# 中山大学移动信息工程学院本科生实验报告

## （2017 年秋季学期）

课程名称：移动应用开发　　任课教师：郑贵锋

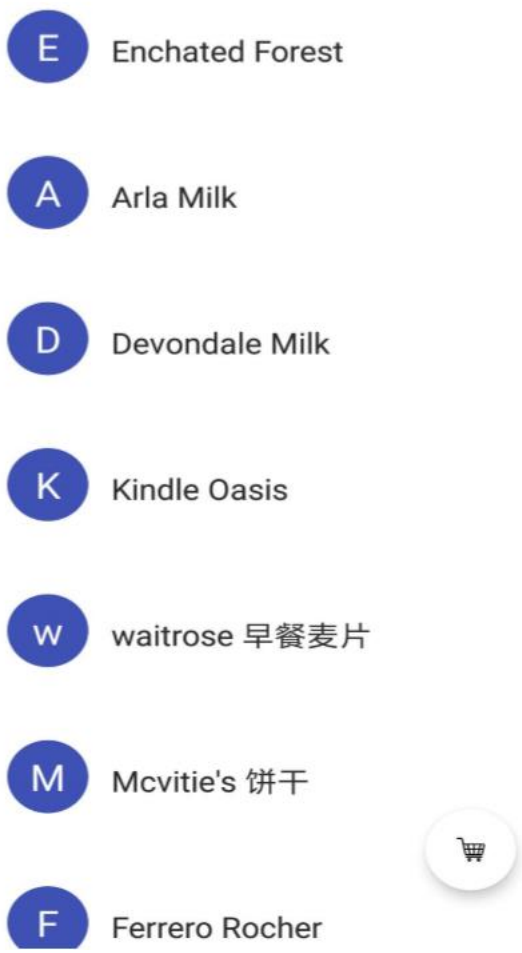
| 年级 | 15 级 | 专业（方向） | 软件工程(移动) |
|---|---|---|---|
| 学号 | 15352133 | 姓名 | 黄少豪 |
| 电话 | 13727024545 | Email | 328730316@qq.com |
| 开始日期 | 2017.10.22 | 完成日期 | 2017.10.25 |

## 一、 实验题目

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

## 二、 实现内容

1. 模拟一个商品表，有两个界面，第一个界面用于呈现商品，如下所示:



2. 点击右下方的悬浮按钮可以切换到购物车:


3. 上面两个列表点击任意一项后，可以看到详细的信息:


4. 使用 RecyclerView 实现商品列表。点击商品列表中的某一项商品会跳转到该商品详情界面，长按第 i 项商品会删除该商品，并且弹出 Toast 提示“移除第 i 个商品”。

5. 点击右下方的 FloatingActionButton，从商品列表切换到购物车或者从购物车切换到商品列表，并且 FloatingActionButton 的图片要做相应改变。

6. 使用 ListView 实现购物车。点击购物车的某一项商品会跳转到商品详情界面，呈现该商品的详情信息；长按购物中的商品会弹出对话框询问是否移除该商品，点击确定则移除该商品，点击取消则对话框消失。


7. 商品详情界面中点击返回图标会返回上一层，点击星标会切换状态，如果原先是空心星星，则会变成实心星星；如果原先是实心星星 1，则会变成空心星星。点击购物车图标会将该商品添加到购物车中并弹出 Toast 提示:“商品已添加到购物车”。

## 三、 课堂实验结果

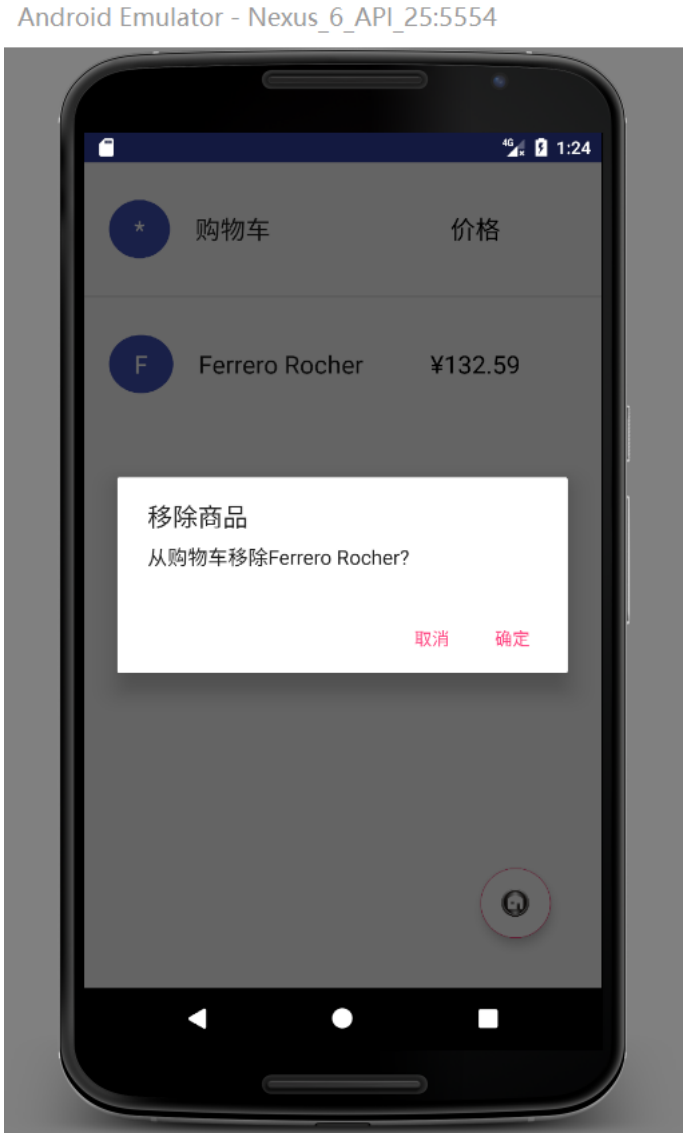
（1） 实验截图

Android Emulator - Nexus_6_API_25:5554


Android Emulator - Nexus_6_API_25:5554
1:23
waitrose 早餐麦片
¥179.00
重量 2Kg
更多产品信息
一键下单
分享商品
不感兴趣
查看更多产品促销信息


Android Emulator - Nexus_6_API_25:5554
1:23
Ferrero Rocher
¥132.59
重量 300g
更多产品信息
一键下单
分享商品
不感兴趣
查看更多
商品已添加到购物车


Android Emulator - Nexus_6_API_25:5554
1:24
*
购物车
价格
F
Ferrero Rocher
¥132.59




（2） 实验步骤以及关键代码

1. 在 activity_main.xml 主界面布局中加入 RecyclerView 控件和 ListView 控件的描述

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/recycler_view"
    android:visibility="visible"
    android:tag="0"/>


<ListView
    android:id="@+id/list_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:visibility="gone"
    android:tag="0"
    />
```

2. 自定义 ViewHolder

```
public class ViewHolder extends RecyclerView.ViewHolder{
    private SparseArray<View> mViews;
    private View mConvertView;
```

上图中，mViews 用来缓存列表项中的 View，mConvertView 用来存储列表项的布局。初始时 mViews 为空，一个 View 都没有缓存，以下是构造函数：

```
public ViewHolder(Context context, View itemView, ViewGroup parent){
    super(itemView);
    mConvertView = itemView;
    mViews = new SparseArray<View>();
}
```

接着实现 getView 函数，通过 getView 函数找到组件 X，先判断 ViewHolder 中是否已缓存了组件 X，若已缓存则直接返回缓存内容；否则则通过 findViewById 方法找到组件 X，将其存入 ViewHolder 的缓存再返回查找结果。

```
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;
}
```

3. 实现 CommonAdapter（RecyclerView 使用）

```
public CommonAdapter(Context context, int layoutId, List<Integer> datas){
    mContext = context;
    mLayoutId = layoutId;
    mDatas = datas;
    tag = true;
}
```

mLayoutId 指向布局组件，mDatas 指向商品列表界面的列表数据。接着为 adapter 创建 ViewHolder：

```
@Override
public ViewHolder onCreateViewHolder(final ViewGroup parent, int viewType){
    ViewHolder viewHolder = ViewHolder.get(mContext, parent, mLayoutId);
    return viewHolder;
}
```

为 adapter 创建点击事件，先定义一个点击事件监听接口：

```
public interface OnItemClickListener{
    void onClick(int position);
    void onLongClick(int position);
}

public void setOnItemClickListener(OnItemClickListener onItemClickListener){
    mOnItemClickListener = onItemClickListener;
}
```

通过 setOnItemClickListener 函数可以为这个 adapter 设置一个点击事件监听对象。接着，继续实现为 ViewHolder 对象传入数据与捕获 ViewHolder 的点击事件，代码如下：

```
@Override
public void onBindViewHolder(final ViewHolder holder, int position){
    convert(holder, mDatas.get(position));
    if (mOnItemClickListener != null){
        holder.itemView.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){
                mOnItemClickListener.onClick(holder.getAdapterPosition());
            }
        });
        holder.itemView.setOnLongClickListener(new View.OnLongClickListener(){
            @Override
            public boolean onLongClick(View v){
                mOnItemClickListener.onLongClick(holder.getAdapterPosition());
                return false;
            }
        });
    }
}
```

convert 函数为 ViewHolder 对象传入数据。当捕获到 ViewHolder 对象的点击事件时，调用自己定义的点击事件监听器的点击事件处理函数。

4. 自定义 ListAdapter（ListView 使用）

```
public class ListAdapter extends BaseAdapter{
    public Context context;
    public List<Integer> goods;

    public ListAdapter(Context context, List<Integer> goods){
        this.context = context;
        this.goods = goods;
    }
```

ListAdapter 继承 BaseAdapter，goods 用来存储购物车列表的列表数据。接着自定义一个 ViewHolder 类：

```
public class ViewHolder{
    public TextView first;
    public TextView name;
    public TextView price;
}
```

因为是继承自抽象类 BaseAdapter，所以需要实现 getCount，getItem，getItemId，getView 四个函数，下面是 getView 函数的实现:

```
@Override
public View getView(int i, View view, ViewGroup viewGroup){
    View convertView;
    ViewHolder viewHolder;
    if (view == null){
        convertView = LayoutInflater.from(context).inflate(R.layout.shopping_list_item, null);
        viewHolder = new ViewHolder();
        viewHolder.first = (TextView) convertView.findViewById(R.id.first);
        viewHolder.name = (TextView) convertView.findViewById(R.id.name);
        viewHolder.price = (TextView) convertView.findViewById(R.id.price);
        convertView.setTag(viewHolder);
    }
    else{
        convertView = view;
        viewHolder = (ViewHolder) convertView.getTag();
    }

    viewHolder.first.setText(goods.get(i).get("firstLetter").toString());
    viewHolder.name.setText(goods.get(i).get("name").toString());
    viewHolder.price.setText(goods.get(i).get("price").toString());

    return convertView;

}
```

convertView 表示列表项的布局组件，viewHolder 则存储列表项中具体的组件。函数的返回值是布局组件，即整个列表项，因为列表是以列表项作为基本单位的，所以在列表中 getView 得到的也应该是列表项。

5. 在主界面定义一个 RecyclerView 和 ListView 对象并分别为其配置一个 CommonAdapter 和 ListAdapter 对象

```
mRecyclerView = (RecyclerView) findViewById(R.id.recycler_view);
mListView = (ListView) findViewById(R.id.list_view);
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
getList();
commonAdapter = new CommonAdapter(this, R.layout.goods_list_item, pl) {
    @Override
    public void convert(ViewHolder holder, int pos){
        TextView name = holder.getView(R.id.name);
        name.setText(listItems.get(pos).get("name").toString());
        TextView first = holder.getView(R.id.first);
        first.setText(listItems.get(pos).get("firstLetter").toString());
    }
};
```

getList 函数是从外部文件获取数据。定义一个 CommonAdapter 对象时需要实现 convert 函数，这个函数的功能是将数据调入 Adapter 对象里面的 ViewHolder 对象中。下面是为 commonAdapter 设置点击事件的代码:

```
commonAdapter.setOnItemClickListener(new CommonAdapter.OnItemClickListener() {
    @Override
    public void onClick(int position) {
        if (commonAdapter.tag == false) return;
        Intent intent = new Intent(MainActivity.this, InfoActivity.class);
        intent.putExtra("name", listItems.get(commonAdapter.mDatas.get(position)).get("name").toString());
        intent.putExtra("type", listItems.get(commonAdapter.mDatas.get(position)).get("type").toString());
        intent.putExtra("price", listItems.get(commonAdapter.mDatas.get(position)).get("price").toString());
        intent.putExtra("info", listItems.get(commonAdapter.mDatas.get(position)).get("info").toString());
        intent.putExtra("pos", commonAdapter.mDatas.get(position));
        intent.putExtra("tag", (Boolean) listItems.get(commonAdapter.mDatas.get(position)).get("tag"));
        startActivityForResult(intent, 1);
    }
    @Override
    public void onLongClick(final int position) {
        if (commonAdapter.tag == false) return;
        commonAdapter.tag = false;
        check(commonAdapter.mDatas.get(position));
        commonAdapter.mDatas.remove(position);
        commonAdapter.notifyDataSetChanged();
        Toast.makeText(MainActivity.this, "移除第" + position + "个商品",
                Toast.LENGTH_SHORT).show();
        commonAdapter.tag = true;
    }
});
```

点击事件发生时，定义一个 Intent 对象实现从主界面向详情界面的跳转，同时使用 putExtra 方法为 Intent 对象传入一系列键值对，实现往商品详情界面传递商品数据。

长按事件发生时，在 commonAdapter 对象中的数据列表删除被长按的数据项，调用 notifyDataSetChanged 方法更新 adapter 的状态，并调用 Toast 显示提示信息。

接着为 RecyclerView 配置 commonAdapter，为 ListView 配置 listAdapter:

```
mRecyclerView.setAdapter(commonAdapter);
listAdapter = new ListAdapter(this, p2);
mListView.setAdapter(listAdapter);
```

为 ListView 对象设置点击事件，点击时使用 Intent 对象实现页面跳转，使用 putExtra 方法传递详情页面需要用到的数据。另外需要注意的是，第一项点击应该是无效的:

```
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        if (position == 0) return;
        if (mListView.getTag().equals("1")) return;
        Intent intent = new Intent(MainActivity.this, InfoActivity.class);
        intent.putExtra("name", listItems.get(listAdapter.goods.get(position)).get("name").toString());
        intent.putExtra("type", listItems.get(listAdapter.goods.get(position)).get("type").toString());
        intent.putExtra("price", listItems.get(listAdapter.goods.get(position)).get("price").toString());
        intent.putExtra("info", listItems.get(listAdapter.goods.get(position)).get("info").toString());
        intent.putExtra("pos", listAdapter.goods.get(position));
        intent.putExtra("tag", (Boolean) listItems.get(listAdapter.goods.get(position)).get("tag"));
        startActivityForResult(intent, 1);
    }
});
```

为 ListView 对象设置长按事件，长按时会弹出一个对话框，点击对话框确定按钮后，将 listAdapter 的数据列表对应项删除，调用 notifyDataSetChanged 方法更新列表状态:

```
mListView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, final int position, long id) {
        if (position == 0) return false;
        if (mListView.getTag().equals("1")) return false;
        mListView.setTag("1");
        AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
        builder.setTitle("移除商品");
        builder.setMessage("从购物车移除" + listItems.get(listAdapter.goods.get(position)).get("name").toString() + "?");
        builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                mListView.setTag("0");
            }
        });
        builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                listAdapter.goods.remove(position);
                listAdapter.notifyDataSetChanged();
                mListView.setTag("0");
            }
        });
        builder.create().show();
        return false;
```

6. FloatingActionButton 的实现

```
public void fabClick(View view){
    fab = (FloatingActionButton) findViewById(R.id.fab);
    if (fab.getTag().equals("0")){
        fab.setImageResource(R.mipmap.mainpage);
        fab.setTag("1");
        mRecyclerView.setVisibility(View.GONE);
        mListView.setVisibility(View.VISIBLE);
    }
    else{
        fab.setImageResource(R.mipmap.shoplist);
        fab.setTag("0");
        mRecyclerView.setVisibility(View.VISIBLE);
        mListView.setVisibility(View.GONE);
    }
}
```

给这个 FloatActionButton 设置一个 tag，通过 tag 来确定当前状态，从而也能确定这个 Button 的图标。当 tag 为 0 时，表示当前是商品列表界面，此时需要隐藏 ListView 而显示 RecyclerView；当 tag 为 1 时，表示当前是购物车界面，此时需要隐藏 RecyclerView 而显示 ListView。

7. 详情界面 InfoActivity 的实现

```
Bundle extras = getIntent().getExtras();
pos = extras.getInt("pos");
tag = extras.getBoolean("tag");
if (extras != null){
    data = extras.getString("name");
    textView = (TextView) findViewById(R.id.name);
    textView.setText(data);
    imageView = (ImageView) findViewById(R.id.show);
    imageView.setImageResource(Id.get(data));
    data = extras.getString("type");
    textView = (TextView) findViewById(R.id.type);
    textView.setText(data);
    data = extras.getString("price");
    textView = (TextView) findViewById(R.id.price);
    textView.setText(data);
    data = extras.getString("info");
    textView = (TextView) findViewById(R.id.info);
    textView.setText(data);
    imageView = (ImageView) findViewById(R.id.star);
    if (!tag) imageView.setImageResource(R.mipmap.empty_star);
    else imageView.setImageResource(R.mipmap.full_star);
}
```

Bundle 对象从 MainActivity 获取数据到本界面，pos 表示传递过来的商品数据在数据源列表的第几项，tag 表示这个商品是否之前被收藏。接着下面很长的一串代码是将传递过来的商品数据与布局中的显示控件进行数据绑定。

下面的代码是星星点击事件的定义，通过设置 tag 来确定该列表项数据是否被收藏:

```
public void starClick(View view){
    star = (ImageView) findViewById(R.id.star);
    if (!tag){
        star.setImageResource(R.mipmap.full_star);
        tag = true;
    }
    else{
        star.setImageResource(R.mipmap.empty_star);
        tag = false;
    }
}
```

接着是回退键点击事件的定义，点击回退键后，会退出当前界面并触发数据回调传递。通过 bundle 对象进行数据传递，pos 表示当前商品是数据源列表第 pos 项数据，num 表示这个商品的购物车按钮被点击了 num 次，tag 表示这个商品是否被点击了收藏，代码如下:

```
public void backClick(View view){
    Bundle bundle = new Bundle();
    bundle.putInt("pos", pos);
    bundle.putInt("num", num);
    bundle.putBoolean("tag", tag);
    Intent intent = new Intent();
    intent.putExtras(bundle);
    setResult(RESULT_OK, intent);
    finish();
}
```

最后是购物车按键的点击事件，当点击购物车按钮后，num 加一，表示该商品的购物次数加一：

```
public void carClick(View view){
    ++num;
    Toast.makeText(InfoActivity.this, "商品已添加到购物车",
            Toast.LENGTH_SHORT).show();
}
```

8. 主界面 MainActivity 的回调数据处理

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data){
    if (resultCode == RESULT_OK){
        Bundle extras = data.getExtras();
        int pos = extras.getInt("pos");
        int num = extras.getInt("num");
        boolean tag = extras.getBoolean("tag");
        listItems.get(pos).put("tag", tag);
        for (int i=0;i<num;++i){
            listAdapter.goods.add(pos);
        }
        listAdapter.notifyDataSetChanged();
    }
}
```

用一个 Bundle 对象获取回调数据，pos 表示需要处理的是数据源列表第 pos 项数据，num 表示需要添加第 pos 项数据 num 次到购物车列表，tag 表示第 pos 项数据是否被点击了收藏。所以先处理 tag，将数据源第 pos 项数据的 tag 标记为传递过来的 tag，然后再将 pos 值往购物车的数据列表传递 num 次，最后调用 notifyDataSetChanged 方法更新购物车列表状态。

（3） 实验遇到困难以及解决思路

1. 当执行长按事件时往往还会触发点击事件，这是不希望看到的。解

决的思路是给 adapter 添加一个 tag，当触发长按事件时标记这个 tag，设置当这个 tag 被标记时无法触发点击事件，长按事件结束后解除这个 tag 的标记。

2. 一开始理解错了 Intent 的机制，以为是跳转到一个 activity 后，之前的 activity 就消失了，所以每次跳转界面包括回退都是新建一个 Intent 跳转界面来实现。这种方式是有问题的，如果每次都要新建一个 Intent 跳到一个新的 activity，这样原来的相同类型的 activity 的数据就无法得到保存了。后来得知使用 Intent 跳转是类似于一个压栈的过程，新的 activity 会在旧的 activity 上方，当调用 finish 后，新的 activity 退栈，旧的 activity 又会重新显示了。

3. 实现点击购物车按钮商品就进入一次购物车列表的功能的时候，一开始我是在 carClick 函数即购物车点击事件中设置每点击一次就回调传递数据一次。然后发现点击多次购物车按钮再点击回退键回到主界面，购物车列表只加了一项数据。后来得知 activity 之间的数据回调只会发生一次（？），所以一次一次的发数据后面的就会覆盖前面的，所以改成先将需要发送的数据集中在一起到按下回退键时一起发送，从而实现点击多次添加多次的功能。

## 四、 课后实验结果

1. 点击某个商品的星星图标表示收藏之后，返回列表界面再回到该商品的详情界面会发现星星图标又回到未点击状态。这里我实现了从列表界面回来依然能保存收藏状态的功能。代码如下:
初始化每个商品的收藏状态 tag 值为 false，表示未收藏:

```
map.put("tag", false);
```

点击事件发生后，传递商品数据时，也将这个 tag 值一并发送:

```
intent.putExtra("tag", (Boolean) listItems.get(commonAdapter.mDatas.get(position)).get("tag"));
intent.putExtra("tag", (Boolean) listItems.get(listAdapter.goods.get(position)).get("tag"));
```

商品详情界面接收这个 tag 值:

```
tag = extras.getBoolean("tag");
```

根据这个 tag 值判断使用哪个星星图标。当 tag 值为真时，表示被收藏，使用 full_star；否则使用 empty_star:

```
if (!tag) imageView.setImageResource(R.mipmap.empty_star);
else imageView.setImageResource(R.mipmap.full_star);
```

当星星图标被点击时，修改这个 tag 值：

```
public void starClick(View view){
    star = (ImageView) findViewById(R.id.star);
    if (!tag){
        star.setImageResource(R.mipmap.full_star);
        tag = true;
    }
    else{
        star.setImageResource(R.mipmap.empty_star);
        tag = false;
    }
}
```

当点击返回键回退到上一个界面时，将这个 tag 值作为回调数据传递回去：

```
bundle.putBoolean("tag", tag);
```

主界面接收这个 tag 值，并更新商品列表数据的 tag 值：

```
boolean tag = extras.getBoolean("tag");
listItems.get(pos).put("tag", tag);
```

2. 当在商品列表将某个商品数据删除后，购物车中相同的商品数据也应该删除，即满足数据的一致性。这里我实现了这个功能，代码如下：

(1) 一个数据列表作为商品数据源，应用整个运行过程的数据增删都没有直接操作这个数据源。

(2) 商品列表与购物车列表使用整数类型的列表存储数据，表示存储数据源第多少项商品数据。

(3) 当长按商品列表界面的某项商品数据使其删除时，先找出该项数据是数据源第几项：

```
check(commonAdapter.mDatas.get(position));
```

get(position)表示取出商品列表第 position 项商品数据对应数据源第几项数据，check 函数是在购物车列表中寻找相同的数据然后删除，具体过程如下：

```
public void check(int pos){
    int i = 1;
    while (i < listAdapter.goods.size()){
        if (listAdapter.goods.get(i).equals(pos)) listAdapter.goods.remove(i);
        else ++i;
    }
    listAdapter.notifyDataSetChanged();
}
```

因为 remove 了第 i 项后，新的第 i 项指向了原先的第 i+1 项，所以 i 不需要加一。

## 五、 实验思考及感想

1. ViewHolder 按照我的理解发挥的作用与 ViewGroup 相似，都是控制着一系列 View 的布局样式。当一个列表展示的每一项数据比较复杂时（内含较多的 View），这时候就可以使用 ViewHolder 来表示这个列表的每一项数据，然后更具体的样式设计可以写到这个 ViewHolder 中。

2. 数据的显示与对数据的增删操作应该要区分开，后台负责对数据进行增删改操作，前段负责显示数据。以本次实验为例，本次实验基本都是前段，需要存储商品数据，较好的做法是只保存一份商品数据作为数据源，商品列表与购物车列表对数据的显示是再新建一个列表，列表保存需要显示的商品数据在数据源的位置，然后通过这个位置在数据源中找到需要的数据并显示数据。删除操作也不需要直接操作数据源，而只要在自己维护的列表中删除这个数据在数据源的位置即可。

3. Intent 跳转到新的 activity 采用的是栈的结构，即进入新界面，新界面压入栈顶；退出当前界面，当前界面退出栈顶。两个界面传递数据有两种，一种是进入新界面时的 Intent 在创建时绑上一些数据到新界面；另一种是新界面回退反馈时，采用 Intent 触发数据回调传递。