

廈門大學



信息学院软件工程系

软件工程导论

题 目 概要设计说明书

班级组别 软件工程 2023 级 2 班 2-7组

小组成员 伊木然 庄子鲲 吴乐言 苏一涵 张翰文 张琳

任课老师 吴清强

提交时间 2025年 12 月 7 日

目录

1. 引言	3
1.1 编写目的	3
1.2 项目背景	3
1.3 定义	3
1.4 参考资料	3
2. 任务概述	4
2.1 目标	4
2.2 运行环境	4
2.3 需求概述	4
2.4 条件与限制	5
3. 总体设计	6
3.1 系统总体结构	6
3.2 运行环境及规范	10
3.2.1 硬件平台	10
3.2.2 软件平台	10
3.3 系统的数据流	13
4. 接口设计	16
4.1 外部接口	16
4.1.1 用户界面	16
4.1.2 软件接口	22
4.1.3 硬件接口	23
4.2 内部接口	23
5. 数据结构设计	24
5.1 逻辑结构设计	24
5.1.1 服务模块	24
5.1.2 售后模块	26
5.2 物理结构设计	27
5.3 数据结构与程序的关系	28
6. 运行设计	31
6.1 运行模块的组合	31
6.2 运行控制	31
7. 出错处理设计	32
7.1 出错输出信息	32
7.1.1 服务器错误	32
7.1.2 数据库错误	32
7.1.3 业务错误	33
7.2 出错处理对策	33
8. 安全保密设计	35
9. 维护设计	37

1. 引言

1.1 编写目的

本项目为厦门大学信息学院软件工程系软件工程专业大三上学期《面向对象分析与设计》、《软件工程导论》、《JavaEE 平台技术》课程大作业。本文档编写目的是对课程大作业中与售后模块、服务模块相关的需求进行系统性的整理与说明。

此次编写为模块第一次迭代，目前项目进度仍处于设计阶段，尚未开始实际代码的编写。根据课程要求，我们计划构建一个能支持高并发、大负载的电子商城系统，其命名为OOMALL。旨在为用户提供与主流电商平台（如京东）相似的购物流程与后台操作，如购物、售后处理以及服务相关操作。

本概要设计说明书服务于上述项目的设计与实现，其预期读者为参与本项目的开发人员及负责评审本项目的审核人员。

1.2 项目背景

- a. 本项目来源于《JavaEE 平台技术》、《软件工程导论》、《面向对象分析与设计》课程设计大作业需求
- b. 项目开发者与设计者：庄子鲲、张瀚文、张林、吴乐言、苏一涵、伊木然
- c. 客户端用户：有电商系统使用需求的用户，包括买家、卖家（商户）、服务商
- d. 后台管理：平台管理员，商户也可有限参与管理

1.3 定义

顾客：Customer，使用电商系统进行购买等功能的用户，是系统的主要用户

商户：Shop，即商户，是系统的第二主要用户

服务商：ProviderService，即服务商，主要参与的是售后与增值服务环节，是对商户/平台能力的补充，是系统的第三主要用户

管理员：Admin，电商平台的管理员，负责日常维护和监控，以及对用户的权限设置

最高管理员：ADMIN，电商平台创建之初的用户，最高权限的管理员，拥有所有权限

客户端：客户访问系统的途径，在本项目中客户端无用户界面，用户通过API 进行访问

服务端：集业务逻辑、数据存储等功能与一身的服务器

1.4 参考资料

- a. OOAD 课程文件《成绩计算办法 V2》
- b. Roger S.Pressman , Bruce R.Maxin. 软件工程 实践者的研究方法[M]: 机械工业出版社.2021.6
- c. B站JavaEE相关课程

2. 任务概述

2.1 目标

市场目标：本课程设计旨在构建一个高并发、大负载的电商后端系统，其出发点源于当前用户对网络购物日益增长的需求。在借鉴主流电商平台（如京东）业务模式的基础上，结合本学期所学的面向对象方法、设计原则、设计思想以及 JavaEE 平台相关技术，实现电商平台售后模块与服务模块的核心功能，并在此基础上进行适当扩展，使用户能够更加便捷、高效地完成各类电商相关操作，满足多样化的使用需求。

技术目标：针对客户端，确保用户可以正常完成注册、登录，并通过系统提供的 API 接口实现商品浏览、下单、支付以及售后等业务操作。针对后台管理系统，以控制端界面的形式呈现管理功能，通过大量测试用例对后端代码进行验证并接收测试反馈，支持管理者在控制端查看运行状态与业务数据，并执行相应的管理操作。针对服务器，采用华为云 Ubuntu 服务器作为运行环境，实现对业务数据的存储、共享与同步。

总体而言，系统建设以“前端请求 + 后端服务”为基本技术路径，通过合理的架构设计与技术选型，支撑高并发电商场景下的稳定运行与功能实现。

2.2 运行环境

操作系统：Ubuntu 18.04

支持环境：Nginx

数据库：MySQL 8.0、Redis 6.2.4

2.3 需求概述

本项目在 OOMALL 电商平台的总体框架下，重点完成“售后模块”和“服务模块”的设计与实现，为顾客、商户、服务商以及平台管理员提供完整的售后处理和增值服务支持，主要需求概括如下：

(黄色底色为非必做模块内容，灰色为必做模块内容)

1. **售后申请与处理流程**：支持顾客针对已购商品发起售后申请（退货、换货、维修、安装等），商户可查看、审核、取消和验收售后单，系统根据审核结果驱动后续物流、退款或重新发货等处理流程（并为后续仲裁提供依据）。
2. **各种查询功能**：对于售后模块、服务模块，每个展示列表信息的页面（包括不限于售后单、服务单、商户信息、服务商信息等）都应默认有全部列表查询功能，并在此基础上有以名称查询功能、以状态查询功能、以id查询功能等各种筛选查询的方式
3. **售后仲裁支持**：当顾客与商户在售后结果上存在争议时，顾客可以发起仲裁申请，商户需要应诉并提交相关材料，平台管理员负责审核仲裁请求并作出仲裁裁决，售后仲裁只允许进行一次，禁止二次仲裁。

4. **售后时效与限制**: 商品超出售后时限后, 顾客不能再发起售后申请; 售后过程中涉及的上门取件、寄回验收等流程, 只支持由系统指定的上门取件方式, 禁止顾客自行发货, 确保流程可控与信息可追踪。
5. **服务商与服务配置**: 商户可以为商品在不同地区配置对应的服务商及服务类型 (如寄修、上门服务), 保证在同一商品、同一地区下服务配置唯一且可管控, 商户在系统配置完后自动生成服务单派发。
6. **服务单管理与执行**: 服务单既可以来源于订单中的增值服务, 也可以来源于售后的寄修服务。系统需要支持服务商接收、处理、完成与撤销服务单, 并区分寄修与上门等不同服务类型, 对服务过程中的预约时间、服务地点等信息进行记录和追踪。
7. **服务合同管理**: 服务合同是商户与服务商合作关系的凭证。系统需支持商户与服务商之间订立、查询、取消/暂停和恢复服务合同。服务合同需记录双方信息、关联商品、服务要求、生效与暂停状态等, 并通过约束合同从“待确认”到“生效”“暂停”“结束”等状态的流转, 限制非合法状态下的操作。
8. **平台对服务商和商户的管控**: 平台管理员可审核商户与服务商开户与信息变更, 查询商户与服务商, 暂停或恢复商户与服务商; 从平台维度统一控制商户和服务商的业务资格, 保障服务质量与平台秩序。
9. **多角色协同与信息安全**: 售后与服务模块需支持顾客、商户、服务商、平台管理员多角色协同操作, 在接口和界面层面对敏感信息进行必要的脱敏和访问控制, 确保用户隐私不被越权访问, 为后续支付、清算等模块提供可靠的数据基础。

2.4 条件与限制

经费限制: 云服务器租用费用 (不足以购买很多服务器部署成庞大的集群)

开发期限: 一个月

硬件限制:

系统只能运行在指定的服务器上, 一般情况下, 拥有五台服务器, 其各自硬件限制如下:

Slave1: Ubuntu 18.04 服务器 2 核 4G 内存虚拟机一台, 支持图形界面

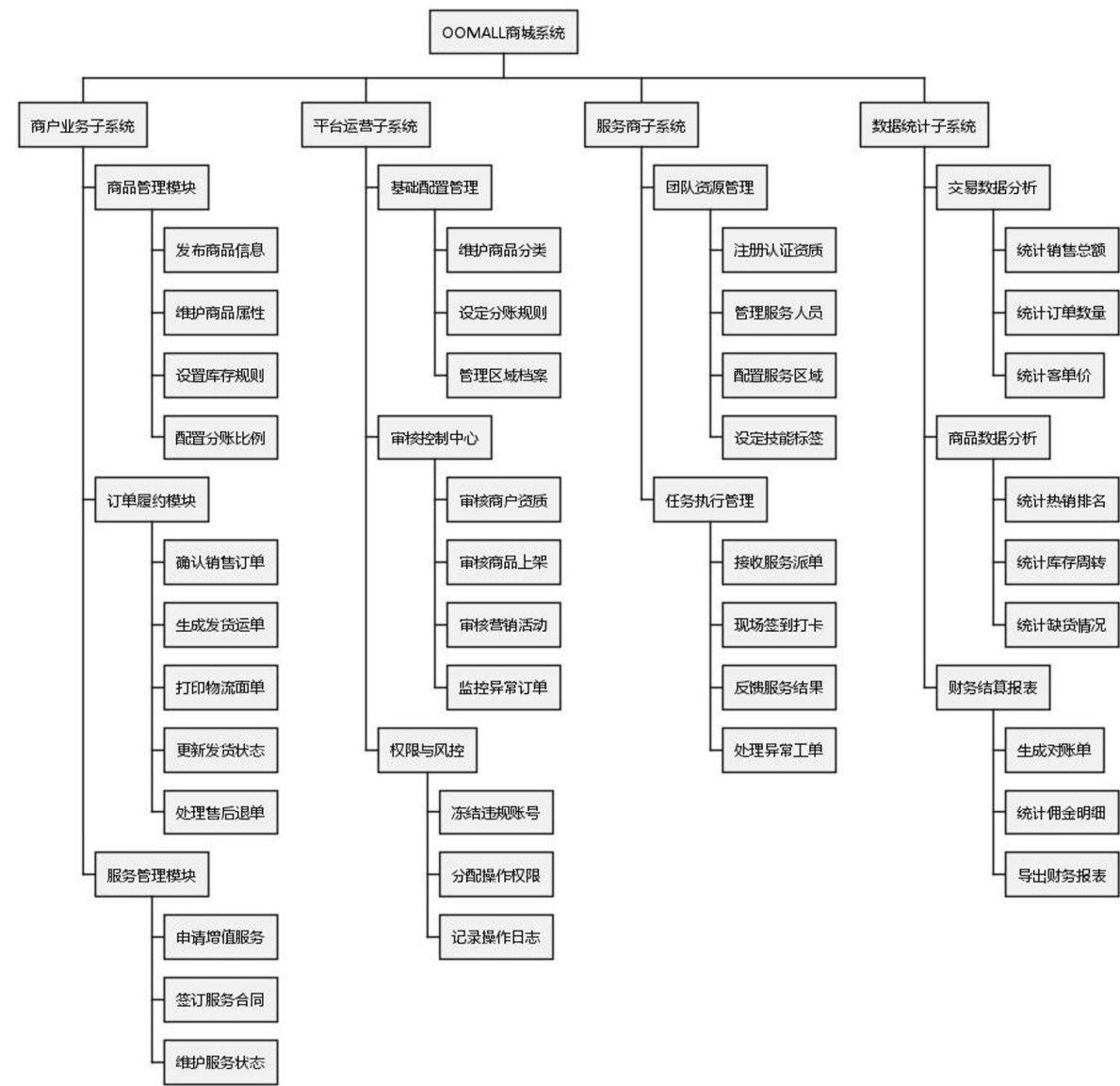
Slave2-5: Ubuntu 18.04 服务器 2 核 2G 内存虚拟机, 仅支持命令行界面所

有服务器需安装maven、jdk 17、git, 硬件必须支持以上需求

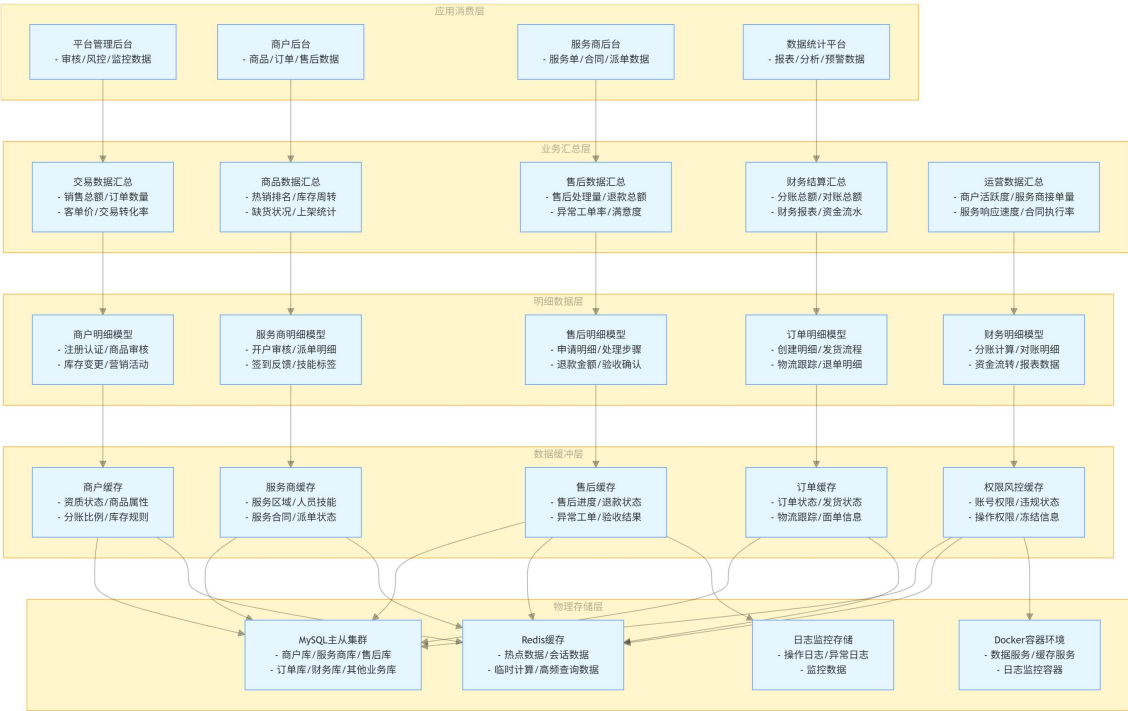
3 . 总体设计

3.1 系统总体结构

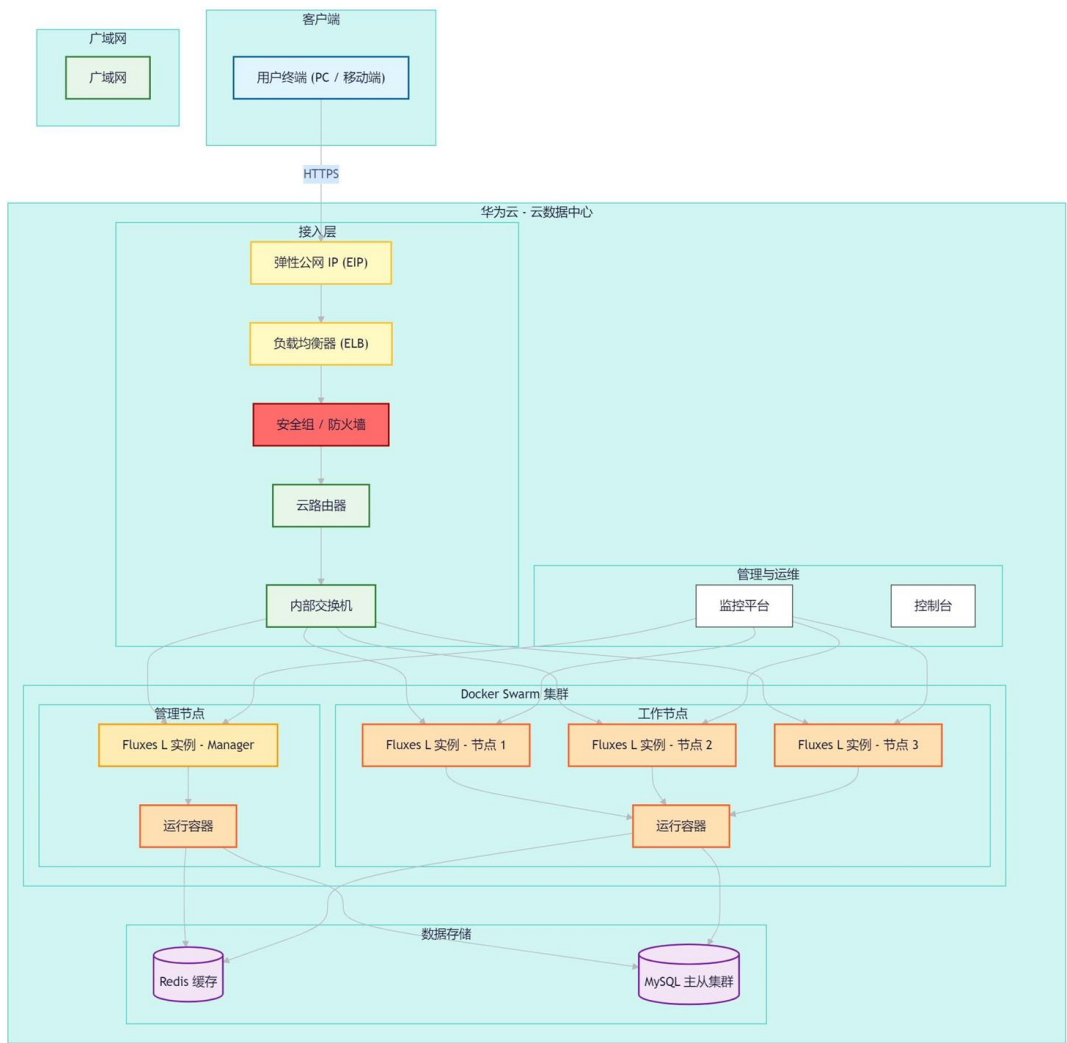
逻辑结构错误！未定义书签。



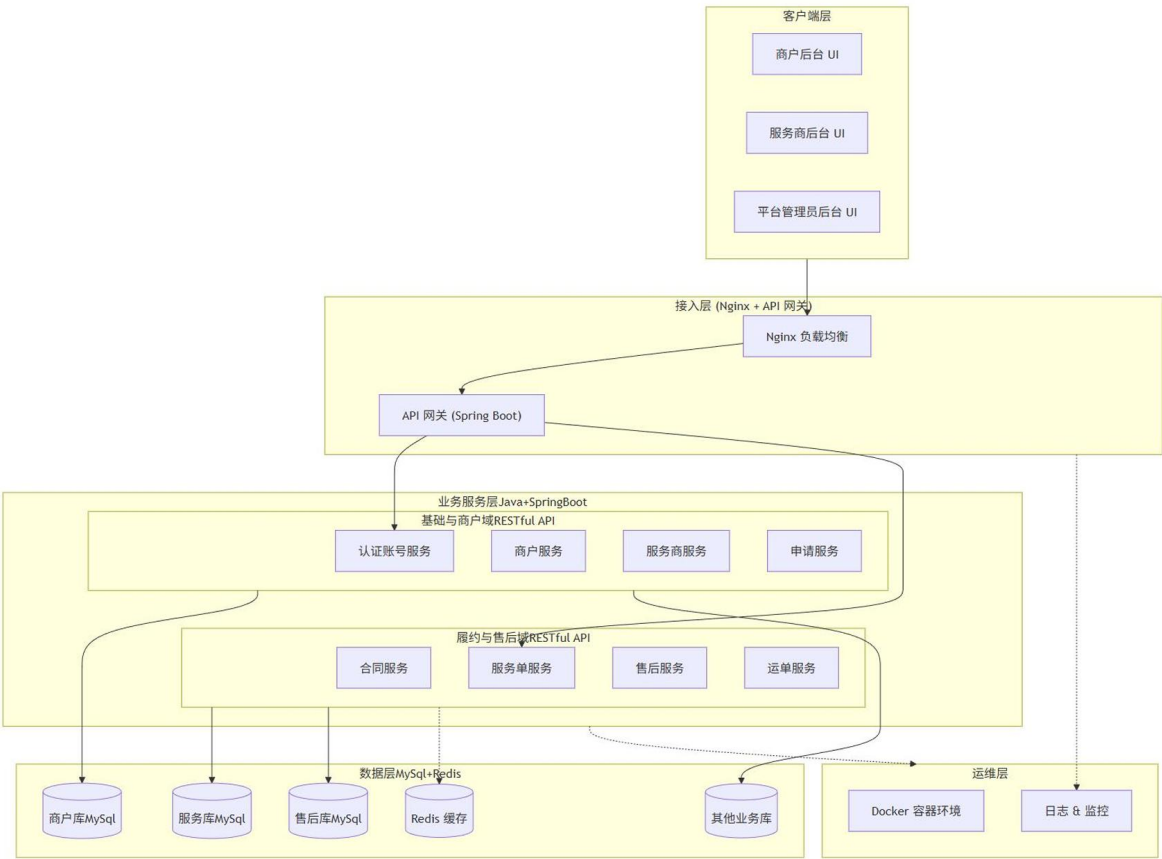
数据结构错误！未定义书签。



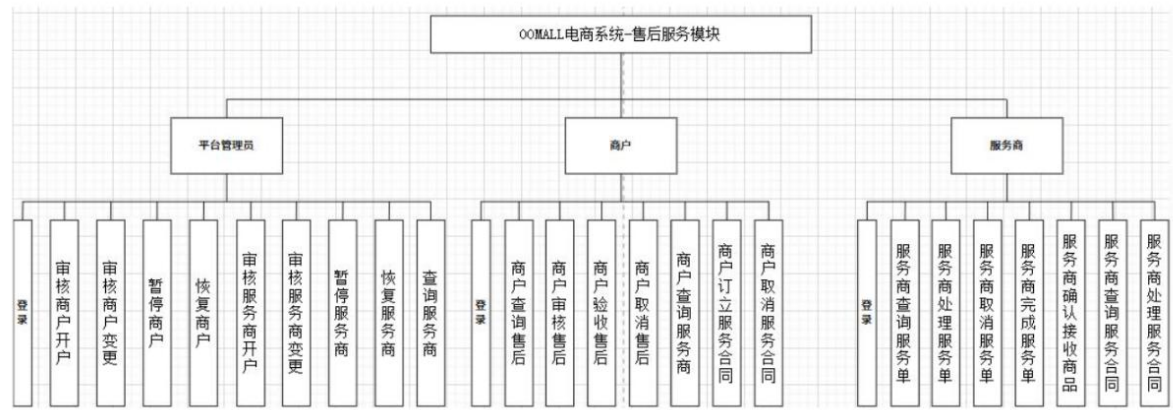
物理结构错误！未定义书签。



技术结构错误！未定义书签。



功能结构错误！未定义书签。



3.2 运行环境及规范

3.2.1 硬件平台

系统运行硬件环境

服务器端配置：采用华为 FlexusL 2 核 2 服务器集群，基于 Docker Swarm 集群部署，包含管理节点和多个工作节点，各节点均运行对应业务容器，支撑商户业务子系统、平台运营子系统、服务商子系统、数据统计子系统等核心模块的稳定运行。

客户端配置：支持 PC 端和移动端设备，无特殊硬件配置要求，满足常规网络访问和业务操作所需的基础硬件性能即可（如普通办公电脑、智能手机、平板电脑等）。

网络配置：依托华为云·云数据中心，配置弹性公网 IP（EIP）、负载均衡器、安全组 / 防火墙、云路由器及内部交换机，构建安全、高效的网络传输通道，保障用户端与服务器端、各子系统间的数据交互。

开发环境硬件平台

客户端硬件配置：无明确特殊要求，采用常规开发用计算机设备，需满足开发工具运行、代码编写与调试、本地测试等基础开发场景的性能需求（如 CPU、内存、存储等配置能支撑开发软件流畅运行）。

服务器配置：可采用与运行环境兼容的测试服务器集群，建议参考运行环境的华为 FlexusL 2 核 2 服务器集群配置标准，或使用同等性能的服务器设备，搭建本地或测试环境的 Docker Swarm 集群，用于开发过程中的功能测试、集成测试等。

网络配置：需搭建本地局域网或连接至测试专用网络，保障开发客户端与测试服务器、数据库等组件间的网络连通性，支持代码部署、数据传输及测试验证等开发流程。

3.2.2 软件平台

系统运行软件环境

操作系统：服务器端适配 Docker Swarm 集群兼容的操作系统（如 Linux 系列操作系统，具体版本需满足 Docker 及集群部署要求）；客户端支持主流操作系统（PC 端：Windows、macOS 等；移动端：iOS、Android 等）。

平台：基于华为云·云数据中心提供的云服务平台，依托 Docker 容器化技术实现应用部署与调度。

中间件：

接入层：API 网关（Spring Cloud Gateway）

缓存中间件：Redis

容器编排：Docker Swarm

数据库：MySQL（主从集群部署），用于存储商户信息、订单数据、售后记录、服务合同等核心业务数据；Redis 作为缓存数据库，提升数据访问效率。

开发工具软件平台

操作系统：支持主流开发操作系统，如 Windows 10/11、macOS、Linux（如 Ubuntu、CentOS 等），需与开发工具、中间件等软件兼容。

开发工具：建议采用 Java 生态相关开发工具（因系统采用 Spring Cloud Gateway 等 Java 技术栈），如 IntelliJ IDEA、Eclipse 等；配套代码管理工具（如 Git）、项目构建工具（如 Maven、Gradle）。

平台：基于 Spring Cloud 微服务开发平台，契合 API 网关（Spring Cloud Gateway）的技术选型，支撑分布式系统开发。

中间件：

接入层测试：本地或测试环境部署 Spring Cloud Gateway，模拟生产环境接入逻辑。

缓存测试：本地或测试环境部署 Redis（版本与生产环境一致），用于开发过程中缓存功能调试。

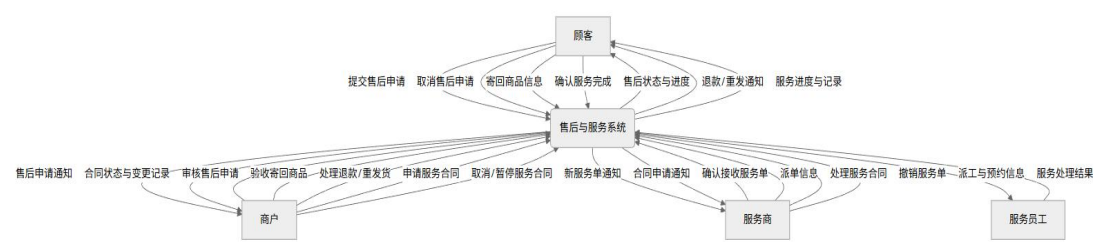
容器化工具：Docker Desktop（客户端开发环境），用于本地容器构建与测试；Docker Swarm（测试服务器集群），用于模拟生产集群部署。

四、概要设计说明书

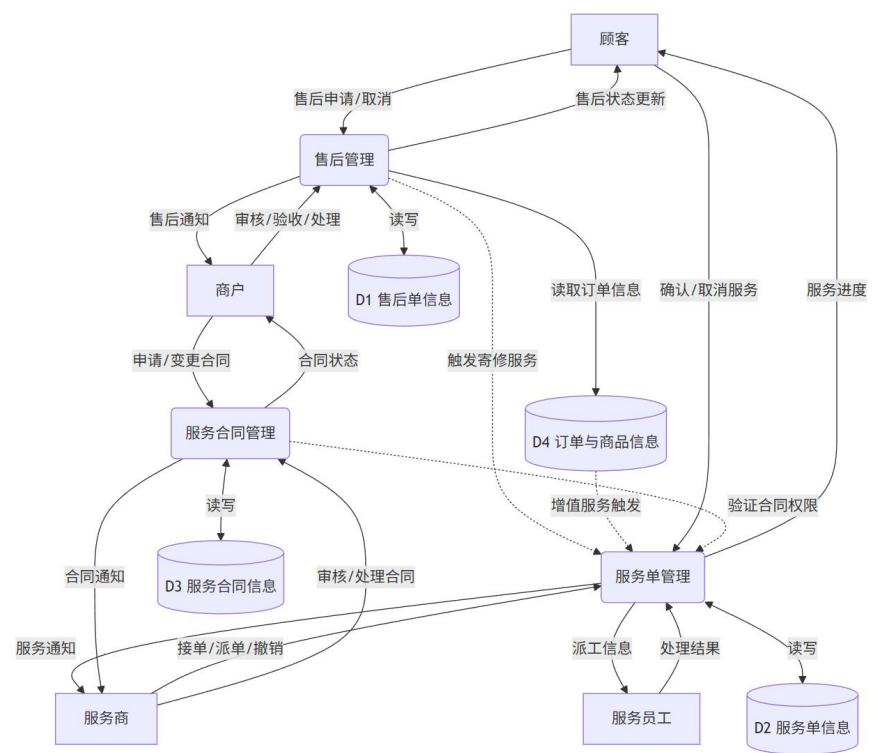
数据库：开发环境部署 MySQL（主从集群架构，版本与生产环境一致），用于本地开发、功能测试的数据存储与验证；Redis（版本与生产环境一致），用于缓存相关功能开发与调试。

3.3 系统的数据流

顶层图:

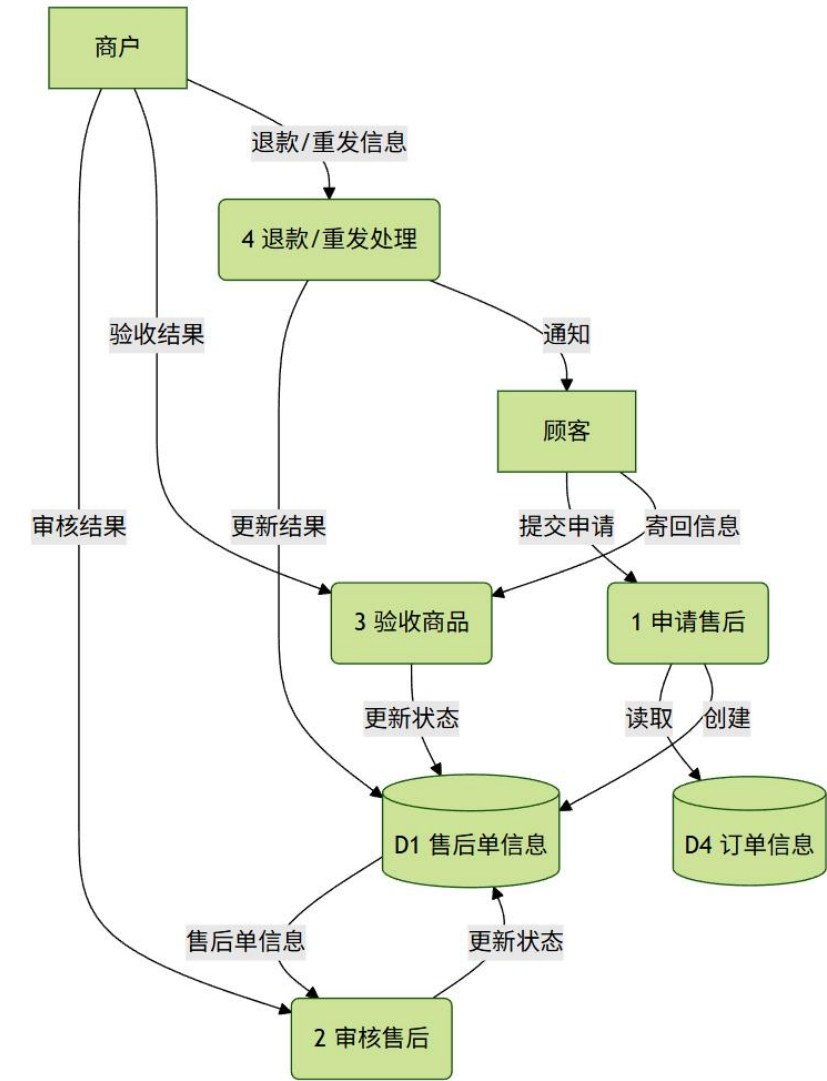


零层图:

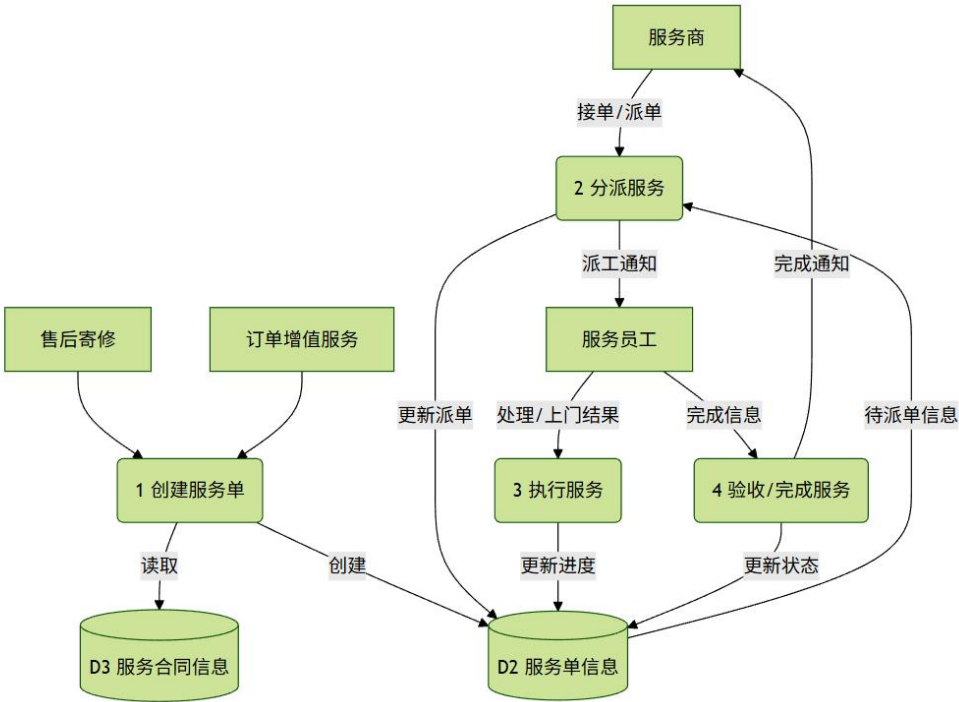


一层图:

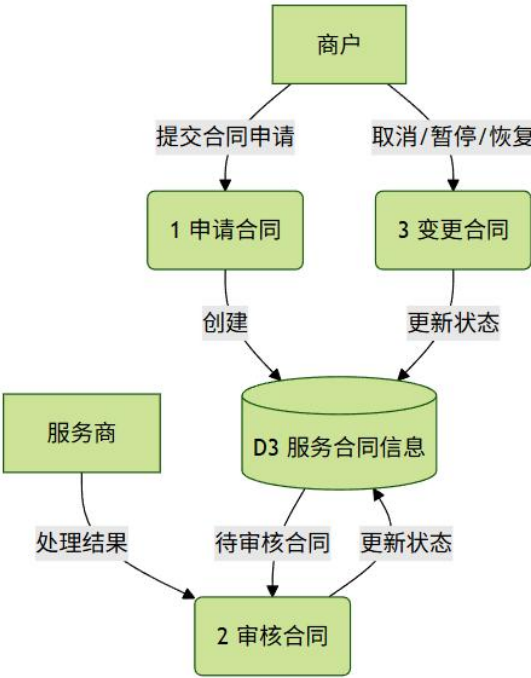
售后管理分解:



服务单管理分解:



服务合同管理分解：



4 . 接口设计

4.1 外部接口

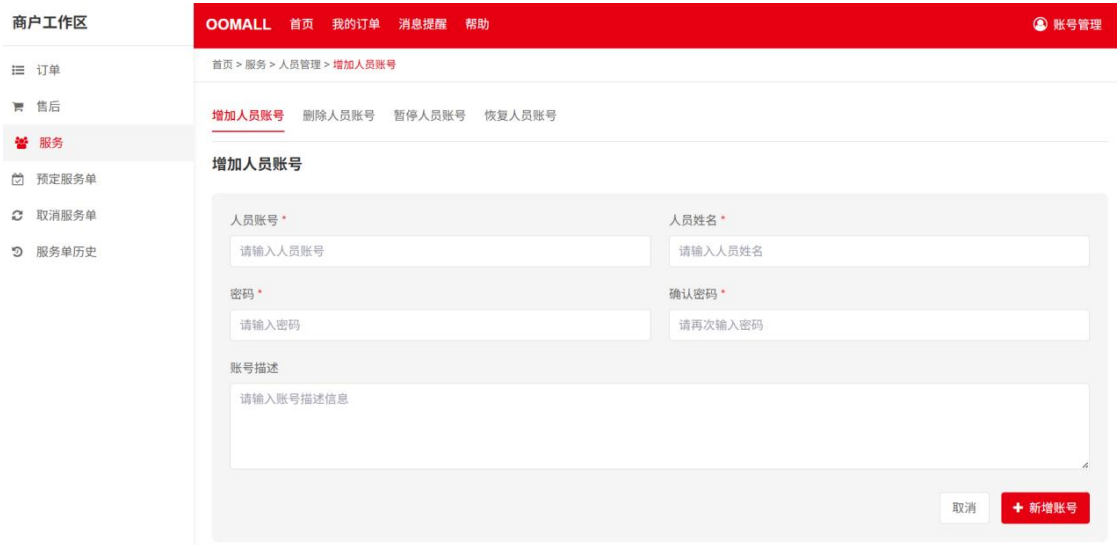
4.1.1 用户界面

本系统前端界面主要分为4个模块，商户界面、服务商界面、平台管理员界面、维修师傅界面。其中维修师傅界面分为Web端界面和手机端界面。

4.1.1.1 商户界面

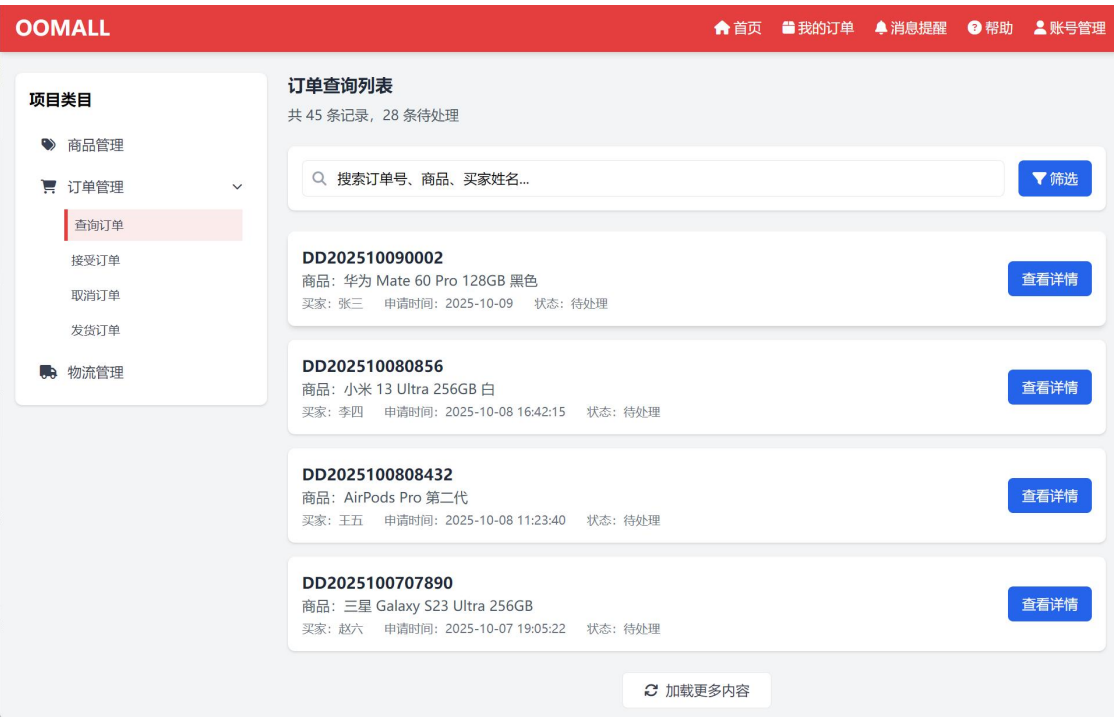


商户-账户管理 界面



商户-人员管理 界面

四、概要设计说明书



商户-订单管理 界面



商户-促销活动 界面

四、概要设计说明书

OOMALL

我的订单

消息提醒

帮助

账号管理

订单

售后

服务

预定服务单

取消服务单

查询服务单

支付单查询

退款单查询

商品管理

查询商品

申请新增商品

申请变更商品

修改商品属性

基本信息

商品名称 *
紫色心情

商品分类 *
个人护理

商品条码 *
114514

商品品牌
请输入商品品牌

原价(元) *
请输入原价

售价(元) *
请输入售价

产地
中国

是否联盟商品
☒ 否 ☐ 是

商品详情

商品描述
请输入商品描述

商品图片

规格参数
请输入规格参数，每行一个参数

库存信息

初始库存数量 *
请输入初始库存数量

库存预警阈值
当库存低于此值时预警

取消

提交申请

商户-商品管理 界面

OOMALL

首页

我的订单

消息提醒

帮助

账号管理

项目类目

售后

查询售后

审核售后

取消售后

验收售后

售后收件

服务

菜单

售后申请列表

共 83 条记录, 12 条待处理

搜索订单号、用户名...

DD20231008001

李小明 申请退货退款 · 2023-10-08 14:20:05

¥428.00

处理申请

DD20231007542

王芳 申请换货 · 2023-10-07 10:12:45

¥670.00

查看详情

DD20231005228

赵建国 申请维修 · 2023-10-05 16:34:21

¥3599.00

查看详情

DD20231004176

周涛 申请补发商品 · 2023-10-04 09:08:52

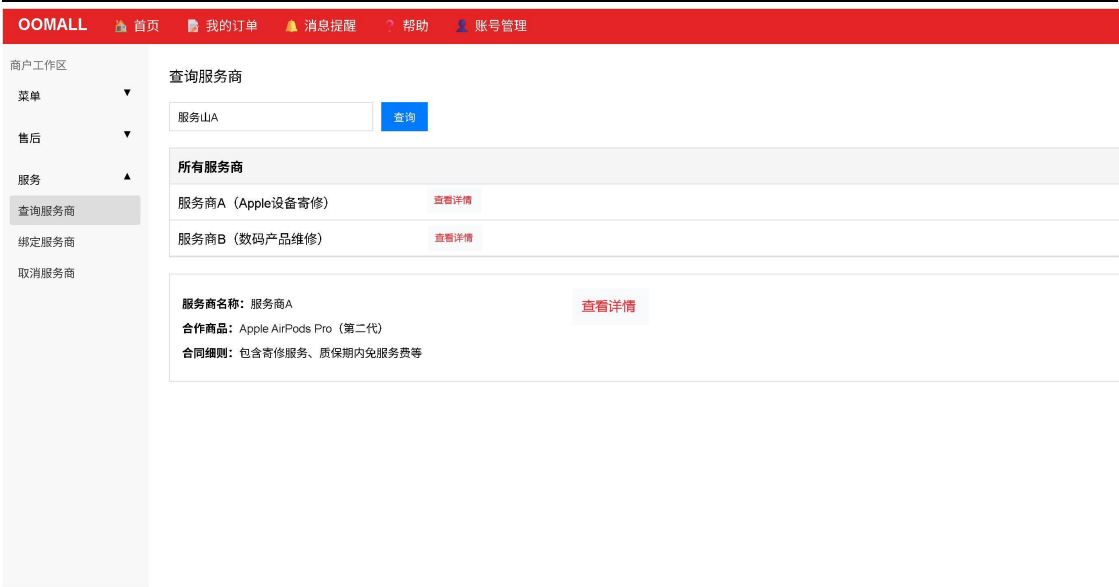
¥158.00

处理申请

加载更多内容...

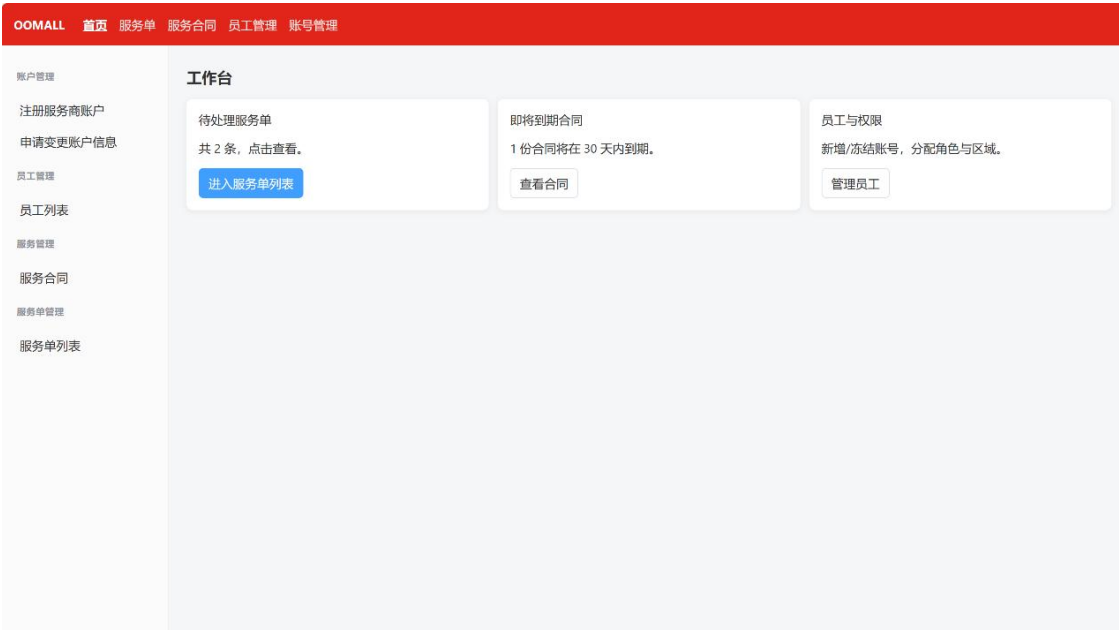
商户-售后 界面

四、概要设计说明书



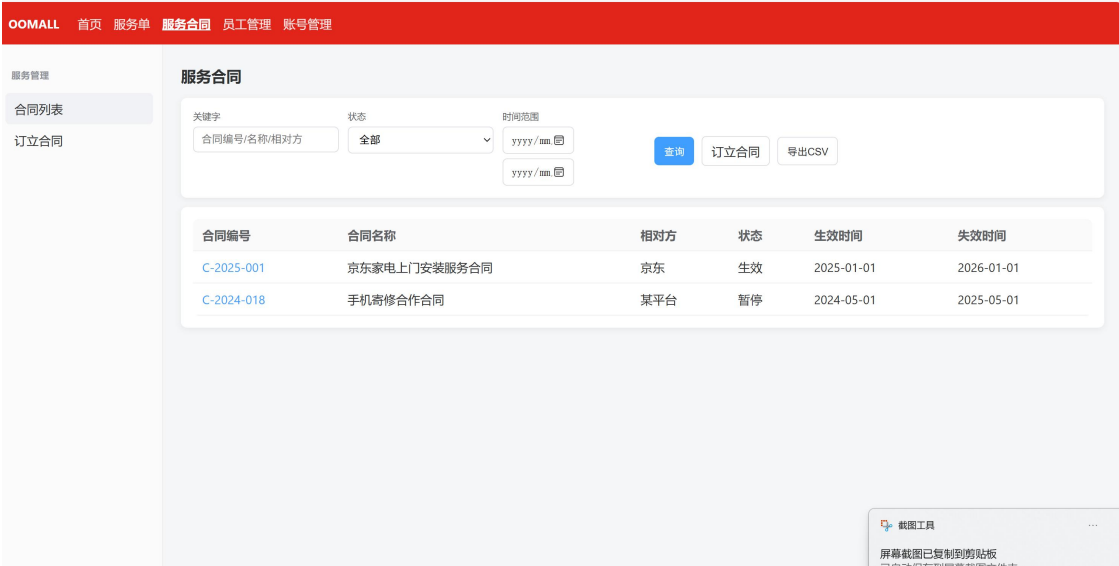
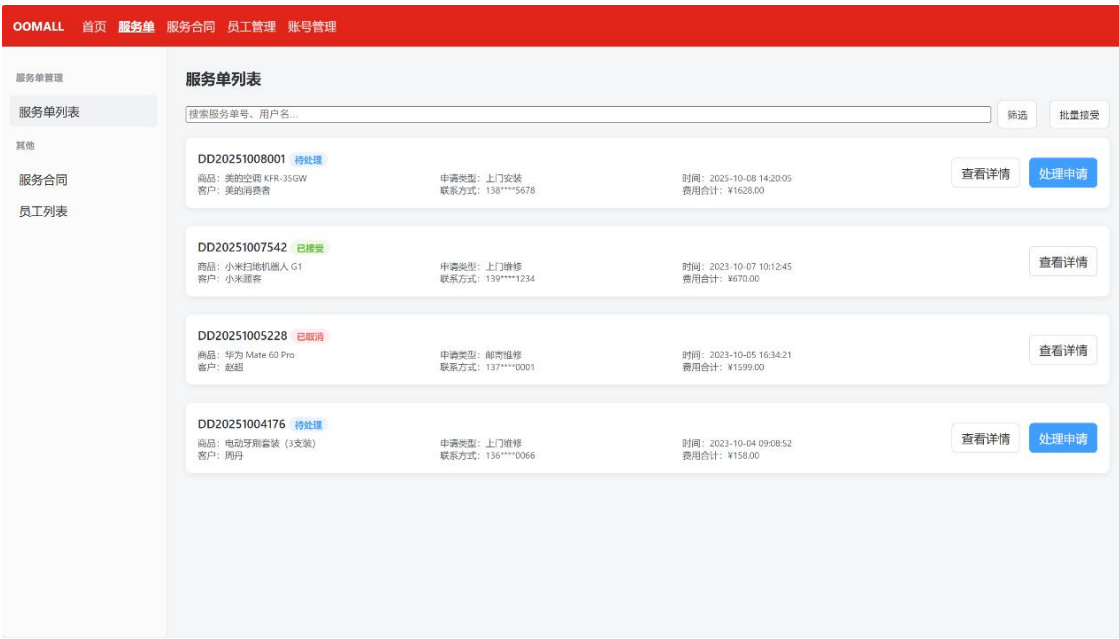
商户-服务商 界面

4.1.1.2 服务商界面

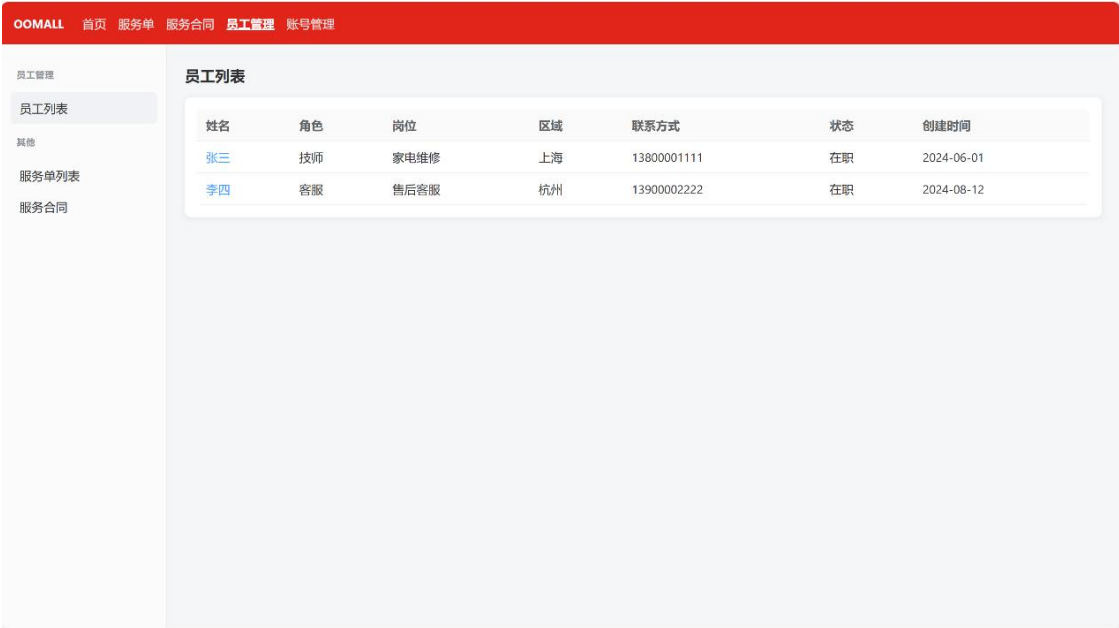


服务商-主页

四、概要设计说明书

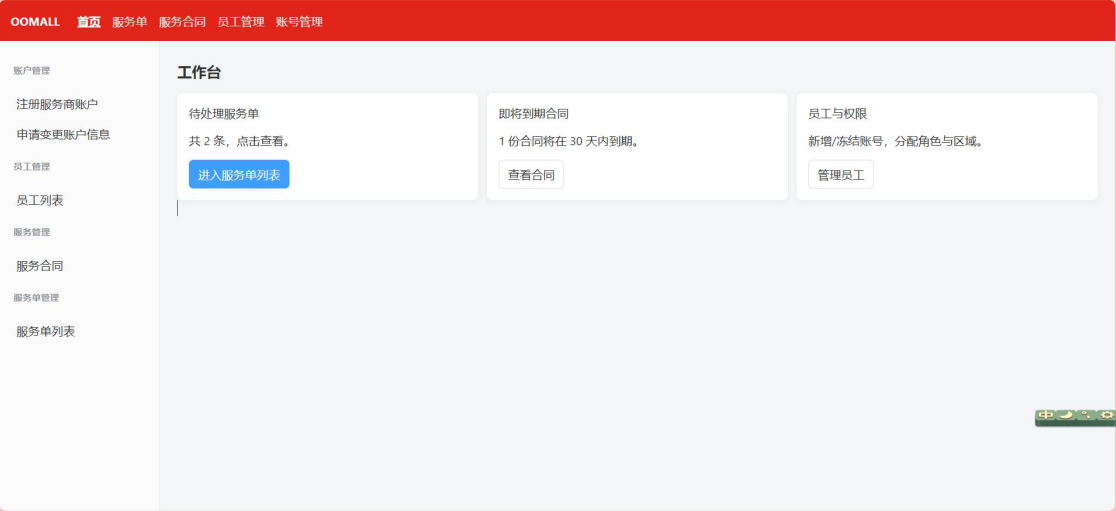


服务商-服务 界面

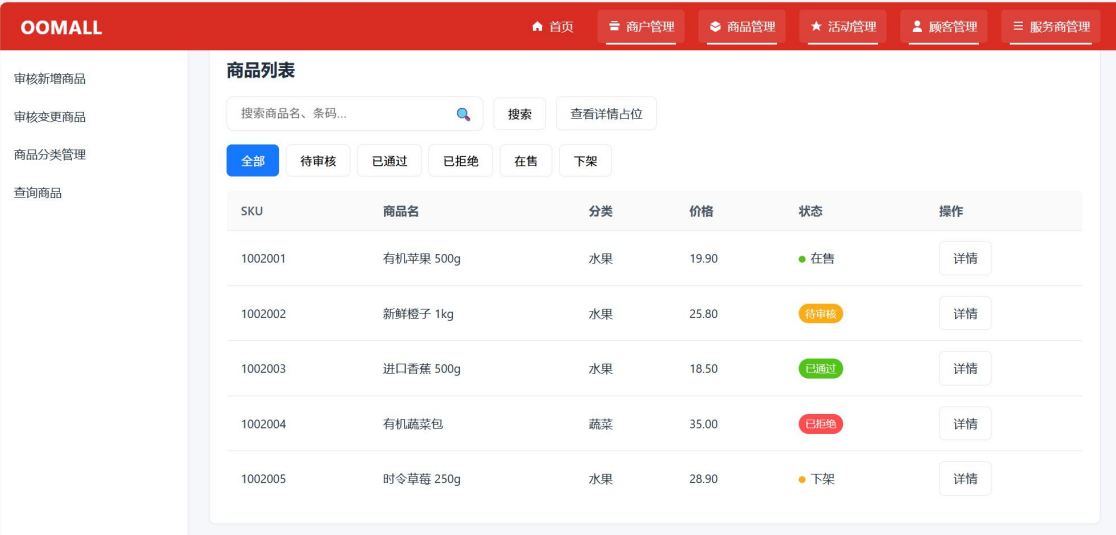


服务商-员工管理 界面

4.1.1.3 平台管理员界面



平台管理员-主页

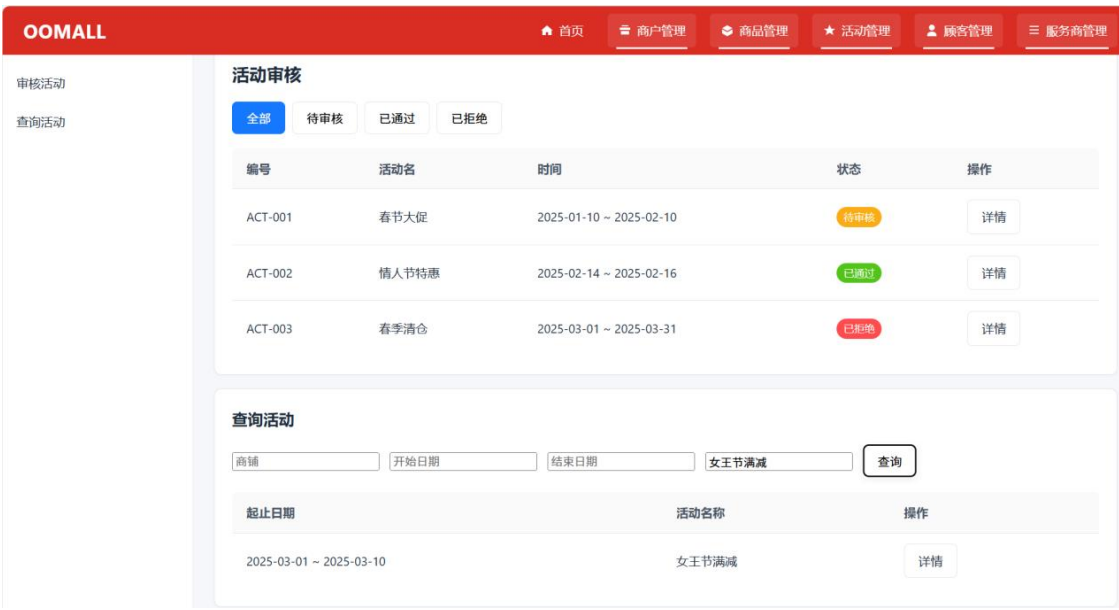


平台管理员-商品管理 界面

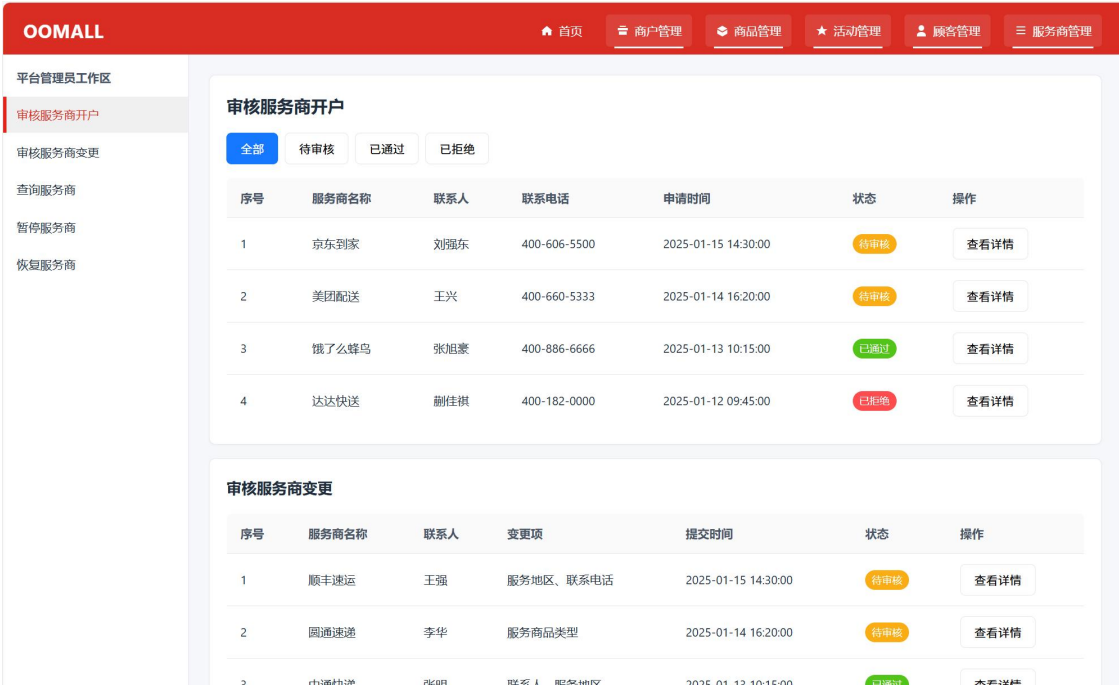


平台管理员-商户管理 界面

四、概要设计说明书



平台管理员-活动管理 界面



平台管理员-服务商管理 界面

4.1.2 软件接口

对于数据库，支持多类数据库，包括 MySQL、MongoDB 和 Redis 等。访问方式通过各数据库的官方驱动实现，用于完成结构化数据、文档数据和缓存数据的读写操作。MySQL 接口基于传统 JDBC 方式进行连接与交互，适用于关系型数据存储；MongoDB 通过官方驱动程序访问，满足文档型数据的高扩展需求；Redis 则由 Redis 客户端实现连接，用于高速缓存与会话管理。所有数据库连接均需具备认证机制，并支持连接池管理，以确保数据访问的性能与稳定性。此外，系统仍可通过 MySQL 的原生命令实现数据备份，以保障关键数据的安全性。

对于访问记录保存，通过 Logback 提供接口支持，负责输出运行日志与系统访问记录。日志需支持分级输出，并可按日期或大小自动切分文件，以保证日志文件可控与易于管理。日志接口应覆盖关键行为记录、错误诊断信息与性能监控信息，以便用于后续的运维分析与审计。

对于模块功能调用，各业务模块之间通过输入的 API 请求触发功能调用，由服务端通过 Java 对 API 内容进行解析并执行业务逻辑。对外提供的所有 API 应符合 RESTful 风格，以保证接口语义清晰、结构统一，并便于各模块间进行独立开发与后期版本扩展。

对于网络软件接口，需支持可靠的数据传输机制，通过滑动窗口方式实现数据在网络中的发送与接收，以确保传输过程中无差错、高吞吐且具备良好的链路利用率。系统还需支持与消息队列服务进行交互，通过消息队列接口完成任务异步处理与模块解耦，以提高系统整体的并发能力与可扩展性。

4.1.3 硬件接口

本系统依托服务器集群进行部署与运行，各组件需与相应硬件资源建立连接并协同工作，以确保系统的稳定性与可扩展性。

系统运行于华为云服务器环境，支持移动端与 PC 端的访问。后端服务部署在服务器集群中，各节点需满足以下基础硬件要求：

CPU：2 核及以上

内存：2 GB 及以上

硬盘：40 GB 及以上

系统需与数据库服务器建立备份接口，用于保障业务数据的安全性。备份机制包括：

备份频率：每日自动备份

备份存储：独立备份服务器

备份保留期限：7 天

系统采用 Nginx 作为 Web 服务器，负责处理外部访问请求并进行静态资源分发。同时，集群前端配置 HAProxy 负载均衡器，实现业务流量的分配与服务节点间的负载均衡，确保整体服务能力与容错性能。

4.2 内部接口

内部接口方面，各模块之间通过 api 调用接口、参数传递、返回值的方式进行传递。具体参数与返回值的接口将在数据结构设计的内容中说明。接口传递的信息是以数据接口封装了的数据，以参数传递或返回值的方式在各模块间传递。

一个内部接口示例如下：



根据售后单id查询售后单信息

5 . 数据库结构设计

5.1 逻辑结构设计

5.1.1 服务模块

Oomall_service_provider 服务商信息表

列名	数据类型	能否为空	说明
id	INTEGER	NOT NULL	服务商 ID（主键）
regionid	INTEGER	NULL	关联区域 ID（外键： Region.RegionID）
name	VARCHAR	NULL	服务商名称
consignee	VARCHAR	NULL	服务商收件人（联系人姓名）
phone	VARCHAR	NULL	服务商联系方式
address	VARCHAR	NULL	服务商地址
status	INTEGER	NULL	状态（0: 有效 ,1: 暂停 ,2: 已删除 ）

Oomall_service 服务信息表

列名	数据类型	能否为空	说明
id	INTEGER	NOT NULL	服务 ID（主键）
service_provider_id	INTEGER	NULL	服务商 ID（外键： ServiceProvider.id）
name	VARCHAR	NULL	服务名称
description	VARCHAR	NULL	服务描述
type	INTEGER	NULL	服务类型
status	INTEGER	NULL	状态（0: 有效 ,1: 暂停 ,2: 已删除 ）

四、概要设计说明书

列名	数据类型	能否为空	说明
create_time	DATETIME	NULL	开始时间
update_time	DATETIME	NULL	更新时间

Oomall_service_order 服务单信息表

列名	数据类型	能否为空	说明
id	INTEGER	NOT NULL	服务单 ID（主键）
serviceordersn	VARCHAR	NULL	服务单序列号
productid	INTEGER	NULL	商品 ID（外键：Product.ProductID）
product_service_id	INTEGER	NULL	商品服务关联 ID（外键：ProductService.id）
regionid	INTEGER	NULL	区域 ID
shopid	INTEGER	NULL	商户 ID
customerid	INTEGER	NULL	顾客 ID（外键：Customer.CustomerID）
service_provider_id	INTEGER	NULL	服务商 ID（外键：ServiceProvider.id）
aftersalesid	INTEGER	NULL	售后 ID（外键：AfterSale.id）
send_expressId	INTEGER	NULL	寄出物流 ID
send_express_billcode	VARCHAR	NULL	运费单号
type	INTEGER	NULL	类型
status	INTEGER	NULL	状态
result	VARCHAR	NULL	结果
description	VARCHAR	NULL	服务描述
ordertime	DATETIME	NULL	订单时间
customername	VARCHAR	NULL	顾客姓名（订单快照信息）
customeraddress	VARCHAR	NULL	顾客地址（订单快照信息）

四、概要设计说明书

列名	数据类型	能否为空	说明
customerphone	VARCHAR	NULL	顾客电话（订单快照信息）
service_provider_name	VARCHAR	NULL	服务商名称（订单快照信息）
service_provider_address	VARCHAR	NULL	服务商地址（订单快照信息）
service_provider_phone	VARCHAR	NULL	服务商电话（订单快照信息）

Oomall_product_service 服务-商品联系表

列名	数据类型	能否为空	说明
id	INTEGER	NOT NULL	关联 ID（主键）
productid	INTEGER	NULL	商品 ID（外键：Product.ProductID）
serviceid	INTEGER	NULL	服务 ID（外键：Service.id）
service_provider_id	INTEGER	NULL	服务商 ID（外键：ServiceProvider.id）
regionid	INTEGER	NULL	区域 ID
shopid	INTEGER	NULL	商户 ID
price	DECIMAL	NULL	价格
status	INTEGER	NULL	状态（0: 正常，1: 暂停）
servicetype	INTEGER	NULL	服务类型

5.1.2售后模块

Oomall_aftersale 售后单信息表

列名	数据类型	能否为空	说明
id	INTEGER	NOT NULL	售后单 ID（主键）

四、概要设计说明书

列名	数据类型	能否为空	说明
aftersale_sn	VARCHAR	NULL	售后单序列号（新增）
shop_id	INTEGER	NULL	商铺 ID（外键）
customer_id	INTEGER	NULL	顾客 ID（外键：Customer.CustomerID）
product_id	INTEGER	NULL	商品 ID（外键：Product.ProductID）
order_item_id	INTEGER	NULL	订单项 ID（新增，外键）
service_id	INTEGER	NULL	服务单 ID（外键）
serial_no	VARCHAR	NULL	服务对象序列号（商品串号）
name	VARCHAR	NULL	产品名称
type	INTEGER	NULL	售后类型（0: 换货 , 1: 退货 , 2: 维修 , 3: 安装）
reason	VARCHAR	NULL	原因
conclusion	VARCHAR	NULL	处理意见
quantity	INTEGER	NULL	数量
region_id	INTEGER	NULL	取货地区 ID
address	VARCHAR	NULL	取货地址
contact	VARCHAR	NULL	联系人
mobile	VARCHAR	NULL	联系电话
status	INTEGER	NULL	售后状态（0: 未完成 , 1: 进行中 , 2: 已完成 , 3: 已取消）

5.2 物理结构设计

（1）持久化对象（PO）构建与映射 系统遵循数据与行为分离原则，在持久化层针对每一个数据库表建立一一对应的 PO (Persistent Object) 对象。PO 严格作为数据库物理结构的镜像（Raw Data Carrier），负责承载从数据库读取的原始记录，仅用于处理对象关系映射（ORM），不包含任何业务逻辑。

（2）业务对象（BO）转换与领域隔离 为实现核心业务逻辑与底层存储实现的解耦，业务层必须基于 BO (Business Object) 对象进行运作。当数据流向业务层时，需通过转换器将 PO 转化为 BO。BO 是对业务实体的抽象，不仅包含业务所需的属性，还封装了相

关行为。业务逻辑仅依赖并调用 BO，严禁直接操作 PO，从而确保业务领域模型的纯粹性与稳定性。

(3) 数据持久化写入机制 数据写入过程遵循反向转换原则。当业务操作完成后，系统需将承载最终业务状态的 BO 对象逆向转换为符合物理表结构的 PO 对象，最后由持久化层执行数据库写操作。此机制屏蔽了数据库 Schema 对上层业务代码的侵入，保证了系统的可维护性。

5.3 数据结构与程序的关系

系统的数据结构由标准数据库语言 SQL 生成
例如:

1. 服务商信息表 (Oomall_service_provider)

```
CREATE TABLE Oomall_service_provider (  
  
id bigint(20) NOT NULL AUTO_INCREMENT,  
  
regionid bigint (20) DEFAULT NULL COMMENT ' 关联区域 ID (外键: Region.RegionID)  
,  
  
name varchar (255) DEFAULT NULL COMMENT ' 服务商名称 ',  
  
consignee varchar (128) DEFAULT NULL COMMENT ' 服务商收件人 (联系人姓名)  
  
phone varchar (64) DEFAULT NULL COMMENT ' 服务商联系方式 address varchar (500)  
DEFAULT NULL COMMENT ' 服务商地址 ',  
  
status int (11) DEFAULT NULL COMMENT ' 状态 (0: 有效, 1: 暂停, 2: 已删除) ',  
  
PRIMARY KEY (id)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT=' 服务商信息表 ';
```

在系统中，PO 对象与数据库表转换所需的代码由MyBatis 自动生成
例如:

oomall_service_provider 所对应的GoodsPo 对象代码如下:

```
public class ServiceProviderPo {  
    private Integer id;  
    private Integer regionid;  
    private String name;  
    private String consignee;  
    private String phone;  
    private String address;  
    private Integer status;  
    public Integer getId() {  
        return id;  
    }  
    public void setId(Integer id) {  
        this.id = id;  
    }  
    public Integer getRegionid() {  
        return regionid;  
    }  
    public void setRegionid(Integer regionid) {  
        this.regionid = regionid;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name == null ? null : name.trim();  
    }  
    public String getConsignee() {  
        return consignee;  
    }  
    public void setConsignee(String consignee) {  
        this.consignee = consignee == null ? null : consignee.trim();  
    }  
    public String getPhone() {  
        return phone;  
    }  
}
```

```
    }  
  
    public void setPhone(String phone) {  
        this.phone = phone == null ? null : phone.trim();  
    }  
  
    public String getAddress() {  
        return address;  
    }  
  
    public void setAddress(String address) {  
        this.address = address == null ? null : address.trim();  
    }  
  
    public Integer getStatus() {  
        return status;  
    }  
  
    public void setStatus(Integer status) {  
        this.status = status;  
    }  
}
```

6. 运行设计

6.1 运行模块的组合

OOMALL 系统由若干协作模块构成完整的数据处理流程。系统在接收到外部输入后，由前端视图层完成数据接收与格式化，并通过网络传输模块将数据发送至服务器端。服务器在保持网络监听的状态下接收请求，并将输入交由数据处理模块进行解析与处理。

服务器端的数据处理模块负责访问并操作 MySQL、MongoDB 等数据库，完成数据查询、更新或计算后将结果返回前端。客户端在接收到处理结果后通过视图层进行展示，从而形成从数据输入、网络传输、数据处理到结果输出的完整闭环流程。系统模块间协同工作，实现了高效、可扩展、易维护的数据处理机制。

6.2 运行控制

系统的运行控制依据各模块间既定的调用关系进行管理，通过在关键模块中设置必要的判断逻辑，以选择正确的执行路径。事务中心模块根据不同业务逻辑与运行条件确定相应的处理流程，确保事务处理过程的正确性、完整性和一致性。

在网络通信过程中，系统采用双向确认机制保证数据可靠传输。客户端发送数据后等待服务器确认，再接收服务器处理后的响应并进行确认；服务器在接收数据后发送确认信号，完成处理后再次返回结果并等待客户端确认。这一运行控制方式确保了网络传输的同步性和数据交互的可靠性。

6.3 运行时间

OOMALL 系统使用 Redis 作为高频访问的数据存储方案，以其亚毫秒级响应能力提升整体性能。系统在设计中将用户响应时间控制在 1–2 秒范围，并针对高并发场景预留性能余量，即使在极端访问压力下，响应时间仍控制在 2.5 秒以内，以确保稳定的用户体验。

7. 出错处理设计

7.1 出错输出信息

OOMALL 系统具备详细的错误处理机制，确保在出错时能够提供清晰的错误码、错误描述以及相应的提示。出错信息分为处理错误、设定错误和系统错误三种情况：

1.输入错误

针对输入信息超出预设范围或不符合指定格式的异常，系统将在操作结果判定与输入数据验证模块中完成错误类型分析，生成精准的错误提示语句，经输出模块同步至客户端，确保用户可明确知晓输入异常的具体原因。

2.配置错误

针对权限不足等系统预设的不可执行场景，系统在信息提交初期即依据权限等预设规则判定错误类别，生成对应错误信息并推送至输出模块，实现系统层面对配置类异常的精准识别与用户侧的及时反馈。

3.系统错误

针对网络传输超时、服务器响应超时等底层系统故障，系统将依据服务器响应数据判定错误类别并输出对应提示，保障系统级异常可被快速识别，同时向用户提供明确的故障描述。

此外，系统对错误信息设置统一规范：一是所有错误提示均需附带具体成因（如“该地区不可达”），提升用户对异常的认知程度；二是采用自定义返回码，规避与 HTTP 原生返回码的冲突，保障错误码的唯一性。整体机制设计兼顾用户体验与系统可维护性，符合行业通用的异常处理标准。

7.1.1 服务器错误

错误码	对应代码	对应描述
500	INTERNAL_SERVER_ERR	服务器内部错误
501	NOT_IMPLEMENTED_ERR	服务器不具备完成请求的功能
502	BAD_GATEWAY_ERR	网关错误
503	SERVICE_UNAVAILABLE_ERR	服务器目前无法请求
504	GATEWAY_TIMEOUT_ERR	错误的网关超时
505	HTTP_VERSION_NOT_SUPPORTED_ERR	服务器不支持请求中使用的HTTP协议版本

7.1.2 数据库错误

错误码	对应代码	对应描述
130	FILE_FORMAT_INVALID_ERR	文件格式不正确
145	FILE_CANNOT_OPEN_ERR	文件无法打开
1005	TABLE_CREATE_FAILED_ERR	创建表失败
1006	DATABASE_CREATE_FAILED_ERR	创建数据库失败
1007	DATABASE_CREATE_FAILED_DUPLICATE_ERR	数据库已存在，创建数据库失败
1008	DATABASE_DELETE_FAILED_NOT_EXIST_ERR	数据库不存在，删除数据库失败
1009	DATABASE_DELETE_FAILED_FILE_ERR	不能删除数据库文件导致删除数据库失败
1020	RECORD_MODIFIED_BY_OTHERS_ERR	该记录已被其他用户更改
1022	RECORD_UPDATE_FAILED_DUPLICATE_KEY_ERR	关键字重复，更改记录失败
1036	TABLE_READ_ONLY_CANNOT_MODIFY_ERR	数据表只读，不可修改

1037	SYSTEM_INSUFFICIENT_MEMORY_ERR	系统内存不足
1038	INSUFFICIENT_MEMORY_FOR_SORT_ERR	用于排序的内存不足
1048	FIELD_CANNOT_BE_NULL_ERR	字段不能为空

7.1.3 业务错误

错误码	对应代码	对应描述
901	GOODS_CATEGORY_SAME	类目名称已存在
902	GOODS_PRICE_CONFLICT	商品销售时间冲突
903	GOODS_CATEGORY_NOTALLOW	不允许加入到一级分类
909	COUPON_NOTBEGIN	未到优惠券领取时间
910	COUPON_FINISH	优惠券领罄
911	COUPON_END	优惠券活动终止
921	CATEALTER_INVALID	对SPU 类别操作无效
924	ACTIVITY_NOTFOUND	无符合条件的优惠活动
925	SHOP_NOTOPERABLE	不可对该商铺进行操作
941	COMMENT_EXISTED	该订单条目已评论
947	LATE_BEGINTIME	开始时间不能晚于结束时间
948	ACT_LATE_PAYTIME	尾款支付时间晚于活动结束时间
949	ACT_EARLY_PAYTIME	尾款支付时间早于活动开始时间
950	COUPON_LATE_COUPONTIME	优惠券领卷时间晚于活动开始时间
966	SHOP_HASDEPOSIT	店铺仍有保证金未结算
967	SHOP_CATEGORY_NOTPERMIT	不允许增加新的下级分类
968	SHOP_NOT_RECON	店铺尚有支付未清算完毕
969	SHOP_USER_HASSHOP	用户已经有店铺
970	COMMENT_USER_NOORDER	用户没有购买此商品
995	FREIGHT_REGIONOBSOLETE	地区已废弃
996	FREIGHT_WRONGTYPE	该运费模板类型与内容不符
997	FREIGHT_REGIONEXIST	该运费模板中该地区已经定义
998	FREIGHT_NOTDELETED	存在上架销售商品，不能删除运费模板
999	FREIGHT_REGIONSAME	运费模板中该地区已经定义

7.2 出错处理对策

1. 数据完整性保障。为保障数据完整性，需维持服务端持续开机，规避停电或电压不稳引发的数据丢失；若服务器意外断电，可依托 SQL Server 日志文档，通过 ROLLBACK 回滚机制恢复数据，将系统还原至稳定状态。
2. 错误信息管控机制。系统返回错误码时，会同步向客户端推送出错信息并在服务端控制台输出；同时自动将错误码、出错详情及出错时间归档至日志文档，为后续运维与问题排查提供依据，提升系统可维护性。

3. 在网络方面，搭建低成本后备网络，在主网络中断时保障网关与外部网络的通信畅通，维持系统网络稳定性。
4. 在硬件方面，配备充足数量的服务器，既满足系统负载需求、降低单点故障风险，也可各模块提供相对独立的运行环境；同时增设备用服务器，确保主服务器故障时系统可无缝切换、持续运行。

8.安全保密设计

1.物理层安全

物理层是安全防护的基础，主要关注对硬件设备、数据中心和网络基础设施的保护。物理安全的目的是防止未经授权的人员访问或破坏硬件设备，从而避免数据泄露、丢失或恶意操作。

1.1 物理访问控制

数据中心物理安全：数据中心应部署先进的安保系统（如监控摄像头、门禁系统、安保人员等），防止未经授权人员接触到设备或数据。

设备锁定：确保服务器和其他重要硬件设备在物理上被锁定在安全区域内，避免外部人员直接接触。

环境控制：数据中心应配备温湿度控制、防火防灾系统、电力管理系统，确保设备的正常运行。

1.2 硬盘加密与销毁

硬盘加密：所有存储敏感数据的硬盘应进行加密，即使硬盘被盗或遗失，数据也无法被访问。

硬盘销毁：对废弃的硬盘或存储设备进行物理销毁，避免敏感数据通过二手设备泄露。

1.3 冗余和备份

电力冗余：使用 UPS（不间断电源）和发电机等设备确保数据中心不受电力中断影响。

数据冗余：配置硬盘阵列 (RAID)，确保数据在硬件故障时不丢失。

2.网络层安全

网络层安全主要针对网络基础设施，防止网络攻击，确保网络通信的完整性和保密性。

2.1 防火墙

部署网络防火墙来阻止恶意的网络流量。防火墙可以过滤进入或离开网络的数据包，基于规则控制数据流动。

内外网隔离：确保内网和外网之间的隔离，防止外部攻击者直接访问内网。

2.2 入侵检测与防御系统

入侵检测系统：检测网络中的异常活动，并发出警告，帮助安全人员发现潜在威胁。

入侵防御系统：在入侵检测的基础上进一步采取措施（如阻断攻击源），主动防止攻击。

2.3 虚拟专用网络 (VPN)

使用 VPN 在公网中建立加密的虚拟通道，确保远程办公人员或分支机构的网络通信安全。

2.4 网络分段

将网络划分为多个子网，并通过路由器或防火墙控制不同子网之间的访问权限。通过内网划分和访问控制策略，减少攻击的传播路径。

2.5 加密协议

使用 HTTPS来加密传输中的数据，防止数据在传输过程中被中间人攻击。

使用 IPsec 等 VPN 协议来确保数据传输的保密性和完整性。

3.系统层安全

系统层安全指的是操作系统层面和硬件平台的保护措施，防止恶意软件、未经授权访问和系统漏洞带来的风险。

3.1 操作系统安全硬化

最小化安装：安装操作系统时，尽量选择最小化的安装选项，移除不必要的服务和软件，减少攻击面。

定期更新与补丁管理：定期对操作系统进行安全补丁更新，修补已知的漏洞。

用户权限管理：确保用户账户的权限设置遵循最小权限原则，不让非授权人员或服务获得过高权限。

3.2 防病毒与反恶意软件

安装并定期更新防病毒软件，监控并阻止恶意软件的入侵。

启用实时监控，自动扫描文件系统和网络流量，及时发现潜在威胁。

3.3 日志管理与审计

系统日志: 启用详细的系统日志记录, 监控所有用户活动和系统行为, 确保发生异常事件时能追溯源头。

审计工具: 配置审计工具, 定期审核系统日志, 查找不正常的系统操作。

4.应用层安全

应用层安全主要针对软件应用本身, 防止代码漏洞 (如SQL注入、XSS等) 被恶意利用, 确保应用程序的安全性。

4.1 输入验证与输出编码

输入验证: 所有从用户输入的内容 (如表单、URL参数) 都必须进行严格的验证, 防止恶意代码注入。

输出编码: 对用户输入的内容进行适当的输出编码, 防止XSS攻击 (如 `<script>` 标签注入)。

4.2 使用安全开发框架

采用经过安全审计的开发框架和库, 避免使用已知存在安全漏洞的开源组件。

在开发过程中, 遵循安全编码规范, 避免不安全的编程实践 (如直接拼接 SQL 语句)。

4.3 身份验证与会话管理

使用多因素认证 (MFA), 防止账户被盗用。

会话管理中, 应保证会话ID的安全性, 避免会话劫持。

4.4 代码审计与漏洞扫描

定期对应用程序进行源代码审计, 查找潜在的安全漏洞。

使用自动化工具扫描应用程序中的安全漏洞, 如 OWASP ZAP、Burp Suite 等。

4.5 跨站请求伪造防护

通过令牌验证机制来防止跨站请求伪造攻击, 确保请求来源的合法性。

5.接口层安全

接口层安全主要涉及 API 接口的安全性, 防止恶意攻击者利用漏洞进行非法操作或窃取数据。

5.1 认证与授权

使用 OAuth 2.0 或 JWT 等安全协议来管理 API 的认证与授权, 确保只有合法用户和客户端能够访问接口。

API 密钥: 为每个用户或应用生成唯一的 API 密钥, 并且应确保密钥的安全存储和传输。

5.2 加密与签名

对 API 的请求和响应数据进行加密, 防止数据在传输过程中被篡改或窃取。

通过数字签名确保请求数据的完整性和来源的合法性。

5.3 速率限制与防滥用

设置速率限制防止 API 被滥用, 避免 DDoS 攻击。

对每个用户或 IP 设置访问频率限制, 防止接口被暴力破解或滥用。

5.4 API 网关

部署 API 网关, 集中管理所有 API 的访问控制、安全策略、日志记录等。API 网关可以进行认证、负载均衡、流量控制等。

6.访问控制

访问控制的目的是限制系统资源的访问权限, 确保只有授权用户能够访问敏感资源。

6.1 身份认证

使用强认证机制 (如多因素认证、SSO) 验证用户身份。

对每个用户实行唯一的标识符 (如用户名、电子邮件地址等), 并进行加密存储。

6.2 访问控制模型

基于角色的访问控制 (RBAC): 根据用户的角色来控制其访问权限, 确保不同角色有不同的资源访问权限。

基于属性的访问控制 (ABAC): 根据用户、资源、环境等属性来动态决定访问权限, 灵活控制复杂场景下的访问。

基于规则的访问控制 (RBAC): 通过事先设定的规则控制访问权限, 如时间、地理位置等限制。

6.3 审计与监控

对所有的访问操作进行记录和监控, 及时发现异常访问行为, 保障系统的安全性。

6.4 权限最小化与定期审计

遵循最小权限原则, 确保用户仅有其完成工作所需的最低权限。

定期审查用户权限, 撤销不再需要的权限, 避免权限滥用。

9.维护设计

维护设计的关键目标:

降低维护难度: 通过设计和预见维护需求, 确保系统能够方便地进行故障排查、修复、更新等。

提高系统稳定性和可靠性: 设计时预留适当的空间和冗余, 以避免单点故障或运行中断。

延长系统生命周期: 通过合理的设计和支持, 减少系统的衰退和性能瓶颈。

1、维护模块化设计

目的: 将系统分为多个可独立维护的模块, 减少对其他模块的影响。

实施方法: 通过功能划分、代码组织、服务解耦等手段, 将系统的各个部分拆分成独立模块。

这样可以在某一模块出现故障时, 单独进行维护和更新, 不影响整体系统的稳定性。

2、便捷的维护接口

目的: 为系统管理员和开发人员提供易于使用的接口, 便于进行系统的调试和维护操作。

实施方法:

监控接口: 提供系统状态、健康检查、日志查看等接口, 便于实时跟踪系统运行状况。

调试接口: 允许开发者远程调试系统或执行一些临时的修复操作。

API文档: 通过清晰的API文档帮助开发人员了解如何进行系统操作, 减少维护时的理解成本。

3、日志与监控系统

目的: 确保系统运行时能够实时监控, 并提供足够的日志记录, 以便在出现故障时进行排查和分析。

实施方法:

日志管理: 记录系统操作、异常、错误等信息。日志格式和内容应统一且结构化, 便于后期分析。

实时监控: 采用自动化工具 (如 Prometheus、Grafana 等) 监控系统的性能、资源使用和故障。

告警机制: 设置合适的告警阈值, 当系统出现故障或性能瓶颈时, 及时通知相关人员处理。

维护设计的核心要素:

1、自动化工具支持

目的：利用自动化工具减少人为操作，提高维护效率。

实施方法：

自动化测试：定期运行自动化测试，确保系统各个模块的功能持续正常。

自动化部署：使用CI/CD（持续集成/持续交付）工具自动化部署，保证每次更新后能快速验证

并修复潜在问题。

故障恢复自动化：建立容灾和故障恢复机制，如自动重启服务、自动切换到备用系统等，确保

系统出现故障时能够迅速恢复。

2、清晰的文档与知识库

目的：确保系统的维护者能方便地获取所需的信息，包括操作步骤、故障处理方案等。

实施方法：

维护文档：系统设计、配置、运行的文档必须齐全，且易于维护人员查阅。

常见问题解决方案：记录常见故障的排查方法、解决步骤和预防措施。

知识库：建立一个团队共享的知识库，记录技术细节、操作规范和常见的维护经验。

3、用户与权限管理设计

目的：通过清晰的权限管理，避免不当操作对系统的破坏，并提高维护效率。

实施方法：

权限划分：根据角色分配不同的系统访问权限，避免错误的操作导致系统崩溃或数据丢失。

审计日志：记录所有关键操作，便于追溯并定位问题。

4、冗余与备份设计

目的：确保在硬件或软件出现故障时，能够有冗余系统或数据备份进行恢复。

实施方法：

系统冗余：关键设备、服务及数据库应配置冗余方案，避免单点故障导致系统停运。

数据备份：定期备份重要数据，并确保备份数据可以迅速恢复。

5、可扩展性与灵活性设计

目的：确保系统能够适应未来的需求变化，并便于扩展和更新。

实施方法：

模块扩展：通过接口、插件等方式，便于后期增加新的功能模块。

灵活配置：设计灵活的配置文件和参数管理系统，便于快速调整系统的运行策略。

定期进行系统审查：定期对系统进行检查，发现潜在的维护问题，提前进行优化。

建立标准化流程：制定标准的维护流程和操作规范，确保所有维护操作遵循统一的规则。

用户反馈与持续改进：通过收集系统使用者的反馈，不断改进和优化系统设计，提高维护体

验。

培训和知识共享：定期对运维人员进行培训，确保他们掌握最新的技术和系统维护方法。