北京郵電大學

面向对象课程设计 C++

<电商交易平台>

兰学超 2019211564

目录

一、实验内容

本次实验要求设计一个电商交易平台，能实现买家和卖家登陆后进行一系列的操作，包括用户管理、商品管理、交易管理等功能。分为单机版和网络版。

网络版要求在单机版的基础上用 C/S 结构实现，并以 TCP 协议进行客户端与服务器的通信。

二、实验环境

macOS，XCode，C++语言

三、设计思路

首先由于每个用户需要进行登陆及相关账户管理，因此需要一个账户模块存储账户信息。将账户相关操作置于账户模块内，包括密码更改、余额管理等。

用户可分为两种：买家或卖家，二者有部分共同的属性及操作，因此设计一个继承体系，由基类 User 派生出 Consumer 和 Merchant 类，同时将部分类似功能函数设计为虚函数，以实现多态，便于泛型编程。User 类中有所有用户共有的操作函数，如商品查询等。对于不同种类用户所拥有的不同种功能，则分别置于其对应的模块中。例如 Merchant 模块中有商品信息更改等函数，而购物车管理等函数置于 Consumer 模块中。

同理商品可分为三类：书本、食品和衣服。因此设计一个继承体系，由基类 Goods 类派生出 Book、Food、Clothing 类，并设计虚函数以实现多态。由于三种商品没有各自独有的操作和属性，故商品属性（名称价格等）以及相关操作（更改名称价格等）都置于 Goods 模块内。

单机版最后需要一个总体性的模块，进行与用户的交互，获取输入、输出结果，将各模块统筹调用，即系统模块。将所有账户信息、商品信息存于该模块中，同时执行所有功能的调度。

网络版需要同时运行两个进程分别作为客户端和服务器，因此设计通过命令行参数以不同方式执行相同程序。当输入-c 时运行客户端，-s 运行服务器。故程序需要两个模块：client 和 server，作为系统的总体调度以及与用户的交互。

程序采用 C/S 结构，是一种胖客户端的结构，因此我将所有的输入输出部分于 client 模块中实现，而服务器仅仅作为一个类似数据库的存在，接收客户端发来的信息后，分析客户端请求执行的操作，然后发回结果。

客户端与服务器的通讯采用 TCP 协议的 socket，是一种面向连接的方法，事先建立连接，之后传送信息快捷且无错，最后需要断开连接。

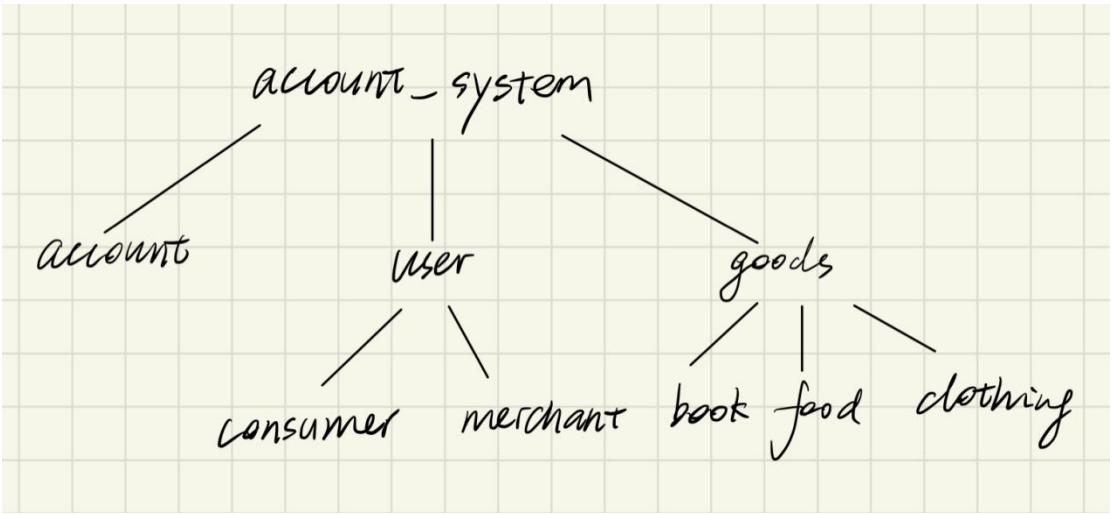
客户端向服务器发送的信息是一个字符串，开头为操作对应的编号，然后是该操作需要的相关信息。两段信息以空格分割，接收时通过 stringstream 流分割字符串获取信息。服务器返回给客户端的信息根据不同操作分为两种，一种是确认信息（如登陆成功发回 1，失败发回 0），另一种是输出信息（如客户端请求输出所有商品信息，服务器发回的字符串即为全部商品信息，之后由客户端格式化输出）。

四、模块设计

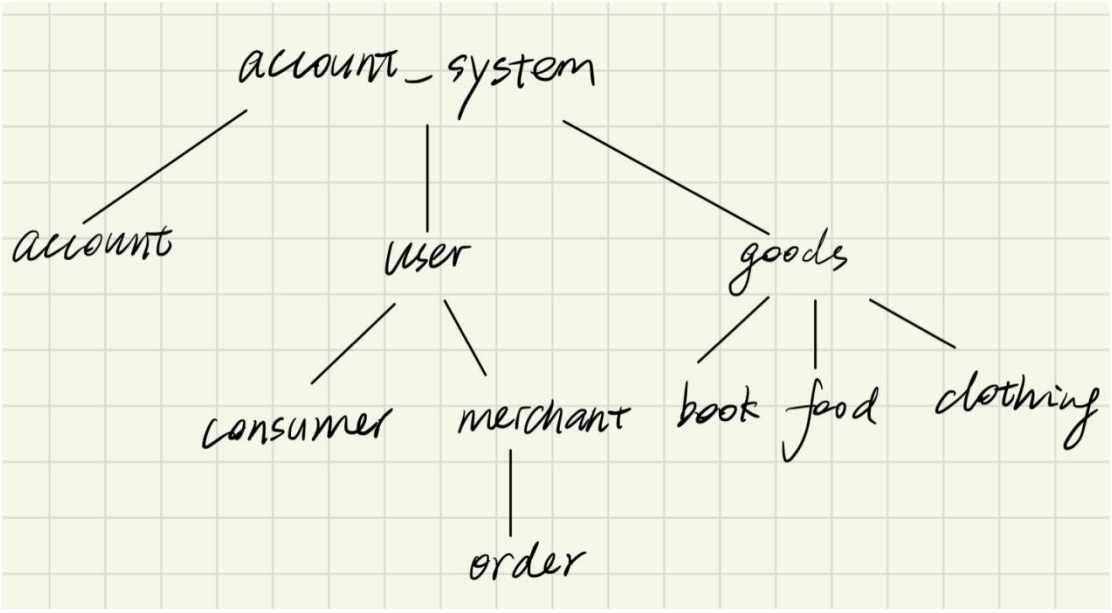
题目一主要模块包括 account_system 模块、account 模块、user 模块、consumer 模

块、merchant 模块、goods 模块、book 模块、food 模块、clothing 模块。

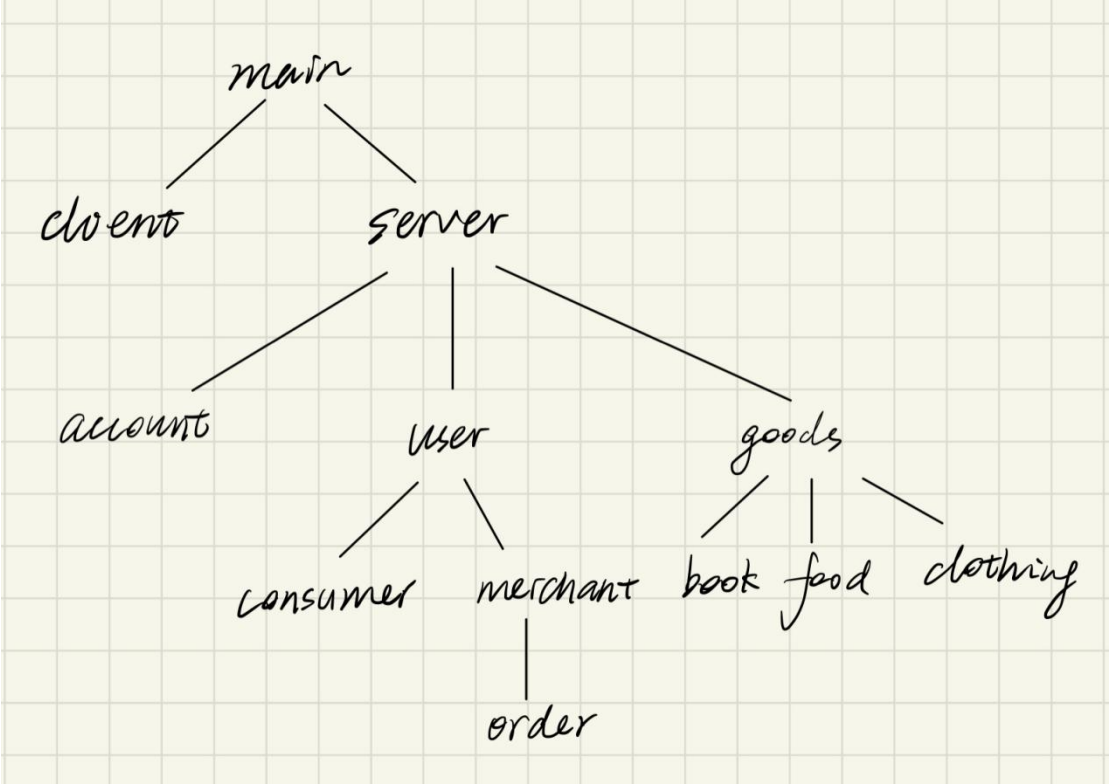
其中包含两个继承体系，user 模块派生出 consumer 和 merchant 模块，goods 模块派生出 book、food、clothing 模块。



题目二在题目一的基础上新增 order 模块。



题目三在题目二的基础上，删去 account_system 模块，新增 client、server 模块。



五、详细设计

(一) account_system 模块

该模块仅在单机版（题目一、题目二）中使用，作为操作主模块，用于串联其他所有模块。

1、数据结构：

（1）map<string, Account*> id2account：以账户 id 作为 key 值，对应的账户 account 作为 value 值，用于存储用户 id 到用户账户的映射，便于查询。

（2）vector<Account*> accounts：指针数组用于存储所有账户的指针。

（3）map<string, Merchant*> merchants：以账户 id 作为 key 值，对应的卖家用户 Merchant 的指针作为 value 值，用于存储用户 id 到用户的映射，便于查询。

（4）map<string, Consumer*> consumers：以账户 id 作为 key 值，对应的卖家用户 Consumer 的指针作为 value 值，用于存储用户 id 到用户的映射，便于查询。

（5）vector<Goods*> goods：指针数组用于存储所有商品的指针。

2、函数设计：

（1）void readFile( )：读取文件中存储的账号、商品信息。

（2）void writeFile( )：把账号、商品信息存入文件。

（3）void mainMenu( )：一级界面函数，提示用户选择账号注册、登陆、退出的功能。

（4）void merchantOperate( )：二级界面函数，当类型为卖家的用户登陆后会转到该界面，提示用户选择账户管理、商品管理等相关操作。

（5）void consumerOperate( )：二级界面函数，当类型为买家的用户登陆后会转到该界面，提示用户选择账户管理、购买商品等相关操作。

（6）void showGoods( )：格式化输出全部商品及相关信息。
（7）void showAccounts( )：格式化输出全部账户及相关信息。
（8）void signin( )：用户注册。
（9）void signin(const string &id)：signin 的重载函数，无需用户第二次输入注册 id。
（10）void login( )：用户登录。

(二) server 模块

该模块仅在网络版（题目三）中使用，作为服务器的主要操作模块。

1、数据结构：

（1）int listen_socket: 监听 socket，用于监听客户端发来的连接请求。
（2）int connect_socket: 连接 socket，用于和客户端进行通信。
（3）struct sockaddr_in server_addr: 服务器的地址，包含 IP 地址、端口号等。
（4）map<string, Account*> id2account: 以账户 id 作为 key 值, 对应的账户 account 作为 value 值，用于存储用户 id 到用户账户的映射，便于查询。
（5）vector<Account*> accounts: 指针数组用于存储所有账户的指针。
（6）map<string, Merchant*> merchants: 以账户 id 作为 key 值，对应的卖家用户 Merchant 的指针作为 value 值，用于存储用户 id 到用户的映射，便于查询。
（7）map<string, Consumer*> consumers: 以账户 id 作为 key 值，对应的卖家用户 Consumer 的指针作为 value 值，用于存储用户 id 到用户的映射，便于查询。
（8）vector<Goods*> goods: 指针数组用于存储所有商品的指针。
（9）Account *online_account: 存储当前在线的账户。

2、函数设计：

（1）void readFile( ): 读取文件中存储的账号、商品信息。
（2）void writeFile( ): 把账号、商品信息存入文件。
（3）void initSocket( ): 创建 listen_socket，绑定地址，监听连接请求。
（4）void waitClient( ): 创建 connect_socket，等待客户端的连接请求，建立连接。
（5）void handleClient( ): 处理客户端发来的信息，根据信息执行不同的操作，并将操作结果发回给客户端。
（6）void start( ): 依次执行上述函数，最后关闭连接。

(三) client 模块

该模块仅在网络版（题目三）中使用，作为客户端的主要操作模块。

1、数据结构：

（1）int connect_socket: 与服务器的连接 socket。
（2）string type: 当前在线用户的类型。
（3）string id: 当前在线用户的账号 id。

2、函数设计：

（1）void init( ): 创建 connect_socket，向服务器发送连接请求，建立连接。

（2）void menu( ): 所有操作的菜单界面。

（3）void find( ): 商品查询，根据用户选择不同的商品查询方式，获取用户输入，向服务器发送相关信息，之后接收服务发回的信息，并执行相关输出。

（4）void start( ): 主操作函数，首先 init( )，然后根据用户输入选择不同的操作，包括商品查询、商品管理、账户管理等，获取用户输入，向服务器发送相关信息，之后接收服务发回的信息，并执行相关输出。

（四）account 模块

存储用户账户信息，以及关于账户的相关操作。

1、数据结构：

（1）string id: 用户账号 id。

（2）string password: 用户账户密码。

（3）double balance: 账户余额。

（4）string type: 账户类型（merchant/consumer）

2、函数设计：

（1）void recharge(double money): 向账户内充值。

（2）void consume(double money): 用账户余额消费。

（3）void show_account( ): 格式化输出该账户相关信息。

（4）void transIn(double money): 向账户中转入一定金额（在支付订单时调用，订单金额从买家账户转出，转入卖家账户）。

（5）void transOut(double money): 从账户中转出一定金额（在支付订单时调用，订单金额从买家账户转出，转入卖家账户）。

（五）user 模块

存储用户信息的基类模块，以及与用户相关的操作。

1、数据结构：

（1）Account* account: 每个用户对应一个账户。

2、函数设计：

（1）virtual string getUserType( ) = 0: 基类中的纯虚函数，用于不同的子类重载返回用户类型。

（2）void changePassword( ): 更改密码。

（3）void getBalance( ): 获取用户账户余额。

（4）void recharge( ): 充值（内部调用 account 的 recharge 函数）

（5）void purchase( ): 购买选定的商品。

（6）void findGood(vector<Goods*> &goods): 商品查询界面，提示用户选择用不同的筛选方式获取商品。

（7）void findGoodName(vector<Goods*> &goods): 通过商品名查询商品。

（8）void findGoodPriceUp(vector<Goods*> &goods)：筛选高于给定价格的所有商

品。

（9）void findGoodPriceDown(vector<Goods*> &goods)：筛选低于给定价格的所有商品。

（10）void findGoodRemainUp(vector<Goods*> &goods)：筛选高于给定余量的所有商品。

（11）void findGoodRemainDown(vector<Goods*> &goods)：筛选低于给定余量的所有商品。

（12）void findGoodType(vector<Goods*> &goods)：通过商品种类查询商品。

（六）merchant 模块

User 类的派生类，存储卖家信息，以及卖家的相关操作。

1、数据结构

由于是 User 类的派生类，故无相关数据结构。

2、函数设计

（1）string getUserType(){return "merchant";}：内联函数，重载 User 中的该函数，返回类型为“merchant”。

（2）void menu()：显示菜单（操作界面）。

（3）void set_name(vector<Goods*> &goods)：更改指定商品的名称。

（4）void set_price(vector<Goods*> &goods)：更改指定商品的价格。

（5）void set_remaining(vector<Goods*> &goods)：更改指定商品的余量。

（6）void set_description(vector<Goods*> &goods)：更改指定商品的描述。

（7）void add_goods(vector<Goods*> &goods)：添加商品。

（8）void onsale(vector<Goods*> &goods)：对同类商品打折（或加价）。

（七）consumer 模块

User 类的派生类，存储买家信息，以及买家的相关操作。

1、数据结构

（1）map<int, int> cart: 购物车，商品编号作为 key，商品数量作为 value，需要映射到 goods 数组中获取商品具体信息。

（2）map<int, int> selected: 已选中的商品，商品编号作为 key，商品数量作为 value，需要映射到 goods 数组中获取商品具体信息。

（3）double total_price: 已选中商品的总价格。

（4）vector<Order> orders: 存储所有未取消的订单（待支付/已支付）

2、函数设计

（1）string getUserType(){return "consumer";}：内联函数，重载 User 中的该函数，返回类型为“consumer”。

（2）void menu(): 显示菜单（操作界面）。

（3）void cartManage(vector<Goods*> &goods): 购物车管理函数，根据用户输入选

择执行不同的购物车管理操作函数。

（4）void cartMenu()：显示购物车管理的菜单（操作界面）。

（5）void showCart(const vector<Goods*> &goods): 输出购物车中所有商品及信息。

（6）void addCart(const vector<Goods*> &goods): 向购物车中添加商品。

（7）void rmCart(const vector<Goods*> &goods): 从购物车中移除商品。

（8）void changeCart(const vector<Goods*> &goods): 更改购物车中商品数量。

（9）void select(const vector<Goods*> &goods): 选中购物车中的商品。

（10）void unselect(const vector<Goods*> &goods): 取消选择指定的已选中的商品。

（11）void generateOrder(vector<Goods*> &goods): 将已选中的商品生成订单。

（12）void payOrder(vector<Goods*> &goods): 支付指定的订单。

（13）void cancelOrder(vector<Goods*> &goods): 取消指定的订单。

（14）void showOrders(vector<Goods*> &goods): 输出所有订单及相关信息。

（八）order 模块

存储订单相关信息，以及订单的相关操作。

1、数据结构：

（1）map<int, int> good_list: 存储订单中所有商品，key 值为商品编号，value 为商品数量，需要映射到 goods 数组中获取商品具体信息。

（2）double total_price: 订单总金额。

（3）int state: 订单状态（0 待支付/1 已支付）

2、函数设计：

（1）void release(vector<Goods*> &goods): 释放该订单中冻结的商品余量。

（2）void showOrder(vector<Goods*> &goods): 输出该订单所有相关信息。

（3）void transfer(vector<Goods*> &goods): 把用户支付的金额转入订单中每个商品对应商家的账户中。

（九）goods 模块

存储商品信息的基类，以及商品的相关操作。

1、数据结构：

（1）string name: 商品名称。

（2）string description: 商品描述。

（3）double price: 商品单价。

（4）int remaining: 商品余量。

（5）string type: 商品类型。

（6）Merchant* merchant: 商品所属商家。

2、函数设计：

（1）void show_good( ): 输出该商品相关信息。

（2）virtual ~Goods( ) = 0: Goods 类是 Book、Food、Clothing 的基类，故其析构函

数作为纯虚函数。

（十）Food、book、clothing 模块

作为 Goods 类的派生类，无特别数据结构与函数，故略过。

五、遇到问题及解决方案

1、头文件多次引用

解决：在每个头文件前后加上#ifndef、#define、#endif，使每个头文件仅链接一份，不会产生符号重复定义的错误

2、符号为定义错误

解决：经搜索后检查代码，发现是因为部分函数声明后却没有定义导致。删去函数声明即可。

3、栈溢出错误

解决：经搜索后查明是由于数组越界产生的问题。查看代码发现我使用 for each 循环遍历数组使删去了数组元素，导致内存访问出现错误。改用迭代器遍历即可。

4、文件最后一行读取两遍

解决：经搜索后查明是由于 C++文件 eof( )函数特性引起。文件终止符在读完数据的下一个，因此读完最后的数据时文件指针并没指向终止符，此时循环读取尚未终止，程序会再读出一次最后一行信息。将 while 的循环条件改为 fin.peek( )!=EOF 即可。

六、心得体会

通过本次实验，我对于 C++类与对象的特性掌握的更加彻底，对语言的理解也更为深入。同时在进行项目的过程中遇到和解决的问题都对我的编程和解决问题的能力有了更大的提升。

此外，由于我没有在完全设计好模块结构后才开始编码，导致我进行了大量的重复工作，将写好的代码推倒重建。因此我明白了进行软件开发时总体设计的重要性，应该先设计好系统的框架再编码实现，可以有效避免结构性的错误。