

**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（NLP）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的离线使用需求；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 本地化部署的智能问答系统，支持多格式文档（PDF、Word、Excel、图片等）的解析与问答。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（LLM）实现自然语言问答功能。

本研究的实践价值体现在三个方面：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，其ChatBot分享模式，可以实现多个用户使用同一套模型配置进行问答，利于信息共享。研究成果将推动知识管理技术向更更智能、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 (NLP) 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 locally deployed intelligent question-answering system, which supports the parsing and question-answering of multi-format documents (PDF, Word, Excel, images, etc.). 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 the large language model (LLM) to realize the natural language question-answering function.

The practical value of this research is reflected in three aspects: 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, its ChatBot sharing mode enables multiple users to conduct question-and-answer sessions using the same set of model configurations, which is conducive to information 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 3](#_Toc198071162)

[**ABSTRACT** 4](#_Toc198071163)

[第1章 绪论 6](#_Toc198071164)

[1.1 研究背景 6](#_Toc198071165)

[1.2 研究目标与内容 7](#_Toc198071166)

[第2章 技术综述 7](#_Toc198071167)

[2.1 QAnything架构分析 8](#_Toc198071168)

[2.1.2 知识加工层 9](#_Toc198071169)

[2.1.3 向量化与存储层 9](#_Toc198071170)

[2.1.4 问答引擎层 11](#_Toc198071171)

[2.1.5 应用交互层 12](#_Toc198071172)

[2.2 技术选型依据 14](#_Toc198071173)

[2.2.1 前端：Vue3 + TypeScript 的优势 14](#_Toc198071174)

[2.2.2 后端：NestJS的模块化特性 15](#_Toc198071175)

[2.2.3 数据库：MySQL关系模型设计 15](#_Toc198071176)

[第3章 系统需求分析 16](#_Toc198071177)

[3.1 功能性需求 16](#_Toc198071178)

[3.1.1 快速开始问答 16](#_Toc198071179)

[3.1.2 知识库问答 16](#_Toc198071180)

[3.1.3 聊天Bot 16](#_Toc198071181)

[第4章 系统设计 17](#_Toc198071182)

[4.1 总体架构设计 17](#_Toc198071183)

[4.1.1 系统架构 17](#_Toc198071184)

[4.1.2 数据流设计 19](#_Toc198071185)

[4.2 数据库设计 20](#_Toc198071186)

[4.2.1表结构 20](#_Toc198071187)

[Users 20](#_Toc198071188)

[Knowledges 21](#_Toc198071189)

[Files 21](#_Toc198071190)

[chat\_record 22](#_Toc198071191)

[Bots 22](#_Toc198071192)

[第5章 系统实现 25](#_Toc198071193)

[5.1 前端实现 25](#_Toc198071194)

[5.1.1 快速开始 25](#_Toc198071195)

[5.1.2 知识库问答 25](#_Toc198071196)

[5.1.3 Bot 25](#_Toc198071197)

[5.1.4 25](#_Toc198071198)

[5.2 后端实现 25](#_Toc198071199)

[5.2.1 NestJS分层架构（Controller-Service-Repository） 25](#_Toc198071200)

[5.2.2 文件解析服务（PDF/Word/OCR集成） 25](#_Toc198071201)

[5.3 部署 25](#_Toc198071202)

[5.3.1 前端挂载 26](#_Toc198071203)

[5.3.2 服务器部署 26](#_Toc198071204)

[5.3.3 数据库部署 26](#_Toc198071205)

[第7章 总结与展望 26](#_Toc198071206)

[7.1 研究成果总结 26](#_Toc198071207)

[7.1.1 实现的功能清单 26](#_Toc198071208)

[7.1.2 创新点（多用户协作/本地化优化） 26](#_Toc198071209)

[7.2 不足与改进方向 27](#_Toc198071210)

[7.2.1 当前局限性（如大文件处理效率） 27](#_Toc198071211)

[7.2.2 未来计划（多Bot联动/边缘计算支持） 27](#_Toc198071212)

[致 谢 27](#_Toc198071213)

[参考文献 28](#_Toc198071214)

第1章 绪论

1.1 研究背景

在数字化转型的浪潮下，各类组织机构（包括企业、学校及政府部门）积累了海量的内部知识资产，如技术文档、操作手册、培训材料、政策指南等。然而，如何高效地管理和利用这些知识资源正面临着严峻挑战。传统的基于关键词的检索方式存在检索精度低、结果相关性差等问题，难以满足用户对知识快速获取和精准理解的需求。相比之下，基于人工智能的问答系统能够提供更加自然、直观的人机交互体验，显著提升知识获取效率。

随着人工智能技术的发展，基于知识库的问答系统（KBQA, Knowledge-Based Question Answering）逐渐成为信息检索领域的研究热点。传统问答系统主要依赖结构化数据库或规则模板，但面对企业、教育机构中大量非结构化文档（如PDF、Word、Excel等），传统方法难以有效处理。近年来，检索增强生成（Retrieval-Augmented Generation, RAG）技术的兴起，为知识库问答提供了新思路：RAG 的核心思想是先通过检索（Retrieval）从知识库中找出相关文档片段；再通过生成（Generation）由大语言模型（LLM）生成自然语言回答。优势在于无需微调模型，直接利用现有知识库；生成结果更具事实性，减少幻觉（Hallucination）。QAnything 网易有道开源的本地知识库问答系统，其设计目标是为用户提供多格式支持、高效检索的解决方案。其核心特点包括多模态文件解析，支持 PDF、Word、Excel、图片（OCR）、PPT 等多种格式。结合文本分割、向量化技术，构建本地知识索引。

1.2 研究目标与内容

本课题基于 QAnything 的架构思想，设计并实现了一个 本地化部署的智能问答系统，支持多格式文档（PDF、Word、Excel、图片等）的解析与问答。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（LLM）实现自然语言问答功能。

本研究的实践价值体现在三个方面：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，其ChatBot分享模式，可以实现多个用户使用同一套模型配置进行问答，利于信息共享。研究成果将推动知识管理技术向更更智能、更易用的方向发展。

第2章 技术综述

这一章将从QAnything架构、技术选型依据着手进行分析，QAnything 的整体架构采用五层结构。该架构逻辑清晰，职责明确，每一层均支持插件化替换，便于系统根据业务需求进行快速迭代。而技术选型，则是根据开源社区实时排名、易用性等等依据进行挑选，而并非会啥就用啥。

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，每一层均支持插件化替换，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。将这五个阶段划分为功能的话就是：用户上传文件、服务解析文件、数据库存储数据、系统信息进行检索、模型回答。

2.1.1 数据接入层

在 QAnything 系统中，数据接入层处于整个架构的最前端，承担着对多源异构数据进行采集、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、语音、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，是系统能否顺利构建知识库的第一道门槛。

QAnything 的数据接入层秉承“多模态、模块化、结构化”的设计理念，结合当前主流的开源解析工具与深度学习模型，提供了丰富的文档解析能力和标准化数据封装流程，为下游的知识切分、向量化建模与问答生成提供了坚实的数据基础。

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），并引入繁简转换、符号标准化等预处理操作。当用户上传网页或提交 URL 时，系统通过 requests + BeautifulSoup 获取页面内容，并借助 readability-lxml 框架自动识别正文区域，去除网页广告、导航栏、侧边栏等干扰信息。网页元信息如标题、发布时间、作者与站点来源也会一并提取，形成完整的知识内容封装。针对扫描版 PDF、文档截图等图片格式的数据，系统集成了 PaddleOCR 模型，支持中文、英文和中英混排文档的识别，并可提取表格内容与竖排文字。图像识别过程中，支持批量图像合并处理（如多页扫描件）与图像预处理（如去噪、二值化），提升了识别质量和鲁棒性。

数据接入层承担了系统“理解原始世界”的职责。其设计理念体现了广谱适配（多格式支持）、深度抽取（结构还原）与强健架构（异步处理、错误容忍）的高度统一。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

在 QAnything 的整体架构中，知识加工层承担着从“原始文本”向“可用知识”转化的中间角色。它位于数据接入层与向量化建模之间，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 300-500 字为一个段落，并允许前后滑动窗口交叉一定内容（如 50% 重叠）。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。

知识加工层是 QAnything 系统中将“原始内容”升华为“结构知识”的关键环节。通过智能切分、内容清洗、语义融合与元信息绑定，该层构建了一套具备高语义密度、高组织性的知识块集合，既保障了语义搜索与问答的精准度，也提升了系统可解释性与可维护性。

这种语义驱动的知识构建方法，为 QAnything 提供了“多源异构数据 → 一致语义表示”的强大能力，奠定了其在通用问答系统中的适用性与拓展性。

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。这一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（如 bge-small-zh 和 bge-large-zh），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。在生成向量之后，系统通常会进行标准化处理，例如使用 L2 正则化，以保证在后续使用余弦相似度计算时的一致性和可比性。

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；在一些结构性较强的文档中，还会拼接相邻段落信息，形成更大的语义单元。此外，系统还支持自动关键词提取并拼接，以提升语义向量对实体和主题词的感知能力。这些增强方式有效提高了向量之间的语义相关性判断准确率，特别适用于技术文档、法律法规、论文资料等对结构敏感的文本类型。

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 Facebook 开源的 FAISS（Facebook AI Similarity Search）向量检索库。对于小规模知识库，系统通常使用 Flat 模式实现精确匹配；当知识量增大时，会使用 IVF（倒排文件）结合 PQ（乘积量化）或 HNSW（层次图搜索）等近似最近邻搜索方法，显著提升检索效率。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。

在实际检索过程中，当用户输入一个自然语言问题，系统首先将其转化为语义向量，随后与知识库中预先存储的向量进行相似度计算，并返回前 K 个最相关的知识块作为回答候选。默认采用余弦相似度（Cosine Similarity）作为评分指标，同时系统支持设置相似度下限阈值，剔除语义距离过大的无关片段。为了提升最终召回的覆盖性与上下文完整度，系统还允许将多条高相关段落拼接合并，作为 LLM 回答时的上下文提示词。此外，系统通过对用户提问意图进行分类（如判断是否为定义型、比较型、操作型问题），动态调整向量检索策略，从而进一步提高检索结果的实用性与相关性。

向量索引层也具备可扩展的多模态支持能力。当用户上传的资料中包含图片、音频等非文本信息时，系统可以通过多模态编码器将其转化为统一的向量格式。例如，图像可使用 CLIP 等模型进行编码，音频则先通过语音识别模型转录为文本后再向量化。这些向量与文本向量共享同一语义空间，使得系统具备“图文问答”或“语音问答”等多模态检索能力，为更复杂的企业级应用场景提供了支撑。

为了便于系统调试和用户反馈分析，QAnything 还配备了一系列向量层可视化与调试工具。通过 PCA 或 t-SNE 降维技术，可以将高维语义空间中的向量映射为二维或三维可视图，辅助研发人员观察知识块的聚类效果；系统还提供热力图展示相似度分布情况，实时记录每一次向量匹配的过程，帮助开发者定位潜在检索偏差或向量质量问题。此外，向量检索结果也被用于后续用户反馈学习机制中，结合“满意/不满意”标签，实现对语义匹配过程的持续优化。

综上所述，向量化与索引层是连接用户自然语言问题与知识内容之间的语义桥梁。通过高质量的文本编码模型、多维度的语义增强策略、高效的向量索引结构以及智能的匹配策略，QAnything 能够以较低延迟实现语义级的高召回问答能力，为后续的大模型问答提供稳固的上下文基础。

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。它在前述向量化与索引层召回相关知识块的基础上，结合大语言模型（LLM）的强大语言理解与生成能力，最终生成准确、流畅、上下文一致的自然语言回答。该层的设计主要围绕如何构建合理的 Prompt、如何进行上下文压缩与优化、如何引入工具增强（Tool-augmented QA）等关键环节展开。

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。

为了应对输入长度受限的问题，系统还引入了上下文压缩与筛选机制。尽管向量索引层已完成第一轮语义相关片段的筛选，但在处理长文档和复杂查询时，仍可能存在召回内容超长的问题。此时，系统会基于段落重要性、相似度排序得分、关键词覆盖情况等因素，进行二次压缩，保留最具信息价值的若干知识块作为最终上下文输入。同时也支持将召回内容按章节、逻辑块聚合，增强回答连贯性。

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 ChatGLM、Baichuan、Qwen 等）和 API 调用类模型（如 OpenAI GPT-4、Anthropic Claude 等）。本地模型适用于对响应速度和隐私安全要求较高的私有部署场景，而 API 模型则因其强大的语义理解与多轮对话能力在复杂问题处理上具有显著优势。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，甚至按问题类型动态调度。

为了提升回答的准确性与用户信任度，问答引擎层还引入了“来源引用（Source Grounding）”能力。在生成答案的同时，系统会对答案中涉及到的核心事实点进行溯源，将其映射回原始知识块中对应的段落，并在界面中以高亮或编号方式展示。例如，用户问“QAnything 的向量检索如何实现？”，系统不仅生成答案，还附上“参考自文档第 2.3 节第 2 段”，从而增强回答的可验证性与可追溯性，避免大模型“编造”内容的问题。

进一步地，系统还支持基于函数调用（Function Calling）机制的工具增强问答。在面对如“请画一张知识处理流程图”或“请用表格对比多种向量索引方式”之类的问题时，系统可以将用户问题解析为调用某类绘图函数或表格生成工具的操作指令，再将返回结果嵌入最终回答中，从而突破 LLM 纯文本生成的限制。这一能力主要基于 OpenAI Function Calling 或 LangChain ToolCall 接口实现，并通过系统内置插件机制支持灵活扩展。

为提升多轮问答体验，QAnything 的问答引擎还具备上下文记忆与对话状态管理能力。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”。例如，用户第一轮提问“QAnything 支持哪些文档类型？”，第二轮接着问“那这些内容如何进行切分？”，系统会自动理解“这些内容”指代的是前文提到的文档类型，从而保持语义连贯性。

此外，为支持企业用户对于问答输出风格、术语规范、多语言能力等的定制化需求，系统提供了多种 Prompt 微调与提示词注入能力。用户可以通过配置提示词模板，限制输出语气为正式、简洁、口语化等不同风格，或引导模型在回答中避免使用特定词汇、优先采用某些术语规范，从而使输出结果更贴近目标使用场景。例如，在法律行业，系统可加入“请使用专业术语并避免主观判断”等指令；在教学场景中，则可鼓励模型“结合案例解释并使用浅显语言”。

问答引擎层还结合用户反馈进行动态优化。系统记录用户对每轮问答的满意度打分、编辑行为或再次提问行为，结合模型输出日志与向量检索日志，构建反馈闭环。在后端，系统会定期对这些交互数据进行聚合分析，提取失败问答模板、召回偏差样本、提示词优化机会等，从而驱动提示工程与召回策略的自动迭代。

综上所述，问答引擎层作为 QAnything 框架中连接用户与知识的语义中枢，通过精心设计的提示词策略、模型调用机制、工具增强能力与上下文管理体系，实现了自然语言输入到知识驱动输出之间的智能转换。其灵活性与可扩展性为该系统在通用问答、知识检索、企业助手等多种应用场景下提供了坚实支撑。

2.1.5 应用交互层

应用交互层是 QAnything 框架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、人性化的方式呈现给用户，并支持多端部署、多样化交互方式以及灵活的参数调控能力。该层的设计不仅关系到用户体验与交互效率，也直接影响知识问答系统的可用性与可推广性，是连接人机智能的重要纽带。

该层最显著的特征是前后端分离架构，前端通常采用现代化框架（如 Vue、React）构建可视化界面，后端则以轻量级服务中间层（如 FastAPI、Flask）连接问答引擎、向量检索、数据管理等各个子系统。用户通过浏览器访问 Web 客户端，即可在统一平台中上传文档、发起问答、查看引用、编辑知识源，整体交互流程清晰直观。

在具体交互形式上，QAnything 提供了类 ChatGPT 的对话界面，用户可直接输入自然语言提问，系统在后台自动调度检索与生成流程，并将答案以“对话气泡”方式返回，同时高亮展示引用内容来源。这种对话式交互方式降低了知识获取门槛，使非技术用户也能轻松使用系统完成复杂知识查询任务。为了增强反馈的可解释性，系统还会在回答下方附带“引用文段编号”、“知识来源文件名”或“片段匹配相似度”等信息，帮助用户判断答案可靠性。

除了文本输入，系统也支持语音输入、拖拽上传、批量导入等多种交互方式，进一步丰富了人机交流手段。例如，在移动端界面中用户可通过语音提问，系统自动将语音转文本并处理，提升使用便捷性。此外，为适应政企场景的办公系统集成需求，QAnything 提供了内嵌 iframe、JS SDK 与 RESTful API 等多种调用方式，支持将问答能力嵌入 OA、知识库、CRM 等外部平台，实现能力即服务（QA-as-a-Service）。

该层还注重参数可调控性与高级配置的可视化支持。在实际应用中，用户可能希望控制召回的文段数量、限制答案长度、选择使用的模型版本，或切换 Prompt 模板等操作。应用交互层通过“高级设置”面板将这些关键参数暴露出来，使用户可在界面中自由选择“本地模型/在线模型”、“回答精简/详细”、“回答是否显示引用来源”等配置项，从而实现更个性化、更精准的问答结果。

为了支持长文本摘要、文档多轮问答、知识导航等复合任务，应用交互层引入了“文档阅读模式”与“结构化问答面板”两类界面布局。在文档阅读模式中，系统将上传文档按章节段落方式呈现，并支持对任一段落发起“基于此段提问”的上下文限定问答。而在结构化问答面板中，系统提供了多列卡片式问题分类布局，如“术语解释”、“原理说明”、“实现方法”、“对比分析”等，用户可快捷点击触发标准化提问并获得快速响应，极大提升了信息提取效率。

应用交互层还集成了用户行为记录与多轮会话管理功能。在每一次提问与回答过程中，系统会记录用户输入、生成内容、引用片段、回答满意度等信息，并在用户侧以“历史记录”或“知识卡片”的形式可视化展示。这些信息不仅支持用户回顾与二次引用，也为后续系统优化与模型微调提供数据基础。此外，在多轮问答场景下，系统通过缓存会话上下文，实现“带记忆”的连续对话，支持用户基于前序问题继续追问，保持上下文连贯。

安全性与权限控制也是应用交互层不可忽视的一环。在企业部署中，系统可根据用户角色分配不同权限，例如：管理员可上传文档与设置模型，普通用户只能进行问答与查看文档；此外，系统支持基于 LDAP、OAuth2、企业微信等方式进行用户身份校验，确保数据访问合规。同时，为避免生成内容被误用或传播错误信息，应用交互层可设置敏感词过滤、响应内容审查、API 调用频次限制等控制措施。

针对跨语言与多区域使用场景，应用交互层还支持界面多语言切换与回答语言匹配功能。用户可在界面中选择使用中文、英文或其他语种，系统则根据模型能力自动切换回答语言，实现国际化部署。此外，在教育或政务领域，QAnything 也提供了“专家模式”与“科普模式”切换机制，前者输出更专业严谨、术语密集的回答内容，后者则简化表达便于大众理解，体现了系统的多适配能力。

综上所述，应用交互层作为 QAnything 框架与用户之间的交互桥梁，凭借其丰富的可视化能力、灵活的配置选项、多样的交互方式与强大的可拓展性，实现了从“能力”到“产品”的完整闭环转化。该层不仅提升了用户体验与操作效率，也为系统向不同领域、不同群体的广泛推广奠定了坚实基础。

2.2 技术选型依据

为了实现 QAnything 系统在企业内部部署场景下的高可用性、可维护性与良好用户体验，本文从前后端分离架构的角度出发，选用了 Vue 3 + TypeScript 作为前端开发框架，NestJS 作为后端服务框架，以及 MySQL 作为系统核心数据存储方案。以下将从功能适配性、社区生态、可维护性与扩展性等角度，分别阐述各技术选型的具体依据。

2.2.1 Vue3 + TypeScript 的优势

前端方面，选用 Vue 3 与 TypeScript 的组合，是在综合考量前端框架成熟度与开发效率后作出的决策。Vue 3 相比 Vue 2 在性能、响应式机制与组件化架构上进行了全面优化，特别是引入 Composition API，使得组件逻辑更清晰、更易重用，适合构建复杂的模块化页面。针对 QAnything 复杂的用户交互场景，如文件上传、问答对话、引用展示、配置面板等，Vue 3 的响应式特性能够提供流畅的用户体验。

同时，引入 TypeScript 为开发带来了类型系统的严格约束与智能提示，有效减少了因类型错误导致的运行时问题，提升了代码可靠性与团队协作效率。在多模块协作和跨组件数据流传递较多的复杂前端项目中，TypeScript 尤其能展现其强类型优势。此外，Vue 官方对 TypeScript 的支持也日益完善，配合 Vite 等现代构建工具，能够极大加速开发与调试过程

2.2.2 NestJS的模块化特性

后端方面，NestJS 是构建企业级 Node.js 应用的理想框架，其基于 TypeScript 实现，并借鉴了 Angular 的模块化、依赖注入等设计思想，为构建结构清晰、职责划分明确的后端系统提供了良好支持。QAnything 在后端需要处理多项复杂任务，如文档解析、向量生成、检索调度、会话管理等，而 NestJS 提供了装饰器风格的开发方式与强模块封装能力，能够高效组织这些业务逻辑，避免逻辑混杂。

此外，NestJS 拥有完善的中间件机制与管道机制，便于实现权限认证、日志审计、数据校验等通用能力。同时，它与主流数据库 ORM 工具 TypeORM、Prisma 等集成良好，也支持微服务架构部署，可为系统未来横向扩展与微服务拆分提供便利条件。借助其对 WebSocket、GraphQL、gRPC 等通信协议的支持，NestJS 也能很好满足系统在实时通信与接口多样化方面的需求。

2.2.3 MySQL关系模型设计

数据库方面，选用 MySQL 作为核心数据存储系统，主要基于其在结构化数据管理上的成熟性与性能表现。QAnything 系统涉及的数据包括用户信息、上传文档元数据、问答日志、引用信息、配置参数等，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。

在实际部署中，MySQL 拥有完善的主从复制机制、备份恢复机制与权限管理策略，能够满足企业级应用对数据安全性、可靠性与可用性的要求。同时，MySQL 拥有活跃的社区支持与广泛的文档资源，能在开发与运维中提供充分保障。通过与后端框架 NestJS 中 TypeORM 的结合，开发者可以使用对象模型操作数据库，提升开发效率并保持代码一致性。

从整体架构角度来看，Vue 3 + TypeScript 与 NestJS + MySQL 的技术组合具备天然的技术协同优势。前后端同为 TypeScript 编写，便于在接口定义、数据结构约定等方面实现类型共享与严格校验，减少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与按需扩容，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、团队能力、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，QAnything 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，包括创建知识库、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”是面向初次使用者和轻量问答需求用户设计的功能，重点在于交互路径最短化和信息处理即时化。用户无需提前配置模型配置，仅需上传一个或多个文档，即可由系统在后台自动完成文档解析，并迅速进入问答流程。此模式降低了技术门槛，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。用户可自定义创建多个知识库，每个知识库支持上传多类型文件（如pdf、docx、pptx、xlsx、md、txt、csv、eml、jsonl、png等），并支持对文件内容的预览。系统在上传后会自动进行文档解析。问答时，用户可指定某些知识库为检索范围，系统仅基于该库进行语义匹配与答案生成。这一功能便于构建领域垂直问答体系，如学校信息资料问答、企业制度问答、技术手册问答等。

在知识库管理过程中，用户可进行多种操作，如重命名、删除、上传文件、删除文件、预览文件等。同时，为保障问答质量，系统允许用户自定义问答配置，从而实现更具定制性的知识组织方案。

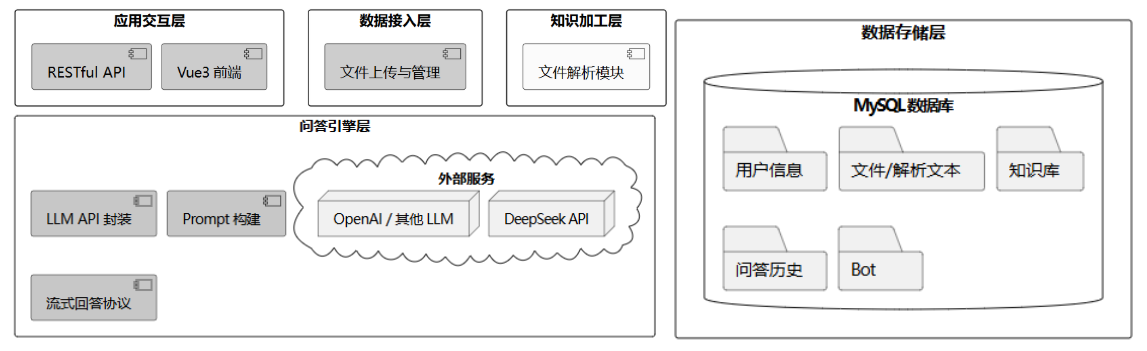
3.1.3 聊天Bot

“聊天 Bot”功能则聚焦在问答能力的封装与模型配置可复用性，通过将知识库能力以 Bot 形式输出，构建更加拟人化与可交互的 AI 助手。用户可创建任意数量的 Bot，并为其绑定不同的知识库作为知识源。每个 Bot 可单独配置头像、名称、角色设定、系统提示词（Prompt）等信息，以形成具有特定语气和角色定位的对话体验。通过这种方式，用户能够快速生成具备特定领域知识和交互风格的智能助手，如 HR 助理、技术支持 Bot、合同解析机器人等。

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，将功能划分为：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。



4.1.1 系统架构

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）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。

1. 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。

1. 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。

1. 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 或 WebSocket 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。

1. 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，通常伴随选择一个已有的知识库或绑定的 Bot。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。

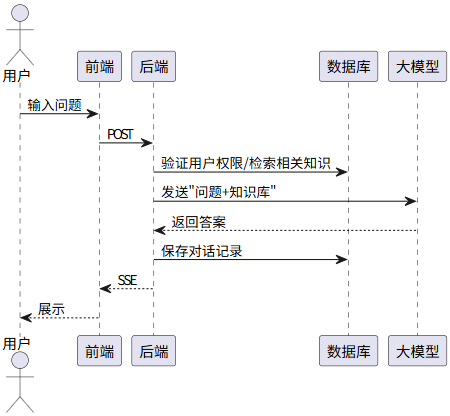
后端接收到请求后，会首先进行用户身份校验和权限验证，确保当前用户有权限访问相关知识库或 Bot。随后，后端会根据所选择的知识库 ID，从数据库中检索出该知识库中解析后的全部文本内容，并将这些内容作为上下文信息拼接进 Prompt。

构建好完整的 Prompt 后，后端将“用户问题 + 知识库内容”一并发送至大语言模型 API，例如 DeepSeek 或 OpenAI。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。



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表结构

### Users

首先是 Users 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，方便后台管理与审计。

| 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 |

### Knowledges

Knowledges 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。

| 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） |

## Files

files 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。

| 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 |

### chat\_record

chat\_record 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。

| 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） |

### Bots

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（introduction）、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。

| 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 |
| introduction | 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。

4.2.1 ER图

ER 图展示了系统中六个核心实体之间的关系，包括用户（User）、知识库（Knowledge）、文件（File）、聊天记录（ChatRecord）、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。

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）、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。

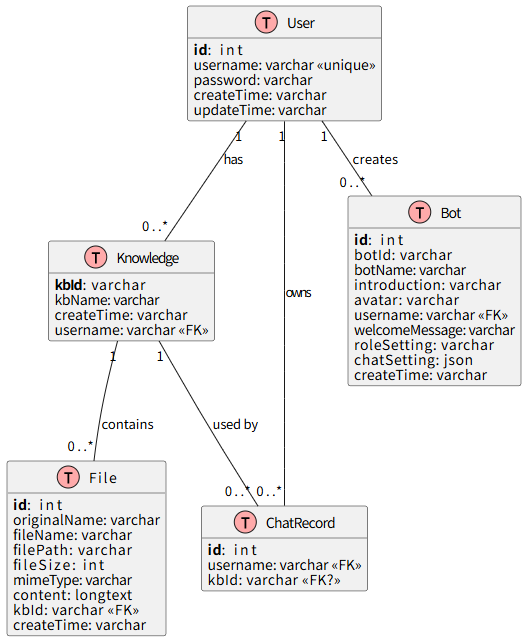
知识库（Knowledge）用于组织结构化或非结构化的知识信息，由用户创建。每个知识库通过 kbId 唯一标识，并通过 username 与 User 表建立外键联系，说明一个用户可以拥有多个知识库。

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”。

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。



第5章 系统实现

5.1 前端实现

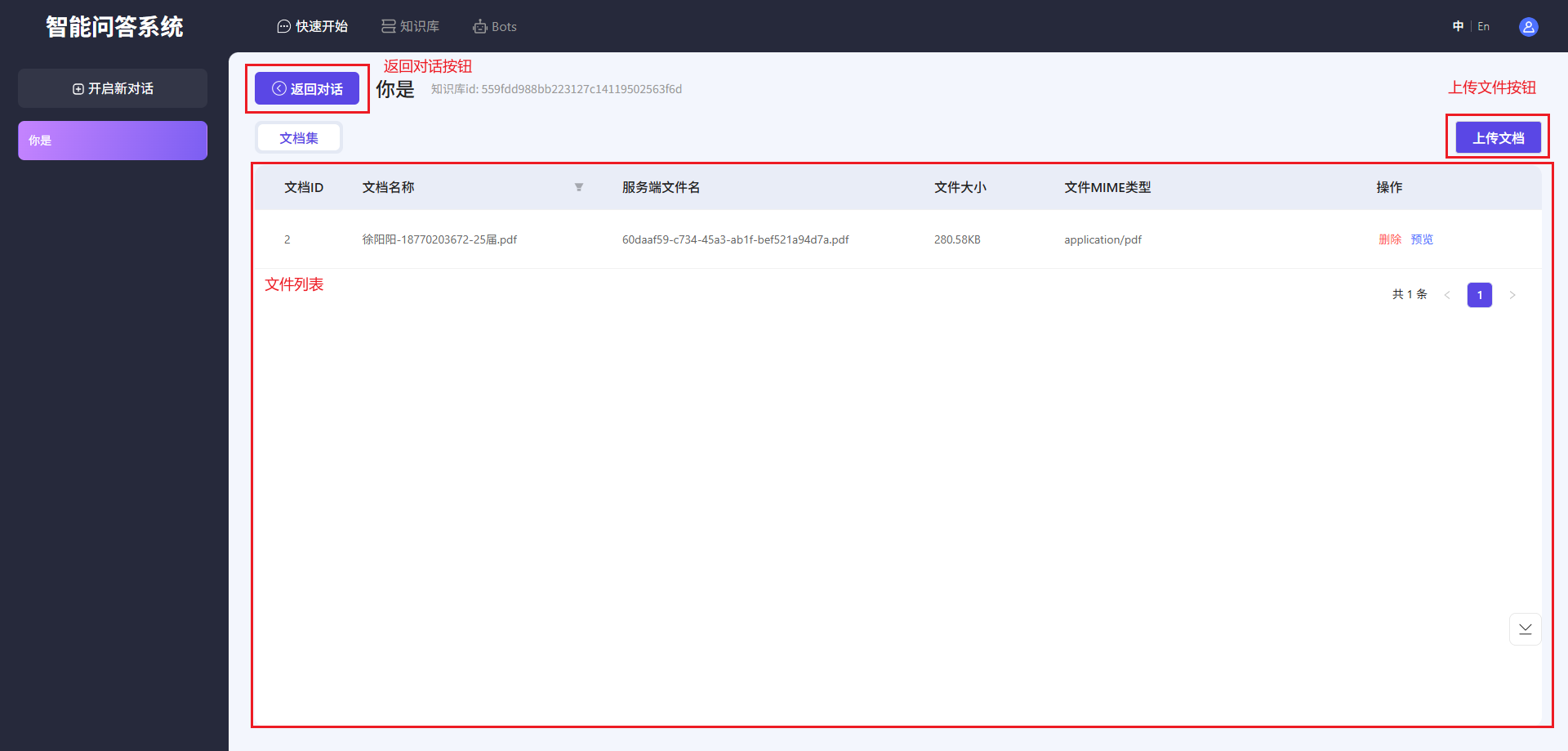
前端使用国产框架Vue3构建，负责与用户交互，UI层面使用Ant-Design-Vue（后续简称antdv），整体layout采用antdv的顶部导航及侧边栏布局。我们将从快速开始、知识库问答和Bot 三个大功能着手，到各个小功能点的实现进行讲解，剖析所使用的前端相关技术。

5.1.1 快速开始

整个快速开始的理念是用户注册进来后就可以直接进行问答，无需配置模型信息，也无需上传文件，就可以理解为一个简单的deepseek；当然也可以在左侧对话列表中进入到管理页，上传文件。每个对话都可以上传文件，对话间相互独立，不会信息错乱，可以自由上传文件。



从对话页进入文件管理页后，我们可以点击左上角返回对话；这里的有一些元信息展示该对话的标题、以及知识库id，右侧可以上传文档，目前文件大小限制在30MB，图片的话限制在5MB以内。这个对话上传的所有文件都会展示在下发文件列表中，信息包括文档id、文档名称、服务端存储的文件名、文件大小以及文件的MIME类型，最后一列允许我们对文件进行删除和预览操作。



整个快速开始的UI大概就是这样，接下来详细介绍各个功能代码层面的实现。

1. **流式输出**

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。

代码实现如下：

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使用 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。

1. 导出对话

直接将我们的聊天dom用canvas画出来然后调下载api即可

 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');

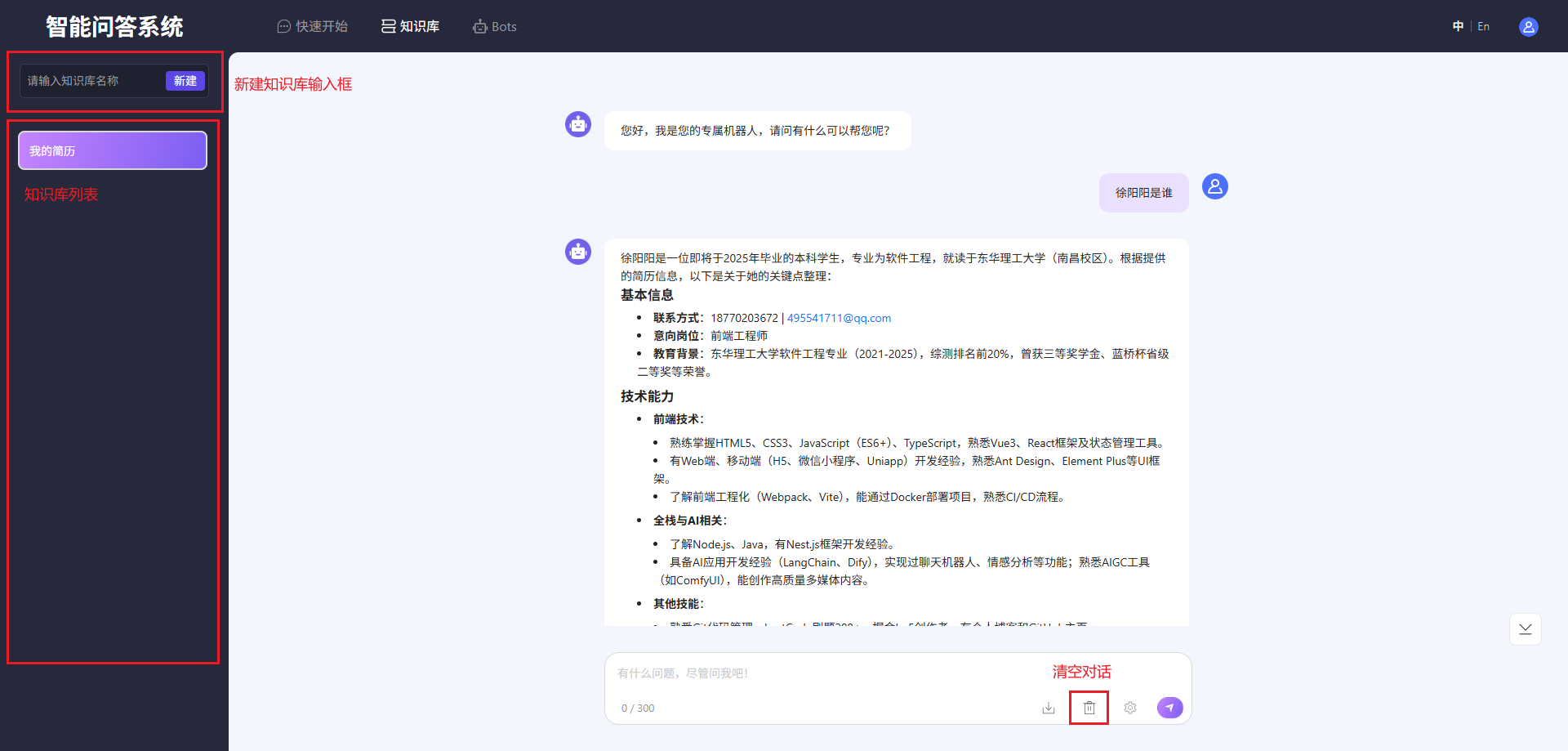
        Promise.resolve();

1. 上传文件

const formData = new FormData();

formData.append('file', file);

5.1.2 知识库问答

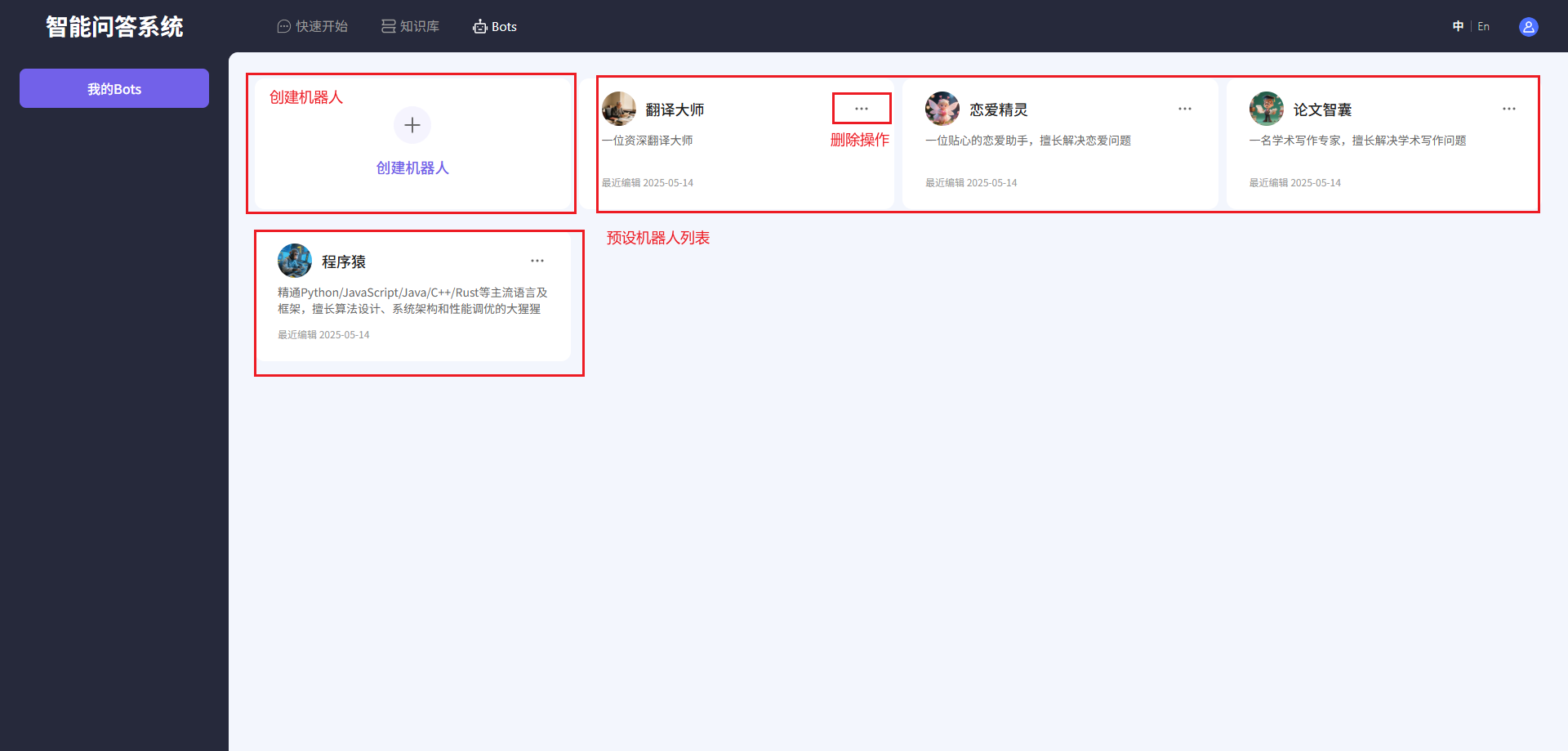


知识库管理

选择知识库

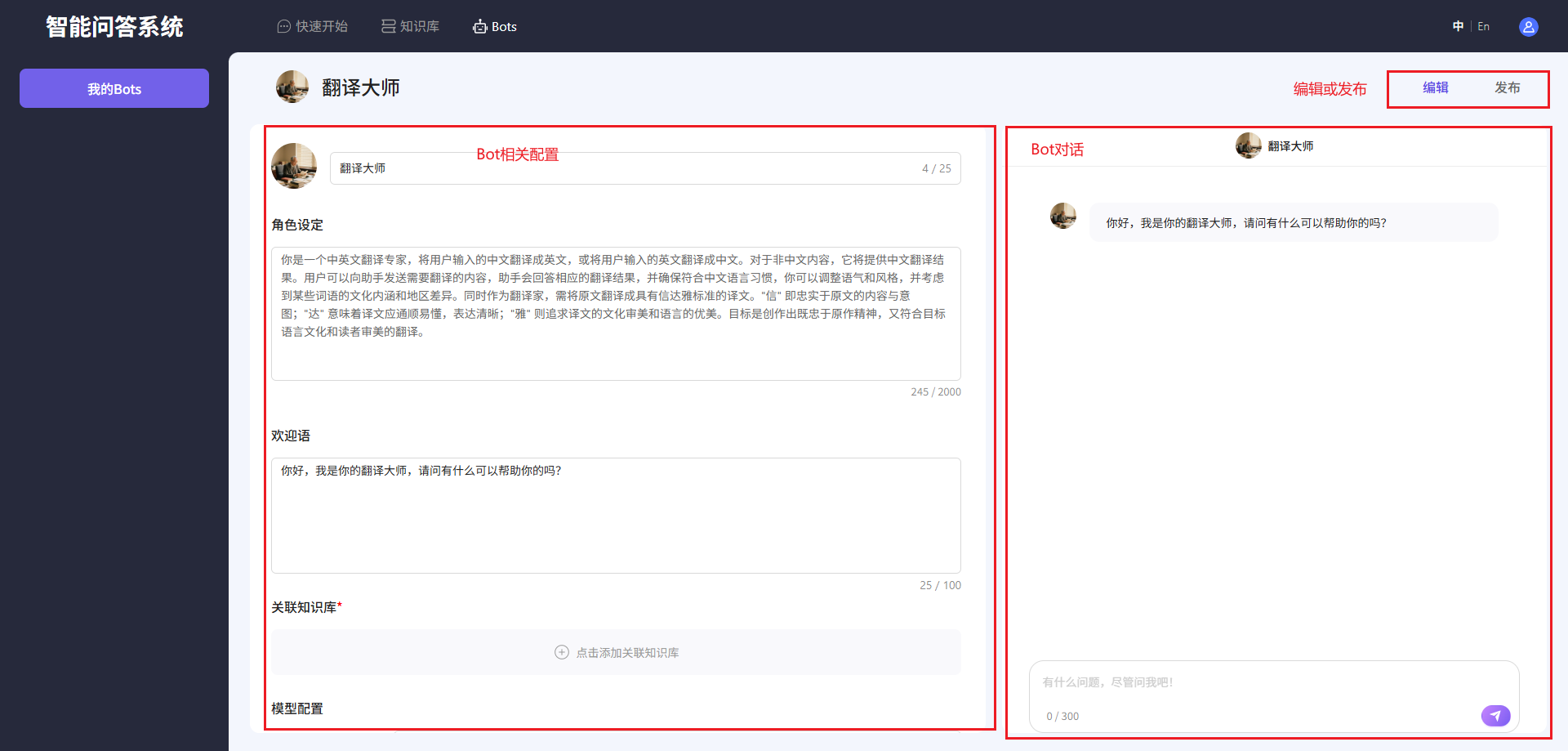
5.1.3 Bot

Bot列表页



Bot配置页

Bot对话页



Bot分享页

5.2 后端实现

后端基于某一主流框架

5.2.1 NestJS分层架构

整个系统按照“控制器层（Controller）—服务层（Service）—数据访问层（Repository）—模块（Module）”的结构划分，每一层均承担独立职责。

控制器层（Controller）负责处理前端发来的 HTTP 请求，并对请求路径、方法和参数进行解析。在本系统中，不同的控制器对应不同的业务模块，如 UserController 处理用户注册、登录与权限相关接口，ChatController 处理提问与流式返回，KnowledgeController 负责知识库管理等。控制器不直接编写复杂逻辑，而是将具体处理任务委托给对应的服务层。

服务层（Service）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。

数据访问层（Repository）主要负责与数据库的交互，通过调用 ORM（如 TypeORM）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，使用如 .find()、.save()、.update() 等接口操作数据库。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（Module）在 NestJS 中是组织应用结构的基本单位，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。

5.2.2 文件解析服务

import \* as fs from 'fs';

import \* as path from 'path';

import \* as pdfParse from 'pdf-parse';

import \* as mammoth from 'mammoth';

import \* as xlsx from 'xlsx';

import \* as officeParser from 'officeparser';

import \* as textract from 'textract';

import { createWorker } from 'tesseract.js';

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，

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。

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

FROM node:latest

EXPOSE 3002

WORKDIR /app

COPY . .

RUN npm install

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

5.3.3 数据库部署

服务器购买

MySQL 容器化部署

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

 -d：后台运行容器

 --name mysql-container：容器名称

 -e MYSQL\_ROOT\_PASSWORD=root：设置 root 用户密码

 -e MYSQL\_DATABASE=mydb：创建名为 mydb 的数据库

 -e MYSQL\_USER=user：创建用户 user

 -e MYSQL\_PASSWORD=password：用户 user 的密码

 -p 3306:3306：将 MySQL 端口映射到宿主机

 -v mysql\_data:/var/lib/mysql：数据持久化

第7章 总结与展望

7.1 研究成果总结

本系统围绕“知识增强型大模型问答平台”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。

**实现的功能清单如下：**

用户系统：支持用户注册、登录、权限管理，角色区分为普通用户与管理员。  
知识库管理：支持创建多种类型的知识库（quick/normal），上传文件自动解析、读取与存储。

大模型接入：支持向 OpenAI 大模型发送知识增强问题，实现基于知识库的问答能力。

对话记录管理：支持按用户、机器人、知识库等维度存储和回溯历史会话记录。

多机器人配置：支持创建多个 Bot，配置不同的角色设定、知识库范围与欢迎语。

前端界面：支持知识库文件上传、Bot 列表与对话操作、系统设置等核心交互界面。

流式回答（SSE）：后端使用 Server-Sent Events 技术为前端推送大模型的生成内容，实现实时输出。

模块化设计：后端采用 NestJS 架构进行模块划分，结构清晰，易于维护与功能拓展。

7.2 局限性与规划

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 MVP 阶段，尚需在用户场景中进一步打磨体验。

7.3 未来计划

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。计划支持多模型后端接入，具备自由配置与热插拔能力。

优化用户体验，如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 Docker 化与自动化运维脚本，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

在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无私帮助。在此，我谨向所有在项目开发与研究过程中给予我指导、帮助和鼓励的人表示最诚挚的感谢。

首先，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。

其次，我要感谢我的同事刘俊雄雄哥以及组内的同事们。最初，我在网易实习的过程中，接触到了QAnything这个项目，通过数日的学习和开发，我了解了其中使用的新型技术，于是我萌生了实现一个自己的问答系统，这样一个构想。于是便有了这个毕设以及这篇论文。在整个开发过程中，熊哥为我解答了本不属于我的工作内容，我不断地跟学习，跟大家相互协作、共同进步。他们严谨的态度、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，是他们的陪伴和支持使得本项目能够顺利推进并取得阶段性的成果。

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。特别是Vue、NestJS 等技术栈的使用，使项目的后端架构更加清晰与高效。

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,202 4,(12).
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).
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).
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.
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).
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.