

1、Adapter 代码如下:

```
public class NodeAdapter extends BaseNodeAdapter {
   public NodeSectionAdapter() {
       super();
       // 注册Provider,总共有如下三种方式
       // 需要占满一行的,使用此方法(例如section)
       addFullSpanNodeProvider(new RootNodeProvider());
       // 普通的item provider
       addNodeProvider(new SecondNodeProvider());
       // 脚布局的 provider
       addFooterNodeProvider(new RootFooterNodeProvider());
   }
   /**
    * 自行根据数据、位置等信息,返回 item 类型
    * /
   @Override
   protected int getItemType(@NotNull List<? extends BaseNode> data, int position) {
       BaseNode node = data.get(position);
       if (node instanceof RootNode) {
           return 0;
       } else if (node instanceof ItemNode) {
           return 1;
       } else if (node instanceof RootFooterNode) {
           return 2;
       return -1;
```

2、 Provider 写法和 BaseProviderMultiAdapter 中的相同,只是继承类改为 BaseNodeProvider ,其他没有区别:

```
public class RootNodeProvider extends BaseNodeProvider {
    @Override
    public int getItemViewType() {
        return 0;
    @Override
    public int getLayoutId() {
        return R.layout.def_section_head;
    @Override
    public void convert(@NotNull BaseViewHolder helper, @NotNull BaseNode data) {
       // 数据类型需要自己强转
        RootNode entity = (RootNode) data;
        helper.setText(R.id.header, entity.getTitle());
    }
    @Override
    public void onClick(@NotNull BaseViewHolder helper, @NotNull View view, BaseNode data, int position) {
        getAdapter().expandOrCollapse(position);
```

3、设置数据

注意,数据需要遵循 树形结构 ,简单来说就是:套娃形式。

下面给出一组示例:

先定义几个节点类,继承于 BaseNode ,用于封装数据:

```
/**
 * 第一个节点FirstNode, 里面放子节点SecondNode
public class FirstNode extends BaseNode {
   private List<BaseNode> childNode;
    private String title;
    public FirstNode(List<BaseNode> childNode, String title) {
        this.childNode = childNode;
       this.title = title;
    public String getTitle() {
        return title;
    * 重写此方法,返回子节点
     * /
   @Nullable
   @Override
    public List<BaseNode> getChildNode() {
        return childNode;
```

```
/**
 * 第二个节点SecondNode,里面没有子节点了
public class SecondNode extends BaseNode {
   private String title;
   public SecondNode(String title) {
       this.title = title;
   public String getTitle() {
       return title;
    /**
    * 重写此方法,返回子节点
    * /
   @Nullable
   @Override
   public List<BaseNode> getChildNode() {
       return null;
```

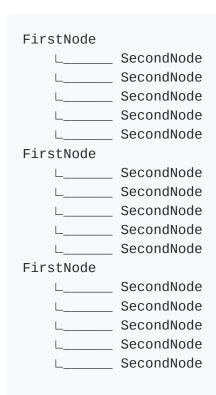
模拟生成数据:

```
private List<BaseNode> getEntity() {
    //总的 list, 里面放的是 FirstNode
    List<BaseNode> list = new ArrayList<>();
    for (int i = 0; i < 8; i++) {</pre>
```

```
//SecondNode #9 list
List<BaseNode> secondNodeList = new ArrayList<>();
for (int n = 0; n <= 5; n++) {
    SecondNode seNode = new SecondNode("Second Node " + n);
    secondNodeList.add(seNode);
}

FirstNode entity = new FirstNode(secondNodeList, "First Node " + i);
    list.add(entity);
}
return list;
}</pre>
```

它的结构如下:



.....

4、脚部Node

如果此 Node 需要脚部,那么此 Node 实现 NodeFooterImp 接口。

每一个 node 都可以有自己的脚部

示例如下:

5、折叠展开

如果此 Node 需要折叠展开功能,请继承 BaseExpandNode 。

示例如下:

Node 默认折叠\展开状态

```
public class RootNode extends BaseExpandNode {
    public RootNode() {
        // 默认不展开
        setExpanded(false);
    }
}
```

```
RootNode entity = new RootNode();
// 也可以在生成 Node 后设置。
// 注意:必须在设置给 Adapter 之前修改。数据设置给Adapter后,不应该再修改
entity.setExpanded(false);
```

折叠\展开Node

```
if (node.isExpanded()) {
    // 折叠某一个位置的Node
    getAdapter().collapse(position);
} else {
    // 展开某一位置的Node
    getAdapter().expand(position);
}
```

```
// 自动展开\折叠某一位置的Node
getAdapter().expandOrCollapse(position);

// 展开某一位置的Node, 并且其子节点的也展开
getAdapter().expandAndChild(position);

// 折叠某一位置的Node, 并且其子节点的也折叠
getAdapter().collapseAndChild(position);
```

查找父节点

```
// 查找某位置Node的 父节点
getAdapter().findParentNode(position)

// 查找此Node的 父节点
getAdapter().findParentNode(node)
```

指定的父Node添加\删除\替换子node

```
// 父 node 下添加子 node
adapter.nodeAddData(parentNode, data)
adapter.nodeAddData(parentNode, childIndex: Int, data)
adapter.nodeAddData(parentNode, childIndex: Int, dataList)

// 删除父 node 下的子 node
adapter.nodeRemoveData(parentNode, childIndex)
adapter.nodeRemoveData(parentNode, childNode)

// 改变指定的父node下的子node数据
```

```
adapter.nodeSetData(parentNode, childIndex: Int, data)

// 替换父node下的,全部子node
adapter.nodeReplaceChildData(parentNode, newDataList)
```

© 2020 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Training Blog About