

系统架构设计文档

一、系统概述

本系统是一个去中心化的分布式局域网匿名聊天Web应用，用户可以在局域网内进行匿名聊天，并享受联机小游戏和智能聊天机器人等额外功能。系统采用Spring Boot作为后端框架，Vue.js作为前端框架。Hazelcast作为分布式内存数据库，用于节点之间同步数据。后端H2数据库作为本地持久化存储。前端通过Axios与后端之间进行通信。

二、系统架构

1. 技术栈

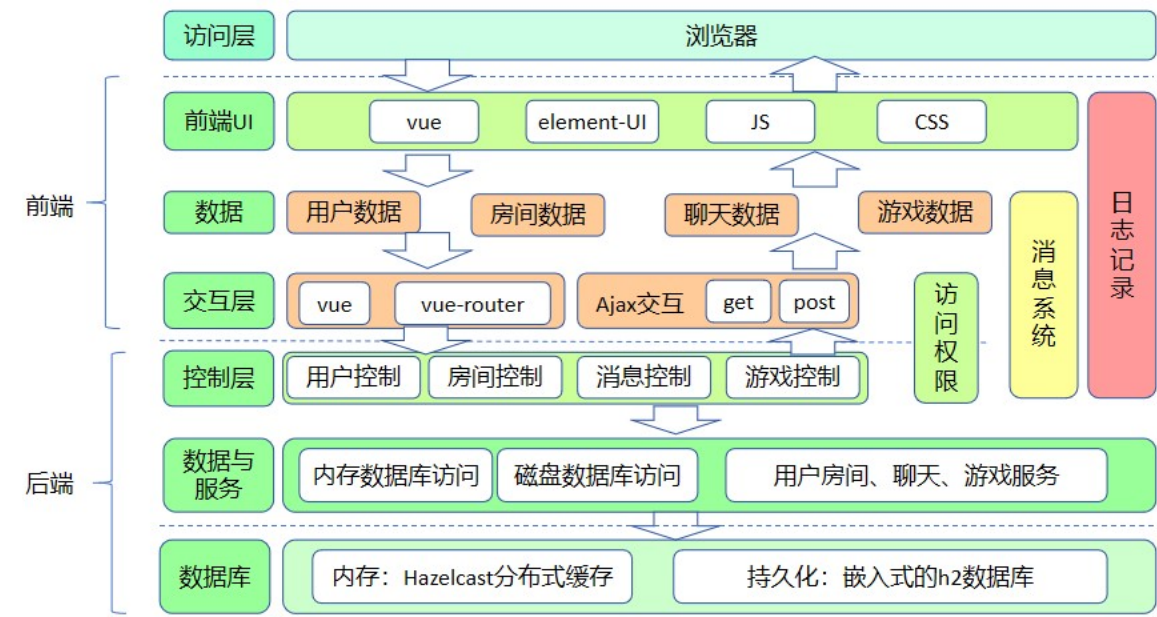
后端技术栈

- **Spring Boot**：用于构建后端服务，提供RESTful API。
- **Hazelcast**：用于分布式缓存和数据共享，实现节点之间同步数据。
- **H2 Database**：轻量级嵌入式数据库，用于持久化存储。
- **Spring Data JPA**：简化数据库操作。
- **Spring Security**：提供用户认证和授权。

前端技术栈

- **Vue.js**：用于构建用户界面。
- **Vue Router**：实现前端路由管理。
- **Axios**：用于前后端通信。
- **Element UI**：UI组件库，用于快速构建用户界面。

2. 架构图



三、模块/子系统

1. 前端模块

前端模块主要负责用户界面的呈现和用户交互，使用Vue.js构建。以下是前端模块的详细描述：

(1) 状态栏模块

功能描述：

- 显示用户头像：展示当前登录用户的头像。
- 登录按钮：提供用户登录的入口。
- 显示当前房间名：显示用户当前所在聊天室的名称。
- 创建房间按钮：允许用户创建新的聊天室。

技术实现：

- 使用Vue.js和Element UI实现界面组件。
- 登录状态的管理使用Vuex进行状态管理。
- Axios与后端进行交互，实现登录、获取头像和房间信息。

(2) 房间信息栏模块

功能描述：

- 显示所有房间：展示当前系统中的所有聊天室列表。
- 房间信息：每个房间显示头像、房间名和最新消息。

技术实现：

- 使用Vue.js和Element UI实现列表和信息展示组件。
- Axios请求后端接口获取房间信息，并实时更新。

(3) 输入窗口模块

功能描述：

- 输入文本：用户可以输入并发送文本消息。
- 表情包：用户可以选择并发送表情包。
- 上传文件：用户可以上传文件并发送到聊天室。
- 新建或加入游戏按钮：提供用户创建或加入游戏的入口。

技术实现：

- 使用Vue.js和Element UI实现文本框、表情选择器、文件上传组件和按钮。
- Axios与后端交互，实现消息发送和文件上传。

(4) 消息窗口模块

功能描述：

- 显示当前房间聊天记录：展示当前聊天室的所有消息。
- 支持多种消息类型：包括文本、图像、文件等。

技术实现：

- 使用Vue.js实现消息显示组件。
- 长轮询或WebSocket与后端保持连接，实现消息的实时更新。

(5) 数据定义模块

功能描述：

- 定义前端存储的数据：包括登录状态、房间信息、聊天记录等。

技术实现：

- 使用Vuex进行状态管理，管理登录状态、房间列表、当前房间聊天记录等数据。

(6) 网络通信模块

功能描述：

- 负责与后端通信：包括用户登录、获取房间信息、发送消息、接收消息等。

技术实现：

- 使用Axios与后端RESTful API进行通信。
- 处理网络请求的统一管理和错误处理。

2. 后端模块

后端模块使用Spring Boot框架，主要负责数据处理、业务逻辑和与前端的通信。以下是后端模块的详细描述：

(1) 数据库定义

功能描述：

- 定义房间、游戏房间、聊天记录、用户等数据模型。

技术实现：

- 使用Spring Data JPA定义实体类。
- H2数据库作为本地持久化存储，Hazelcast作为分布式缓存。

(2) model模块

功能描述：

- 定义各种数据类型：包括用户、房间、聊天记录、游戏等。

技术实现：

- 使用Java类定义数据模型，使用注解映射到数据库表。

(3) Utils模块

功能描述：

- 提供一些工具类：如内存工具、身份验证、加密等。

技术实现：

- Java工具类实现，提供通用的工具方法。

(4) Service模块

功能描述：

- 实现业务逻辑：包括用户管理、房间管理、消息处理等。

技术实现：

- 使用Spring Boot的Service层，封装业务逻辑。
- Service类通过注入Repository类访问数据库。

(5) config模块

功能描述：

- Spring Boot的配置模块：包括Hazelcast配置、Web配置等。

技术实现：

- 使用Spring Boot的@Configuration注解配置Hazelcast、Web、Security等。

(6) 聊天机器人模块

功能描述：

- 负责聊天机器人的连接、消息收发等。

技术实现：

- 与第三方聊天机器人服务集成，通过API进行消息的发送和接收。

(7) 游戏模块

功能描述：

- 实现各种游戏机制：如五子棋、围棋、翻转棋等。

技术实现：

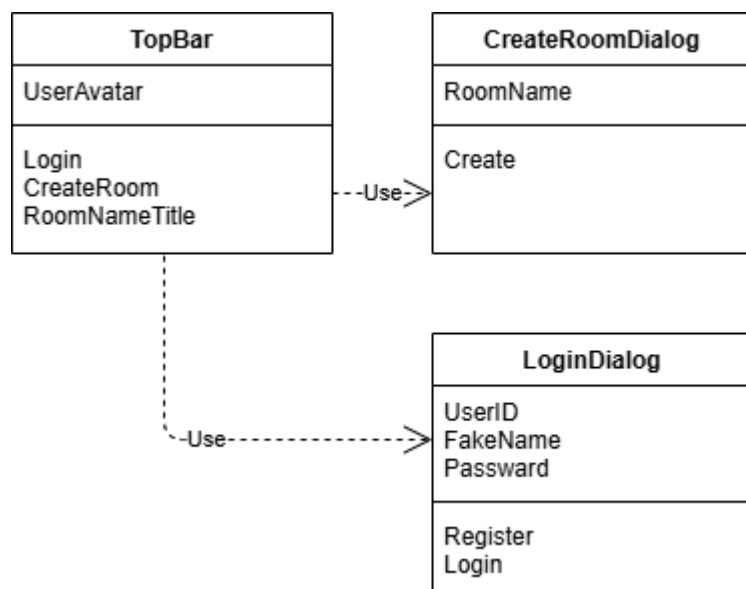
- 定义游戏逻辑和规则，使用Java实现游戏引擎。
- 与前端通过WebSocket或REST API进行实时交互。

四、核心模块设计

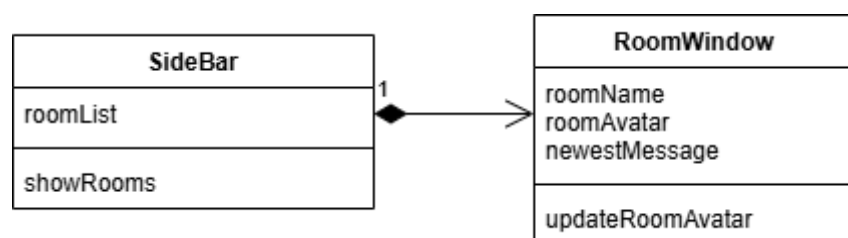
类图、序列图

1.前端核心模块设计

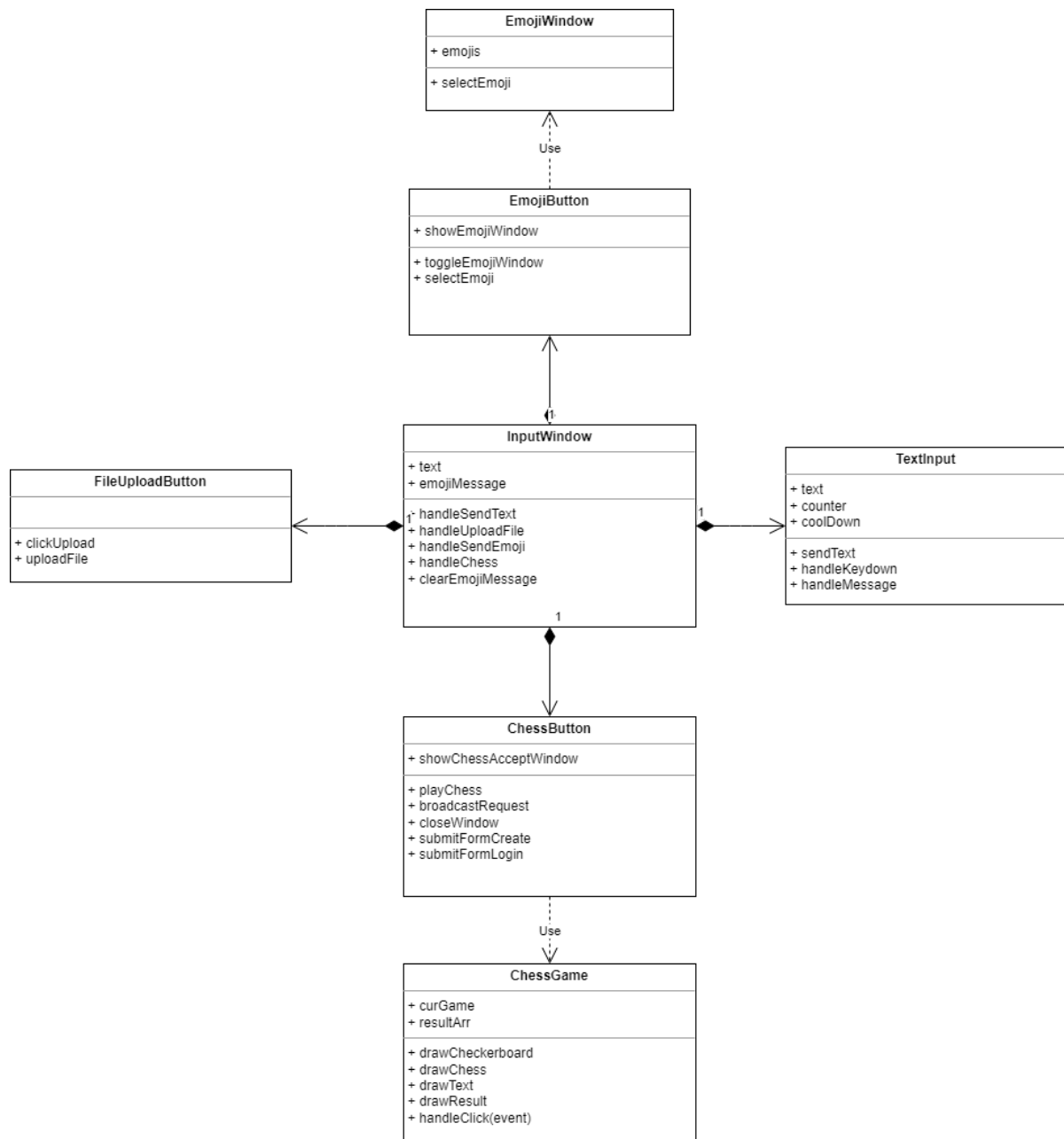
(1) 状态栏模块



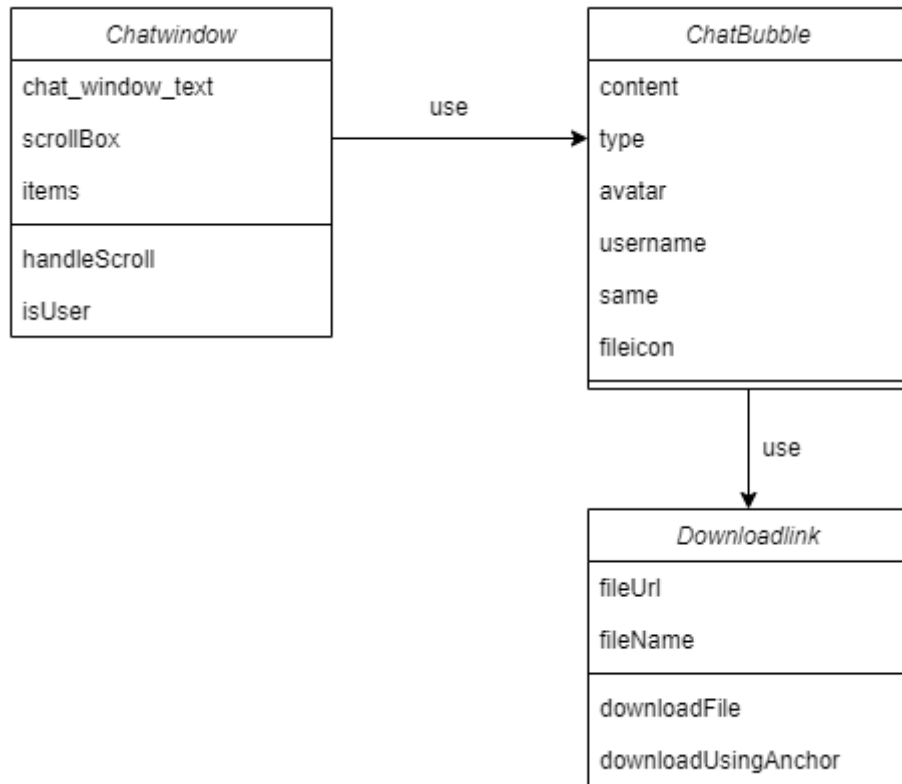
(2) 房间信息栏模块



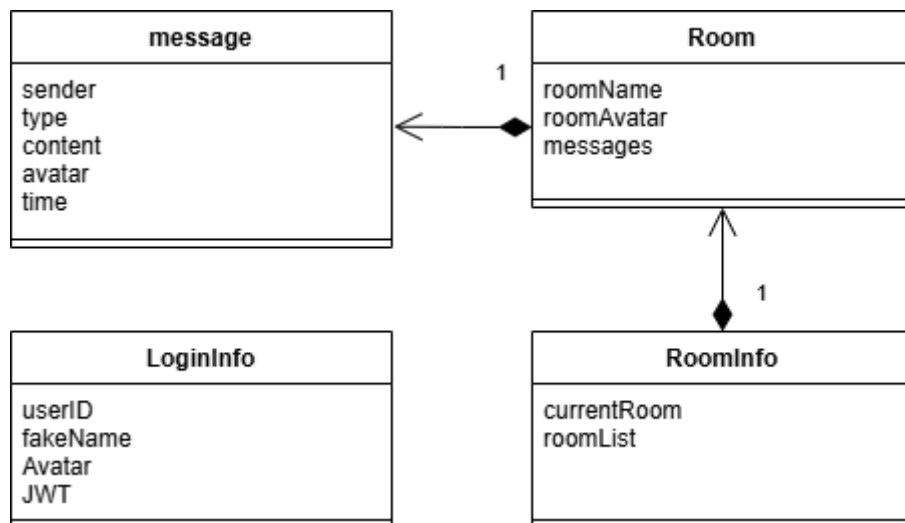
(3) 输入窗口模块



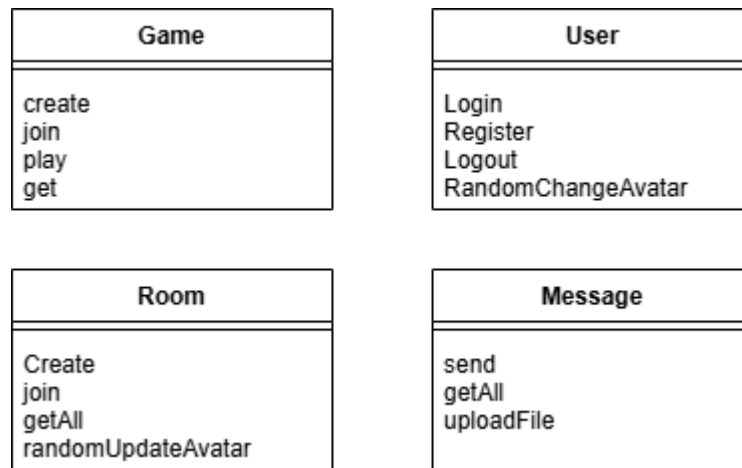
(4) 消息窗口模块



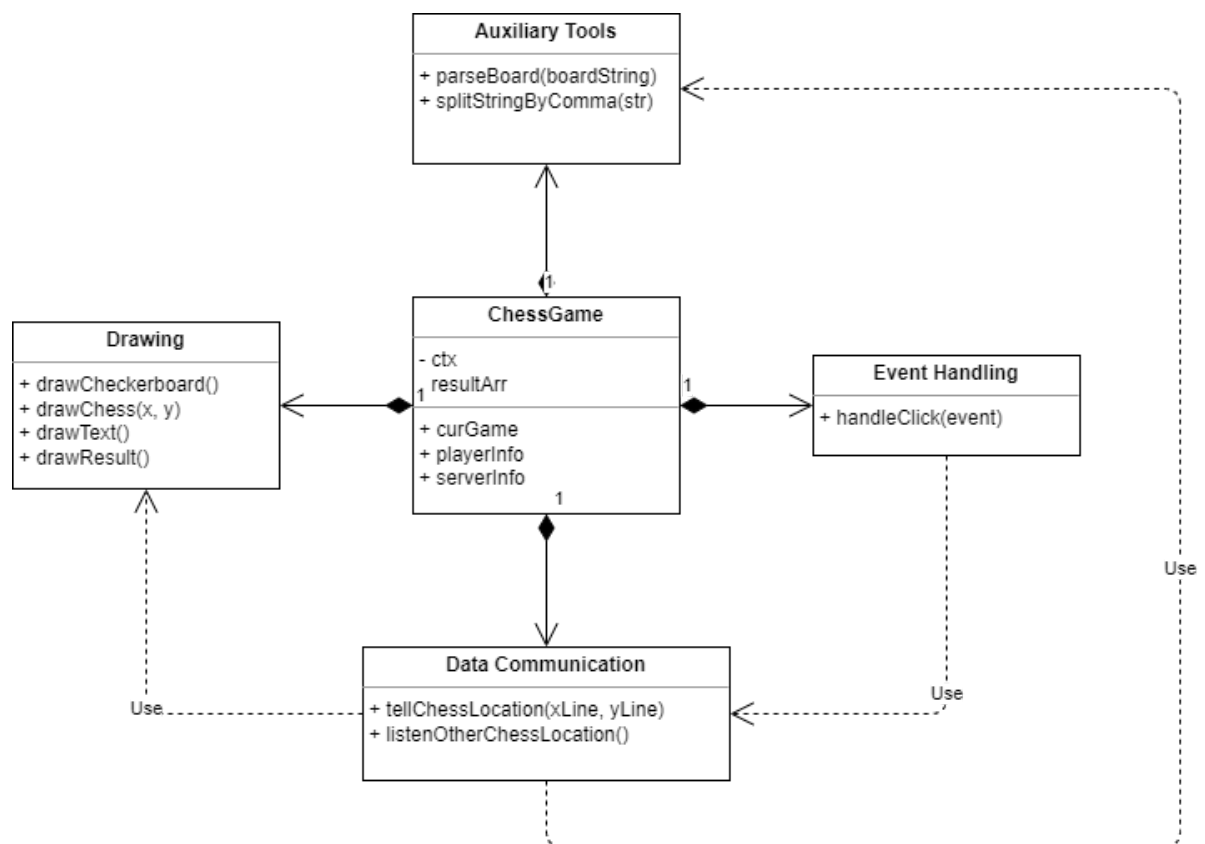
(5) 数据定义模块



(6) 网络通信模块



(7) 游戏模块



2.后端核心模块设计

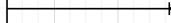
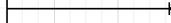
(1) 数据库定义

T_USER	
PK	<u>id VARCHAR(255)</u>
	FAKENMAE VARCHAR(255)
	PASSWD VARCHAR(255)
	cookie VARCHAR(255)
	USERPIC VARCHAR(255)

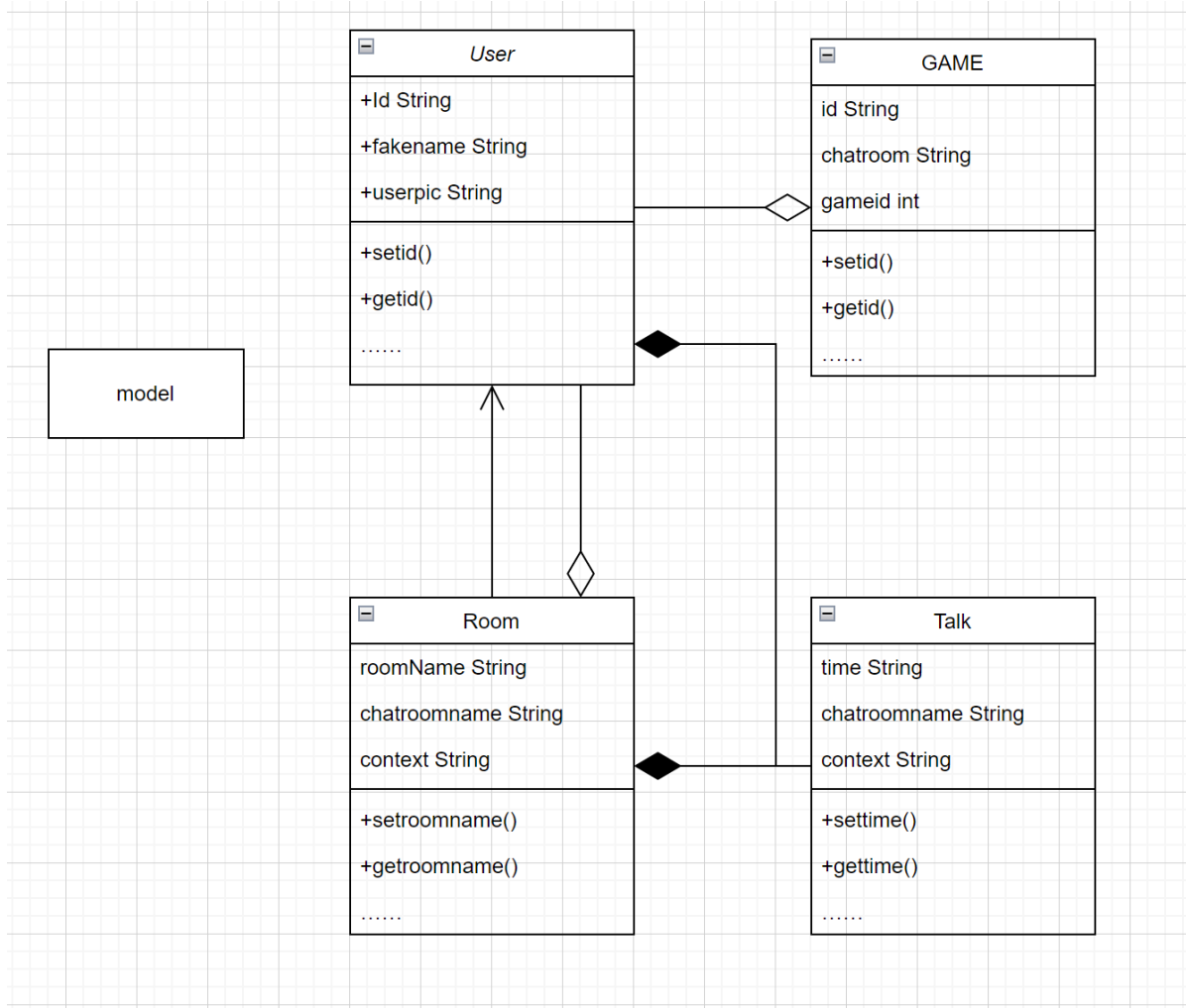
ROOM	
PK	<u>ROOMNAME VARCHAR(255)</u>
	ROOM_OWER_ID VARCHAR(255)
	ROOMPIC VARCHAR(255)
	NUMOFPEOPLE INT
	PASS_WORD VARCHAR(255)
	MEMBERS_ID VARCHAR(255)

GAME	
PK	<u>ID VARCHAR(255)</u>
	CHATROOM VARCHAR(255)
	TIME VARCHAR(255)
	GAMEID INT
	WINNER INT
	TURN INT
	BOARD TEXT
	PLAYERS VARCHAR(255)
	ISEND INT

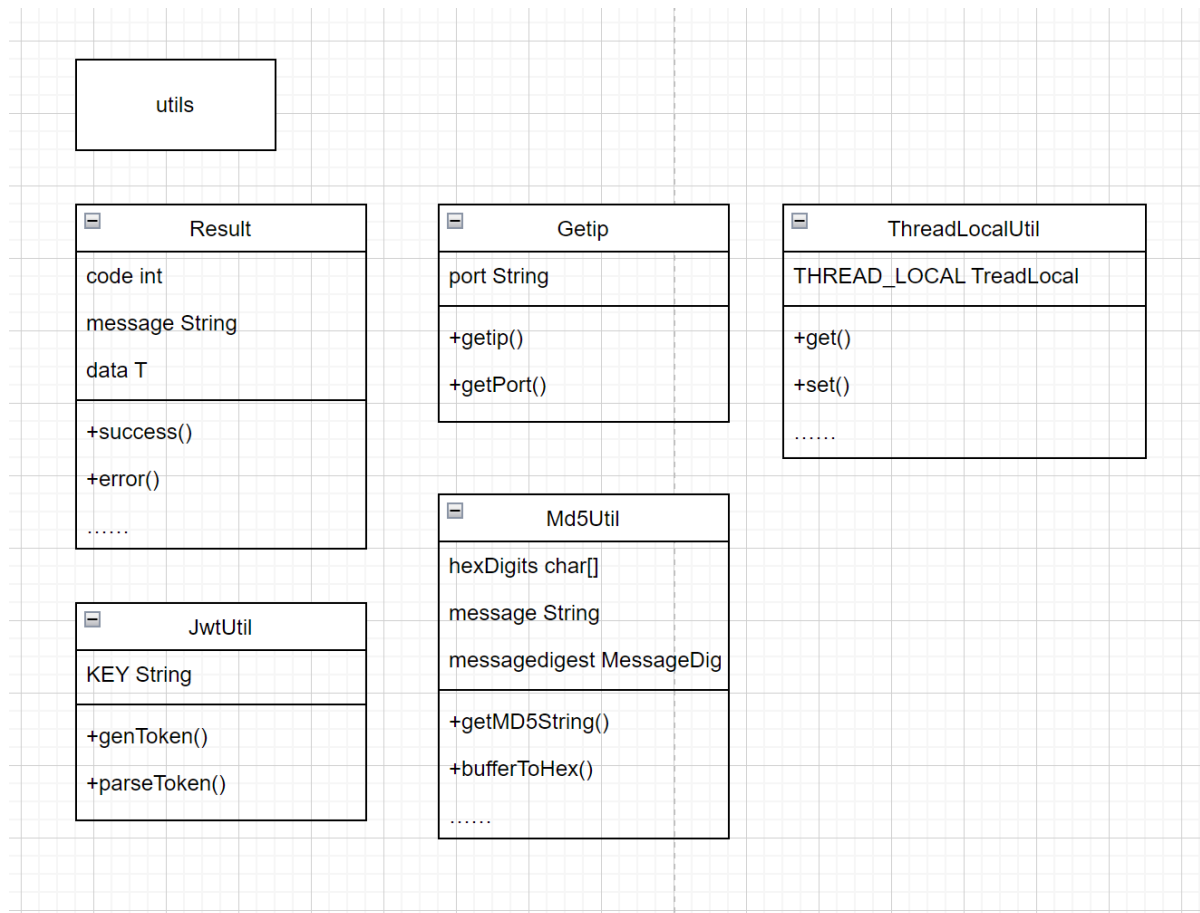
TALK	
PK	<u>TIME VARCHAR(255)</u>
	CHATROOMNAME VARCHAR(255)
	SENDERID VARCHAR(255)
	SENDERNAME VARCHAR(255)
	SENDERPIC VARCHAR(255)
	CONTEXT TEXT
	TYPE VARCHAR(255)



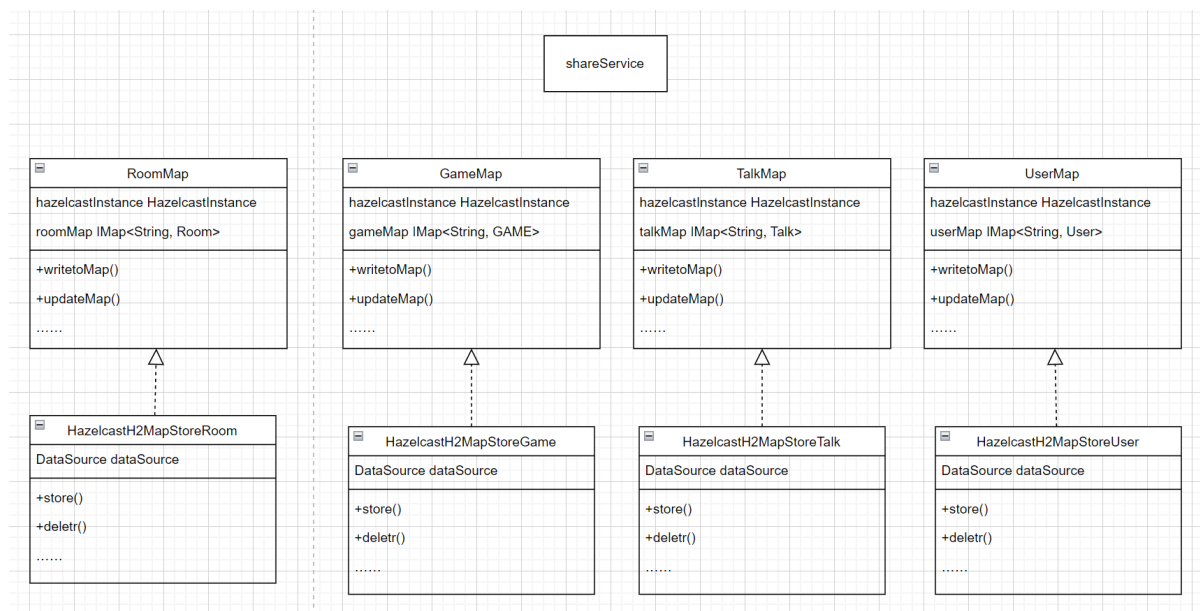
(2) model模块



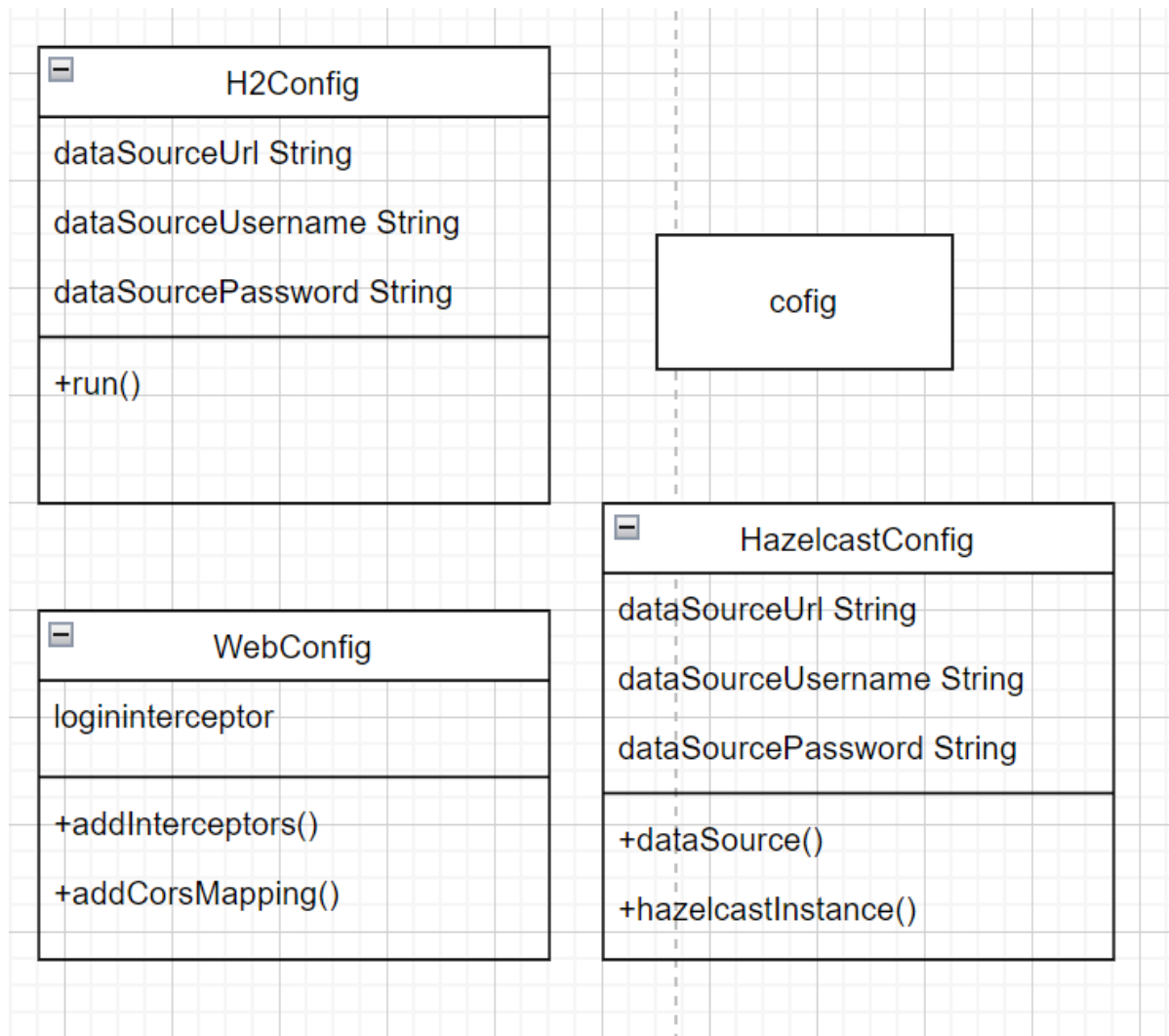
(3) Utils模块



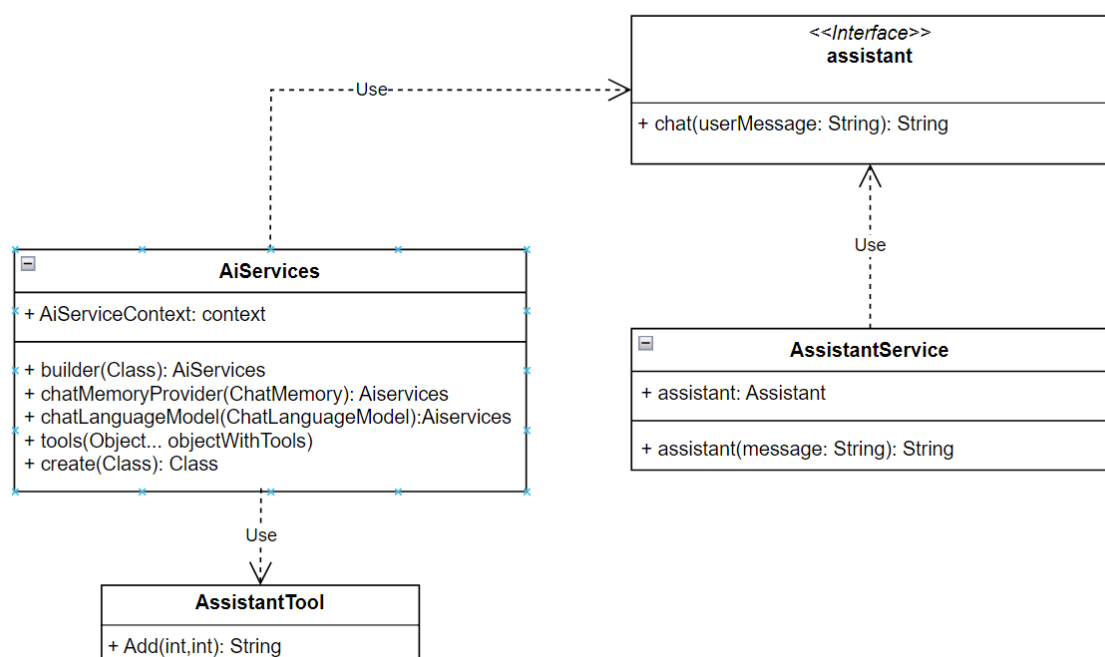
(4) Service模块



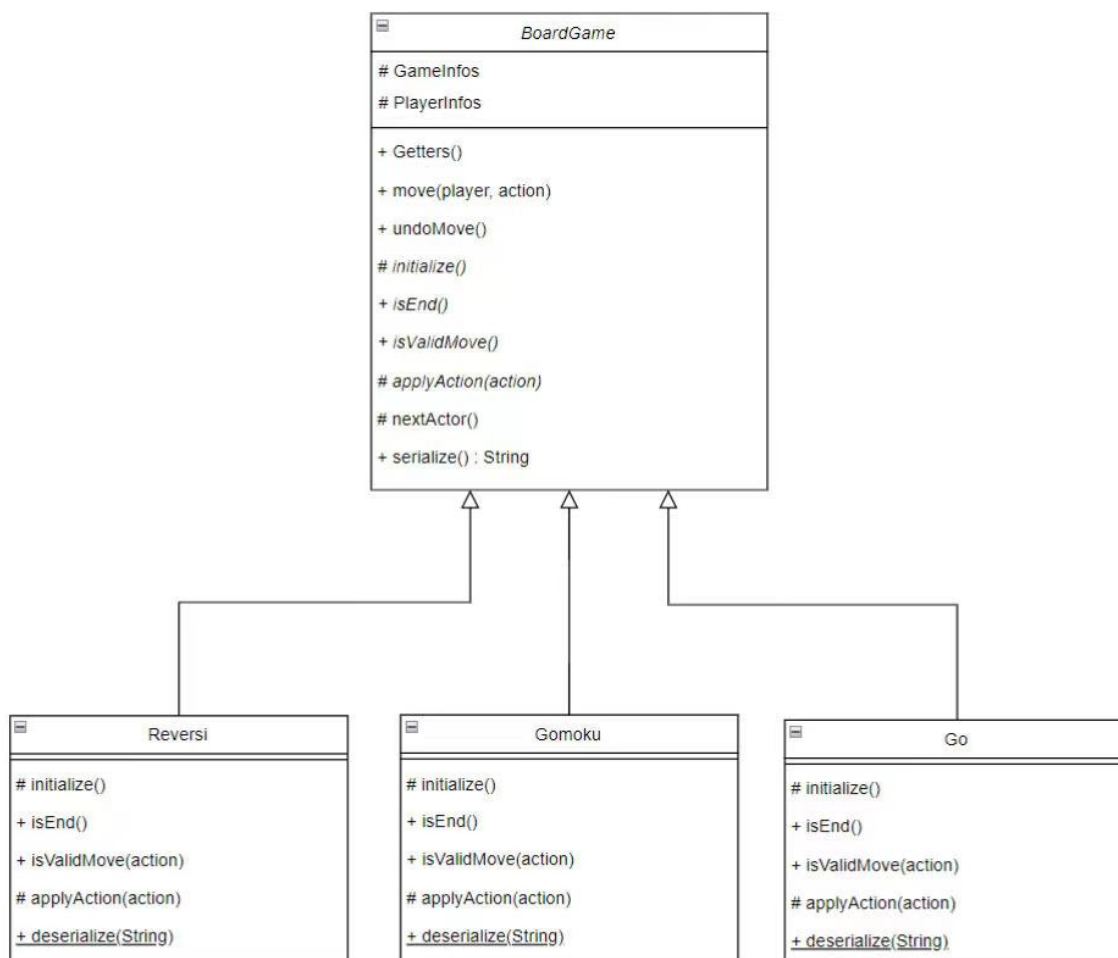
(5) config模块



(6) 聊天机器人模块



(7) 游戏模块



五、设计模式

在设计和开发去中心化的分布式局域网匿名聊天Web应用过程中，我们采用了多种设计模式，以确保系统的模块化、可扩展性和可维护性。以下是本系统中使用的主要设计模式：

1. 单例模式 (Singleton Pattern)

应用场景： Hazelcast实例管理

描述： 单例模式确保一个类只有一个实例，并提供一个全局访问点。在本系统中，Hazelcast实例用于分布式缓存和数据共享，我们采用单例模式来确保其在应用中只有一个实例。

2. 观察者模式 (Observer Pattern)

应用场景： 消息通知和事件处理

描述： 观察者模式定义了一种一对多的依赖关系，使得一个对象状态发生变化时，所有依赖于它的对象都会得到通知并被自动更新。在本系统中，聊天室中的消息通知和用户状态变化使用了观察者模式。

3. 策略模式 (Strategy Pattern)

应用场景： 各种游戏机制

描述： 策略模式定义了一系列算法，并将每个算法封装起来，使它们可以互换。本系统中的游戏机制采用策略模式，根据不同的输入选择不同的回复策略。

4. 装饰者模式 (Decorator Pattern)

应用场景：消息功能扩展

描述：装饰者模式允许动态地给对象添加额外的职责。在本系统中，消息发送模块使用装饰者模式，以添加不同类型的消息功能（如文本、图片、视频等）。

5. 生成器模式 (Builder Pattern)

应用场景：前端窗口

描述：生成器模式将一个复杂对象的构建过程与其表示分离，使得同样的构建过程可以创建不同的表示。在本系统中，可以使用生成器模式分组件创建前端窗口。

6. 责任链模式 (Chain of Responsibility Pattern)

应用场景：事件的抛出

描述：责任链模式为请求创建一个接收对象的链。每个接收对象包含对另一个接收对象的引用。如果请求在一个对象中处理不了，则传递给下一个对象。在前端中，子组件抛出的事件会父组件进一步向上抛出，直至被根节点分配给指定组件进行处理。