|  |
| --- |
| 1、纵向布局、横向布局、网格布局  设置对应的布局管理器：线性布局、网格布局、瀑布流布局。  LinearLayoutManager、GridLayoutManager、StaggeredGridLayoutManager。  Adapter需要重写的三个方法：  public abstract VH onCreateViewHolder(@NonNull ViewGroup parent, int viewType);  public abstract void onBindViewHolder(@NonNull VH viewHolder, int position);  public abstract int getItemCount();  需要定义一个ViewHolder内部类。在一些有不同的类型viewHolder时，要会进行继承、拓展viewHolder。  public class GridViewHolder extends RecyclerView.ViewHolder;  2、点击事件不断迭代优化  （1）在Adapter里面直接对控件做点击事件  holder.view.setOnClickListener(new View.OnClickListener() {  .....  //todo  });  （2）写接口，在Activity或Fragment上实现接口中定义的方法  在adapter中定义接口Listener；  在activity中实现接口，并给adapter设置listener；  然后  @Override  public void onBindViewHolder(@NonNull XxxViewHolder holder, int position) {  holder.控件名.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  mListener.onItemClick(content);  }  });  }  （3）复用点击事件监听器。  复用OnClickListener对象，然后在onBindViewHolder()方法中通过setTag(position) 和getTag() 的方式，来传递点击事件的position给listener，listener保存在ViewHolder对象中。  3、分组： 组名+内容在同一布局，通过逻辑判断显示/隐藏。  顶部悬浮（吸顶）：分组逻辑 + 实现组名view的动画。  4、拖动、滑动删除  关键字：ItemTouchHelper  ItemTouchHelper.Callback callback = new DragItemTouchHelper(adapter);  ItemTouchHelper touchHelper = new ItemTouchHelper(callback);  touchHelper.attachToRecyclerView(rcvDrag);    5、下拉刷新：  布局中使用SwipeRefreshLayout包裹RecyclerView，SwipeRefreshLayout设置一个监听器即可。  上拉加载：  recyclerview.addOnScrollListener，在onScrollStateChanged里判断RecyclerView的状态是空闲时，同时是最后一个可见的item时才加载，在onScrolled里获取最后一个可见的item。  6、双向滑动  采用adapter内部创建的ViewHolder是一个列表recyclerView，列表嵌套。  7、居中对齐snapHelper  LinearSnapHelper snapHelper = new LinearSnapHelper();  snapHelper.attachToRecyclerView(rcvSnapHelper);    8、性能优化  （1）数据处理与视图绑定分离：一些耗时的，复杂是数据处理不要在bindViewHolder()这里进行。  （2）获取数据的优化：分页加载远端数据、http请求接口缓存、局部刷新列表  （3）布局优化  （4）减少view对象的创建；当然其他的对象也尽量减少创建，尽量复用。  （5）设置固定高度：如果item高度是固定的话，可以使用RecyclerView.setHasFixedSize(true);来避免requestLayout浪费资源。  （6）共用RecycledViewPool：在嵌套RecyclerView中，如果子RecyclerView具有相同的adapter，那么可以设置RecyclerView.setRecycledViewPool(pool)来共用一个RecycledViewPool。  （7）加大RecyclerView的缓存  用空间换时间，来提高滚动的流畅性。  recyclerView.setItemViewCacheSize(20);  recyclerView.setDrawingCacheEnabled(true);  recyclerView.setDrawingCacheQuality(View.DRAWING\_CACHE\_QUALITY\_HIGH);  RecyclerView可以设置自己所需要的ViewHolder缓存数量，默认大小是2。cacheViews中的缓存只能position相同才可得用，且不会重新bindView，CacheViews满了后移除到RecyclerPool中，并重置ViewHolder，如果对于可能来回滑动的RecyclerView，把CacheViews的缓存数量设置大一些，可以减少bindView的时间，加快布局显示。  注：此方法是拿空间换时间，要充分考虑应用内存问题，根据应用实际使用情况设置大小。  （8）减少ItemView监听器的创建  在onBindViewHolde方法中，减少逻辑判断，减少临时对象创建。例如：复用事件监听，在其方法外部创建监听，可以避免生成大量的临时变量。  复用OnClickListener对象，然后在onBindViewHolder()方法中通过setTag(position) 和getTag() 的方式，来传递点击事件的position给listener，listener保存在ViewHolder对象中。  （9）优化滑动操作  设置RecyclerView.addOnScrollListener();来在滑动过程中停止加载的操作。这个停止了加载，怎么恢复请求呢，比如加载到一半，下滑不见了，然后上滑，又可见了。  其实只需要重写一下RecycleView的onScrollStateChanged方法在里面约定好Gilde是否可以加载图片。  （10）最重要还是列表中图片加载的优化 |

# 纵向布局、横向布局、网格布局

设置对应的布局管理器：线性布局、网格布局、瀑布流布局。

LinearLayoutManager、GridLayoutManager、StaggeredGridLayoutManager。

Adapter需要重写的三个方法：

public abstract VH onCreateViewHolder(@NonNull ViewGroup parent, int viewType);

public abstract void onBindViewHolder(@NonNull VH viewHolder, int position);

public abstract int getItemCount();

需要定义一个ViewHolder内部类。在一些有不同的类型viewHolder时，要会进行继承、拓展viewHolder。

public class GridViewHolder extends RecyclerView.ViewHolder;

|  |
| --- |
| 纵向布局  recyclerview.setLayoutManager(new LinearLayoutManager(this));  或  LinearLayoutManager manager = new LinearLayoutManager(this);  manager.setOrientation(LinearLayoutManager.VERTICAL);  recyclerview.setLayoutManager(manager); |
| 横向布局  LinearLayoutManager manager = new LinearLayoutManager(this);  manager.setOrientation(LinearLayoutManager.HORIZONTAL);  recyclerview.setLayoutManager(manager); |
| 网格布局  recyclerview.setLayoutManager(new GridLayoutManager(this, 横排数量));  或  GridLayoutManager manager = new GridLayoutManager(this, 横排数量);  recyclerview.setLayoutManager(manager); |
| public class GridAdapter extends RecyclerView.Adapter<GridAdapter.GridViewHolder> {  private Context mContext;  private List<String> mList = new ArrayList<>();  public GridAdapter(Context context) {  mContext = context;  }  public void setGridDataList(List<String> list) {  mList = list;  notifyDataSetChanged();  }  @NonNull  @Override  public GridViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  View view = LayoutInflater.from(mContext).inflate(R.layout.grid\_recycle\_item, parent, false);  return new GridViewHolder(view);  }  @Override  public void onBindViewHolder(@NonNull GridViewHolder holder, int position) {  holder.tvNum.setText(position + 1 + "");  holder.tvContent.setText(mList.get(position));  }  @Override  public int getItemCount() {  return mList == null ? 0 : mList.size();  }  public class GridViewHolder extends RecyclerView.ViewHolder {  TextView tvNum, tvContent;  public GridViewHolder(View itemView) {  super(itemView);  tvNum = itemView.findViewById(R.id.tv\_num);  tvContent = itemView.findViewById(R.id.tv\_content);  }  }  } |

# 2、点击事件

RecyclerView的点击事件有两种写法：

1. 在Adapter里面直接对控件做点击事件

holder.view.setOnClickListener(new View.OnClickListener() {

.....

//todo

});

② 写接口，在Activity或Fragment上实现接口中定义的方法

在adapter中定义接口Listener；

在activity中实现接口，并给adapter设置listener；

然后在

@Override

public void onBindViewHolder(@NonNull XxxViewHolder holder, int position) {

holder.控件名.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

mListener.onItemClick(content);

}

});

}

# 3、分组

第一种分组写法也是最简单的一种，将组名和内容写在同一个布局，在onBindViewHolder里根据位置来进行组名的显示与隐藏：

|  |
| --- |
| if (position == 0) {  holder.组名.setVisibility(View.VISIBLE);  } else {  if (list.get(position).get组().equals(list.get(position - 1).get组())) {  holder.组名.setVisibility(View.GONE);  } else {  holder.组名.setVisibility(View.VISIBLE);  }  } |

# 4、顶部悬浮（吸顶）

结合上面分组的实现 + 滑动监听器里面实现顶部view的上滑的动画。

rcvSticky.addOnScrollListener(new RecyclerView.OnScrollListener() {...});

在recyclerview中使API传递特定的值。

setTag(); / getTag();

getContentDescription(); / setContentDescription();

holder.itemView.setTag(HAS\_STICKY\_VIEW);

holder.itemView.setContentDescription(stickyData.area);

# 5、拖动、滑动删除

关键字：ItemTouchHelper

ItemTouchHelper.Callback callback = new DragItemTouchHelper(adapter);

ItemTouchHelper touchHelper = new ItemTouchHelper(callback);

touchHelper.attachToRecyclerView(rcvDrag);

# 下拉刷新

使用SwipeRefreshLayout包裹RecyclerView.

SwipeRefreshLayout设置一个监听器即可。

// 下拉刷新

mRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {

@Override

public void onRefresh() {

initData();

}

});

# 7、上拉加载

关键字：recyclerview.addOnScrollListener，在onScrollStateChanged里判断RecyclerView的状态是空闲时，同时是最后一个可见的item时才加载，在onScrolled里获取最后一个可见的item。

rcvLoad.addOnScrollListener(new RecyclerView.OnScrollListener() {

@Override

public void onScrollStateChanged(RecyclerView recyclerView, int newState) {

super.onScrollStateChanged(recyclerView, newState);

// 判断 RecyclerView 的状态是空闲时，同时是最后一个可见的 item 时才加载

if (newState == RecyclerView.SCROLL\_STATE\_IDLE &&

lastVisibleItem + 1 == mAdapter.getItemCount()) {

Log.d(TAG, "onScrollStateChanged: 上拉加载");

total = total - count;

Log.d(TAG, "onScrollStateChanged: " + total);

if (total > count) {

Log.d(TAG, "onScrollStateChanged: >");

count = 10;

start = 10;

initData(start, count);

} else if (total < count && total > 0) {

Log.d(TAG, "onScrollStateChanged: <");

count = total;

start = 20;

initData(start, count);

} else if (total == 0) {

Log.d(TAG, "onScrollStateChanged: =");

count = total;

Toast.makeText(LoadActivity.this, "已经没有数据了", Toast.LENGTH\_SHORT).show();

}

}

}

@Override

public void onScrolled(RecyclerView recyclerView, int dx, int dy) {

super.onScrolled(recyclerView, dx, dy);

// 最后一个可见的 item

lastVisibleItem = mLinearLayoutManager.findLastVisibleItemPosition();

}

});

# 8、双向滑动

采用adapter内部创建的ViewHolder是一个列表recyclerView，列表嵌套。

# 9、居中对齐snapHelper

LinearSnapHelper snapHelper = new LinearSnapHelper();

snapHelper.attachToRecyclerView(rcvSnapHelper);

# 10、展开和收缩

在item里将主内容和副内容写出来，通关点击item副内容现实和隐藏来达到效果。

private int expandedPosition = -1;

final boolean isExpanded = position == expandedPosition;

holder.rlChild.setVisibility(isExpanded ? View.VISIBLE : View.GONE);

holder.rlParent.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

if (mViewHolder != null) {

mViewHolder.rlChild.setVisibility(View.GONE);

notifyItemChanged(expandedPosition);

}

expandedPosition = isExpanded ? -1 : holder.getAdapterPosition();

mViewHolder = isExpanded ? null : holder;

notifyItemChanged(holder.getAdapterPosition());

}

});

# 11、性能优化

## （1）数据处理与视图绑定分离

RecyclerView的bindViewHolder方法是在UI线程进行的，如果在该方法进行耗时操作，将会影响滑动的流畅性。

一些耗时的，复杂是数据处理不要在bindViewHolder()这里进行。

## 获取数据的优化

分页加载远端数据；

http请求缓存：对拉取的远端数据进行缓存，提高二次加载速度；

局部刷新：对于新增或删除数据通过DiffUtil，来进行局部数据刷新，而不是一味的全局刷新数据。

## （3）布局优化

减少过度绘制：减少布局层级，可以考虑使用自定义View来减少层级，或者更合理的设置布局来减少层级。

减少xml文件inflate时间。

减少view对象的创建，使用自定义控件，但是设计的问题也很多。

当然其他的对象也尽量减少创建，尽量复用。

设置固定高度：如果item高度是固定的话，可以使用RecyclerView.setHasFixedSize(true);来避免requestLayout浪费资源。可以的话，设置RecyclerView布局等高，然后设置recyclerView.setHasFixedSize(true)这样可以避免每次绘制Item时，不再重新计算Item高度。

## （4）共用RecycledViewPool

在嵌套RecyclerView中，如果子RecyclerView具有相同的adapter，那么可以设置RecyclerView.setRecycledViewPool(pool)来共用一个RecycledViewPool。

|  |
| --- |
| 在TabLayout+ViewPager+RecyclerView的场景中，当多个RecyclerView有相同的item布局结构时，多个RecyclerView共用一个RecycledViewPool可以避免创建ViewHolder的开销，避免GC。RecycledViewPool对象可通过RecyclerView对象获取，也可以自己实现。  RecycledViewPool mPool = mRecyclerView1.getRecycledViewPool();  下一个RecyclerView可直接进行setRecycledViewPool  mRecyclerView2.setRecycledViewPool(mPool);  mRecyclerView3.setRecycledViewPool(mPool); |

## （5）加大RecyclerView的缓存

用空间换时间，来提高滚动的流畅性。

recyclerView.setItemViewCacheSize(20);

recyclerView.setDrawingCacheEnabled(true);

recyclerView.setDrawingCacheQuality(View.DRAWING\_CACHE\_QUALITY\_HIGH);

|  |
| --- |
| RecyclerView可以设置自己所需要的ViewHolder缓存数量，默认大小是2。cacheViews中的缓存只能position相同才可得用，且不会重新bindView，CacheViews满了后移除到RecyclerPool中，并重置ViewHolder，如果对于可能来回滑动的RecyclerView，把CacheViews的缓存数量设置大一些，可以减少bindView的时间，加快布局显示。  注：此方法是拿空间换时间，要充分考虑应用内存问题，根据应用实际使用情况设置大小。  网上大部分设置CacheView大小时都会带上：  setDrawingCacheEnabled(true)和setDrawingCacheQuality(View.DRAWING\_CACHE\_QUALITY\_HIGH)  setDrawingCacheEnabled这个是View本身的方法，意途是开启缓存。通过setDrawingCacheEnabled把cache打开，再调用getDrawingCache就可以获得view的cache图片，如果cache没有建立，系统会自动调用buildDrawingCache方法来生成cache。一般截图会用到，这里的设置drawingcache，可能是在重绘时不需要重新计算bitmap的宽高等，能加快dispatchDraw的速度，但开启drawingcache，肯定也会耗应用的内存，所以也慎用。 |

## （6）减少ItemView监听器的创建

对ItemView设置监听器，不要对每个item都创建一个监听器，而应该共用一个XxListener，然后根据ID来进行不同的操作，优化了对象的频繁创建带来的资源消耗。

或者使用这个替代

rcvVertical.addOnItemTouchListener(new RecyclerView.OnItemTouchListener());

在onBindViewHolde方法中，减少逻辑判断，减少临时对象创建。例如：复用事件监听，在其方法外部创建监听，可以避免生成大量的临时变量。

复用OnClickListener对象，然后在onBindViewHolder()方法中通过setTag(position) 和getTag() 的方式，来传递点击事件的position给listener，listener保存在ViewHolder对象中。

|  |
| --- |
| private class XXXHolder extends RecyclerView.ViewHolder {  private EditText mEt;  EditHolder(View itemView) {  super(itemView);  mEt = (EditText) itemView;  mEt.setOnClickListener(mOnClickListener);  }    private View.OnClickListener mOnClickListener = new View.OnClickListener() {  @Override  public void onClick(View v) {  //do something  }  }  } |

## （7）优化滑动操作

设置RecyclerView.addOnScrollListener();来在滑动过程中停止加载的操作。这个停止了加载，怎么恢复请求呢，比如加载到一半，下滑不见了，然后上滑，又可见了。

|  |
| --- |
| 其实只需要重写一下RecycleView的onScrollStateChanged方法在里面约定好Gilde是否可以加载图片。  我们先说一下onScrollStateChanged的几种状态。  SCROLL\_STATE\_IDLE 屏幕停止滚动  SCROLL\_STATE\_DRAGGING 屏幕滚动且用户使用的触碰或手指还在屏幕上  SCROLL\_STATE\_SETTLING 由于用户的操作，屏幕产生惯性滑动  而Gilde同时也为我们提供了两个方法  resumeRequests() 开始加载图片  pauseRequests() 停止加载图片 |
| //监听滚动来对图片加载进行判断处理  public class ImageAutoLoadScrollListener extends OnScrollListener{  @Override  public void onScrolled(RecyclerView recyclerView, int dx, int dy) {  super.onScrolled(recyclerView, dx, dy);  }  @Override  public void onScrollStateChanged(RecyclerView recyclerView, int newState) {  super.onScrollStateChanged(recyclerView, newState);  switch (newState){  case SCROLL\_STATE\_IDLE: // The RecyclerView is not currently scrolling.  //当屏幕停止滚动，加载图片  try {  if(getContext() != null) Glide.with(getContext()).resumeRequests();  }  catch (Exception e) {  e.printStackTrace();  }  break;  case SCROLL\_STATE\_DRAGGING: // The RecyclerView is currently being dragged by outside input such as user touch input.  //当屏幕滚动且用户使用的触碰或手指还在屏幕上，停止加载图片  try {  if(getContext() != null) Glide.with(getContext()).pauseRequests();  }  catch (Exception e) {  e.printStackTrace();  }  break;  case SCROLL\_STATE\_SETTLING: // The RecyclerView is currently animating to a final position while not under outside control.  //由于用户的操作，屏幕产生惯性滑动，停止加载图片  try {  if(getContext() != null) Glide.with(getContext()).pauseRequests();  }  catch (Exception e) {  e.printStackTrace();  }  break;  }  }  } |

## （8）最重要还是列表中图片加载的优化

好好看看。

## setTag传递position

在onBindViewHolder方法中将position通过tag方式绑定到控件上。

RecyclerView是support:recyclerview-v7中提供的控件，最低兼容到android 3.0版本。

官方介绍RecyclerView为在有限的窗口展现大量数据的控件。拥有类似功能的控件有ListView、GridView以及被Google遗弃的Gallery等，为毛已经有了它们，Google还推出RecyclerView呢，那就要说说RecyclerView所具有的一些优势了。

那RecyclerView到底有啥优势呢？总结起来六颗字：低耦合高类聚。RecyclerView已经标准化ViewHolder，我们自定义的ViewHoler需要继承 RecyclerView.ViewHolder，然后在构造方法中初始化控件，后面会有具体介绍。通过设置不同的LayoutManager，以及结合ItemDecoration , ItemAnimator，ItemTouchHelper，可以实现非常炫酷的效果，这些是ListView等控件难以企及的。

# 基本使用：

## 1.1 添加依赖

implementation 'com.android.support:recyclerview-v7:27.0.2'

## 1.2 RecyclerView的使用

编写代码，首先我们需要在Xml中写RecyclerView的布局

<android.support.v7.widget.RecyclerView

android:id="@+id/recyclerView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

然后在activity中获取RecyclerView，并设置LayoutManager以及adapter。

//通过findViewById拿到RecyclerView实例

mRecyclerView = findViewById(R.id.recyclerView);

//设置RecyclerView布局管理器

mRecyclerView.setLayoutManager(new LinearLayoutManager(this,

LinearLayoutManager.VERTICAL, false));

//初始化适配器

mAdapter = new MyRecyclerViewAdapter(list);

//设置添加或删除item时的动画，这里使用默认动画

mRecyclerView.setItemAnimator(new DefaultItemAnimator());

//设置适配器

mRecyclerView.setAdapter(mAdapter);

◦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

## 1.3 MyRecyclerViewAdapter的代码

package com.sharejoys.recyclerviewdemo.actvity;

import android.support.v7.widget.RecyclerView;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

import com.sharejoys.recyclerviewdemo.R;

import java.util.List;

/\*\*

\* Created by 青青-子衿 on 2018/1/15.

\*/

public class MyRecyclerViewAdapterextends RecyclerView.Adapter<MyAdapter.ViewHolder> {

private List<String> list;

public MyAdapter(List<String> list) {

this.list = list;

}

@Override

public MyAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item\_base\_use, parent, false);

MyAdapter.ViewHolder viewHolder = new MyAdapter.ViewHolder(view);

return viewHolder;

}

@Override

public void onBindViewHolder(MyAdapter.ViewHolder holder, int position) {

holder.mText.setText(list.get(position));

}

@Override

public int getItemCount() {

return list.size();

}

class ViewHolder extends RecyclerView.ViewHolder {

TextView mText;

ViewHolder(View itemView) {

super(itemView);

mText = itemView.findViewById(R.id.item\_tx);

}

}

}

这里item\_normal的布局也非常简单

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_content"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView

android:id="@+id/item\_tx"

android:layout\_width="match\_content"

android:layout\_height="wrap\_content"

android:gravity="center"

android:padding="10dp"

android:layout\_gravity="center\_horizontal"

android:text="Item"/>

</LinearLayout>

从例子也可以看出来，RecyclerView的用法并不比ListView复杂，反而更灵活好用，它将数据、排列方式、数据的展示方式都分割开来，因此可定制型，自定义的形式也非常多，非常灵活。

（1）RecyclerView.Adapter扮演的角色

•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

重写的方法:

一般常用的重写方法有以下这么几个：

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。

public int getItemCount()

返回该Adapter所持有的Itme数量

public int getItemViewType(int position)

用来获取当前项Item(position参数)是哪种类型的布局

（2）notifyDataSetChanged()刷新数据

•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

（3）ViewHolder与复用

•在复写RecyclerView.Adapter的时候，需要我们复写两个方法：

◦onCreateViewHolder

◦onBindViewHolder

◦这两个方法从字面上看就是创建ViewHolder和绑定ViewHolder的意思

•复用机制是怎样的？

◦模拟场景：只有一种ViewType，上下滑动的时候需要的ViewHolder种类是只有一种，但是需要的ViewHolder对象数量并不止一个。所以在后面创建了5个ViewHolder之后，需要的数量够了，无论怎么滑动，都只需要复用以前创建的对象就行了。那么逗比程序员们思考一下，为什么会出现这种情况呢

◦看到了下面log之后，第一反应是在这个ViewHolder对象的数量“够用”之后就停止调用onCreateViewHolder方法，但是onBindViewHolder方法每次都会调用的

•对于ViewHolder对象的数量“够用”之后就停止调用onCreateViewHolder方法，可以查看

◦获取为给定位置初始化的视图。

◦此方法应由{@link LayoutManager}实现使用，以获取视图来表示来自{@LinkAdapter}的数据。

◦如果共享池可用于正确的视图类型，则回收程序可以重用共享池中的废视图或分离视图。如果适配器没有指示给定位置上的数据已更改，则回收程序将尝试发回一个以前为该数据初始化的报废视图，而不进行重新绑定。

## 1.4布局管理器

设置竖向布局：

mRecyclerView.setLayoutManager(new LinearLayoutManager(this,

LinearLayoutManager.VERTICAL, false));

设置横向布局：

mRecyclerView.setLayoutManager(new LinearLayoutManager(this,

LinearLayoutManager.HORIZONTAL, false));

设置网格布局：

mRecyclerView.setLayoutManager(new GridLayoutManager(this, 3));

设置瀑布流：

mRecyclerView.setLayoutManager(new StaggeredGridLayoutManager(3,StaggeredGridLayoutManager.VERTICAL));

从以上可知，我们可以通过设置不同的管理器，实现不同的效果  
LinearLayoutManager：以线性布局展示，可以设置横向和纵向  
GridLayoutManager：以网格形式展示，类似GridView效果  
StaggeredGridLayoutManager：以瀑布流形式的效果  
  
RecyclerView条目之间默认没有分割线，那是否可以像ListView一样设置divider以及dividerHight搞一条分割线出来呢，答案是不可以的，google并没有提供这样的属性。但是谷歌为我们提供了可以定制的解决办法，那就是以下要说ItemDecoration。

## 1.5 利用ItemDecoration实现条目分割线

ItemDecoration是谷歌定义的可用于画分割线的类， 是抽象的，需要我们自己去实现

## 1.6 点击事件

RecyclerView并没有像ListView的那样可以设置点击事件以及长按点击事件，这个需要我们可以在adapter中去设置回调的方式实现。

## 1.7 temAnimator

我们可以为RecyclerView设置增加和删除动画，这里我们可以使用默认动画

//设置添加或删除item时的动画，这里使用默认动画

mRecyclerView.setItemAnimator(new DefaultItemAnimator());

然后在Adapter中增加删除和添加数据的方法

/\*\*

\* 插入一条数据

\*

\* @param index 下标

\* @param s 数据

\*/

public void addItem(int index, String s) {

list.add(index, s);

notifyItemInserted(index);

}

/\*\*

\* 删除一条数据

\*

\* @param index 下标

\*/

public void deleteItem(int index) {

list.remove(index);

notifyItemRemoved(index);

}

# 复杂的列表布局

复杂的布局：轮播图、列表布局、2列的网格布局、3列的网格布局、最后是瀑布流式布局。

## 2.1 方案一RecyclerView的2级嵌套

看到同一个滚动控件中出现了3种混合布局，多数人第一映像就是进行嵌套。

如果进行嵌套的话，嵌套什么？从效果图来看，上图的一个栏目中的视图数量似乎是固定的，这意味着可以使用RelativeLayout等布局进行硬排版。确实如果真是固定的这样做当然更好，但是注意到点击每个栏目上的刷新按钮的时候，偶尔会出现两个视图交换位置的动画，这是RecyclerView特有的，而且也没有人告诉我每个栏目中的视图数量就是固定，万一哪天又多了一排呢，所以我们还是嵌套RecyclerView。

这种方式我不多介绍，因为在前一篇也说过，google不建议对RecyclerView、ListView进行嵌套，嵌套使用的话那么视图复用机制就等于白费了，**依然会在第一次加载时初始化所有的item，如果数据量过大则会产生性能障碍，**所以我也遵循google的建议不推荐使用嵌套解决问题，我更加推荐下面一种方案。

## 2.2 方案二：使用特殊的(自定义)布局管理器

这种方案是我认为最为优秀的做法，它完全符合Google制定的标准：使用布局管理器来管理布局。我们继续观察首页布局的图示，我们真的要为了实现这种混合布局自己去写一个布局管理器吗？我们发现上面出现了列表、网格、瀑布流3种交叉混排的混合布局。我们先把瀑布流放在一边，仔细想想如果我们把网格的列数设置为1列，那不就是一个列表布局吗，也就是说我们使用网格布局管理器就可以做出列表的样式，所以说虽然是说用自定义布局管理器，但实际上不需要我们自定义，GridLayoutManager为我们提供了动态改变每个item所占列数的方法：

gridManager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {

@Override

public int getSpanSize(int position) {

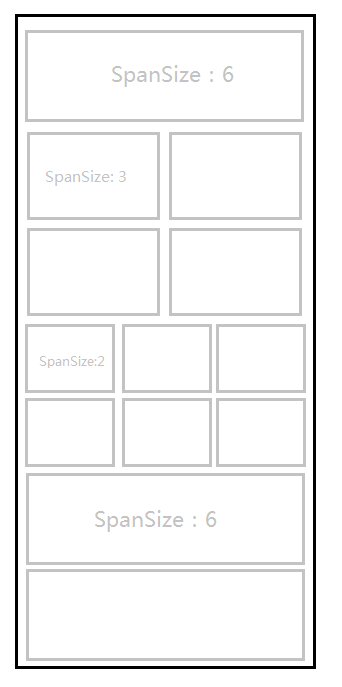
return gridManager.getSpanCount();

}

}

getSpanSize方法，返回值就表示当前item占多少列，例如如果我们列数设置的为3列，返回3就表示铺满，也就是和列表一样了。

如图所示，我们给RecyclerView设置一个列数为6的GridLayoutManager，然后再动态地为不同部位的item分别设置SpanSize为6（铺满）、3（1/2）、2（1/3）就行了。



|  |
| --- |
| 根据GridLayoutManager的getSpanSize()方法可以动态的设置item跨列数；  mRecyclerView.setLayoutManager(new GridLayoutManager(getContext(), 6));  return 6表示独占一个item 比如一张图片占满一行  return 3 3是6的多少？2分之1啊，那就是占一行的一半 也就是说 如果放图片 一行可以最大放2张！！！  return 2 2是6的多少？3分之1啊，那就是占一行的3分之1 也就是说 如果放图片 一行可以最大放3张！！！ |

设置一个列数为6的GridLayoutManager：

recyclerView.setLayoutManager(new GridLayoutManager(recyclerView.getContext(), 6,

GridLayoutManager.VERTICAL, false));

在onAttachedToRecyclerView方法中动态为不同position设置不同的SpanSize：

@Override

public void onAttachedToRecyclerView(final RecyclerView recyclerView) {

super.onAttachedToRecyclerView(recyclerView);

RecyclerView.LayoutManager manager = recyclerView.getLayoutManager();

if(manager instanceof GridLayoutManager) {

final GridLayoutManager gridManager = ((GridLayoutManager) manager);

gridManager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {

@Override

public int getSpanSize(int position) {

int type = getItemViewType(position);

switch (type){

case TYPE\_SLIDER:

case TYPE\_TYPE2\_HEAD:

case TYPE\_TYPE3\_HEAD:

return 6;

case TYPE\_TYPE2:

return 3;

case TYPE\_TYPE3:

return 2;

default:

return 3;

}

}

});

}

}

# 列表控件卡顿优化

1. 使用ConstraintLayout减少布局层级。

减少过度绘制，减少布局层级，可以考虑使用自定义View来减少层级，或者更合理的设置布局来减少层级。

（2）可以的话，设置RecyclerView布局等高，然后设置recyclerView.setHasFixedSize(true)这样可以避免每次绘制Item时，不再重新计算Item高度。

（3）根据需求修改RecyclerView默认的绘制缓存选项

recyclerView.setItemViewCacheSize(20);

recyclerView.setDrawingCacheEnabled(true);

recyclerView.setDrawingCacheQuality(View.DRAWING\_CACHE\_QUALITY\_HIGH);

（4）在onBindViewHolder/getView方法中，减少逻辑判断，减少临时对象创建。例如：复用事件监听，在其方法外部创建监听，可以避免生成大量的临时变量。

（5）减少View对象的创建

一个稍微复杂的 Item 会包含大量的 View，而大量的 View 的创建也会消耗大量时间，所以要尽可能简化 ItemView；设计 ItemType 时，对多 ViewType 能够共用的部分尽量设计成自定义 View，减少 View 的构造和嵌套。

（6）RecyclerView的bindViewHolder方法是在UI线程进行的，如果在该方法进行耗时操作，将会影响滑动的流畅性

（7）数据优化。分页加载远端数据，对拉取的远端数据进行缓存，提高二次加载速度；

（8）对于新增或删除数据通过DiffUtil，来进行局部数据刷新，而不是一味的全局刷新数据。