中山大学移动信息工程学院本科生实验报告

年级	1501	专业（方向）	移动（互联网）
学号	15352015	姓名	曹广杰
电话	13727022190	Email	1553118845@qq.com

Content

一、实验题目

Intent、Bundle的使用以及RecyclerView、ListView的应用

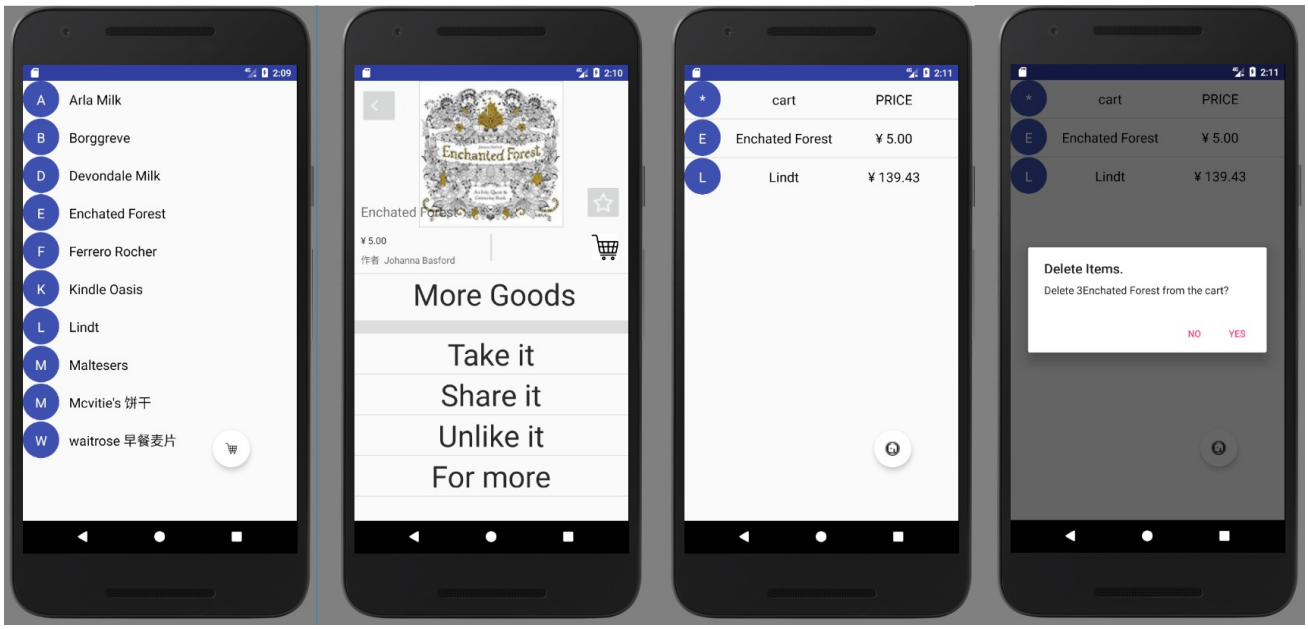
二、实现内容

本次实验模拟实现一个商品列表。

- 有两个界面，一个界面用于展示商品
- 一个界面用于查看购物车

三、课堂实验结果

实验截图



实验步骤以及关键代码

构建商品列表的布局

构建商品列表的布局的时候需要两个控件，RecyclerView和ListView，这两个控件放在同一个布局文件之下，随着布局文件中的按钮的触发实现两个界面的切换。

首先需要在**布局文件**中（本次实验的主战场是activitymain.xml文件）设置添加RecyclerView，继而设置二者的格式。二者在设计的时候都需要为之赋予一个id，以便之后的调用与检查。布局文件中的信息设置完成之后，则需要在MainActivity.Java文件中调用相关的函数以实现对应的功能。

设置**RecyclerView**的方式

由函数findViewById得到我们设置的RecyclerView的id在当前文件内的索引，以后我们对该id进行的操作，都将会相对应地作用在RecyclerView中。

RecyclerView在本次实验中的实现是纵向的滚动条，因此我们使用setLayoutManager函数，将该控件的格式设置为Linear：

```
recyclerView.setLayoutManager(new LinearLayoutManager(this));
```

至此，RecyclerView的布局就已经告一段落了，RecyclerView的信息已经在activitymain.xml的design布局中显现出来，虽然没有什么内部填充。

为RecyclerView传入数据

之前我们实现了对RecyclerView的布局设置，但是这个布局是空的，也就是尽管我们已经获得了RecyclerView的纵向表格，都是没有文字在内部或者没有内容填充的情况——因而我们需要传入数据。这主要依赖RecyclerView的函数`setAdapter`，但是setAdapter的参数（即适配器）需要进行一定的控制，并不是所有的适配器都能与之契合的。该适配器需要与该函数参数为同一类同时由于传入数据的行为需要很多信息，从图像到按钮，这种情况下笔者需要使用自定义的适配器为当前的RecyclerView传入信息。

自定义的适配器

之前说过，对于适配器有两个铁打的要求：

1. 可以容纳复杂的数据
2. 可以作为setAdapter函数的参数

为了容纳大量的数据我们实现自定义的适配器。而为了符合setAdapter的参数要求，我们需要在自定义的时候，使之继承于一个已有的类：

```
public abstract class CommonAdapter<T> extends RecyclerView.Adapter<ViewHolder>{}
```

该适配器主要针对几类对象：

1. `Context`该成员变量在后续的自定义实现中不能起到多大的作用——我们很少去使用它，但是对于系统而言，这是系统理解一切函数的前提与标准。
2. `int layoutid`该参数将会用于ViewHolder的get函数（之后再说）
3. `List<T>`该参数是传入的参数，是Template的集合，正是这样的数据结构允许我们传入复杂的多样的信息

为了几个成员变量的交替与计算，我们需要添加几个函数，以此实现对点击屏幕的监听操作。

- 初始化，即传入参数的接口，传入的信息作为参数被当前的适配器所得到，用于之后的函数调用：

```
public CommonAdapter(Context context, List<T> datas, int layoutid){}
```

- convert。将传入的参数转换为布局文件中的控件信息：

```
public abstract void convert(ViewHolder holder, T position);
```

该函数在适配器声明中没有具体实现，是因为在适配器中还不知道将会面对什么样子的布局文件，同时也不知道布局文件中有哪些控件，故而不能实现转换。但是在此实现了抽象的函数声明，这就使得在后续重新实现这个函数的时候，该类的实例化依然可以调用这个函数。

- OnItemClick 设置控件触发的函数。每一个布局之中都会有多个控件，而不同的控件的触发又会有不同的效应——由于这是我们自己定义的适配器，所以常规的OnClick监听是不能直接调用的，除非重定义一次——这正是这个函数所做的工作。当然，除了这个函数，还有与之匹配的set函数负责调用这个函数。

```
public interface OnItemClickListener{}
public void setOnItemClickListener(OnItemClickListener onItemClickListener)
```

- removeItem 移除函数。之前我们传入的数据中有一个Template的List，该数据结构是一个集合，之后我们的设计中会遇到对于集合中的元素进行增删的操作，这时候就需要这个函数。

```
public void setOnItemClickListener(OnItemClickListener onItemClickListener)
```

除了以上这些之外，还有几个override的函数，即基类中出现过但是又没有实现的函数，一定要我们自己去重新实现一遍，尽管我们不会调用这些函数，但是对于系统来说，这是至关重要的，这几个函数如下：

```
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType);
public int getItemCount();
public void onBindViewHolder(final ViewHolder holder, int position);
```

使用自定义适配器填充数据

之前我们已经完成了对自定义适配器的实现，接下来就是使用了。所谓传入数据实际上就是传入参数，参数分析如下：

```
CommonAdapter<T>(this, data, R.layout.layoutfile);
```

1. 第一个参数事实上就是context参数，这种情况大多数条件下我们都不需要纠结，传入this即可；
2. `<T>` T代表一种映射，以 `Map<String, Object>` 为例，表示将当前的控件或者无论是什么东西，都让其对应一个字符串，之后想要调用的时候，只要将 `Map<String, Object>` 的实例化值调用get()函数，参数为之前的字符串，函数便会返回对应的Object值；
3. data的数据类型为 `List<Map<String, Object>>` ，真正的集合，我们所做的就是把各种数据添加打包为这个大的集合中，传入函数而已；
4. layoutfile 布局文件，之后我们需要为了界面的切换实现很多布局文件。这里的layoutfile就是我们需要的布局文件的文件名，如果传入的参数与文件名毫无关联，可能会出现闪退等错误。

自定义适配器**convert**转化数据

在传入数据之后，函数并不能直接理解我们传入的参数是什么，也不知道如何处理它们，这就需要我们将传入的形形色色的参数与布局文件中的控件相对应，控件才能获得对应的数据信息。

举例如下：

```
List<Map<String, Object>> listItems = new ArrayList<>();
final CommonAdapter adapter = new CommonAdapter<Map<String, Object>>
        (this, listItems, R.layout.goods_list_item) {
    @Override
    public void convert(ViewHolder holder, Map<String, Object> s) {
        TextView name = holder.getView(R.id.name);
        name.setText(s.get("对应的字符串").toString());
    }
};
```

listItem就是我们所说的数据包，其填充过程没有写，这个数据包在传入的时候使用初始化函数传入。内部就可以直接使用convert函数对数据包进行解析操作：

convert的第二个参数s是对应于listItem数据包的一个单元，我们将会使用ViewHolder的getView函数得到需要控件的映射，对应的实例化再调动set函数就可以修改控件内部的信息了。我们看到convert函数是在初始化函数的内部嵌套写入的，因为convert函数常规情况下只能对一个单元进行处理，这种嵌套结构是java在设计中的故意，它可以帮你省去循环遍历的步骤，直接对整个数据集进行处理。

ViewHolder的分析

之前我们说过ViewHolder的使用，可以根据Resource中的id信息直接获得对应的类的实例化（该实例化可以调用id对应的控件的所有函数）。

ViewHolder，顾名思义就是一个View 的holder。该模块用于储存所有的View，即控件。成员变量有SparseArray对于View的集合，以及View。View就是根据id返回的实体化信息，而get是这种信息的集合。而之前我们使用的通过id值返回View的操作，正是通过以下函数实现的：

```
public <T extends View> T getView(int viewid){
    View view = mViews.get(viewid);
    if(view == null){
        view = mConvertView.findViewById(viewid);
        mViews.put(viewid, view);
    }
    return (T)view;
}
```

我们通过Inflation对id进行扩展，将得到的ViewHolder作为返回值，从函数get中返回出来。这时候的数据类型还是ViewHolder，所以对View类型的变量mViews重新赋值，这时候对其强制转化的结果就是我们需要的View值。

这样，RecyclerView的函数就设置完成了，按照模板向函数里传递参数就可以在模拟机上运行处出想要的结果了。

设置ListView的信息

该界面的两个布局分别为RecyclerView布局和ListView布局，RecyclerView布局用于表示所有的商品信息，该布局会将所有的商品都罗列出来，供使用者进行操作，而该部分我们已经把框架搭构结束了。这里我们使用的是ListView，ListView表示是购物车里的商品种类。

所谓购物车布局表示，在商品界面我们可以通过单击商品条框进入商品的详细信息部分，并通过购物车按钮将商品信息添加到购物车内，这个布局表示的正是购物车内部的商品条目。

获得购物车列表信息

之前有过根据id获得控件的函数，但是那是我们自己实现的，当然，系统也有这样的函数，那就是：

> (控件数据结构)findViewById(R.id.控件id);

该函数可以根据传入的控件id返回相应的控件信息。现在我们为了修改ListView的内容，为其添加数据，我们需要获得ListView的控件信息，就正好使用这个函数。

通过适配器为购物车传入数据

由上文我们知道，获得了购物车的控件之后，由于购物车本身就是一个View，这和RecyclerView一样，我们可以为之传递一个包含信息的适配器adapter：

```
ListView cart_list = (ListView)findViewById(R.id.cart_list);

cart_list.setAdapter(simpleAdapter);
```

使用**simpleAdapter**传入数据

比自定义的Adapter要简单得多，simpleAdapter不需要自己另写文件，只要调用simpleAdapter的初始化函数即可，初始化函数只有一种格式，这显然没有自定义的函数多样化，但是方便很多：

```
SimpleAdapter simpleAdapter
             = new SimpleAdapter(this, 传入的List数据包, R.layout.布局文件名,
            new String[]{"布局文件控件1绑定的字符串", "布局文件控件2绑定的字符串",
                         "布局文件控件3绑定的字符串"},
            new int[] {R.id.布局文件控件1, R.id.布局文件控件2, R.id.布局文件控件3});
```

这样，在ListView的布局中就有需要表示的信息了。

注：*SimpleAdapter使用的数据包与自定义的adapter使用的数据包一样，都是List，这一点可以直接借鉴RecyclerView的使用方法*

在布局中添加悬浮按钮

在商品表示界面的布局信息完成之后，就需要一个触发点以实现两个界面之间的切换。这里的触发点正是悬浮按钮——FloatingActionButton：

悬浮按钮的控件设置

FloatingActionButton虽然与一般的按钮不太一样，可是毕竟也是一个控件，既然是控件就要遵循几个使用步骤：

1. 在布局文件.xml中添加该控件

   方法有二：

   - 可以直接从design窗口把找到的FloatingActionButton拖进布局文件里
   - 在布局文件里用代码表示

2. 在调用函数.java中调取该控件

   - 获取该控件：获取控件的方法与RecyclerView和ListView一样，可以使用findViewById函数
   - 在id之前一定要加上一些路径，比如：R.id.控件id，表示res/id查询/控件id

3. 使用该控件调取其他的函数

   在本界面中需要使用这一个按钮实现“**商品列表**”与“**购物车内商品列表**”的切换，那么很显然就需要一个Click监听，即如果点击就会切换界面。

   设置OnClick监听：

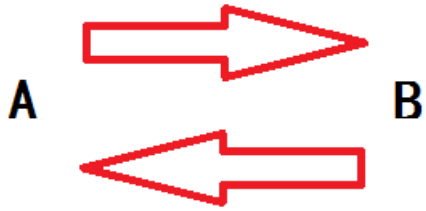
   ```
   悬浮按钮.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {}
   ```

   三层触发：setOnClickListener => OnClickListener => onClick;

触发函数中的界面切换

在本次实验中使用的悬浮按钮触发会使得界面从一个布局转到另一个布局，这里的实现方案是：

- 在条件语句中执行：如果按钮被触发，而按钮处于A状态，则切换到B状态：如果是B状态则切换回A状态：



- 切换操作是——当前界面不可见，另一个界面可见：

```
当前布局.setVisibility(View.INVISIBLE);
隐藏布局.setVisibility(View.VISIBLE);
```

这样就可以实现界面的切换了。实际效果是每点击一次悬浮按钮，界面会从当前切换到另一个。

至此，在整体结构上的布局就已经结束了。按照期望，我们是可以看到：

- 初始界面是商品的条目信息，包含首字母与商品名称，右下角有悬浮按钮图为：

- 点击悬浮按钮，界面切换到购物车模式，内部显示购物车中所有的商品信息，悬浮按钮图片换为：

- 点击悬浮按钮，界面回到商品列表。

商品列表的点击与长按（适配器点击监听）

在实现了商品列表之后，商品列表其本身还应该是具有按钮功能的控件。而很显然，RecyclerView中并不具备调用OnClick监听的条件——调用OnClick监听需要是单独的Button，而RecyclerView是一个表格，它的调用会使得Click触发的前提不明确，因此，我们可以在适配器中调用，适配器在传入数据的时候，更多的是处理RecyclerView的每一个条目的信息，这正符合调用Click监听的条件。

巧在之前我们恰好设定了ItemClick的监听，这里正好用上：

```
适配器.setOnItemClickListener(new CommonAdapter.OnItemClickListener(){
    @Override
    public void onLongClick(int position){}
```

setOnClickListener函数是自己写的函数，该函数出现在自定义适配器的实现文件中，其内部可以根据对ItemClick监听（该监听是继承于系统原有的监听函数）内部的Click函数的检查，实现对长按功能和点击功能的设计。

实验遇到困难以及解决思路

RecyclerView的布局设置与参数显示

RecyclerView的布局设置之后，传入的参数不能够在activitymain.xml的布局文件中展现出来——正常情况下，就是不会展现出来的——在activitymain中的design模块下，包含太多的信息并不利于设置布局格式，对于RecyclerView的布局传入的数据只有在虚拟机上运行的时候才可以展现出来。

为布局文件传入的参数控件绑定字符串

```
    Map<String, Object> tmp = new LinkedHashMap<>();
    tmp.put("name", Name[i]);
    tmp.put("first_letter", Letter[i]);
    listItems.add(tmp);
    /*****************************/
    TextView name = holder.getView(R.id.name);
    name.setText(s.get("name").toString());
```

在Map的使用中，需要向Map中添加各种各样的信息，这里使用的就是put函数，这个函数之前笔者总是在这个问题上出差错：

- 如果在get的时候使用的字符串未曾出现过，那么就会导致免疫数据的输出；
- 如果get传入的参数不是我们指定的Object对应的字符串，就会有错误的输出；

原因：put函数的使用不当

- put函数的调用对象：`Map<String, Object>`
- 格式：放映射的map容器.put("a对应的字符串_随便写一个就行"， 对象a);

Map的存在使得不容易表示的、不一定是什么格式的Object有办法表示，都表示成一个使用者命名的字符串——能表示也就能操作，这为数据的处理提供了更多的可能。

悬浮按钮的图片变化

在笔者设置悬浮按钮的时候，期望其在点触之后，图片有所改变。按照笔者的习惯，笔者将会在布局文件中使用`background()`传入所需要的布局文件，然后依据push_state来切换相应的图片。这种方法虽然常规但是很麻烦，而且在笔者的使用过程中出现了奇怪的错误。

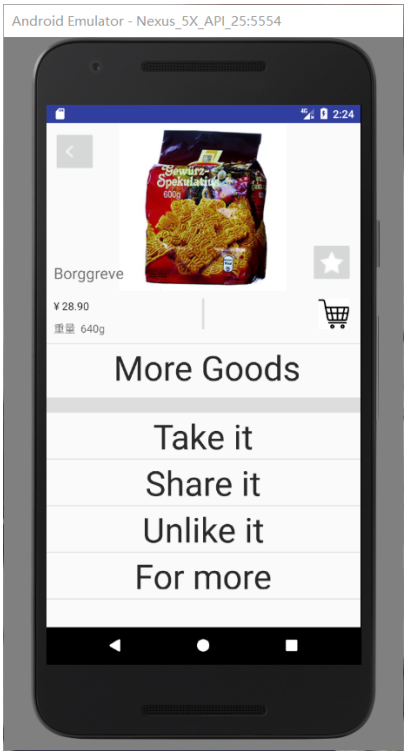
好在笔者并没有纠结这种错误，而是使用了一种新的方法：

```
悬浮按钮.setImageDrawable(getResources().getDrawable(R.mipmap.图片名字));
```

一行解决问题——把这个条件放到判断语句中即可对悬浮按钮的图片进行修改了。

四、课后实验结果

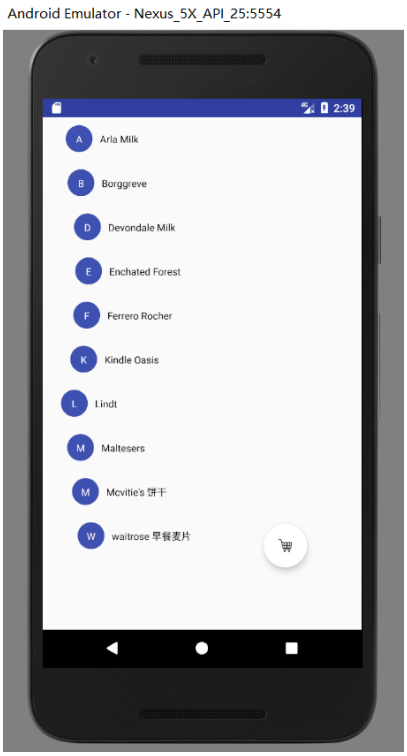
对于星星图标信息的同步化



按照实验要求，星星图标是可以根据点触而在空心星星图标与实心星星图标之间变化的，但是修改之后的状态并不能在商品界面与购物车界面实现同步，每次进入的时候都是空的，没有起到应有的标志作用——因为在商品界面的顺序与购物车中的顺序不同，同一件商品可能根据我们收藏时间的不同而处在购物车中的任意位置，这种情况下，使用全局变量也不能实现商品信息的同步。笔者在后续的实验中使用了数组的映射实现了序号的对应，最终实现了同步操作。

动画的设计

笔者在原有的基础上添加了一点动画的设计：



其实无非就是修改一下传入RecyclerView的适配器的数据。而同样的操作，笔者也将其作用在了ListView上，这样使得切换的界面不会太过突兀，有些随性自然。

五、实验思考及感想

本次实验的主要任务是完成商品列表清单以及相关功能的设计。目前的完成情况来看，所有的使用布局问题，最后都能转化为控件问题，无论是RecyclerView还是ListView，无论是ImageView还是FloatingActionButton，无论是ViewHolder还是CommonAdapter通通可以根据其id获得对应的控件信息，以实现对应的控件数据的导入。而导入过程也非常有趣，尽管以上面对的数据结构千差万别，或者是系统本身的，或者是自己定义的，但是它们在传入数据的时候，对于数据的要求都非常低——都仅仅要求一个`List<Map<String, Object>>`这就相当于在日常操作系统中尽管有很多的数据格式：txt、png、doc或者pdf，但是这些信息最后都可以打包提交到ftp上。至于后续的解码操作，有很大一部分并不需要我们去操作。

图片系列文件的获取问题，因为对图片的操作大多类似，这种情况下，如何能够为图片的系列使用同一个处理方法就是节省代码的关键。本次实验中笔者使用的是TypedArray的格式，由该数据结构从Resource中读取已经存档的资源文件——这些文件于是才获得了各自的序号。这正是笔者后来发现的问题，因为笔者在后续过程中希望使用int的集合容器，但是发现并不可用，vector和Array都不支持基本数据格式的使用。万般无奈笔者才使用的映射数组来实现int到int的切换。

从博客到代码。如果在了解使用RecyclerView的过程中不能理解到这些量的含义怎么办，不能理解内部的含义，很多时候，这些教程就不能起到实际的作用。阅读教程以精为贵，更重要的是从已有的代码中获取对于整体的认知，尽管这种认知可能是不全面的，也可能是有偏颇的，但是随着理解结构的拓展，对设计者的揣测就会更加贴近。有一个不得不说的注意事项，在尝试进行代码的实现之前，一定要设置好一切依赖，避免可能会出现的任何不必要的麻烦。事实上，从网络上获取的代码也并不急于尝试运行，更多的是了解代码中蕴藏的结构层次，至于关键的函数名称以及使用范围，反而是非常容易获得的信息了。除此之外，独自前行的后果一定要估计清楚，有时候同伴的帮助可以简化很多不必要的麻烦，而这种来自同伴帮助的契机，正是时间，在恰当的时间做恰当的事情，事半功倍是理所当然。