链滴

BaseRecyclerViewAdapterHelper 开源项目之 BaseSectionQuickAdapter 实现 Expandable And collapse 效果的源码学习

作者：angels

原文链接：https://ld246.com/article/1496286719632

来源网站：链滴

version:2.8.5

更多分享请看：http://cherylgood.cn

今天我们来学习BaseRecyclerViewAdapterHelper中有关实现可展开和折叠二级Item或多级Item的码。在开始学习之前，我想先分析下实现的思路，这样对于进行源码的理解效果比较好。

实现伸展and折叠，很多控件都有，网上也有用linearlayout实现的功能很强大、很炫酷的开源项目平时要实现一些伸缩性的自定义控件，我们也可以是用属性动画，或者动态控制控件的Layout属性等可以实现。那么现在我们来想象一下，如果在recyclerview中实现该功能，相对来说能想到的比较合的方式是什么呢?

其实我们可以很好的利用RecyclerView.Adapter给我们提供的如下一些通知数据源更新的方法来实现们的动态伸展and折叠功能。当要伸展时，我们动态将下一级item的数据添加在与adapter绑定的数集合中，然后通知layoutManger更新数据源。当要收缩时，同理，将下一级的item的数据源从与adater绑定的数据集合中移除，然后通知更新。

```
    * @see #notifyItemChanged(int)
    * @see #notifyItemInserted(int)
    * @see #notifyItemRemoved(int)
    * @see #notifyItemRangeChanged(int, int)
    * @see #notifyItemRangeInserted(int, int)
    * @see #notifyItemRangeRemoved(int, int)
```

思路：

1. 数据bean应该有存储自己数据的字段
2. 数据bean应该有存储下一级item列表的集合类型的字段
3. 数据bean应该有一个字段标识当前item的状态（伸展or收缩）
4. 初始化adapter时只渲染顶级的item
5. 点击item是检测该item是否支持伸缩
6. 支持伸缩：当前状态展开->折叠（将次级list插入adapter绑定的data集合中，刷新数据）；当前态折叠->展开(将次级的list从与adapter绑定的data集合中移除，刷新数据)
7. 插入或移除的位置根据点击的item确定，插入量与移除量根据下一级item数量确定
8. 插入移除过程中可以使用动画效果

思路理清之后我们接下来开始学习源代码：

实现Expandable And collapse 效果我们仍然是使用BaseMultiItemQuickAdapter实现即可

然后我们需要先看两个相关的类：IExpandable接口；AbstractExpandableItem： 对数据bean的再封装，某个bean如果有次级的list 可以实现该抽象类。

```
package com.chad.library.adapter.base.entity;

import java.util.List;

/**
 * implement the interface if the item is expandable
 * Created by luoxw on 2016/8/8.
 */
```

```
public interface IExpandable {
    boolean isExpanded();
    void setExpanded(boolean expanded);
    List getSubItems();

    /**
     * Get the level of this item. The level start from 0.
     * If you don't care about the level, just return a negative.
     */
    int getLevel();
}
```

可以看到，IExpandable 里面定义了四个接口方法：

1. isExpanded判断当前的bean是否已展开
2. setExoanded更新bean的当前状态
3. getSubItems返回下一级的数据集合
4. getLevel 返回当前item属于第几个层级， 第一级from 0

```
package com.chad.library.adapter.base.entity;

import java.util.ArrayList;
import java.util.List;

public abstract class AbstractExpandableItem implements IExpandable {
    protected boolean mExpandable = false;
    protected List mSubItems;

    @Override
    public boolean isExpanded() {
        return mExpandable;
    }

    @Override
    public void setExpanded(boolean expanded) {
        mExpandable = expanded;
    }

    @Override
    public List getSubItems() {
        return mSubItems;
    }

    public boolean hasSubItem() {
        return mSubItems != null && mSubItems.size() > 0;
    }

    public void setSubItems(List list) {
        mSubItems = list;
    }

    public T getSubItem(int position) {
```

```
        if (hasSubItem() && position < mSubItems.size()) {
            return mSubItems.get(position);
        } else {
            return null;
        }
    }

    public int getSubItemPosition(T subItem) {
        return mSubItems != null ? mSubItems.indexOf(subItem) : -1;
    }

    public void addSubItem(T subItem) {
        if (mSubItems == null) {
            mSubItems = new ArrayList<>();
        }
        mSubItems.add(subItem);
    }

    public void addSubItem(int position, T subItem) {
        if (mSubItems != null && position >= 0 && position < mSubItems.size()) {
            mSubItems.add(position, subItem);
        } else {
            addSubItem(subItem);
        }
    }

    public boolean contains(T subItem) {
        return mSubItems != null && mSubItems.contains(subItem);
    }

    public boolean removeSubItem(T subItem) {
        return mSubItems != null && mSubItems.remove(subItem);
    }

    public boolean removeSubItem(int position) {
        if (mSubItems != null && position >= 0 && position < mSubItems.size()) {
            mSubItems.remove(position);
            return true;
        }
        return false;
    }
}
```

字段方法解析：

1. mExpandable 保存当前的状态值，默认为false
2. mSubItems 存储数据bean集合

里面还包装了一些常用的方法，这里就不一一解析了。

接下来我们以一个使用demo的实现来进行分析：

我们可以看群主demo中的ExpandableUseActivity ：

```
 private ArrayList generateData() {
        int lv0Count = 9;
        int lv1Count = 3;
        int personCount = 5;

        String[] nameList = {"Bob", "Andy", "Lily", "Brown", "Bruce"};
        Random random = new Random();

        ArrayList res = new ArrayList<>();
        for (int i = 0; i < lv0Count; i++) {
            Level0Item lv0 = new Level0Item("This is " + i + "th item in Level 0", "subtitle of " + i);
            for (int j = 0; j < lv1Count; j++) {
                Level1Item lv1 = new Level1Item("Level 1 item: " + j, "(no animation)");
                for (int k = 0; k < personCount; k++) {
                    lv1.addSubItem(new Person(nameList[k], random.nextInt(40)));
                }
                lv0.addSubItem(lv1);
            }
            res.add(lv0);
        }
        return res;
    }
```

这段代码的作用是生成一个支持Expandable and collapse 的数据集合，创建一个0级的Level0Item 然后将下一级的Level1Item添加到Level0Item中。

```
public class Level0Item extends AbstractExpandableItem implements MultiItemEntity {
    public String title;
    public String subTitle;

    public Level0Item( String title, String subTitle) {
        this.subTitle = subTitle;
        this.title = title;
    }

    @Override
    public int getItemType() {
        return ExpandableItemAdapter.TYPE_LEVEL_0;
    }

    @Override
    public int getLevel() {
        return 0;
    }
}
```

可以看到Level0Item继承了AbstractExpandableItem 并实现MultiItemEntity接口。里面根据实际求定义相应的字段即可。

Level1Item 与Level0Item一样，只是返回的Level =1：

```
public class Level1Item extends AbstractExpandableItem implements MultiItemEntity{
    public String title;
    public String subTitle;
```

```
    public Level1Item(String title, String subTitle) {
        this.subTitle = subTitle;
        this.title = title;
    }

    @Override
    public int getItemType() {
        return ExpandableItemAdapter.TYPE_LEVEL_1;
    }

    @Override
    public int getLevel() {
        return 1;
    }
}
```

当如过某一级的item没有下一级的list时，就不需要在实现AbstractExpandableItem了

然后我们的切入点时adapter，因为默认是折叠状态，当我们点击具备展开折叠能力的item时才会触该功能，所以逻辑的控制是在adapter中的。

```
package com.chad.baserecyclerviewadapterhelper.adapter;

import android.util.Log;
import android.view.View;
import android.view.ViewGroup;

import com.chad.baserecyclerviewadapterhelper.R;
import com.chad.baserecyclerviewadapterhelper.entity.Level0Item;
import com.chad.baserecyclerviewadapterhelper.entity.Level1Item;
import com.chad.baserecyclerviewadapterhelper.entity.Person;
import com.chad.library.adapter.base.BaseMultiItemQuickAdapter;
import com.chad.library.adapter.base.BaseViewHolder;
import com.chad.library.adapter.base.entity.MultiItemEntity;

import java.util.List;

/**
 * Created by luoxw on 2016/8/9.
 */
public class ExpandableItemAdapter extends BaseMultiItemQuickAdapter<MultiItemEntity, B
seViewHolder> {
    private static final String TAG = ExpandableItemAdapter.class.getSimpleName();

    public static final int TYPE_LEVEL_0 = 0;
    public static final int TYPE_LEVEL_1 = 1;
    public static final int TYPE_PERSON = 2;

    /**
     * Same as QuickAdapter#QuickAdapter(Context,int) but with
     * some initialization data.
     *
     * @param data A new list is created out of this one to avoid mutable list
```

```
     */
    public ExpandableItemAdapter(List data) {
        super(data);
        addItemType(TYPE_LEVEL_0, R.layout.item_expandable_lv0);
        addItemType(TYPE_LEVEL_1, R.layout.item_expandable_lv1);
        addItemType(TYPE_PERSON, R.layout.item_expandable_lv2);
    }

    @Override
    protected void convert(final BaseViewHolder holder, final MultiItemEntity item) {
        switch (holder.getItemViewType()) {
            case TYPE_LEVEL_0:
                switch (holder.getLayoutPosition() %
                        3) {
                    case 0:
                        holder.setImageResource(R.id.iv_head, R.mipmap.head_img0);
                        break;
                    case 1:
                        holder.setImageResource(R.id.iv_head, R.mipmap.head_img1);
                        break;
                    case 2:
                        holder.setImageResource(R.id.iv_head, R.mipmap.head_img2);
                        break;
                }
                final Level0Item lv0 = (Level0Item)item;
                holder.setText(R.id.title, lv0.title)
                        .setText(R.id.sub_title, lv0.subTitle)
                        .setImageResource(R.id.iv, lv0.isExpanded() ? R.mipmap.arrow_b : R.mipmap.ar
ow_r);
                holder.itemView.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        int pos = holder.getAdapterPosition();
                        Log.d(TAG, "Level 0 item pos: " + pos);
                        if (lv0.isExpanded()) {
                            collapse(pos);
                        } else {
//                            if (pos % 3 == 0) {
//                                expandAll(pos, false);
//                            } else {
                             expand(pos);
//                            }
                        }
                    }
                });
                break;
            case TYPE_LEVEL_1:
                final Level1Item lv1 = (Level1Item)item;
                holder.setText(R.id.title, lv1.title)
                        .setText(R.id.sub_title, lv1.subTitle)
                        .setImageResource(R.id.iv, lv1.isExpanded() ? R.mipmap.arrow_b : R.mipmap.ar
ow_r);
                holder.itemView.setOnClickListener(new View.OnClickListener() {
                    @Override
```

```
                public void onClick(View v) {
                    int pos = holder.getAdapterPosition();
                    Log.d(TAG, "Level 1 item pos: " + pos);
                    if (lv1.isExpanded()) {
                        collapse(pos, false);
                    } else {
                        expand(pos, false);
                    }
                }
            });
            break;
        case TYPE_PERSON:
            final Person person = (Person)item;
            holder.setText(R.id.tv, person.name + " parent pos: " + getParentPosition(person));
            break;
    }
  }
}
```

可以看到里面我们先添加3个level的布局资源文件。重点在convert回调方法；

1. 最外层进行viewholder 的类型判断进行数据绑定

2. 添加点击事件的监听

3. 当被点击时，判断当前的levelitem是不是展开的或折叠的，然后根据你的需要调用collapse或者exand进行折叠或展开操作。

重点来的，最终实现展开、折叠功能其实是依赖collapse和expand这些api;那我们来看下这些api到内部是怎么实现的，我们从expand开始。代码中expand(pos);传了一个pos进来，而这个pos就是被击的item在adapter数据集合中的index。

```
    /**
     * Expand an expandable item
     *
     * @param position     position of the item
     * @param animate      expand items with animation
     * @param shouldNotify notify the RecyclerView to rebind items, false if you want to do it
     *                     yourself.
     * @return the number of items that have been added.
     */
    public int expand(@IntRange(from = 0) int position, boolean animate, boolean shouldNotif
) {
        position -= getHeaderLayoutCount();

        IExpandable expandable = getExpandableItem(position);
        if (expandable == null) {
            return 0;
        }
        if (!hasSubItems(expandable)) {
            expandable.setExpanded(false);
            return 0;
        }
        int subItemCount = 0;
```

```
        if (!expandable.isExpanded()) {
            List list = expandable.getSubItems();
            mData.addAll(position + 1, list);
            subItemCount += recursiveExpand(position + 1, list);

            expandable.setExpanded(true);
            subItemCount += list.size();
        }
        int parentPos = position + getHeaderLayoutCount();
        if (shouldNotify) {
            if (animate) {
                notifyItemChanged(parentPos);
                notifyItemRangeInserted(parentPos + 1, subItemCount);
            } else {
                notifyDataSetChanged();
            }
        }
        return subItemCount;
    }

    /**
     * Expand an expandable item
     *
     * @param position position of the item, which includes the header layout count.
     * @param animate  expand items with animation
     * @return the number of items that have been added.
     */
    public int expand(@IntRange(from = 0) int position, boolean animate) {
        return expand(position, animate, true);
    }

    /**
     * Expand an expandable item with animation.
     *
     * @param position position of the item, which includes the header layout count.
     * @return the number of items that have been added.
     */
    public int expand(@IntRange(from = 0) int position) {
        return expand(position, true, true);
    }
```

可以看到expand是一个方法多态，提供了三种参数类型的调用。支持是否需要动画，是否更新数据。

排除headerview的干扰，获得实际的位置position

```
position -= getHeaderLayoutCount();
```

判断其是否支持展开折叠，是否有下一级items需要展开，没有就直接返回0

```
IExpandable expandable = getExpandableItem(position);
        if (expandable == null) {
            return 0;
        }
        if (!hasSubItems(expandable)) {
```

```
            expandable.setExpanded(false);
            return 0;
        }
```

下面代码作用：如果处于折叠状态且需要展开，则执行到下面代码，通过getSubItems获得要展开的list，将其添加到mdata中，通过recursiveExpand获得要展开的items的数量

```
int subItemCount = 0;
        if (!expandable.isExpanded()) {
            List list = expandable.getSubItems();
            mData.addAll(position + 1, list);
            subItemCount += recursiveExpand(position + 1, list);

            expandable.setExpanded(true);
            subItemCount += list.size();
        }
```

我们可以看到recursiveExpand的源码如下：下面是一个递归调用，一直遍历到最后一层不支持展开叠的item才会回溯回来，遍历过程中可以看到一个判断，if(item.isExpanded) 就是如果下一级的item原来已经是处于展开状态的，此时我们也需要展开他。最终返回的是所需展开的items的数量。

```
    private int recursiveExpand(int position, @NonNull List list) {
        int count = 0;
        int pos = position + list.size() - 1;
        for (int i = list.size() - 1; i >= 0; i--, pos--) {
            if (list.get(i) instanceof IExpandable) {
                IExpandable item = (IExpandable) list.get(i);
                if (item.isExpanded() && hasSubItems(item)) {
                    List subList = item.getSubItems();
                    mData.addAll(pos + 1, subList);
                    int subItemCount = recursiveExpand(pos + 1, subList);
                    count += subItemCount;
                }
            }
        }
        return count;

    }
```

获得需要展开的items的数量值，也将数据集合添加到了mData中，此时我们通知layoutManager刷数据即可

```
int parentPos = position + getHeaderLayoutCount();
        if (shouldNotify) {
            if (animate) {
                notifyItemChanged(parentPos);
                notifyItemRangeInserted(parentPos + 1, subItemCount);
            } else {
                notifyDataSetChanged();
            }
        }
```

刷新的时候我们要先确定开始刷新位置，所以需要加上headerview的数量

然后调用如上代码即可。折叠是反向进行的，根据这个思路看就可以了。

总结：折叠->展开：mData添加需展开的数据集，更新数据源；展开->折叠：mData移除需折叠的据集，更新数据源。

后面会继续分析其他功能的实现源码，欢迎一起学习！