

2022 年英特尔杯大学生电子设计竞赛嵌入式系统专题邀请赛

2022 Intel Cup Undergraduate Electronic Design Contest

Embedded System Design Invitational Contest

## 基于声纹识别的语音控制器

# 设计报告



Intel Cup Embedded System Design Contest

项目成员： 付东源、姚辰龙、朱子虚

指导教师： 孙文生

参赛学校： 北京邮电大学

## 2022 年英特尔杯大学生电子设计竞赛嵌入式系统专题邀请赛

### 参赛作品原创性声明

项目团队郑重声明：所呈交的参赛作品报告，是项目成员（付东源、姚辰龙、朱子虚）独立进行研究工作所取得的成果。除文中已注明的引用内容外，本报告不包含任何其他个人或集体已经发表或撰写过的作品成果，不侵犯任何第三方知识产权或其他权利。项目团队完全意识到本声明的法律结果由项目团队承担。

参赛队员签名： 付东源 姚辰龙 朱子虚

2022 年 7 月 25 日

## · 摘 要 ·

**声纹**（Voiceprint）是语音中蕴含的、可表征和标识说话人的生物识别特征，以及基于这些特征和参数所建立的语音模型的总称。**声纹识别**（Voiceprint Recognition, VPR），又称说话人识别（Speaker Recognition），是一种根据语音信号中能够表示说话人信息的声纹特征，利用计算机以及各种信息识别技术，自动确认或分辨说话人身份的生物特征识别技术。

随着信息技术的飞速发展和各类智能终端的应用普及，生物特征识别技术广泛应用在各类身份验证场景。声纹作为一种长期稳定、隐私保护性强、不受距离限制、带有生理特性的特殊行为特征，其便捷性与安全性在众多认证方式中脱颖而出。本项目主要研究**声纹识别算法和语音识别技术**，并利用 GNU-V40 平台（下称“竞赛平台”）实现一个**具有身份认证功能的语音声控制器**，该控制器可以接受特定人发出的指令，并驱动执行机构完成相应的操作。该项目要完成的具体任务如下：

为提升声纹模型训练过程的收敛速度和模型性能，提出一种 Softmax 交叉熵损失函数超参数优化算法，该算法通过在每次迭代时调整超参数比例因子和间距提升模型的收敛速度，找到网络达到性能最优的参数值，提升说话人网络模型的性能。为降低模型的资源利用率，使用深度可分离卷积替换一维扩张卷积，构建轻量级模型；使用 ReLU6 替换 ReLU 激活函数避免嵌入式平台运行时的精度损失；通过模型量化技术将模型中浮点实数以单字节整数格式存储，并充分利用硬件所支持的整数运算指令集，提升运算速度。

对于项目中的语音识别系统，基于主流开放平台的“短语音识别”服务进行快速构建。即通过调用网络接口，获取预先录制的说话人音频文件所对应的文本内容，进而进行语义分析。项目系统采集并存储说话人的语音信号样本后，声纹识别系统首先处理样本，通过与声纹数据库进行比对，确认说话人身份。系统则根据说话人身份确认结果，执行后续操作。

**关键词：**声纹识别，深度学习，身份认证，语音识别

## Abstract

Voiceprint is a general term for the biometric features contained in speech that can represent and identify the speaker, as well as the speech model based on these features and parameters. Voiceprint recognition, also known as speaker recognition, is a biometric recognition technology that automatically confirms or distinguishes the speaker's identity by using computers and various information recognition technologies according to the voiceprint features that can represent the speaker's information in the speech signal.

With the rapid development of information technology and the popularity of various intelligent terminals, biometric technology is widely used in various authentication scenarios. Voiceprint, as a special behavior feature with long-term stability, strong privacy protection, unlimited distance and physiological characteristics, its convenience and security stand out among many authentication methods. This project mainly studies voiceprint recognition algorithm and speech recognition technology, and uses the GNU-V40 platform to complete the implementation of a voice controller with identity authentication, which can accept the instructions sent by specific people and drive the actuator to complete the corresponding operations. The specific tasks of the project are as follows:

In order to improve the convergence speed and model performance of voiceprint model training process, a super parametric optimization algorithm of Softmax cross entropy loss function is proposed. This algorithm improves the convergence speed of the model by adjusting the super parametric scale factor and spacing at each iteration, finds the parameter value of the network to achieve the optimal performance, and improves the performance of the speaker network model. In order to reduce the resource utilization of the model, the depth separable convolution is used to replace the one-dimensional expansion convolution, and a lightweight model is constructed; Replace the relu activation function with relu6 to avoid the loss of accuracy when the embedded platform is running; The floating-point real numbers in the model are stored in single byte integer format through model quantization technology, and the integer operation instruction set supported by hardware is fully utilized to improve the operation speed.

The speech recognition system in the project is quickly implemented with short speech recognition service on the mainstream open platform. That is, by calling the network interface, the text content corresponding to the pre recorded speaker audio file is obtained, and then semantic

analysis is carried out. After the project system collects and stores the voice signal samples of the speaker, the voiceprint recognition system first processes the samples and confirms the identity of the speaker by comparing them with the voiceprint database. The system performs subsequent operations according to the speaker's identity confirmation results.

**Keywords:** Voiceprint Recognition, Deep Learning, Identity Authentication, Speech Recognition

## 。 目 录 。

(1) 项目背景	6
(1.1) 项目背景与意义	6
(1.1.1) 项目背景	6
(1.1.2) 项目研究意义	8
(1.2) 声纹识别发展趋势	9
(1.2.1) 研究趋势：深度学习效果拔群，时变鲁棒成为焦点	9
(1.2.2) 应用趋势：国标与商用并进，于实践中积累经验	9
(1.3) 项目应用前景	11
(2) 项目设计方案	12
(2.1) 需求分析	12
(2.2) 系统架构与技术选型	13
(2.3) 概要设计	15
(2.3.1) 改进的时延神经网络文本无关声纹识别算法	15
(2.3.2) 基于 2.3.1 中算法的声纹识别语音控制器系统	18
(2.4) 详细设计	20
(2.4.1) 综述	20
(2.4.2) 用户终端应用	22
(2.4.3) 服务器	29
(2.4.4) 硬件网关	76
(2.4.5) 管理系统	80
(3) 项目系统测试	87
(3.1) 单元测试	87
(3.2) 集成测试	92
(4) 项目总结	95

---

（4.1）成员分工	95
（4.2）成员心得	96
附录 参考文献与引用	97
附录 项目程序清单	98

# 1 项目背景

## 1.1 项目背景与意义

### 1.1.1 项目背景

#### （1）传统生物识别方式漏洞频发，个人隐私泄露风险加剧：

随着移动互联网设备的应用场景越发丰富，人们的生活与互联网变得密不可分。截至2018年12月，中国网民的规模达到了8.29亿，普及率攀升至59.6%。然而，计算机及网络技术也是一把“双刃剑”。万物互联的浪潮之下，网络身份认证问题日益凸显。

生物识别技术的突破，为人们提供了很好的解决方案。受益于近年来以大数据、云计算、深度学习为首的新一代人工智能技术的快速发展，生物识别技术，这一项以人体生理特征或行为特征作为身份标识来进行身份识别的技术在应用方面不断取得突破。以指纹识别及人脸识别技术为代表的传统生物识别方式，已在人们的生产生活中得到广泛应用。

然而，被“3·15”晚会曝光的人脸识别系统安全漏洞、熟人趁受害者不备使用其人脸与指纹完成验证、Deepfake 诈骗兴起等在生物识别领域频发的安全事件，令生物特征数据原本作为优势而存在的“唯一性”这一特征，被发觉含有巨大风险。相较于数据在传输和认证过程中的安全漏洞，生物特征数据一旦被盗，大量带有唯一性的生物特征数据被盗取，给用户带来的风险将会更大。数据唯一，则意味着其不可撤销，也意味着一旦被泄露，被仿造，识别系统将存在崩溃的风险；而某些可能在用户不知情条件下被他人盗用的生物识别信息（如人脸、指纹等），也对个人隐私造成了严重的安全隐患。

#### （2）生物识别样本形式从生理特征转向行为特征：

生物特征识别的最大共性是唯一性——每个人都有独一无二的面部、指纹、虹膜等。由于每个人的生理特征具有与其他人不同的唯一性和在一定时期内不变的稳定性，故利用生物识别技术进行身份认定，相较于其他身份认证技术，具有更高的准确率。而将人体本身固有的、可唯一标记本人的生理和行为特征作为身份认证方式，也很好解决了密钥的丢失和被破解等问题。

然而，生理特征的高度唯一性，使其被不法分子盗取、复现时对用户造成极为严重的安全风险。相比之下，基于行为特征的生物识别技术，因其样本隐私性弱、难以复现，且能体现用户意愿，具有更高的安全性。



人们认为，规避传统生物识别漏洞引发风险的一种可行方法，是基于生物行为特征的认证方法。自 2017 年起，智能语音交互技术发展加速迅猛，声纹识别技术逐渐成为最契合需求的选择。

声纹识别技术，作为一种基于带有生理特征的生物行为特征识别方式，相较于以指纹、人脸识别等技术为代表的完全基于生理特征的传统生物识别方式，具有隐私性弱、不怕丢失、难以伪造三大优势。

#### （1）隐私性弱：

声纹是一种具有生理特征的行为特征，其隐私性相对较弱，采集涉及到的用户隐私信息较少。跟读、语音提示等易于操作的样本采集方式，更易被用户接受。

#### （2）不怕丢失：

不同于指纹、虹膜、人脸等静态的生理特征，声纹作为一种动态的行为特征，不容易丢失，可以做到“失声（音）不失身（份）”。

#### （3）难以伪造：

对于不同的说话人，发音习惯和声纹特征是本人特有的，且在成年后基本不变。而同一人两次朗读相同的内容，也不可能发出完全相同的声音。声纹这种“大同小异”“蕴不变于千变万化之中”的特性，使其伪造难度极高。

#### （3）后疫情时代，“非接触”识别带来新机遇：

2020 年 2 月，国务院在印发复工复产疫情防控措施指南的通知中提出“使用指纹考勤机的单位应暂时停用。”同月，由中国人民银行营业管理部制定的《北京市非银行支付机构复工复产防疫工作指引》提出要优化和丰富“非接触式服务”渠道和场景，强调疫情防控期间，暂缓人脸识别支付商户拓展。

针对疫情期间全民戴口罩时身份识别难度提高，以及公共场所进行身份认证时接触设备带来的交叉感染风险，如何在类似场景下准确识别用户身份、保护个人隐私和保障信息安全，是后疫情时代需要解决的重要问题。而目前广泛采用的人脸识别与指纹识别技术不能很好解决这一问题。摘下口罩进行人脸识别，将增大病毒空气传播、飞沫传播风险；按压指纹识别器进行指纹识别，可带来交叉感染风险。而声纹识别作为一种“非接触”识别方式，可直接阻断病毒传播途径，并间接降低疫情风险，在后疫情时代，具有重要应用价值。

为阻断病毒传播链条，机器人承接了消毒清洁、送药送餐、诊疗辅助等“一线工作”，VR 看房、在线娱乐、在线教育等服务成为大众充实居家生活的必备之选，而打造远程银行、无人工厂的需求也比以往任何时候都更为迫切。这些“非接触式”服务的变革，不仅催生了

新的经济模式——“非接触经济”，如在线办公、在线医疗等，还为声纹识别带来新的产业机遇。

#### (4) 语音交互个性化、声纹识别场景化

目前，主流品牌智能手机系统应用与越来越多的第三方移动应用软件已配备语音交互功能，许多智能硬件厂商也在发力，积极布局以智能手机为中心，以语音交互为导向的智慧家居、智慧出行产品生态。语音识别与物联网技术赋能智能终端，将以往繁杂的操作变为一句句简单的语音指令，极大程度上方便了人们的生活。

声纹识别及其相关技术的发展，使语音交互的个性化、场景化解决方案不断涌现。如利用声纹确认技术完成个人日常生活中的各种事物访问控制的授权——声控锁屏、声控安全门、汽车声控锁等；利用声纹辨认技术，可支持智能音箱、智能语音助手等提供个性化服务，如针对家庭用户中的老年人、儿童等不同年龄段用户，按照兴趣推荐不同的歌曲、新闻，以及开放特定的功能权限等；利用声纹检出和追踪技术，可取代人工完成会议纪要，通过语音识别和声纹识别技术的结合，将会议录音通过语音识别技术识别说话内容、通过声纹识别技术标注每段话所对应的说话人，即可轻松完成多人会议记录，大大提高工作效率。

### 1.1.2 项目研究意义

项目实现改进时延神经网络文本无关声纹识别算法，并以该算法为核心，实现基于竞赛平台，软硬件结合的声纹识别系统，进而完成基于声纹识别的语音控制器的系统设计。综上所述，研究意义归纳如下：

(1) 实现综述中的改进的时延神经网络文本无关声纹识别算法。考虑在有限资源硬件端部署的问题，对模型进行系统的优化，减少参数和计算量，使其更适合部署在更多低性能嵌入式平台。

(2) 通过语音识别、语义识别，实现由说话人语音到控制外设指令的转换。

(3) 打通多场景适用的，基于声纹识别的语音控制器系统的研发流程，将项目软硬件设计实现方案开源，作为声纹识别嵌入式开发的最佳实践案例，有效助力声纹识别及相关技术在企业、工厂、实验室等场景落地应用。

## 1.2 声纹识别发展趋势

### 1.2.1 研究趋势：深度学习效果拔群，时变鲁棒成为焦点

随着深度学习的快速发展和应用，尝试将声纹识别与之结合的思路得到了广泛关注，各项相关研究效果拔群。GeorgHeigold 等人提出了一种端到端的声纹确认方法，其取网络最后一层隐藏层的激活作为说话人表征，使用余弦距离判断两个表征向量是否为同一个说话人。MircoRavanelli 等人提出 SincNet 架构，以 sinc 函数限定网络第一层卷积结构，让网络学习滤波器的截止频率，实现从原始语音信号直接学习，完成声纹识别任务。JohanRohdin 等人则模仿当前主流模型 i-vector-PLDA 模型的工作流，使用深度神经网络 DNN 实现工作流的每个部件，得到了不错的效果。

随着声纹识别技术逐渐成熟、趋于实用，与声纹识别相关的鲁棒性、安全性问题，也受到了研究和开发人员的关注，包括噪声、跨信道、多说话人、身体条件变化、说话方式变化、短语音等鲁棒性问题。

2000 年至 2010 年，清华大学语音和语言中心对由于声纹随说话人年龄变化而发生变化从而导致系统识别性能下降的声纹时变问题进行了研究，提出了时变鲁棒的声纹特征；对使用录音和录音拼接攻击声纹识别系统这一安全问题进行了研究，并提出了切实可行的录音检测方法。

### 1.2.2 应用趋势：国标与商用并进，于实践中积累经验

为规范和正确引导声纹识别发展，国内已公布多项关于声纹识别的标准。2008 年，原信息产业部正式颁布实施了《自动声纹识别（说话人识别）技术规范》，这是我国第一个关于声纹识别的行业标准。2010 年 12 月 2 日，公安部颁布实施了《安防声纹确认应用算法技术要求和测试方法》。2018 年 10 月 9 日，中国人民银行正式对外发布《移动金融基于声纹识别的安全应用技术规范》金融行业标准。这是第一个被金融监管部门认可的生物识别标准，为声纹识别技术进入移动金融领域解决了标准难题。

声纹识别技术因其广阔的应用场景和巨大的潜在价值，从特定领域到民用领域，在国内外正迎来商用化浪潮。声纹识别首先在针对特定人群的国防安全、公安刑侦等领域投入使用，有力保障了国家和公共安全；国内外主流支付平台加持声纹确认技术，通过动态声纹密码的方式进行客户端身份认证，有效提升个人资金与交易支付安全性；互联网公司、智能终端厂

商构建以“声纹识别+语音识别+物联网”为核心技术的智能家居、智慧生活、智能出行体系，令百姓生活更加便利。

### 1.3 项目应用前景

根据《2019 中国声纹识别产业发展白皮书》所述，声纹驱动控制器的未来发展方向，将不仅仅局限于安全保障。声纹特征的丰富性、稳固性与动态性，将使得其在**公共安全、移动金融、社会保险及个性化语音助手等技术领域的应用场景愈发多样**。

由于相关设备造价低廉，性能较好，声纹识别在**日常生活中应用前景广泛**。相关设备可用于智能家居，便捷百姓生活；可用于小区门禁系统，对进出小区住户进行记录；可用于银行自助取款机，快速识别取款人身份，为用户带来安全、便捷的使用体验。

**国防安全及公共安全等领域**是目前声纹辨认技术应用较为广泛的场景，如公检法人员能够通过通话记录中的声纹信息初步锁定嫌疑人，降低刑侦难度。公共安检处也可以通过在安检口设置声纹检测装置，初步筛查危险人物，提高公共保障安全性。

**移动金融方面**，以中国建设银行为首的一众银行逐步将声纹信息与动态密码相结合，通过多重身份认证的方式来提高安全性能。目前的主要实现方式是让用户读一段动态密码，将结果与用户事先认证好的声纹信息进行匹配，验证通过即可进行后续操作。

如今，就**准确率而言**，声纹识别技术的识别准确率在理论上已超过 **90%**，但在现实生活中，说话人自身的独有特性、身体状况、年龄增长、情感波动等其他干扰因素，导致实际与实验中的理想值还存在一定偏差，仍可能出现对说话人身份产生误判的情况，因此可以提高准确率的方法还须进一步被发现、探讨、研究。但可以预见的是，在生物识别技术日趋成熟，生物识别设备加速普及的背景下，**声纹识别作为多生物特征识别融合技术体系中的关键一环**，将在与其所应用领域业务形成发展的良性闭环同时，为信息时代的数字安全提供更为坚实的保障。

## 2 项目设计方案

### 2.1 需求分析

#### （1）快速声纹辨认能力：

系统须识别说话人的靠近，给出说话人语音提示，并在说话人位于恰当距离处开始讲话时采集数秒的音频样本，通过将该样本与本地声纹模型数据库进行比对，确认某个声纹模型已存在于本地声纹模型数据库中用户的身份（已注册用户），或判断该用户的声纹模型不存在（未注册用户），并给出秒级响应。

#### （2）快速声纹注册能力：

需要时，系统可为用户提供注册功能。在提示下，令未注册用户与系统进行含有讲话、录音环节的交互，系统可处理用户讲话录音，生成并保存该用户声纹模型。此后，该用户可使用声纹识别完成身份验证。

#### （3）根据声纹辨认与语音识别结果进行语义识别，执行简单指令：

当系统辨认出用户为已注册用户时，须对其录音进行语音识别，提取文本信息。通过语义识别，将提取的文本信息映射至预设的简单指令集，系统据此决定并执行后续操作。

#### （4）声纹管理能力：

系统可为管理员提供交互界面，方便管理员进行系统配置、用户授权、声纹识别记录监控、声纹模型数据库管理等操作。

用户使用项目系统，主要完成**声纹注册识别、语音识别功能**；管理员使用项目系统，主要完成**声纹识别设备、声纹识别记录等管理功能**。鉴于用户与管理员的应用场景耦合度较低，且在实际应用时管理员可能需要管理若干声纹识别设备，故在**模块设计上**也应将**用户与管理员相关模块分离**。由此，采用基于B-S、C-S架构与终端的系统设计，分别实现系统的管理与注册/辨认功能。

## 2.2 系统架构与技术选型

项目系统采用基于 B-S、C-S 架构与终端的系统设计，包含 4 个功能模块：用户终端应用、服务器、硬件网关、管理系统。

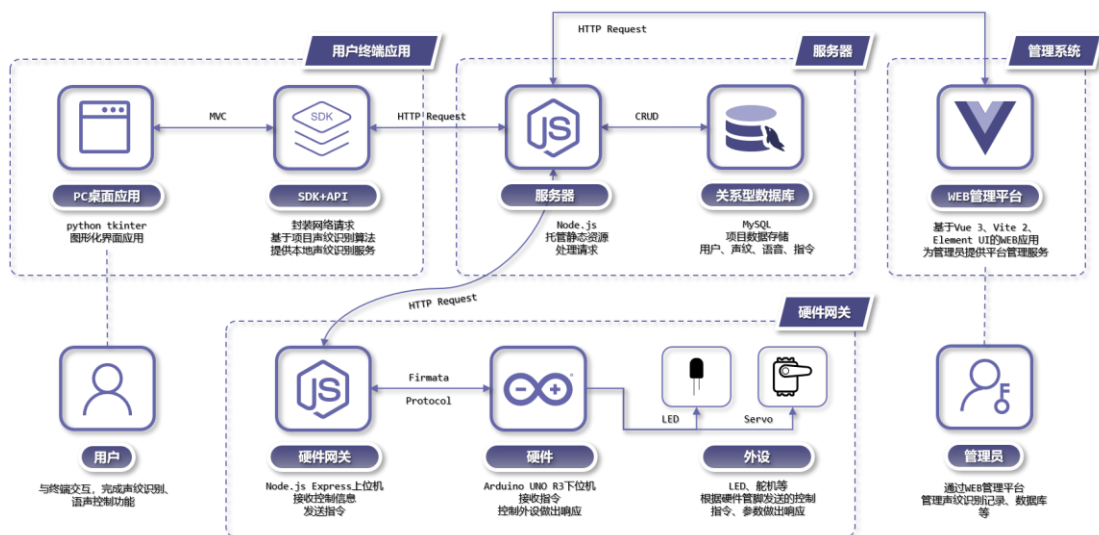


图 1 项目系统架构示意图

### (1) 用户终端应用：

基于 python tkinter 实现，面向用户，提供带有录音功能的，具有图形化界面的 PC 桌面应用。该应用通过集成包含项目声纹识别算法的软件开发工具包（Soniccredible SDK）并封装网络请求，为用户提供注册、声纹注册、声纹辨认与语音控制功能。

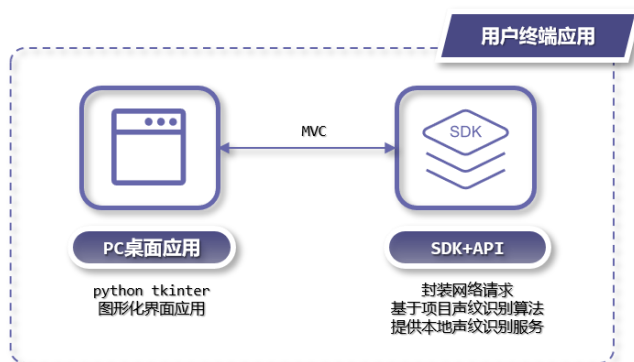


图 2 用户终端应用系统架构示意图

### (2) 服务器：

基于 Node.js Express 实现，使用 MySQL 关系型数据库，集成鉴权、注册、静态资源上传、托管等多种功能。作为项目系统功能模块的核心，为声纹识别用户终端应用、硬件网关提供 HTTP RESTful API，进而实现数据传输与指令控制功能。



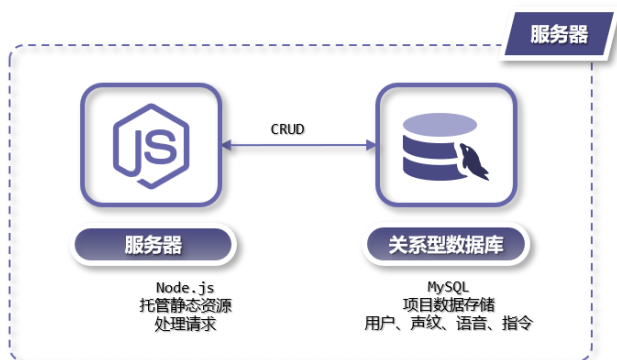


图 3 服务器系统架构示意图

### (3) 硬件网关:

基于 Node.js Express 构建上位机，以接有外设的 Arduino UNO R3 开发板为下位机，上位机作为网关，接收来自服务器的控制请求，通过 Firmata 协议，将控制指令发送至下位机，实现外设的控制。

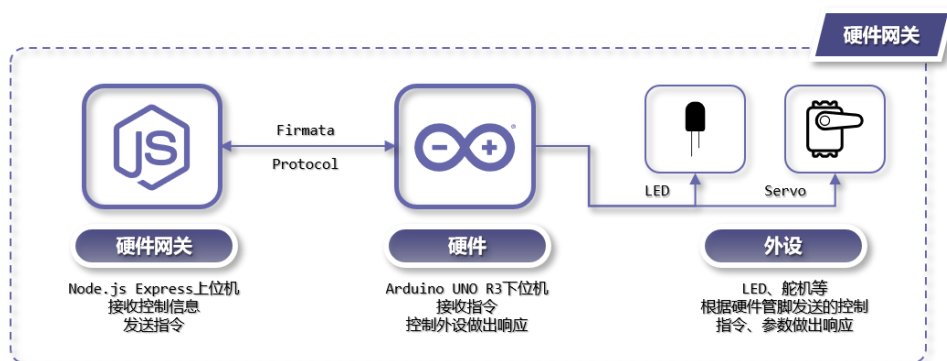


图 4 硬件网关系统架构示意图

### (4) 管理系统:

基于 Vue 实现，面向管理员，提供 WEB 应用。该应用通过集成用户管理、声纹识别音频管理、语义识别音频管理、音频指令管理等模块，为管理员监视系统状态、操作数据提供便利。



图 5 管理系统系统架构示意图



## 2.3 概要设计

### 2.3.1 改进的时延神经网络文本无关声纹识别算法

#### (1) 算法功能：

该算法尝试实现判定待测语音属于系统中已注册的说话人模型中的身份，即从数据库找出是否有声纹与之匹配，是  $1:N$  的身份辨认问题。完成任务的方法即设定一个声纹阈值，当待测语音与系统中所有人的相似度都低于该阈值，则认为待测语音对应的身份未在系统中注册过。该问题属于一项开集识别问题。

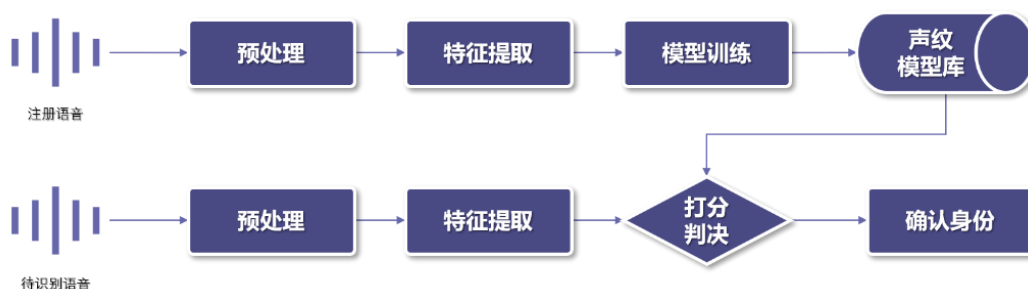


图 6 改进的时延神经网络文本无关声纹识别算法工作流程

算法首先需要实现声纹信息的数据库录入，即将注册者的语音信息进行预处理、特征提取；使用提取出的特征进行模型训练得出注册者的声纹并存入数据库。模型工作时，将收到的待识别语音预处理并提取出特征后，将特征与声纹模型库中的已有特征进行比对，使用打分算法判决确定说话人身份。

#### (2) 算法特征：

根据系统功能框图，算法的部署分为 3 个模块：数据预处理，数据特征提取，模型训练及预测打分。

##### (1) 数据预处理：

首先对训练集的数据进行了数据扩增。扩增方式参考如今多种较为流行的语音数据扩增方式，诸如加噪扩增（MUSAN, RIR）、变说话方式、On-the-fly 扩增、Voice Conversion（VC）和 TTS 扩增技术，最终采取加噪音混响、加背景音乐、倍速、频谱遮盖等方式作为数据增强的方式。对于每条语音信息，都从中随机选择若干类型进行扩增后投入模型训练。

对于待识别语音采集后的数据，为了最大程度避免失真，提高预测准确率，我们在预处理阶段分别采用**预加重**（Pre-emphasis）、**分帧**（Framing）、**加窗**等过程对其进行了降噪。

**预加重**过程采用一阶 FIR 高通数字滤波器补偿语音信号高频分量的损失，使信号的频

谱变得平坦的同时，能够用同样的信噪比低频到高频求信号频谱，以便于更好地进行频谱分析或者声道参数分析。假设输入信号第  $n$  时刻语音采样值表示为  $x[n]$ ，经过预加重处理后的输出为  $x'[n] = x[n] - ax[n-1]$ 。

**分帧**过程主要涉及两个参数：帧长（帧本身的长度）和帧移（帧与帧之间的间隔）。其基本思想是基于对语音信号的“短时平稳假设”以及希望更大限度的降低由于傅里叶变换带来的音频时域信息丢失，因此加入分帧处理过程，对于声纹识别系统将带来极大的提升。

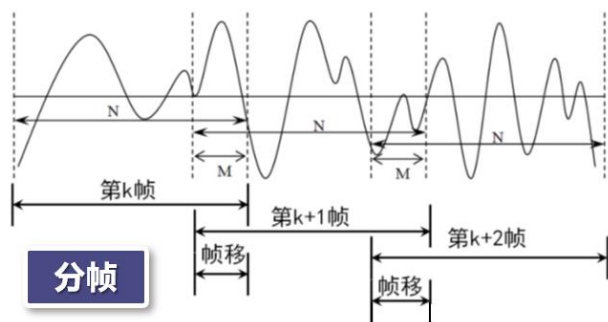


图 7 数据预处理——音频分帧

**加窗**过程的引入是为了解决分帧之后可能引起的吉布斯现象（Gibbs Phenomenon）与频谱泄露（Spectral Leakage）问题。即由于分帧产生信号的不连续性与截断效应，进而出现傅里叶变换后的频谱局部峰值，并且频域通带边界处出现多个旁瓣的现象。通过加窗处理，分帧后的每帧音频信号的每个值会被赋予不同权重。本项目中采用汉明窗（Hamming Window），此类型的窗函数中心到边缘权重值依次减小，可以有效减小分帧造成的信号不连续性。

## （2）数据特征提取：

语音信号的特征提取方面，训练集与待识别数据的处理方式基本一致，均采用了基于梅尔频率尺度的语音信号特征提取技术，主要包括傅里叶变换、梅尔滤波器组及离散余弦变换。其目的在于对语音数据进行正弦提升（Sinusoidal Liftering），改善在噪声环境中的声纹识别效果。

## （3）模型训练与预测打分：

项目所用模型，是参考 ECAPA-TDNN 模型后提出的一种基于改进时延神经网络的文本无关说话人辨认网络模型。预处理并通过声学特征提取得到的 MFCC 特征将输入构建好的神经网络模型，随后经过 TDNN 层和 SE-ResNet2Block 层扩展时间上下文提取帧级特征，加入 SE-Block 以明确对通道相关性进行建模，系统连接所有 SE-ResNet2Block 的输出进行多层特征聚合。此后，注意力统计池化层关注重要的帧级特征，计算加权平均值和标准差。最后，前馈神经网络提取音频段级特征，得到嵌入向量。

在对未注册声纹信息说话人的语音进行识别验证时，对特征向量进行采用余弦相似度（Cosine Similarity, CS）计算，选择相等错误率（EER）和最小检测代价函数（minDCF）作为评价指标。

### （3）算法改进：

对损失函数与模型部署进行了优化。

#### （1）损失函数优化：

对于 Softmax 损失函数及其改进方案进行综合分析，发现它们都是通过在角度函数中引入各种类型的间距以增加类间距离。训练过程的收敛速度和模型性能在很大程度上取决于损失函数变体的超参数选择，这通常需要通过使用不同的超参数值多次重复训练来进行调整得到一个最优值。

为解决上述问题，提出对 AAM-Softmax 进行改进，使用**超参数随迭代次数动态调整策略**，该方法可以在迭代时动态调整超参数比例因子和间距，从而显著提高网络训练的收敛速度和说话人识别的精度。

分析损失函数，得出需要调制的超参数主要有两个，分别为  $m$  和  $s$ ：参数  $s$  对于狭窄的一些距离的范围进行了缩放，使得全连接层的输出结果更加可分；而参数  $m$  增大了不同类别之间的间距，提升了分类的能力。这些超参数最终会影响到预测分类概率，理想的超参数的设置应该使得全连接层的输出覆盖  $[0,1]$  的范围，并且在角度参数附近的梯度要大，使得训练更加有效。

对于尺度参数  $s$  的动态调整方面，引入表示第  $t$  次迭代时样本  $\mathbf{x}_i$  和其所属说话人类别  $y_i$  的权重向量  $\mathbf{w}_j$  间的角度的  $\theta'_{i,y_i}$  参数，并使用迭代次数为  $t$  时中全部  $N$  个样本与目标样本类别权重夹角的中位数  $\theta_{med}^{(t)}$  表示网络在当前批次上的优化程度。通过监控  $\theta_{med}^{(t)}$  的相对大小，明确网络参数的优化状态，调整监督严格程度，经过多次迭代得到动态尺度参数。

类似地，对于  $m$  参数的调整，定义全部  $N$  个样本与所属说话人类别  $y_i$  权重夹角  $\mathbf{w}_j$  的中位数  $\theta_{med}^{(t)}$ ，并使用此值对于优化程度进行表征，根据网络收敛情况调整间距  $m$ 。

#### （2）模型部署优化：

考虑到项目需要嵌入式部署与运行声纹模型，需要减少模型参数量与浮点运算次数，项目通过将**深度可分离一维卷积**（One-dimensional Depth-wise Separable Convolution, 1DDSC）方案代替作为参考的 ECAPA-TDNN 模型使用的一维扩张卷积（Res2 DilatedConv1D）方案，

实现系统性能的大幅改善。

同时，鉴于嵌入式设备通常采用 float16 或 int8 等较低精度的模型，ECAPA-TDNN 模型中作为激活函数使用的修正线性单元 ReLU 将存在精度损失。因此选择采用 ReLU 的改进版本——ReLU6，作为网络模型的激活函数，这样既保证了模型在嵌入式系统中正常的数值分辨率，也防止了“梯度爆炸”等情况的出现。

为了最大限度提升模型部署的资源瓶颈，项目引入模型量化技术。通过线性量化，将 float 32 权重矩阵用 8 位无符号整数（uint8）表示，大幅节省内存占用。项目团队还将结合大赛提供平台的嵌入式特性，对算法与模型进行进一步优化。

### 2.3.2 基于 2.3.1 中算法的声纹识别语音控制器系统

面向需求分析中的声纹注册、辨认、处理与管理等能力，将题述系统划分为 3 个子系统，分别为认证系统、命令系统、管理系统：

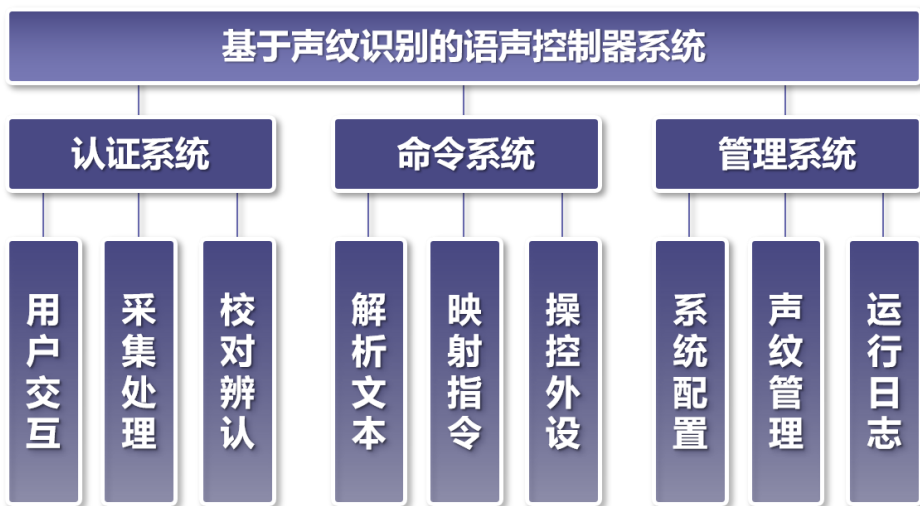


图 8 基于声纹识别语音控制器系统模块概要设计

#### （1）认证系统：

具有采集、处理、校对、辨认用户声纹信息及与用户交互的功能。

终端方面，接入外设，指引用户完成注册或验证操作，实现对用户语音信息的采集；接入网络，获取在线声纹模型库；通过改进的时延神经网络文本无关声纹识别算法，对样本进行相似度打分，给出用户身份预测结果，最终根据结果响应用户，并将相关数据报送后端。

后端方面，存储前端发送的声纹音频；在终端请求获取数据进行比对时，根据查询结果作出响应；对终端声纹识别结果（成功、失败、异常等）进行日志记录。

#### （2）命令系统：

具有解析用户语音对应的文本内容，将其映射至预设的简单指令集，并向测试用受控外设发出指令的功能。

后端方面，在收到终端的语音识别请求后，调用基于主流开放平台的“短语音识别”网络接口，获取预先录制的说话人音频文件所对应的文本内容，通过语义分析提取关键信息，根据预设简单指令集，映射为带有参数的简单指令，以响应的形式报送终端。

终端方面，在成功验证说话人身份并向后端发送语音识别请求后，根据后端响应中的指令操作测试用受控外设，并向用户作出指令执行结果的提示。

### （3）管理系统：

具有管理系统配置、用户信息、声纹模型数据库、运行日志等功能。

管理系统部署于项目后端，在 WEB 应用中实现上述功能。管理员可登录此 WEB 应用进行上述操作。该系统同样可用于项目系统的监控，例如基于 WebSocket 或 HTTP 轮询方式检查在线声纹识别设备数量及状态，或根据声纹识别请求数量判断是否存在恶意使用现象，及时生成工单或进行告警，通知管理员采取相应措施。

在完成项目软硬件最终技术选型与细节设计后，项目团队展开详细设计，遵照“自顶而下，逐步求精”的程序设计思想，秉持鲁棒性、复用性、高内聚、低耦合的设计原则，对上述系统与子系统进行编码实现。

## 2.4 详细设计

### 2.4.1 综述

项目系统的核心功能，是令用户录下一段音频，并将该音频与预先存储的用户声纹音频进行比对，完成用户身份核验，并对上述音频进行语义识别与指令转换，最终使用转换得到的指令控制外设。由此，结合 2.3.2 中的系统设计，得到项目核心功能时序图。

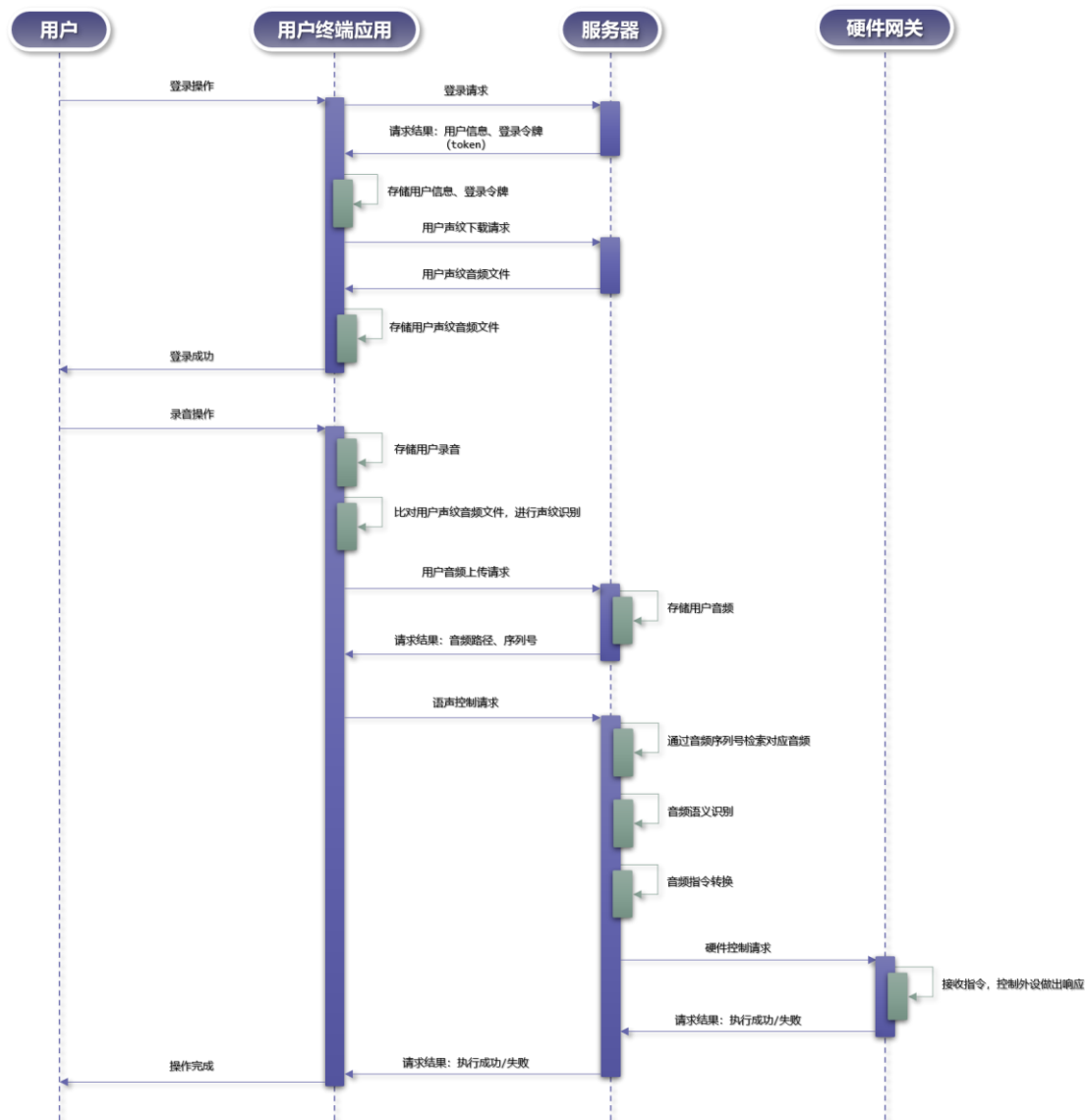


图 9 项目核心功能时序图

该时序图体现了项目核心功能业务逻辑的关键步骤：**登录、声纹加载、录音、声纹识别、语音控制、硬件控制。**

#### (1) 登录：

用户通过用户终端应用，执行登录操作。用户终端应用向服务器发起登录请求，执行成功时，服务器返回**用户信息与登录令牌**。

#### **(2) 声纹加载：**

用户终端应用将用户信息、登录令牌保存至本地，携带登录令牌，向服务器发起**用户声纹下载请求**，执行成功时，服务器返回用户声纹下载地址，用户终端**下载用户声纹音频**，并将其保存至本地。

#### **(3) 录音：**

用户通过用户终端应用，执行**录音**操作。用户终端给予用户一定提示，并**现场录下用户的一端音频**，将其保存。

#### **(4) 声纹识别：**

用户终端应用将（3）中所述**音频**与（2）中所述**用户声纹**进行比对，**确认（3）中所述音频是否为用户本人**，若是，则进行后续操作，若否，不进行后续操作。“后续操作”指本节“（4）声纹识别”后任一操作。

#### **(5) 语声控制：**

用户终端向服务器发起语声控制请求，服务器根据该请求的请求体数据**检索对应音频**，对该音频进行**语义识别**，提取其中的文本内容与参数内容，此后进行**指令转换**，**将音频映射为硬件控制指令**。

#### **(6) 硬件控制：**

服务器携带（5）中所述控制指令，向硬件网关发起硬件控制请求，执行成功时，硬件网关根据该请求的请求体数据获取**控制信息**，**确认外设与参数**，此后**控制外设**，并将控制结果响应至服务器，服务器将此结果响应至用户终端应用。

下文将以上述核心业务逻辑为主线，围绕用户终端应用、服务器、硬件网关、后台管理系统等系统功能模块，对项目的详细设计进行系统而精到的阐述。

您可访问[项目 GitHub 仓库](#)，将其克隆（Clone）至本地，以便对照此文档查看代码。



## 2.4.2 用户终端应用

项目用户终端应用基于 python tkinter 实现，面向用户，提供带有录音功能的，具有图形化界面（GUI）的 PC 桌面应用。该应用通过集成包含项目声纹识别算法的软件开发工具包（Soniccredible SDK）并基于 Request 库封装网络请求，在实现声纹对比，音频信号处理以及



图 10 用户终端应用业务模块设计

用户 Token 的本地存储等功能同时，为用户提供注册、声纹注册、声纹辨认与语音控制功能。

### 2.4.2.1 GUI

#### （1）首页：

用户打开程序，首先进入首页。

首页的功能设计分为两部分：左上角菜单栏、“用户注册”和“用户登录”两项按钮。菜单栏引导使用者查看开发者的联系方式以及关于项目的简要信息。“用户注册”和“用户登录”两项按钮引导使用者进入登录或注册界面。



图 11 GUI 首页

#### （2）用户注册：

通过注册页（register）进行注册。注册页（register）提供了注册表单。在表单中，用户需要输入以下字段：昵称、用户名、密码、电话号码。点击用户注册进行注册，并按照提示点击弹出的录入按钮进入声纹录入页面。

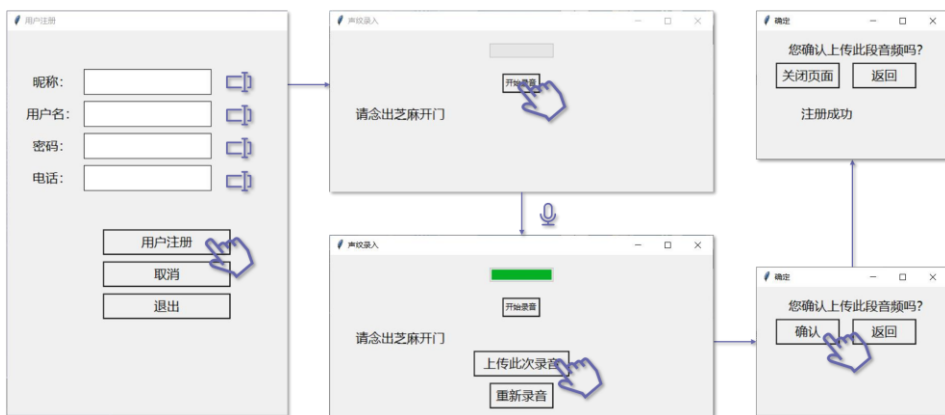


图 12 用户注册相关页面

在声纹录入页面中，点击开始录音即可开始录入声音，按照提示，在进度条完成之前，



大声而清晰地说出口令（图为“芝麻开门”）。

完成录制后可点击“上传此次录音”按钮进行录音上传，点击后弹出“确定页面”，提醒用户对于是否要上传音频进行再次确认，防止误触。

### （3）用户登录：



图 13 用户登录页面

用户完成注册后，即可通过登录页（login）进行登录。

登录页（login）提供了登录表单，用户可采用输入用户名与密码的方式登录。用户输入正确的用户名和密码后点击登录按钮，即可进行登录，并弹出提示消息，引导用户进行声纹识别。

#### 2.4.2.2 SDK

声纹识别的关键步骤，如 [1.1.1 项目背景](#) 所述，是对声音信号特征的准确提取与对比。项目采用深度学习训练特征提取与比对算法，所用算法模型框架为 **Resnet34 深度残差网络**。这是因为声纹特征的提取相较于图片、视频等特征明显的模式识别领域而言，需要更深的网络层次。若使用普通卷积网络，则存在随着网络层数的增加，网络发生退化（Degradation）的现象，训练集 loss 将从饱和开始反向增大。而残差网络通过引入残差块，网络可以达到很深，网络的效果也随之变好。

依托学校提供的 GPU 训练平台，经过 70 轮次共计 140 余小时的训练后，模型评估达到理想误差阈值，保存模型为 resnet34.pth。借助于此模型，封装 infer\_contrast.py 文件作为声纹识别 SDK，为主程序通过声纹识别相关接口，该接口参数设计如下：

infer_contrast.py	
参数名称	内容
audio_path1	用户的本地声纹
audio_path2	用户的服务器注册声纹
threshold	判断是否为同一声纹的阈值
input_shape	输入数据规模
model_path	所用特征提取模型的存储路径

文件内包含两个可执行函数，分别为执行音频特征值预测的 infer 函数与执行两音频

声纹信息对比的 `contrast` 函数。

`infer` 函数根据 `model_path` 指向的算法模型，对传入的音频文件进行特征提取，并返回特征数据：

```
1. # 加载模型
2. model = torch.jit.load(args.model_path)
3. model.to(device)
4. model.eval()
5.
6.
7. # 预测音频
8. def infer(audio_path):
9.     input_shape = eval(args.input_shape)
10.    data = load_audio(audio_path, mode='infer', spec_len=input_shape[2])
11.    data = data[np.newaxis, :]
12.    data = torch.tensor(data, dtype=torch.float32, device=device)
13.    # 执行预测
14.    feature = model(data)
15.    return feature.data.cpu().numpy()
```

`contrast` 函数首先调用 `infer` 函数，对传入的两个音频信号分别提取声纹特征数据，并求取两个特征数据的对角余弦值，其结果即为二者的相似度，若相似度大于所设置的阈值，则认为两段声音来自同一用户：

```
1. def contrast(audio_path1, audio_path2):
2.     args.audio_path1=audio_path1
3.     args.audio_path2=audio_path2
4.     print_arguments(args)
5.     # 要预测的两个人的音频文件
6.     feature1 = infer(args.audio_path1)[0]
7.     feature2 = infer(args.audio_path2)[0]
8.     # 对角余弦值
9.     dist = np.dot(feature1, feature2) / (np.linalg.norm(feature1) * np.linalg.norm(feature2))
10.    if dist > args.threshold:
11.        print("%s 和 %s 为同一个人，相似度
为: %f" % (args.audio_path1, args.audio_path2, dist))
12.    else:
13.        print("%s 和 %s 不是同一个人，相似度
为: %f" % (args.audio_path1, args.audio_path2, dist))
```

此外，SDK 还包含有通过 Pyaudio 实现的本地音频录制的模块，即 `record.py` 程序。通过设置音频采样率（`rate`）、单双声道（`channels`）以及录制时长（`record_seconds`）等参数，可定义音频录制类 `RecordAudio`，通过调用其对应的 `record` 方法来实现音频录制，并

返回录制完成的音频地址:

```
1. class RecordAudio:
2.     def __init__(self):
3.         # 录音参数
4.         self.chunk = 1024
5.         self.format = pyaudio.paInt16
6.         self.channels = 1
7.         self.rate = 16000
8.
9.         # 打开录音
10.        self.p = pyaudio.PyAudio()
11.        self.stream = self.p.open(format=self.format,
12.                                   channels=self.channels,
13.                                   rate=self.rate,
14.                                   input=True,
15.                                   frames_per_buffer=self.chunk)
16.
17.    def record(self, output_path="audio/Local_voice.wav", record_seconds=5):
18.        """
19.        录音
20.        :param output_path: 录音保存的路径, 后缀名为 wav
21.        :param record_seconds: 录音时间, 默认 3 秒
22.        :return: 录音的文件路径
23.        """
24.        i = input("按下回车键开机录音, 录音 5 秒: ")
25.        print("开始录音.....")
26.        frames = []
27.        for i in range(0, int(self.rate / self.chunk * record_seconds)):
28.            data = self.stream.read(self.chunk)
29.            frames.append(data)
30.
31.        print("录音已结束!")
32.        wf = wave.open(output_path, 'wb')
33.        wf.setnchannels(self.channels)
34.        wf.setsampwidth(self.p.get_sample_size(self.format))
35.        wf.setframerate(self.rate)
36.        wf.writeframes(b''.join(frames))
37.        wf.close()
38.        return output_path
```

### 2.4.2.3 API

API 部分根据[服务器接口文档](#)设计，参考 Request 网络请求包的设计，将每一接口的前端请求进行封装。封装后的接口函数整合于 post.py 文件中，总览如下：

post.py		
函数	参数	备注
register	nickname, username, password, tel	用户注册接口
login	username, password	用户登录接口
get_users	page_index	获取用户信息（仅限管理员）
update	id, nickname, file, password, realname, email, tel	用户信息更新
audio_recognition	file	待语义识别音频上传
avatar	file	用户头像上传
get_audios	page_index	获取音频信息（仅限管理员）
recognition	audio_id	音频语音识别
voiceprint	file	用户声纹注册
get_voiceprint		用户个人声纹下载
command	serial	音频驱动硬件响应

其中各参数对应含义如下：

post.py	
参数名称	内容
nickname	昵称
username	用户名
password	登录密码
tel	电话号码
page_index	页码编号
file	文件路径，可以是音频或者图片
audio_id	音频编号(audio_recognition 请求返回值中包含)
serial	音频序列号(audio_recognition 请求返回值中包含)

其中 login 前端请求相对特殊，其发送请求的同时，还承担有本地 token 的数据更新任务。token 作为服务器对用户的重要识别码，需要在登录后的其他请求中被包含在 Header 内进行传输。因此前端采用 Shelve 存储方案，将每次调用 login 请求后更新的 token 值存储在本地 Storage 内，以便其他请求发送时读取：

```

1. def login (username,password):
2.     global token
3.     url='/api/v1/user/login'
4.     param_dict={
5.         "username": username,
6.

```

```
7.     "password": password
8.
9.     }
10.    response_code = None
11.    response = None
12.    try:
13.
14.        ret = requests.post(Server+url, data=param_dict)
15.
16.        response_code = ret.status_code
17.        response = ret.text
18.        try:
19.            with closing(shelve.open('storage/storage', 'c')) as shelf:
20.                shelf['token'] = eval(response)['info']['token']
21.        except:
22.            print('请求失败')
23.    except requests.HTTPError as e:
24.        response_code = e.getcode()
25.        response = e.read().decode("utf-8")
26.
27.    return response_code, eval(response)
```

在声纹认证及语义驱动部分，为保证服务器数据存储格式统一，API 接口辅助有音频转码模块 trans.py。通过 Pydub 库实现了 wav 格式与 mp3 格式的互转，将上传至服务器的音频格式统一至 mp3 格式，将本地声纹对比的两段音频统一至 wav 格式：

```
1. from pydub import AudioSegment
2.
3.
4. def mp32wav(filepath):
5.     sourcefile = AudioSegment.from_mp3(filepath)
6.     filename = 'audio/'+filepath.split('/')[1].split('.mp3')[0].replace(' ', '_') + '.wav'
7.     # print(filename)
8.     sourcefile.export(filename, format="wav")
9.     return filename
10.
11. def wav2mp3(filepath):
12.     sourcefile = AudioSegment.from_wav(filepath)
13.     filename = 'audio/'+filepath.split('/')[1].split('.wav')[0].replace(' ', '_') + '.mp3'
14.     # print(filename)
15.     sourcefile.export(filename, format="mp3")
16.     return filename
```

通过调用该模块下的方法函数，可以在原始音频同一目录下生成转码后的同名音频文件，

并返回文件路径。此方法被复用于 API 接口部分的 voiceprint 及 get\_voiceprint 请求中，以 voiceprint 为例，截取片段如下：

```
1. def voiceprint (file):
2.     param_header={'token':token}
3.     url='/api/v1/upload/voiceprint'
4.     file = wav2mp3(file)
5.     files = {'audio':open(file,'rb')}
6.     response_code = None
7.     response = None
8.     try:
9.         ret = requests.post(Server+url,files = files, headers=param_header)
10.         response_code = ret.status_code
11.         response = eval(ret.text)
12.     except requests.HTTPError as e:
13.         response_code = e.getcode()
14.         response = e.read().decode("utf-8")
15.
16.     return response_code,response
```

## 2.4.3 服务器

项目服务器基于 Node.js Express 实现，使用 MySQL 关系型数据库，遵照 MVC 模式设计，集成鉴权、注册、静态资源上传、托管等多种功能。作为项目系统功能模块的核心，项目服务器为声纹识别用户终端应用、硬件网关提供 HTTP RESTful API，进而实现数据传输与指令控制功能。

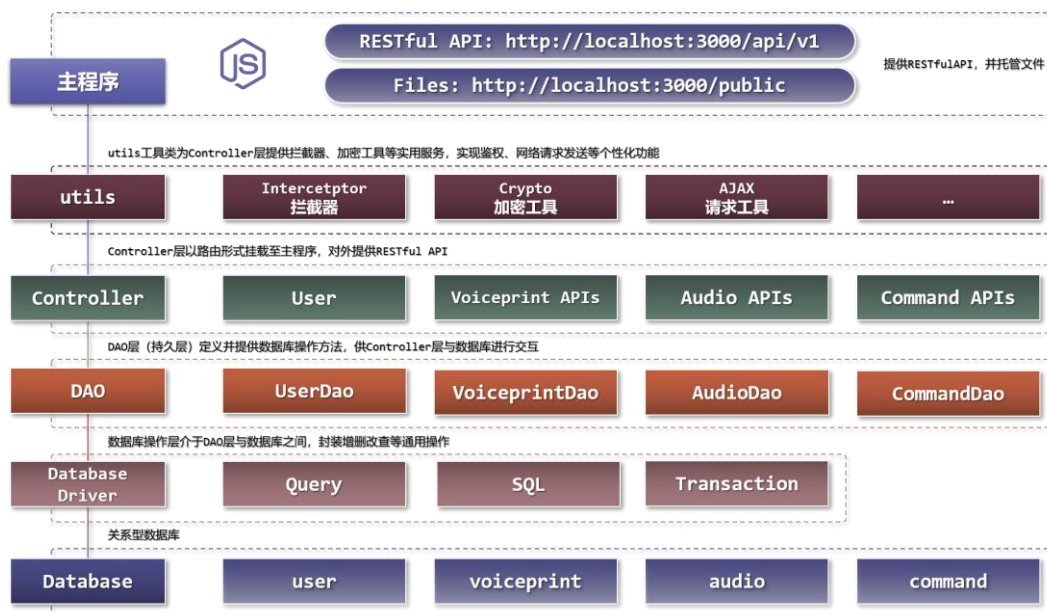


图 14 遵照 MVC 模式的服务器业务模块层级设计

### 2.4.3.1 数据表

通过项目需求分析与概要设计，提取数据实体：**用户（user）**、**声纹（voiceprint）**、**音频（audio\_recognition）**、**指令（command）**。

作出包含部分字段的项目数据实体 E-R 图。

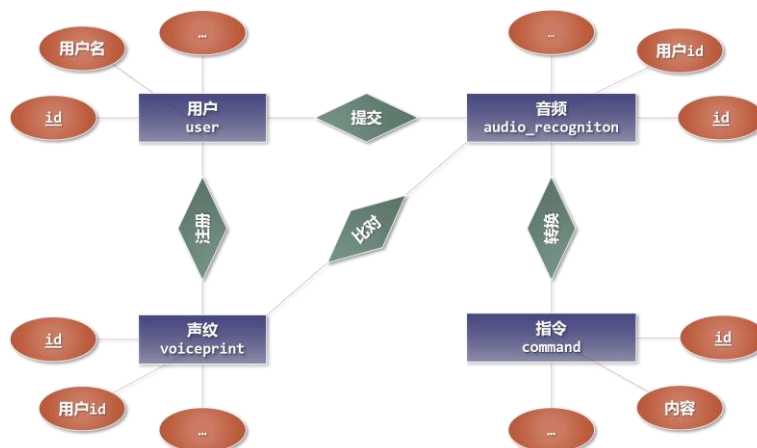


图 15 包含部分字段的项目数据实体 E-R 图

故在 MySQL 关系型数据库中，设计数据表 4 个：用户（user）、声纹（voiceprint）、指令（audio\_command）、音频（audio\_recognition）。



图 16 数据库层设计

### (1) 用户（user）:

user			
字段	类型	备注	索引及属性
id	int(11)	编号	PRIMARY, AUTO_INCREMENT
realname	varchar(20)	真实姓名	
nickname	varchar(20)	昵称	
username	varchar(20)	用户名	
password	varchar(200)	密码（AES 加密）	
avatar	varchar(200)	头像 URL	
tel	char(11)	电话号码	UNIQUE
email	varchar(200)	电子邮箱	UNIQUE
authority	varchar(10)	权限 (root、manager、user)	
is_available	tinyint(1)	是否有效	
time_created	timestamp	创建时间	
time_modified	timestamp	最近修改时间	ON UPDATE CURRENT_TIMESTAMP

### (2) 声纹（voiceprint）:

voiceprint			
字段	类型	备注	索引及属性
id	int(11)	编号	PRIMARY, AUTO_INCREMENT
user_id	int(11)	用户 id	
url	varchar(200)	路径	
time_created	timestamp	创建时间	
time_modified	timestamp	最近修改时间	ON UPDATE CURRENT_TIMESTAMP

### (3) 指令（audio\_command）:

audio_command			
字段	类型	备注	索引及属性
id	int(11)	编号	PRIMARY, AUTO_INCREMENT
keyword	varchar(200)	对应关键词	UNIQUE
content	text	指令内容	
description	varchar(200)	描述	
is_valid	tinyint(1)	是否可用	
time_created	timestamp	创建时间	
time_modified	timestamp	最近修改时间	ON UPDATE CURRENT_TIMESTAMP



#### (4) 音频 (audio\_recogniton):

audio_recogniton			
Field	Type	Comment	Extra
id	int(11)	编号	PRIMARY, AUTO_INCREMENT
serial	varchar(20)	序列号	
user_id	int(11)	提交该语音的用户编号	
url	varchar(200)	路径	
base64	longtext	音频 (base64) 编码	
result	longtext	语义识别结果	
time_created	timestamp	创建时间	
time_modified	timestamp	最近修改时间	ON UPDATE CURRENT_TIMESTAMP

#### 2.4.3.2 数据库操作类

在 MVC 设计模式中，数据库操作类介于 DAO 层与数据库之间，通过封装增删改查等通用操作，简化 DAO 层设计，并增强复用性。项目数据库操作类包含数据库驱动类 Query、SQL 语句类 SQL 与数据库事务类 Transaction。



图 17 数据库操作类设计

数据库驱动类 Query 适用于执行单条 SQL 语句的情况，通过静态方法 query 执行查询：

```

1. class Query {
2.
3.   static query(sql, params) {
4.     return new Promise((resolve, reject) => {
5.       db.getConnection(function (err, connection) {
6.         if (err) {
7.           reject(err);
8.         } else {
9.           connection.query(sql, params, (err, result, fields) => {
10.
11.             if (err) {
12.               reject(err);
13.             } else {
14.               resolve(result);
15.             }
16.           connection.release();
17.         }
18.       });
19.     });
20.   }
21. }
  
```

```

16.         })
17.     }
18. })
19. })
20. }
21. }

```

SQL 语句类 SQL 设计如下:

```

1.  class SQL {
2.
3.      static generateSQL(sql, params) {
4.          return {
5.              sql,
6.              params
7.          }
8.      }
9.
10.     static selectById({table_name, id}) {
11.         return `SELECT * FROM ${table_name} WHERE id = ${id}`;
12.     }
13.
14.     static select({table_name, condition = ""}) {
15.         return `SELECT * FROM ${table_name} ${condition}`;
16.     }
17.
18.     static count({table_name, condition = ""}) {
19.         return `SELECT COUNT(*) as count FROM ${table_name} ${condition}`;
20.     }
21.
22.     static insert({table_name, params}) {
23.         let objectToInsert = this.#objectToInsert(params);
24.         return `INSERT INTO ${table_name} (${objectToInsert.fieldString}) VA
LUES (${objectToInsert.placeholderString})`;
25.     }
26.
27.     static update({table_name, params, condition = ""}) {
28.         return `UPDATE ${table_name} SET ${this.#objectToUpdate(params)} ${c
ondition}`;
29.     }
30.
31.     static fuzzySearch({keyword, table_name, fields = "*", column, condition
= ""}) {
32.         keyword = keyword.replace(/\s+/g, ' ');

```

```

33.         keyword = keyword.replace(/(^\\s*)|(\\s*$)/g, '');
34.         let key_array = keyword.split(' ');
35.         let sql = `SELECT ${fields} FROM ${table_name} WHERE ${column} LIKE
        `;
36.         key_array.forEach((item, index, arr) => {
37.             sql += "'%'" + item + "%'";
38.             if (index !== arr.length - 1) {
39.                 sql += ` OR ${column} LIKE `;
40.             }
41.         });
42.         return `${sql} ${condition === "" ? "" : condition}`;
43.     }
44.
45.
46.     static #objectToInsert(o) {
47.         let fields = Object.getOwnPropertyNames(o);
48.         let fieldString = fields.join(", ");
49.         let placeholder = new Array(fields.length).fill("?");
50.         let placeholderString = placeholder.join(", ");
51.         return {
52.             fieldString: fieldString,
53.             placeholderString: placeholderString
54.         }
55.     }
56.
57.     static #objectToUpdate(o) {
58.         let fields = Object.getOwnPropertyNames(o);
59.         fields.pop();
60.         return fields.join(" = ?, ") + " = ?";
61.     }
62.
63. }

```

SQL 类提供了 **insert**（数据插入）、**select**（数据查询）、**update**（数据修改）、**count**（统计数据条目）、**fuzzySearch**（模糊搜索）等静态方法，通过传入参数，可生成包含模板语句与参数的数据库操作对象。

鉴于项目中涉及指令匹配等须一次执行若干 SQL 语句的情况，项目提供了**数据库事务类 Transaction**：

```

1. class Transaction {
2.
3.     static transaction(sqls, params) {

```

```

4.         return new Promise((resolve, reject) => {
5.             db.getConnection(function (err, connection) {
6.                 if (err) {
7.                     return reject(err);
8.                 }
9.                 if (sqls.length !== params.length) {
10.                    connection.release();
11.                    return reject(new Error("database_transaction: sql and p
12.                    aram don't match"));
13.                }
14.                // execute transaction
15.                connection.beginTransaction((beginErr) => {
16.                    if (beginErr) {
17.                        connection.release();
18.                        return reject(beginErr);
19.                    }
20.                    // console.log("database_transaction: executing " + sqls
21.                    .length + " SQLs");
22.                    // return promise array
23.                    let promises = sqls.map((sql, index) => {
24.                        return new Promise((sqlResolve, sqlReject) => {
25.                            const data = params[index];
26.                            connection.query(sql, data, (err, result) => {
27.                                if (err) {
28.                                    return sqlReject(err);
29.                                }
30.                                sqlResolve(result);
31.                            });
32.                        });
33.                    });
34.                    // check all promises resolved
35.                    Promise.all(promises)
36.                        .then((results) => {
37.                            // commit transaction
38.                            connection.commit(function (err, info) {
39.                                if (err) {
40.                                    console.log("database_transaction: commi
41.                                    t failed" + err);
42.                                    connection.rollback(function (err) {
43.                                        if (err) console.log("database_trans
44.                                        action: rollback failed" + err);
45.                                        connection.release();
46.                                    });
47.                                }
48.                                return reject(err);

```

```

44.         }
45.         connection.release();
46.         resolve(results);
47.     });
48. })
49. .catch((err) => {
50.     connection.rollback(function () {
51.         console.log("database_transaction: has error
    " + err);
52.         connection.release();
53.         reject(err);
54.     });
55. });
56. });
57. });
58. });
59. }
60.
61. }
  
```

Transaction 类提供了 transaction（执行数据库事务）静态方法。调用该方法时，传入 SQL 语句列表（`sqls`）与对应的参数列表的列表（`params`），两列表 SQL 语句、参数表一一对应，通过 `connection.query` 方法执行预操作，并返回 Promise 对象。全部查询执行完毕后，使用 `Promise.all` 判断语句预操作是否全部成功，若全部成功，则调用 `connection.commit` 方法，提交批量操作，完成此次事务，否则调用 `connection.rollback` 方法，回滚数据至调用 `transaction` 方法之前的状态。

数据库事务类的应用，极大简化项目中批量执行 SQL 语句的业务逻辑。数据库事务的原子性，也为项目数据库数据一致性提供了充分的保障，使系统更加稳定可靠。

### 2.4.3.3 DAO 层相关类

在 MVC 设计模式中，DAO（Data Access Object，数据访问对象）层位于 Controller 层与数据库之间。DAO 层定义并提供了一组数据库操作方法，使得 Controller 层模块可调用这些方法与数据库进行交互，完成业务逻辑。一般地，数据表（数据模型）与 DAO 模块之间具有一一对应关系。参照 [2.4.3.1](#) 中的数据表设计，项目为用户（`user`）、声纹（`voiceprint`）、音频（`audio_recognition`）、指令（`command`）四个数据实体分别提供 DAO 类：UserDao、VoiceprintDao、AudioDao、CommandDao。

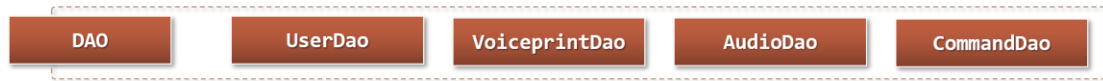


图 18 DAO 层相关类设计

用户数据表访问类 UserDao 定义方法如下：

UserDao		
方法	参数	备注
selectUser	id	根据 id 获取用户信息
selectUsers	page_index	批量获取用户信息
login	username, password	用户登录
register	nickname, username, password, tel	用户注册（简）
fullRegister	username, password, realname, nickname, tel, email	用户注册（全）
update	realname, nickname, avatar, tel, email, id	用户信息修改

```

1. class UserDao {
2.
3.     static async selectUser(id) {
4.         let sql = SQL.generateSQL(SQL.selectById({table_name: "user", id: id
5.         }));
6.         return await Query.query(sql.sql);
7.     }
8.
9.     static async selectUsers(page_index) {
10.        let sql = SQL.generateSQL(SQL.select({
11.            table_name: "user",
12.            condition: `LIMIT ${page_index * Config.batch}, ${page_index +
13.            1) * Config.batch}`
14.        }));
15.        let userData = await Query.query(sql.sql);
16.        sql = SQL.generateSQL(SQL.count({table_name: "user"}));
17.        let userCount = await Query.query(sql.sql);
18.        return {
19.            batch: Config.batch,
20.            total_pages: Math.ceil(userCount[0]["count"] / Config.batch),
21.            user_data: userData,
22.            user_count: userCount[0]["count"]
23.        };
24.    }
25.
26.     static async login({username, password}) {
27.         let sql = SQL.generateSQL(SQL.select({
28.             table_name: "user",

```

```

28.         condition: `WHERE password = '${password}' AND username = '${use
    rname}' OR tel = '${username}' OR email = '${username}'`
29.     });
30.     return await Query.query(sql.sql, sql.params);
31. }
32.
33. static async register({nickname, username, password, tel}) {
34.     let params = {nickname, username, password, tel};
35.     let sql = SQL.generateSQL(SQL.insert({
36.         table_name: "user",
37.         params: params
38.     }), [nickname, username, password, tel]);
39.     return await Query.query(sql.sql, sql.params);
40. }
41.
42. static async fullRegister({username, password, realname, nickname, tel,
    email}) {
43.     let params = {username, password, realname, nickname, tel, email};
44.     let sql = SQL.generateSQL(SQL.insert({
45.         table_name: "user",
46.         params: params
47.     }), [username, password, realname, nickname, tel, email]);
48.     return await Query.query(sql.sql, sql.params);
49. }
50.
51.
52. static async update({realname, nickname, avatar, tel, email, id}) {
53.     let params = {realname, nickname, avatar, tel, email, id};
54.     let sql = SQL.generateSQL(SQL.update({table_name: "user", params: pa
    rams, condition: `WHERE id = ?`}), [realname, nickname, avatar, tel, email,
    id]);
55.     return await Query.query(sql.sql, sql.params);
56. }
57.
58. }
    
```

声纹数据表访问类 VoiceprintDao 定义方法如下：

VoiceprintDao		
方法	参数	备注
selectVoiceprint	id	根据 id 获取声纹信息
selectVoiceprintByUserId	user_id	根据用户 id 获取声纹信息
selectVoiceprints	page_index	批量获取声纹信息
insertVoiceprint	user_id, url	添加声纹

```
1. class VoiceprintDao {
2.
3.     static async selectVoiceprint(id) {
4.         let sql = SQL.generateSQL(SQL.selectById({table_name: "voiceprint",
5.             id: id}));
6.         return await Query.query(sql.sql);
7.     }
8.
9.     static async selectVoiceprintByUserId(user_id) {
10.        let sql = SQL.generateSQL(SQL.select({
11.            table_name: "voiceprint",
12.            condition: `WHERE user_id = ${user_id} ORDER BY time_modified DE
13.                SC`
14.        }));
15.        return await Query.query(sql.sql);
16.    }
17.
18.    static async selectVoiceprints(page_index) {
19.        let sql = SQL.generateSQL(SQL.select({
20.            table_name: "voiceprint",
21.            condition: `LIMIT ${page_index * Config.batch}, ${page_index +
22.                1) * Config.batch}`
23.        }));
24.        let voiceprintData = await Query.query(sql.sql);
25.        sql = SQL.generateSQL(SQL.count({table_name: "voiceprint"}));
26.        let voiceprintCount = await Query.query(sql.sql);
27.        return {
28.            batch: Config.batch,
29.            total_pages: Math.ceil(voiceprintCount[0]["count"] / Config.batc
30.                h),
31.            voiceprint_data: voiceprintData,
32.            voiceprint_count: voiceprintCount[0]["count"]
33.        };
34.    }
35.
36.    static async insertVoiceprint({user_id, url}) {
37.        let params = {user_id, url};
38.        let sql = SQL.generateSQL(SQL.insert({table_name: "voiceprint", para
39.            ms: params}), [user_id, url]);
40.        return await Query.query(sql.sql, sql.params);
41.    }
42.}
```



音频数据表访问类 **AudioDao** 定义方法如下：

VoiceprintDao		
方法	参数	备注
selectAudio	id	根据 id 获取音频信息
selectAudioByUserId	user_id	根据用户 id 获取音频信息
selectAudioBySerial	serial	根据序列号获取音频信息
selectAudios	page_index	批量获取音频信息
insertAudio	serial, user_id, url, base64	添加音频
updateResult	result, id	修改音频语义识别结果

```

1. class AudioDao {
2.
3.     static async selectAudio(id) {
4.         let sql = SQL.generateSQL(SQL.selectById({table_name: "audio_recognition", id: id}));
5.         return await Query.query(sql.sql);
6.     }
7.
8.     static async selectAudioByUserId(user_id) {
9.         let sql = SQL.generateSQL(SQL.select({table_name: "audio_recognition", condition: `WHERE user_id = ${user_id}`}));
10.        return await Query.query(sql.sql);
11.    }
12.
13.    static async selectAudioBySerial(serial) {
14.        let sql = SQL.generateSQL(SQL.select({table_name: "audio_recognition", condition: `WHERE serial = ${serial}`}));
15.        return await Query.query(sql.sql);
16.    }
17.
18.    static async selectAudios(page_index) {
19.        let sql = SQL.generateSQL(SQL.select({
20.            table_name: "audio_recognition",
21.            condition: `LIMIT ${page_index * Config.batch}, ${page_index + 1) * Config.batch}`
22.        }));
23.        let audioData = await Query.query(sql.sql);
24.        sql = SQL.generateSQL(SQL.count({table_name: "audio_recognition"}));
25.        let audioCount = await Query.query(sql.sql);
26.        return {
27.            batch: Config.batch,
28.            total_pages: Math.ceil(audioCount[0]["count"] / Config.batch),
29.            audio_data: audioData,

```

```

30.         audio_count: audioCount[0]["count"]
31.     };
32. }
33.
34.     static async insertAudio({serial, user_id, url, base64 = ""}) {
35.         let params = {serial, user_id, url, base64};
36.         let sql = SQL.generateSQL(SQL.insert({table_name: "audio_recognition",
37.             params: params}), [serial, user_id, url, base64]);
38.         return await Query.query(sql.sql, sql.params);
39.     }
40.
41.     static async updateResult({result, id}) {
42.         let params = {result, id};
43.         let sql = SQL.generateSQL(SQL.update({table_name: "audio_recognition",
44.             params: params, condition: `WHERE id = ?`}), [result, id]);
45.         return await Query.query(sql.sql, sql.params);
46.     }
47. }
    
```

指令数据表访问类 CommandDao 定义方法如下：

VoiceprintDao		
方法	参数	备注
containsParam	content	检查指令内容是否含有参数
extractParams	wordList	自关键词表中提取参数
setParams	commandData, params	使用模板与参数生成指令
selectCommand	id	根据 id 获取指令信息
selectCommands	page_index	批量获取指令信息
register	keyword, description, content	注册指令
update	keyword, description, content, id	指令信息修改
switch	is_valid, id	指令开关
matchCommandsByKeyword	wordList	按关键词匹配指令

```

1. class CommandDao {
2.
3.     static containsParam(content) {
4.         return content.indexOf("?") !== -1;
5.     }
6.
7.     static extractParams(wordList) {
8.         let filteredList = wordList.filter((word) => {
9.             return VerificationHelper.numberVerification(word.Word);
10.        });
11.        let paramList = [];
    
```

```
12.         filteredList.forEach((item) => {
13.             paramList.push(item.Word);
14.         });
15.         return paramList;
16.     }
17.
18.     static setParams(commandData, params) {
19.         let flag = 0;
20.         let content = commandData.content.split("");
21.         for (let i = 0; i < content.length; i++) {
22.             if (content[i] === "?") {
23.                 content[i] = params[flag++];
24.             }
25.         }
26.         commandData.content = content.join("");
27.         return commandData;
28.     }
29.
30.     static async selectCommand(id) {
31.         let sql = SQL.generateSQL(SQL.selectById({table_name: "audio_command", id: id}));
32.         return await Query.query(sql.sql);
33.     }
34.
35.     static async selectCommands(page_index) {
36.         let sql = SQL.generateSQL(SQL.select({
37.             table_name: "audio_command",
38.             condition: `LIMIT ${page_index * Config.batch}, ${page_index + 1) * Config.batch}`
39.         }));
40.         let commandData = await Query.query(sql);
41.         sql = SQL.generateSQL(SQL.count({table_name: "audio_command"}));
42.         let commandCount = await Query.query(sql.sql);
43.         return {
44.             batch: Config.batch,
45.             total_pages: Math.ceil(commandCount[0]["count"] / Config.batch),
46.
47.             command_data: commandData,
48.             command_count: commandCount[0]["count"]
49.         };
50.     }
51.     static async register({keyword, description, content}) {
52.         let params = {keyword, description, content};
```

```
53.         let sql = SQL.generateSQL(SQL.insert({
54.             table_name: "audio_command",
55.             params: params
56.         })), [keyword, description, content]);
57.         return await Query.query(sql.sql, sql.params);
58.     }
59.
60.     static async update({keyword, description, content, id}) {
61.         let params = {keyword, description, content, id};
62.         let sql = SQL.generateSQL(SQL.update({table_name: "audio_command", p
        arams: params, condition: `WHERE id = ?`}), [keyword, description, content,
        id]);
63.         return await Query.query(sql.sql, sql.params);
64.     }
65.
66.     static async switch({is_valid, id}) {
67.         let params = {is_valid, id};
68.         let sql = SQL.generateSQL(SQL.update({table_name: "audio_command", p
        arams: params, condition: `WHERE id = ?`}), [is_valid, id]);
69.         return await Query.query(sql.sql, sql.params);
70.     }
71.
72.     static async matchCommandsByKeyword(wordList) {
73.         let sqls = [];
74.         let params = [];
75.         wordList.forEach((word) => {
76.             sqls.push(SQL.fuzzySearch({
77.                 keyword: word.Word,
78.                 table_name: "audio_command",
79.                 column: "keyword",
80.                 condition: "AND is_valid = 1"
81.             }));
82.             params.push([]);
83.         });
84.         return await Transaction.transaction(sqls, params);
85.     }
86.
87. }
```

#### 2.4.3.4 Controller 层

在 MVC 设计模式中，Controller（控制器）层负责具体业务逻辑的控制。Controller 层位于 DAO 层之上，通过处理请求、执行业务逻辑、做出响应，完成后端对前端发起请求的

处理。一般地，控制器与具体业务逻辑之间具有一一对应关系。项目针对**用户（user）**、**声纹（voiceprint）**、**音频（audio\_recognition）**、**指令（command）**的相关操作，将 Controller 分为 4 个类别，以路由（router）形式挂载至服务器主程序。



图 19 Controller 层设计

用户路由定义控制器如下：

user	
控制器	业务逻辑
register	面向用户终端应用的用户注册
full_register	面向管理系统的用户注册
login	用户登录
get_users	批量获取用户信息
update	修改用户信息

```

1. router.post("/test", PermissionHelper.tokenVerification, async (req, res) =>
  {
2.   try {
3.     let result = await UserDao.selectUser(1);
4.     res.status(200).send({
5.       code: 200,
6.       message: MessageHelper.success,
7.       info: result
8.     });
9.   } catch (err) {
10.    console.log(err);
11.    res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({}
12.    ));
13.  }
14. });
15. router.post("/register", async (req, res) => {
16.   try {
17.     let data = req.body;
18.     let nickname = data.nickname;
19.     let username = data.username;
20.     let password = CryptoHelper.aesEncrypt(data.password);
21.     let tel = data.tel;
22.     let result = await UserDao.register({nickname: nickname, username: u
23.     sername, password: password, tel: tel});
24.     res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({}));
25.   } catch (err) {

```

```
25.         console.log(err);
26.         if (err.code === EnumHelper.databaseException.ER_DUP_ENTRY) {
27.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noConte
                nt({message: MessageHelper.user_register_duplication}));
28.             return;
29.         }
30.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({
            }));
31.     }
32. });
33.
34. router.post("/full_register", PermissionHelper.tokenVerification, Permission
    Helper.managerVerification, async (req, res) => {
35.     try {
36.         let data = req.body;
37.         let username = data.username;
38.         let password = CryptoHelper.aesEncrypt(data.password);
39.         let realname = data.realname;
40.         let nickname = data.nickname;
41.         let tel = data.tel;
42.         let email = data.email;
43.         let result = await UserDao.fullRegister({
44.             username: username,
45.             password: password,
46.             realname: realname,
47.             nickname: nickname,
48.             tel: tel,
49.             email: email
50.         });
51.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({}));
52.     } catch (err) {
53.         console.log(err);
54.         if (err.code === EnumHelper.databaseException.ER_DUP_ENTRY) {
55.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noConte
                nt({message: MessageHelper.user_register_duplication}));
56.             return;
57.         }
58.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({
            }));
59.     }
60. });
61.
62. router.post("/login", async (req, res) => {
63.     try {
```

```
64.         let data = req.body;
65.         let username = data.username;
66.         let password = CryptoHelper.aesEncrypt(data.password);
67.         let result = await UserDao.login({username: username, password: password});
68.         if (!result.length) {
69.             // user DNE
70.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noContent({
71.                 message: MessageHelper.login_failed,
72.             }));
73.         } else {
74.             // generate token
75.             let userinfo = result[0];
76.             let user_id = userinfo.id;
77.             let token = new JWTHelper(user_id);
78.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({
79.                 info: {
80.                     userinfo: userinfo,
81.                     token: token.generateToken()
82.                 }
83.             }));
84.         }
85.     } catch (err) {
86.         console.log(err);
87.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({}));
88.     }
89. });
90.
91. router.post("/get_users", PermissionHelper.tokenVerification, PermissionHelper.managerVerification, async (req, res) => {
92.     try {
93.         let data = req.body;
94.         let page_index = data.page_index;
95.         let result = await UserDao.selectUsers(page_index);
96.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({info: result}));
97.     } catch (err) {
98.         console.log(err);
99.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({}));
100.    }
101. });
```

```

102.
103. router.post("/update", PermissionHelper.tokenVerification, PermissionHelper
    .managerVerification, async (req, res) => {
104.     try {
105.         let data = req.body;
106.         let id = data.id;
107.         let realname = data.realname;
108.         let nickname = data.nickname;
109.         let avatar = data.avatar;
110.         let tel = data.tel;
111.         let email = data.email;
112.         let result = await UserDao.update({
113.             realname: realname,
114.             nickname: nickname,
115.             avatar: avatar,
116.             tel: tel,
117.             email: email,
118.             id: id
119.         });
120.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({}));
121.     } catch (err) {
122.         console.log(err);
123.         if (err.code === EnumHelper.databaseException.ER_DUP_ENTRY) {
124.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noCont
                ent({message: MessageHelper.user_register_duplication}));
125.             return;
126.         }
127.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({
            }));
128.     }
129. });
  
```

声纹路由定义控制器如下：

voiceprint	
控制器	业务逻辑
get_voiceprint	根据用户登录令牌获取用户声纹信息
get_voiceprints	批量获取用户声纹信息

```

1. router.post("/get_voiceprint", PermissionHelper.tokenVerification, async (re
    q, res) => {
2.     try {
3.         let tokenData = JWTHelper.parseToken(req);
4.         let user_id = tokenData.user_id;
  
```



```

5.         let result = await VoiceprintDao.selectVoiceprintByUserId(user_id);
6.         console.log(result);
7.         if (!result.length) {
8.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noContent({info: MessageHelper.voiceprint_unregistered}));
9.         } else {
10.            let data = result[0];
11.            res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({info: data}));
12.        }
13.    } catch (err) {
14.        console.log(err);
15.        res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({}));
16.    }
17. });
18.
19. router.post("/get_voiceprints", PermissionHelper.tokenVerification, PermissionHelper.managerVerification, async (req, res) => {
20.     try {
21.         let data = req.body;
22.         let page_index = data.page_index;
23.         let result = await VoiceprintDao.selectVoiceprints(page_index);
24.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({info: result}));
25.     } catch (err) {
26.         console.log(err);
27.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({}));
28.     }
29. });

```

音频路由定义控制器如下：

audio	
控制器	业务逻辑
recognition	根据 id 对语音进行语义识别，并保存语义识别结果
get_audios	批量获取音频信息
instruction	根据 id 对语音进行指令转换
command	根据音频序列号，将音频映射为指令，并向硬件网关发送控制请求

```

1. router.post("/recognition", PermissionHelper.tokenVerification, async (req, res) => {
2.     try {

```

```
3.         let tokenData = JWTHelper.parseToken(req);
4.         let user_id = tokenData.user_id;
5.         let data = req.body;
6.         let audio_id = data.audio_id;
7.         let result = await AudioDao.selectAudio(audio_id);
8.         if (!result.length) {
9.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noConte
nt({
10.                 message: MessageHelper.audio_empty,
11.             }));
12.         } else {
13.             let audioData = result[0];
14.             let recognitionResult = await TencentCloudHelper.sentenceRecogni
tion({
15.                 userAudioKey: audioData.serial,
16.                 audioFileDirectory: audioData.url
17.             });
18.             result = await AudioDao.updateResult({result: JSON.stringify(rec
ognitionResult), id: audio_id});
19.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({inf
o: recognitionResult}));
20.         }
21.     } catch (err) {
22.         console.log(err);
23.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({}
));
24.     }
25. });
26.
27. router.post("/get_audios", PermissionHelper.tokenVerification, PermissionHel
per.managerVerification, async (req, res) => {
28.     try {
29.         let data = req.body;
30.         let page_index = data.page_index;
31.         let result = await AudioDao.selectAudios(page_index);
32.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({info: r
esult}));
33.     } catch (err) {
34.         console.log(err);
35.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({}
));
36.     }
37. });
38.
```

```
39. router.post("/instruction", PermissionHelper.tokenVerification, async (req,
    res) => {
40.     try {
41.         let tokenData = JWTHelper.parseToken(req);
42.         let user_id = tokenData.user_id;
43.         let data = req.body;
44.         let audio_id = req.body.audio_id;
45.         let result = await AudioDao.selectAudio(audio_id);
46.         if (!result.length) {
47.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noConte
                nt({
48.                 message: MessageHelper.audio_empty,
49.             }));
50.         } else {
51.             let audioData = result[0];
52.             audioData.result = JSON.parse(audioData.result);
53.             if (!audioData.result.hasOwnProperty("RequestId")) {
54.                 res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noC
                    ontent({message: MessageHelper.audio_instruction_unabled}));
55.                 return;
56.             }
57.             let wordList = audioData.result.WordList;
58.             result = await CommandDao.matchCommandsByKeyword(wordList);
59.             let count = [];
60.             result.forEach((sub) => {
61.                 sub.forEach((item) => {
62.                     count.push(item.id);
63.                 });
64.             });
65.             if (!count.length) {
66.                 res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noC
                    ontent({message: MessageHelper.audio_instruction_failed}));
67.                 return;
68.             }
69.             result = await CommandDao.selectCommand(Number(ObjectHelper.find
                Most(count)));
70.             let commandData = result[0];
71.             if (CommandDao.containsParam(commandData.content)) {
72.                 let params = CommandDao.extractParams(wordList);
73.                 commandData = CommandDao.setParams(commandData, params);
74.             }
75.             // result = await HardwareRequest.excute(JSON.parse(commandData.
                content));
76.             // console.log(result.message);
```

```
77.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({
78.             info: {
79.                 audio_serial: audioData.serial,
80.                 keyword: commandData.keyword,
81.                 content: commandData.content
82.             }
83.         }));
84.     }
85. } catch (err) {
86.     console.log(err);
87.     res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noContent({
88.         message: MessageHelper.audio_instruction_failed}));
89. });
90.
91. router.post("/command", PermissionHelper.tokenVerification, async (req, res)
92.     => {
93.     try {
94.         let tokenData = JWTHelper.parseToken(req);
95.         let user_id = tokenData.user_id;
96.         let data = req.body;
97.         let serial = data.serial;
98.         let result = await AudioDao.selectAudioBySerial(serial);
99.         if (!result.length) {
100.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noContent({message: MessageHelper.audio_empty}));
101.         } else {
102.             let audioData = result[0];
103.             if (audioData.result === "{}") {
104.                 let recognitionResult = await TencentCloudHelper.sentenceRecognition({
105.                     userAudioKey: audioData.serial,
106.                     audioFileDirectory: audioData.url
107.                 });
108.                 audioData.result = recognitionResult;
109.                 await AudioDao.updateResult({result: JSON.stringify(recognitionResult), id: audioData.id});
110.             } else {
111.                 audioData.result = JSON.parse(audioData.result);
112.             }
113.             // audioData.result should be usable here
114.             let wordList = audioData.result.WordList;
115.             result = await CommandDao.matchCommandsByKeyword(wordList);
116.             let count = [];
```

```

116.         result.forEach((sub) => {
117.             sub.forEach((item) => {
118.                 count.push(item.id);
119.             });
120.         });
121.         if (!count.length) {
122.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.no
Content({message: MessageHelper.audio_instruction_failed}));
123.             return;
124.         }
125.         result = await CommandDao.selectCommand(Number(ObjectHelper.fin
dMost(count)));
126.         let commandData = result[0];
127.         if (CommandDao.containsParam(commandData.content)) {
128.             let params = CommandDao.extractParams(wordList);
129.             commandData = CommandDao.setParams(commandData, params);
130.         }
131.         result = await HardwareRequest.excute(JSON.parse(commandData.co
ntent));
132.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({
133.             info: {
134.                 keyword: commandData.keyword,
135.                 content: commandData.content,
136.                 hardware: result
137.             }
138.         }));
139.     }
140. } catch (err) {
141.     console.log(err.code);
142.     if (err.code === EnumHelper.networkException.ECONNREFUSED) {
143.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noCont
ent({message: MessageHelper.device_disconnected}));
144.         return;
145.     }
146.     res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noContent(
{message: MessageHelper.audio_instruction_failed}));
147. }
148. });
  
```

指令路由定义控制器如下：

command	
控制器	业务逻辑
register	根据 id 对语音进行语义识别，并保存语义识别结果

get_commands	批量获取指令信息
switch	启用或禁用指令

```

1. router.post("/register", PermissionHelper.tokenVerification, PermissionH
 elper.managerVerification, async (req, res) => {
2.   try {
3.     let data = req.body;
4.     let keyword = data.keyword;
5.     let description = data.description;
6.     let content = data.content;
7.     let result = await CommandDao.register({keyword: keyword, content: c
  ontent, description: description});
8.     res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({}));
9.   } catch (err) {
10.    console.log(err);
11.    if (err.code === EnumHelper.databaseException.ER_DUP_ENTRY) {
12.      res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noConte
  nt({message: MessageHelper.command_keyword_duplicate}));
13.      return;
14.    }
15.    res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({
  }));
16.  }
17. });
18.
19. router.post("/get_commands", PermissionHelper.tokenVerification, PermissionH
  elper.managerVerification, async (req, res) => {
20.   try {
21.     let data = req.body;
22.     let page_index = data.page_index;
23.     let result = await CommandDao.selectCommands(page_index);
24.     res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({info: r
  esult}));
25.   } catch (err) {
26.     console.log(err);
27.     res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({
  }));
28.   }
29. });
30.
31. router.post("/switch", PermissionHelper.tokenVerification, PermissionHelper.
  managerVerification, async (req, res) => {
32.   try {
33.     let data = req.body;
34.     let command_id = data.id;
  
```

```

35.         let is_valid = data.is_valid;
36.         let result = await CommandDao.switch({is_valid: is_valid, id: command_id});
37.         res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({}));
38.     } catch (err) {
39.         console.log(err);
40.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({}));
41.     }
42. });
  
```

### 2.4.3.5 指令模板与匹配机制

#### (1) 指令模板：

项目中，指令模板的结构与 JSON 字符串类似。每个指令模板都包含外设（**device**）字段与参数字段。不同外设的指令，参数名也不尽相同。参数值以“?”占位符代替。

比如 LED 控制指令，其外设为 LED，参数为 **brightness**：

```

1. {
2.     "device": "LED",
3.     "brightness": ?
4. }
  
```

在数据库中，每个指令模板可对应一个或多个关键词，如 LED 控制指令对应了“调整 LED 亮度”、“打开 LED”、“关闭 LED”等指令，而对于“打开 LED”“关闭 LED”，其亮度参数 **brightness** 直接为可设定亮度的最大值（255）与最小值（0）。

#### (2) 匹配机制：

项目中，指令为语音语义识别的词组对数据库指令表（**audio\_command**）的关键词字段（**keyword**）进行匹配得到。

在前端应用（用户终端应用、管理系统）向项目服务器上传音频，并调用相关接口时，服务器首先调用第三方接口，对该音频进行语义识别：

```

1. let recognitionResult = await TencentCloudHelper.sentenceRecognition({
2.     userAudioKey: audioData.serial,
3.     audioFileDirectory: audioData.url
4. });
  
```

得到语义识别词组：

```

1. [
2.     { Word: '调整', StartTime: 0, EndTime: 510 },
3.     { Word: 'led', StartTime: 510, EndTime: 1230 },
  
```

```

4.    { Word: '亮度', StartTime: 1230, EndTime: 1830 },
5.    { Word: '至', StartTime: 1920, EndTime: 2250 },
6.    { Word: '127', StartTime: 2610, EndTime: 3390 }
7.  ]

```

此后调用 `CommandDao.matchCommandsByKeyword` 方法，检索指令：

```
1. result = await CommandDao.matchCommandsByKeyword(wordList);
```

`CommandDao.matchCommandsByKeyword` 以上述词组为参数，在数据库中，以指令表（`audio_command`）的关键词字段（`keyword`）为匹配对象，对词组中的每个单词进行模糊匹配：

```

1. let sqls = [];
2. let params = [];
3. wordList.forEach((word) => {
4.   sqls.push(SQL.fuzzySearch({
5.     keyword: word.Word,
6.     table_name: "audio_command",
7.     column: "keyword",
8.     condition: "AND is_valid = 1"
9.   }));
10.  params.push([]);
11. });
12. return await Transaction.transaction(sqls, params);

```

其结果为一个二维列表，其长度等于上述词组长度，每个元素均为上述词组对应位置的单词在指令表中检索出的指令信息：

```

1. [
2.   [
3.     {
4.       id: 1,
5.       keyword: '调整 LED 亮度',
6.       content: '{"device": "LED", "brightness": ?}',
7.       description: '该指令用于调整 LED 亮度',
8.       is_valid: 1,
9.       time_created: 2022-07-21T06:37:31.000Z,
10.      time_modified: 2022-07-24T13:53:23.000Z
11.    }
12.  ],
13.  [
14.    {
15.      id: 1,
16.      keyword: '调整 LED 亮度',
17.      content: '{"device": "LED", "brightness": ?}',
18.      description: '该指令用于调整 LED 亮度',

```



```
19.     is_valid: 1,
20.     time_created: 2022-07-21T06:37:31.000Z,
21.     time_modified: 2022-07-24T13:53:23.000Z
22.   },
23.   {
24.     id: 2,
25.     keyword: '关闭 LED',
26.     content: '{"device": "LED", "brightness": 0}',
27.     description: '该指令用于关闭 LED',
28.     is_valid: 1,
29.     time_created: 2022-07-22T09:14:56.000Z,
30.     time_modified: 2022-07-24T13:53:23.000Z
31.   }
32. ],
33. [
34.   {
35.     id: 1,
36.     keyword: '调整 LED 亮度',
37.     content: '{"device": "LED", "brightness": ?}',
38.     description: '该指令用于调整 LED 亮度',
39.     is_valid: 1,
40.     time_created: 2022-07-21T06:37:31.000Z,
41.     time_modified: 2022-07-24T13:53:23.000Z
42.   }
43. ],
44. [],
45. []
46. ]
```

将得到的指令 id（不去重）保存至计数列表 **count** 中：

```
1. let count = [];
2. result.forEach((sub) => {
3.   sub.forEach((item) => {
4.     count.push(item.id);
5.   });
6. });
```

若 **count** 仍为空列表，则表示未找到目标指令，指令匹配失败，响应用户，结束本次业务逻辑：

```
1. if (!count.length) {
2.   res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noContent({message: MessageHelper.audio_instruction_failed}));
3.   return;
4. }
```

调用预先封装的 `ObjectHelper.findMost` 方法，找出列表中出现次数最多的指令 `id`，并获取该 `id` 对应的指令信息：

```
1. result = await CommandDao.selectCommand(Number(ObjectHelper.findMost(count))  
    );
```

若上述指令的指令模板含有待定参数“？”，则调用预先封装的基于正则匹配的参数提取方法 `CommandDao.extractParams`，提取参数，并调用参数设定方法 `CommandDao.setParams`，生成确定参数的指令控制信息：

```
1. let commandData = result[0];  
2. if (CommandDao.containsParam(commandData.content)) {  
3.     let params = CommandDao.extractParams(wordList);  
4.     commandData = CommandDao.setParams(commandData, params);  
5. }
```

对于上述指令，生成的控制信息为：

```
1. '{"device": "LED", "brightness": 127}'
```

至此，带有外设（LED）与确定参数（`brightness`）的硬件控制指令已经生成。

#### 2.4.3.6 鉴权机制与拦截器

##### （1）鉴权机制：

项目服务器采用基于 **Token** 的鉴权机制。

**Token**，又称登录令牌，是服务端使用密钥加密数据后生成的一个字符串。一般地，首次登录后，服务器根据用户信息，签发 **Token** 并将其返回给客户端，此后客户端只须携带此 **Token** 请求数据即可。当服务器收到请求时，将首先验证 **Token**，若验证成功，则执行后续操作，否则返回错误信息。

在 `jwt_helper.js` 中，定义了 **Token** 操作与验证类 `JWTHelper`：

```
1. class JWTHelper {  
2.  
3.     #data;  
4.  
5.     constructor(user_id) {  
6.         this.#data = {  
7.             user_id: user_id,  
8.             timestamp: Date.now()  
9.         };  
10.    }  
11.  
12.    generateToken() {
```

```
13.         return JWT.sign(this.#data, Config.aes_key);
14.     }
15.
16.     static verifyToken(req) {
17.         let token = req.headers["token"];
18.         if (token) {
19.             return JWT.verify(token, Config.aes_key, (err, decode) => {
20.                 if (err) {
21.                     return false;
22.                 } else {
23.                     let tokenData = JWT.decode(token);
24.                     if (Date.now() - tokenData.timestamp >= expire_time) {
25.                         return false;
26.                     } else {
27.                         return true;
28.                     }
29.                 }
30.             })
31.         } else {
32.             return false;
33.         }
34.     }
35.
36.     static parseToken(req) {
37.         let token = req.headers["token"];
38.         return JWT.decode(token);
39.     }
40.
41. }
```

在构造方法 `constructor` 中，传入用户 `id`，即生成具有用户 `id` 与签发 `Token` 时间戳的 `JWTHelper` 对象。

`generateToken` 方法用于生成 `Token`。该方法使用服务器配置文件 (`config.js`) 中的 `AES` 密钥对具有用户 `id` 与时间戳的数据对象进行加密，并返回加密后的字符串，即 `Token`。

`verifyToken` 方法用于检验 `Token`。该方法使用服务器配置文件 (`config.js`) 中的 `AES` 密钥对请求头 (Header) 中的 `Token` 进行解密，若解密失败，则返回 `false`；解密成功时，还须检查解出的时间戳与当前时间的间隔时间。若该段间隔时间长于服务器配置文件 (`config.js`) 中的间隔时间，说明此 `Token` 过期，则返回 `false`，否则返回 `True`。

`parseToken` 方法用于解析 `Token` 数据。该方法可实现接口参数的简化，即直接自 `Token` 中解析用户身份，而无须在接口中传输表征用户身份的唯一标识。显而易见的是，此方法对

于防止横向越权具有极好的效果。

```
1. class PermissionHelper {
2.
3.     static tokenVerification(req, res, next) {
4.         if (JWTHelper.verifyToken(req)) {
5.             next();
6.         } else {
7.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.
unauthorized({}));
8.         }
9.     }
10.
11.    static async rootVerification(req, res, next) {
12.        try {
13.            let tokenData = JWTHelper.parseToken(req);
14.            let result = await UserDao.selectUser(tokenData.user_id);
15.
16.            if (!result.length) {
17.                res.status(EnumHelper.HTTPStatus.OK).send(ResponseHel
per.noContent({}));
18.            } else {
19.                let authority = result[0].authority;
20.                if (authority === EnumHelper.authorityType.root) {
21.                    next();
22.                } else {
23.                    res.status(EnumHelper.HTTPStatus.OK).send(Respons
eHelper.authorityUltraVires({}));
24.                }
25.            } catch (err) {
26.                console.log(err);
27.                res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHel
per.error({}));
28.            }
29.        }
30.
31.    static async managerVerification(req, res, next) {
32.        try {
33.            let tokenData = JWTHelper.parseToken(req);
34.            let result = await UserDao.selectUser(tokenData.user_id);
35.
36.            if (!result.length) {
37.                res.status(EnumHelper.HTTPStatus.OK).send(ResponseHel
per.noContent({}));
```

```

37.         } else {
38.             let authority = result[0].authority;
39.             if (authority === EnumHelper.authorityType.root || au
               thority === EnumHelper.authorityType.manager) {
40.                 next();
41.             } else {
42.                 res.status(EnumHelper.HTTPStatus.OK).send(Respons
                   eHelper.authorityUltraVires({}));
43.             }
44.         }
45.     } catch (err) {
46.         console.log(err);
47.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelp
           er.error({}));
48.     }
49. }
50.
51. }

```

## (2) 拦截器:

**拦截器 (Interceptor)** 源于面向切面编程 (AOP) 理念, 它提供了一种机制, 可使开发者自定义在某行为的前后执行指定代码, 通过拦截的数据与其内部逻辑决定该行为是否应继续进行, 或在某个行为执行前阻止其执行。

在 permission\_helper.js 中, 将拦截器内部逻辑封装为拦截器类 PermissionHelper:

```

1. class PermissionHelper {
2.
3.     static tokenVerification(req, res, next) {
4.         if (JWTHelper.verifyToken(req)) {
5.             next();
6.         } else {
7.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.unautho
               rized({}));
8.         }
9.     }
10.
11.     static async rootVerification(req, res, next) {
12.         try {
13.             let tokenData = JWTHelper.parseToken(req);
14.             let result = await UserDao.selectUser(tokenData.user_id);
15.             if (!result.length) {
16.                 res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noC
                   ontent({}));
17.             } else {

```

```
18.         let authority = result[0].authority;
19.         if (authority === EnumHelper.authorityType.root) {
20.             next();
21.         } else {
22.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper
                .authorityUltraVires({}));
23.         }
24.     }
25. } catch (err) {
26.     console.log(err);
27.     res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.erro
        r({}));
28. }
29. }
30.
31. static async managerVerification(req, res, next) {
32.     try {
33.         let tokenData = JWTHelper.parseToken(req);
34.         let result = await UserDao.selectUser(tokenData.user_id);
35.         if (!result.length) {
36.             res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.noC
                ontent({}));
37.         } else {
38.             let authority = result[0].authority;
39.             if (authority === EnumHelper.authorityType.root || authority
                === EnumHelper.authorityType.manager) {
40.                 next();
41.             } else {
42.                 res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper
                    .authorityUltraVires({}));
43.             }
44.         }
45.     } catch (err) {
46.         console.log(err);
47.         res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.erro
            r({}));
48.     }
49. }
50.
51. }
```

拦截器在项目服务器中具有两个重要应用：Token 验证、用户身份验证。

`tokenVerification` 方法用于 Token 验证。对于传入的请求 `req`，通过 `JWTHelper.verifyToken` 检验登录令牌的有效性。有效时，通过 `next` 方法，放行请求，

进行后续操作；无效时，直接使用 `res.send` 方法，向客户端发送错误信息，拒绝处理请求。

`managerVerificaion`、`rootVerification` 方法用于用户身份验证。在某些仅允许超级管理员，或仅允许管理员、超级管理员调用的接口 `Controller` 中，在插入 `tokenVerification` 拦截器后，按需插入 `managerVerificaion` 或 `rootVerification` 拦截器。对于传入的请求 `req`，此二拦截器通过 `JWTHelper.parseToken` 方法解析出用户 `id` (`user_id`)，并在数据库中检索对应用户的权限信息。得到用户权限信息后，与执行请求所需权限进行比较，若用户权限不低于最低所需权限，则通过 `next` 方法放行请求；否则直接使用 `res.send` 方法，向客户端发送错误信息，拒绝处理请求。显而易见的是，此方法对于防止纵向越权具有极好的效果。

通过应用 `Token` 鉴权机制与拦截器，项目服务器在安全性获得明显提升同时，也为开发者配置接口权限提供了灵活的方法。

如需要 `Token`，且只允许管理员、超级管理员调用的批量获取用户信息控制器 (`/user/get_users`):

```
1. router.post("/get_users", PermissionHelper.tokenVerification, PermissionHelper.managerVerification
```

又如需要 `Token`，但允许任意身份用户调用的音频语义识别控制器 (`/audio/recognition`):

```
1. router.post("/instruction", PermissionHelper.tokenVerification
```

再如无需 `Token` 的用户登录控制器 (`/user/login`):

```
1. router.post("/login"
```

可以看到，按业务逻辑需求，插拔对应的拦截器，即可实现接口鉴权与权限控制。

#### 2.4.3.7 硬件网关通信模块

在 2.4.1 节所述时序中，当指令解析完毕，需要控制硬件外设时，服务器须携带控制指令，向硬件网关发起硬件控制请求。为增强可重用性与可维护性，对该请求进行两层封装——首先基于 `AXIOS` 封装请求工具，可通过指定 `URL`，列出参数(`params`)，说明方法(`method`)快速发送网络请求，此后利用请求工具，以简洁而统一的格式封装为 `API` 文件。

##### (1) 第一层封装——基于 `AXIOS` 封装请求工具：

请求类 `AjaxHelper` 以 `promise` 形式对 `POST` 与 `GET` 请求进行封装。根据传入的参数，自动生成请求体 (`Body`, `POST` 请求) 或查询字符串 (`Query String`, `GET` 请求)，此后使用 `AXIOS` 库提供的请求方法发送请求。

```

1. class AjaxHelper {
2.
3.     static ajax({url, headers, params, method}) {
4.         let promise;
5.         return new Promise((resolve, reject) => {
6.             if (method === EnumHelper.HTTPMethod.GET) {
7.                 let queryString = "";
8.                 Object.keys(params).forEach(key => {
9.                     queryString += key + "=" + params[key] + "&";
10.                });
11.                if (queryString !== "") {
12.                    queryString = queryString.substr(0, queryString.lastIndexOf("&"));
13.                }
14.                url += "?" + queryString;
15.                promise = axios.get(url, {headers: headers, params: params});
16.            } else if (method === EnumHelper.HTTPMethod.POST) {
17.                promise = axios.post(url, params, {headers: headers});
18.            }
19.            promise.then((res) => {
20.                resolve(res.data);
21.            }).catch((err) => {
22.                reject(err);
23.            })
24.        });
25.    }
26.
27. }
  
```

## (2) 第二层封装——基于请求工具封装 AJAX 请求：

在完成请求工具的封装后，只需指定 URL，列出参数（params），说明方法（method），即可快速发送请求。而在 API 文件中，通过 BASE\_URL 定义了请求 URL 的主机与端口信息，每个请求均引用此 BASE\_URL，达到“一次修改，全局适配”的目的。以该格式封装硬件控制请求类 HardwareRequest：

```

1. class HardwareRequest {
2.
3.     static execute = (command_content) => AjaxHelper.ajax({
4.         url: `${BASE_URL}/hardware/execute`,
5.         params: {
6.             command_content: command_content
7.         },
8.         method: EnumHelper.HTTPMethod.POST
  
```



```
9.     });  
10.  
11. }
```

应用时，只需引入 `HardwareRequest` 类，传入指令参数，并异步执行 `execute` 方法：

```
1. let result = await HardwareRequest.execute(JSON.parse(commandData.content));
```

即可向硬件网关发送硬件控制请求。

#### 2.4.3.8 项目实例与主程序

##### (1) 项目实例：

项目实例的定义位于 `app.js` 中。

在 Express 框架下，首先创建服务器应用实例：

```
1. const app = express();
```

此后，引入以路由形式封装的 Controller 层：

```
1. const userRouter = require("./routes/user");  
2. const uploadRouter = require("./routes/upload");  
3. const audioRouter = require("./routes/audio");  
4. const commandRouter = require("./routes/command");  
5. const voiceprintRouter = require("./routes/voiceprint");
```

通过 `app.use` 方法，将 Controller 路由以中间件形式挂载至服务器：

```
1. app.use("/api/v1/user", userRouter);  
2. app.use("/api/v1/upload", uploadRouter);  
3. app.use("/api/v1/audio", audioRouter);  
4. app.use("/api/v1/command", commandRouter);  
5. app.use("/api/v1/voiceprint", voiceprintRouter);
```

通过 `static` 方法托管静态资源，将服务器目录 `/public` 下的文件暴露至公网：

```
1. app.use(express.static(path.join(__dirname, "")));
```

配置跨域响应头（CORS），使项目中的请求响应不必遵循同源策略，便于调试：

```
1. app.all('*', function(req, res, next) {  
2.   res.header("Access-Control-Allow-Origin", "*");  
3.   res.header("Access-Control-Allow-Headers", "Access-Control-Allow-  
    Origin,Content-Type,X-Requested-With,accept,Origin,Access-Control-Request-  
    Method,Access-Control-Request-Headers");  
4.   res.header("Access-Control-Allow-  
    Methods", "PUT,POST,GET,DELETE,OPTIONS");  
5.   res.header("X-Powered-By", ' 3.2.1');  
6.   res.header("Access-Control-Allow-Credentials", true);  
7.   res.header("Content-Type", "application/json;charset=utf-8");
```

```

8.   next();
9. });

```

最后，对外暴露 app 对象：

```

1. module.exports = app;

```

## （2）项目主程序设计：

项目主程序（入口文件）为 **www.js**。

首先规定项目运行端口号（3000）：

```

1. const port = normalizePort(process.env.PORT || "3000");
2. app.set("port", port);

```

通过 **http.createServer** 方法创建服务器，在端口（3000）开启监听：

```

1. const server = http.createServer(app);
2. server.listen(port);

```

配置事件（error、listening、close）监听：

```

1. server.on("error", onError);
2. server.on("listening", onListening);
3. server.on("close", onClose);

```

通过检测 **SIGINT** 监听程序终止，待完成全部请求处理后再终止程序，实现“优雅退出”。

```

1. process.on("SIGINT", () => {
2.   debug("process: exiting...");
3.   server.close(() => {
4.     process.exit(0);
5.   });
6.   debug("process: service terminated (SIGINT)");
7. });

```

### 2.4.3.9 接口文档

本文仅给出接口用途、入参与返回数据示例。详情见[在线接口文档](#)。

项目接口均为 HTTP POST 请求。

#### （1）用户（/user）：

/register	
用途	面向用户终端应用的用户注册
权限	无
入参	<pre> {   "nickname": "asdfasdf",   "username": "username_2",   "password": "password", </pre>

	<pre>"tel": "18810559476" }</pre>
出参	<pre>{   "code": 204,   "message": "该用户已被注册！请重试！" }</pre>

/full_register	
用途	面向管理系统的用户注册
权限	manager root
入参	<pre>{   "nickname": "asdfasdf",   "username": "username_2",   "password": "password",   "tel": "18810559476",   "realname": "付东源",   "email": "951947409@qq.com" }</pre>
出参	<pre>{   "code": 204,   "message": "该用户已被注册！请重试！" }</pre>

/login	
用途	用户登录
权限	无
入参	<pre>{   "username": "username_1",   "password": "Sam_2019" }</pre>
出参	<pre>{   "code": 200,   "message": "操作成功！",   "info": {     "userinfo": {       "id": 1,       "realname": "付东源",       "nickname": "asdfasdf",       "username": "username_1",       "password": "a43cf888951388c4",       "avatar": "public/images/avatar/default.png",       "tel": "18810559476", </pre>

	<pre>         "email": "951947409@qq.com",         "authority": "root",         "is_available": 1,         "time_created": "2022-07-10T14:23:44.000Z",         "time_modified": "2022-07-20T16:26:53.000Z"     },     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjox     LCJ0aW1lc3RhbnXAiOiE2NTg3NTIzNTY3OTgsIm1hdCI6MTY1ODc1MjM1Nn0.VBxXMwSc7N     d386bmdcrjq9Y1SdRoZYp6ZVN2vF4uj7M"     }     }</pre>
--	--

/get_users	
用途	面向管理系统的用户信息批量获取
权限	manager root
入参	<pre> {     "page_index": 0 }</pre>
出参	<pre> {     "code": 200,     "message": "操作成功!",     "info": {         "batch": 8,         "total_pages": 1,         "user_data": [             {                 "id": 1,                 "realname": "付东源",                 "nickname": "asdfasdf",                 "username": "username_1",                 "password": "a43cf888951388c4",                 "avatar": "public/images/avatar/default.png",                 "tel": "18810559476",                 "email": "951947409@qq.com",                 "authority": "root",                 "is_available": 1,                 "time_created": "2022-07-10T14:23:44.000Z",                 "time_modified": "2022-07-20T16:26:53.000Z"             },             {                 "id": 2,                 "realname": "姚辰龙",                 "nickname": "Oaksssss", </pre>

```

        "username": "username_2",
        "password": "a43cf888951388c4",
        "avatar":
"public/images/avatar/PsKc4FQd5suLQCKv7pgkEv8b.png",
        "tel": "18810559477",
        "email": "951947408@qq.com",
        "authority": "manager",
        "is_available": 1,
        "time_created": "2022-07-10T14:24:44.000Z",
        "time_modified": "2022-07-15T09:31:26.000Z"
    },
    {
        "id": 3,
        "realname": "朱子虚",
        "nickname": "Pine",
        "username": "username_3",
        "password": "a43cf888951388c4",
        "avatar": "public/images/avatar/default.png",
        "tel": "18810559478",
        "email": "951947407@qq.com",
        "authority": "user",
        "is_available": 1,
        "time_created": "2022-07-10T14:25:44.000Z",
        "time_modified": "2022-07-15T09:30:37.000Z"
    },
    {
        "id": 8,
        "realname": "小明",
        "nickname": "asdfasdfsfa",
        "username": "username_4",
        "password": "a43cf888951388c4",
        "avatar": "public/images/avatar/default.png",
        "tel": "17304381793",
        "email": "fudongyuan@bupt.edu.cn",
        "authority": "user",
        "is_available": 1,
        "time_created": "2022-07-14T12:07:23.000Z",
        "time_modified": "2022-07-15T09:30:37.000Z"
    }
],
"user_count": 4
}
}

```

/update	
用途	面向管理系统的用户信息修改
权限	manager root
入参	<pre>{   "id": 1,   "nickname": "asdfasdf",   "avatar": "public/images/avatar/default.png",   "password": "password",   "tel": "18810559476",   "realname": "付东源",   "email": "951947409@qq.com" }</pre>
出参	<pre>{   "code": 200,   "message": "操作成功！" }</pre>

## (2) 声纹 (/voiceprint):

/get_voiceprint	
用途	面向用户终端应用的用户声纹信息获取
权限	user manager root
入参	无
出参	<pre>{   "code": 200,   "message": "操作成功！",   "info": {     "id": 4,     "user_id": 1,     "url":       "public/audios/voiceprint/iT_6BbjXiwzjT3K8eNfdNk0a.mp3",     "time_created": "2022-07-23T15:54:20.000Z",     "time_modified": "2022-07-23T15:54:20.000Z"   } }</pre>

/get_voiceprints	
用途	面向管理平台的用户声纹信息批量获取
权限	manager root
入参	<pre>{   "page_index": 0 }</pre>

出参

```
{
  "code": 200,
  "message": "操作成功!",
  "info": {
    "batch": 8,
    "total_pages": 1,
    "voiceprint_data": [
      {
        "id": 1,
        "user_id": 1,
        "url":
"public/audios/voiceprint/3bCG57nSBby1TdrNiC52ZuaA.mp3",
        "time_created": "2022-07-23T12:30:25.000Z",
        "time_modified": "2022-07-23T12:30:25.000Z"
      },
      {
        "id": 2,
        "user_id": 1,
        "url":
"public/audios/voiceprint/AYF7IUvxlnsSA5xDZb4C2icn.mp3",
        "time_created": "2022-07-23T13:30:17.000Z",
        "time_modified": "2022-07-23T13:30:17.000Z"
      },
      {
        "id": 3,
        "user_id": 1,
        "url":
"public/audios/voiceprint/3g1MNMh7KBnphFeCfPU8jYvT.mp3",
        "time_created": "2022-07-23T13:32:23.000Z",
        "time_modified": "2022-07-23T13:32:23.000Z"
      },
      {
        "id": 4,
        "user_id": 1,
        "url":
"public/audios/voiceprint/iT_6BbjXiwzjT3K8eNfdNkOa.mp3",
        "time_created": "2022-07-23T15:54:20.000Z",
        "time_modified": "2022-07-23T15:54:20.000Z"
      }
    ],
    "voiceprint_count": 4
  }
}
```

## (3) 录音 (/audio):

/get_audios	
用途	面向管理平台的录音信息批量获取
权限	manager root
入参	<pre>{   "page_index": 1 }</pre>
出参	<pre>{   "code": 200,   "message": "操作成功!",   "info": {     "batch": 8,     "total_pages": 2,     "audio_data": [       {         "id": 9,         "serial": "1658592080235612081",         "user_id": 1,         "url": "public/audios/recognition/K2qGApb402x2Rr5SlzMd0HBb.mp3",         "base64": "//uQZAAAAA...",         "result": "{}...",         "time_created": "2022-07-23T16:01:20.000Z",         "time_modified": "2022-07-23T16:01:20.000Z"       },       {         "id": 10,         "serial": "1658595963136852192",         "user_id": 1,         "url": "public/audios/recognition/Epd7QgLGmkeNGlRcXtP9N06l.mp3",         "base64": "//MoxAAL2A...",         "result": "{\\"RequestI...",         "time_created": "2022-07-23T17:06:03.000Z",         "time_modified": "2022-07-23T17:06:46.000Z"       }     ],     "audio_count": 10   } }</pre>

/recognition	
用途	录音语义识别



权限	user manager root
入参	<pre>{   "audio_id": 1 }</pre>
出参	<pre>{   "code": 200,   "message": "操作成功!",   "info": {     "RequestId": "0555dbeb-ea70-4a3a-bf8e-2f5852a6b009",     "Result": "英特尔被嵌入式开发大赛?",     "AudioDuration": 2520,     "WordSize": 5,     "WordList": [       {         "Word": "英特尔",         "StartTime": 0,         "EndTime": 540       },       {         "Word": "被",         "StartTime": 540,         "EndTime": 690       },       {         "Word": "嵌入式",         "StartTime": 690,         "EndTime": 1200       },       {         "Word": "开发",         "StartTime": 1200,         "EndTime": 1650       },       {         "Word": "大赛",         "StartTime": 1650,         "EndTime": 2100       }     ]   } }</pre>
/instruction	

用途	录音转控制指令
权限	user manager root
入参	<pre>{   "audio_id": 6 }</pre>
出参	<pre>{   "code": 200,   "message": "操作成功!",   "info": {     "audio_serial": "1658481929588298736",     "keyword": "关闭 LED",     "content": "{\"device\": \"LED\", \"brightness\": 0}"   } }</pre>

/command	
用途	录音转控制指令，并向硬件网关发送控制请求
权限	user manager root
入参	<pre>{   "serial": "1658595963136852192" }</pre>
出参	<pre>{   "code": 204,   "message": "硬件网关未连接!" }</pre>

#### (4) 指令 (/command):

/register	
用途	面向管理系统的指令注册
权限	manager root
入参	<pre>{   "keyword": "调整 LED 亮度",   "description": "该指令用于调整 LED 亮度",   "content": '{  \"device\": \"LED\",    \"brightness\": ? }'</pre>
出参	<pre>{   "code": 204,   "message": "指令关键词已被注册!" }</pre>

/get_commands
---------------

用途	面向管理系统的指令信息批量获取
权限	manager root
入参	<pre>{   "page_index": 0 }</pre>
出参	<pre>{   "code": 200,   "message": "操作成功!",   "info": {     "batch": 8,     "total_pages": 1,     "command_data": [       {         "id": 1,         "keyword": "调整 LED 亮度",         "content": "{\"device\": \"LED\", \"brightness\": ?}",         "description": "该指令用于调整 LED 亮度",         "is_valid": 1,         "time_created": "2022-07-21T06:37:31.000Z",         "time_modified": "2022-07-24T13:53:23.000Z"       },       {         "id": 2,         "keyword": "关闭 LED",         "content": "{\"device\": \"LED\", \"brightness\": 0}",         "description": "该指令用于关闭 LED",         "is_valid": 1,         "time_created": "2022-07-22T09:14:56.000Z",         "time_modified": "2022-07-24T13:53:23.000Z"       },       {         "id": 5,         "keyword": "测试舵机",         "content": "{\"device\": \"servo\", \"action\": \"test\"}",         "description": "该指令用于测试舵机摆动",         "is_valid": 1,         "time_created": "2022-07-23T07:34:52.000Z",         "time_modified": "2022-07-24T13:53:25.000Z"       }     ],     "command_count": 3   } }</pre>

/switch	
用途	面向用户终端应用的用户注册
权限	manager root
入参	<pre>{   "id": 3,   "is_valid": 0 }</pre>
出参	<pre>{   "code": 200,   "message": "操作成功!" }</pre>

#### (5) 上传 (/upload):

/audio_recognition	
用途	录音上传
权限	user manager root
入参	不大于 500KB 的 MP3 文件
出参	<pre>{   "code": 200,   "message": "操作成功!",   "info": {     "serial": "1658754454346414171",     "path":       "public/audios/recognition/jZvVFB5cHORDx1WSj0EinM0c.mp3",     "base64": "//MoxAAL2AbaXUAQAm4yU5..."   } }</pre>

/avatar	
用途	头像上传
权限	manager root
入参	不大于 500KB 的 png/jpg/jpeg 文件
出参	<pre>{   "code": 200,   "message": "操作成功!",   "info": {     "path": "public/images/avatar/df0VZmqAPo96xbRB2w7Cpa1N.png"   } }</pre>

/voiceprint	
用途	声纹上传
权限	user manager root
入参	不大于 500KB 的 MP3 文件
出参	<pre>{   "code": 200,   "message": "操作成功!",   "info": {     "path":     "public/audios/voiceprint/0298eETpVWH9cKTz6CzfMgUE.mp3"   } }</pre>

## 2.4.4 硬件网关

项目硬件网关包含基于 Node.js Express 的上位机与 Arduino UNO R3 开发板下位机。上位机作为网关，接收来自服务器的控制请求，通过 Firmata 协议，将控制指令发送至下位机，实现外设的控制。

### 2.4.4.1 Firmata、Arduino UNO R3 与 Arduino IDE

#### (1) Firmata:

Firmata 是一个 PC 与 MCU 通讯的一个常用协议，其设计旨在提供一套与任何主机 PC 软件包兼容的软硬件通讯协议。Firmata 协议起初面向 PC 与 Arduino 通讯的固件而设计，其目标是让开发者通过 PC 软件完全控制 Arduino。时至今日，Firmata 协议已得到若干语言的支持，这样可方便地将对协议的支持加入软件系统中。

#### (2) Arduino UNO R3:

Arduino Uno R3 是一款基于 ATmega328P 单片机的开发板。具有 14 个 Digital I/O 引脚，6 个 Analog In 引脚，16 MHz 石英晶振，USB 接口，电源接口。支持在线串行编程以及复位按键。用户只需将开发板与电脑通过 USB 接口连接，即可使用。

#### (3) Arduino IDE:

Arduino IDE 是 Arduino 团队提供的一款专门为 Arduino 开发板设计的编程软件。开发者可在 Arduino IDE、VSCode 等集成开发环境中，使用类 C 语言编写 Arduino 程序（扩展名为.ino）。开发完成后，将 Arduino 程序导入 Arduino IDE，选择对应的串口，即可将程序烧录至 Arduino 开发板。

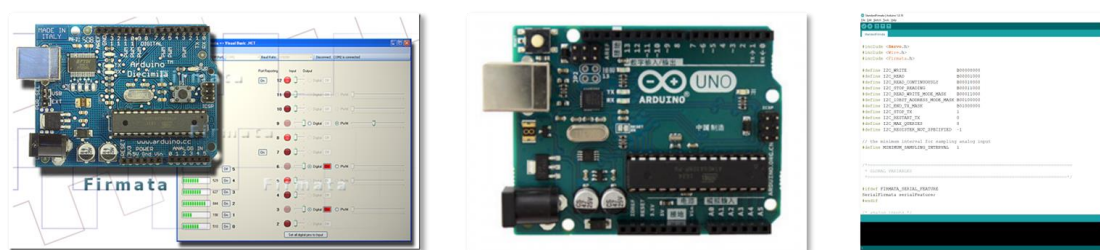


图 20 Firmata 协议（左）、Arduino UNO R3 开发板（中）、Arduino IDE（右）

### 2.4.4.2 Arduino UNO R3 嵌入式程序与系统搭建

由于硬件网关系统使用 Firmata 协议通信，故先将嵌入式程序 StandardFirmata.ino 烧入

Arduino 开发板，完成 Firmata 协议适配。

准备小型 LED 一个、9g 舵机一个、面包板一个（非必需）、公-公杜邦线若干，如下图所示（LED 正极接 Digital 11，负极接 GND；舵机正极接 5V，负极接 GND，信号线接 Digital 10），完成硬件系统的连接。

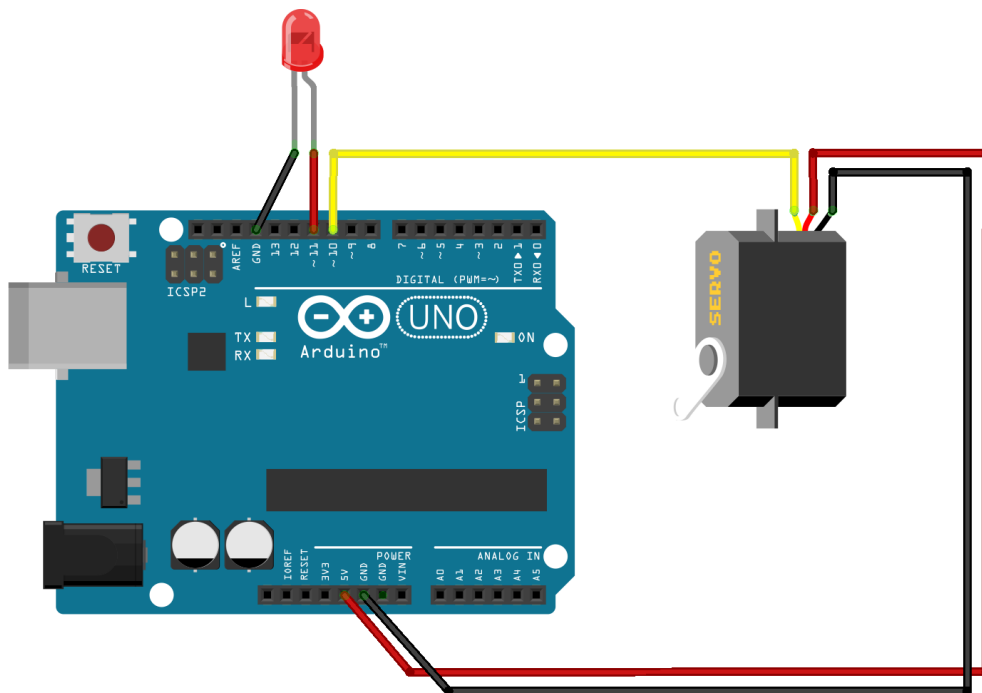


图 21 硬件系统接线示意图

使用串口数据线将已接好的硬件系统连接至竞赛平台 USB COM7，如图所示：

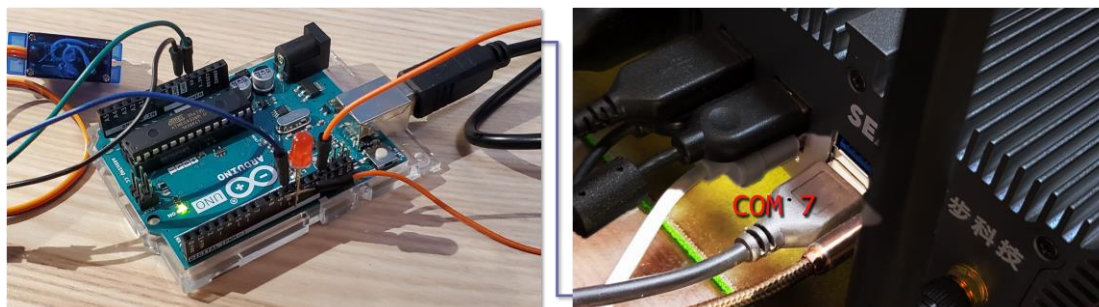


图 22 使用串口数据线连接硬件系统至竞赛平台 USB COM7

#### 2.4.4.3 硬件网关程序设计

硬件网关可视为一个具有硬件（Arduino 开发板）驱动功能的 Node.js Express WEB 服务器。WEB 服务器监听并接收控制请求，解析指令，并将其作用于硬件。

首先，引入服务器框架 Express 与 Firmata 协议库 johnny-five：

```
1. const express = require("express");
2. const five = require("johnny-five");
```

传入预先配置的端口名称（此处为 COM 7），初始化开发板实例：

```
1. const board = new five.Board({port: Config.hardwarePort});
```

在监听到硬件就绪（ready 事件）后，初始化外设 led、servo，创建服务器实例 app，定义控制器/hardware/execute，监听控制请求：

```
1. board.on("ready", () => {  
2.  
3.     const app = express();  
4.  
5.     const led = new five.Led(11);  
6.     const servo = new five.Servo(10);  
7.  
8.     app.post("/hardware/execute", JsonParser, (req, res) => {  
9.         ...  
10.    });  
11.  
12. });
```

若初始化成功，LED 闪烁两次：

```
1. // blink signals ready  
2. led.blink(500);  
3. setTimeout(() => {  
4.     led.stop().off();  
5.  
6.     // start command listening  
7.     app.listen(Config.port);  
8. }, 2000);
```

指令信息一般包含外设（device）与参数，示例如下：

```
1. {  
2.     "device": "LED",  
3.     "brightness": 0  
4. }
```

在/hardware/execute 控制器中，提取指令的 device 字段，判定外设，进入相应分支后，提取指令参数字段（如 LED 指令参数为 brightness），最终向硬件发送带有参数的控制指令：

```
1. let data = req.body;  
2. let commandContent = data.command_content;  
3. if (commandContent.device.toUpperCase() === "LED") {  
4.     let brightness = Math.abs(commandContent.brightness) % 256;  
5.     led.brightness(brightness);  
6.     console.log(`LED brightness set to ${brightness}`);  
7. }
```



```
8. if (commandContent.device.toUpperCase() === "SERVO") {  
9.     ...  
10. }
```

若指令执行成功，则向服务器返回成功响应，否则返回失败响应：

```
1. try {  
2.     ...  
3.     res.status(EnumHelper.HTTPStatus.OK).send(ResponseHelper.ok({message: MessageHelper.device_execute_successful}));  
4. } catch (err) {  
5.     console.log(err);  
6.     res.status(EnumHelper.HTTPStatus.ERROR).send(ResponseHelper.error({message: MessageHelper.device_execute_failed}));  
7. }
```

在断开连接（exit 事件）前，程序将注销对外设的控制：

```
1. board.on("exit", () => {  
2.     led.stop().off();  
3.     servo.stop();  
4. });
```

## 2.4.5 管理系统

项目管理系统基于 Vue 实现，面向管理员，提供前后端分离的 WEB 应用。该应用通过集成用户管理、声纹识别音频管理、语义识别音频管理、音频指令管理等模块，为管理员监视系统状态、操作数据提供便利。网站采用 Element UI 快速成型，简洁美观，功能全面。

### 2.4.5.1 用户登录

管理系统启动后，自动打开浏览器，若用户此前未保存登录令牌，则进入用户登录页面 (/login)，提示用户登录。

登录面板表单包含用户名输入框与密码输入框。登录时，在用户名输入框输入用户名、电话号码或电子邮箱，在密码输入框输入密码，点击“登录”按钮，则系统调用 </user/login> 接口进行登录。登入系统后，进入欢迎页 (/dashboard)。

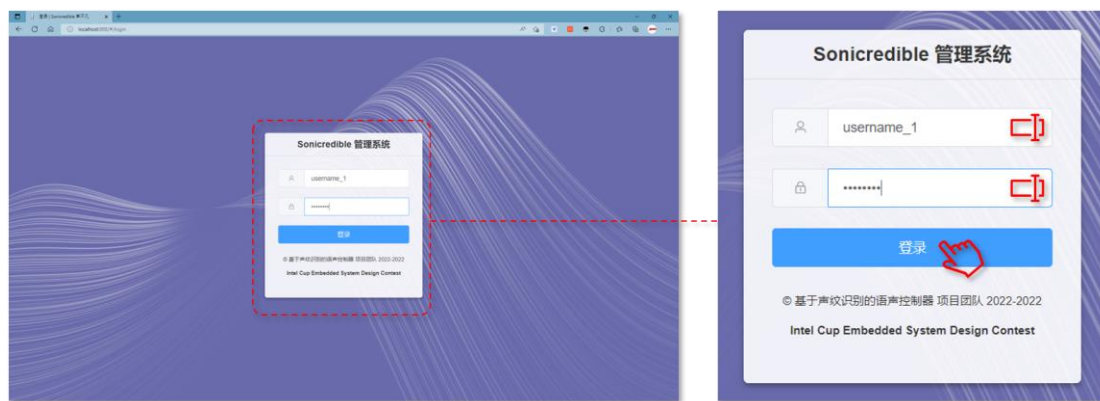


图 23 用户登录

若登录成功，管理系统将使用 `localStorage` 保存用户信息与 Token，即进入登录态。除非用户主动登出系统或 Token 过期，否则登录态将一直保持，即用户关闭浏览器（不清除缓存），重新进入管理系统时，无须再次登录。

### 2.4.5.2 主页面布局

用户登录成功后，系统自动跳转至主页面。主页面由 4 个组件构成：顶部栏 (Header)、侧边栏 (Sidebar)、选项卡 (Tab)、页面路由 (Router View)。

#### (1) 顶部栏 (Header):

顶部栏固定于页面顶部。

顶部栏最左侧图标按钮为侧边栏开关，点击可展开或折叠侧边栏。

该图标按钮右侧为项目名：**Soniccredible** 管理系统。

顶部栏右侧为用户头像、用户名。点击用户名，弹出选项卡，可选择“登出”进行登出操作。

## （2）侧边栏（Sidebar）:

侧边栏固定于页面左侧，可展开或折叠。

侧边栏自上而下，以页面树结构呈现管理系统当前包含的全部页面。点击任一页面，页面路由即跳转至对应页面。



图 24 主页面布局：顶部栏、侧边栏、选项卡与页面路由

## （3）选项卡（Tab）:

选项卡固定于顶部栏下方，侧边栏右侧。

与浏览器选项卡类似，选项卡按打开时间顺序，列出当前用户已打开的管理系统页面。每个打开的页面均以页面按钮形式呈现。点击页面按钮主体，可令页面路由跳转至响应页面；点击页面按钮关闭图标，可关闭对应页面、

选项卡右侧为标签选项按钮，点击可弹出选项卡，选择“关闭其他”，可关闭选项卡中当前选项卡之外的全部页面；选择“关闭所有”，可关闭选项卡中的全部页面，并令页面路由跳转至欢迎页（/dashboard）。

## （4）页面路由（Router View）:

页面路由位于页面右下方，是主页面布局的主体，可以上下滚动。

页面路由呈现选项卡当前选中的页面，在切换页面时，具有渐变效果。

### 2.4.5.3 用户管理

用户登录后，点击侧边栏“用户管理”菜单下的“用户管理”按钮，页面路由即跳转至用户管理页面（/user\_management）。

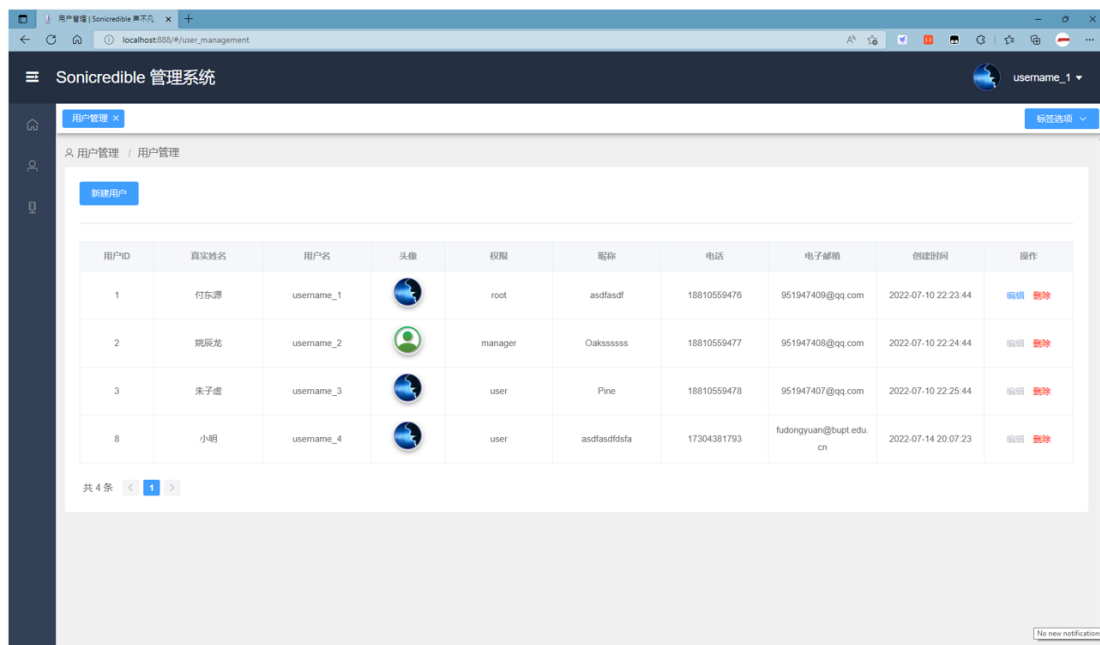


图 25 用户管理页面（/user\_management）

用户管理页面由“新建用户”按钮与用户列表构成。

若进行管理员用户注册，则点击“新建用户”按钮，弹出“新建用户”对话框。“新建用户”对话框表单包含 6 个输入框：用户名、密码、真实姓名、昵称、电话、电子邮箱。填写完毕后点击底部“确定”按钮，系统将验证表单各字段的有效性。若全部有效，则调用 [/user/full\\_register](#) 接口，完成用户注册。注册成功后，用户列表将刷新。



图 26 “新建用户”对话框

用户列表呈现当前系统中注册的用户信息。每个用户条目包含用户 ID、真实姓名、用户名、头像、权限（user、manager 或 root）、昵称、电话、电子邮箱、创建时间。

当前登录的用户可点击用户条目右侧“编辑”按钮，编辑本人用户信息。点击“编辑”按钮后，弹出“编辑用户”对话框。“编辑用户”对话框表单在“新建用户”对话框表单的基础上，



图 27 “编辑用户”对话框

增加了头像上传组件。该组件允许用户上传（[/upload/avatar](#)）不大于 500KB 的 png/jpg/jpeg 格式图片作为头像。完成表单填写后，点击底部“确定”按钮，则系统调用[/user/update](#)接口提交用户信息修改。此时，为确保数据一致性，系统将强制用户下线并重新登录，以获取修改后的用户信息。

用户列表底部为分页组件。每页显示 8 条数据，可通过点击数字前往对应页码。

#### 2.4.5.4 声纹识别音频管理

用户登录后，点击侧边栏“音频管理”菜单下的“声纹识别音频管理”按钮，页面路由即跳转至声纹识别音频管理页面（[/audio\\_print\\_management](#)）。

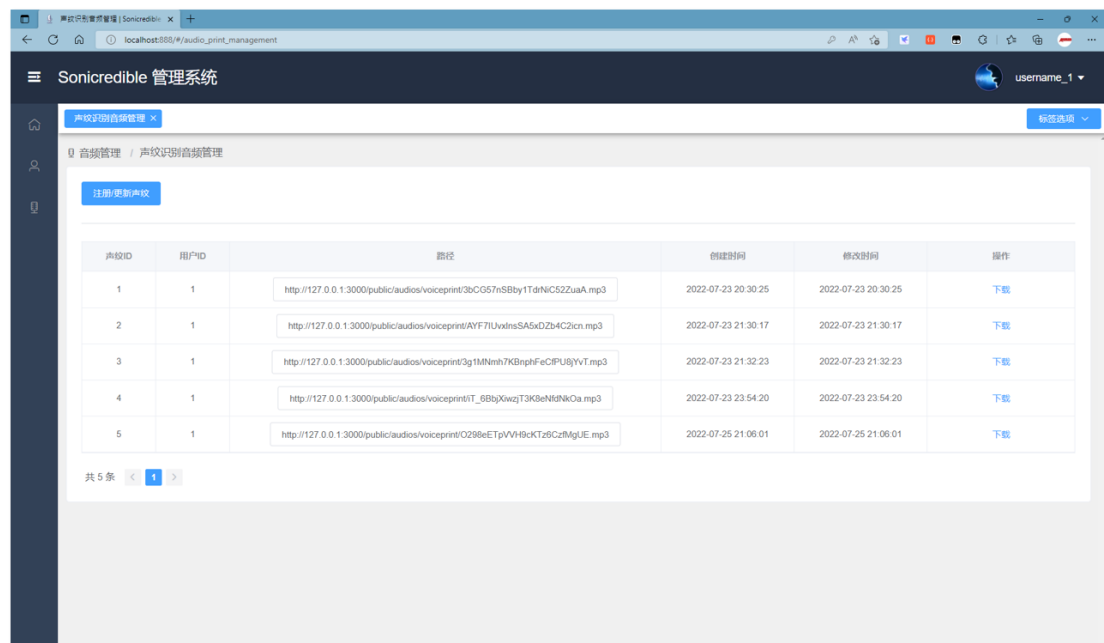


图 28 声纹识别音频管理页面（[/audio\\_print\\_management](#)）

声纹识别音频管理页面由“注册/更新声纹”按钮与声纹列表构成。

若进行注册/更新声纹操作，则点击“注册/更新声纹”按钮，弹出“注册/更新声纹”对话框。当前登录的用户可通过对话框上传组件上传（[/upload/voiceprint](#)）该账户对应的声纹。上传成功后，声纹列表将刷新。需要注意的是，与用户终端应用交互时，系统将以用户最新上传的声纹作为声纹识别的样本。

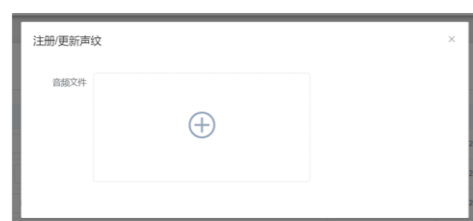


图 29 “注册/更新声纹”对话框

声纹列表呈现当前系统中注册的声纹信息。每个声纹条目包含声纹 ID、用户 ID、路径、创建时间、修改时间。点击声纹条目右侧“下载”按钮，可下载对应的声纹音频。

声纹列表底部为分页组件。每页显示 8 条数据，可通过点击数字前往对应页码。

#### 2.4.5.5 语义识别音频管理

用户登录后，点击侧边栏“音频管理”菜单下的“语义识别音频管理”按钮，页面路由即跳转至语义识别音频管理页面（/audio\_recognition\_management）。

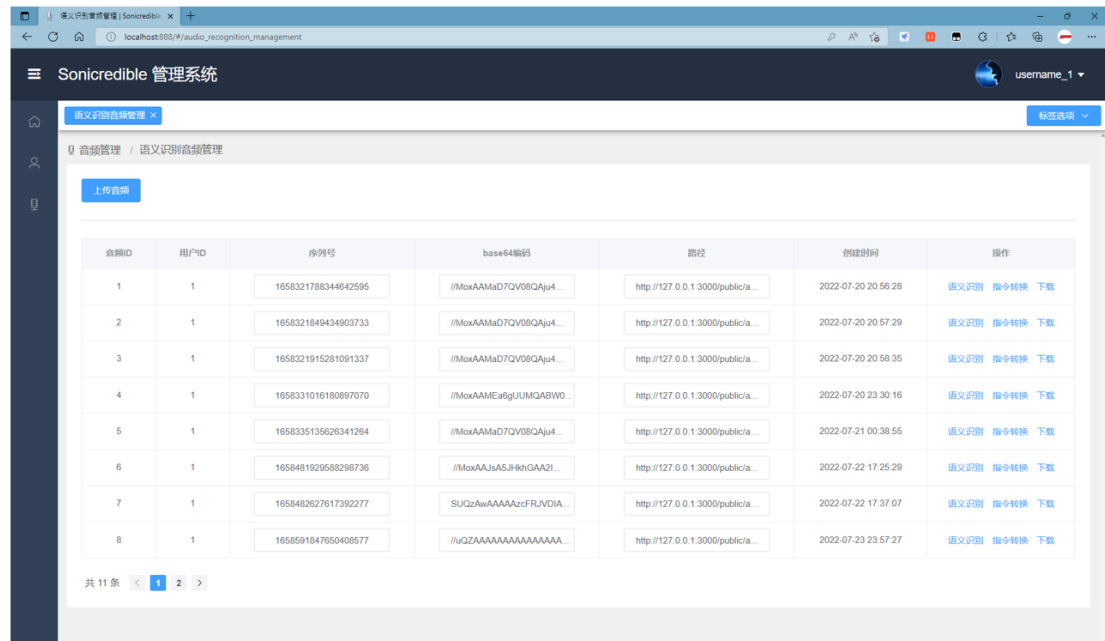


图 30 语义识别音频管理页面（/audio\_recognition\_management）

语义识别音频管理页面由“上传音频”按钮与音频列表构成。

若进行音频上传操作，则点击“上传音频”按钮，弹出“上传音频”对话框。当前登录的用户可通过“上传音频”对话框提供的上传组件上传（/upload/audio\_recognition）一段不大于 500KB 的 mp3 音频（对应于用户终端应用中的录音）。上传成功后，音频列表将刷新。

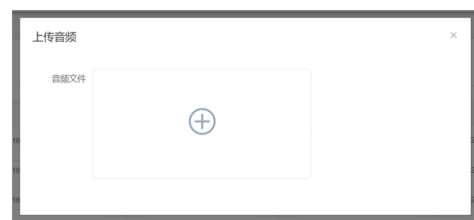


图 31 “上传音频”对话框

音频列表呈现当前系统中的音频信息。每个音频条目包含音频 ID、用户 ID、序列号、base64 编码、路径、创建时间。

点击音频条目右侧“语义识别”按钮，弹出“语义识别”对话框。若该条音频尚未进行语义识别，



图 32 “语义识别”对话框

可点击“识别”按钮可对当前音频进行语义识别（/audio/recognition）；对于已进行语义识别

的音频，则直接显示语义识别结果与分词结果。

点击音频条目右侧“指令转换”按钮，弹出“指令转换”对话框。对于已进行语义识别的音频，系统将尝试指令转换（</audio/instruction>），并呈现结果；对于尚未进行语义识别的音频，系统将提示用户先进行语义识别，再进行指令转换。指令转换成功时，点击“测试”按钮，可令项目服务器向硬件网关发送控制请求（</audio/command>）。



图 33 “指令转换”对话框

点击音频条目右侧“下载”按钮，可下载对应的音频。

音频列表底部为分页组件。每页显示 8 条数据，可通过点击数字前往对应页码。

#### 2.4.5.6 音频指令管理

用户登录后，点击侧边栏“音频管理”菜单下的“音频指令管理”按钮，页面路由即跳转至音频指令管理页面（[/audio\\_command\\_management](/audio_command_management)）。

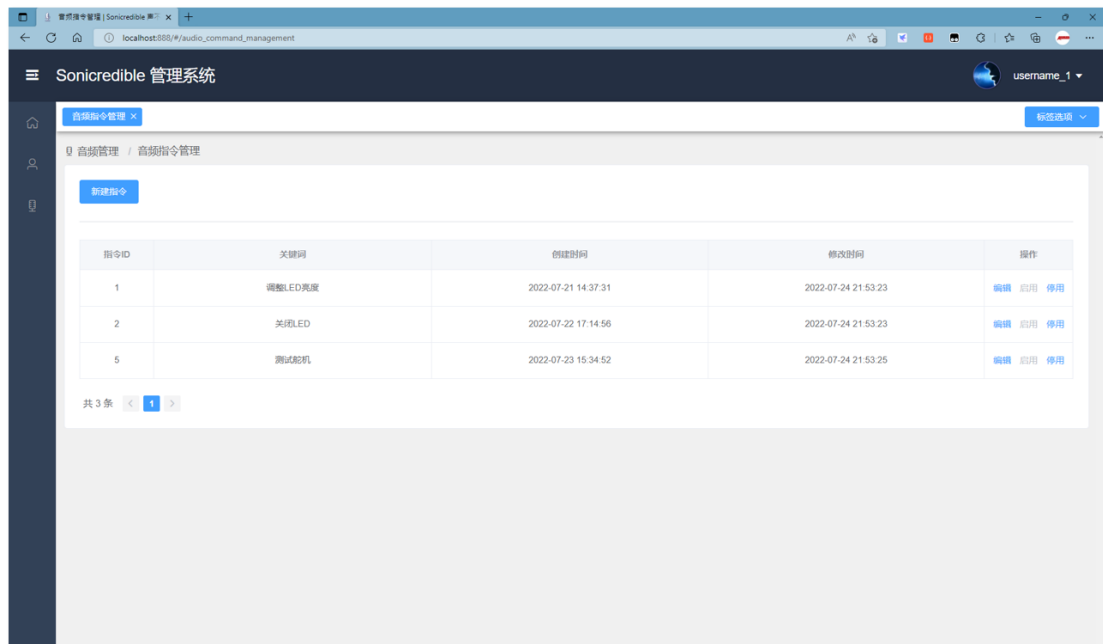


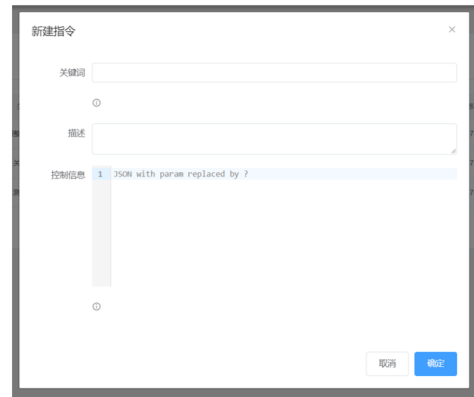
图 34 音频指令管理页面（[/audio\\_command\\_management](/audio_command_management)）

音频指令管理页面由“新建指令”按钮与指令列表构成。

若进行新建指令操作，则点击“新建指令”按钮，弹出“新建指令”对话框。“新建指令”对话框表单包含关键词、描述、控制信息。按 2.4.3.5 节“指令模板”部分提供的规范，填写对应字段，点击“确定”，系统即调用</command/register>接口。新建指令成功后，指令列表将刷新。

指令列表呈现当前系统中的指令信息。每个指令条目包含指令 ID、关键词、创建时间、修改时间。

点击指令条目右侧“启用”或“停用”按钮，可控制该条指令是否可用（[/command/switch](#)）。点击指令条目右侧“编辑”按钮，可编辑指令信息。



新建指令对话框包含以下元素：

- 关键词：输入框
- 描述：输入框
- 控制信息：列表，显示 1 条记录，内容为 "JSON with param replaced by ?"
- 底部按钮：取消、确定

图 35 “新建指令”对话框



## 3 项目系统测试

项目系统测试中，用户终端应用、服务器、硬件网关在竞赛平台（GNU-V40）上运行，硬件平台 Arduino UNO R3 须连接至竞赛平台的 USB COM 7 端口。为方便调试，将竞赛平台通过 HDMI 线缆连接 Lenovo ThinkVision 1080P 60Hz 显示器。



图 36 项目系统测试

### 3.1 单元测试

#### 3.1.1 用户登录

##### (1) 测试说明：

用于模拟在用户终端应用与管理系统中的登录操作。

##### (2) 测试执行：

编写用户登录单元测试 user\_login.py:

```
1. from post import *
2. if __name__ == '__main__':
3.     print('欢迎使用声不凡语音控制系统')
4.     username = str(input("用户名: "))
5.     password = str(input("密码: "))
6.     #登陆
7.     response_code,response=login(username=username,password=password)
8.
9.     print(response_code)
10.    print(response)
```

执行测试：

```
1. >>> 欢迎使用声不凡语音控制系统
2. >>> 用户名:
3. <<< username_1
4. >>> 密码:
5. >>> Sam_2019
6. >>> 200
```

```

7. >>> {'code': 200, 'message': '操作成功!
      ', 'info': {'userinfo': {'id': 1, 'realname': '付东源
      ', 'nickname': 'asdfasdf', 'username': 'username_1', 'password': 'a43cf88895
      1388c4', 'avatar': 'public/images/avatar/default.png', 'tel': '18810559476',
      'email': '951947409@qq.com', 'authority': 'root', 'is_available': 1, 'time_
      created': '2022-07-10T14:23:44.000Z', 'time_modified': '2022-07-
      20T16:26:53.000Z'}, 'token': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX
      2lkIjoxLCJ0aW1lc3RhbXAiOiJlbnR5cCI6IkpXVCJ9.eyJ1c2VyX
      HsJ8JSiUqEvsNwcaUx5Nqj4SCPT8nvWdbk'}}

```

### 3.1.2 声纹注册

#### (1) 测试说明:

用于模拟在用户终端应用与管理系统中的声纹注册操作。

#### (2) 测试执行:

编写声纹注册单元测试 voiceprint\_register.py:

```

1. from post import *
2. from utils.record import RecordAudio
3.
4. if __name__ == '__main__':
5.     print('欢迎使用声不凡语音控制系统')
6.     username = str(input("用户名: "))
7.     password = str(input("密码: "))
8.     #登陆
9.     response_code,response=login(username=username,password=password)
10.
11.     #录音
12.     record_audio = RecordAudio()
13.     print('请说出“芝麻开门”')
14.     audio_path1 = record_audio.record()
15.
16.     #上传声纹信息
17.     response_code,response=voiceprint(file=audio_path1)
18.
19.     print(response_code)
20.     print(response)

```

执行测试:

```

1. >>> 欢迎使用声不凡语音控制系统
2. >>> 用户名:
3. <<< username_1
4. >>> 密码:
5. <<< Sam_2019

```

```

6. >>> 请说出“芝麻开门”
7. >>> 按下回车键开机录音，录音 5 秒：
8. >>> 开始录音.....
9. <<< · 动作：用户大声说“芝麻开门”，并被录下音频
10. >>> 录音已结束！
11. >>> 200
12. >>> {'code': 200, 'message': '操作成功！
      ', 'info': {'path': 'public/audios/voiceprint/IngSkYWnN5z9lupC3FoqXn5k.mp3'}}
  
```

### 3.1.3 声纹识别

#### (1) 测试说明：

用于模拟在用户终端应用进行的用户声纹与录音声纹比对（声纹识别）操作。

#### (2) 测试执行：

编写声纹识别单元测试 voiceprint\_register.py:

```

1. from post import *
2. from utils.record import RecordAudio
3.
4. if __name__ == '__main__':
5.     print('欢迎使用声不凡语音控制系统')
6.     username = str(input("用户名: "))
7.     password = str(input("密码: "))
8.     #登陆
9.     response_code,response=login(username=username,password=password)
10.
11.     #录音
12.     record_audio = RecordAudio()
13.     print('请说出“芝麻开门”')
14.     audio_path1 = record_audio.record()
15.
16.     #上传声纹信息
17.     response_code,response=voiceprint(file=audio_path1)
18.
19.     print(response_code)
20.     print(response)
  
```

执行测试：

```

1. >>> 欢迎使用声不凡语音控制系统
2. >>> 用户名：
3. <<< username_1
4. >>> 密码：
5. <<< Sam_2019
  
```

```
6. >>> 请说出“芝麻开门”
7. >>> 按下回车键开机录音，录音 5 秒：
8. <<< 开始录音.....
9. <<< · 动作：用户大声说“芝麻开门”，并被录下音频
10. >>> 录音已结束！
11. >>> ----- Configuration Arguments -----
12. >>> audio_path1: audio/Server_voice.wav
13. >>> audio_path2: audio/Local_voice.wav
14. >>> input_shape: (1, 257, 257)
15. >>> model_path: models/resnet34.pth
16. >>> threshold: 0.5
17. >>> -----
18. >>> audio/Server_voice.wav 和 audio/Local_voice.wav 为同一个人，相似度为：
    0.533771
```

### 3.1.4 语音控制

#### (1) 测试说明：

用于模拟在用户终端应用进行的语音控制硬件操作。

#### (2) 测试执行：

编写声纹识别单元测试 voice\_control.py:

```
1. from post import *
2. from utils.record import RecordAudio
3. from infer_contrast import*
4.
5. if __name__ == '__main__':
6.     print('欢迎使用声不凡语音控制系统')
7.     username = str(input("用户名: "))
8.     password = str(input("密码: "))
9.     #登陆
10.    response_code,response=login(username=username,password=password)
11.
12.    #录音
13.    record_audio = RecordAudio()
14.    print('请说出指令')
15.    audio_path1 = record_audio.record()
16.
17.    #上传音频
18.    response_code,response=audio_recognition(audio_path1)
19.    print(response['info']['serial'])
20.
21.    #语义识别转指令
```

```
22. response_code,response=command(response['info']['serial'])
23.
24. print(response)
```

执行测试：

```
1. >>> 欢迎使用声不凡语音控制系统
2. >>> 用户名:
3. <<< username_1
4. >>> 密码:
5. <<< Sam_2019
6. >>> 请说出指令
7. >>> 按下回车键开机录音，录音 5 秒:
8. >>> 开始录音.....
9. <<< . 动作：用户大声说“请关闭 LED”，并被录下音频
10. >>> 录音已结束!
11. >>> 1658813493456949276
12. >>> {'code': 200, 'message': '操作成功!', 'info': {'keyword': '关闭
LED', 'content': '{"device": "LED", "brightness": 0}', 'hardware': {'code':
200, 'message': '硬件端：指令执行成功!'}}}
```

观察到 Arduino 已将 LED 关闭。



图 37 测试前（左）、测试后（右，硬件网关收到控制指令，控制硬件将 LED 关闭）

## 3.2 集成测试

### (1) 测试说明:

用于模拟在用户终端应用进行的声纹识别语音控制操作。

### (2) 测试执行:

参照项目[核心功能时序图](#)，编写集成测试 integrated\_test.py:

```
1. from post import *
2. from utils.record import RecordAudio
3. from infer_contrast import *
4.
5. if __name__ == '__main__':
6.     print('欢迎使用声不凡语音控制系统')
7.     username = str(input("用户名: "))
8.     password = str(input("密码: "))
9.     # 登陆
10.    response_code, response = login(username=username, password=password)
11.
12.    # 下载声纹信息
13.    response_code, response, audio_path2 = get_voiceprint()
14.
15.    # 声纹认证
16.    record_audio = RecordAudio()
17.    print('请说出“芝麻开门”')
18.    audio_path3 = record_audio.record()
19.    result = contrast(audio_path2, audio_path3)
20.
21.    if result:
22.        # 声纹认证通过
23.        # 录音
24.        record_audio = RecordAudio()
25.        print('请说出指令')
26.        audio_path1 = record_audio.record()
27.
28.        # 上传音频
29.        response_code, response = audio_recognition(audio_path1)
30.        print(response['info']['serial'])
31.
32.        # 语义识别转指令
33.        response_code, response = command(response['info']['serial'])
34.        print(response)
35.    else:
36.        # 声纹认证不通过
```

```
37. print('声纹认证不通过, 不能控制硬件')
```

执行测试（声纹认证通过）:

```
1. >>> 欢迎使用声不凡语音控制系统
2. >>> 用户名:
3. <<< username_1
4. >>> 密码:
5. <<< Sam_2019
6. >>> 请说出“芝麻开门”
7. >>> 按下回车键开机录音, 录音 5 秒:
8. >>> 开始录音.....
9. <<< . 动作: 用户大声说“芝麻开门”, 并被录下音频
10. >>> 录音已结束!
11. >>> ----- Configuration Arguments -----
12. >>> audio_path1: audio/Server_voice.wav
13. >>> audio_path2: audio/Local_voice.wav
14. >>> input_shape: (1, 257, 257)
15. >>> model_path: models/resnet34.pth
16. >>> threshold: 0.5
17. >>> -----
18. >>> audio/Server_voice.wav 和 audio/Local_voice.wav 为同一个人, 相似度为:
    0.591201
19. >>> 请说出指令
20. >>> 按下回车键开机录音, 录音 5 秒:
21. >>> 开始录音.....
22. <<< . 动作: 用户大声说“请关闭 LED”, 并被录下音频
23. >>> 录音已结束!
24. >>> 1658814690237464270
25. >>> {'code': 200, 'message': '操作成功!', 'info': {'keyword': '关闭
    LED', 'content': '{"device": "LED", "brightness": 0}', 'hardware': {'code':
    200, 'message': '硬件端: 指令执行成功!'}}
```

观察到 Arduino 已将 LED 关闭。

执行测试（声纹认证不通过）:

```
1. >>> 欢迎使用声不凡语音控制系统
2. >>> 用户名:
3. <<< username_1
4. >>> 密码:
5. <<< Sam_2019
6. >>> 请说出“芝麻开门”
7. >>> 按下回车键开机录音, 录音 5 秒:
8. >>> 开始录音.....
9. <<< . 动作: 另一用户大声说“芝麻开门”, 并被录下音频
10. >>> 录音已结束!
11. >>> ----- Configuration Arguments -----
```

```
12. >>> audio_path1: audio/Server_voice.wav
13. >>> audio_path2: audio/Local_voice.wav
14. >>> input_shape: (1, 257, 257)
15. >>> model_path: models/resnet34.pth
16. >>> threshold: 0.5
17. >>> -----
18. >>> audio/Server_voice.wav 和 audio/Local_voice.wav 不是同一个人，相似度为：
    0.355747
19. >>> 声纹认证不通过，不能控制硬件
```



## 4 项目总结

### 4.1 成员分工

**(1) 付东源:**

负责项目管理与服务端开发，具体工作为：架构设计与技术选型、项目线上资源部署、后端开发、WEB 端开发、辅助嵌入式开发、语音识别系统集成测试。

**(2) 姚辰龙:**

负责嵌入式程序开发，具体工作为嵌入式程序及其 GUI 设计、针对嵌入式平台的算法性能优化、嵌入式平台算法测试、嵌入式平台系统测试、辅助后端开发与 WEB 端开发。

**(3) 朱子虚:**

负责算法设计、实现与测试，具体工作为声纹识别优化算法的设计、实现与测试、语义识别算法的设计与实现、嵌入式平台外设接入与调试、辅助后端开发与WEB端开发。

## 4.2 成员心得

### (1) 付东源:

感谢英特尔公司与学校为参赛团队提供的大力支持,感谢指导教师的悉心指导,感谢队友们为 Sonicredible 项目贡献的智慧与付出的汗水。本人通过在项目中进行后端开发,再次深刻理解项目管理理论与软件工程知识,对保障项目高效高质量实现起到的重要作用,并进一步提升了面向对象程序设计能力,加深了对 MVC 模式的理解。

项目开发进程中,项目成员大都进行线上协作,并定期召开项目进度会议,确保按时完成计划任务。7 月 16 日,本人在参赛过程中突发意外,中断开发进程之际,姚、朱两位队友主动担负起项目开发职责,着实令人感动。与这两位队友合作,是本人毕生之荣幸。

### (2) 姚辰龙:

首先,能够入围并参与此次比赛我感到十分荣幸。我们的项目是一项关于声纹识别的语音控制器,我在项目中负责了 GUI 设计以及部分的文案写作。一转眼,紧张的比赛就已经临近结束,感觉确实任务十分的紧张,但是也给我带来了许多宝贵的经验和良好的学习的机会。这次竞赛让我学习到了基于 python 进行 GUI 设计的技术,了解到了关于项目报告书写的许多规范。同时,由于赛程较长,组长为了方面管理也进行了不少的会议和规范设置,这让我受益良多,学习到了项目管理的许多实践性的经验。

最后感谢队长和队友在整个竞赛过程中的帮助和支持,在疫情严峻,无法返校进行线下面对面合作的情况下,仍然通过会议和多种方式让大家团结在一起,共同完成这次比赛。

### (3) 朱子虚:

很感谢本次英特尔杯为我们提供的平台与参赛机会,一方面让我领略到了现阶段嵌入式硬件平台的搭载模块与算力水平,另一方面也让我对于嵌入式开发从全栈角度有了全新的认识。对于一个嵌入式项目而言,清晰的规划、合理的分工以及及时的交流是支撑整个小组砥砺前行助推器。能做到以上三点,那么无论是项目设计、代码实现还是文档撰写环节,都将按照既定安排有条不紊的进行。我在本次项目负责部分的实现过程中也遇到了不少问题,也有过退缩和止步不前,但队友们从没有放弃过,为我提供了莫大的帮助,帮着我迈过一道道坎坷,最终走完全程,不负初心。回顾整个参赛过程,实在是我人生中不可多得的一次经历。

## 附录 参考文献与引用

- [1] 中国人工智能产业发展联盟. 中国声纹识别产业发展白皮书 [R/OL], 2019
- [2] 郑方, 李蓝天, 张慧等. 声纹识别技术及其应用现状[J]. 信息安全研究
- [3] Yu Y Q, Fan L, Li W J. Ensemble Additive Margin Softmax for Speaker Verification[C]// ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2019
- [4] vue-admin-template, GitHub, 2021

## 附录 项目程序清单

项目代码已托管至 GitHub，请通过以下链接访问：

<https://github.com/TheWayForward/Soniccredible>