# 订单管理

订单模块工作内容如下：

1. 从服务器获取订单列表数据。
2. 点击Item时传递订单数据（如：标识、状态）到订单详情界面，其他详细数据可以通过订单标识到服务器查询。
3. 主要工作内容：数据展示，订单状态的同步处理。

## 数据展示

从服务器获取订单列表，使用RecyclerView展示即可。下拉刷新使用: **android.support.v4.widget.SwipeRefreshLayout**

由于已进入该界面就需要到服务器获取最新订单列表信息，所以进入到该界面需要显示**SwipeRefreshLayout**的滚动条信息。

显示滚动条：

**swipeRefreshLayout**.post(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **swipeRefreshLayout**.setRefreshing(**true**);  
 }  
});

业务层处理服务器数据获取及处理。

## 订单状态的同步

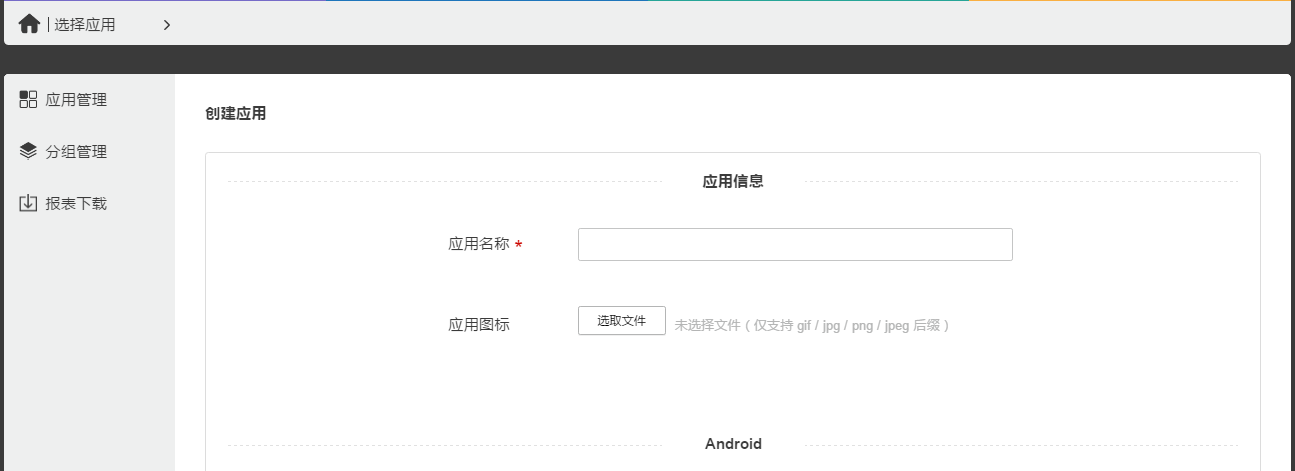
当订单状态发生变动时，我们需要尽快更新用户的订单展示信息，同时要保证两个界面的数据同步。

当用户下单后我们需要实时监听订单的状态，我们使用激光推送进行模拟。

### 极光推送

主页：<https://www.jiguang.cn/>

需要注册开发者，并添加应用信息。



其服务器会针对你添加的包名生成对应的DEMO

注意：由于libs中的内容含有.so文件所以我们需要将内容放到jnilibs目录下面给，在main下面创建即可。Project视图/app/src/main/jnilibs

#### Manifests配置

权限：

<**permission  
 android:name="com.example.elem.permission.JPUSH\_MESSAGE"  
 android:protectionLevel="signature"** />  
  
*<!-- Required 一些系统要求的权限，如访问网络等-->*<**uses-permission android:name="com.example.elem.permission.JPUSH\_MESSAGE"** />  
<**uses-permission android:name="android.permission.RECEIVE\_USER\_PRESENT"** />  
<**uses-permission android:name="android.permission.INTERNET"** />  
<**uses-permission android:name="android.permission.WAKE\_LOCK"** />  
<**uses-permission android:name="android.permission.READ\_PHONE\_STATE"** />  
<**uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"** />  
<**uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"** />  
<**uses-permission android:name="android.permission.WRITE\_SETTINGS"** />  
<**uses-permission android:name="android.permission.VIBRATE"** />  
<**uses-permission android:name="android.permission.MOUNT\_UNMOUNT\_FILESYSTEMS"** />  
<**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"** />  
<**uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"** />

其他内容：

*<!-- Rich push 核心功能 since 2.0.6-->*<**activity  
 android:name="cn.jpush.android.ui.PopWinActivity"  
 android:exported="false"**>  
</**activity**>  
  
*<!-- Required SDK核心功能-->*<**activity  
 android:name="cn.jpush.android.ui.PushActivity"  
 android:configChanges="orientation|keyboardHidden"  
 android:theme="@android:style/Theme.NoTitleBar"  
 android:exported="false"**>  
 <**intent-filter**>  
 <**action android:name="cn.jpush.android.ui.PushActivity"** />  
 <**category android:name="android.intent.category.DEFAULT"** />  
 <**category android:name="com.example.elem"** />  
 </**intent-filter**>  
</**activity**>  
*<!-- Required SDK核心功能-->*<**service  
 android:name="cn.jpush.android.service.DownloadService"  
 android:enabled="true"  
 android:exported="false"** >  
</**service**>  
  
  
*<!-- Required SDK 核心功能-->  
<!-- 可配置android:process参数将PushService放在其他进程中 -->*<**service  
 android:name="cn.jpush.android.service.PushService"  
 android:enabled="true"  
 android:exported="false"**>  
 <**intent-filter**>  
 <**action android:name="cn.jpush.android.intent.REGISTER"** />  
 <**action android:name="cn.jpush.android.intent.REPORT"** />  
 <**action android:name="cn.jpush.android.intent.PushService"** />  
 <**action android:name="cn.jpush.android.intent.PUSH\_TIME"** />  
 </**intent-filter**>  
</**service**>  
  
*<!-- since 1.8.0 option 可选项。用于同一设备中不同应用的JPush服务相互拉起的功能。 -->  
<!-- 若不启用该功能可删除该组件，将不拉起其他应用也不能被其他应用拉起 -->*<**service  
 android:name="cn.jpush.android.service.DaemonService"  
 android:enabled="true"  
 android:exported="true"**>  
 <**intent-filter**>  
 <**action android:name="cn.jpush.android.intent.DaemonService"** />  
 <**category android:name="com.example.elem"** />  
 </**intent-filter**>  
  
</**service**>  
  
*<!-- Required SDK核心功能-->*<**receiver  
 android:name="cn.jpush.android.service.PushReceiver"  
 android:enabled="true"  
 android:exported="false"**>  
 <**intent-filter android:priority="1000"**>  
 <**action android:name="cn.jpush.android.intent.NOTIFICATION\_RECEIVED\_PROXY"** /> *<!--Required 显示通知栏 -->* <**category android:name="com.example.elem"** />  
 </**intent-filter**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.USER\_PRESENT"** />  
 <**action android:name="android.net.conn.CONNECTIVITY\_CHANGE"** />  
 </**intent-filter**>  
 *<!-- Optional -->* <**intent-filter**>  
 <**action android:name="android.intent.action.PACKAGE\_ADDED"** />  
 <**action android:name="android.intent.action.PACKAGE\_REMOVED"** />  
 <**data android:scheme="package"** />  
 </**intent-filter**>  
  
</**receiver**>  
  
*<!-- Required SDK核心功能-->*<**receiver android:name="cn.jpush.android.service.AlarmReceiver" android:exported="false"**/>  
  
*<!-- User defined. For test only 用户自定义的广播接收器-->*<**receiver  
 android:name="com.example.jpushdemo.MyReceiver"  
 android:exported="false"  
 android:enabled="true"**>  
 <**intent-filter**>  
 <**action android:name="cn.jpush.android.intent.REGISTRATION"** /> *<!--Required 用户注册SDK的intent-->* <**action android:name="cn.jpush.android.intent.MESSAGE\_RECEIVED"** /> *<!--Required 用户接收SDK消息的intent-->* <**action android:name="cn.jpush.android.intent.NOTIFICATION\_RECEIVED"** /> *<!--Required 用户接收SDK通知栏信息的intent-->* <**action android:name="cn.jpush.android.intent.NOTIFICATION\_OPENED"** /> *<!--Required 用户打开自定义通知栏的intent-->* <**action android:name="cn.jpush.android.intent.ACTION\_RICHPUSH\_CALLBACK"** /> *<!--Optional 用户接受Rich Push Javascript 回调函数的intent-->* <**action android:name="cn.jpush.android.intent.CONNECTION"** />*<!-- 接收网络变化 连接/断开 since 1.6.3 -->* <**category android:name="com.example.elem"** />  
 </**intent-filter**>  
</**receiver**>  
  
  
*<!-- Required . Enable it you can get statistics data with channel -->*<**meta-data android:name="JPUSH\_CHANNEL" android:value="developer-default"**/>  
<**meta-data android:name="JPUSH\_APPKEY" android:value="0a27061f"** />

#### 使用

1. 初始化极光推送：JPushInterface.*init*(**this**)在用到极光推送之间调用就可以了。
2. 信息由自己的广播接收者接受。相关参数信息。
3. @Override  
   **public void** onReceive(Context context, Intent intent) {  
    Bundle bundle = intent.getExtras();  
    **if** (JPushInterface.***ACTION\_MESSAGE\_RECEIVED***.equals(intent.getAction())) {  
    *// SDK 向 JPush Server 注册所得到的注册 全局唯一的 ID ，可以通过此 ID 向对应的客户端发送消息和通知。* String id = bundle.getString(JPushInterface.***EXTRA\_REGISTRATION\_ID***);  
    *//保存服务器推送下来的消息的标题。* String title = bundle.getString(JPushInterface.***EXTRA\_TITLE***);  
    *//保存服务器推送下来的消息内容* String message = bundle.getString(JPushInterface.***EXTRA\_MESSAGE***);  
    *//保存服务器推送下来的附加字段。这是个 JSON 字符串。* String extras = bundle.getString(JPushInterface.***EXTRA\_EXTRA***);  
    *//保存服务器推送下来的内容类型。* String type = bundle.getString(JPushInterface.***EXTRA\_CONTENT\_TYPE***);  
    }  
   }

### 观察者处理同步更新订单状态

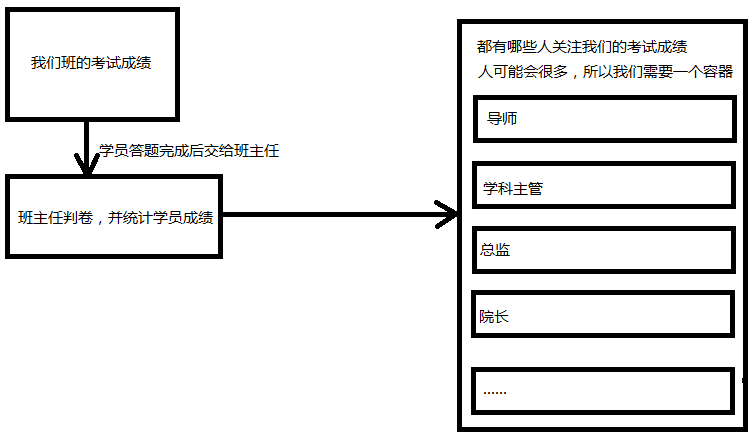
当获取到服务器推送的信息后，广播接受者需要对数据做简单处理后，传递给相关界面。

问题来了：如何从广播接受者中将数据交给某个Fragment，某个Activity，特别是已经启动的界面。

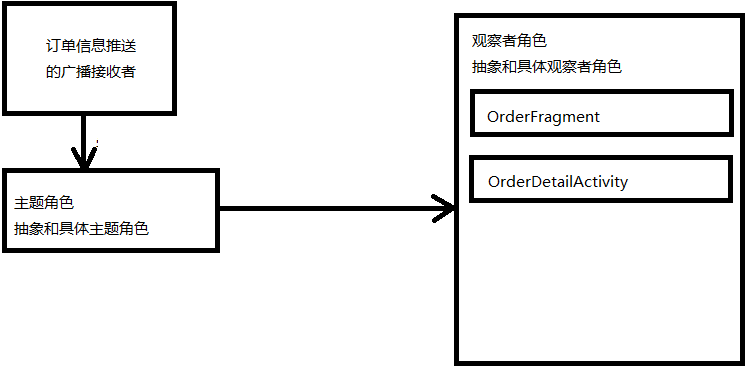
1. 如果是新开的Activity我们可以通过Intent传递数据。
2. 对于已经启动的界面我们可以通过观察者进行数据的传递，类似与ListView的Adapter，当数据变动了，我们只需要notify一下。

观察者模式：

生活实例

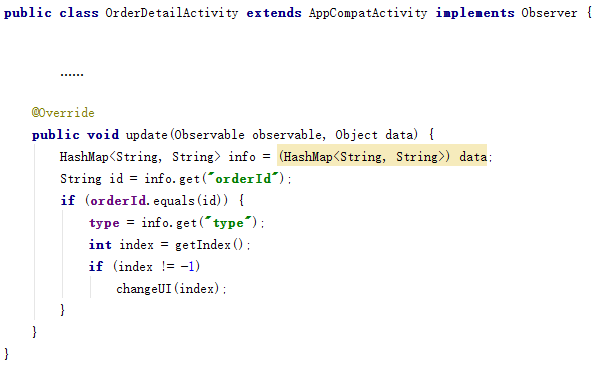


对照关系

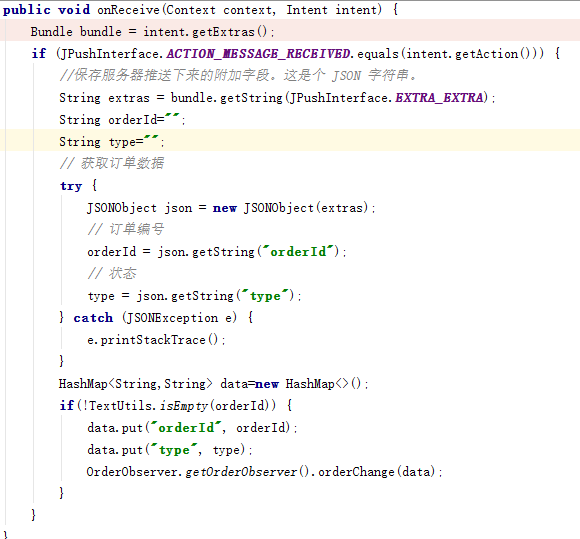
****

利用系统提供的观察者抽象主题角色（Observable）和抽象观察者角色（Observer）实现代码如下：





极光推送的数据获取到后，直接触发OrderObserver的orderChange方法即可。



## 配送信息展示

### 配送需求

餐品在配送阶段，用户会非常关注何时餐品能够送到自己手中，所以我们需要将配送的过程中信息一一展示给用户，而这些信息最核心的就是骑手的信息。比如：骑手去取餐了，距离餐厅还有多远，已经拿到餐品了，距离买家还有多远等。

### 配送的处理流程

1. 首先修改订单状态为配送中，需要显示地图，并在地图上需要显示出商家和用户的标识。
2. 当骑手发现有送单的需求，并且自己离商家不远的情况下，可以接单。如果没有就需要单独从配送站出发到商家取餐。
3. 接单后会在地图上显示骑手的位置，此时地图的核心是骑手位置信息，所以整个地图的中心点都是跟着骑手在走。
4. 骑手会向商家骑行，过程中会发送位置信息，我们可以通过当前的骑手位置与商家的位置计算出两者之间的距离。随着不断接受位置信息我们可以快速更新骑手位置在用户的应用中显示信息。
5. 当骑手拿到餐品后，状态为送餐状态，此时需要显示骑手距离买家的距离信息，处理同4。

### 处理步骤

1. 地图引入
2. 细化配送状态：包括订单开始配送，骑手已接单，骑手取餐、骑手送餐。对于订单来说，这些都是配送状态。

*// 配送中*

**public static final** String ***ORDERTYPE\_DISTRIBUTION*** = **"40"**;

*// 配送中的状态变化:骑手接单、骑手取餐、骑手送餐***public static final** String ***ORDERTYPE\_DISTRIBUTION\_RIDER\_RECEIVE*** = **"43"**;  
**public static final** String ***ORDERTYPE\_DISTRIBUTION\_RIDER\_TAKE\_MEAL*** = **"46"**;  
**public static final** String ***ORDERTYPE\_DISTRIBUTION\_RIDER\_GIVE\_MEAL*** = **"48"**;

1. 当手机端接收到***ORDERTYPE\_DISTRIBUTION***状态时，我们可以将买卖两家的位置在地图中标记出来。注意显示级别和中心点设置。
2. 当手机端接收到***ORDERTYPE\_DISTRIBUTION\_RIDER\_RECEIVE***后，我们需要在用户的手机上显示骑手的信息（记录骑手的位置，这将是我们绘制骑手行进路线的起点），并且骑手将作为地图的中心。
3. 手机端后续会陆续接收到***ORDERTYPE\_DISTRIBUTION\_RIDER\_TAKE\_MEAL***信息，我们需要在手机上不断更新骑手的位置，过程中需要保持骑手一直在地图的中心点，并标记出骑手距离商家的距离。同时，为了能够满足骑手行走路线绘制的需求，我们可以建立一个骑手经过位置点的集合，将前后相邻的两个点坐标做连线处理，最终能够绘制出骑手的行走轨迹。两个点分别为，当前坐标和历史记录集合中下标离这个点最近的点。
4. 当骑手拿到餐品后，修改状态为***ORDERTYPE\_DISTRIBUTION\_RIDER\_GIVE\_MEAL***，即骑手向着买家来了，我们会不断获取到服务器推送的骑手坐标信息，通过骑手的坐标我们可以计算出骑手与买家之间的距离。即在第4和5步中我们都会不停的跟新骑手信息。

### 代码实现

1. 地图引入

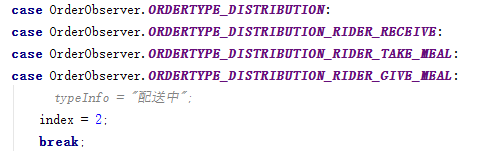
libs中添加2D地图包



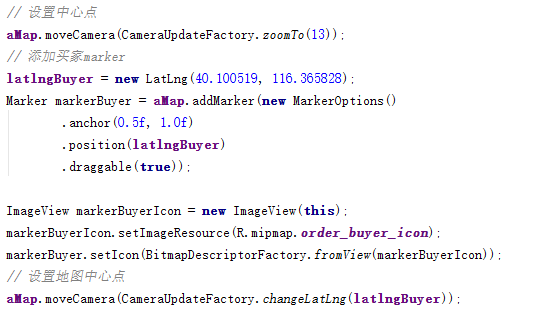
添加地图key的配置

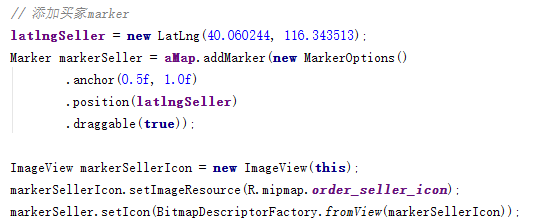


1. 修改状态信息，骑手的几个状态均属于订单配送

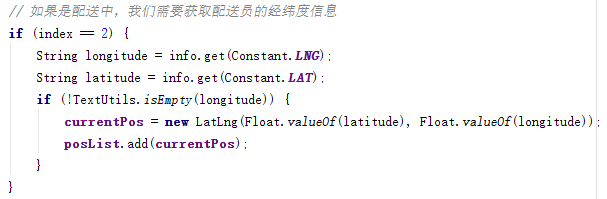


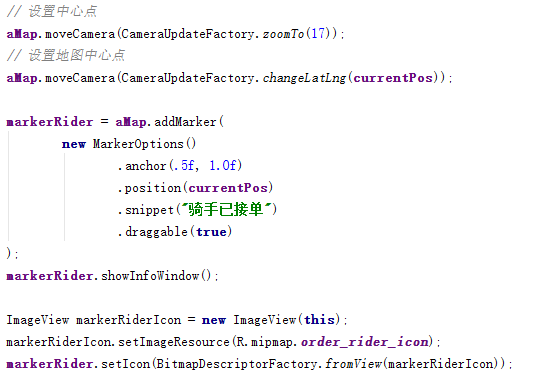
1. 当手机端接收到***ORDERTYPE\_DISTRIBUTION***状态时，我们可以将买卖两家的位置在地图中标记出来。注意显示级别和中心点设置。



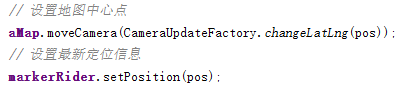


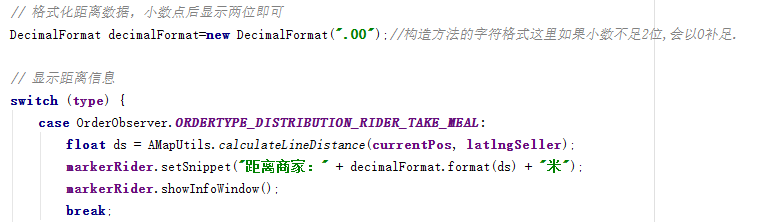
1. 当手机端接收到***ORDERTYPE\_DISTRIBUTION\_RIDER\_RECEIVE***后，我们需要在用户的手机上显示骑手的信息（记录骑手的位置，这将是我们绘制骑手行进路线的起点），并且骑手将作为地图的中心。

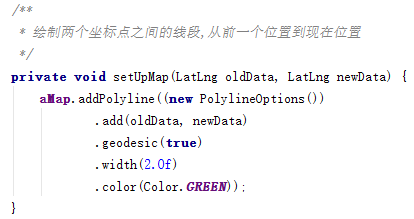




1. 手机端后续会陆续接收到***ORDERTYPE\_DISTRIBUTION\_RIDER\_TAKE\_MEAL***信息，我们需要在手机上不断更新骑手的位置，过程中需要保持骑手一直在地图的中心点，并标记出骑手距离商家的距离。同时，为了能够满足骑手行走路线绘制的需求，我们可以建立一个骑手经过位置点的集合，将前后相邻的两个点坐标做连线处理，最终能够绘制出骑手的行走轨迹。两个点分别为，当前坐标和历史记录集合中下标离这个点最近的点。







1. 当骑手拿到餐品后，修改状态为***ORDERTYPE\_DISTRIBUTION\_RIDER\_GIVE\_MEAL***，即骑手向着买家来了，我们会不断获取到服务器推送的骑手坐标信息，通过骑手的坐标我们可以计算出骑手与买家之间的距离。即在第4和5步中我们都会不停的跟新骑手信息。

