面向对象课程设计(荣誉课)实验报告·终

一、 需求分析

本程序的功能需求将从服务端(server)和客户端(client)进行细化分析。

1.1 客户端主要需求

1.1.1 客户端主要功能需求

- 登陆界面的实现

  在本功能中，在用户输入正确的用户和密码后可以进行成功登录，并在此时创建异步 socket 套接字建立于服务端的连接，并对之后的页面主操作页面进行操作与优化。如图 1 为登录页面：

图 1 登陆界面样式

  页面采用了 QSS 进行优化，添加页面，输入框，按钮的圆角优化和样式改变，去掉了原有的页面控件，添加了自定义关闭最小化控件，并对页面进行了自定义拖拽设置，使页面更加美观，更加符合大众使用。

- 注册界面的实现

  在登录界面的基础上我们实现了注册界面，在点击注册账号后，可以跳转进入注册账号页面，实现新账号的注册，可以在此时随意返回登陆界面和关闭，在注册时，用户名和密码为必填项，id 将自动分配，注册成功后，用户会返回登陆界面进行自动登录。在页面创建过程中，页面之间将共享 socket，通

过自定义 socket 编码识别注册信息，注册成功会自动跳转。下面是注册界面：

图 2 注册页面

在 UI 上，本程序也使用了 QSS 页面优化，添加了按钮触控效果，其他与登录窗口大致相同。

- Client 端主界面实现

  Client 端的主要功能为购物，因此所有的功能围绕着购物进行。具体功能有：

  1. 轮播图跳转功能

     轮播图采用的是 QGraphicsObject 中其可以较方便变换伸缩的特性，加入了定时器的播放与按钮组对每一块图片的索引功能，当点击对应按钮的时候可以实现对应编号页面的跳转，也可以通过左右两边的前进后退按钮可以实现按顺序播放，并支持图片的按压切换。

     轮播图在鼠标右击的情况下会跳转到轮播图对应的功能页面，在剩下的三页中，索引了网页链接，包括了作者的 github 仓库，blog，gitee 仓库地址，方便与作者进行交流学习。

  2. 推荐商品按压跳转功能

     推荐商品按压为 Qt 中信号与槽的运用，在触发点击后可以进行接下来的一系槽函数的操作，这项功能在之后的多线程之间传输数据与多页面之间传输数据信号有着至关重要的作用，推荐商品项目栏由三层 UI

界面组成，第一层为单个商品推荐图块，显示商品的图片信息和价格信息。第二层为四个一组的容器页面，将四个商品组块整合进行横向排列，第三层为主页面的 List 页面，在此页面上可以添加多组推荐容器，形成 List 结构，右面的滚动条可以进行拖动，商品过多本程序只进行八种推荐商品的展示。

3. 促销活动商品跳转功能

促销商品跳转与上条功能相似，对其商品图块进行了复用，跳转功能相同，在第二层上采用了水平滚动界面的形式，通过对单个 Widget 强制塞入多个图块，形成了加长的 Widget，可以通过自定义前进后退控件进行左右滚动，浏览打折商品。水平滚动界面设置了两行。

4. 购物车多选式的结算和删除

在购物车方面，设置了其他微商系统常使用的多选化管理。可以多项选中进行统一操作，进行选定式的结算和删除，方便而快捷。

5. 历史记录多选式的退货和删除

历史记录与购物车相似，历史记录也记录了商品的图片，编号，名称，购买数量，价钱，种类，可以多选进行退货和删除，更加方便快捷。

6. 购物车销售商品的按压跳转功能

仿照了现生活中常用的微商形式，在购物车中可以通过双击进行跳转，查看购买商品的详细信息，并对购物车进行补充和修改。购物车统一购买后，会直接加入到销售记录栏。

7. 历史销售记录的按压跳转功能

仿照了现生活中常用的微商形式，在消费记录中可以通过双击进行跳转，查看购买商品的详细信息，并在此基础上引入退货和删除订单功能。并且在退货后，该条信息自动加入到购物车中，方便客户再次选择。

8. 跳转和搜索记录的保存功能

在进行商品选择后，会跳转到统一的搜索记录中，在搜索记录中可以进行最终的购买和加入购物车的选择，也可以选择对搜索记录的删除。

9. 商品的购买功能和添加购物车功能

在跳转搜索后，得到了购买页面，每一个购买页面可以进行购买操

作，包括选择商品的种类，商品的个数，然后选择添加到购物车或者是直接结算，整个流程不需要键盘的参与，方便快捷。

10. 商品的模糊搜索功能

可以对商品进行及时的模糊搜索功能，通过搜索，可以及时给予搜索提示，在搜索完毕后，可以通过点击，直接选择搜索的商品进行跳转并购买，购买流程为上一条目实现的功能。实现原理为：发送信号给客户端进行实时的数据传输，实时返回查询结果。并通过加工，生成列表的每一项，显示在页面上并进行信号与槽的绑定，进行数据的传输操作。

11. 页面的风格转换

在美观设计上，完成了项目的要求，可以对整个项目进行风格的转换，通过重新加载 QSS 格式样式进行统一的修改，可以改变局部特征颜色，具体有六种风格。

12. 用户信息修改功能

用户信息在右上角的设置中可以被打开，打开后鼠标离开页面页面将会被隐藏，在该页面上有用户的基本信息，在锁定的情况下，其信息不允许修改，在开锁的情况下可以修改信息，修改信息时，可以通过点击界面上用户的头像进行修改，修改后进行保存，可以实时更新用户的数据。

13. 操作完成提示功能

在每一个关键操作完成的时候，比如购买，结算，删除，添加购物车，修改信息等操作，都会在屏幕上出现操作提示符，供使用者知晓其操作的成功性。

14. 客服聊天功能

在用户登录后，可以通过设计的对话框进行聊天交流，通过异步 socket 进行实时通信，且在界面上进行了对话框气泡界面美化。使得界面显得更加美观，且聊天的过程中还会有隔断的时间提醒。

15. 版本信息显示功能

在页面的上半部分有着关于两个字，鼠标点击后会蹦出本程序的版本，在后续的优化中，可以对本程序进行适当的优化和界面动画的操作。

16. 其他页面美观设计

在界面美观上，还设计了 tab 选中后的字体放大和选中下划线的显在左侧也加入了列表形式的索引美化，可以通过点击左侧列表的图标进行每个功能模块的选择。

这十六个具体功能将在之后的 UI 展示中统一进行。

1.1.2 性能需求

- 本客户执行操作时，采取的是尽量少申请数据库操作指令，在迫不得已时才与数据库进行间接交换，保证了在高并发模式下，客户端一定的高性能。
- 在数据传输的过程中，采用的是 url 进行数据传输，通过 setStyleSheet 进行图片的展示，因此 url 相对于传输整个数据来说，其效率要大大高于整个图片的传输。
- 其他功能中，均是局部访问，数据均为局部操作，因此，操作量较小，没有进行统一优化。

1.2 服务端主要需求

1.2.1 服务端主要功能需求

由于服务端没有登陆界面的问题，因此直接对主页面进行探讨。可以具体分为五大功能，商品综合操作，用户交流反馈，活动商品的添加和删除，数据综合统计，用户信息的综合处理。

- 商品综合操作

  本模块中涉及了三项操作：新增修改商品，查找商品，删除商品。

  - 查找商品

    同客户端查找商品类似，都是通过数据库模糊查询进行查找，并存在下拉框进行查找提示，达到模糊查询的效果。查询完毕后可以得到相应的商品信息，双击后可以跳转到新增修改商品页面进行统一操作。

  - 删除商品

    商品的删除于查找商品功能类似，在搜索过后可以进行删除操作，其双击没有办法进行跳转，可以通过鼠标右击后出现的删除按钮进行删除，删除后，动态更新数据库中的内容，并且在之后的搜索中不会再出现这项商品。

- 新增、修改商品

  该页面主要是对整个商品信息的完全更新和添加功能。支持对整个商品的图片，姓名，数量，种类的修改和添加。图片通过 url 进行展示与传输。点击添加按钮后，进行统一操作，后台的实现为：对数据库通过商品编号查找一遍后，若发现有同编号的商品，则更新该商品，若发现没有该商品，则进行添加商品操作。

具体界面实现将放到后面的 UI 设计进行具体叙述。

- 用户交流反馈

客户端支持多用户聊天室操作。在每一个账户登陆后，都实时创建一个与之相对应的客户聊天框，通过该聊天框可以对该客户发送消息，发送消息之后可以进行消息的接受和回复。且在此过程中，服务端可以对每个单独的聊天进行删除操作，删除之后，当对面发送来消息时，又会通过连接情况自动生成相应的聊天对话框，方便双方的交流通信。当客户端断开连接时，会产生通信提示。

- 活动商品的添加与删除

活动商品的添加包括了打折优惠的添加和对打折优惠的删除操作，打折优惠的删除操作包括搜索商品，商品选中进行打折处理，处理成功后，加入到打折队列中，打折队列中可以自由删除不想打折的商品。且添加了选中的动画效果。

- 数据综合统计

在第四个模块，允许服务商进行数据统计和处理，可以统计单个人和所有人的购买情况，可以统计单个货物和全部货物的销售情况，将两个时间段的时间大致分为十二个月，以每月的数量进行集中统计，且时间段的开始和结束都可以选择，最终表示形式为柱状图与折线图的混杂形式和饼状图的表现形式。且可以手动刷新表格。在数据可视化旁边，也存在着数据表的陈列，方便从数据和单个月进行综合考察。

- 用户信息综合处理

用户综合信息处理是服务器端后台的自动操作，管理者是无法进行操作的，目的是为了响应客户端数据传输请求，响应客户端对数据的修改和操作，

响应客户端的支付功能。主要包括客户端页面初始化，客户端与服务端之间的购买请求，搜索请求，退货请求，退货请求。且在响应后，会在服务端进行直接提示，说明购买成功。

1.2.2 服务端主要性能要求

- 高并发模式下的多线程使用

  在考虑到秒杀功能的实现时，可能会遇到每个用户端多次申请同请求，且多个用户同时申请，若使用单线程，则会产生速度过慢而导致的多服务器卡死问题，因此在此基础上我们需要多线程的帮助，才能在高并发模式下，提高信息接收的速率。

  本程序在异步 socket 连接上采用了多线程的处理方式，并考虑到了高性能的问题，即每一个 socket 分配一个线程，在此条件下，多个 socket 将产生多个线程，且线程内部 实质性工作在大部分时刻都处于空闲状态，因此过多的线程运行会导致主机的内存占用比升高，带不动更多的服务器，因此本程序采用了 线程管理的方式，将线程集中起来管理，形成线程池，负责控制线程的产生和释放，且一个线程负责控制多个 socket 的运行，即防止了多线程产生的高性能问题，也进一步实现了高并发。

  具体步骤将在下文进行逐一叙述。

二、 系统设计(内部逻辑)

2.1 数据库设计

数据库中包含了四个大表，包括了 product，client，product_type，order 四个表。其中有部分信息具有关联型，即含有外键，每个表下具有不同的属性和功能，包括了用户，商品，商品类型和销售记录四种类型，其中销售记录中蕴含着三种不同状态的信息，包含了销售记录，购物车记录，和客户端删除的客户销售记录，通过标记进行区分辨别，下图是数据库图表的 E-R 图，其中每个模块代表着每个表的键属性，连线表示每个键属性与其他表之间的联系：

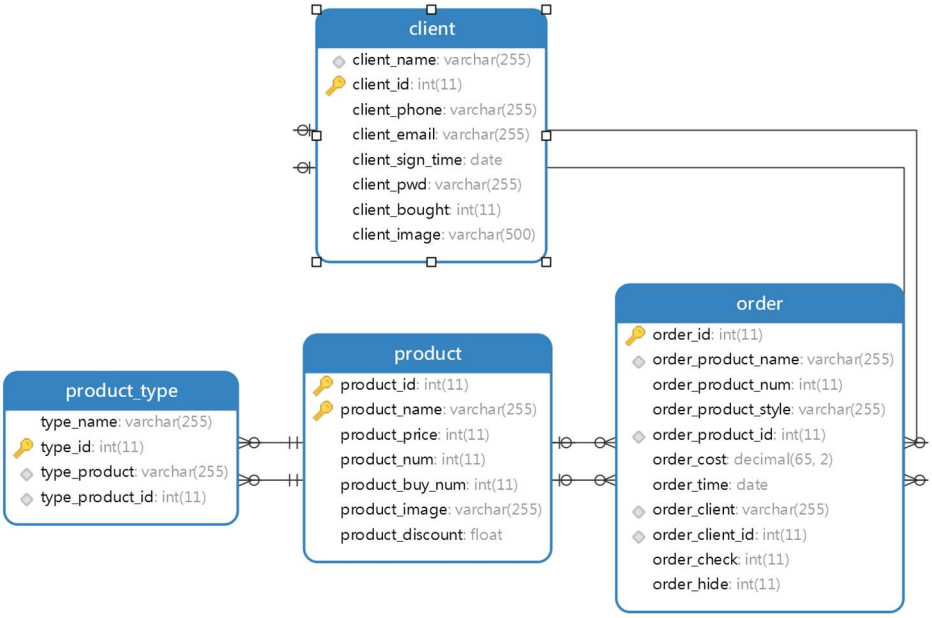

图 3 数据库的表组成

在 Client 表中，设置 client_id 作为主键，其中 client_id 保持数据库内部自增，在删除特定 id 的对象后，不会有重复的 id 继续生成。且该表中存在必须填写的项目：如姓名和密码是必须填写的，其他选填，购买的数量是自动统计的。

在 product 表中，设置 product_id 作为主键，不可重复，且保持自增，其他属性中：名称，价格，为不可缺少的属性，其他属性可以选填。

在 order 表中，其属性较多，且将 client 表，product 其中的属性作为外键，遵循当货物与顾客被删除时，存在 order 的顾客与货物 id 置空的操作，其中所有的属性均为非空，由于 order 是由订单产生，得到了最后的结果一般为完整信息。

在 product_type 表中，其与 product 关联较为紧密，每一个 product 可以对应多个属性和样式，通过该表进行关联索引即可得到每个商品其所有的属性，便于客户选择。

本程序的数据库设计，较为简单，为了避免复杂的多表联查，采用的是**动态 sql** 语句进行综合查询，使得一条查询语句可以进行重复使用，因此更加方便快捷。

2.2 UI 设计

2.2.1 客户端主页面

前面已经展示了登录窗口的界面设计，现在展示在登录成功后的客户端主页面。以及各项功能。

- 客户端推荐页面：

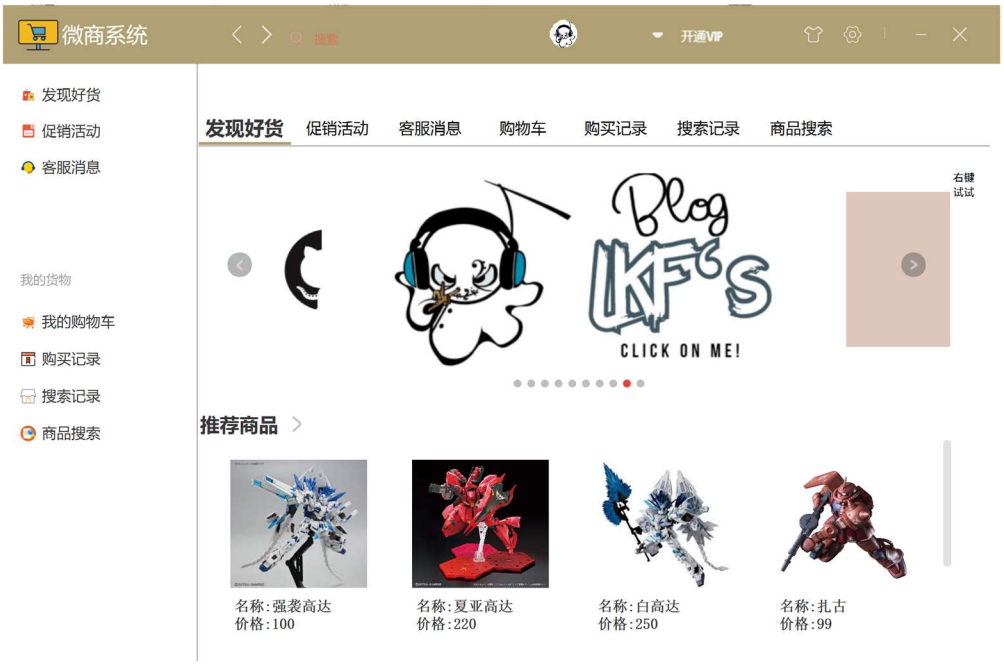

图 4 推荐商品界面

在这个界面上我们可以进行多功能化的跳转，整个界面采用白色为主基调，做到尽量简约和操作简便的效果。上栏是微商标志，左右箭头循环跳转，头像，关于，VIP 设置(未实现)，换肤色，修改用户的基本信息，最小化，关闭的功能。

- 促销互动界面

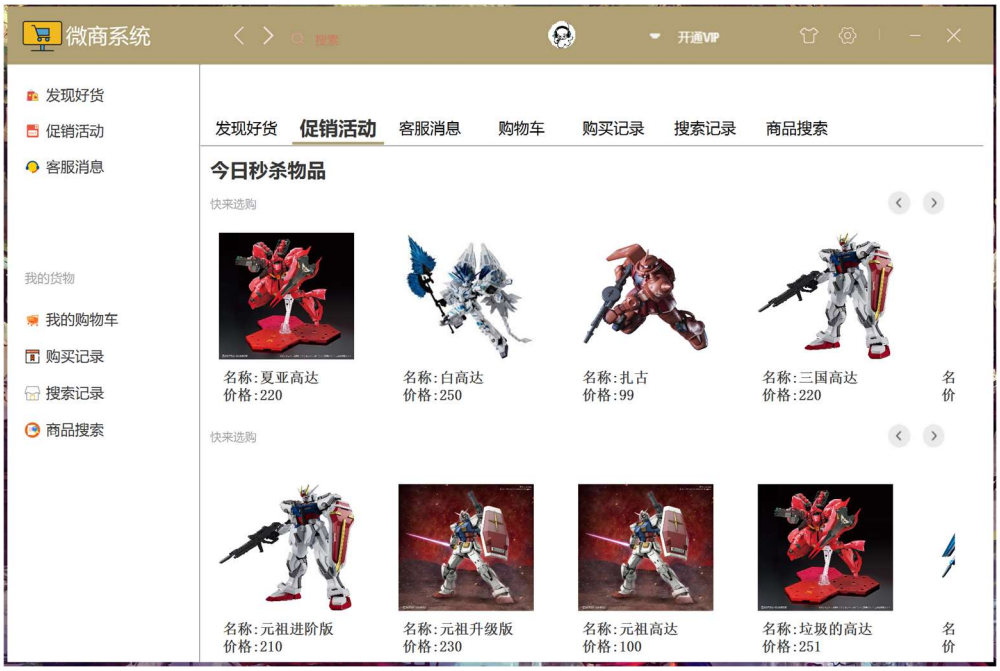

图 5 促销活动界面

促销活动大部分为打折物品，当打折物品没有那么多的时候将非打折物品进行

补充。

- 客服消息界面

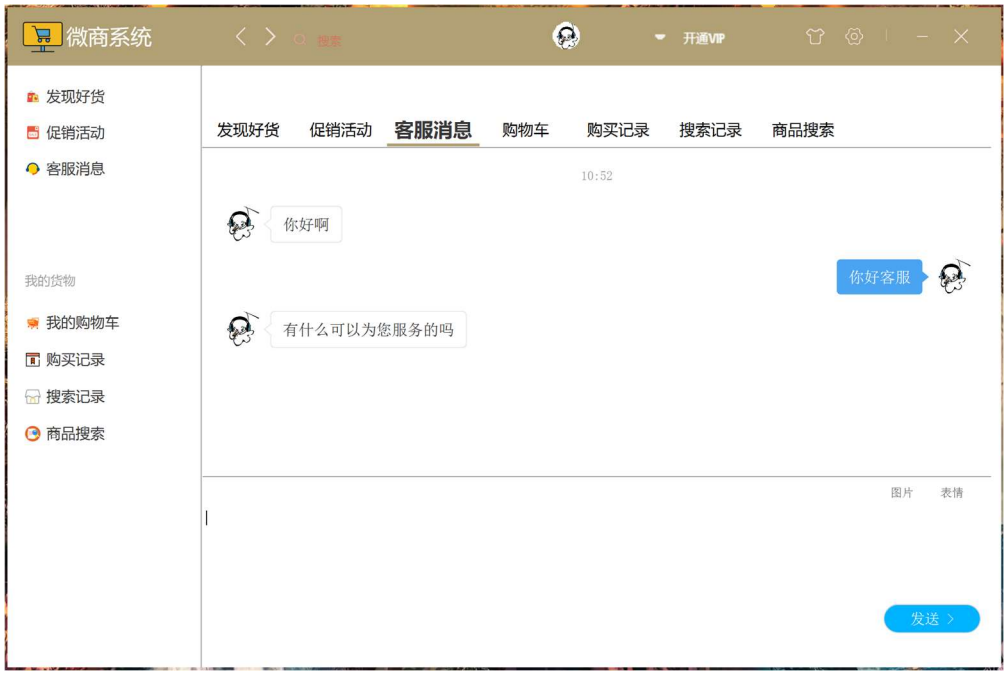

图 6 客服消息界面

客户消息界面主要负责与客服进行通信，客服端是多聊天窗口程序。

- 购物车界面：

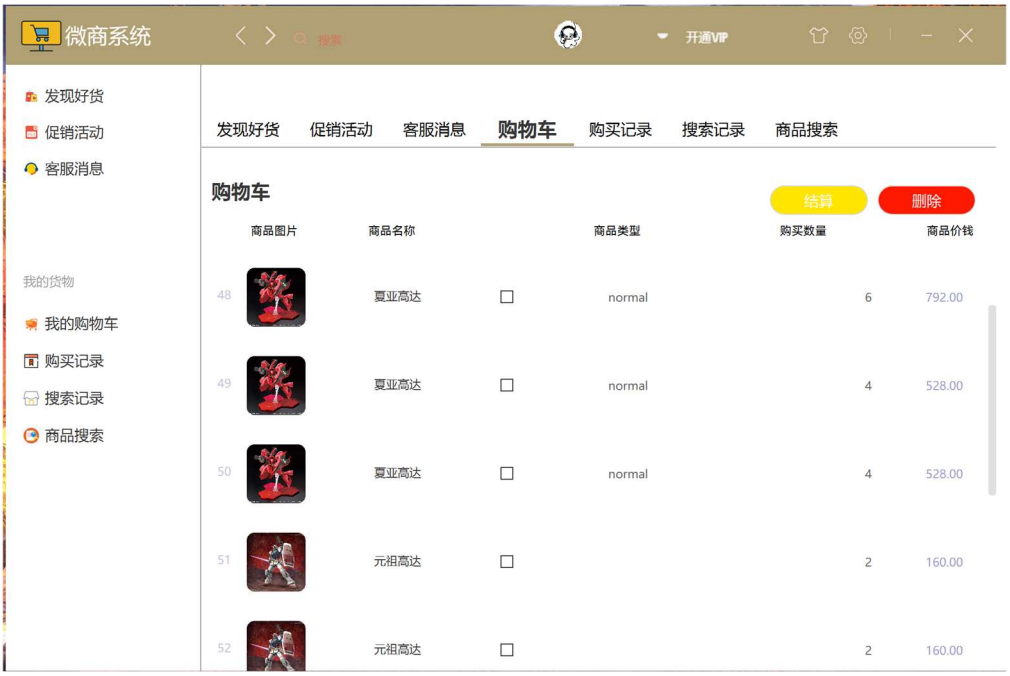

图 7 购物车界面

主要负责购物车的选购，统一结算和删除功能。

- 购买记录(历史信息)功能

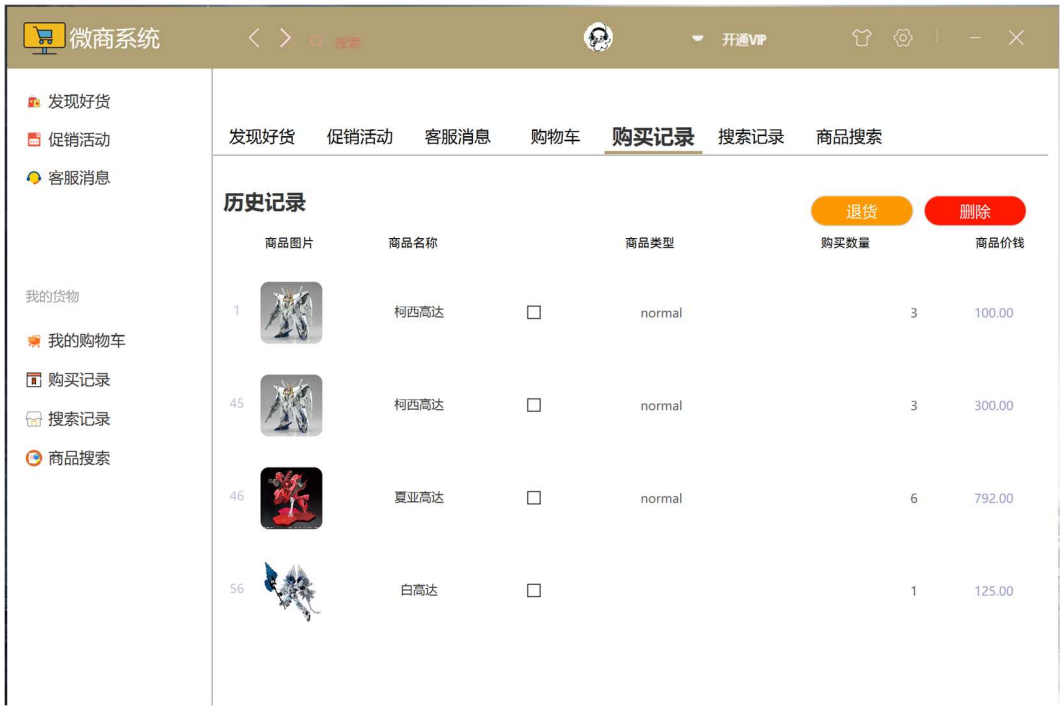

图 8 购买记录方面

与购物车相似可以进行退货功能和删除功能。

- 搜索记录显示

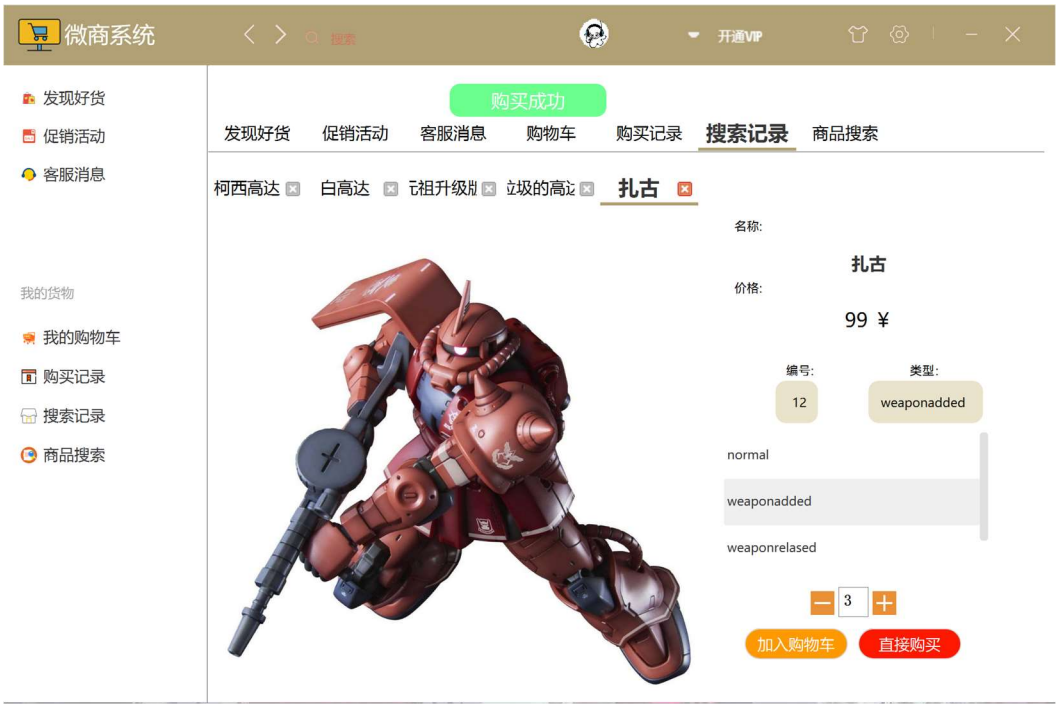

图 9 搜索记录与购买界面

可以进行单商品的单独购买和记录保存功能，可以删除搜索记录，或是一直保

留。

- 商品搜索功能

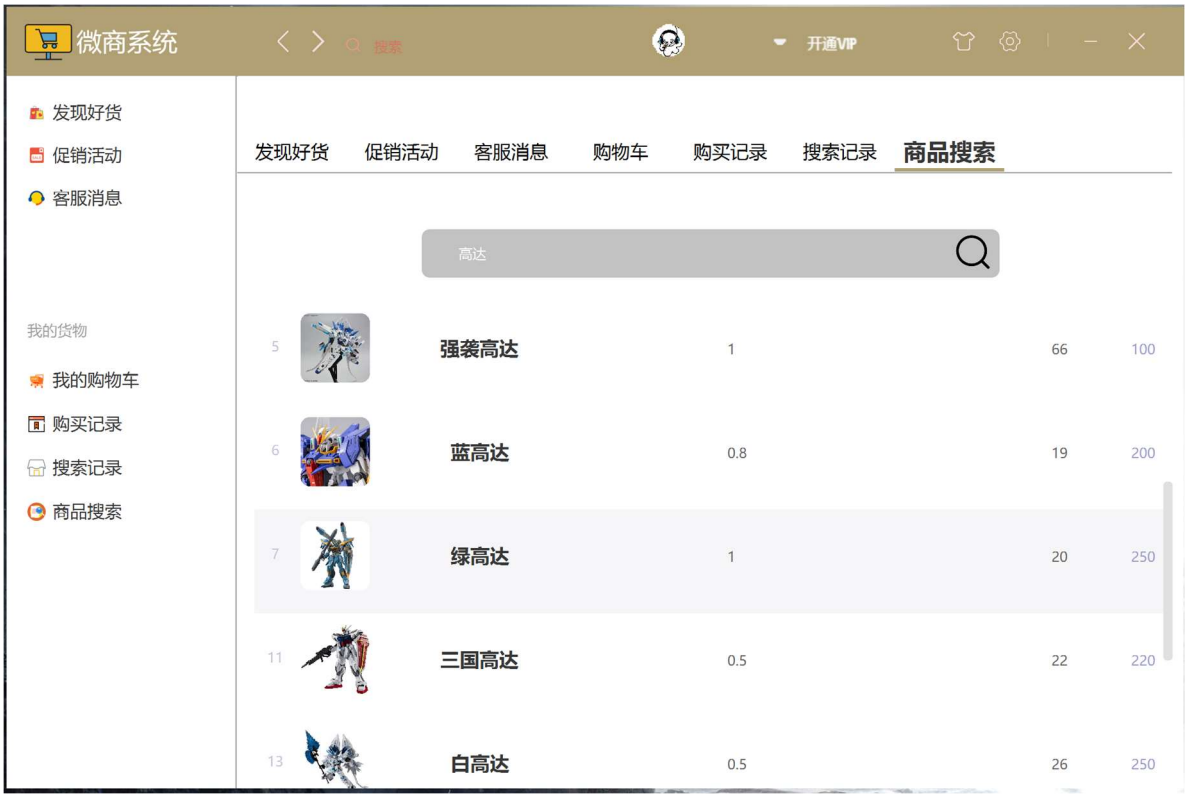

图 10 商品搜索功能

- 换肤功能和版本信息功能

图 11 版本信息显示

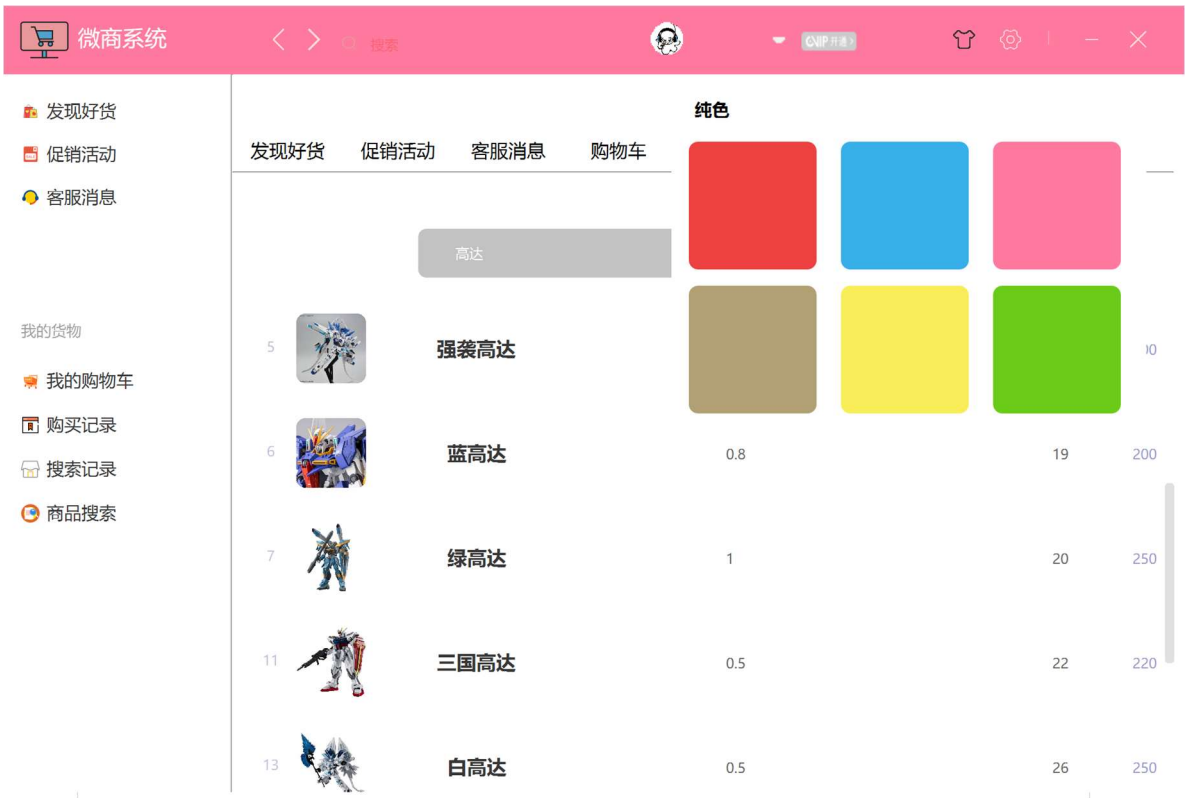

图 12 换肤功能

这两项是对客户端的补充和美化，可以选择六种颜色，并且查看当前版本信息。

- 用户信息修改功能

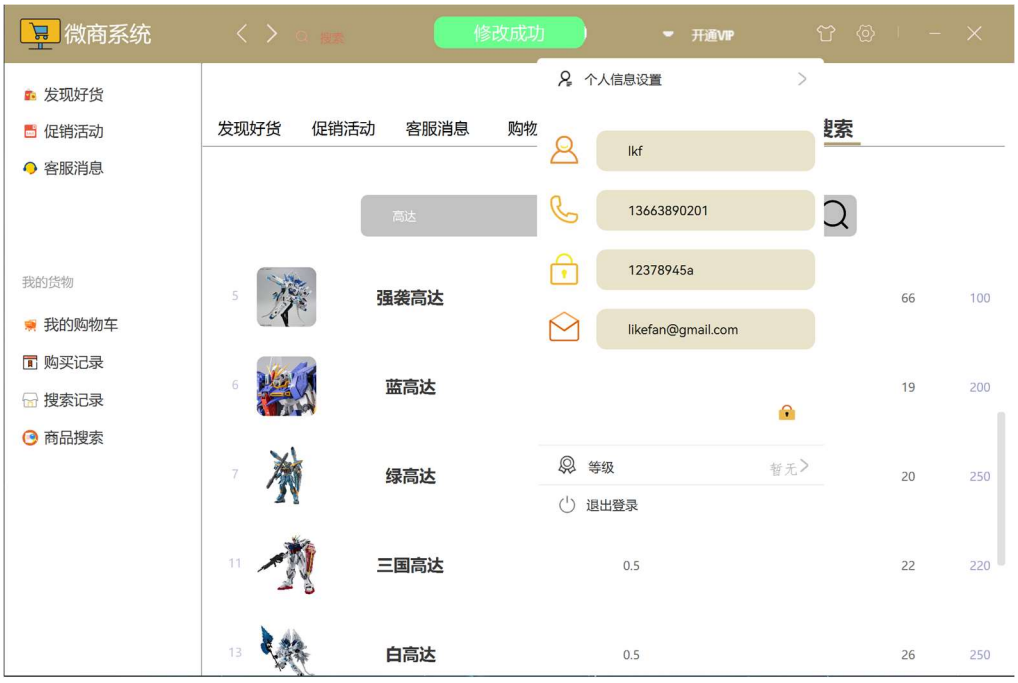

图 13 用户信息修改功能

右下角的锁打开前，所有信息被锁定，锁打开后，整个用户信息可以开始修改编辑，由于用户姓名是非常关键的信息，所以不给予修改权限。

修改完毕后关锁保存，上方在接受服务端发来的更新成功后，弹出提示。

2.2.2 服务端 UI 设计和实现

- 货物综合处理
  - 货物的查询界面：

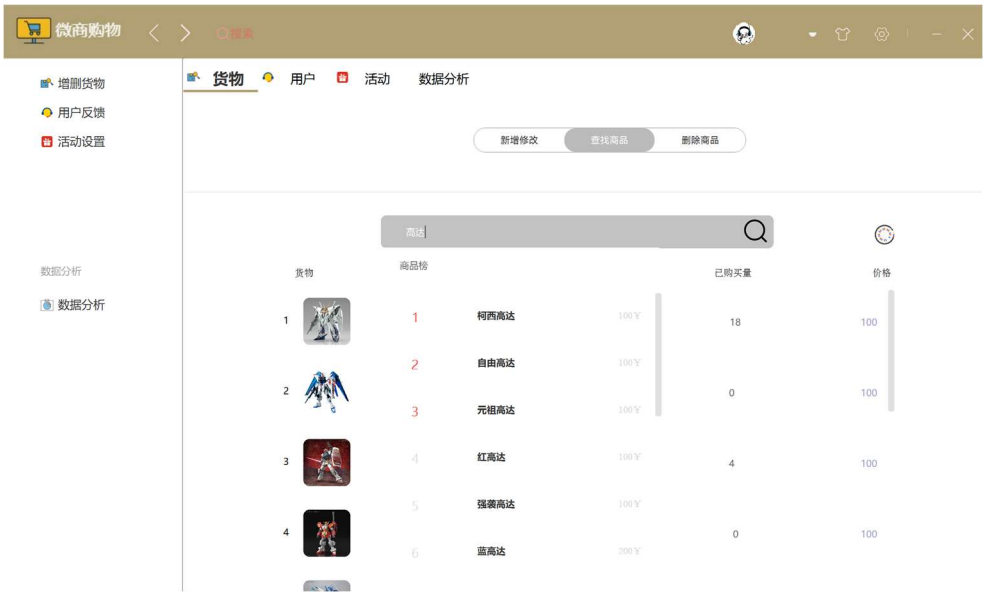

图 14 货物的查询

通过下拉框可以给予搜索提示，然后点击放大镜按钮后，可以进行搜索。

  - 删除界面：

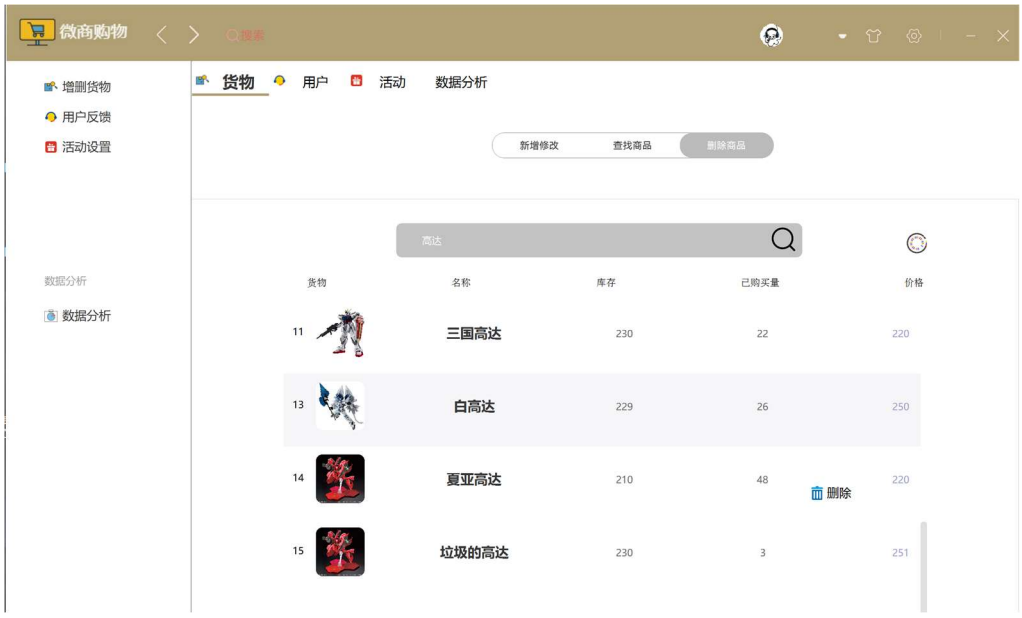

图 15 删除界面

可以通过搜索对商品进行删除，右键点击商品，会出现删除效果。

- 商品添加功能

图 16 商品添加界面

在本页面中，可以进行商品的添加和修改，并在点击添加后，弹出添加成功操作符。

- 与用户多人聊天功能

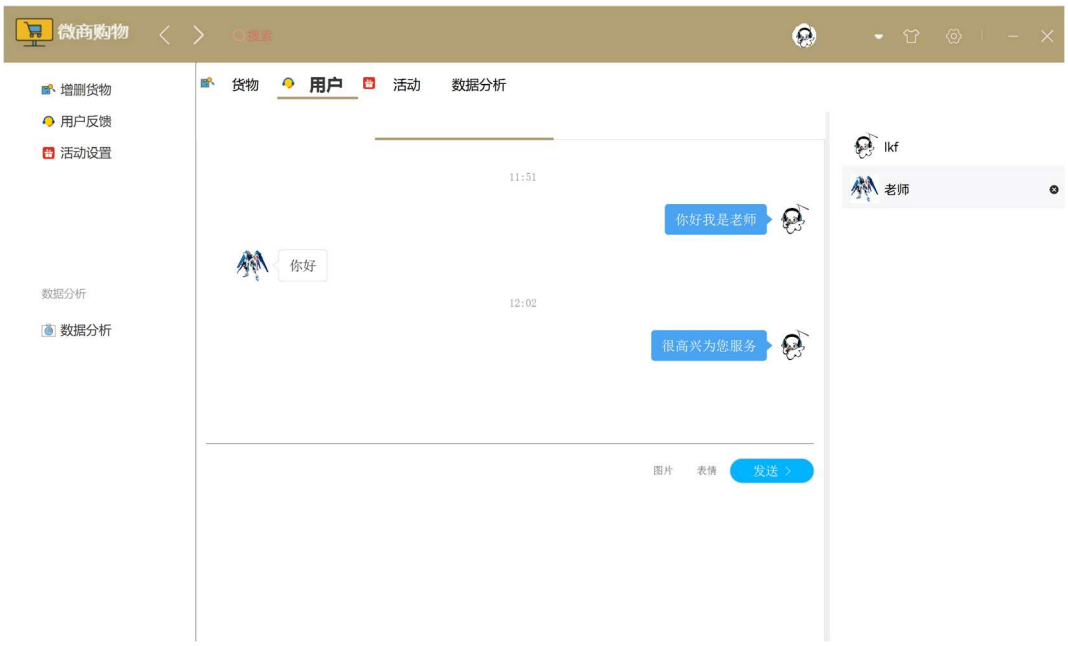

图 17 多人分时聊天功能

一个客服可以转接多个用户，与每个用户聊天，在聊天过程中，可以任意的删除在线或不在线的用户聊天框，当该用户登陆后或者重发消息后，又会进行聊天室的创建。

- 活动功能

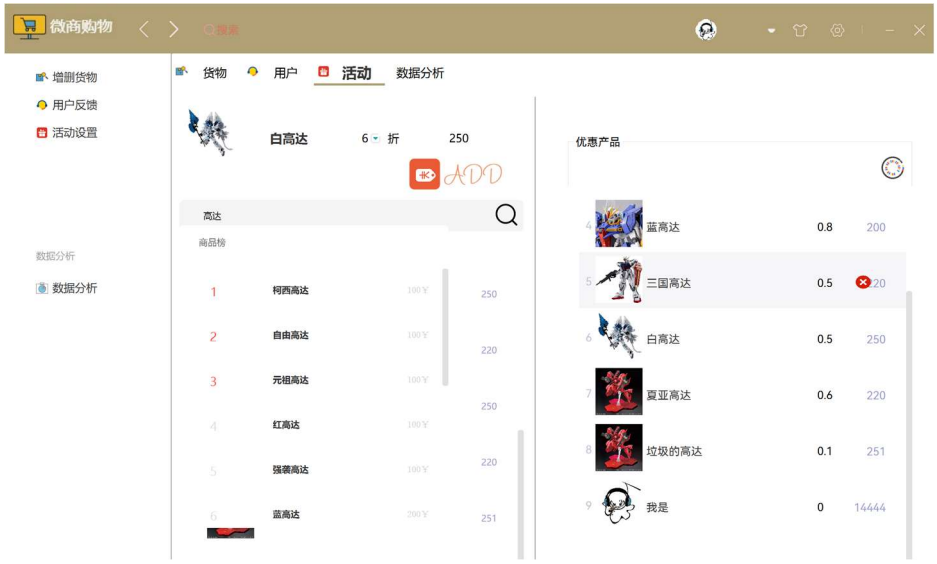

图 18 活动打折添加界面

在本页面可以自由添加活动的力度，及折数，然后对应着显示到优惠产品栏中。可以在优惠栏中进行随意的删除。然后点击右上角的刷新，会刷新优惠产品栏中的所有商品信息。

- 数据综合分析模块
  - 表格综合信息如下：

图 19 数据分析表格形式

在本页面可以进行表格和页面的拖拽，实现最大化的阅读体验。

- 图表分析功能

图 20 微商购物的数据分析功能

在此页面上可以通过选择分析对象，分析特定查找对象，分析分析全部对象，选择分析区间，进行分析，然后根据数据的综合处理自动生成图表，显示在右侧，且右侧的图标可以拖动以达到图表的放大和缩小功能。

2.2.3 流程设计

下面将展示整个程序的设计流程和工作流程。

- 客户端功能流程
  - 主干功能流程

    主干分为登录，注册，进行数据传输，进入到整个页面等功能，在此基础上获得多个子流程，这些子流程将在之后分开叙述，现在将展示主流程：

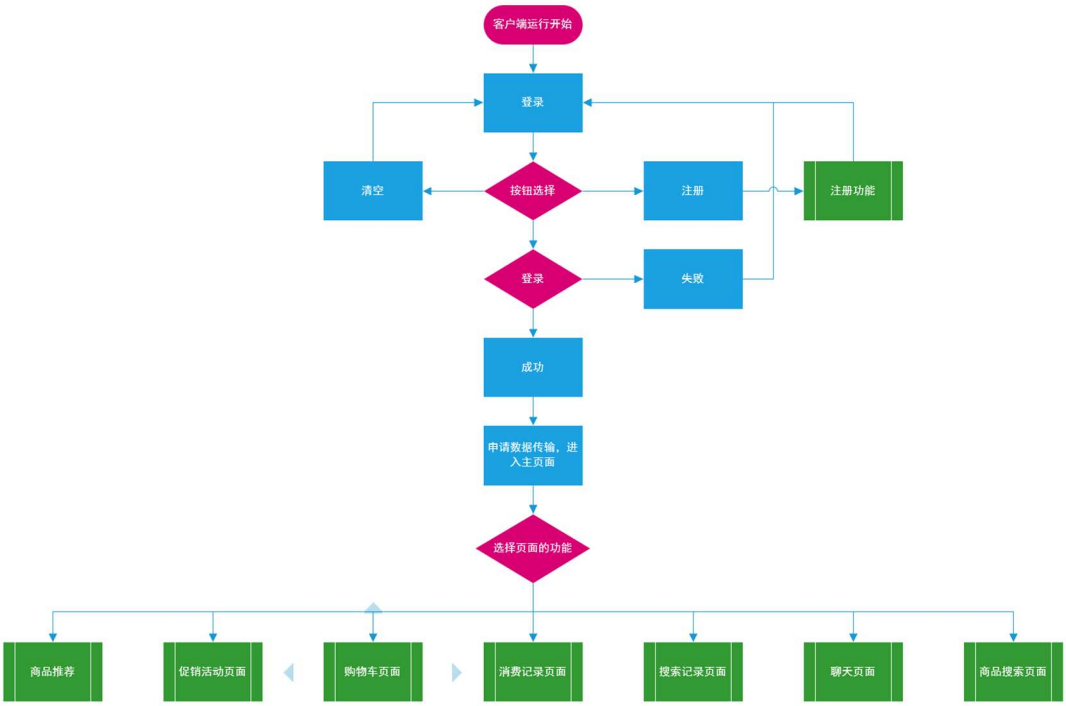

图 21 主干流程图

- 子功能全部流程图

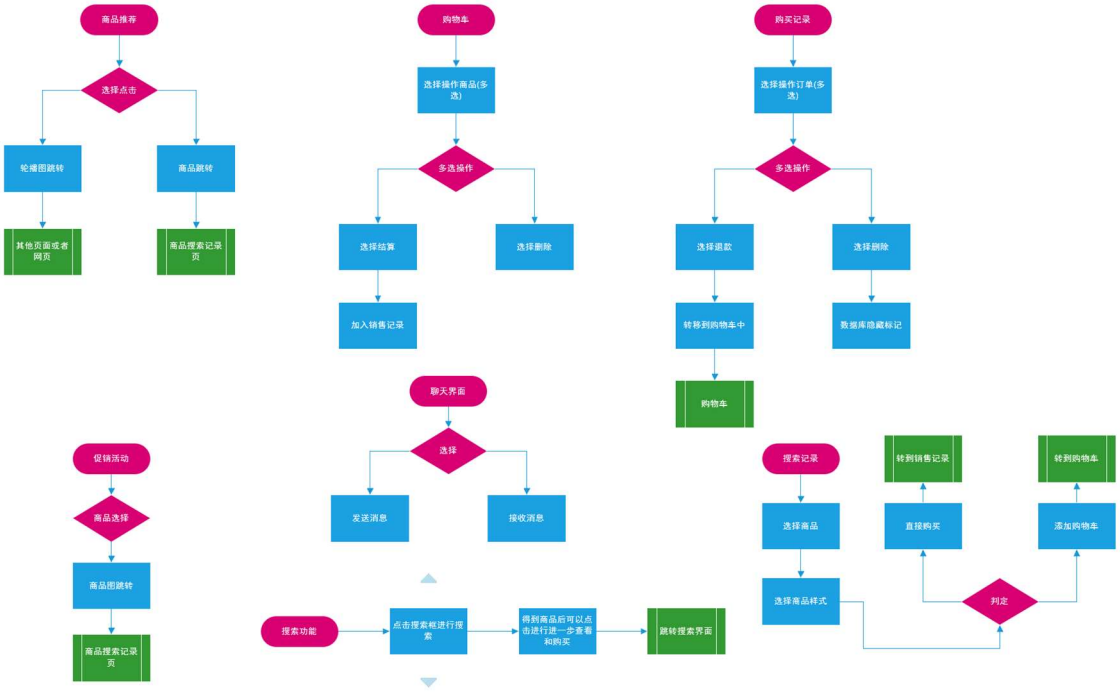

图 22 子程序流程图

- 服务端流程图
  - 服务端主干流程图

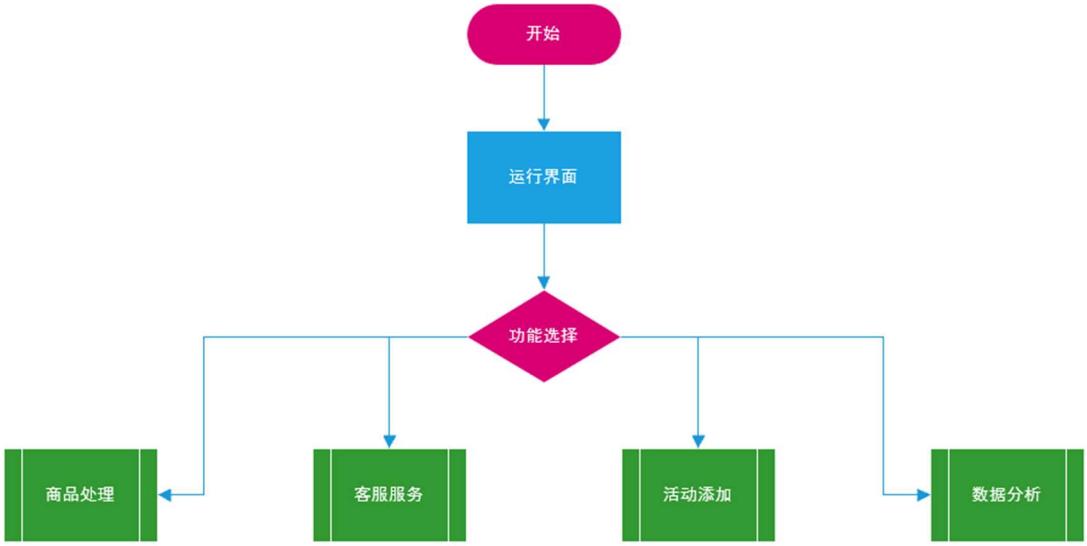

图 23 客户端流程图

■ 服务端子流程图

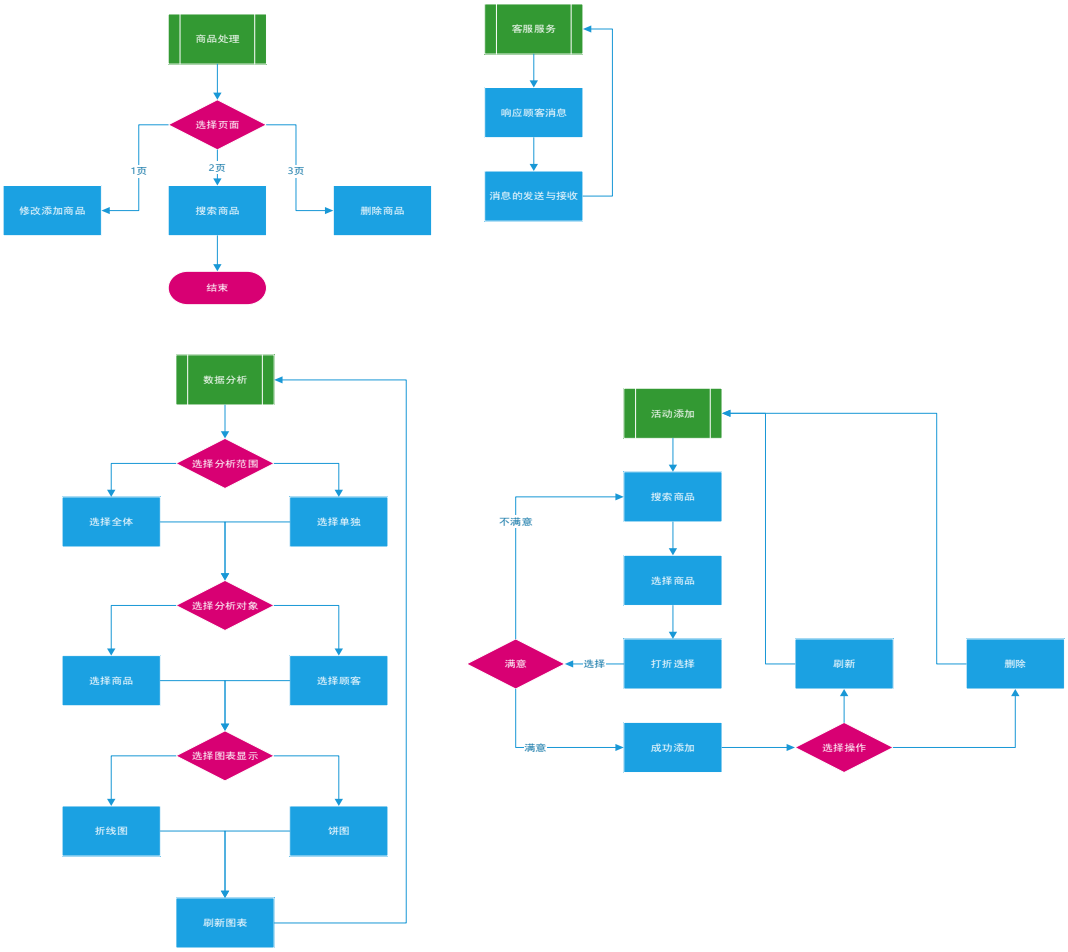

图 24 服务端子流程图

2.2.4 类设计以及类之间关系

持久层类与接口之间的关系：

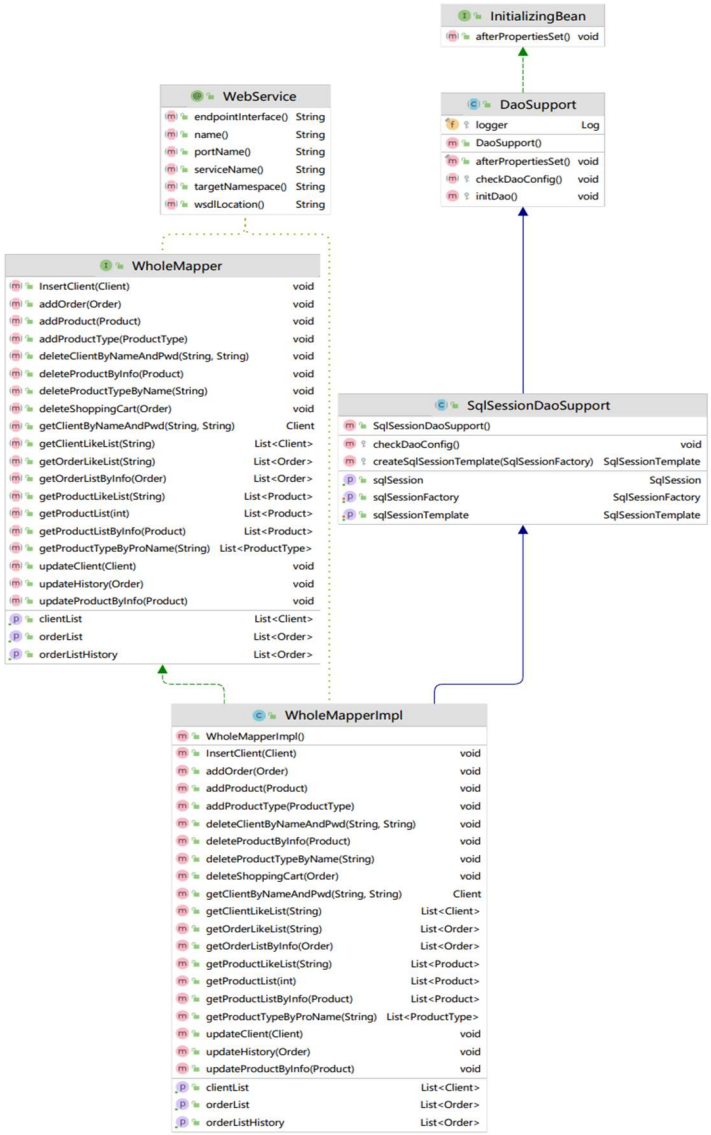

图 25 持久层接口与实现类之间的关系

接下来显示的是 spring 框架中注入的 bean 之间的关系图：

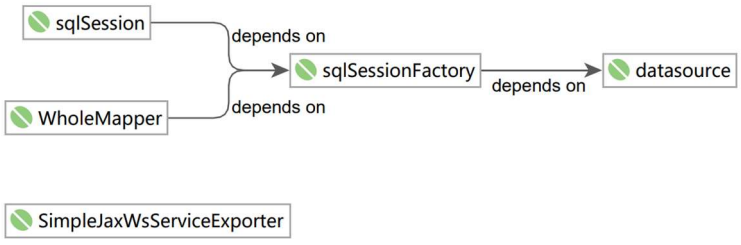

图 26 持久层 spring 框架中的 bean 注入结构

简单普通类和对应的操作接口类：

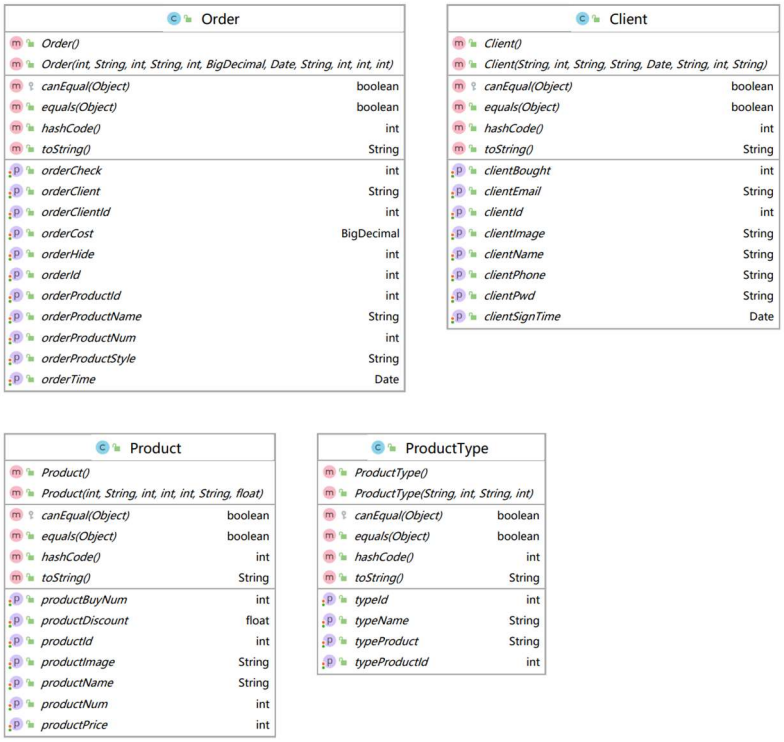

图 27 简单普通类

对应的操作接口类：

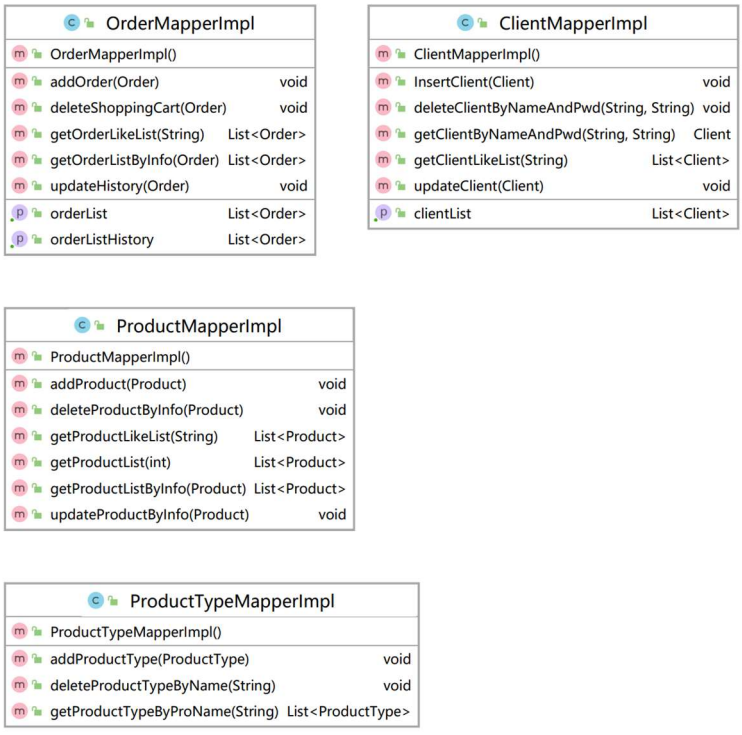

图 28 pojo 对应简单操作接口类

下面展示 C++端客户端的类之间关系：

图 29 Client 端的 UML 实现类

上图为大致的类之间关系图，其中有相当一部分的类间函数并没有被标出，因此并不是非常完整，代码阶段将对其进行具体说明。

下面是服务端的 C++类间说明 UML 图：

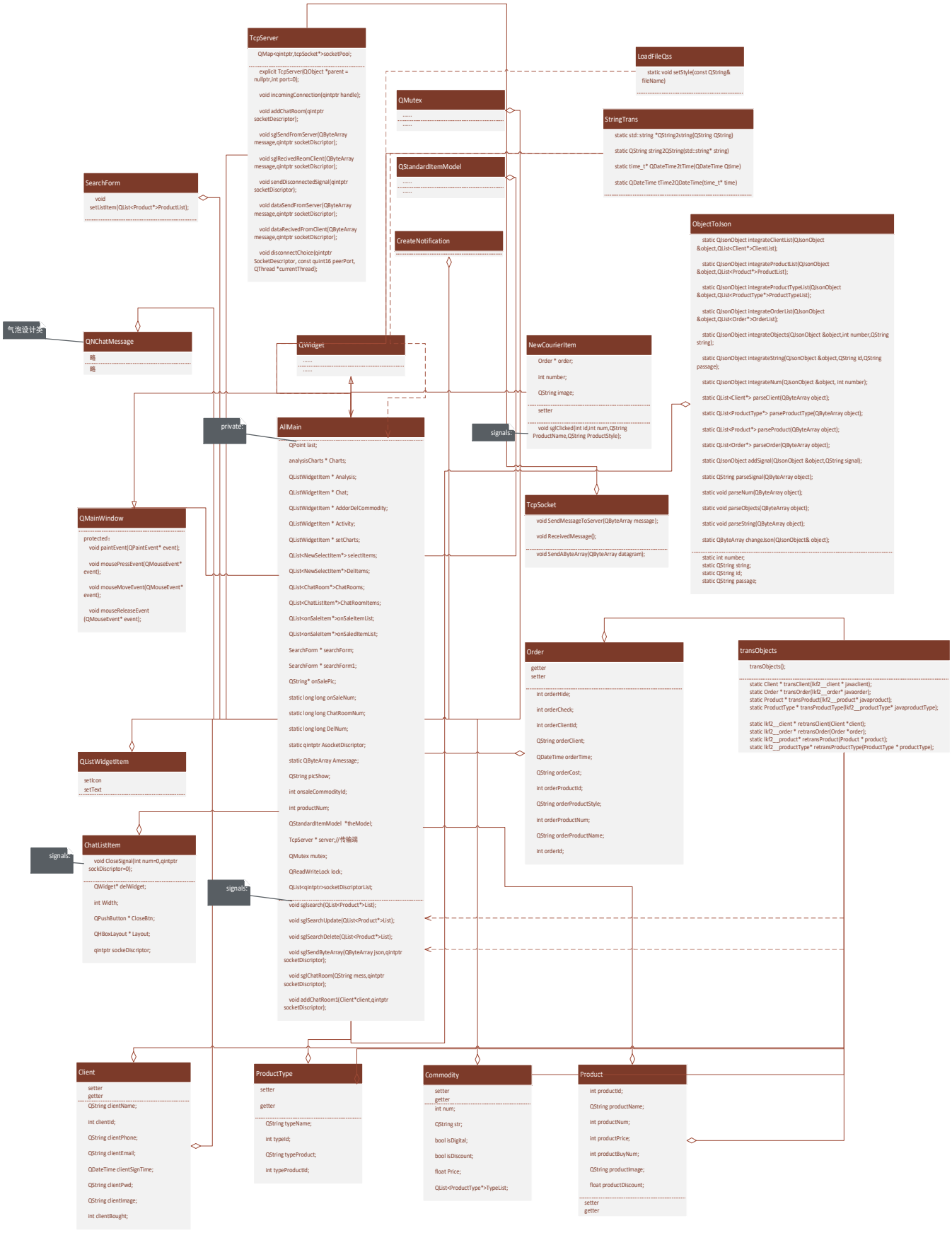
TcpServer
QMap<qintptr,tcpSocket*>socketPool;
explicit TcpServer(QObject *parent = nullptr,int port=0);
void incomingConnection(qintptr handle);
void addChatRoom(qintptr socketDescriptor);
void sglSendFromServer(QByteArray message,qintptr socketDiscriptor);
void sglRecivedReomClient(QByteArray message,qintptr socketDiscriptor);
void sendDisconnectedSignal(qintptr socketDiscriptor);
void dataSendFromServer(QByteArray message,qintptr socketDiscriptor);
void dataRecivedFromClient(QByteArray message,qintptr socketDiscriptor);
void disconnectChoice(qintptr SocketDescriptor, const quint16 peerPort, QThread *currentThread);
LoadFileQss
static void setStyle(const QString& fileName)
QMutex
StringTrans
static std::string *QString2string(QString QString)
static QString string2QString(std::string* string)
static time_t* QDateTime2tTime(QDateTime Qtime)
static QDateTime tTime2QDateTime(time_t* time)
QStandardItemModel
SearchForm
void setListItem(QList<Product*>ProductList);
CreateNotification
ObjectToJson
static QJsonObject integrateClientList(QJsonObject &object,QList<Client*>ClientList);
static QJsonObject integrateProductList(QJsonObject &object,QList<Product*>ProductList);
static QJsonObject integrateProductTypeList(QJsonObject &object,QList<ProductType*>ProductTypeList);
static QJsonObject integrateOrderList(QJsonObject &object,QList<Order*>OrderList);
static QJsonObject integrateObjects(QJsonObject &object,int number,QString string);
static QJsonObject integrateString(QJsonObject &object,QString id,QString passage);
static QJsonObject integrateNum(QJsonObject &object, int number);
static QList<Client*> parseClient(QByteArray object);
static QList<ProductType*> parseProductType(QByteArray object);
static QList<Product*> parseProduct(QByteArray object);
static QList<Order*> parseOrder(QByteArray object);
static QJsonObject addSignal(QJsonObject &object,QString signal);
static QString parseSignal(QByteArray object);
static void parseNum(QByteArray object);
static void parseObjects(QByteArray object);
static void parseString(QByteArray object);
static QByteArray changeJson(QJsonObject& object);
static int number;
static QString string;
static QString id;
static QString passage;
气泡设计类
QNChatMessage
略
略
QWidget
NewCourierItem
Order * order;
int number;
QString image;
setter
void sglClicked(int id,int num,QString ProductName,QString ProductStyle);
private:
AllMain
QPoint last;
analysisCharts * Charts;
QListWidgetItem * Analysis;
QListWidgetItem * Chat;
QListWidgetItem * AddorDelCommodity;
QListWidgetItem * Activity;
QListWidgetItem * setCharts;
QList<NewSelectItem*> selectItems;
QList<NewSelectItem*>DelItems;
QList<ChatRoom*>ChatRooms;
QList<ChatListItem*>ChatRoomItems;
QList<onSaleItem*>onSaleItemList;
QList<onSaleItem*>onSaleIdItemList;
SearchForm * searchForm;
SearchForm * searchForm1;
QString* onSalePic;
static long long onSaleNum;
static long long ChatRoomNum;
static long long DelNum;
static qintptr AsocketDiscriptor;
static QByteArray Amessage;
QString picShow;
int onsaleCommodityId;
int productNum;
QStandardItemModel *theModel;
TcpServer * server;//传输端
QMutex mutex;
QReadWriteLock lock;
QList<qintptr>socketDiscriptorList;
void sglsearch(QList<Product*>List);
void sglSearchUpdate(QList<Product*>List);
void sglSearchDelete(QList<Product*>List);
void sglSendByteArray(QByteArray json,qintptr socketDiscriptor);
void sglChatRoom(QString mess,qintptr socketDiscriptor);
void addChatRoom1(Client*client,qintptr socketDiscriptor);
signals:
TcpSocket
void SendMessageToServer(QByteArray message);
void ReceivedMessage();
void SendAByteArray(QByteArray datagram);
QMainWindow
protected:
void paintEvent(QPaintEvent* event);
void mousePressEvent(QMouseEvent* event);
void mouseMoveEvent(QMouseEvent* event);
void mouseReleaseEvent (QMouseEvent* event);
Order
getter
setter
int orderHide;
int orderCheck;
int orderClientId;
QString orderClient;
QDateTime orderTime;
QString orderCost;
int orderProductId;
QString orderProductStyle;
int orderProductNum;
QString orderProductName;
int orderId;
transObjects
transObjects();
static Client * transClient(lkf2__client * javaclient);
static Order * transOrder(lkf2__order* javaorder);
static Product * transProduct(lkf2__product* javaproduct);
static ProductType * transProductType(lkf2__productType* javaproductType);
static lkf2__client * retransClient(Client *client);
static lkf2__order * retransOrder(Order *order);
static lkf2__product* retransProduct(Product * product);
static lkf2__productType* retransProductType(ProductType * productType);
QListWidgetItem
setIcon
setText
ChatListItem
void CloseSignal(int num=0,qintptr sockDiscriptor=0);
QWidget* delWidget;
int Width;
QPushButton * CloseBtn;
QHBoxLayout * Layout;
qintptr sockeDiscriptor;
Client
setter
getter
QString clientName;
int clientId;
QString clientPhone;
QString clientEmail;
QDateTime clientSignTime;
QString clientPwd;
QString clientImage;
int clientBought;
ProductType
setter
getter
QString typeName;
int typeId;
QString typeProduct;
int typeProductId;
Commodity
setter
getter
int num;
QString str;
bool isDigital;
bool isDiscount;
float Price;
QList<ProductType*>TypeList;
Product
int productId;
QString productName;
int productNum;
int productPrice;
int productBuyNum;
QString productImage;
float productDiscount;
setter
getter

图 30 Server 端的 UML 实现

- 网络通信时序图

  下面是整个网络通信的时序图，大致展示了整个网络通信的实现步骤和实现的情况，包括了客户端的 socket 连接和信息发送与接收，服务端的 Tcpserver 端口的识别，socket 的创建，多线程的创建，以及通过异步 socket 的通信进行信息的发送：

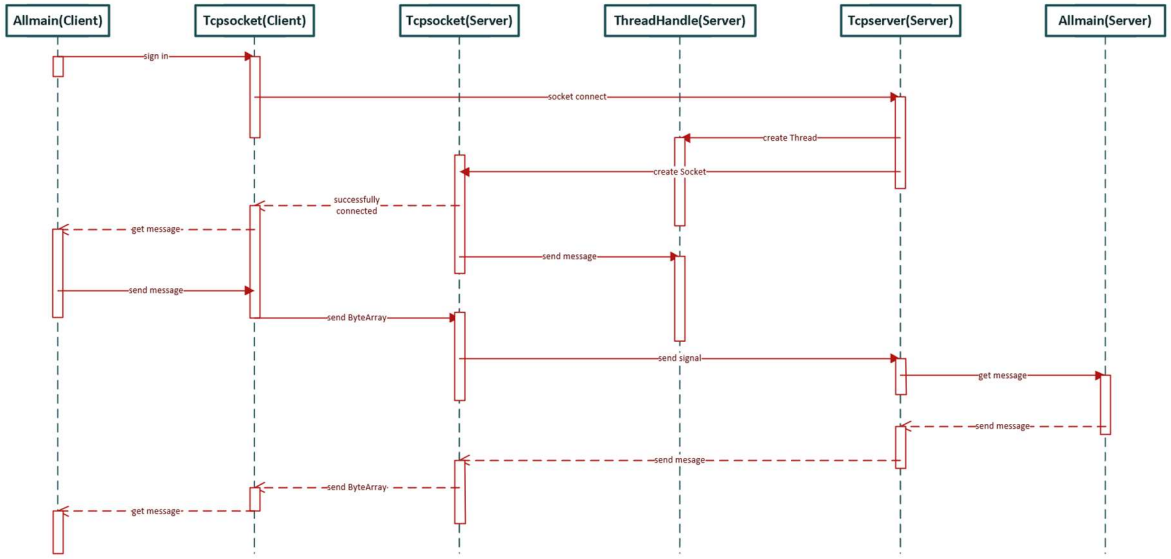

图 31 网络通信时序图

三、 系统实现

3.1 主要技术

本模块将主要探讨重点技术的实现和操作。

3.1.1 基于 Qt 框架的 C++可视化编程

本程序的所有可视化窗口均为基于 Qt 框架的编程，涉及了 Qt 中 QWidget 的各种变换，数据传递，界面设计，控件设计，并涉及到了界面美观技术等。具体包括：

- UI 可视化设计
- 混合方式 UI 设计
- 信号与槽机制的使用
- Qt 容器的使用，和 Qt 小控件的使用

- QString 的常用功能
- Qt 的布局与高级 Widget 的使用
- TabWidget 的使用
- Model 和 View 的使用
- 多窗体应用程序的设计
- 图片信息的处理
- 坐标系统和坐标变换
- Graphics View 的场景应用
- QCharts 的使用操作
- 常见的柱状图，饼图的绘制
- Qt 中的异步 socket 的使用
- Qt 中 TcpServer 的使用

3.1.2 基于多线程的 socket 通信与线程池的应用

为了应对高并发与高内存消耗之间不可调和的矛盾，本程序使用了在这种矛盾下的线程分配管理操作，两方面折中进行管理，创建了线程管理类，该管理类将会动态管理线程。假定每一条线程最多管理三个 socket 可以保持优良的通讯并发情况，因此在初始子线程已经达到管理上限时，再次主动在堆区创建线程，并将之后的 socket 移入该线程进行处理，并且对每一个线程进行计数，每次分配 socket 时，永远将 socket 管理数最小的线程作为第一分配对象，让该线程管理即将到来的 socket。在有 socket 断开连接时，可以自动识别 socket 断开连接，将某一线程的管理数进行改变，便于之后的判断。

在客户端和服务端的操作中需要大批量数据的传送，因此采用的是与 QByteArray 相转换的 QJsonObject 进行数据传输，两边同时存在着一般数据转 Json 和 Json 转一般数据的方法类进行转换，其转换过程直接且系统化，便于整个程序的阅读与理解。

3.1.3 持久层的实现

本程序采用的是经典持久层 Mybatis 对对数据库的增删改查进行操作，将所有的数据库操作全部写在一起，便于管理和调试，在 java 端程序中只要固定接口，其中的内容通过较为简单的 xml 编辑文件可以快速实现 sql 语句的快速封装，且由于在服

务端需要大量的情况不一的数据库操作，因此采用 Mybatis 的动态 sql 语句进行封装，起到了 java 端类 map 的操作，方便快捷，使得开发时间大大节省。

且在持久层方面，通过 Spring 框架的搭建，将 sqlSessionFactory 和 sqlSession 两个类的操作通过 bean 注入到对应的 Spring 配置文件中，起到了节约时间的作用，方便前端的测试和调用。

在持久层设计时，采用了 IDEA 特有的插件 lombok，方便快捷的动态生成了 getter 和 setter，有参无参构造函数，toString 函数等一些列 pojo 的常用函数，使得开发更加高效方便。

整个 java 端采用的是 maven 工程进行设计，对其远地仓库进行对应的配置，方便了对大量 java 包的引入，带来不必要的麻烦。

3.1.4 Webservice 的实现

本模块是学习时遇到困难最多的地方。Webservice 本质就是发布远端接口，通过服务器的地址进行各个语言间，跨系统间接口的随意调用。网上对 Webservice 的教程较为少，且鱼龙混杂，经常胡出现一些错误的引导，因此花费了不少的功夫。

本程序采用的是传统的在 soap 协议下解析 Webservice 发布服务后生成的相应 xml 文件，用 gsoap 生成 web 服务程序文件，在 C++端进行接口调用，且由于 Qt 的特性，其 UI 界面有独有的名词空间，而 gsoap 生成的程序中是在 std 命名空间下进行的，因此存在名词空间冲突和类型转的等问题。针对这一问题本程序编写了工具类对整个结果进行了逐个类型转换，最后得到符合 Qt 中数据的类型格式，便于之后数据流在各个 Qt 层面上的操作。

3.2 核心函数流程和参数说明

3.2.1 客户端核心函数功能流程：

- `void initAtBeginng(TcpSocket*socket,Client*client);`

  本函数是为了初始化不需要服务端数据的基础页面而独立出来的初始化函数，并且将登录时创建的 socket 进行共享，将其运用于之后 CS 两端之间的数据传输。并在此集中建立了多个槽函数的连接，为接下来数据的申请与传输做准备。

- `void initData();`

  本函数是在得到数据后进行的其他初始化操作，如像商品块之间的连接和跳转，购买界面的初始化，购物车和历史纪录的初始化以及搜索功能的初

始化。

- `void getByteArray(QByteArray json);`

  本函数实现的功能是，在接收到相应的 QByteArray 后进行编码提取，然后通过条件句判断得到了最后具体的操作实现，并调用相应的函数进行后续的小关操作。

- `void connectBuySignaal(Commodity* commodity);`

  搜索记录生成与跳转功能，在出发跳转功能后都会调用该函数进行 Commodity 信号的生成，相当于工厂模式，将 Commodity 页面进行加工，使得该页面具有点单和添加购物车的功能。

- ```
  void setCommendlistWall(QList<Product*>Plist);
  void setGallery(QList<Product*>Plist);
  void setInitData(QList<Order*>shoppingCart);
  void setHistoryData(QList<Order*>HistoryList);
  void setselectData(QString text);
  ```

  上述五个函数是各个跳转功能的实现体，每个函数中实现了包括控件生成和对应控件跳转功能，每个函数都有独有的静态变量计数器，防止在 List 栏目(item)状态位置调整的时候顺序混杂。

3.2.2 服务端核心函数工作流程

- `void initall();`

  在服务端中该函数起着各个控件中属性的调整和适应，比如界面原控件的删除与其他控件的鼠标相应，并在该函数中分别调用了 void AllMain::**setTabName**() ， void AllMain::**setAddorDelBtn**() ， void AllMain::**initOnSaleList**() ， void AllMain::**initCharts**() ， void AllMain::**initChartData**()函数，对 tabWidget,一些 Button,一些 list 和一些 Chart 进行了初始化设置。使得整个程序条理清晰，避免全部放到构造函数中的杂乱无章。

- `void setSlots();`

  与之前客户端的函数相似，将所有可以当即设置的信号和槽的连接进行统一连接，方便连接的查找和统一分析。

- `void dealMessage(QByteArray message,qintptr socketDiscriptor);`

  作为整个主类的通信消息处理函数，通过接收 socket 传来的 message,并通过识别进行之后选择分配，执行接下来的操作。
  ```

- void **initChartData**();void initOnSaleList();

```
void setDelPage(QList<Product*>List);
void setChangePage();
void addOnSaleItem();
void initCharts();
void setList1();
void setList2();
```

上述操作均为初始化操作和信号操作，及对所有的可视化界面上的页面和控件进行初始化，和添加的功能，包括了初始化表格，初始化打折商品栏，初始化列表栏，初始化 tabWidget 等等。作为重要的初始化函数存在。

- **TcpServer** 的操作函数

```
public:
    explicit TcpServer(QObject *parent = nullptr,int port=0);
    void incomingConnection(qintptr handle);
signals:
    void addChatRoom(qintptr socketDescriptor);
    void sglSendFromServer(QByteArray message,qintptr
socketDiscriptor);
    void sglRecivedReomClient(QByteArray message,qintptr
socketDiscriptor);
    void sendDisconnectedSignal(qintptr socketDiscriptor);
public slots:
    void dataSendFromServer(QByteArray message,qintptr
socketDiscriptor);
    void dataRecivedFromClient(QByteArray message,qintptr
socketDiscriptor);
    void disconnectChoice(qintptr SocketDescriptor, const quint16
peerPort, QThread *currentThread);
```

这些函数均为操作通信相关的操作函数，其中包括了主页面向特定 socket 发送消息的 slot 和 signal 以及 Client 端对 Server 端发送消息的时候，传输到主页面的 slot 和 signal 函数。由于数据的传输如果采用的是直接调用形式显示到主页面上，则会产生线程混杂而导致出错，此时需要信号与槽进行数据的传输，才能将数据输送到主线程中，主线程继续进行操作。

- **ThreadHandle** 的操作函数

```
static ThreadHandle& getClass();
QThread * getThread();
QThread * findThread();
void removeThread(QThread*th);
```

```
void clear();
```

该类中的函数控制着线程的产生与释放，本类为单例类，只能产生一个对象，且该对象可以在其中产生多个线程，并对其进行着添加，修改，删除，获取的功能。

3.2.3 简要功能说明

本程序的功能已经大致在要求模块阐述完毕，并在 UI 展示模块进行了初步展示。

四、 那些方面可以做的更好

4.1 界面设计方面

界面的设计已经达到我的标准，但是整个界面的流畅感还有待提升，其中对动画效果的设计并没有考虑在设计范围内，可以在后续对整个项目进行优化。

4.2 结构设计上和设计模式应用方面

本次程序由于是在没有深入了解设计模式下进行的，因此对于 MVC 的设计模式理解较少，导致分层概念比较模糊，因此代码的可读性没有那么高。结构之间解耦合仅仅体现在 Controller 层与 Dao 层之间，而 View 层与 Model 层之间没有明显的区分，且 Controller 层数据操作也比较松散，需要进一步的调整，接下来将从这一方面对相关知识进行系统学习，并对项目进行进一步优化设计。