**RecyclerView技术栈**

字数2676 阅读5692 评论21 喜欢88

**概述**

随着2014年Google IO的召开，[Android L Preview](http://developer.android.com/preview/index.html)版随之发布，对于开发着来说，带来了性能上的改善，而对于消费者来说，得到了体验上的提升。我想，无论是开发者还是使用者，一定都非常喜欢这次的版本跟新。

同时，这次也带来了两个全新的View控件：**RecyclerView**和**CardView**。这篇文章将重点介绍**RecyclerView**，它有许多内部类和接口。接下来，我将介绍它们的功能，已经如何使用。

当然，在这之前，我要声明的是：**RecyclerView 是Support Library的一部分**。所以只需要在app/build.gradle中添加以下依赖，便能立即使用：

dependencies {

compile 'com.android.support:recyclerview-v7:23.2.0'

}

然后点击“Sync Project with Gradle files”，让IDE去下载适当的资源文件。

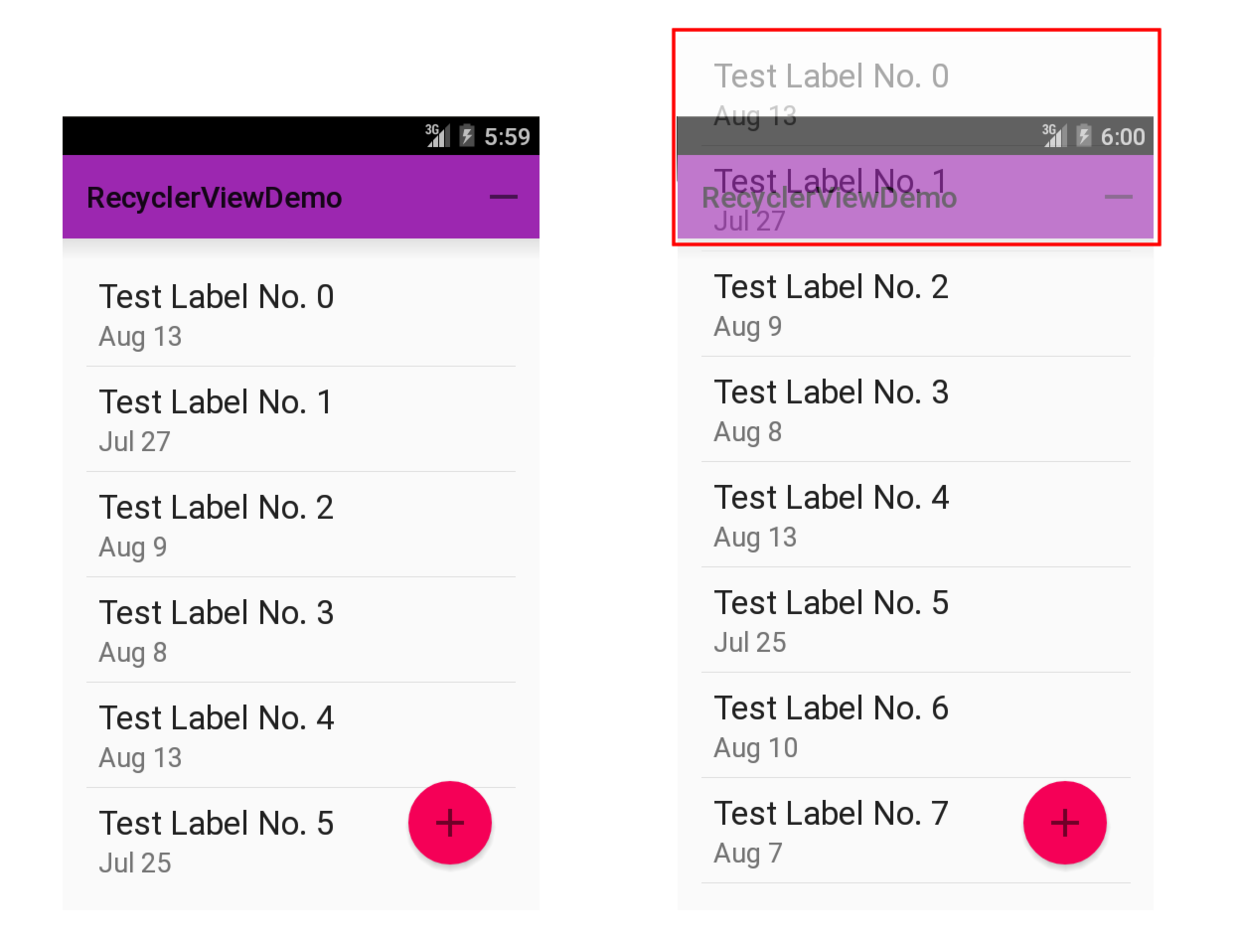
**为什么命名为RecyclerView？**

先让我们来看看Google在**L Preview**中是如何定义**RecyclerView**的：

A flexible view for providing a limited window into a large data set.  
(能够在有限的窗口中展示大数据集合的灵活视图。)

所以我们能够理解为，RecyclerView一个恰当的使用场景是：由于尺寸限制，用户的设备不能一次性展现所有条目，用户需要上下滚动以查看更多条目。滚出可见区域的条目将被回收，并在下一个条目可见的时候被复用。

我们可以从下图中得到更直观的解释：



左边的图是数据初始化后的示例，当向上滚动视图的时候，当条目不可见之后将被回收。右图中红色区域内的两条不可见条目，将被放到缓存队列中以便新的条目可见时进行复用。

对于减少内存开销和CPU的计算，缓存条目是一个非常有用的方法，因为这意味着我们不必每次都创建新的条目，从而减小内存开销和CPU的计算，而且还能够有效降低屏幕的卡顿，保证滑动的顺滑和16ms准则。

看到这里，你可能不禁会问：并没有什么新东西啊，这和**ListView**有什么区别呀？我们已经使用**ListView**很长一段时间了呀，它一样可以做到呀。不过，视图回收本身并不是什么新鲜事。但是回想之前我们写的**ListView**，无论从它的的性能表现着手，还是语法的书写，甚至数据的绑定都未免略显臃肿。那么现在，我们将再也不会出现上述症状，因为Google提供了一个更好，更灵活的控件——**RecyclerView**。

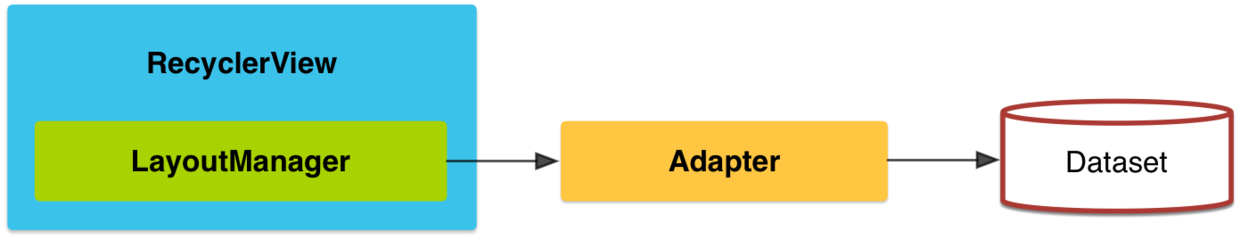
OK，从现在开始，让我们一步一步，开始了解它。

**结构**

如果你想使用**RecyclerView**，需要做以下操作：

* RecyclerView.Adapter - 处理数据集合并负责绑定视图
* ViewHolder - 持有所有的用于绑定数据或者需要操作的View
* LayoutManager - 负责摆放视图等相关操作
* ItemDecoration - 负责绘制Item附近的分割线
* ItemAnimator - 为Item的一般操作添加动画效果，如，增删条目等

我们可以从下图更直观的了解到**RecyclerView**的基本结构：



由此可见，想要在**ListView**中实现条目的增删动画是一件非常困难的事情，但是**RecyclerView**为我们提供了很好的便利。而且**RecyclerView**增强了[ViewHolder设计模式](https://guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView#improving-performance-with-the-viewholder-pattern)，这在当前所使用的ListView中是不曾有的。

**与传统ListView比较**

**RecyclerView**与老前辈**ListView**的不同点，主要在于以下几个特性：

* **Adapter中的ViewHolder模式** - 对于ListView来说，通过创建ViewHolder来提升性能并不是必须的。因为ListView并没有严格的**ViewHolder**设计模式。但是在使用**RecyclerView**的时候，Adapter必须实现至少一个**ViewHolder**，必须遵循**ViewHolder**设计模式。
* **定制Item条目** - **ListView**只能实现垂直线性排列的列表视图，与之不同的是，**RecyclerView**可以通过设置*RecyclerView.LayoutManager*来定制不同风格的视图，比如水平滚动列表或者不规则的瀑布流列表。
* **Item动画** - 在**ListView**中没有提供任何方法或者接口，方便开发者实现Item的增删动画。相反地，可以通过设置**RecyclerView**的RecyclerView.ItemAnimator来为条目增加动画效果。
* **设置数据源** - 在**LisView**中针对不同数据封装了各种类型的Adapter，比如用来处理数组的ArrayAdapter和用来展示Database结果的CursorAdapter。相反地，在**RecyclerView**中必须自定义实现RecyclerView.Adapter并为其提供数据集合。
* **设置条目分割线** - 在**ListView**中可以通过设置android:divider属性来为两个Item间设置分割线。如果想为**RecyclerView**添加此效果，则必须使用RecyclerView.ItemDecoration，这种实现方式不仅更灵活，而且样式也更加丰富。
* **设置点击事件** - 在**ListView**中存在AdapterView.OnItemClickListener接口，用来绑定条目的点击事件。但是，很遗憾的是在**RecyclerView**中，并没有提供这样的接口，不过，提供了另外一个接口RcyclerView.OnItemTouchListener，用来响应条目的触摸事件。

**RecyclerView组件**

**RecyclerView.Adapter**

确切的说，Adapter扮演着两个角色。一是，根据不同ViewType创建与之相应的的Item-Layout，二是，访问数据集合并将数据绑定到正确的View上。这就需要我们重写以下两个函数：

* public VH onCreateViewHolder(ViewGroup parent, int viewType) 创建Item视图，并返回相应的ViewHolder
* public void onBindViewHolder(VH holder, int position) 绑定数据到正确的Item视图上。

另外我们还需要重写另一个方法，像ListView-Adapter那样，同样地告诉RecyclerView-Adapter列表Items的总数：

* public int getItemCount() 返回该Adapter所持有的Itme数量

**RecyclerView.ViewHolder**

ViewHolder的基本用法是用来存放View对象。Android团队很早之前就推荐使用“ViewHolder设计模式”，但实际上他们并没有把这种概念强加给开发者，而且也没有要求开发者在Adapter中必须使用**ViewHolder pattern**。那么现在对于这种新型的**RecyclerView.Adapter**，我们必须实现并使用它。

另外值得一提的是，可以通过打印ViewHolder.toString来获取更多有效信息：

@Overridepublic String toString() {

final StringBuilder sb = new StringBuilder("ViewHolder{" +

Integer.toHexString(hashCode()) + " position=" + mPosition + " id=" + mItemId +

", oldPos=" + mOldPosition + ", pLpos:" + mPreLayoutPosition);

if (isScrap()) sb.append(" scrap");

if (isInvalid()) sb.append(" invalid");

if (!isBound()) sb.append(" unbound");

if (needsUpdate()) sb.append(" update");

if (isRemoved()) sb.append(" removed");

if (shouldIgnore()) sb.append(" ignored");

if (isChanged()) sb.append(" changed");

if (isTmpDetached()) sb.append(" tmpDetached");

if (!isRecyclable()) sb.append(" not recyclable(" + mIsRecyclableCount + ")");

if (isAdapterPositionUnknown()) sb.append("undefined adapter position");

if (itemView.getParent() == null) sb.append(" no parent");

sb.append("}");

return sb.toString();

}

因此，一个基本的RecyclerView.Adapter如下：

public class SimplerItemAdapter extends RecyclerView.Adapter<SimplerItemAdapter.SimpleItemViewHolder> {private List<String> items;

public SimplerItemAdapter(@NonNull List<String> dateItems) {

this.items = (dateItems != null ? dateItems : new ArrayList<String>());

}

@Override public SimpleItemViewHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {

View itemView = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item, viewGroup, false);

return new SimpleItemViewHolder(itemView);

}

@Override public void onBindViewHolder(SimpleItemViewHolder viewHolder, int position) {

viewHolder.textView.setText(items.get(position));

}

@Override public int getItemCount() {

return (this.items != null) ? this.items.size() : 0;

}

protected final static class SimpleItemViewHolder extends RecyclerView.ViewHolder {protected TextView textView;

public SimpleItemViewHolder(View itemView) {

super(itemView);

this.textView = (TextView) itemView.findViewById(R.id.text);

}

}

}

**RecyclerView.LayoutManager**

LayoutManager的职责是摆放Item的位置，并且负责决定何时回收和重用Item。

必须为RecyclerView指定LayoutManager，否则会出现以下异常：

AndroidRuntime java.lang.NullPointerException: Attempt to invoke virtual method ‘void android.support.v7.widget.RecyclerView$LayoutManager.onMeasure(android.support.v7.widget.RecyclerView$Recycler, android.support.v7.widget.RecyclerView$State, int, int)’ on a null object reference

* LinearLayoutManager 水平或者垂直的Item视图。
* GridLayoutManager 网格Item视图。
* StaggeredGridLayoutManager 交错的网格Item视图。

当然还有一些很实用的API：

* findFirstVisibleItemPosition() 返回当前第一个可见Item的position
* findFirstCompletelyVisibleItemPosition() 返回当前第一个完全可见Item的position
* findLastVisibleItemPosition() 返回当前最后一个可见Item的position
* findLastCompletelyVisibleItemPosition() 返回当前最后一个完全可见Item的position

LayoutManager当前有且仅有一个抽象函数：

public LayoutParams generateDefaultLayoutParams()

另外值得注意的是，自定义LayoutManager还应该实现以下方法：

/\*\*

\* Scroll to the specified adapter position.

\*

\* Actual position of the item on the screen depends on the LayoutManager implementation.

\* @param position Scroll to this adapter position.

\*/public void scrollToPosition(int position) {

if (DEBUG) {

Log.e(TAG, "You MUST implement scrollToPosition. It will soon become abstract");

}

}

**RecyclerView.ItemDecoration**

通过设置recyclerView.addItemDecoration(new DividerDecoration(this));来改变Item之间的偏移量或者对Item进行装饰。

当然，你也可以对RecyclerView设置多个ItemDecoration，列表展示的时候会遍历所有的ItemDecoration并调用里面的绘制方法，对Item进行装饰。

RecyclerView.ItemDecoration是一个抽象类，可以通过重写以下三个方法，来实现Item之间的偏移量或者装饰效果：

* public void onDraw(Canvas c, RecyclerView parent) 装饰的绘制在Item条目绘制之前调用，所以这有可能被Item的内容所遮挡
* public void onDrawOver(Canvas c, RecyclerView parent) 装饰的绘制在Item条目绘制之后调用，因此装饰将浮于Item之上
* public void getItemOffsets(Rect outRect, int itemPosition, RecyclerView parent) 与padding或margin类似，LayoutManager在测量阶段会调用该方法，计算出每一个Item的正确尺寸并设置偏移量。

**RecyclerView.ItemAnimator**

ItemAnimator能够帮助Item实现独立的动画。

ItemAnimator作触发于以下三种事件：

1. 某条数据被插入到数据集合中
2. 从数据集合中移除某条数据
3. 更改数据集合中的某条数据

幸运的是，在Android中默认实现了一个DefaultItemAnimator，我们可以通过以下代码为Item增加动画效果：

recyclerView.setItemAnimator(new DefaultItemAnimator());

在之前的版本中，当时据集合发生改变时，我们通过调用.notifyDataSetChanged()，来刷新列表，因为这样做会触发列表的重绘，所以并不会出现任何动画效果，因此需要调用一些以notifyItem\*()作为前缀的特殊方法，比如：

* public final void notifyItemInserted(int position) 向指定位置插入Item
* public final void notifyItemRemoved(int position) 移除指定位置Item
* public final void notifyItemChanged(int position) 更新指定位置Item

**Listeners**

很遗憾，RecyclerView并没有像ListView那样提供以下两个Item的点击监听事件

* public void setOnItemClickListener(@Nullable OnItemClickListener listener) Item点击事件监听
* public void setOnItemLongClickListener(OnItemLongClickListener listener) Item长按事件监听

但是存在这样一个触摸事件的监听RecyclerView.OnItemTouchListener虽然变得更灵活，但是对应的代码量和书写难度却有了一定的增长，至少对我是这样的。

至此，所有与本文章相关的代码都可以从[Github](https://github.com/SmartDengg/RecyclerViewingg)上获取到，另外这个仓库中还有一份本人精心制作的PPT，可供参考。

**参考资料：**

[Codepath](https://guides.codepath.com/android) - Codepath Website

来源： <http://www.jianshu.com/p/16712681731e>