天津大学

《专业课程设计2》结课报告

北洋Wal-Mart——天大人的电子购物平台

学　　院　智能与计算学部

专　　业　软件工程

年　　级　2017级

姓　　名　赵为、宋高超

　　张昊臻、刘宇

学　　号 3017210149 3017214075

　　3017226054 3017212044

2019 年 9 月 6 日

目　　录

第一章　软件需求规格说明书

1.1　项目背景

沃尔玛百货有限公司由美国零售业的传奇人物山姆·沃尔顿先生于1962年在阿肯色州成立。沃尔玛公司已经成为美国最大的私人雇主和世界上最大的连锁零售企业。沃尔玛在全球27个国家开设了超过10,000家商场，下设69个品牌，全球员工总数220多万人，每周光临沃尔玛的顾客达到2亿人次。在如此庞大的公司的影响下，本团队打算打造一个属于天大学子自己的电子购物平台，这也是“北洋Wal-Mart”项目名称的由来。在本平台，学子们可以通过虚拟货币“洋克拉”进行各种网上商品交易，学子们不仅可以通过“洋克拉”进行商品的买入，可也以通过自己创建店铺的方式，向其他同学售卖自己的物品，赚取货币。本平台更是提供了商品按不同分类进行排列、添加关注店铺、一键清空购物车等方便的功能，使得本电子购物平台更加人性化。

1.2　开发环境

- 前端IDE：微信开发者工具（微信小程序）、Android Studio（安卓app）、Pycharm（浏览器前端）
- 后端IDE：Pycharm
- 框架使用：Django、Django REST framework
- 服务器配置：阿里云服务器，Linux发行版CentOS 7.4
- 服务器域名：http://www.mallproject.cn
- 编程语言：wxml、wxss、JavaScript、java、python、html
- 操作系统：Windows 10

1.3　需求分析

- 提供用户注册登录基本功能
- 提供用户的个人基本相关信息设置
- 提供提供商品大类与子类的划分，以及商品的各种基本相关信息
- 提供商品轮播图展示的功能
- 提供虚拟货币与购物车购物的功能
- 提供个人用户自己创建店铺的功能
- 提供商品的用户关注商品与店铺功能
- 提供浏览记录功能
- 根据各种关键字检索商品功能
- 提供商品按不同属性排序功能

- 提供收货地址功能

1.3.1 系统框架图

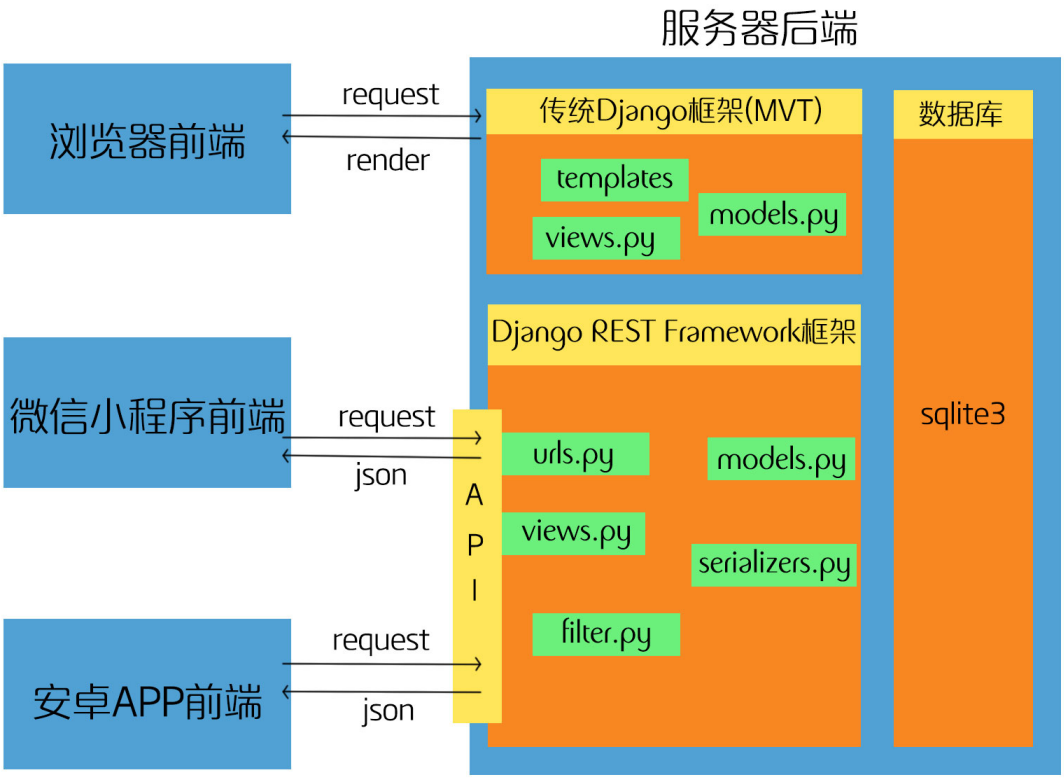

图 1-1 系统框架图

1.3.2 系统流程图

1.3.3 部分顺序图

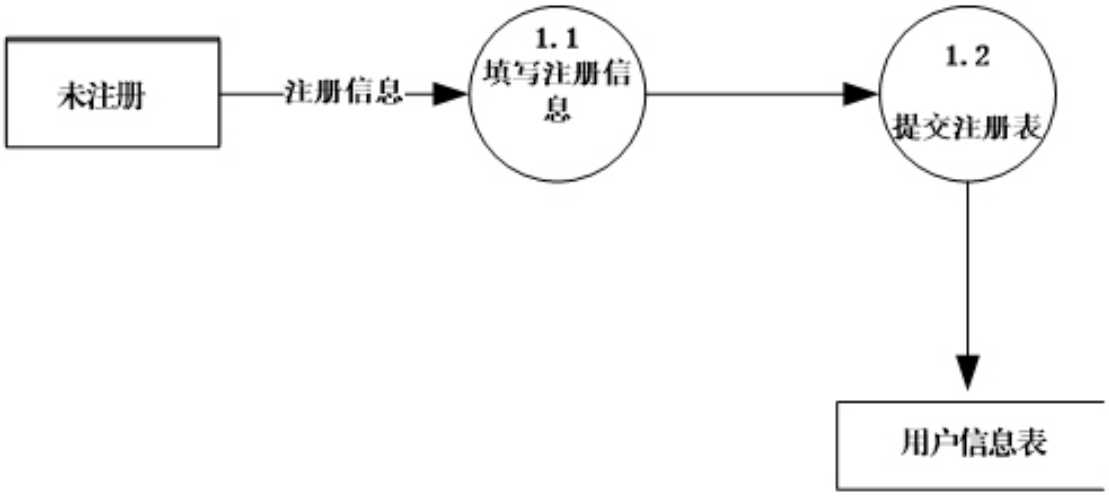

图 1-2　　注册数据流图

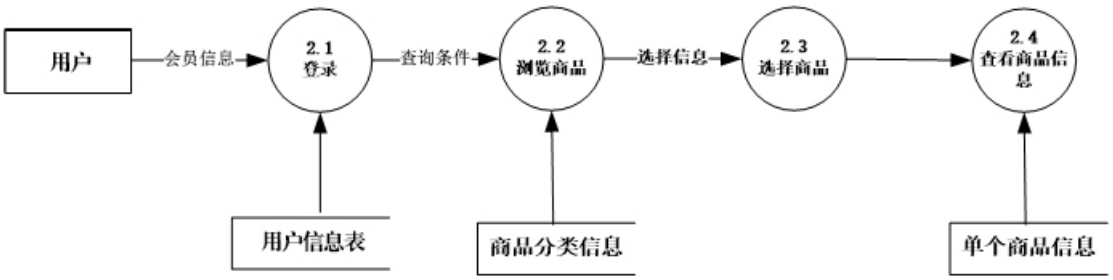

图 1-3　　查询商品数据流图

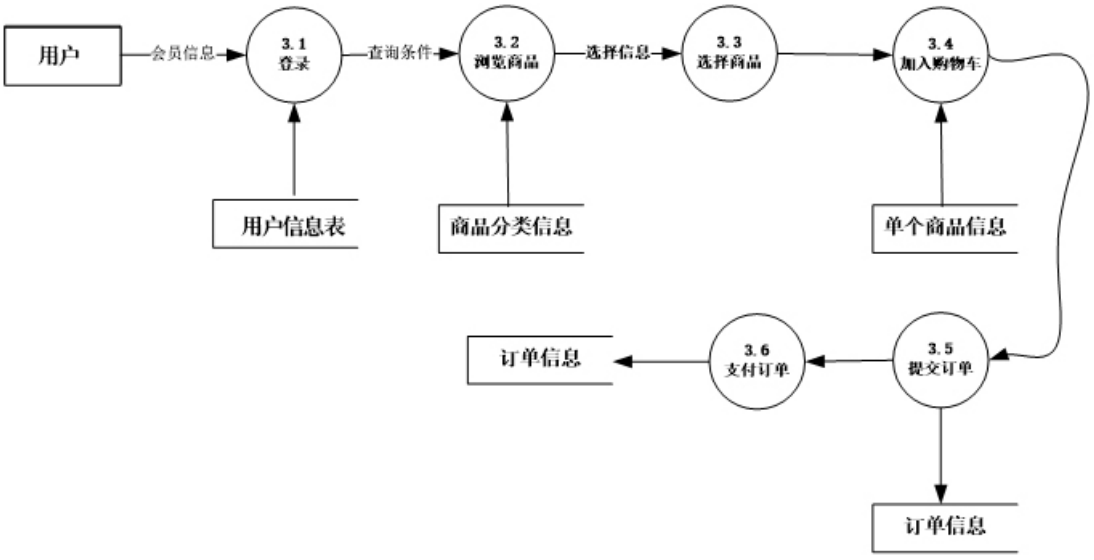

图 1-4　　购买商品数据流图

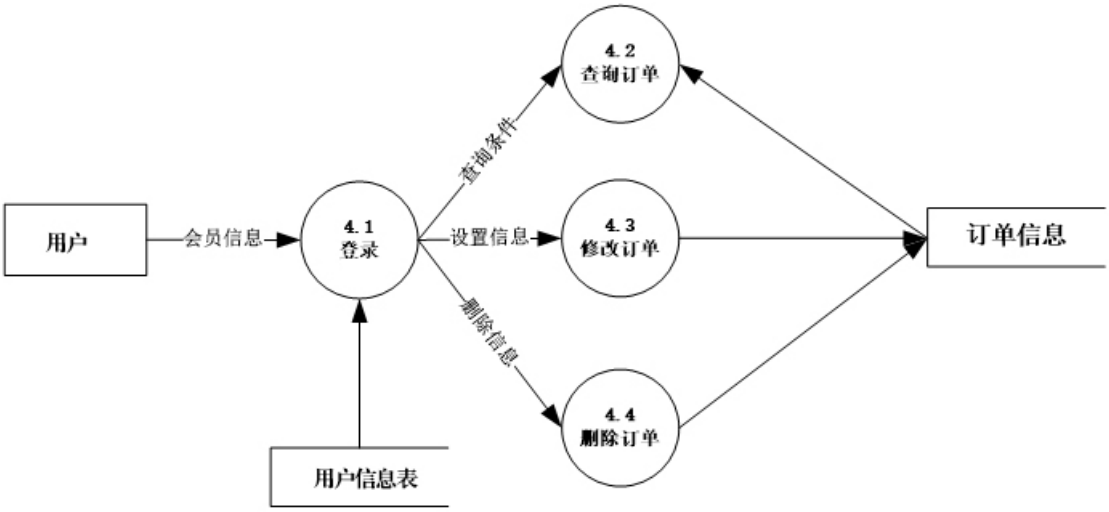

图 1-5　　订单管理数据流图

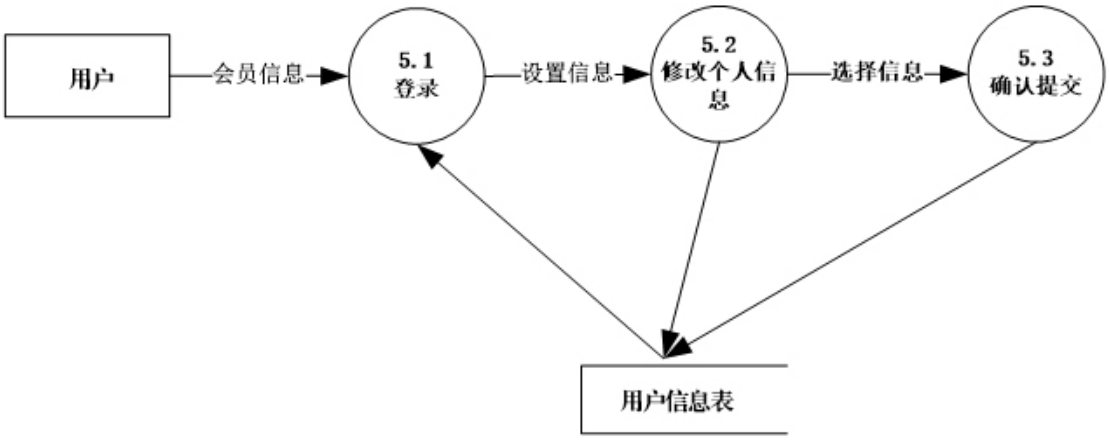

图 1-6　　个人信息设置数据流图

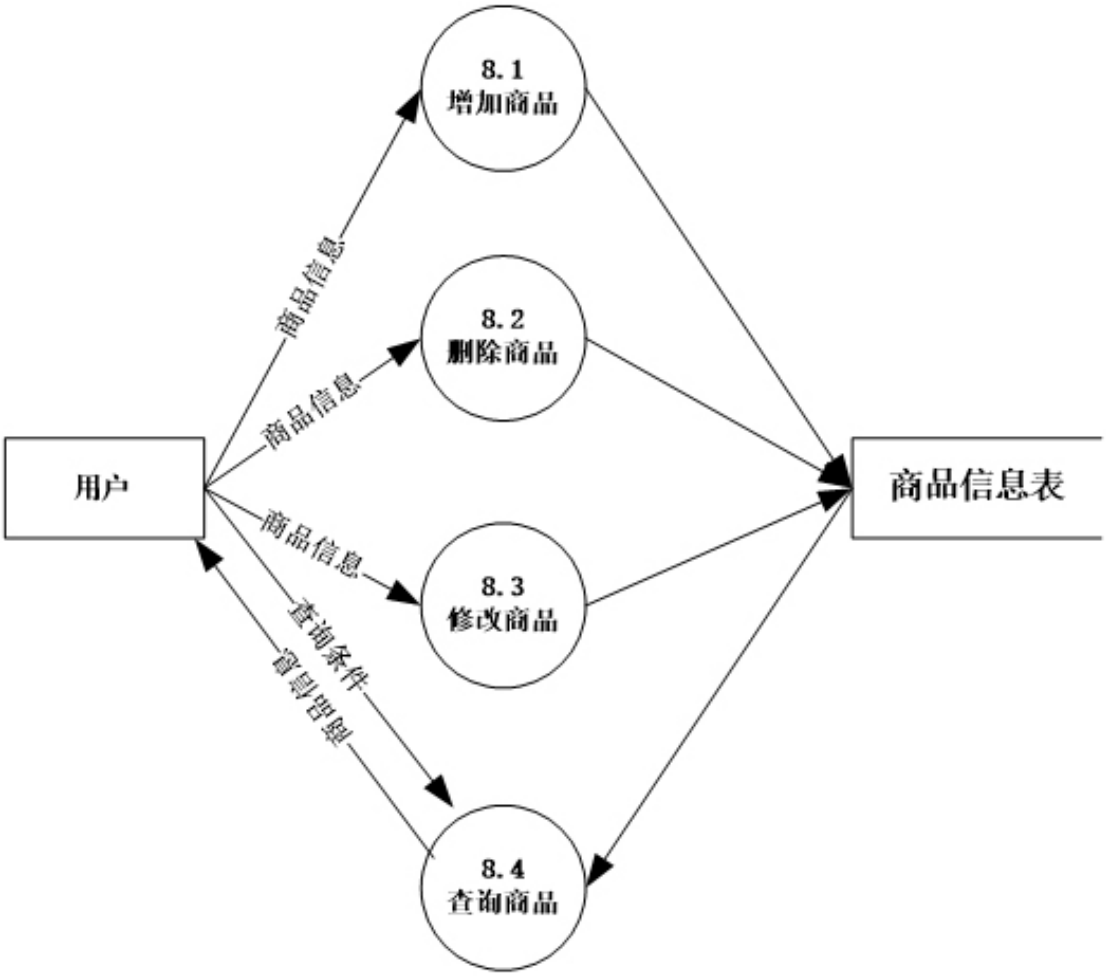

图 1-7　　商店管理数据流图

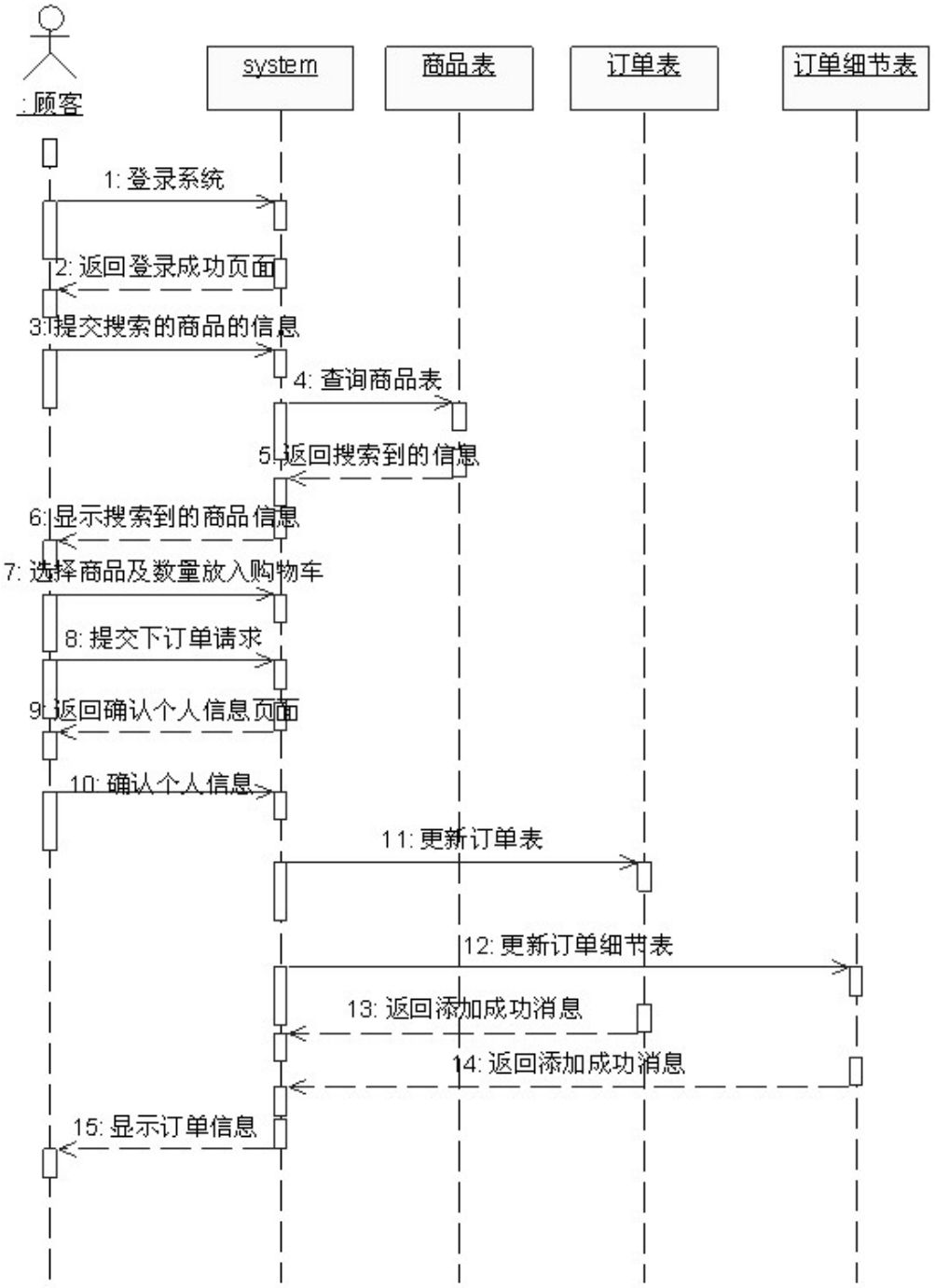
: 顾客
system
商品表
订单表
订单细节表
1: 登录系统
2: 返回登录成功页面
3: 提交搜索的商品的信息
4: 查询商品表
5: 返回搜索到的信息
6: 显示搜索到的商品信息
7: 选择商品及数量放入购物车
8: 提交下订单请求
9: 返回确认个人信息页面
10: 确认个人信息
11: 更新订单表
12: 更新订单细节表
13: 返回添加成功消息
14: 返回添加成功消息
15: 显示订单信息

图 1-8　　顾客下订单图

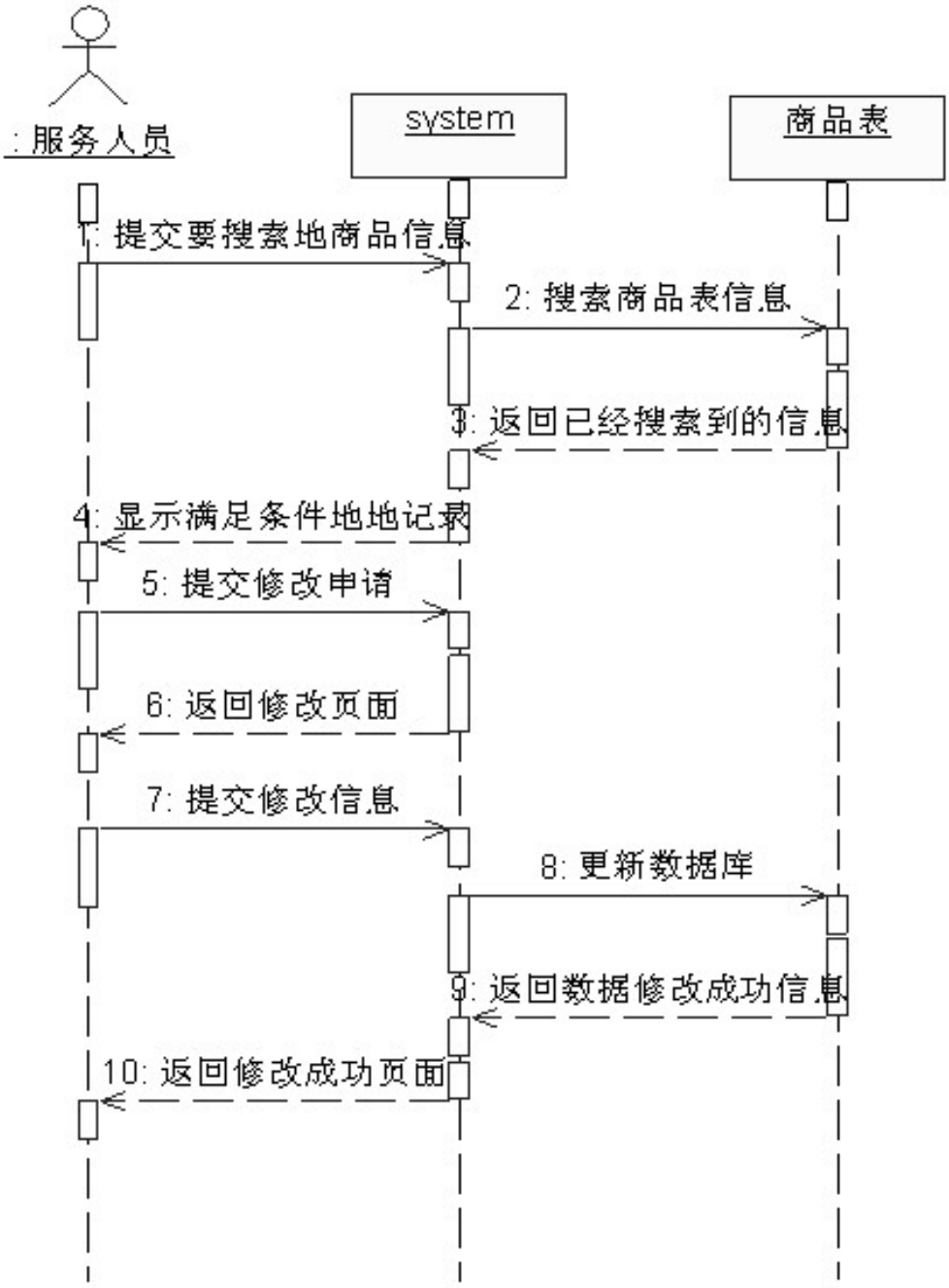
:服务人员
system
商品表
1: 提交要搜索地商品信息
2: 搜索商品表信息
3: 返回已经搜索到的信息
4: 显示满足条件地地记录
5: 提交修改申请
6: 返回修改页面
7: 提交修改信息
8: 更新数据库
9: 返回数据修改成功信息
10: 返回修改成功页面

图 1-9　　店铺管理图

第二章　软件设计文档

2.1　项目整体说明

本项目的名称为“北洋Wal-Mart”，定位是一个多平台适配的网上电子购物平台。本项目设计了微信小程序、安卓app、浏览器网页三个前端，设计了一个基于Django框架的Python后端，浏览器后端通过传统Django框架搭建，使用了MTV模式，而微信小程序和安卓app后台使用了基于Django REST Framework的API接口实现数据传输，也就是实现了前后端分离。另外本项目的服务器部署到了阿里云发行版LinuxCentOS7.4的服务器，三大前端通过公网即可使用该项目。

2.2　版本更新日志

2.2.1　版本V1

重构表结构，重建十张基本数据表

2.2.2　版本V2

新加入ItemSubCategory与Banner两张表，分别描述项目的子类与轮播图相关信息。

在Item中加入新字段item_sub_category表示项目所属子类。

在Item中加入新字段item_off表示商品降价打折信息。

在Item中加入新字段item_PrePrice表示商品打折前的原价信息。

更新排序检索API

2.2.3　版本V3

新增对字段的设计理念说明

更新Web API：加入根据用户user_id获取该用户购物车内的所有商品信息的接口

更新Web API：加入根据用户user_id获取该用户所关注的所有商店信息

更新Web API：加入根据用户user_id获取该用户所关注的所有商品信息

更新Web API：加入根据商品id，添加该商品的多张详情图片或返回该商品的所有详情图片路径

更新Web API：加入根据商品id，添加该商品的多张轮播图片或返回该商品的所有轮播图片路径

新加入SystemMessage表，描述系统消息，比如某某商品打折信息、发货信息等等系统消息。

新加入ChatMessage表，描述用户或组内聊天消息。

新加入Coupon表，描述优惠券信息。

新加入DailyOffItem表，描述每日特价商品信息

新加入ItemDetailImage表，存储商品图片信息

新加入ItemBannerImage表，存储商品轮播图信息

新加入Address表，存储用户收货地址信息

新加入Brand表，表示商品品牌

在Item中加入新字段item_sale表示某商品的销量。

在Item中加入新字段item_create_time表示商品上架时间。

在Item中加入新字段item_VIPOff表示该商品VIP特惠部分的价格。

在Item中加入新字段item_brand表示该商品的品牌

在User中加入新字段user_coupon表示用户所拥有的优惠券

在User中加入新字段user_TJUCarat描述用户所具有的“洋克拉“数。

在User中加入新字段user_isVIP描述该用户是否为会员。

在Trade中加入新字段trade_TJUCarat表示该笔交易获得的洋克拉。

在ShoppingCart中加入新字段cart_item_amount表示该商品的数量。

在ItemCategory中新加入item_category_isHaveSub字段，表示是否有子类

在ItemCategory中新加入item_category_own_sub字段，表示该大类拥有的子类id集合。

修改Item中的item_category和item_sub_category字段，均外键到对应表的标识id上，且为item_sub_category设置缺省值1

修改Item中的item_ image字段，改名为item_preview_image，表示该商品的单张预览图。

删除ItemSubCategory中的item_subCategory_belong字段

删除ItemCategory的item_category_own_sub字段

在ItemSubCategory中加入item_subCategory_belong字段

2.2.4 版本V4

修复了若干BUG

更新Web API：加入当前端浏览商品时自动更新浏览记录并合并记录、自动给商品浏览量+1、并返回该商品的信息的功能

更新Web API：加入当前端搜索某字符串时后端自动更新搜索记录并合并。

更新Web API：加入根据用户id返回该用户所有的优惠券信息

更新Web API：加入根据用户id、商品id直接删除收藏夹的记录

更新Web API：加入根据用户id返回该用户收藏夹中所有商品信息

更新Web API：加入根据商店id返回该商店的综合评价信息并更新该商店store_evaluate字段

更新Web API：加入根据用户id和商品id判断该用户是否收藏了该商品

更新Web API：加入根据评价id，可以添加该评价的多张图片或返回该评价的图片路径

更新Web API：加入根据商品id返回该商品有多少条评价信息

更新Web API：加入根据商品id返回该商品的综合评价信息

更新Web API：加入根据用户id和商店id返回该用户是否在该店购买过商品

更新Web API：加入根据商品id返回该商品被多少人收藏

更新Web API：加入根据商品id返回“推荐商品”的详细信息（智能推荐算法）

更新Web API：加入根据商品id返回该商品的好评数、中评数、差评数以及有图的评价数并返回评价信息

更新Web API：加入根据用户id返回待收货的商品信息以及数量

更新Web API：加入根据用户id返回待评价的商品信息以及数量

更新Web API：加入根据用户id返回该用户所有已完成的订单数及商品详情

更新Web API：加入根据用户id、商品id、商店id、购买数量，处理前端点击“立即购买”后的逻辑

更新Web API：加入根据用户id、商品id、购买数量，处理前端在购物车页面进行购买的逻辑

更新Web API：加入根据用户id、商品id、商店id、购买数量，处理前端点击加入购物车的逻辑

新加入Collection表，表示收藏夹。

新加入BrowseRecord表，表示浏览记录

新加入SearchRecord表，表示搜索记录

新加入Evaluate表，表示评价

新加入EvaluateImage表，表示评价的图片

在Store表中新加入字段store_evaluate表示商店的评价

在Banner表中新加入字段banner_belong

修改Trade表中的trade_evaluate字段，改为trade_isEvaluate表示该交易是否评价。

为User表中user_nickname增加unique属性

删除Banner表中的banner_clickUrl

更新Web API：完善了加入购物车的后端逻辑（增加了自动合并同类的功能）

更新Web API：新增根据trade_id进行确认收货的后端逻辑

更新Web API：新增根据user_id返回该用户浏览记录的商品详情（按时间降序）

更新Web API：新增根据user_id判断用户是否开店。若开店返回商店详情；若没开，返回数字0

更新Web API：新增根据user_id以及地址信息进行地址上传。若已存在该用户的地址则更新记录。

更新Web API：新增根据user_id上传用户的头像

更新Web API：新增根据item_id上传商品预览图

User表中新增user_password字段

在ItemCategory表中新增item_category_image字段

在Item表中新增item_trade_amount字段，表示商品交易数量

2.3 API文档

Web API提供基本的“增删改查“、排序、以及单独编写的API接口。

这些功能由前端发起Request不同Method的请求来实现。

例如：

“POST”增加数据

“PUT”修改数据

“DELETE”删除数据

“GET”查询数据

- 访问某张表的所有行信息:
  在网址后追加表名即可，注意大小写敏感。
  例如172.23.175.252:8000/api/user/访问user表，网址最后最好加上/防止错误。
- 访问某张表中某一行数据的信息:
  在网址后追加行表示id即可。
  例如172.23.175.252:8000/api/user/1/ 表示访问user表中id为1的行数据。
  另外如需删除或修改某行信息，需在类似172.23.175.252:8000/api/user/1/下进行修改，针对某一行数据的访问才具备DELETE与PUT的能力。对表访问只具备GET和POST的能力。
- 查询某张表中符合特定列条件的行信息:
  以user表为例，看一个例子： 172.23.175.252:8000/api/user/?user_id=1&user_nickname__icontains=&user_sex=&user_tel=&user_isAdmin=&user_location__icontains= 每个表支持搜索的字段都不同，模式都是这样。多字段联合搜索用&连接起来即可。例如访问id为2，昵称为xx的用户信息：

172.23.175.252:8000/api/user/?user_id=2&user_nickname__icontains=xx即可。 Icontains表示该字段是模糊搜索的，没有icontains标识的就是完全匹配搜索。注意icontains前是两个连续的下划线每个字段（除了外键）都可以同时提供模糊搜索和完全匹配两种搜索模式.

- 查询某张表中某列符合特定范围的行信息:
  有一些字段的含义可以进行范围搜索，比如某商品的价格，某店的开张日期。像这种字段都同时提供了精确匹配、范围搜索两种模式。
  以Store表查询开店日期为例： 精确搜索 172.23.175.252:8000/api/-store/?create_date=2019-08-22 日期精确等于2019-08-22 范围搜索 172.23.175.252:8000/api/store/?min_date=2019-08-21 日期大于2019-08-21 日期搜索必须给出精确到日的格式，比如：2019-08-06
- 排序:
  关键字段：ordering=xxx 例如：访问user表中所有行的信息，并以降序返回。 http://127.0.0.1:8000/api/user/?ordering=-user_id
- 单独提供的API接口：
  此处可以跳到软件测试文档查看

2.4 数据库表说明

共有二十五张表
所有表都会返回一个唯一标识一条数据的id
所有表的主键都是自增列
Tips：所有外键指向的字段均指向对应表的标识id字段。

User：用户表

User	类型	描述
User_id	自增列	用户 ID
User_password	字符串	用户密码
User_nickname	字符串	用户昵称
User_sex	字符串	用户性别
User_tel	字符串	用户手机号
User_headImage	可点击的 URL	用户头像
User_isAdmin	布尔	用户是否为管理员。系统预先内置一些管理员账户，前端提交用户注册表单时此项默认置 0.当用户登陆时，检查此项是否为 1，为 1 开启管理员权限。
User_isBindQQ	布尔	用户是否与 QQ 绑定
User_isVIP	布尔	用户是否为 VIP
User_location	字符串	用户位置
User_location_x	十进制小数	用户经度
User_location_y	十进制小数	用户纬度
User_coupon	多对多字段	表示用户拥有的优惠券的 id 的集合
User_TJUCarat	整数	“洋克拉”，类似于京东京豆的一种优惠机制。


```
{
    "user_id": 1,
    "user_nickname": "asd",
    "user_password": "123456",
    "user_sex": "男",
    "user_tel": "21321321",
    "user_headImage": "http://127.0.0.1:9000/media/user/headImage/5_2PsJtz6.jpg",
    "user_isAdmin": true,
    "user_isBindQQ": false,
    "user_location": "樱大",
    "user_location_x": "0.00000",
    "user_location_y": "0.00000",
    "user_TJUCarat": 0,
    "user_isVIP": false,
    "user_coupon": []
},
```


图 2-1　　user表

Store：商店表

Store	类型	描述
Store_id	自增列	商店 ID
Store_name	字符串	商店名
Store_user	外键：引用 User 中的 user_id，CASCADE。	店主的 user_id
Store_contact_info	字符串	商店联系信息
Store_info	字符串	商店简介
Store_create_time	标准年-月-日的日期格式	商店创建时间，此项在行记录创建时自动生成，后续对其他列记录的修改不影响此项。
Store_evaluate	十进制小数	商店的整体评价，10 代表未有评价信息


```
{
    "store_id": 1,
    "store_name": "123",
    "store_contact_info": "1231",
    "store_info": "231",
    "store_evaluate": "10.0",
    "store_create_time": "2019-08-23",
    "store_user": 1
},
```


图 2-2　　store表

ItemCategory：商品类表

ItemCategory	类型	描述
Item_category_id	自增列	大类 ID，唯一标识一个大类
Item_category	字符串	大类名
Item_category_isHaveSub	布尔	是否有子类
Item_category_image	可点击的 URL	大类图片


```
{
    "item_category_id": 1,
    "item_category": "IT开发",
    "item_category_isHaveSub": true,
    "item_category_image": "http://127.0.0.1:9000/media/category/category_image/itemDefault.jpg"
},
```


图 2-3　　itemcategory表

ItemSubCategory：商品子类表

ItemSubCategory	类型	描述
Item_subCategory_id	自增列	商品子类 ID
Item_subCategory	字符串	子类名
Item_subCategory_image	可点击的 URL	子类图片
Item_subCategory_belong	外键：指向 ItemCategory 的 item_category_id，SET_NULL	该子类属于哪个大类


```
{
    "item_subCategory_id": 1,
    "item_subCategory": "叮樘大",
    "item_subCategory_image": "http://127.0.0.1:9000/media/category/image/5.jpg",
    "item_subCategory_belong": null
}
```


图 2-4　　itemSubcategory表

Item：商品表

Item	类型	描述
Item_id	自增列	商品 ID
Item_name	字符串	商品名
Item_price	十进制小数	商品价格
Item_stoke	整数	商品库存
Item_preview_image	可点击的 URL	商品预览图片，可不上传，有缺省图片
Item_category	外键：引用 ItemCategory 中的 item_category_id，SET_NULL	商品大类
Item_sub_category	外键：引用 ItemSubCategory 中的 item_SubCategory_id，SET（1）	商品子类，缺省为 1
Item_browse	整数	商品浏览量
Item_store	外键：引用 Store 中的 store_id，CASCADE	商品所属的店铺
Item_off	字符串	商品打折降价的说明，比如"4 折"、"9 折"

Item_prePrice	十进制小数	商品原价
Item_create_time	标准日期格式	商品上架时间
Item_VIPOff	整数	VIP 特惠部分的价格，比如优惠 20 块
Item_sale	整数	商品销量
Item_brand	外键：引用 Brand 中的 brand_id，SET_NULL	商品品牌
Item_trade_amount	整数	特殊提供，其余时候无用。当返回交易商品详情时表示交易数量，方便前端取数据


```
{
    "item_id": 1,
    "item_name": "1",
    "item_price": "1.000",
    "item_stoke": 72,
    "item_preview_image": "http://127.0.0.1:9000/media/item/previewImage/5_F5q8h9T.jpg",
    "item_browse": 32,
    "item_off": "1",
    "item_prePrice": "1.000",
    "item_sale": 128,
    "item_create_time": "2019-08-25",
    "item_VIPOff": 10,
    "item_trade_amount": 0,
    "item_category": null,
    "item_sub_category": 1,
    "item_store": 1,
    "item_brand": null
},
```


图 2-5　　item表

ShoppingCart：购物车表，主要记录用户添加到购物车的商品信息

ShoppingCart	类型	描述
Cart_id	自增列	购物车 ID
Cart_user	外键：引用 User 中的 user_id，CASCADE	描述改行记录的购物车添加记录是由哪个用户发起的
Cart_item	外键：引用 Item 中的 item_id，CASCADE	描述加入购物车的商品
Cart_item_amount	整数	加入购物车的某商品的数量


```
{
    "cart_id": 1,
    "cart_item_amount": 1,
    "cart_user": 1,
    "cart_item": 1
},
```


图 2-6　　shoppingcart表

Trade：交易表，描述用户下单的商品的相关信息。

Trade	类型	描述
Trade_id	自增列	交易 ID
Trade_user	外键：引用 User 中的 user_id，CASCADE	描述改交易由哪个用户发起
Trade_item	外键：引用 Item 中的 item_id，CASCADE	描述交易的商品
Trade_amount	整数	描述交易商品的数量
Trade_store	外键：引用 Store 中的 store_id，CASCADE	描述该交易是在哪个店铺发起的
Trade_isEvaluate	布尔	表示该次交易是否评价
Trade_create_time	标准日期格式	当行记录创建时自动生成时间，后每次对行数据修改不会对其产生影响。
Trade_finish_time	标准日期格式	每次修改该行任意数据都会自动变更为最后修改时间。
Trade_info	整数	暂定表示发货信息。0 代表未发货，1 代表正在运输中，2 代表已到达目的地，3 代表已确认收货，交易完成


```
{
    "trade_id": 1,
    "trade_amount": 1,
    "trade_isEvaluate": false,
    "trade_create_time": "2019-08-26",
    "trade_finish_time": "2019-08-26",
    "trade_info": 3,
    "trade_TJUCarat": 10,
    "trade_user": 1,
    "trade_item": 1,
    "trade_store": 1
}
```


图 2-7　　trade表

Group：组表

Group	类型	描述
Group_id	自增列	组 ID
Group_name	字符串	组名
Group_info	字符串	组简介
Group_create_time	标准日期格式	组创建时间，首次创建自动生成时间，后续修改不影响。


```
{
    "group_id": 1,
    "group_name": "1",
    "group_info": "dasdhi",
    "group_create_time": "2019-08-22"
}
```


图 2-8　　group表

StoreFollow：商店关注表，该表表示用户关注商店的信息

StoreFollow	类型	描述
Follow_id	自增列	关注 ID
Follow_user	外键：引用 User 中的 user_id，CASCADE	表示关注发起用户的 id
Follow_store	外键：引用 Store 中的 store_id，CASCADE	表示用户关注的店 id

图 2-9　　storefollow表

ItemFollow：商品关注表，表示用户关注商品的信息

ItemFollow	类型	描述
Follow_id	自增列	关注 ID
Follow_user	外键：引用 User 中的 user_id，CASCADE	表示关注发起用户的 id
Follow_item	外键：引用 Item 中的 item_id，CASCADE	表示用户关注的商品 id

图 2-10　　itemfollow表

GroupFollow：组关注表，表示用户所在群组的信息

GroupFollow	类型	描述
Follow_id	自增列	关注 ID
Follow_group	外键：引用 Group 中的 group_id，CASCADE	表示用户所处的组的 id
Follow_user	外键：引用 User 中的 user_id，CASCADE	表示是哪个用户处于某组


```
{
    "follow_id": 1,
    "follow_group": 1,
    "follow_user": 1
}
```


图 2-11　　groupfollow表

Banner：轮播图表

Banner	类型	描述
Banner_id	自增列	轮播图 ID
Banner_number	整数	轮播图序号
Banner_image	可点击的 URL	轮播图的访问路径
Banner_belong	外键：引用 Item 中的 item_id，CASCADE	该轮播图属于哪个 item


```
{
    "banner_id": 1,
    "banner_number": 1,
    "banner_image": "http://127.0.0.1:9000/media/banner/image/5.jpg",
    "banner_belong": 1
}
```


图 2-12　　banner表

SystemMessage：系统消息

SystemMessage	类型	描述
Message_id	自增列	消息 ID
Message_content	字符串	消息内容
Message_type	整数	消息类型。暂定 0 代表系统消息，优先级最高；1 代表一些优惠促销消息等等,2 代表交易物流信息等等.....
Message_create_time	标准日期格式	消息创建时间，首次创建自动生成时间，后续修改不影响。


```
{
    "message_id": 1,
    "message_content": "惊喜！大降价",
    "message_type": 1,
    "message_create_time": "2019-08-23"
}
```


图 2-13　　systemmessage表

ChatMessage：聊天消息表

ChatMessage	类型	描述
Message_id	自增列	消息 ID
Message_type	整数	消息类型。暂定 0 代表用户之间的消息记录，1 代表组内消息记录
Message_ori	外键：引用 User 中的 user_id，SET_NULL	消息发起者的 user_id
Message_rec	外键：引用 User 中的 user_id，SET_NULL	消息接收者的 user_id
Message_create_time	标准日期格式	消息创建时间，首次创建自动生成时间，后续修改不影响。
Message_content	字符串	消息内容
Message_group	外键：引用 Group 中的 group_id，SET_NULL	该消息属于哪个群组


```
{
    "message_id": 1,
    "message_type": 0,
    "message_create_time": "2019-08-23",
    "message_content": "你好",
    "message_ori": 1,
    "message_rec": 2,
    "message_group": null
}
```


图 2-14　　chatmessage表

Coupon：优惠券表

Coupon	类型	描述
Coupon_id	自增列	优惠券 ID
Coupon_off	整数	优惠券优惠的价格
Coupon_require	整数	使用优惠券需达到的总购买价
Coupon_category	外键：引用 ItemCategory 中的 item_category_id，CASCADE	优惠券所适用的商品大类。与上两个字段合起来表示：该优惠券满 XX 元减 XX 元，只能购买 XX 类的商品。
Coupon_last	DurationField：表示一段时间，具体参考 api 网站	优惠券的有效期，该字段表示一段时间，并非日期。比如表示 1 小时。可精确到秒。


```
{
    "coupon_id": 1,
    "coupon_off": 10,
    "coupon_require": 100,
    "coupon_last": "00:00:30",
    "coupon_category": 1
},
```


图 2-15　　coupon表

DailyOffItem：每日特价商品表

DailyOffItem	类型	描述
Item_id	自增列	特价 ID
Item_off_id	外键：引用 Item 中的 item_id，CASCADE	特价商品的 item_id
Item_off_create_time	标准日期格式	该特价商品的创建时间

Tips：该表可每天由后端人员负责修改，前端只需适配好数据格式和 UI 界面即可。


```
{
    "item_id": 1,
    "item_off_create_time": "2019-08-23",
    "item_off_id": 1
}
```


图 2-16　　dailyoffitem表

ItemDetailImage：商品详情图片表

ItemDetailImage	类型	描述
Image_id	自增列	详情图 ID
Image_url	可点击 URL	详情图
Image_belong	外键：引用 Item 中的 item_id，CASCADE	该图属于哪个商品


```
{
    "image_id": 2,
    "image_url": "http://127.0.0.1:9000/media/item/detailImage/5.jpg",
    "image_belong": 3
},
```


图 2-17　　itemdetailimage表

ItemBannerImage：商品轮播图片表

ItemBannerImage	类型	描述
Image_id	自增列	轮播图 ID
Image_url	可点击 URL	轮播图
Image_belong	外键：引用 Item 中的 item_id，CASCADE	该图属于哪个商品


```
{
    "image_id": 2,
    "image_url": "http://127.0.0.1:9000/media/item/bannerImage/5.jpg",
    "image_belong": 2
},
```


Tips：该表在 Web API 部分有专门的接口来实现上传图片和查询图片

图 2-18　　itembannerimage表

EvaluateImage：评价图片表

EvaluateImage	类型	描述
Image_id	自增列	评价图 ID
Image_url	可点击 URL	评价图
Image_belong	外键：引用 Evaluate 中的 evaluate_id，CASCADE	该图属于哪个评价


```
{
    "image_id": 1,
    "image_url": "http://127.0.0.1:9000/media/evaluate/image/5.jpg",
    "image_belong": 1
}
```


图 2-19　　evaluateimage表

Address：用户收货地址表

Address	类型	描述
Address_id	自增列	地址 ID
Address_username	字符串	用户的收货名字
Address_tel	字符串	用户的收货手机
Address_detail	字符串	用户的收获详细地址
Address_type	字符串	用户的收货地址类型，比如学校、家庭等等
Address_isDefault	布尔	该地址是否为默认地址
Address_user	外键：引用 User 中的 user_id 字段，CASCADE	该地址所属的用户


```
{
    "address_id": 1,
    "address_username": "小明",
    "address_tel": "17333333",
    "address_detail": "天津",
    "address_type": "学校",
    "address_isDefault": true,
    "address_user": 1
}
```


图 2-20　　address表

Brand：品牌表

Brand	类型	描述
Brand_id	自增列	品牌 ID
Brand_name	字符串	品牌名


```
{
    "brand_id": 1,
    "brand_name": "吊牌"
}
```


图 2-21　　brand表

Collection：收藏夹表

Collection	类型	描述
Collection_id	自增列	收藏夹记录 ID
Collection_user	外键：引用 User 中的 user_id，CASCADE	该记录所属用户
Collection_item	外键：引用 Item 中的 item_id，CASCADE	关注的商品


```
{
    "collection_id": 1,
    "collection_user": 1,
    "collection_item": 1
}
```


图 2-22　　collection表

BrowseRecord：浏览记录

BrowseRecord	类型	描述
Record_id	自增列	记录 ID
Record_user	外键：引用 User 中的 user_id，CASCADE	产生该浏览记录的用户
Record_item	外键：引用 Item 中的 item_id，CASCADE	浏览的商品
Record_flag	整数	记录修改标志位，前端无需了解该字段作用，只需在浏览商品时返回后端用户 id 和商品 id 即可，后端已封装好 API。
Record_create_time	标准日期+时间格式	记录的创建时间，精确到毫秒级


```
{
    "record_id": 4,
    "record_flag": 0,
    "record_create_time": "2019-08-25T14:08:01.497248Z",
    "record_user": 1,
    "record_item": 1
}
```


图 2-23　　browseRecord表

SearchRecord：搜索记录

SearchRecord	类型	描述
Record_id	自增列	记录 ID
Record_user	外键： 引用 User 中的 user_id， CASCADE	产生该搜索记录的用户
Record_content	字符串	搜索的内容
Record_flag	整数	记录修改标志位，前端无需了解该字段作用，只需在搜索商品时返回后端用户 id 和搜索内容即可，后端已封装好 API。
Record_create_time	标准日期+时间格式	记录的创建时间，精确到毫秒级


```
{
    "record_id": 1,
    "record_content": "十大阿三",
    "record_flag": 0,
    "record_create_time": "2019-08-25T15:12:43.978900Z",
    "record_user": 1
}
```


图 2-24　　searchrecord表

Evaluate：评价表

Evaluate	类型	描述
Evaluate_id	自增列	评价 ID
Evaluate_user	外键：引用 User 中的 user_id，CASCADE	评价者
Evaluate_item	外键：引用 Item 中的 item_id，CASCADE	评价的商品
Evaluate_TJUCarat	整数	完成该次评价获得的洋克拉数，默认为 10
Evaluate_isAnonymous	布尔	评价是否匿名
Evaluate_content	字符串	评价内容
Evaluate_item_score	整数	评价商品的分数，默认为 5，表示满评，下同
Evaluate_express_package	整数	评价的快递包裹分数
Evaluate_express_speed	整数	评价的快递速度分数
Evaluate_express_service	整数	评价的快递服务分数
Evaluate_create_time	标准日期格式	评价时间


```
{
    "evaluate_id": 1,
    "evaluate_TJUCarat": 10,
    "evaluate_isAnonymous": true,
    "evaluate_content": "sdasdsa",
    "evaluate_item_score": 5,
    "evaluate_express_package": 5,
    "evaluate_express_speed": 5,
    "evaluate_express_service": 5,
    "evaluate_create_time": "2019-08-26",
    "evaluate_user": 1,
    "evaluate_item": 1
}
```


图 2-25　　evaluate表

第三章　　软件测试文档

3.1　说明

需要1个参数的请求，只需传单个值如api/1这样即可。如果有两个或两个以上的参数，那么需要按键值对的方式给出具体参数以避免调用顺序错误，如api/?id=1&name=x这种方式，参数是可以乱序的，而用第一种方式参数必须定序。

以下均为单独提供的API接口。

3.2　测试功能及结果

- 根据用户user_id获取该用户购物车内的所有商品信息

  接口：http://127.0.0.1:8000/api/get_user_cart_item/pk

  网址最后的参数pk是用户的user_id。此接口只允许GET请求。

  示例：获取用户id为3的购物车商品信息：

  http://127.0.0.1:8000/api/get_user_cart_item/3

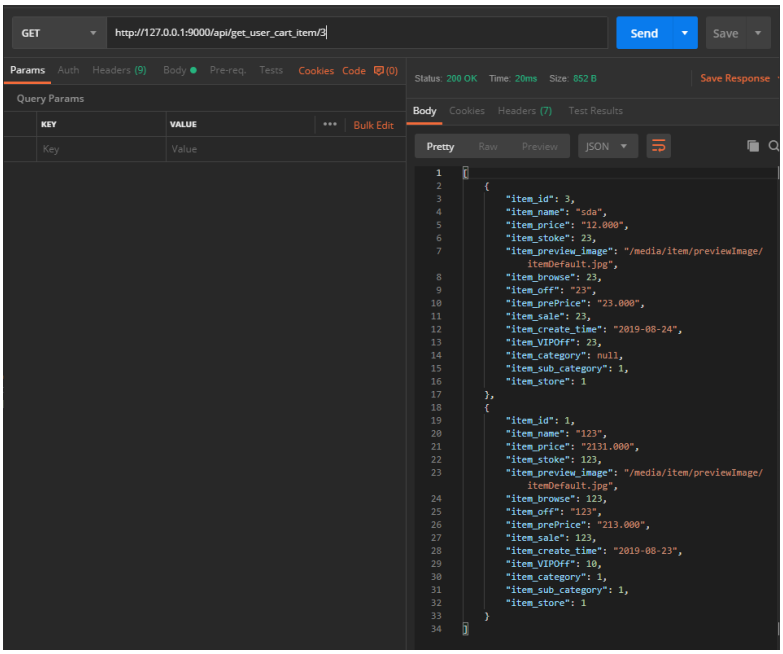

图 3-1

- 根据用户user_id获取该用户关注的所有商店的信息

  接口：http://127.0.0.1:8000/api/get_user_store_follow/pk

  网址最后的参数pk是用户的user_id。此接口只允许GET请求。

示例：获取用户id为3的关注商店信息

http://127.0.0.1:8000/api/get_user_store_follow/3

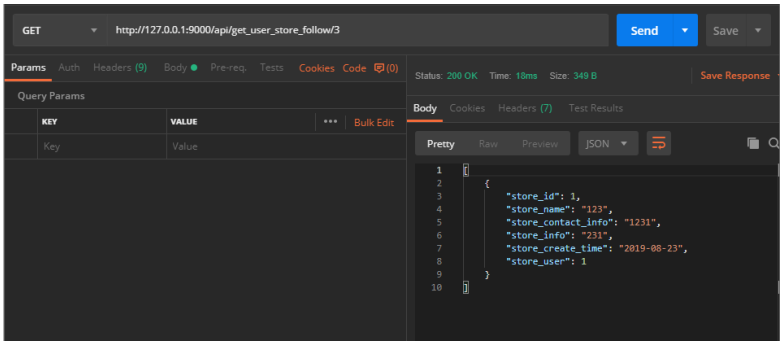

图 3-2

- 根据用户user_id获取该用户关注的所有商品的信息

  接口：http://127.0.0.1:8000/api/get_user_item_follow/pk

  网址最后的参数pk是用户的user_id。此接口只允许GET请求。

  示例：获取用户id为1的关注商品信息

  http://127.0.0.1:8000/api/get_user_item_follow/1

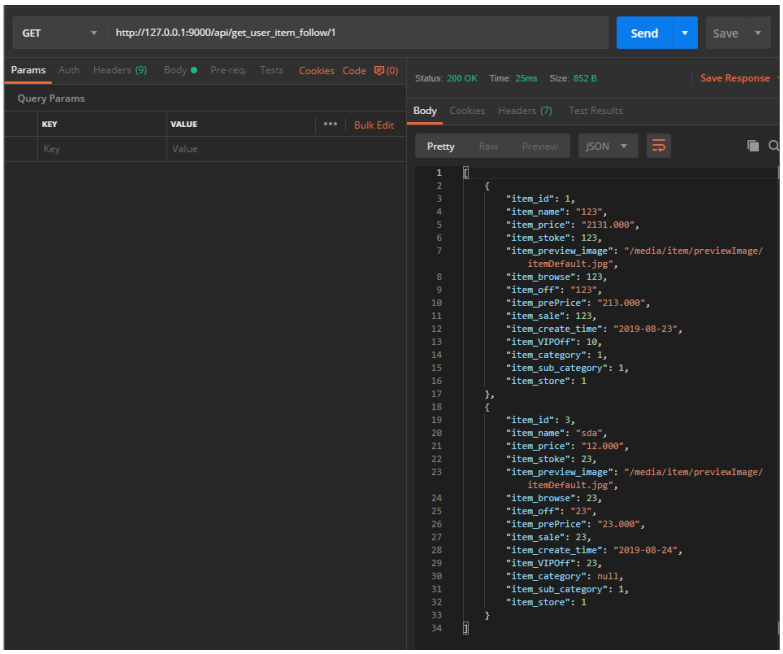

图 3-3

- 根据user_id上传用户头像

  接口：http://127.0.0.1:8000/api/upload_user_head_image/pk

网址最后的参数pk是用户的user_id。此接口只允许POST请求。
上传图片时POST参数name需设为img，这样接口才能取到文件数据。
后端获取文件列表，根据前端name参数来获取。
示例：上传user_id为1的用户的头像
http://127.0.0.1:8000/api/upload_user_head_image/1

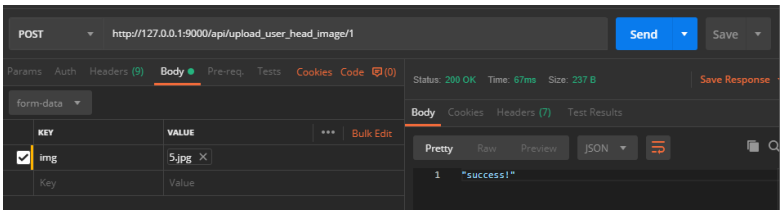

图 3-4

- 根据item_id上传商品预览图
  接口：http://127.0.0.1:8000/api/upload_item_preview_image/pk
  网址最后的参数pk是商品的item_id。此接口只允许POST请求。
  上传图片时POST参数name需设为img，这样接口才能取到文件数据。
  后端获取文件列表，根据前端name参数来获取。
  示例：上传item_id为1的商品的预览图
  http://127.0.0.1:8000/api/upload_item_preview_image/1

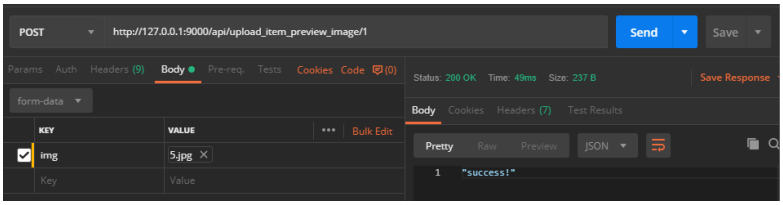

图 3-5

- 根据item_id，添加该商品的详情图或返回该商品的详情图路径
  接口：http://127.0.0.1:8000/api/item_detail_image/pk
  网址最后的参数pk是商品的item_id。此接口允许GET和POST请求。
  上传图片和查询图片使用同一个接口，支持同时上传多图。
  上传图片发POST请求，查询图片发GET请求。
  上传图片时POST参数name需设为img，这样接口才能取到文件数据。
  后端获取文件列表，根据前端name参数来获取。
  示例：为id为5的商品上传多张图片
  http://127.0.0.1:8000/api/item_detail_image/5

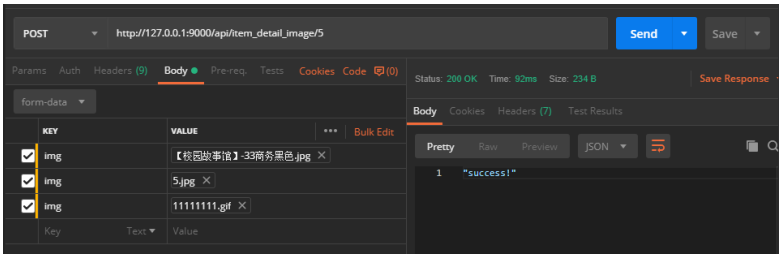

图 3-6

- 示例：查询id为5的商品的所有详情图
  http://127.0.0.1:8000/api/item_detail_image/5

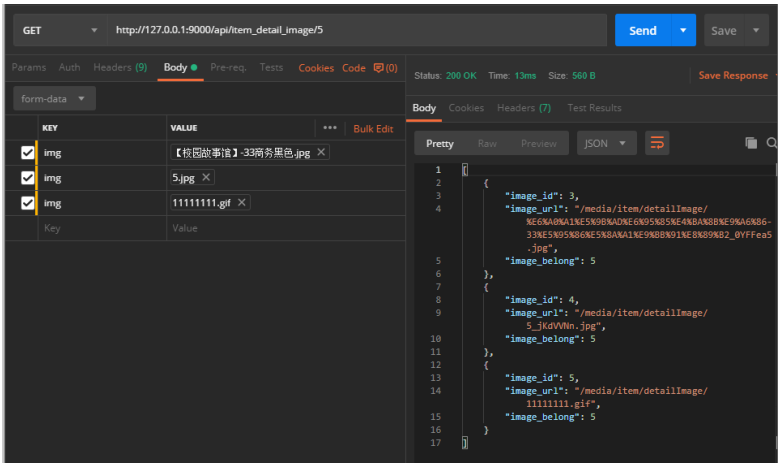

图 3-7

- 根据item_id，添加该商品的轮播图或返回该商品的轮播图路径
  接口：http://127.0.0.1:8000/api/item_banner_image/pk
  网址最后的参数pk是商品的item_id。此接口允许GET和POST请求。
  上传图片和查询图片使用同一个接口，支持同时上传多图。
  上传图片发POST请求，查询图片发GET请求。
  上传图片时POST参数name需设为img，这样接口才能取到文件数据。
  后端获取文件列表，根据前端name参数来获取。
  示例：为id为5的商品上传多张图片
  http://127.0.0.1:8000/api/item_banner_image/5
- 根据评价id，添加该评价的多张图片或返回该评价的图片路径
  接口：http://127.0.0.1:8000/api/evaluate_image/pk
  网址最后的参数pk是评价的evaluate_id。此接口允许GET和POST请求。
  上传图片和查询图片使用同一个接口，支持同时上传多图。

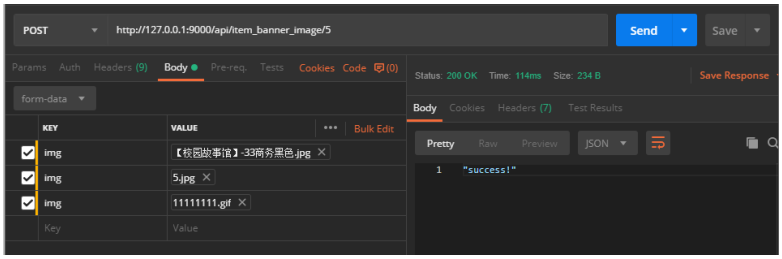

图 3-8

上传图片发POST请求，查询图片发GET请求。

上传图片时POST参数name需设为img，这样接口才能取到文件数据。

后端获取文件列表，根据前端name参数来获取。

示例：为id为1的评价上传多张图片

http://127.0.0.1:8000/api/evaluate_image/1

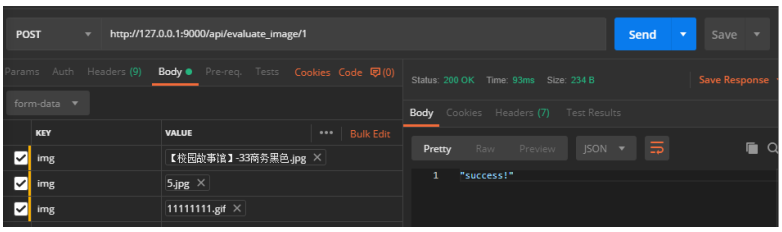

图 3-9

- 示例：查询id为1的评价的所有评价图

  http://127.0.0.1:8000/api/evaluate_image/1

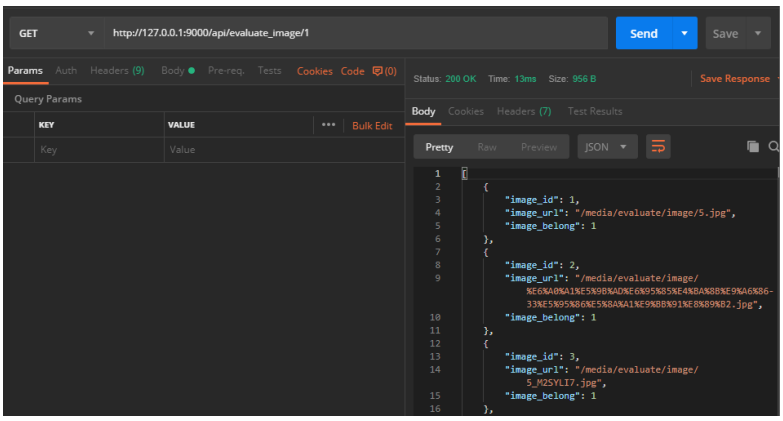

图 3-10

- 根据user_id，返回该用户拥有的优惠券信息

  接口：http://127.0.0.1:8000/api/get_user_coupon/pk

  网址最后的参数pk是用户的user_id。此接口只允许GET请求。

  示例：获取用户id为1的所有优惠券信息：

  http://127.0.0.1:8000/api/get_user_coupon/1

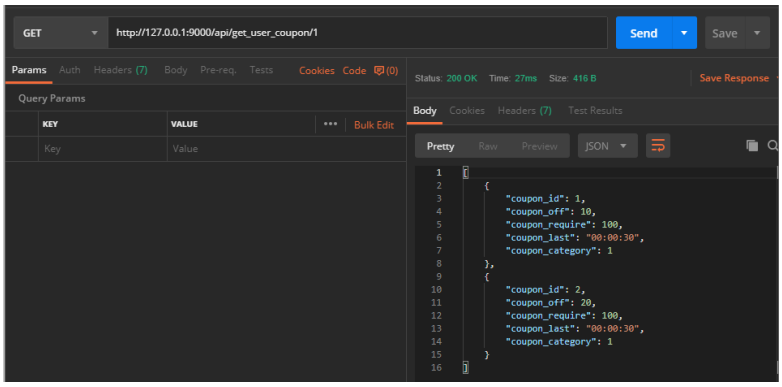

图 3-11

- 根据user_id、item_id删除用户收藏夹的信息

  接口：http://127.0.0.1:8000/api/delete_user_collection/?item_id=x&user_id=x

  访问接口的基本格式如上，只支持DELETE方法。

  Item_id和user_id的位置可以任意互换，只需保证变量名对应就好。

  示例：删除用户id为1，商品id为1的收藏夹信息：

  http://127.0.0.1:8000/api/delete_user_collection/?item_id=1&user_id=1

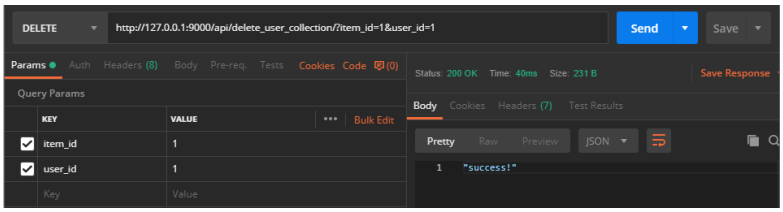

图 3-12

- 根据用户id和商品id判断该用户是否收藏了该商品

  接口：

  http://127.0.0.1:8000/api/whether_user_collect_item/?item_id=x&user_id=x

  基本请求格式如上所示，参数顺序可随意，但参数名一定要匹配。

  此接口只允许GET请求。

  示例：

判断用户id为1的是否收藏了商品id为1的

http://127.0.0.1:8000/api/whether_user_collect_item/?item_id=1&user_id=1

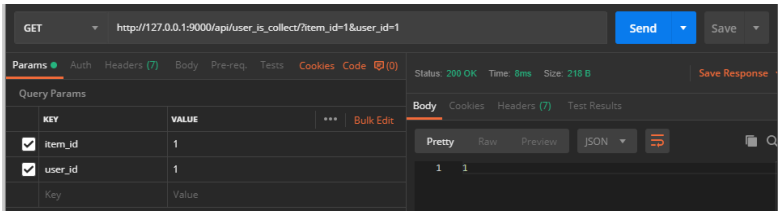

图 3-13

- 示例：判断用户id为2的是否收藏了商品id为1的

  http://127.0.0.1:8000/api/whether_user_collect_item/?item_id=1&user_id=2

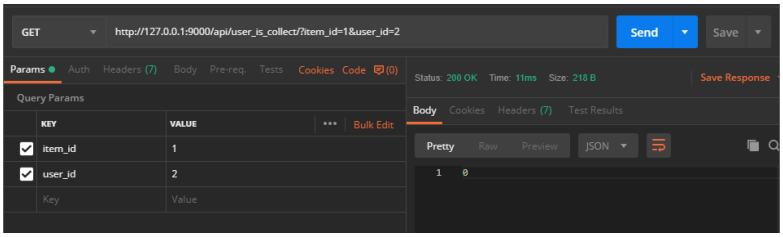

图 3-14

- 根据user_id返回用户收藏夹的所有商品信息

  接口：http://127.0.0.1:8000/api/get_user_collection/pk

  网址最后的参数pk是用户的user_id。此接口只允许GET请求。

  示例：返回用户id为1的收藏夹商品全部信息：

  http://127.0.0.1:8000/api/get_user_collection/1

- 根据user_id、item_id，当前端浏览商品时自动更新浏览记录并合并记录、自动给商品浏览量+1、并返回该商品的信息

  接口：http://127.0.0.1:8000/api/browse_item/?user_id=1&item_id=2

  基本请求格式如上所示，各个参数可以乱序。

  此接口只允许GET请求。

  示例：以id为1的用户浏览商品id为3为例

  浏览一个曾经未浏览过的商品：

  http://127.0.0.1:8000/api/browse_item/?user_id=1&item_id=3

- 示例：浏览一个曾经浏览过的商品：

  http://127.0.0.1:8000/api/browse_item/?user_id=1&item_id=3

- 根据用户id和搜索内容更新搜索记录并合并

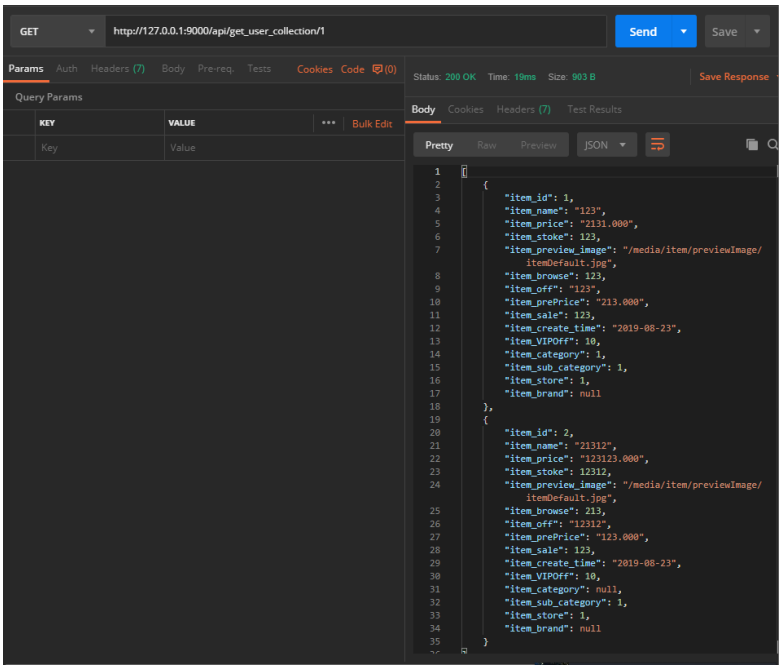
GET
http://127.0.0.1:9000/api/get_user_collection/1
Send
Save
Params Auth Headers (7) Body Pre-req. Tests Cookies Code (0)
Query Params
KEY VALUE Bulk Edit
Key Value
Status: 200 OK Time: 19ms Size: 903 B
Save Response
Body Cookies Headers (7) Test Results
Pretty Raw Preview JSON
[
{
"item_id": 1,
"item_name": "123",
"item_price": "2131.000",
"item_stoke": 123,
"item_preview_image": "/media/item/previewImage/itemDefault.jpg",
"item_browse": 123,
"item_off": "123",
"item_prePrice": "213.000",
"item_sale": 123,
"item_create_time": "2019-08-23",
"item_VIPOff": 10,
"item_category": 1,
"item_sub_category": 1,
"item_store": 1,
"item_brand": null
},
{
"item_id": 2,
"item_name": "21312",
"item_price": "123123.000",
"item_stoke": 12312,
"item_preview_image": "/media/item/previewImage/itemDefault.jpg",
"item_browse": 213,
"item_off": "12312",
"item_prePrice": "123.000",
"item_sale": 123,
"item_create_time": "2019-08-23",
"item_VIPOff": 10,
"item_category": null,
"item_sub_category": 1,
"item_store": 1,
"item_brand": null
}

图 3-15

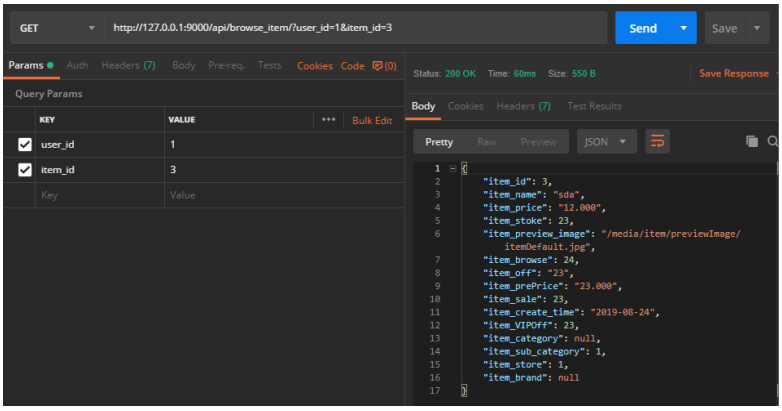
GET
http://127.0.0.1:9000/api/browse_item/?user_id=1&item_id=3
Send
Save
Params Auth Headers (7) Body Pre-req. Tests Cookies Code (0)
Query Params
KEY VALUE Bulk Edit
user_id 1
item_id 3
Key Value
Status: 200 OK Time: 60ms Size: 550 B
Save Response
Body Cookies Headers (7) Test Results
Pretty Raw Preview JSON
{
"item_id": 3,
"item_name": "sda",
"item_price": "12.000",
"item_stoke": 23,
"item_preview_image": "/media/item/previewImage/itemDefault.jpg",
"item_browse": 24,
"item_off": "23",
"item_prePrice": "23.000",
"item_sale": 23,
"item_create_time": "2019-08-24",
"item_VIPOff": 23,
"item_category": null,
"item_sub_category": 1,
"item_store": 1,
"item_brand": null
}

图 3-16

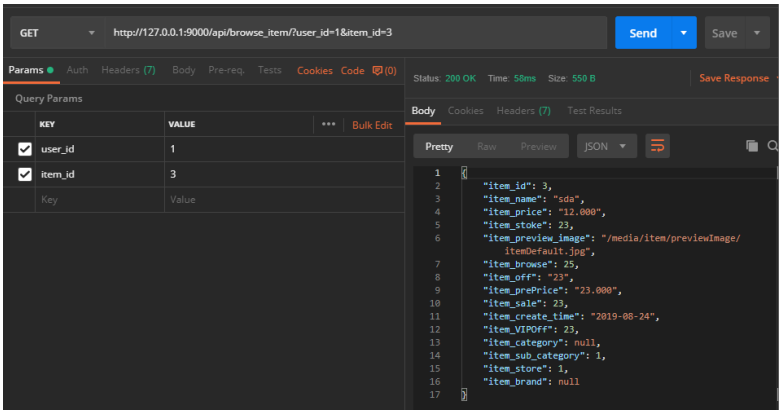

图 3-17

接口：http://127.0.0.1:8000/api/search_content/pk/?content=xx

网址中的pk表示用户id。此接口只允许POST请求。

示例：用户id为2的人搜索内容为测试的字符串：

第一次搜索：

http://127.0.0.1:8000/api/search_content/2/?content=测试

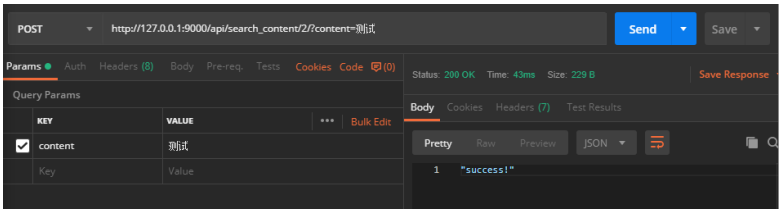

图 3-18

- 示例：隔一段时间，第二次再搜索同样的内容：
  http://127.0.0.1:8000/api/search_content/2/?content=测试
  每次访问某商品，只要向这个接口发送请求，record_create_time就会自动更新到当前请求的时间，前端就可以利用原生接口提供的排序功能来构造历史记录列表。
- 根据商店id返回该商店的综合评价信息并更新该商店store_evaluate字段
  接口：http://127.0.0.1:8000/api/get_store_evaluate/pk
  网址最后的参数pk是用户的user_id。此接口只允许GET请求。
  示例：获取商店id为1的商店综合评价
  http://127.0.0.1:8000/api/get_store_evaluate/1
- 根据商品id返回该商品有多少条评价信息
  接口：http://127.0.0.1:8000/api/get_item_evaluate_amount/pk

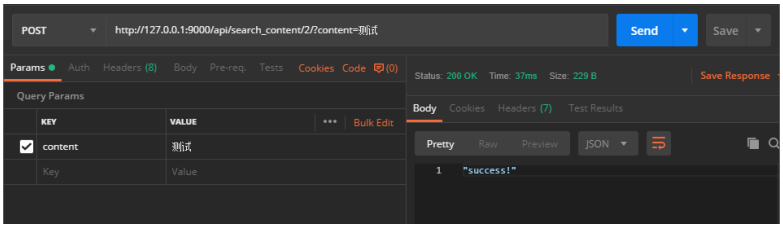

图 3-19

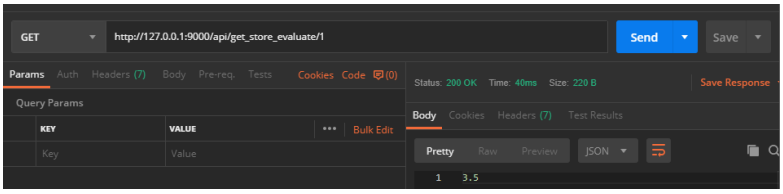

图 3-20

网址最后的参数pk是商品的item_id。此接口只允许GET请求。

示例：获取商品id为1的评价数

http://127.0.0.1:8000/api/get_item_evaluate_amount/1

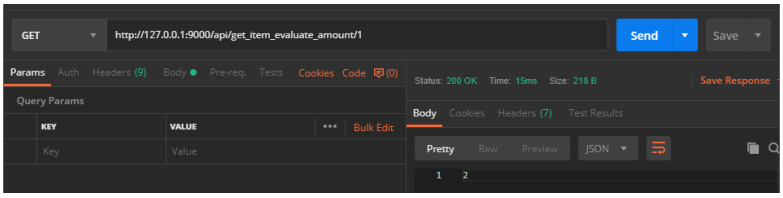

图 3-21

- 根据商品id返回该商品的综合评价信息。

  默认以好评率字符串返回，例如：98

  接口：http://127.0.0.1:8000/api/get_item_evaluate_info/pk

  网址最后的参数pk是商品的item_id。此接口只允许GET请求。

  示例：获取id为1的商品的综合评价

  http://127.0.0.1:8000/api/get_item_evaluate_info/1
- 若该商品未有评价信息，则返回“暂无评价”
- 根据用户id和商店id返回该用户是否在该店购买过商品

  接口：

  http://127.0.0.1:8000/api/whether_user_buy_item_in_store/?user_id=x&store_id=x

  基本请求格式如上，参数可乱序但参数名一定要对应，只允许GET

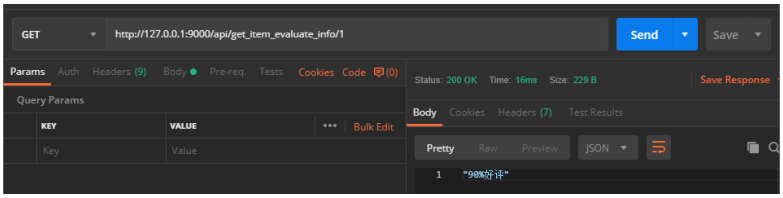

图 3-22

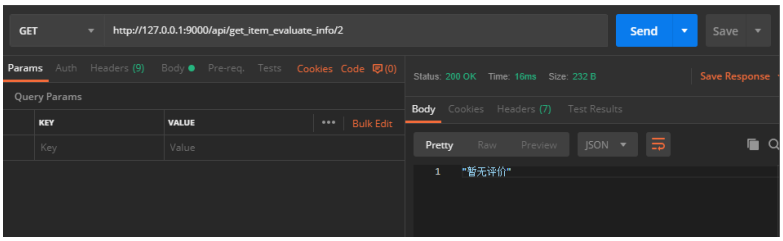

图 3-23

返回0代表未买过，返回1代表买过

此接口主要根据Trade表中是否有数据以及trade_info进行判断，当trade表中有数据且有至少一条记录的trade_info为3（3代表已完成交易）时，返回1。否则返回0

示例：用户1是否在商店1买过商品

http://127.0.0.1:8000/api/whether_user_buy_item_in_store/?user_id=1&store_id=1

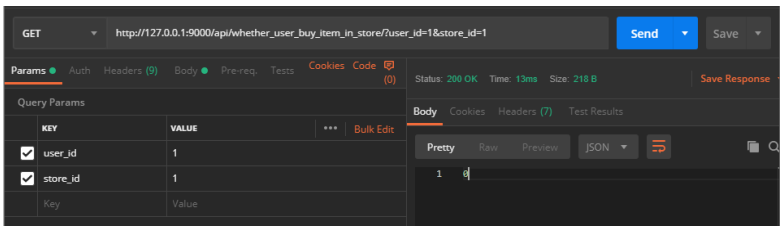

图 3-24

- 根据商品id返回该商品被多少人收藏

  接口：http://127.0.0.1:8000/api/get_item_collection_amount/pk

  网址最后的参数pk是商品的item_id。此接口只允许GET请求。

  示例：获取id为1的商品收藏数

  http://127.0.0.1:8000/api/get_item_collection_amount/1

- 根据商品id返回“推荐商品”的详细信息（智能推荐算法）

  接口：http://127.0.0.1:8000/api/get_recommend_item/pk

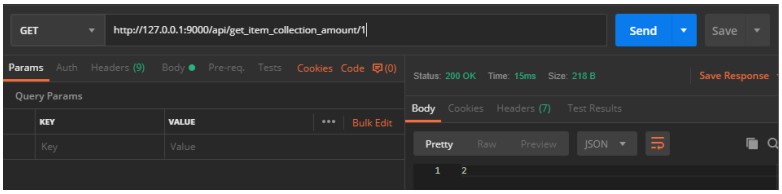

图 3-25

网址最后的参数pk是商品的item_id。此接口只允许GET请求。

后端内部是先筛选出与商品属于同一子类的所有商品，然后根据商品名字的相似度进行推荐

示例：获取id为3的商品的推荐商品

http://127.0.0.1:8000/api/get_recommend_item/3

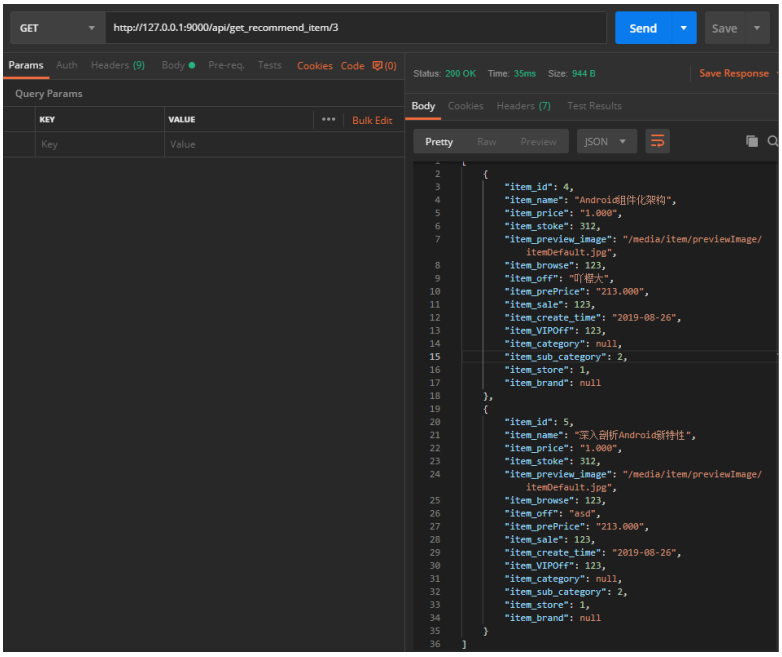

图 3-26

- 根据商品id返回该商品的好评数、中评数、差评数和有图的评价数并返回评价信息

  接口：

  http://127.0.0.1:8000/api/get_evaluate_type/?item_id=x&evaluate_type=xxxx&return_type=xxxx

  基本请求格式如上所示，参数可以乱序但参数名一定要对应。

  此接口只允许GET

  Evaluate_type表示评价类型，good代表好评、normal代表中评、bad代表差评、image代表有图

Return_type代表返回类型，amount表示返回数字、info表示返回具体信息

示例：获取id为1的商品的有图评价数

http://127.0.0.1:8000/api/get_evaluate_type/?item_id=1&evaluate_type=image&return_type=amou

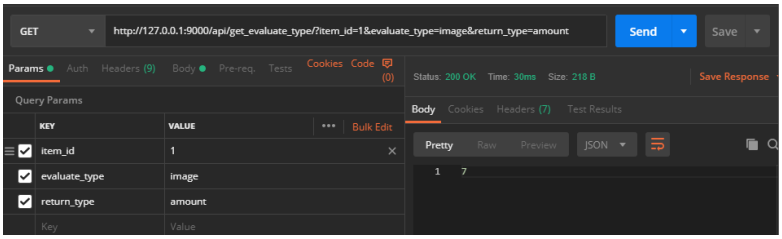

图 3-27

- 示例：获取id为1的商品的好评信息

  http://127.0.0.1:8000/api/get_evaluate_type/?item_id=1&evaluate_type=good&return_type=info

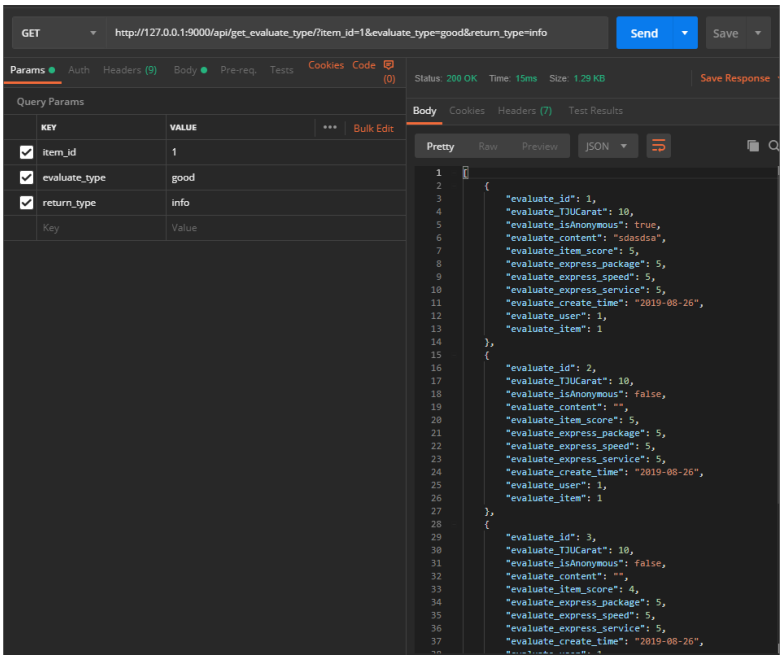

图 3-28

- 根据用户id返回待收货的商品信息以及数量

  接口：http://127.0.0.1:8000/api/get_wait_receive/?user_id=x&return_type=xxx

  请求格式如上所示，参数可以乱序但参数名一定要对应

  此接口只允许GET

  Return_type表示返回类型，info代表返回商品详情、amount返回数量

示例：返回id为1的用户的待收货数量

http://127.0.0.1:8000/api/get_wait_receive/?user_id=1&return_type=amount

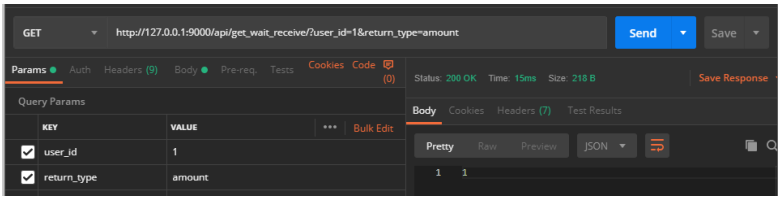

图 3-29

- 示例：返回id为1的用户的待收货的商品详情

  http://127.0.0.1:8000/api/get_wait_receive/?user_id=1&return_type=info

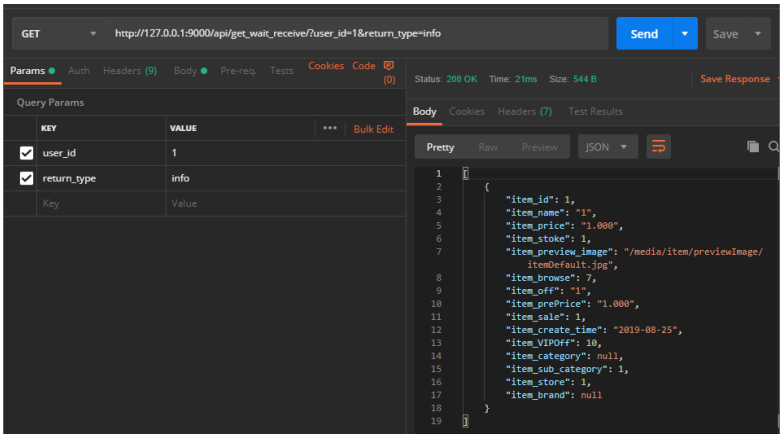

图 3-30

- 根据用户id返回待评价的商品信息以及数量

  接口：http://127.0.0.1:8000/api/get_wait_evaluate/?user_id=x&return_type=xxxx

  请求格式如上所示，参数可以乱序但参数名一定要对应

  接口只允许GET

  Return_type为返回类型，amount表示返回数量，info表示返回商品详情

  示例：返回id为1的用户的待评价商品信息

  http://127.0.0.1:8000/api/get_wait_evaluate/?user_id=1&return_type=info
- 示例：返回id为1的用户的待评价数

  http://127.0.0.1:8000/api/get_wait_evaluate/?user_id=1&return_type=amount
- 根据用户id返回该用户所有已完成的订单数及商品详情

  接口：

  http://127.0.0.1:8000/api/get_complete_trade/?user_id=x&return_type=xxxx

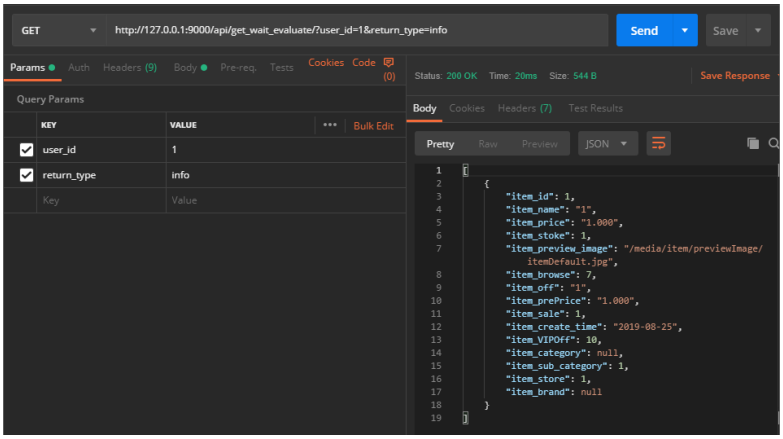
GET
http://127.0.0.1:9000/api/get_wait_evaluate/?user_id=1&return_type=info
Send
Save
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Cookies
Code
(0)
Status: 200 OK
Time: 20ms
Size: 544 B
Save Response
Query Params
KEY
VALUE
Bulk Edit
user_id
1
return_type
info
Key
Value
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
JSON
[
{
"item_id": 1,
"item_name": "1",
"item_price": "1.000",
"item_stoke": 1,
"item_preview_image": "/media/item/previewImage/
itemDefault.jpg",
"item_browse": 7,
"item_off": "1",
"item_prePrice": "1.000",
"item_sale": 1,
"item_create_time": "2019-08-25",
"item_VIPOff": 10,
"item_category": null,
"item_sub_category": 1,
"item_store": 1,
"item_brand": null
}
]

图 3-31

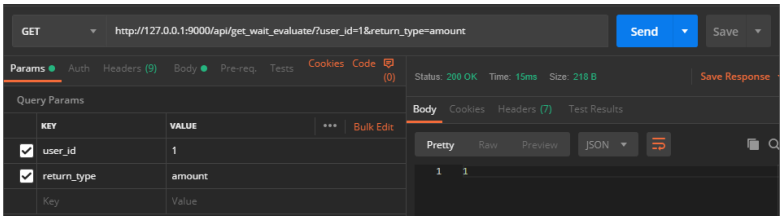
GET
http://127.0.0.1:9000/api/get_wait_evaluate/?user_id=1&return_type=amount
Send
Save
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Cookies
Code
(0)
Status: 200 OK
Time: 15ms
Size: 218 B
Save Response
Query Params
KEY
VALUE
Bulk Edit
user_id
1
return_type
amount
Key
Value
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
JSON
1 1

图 3-32

请求格式如上所示，参数可以乱序但参数名一定要对应

接口只允许GET

Return_type为返回类型，amount表示返回数量，info表示返回商品详情

示例：返回id为1的用户的所有已完成的订单数

http://127.0.0.1:8000/api/get_complete_trade/?user_id=1&return_type=amount

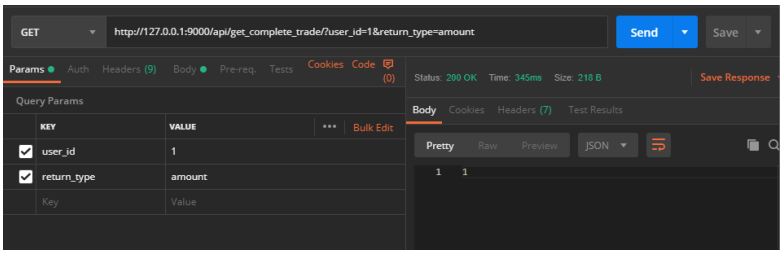

图 3-33

- 示例：返回id为1的用户的所有已完成的订单的商品详情

  http://127.0.0.1:8000/api/get_complete_trade/?user_id=1&return_type=info

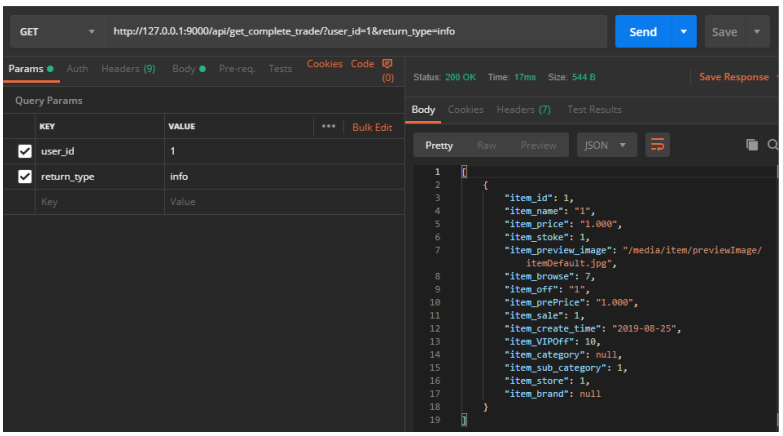

图 3-34

- 根据用户id、商品id、商店id、购买数量，处理前端点击“立即购买”后的逻辑

  接口：

  http://127.0.0.1:8000/api/buy_now/?user_id=x&item_id=x&store_id=x&amount=xx

  请求格式如上，参数可以乱序但参数名必须对应。

  该接口只允许POST。

  后端封装的功能：先检测商品的库存够不够，不够的话直接返回字符串“stoke is not enough”。若足够，则给商品的库存减去购买量，销量加上

购买量。然后在Trade中加入新记录，trade_info置0，表示未发货。上述过程成功后返回success。

示例：在商店1中立即购买商品1 99件

http://127.0.0.1:8000/api/buy_now/?user_id=1&item_id=1&store_id=1&amount=99

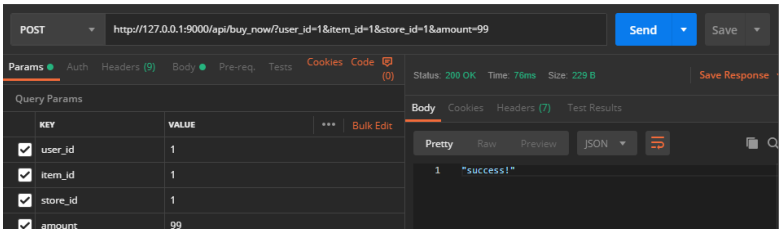

图 3-35

- 再买一次200件：

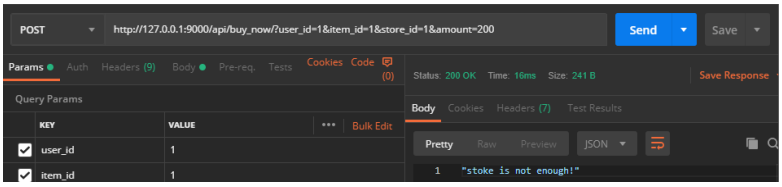

图 3-36

- 根据用户id、商品id、购买数量，处理前端点击加入购物车的逻辑

  接口：

  http://127.0.0.1:8000/api/add_into_cart/?user_id=x&item_id=x&amount=xx

  请求格式如上，参数可以乱序但参数名必须对应。

  该接口只允许POST。

  后端封装的功能：先检测商品的库存够不够，不够的话直接返回字符串“stoke is not enough”。若足够，在购物车中加记录（不会减该商品的库存，会合并同用户同商品的记录）。上述成功后返回success。

  示例：用户1对商品1立即加入购物车20件

  http://127.0.0.1:8000/api/add_into_cart/?user_id=1&item_id=1&amount=20

- 根据用户id、商品id、商店id、购买数量，处理前端在购物车页面进行购买的逻辑

  接口

  http://127.0.0.1:8000/api/buy_in_cart/?user_id=x&item_id=x&store_id=x&amount=xx

  请求格式如上，参数可以乱序但参数名必须对应。

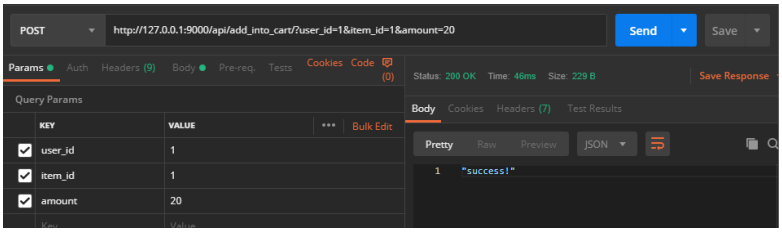

图 3-37

该接口只允许POST。

后端封装的功能：先检测商品的库存够不够，不够的话直接返回字符串“stoke is not enough”。若足够，则给商品的库存减去购买量，销量加上购买量。然后在Trade中加入新记录，trade_info置0，表示未发货。购物车删除对应记录。上述过程成功后返回success。

示例：用户1对商品1商店1在购物车下单20件

http://127.0.0.1:8000/api/buy_in_cart/?user_id=1&item_id=1&store_id=1&amount=20

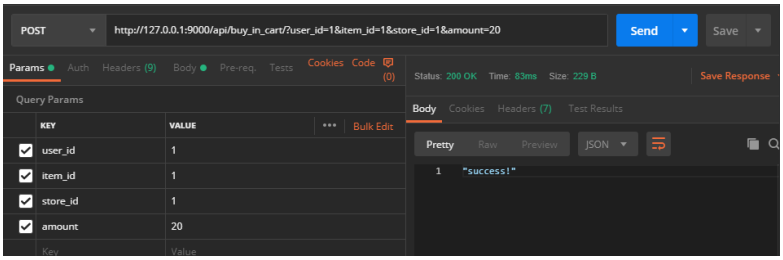

图 3-38

- 根据trade_id进行收货

  接口：http://127.0.0.1:8000/api/confirm_receive/pk

  网址最后的参数pk是交易的trade_id。此接口只允许POST请求。

  示例：对交易记录第一条进行收货处理（trade_info置3）

  http://127.0.0.1:8000/api/confirm_receive/1
- 根据user_id返回改用户浏览记录的商品详情（按时间降序）

  接口：http://127.0.0.1:8000/api/get_history_item/pk

  网址最后的参数pk是用户的user_id。此接口只允许GET请求。

  示例：返回用户1的浏览记录的商品详情
- 根据user_id判断判断是否开店，若开店返回商店详情；若没开，返回数字0

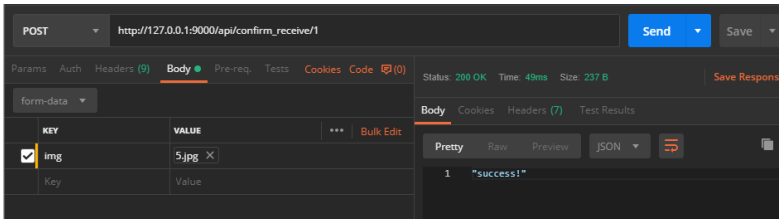

图 3-39

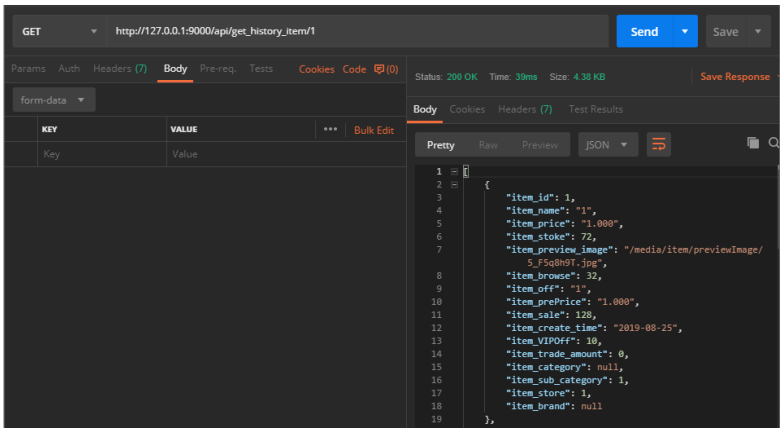

图 3-40

接口：http://127.0.0.1:8000/api/get_user_store_info/pk
网址最后的参数pk是用户的user_id。此接口只允许GET请求。
示例：判断用户1是否开店

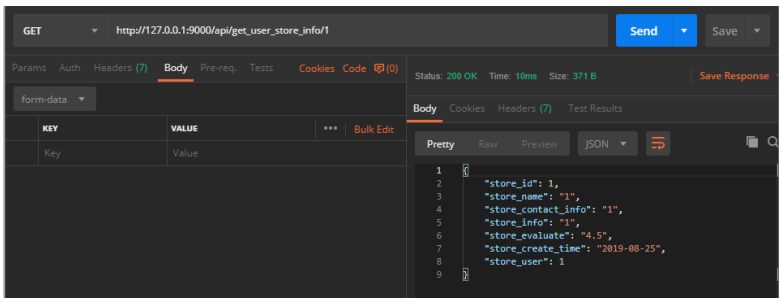

图 3-41

- 根据user_id以及其他数据进行地址上传，如果已存在该用户的地址则更新记录
  接口：
  http://127.0.0.1:8000/api/update_user_address/?user_id=x&detail=x&name=x&tel=x
  此接口只允许POST请求。

Detail表示地址详情、name表示收货人姓名、tel表示收货人联系方式

示例：上传用户1的收货地址

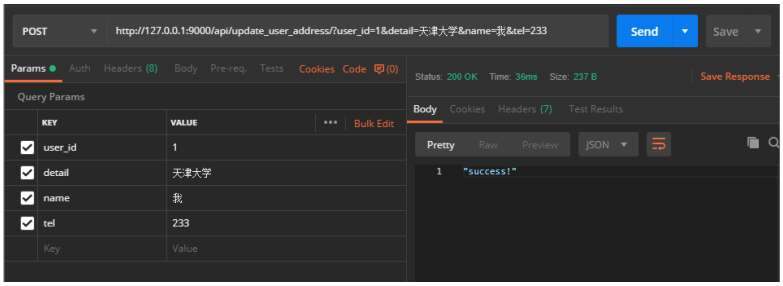

图 3-42

第四章　　软件部署使用文档

4.1　数据库建表

本项目使用python的Django框架，于是在项目开始的时候建立了两个app,分别名为api和backend,api用于提供各种api接口，backend用于后台的数据库部署，本项目没有再另外和其它数据库对接，而是使用的框架自带的db.sqlite3。

要使用或者为本软件添加新功能，可能会需要创建新的数据库表，需要在backend文件夹下的models.py文件里创建对象，这样其实就相当于在Python的db.sqlite3里进行了数据库表的添加

```python
#用户
class User(models.Model):
    user_id = models.AutoField(verbose_name=u'用户', primary_key=True)
    user_nickname = models.CharField(verbose_name=u'昵称', max_length=30, default='none', unique=True)#用户设置
    user_password = models.CharField(verbose_name=u'密码', max_length=18, default='123456')
    user_sex = models.CharField(verbose_name=u'性别', max_length=1, default='男')
    user_tel = models.CharField(verbose_name=u'手机号', max_length=10, default='0')
    user_headImage = StdImageField(max_length=100, upload_to='user/headImage', verbose_name='头像',
                                   variations={'thumbnail':{'width':50, 'height':50}}, default='user/headImage/headDefault.jpg')
    user_isAdmin = models.BooleanField(verbose_name=u'用户是否为管理员', default=False)#系统预先内置一些管理员账户，前端提交用户注册表单时此项默认
    user_isBindQQ = models.BooleanField(verbose_name=u'用户是否绑定QQ', default=False)
    user_location = models.CharField(verbose_name=u'位置', max_length=30, default=u'中国')
    user_location_x = models.DecimalField(verbose_name=u'用户经度', max_digits=100, decimal_places=5, default=0)
    user_location_y = models.DecimalField(verbose_name=u'用户纬度', max_digits=100, decimal_places=5, default=0)
    user_coupon = models.ManyToManyField(verbose_name=u'用户持有的优惠券', to='Coupon', null=True, blank=True)
    user_TJUCarat = models.IntegerField(verbose_name=u'洋克拉', default=0)#洋克拉：类似于京豆的一种优惠机制
    user_isVIP = models.BooleanField(verbose_name=u'用户是否为VIP', default=False)



    def __str__(self):
        return '%s' % self.user_id

    def isAdmin(self):
        if self.user_isAdmin:
            return '是'
        else:
            return '否'

    def isBindQQ(self):
        if self.user_isBindQQ:
            return '是'
        else:
            return '否'

```

图 4-1　　数据库建表

4.2　序列化设置

建立了数据库表之后，如何将其变为json向前台返回是个问题，在api文件夹下的serializers.py文件便是处理这种问题的文件，相当于在这里定义好相关的函数，对表中的数据项全部序列化还是部分序列化进行配置与选择

4.3　过滤器设置

前端会向后端数据库请求不同的查询信息，在api文件夹下的filter.py文件里可以配置表里各数据项的匹配模式，如完全匹配、模糊匹配、范围匹配等（其实通过不同匹配的组合就相当于实现了SQL语言的各种查询）

```
5
6   class UserSerializers(serializers.ModelSerializer):
7       class Meta:
8           model = User
9           fields = '__all__'
10
11  class StoreSerializers(serializers.ModelSerializer):
12      class Meta:
13          model = Store
14          fields = '__all__'
15
16
```

图 4-2　　序列化设置

```
4
5   class UserFilter(django_filters.rest_framework.FilterSet):
6       min_carat = django_filters.NumberFilter(field_name='user_TJUCarat', lookup_expr='gte')
7       max_carat = django_filters.NumberFilter(field_name='user_TJUCarat', lookup_expr='lte')
8
9       user = django_filters.CharFilter(field_name='user_nickname', lookup_expr='exact')
10
11      class Meta:
12          model = User
13          fields = {
14              'user_id': ['exact'],
15              'user_nickname': ['icontains'],
16              'user_sex': ['exact'],
17              'user_tel': ['exact'],
18              'user_isAdmin': ['exact'],
19              'user_isVIP': ['exact'],
20              'user_location': ['icontains'],
21              'user_TJUCarat': ['exact'],
22              'user_password': ['exact'],
23          }
24
25
```

图 4-3　　过滤器设置

4.4 路径设置

api文件夹下的urls.py负责路径的配置，将不同的请求分发给不同的views处理

```
7
8   route.register(r'user', views.UserViewSet)
9   route.register(r'store', views.StoreViewSet)
10  route.register(r'itemCategory', views.ItemCategoryViewSet)
11  route.register(r'itemSubCategory', views.ItemSubCategoryViewSet)
12  route.register(r'item', views.ItemViewSet)
13  route.register(r'shoppingCart', views.ShppingCartViewSet)
14  route.register(r'trade', views.TradeViewSet)
15  route.register(r'group', views.GroupViewSet)
16  route.register(r'storeFollow', views.StoreFollowViewSet)
17  route.register(r'itemFollow', views.ItemFollowViewSet)
18  route.register(r'groupFollow', views.GroupFollowViewSet)
19  route.register(r'banner', views.BannerViewSet)
20  route.register(r'systemMessage', views.SystemMessageViewSet)
21  route.register(r'chatMessage', views.ChatMessageViewSet)
22  route.register(r'coupon', views.CouponViewSet)
23  route.register(r'dailyOffItem', views.DailyOffItemViewSet)
24  route.register(r'itemDetailImage', views.ItemDetailImageViewSet)
25  route.register(r'itemBannerImage', views.ItemBannerImageViewSet)
26  route.register(r'evaluateImage', views.EvaluateImageViewSet)
27  route.register(r'address', views.AddressViewSet)
28  route.register(r'brand', views.BrandViewSet)
29  route.register(r'collection', views.CollectionViewSet)
30  route.register(r'browseRecord', views.BrowseRecordViewSet)
31  route.register(r'searchRecord', views.SearchRecordViewSet)
32  route.register(r'evaluate', views.EvaluateViewSet)
33
```

图 4-4　　路径设置

4.5 功能函数设置

views.py是本项目内的核心模块，负责根据前端提供的各种信息来执行相应的操作，如查询，将文件上传数据库等，此处将调用serializers.py里配置好的函数，在有需要时返回给前端json对象

```
245
246  #输入用户id，返回该用户购物车的所有商品
247  @api_view(['GET'])
248  def get_user_cart_item(request, pk):
249      if request.method == 'GET':
250          cart_info = ShoppingCart.objects.filter(cart_user=pk)
251
252          queryset = set()#查询结果信息放到queryset
253          for cart in cart_info:
254              item_id = cart.cart_item.item_id #获取到item对象的item_id
255              items = Item.objects.filter(item_id=item_id) #构造queryset
256              queryset = chain(queryset, items) #链接queryset
257          serializer = ItemSerializers(queryset, many=True)
258          return Response(serializer.data)
259      else:
260          return Response('Request Method Error! This API Only Allow GET Method! ')
261
```

图 4-5　　功能函数设置

第五章　人员分工与项目亮点

5.1　团队人员分工

- 宋高超: 项目经理：API后端控制器功能模块的编写与更新，API测试，文档编写与排版
- 刘宇：全栈开发师：浏览器前端与后端的编写，
- 赵为：服务器运维师：微信小程序前端的编写，服务器的租用，域名解析与备案、系统项目的远程服务器部署
- 张昊臻：首席架构师：安卓小程序开发，API后端架构，提供API文档

5.2　项目工作量

后台共建立了25张数据库表，总代码量超过20000行，日均代码量1000多行，文档达到上万字

5.3　项目亮点

- 项目规模庞大：代码量巨多，开发环境多样化，编程语言多样化
- 多平台适配：提供微信小程序、安卓app、浏览器三个前端，同时又提供了能够同时支持三个前端的后端。而此后端采用了API接口与模板渲染相结合的方式，将前后端分离与MTV模式相结合，是一种比较有创意的想法。
- 服务器远程部署：团队成员自行租用阿里云服务器，并且成功完成了域名的解析与备案，现在，直接访问http://www.mallproject.cn:8000/login即可进入登录界面，直接访问http://www.mallproject.cn:8000/api/即可查看Api root界面，倘若再有足够时间完善安全性相关问题，本项目完全可以成为一个可以正常运营的电子购物商城!
- 一些人性化的功能：在完成了电子购物平台基本的用户与商品信息查询、购买之外，还添加了用户可以开店铺自行售卖物品、用户收藏商品与店铺、一键清空购物车、查询浏览记录、根据用户提供的关键词信息全站搜索商品、对不同类别商品按综合、销量、价格排序等功能。

第六章　　项目心得

通过本次项目的学习，团队的成员们对某一项目多平台开发积累了宝贵的经验，对软件设计中前后端分离与否有了更深入的认识，学会了往更高的架构层面看待问题，对服务器的部署，Linux基本操作命令方面也进一步地加深了了解，团队成员们分工明确，通过API文档等进行沟通，进一步提高了效率，最终实现了1+1+1+1>4的效果!