aA 冠军队

AI 虚拟角色聊天平台
架构设计文档

设计小组：aA 冠军队

成员：宋子晗（前端开发工程师）

刘浩（后端开发工程师）

项目架构设计文档（B）

目录

1. 引言 3
2. 系统概述 3
2.1 系统目标 3
2.2 系统功能 3
2.3 系统约束 3
3. 架构设计 4
3.1 架构图 4
3.2 系统组件说明 4
3.2.1 前端 4
3.2.2 后端 5
3.2.3 数据库 5
3.3 系统特点 5
4. 模块设计 5
4.1 前端模块设计 5
4.2 后端模块设计 ..... **错误!未定义书签。**
5. 技术选型 6
5.1 前端技术 6
5.2 后端技术 6
6. 数据流与交互 6
6.1 文本消息流程 6
7. 安全设计 7
8. 性能与扩展性 7
8.1 前端优化 7
8.2 后端优化 7
8.3 系统扩展性 7
9. 测试计划 8
10. 部署与运维 8
11. 总结 ..... **错误!未定义书签。**

1. 引言

文档目的

本文档描述了系统的架构设计，包括系统的整体结构、模块划分、各模块间的交互以及关键技术选型。目标是为开发人员、测试人员和其他相关人员提供详细的技术指导。

适用范围

本文档适用于系统开发过程中的设计阶段，尤其是在开发、集成和部署阶段。

2. 系统概述

2.1 系统目标

该系统旨在为用户提供一个高度智能化和交互性极强的客服平台，利用自然语言处理（NLP）、语音识别与语音合成（TTS）技术实现人与虚拟角色的高效沟通。系统支持多角色选择与个性化交互，每个角色可根据用户偏好呈现不同的行为与回答风格，同时支持文本与语音双向输入输出，让用户可以灵活选择交互方式。

在功能层面，系统实现了实时聊天、消息记录、会话管理、多设备同步等核心能力，确保用户在不同设备间无缝切换，保持会话状态一致。通过 WebSocket 与流式传输技术，系统能够实时推送消息与音频数据，实现低延迟、高响应的聊天体验。

在性能与可靠性方面，系统采用前端按需加载技术减少初次加载时间；后端引入缓存机制（Redis），支持高并发请求处理。数据库方面，通过索引优化，确保海量用户数据与聊天记录的高效存储与查询。

2.2 系统功能

系统最终实现的功能包括但不限于以下几类：

用户登录与身份管理：

1. 提供安全的用户注册、登录、注销功能。

角色选择与切换：

1. 提供多种虚拟角色选择，每个角色具有不同的交互风格和背景设定。
2. 支持用户在聊天过程中实时切换角色，角色切换后聊天上下文清空。
3. 每个角色配置头像、背景、语音风格等个性化信息。

文本与语音输入：

1. 用户可以通过文本输入发送消息，系统实时响应。
2. 支持语音输入，系统通过语音识别将语音转换为文本，并生成回复。

消息记录与实时聊天：

1. 支持实时聊天，确保用户与虚拟角色的交互即时响应。

前后端交互与服务支持：

1. 前端通过 REST API 或 WebSocket 与后端交互，实现消息的实时传输。
2. 后端提供高并发处理能力、RESTful API 等服务。

2.3 系统约束

系统在设计和实现过程中需要遵循以下约束条件：

浏览器兼容性：

系统前端支持主流浏览器：Chrome、Firefox、Edge，保证跨平台用户体验。

部署环境：

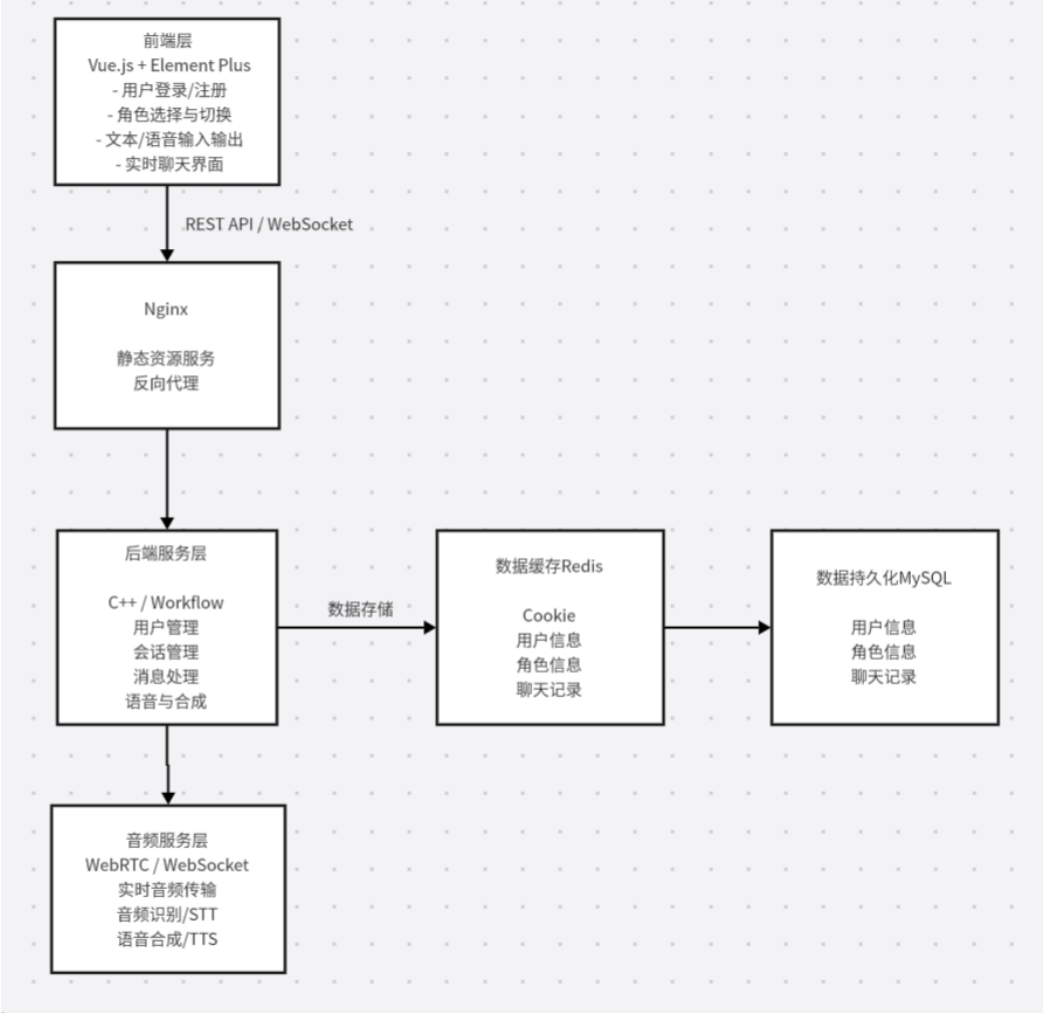
系统部署在

安全性与数据保护
用户数据加密存储，并通过认证和授权机制保护敏感信息。
支持访问控制策略，限制用户不同角色对系统功能的权限。
性能与可靠性
系统需满足高并发访问场景下的稳定性要求。

3. 架构设计

3.1 架构图

描述：
本系统采用 典型的前后端分离架构，同时支持文本与语音的实时交互。前端使用 Vue.js 构建界面，并通过 WebSocket 与后端进行实时通信。后端基于 C++/Workflow 提供 API 服务，负责处理用户请求、管理会话、处理语音流。数据库使用 Redis 缓存+MySQL 持久化。存储用户数据、角色信息和聊天记录等。。



3.2 系统组件说明

3.2.1 前端
技术栈： Vue.js、Element Plus、Axios、WebSocket
功能：

1. 负责用户界面展示与交互。
2. 提供多角色选择和切换界面。
3. 支持文本消息输入和语音输入。
4. 使用 WebSocket 或 WebRTC 实时接收后端消息和音频流。
5. 消息列表自动滚动、加载动画和状态提示（正在输入/思考）。

3.2.2 后端
技术栈： C++11、Workflow、wfrest
功能：
1. 提供 RESTful API 服务处理用户请求。
2. 管理聊天会话和消息队列，实现会话的持久化和多用户并发。
3. 调用语音识别 (ASR) 和语音合成 (TTS) 服务，实现语音聊天功能。
4. 支持 AI 角色对话生成，包括上下文管理和多角色个性化回复。
5. 提供安全认证和授权接口，保证用户数据安全。

3.2.3 数据库
技术栈： Redis、MySQL
功能：
1. 存储用户账户信息、角色配置和聊天历史。
2. 提供高性能查询和扩展能力，支持多会话和多角色管理。
3. 结合 Redis 缓存，存储活跃会话状态，提高响应速度。

3.2.4 音频处理
技术栈： 七牛云 TTS/ASR
功能：
1. 实现前端与后端之间的低延迟音频流传输。
2. 通过 ASR 将语音转换为文本输入给 AI 角色处理。
3. 将 AI 回复文本通过 TTS 转为音频

3.3 系统特点
前后端分离：前端专注界面交互，后端处理业务逻辑和数据存储。
多角色支持：用户可选择不同角色与 AI 进行对话，角色个性化配置可独立加载。
语音与文本融合：同时支持文本聊天和语音聊天，实现多模态交互体验。
高可扩展性：基于 wfrest 蓝图，接口模块分离，方便扩展

4. 模块设计

4.1 前端模块设计
模块 1：用户管理
功能描述：负责用户的注册、登录、身份验证。
登录注册账号与密码通过正则表达式添加了数据校验，账号只能包含数字、字母和下划线，密码要求大于 6 位
接口：
POST /api/user/login：用户登录接口，参数：user:账号，pwd:经过每位加 1 并 md5 加密后的密码字符串
POST /api/user/register：用户注册接口，参数：user:账号，pwd:经过每位加 1 并 md5 加密后的密码字符串

模块 2：聊天主界面—左边栏
功能描述：实现 AI 角色的选择、头像与信息的显示
接口：
POST /api/assistant/list 获取角色信息
模块 3：聊天主界面—信息框
功能描述：获取用户与 AI 传入的信息并显示在界面上
POST /api/assistant/stream：上传用户消息并接收后端返回消息
GET /public/{id}.mp3：获取语音消息

5. 技术选型

5.1 前端技术

Vue.js：用于构建单页面应用，支持响应式设计。
Element Plus：用于构建 UI，提供丰富的组件和样式。
Axios：用于发送 HTTP 请求，获取数据。
WebSocket：用于实现实时的消息传输与音频流传输。

5.2 后端技术

Workflow：高性能 C++异步服务器引擎，适合处理高并发的请求，且充当数据库访问客户端。
wfrest：用于构建 RESTful API 服务。
Redis、MySQL：NoSQL 内存数据库做缓存，配合关系型数据库持久化，高效存储用户信息、消息数据等。

6. 数据流与交互

6.1 文本消息流程

6.1.1 用户输入消息

用户在前端输入框中输入文本消息并点击发送。消息请求发送前端通过 POST /api/chat 接口，将用户消息、会话 ID、角色信息等参数发送至后端。后端消息处理后端接收请求后，对消息进行解析和处理。根据用户选择的角色，调用 AI 对话引擎生成回复内容。消息存入数据库（MongoDB），保证会话历史可追溯。消息推送后端通过 WebSocket 将处理后的消息推送到前端。前端实时更新聊天窗口，展示新的消息。

6.1.2 语音消息流程

用户语音输入前端通过麦克风调用浏览器的 MediaRecorder / WebRTC API，获取音频流数据。音频上传前端将音频流片段化（chunk），通过 WebSocket 或 POST /api/audio/upload 上传到后端。后端语音处理后端对接第三方 STT（Speech-to-Text）服务，将音频转换为文本。转换后的文本交由 AI 引擎生成回复内容。将回复内容同时存储为 文本消息 与 TTS 语音消息。返回音频流后端将生成的 TTS 音频通过 WebSocket 流式传输到前端。前端实时播放音频，同时在消息窗口显示对应文本内容。

6.1.3 WebSocket 流程建立连接

用户进入聊天界面后，前端通过 WebSocket 与后端建立长连接。WebSocket 连接用于实时收发消息和音频数据。消息收发前端发送的文本或语音消息经后端处理后，通过 WebSocket 推送到对方。这样可实现实时通信，而不依赖频繁的 HTTP 轮询。音频流传输后端将生成的语音回复（TTS 转换结果）分片流式推送给前端。前端边接收边播放，用户几乎可以无延迟地听到虚拟角色的语音回复。连接维护与状态管理系统支持心跳机制检测

WebSocket 连接状态。若连接断开，前端可自动重连，确保用户体验稳定。消息确认机制（ACK）用于保证消息不会丢失。

6.1.4 数据流特点

实时性强：

消息和音频流通过 WebSocket 长连接实现低延迟交互。

可靠性高：

采用消息存储 + ACK 机制，确保消息和语音不会丢失。双通道支持：文本与语音两种消息类型均支持，并可同时展示。可扩展性：架构支持水平扩展，能应对高并发场景。

7. 安全设计

身份认证：使用 JWT（JSON Web Token）进行用户认证，所有敏感接口需要带上 Authorization 头。

数据加密：通过 HTTPS 协议加密前后端数据传输，确保传输过程中的数据安全。

8. 性能与扩展性

8.1 前端优化

懒加载与按需加载

使用 Vue Router 的路由懒加载，仅在需要时加载对应组件，减少首屏加载时间。

使用 Element Plus 的按需引入（按组件打包），避免引入整个库，提高打包效率。

资源压缩与缓存

对 JS、CSS 进行压缩（Terser、cssnano），减小资源体积。

对静态资源（图片、音频、字体）启用浏览器缓存或 CDN 加速。

虚拟滚动与消息列表优化

当聊天记录过多时，使用虚拟列表（Virtual Scroller）渲染可视区域，减少 DOM 节点，提高渲染性能。

8.2 后端优化

缓存机制

对频繁请求的数据使用 Redis 缓存，例如用户信息、角色列表等，减少数据库压力。

对聊天会话历史可使用短期缓存加快接口响应速度。

异步与并发处理

对语音识别、文本生成等耗时任务使用异步队列（如 RabbitMQ/Kafka），避免阻塞主线程。

对高并发请求使用线程池或异步 IO 提升吞吐量。

接口与数据库优化

SQL 查询使用索引、分页查询，避免全表扫描。

数据库分库分表，支持海量用户数据存储和高并发访问。

后端服务采用连接池管理数据库连接，减少连接开销。

8.3 系统扩展性

水平扩展

后端服务支持多实例部署，通过负载均衡（如 Nginx、HAProxy）分发请求。

WebSocket 服务可通过集群模式共享消息（使用 Redis Pub/Sub 或消息中间件实现）。

模块化设计

前端组件化，便于新增功能模块（如语音输入、表情包）。

后端服务拆分为独立微服务（用户服务、聊天服务、角色服务），便于独立扩展与维护。

日志与监控

集成日志系统（ELK、Prometheus+Grafana）监控服务性能和用户行为，快速定位性能瓶颈。

9. 测试计划

单元测试
目标：验证系统中每个模块或功能单元的逻辑正确性。
内容：
功能函数输入输出的正确性验证
异常处理与边界条件的测试
数据校验规则（如必填字段、格式校验）的测试
第三方依赖或外部接口调用的模拟（Mock）验证
工具：可使用 JUnit、pytest、Mocha 等框架。
集成测试
目标：确保模块之间的接口与数据交互正常，系统整体业务流程通畅。
内容：
接口调用链路测试
数据在不同模块之间传递的完整性与一致性验证
异常情况下的事务回滚与容错能力测试
前后端联调，确保 API 与页面交互无误
工具：Postman、Swagger、JUnit、Cypress 等。
性能测试
目标：评估系统在不同负载条件下的性能表现，确保系统具备可扩展性和稳定性。
内容：
负载测试：逐步增加用户并发量，测试系统在高负载下的响应时间和资源使用情况。
压力测试：在超过预期极限的条件下运行，检验系统的崩溃点及恢复能力。
稳定性/耐久性测试：长时间运行大规模业务请求，验证系统是否存在内存泄漏、性能衰减等问题。
容量测试：确定系统支持的最大用户数或数据处理量。
工具：JMeter、LoadRunner、Locust 等。

10. 部署方式

部署环境
前端：通过 Nginx 部署静态文件，支持 CDN 加速。

后端：
假设~/为项目根目录
1. nginx
nginx 通过配置文件启动即可
nginx -c ~/server/config/nginx.conf
2. 服务器
启动服务进程即可
~/server/AIchat