一、代码实现，包括；

（1）主要业务逻辑实现；

- 注册对话窗口实现

```
// 确认注册
void RegisterDialog::on_pushButtonConfirm_clicked() {
    QJsonObject body;
    body["username"] = ui->lineEditUsername->text();    // 用户名编辑栏
    body["password"] = ui->lineEditPassword->text();    // 密码编辑栏
    My::Response res = client->post("/register", QJsonObject(), body);
    if (res.status == 200) {
        QMessageBox::information(this, "注册成功", "请回到用户中心继续登录");
    } else {
        QMessageBox::critical(this, "注册失败", res.error);
    }
    close();
}

// 取消注册
void RegisterDialog::on_pushButtonCancel_clicked() { close(); }
```

- 登录对话窗口实现

```
// 确认登录
void LoginDialog::on_pushButtonConfirm_clicked() {
    My::TcpClient *client = new My::TcpClient(this);
    QJsonObject body;
    body["username"] = ui->lineEditUsername->text();    // 用户名编辑栏
    body["password"] = ui->lineEditPassword->text();    // 密码编辑栏
    My::Response res = client->post("/login", My::Headers(), body);
    if (res.status == 200) {
        int userId = res.body["id"].toInt();

        qDebug() << "LoginDialog::on_pushButtonConfirm_clicked:" << userId
                 << res.body["username"].toString()
                 << res.body["password"].toString();
    } else {
        QMessageBox::critical(this, "login failed", res.error);
    }
    client->deleteLater();
    close();
}

// 取消登录
```

```
void LoginDialog::on_pushButtonCancel_clicked() { close(); }
```

- 浏览商店，获取最新商品信息并显示

```
// 刷新界面
void ShopWidget::refresh() {
    int index = 1;
    client->postAsync("/shop", My::Headers(), My::Body(index));
    connect(client, &My::TcpClient::readyRead, this, [&](const My::Response &res)
{
        if (res.status == 200) {
            ui->tableWidget->clearContents();
            QPixmap image;
            QTableWidgetItem *imageItem = new QTableWidgetItem;
            QTableWidgetItem *nameItem = new QTableWidgetItem;
            QTableWidgetItem *priceItem = new QTableWidgetItem;
            QTableWidgetItem *stockItem = new QTableWidgetItem;
            QTableWidgetItem *descriptionItem = new QTableWidgetItem;

if(image.loadFromData(QByteArray::fromBase64(res.body["image"].toString().toUtf8()
))) {
                imageItem->setIcon(image);
            } else {
                imageItem->setText("图片加载失败");
            }
            nameItem->setText(res.body["name"].toString());
            priceItem->setText(QString::number(res.body["price"].toDouble()));
            stockItem->setText(QString::number(res.body["stock"].toInt()));
            descriptionItem->setText(res.body["description"].toString());
            ui->tableWidget->setItem(index - 1, 0, imageItem);
            ui->tableWidget->setItem(index - 1, 1, nameItem);
            ui->tableWidget->setItem(index - 1, 2, priceItem);
            ui->tableWidget->setItem(index - 1, 3, stockItem);
            ui->tableWidget->setItem(index - 1, 4, descriptionItem);
            client->postAsync("shop", My::Headers(), My::Body(++index));
        }
    });
}
```

- 支付结算

```
    // 敬请期待
```

- 添加到购物车

```
    // 敬请期待
```

（2）通信实现；

以下是通信交互对象的各种属性，以及与字节流和JSON对象的转换，方便后续处理。交互对象类被请求类和响应类继承，各自扩写其特殊属性。

```
class TCPINTERACTION_EXPORT TcpInteraction {
public:
    enum Item {          // Qt           QJsonValue  注释
        // 公共属性
        // 报文定长前缀（解决粘包问题）
        // ContentType  // qint64       Integer     内容长度
        // 报文内容
        IsValid,        // bool         Bool        是否有效
        DateTime,       // QDateTime    String      发送时间
        HostAddress,    // QHostAddress String      服务器地址
        Port,           // quint64      Integer     服务器端口号
        Body,           // QJsonObject  Object      报文主体
        // 请求属性
        AuthorizedToken,// QString      String      授权校验字段
        Route,          // QString      String      路由
        Timeout,        // qint64       Integer     超时阈值
        // 响应属性
        Success,        // bool         Bool        是否成功
        StatusType,     // QString      String      状态类型
        StatusDetail    // QString      String      具体状态信息
    };  // 交互对象属性

public:
    TcpInteraction();
    TcpInteraction(bool isValid, const QDateTime &dateTime,
                   const QHostAddress &hostAddress, quint64 port,
                   const QJsonObject &body);
    virtual ~TcpInteraction();
    virtual operator QJsonObject() const = 0;   // 将自身转换成JSON对象
    operator QByteArray() const;    // 将自身转换成字节流
    static QString toString(Item data); // 将属性转换成报文中的字符串
};
```

（3）持久层（数据库）实现；

以响应登录请求为例，以下是部分持久层的代码实现

```
Response LoginController::post(const Headers &headers, const Body &body) {
    Q_UNUSED(headers);
    QSqlQuery query(db);
```

```
    User user(body.toObject());
    if (user.username.isEmpty() || user.password.isEmpty()) {
        return Response(400, Headers(), Body(),
                        "username and password cannot be empty");
    }
    query.prepare(
        "SELECT rowid, * FROM users WHERE username = :username AND password = "
        ":password AND active = 1;");
    query.bindValue(":username", user.username);
    query.bindValue(":password", user.password);
    if (!query.exec()) {
        return Response(500, Headers(), Body(), "error occurred when quering
user");
    }
    if (!query.next()) {
        return Response(404, Headers(), Body(), "user not found");
    }
    User responseUser(query.value("rowid").toInt(),
                      query.value("username").toString(),
                      query.value("password").toString(), true);
    return Response(200, Headers(), QJsonObject(responseUser), "");
}
```

（4）UI实现

二、遇见了哪些技术问题，如何解决的？

问题	描述	解决方案
粘包问题	发送的字节流前后粘连，可能导致对方解析失败	在所有发送的报文添加固定长度的前缀表示将要读取的报文长度，从而保证读取准确长度的信息
超时问题	未检测超时，不能给前端返回超时信息，同时在服务器断开并重连后会处理本不应处理的超时请求	客户端启用计时器检测超时，而在服务端通过校验报文中的发送时间来检测
高并发问题	多个客户端同时发送请求，采用单线程的服务器无法快速响应	利用线程池启动多线程，同时处理请求

三、你现在完全掌握的部分和困难的部分都是什么？

- 完全掌握的部分

线程池、数据库、网络通信等关键技术成功掌握，通信协议成功建立并最终确定。

- 困难的部分

业务逻辑的分析和实现

四、设计时间已经过半，你的计划进度有无偏差？

目前进度慢于原先计划，主要原因是在技术学习和通信协议设计上耗费了大量时间，经历了多次大规模的重构，导致业务逻辑难以进行。

五、本周拟解决什么问题，如何解决？

集中精力攻克业务逻辑，在搭建好的完备的通信、技术、架构基础上添加代码，并最终整合成完整项目。

（3）持久层（数据库）实现；

表 (8)

名称	类型	架构
cart_items		CREATE TABLE "cart_items" ("id" TEXT, "cart_id" TEXT NOT NULL, "product_id" TEXT NOT NULL, "quantity" INTEGER NOT NULL, "is_deleted" BOOLEAN NOT NULL, PRIMARY KEY("id"))
id	TEXT	"id" TEXT
cart_id	TEXT	"cart_id" TEXT NOT NULL
product_id	TEXT	"product_id" TEXT NOT NULL
quantity	INTEGER	"quantity" INTEGER NOT NULL
is_deleted	BOOLEAN	"is_deleted" BOOLEAN NOT NULL
carts		CREATE TABLE "carts" ("id" TEXT, "user_id" TEXT NOT NULL, "created_at" DATETIME NOT NULL, "deleted_at" INTEGER NOT NULL, "is_deleted" BOOLEAN NOT NULL, PRIMARY KEY("id"))
id	TEXT	"id" TEXT
user_id	TEXT	"user_id" TEXT NOT NULL
created_at	DATETIME	"created_at" DATETIME NOT NULL
deleted_at	INTEGER	"deleted_at" INTEGER NOT NULL
is_deleted	BOOLEAN	"is_deleted" BOOLEAN NOT NULL
login_records		CREATE TABLE "login_records" ("authorized_token" TEXT, "user_id" TEXT NOT NULL, "loggedin_at" DATETIME NOT NULL, "loggedout_at" DATETIME NOT NULL, "is_deleted" BOOL NOT NULL, PRIMARY KEY("authorized_token"))
authorized_token	TEXT	"authorized_token" TEXT
user_id	TEXT	"user_id" TEXT NOT NULL
loggedin_at	DATETIME	"loggedin_at" DATETIME NOT NULL
loggedout_at	DATETIME	"loggedout_at" DATETIME NOT NULL
is_deleted	BOOL	"is_deleted" BOOL NOT NULL
notifications		CREATE TABLE "notifications" ("id" TEXT, "user_id" TEXT NOT NULL, "content" TEXT NOT NULL, "created_at" DATETIME NOT NULL, "deleted_at" DATETIME NOT NULL, "is_deleted" BOOLEAN NOT NULL, PRIMARY KEY("id"))
id	TEXT	"id" TEXT
user_id	TEXT	"user_id" TEXT NOT NULL
content	TEXT	"content" TEXT NOT NULL
created_at	DATETIME	"created_at" DATETIME NOT NULL
deleted_at	DATETIME	"deleted_at" DATETIME NOT NULL
is_deleted	BOOLEAN	"is_deleted" BOOLEAN NOT NULL
order_items		CREATE TABLE "order_items" ("id" TEXT, "order_id" TEXT NOT NULL, "product_id" TEXT NOT NULL, "quantity" INTEGER NOT NULL, "cost" REAL NOT NULL, "is_deleted" BOOLEAN NOT NULL, PRIMARY KEY("id"))
id	TEXT	"id" TEXT
order_id	TEXT	"order_id" TEXT NOT NULL
product_id	TEXT	"product_id" TEXT NOT NULL
quantity	INTEGER	"quantity" INTEGER NOT NULL
cost	REAL	"cost" REAL NOT NULL
is_deleted	BOOLEAN	"is_deleted" BOOLEAN NOT NULL
orders		CREATE TABLE "orders" ("id" TEXT, "user_id" TEXT NOT NULL, "cost" REAL NOT NULL, "status" TEXT NOT NULL, "created_at" DATETIME NOT NULL, "cancelled_at" DATETIME NOT NULL, "is_deleted" BOOLEAN NOT NULL, PRIMARY KEY("id"))
id	TEXT	"id" TEXT

（3）持久层（数据库）实现；

名称	类型	架构
user_id	TEXT	"user_id" TEXT NOT NULL
cost	REAL	"cost" REAL NOT NULL
status	TEXT	"status" TEXT NOT NULL
created_at	DATETIME	"created_at" DATETIME NOT NULL
cancelled_at	DATETIME	"cancelled_at" DATETIME NOT NULL
is_deleted	BOOLEAN	"is_deleted" BOOLEAN NOT NULL
products		CREATE TABLE "products" ("id" TEXT, "name" TEXT NOT NULL, "description" TEXT NOT NULL, "price" REAL NOT NULL, "stock" INTEGER NOT NULL, "category" TEXT NOT NULL, "image_url" TEXT, "listed_at" DATETIME NOT NULL, "delisted_at" DATETIME NOT NULL, "is_deleted" BOOLEAN NOT NULL, PRIMARY KEY("id"))
id	TEXT	"id" TEXT
name	TEXT	"name" TEXT NOT NULL
description	TEXT	"description" TEXT NOT NULL
price	REAL	"price" REAL NOT NULL
stock	INTEGER	"stock" INTEGER NOT NULL
category	TEXT	"category" TEXT NOT NULL
image_url	TEXT	"image_url" TEXT
listed_at	DATETIME	"listed_at" DATETIME NOT NULL
delisted_at	DATETIME	"delisted_at" DATETIME NOT NULL
is_deleted	BOOLEAN	"is_deleted" BOOLEAN NOT NULL
users		CREATE TABLE "users" ("id" TEXT, "username" TEXT NOT NULL, "password" TEXT NOT NULL, "avatar_url" TEXT, "registered_at" DATETIME NOT NULL, "unregistered_at" DATETIME NOT NULL, "is_deleted" BOOLEAN NOT NULL, PRIMARY KEY("id"))
id	TEXT	"id" TEXT
username	TEXT	"username" TEXT NOT NULL
password	TEXT	"password" TEXT NOT NULL
avatar_url	TEXT	"avatar_url" TEXT
registered_at	DATETIME	"registered_at" DATETIME NOT NULL
unregistered_at	DATETIME	"unregistered_at" DATETIME NOT NULL
is_deleted	BOOLEAN	"is_deleted" BOOLEAN NOT NULL

索引 (0)

名称	类型	架构

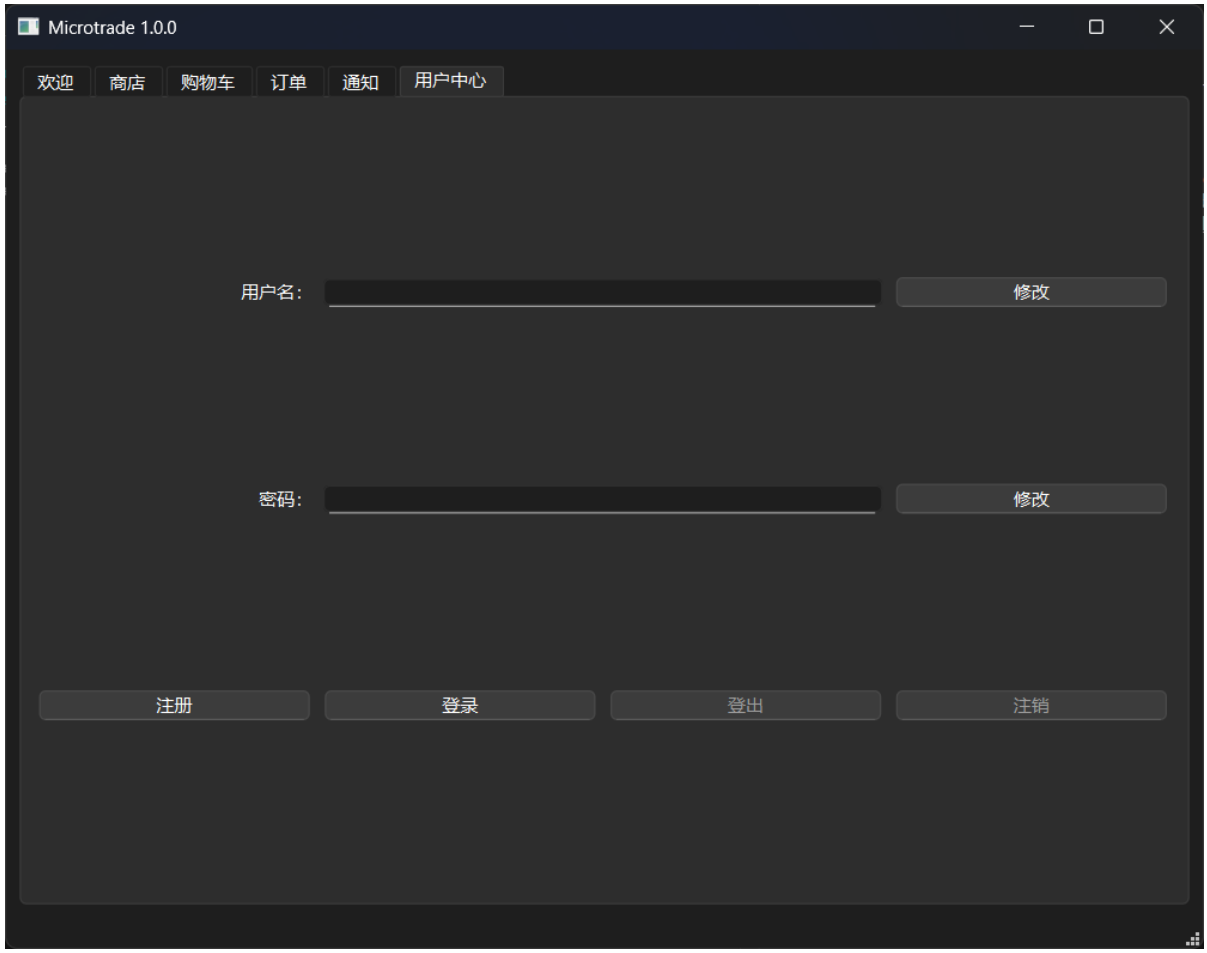
视图 (0)

名称	类型	架构

触发器 (0)

名称	类型	架构

（4）UI实现


Microtrade 1.0.0
欢迎
商店
购物车
订单
通知
用户中心
用户名:
修改
密码:
修改
注册
登录
登出
注销


Microtrade Server 1.0.0
欢迎
控制面板
勾选以开启监听