总体报告

1. 需求分析

1.1 项目背景

该项目开发的软件为一个自由、双向的商品交易平台。随着互联网的发展，电子商务市场的规模不断扩大。越来越多的消费者开始选择在线购物，而不是传统的线下零售店。在这样的背景下，一个商品交易平台是一个有潜力的项目，可以为消费者和卖家提供一个方便、快捷的交易平台。

这个商品交易平台的目标是为所有用户提供一个平等、开放的市场。无论是个人还是企业，都可以在平台上出售自己的商品。所有商品都将由卖家自己上传，并根据平台的规定进行管理和审核。平台将提供安全、妥善的订单管理系统确保消费者可以放心地购买商品。除消费系统之外，平台也将提供一系列功能，以增强用户体验。这些功能包括搜索功能、商品分类、商品收藏等。在市场竞争激烈的情况下，平台将积极探索市场需求，调整平台的经营策略，以保持平台的竞争力和长期发展。

1.2 项目功能

1.2.1 用户需求

本节根据用户提出的需求描述系统的功能，首先列出各类用户的需求

客户需求：

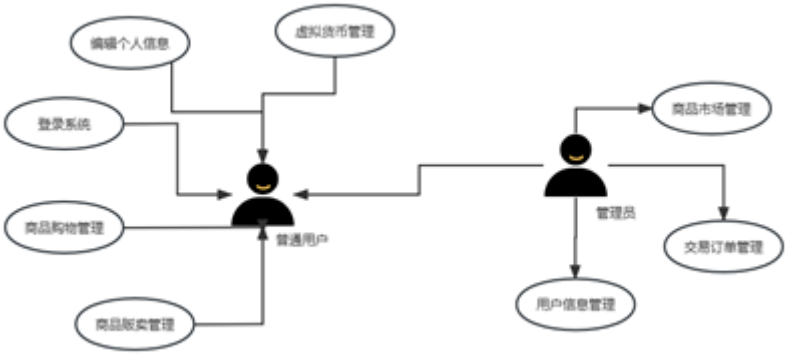
序号	优先级	需求内容
1	高	网站能提供清晰的购物商场界面，用户可以作为买家直观地进行购买操作
2	高	网站能提供清晰的商品上架界面，用户可以作为卖家上架自己想要投放在市场上的产品
3	高	网站能够提供清晰的订单追踪功能，用户可以作为买家，查看自己任何一笔交易的相关情况，并确认收货
4	高	网站可以提供订单处理功能，用户可以作为卖家，查看自己上架的商品有哪些购买记录，进行出货处理

5	中	网站能提供用户信息编辑界面，用户可以修改个人的对应信息并进行相应内容的查看
6	中	网站能提供虚拟货币管理系统，可以通过触发特定事件，修改个人的网络交易资产

管理员需求：

1	高	网站能够为管理员提供所有用户的个人信息管理
2	高	网站能够使管理员对交易的所有订单进行查看管理，并依据要求生成可视化图表反应交易情况
3	高	网站能够使管理员进行市场管理，可以强行对不合规定的商品进行下架操作
4	中	网站能够使得管理员对每个用户的充值记录进行管理，进行后台的累计
5	中	网站能够使得管理员检索商品市场的上架记录

1.2.2 用例图

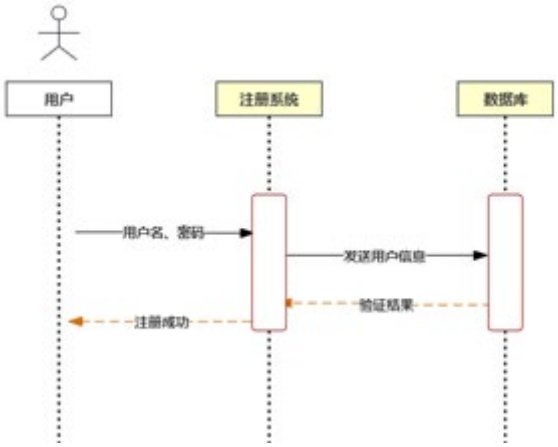


1.2.3 功能列表

1.2.3.1 注册

优先级：高

用户在使用系统前必须进行注册，只有注册的信息合规，才能够注册成功并进入下层的登录之中。



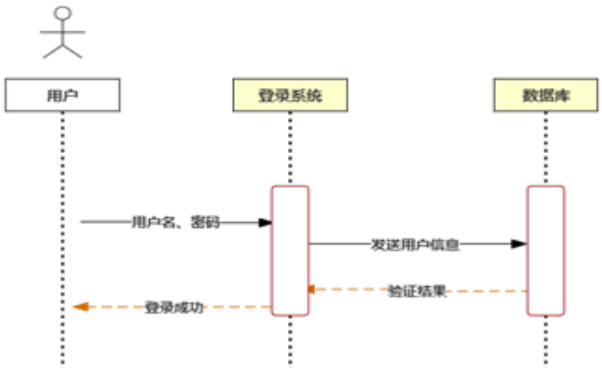
用例文档

用例名称	用户注册
用例编号	USE-CASE-1
行为角色	普通用户
简要说明	所有用户使用本系统前必须完成注册
前置条件	用户进入注册界面
后置条件	登录成功后，系统会跳转到登录界面
流程	1、用户打开注册界面 2、用户输入电话号码、邮箱号、昵称和密码 3、系统访问用户数据库验证账户电话及邮箱未被注册且合规 4、如果验证失败，显示注册失败提示信息，返回第 2 步 5、如果验证成功，显示注册成功，进入登录页面
异常处理	无论验证结果如何，都将给予反馈
备注	无

1.2.3.2 登录

优先级：高

用户在使用系统前必须进行登录。用户登录系统应该能根据用户输入的账户与密码验证其身份，具体的验证功能由登录系统提供。验证身份成功后，用户方可进入该系统进行后续操作。



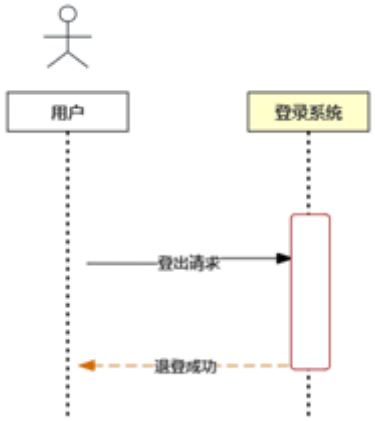
用例文档

用例名称	用户登录
用例编号	USE-CASE-2
行为角色	普通用户、管理员
简要说明	所有用户进入本系统前必须完成登录
前置条件	用户进入登录页面
后置条件	登录成功后，系统会根据用户不同的用户名密码登录相应的主页
流程	1、用户打开登录界面 2、用户输入账户与密码 3、系统访问用户数据库验证身份 4、如果验证失败，显示登陆失败提示信息，返回第 2 步 5、如果验证成功，显示登录成功，进入用户主页
异常处理	无论验证结果如何，都将给予反馈
备注	无

1.2.3.3 登出

优先级：高

用户在登录系统后可以随时选择登出，这让用户退出登录状态，返回登录界面。



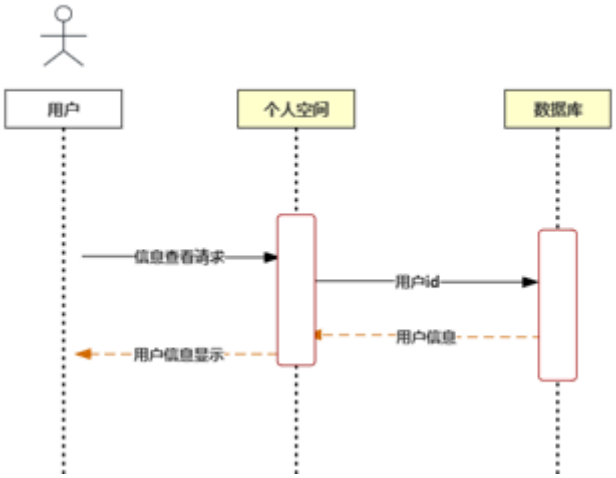
用例文档

用例名称	用户登出
用例编号	USE-CASE-3
行为角色	普通用户、管理员
简要说明	所有登录后的用户均可以登出
前置条件	用户已成功登录
后置条件	登出后，将会返回登陆界面
流程	1、用户点击登出按钮 2、系统返回登录界面
异常处理	无
备注	无

1.2.3.4 用户信息查看

优先级：中

用户在登录系统后可以选择查看个人信息。



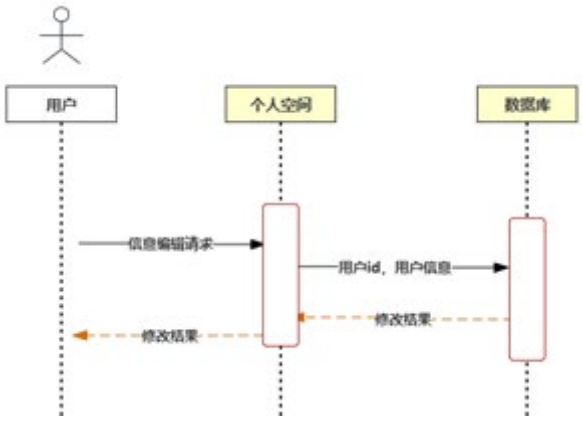
用例文档

用例名称	用户信息查看
用例编号	USE-CASE-4
行为角色	普通用户
简要说明	所有登录后的用户均可以在个人中心查看信息
前置条件	用户进入个人中心的查看信息界面
后置条件	进入后，将显示个人信息
流程	1、用户点击侧边栏查看信息分栏按钮 2、系统返回个人信息界面
异常处理	无
备注	无

1.2.3.5 用户信息修改

优先级：高

用户在登录系统后可以随时选择对个人信息进行修改，并进行相应的提交。



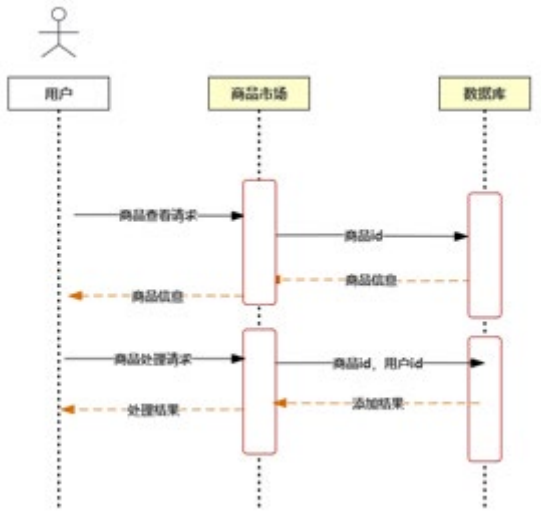
用例文档

用例名称	用户信息修改
用例编号	USE-CASE-5
行为角色	普通用户
简要说明	登录后的用户可以在个人中心对自身信息进行修改编辑
前置条件	用户进入个人中心的信息修改界面
后置条件	修改相应信息后将会进行保存
流程	1、用户进入侧边栏的信息修改界面 2、用户提交信息修改 3、后台数据库检查信息修改是否合规 4、若合规，返回修改成功 5、若不合规，返回修改失败
异常处理	无
备注	无

1.2.3.6 商品购买

优先级：高

用户可以浏览商品市场，并对商品进行点击，并选择是加入自己收藏栏，还是直接进行购买。



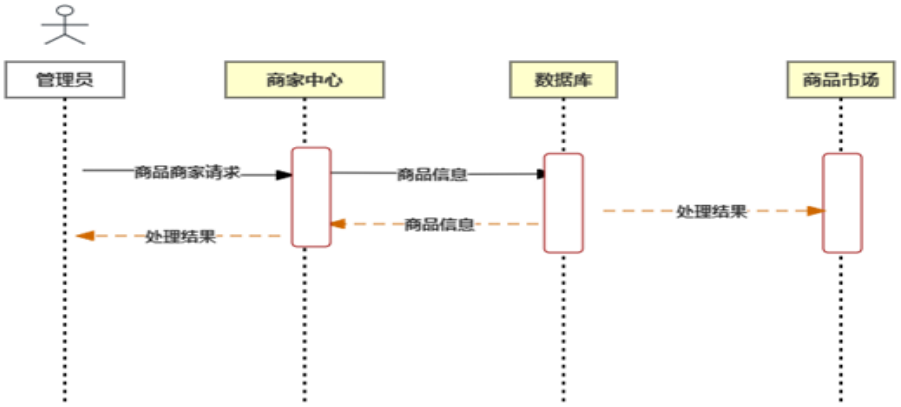
用例文档

用例名称	商品购买
用例编号	USE-CASE-6
行为角色	普通用户
简要说明	所有用户可以在商品市场对心仪商品进行购买
前置条件	用户进入商品市场界面，并浏览商品
后置条件	点击商品，并进行相应操作
流程	1、用户打开商品市场界面 2、用户点击心仪的商品 3、选择对商品加入收藏，还是直接购买 4、若选择直接购买，进行付款判定 5、付款成功后，弹出提示 6、付款失败后，弹出失败原因 7、返回商品市场
异常处理	若进行付款，保证会返回付款结果
备注	无

1.2.3.7 商品上架

优先级：高

用户可以在商家中心上架自己的商品，对商品信息进行相应的完善后，商品市场中将会显示相应的商品界面



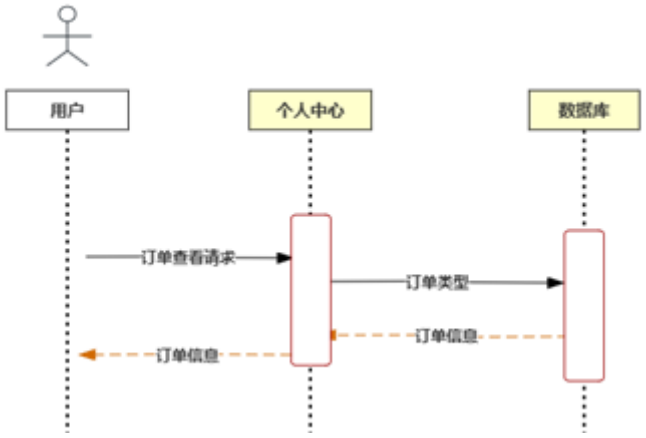
用例文档

用例名称	商品上架
用例编号	USE-CASE-7
行为角色	普通用户
简要说明	所有用户可以在商家中心上架自己的商品
前置条件	用户进入商家中心界面，并点击上架商品按钮
后置条件	用户完善上架商品所需要的信息
流程	1、用户打开商家中心界面 2、用户点击上架商品按钮 3、对上架商品完善要填写的信息 4、若填写无误，添加成功 5、若填写有误，将会返回错误原因
异常处理	若商品上架有错误，将会返回原因
备注	无

1.2.3.8 订单查看

优先级：高

用户可以在个人中心查看所有自己购物的订单信息。



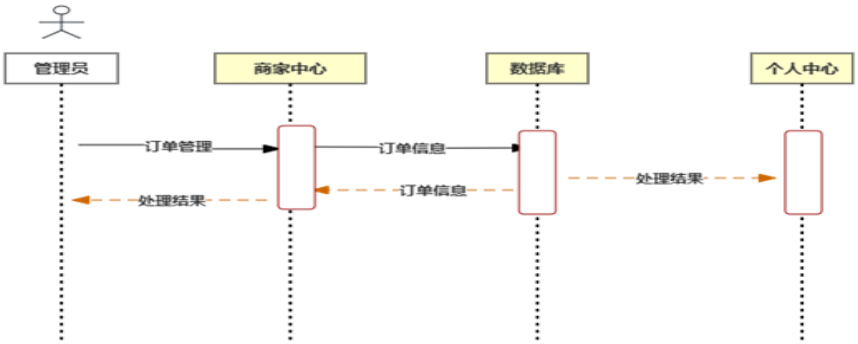
用例文档

用例名称	订单查看
用例编号	USE-CASE-8
行为角色	普通用户
简要说明	所有用户可以在个人中心，查看自身购物的订单
前置条件	用户进入个人中心界面，并点击订单查看按钮
后置条件	用户上下滑动进行浏览
流程	1、用户打开个人中心界面 2、用户点击订单查看按钮 3、用户选择是查看历史订单还是当前订单 4、点击后，将会显示所有对应的订单内容
异常处理	无
备注	无

1.2.3.9 订单管理

优先级：高

用户可以在商家中心对自身的货物进行发货。



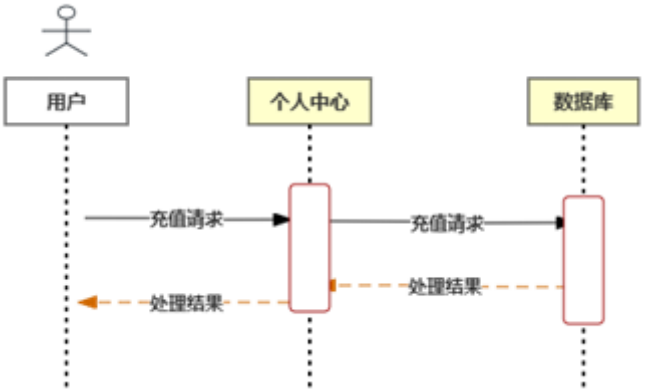
用例文档

用例名称	订单管理
用例编号	USE-CASE-9
行为角色	普通用户
简要说明	所有用户可以在商家中心，处理对自身商品进行购买的订单
前置条件	用户进入商家中心界面，查看所有购物订单
后置条件	用户点击订单进行处理
流程	1、用户打开商家中心界面 2、用户点击订单查看按钮 3、用户选择是否对订单进行发货处理 4、发货后，该订单状态将会变成已发货
异常处理	无
备注	无

1.2.3.10 货币管理

优先级：中

用户可以个人中心查看并充值货币。



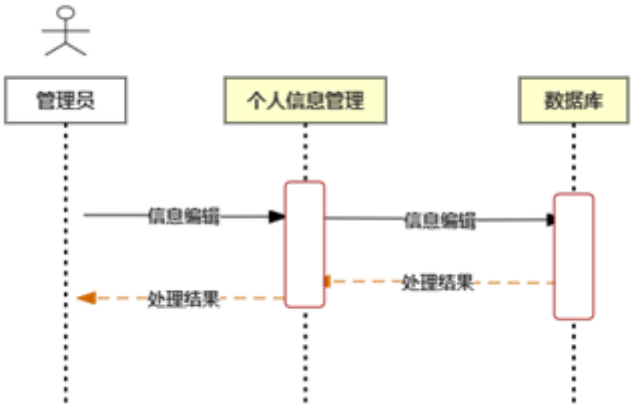
用例文档

用例名称	货币管理
用例编号	USE-CASE-10
行为角色	普通用户
简要说明	所有用户可以在个人中心，进行虚拟货币充值
前置条件	用户进入个人中心界面金币充值分栏，点击充值按钮
后置条件	用户选择充值额度
流程	1、用户进入个人中心 2、用户点击金币充值 3、用户选择充值额度 4、提交充值 5、充值完成后，对应的个人资产将会增加
异常处理	无
备注	无

1.2.3.11 用户信息管理

优先级：中

管理员可以在后台查看所有用户的个人信息，并且可以对用户进行增删改操作。



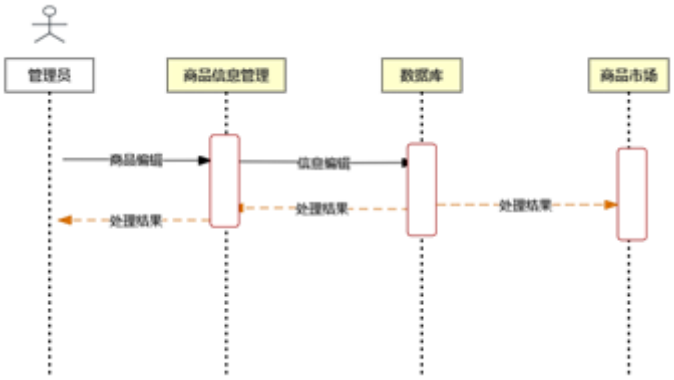
用例文档

用例名称	用户信息管理
用例编号	USE-CASE-11
行为角色	管理员
简要说明	管理员进行对所有用户的信息管理
前置条件	管理员点击用户信息管理模块
后置条件	管理员对用户进行操作
流程	1、管理员进入信息管理中心 2、管理员选择要编辑的用户 3、管理员对用户进行信息的修改或者删除用户 4、返回操作成功或失败的提示
异常处理	无
备注	无

1.2.3.12 商品市场管理

优先级：高

管理员可以在后台查看商品市场内所有的商品，并对不合规的商品进行删除。

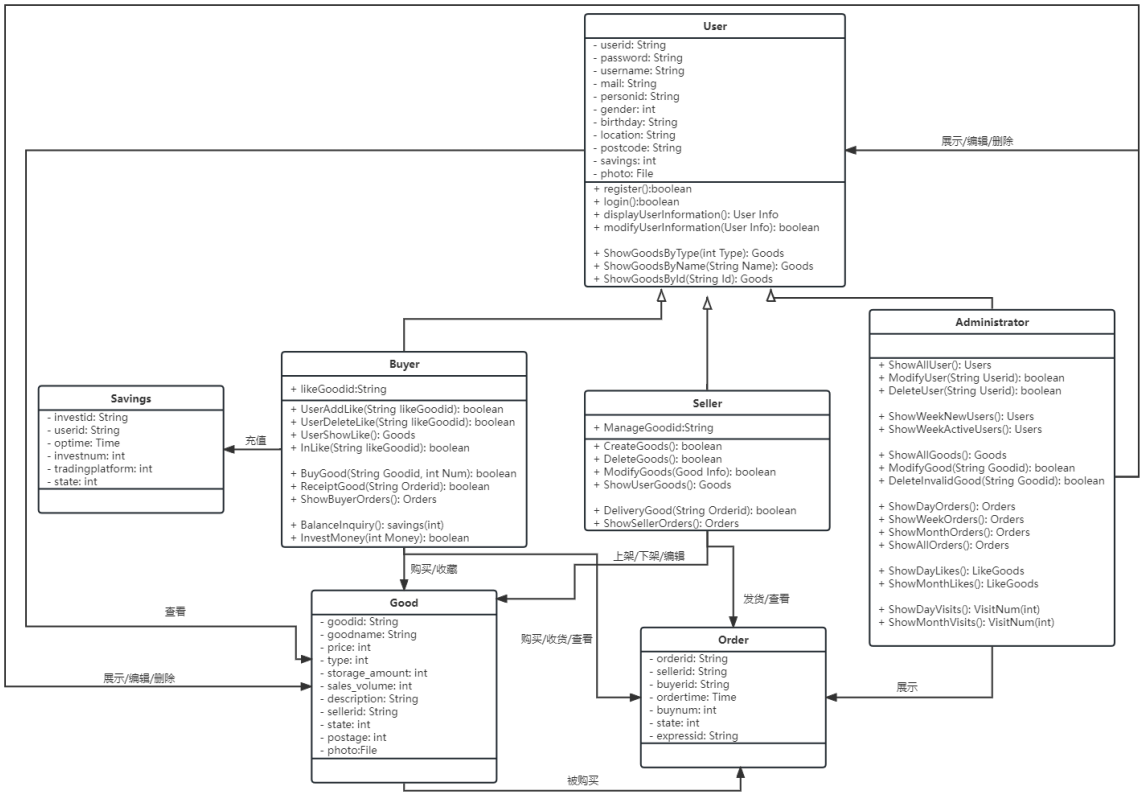


用例文档

用例名称	商品市场管理
用例编号	USE-CASE-12
行为角色	管理员
简要说明	管理员进行对所有商品的管理
前置条件	管理员点击商品管理模块
后置条件	管理员对具体商品进行操作
流程	1、管理员进入商品管理中心 2、管理员点击具体商品 3、若商品内容或其他方面违规，管理员对其进行删除
异常处理	无
备注	无

1.3 类图与 CRC 模型

1.3.1 类图



1.3.2 CRC 模型

1.3.2.1 User

类：User	
编号：CLASS-1	
描述：本系统内全部的角色，包括买家、卖家、管理员。	
功能	合作类
根据商品类型查询商品	Good
根据商品名称查询商品	Good
根据商品 id 查询商品	Good

1.3.2.2 Buyer

类：Buyer	
编号：CLASS-2	
描述：使用系统进行商品购买的角色，继承自 User 类。	
功能	合作类
购买商品	Good, Order

修改收货状态	Order
展示全部订单	Order

1.3.2.3 Seller

类：Seller	
编号：CLASS-3	
描述：使用系统进行商品销售的角色，继承自 User 类。	
功能	**合作类**
上架商品	Good
下架商品	Good
修改商品信息	Good

1.3.2.4 Administrator

类：Administrator	
编号：CLASS-4	
描述：使用系统进行信息统计、高层操作的角色，继承自 User 类。	
功能	**合作类**
展示全部用户	Buyer, Seller
修改用户信息	Buyer, Seller
删除用户	Buyer, Seller
显示本周新增用户信息	Buyer, Seller
显示本周活跃用户信息	Buyer, Seller

1.3.2.5 Good

类：Good
编号：CLASS-5
描述：代表系统内的一种商品。

1.3.2.6 Order

类：Order
编号：CLASS-6
描述：代表系统内的一次订单。

1.3.2.7 Savings

类：Savings
编号：CLASS-7
描述：代表系统内的一次充值记录。

2. 总体设计

2.1 需求规定

该项目开发的软件为一个自由、双向的商品交易平台，为所有用户提供一个平等、开放的市场。无论是个人还是企业，都可以在平台上出售自己的商品。所有商品都将由卖家自己上传，并根据平台的规定进行管理和审核。平台也将提供一系列功能，以增强用户体验。这些功能包括搜索功能、商品分类、商品收藏等。在市场竞争激烈的情况下，平台将积极探索市场需求，调整平台的经营策略，以保持平台的竞争力和长期发展。

2.2 运行环境

a.运行系统：windows 2010

b.数据库管理软件：Mysql，Navicat

c.编程软件：IDEA、Vscode

2.3 体系结构设计

2.3.1 体系结构风格

体系结构风格是施加在整个系统构件上的一种变换，目的是为系统的所有构件建立一个结构。在对已有体系结构再工程时，体系结构风格的转化会导致软件结构的根本性变化，包括对软件构件的再分配等等。恰当的软件结构风格与模式合起来构成了软件的外形。在过去数百万的计算机系统中，绝大多数可以归结成以下少数模式中的一种：

a.以数据为中心的体系结构

b.数据流体系结构

c.调用和返回体系结构

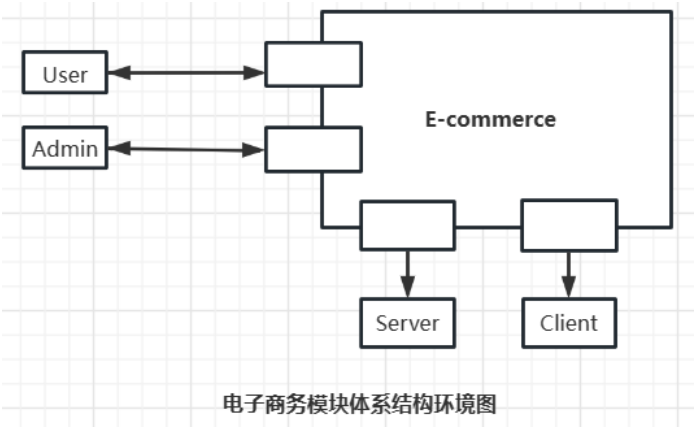
d.面向对象体系结构

e.层次体系结构

2.3.2 层次体系结构设计

2.3.2.1 顶层 - 双向交易的电子商务系统

我们设计的系统是一个简单的双向交易电子商务系统，不是其他系统的组成部分，也不被其他系统所使用，因此它没有上级系统，也没有同级系统。在这里它只有子系统：服务器端子系统和客户端子系统，参与者：用户、管理员。



电子商务模块体系结构环境图

2.3.2.2 次顶层 - 服务器端设计 (Server)

服务器端设计一般运行在远程计算机上，用于处理来自客户端的请求。为了便于系统的维护和扩展，服务器端需要与数据库相连。在电子商务系统中，面临着大量的请求和响应，因此需要采用分层设计，将不同的功能进行模块化处理：

网络模块负责服务器端系统与外界的通信，即接收客户端发来的请求，同时向客户端发送响应。

编码模块负责适应网络传输过程且提高性能的传输问题。

数据库模块主要服务于服务器端系统与数据库之间的通信，负责处理数据库操作。

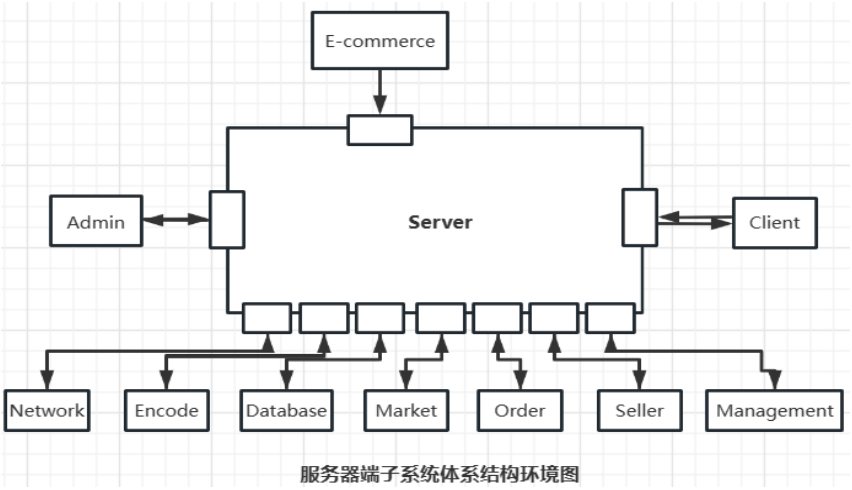
商品市场模块则负责管理商品信息，并提供浏览、搜索等功能，方便买家查找商品。

订单交易管理模块负责管理订单信息，并提供下单、支付、退款等功能，方便买家进行交易。

卖家管理模块则负责管理卖家信息，以及卖家商品发布、库存管理等操作。

后台数据管理模块负责管理整个系统的数据，包括用户信息、商品信息、订单信息等，以及对数据进行统计、分析等操作，方便管理员进行后台管理。

通过将不同的功能模块进行分层设计，可以使系统结构更加清晰，方便系统的维护和扩展。同时，各个模块之间的责任明确，也有利于提高系统的稳定性和可靠性。



服务器端子系统体系结构环境图

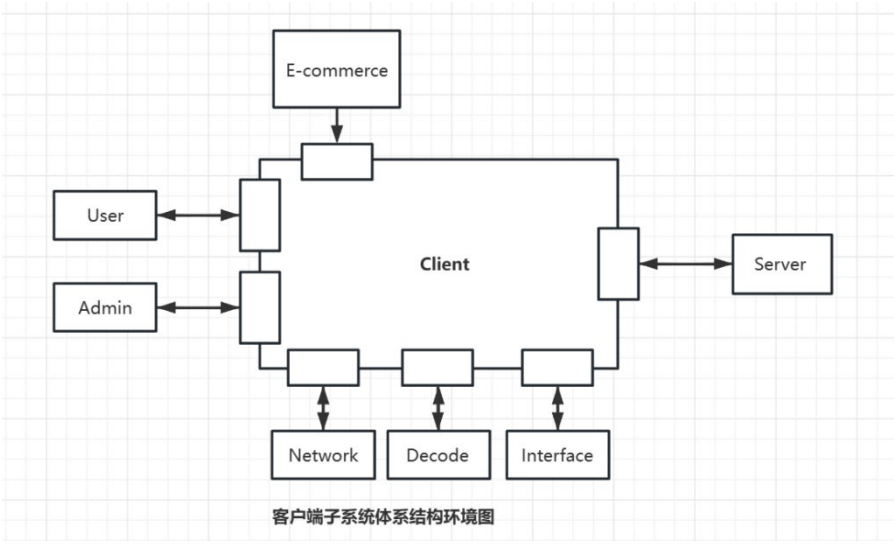
2.3.2.3 次顶层 - 客户端子系统(Client)

为了能够实现与服务器的通信，客户端子系统也需要包括网络模块和解码模块，同时，在电商系统中，一个优美的 UI 设计可以提升用户体验，并且促进用户在平台上的停留时间，因此客户端子系统还需要包括用户界面模块.

网络模块主要负责处理客户端系统与外界的通信，接收服务器端发来的响应，并向服务器端发送请求。

解码模块则负责对从服务器端传送过来的数据进行解析和展示，使得用户能够充分理解数据内容。

用户界面模块主要负责设计直观、易用、美观的用户界面，使得用户能够方便地进行想要的操作。

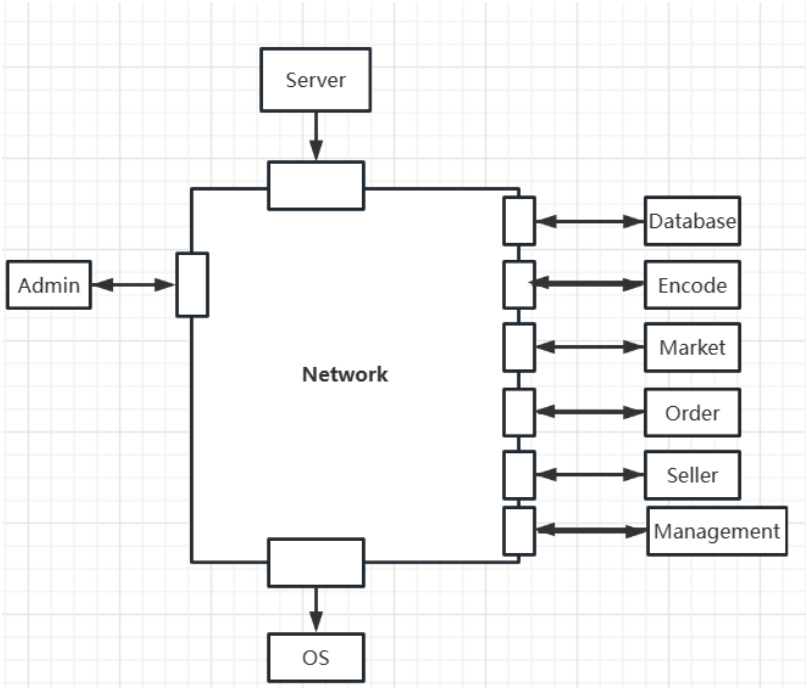


客户端子系统体系结构环境图

完成次顶层的环境搭建之后，我们需要完成更细致的中间层的功能模块的环境和原型设计。

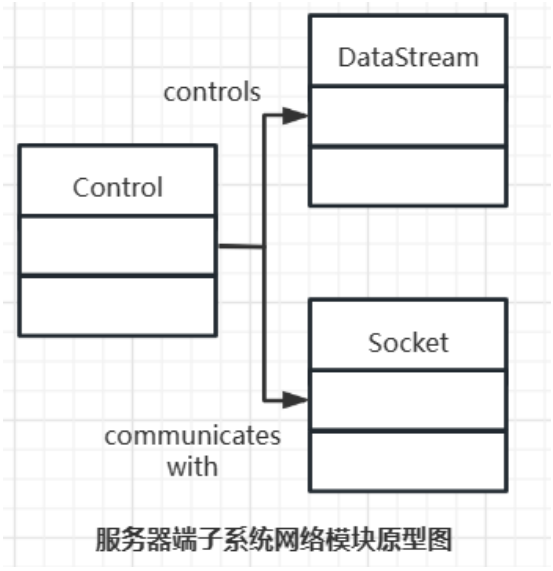
2.3.2.4 中间层 - 服务器端子系统的网络模块 (Network)

此模块是基于操作系统提供的网络接口实现的。



服务器端子系统网络模块环境图

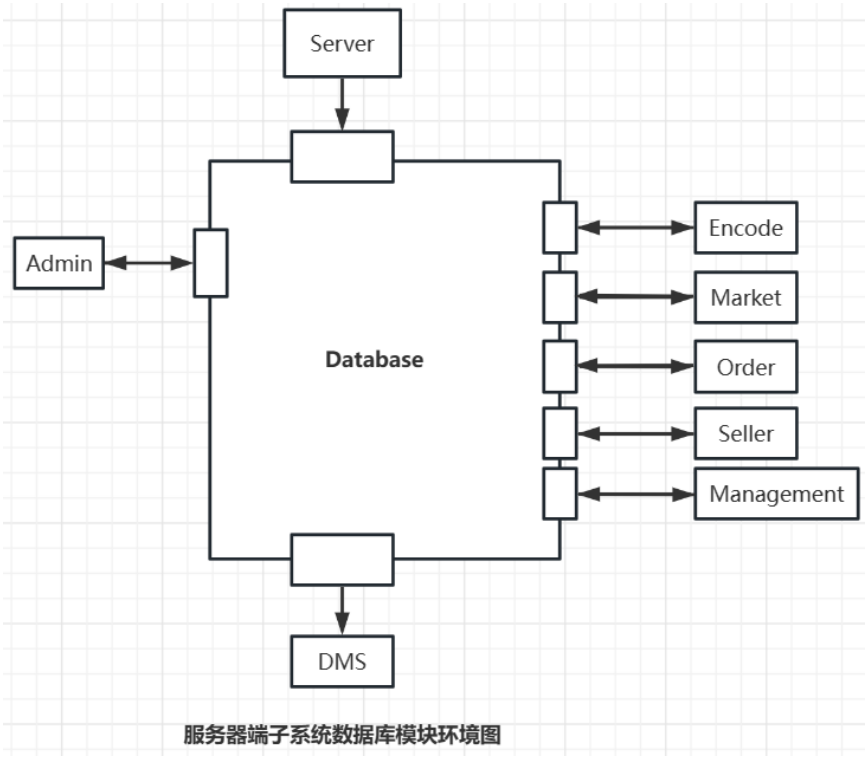
此模块是负责请求和响应的功能组件，联系操作系统和计算机网络相关知识，我们暂定设计该模块原型如图所示：



服务器端子系统网络模块原型图

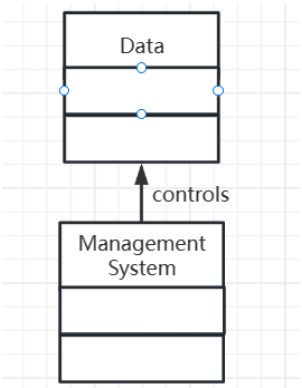
2.3.2.5 中间层 - 服务器端子系统的数据库模块 (Database)

此模块是基于数据库提供的数据接口。



服务器端子系统数据库模块环境图

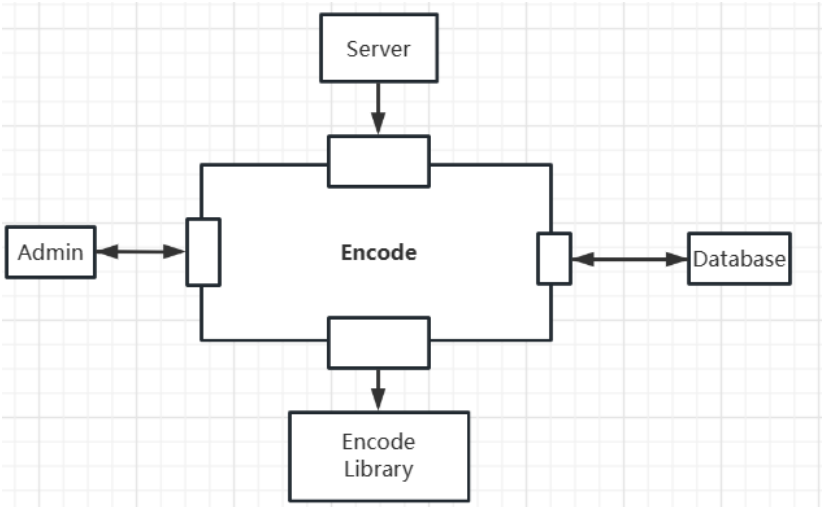
此模块负责响应客户端对数据的增删查改等操作，大致抽象原型如下图所示：



服务器端子系统模块原型图

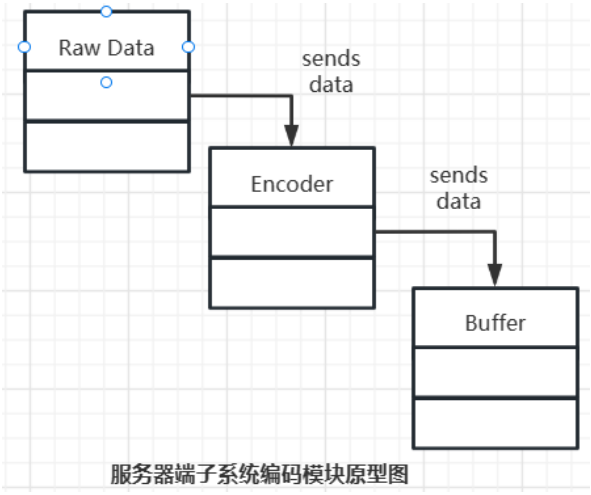
2.3.2.6 中间层 - 服务器端子系统的编码模块（Encode）

此模块是配合网络传输的特定场景所需的功能完善性模块，在之后的性能优化中同样起到非常重要的作用。为了实现数据传输的稳定、安全和高效，我们计划采用已有的成熟库，来满足数据传输过程中的各种需求。



服务器端子系统编码模块环境图

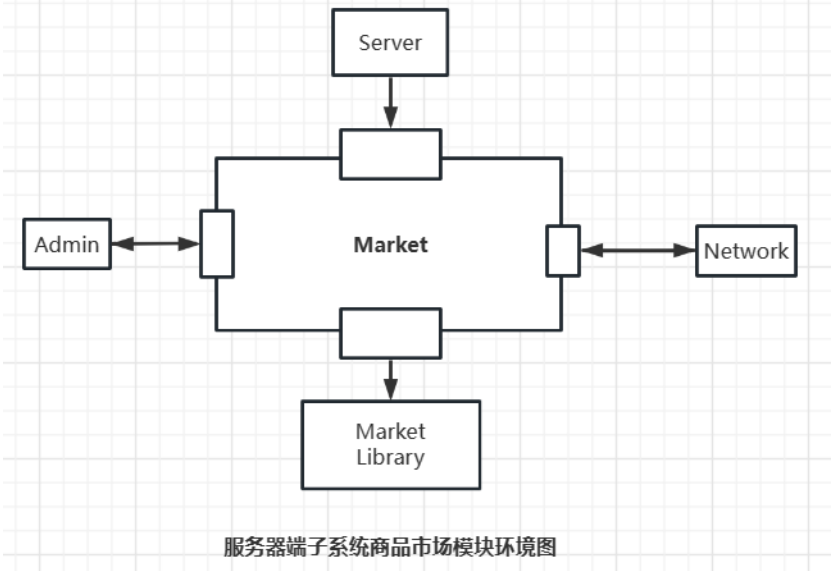
该模块本质上是对数据流的加工转换和传输，因此同样参考数据流模型，将该模型的原型集合设计为数据 - 编码 - 缓冲。原型图如下所示：



服务器端子系统编码模块原型图

2.3.2.6 中间层 - 服务器端子系统的商品市场模块（Market）

此部分基于提供的商品市场库实现。

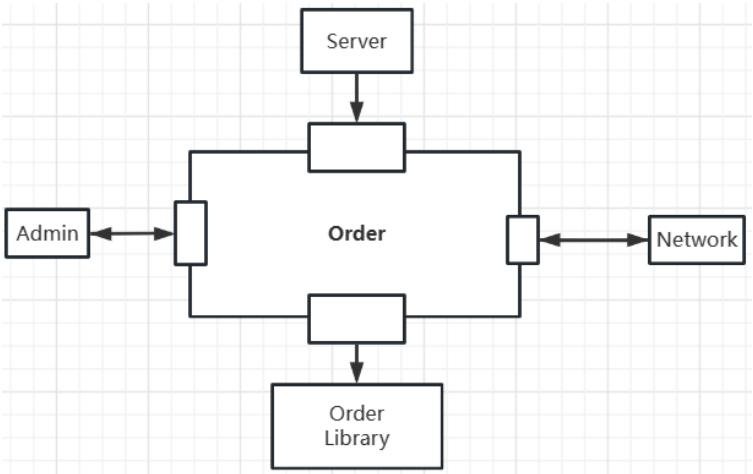


服务器端子系统商品市场模块环境图

该模块封装了所有与商品展示、收藏、添加到购物车相关的方法，保障在商品交易中最美观、准确的过程，抽象成原型即为商品信息的获取、标记。

2.3.2.7 中间层 - 服务器端子系统的订单交易管理模块（Order）

此部分基于提供的订单管理库实现。

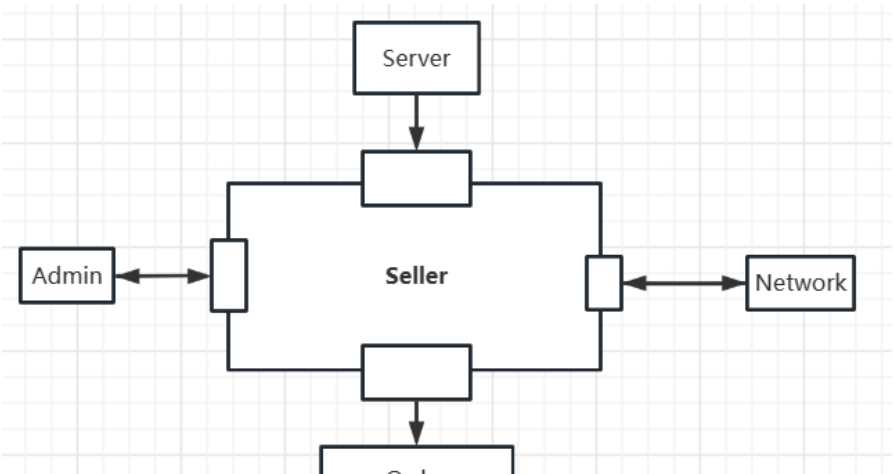


服务器端子系统订单交易管理模块环境图

该模块封装了所有与商品买卖相关的方法，确保商品交易的安全、准确，抽象成原型即为不同账户之间的交易。

2.3.2.8 中间层 - 服务器端子系统的卖家管理模块（Order）

此部分基于提供的卖家管理库实现。

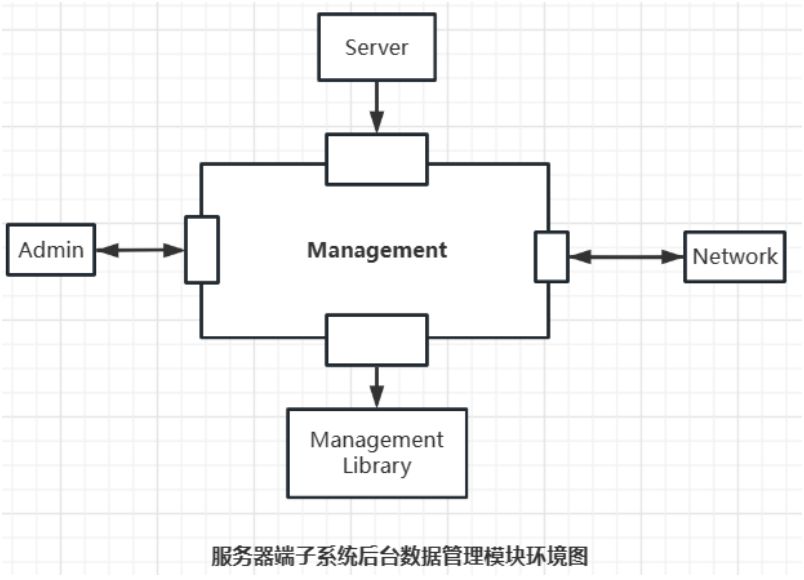


服务器端子系统卖家管理模块环境图

该模块封装了所有与卖家管理商品信息相关的方法，确保商品上架的正确性。

2.3.2.9 中间层 - 服务器端子系统的后台数据管理模块（Management）

此部分基于提供的后台数据管理库实现。

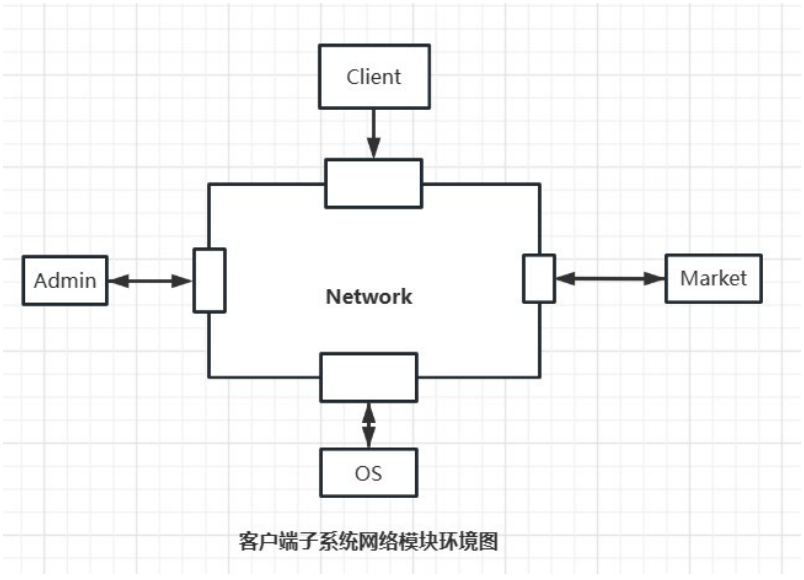


服务器端子系统后台数据管理模块环境图

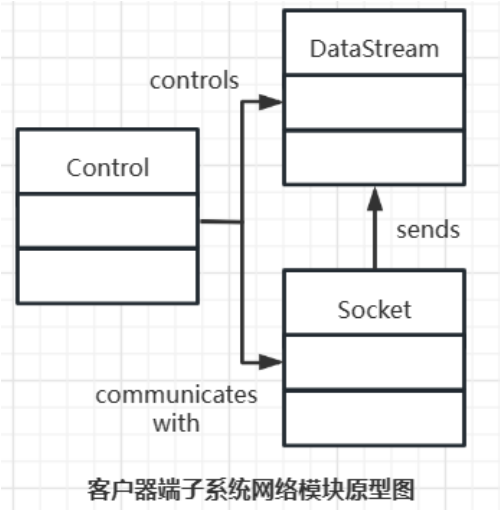
该模块封装了所有与管理员获取信息，管理商品相关的方法，确保管理员能获取正确信息、下架不合理商品。

2.3.2.10 中间层 - 客户端子系统的网络模块 (Network)

该设计与大部分设计一样，此模块与服务器端设计模块大致相同。只需要注意此处需解码而非编码，因此将原型图中的数据流反过来即可。



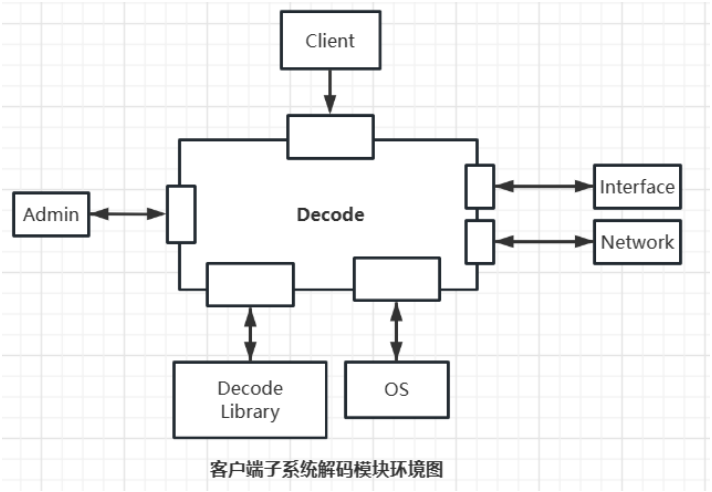
客户端子系统网络模块环境图



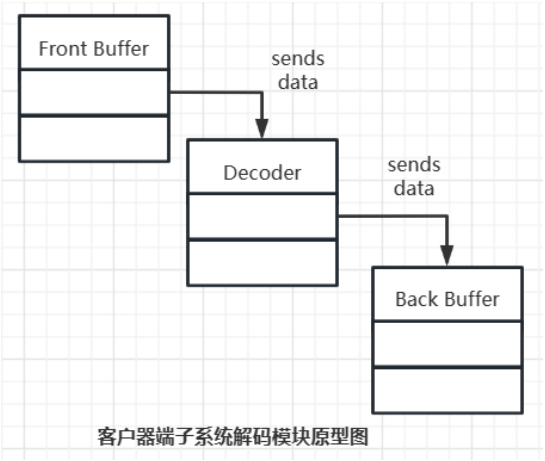
客户器端子系统网络模块原型图

2.3.2.11 中间层 - 客户端子系统的解码模块 (Decode)

此部分和服务器子系统的编码模块大致相同，同样是为了配合网络传输的功能性模块，细微差别在于数据流方向不同。



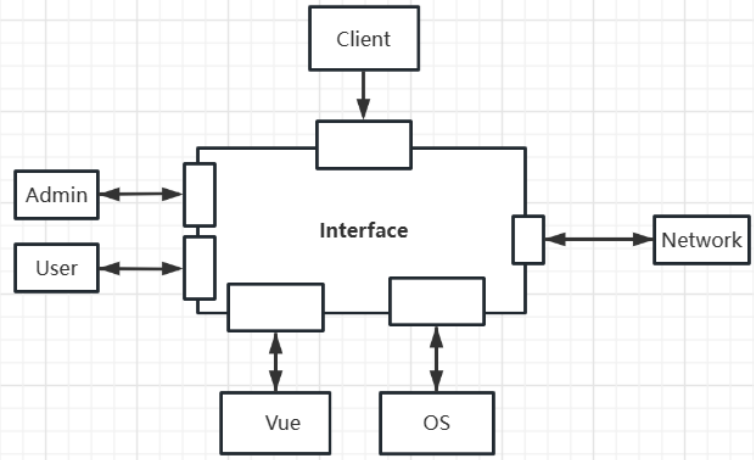
客户端子系统解码模块环境图



客户器端子系统解码模块原型图

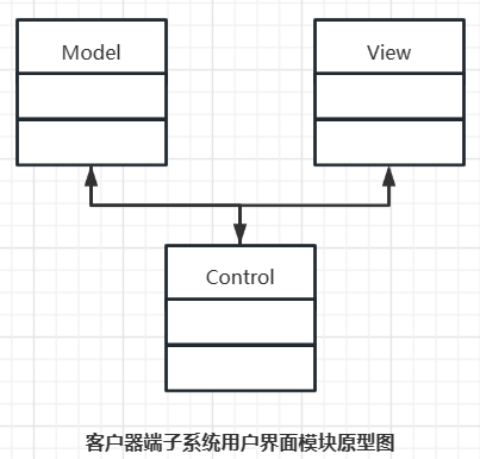
2.3.2.12 中间层 - 客户端子系统的用户界面模块 (Interface)

此模块使用的底层模块是 vue 组件库，这是用户和管理员都可以交互的模块，因此也需要根据用户需求进行测试和维护，达到更好的视觉效果。



客户端子系统用户界面模块环境图

原型方面，我们使用经典的 MVC 模式接管我们的设计方案，具体关系如下图所示：



客户器端子系统用户界面模块原型图

2.4 服务器端设计

2.4.1 服务器架构

服务器分为三个层次：表现层，业务逻辑层，数据访问层。

2.4.1.1 表现层

用于显示数据和接收用户输入的数据，为用户提供一种交互式操作的界面。根据用户的操作，调用业务逻辑层进行处理，并返回响应数据。

2.4.1.2 业务逻辑层

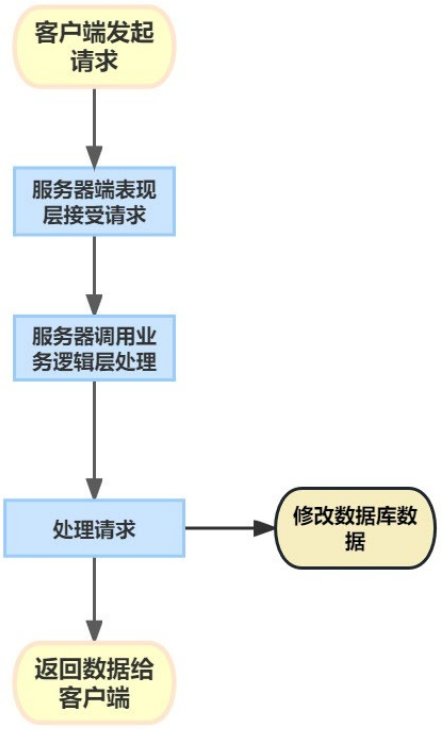
处于数据访问层与表示层中间，对于数据访问层而言，它是调用者；对于表示层而言，它却是被调用者。负责系统领域业务的处理，负责逻辑性数据的生成、处理及转换。对所输入的逻辑性数据的正确性及有效性负责。

2.4.1.3 数据访问层

负责对数据库数据的增删改查等操作，可以访问数据库系统、二进制文件、文本文档。

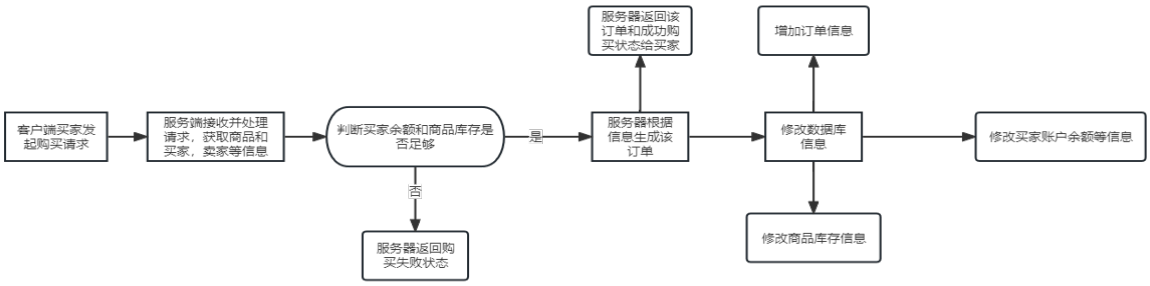
2.4.2 服务器处理流程

根据服务器端的分层，可以将其处理流程大致划分如下图：



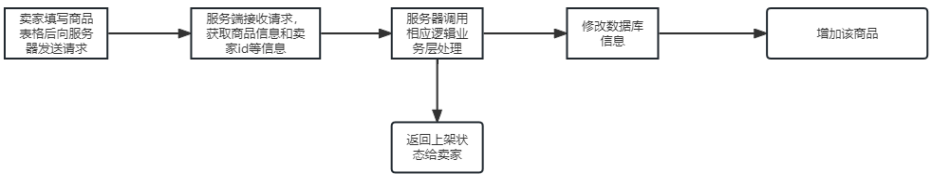
用户通过界面操作发送请求给服务器。服务器接收到请求后，调用相应的业务逻辑层对其处理。下面以将购物和上架货物的过程为例，来解释服务器的处理流程。

买家购买商品流程：



买家购买商品流程图

卖家上架商品流程：



卖家上架商品流程图

2.5 接口设计

2.5.1 用户接口

2.5.1.1 个人中心

/register：用于接收用户注册信息，并返回后台认证情况

/login：用于接收注册登录信息，并返回后台认证情况

/displayUserInformation：用于展示用户信息，并返回用户的信息包

/modifyUserInformation：用于修改用户信息，并返回后台修改情况

/UserShowLike：用于展示用户喜爱商品，并返回所有用户喜欢的商品数据包

/UserOrderShow：用于展示用户所有的交易订单，并返回用户喜爱的所有商品数据包

/UserGetBalance：用于展示用户的余额，并返回余额数据

/UserAddBalance：用于对用户的余额进行充值，并返回充值结果

2.5.1.2 卖家中心

/CreateGoods：卖家上架商品，并返回后台修改情况

/DeleteGoods：卖家下架商品，并返回后台修改情况

/ModifyGoods：卖家修改商品，并返回后台修改情况

/ShowUserGoods：卖家商品展示，并返回该卖家的所有商品数据包

/DeliveryGoods：卖家发货商品，并返回发货处理结果

2.5.1.3 交易市场

/ShowAllGoods：商品市场中，商品展示，返回商品市场中的所有商品数据包

/ShowGoodsByType：商品市场中，商品按类展示，返回商品市场中该类的所有商品数据包

/ShowGoodsByName：商品市场中，商品按名字展示，返回商品中与该名称匹配的商品数据包

/ShowGoodsById： 进入商品详情页时，通过页面 url 中的 id 调取商品，返回商品中与该 id 匹配的商品数据包

. /UserAddLike：商品市场中，用户添加自身喜爱的商品到个人中心，返回后台修改情况

/UserBuyGood：商品市场中，用户对特定的商品进行购买，并返回购买结果

2.5.2 管理员接口

/AdminShowUser：管理员界面中，后台返回所有的用户信息数据包

/AdminEditUser：管理员界面中，修改具体用户信息，后台返回修改结果

/AdminDeleteUser：管理员界面中，删除具体用户，后台返回修改结果

/AdminModifyGood：管理员页面中，编辑具体商品，后台返回修改结果

/AdminDeleteGood：管理员界面中，删除商品，后台返回修改结果

3. 设计模式

3.1 体系结构分析

本项目是围绕着数据的增删改查的数据中心模型，项目的主要操作是有关数据的操作。因此我们的设计模式也应该围绕着这一特点展开。同时，由于我们的项目围绕着交易市场展开，市场中的商品，用户，订单等都可以用面向对象的思想进行建模。

3.2 设计模式选用

设计模式是针对特定软件设计问题的通用解决方案，它们在软件工程领域中出现，并通过多年经验和应用不断发展和改进。设计模式提供了一种通用的术语和符号，能够帮助开发人员更清晰和明确地描述软件设计，并可极大地促进开发团队间的交流。按照设计模式去处理软件设计问题，可以提高软件开发效率、可读性、可维护性和可扩展性。在我们的项目中，我们采用了不少经典的设计模式来优化提高团队之间的沟通效率与协作能力。

根据《设计模式：可复用面向对象软件的基础》，可以将课程项目采用的设计模式分为“创建型模式”“结构型模式”与“行为型模式”。

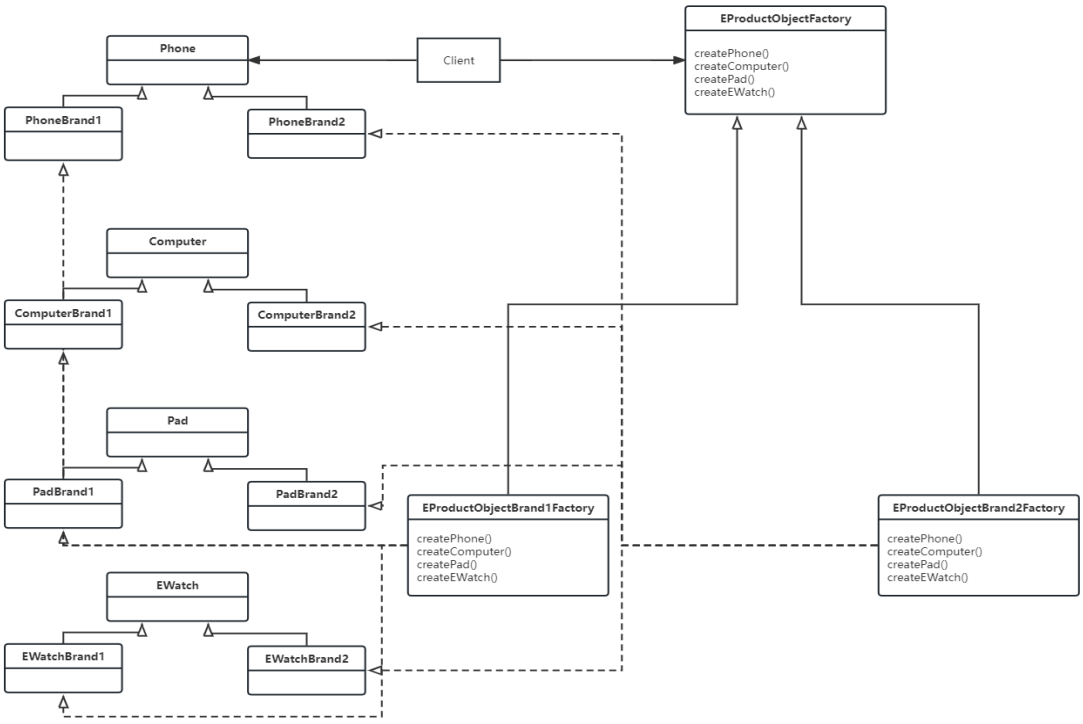
3.2.1 创建型模式

1. 抽象工厂（Abstract Factory）——对象创建型模式

抽象工厂能够提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。这非常适合用于系统内一系列相关产品的创建设计。

抽象工厂可将整个对象创建过程封装在工厂接口和工厂实现中，从而有效消除重复的代码，并提高代码复用率。在我们的项目中，市场内的产品主要分为“图书音像”“家居生活”“服饰箱包”和“电子产品”4 种类型，而这些商品大类可以认为是由具体种类的商品类组合实现的。这样的业务逻辑非常适合应用抽象工厂去实现。以“电子产品”类型为例，客户可以通过 EProductObjectFactory 进行生产 Phone、Computer、Pad、EWatch 等类型。以上的创建过程逻辑相似，而运用抽象工厂的设计模式可以大大提高代码的可重用性。

同时，抽象工厂通过工厂接口和工厂实现，可轻松添加和扩展新的对象类型，而无需更改现有对象的创建过程。同样以“电子产品”类型为例。假如在后续的开发设计中，我们想要在“电子产品”类型的商品中增加新品牌的一系列产品，那么我们不需要对代码进行大规模的修改，只需要在 EProductObjectFactory 中添加对应的一个子类，并在这个子类内实现具体的创建逻辑即可。这样的设计模式能够为项目的后续维护完善提供非常优秀的基础。



2. 生成器（Builder）——对象创建型模式

生成器能够将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。使用生成器模式，能够显著优化复杂对象的创建。

生成器允许我们在创建对象时根据需求灵活地添加/删除组件，这意味着使用生成器创建的对象具有更大的可配置性，能够适应不同的使用场景。在我们的项目中，商品类是一个比较复杂的对象，它包括商品 ID、商品名称、商品价格、商品类型、存货数量、销量、商品描述、销售者 ID、邮费等属性。而假如销售者在提供商品信息时，并没有提供完整的信息，那么可以利用生成器模式在创建商品时调整对应的属性组合，从而达到成功创建商品的目的。

相关的简略代码如下所示。

```
1.    相关的简略代码如下所示。
2.    import  java.util.UUID;
3.
4.    public  class  Good  {
5.            private  String  goodId;
6.            private  String  goodName;
7.            private  double  price;
8.            private  int  type;
9.            private  int  storageAmount;
10.           private  int  salesVolume;
11.           private  String  description;
12.           private  String  sellerId;
```

```
13.         private double postage;
14.
15.         // 构造函数
16.         public Good(String goodId, String goodName, double price, int type, int storageAmount, int salesVolume, String description, String sellerId, double postage) {
17.             this.goodId = goodId;
18.             this.goodName = goodName;
19.             this.price = price;
20.             this.type = type;
21.             this.storageAmount = storageAmount;
22.             this.salesVolume = salesVolume;
23.             this.description = description;
24.             this.sellerId = sellerId;
25.             this.postage = postage;
26.         }
27.
28.         // Getter和Setter方法
29.         // ...
30.
31.         //Builder设计模式
32.         public static class GoodBuilder {
33.             private String goodId = UUID.randomUUID().toString();
34.             private String goodName;
35.             private double price;
36.             private int type;
37.             private int storageAmount;
38.             private int salesVolume;
39.             private String description;
40.             private String sellerId;
41.             private double postage;
42.
43.             public GoodBuilder setGoodName(String goodName) {
44.                 this.goodName = goodName;
45.                 return this;
46.             }
47.
48.             public GoodBuilder setPrice(double price) {
```

```
49.                         this.price = price;
50.                         return this;
51.                     }
52.
53.                     public GoodBuilder setType(int type) {
54.                         this.type = type;
55.                         return this;
56.                     }
57.
58.                     public GoodBuilder setStorageAmount(int storageAmou
    nt) {
59.                         this.storageAmount = storageAmount;
60.                         return this;
61.                     }
62.
63.                     public GoodBuilder setSalesVolume(int salesVolume)
     {
64.                         this.salesVolume = salesVolume;
65.                         return this;
66.                     }
67.
68.                     public GoodBuilder setDescription(String descriptio
    n) {
69.                         this.description = description;
70.                         return this;
71.                     }
72.
73.                     public GoodBuilder setSellerId(String sellerId) {
74.                         this.sellerId = sellerId;
75.                         return this;
76.                     }
77.
78.                     public GoodBuilder setPostage(double postage) {
79.                         this.postage = postage;
80.                         return this;
81.                     }
82.
83.                     public Good build() {
```

```
84.                         return new Good(goodId, goodName, price,
    type, storageAmount, salesVolume, description, sellerId, postage);
85.                     }
86.             }
87.     }
```

我们利用上述代码来创建不同种类的商品，这使得我们不需要关心商品构造的具体过程与细节，简化了创建思路，同时也提高了提高代码的可重用性、可读性和可维护性。

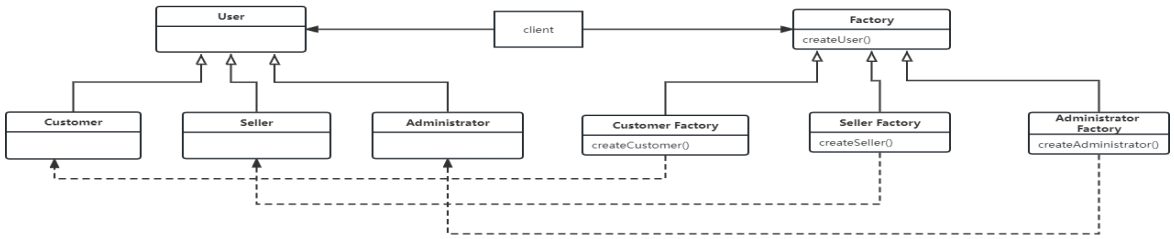
3. 工厂方法（Factory Method）——对象创建型模式

工厂方法能够定义一个用于创建对象的接口，让子类决定实例化哪一个类。简而言之，工厂方法使一个类的实例化延迟到其子类。这非常适合当一个类希望由它的子类来指定它所创建的对象的时候。

在我们的项目中，针对具有相似属性的用户分为三种：买家，卖家和管理员。为了创建这些用户，我们使用了工厂方法设计模式。在实现中，该工厂方法将延迟用户类的实例化，直到我们获得更多信息并可以确定用户应该属于哪个子类时。

我们定义了一个用户工厂类，它接受用户的一组初始信息，并根据需要创建适当的用户子类。这样，当工厂方法接收到关于新用户的信息时，它将检查一些逻辑以确定用户类的类型，并使用相应的子类进行实例化，然后返回该实例。

使用工厂方法的好处在于我们可以将用户类的创建和实现从客户端代码中分离出来，这使得我们可以更加灵活地管理代码，可以很方便地添加新用户类的子类或者调整对象的实现方式。同时，我们也可以更容易地测试代码。



4. 原型（Prototype）——对象创建型模式

原型这种设计模式能够用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。这非常适合用于需要频繁创建的对象，并且需要在不同的环境下复用现有对象的场景。在这种情况下，原型模式可以复用已有对象，从而节省了创建对象的时间和开销。同时，原型模式能够降低代码的耦合性，易于维护和升级，因此在大规模复杂应用系统中广泛应用。

在我们的项目中，并没有上述频繁创建对象的需求，因此项目程序没有涉及原型这一设计模式的相关实现代码。但是为了方便对象的拷贝复制，我们利用相关的拷贝构造函数，实现了创建一个新对象并复制原始对象的值。

拷贝构造函数适用于需要在不同的环境下复用现有对象，并且这些对象的创建不需要进行大规模性能优化的场合。在这种情况下，我们可以通过拷贝构造函数来实现对象的拷贝复制，从而避免手动复制对象值的繁琐过程，提高代码编写的效率。

通过综合考虑，在本项目中，我们采用拷贝构造函数进行对象的复制来进行相关代码的管理与维护。

5. 单件（Singleton）——对象创建型模式

单件模式能够保证一个类只有一个实例，并提供访问这个实例的全局点。单件模式可以防止多个实例被创建，从而减少了资源的消耗，也避免了对象状态和行为的分散。在程序开发中，开发者可以利用静态变量和枚举类型等方法来实现单件模式。在多线程环境中，多个线程可以共享同一个单件对象，避免了线程之间的竞争和锁定，能够提高程序的执行效率。

在我们的项目中，没有相关的类有唯一实例的需求，因此我们没有在实际代码中运用单件这一设计模式。

3.2.2 结构型模式

1. 适配器（Adapter）——类对象结构型模式

适配器模式的主要目的是将一个类的接口转换成客户端所期望的另一个接口，从而使原本不兼容的类能够协同工作。适配器模式非常适合用于旧代码的重构或在组件间接口不兼容时进行协调。

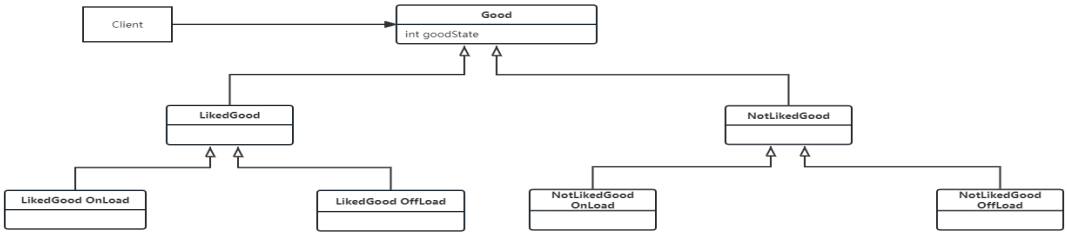
由于本项目由多人进行合作开发，在实际开发推进的过程中，可能会产生接口不兼容的问题，导致代码无法正常运行。此时，我们可以利用适配器模式进行调整与完善。假如我们原本设计了一个类 Original，而在后续项目推进的过程中，我们最终决定用另一个类 Lateral 去调用 Original，这就会造成类之间接口的不匹配问题。为了解决这个问题，我们可以再设计一个 Adapter 类来实现这两个类之间接口的匹配。

当然，上述是单向适配器的设计模式。如果有需求，也可以设计双向适配器。双向适配器可以将一个类转换为另一个接口，同时也可以将另一个接口转换为该类，从而实现双向转换。这种模式在需要在多个类之间进行复杂而灵活的对象转换时非常有用。

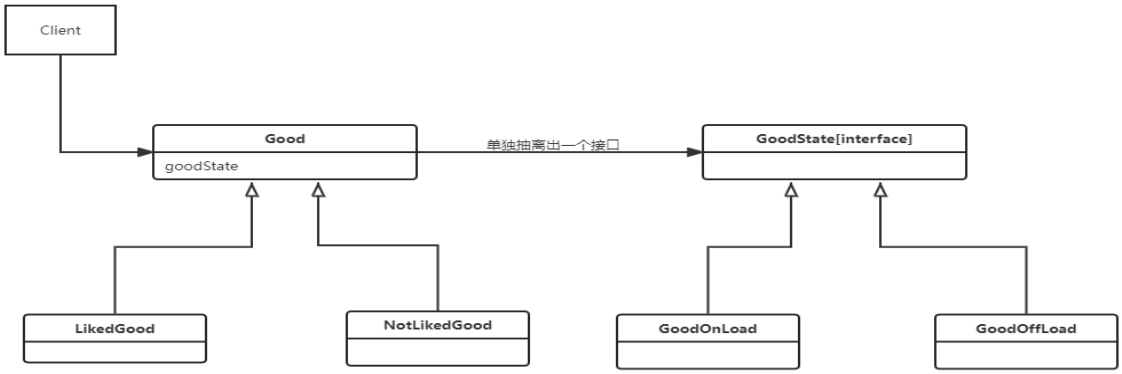
2. 桥接（Bridge）——对象结构型模式

桥接模式能够将抽象部分与它的实现部分分离，使它们都可以独立地变化。桥接模式的核心理念是"组合优于继承"，它通过将接口和实现分离开来，从而使得它们可以随意组合，并且能够独立地改变这两个部分中的任何一个而不影响到另一个。

在本项目中，商品类的 state 属性根据买方操作，可以分为未收藏和收藏这 2 种状态。而 state 属性根据销售者操作，可以分为已上架和已下架这 2 种状态。如果直接进行继承，那么整体结构会相对复杂。同时，这样的继承方式存在一定的冗余信息，非常不利于提高代码的可复用性。



如果采用桥接模式，那么继承模式会被优化到如下所示。



在采用桥接模式之后，我们能够对 Like 和 Load 层次结构进行各自的进一步扩充，这大大优化了代码之间的解耦合，使得模块化的思想原则在项目程序中得到了更好的体现。

3. 组合（Composite）——对象结构型模式

组合这一设计模式能够将对象组合成树形结构以表示“部分-整体”的层次结构，使得用户对单个对象和组合对象的使用具有一致性。组合模式中通常有两种类型的对象：叶子节点（Leaf）和组合节点（Composite）。叶子节点代表单个对象，而组合节点则包含叶子节点和其他组合节点。组合这一设计模式比较适合用于以下场景：系统内存在复杂对象，且该复杂对象是由多个系统内的简单对象组合而成。针对上述情况，组合模式可以统一接口。即使在处理复杂的树形结构时，用户也只需要调用同一个接口即可，而不需要关心具体的实现细节。

在我们的项目中，由于涉及到的对象彼此之间并没有包含和从属的明显关系，因此在项目实现中，并没有运用组合这一设计模式。

4. 装饰（Decorator）——对象结构型模式

装饰这一设计模式允许在运行时动态地为对象添加新的职责，同时又不改变其原有的结构和行为。这样可以在不修改原始代码的基础上进行功能扩展或修改，从而提高代码的可维护性和可复用性。与直接生成子类相比，装饰模式能够更加灵活地为对象添加功能。

在我们的项目中，由于我们实现的是一个基础的电子商务系统，涉及到的主要操作是商品买卖和金额充值等，因此没有明显的为对象添加新功能的需求。

但是我们设计保留了部分接口，可以在之后拓展商品促销价格计算的功能。而该功能可以利用装饰模式来进行高效合理的实现。在系统中，每个商品都可以看作是一个基础的组件，我们可以为其添加一些额外的促销策略和赠品策略，同时也可以根据不同的用户信息和订单信息，动态地为不同的商品添加不同的装饰器来实现更具体的促销策略和赠品策略。

大致的代码思路如下：

```
// 定义商品接口
Interface Good
        getName() // 获取商品名
        getPrice() // 获取商品价格
End Interface

// 实现商品接口的基类
Class BaseGood Implements Good
        String name // 商品名
        Decimal price // 商品价格

        Constructor(name, price) // 构造函数
                this.name = name
                this.price = price

        Function getName() // 获取商品名

        Function getPrice() // 获取商品价格

End Class

// 定义装饰模式的核心部分 - 装饰器基类
Abstract Class GoodDecorator Implements Good
        Protected good // 待装饰的商品

        Constructor(good) // 构造函数
                this.good = good

        Function getName() // 获取商品名

        Function getPrice() // 获取商品价格

End Class

// 一个具体装饰类 - 折扣商品类
Class DiscountGood Extends GoodDecorator
        Decimal discount // 折扣率

        Constructor(good, discount) // 构造函数
```

```
40.                 Super(good)  //  调用父类构造函数，传入待装饰商品
41.                 this.discount  =  discount
42.
43.         Function  getPrice()  //  获取商品价格
44.                 //  调用待装饰商品的 getPrice()方法，并乘以折扣率
45.                 Return  this.good.getPrice()  *  this.discount
46.         End  Function
47.  End  Class
48.
49.  //  另一个具体装饰类  -  赠品商品类
50.  Class  GiftGood  Extends  GoodDecorator
51.         String  gift  //  赠品
52.
53.         Constructor(good,  gift)  //  构造函数
54.                 Super(good)  //  调用父类构造函数，传入待装饰商品
55.                 this.gift  =  gift
56.
57.         Function  getName()  //  获取商品名
58.                 //  调用待装饰商品的 getName()方法，并在其后面添加赠品
    信息
59.                 Return  this.good.getName()  +  ″  +  ″  +  this.gift
60.         End  Function
61.  End  Class
```

抽象类 GoodDecorator 作为装饰类，实现了 Good 接口。DiscountGood 和 GiftGood 分别作为 GoodDecorator 类的子类，实现了不同的装饰行为。

5. 外观（Facade）——对象结构型模式

外观这一设计模式能够为复杂的子系统提供一个简化的接口。这个接口隐藏了子系统的复杂性和让其易于使用。这一设计模式非常适合用于系统子模块功能比较复杂和数量较多的情况。在这种场景下，外观模式有助于松耦合系统中的对象，并将系统分解为更小的模块，这些模块可以更易于管理、维护和升级。外观模式可以使得各个子系统拥有自己的独立实现，系统整体表现为一致的接口，对客户端隐藏了子系统的复杂性。

同时，在多人协作的项目中，外观模式也能在一定程度上实现模块兼容协调的问题。

6. 享元（Flyweight）——对象结构型模式

享元这一设计模式通过共享细粒度对象来实现对大量对象的有效管理，以减少程序运行时的内存开销。享元模式的核心思想是共享对象，它在多次使用相同的对象时可以避免重复对象的创建，只需引用已有对象即可。这一设计模式比较适合用于存在大量类似对象或存在不可变对象的场景。在这些情况下，使用共享的方法可以在保证安全和效能的前提下降低开销。

在我们的项目中，由于涉及到的对象彼此之间相互独立且大部分属性可变，因此在项目实现中，并没有运用享元这一设计模式。

7. 代理（Proxy）——对象结构型模式

代理这一设计模式能够在访问对象时引入一定程度的间接性，从而可以在不改变对象的情况下控制访问。代理模式可以通过定义代理类来实现，代理类与所代理对象具有相同的接口，从而可以对外表现为同样的对象。这一设计模式非常适合用于以下场景：需要访问远程对象、需要限制对真实对象的访问、需要在访问真实对象时添加额外的行为（例如记录日志、计算执行时间等）等。

在我们的项目中，由于并没有涉及到上述的相似场景，因此在项目实现中，并没有运用代理这一设计模式。

3.2.3 行为型模式

1. 职责链（Chain of Responsibility）——对象行为型模式

职责链这一设计模式能够使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合关系。这些对象会被连成一条链，并沿着这条链传递该请求，直到有一个对象处理它为止。

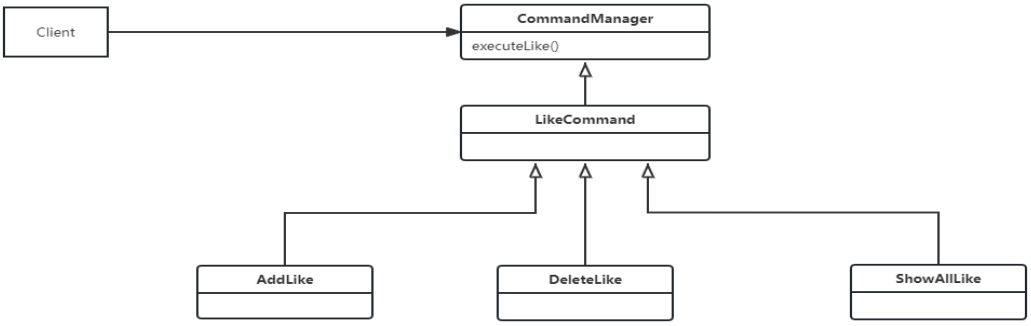
职责链模式能够降低耦合，从而可以降低系统的耦合度和增加系统的可维护性。同时，职责链模式可以动态添加和删除处理器，从而提高系统的灵活性和可扩展性。此外，职责链模式可以将类的功能单一化，使得每个类只需要专注于处理自己的角色，简化了对象的复杂度。
在我们的项目中，由于并没有成链的业务逻辑关系，因此在项目实现中，并没有运用职责链这一设计模式。

2. 命令（Command）——对象行为型模式

命令这一设计模式能够将请求转换为一个包含请求信息的对象，从而允许发送方和接收方彻底解耦。它允许请求的发送者与接收者完全解耦，使得发送者不需要知道接收者的存在，而接收者也不需要知道发送者的存在。该模式将请求封装在一个对象中，并将参数和操作绑定在一起。这一设计模式非常适合用于需要支持撤销和重做操作、需要在系统中基于事件驱动的架构实现等场景。

命令模式通过将命令封装为独立的对象，使得其在不同的请求和操作之间能够提供更好的参数化和抽象。另外，命令模式也能够增强灵活性、降低耦合度、简化代码设计等方面都有所改善。

在我们的项目中，用户收藏商品这一行为是通过命令模式实现的。



在购物流程中每个用户执行商品收藏操作时，可以创建相应的具体命令对象，然后将命令对象作为参数传递给 CommandManager 对象的 executeLike()方法，执行命令时通过 CommandManager 对象调用 executeLike()方法触发具体命令对象的操作。

3. 解释器（Interpreter）——类行为型模式

解释器这一设计模式定义了一种语言来解释表达式，其主要目的是为给定语言定义一种文法，并按照该文法解释语言中的句子。这一设计模式可以简化语法树中的节点数量，从而简化代码的实现。同时，解释器通过把复杂的语言文法分解为一些简单的小部件，可以为开发人员提供更好的掌握语言的体系结构。

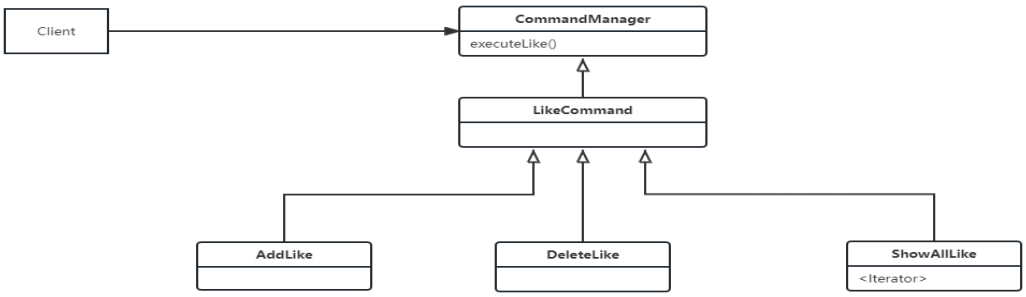
在我们的项目中，由于并没有涉及到对某种语言的文法规定需求，因此在项目实现中，并没有运用代理这一设计模式。

4. 迭代器（Iterator）——对象行为型模式

迭代器这一设计模式能够提供一种方法顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示。这种模式提供了一种简单而通用的方式，用于在访问集合中的元素时避免代码中的大量重复。

迭代器模式的核心是创建一个迭代器类，负责遍历集合并提供对其元素的访问操作（比如 next()方法）。然后，客户端代码使用该迭代器类遍历集合，而不必关心其内部实现。这可使代码更加干净，可维护性更高。而且，迭代器可以提供不同的遍历方式，例如倒序遍历。

在我们的项目中，显示用户收藏的商品这一行为是通过迭代器模式实现的。



和直接使用 for 循环相比，使用迭代器可以避免将整个集合都加载到内存中，从而提高了代码的运行效率和内存使用效率。此外，迭代器可以让代码更加直观，更加简洁易懂，特别是遍历嵌套结构时，使用迭代器可以使代码更加清晰。

5. 中介者（Mediator）——对象行为型模式

中介者这一设计模式通过用一个中介对象来封装一系列的对象交互。这使得各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。这一设计模式比较适合用于以下场景：系统涉及一组对象，其中每个对象都需要相互通信，但不希望彼此之间直接耦合。这时，可以引入一个中介者对象，负责协调对象之间的交互。可以将每个对象的消息转发给中介者，由中介者处理和分发，而不是互相之间直接通信。

使用中介者模式不仅能够减少对象之间的直接耦合，还能提高代码可重用性。同时，中介者可以对对象之间的交互进行统一管理，这使得对代码进行修改和维护更加便捷。

在我们的项目中，买家与卖家通过商品进行交互，由于此处的交互较为简单直接，因此并没有设计中介者来介入。如果之后有更复杂的交互功能待开发，那么可以在此处引入中介者来优化整体代码。

6. 备忘录（Memento）——对象行为型模式

备忘录这一设计模式能够在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。因此，系统就具备了将该对象恢复到原先保存的状态的能力。使用这一设计模式还能隐藏状态的实现细节：通过备忘录对象，状态信息被隐藏在内部，其他对象不能直接访问，提高了程序的安全性。同时，备忘录模式支持撤销操作，以实现类似“撤销”和“重做”的功能。

在我们的项目中，不允许购买操作、充值操作的“撤回”，因此在项目实现中，并没有运用备忘录这一设计模式。

7. 观察者（Observer）——对象行为型模式

观察者这一设计模式使得一个对象的状态变化能够通知其他依赖于该对象的对象。观察者模式提供了一种松散耦合的方式，使得我们可以在不影响其他对象的情况下，更改一个对象的状态。

在观察者模式中，存在一个主题对象（也称为被观察者或可观察者对象），其状态的变化可能会影响其他对象。此外，还存在一组观察者对象，它们依赖于主题对象的状态并希望在状态变化时得到通知。当主题对象的状态发生变化时，通常会发送通知给所有的观察者对象，以便它们可以更新自己。

在我们的项目中，买家购买商品，卖家销售商品，两者通过商品进行交互，并没有存在更新通知的业务需求，因此在项目实现中，并没有运用观察者这一设计模式。

8. 状态（State）——对象行为型模式

状态这一设计模式允许对象在其内部状态发生变化时改变其行为。状态模式将对象的行为委托给表示其内部状态的一个或多个对象，从而实现状态转换时的行为变化。使用状态模式的核心思想是将与特定状态相关的行为分离到一个状态类中，以便对象在其内部状态发生变化时可以更改其行为而不是通过大量条件语句来实现。

在网站搭建中，TCP 连接是典型的运用状态模式的例子。TCP 连接是一种有序、可靠、双向、基于字节流的面向连接协议。在 TCP 连接的建立、传输和终止过程中，客户端和服务器端会根据不同的阶段采用不同的状态。因此，TCP 连接的不同阶段可以使用状态模式来进行有用的建模。

TCP 连接通常会经过以下三个阶段：

建立连接（establishment）

在建立连接时，TCP 连接的状态由 CLOSED 转变为 SYN_SENT。在这个过程中，客户端向服务器端发送一份 SYN 请求报文，提示自己想要开始一个连接。

数据传输（data transfer）

在数据传输时，TCP 连接的状态被改为 ESTABLISHED，即客户端和服务器端之间的连接已经建立起来。在这个状态下，数据不断地在连接上进行传输。

断开连接（termination）

在断开连接时，TCP 连接的状态由 ESTABLISHED 转变为 FIN_WAIT_1，即开始发送关闭连接报文的一方处于“半关闭”状态。随后，另一方收到“关闭连接”报文时会进入 TIME_WAIT 状态，等待最终关闭。最终，双方都进入 CLOSED 状态，连接关闭完成。

状态模式可以将每个阶段视为一个单独的状态对象，这个对象实现了一个公共接口，其中包括能够表示当前状态的所有方法。当 TCP 连接从一个状态转换到另一个状态时，将把其状态更改为新状态的对象。

9. 策略（Strategy）——对象行为型模式

策略这一设计模式定义了一系列的算法，把它们一个个封装起来，并且使它们可相互替换。这种做法使得算法可独立于使用它的客户而变化。在策略模式中，一个算法被封装在一个对象中，并且可以由其他对象调用。这些对象可以共享相同的算法，也可以使用不同的算法。使用策略模式能够允许客户端在运行时选择算法，这可以根据不同的输入参数动态地选择算法。

在我们的项目中，并没有涉及到多种算法选择的业务需求，因此在项目实现中，并没有运用策略这一设计模式。

10. 模板方法（Template Method）——类行为型模式

模板方法这一设计模式能够定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。它使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

在用户注册登录模块，注册与登录属于两种操作，但是其实现逻辑存在共通之处。可以通过如下思路用模板方法实现用户登录注册。

```
1.  // 抽象类
2.  abstract class User {
3.          // 输入用户信息（抽象方法）
4.          abstract void inputUserInfo();
5.
6.          //处理用户信息（抽象方法）
7.          abstract void handleInfo();
8.
9.          // 登录或注册流程（模板方法）
10.         public final void loginOrRegister() {
11.                 inputUserInfo();
12.                 handleInfo();
13.         }
14. }
15.
16. // 具体子类--登录
17. class Login extends User {
18.         // 输入用户信息
19.         @Override
20.         void inputUserInfo() {
21.                 ...
22.         }
```

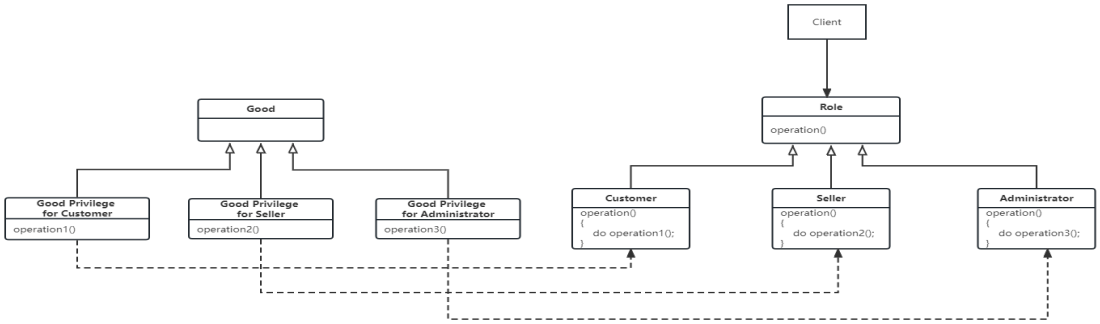
```
23.
24.         //  处理用户信息
25.         @Override
26.         void  handleInfo()  {
27.                 ...
28.         }
29.  }
30.
31.  //  具体子类--注册
32.  class  Register  extends  User  {
33.         //  输入用户信息
34.         @Override
35.         void  inputUserInfo()  {
36.                 ...
37.         }
38.
39.         //  处理用户信息
40.         @Override
41.         void  handleInfo()  {
42.                 ...
43.         }
44.  }
```

上述的设计思路能够提高代码复用性，降低代码维护成本，同时提供了更好的扩展性与维护了一致性。

11. 访问者（Visitor）——对象行为型模式

访问者这一设计模式能够表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。该模式定义了一个称为访问者的新对象，该对象通过对其他对象执行操作来更改其行为。访问者模式最适合于处理复杂的结构而不影响其元素的类层次结构。

在我们的项目中，存在三种身份的用户：顾客、销售者与管理员。不同用户的权限不同，因此他们的可执行操作也并不相同，用访问者模式可以较好地实现该逻辑。



使用访问者模式可以将数据结构和操作分开，使得代码更加清晰、易于扩展和维护。而且，假如需要新增操作，也只需要添加相应的访问者，不需要修改原有代码，符合 “开闭原则”。总体而言，访问者模式适用于大型复杂结构的数据操作，可以避免大量嵌套的条件语句。

4. 测试

4.1 测试概要

结合前期的《软件需求规格说明书》和《软件工程总体设计报告》所确定的功能模块，以及测试本身所设计到的方面，拟将从如下角度对该软件做出详细的测试。在接下来的测试文档里面，会以各种功能模块进行测试，在模块里面，会涵盖表中所示的测试内容

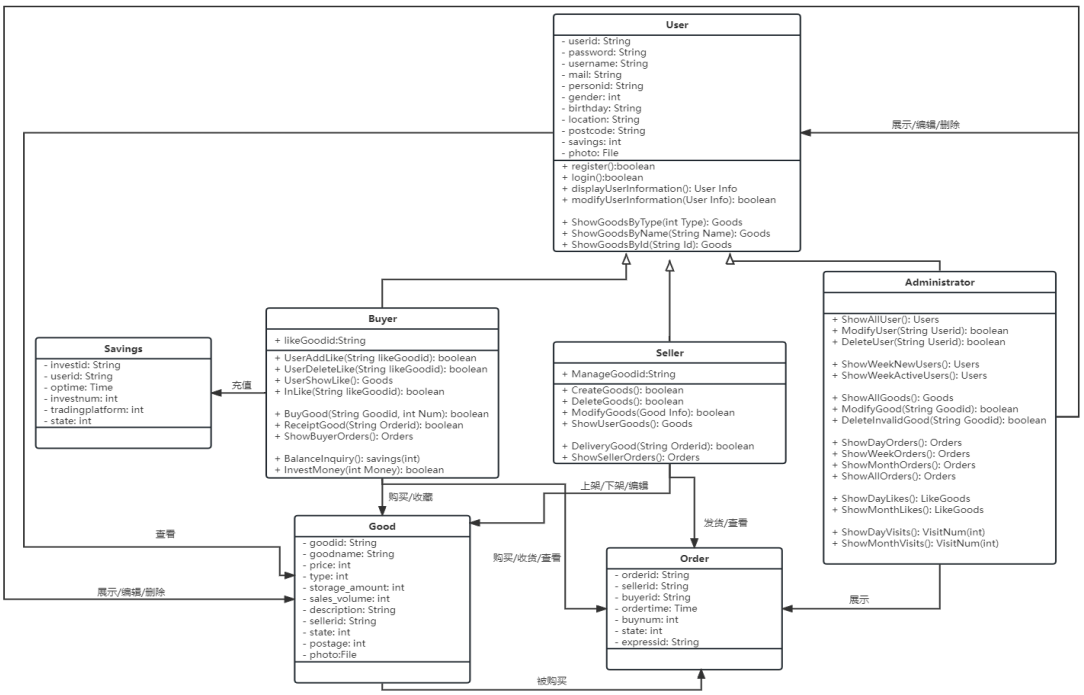
测试项目名称	测试目的	测试内容
面向对象测试	针对每一个类，测试是否设计正确	测试整体功能
功能验证测试	利用黑盒测试系统功能是否齐全，各个功能是否正确执行	登录注册模块 个人中心模块 商品市场模块 卖家中心模块 管理员模块

边界测试	测试程序对边界情况是否正确处理	登录注册模块 个人中心模块 商品市场模块 卖家中心模块 管理员模块
压力测试	测试系统在高负载情况下的功能和性能的承受情况	登录注册 同时购买商品 在线浏览商品 余额变更测试
用户接口测试	测试用户能否通过网页界面完成想要执行的操作	登录界面 注册界面 个人中心界面 卖家中心界面 商品市场界面 管理员界面

4.2 面向对象测试

4.2.1 系统整体架构

电子商务系统对应的基本类图如下所示。



在本系统内，用户类是基类，它的 3 个子类代表了 3 种不同的角色：买家、卖家、管理员。前两种角色是本系统的核心角色，本系统最重要的交易部分与这两类角色息息相关。另外，商品类也是本系统内非常重要的类，买家类与卖家类通过商品类产生一系列的交互。而在买家购买商品的同时，也会产生订单。为了更加合理高效地管理订单，本系统创建了订单类。为了能够实现购买操作，买家需要对自身的账户进行充值。由于充值也是系统内非常重要的业务之一，因此本系统创建了充值类。管理员作为本系统内权限最高的角色，拥有对其他类的编辑等权限。

面向对象测试主要根据上述类图，对系统涉及到的类进行逐一的功能测试。

4.2.2 具体类测试

4.2.2.1 单元测试

1. 用户基类

测试内容	测试结果
用户注册	通过
用户登录	通过
显示用户个人信息	通过
修改用户个人信息	通过
根据商品类型查询商品	通过

根据商品名称查询商品	通过
根据商品 id 查询商品	通过

2. 买家类

测试内容	测试结果
添加商品收藏	通过
删除商品收藏	通过
展示商品收藏	通过
查询商品是否在收藏内	通过
购买商品	通过
修改收货状态	通过
展示全部订单	通过
查询账号余额	通过
账号充值	通过

3. 卖家类

测试内容	测试结果
上架商品	通过
下架商品	通过
修改商品信息	通过
展示管理的全部商品	通过
订单发货	通过
展示全部订单	通过

4. 管理员类

测试内容	测试结果
展示全部用户	通过
修改用户信息	通过

删除用户	通过
显示本周新增用户信息	通过
显示本周活跃用户信息	通过
展示全部商品	通过
修改商品信息	通过
删除违规商品	通过
显示本日订单	通过
显示本周订单	通过
显示本月订单	通过
显示全部订单	通过
显示本日收藏	通过
显示本月收藏	通过
显示本日浏览量	通过
显示本月浏览量	通过

5. 商品类

由于商品类的相关操作直接在用户类部分实现，因此商品类内部仅有相关构造函数。

6. 订单类

由于订单类的相关操作直接在用户类部分实现，因此订单类内部仅有相关构造函数。

7. 存储类

由于存储类的相关操作直接在用户类部分实现，因此存储类内部仅有相关构造函数。

4.2.2.2 类模型一致性测试

为了确保类与类之间可以正常合作和相关接口的匹配，需要对所有类的 CRC 模型进行测试。下面列举了需要类间合作才能实现的测试。

1. 用户基类

重点测试用户基类能否进行不同类型的商品查询操作。

类：User	
编号：CLASS-1	
描述：本系统内全部的角色，包括买家、卖家、管理员。	
功能	合作类
根据商品类型查询商品	Good
根据商品名称查询商品	Good
根据商品 id 查询商品	Good

2. 买家类

重点测试买家类能否购买商品、修改展示订单以及进行账户充值。

类：Buyer	
编号：CLASS-2	
描述：使用系统进行商品购买的角色，继承自 User 类。	
功能	合作类
购买商品	Good, Order
修改收货状态	Order
展示全部订单	Order
账户充值	Savings

3. 卖家类

重点测试卖家类能否进行商品的相关操作，以及对订单进行发货查看处理。

类：Seller	
编号：CLASS-3	
描述：使用系统进行商品销售的角色，继承自 User 类。	
功能	合作类
上架商品	Good
下架商品	Good
修改商品信息	Good

展示管理的全部商品	Good
订单发货	Order
展示全部订单	Order

4. 管理员类

重点测试管理员类能否对用户、商品、订单等信息进行相关权限的操作和展示。

类：Administrator	
编号：CLASS-4	
描述：使用系统进行信息统计、高层操作的角色，继承自 User 类。	
功能	**合作类**
展示全部用户	Buyer, Seller
修改用户信息	Buyer, Seller
删除用户	Buyer, Seller
显示本周新增用户信息	Buyer, Seller
显示本周活跃用户信息	Buyer, Seller
展示全部商品	Good
修改商品信息	Good
删除违规商品	Good
显示本日订单	Order
显示本周订单	Order
显示本月订单	Order
显示全部订单	Order
显示本日收藏	Buyer, Good
显示本月收藏	Buyer, Good
显示本日浏览量	User, Good
显示本月浏览量	User, Good

5. 商品类

类：Good
编号：CLASS-5
描述：代表系统内的一种商品。

6. 订单类

类：Order
编号：CLASS-6
描述：代表系统内的一次订单。

7. 存储类

类：Savings
编号：CLASS-7
描述：代表系统内的一次充值记录。

4.3 功能验证测试

4.3.1 登录注册模块

功能名称	操作	预期输出	实际输出
用户登录	正确的用户名、密码	直接跳转到总主界面	与预期输出相符
管理员登录	正确的管理员用户名、密码	跳转到管理员页面	与预期输出相符
登录	错误的用户名 （或密码或用户类型）	在登录框下方会显示出 登录失败的提示信息	与预期输出相符

注册	注册原来不存在的号码	在注册框下方会显示出注册成功的提示信息	与预期输出相符
注册	注册已有的号码	在注册框下方会显示出注册失败的提示信息	与预期输出相符

4.3.2 个人中心模块

功能名称	操作	预期输出	实际输出
基本信息查看	点击信息栏目下的基本信息	基本信息发生变化	与预期输出相符
信息修改	输入想要修改的信息，点击“提交修改”	基本信息被修改	与预期输出相符
头像修改	上传想要修改的图像，并点击“确定”	头像被修改	与预期输出相符
金钱充值	可以查看余额数目，选择充值金额并充值	充值成功	与预期输出相符
购物管理	查看被收藏的商品，可以通过该点击收藏的商品跳转到商品页面	显示收藏商品，成功通过收藏商品项目跳转到商品详情页面	与预期输出相符
订单管理	查看当前订单	显示当前用户的订单	与预期输出相符

4.3.3 商品市场模块

功能名称	操作	预期输出	实际输出

商品显示	进入商品市场页面	下方商品展示区域显示所有类别商品及信息	与预期输出相符
商品分类查看	点击左侧分类栏处的商品类别	下方商品展示区域显示特定类别商品	与预期输出相符
推荐广告展示	点击左右滑动按钮可以查看广告	广告栏处显示广告	与预期输出相符
开业福利查看	点击首页右侧“开业福利”按钮	页面上方显示开业福利	与预期输出相符
商品详情查看	点击商品展示区域展示的商品	跳转到商品详情页面	与预期输出相符

4.3.4 卖家中心模块

功能名称	操作	预期输出	实际输出
上架商品	按要求输入商品基本信息	提示商品成功上架	与预期输出相符
编辑商品	按需求修改商品信息	提示修改成功	与预期输出相符
下架商品	删除已上架的商品	提示商品成功下架	与预期输出相符
显示所有商品	点击对应商品展示	显示所有已上架商品	与预期输出相符
查询所有订单	点击对应订单查询	显示所有订单信息	与预期输出相符
处理未发货订单	点击该订单进行处理	提示已经成功处理，订单状态改为已处理	与预期输出相符

4.3.5 管理员模块

功能名称	操作	预期输出	实际输出

销售信息	点击对应的表格	表格数值联动变换	与预期输出相符
商品信息	点击对应商品	显示商品对应信息	与预期输出相符
用户展示	点击对应的用户	显示对应的用户信息	与预期输出相符
用户编辑	点击用户并编辑其信息	提示用户信息修改成功	与预期输出相符
用户删除	删除已存在的用户	提示用户删除成功	与预期输出相符
用户添加	添加一位新用户	提示用户创建成功	与预期输出相符

4.4 边界测试

4.4.1 登录注册模块

功能名称	操作	预期输出	实际输出
用户登录	输入错误的用户名或密码	出现报错提示	与预期输出相符
管理员登录	输入错误的用户名或密码	出现报错提示	与预期输出相符
注册	注册的格式出现各类错误	注册时会出现自动报错，并且直接报错	与预期输出相符

4.4.2 个人中心模块

功能名称	操作	预期输出	实际输出
信息修改	输入不符合规则的信息	出现报错提示，信息修改失败	与预期输出相符
头像修改	上传不符合格式或大小要求的图片	出现报错提示，头像修改失败	与预期输出相符
金钱充值	输入格式错误的充值金额	出现报错提示，充值失败	与预期输出相符

购物管理	查看被收藏的商品，可以通过该点击收藏的商品跳转到商品页面	出现未登录或未收藏提示	与预期输出相符
订单管理	查看当前订单	出现未登录或无订单提示	与预期输出相符

4.4.3 商品市场模块

功能名称	操作	预期输出	实际输出
商品显示	进入商品市场页面	出现无商品提示	与预期输出相符
商品分类查看	点击左侧分类栏处的商品类别	出现无此类商品提示	与预期输出相符
开业福利查看	点击首页右侧“开业福利”按钮	出现“福利暂无”提示	与预期输出相符
商品详情查看	点击商品展示区域展示的商品	出现“该商品不存在”报错	与预期输出相符

4.4.4 卖家中心模块

功能名称	操作	预期输出	实际输出
上架商品	不输入任何商品信息提交	提示商品信息不能为空	与预期输出相符
编辑商品	未修改任何商品提交	提示商品修改成功	与预期输出相符

4.4.5 管理员模块

功能名称	操作	预期输出	实际输出
编辑用户	编辑用户使用了已存在的手机号	出现报错提示	与预期输出相符
添加用户	添加的用户手机号已存在	出现报错提示	与预期输出相符

删除用户	查看商品市场	删除用户对应的商品已经消失	与预期输出相符

4.5 压力测试

功能名称	操作	输出
注册	正常注册	系统运行正常
登录	正常登录	系统运行正常
登出	正常登出	系统运行正常
查看个人信息	正常查看个人信息	系统运行正常
编辑个人信息	正常编辑个人信息	系统运行正常
查看喜爱商品	正常查看喜爱商品	系统运行正常
查看个人订单	正常查看个人订单	系统运行正常
上架商品	正常上架商品	系统运行正常
查看商家订单	正常查看订单	系统运行正常
查看商品市场	正常查看商品市场	系统运行不正常
购物购买	正常购物购买	系统运行不正常
添加收藏商品	正常添加喜爱商品	系统运行正常

4.6 用户接口测试

4.6.1 登陆界面

功能名称	预期操作	实际操作
输入用户名、密码	用户操作方便	与预期相符
记住密码	用户下次登陆无需再次输入密码	与预期相符
登陆成功时显示登陆结果	用户易于理解	与预期相符

登陆失败时显示登陆结果	用户易于理解	与预期相符

4.6.2 注册界面

功能名称	预期操作	实际操作
输入用户名、密码、邮箱、昵称	用户操作方便	与预期相符
注册成功时显示登陆结果	用户易于理解	与预期相符
注册失败时显示登陆结果	用户易于理解	与预期相符

4.6.3 个人中心页面

功能名称	预期操作	实际操作
查看基本信息	用户易于查看自己的信息	与预期相符
修改基本信息	用户可以修改自己的信息	与预期相符
上传头像	用户易于创建和修改自己的头像	与预期相符
金钱充值	用户易于查看余额和充值	与预期相符
购物管理	用户易于查看和访问自己收藏的商品	与预期相符
订单管理	用户易于查看自己的订单状态	与预期相符

4.6.4 商品市场页面

功能名称	预期操作	实际操作
根据分类显示商品	用户易于找到自己想要的类别的商品	与预期相符
根据商品名称搜索商品	用户易于找到自己想要的名字的商品	与预期相符
查看开业福利	用户易于易于找到该按钮	与预期相符
查看商品基本信息	用户易于查看商品基本信息	与预期相符
进入商品详情页	用户易于通过点击商品图片进入商品详情页	与预期相符

4.6.5 商品详情页面

功能名称	预期操作	实际操作
查看商品详细信息	用户易于查看商品详细信息	与预期相符
选择商品数量	用户易于通过加减按钮或直接输入修改商品数量	与预期相符

收藏商品	用户易于易于找到该按钮	与预期相符
购买商品	用户易于找到该按钮	与预期相符

4.6.6 卖家后台界面

功能名称	预期操作	实际操作
查看所有上架商品	卖家易于查看所有已上架商品	与预期相符
编辑修改上架商品	卖家易于发现修改按键并修改	与预期相符
下架商品	卖家易于找到下架按键	与预期相符
处理订单	卖家易于发现未处理订单	与预期相符
查看所有订单	卖家易于查看所有订单	与预期相符

4.6.7 管理员界面

功能名称	预期操作	实际操作
销售信息	管理员易于查看当天销售信息	与预期相符
商品信息	管理员易于监督所有商品信息	与预期相符
用户展示	管理员易于查看所有用户	与预期相符
用户删除	管理员易于发现删除按键	与预期相符

4.7 对软件功能的结论

4.7.1 登录注册模块

该模块实现了用户登陆登出和不同用户登录到不同用户界面的功能。经过测试，功能正常，在性能上具有一定的稳定性和鲁棒性，并且与用户交互界面友好。

4.7.2 个人中心模块

该模块实现了用户的个人信息查看编辑，订单的查看，喜爱商品的查看处理功能。经过测试，功能正常，稳定性较强，与用户交互界面良好。

4.7.3 商品市场模块

该模块实现了商品市场的商品展示，商品搜索，商品购买的功能。经过测试，功能正常，并发性略差，与用户交互界面良好，美观。

4.7.4 卖家中心模块

该模块实现了卖家上传商品，商品查看，以及对订单的发货处理等功能。经过测试，功能正常，稳定性较强，与用户交互界面良好，美观。

4.7.5 管理员模块

该模块实现了整个交易市场的信息可视化，用户管理，商品管理的功能。经过测试，功能正常，稳定性较强，交互界面良好，美观。