

课设总结报告

一、需求分析

- 主要从服务端 (Server) 和客户端 (Client) 进行需求分析

1.1 Client 端

1.1.1 注册与登录功能

- 注册账号

新账号注册，输入账号密码（必填）以及一些可填可不填的账户信息（手机号，邮箱，地址）。注册成功则自动跳转登录界面

- 登录账号

用户输入正确的账号与对应的密码，即可登录进入系统

- 联系客服

若忘记密码，可以以匿名者的身份联系客服，让管理员修改你的密码。

1.1.2 商品检索功能

- 推荐商品功能

在首页轮转推荐商品，可以直接加入购物车。实现换一批功能，每次推荐一批商品在首页上。

- 商品的模糊搜索

通过模糊搜索（商品名或者简介中包含的关键词），找到想要找的商品。并可以点击并加入购物车。

1.1.3 购物车功能

- 加入购物车功能

在上述的商品检索中，都可以加入购物车

- 购物车多选式的结算和删除

可以查看自己的购物车。

在购物车中，可以多选并执行结算、单个商品删除操作、调整购物车中商品的数量。

1.1.4 历史记录功能

- 订单历史记录

可以查看历史记录。

- 退货

可以在此界面选择订单实现退货。

1.1.5 客服功能

实现与 Server 端的客服交流的功能

1.1.6 个人信息修改功能

实现修改个人信息，包括：姓名（不可重名），头像，性别，修改密码，地址，手机号，邮箱。

1.2 Server 端

1.2.1 数据统计功能

首页展示两个图表：商品销量对比柱状图，商品销量占比饼状图。方便商家分析数据。

1.2.2 商品检索功能

实现了类似 Client 端的模糊搜索。并可以点击修改商品信息。

1.2.3 商品增改功能

相同的 UI 界面，可以填写空表实现商品的增加。也可以修改已有的商品信息。

可以在这个界面设置商品参与活动。促销策略提供模板选择，可以自定义折扣力度，数量等等

1. 直接折扣：打 xx 折
2. 满减活动：满 xx 减 yy
3. 买赠活动：买 xx 赠 yy
4. 限量促销：限量 xx 件 yy 元

1.2.4 订单历史功能

查看所有的订单，显示时间状态。

1.2.5 聊天功能

优先显示在线客户。每个客户都可以联系。收到消息会有红点提示。

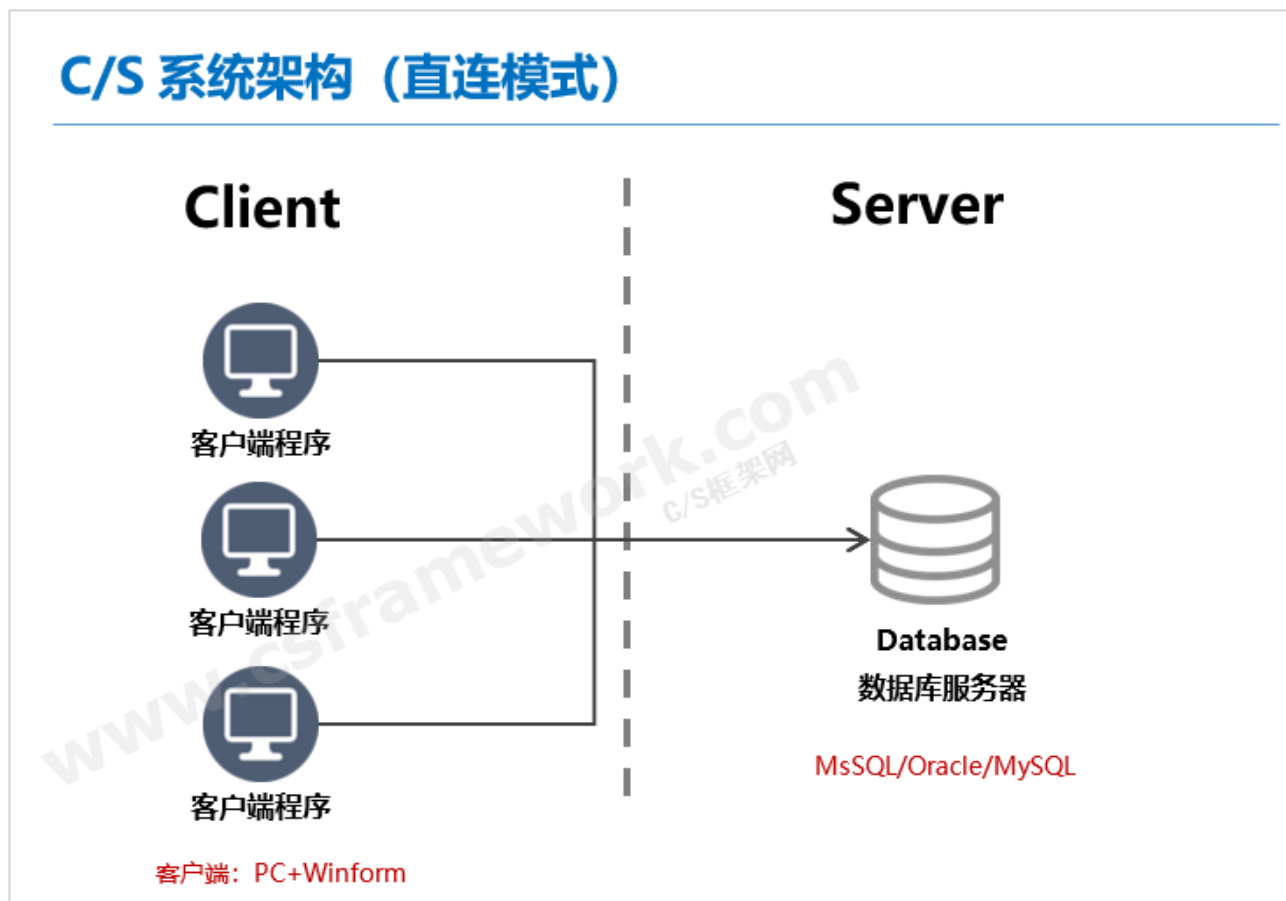
二、系统设计

2.1 C/S 架构

- C/S（客户端/服务器）架构是一种分布式系统模型，其中客户端负责与用户交互，发送请求并显示结果，而服务器负责处理请求、管理数据和业务逻辑。
- 设计 Client 端和 Server 端，两端之间使用 TCP/IP 通信。

- C/S架构的优点是功能分离、效率高、安全性好，但也存在客户端依赖性强、扩展性受限等缺点。
- Client 端：实现业务层和 UI 界面，但是不允许访问数据库
- Server 端：实现业务层和 UI 界面，设计持久层负责和数据库的连接与交互。

C/S 系统架构（直连模式）



2.2 MVC 框架

- **MVC**是三个单词的缩写:
- ○ **M**, Model (模型);
- ○ **V**, View (视图),

- `C`，Control (控制)。
- **Model层**：实现系统的业务逻辑
- **View层**：负责与用户交互，即在界面上展示数据对象给用户
- **Control层**：Model 与 View 之间沟通的桥梁，它可以分派用户的请求并选择恰当的视图以用于显示，同时它也可以解释用户的输入并将它们映射为模型层可执行的操作
- Client 端实现一个 `Allmain` 类 (Control层)，利用 QT 的信号与槽机制，负责处理所有 UI 类发送的信号，并传送给 Server 端。同时，也负责处理 Server 端发送来到讯息，并转义控制 UI 类。

2.3 持久层

- 持久层使用 C++ 模拟，用 Qt 中的 `QSqlDatabase` 和 `QSqlQuery` 访问数据库，并读取（写入）数据，转化成对象。
 1. `QSqlDatabase` 负责与数据库建立连接，并管理数据库。
我是用 `QSqlDatabase` 的 `QODBC` 驱动连接了 MySQL 数据库。
 2. `QSqlQuery` 负责执行Sql语句，通过执行 (`QSqlQuery.exec()`) 可以完成写入，也能读取数据，并使

用 `.next()` `.value()` 逐个读取数据，并手动转化成对象，完成持久层的任务。

2.4 异步 socket 通讯

- 异步通讯的通信协议采用 TCP/IP 协议，并使用 Qt 的 `QTcpSocket` 类和 `QTcpServer` 类实现异步通信。
- 1. `QTcpServer` 是 Server 端用于监听和接受 TCP 网络连接的类。当有客户端连接时，`QTcpServer` 会创建一个新的 `QTcpSocket` 实例来与客户端进行通信，从而也就实现了服务器端与客户端的连接。
- 2. `QTcpSocket` 是用于处理 TCP 网络通信的类。可以发送和接收数据。支持异步通讯，是通信的基础实例类。
- 利用好 Qt 提供的信号与槽机制。可以实现很多功能，例如 socket 与用户的映射、用户在线列表，断连处理等。

2.5 多线程：并发和互斥

- 本项目使用了多线程提高程序的效率，实现了并发的处理多个 Client 端的请求
- 实现了线程池统一的管理线程，开源项目 [ThreadPool](#)，通过任务队列分配任务，通过条件变量实现线程同步。提高

线程的效率与利用率

多线程注意事项：

1. 原因

1. 需要多线程的原因是多个 Client 端连接上 Server 时，在处理数据时会有很大的压力
2. 每个 Client 的请求都开一个新线程来处理，提高程序效率，利用率，达到更好的负载均衡。
3. 再使用线程池来管理线程，让子线程不要空跑，提高子线程的利用率。同时也能更好地管理，进行统一顺序地上锁，注销。保障线程安全。
4. 线程池借鉴于 github 上的开源项目，我自己修改增加了线程名称，用于标记不同的子线程。

2. 子线程限制

1. 不许直接操作 UI，否则线程会很慢
2. 不许使用 socket 通讯。这一点十分重要。在我的程序中，我的解决办法是：
 1. 当前线程需要发送请求时，发送信号，由主线程统一处理，不阻塞子线程。
3. 不许使用主线程的数据库。由于数据库的接口越多越好，我的解决办法是：
4. 在每个线程开始时，才连接上数据库，并把这个接

1 | 口以当前线程的名称标记。当前进程中只使用此接口。

2. 把数据库操作设计为原子操作，放置资源竞争和冲

1 | 突。

3. 处理完数据，及时删除这个接口和连接。

三、数据设计（类与通讯协议）

3.1 类的划分

3.1.1 对象类 —— objects 模块

- objects.h

```
1 //快速新建 object 的属性
2 #define PROPERTY_CREATE_H(TYPE, M) \
3 private: \
4     TYPE _##M; \
5 public: \
6     void set##M(const TYPE &value) { \
7         _##M = value; } \
8     TYPE get##M() const { return _##M; }
```

- class Client | 用户类
- class Product | 商品类
- class Order | 订单类
- class OrderList | 订单详情类
- class Shopping | 购物车类
- class Chat | 聊天记录类

该模块针对每一种需要用到的对象实现了一个类，并设计了类的成员函数。

我使用了宏函数来简化代码，更加高效地设计好了所有的对象。

3.1.2 持久层类 —— dao 模块

- class Mapper | 总表类
- class ClientMapper | client表的持久层类
- class ProductMapper | product表的持久层类
- class OrderMapper | order表的持久层类
- class OrderListMapper | orderlist表的持久层类
- class ChatMapper | chat表的持久层类
- class ShoppingMapper | shopping表的持久层类

该模块实现了持久层的任务，负责与数据库传输数据。

Mapper 总表类，实现表的整体管理和删除。后续每个 mapper 都继承这个类。

3.1.3 UI类 —— view 模块

- class BasePage | UI基类
- Client 端：
 - class HomePage | 首页
 - class SearchPage | 搜索商品页
 - class ShoppingPage | 购物车页
 - class HistoryPage | 订单历史页
 - class PersonPage | 个人信息页
 - class LogIn | 登录窗口
 - class SignIn | 注册窗口
 - class ChatRoom | 客服聊天窗口
- Server 端
 - class HomePage | 首页
 - class SearchPage | 搜索商品页
 - class ProductPage | 商品详情增改页
 - class HistoryPage | 订单历史页
 - class ChatPage | 聊天页
 - class ChatRoom | 聊天窗口

BasePage 完成基础的页面设计。后续具体的每一个 Page 都继承 BasePage，并完成细化的设计实现。

Page 页继承 `ElascrollPage` 是主窗口上的一个页面。

窗口 继承 `Elawidget` 是独立的窗口

3.1.4 控制类 —— control 模块

- class `QNChatMessage` | 聊天信息控制类

实现聊天信息的时间处理，气泡外观处理等。并将一条 `QString` 信息处理成 `Chat` 类的对象实例

- class `ObjectToJson` | 解析json类和转换json类

负责 socket 通讯时传输的 json 的编码和解码

- class `ThreadPool` | 线程池类

负责实现线程池，管理线程

3.1.5 业务类 —— model 模块

- class `Allmain` | 主类

主类，实现所有的业务层操作。包括：处理 socket 通讯，UI 管理，线程管理，实例线程锁等。

3.2 通讯协议设计

- 使用 socket 传输 `QByteArray`。字节数组存储的是 `QJsonDocument` (JSON文档由JSON格式的字符串生成), 利用JSON格式传输自己设计的对象类。

```
1   QByteArray
   ObjectToJson::changeJson(QJsonObject
   &object)
2   {
3       QJsonDocument document;
4       document.setObject(object);
5       QByteArray byteArray =
   document.toJson(QJsonDocument::Compact);
   // 无空格
6       byteArray.append("\r\n");    //分隔符
7       return byteArray;
8   }
```

- JSON格式传输数据时都采用传送 `QList` 的形式, 方便多个数据传输。以传输 `QString` 为例子。
- 编码JSON

```

1  QObject
   ObjectToJson::addStrings(QObject
   &object, QList<QString> strings)
2  {
3      for (int i = 0; i < strings.size();
4          i++) {
5          object.insert(QString("string%1").arg(i),
6                          strings[i]);
7      }
   return object;
}

```

- 解析JSON

```

1  QList<QString>
   ObjectToJson::parseStrings(QByteArray
   byteArray)
2  {
3      QList<QString> ret;
4
5      QJsonParseError jsonError;
6      QJsonDocument document =
   QJsonDocument::fromJson(byteArray,
   &jsonError);
7      if (!document.isNull() &&
   (jsonError.error ==
   QJsonParseError::NoError)){

```

```

8         if (document.isObject()){
9             QJsonObject object =
document.object();
10             for (int i = 0; ; i++)
11                 {
12                     QString name =
QString("string%1").arg(i);
13                     if (object.contains(name))
14                     {
15                         QJsonValue value =
object.value(name);
16                         ret.append(value.toString());
17                     }
18                     else break;
19                 }
20             }
21         return ret;
22     }

```

- 信号设计

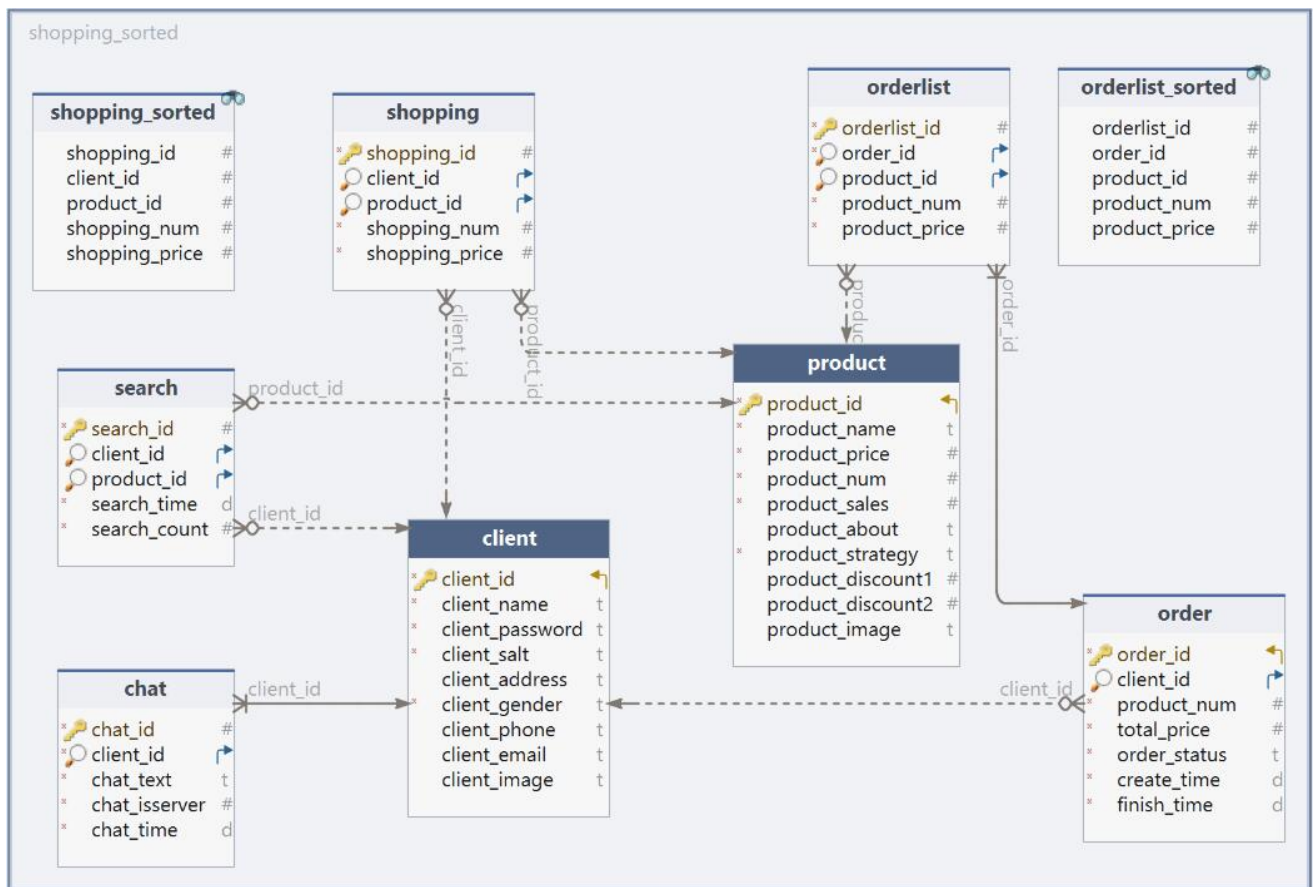
```

1 //200 成功; 302 重定向; 403 拒绝; 404 找不到;
  500 运行错误;
2 #define LOGIN 20010101 //登录请求, 登录成功
3 #define LOGINFAIL 40410101 //登录失败: 账号或
  密码错误

```

```
4 #define SIGNIN 20010201 //注册请求, 注册成功
5 #define SIGNINFAIL 40410201 //注册失败: 账
  号已存在
6 #define SIGNINERROR 30410201 //NULL
7 #define CHATMSG 20010301 //发送聊天信息
8 #define CHATHISTORY 20010302 //请求聊天历史
9 #define PERSONCHANGE 20010401 //修改个人账
  号, 修改成功
10 #define PERSONCHANGEFAIL 40410401 //修改个
    人账号失败
11 #define PERSONCHANGEERROR 40310401 //修改个
    人密码错误
12 #define SEARCHPRODUCT 20010501 //搜索商品
13 #define REQUESTHOME 20010601 //首页请求
14 #define ADDSHOPPING 20010701 //加入购物车
15 #define REQUESTSHOPPING 20010702 //购物车
    请求
16 #define DELSHOPPING 20010703 //删除购物车
17 #define UPDATESHOPPING 20010704 //更新购物车
18 #define CREATEORDER 20010801 //创建订单
19 #define UPDATEORDER 20010802 //更新订单
20 #define CHECKORDER 20010803 //检查库存是否充
    足
21 #define CREATEORDERLIST 20010901 //创建订
    单List
22 #define REQUESTORDER 20011001 //请求订单历
    史
```


四、数据库设计



4.1 client 表

- 用户信息表

列名	备注	类型	是否 NULL	其他属性
client_id	客户 id	int	NOT NULL	主键、无 符号、自 动递增
client_name	客户 名称	varchar	NOT NULL	utf8
client_password	客户 密码	varchar	NOT NULL	utf8
client_salt	客户 盐值	varchar	NOT NULL	utf8
client_address	客户 地址	varchar	NULL	utf8
client_gender	客户 性别	enum{男, 女,未知}	NOT NULL	utf8、默 认：未知
client_phone	客户 手机 号	varchar	NULL	utf8
client_email	客户 邮箱	varchar	NULL	utf8
client_image	客户 头像	varchar	NULL	utf8

- client_id 为主键且自动递增，作为识别客户的唯一关键。
- 用户名不能为空，性别无法置空（默认为“未知”），其余列可以为空。
- client_image 存照片的绝对地址，所有照片都是这种存法。头像默认为："C:/Users/PC/Desktop/ClassPro/MyPro/Client/include/Resource/T.jpg"
- 为了满足数据库中不能存密码明文的原则，我使用了 sha256 哈希加密算法，在数据库中存储密文。同时引入加盐值 salt 加强加密的安全性，保证两个密码即使一样，在加密之后也会不同。具体加密算法如下：

```
1  QString Allmain::generateRandomSalt(int
   length)
2  {
3      QByteArray salt;
4      salt.resize(length);
5      for (int i = 0; i < length; ++i) {
6          salt[i] = static_cast<char>
   (QRandomGenerator::global()->bounded(0,
   256));
7      }
8      return
   QString::fromUtf8(salt.toHex());
9  }
```

- 生成了指定长度的 salt 值

```
1  QString Allmain::sha256Hash(const QString
   &data, const QString &salt)
2  {
3      QByteArray combinedData =
   data.toUtf8() + salt.toUtf8();
4      QByteArray hash =
   QCryptographicHash::hash(combinedData,
   QCryptographicHash::Sha256);
5      return
   QString::fromUtf8(hash.toHex());
6  }
```

- 完成加盐 sha256哈希加密，同时保存成 hex 16进制形式。既处理了加密之后出现的 QString 无法打印的字符，又实现了定长，使格式更加整齐，方便后续需要。

4.2 product 表

- 商品信息表

列名	备注	类型	是否 NULL	其他属 性
product_id	商品 id	int	NOT NULL	主键、 无符 号、自 动递增
product_name	商品 名称	varchar	NOT NULL	utf8
product_price	商品 价格	double	NOT NULL	无符号
product_num	商品 存量	int	NOT NULL	无符 号、默 认：0
product_sales	商品 销量	int	NOT NULL	无符 号、默 认：0
product_about	商品 描述	text	NULL	utf8
product_strategy	商品 促销 策略	enum{'0', '1','2', '3','4'}	NOT NULL	默 认：'0'

列名	备注	类型	是否 NULL	其他属 性
product_discount1	商品折扣信息	double	NULL	
product_discount1	商品折扣信息	double	NULL	
product_image	商品图片	varchar	NULL	utf8

- product_id 为主键且自动递增，作为识别商品的唯一关键。
- 商品名和商品价格不能为空，商品数量和商品销量无法置空（默认为0），商品折扣不能为空（默认为100，即不打折），其余列可以为空。
- 商品图片存储的是图片的绝对地址
- 商品图片默认为："C:/Users/PC/Desktop/ClassPro/MyPro/Server/include/Resource/NULLProduct.jpg" 表示暂无图片

4.3 shopping 表

- 购物车表

列名	备注	类型	是否 NULL	其他属性
shopping_id	购物车 id	int	NOT NULL	主键、无符 号、自动递 增
client_id	购物车 所属者	int	NULL	无符号
product_id	购物车 商品	int	NULL	无符号
shopping_num	商品数 量	int	NOT NULL	无符号、默 认：1
shopping_price	商品价 格	double	NOT NULL	无符号

- product_id 为主键且自动递增
- 商品数量：客户希望购买多少件商品，非空，默认：1
- 商品价格：客户加入购物车时商品的价格，用于后续比较价格的变化，非空，不可修改
- client_id, product_id 为两个外键

字段	被引用表	被引用字段	删除时	更新时
client_id	client	client_id	SET NULL	RESTRICT
product_id	product	product_id	SET NULL	RESTRICT

- 通过外键，实现当用户或商品被删除时，对应的购物车被置空，造成逻辑删除。但是在后续商家统计数据时，该记录还可以利用。
- 新建一个视图方便购物车的逻辑查询

```
1 | SELECT * FROM shopping order by
   | shopping.client_id
```

- 由于 client_id 的主键性质，在找到当前id之后，后续连续的商品都是该用户购物车中的商品，提高搜索效率。

4.4 order 表

- 订单记录表

列名	备注	类型	是否 NULL	其他属 性
order_id	订单 id	int	NOT NULL	主键、 无符 号、自 动递增
client_id	订单 用户 id	int	NULL	无符号
product_num	订单 商品 数量	int	NOT NULL	无符 号、默 认0
total_price	订单 总价	double	NOT NULL	无符号
order_status	订单 状态	enum{'未支 付','已完成','已取 消','已退款'}	NOT NULL	utf8
create_time	订单 下单 时间	datetime	NOT NULL	

列名	备注	类型	是否 NULL	其他属性
finish_time	订单完成时间	datetime	NOT NULL	

- 订单在我的系统中属于“快照”，只会存下下单时相关属性，并不会更新。order_id 为主键且自动递增
- 订单快照中会存储：订单的总价，下单时间，完成时间以及订单状态。
- client_id 为外键

字段	被引用表	被引用字段	删除时	更新时
client_id	client	client_id	SET NULL	RESTRICT

- 实现当客户被删除时，这条订单被逻辑删除，只作为数据统计

4.5 orderlist 表

- 订单记录详情表

列名	备注	类型	是否 NULL	其他属性
orderlist_id	订单 详情id	int	NOT NULL	主键、无符 号、自动递增
order_id	订单id	int	NOT NULL	无符号
product_id	商品id	int	NULL	无符号
product_num	商品 数量	int	NOT NULL	无符号，默 认：1
product_price	商品 价格	double	NOT NULL	无符号

- orderlist_id 为主键且自动递增
- 商品数量：客户希望购买多少件商品，非空，默认：1
- 商品价格：客户加入购物车时商品的价格，用于后续比较价格的变化，非空，不可修改
- order_id, product_id 为外键

字段	被引用表	被引用字段	删除时	更新时
order_id	order	order_id	RESTRICT	RESTRICT
product_id	product	product_id	SET NULL	RESTRICT

- 实现当商品被删除时，这条订单被逻辑删除，只作为数据统计
- 拒绝 order_id 被删除，实际上也不会被删除，加一道保险
- 新建一个视图方便订单详情的逻辑查询

```
1 | SELECT * FROM orderlist order by
   | orderlist.order_id
```

- 实际查询逻辑与购物车类似，在找到当前id之后，后续连续的商品都是该用户购物车中的商品，提高搜索效率。

4.6 chat 表

- 聊天记录表

列名	备注	类型	是否 NULL	其他属性
chat_id	聊天记录id	int	NOT NULL	主键、 无符号、自动递增
client_id	聊天用户id	int	NOT NULL	无符号、外键
chat_text	聊天记录	text	NOT NULL	utf8
chat_isserver	是否客服发送的信息	tinyint	NOT NULL	默认：0
chat_time	聊天时间	datetime	NOT NULL	

- search_id 为主键且自动递增
- chat_isserver 用于标记是 Server 发送的信息还是 Client 发送的。
- client_id 为外键

字段	被引用表	被引用字段	删除时	更新时
client_id	client	client_id	CASCADE	RESTRICT

- 这里有一个特殊设计：当用户被删除时，他的聊天记录也会被删除。为了尊重用户隐私，而且聊天记录没有利用价值。于是，聊天记录会随着 client_id 的删除而被删除。

五、需求实现与 UI 展示

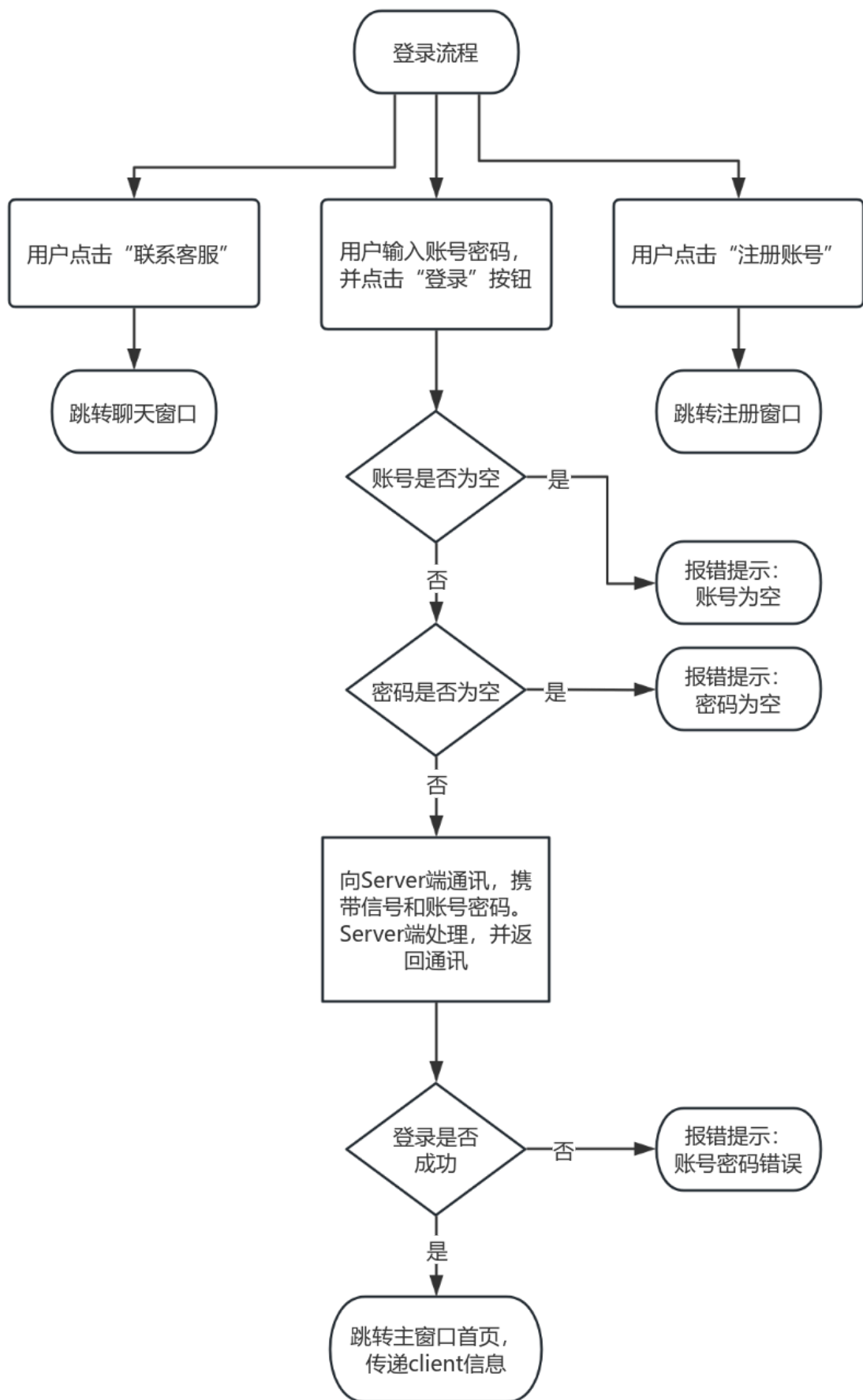
5.1 Client 端

5.1.1 注册与登录功能

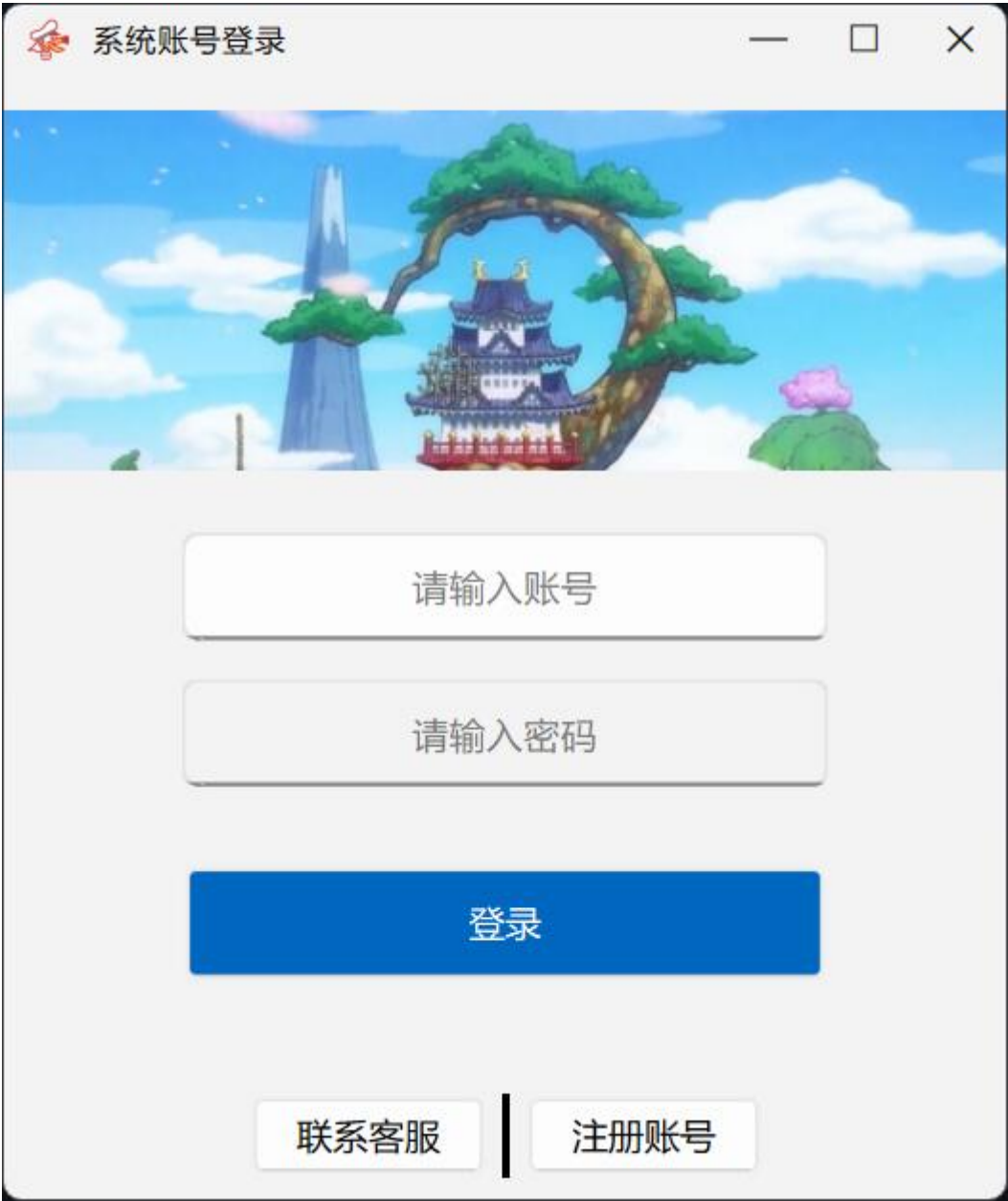
- 登录流程：
 - 用户输入账号密码，并点击“登录”按钮。
 - 若账号为空，报错提示
 - 若密码为空，报错提示
 - 否则向Server端通讯，携带信号和账号密码。Server端查询数据库。读取该账号的密码明文pwd和盐值salt。若找不到账号，向Client端返回报错。否则加密用户输入的密码，得到明文进行比对。比对成功向Client端返回成功，并携带client的详细信息。否则向

Client端返回报错。成功登录，跳转到首页，并向Allmain传输client信息。

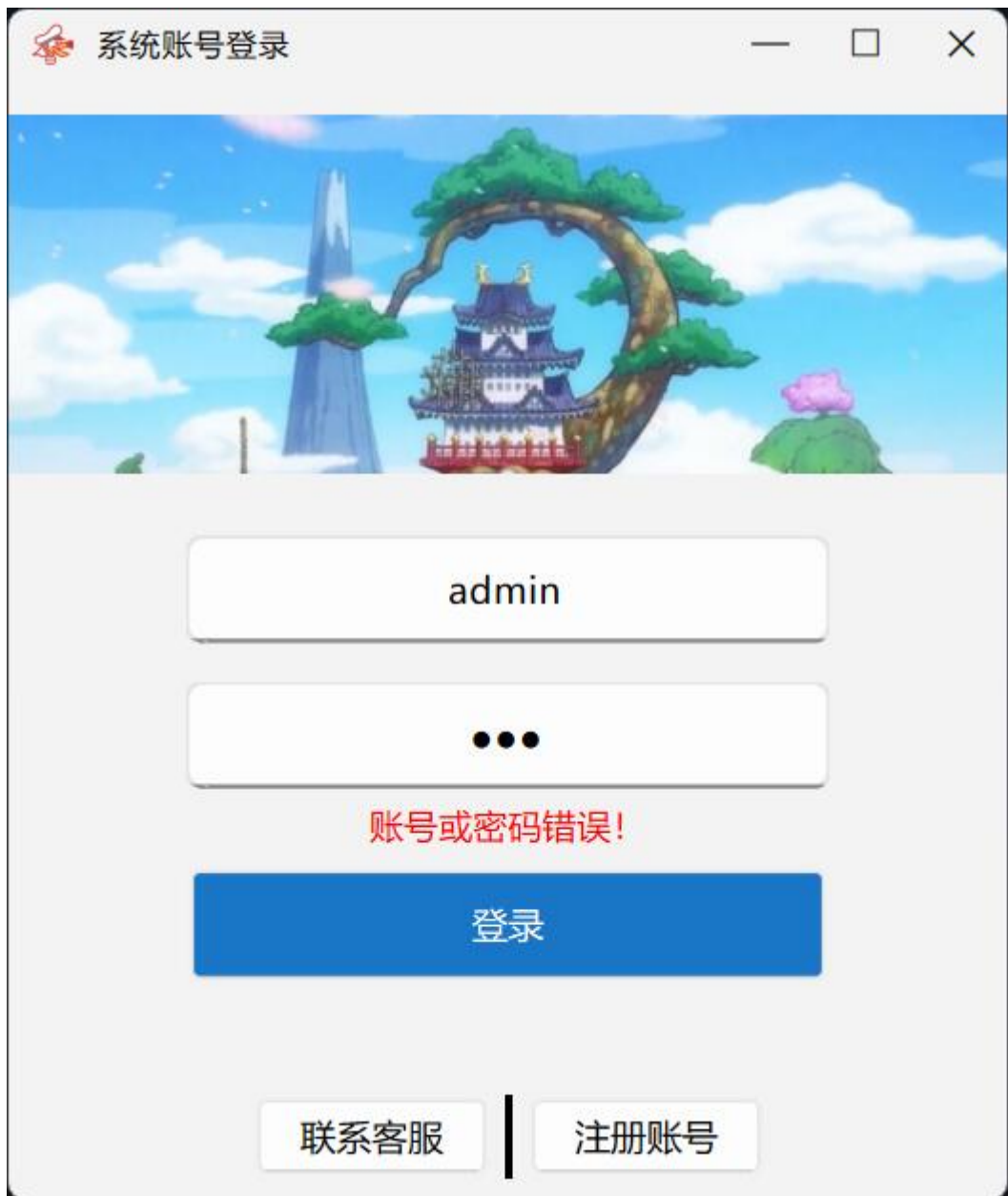
- 用户点击“注册账号”，跳转注册窗口。
- 用户点击“联系客服”，跳转聊天窗口。



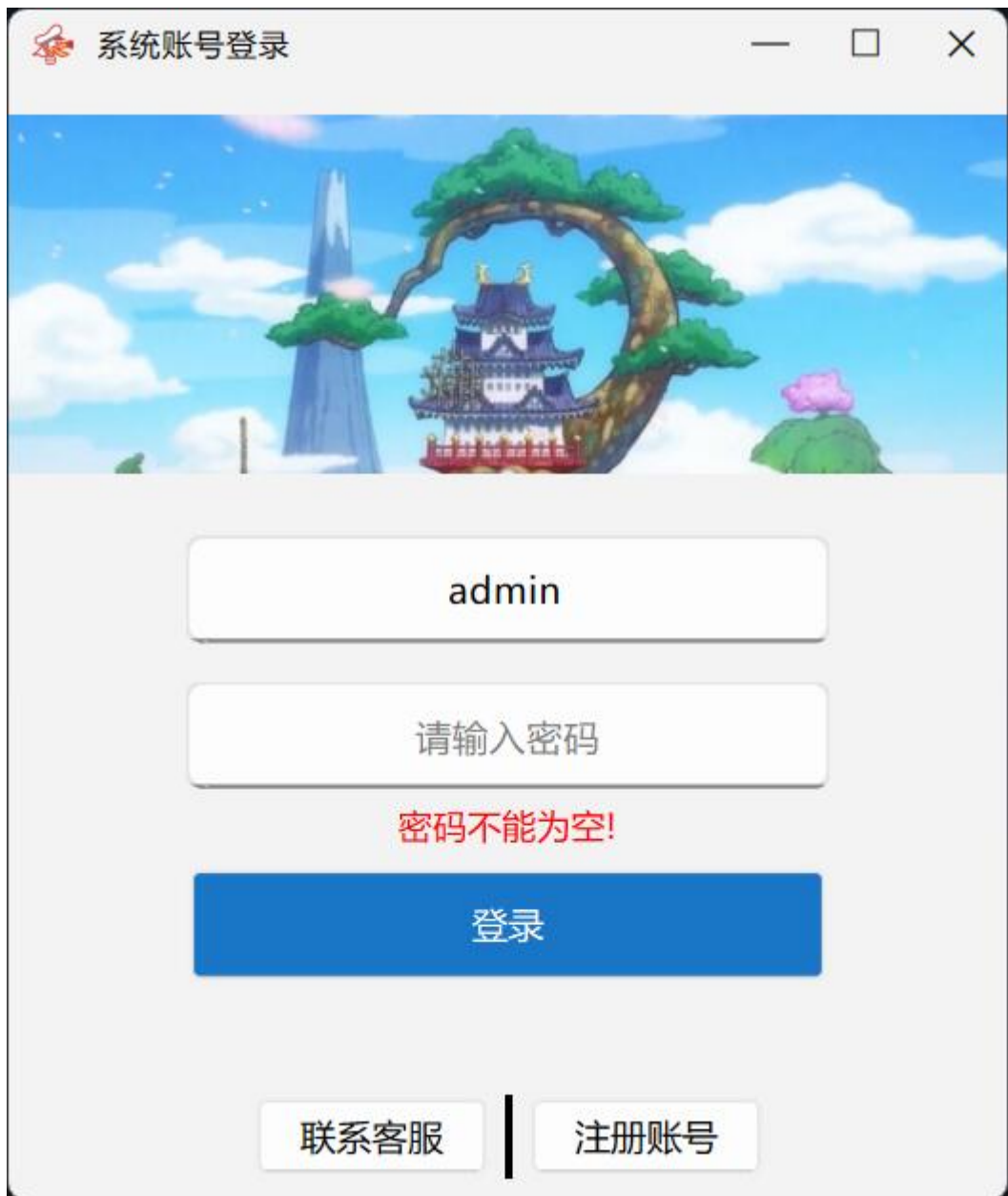
- 登录窗口 UI



- 登录账号密码错误提示



- 登录密码为空提示



The image shows a window titled "系统账号登录" (System Account Login). The window has a header bar with the title and standard window controls (minimize, maximize, close). Below the header is a decorative banner featuring a stylized illustration of a traditional Chinese building with a curved roof, surrounded by green trees and a blue sky with clouds. The main content area contains a login form with the following elements: a text input field containing the username "admin"; a text input field with the placeholder text "请输入密码" (Please enter password); a red error message "密码不能为空!" (Password cannot be empty!) displayed below the password field; a blue button labeled "登录" (Login); and at the bottom, two buttons labeled "联系客服" (Contact Customer Service) and "注册账号" (Register Account) separated by a vertical line.

系统账号登录

admin

请输入密码


密码不能为空!

登录

联系客服 | 注册账号

- 登录用户名为空提示

系统账号登录



请输入账号

...

用户名不能为空!

登录

联系客服 | 注册账号

- 注册流程：
 - 用户输入账号密码，并点击“登录”按钮。
- 注册窗口 UI

 系统账号注册

— □ ×

欢迎!! (*´ω`*)つ口



请输入账号，必填



请输入密码，必填



请输入手机号



请输入邮箱



请输入地址

返回登录

注册账号

- 注册用户名为空提示

 系统账号注册

— □ ×

欢迎!! (*´ω`*)つ口



请输入账号，必填

账号名不能为空!



...



请输入手机号



请输入邮箱



请输入地址

返回登录

注册账号

- 注册密码为空提示

 系统账号注册

— □ ×

欢迎!! (*´ω`*)つ口



abc



请输入密码, 必填

密码不能为空!



请输入手机号



请输入邮箱



请输入地址

返回登录

注册账号

- 注册用户名已存在提示

系统账号注册

—□×

欢迎!! (*´ω`*)つ口

😊

npc

🔒

请输入密码, 必填

☎

请输入手机号

✉

请输入邮箱

🏠

请输入地址

已存在该账号名! 请尝试其他。

返回登录

注册账号

5.1.2

六、其他项目相关
