```
<p>使用 margins 测量 childView 时会用到</p>
</li>
<li>
<p>结论，在 getItemOffsets 方法中 outRect 会影响到 recyclerview 中 childView 的布局。</p>
</li>
</ul>
<p>使用 ItemDecoration 实现分割线的都调用过 addItemDecoration 方法。发现，只要调用一次
ddItemDecoration 将自定义的分割线 ItemDecoration 添加进去就可以实现分割线效果了，如果我
添加多次会如何呢？</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">public void addItemDecoration(ItemDecoration decor, int index) {
</span></span><span class="highlight-line"><span class="highlight-cl">    if (mLayout !=
ull) {
</span></span><span class="highlight-line"><span class="highlight-cl">        mLayout.asse
tNotInLayoutOrScroll("Cannot add item decoration during a scroll  or"
</span></span><span class="highlight-line"><span class="highlight-cl">                + " layo
t");
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    if (mItemDecor
tions.isEmpty()) {
</span></span><span class="highlight-line"><span class="highlight-cl">        setWillNotDr
w(false);
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    if (index &lt; 0) {
</span></span><span class="highlight-line"><span class="highlight-cl">        mItemDecora
ions.add(decor);
</span></span><span class="highlight-line"><span class="highlight-cl">    } else {
</span></span><span class="highlight-line"><span class="highlight-cl">        mItemDecora
ions.add(index, decor);
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    markItemDecor
nsetsDirty();
</span></span><span class="highlight-line"><span class="highlight-cl">    requestLayout();
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span></code></pre>
<ul>
<li>
<p>从 RecyclerView.addItemDecoration 方法源码可以看到，内部使用了一个 ArrayList 类型的 mI
emDecorations 存储我们添加的所有 ItemDecoration。markItemDecorInsetsDirty 方法有什么用
？我们看下源码</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">void markItemDecorInsetsDirty() {
</span></span><span class="highlight-line"><span class="highlight-cl">  final int childCo
nt = mChildHelper.getUnfilteredChildCount();
</span></span><span class="highlight-line"><span class="highlight-cl">  for (int i = 0; i &lt
 childCount; i++) {
</span></span><span class="highlight-line"><span class="highlight-cl">      final View chil
 = mChildHelper.getUnfilteredChildAt(i);
</span></span><span class="highlight-line"><span class="highlight-cl">      ((LayoutParam
) child.getLayoutParams()).mInsetsDirty = true;
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">  mRecycler.markI
emDecorInsetsDirty();
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span></code></pre>
</li>
<li>
<p>里面有一个 mInsetsDirty 被重置为 true，最终调用 mRecycler.markItemDecorInsetsDirty();
们继续看 mRecycler.markItemDecorInsetsDirty();方法源码：</p>
<p>void markItemDecorInsetsDirty() {</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">      final int cachedCount = mCachedViews.size();
</span></span><span class="highlight-line"><span class="highlight-cl">      for (int i = 0; i
&amp;lt; cachedCount; i++) {
</span></span><span class="highlight-line"><span class="highlight-cl">          final ViewHo
der holder = mCachedViews.get(i);
</span></span><span class="highlight-line"><span class="highlight-cl">          LayoutPara
s layoutParams = (LayoutParams) holder.itemView.getLayoutParams();
</span></span><span class="highlight-line"><span class="highlight-cl">          if (layoutPar
ms != null) {
</span></span><span class="highlight-line"><span class="highlight-cl">              layoutPar
ms.mInsetsDirty = true;
</span></span><span class="highlight-line"><span class="highlight-cl">          }
</span></span><span class="highlight-line"><span class="highlight-cl">      }
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span></code></pre>
</li>
<li>
<p>里面也是将 layoutParams 的 mInsetsDirty 重置为 true，这个 mInsetsDirty 有什么用呢？ 我
继续看源码：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">Rect getItemDecorInsetsForChild(View child) {
</span></span><span class="highlight-line"><span class="highlight-cl">  final LayoutPara
s lp = (LayoutParams) child.getLayoutParams();
</span></span><span class="highlight-line"><span class="highlight-cl">  if (!lp.mInsetsDir
y) {
</span></span><span class="highlight-line"><span class="highlight-cl">      return lp.mDe
rInsets;
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">  if (mState.isPreL
yout() &amp;amp;&amp;amp; (lp.isItemChanged() || lp.isViewInvalid())) {
</span></span><span class="highlight-line"><span class="highlight-cl">      // changed/in
alid items should not be updated until they are rebound.
</span></span><span class="highlight-line"><span class="highlight-cl">      return lp.mDe
orInsets;
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">  final Rect insets
 lp.mDecorInsets;
</span></span><span class="highlight-line"><span class="highlight-cl">  insets.set(0, 0, 0,
);
</span></span><span class="highlight-line"><span class="highlight-cl">  final int decorCo
nt = mItemDecorations.size();
</span></span><span class="highlight-line"><span class="highlight-cl">  for (int i = 0; i &amp;lt
 decorCount; i++) {
```

      mTempRect.se
(0, 0, 0, 0);
      mItemDecorat
ons.get(i).getItemOffsets(mTempRect, child, this, mState);
      insets.left +=
mTempRect.left;
      insets.top +=
mTempRect.top;
      insets.right +=
mTempRect.right;
      insets.bottom
+= mTempRect.bottom;
  }
  lp.mInsetsDirty =
false;
  return insets;
}

</code></pre>

看到这段代码感觉应该是它了，可以看到，

判断 childView 的 layoutParams 的 mInsetsDirty 是不是 false 是 false 直接返回 mDecorInset
。
判断 itemDecoration 是否已改变或者已不可用，mState.isPreLayout 是 recyclerview 用来处
动画的。
如果前面的都不是，就会从新调用 itemDecoration 的 getItemOffsets 方法，重新计算 layout
离值之后返回。

出于性能的考虑，如果之前为 ChildView 生成过 DecorInsets，那么会缓存在 ChildView 的 Lay
utParam 中(mDecorInsets), 同时为了保证 mDecorInsets 的时效性，还同步维护了一个 mInsetsDir
y 标记在 LayoutParam 中

在获取 ChidlView 的 DecorInsets 时，如果其 mInsetsDirty 为 false，那么代表缓存没有过期
直接返回缓存的 mDecorInsets。

如果 mInsetsDirty 为 true，表示缓存已过期，需要根据 ItemDecoration 集合重新生成

添加或者删除 ItemDecoration 的时候，会将所有 ChildView 包括 Recycler 中的 mInsetsDirty
设置为 true 来使 DecorInsets 缓存失效

<blockquote>
总结：其实 getItemDecorInsetsForChild 方法我们之前在本章前面有分析到。他就是在测量 chi
dView 的时候会调用，所以如果我们的 itemDecortion 中途需要更新，我们需要调用 markItemDec
rInsetsDirty 方法，然后调用 requestLayout 请求重新绘制，这样在重新绘制 childView 的时候，就
重新计算 ItemDecortion 中返回的 layout 偏离值。达到我们想要的效果。
</blockquote>

更多分享请看：http://www.cherylgood.cn
