南京大学

NANJING UNIVERSITY

研 究 生 毕 业 论 文

（申请硕士专业学位）

论 文 题 目　基于分布式任务队列的表达力评测系统设计与实现

作 者 姓 名　褚东宇

专 业 名 称　软件工程

研 究 方 向　软件工程领域

指 导 教 师　刘嘉 副教授

2020 年 5 月 6 日

学　　号：**MF1832027**

论文答辩日期：**2020 年 5 月 23 日**

指 导 教 师：　　**（签字）**

The Design and Implementation of Expressive Ability Evaluation System Based on Distributed Task Queue

By

Dongyu Chu

Supervised by

Associate Professor **Jia Liu**

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2020

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于分布式任务队列的表达力评测系统设计与实现

软件工程 专业 2018 级硕士生姓名：褚东宇

指导教师（姓名、职称）：刘嘉 副教授

摘 要

表达能力是一种非常重要的技能，但当前常见的语言类测试过于侧重词汇、语法以及文字书写的考察，不完全适用于表达能力的评估。快速表达力测试方法是专门针对这种不足设计的表达能力评估方法，但尚无配套的软件系统为其提供服务支撑。

本文对目标系统进行了需求分析，基于分布式任务队列设计并实现了表达力评测系统，可为普通用户提供评测服务，并为管理员提供相关的系统管理支持。在本文设计的测试流程中，考生可以选择个人电脑、平板以及手机等多种设备，使用浏览器或移动端的微信小程序登录系统，使用普通话口语回答问题，系统对考生的作答进行录音、存储与分析工作，最后将成绩展示给考生。系统由评测、权限管理、试题管理以及异步任务等四大模块提高功能支撑，评分算法由外部系统提供，并由异步任务模块进行调用。在架构上，系统被设计为以 Celery 任务队列为核心的分布式系统，包含多个应用服务器节点作为生产者，多个评分服务器节点作为消费者。当用户作答音频被上传后应用服务器节点生成异步评分任务放入消息中间件，评分节点监视消息中间件以获取任务，并在评分结束后将结果存储到数据库。系统的开发遵循逻辑与数据分离的原则，基于 Flask 框架完成应用服务器的开发，使用 MongoDB 进行数据持久化，使用 Redis 集中存储 session 解决会话一致性问题。此外 Redis 被用作缓存数据库提高系统性能，并通过分布式锁实现多应用服务器节点的同步，避免任务的重复执行。系统使用 Nginx 实现负载均衡，使用 Docker 和 Jenkins 完成快速部署与持续集成工作。浏览器端页面使用 Vue.js 进行开发，并编译为单页应用进行部署，提高了交互时的响应速度。

本文设计并实现的表达力评测系统为快速表达力测试方法提供了良好的服务支持，达到了预期性能，且系统扩展性和伸缩性良好。系统降低了表达能力测试中的资金、场地以及人力成本，提高了测试结果客观性和测试效率，给用户带来了更便捷的测试体验。能够为招生、招聘等工作提供参考，也为希望提升自身表达能力的人群提了供评测与练习的平台。

关键词：表达能力，评测系统，分布式，任务队列

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Design and Implementation of Expressive Ability Evaluation System Based on Distributed Task Queue

SPECIALIZATION: Software Engineering

POSTGRADUATE: Dongyu Chu

MENTOR: Associate Professor Jia Liu

Abstract

Expressive ability is a very import ability in people's daily lives. But the current common language tests are not entirely suitable for its evaluation as they focus too much on vocabulary, grammar, and handwriting. The Quick Expressive Ability Evaluation Method is a new test method dedicated for expressive ability evaluation. But currently there is no software system to provide service for it.

This thesis analyzes the requirements of the target system, designs and implements an expressive ability evaluation system based on distributed task queue. It can provide evaluation services for ordinary users and provide related system management functions for administrator users. To take an expressive ability test, users can choose a variety of devices such as personal computers, tablets and mobile phones, log in to the system using a browser or the WeChat micro program. The answers should be made by speech in Mandarin. The system will record, store, and score users' speeches. The system is divided into four modules: evaluation module, permission management module, questions management module, and asynchronous tasks module. The scoring algorithm is provided by an external system and called by the asynchronous task module. The system is designed as a distributed system with Celery task queue as the core, with multiple application server nodes as task producers and multiple scoring server nodes as task consumers. When the audio file of the user's answer is uploaded, the application server node will create an asynchronous scoring task and put it into the message middleware. Scoring nodes monitor the message middleware to obtain the task, and store the result to the database after the scoring process is finished. The development of the system follows the principle of separation of logic and data. The application server is developed based on the Flask framework. MongoDB is used as the database. Sessions are

stored centrally in Redis, which solves the problem of session consistency. Also, Redis is used as the cache database to improve system performance, and distributed locks are implemented to achieve synchronization of multiple application server nodes to avoid repeated execution of tasks. The system uses Nginx to achieve load balancing, uses Docker and Jenkins to realise rapid deployment and continuous integration features. The browser-side pages are developed using Vue.js and compiled into a single-page application for deployment, which improves the client-side response speed.

The expressive ability evaluation system designed and implemented in this thesis meets the expected performance requirements, and provides a good support for the quick expressive ability test method. The system can reduce the capital, site, and labor costs in the expressive ability test, improve the test efficiency and test result's objectivity. It also brings a more convenient test experience to users. It can be used to provide reference for recruitment and to provide a evaluation or exercise platform for people who want to improve their expressive abilities.

Keywords: Expressive Ability, Test System, Distributed System, Task Queue

目录

表 目 录 ····· **x**

图 目 录 ····· **xii**

第一章 绪论 ····· **1**

1.1 项目背景与意义 ····· 1

1.2 国内外研究现状 ····· 2

1.3 本文主要工作 ····· 3

1.4 本文组织结构 ····· 4

第二章 相关技术综述 ····· **5**

2.1 系统架构相关技术 ····· 5

2.1.1 分布式系统与任务队列 ····· 5

2.1.2 RESTful 架构 ····· 6

2.2 服务端相关技术 ····· 8

2.2.1 Flask 框架 ····· 8

2.2.2 MongoDB ····· 9

2.2.3 Redis ····· 10

2.3 客户端相关技术 ····· 10

2.3.1 Vue 框架 ····· 11

2.3.2 微信小程序 ····· 12

2.4 系统部署和 DevOps 相关技术 ····· 12

2.4.1 Nginx ····· 12

2.4.2 Docker ····· 12

2.4.3 Jenkins ····· 13

2.5 本章小结 ····· 14

第三章 系统需求分析与概要设计 ····· 15
3.1 项目整体概述 ····· 15
3.2 系统需求分析 ····· 16
3.2.1 系统涉众分析 ····· 17
3.2.2 功能性需求 ····· 17
3.2.3 用例描述 ····· 18
3.2.4 非功能性需求 ····· 23
3.3 系统总体设计 ····· 24
3.3.1 系统架构设计 ····· 24
3.3.2 系统模块及职责划分 ····· 26
3.3.3 4+1 视图 ····· 27
3.4 持久化数据模型设计 ····· 32
3.5 本章小结 ····· 35

第四章 系统详细设计与实现 ····· 37
4.1 权限管理模块 ····· 37
4.1.1 用户识别与访问权限处理 ····· 37
4.1.2 用户评测权限的管理 ····· 38
4.1.3 STS 凭证管理 ····· 39
4.2 评测模块 ····· 40
4.2.1 测试准备流程 ····· 40
4.2.2 正式测试流程 ····· 41
4.3 试题管理模块 ····· 44
4.3.1 试题推荐子模块 ····· 44
4.3.2 业务子模块 ····· 47
4.4 异步任务模块 ····· 48
4.4.1 异步评分服务 ····· 48
4.4.2 定时任务调度服务 ····· 52
4.5 系统部署和可用性提升 ····· 53
4.5.1 使用 Docker 和 Jenkins 进行系统部署与持续集成 ····· 53
4.5.2 提高可用性 ····· 54
4.6 本章小结 ····· 56

第五章 系统测试和实验评估 ····· **57**
5.1 系统测试环境 ····· 57
5.2 系统功能测试 ····· 58
5.2.1 权限管理模块功能测试 ····· 58
5.2.2 评测模块功能测试 ····· 60
5.2.3 试题管理模块功能测试 ····· 61
5.2.4 异步任务模块功能测试 ····· 62
5.2.5 自动部署功能测试 ····· 63
5.3 系统性能测试 ····· 64
5.3.1 系统首页压测 ····· 64
5.3.2 系统接口响应速度压测 ····· 65
5.4 系统实验评估 ····· 66
5.5 本章小节 ····· 67

第六章 总结和展望 ····· **69**
6.1 总结 ····· 69
6.2 展望 ····· 70

参考文献 ····· **71**

简历与科研成果 ····· **75**

致谢 ····· **77**

版权与原创性说明 ····· **79**

表 目 录

3.1 系统涉众分析结果 …… 16
3.2 功能需求列表 …… 17
3.3 系统用例表 …… 18
3.4 邀请码管理用例描述 …… 19
3.5 试题管理用例描述 …… 20
3.6 定时任务管理用例描述 …… 20
3.7 评测功能用例描述 …… 21
3.8 试题反馈用例描述 …… 22
3.9 成绩查询用例描述 …… 22
3.10 定时任务执行询用例描述 …… 22
3.11 非功能性需求列表 …… 23
3.12 User Collection 文档结构详述 …… 32
3.13 Invitation Collection 文档结构详述 …… 33
3.14 PaperTemplate Collection 文档结构详述 …… 33
3.15 Question Collection 文档结构详述 …… 34
3.16 CurrentTest Collection 文档结构详述 …… 34
3.17 CurrentTest 之内嵌题目信息结构详述 …… 35

4.1 评测模块接口设计表 …… 42
4.2 基于用户的协同过滤 …… 45

5.1 测试环境说明 …… 57
5.2 权限控制功能测试用例 …… 58
5.3 STS 凭证管理功能测试用例 …… 59
5.4 邀请码功能测试用例 …… 59
5.5 评测功能测试用例 …… 60
5.6 试题管理模块测试用例 …… 61
5.7 异步任务模块测试用例 …… 63

5.8 自动部署功能测试 ····· 63
5.9 系统首页框架页面压测结果 ····· 64
5.10 系统首页全部资源压测结果 ····· 65
5.11 评测功能接口压测结果 ····· 66
5.12 评测接口压测 - 服务器负载 ····· 66
5.13 实验评估 - 服务器负载 ····· 67

插图

2.1 使用 Celery 部署应用的结构图 …… 6
2.2 Vue 结构示意图 …… 11

3.1 系统用例图 …… 19
3.2 系统架构图 …… 25
3.3 系统模块划分 …… 26
3.4 系统逻辑视图 …… 28
3.5 系统开发视图 …… 29
3.6 系统进程视图 …… 30
3.7 系统物理视图 …… 31

4.1 通过 token 识别用户 …… 37
4.2 权限控制装饰器的定义与使用方式 …… 38
4.3 邀请码管理页面实现效果 …… 39
4.4 获取语音预测试结果关键代码 …… 41
4.5 评测准备流程 …… 41
4.6 评测模块在系统中的位置 …… 42
4.7 评测初始化伪代码 …… 43
4.8 兴趣反馈按钮实现效果 …… 45
4.9 计算用户兴趣相似度矩阵核心代码 …… 46
4.10 试题管理首页实现效果 …… 47
4.11 试题添加页面实现效果 …… 48
4.12 评测服务数据流图 …… 49
4.13 异步评分任务创建接口 …… 50
4.14 使用 FFmpeg 在内存中完成音频文件流式转换 …… 52
4.15 定时任务管理界面实现效果 …… 53
4.16 定时任务 Redis 锁装饰器 …… 54
4.17 开机时注册为 Jenkins 从节点 …… 55

5.1 Flower 部分信息示例 ····· 62
5.2 实验评估结果统计 ····· 67

第一章　绪论

1.1　项目背景与意义

表达能力是一种非常重要的技能，优秀的表达能力能够让人们在生活、工作和学习等多方面建立优势。口语表达是日常生活中人与人交流最常见、最直接的交流方式。人们通过口耳相传，将高维度的思维映射到低维度的语言文字，然后再重新构建成高维度的思维。在这个过程中，清晰、明确的语言表达可以减少信息的丢失并提高交流效率。社会生活中各种企事业单位的招聘及招生工作也体现了对这种能力的要求。但目前常见第二语言水平测试的测试内容主要有听力、阅读与书写等内容，通常要求考生以书面作答，语法和词汇内容的考察仍占有较大比重。普通话水平考试虽以朗读和自由交谈为主要考察形式，但主要以考察考生运用普通话口语进行表达过程中的语音、词汇、语法规范程度为目的。且普通话水平考试的组织与判卷仍由人工完成，计算机技术的应用仍比较初级。

计算机技术在各类测试中的应用让人力物力成本得以大量节省，并为组织更大规模的考试提供了可行性。最初计算机在考试系统中只是负责考务相关信息的管理，然后试题被搬到屏幕，考生可以直接在相应的考试软件系统进行考试，选择题可以直接由计算机进行评分，但其他题目仍需要考生进行文本输入，而后由评分人员进行评分，将结果录入计算机系统。将计算机技术应用于语言测试也有一定的发展历史，且越来越多的语言类测试开始采用计算机技术。如今网络应用非常普及，统计学和机器学习也得到了大规模发展应用，语音识别技术得到了较好的发展，这给创建基于网络和机器学习的个性化表达力测试软件系统提供了所必需的条件。

快速表达力测试方法是针对针对上述情况出现的一种新型测试方法，其设计之初就定位为采用计算机软件和网络技术实现表达能力的自动化量化。快速表达力测试要求考生以口语语音回答问题，通过朗读考察其发音质量，通过转述考察其主旨与细节把控能力，通过自由表述考察其语言逻辑与组织能力，并综合考虑口语流畅性因素 [1] 对整体表达能力进行量化。快速表达力测试的考察内容弱化了词汇与语法知识，测试形式更贴近实际应用场景，弥补了传统语言类测试无法较好题型表达能力的不足。但快速表达力测试方法尚未有配套的软件系统为其提供支持，设计并实现与之相匹配的测试系统，可以实现表达能力量化流程的自动化，增强评分结果的客观性，同时测试的计算机化与网络化可

以大幅降低测试的资金、场地和人力成本，让组织大规模的测试成为可能，也能为考生提供更加便捷的考试体验。

1.2　国内外研究现状

（1）表达能力相关测试的发展情况

目前与表达能力相关的常见测试多为语言类测试，如汉语水平考试（HSK），普通话水平考试（PSC），各种外语测试等。其中汉语水平考试与多种外语测试均是面向母语非本语言的学习者而设立的第二语言能力标准化考试，考试形式笔试为主，对应的口语考试为可选考试。以汉语水平考试为例，其笔试题目的测试形式包括听力、阅读、书写方式等，侧重于对词汇、语法以及文字书写能力的考察 [2]。这些测试项目可以在一定程度上体现书面表达能力。汉语水平口语考试（HSKK）和普通话水平测试采用口语回答的测试形式。其中汉语水平考试题目类型包括听一段语料然后回答问题以及选定议题发表观点等。普通话水平测试题目类型以朗读和自由交谈为主，更侧重普通话口语的语音、词汇和语法规范程度的考察。不同的口语表达能力可以对 HSKK 和 PSC 的成绩产生影响，但这两种测试并不直接以表达能力为测试目标，难以直观反映被测试者的口语表达能力。

（2）计算机技术在测试领域的研究与应用

目前计算机技术在各类测试领域的应用已经广泛被业界接受，计算机的应用主要体现在计算机辅助考试和计算机化考试。计算机辅助考试即计算机参与考务辅助管理和辅助评卷。考务辅助管理即计算机协助或代替人工完成如考试报名、考生信息存储与管理、考卷信息存储与分发、考试监控和成绩查询等任务。辅助判卷常见的是基于红外感光或机器视觉技术的答题卡识别。而计算机化考试则需要提供在线考试界面，以图片、文字、音视频等方式将题目呈现给考试，以按钮点击和文本输入的方式进行交互，获取考生的答案，并将答案实时提交或汇总统一提交至系统的评分服务进行评分。

计算机技术在测试领域的应用让人力物力成本得以大量节省，让组织更大规模的考试成为可能。计算机化测试相较于计算机辅助测试参与程度更高，提供了更高的便利性。

早期的计算机考试系统仅实现了将笔试题目搬移到电脑屏幕，在组织上仍高度复制了笔试的管理模式，如考试需要按场次组织，为防止试题泄漏需要指定考试时间等，测试题库机制的引入在一定程度上解决了这些问题。并且随着网络技术的发展，出现了许多基于网络题库的计算机考试系统 [3]，和各种基于

题库的组卷算法 [4]。这类系统可以方便地对题库进行扩充与修改，提高了系统的灵活性。

伴随着计算机技术的发展与电化评测技术研究的深入，国外学者提出了一些计算机化自适应测试（CAT）方法与应用 [5]。CAT 的基本思想是由计算机根据被测试者的能力自动选择与其能力相适合的题目，并根据被测试者的反馈进行能力评估。与传统的基于计算机的测试（CBT）相比，这种测试方法更接近于人工测试过程中根据被测试者表现进行选题的特性，通常具有更快的测试效率以及准确度。

将计算机技术应用于语言测试领域也有较长的发展历史，由基于计算机的语言测试到计算机化适应性语言测试同样被认为是计算机化语言测试的发展趋势 [6]。有学者认为，语言测试只有借助现代技术，特别是计算机和网络技术，才能真正实现语言测试的科学化和人性化 [7]。目前越来越多的测试开始采用网络机考的方式进行，如 TOEFL 考试在 2005 年在部分国家实现了网络考试（iBT），并逐渐向更多国家推广，HSK 和 PSC 也实现了网络考试。得益于智能语音识别技术和 NLP 技术的发展，出现了用于自动语言评分的特征提取方法 [8] 以及基于可靠性特征加权的自动作文评分方法 [9]，计算机进一步参与到语言测试的评分环节中。

1.3　本文主要工作

针对以上研究现状和表达力测试中存在的问题，本文基于网络和计算机软件技术，为快速表达力测试方法设计并实现了与之匹配的表达力评测系统。在本系统设计的测试流程中，考生通过网络登录测试系统，使用普通话口语回答问题，测试系统对考生的作答进行录音、存储与分析工作，最后将成绩展示给考生。考察内容从发音质量、主旨与细节表述能力、语言逻辑与结构等方面进行。系统主要由四个模块满足表达力测试系统的功能要求。

（1）权限管理模块：考生需要通过邀请码获取参与表达能力测试的权限，登录系统后参加测试，并可对自身信息进行维护。首先，交互模块需要提供完整的注册登录与用户访问权限控制功能，管理员需要具备考生权限的管理功能。其次，测试过程中涉及到外部服务的调用，由于前后端分离开发，前端需要从后端进行凭证获取等操作，该部分同样由权限管理模块进行处理。

（2）评测模块：该模块是负责评测流程的核心模块，为考生进行表达能力测试以及练习提供支撑。该模块应能正确地将题目呈现给用户，引导用户使用语音作答并采集用户作答音频，传输给异步任务模块进行评分。之后，该模块应正确地将异步任务模块的答案分析结果展示给用户。

（3）试题管理模块：该模块主要负责两个工作，一是为评测模块提供题目的推荐选取功能，二是为管理员对试题的管理功能提供业务支撑。

（4）异步任务模块：异步任务模块主要完成异步评分工作和定时任务的管理工作等。异步评分是整个评测流程的重要一环，使用智能语音技术对用户作答音频进行识别，并利用专家规则对题目评分。确保异步评分任务的顺利下发与评分节点的稳定运行才能保证系统的可用性。对于异步评分功能，本文不涉及具体的评分规则和评分算法设计，只对异步评分子模块的架构与调用方式等进行设计与实现。具体评分规则由面向表达力评测的题目建设与分析系统完成。定时任务如系统数据库的冷热数据分区存储，无效数据的清理工作，以及题目评分优化脚本的触发等也是维持系统稳定运行不可或缺的任务。

除功能要求外，该表达力测试系统应具有一定的扩展弹性，完成多端适配工作，至少提供网页和微信小程序两种接入方式。应减少用户错误操作的可能，对于未能避免的用户错误操作，应保证系统可用，并给用户恰当的提示信息；对于因网络故障应有相应机制提供重连后继续测试的能力；对于系统自身错误或调用其他服务引起的错误，应有错误记录与追踪机制。由于快速表达力测试以快速为目标之一，为了提高用户体验，系统需要有一定的并发能力，不但需要保证接口的响应速度，也应减少异步评分过程中用户的等待时间。

1.4　本文组织结构

第一章为引言。介绍了本文背景，简述了计算机技术在各测试领域尤其是语言测试领域的应用和国内外研究现状，并对本文的主要工作进行了描述。

第二章为相关技术综述。介绍了本文使用的基于任务队列的分布式软件结构，以及 Resuful 架构、Flask、Vue 框架，以及 Docker 等理论与技术。

第三章为系统的需求分析与概要设计。首先对系统做了整体概述，然后以用例图的方式介绍各个模块的具体需求，对系统的整体架构与部署方式做了阐述，最后介绍了数据库实体的设计。

第四章为系统的详细设计与实现。在第三章的基础上，先以模块为单位，以顺序图、代码和系统界面展示等方式详细介绍各模块的实现细节，然后介绍了系统在错误追踪和部署、代码集成等方面的实现细节。

第五章为系统测试和实验评估。该章首先介绍了系统的测试环境，然后对系统各模块进行了功能测试，并对评测功能相关接口进行了性能测试，最后对系统进行了实验评估。

第六章为总结与展望。对论文所做工作进行了总结，分析了工作中存在的问题，展望了表达力测试系统未来的完善与发展方向。

第二章　相关技术综述

2.1　系统架构相关技术

2.1.1　分布式系统与任务队列

分布式系统由分布在不同计算机上的子系统（模块）组成整个系统对外提供服务，各个子系统之间通过网络进行通信，为了完成共同的任务而协调工作，对于系统的用户来说好像是一台计算机 [10]。分布式系统的出现最初是为了使用普通的机器以低廉的价格完成单个计算机无法完成的计算或存储任务。所以分布式系统天生具有良好的横向扩展能力，这意味着负载升高时，我们可添加更多计算机节点（水平扩展）运行计算任务，而不是依赖于升级计算机的硬件配置（垂直扩展）[11] 。试图靠提升硬件性能来提高系统性能会很容易就触碰到硬件的性能瓶颈，但一个设计良好的分布式系统，可以根据需求对节点数量进行缩放，由于不同计算机间使用网络进行通信，这使得空间的约束被打破，不同计算机不必在同一机房，甚至可以部署在不同城市，购买自不同云服务器供应商，整个系统性能瓶颈的高度远非升级硬件所能比拟。

分布式系统诞生之初主要用来实现分布式计算和分布式存储任务 [12]。随着技术的发展与迭代，分布式系统也用于分布式应用与服务、分布式静态资源等更广泛的任务。构建分布式应用与服务，首先需要根据业务特点对应用和服务进行分层与分割，然后将分割后的模块进行分布式部署，以此提高系统扩展弹性，提高系统并发能力。同时，功能模块的划分让服务的复用成为可能，易于进行业务扩展。分布式静态资源则是对网站的 js、css 及图片等资源进行多 ISP 多地区的分布式部署，加快浏览器访问资源的速度，减少应用服务器的负载压力，提高访问速度。

为了应对并发需求的逐渐增大，许多应用通过进行功能分解实现分布式应用以提高部署规模，提高任务的处理能力，这种功能分解推动了作业管理系统的需求 [13]。在分布式网络应用中，将网络技术与消息队列技术相结合, 可有效地解决不同应用系统之间的异步任务交互问题。通过消息队列提供的异步交互机制可实现网络应用各系统或组件间的同步，利用网络应用技术的平台通用性可有效扩大异步消息的使用范围，为基于消息队列的跨平台异步交互提供有效解决方案 [14]。

本系统的 Web 应用服务器和评分服务器可能面临不同的负载压力，需要有

不同的部署规模，单纯靠提升硬件配置难以满足性能需求。为了具有较好的扩展弹性，减少各模块的耦合，本文将系统设计为分布式网络应用，实现 Web 应用服务端和评分服务的单独部署，并通过任务队列进行交互。本系统使用 Python 进行开发，因此在任务队列工具的选取上，选择了 Python 生态中常用的 Celery 任务队列。

Celery 同样基于 Python 开发，并被设计为分布式任务队列框架。它侧重于任务实时操作，但对调度的支持也很好，可方便地完成异步任务和定时任务的调度。Celery 的架构是比较典型的生产者-消费者模型，主要包括消息中间件（Broker）、任务执行单元（Worker）和结果存储（Backend）三部分组成。使用 Celery 进行部署的应用结构如图 2.1所示。

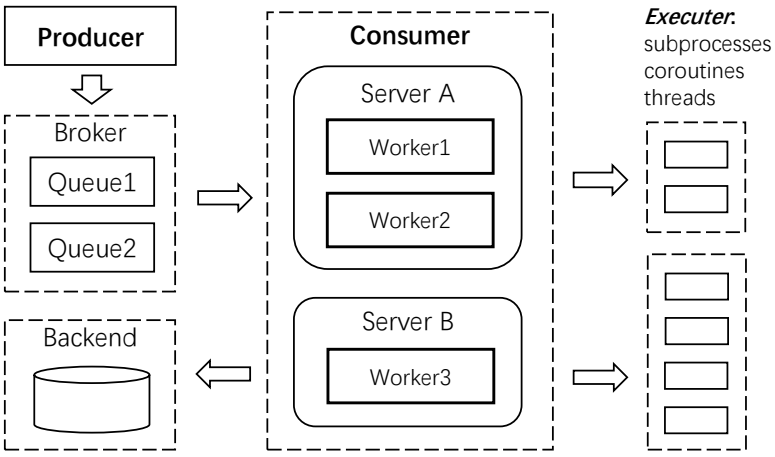

图 2.1: 使用 Celery 部署应用的结构图

Celery 的使用非常灵活，它可以使用 Redis、RabbitMQ 等多种消息中间件作为 broker，且用户可根据需要针对 Celery 的大多数组件进行扩展或定制。同时，Celery 具有高效与稳定的特点，一个单进程的 Celery 实例每分钟可完成上百万个任务的处理，若任务执行失败，Celery 会自动重新执行任务。这些灵活、高效与可靠的优点让 Celery 被广泛应用于各种使用场景。

2.1.2　RESTful 架构

REST 是 Representational state transfer（表述性状态转移）的缩写，由 Roy T. Fielding 在 2000 年提出 [15]。它是一种提供网络服务的软件体系结构样式，在 HTTP 的基础上为不同软件或系统在网络中互相传递信息建立了一种方便使用的约束与属性。

在 RESTful 架构风格中，每个资源都由 URI 来指定，客户端可以对资源进行增删改查等操作，这与 HTTP 的 POST、DELETE、PUT 及 GET 等方法相对

应。RESTful 架构风格有 6 个主要的限制：

（1）客户端-服务器（Client-Server）：客户端与服务器彼此独立，可单独进行开发，客户端只知道资源的 URI，不对资源数据进行实质性直接操作，同时服务器不负责任何与用户的界面交互。这种逻辑上的分离降低了客户端和服务器的代码复杂度，有利于系统的开发与维护，同时这种分离为服务的分布式部署提供了天然的便利性，对不同终端设备，只需单独为其适配用户界面，而无需更改服务器。

（2）无状态（Stateless）：客户端每次发送的请求需要包含必须的状态信息，服务端不保存客户端的状态信息，但可以将会话信息传递给其他服务（如数据库服务）从而实现用户的识别与短时的状态保存。无状态的特点可以实现数据与逻辑的分离，让不同的服务器处理一系列请求中的不同请求，从而提高服务器的扩展性。

（3）可缓存（Cacheable）：响应必须定义自身是否可以被缓存。不可缓存的数据通常具有时效性或设计资源的修改操作，对于可缓存的数据，客户端和中间的通讯传递者可将数据缓存起来从而减少对服务器的访问，这可以提高系统性能并进一步提高扩展灵活性。

（4）统一接口（Uniform interface）：统一接口简化了系统架构，降低了耦合性，是 RESTful 系统设计的基本出发点。这一限制具体包括四个方面：请求中需要包含资源的标识符（如 URI）；资源需要通过标识符来进行操作；消息应具有自我描述性；使用超媒体作为应用程序状态引擎。请求的载体通常可以是 JSON、HTML 或 XML 等。资源需要通过标识符进行操作是指客户端要对资源进行操作时，只需提供资源的标识符和元数据，即满足操作资源所需的所有信息。消息的自我描述性是指每个信息都要包含足够的信息来描述如何处理这个消息，如 HTTP 请求与响应中的 Content-Type 即可告知对方将消息视作何种类型进行处理。使用超媒体作为应用程序状态引擎是指客户端对资源进行访问后，服务器响应信息中，可以通过查询 header 获取请求状态，在消息体中提供可用的操作。

（5）分层系统（Layered system）：通过约束组件行为使组件不能看到不与其直接交互的其他层。中间层可以通过负载均衡和共享缓存等机制提高系统的可扩展性与健壮性。

（6）按需代码（Code on demand）：按需代码使可选约束。允许服务器发送脚本等代码给客户端，以扩展客户端的功能。

与 SOAP 和 XML-RPC 这两种主流 Web 服务实现方案相比，RESTful API 更加简洁 [16]，且可以与前后端分离的思想有较好的匹配。表达力测试系统需

要适配多终端设备，采用了客户端-服务器的模式进行设计。由客户端负责与用户的交互，服务端只负责业务逻辑的处理与数据的操作，二者间使用约定好的RESTful 接口进行通信，从而达到一个服务端为多种客户端应用提供服务的目的。同时，由于客户端与服务端实现了解耦，二者可单独进行开发，最后再进行联调。在客户端与服务器之间还可以使用负载均衡和内容分发网络作为中间层以提高系统性能。

2.2　服务端相关技术

2.2.1　Flask 框架

Flask 是 Python 生态中的轻量级 Web 框架，基于 Werkzeug [1] WSGI 工具箱和 Jinja2 [2]模板引擎。WSGI(Python Web Server Gateway Interface) 意即 Python Web 服务器网关接口，是基于 CGI(Common Gateway Interface) 标准为 Python 语言定义的 Web 服务器和 Web 应用程序或框架之间的简单而通用的一套接口标准协议。所谓网关，即客户端请求首先到达的地方，然后请求消息依据路由被转发处理。而对于服务端发送的相应消息，也总是先通过网关层，然后再转发至客户端。由于 Python 当前拥有众多种类的 Web 应用程序框架，在没有一个便准的协议时，用户对不同 Web 应用框架的选择将限制他们对可用 Web 服务器的选择，而对 Web 服务器的选择也将反过来限制用户对 Web 应用框架的选择。WSGI 就是为了解决这一问题而出现，作为连接应用和网关的一个接口，接受网关传来的请求信息，然后调用后台应用进行处理，最后将处理结果重新交回给网关。通过建立一套标准的接口协议，可以保证不同 Web 服务器和不同的 Web 应用框架之间可以互相通信。

与操作系统内核架构的“微内核”类似，Flask 被称为“微框架”（Micro Framework），因为其核心简单小巧，其他众多功能可以使用各种扩展来灵活增加。这种设计使得依赖 Flask 搭建起来服务各功能部件直接耦合度较低，可以以较小的代价更改替换部分功能部件。Flask 提供了建议，但不强制执行任何依赖关系或项目布局。由开发人员决定他们想使用的工具和库 [17]。

Flask 处理请求的基本流程有 5 个阶段组成：1. 创建上下文；2. 把上下文入栈；3. 分发请求进行处理；4. 上下文出栈；5. 返回结果。所谓上下文，即语义、语境，对应到 Web 应用程序就是程序中所需要共享的信息，及程序运行交互过程中所需要保持或传递的信息。Flask 中有应用上下文（App Context）和请求上下文（Request Context）两种上下文。其中应用上下文保存着应用运行或交互中

[1]Werkzeug project, URL: https://palletsprojects.com/p/werkzeug/

[2]Ronacher A. jinja2. URL: http://jinja. pocoo. org, 2014.

需要保持和传递的信息，如应用名称、应用的数据库配置、应用路由规则、视图函数等。请求上下文处理请求过程中所需要保持和传递的信息，如请求的 uri，请求的方法（method），以及所附带的参数等。请求上下文的生命周期在一次请求期间，请求处理完成后其生命周期就结束了。而应用上下文的生命周期是整个应用的运行周期，应用程序退出后应用上下文的生命周期才结束。

在路由处理上，Flask 使用 Werkzeug 路由系统，自动根据复杂度来为路由排序 [3]。这种机制解决了应用被切分成多个模块时，基于路由的装饰器以乱序触发的问题。此外，Werkzeug 路由系统会尝试确保 URL 唯一，当如果路由不明确时，它会将模糊路由自动重定向到规范的 URL，增加了路由系统的稳健性。

2.2.2 MongoDB

MongoDB 是一个使用非常广泛的开源非关系型数据库，它使用 C++ 语言开发而来，并基于分布式文件系统进行存储，具有高性能和模式自由的特点。所谓模式自由，是指使用存储时无需告知数据库以何种结构存储数据，如果必要，用户完全可以把不同结构的数据存储在一起。这也是非关系型数据库区别于关系型数据库的一个重要特征 [18]。MongoDB 面向文档和集合存储，将数据以 BSON（Binary JSON）键值对的形式分组存储于集合（Collection）中 [19]，键为字符串类型，用来唯一标识一个文档，值可以是多种数据类型，且可以有复杂的嵌套结构。MongDB 的这种特性让其可以很容易地将大多数编程语言的对象结构直接映射到 BSON 形式，而无需任何转换器，从而方便地进行对象类型数据的存储。而且，即使在有大量数据负载的情况下，其仍能保持很高的灵活性、可伸缩性以及查询速度 [20]。

基于 MongoDB 进行数据持久化时，如需给对象增加新属性，无需像传统关系型数据库那样首先对数据库设计进行变更，然后再对数据进行迁移，只需要在应用层做必要的改动即可，这让其可以非常方便地处理半结构化数据，也显著降低了系统的运维成本。在索引机制上，MongoDB 支持完全索引，可在包括内部对象的任意属性上建立索引，支持丰富的 JSON [21] 形式的查询表达式。其分片功能可将数据分散存储到 MongoDB 集群中，并利用数据在结点间自动迁移的特性使各节点的数据量保持均衡，这给 MongoDB 带来高性能的同时也简化了运维和部署操作。这些高性能和简单易用的特性，让开发人员能够把精力集中在核心功能开发上，非常适合网站应用开发与实时数据处理，在内容管理系统、移动设备、游戏和归档等存储场景中均有典型应用 [22]。

本系统中测试相关数据具有多层嵌套的结构，同时项目开发中常面临较多

[3] Design Decisions in Flask, URL: https://flask.palletsprojects.com/en/1.1.x/design/

数据与功能需求的更改，故本文选用 MongoDB 数据库进行数据持久化，相较于关系型数据库可以提供更好的灵活性与存取便捷性。本系统使用 Python 进行开发，为了进一步简化数据存取代码的编写，本文使用 MongoEngine 作为 ODM 层。通过继承 MongoEngine 定义的基础类，可以便捷地建立相关数据模型，还可通过在类中添加类方法，实现自定义的数据处理逻辑。同时它简化了数据库交互，使用其进行数据操作时无需关注 JSON 查询语句细节，直接以 Python 代码即可构建数据库查询语句。

2.2.3 Redis

Redis 是目前主流的缓存中间件之一，是一种是一个基于内存的 Key-value 存储数据库。本系统使用 Redis 作为分布式任务队列的后端，并用 Redis 来缓存用户会话和热点数据。与 Memcached 等缓存中间件相比，Redis 支持包括 String、Hash、List、Set、ZSet 等在内的多种数据类型，使用这些数据类型，可以非常方便地实现高速缓存与消息队列等应用逻辑。

Redis 客户端对服务端的每次调用都需要经历命令的发送、执行、返回结果三个过程。主线程执行 I/O 多路复用程序文件事件分派器以及事件处理器，虽然请求可能同时到达，但这些请求并不会立刻执行，而是进入队列后逐个执行，这种单进程单线程模型简化了 Redis 架构设计，同时避免了多线程的上下文切换和同步开销。Redis 还运行多个后端线程执行清理操作，如过期数据的清理和文件描述符的关闭等。

Redis 还提供了分布式集群和节点间同步的特性，便于水平扩展，哨兵模式可以自主进行故障转移，提高了集群可用性。Redis 支持两种数据持久化机制：RDB 和 AOF。RDB 方式是通过创建子进程对父进程内存生成快照的全量数据的二进制备份，恢复数据时速度较快，但可能丢失最新的数据；AOF 方式则是以日志文件志记所执行的指令，因此可以包含最新的数据，在必要时可通过指令重放来进行恢复，但随着使用日志文件会占用大量空间。在实际应用中可以将二者结合起来，以定期快照的方式配合 AOF 志记增量数据，在保证数据完整的情况下使得 AOF 日志文件的大小得到控制。

2.3 客户端相关技术

本系统的主要实现面向普通用户的评测功能和面向管理员的相关管理功能。为方便普通用户使用系统服务，系统使用 PC 端浏览器和移动端的微信小程序作为客户端进行开发，实现评测功能的多终端支持。下面本文对客户端用到的相关技术做简要介绍。

2.3.1　Vue 框架

Vue.js 是前 Angular 开发成员尤雨溪开发一款渐进式前端框架。Vue 在设计上不完全遵循 MVVM 模式，仅实现 State 和 View 两部分，其核心强调的是通过状态与界面的映射来简化 Web 开发，而不重视代码的组织结构。所谓渐进式，是指框架的主张较少，对使用者的强制性要求较少，使用者可完全使用 Vue 开发完成开发，也可以仅使用其中的部分组件进行开发。这些特性不仅使其简单易用，还便于与第三方库或者已有项目进行整合。

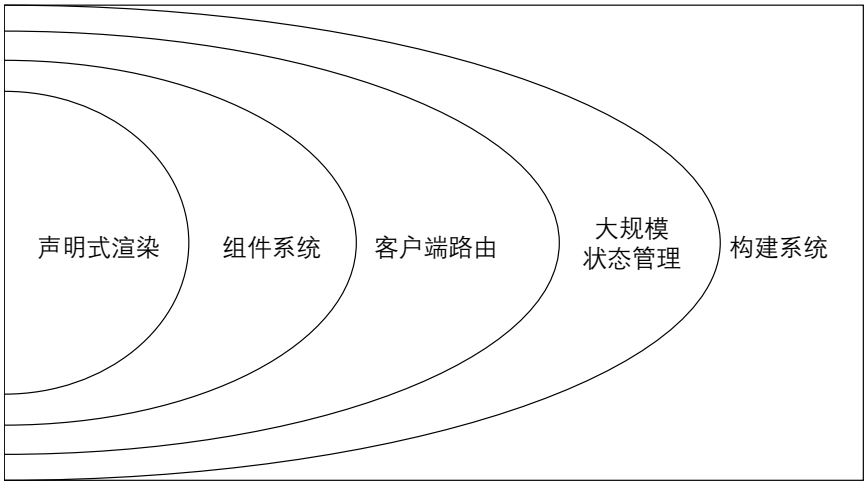

图 2.2: Vue 结构示意图

图 2.2是 Vue 的结构示意图 [23]。Vue 的核心是声明式渲染，利用 Virtual DOM [24] 实现了应用状态与 DOM 的状态进行绑定，当应用数据变化时可自动更新 DOM 状态，避免了手动操作 DOM 带来的不便，同时当用户进行输入操作更新视图后应用的数据也得到更新。组件系统是目前主流前端框架常见的解决方案，众多组件组织为树状结构，从而构成了整个页面。Vue 组件之间进行通信的方式可概括为“属性传入，事件传出”，即父组件使用属性向子组件传递数据，子组件则将数据的变化注册为事件，由父组件进行事件的捕捉从而获得需要传递的数据。客户端路由使得前端众多页面片段可作为一个单页应用提供服务，页面片段间的跳转无需后端渲染，从而提高了交互时的响应速度。大规模状态管理则为复杂应用的多组件通信和状态共享提供了解决方案。构建系统则类似于传统语言中的编译程序，Vue 使用基于 Webpack [4]的 vue-cli 作为构建工具，让开发人员能够方便地实现模块化开发。

[4]Webpack Official Website, URL: https://webpack.js.org

2.3.2　微信小程序

微信小程序是腾讯推出的一种运行在微信平台上的应用程序，主要提供给企业、政府、媒体、其他组织或个人的开发者在微信平台上提供服务。小程序本身是一种 JS 应用程序，其框架包含两部分：View 视图层与 App Service 逻辑层。视图层负责页面的渲染，逻辑层负责逻辑与数据处理，以及接口调用等，二者通过 JSBridage 进行通信，逻辑层的数据变化将触发视图层的自动更新，同时视图层产生的事件也将触发逻辑层数据的更新或相应操作的执行。

用户使用小程序时，只需在微信中搜索小程序或扫描小程序二维码即可进入小程序，然后微信客户端从其平台下载所需的程序文件，由于需要下载的文件体积非常小，通常能带给用户“不需要下载安装程序”即可使用的便捷使用体验。同时由于微信具有庞大的用户群体，小程序的宣传和推广效果也较好。

2.4　系统部署和 DevOps 相关技术

2.4.1　Nginx

Nginx 是一个使用 C 语言编写的高性能轻量级 HTTP 和反向代理 web 服务器，同时也提供了 IMAP/POP3/SMTP 服务，在 BSD-like 协议下发行。Nginx 以 Master-Worker 的多进程模式工作。程序启动时，Master 进程验证并加载配置文件，初始化需要监听的 socket，然后通过 fork 机制创建多个 worker 进程。Worker 进程通过竞争的方式接受传入请求，获胜进程通过 TCP 三次握手机制建立连接并处理请求。Worker 进程的状态由 master 进程进行监控，如果 worker 进程异常终止，master 进程会自动重启 worker 进程 [25]。

由于 Nginx 对静态文件的处理能力很强，在使用 Nginx 作为反向代理服务器时，通常会将静态文件直接交由 Nginx 处理，仅把动态请求传递给后端应用程序。同时使用 Nginx 的反向代理功能可以十分方便地实现负载均衡，并能够以轮询、设定权重、根据 IP 哈希值分配等多种策略进行请求的分配。Nginx 并发能力非常强，内存开销和 CPU 使用效率表现都很好，而且安装简单，配置灵活，支持热部署，在不间断服务的情况下即可对软件或配置版本进行升级，运行数月也无需重启。由于 Nginx 具有上述众多优点，国内外有大量知名企业选择使用 Nginx 搭建反向代理与负载均衡服务器。本系统同样使用 Nginx 作为网关服务器，并使用其反向代理功能实现负载均衡，以提高系统的并发能力。

2.4.2　Docker

Docker 是一个开源的应用容器引擎，使用 Go 语言进行开发实现，是目前非常流行的 Linux 容器解决方案。其利用 Linux 核心中的资源分离机制（如 cgroups）

和 Linux 核心命名空间来创建独立的容器，实现了进程资源的隔离，又使得进程可以在和主机同一 Linux 实体下运作。这种通过 Linux 进程虚拟隔离底层设备的方式非常轻量，避免了引入虚拟机造成的额外资源消耗 [26]。

Docker 在架构设计上采用了客户端-服务器的模式，使用 Docker daemon 作为服务端接受并处理客户端的请求（创建、运行、分发容器等）。客户端和服务端可使用 socket 或 RESTful API 进行通信，二者无需部署于同一物理机。同时 Docker 客户端提供了简单易用的容器使用接口，让开发者可以方便地将应用与依赖打包为独立的 Docker 镜像，并发布到众多发行版本的 Linux 服务器，从而可以大幅简化部署工作，避免运行环境不一致带来的影响 [27]。

Docker 镜像是应用与依赖的静态表现，应用的运行则需要在基于镜像创建的 Docker 容器中进行，二者的关系类似于面向对象编程中的类与对象的关系。Docker 容器的启停效率非常高，可以很好地支持大规模分布式系统水平扩展的需求。Docker 通过控制组实现资源的可度量及可配额，通过联合文件系统实现移动性 [28]。在实际使用中，Docker 有如下三种典型的应用方式：（1）将 Docker 镜像作为独立部署单元，通过对应用依赖的封装简化环境配置，使生产环境与测试环境保持高度一致性；（2）提供项目镜像构建和快速部署能力，便于快速自动化打包和部署发布；（3）利用轻量化和快速启停的特性提供弹性服务，提高系统可扩展性。

本文所设计实现的表达力测试系统中，服务端与评分服务作为独立的逻辑实体，使用 Dockerfile 分别创建其 Docker 镜像，将其运行环境进行打包，不受服务器初始系统环境的影响。同时利用 Docker 简单易用的命令接口，使用 Jenkins 实现构建部署过程的自动化，大幅提高了运维效率。

2.4.3　Jenkins

持续集成的概念由 Martin Fowler 在 2006 年提出 [29]。其基本思想是对软件项目进行持续化、自动化的编译、打包、测试和发布，尽早发现集成中的问题并得到及时反馈，通过提高自动化程度提高软件开发效率。

Jenkins 是一款以 MIT 许可协议发布的开源的持续集成工具。它支持多种软件配置管理（SCM）工具（Git、Subversion 等），可执行基于 Apache Ant 和 Maven 的构建项目，以及任意的 Shell 脚本，既可以通过 Git 仓库的 Hook 机制被动触发构建过程，也可使用 Cron 机制主动监测代码更新并构建项目。此外 Jenkins 具有丰富的插件支持，可方便地对其功能进行扩展。Jenkins 支持 M/S 工作模式，仅需要配置一台服务器作为 Jenkins 管理端，便可通过多种方式在多台从节点服务器进行软件集成工作。此外，Jenkins 拥有良好的可视化 Web 管理界面，可以大

大提高开发人员的集成工作效率。

本系统使用 GitLab 托管代码，使用 Jenkins 作为持续集成工具，配置构建脚本，以 GitLab 的 hook 机制触发 Jenkins 执行集成任务，通过 shell 脚本和 Docker 容器完成各节点代码的自动化部署工作。

2.5　本章小结

本章主要概括性地介绍了开发本系统所使用的相关技术。首先，系统架构设计上考虑了分布式和 RESTful 架构的思想，采用前后端分离的方式进行开发。其次，系统服务端使用的 Flask 作为主要框架，使用 MongoDB 持久化数据，使用 Redis 作为分布式部署的任务队列，及作为缓存热点数据的场所。第三，在前端界面的开发上，系统主要适配了 PC 浏览器和微信小程序。适配浏览器的前端程序使用 Vue.js 进行开发。最后介绍了系统部署中的可用性优化和 DevOps 措施，使用 Nginx 作为反代服务器，分离动静请求的处理，并使用其负载均衡功能提高可用性。使用 Docker 和 Jenkins 实现持续集成并提高集成工作的效率。

第三章　　系统需求分析与概要设计

3.1　项目整体概述

快速表达力测试方法是专门针对表达能力评估而设计的一种测试方法。与传统语言测试不同，快速表达力测试在设计之初就定位于采用基于网络的软件系统为口语测试及自动化评分提供支持。这让其更侧重于实际应用中表达能力的评估，避开了过度侧重考查词汇和语法知识的缺点。同时计算机和网络技术的应用解决了传统考试组织困难、成本高、效率低以及难以保证结果客观性的问题，让测试方法迁移性更强。与依赖按钮点击和文字输入等交互方式的计算机辅助测试相比，直接以口语语音作答的形式更加方便快捷。

在功能上，本系统除了可以提供表达力评测考试和成绩报告查询服务，还可提供评测练习服务帮助用户进行表达能力的强化训练，通过用户反馈和协同过滤向用户推荐其感兴趣的题目，提高用户的学习体验。用户可以灵活地选用 PC、平板以及手机等多种设备使用浏览器或微信小程序使用评测服务。此外，系统为管理员提供了如权限管理、试题管理以及定时任务管理等辅助系统。系统可预见的主要应用场景有：为招聘相关领域提供初面参考，为有兴趣了解自身表达能力水平的人提供自测平台，为需要提高表达能力的人群提供训练平台，对接其他平台提供商业服务等。根据一次完整测试的流程和管理员执行维护管理任务的需求，本系统可大致划分权限管理模块、评测模块、试题管理模块和异步任务模块四大模块。

权限管理模块为用户身份的识别提供了支持，同时也是用户访问权限控制、评测权限管理以及存储服务凭证管理等功能的基础。其中评测权限的管理使用邀请码机制进行控制。

评测模块作为系统核心模块，主要面向普通考生用户提供表达力评测与练习与成绩查询的功能。该模块对服务器的并发能力与响应时间有一定的需求，可通过分离动静态资源和设置缓存等技术降低对应用服务器的压力。同时，系统采用客户端-服务器的架构进行前后端分离开发，将用户交互的处理置于前端来提高交互响应速度，并能够简化后端服务器的处理逻辑与压力。用户作答产生的音频文件可以由前端直接传输到指定的存储服务，这样可以避免对后端服务器造成过大的带宽压力。

试题管理模块是辅助系统良好运行的重要部分。该模块不仅为评测模块提供评测初始化过程中的题目选取与推荐功能，也为管理员管理题库、修改题目

提供了支持。

异步任务模块同样可分为两部分，一是负责异步评分任务管理的异步评分子模块，二是负责定时任务调度的定时任务子模块。对于异步评分子模块，本文仅讨论其架构的设计、实现与部署方式，所调用的外部评分服务和算法不在本文讨论范围内。但为使读者对系统结构有较清晰的整体认识，本文在此对外部评分服务进行粗略介绍：评分服务由面向表达力评测的题目建设与分析系统提供，以用户作答的音频文件为输入，提取音频和文本特征，并使用这些特征给出得分，最后将特征与分数存入数据库；除提供评分服务外，题目建设与分析系统还负责通过爬虫技术从互联网进行语料的抓取，对语料进行分词后存入数据库，为备用题目构建语料库。为维持系统的良好运行，系统需要运行定时任务进行数据库的临时数据清理、冷热数据分离、触发执行协同过滤算法以及触发评分服务中题目关键词权重的自动优化等定时任务，这些任务均由定时任务子模块负责调度。

由于涉及音频文件的传输和处理，评分模块可能对所部署的主机造成较大的带宽压力。且大量任务时为加快处理速度可采用多线程或协程等方式进行并发的分析和网络请求，对内存和 CPU 有一定的需求。因此，单个评分服务器节点难以满足并发量较高时的处理需求，这要求评分模块需要有较好的快速扩容能力。

为使系统具有较高的性能和可用性，本文使用 Redis 缓存热点数据，使用 Nginx 的反向代理实现应用服务器集群各节点的负载均衡，并使用 Keepalived 实现 Nginx 自身的高可用。为方便错误追踪与解决，系统对各功能所执行的操作均有必要且充分的日志记录。

3.2　系统需求分析

表 3.1: 系统涉众分析结果

涉众名称	**涉众特征与期望**
考生	1. 有量化测试自身的表达能力的需要，希望有一套系统能够方便快捷地完成快速表达力测试，并能够较快地获取考试成绩。 2. 有一定的软件操作能力。
管理员	1. 管理员具备最高的账户管理权限，希望能够使用邀请码机制管理其他用户使用系统的权限。 2. 管理员也是表达能力测试的题库的建设者，希望能够通过系统进行题目设计与投放，并能够查看考生做题状况，可以对题目进行必要的修改维护。 3. 有一定的计算机和软件操作能力。

3.2.1　系统涉众分析

本系统的涉众主要包括考生用户（下称考生或普通用户）、管理员两种类型，各自的特征与期望如表 3.1所示。普通考生具有量化测试自身表达能力的需要，并具备一定的软件操作能力，希望可以通过系统便捷地完成测试，并能够较快拿到结果反馈。管理员可以进行用户信息的管理工作，可以创建并发放邀请码给予考生考试权限，也可以进行题库建设与调整。

3.2.2　功能性需求

根据涉众特征，并结合对涉众群体的实际访谈结果，通过进一步分析与总结得到如表 3.2所示针对系统的主要功能性需求列表。

表 3.2: 功能需求列表

需求 ID	需求名称	需求描述
R1	权限管理	管理员可以通过邀请码的方式赋予用户参加评测考试和评测练习的权限，考生用户可使用管理员发放的邀请码更新自身评测权限。
R2	试题管理	管理员希望可以通过系统界面添加新的题目到题库，可以查看现有题目信息，并可根据考生对题目的反馈进行题目内容的手动调整，还可以删除不想继续使用的题目。
R3	定时任务执行	为保证系统的良好运行，并为其他功能提供支撑，系统需要在预设时间执行定时任务，如设置清理、存档临时数据库的频率，用户相似度的计算，题目自动优化脚本等。
R4	定时任务管理	管理员希望可以根据需要对定时任务的启停与运行频率等进行设定。
R5	评测功能	考生希望可以通过系统顺利进行正式表达能力测试，同时考生希望系统能够提供评测练习功能供其自我练习以提高专项能力。
R6	试题推荐	管理员希望系统可以在用户进行评测练习时代替人工为考生用户推荐题目。
R7	成绩查询	考生希望考试后能够通过系统进行个人成绩报告的查询操作。
R8	试题反馈	为协助管理员进行题目分析与管理，以及为试题推荐提供支撑，考生应能够通过系统进行题目质量和自身对题目兴趣程度的反馈。

在管理员的业务流程中，首先管理员可以进行用户权限管理，通过邀请码的机制控制系统用户是否具有参与评测考试或评测练习的权限，可以一次生成

一个或多个邀请码并导出为 csv 表格文件。其次，管理员也是系统题库的建设与维护者，可以通过系统管理题目，可以向题库中添加新的题目，可以查看已有题目的信息（包括用户对题目质量的反馈），并按需对题目内容进行调整，对于不想继续使用的题目，可以执行删除操作。为了协助自身进行题目质量的分析，管理员希望考生在评测时能够通过系统对题目质量进行反馈。此外，用户进行评测练习时，管理员希望系统能够代替人工为用户推荐其感兴趣的题目，提高用户使用系统的体验。最后，为维护系统稳定并为其他功能提供支撑，系统应能执行定时任务，而管理员应能够对这些定时任务进行管理，比如控制定时任务的启用或关闭，或调整其运行频率等。

在考生用户的业务流程中，由于邀请码机制的限制，考生需要从管理员处获得邀请码才可取得参加评测考试或评测练习的权限（邀请码如何分发到考生手中不在本文讨论范围内）。其次，考生具有权限的情况下应能够正常进行表能能力测试的评测考试或评测练习，在评测练习过程中，用户应能够对所完成的题目通过点击标签按钮的方式进行题目质量的反馈，同时用户可以对题目标记喜欢或不喜欢。用户对题目质量的反馈将用于辅助管理员进行题目调整，而用户的喜好反馈将用于对用户进行评测练习题目的协同推荐。对于评测考试和评测练习，用户均应能够查询自己的成绩报告。同时，用户应能够查看自己的历史成绩，以了解自己评测成绩的变化趋势。

3.2.3　用例描述

本系统用例图如图 3.1所示。包括适用于管理员的试题管理、定时任务管理、权限管理用例，适用于普通考生用户的评测和成绩报告查询用例。同时，由于系统有执行定时任务的需要，本文抽象出时钟作为系统的第三个参与者，其对应的用例为定时任务的执行。

表 3.3: 系统用例表

用例 ID	用例名称	对应需求 ID
UC1	邀请码管理	R1
UC2	试题管理	R2
UC3	定时任务管理	R4
UC4	评测功能	R5
UC5	试题反馈	R8
UC6	成绩查询	R7
UC7	定时任务执行	R3

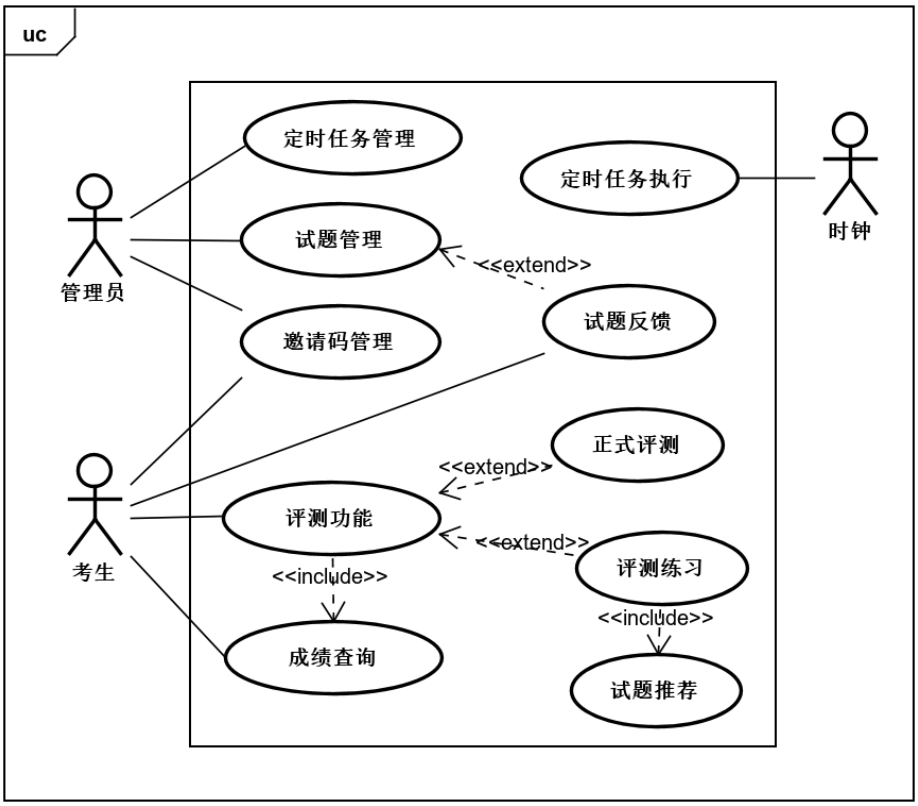

图 3.1: 系统用例图

表 3.3为对应的系统主要用例表，每个用例包括用例编号和用例名称，第三列对应的为功能性需求列表中的需求 ID。以下为重要系统用例的详细描述。

用例 UC1：邀请码管理

表 3.4: 邀请码管理用例描述

ID	UC1
名称	邀请码管理
参与者	管理员、考生
触发条件	用户打开邀请码管理或使用页面
前置条件	用户必须被识别和授权
正常流程	1. 用户完成系统登录； 2. 用户点击系统界面的功能入口进入权限管理页面； 3. 管理员在页面进行邀请码创建或查询等相关操作； 4. 考生在页面填写邀请码以更新做题权限； 5. 若用户确认操作，系统执行对应操作； 6. 若用户取消操作，系统不执行操作。
异常流程	1. 用户尝试使用错误的邀请码更新自身权限； 2. 系统拒绝用户请求。

用例 UC1 描述的是管理员和考生使用系统进行邀请码管理的过程。系统对于评测权限的管理通过邀请码机制完成。管理员可以通过系统完成邀请码的创建与管理，考生则可通过系统使用邀请码更新自身参加评测的权限。无论管理员还是普通考生用户，执行相关操作之前均需登录系统，并被系统识别。若用户尝试使用错误的邀请码，系统将拒绝执行用户请求。

用例 UC2：试题管理

用例 UC2 描述的是管理员使用系统进行试题管理的过程。用户登录系统并被识别为管理员后，可以使用系统进行的题目的查看和增删改查等操作。其中题目的添加又包括从空白模板新增题目与从备用题库添加题目。对于现存的题目与将要添加到题库中的备用题目，管理员均可通过系统进行查询、修改或删除操作。

表 3.5: 试题管理用例描述

ID	UC2
名称	试题管理
参与者	管理员
触发条件	用户打开试题管理页面
前置条件	用户必须被识别和授权
正常流程	1. 用户正常登录系统； 2. 用户点击系统界面的试题管理入口进入试题管理页面； 3. 用户选择从空白添加题目、导入备用题目或查看题目； 4. 若用户选择查看题目，系统展示考生对题目的反馈信息和题目的修改、删除选项； 5. 用户对题目进行添加、查看、修改或删除操作，并对操作进行确认； 6. 系统执行用户请求并返回执行结果状态。

表 3.6: 定时任务管理用例描述

ID	UC3
名称	定时任务管理
参与者	管理员
触发条件	用户打开定时任务管理界面
前置条件	用户必须被识别和授权
正常流程	1. 用户通过系统界面功能入口打开定时任务管理页面； 2. 系统展示可用的定时任务； 3. 用户选择需要管理的定时任务； 4. 用户对选择的定时任务进行配置； 5. 系统执行对定时任务的更改。

用例 UC3：定时任务管理

用例 UC3 描述的是管理员使用系统进行定时任务管理的过程。对于现有的定时任务，管理员可以控制任务的启用或停用。定时任务分为在指定时间运行的简单定时任务与每间隔一定时间即运行一次的 Cron 循环任务，对这两种任务的执行时间或频率，管理员均可进行修改。

用例 UC4：评测功能

用例 UC4 描述了普通考生用户使用系统进行正式评测与评测练习的过程。评测功能是系统为考生用户提供的核心功能。进行正式评测与评测练习之前，用户均需要通过简短的预测试，以确认用户设备正常，测试环境噪声较低，保证测试的正常进行。

表 3.7: 评测功能用例描述

ID	UC4
名称	评测考试
参与者	考生
触发条件	用户点击评测功能入口
前置条件	用户必须被识别和授权
正常流程	1. 用户正常登录系统； 2. 用户点击功能入口进入评测页面； 3. 系统展示预测试题目； 4. 用户通过预测试，选择进入正式评测或评测练习； 5. 系统展示测试题目； 6. 用户作答直至完成所有题目； 7. 系统提示考试完成并进行判卷。

用例 UC5：试题反馈

用例 UC5 描述了考生使用系统进行试题反馈的过程。在评测过程中，考生可以通过点击试题反馈栏中的点赞或点踩按钮对题目质量进行反馈，并可以通过点击喜欢按钮向系统反馈自身对该题目的感兴趣程度。

用例 UC6：成绩查询

用例 UC6 描述了考试使用系统进行成绩查询的过程。成绩查询功能的触发包括用户正常完成评测后自动调整至成绩报告页与用户主动点击成绩查询功能入口两种方式。如果用户未进行过测试导致评测成绩历史为空，则系统处理过程中不应报错，并需要给出提示信息。

表 3.8: 试题反馈用例描述

ID	UC5
名称	试题反馈
参与者	考生
触发条件	用户在评测练习中完成单个题目的练习
前置条件	用户必须被识别和授权
正常流程	1. 用户正常进行评测练习，系统展示题目与试题反馈栏； 2. 用户点击反馈栏按钮对题目质量和兴趣程度进行反馈； 3. 用户选择继续练习或退出。

表 3.9: 成绩查询用例描述

ID	UC6
名称	成绩查询
参与者	考生
触发条件	评测结束自动跳转或用户主动点击功能入口进入成绩查询页面
前置条件	用户必须被识别和授权
正常流程	1. 用户正常完成评测，或用户点击历史成绩入口； 2. 系统跳转至成绩展示页面； 3. 系统显示用户的成绩详情。
异常流程	1. 用户未进行过评测； 2. 系统提示用户未进行过评测，无评测成绩。

用例 UC7：定时任务执行

不同于前述用例，用例 UC7 由系统时钟进行触发。表 3.10对此进行了描述。定时任务需要由相应的任务脚本进行支持，当系统时间到达定时任务设定的时间时，系统需获取任务的元数据，并按元数据执行指定的定时任务。若系统存在多个定时任务执行线程，系统需要保证定时任务不被重复执行。

表 3.10: 定时任务执行询用例描述

ID	UC7
名称	定时任务执行
参与者	系统时钟
触发条件	系统时间到达管理员设定的定时任务执行时间
前置条件	定时任务被启用
正常流程	1. 系统正确获取定时任务； 2. 系统根据定时任务信息正确执行对应的任务。

3.2.4　非功能性需求

为了保证良好的用户体验，必须对表达能力测试系统的稳定性、扩展弹性和性能有一定的要求。本系统的非功能性需求如表 3.11所示。

表 3.11: 非功能性需求列表

需求名称	需求描述
可用性	系统 3 个月内崩溃次数不能超过 1 次，若因意外导致系统崩溃或服务不可用，应在 15 分钟内恢复或启用备用服务器提供服务。
性能	系统前端各界面响应时间 99% 不应高于 1 秒；在 500 并发的情况下，对评测流程各后台接口 99% 的请求应在 3 秒内得到响应，处理逻辑复杂或涉及较多 IO 的请求应在 10 秒内得到响应。
可靠性	系统要有一定的容错能力。如遇网络故障导致前端与服务器通信失败时，前端应能重试发送请求。用户在表达能力测试过程中，网络故障恢复正常后应能让用户继续之前的测试。服务器端因网络故障接收到前端页面的重复请求时，需保证不会出错。
安全性	系统应采用严密的身份认证，对所有来源的请求进行辨别。对数据的操作权限要用正确的控制措施。
可扩展性	系统内部应对关键业务方法抽象出接口（如数据操作，题目选取与推荐操作等），以便实现未来可能的升级或替换。添加新的系统模块时，应无需修改已有业务模块。
可维护性	系统应用服务器和评分节点均需要有完备的日志记录，代码应有详细注释，业务类的设计应符合常用的编程原则，实现业务模块间的解耦，提高可维护性。
易用性	各功能接口设计应符合通用的 API 设计风格，命名方法选用通用、流行的方法；前端界面应简洁友好、自解释性强，需要有合适友好的提示语，符合一般的人机交互认知，满足大部分用户的要求，前端提供及时、通俗的用户操作反馈。
可伸缩性	为了应对业务增长带来的访问压力，系统需要分别保持应用服务器节点与评分节点数量灵活扩大与缩小的能力，在需要时应能够快速部署新节点并对上游的客户透明。

本系统被设计用来服务真实的表达能力测试，需要具有一定的可用性和稳定性。为了应对业务增长带来的访问请求压力，系统需要能够进行扩展。为了应对未来可能的数据库选型等变更，系统应对其抽象出接口，方便切换到不同业务实现。同时对用户进行操作时系统的响应速度有一定的要求。如果用户在考试过程中遇到网络连接断开等意外情况，系统应该能够给用户友好的提示信息。此外，为了保证数据安全，系统需要在多方面对用户的数据操作权限进行控制。

3.3　系统总体设计

通过对上述需求分析得到的功能性需求进行拆分重组，并结合非功能性需求，本节针对系统进行整体架构设计与功能模块划分，并以 4+1 视图从不同角度描述系统总体设计。

3.3.1　系统架构设计

表达力评测系统整体架构如图 3.2所示。系统的架构设计采用了典型的分层体系结构，分为客户端、服务器和基础数据服务。客户端与服务端通过内容分发网络（CDN）和反向代理服务器（Nginx）以 RESTful 接口进行通信。

系统客户端主要包括浏览器和微信小程序，主要完成的功能有：各功能界面的展示，引导用户作答并自动采集、提交用户作答音频，获取评分结果并展示等。微信小程序需采用微信官方提供的开发工具基于 WXML 和 WXSS 进行开发。浏览器中运行的前端页面则采用 Vue.js 框架，以能够快速完成相对复杂的前端逻辑，同时保持较好的组件化，便于修改维护。同时，系统使用 D2Admin 模板方案加速开发。UI 组件库选用饿了么开发的 ElementUI，由于 ElementUI 的诞生恰是饿了么自身使用 Vue.js 时开发的标准组件库，所以其和 Vue.js 框架可以很好地集成，通过预定义的样式和便捷的使用方式可以方便地进行页面 UI 开发。前端应用的静态页面使用内容分发网络（CDN）进行托管，可以让不同地区的终端用户都有较好的下载速度，同时可以大幅减少对服务端的带宽压力。浏览器中的客户端应用与服务端的通信通过 axios 进行异步请求，可有效提升前端的用户体验。

系统服务端包括应用服务器和评分服务器。应用服务器采用 Flask 作为 Web 应用框架进行开发，Flask“微框架”的设计让其具有大量的高质量第三方组件库，能够快速和其他服务组件进行集成。由于 Flask 自带的 Werkzeug 服务器仅为方便开发而出现，高负载情况下的稳定性和效率都较差，故选用 Python Web 开发中主流的 uWSGI 作为应用服务器，能够提供更好的性能和更高的稳定性。应用服务器包括四个模块：评测模块、权限管理模块、试题管理模块和异步任务模块。其中评测模块为系统核心模块，其他模块围绕评测模块提供相关功能支撑，并处理其他相关业务。异步任务模块是对分布式任务队列 Celery 服务的封装，工作于应用服务器和评分服务器上，并作为二者的桥梁进行任务数据的传输。此外，评分服务器运行有评分服务，该服务由面向表达力评测的题目建设与分析系统提供。

系统的基础数据服务依赖于 MongoDB、Redis 和 BOS 对象存储三部分。其中 MongoDB 用于系统业务数据的持久化存储；Redis 集中存储用户 session，解决会

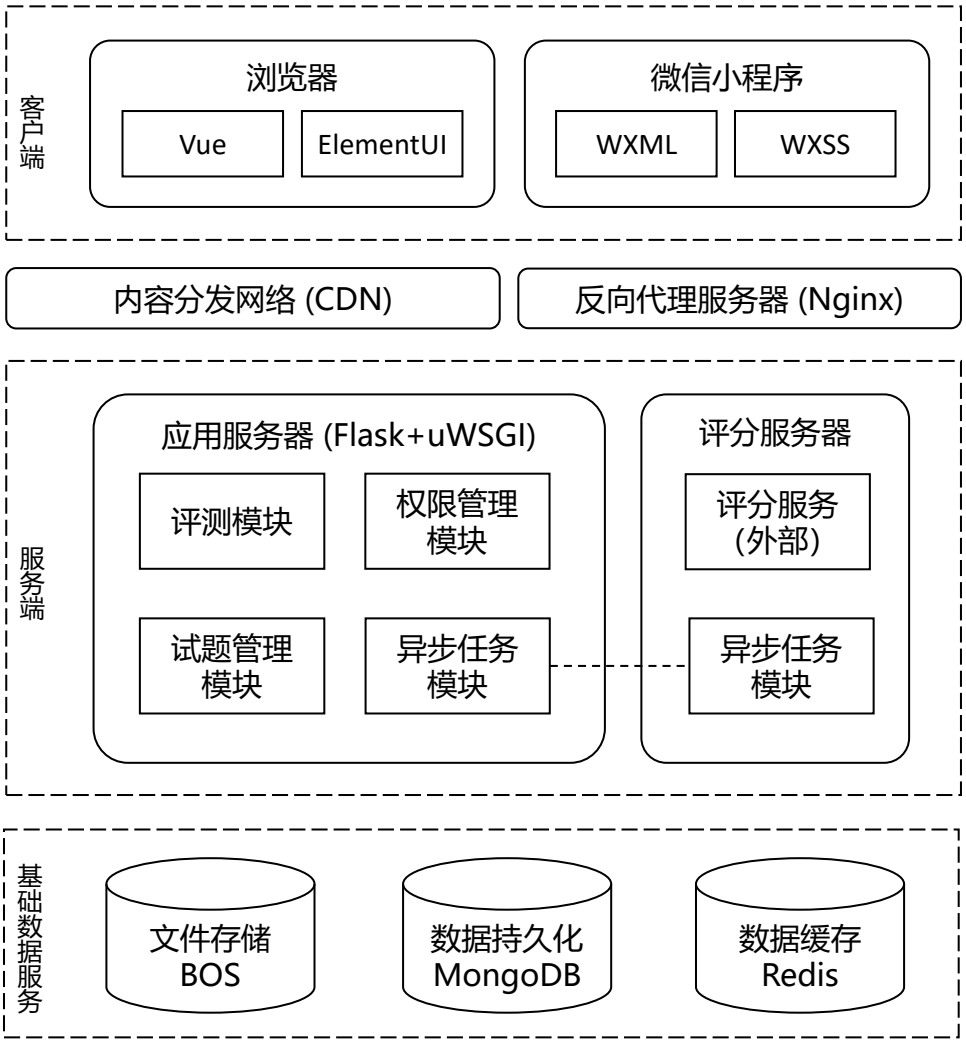

图 3.2: 系统架构图

话一致性问题，同时 Redis 也是其他热点数据的缓存数据库，以减少对 MongoDB 数据库的读写频率，提高系统响应速度。此外，系统使用 Redis 实现简易分布式锁，以实现多节点间的同步，或防止特定任务的重入。用户作答音频文件的存储由 BOS 对象存储服务完成，客户端在向服务端取得访问凭证后可以直接将音频文件上传至 BOS 对象存储服务，这种方式避免了文件经过应用服务器的中转，不仅可以加快文件存取速度，还可避免对应用服务器的带宽压力。此外，相较于将文件直接存储于服务器，BOS 对象存储服务无论在存储容量还是数据可靠性方面都时更好的选择，使用可靠方便的基础存储服务可以大幅降低开发工作量，将开发重心转移到核心业务部分。系统各组件通过 MongoDB、Redis 和 BOS 对象存储进行数据交换和共享，参考了典型的仓库模式进行设计。

当客户端将用户作答的音频文件上传到 BOS 对象存储后，客户端需要通知

应用服务器已完成上传操作，请求应用服务器进行处理（系统设计时 BOS 尚不支持上传成功回调操作），应用服务器收到请求后产生一个音频分析异步任务并放入任务队列，评分服务器需实时监控任务队列并获取任务。根据获取到的任务元数据，评分节点从 BOS 对象存储服务取得客户端上传的音频数据并完成特征提取和分析，给出得分，并将特征和得分存入数据库。因处理速度的要求，高并发时评分服务器通常需要同时部署较多的节点。多个评分节点间与应用服务器节点通过 Celery 任务队列进行连接。

为应对未来业务增长可能带来的访问压力，系统应能通过增加服务器搭架集群的方式保证服务稳定。如此，服务端应做到数据集中存储，将逻辑与数据分离，这样可保持系统较好的扩展弹性，方便增加服务端数量。多应用服务器节点使用 Nginx 负载均衡进行流量的分发。为应对 Nginx 可能出现的单点故障，需搭建两个 Nginx 节点并使用 keepalived 将其组成双机热备。正常情况下仅标记为 master 的 Nginx 节点占有虚拟 IP 并对外提供服务，当其失效时，标记为 backup 的 Nginx 节点可以快速地自动接管 master 节点的虚拟 IP 和服务。

3.3.2　系统模块及职责划分

系统共包括评测模块、权限管理模块、试题管理模块以及异步任务模块四大模块，图 3.3描述了各模块的职责及其调用关系，以及异步任务模块与外部的题目建设与分析子系统的交互情况。

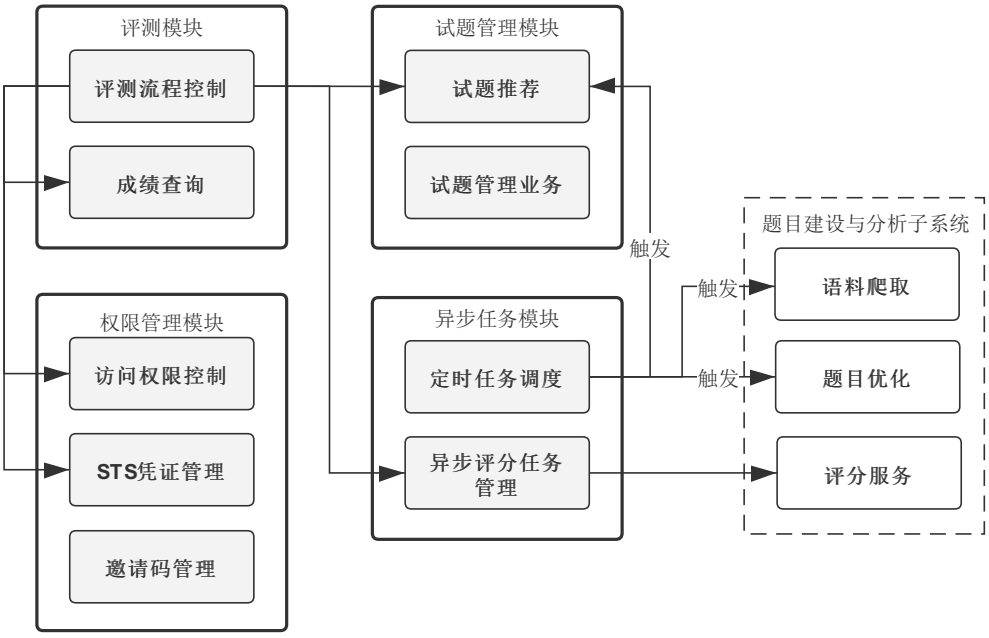

图 3.3: 系统模块划分

评测模块是系统的核心模块，主要负责整个评测流程的控制以及用户的成绩查询等操作，需要调用其余三个模块的相关功能以提供完整的评测服务。权限管理模块主要负责为评测模块提供用户身份识别与访问权限控制、客户端访问 BOS 对象存储服务所需的 STS 凭证的管理等支持，同时该模块也负责为邀请码的创建、查询与应用等操作提供支持。试题管理模块由试题推荐子模块和业务子模块构成，试题推荐子模块主要负责为评测模块提供试题选择的服务，同时也负责收集用户对试题喜好的反馈，并以此为依据利用基于用户的协同过滤算法向用户推荐其可能感兴趣的题目；业务子模块则负责提供管理员进行试题管理的业务支撑，以及用户对题目质量反馈的收集。异步任务模块主要从两个层面提供服务，一个是基于分布式任务队列对系统与评分服务的交互进行封装，提供应用服务器与评分服务器的数据交换支持，另一层面则是提供定时任务调度及定时任务的管理等功能。

以上服务端各模块均可使用 Flask 的蓝图功能，每个模块对应一个蓝图，在需要对外提供视图处理服务时注册到 Flask App 即可。同时各模块应保证良好的封装性，在设计与编码过程中需遵循常见的设计原则，抽象出友好的函数接口供其他模块进行调用，以保证各组件的高内聚和低耦合。

3.3.3　4+1 视图

系统架构图用来处理较高层次的系统设计与实现，但难以体现所有系统要点。为了从不同角度对系统架构进行描述，Philippe Kruchten 教授提出了“4+1”视图 [30] 方法。其中“4”表示逻辑视图、开发视图、进程视图和物理视图，“1”则表示场景视图。场景视图从用户的角度出发，识别业务需求，描述业务场景，与 UML 中用例图对应，系统用例图在本文中为图 3.1。下面本小节将从其余四个视图角度进一步描述系统的架构设计。

（1）逻辑视图

系统逻辑视图如图 3.4所示，从用户的角度描述了系统逻辑结构设计方案。在逻辑上，系统可被分为五个单元，其中中应用视图逻辑单元是对整个系统负责与用户发生直接交互的应用视图层的统称。包括适用于考生的评测考试及练习、题目质量反馈与题目收藏、成绩查询和使用邀请码更新权限，及适用于管理员的试题管理、邀请码创建与查询，以及定时任务管理等业务。逻辑控制单元是整个系统的业务逻辑控制中心，负责与视图层的交互，为其提供服务支撑。逻辑控制中心需要依赖访问权限管理、试题管理以及工具组件提供的基础服务。异步任务服务作为由定时任务、任务队列和异步分析服务三部分组成的逻辑单元，主要为逻辑控制中心提供异步评分服务，并负责触发预设的定时任务。数据

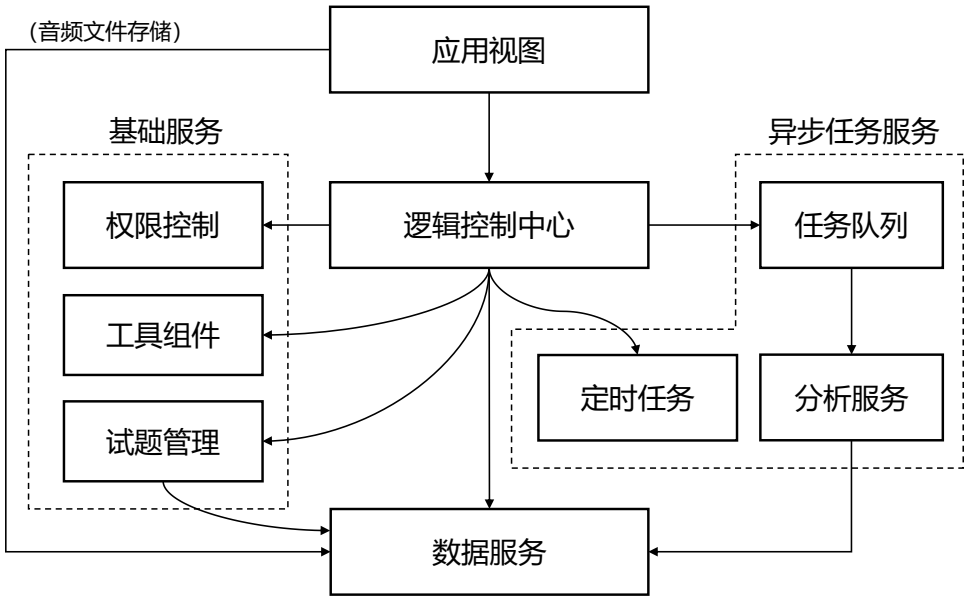

图 3.4: 系统逻辑视图

服务为整个系统提供各种层面的数据存储与访问的支持，包括各层面的业务数据持久化、文件存储、缓存等。系统各逻辑分区以数据服务为中心进行数据交换和共享。

（2）开发视图

系统的开发视图如图 3.5所示，主要从开发者的角度对系统工程结构进行了描述。为了降低开发难度，便于系统代码维护，系统总体采用分层设计的思想，分为视图层、业务逻辑层和数据层。

视图层从用户使用场景出发，包括评测页面，考试组织页面，权限管理页面和问题管理页面。由于 Vue 框架的 MVVM 特性，页面视图与数据得到天然绑定，可以有效简化开发。各页面的开发需充分利用 Vue 的组件化优势，合理组织各组件关系，维持较好的模块化。模块内父组件通过属性向下传递数据给子组件，子组件则通过事件向父组件传递数据。对于各页面间需要共享的数据，应通过 dispatch 或 commit 方式交由 Vue 状态管理模块（store）保存，以减少不必要的组件间引用，降低耦合。此外，评测页面需要调用录音组件进行用户作答音频的采集，在音频成功采集之后，需要调用 BOS 上传组件将用户作答音频直接上传到 BOS 对象存储。各页面与业务逻辑层的通信均通过 axios 异步请求进行。

与架构图中服务端模块拆分类似，业务逻辑层同样分为评测模块、试题管理模块、权限管理模块和异步任务模块。包含于试题管理模块中的试题推荐子模块与外部视图层不发生直接关系。其余模块均有视图层请求与之对应，需要

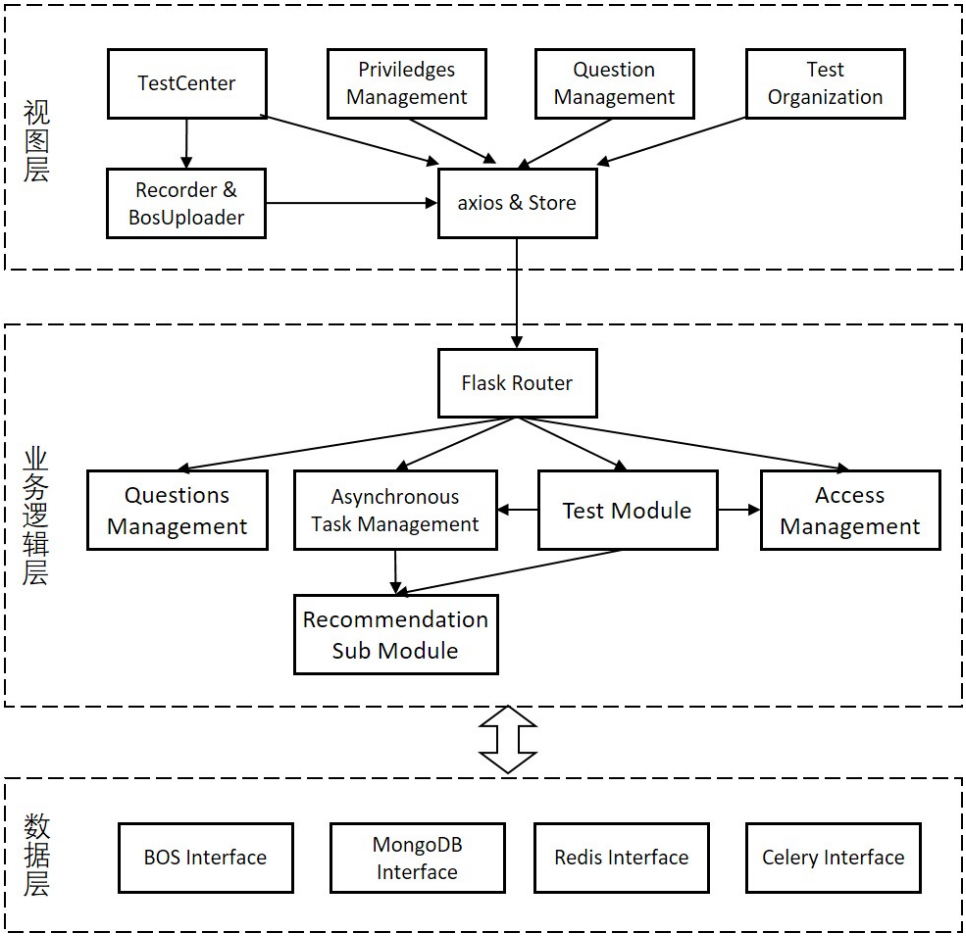

图 3.5: 系统开发视图

其提供业务支持。评测模块处理用户新的评测请求时，首先需要对用户是否有参加评测的权限进行鉴定，需要调用权限管理模块提供的接口。当用户作答完毕并将音频文件上传至 BOS 对象存储时，需要先从服务端获取访问 BOS 的凭证，该凭证获取步骤也交由权限管理模块完成。当文件成功上传之后，评测模块需要创建一个答案分析任务，并将该任务放入任务队列，由评分节点完成答案分析与评分，该过程中由异步任务模块负责与任务队列的互动，故异步任务模块需要提供相应接口供评测模块使用。试题推荐子模块负责利用协同过滤算法寻找与用户具有相似题目偏好的其他用户，以便于向用户推荐其可能感兴趣的问题。该子模块对于用户相似性的计算可定期进行，由异步任务模块进行调度。

数据层作为系统的基础层，封装了与 BOS 对象存储服务，与 MongoDB 及 Redis 进行交互的接口，方便业务逻辑层各模块的调用。

（3）进程视图

系统进程视图如图 3.6所示，主要描述了评测流程以及各管理流程中系统各

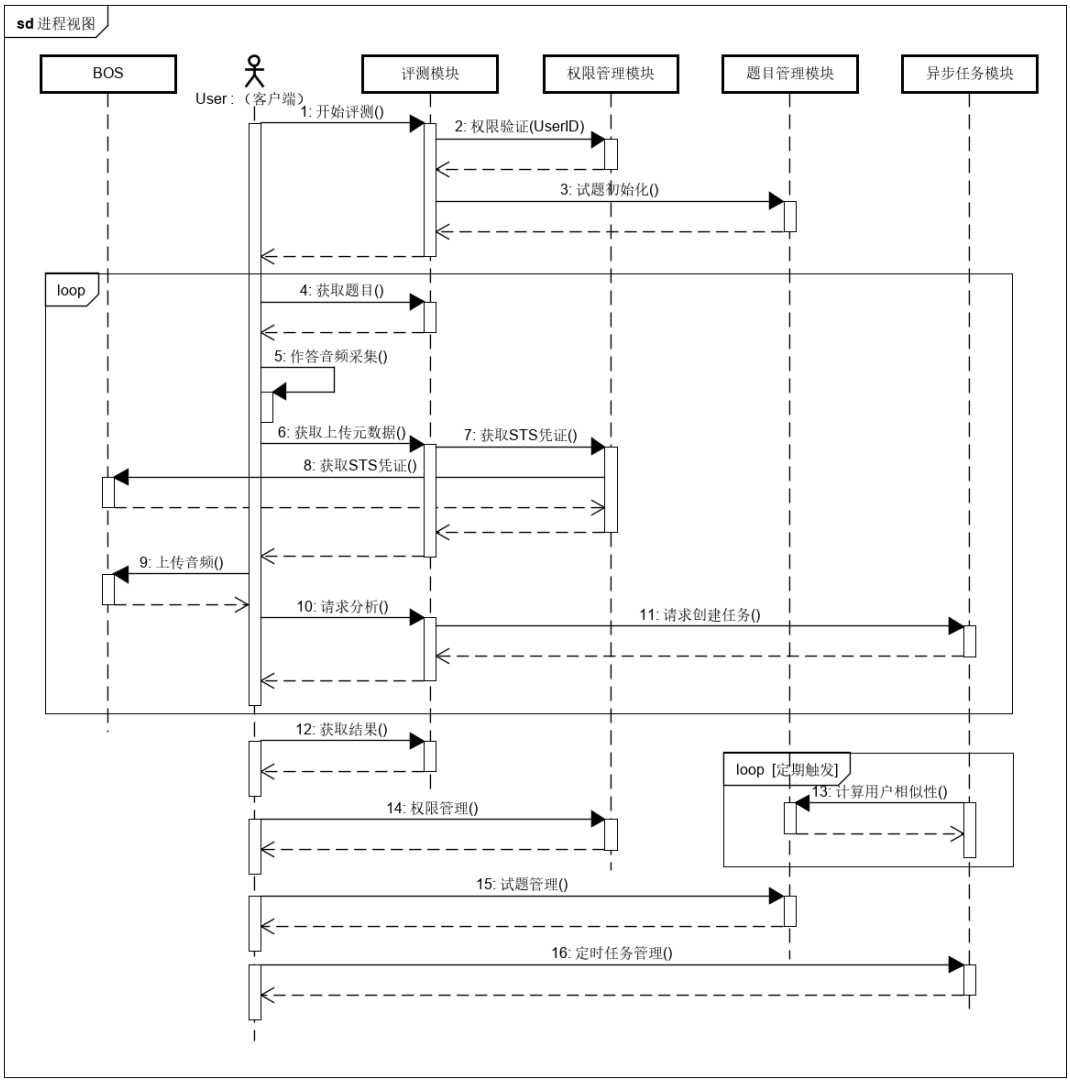

图 3.6: 系统进程视图

进程的整体交互关系图。在评测流程中, 用户请求开始评测考试或评测练习后, 评测模块需要调用权限管理模块进行用户权限的鉴定。用户权限鉴定通过后, 评测模块调用试题管理模块进行测试题目的初始化。在题目初始化过程中，如果判定用户进行的是评测练习，试题管理模块的试题推荐子模块将尝试选取用户感兴趣的题目进行测试初始化。然后，用户在前端页面的指引下完成答题，客户端完成用户作答音频的采集。如果客户端是浏览器，音频采集将由 JavaScript 脚本完成，如果客户端是微信小程序则通过调用微信 API 完成。录音结束后，客户端需要将音频文件上传至 BOS 对象存储供异步评分使用，这之前客户端需先向服务端请求访问 BOS 所需的 STS 凭证，这一操作需要评测模块调用权限管理模块，再由权限管理模块与 BOS 交互取得。当前端完成音频文件的上传后，需

告知评测模块上传成功，以便其调用异步任务处理模块进行异步评分任务的创建。需要进行这一操作是由于系统设计之时，BOS 服务尚未支持上传成功后的回调操作，若 BOS 服务支持上传成功后的回调操作，可由 BOS 直接回调评测模块接口。

除评测流程外，权限管理模块需要负责用户登录的处理，实现对用户身份的识别，并实现对用户的各页面访问权限进行控制。试题管理模块的业务子模块需要为管理员对题目的增删改查提供功能支撑。异步任务模块需要执行预先设定的定时任务，如定期触发推荐子模块进行用户相似性的计算并更新数据库，以便进行基于用户的推荐操作等。异步任务模块还需为管理员提供定时任务的管理功能，如任务的启用、禁用切换，任务的执行时间与频率编辑等。除定时任务的执行需异步任务模块调用相应模块的功能之外，这些功能均由相应模块直接与客户端进行交互。

（4）物理视图

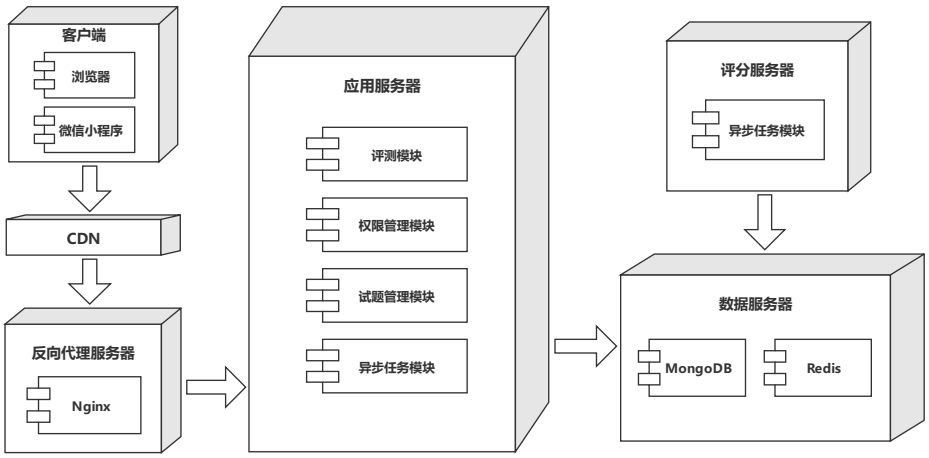

图 3.7: 系统物理视图

系统物理视图如图 3.7所示，通过该图可以了解到软硬件间的对应关系。用户通过浏览器或者微信小程序访问本系统的服务。用户的请求会先发送到 CDN，对于命中的规则（通常是图片、字体和样式等静态文件）直接由 CDN 返回数据。CDN 未能命中的请求将被发送到 Nginx Web 服务器处理，Nginx 通过反向代理与负载均衡将请求交由后端的应用服务器处理。Nginx 可部署在单独的服务器上，也可部署在应用服务器上，但为了增强系统可用性避免单点故障，需要部署至少两个 Nginx 实例。系统应用服务器的各模块部署在同一服务器上，各模块内部不保存状态，所用到的数据由数据服务器统一存储。这样可降低应用服务器的复杂性，也可给数据的管理带来便利。评分服务器中的外部评分服务运行

在 Celery 框架内，作为 Consumer 监控任务队列，不直接与应用服务器交互。

3.4　持久化数据模型设计

本系统采用 MongoDB 作为数据库。MongoDB 采用 BSON 形式的文档结构进行存储，将文档数据组织为键值对的形式，这让它非常适合对象类型数据的存储。而且 MongoDB 具有 schema-free（模式自由）的特点，不要求存储的数据具有相同的结构定义，给对象类型的修改也带来较大便利。对于 MongDB 数据库的设计，其实就是对系统底层数据模型的设计。

表 3.12: User Collection 文档结构详述

字段	数据类型	含义
id	ObjectId	文档被创建时分配的 id
_UserModel_role	String	用户角色（user/admin），通过实体类引入的枚举类型进行限制
vip_start_time	Datetime	用户具有做题权限的开始时间
vip_end_time	Datetime	用户具有做题权限的结束时间
remaining_exam_num	Integer	用户剩余考试次数
remaining_exercise_num	Integer	用户剩余练习次数
questions_history	Dict	用户做过的题目和对应时间
questions_liked	Dict	用户喜欢的题目和对应时间
users_similar	Dict	基于题目喜好计算的相似用户及相似度
...	...	用户的其他信息

从系统功能需求的角度出发，系统需要支持用户层面的管理操作，故将用户映射为 User 实体类，用户需要通过邀请码得到参与评测的权限，每个邀请码赋予用户权限可以有不同的时效与评测次数，故将邀请码映射为 Invitation 实体类。用户可选择进行评测考试或评测练习，如果把每个评测类比为一张考卷，映射为 CurrentTest 实体类，那么不同的考卷类型可类比为试卷模板，映射为 PaperTemplate 实体类，考试对应的题目则被映射为 Questions 实体类。考生用户对于试题质量的反馈将被直接记录到对应的 Question 实例中，对于感兴趣题目的收藏操作将被记录在用户对应的 User 实例中。

User 实体类对应的 Collection 文档结构如表 3.12所示。主要包括用户的角色信息、评测权限信息、评测历史信息以及相似用户用户信息等。用户的其他信息包括用户注册登录相关信息，用户的性别、职业、所在城市等信息，这些内容为业务扩展与将来推荐功能的改进而设，与当前系统其他部分相关性不大，在此不详细列出。

表 3.13: Invitation Collection 文档结构详述

字段	数据类型	含义
id	ObjectId	文档被创建时分配的 id
creator	ObjectId	创建邀请码的用户
create_time	Datetime	创建时间
code	String	邀请码
available_times	Integer	邀请码剩余可用次数
vip_start_time	Integer	做题权限开始时间
vip_end_time	Datetime	做题权限结束时间
exam_num	Integer	包含评测考试次数
exercise_num	Integer	包含评测练习次数
activate_users	Dict	被哪些用户使用，以及使用时间

邀请码 Invitation 实体类对应的 Collection 文档结构如表 3.13所示。主要包括邀请码的创建信息、邀请码字符串、邀请码包含的权限更新信息以及被何用户所使用等信息。

表 3.14: PaperTemplate Collection 文档结构详述

字段	数据类型	含义
id	ObjectId	文档被创建时分配的 id
name	String	模板名称
desc	String	模板描述信息
questions	List	逐题指定选题策略或直接指定试题 id
deprecated	Boolean	模板是否被弃用，默认 False
need_auth	Boolean	使用模板是否需要验证权限
password	String	需要验证时的密码
duration	Integer	评测时长

表 3.14描述了试卷模板 PaperTemplate 实体类对应的 Collection 文档结构。主要包括试卷模板的描述信息、选题策略、有效性及访问控制信息等。questions 属性中，依次放置每个题目的选题策略，包括试题类型和试题 id。如果试题 id 是合法的数据库 id，则为直接指定题目，否则将根据其数值匹配不同的选题算法，如随机选题、按题目喜好推荐题目等。

题目的 Question 实体类对应的 Collection 文档结构如表 3.15所示。主要包括题目的类型、使用次数、题目正文以及题目的反馈信息等内容。题目的其他信息包括题目难度的分类，题目的优化历史等内容，题目的优化是由当前系统的异步任务模块触发外部的评分子系统完成，其优化记录暂时与当前系统各模块相

表 3.15: Question Collection 文档结构详述

字段	数据类型	含义
id	ObjectId	文档被创建时分配的 id
type	Integer	题目类型
used_times	Integer	使用次数
text	String	题目内容
feedback_ups	Integer	题目被好评次数
feedback_downs	Integer	题目被差评次数
feedback_likes	Integer	题目被喜欢次数
...	...	题目其他信息

关性不大，在此不详细列出。

表 3.16: CurrentTest Collection 文档结构详述

字段	数据类型	含义
id	ObjectId	文档被创建时分配的 id
type	String	测试类型，表明该评测为考试或练习
user_id	ObjectId	与该测试关联的用户
test_start_time	Datetime	测试开始时间
test_end_time	Datetime	测试结束时间
questions	Object	当前测试所包含的题目相关信息
current_q_num	String	当前进行到的题目
results	Object	存储当前测试的评分结果

表示每次测试的 CurrentTest 实体类对应的 Collection 文档结构如表 3.17所示。主要包括当前测试所对应的测试用户、测试时间、测试题目、测试进度以及测试结果等信息。CurrentTest Collection 中 questions 对应 Python 中的字典类型，以键来表示题号，以值来存储题目相关信息。由于 MongoDB 在序列化时要求键不能为数字，故将其键从数字转为字符串再存储，current_q_num 是当前进行到的题目序号，可作为键从字典 questions 中直接取出题目，故相应的也存储为字符串格式。内嵌的 questions 中题目信息结构如表 3.17所示。

CurrentTest 文档内含的 questions 文档对应于 Python 的 Dict 数据类型，各题目的顺序无法保证，故每个内含的题目信息要包含一个 index 字段，表示其对应第几题。status 是当前题目的状态，共包括 none、url_fetched、handling 和 done 四种状态。其中 none 表示用户尚未做到该题，url_fetched 表示用户已做完该题并获取了答案需要上传的路径，handling 表示正在进行异步评分处理，done 则表示分析任务已经完成。

表 3.17: CurrentTest 之内嵌题目信息结构详述

字段	数据类型	含义
q_bdid	String	题目原始 id 对应的字符串
q_type	Integer	题目类型
index	Integer	该题目在当前测试中的序号
file_location	String	用户作答的音频文件存储位置，为业务实现的弹性而设，当前为 BOS
wav_upload_url	String	文件存储的具体路径
status	String	当前题目状态
content	Object	题目内容
analysis_results	Object	当前题目的评分结果

3.5　本章小结

本章首先从项目的背景、项目愿景和预见的使用场景等方面对项目进行了整体概述。然后对系统进行了涉众分析，再进一步得到系统的功能性需求和非功能性需求，并通过用例图对系统的功能性需求进行了场景描述，通过详细用例描述明确了需求细节。在此基础上，本章给出了系统的总体设计与模块划分，并结合 4+1 视图从多角度对系统总体设计进行了说明。最后给出了系统的持久化数据模型设计，亦即系统的数据库设计。

第四章　系统详细设计与实现

4.1　权限管理模块

权限管理模块主要负责如下三个功能：用户识别和访问权限的处理，用户评测权限管理，BOS 对象存储服务 STS 凭证的获取与发放。用户登录与访问权限的处理在页面层和接口层为系统自身安全提供了保障机制，用户评测权限的管理则从业务逻辑出发对用户使用系统功能做了权限要求。STS 凭证管理则是为了降低系统使用 BOS 对象存储服务中因验证信息泄露而造成的危害。基于面向对象设计的思想，在代码层面将该模块的后端代码单独组织为一个 Python package，以方便调用与维护。

4.1.1　用户识别与访问权限处理

系统在登录控制上，使用 Flask 的 flask-login 扩展完成用户认证状态在 session 中的保存。前端应用只需在访问服务端时带上 cookie，系统即可使用扩展提供的 current_user 识别发起请求的用户。但由于小程序的请求不带 cookie，我们需要自行定义从请求中获取用户的方法，由 flask-login 调用，以继续使用 current_user 带来的便利。同时这种无状态的机制更好地符合 RESTful API 风格。

```
# Store user_id and get token
def get_token(user_id:str) -> str:
    token = utils.random_code(length=12)
    redis.set(token, user_id, EXPIRE_TIME)
    return token

# Loader user and refresh token expire time
@login_manager.request_loader
def load_user_from_request(request):
    token = request.headers.get('token')
    if token:
        user_id = redis.get_and_renew(token, EXPIRE_TIME)
        if user_id:
            user = User.objects(id=user_id).first()
            return user if user else None
    return None
```

图 4.1: 通过 token 识别用户

系统采用 token 的机制从请求中辨识用户。用户登录系统后，系统需要维护

一个时效性的 token 与用户 id 的映射关系，并返回 token 给前端作为凭证。系统将这种映射关系存储在 Redis 中。该部分代码如图 4.1所示。

访问权限控制上，系统前端使用动态路由的方式进行控制。用户登录时，后端返回用户的角色信息，前端根据用户角色动态生成路由。前端整体路由组织为树状结构，用户必须先获取访问父节点路由的权限才可访问子节点对应的路由。

前端路由机制可以满足页面层的访问控制需求，但后端接口同样需要严格的验证措施，以避免恶意攻击。系统使用 flask-login 提供的 login_required 装饰器用来判断用户登录状态，并基于此包装出“需要以管理员身份登录”的装饰器，在需要限制的视图处理逻辑函数前加上该装饰器将函数包装为函数对象，函数执行前会先对用户登录状态与权限进行验证，验证通过才会执行用户请求的操作。从而达到管理用户对接口的访问控制。该部分关键代码如图 4.2所示。

```
# Definition of decorator for access control
def admin_login_required(func):
    @login_required
    @wraps(func)
    def decorated_view(*args, **kwargs):
        if not current_user.is_admin():
            return jsonify(errors.Admin_login_required)
        return func(*args, **kwargs)
    return decorated_view

# Usage:
@admin.route('/question/<id>', methods=['DELETE'])
@admin_login_required
def del_question(id):
    ...
    return jsonify(errors.success())
```

图 4.2: 权限控制装饰器的定义与使用方式

4.1.2　用户评测权限的管理

系统对用户评测权限通过邀请码机制进行管理。用户在系统注册后，需从管理员处取得邀请码对自身权限进行更新才可使用评测服务。管理员通过邀请码管理界面实现邀请码的创建与查询。邀请码管理页面如图 4.3所示。

页面整体分为上下两部分，上面为邀请码创建，下方为邀请码查询。管理员进行邀请码创建时，需要填写邀请码赋予用户使用评测服务的起止时间，可参加正式评测及评测练习的次数，以及需要创建的邀请码个数等信息。每个邀请

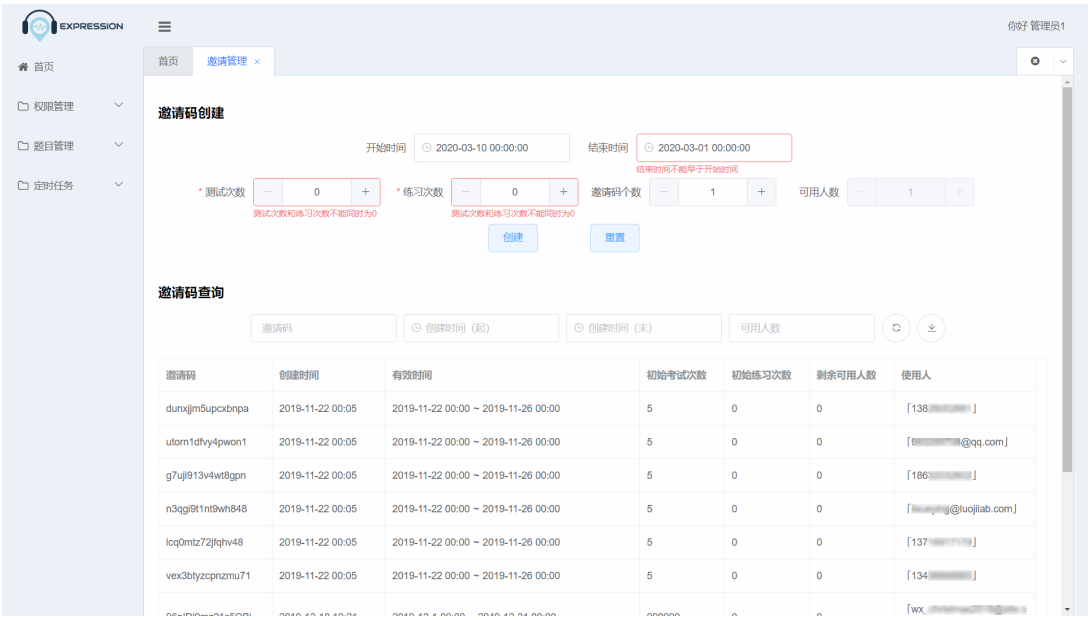

图 4.3: 邀请码管理页面实现效果

码默认可供一人使用，暂不支持调整。用户填写上述信息后以及点击提交按钮时，前端页面自动进行表单有效性检验，对于不符合邀请的项目给出提示，仅当所有数据项通过校验后才向后端接口发起请求。

在邀请码查询功能中，除上述信息外，页面还需显示邀请码的随机字符串、创建时间以及使用人等信息。对邀请码的查询支持使用随机字符串和起止时间等信息进行过滤，使用后端分页的方式进行查询与展示。

4.1.3　STS 凭证管理

评测过程中，由前端应用直接上传用户作答音频到 BOS 对象存储服务。为防止 BOS 原始访问凭证泄漏造成恶意攻击，以及更好地控制访问权限，客户端在上传音频文件时需要向服务端获取临时的 STS 凭证。

服务端在收到凭证获取请求后首先检查 Redis 缓存中是否有 STS 凭证，如果缓存有 STS 凭证，则需进一步检查凭证是否有效。在 Redis 中有 STS 凭证且凭证有效时，模块直接返回缓存的凭证给调用者，否则模块需要向 BOS 重新获取并缓存 STS 凭证，然后进行发放操作。

为优化处理逻辑，系统将 STS 凭证的 Redis 缓存过期时间设置为与 STS 凭证的过期时间相同，这样只要有 Redis 中有 STS 凭证缓存即可保证其为有效状态，从而省略有效性检查。此外，当且仅当用户正在进行测试时，前端应用需要上传音频文件至 BOS。故 STS 凭证管理功能不直接向前端应用开放 API，而是

在前端应用获取音频问上传路径时，由相关逻辑处理函数直接调用其类方法接口。对于用户有无正在进行的测试等检查同样由调用者完成，这种协作方式也更符合单一职责原则。

4.2　评测模块

评测模块主要用来为用户提供评测考试和评测练习服务，是系统的核心功能模块。考生通过电脑、平板、手机等多种设备以浏览器或微信小程序等方式接入系统参与评测，回答系统给出的题目。表达能力测试的题型有三种，分别是朗读题、转述题和自由表述题。这三种题型都要求用户以普通话语音进行回答。由于系统允许用户使用多种设备以浏览器或微信小程序接入系统，给用户带来便利的同时也带来了一些列其他问题。比如，用户多种多样的浏览器选择可能给前端程序的兼容性带来很大挑战，用户的网络环境可能不是很可靠，用户的测试环境可能很吵闹导致录音无法识别等。对于这些问题，我们在进行评测流程的设计与实现时要有充分考虑。

用户进行表达能力测试的整体流程概况描述为：用户进入做题界面并点击开始测试，系统对用户的测试环境进行检测。当用户的测试环境检测通过后，进入正式测试流程，系统根据用户的个人情况选择题目生成相应的试卷，评测练习可看作一种特殊的考试。考生需要在规定时间内，在前端界面的指导下逐题作答。答题完成后用户必须点击确认并提交当前题目才可进入下一题，如果用户未能在规定时间内完成作答，系统将自动停止录音并提交该题的作答音频。当用户完成所有题目并提交后，前端界面自动跳转至成绩报告页面，并以轮询的方式向应用服务器获取用户成绩，应用服务器在评分结束并正确返回用户成绩后，整个评测过程结束。下面主要从测试准备流程、正式测试流程以及正式测试流程的接口设计方面阐述评测模块的设计与实现。

4.2.1　测试准备流程

从用户点击开始到进入正式测试流程之前，可称为测试准备阶段。这一阶段的流程顺序如图 4.5所示。用户点击评测功能入口进入评测页面，然后客户端页面展示测试说明与注意事项等提示信息，若客户端为浏览器，JavaScript 脚本将自动检查浏览器是否为系统所支持的浏览器，并对浏览器的录音功能进行检查。若浏览器不受系统支持或录音功能无法使用，系统将拒绝进入后续流程，并向用户展示解决指导。若客户端检测通过，用户可点击开始测试按钮进入预测试页面。

```
@exam.route('/pretest-result', methods=['GET'])
@login_required
def get_pretest_result():
    test_id = request.args.get('testID')
    wav_pretest = WavPretestModel.objects(id=test_id).first() \
        if test_id else None
    if wav_pretest is None:
        return jsonify(errors.Test_not_exist)
    status = wav_pretest['result']['status']
    origin_text = wav_pretest['text']
    if status == 'handling':
        return jsonify(errors.WIP)
    elif status == 'finished':
        rcg_text = wav_pretest['result']['feature']['rcg_text']
        d = {
            "qualityIsOk": Levenshtein.ratio(origin_text, rcg_text) >= 0.6,
            "canRcg": True
        }
    else:
        d = {"canRcg": False, "qualityIsOk": False}
    return jsonify(errors.success(d))
```

图 4.4: 获取语音预测试结果关键代码

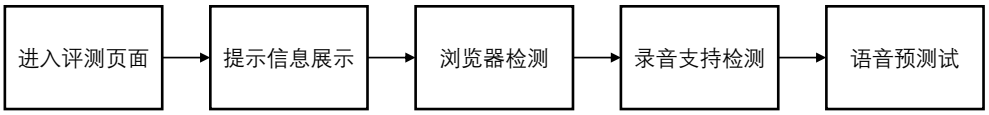

图 4.5: 评测准备流程

如果用户参加测试时的环境过于嘈杂，会造成异步评分过程中识别结果大幅下降或无法识别。语音预测试的目的即为了确保用户的录音环境合格才可进入正式测试流程。语音预测试的处理步骤同正式测试类似，先由用户朗读固定的一句话并确认提交，然后异步任务模块会调用外部服务对提交的音频进行语音识别，当识别结果与该语句原文相似度达到要求方可。这里的相似度使用莱文斯坦比[1]进行定义，达到 0.6 为符合要求。该部分关键代码如图 4.4所示（受篇幅限制，已省略日志处理记录等相关代码）。

4.2.2　正式测试流程

评测模块在系统中的位置及与其他模块的交互如图 4.6所示。作为直接与考生交互的模块，评测模块需要调用其他模块完成相关功能：调用权限管理模块完成用户参与评测的权限鉴定与 STS 凭证发放；调用试题管理模块完成试题的

[1]计算公式为 r = 1 - ld / sum，ld 为两字符串类编辑距离，sum 为其长度之和

初始化与练习题目的推荐；调用异步任务模块完成异步评分任务顶等。评测过程中各模块间交互的进程图已由第三章中系统进程视图（图 3.6）给出。下面本文将围绕后台接口的详细设计与实现对正式评测的流程的实现进行阐述。

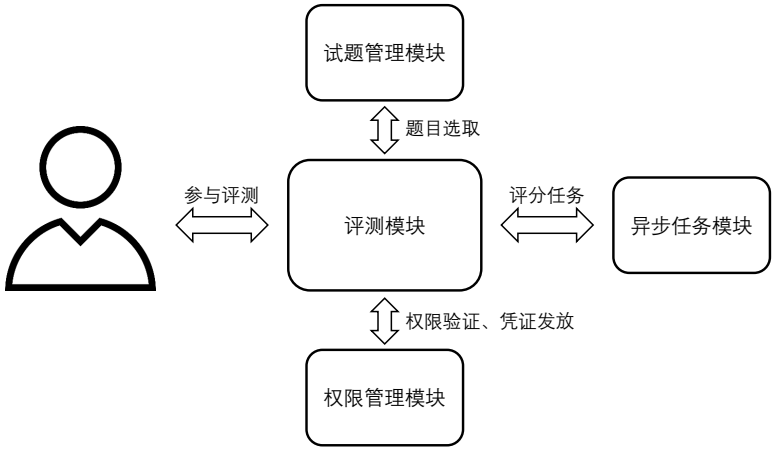

图 4.6: 评测模块在系统中的位置

在代码层面，评测模块被设计为一个蓝图，并在应用初始化时被注册，内部定义的所有 API 统一挂接到/api/exam 目录下。评测流程中涉及到的接口以 RESTful 风格设计，如表 4.1所示。但不同于预测试，正式测试对安全的要求更高，exam_id 应由后端 session 进行控制，不应由前端传入，故 URL 中未包含 exam_id。

表 4.1: 评测模块接口设计表

方法	**URL**	**附加参数**
POST	/api/exam/<paper_template_id>	[forceCreate]
GET	/api/exam/<question_num>/next-question	无
GET	/api/exam/<question_num>/upload-url	无
POST	/api/exam/<question_num>/upload-success	无
GET	/api/exam/result	无

为应对用户测试途中因断网、断电及意外关闭页面等因素导致的测试中断，系统需要实现断点续做的功能。用户选取测试类型并点击开始测试之后，客户端携带对应试卷模板的编号访问评测初始化接口以尝试创建新的测试。服务端收到请求后需先判断用户有无尚未完成测试，若有未完成测试，直接返回用户

```
@exam.route('/exam', methods=['POST'])
def init_exam():
    force_create = request.args.get('forceCreate', False)
    has_left_exam, now_question_num = find_left_exam(current_user.id)
    if has_left_exam and force_create:
        give_up_left_exam(current_user.id)
    if not has_left_exam or force_create:
        # Check if has priviledge to init a test
        if Setting.LIMIT_EXAM_TIMES and current_user.can_do_exam():
            return jsonify(errors.No_exam_times)
        now_question_num = 0
        # Init exam
        test_id = QuestionUtils.init_question(str(current_user.id))
        if not test_id:
            return jsonify(errors.Init_exam_failed)
        else:
            if Setting.LIMIT_EXAM_TIMES:
                current_user.remaining_exam_num -= 1
                current_user.save()
            ExamSession.set(current_user.id, 'test_id', test_id)
    return jsonify(errors.success({'nowQuestionNum': now_question_num}))
```

图 4.7: 评测初始化伪代码

退出时做到的试题序号，若没有未完成测试，则调用试题管理模块为该次测试生成试题组合并放入数据库，返回试题序号 0。客户端收到服务端响应后需判断当前题号是否为 0，若为 0 则表示已创建新测试，否则应提示用户是否继续未完成的测试。若用户选择放弃未完成的测试，则客户端需带上 “forceCreate=True” 作为参数再次请求/api/exam 接口，服务端检测到值为 True 的 forceCreate 参数后将用户未完成的测试标记为完成，并重新创建新测试及返回题目序号 0。该部分伪代码如图 4.7所示。

然后客户端需要访问 next-question 接口从服务端获取题目，接口中 <question_num> 为发送请求时最后完成题目的序号。题目的获取设计为逐题进行，以降低题目泄漏与用户作弊的风险。服务端则通过缓存用户当前做到的题目以限制重复或不合理请求，具体措施为：记缓存中记录的当前题目序号为 A，收到请求携带的当前题目序号为 B，当且仅当 B 与 A 相等时，服务端正常返回题目；若 B 小于 A，说明到达的请求为无效的重复请求，若 B 大于 A，则到达的请求为不合理的请求，服务端直接返回参数错误即可。客户端获取题目后按题目中的作答时间限制、题目提示、题目正文等内容进行展示，引导用户正确完成作答，并自动完成用户作答音频的采集。

用户完成作答点击结束回答时或题目作答时间结束后，客户端自动停止音

频采集。然后，客户端访问 upload-url 接口获取题目上传至 BOS 所对应的文件路径（即文件保存需分配的 Key 值）与访问 BOS 所需的 STS 凭证。接口中 <question_num> 为当前音频对应的题目序号。服务端收到请求后根据用户信息、当前测试信息与当前时间为音频分配上传路径，并调用权限管理模块获取访问 BOS 所需的 STS 凭证，一并返回给客户端。

客户端使用后端返回的 STS 凭证将音频上传至 BOS 后，需要访问 upload-success 接口，告知服务端题目上传已完成，请求服务端创建异步评分任务。服务端对异步评分任务的创建通过调用异步任务模块相关函数接口完成（该部分将在异步任务模块部分详细阐述），并将其返回的异步任务 ID 放入 Redis 缓存，以供后续进行任务状态查询。

用户正常完成所有题目的作答后，客户端再次向服务端请求题目时，服务端返回测试结束标志，客户端自动跳转至结果页，并以轮询的方式从服务端获取结果。服务端收到请求后首先使用缓存的异步任务 ID 构建 Celery 的 AsyncResult 对象，并获取任务状态。若有任务尚未完成则返回 WIP（Work In Progress），否则说明所有题目的异步评分任务已结束，服务端访问数据库查询测试各题目得分并得到总分数，返回给客户端供其展示。

4.3　试题管理模块

试题管理模块主要完成的工作有两个：一是为评测模块提供题目选取服务，二是为管理员对题目进行管理的业务提供支撑。下面从试题推荐和业务两个子模块对其设计与实现进行描述。

4.3.1　试题推荐子模块

该子模块主要用于为评测模块生成试题组合。评测模块进行评测初始化时根据试卷模板逐题指定选题策略并记入数据库对应的 CurrentTest 文档。如果模板直接指定合法的题目 id，则该子模块直接取得题目放入组合，否则将视 id 为预设策略编号，并依其进行试题选取或推荐。在系统当前策略中，编号为 0 表示不考虑用户的偏好，直接从用户未做过的题目中随机取出所需类型与数量的题目，存入数据库对应的 CurrentTest 文档并返回该数据对象给调用者，通常用于评测考试；策略编号为 1 则表示使用基于用户的协同过滤算法提供简单的试题推荐功能，通常用于评测练习。

概括来说，要给用户 A 推荐感兴趣的题目，我们需要先找到与用户 A 兴趣相似度较高的 k 个用户，然后从这 k 个用户感兴趣的题目中找出 A 没有做过的题目，推荐给 A。如表 4.2所示，我们根据用户的喜好信息找到与用户 A 相似的

用户 C，然后便可选取用户 C 感兴趣但用户 A 尚未做过的题目 e 推荐给用户 A。

表 4.2: 基于用户的协同过滤

	题目 a	**题目 b**	**题目 c**	**题目 d**	**题目 e**
用户 A		√	√		**推荐**
用户 B	√			√	
用户 C		√	√		√
用户 D	√				√

为完成推荐功能，首先需要建立用户兴趣与题目的联系，用户兴趣数据的收集通过用户做题时点击题目下方的反馈栏中的喜欢按钮（如图 4.8所示）完成。用户点击“喜欢该题”按钮后，该题会记入用户的喜欢记录，同时该题目的被喜欢次数加 1，用户再次点击按钮可取消喜欢。

图 4.8: 兴趣反馈按钮实现效果

系统选用余弦相似度公式计算用户兴趣相似度。由于每两个用户之间都需要计算相似度，时间复杂度会达到 $O(n^2)$。在实际操作中，如果两个用户所喜欢的题目完全没有交集，我们可直接跳过该次计算。为达到此目的，需首先建立题目到用户的倒查表，然后即可遍历倒查表建立用户行为相交矩阵进行过滤操作。即参考图的邻接矩阵存储方式，以每个用户为顶点，初始化不同用户间的连接权重为 0，然后遍历每个题目，如果该题目同时被某两个用户 A、B 喜欢，则在用户行为相交矩阵上将两用户间的连接权重加 1，最后遍历矩阵，计算矩阵中连接权重不为 0 的用户相似度即可。该部分核心代码如图 4.9所示。

经过上述计算后可得到用户的兴趣相似度矩阵，通过查阅矩阵我们可得知每个用户与其他用户的兴趣相似度。故接下来只需遍历矩阵中每个用户，根据

```
class CollaborativeFilter(object):
    ...
    def create_reverse_table(self):
        users = UserModel.Objects({'questions_liked': {'$ne': '{}'}})
        for u in users:
            questions = u.questions_liked
            for q in questions:
                if q not in self.reverse_table:
                    self.reverse_table[q] = set()
                self.reverse_table[q].add(u)

    def create_weight_matrix(self):
        for question, users in self.reverse_table.items():
            for u1 in users:
                for u2 in users:
                    if u1 == u2:
                        continue
                    self.weight_matrix.setdefault(u1, {})
                    self.weight_matrix[u1].setdefault(u2, 0)
                    self.weight_matrix[u1][u2] += 1

    def calc_similarity_matrix(self):
        for u1, v1 in self.weight_matrix.items():
            for u2, cnt in v1.items():
                m = self.user_like_counts[u1]
                n = self.user_like_counts[u2]
                sim = cnt / math.sqrt(m * n)
                self.similarity_matrix[u1][u2] = sim
```

图 4.9: 计算用户兴趣相似度矩阵核心代码

其与关联用户的兴趣相似度从到到低排序，即可得到与其最相似的 k 各用户。该 k 各用户即数据模型中 User.users_similar 的数据来源。为减小选择推荐题目时的计算量，系统暂将存入数据库的相似用户数量限制在 10 以内。推荐题目时，系统从用户数据实体的 users_similar 中取得与其相似的用户，并对相似用户喜欢的题目进行汇总。然后系统从中剔除当前用户已做过的题目，对剩余题目分别计算用户的兴趣度并排序，按顺序向其推荐剩余题目。为了避免用户每次请求都需要重复进行数据库的查询与题目排序，系统使用 redis 作为队列，把排序后的结果以 list 存入其中，每次进行试题推荐时从中逐一取出题目。若存储的题目均被用户获取，则系统从剩余题目中随机选择用户未做过的题目。

4.3.2　业务子模块

试题管理模块的业务子模块的用户为管理员，主要为其提供题目相关的后台管理功能。用户在点击试题管理后，Vue 框架首先通过动态路由调起试题管理首页进行展示。其实现效果如图 4.10所示。

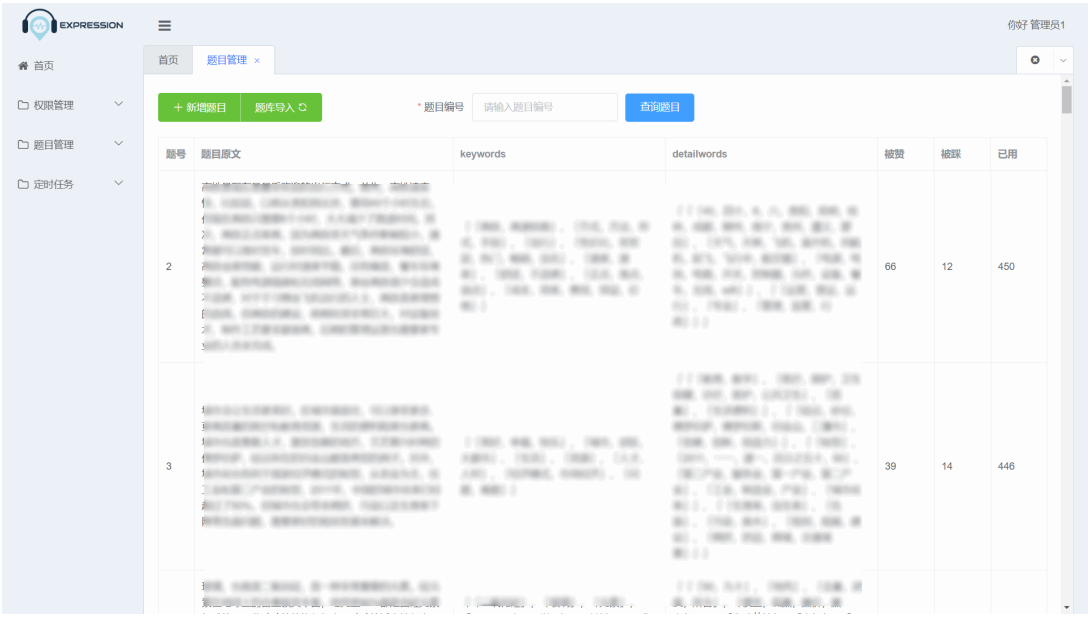

图 4.10: 试题管理首页实现效果

试题管理主页分为三个区域：顶部快捷栏左侧为添加题目入口，快捷栏中间区域为题目查询入口，快捷栏下方的表格对系统已有题目进行分页展示，主要包含题目的编号、题目正文、用于评分的词库以及题目的反馈等信息。点击题目所在行，可进入题目查看界面。点击左上角的新增题目与题库导入均可直接进入题目添加界面，不同之处在于点击新增题目进入界面仅提供空白的题目模板，点击题库导入按钮进入界面会从语料库中选取语料填入模板。语料库由面向表达力评测的题目建设与分析系统使用爬虫技术定期爬取并处理获得。该部分页面实现效果如图 4.11所示。

系统服务端返回语料库的备用题目后，前端进行加载展示，管理员可直接修改题目文本，也可通过点击操作便捷地修改题目评分所用的主旨与细节关键词。对于修改后的语料，用户还可通过点击一键生成关键词按钮调用后端关键词提取与同义词扩充功能重新填充主旨与细节关键词。这些修改在未提交时仅保存于前端界面，用户随时可选择对修改进行保存或放弃，对于语料库中质量过低的备用题目，用户可点击丢弃进行删除操作。

用户通过点击试题管理首页展示的题目进入查看页面后，同样可选择对题

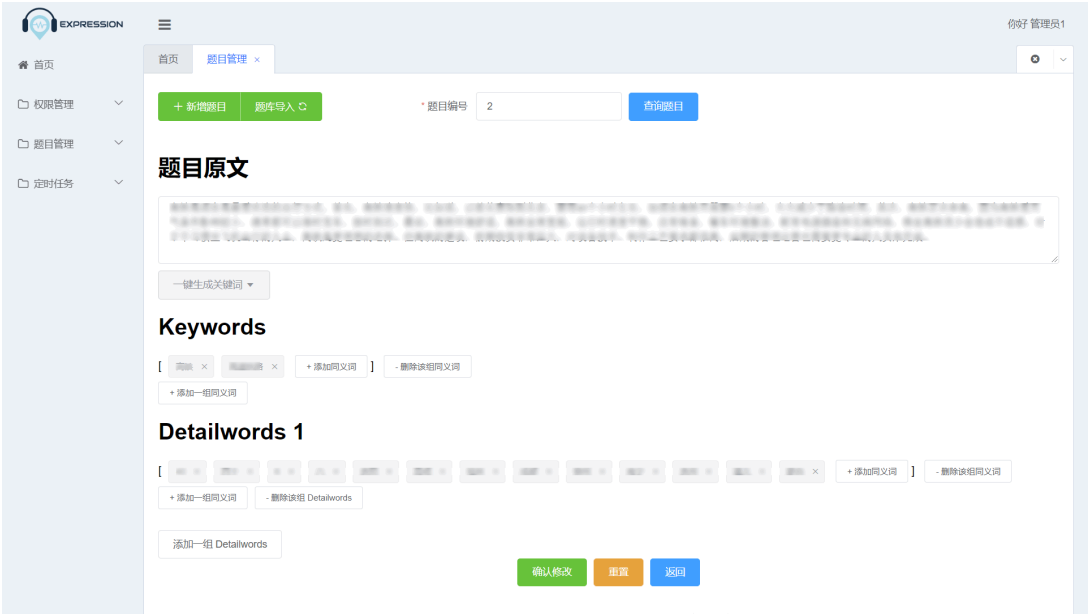

图 4.11: 试题添加页面实现效果

目进行编辑与删除操作，其界面实现与题目添加界面一致。

4.4　异步任务模块

异步任务模块是对应用服务器上的异步任务创建组件、Celery 任务队列以及评分节点上的异步任务消费组件的总称。本系统中 Celery 任务队列使用 Redis 作为 Broker 与结果存储的 Backend。该模块主要提供两个服务：一是执行异步评分任务，二是进行定时任务调度。异步任务管理组件负责定时任务的调度，同时也是异步评分任务的生产者，评分节点是异步评分任务的消费者。二者通过 Celery 任务队列进行同步，通过 MongoDB 数据库和 BOS 对象存储服务进行数据交换与共享。下面对这两个服务的相信设计与实现分别进行描述。

4.4.1　异步评分服务

首先需要再次申明的一点是，对于异步评分服务，本文仅讨论模块结构与流程的具体设计和实现，评分节点中所调用的具体评分模型与算法由外部服务进行提供。

（1）异步评分流程

图 4.12为系统的数据流图，展示了系统中各种数据的流动方向与存储关系。在用户参加评测考试或练习的过程中，用户的作答音频将首先被前端直接上传

至 BOS 对象存储服务，然后前端会告知应用服务器音频上传成功。此后应用服务器根据当前测试的 id 与前端上传的题目编号创建一个异步评分任务，并将任务元数据放入对应的 Celery 任务队列。评分节点作为消费者对任务队列进行监测。一旦评分节点从任务队列获取到新任务的元数据，便根据元数据所包含信息访问 MongoDB 获取所需分析的任务所涉及的题目类型、音频文件存储位置、题目关键词权重等信息。接下来评分节点便可从 BOS 获取音频文件进行分析。最后，评分节点将题目分析结果写入数据库，并在 Celery 任务队列设置执行状态为完成。前端在测试的最后一题提交完成之后，便自动跳转到结果页面，以轮询的方式向应用服务器询问考试结果。应用服务器在收到请求后首先验证各题是否都已完成评分，若所有题目都是评分完成状态，则计算总分并返回给用户。

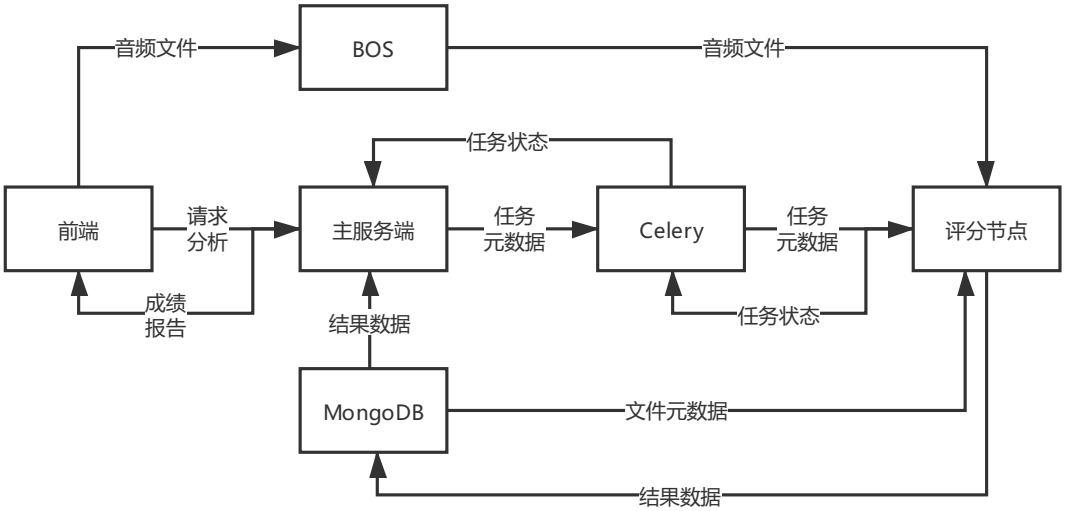

图 4.12: 评测服务数据流图

使用轮询机制进行结果查询，可以简化系统服务端设计，同时可以避免长连接和长轮询机制对服务端造成的额外开销。为了减少读数据库次数，系统会先从 Redis 中读取异步评分任务的状态，并仅在任务已经结束后再去读取数据库。这种验证机制需要任务投放时返回 AsyncResult 的 id，并在查询任务状态时重新生成 AsyncResult 对象。评测考试具有多道题目，对应着多个异步评分任务，若系统在计算总分之前对每个异步任务的状态都进行判断，则需保存所有异步任务的 id。但由于用户每作答一道题目便提交进行评分，异步任务完成的顺序有很大概率与用户提交作答顺序相同，所以我们可以简化设计，仅判断最后一次异步评分任务是否完成即可。

（2）异步评分任务的创建

为了提高代码的可维护性，异步任务模块需要提供一个评分任务创建接口，

评测模块只需调用该接口即可将任务放入任务队列，交由评分节点进行评分处理。虽然系统评分行为具有幂等性，重复的评分不会导致结果改变，但须知注意评分服务器资源是系统开销较大的资源，无用的重复评分行为会导致服务器资源的浪费。为了避免因网络问题或用户恶意伪造导致创建评分任务的请求重复到达而造成的资源浪费，系统需要避免将同一任务重复放入任务队列。对此系统采用 redis 分布式锁进行控制，每个任务被放入队列之前都需要获取任务对应的锁。不同的是锁的获取过程中如果获取不到即放弃获取，获取成功后亦无需主动解锁，等待其过期自动被 redis 清除即可。该部分代码如图 4.13所示。由于系统压力测试时需要允许任务的重复放入，所以留有是否使用锁的可设置项。

```
class CeleryQueue(object):
    @staticmethod
    def put_task(q_type, test_id: str, q_num=None, use_lock=True):
        try:
            if use_lock:
                key = 'celery_lock:%s-%s-%s' % (q_type, test_id, q_num)
                val = ''
                lock = redis_client.set(key, val, ex=60, nx=True)
                if not lock:
                    raise Exception('Failed to get redis lock')
            if q_type == 'pretest':
                ret = analysis_wav_pretest.apply_async(
                    args=(str(test_id),),
                    queue='q_pre_test', priority=20)
            elif q_type in [3, '3']:
                ret = analysis_main_3.apply_async(
                    args=(str(test_id), str(q_num)),
                    queue='q_type3', priority=10)
            elif q_type in [1, 2, '1', '2']:
                ret = analysis_main_12.apply_async(
                    args=(str(test_id), str(q_num)),
                    queue='q_type12', priority=2)
            else:
                raise Exception('Unknown q_type')
            current_app.logger.info("AsyncResult id: %s" % ret.id)
            return ret.id, None
        except Exception as e:
            return None, e
```

图 4.13: 异步评分任务创建接口

评分任务创建接口代码中的 analysis_wav_pretest 等对象是经 Celery 的 task 装饰器对任务函数进行修饰得到。Celery 本身是一个分布式异步任务框架，依赖于中间人和反射机制完成远程调用，任务的生产者和消费者不必在同一物理机上，正如这里调用的任务创建接口即可生产任务，而评分服务器却是单独的节

点。无论是生产者还是消费者，都需要对 Celery 进行配置，并告知其可执行的任务。但不同的是，任务代码在消费者上需要完整的声明与定义，在生产者上却只需要声明函数以供调用即可，函数体部分完全可由 pass 关键字代替。另外，Celery 的 task 装饰器可选地接受参数，若无参数指定任务名称，其默认将采取包名加函数名的方式为任务命名，为了保证反射机制的正常使用，生产者需要和消费者保持完全一样的文件目录结构，这会极大地限制代码文件在文件系统中的组织方式，故使用 task 装饰器时应务必显示指明任务的名称，便于代码文件在文件系统中的组织。

（3）提升评分节点性能

评分节点作为 Celery 任务队列的消费者，其主体程序运行于 Celery 框架中。对于评分任务，系统划分了三个任务队列：预测试队列单独为一个队列，题型三（自由表述）单独一个队列，题型一（朗读题）和题型二（转述题）共用一个队列。三个队列的优先级依次降低。这样划分的目的是为了尽快处理完一个用户的所有评分任务，减少大量用户同时等待的情况出现。每个评分节点，均有若干个 worker 监控着对应的任务队列，一旦获取到任务，即从 BOS 对象存储服务获取用户作答音频，并对音频进行转码切分等工作提取音频特征，之后使用商用语音识别接口将音频专为文字，进一步提取文本特征。这个过程中，评分节点需要并行执行的任务数量较多，且从 BOS 下载文件和调用语音识别接口有相当一部分时间花在等待网络应答上。为了减少评分节点在多进程或多线程环境下的内存占用，以及进程或线程调度的 CPU 占用，系统启动少量的 worker 进程，并在每个 worker 进程之内采用协程的方式进行并发任务调度。从 BOS 对象存储服务下载音频进行处理，不可避免的会产生临时文件。为了避免磁盘 IO 带来的额外开销，系统采用 Python 的 BytesIO 实现内存文件，并封装相应的工具函数完成音频或文本数据的存取。整个过程中不涉及磁盘读写操作，任务结束后由垃圾收集器进行内存的回收，提升文件存取效率的同时也避免了使用文件系统时需要手动删除临时文件的麻烦。用于微信小程序录音 API 提供的音频文件格式是 aac 编码的 m4a 格式，无法直接进行空白切除和特征提取等操作，系统使用 FFmpeg 将 m4a 文件转成 pcm 编码的 wav 文件之后再进行进一步操作。由于 FFmpeg 支持基于 pipe 的流式转换，我们使用 Python 实现从 stdin 进行音频内容的输入，并收集 stdout 的输出写入内存文件，以避开文件系统的操作。该部分代码如图 4.14所示。

此外，Celery 的 worker 默认寿命是无限长的，实践中发现 worker 占用的内存存在无法及时释放最终造成部分程序崩溃的可能。对此，Celery 提供了 CELERYD_MAX_TASKS_PER_CHILD 的选项，通过制定该值的大小，可让 worker

```
from io import BytesIO
from subprocess import Popen, PIPE

def m4a2wav_bytes_io(bytes_io_file):
    bytes_io_file.seek(0)
    content = bytes_io_file.getvalue()
    cmd = ['ffmpeg', '-n', '-i', 'pipe:', '-acodec', 'pcm_s16le',
           '-f', 'wav', '-ac', '1', '-ar', '8000', 'pipe:']
    p = Popen(cmd, stdin=PIPE, stdout=PIPE, stderr=PIPE, bufsize=-1)
    out, _ = p.communicate(input=content)
    p.stdin.close()
    if out.startswith(b'RIFF\xff\xff\xff'):
        return BytesIO(out)
    else:
        return None
```

图 4.14: 使用 FFmpeg 在内存中完成音频文件流式转换

在执行一定数量的任务之后自动退出以归还内存，并启动新的 worker 处理其余任务。

4.4.2　定时任务调度服务

为了维护系统正常稳定运行，系统需要执行一些定时任务。例如，系统数据库在设计时采取了冷热数据分离的机制，需要定期从高频访问的 CurrentTest 集合中将冷数据移动到 History 集合以提高 CurrentTest 读写性能；试题管理模块提供的试题推荐功能需要依赖于用户相似度的定时计算；此外，系统需要定期触发题目建设与分析系统的评分优化脚本的执行，对题目关键词在评分中所占的权重进行调整等。这些定时任务统一由异步任务模块提供的定时任务调度服务进行管理。

Python 生态下的定时任务管理常见的方案有使用系统 Cronie、使用 Celery 调度以及使用 Apscheduler 等。相较于 Cronie 和 Celery 而言，Apscheduler 使用更加方便。它提供了基于日期、固定时间间隔以及 Cronie 类型的任务触发机制，在程序运行过程中即可快速增删任务。故系统使用 Apscheduler 作为定时任务触发器，并对其进行包装以提供基于浏览器页面的管理功能。同时系统使用数据库作为 Apscheduler 的任务存储场所，当调度器重启时可自动加载已有作业，不会丢失任务之前的状态。

由于所有定时任务都有相应的后台程序来执行，所以系统并不支持从前端直接添加任务代码，管理员通过前端界面只能控制已有任务的启停，或更改任务的触发方式与执行时间。定时任务管理页面实现效果如图 4.15所示。

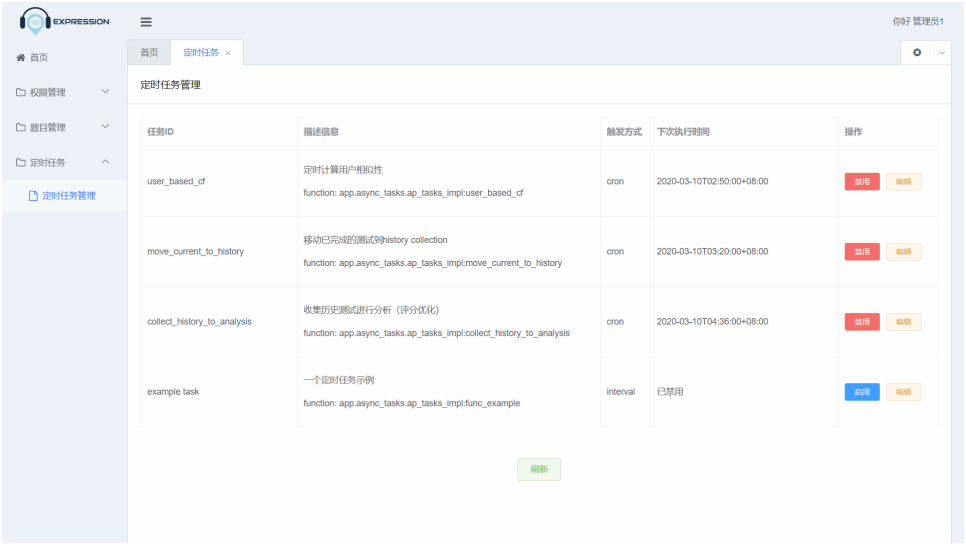

图 4.15: 定时任务管理界面实现效果

在后端代码上，需要动态生成 Apscheduler 的初始配置文件以避免代码中定义的任务与数据库已有任务的冲突。由于 Apscheduler 自身无单独进程，必须依赖其他程序才能执行，在初始化之前任何对其 API 的调用都会失败，所以系统启动时，需要先从数据库获取已有任务的 id，仅将不存在于数据库中的代码定义任务添加到初始化配置文件。

同样是因为 Apscheduler 需要依赖于其他程序的机制，当服务器的 uWSGI 启动多个进程时也将出现多个 Apscheduler 调度器，导致定时任务的重复执行。对于单机部署的 Apscheduler，可以采用文件锁的方式进行避免，但对于分布式或集群部署的情况，我们需要使用 Redis 分布式锁予以解决。为方便实现，在不改变原代码的基础上对函数功能进行扩充，系统定义了接受参数的 Redis 锁装饰器，仅需传递时间参数给装饰器并对原函数进行修饰，即可保证原函数在指定时间内不会重复执行。装饰器定义与使用示例代码如图 4.16所示，通过指定时间参数 600 即可限制 move_current_to_history 函数在 600 秒内不会重复执行。

4.5　系统部署和可用性提升

4.5.1　使用 Docker 和 Jenkins 进行系统部署与持续集成

系统使用 Docker 进行快速部署，以降低运维工作的时间和人力成本，提高开发测试效率。同时，Docker 轻量级虚拟化的特性可以给系统带来良好的可迁移性。系统的代码托管在 GitLab 仓库中，并配合 Jenkins 进行持续集成。相较于功能大而全的容器编排方案，Docker 与 Jenkins 的组合更加轻便，占用很少的服务器资源，更符合项目的实际情况。

```
def mutex_with_redis(lock_time):
    def inner_deco(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            key = 'aps_lock:%s:%s' % (func.__module__, func.__name__)
            val = str(datetime.datetime.utcnow())
            if redis_client.set(key, val, ex=lock_time, nx=True):
                func(*args, **kwargs)

        return wrapper

    return inner_deco

# Usage:
@mutex_with_redis(600)
def move_current_to_history():
    ...
```

图 4.16: 定时任务 Redis 锁装饰器

在具体实现中，使用 Docker 将应用服务器、评分服务器的软件环境与代码分别打包为镜像。部署工作由 GitLab 的 Hook 机制触发 Jenkins（同样使用 Docker 部署运行）进行。应用服务器节点与评分服务器节点均配置可使用 SSH 秘钥登录，并在开机时通过 Python 脚本访问 Jenkins 的 REST API 接口，将自身注册为 Jenkins 的从节点。开机注册为从节点的脚本关键代码如图 4.17所示。节点以自身 IP 生成名称，以便与其他节点区分，同时以标签为标志归属于应用服务器或评分服务器部署任务组。

当有新的代码被合并到生产分支时，Jenkins 管理器便按照部署任务设置中绑定的标签，通过 SSH 连接至各个节点拉取相应代码，并重新打包 Docker 镜像进行部署。在编写 Dockerfile 时需要将稳定的、改变频率较低的环境配置工作作为镜像底层，以使 Docker 充分使用缓存的中间层，加快镜像的打包速度。

4.5.2　提高可用性

（1）使用 Keepalived 提高 Nginx 可用性

为应对可能激增的业务压力，服务端使用过 Nginx 实现多服务器的负载均衡。由于应用服务器的逻辑与数据分离，负载均衡模式可灵活采用轮询、ip_hash 或权重等任意方式进行配置。这不仅可减小单台服务器节点的负载，而且当某一节点无法提供服务时，Nginx 可以快速将请求发送到其他服务器节点进行处理。同时，使用 Nginx 的反向代理和负载均衡功能可以实现服务的不停机更新，只需通过 uWSGI 和 Flask 以新的端口运行新版服务代码，然后将 Nginx 配置文件

```
import jenkins

jenkins_server = jenkins.Jenkins(host, username=name, password=pwd)

node_ip, node_name = get_my_ip_and_name()

for nd in jenkins_server.get_nodes():
    if nd.get('name') == node_name:
        break
else:
    # create node
    params = {
        'port': '22',
        'username': 'root',
        'credentialsId': 'c59ad161-bbd5-4c82-8f23-f6e432a07067',
        'host': node_ip
    }
    jenkins_server.create_node(
        node_name, nodeDescription='scoring node',
        remoteFS='/root/jenkins', labels='score-servers',
        exclusive=True, launcher=jenkins.LAUNCHER_SSH,
        launcher_params=params)
```

图 4.17: 开机时注册为 Jenkins 从节点

upstream 中的端口切换到新端口，令 Nginx 重新加载之即可。

Nginx 的反向代理和负载均衡简单实现了后端应用服务器的高可用，但单台 Nginx 负载均衡服务器可能造成系统的单点故障。对此，系统使用 Keepalived 搭建 Nginx 的高可用主备集群的解决方案。即多个 Nginx 服务器中均运行 Keepalived 以监控主 Nginx 服务器状态，主服务器的 keepalived 定时向备份机同步自身存活状态。主服务器 Nginx 意外结束时 Keepalived 也将结束自身生命，释放服务器所占用的虚拟 IP。备份机在一定时间内无法收到主服务器的存活消息时，即将自身与虚拟 IP 绑定实现切换动作。在云服务器进行部署时，需要云服务器商提供私有网络功能，VIP 的切换操作则通常需要调用其提供的路由表 API 接口完成。

（2）提高数据存储服务可用性

系统的数据存储服务主要包括 MongoDB、Redis 和 BOS 对象存储三部分。系统分别使用 Redis 的多哨兵模式和 MongoDB 的副本集模式提升数据存储服务的可用性。对于 Redis，所有 Redis 服务器上通过配置并运行 redis-sentinel 进程实现对 Redis 主服务器的监控，当多个哨兵监测到 Redis 主服务器不可用时会发起投票，于从服务器中选取新的 Master 服务器。MongoDB 的副本集模式是其官

方推荐的高可用机制，相较于主从模式副本集模式的各节点组成了互为主从的关系，数据被复制多份保存，且与 Redis 哨兵模式相似，当副本集的主节点宕机后，副本节点可选举出新的主节点，实现自动切换。

（3）提升静态文件处理能力

系统以前后端分离的模式开发，前端多个页面片段组成一个单页应用。用户首次访问系统首页时，浏览器会将所有静态资源下载至用户设备，此后通过应用服务器提供的 API 进行数据的传输。单页应用比多页应用有更好的页面响应速度和更平滑的用户体验，但由于所有静态资源体积较大，组件较多，首次加载需要等待较长时间。对此，系统从四个角度进行优化。

首先，浏览器 Vue 前端使用路由懒加载机制，使用 Webpack 的代码分割功能将单独的较大 JavaScript 代码块按路由分割为不同的多个较小代码块，当路由被访问时才加载相应的代码文件，以提高加载效率。其次，前端代码构建时使用 compression-webpack-plugin 实现 Gzip 压缩以减小代码打包体积，需要注意的是 Web 服务器 Nginx 也需要开启 Gzip 压缩功能才可正常通信。通过实际对比发现启用 Gzip 压缩可有效减少约 52% 文件体积。第三，前端页面优先使用设备已有字体，对于必须从服务器下载字体的情况需要做好字体裁剪，并使用 woff2 文件格式以减小字体文件体积。最后，将静态资源文件使用 CDN 托管，用户访问首页时可就近下载静态文件，同时可很大程度上过滤对后端服务器的带宽压力。根据实际使用的统计数据，CDN 平均为后端服务器过滤了 80% 以上的流量。

4.6　本章小结

本章首先介绍了表达力评测系统中各模块的详细设计与实现，包括权限管理模块、评测模块、试题管理模块和异步任务模块。通过顺序图、数据流图、关键代码展现以及实现效果截图等方式对各模块的关键功能做了详细阐述。最后从前后端工具链等角度对系统部署方式和可用性提升措施做了说明。

第五章　系统测试和实验评估

本文第三、四两章对系统设计与实现方案进行了阐述，本章将从功能和性能两方面对系统进行测试。功能测试用于验证各模块功能的完整性与正确性；性能测试主要验证系统首页、主服务端接口及评分节点面对高并发请求时的处理速度与正确性。

5.1　系统测试环境

系统各物理机部署详情如表 5.1所示。系统评分节点选用百度智能云提供的语音识别服务，为了利用内网带宽增加系统吞吐量，系统选用 BOS 对象存储服务，并将其他服务器一并部署于百度云服务器上。各服务器运行的操作系统均为 Ubuntu18.04 LTS，并配备 40G SSD 云盘，不再重复列出。

表 5.1: 测试环境说明

服务器类型	物理环境	软件环境
主服务端	2 台 通用性 g3 2 核 8G 4Mbps	Nginx 1.14.0 Docker 18.09.7 Vue App 2.6.10 Keepalived 1.3.9
数据服务器	3 台 密集计算型 ic3 4 核 4G 4Mbps	MongoDB 3.6.3 Redis 4.0.9
评分服务器	2 台 密集计算型 ic3 4 核 4G 1000Mbps	Python 3.6.9 Docker 18.09.7
部署服务器	1 台 密集计算型 ic3 2 核 2G 4Mbps	Docker 18.09.7 Jenkins 2.204.5
测试服务器 （临时）	1 台 密集计算型 ic3 4 核 4G 1000Mbps	ApacheBench 2.3 Jmeter 5.2.1

系统在架构设计上充分考虑了业务增长对扩展性和可维护性的需求，实现了各组件部署逻辑上的分离。但综合考虑项目初期系统负载压力与服务器资源成本，实际部署中将 Nginx 负载均衡服务器、后端 Web 应用以及前端 Vue App 静态资源文件部署于同一台物理机，负责数据存储的 MongoDB 和 Redis 也部署

于同一台物理机。Nginx 服务器需要另外部署一台相同环境的服务器用于热备，数据服务器则部署三台以实现 Redis 和 MongoDB 的一主二从结构，在保证可用性的同时便于主节点故障时顺利选举新的主节点。主服务端的 Docker 用于运行后端业务逻辑代码，评分服务器的 Docker 用于运行异步评分服务代码，Docker 外的 Python 用于运行开机脚本自动注册为 Jenkins 从节点。Jenkins 管理端部署于单独的部署服务器 Docker 环境中。

5.2　系统功能测试

5.2.1　权限管理模块功能测试

本小节对权限管理模块后端各功能进行测试，包括针对不同登录状态与用户角色对页面与接口访问权限的控制、邀请码生成与使用、BOS 对象存储服务 STS 凭证的发放等。

模块针对不同登录状态和用户角色提供的权限访问控制功能对应的测试用例如图 5.2所示。

表 5.2: 权限控制功能测试用例

编号	测试项目	输入/步骤	预期输出
TC1-1	拦截未登录访问	1. 直接通过浏览器请求评测页面和管理页面； 2. 使用 Postman 直接访问评测接口和管理接口。	1. 跳转至登录页面； 2. 拒绝访问，提示需要登录。
TC1-2	前端路由适配角色	1. 以普通用户登录； 2. 重新以管理员登录。	1. 页面仅生成普通用户菜单； 2. 页面仅生成管理员用户菜单。
TC1-3	拦截越权访问	1. 普通用户通过浏览器访问管理员页面； 2. 普通用户登录后使用 Postman 访问后端管理员接口。	1. 跳转至首页； 2. 拒绝访问，提示无权限。

模块提供的 STS 凭证管理功能主要用于缓存服务端从百度智能云获取的 STS 凭证，并在客户端需要上传文件至 BOS 时返回凭证。该部分功能对应测试用例如表 5.3所示。该模块提供函数接口供其他模块进行调用，对于非必要请求的拦截需由调用者自行处理，不包括在该模块的测试范围内。

邀请码功能对应测试用例包括前端页面邀请码创建和历史邀请码查询功能的测试，以及后端接口中邀请码创建、使用、历史邀请码查询接口的测试。对于后端接口的测试，直接使用 Postman 软件发送请求，对于前端页面的测试则直接在浏览器中进行。如表 5.4所示。

表 5.3: STS 凭证管理功能测试用例

编号	测试项目	输入/步骤	预期输出
TC1-4	STS 凭证管理	1. 模拟普通用户进行测试； 2. 访问 STS 凭证获取接口； 3. 使用返回的 STS 凭证访问 BOS。	1. 正确返回 STS 凭证； 2. 成功访问 BOS。

表 5.4: 邀请码功能测试用例

编号	测试项目	输入/步骤	预期输出
TC1-5	邀请码创建后端接口	1. 以管理员身份登录并保持 token 信息； 2. 发送参数正确的创建请求至后端接口； 3. 发送各种可能的异常输入至后端接口。	1. 执行合理的创建请求并返回邀请码； 2. 拒绝执行不合理的创建请求，并返回参数错误提示信息。
TC1-6	邀请码创建前端页面	1. 管理员在邀请码管理页面正确填写待创建的邀请码信息； 2. 点击创建按钮发送创建请求； 3. 尝试各种可能的异常输入； 4. 点击创建按钮发送创建请求。	1. 正确处理用户输入，对不符合要求的参数给出提示； 2. 用户点击创建按钮时进行参数校验； 3. 对未通过校验的参数给出提示，拒绝发送请求； 4. 正确发送通过参数校验的创建请求至后端接口； 5. 正确显示后端返回的邀请码。
TC1-7	邀请码查询后端接口	1. 以管理员身份登录并保持 token 信息； 2. 发送查询条件合理的请求至后端接口； 3. 发送各种查询条件不合理的请求至后端接口。	1. 执行合理的查询请求并返回查询结果； 2. 拒绝不合理的查询请求，并返回参数错误提示消息。
TC1-8	邀请码查询前端页面	1. 管理员在邀请码管理页正常输入邀请码查询条件； 2. 点击查询按钮进行查询； 3. 尝试各种可能的异常输入； 4. 点击查询按钮进行查询。	1. 正确处理用户输入的查询条件，对不符合要求的条件应给出提示； 2. 用户点击查询按钮时正确校验查询条件； 3. 对不符合要求的查询条件给出提示，拒绝发送请求； 4. 正确发送通过查询条件校验的查询请求至后端接口； 5. 正确显示后端接口返回的查询结果。

见下页

表 5.4 – 邀请码功能测试用例（续表）

编号	测试项目	输入/步骤	预期输出
TC1-9	邀请码应用功能	1. 考生在个人信息页输入正确的邀请码并确认使用； 2. 输入错误的邀请码并确认使用。	1. 后端接口执行正确的邀请码应用请求，返回更新后的用户评测权限； 2. 后端接口拒绝执行错误的邀请码应用请求，返回邀请码无效提示； 3. 前端页面显示邀请码应用状态与用户评测权限。

5.2.2　评测模块功能测试

系统评测模块主要功能是引导用户正确完成评测。需测试的功能包括预测试、正式评测、评测练习等。预测试和正式评测功能测试用例如表 5.5所示。评测练习可看作仅包含一道转述题目的正式评测，在表现形式上与正式评测相同，不再重复给出测试用例。此外，不同于后台管理功能仅支持浏览器前端页面，评测功能同时支持浏览器前端页面和微信小程序接入，对涉及前端应用的测试需在浏览器页面和微信小程序中分别进行测试。

表 5.5: 评测功能测试用例

编号	测试项目	输入/步骤	预期输出
TC2-1	预测试	1. 在客户端点击开始测试； 2. 正常朗读题目内容； 3. 再次测试，读与题目不相关内容。	1. 正常展示预测试题目； 2. 提示用户预测试通过； 3. 提示用户预测试未通过。
TC2-2	评测初始化	1. 正常开始评测并记录做到的题目； 2. 重复步骤 1 多次，对比题目是否有重复。	1. 服务端正确初始化测试，并将测试数据存入数据库； 2. 题目用尽之前不应对同一用户使用重复题目。
TC2-3	断点续测	1. 开始正式测试； 2. 退出系统或关闭测试页； 3. 重新进入评测页面并通过预测试。	1. 服务端正确检测到用户的未完成测试； 2. 客户端提示用户继续测试。
TC2-4	题目获取与展示	用户正常进行评测。	1. 客户端从后端接口取得题目； 2. 客户端正确控制题目提示信息与正文的显示与隐藏。
TC2-5	作答时间控制	1. 正常进行评测，主动结束回答； 2. 正常进行评测，等待题目超时自动结束。	1. 客户端正确显示题目作答倒计时； 2. 若倒计时结束用户仍未完成作答，自动结束回答并进入下一题。

见下页

表 5.5 – 评测功能测试用例（续表）

编号	测试项目	输入/步骤	预期输出
TC2-6	音频采集与上传	1. 用户正常进行评测； 2. 作答结束后点击完成，或等待超时自动进入下一题。	1. 客户端采集用户作答音频； 2. 客户端向服务端请求音频上传路径与 STS 凭证，服务端正确返回； 3. 客户端上传音频至 BOS。
TC2-7	异步任务创建	1. 每题作答结束后客户端正常上传题目至 BOS； 2. 客户端告知服务端音频已上传，请求处理。	服务端正确调用异步任务模块生成任务。
TC2-8	评测结果获取	用户正常完成评测，客户端页面自动跳转至结果页。	1. 客户端轮询评测结果； 2. 服务端正确返回结果数据； 3. 客户端正确展示结果数据。

5.2.3　试题管理模块功能测试

试题管理模块完成的任务有二：一是为评测模块提供测试题目初始化功能，二是为管理员对题目的管理提供支持。其中对于测试题目初始化功能，需要保证在用户做完所有题目之前不会遇到重复题目，对此功能的测试通过编写测试函数直接调用题目初始化接口进行。此外，根据需求分析结果与具体实现，需要前后端联合对题目的反馈收集功能与题目的查询修改等功能进行测试。进行试题反馈功能测试时应注意反馈的取消操作时以及题目点赞与点踩进行切换时后端数据是否可正确处理。该模块对应的测试用例如表 5.6所示。

表 5.6: 试题管理模块测试用例

编号	测试项目	输入/步骤	预期输出
TC3-1	题目初始化接口	1. 传入用户 id 与各类型题目数量进行调用，记录返回题目编号； 2. 重复多次调用，观察题目是否重复。	1. 接口正确返回各类型试题； 2. 多次测试不返回重复题目。
TC3-2	试题反馈功能	1. 以普通用户身份进入评测； 2. 评测时点击试题反馈按钮进行反馈； 3. 尝试反馈的取消与重新反馈操作； 4. 点赞与点踩直接切换； 5. 查询并确认题目数据。	1. 页面正确展示反馈状态； 2. 后端接口正确更新题目数据。
TC3-3	题目查询功能	1. 管理员进入试题管理首页； 2. 输入题目编号并点击查询。	1. 自动查询并分页展示题目； 2. 正确处理单题查询与显示。

见下页

表 5.6 – 试题管理模块测试用例（续表）

编号	测试项目	输入/步骤	预期输出
TC3-4	题目修改功能	1. 点击题目进入修改页面，或在查看页面点击修改按钮； 2. 用户进行题目修改； 3. 用户确认修改操作。	1. 正确加载并显示当前题目； 2. 页面正确响应用户修改操作； 3. 后端正确修改题目； 4. 页面重新加载题目信息。
TC3-5	题目添加功能	1. 用户点击添加题目按钮； 2. 用户编辑题目； 3. 用户确认添加或丢弃。	1. 正确显示添加页面； 2. 页面正确响应题目编辑操作； 3. 正确处理添加或丢弃操作并提示状态。
TC3-6	题目导入功能	1. 用户点击导入题目按钮； 2. 用户编辑题目； 3. 用户确认添加或丢弃。	1. 正确显示加载备用题目并显示； 2. 页面正确响应用户编辑操作； 3. 正确处理添加或丢弃操作并提示状态。
TC3-7	题目删除功能	1. 在题目查看也点击删除按钮； 2. 确认或放弃删除操作。	正确处理删除或放弃操作并提示状态。

5.2.4　异步任务模块功能测试

异步任务模块主要负责异步评分任务和定时任务的调度与管理。对于异步评分任务，前端应用仅负责提交任务至后端接口，限于篇幅略去对其测试用例的描述，仅对后端接口的任务创建与评分节点的任务获取进行测试，该功能使用 Celery 监控工具 flower 组件协助测试。该工具可以实时显示 Celery 任务队列数量与状态，并可对队列中任务的执行状态、执行时间、返回结果等信息进行详细跟踪。Flower 页面部分功能元素如图 5.1所示。

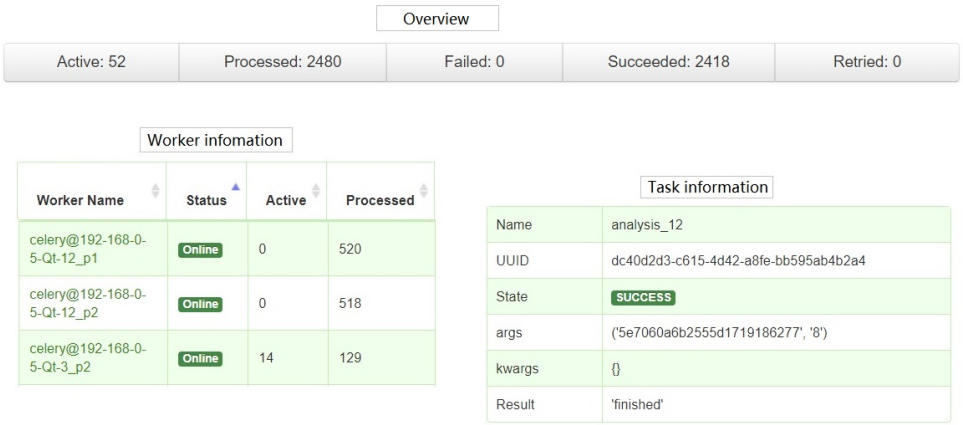

图 5.1: Flower 部分信息示例

对于定时任务相关的功能，本小节将从任务执行、任务恢复与任务管理等方面进行测试，对定时任务是否正确执行通过函数日志进行判断。该模块对应

的测试用例如表 5.7所示。

表 5.7: 异步任务模块测试用例

编号	测试项目	输入/步骤	预期输出
TC4-1	评分任务创建与获取	1. 传入 CurrentTest 数据库 id、题目编号、题目类型参数调用评分任务创建接口； 2. 通过 Flower 查看任务队列变化情况和任务执行结果。	1. 接口返回任务的 AsyncResult ID，错误信息为 None； 2. Flower 界面活动任务数增加 1，任务 UUID 与步骤 1 获取的 ID 相同； 3. 通过 Flower 界面查得任务被执行。
TC4-2	避免评分任务重复创建	1. 以相同参数重复多次调用评分任务创建接口； 2. 通过 Flower 查看任务队列变化情况。	1. 第一次请求返回正确 AsyncResult ID，错误信息为 None； 2. 后续请求不返回 ID，错误信息为“Failed to get redis lock”； 3. Flower 界面活动任务数仅增加 1.
TC4-3	定时任务调度	1. 设定定时任务并等待至设定执行时间； 2. 检查两台应用服务器日志，查看任务是否执行。	1. 一台应用服务器在指定时间执行一次定时任务； 2. 其余线程显示获取 redis 锁失败，放弃执行。
TC4-4	定时任务管理页面	1. 管理员进入定时任务管理页面； 2. 管理员查看、修改定时任务； 3. 监测定时任务是否正确按时执行。	1. 页面正确显示定时任务信息； 2. 定时任务被正确修改，于正确时间执行。
TC4-5	定时任务自动恢复	1. 设定定时任务； 2. 重启应用服务器； 3. 等待任务执行，记录执行时间。	定时任务执行时间为初始设定时间。

5.2.5　自动部署功能测试

为应对突发的高并发访问，系统具有多个应用服务器与多个评分节点。为提高部署效率，系统应用服务器和评分服务器部署完成后分别创建用于部署的光盘镜像，需要创建新节点时直接从镜像创建即可。

表 5.8: 自动部署功能测试

编号	测试项目	输入/步骤	预期输出
TC5-1	从节点自动注册	1. 从评分服务器镜像新建评分服务器节点并启动； 2. 登录 Jenkins 管理页面查看从节点变化情况； 3. 手动触发评分节点构建。	1. 发现新增从节点，名称为新节点 IP，标签与其他评分服务器节点相同； 2. Jenkins 在新节点成功部署评分服务。

新节点每次开机后自动向部署服务器上的 Jenkins 管理端检查自身是否已注册成为从节点，若尚未注册则调用 Jenkins API 进行自动注册。当生产环境代码更新时，由 Jenkins 连接至从节点进行自动化部署。该部分自动部署脚本开发与配置工作虽不直接为普通用户与管理员用户提供功能，但在提高系统开发效率、减少部署成本等方面发挥了重要作用。新建服务器节点的开机自动注册从节点功能对应的测试用例如表 5.8所示。

5.3　系统性能测试

本节中，我们将对系统性能进行测试。包括对首页吞吐量的压力测试，对评测接口的压力测试，以及通过实验进行评估等。

5.3.1　系统首页压测

网站首页的加载速度对用户体验影响很大。本系统将前端页面加载方式设计为分步加载，即用户访问系统首页时浏览器仅下载一个体积较小的框架页面，框架页面下载后再通过 JavaScript 加载其他静态资源，以提升用户体验。进行首页的吞吐量测试时，我们首先直接对框架页面吞吐量进行测试，然后使用 tar 命令将首页完全加载所需的全部静态资源打包为一个档案文件，并对系统处理该档案文件的吞吐量进行测试。系统首页使用 CDN 进行缓存加速，我们将分别测试关闭与启用 CDN 两种情况。

对首页的压测通过 ApacheBench 工具完成。该工具提供一个命令行工具并以控制台输出形式打印测试结果，其通过创建大量并发访问线程来模拟多用户对同一 URL 的访问。使用 ApacheBench 进行测试时，需要指定的主要参数有三个：访问总量，访问并发数，及待测 URL。当访问总量与访问并发数过大时，可对被测服务器造成巨大压力，故这两个参数需要从小到大逐渐增加，待系统吞吐量不再继续增加时可结束测试，以最终结果为测试结果。

为减小测试机本身带宽等因素的影响，我们使用临时的测试服务器安装 ApacheBench 进行测试。对首页框架页面的压测数据如表 5.9所示。

表 5.9: 系统首页框架页面压测结果

CDN 状态	请求数量 (#)	每秒处理 (#/sec)	平均响应 (ms)	95% 完成 (ms)	传输速率 (kB/sec)
开启	1000	833.52	1.2	554	1797.62
关闭	1000	735.71	1.359	740	1693.43

通过测试结果可知，无论是否开启 CDN 缓存加速，系统处理首页框架页面请求 TPS（每秒处理事务数）均超过 700（#/sec），可以提供较好的首屏响应体验。然后，首页框架页面需要加载静态资源，系统对这些静态资源请求的处理能力压测数据如表 5.10所示。

表 5.10: 系统首页全部资源压测结果

CDN 状态	请求数量 (#)	每秒处理 (#/sec)	平均响应 (ms)	95% 完成 (ms)	传输速率 (kB/sec)
开启	1000	69.83	14.32	4477	167623.56
关闭（外网）	1000	1.07	931.155	225221	2577.78
关闭（内网）	1000	70.86	14.112	3840	170091.17

通过测试结果可知，在首屏完成完全加载的过程中，开启与关闭 CDN 对系统吞吐量有巨大的影响。未开启 CDN 时，直接通过内网 IP 进行压测时，系统对首页全部静态资源请求的吞吐量达到约 70（#/sec），仅需 3.84 秒即可完成 950 个静态资源请求，可以提供较好的首页加载体验。但通过外网进行测试时，受限于服务器的 4M 公网带宽，系统吞吐量仅为 1.07（#/sec），难以提供满意的首页加载体验。开启 CDN 时，通过外网进行压测结果与直接通过内网 IP 压测相似，TPS 同样达到约 70（#/sec），4.5 秒内完成 950 个静态资源请求，可以提供较好的首页加载体验。该测试也说明了系统使用的 CDN 缓存良好地解决了小带宽带来的静态资源处理问题。

5.3.2　系统接口响应速度压测

评测功能是系统面向最广大用户提供的主要功能，其相关接口也是系统并发压力的主要来源，评测接口的响应速度很大程度上决定了用户的系统使用体验，该小节我们将使用 Jmeter 对评测功能的接口进行压测。为使结果准确，我们同样使用单独的 Linux 测试服务器安装 Jmeter，并按照并发数 100tps，200tps，500tps 对系统进行测试。并发数量为 500tps 时，Jmeter 测试结果如表 5.11所示；测试过程中服务器负载如表 5.12所示，其中 DISK 选项是主数据服务器的写盘速度，其他选项数据则来自于应用服务器。

通过上述测试结果可知在 500tps 并发条件下，评测功能各接口性能表现良好，请求失败率为 0%，评测初始化接口需要较多的数据读写与去重操作，99% 的请求在约 8 秒内得到应答，满足非功能性需求中 10 秒响应时间的限制；其他接口响应速度均较快，99% 的请求均在 1 秒左右得到应答，满足非功能性需求

表 5.11: 评测功能接口压测结果

API	Executions	Response Time （ms）				Throughput	Network（kB/sec）	
Label	Fail	Avg	Min	Max	99%	TPS	Received	Sent
init.exam	0%	3276.88	319	8011	7968.49	49.76	16.18	10.78
next.question	0%	79.94	11	1016	953.99	75.32	24.79	10.45
upload.url	0%	96.38	21	1069	982.87	70.52	29.48	9.71
upload.success	0%	115.15	22	1151	1011.84	78.35	40.93	17.52
result	0%	109.67	24	988	921.93	75.98	36.73	10.02

表 5.12: 评测接口压测 - 服务器负载

API Label	CPU	MEM（GB）	MEM%	DISK（kB/s）
init.exam	14.10%	2.52	33.38%	113.07
next.question	7.89%	2.52	32.37%	66.93
upload.url	8.94%	2.55	33.23%	62.93
upload.success	4.76%	2.55	33.22%	59.6
result	4.07%	2.55	33.18%	39.07

中 3 秒响应时间的限制。在服务器负载方面，系统的 CPU 占用始终在 20% 以下，内存占用则稳定在 33% 左右。由于 MongoDB 对内存的充分使用，数据服务器磁盘的读写均较少。综合来说，系统的资源消耗表现良好。

5.4　系统实验评估

为初步评估系统评测功能在实际使用中的功能与性能表现，我们邀请得到大学第一期学生参加表达力评测，对系统进行了实验评估。本次实验评估用时两天，共计发放邀请码 3000 个，实际使用 515 个，共计测试次数 532 次，统计结果如图 5.2所示。其中测试成绩正常且测试过程中日志无报错的有 431 次，测试结果中部分题目得分为零的测试共计 101 次，其中 88 次是因考生中途主动结束所致，其余 13 次通过查验日志发现均为评分过程中音频文件识别结果无效，进一步聆听这 13 次测试中用户原始作答音频与预测试音频，发现这些测试中用户预测试音频均清晰正常，而得分为零的题目作答音频中均无有效作答内容（包括无人声、与他人交谈而未作答及作答内容与题目完全不相关等情况），但背景噪声仍控制在较好的范围内，说明预测试对拾音质量起到了较好的保障作用，系统评测功能与各组件协作正常。

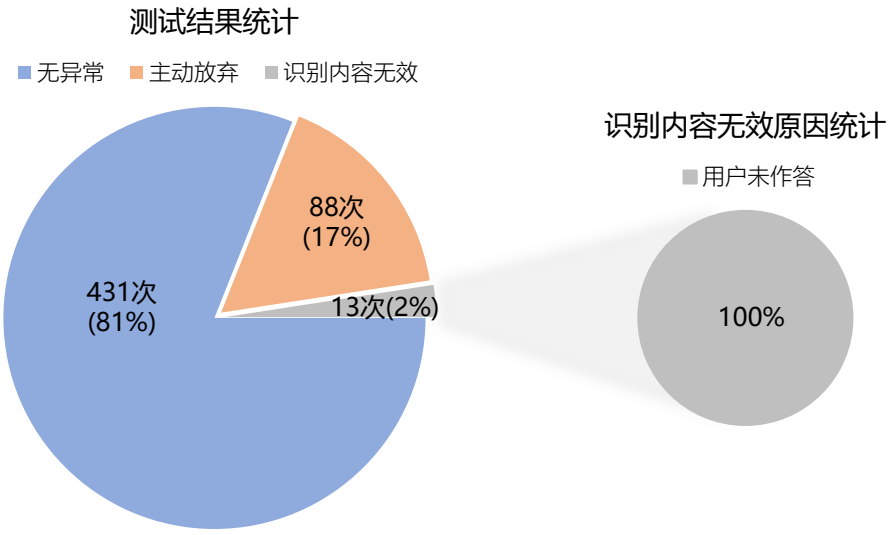

图 5.2: 实验评估结果统计

测试过程中，系统应用服务器和评分节点服务器的峰值负载情况如表 5.13所示，由于实际实验评估过程中由用户自行选取测试时间，系统并发请求数较低。通过实验数据可知各服务器负载始终处于较低水平，资源占用情况良好。

表 5.13: 实验评估 - 服务器负载

Item	CPU	MEM(GB)	MEM%	Network In(kB/s)	Network Out(kB/s)
应用服务器	10.68%	2.28	29.69%	20.71	192.14
评分服务器	3.77%	2.01	52.34%	101.69	169.00

5.5　本章小节

结合系统需求分析与系统部署的软硬件情况，本章首先对系统各功能模块进行了功能测试，然后分别使用 ApacheBench 和 Jmeter 对系统的首页的首页请求处理能力和评测接口性能进行了测试，最后通过邀请用户使用系统进行实际测试对系统评测功能与性能进行了实验评估。最终结果表明系统各功能部件运行良好，系统运行中资源占用情况良好，性能表现达到预期要求。

第六章　总结和展望

6.1　总结

基于快速表达力测试方法对软件系统的实际需求，本文设计并实现了基于分布式任务队列的表达力评测系统。该系统可为用户进行快速表达力测试及获取成绩报告提供软件支持，并能够让管理员围绕表达能力测试完成用户权限、题目、系统任务等各种管理工作。同时，相较于人工组织测试与人工评判，本系统可以更为考生提供更便捷、快速的测试体验，对于测试主办方，使用本系统可节约大量时间和人力成本，让组织大规模的测试成为可能。

本文首先从项目的背景、项目愿景和预见的使用场景等方面对系统进行了整体概述。然后分析了系统的业务需求与可预见的使用场景，进而得到系统的功能性需求与非功能性需求。接下来本文结合网络系统开发中常用的技术方案，确定了本系统开发的关键技术：系统基于网络技术构建，并支持多终端设备，使用户可以方便快速地使用系统；使用任务队列实现分布式系统，采用逻辑与数据分离的策略，数据集中存储，主服务端与评分服务均可以集群的方式进行部署和扩展；采用前后端分离的开发模式并采用 RESTful 接口通信，使得系统服务端和客户端可以单独进行开发，并降低了系统开发的复杂度。在此基础上，本文给出了系统结构与功能模块划分，使用 4+1 视图对系统总体设计进行了描述，并给出了系统持久化数据模型的设计。

在系统具体设计与实现中，本文对逐个功能模块进行了具体描述，以图表的形式进行补充说明，讲述了系统功能的实现细节与模块交互关系，并给出了各模块主要功能的核心代码或简化后的伪代码，对部分功能的前端页面实现效果使用图片进行了展示。在具体设计与实现章节的最后，本文对系统部署相关内容进行了详细说明，并给出了集群各节点自动注册为 Jenkins 从节点的代码，对系统快速部署和可用性的提升措施进行了描述。

在系统测试与实验评估环节，本文首先设计测试用例对系统各功能进行了测试，然后使用 ApacheBench 和 Jmeter 对系统性能与资源占用情况进行了测试，最后邀请用户使用系统，根据用户反馈与系统表现数据对系统功能和实际性能进行了评估。系统各功能模块能够较好地完成相应功能，系统性能达到预期要求，且资源占用情况良好。

6.2　展望

本系统已经成功部署并能够稳定地对外提供表达力评测服务，为快速表达力测试方法提供了良好的软件支持。但本系统仍在诸多方面存在可改进的地方：

（1）当前用户使用系统评测时，需要客户端完整采集用户作答音频后进行上传，为减少大量用户同时提交答案时的评分等待时间，未来可针对其进行优化，实现用户边作答边评分，减少用户获取成绩的等待时间。

（2）目前系统对于用户评测练习的试题推荐功能仅基于用户喜好进行协同过滤，尚不具有根据考生水平推荐相匹配难度题目的能力。对此我们提出了基于题目分数分布峰度与偏度的题目难度分类与推荐方法，需要进一步落实到系统，实现更好的计算机适应性测试效果。

（3）除语音外，表述者的表情与动作同样影响着表达能力，当前系统仅适用于对考生口语表达能力的量化，未来可采用视频分析获取更多维度的信息，对用户的表达能力给出更多维度的量化。

（4）目前系统的定时任务需要系统代码实现相应的功能，无法动态进行新任务的添加。未来可对其进行优化，让管理员可直接在前端页面自行编写任务脚本，实现新任务的动态添加。

参考文献

[1] 张文忠, 吴旭东, 第二语言口语流利性发展定量研究, 现代外语 (4) (2001) 341–351.

[2] 任筱萌, 高等 hsk 命题、评分过程再解析, 汉语学习 (2) (2000) 49–53.

[3] 余胜泉, 何克抗, 网络题库系统的设计与实现, 中国远程教育 (9) (2000) 53–57+64.

[4] 李小勇, 王瑛, 题库管理系统中的自动化组卷算法 generating paper algorithms in paper management system, 西北师范大学学报 (自然科学版) 038 (4) (2002) 41–43.

[5] D. J. Weiss, G. G. Kingsbury, Application of computerized adaptive testing to educational problems, Journal of Educational Measurement 21 (4) (1984) 361–375.

[6] J. D. Brown, Computers in language testing: Present research and some future directions., Language Learning & Technology 1 (1) (1997) 44–59.

[7] 李清华, 基于纸笔的语言测试与基于计算机的语言测试的等效研究综述, 外语界 (4) (2006) 75–80.

[8] A. Loukina, K. Zechner, L. Chen, M. Heilman, Feature selection for automated speech scoring, in: Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications, 2015, pp. 12–19.

[9] Y. Attali, Reliability-based feature weighting for automated essay scoring:, Applied Psychological Measurement 39 (4) (2015) 303–313.

[10] P. H. E. Jr., What is a ”distributed” data processing system?, IEEE Computer 11 (1) (1978) 13–21.

[11] A. S. Tanenbaum, M. van Steen, Distributed systems - principles and paradigms, 2nd Edition, Pearson Education, 2007.

[12] S. Ghemawat, H. Gobioff, S. Leung, The google file system, in: M. L. Scott, L. L. Peterson (Eds.), Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003, ACM, 2003, pp. 29–43.

[13] I. Sadooghi, S. Palur, A. Anthony, I. Kapur, K. Belagodu, P. Purandare, K. Ramamurty, K. Wang, I. Raicu, Achieving efficient distributed scheduling with message queues in the cloud for many-task computing and high-performance computing, in: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014, IEEE Computer Society, 2014, pp. 404–413.

[14] 梅雪峰, 赵文静, 基于消息队列和 web 服务的分布式系统异步交互方式体系架构, 西北大学学报：自然科学版 34 (6) (2004) 655–658.

[15] C. Pautasso, O. Zimmermann, F. Leymann, Restful web services vs. "big"' web services: making the right architectural decision, in: J. Huai, R. Chen, H. Hon, Y. Liu, W. Ma, A. Tomkins, X. Zhang (Eds.), Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008, ACM, 2008, pp. 805–814.

[16] F. Belqasmi, J. Singh, S. Y. B. Melhem, R. H. Glitho, Soap-based vs. restful web services: A case study for multimedia conferencing, IEEE Internet Comput. 16 (4) (2012) 54–63.

[17] M. Grinberg, Flask Web Development - Developing Web Applications with Python, O'Reilly, 2014.

[18] S. Chickerur, A. Goudar, A. Kinnerkar, Comparison of relational database with document-oriented database (mongodb) for big data applications, in: 2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA), 2015, pp. 41–47.

[19] Y. Gu, X. Wang, S. Shen, J. Wang, J.-U. Kim, Analysis of data storage mechanism in nosql database mongodb, in: 2015 IEEE International Conference on Consumer Electronics - Taiwan, 2015, pp. 70–71.

[20] K. Banker, MongoDB in Action, Manning Publications Co., 2011.

[21] D. Crockford, The application/json media type for javascript object notation (json), RFC 4627 (2006) 1–10.

[22] K. Kaur, R. Rani, Modeling and querying data in nosql databases, in: 2013 IEEE International Conference on Big Data, 2013, pp. 1–7.

[23] E. You, Vue.js: the past and the future, https://www.youtube.com/watch?v=EiTORdpGqns (2016).

[24] G. Psaila, Virtual DOM: an efficient virtual memory representation for large XML documents, in: 19th International Workshop on Database and Expert Systems Applications (DEXA 2008), 1-5 September 2008, Turin, Italy, IEEE Computer Society, 2008, pp. 233–237.

[25] W. Reese, Nginx: the high-performance web server and reverse proxy, Linux Journal 2008 (173) (2008) 2.

[26] C. Boettiger, An introduction to docker for reproducible research, Operating Systems Review 49 (1) (2015) 71–79.

[27] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, Linux Journal 2014 (239) (2014) 2.

[28] C. Kan, Docloud: An elastic cloud platform for web applications based on docker, in: 2016 18th International Conference on Advanced Communication Technology (ICACT), 2016, pp. 1–1.

[29] M. Meyer, Continuous integration and its tools, IEEE Software 31 (3) (2014) 14–16.

[30] P. Kruchten, Architecture blueprints—the "4+1" view model of software architecture, in: Tutorial proceedings on TRI-Ada '91, 1995, pp. 540–555.

简历与科研成果

基本情况 褚东宇，男，汉族，1991 年 10 月出生，山东省枣庄市人。

教育背景

2018.9 ~ 2020.6 南京大学软件学院 硕士

2010.9 ~ 2014.6 东南大学机械工程学院 本科

读研期间成果

1. 马徐骏，刘嘉，詹晨，孟磊，王浩宇，**褚东宇**，汤大业，王磊，“一种为快速表达力测试生成题目的方法”，申请号：202010254294.6，已受理。
2. 马徐骏，刘嘉，詹晨，孟磊，王浩宇，**褚东宇**，汤大业，王磊，“一种动态优化表达力评测题目的方法”，申请号：202010254255.6，已受理。
3. 马徐骏，**褚东宇**，汤大业，刘嘉，詹晨，孟磊，王浩宇，王磊，“一种用于快速表达力测试的智能化题目分类及推荐方法”，申请号：202010287246.7，已受理。

致　　谢

日月如流。两年短暂而充实求学时光里，许多老师和同学给予了我无私的教导、关心和帮助。在论文完成之际，我想向他们致以最诚挚的感谢。

感谢刘嘉老师和陈振宇老师在我的学习、项目开发以及论文写作过程中给予我全面支持和悉心指导，感谢他们给予我的督促和帮助，他们真诚严谨的人格魅力与极强的行动力让我受益终生。

感谢 iSE 实验室的黄勇老师、房春荣老师，以及各位师兄师姐。他们在项目开发过程中给予我许多指导，传授了诸多经验并提供了许多资源。

感谢与我共同进行项目开发的汤大业同学和项目测试阶段提供有力帮助的李林昱同学。感谢他们能够认真负责，与我一起完成项目。

感谢学院和母校为我提供了优秀的教学资源和良好的基础设施，感谢学院两年来对我的培养，感谢学院老师们对我孜孜不倦的教诲。

感谢我的家人一直以来的鼓励和支持，他们虽然不能给予我项目工作上的指导，但他们在精神上为我提供力量，在生活上为我提供帮助，他们给我的关怀和照顾是无可比拟的。

最后感谢每一位参与本文答辩评审的老师，感谢你们的辛苦工作。

版权与原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期：2020 年 5 月 27 日