

**毕业设计**

|  |  |
| --- | --- |
| 题目 | **基于QAnything的智能问答系统设计与实现** |
| **英文题目** | **Design and Implementation of Intelligent Question-Answering System Based on QAnything** |

**学生姓名**： **徐阳阳 申请学位门类： 工学**

**学 号： 2021213737**

**专 业：** **软件工程G**

**学 院**： **软件学院**

**指导教师：** **王红玲** **职称：** **副教授**

**二0二五年四月 三 十 日**

作 者 声 明

本人以信誉郑重声明：所呈交的学位毕业设计（论文），是本人在指导教师指导下由本人独立撰写完成的，没有剽窃、抄袭、造假等违反道德、学术规范和其他侵权行为。文中引用他人的文献、数据、图件、资料均已明确标注出，不包含他人成果及为获得东华理工大学或其他教育机构的学位或证书而使用过的材料。对本设计（论文）的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本毕业设计（论文）引起的法律结果完全由本人承担。

本毕业设计（论文）成果归东华理工大学所有。

特此声明。

毕业设计（论文）作者（签字）：

签字日期： 年 月 日

本人声明：该学位论文是本人指导学生完成的研究成果，已经审阅过论文的全部内容，并能够保证题目、关键词、摘要部分中英文内容的一致性和准确性。

学位论文指导教师签名：

年 月 日

摘 要

近年来，随着自然语言处理和深度学习技术的突破性进展，知识库问答系统已成为智能信息检索领域的研究热点。然而，现有系统普遍存在两个关键局限：一是高度依赖云端服务，无法满足数据敏感场景下使用需求；二是文件格式兼容性有限，难以处理多样化的知识载体。这些问题严重制约了知识管理系统在现实场景中的应用效果。

本课题基于QAnything的架构思想，设计并实现了一个本地化智能问答系统，针对上述两个场景给出了对应的解决方案，既支持敏感数据数据的问答，又提供多格式文档的解析。系统采用Vue.js构建前端交互界面，NestJS实现后端服务，结合MySQL进行知识存储与管理，并集成大语言模型实现自然语言问答功能。主要价值体现在以下三点：首先，为各类组织构建安全可靠的私有知识库提供解决方案；其次，通过智能化的知识提取和检索技术，显著提升组织内部的知识共享效率；最后，ChatBot支持多个用户之间共享模型配置，提供实时的数据资源共享。

研究成果将推动知识管理技术向更智能、更易用的方向发展。

关键词：人工智能； 智能问答系统； 知识库问答； ChatBot； QAnything

**ABSTRACT**

In recent years, with the breakthrough progress of natural language processing and deep learning technologies, knowledge base question-answering systems have become a research hotspot in the field of intelligent information retrieval. However, the existing systems generally have two key limitations: First, they are highly dependent on cloud services and cannot meet the usage requirements in data-sensitive scenarios; Second, the compatibility of file formats is limited, making it difficult to handle diverse knowledge carriers. These problems seriously restrict the application effect of the knowledge management system in real scenarios.

Based on the architectural concept of QAnything, this project has designed and implemented a localized intelligent question-answering system, providing corresponding solutions for the above two scenarios. It not only supports question-answering for sensitive data but also offers the parsing of multi-format documents. The system adopts Vue.js to build the front-end interactive interface, NestJS to implement the back-end services, combines MySQL for knowledge storage and management, and integrates a large language model to realize the natural language question-answering function. The main values are reflected in the following three points: Firstly, it provides solutions for various organizations to build secure and reliable private knowledge bases; Secondly, through intelligent knowledge extraction and retrieval technologies, the efficiency of knowledge sharing within the organization is significantly enhanced; Finally, ChatBot supports sharing model configurations among multiple users, providing real-time data resource sharing.

The research results will promote the development of knowledge management technology in a more intelligent and user-friendly direction.

**Key words**: Artificial intelligence; Intelligent question-answering system; Knowledge base Q&A; ChatBot; QAnything

目 录

[摘 要 Ⅰ](#_Toc198508407)

[**ABSTRACT** Ⅱ](#_Toc198508408)

[第1章 绪论 1](#_Toc198508409)

[1.1 研究背景 1](#_Toc198508410)

[1.2 研究意义 1](#_Toc198508411)

[第2章 技术综述 2](#_Toc198508412)

[2.1 QAnything架构分析 2](#_Toc198508413)

[2.1.1 数据接入层 2](#_Toc198508414)

[2.1.2 知识加工层 3](#_Toc198508415)

[2.1.3 向量化与存储层 3](#_Toc198508416)

[2.1.4 问答引擎层 4](#_Toc198508417)

[2.1.5 应用交互层 4](#_Toc198508418)

[2.2 技术选型依据 4](#_Toc198508419)

[2.2.1 Vue3 + TypeScript 的优势 5](#_Toc198508420)

[2.2.2 NestJS的分层架构 5](#_Toc198508421)

[2.2.3 MySQL关系模型设计 6](#_Toc198508422)

[第3章 系统需求分析 8](#_Toc198508423)

[3.1 功能性需求 8](#_Toc198508424)

[3.1.1 快速开始问答 8](#_Toc198508425)

[3.1.2 知识库问答 8](#_Toc198508426)

[3.1.3 聊天Bot 8](#_Toc198508427)

[第4章 系统设计 9](#_Toc198508428)

[4.1 总体架构设计 9](#_Toc198508429)

[4.1.1 系统架构分析 9](#_Toc198508430)

[4.1.2 数据流设计 11](#_Toc198508431)

[4.2 数据库设计 12](#_Toc198508432)

[4.2.1表结构 12](#_Toc198508433)

[4.2.1 ER图 15](#_Toc198508434)

[第5章 系统实现 17](#_Toc198508435)

[5.1 前端实现 17](#_Toc198508436)

[5.1.1 快速开始 17](#_Toc198508437)

[5.1.2 知识库问答 21](#_Toc198508438)

[5.1.3 Bots 22](#_Toc198508439)

[5.2 后端实现 23](#_Toc198508440)

[5.2.1 分层架构的实现 24](#_Toc198508441)

[5.2.3 文件解析服务 27](#_Toc198508442)

[5.3 部署 27](#_Toc198508443)

[5.3.1 前端挂载 28](#_Toc198508444)

[5.3.2 服务端部署 29](#_Toc198508445)

[5.3.3 数据库部署 29](#_Toc198508446)

[结束语 32](#_Toc198508451)

[致 谢 32](#_Toc198508451)

[参考文献 33](#_Toc198508452)

第1章 绪论

## 1.1 研究背景

在当代社会数字化转型进程中，企事业单位、教育机构及政府组织逐渐形成了规模庞大的知识资产库，涵盖技术规范、业务流程、教学资源和行政文件等多种类型。但这类资源的有效管理与应用仍存在显著瓶颈。现有以关键词匹配为核心的搜索技术普遍存在召回率低、排序不合理等缺陷，无法适应用户对知识服务的即时性与准确性要求。而融合认知智能的交互式问答平台则通过拟人化对话模式，大幅优化了知识传递效能。

人工智能领域的突破性进展使得知识驱动型问答系统（KBQA，Knowledge-Based Question Answering）在信息科学界获得广泛关注。早期问答技术多采用结构化查询或预设规则，当应用于组织机构内部的非标准化文档，如PDF、Word、Excel等文档资源时表现欠佳。最新研究表明，检索增强生成[1]（Retrieval-Augmented Generation，RAG）为知识库问答提供了新思路：RAG的核心思想是先通过检索从知识库中找出相关文档片段；再通过生成由大语言模型生成自然语言回答。优势在于无需对预训练语言模型进行参数微调，可直接利用现有知识库作为外部知识源；通过检索增强生成（RAG）机制，模型生成的回答能够紧密锚定知识库中的事实依据，显著提升回答的事实准确性，有效减少大语言模型常见的幻觉现象。

作为典型实践案例，网易有道开源的QAnything本地知识库问答系统，其设计目标是为用户提供安全可控、多格式兼容且检索高效的私有化知识管理解决方案。其核心特点包括：（1）多模态文件解析能力，全面支持PDF、Word、Excel、PPT等文档以及图片数据源；（2）智能化知识处理流程，通过语义感知的文本分割算法和基于bge模型的向量化技术，构建高效的本地化知识索引；（3）检索增强生成（RAG）架构，将语义检索与大语言模型生成能力相结合，显著提升问答的准确性与事实性。该系统特别适用于对数据隐私要求严格的金融、医疗等行业场景，为组织级知识管理提供了开箱即用的技术方案。

## 1.2 研究意义

本研究基于QAnything的架构思想，设计并实现了一套本地化智能问答系统，该系统在知识管理领域具有重要的理论价值和实践意义。从技术实现层面来看，系统采用Vue.js构建响应式前端交互界面，结合NestJS框架开发高性能后端服务，通过MySQL实现结构化知识存储与管理，并创新性地集成大语言模型实现自然语言问答功能，形成了一套完整的技术解决方案。该系统最突出的创新点在于实现了对PDF、Word、Excel、PPT等多种格式文档的智能解析与语义理解能力，突破了传统知识管理系统在文件格式兼容性方面的局限。

在实践应用层面，本研究为企事业单位构建私有化知识库提供了全新的技术路径。系统采用本地化部署模式，所有数据处理和存储均在组织内部完成，从根本上解决了云端服务可能存在的数据安全隐患，确保了核心知识资产的自主可控性。特别值得一提的是，系统创新的ChatBot共享机制允许多个用户基于同一套模型配置进行协同问答，不仅降低了技术使用门槛，还通过标准化的知识交互模式促进了团队间的信息同步和认知对齐，为组织知识管理提供了全新的协作范式。

从行业发展角度来看，本研究成果将推动知识管理系统向智能化、易用化方向转型升级。系统将前沿的人工智能技术与传统知识管理需求深度融合，通过自然语言交互界面大大降低了系统使用难度，同时借助大语言模型的强大理解能力显著扩展了系统功能边界。这种智能化升级不仅体现在技术层面，更重塑了人机知识交互的方式，使知识获取从被动检索转变为主动对话，极大提升了知识传递的效率和体验。随着数字化转型的深入推进，本研究提出的技术方案有望在政府机构、教育科研、企业研发等多个领域产生示范效应，为构建新一代智能知识基础设施提供重要参考。

第2章 技术综述

本章重点剖析QAnything的系统架构设计以及问答系统的技术选型依据。QAnything架构采用分层模块化设计，各功能层之间通过标准化接口进行通信，实现了高内聚低耦合的系统特性。在技术选型方面，本研究遵循严格的评估流程，主要考量因素包括技术栈在开源社区的活跃度、开发维护的便利性及学习曲线。

## 2.1 QAnything架构分析

QAnything这款多模态问答系统整体架构采用“数据接入→知识加工→向量化存储→检索增强生成→应用交互”的五层结构，该架构逻辑清晰，职责明确，便于系统根据业务需求进行快速迭代。它的核心功能是将非结构化知识转化为可交互的问答形式。QAnything系统架构图如图2-1所示。

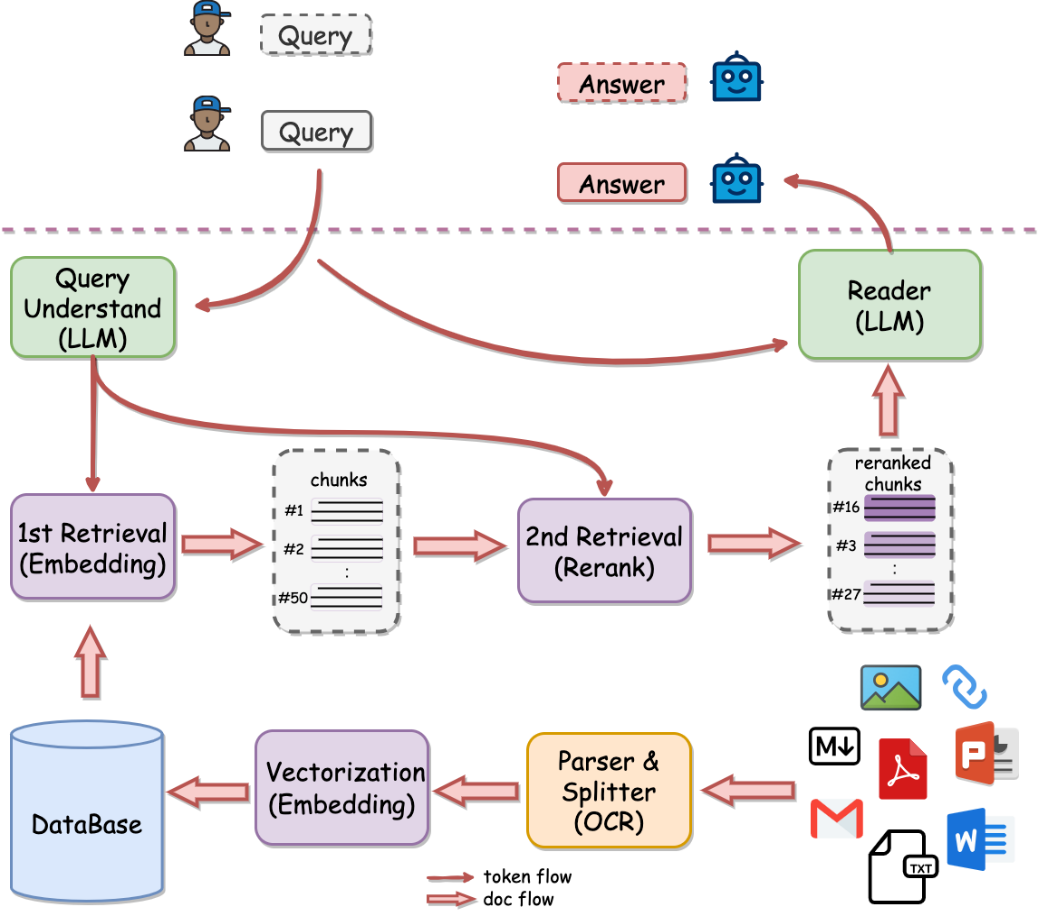


图2-1 QAnything系统架构图

### 2.1.1 数据接入层

数据接入层处于QAnything架构的最前端，承担着对数据采集、识别、提取和规范化的关键任务。由于面向知识问答系统的数据来源极为丰富，包括文本文档、图片、网页、表格甚至混合多模态内容，如何高效、准确地处理这些非结构化数据，是构建知识库的第一道门槛。

QAnything针对结构化程度较高的传统文档类型，构建了稳定、准确的提取机制，为支持中文语境下的处理，QAnything进行了编码自动识别（UTF-8/GBK）。当用户上传网页或提交URL时，系统通过爬取页面内容，将网页元信息（如标题、发布时间、作者）进行组合成完整的知识内容。针对PDF、文档截图等图片格式的数据，系统集成了OCR模型，支持中文、英文和中英混排文档的识别。

其设计理念体现了多格式支持与结构还原的统一。该层的表现直接影响到知识切分的粒度、向量检索的准确度以及问答内容的上下文丰富性，是构建高质量问答系统不可或缺的基础组件。

### 2.1.2 知识加工层

知识加工层承担着从“原始文本”向“可用知识”转化的关键角色。它的核心目标是对接入的数据进行语义理解、结构切分、内容清洗和增强，为构建高质量的语义索引与精准问答提供语义粒度更佳、上下文组织更合理的知识单元。

原始文本往往是连贯的长文档或复杂结构，需要被合理地切分成适合语义索引的小段落。QAnything的知识切分策略不仅考虑字符长度，还综合考虑语义完整性与段落结构。最基本的切分方法是固定长度切分，例如每500字左右为一个段落。该方法简单高效，适用于内容结构较松散、篇幅平均的场景，如新闻文章、百科文本等。

### 2.1.3 向量化与存储层

在知识加工层完成了结构化文本的提取与切分之后，系统进入了向量化与索引阶段。该阶段的核心目标是将每一个知识块转换为在语义空间中可用于相似度检索的高维向量表示，并通过建立高效的向量索引，支撑后续的大语言模型基于语义的上下文感知问答能力。

首先，系统需要选取适当的向量化模型对文本进行语义编码。QAnything默认采用适配中文语义检索任务的bge系列模型，该模型在大规模中文问答与文档匹配数据上训练，能够更好地理解句子之间的语义关系。为了增强向量的检索性能，系统在编码文本前会引入提示词模板，例如“为这个句子生成表示以用于检索：”等方式，对文本进行prompt增强，引导模型生成更具可区分性的语义向量。

考虑到文档结构信息对于语义表示的重要性，系统还设计了一系列语义增强策略。常见做法包括将当前段落的标题、上级章节标题拼接到段落前部，以引入更丰富的上下文线索。向量生成完成后，下一步是构建高效的向量索引结构。QAnything默认集成了FAISS（Facebook AI Similarity Search）向量检索库。每一个知识库都对应一个独立的向量索引文件，系统支持定时构建、增量更新与实时持久化，确保用户在添加、删除文档之后仍能保持向量检索结果的时效性与一致性。

### 2.1.4 问答引擎层

问答引擎层是QAnything框架中实现用户自然语言问题与知识内容之间的语义交互的核心模块。该层的设计主要围绕合理的Prompt工程、模型集成以及上下文关联这三个关键环节展开。

在问答引擎层的首要任务是构造高质量的提示词模板（Prompt），以引导语言模型在给定的知识上下文中生成符合预期的问题答案。QAnything针对不同任务类型（如摘要、解释、定义、比较、流程等）设计了多种Prompt模板，涵盖“基于以下内容回答问题”、“只回答与提供内容相关的信息”、“请使用完整句子输出结果”等指令性语句。这些模板不仅提供了模型生成时的行为边界，还可增强模型对于原始知识块的聚焦能力，减少幻觉现象的出现。

在模型选择方面，QAnything支持多种类型的大语言模型，包括本地部署的开源模型（如Ollama、Qwen等）和API调用类模型（如GPT 4.0、DeepSeek V3等）。本地模型适用于对隐私安全要求较高的私有场景，而API模型则因其强大的算力与响应速度具有显著优势。为此，系统在架构上预留了统一的模型调用接口，允许用户自由切换或组合模型类型。

为提升多轮问答体验，QAnything的问答引擎还具备上下文记忆能力。通过记录用户上一轮的提问与系统回答，并将其简要摘要作为本轮输入的补充提示，系统实现了“带记忆的对话模式”。

### 2.1.5 应用交互层

应用交互层是QAnything面向最终用户的接口承载层，主要负责将底层知识问答能力以图形化的方式呈现给用户，并支持灵活的参数调控能力。该层的设计不仅关系到用户体验，也直接影响知识问答系统的可用性与可推广性。

前端基于采用响应式Web界面，通过RESTful API与后端服务通信，支持自然语言提问、文件上传、对话历史管理等核心功能。交互层实现了知识库的灵活配置界面，允许用户可视化创建和管理不同领域的知识库，并采用SSE技术实现对话内容的实时流式返回，显著提升用户体验。针对跨语言与多区域使用场景，应用交互层还支持界面中英文切换，实现国际化推广。

综上所述，数据接入层支持多格式文档处理与OCR识别；知识加工层通过智能切分将原始数据转化为结构化知识；向量存储层利用bge模型和FAISS实现高效语义检索；问答引擎层集成多模型并设计专用Prompt模板实现精准回答；应用交互层提供可视化界面和实时对话功能。各层协同实现从数据到知识再到智能问答的全流程处理，组成了QAnything的架构体系，为此问答系统的架构提供了重要参考。

## 2.2 技术选型依据

为了实现系统在企业、学校内部场景下的高可用性、可维护性与良好用户体验，本系统从前后端分离架构的角度出发、基于本人在实习过程中的文档学习、使用体验，选用了Vue 3+TypeScript作为前端开发框架，UI组件库使用前端对Vue较为友好的Ant Design Vue框架，NestJS作为后端服务框架，以及MySQL作为系统核心数据存储方案。

### 2.2.1 Vue3+TypeScript的优势

在前端技术方案的选择上，经过充分的技术调研和评估，最终确定采用Vue 3与TypeScript相结合的技术栈。Vue 3作为当前主流的前端框架，在性能表现、响应式系统和组件化开发等方面较Vue 2有了显著提升，其特有的组合式（Composition）API设计使得复杂业务逻辑的组织更加清晰和可复用，这为项目中涉及的多功能交互场景提供了理想的技术支持，包括文件上传处理、智能问答交互、路由跳转等复杂功能模块的开发需求。

TypeScript的引入则为项目开发带来了类型安全的保障，其静态类型检查机制有效避免了常见的类型相关运行时错误，同时通过智能代码提示显著提升了开发效率。在涉及多模块协作和跨组件数据交互的复杂前端架构中，TypeScript的强类型特性展现出独特的优势。值得一提的是，随着Vue官方对TypeScript支持力度的不断加强，配合Vite等现代化构建工具的使用，整个项目的开发调试过程变得更加高效和可靠。

Ant Design Vue（简称antdv）是阿里开源的专为Vue开发的UI组件库，由Vue 3和TypeScript构建而成，为企业级应用开发提供高质量的组件实现。提供统一的视觉风格和交互体验，包含60+高质量组件，覆盖表单、数据展示、导航、反馈等常见场景。全面支持Vue 3 Composition API，与TypeScript深度集成，提供完整的类型定义。采用现代前端工具链，可无缝对接Vite、Webpack等构建工具。

### 2.2.2 NestJS的分层架构

后端采用NestJS作为开发框架，首先NestJS基于Node.js平台，采用TypeScript作为开发语言，其语法特性对前端工程师较为友好；其次NestJS采用了清晰的分层架构设计[10]，主要包括控制器层、业务服务层、数据持久层和功能模块四个核心组成部分，各层职责明确且相互独立。

控制器层作为请求入口，主要负责接收和解析HTTP请求。系统根据业务领域划分了多个控制器，包括处理用户认证的UserController、管理对话流程的ChatController、知识库的KnowledgeController以及负责操作Bot的BotController等。这些控制器仅负责请求路由和参数校验，具体业务处理则委托给下层服务。

业务服务层是整个业务逻辑的核心部分，封装了大多数处理逻辑，如调用数据库、判断用户权限、组合知识库内容、与大模型接口交互、记录对话内容等。在本系统中，例如ChatService中包含对大模型的调用流程、知识拼接逻辑和SSE的数据格式处理，而KnowledgeService则处理知识库的新增、编辑、删除等操作。

数据持久层主要负责与数据库的交互，通过调用Object-Relational Mapping（ORM，如TypeORM）[12]来进行实体数据的持久化管理。在NestJS中，可以直接通过@InjectRepository注解引入实体类对应的仓库，支持通过标准化的API完成数据存取操作，如查询检索、数据更新等基础操作，分别对应.find()、.save()、.update()等方法。本项目中各实体类如UserEntity、ChatRecordEntity、KnowledgeEntity等，均有独立的repository管理数据访问逻辑。

模块化设计在是NestJS的重要特性，每个模块封装相关的控制器、服务、实体等组件，形成内聚的功能集合。本项目划分了多个业务模块，如UserModule、BotModule、KnowledgeModule、ChatModule等，模块之间通过imports机制进行依赖注入，保证功能的独立性和模块间的解耦。NestJS分层架构图如图2-1所示。

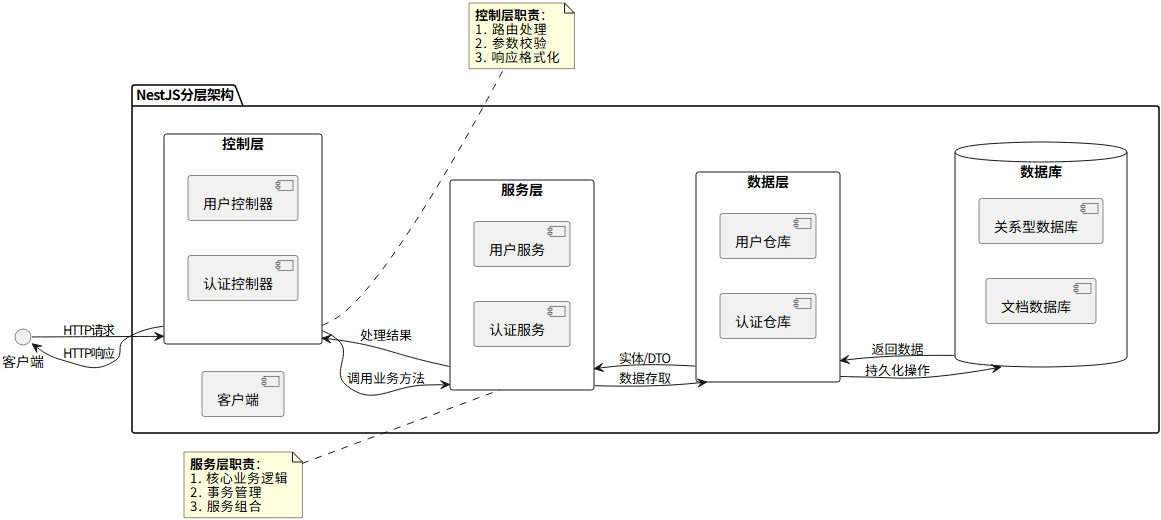


图2-1 NestJS分层架构图

### 2.2.3 MySQL关系模型设计

数据库方面，选择MySQL作为核心的关系型数据库系统，主要基于其在结构化数据管理上的成熟性与性能表现。系统需要处理的数据类型包括用户信息、上传文档元数据、问答历史记录、Bot信息等，这些数据具有较强的结构性与关系性，适合以关系型数据库进行管理。MySQL在读写性能、事务一致性、索引优化等方面均表现优异，且支持视图、存储过程、外键等特性，能够灵活支撑数据管理需求。

在技术整合方面，MySQL与NestJS框架中的TypeORM模块深度集成，实现了面向对象的数据操作模式。这种组合既保持了数据库操作的灵活性，又提高了开发效率和代码可维护性。前端采用Vue3+TypeScript与后端NestJS+MySQL的技术组合形成了良好的技术协同效应。前后端统一的TypeScript语言环境使得接口定义和数据结构的类型校验更加严谨，有效减少了因接口变更造成的问题。同时，采用前后端分离架构可实现独立部署，有利于后期系统迭代与性能调优。MySQL则作为稳定的数据支撑，保障了业务核心数据的高一致性与可维护性。

综上所述，该技术选型既考虑到了系统功能实现的需求，也兼顾了技术生态、开发效率与系统可扩展性。在实际开发与部署过程中，该技术栈表现出良好的稳定性与灵活性，能够满足系统在不同应用场景中的快速适配与高质量交付需求。

第3章 系统需求分析

## 3.1 功能性需求

为了实现用户对非结构化知识的快速访问与高效利用，本系统架构体系需要支撑一整套围绕“文件—知识—问答—Bot”的闭环应用流程。系统整体功能可划分为三大核心模块：快速开始问答、知识库问答与聊天Bot。在此基础上，进一步细化为若干关键子功能，包含但不仅限于创建知识库、文件上传、聊天关联、Bot创建、Bot配置与分享等，旨在为用户提供低门槛、高体验的一体化智能问答平台。

### 3.1.1 快速开始问答

首先，“快速开始问答”模块是专为两类典型用户场景设计的轻量化解决方案：一是面向初次接触系统的用户提供零门槛的入门体验；二是满足用户临时性、轻量级的文档问答需求。该功能的核心设计理念是“即传即问”的极简交互模式，用户无需提前配置模型参数，仅需上传一个或多个文档，即可迅速进入问答流程。每个用户可以在快速开始中创建多个对话，每个对话相对独立，一个对话对应关联一个知识库，此模式降低了使用门槛，尤其适用于临时性资料阅读、会议文档解读等场景。系统自动将上传文档临时存入一个知识库中，并绑定一个默认问答界面，实现即问即答的体验。

### 3.1.2 知识库问答

“知识库问答”是系统的核心功能模块，适用于持续性知识管理与长期问答需求。该模块支持用户创建任意数量的领域知识库，每个知识库具备独立的文档存储空间和问答配置体系，在文件兼容性方面，系统采用模块化解析引擎，可处理包括PDF、Word、PPT、Excel等办公文档，以及Markdown、纯文本、CSV结构化数据等共计十余种文件格式，并通过集成OCR[13]技术实现对PNG等图片格式的文字识别。在交互放慢，为了方便问答时的查阅，还支持了对各文件内容的预览。问答时，允许用户通过可视化界面自由组合多个知识库作为检索源，系统基于指定的知识库进行语义匹配与回答生成，确保问答结果既包含精确匹配又涵盖语义相关的内容。

不管是快速开始还是知识库问答模式，系统都为用户提供了完善的知识库管理能力。用户可以对知识库进行新增、重命名或删除等基础操作，同时在知识库内部支持文件的上传、删除及预览功能。这种设计确保了用户能够灵活地组织和维护自己的知识资源。

与“快速开始”模块相比最显著的区别在于：首先，知识库问答采用多知识库共享单一对话的机制，用户可以根据需求自由选择参与问答的知识库范围，既支持单选也允许多选，大大提升了知识检索的自由度和可控性。其次，在问答交互环节，系统开放了模型参数的配置权限，用户不再受限于预设的默认配置，可以根据实际需求调整本地模型或第三方API（如OpenAI）的各项参数。这种灵活的配置机制不仅满足了不同场景下的性能需求，也为专业用户提供了深度定制的可能性。

为持续保障问答质量，系统还提供了问答配置的自定义功能，包括但不限于调整上下文窗口大小、设置温度系数、输出token数等。通过这些细粒度的配置选项，用户能够构建出更符合特定领域需求的知识组织方案，从而获得更精准、更专业的问答体验。

### 3.1.3 聊天Bot

“聊天Bot”功能聚焦于问答能力的模块化封装与模型配置的可复用性设计，通过将知识库能力以Bot形式进行标准化输出，构建具备拟人化交互特征的AI助手体系。该功能采用多实例化架构设计，支持用户创建任意数量的Bot实例，每个实例可灵活绑定一个或多个知识库作为语义理解基础。在个性化配置方面，系统提供了多维度的Bot属性定制功能，每个Bot可单独配置头像、名称、角色设定、系统提示词等信息，以形成具有特定语气和角色定位的对话体验。

通过这种高度可配置的Bot生成机制，用户能够快速构建面向垂直领域的专业化智能助手，典型应用场景包括但不限于：心理咨询场景的心理辅导Bot、技术开发场景的编程助手Bot、情感社交场景的恋爱顾问Bot等，满足不同场景下的智能化交互需求。

第4章 系统设计

## 4.1 总体架构设计

本系统在架构设计上参考了QAnything所提出的模块化问答系统思想，但结合毕设开发周期短、功能需求相对聚焦的实际情况，进行了针对性的裁剪与优化。系统整体采用分层解耦的设计思想，将架构划分为：数据接入层、数据存储层、知识加工层、问答引擎层与应用交互层四大部分。每一层职责清晰，协同工作，构建起一个简洁高效、易于部署的智能问答平台。系统架构图如图4-1所示。

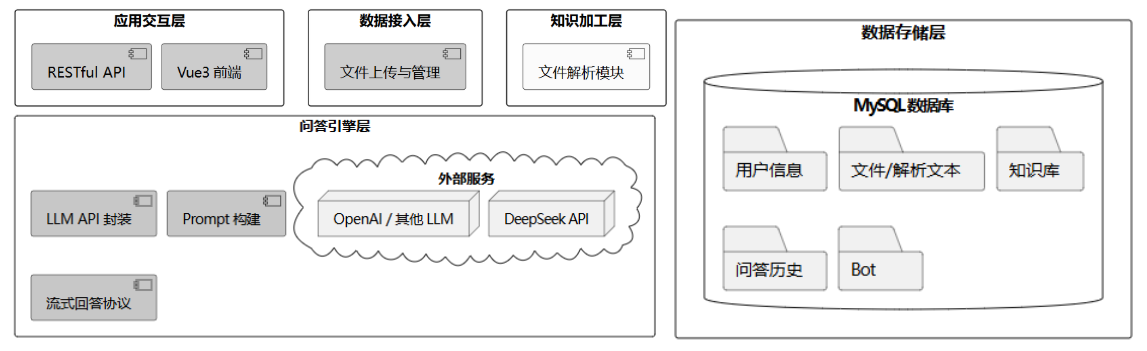
****

图4-1 系统架构图

### 4.1.1 系统架构分析

本系统采用了分层解耦的架构设计思想，将整体功能模块划分为五大核心层次：应用交互层、数据接入层、知识加工层、问答引擎层以及数据存储层，并通过对接外部大模型服务，如DeepSeek和OpenAI API服务，实现了高效、稳定的问答能力。该架构不仅保障了系统的清晰性与可维护性，也为后续功能扩展和模块替换提供了良好的支撑。

#### （1）应用交互层

应用交互层是用户与系统交互的直接入口，包含Vue3+TypeScript构建的Web前端以及采用NestJS框架开发的后端API接口服务。前端提供了用户登录、Bot创建、文件上传、知识库管理、问答对话等一系列功能页面，通过统一的UI设计提升了使用体验。后端则承担了身份鉴权、请求路由、数据分发等职责，是系统运行的中枢神经。

用户的所有行为操作都通过RESTful接口发送给后端服务，后端根据业务逻辑进一步调用数据处理模块或问答引擎模块，并将最终的处理结果返回给前端展现。前后端解耦的模式提升了开发效率与协作灵活性，便于并行迭代和独立部署。

#### （2）数据接入层

数据接入层主要负责用户上传原始数据文件的接收与初步处理，支持包括txt、md、pdf等常见文档格式。文件上传模块在用户上传文件后，先对其进行格式解析和内容提取，再统一命名和存储到系统目录，并在数据库中登记相应的元信息，如文件名、用户ID、知识库归属等。

该层将文件上传逻辑与后续知识处理过程隔离，确保了系统对数据的清晰分层与可追溯性。模块设计上预留了多种文件格式的处理接口，便于未来扩展至图像、音频等多模态数据。

#### （3）知识加工层

知识加工层承担了对原始文件的深入理解与组织工作，是系统“喂知识”的核心步骤。目前的设计中，该层通过文件解析模块提取纯文本内容，并进行一定程度的清洗与分段。不同于传统RAG系统使用向量化索引进行知识召回的方式，本系统将解析后的文本作为系统级提示信息（system prompt）直接传递给大语言模型，在问答过程中引导模型给出基于知识上下文的回答。

该方式降低了系统对嵌入库和向量数据库的依赖，减少了基础设施搭建的复杂性，适合轻量化部署场景。与此同时，解析结果也会被存储进数据库，支持知识内容的可视化管理与手动维护。

#### （4）问答引擎层

问答引擎层是整个系统的智能核心，负责构建模型调用所需的上下文信息，并封装对大语言模型的API调用流程。其中，Prompt构建模块根据用户输入问题、Bot配置信息、知识库内容等构造出符合预期的Prompt结构，融合知识背景提升问答准确性。

LLM API封装模块对接包括DeepSeek API、OpenAI等多个大型语言模型服务，封装了统一的请求结构与响应解析逻辑，便于切换模型提供商或实现多模型并行选择。对于用户而言，该部分的切换是透明的，只需关注最终问答效果。流式回答协议模块实现了模型响应的分段推送机制，通过SSE技术将模型的中间输出实时反馈给前端用户，提高了交互响应速度和体验流畅性。

#### （5）数据存储层

系统使用关系型数据库MySQL进行数据存储。数据库表结构包括用户信息、知识库结构、上传文件记录、解析文本内容、问答历史记录以及Bot配置信息等多个维度，实现了用户数据的持久化与管理。所有模块间的数据交互最终都会落地到数据库中，支撑系统的持续运行与审计需求。

数据库结构设计注重规范性与可扩展性，如问答历史表中记录每轮问答的时间戳、上下文内容、所属Bot与知识库等，便于后续功能如问答检索、用户分析等开发。

总体而言，本系统采用了清晰的分层结构与现代化的前后端技术栈，满足了从数据接入到智能问答的全流程需求。相较于传统QA系统，本系统更加轻量化，依赖少、部署灵活，特别适合在企业内部、教学实验、知识助手等中小规模场景落地应用。未来可在当前架构基础上进一步扩展向量化模块、知识图谱支撑、多模态输入等能力，以增强系统的鲁棒性与智能性。

### 4.1.2 数据流设计

用户首先在系统的Web前端界面中输入问题，前端将用户输入的内容以及所选上下文信息封装为标准POST请求，并发送至后端服务。

后端接收到请求后，会首先进行用户身份校验和权限验证，确保当前用户有权限访问相关知识库。随后，后端会根据所选择的知识库ID，从数据库中检索出该知识库中解析后的全部文本内容，并将这些内容作为上下文信息拼接进Prompt。

构建好完整的Prompt后，后端将“用户问题+知识库内容”一并发送至大模型。由于系统没有引入向量化检索，因此全部知识内容会被直接插入到Prompt的System角色段中，以使大模型获得尽可能全面的参考背景。

大模型处理完请求后，会生成针对用户问题的自然语言回答，并将答案返回给后端。为了提升响应速度和用户体验，后端采用SSE（Server-Sent Events）协议将答案逐段流式地推送给前端，实现内容的实时渲染展示。用户可以边看边读，避免长时间等待完整返回结果。

同时，系统也会将本次问答记录同步写入数据库，包括用户提问内容、模型生成的回答、所使用的知识库ID、时间戳、用户标识以及Bot ID等信息。这一部分数据为用户历史记录功能提供支撑，也便于后期管理端进行cr问答质量分析与内容审计。

前端接收到后端流式返回的文本后，会持续监听SSE事件流并动态拼接完整答案，最终将渲染好的结果展示在对话框中。整个过程中，用户感知到的是一个近似于ChatGPT式的流畅问答体验。数据交互图如图4-2所示。

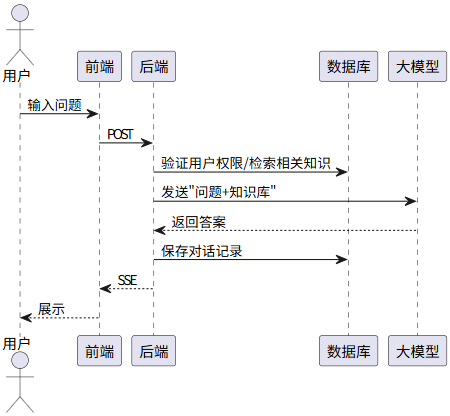


图4-2 数据交互图

## 4.2 数据库设计

系统的数据库采用MySQL进行设计与实现，整体结构围绕用户、知识库、文件管理、聊天记录与Bot配置五大核心功能展开，保证数据存储的完整性与查询的高效性。

### 4.2.1 数据库实体关系设计

ER图展示了系统中六个核心实体之间的关系，如图4-3所示。包括用户User、知识库Knowledge、文件File、聊天记录ChatRecord、聊天机器人Bot。每个实体通过主键唯一标识自身，通过外键与其他实体形成逻辑关联，支持系统的权限控制、知识查询、对话存储等功能。

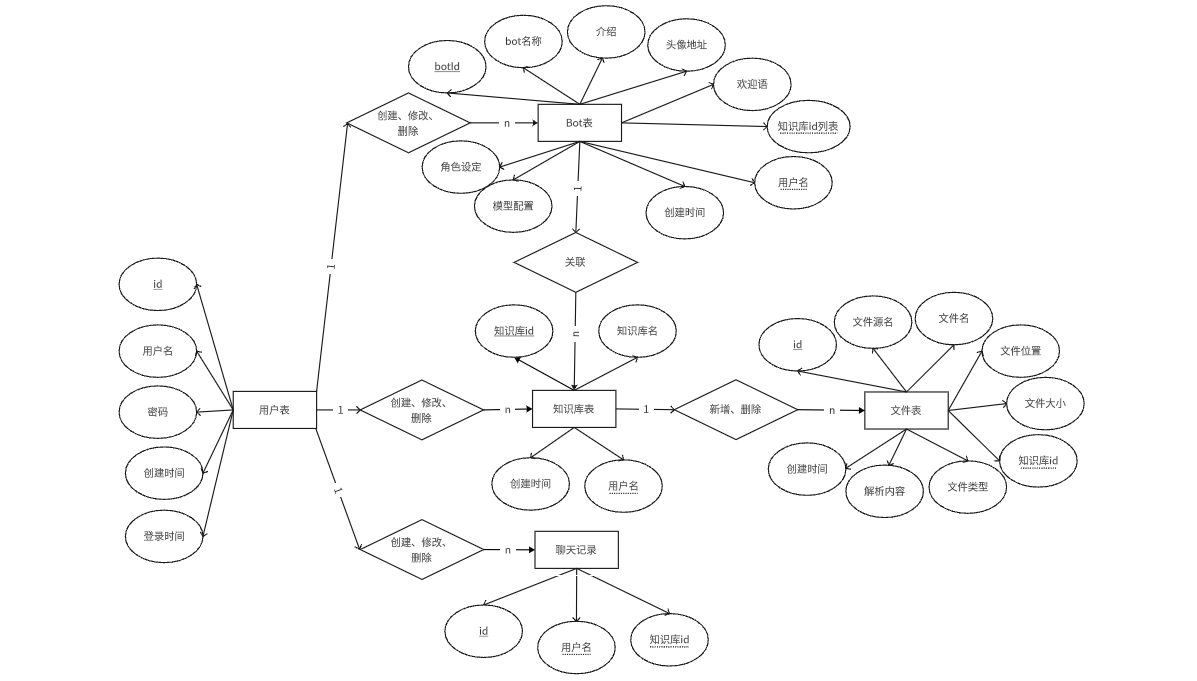


图4-3 ER图

用户User是系统的核心，每个用户通过唯一的username与多个知识库Knowledge、聊天机器人Bot和聊天记录ChatRecord关联。用户的id为自增主键，username是唯一字段，用于建立所有外键关系。

知识库Knowledge用于组织结构化或非结构化的知识信息，由用户创建。每个知识库通过kbId唯一标识，并通过username与User表建立外键联系，说明一个用户可以拥有多个知识库。

文件File属于知识库，是知识内容的具体载体。每个文件通过kbId关联到所属的知识库，支持多对一的关系，即一个知识库可以包含多个文件。文件信息包括原始文件名、实际存储名、路径、大小、类型、内容等字段。

聊天记录ChatRecord记录了用户与机器人或系统的对话历史。每条记录通过username和kbId与User和Knowledge建立外键关联。ChatRecord的messages字段以JSON字符串形式存储完整的多轮对话内容，type字段用于区分不同对话模式，如“quick”或“home”。

聊天机器人Bot由用户创建，并绑定一个或多个知识库以支持问答任务。机器人通过username与用户关联，字段kbIds是字符串数组，存储多个知识库ID，实现一个机器人对多个知识库的访问能力。机器人还包含头像、欢迎语、角色设定、聊天配置等属性，以支持自定义化的人机交互体验。

整个ER图反映出以用户为中心的设计理念，用户创建知识库、上传文件、构建机器人并产生对话记录，各模块之间通过清晰的外键关系串联，确保数据的完整性和操作的权限边界。

### 4.2.2数据库逻辑结构设计

#### 用户表

首先是用户表，用于存储系统中的用户信息。每位用户拥有唯一的自增主键id，其身份凭证由username和password字段记录。系统支持基于角色的权限控制机制，通过role字段区分不同的用户类型，默认为“user”。此外，该表还包含createTime与updateTime字段，用于记录用户的注册和最近一次信息变更时间，方便管理员管理与审计。

表4-1用户表

| 字段名 | 类型 | 说明 |
| --- | --- | --- |
| id | INT | 主键，自增 |
| username | VARCHAR | 用户名 |
| password | VARCHAR | 用户密码 |
| role | VARCHAR | 用户角色 |
| createTime | VARCHAR | 创建时间 |
| updateTime | VARCHAR | 更新时间 |

#### 知识库表

知识库表用于管理知识库的基本信息。表中的主键为kbId，采用字符串类型以支持灵活生成。知识库名称保存在kbName字段，type字段用于标识知识库类型，例如“quick”表示快速问答知识库，“normal”表示标准知识库。每条知识库数据通过username字段关联到具体的用户，体现出知识库对用户的归属关系。这种设计支持用户之间的知识库隔离，增强数据安全性。

表4-2知识库表

| 字段名 | 类型 | 说明 |
| --- | --- | --- |
| kbId | VARCHAR | 知识库主键 |
| kbName | VARCHAR | 知识库名称 |
| type | VARCHAR | 类型：“quick”/“normal” |
| createTime | VARCHAR | 创建时间 |
| username | VARCHAR | 关联用户（外键） |

#### 文件表

文件表用于管理用户上传的原始文件与解析后内容。每个文件具有唯一的自增主键id，文件原始名称与实际存储名称分别记录在originalName和fileName字段中，便于前后端映射与展示。文件的物理存储路径由filePath字段标识，fileSize与mimeType字段则提供了文件大小与类型信息。在文件内容处理上，系统将解析后的纯文本内容保存在content字段，并以kbId建立与知识库的外键关联。该表还记录了上传时间，用于回溯与管理。

表4-3文件表

| 字段名 | 类型 | 说明 |
| --- | --- | --- |
| id | INT | 文件主键（自增） |
| originalName | VARCHAR | 上传时的原始名称 |
| fileName | VARCHAR | 实际保存的文件名 |
| filePath | VARCHAR | 文件存储路径 |
| fileSize | INT | 文件大小（字节） |
| mimeType | VARCHAR | MIME类型 |
| content | LONGTEXT | 文件内容 |
| kbId | VARCHAR | 外键，关联知识库 |
| createTime | VARCHAR | 上传时间 |

#### 历史对话表

历史对话表用于记录用户的所有对话历史。每条记录具有唯一的自增主键id，通过username关联用户，kbId用于关联知识库，支持对话与知识上下文的联动。type字段表明聊天记录的类型，例如“quick”表示快速问答模式，“home”表示主页普通对话模式。系统将每次对话以JSON数组格式序列化后存储在messages字段中，便于后续渲染与分析。这样的结构设计既能保持对话完整性，也便于实现多轮会话的回溯与管理。

表4-4历史对话表

| 字段名 | 类型 | 说明 |
| --- | --- | --- |
| id | INT | 自增主键 |
| username | VARCHAR | 外键，关联 User.username |
| kbId | VARCHAR | 外键，关联Knowledge.kbId，可为空 |
| type | VARCHAR | 枚举：“quick”或“home” |
| messages | TEXT | JSON 字符串（序列化的数组） |

#### 聊天Bots表

Bots表为系统中的智能机器人配置表。每个Bot拥有一个唯一标识botId和展示名称botName，用户可以配置机器人的头像avatar、简介description、欢迎语welcomeMessage、角色设定roleSetting等参数。通过username字段将Bot与用户绑定，确保每个用户只能管理自己的Bot。一个Bot可以关联多个知识库，系统通过kbIds字段使用字符串数组形式存储多个知识库ID，从而支持复杂场景下的知识融合问答。此外，chatSetting字段以JSON格式存储Bot的个性化聊天行为参数，如温度值、最大响应长度等，以支持灵活的问答调优。

表4-5 Bots表

| 字段名 | 类型 | 说明 |
| --- | --- | --- |
| id | INT | 主键，自增 |
| botId | VARCHAR | 机器人唯一标识 |
| botName | VARCHAR | 名称 |
| description | VARCHAR | 简介 |
| avatar | VARCHAR | 头像 |
| username | VARCHAR | 外键，关联users.username |
| kbIds | SIMPLE-ARRAY | 知识库ID数组（字符串数组） |
| welcomeMessage | VARCHAR | 欢迎语 |
| roleSetting | VARCHAR | 角色设置 |
| chatSetting | JSON | 聊天配置（结构体） |
| createTime | VARCHAR | 创建时间 |

整个数据库逻辑结构在功能分工上高度清晰，不同表之间通过外键保持良好的数据耦合性，有效支撑起用户管理、知识库构建、问答服务和Bot应用等核心业务流程。同时，通过适当的数据冗余（如记录用户名而非用户ID），也使得前端数据加载与权限校验更加方便，提升了系统整体的扩展性与可维护性。

第5章 系统实现

## 5.1 前端实现

前端使用主流框架Vue3作为核心开发框架，结合TypeScript语言特性构建响应式用户界面，主要负责实现系统与用户的交互功能。在UI组件层面，选用阿里巴巴开源的Ant Design Vue（简称antdv）作为基础UI库，该组件库提供丰富的企业级UI组件和设计规范。系统整体布局采用antdv推荐的顶部导航栏配合左侧边栏的设计模式，既保证了导航的直观性，又提供了充足的功能展示空间。从功能架构角度，系统前端主要划分为三大核心模块：快速开始问答、知识库问答和聊天Bot管理。本文将按照这三个主要功能模块展开详细说明，并深入分析每个功能点实现过程中所采用的关键前端技术。

### 5.1.1 快速开始

“快速开始”功能的设计理念是让用户能够零门槛体验问答服务。用户完成注册后即可直接开始对话，无需进行复杂的模型配置或文件上传操作，其功能定位类似于简化版的DeepSeek，底层实现即调用了DeepSeek的API，模型为DeepSeek V3。同时，系统也提供了更丰富的功能选项，用户可以通过左侧导航栏进入管理页面，在任意对话中上传文件。系统采用对话隔离机制，确保不同对话间的文件互不干扰，用户可以自由地在各个对话中上传和管理文件。快速开始界面如图5-1所示。

图5-1快速开始界面

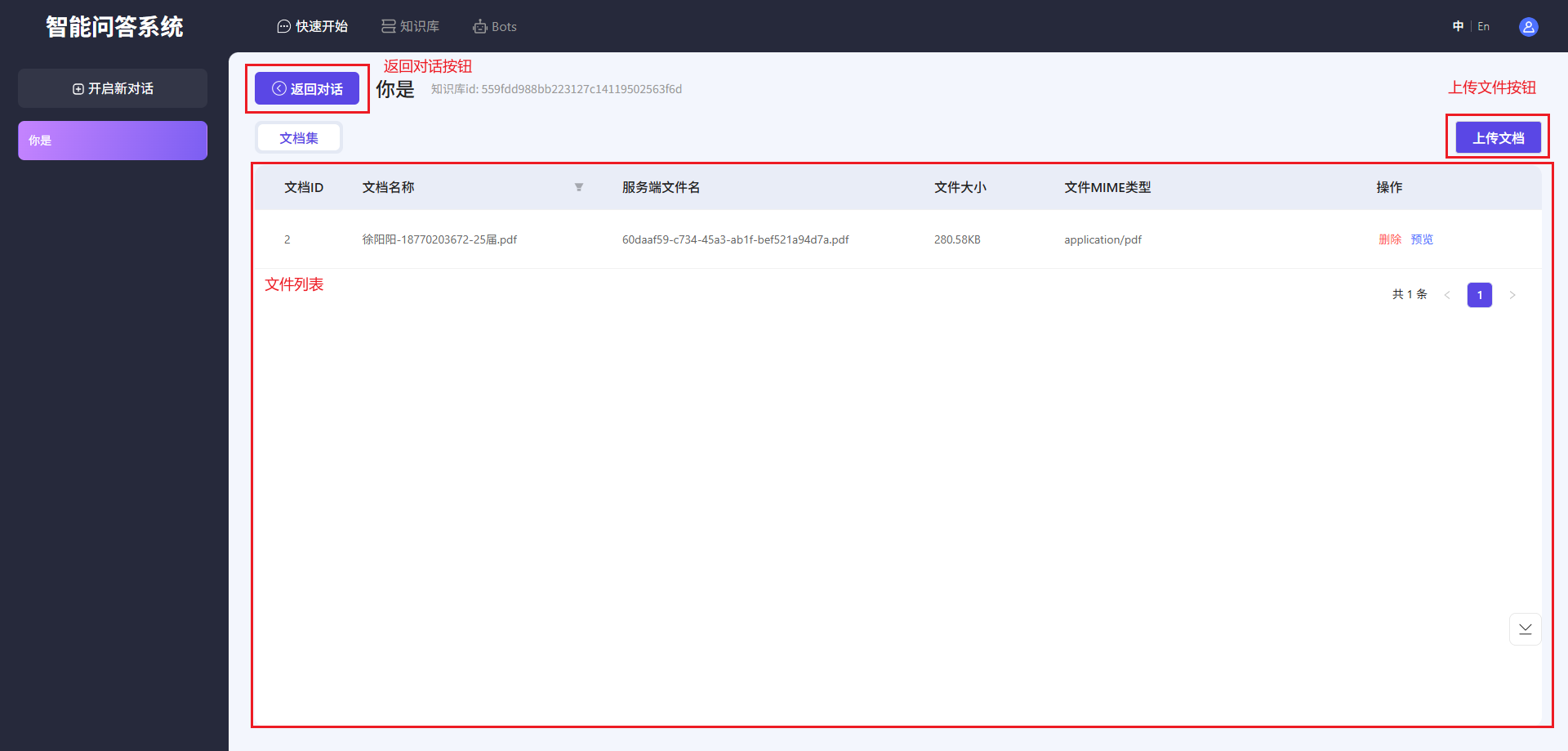
在文件管理页面中，用户可通过左上角的返回按钮快速切换回对话界面。页面顶部清晰展示了当前对话的标题和关联知识库ID等元信息。右侧区域提供文件上传功能，系统对上传文件设置了大小限制：普通文档不超过30MB，图片文件不超过5MB。页面下方以表格形式列出该对话关联的所有文件，包含文档ID、原始文件名、服务器存储名、文件大小和MIME类型等详细信息。表格最后一列提供操作按钮，支持文件的删除和预览功能。文件管理界面如图5-2所示。

图5-2文件管理界面

以上展示了快速开始功能的整体界面设计，接下来将从代码实现层面深入解析各核心功能模块。

#### 开启新对话

点击左侧开启新对话，即创建一个知识库，且开启一个全新对话，CRUD就不多介绍了

#### 问答交互

用于实现大语言模型的逐字输出模拟，优化用户在对话过程中的实时交互体验。该方案采用SSE，结合前端动态渲染技术，模拟“打字机”效果，用户可以在回答后即可看到回复，有效减少用户等待焦虑，提升交互流畅度。

SSE基于HTTP长连接，服务端可主动向客户端推送事件流（text/event-stream）。前端通过fetchEventSource发起请求，通过AbortController（signal:ctrl.signal）实现请求中断，利用typewriter对象动态追加文本实现逐字输出的"打字机"效果。

核心实现如下：

fetchEventSource(apiBase + '/chat/stream', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

Accept: 'text/event-stream, application/json',

},

body: JSON.stringify({

username,

...sendData,

}),

signal: ctrl.signal,

async onopen(e: any) {

addAnswer(q);

if (e.ok && e.headers.get('content-type') === 'text/event-stream') {

typewriter.start();

} else if (e.headers.get('content-type') === 'application/json') {

typewriter.add('Error 请检查模型是否配置正确');

}

},

onmessage(msg: IMessage) {

console.log('message', msg);

if (msg.data !== '[DONE]') {

const res: any = JSON.parse(msg.data);

if (res.content) {

typewriter.add(res.content);

scrollBottom();

}

}

},

onclose() {

typewriter.done();

ctrl.abort();

},

onerror(err: any) {

typewriter.done();

ctrl.abort();

throw err;

},

});

向目标服务器发送POST请求，设置请求头：Content-Type:application/json声明发送 JSON数据Accept:text/event-stream表明客户端支持SSE，请求体携带用户信息和一些模型配置用AbortController的信号允许后续中断请求。连接建立(onopen)，首先调用addAnswer(q)在前端界面添加问题展示。检查响应，如果是SSE流(text/event-stream)，启动打字机动画，否则直接显示错误提示。消息处理(onmessage)忽略结束标记[DONE]，解析每条消息的JSON数据：当有内容(res.content)时，通过打字机逐字显示(typewriter.add())。连接关闭(onclose)

结束打字机动画、中断请求。错误处理(onerror)同样完成动画并中断请求，抛出错误供上层处理。

#### 将会话保存为图片

获取到我们的聊天dom，用html2canvas这个三方库即可将我们的dom使用canvas画出来，然后往dom树中丢一个带连接的a标签，在模拟一下点击，这一翻操作即可触发浏览器的下载。

核心代码如下：

const ele = document.getElementById('chat-ul');

const canvas = await html2canvas(ele as HTMLDivElement, {

useCORS: true,

});

const imgUrl = canvas.toDataURL('image/png');

const tempLink = document.createElement('a');

tempLink.style.display = 'none';

tempLink.href = imgUrl;

tempLink.setAttribute('download', 'chat-shot.png');

if (typeof tempLink.download === 'undefined') tempLink.setAttribute('target', '\_blank');

document.body.appendChild(tempLink);

tempLink.click();

document.body.removeChild(tempLink);

window.URL.revokeObjectURL(imgUrl);

message.success('下载成功');

Promise.resolve();

#### 上传文件

文件上传在前端的工作量并不大，只需要使用type为file的input框，将文件，就可以调起系统文件选择器，然后在请求接口的时候，设置请求体为formData格式即可，具体的后端实现我们会在后面进行介绍。

<input

class="hide input"

type="file"

accept="acceptList.join(',')"

multiple

@change="upload"

@click="(e) => ((e.target as HTMLInputElement).value = '')"

/>

const formData = new FormData();

formData.append('file', file);

#### 文件预览

在对文件进行操作时，我们可以选择预览，这里就涉及两种预览方式，第一种方案是采用弹窗预览，第二种是跳转至新页面预览，因为这里我希望用户能够边看文件内容边提问，提供更便捷的查看方式，不用频繁地打开弹窗，于是采取了新开窗口预览的方式。点击预览会另打开一个窗口，然后在前端我们使用iframe标签，将src属性直接访问服务器对应的静态资源，即可实现大部分文本文件以及pdf文件的预览。

这里就存在一个问题，iframe仅支持部分文件[9]，像docx、pptx、xlsx这些类型的文件都不支持，所以我们必须采取其他办法解决，我这里使用的办法是通过条件编译，通过文件名后缀判断文件类型，如果不属于docx、pptx、xlsx这三者之一，则使用iframe进行渲染，否则使用其对应的预览库，在npm包管理平台中，找到了这三者对应的vue项目中可适用的包。分别是“@vue-office/docx”、“@vue-office/ pptx”和“@vue-office/excel”，然后借助vue的自定义组件实现自动识别文件并用对应组件进行渲染。

### 5.1.2 知识库问答

知识库问答模块相比于快速开始，功能上没有很大的差异，但是提升了的一些配置项。整个知识库问答共用一个对话，可以新增多个知识库，用户可以手动选择提问的知识库（背景紫色代表选中），支持选中多个知识库，针对模块化的数据又增加一定耦合。我们只需要输入知识库的名字，点击“新建”即可新增知识库，支持重命名。在问答操作栏，新增了“清空会话”与“模型配置”选项，用户可以根据自己的模型，拓展配置，使模型回答更符合需求。知识库问答界面如图5-3所示。

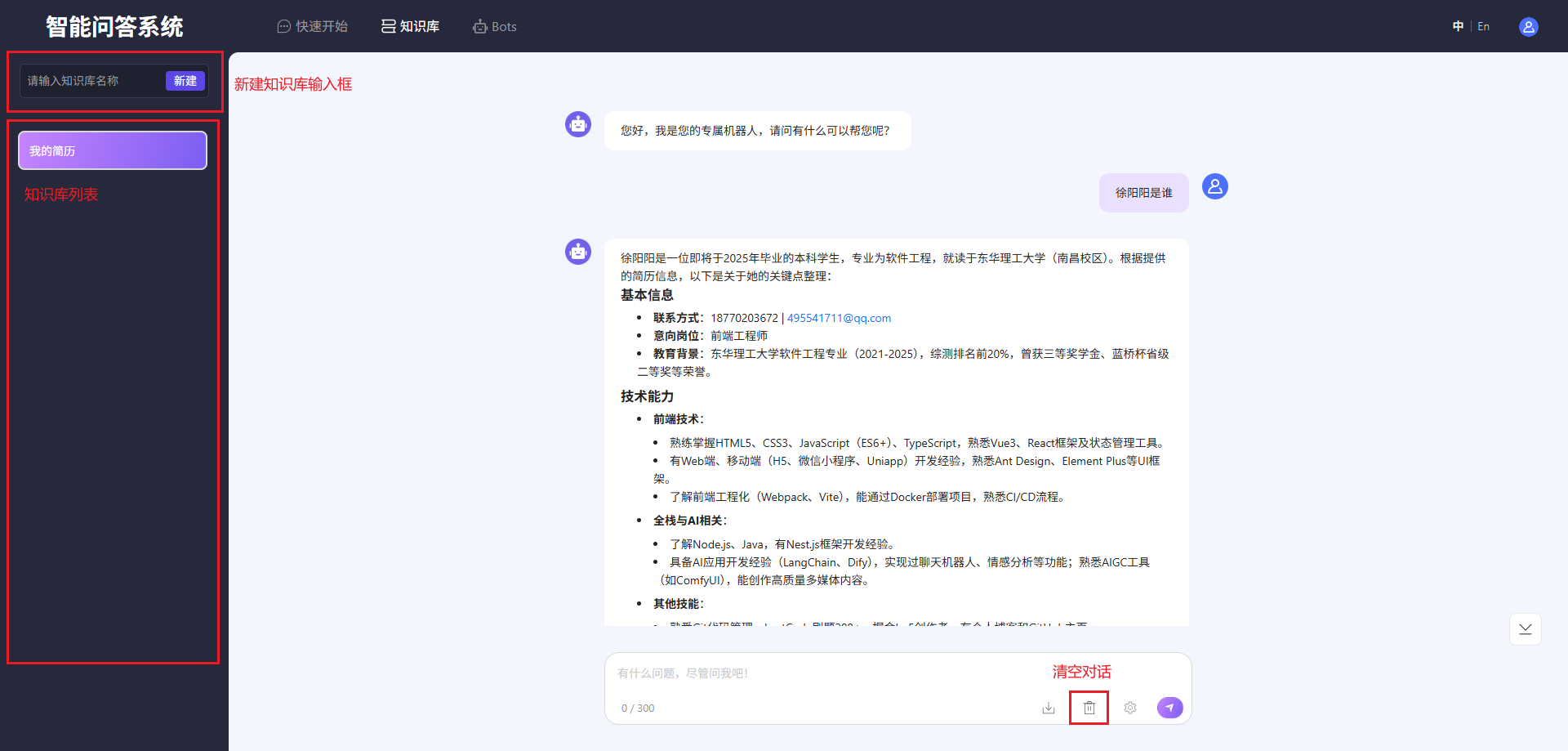


图5-3知识库问答界面

### 5.1.3 Bots

Bots，可以理解为在知识库问答的基础上又增加了一些配置。首先bot列表页，我们预设了一些聊天bot，针对不同的bot配置的不同的欢迎语和提示词。同时也支持用户自定义自己的bot，点击新增，输入bot名称、上传头像、介绍一下bot的功能，即可创建一个bot。Bots列表界面如图5-4所示。

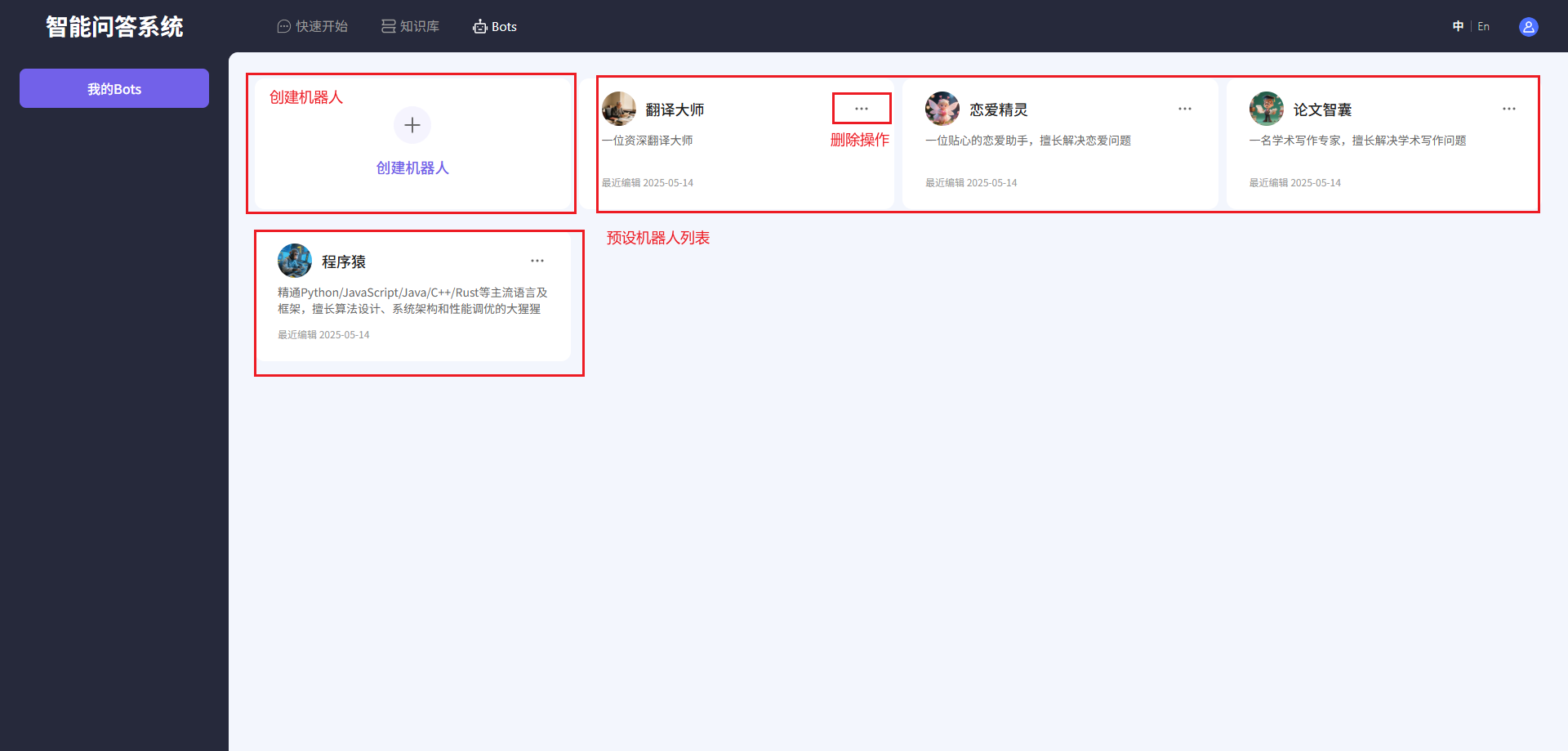


图5-4 Bots列表页

点击对应bot会跳到对应Bot详情页，左侧是bot的基本信息，右侧是bot对话栏，这里可以对bot配置信息修改后进行测试。

关于bot的配置项，从bot的基本信息：头像、名称、欢迎语，再到与模型交互所需的系统角色设定、模型配置，甚至还能够管理我们创建的知识库，提供给用户一个自由度极高的bot创建平台。

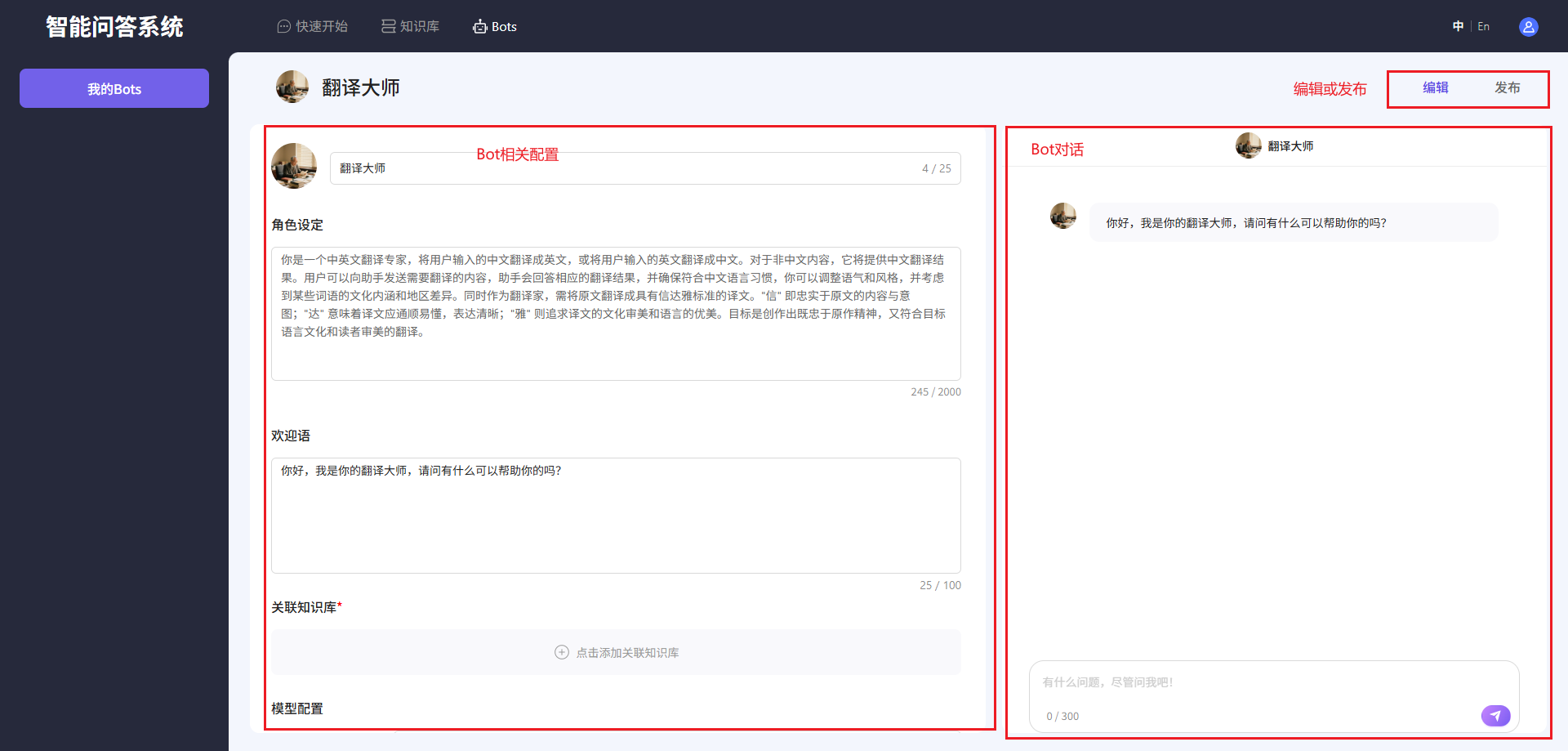


图5-5 Bots配置页

Bots中还有一个极为重要的功能，创建的bot支持发布并分享，所有用户都可以通过点击链接或者扫描二维码实现bot的预览与对话，分享出来的bot所使用的基本信息与模型配置为最后一次保存的结果。Bots发布页与分享对话页如图5-6所示。

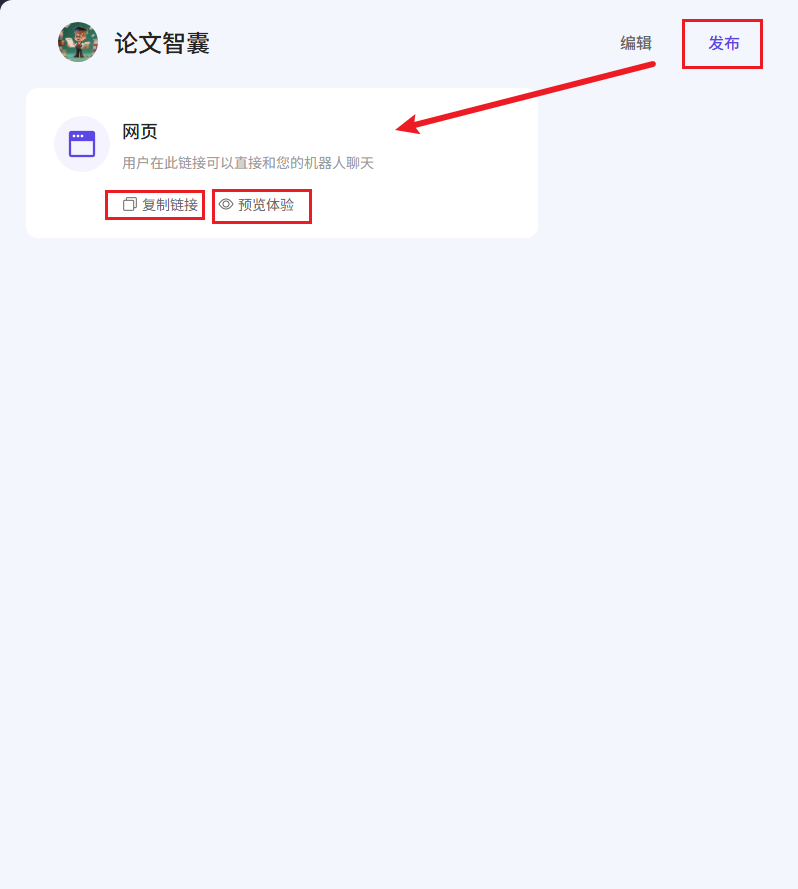


图5-6 Bots发布页与分享对话页

## 5.2 后端实现

后端基于主流框架Nestjs实现，它是一个使用TypeScript的渐进式Node.js框架，它采用了清晰的分层架构设计，非常适合构建高效、可扩展的服务器端应用程序。

### 5.2.1 分层架构的实现

在技术综述时已经介绍了NestJS的分层架构，通过控制器层、业务逻辑层、数据访问层的明确分离，结合模块化设计，构建了高内聚低耦合的系统架构在。本系统在分层架构的加持下，代码质量和开发效率得到了飞速提升。代码的目录结构如下：

src/

├── modules/ # 所有功能模块统一管理

│ ├── users/ # 用户模块

│ │ ├── dto/ # 数据传输对象

│ │ │ ├── users.dto.ts

│ │ ├── entities/ # 实体类

│ │ │ └── user.entity.ts

│ │ ├── users.controller.ts

│ │ ├── users.service.ts

│ │ └── users.module.ts

│ │

│ ├── knowledges/ # 知识库模块（同user结构）

│ ├── chats/ # 聊天模块

│ ├── files/ # 文件模块

│ └── bots/ # Bot模块

├── common/ # 公共资源

│ ├── utils.ts/ # 工具函数

│ ├── fileParser.ts/ # 文件解析相关

├── config/ # 配置文件

│ └── multer.config.ts

└── main.ts # 应用入口

下面以用户管理模块为例，详细阐述了基于NestJS框架的分层架构实现方案。

#### 表现层实现

用户控制器通过构造函数注入服务实例，每个公共方法对应一个API端点。如用户注册接口实现：

@Post('register')

@UsePipes(new ValidationPipe())

async register(@Body() dto: CreateUserDto) {

return this.userService.register(dto);

}

使用@Body()自动解析请求体，通过ValidationPipe进行参数校验，业务逻辑完全委托给服务层

#### 业务逻辑层实现

用户服务类包含核心业务规则，典型实现如下：

async register(dto: CreateUserDto) {

// 业务规则验证

const exists = await this.userRepo.exist({

where: { username: dto.username }

});

if (exists) throw new ConflictException('用户已存在');

// 业务操作

const user = this.userRepo.create(dto);

await this.userRepo.save(user);

// 生成业务结果

return {

token: this.jwtService.sign({ sub: user.id }),

userInfo: \_.omit(user, ['password'])

};

}

#### 数据访问层实现

通过TypeORM的Repository模式提供数据持久化能力：

@EntityRepository(User)

export class UserRepository extends Repository<User> {

async findActiveUsers(): Promise<User[]> {

return this.find({ where: { isActive: true } });

}

}

总而言之，通过模块化组织实现功能内聚，依赖注入机制降低耦合度，清晰的层级边界提升可维护性，完善的异常处理保障系统健壮性。

### 5.2.2 模型对话服务

模型对话服务采用流式响应设计，基于Server-Sent Events（SSE）技术构建实时交互能力。服务通过灵活的配置架构同时支持原生API调用和Bot代理模式，当请求携带botId参数时会自动加载预设的对话配置，包括API端点、模型参数、知识库关联和角色设定等。核心处理流程首先构建包含系统提示、知识库内容和历史消息的完整对话上下文，其中知识库内容通过文件服务动态检索注入系统消息。

服务采用Axios的流式请求方式与底层模型API交互，实现逐块处理响应数据并实时转发给客户端，过程中严格遵循SSE协议规范，每个数据块都包含标准格式的事件流消息。错误处理机制覆盖了从API请求到流式传输的全链路，确保任何环节出现异常都能向客户端发送结构化的错误信息并优雅关闭连接。该实现特别优化了大规模上下文和历史消息的处理效率，通过可配置的上下文窗口大小控制内存消耗，同时保持对话连贯性。

核心代码如下：

axiosResponse.data.on('data', (chunk: Buffer) => {

const lines = chunk

.toString()

.split('\n')

.filter((line) => line.trim() !== '');

for (const line of lines) {

if (line.includes('[DONE]')) {

response.write('data: [DONE]\n\n');

return;

}

if (line.startsWith('data: ')) {

try {

const jsonData = JSON.parse(line.slice(6));

const content = jsonData.choices?.[0]?.delta?.content || '';

if (content) {

response.write(`data: ${JSON.stringify({ content })}\n\n`);

}

} catch (e) {

response.write(

`data: ${JSON.stringify({

error: '解析响应数据出错',

details: e.message,

})}\n\n`,

);

}

}

}

});

服务设计充分考虑了扩展性，模型参数、能力开关和流式控制等均可通过请求参数动态调整，为不同类型的对话应用场景提供了统一的接入接口。这种流式对话服务架构既满足了实时交互的低延迟需求，又通过完善的错误处理和资源管理保障了系统稳定性，可作为智能对话系统的核心组件服务于各类应用场景。

### 5.2.3 文件解析服务

基于Node.js的多格式文件解析服务实现方案。该服务支持PDF、Word、Excel、PPT、文本文件及图片等多种格式的内容提取，通过模块化设计和多种解析库的组合应用，提供了统一的文件内容提取接口。

在PDF解析方面采用pdf-parse库实现文本提取，Word文档处理使用mammoth库保留基础格式信息，Excel文件通过xlsx库转换为CSV格式输出，PPT内容借助officeparser库提取，而图片OCR识别则基于Tesseract.js引擎实现中英文字符识别[11]。

服务采用标准的Promise异步接口设计，内部通过文件扩展名自动路由到对应解析器，并实现了统一的错误处理机制，将底层各类解析异常转换为服务级错误信息。该服务具有良好的可扩展性，新增文件类型支持只需添加对应的解析函数和路由分支即可，各解析器之间的实现相互独立使得组件替换和升级变得简单。

在实现细节上，针对不同文件类型特点采用了差异化的处理策略，如图片解析时特别注意了OCR工作线程的创建和释放，确保资源及时回收；同时兼顾了同步文件读取和异步内容解析的性能平衡。

核心代码如下：

async function extractImageContent(filePath: string): Promise<string> {

const worker = await createWorker();

try {

await worker.reinitialize('eng+chi\_sim');

const { data } = await worker.recognize(filePath);

return data.text;

} finally {

await worker.terminate();

}

}

## 5.3 部署

部署部分包括前后端服务的构建与上线。前端项目经过打包后托管在Nginx或静态资源服务中。后端服务容器化打包，使用Docker部署，配合Nginx实现反向代理，支持HTTPS与接口转发。数据库部分选择MySQL部署。最终部署环境选择华为云服务器，并结合日志监控系统稳定运行。

### 5.3.1 前端挂载

整个前端部署流程从构建到Docker容器运行，首先使用pnpm build命令将前端项目打包，生成静态文件并输出到dist目录中。接着通过Dockerfile构建镜像，基于最新的nginx官方镜像，设置工作目录为/app，将dist目录下的所有文件复制到nginx默认的静态文件目录/usr/share/nginx/html中。

Docker配置如下：

FROM nginx:latest

EXPOSE 81

WORKDIR /app

COPY dist/ /usr/share/nginx/html

COPY nginx.conf /etc/nginx/conf.d/default.conf

CMD ["nginx","-g","daemon off;"]

同时将自定义的nginx.conf配置文件覆盖默认配置。nginx配置中特别设置了监听81端口（与Dockerfile中EXPOSE的端口一致），配置了访问日志和错误日志路径，并通过try\_files指令实现前端路由的History模式支持，确保直接访问子路由或刷新页面时能正确返回index.html。

nginx.conf配置如下：

server {

listen 81;

server\_name localhost;

access\_log /var/log/nginx/access.log main;

error\_log /var/log/nginx/error.log error;

location / {

root /usr/share/nginx/html;

index index.html index.htm;

# 浏览器请求的资源不存在时，返回404

try\_files $uri $uri/ /index.html;

}

# error\_page 404 /404.html;

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root /usr/share/nginx/html;

}

}

当运行该Docker容器时，nginx会以前台模式启动（通过daemon off参数），持续监听81端口请求。对于所有静态资源请求，nginx会从/usr/share/nginx/html目录提供文件；当请求的路由不存在时会回退到index.html交由前端路由处理。错误处理方面配置了50x错误的默认页面，但注释掉了404自定义页面。整个部署方案通过容器化实现了环境一致性，利用nginx高效地提供静态文件服务，并通过合理的配置解决了单页应用的路由问题。

### 5.3.2 服务端部署

服务端部署的流程与前端类似，首先同样pnpm build命令将前端项目打包，生成静态文件并输出到dist目录中，针对Nestjs后端应用的特点进行了调整,基于最新版的Node官方镜像构建，暴露服务运行的3002端口，并设置工作目录为/app。通过COPY指令将整个项目代码复制到容器内，然后运行npm install安装依赖。最后通过CMD启动编译后的入口文件dist/main.js

核心代码如下：

FROM node:latest

EXPOSE 3002

WORKDIR /app

COPY

RUN npm install

CMD ["node","dist/main.js"]

### 5.3.3 数据库部署

我们需要在服务器上安装MySQL8.0镜像，接着执行运行容器的指令：

docker run -d \

--name mysql-container \

-e MYSQL\_ROOT\_PASSWORD=root \

-e MYSQL\_DATABASE=mydb \

-e MYSQL\_USER=user \

-e MYSQL\_PASSWORD=password \

-p 3306:3306 \

-v mysql\_data:/var/lib/mysql \

mysql:8.0

通过一条命令就能快速部署一个功能完整的MySQL服务。这里配置了root用户密码为"root"，并创建了名为"mydb"的数据库，同时设置了普通用户访问密码。服务端口映射将容器内的3306端口直接暴露到宿主机的3306端口，确保外部应用可以方便连接。数据持久化方面，通过volume挂载将容器内的/var/lib/mysql目录映射到名为mysql\_data的Docker卷，保证数据库文件在容器重启或删除后依然保留。

这种部署方式轻量快速，大大简化了传统数据库安装配置的复杂流程。

结束语

在大模型快速发展的时代，一个优质的知识库问答系统是企业或者学校本系统围绕“智能问答”的核心目标展开设计与实现，基于NestJS+MySQL构建后端服务，前端采用Vue3技术栈实现交互界面。系统支持用户注册与登录、知识库文件上传与解析、大模型问答、对话记录保存、机器人配置管理等功能模块，并采用SSE实现流式回答，提升用户体验。系统整体架构清晰，功能模块划分合理，具备较强的可扩展性和可维护性。

系统当前的知识检索能力仍处于基础阶段，主要依赖全文匹配，缺少更精确的语义理解与向量化召回。缺少大量用户实测数据，尚未进行性能压测与用户反馈收集，部分功能仍停留在初级阶段。

计划引入如FAISS、Milvus等向量数据库提升知识库检索精度，实现更高质量的知识增强问答功能。在优化用户体验方面，提升聊天界面流畅度、增加对话多轮追问支持、支持知识库文件的增量更新与批量管理。建设后台运营系统，支持对用户数据、对话数据、知识库使用情况进行统计分析，为系统优化与推广提供决策依据。通过部署CI/CD脚本，实现一键部署与多环境支持，进一步增强系统的工程实用性和部署便利性。

致 谢

至此，大学生涯就要在这里画上句号了。

本项目的完成过程中，我得到了许多老师、同事、朋友和家人的大力支持与帮助。在此，我谨向所有在项目开发与研究过程中给予我灵感、指导、帮助和鼓励的人表示最诚挚的感谢。

首先，感谢QAnything给了我本次项目的开发灵感，感谢我的同事刘俊雄以及组内的同事们。在网易实习的这段经历中，从接触到这个项目，到夜以继日的学习，让我萌生了“实现一个自己的问答系统”的想法。借着空闲时间去学习它的架构，开发时请教雄哥，我不断地深挖，请教大家，最终将这个系统以毕设的形式展示给大家。

其次，我要特别感谢我的导师王红玲，她不仅在专业知识上给予了我极大的帮助，更在项目设计思路、系统结构架构以及技术细节方面给予了我悉心的指导。在项目遇到瓶颈和困难时，她总能耐心地提供思路，帮助我重新理清方向。

接着，我要感谢从2024年6月份至今，所有参与过项目合作的同事们，不论是前端、后端、测试、产品、UI等等，你们严谨的态度、细致的代码风格以及对问题的深刻见解都给我留下了深刻印象，是你们的陪伴和支持给予了我无限进步的动力。

同时，我也感谢项目过程中参考和学习过的优秀开源项目、文档及社区资源，是这些宝贵的资料为我的开发提供了理论与实践基础。

最后，我要感谢我的家人对我的理解与支持，是他们给予我充足的时间和良好的环境，让我能够安心地投入到项目之中。正是因为有你们的默默支持，我才能坚持走到最后。

再次衷心感谢所有帮助、鼓励和支持我的人！

参考文献

1. 梅忆寒,王琳琳,王鹏飞,等.基于多模态与检索增强生成的数据库知识问答系统[J].计算机教育,2024,(12):20-34.
2. 杜恒峰.DeepSeek-R1惊艳全球展示中国AI发展巨大潜力[N].每日经济新闻,2025-01-27(001):1.
3. 穆肃,陈孝然,周德青.生成式人工智能赋能教学设计分析：需求、方法和发展[J].开放教育研究,2025,31(01):12-23.
4. 刘永东,王文涛,胡鹏.本地化知识库问答系统研究与实现[J].软件学报,2022,33(6):1428-1439.
5. 姜嘉伟.基于Langchain-LLMs框架的智能问答系统的设计与实现[D].延边大学,2024:35.
6. 任海玉,刘建平,王健,等.基于大语言模型的智能问答系统研究综述[J/OL].计算机工程与应用,1-24[2025-02-06].
7. Radeva I ,Popchev I ,Doukovska L , et al.Web Application for Retrieval-Augmented Generation: Implementation and Testing[J].Electronics,2024,13(7):15.
8. Mansurova A ,Mansurova A ,Nugumanova A .QA-RAG: Exploring LLM Reliance on External Knowledge[J].Big Data and Cognitive Computing,2024,8(9):115-115.
9. 刘喆昕.网页中PDF文档展示方案的研究[J].现代信息科技,2023,7(20):12-13.
10. Tyson M .Intro to Nest.js: The higher-order JavaScript and TypeScript server[J].InfoWorld.com,2023:5-12.
11. 赵佳英.多模态文件内容检索程序设计与实现[J].电脑知识与技术,2024,20(23):1-10.
12. 赵率宏.基于Node.js的ORM框架研究与实现[D].西南科技大学,2023:4-12.
13. 芬兰：国家档案馆推出新的开源OCR工具[J].陕西档案,2025,(02):61.