电商交易平台设计与实现—实验报告

北京邮电大学计算机学院 2019211301班 池纪君 2019213688

实验内容

实验目的

随着移动互联网的发展，电商平台已经广泛地融入人们的生活。此次作业的任务是使用C++语言，基于面向对象的程序设计方法，设计并实现一个简单的电商交易平台，提供**用户管理、商品管理、交易管理**等功能。

题目分析

1. 题目一：账户管理子系统和商品管理子系统（单机版）
    1. 系统总体信息管理

        建立`general.data`文件，保存当前系统中所有的用户数量和商品数量

    2. 用户信息管理

        建立用户模块，买家类和商家类对用户基类进行继承。用户信息转化为**等长的二进制块**存放在`data/user.data`中，用户id即为在二进制文件中的数据块索引。

    3. 商品信息管理

        建立用户模块，书本类、食物类和服饰类对商品基类进行继承。商品信息转化为**等长的二进制块**存放在`data/product.data`中，商品id即为在二进制文件中的数据块索引。

2. 题目二：交易管理子系统（单机版）
    1. 购物车管理

        买家的购物车信息存储在买家的用户信息中，在读取用户信息 时同时会将购物车信息读出。由于用户信息定长，故购物车具有数量限制。在实现过程中，取购物车最大商品数量为5，符合实际购买需求。

    2. 订单和支付管理

        选择购物车的商品生成订单，冻结系统中的相关商品，计算并显示订单总金额。若订单支付成功，则释放订单数据，扣除用户的余额，给商家充值。若支付失败，则将订单中的所有商品数量复原。

3. 题目三：电商交易平台（网络版）
    1. 客户端和服务端之间的通信问题

        采用`QTcpSocket`进行通信，系统服务端设置TCP服务器，并绑定系统指定端口（实现过程中设置`8888`），开始监听。系统客户端设置TCPSocket并连接到TCP服务端，实现可靠的TCP传输服务。实现过程中，对于服务端→客户端和客户端→服务端的TCP报文分别进行了协议设计，以传输不同类型的数据信息。

    2. 错误处理能力

        实现过程中，为了应对用户的错误输入，提高程序鲁棒性，在用户界面（UI），客户端和服务端都进行了相应的错误检测和错误处理。

    3. 多客户端处理能力

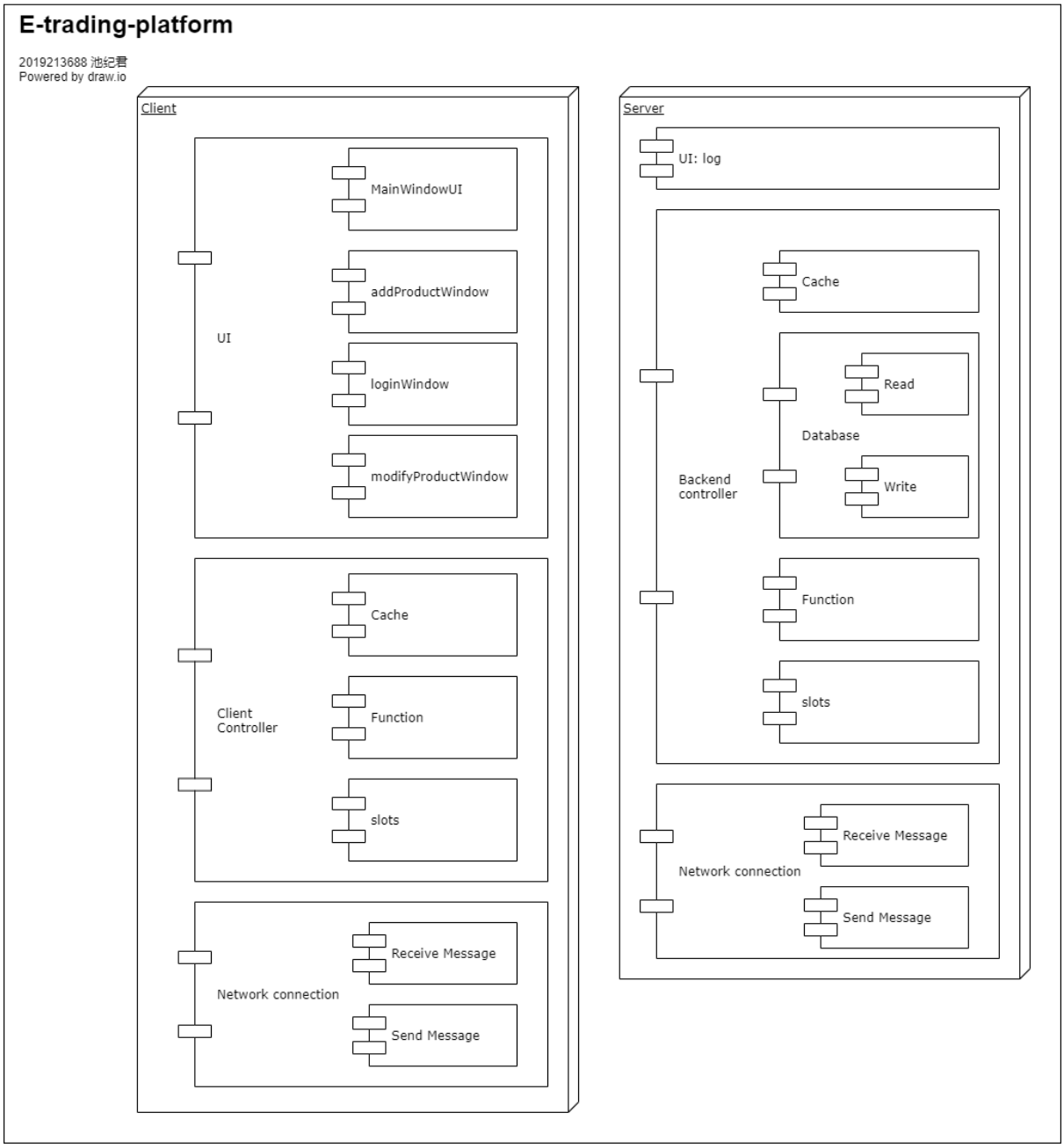
在本系统中，只有服务端具有数据处理能力，采用单线程的方式实现多客户端的处理。每个客户端通过TCPSocket连接到服务端的TCP服务器上，TCP服务器将每个TCP连接依次标号，并将描述信息保存在`服务列表`中。同时，TCP服务器给客户端发送其分配的编号，客户端在此后发送的请求包中的ID均为被分配到的编号。TCP服务器上触发一个`接受缓冲区就绪信号`时，将扫描每个`服务列表`中的TCPSocket描述符，如果当前TCPSocket对应的缓冲区不为空，则读取信息并处理。由于本系统中交互的数据量并不大，因此不会产生TCP传送部分包的情况，直接`readAll()`即可。

实验环境

1. C++ 11
2. QT6.1 (mingw810_64)
3. Windows

实验总体设计

模块设计



数据类设计

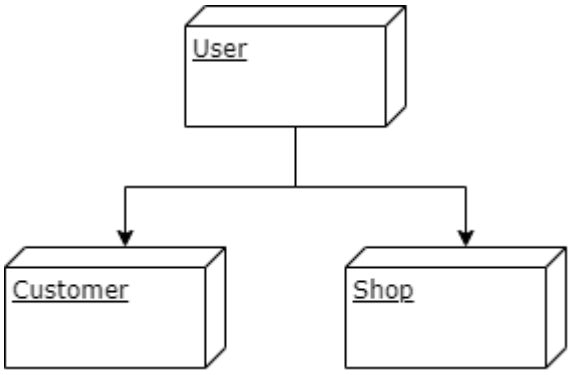
用户User类：

1. 存储结构：

```
/*
data/user.data ( USER DATA BLOCK )
                                1  1  1  1  1  1
   0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  (byte)
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 |     id    |    type   | chart_num | order_num |
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
 |       balance         |       nickname        |
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 |    nickname  |            password            |
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 |     |
 +--+--+
id: 4(int)
type: 4(int)
chart_num: 4(int)
order_num: 4(int)
balance: 8(double)
nickname: 13(13*char: 12 + '\0')
password: 13(13*char: 12 + '\0')
(total 50 bytes)
*/
```

2. 类继承关系：



3. 类友元函数：

   QT中的QDataStream可以利用流运算`<<`和`>>`，将一个常见的C++数据结构（`int`，`double`，`string`等）写入QByteArray中。使用该友元函数的目的在于，可以通过`in<<customer`或`in<<shop`这一简单的语句直接完成Customer和Shop读取QByteArray中存储的值，也可以通过`out>>customer`或`out>>shop`将Customer和Shop的直接存储到QByteArray中。在客户端和服务端进行通信的时候，通过这种流运算的方式使得数据交换十分简单高效。

```
// friend function
// overload the input and output functions
friend QDataStream& operator<<(QDataStream& out, User& user);
friend QDataStream& operator>>(QDataStream& in, User& user);
```

4. 类实现见代码。

商品Product类

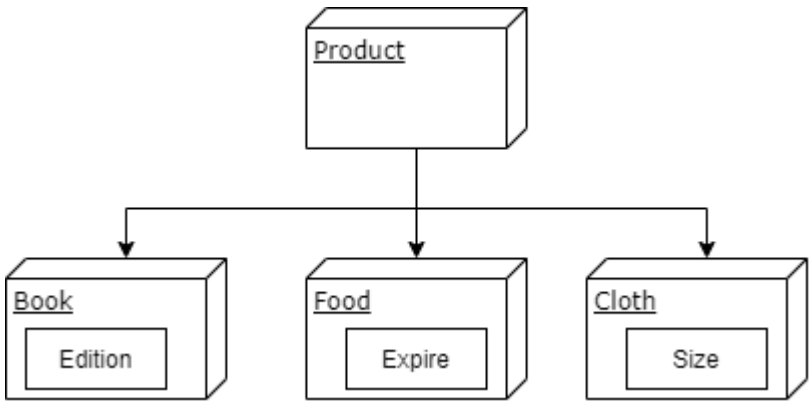
1. 存储结构：

```
/*
data/product.data ( PRODUCTS DATA BLOCK )
                                1  1  1  1  1  1
   0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  (byte)
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 |     id    | belong_id |   stock   |    type   |
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 |           price       |        discount       |
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 |                     name                      |
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 |  feature  |                 description
 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

 +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
             |
 +--+--+--+--+
id: 4(int)
belong_id: 4(int)
stock: 4(int)
type: 4(int)
price: 8(double)
discount: 8(double)
name: 16(16*char)
feature: 4(int)
description: 48(48*char)
(total: 100 bytes)
*/
```

2. 类继承关系：



3. 类友元函数：

   类友元函数的作用同上，可以更便携高效地进行Product数据读写QByteArray操作。

```
// friend function
// overload the input and output functions
friend QDataStream& operator<<(QDataStream& out, Product& product);
friend QDataStream& operator>>(QDataStream& in, Product& product);
```

4. 类实现见代码。

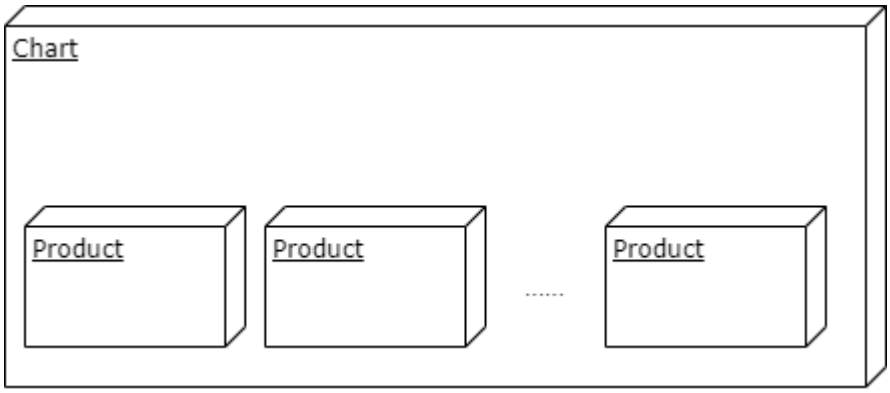
购物车Chart类

1. 存储结构：

```
/*
data/chart.data ( CHART DATA BLOCK )

 0 (4 bytes) 4 (4 bytes) 8 (100 bytes) 108           208    408           508
 +--+--+--+--+--+--+--+--+-------------+-------------+------+-------------+
 |customer_id|    size   |  product_0  |  product_1  | ...  |  product_4  |
 +--+--+--+--+--+--+--+--+-------------+-------------+------+-------------+
 product: 100 bytes
 ** CAUTION: if item_num[i]==0, we consider this item is deleted. **
(total:  508 bytes)
*/
```

2. 类结构：



3. 类友元函数：

   类友元函数的作用同上，可以更便携高效地进行Chart数据读写QByteArray操作。

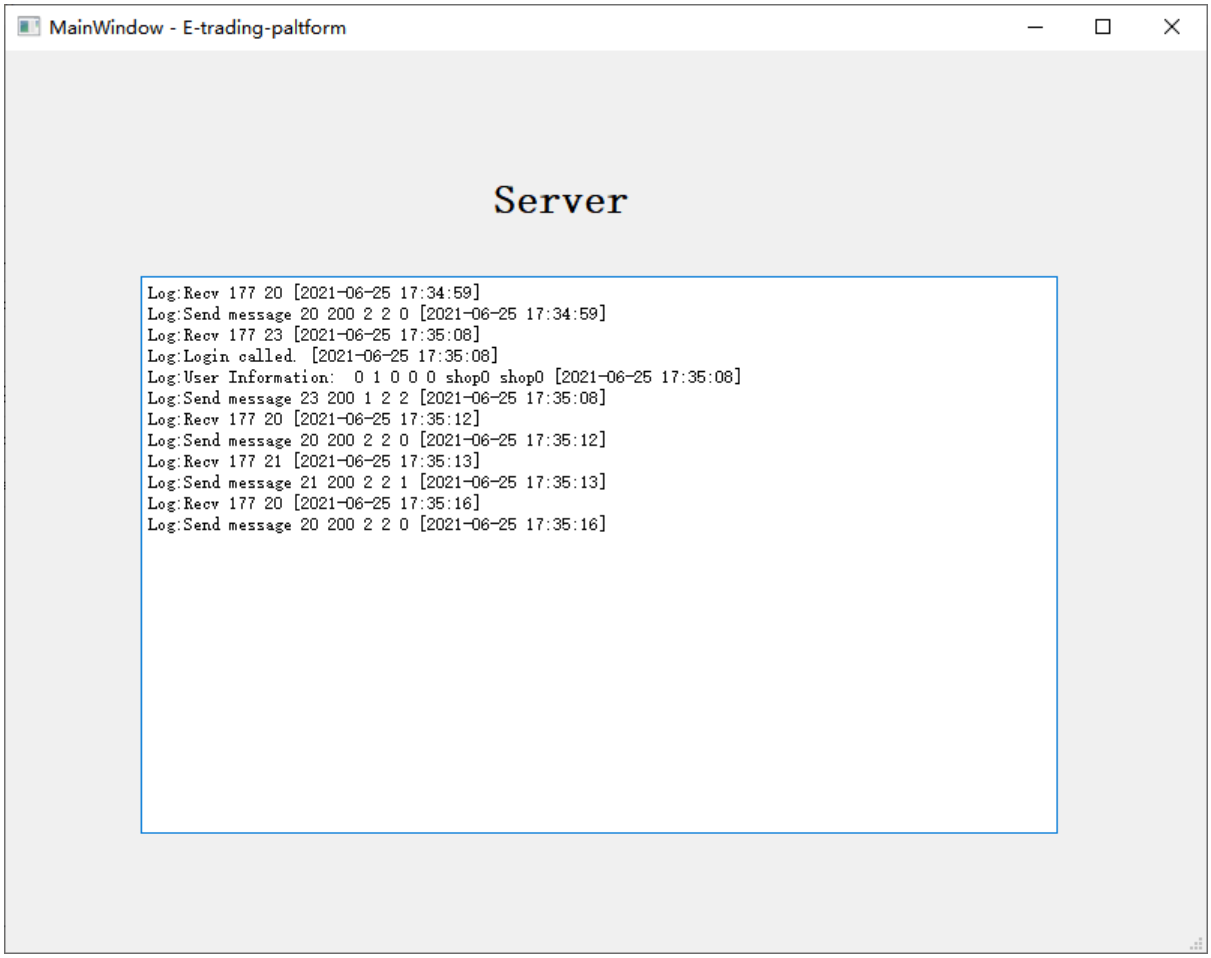
```
// friend function
// overload the input and output functions
friend QDataStream& operator<<(QDataStream& out, Chart& chart);
friend QDataStream& operator>>(QDataStream& in, Chart& chart);
```

4. 类实现见代码。

界面设计和展示

服务端界面设计

服务端界面设计简单，主要包含和各个客户端之间的通信记录。

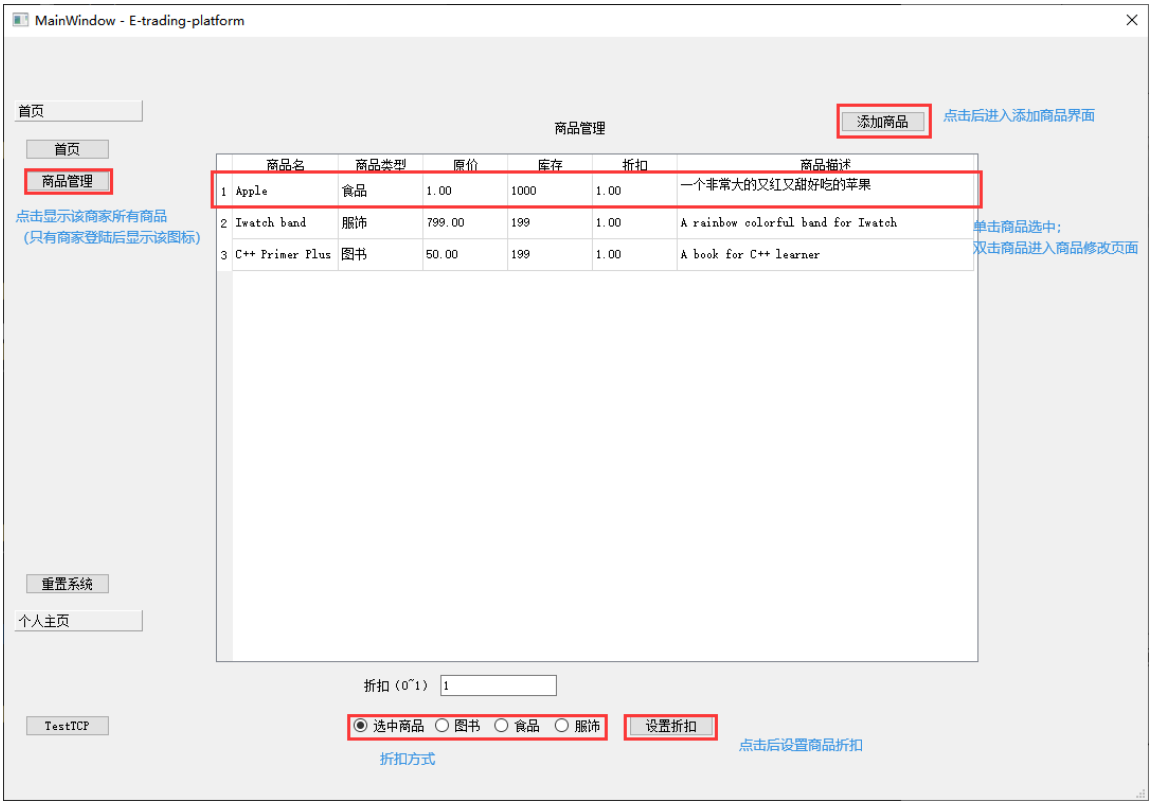


客户端界面设计

1. 首页

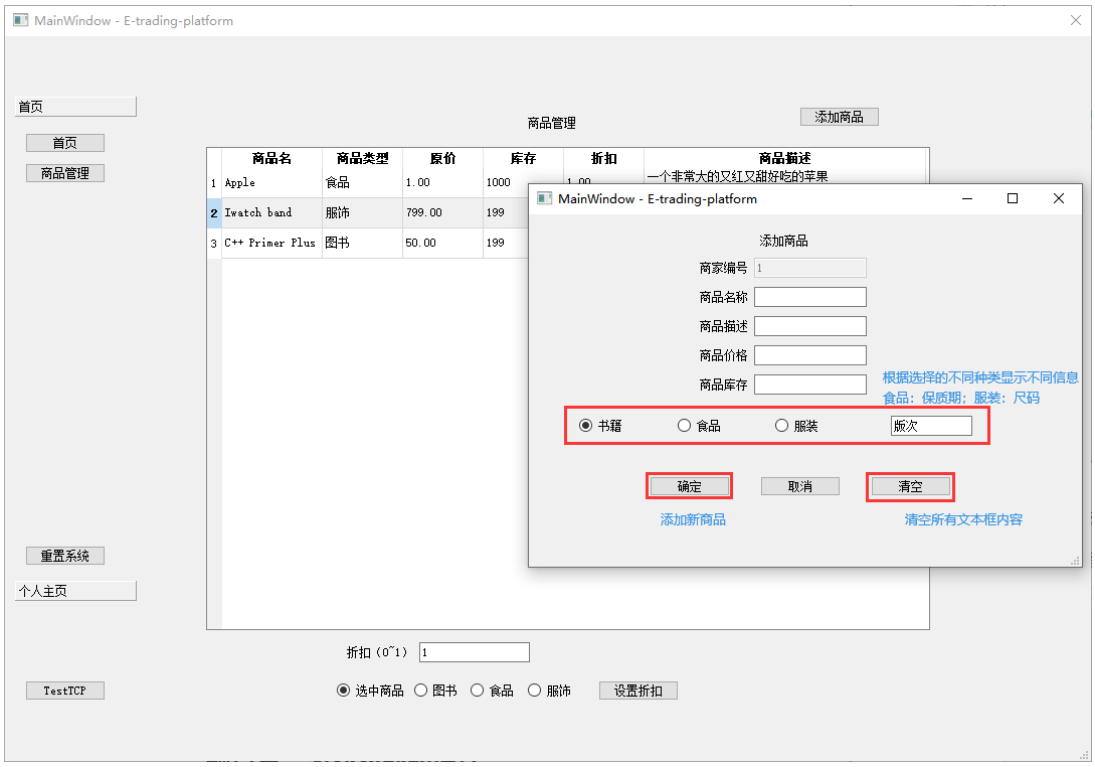
2. 登陆界面

3. 商品管理页面

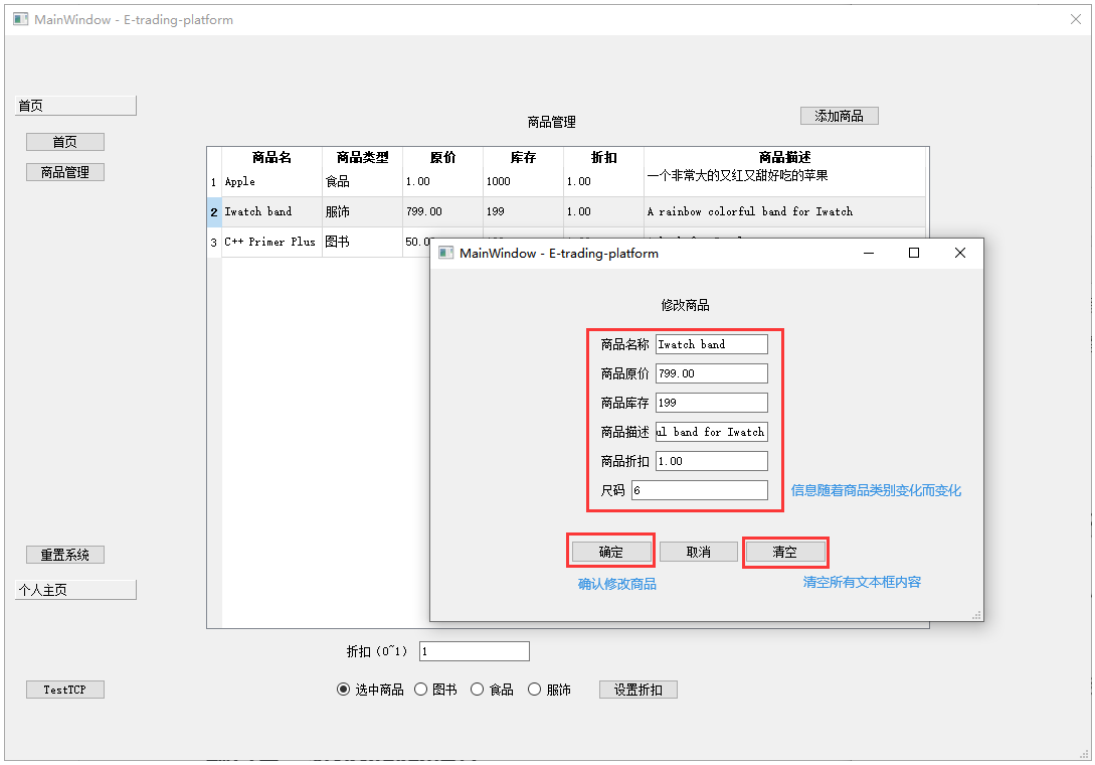


4. 商品添加、修改页面

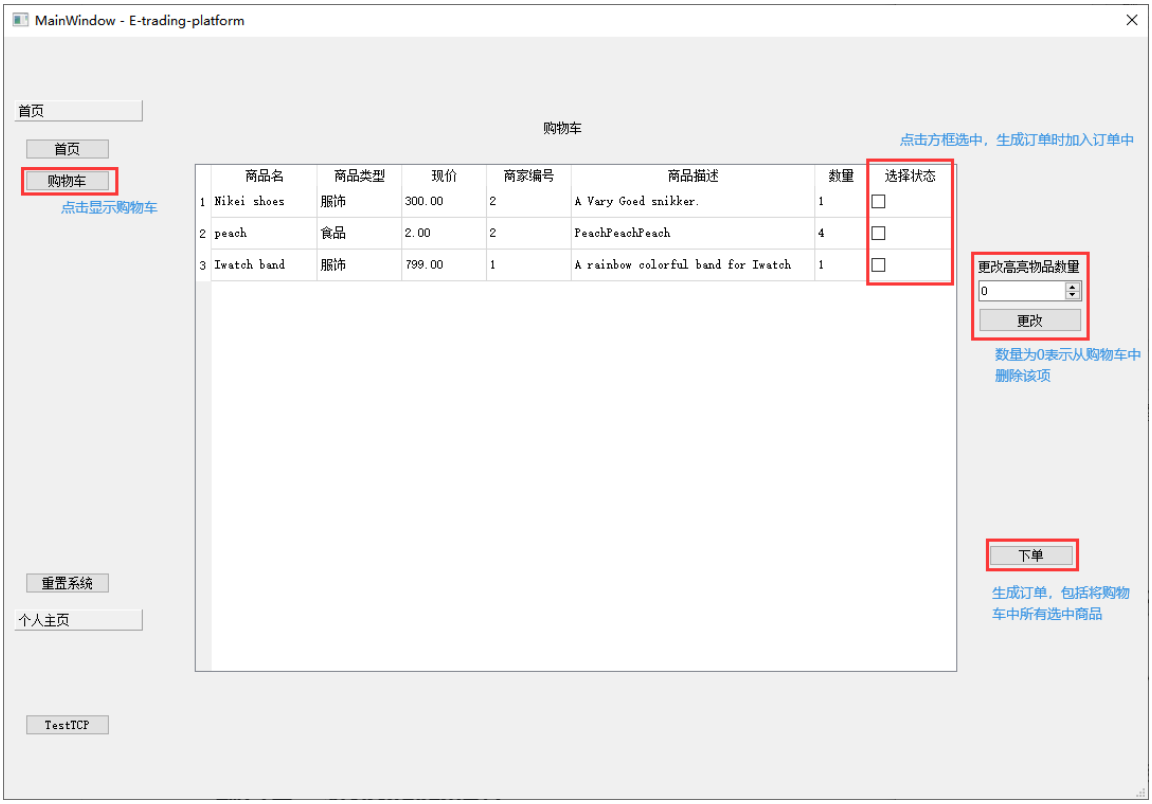
1. 商品添加



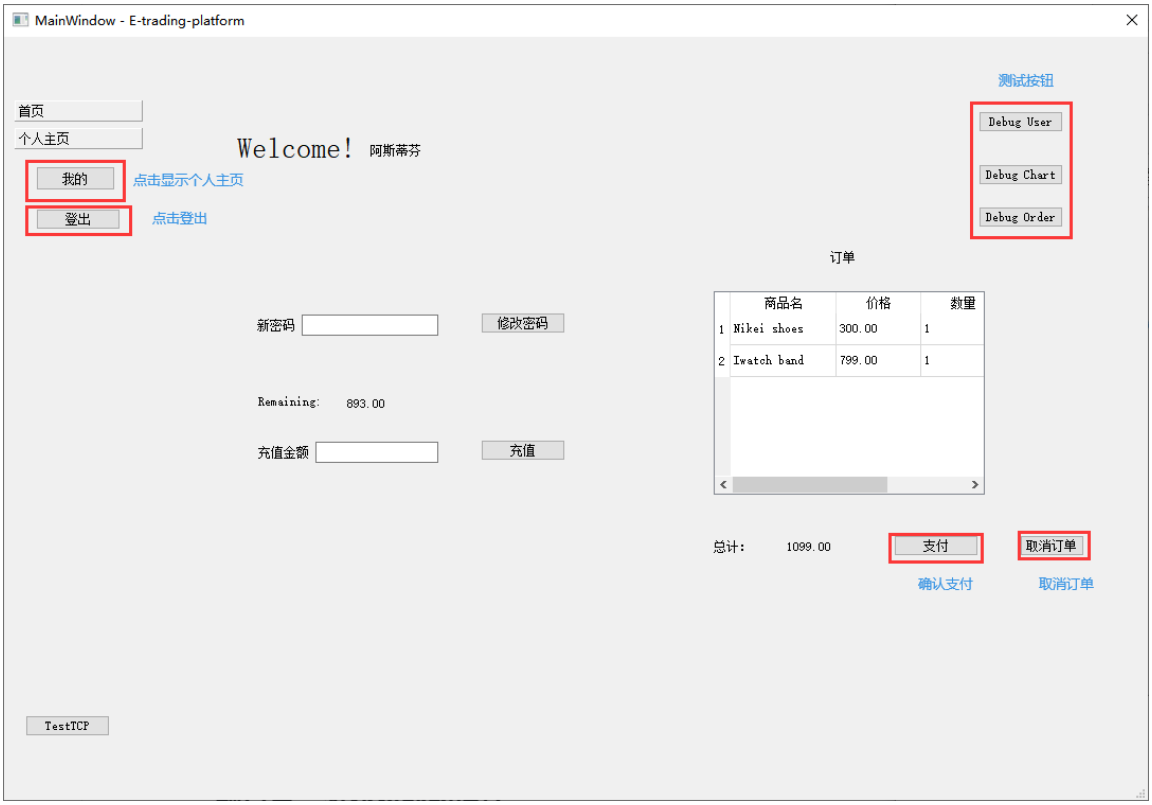
2. 商品修改



5. 购物车界面



6. 个人页面



网络协议设计

服务端→客户端

1. 数据报格式如下为：

```
/*
   0  1  2  3  4  5  6  (byte)
 +--+--+--+--+--+--+--+~+--+
 |id|op| data       ....   |
 +--+--+--+--+--+--+--+~+--+
*/
```

2. 数据包解释：

   id：服务端给客户端分配的ID。

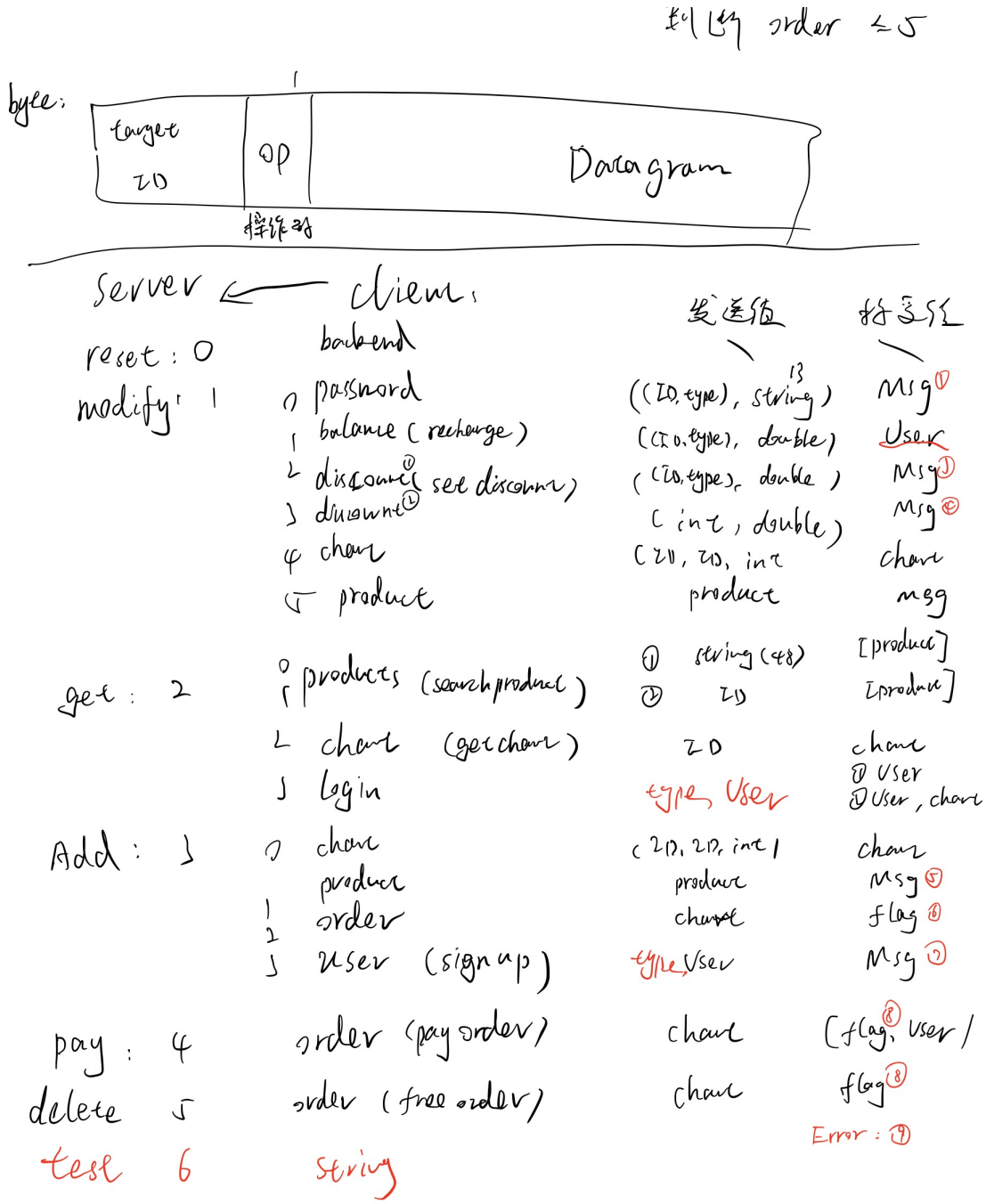
   op：服务类型，共有20种不同的Opcode，每种类型的请求将不同的数据加入到data字段，传送给服务端。

```
enum Opcode {
  RESET = 0,
  MODIFY_PASSWORD = 10,
  RECHARGE = 11,
  SET_DISCOUNT_INDEX = 12,
```

```
  SET_DISCOUNT_TYPE = 13,
  MODIFY_CHART = 14,
  MODIFY_PRODUCT = 15,
  CONSUME = 16,
  SEARCH_PRODUCT_DESC = 20,
  SEARCH_PRODUCT_ID = 21,
  GET_CHART = 22,
  LOGIN = 23,
  ADD_TO_CHART = 30,
  ADD_PRODUCT = 31,
  ADD_ORDER = 32,
  ADD_USER = 33,
  PAY_ORDER = 4,
  FREE_ORDER = 5,
  TEST = 6,
  CONNECT = 7
};
```

data：客户端给服务端发送的数据信息，因服务类型而异。

参考手稿：



客户端→服务端

1. 数据报格式

```
/*
   0   1   2   3   4   5 (byte)
 +---+---+---+---+---+---+~+---+
```

```
 |id |op |fg1|fg2|fg3|fg4|data |
 +---+---+---+---+---+---+~+---+
max size: 6 + 50(user size) + 508(chart size) + product_num * 100(product size)
*/
```

2. 数据报解释

   id：客户端id

   op：客户端的服务类型，同上。

   fg1：请求的执行结果代号（`200`为正常执行，其他为不正常结果，客户端对不同的结果码进行错误报告）。

   fg2：发送用户信息标志。

   fg3：发送购物车信息标志。

   fg4：发送多个商品信息标志。

   data：当有标志位fg=1时，data将依次携带用户、购物车、多个商品等信息。

实验系统特性

- 使用C++11编写
- 具有良好的鲁棒性。系统从输入界面，客户端和系统服务端的所有处理函数都采用`try catch`操作执行，对错误有良好的处理。
- 采用ID进行地址索引，加快读写用户信息的效率。
- 人机交互GUI界面，支持多客户端同时操作。
- 对异常的程序结束进行有效处理，包括保存当前已修改数据，取消用户订单，恢复已冻结的商品。

实验总结

重要问题实现和解决方案

调用同基类的多个派生类对象

在实现的过程中，没用基类的指针进行赋值，而是采用先传输类型编号的方式，预先获取商品和用户派生类的类别，再通过判断语句分别定义派生类的对象进行赋值。

信息存储和内存读写

本系统中，根据用户和商品的ID索引其在文件数据中的块下标，通过`std::fstream`中的`seekg()`和`seekp()`快速定位。在此之后，通过`std::fstream`中的`write`和`read`对用户或商品的逐个属性读写。这种做法的优点是不用遍历整个数据文件，完成对数据文件的读写，时间复杂度$O(1)$。

网络通信

使用QDataStream和QByteArray读取socket，可以很方便快捷地在两个TCPSocket之间进行数据的传输，并且保持数据的先后顺序。

经验总结

系统的模型

由于本项目在第一、二阶段中就基于QT开发，存在大量的信号和槽函数，系统的控制，数据文件的读写和用户界面融合在一段文件中，导致很难进行下一个阶段的开发。最后，我将阶段二中的代码进行重构，将服务端所需要的所有的数据读写，控制和用户界面分离开来，加上网络通信模块，最后用`Backend`类将这四个模块整合在一起，最后形成了最终的版本。在系统的构建初期，就应该先将控制，数据读写和用户界面分离开来，以更好地进行开发。

调试信息

系统级项目很难进行断点调试，因此依靠输出的信息进行调试十分重要。同时，应该将不同种类调试信息加上不同的标记。本系统中共存在5种调试信息，分别是：`Log`，`Warning`，`Critical`，`Fatal`，`Info`。输出函数将通过不同的调试等级输出不同的标签信息。