**分类号：TP311.1 U D C：D10621-408-(2024)1624-0**

**密 级：公 开 编 号：2020053021**

**成都信息工程大学**

**学位论文**

**基于在线相似度算法的音乐播放器的设计与实现**

|  |  |
| --- | --- |
| **论文作者姓名：** | **曹宇恒** |
| **申请学位专业：** | **数字媒体技术** |
| **申请学位类别：** | **工学学士** |
| **指导教师姓名（职称）：** | **叶斌（讲师）** |
| **论文提交日期：** | **2024年05月16日** |

**基于在线相似度算法的音乐播放器的设计与实现**

**摘要：**在数字化时代，流媒体和短视频平台兴起，但个性化音乐收听的需求也依然存在。为解决本地音乐库中不同风格歌曲带来的听歌体验不连贯的问题，本文提出了音频相似度分析的重要性，并且在音频相似度分析的基础上实现一款具有音频相似度排序功能的音乐播放器。

本次毕业设计基于在线相似度算法的音乐播放器，采用了Python作为编程语言，使用了Qt框架进行界面设计，并通过QSS样式对界面进行定制，并利用了音频库Librosa和数学库Pandas、Numpy、Sklearn来实现音频数据的解析和分析，采用了余弦相似度算法进行音频相似性分析，实现了音乐播放器的播放控制、播放列表、音频内核的主要功能，实现了音频特征的提取和相似度计算和排序功能。音乐播放器旨在通过完善的播放功能和先进的关于音频相似度分析的算法实现，为用户提供更为连续统一的收听体验。

**关键词：**Qt(PySide6)；音乐播放器；音频分析；余弦相似度算法；在线算法

**Design and Implementation of a Music Player Based on Online Similarity Algorithm**

**Abstract:** In the digital era, streaming media and short video platforms have risen, but the demand for personalized music listening still exists. To address the issue of inconsistent listening experiences caused by different styles of songs in local music libraries, this paper proposes the importance of audio similarity analysis and implements a music player with audio similarity sorting functionality based on audio similarity analysis.

This graduation project is based on a music player using online similarity algorithms, employing Python as the programming language. It utilizes the Qt framework for interface design and customizes the interface using QSS styles. The audio library Librosa and the math libraries Pandas, Numpy, and Sklearn were used to parse and analyze the audio data. The cosine similarity algorithm is adopted for audio similarity analysis, realizing the main functions of music player control, playlist management, and audio kernel. Additionally, it implements feature extraction, similarity calculation, and sorting functions. The aim is to provide users with a smoother listening experience through improved playback functionality and advanced algorithms for audio similarity analysis.

**Key words:** Qt(PySide6); Music Player; Audio Analysis; Cosine Similarity Algorithm; Online Algorithm

目 录

论文总页数：40页

[引 言 1](#_Toc16372)

[1 绪论 2](#_Toc30914)

[1.1 背景与意义 2](#_Toc32087)

[1.2 系统开发技术准备 2](#_Toc17168)

[1.2.1 Python 2](#_Toc5676)

[1.2.3 Qt(PySide6) 3](#_Toc5105)

[1.2.4 QSS 3](#_Toc18827)

[1.2.5 音频分析库Librosa 3](#_Toc8045)

[1.2.6 数据分析库Pandas、Numpy、Sklearn 3](#_Toc28516)

[1.2.7 音频播放库VLC 4](#_Toc10654)

[1.2.8 余弦相似度算法 4](#_Toc24515)

[1.2.9 在线算法 4](#_Toc9547)

[1.3 主要研究内容 4](#_Toc6523)

[1.4 论文组织结构 5](#_Toc14464)

[2 系统需求分析 6](#_Toc19177)

[2.1 需求概述 6](#_Toc24247)

[2.2 功能需求 6](#_Toc13793)

[2.2.1 音乐播放器本体 6](#_Toc23958)

[2.2.2 音乐播放器软件窗口 7](#_Toc28686)

[2.2.3 音频内核 7](#_Toc10317)

[2.2.4 音频特征获取 7](#_Toc19276)

[2.2.5 音频频谱分析 8](#_Toc28430)

[2.2.6 音频的余弦相似度分析 8](#_Toc19900)

[2.2.7 其他功能 9](#_Toc17219)

[2.3 非功能需求 9](#_Toc8108)

[3 系统设计与实现 11](#_Toc15974)

[3.1 系统整体架构 11](#_Toc17799)

[3.2 音乐播放器的设计与实现 11](#_Toc18601)

[3.2.1 音乐播放器本体 11](#_Toc28258)

[3.2.2 音乐播放器窗口 23](#_Toc17873)

[3.2.3 音频内核 25](#_Toc9190)

[3.3 音频分析模块的设计与实现 26](#_Toc25070)

[3.3.1 音频特征获取 26](#_Toc20140)

[3.3.2 音频频谱分析 26](#_Toc17628)

[3.3.3 音频的余弦相似度分析 27](#_Toc23105)

[3.4 其他功能的设计与实现 31](#_Toc27303)

[3.4.1 响度统一化 31](#_Toc24164)

[3.4.2 样式替换 31](#_Toc4177)

[4 疑难问题与解决方案 33](#_Toc8011)

[4.1 关于PySide6样式方案的问题 33](#_Toc24460)

[4.1.1 问题描述 33](#_Toc1259)

[4.1.2 解决方案 33](#_Toc6491)

[4.2 PySide6无法运行的问题 33](#_Toc27026)

[4.2.1 问题描述 33](#_Toc16067)

[4.2.2 解决方案 33](#_Toc25184)

[4.3 关于音频特征选取的问题 33](#_Toc28652)

[4.3.1 问题描述 33](#_Toc30805)

[4.3.2 解决方案 33](#_Toc15714)

[4.4 关于添加文件时窗口卡住的问题 34](#_Toc16149)

[4.4.1 问题描述 34](#_Toc17500)

[4.4.2 解决方案 34](#_Toc24213)

[4.5 关于在响度统一化标准在程序效率和实现上的问题 34](#_Toc30255)

[4.5.1 问题描述 34](#_Toc3907)

[4.5.2 解决方案 34](#_Toc11347)

[4.6 关于librosa对音频解析速度较慢的问题和关闭窗口后音频解析线程仍然继续的问题 35](#_Toc17518)

[4.6.1 问题描述 35](#_Toc30122)

[4.6.2 解决方案 35](#_Toc24316)

[4.7 关于选择余弦相似度算法在音频相似度分析的应用问题 35](#_Toc12008)

[4.7.1 问题描述 35](#_Toc21160)

[4.7.2 解决方案 35](#_Toc18253)

[结 论 36](#_Toc14946)

[参考文献 37](#_Toc23562)

[致 谢 39](#_Toc31882)

[声 明 40](#_Toc26642)

# 引 言

在当今数字化的时代，音乐已经成为人们日常生活中不可或缺的一部分，无论出行还是娱乐，音乐的陪伴都是不可或缺的重要内容。一个优秀的音乐播放器不仅能够提供丰富多样的音乐体验，还能够满足用户对音乐风格的个性化需求。对于用户来说，连续统一的收听体验，更能来给人们美妙的音乐享受。

目前市面上的音乐播放器，除了平台基于用户画像推送用户可能喜欢的歌曲的流媒体播放器外，本地播放器领域还没有出现能够排序相似音乐的音乐播放器。本文将在前人对于音乐分析和相似度研究系统的基础上，结合计算机学科的学科素养和个人大批量多风格的音乐收听经历，提出了一个基于相似度算法的音乐播放器的设计与实现方案。

# 1 绪论

## 1.1 背景与意义

在当今数字化时代，流媒体和短视频平台如雨后春笋般涌现，为用户提供了前所未有的内容消费体验。这些平台凭借先进的算法技术，对用户的观看行为、喜好、互动等进行深入分析，从而构建出精细化的用户个性化画像[1]。

与此同时，本地音乐作为一种独特的文化表达形式，具有浓厚的个人风格和情感色彩，以及不同时间时数据的差异性。但对于个人来说，以平台先进的算法技术等来构建用户个性化画像和当前的情感分析等内容的可行度不高。在这样的前提下，对于听歌风格范围较大的个人来说，在本地音乐收集过程中难免会因为数据量过大，以及听歌个人风格的转变和偏移，导致在一个本地音乐库中有各种风格迥异的歌曲，在收听时难免会因为风格差异过大，造成收听过程中情绪的波动和不统一。因此，在这种背景下，音频相似度分析显得尤为重要。通过对音频数据的深入挖掘和分析，识别出不同音频之间的相似性和差异性，再通过相似度对音乐进行排序，以此来获得更为连续、情感统一的收听体验。

## 1.2 系统开发技术准备

本项目采用Qt（PySide6）框架进行开发，使用QSS样式进行界面设计。代码编辑和编写主要在Visual Studio Code中完成。项目的编程语言是Python，用于音频文件的解析和分析是Librosa库，数据处理依赖于Pandas数据框架以及数学和算法库Numpy和Sklearn。

### 1.2.1 Python

Python是一种广泛使用的解释型、高级和通用的编程语言。Python支持多种编程范型，包括结构化、过程式、反射式、面向对象和函数式编程。它拥有动态类型系统和垃圾回收功能，能够自动管理内存使用，并且其本身拥有一个巨大而广泛的标准库。它的语言结构以及面向对象的方法，旨在帮助程序员为小型的和大型的项目编写逻辑清晰的代码[2]。Python的特点包括易于学习、易于阅读、易于维护、广泛的标准库、互动模式、可移植性、可扩展性、数据库支持、GUI编程能力和可嵌入性，这些特点使得Python成为一个非常灵活和强大的编程语言，适用于各种领域的开发，因此Python广泛应用于Web应用程序、软件开发、数据科学和机器学习的编程语言[3]。

1.2.2 Visual Studio Code

Visual Studio Code（简称VS Code）是一款由微软开发且跨平台的免费源代码编辑器。它以扩展的方式支持语法高亮、代码自动补全（IntelliSense）、代码重构功能，并内置了命令行工具和Git版本控制系统。用户可以通过内置的扩展程序商店安装其他扩展以拓展软件功能。VS Code通过Microsoft Python扩展提供了强大的Python编辑功能，支持自动完成、IntelliSense、代码格式化、签名帮助、重构、linting和类型提示等功能。这个扩展使得VS Code成为一个优秀的Python编辑器，适用于任何操作系统，并支持多种Python解释器[4]。

### 1.2.3 Qt(PySide6)

Qt是一个跨平台的应用程序开发框架，用于创建图形用户界面（GUI）以及跨平台应用程序，这些应用程序可以在各种软件和硬件平台上运行，如Linux、Windows、macOS、Android或嵌入式系统，几乎不需要改变底层代码库，同时仍然具有原生应用程序的原生能力和速度。Qt支持各种C++编译器，包括GCC和Clang C++编译器以及Visual Studio套件。它还支持其他语言的绑定或扩展，如通过Python绑定支持Python[5]。

PySide6是由The Qt Company开发的Python库，用于使用Qt工具包创建GUI应用程序。它是Python上Qt的官方绑定，提供了对Qt 6.0+框架的完整访问权限。PySide6提供了一个完整的Python界面，允许开发者使用Python语言来创建跨平台的桌面应用程序、移动应用程序和网络应用程序[6]。

### 1.2.4 QSS

QSS（Qt Style Sheets）是一种用于描述和定制Qt应用程序用户界面元素样式的语言。类似于CSS（Cascading Style Sheets），但QSS专门用于Qt应用程序的界面设计。QSS允许开发者通过简单的语法来定义窗口、菜单、按钮、表单和其他控件的外观属性，如颜色、边框、字体、背景图像等。这种方式使得界面设计更加集中化，便于维护和更新。[7]。

### 1.2.5 音频分析库Librosa

Librosa是一个强大的Python库，专注于音乐和声音研究，提供了处理音频文件的广泛工具和功能。Librosa支持多种与音频记录处理和提取相关的功能，包括从磁盘加载音频、注册不同类型的频谱描述符、分离和识别音调和脉动源、计算传统频谱衰减、处理时间-频域音频、序列建模、并行处理、节奏检测等。它还提供了音频信号的可视化工具，并支持多种信号处理方法来提取音频特征[8]。

### 1.2.6 数据分析库Pandas、Numpy、Sklearn

Pandas专门用于数据操纵和分析。它提供了数据结构和函数，可以用于高效地操作数据。Pandas的数据结构（Series和DataFrame）使得数据清洗、合并、分组和可视化变得非常容易[9]。

Numpy专注于提供高效的数组对象和数学函数。Numpy的核心是一个多维数组对象（即NumPy数组），它提供了与Python内置数据类型相同的接口，但具有更高的性能和更多的功能。Numpy还提供了大量的数学函数，用于执行向量化运算，这意味着可以对整个数组进行操作，而不是逐个元素地进行操作[10]。

Scikit-learn（简称sklearn）是一个用于机器学习的Python库，它提供了大量的算法实现，包括分类、回归、聚类和降维等。sklearn库也提供了一些与机器学习相关的基础算法，便于开发者和研究人员从基础开始构建需求的实现[11]。

### 1.2.7 音频播放库VLC

VLC是一个免费且开源的跨平台媒体播放器软件和流媒体服务器，由VideoLAN项目开发。VLC支持多种桌面操作系统和移动平台。

VLC Python库是VLC媒体播放器的Python绑定，允许开发者使用Python编程语言来控制VLC媒体播放器。通过VLC Python库，开发者可以在Python脚本中播放视频、音频、图片等媒体文件，也可以控制媒体播放器的各种设置，如调整音量、暂停、快进等[12]。

### 1.2.8 余弦相似度算法

余弦相似性通过测量两个向量的夹角的余弦值来度量它们之间的相似性。余弦相似度通常用于正空间，因此给出的值为0到1之间。

两个向量间的余弦值可以通过使用欧几里得点积公式求出：

给定两个属性向量， A 和B，其余弦相似性θ由点积和向量长度给出，如下所示：

这里的分别代表向量的各分量。

给出的相似性范围从-1到1。-1意味着两个向量指向的方向正好截然相反，1表示它们的指向是完全相同的，0通常表示它们之间是独立的，而在这之间的值则表示中间的相似性或相异性。

在信息检索的情况下，由于一个词的频率（TF-IDF权）不能为负数，所以这两个文档的余弦相似性范围从0到1。并且，两个词的频率向量之间的角度不能大于90°[13]。

### 1.2.9 在线算法

在计算机科学中，在线算法是一种处理输入资料的独特形式，其演算过程中并不要求所有输入资料在算法开始运始之一刻即完备，反而可对逐步输入的资料加以处理并在输入完最后一项资料之后输出运算结果[14]。本文使用的插入排序就是经典的在线算法，插入排序在工业级库中也有着广泛的应用，在STL的sort算法和stdlib的qsort算法中，都将插入排序作为快速排序的补充，用于少量元素的排序，一般来说量级小于千，插入排序都还是不错的选择。

## 1.3 主要研究内容

本项目主要研究为以下内容的设计思路与实现方法：

1. 音乐播放器的核心功能：如播放及相关操作，播放列表和文件路径管理。
2. 音频分析相关内容：如音频特征获取，音频频谱分析，音频的余弦相似度分析和排序。
3. 其他功能和扩展功能：如响度统一化，样式替换。

## 1.4 论文组织结构

本文总共分为四个章节，每个章节的大概内容如下所述：

第一章是论文引言部分，整体阐述课题的研究背景与意义，介绍系统开发所需的技术准备，点明主要研究内容，引出后文的论述。

第二章是需求分析部分，从该系统的功能需求和非功能需求两方面分别进行介论述，其中功能需求模块主要介绍系统的整体框架、项目预期，以及细分各个模块的具体需求，非功能需求则主要从系统的性能需求、稳定性需求、图形需求、扩展性需求、安全性需求展开论述。

第三章主要阐述本系统的设计与实现细节，详细介绍各个功能模块的设计思想和实现方法、重要功能的核心逻辑、算法的研究与应用等。

第四章为疑难问题记录部分，介绍开发和研究过程中遇到的疑难问题与解决方案。

# 2 系统需求分析

## 2.1 需求概述

需求分析是软件开发过程中的重要一环，为项目的设计、开发和实施提供了指导。在设计音乐播放器系统时，需要对系统的功能和性能需求进行明确定义，以确保系统能够满足期望和需求。

对于音乐播放器，需要具备音乐播放器的常规功能：提供完善的音频播放操作功能，包括播放、暂停、停止、上一首、下一首、随机播放等。具备播放列表管理功能，包括添加歌曲、添加文件夹、删除歌曲、添加播放列表、删除播放列表等操作。设计简洁明了的用户界面，方便用户进行操作和了解当前播放状态和软件外观。本音乐播放器还需要具备以下功能：提供音频特征获取和频谱分析功能，以支持基于相似度的播放列表排序和响度统一化等功能。实现音频的余弦相似度分析，用于对相似的歌曲进行排序并生成基于余弦相似度排序的播放列表。

## 2.2 功能需求

本节主要针对系统的功能需求展开介绍。

### 2.2.1 音乐播放器本体

音乐播放器本体是整个音乐播放器系统的核心模块，负责管理音频文件的播放、播放列表的管理、用户界面的交互以及整体的功能实现。音乐播放器本体应当拥有以下功能：

1. 音乐播放器本体应该提供完善的音频播放操作功能，包括播放、暂停、停止、上一首、下一首、随机播放、随机播放上一首返回的操作。播放操作要求可以通过双击播放列表内歌曲进行对应歌曲的播放操作。播放和暂停操作要求可以在音乐播放器本体界面上得到按键变化的反馈。下一首有两种模式，顺序播放下一首和随机播放下一首，在随机下一首播放时保存最近播放历史共五首，供随机播放时上一首使用。上一首有两种模式，顺序播放上一首和随机播放上一首，在随机播放上一首是，如果有随机播放下一首时保存的最近播放历史，需要先在最近播放历史中倒序顺序播放上一首，在没有随机播放下一首时保存的最近播放历史时，随机播放上一首。随机播放通过界面按键切换当前播放状态为顺序播放或随机播放。
2. 用户应该可以进行播放列表管理。音乐播放器本体应该提供方便的播放列表管理功能，包括添加歌曲、添加文件夹、删除歌曲、添加播放列表、删除播放列表的操作，同时本音乐播放器还需要有基于相似度排序的播放列表页。添加歌曲要求弹出文件窗口，供用户选择需要添加的单个音频文件。添加文件夹需要扫描用户选择文件夹中，包含的所有本音乐播放器指定的音频格式，并且包含文件夹内的子文件夹。删除歌曲需要在用户右键播放列表中歌曲并点击删除后，删除在该播放列表中的音频路径和特征信息记录。添加播放列表需要音乐播放器本体在本地生成新的播放列表文件，并且要求本音乐播放器可以识别用户通过系统文件系统直接添加到播放列表文件夹内的本音乐播放器生成的播放列表。删除播放列表需要删除在本地的播放列表文件。在添加歌曲或添加文件夹时，会同时向基于相似度排序的播放列表页添加歌曲信息。
3. 音乐播放器本体的用户界面应该设计简洁明了，包括播放/暂停、上一首、下一首、停止、播放模式、播放列表显示、添加文件、添加文件夹、新建播放列表、切换软件样式，以便用户能够方便地进行操作和了解当前播放状态和软件外观。为了用户体验，音乐播放器本体应该响应用户的操作及时，并提供友好的提示和反馈。例如音量控制、音乐播放进度条、随机播放列表页面跳转等，以便用户了解当前播放状态。

### 2.2.2 音乐播放器软件窗口

音乐播放器窗口软件要求用户可以进行窗口的拖拽，窗口大小的调整，最小化，最大化和还原，关闭窗口等操作。其中在实现音乐播放器窗口的关闭功能时，确保正确关闭所有子线程是非常重要的。子线程可能在执行音频播放、数据处理或其他后台任务时运行，如果不正确关闭，可能会导致资源泄漏、数据不一致或应用程序崩溃。另外在Qt的FramelessWindowHint的情况下，QMainWindow无法直接使用Windows窗口功能，因此上述功能和操作需要在编写音乐播放器软件窗口时手动实现。

### 2.2.3 音频内核

音频内核是音乐播放器的核心引擎，负责处理音频文件的解码、播放控制以及音频数据的处理。音频内核应当支持常见音频格式的解码，包括MP3、WAV、FLAC格式。同时，应当受控制于音乐播放器本体的播放控制功能，如播放、暂停、停止、上一首、下一首等，本音乐播放器中还需要能够接收来源于音乐播放器响度统一化信息，来对音量进行调整。

### 2.2.4 音频特征获取

音频特征获取是本音乐播放器系统的重要组成部分，它负责从音频文件中提取有用的特征信息，如音调、节奏、响度等，以支持播放列表的基于相似度排序、响度统一化等功能。音频特征获取模块应当具备以下功能：支持多种音频格式的特征提取，包括常见的MP3、WAV、FLAC格式。提供丰富的音频特征信息，包括但不限于频谱带宽（Spectral Bandwidth）、频谱对比度[15]（Spectral Contrast）、BPM（音频文件每分钟节拍数）等。这些特征信息应当能够反映音频文件的主要特征，以支持后续的播放相似度排序列表。音频特征获取模块应当能够调用计算音频文件与标准值之间的相似度的模块，基于这些相似度信息对播放列表进行排序。提供API接口，供音乐播放器本体和其他系统组件调用。这些API接口应当能够返回音频文件的特征信息，以支持播放列表的动态更新和用户的个性化推荐。支持音频特征的实时更新。音频特征获取模块应当能够在音频文件播放过程中实时更新特征信息，以支持实时的播放控制和用户体验。音频特征获取模块应当具备良好的性能和稳定性，能够在多核处理器环境下高效运行，并且在处理大规模音频文件时能够保持良好的性能。

### 2.2.5 音频频谱分析

音频频谱分析是本音乐播放器系统音频分析模块的核心功能之一，负责从音频信号中提取频谱信息，以支持音频的分析、处理和特征提取。该模块应具备以下特性：支持多种音频格式（如MP3、WAV、FLAC）的频谱分析，自动识别音频格式并分析音频频谱。支持实时的音频数据处理，确保音频处理运算不与音乐播放器播放等主进程冲突导致软件卡顿甚至卡死。提供梅尔频谱系数（Mel Spectrogram）和dB量表（Decibel Scale）的频谱信息。通过对梅尔频谱的分析和处理，可以提取出音频信号中的各种特征，如音调、音色等。这些特征可以用于语音识别、音乐信息检索、音频水印等领域[15]。音频特征提取部分要求音频频谱模块返回一个数表示需要分析的频谱的特征，所以需要计算梅尔频谱和基于梅尔频谱运算得到的dB量表的熵和标准差。其中熵可以帮助理解梅尔频谱中的不确定性或随机性，从而衡量音乐的复杂性和多样性。对于那些具有高度变化和多样性的音乐结构，熵可以提供一个直观的度量。标准差可以帮助理解梅尔频谱的稳定性和变化程度。对于那些具有明显节奏和和声进行变化的音乐，标准差可以提供一个直观的度量，标准差还可以反映音乐的动态范围，即音量的变化范围。这对于理解音乐的整体音量和强度非常有用。

### 2.2.6 音频的余弦相似度分析

音频的余弦相似度分析是音乐播放器系统中用于衡量不同音频文件和标准值之间相似度的关键功能。通过计算由音频特征获取模块和音频频谱分析模块获得的音频特征与标准值的余弦相似度，来支持播放列表的基于相似度排序功能。支持对不同音频特征的重要性进行权重调整，通过调整特征的权重，可以更准确地反映音频文件之间的相似度。

关于余弦相似度算法在音频相似度的应用，MUSIC RECOMMENDATION SYSTEM BASED ON COSINE SIMILARITY AND SUPERVISED GENRE CLASSIFICATION[16]文中通过计算用户听歌历史与音乐库中其他歌曲的余弦相似度，推荐与用户喜好相似的歌曲。文中首先对于音乐库中的每首歌曲，提取相应的音乐特征，并将其表示为特征向量。音乐特征包括音频特征（如MFCC（Mel频谱系数）、频谱特征等）和元数据（如歌曲流派、歌手、专辑等），然后对于每个用户，将其听歌历史中的歌曲表示为特征向量。然后，计算用户听歌历史中每首歌曲与音乐库中其他歌曲的余弦相似度，最后得到一个用户听歌历史中每首歌曲与音乐库中其他歌曲的相似度得分列表。在本音乐播放器中，采用了类似的方法来实现音频的余弦相似度分析。首先，从音频文件中提取了多种音频特征，包括频谱特征、BPM和MFCC的处理后数据。这些特征被用来构建每个音频文件的特征向量。接下来，使用余弦相似度算法来计算音频文件与标准值之间的相似度。这里的标准值可以认为代表了十分安静的经典音乐。在实现音频的余弦相似度分析时，还引入了特征权重的概念。通过调整特征的权重，可以更灵活地控制哪些特征对最终的相似度得分有更大的影响。最后通过上面的特征和权重获得一个相对于标准值的相似度得分列表。

### 2.2.7 其他功能

在音乐播放器系统中，除了核心功能如音频播放、播放列表管理、音频频谱分析和余弦相似度分析外，还包含了一些其他重要的功能，以提升用户体验和系统的功能完善度。

响度统一化：响度统一化是指将不同音频文件的音量调整到一个统一的水平，以确保用户在播放不同音质或音量的音频文件时，能够获得一致的音量体验。通过响度统一化，可以减少音量调整的需要，提升用户的使用体验。实现响度统一化的关键在于对音频文件进行扫描，利用RMS音频响度测量算法获得最大响度，在播放时对于该最大响度在音频内核内自动调整响度，使得所有音频文件在播放时具有相同的音量级别。

样式替换：样式替换功能允许用户根据自己的喜好或需要，动态更改音乐播放器的外观和界面样式。用户可以选择不同的主题，以适应自己的审美需求或提高使用体验。用户还可以通过自行编写的QSS文档来对本音乐播放器的样式进行个性化调整。

## 2.3 非功能需求

本章节主要对本系统的非功能需求展开阐述，共分为以下几个要点：

（一）性能需求

本软件的性能需求主要集中在数据处理能力和时间特性上。考虑到这是一个个人本地使用的音乐播放器，其数据处理能力主要体现在音频文件的加载和播放上。软件应能够快速加载音频文件，并在播放过程中保持流畅的播放体验。此外，软件还需要处理用户的播放列表，包括添加、删除音频路径和播放列表。这些操作应具有较低的响应时间，并且不在主线程内执行，以确保用户体验的流畅性。

时间特性方面，软件在启动时应能快速加载用户的播放列表，并在用户操作（如播放、暂停、停止等）时快速响应。数据的转换和传送时间应尽可能短，以避免用户在播放过程中出现延迟。

（二）稳定性需求

本软件应具备高稳定性，以确保用户在使用过程中不会遇到意外的崩溃或错误。软件在一个运行周期内，应能够稳定运行，不会出现频繁的崩溃或性能下降。在一定的压力条件下，如处理大量音频文件，软件也应能够保持良好的性能，不会出现明显的性能劣化趋势。

（三）图形需求

界面设计应保持简洁，避免过于复杂的布局和过多的信息干扰用户。界面设计应保持视觉一致性，包括颜色方案、字体、图标等元素的一致性。

1. 扩展性需求

本软件应当具有高扩展性，在开源的基础上，应当能较为方便的添加用户需求的音频文件格式，添加需求的样式文件。

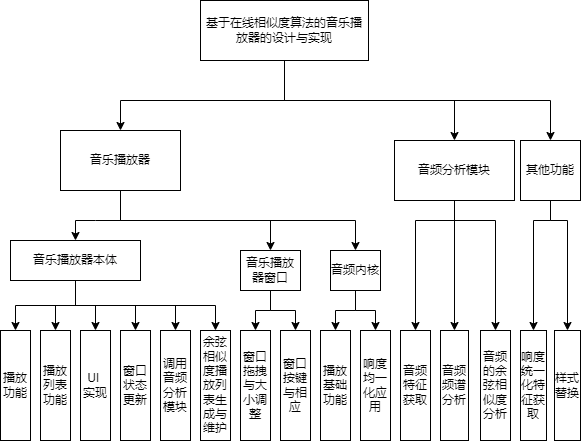
1. 安全性需求

本软件应当具有高安全性，软件不会对本地文件写相关的操作，保护本地文件的安全性。

3 系统设计与实现

## 3.1 系统整体架构

本系统总共分为以下三大模块，七个小模块，系统的整体架构框图如图3.1所示。

图3.1 系统整体架构图

## 3.2 音乐播放器的设计与实现

在基于在线相似度算法的音乐播放器中，音乐播放器部分是最核心的内容，包括音乐播放器本体，音乐播放器窗口和音频内核。音乐播放器本体包含了播放功能、播放列表功能、窗口状态更新、UI实现、音频内核调用、音频分析模块调用、余弦相似度播放列表的生成与维护功能。音乐播放器窗口包含了窗口拖拽与大小调整、窗口按键与相应功能。音频内核模块包括播放基础功能和响度均一化应用功能。

### 3.2.1 音乐播放器本体

在模块化编程的情况下，音乐播放器本体是整个音乐播放器系统中的一个模块。在PySide6中，QWidget是Qt中所有用户界面对象的基类，它提供了基本的绘图和事件处理能力，同时也是构建用户界面的基本单元[5]。音乐播放器本体基于QWidget构建，并且记录下基于QMainWindow的主窗体main.py。在初始化该模块时，需要初始化当前播放列表self.current\_playlist\_filename，播放模式self.playback\_mode，随机播放上五首列表self.last\_five\_tracks，音频特征范围self.ranges。声明实例变量后，加载QSS样式和声明音频内核AudioCore()实例，运行成员函数加载播放列表self.load\_existing\_playlists()和成员函数加载上一次关闭时打开的播放列表函数self.load\_last\_opened\_playlist()，上述操作执行完成后，创建一个新的线程执行成员函数更新余弦相似度列表self.scan\_and\_update\_cosine\_similarity\_list()。相关成员函数的详细操作内容在分功能中描述。

在这里介绍Qt的特色机制。在PySide6中，信号（Signals）和槽（Slots）是实现事件驱动编程的核心机制。它们允许开发者在不需要编写大量代码的情况下实现复杂的功能。信号是用于在对象之间传递信息，而槽是用于响应信号的槽函数。这种机制极大地简化了事件处理的编写，使得代码更加清晰和易于维护。在PySide6中，信号是类的属性，可以在类中定义。每个信号都有一个唯一的名称，并且可以有任意数量的参数。信号通过connect()方法连接到槽。连接可以在对象的构造函数中完成，也可以在运行时动态完成。信号和槽的使用使得代码更加模块化，易于阅读和维护。每个功能都可以封装在一个槽中实现，通过信号进行连接。通过使用信号和槽，开发者可以更专注于实现应用逻辑，而不是事件处理机制[4]。例如，用户界面上的按钮点击可以直接连接到播放控制的槽，而不需要编写额外的事件处理代码。以文中停止歌曲播放为例，通过信号和槽机制，音乐播放器本体调用音频内核内的方法，样例代码如下：

（一）播放功能

*self.stop\_button = QPushButton("Stop")*

*self.control\_box.addWidget(self.stop\_button)*

*self.stop\_button.clicked.connect(self.audio\_player.stop)*

音乐播放器本体的播放功能包含播放与暂停、调用音频内核、停止、上一首、下一首、播放模式（顺序播放或随机播放）、音量调整、进度条及其调整。

播放功能是主要调用音频内核的部分，当用户选择播放某个音频文件时，播放功能会调用音频内核模块来加载并播放该文件。在播放过程中，播放功能还提供了丰富的控制选项，如播放、暂停、停止、上一首、下一首等。这些控制选项通过用户交互界面呈现给用户，用户可以通过点击按钮来触发相应的操作。播放功能还支持播放模式的切换，如顺序播放、随机播放等，以满足用户不同的播放需求。音量调整允许用户根据自己的喜好调整音频的播放音量，以获得更舒适的听觉体验。进度条则显示了音频的播放进度，用户可以通过拖动进度条来快速定位到音频的任意位置，实现快速跳转和回放。

用户以任何形式开始播放音乐时，在系统中包括双击播放列表中音乐、下一首、上一首，音乐播放器本体会运行成员函数play\_audio(self, item)，其中item为QListWidgetItem类型，play\_audio(self, item)从item中获取歌曲名，读取当前播放列表的json文件，通过文件名在json文件中寻找歌曲路径，在找到文件路径后，将路径和音频响度为参数调用实例化音频内核的打开文件方法audio\_player.open\_file(file\_path, song\_info['loudness'])，后续播放操作由音频内核完成。代码实现如下：

*def play\_audio(self, item):*

*file\_name = item.text()*

*# 读取JSON文件获取歌曲地址*

*playlists\_dir = os.path.join(self.mainWindow.baseDir, 'playlists')*

*playlist\_json\_path = os.path.join(playlists\_dir, self.current\_playlist\_filename)*

*with open(playlist\_json\_path, 'r', encoding='utf-8') as f:*

*playlist\_info = json.load(f)*

*for song\_info in playlist\_info:*

*if song\_info["name"] == file\_name:*

*file\_path = song\_info["path"]*

*break*

*else:*

*print(f"Song {file\_name} not found in playlist.")*

*return*

*self.audio\_player.open\_file(file\_path, song\_info['loudness'])*

用户在点击播放与暂停按键时，音乐播放器本体运行成员函数toggle\_play\_pause(self)，通过音频内核传回的播放状态，调用实例化音频内核的播放切换函数audio\_player.play\_pause()以及更新按钮状态。代码实现如下：

用户点击停止按钮时，直接通过Qt的Signals & Slots机制，调用音频内核的audio\_player.stop()函数停止音乐播放。

*self.play\_button = QPushButton("Play", self)*

*self.play\_button.clicked.connect(self.toggle\_play\_pause)*

*self.control\_box.addWidget(self.play\_button)*

*def toggle\_play\_pause(self):*

*if self.audio\_player.is\_playing():*

*self.audio\_player.play\_pause()*

*self.play\_button.setText("Play")*

*else:*

*self.audio\_player.play\_pause()*

*self.play\_button.setText("Pause")*

在用户点击上一首或下一首按钮时，需要根据顺序播放和随机播放的不同情况加以区分。上一首和下一首的功能受实例变量self.playback\_mode的影响，其中，变量值为"sequential"表示顺序播放，变量值为"random"表示随机播放。在随机播放模式下，需要利用实例变量队列self.last\_five\_tracks用于存储最近五首播放历史。当用户点击下一首按钮时，音乐播放器本体会执行成员函数play\_next(self)。若self.playback\_mode的值为"sequential"则按照顺序播放，首先获取当前播放列表的索引，然后通过计算得到下一首歌曲的索引next\_index，并实现顺序播放和循环播放，利用该索引更新界面状态中的当前播放歌曲，并通过调用成员函数play\_audio(self,item)进行音频播放。当self.playback\_mode的值为其他情况（即"random"），首先获取当前播放歌曲的名称，并将歌曲名称入队实例变量队列self.last\_five\_tracks，并检查队列长度。若长度大于5，则将最早加入的歌名出队列，接着获取0到当前播放列表长度-1的随机整数作为下一首的播放索引random\_index，利用该索引更新界面状态中的当前播放歌曲，并调用成员函数play\_audio(self,item)进行音频播放。代码实现如下：

用户点击上一首按钮时，音乐播放器本体会执行成员函数play\_previous(self)。在"sequential"模式下，则按照顺序播放。首先获取目前在播放列表的索引current\_index，然后通过计算得到上一首歌曲的索引previous\_index。根据该索引更新界面状态中的当前播放歌曲，并调用成员函数play\_audio(self,item)进行音频播放。在其他模式（即"random"）下，首先检查self.last\_five\_tracks是否为空或长度大于5。若是，则不应当按照播放历史来进行上一首播放，清空self.last\_five\_tracks，然后获取0到当前播放列表长度-1长度的随机整数作为下一首的播放索引random\_index。更新界面状态中的当前播放歌曲，并调用成员函数play\_audio(self, item)进行音频播放。若self.last\_five\_tracks不为空且长度不大于5，即需要倒序按照播放记录播放上一首，从self.last\_five\_tracks中出队最后一个添加的歌曲名称。通过遍历self.playlist\_widget的所有项，找到与最后一个添加的歌曲名称匹配的项的索引，然后更新界面状态中的当前播放歌曲，并调用成员函数play\_audio(self,item)进行音频播放。代码实现如下：

*self.next\_button = QPushButton("Next", self)*

*self.next\_button.clicked.connect(self.play\_next)*

*self.control\_box.addWidget(self.next\_button)*

*def play\_next(self):*

*if self.playback\_mode =='sequential':*

*current\_index = self.playlist\_widget.currentRow()*

*next\_index = (current\_index + 1) % self.playlist\_widget.count()*

*self.playlist\_widget.setCurrentRow(next\_index)*

*self.play\_audio(self.playlist\_widget.item(next\_index))*

*else:*

*current\_track = self.playlist\_widget.currentItem().text()*

*self.last\_five\_tracks.append(current\_track)*

*if len(self.last\_five\_tracks) > 5:*

*self.last\_five\_tracks.pop(0)*

*random\_index = random.randint(0, self.playlist\_widget.count() - 1)*

*self.playlist\_widget.setCurrentRow(random\_index)*

*self.play\_audio(self.playlist\_widget.item(random\_index))*

用户点击随机/顺序切换按钮时，音乐播放器本体运行成员函数toggle\_playback\_mode(self)，通过实例变量识别当前的播放模式，根据当前播放模式更新播放模式成员变量，通过QPushButton的setText方法更新按钮状态。

*def play\_previous(self):*

*if self.playback\_mode == 'sequential':*

*current\_index = self.playlist\_widget.currentRow()*

*previous\_index = (current\_index - 1) % self.playlist\_widget.count()*

*self.playlist\_widget.setCurrentRow(previous\_index)*

*self.play\_audio(self.playlist\_widget.item(previous\_index))*

*else:*

*if not self.last\_five\_tracks or len(self.last\_five\_tracks) > 5:*

*self.last\_five\_tracks.clear()*

*random\_index = random.randint(0, self.playlist\_widget.count() - 1)*

*self.playlist\_widget.setCurrentRow(random\_index)*

*self.play\_audio(self.playlist\_widget.item(random\_index))*

*else:*

*last\_track = self.last\_five\_tracks.pop()*

*for i in range(self.playlist\_widget.count()):*

*if self.playlist\_widget.item(i).text() == last\_track:*

*self.playlist\_widget.setCurrentRow(i)*

*self.play\_audio(self.playlist\_widget.item(i))*

*break*

用户在拖动音量音量调整时，在volumeslider的数值发生变化时，直接通过Qt的Signals & Slots机制，调用音频内核的set\_volume方法进行音量调整[4]。

进度条及其调整，在positionslider的数值发生变化时，直接通过Qt的Signals & Slots机制，调用音频内核的set\_position方法进行进度调整。

1. 播放列表功能

音乐播放器本体的播放列表功能包括加载播放列表、保存播放列表、读取现有播放列表、打开上一次关闭时打开的播放列表、打开播放列表选择器中选择的播放列表、更新播放列表选择器、添加新的播放列表、音频文件添加到播放列表、添加文件夹、添加文件、更新余弦相似度播放列表数据、更新所有播放列表音频响度、记录音频特征范围、基于余弦相似度的插入排序、删除选择项目。

加载播放列表为该功能部分的基础功能，需要通过播放列表文件名，在音乐播放器本体的相对路径下，通过系统os库获取播放列表文件，在有该选择文件的情况下，加载json播放列表文件，更新播放列表UI内playlist\_widget列表元素[2]。该功能会在选择播放列表、打开上一次关闭时打开的播放列表、添加新的播放列表时使用到。代码实现如下：

*def load\_playlist(self, filename):*

*playlists\_dir = os.path.join(self.mainWindow.baseDir, 'playlists')*

*playlist\_json\_path = os.path.join(playlists\_dir, filename)*

读取现有播放列表通过在音乐播放器本体的相对路径下的文件夹playlists内扫描所有可读取到的json文件来添加播放列表，读取到的文件名全部添加到播放列表选择器中，并更新UI内playlist\_selector列表元素。在该文件夹中，除了播放列表还存在一个保存上一次打开的播放列表的后缀为.txt的文件，因此需要筛选json文件读取。

*if os.path.exists(playlist\_json\_path):*

*with open(playlist\_json\_path, 'r', encoding='utf-8') as f:*

*playlist\_info = json.load(f)*

*self.playlist\_widget.clear()*

*for song\_info in playlist\_info:*

*self.playlist\_widget.addItem(song\_info["name"])*

*else:*

*print(f"Playlist file {filename} not found.")*

打开上一次关闭时打开的播放列表需要在音乐播放器本体初始化时运行，读取文件夹playlists中的文件lastOpenedPlaylist.txt中存储的文件名，打开上一次关闭时打开的播放列表，其中上一次播放的文件名在音乐播放器本体的closeEvent中更新。在PySide6中，closeEvent是一个重写的事件处理函数，用于在窗口或对话框关闭时执行自定义的清理任务。在窗口关闭时，通常需要执行一些清理任务，这里功能要求在closeEvent中更新lastOpenedPlaylist.txt，保存关闭时当前打开的播放列表。

打开播放列表选择器中选择的播放列表通过选择列表元素的文本，在播放列表文件中搜索同名文件，并调用成员函数load\_playlist(playlist\_name)读取同名播放列表。

更新播放列表选择器需要在添加播放列表时调用，从文件夹中搜索所有播放列表，清空并添加到playlist\_selector中。

添加新的播放列表在用户点击添加播放列表按钮时调用，首先通过QInputDialog弹出对话框，要求用户输入新播放列表名，在用户输入并点击确认后，在播放列表文件夹中添加以下用户输入名称加后缀.json的文件，并且添加json需要的格式内容“[]”，最后调用更新播放列表选择器和加载播放列表加载该新的播放列表。

音频文件添加到播放列表是该部分比较重要的功能，在添加文件夹和添加文件时需要调用此功能。在音频文件添加到播放列表时，需要传入文件的路径和文件名，在功能内先获取到目前的播放列表的json文件，读取到该播放列表信息playlist\_info中，确保文件路径在playlist\_info不存在防止重复添加文件，完成上述操作后，添加音频文件到播放列表文件信息，并且与预先设置好后续需要处理的数据类型，将播放列表文件信息更新到播放列表文件。在上述操作完成后，还需要更新余弦相似度播放列表，和上述进行相同的操作，更新到歌曲信息到余弦相似度播放列表。代码实现如下：

添加文件夹因为涉及到大批量文件处理和大批量音频文件的特征提取和处理，为了避免主进程卡死，影响用户在添加文件和对音频文件进行提取时使用音乐播放器播放和其他功能，该部分使用多线程的方式处理。用户点击添加文件夹按钮后，调用成员函数select\_folder(self)，首先使用QFileDialog的getExistingDirectory()，让用户选择需要添加的文件夹，在确认用户选择文件夹路径后，检查主进程中的多线程标志self.mainWindow.thread\_stop\_flag，如果主进程没有更新该标志通知子线程随主进程退出任务时，开启一个线程运行成员函数process\_folder(self,folder\_path)。在process\_folder(self,folder\_path)中，功能主要分为两部分，添加文件和开启新的子线程用于更新余弦相似度播放列表数据。第一部分功能添加文件，首先通过os.walk(folder\_path)遍历该文件夹和该文件夹中的子文件夹获得所有文件，对于其中后缀为'.mp3', '.wav', '.flac'的文件，通过调用音频文件添加到播放列表成员函数添加到当前播放列表和余弦相似度播放列表中。第二部分功能开启新的子线程用于更新余弦相似度播放列表数据会在第一部分将文件全部添加完成后开始运行，开启新的线程运行成员函数scan\_and\_update\_cosine\_similarity\_list(self)，该函数详细内容在更新余弦相似度播放列表数据中描述。代码实现如下：

*def add\_to\_playlist(self, file\_path, file\_name):*

*# current playlist*

*playlists\_dir = os.path.join(self.mainWindow.baseDir, 'playlists')*

*playlist\_json\_path = os.path.join(playlists\_dir, self.current\_playlist\_filename)*

*with open(playlist\_json\_path, 'r', encoding='utf-8') as f:*

*playlist\_info = json.load(f)*

*for song in playlist\_info:*

*if song["path"] == file\_path:*

*print(f"File {file\_name} is already in the playlist.")*

*return*

*playlist\_info.append({*

*"name": file\_name,*

*"path": file\_path,*

*"spectral\_bandwidth": None,*

*"spectral\_contrast": None,*

*"bpm": None,*

*'wav\_entropy': None,*

*'wav\_std\_dev': None,*

*"loudness": None,*

*"play\_count": 0,*

*"weighted\_cosine\_similarity": None,*

*})*

*with open(playlist\_json\_path, 'w', encoding='utf-8') as f:*

*json.dump(playlist\_info, f, ensure\_ascii=False, indent=4)*

*self.playlist\_widget.addItem(file\_name)*

*# 添加到余弦相似度播放列表相似部分略*

添加文件与添加文件夹同理，调用音频文件添加到播放列表来实现该部分功能。

*def select\_folder(self):*

*# 开多线程*

*folder\_path = QFileDialog.getExistingDirectory(self, "Select Folder")*

*if folder\_path:*

*if not self.mainWindow.thread\_stop\_flag:*

*threading.Thread(target=self.process\_folder, args=(folder\_path,)).start()*

*def process\_folder(self, folder\_path):*

*# 遍历文件夹中的所有文件，并添加到播放列表中*

*total\_files = sum([len(files) for \_, \_, files in os.walk(folder\_path)])*

*processed\_files = 0*

*for root, dirs, files in os.walk(folder\_path):*

*for file in files:*

*if file.endswith(('.mp3', '.wav', '.flac')): # 根据需要添加更多文件类型*

*file\_path = os.path.join(root, file)*

*self.add\_to\_playlist(file\_path, file)*

*processed\_files += 1*

*print(f"Processed {processed\_files} of {total\_files} files.")*

*print("Folder processing completed.")*

*threading.Thread(target=self.scan\_and\_update\_cosine\_similarity\_list).start()*

更新余弦相似度播放列表数据在打开音乐播放器初始化时和添加文件或文件夹后运行。首先扫描余弦相似度播放列表，对该播放列表信息进行遍历，在确认主进程多线程停止信号mainWindow.thread\_stop\_flag不为真后，检索任意任一歌曲特征为None的歌曲信息，通过调用获取音频特征和计算余弦相似度内核，测量并填充频谱带宽spectral\_bandwidth、频谱对比度spectral\_contrast、每分钟节拍数bpm、梅尔频谱熵wav\_entropy、梅尔频谱标准差wav\_std\_dev、最大响度loudness、音频文件与标准值的余弦相似度weighted\_cosine\_similarity以上歌曲信息[14][16][19]，将这些信息写回余弦相似度播放列表，通过调用更新所有播放列表音频响度同步在其他播放列表中的响度信息，接着运行sort\_cosine\_similarity\_list()对已经有测量数据的音频信息，使用插入排序进行余弦相似度排序，最后记录音频特征范围[13]。其中为了在调用余弦相似度内核的需求，需要使用Pandas库的DataFrame将信息封装并送入余弦相似度测量模块。代码实现如下：

更新所有播放列表音频响度在更新余弦相似度播放列表数据时调用。在更新余弦相似度播放列表数据部分会获得音频的最大响度信息，通过对播放列表文件夹内所有播放列表包含和当前处理音频文件路径相同的数据，更新其最大响度信息，在播放时通过该信息实现响度统一化。

*def scan\_and\_update\_cosine\_similarity\_list(self):*

*playlists\_dir = os.path.join(self.mainWindow.baseDir, 'playlists')*

*cosine\_similarity\_list\_path = os.path.join(playlists\_dir, 'cosineSlimilarityList.json')*

*with open(cosine\_similarity\_list\_path, 'r', encoding='utf-8') as f:*

*cosine\_similarity\_list = json.load(f)*

*updated\_songs = []*

*for song in cosine\_similarity\_list:*

*if self.mainWindow.thread\_stop\_flag == False:*

*if song["bpm"] is None or song["spectral\_bandwidth"] is None or song["spectral\_contrast"] is None or song["weighted\_cosine\_similarity"] is None:*

*print(f"Updating {song['name']}")*

*features = get\_song\_features(song["path"])*

*features\_df = pd.DataFrame(features, index=[0])*

*# 更新歌曲信息*

*song["spectral\_bandwidth"] = features['spectral\_bandwidth']*

*song["spectral\_contrast"] = features['spectral\_contrast']*

*song["bpm"] = features['bpm']*

*# 计算加权余弦相似度*

*song["wav\_entropy"] = features['wav\_entropy']*

*song["wav\_std\_dev"] = float(features['wav\_std\_dev'])*

*song["loudness"] = float(features['loudness'])*

*song["weighted\_cosine\_similarity"] = calculate\_weighted\_cosine\_similarity(features\_df, self.ranges)*

*updated\_songs.append(song)*

*self.update\_loudness\_in\_other\_playlists(song["path"], song["loudness"])*

*with open(cosine\_similarity\_list\_path, 'w', encoding='utf-8') as f:*

*json.dump(cosine\_similarity\_list, f, ensure\_ascii=False, indent=4)*

*self.sort\_cosine\_similarity\_list(cosine\_similarity\_list\_path)*

*self.record\_feature\_ranges()*

记录音频特征范围记录歌曲信息中各特征的最大和最小值，在计算更新余弦相似度播放列表数据时作为参数传入，参与特征归一化操作。

基于余弦相似度的插入排序通过使用在线算法的方式，避免在连续添加数据时不停的对数据集进行整体上的排序改变。首先获取余弦相似度播放列表，过滤其中还未测量的音频文件信息，在已经测量的数据上运行插入排序，将排序好的内容和未测量的音频文件信息合并写回余弦相似度播放列表文件。代码实现如下：

*def sort\_cosine\_similarity\_list(self, cosine\_similarity\_list\_path):*

*with open(cosine\_similarity\_list\_path, 'r', encoding='utf-8') as f:*

*cosine\_similarity\_list = json.load(f)*

*# 过滤weighted\_cosine\_similarity为None*

*filtered\_list = [item for item in cosine\_similarity\_list if item['weighted\_cosine\_similarity'] is not None]*

*add\_list = [item for item in cosine\_similarity\_list if item['weighted\_cosine\_similarity'] is None]*

*# 插入排序*

*for i in range(1, len(filtered\_list)):*

*key = filtered\_list[i]*

*j = i - 1*

*while j >= 0 and filtered\_list[j]['weighted\_cosine\_similarity'] < key['weighted\_cosine\_similarity']:*

*filtered\_list[j + 1] = filtered\_list[j]*

*j -= 1*

*filtered\_list[j + 1] = key*

*# 写回文件*

*with open(cosine\_similarity\_list\_path, 'w', encoding='utf-8') as f:*

*json.dump(filtered\_list+add\_list, f, ensure\_ascii=False, indent=4)*

删除选择项目时，分为删除播放列表内歌曲信息和删除播放列表，这一点根据判断右键单击的list\_widget与组件对比即可分辨。在处理歌曲信息删除时，通过选择list\_widget的内容获得需要删除歌曲的名称，打开当前播放列表，筛选出该歌曲信息，并将该歌曲从文件中删除，这里如果需要删除余弦相似度播放列表内的信息，需要用户手动在余弦相似度播放列表中删除。在处理播放列表删除时，选择list\_widget的内容获得需要删除播放列表的名称，打开播放列表文件夹，再通过系统os库实现文件删除，然后检查删除的播放列表是否为目前打开的播放列表，如果是，则清除UI中的播放列表信息。

1. UI组件实现与样式编写

UI实现部分主要包括页面组件的声明和QSS文件的编写。在音乐播放器初始化时，调用成员函数self.create\_ui()进行UI部件的初始化，再通过self.load\_qss("component\\qss\\musicPlayer.qss")加载QSS样式文件。以播放列表部分为例，首先创建一个self.splitter = QSplitter(Qt.Horizontal)用于进行鼠标调整播放列表和播放列表选择器的宽度调整，再创建一个self.playlist\_box = QHBoxLayout()用于存放播放列表和播放列表选择器组件，声明播放列表和播放列表选择器为QListWidget，将两者放入self.splitter中，编写相关的信号与槽机制，再将self.splitter放入self.playlist\_box用于全局窗口的样式调整，最后通过QSplitter的方法调整播放列表和播放列表选择器的默认宽度比。在QSS部分，需要较为详细地调整Qt各组件的样式，样式以QPushButton为例，需要调整QPushButton在默认情况、选中、鼠标移到上方时、鼠标按下时等样式，因为虽然QSS在很大程度上受到CSS2规范的启发，但它并未完全实现CSS的所有功能，在某些复杂的样式设计中，比起CSS会有更大的局限，也因此QSS文件也会显得比较臃肿[6]。部分代码实现如下：

*self.control\_box = QHBoxLayout()*

*self.create\_control\_buttons()*

*self.create\_playback\_control\_buttons()*

*self.create\_volume\_slider()*

*self.create\_position\_slider()*

*self.main\_box.addLayout(self.control\_box)*

*def create\_style\_switch\_button(self):*

*self.style\_switch\_button = QPushButton("Switch Style", self)*

*self.style\_switch\_button.clicked.connect(self.switch\_style)*

*self.headline\_box.addWidget(self.style\_switch\_button)*

*def create\_control\_buttons(self):*

*self.play\_button = QPushButton("Play", self)*

*self.play\_button.clicked.connect(self.toggle\_play\_pause)*

*self.control\_box.addWidget(self.play\_button)*

*# 组件声明后略*

*QPushButton {*

*background-color: #c0c4c8;*

*color: #19232d;*

*border-radius: 4px;*

*padding: 2px;*

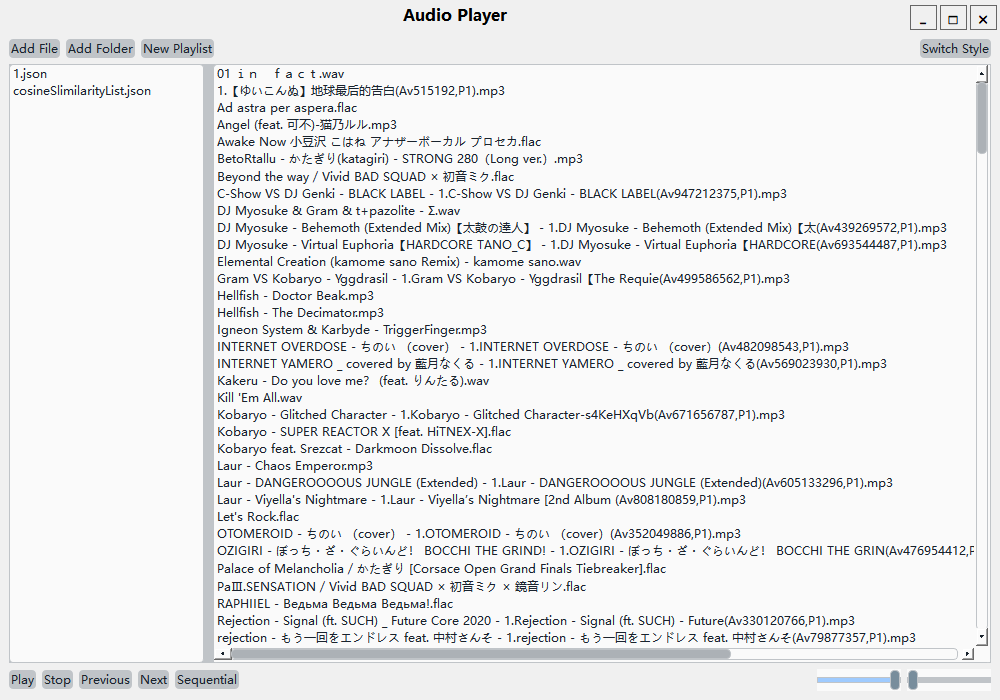
*outline: none;*

*border: none;*

*}*

*QPushButton:disabled {*

*background-color: #c0c4c8;*

创建的UI如图3.2所示。

*color: #9da9b5;*

*border-radius: 4px;*

*padding: 2px;*

*}*

*QPushButton:hover {*

*background-color: #b4b8bc;*

*color: #19232d;*

*}*

*QPushButton:pressed {*

*background-color: #acb1b6;*

*}*

*QPushButton:selected {*

*background: #acb1b6;*

*color: #19232d;*

*}*

*# 样式文件后略*

图3.2 UI组件创建与样式

1. 窗口状态更新

窗口状态使用QTimer的计时器机制，以100ms每次的频率运行成员函数update\_progress(self)，成员函数内只更新播放时进度条部分，其他需要更新的部分在其他功能内实现，提升软件总体速度。

1. 调用音频分析模块

调用音频分析模块时，主要在添加文件到播放列表时，调用音频特征获取模块和音频的余弦相似度分析模块，在调用音频特征获取模块时需要传输音频文件的路径，在调用音频的余弦相似度分析模块时需要传输音频文件基于Pandas的DataFrame打包好的数据类型，以及当前的特征范围标准值。调用代码如下：

*features = get\_song\_features(song["path"])*

*song["weighted\_cosine\_similarity"] = calculate\_weighted\_cosine\_similarity(features\_df, self.ranges)*

1. 余弦相似度播放列表的生成与维护

余弦相似度播放列表在设计上，包含了用户在本音乐播放器内添加在不同播放列表的所有音乐，让用户可以在一个播放列表中连续收听到在不同时期在接近曲风、风格和情感下的收听体验。当用户向音乐播放器中添加新的音频文件时，系统会自动调用音频分析模块，提取音频的特征信息，如节奏、旋律、音色等。余弦相似度测量模块会计算新添加的音频文件与已有播放列表中所有音频文件的相似度。这个计算过程基于余弦相似度公式，通过比较音频特征向量的夹角余弦值，来评估音频与标准值间的相似程度。在获得相似度结果后，音乐播放器会运用插入排序算法，将新音频文件以在线算法的形式按照相似度大小插入到余弦相似度播放列表的适当位置[13][15]。这样，列表中的音频文件将按照相似度从高到低进行排序，保证了用户在播放时能够连续收听到风格相近的音乐。在维护余弦相似度播放列表部分，会在添加文件和开启音乐播放器时开始静默扫描该播放列表，完善还没有完成扫描的音频文件信息，不会影响到用户正常使用音乐播放器其他功能。

### 3.2.2 音乐播放器窗口

音乐播放器窗口实现包括音乐播放器主窗口的功能实现和导航栏的行为编写，以及两者的样式编写。

主窗口需要实现的功能有实现模块的调用、和窗口大小和位置的拖拽和调整、多线程的信号管理。

主窗口在初始化时引入了导航栏TitleBar模块和音乐播放器本体MusicPlayer模块，初始化一个主QWidget，并且设置该QWidget为QMainWindow的主widget，接着在widget中添加QVBoxLayout(main\_widget)用于模块在竖直方向上的排列，将导航栏TitleBar模块和音乐播放器本体MusicPlayer模块添加到layout中，至此完成模块的调用。

窗口大小的拖拽和调整在Qt.FramelessWindowHint的标志下，无法使用Windows或Qt自带窗口的功能，因此需要手动实现该功能。在主窗口内设置各方向的拖拽信号，重写鼠标按下、鼠标移动、鼠标释放的方法，在鼠标按下时，如果按下范围在可拖拽位置范围内，设置拖拽窗口信号为True，如果按下范围在可拖拽位置范围内，根据鼠标位置判断在哪个边缘或角进行拖动，将该方向的拖拽信号设置为True，在按住鼠标移动时调整窗口的大小，在释放鼠标时将以上拖拽信号重新设置为False。实现代码如下（仅展示拖拽窗口和右侧部分）：

*self.rubberBand = QRubberBand(QRubberBand.Rectangle, self)*

*self.dragging\_right = False*

*self.dragging\_window = False*

*def mousePressEvent(self, event: QMouseEvent):*

*if event.button() == Qt.LeftButton:*

*if self.rect().adjusted(5, 5, -5, -5).contains(event.pos()):*

*self.dragging\_window = True*

*self.drag\_position = event.globalPos() - self.frameGeometry().topLeft()*

*event.accept()*

*else:*

*self.dragging\_window = False*

*self.dragging\_right = event.pos().x() > self.width() - 5*

*if self.dragging\_top or self.dragging\_bottom or self.dragging\_left or self.dragging\_right:*

*self.rubberBand.setGeometry(QRect(self.mapToGlobal(self.rect().topLeft()), self.mapToGlobal(self.rect().bottomRight())))*

*self.rubberBand.show()*

*event.accept()*

*def mouseMoveEvent(self, event: QMouseEvent):*

*if event.buttons() == Qt.LeftButton:*

*if self.dragging\_window:*

*self.move(event.globalPos() - self.drag\_position)*

*event.accept()*

*elif self.dragging\_top or self.dragging\_bottom or self.dragging\_left or self.dragging\_right:*

*elif self.dragging\_right:*

*self.rubberBand.setGeometry(QRect(self.mapToGlobal(self.rect().topLeft()), self.mapToGlobal(event.pos())))*

*event.accept()*

*def mouseReleaseEvent(self, event: QMouseEvent):*

*if event.button() == Qt.LeftButton:*

*if self.dragging\_window:*

*self.dragging\_window = False*

*elif self.dragging\_top or self.dragging\_bottom or self.dragging\_left or self.dragging\_right:*

*self.dragging\_right = False*

*self.resize(self.rubberBand.size())*

*event.accept()*

主窗口的多线程的信号管理主要在主窗口关闭时，通过closeEvent更新多线程关闭信号self.thread\_stop\_flag，在关闭时通过该信号通知子线程在完成功能后退出程序，保证程序整体的关闭。需要通知的模块主要是音乐播放器本体内的更新余弦相似度列表功能。

导航栏的编写主要包括导航栏样式、最小化、最大化/还原、关闭按钮控制主进程行为。在导航栏初始化时创建布局，添加标题文本，添加最小化、最大化/还原、关闭按钮并且通过信号与槽机制连接到主进程的功能函数，最后添加一个事件过滤器，用于实现主进程拖拽窗口移动的行为。其中最小化、最大化/还原、关闭都通过QMainWindow的方法实现，事件过滤器用于处理事件已经被处理的情况，避免了事件的重复处理。

### 3.2.3 音频内核

音频内核主要通过应用vlc音频库，在该音频库的基础上，实现在本音乐播放器内需要的逻辑操作，以及在播放时实现响度统一化[12]。在播放时，音频内核会获取当前音频文件的路径和响度信息，在播放前进行响度统一化相关计算在进行播放。响度统一化内容在3.4.1中详细介绍。音频内核需要相应音乐播放器本体的操作，比如播放控制、音量控制、位置控制、状态更新，并将逻辑传递到vlc内核实现最终的播放逻辑。音频内核重要方法代码实现如下：

*def \_\_init\_\_(self, musicPlayer):*

*self.instance = vlc.Instance()*

*self.musicPlayer = musicPlayer*

*self.media = None*

*self.mediaplayer = self.instance.media\_player\_new()*

*self.is\_paused = False*

*self.target\_loudness = -5*

*self.volume\_adjustment = 1*

*def play\_pause(self):*

*if self.mediaplayer.is\_playing():*

*self.mediaplayer.pause()*

*self.is\_paused = True*

*else:*

*if self.mediaplayer.play() == -1:*

*self.open\_file()*

*return*

*self.mediaplayer.play()*

*self.is\_paused = False*

*def calculate\_volume\_adjustment(self, current\_loudness, target\_loudness):*

*rms\_db = 20\*np.log10(current\_loudness)*

*adjustment\_factor = np.power(10, (target\_loudness - rms\_db) / 20)*

*return adjustment\_factor*

## 3.3 音频分析模块的设计与实现

*def open\_file(self, filepath, loudness):*

*self.media = self.instance.media\_new(filepath)*

*self.mediaplayer.set\_media(self.media)*

*self.media.parse()*

*if loudness is not None:*

*# 统一响度到 self.target\_loudness*

*self.volume\_adjustment = self.calculate\_volume\_adjustment(loudness, self.target\_loudness)*

*self.set\_volume(self.musicPlayer.volumeslider.value())*

*self.play\_pause()*

本节介绍音频分析模块在音频特征提取和相似度分析的方法和设计，各参数的相关介绍和研究在3.3.3音频的余弦相似度分析中详细介绍与分析。

### 3.3.1 音频特征获取

在音频特征获取中，从librosa库中选取了spectral\_bandwidth，spectral\_contrast，bpm三项参数[16][19]。该部分通过librosa库，提取该三项参数，然后通过调用音频频谱分析模块来获得需要返回的完整参数，另外在这里也进行响度统一化需要的最大响度测量。代码实现如下：

### 3.3.2 音频频谱分析

*def get\_song\_features(file\_path):*

*y, sr = librosa.load(file\_path)*

*spectral\_bandwidth = librosa.feature.spectral\_bandwidth(y=y, sr=sr)*

*spectral\_contrast = librosa.feature.spectral\_contrast(y=y, sr=sr)*

*tempo, \_ = librosa.beat.beat\_track(y=y, sr=sr)*

*wave\_analyzer = WaveAnalyzer(file\_path)*

*wav\_entropy = wave\_analyzer.entropy()*

*wav\_std\_dev = wave\_analyzer.std\_dev()*

*loudness\_frames = librosa.feature.rms(y=y)*

*return {*

*'spectral\_bandwidth': spectral\_bandwidth.mean(),*

*'spectral\_contrast': spectral\_contrast.mean(),*

*'bpm': tempo,*

*'wav\_entropy': wav\_entropy,*

*'wav\_std\_dev': wav\_std\_dev,*

*'loudness': np.max(loudness\_frames)*

*}*

音频频谱分析时，选择提取梅尔频谱熵wav\_entropy和梅尔频谱标准差wav\_std\_dev两项参数[20]。这里使用梅尔频谱是为了从音乐的复杂性和多样性，以及梅尔频谱的稳定程度，在余弦相似度分析起一定的辅助作用。代码实现如下：

### 3.3.3 音频的余弦相似度分析

*class WaveAnalyzer:*

*def \_\_init\_\_(self, filename):*

*self.filename = filename*

*self.y, self.sr = librosa.load(filename)*

*self.audio\_length = len(self.y) / self.sr*

*# 长度 0->1*

*self.start\_time = self.audio\_length \* 0*

*self.end\_time = self.audio\_length \* 1*

*self.y\_segment = self.y[int(self.start\_time \* self.sr):*

*int(self.end\_time \* self.sr)]*

*self.S = librosa.feature.melspectrogram(y=self.y\_segment,*

*sr=self.sr, n\_mels=128)*

*self.log\_S = librosa.amplitude\_to\_db(self.S, ref=np.max)*

*def entropy(self):*

*# 计算熵*

*log\_S\_normalized = self.log\_S / np.sum(self.log\_S)*

*# 添加一个非常小的常数到每个概率值中，避免零概率的情况*

*epsilon = 1e-10*

*log\_S\_normalized\_smoothed = (log\_S\_normalized + epsilon) / (np.sum(log\_S\_normalized) + epsilon \* log\_S\_normalized.shape[0])*

*entropy = -np.sum(log\_S\_normalized\_smoothed \* np.log2(log\_S\_normalized\_smoothed))*

*return 1/entropy*

*def std\_dev(self):*

*# 计算标准差*

*std\_dev = np.std(self.log\_S)*

*return std\_dev*

在这一节中，将详细分析余弦相似度分析中选取的各参数的意义与作用，以及在研究和分析基础上的系统的设计与实现。

由音频特征获取和音频频谱分析，可以得到总共五个参数：频谱宽度spectral\_bandwidth，频谱对比度spectral\_contrast，每分钟节拍数bpm，梅尔频谱熵wav\_entropy和梅尔频谱标准差wav\_std\_dev。

1. 参数介绍

频谱宽度spectral\_bandwidth是指围绕谱心（spectral centroid）的频谱范围，也就是从谱心的方差。其中，谱心是指频谱的质量中心位置，它通过计算信号中存在的频率的幅度加权平均值来得到。从感知角度来看，谱心与声音的亮度有着强烈的联系，广泛用于衡量任何音频文件的音调质量。谱心的位置越高，声音就越被感知为亮度更高。频谱带宽的计算涉及到从谱心的方差，这意味着它是基于频率带与谱心之间的距离的加权平均值[17]。这种方法允许我们量化信号在频率域上的扩散程度，从而提供关于音色特性的重要信息。频谱带宽的大小与音频信号的质地有关，例如，低频信号（如低音）通常具有较小的频谱带宽，而高频信号（如高音）则可能具有较大的频谱带宽。这反映了高频信号在其频率范围内的能量分布更广泛。由此我们可以通过这一参数大概上判断音频文件在流派上的大致范围，比如低音较重的Hardcore、Rock等流派，或低音较轻的Jazz[18]。这一点在前人的研究上也可以得到印证，根据该研究的描述结果，具有阴暗、低沉品质的声音倾向有较多低频内容，谱心相对较低。具有明亮、欢快品质的多数集中在高频，谱心相对较高。获得信号的频谱带宽说明整段旋律的频率是否平稳，还是律动感十足，在该研究中，可以得到以下结果：jazz和eletric高频多，轻快，律动感十足，和orchestral低频多，沉稳，律动感不强[19]。在这里，频谱宽度在余弦相似度分析中主要在风格和流派主要方向的判断上作用。

频谱对比度spectral\_contrast是一种用于描述声音频谱结构的特征。它通常用于衡量频谱中不同频率区段之间的能量差异。在分析频谱对比度时，会将频谱划分为多个频带，然后计算每个频带的能量，这里对于每个频带，计算其内部的频率范围内的信号能量，通常通过对频带内的信号进行傅立叶变换，并计算其幅度或功率来实现，最后计算频谱对比度。高频谱对比度可能表示在不同频率区段之间存在明显的能量差异，而低频谱对比度可能表示声音的频谱结构相对均匀。在前人研究中，该实验使用了大约1500首音乐作品，涵盖了巴洛克音乐、浪漫音乐、流行歌曲、爵士和摇滚等五种类型，其中巴洛克音乐、爵士和浪漫音乐，流行音乐和摇滚音乐均有一部分可能存在的风格和情感上的重合，而实验结果表明，频谱对比特征在10秒音频片段和整个音乐作品的分类中都取得了很好的性能，并且在与以MFCC特征在音乐类型分类中的表现中表现了该特征在音乐类型分类中的优越性[17]。这里使用该参数是为了在大致相似的风格流派中，通过分析频谱中不同频率区段之间的能量差异，来对更细节的风格和流派的判断上作用比如在Hardcore领域，有低频和失真相当突出的主流核和工业核，有低频和失真不是很多的UK Hardcore和Happy Hardcore。频谱对比度会在主要风格和流派确定的情况下，用于辨别更为细节的曲风。

每分钟节拍数bpm是一种用于描述音乐节奏速度的度量单位，它表示在一分钟内发生的节拍数或拍子数。并且对于曲风识别来说，该数值是一种有相当明确指向性的特征。比如在相同在Hard流派中，bpm180以上是正统的Hardcore，而bpm150-160这个区间，一般都属于Hardstyle，而这两者在编写手法和音频特征上有很大程度上比较相似，所以需要引入每分钟节拍数bpm来对这种情况进行区分[19]。

梅尔频谱（Mel Spectrogram）是一种用于音频信号分析的重要工具，它结合了梅尔频率尺度（Mel scale）和频谱分析技术，旨在模拟人类听觉系统的特性。研究表明，人类对频率的感知并不是线性的，并且对低频信号的感知要比高频信号敏感。例如，人们可以比较容易地发现500和1000Hz的区别，确很难发现7500和8000Hz的区别。这时，梅尔频度(the Mel Scale)被提出，它是Hz的非线性变换，对于以mel scale为单位的信号，可以做到人们对于相同频率差别的信号的感知能力几乎相同[21]。赫兹频度到梅尔频度的转换公式如下：

而将人耳不好感知到的赫兹频度转换为梅尔频度分析，更能体现在人主观上对于音频文件在风格、情感上的判别[20]。

这里，我们在得到音频文件的梅尔频谱后，计算出音频文件梅尔频谱的梅尔频谱熵wav\_entropy和梅尔频谱标准差wav\_std\_dev用于特征提取和余弦相似度计算和分析，并且这部分分析更加考虑到了人类在主观上对音乐风格和情感的理解与分析。梅尔频谱熵可以帮助理解梅尔频谱中的不确定性或随机性，从而衡量音乐的复杂性和多样性。对于那些具有高度变化和多样性的音乐结构，熵可以提供一个偏向人类主观判断上的直观的度量。梅尔频谱熵还可以反映音乐中的情感和主题变化。例如，一首音乐的熵会随着情感的变化而变化，该数值可以帮助理解音乐的情感演变的程度。梅尔频谱标准差可以帮助理解梅尔频谱的稳定性和变化程度。对于那些具有明显节奏和和声进行变化的音乐，标准差可以提供一个直观的度量。标准差还可以反映音乐的动态范围，即音量的变化范围，能够反映在人类听觉上更主观的动态范围。在基于梅尔频谱的两个特征数值下，可以让余弦相似度排序系统更加偏向人类主观认知，而不是简单的通过数值对频率的计算得到的纯数据化理论化的播放列表。

1. 参数处理

测得参数因为量表差距很大，不能直接用来进行余弦相似度的测量，因此在执行操作前需要对参数进行一些处理。首先在进行音频特征获取和音频频谱分析时，音乐播放器会维护一个音频特征的范围值self.ranges，通过该范围值和运算，实现参数的归一化。为了强调一部分参数的作用或削弱一部分参数的作用，需要增加一个特征权值来调整余弦相似度算法。最后通过sklearn库的余弦相似度函数进行余弦相似度的计算。

1. 权值分配

在权值分配上，首先频谱宽度spectral\_bandwidth起到了主要的流派和风格分析作用，因此权值较高，在程序中设定权值0.6。频谱对比度spectral\_contrast需要在相同流派和风格的情况下分辨更加细节的曲风，但在总体的情感和风格的分辨上作用不是非常大，该数值主要是在梅尔频谱熵在主观判断上做一定辅助作用，在程序中设定权值0.1。每分钟节拍数bpm在相同流派的曲风辨别中有非常强的特征辨别价值，在程序内设定权值0.2。梅尔频谱熵wav\_entropy在主观情感和风格有一定特征价值，而音乐本身上也是很偏主观的文化内容，因此在程序中设定权值0.15。梅尔频谱标准差wav\_std\_dev的意义是在spectral\_bandwidth的基础上，加入主观上对动态范围的判断，在余弦相似度排序中主要起一定的辅助作用而不是主要作用，在程序中设定权值0.05。

1. 功能实现

以下是该部分的代码实现：

*from sklearn.metrics.pairwise import cosine\_similarity*

*from sklearn.preprocessing import MinMaxScaler*

*def calculate\_weighted\_cosine\_similarity(features, feature\_ranges):*

*standard\_values = [1, 100, 1, 1, 100]*

*weightFunc = [0.6, 0.1, 0.2, 0.15, 0.05]*

*print(feature\_ranges)*

*features\_line = features[['spectral\_bandwidth', 'spectral\_contrast', 'bpm', 'wav\_entropy', 'wav\_std\_dev']]*

*for feature, (min\_val, max\_val) in feature\_ranges.items():*

*features\_line[feature] = (features\_line[feature] - min\_val) / (max\_val - min\_val) \* 99 + 1*

*weighted\_features = features\_line \* weightFunc*

*similarity\_score = cosine\_similarity(weighted\_features, [standard\_values])*

*return similarity\_score[0][0]*

1. 实现效果

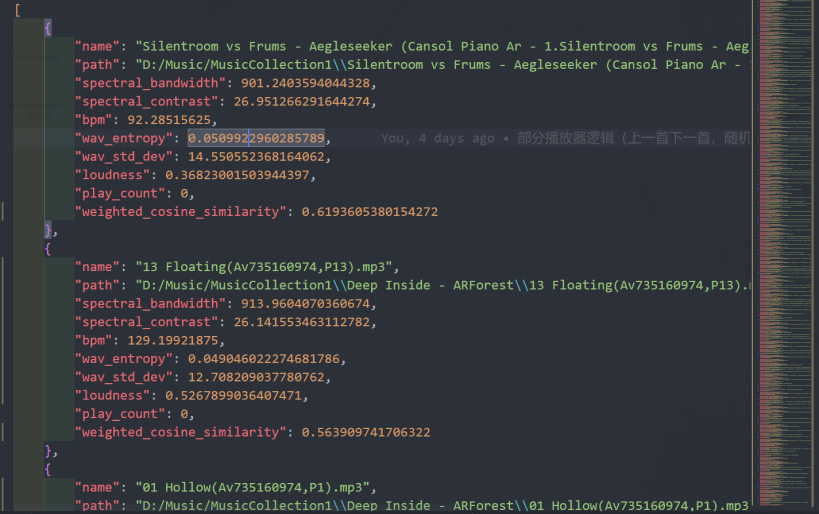
排序后的播放列表文件和数据类型3.3图所示：

图3.3 余弦相似度播放列表文件

首先在风格流派的排序上，在频谱宽度和梅尔频谱标准差的作用下算法很好的将情绪平缓且略带忧伤的曲目排序在了一起，而将主流核、碎核、工业核等在听感上相当强烈的曲目排序在了一起。而在频谱对比度、每分钟节拍数和梅尔频谱熵的共同作用下，在相同风格流派的排序上，制作比较极端的碎核、工业核排序要比常规的主流核更加靠后，一定程度上分离了相同风格流派的曲目。例如在Deep Inside - ARForest专辑中，负责情感导入的01 Hollow和负责情感回收的13 Floating两者，因为编排手法上更偏向情感引入的实验音乐，排在该播放列表的第2首和第3首，而同样为Psytrance的04 ARForest & KARUT - Wanderer of Laby和STEREOTYPE - Abracadabra - (PsyCore Series No.1)排在第210首和209首，前者因为有一部分片段略微偏离Psytrance的编排手法，而后者是编排平稳的Psytrance曲目，余弦相似度算法也可以在该种类似情况下有所表现。

## 3.4 其他功能的设计与实现

### 3.4.1 响度统一化

本音乐播放器中的响度统一化标准使用RMS算法。响度均一化（RMS）是一种用于测量音频响度的单位，它代表均方根（Root Mean Square）。RMS通过计算整个音轨或音频片段的平均响度水平来定义，从而提供了音频信号的整体能量水平[21]。相关代码实现如下：

*# 响度测量*

*loudness\_frames = librosa.feature.rms(y=y)*

*# 响度统一化应用*

*def open\_file(self, filepath, loudness):*

*self.media = self.instance.media\_new(filepath)*

*self.mediaplayer.set\_media(self.media)*

*self.media.parse()*

*if loudness is not None:*

*# 统一响度到 self.target\_loudness*

*self.volume\_adjustment = self.calculate\_volume\_adjustment(loudness, self.target\_loudness)*

*self.set\_volume(self.musicPlayer.volumeslider.value())*

*self.play\_pause()*

*def calculate\_volume\_adjustment(self, current\_loudness, target\_loudness):*

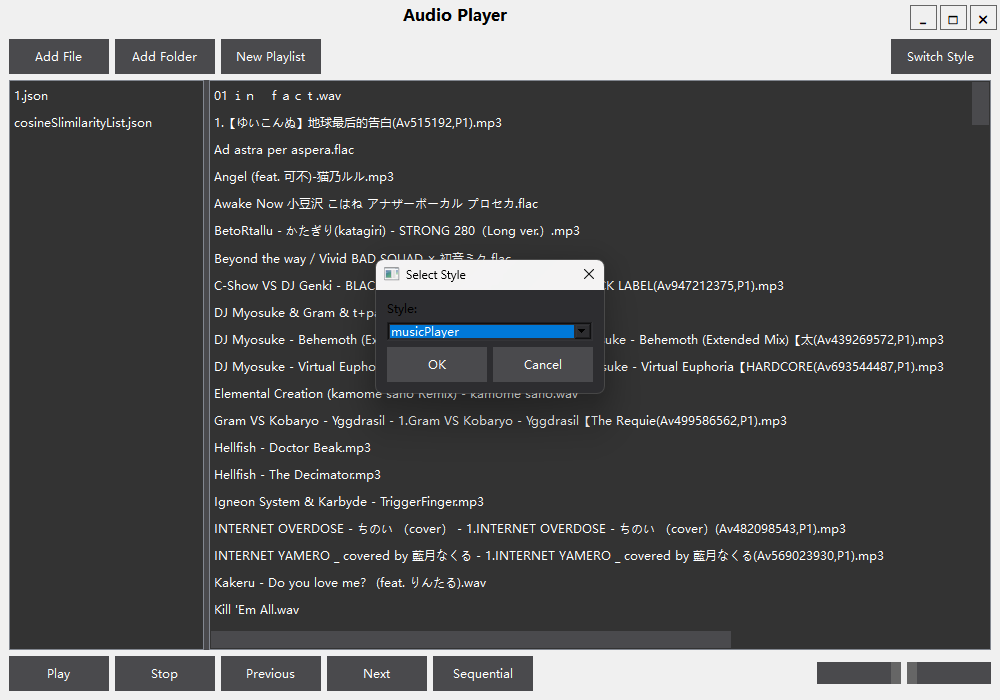
*rms\_db = 20\*np.log10(current\_loudness)*

*adjustment\_factor = np.power(10, (target\_loudness - rms\_db) / 20)*

*return adjustment\_factor*

### 3.4.2 样式替换

本音乐播放器可以通过替换QSS文件达到简单的样式替换功能。在音乐播放器本体中，通过成员函数switch\_style(self)内预设的样式，通过QInputDialog弹出对话框来让用户选择需要替换的样式，在用户选择样式后更换样式。用户也可以通过将样式文件放置在样式文件夹内，再添加该样式名称，实现样式的自定义添加。实现代码如下：

样式替换如图3.4所示：

*def switch\_style(self):*

*styles = ["musicPlayer", "brown", "black", "green", "white"] # List of available styles*

*style, ok = QInputDialog.getItem(self, "Select Style", "Style:", styles, 0, False)*

*if ok and style:*

*qss\_path = os.path.join(self.mainWindow.baseDir, f"component\\qss\\{style}.qss")*

*with open(qss\_path, "r") as f:*

*self.setStyleSheet(f.read())*

图3.4 UI样式替换和效果

# 4 疑难问题与解决方案

## 4.1 关于PySide6样式方案的问题

### 4.1.1 问题描述

在研究PySide6样式编写相关内容时，由于网络资源的匮乏和官方文档的庞大阅读量，在决定采用方案时遇到了一些困难。

### 4.1.2 解决方案

首先尝试使用官方的UI工具Qt Designer来绘制UI，并尝试导出UI文档。然而，发现无法直接使用pyuic工具将UI文档转换为.qss样式文件，只能转换为.py文件。接着，考虑使用Python内置的样式进行编写，但但在实际测试中发现这会带来内容和样式管理上的较大阻碍。最终，在参考了官方教程和多个应用案例，最后选择了使用PySide6开发应用功能，并采用样式QSS的技术方案[6]。

## 4.2 PySide6无法运行的问题

### 4.2.1 问题描述

在决定使用PySide6作为框架技术方案后，在测试运行时，PySide6无法运行。

### 4.2.2 解决方案

经过查询发现，这个问题是由于Shiboken6与Python和PySide6版本不匹配造成的。Shiboken6是PySide6的核心组件，负责将Python代码转换为C++代码以实现与Qt库的交互。如果Shiboken6与Python或PySide6的版本不兼容，就可能导致PySide6无法正常运行。我使用的Python版本号为3.10.6，要求PySide6运行需要匹配的版本号6.6.1的PySide6和版本号为6.6.1的Shiboken6，但由于可能pip错误，安装了错误的Shiboken6版本[5]。

## 4.3 关于音频特征选取的问题

### 4.3.1 问题描述

在音乐推荐相似度分析中。正确的特征选择可以帮助系统更好地理解和比较不同音乐的相似性。同时，特征的选择还需要考虑到计算成本和特征的可解释性。

### 4.3.2 解决方案

在刚开始测试该部分内容时，为了追求音乐推荐相似度的准确度，我一共选择了spectral bandwidth、spectral centroid、spectral contrast、spectral flatness、bpm五个音频特征分析特征，但是在实际测试中发现其中spectral centroid和spectral bandwidth，spectral contrast和spectral flatness都有一定功能重复，在仔细研究了spectral bandwidth该项在曲库中表现最好，并且在仔细阅读了spectral contrast的相关论文研究和关于spectral bandwidth前人的实验后，我删除了spectral centroid和spectral flatness这两个特征继续进行测试，但是在测试途中发现有一部分数据很不符合我个人对于我曲库的认知，于是我继续寻找有没有更能体验用户主观的音频参数，在基于自己对耳机的了解，以及进一步了解和学习人类听觉上和频谱表现的差异之后，我选择将梅尔频谱引入该算法，但梅尔频谱得到的结果为一个二维数组，无法在算法中直接应用，我接着继续测试如何提取梅尔频谱而不损失过多信息，在考虑到前面spectral bandwidth、spectral contrast、bpm三个频谱特征后[14][15][19]，我选择将梅尔频谱在算法中作为辅助参数来使用，于是在定下这个大方向后，继续不断尝试，获得了目前程序内使用到的音频参数和权值[16][17][18]。

## 4.4 关于添加文件时窗口卡住的问题

### 4.4.1 问题描述

在添加音频文件到音乐播放器时，会有明显的卡顿，添加文件夹时还会直接卡死。

### 4.4.2 解决方案

在认知到处理大量音频文件需要消耗大量的计算资源和时间后，我很快意识到，如果仅仅主进程，会导致用户界面（UI）线程被阻塞，进而使窗口无法及时响应用户的操作。为了避免这种情况，我将音乐文件添加、音乐文件特征获取、余弦相似度计算以及余弦相似度列表排序算法等操作进行合理的性能平衡。在开发过程中，优先考虑了音乐文件特征获取和余弦相似度计算这两部分，因为这两部分耗时较长，对整体性能影响较大。我们将这两部分合在一起一并在后续进行静默处理。在余弦相似度列表排序算法中，采用了插入排序这一在线算法。插入排序能够减少过多的操作对播放列表序列的影响，提高整体性能。接着，我继续优化了文件添加操作。在添加文件或文件夹时，首先仅读取歌曲的名称和路径，而不是一次性读取所有信息。同时，我们还采用多线程操作，以便更快地开始播放音乐。在音乐文件特征获取和余弦相似度计算这两部分，采用静默运行的方式，尽量减小对音乐播放器使用的影响。通过这些性能优化措施，我们确保了音乐播放器在处理大量音频文件时的流畅性和高效性。

## 4.5 关于在响度统一化标准在程序效率和实现上的问题

### 4.5.1 问题描述

将响度统一化处理放在音频内核内会导致播放前出现卡顿现象，将响度统一化处理写在添加文件部分也会带来一些问题，在这种情况下，响度统一化处理会随着文件添加过程的进行而逐步完成，严重影响了添加文件的速度。

### 4.5.2 解决方案

首先考虑响度统一化在音频内核中的卡顿现象，但是经过测试，这是librosa本身的性能不能满足该需求，因此不再考虑将响度统一化功能集成到音频播放内核或文件添加中，最后考虑放弃一部分功能需求，将这一部分功能放到音频特征提取和音频频谱分析模块中，虽然会有一定时间的空窗期让一部分音频没有得到响度统一化，但是在用户体验上更加强了音乐播放器本体的使用流程体验，我认为这样的舍弃是值得的。

## 4.6 关于librosa对音频解析速度较慢的问题和关闭窗口后音频解析线程仍然继续的问题

### 4.6.1 问题描述

在运行音频特征提取和音频频谱分析模块时，添加文件和特征提取的操作过慢，另外在关闭音乐播放器后，音频特征提取和音频频谱分析模块的音频解析仍然在继续按顺序读取并分析音频文件。

### 4.6.2 解决方案

这两个问题同为子线程问题，所以并在一起说。这里最大的问题是在声明线程时，将线程内放入了循环，而且这里的操作非常费时，于是就造成了不仅单线程解析速度慢，而且关闭窗口后线程迟迟不结束的问题。这里我先将这个循环拆分成了两个函数，一个负责控制循环和开线程，一个负责音频特征提取和音频频谱分析的任务，但是在测试时发现依然存在问题，在查询资料和检查整体代码后，发现应该将这一个线程结束信号标记放在主进程中，于是重新编写了主进程中的closeEvent，然后优化了主进程向模块的传值问题，最后该问题得到了解决。

4.7 关于选择余弦相似度算法在音频相似度分析的应用问题

### 4.7.1 问题描述

在刚开始选择算法时，因为在实现效果和算法性能要求的取舍上有疑问，而且在该领域缺乏研究，造成选定算法方面较为困难。最初的想法包括基于某种聚类算法，引入机器学习的思想，相似度算法。

### 4.7.2 解决方案

在阅读了多篇文章[16][17]和一些实验内容后[18]，明确了是可以通过少数特征来完成音频相似度分析这一需求的，而在文中的音乐推荐系统中，使用到了余弦相似度分析，在进一步了解多种相似度算法和余弦相似度的算法思想之后，我认定余弦相似度算法确实是可以完整项目需求的，所以最后选择了该算法用于音频相似度分析。

# 结 论

本研究成功设计并实现了一款基于在线相似度算法的音乐播放器，以解决数字化时代个性化音乐收听的需求。通过引入音频相似度分析的概念，我们提出并实现了一种具有音频相似度排序功能的播放器，有效解决了本地音乐库中不同风格歌曲导致的听歌体验不连贯的问题。该播放器采用Python编程语言，结合Qt框架进行界面设计，并借助多个音频和数学库来实现音频数据的解析和分析。本论文针对该音乐播放器的设计与实现的过程和细节进行了阐述，完成的工作主要有以下几个方面：

1. 本文介绍了开发这款音乐播放器的背景与意义，介绍了在开发本音乐播放器前需要的技术准备和理论基础。
2. 本文针对常见的音乐播放器和本音乐播放器需求的特色功能，从功能需求和非功能需求两个方面展开了需求分析，通过需求来明确整个项目的预期结果。

（三）本文结合了个人在音乐风格和流派的广泛认知、音频分析知识和积累的计算机学科素养，分三点从音乐播放器、音频分析模块、其他功能及其子内容进行了设计与实现方案的详细阐述。

（四）本文介绍了在开发该基于在线相似度算法的音乐播放器的过程中遇到的一些疑难问题和解决方案。

与商业的成熟方案相比，本音乐播放器系统在用户体验和算法优化方面还有一定的差距和进步空间。本文实现的音乐播放器和音频分析方法只是接触音频分析和应用领域的一个探索点，在应用实现和算法优化方面都有更多的技术要点待日后学习和优化。

# 参考文献

[1] Wikipedia.Python - 维基百科，自由的百科全书[EB/OL].(2024-04-07)[2024-05-05]. https://zh.wikipedia.org/wiki/Python

[2] 菜鸟教程.Python 简介 | 菜鸟教程[EB/OL].(2024-03-29)[2024-05-05]. https://www.runoob.com/python/python-intro.html

[3] Microsoft. Python in Visual Studio Code[EB/OL]. (2024-04-11)[2024-05-05]. https://code.visualstudio.com/docs/languages/python

[4] Qt Software. Qt for Beginners - Qt Wiki[EB/OL]. (2024-04-16)[2024-05-05]. https://wiki.qt.io/Qt\_for\_Beginners

[5] Qt Software.Qt for Python[EB/OL]. (2024-04-13)[2024-05-05]. https://doc.qt.io/qtforpython-6/

[6] Qt Software. Qt Style Sheets Reference | Qt Widgets 6.7.1[EB/OL]. (2024-03-06)[2024-05-06]. https://doc.qt.io/qt-6/stylesheet-reference.html

[7] librosa development team. librosa — librosa 0.10.2 documentation[EB/OL]. (2024-04-22)[2024-05-07]. https://librosa.org/doc/latest/index.html

[8] OVHcloud. pandas - Python Data Analysis Library[EB/OL]. (2024-05-01)[2024-05-07]. https://pandas.pydata.org/

[9] NumPy team. NumPy - [EB/OL]. (2024-05-03)[2024-05-07]. https://numpy.org/

[10] :probabl,Nvidia,Microsoft等. scikit-learn: machine learning in Python — scikit-learn 1.4.2 documentation[EB/OL]. (2024-04-28)[2024-05-07]. https://scikit-learn.org/stable/index.html

[11] . [EB/OL]. ()[2024-05-08]. https://wiki.videolan.org/Python\_bindings/

[12] OlivierAubert,DoesItReallyMatter,Rsm等. Python bindings - VideoLAN Wiki[EB/OL]. (2019-06-28)[2024-05-08]. https://zh.wikipedia.org/wiki/%E4%BD%99%E5%BC%A6%E7%9B%B8%E4%BC%BC%E6%80%A7

[13] WPCleaner (bot),龙方淞,Alexander Misel等. 在线算法 - 维基百科，自由的百科全书[EB/OL]. (2017-12-02)[2024-05-08]. https://zh.wikipedia.org/zh-cn/%E7%B7%9A%E4%B8%8A%E6%BC%94%E7%AE%97%E6%B3%95

[14] da吃一鲸886. 深入理解基于梅尔频谱的音频信号分类识别-百度开发者中心[EB/OL]. (2024.02.18)[2024-5-8]. https://developer.baidu.com/article/details/3068892

[15] Alyza, J.M., Utomo, F.S., Purwati, Y.E., Kusuma, B.A., & Azmi, M.S. MUSIC RECOMMENDATION SYSTEM BASED ON COSINE SIMILARITY AND SUPERVISED GENRE CLASSIFICATION[J]. JITK (Jurnal Ilmu Pengetahuan dan Teknologi Komputer).2023:77-79

[16] Dan-Ning Jiang, Lie Lu, Hong-Jiang Zhang等.Music type classification by spectral contrast feature[J],Proceedings. IEEE International Conference on Multimedia and Expo, Lausanne, Switzerland, 2002, 113-116

[17] Orangeadetea. 音频&深度学习Lesson4\_谱质心和子带带宽\_谱质心带宽-CSDN博客[EB/OL]. (2023-04-07)[2024-05-09]. https://blog.csdn.net/m0\_73471144/article/details/130019781

[18] Katsiaryna Ruksha. Music Information Retrieval: Feature Engineering | by Katsiaryna Ruksha | Medium[EB/OL]. (2024-02-15)[2024-05-09]. https://medium.com/@kate.ruksha/music-information-retrieval-feature-engineering-05748d26df48#:~:text=Spectral%20bandwidth%20is%20the%20spectral,bandwidth%20correlates%20with%20perceived%20timbre.

[19] ZCHBImpulseX. 电子音乐刻板印象大赏 第六期 | Hardcore\_哔哩哔哩\_bilibili[EB/OL]. (2023-06-01)[2024-05-09]. https://www.bilibili.com/video/BV1Yo4y1u7ri/?spm\_id\_from=333.337.search-card.all.click&vd\_source=8bf6c431f6d1b56d9294921785ef8d35

[20] 转行的炼丹师. 梅尔频谱(mel spectrogram)原理与使用 - 知乎[EB/OL]. (2021-02-22)[2024-05-09]. https://zhuanlan.zhihu.com/p/351956040

[21] 人在火星-刚下飞船.音频 响度标准化 EBU R.128[EB/OL]. (2020-09-20)[2024-05-10]https://www.bilibili.com/read/cv7678034/

# 致 谢

大学四年转眼即逝，这几年的大学生活让我学到了很多东西。我学到了很多以前从未接触过的专业知识，遇到了德高望重、平易近人的老师，结识了优秀的同学和志同道合的伙伴。如今，我即将毕业，即将离开校园，踏入社会，迎接人生新的舞台和新的挑战。在此，我要向所有帮助过我的老师、同学和亲朋好友表示最诚挚的谢意！

首先，我要感谢我的毕业设计指导老师叶斌老师。叶老师责任心强，知识渊博，总是及时而且详细地解答我的问题和困惑，经常给我提出宝贵的建议和指导。叶老师严谨的治学态度和深厚的学术素养让我受益匪浅，将对我未来的人生道路产生深远的影响。

其次，我要感谢我的父母和朋友，感谢你们一直以来对我的关心和鼓励。在求学和毕业设计非一日之功，在这个过程中，我遇到了很多困难和挑战。是你们的关心和支持给了我毅力、信心去面对遇到的困难和克服困难的决心。

最后，我要感谢培养了我四年的母校成都信息科技大学，感谢所有指导我、教育我的老师们。感谢你们的无私帮助。在此，我向你们表示衷心的感谢和深深的敬意！

作者简介：

姓 名：曹宇恒 性别：男

出生年月：1999年12月01日 民族：汉族

E-mail:844288610@qq.com

# 声 明

本论文的工作是2023年11月至2024年5月在成都信息工程大学计算机学院完成的。文中除了特别加以标注地方外，不包含他人已经发表或撰写过的研究成果，也不包含为获得成都信息工程大学或其他教学机构的学位或证书而使用过的材料。

关于学位论文使用权和研究成果知识产权的说明：

本人完全了解成都信息工程大学有关保管使用学位论文的规定，其中包括：

（1）学校有权保管并向有关部门递交学位论文的原件与复印件。

（2）学校可以采用影印、缩印或其他复制方式保存学位论文。

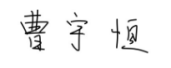
（3）学校可以学术交流为目的复制、赠送和交换学位论文。

（4）学校可允许学位论文被查阅或借阅。

（5）学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

除非另有科研合同和其他法律文书的制约，本论文的科研成果属于成都信息工程大学。

特此声明！



作者签名：

2024年5月23日