架构设计

AI Talk 项目架构文档

1. 项目概述

AI Talk 是一个基于 Spring Boot 的人工智能对话系统，支持角色扮演、语音合成和历史对话管理功能。该项目整合了阿里云 DashScope 平台的文本生成和语音合成能力，允许用户与不同角色进行对话，并以相应角色的声音进行回复。

2. 技术架构

2.1 后端技术栈

- **核心框架**: Spring Boot 3.5.5
- **AI框架**: Spring AI 1.0.2
- **数据库**: MySQL 8.0.33
- **持久层**: MyBatis-Plus 3.5.11
- **日志框架**: Log4j2 2.5.0
- **AI服务**: 阿里云 DashScope (兼容 OpenAI API)
- **Java版本**: Java 17

2.2 前端技术栈

- **框架**: Vue 3.5.21
- **构建工具**: Vite 6.x
- **包管理**: npm

2.3 核心依赖

组件	版本	用途
spring-ai-starter-model-openai	1.0.2	OpenAI 兼容模型支持
spring-ai-starter-model-ollama	1.0.2	Ollama 模型支持
dashscope-sdk-java	2.17.1	阿里云 DashScope SDK
mysql-connector-java	8.0.33	MySQL 数据库连接
mybatis-plus-spring-boot3-starter	3.5.11	MyBatis-Plus 持久层框架
spring-ai-markdown-document-reader	1.0.0-M6	Markdown 文档读取（RAG功能）

3. 项目结构

```
E:\AItalk\AItalk\
├── AItalkFrontend/              # 前端项目
```

```
├── db/                          # 数据库脚本
├── src/                         # 后端源码
│   ├── main/
│   │   ├── java/
│   │   │   └── com.yyxxlu.aitalk/
│   │   │       ├── componet/    # 组件模块
│   │   │       ├── config/      # 配置模块
│   │   │       ├── controller/  # 控制器层
│   │   │       ├── entity/      # 实体类
│   │   │       ├── mapper/      # 数据访问层
│   │   │       ├── rag/         # RAG相关模块
│   │   │       ├── service/     # 业务逻辑层
│   │   │       └── AiTalkApplication.java # 启动类
│   │   └── resources/
│   │       ├── mapper/          # MyBatis映射文件
│   │       └── application.yaml # 配置文件
│   └── test/                    # 测试代码
└── pom.xml                      # Maven配置文件
```

4. 核心模块设计

4.1 AI 对话模块

主要组件:

- aiController: 处理所有对话相关的 HTTP 请求
- aiConfig: 配置 ChatClient 和聊天记忆
- PromptConfig: 构建角色扮演的系统提示词

功能特性:

- 基础文本对话 (/chat)
- 角色扮演对话 (/voice)
- 对话历史记录管理 (/history)
- 音色克隆 (/copyVoice)

4.2 语音合成模块

主要组件:

- VoiceService: 语音服务接口
- VoiceServiceimpl: 语音服务实现类
- copyVoice: 音色克隆组件

功能特性:

- 基础文本转语音 (TTS)
- 角色定制语音合成
- 音色克隆功能

4.3 数据管理模块

主要组件:

- RoleProfile: 角色信息实体
- RoleProfileMapper: 角色信息数据访问接口
- RoleProfileMapper.xml: MyBatis 映射文件

数据表结构:

- `roleprofile`: 存储角色信息，包括背景、性格、说话风格、知识范围和音色ID等

4.4 聊天记忆模块

主要组件:

- MysqlChatMemory: 自定义 MySQL 聊天记忆实现
- HistoryMessageVo: 历史消息展示对象

功能特性:

- 对话历史记录存储
- 对话上下文管理

4.5 RAG 模块（待完善）

主要组件:

- AppDocumentLoader: Markdown 文档加载器
- AppVectorStoreConfig: 向量存储配置

功能特性:

- Markdown 文档加载
- 向量存储（计划中）

5. API 接口设计

5.1 对话接口

接口	方法	说明
`/chat`	GET	基础文本对话
`/voice`	GET	角色语音对话
`/history`	GET	获取对话历史
`/roleList`	GET	获取角色列表
`/copyVoice`	GET	音色克隆

5.2 请求参数

文本对话

```
GET /chat?prompt={对话内容}
```

角色语音对话

```
GET /voice?prompt={对话内容}&id={角色ID}
```

对话历史

```
GET /history?id={会话ID}
```

6. 配置说明

6.1 application.yaml 配置

主要配置项包括:

- 数据库连接配置
- Redis 配置
- AI 服务配置 (DashScope 兼容 OpenAI)
- 日志级别配置

6.2 环境变量

- `BL_API_KEY`: 阿里云 DashScope API 密钥

7. 部署架构

7.1 后端部署

1. 使用 Maven 构建项目:
   ```
   mvn clean package
   ```
2. 运行应用:
   ```
   java -jar target/aiTalk-0.0.1-SNAPSHOT.jar
   ```

7.2 前端部署

1. 安装依赖:
   ```
   npm install
   ```
2. 构建项目:
   ```
   npm run build
   ```

3. 预览构建结果:

```
npm run preview
```

8. 数据库设计

8.1 roleprofile 表

存储角色信息，包括:

- 角色基本信息（ID、名称）
- 角色设定（背景、性格、说话风格、知识范围）
- 语音相关（对话示例、音色ID）
- RAG 相关（是否启用RAG、知识库名称）

9. 扩展性设计

9.1 角色扩展

通过在 `roleprofile` 表中添加新的角色记录，可以轻松扩展新的角色。

9.2 模型扩展

通过配置不同的 AI 模型参数，可以支持多种大语言模型。

9.3 功能扩展

模块化设计使得添加新功能（如 RAG、多轮对话优化等）更加容易。

10. 未来规划

1. 完善 RAG 功能，支持基于知识库的问答
2. 增强对话记忆管理，支持持久化存储
3. 优化前端界面，提升用户体验
4. 增加更多角色和语音定制选项
5. 支持更多语言和音色

11.项目分工

杨永： 项目设计 & 技术选型 & 项目搭建
陈億： 项目设计 & 技术选型 & 编码开发 & 测试优化