

## 目录

小鳄鱼聊天室设计与实现 .....	1
1 项目简介 .....	1
2 功能简介 .....	1
3 技术说明 .....	2
3.1 JavaFX .....	2
3.2 WebSocket .....	4
3.3 SSL/TLS .....	5
3.4 策略模式 .....	7
4 项目设计 .....	8
4.1 客户端设计 .....	9
4.2 服务端设计 .....	10
4.3 数据库设计 .....	11
4.4 通信协议设计 .....	12
4.4.1 报文格式 .....	12
4.4.2 通信规则 .....	14
5 核心功能代码说明 .....	15
5.1 通信机制 .....	15
5.2 登录功能 .....	17
5.3 注册功能 .....	20
5.4 添加、删除好友功能 .....	26
5.5 实时聊天功能 .....	35
5.6 文件传输 .....	39
6 功能演示 .....	43
7 遇到的问题 .....	49
8 结论与展望 .....	51

# 小鳄鱼聊天室设计与实现

## 1 项目简介

本项目是一个基于 C/S 架构的轻量级通信软件，名为“小鳄鱼聊天室”，旨在提供一个简洁、直观且流畅的在线交流平台。在 Java 环境下独立开发完成的本项目，实现了基本的文本聊天功能以及文件传输功能，并采用 SSL/TLS 技术确保了客户端与服务器之间通信的加密，保障了用户数据的安全性。

## 2 功能简介

小鳄鱼聊天室是一个基于计算机网络的通信软件，旨在模仿经典的即时通讯工具 QQ，为用户提供一个简洁直观的界面和流畅的用户体验，以及安全的通信环境。

### 主要功能：

- **用户注册与登录：**新用户可以通过注册功能创建自己的账号，而已注册用户可以通过登录功能进入聊天室。
- **添加与删除好友：**用户能够搜索其他用户并发送好友请求，一旦对方接受，双方即成为好友。用户也可以管理自己的好友列表，包括删除不再联系的好友。
- **实时文本聊天：**用户可以与好友进行单独的实时文本对话，支持发送消息和接收即时回复，让沟通变得轻松而高效。

### 附加功能：

- **文件传输：**用户可以在聊天中发送和接收文件，使得共享文档和图片变得简单快捷。
- **通信加密（非功能性需求）：**使用 SSL/TLS 技术对所有通信进行加密，保护用户数据安全，防止信息泄露。

## 3 技术说明

小鳄鱼聊天室软件在技术层面采用了现代且成熟的技术栈，以满足性能、安全性和用户体验的需求：

- **开发环境：**使用了 JDK17，它是目前 Java 开发中稳定且功能丰富的版本，为项目提供了强大的语言特性和性能优势。
- **界面设计：**采用 JavaFX 作为 GUI 框架，构建了一个既美观又直观的用户界面，提升了用户的交互体验。
- **通信协议：**使用 WebSocket 协议实现了客户端与服务器之间的双向实时通信，确保了消息传输的及时性和高效性。此外，使用 SFTP 协议实现了文件上传的功能，它在文件传输过程中提供了强大的安全性，确保文件在网络中的传输过程是加密的，防止数据在传输过程中被拦截或篡改。
- **数据存储：**选择了 MySQL 数据库来存储用户数据、好友关系和聊天记录，因其稳定性和广泛的社区支持，适用于处理大量的数据操作。
- **安全性：**通过 SSL/TLS 加密协议以及数字签名和单向哈希等技术，所有通过网络传输的数据都进行了加密处理，显著提高了通信的安全性，防止了数据在传输过程中的潜在威胁。
- **服务器部署：**服务端部署至阿里云服务器，这不仅提供了稳定可靠的服务运行环境，还便于进行规模扩展。
- **设计模式：**引入了策略模式来处理客户端发送的各种不同类型的请求，通过定义一系列的算法族，封装每个算法，并使它们之间可以互换。策略模式让算法的变化独立于使用算法的客户，这为不同的请求处理提供了更高的灵活性和扩展性。

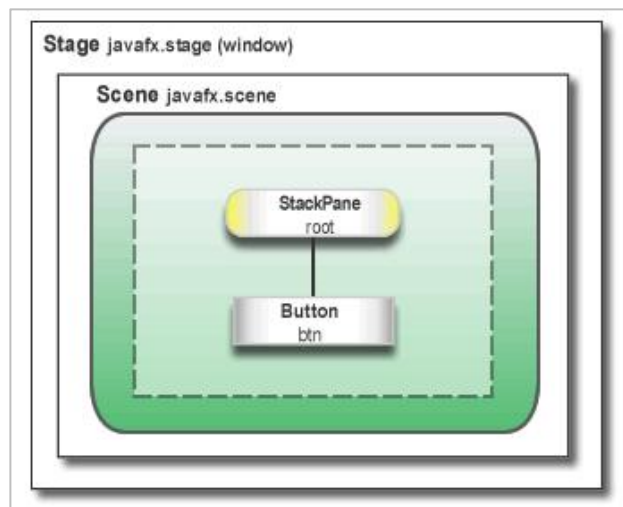
### 3.1 JavaFX

JavaFX 是一个用于开发桌面应用和丰富互联网应用程序（RIA）的图形库。它提供了一系列先进的 UI 控件，并支持 3D 图形、多媒体内容处理以及各种动画效果。JavaFX 支持 FXML，允许开发者以声明性方式设计用户界面，与后端逻辑

分离，提高了开发效率和界面的可维护性。JavaFX 的 CSS 样式支持使得定制化 UI 变得非常灵活，能够创造出既美观又符合品牌特色的用户界面。

JavaFX 使用硬件加速的图形管道进行渲染，称为 Prism。此外，为了完全加速图形的使用，它通过内部使用 DirectX 和 OpenGL 来利用软件或硬件渲染机制。JavaFX 具有依赖于平台的 Glass 窗口化工具包层，用于连接到本机操作系统。它使用操作系统的事件队列来计划线程使用。此外，它还异步处理窗口、事件、计时器、媒体和 Web 引擎，支持媒体播放和 HTML/CSS。

JavaFX 应用程序的主要结构如下：



在这里，我们注意到两个主要容器：

- Stage 是应用程序的主要容器和入口点。它表示主窗口并作为 start () 方法的参数传递。

- Scene 是用于保存 UI 元素（如图像视图、按钮、网格、文本框）的容器。场景可以替换或切换到另一个场景。这表示分层对象的图，称为场景图。该层次结构中的每个元素都称为一个节点。单个节点有其 ID、样式、效果、事件处理程序和状态。此外，场景还包含布局容器、图像、媒体。以下是 JavaFX 的特点介绍：

## 1. 线程

在系统级别，JVM 创建单独的线程来运行和呈现应用程序：

- Prism rendering thread——负责单独渲染场景图。
- Application thread——是任何 JavaFX 应用程序的主线程，所有活动节点和组件都附加到此线程。

## 2. 生命周期

`javafx.application.Application` 类具有以下生命周期方法：

- `init()`——在创建应用程序实例后调用。此时，JavaFX API 尚未准备就绪，因此无法在此处创建图形组件。
- `start(Stage stage)`——所有图形组件都在此处创建，图形活动的主线从这里开始。
- `stop()`——在应用程序关闭之前调用；例如，当用户关闭主窗口时。在应用程序终止之前重写此方法。
- `launch()` 方法启动 JavaFX 应用程序。

## 3. FXML

JavaFX 使用特殊的 FXML 标记语言来创建视图接口。这提供了一个基于 XML 的结构，用于将视图与业务逻辑分离。XML 在这里更合适，因为它能够非常自然地表示 Scene Graph 层次结构。

此外，使用 SceneBuilder 软件（或者 idea 的 SceneBuilder 插件）可以使用拖拽的方式自动生成 `.fxml` 文件，更进一步地简化了 UI 的开发。

最后，为了加载 `.fxml` 文件，我们使用 `FXMLLoader` 类，它生成了场景层次结构的对象图。

## 3.2 WebSocket

WebSocket 是一个网络通信协议，提供了一种在客户端和服务器之间建立持久连接的方式，并能够实现全双工通信。这意味着服务器可以随时向客户端发送消息，客户端也可以随时向服务器发送消息，而不需要像传统的 HTTP 请求那样每次都建立一个新的连接。

### 1. 握手过程

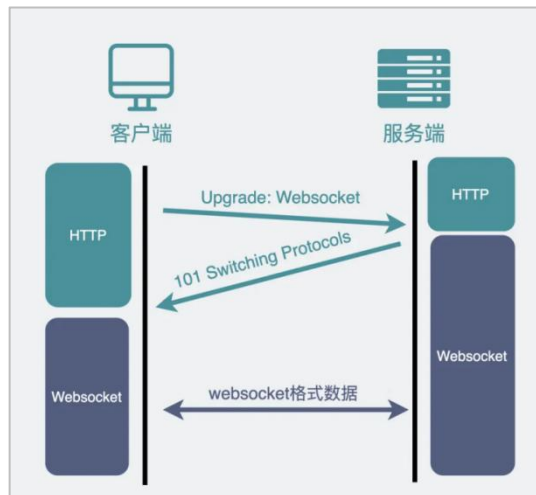
WebSocket 连接的建立始于一个 HTTP 请求，这个过程称为“握手”。客户端发送一个 HTTP 请求到服务器

这个 HTTP 请求告诉服务器，客户端希望将通信升级到 WebSocket。如果服务器支持 WebSocket 并同意升级，它将响应一个 HTTP 101 状态码，表示切换协议，并包含确认的头信息。

## 2. 数据帧和消息

一旦握手成功，客户端和服务端之间将建立 WebSocket 连接。数据通过称为“帧”的小片段来传输。WebSocket 定义了几种不同类型的帧，每种类型都有不同的用途。例如，有文本帧、二进制帧、关闭帧等。

消息可能会分成多个帧进行发送，接收方需要将这些帧重新组装成完整的消息。



本次项目使用了 Java-WebSocket 开源代码库实现的 WebSocket 通信协议。

## 3.3 SSL/TLS

SSL (Secure Sockets Layer) 和 TLS (Transport Layer Security) 是网络通信协议，用于在互联网上为数据传输提供安全和数据完整性保护。TLS 是 SSL 的后续版本，提供了更强的安全性，但两者在基本原理上是相似的。以下是 SSL/TLS 的工作原理：

### 1. 握手过程

SSL/TLS 的握手过程是建立安全通信的关键，分为几个步骤：

#### a. 客户端 Hello (ClientHello)

客户端开始 SSL/TLS 握手，发送一个 ClientHello 消息，其中包含客户端支持的 SSL/TLS 版本、加密套件列表以及一个客户端随机数 (Client Random)。

#### b. 服务器 Hello (ServerHello)

服务器回应一个 ServerHello 消息，选择一个客户端也支持的加密套件和 SSL/TLS 版本，并提供一个服务器随机数 (Server Random)。

### c. 服务器证书 (Server Certificate)

服务器发送它的证书给客户端，证书中包含了服务器的公钥。

### d. 密钥交换 (Key Exchange)

客户端验证服务器的证书是否可信，然后使用证书中的公钥来加密一个预主密钥 (Pre-Master Secret) 并发送给服务器。服务器用自己的私钥解密得到预主密钥。

### e. 握手完成

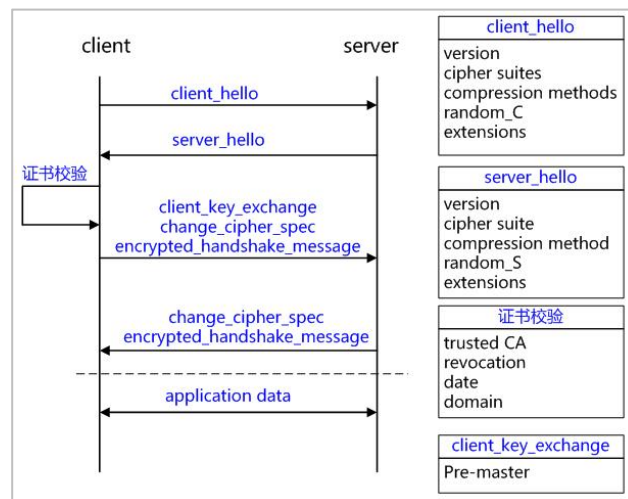
客户端和服务器都使用预主密钥和之前交换的随机数生成最终的会话密钥 (Session Key)。接下来的通信将使用这个会话密钥进行加密，确保传输数据的机密性。

## 2. 数据传输

在握手过程成功完成后，客户端和服务器就建立了一个安全的加密通道。所有传输的数据都将使用会话密钥进行对称加密。只有客户端和服务器才有对应的密钥来加密和解密数据，从而保证了传输过程中的安全性。

## 3. 会话结束

一旦通信结束或者一方想要结束会话，会发送一个 close\_notify 警告，随后关闭连接。之后的任何尝试重新连接都需要进行新的握手过程。



## 加密组件

SSL/TLS 的安全性基于以下几个加密组件：

### a) 对称加密

客户端和服务器使用相同的密钥来加密和解密数据，如 AES、3DES 等。

### b) 非对称加密



在握手阶段使用，如 RSA、ECC 等。服务器的公钥用来加密信息，私钥用来解密。

#### c) 数字证书和签名

数字证书用来验证服务器的身份，通常由可信的证书颁发机构（CA）签发。数字签名保证了证书的真实性。

#### d) 安全散列算法

如 SHA-256 等，用于验证数据的完整性，防止篡改。

SSL/TLS 通过这些加密技术和严格的握手过程，确保了在不安全的网络中进行安全的数据传输。

项目中我在使用 WebSocket 时，配合上了 SSL/TLS 协议，这保证了通信信道的安全。

### 3.4 策略模式

该设计模式定义了一系列算法，并将每个算法封装起来，使它们可以相互替换，且算法的变化不会影响使用算法的客户。策略模式属于对象行为模式，它通过对算法进行封装，把使用算法的责任和算法的实现分割开来，并委派给不同的对象对这些算法进行管理。策略模式主要解决在有多种算法相似的情况下，使用 if...else 所带来的复杂和难以维护。当一个系统有许多许多类，而区分它们的只是他们直接的行为，我们就可以将这些算法封装成一个一个的类，任意地替换。实现关键是实现同一个接口。

#### ● 优点：

- 1、算法可以自由切换。
- 2、避免使用多重条件判断。
- 3、扩展性良好。

#### ● 缺点：

- 1、策略类会增多。
- 2、所有策略类都需要对外暴露。

#### ● 使用场景：

- 1、如果在一个系统里面有许多类，它们之间的区别仅在于它们的行为，

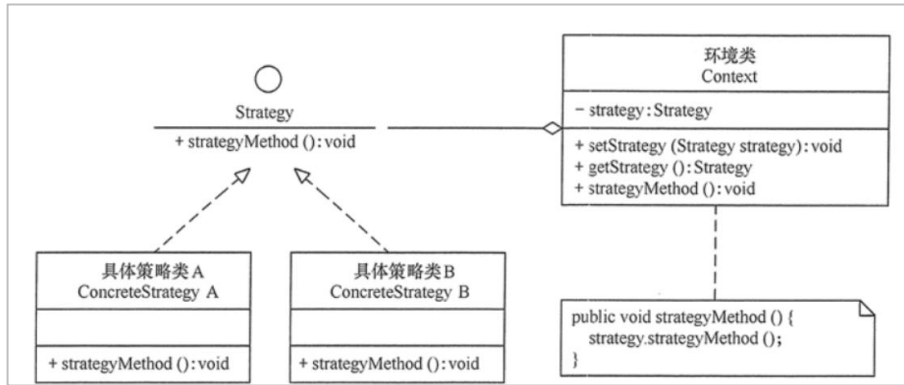


那么使用策略模式可以动态地让一个对象在许多行为中选择一种行为。

2、一个系统需要动态地在几种算法中选择一种。

3、如果一个对象有很多的行为，如果不用恰当的模式，这些行为就只好使用多重的条件选择语句来实现。

策略模式结构图：



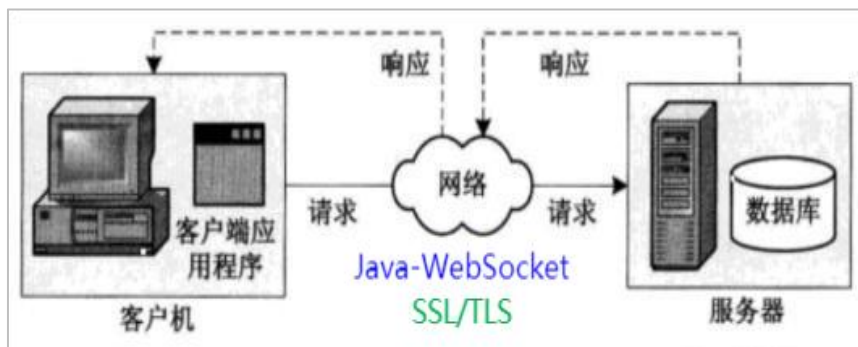
可以看出策略模式有三个组成角色：

- 抽象策略(Strategy)类
- 具体策略(Concrete Strategy)类
- 环境(Context)类

策略模式通过将算法与使用算法的代码解耦，提供了一种动态选择不同算法的方法。客户端代码不需要知道具体的算法细节，而是通过调用环境类来使用所选择的策略。

## 4 项目设计

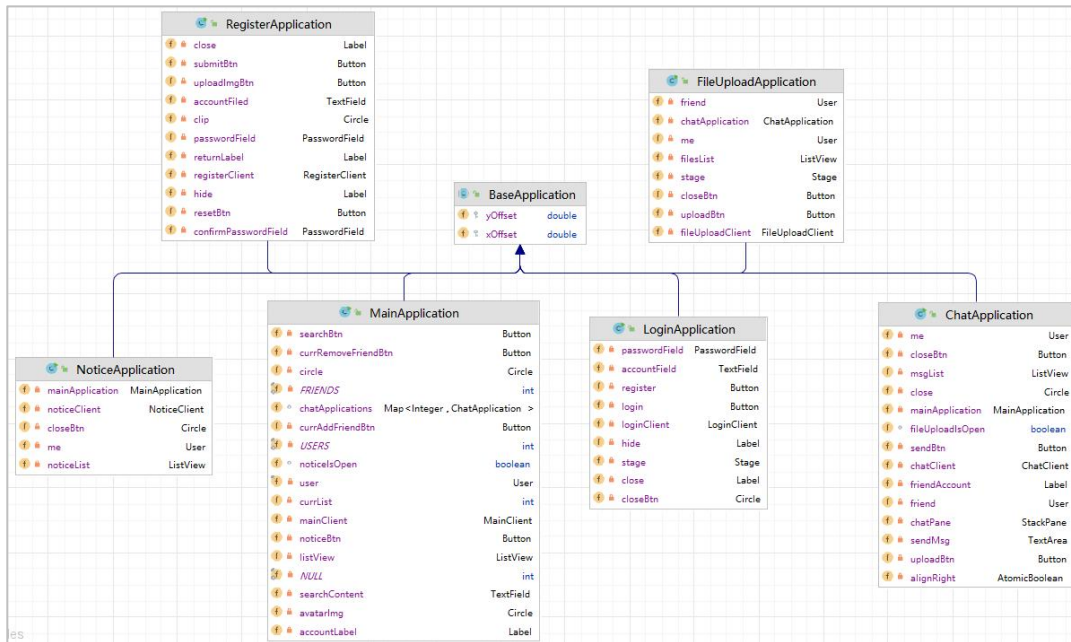
本项目整体采用 C/S 架构，使用 Java-WebSocket 作为中间件进行客户端和服务端的连接。



## 4.1 客户端设计

### 1. 界面设计（Application）

泛化 JavaFX 的 Application 类，自定义一个 BaseApplication 抽象类，在里面定义一些适用于项目的方法。对于每一个程序的 UI 界面，均有一个 BaseApplication 的子类与之相对应，以下是 application 软件包下的 UML 类图：

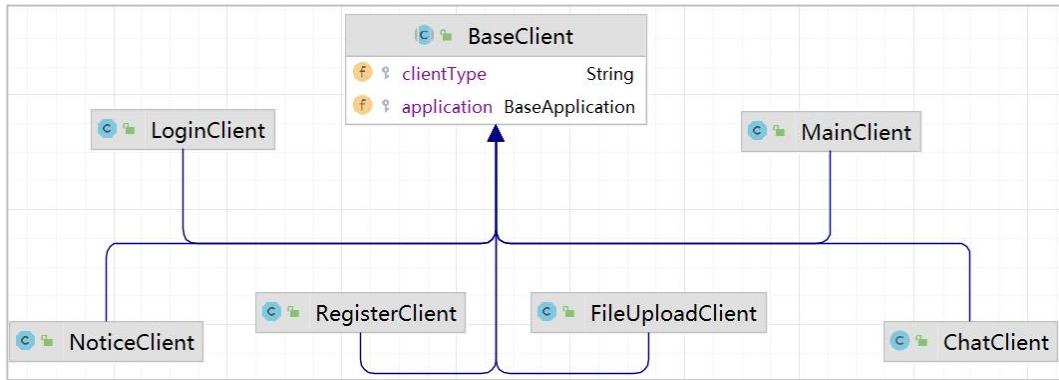


对于每一个 Application，均有一个 fxml 文件与之对应，Application 内部只需要加载该 fxml 文件即可生成 UI，我只需要在该类内部组件添加合适的监听器即可。

### 2. 客户端通信设计（client）

与 application 类似，泛化 WebSocketClient 类，自定义一个 BaseClient 抽象类，在里面实现最重要的 onMessage 方法，这个方法在接收到服务端发来的消息后，立刻转发给 application 下持有该类对象的 BaseApplication 的子类，子类作出相应的响应。

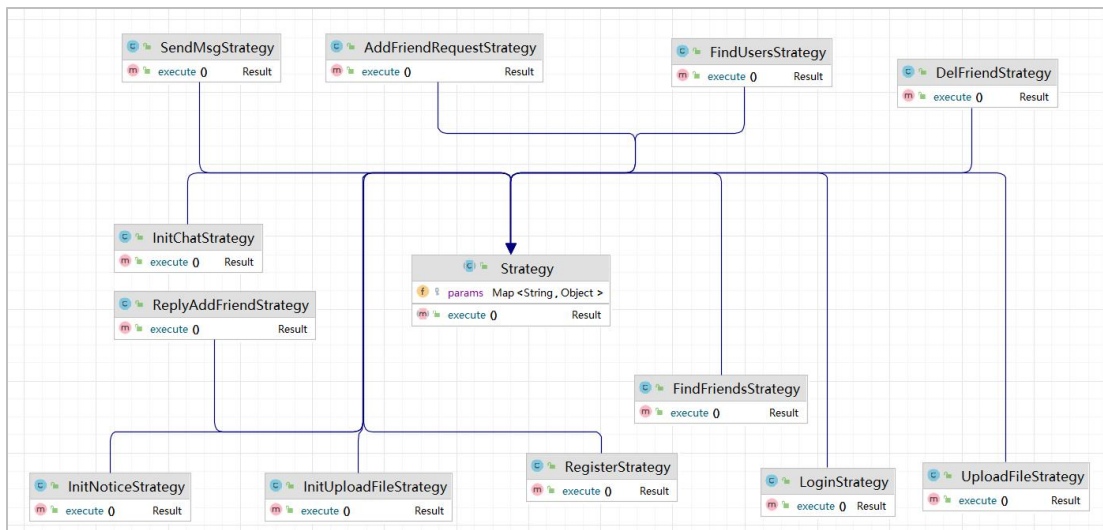
以下是 client 软件包下的 UML 类图：



这样的设计使得客户端实现了松耦合，每一个 Client 子类均不需要自己实现具体方法（`onOpen()`、`onClose()`方法每个类在内部重载了，但是只做了打印操作，便于我观察程序的运行情况），这使得扩展客户端功能十分方便，主要的工作量在 `application` 软件包下。

## 4.2 服务端设计

服务端主要采用策略模式，而对于每一个具体的策略类，都去调用 DAO 层访问数据库，并将结果封装进 `Result` 类里面返回。以下是 `strategy` 软件包下的 UML 类图：



上述每一个子类均对应与一个客户端请求类型，`Server` 类作为上下文持有 `Strategy` 类的对象，根据请求类型执行不同的算法。`Server` 类图如下：

Server		
f	strategy	Strategy
f	onlineUsers	Map<Integer, WebSocket>
f	onlineChatsOfUploadFile	Map<String, WebSocket>
f	onlineNotices	Map<Integer, WebSocket>
f	strategyMap	Map<Integer, Function<Map<String, Object>, Strategy>>
f	onlineChats	Map<String, WebSocket>
m	onOpen (WebSocket, ClientHandshake)	void
m	onError (WebSocket, Exception)	void
m	onMessage (WebSocket, String)	void
m	doOperation ()	Result?
m	initStrategy ()	void
m	onClose (WebSocket, int, String, boolean)	void
m	onStart ()	void
m	initSSL()	void
m	broadcast (int, boolean)	void
m	setStrategy (Integer, Map<String, Object>)	void

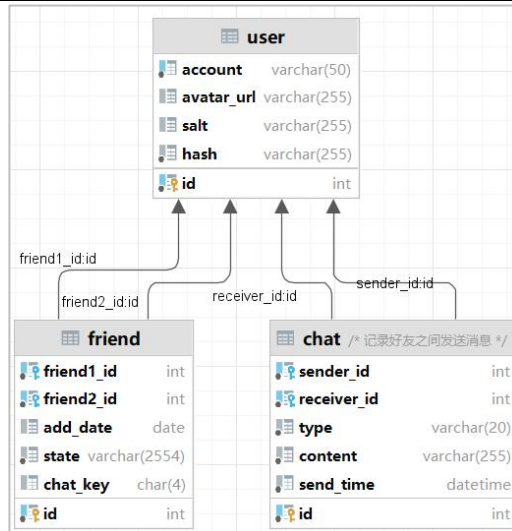
可以看到，Server 类内部持有 4 个 public 修饰的静态 map 对象，这 4 个 map 记录了客户端用户在线情况：

- onlineUsers 记录登录进系统的用户；
- onlineChats 记录打开聊天窗口的用户；
- onlineChatsOfUploadFile 记录打开文件上传窗口的用户；
- onlineNotices 记录打开通知窗口的用户。

服务器可以根据这 4 个 map 去对客户端的情况进行更新，比如更新用户好友的在线情况。

### 4.3 数据库设计

由于本项目侧重点在通信部分，所以数据库设计简单，只有 3 个实体：user、friend、chat，记录了一些必要的信息，E-R 图如下：



值得一提的是，user 表中并没有用户的 password 字段，而是用 salt 和 hash 字段代替。这使用了密码学中的单向哈希，服务端不是从数据库中直接获取用户密码之后再与用户输入的密码进行比对，而是通过 salt 值与用户输入的 password 经过一个单向哈希函数之后得到的哈希值与数据库中的 hash 值进行比对，这使得即使有人恶意窃取数据库内容之后，也无法获知用户的密码，从而保护了用户的安全。

## 4.4 通信协议设计

### 4.4.1 报文格式

通信报文分为两类，一类是正常通信报文（软件业务相关），一类是关闭连接报文。

#### ➤ 正常通信报文

报文格式为：type=()&category=()&data，其中 type 标识是请求还是响应，category 标识请求（或响应）类型。

type 和 category 均根据 CONST 类的常量进行赋值。CONST 类是我定义的一个专门用来存储常量的类，类图如下：

CONST	
NORMAL_STATE	String
chatClient	String
SALT_LENGTH	int
SERVER_PORT	int
INIT_CHAT	int
INIT_UPLOAD_FILE	int
UPLOAD_FILE	int
FRIEND_STATE_CHANGED	int
SEND_MSG	int
KEY_STORE_PASSWORD	String
FIND_USER	int
DELD_FRIEND	int
RESPONSE	int
AVATAR_URL_PREFIX	String
REQUEST	int
REGISTER	int
FIND_FRIEND	int
loginClient	String
registerClient	String
fileUploadClient	String
REPLAY_REQUEST	int
REQUEST_NOTICE	int
noticeClient	String
ADD_FRIEND_REQUEST	int
mainClient	String
LOGIN	int
ADD_FRIEND_REQUEST_ACCEPT	int
SAFE_CLOSE	int
DEL_FRIEND	int
SERVER_KEY_STORE_FILE_PATH	String

而对于数据 data，是 Result 类返回体的字符串表示，每个字段之间也是使用&进行分隔，Result 类图如下：

Result<T>	
f	code int
f	data T
p	data T
p	code int
m	failed(T) Result
m	toString() String
m	succeed(T) Result
m	succeed() Result
m	failed() Result
m	Result(int, T)
m	Result(int)

可以看到 data 是使用泛型进行定义的，返回体包含一个 code 码，标识客户端操作是否成功。

### ➤ 关闭连接报文

关闭连接的发送使用 WebSocketClient 类的 close 方法，参数有两个，一个是关闭原因（int 类型），我在程序中全部设置为 1000（正常关闭）；另一个



参数是一个字符串，我把它定义为报文的内容，格式为：(clientType)：(客户端对象标识)，其中 clientType 为客户端类型，取值范围是 {loginClient, registerClient, mainClient, chatClient, fileUploadClient, noticeClient}，对应于 6 个 BaseClient 的子类，而客户端对象标识在每个 application 则不同：

- ✧ 当 clientType=loginClient/registerClient 时，无客户端对象标识字段
- ✧ 当 clientType=mainClient/noticeClient 时，客户端对象标识为当前登录用户的 id
- ✧ 当 clientType=chatClient/fileUploadClient 时，客户端对象标识为当前登录用户 id+"-"+聊天对象用户 id

关闭连接报文只能由客户端发送给服务端。

#### 4.4.2 通信规则

##### ➤ 正常通信

客户端通过 socket 向服务端发送报文后，服务端接收到报文，根据报文的 category 字段去调用相应的策略，策略算法返回一个 Result 对象给 Server，Server 将 type 和 category 字段加在 Result 对象的字符串表示之前，然后通过相同的 socket 发送给客户端。

##### ➤ 关闭连接

当客户端应用程序关闭时，先向服务端发送一条关闭连接报文，服务端根据报文内容作出相应的动作：

- ✧ 当 clientType=loginClient/registerClient 时，服务端不做动作
- ✧ 当 clientType=mainClient 时，在 onlineUsers 中删除键值为客户端对象标识的节点，并向该用户的好友广播该用户的离线状态
- ✧ 当 clientType=chatClient 时，在 onlineChats 中删除键值为客户端对象标识的节点
- ✧ 当 clientType=fileUploadClient 时，在 onlineChatsOfUploadFile 中删除键值为客户端对象标识的节点
- ✧ 当 clientType=noticeClient 时，在 onlineNotices 中删除键值为客户端对象标识的节点



## 5 核心功能代码说明

### 5.1 通信机制

每个 application 对象持有对应的 client 对象，每次向 server 端发送报文时均使用 client 对象进行发送，例如：

```
1. try {
2.     if (!loginClient.isOpen()){
3.         loginClient.connectBlocking();
4.     }
5. } catch (InterruptedException ex) {
6.     ex.printStackTrace();
7.     System.out.println("连接失败");
8.     MsgBox.showDialog(Alert.AlertType.ERROR, "连接失败", "请检查网络连接", false);
9. }
10. if (loginClient.isOpen()) {
11.     System.out.println("连接成功");
12.     String message = "type="+CONST.REQUEST+"&category="+CONST.LOGIN+"&account="+account+"&password="+password;
13.     loginClient.send(message);
14.     MsgBox.showDialog(Alert.AlertType.INFORMATION, "登录中", "请稍后", true);
15. } else {
16.     System.out.println("连接失败");
17.     MsgBox.showDialog(Alert.AlertType.ERROR, "连接失败", "请检查网络连接", false);
18. }
```

客户端在接收到服务端发来的消息之后，立马转发给相应的 application 类进行处理：

```
1. /**
2.  * 接收服务端消息并转发给应用程序处理
3.  * @param message
4.  */
5. public void onMessage(String message) {
6.     System.out.println(this.getClass().getName()+"received message: "+message+"\n转发给应用程序处理");
7.     application.receiveMessageFromServer(message);
8. }
```

服务端在 onMessage 方法中，每接收到客户端发来的消息后里面使用 Paser

类进行解析，解析成一个 map，该 map 就是策略模式下算法执行的参数。当然，这个 map 也包含了消息类型，通过该消息类型决定执行哪个策略。

```

1. public void onMessage(WebSocket websocket, String s) {
2.     System.out.println("原始信息: "+s);
3.     System.out.println("收到客户
   端 "+websocket.getRemoteSocketAddress().getAddress()+
4.         ":"+websocket.getRemoteSocketAddress().getPort()+" 发
   送过来的消息: "+s);
5.     // 向客户端回馈消息
6.     Map<String,Object> parameters= Parser.getAttr(s);
7.     int key=Parser.getCategory(s);
8.     parameters.put("websocket",websocket);
9.     System.out.println("客户端请求的操作类型: "+key);
10.    setStrategy(key,parameters);
11.    Result result=doOperation();
12.    // 打印信息
13.    if(result!=null){
14.        String response="type="+ CONST.RESPONSE+"&category="+key+
   "&"+result;
15.        System.out.println(response);
16.        websocket.send(response);
17.        System.out.println("发送完毕"+response);
18.    }else {
19.        websocket.send("客户端操作失败! ");
20.    }
21. }

```

初始化策略表:

```

1. private void initStrategy(){
2.     System.out.println("初始化策略");
3.     strategyMap.put(LOGIN, LoginStrategy::new);
4.     strategyMap.put(REGISTER, RegisterStrategy::new);
5.     strategyMap.put(FIND_FRIEND, FindFriendsStrategy::new);
6.     strategyMap.put(FIND_USER, FindUsersStrategy::new);
7.     strategyMap.put(ADD_FRIEND_REQUEST, AddFriendRequestStrategy::
   new);
8.     strategyMap.put(REPLAY_REQUEST, ReplyAddFriendStrategy::new);
9.     strategyMap.put(INIT_CHAT, InitChatStrategy::new);
10.    strategyMap.put(SEND_MSG, SendMsgStrategy::new);
11.    strategyMap.put(DEL_FRIEND, DelFriendStrategy::new);
12.    strategyMap.put(INIT_UPLOAD_FILE,InitUploadFileStrategy::new)
   ;
13.    strategyMap.put(UPLOAD_FILE,UploadFileStrategy::new);
14.    strategyMap.put(REQUEST_NOTICE,InitNoticeStrategy::new);

```

```
15.      System.out.println("初始化策略完成");
16.  }
```

选择策略并执行策略：

```
1.      private void setStrategy(Integer key,Map<String,Object> param
eters){
2.          System.out.println("设置策略中: key="+key+"参数
="+parameters.toString());
3.          Function<Map<String,Object>,Strategy> function=strategyMa
p.getOrDefault(key,params->{
4.              System.out.println("未找到策略! ");
5.              return new Strategy() {
6.                  @Override
7.                  public Result execute() {
8.                      return null;
9.                  }
10.             };
11.         });
12.         strategy=function.apply(parameters);
13.     }
14.     private Result doOperation(){
15.         if(strategy!=null){
16.             return strategy.execute();
17.         }else{
18.             System.out.println("未设置策略");
19.             return null;
20.         }
21.     }
```

## 5.2 登录功能

### • 客户端发送登录请求

在登录应用程序内为登录按钮添加监听器，一旦登录按钮被点击，则获取账号输入框和密码输入框的内容，坐空值校验之后将校验后的数据利用持有的 client 对象发送给服务端：

```
1.  // 设置登录按钮
2.  login = (Button) namespace.get("login");
3.  login.setOnMouseClicked(e -> {
4.      String password = passwordField.getText();
5.      String account = accountField.getText();
6.      if (password != null && !"".equals(password) && account != nu
ll && !"".equals(account)) {
7.          System.out.println("登录请求");
```

```

8.         try {
9.             if (!loginClient.isOpen()){
10.                 loginClient.connectBlocking();
11.             }
12.         } catch (InterruptedException ex) {
13.             ex.printStackTrace();
14.             System.out.println("连接失败");
15.             MsgBox.showDialog(Alert.AlertType.ERROR, "连接失败", "请检查网络连接", false);
16.         }
17.         if (loginClient.isOpen()) {
18.             System.out.println("连接成功");
19.             String message = "type="+CONST.REQUEST+"&category="+CONST.LOGIN+"&account="+account+"&password="+password;
20.             loginClient.send(message);
21.             MsgBox.showDialog(Alert.AlertType.INFORMATION, "登录中", "请稍后", true);
22.         } else {
23.             System.out.println("连接失败");
24.             MsgBox.showDialog(Alert.AlertType.ERROR, "连接失败", "请检查网络连接", false);
25.         }
26.     } else {
27.         MsgBox.showDialog(Alert.AlertType.ERROR, "登录失败", "账号和密码均不能为空", false);
28.     }
29. });

```

#### • 服务端接收登录请求

服务端由于使用了策略模式，收到了客户端发来的消息后，调用 LoginStrategy 的 execute 方法，调用 DAO 层接口查找用户信息，返回结果并由服务器对象将结果返回给客户端

```

1. public class LoginStrategy extends Strategy {
2.     public LoginStrategy(Map<String, Object> parameters){
3.         super(parameters);
4.     }
5.     @Override
6.     public Result execute() {
7.         System.out.println("LoginStrategy execute");
8.         User user = UserDAO.login((String)params.get("account"));
9.         if(user==null)return Result.failed(null);
10.        Integer id=user.getId();
11.        //不能同时登录同一个号
12.        synchronized (Server.onlineUsers){

```

```

13.         if (Server.onlineUsers.containsKey(id)) return Result.
           failed("不能登录同一个号");
14.     }
15.     String password = (String)params.get("password");
16.     String slat= user.getSalt();
17.     String hash=user.getHash();
18.     String inputHash= Security.hash(password, slat);
19.     System.out.println(inputHash+","+hash);
20.     if(!hash.equals(inputHash))return Result.failed(null);
21.     user.setHash(null);
22.     user.setSalt(null);
23.     return Result.succeed(user);
24. }
25. }

```

这其中调用了 DAO 层的方法：

```

1. public static User login(String account) {
2.     User user = null;
3.     try {
4.         connection = ConnectUtil.getConnection();
5.         preparedStatement = connection.prepareStatement("select *
           from user where user.account=?");
6.         preparedStatement.setString(1, account);
7.         resultSet = preparedStatement.executeQuery();
8.         while (resultSet.next()) {
9.             user = new User();
10.            user.setId(resultSet.getInt("id"));
11.            user.setAccount(resultSet.getString("account"));
12.            user.setHash(resultSet.getString("hash"));
13.            user.setSalt(resultSet.getString("salt"));
14.            user.setAvatarUrl(resultSet.getString("avatar_url"));
15.            break;
16.        }
17.    } catch (SQLException e) {
18.        throw new RuntimeException(e);
19.    }finally {
20.        close();
21.    }
22.    return user;
23. }

```

客户端接收消息，将消息转发给相应的用户登录程序窗口进行处理，应用程序接收到消息后，若登录成功则进入聊天室主界面，反之则弹出对话框提示登录失败：

```

1. public void receiveMessageFromServer(String message) {

```

```

2.      System.out.println("receiveMessageFromServer: " + message);
3.      Map<String, Object> map = Parser.getResponseData(message);
4.      if (map.containsKey("code")){
5.          if (Integer.parseInt((String) map.get("code"))==0){
6.              MsgDialog.showDialog(Alert.AlertType.ERROR, "登录失败
", "账号或密码错误", false);
7.          }else{
8.              User user = new User();
9.              user.setId(Integer.parseInt((String) map.get("id")));
10.             user.setAccount((String) map.get("account"));
11.             user.setAvatarUrl((String) map.get("avatarUrl"));
12.             System.out.println("登录成功! "+user);
13.             try {
14.                 Platform.runLater()->{
15.                     try {
16.                         new MainApplication(user).start(new Stage
17.                             ());
18.                         handleClose();
19.                         this.stage.close();
20.                     } catch (Exception e) {
21.                         e.printStackTrace();
22.                         System.out.println("进入主界面失败");
23.                         MsgDialog.showDialog(Alert.AlertType.ERROR, "进入
24.                             主界面失败", "", false);
25.                     }
26.                 });
27.             } catch (Exception e) {
28.                 e.printStackTrace();
29.                 MsgDialog.showDialog(Alert.AlertType.ERROR, "进入
30.                     主界面失败", "", false);
31.             }
32.         }else{
33.             System.out.println("receiveMessageFromServer: 未知消息");
34.             MsgDialog.showDialog(Alert.AlertType.ERROR, "登录出错", "请
35.                 稍后重试", false);
36.         }
37.     }

```

## 5.3 注册功能

给登录界面的注册按钮添加鼠标监听事件，一旦点击后则进入注册界面

```

1. register = (Button) namespace.get("register");
2. register.setOnMouseClicked(e -> {
3.     RegisterApplication registerApplication = new RegisterApplica
        tion();
4.     try {
5.         registerApplication.start(new Stage());
6.         handleClose();
7.         stage.close();
8.     } catch (Exception ex) {
9.         ex.printStackTrace();
10.        MsgBox.showDialog(Alert.AlertType.ERROR, "进入注册界面
        失败", "", false);
11.    }
12.    System.out.println("进入注册界面");
13. });

```

在注册界面为注册按钮添加鼠标监听事件，一旦点击，则获取账号密码以及头像文件名称，对其进行校验之后将校验后的结果传入服务端

```

1. submitBtn = (Button) namespace.get("submitBtn");
2. submitBtn.setOnMouseClicked(e -> {
3.     String account = accountFiled.getText();
4.     String password = passwordField.getText();
5.     String confirmPassword = confirmPasswordField.getText();
6.     String avatarName = ((Map<String, String>) clip.getUserData())
        .get("avatarName");
7.     System.out.println("account:" + account);
8.     System.out.println("password:" + password);
9.     System.out.println("confirm:" + confirmPassword);
10.    System.out.println("avatarName:" + avatarName);
11.    if (account == null || password == null || account.equals("")
        || password.equals("")) {
12.        System.out.println("输入为空!");
13.    } else {
14.        if (avatarName == null){
15.            MsgBox.showDialog(Alert.AlertType.ERROR, "请上传头
        像", "", false);
16.            return;
17.        }
18.        if (confirmPassword != null && confirmPassword.equals(pas
        sword)) {
19.            System.out.println("注册请求发送中...");
20.            try {
21.                if (!registerClient.isOpen()){
22.                    registerClient.connectBlocking();
23.                }

```



```

24.         } catch (InterruptedException ex) {
25.             ex.printStackTrace();
26.             System.out.println("连接失败");
27.             MsgDialog.showDialog(Alert.AlertType.ERROR, "连接
失败", "无法连接到服务器", false);
28.         }
29.         if (registerClient.isOpen()) {
30.             String request="type=" + CONST.REGISTER + "&categ
ory=" + CONST.REGISTER +
31.                 "&account=" + account + "&password=" + pa
ssword +
32.                 "&avatarName=" + avatarName;
33.             registerClient.send(request);
34.             MsgDialog.showDialog(Alert.AlertType.INFORMATION,
"注册请求发送成功", "请等待", true);
35.         } else {
36.             System.out.println("连接失败");
37.             MsgDialog.showDialog(Alert.AlertType.ERROR, "连接
失败", "无法连接到服务器", false);
38.         }
39.     } else {
40.         System.out.println("前后密码不一致");
41.         MsgDialog.showDialog(Alert.AlertType.ERROR, "密码不一
致", "前后密码不一致", false);
42.     }
43. }
44. });

```

在这之前还实现了头像上次的功能：为上次头像的按钮添加鼠标监听事件，一旦被点击则创建一个 FileChooser 类的对象，弹出文件选择框，用户选择后则将该文件利用 JSch 包实现 SFTP 上传文件至阿里云服务器中。

```

1.  // 头像上传按钮设置
2.  uploadImgBtn = (Button) namespace.get("upload_img");
3.  Map<String, String> filenameMap = new HashMap<>();
4.  clip.setUserData(filenameMap);
5.  uploadImgBtn.setOnMouseClicked(e -> {
6.      FileChooser fileChooser = new FileChooser();
7.      fileChooser.setTitle("请选择头像");
8.      fileChooser.getExtensionFilters().addAll(
9.          new FileChooser.ExtensionFilter("图片文件
", "*.jpg", "*.jpeg", "*.png")
10.     );
11.     File selectedFile = fileChooser.showOpenDialog(stage);
12.     String filename = null;

```

```

13.     if (selectedFile != null) {
14.         String[] split = selectedFile.getName().split("\\.");
15.         String fix = split[split.length - 1];
16.         boolean flag = true;
17.         try {
18.             filename = UUID.randomUUID().toString() + "." + fix;
19.             FileUploader.sshSftpUpload(selectedFile, filename);
20.             MsgDialog.showDialog(Alert.AlertType.INFORMATION, "图片
    上传中...", "请稍后", false);
21.         } catch (Exception ex) {
22.             ex.printStackTrace();
23.             flag = false;
24.         }
25.         if (flag) {
26.             clip.setFill(new ImagePattern(
27.                 new Image(selectedFile.toURI().toString())));
28.             filenameMap.put("avatarName", filename);
29.         } else {
30.             MsgDialog.showDialog(Alert.AlertType.ERROR, "图片上传
    失败", "", false);
31.             filename = null;
32.         }
33.     }
34. });

```

#### 文件上传方法

```

1.  /**
2.   * 利用 JSch 包实现 SFTP 上传文件
3.   * @param file 文件
4.   * @param fileName 文件名
5.   * @throws Exception
6.   */
7.  public static void sshSftpUpload(File file, String fileName) thro
    ws Exception{
8.      Session session = null;
9.      Channel channel = null;
10.     JSch jsch = new JSch();
11.     if(CONST.UPLOAD_IMG_PORT <=0){
12.         //连接服务器, 采用默认端口
13.         session = jsch.getSession(CONST.UPLOAD_IMG_USERNAME, CONS
    T.UPLOAD_IMG_URL);
14.     }else{
15.         //采用指定的端口连接服务器
16.         session = jsch.getSession(CONST.UPLOAD_IMG_USERNAME, CONS
    T.UPLOAD_IMG_URL ,CONST.UPLOAD_IMG_PORT);

```

```

17.     }
18.     //如果服务器连接不上，则抛出异常
19.     if (session == null) {
20.         throw new Exception("session is null");
21.     }
22.     //设置登陆主机的密码
23.     session.setPassword(CONST.UPLOAD_IMG_PASSWORD); //设置密码
24.     //设置第一次登陆的时候提示，可选值: (ask | yes | no)
25.     session.setConfig("StrictHostKeyChecking", "no");
26.     //设置登陆超时时间
27.     session.connect(30000);
28.     OutputStream outstream = null;
29.     try {
30.         //创建 sftp 通信通道
31.         channel = (Channel) session.openChannel("sftp");
32.         channel.connect(1000);
33.         ChannelSftp sftp = (ChannelSftp) channel;
34.         //进入服务器指定的文件夹
35.         sftp.cd(CONST.UPLOAD_IMG_PATH);
36.         //以下代码实现从本地上传一个文件到服务器，如果要实现下载，对换以下流就可以了
37.         outstream = sftp.put(fileName);
38.         outstream.write(fileToByteArray(file));
39.     } catch (Exception e) {
40.         e.printStackTrace();
41.     } finally {
42.         //关流操作
43.         if(outstream != null){
44.             outstream.flush();
45.             outstream.close();
46.         }
47.         if(session != null){
48.             session.disconnect();
49.         }
50.         if(channel != null){
51.             channel.disconnect();
52.         }
53.     }
54. }

```

服务端接收到注册请求后进行处理

```

1. public class RegisterStrategy extends Strategy {
2.     public RegisterStrategy(Map<String, Object> parameters){
3.         super(parameters);
4.     }

```

```

5.
6.     @Override
7.     public Result execute() {
8.         System.out.println("RegisterStrategy execute");
9.         String account= (String) params.get("account");
10.        String password= (String) params.get("password");
11.        String salt = Security.generateSalt(CONST.SALT_LENGTH);
12.        String hash=Security.hash(password,salt);
13.        if (hash==null){
14.            return Result.failed(null);
15.        }
16.        boolean res = UserDAO.register(account, hash, salt, (String) params.get("avatarName"));
17.        if(res){
18.            return Result.succeed(null);
19.        }
20.        return Result.failed(null);
21.    }
22. }

```

这里调用了 DAO 层的方法：

```

1. public static boolean register(String account, String hash,String
   salt,String avatarName) {
2.     try {
3.         connection = ConnectUtil.getConnection();
4.         preparedStatement = connection.prepareStatement(
5.             "insert into user(account,hash,avatar_url,salt) v
   alues(?,?,?,?)");
6.         preparedStatement.setString(1, account);
7.         preparedStatement.setString(2, hash);
8.         preparedStatement.setString(3, CONST.AVATAR_URL_PREFIX+av
   atarName);
9.         preparedStatement.setString(4,salt);
10.        int i = preparedStatement.executeUpdate();
11.        return i >= 1;
12.    } catch (Exception e) {
13.        e.printStackTrace();
14.    }finally {
15.        close();
16.    }
17.    return false;
18. }

```

客户端接收到反馈消息后进行相应提示：

```

1. public void receiveMessageFromServer(String message) {
2.     System.out.println("receive message from server: "+ message);

```

```
3.      Map<String, Object> map = Parser.getResponseData(message);
4.      if(map.containsKey("code")){
5.          if (Integer.parseInt(map.get("code").toString()) == 0){
6.              System.out.println("注册失败");
7.              MsgDialog.showDialog(Alert.AlertType.ERROR, "注册失败",
8.                  "账号已存在", false);
9.          }else{
10.             System.out.println("注册成功");
11.             MsgDialog.showDialog(Alert.AlertType.INFORMATION, "注册
成功",
12.                 "注册成功, 请登录", false);
13.         }
14.     }else{
15.         System.out.println("receiveMessageFromServer: 未知消息");
16.         MsgDialog.showDialog(Alert.AlertType.ERROR, "注册出错",
17.             "请稍后重试", false);
18.     }
19. }
```

## 5.4 添加、删除好友功能

对应用程序主体界面的搜索按钮添加监听器:

```
1.  searchBtn.setOnMouseClicked(e -> {
2.      String text = searchContent.getText();
3.      if(text.isEmpty()){
4.          requestFriendList();
5.      }else{
6.          try {
7.              if (!mainClient.isOpen()) {
8.                  mainClient.connectBlocking();
9.              }
10.         }catch (InterruptedException eX) {
11.             eX.printStackTrace();
12.             System.out.println("连接失败");
13.             MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败
", "无法连接到服务器", false);
14.         }
15.         if (mainClient.isOpen()){
16.             String request = "type=" + CONST.REQUEST + "&category
=" + CONST.FIND_USER + "&currUserId=" + user.getId() + "&searchAcc
ount=" + text;
17.             System.out.println(request);
18.             mainClient.send(request);
19.             System.out.println("请求查找用户");
```

```

20.         MsgDialog.showDialog(Alert.AlertType.INFORMATION, "请
           求查找用户", "正在请求服务器，请稍候", true);
21.     }else {
22.         System.out.println("连接失败");
23.         MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败
           ", "无法连接到服务器", false);
24.     }
25. }
26. });

```

服务端接收到消息后查找用户：

```

1.  public class FindUsersStrategy extends Strategy {
2.      public FindUsersStrategy(Map<String, Object> params) {
3.          super(params);
4.      }
5.      @Override
6.      public Result execute() {
7.          System.out.println("FindUsersStrategy execute");
8.          MyList<User> users= UserDao.findUsers(params.get("searchA
           ccount").toString());
9.          Integer currUserId=Integer.parseInt(params.get("currUserI
           d").toString());
10.         class UserAndIsFriendAndIsOnline{
11.             User user;
12.             boolean isFriend;
13.             boolean isOnline;
14.             public String toString(){
15.                 return user.toString()+"&isFriend="+isFriend+"&is
           Online="+isOnline;
16.             }
17.         }
18.         MyList<UserAndIsFriendAndIsOnline> userAndIsFriendAndIsOn
           lines=new MyList<>();
19.         for (User user : users) {
20.             if (Objects.equals(user.getId(), currUserId)) continu
           e;
21.             UserAndIsFriendAndIsOnline u=new UserAndIsFriendAndIs
           Online();
22.             u.user=user;
23.             u.isFriend= FriendDAO.isFriend(currUserId, user.getId
           ());
24.             u.isOnline= Server.onlineUsers.containsKey(user.getId
           ());
25.             userAndIsFriendAndIsOnlines.add(u);
26.         }

```

```
27.         return Result.succeed(userAndIsFriendAndIsOnlines);
28.     }
29. }
```

服务端将结果反馈给客户端后，客户端界面刷新，显示查找到的用户列表，并在列表中为按钮添加相应的监听器（删除或添加监听）：

```
1. private void initFindUsersListView(String s){
2.     currList=USERS;
3.     Platform.runLater(() -> {
4.         // 更新 UI 的代码放在这里
5.         listView.getItems().clear();
6.     });
7.     List<Map<String, String>> usersAndIsFriendAndIsOnline = Parser.getList(s);
8.     usersAndIsFriendAndIsOnline.forEach(userAndIsFriendAndIsOnline -> {
9.         FXMLLoader loader = new FXMLLoader(getClass().getClassLoader().getResource("userpane.fxml"));
10.        try {
11.            Pane pane = loader.load();
12.            ((UserPaneController) loader.getController()).setAccountLabelText(userAndIsFriendAndIsOnline.get("account"));
13.            boolean isOnline=Boolean.parseBoolean(userAndIsFriendAndIsOnline.get("isOnline"));
14.            ((UserPaneController) loader.getController()).setViewImage(new Image(userAndIsFriendAndIsOnline.get("avatarUrl")),isOnline);
15.            boolean isFriend = Boolean.parseBoolean(userAndIsFriendAndIsOnline.get("isFriend"));
16.            System.out.println("isfriend:" + isFriend);
17.            ((UserPaneController) loader.getController()).setButton(isFriend);
18.            Platform.runLater(() -> {
19.                // 更新 UI 的代码放在这里
20.                listView.getItems().add(pane);
21.            });
22.            // 不是好友，则添加添加好友按钮
23.            if (!isFriend){
24.                ((UserPaneController) loader.getController()).setAddListener(e -> {
25.                    if (e.getButton() == MouseButton.PRIMARY) {
26.                        try {
27.                            new AddFriendDialog(Integer.parseInt(userAndIsFriendAndIsOnline.get("id")),this).start(new Stage());
28.                        } catch (Exception ex) {
```



```

29.                ex.printStackTrace();
30.                MsgDialog.showDialog(Alert.AlertType.
    ERROR, "程序错误", "窗口打开失败", false);
31.            }
32.        }
33.    });
34.    }else { // 是好友，则添加删除好友按钮
35.        ((UserPaneController) loader.getController()).set
    RemoveListener(e -> {
36.            if (e.getButton() == MouseButton.PRIMARY) {
37.                try {
38.                    if (!mainClient.isOpen()) {
39.                        mainClient.connectBlocking();
40.                    }
41.                } catch (InterruptedException ex) {
42.                    ex.printStackTrace();
43.                    System.out.println("连接失败");
44.                    MsgDialog.showDialog(Alert.AlertType.
    ERROR, "连接失败", "无法连接到服务器", false);
45.                }
46.                if (mainClient.isOpen()) {
47.                    String request = "type=" + CONST.REQU
    EST + "&category=" + CONST.DEL_FRIEND + "&id1=" + userAndIsFriendA
    ndIsOnline.get("id") + "&id2=" + this.user.getId();
48.                    mainClient.send(request);
49.                    MsgDialog.showDialog(Alert.AlertType.
    INFORMATION, "请求删除好友", "正在请求服务器，请稍候", true);
50.                    currRemoveFriendBtn=((UserPaneControl
    ler) loader.getController()).getRemoveBtn();
51.                } else {
52.                    System.out.println("连接失败");
53.                    MsgDialog.showDialog(Alert.AlertType.
    ERROR, "连接失败", "无法连接到服务器", false);
54.                }
55.            }
56.        });
57.    }
58.    } catch (Exception e) {
59.        e.printStackTrace();
60.    }
61.    });
62. }

```

其中用户的添加步骤为：点击添加按钮，随后会弹出一个对话框，在对话框输入聊天密钥（4 位），等到对方同意即可：

```

1.  confirmBtn.setOnMouseClicked(e->{
2.      String key=chatKey.getText();
3.      String confirm=confirmKey.getText();
4.      if (key.length()!=4){
5.          MsgDialog.showDialog(Alert.AlertType.ERROR,"设置错误","聊
           天密钥长度必须为 4 位",false);
6.      } else if (!key.equals(confirm)) {
7.          MsgDialog.showDialog(Alert.AlertType.ERROR,"设置错误","聊
           天密钥不一致",false);
8.      } else{
9.          mainApplication.receiveAddFriendRequestFromDialog(otherId,
           key);
10.         handleClose();
11.         stage.close();
12.     }
13. });

```

主界面接收到对话框传来的参数后，向服务端发送请求：

```

1.  public void receiveAddFriendRequestFromDialog(int userId,String c
           hatKey){
2.      try{
3.          if (!mainClient.isOpen()){
4.              mainClient.connectBlocking();
5.          }
6.      }catch (InterruptedException e){
7.          e.printStackTrace();
8.          MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败", "
           无法连接到服务器",false);
9.      }
10.     if (mainClient.isOpen()){
11.         String request="type="+ REQUEST+"&category="+ CONST.ADD_F
           RIEND_REQUEST + "&requestUserId="+user.getId()+"&targetUserId="+use
           rId+"&chatKey="+chatKey;
12.         mainClient.send(request);
13.         MsgDialog.showDialog(Alert.AlertType.INFORMATION,"发送请求
           中","等待对方同意即可",false);
14.     }else{
15.         System.out.println("连接失败");
16.         MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败", "
           无法连接到服务器",false);
17.     }
18. }

```

服务端接收后调用相应策略，不仅要向数据库插入数据，也要向对方的通知中加入此条添加好友申请，等待对方同意：

```

1.  public class AddFriendRequestStrategy extends Strategy {
2.
3.      public AddFriendRequestStrategy(Map<String, Object> params) {
4.          super(params);
5.      }
6.      @Override
7.      public Result execute() {
8.          System.out.println("AddFriendRequestStrategy execute");
9.          Integer requestUserId= Integer.parseInt(params.get("requestUserId").toString());
10.         Integer targetUserId= Integer.parseInt(params.get("targetUserId").toString());
11.         String chatKey= (String) params.get("chatKey");
12.         boolean b = FriendDAO.addFriendRequest(requestUserId, targetUserId, chatKey);
13.         if (b) {
14.             synchronized (Server.onlineNotices){
15.                 if (Server.onlineNotices.containsKey(targetUserId))
16.                 ){
17.                     WebSocket socket=Server.onlineNotices.get(targetUserId);
18.                     UserAndChatKey u= UserDAO.findUserAndChatKey(targetUserId,requestUserId);
19.                     String message="type="+ CONST.RESPONSE+
20.                                     "&category="+CONST.ADD_FRIEND_REQUEST
21.                                     +"&code=1"+"&"+u;
22.                     socket.send(message);
23.                 }
24.             }
25.             return Result.succeed(null);
26.         }
27.         return Result.failed();
28.     }
29. }

```

在对方通知界面中, 用户点击删除或同意按钮即可拒绝或同意此次申请:

```

1.  private void sendAcceptOrRefuseToOther(int userId, boolean isAccepted){
2.      try{
3.          if (!noticeClient.isOpen()){
4.              noticeClient.connectBlocking();
5.          }
6.      }catch (InterruptedException e){
7.          e.printStackTrace();
8.      }
9.  }

```

```

8.         MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败", "
           无法连接到服务器", false);
9.     }
10.    if (noticeClient.isOpen()){
11.        String request="type="+ REQUEST+"&category="+CONST.REPLAY
           _REQUEST+
12.            "&targetUserId="+me.getId()+"&requestUserId="+use
           rId+"&isAccept="+isAccepted;
13.        System.out.println("是否添加: "+request);
14.        noticeClient.send(request);
15.        MsgDialog.showDialog(Alert.AlertType.INFORMATION, "答复对
           方中", "正在请求服务器, 请稍候", true);
16.    }else{
17.        System.out.println("连接失败");
18.        MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败", "
           无法连接到服务器", false);
19.    }
20. }

```

服务端接收到该消息后调用相应策略:

```

1. public class ReplyAddFriendStrategy extends Strategy {
2.     public ReplyAddFriendStrategy(Map<String,Object> params){
3.         super(params);
4.     }
5.
6.     @Override
7.     public Result execute() {
8.         System.out.println("AcceptAddFriendStrategy execute");
9.         Integer targetUserId= Integer.parseInt(params.get("target
           UserId").toString());
10.        Integer requestUserId= Integer.parseInt(params.get("reque
           stUserId").toString());
11.        boolean isAccept=Boolean.parseBoolean(params.get("isAccep
           t").toString());
12.        if (isAccept){
13.            boolean b= FriendDAO.addFriendTruly(targetUserId, requ
           estUserId);
14.            if (b) {
15.                // 同意成功
16.                synchronized (Server.onlineUsers){
17.                    if(Server.onlineUsers.containsKey(requestUser
           Id)){
18.                        WebSocket socket=Server.onlineUsers.get(r
           equestUserId);

```

```

19.         UserAndChatKey u=UserDAO.findUserAndChatKey(
            requestUserId,targetUserId);
20.         UserAndIsOnline userAndIsOnline=new UserAndIsOnline();
21.         userAndIsOnline.setOnline(true);
22.         userAndIsOnline.setUserAndChatKey(u);
23.         String message="type="+ CONST.RESPONSE+
24.             "&category="+CONST.ADD_FRIEND_REQUEST_ACCEPT+"&code=1"+"&"+userAndIsOnline;
25.         System.out.println("发送给对面: "+message);
26.         socket.send(message);
27.     }
28.     //自己也需要添加对方条目
29.     if(Server.onlineUsers.containsKey(targetUserId)){
30.         WebSocket socket=Server.onlineUsers.get(targetUserId);
31.         UserAndChatKey u=UserDAO.findUserAndChatKey(targetUserId,requestUserId);
32.         UserAndIsOnline userAndIsOnline=new UserAndIsOnline();
33.         userAndIsOnline.setOnline(true);
34.         userAndIsOnline.setUserAndChatKey(u);
35.         String message="type="+ CONST.RESPONSE+
36.             "&category="+CONST.ADD_FRIEND_REQUEST_ACCEPT+"&code=1"+"&"+userAndIsOnline;
37.         socket.send(message);
38.     }
39. }
40. synchronized (Server.onlineNotices){
41.     if (Server.onlineNotices.containsKey(requestUserId)){
42.         WebSocket socket=Server.onlineNotices.get(requestUserId);
43.         String message="type="+CONST.RESPONSE+
44.             "&category="+CONST.ADD_FRIEND_REQUEST_ACCEPT+"&code=1"+"&targetUserId="+targetUserId;
45.         socket.send(message);
46.     }
47. }
48. return Result.succeed("&requestUserId="+requestUserId);
49. }
50. return Result.failed();
51. }else{

```

```

52.          // 拒绝
53.          boolean b=FriendDAO.delAddFriendRequest(targetUserId,
requestUserId);
54.          if (b) {
55.              return Result.succeed("&requestUserId="+requestUs
erId);
56.          }
57.          return Result.failed();
58.      }
59.  }
60. }

```

在回复成功后，用户通知界面该申请条目将会消失，同时申请者主界面也会有相应的变化（被拒绝则无变化，同意则会在好友列表出现该用户的条目），此处便不一一展示代码了。

相应的删除好友策略如下：

```

1.  public class DelFriendStrategy extends Strategy {
2.      public DelFriendStrategy(Map<String, Object> params) {
3.          super(params);
4.      }
5.      @Override
6.      public Result execute() {
7.          System.out.println("删除好友策略执行");
8.          Integer id1= Integer.parseInt(params.get("id1").toString()
);
9.          Integer id2= Integer.parseInt(params.get("id2").toString()
);
10.         boolean b = FriendDAO.delFriend(id1, id2);
11.         if (b) {
12.             synchronized (Server.onlineUsers){
13.                 if (Server.onlineUsers.containsKey(id1)){
14.                     //在 id1 的界面中删除 id2 好友
15.                     String response="type="+ CONST.RESPONSE+"&cat
egory="+CONST.DELD_FRIEND+"&code=1"+"&userId="+id2;
16.                     WebSocket socket=Server.onlineUsers.get(id1);
17.                     socket.send(response);
18.                 }
19.             }
20.             return Result.succeed(null);
21.         }
22.         else return Result.failed();
23.     }
24. }

```

## 5.5 实时聊天功能

在用户主界面的好友列表条目上，每个条目均有鼠标监听，鼠标左键双击后会弹出聊天密钥输入框，输入正确即可进入聊天界面：

```
1. pane.setOnMouseClicked(e -> {
2.     if (e.getButton() == MouseButton.PRIMARY && e.getClickCount()
3.         == 2) {
4.         try {
5.             System.out.println("friend:"+friend);
6.             User friendUser = new User();
7.             friendUser.setId(Integer.parseInt(friend.get("id")));
8.             friendUser.setAccount(friend.get("account"));
9.             friendUser.setAvatarUrl(friend.get("avatarUrl"));
10.            if (!chatApplications.containsKey(friendUser.getId()))
11.            {
12.                String chatKey=MsgDialog.getInputFromDialog();
13.                System.out.println("聊天密钥="+chatKey);
14.                if (chatKey!=null){
15.                    String trueKey=friend.get("chatKey");
16.                    if (trueKey.equals(chatKey)) new ChatApplicat
17.                    ion(this.user, friendUser,this).start(new Stage());
18.                    else MsgDialog.showDialog(Alert.AlertType.ERR
19.                    OR,"聊天密钥错误","请再次尝试",false);
20.                }
21.            }
22.        } catch (Exception ex) {
23.            ex.printStackTrace();
24.        }
25.    }
26.});
```

一旦进入聊天框，客户端会自动请求消息列表，获取历史消息：

```
1. /**
2.  * 请求消息列表
3.  */
4. private void requestMessageList() {
5.     try {
6.         if (!chatClient.isOpen()) {
7.             chatClient.connectBlocking();
8.         }
9.     } catch (InterruptedException e) {
10.        e.printStackTrace();
11.        System.out.println("连接失败");
12.    }
```



```

12.         MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败", "
            无法连接到服务器", false);
13.     }
14.     if (chatClient.isOpen()) {
15.         String request = "type=" + CONST.REQUEST + "&category=" +
            CONST.INIT_CHAT + "&me=" + me.getId() + "&friend=" + friend.getId
            ();
16.         chatClient.send(request);
17.         MsgDialog.showDialog(Alert.AlertType.INFORMATION, "请求消
            息列表", "正在请求消息列表, 请稍候", true);
18.     } else {
19.         System.out.println("连接失败");
20.         MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败", "
            无法连接到服务器", false);
21.     }
22. }

```

服务端接收消息之后调用对应的策略, 并将结果返回给客户端, 客户端根据结果获取聊天记录, 并加载 UI, 这里只展示客户端对应的代码:

```

1.  if (category == CONST.INIT_CHAT) {
2.      List<Map<String, String>> list = Parser.getList(message);
3.      list.forEach(this::addChatMessage);
4.  }
    将每一条聊天记录添加到界面中:
1.  /**
2.   * 添加聊天记录
3.   *
4.   * @param chat 聊天记录
5.   */
6.  private void addChatMessage(Map<String, String> chat) {
7.      System.out.println("添加聊天记录:" + chat);
8.      Integer senderId = Integer.parseInt(chat.get("senderId"));
9.      String content = chat.get("content");
10.     String sendTime = chat.get("sendTime");
11.     if (senderId.equals(me.getId())) {
12.         sendMsg.setText("");
13.         FXMLLoader loader = new FXMLLoader(getClass().getClassLoa
            der().getResource("chatcomponent-me.fxml"));
14.         try {
15.             Parent parent = loader.load();
16.             ((ChatMeController) loader.getController()).setAvatar
                (new Image(me.getAvatarUrl()));
17.             ((ChatMeController) loader.getController()).setSendTi
                me(sendTime);

```

```

18.         ((ChatMeController) loader.getController()).setSendCo
           ntent(content);
19.         alignRight.set(true);
20.         Platform.runLater(() -> {
21.             msgList.getItems().add(parent);
22.             // 获取列表中的最后一个项目的索引
23.             int lastIndex = msgList.getItems().size() - 1;
24.             // 将列表滚动到最后一个项目
25.             msgList.scrollTo(lastIndex);
26.         });
27.     } catch (IOException ex) {
28.         ex.printStackTrace();
29.     }
30. } else {
31.     FXMLLoader loader = new FXMLLoader(getClass().getClassLoa
           der().getResource("chatcomponent-friend.fxml"));
32.     try {
33.         Parent parent = loader.load();
34.         ((ChatFriendController) loader.getController()).setAv
           atar(new Image(friend.getAvatarUrl()));
35.         ((ChatFriendController) loader.getController()).setSe
           ndTime(sendTime);
36.         ((ChatFriendController) loader.getController()).setSe
           ndContent(content);
37.         alignRight.set(false);
38.         Platform.runLater(() -> {
39.             msgList.getItems().add(parent);
40.             // 获取列表中的最后一个项目的索引
41.             int lastIndex = msgList.getItems().size() - 1;
42.             // 将列表滚动到最后一个项目
43.             msgList.scrollTo(lastIndex);
44.         });
45.     } catch (IOException ex) {
46.         ex.printStackTrace();
47.     }
48. }
49. }

```

用户在输入框输入消息后，点击发送按钮，程序会读取输入框的文本，并将消息发送给服务端：

```

1.  /**
2.   * 发送聊天消息
3.   */
4.   private void sendChatMessage() {
5.       String content = sendMsg.getText();

```

```

6.         if (content != null && !content.trim().isEmpty()) {
7.             try {
8.                 if (!chatClient.isOpen()) {
9.                     chatClient.connectBlocking();
10.                }
11.            } catch (InterruptedException ex) {
12.                ex.printStackTrace();
13.                MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败", "请稍后重试", false);
14.                System.out.println("连接失败");
15.            }
16.            if (chatClient.isOpen()) {
17.                System.out.println("连接成功");
18.                String message = "type=" + CONST.REQUEST + "&category=" + CONST.SEND_MSG +
19.                    "&senderId=" + me.getId() + "&receiverId=" + friend.getId() + "&type=" + "文本" + "&content=" + content;
20.                chatClient.send(message);
21.                MsgDialog.showDialog(Alert.AlertType.INFORMATION, "发送消息", "正在发送消息, 请稍候", true);
22.            } else {
23.                System.out.println("连接失败");
24.                MsgDialog.showDialog(Alert.AlertType.ERROR, "连接失败", "无法连接到服务器", false);
25.            }
26.        } else {
27.            System.out.println("发送消息为空!");
28.            MsgDialog.showDialog(Alert.AlertType.ERROR, "消息不能为空!", "请填写消息内容", false);
29.        }
30.    }

```

服务端接收到后, 调用发送消息策略, 并将消息通过接收方的 socket 发送给接收方

```

1. public class SendMsgStrategy extends Strategy {
2.     public SendMsgStrategy(Map<String, Object> params) {
3.         super(params);
4.     }
5.     @Override
6.     public Result execute() {
7.         System.out.println("执行了 SendMsgStrategy");
8.         boolean b = ChatDAO.insertChat(Integer.parseInt(params.get("senderId").toString()), Integer.parseInt(params.get("receiverId").toString()), "文本", (String) params.get("content"), new Date());

```

```

9.         if(b) {
10.             String type= (String) params.get("type");
11.             String content= (String) params.get("content");
12.             Chat chat=new Chat();
13.             chat.setSenderId(Integer.parseInt(params.get("senderI
d").toString()));
14.             chat.setReceiverId(Integer.parseInt(params.get("recei
verId").toString()));
15.             chat.setType(type);
16.             chat.setContent(content);
17.             chat.setSendTime(new java.sql.Date(new Date().getTime
()));
18.             WebSocket socket= Server.onlineChats.get(params.get("
receiverId")+"-"+params.get("senderId"));
19.             System.out.println("此时
onLineChats:"+Server.onlineChats);
20.             System.out.println("此时对方 socket:"+socket);
21.             if(socket!= null&&!socket.isClosed()) {
22.                 socket.send("type="+ CONST.RESPONSE+"&category="+
CONST.SEND_MSG+"&code="+1+"&"+chat.toString());
23.                 System.out.println("发消息给对面成功");
24.             }
25.             return Result.succeed(chat);
26.         } else {
27.             return Result.failed();
28.         }
29.     }
30. }

```

接收方接收到消息后显示在自己的聊天框中，这部分的代码调用的也是上面的 addChatMessage 方法，不再展示。

## 5.6 文件传输

用户在聊天框界面点击文件上传按钮，即可弹出文件传输界面：

```

1. uploadBtn= (Button) namespace.get("uploadBtn");
2. uploadBtn.setOnMouseClicked(e->{
3.     Platform.runLater(()->{
4.         try {
5.             if (!fileUploadIsOpen){
6.                 new FileUploadApplication(me,friend,this).start(n
ew Stage());
7.             }
8.         } catch (Exception ex) {

```

```

9.         ex.printStackTrace();
10.        MsgDialog.showDialog(Alert.AlertType.ERROR, "进入文件
    上传界面失败", "", false);
11.    }
12.    });
13. });

```

与文本聊天功能类似，一进入界面，客户端会自动请求历史记录，这里不再详细展示。值得一提的是，在数据库中，文件的存储方式是使用链接文本的方式存储（与用户头像存储方式类似）

对‘上传文件按钮’添加监听器，打开一个文件选择器，上传文件，上传到服务器之后向服务端发送请求，将传输记录插入到数据库中：

```

1. uploadBtn.setOnMouseClicked(e -> {
2.     FileChooser fileChooser = new FileChooser();
3.     fileChooser.setTitle("请选择要上传的文件");
4.     File selectedFile = fileChooser.showOpenDialog(stage);
5.     String filename = null;
6.     boolean flag = true;
7.     if (selectedFile != null) {
8.         try {
9.             FileUploader.sshSftpUpload(selectedFile, selectedFile.
    getName());
10.            MsgDialog.showDialog(Alert.AlertType.INFORMATION, "文
    件上传中", "请稍后", true);
11.        } catch (Exception ex) {
12.            ex.printStackTrace();
13.            MsgDialog.showDialog(Alert.AlertType.ERROR, "文件上传
    失败", "请检查网络连接", false);
14.            flag = false;
15.        }
16.        if (flag) {
17.            try {
18.                if (!fileUploadClient.isOpen()) {
19.                    fileUploadClient.connectBlocking();
20.                }
21.            } catch (InterruptedException ex) {
22.                ex.printStackTrace();
23.                MsgDialog.showDialog(Alert.AlertType.ERROR, "连接
    失败", "请稍后重试", false);
24.                System.out.println("连接失败");
25.            }
26.            if (fileUploadClient.isOpen()) {
27.                System.out.println("连接成功");

```

```

28.         String message = "type=" + CONST.REQUEST + "&category=" + CONST.UPLOAD_FILE +
29.             "&senderId=" + me.getId() + "&receiverId="
30.             " + friend.getId() + "&type=" + "文件"
31.             " + "&content=" + selectedFile.getName();
32.         fileUploadClient.send(message);
33.         MsgDialog.showDialog(Alert.AlertType.INFORMATION,
34.             "发送文件", "正在发送文件, 请稍候", true);
35.     } else {
36.         System.out.println("连接失败");
37.         MsgDialog.showDialog(Alert.AlertType.ERROR, "连接
38.             失败", "无法连接到服务器", false);
39.     }
40. }
41. });

```

服务端接收到消息后, 调用文件上传策略, 这里的操作与文本聊天类似:

```

1. public class UploadFileStrategy extends Strategy {
2.     public UploadFileStrategy(Map<String, Object> params){
3.         super(params);
4.     }
5.     @Override
6.     public Result execute() {
7.         System.out.println("执行了 UploadFileStrategy");
8.         boolean b = ChatDAO.insertChat(Integer.parseInt(params.get("senderId").toString()), Integer.parseInt(params.get("receiverId").toString()), "文件",
9.             (String) params.get("content"), new Date());
10.        if(b) {
11.            String type= (String) params.get("type");
12.            String content=(String)params.get("content");
13.            Chat chat=new Chat();
14.            chat.setSenderId(Integer.parseInt(params.get("senderId").toString()));
15.            chat.setReceiverId(Integer.parseInt(params.get("receiverId").toString()));
16.            chat.setType(type);
17.            chat.setContent(CONST.AVATAR_URL_PREFIX+content);
18.            chat.setSendTime(new java.sql.Date(new Date().getTime()));
19.            WebSocket socket= Server.onlineChatsOfUploadFile.get(params.get("receiverId")+"-"+params.get("senderId"));
20.            System.out.println("此时
                onlineChatsOfUploadFile:"+Server.onlineChatsOfUploadFile);

```

```

21.         System.out.println("此时对方 socket:"+socket);
22.         if(socket!= null&&!socket.isClosed()) {
23.             String message="type="+ CONST.RESPONSE+"&category
                ="+CONST.UPLOAD_FILE+"&code="+1+"&"&chat;
24.             System.out.println("发送给对面:"+message);
25.             socket.send(message);
26.             System.out.println("发文件给对面成功");
27.         }
28.         return Result.succeed(chat);
29.     } else {
30.         return Result.failed();
31.     }
32. }
33. }

```

每一个文件条目都添加了鼠标监听，鼠标右键点击后即可弹出路径选择框进行文件的下载：

```

1. parent.setOnMouseClicked(e -> {
2.     if (e.getButton() == MouseButton.SECONDARY) {
3.         // 下载文件
4.         System.out.printf("下载文件" + fileUrl);
5.         DirectoryChooser directoryChooser = new DirectoryChooser()
6.         ;
7.         directoryChooser.setTitle("下载文件");
8.         File selectedDir = directoryChooser.showDialog(stage);
9.         if (selectedDir != null) {
10.            try {
11.                URL url = new URL(fileUrl);
12.                // 从URL 中获取文件名，例如 "COPYRIGHT"
13.                String fileName = new File(url.getPath()).getName
14.                ();
15.                // 创建保存文件的完整路径
16.                Path savePath = Paths.get(selectedDir.getAbsolutePath(), fileName);
17.                try (InputStream in = new BufferedInputStream(url.
18.                    openStream());
19.                    FileOutputStream fileOutputStream = new File
20.                    OutputStream(savePath.toFile())) {
21.                    byte[] dataBuffer = new byte[1024];
22.                    int bytesRead;
23.                    while ((bytesRead = in.read(dataBuffer, 0, 10
24.                        24)) != -1) {

```

```

22.                                fileOutputStream.write(dataBuffer, 0, byt
    esRead);
23.                                }
24.                            }
25.                    } catch (Exception ex) {
26.                        ex.printStackTrace();
27.                        // 处理异常
28.                        MsgBox.showDialog(Alert.AlertType.ERROR, "下载
    异常",
29.                                "下载失败", false);
30.                    }
31.            }
32.        }
33.    });
    
```

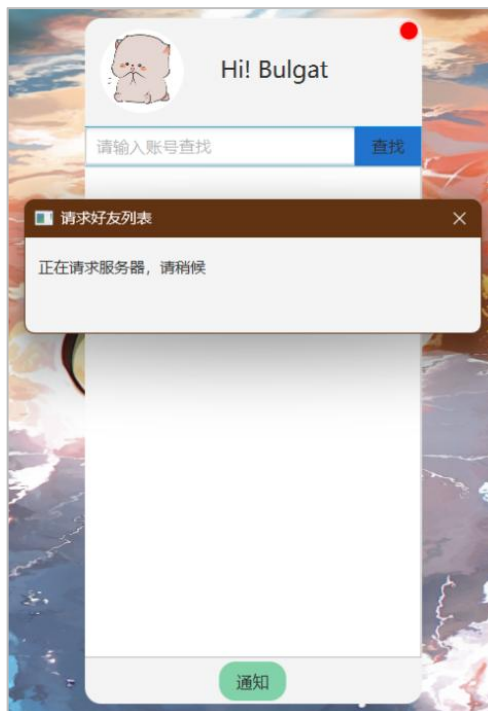
## 6 功能演示

### • 新用户注册

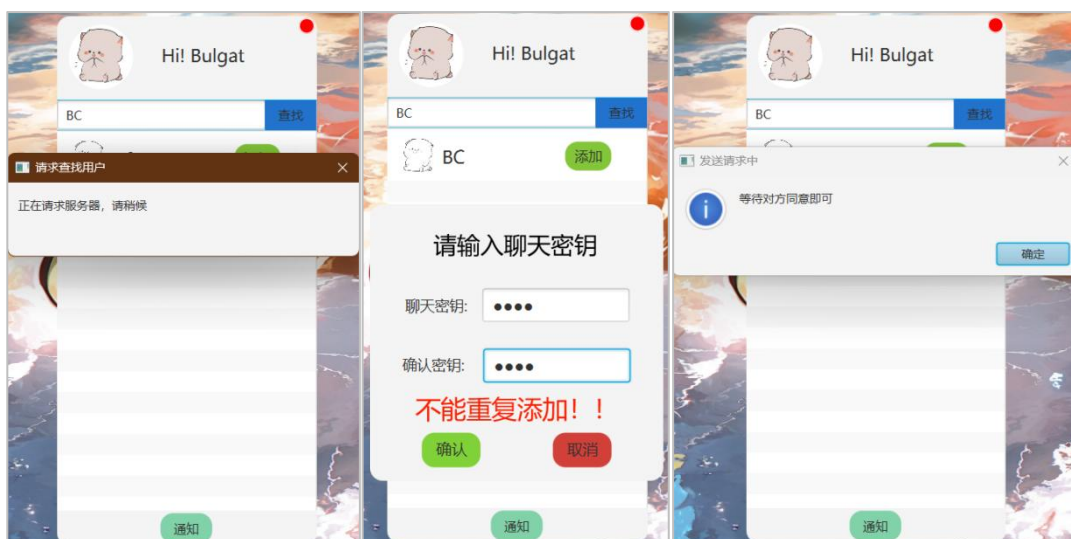




## • 用户登录



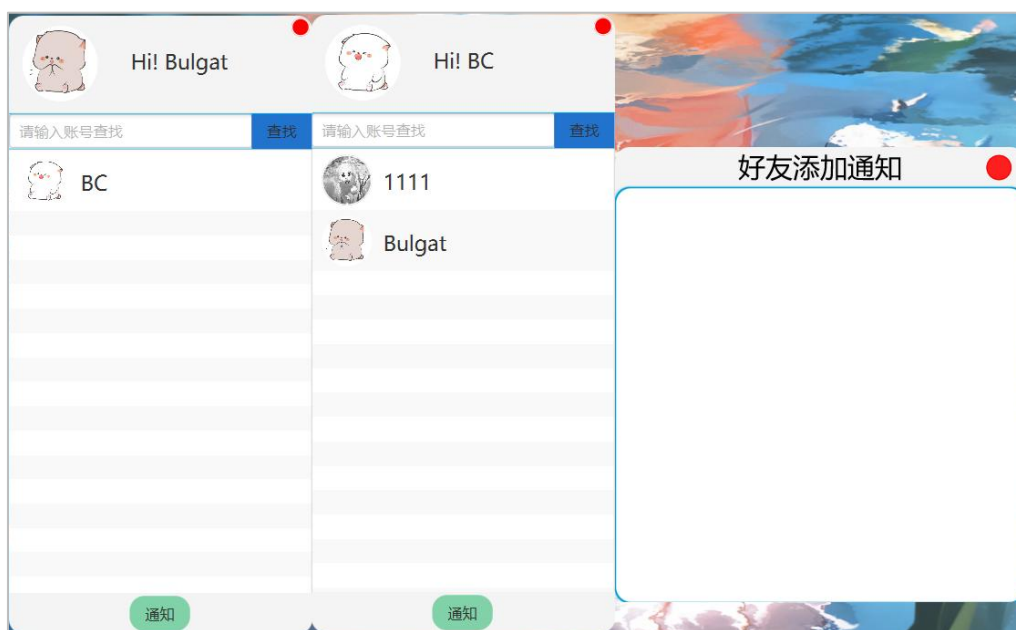
## • 添加好友



此时对方界面：

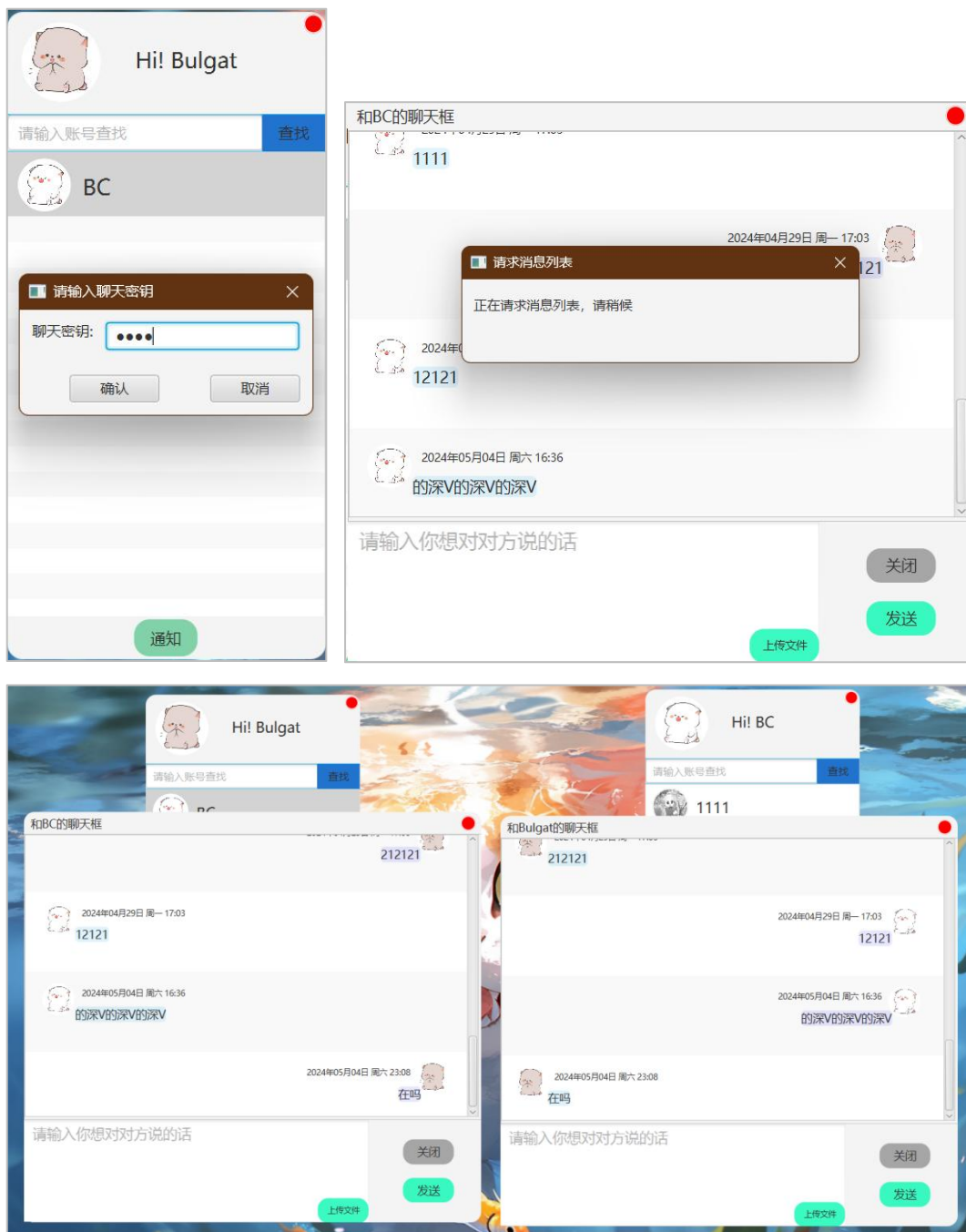


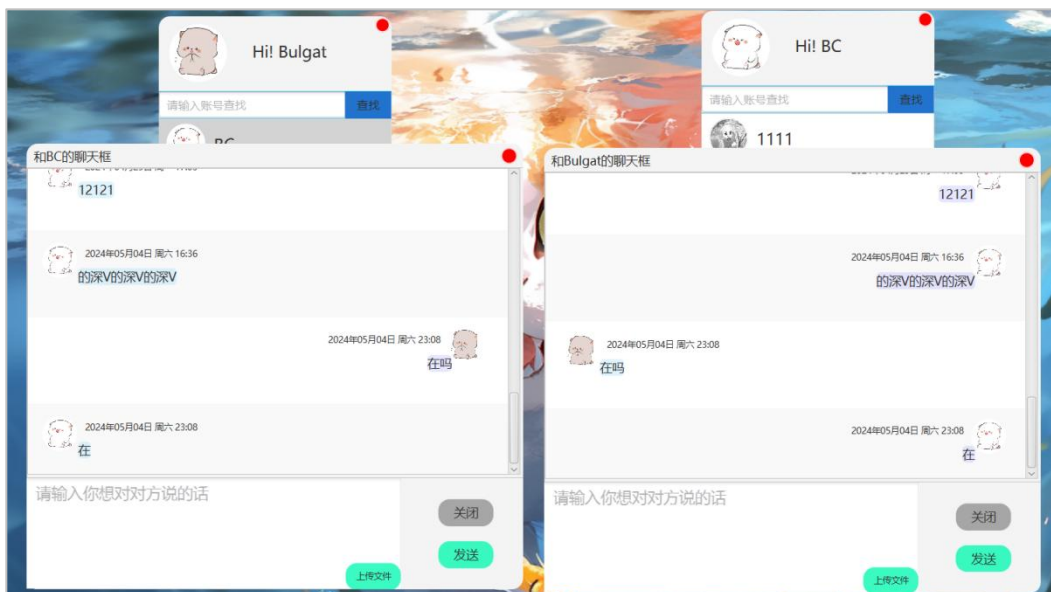
点击同意：



• 文本聊天

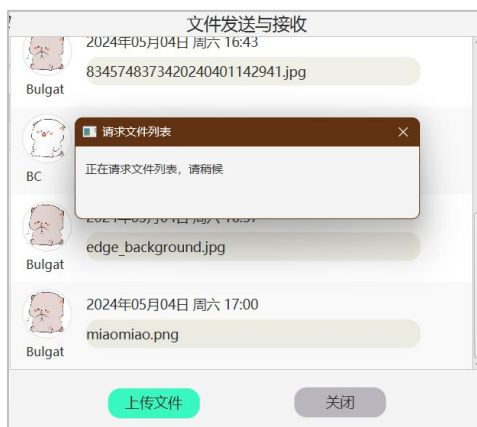
点击聊天框进行聊天：



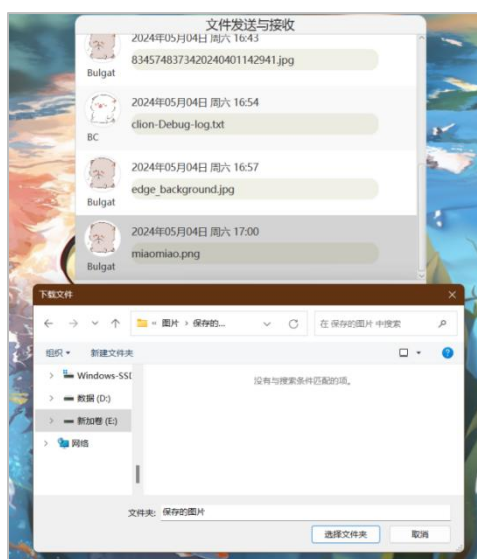


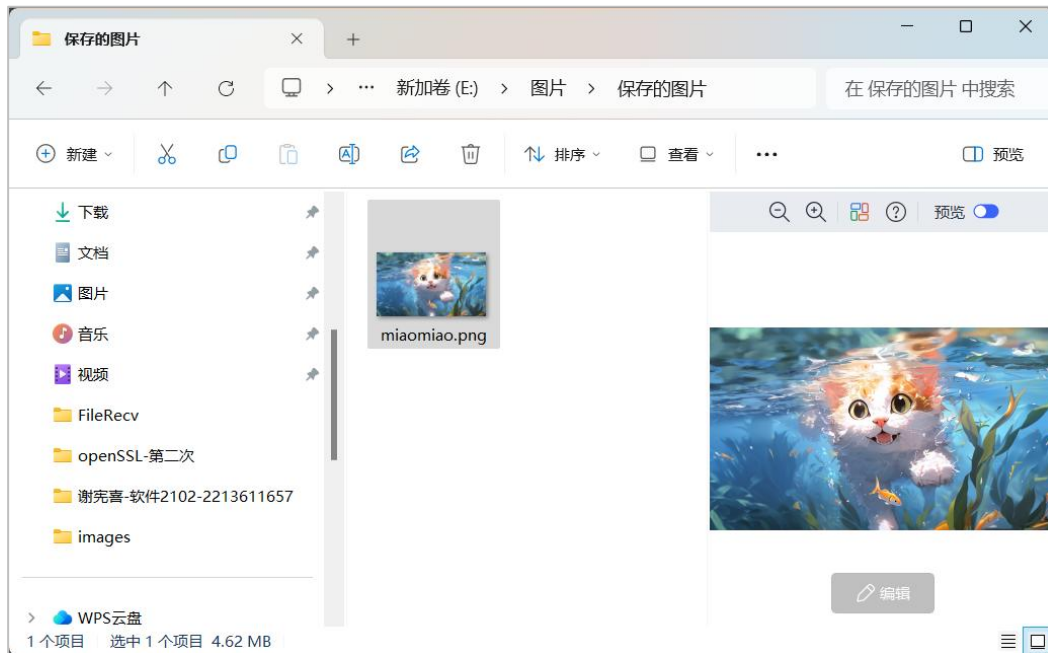
### • 文件上传与下载

点击上传文件按钮：

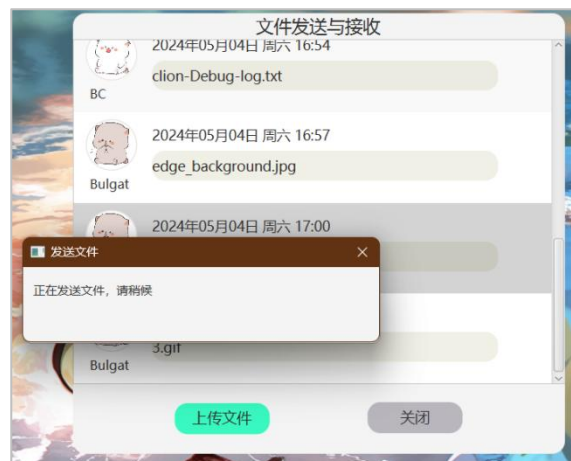


下载文件：





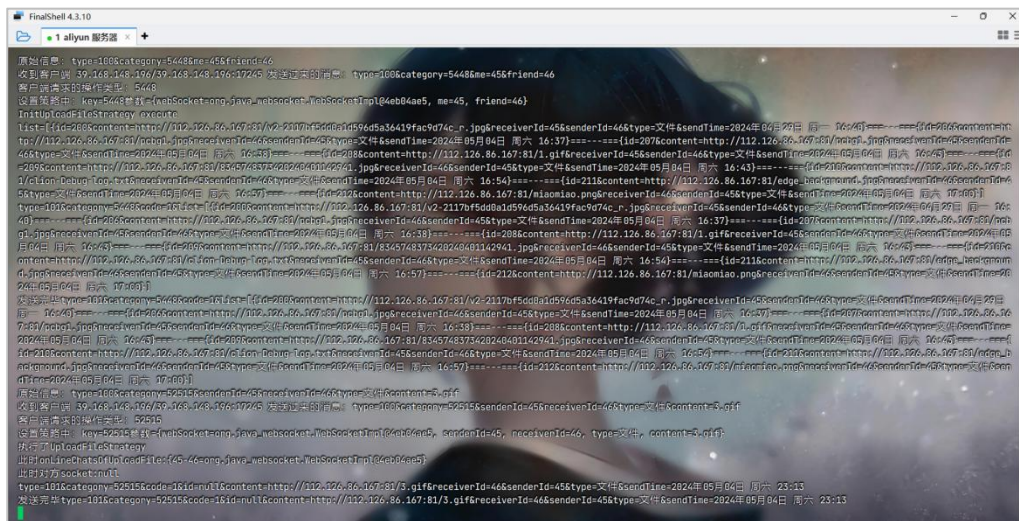
上传文件：



上传成功！



查看服务器运行情况：



## 7 遇到的问题

1. 当编写好一个示例的服务端项目文件，打包放置在阿里云服务器上运行时，由于在 Windows 本地使用的是 maven 进行依赖的管理，在云服务器上的 maven 配置不当，导致程序一直找不到 Java-WebSocket 依赖项，后面重新配置一遍 maven 和 java17 环境后才成功运行。

2. 在项目设计初期，只是一股脑的在服务端接收客户端的消息后处理进行回复，后面发现消息类型多种多样，有登录请求、注册请求、添加（或删除）好友请求、对话请求等，这样会导致很长的 switch-case 语句出现，导致 server 端代码过长，不易于维护与扩展，后面受 web 项目中常用的 springboot 框架中的 controller 层的启发，web 前端向后端发送的消息是通过路径来进行定位到不同的 controller；而对于 C/S 架构，简化来看，server 端只是对 client 端发送的消息作出处理，并将处理后的结果数据返回给 client 端，而不同的消息类型对于与 server 端不同的处理操作，这类似于操作系统中的系统调用，用户程序调用系统调用陷入操作系统中，事先会将调用的系统调用号和调用参数存入寄存器中，内核程序读取这些寄存器然后执行，这种策略模式极大减小了 C/S 架构的代码耦合度，于是我便在我的项目中加入该设计模式，这也有利于我后期的开发与维护。

3. 在项目初期，我不知道消息报文的格式该如何设计比较好，后面我参考了浏览器中地址栏参数是采用 ‘&’ 符号进行分隔，于是我也这样设计，难点是

如何解析这样格式的报文，特别是如何解析出 list（比如当请求好友列表时，必须使用数组或者 list 实现），后面我参考以前做过的 web 项目，设计一个从服务端向客户端返回的返回体，解决了这个问题。同样，在客户端端也可以设计一个请求体向服务端发送请求，这样有利于后期的维护，但是碍于时间原因，并且目前项目也并没有因为没有设计请求体而出现问题，这个想法并未落实。

4. 一开始我使用的是 Socket 类，并且在开发的初期，并未关闭 socket 的 io 流，在项目基本功能完成的时候，程序非常卡顿，后面逐步关闭 io 流发现程序无端报错，出现 `IOException`，后续一步一步调试后将报错解决，依旧发现程序运行卡顿，客户端和服务端之间的通信特别慢，怀疑是每通信一次都需要有 IO 流的打开与关闭，非常耗费计算机资源，于是改用 `Java-WebSocket` 进行项目的重构，好在服务端一开始被设计的耦合度比较小，重构起来没有遇到设计模式上面的困难。

5. `JavaFX` 在设计静态（不会随着程序的运行而改变样式）的 UI 上是非常方便的，但是在设计会随着程序运行而改变样式的 UI 时（比如聊天对话框自适应聊天文本），需要使用 `Controller` 类在代码里面手动改变 UI 样式，这花费了我一个晚上查阅相关资料和调试代码。

6. Server 端需要维护 4 个 map 来存储客户端程序用户的状态，以此来更新用户好友是否在线、用户聊天框消息发送和文件发送、用户通知。

7. 由于本项目也作为网络安全课程大作业，项目中也实现通信加密功能，在该功能实现的初期，我最初不是使用的 `SSL/TLS` 协议，而是手动在代码中加入消息的加密和解密，这样要在代码中修改很多地方，不利于维护。后面使用 `SSL/TLS` 协议之后，代码改动小，但是在最初编写加密通信的测试小程序时，由于我对 `SSL/TLS` 协议的不熟悉，导致程序一直无法实现握手成功，后续花费一天的时间才解决。

8. 当测试多个用户同时使用客户端时，出现了线程之间的竞争，导致我服务端中出现资源竞争，特别是对 4 个 map 的资源竞争，比如可能在一个用户退出登录时，服务端应当删除 map 中有关该用户的记录，但是这个时候还有别的用户退出系统，map 中的 `remove` 不是原子操作，这导致了删除操作的错误。后面我使用了 `Collections.synchronizedMap()` 方法获取到了增删改查操作是原子操作的 map；此外，我还加入了 `synchronized` 语句，实现了 map 的互斥访问，保障

了服务端多线程程序的正确执行。

9. 项目中有提到更新好友在线状态的功能，这不是主要功能，但是我在此上花费的时间并不少。这个功能是我后期才添加上去的，起初在设计项目的时候并未考虑到这个功能，设计不周全，导致我加上这个功能时花费了一定的精力。

10. 项目并未考虑到大量用户的情况，对此情况，我打算加入队列来优化程序，但是学业繁忙，时间有限，并未落实。

## 8 结论与展望

### 项目亮点：

1. 本项目完成了一个聊天软件的基本功能，能够实现登录、注册、聊天、好友管理这些核心的功能，效果良好，并且考虑到了加密通信，适合做一个对安全性要求很高的通信软件。
2. 程序中有大量的异常处理语句，同样也会使用弹窗的方式反馈给用户，这提高了程序的鲁棒性。
3. 软件的架构设计耦合度低，代码可读性高，利于后续维护和功能的扩展。

### 项目不足：

1. 软件 UI 设计不够完美，动画过渡地不够流畅，字体和 UI 样式可能存在一些不和谐，而且 UI 主要是模仿腾讯 QQ，没有创新性。并且，虽然相比于 JavaSwing，JavaFX 的 UI 设计更加方便，但是 JavaFX 的性能不如 JavaSwing。
2. 软件并未考虑到大量请求的情况，并且由于设备有限，不太容易模拟出大量请求的测试。

### 展望：

#### 1. 改进 UI 设计

采用现代化的 UI 设计理念，使用流畅的动画效果，和谐的色彩搭配以及直观的用户交互设计。

#### 2. 考虑大量请求的情况

优化服务器端的架构，消息队列来缓冲请求，使用负载均衡器分发用户请求到不同的服务器实例，以及利用缓存策略减轻数据库的压力。

#### 3. 设计请求体



定义统一的请求和响应数据模型，如使用 JSON 格式来标准化数据结构，这样可以降低数据处理的复杂性，并便于扩展新的功能。

#### 4. 增加发送表情包的功能

在客户端 UI 中集成表情包选择器，允许用户快速挑选和发送表情。服务器端需要处理和传输表情数据，确保表情包在不同客户端间正确显示。

#### 5. 增加群聊功能

设计群聊的数据模型，实现群管理功能（如创建群、加入群、退出群等），以及优化消息广播机制以支持大量用户同时在线的场景。

#### 6. 增加消息提示功能

实现桌面通知功能，当应用程序在后台运行时，新消息到来可以通过操作系统的通知中心提示用户。

#### 7. 开发适配安卓系统的客户端

开发适用于 Android 设备的客户端应用，确保与现有的服务端兼容，并提供与桌面版相似的功能和用户体验。

#### 8. 提高性能

进行代码审核和性能测试，优化算法和数据结构，减少资源消耗，以及考虑使用更高效的 C++QT 框架对软件进行重写。