简易 QQ 聊天系统

信管创新23-1班

张恒睿，吴春晓，周元华，段云枫

项目简介

本项目是一个基于 Java 的简易实时聊天系统，类似于 QQ。通过 WebSocket 实现数据交互与传输，支持单用户间的通讯和群组通信。

功能特性

- **单用户聊天**：用户可以与其他单个用户进行实时聊天。
- **群组聊天**：用户可以创建群组并在群组中进行聊天。
- **消息通知**：当有新消息时，系统会进行通知。
- **用户管理**：支持用户注册、登录和管理。

技术栈

- **后端**：Java, WebSocket, MySQL
- **前端**：Java Swing (GUI)
- **依赖管理**：Maven

项目结构

```
my_qq_client/
├── src/
│   ├── main/
│   │   ├── java/cn/amatrix/
│   │   │   ├── controller/ # 控制器层，处理用户请求
│   │   │   ├── model/ # 数据模型层，定义实体类
│   │   │   ├── service/ # 服务层，包含业务逻辑
│   │   │   ├── DAO/ # 数据访问层，向数据库或后端服务器请求资源
│   │   │   ├── util/ # 工具类
│   │   │   └── Main.java # 主程序入口
│   │   └── resources/ # 图片等资源文件
│   └── test/ # 测试代码
├── target/ # 导出的 jar 文件
├── doc/ # 额外的说明文档
├── pom.xml # Maven 配置文件
└── README.md # 项目说明文件
```

安装与运行

克隆项目：

```
git clone <仓库地址>
cd <项目目录>
```

配置数据库：

- 创建 MySQL 数据库并导入相关表结构。
- 使用以下 SQL 脚本创建数据库表：

```
    CREATE TABLE users (
        user_id INT AUTO_INCREMENT PRIMARY KEY,
        username VARCHAR(50) NOT NULL UNIQUE,
        password VARCHAR(255) NOT NULL,
        email VARCHAR(100) NOT NULL UNIQUE,
        avatar TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        log_status ENUM('online', 'offline') DEFAULT 'offline'
        last_login_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        last_logout_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
```

```sql
CREATE TABLE friends (
    user_id INT NOT NULL,
    friend_id INT NOT NULL,
    added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, friend_id),
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (friend_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

```sql
CREATE TABLE friend_requests (
    request_id INT AUTO_INCREMENT PRIMARY KEY,
    sender_id INT NOT NULL,
    receiver_id INT NOT NULL,
    request_message TEXT,
    request_status ENUM('pending', 'approved', 'rejected') DEFAULT 'pending',
    requested_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sender_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (receiver_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

```sql
CREATE TABLE private_messages (
    message_id INT AUTO_INCREMENT PRIMARY KEY,
    sender_id INT NOT NULL,
    receiver_id INT NOT NULL,
    message TEXT NOT NULL,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sender_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (receiver_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

```sql
CREATE TABLE user_groups (
    group_id INT AUTO_INCREMENT PRIMARY KEY,
    group_name VARCHAR(100) NOT NULL,
    avatar TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```sql
CREATE TABLE group_members (
    group_id INT NOT NULL,
    user_id INT NOT NULL,
    power ENUM('owner', 'admin', 'member') DEFAULT 'member',
    joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (group_id, user_id),
    FOREIGN KEY (group_id) REFERENCES user_groups(group_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

```sql
CREATE TABLE group_messages (
    message_id INT AUTO_INCREMENT PRIMARY KEY,
    group_id INT NOT NULL,
    sender_id INT NOT NULL,
    message TEXT NOT NULL,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (group_id) REFERENCES user_groups(group_id) ON DELETE CASCADE,
    FOREIGN KEY (sender_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

```sql
CREATE TABLE group_join_requests (
    request_id INT AUTO_INCREMENT PRIMARY KEY,
    group_id INT NOT NULL,
    user_id INT NOT NULL,
    request_message TEXT,
    request_status ENUM('pending', 'approved', 'rejected') DEFAULT 'pending',
    requested_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (group_id) REFERENCES user_groups(group_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

编译与运行：

- 修改 `src/main/resources/config/application.properties` 文件中的配置信息，更改服务器地址。

```
cd < project mainDir >
mvn clean package
java -jar /target/my_qq_client-1.0-SNAPSHOT-jar-with-dependencies.jar
```

使用说明

1. **注册与登录**：
    - 启动程序后，用户可以通过 GUI 进行注册和登录。
2. **单用户聊天**：
    - 登录后，选择联系人进行聊天。
3. **群组聊天**：
    - 创建群组并邀请成员后，可以在群组中进行聊天。

贡献

欢迎提交 Issue 和 Pull Request 来贡献代码。

许可证

本项目采用 MIT 许可证，详情请参阅 LICENSE

前期功能设计

面板及其功能设计

登录流程

1. 启动应用后进入登录流程。
2. 在登录界面，用户可以：
    - 输入用户名密码进行登录。
        - 验证通过则登录成功，进入控制面板。
        - 验证失败则返回登录界面。
    - 点击注册进入注册界面。
    - 点击找回密码进入找回密码界面。

注册流程

1. 在注册界面，用户输入注册信息。
2. 发送验证邮件进行邮箱验证。
    - 验证通过则注册成功，返回登录界面。
    - 验证失败则返回注册界面。

修改密码流程

1. 在找回密码界面，用户输入找回信息。
2. 发送验证邮件进行邮箱验证。
    - 验证通过则找回成功，返回登录界面。
    - 验证失败则返回找回密码界面。

主要业务流程

1. 登录成功后进入主要业务流程，进入主界面。
2. 在主界面，用户可以选择：
    - 好友相关功能，进入好友界面。
    - 群组相关功能，进入群组界面。
    - 查看消息，进入查看聊天界面。

聊天界面

1. 在查看消息界面，用户可以：
    - 查看消息。
    - 发送消息。

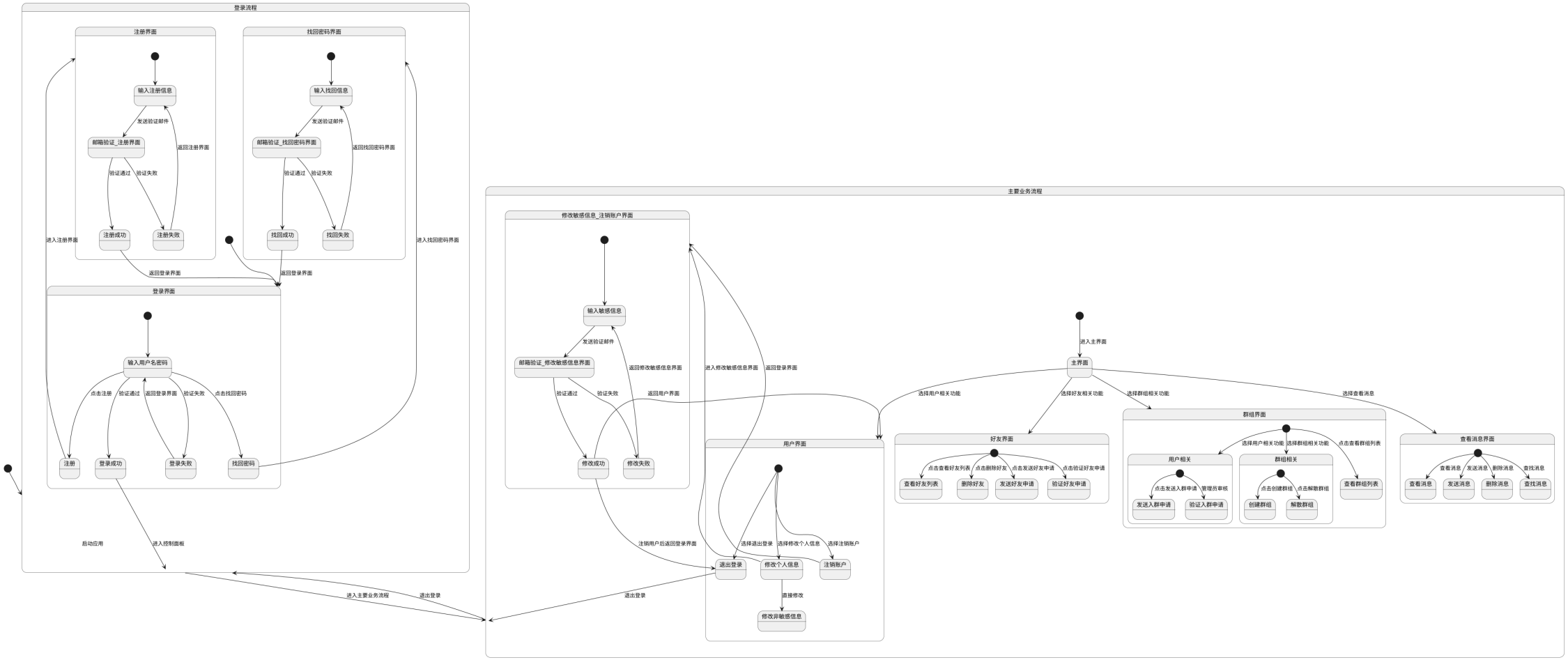
用户相关界面

1. 在用户相关界面，用户可以：
    - 发送好友申请
    - 处理他人申请

群组相关界面

1. 在群组相关界面，用户可以：
    - 加入群组
    - 管理员及群主处理他人申请
    - 创建群组
    - 解散群组

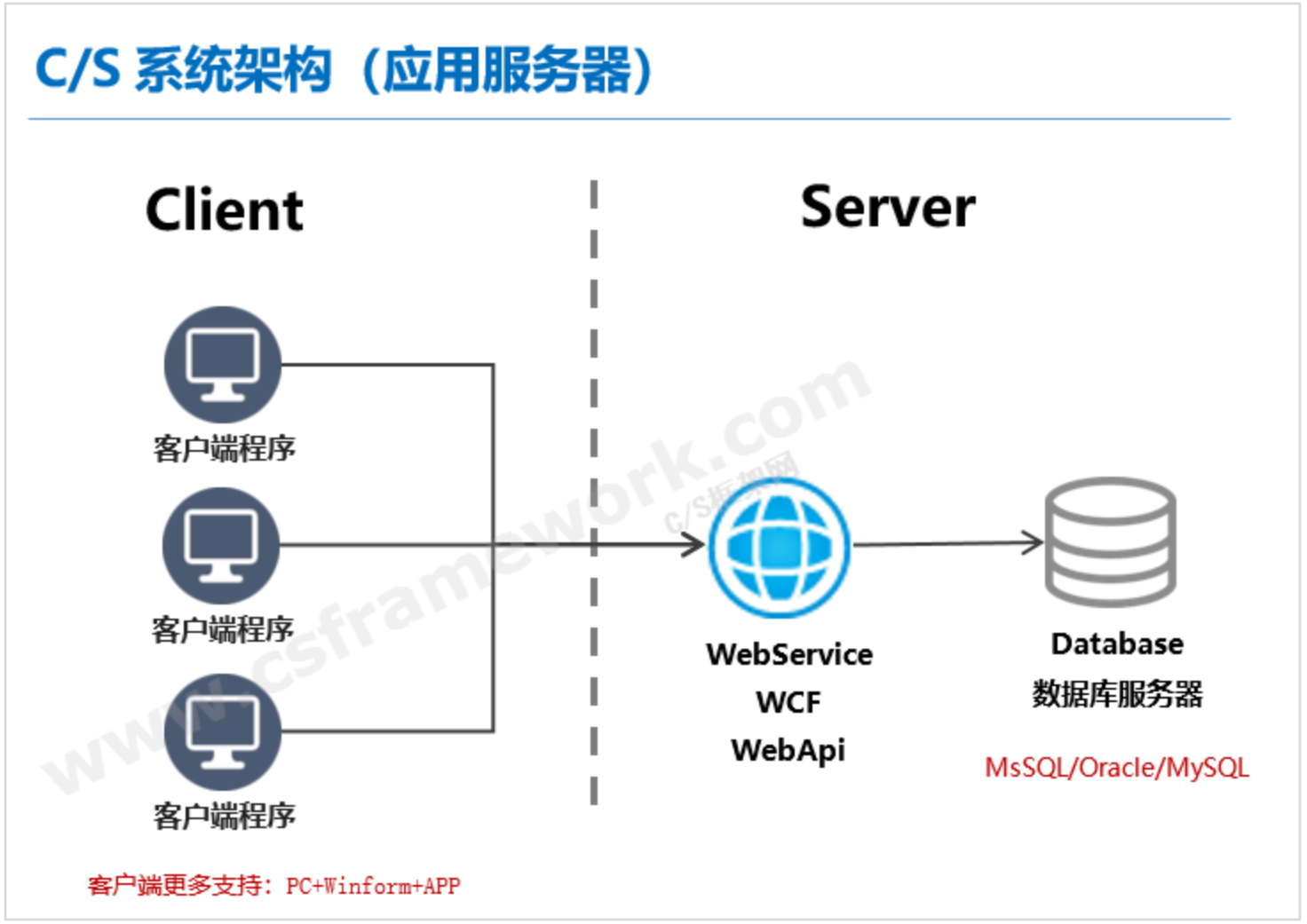
面板功能转化自动机图

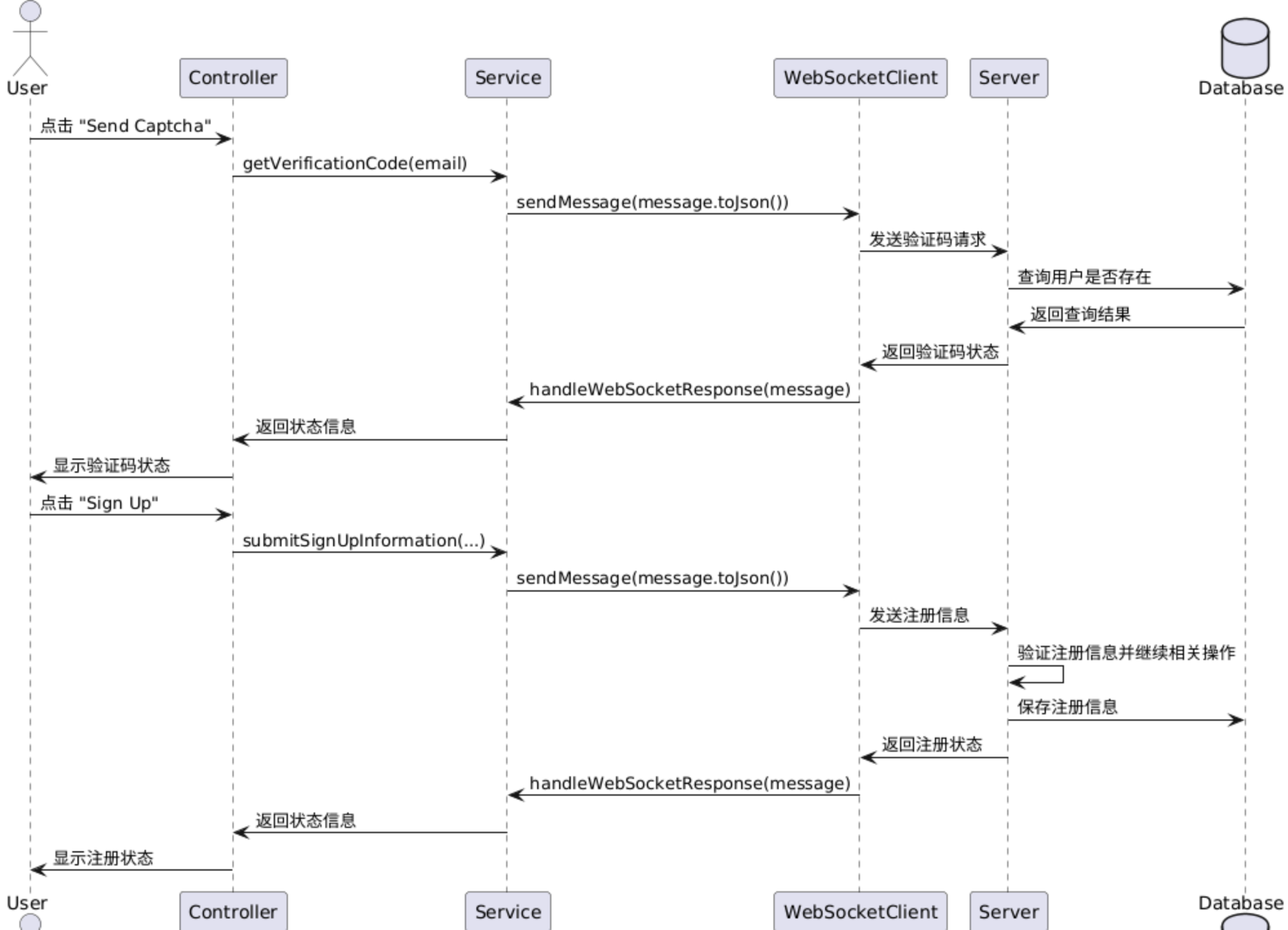

登录流程
注册界面
输入注册信息
发送验证邮件
邮箱验证_注册界面
验证通过
验证失败
返回注册界面
注册成功
注册失败
找回密码界面
输入找回信息
发送验证邮件
邮箱验证_找回密码界面
验证通过
验证失败
返回找回密码界面
找回成功
找回失败
进入注册界面
进入找回密码界面
返回登录界面
返回登录界面
登录界面
输入用户名密码
点击注册
验证通过
返回登录界面
验证失败
点击找回密码
注册
登录成功
登录失败
找回密码
启动应用
进入控制面板
进入主要业务流程
退出登录
主要业务流程
修改敏感信息_注销账户界面
输入敏感信息
发送验证邮件
邮箱验证_修改敏感信息界面
返回修改敏感信息界面
验证通过
验证失败
返回用户界面
修改成功
修改失败
进入修改敏感信息界面
返回登录界面
注销用户后返回登录界面
用户界面
选择退出登录
选择修改个人信息
选择注销账户
退出登录
修改个人信息
注销账户
直接修改
修改非敏感信息
退出登录
进入主界面
主界面
选择用户相关功能
选择好友相关功能
选择群组相关功能
选择查看消息
好友界面
点击查看好友列表
点击删除好友
点击发送好友申请
点击验证好友申请
查看好友列表
删除好友
发送好友申请
验证好友申请
群组界面
选择用户相关功能
选择群组相关功能
点击查看群组列表
用户相关
点击发送入群申请
管理员审核
发送入群申请
验证入群申请
群组相关
点击创建群组
点击解散群组
创建群组
解散群组
查看群组列表
查看消息界面
查看消息
发送消息
删除消息
查找消息
查看消息
发送消息
删除消息
查找消息

前期功能设计

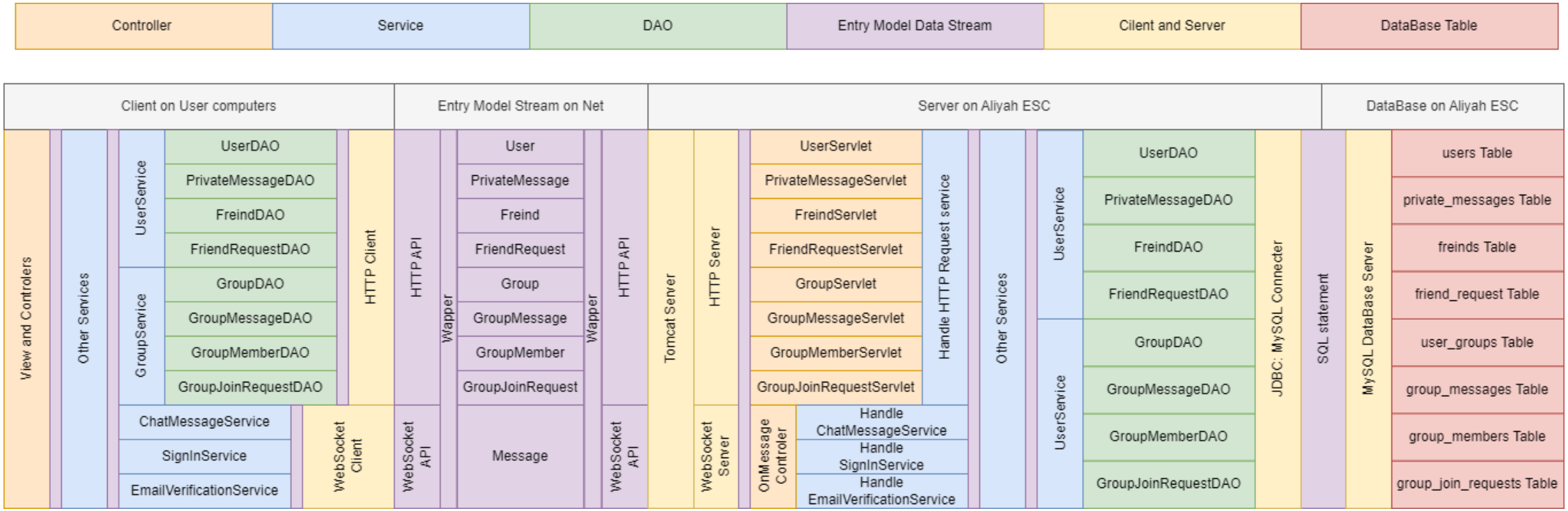
前后端交互设计

实现多个用户之间的通信： CS 模式

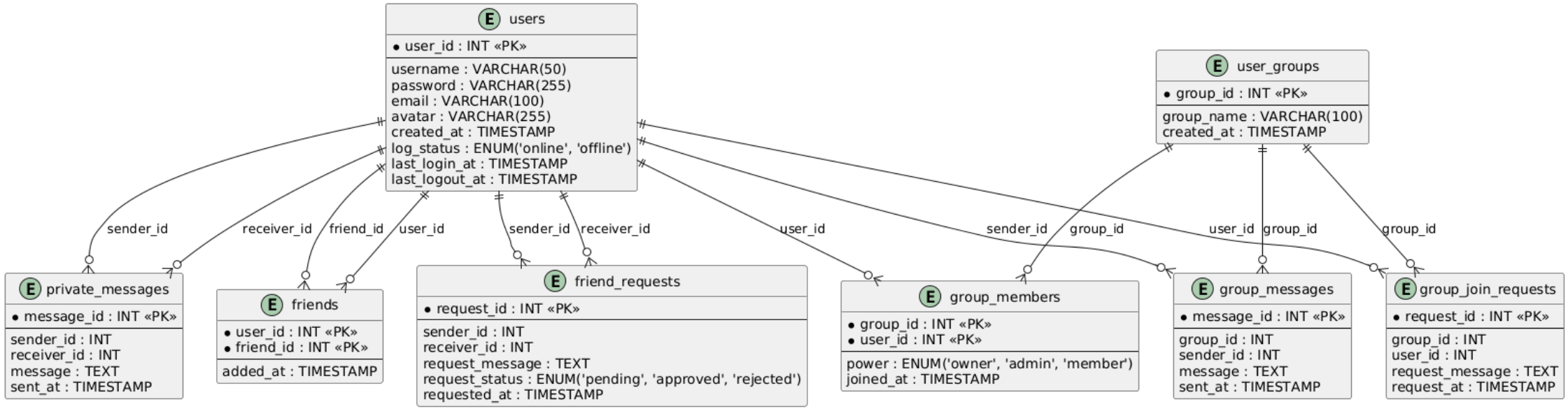



User
Controller
Service
WebSocketClient
Server
Database
点击 "Send Captcha"
getVerificationCode(email)
sendMessage(message.toJson())
发送验证码请求
查询用户是否存在
返回查询结果
返回验证码状态
handleWebSocketResponse(message)
返回状态信息
显示验证码状态
点击 "Sign Up"
submitSignUpInformation(...)
sendMessage(message.toJson())
发送注册信息
验证注册信息并继续相关操作
保存注册信息
返回注册状态
handleWebSocketResponse(message)
返回状态信息
显示注册状态
User
Controller
Service
WebSocketClient
Server
Database

前后端数据交互的代码基础



数据库表及其相互关系


users
• user_id : INT «PK»
username : VARCHAR(50)
password : VARCHAR(255)
email : VARCHAR(100)
avatar : VARCHAR(255)
created_at : TIMESTAMP
log_status : ENUM('online', 'offline')
last_login_at : TIMESTAMP
last_logout_at : TIMESTAMP
user_groups
• group_id : INT «PK»
group_name : VARCHAR(100)
created_at : TIMESTAMP
sender_id
receiver_id
friend_id
user_id
sender_id
receiver_id
user_id
sender_id
group_id
user_id
group_id
group_id
private_messages
• message_id : INT «PK»
sender_id : INT
receiver_id : INT
message : TEXT
sent_at : TIMESTAMP
friends
• user_id : INT «PK»
• friend_id : INT «PK»
added_at : TIMESTAMP
friend_requests
• request_id : INT «PK»
sender_id : INT
receiver_id : INT
request_message : TEXT
request_status : ENUM('pending', 'approved', 'rejected')
requested_at : TIMESTAMP
group_members
• group_id : INT «PK»
• user_id : INT «PK»
power : ENUM('owner', 'admin', 'member')
joined_at : TIMESTAMP
group_messages
• message_id : INT «PK»
group_id : INT
sender_id : INT
message : TEXT
sent_at : TIMESTAMP
group_join_requests
• request_id : INT «PK»
group_id : INT
user_id : INT
request_message : TEXT
request_at : TIMESTAMP

前后端数据交互的基础：Entry 实体类

```
public class User {
    private int user_id;
    private String username;
    private String password;
    private String email;
    private String avatar;
    private Timestamp created_at;
    private String log_status;
    private Timestamp last_login_at;
    private Timestamp last_logout_at;

    /* setters and getters */
    public static User fromJson(String json);
    public String toJson();
}
```

通过 JSON 实现 Java 对象与字节流的转换


```
{
    "avatar":"Base64String",
    "created_at":"2024-12-21 15:37:43.219",
    "email":"email@123.com",
    "last_login_at":"2024-12-21 15:37:43.219",
    "last_logout_at":"2024-12-21 15:37:43.219",
    "log_status":"online",
    "password":"password",
    "user_id":1,
    "username":"admin"
}
```

实时性需求不高的通信接口：HTTP API

```
Post http://<Base URL: 部署的服务器 ip 地址>/demo_webapps/<Sub_URLs>
RequestBody: {
    "type" : "getAll",
    "param" : "String 类型的对应参数，有具体 http api 规定"
}
```

实时性需求高的通信接口：WebSocket API

WebSocket 简介

WebSocket 是一种在单个 TCP 连接上进行全双工通信的协议。WebSocket 使得客户端和服务器之间可以进行实时的双向通信，适用于需要频繁数据交换的应用场景，如在线聊天、实时通知、游戏等。

WebSocket 的特点

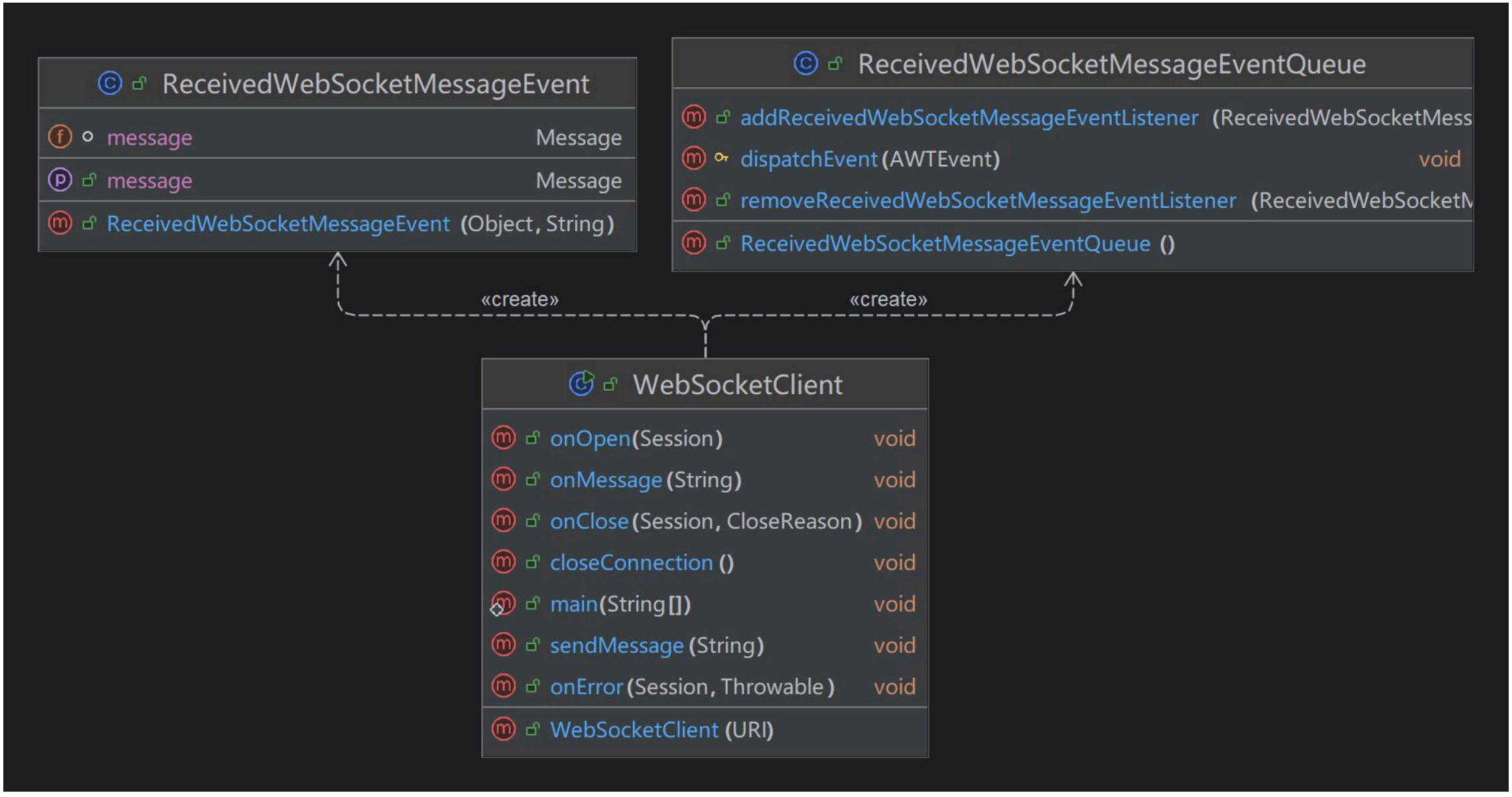
- **全双工通信**：
  WebSocket 允许客户端和服务器之间同时发送和接收数据，而不需要像 HTTP 那样每次请求都要建立新的连接。
- **低延迟**：
  WebSocket 连接建立后，数据可以在客户端和服务器之间低延迟地传输，适用于实时应用。
- **持久连接**：
  一旦 WebSocket 连接建立，除非被显式关闭，否则连接将一直保持打开状态，减少了频繁建立和关闭连接的开销。

WebSocket API 的设计

```
{
  "receiver": {"name": "John Doe", "id": "user123", "type": "user"},
  "sender": {"name": "Group Chat", "id": "group456", "type": "group"},
  "messageId": "msg0",
  "type":"getVerificationCode",
  "content":"3432900546@qq.com",
  "timestamp": "2024-11-16T23:51:37.571112700+08:00[GMT+08:00]",
  "status": "sent"
}
```

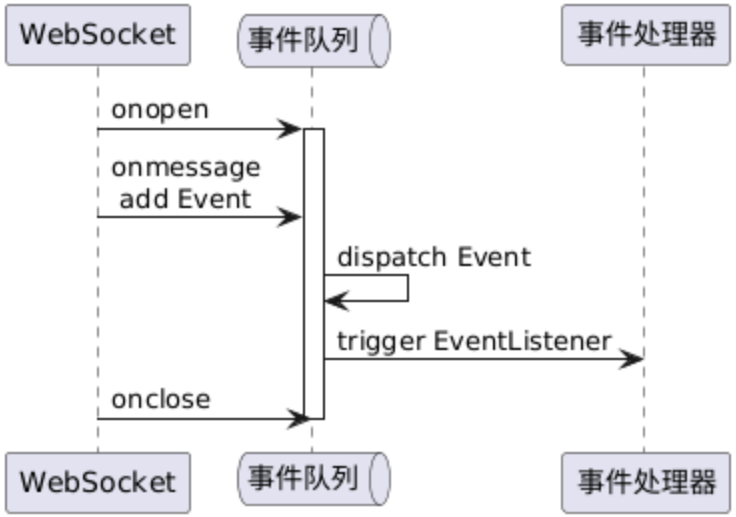
客户端响应 WebSocket Server 返回的响应信息

WebSocketClient 包简介



WebSocket的事件分发与捕获

- `ReceivedWebSocketMessageEvent.java`：将接收到的消息封装为一个事件
- `ReceivedWebSocketMessageEventQueue.java`：继承自 `EventQueue` 实现事件分发和监听器管理。
- `WebSocketClient.java`：WebSocket 激活后将系统队列替换为自定义的事件队列 `ReceivedWebSocketMessageEventQueue`。onMessage 接收到服务器回传的消息并向上述事件队列中发布对应消息事件。



WebSocket的事件响应

- `ReceivedWebSocketMessageEventListener.java`：定义处理接收到的 WebSocket 消息事件的方法的接口，继承自 `ActionListener`。
- `ReceivedWebSocketMessageEventQueue.java`：用于处理接收到的 WebSocket 消息事件的事件队列类，继承自 `EventQueue`，并实现事件分发和监听器管理。