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

Broadcast 使用

二、实现内容

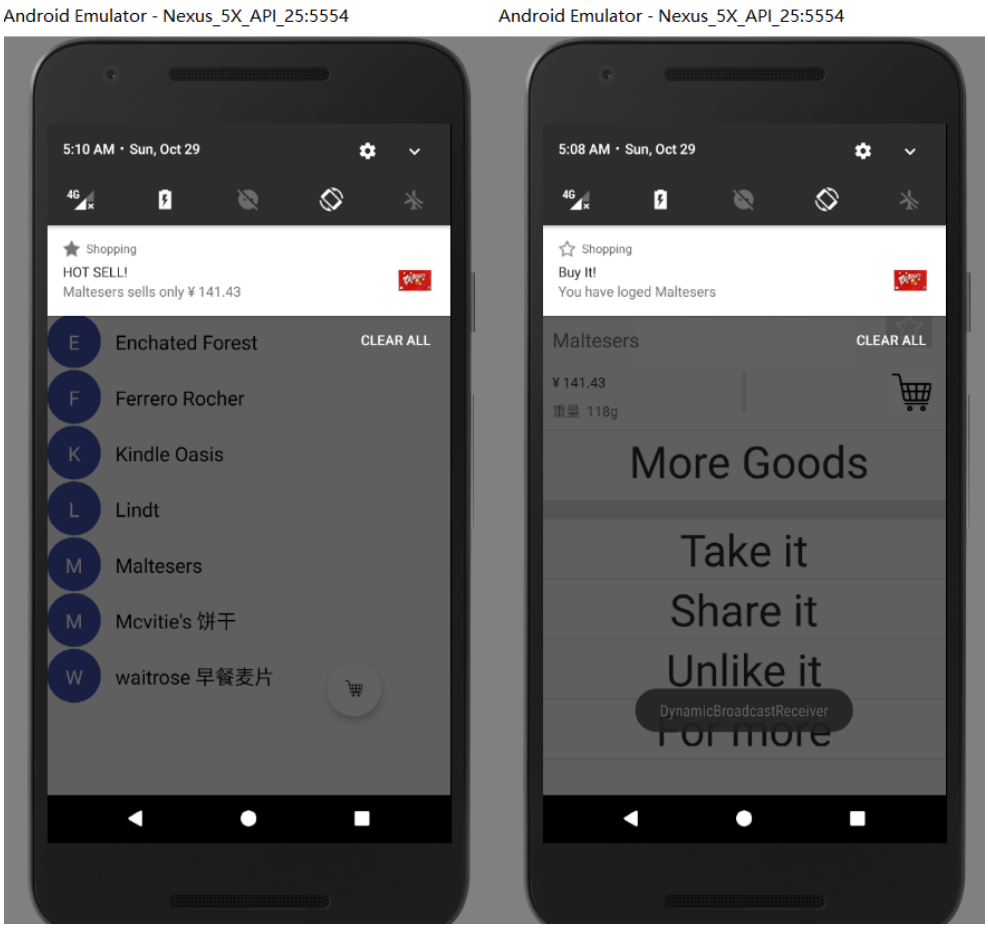
本次实验模拟实现一个商品列表。

- 在启动应用时，会有通知产生，随机推荐一个商品
- 点击通知跳转到该商品详情界面
- 点击购物车图标，会有对应通知产生，并通过Eventbus在购物车列表更新数据
- 点击通知返回购物车列表

实现方式要求:启动页面的通知由静态广播产生，点击购物车图标的通知由动态广播产生

三、课堂实验结果

实验截图



实验步骤以及关键代码

广播接收器，也被称为全局事件，或系统事件。

当Android系统中任何程序有动作时，如果想通知其他程序，采用广播的方式进行传播是非常有效的。

“由于android系统中应用程序之间不能共享内存，因此，在不同应用程序之间交互数据（跨进程通讯）就稍微麻烦一些。Android广播有两种类型，一种是程序内的，一种程序间的。程序内的广播体现不出其真正意义。而程序间的广播，比如，通知时区改变、电池电量低、拍摄了一张照片或者用户改变了语言选项，**这些才是其真正的用途**。Android广播一般用于一个程序告诉另一个程序，某些信息改变了，发生变化了，需要作出相关的应对，并不管有哪些程序接收到这些信息。[1]”

本次实验中的广播机制与Notification机制共同作用，而广播又分为两种类型：静态广播与动态广播。此二者大相径庭，注册方式也不一样。

“某些信息改变了，发生变化了，需要作出相关的应对，并不管有哪些程序接收到这些信息。”

综上，广播有几个特性：发送面向全体成员，不负责接收器行为，接收端针对指定的广播信息执行指定的行为。

为此，需要实现：

- 广播发生器
- 广播接收器
- 广播接收器的过滤机制注册

建立静态广播

静态广播的管理比较智能化和简单化，静态广播属于常驻型广播，在关闭该应用之后依然存在，这种情况下，很显然需要系统的权限与来自系统的服务——因此需要对静态广播进行注册。

注册静态广播

静态的在AndroidManifest.xml中用`<receiver>`标签生命注册，并在标签内用`<intent- filter>`标签设置过滤器。

```
<receiver
    android:name=".StaticBroadcastReciver">
    <intent-filter>
        <action android:name="staticbroadcast" />
    </intent-filter>
</receiver>
```

- 这里第二个参数action是需要监听的消息的名字。

  缘起于广播的机制，之前说过，广播是不在乎接收方是否接收到消息的，广播方只负责发送广播。那么对于接收方的设置，就需要自己写一个广播接收器了——正是笔者正在注册的广播接收器。广播接收器将会接收到指定的信息，继而采取相应的行为，这里我们就设置其只接受第二个参数的信息。

- 这里第一个name是静态广播接收器的名字。

  静态广播接收器也需要一个标志的，静态广播的名字就起到这么一个作用，静态广播的名字就是我们重写的广播的类名字。

静态广播传递数据

静态广播用于发送广播，而发送广播也是需要有内容的，对于静态广播我们使用intent储存静态广播所包含的一系列数据。

```
// STATICATION=静态广播的标志
String STATICATION = "staticbroadcast";
Intent staticintent = new Intent(STATICATION);// intent即为广播内容
// 为bundle储存数据
Bundle bundle = new Bundle();
int note_id = new Random().random.nextInt(10);
bundle.putInt("note_id", goodid[note_id]);
bundle.putString("note_name", Name[note_id]);
// 将广播数据设置为bundle内的数据
staticintent.putExtras(bundle);
sendBroadcast(staticintent);// 发送广播
```

这样，静态广播发生器就发送了这一次的广播内容。由于sendBroadcast函数是在MainActivity中调用的，这就说明，一旦该APP打开，则广播发送。

接收静态广播

静态广播发送出之后，我们在系统中注册了静态广播的接收器。

- 静态广播发送的信息标志是："staticbroadcast"
- 而我们定义的静态广播接收器的信息标志也是："staticbroadcast"
- 那么，接收器就启动了

接收到静态广播之后，应该采取一些行动的。但是采取什么行动，在广播中是没有定义的，这正是我们之前所提及的，广播仅仅负责发送信息，接收方的行为不在考虑范围之内，也正是因此，在实现基类BroadcastReceiver的时候，设置onReceive()函数为@Override，意为一定要重新实现一下。

重新实现**onReceive()**函数

我们希望在接收方（这里因为是在系统中注册的，所以是系统）接收到广播的信息之后，可以采取相应的行为，那么就需要在接收之后的部分添加内容：

```
public class StaticBroadcastReciver extends BroadcastReceiver {
    private static final String STATICATION = "staticbroadcast";

    @Override
    public void onReceive(Context context, Intent intent) {
        if(intent.getAction().equals(STATICATION)) {具体行为}
```

为了使用notification在下拉栏中添加我们需要的信息，我们需要使用notification类。

- 因为下拉栏属于系统，因此我们需要使用manager向系统交涉

  获取启动系统的通知权限：`getSystemService`

```
NotificationManager manager
    = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
```

- 并为通知栏添加信息

  使用builder，为Notification添加信息。

```
Notification.Builder builder = new Notification.Builder(context);
builder.setContentText("BUY IT!")
  .setContentTitle("HOT SELL!")
  .setAutoCancel(true);
```

- 发送通知（具有权限不代表采取行动）

```
Notification notify = builder.getNotification();
manager.notify(0, notify);
```

这样，在开始APP的时候，就可以看到下拉栏的推荐信息了。

通知栏的界面切换

如果希望在点击通知界面的时候，界面可以跳转到一个新的界面。需要使用一个类：`PendingIntent`：

```
PendingIntent Jump_java
        = PendingIntent.getActivity
            (context, 0, To_java, PendingIntent.FLAG_UPDATE_CURRENT);
// To_java 表示跳转界面的intent
```

Pendingintent表示尚未成熟的intent，只需要按照参数要求传入参数，就可以运行。这个部分放在了设置通知builder的前面，而在通知出现之后，该intent也没有马上作用，而是等到我们点击通知的时候，Pendingintent成熟，继而实现跳转。

表示跳转的intent——To_java

```
Intent To_java = new Intent(context, GoodsDetail.class);
```

这表示我们将要跳转到GoodsDetail的界面，当然这与一般的intent用法一致，如果希望该界面显示特定的信息，请传入相关的参数。

建立动态广播

动态广播的注册方式不同于静态广播，动态广播不需要在Manifest.xml文件里声明，而是在MainActivity函数中声明，与静态广播一样，也需要定义广播接收器以及过滤信息：

```
// 声明动态广播
DynamicBroadcastReceiver dynamicBroadcastReceiver
        = new DynamicBroadcastReceiver();
// 接收器的接受信息限制
IntentFilter dynamic_filter = new IntentFilter("dynamicbroadcast");
// 注册
registerReceiver(dynamicBroadcastReceiver, dynamic_filter);
```

发送与接收动态广播

之前是在MainActivity中发送广播的，而这一次如果我们需要在特定的情况下发送，就可以在其他界面的某一个函数下调用sendBroadcast函数。

接收广播时候也一定会采取特殊的行为，该过程与静态设置一致，也需要重写onReceive()函数：

```
public class DynamicBroadcastReceiver extends BroadcastReceiver {
    private static final String DYNAMICATION = "dynamicbroadcast";
    @Override
    public void onReceive(Context context, Intent intent) {
        if(intent.getAction().equals(DYNAMICATION)){}
```

点击通知栏实现界面跳转

具体实现过程与之前很相似。

设置跳转界面intent:

```
Intent To_java = new Intent(context, MainActivity.class);
```

完成Pendingintent:

```
PendingIntent Jump_java
        = PendingIntent.getActivity
            (context, 0, To_java, PendingIntent.FLAG_UPDATE_CURRENT);
```

将跳转intent传入builder中:

```
builder.setContentIntent(Jump_java)
```

只是跳转之后，需要显示RecyclerView隐藏而ListView显示的界面， 为了实现应有的跳转，我们先找到负责跳转操作的变量:

```
// cart 就是控制界面切换的按钮
cart.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(cart_attitude == false){ }// boolen 值就是控制切换的变量啦
        else if(cart_attitude == true){ }
    }
});
```

接下来的工作就是允许这个值受到来自dynamicBroadcast的影响:

```
To_java.putExtra("cart_attitude", false);
```

这里笔者使用了intent实现boolen值的传递:

```
Intent from_dynamic = getIntent();
cart_attitude = from_dynamic.getBooleanExtra("cart_attitude", true);
```

这样，就可以得到从dynamicBroadcast跳转之后的界面了。

实验遇到困难以及解决思路**

动态广播的smallicon必要性

实现动态广播的时候，笔者为之传入通知栏显示的数据：

```
builder.setContentText("waiting for what")
        .setContentTitle("HOT SELL!")
        .setTicker("HouHouHouHou")
        .setAutoCancel(true)
        .setContentIntent(Jump_java)
        .setSmallIcon(R.mipmap.empty_star);
```

但是传入的数据有一个问题，就是如果不传入SmallIcon的数据，就会出现闪退，并且在该onReceive()函数中，不能为bundle赋值，否则也会闪退——这是因为在sendBroadcast的函数中没有为当前的广播传入信息，这导致在后续申请显示图片的时候，内存空间超出，故而闪退。

OnClick修改全局变量不能保持

在笔者使用全局变量的时候，有一些变量需要在OnClick内部进行修改，笔者希望修改之后的变量可以保持到Clock监听之外，但是事实上不能如愿。

解决方案：使用onNewIntent函数。

onNewIntent 避免重复启用一个新的界面。使用Intent实现开启新的界面的时候，计时已经存在一个界面，Intent依然会创建一个新的界面，而由于我们之前在manifest.xml中设置了单任务模式，使用onNewIntent则不会发生这种情况，如果存在这样一个实例，则自动启用onNewIntent，不再使用OnCreate函数。

```
@Override
protected void onNewIntent(Intent intent){
    recyclerView.setVisibility(View.INVISIBLE);
    cart_list.setVisibility(View.VISIBLE);
    cart.setImageDrawable(getResources().getDrawable(R.mipmap.mainpage));
    cart_attitude = true;
}
```

onNewIntent不能实时更新的问题

如果发一个通知给状态栏，然后点击这个通知，自然会执行 PendingIntent 里边的Intent。

但是，有一次在Activity那边的 onNewIntent()方法里边得到的数据不是最新的——在使用PendingIntent的时候（就是延迟启用的intent），需要在PendingIntent的后面添加一个Flag：

```
PendingIntent Jump_java
        = PendingIntent.getActivity
            (context, 0, To_java, PendingIntent.FLAG_UPDATE_CURRENT);
```

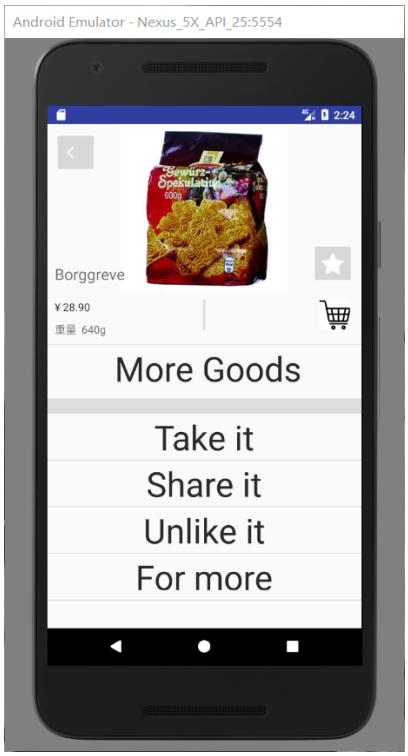
Smallicon不能显示的问题

Smallicon本应该显示在下拉栏的通知栏中，但是无论如何笔者都不能使得其清晰地显示，在下拉栏中的小图标总是灰色的矩形——可能是版本问题，总之笔者至今还没有修复。

四、课后实验结果

标记星信息的同步化

本次实验笔者依然延续了之前对于标记星的处理，就是无论在哪一个界面点击了标记星，在进入另一个界面的时候，都可以实现同步：



在安全性角度的考虑

笔者在实现跳转界面的功能的时候，使用了函数onNewIntent，该函数可以设置在已有界面的情况下，在单任务模式的基础上，重新调用该界面的时候，申请已有的界面布局。

”当我们把Activity 启动模式设置为 singleTask 之后 当我们下次再去用Intent 启动这个 Activity时候就不会去调用onCreate方法而是去调用onNewIntent()方法然后把Intent中的数据传给它。[2]“

但是系统可能会随时杀掉后台运行的Activity，如果这一切发生，那么系统就会调用onCreate方法，而不调用onNewIntent方法，在此情况下我们就应该在函数OnCreate中根据条件判断使用同样处理数据的方法：

```
recyclerView.setVisibility(View.INVISIBLE);
cart_list.setVisibility(View.VISIBLE);
cart.setImageDrawable(getResources().getDrawable(R.mipmap.mainpage));
cart_attitude = true;
```

五、实验思考及感想

本次实验的主要任务是完成静态广播以及动态广播的设计。

目前的完成情况来看，广播需要的步骤如下：

- 添加广播的内容，调用sendBroadcast函数发送广播
- 注册广播接收器，动态或者静态
- 重写广播接收器函数onReceive();

如果需要发送通知，使用Notification：

- 申请权限需要NotificationManager

- 设置属性需要Notification.builder
- 发送通知需要Notification

如果需要实现界面的转移：

- Intent关联用于切换的界面
- PendingIntent用于显示切换的内容
- 切换的界面如果已经存在则使用onNewIntent函数配合单任务注册，避免重建

广播的布局问题，内容问题，触发问题都是相对独立的，完全可以各自独立地完成。广播的设计还是相对独立的，至于内容的设定则是在重构函数内部实现。这就是开发体系结构上的层次化，使得开法国车结构清晰，分级完整。而拼接组合也非常有趣，尽管以上面对的函数功能千差万别，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，都在遵循函数运行的基本架构——如函数sendBroadcast的使用，就是简单清晰、容易理解的。至于Intent_filter的过滤，有很大一部分并不需要我们去操作。

从博客到代码。如果在了解使用DynamicBroadcast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。

另外，宏观把控在实现某一种功能的时候意义非常，如在实现通知界面的时候，需要首先知道应该有的几个类，以及可能会用到的几个函数。不一定可以一次跑出期望的运行结果，但是很显然可以加快迭代测试的步骤，如果跑出了期望的结果，则可以节省很多测试的时间，加快开发的步骤。建议在实现某一个功能之前先理解大致的步骤（或者流程），罗列相关的函数，理清思路，再根据博客上的教程进行测试和编辑格式上的调整。

对于隐秘的函数接口问题。编程的时候需要对内存设置添加一些考虑，有时候这一点可能表现在传入数据结构的空间开辟上，有时候可以出现在函数中的参数设置上，有时候可能出现在报错信息上。内当使用一个数据结构的时候，很明显的一点就是证明其有存在的证据——空间。单纯的输出信息还则罢了，一旦为之添加任何的数据结构信息的时候，都可能出现意想不到的错误，后续处理是很麻烦的。在此基础上，使用数据结构就一定要谨慎——很简单，没有对于数据结构的了解而随表调用或者修改数据结构，很可能引起内存错误的问题——因为我们不知道其需要什么，也不知道什么情况下需要位置添加怎样的控件信息。

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。

---

1. 摘自CoderSun的博客http://blog.csdn.net/limonzet/article/details/52338019↩

2. 摘自lihenair的博客http://blog.csdn.net/lihenair/article/details/28892921↩