电商交易平台设计与实现

黄梓桐
2023211045

2025 年 6 月 20 日

目录

1 前言

随着移动互联网的快速发展，电商平台已经成为人们日常生活不可或缺的一部分。为了加深对面向对象程序设计与实践的理解，本项目基于 C++ 语言，采用面向对象设计思想，分阶段实现了一个功能完善的电商交易平台。

本项目的开发分为三个阶段，每一阶段的实现都在前一阶段的基础上逐步扩展，最终完成了一个具备完整用户管理、商品管理、交易管理和网络通信能力的综合性电商平台。虽然开发流程分为三阶段，但系统设计始终围绕同一个目标题目展开，所有阶段的代码和架构都是连续演进、互相兼容的。

平台主要实现了如下核心需求：

- **用户系统**：支持新用户注册、用户登录、密码修改、余额查询和充值。用户分为商家和消费者两类，信息持久化保存，权限分明。
- **商品系统**：支持多种商品类型（如图书、食品、服装），商品信息由商家发布并管理，支持商品增删改查、品类折扣、信息持久化等。
- **购物与订单系统**：消费者可通过购物车管理拟购买商品，实现订单生成与管理、余额支付、支付后的资金转移、订单状态跟踪、库存冻结与恢复等关键业务逻辑。
- **网络通信与并发处理**：第三阶段实现了典型的 CS 结构，客户端与服务器端采用 socket 进行通信，服务器端支持多线程并发、会话互斥管理（同一账户不可多端登录）、购物车/订单等状态的持久化存储。

本系统不仅在功能上覆盖了常见电商平台的基础业务，还在设计和实现过程中严格遵循面向对象的封装、继承和多态思想，采用模块化结构、分层管理、友元函数与接口协议规范等现代编程理念，具备良好的可扩展性与维护性。

通过本实验项目，不仅锻炼了面向对象建模、复杂系统分层设计和 C++ 高级特性的实践能力，也积累了多线程、文件存储、网络通信、协议设计等实际工程经验，为后续更大型项目开发打下坚实基础。

2 总体设计

本电商交易平台系统采用面向对象的分层和模块化架构设计，整个系统主要划分为五大核心子系统：用户子系统、商品子系统、订单与购物车子系统、网络通信子系统和数据存储子系统。各子系统各司其职，并通过明确定义的接口进行交互，协同完成平台的各项业务功能。

2.1 子系统划分与功能简介

- **用户子系统：**负责平台用户的注册、登录、密码与余额管理、用户身份权限控制（顾客/商家）、会话互斥管理等功能。核心类包括 `User`（基类）、`Customer`、`Seller`、`UserManager`。
- **商品子系统：**实现商品的增删查改、商品分类（如图书、食品、服装）、折扣策略、库存管理、商品持久化等功能。核心类包括 `Product`（基类）、`Book`、`Food`、`Clothes`、`ProductManager`。
- **订单与购物车子系统：**负责顾客购物车的管理、订单的生成与支付、订单取消、库存冻结与恢复、订单和购物车的数据持久化。核心类包括 `CartItem`、`ShoppingCart`、`Order`、`OrderManager`。
- **网络通信子系统：**采用 C/S 结构，通过 Socket 和自定义 JSON 协议实现客户端和服务器之间的消息通信，支持多线程并发、会话唯一性控制。核心类包括 `SocketStream`、网络协议处理模块等。
- **数据存储子系统：**将所有核心业务数据（用户、商品、订单、购物车、会话锁等）以文件形式持久化保存，确保系统重启后数据不丢失。典型数据文件包括 `users.txt`、`products.txt`、`orders.txt`、`carts/`、`sessions/` 等。

2.2 子系统之间的关系与协作

各子系统之间通过明确的接口进行解耦协作，彼此依赖关系如下：

- 用户通过客户端登录，所有业务操作请求经由网络通信子系统转发到服务器端；
- 用户子系统提供身份认证与权限分配，驱动所有业务行为；
- 商品子系统为订单和购物车子系统提供商品信息、库存、价格、折扣等支撑；
- 订单与购物车子系统实现用户的购物与交易过程，涉及对商品和用户余额的联动修改；
- 数据存储子系统为所有核心数据对象提供持久化支撑，所有变动及时保存至本地数据文件；
- 网络通信子系统连接客户端与服务器，保障多用户并发场景下的业务一致性与互斥性。

2.3 系统架构关系图

系统结构关系如下图所示：

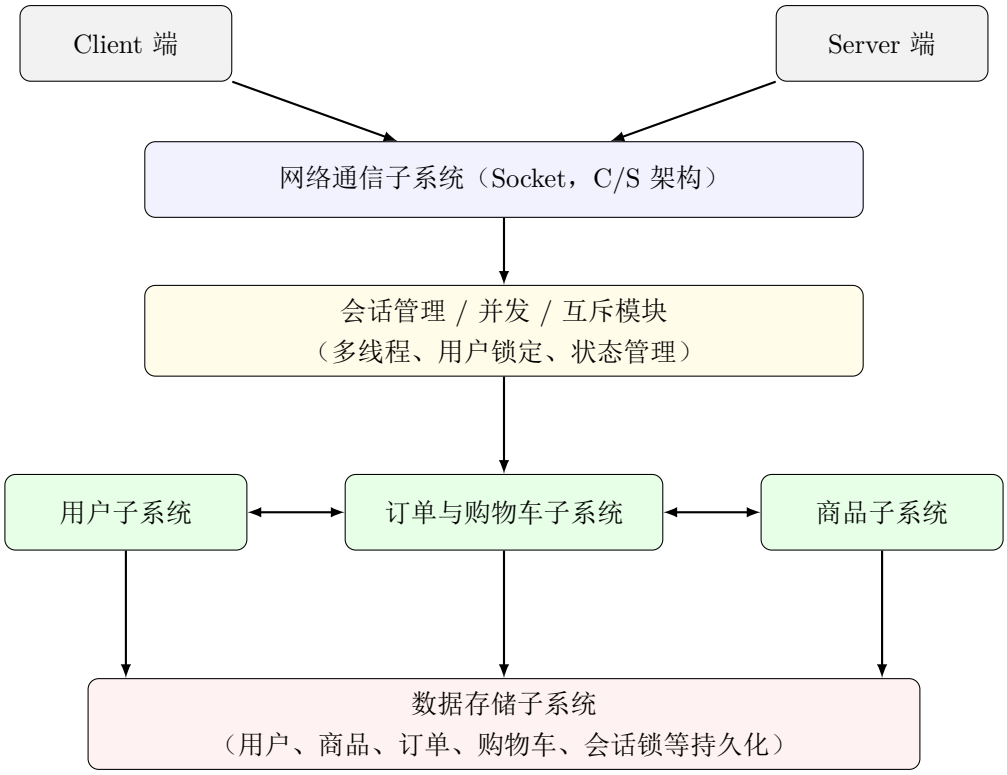

图 1: 系统架构分层与子系统关系图

各子系统之间层次分明，模块边界清晰，既便于扩展升级，又便于维护和测试。

3 详细设计

3.1 用户子系统详细设计

用户子系统负责平台的用户注册、登录、权限区分、信息维护等核心功能。其主要任务是实现对用户身份、账户状态、用户信息（如用户名、密码、余额、用户类型等）的管理，并支持用户的认证、密码修改、余额充值等操作。

主要类及关系：

- **User（用户基类）**：定义了所有用户共有的属性和操作，如用户名、密码、余额、登录接口、权限判断等，是后续派生类的公共基类。
- **Customer（顾客类）**：继承自 User，代表普通顾客，拥有购物车管理、下单、订单支付等功能。
- **Seller（商家类）**：继承自 User，代表平台商家，拥有商品上架、库存管理、折扣设置、销售管理等功能。
- **UserManager（用户管理器）**：负责所有用户的注册、登录、查找、用户数据的持久化与加载（文件读写），以及用户集合的统一管理。对外提供用户身份校验、注册、密码修改、余额修改、用户数据保存等接口，是连接系统其余子模块（如订单、商品管理等）的桥梁。

类图与关系示意：

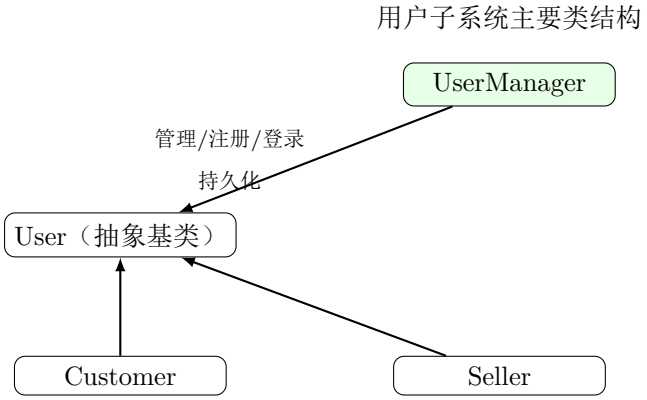

图 2: 用户子系统关系图

主要接口说明：

- `UserManager::RegisterUser()`:注册新用户,校验用户名唯一性并支持类型选择（顾客/商家）。
- `UserManager::Login()`：用户登录验证，检查用户名和密码是否匹配。
- `UserManager::SaveToFile() / LoadFromFile()`：将所有用户数据持久化到文件或从文件恢复。

- `User::CheckPassword()`：密码校验。
- `User::Recharge()`：余额充值。
- `User::SetPassword()`：修改密码。

用户子系统通过 `UserManager` 实现了所有用户的统一管理、认证、状态维护和信息持久化，是平台安全、账户分离、权限控制和用户体验的基础。其对外接口为订单、购物车、商品子系统等提供了身份支撑与数据保障。

3.2 商品子系统详细设计

商品子系统负责平台所有商品的统一管理，包括商品的上架、展示、搜索、库存调整、折扣设置等功能。该子系统为用户（顾客/商家）提供了商品浏览、搜索、购买等基础服务，同时支持商家对其商品的维护操作。

主要类及关系：

- **Product（商品基类）**：抽象类，包含商品的基本属性（如名称、描述、类别、单价、库存、拥有者、商品编号等）和通用操作。派生出具体商品类型。
- **Book / Food / Clothes（具体商品类）**：分别代表图书、食品、服装，继承自 Product。可扩展更多类别，每类可有自身特性或重载方法。
- **ProductManager（商品管理器）**：负责所有商品对象的管理、搜索、增删改查、库存与折扣管理、商品数据的持久化与加载（文件读写），以及商品与卖家、订单之间的关联。

类图与关系示意：

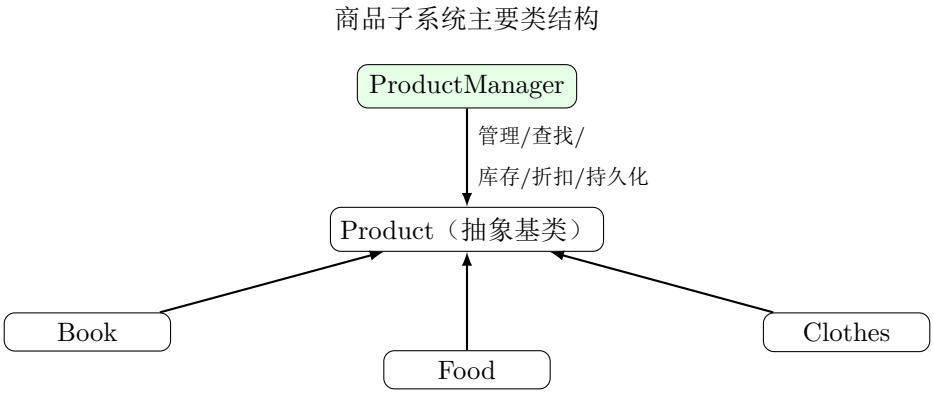

图 3: 商品子系统关系图

主要接口说明：

- `ProductManager::AddProduct()`：上架新商品，指定类别、名称、描述、价格、库存等信息。
- `ProductManager::SearchByName()`：支持按名称或关键词搜索商品，返回匹配集合。

- `ProductManager::GetProductByName()`：根据名称查找单个商品对象。
- `ProductManager::ApplyDiscount()`：为指定类别商品批量设置折扣率。
- `ProductManager::SaveToFile() / LoadFromFile()`：商品数据的持久化与恢复。
- `Product::SetStock()/GetStock()`：设置/获取商品库存。
- `Product::SetOwner()/GetOwner()`：设置/获取商品所有者（商家）。

商品子系统通过 `ProductManager` 实现所有商品的集中存储、查找、展示和库存、折扣等管理，确保了商品信息的完整性、一致性和高效可用。系统各业务模块（如用户子系统、订单与购物车子系统）均可通过该子系统接口访问和操作商品数据。

3.3 订单与购物车子系统详细设计

订单与购物车子系统负责平台中顾客的购物行为流程，包括商品加入购物车、购物车管理、订单生成、订单支付、订单状态变更等功能。该子系统直接关联商品子系统和用户子系统，是交易流程的核心实现部分。

主要类及关系：

- **CartItem（购物车条目类）**：表示购物车中单一商品条目，包含商品指针、数量、价格等信息。
- **ShoppingCart（购物车类）**：维护顾客当前购物车所有商品及其数量，提供添加、删除、更新、清空等操作，并支持购物车内容的持久化。
- **Order（订单类）**：封装一次购物流程的全部信息，包括订单编号、顾客、商品明细（CartItem 集合）、订单总金额、状态（未支付、已支付、已取消）等。
- **OrderManager（订单管理器）**：负责所有订单的创建、支付、取消、加载、保存等管理操作。订单的生命周期、支付逻辑、商家分账、库存变更等也由此类协调完成。

类图与关系示意：

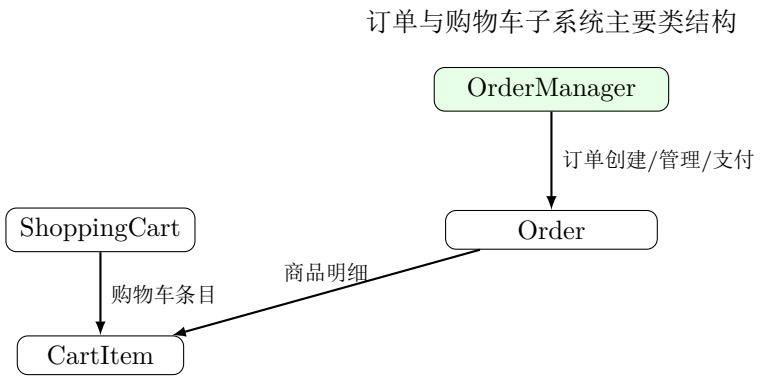

图 4: 订单与购物车子系统关系图

3.4 网络子系统详细设计

网络子系统负责实现平台客户端与服务器端之间的通信，采用传统的 C/S 架构，通过 Socket 编程进行消息收发，支持多用户并发访问、会话管理、错误处理等功能。该子系统是平台从单机模式升级到网络分布式的重要基础，保障了业务模块的远程调用和数据同步。

主要功能与职责：

- 负责客户端与服务器之间的连接建立、断开与管理。
- 使用 TCP Socket 实现可靠的数据传输，采用 JSON 作为消息格式，确保跨平台数据结构解析。
- 支持多线程并发处理多个客户端会话，保证各用户操作互不干扰。
- 实现用户登录互斥（防止同一用户多端同时登录）、会话状态同步、用户锁定等机制。
- 提供底层 SocketStream 抽象，屏蔽底层网络细节，对业务模块透明，提升代码复用与健壮性。
- 支持通信异常、网络中断、非法请求等多种错误场景处理，保障系统鲁棒性。

主要类及结构：

- **SocketStream**：自定义的网络通信封装类，负责 socket 的创建、连接、监听、读写数据（支持按行、按块收发），并提供初始化、关闭等操作接口，内部自动管理资源释放。
- **客户端/服务器主循环**：服务器端主进程监听端口、接受新连接，为每个会话分配独立线程，由线程执行 Handle 逻辑进行完整一次业务交互，线程退出时释放资源。客户端主循环负责用户输入、命令解析、与服务器 JSON 消息收发、结果展示等。
- **会话互斥模块**：采用会话锁文件方式（如 `data/sessions/username.lock`），防止同一账号多端登录，退出/异常时自动释放锁，保证会话一致性和安全性。

通信协议说明：

- **底层协议**：采用 TCP/IP 进行数据传输，保证消息可靠送达。
- **消息格式**：统一采用 JSON 格式进行请求与响应的编解码，字段包括 type（请求类型）、data（具体数据）、ok（结果标识）、msg（错误提示）等。
- **业务协议**：平台所有操作（如登录、注册、查询、购物、订单管理等）均以 JSON 消息包形式在客户端和服务器之间双向流转，详见接口协议说明章节。

网络子系统为平台提供了高效、可靠、可扩展的通信能力，实现了分布式部署与远程访问，为用户和业务逻辑子系统之间的交互建立了桥梁。其高鲁棒性的并发处理与错误管理机制，显著提升了平台的实际可用性与工程质量。

3.5 数据库说明

本系统采用结构化文本文件作为数据持久化介质，模拟关系型数据库表的存储与操作。每类业务数据采用独立的文本文件存储，每行表示一条数据记录，字段间以空格或竖线等分隔符分隔。各管理模块在内存中实现相应“表”的增删改查功能。下表详细列出各文件的数据结构与字段含义：

1. 用户数据表（users.txt）

字段名	类型	说明	举例
username	string	用户名，唯一	Coco
password	string	密码（明文）	coco
balance	double	账户余额	3400.0
userType	string	用户类型（Seller/Customer）	Customer

2. 商品数据表（products.txt）

字段名	类型	说明	举例
category	string	商品类别	Food
name	string	商品名称	Milk
desc	string	商品描述	Pure
price	double	单价	10.0
stock	int	库存数量	100
owner	string	所有者用户名	Admin
productId	string	商品唯一编号	Food-0

3. 订单数据表（orders.txt）

字段名	类型	说明	举例
orderId	string	订单编号	ORD-000000
customer	string	顾客用户名	Coco
itemCount	int	商品项数	1
orderTime	int	下单时间（时间戳）	1749480442
total	double	订单总金额	200.0
status	string	订单状态（Paid/Unpaid/Failed）	Paid

订单明细项：

字段名	类型	说明	举例
productId	string	商品编号	Food-0
quantity	int	数量	20

4. 购物车数据表（carts/[username].txt）

字段名	类型	说明
productId	string	商品编号
quantity	int	数量

5. 会话锁定（sessions/[username].lock）

字段名	类型	说明
lockFlag	string	锁标志，内容为"LOCKED" 时表示账号已登录

备注：

- 所有数据文件均为结构化文本文件，便于人工审阅与调试。
- 如需升级为关系型数据库，可按上述字段设计建表语句，数据类型可直接映射为 SQL 标准类型（VARCHAR/DOUBLE/INT 等）。
- 每次数据变更操作（注册、上架、下单、支付等）均立即同步持久化，确保数据一致性和可靠性。

通过该数据存储子系统，平台能够高效、安全、结构化地管理所有业务核心数据，支撑系统稳定运行和用户良好体验。

3.6 接口协议说明

本系统采用自定义的应用层协议，客户端与服务器端之间的通信全部基于 TCP Socket，通过 JSON 格式的消息进行数据封装与解析。所有请求与响应均为结构化的 JSON 对象，保证了协议的跨平台兼容性和良好的可扩展性。

底层承载协议：

- 采用 TCP/IP 协议作为通信的底层承载，保障消息可靠送达和顺序一致性。
- 客户端与服务器通过 Socket 建立连接，所有业务数据通过该连接收发。

消息格式规范：

- 所有请求与响应均采用 JSON 文本格式，每个消息为一行字符串，通过 SocketStream 的 SendLine/RecvLine 实现。
- 通用消息结构如下：

```
{
  "type": "请求类型",
  "data": { ... }
}
```

- 响应消息增加如下字段：

```
{
  "ok": true/false,     // 请求是否成功
  "data": { ... },      // 返回结果数据
  "msg":  "错误提示"    // 可选，出错时携带错误原因
}
```

典型协议数据单元说明：

- **注册请求/响应**

  请求：`{ "type": "Register", "data": { "username": "...", "password": "...", "userType":`
  响应：`{ "ok": true/false, "msg": "..." }`

- **登录请求/响应**

  请求：`{ "type": "Login", "data": { "username": "...", "password": "..." } }`
  响应：`{ "ok": true/false, "data": { "userType": "Customer/Seller" }, "msg": "..." }`

- **商品查询/响应**

  请求：`{ "type": "List", "data": {} }`
  响应：`{ "ok": true, "data": [ { "cat": "...", "name": "...", "desc": "...", "price": ..`

- **购物车管理、订单、支付等**所有请求均类似，”type” 字段标明操作类型，”data” 携带操作参数。所有响应都带”ok” 字段指示结果。

协议语义说明：

- 客户端每次发送操作请求，服务端解析 type 字段分派到相应处理逻辑，处理结果打包成 JSON 响应返回。
- 出错时，”ok” 置为 false，”msg” 字段详细说明原因。
- 所有操作请求与响应均为一问一答模式，保证了状态一致性和时序逻辑清晰。
- 连接建立后保持长连接，直到用户显式退出或网络断开。

补充：

- 所有通信协议细节在客户端/服务器的网络主循环及 SocketStream 模块内实现，业务代码与网络代码严格解耦。
- 如需协议升级，仅需约定 JSON 字段，不影响底层 Socket 通信及主业务逻辑。

4 实现

4.1 实验过程中遇到的主要问题和解决方案

在系统开发过程中，遇到如下主要问题，并逐一进行了有效的分析和解决：

1. **并发会话与数据一致性**
   服务器端需要支持多用户并发访问，且保证数据一致性。采用多线程为每个客户端分配独立会话，所有用户管理、订单管理、商品管理等核心数据模块均加锁处理（如 `std::mutex` 保护用户/订单容器），防止并发修改导致的数据错乱。

2. **用户多端登录互斥机制**
   平台要求同一账号不能同时在多个终端登录。为此，设计了基于 lock 文件的互斥机制（如 `data/sessions/[username].lock`）。登录前检测锁文件是否存在，若存在则拒绝登录，登出或异常断开时自动删除锁文件，防止“僵尸会话”。

3. **数据持久化与文件格式解析**
   初始阶段由于没有关系型数据库，全部数据通过文本文件存储。为保证可靠性和可扩展性，所有存取均采用结构化格式，分隔符严格规定。针对文件读写的各种边界条件（如数据缺失、重复、非法字符），逐步补充了健壮的输入校验和异常处理，确保系统在异常输入下也能稳定运行。

4. **客户端/服务器端界面友好性与输入健壮性**
   为保证用户体验，客户端所有输入均有详细提示，非法输入自动重试，关键功能均有明确错误提示（如余额不足、商品不存在、库存不足、重复下单等），保证系统使用流畅、易懂。

5. **订单与购物车数据同步和自动清理机制**
   用户支付后购物车内容应自动清空。通过在支付/下单逻辑中加入清理调用，同时对购物车持久化文件进行同步删除或覆盖，避免历史遗留数据干扰用户后续操作。

6. **断线重连与数据恢复能力**
   系统支持断线后重新登录，购物车数据自动从持久化文件恢复，未支付订单保留，保障了良好的用户体验和平台健壮性。

综上，针对实验过程中遇到的各类典型问题，均通过模块化设计、异常处理、数据校验、互斥机制等手段实现了高质量、健壮的系统架构，保证了多用户环境下的正确性和易用性。

4.2 想法

在本系统的设计与实现过程中，深刻体会到模块化、面向对象编程和工程化思想的重要性。通过将用户、商品、订单、购物车等业务逻辑进行清晰划分和解耦，大大提升了系统的可维护性与可扩展性。采用 C/S 架构和 Socket 通信，既锻炼了网络编程能力，也使系统具备了多用户并发和分布式部署的基础。在数据存储方面，通过类比数据库表设计结构化文本文件，为后续系统升级和迁移数据库打下良好基础。

4.3 经验

- 模块划分清晰、接口设计合理，可以有效降低后续功能扩展和维护成本。
- 持久化机制要与内存模型保持同步，关键节点须及时保存和恢复，避免数据丢失。
- 多线程与多用户并发访问时，必须严格加锁，提前考虑数据一致性问题。
- 错误处理和输入校验不容忽视，友好的用户交互与明确的错误提示能极大提升体验。
- 开发过程中，先实现单机功能再逐步升级为网络版，是比较稳妥的策略。

4.4 教训

- 初期未充分考虑并发与异常场景，导致后续补充加锁和错误处理时较为繁琐，说明系统设计时需有前瞻性。
- 用户购物车的持久化和订单生成过程涉及多步操作，为避免“脏数据”或部分提交，所有涉及资金扣除、库存扣减、订单生成等步骤均严格顺序执行，操作失败时及时回滚或报错，保证业务原子性。

4.5 致谢

在此由衷感谢双锴老师和助教老师的悉心指导与认真负责的验收。他们不断敦促我们关注工程细节、重视规范，同时也鼓励和启发我们思考更优的系统架构与实现方案。老师们的高标准、严要求使我收获颇丰，也为日后更复杂、安全性要求更高的开发打下了坚实基础。