华南理工大学

South China University of Technology

本科毕业设计（论文）

基于 WEB 的演奏机器人中控软件

学　　院	软件学院
专　　业	软件工程
学生姓名	金子
学生学号	201830661069
指导教师	朱金辉
提交日期	2022 年 5 月 21 日

摘　要

进入 21 世纪以来，人们对机器人的使用逐渐增多，其中涌现出各式各样的机器人类型，其中不乏能歌善舞的音乐机器人。音乐机器人能够模仿人类的敲击乐器的动作或者用自己的方式进行声音的展示，能够供人类所欣赏，并且其动作的展示也具有教育意义。但传统的基于单机的音乐机器人控制方式存在许多例如控制距离短，音乐分析结果单一，难以同时控制多个音乐机器人等局限性，用户难以将一首歌曲各个音轨上的音乐信息同时发送到不同的机器人，且对于音乐解析的结果不能很好地根据用户需求进行绑定动作与发送。

本文旨在为音乐机器人开发一款中控软件，提供良好的人机交互界面，从而让使用者简易地控制自己的音乐机器人，完成机器人初始化，音频解析，将动作和机器人绑定以及将数据传输到机器人进而产生音乐动作。本文使用 Electron 和 Vue 开发客户端，通过结合 Node.js 调用本地的 python 代码，分析用户上传的 MIDI 格式的音乐文件按照绑定逻辑得出相应的指令结果，并通过 MQTT 协议特有的发布/订阅模式将音乐信息按照时间发送给由用户指定的已经与客户端绑定的机器人。同时机器人通过在心跳主题发布消息达到维持在线状态的功能。经过测试，验证了软件的可用性，提高了用户对音乐机器人的管理与控制的便利性，同时做到了发送音乐信息的低延迟。

关键词：音乐机器人控制中心；Electron；MQTT；MIDI；Vue

Abstract

Since the beginning of the 21st century, people's use of robots has gradually increased, and various types of robots have emerged, including musical robots that can sing and dance. The musical robot can imitate the movements of human percussion instruments or display sounds in its own way, which can be appreciated by human beings, and the display of its movements is also of educational significance. However, the traditional single-machine-based music robot control methods have many limitations, such as short control distance, single music analysis results, and difficulty in controlling multiple music robots at the same time. It is difficult for users to send the music information on each track of a song to different robots at the same time, and the results of music analysis cannot be bound and sent according to user needs.

This paper aims to develop a central control software for the music robot, which provides a good human-computer interaction interface, so that users can easily control their own music robot, complete the robot initialization, audio analysis, bind the action to the robot, and transmit the data. to the robot to generate musical action. This article uses Electron and Vue to develop the client, and calls the local python code in combination with Node.js, analyzes the music files in MIDI format uploaded by the user, and obtains the corresponding instruction results according to the binding logic, and uses the unique publish/subscribe mode of the MQTT protocol. Send music information to the robot specified by the user and bound to the client according to the time. At the same time, the robot maintains the online status by publishing messages on the heartbeat topic. After testing, the usability of the software is verified, the convenience of the user's management and control of the music robot is improved, and the low latency of sending music information is achieved.

Keywords: Music Robot Control Center; Electron; MQTT; MIDI; Vue

目　录

第一章 绪论

1.1 研究背景及意义

机器人作为 20 世纪人类最伟大的发明之一，在二十多年的发展中发生的变化十分迅速。近年来，机器人作为高科技领域具有代表性的战略目标。机器人技术的出现和发展在从根本上改变了传统的工业生产方式的同时，还对人类社会产生了深远的影响。而现在人们正在走向信息时代，计算机化已经成为当前时代的主要话题。从计算机化到知识化再到智能化，已成为现阶段机器人发展的必然趋势[1]。娱乐机器人作为在机器人研究领域中的一个全新发展方向，其中音乐演奏机器人[2]以其更好的展示效果和交互效果赢得了研究人员的青睐。音乐演奏机器人通过控制装置控制所连接的外设机械结构，达到模拟人类演奏乐器的效果。

音乐演奏机器人是通过数字或机械的程序输入而实现演奏音乐的仿真机器。按智能程度分类可以大致分为自动机类音乐机器人和智能类音乐机器人两类[3]。

音乐演奏机器人可以通过各种各样演奏的方式，通过自定义输入音乐可以模拟复现出人类演奏的动作以及乐器对应的演奏方式，从而让人观察并欣赏以达到教育意义。

然而，音乐演奏机器人需要通过指定音乐源来通过自己的方式进行演奏，那么就必须面对音乐传输与解析的问题。演奏机器人需要接收到从用户传过来的音乐并进行解析得出需要做出的动作。而音乐的传输是一个数据传输的过程，就需要选择恰当的方案把音乐内容或者音乐解析的结果指令发送至机器人。综上所述，我们希望实现一款控制系统能与演奏机器人进行连接，并且能够解析音乐和发送演奏指令至机器人。

1.2 国内外研究现状

1.2.1 音乐演奏机器人

根据乐器的种类，演奏机器人主要有管乐(笛子、萨克斯等)、弦乐(吉他[4]、小提琴[5]等)、键盘打击乐(钢琴[6]、架子鼓[7]等)3 种类型。21 世纪以来，随着机器人学相关领域的技术进步与创新成果的结合，为音乐机器人的研究与开发带来了突破性的进展，这些关键技术包括：对人类运动控制的精确理解、表情化音乐演奏特质的提取、多通道的音乐交互方式的实现等[8]。有了这些技术，音乐机器人在音乐分析和创作，运动控制和协同合作即兴表演上的艺术才能大幅提升。

1.2.2 机器人控制系统

为了集中控制机器人，国内外给出了很多机器人控制系统的方案。

如图 1-1，传统的有线演奏机器人，需要将音乐内容通过 USB 线的方式传输数据与指令至机器人。有线的方式使得机器人的灵活性降低，并且不能脱离数据传输线的控制，给用户带来麻烦。

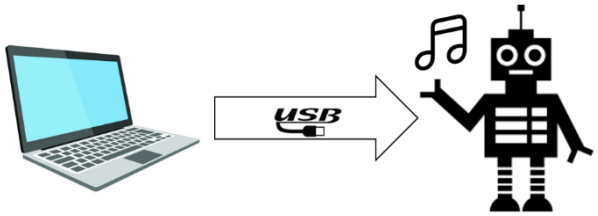

图 1-1 传统有线演奏机器人架构

市场中也存在不使用数据线而采用了遥传控制技术的演奏机器人，其操作如图 1-2 所示。其克服了有线控制的方式，变得更加灵活，但存在的缺陷是机器人演奏的音乐往往是提前预设而不能通过用户输入来进行灵活的变更，因此只能演奏固定的一些音乐。

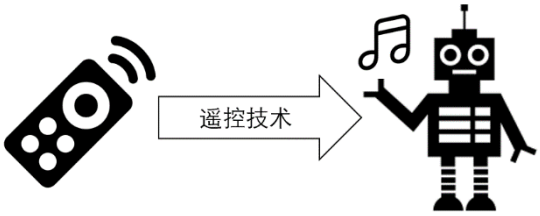

图 1-2 通过遥控技术控制音乐演奏机器人

而如图 1-3，通过局域网与演奏机器人相连解决了数据传输的问题，但由于局域网的局限性，控制中心只能与在同一个网络下的机器人相连，故极大地限制了控制距离。此外，这种方式也存在控制不便捷，控制中心设备要求高等不足。

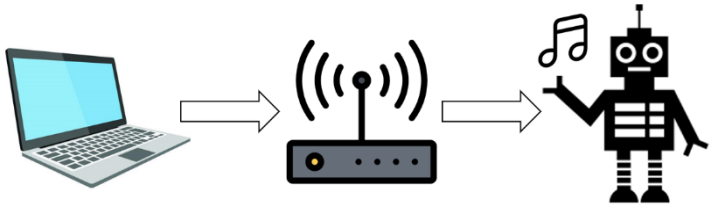

图 1-3 通过局域网与音乐演奏机器人连接

近年来随着互联网的蓬勃发展，出现了许多借助云服务器开发的控制中心，例如云

服务器环境下基于 Android 移动平台的机器人控制系统[9]等。基于 WEB 的移动机器人系统[10]在不同环境下有着不同实现，随着桌面端开发的发展，我们可以将控制中心与演奏机器人接入互联网，并且能够借助桌面程序开发技术开发出具有跨平台功能与良好的人机交互页面的控制中心，并且实现音乐解析等功能。其结构如图 1-4。

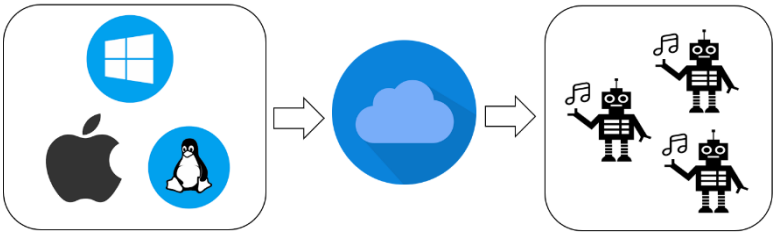

图 1-4 基于 WEB 的音乐演奏机器人中控系统

基于 WEB 的演奏机器人中控系统旨在将演奏机器人与互联网连接起来，人们可以在 Windows，macOS 或 Linux 下通过中控软件访问机器人，实现远程控制演奏机器人。以互联网为构架，不仅降低了遥操作系统的成本，而且使机器人能够在网络中被人们共享与操作。

1.2.3 物联网通信协议

随着物联网产业的飞速发展，物联网设备深入人们生活和生产的方方面面。因而，物联网相关的传输协议的安全性，也越来越受到国家政府机关及相关企业的重视，目前物联网行业主流的传输协议包括：蓝牙、ZigBee、Z-Wave、LoRaWAN 以及 MQTT 协议[11]。其中，在设备计算能力有限，环境复杂以及带宽传输速率低的时候，MQTT 协议仍能保持正常连接，并且为了保证物联网各接入设备之间传输的数据真实、完整、可靠，在网络层、传输层、应用层三个层面提供了多重安全设计。

目前有很多系统使用 MQTT 协议代替 HTTP 协议作为数据传输的核心。这是因为 MQTT 协议每小时运行消耗的电量比 HTTP 协议少，而 MQTT 在一小时内发送的消息是 HTTP 协议的十倍。即 MQTT 协议可以降低功耗，也比 HTTP 更快[12]。

1.3 本文的主要研究内容

1.3.1 中控软件的前端开发

用户通过中控软件可以进行一系列例如绑定机器人，选择音频文件并本地解析，选择机器人进行演奏等等的功能。这些功能需要一个应用作承载，那么就需要对桌面端程

序开发的方案进行选取和分析，分析软件的需求，研究内容结构以及设计出良好的人机交互界面，才能给用户良好的使用体验，提供清晰明确的指引和完善的功能。

1.3.2 音频解析模块的开发

当用户选取音乐后，可以通过中控软件的音频解析模块得到时间和动作的序列。那么需要研究音频解析的方法和技巧，选择贴合中控软件并且解析效果好的解析方式，提供从文件解析得到音符时间序列的功能。

1.3.3 机器人绑定模块的开发

想要控制具体指定的机器人，就需要通过中控软件连接机器人。研究绑定机器人的方案，除了需要连接机器人以外，还要能达到粗粒度可以将音乐序列绑定某个机器人，细粒度可以将音乐序列绑定某个机器人的具体动作。

1.3.4 网络通信模块的开发

与传统遥传操作技术不同，基于 WEB 的远程机器人控制因为受带宽和网络负载变化的影响，网络的延迟具有不确定性。中控软件需要选择合适的网络通信协议将解析结果和动作指令发送给对应的机器人，并且保证网络通信的稳定与低延迟。

1.4 论文结构

本文分为六章。其中第一章简述了演奏机器人中控系统的研究背景和意义。第二章节从桌面程序开发技术，音乐解析方式，通信协议技术和机器人开发板四个方面详细地讲述了音乐机器人中控系统的基础知识。第三章展示了演奏机器人中控系统的需求分析及设计思路等。第四章详细地分析了各个模块的具体实现结果。第五章为软件系统的部署以及测试。最后是对此次设计工作的总结与未来展望。

第二章 相关技术介绍

2.1 桌面程序开发相关技术介绍

2.1.1 Electron 框架介绍

Electron[13]是一个由 OpenJS 基金会和一个活跃的贡献者社区管理的开源项目，同时也是一个基于 Chromium 和 Node.js，让开发者可以使用 HTML，CSS 和 JavaScript 等 Web 技术创建原生程序的框架。通过将 Chromium 和 Node.js 嵌入到二进制文件中，Electron 允许开发者维护一个 JavaScript 代码库并创建可在 Windows、macOS 和 Linux 上运行的跨平台应用程序。目前主流的代码编辑器 Visual Studio Code 就是使用 Electron 开发的。

因为继承了来自 Chromium 的多进程架构，此框架在架构上非常相似于一个现代的网页浏览器。但在 web 页面里，调用系统底层的 API 是不被允许的，因为很容易导致资源泄漏。但由于 Node.js 能够运行 JavaScript 和具有本地资源访问的接口，Electron 拥有了访问文件系统和网络权限的能力。

Electron 有两种进程：主进程和渲染进程。一个 Electron 应用只能有一个主进程，但可以有多个渲染进程。主进程通过创建 BrowserWindow 示例来创建页面，每一个 BrowserWindows 的实例都会由一个独立的渲染进程来渲染页面。其结构如图 2-1 所示。

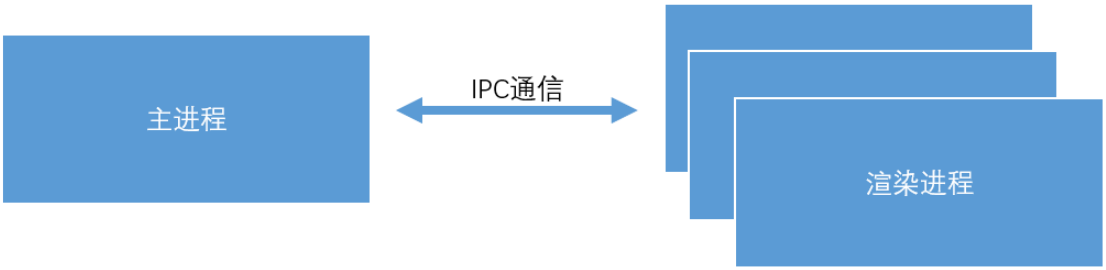

图 2-1 Electron 中主进程与渲染进程通信图

2.1.2 Vue 框架介绍

Vue[14]是一套用于构建交互式前端用户界面的渐进式框架，比起其他大型框架，Vue 是自底向上逐层应用的。可以用 Vue 开发一个全新的项目，也可以将 Vue 引入到一个现有的项目中。Vue 的核心代码库集中关注 MVVM 模式中的视图模型层层，通过双向数

据绑定连接视图和模型，API 十分简单和灵活，易于开发者上手。另外，Vue 能够为复杂的单页引用提供驱动，当开发与现代化的工具链和各种类库结合使用时。Vue 的视图模型图 2-2 所示。

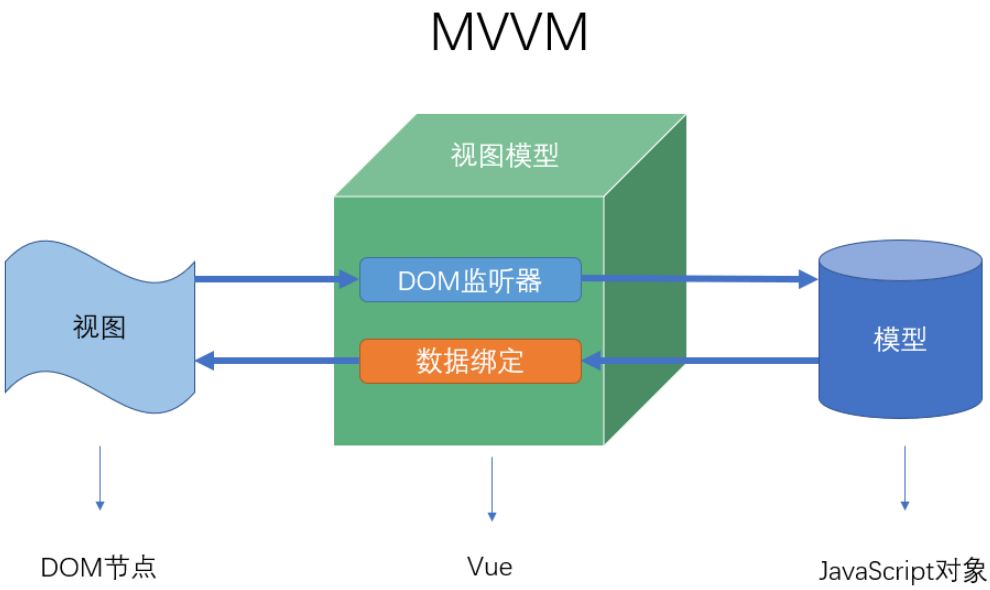

图 2-2 Vue 中的视图模型图

2.1.3 全局状态管理器 Pinia.js 介绍

状态管理是 WEB 应用程序开发的基石之一。任何重要的应用程序都需要某种状态管理。多年来，Vuex 是 Vue 应用程序事实上的状态管理工具。

而 Pinia.js[15]（下称 Pinia）是由 Vue.js 的团队主要核心成员使用 Composition Api 重新设计开发的最新一代状态管理器，也视为下一代 Vuex。Pinia 是 Vue 的状态管理库，允许跨组件和页面全局共享状态。其完全支持 TypeScript 并为 JavaScript 代码提供自动完成功能，不需要重新进行复杂的配置。除此之外，Vue2 和 Vue3 都支持 Pinia，并且都支持 devtools。

2.2 音乐解析相关技术介绍

2.2.1 MIDI 音乐格式

MIDI[16]音乐格式是一种技术标准，是一个包含连接各种电子乐器、计算机和演奏、编辑和录制音乐的相关音频设备的通信协议、数字接口和电子连接器。

与 MP3 或 WAV 文件等常规音频文件不同，MIDI 文件所存储的并不是实际的音频

数据，而是具体什么时间包含什么音符和每个音符所持续的时间和音量，因而尺寸更小。MIDI 的优点除了文件尺寸小以外，还易于修改和操作，并且有多种电子乐器和合成器或数字采样声音可供选择。

2.2.2 Music21 音乐解析库

Music21 库[17]是一套计算机辅助音乐学工具包，可以在 Python 环境下导入 Music21 库对包括 MIDI，MusicXML，abc 等音乐格式的文件进行解析和处理。可以借助该工具包完成对音乐文件的音乐分析，操作音符，和弦等等元素。Music21 中的基本单位是类似列表的 Stream，可以存储任意 Music21 对象及其组合。可以通过分析读取 Stream 进行分析出 MIDI 音乐格式的具体音符序列。

2.3 通讯协议技术介绍

2.3.1 HTTP 协议

超文本传输协议（Hyper Text Transfer Protocol，HTTP）[18]是一个用于分布式、协作式和超媒体信息系统的应用层协议，是运行在 TCP 之上的请求-响应机制的协议。

2.3.2 MQTT 协议

MQTT 是用于物联网（IoT）的 OASIS 标准消息传递协议。它被设计为一种极其轻量级的发布/订阅消息传输，非常适合连接代码空间小和网络带宽小的设备。如今，MQTT 被广泛伊宁用与各个行业，例如汽车、制造、电信、石油和天然气等。

MQTT 的客户端占用空间非常小，所需要的资源也很少，所以对于带宽的需求很小。此外，通过设置消息的服务质量级别，可以保证消息可靠的消息传递。通过订阅消息的方式，可以做到设备和云之间的双向通信，使得广播消息变得容易。在许多不稳定和不可靠的物联网设备环境下，可以保持客户端与代理的持久对话。在安全性上，MQTT 使用 TLS 加密消息和使用现代身份验证协议，使得对客户端的身份验证变得更加容易。一个常见的基于 MQTT 的发布订阅结构如图 2-3 所示。

图 2-3 常见的基于 MQTT 的发布订阅结构

2.4 机器人开发板相关技术介绍

ESP8266[19]是适用于物联网应用的经济高效且高度集成的具有 Wi-Fi 功能的微控制器，具有高耐久性，紧凑，节能和处理器利用率高等特点。ESP8266 板支持众多编程语言，包括原生的 C 语言，MicroPython，Lua 和 JavaScript 等。

ESP8266 网络模块具有三种工作模式，分别是：

（1）STA 模式：通过连接路由器连接局域网或互联网，其他设备可以通过路由器或者互联网实现对设备的远程控制。

（2）AP 模式：通过作为可以使手机和电脑相连的热点，实现手机和电脑与 ESP8266 的交互。

（3）STA+AP 模式：上述两种模式共存的模式。

第三章 演奏机器人中控系统设计

上一章介绍了设计演奏机器人中控系统所需要的技术方法。本章将根据上述技术方法对演奏机器人中控系统进行设计，具体包括系统设计思想、需求分析和系统总体设计。

3.1 系统设计目标

我们需要一套管理机器人并且指挥机器人演奏的中控系统软件。在此之前，控制机器人使用 Python 代码直接解析音乐结果并运行和发送指令，这种方案没有直观的用户界面，每次修改音乐来源和参数都需要修改原代码并重新运行，具体的音乐绑定规则需要提前固定等等缺点。因此根据控制机器人的需求，需要设计一套针对演奏机器人的具有良好用户界面的中控系统。用户可以通过系统实现自定义 MQTT 服务器，对机器人网络进行初始化，并通过系统管理机器人列表，查看机器人状态。同时通过查看系统解析返回的音乐轨道和音符信息，对机器人或机器人动作进行绑定，从而执行相应的音乐信息发送。

3.2 系统需求分析

3.2.1 系统功能性需求分析

演奏机器人中控系统需要正常连接对应的 MQTT 服务器，并且连接和管理自己的音乐演奏机器人，在具有机器人的情况下进行选择音乐文件解析并且绑定相关动作，最后发送指令至对应的机器人。用例图如 3-1 所示。

从用例图可以了解，在系统中存在三个角色，分别是用户，演奏机器人和 MQTT 服务器。用户的用例可以分为管理 MQTT 连接，管理机器人列表和音乐演奏三个部分。演奏机器人则分为网络初始化，消息订阅和演奏三个部分。则 MQTT 服务器则需要管理客户端的连接和将消息分发。下面对重要的部分进行详细的需求分析。

（1）管理 MQTT 服务器

用户需要管理客户端和机器人所共同连接的 MQTT 服务器，包括 MQTT 服务器的连接，断开和修改配置。

用户可以通过输入 IP 地址，端口，终端，客户端 ID，选择性输入账号和密码实现与 MQTT 服务器的连接。

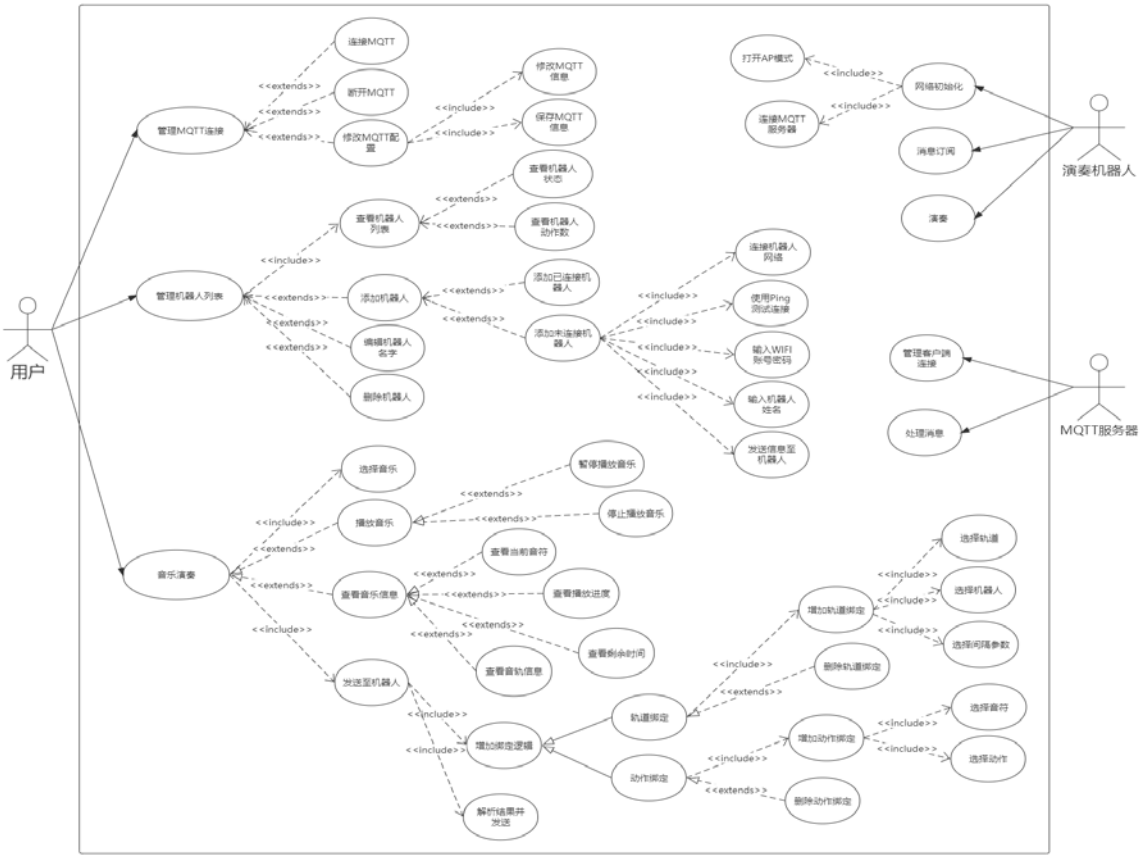

图 3-1 演奏机器人中控系统用例图

在 MQTT 服务器已经连接的状态，可以断开与服务器的连接，并且修改 MQTT 服务器的配置重新连接。

（2）管理机器人列表

用户可以查看当前已在客户端中添加过的机器人列表，以便管理与知悉当前的机器人具体连接。通过列表可以查看机器人的名字，机器人 ID，动作数量和状态。

用户可以选择某一行的机器人对机器人的名字进行修改，以便区分不同机器人。

用户对于已经废弃或者不希望出现的机器人，可以选择删除操作。

（3）音乐演奏

当用户已经连接好机器人时，可以进行音乐演奏模块的流程。

用户在解析音乐之前，首先要指定解析的音乐文件。用户可以选择接下来希望执行的音乐文件。选择某音乐文件后，可以播放音乐，暂停播放和停止播放音乐。

在音乐播放的同时，用户可以查看当前播放的情况，包括当前播放的进度条，音乐当前音轨，音乐当前播放的音符和剩余时间。

用户可以通过增加轨道绑定和动作绑定，进行将动作指令发送至机器人。轨道绑定指的是用户将某一轨道上的全部音乐与某一音乐机器人绑定，并指定间隔参数后，当音乐开始播放时向对应音乐机器人发送开始演奏指令，并在停顿将大于间隔参数时间时向对应音乐机器人发送停止演奏指令。而动作绑定是指用户可以通过选择某个轨道上的某个音符号与某个机器人的某号动作进行绑定，则当音乐播放至该轨道上的该音符时向该机器人发送执行该号动作指令。

（4）网络初始化

对于尚未网络初始化的机器人是没有办法连入用户指定的 MQTT 服务器的，所以机器人需要进行网络初始化，机器人的网络初始化分为两个部分，一是为了接入互联网或者与客户端所在的局域网，用户需要告知机器人需要连接的路由器的 WIFI 名字和密码，从而使机器人接入互联网或局域网。二是机器人需要连接用户指定的 MQTT 服务器，则网络初始化时也应该连接指定的 MQTT 服务器。

故在系统中添加未连接机器人时，通过连接机器人开启 AP 模式后的热点，可以通过 Ping 测试与机器人的连接，并且可以输入路由器 WIFI 名字和密码，将参数传输给机器人。

（5）接收消息

机器人可以通过订阅不用的主题来接收并区分不同的消息，并且根据相应主题所接受到的消息判断需要执行的动作。可以订阅的主题为控制主题和动作主题。

（6）管理客户端连接

对于 MQTT 服务器而言，可以管理不同的客户端发起的连接请求并保持客户端的连接状态，在客户端发起断开时可以进行连接的断开。

（7）消息分发

对于 MQTT 服务器，当某个主题有消息发送过来时，可以向下游订阅该主题的所有客户端转发此条消息。

3.2.2 系统非功能性需求分析

（1）准确性

对于音乐解析的结果需要保证准确，不存在绑定了但没执行或者没有绑定但是执行的情况。

（2）可靠性

对于用户所执行的可能对系统造成错误影响的流程，不使其影响整体系统的运行，减少导致程序崩溃或报错的异常 Bug 出现，具有良好的容错性并且

（3）可维护性

系统应该保持可维护性，保证部署和维护简单。

（4）交互体验良好

用户在操作系统时应保证用户的交互体验良好，具有良好的人机交互的界面。

（5）低延迟性

用户在使用系统时，不应该感受到卡顿，在解析音乐获取结果、根据绑定逻辑获取动作指令以及发送指令至机器人过程应该保证低延迟。

3.3 系统总体设计

3.3.1 系统架构设计

演奏机器人中控系统设计的层次架构如图 3-2 所示。

第一层为前端，由 Electron 构建客户端框架，由 Vue.js 负责页面渲染和数据响应，JavaScript 负责页面逻辑。

第二层为用户层，包括选择音乐并绑定逻辑的用户和所指定进行演奏的机器人。

第三层为视图层，主要负责了页面的展示逻辑，包括机器人管理页面，MQTT 管理页面，音乐演奏页面和逻辑绑定页面。

第四层为控制层，将系统的逻辑分为机器人维护，MQTT 管理，音乐解析和逻辑运算四个部分。通过将这些模块和接口可以保证系统核心逻辑部分的正常运行。

第五层为数据传输层，在系统运行过程中需要将数据进行传输，通过 HTTP 协议可以请求本地的 Python 构建的 Flask 服务器，通过 MQTT 协议可以与 MQTT 服务器进行连接。

第六层为服务层，包括可以解析音乐文件并返回逻辑处理结果的本地 Python 环境下的 Flask 服务，对本地接口进行调用的 Node.js 的 Express 服务和需要进行主题订阅和发布的 MQTT 服务。

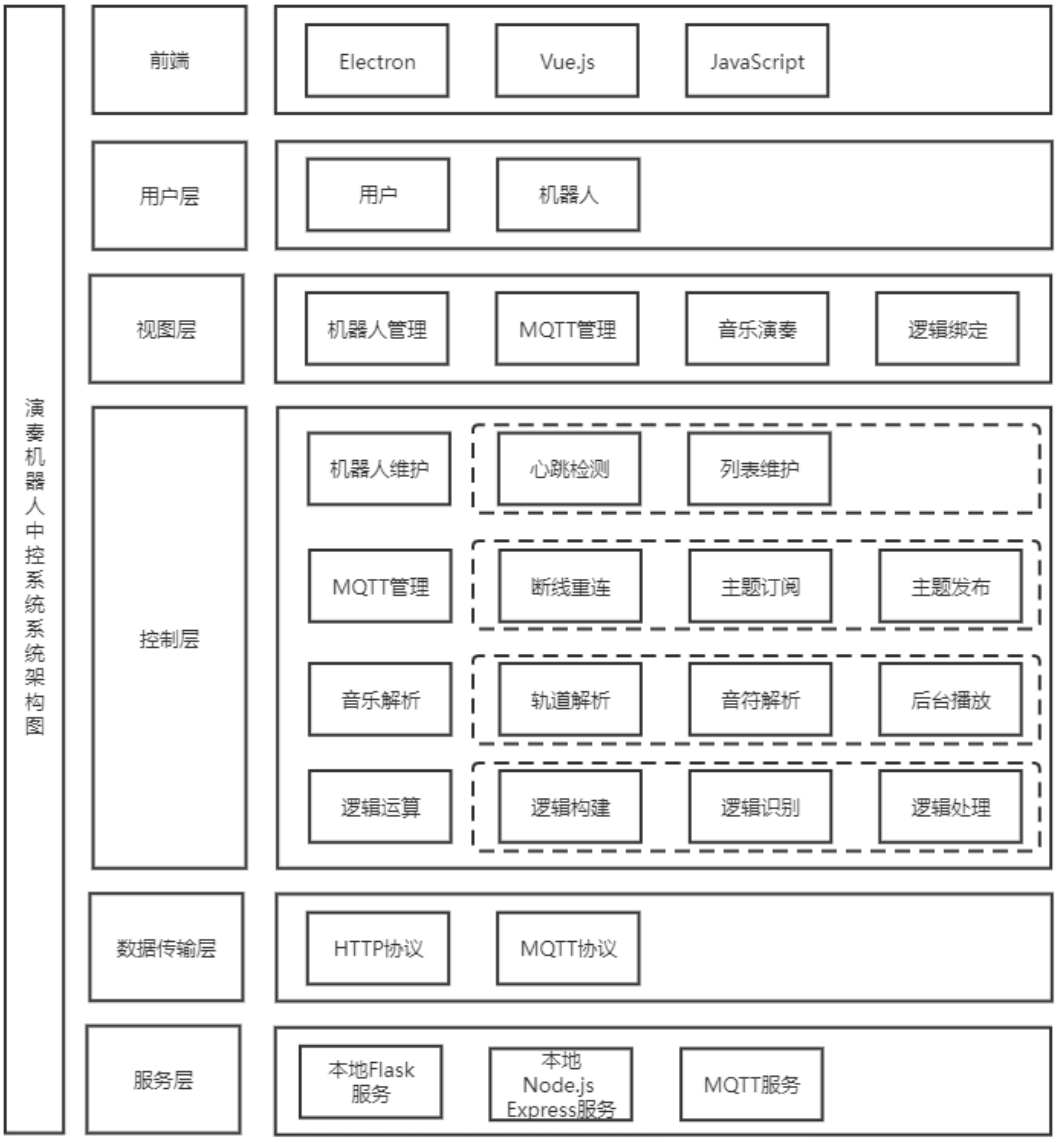

图 3-2 演奏机器人中控系统架构图

3.3.2 系统模块设计

根据需求分析和整体方案设计进行软件的模块的总体设计与开发。本软件主要包括：机器人管理模块，MQTT 管理模块，音乐演奏模块和音乐解析模块。各模块如图 3-3 所示。

MQTT 管理模块主要负责客户端与 MQTT 服务器之间的连接，包括连接 MQTT，断开 MQTT 连接，修改 MQTT 配置并保存。此模块确保了后续的消息可以正常发送至机器人。

机器人管理模块主要负责呈现展示当前的机器人列表详细信息，功能包括机器人添

加，当机器人网络未初始化时对机器人的网络进行初始化，还有修改机器人名字和删除列表中的机器人等功能。

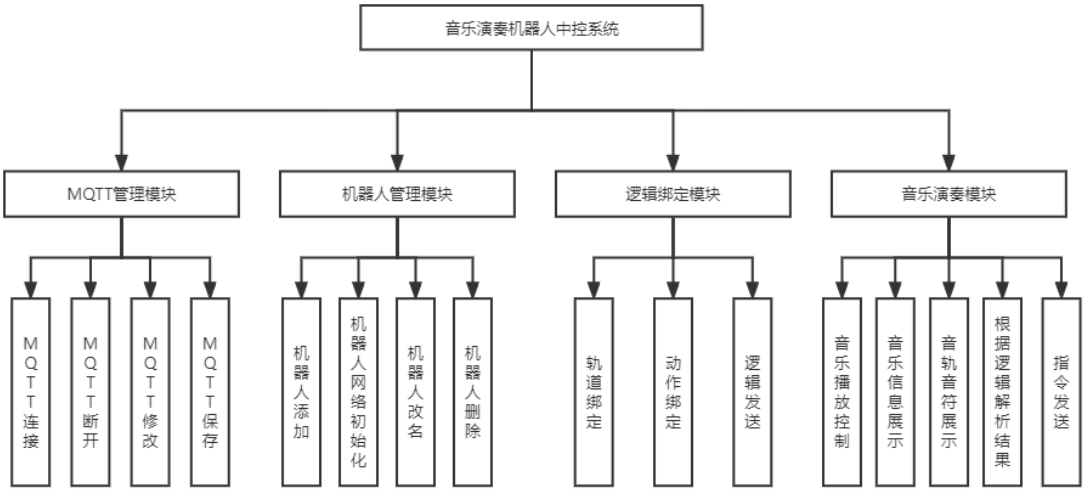

图 3-3 演奏机器人中控系统模块图

逻辑绑定模块主要负责用户绑定逻辑的实现。用户在此模块进行轨道绑定和动作绑定。

音乐演奏模块主要负责在用户选择文件之后对音乐文件进行解析，并且能够播放音乐，暂停和停止播放。在用户选择了逻辑绑定之后可以将音乐文件根据逻辑转化为动作指令并返回，从而达到将动作指令发送至机器人的效果。

系统模块的逻辑如图 3-所示，用户通过交互界面对机器人进行管理以及对音乐逻辑进行轨道绑定和动作绑定，并通过 MQTT 模块将指令发送至对应机器人。而机器人通过心跳主题将心跳包发送至客户端，由机器人管理模块进行心跳的检测并更新机器人列表中机器人的在线情况。

3.4 本章小结

本章对音乐演奏机器人中控系统进行了详细的需求分析，从用户的需求的角度出发，分析我们系统的功能需求和非功能需求。再由需求分析和技术栈梳理设计出系统的总体架构和模块结构。

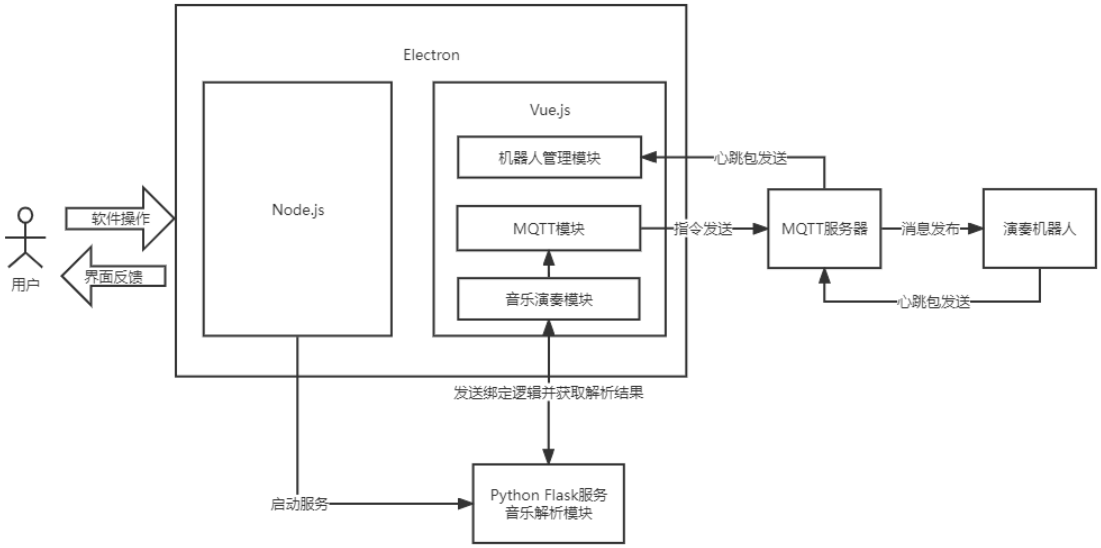

图 3-4 演奏机器人中控系统模块逻辑图

第四章 演奏机器人中控系统实现

4.1 系统项目组织结构

整个系统的项目组织结构如表 4-1 所示。

表 4-1 系统项目组织结构表

目录名称/文件名称	目录说明
Dist	项目正常打包后的文件目录
Dist_electron	项目通过 electron 打包后的文件目录
Node_modules	项目依赖文件目录
Public/js	引入的外部 js 文件夹
Public/midifiles	存放 midi 文件的文件夹
Public/robot	机器人运行代码文件夹
Public/soundfont	音质 js 文件
Resource	存放音乐解析的 Python 代码的文件夹
Src/assets	存放静态图片的文件夹
Src/assets/styles	存放 CSS 文件的文件夹
Src/components	存放组件的文件夹
Src/homepage	存放一级页面的文件夹
Src/secondPage	存放二级页面的文件夹
Src/router/index.js	统一路由配置文件
Src/stores/mqttClient.js	MQTT 全局状态存储
Src/stores/pathStore.js	路径全局状态存储
Src/App.vue	主页面
Src/background.js	Electron 中主进程文件
Src/main.js	Vue 中入口文件
Src/preload.js	Electron 中预加载文件，node.js 实现的 express 服务代码
package.json	项目依赖管理文件
vue.config.js	Vue 项目配置文件

其中关键页面组件如下表 4-2 所示：

表 4-2 关键页面组件表

组件名	页面介绍
RobotListPage.vue	机器人列表页面
MusicPlayPage.vue	音乐解析演奏页面
SetPage.vue	配置 MQTT 页面
PlsConnectWifiPage.vue	机器人网络初始化页面
InfoConfirmPage.vue	信息确认页面
WifiPing.vue	Ping 测试连接页面
WifiInput.vue	Wifi 信息输入页面

4.2 MQTT 管理模块实现

在此模块中，管理着 MQTT 客户端的连接，断开，修改和保存配置，并负责主题的订阅与消息发布。

4.2.1 MQTT 管理模块流程

用户在进入系统界面时，会判断当前 MQTT 客户端连接状态是否正常，当当前状态处在断开时，会提示用户进行 MQTT 客户端参数的配置。用户通过输入参数并且连接 MQTT 服务器成功时，则会弹出连接成功的提示。但连接超过三秒仍然未连接成功时，会弹出连接失败的提示，用户可以通过检查网络或者更改当前客户端参数再次尝试连接。当用户需要更改客户端参数时，需要先断开客户端的连接，编辑参数并保存后，方可再次连接。

4.2.2 MQTT 客户端参数

系统需要连接 MQTT 服务器，则需要指定连接的一些参数。创建 MQTT 客户端所需要的选项如表 4-3 所示：

表 4-3 MQTT 客户端参数

字段名	类型	含义
Host	String	MQTT 服务器主机
Port	String	MQTT 服务器端口

表 4-3 MQTT 客户端参数（续）

字段名	类型	含义
Endpoint	String	MQTT 服务器终端
Clean	Boolean	是否保留会话标志
connectTimeout	Int	超时时间
reconnectPeriod	Int	重连时间间隔
ClientId	String	客户端 ID
Username	String	用户名
Password	String	密码

4.2.3 MQTT 客户端回调方法

通过定义客户端触发不同的事情所执行的不同动作，即通过客户端的回调函数，可以帮助用户获取客户端的状态，增加事件触发时的自定义处理，同时可以接收消息。常用的 MQTT 客户端回调方法如表 4-4 所示。

表 4-4 MQTT 客户端中的回调方法

触发事件	触发方式
Connect	客户端连接成功时
Reconnect	客户端重新连接时
Close	断开连接时
Disconnect	收到代理发出的断开数据包时
Offline	客户端离线时
Error	无法连接或发生解析错误时
End	客户端调用 end()时
Message	客户端接收到消息时

MQTT 管理模块中对于回调方法的使用如客户端连接成功时触发 connect 事件，将模块中记录 MQTT 连接状态的参数设置为连接成功，从而可以在外部获取当前 MQTT 客户端的连接状态，代码如下所示。

```
this.client.on('connect', () => {
        this.createSuccess = true
        console.log('Connection succeeded!')
        ElMessage({
          message: 'MQTT 服务器连接成功',
          type: 'success',
          grouping: true
        })
        console.log(this.client.connected)
        this.clients.connected = true
      })
```

MQTT 客户端在订阅了主题之后，一旦接收到消息，就会触发 message 回调方法，在此方法中可以判断消息的来源主题是哪个机器人的心跳主题，从而更新该机器人的“上次活跃时间”，为之后的判断机器人是否处在正常连接中做准备，代码如下。

```
this.client.on('message', (topic, message) => {
        console.log(`Received message ${message} from topic ${topic}`)
        var dataString = ''
        for (var i = 0; i < message.length; i++) {
          dataString += String.fromCharCode(message[i])
        }
        if (this.robotConnectedList.length > 0) {
          for (let index = 0; index < this.robotConnectedList.length; index++) {
            const robot = this.robotConnectedList[index]
            if (robot.heartbeatTopic == topic) {
              robot.actionNum = parseInt(dataString)
              robot.lastBeatTime = new Date().valueOf()
            }
          }
```

```
        }
    })
```

4.2.4 MQTT 客户端界面展示

MQTT 客户端连接界面如图 4-1 所示。

图 4-1 MQTT 客户端连接界面

4.3 机器人管理模块实现

机器人的管理模块根据功能可以分为增加机器人，查看机器人列表，改动机器人和删除机器人四个部分。

4.3.1 机器人管理模块流程

用户首先查看机器人列表，若当前 MQTT 服务尚未连接，则提示用户前往配置 MQTT 服务器。若 MQTT 服务已连接且机器人列表为空，即用户尚未添加机器人时，系统将提示用户添加机器人。

用户添加机器人的流程如图 4-2 所示。机器人可以通过两种方式添加：添加网络已初始化的机器人和添加网络尚未初始化的机器人。机器人网络已初始化则可以通过机器人 ID 号增加，若机器人网络尚未初始化，则需要通过一系列流程来将机器人网络初始化；机器人网络初始化流程包括：连接机器人热点，Ping 测试与机器人的连接，填写路由器 WIFI 账号和密码，确认消息四个部分。通过这四个部分则将机器人所需要连接网络的各个信息发送至机器人，等待机器人回复，机器人回复成功则将机器人添加至机器人列表。

用户可以通过机器人列表查看机器人信息，信息包括机器人名字，机器人 ID，动作数和连接状态。

用户可以修改机器人的名字，以便区分不同机器人。

用户可以在认为机器人废弃或机器人发生变动时将机器人从列表中删除。

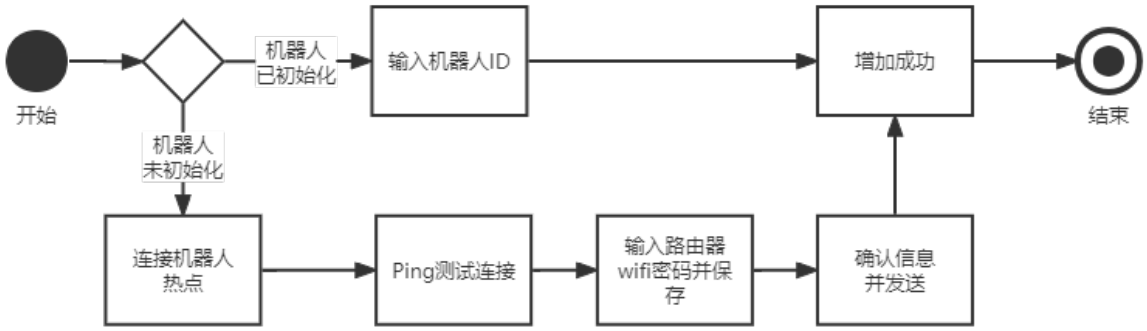

图 4-2 增加机器人流程图

4.3.2 机器人列表字段

机器人列表中每个机器人存储的字段如表 4-4 所示。

表 4-4 机器人字段表

字段名	类型	含义
robotName	String	机器人名字
robotId	Number	机器人 ID
heartbeatTopic	String	心跳主题
commandTopic	String	控制主题
actionTopic	String	动作主题
status	Boolean	状态（在线或离线）
lastBeatTime	Number	上次心跳时间
actionNum	Number	动作数

4.3.3 机器人管理模块展示

机器人管理模块中的机器人列表如图 4-3 所示，机器人网络未初始化进入机器人网络初始化的指引界面如图 4-4 所示。

图 4-3 机器人列表展示图

图 4-4 添加机器人指引界面展示

4.4 逻辑绑定模块实现

4.4.1 逻辑绑定模块流程

逻辑绑定模块时序图如图 4-5 所示。用户想要绑定某个轨道对应某个机器人，和绑定某个轨道上的某个音符对应某个动作，首先要保证当前音乐文件解析出的轨道和音符结果正常和当前机器人列表中存在可以使用的机器人。在选择了音乐文件并且获取了当前轨道和音符序列后，将轨道和音符序列初始化为 JavaScript 所能识别的 Json 类型，

方能被用户用 Cascader 级联选择器选择。并且由机器人的动作数初始化机器人的动作列表，然后供用户选择动作。

用户选择轨道绑定则选择某一条轨道，并且选择某一个机器人，点击保存则完成了轨道绑定。用户选择任意轨道上的任意音符，然后选择某存在动作的机器人的某个动作，点击保存则完成了动作绑定。

完成绑定后的逻辑绑定序列将发送至 Python Flask 服务进行将逻辑解析为最终的动作序列，以供发送。

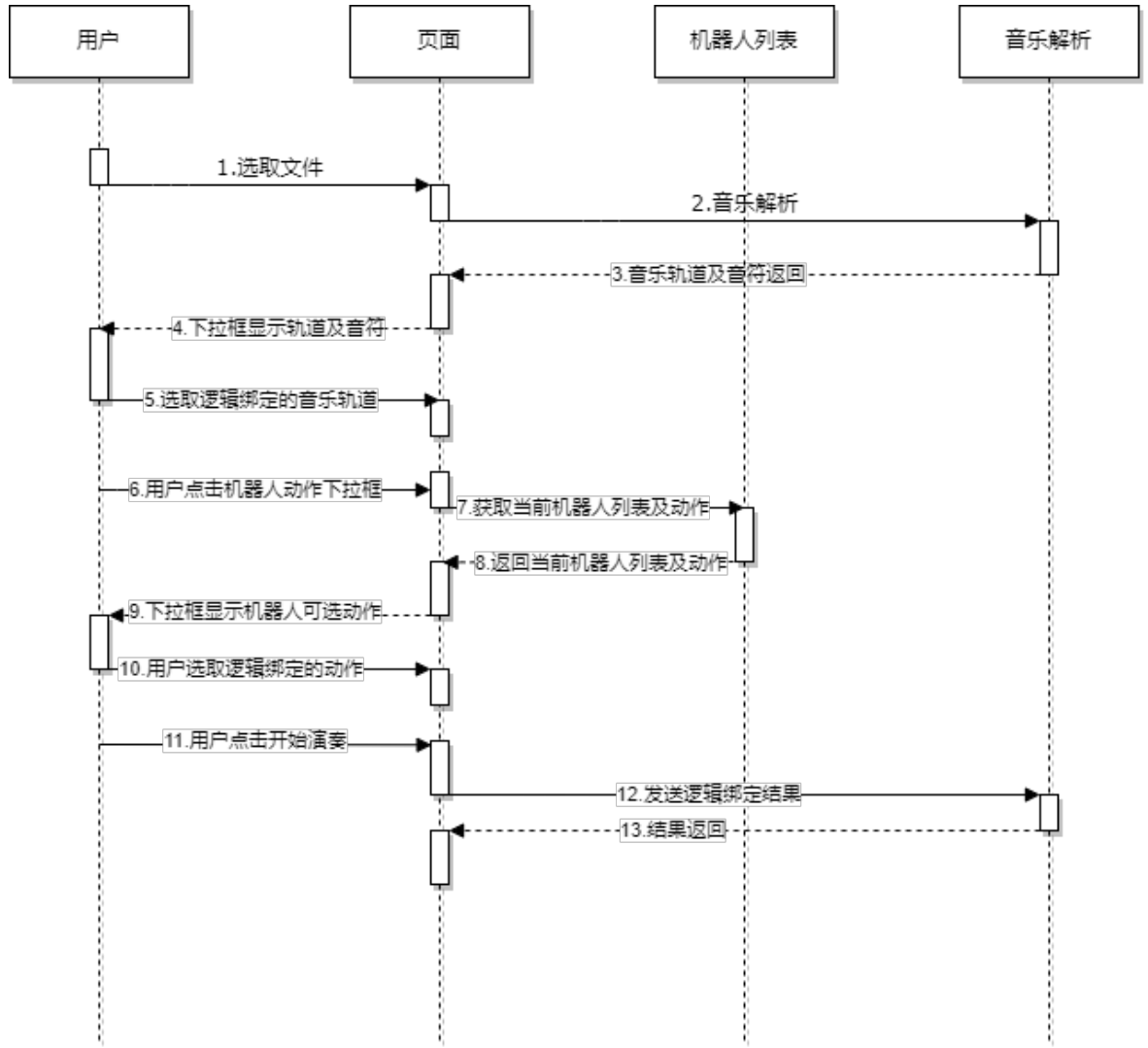

图 4-5 逻辑绑定模块时序图

4.4.2 逻辑绑定模块关键代码

4.4.2.1 创建选择列表

解析音乐并返回轨道与去重后的音符的可供用户选择的列表的伪代码如下，通过

python 的 music21 库解析出音乐的各个轨道信息，再分析每个轨道上的每个单位是否为音符，如果为音符则加入当前音轨上的 map 中去重保存，如果为和弦则取出其中的每个单个音符加入 map 中去重保存。同时为了满足前端级联选择器的格式，需要加入 value，label，disabled 和 chidren 等键值：

```
# 获得前端所需要的轨道和音符 Map
def getGetTrackAndNode(filepath):
    初始化 trackAndNodeList
    获取对于音乐的解析流
    for 轨道 in 音乐解析流:
        返回列表中以级联选择器格式加入该轨道
        for 元素 in 轨道内容:
            if 元素是音符&&元素没有被加入过
                将音符以级联选择器格式加入返回列表
                标识该元素已被加入
            if 元素是和弦
                for 音符 in 和弦
                    if 元素是音符&&元素没有被加入过
                    将音符以级联选择器格式加入返回列表
                    标识该元素已被加入
    返回 trackAndNodeList
```

根据机器人的动作数创建可供用户选择的选择列表代码如下：

```
//返回机器人的动作列表
    getActionOptions() {
      var actionOptions = []
      for (let index = 0; index < this.robotConnectedList.length; index++) {
        const robot = this.robotConnectedList[index]
        console.log(robot.actionNum)
        if (robot.actionNum > 0) {
```

```
                console.log('robot.actionNum>0')
                var robotjson = {}
                robotjson.value = index
                robotjson.robotId = robot.robotId
                robotjson.label = robot.robotName
                robotjson.children = []
                for (let i = 0; i < robot.actionNum; i++) {
                  var actionjson = {}
                  actionjson.value = i
                  actionjson.label = '动作' + String(i + 1)
                  actionjson.disabled = false
                  robotjson.children.push(actionjson)
                }
                actionOptions.push(robotjson)
              }
            }
            console.log('actionOptions', actionOptions)
            return actionOptions
          }
```

4.4.2.2 维护逻辑列表

最终的逻辑绑定结果存放在 thickLogicList（粗绑定逻辑列表，轨道绑定）和 thinLogicList（细绑定逻辑列表中，动作绑定）。当增加逻辑列表时会检测当前列表中是否存在未保存的逻辑项，并且保存时也会检查逻辑项是否为空。使用 Vue 的双向绑定和两个级联选择器来供用户选择，动作绑定的关键代码如下：

```
      <el-row style=″margin-top: 10px″ v-for=″(logic, index) in thinLogicList″ :key=″index″>
              <el-col :span=″10″>
    <el-cascader :disabled=″logic[2]″ :options=″NodeOptions″ v-model=″logic[0]″ placeholde
r=″请选择″
```

```
                    ></el-cascader>
                </el-col>
                <el-col :span="10">
    <el-cascader :disabled="logic[2]" :options="ActionOptions" v-model="logic[1]" pla
ceholder="请选择"
                    ></el-cascader>
                </el-col>
                <el-col :span="4">
    <el-button v-if="!logic[2]" type="success" @click="saveLogic(index)" icon="check"
circle></el-button>
    <el-button type="danger" @click="delLogic(index)" icon="delete" circle></el-button>
                </el-col>
            </el-row>
```

4.4.2.3 将逻辑转化为动作指令

由上述步骤获取轨道绑定和动作绑定的逻辑列表后，通过 Axios 请求 Python 环境下的 Flask 服务开启的将逻辑转化为动作指令的接口，从而获得动作指令。请求接口代码如下：

```
    axios({
            method: 'post',
            url: 'http://localhost:5000/midiAlgorithm',
            data: {
               thickLogicList: this.thickLogicList,
               thinLogicList: this.thinLogicList,
               url: this.pathArr
            },
            headers: {'Content-Type': 'application/json; charset=utf-8'}
        }).then(…)
```

Python 中当请求/midiAlgorithm 这个接口时，获取传过来的轨道绑定序列和动作绑

定序列，并且判断他们的有效性，若无效则不参加到后续的动作转换中。后续分别获取每一条绑定逻辑进行查询，若存在则将对应的动作加入到动作指令结果中。轨道绑定逻辑处理，使用 start 标记当前音乐是否正在播放，当音符出现时，倘若当前尚未播放，则记录当前时间执行 Start 动作至指令结果集 resultMap 中。倘若当前处在播放状态，则使用当前时间减去上次音符时间，判断是否大于用户所指定当前轨道的 interval 时间间隔参数，若大于该参数则将两次结果分段，若小于该参数则不分段，并更新上次音符时间。关键代码如下：

```
    # 粗绑定添加时间逻辑
    def midThick(resultMap, part, robotIndex, interval):
        current_ts = 0.0
        past_ts = 0.0
        start = False
        for item in part.secondsMap:
            e = item['element']
            current_ts = item['offsetSeconds']  # 记录时刻
    if not (isinstance(e, music21.note.Note) or isinstance(e, music21.chord.Chord)):  # 不是音符或和弦
                continue  # 忽略非 note
            # 第一次 start
            if start == False:
                # -2: stop,-1:start
                append_value(resultMap, current_ts, (robotIndex, -1))
                start = True
                past_ts = current_ts
                continue
            # 时间间隔超过阈值，添加 stop 和 start
            if current_ts - past_ts > interval:
                # print(past_ts,"stop")
```

```
                append_value(resultMap, past_ts, (robotIndex, -2))
                # print(current_ts,"start")
                append_value(resultMap, current_ts, (robotIndex, -1))
            #  修改时间，为下次准备
            past_ts = current_ts
        # print(past_ts,"stop")
        append_value(resultMap, past_ts, (robotIndex, -2))
```

在将动作绑定逻辑转化为动作指令时，根据每一个逻辑找到对应轨道的音符，记录当前音符的对应时间，并添加至 resultMap 完成动作的添加。

4.4.3 逻辑绑定模块界面

逻辑绑定模块界面如图 4-6 所示。上方为增加轨道绑定区域，下方为增加动作绑定区域。

图 4-6 逻辑绑定模块图

4.5 音乐演奏模块实现

4.5.1 音乐演奏流程

用户执行选择文件操作时，选择 MIDI 文件格式的音乐之后，触发 openPath()函数，

请求 python 环境下 Flask 服务的/musicChoose 接口，使得 python 中加载音乐文件，与此同时。随后通过请求/midiGetTrackAndNode 接口获取轨道和音符列表。当用户点击播放时，客户端请求/musicPlay 以达到调用 pygame 播放 MIDI 音乐。在播放的过程中，通过 MidiPlayer.js 可以获取当前播放的轨道信息和当前播放的音符，并且能够通过 getSongPercentRemaining()获取音乐剩余播放时间，以计算音乐播放占比，从而呈现出进度条。通过请求/musicPause 或/musicStop 接口以达到音乐暂停或停止。在开始演奏获取了动作执行指令后使用对每一个即将发送的动作指令进行延迟对应的时间发送，延迟使用 setTimeout()实现，并将每个 setTimeout()所标记的 ID 存入 timeoutList 以供后续更改。在用户选择执行停止演奏时将 timeoutList 中的所有标识 setTimeout 的 ID 进行 clearTimeout()。

4.5.2 音乐演奏接口实现

使用 python 实现的 Flask 服务器实现了众多操作解析 midi 与播放 midi 的接口，接口列表如表 4-5 所示：

表 4-5 Flask 音乐操作接口列表

接口名	作用
/musicChoose	选择音乐
/musicPlay	播放音乐
/musicPause	暂停播放音乐
/musicStop	停止播放音乐
/musicMute	将音乐静音
/musicUnMute	接触音乐静音
/midiGetTrackAndNode	获取轨道和音符序列
/midiAlgorithm	获取由逻辑的解析结果

4.5.3 音乐演奏关键代码实现

4.5.3.1 播放音乐

使用 pygame 的 mixer.music 模块，可以解析 MIDI 音乐并在后台播放

```
freq = 44100    # audio CD quality
```

```
bitsize = -16    # unsigned 16 bit
channels = 2    # 1 is mono, 2 is stereo
buffer = 2048    # number of samples (experiment to get right sound)
pg.mixer.init(freq, bitsize, channels, buffer)
# 音乐开始或者恢复播放
@app.route('/musicPlay', methods=['GET'])
def musicPlay():
    print(pg.mixer.music.get_pos())
    if(pg.mixer.music.get_pos() >= 0):
        pg.mixer.music.unpause()
        clock.tick(30)
        return {"data": "unpause"}
    try:
        pg.mixer.music.play()
        # 如果在播放则是 True，如果不在播放则返回 False
        while pg.mixer.music.get_busy():
            print(pg.mixer.music.get_pos())  # 获得音乐播放时间
            clock.tick(30)
    except pg.error:
        return {"data": "playFailed"}
    finally:
        return {"data": "MusicStart"}
```

4.5.3.2 实时获取音符

使用 MidiPlayer.js 获取当前播放音符并实时显示在界面中。

```
reader.addEventListener('load', function () {
   Soundfont.instrument(_this.ac, './js/acoustic_guitar_nylon-mp3.js').then(() => {
      _this.Player = new MidiPlayer.Player(event => {
         if (event.name == 'Note on') {
```

```
                _this.currentTrack = event.track - 1
                _this.currentNodeName = event.noteName
                _this.sendToRobot(event.noteName, event.track - 1)
              }
                _this.musicPercent = 100 - _this.Player.getSongPercentRemaining()
                _this.changeRemainTime()
            })
            _this.Player.loadArrayBuffer(reader.result)
            _this.pushToTrackList()
            _this.haveMid = true
            _this.changeRemainTime()
          }, false)
    })
```

4.5.4 音乐演奏界面展示

音乐演奏的界面如图 4-6 所示。上部分是文件选择，播放控制，信息展示，底部是绑定与演奏控制。


rcc
File Edit View Window Help
音乐机器人控制中心
v1.0
机器人列表
音乐演奏
MQTT配置
MQTT服务器状态:
C:\Users\Kim\Desktop\drum_bang_piano.mid
选择
暂停
停止
现在的音符:
B4
剩下的时间:
0:10
音轨数:
3
当前音轨:
2
动作发送
音符发送
轨道绑定
轨道
机器人
间隔参数
操作
架子鼓
机器人1
2.00
钢琴
机器人2
2.00
增加粗绑定
动作绑定
轨道/音符
机器人/动作
操作
响棒 / E-
机器人1 / 动作2
钢琴 / B
机器人2 / 动作3
增加细绑定
演奏控制
停止演奏

图 4-6 音乐演奏界面图

第五章 系统部署与测试

5.1 系统部署的硬件配置与软件环境

系统部署的硬件配置：

CPU：AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz

运行内存：16GB

GPU：NVIDIA GeForce RTX 3050 Laptop GPU

软件环境：

操作系统：Windows 11 家庭中文版

Node 版本：16.14.0

Npm 版本：8.5.4

Vue cli 版本：5.0.1

Python 版本：3.10.4

Pip 版本：22.0.4

Flask 版本：2.1.2

Flask-Cors：3.0.10

Music21 版本：6.7.1

Pygame 版本：2.1.2

5.2 系统的构建与部署

通过 npm run electron:build 可以打包软件为 exe 可执行文件，打包好的文件存在于同目录中下的 dist_electron 文件夹，文件名为 Music Robot Control Center Setup 0.1.0.exe。双击打开 Music Robot Control Center Setup 0.1.0.exe 之后会自动安装，安装完成后会自动打开界面并且在桌面创建名为 Music Robot Control Center 的软件快捷方式。

5.3 系统测试

5.3.1 测试方法

系统测试采用的方法是功能测试，根据软件中的各个功能需求点进行用例测试，查看并记录软件的测试结果。

5.3.2 测试结果

测试结果如表 5-1 所示。

表 5-1 系统测试结果

模块	功能	用例点	测试结果	是否通过
MQTT 管理模块	连接 MQTT	输入正确的连接参数	连接成功	通过
		输入错误的连接参数	三秒后显示连接失败	通过
	断开 MQTT 连接	点击断开连接	断开成功	通过
		断开网络		通过
	修改 MQTT 配置	连接中点击编辑	弹出提示先断开 MQTT 连接	通过
		断开 MQTT 后点击编辑	可以编辑	通过
		编辑后点击保存	配置被修改，保存成功	通过
机器人管理模块	查看机器人列表	机器人为空	显示提示“请添加您的机器人”	通过
		机器人存在	机器人列表展示正常	通过
	添加机器人按钮	MQTT 未连接时点击	弹出请连接 MQTT 提示	通过
		MQTT 已连接时点击	弹出机器人网络是否已初始化选择项	通过
	已初始化机器人添加	输入正确机器人 ID 并点击确认	添加成功	通过
		输入错误机器人 ID 格式并点击确认	提示请输入正确的机器人 ID	通过
	未初始化机器人添加	点击步骤一的下一步按钮	弹出步骤二	通过
		已正确连接机器人热点下点击步骤二的终端 Ping 测试按钮	Ping 结果框中显示正确连接的结果	通过
		非正确连接机器人热点下点击步骤二的终端 Ping 测试按钮	Ping 结果框中显示请求超时的结果	通过

表 5-1 系统测试结果（续）

模块	功能	用例点	测试结果	是否通过
		步骤三中修改 WIFI 名称和密码后点击保存	提示保存成功，并且使得下一步可用	通过
		步骤四中更改机器人名字	更改成功	通过
		步骤四中点击发送至机器人	显示发送成功和用户网络切换提示	通过
		步骤四中点击完成机器人添加	列表增加成功	通过
音乐演奏模块	选择音乐	选择 midi 格式音乐	选择后显示文件路径，音轨数，轨道中显示轨道名	通过
	播放音乐	点击播放音乐	音乐开始播放，进度条开始启动，剩余时间变化	通过
		暂停后，重新播放音乐	音乐正常播放	通过
	暂停音乐	点击暂停音乐	音乐暂停播放	通过
	停止播放音乐	点击停止音乐	音乐停止，重新播放时从头开始播放	通过
	增加轨道绑定	列表为空时，点击增加轨道绑定	轨道绑定列表增加一行供用户填	通过
		列表不为空且时，点击增加轨道绑定	轨道绑定列表增加一行供用户填	通过
		列表中存在增加但未保存的轨道绑定时点击增加	提示当前列表存在未保存的绑定	通过
	删除轨道绑定	点击对应行右侧的删除按钮	对应行的轨道绑定结果被删除	通过
	轨道绑定选择	点击轨道下拉框	显示可选择音乐中的不同轨道	通过
		点击机器人下拉框	显示可选择机器人	通过

表 5-1 系统测试结果（续）

模块	功能	用例点	测试结果	是否通过
		点击轨道/音符下拉框	显示可选择的轨道上的音符	通过
		点击动作下拉框	显示可选择的机器人动作列表	通过
	开始演奏	逻辑绑定后点击开始演奏	音乐播放，同时机器人接收到对应指令	通过
	停止演奏	点击停止演奏	音乐停止，同时机器人接收指令停止	通过

5.3.3 测试结果分析

根据以上测试结果可知，软件基本功能较为完善，根据用户的需要制定的功能能够正常运行。对于取得最后的演奏结果需要前面很多的步骤来确保最后的正确性，所以有很多用户的引导，满足用户的使用需要。不过页面风格设计仍然存在改善的空间。

第六章 结论

6.1 论文工作总结

本文面向控制音乐演奏机器人的场景，聚焦控制方案中的控制机器人连接问题、逻辑绑定和音乐解析发送问题，设计了一款基于 Electron 和 Vue，采用 Element-plus 作为 UI 框架，采用 python 解析音乐，并通过 MQTT 协议向机器人进行发布指令的音乐演奏机器人控制系统。

在软件开发的过程中，本文首先需要确定用户的需求，通过与老师与助教的不断交流中明确用户使用软件的目标需求，然后通过用户需求思考系统设计，模块设计，界面设计，技术选型。模块设计中将软件分为四个部分，分别为 MQTT 管理模块，机器人管理模块，逻辑绑定模块和音乐演奏模块。接下来对各个模块进行逐步开发，最后对各个模块的功能进行不断地改善与测试。本文的主要工作总结如下：

（1）界面及风格设计。设计了呈现给用户的页面，界面的布局，内容结构和操作流程等等。

（2）客户端与本地代码通信。软件中有多处客户端与本地脚本之间的通信，我们需要决定什么时候去调用本地脚本，并从本地脚本中获取想要的结果并呈现。例如调用命令行中的 Ping 命令从而查看客户端与机器人是否通过 AP 模式连接成功，通过初始化本地调用 python 执行 Flask 脚本启动本地服务器等等。我们需要结合 Electron 基于 Node.js 的特性，从而更方便地开启和关闭子进程。

（3）MQTT 客户端管理。在客户端运行的过程中我们是需要保证 MQTT 客户端处于连接状态的，但假如将客户端的连接存放与某个页面中，则页面销毁的时候客户端也会随之销毁，所以寻求了全局状态管理的方案。将 MQTT 客户端的连接与管理放在 pinia.js 全局状态管理中，则可以在任何页面获取 MQTT 的状态以及操作。

（4）音乐播放控制。MIDI 音乐格式只是记录了音符，但没有记录音质，故并不能直接播放，而在试用了 midi.js 和 midi-player-js 和 soundfont 播放库结合解析之后，发现由于音质呈现的限制，更改方案使用 python 中的 pygame 库进行播放，并以接口的形式对 python 代码进行控制。

（5）音乐逻辑绑定与解析。MIDI 音乐格式的解析在 python 代码处，但用户对于轨道和音符的选择在客户端用户界面处，所以我们需要将轨道和去重后的音符，以级联

选择器格式方式编入集合中，并返回到客户端，从而让用户得以选择。并且在用户对轨道或者动作绑定选择后，需要再次将逻辑绑定结果发送至 python 代码处，才能进行后续的解析。在后续的解析中，根据用户所选择的逻辑解析出最后的指令也花费了不少时间。我们需要通过用户所选择的逻辑去定位音乐信息中对应音符和开始停止的时间位置，记录下来再返回至客户端的 JavaScript 部分进行 MQTT 的消息发布。

（6）用户提示和错误处理。在从保证客户端连接到增加机器人直至选择音乐绑定并解析最后开始发送至机器人开始演奏这个过程中，中间难免会出现用户操作失误。我们需要关心并且想象自己初次使用此款软件时的场景并不断模拟，在操作过程中进行适当的提示，和假设用户进行了错误的操作，加以判断并进行处理。

6.2 未来展望

本文面向控制音乐演奏机器人的场景，聚焦控制方案中的控制机器人连接问题、逻辑绑定和音乐解析发送问题设计了此系统，但仍然有许多需要改进的地方：

（1）界面设计方面。未来可以调整 CSS 和响应时的样式进行更加优化用户的观感体验，并且可以调整页面的布局和适配度等细节。

（2）用户指引方面。系统目前存在的指引需要用户仔细阅读，未来可以将一些操作变为后台自动操作，例如与机器人热点的连接与断开等等。

参考文献

[1] 徐国保, 尹怡欣, 周美娟. 智能移动机器人技术现状及展望[J]. 机器人技术与应用, 2007(02): 29-34.

[2] T. Wang, S. Hu, J. Xu, D. Yan and J. Bi, ″Simulation Design and Application of Music Playing Robot Based on SolidWorks, ″ 2009 International Conference on Measuring Technology and Mechatronics Automation, 2009, pp. 339-342, doi: 10. 1109/ICMTMA. 2009. 663.

[3] 付晓东. 音乐机器人的发展历史与技术成果[J]. 演艺科技, 2015(05): 12-17.

[4] 郭峰, 钱黎明, 沈煜, 张雨凌. 吉他演奏机器人的机械结构设计[J]. 机械设计与制造, 2020(01): 248-250+255. DOI: 10. 19356/j. cnki. 1001-3997. 2020. 01. 061.

[5] K. Shibuya, K. Kosuga and H. Fukuhara, ″Bright and Dark Timbre Expressions with Sound Pressure and Tempo Variations by Violin-playing Robot*, ″ 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2020, pp. 482-487, doi: 10. 1109/RO-MAN47096. 2020. 9223503.

[6] Y. Li and C. Lai, ″Development on an intelligent music score input system — Applied for the piano robot, ″ 2016 Asia-Pacific Conference on Intelligent Robot Systems (ACIRS), 2016, pp. 67-71, doi: 10. 1109/ACIRS. 2016. 7556190.

[7] 赵恒, 郭峰, 钱黎明. 架子鼓演奏机器人控制系统设计[J]. 现代制造技术与装备, 2021, 57(05): 60-61. DOI: 10. 16107/j. cnki. mmte. 2021. 0368.

[8] 付晓东. 音乐机器人的发展历史与技术成果[J]. 演艺科技, 2015(05): 12-17.

[9] 朱昱林. 云服务器环境下基于 Android 移动平台的机器人控制系统研究[J]. 机床与液压, 2021, 49(15): 71-76.

[10] LIU Peter Xiao-ping. Internet-based Teleoperation Platform For Mobile Robot[J]. 哈尔滨工程大学学报：英文版, 2002(2): 46-50.

[11] B. Mishra and A. Kertesz, ″The Use of MQTT in M2M and IoT Systems: A Survey, ″ in IEEE Access, vol. 8, pp. 201071-201086, 2020, doi: 10. 1109/ACCESS. 2020. 3035849.

[12] N. Tantitharanukul, K. Osathanunkul, K. Hantrakul, P. Pramokchon and P. Khoenkaw, ″MQTT-Topics Management System for sharing of Open Data, ″ 2017 International

Conference on Digital Arts, Media and Technology (ICDAMT), 2017, pp. 62-65, doi: 10. 1109/ICDAMT. 2017. 7904935.

[13] 褚孔统, 朱勇. 开发跨平台桌面应用的探讨[J]. 机电信息, 2019(33): 55-56. DOI: 10. 19514/j. cnki. cn32-1628/tm. 2019. 33. 030.

[14] 尤雨溪. Vue.js[OL]. https://vuejs.org/guide/introduction.html, 20220401

[15] posva. Pinia[OL].https://pinia.vuejs.org, 20220401.

[16] 景宇阳. 基于乐谱识别的深度学习算法作曲系统[D]. 南京艺术学院, 2020. DOI: 10. 27250/d. cnki. gnjyc. 2020. 000005.

[17] cuthbertLab. What is music21?[OL]. https://web.mit.edu/music21/doc/about/what.html. 20220401

[18] 刘华星,杨庚.HTML5——下一代 Web 开发标准研究[J].计算机技术与发展,2011,21(08):54-58+62.

[19] ESPRESSIF, ESP8266 A cost-effective and highly integrated Wi-Fi MCU for IoT applications[OL]. https://www.espressif.com.cn/en/products/socs/esp8266. 20220401

致谢

非常感谢朱金辉导师和助教在毕业设计论文工作过程中提供的帮助与指导，感谢家人，朋友和同学们在这几个月里的陪伴与支持。

这个课题非常有意思，在开发中一直都在感受着学习新知识与接触新领域的快乐，也思考并解决了一些问题和设计了一些方案。虽然在开发过程中有很多奇奇怪怪感觉不应该出现的 bug，但我总相信对于开发者而言，没有什么问题是不能被解决的。先贤们设计硬件，设计语言，设计系统，设计软件，每一项既有用途，也有限制。倘若被框架束缚住了，就打破框架，重新设计。

在论文工作中使得我学习了更多的相关技术。此次设计工作与我先前所熟悉的技术栈 Java，Spring Boot，数据库和服务器相关的开发知识相去甚远，但由于对未知事物的好奇心与对论文题目的兴趣，先后学习了桌面端开发技术 Electron，前端开发框架 Vue，机器人控制板 ESP8266，在阅读相关文档和编写样例代码后，随着开发的深入，又陆续接触与学习到 UI 框架 Element-plus，为了解析 MIDI 音乐格式深入了解格式编码，钢琴谱的阅读，midi.js 解析库和 music21 解析库的知识，为了客户端与本地脚本通信，学习了 Node.js 和 python、python 环境下的 Flask web 框架知识，并且由于客户端与机器人的连接需要 MQTT 通信，学习了 MQTT 协议相关知识和在服务器上安装 MQTT 服务器 EMQ X，还在机器人板子上编写 MicroPython 代码等等。在这个过程体验并尝试了很多不同的新技术，感觉非常充实。

最后的最后，希望疫情早日过去，人民幸福安康。