**面向对象程序设计实践（C++）**

**课程实验报告**

|  |  |
| --- | --- |
| 题目 | 电商交易平台设计与实现 |
| 姓名 | 肖璨 |
| 学号 | 2023211209 |
| 班级 | 2023211311 |

目录

[1 前言 3](#_Toc200875511)

[2 总体设计 4](#_Toc200875512)

[2.1 第一阶段 4](#_Toc200875513)

[2.2 第二阶段 4](#_Toc200875514)

[2.3 第三阶段 5](#_Toc200875515)

[3 详细设计 7](#_Toc200875516)

[3.1 账户管理子系统user详细设计 7](#_Toc200875517)

[3.1.1 User子系统设计概述 7](#_Toc200875518)

[3.1.2 类结构与功能模块说明 7](#_Toc200875519)

[3.2 商城管理子系统Store详细设计 9](#_Toc200875520)

[3.2.1 Store子系统设计概述 9](#_Toc200875521)

[3.3 页面管理子系统page/ui详细设计 14](#_Toc200875522)

[3.3.1 终端Terminal页面显示 14](#_Toc200875523)

[3.3.2 图形化界面ImgUi页面显示 15](#_Toc200875524)

[3.4 订单管理子系统order详细设计 16](#_Toc200875525)

[3.4.1 订单属性类OrderItem 17](#_Toc200875526)

[3.4.2 订单逻辑类Order 17](#_Toc200875527)

[3.4.3 订单管理类Ordermanager 18](#_Toc200875528)

[3.5 网络通信管理子系统network详细设计（接口协议说明） 20](#_Toc200875529)

[3.5.1 通信协议命名空间Protocol 21](#_Toc200875530)

[3.5.2 客户端通信管理模块Client 23](#_Toc200875531)

[3.5.3 服务端通信与会话管理模块Server 24](#_Toc200875532)

[3.6 数据库说明 26](#_Toc200875533)

[3.6.1 用户数据文件user.txt 27](#_Toc200875534)

[3.6.2 购物车数据文件 carts/<username>\_cart.txt 28](#_Toc200875535)

[3.6.3 订单数据文件（orders/ORD-xxxx.txt） 28](#_Toc200875536)

[3.6.4 商品数据文件（store/sellers/商家名.txt） 29](#_Toc200875537)

[4 实现与测试 31](#_Toc200875538)

[4.1 项目大致流程 31](#_Toc200875539)

[4.1.1 单机版大致流程 31](#_Toc200875540)

[4.1.2 网络版大致流程 33](#_Toc200875541)

[4.2 项目基本实现 35](#_Toc200875542)

[4.3 实现过程中遇到的主要问题和解决方案 36](#_Toc200875543)

[4.3.1 Ui选择 36](#_Toc200875544)

[4.3.2 网络通信协议设计 37](#_Toc200875545)

[5 实验经验与个人总结 37](#_Toc200875546)

[5.1 拓展想法 37](#_Toc200875547)

[5.1.1 通信协议改进 37](#_Toc200875548)

[5.1.2 数据库管理改进 39](#_Toc200875549)

[5.2 个人经验与教训总结 39](#_Toc200875550)

# 前言

近年来，随着移动互联网技术的高速发展，电子商务平台已广泛融入人们的日常生活，并成为现代社会商业活动的重要组成部分。电商系统作为连接消费者与商家的桥梁，其系统设计与开发已成为高校计算机专业教学中的重要实践内容。为了帮助学生深入理解面向对象程序设计的思想，掌握C++语言在中大型系统开发中的实际应用能力，本课程设计以“电商交易平台的设计与实现”为主题，要求学生基于C++语言和面向对象编程方法，开发一个具有典型功能的电商交易平台。

本课程设计任务分为三个阶段，采用逐步增强的方式，逐层引导学生掌握复杂系统的设计与实现方法。第一阶段聚焦于账户管理和商品管理子系统，完成用户注册登录、余额管理、商品展示与维护等基础功能；第二阶段在前者基础上引入交易管理子系统，实现购物车管理、订单生成与支付功能，初步模拟电商交易流程；第三阶段则将系统扩展为网络版本，采用客户端-服务器（Client-Server，简称CS）架构，实现多进程环境下的分布式交互，并通过Socket通信方式支持客户端远程访问平台核心服务。

本次实验特别强调系统的模块化、继承体系的构建与文件持久化的应用。在用户与商品管理模块中，学生需设计用户与商品的基类，并通过继承机制区分消费者与商家、图书与服装等具体类型，体现出面向对象的设计原则。此外，实验还对程序的规范性提出严格要求，包括对友元函数的合理使用、const修饰的明确标注、注释的完整性等，旨在培养学生良好的编码习惯与工程素养。

# 总体设计

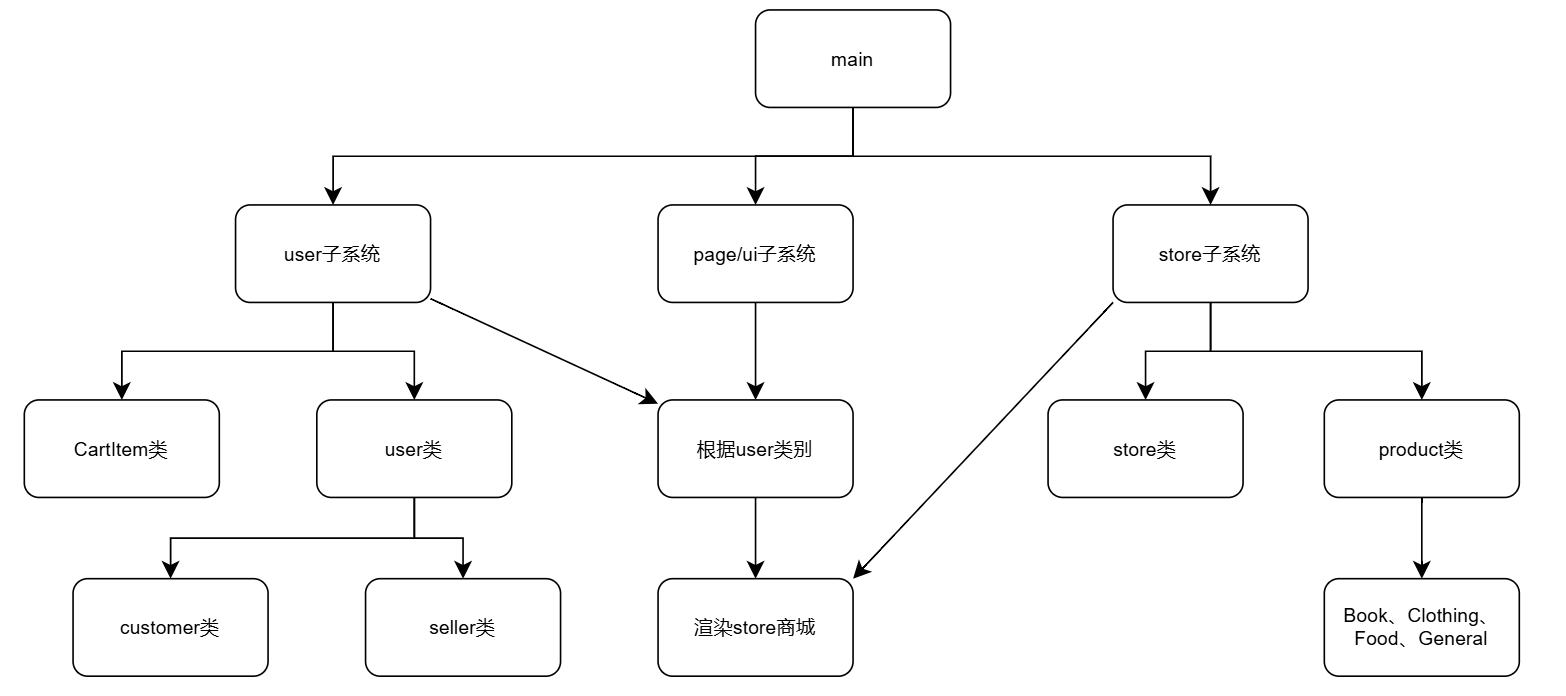
由于实验分为三个阶段，因此这里也将设计分为三个阶段，分别对各个阶段的核心部分进行讲解：

## 第一阶段

第一阶段要求实现的功能主要在于账户管理子系统和商品管理子系统，这要求我们实现一个user部分的管理和store部分的管理，进一步细化可以衍生出user类、store类以及product类，分别负责用户部分管理、商城平台部分管理和商品管理。同时，各个类别中又可以继续细分到各自的继承体系。

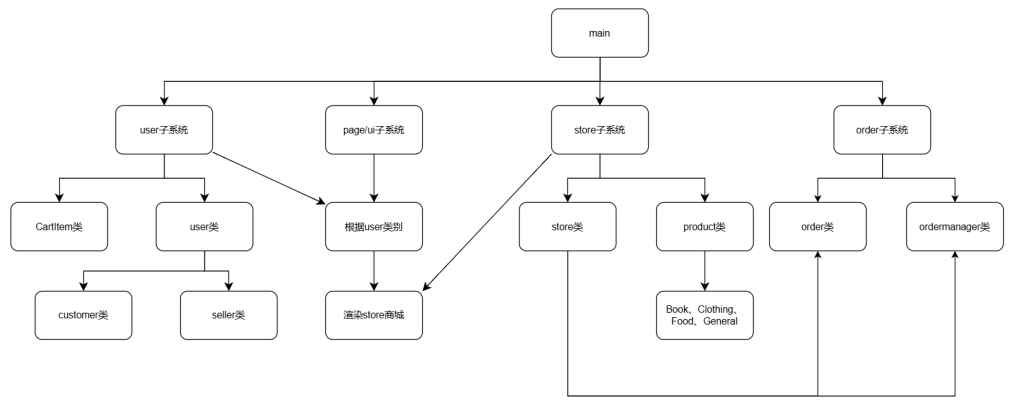
显示页面子系统并不是题目指定部分，但是根据面向对象编程，我还是将其从主函数中分离出来，使用page类和UI类分别实现。起初，page类也是根据用户类型继承实现的，但是在UI类中摒弃了这一点，转而使用更直接的方式实现页面UI。

第一阶段子系统联系具体图示如下：



## 第二阶段

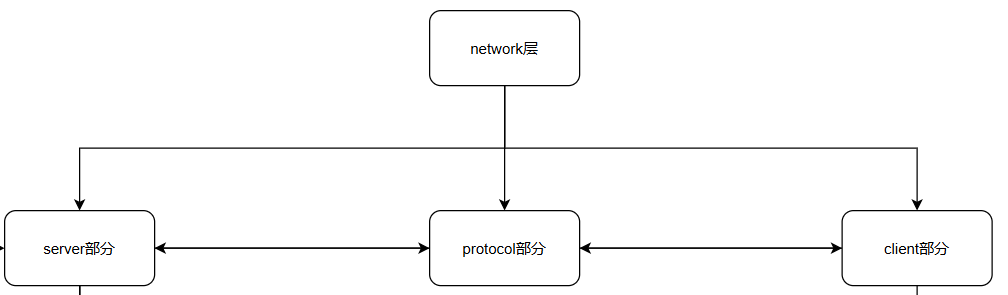
第二阶段在第一阶段的基础上，添加了一个新的子系统：订单管理子系统，而其下又细分了order部分和ordermanager部分进行订单划分和订单管理。在此之上，第二阶段又添加了购物车部分CartItem，以将订单生成与直接购买和购物车结算结合起来。同时，第二阶段还通过购物车类和订单类加强了user层与store层之间的联系。



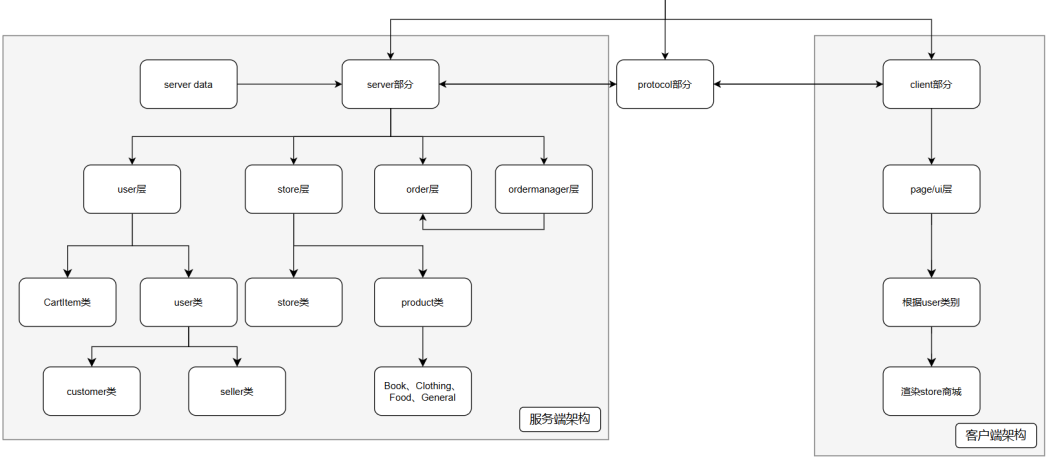
## 第三阶段

第三阶段要求实现网络版，将程序重构为CS结构，分离成类似前端和后端的客户端/服务端两个部分。这要求在保留之前的处理结构的前提下，引入socket通信，使得客户端的任何请求都需要发送到服务端上，并在服务器响应回答之后才能够得到同步。同时，不论客户端还是服务端都需要具备一定的信息封装能力和信息解析能力，将请求和响应更换成更符合网络通信的形式进行传输。

网络部分主要如下：



下层整体部分保持不变，但也被分成了服务端的架构和客户端的架构，具体部分如下：



代码项目架构图（文件夹与关键文件的树状结构），这里先只展示网络版：

lab/

├── client\_main.cpp # 客户端主程序入口

├── client.exe

├── imgui.ini # ImGui 配置文件

├── server\_main.cpp # 服务端主程序入口

├── server.exe

├── client/ # 客户端 UI 相关代码

│ ├── client\_ui.cpp

│ ├── client\_ui.h

│ └── imgui.ini

├── glfw/ # 第三方库 GLFW（窗口/输入处理）

├── imgui/ # 第三方库 ImGui（图形界面）

├── network/ # 网络通信相关代码

│ ├── client.cpp/.h # 客户端网络版逻辑

│ ├── protocol.cpp/.h # 通信协议定义

│ ├── server.cpp/.h # 服务端网络版逻辑

├── order/

│ ├── order.cpp/.h # 订单类实现

│ ├── ordermanager.cpp/.h # 订单管理器

├── server\_data/ # 服务端数据存储目录

│ ├── users.txt # 用户数据

│ ├── carts/ # 购物车数据

│ ├── orders/ # 订单数据

│ └── store/ # 商品数据

│ └── sellers/ # 各卖家商品数据

│ └── ab.txt

├── store/ # 商品相关代码

│ ├── store.cpp/.h

│

├── ui/ # UI 相关代码（通用/主界面）

│ ├── ui.cpp/.h

│

└── user/ # 用户相关代码

├── user.cpp/.h

└── users/

# 详细设计

分子系统给出详细设计，每个子系统用到的类，类之间的的关系（说明+图示）

整体项目除了第二阶段与第三阶段的整体框架有些不同之外，其余都是一一递进的关系。接下来我将根据阶段的层次递进关系，进行各个部分详细的子系统说明。

## 账户管理子系统user详细设计

### User子系统设计概述

本实验中，User子系统的基础部分主要负责用户的账户管理，包括注册、登录、密码修改、充值与扣费、账户信息的保存与加载等功能。系统采用面向对象的设计思想，以User为基类，通过继承分别实现三种用户类型：Customer（消费者）、Seller（商家）与Admin（管理员）。

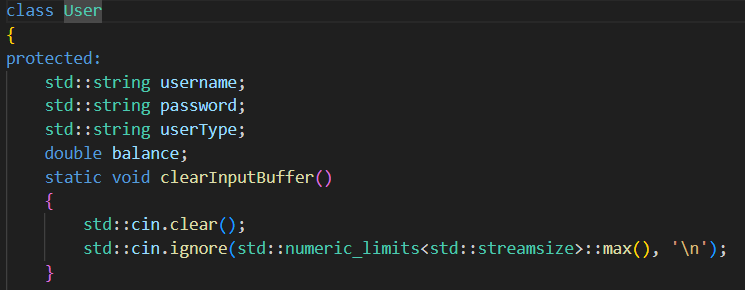
在后续阶段中user子系统中加入了购物车功能CartItem类，对消费者的特定功能进行细化，将其与商家和管理员身份具体区分开来。该类除了继承账户管理能力外，扩展了完整的购物车功能，包括本地化的购物车文件读取与保存机制、购物车项结构定义、购物车操作方法（如添加、删除、更新商品数量）以及订单结算的基本流程。内部的购物车通过 CartItem 结构体来表示单个商品项，记录商品编号、商品名称、数量、加入购物车时的价格以及卖家信息。这种设计既简洁明了，又方便与订单系统进行衔接。

User子系统部分的代码主要放在user目录下，包含两个文件：user.cpp、user.h、user.txt。其中，user.h封装了用户属性与行为，user.cpp主要包含账户管理子系统的运行逻辑，而user.txt用来存储注册的用户数据。存储功能在最终阶段的位置有所改变，但整体框架不变。

### 类结构与功能模块说明

#### 抽象基类 User

User类作为系统核心抽象类，封装了通用的用户属性与行为，如用户名、密码、余额及一系列静态和成员函数。通过virtual std::string getUserType() const = 0;将User类定义为抽象类，实现用户类型的多态识别。

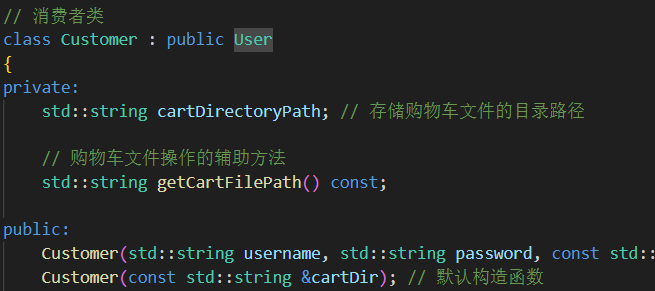


主要成员函数包括：

* 注册与登录相关：registerUser()、userLogin()、login()；
* 账户操作：deposit()、withdraw()、changePassword()、checkBalance()；
* 文件操作：saveUsersToFile()、loadUsersFromFile()；
* 辅助操作：findUser()、displayInfo()、isUsernameExists()等。

#### 消费者子类 Customer

在继承体系中，Customer 类继承自 User，是面向普通消费者用户的具体实现。在最初版本中，消费者的功能函数与其他两个用户类并无区别；但是在要求加入购物车功能后，该类除了继承账户管理能力外，扩展了完整的购物车功能，包括本地化的购物车文件读取与保存机制、购物车项结构定义、购物车操作方法（如添加、删除、更新商品数量）以及订单结算的基本流程。



其主要功能拓展如下：

* 新增数据结构：CartItem 表示购物车商品项，成员包括商品ID、名称、数量、价格和卖家用户名。
* 成员变量：shoppingCartItems 用于存储购物车条目；cartDirectoryPath 指定购物车文件的读写路径。
* 关键方法：
  + loadCartFromFile() / saveCartToFile()：购物车数据持久化。
  + addToCart() / removeCartItem() / updateCartItemQuantity()：购物车增删改。
  + viewCart()：展示购物车内容并支持结算逻辑。
  + clearCartAndFile()：结算后清空购物车。

#### 子类seller和子类admin

Seller 类同样继承自 User，目前主要作为商品管理等商家操作的接口预留。虽然在当前版本中其功能相对简单，仅继承账户相关操作，但从架构上已为将来添加商品发布、库存管理等功能留下了接口空间。

最后，Admin 类代表系统管理员账号，用于处理平台级的操作权限，如用户管理、平台状态监控等。其结构上也继承自 User，体现出系统中所有用户角色的统一管理的面向对象思想，同时又通过多态机制赋予不同角色以独特功能。

## 商城管理子系统Store详细设计

### Store子系统设计概述

本系统的商品子系统（Store 子系统）以面向对象的方式构建了一个灵活、可扩展的商品管理体系，涵盖了商品分类、商品对象的抽象与继承实现，以及商家商品集合的统一管理与操作。系统以 Product 类为核心抽象基类，定义了所有商品共有的基本属性与操作接口，包括商品名称、描述、原价、数量、折扣率和所属商家账号等信息。该类还提供了一系列用于显示、保存和数据存取的接口，并使用虚函数机制支持多态显示与类型标识，尽管在并不在最终验收需求之内，但仍能使得系统能够在运行时区分不同商品类型并调用相应行为逻辑。

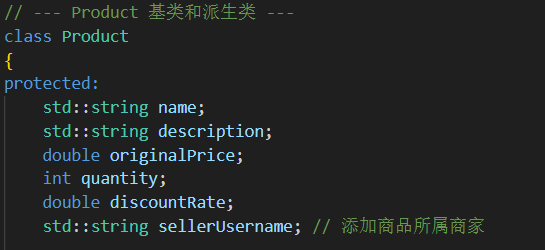
围绕 Product 类，系统定义了多个派生商品类（Book、Clothing、Food、GenericProduct）以满足不同商品类别的具体需求。这些派生类均重写了 display() 与 save() 等核心函数，确保商品信息展示与持久化的准确性。

为了支撑系统级商品管理，系统引入了 Store 类作为商品子系统的控制中心。该类内部维护了一个统一的商品列表 allProducts，以及一个按商家划分的商品映射表 sellerProducts，实现商品对象的集中存储与管理。商品数据的加载与保存通过文件系统与商家账户信息进行绑定，使得不同商家可以拥有各自独立的数据文件。同时，系统通过互斥锁（mutex）与库存锁定表（lockedInventory）机制，实现了多线程环境下对库存访问的并发控制，进而实现了订单生成后的商品冻结功能。

#### 商品类型详述

##### 抽象基类Product

Product 是 store 子系统中所有商品类型的抽象基类，定义了商品的通用属性与接口，如商品名称、描述、价格、库存等一系列成员函数，为派生类提供统一的数据结构和行为框架。

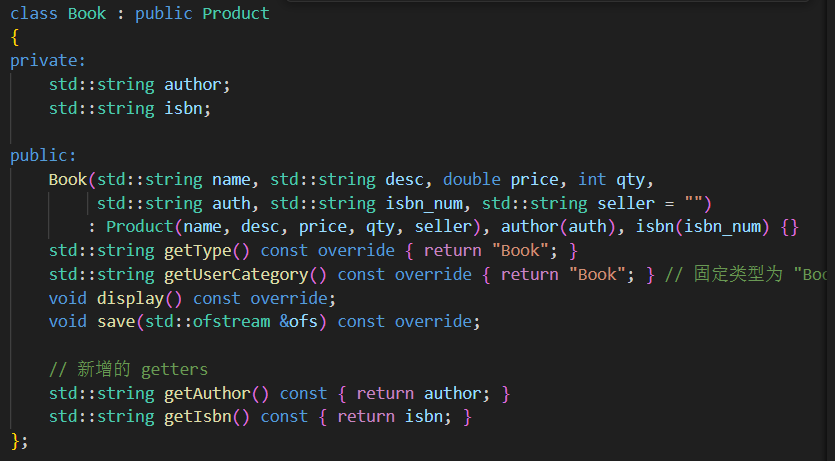


* 主要属性：
  + name, description：商品名称与描述。
  + originalPrice, discountRate：商品原价与折扣率。
  + quantity：当前库存数量。
  + sellerUsername：所属商家用户名。
* 核心功能：
  + 提供获取最终价格的计算逻辑（getPrice()）。
  + 提供虚函数 display() 与 save()，支持派生类自定义展示与持久化格式。
  + 定义 getType() 为纯虚函数，强制派生类标识自身类型。提供一组 Set/Get 接口，允许外部模块对商品属性进行灵活修改。

该类作为商品通用接口，实现了封装性与多态性的良好结合，是整个商品管理系统的基础。

##### 派生标准商品类Book, Clothing, Food

这三类是 Product 的标准派生类，分别扩展了图书、服饰、食品商品的特有属性，并重写了基类中的展示与保存接口。

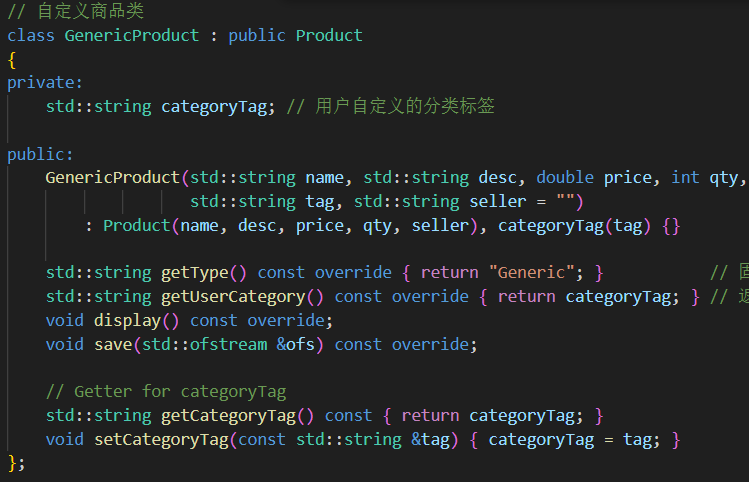


* **Book**：
  + 新增属性：author, isbn。
  + 功能说明：展示和保存时包含图书作者与 ISBN；用户分类为固定值 "Book"。
* **Clothing**：
  + 新增属性：size, color。
  + 功能说明：展示和保存时包括尺码与颜色；用户分类为固定值 "Clothing"。
* **Food**：
  + 新增属性：expirationDate。
  + 功能说明：展示和保存时注明保质期；用户分类为固定值 "Food"。

这三类通过重写 getType() 与 getUserCategory() 实现商品类型与分类的双重标识，便于系统进行分类展示与批量折扣等操作。

##### 通用商品类GenericProduct

该类面向用户自定义商品类型的需求设计，允许商家为商品指定自定义分类标签，并且在分类设置中可以基于此设置不同类别的商品的属性。

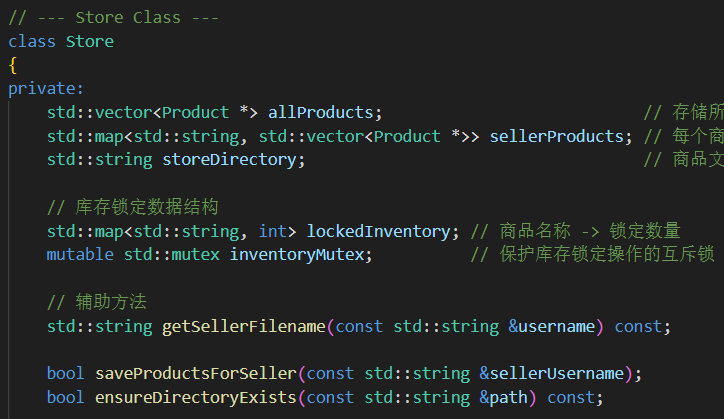


* **新增属性**：
  + categoryTag：用户定义的商品分类标签。
* **主要功能**：
  + 实现 getUserCategory() 返回用户指定标签；
  + 支持展示与保存通用格式信息；
  + 允许外部设置与修改分类标签（通过 setCategoryTag()）；
  + 类型固定为 "Generic"，但具备高度灵活性。

GenericProduct 类通过用户自定义标签实现对商品种类的扩展，具备良好的灵活性和可拓展性。

#### 商品控制类Store

Store 是整个子系统的核心控制类，负责平台上商品的集中管理、文件读写、库存控制以及对商家的商品管理接口。



**关键数据结构**：

* allProducts：平台所有商品的统一列表。
* sellerProducts：以用户名为键的商家商品映射。
* storeDirectory：商品数据存储的主目录。
* lockedInventory：记录已锁定库存的映射表。
* inventoryMutex：保护锁定操作的互斥锁。

在系统控制逻辑层面，Store 类承担起商品的集中式管理职能。它统一维护一个商品总表（allProducts）和按商家划分的商品映射表（sellerProducts），同时通过互斥锁机制与库存锁定表确保商品库存的线程安全。

商品数据以文件形式持久化存储，文件路径与商家账号绑定，支持一个商家的所有物品对应一个商家文件，使平台能够高效管理多个卖家的商品数据并支持增量更新。此外，Store 类还提供商品搜索（包含搜索）、商品折扣设置、分类提取等功能，为上层用户交互界面提供丰富的数据支持。



## 页面管理子系统page/ui详细设计

### 终端Terminal页面显示

终端显示子系统的内容主要放在page目录下，使用简单的std::cin和std::cout流操作在终端中对各个页面进行显示。其中，系统根据当前登录用户的身份，将业务子系统（如用户、商店、订单等）封装成一致、直观的菜单界面。系统以抽象类 Page 作为顶层父类，定义了清晰的多态接口，既保证了各类用户在视觉与交互逻辑上的统一性，又为差异化的功能呈现预留了扩展空间。

Page 类本身并不持有业务数据，而是作为“页面控制器”协调外部对象：它通过引用或指针拿到当前用户对象 User\*、用户列表 std::vector<User\*> 以及商店对象 Store&，执行菜单路由和输入解析；所有与 UI 无关的业务操作仍委托给相应子系统（如用户子系统、Store 子系统）处理，从而保持了“表示-逻辑”分离的良好架构。类中定义了一系列 pagemain 风格的成员函数：pagemain() 负责未登录状态的主界面；pagemainCustomer()、pagemainSeller()、pagemainAdmin() 对应不同角色的登录后主菜单。为了将“进入商城”这一操作具象化为独立页面，Page 声明了纯虚函数 pagestore()，并强制派生类实现，以便在不同用户身份下呈现不同的商品浏览与处理逻辑。

在继承体系中，Customer\_page 与 Seller\_page 继承自 Page，分别面向消费者与商家用户。两者重写 pagestore()：消费者版本聚焦于商品浏览、购物车与结算；商家版本则聚焦于商品上架、库存和价格管理。多态机制使主程序在调用 pagestore() 时无需关心实际对象类型，增强了系统的灵活性和可维护性。

所有页面方法均返回 int 类型状态码，用作主程序的页面跳转或退出条件；输入合法性校验统一通过 clearInputBuffer() 辅助完成，避免因流状态异常导致程序循环失控。

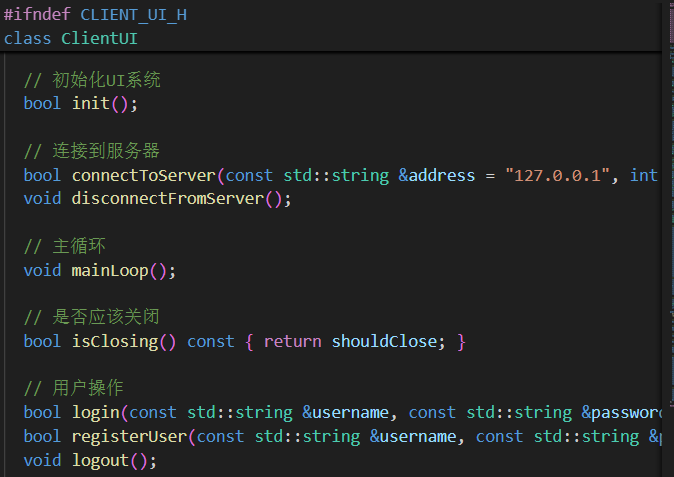
### 图形化界面ImgUi页面显示

图形化界面是在整体项目完成之后才添加的，由于先使用了类似WPF结构（Windows Presentation Foundation），导致目前已经实现的项目结构无法直接移植到QT上进行UI制作。在网上搜索相关资料后，我决定使用ImGUi库在现有代码基础上实现ui功能。

图形化页面显示子系统（ClientUI）是用户交互层的核心，基于 ImGui、OpenGL 与 GLFW 构建，采用即时渲染 GUI（Immediate Mode GUI）范式，主要服务于客户端应用的图形化界面需求。该系统封装于 ClientUI 单一类中，负责窗口创建、渲染调度、事件响应及与后端网络模块的通信，为用户提供流畅、直观的操作体验。

在架构上，ClientUI 具备完整的状态驱动式 UI 显示逻辑。其内部维护了多个关键状态标志（如 currentPage, isLoggedIn, userType, showCartWindow 等）以驱动界面渲染的条件分支，同时通过多个 render 方法模块化各类界面的显示逻辑。渲染方法遵循“只读状态 + 呈现组件”的设计，确保 UI 层与业务数据解耦。界面响应采用主循环驱动 mainLoop()，保证UI界面的实时渲染。

此外，ClientUI 还集成了完整的业务支持能力：用户登录注册、商品查询与展示、购物车管理、订单操作、充值密码修改等功能，均通过封装后的接口方法对外暴露，底层依赖 NetworkClient 进行服务器通信，发送与接收基于 Protocol 结构体的数据协议，确保前后端交互的一致性。



美观性方面，该系统支持暗色 / 亮色主题切换及 UI 缩放，增强跨平台体验的可用性；界面元素如商品卡片、状态栏、确认弹窗等以组件化函数组织，提高代码复用性与可维护性。

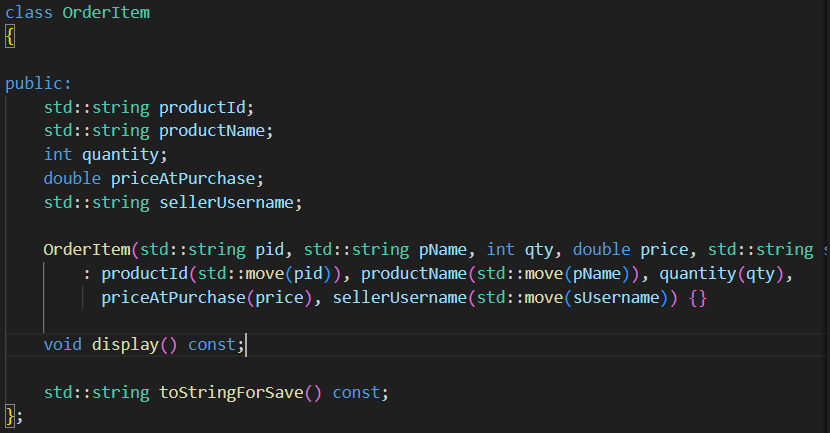
## 订单管理子系统order详细设计

订单子系统主要职责是处理用户从商城发起的订单请求，并根据系统资源（库存、余额）验证和处理订单的生命周期。该子系统由两大类组成：

* Order 类：抽象用户订单的数据模型，包括商品列表、订单状态、时间戳、总金额等字段，同时封装了订单项添加、总价计算、状态更新等核心逻辑。
* OrderManager 类：承担订单调度器的职责，维护一个待处理订单队列，通过后台线程异步地处理每一笔订单，避免阻塞主线程并提高系统响应效率。支持订单的持久化保存和多用户订单历史的查询。

### 订单属性类OrderItem

OrderItem 类用于表示订单中的单个商品项。它包含了商品的核心信息，包括 productId（商品ID）、productName（商品名称）、quantity（购买数量）、priceAtPurchase（购买时的单价）以及 sellerUsername（卖家用户名）。这些成员变量能够完整描述订单中每一项商品的详细情况：

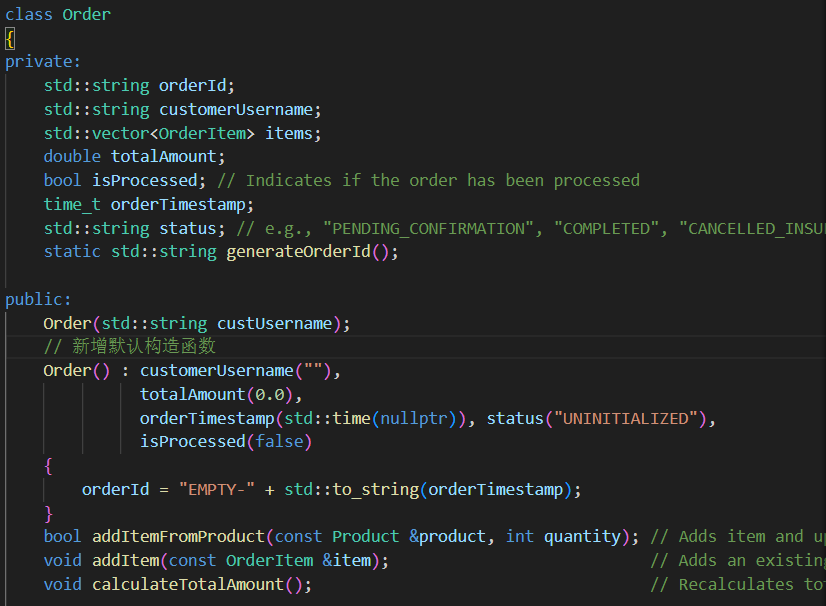


构造函数 OrderItem(std::string pid, std::string pName, int qty, double price, std::string sUsername) 采用参数初始化列表，将传入的参数直接赋值给对应的成员变量，并通过 std::move 优化字符串的构造效率，避免不必要的拷贝。

此外，类还声明了两个成员函数：display() 用于显示订单项的信息，toStringForSave() 用于将订单项序列化为字符串，方便保存到文件或数据库。如此设计是为了方便订单的管理、展示和持久化，为后续逻辑调用奠定基础。

### 订单逻辑类Order

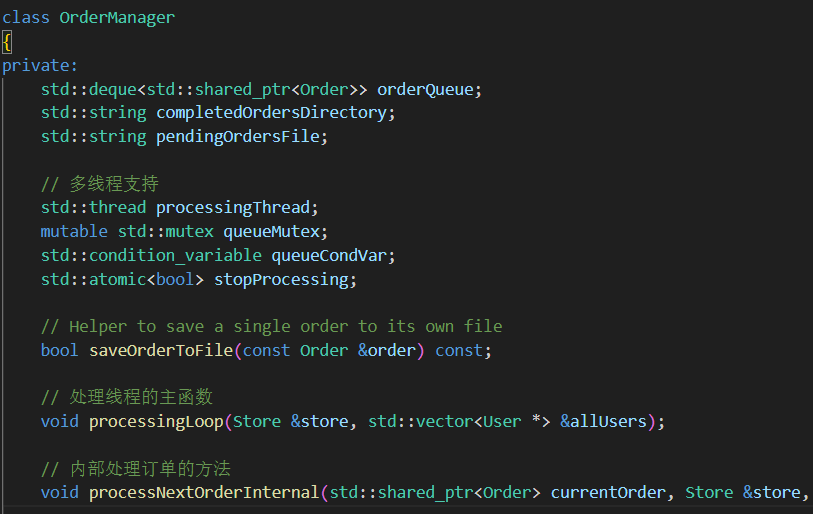
Order类类用于构造一个完整的订单结构，包含订单的基本信息、商品项列表以及订单的状态管理。类的私有成员包括订单编号（orderId）、下单用户（customerUsername）、订单商品项（items，为 OrderItem 的向量）、订单总金额（totalAmount）、处理状态（isProcessed）等订单必要属性。此外，还声明了一个静态私有方法 generateOrderId() 用于生成唯一订单号。



在公有部分，订单的商品项可以通过 addItemFromProduct（根据商品和数量添加）或 addItem（直接添加已有的订单项）方法加入，每次添加后都可以调用 calculateTotalAmount 重新计算总金额。同时，类还提供了丰富的 getter 和 setter 方法，方便外部访问和修改订单的各项属性，包括订单号、用户名、商品项、总金额、状态、时间戳等。

### 订单管理类Ordermanager

订单管理Ordermanager类是订单子系统的核心管理器，负责订单的排队、异步处理、归档、查询等功能，并且支持多线程安全。在设计的起初，它的目标是让订单的处理从程序的基础逻辑中独立出来，使用多线程模拟程序在后台运作而不影响前端的操作逻辑，为之后的客户端/服务端的分离做好铺垫。

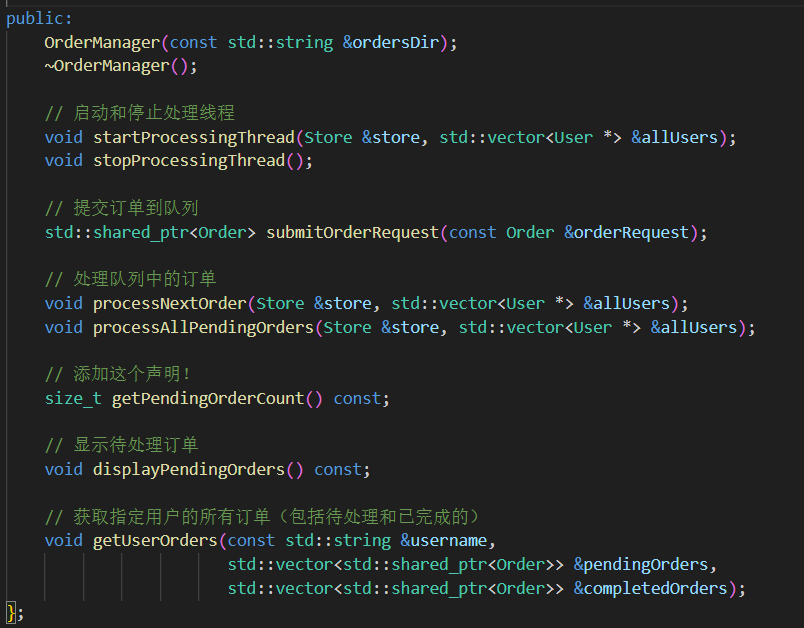


其主要属性详细说明如下：

* orderQueue：一个双端队列（std::deque<std::shared\_ptr<Order>>），用于缓存待处理订单请求。
* completedOrdersDirectory：已完成订单的保存路径（字符串）。
* pendingOrdersFile：待处理订单的缓存文件名（用于持久化保存）。
* processingThread：后台独立线程，用于异步处理订单。
* queueMutex：互斥锁，用于保护订单队列并发访问。
* queueCondVar：条件变量，用于处理线程在订单提交时被唤醒。
* stopProcessing：原子布尔值，指示后台处理线程是否应终止。
* Store &store：提供商品库存信息和销售逻辑。
* std::vector<User\*> &allUsers：提供所有用户的信息与余额校验。

OrderManager部分的核心功能包括订单接收、异步处理、状态管理与结果持久化。通过一个线程安全的队列结构（std::deque<std::shared\_ptr<Order>>），系统可将用户提交的订单加入等待队列中，并使用独立的后台线程对这些订单进行依次处理。线程处理逻辑中，系统会检查每一笔订单是否满足库存与用户余额的条件，只有在满足时才会更新库存、扣除金额，并将订单状态设为 "COMPLETED"；否则则根据具体失败原因设置状态（如 "CANCELLED\_INSUFFICIENT\_STOCK" 或 "CANCELLED\_INSUFFICIENT\_FUNDS"）。每笔订单处理后都会被保存到磁盘指定路径下，确保结果可追溯与持久化。

此外，OrderManager 支持手动模式下对订单进行逐条或批量处理，便于调试或在无后台线程的情况下运行。它还提供用户维度的订单查询接口，可同时返回用户的待处理订单与已完成订单。



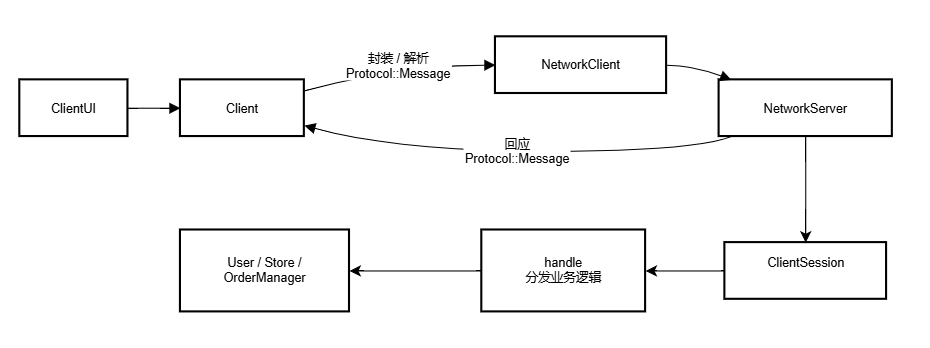
## 网络通信管理子系统network详细设计（接口协议说明）

网络通信管理子系统作为第三阶段的关键基础组件，承担客户端与服务器之间所有数据传输的任务。系统采用标准的 C/S 架构（Client/Server），底层通信基于 TCP 协议 实现，确保数据传输的可靠性、顺序性与完整性。

该子系统主要由以下模块组成：

* 协议定义模块（protocol.h/.cpp）：定义所有消息类型及其格式，其中包括协议自定义的协议接口
* 客户端通信管理模块（client.h/.cpp）：负责连接建立、消息发送与接收。
* 服务器通信与会话管理模块（server.h/.cpp）：负责监听、会话维护、消息分发与业务处理。

详细处理流程如下：



### 通信协议命名空间Protocol

通信协议模块（Protocol）是商城系统网络通信子系统的基础核心，承担客户端与服务器之间所有消息的定义、封装与序列化工作。该模块位于 network 层的最底层，主要职责是提供一个统一、稳定、可扩展的数据交换协议，使得不同子系统之间的交互能够以结构化、明确且高兼容性的方式进行。

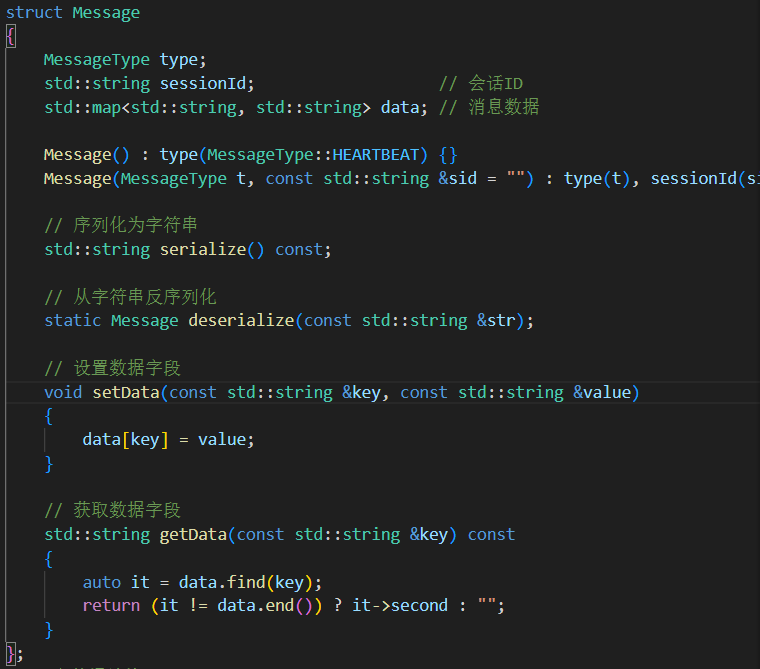


在架构上，Protocol 模块通过一组结构体与枚举，定义了从“用户行为 → 协议消息 → 数据结构 → 网络字节流”的完整转换链。其核心组件包括消息类型枚举（MessageType）、用户与业务状态枚举（如 UserType, OrderStatus）、统一的消息容器结构（Message），以及业务数据结构（如 UserData, ProductData, CartItemData, OrderData）等。这些结构全部支持双向序列化（serialize / deserialize），确保任意通信数据都可以在网络上传输并在对端准确还原。

#### 消息封装逻辑Message结构体

Protocol 的中心是 Message 结构体，它统一封装了所有从客户端到服务器（或反向）的消息内容，包括：

* 消息类型字段 type：标识业务意图，使用 MessageType 枚举值进行约束；
* 会话标识符 sessionId：用于多用户身份区分与会话追踪；
* 数据字段 data：键值对形式的业务内容，支持动态填充。



在其中本人自定义了序列化与反序列化机制，允许通过字符串形式传输完整消息，实现不依赖语言、系统的兼容通信。同时，Message 内部封装了设置与读取字段的便利函数，可以直接简便地调用解析。

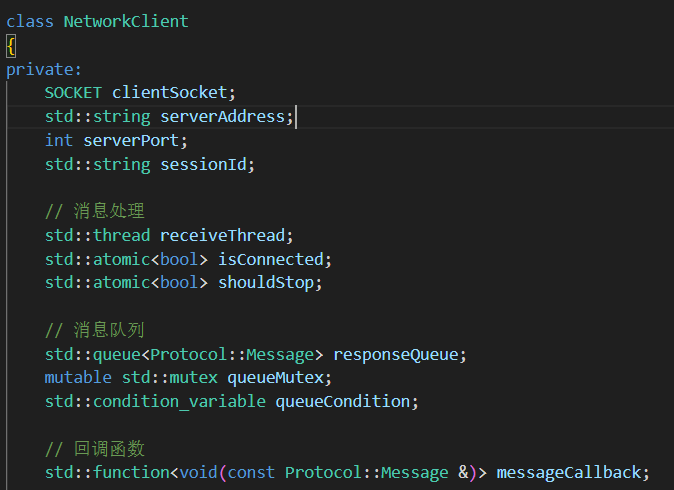
通信协议模块作为商城系统客户端与服务器之间信息流通的桥梁，承担着“统一语言”的职责。它通过结构化、类型化的方式标准化了所有业务交互内容，屏蔽了底层传输细节，为网络层与业务层之间提供了清晰、稳定、可扩展的数据协定。

### 客户端通信管理模块Client

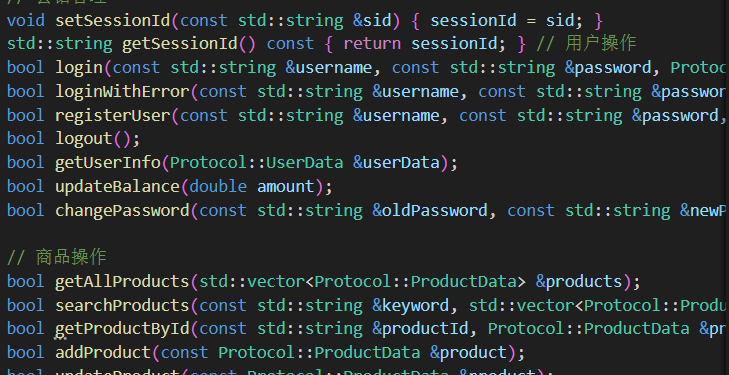
客户端通信管理模块Client中只创建了NetworkClient这一个类结构，涵盖客户端的底层操作需求。

#### 客户端

NetworkClient 是客户端应用中负责网络通信的核心类，封装了与服务端（NetworkServer）之间基于 TCP 套接字的连接管理、消息发送接收、协议封装解析及会话状态维护等功能。该类通过 Protocol::Message 结构与服务器进行标准化通信，支持阻塞与非阻塞两种响应处理方式，并可通过回调机制实现异步事件响应。



消息收发方面，类内部维护了一个消息队列（responseQueue），配合互斥锁和条件变量实现线程安全的消息缓存和同步。通过 sendMessage 可以向服务器发送消息，waitForResponse 和 getNextResponse 等方法则用于等待或获取服务器响应。还可以通过 setMessageCallback 设置消息到达时的回调函数，实现事件驱动的消息处理。



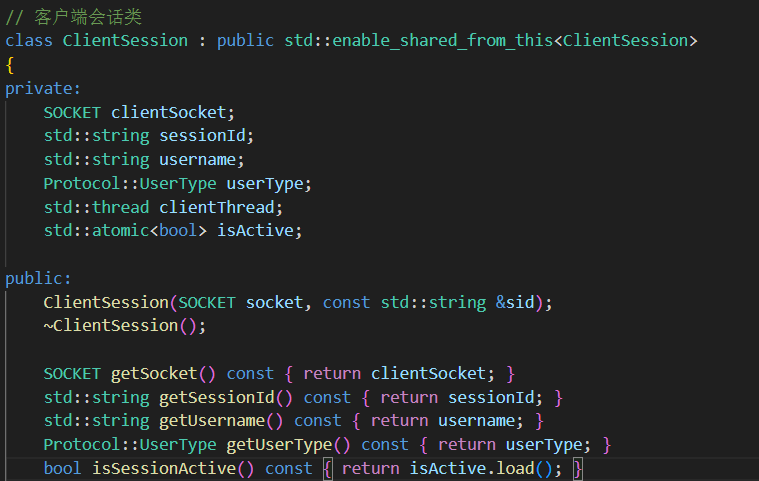
在业务操作层面，NetworkClient 封装了用户登录、注册、信息查询、余额和密码修改等用户相关操作；商品的查询、添加、更新、删除、库存管理等商品操作；购物车的增删改查；订单的创建、查询、状态更新等订单操作；以及库存锁定、商家商品管理等高级功能。这些方法都通过协议消息与服务器交互，简化了客户端业务开发。

### 服务端通信与会话管理模块Server

Server模块主要负责与客户端建立网络连接、接收请求消息、调度协议指令并与后端业务组件交互处理逻辑。其封装于 NetworkServer 主类中，结合多线程会话处理机制 ClientSession 实现多客户端并发通信支持。在实际通信中，服务端子系统通过统一的 Protocol::Message 消息协议完成前后端解耦通信，进而调用账户管理子系统和商城管理子系统对后端的数据文件进行读取和写入等操作。

#### 客户端会话类ClientSession

ClientSession类用于表示服务器端与单个客户端之间的会话连接。在服务器端，每当有一个客户端（比如你的 NetworkClient）连接到服务器，服务器就会为这个连接创建一个 ClientSession 对象。它负责维护这个连接的 socket、会话ID、用户名、用户类型、会话线程等，并负责与该客户端的数据收发和会话管理。

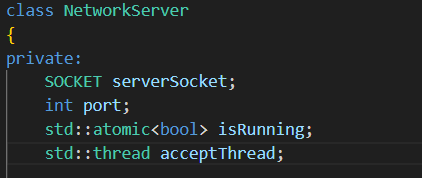


该类提供了基本的 getter 和 setter 方法，方便获取和设置会话相关属性，如用户名和用户类型。sendMessage 和 receiveRawData 方法分别用于向客户端发送消息和接收原始数据，实现了与客户端的数据通信。startSession 方法会启动会话线程，进入会话主循环sessionLoop，处理来自客户端的请求；stopSession 用于安全地关闭会话。

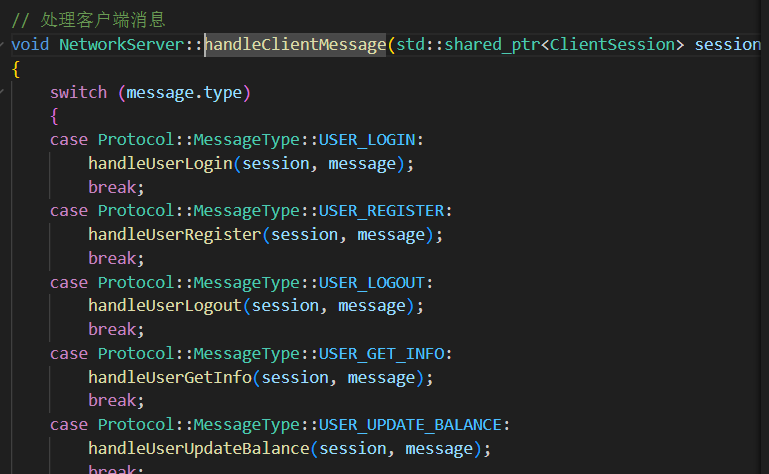
为了确保在多线程环境中的内存安全性，ClientSession 继承了 std::enable\_shared\_from\_this，使得类实例可以通过 shared\_ptr 被安全地传递与引用，防止在异步环境下出现悬挂指针或提前析构的问题。该设计保证了服务器端可以对每个连接保持稳定而可靠的生命周期管理。

#### 服务端主控类NetworkServer

NetworkServer 类作为整个服务器子系统的核心调度器，负责网络监听、客户端连接管理、协议解析与业务处理的统一协调。该类封装了 TCP 服务端的完整生命周期管理流程，从端口监听到多客户端连接接入，再到请求分发与响应发送，形成了完整的服务器端处理流程。



在客户端连接与会话管理方面，NetworkServer 维护了一个由 shared\_ptr<ClientSession> 构成的 sessions 容器，用于追踪当前所有活跃的客户端连接。为了确保多线程环境下访问的线程安全，访问该容器的所有操作均受 sessionsMutex 保护。每当有新的客户端连接建立时，服务器会生成唯一的 sessionId，创建对应的 ClientSession 对象，并由 acceptLoop 子线程持续监听与管理连接接入。辅助函数如 findSession、removeSession、findSessionByUsername 等，则用于会话的查找与维护。



在新的阶段中，通过之前子系统实现的层层递进，服务器内部集成了多个子系统的核心组件，包括 users、store和 orderManager。这些组件分别对用户信息、商品信息及订单生命周期进行集中管理，服务于服务器对不同业务请求的处理。配合 Protocol 模块定义的统一消息结构，NetworkServer 提供了一系列以 handle 命名的业务处理方法，覆盖了用户登录注册、商品查询与管理、购物车操作、订单创建与状态变更、库存锁定等典型场景，通过子系统之间的功能协作。

在网络通信细节方面，服务器通过 sendSuccessResponse、sendErrorResponse、sendDataResponse 等封装方法统一发送客户端响应，返回协议格式统一的数据结构，提升通信可靠性与代码复用性。同时，类还通过 convertToUserData、convertToProductData 等方法，将后端业务对象转换为 Protocol 协议层结构体，便于序列化与跨网络传输，支撑了系统数据在前后端之间的解耦传递。

## 数据库说明

商城系统的数据存储采用文件系统为主的持久化方案，所有用户、商品、购物车、订单等数据均以文本文件或目录结构进行管理，并主要采用类似CSV格式进行储存,存放在server\_data目录下。系统未使用关系型数据库（如MySQL、SQLite），但其数据文件结构和字段设计与关系数据库表类似，便于后续迁移或扩展。数据存储目录具体如下：

server\_data/

├── users.txt # 所有用户账户数据

├── carts/ # 每个用户的购物车文件

│ └── alice\_cart.txt

├── orders/ # 每个订单一个文件

│ └── ORD-20250601-203012-001.txt

├── store/

│ └── sellers/ # 每个商家一个商品列表

│ └── sellerA.txt

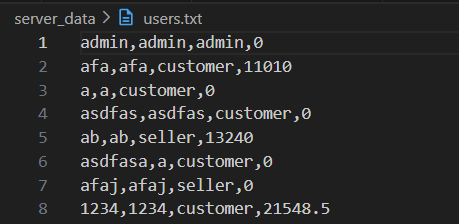
### 用户数据文件user.txt

用户数据是商城系统最基本的数据之一，用于记录所有注册用户的账户信息、登录凭据以及账户余额等基础资料。系统将所有用户信息统一存储在 server\_data/users.txt 文件中。该文件采用文本格式，每一行为一个用户的完整记录，字段之间使用英文逗号（,）进行分隔，便于解析和批量处理。

每行记录包含以下四个字段：

1. **username（string）**：用户的唯一标识符，系统要求所有用户名全局唯一，用于登录认证及用户身份关联。例如购物车、订单等数据均依赖用户名进行绑定。
2. **password（string）**：用户密码，目前以明文方式存储，仅作为原型系统设计，建议后续上线版本采用哈希加密（如SHA-256）以提高安全性。
3. **userType（string）**：标记用户的角色类型，系统共支持三类用户角色：普通顾客（customer）、商家（seller）和系统管理员（admin）。不同类型的用户在系统中拥有不同的操作权限。例如，顾客可浏览商品和下单，商家可发布商品，管理员拥有管理权限。
4. **balance（double）**：用户当前的账户余额，以浮点数形式存储，单位为人民币（元）。顾客下单扣费、商家收款入账等行为都会动态更新该字段。

该文件的设计简单清晰，适合进行快速读写和逻辑判断，也支持日后导出进行分析处理。用户信息文件的管理通常在用户注册、登录验证、修改密码、充值扣款等操作中被调用。



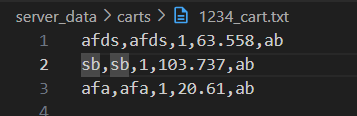
### 购物车数据文件 carts/<username>\_cart.txt

每个用户的购物车数据独立存储在 server\_data/carts/ 目录下，以 <用户名>\_cart.txt 命名，例如 alice\_cart.txt。该文件记录了用户当前购物车中的所有商品项，每行表示一个购物车条目，字段之间使用英文逗号分隔。

购物车记录包含以下字段：

1. **productId（string）**：商品ID，用于唯一标识该商品。通常该ID与商品名称保持一致，便于商家和顾客识别。
2. **productName（string）**：商品的显示名称，供用户界面展示使用。
3. **quantity（int）**：用户添加到购物车中的数量。
4. **priceAtAddition（double）**：用户将该商品加入购物车时的价格。由于商品价格可能在后续发生变动，系统需要保留添加时的价格，便于生成订单时对账。
5. **sellerUsername（string）**：该商品所属的商家用户名，用于订单生成和收入归属。

该设计允许每个用户独立管理自己的购物车，支持快速的读取和修改。文件结构、格式统一，便于在登录后加载并在下单时转化为订单数据。



### 订单数据文件（orders/ORD-xxxx.txt）

订单数据采用“一单一文件”的方式存储，每个订单保存在 server\_data/orders/ 目录下，文件名以订单编号命名，格式为 ORD-YYYYMMDD-HHMMSS-序号.txt，例如 ORD-20250601-203012-001.txt。

每个订单文件由两部分组成：订单头部 和 订单项列表。前几行为头部信息，以“键:值”形式存储；后续行为该订单中每个商品项的明细，字段以逗号分隔

 **OrderID（string）**：唯一订单编号，系统根据时间戳自动生成。

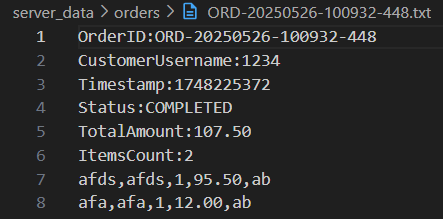
 **CustomerUsername（string）**：发起订单的顾客用户名。

 **Timestamp（time\_t）**：下单时间的 UNIX 时间戳，精确到秒。

 **Status（string）**：订单状态，如 PENDING（待支付）、COMPLETED（已完成）、CANCELLED（已取消）等。

 **TotalAmount（double）**：订单总金额（所有商品项价格总和）。

 **ItemsCount（int）**：订单中包含的商品数量（条目数）。



该设计使得订单文件既具备可读性，也易于程序按行解析，同时支持对历史订单进行归档或审计处理。

### 商品数据文件（store/sellers/商家名.txt）

每位商家的商品信息存储在 server\_data/store/sellers/ 目录下，对应的文件名为 <商家名>.txt，例如 sellerA.txt。该文件中每行记录一个商品的完整信息，字段间用英文逗号分隔。

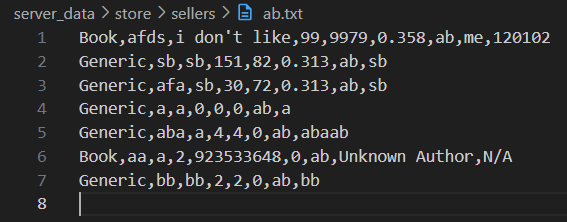
通用字段：

1. **type（string）**：商品类别，如 Book、Clothing、Food、Generic 等。
2. **name（string）**：商品名称。
3. **description（string）**：商品描述。
4. **originalPrice（double）**：商品原始价格。
5. **quantity（int）**：商品库存数量。
6. **discountRate（double）**：当前折扣比例（范围0~1，1表示无折扣）。
7. **sellerUsername（string）**：商品所属商家。

类型特有片段：

* **Book**：author（作者），isbn（书号）
* **Clothing**：size（尺码），color（颜色）
* **Food**：expirationDate（保质期）
* **Generic**：categoryTag（分类标签）

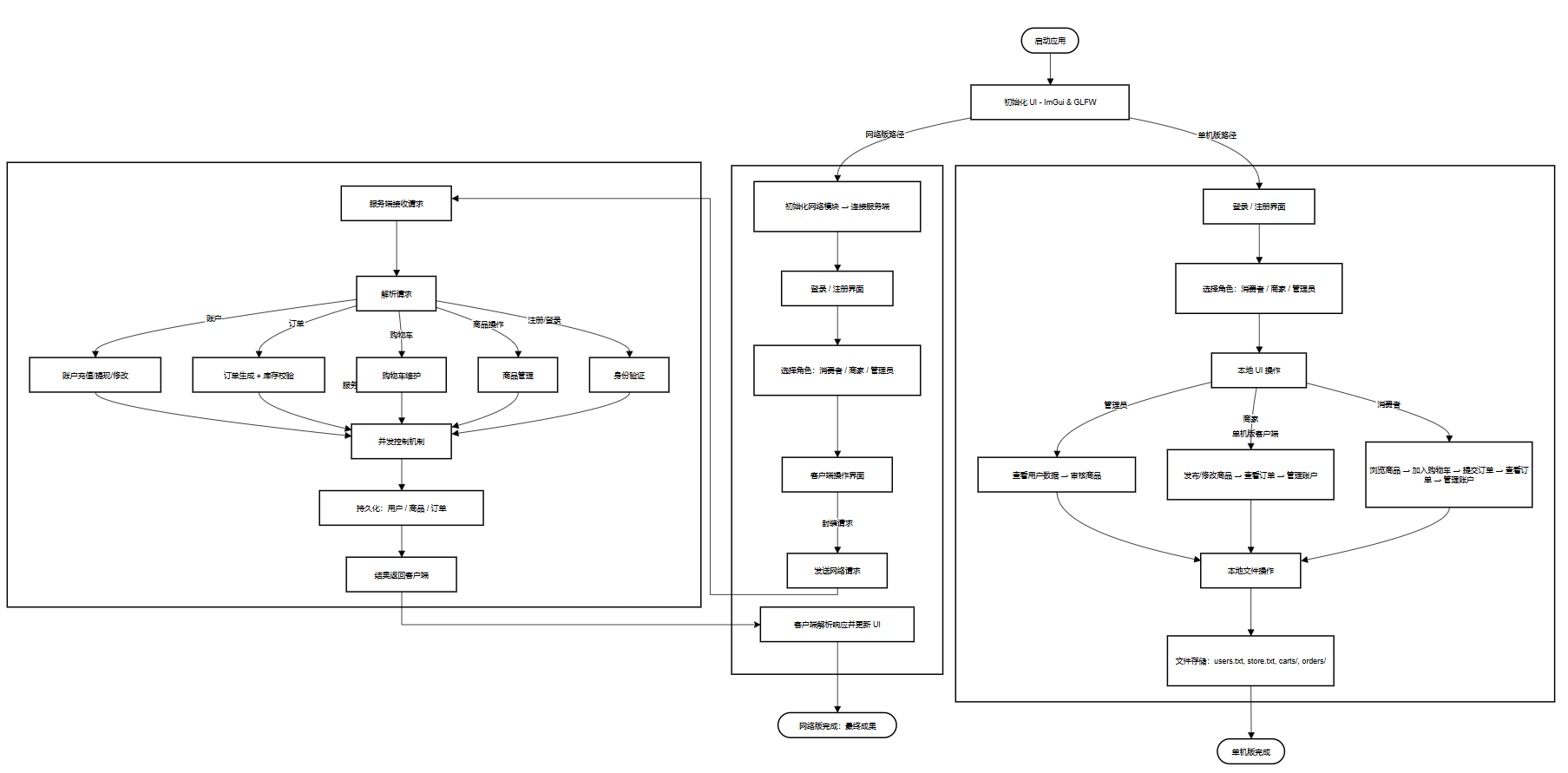
示例内容如下：



综上所述，本商城系统采用了基于文本文件的持久化存储方案，以简洁的目录结构和统一的字段格式实现了用户、商品、购物车及订单等核心数据的管理。尽管系统未使用关系数据库，但仍具有良好的可读性和可扩展性，便于实现系统的进一步升级与规模化部署。

# 实现与测试

## 项目大致流程



### 单机版大致流程

单机版是基于 C++ 实现的本地简易电商平台，采用 ImGui+GLFW 构建图形界面，支持用户通过图形交互完成完整的电商操作流程。系统由 UI 客户端与本地文件系统组成，所有服务端逻辑通过本地数据读写完成，无需实际网络通信。



程序启动后，用户首先进入登录/注册界面。注册时可选择角色类型，包括消费者、商家与管理员。登录成功后，系统根据用户类型切换到对应的主界面，展示特定功能入口。消费者可浏览商品、添加至购物车、提交订单、查看历史订单并进行账户管理操作；商家则可发布商品、修改或下架商品，查看销售订单及账户余额。



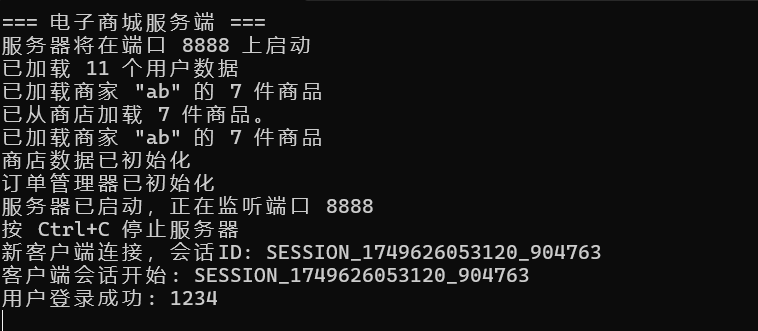
客户端在用户操作时，调用本地逻辑模块处理数据操作，例如注册/登录通过读取或写入 users.txt 实现身份校验与数据保存，购物车数据以用户独立文件形式保存在 carts/ 目录下，商品信息则统一存储于 store.txt。订单在生成时写入 orders/ 目录，同时自动扣除账户余额，并更新用户数据文件。商品上架与修改同步修改商品文件，账户充值与提现操作也会直接更新用户余额信息。

系统的“服务端”操作本质上通过本地文件 I/O 完成，对用户无感知。所有功能的后端逻辑由客户端本地统一管理，操作完成后立即进行数据持久化，保证数据一致性与恢复性。

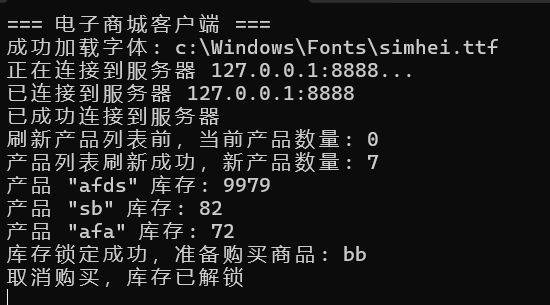
### 网络版大致流程

网络版使用了稍做优化的相同UI工具，但是不同的是，这里将本地文件系统分离成服务端，并将UI与文件系统的直接联系转换成了网络通信，以下简要展示两者的运行流程和主要功能。

客户端启动后，首先初始化网络模块并尝试与服务端建立连接。之后，客户端渲染UI，用户可以在单机版上完成的操作同样可以在网络版上完成。除此之外，客户端的主要职责是收集用户操作行为，封装为网络请求，并通过统一的协议发送至服务端，再将服务端返回的数据进行解析和展示，实现前端与后端的交互。



服务端在之前的单机版上拓展了一些功能：服务端在启动后监听固定端口，持续接受客户端连接。其核心任务是解析来自客户端的数据请求，根据不同类型执行相应的处理逻辑。服务端功能包括用户账户管理（如注册信息验证、密码校验）、商品信息的增删改查、购物车状态维护、订单生成与库存校验等。每项操作均需保证多用户并发访问下的数据一致性，服务端通过并发控制机制确保操作的可靠性。此外，服务端还实现了基本的异常处理与退出保护机制，支持将用户与商品数据持久化存储，从而保障系统运行的稳定性与容错能力。



在整体流程中，客户端注重用户操作的便利性与交互体验，服务端则作为后端，承担所有业务逻辑与数据处理任务。客户端所有功能操作，如商品搜索、下单、用户修改信息等，都会通过网络协议转换为请求消息发送给服务端，由服务端完成数据处理、结果反馈，再由客户端解析结果并更新界面。使用前后端职责分离与通信机制，才实现了多用户电子商城网络版系统。

## 项目基本实现

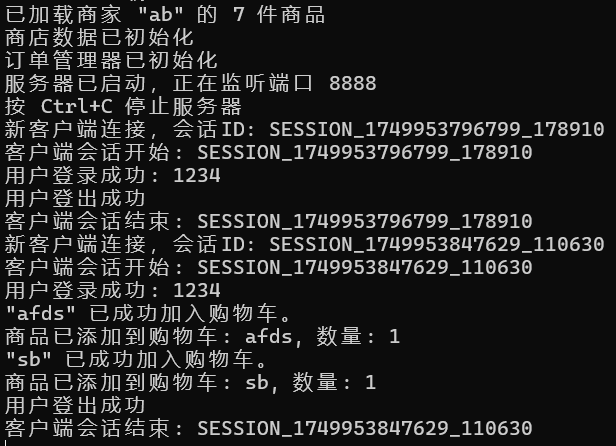
整体项目构建了一个基础的电子商务交易平台，采用包括普通单机架构、客户端-服务端架构的实现，整体功能覆盖用户管理、商品管理、交易流程等多个模块，基本满足实验的主要目标。

客户端支持用户注册、登录、信息修改等账户管理操作，用户登录后可修改余额、浏览商品、搜索筛选、添加购物车、下单支付、查询订单等常用电商功能。系统支持卖家身份登录，提供商品上架、信息维护、销售订单查看等功能，并实现了商家和顾客支付过程余额增减的逻辑。其中，客户端界面不仅支持命令行，还增加了图形界面的模式。

其中，购物车页面和订单页面的详细设计展示如下：

服务端负责处理所有核心业务逻辑，包括用户验证、商品数据维护、订单生成与存储等。系统支持多用户并发访问，通过本地文件系统进行数据持久化，管理结构清晰，数据组织合理。服务端实现了统一的请求解析与合法性校验机制，确保数据一致性与操作安全，同时支持断开连接时的自动关闭，保障数据完整。



项目整体采用面向对象设计方法，对各个模块系统进行了层次分离，并在用户管理、商品处理、订单生成等关键功能中实现了类的封装与职责划分，为后续代码中的维护和扩展奠定了基础。

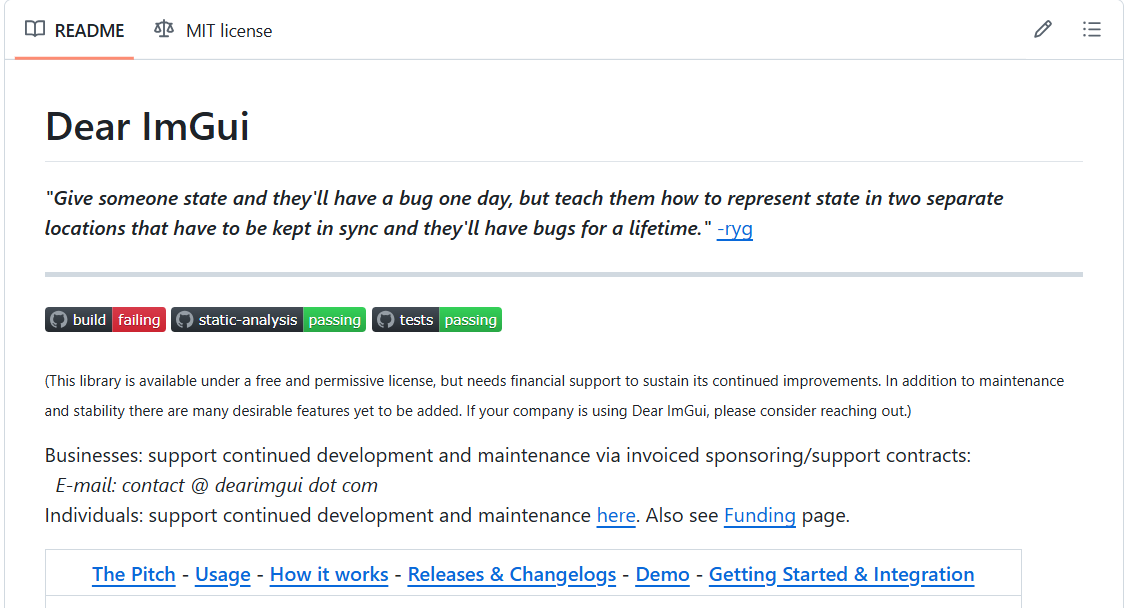
## 实现过程中遇到的主要问题和解决方案

### Ui选择

在本项目的图形化界面开发初期，原计划采用较为成熟的 Qt 框架来构建 UI 界面。Qt 拥有丰富的控件库与可视化设计器，适合快速开发功能完善、美观统一的桌面应用界面。然而在实际开发过程中，受到兼容性限制，Qt 的使用受到严重影响。

由于开发环境主要基于 Windows，部分系统存在与 WPF（Windows Presentation Foundation）等组件的底层冲突，导致在尝试使用QT对原代码逻辑进行修改的过程中出现诸多问题。例如，Qt 的构建过程依赖较多外部工具链，部分版本要求使用特定的编译器（如 MSVC），而原项目使用的是 MinGW，贸然更改将产生兼容性问题。

为规避这些问题，我最终选择了 ImGui 与 GLFW 的组合来实现图形化界面。ImGui 是一款轻量级、跨平台的即时模式 GUI 库，配合 GLFW 可快速搭建自定义界面，且与WPF架构的代码结合较好，使用少量代码即可实现完整交互逻辑，便于嵌入原有 C++ 项目；同时，其支持即时渲染与调试，有利于我当前所需的快速迭代开发。



在实现上，项目将原命令行交互逻辑（如用户输入、菜单选择等）迁移至 ImGui 控件中。登录界面通过输入框与按钮实现用户认证，主界面则利用标签页、表格等控件展示不同用户类型所对应的功能。业务逻辑继续沿用已有的 C++ 类结构，仅改变了输入输出方式，实现了与图形界面的自然融合。

通过这一替代方案，项目有效规避了 Qt 带来的环境兼容性问题，同时提升了用户交互体验，为后续功能拓展提供了更灵活的平台支持。

### 网络通信协议设计

网络版直接使用 Socket 进行收发操作时，容易出现消息边界不明确的问题，典型现象包括粘包与分包，导致接收端无法准确判断每条消息的起止位置。此外，若未统一消息格式，不同功能模块之间的数据解析也容易出错，进而影响系统稳定性与扩展性。

本系统采用自定义消息协议，设计了独特的消息结构，每条消息包含消息类型、会话ID等必要字段，确保通信双方能够准确识别每条消息的起止和内容：

* 在 protocol.h 和 protocol.cpp 中定义了 Protocol::Message 结构体，包含消息类型（MessageType）、会话ID、数据字段（map）。
* 消息通过 serialize() 方法转为字符串，格式为：类型|会话ID|字段数|key1=value1|key2=value2|...，并在网络传输时发送。
* 接收方用 deserialize() 方法解析字符串，恢复为结构体，确保消息边界和内容都能被准确识别。

通过这种方式，保证了数据传输的完整性和解析的高效性，同时为后续功能扩展提供了良好的基础。

# 实验经验与个人总结

## 拓展想法

### 通信协议改进

在项目开发初期，出于对 C++ 网络编程不熟悉的考虑，自行设计了一套简易的通信协议。虽然这种方式能灵活地满足当前的功能需求，但实际开发过程中也暴露出不少问题。例如，必须手动处理消息边界、粘包/分包、数据的序列化与反序列化，以及协议的兼容性扩展等，每当新增一个功能，都需要同步修改协议结构和解析逻辑，使开发和维护工作量显著增加。此外，由于协议自定义，缺乏现成的调试工具，调试过程也相对繁琐。

在进一步学习中了解到，目前主流系统广泛采用标准化通信协议（如 HTTP/JSON、WebSocket、gRPC 等）进行客户端与服务端的数据交换。这些协议在实际开发中具有诸多优势：

* 开发效率高：拥有成熟的库和框架支持，客户端与服务端可以快速集成，避免了重复造轮子的负担；
* 跨平台支持好：标准协议被多种语言和平台广泛支持，便于未来拓展 Web、移动端等多终端系统；
* 调试和测试友好：可借助 curl、Postman 等工具快速进行接口测试，配合抓包工具更容易定位问题；
* 便于维护和升级：协议结构规范清晰，文档丰富，利于团队协作与系统长期演进。



因此，如果后续对项目进行功能升级或系统重构，我被推荐采用如 HTTP+JSON 或 gRPC 等标准通信协议。这不仅能显著减少底层通信逻辑的实现与维护成本，也更有利于系统向现代化架构（如微服务、多端同步、API 对接等）演进。对于 C++ 开发者，目前也已有如 cpp-httplib、Pistache、grpc-cpp 等高质量的开源库可供选择，能够很好地支持实际应用开发。

### 数据库管理改进

在本项目的当前实现中，数据持久化依赖于文本文件的读写操作。用户信息、商品清单、订单记录等均以结构化文本形式保存在本地文件系统中。这种方式简单直观，开发门槛较低，便于快速搭建原型，尤其适合教学或小规模的实验项目。然而，随着系统复杂度的提高，尤其是当用户数量、商品数量、并发访问频率不断增长时，基于文件的存储机制逐渐暴露出明显的局限性。

首先是查询效率问题。文本文件通常需要全量读取或线性遍历才能完成一次查询，面对复杂的筛选、排序或联合查询需求时，性能瓶颈尤为明显。其次是数据一致性与并发控制的缺失。在多用户同时读写同一文件时，若无良好的锁机制，很容易引发数据冲突或写入错误。此外，文件结构难以灵活调整，不利于后期的功能拓展与维护。

引入数据库系统（如 MySQL、SQLite、MongoDB 等）可作为下一阶段系统优化的重要方向。关系型数据库提供了成熟的数据建模能力和高效的 SQL 查询接口，能够大幅提升数据访问效率。同时，数据库的事务机制可有效保证在并发场景下的数据一致性，防止因中断或异常操作造成的数据错乱。安全性方面，数据库具备完整的用户权限管理系统，可控制不同角色对数据的访问范围，提升系统整体的安全等级。



## 个人经验与教训总结

本次电子商城系统的开发，是一次较为完整的C++工程实践。在这个过程中，我不仅深入接触了网络编程和Socket通信的底层机制，还通过设计和实现自定义通信协议、管理多线程服务端逻辑，全面理解了一个简易客户端-服务端架构系统的运行方式。同时，在图形界面的构建中，我也从最初尝试使用Qt，转而选择ImGui+GLFW的过程中，体会到开发环境兼容性与技术选型之间的权衡。项目开发中大量面向对象的设计实践，让我深刻体会到良好模块划分对后期维护与扩展的帮助，例如用户、商品、订单、网络通信等模块的相互独立和分层设计，使整体结构更清晰、职责更明确。

然而，在实际开发过程中，也暴露出不少问题。例如，为了快速实现通信功能，初期自定义了Socket通信协议，但这带来了粘包分包、消息边界、序列化兼容等复杂问题，导致协议扩展和调试工作量陡增。相比之下，标准协议如HTTP/JSON或gRPC无论在开发效率、跨平台支持还是调试工具上都更具优势，未来更应优先考虑。此外，目前项目仍采用文本文件作为数据存储手段，虽然实现简单，适合小规模系统，但在数据一致性、并发访问以及查询性能方面都存在明显短板。后续若考虑将系统长期运行或扩展使用场景，引入数据库系统将更为稳妥。

另一个重要反思教训是项目结构设计的不足。由于开发初期缺乏完整的系统规划，代码组织和模块划分显得杂乱，部分逻辑模块之间耦合度较高，职责不清，影响了后期功能扩展和调试效率。这一教训凸显了在项目伊始就进行合理架构设计的重要性，未来应更加重视先期任务理解、初步模块分化、功能详细规划的准备，明确具体做什么、如何做再下手的先后顺序。

综上所述，这次项目经历不仅让我收获了工程实践能力，也深刻体会到“设计先行、结构清晰、标准优先、健壮为本”的重要性。每一个问题背后都映射出一个工程环节的短板，只有持续总结、反思和优化，才能在今后的开发中构建出更高质量、更具扩展性、更符合实际需求的系统架构。