第四周进度报告

一、代码实现(提炼出最核心的部分)

一、代码实现

（1）主要业务逻辑实现

服务端采用C++/Java混合架构，核心业务逻辑已实现：用户管理、商品管理、购物车、订单处理、促销与优惠券、地址管理等。

订单系统支持完整状态流转（待确认、已支付、已发货、已完成、取消、退款），并实现了退款和物流跟踪。

促销系统已支持优惠券、基础折扣，价格计算逻辑健全。

// 以订单为例，核心业务流程：

// 1. 校验用户、商品、库存、地址等信息

// 2. 计算金额、促销、优惠券

// 3. 创建订单、订单明细、扣减库存

// 4. 订单状态流转（pending→paid→shipped→delivered→completed）

// 5. 售后/退款/取消等业务处理

// 伪代码示例：

bool createOrder(userId, cartItems, addressId, couponCode) {

if (!checkUser(userId) || !checkAddress(addressId)) return false;

double total = calcTotal(cartItems);

double discount = calcDiscount(couponCode, total);

double finalAmount = total - discount;

long orderId = insertOrder(userId, finalAmount, addressId);

for (auto item : cartItems) {

insertOrderItem(orderId, item);

updateStock(item.productId, -item.quantity);

}

return true;

}

// 购物车、商品、用户、促销、评论等业务核心代码片段可参考前述各服务类实现。

// 用户服务核心方法

class UserService {

public:

bool registerUser(const std::string &username, const std::string &password) {

if (username.empty() || password.empty()) return false;

if (userExists(username)) return false;

std::string hash = hashPassword(password);

return insertUser(username, hash);

}

bool login(const std::string &username, const std::string &password) {

std::string hash = getPasswordHash(username);

return verifyPassword(password, hash);

}

};

// 商品服务核心方法

class ProductService {

public:

std::vector<Product> listProducts(const std::string &category) {

return queryProducts(category);

}

bool updateStock(int productId, int delta) {

int stock = getStock(productId);

if (stock + delta < 0) return false;

return setStock(productId, stock + delta);

}

};

// 促销与优惠券

class PromotionService {

public:

double calcDiscount(const std::string &couponCode, double total) {

if (couponCode.empty()) return 0.0;

Coupon coupon = getCoupon(couponCode);

if (!coupon.valid) return 0.0;

if (total < coupon.minAmount) return 0.0;

return coupon.discountAmount;

}

};

// 评论服务

class ReviewService {

public:

bool addReview(int userId, int productId, const std::string &content) {

if (content.empty()) return false;

return insertReview(userId, productId, content);

}

std::vector<Review> getReviews(int productId) {

return queryReviews(productId);

}

};

（2）通信实现

服务端基于Netty框架，支持高并发TCP长连接，协议采用文本命令+换行分隔，客户端与服务端双向通信。

客户端（命令行/Qt）通过TCP/IP协议与服务端交互，命令格式与响应格式已标准化，支持自动重连和会话管理。

业务层与持久层通过JNI与C++后端集成，部分功能已支持JSON协议扩展。

// 服务端 C++/Netty 处理 TCP 消息

struct Message {

int type; // 消息类型

int userId;

std::string data; // JSON数据

};

// 客户端 Qt 发送/接收消息

void NetworkManager::sendMessage(const Message &message) {

QJsonObject json;

json["type"] = message.type;

json["userId"] = message.userId;

json["data"] = QString::fromStdString(message.data);

QByteArray jsonData = QJsonDocument(json).toJson(QJsonDocument::Compact);

quint32 dataSize = jsonData.size();

socket->write(reinterpret\_cast<const char\*>(&dataSize), sizeof(dataSize));

socket->write(jsonData);

}

void NetworkManager::onReadyRead() {

quint32 dataSize;

socket->read(reinterpret\_cast<char\*>(&dataSize), sizeof(dataSize));

QByteArray jsonData = socket->read(dataSize);

QJsonDocument doc = QJsonDocument::fromJson(jsonData);

// 解析并分发消息

}

// 协议示例：

// 1001（商品展示），10020001（1号商品加入购物车），1003（结算当前购物车）；

// 20020001（1号商品加入购物车成功），2003结算成功

// 服务端消息分发

void Server::onMessageReceived(const Message &msg) {

switch (msg.type) {

case 1001: showProductList(msg.userId); break;

case 1002: addToCart(msg.userId, msg.data); break;

case 1003: checkout(msg.userId); break;

default: sendError(msg.userId, "Unknown command");

}

}

// 客户端命令处理

void Client::handleServerResponse(const Message &msg) {

if (msg.type == 20020001) {

showToast("加入购物车成功");

} else if (msg.type == 2003) {

showToast("订单已提交");

} else {

processOther(msg);

}

}

// 心跳与重连机制

void Client::startHeartbeat() {

heartbeatTimer->start(30000); // 每30秒发送一次心跳

}

void Client::onHeartbeatTimeout() {

sendHeartbeat();

if (!isConnected()) reconnect();

}

（3）持久层（数据库）实现

数据库采用MySQL，表结构设计规范，涵盖用户、商品、订单、购物车、促销、地址等核心数据。

持久层实现了SQL操作封装、连接池管理、事务处理，支持高并发读写。

已实现数据初始化脚本和部分数据迁移脚本。

// MySQL表结构（SQL脚本）

CREATE TABLE users (

user\_id INTEGER PRIMARY KEY AUTOINCREMENT,

username VARCHAR(50) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

phone VARCHAR(20),

create\_time DATETIME DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE products (

product\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name VARCHAR(200) NOT NULL,

price DECIMAL(10,2) NOT NULL,

stock INTEGER DEFAULT 0,

category VARCHAR(50),

description TEXT,

create\_time DATETIME DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE orders (

order\_id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id INTEGER NOT NULL,

total\_amount DECIMAL(10,2) NOT NULL,

status VARCHAR(20) DEFAULT 'pending',

create\_time DATETIME DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE order\_items (

item\_id INTEGER PRIMARY KEY AUTOINCREMENT,

order\_id INTEGER NOT NULL,

product\_id INTEGER NOT NULL,

quantity INTEGER NOT NULL,

price DECIMAL(10,2) NOT NULL

);

CREATE TABLE cart (

cart\_id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id INTEGER NOT NULL,

product\_id INTEGER NOT NULL,

quantity INTEGER NOT NULL,

add\_time DATETIME DEFAULT CURRENT\_TIMESTAMP

);

// C++持久层实现：

// executeQuery(sql) 封装所有SQL操作，自动处理连接池、异常、结果集解析。

// C++数据库操作封装

class DBManager {

public:

bool executeQuery(const std::string &sql) {

auto conn = getConnection();

try {

conn->exec(sql);

return true;

} catch (std::exception &e) {

logError(e.what());

return false;

}

}

bool beginTransaction() { return executeQuery("BEGIN;"); }

bool commit() { return executeQuery("COMMIT;"); }

bool rollback() { return executeQuery("ROLLBACK;"); }

};

// Java持久层（Spring Data JPA）

@Repository

public interface OrderRepository extends JpaRepository<Order, Long> {

List<Order> findByUserId(Long userId);

}

@Service

public class OrderService {

@Autowired

private OrderRepository orderRepository;

public Order createOrder(Order order) {

return orderRepository.save(order);

}

public List<Order> getUserOrders(Long userId) {

return orderRepository.findByUserId(userId);

}

}

（4）UI实现

控制台命令行客户端已实现全部业务流程，支持实时交互和命令提示。

Qt图形化客户端已完成项目结构和部分界面原型，支持商品浏览、购物车、订单管理等基础功能，后续将继续完善界面美化和交互细节。

// 主窗口 MainWindow.h

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void onLoginSuccess();

void showProductView();

void showCartView();

void showOrderView();

private:

void setupUI();

NetworkManager \*networkManager;

QStackedWidget \*centralWidget;

LoginWindow \*loginWindow;

ProductView \*productView;

CartWindow \*cartWindow;

OrderWindow \*orderWindow;

};

// 商品列表界面 ProductView.h

class ProductView : public QWidget {

Q\_OBJECT

public:

ProductView(QWidget \*parent = nullptr);

void displayProducts(const QList<Product> &products);

signals:

void addToCart(int productId);

};

// 购物车界面 CartWindow.h

class CartWindow : public QWidget {

Q\_OBJECT

public:

CartWindow(QWidget \*parent = nullptr);

void displayCart(const QList<CartItem> &items);

signals:

void checkout();

};

// Qt信号槽与数据绑定

void ProductView::displayProducts(const QList<Product> &products) {

productListWidget->clear();

for (const auto &product : products) {

QListWidgetItem \*item = new QListWidgetItem(product.name);

item->setData(Qt::UserRole, product.id);

productListWidget->addItem(item);

}

}

void CartWindow::displayCart(const QList<CartItem> &items) {

cartListWidget->clear();

for (const auto &item : items) {

QListWidgetItem \*listItem = new QListWidgetItem(item.productName + " x" + QString::number(item.quantity));

cartListWidget->addItem(listItem);

}

}

// 命令行客户端交互

void showCommandMenu() {

std::cout << "1. 浏览商品\\n2. 加入购物车\\n3. 查看购物车\\n4. 提交订单\\n5. 退出\\n";

int choice;

std::cin >> choice;

// 根据choice调用对应业务逻辑

}

二、遇到的技术问题及解决方法

Qt客户端MSVC编译器兼容性问题

问题：Qt头文件要求 /Zc:\_\_cplusplus 编译选项，导致MSVC编译失败。

解决：在CMake中为MSVC添加 /Zc:\_\_cplusplus /permissive-，并尝试切换到MinGW工具链，最终成功编译。

Netty与C++后端集成（JNI）

问题：Java与C++数据结构映射复杂，Native方法参数校验易出错。

解决：统一协议格式，严格类型转换，增加错误日志和断言。

数据库并发与事务一致性

问题：高并发下订单、支付等操作易出现死锁。

解决：优化SQL事务处理，合理加锁，使用连接池。

促销策略灵活性不足

问题：复杂促销（满减、阶梯价）实现难度大。

解决：采用策略模式和反射机制，逐步扩展促销逻辑。

三、已完全掌握与困难部分

完全掌握部分

服务端核心业务逻辑（用户、商品、订单、购物车、促销、地址等）

MySQL数据库设计与操作

Netty网络通信与协议处理

控制台命令行客户端开发

CMake与MinGW/Qt项目构建

困难部分

Qt图形化界面复杂交互与美化

高级促销策略（满减、买N送M、阶梯价等）

复杂并发场景下的性能优化

各种业务逻辑在实际应用中的优化操作

四、进度偏差分析

设计时间已过半，整体进度约为75%，核心业务和数据库已基本完成，通信和命令行客户端稳定。

Qt图形化客户端进度略有滞后，界面美化和交互细节尚未完善。

各种业务逻辑在实际应用中仍有不如人意的问题，需后续重点推进优化。

五、本周拟解决问题及方案

Qt客户端界面完善

目标：完成商品浏览、购物车、订单管理等主要界面，提升用户体验。

方案：细化UI模块，优化布局与交互，增加主题切换和响应式设计。

促销策略扩展

目标：实现满减、买N送M、阶梯价等复杂促销。

方案：采用策略模式，抽象促销接口，逐步实现各类促销逻辑。

支付系统真实对接

目标：对接微信、支付宝等第三方支付平台，完善支付回调与安全加密。

方案：调研API，设计支付回调流程，实现加密与状态同步。

系统集成测试与性能优化

目标：进行多端集成测试，优化数据库和网络性能。

方案：编写自动化测试脚本，分析瓶颈，优化SQL和通信协议。

总结：  
本周重点推进Qt客户端UI开发和促销系统扩展，逐步完善支付和系统集成，争取下周实现核心功能的全面联调和性能优化。

更详细项目信息：[AshIgnis/JLU\_Emshop\_System](https://github.com/AshIgnis/JLU_Emshop_System)

55240425

屈熙宸