**[RecyclerView添加Header的正确方式](http://blog.csdn.net/qibin0506/article/details/49716795)**

这篇博客我们要解决的问题有：

1. 如何为RecyclerView添加Header
2. 如何让Header适配各种LayoutManager
3. 在有Header的情况下，我们的分割线该怎么画
4. 作为一个懒惰的程序员，如何将这些做到最简便

## 如果为RecyclerView添加Header

大家在使用ListView的时候可以很轻松的添加headers， 但是不知道大家发现没有，RecyclerView和各种LayoutManager都没有哪个方法是为添加header而设立的，这个时候我们就开始思考如何为RecyclerView添加header了。 这里我们的解决方案和网上你能搜到的大多数方案一样，是通过控制Adapter的itemType来设置的，思路就是**根据不同的itemType去加载不同的布局**。

/\*\*

\* Created by qibin on 2015/11/5.

\*/public class MyAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {

public static final int TYPE\_HEADER = 0;

public static final int TYPE\_NORMAL = 1;

private ArrayList<String> mDatas = new ArrayList<>();

private View mHeaderView;

private OnItemClickListener mListener;

public void setOnItemClickListener(OnItemClickListener li) {

mListener = li;

}

public void setHeaderView(View headerView) {

mHeaderView = headerView;

notifyItemInserted(0);

}

public View getHeaderView() {

return mHeaderView;

}

public void addDatas(ArrayList<String> datas) {

mDatas.addAll(datas);

notifyDataSetChanged();

}

@Override

public int getItemViewType(int position) {

if(mHeaderView == null) return TYPE\_NORMAL;

if(position == 0) return TYPE\_HEADER;

return TYPE\_NORMAL;

}

@Override

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

if(mHeaderView != null && viewType == TYPE\_HEADER) return new Holder(mHeaderView);

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

return new Holder(layout);

}

@Override

public void onBindViewHolder(RecyclerView.ViewHolder viewHolder, int position) {

if(getItemViewType(position) == TYPE\_HEADER) return;

final int pos = getRealPosition(viewHolder);

final String data = mDatas.get(pos);

if(viewHolder instanceof Holder) {

((Holder) viewHolder).text.setText(data);

if(mListener == null) return;

viewHolder.itemView.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

mListener.onItemClick(pos, data);

}

});

}

}

public int getRealPosition(RecyclerView.ViewHolder holder) {

int position = holder.getLayoutPosition();

return mHeaderView == null ? position : position - 1;

}

@Override

public int getItemCount() {

return mHeaderView == null ? mDatas.size() : mDatas.size() + 1;

}

class Holder extends RecyclerView.ViewHolder {

TextView text;

public Holder(View itemView) {

super(itemView);

if(itemView == mHeaderView) return;

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

}

}

interface OnItemClickListener {

void onItemClick(int position, String data);

}

}

这里我们重写了getItemViewType方法，并根据位置来返回不同的type，这个type是我们预先商定好的常量，接在onCreateViewHolder方法中来判断itemType，如果是header，则返回我们设置的headerView，否则正常加载item布局，相信大家对于上面的代码不会有任何疑问，接下来我们就在Activity中用一下试试看，

@Overrideprotected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mRecyclerView = (RecyclerView) findViewById(R.id.list);

mLayoutManager = new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false);

mRecyclerView.setLayoutManager(mLayoutManager);

mRecyclerView.setItemAnimator(new DefaultItemAnimator());

mAdapter = new MyAdapter();

mRecyclerView.setAdapter(mAdapter);

mAdapter.addDatas(generateData());

setHeader(mRecyclerView);

mAdapter.setOnItemClickListener(new MyAdapter.OnItemClickListener() {

@Override

public void onItemClick(int position, String data) {

Toast.makeText(MainActivity.this, data, Toast.LENGTH\_SHORT).show();

}

});

}

private void setHeader(RecyclerView view) {

View header = LayoutInflater.from(this).inflate(R.layout.header, view, false);

mAdapter.setHeaderView(header);

}

这里LayoutManager我们使用了LinearLayoutManager，并且给Adapter设置了一个header，运行一下   
看看效果：

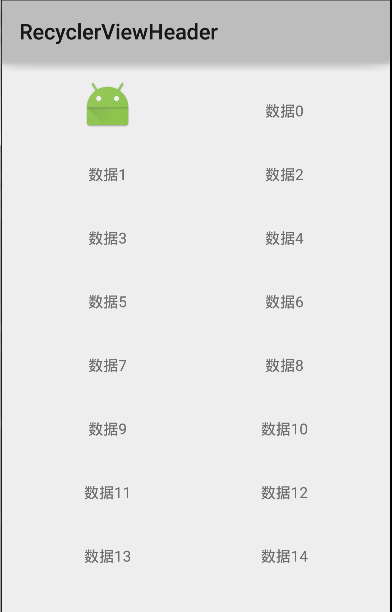


恩，还不错，item的点击事件也很完美，那接下来，我们将LayoutManager换成GridLayoutManager看看咋样。

## 为GridLayoutManager添加header

// mLayoutManager = new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false);

mLayoutManager = new GridLayoutManager(this, 2);



哎哟，我的小心脏啊，快受不了了，这是什么玩意，我们的header竟然作为一个cell出现在了界面上，这完全不是我们想要的效果啊！ 冷静下来想想，肯定会有解决方法的吧。这时候我们就该引入一个不太常用的方法了：

gridManager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {

@Override

public int getSpanSize(int position) {

return getItemViewType(position) == TYPE\_HEADER

? gridManager.getSpanCount() : 1;

}

});

我们解释一下这段代码，首先我们设置了一个SpanSizeLookup，这个类是一个抽象类，而且仅有一个抽象方法getSpanSize，这个方法的返回值决定了我们每个position上的item占据的单元格个数，而我们这段代码综合上面为GridLayoutManager设置的每行的个数来解释的话，   
就是**当前位置是header的位置，那么该item占据2个单元格，正常情况下占据1个单元格**。那这段代码放哪呢？ 为了以后的封装，我们还是在Adapter中找方法放吧。   
我们在Adapter中再重写一个方法onAttachedToRecyclerView，

@Overridepublic void onAttachedToRecyclerView(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) {

return getItemViewType(position) == TYPE\_HEADER

? gridManager.getSpanCount() : 1;

}

});

}

}

这个时候我们再来看一下效果，



恩，这次达到我们的要求了，不过对于StaggeredGridLayoutManager我们还没做处理，而且我们还发现StaggeredGridLayoutManager中并没有像GridLayoutManager中这样的方法，我们还需要单独为StaggeredGridLayoutManager单独处理一下。

## 为StaggeredGridLayoutManager添加header

我们继续重写Adapter中另外一个方法。

@Overridepublic void onViewAttachedToWindow(RecyclerView.ViewHolder holder) {

super.onViewAttachedToWindow(holder);

ViewGroup.LayoutParams lp = holder.itemView.getLayoutParams();

if(lp != null

&& lp instanceof StaggeredGridLayoutManager.LayoutParams) {

StaggeredGridLayoutManager.LayoutParams p = (StaggeredGridLayoutManager.LayoutParams) lp;

p.setFullSpan(holder.getLayoutPosition() == 0);

}

}

这里的处理方式是用通过LayoutParams，而且这里更简单，StaggeredGridLayoutManager.LayoutParams为我们提供了一个setFullSpan方法来设置占领全部空间，好开心，看一下StaggeredGridLayoutManager的效果，



啊， 怎么和上面的效果一样？ 很简单嘛，我们的item都是等高的。

## 处理分隔符

这是我们开开心心的继续写代码，并且为我们的item添加了分隔符，分隔符我还是用的翔哥写的那个，毕竟翔哥写的太好了，而且我们没有必要重复造轮子，不过这时候问题出现了，相信你也肯定能猜到应该会出现问题了，因为不管我们怎么处理，header对于RecyclerView来说还是一个普普通通的item，这时候我们添加分割线，肯定也会对header产生影响，那下面，我们再来对翔哥的分割线改造一下吧。

public class GridItemDecoration extends RecyclerView.ItemDecoration {

private static final int[] ATTRS = new int[]{android.R.attr.listDivider};

private Drawable mDivider;

private boolean hasHeader;

public GridItemDecoration(Context context) {

final TypedArray a = context.obtainStyledAttributes(ATTRS);

mDivider = a.getDrawable(0);

a.recycle();

}

public GridItemDecoration(Context context, boolean header) {

this(context);

hasHeader = header;

}

...

@Override

public void getItemOffsets(Rect outRect, View view,

RecyclerView parent, RecyclerView.State state) {

int position = parent.getChildAdapterPosition(view);

int spanCount = getSpanCount(parent);

int childCount = parent.getAdapter().getItemCount();

int pos = position;

if(hasHeader) {

if(position == 0) {

outRect.set(0, 0, 0, mDivider.getIntrinsicHeight());

return;

} else {

pos = position - 1;

}

}

if (isLastColum(parent, pos, spanCount, childCount)) {

outRect.set(0, 0, mDivider.getIntrinsicWidth(), mDivider.getIntrinsicHeight());

} else {

outRect.set(0, 0, mDivider.getIntrinsicWidth(),

mDivider.getIntrinsicHeight());

}

}

}

改造的地方是获取偏移量的方法我们换了一个，因为原来的那个已经过时了，而且，这里我们还加了一个boolean类型的hasHeader变量来表示是不是有header，如果hasHeader并且position为0，那么我们仅仅绘制底部的分割线，其他的地方不绘制，在有header的情况下，我们还需要将position减1，因为我们认为的第1个item其实是第2个。这个时候我们再来看看有分割线的效果。



看来我们的想法是对的，header部分除了底部有一个分割线外，并没有其他的分割线，这也完全符合我们的需求。

## 封装

这下好了，基本上完美的处理好了，可是难道我们对于不同的Adapter都需要写那么多代码吗？ 对于一个懒程序员来说，这肯定是一个可怕的事情，所以，我们还需要对我们的Adapter进行封装，目的就是可以轻轻松松的写代码，

/\*\*

\* Created by qibin on 2015/11/5.

\*/public abstract class BaseRecyclerAdapter<T> extends RecyclerView.Adapter<RecyclerView.ViewHolder> {

public static final int TYPE\_HEADER = 0;

public static final int TYPE\_NORMAL = 1;

private ArrayList<T> mDatas = new ArrayList<>();

private View mHeaderView;

private OnItemClickListener mListener;

public void setOnItemClickListener(OnItemClickListener li) {

mListener = li;

}

public void setHeaderView(View headerView) {

mHeaderView = headerView;

notifyItemInserted(0);

}

public View getHeaderView() {

return mHeaderView;

}

public void addDatas(ArrayList<T> datas) {

mDatas.addAll(datas);

notifyDataSetChanged();

}

@Override

public int getItemViewType(int position) {

if(mHeaderView == null) return TYPE\_NORMAL;

if(position == 0) return TYPE\_HEADER;

return TYPE\_NORMAL;

}

@Override

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

if(mHeaderView != null && viewType == TYPE\_HEADER) return new Holder(mHeaderView);

return onCreate(parent, viewType);

}

@Override

public void onBindViewHolder(RecyclerView.ViewHolder viewHolder, int position) {

if(getItemViewType(position) == TYPE\_HEADER) return;

final int pos = getRealPosition(viewHolder);

final T data = mDatas.get(pos);

onBind(viewHolder, pos, data);

if(mListener != null) {

viewHolder.itemView.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

mListener.onItemClick(pos, data);

}

});

}

}

@Override

public void onAttachedToRecyclerView(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) {

return getItemViewType(position) == TYPE\_HEADER

? gridManager.getSpanCount() : 1;

}

});

}

}

@Override

public void onViewAttachedToWindow(RecyclerView.ViewHolder holder) {

super.onViewAttachedToWindow(holder);

ViewGroup.LayoutParams lp = holder.itemView.getLayoutParams();

if(lp != null

&& lp instanceof StaggeredGridLayoutManager.LayoutParams) {

StaggeredGridLayoutManager.LayoutParams p = (StaggeredGridLayoutManager.LayoutParams) lp;

p.setFullSpan(holder.getLayoutPosition() == 0);

}

}

public int getRealPosition(RecyclerView.ViewHolder holder) {

int position = holder.getLayoutPosition();

return mHeaderView == null ? position : position - 1;

}

@Override

public int getItemCount() {

return mHeaderView == null ? mDatas.size() : mDatas.size() + 1;

}

public abstract RecyclerView.ViewHolder onCreate(ViewGroup parent, final int viewType);

public abstract void onBind(RecyclerView.ViewHolder viewHolder, int RealPosition, T data);

public class Holder extends RecyclerView.ViewHolder {

public Holder(View itemView) {

super(itemView);

}

}

public interface OnItemClickListener<T> {

void onItemClick(int position, T data);

}

}

我们将BaseRecyclerAdapter抽象起来，并且提供两个抽象方法onCreate和onBind用来创建holder和绑定数据，而对于header做的一系列工作，我们都放到了BaseRecyclerAdapter中，而继承BaseRecyclerAdapter后，我们仅仅关心我们的holder怎么创建和数据怎么绑定就ok。例如下面代码：

/\*\*

\* Created by qibin on 2015/11/7.

\*/public class MyAdapter extends BaseRecyclerAdapter<String> {

@Override

public RecyclerView.ViewHolder onCreate(ViewGroup parent, int viewType) {

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

return new MyHolder(layout);

}

@Override

public void onBind(RecyclerView.ViewHolder viewHolder, int RealPosition, String data) {

if(viewHolder instanceof MyHolder) {

((MyHolder) viewHolder).text.setText(data);

}

}

class MyHolder extends BaseRecyclerAdapter.Holder {

TextView text;

public MyHolder(View itemView) {

super(itemView);

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

}

}

}

这样我们再用起来就简单多了，对于这样的封装，我们还算满意，再做完添加header后，相信大家对于footer也有想法了，有想法就实现它吧，扩展一下BaseRecyclerAdapter就ok啦。