

华中科技大学

《软件工程》项目报告

题目：听曲记谱 APP

课程名称：软件工程

专业班级：CS2010

组 名：听曲记谱 APP

同组成员：学号：U202015598

姓名：何梦杰

学号：U202015595

姓名：常梦成

学号：U202015610

姓名：徐海林

指导教师：冯琪

报告日期：2022.12.14

计算机科学与技术学院

任 务 书

一 总体要求

1. 综合运用软件工程的思想，协同完成一个软件项目的开发，掌软件工程相关的技术和方法；
2. 组成小组进行选题，通过调研完成项目的需求分析，并详细说明小组成员的分工、项目的时间管理等方面。
3. 根据需求分析进行总体设计、详细设计、编码与测试等。

二 基本内容

根据给出的题目任选一题，自行组队，设计与开发中软件过程必须包括：

1. **问题概述、需求分析：**正确使用相关工具和方法说明所开发软件的问题定义和需求分析，比如 NABCD 模型，Microsoft Visio，StarUML 等工具（20%）；
2. **原型系统设计、概要设计、详细设计：**主要说明所开发软件的架构、数据结构及主要算法设计，比如墨刀等工具（35%）；
3. **编码与测试：**编码规范，运用码云等平台进行版本管理，设计测试计划和测试用例（30%）；
4. **功能创新：**与众不同、特别吸引用户的创新（10%）；
5. **用户反馈：**包括用户的使用记录，照片，视频等（5%）。

目 录

1 问题定义	4
1.1 项目背景与意义	4
1.1.1 需求 NEED	4
1.1.2 Approach 做法	4
1.1.3 Benefit 好处	5
1.1.4 Competitors 竞争	5
1.1.5 Delivery 推广	5
1.2 项目基本目标	5
1.3 可行性分析	6
1.4 人员管理和项目进度管理	7
1.4.1 人员管理	7
1.4.2 项目进度管理	8
2 需求分析	9
2.1 需求分析概述	9
2.2 UML 相关需求分析图	10
2.2.1 用例图	10
2.2.2 顺序图	10
2.2.3 E-R 图	11
2.3 原型系统设计	12
3 概要设计和详细设计	19
3.1 系统结构	19
3.2 类图等	21
3.3 关键数据结构定义	21
3.3.1 系统功能数据结构	21
3.3.2 用户管理数据结构	24
3.3.3 数据关联结构设计	25
3.4 关键算法设计	26
3.5 数据管理说明	34
4 实现与测试	35
4.1 实现环境与代码管理	35
4.2 关键函数说明	36
4.2.1 index 模块	36
4.2.2 collection 和 histRecord 模块	36
4.2.3 display 模块	37
4.2.4 mic 模块	37
4.2.5 audioFile 模块	38
4.2.6 analysis 模块	39

4.3 测试计划和测试用例	40
4.3.1 常用的软件测试方法	40
4.3.2 测试用例	41
4.4 结果分析	45
5 总结	46
5.1 用户反馈	46
5.2 全文总结	47
6 体会	49
附录	52

1 问题定义

1.1 项目背景与意义

对于本次软件工程项目的选题，在经过小组的讨论和头脑风暴之后，我们决定开发一款听曲记谱的 APP。对于很多音乐爱好者或是从事音乐相关工作的人来说，简谱的来源途径非常有限：书本形式的简谱所能记录的曲目有限，而且无法随着时间的推移不断更新；而在网络上查找有关曲目的简谱一方面需要知道曲目的名称，另一方面，网络查找的结果良莠不齐，会耗费大量的时间。我们也注意到也有一些专业音乐工具提供了曲谱转换的功能，但是这些专业软件有一定的学习成本，使用操作过于复杂，同样不利于广大音乐爱好者的使用。经过讨论我们小组一致认为，目前市面上缺少一款便于广大音乐爱好者和从业者使用的简单易用的曲谱转换工具。

1.1.1 Need 需求

简谱作为一种简易的记谱法，相较于五线谱有着简单和便于使用的特点，便于群众性音乐活动的开展。而对于许多希望使用简谱的人来说，不同音乐的简谱获取成为了一个问题，音乐爱好者们在听到一首歌曲时，希望得到相应的简谱进行跟唱学习或者是进行演奏；还有一些乐器练习者（如钢琴等），在训练的过程中希望能够知道自己练习的音准是否正确。他们都希望通过一个可靠的平台快速地完成对于一段音乐的简谱解析，得到可以直接使用的简谱。

1.1.2 Approach 做法

技术上，平台可以直接录制音乐进行解析，用户也可以上传本地的音乐，对声音进行频谱分析，利用机器学习或频谱分析算法训练已有音阶数据，用得到的模型对当前音频进行学习，解析后生成简谱。用户可以对已生成的简谱进行收藏，之前解析过的简谱也会在历史记录中保存供用户查找。商业模式上，联合音乐软件进行宣传，用户听到感兴趣的音乐时可以通过本软件进行简谱转换。

1.1.3 Benefit 好处

用户可以通过平台对某一段音乐快速可靠地完成简谱的转换并直接使用，即录即用，免去了用户上网查找简谱的麻烦。对于乐器练习者们，能够在弹奏乐器的同时了解到自己音准练习效果如何，方便了乐器学习过程。同时平台提供的历史记录和收藏功能可以使用户在下一次使用时快速找到之前转换过的简谱。

1.1.4 Competitors 竞争

网易云音乐、QQ 音乐等音乐平台中，都会有听音识曲的功能，在进入一首歌曲之后，在平台内部就能够查找当前播放歌曲的各种信息，包括简谱。它们的听音识曲功能比较准确，简谱资源也很全，但是简谱只能在找到一首歌曲后在这首歌曲的界面上获取，不能由麦克风收集音频直接产生简谱。而目前直接提供音乐转谱的软件较少，并且操作复杂，专业性较强，需要用户具有一定的相关知识，给用户造成了困难。

1.1.5 Delivery 推广

通过 B 站、抖音等视频平台演示产品相关功能，进行宣传。同时也可以联合音乐软件进行推广，用户听到感兴趣的音乐时可以通过本软件进行简谱转换。还能将其推荐给一些声乐培训班，让他们推荐我们的产品给声乐练习生使用。

1.2 项目基本目标

本次软件工程项目的基本目标是实现一款听曲记谱的 APP。对于这一总体目标，我们小组讨论出了我们需要实现的听曲记谱的基本功能，包括音频的采集和导入、对音频的解析和结果处理等等。在研究了其他音乐 APP 的功能之后，为了增加用户的体验，我们也加入了历史记录和收藏功能，便于用户快速地定位到需要解析的曲谱。最后，我们的讨论成果绘制为如图 1-1 所示的思维导图。

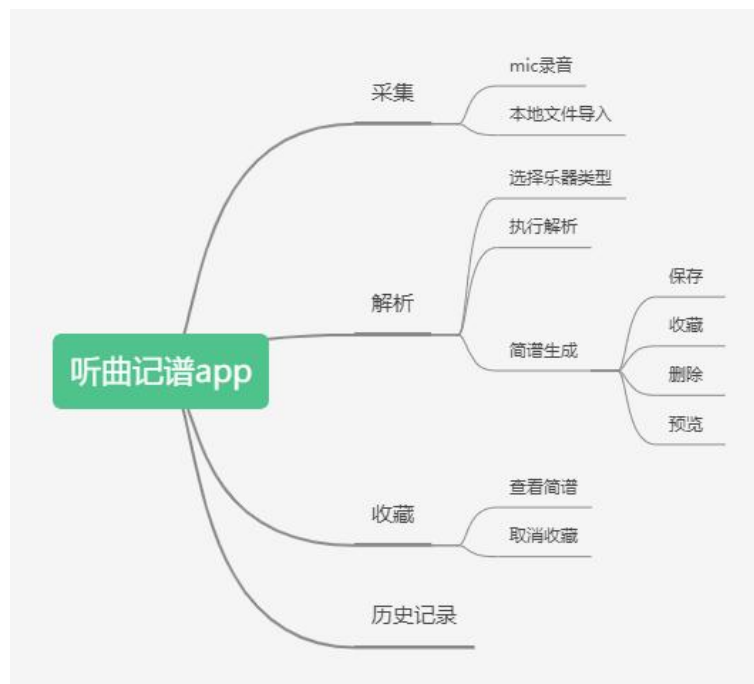


图 1-1 听曲记谱 APP 思维导图

通过思维导图的任务划分，我们可以将总体的项目目标分解为如下的几个子目标。

- 1、实现音频的采集功能，音频的具体来源可以分为两种，一种是现场通过 mic 录制音频，另一种方式是在本地文件中导入音频。用户在使用时可以根据自身的需求自行选择。
- 2、实现音频的解析。具体的解析过程为，用户首先需要选择乐器的类型，之后开始解析音频文件，解析完成之后的结果以简谱的形式输出，对于解析后简谱，用户可以保存相关简谱，也可以收藏，便于下次使用。
- 3、实现简谱的收藏功能。用户在收藏简谱后，可以在软件内查看所有已收藏的简谱，同时用户也可以在其中取消收藏。
- 4、实现历史记录功能，和收藏功能相似，用户既可以查看历史解析记录，也可以对历史记录进行删除。

1.3 可行性分析

在确定了项目的基本目标之后，小组根据每一个具体目标的要求，实现的难度和可行性进行考量，最终我们确定了目前的计划方案是可行的。

在市场需求方面对项目进行分析，我们认为设计的听曲记谱 APP 有着较为

广阔的市场前景和应用前景。在用户的需求上来看，目前仍有相当一部分音乐爱好者通过简谱来学习和演奏相关乐器。对于那些期望找到曲目简谱的用户来说，我们设计的 APP 可以提供快速便捷的曲谱转换功能；而对于那些希望得知自己演奏的曲目音准是否正确的用户，我们的 APP 也可以现场录制曲目，用户可以通过转换后的简谱和原谱相比较，从而得知自己的音准。

从市场竞争上来讲，目前市面上存在一些带有曲谱转换功能的专业的音乐软件，这些软件的体量较大，功能全面，但是操作复杂，需要用户具备一定的专业能力，并且部分专业软件还需要付费使用，这对于那些仅仅希望使用曲谱转换功能的用户来说就设置了一定的门槛。而我们的曲谱转换 APP 完全免费，在手机上即可运行，软件没有其他复杂的操作。我们认为这款 APP 可以在市场上具备一定的差异化竞争能力。

在开发成本方面，本次项目的工作量适中，实现我们所设计的功能的成本可以接受。小组成员在完成其他课业的基础上可以完成项目，项目的开发也无需专业的设备和场地。

1.4 人员管理和项目进度管理

确定了项目的基本目标和可行性分析之后，我们开始制定项目的开发计划，根据每一个小组成员的情况分配任务。

1.4.1 人员管理

小组内共有三名成员，经过最终的讨论决定具体分工如下：

何梦杰：在项目开始前进行需求分析，绘制 UML 图，同时进行系统的概要设计与详细设计，参与原型的部分设计，在项目过程中，使用 js 语言负责系统逻辑与后台实现，各类算法的实现，以及代码测试等工作，在项目完成后也负责对项目文档以及最终报告的整理与编写。

常梦成：负责前端页面 UI 设计和界面渲染的实现。以 html+css 为主要程序语言，主要负责各个页面模块的渲染呈现和页面跳转逻辑关系设计，同时最后负责进行用户反馈收集和整合，反馈给团队，进行后续的版本迭代。

徐海林：在本次的项目开发过程中主要负责原型系统的设计和页面的 UI 设计。在项目正式开发之前负责确定需求分析，根据需求模型和项目基本目标设计初步的原型系统和页面的布局管理，开发过程中根据开发情况对页面和原型系统做适当的更改和优化。

1.4.2 项目进度管理

在确定了项目的基本目标和分工，小组在经过综合的考量后，制定的项目进度计划表如表 1-1 所示。

表 1-1 项目进度计划表

任务	开始时间	计划天数
需求分析	2022. 9. 16	2
系统原型设计	2022. 9. 18	5
UI 设计	2022. 9. 23	7
登录模块设计	2022. 10. 4	5
文件上传模块设计	2022. 10. 9	5
录音模块设计	2022. 10. 14	5
解析模块设计	2022. 10. 19	10
简谱生成模块设计	2022. 10. 29	5
个人中心模块设计	2022. 11. 4	2
代码测试	2022. 11. 6	2
系统测试	2022. 11. 6	2
用户反馈收集	2022. 11. 8	1
用户反馈处理	2022. 11. 9	1

2 需求分析

2.1 需求分析概述

针对不同的用户群体，我们做出了以下合理需求分析：

- 1、对于掌握一定音乐知识的爱好者，在演奏准备过程中需要得到很多简谱以供演奏，曲谱转换 APP 就可以通过现场采集音频或者是读取本地文件来实现音频到曲谱的快速转换。在得到不同种类的曲谱之后，演奏者可以根据偏好进行收藏添加，APP 也会记录解析历史。这样在下一次需要某个曲谱的时候，用户就可以快速地在 APP 内找到记录。
- 2、对于正在学习音乐知识的练习生来说，他们练习的曲目比较固定，在获取曲谱以供演奏的需求方面弱于前者。但是练习生在练习的过程中希望得知自己的练习是否标准正确，这时就可以使用听曲记谱 app 通过现场采集音频进行解析，输出后的结果可以供用户比较，从而得知自己的练习效果。

通过用例场景的需求分析，可以列出如下功能：

- 1、用户登录：根据用户的 id 和密码进行登录，新用户需要注册账号。
- 2、主界面：点击开始按钮即可开始选择文件源进行解析，同时提供其他功能入口按钮。
- 3、解析：在选择音频源之后，开始解析音频。
- 4、结果展示：以简谱的形式输出解析结果，提供收藏和保存功能。
- 5、收藏：点击收藏后，当前解析文件将被加入收藏队列中，用户可以在之后的收藏中进行查看。
- 6、历史记录：保存近期解析过的所有音频记录。

2.2 UML 相关需求分析图

2.2.1 用例图

根据用户的需求分析和操作，得出用例图如图 2-1 所示。

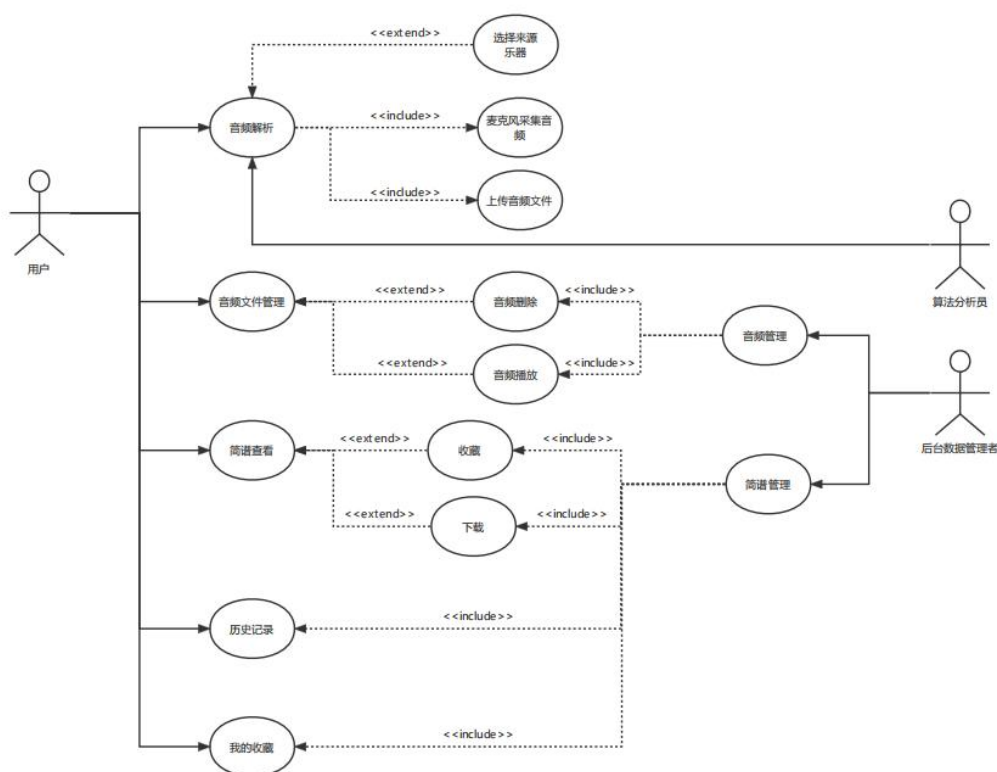


图2-1 用例图

用户打开软件，APP 首先对其提供登录功能，通过用户输入的用户名和密码进行登录操作。之后进入主页面的展示，通过主界面的功能提示，用户以点击的方式进入相应的功能。主页面提供的功能有：音频解析、音频文件管理、收藏查看、历史查看和简谱查看。

音频解析包含的子功能有：选择来源乐器、麦克风采集音频，上传音频文件。音频文件管理包含的子功能有：音频删除、音频播放。简谱查看包含的子功能有：收藏和下载。历史记录和我的收藏同时提供简谱查看和下载。

2.2.2 顺序图

根据用例分析，展示了各功能之间的层级关系，现在通过用户的一次使用过

程，做如图 2-2 所示的顺序图。

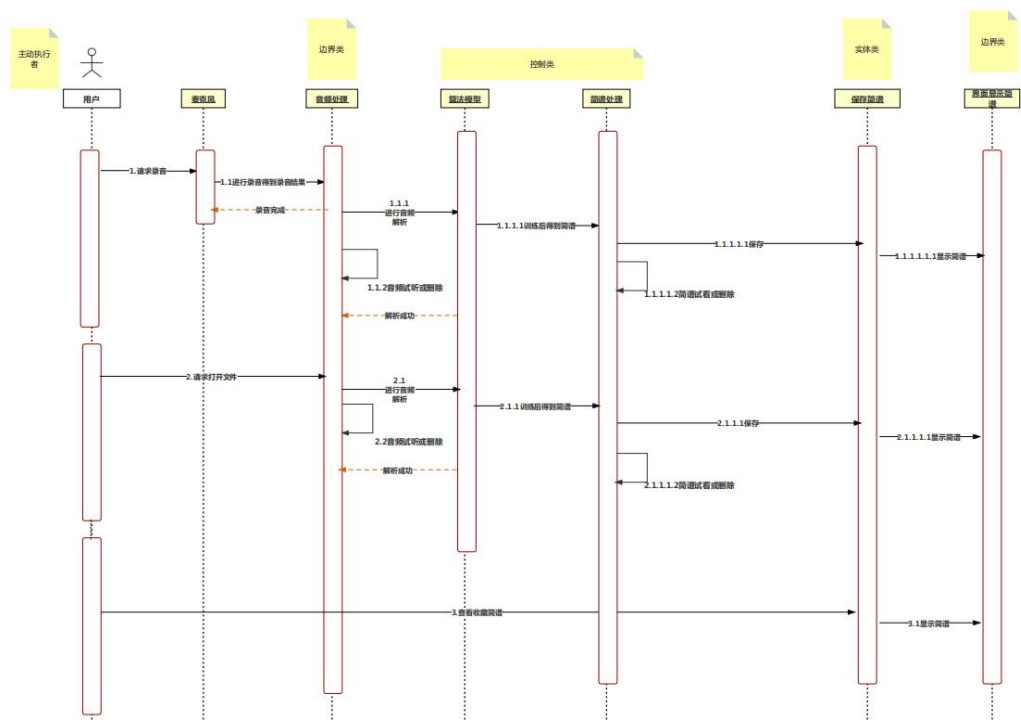


图2-2 顺序图

对于需要解析音频的用户来说，用户登录后，若选择录音采集音频进行解析，则先跳转至麦克风模块，麦克风录音得到录音结果，同时音频处理模块向麦克风模块返回录音完成结果。跳转至音频处理模块。若选择本地文件解析，则在读出文件后跳转至音频处理模块。在音频处理模块，用户可以对要解析的音频进行确认或者删除，若确认解析，则跳转至算法模型模块交由算法模型进行解析和训练。算法模型在训练后得到的简谱交由简谱处理模块进行处理。同样地，简谱处理模块也支持用户对简谱的试看和删除，若用户确认简谱，则跳转至保存简谱模块。简谱保存后跳至界面显示模块，进行简谱的最终显示。

对于需要直接查看历史记录和我的收藏简谱的用户来说，用户在选择相应功能后直接跳至简谱的保存模块，读出相应的简谱后跳至显示模块进行显示。

2.2.3 E-R 图

根据上述系统用例图分析，我们将对其进行具体的实体-联系分析，从而获取所需的实体与属性，以及实体间的联系关系。听曲记谱系统 E-R 图如图 2-2 所示。

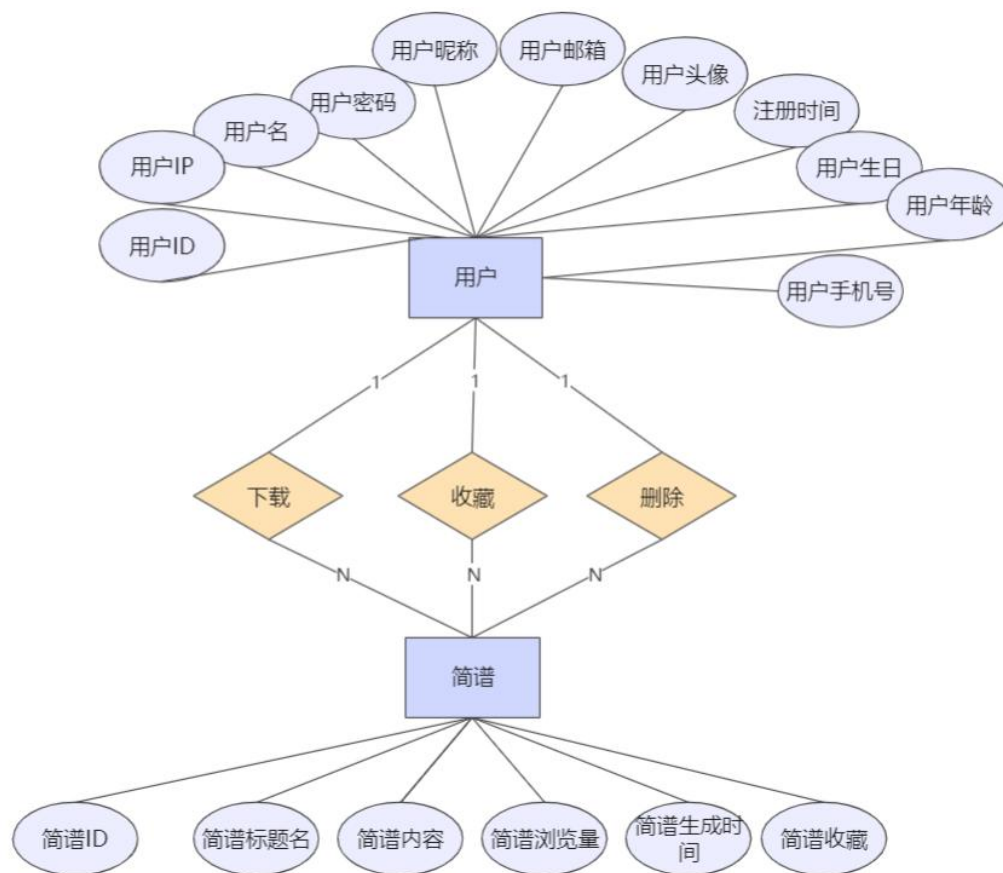


图2-3 E-R图

该系统围绕用户和简谱两类实体。

对于用户这一实体而言，包含用户登录注册、使用过程的有关信息属性，如用户 ID、用户 IP、用户名、用户密码、用户昵称、用户邮箱、用户头像、注册时间、用户生日、用户年龄以及用户手机号，用来作为用户账号模块和个人信息模块的信息准备，便于用户授权对应信息，了解当前系统状态。

简谱实体则包含简谱 ID、简谱标题名、简谱内容、简谱浏览量、简谱生成时间、简谱收藏等属性，用于简谱对应操作功能的实现。

我们提取出了 3 个用户与简谱间的联系：下载简谱、收藏简谱以及删除简谱。并将相关联系包含到各项功能的实现中。

2.3 原型系统设计

原型系统的设计详情请见：

<https://modao.cc/app/Mrs1ueerkgb59t1wJ008c#screen=s19oe6eeid7362x>

现在以一个用户的完整使用过程来展示原型系统的设计。

用户点击 APP，进入初始的登录页面，如图 2-4 所示。

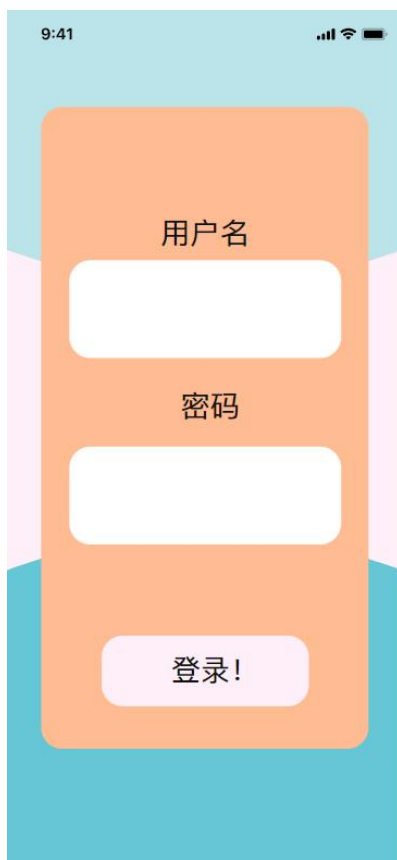


图 2-4 登录界面

用户输入正确无误的用户名和密码之后，点击登录按钮，正式进入主界面，如图 2-5 所示。

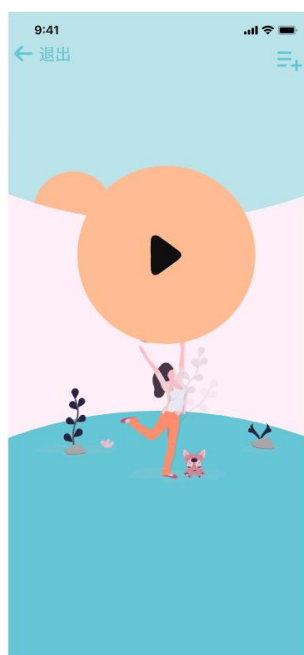


图 2-5 主界面

对于想要进行音频解析的用户，在进入主界面后，可以点击左上角登出按钮进行登出，也可点击中间原型按钮进行解析，如图 2-6 所示。



图 2-6 主菜单界面

此时用户需要选择音频来源，若要选择本地文件，则在选择文件界面（如图 2-7 所示）选好文件后跳转至解析页面（如图 2-10 所示）。若选择录制，则用户可以在录制页面（如图 2-8）控制录制进度。



图 2-7 选择文件界面



图 2-8 录制界面

点击结束录制后进入录制确认页面，用户可以在此页面进行录音确认，若不满意则可以点击重新录制，否则点击解析开始音频解析。



图 2-9 录制完成界面

解析页面如图 2-10 所示。



图 2-10 解析界面

经过短暂解析等待，页面跳至解析完成页面，如图 2-11 所示，用户可以根据自身喜好为简谱取名。



图 2-11 简谱命名界面

取名确认后，进入简谱的展示页面，如图 2-12 所示，用户可以在此页面查

看解析生成的简谱，也可以点击心形按钮进行收藏添加、点击下载按钮进行本地保存，或点击删除按钮删除此简谱。右上角设计了选项按钮可以供用户快速跳转至我的收藏和历史记录页面，如图 2-13 所示。



图 2-12 简谱展示界面



图 2-13 跳转至个人中心展示

此选项按钮功能和主页面的选项按钮功能一致,设计在这里只是为了方便用户在查看简谱的同时快速查看其他简谱。点击我的收藏,进入我的收藏页面,如图 2-14 所示。点击历史记录进入历史记录页面,如图 2-15 所示,两个页面之间可以跳转。两个页面都以目录的形式保存解析过的简谱。



图 2-14 我的收藏页面

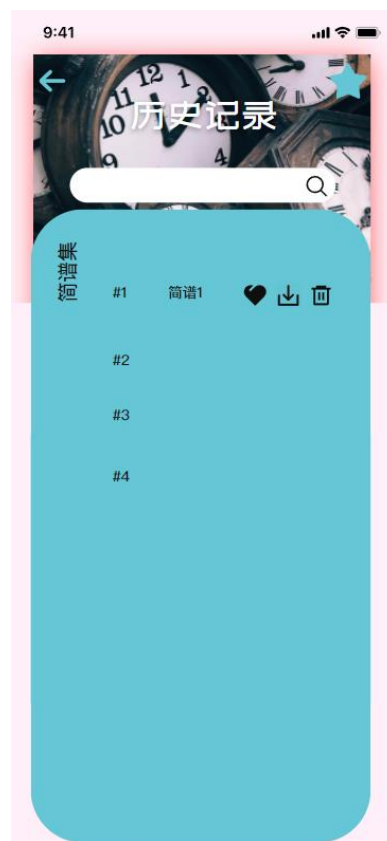


图 2-15 历史记录页面

3 概要设计和详细设计

3.1 系统结构

本系统从功能上可以分解为多个模块，依靠模块自身功能的实现以及模块之间的协同合作达到预期的效果。具体到各个模块的功能，可以将其分为：音源采集模块、文件上传模块、解析模块、简谱生成模块、个人中心模块、用户账号模块。

1.音源采集模块。主要功能：通过麦克风采集音频，传入解析部分进行音源的乐谱解析。解析部分完成解析后返回解析结果，随着录音过程的进行，数字简谱展现在屏幕前。录音结束后可以试听或者删除。

2.文件上传模块。主要功能：上传本地音频文件进行解析，支持MP3、wav格式在内的多种格式音源，将音源文件处理成音频数组传入解析部分。解析部分完成解析后返回解析结果，解析结果会同音乐播放实时显示。

3.解析模块。主要功能：对传入的音频文件或采集到的音频进行解析。接受参数是和音频相关的audioBuffer数组，通过频谱分析方法FFT，得到频率数组，提取频率分量最大的频率为特征值，和标准音频率比较，选择最接近的频率作为解析结果。该模块调用频率和系统帧率有关，可以自行调节。完成对应的解析操作后，对应信息写入finalNoteArray数组中，并向前端传递对应的响应信息。

4.简谱生成模块。主要功能：根据解析结果写简谱。接受的参数为finalNoteArray数组，该数组信息会随解析过程呈现在前端页面上，同时解析完成后可以选择对解析结果进行保存，保存的结果是解析完成后的简谱，存格式可以选择txt/pdf格式。同时简谱信息传入后台，后台数据进行更新，并更新历史记录部分。生成的简谱可以选择加入收藏或删除。若选择加入收藏，则在我的收藏中进行数据更新；若选择删除，则在数据库中删除这一条解析数据。

5.个人中心模块主要可以分为我的收藏模块、历史记录模块、查找简谱模块等子模块。

(1)我的收藏模块。主要功能：保存个人收藏的简谱。从前端获取数据，通过数据url获取相应数据文件，并且返回对应渲染页面。对于已经收藏的简谱，可以选择取消收藏，则在相关的收藏数据后台更新删除后的收藏库。

(2)历史记录模块。主要功能：记录近期解析过的音频简谱。在数据库中

查找一定时间范围内的简谱，将其记录呈现出来。

(3) 查找简谱模块。主要功能：查找在记录中的简谱。根据每个简谱解析完成后的命名，调用底层数据查找的接口，快速定位某一个想要的简谱。

6. 用户账号模块包括用户账号注册、用户账号登陆、密码修改等子模块。

(1) 用户账号注册模块。主要功能：实现用户注册页面的展示，用户创建账号的数据提交与收取，数据检验和数据入库，以及前端反馈等。对应的接口为register函数，可接受前端传入的post数据，进行数据结构检错后调用数据库进行用户数据库的更新，并向前端传递对应的响应信息。

(2) 用户账号登陆模块。主要功能：用户登录页面的展示，用户登录的数据提交与收取，数据检验和数据入库，以及前端反馈等。对应的接口为login函数，可接受前端传入的post数据，进行数据结构检错后调用数据库进行数据的查找，并向前端传递对应的响应信息。

(3) 密码修改模块。主要功能：用户密码修改，后台数据库密码更新，前端反馈等。对应的接口为modify函数，可接受前端传入的post数据，进行数据结构修正后进行数据的更新，并向前端传递对应的响应信息。

完整的系统结构图如下：

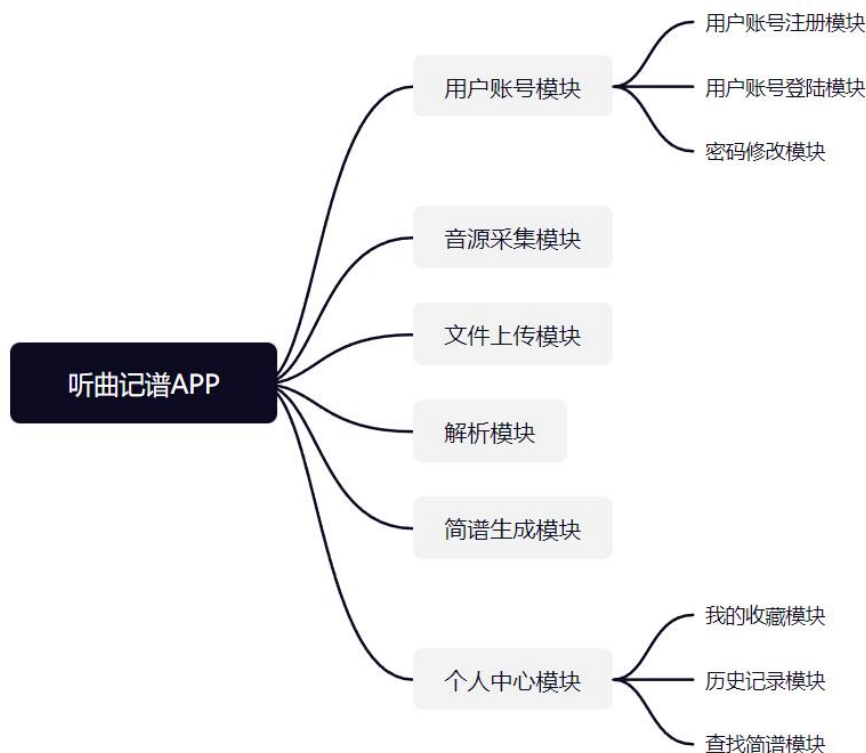


图 3-1 系统结构图

3.2 类图等

系统设计类主要分为各个模块划分，实体类有用户类、简谱类、游客类，方法类有录音类、文件上传类、解析类。

其中用户类继承游客类，同时用户类和游客类在访问简谱类的时候会有权限的差异。而用户类和游客类都可以进行文件上传与录音，同时也可以进行音频的简谱解析，生成简谱，不同地，游客类仅仅对于简谱有查看的权限，而用户不仅能查看生成的简谱，还可以对简谱进行收藏等管理操作。另外，进行社区访问以及互动的权限也只存在用户类中。

录音类和文件上传类主要负责音频的采样，以及采样后解码后传给解析类。解析类则负责音频的解析，解析完成后将结果在简谱类中更新。

系统类图如下所示：

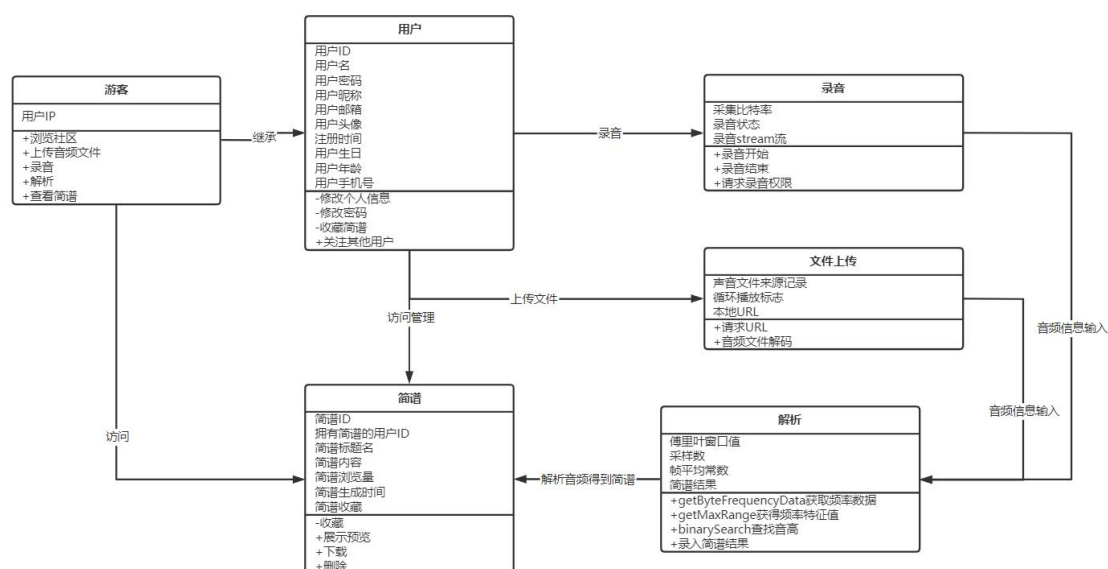


图 3-2 系统类图

3.3 关键数据结构定义

3.3.1 系统功能数据结构

本系统实现音频转简谱的功能，主要处理的是和音频有关的数据，使用到包括音频分析、音频处理等相关的数据结构。

1. 音高相关的数据结构

(1) noteArr

一个浮点数数组，用于存放钢琴 88 音对应的频率值。

(2) keyList

一个字符串数组，用于存放钢琴 88 音对应的简谱表示。

(3) finalNoteArray

一个字符串数组，用于存放解析过程中生成的简谱，数组的每个元素都是音乐某一帧的音高，最终解析完成后的简谱结果保留在该数组中。

2.音源采集模块的数据结构

(1) audioStream

该数据结构记录当前的录音流，其数据项包括：

布尔型变量status，用于记录录音状态；

数组streams，用于存储录音流；

对象blob，用于存储最终生成的录音blob。

相关代码定义如下：

```
const audioStream = {  
  status: false, // 录音状态  
  streams: [], // 用于存储录音stream  
  blob: null, // stream转换成的blob  
}
```

(2) MediaRecorder

MediaRecorder是一个web api接口类，其中定义了有关媒体流采集时的参数，其中：audioBitsPerSecond 用于记录音频采集每秒的比特率，status 表示采集状态。代码定义如下：

```
const MediaRecorder={  
  audioBitsPerSecond: 128000, //音频采集每秒的比特率  
  state: inactive, // inactive 休息,recording 录音中,paused 暂停  
}
```

其实例方法如下，后台可以调用方法控制录音的开始与结束：

```
MediaRecorder.start () //控制录音的开始
```

MediaRecorder.stop () //控制录音的结束

3. 文件上传模块数据结构。

(1) XMLHttpRequest

主要用于文件的载入，其使用过程中部分实例化方法如下。

```
var request = new XMLHttpRequest();
```

```
request.open('GET', url, true); //初始化请求
```

```
request.responseType = 'arraybuffer'; // 设置数据类型为arraybuffer
```

```
var audioData = request.response; //返回一个 ArrayBuffer,存放的是文件内的音频数据
```

```
request.send();
```

(2) AudioBufferSourceNode

主要用于存放音频文件的解码内容，将音频转换为计算机能够理解的语言，其中相关的数据结构有：

数组buffer，用于存放声音文件的来源，记录声音数据；

布尔型变量loop，用于标志是否循环播放声音文件。

代码定义如下：

```
const AudioBufferSourceNode={
```

```
  buffer: null, //存放使用 AudioBuffer 作为声音文件的来源
```

```
  loop: false //是否循环
```

```
}
```

4. 解析模块数据结构。

(1) AnalyserNode

主要用于定义分析音频数据相关算法需要采用的参数值，其中相关的数据结构有：

整型变量fftSize，表示傅里叶变换过程中的傅里叶窗口大小，即频率轴的频率间隔或分辨率；

整型变量frequencyBinCount，表示采样数目，即音频信息数组大小，采样数目决定了两个频率之间的最小差；

浮点型变量smoothingTimeConstant，表示平均常数，在0-1之间，用于设置时间间隔取样的圆滑性。

```
const AnalyserNode={
    fftSize : 4096, //傅里叶窗口大小
    frequencyBinCount: 2048, //采样数，也就是音频信息数组大小
    smoothingTimeConstant: 1 //分析帧的平均常数，在0-1之间
}
```

通过调用 `getByteFrequencyData` 实例化方法，将音频频域数据复制到传入的Uint8Array数组，从而得到想要的频率相关的数据。便于之后的数据提取特征值、分析数据过程。

3.3.2 用户管理数据结构

用户管理主要记录用户信息和简谱信息，处理是和某个用户相关的数据，包括用户的注册，用户对个人简谱中心的访问等。

1.用户信息主要数据：用户的注册和登录，操作简谱。

Uid，用户ID，主码，长整型数据，用来唯一标识一名用户；

Uip，用户IP，字符串数据，用来表示用户的所在地；

Uname，用户名，字符串数据，用于用户登录；

Upassword，用户密码，字符串数据，用于用户登录；

Unickname，用户昵称，字符串数据，作为用户在系统中的自起名；

Umail，用户邮箱，字符串数据，用来表示用户邮箱；

Upphoto，用户头像，字符串数据，用来记录用户头像图片关联的url；

Uregister_time，注册时间，日期数据，用来表示用户注册时间；

Ubirthday，用户生日，日期数据，用于记录用户生日；

Uage，用户年龄，整型数据，用于记录用户年龄；

Uphone，用户手机号，字符串数据，用于记录用户联系方式。

2.简谱信息主要数据：用户可以在系统中添加、删除、收藏简谱。

Nid，简谱ID，主码，长整型数据，用来唯一标识一个简谱；

Nuid, 拥有简谱的用户ID, 外码, 长整型数据, 用来表示拥有该简谱的用户;
Ntitle, 简谱标题名, 字符串数据, 用于表示简谱的名称;
Ncontent, 简谱内容, 长字符串数组数据, 用于记录简谱内容;
Nviews, 简谱浏览量, 整型数据, 用于记录用户浏览简谱次数;
Ndate, 简谱生成时间, 日期数据, 用于记录简谱生成时间;
Nlike, 简谱收藏, 布尔型数据, 用于表示用户是否将该简谱加入个人收藏,
true表示加入, false表示未加入。

3.3.3 数据关联结构设计

前后端数据关联, 分页面介绍如下

register.html页面的前后端数据关联 前端传入Uname、Upasssword和Unickname到用户数据库Uname、Upasssword和Unickname中。

login.html页面的前后端数据关联 前端传入的Uname和Upasssword和用户数据库的数据进行匹配。

Index.html页面的前后端数据关联 后端读取出来的用户信息, 读出所有课程的Unickname, Umail, Uphoto, Ubirthday等用户信息, 然后将相关数据发给前端界面进行渲染展示, 然后对于没有登录的用户, 其历史记录和收藏界面的简谱文件内容显示为空, 对于登录的用户, 在历史记录和收藏界面显示后端数据库中的Ntitle和Ncontent等数据。

个人信息页面的前后端数据关联 用户登录后, 使用当前会话的Uid从数据库中获取一条用户的信息, 前端的name对应后端数据库信息中的Nickname, 前端获取后端的Uphoto并渲染用户头像。

收藏和历史记录页面的前端将当前用户的Uid传入后端, 后端查找数据库中所有简谱Nuid与当前Uid相等的简谱信息发给前端, 如果Nlike = true, 前端将该简谱划分到收藏界面显示, 同时不管Nlike 的值是否为TRUE, 都将其划分到历史记录中, 将Nviews、Ndate、Nlike等信息渲染并显示。

数据关联示意图如图3.3所示

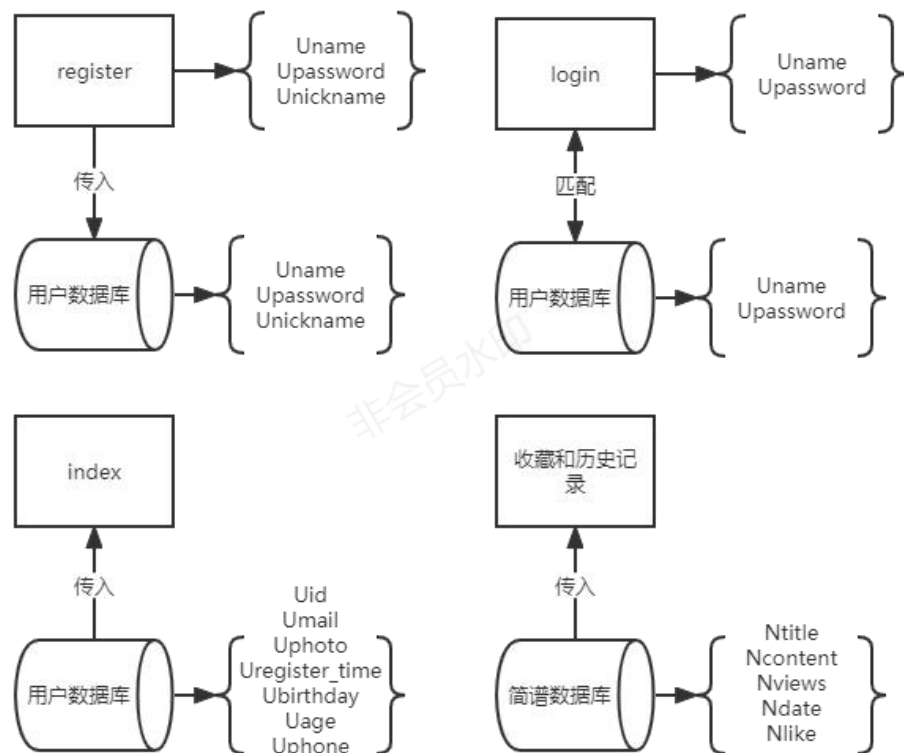


图 3-3 数据关联示意图

3.4 关键算法设计

1. 音频采集模块算法设计。

采集音频首先要创建一个音频记录MediaRecorder对象，获得页面设备的录音权限，之后反复检查“录制”（包含“开始录制”和“结束录制”）按钮是否被点击，若被点击则继续判断MediaRecorder的状态（即是否正在录制）。若录音对象处于休息状态，说明此时点击的按钮事件为“开始录制”，这样需要将之前的录音记录清空，并且请求录音，将录音帧传递到解析模块，同时前端部分要将按钮字样变为“结束录制”。若录音对象处于录音状态或暂停状态，就说明此时点击的按钮事件为“结束录制”，这样需要进行录音结果和解析简谱结果传入前端，前端可以选择保存简谱或试听录音，同时前端部分要将按钮字样变为“开始录制”。

模块的具体流程图如下：

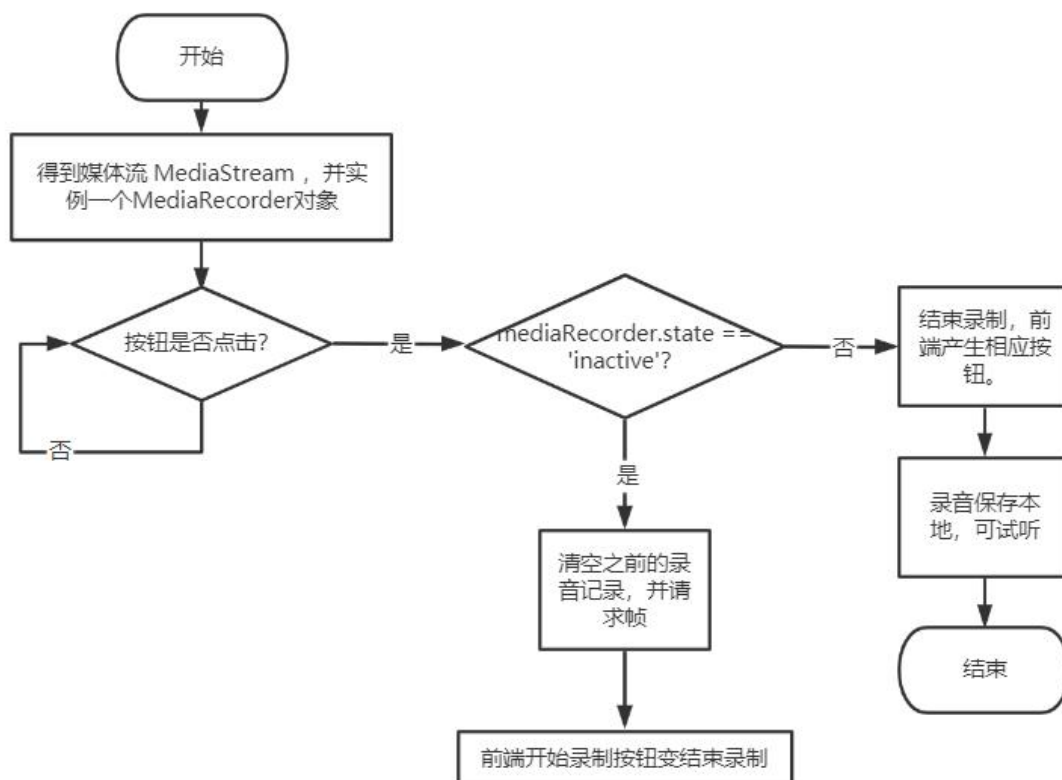


图 3-4 音频采集模块算法设计流程图

2. 文件上传模块算法设计。

文件上传支持从本地读入文件，本地上传的音频文件可以是mp3或wav，上传完成后系统会将该文件与主机的一个URL相关联，这样可以在当前设备上对这个文件的定义。由于上传的是音频文件，所以会有一个对文件解码的过程，将音频文件内的信息写入具体的多个浮点数中，这样就得到了数字版的时域音频信息。之后需要将解码的结果传入AudioBufferSourceNode对象中，可以将AudioBufferSourceNode理解为文件上传模块和解析模块之间的接口，解析模块收到文件上传模块的解码结果后对音频解码结果进行时频转换。

模块的具体流程图如下：



图 3-5 文件上传模块算法设计流程图

3.解析模块算法设计。

解析模块算法是整个系统的关键，其中包括快速傅里叶变换FFT，频率数组数据的初始化，特征值的提取，频率长度（步长）的选取，查找算法等。

首先需要设计的是解析的宏观过程。我们知道，不论是录音还是文件，音频数据总是以帧为单位出现，我们想要在时间上得到最终音频数据的简谱，需要采集某些特定帧时候的音频特征数据。由于一帧的速度很快，而在简谱上的情况可能是某个音占用了好几帧，即在连续的几帧内，获得的频率特征一样，这样可能会导致简谱中音的重复，故采用音频切片的思想，每个特定时间（如1s）对音频采样结果进行分析，而这个特定时间是由系统帧数所决定的（系统帧数通常为60Hz或120Hz，在60Hz情况下，隔60帧采样一次，采样间隔时间就为1s）。

根据以上思想建立解析的大致过程，在解析开始需要有一个AnalyserNode对象，来表示对音频的解析，之后请求一帧，判断时间间隔是否已经到所设置的时间间隔（本系统设为1s），若到时间则进行频率到简谱音高的解析转换，否则继

续请求新的一帧，最后需要通过AudioBufferSourceNode判断音频是否已经解析完成。

模块的具体流程图如下：

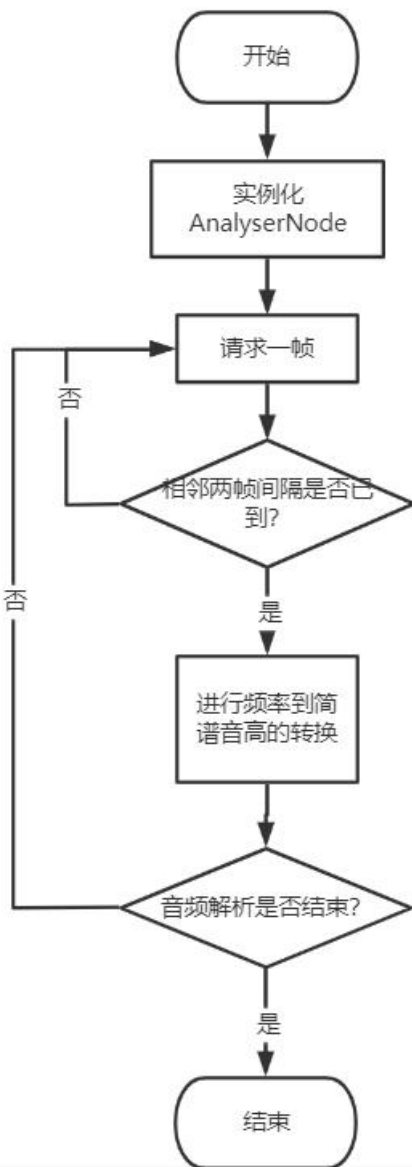


图 3-6 解析模块算法设计宏观过程流程图

频率到简谱音高的解析转换，先要将时间域上的信息转换为频率域上的信息，这里采用的方法为快速傅里叶变换FFT，它是离散傅里叶变换的快速算法，能将时间复杂度降到 $O(n \log_2 n)$ 。这样能够从AnalyserNode对象中提取出频率数据，存放在数组arr中。接下来需要对arr中的频率数据进行初始化，使得数据具有区分度，可以采用min-max方法归一化。之后需要找到那个特征最强的频率分量，由于arr数组中元素的大小代表某一频率的强度，我们需要找到那个最强强度的

数组下标，用index记录。

这样，我们就确定了当前音频率的具体位置，我们接下来要将该频率定位到某一个音高上，首先我们要了解两个相邻数组元素之间间隔的频率步长，我们根据这个步长和index值确定一个最低估计频率和一个最高估计频率，用二者的平均值avgindex来表示当前音的频率。我们可以得到钢琴88音各个音的频率，我们只需要将当前音频率avgindex和88音中的音频率比较，选择最接近avgindex的频率，记录该位置，根据频率和标准简谱音高的标识符对应关系，得到一个简谱符号。特别的，我们得到的音频率可能与标准音频率最低频率还要相去甚远，这种数据应该被丢弃，即放弃当前帧。将符合标准的音标识符写入finalNoteArray中，finalNoteArray内即为最终的简谱，传递给简谱生成模块来进行简谱的展示功能。

模块的具体流程图如下：

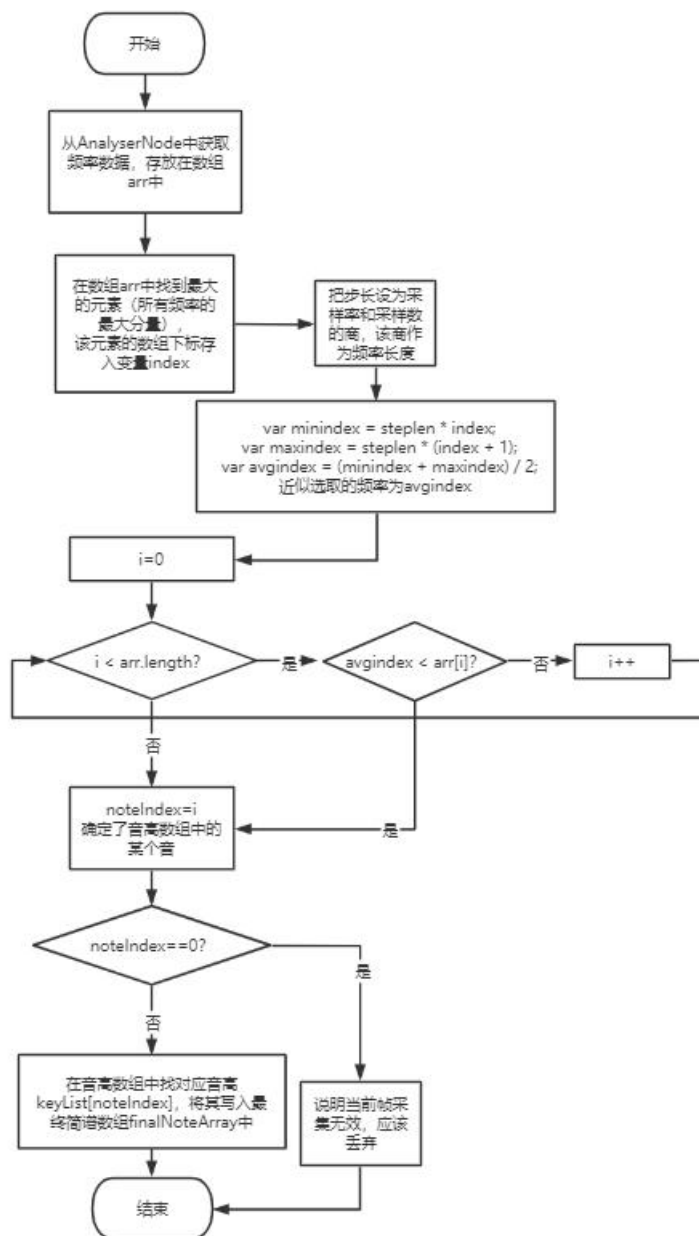


图 3-7 频率到简谱音高的解析转换流程图

4. 简谱生成模块算法设计

简谱生成过程主要是将解析结果音高数组finalNoteArray中的数据可视化, 转换到系统页面上, 同时支持下载等功能。在接受到finalNoteArray后, 判断遍历是否完成, 如在遍历过程中, 直接将当前位置的音传入innerText, innerText再返回前端交互, 显示当前内容, 如果节拍已经到了4个小节 (这里认为一个音出现即为一个小节), 就写入一个“|”符号进行分割。在finalNoteArray遍历结束后需要将innerText内容计入后台, 与当前数据库内容相联系, 进入历史记录模块管

理部分。如果需要下载到本地，处理方式是用blob对象，调用其中msSaveOrOpenBlob方法生成对应文件，文件可以由用户自由命名，并选择下载路径，文件命名后后台数据同步更新。

模块的具体流程图如下：

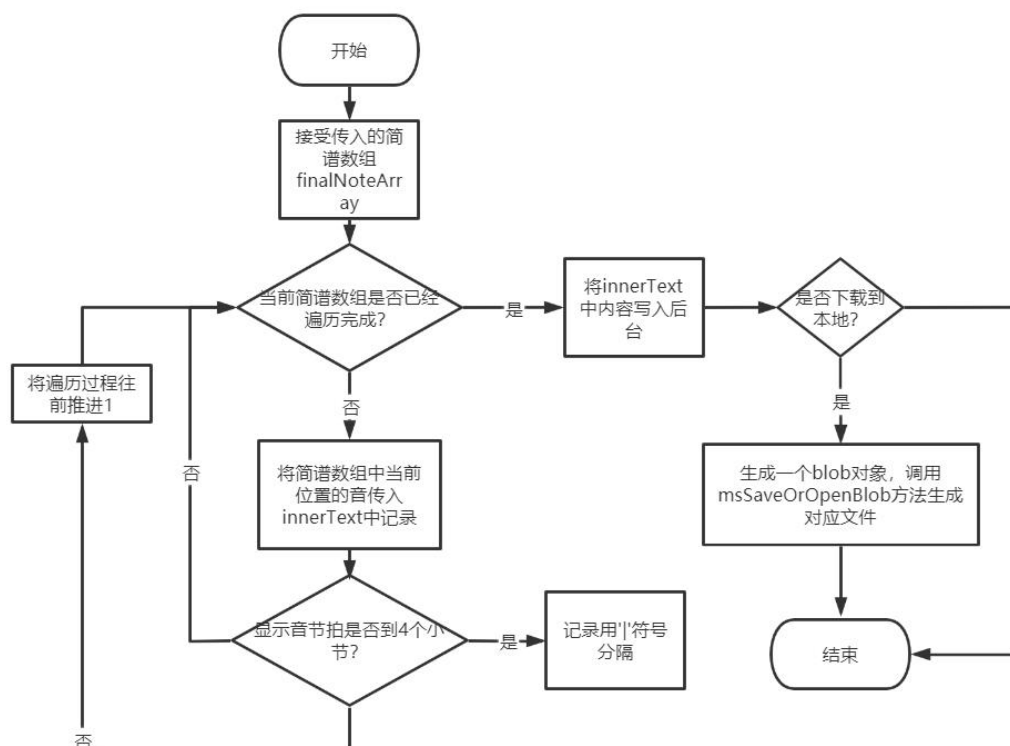


图 3-8 简谱生成模块算法设计流程图

5. 个人中心模块算法设计

个人中心模块主要实现收藏简谱和查找简谱，以及对历史数据的记录。

(1) 收藏简谱

收藏简谱功能包括加入收藏与取消收藏，其中加入收藏操作执行时，关联用户ID和当前简谱ID，将当前简谱表中的简谱收藏数据项置为true，并且将数据库中新信息传递给前端页面，前端根据数据库内容对页面重新更新。执行取消收藏操作时，把当前简谱表中简谱收藏数据项置为false，并交给前端同步更新。

(2) 查找简谱

查找简谱接受输入框中输入的信息，遍历简谱表中所有简谱，将Ntitle表项内容提取出来，同输入的信息一起采用KMP算法进行字符串匹配，如果能够匹配上，说明查找的简谱可能就是当前简谱表中遍历到的简谱，将该简谱信息提取出来传入前端，显示到前端界面上。

(3) 历史记录

查找Ndate在一定范围内的简谱，将这些简谱传给前端显示，同时如果Ndate大于某一时间范围同时Nlike字段值为false，则将这条简谱数据从简谱表中移除。

6.用户账号模块算法设计

用户账号模块主要实现用户的登录，读取用户信息，注册与密码修改服务，其中新注册的用户与修改密码的用户需要在后台数据库中更新相关内容。

模块处理流程如下：

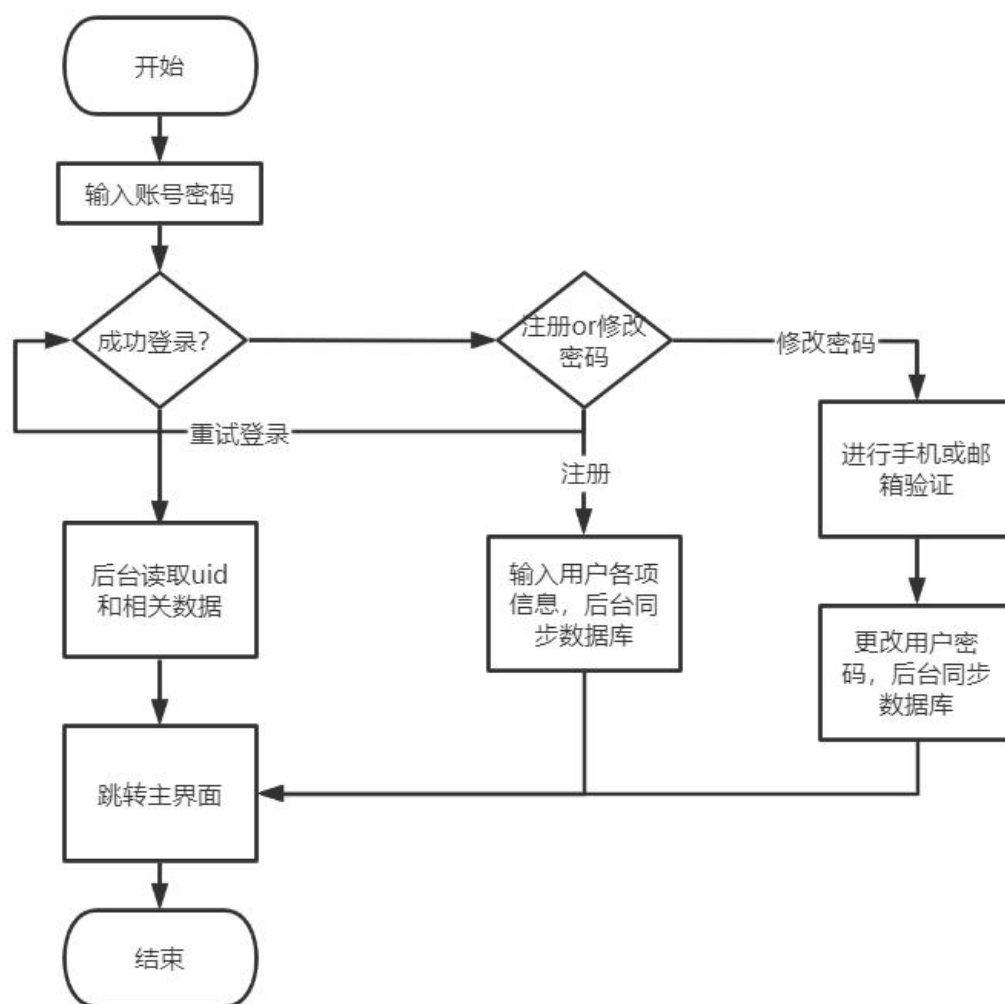


图 3-9 用户账号模块算法设计流程图

3.5 数据管理说明

1. 本地缓存

本地缓存主要存储一些后台代码执行过程中需要经常用到的数据，如钢琴各个音高的频率值以及简谱表示，其中钢琴各个音高的频率值存放在noteArr数组中，简谱标识符存放在keyList数组中，系统运行相关的解析过程时需要将当前音高的频率和标准频率数据建立联系，访问时直接访问对应本地数组即可。

同时最近一次生成的简谱数据也会暂存到finalNoteArray数组中，根据需要可以随时对该数组读取进行查看。

2. 数据库存储

使用单机的Mysql数据库进行数据的存储。MySQL 是最流行的关系型数据库管理系统，在 WEB 应用方面 MySQL 是最好的 RDBMS(Relational Database Management System: 关系数据库管理系统)应用软件之一。MySQL 是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

数据库存储主要是存储每个用户的信息以及解析收藏记录，其中存储形式是以多个表项的形式对用户内容进行存储，而数据库实现和远端服务器相联，访问时需要先获得服务器访问权，之后使用MySql语句中的增删改查语句进行数据访问操作。

4 实现与测试

4.1 实现环境与代码管理

计算机环境：使用的是windows10进行的开发。处理器：Intel (R) Core (TM) i7-10750H @ 2.20Ghz 2.21Ghz 内存 (RAM)：16.0GB (15.9GB可用)

系统类型：64位操作系统，基于x64的处理器。

编程语言：JavaScript

程序框架：前端：html+css+js，后端：nodejs 打包工具：Cordova

IDE：vscode

实现语言：Html5+CSS+JavaScript+Cordova打包

代码管理平台：github

仓库地址：

<https://github.com/GentleWhisperQAQ/app-convert-music-to-sheet>

代码签入记录如下：

何梦杰 (GentleWhisperQAQ)、常梦成 (CMCgithub)

Sep 25, 2022 – Dec 15, 2022

Contributions: Commits ▼

Contributions to main, excluding merge commits and bot accounts

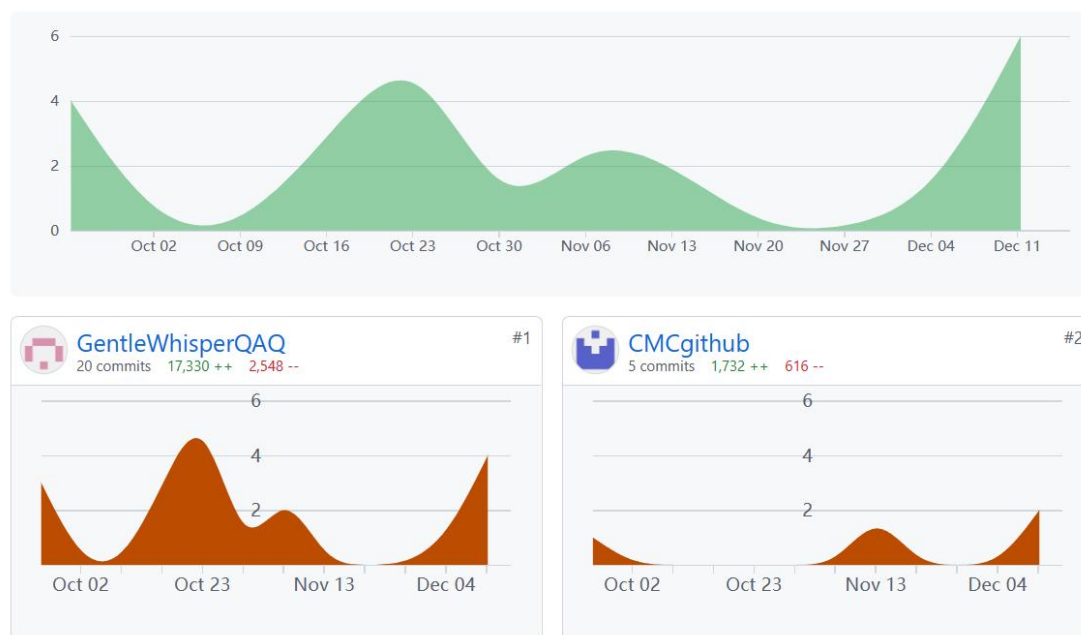


图4-1 GitHub平台代码签入页面

4.2 关键函数说明

网页应用的每个html均可视为独立的模块。本项目主要有 6个不同页面，即 6 个模块组成。其中，index页面为软件主界面，实现软件跳转到其他模块的按钮入口；collection页面为简谱收藏模块，实现用户收藏简谱集的展示；histRecord页面为历史记录模块，实现用户曾经上传或录制的音频简谱的保存和展示；display页面为简谱文件显示模块，用于显示某一简谱文件的详细内容；mic.html为录音模块，实现用户通过录音上传音频文件进行简谱解析的功能；analysis.html为用户上传音频文件并进行简谱解析的模块。

4.2.1 index 模块

功能：作为软件起始页，提供其他功能模块的跳转按钮，并能够根据用户的点击情况实现界面的跳转。

实现：顶部用两个div封装的a标签实现收藏和历史记录的模块跳转，中间显示软件ui设计的logo，底部用两个按钮加事件的方式实现上传文件和录音的模块跳转。有关html界面跳转的函数都在Return.js中实现，需要引用相关函数的html文件通过script标签引用即可。给“本地上传”和“录音”两个按钮分别加onclick事件调用JieXi()和LuYin()两个函数即可实现用户点击是界面跳转。

4.2.2 collection 和 histRecord 模块

功能：显示用户的收藏简谱和历史记录，可以通过右上角的图标实现两页面之间的切换。

实现：Web Storage API 提供了存储机制，通过该机制，浏览器可以安全地存储键值对，根据该API提供的特定域名下的本地存储的功能，调用localStorage对象的方法存储对应简谱文件和其文件路径（URL）的键值对。

4.2.3 display 模块

功能：在页面相应位置显示简谱文件的具体内容。

实现：通过简谱文件和其对应文件路径显示在页面地内容区，函数 `display(filename,url)`接收指定音频文件，并将其内容显示在html页面地对应区域。

4.2.4 mic 模块

1.addEventListener函数

入口参数：

①type表示监听事件类型的大小写敏感的字符串

②listener表示当所监听的事件类型触发时，会接收到一个事件通知对象。

listener 必须是一个实现了 `EventListener` 接口的对象，或者是一个函数。

返回值：无

功能：给录音按钮添加点击监听事件

实现：根据录音状态来确定点击事件发生后进行的操作

2. createMediaStreamSource函数

入口参数：一个MediaStream 对象

返回值：MediaStreamAudioSourceNode对象

功能：创建一个新的MediaStreamAudioSourceNode 对象，然后来自MediaStream 的音频就可以被播放和操作。

实现：获取录音权限成功得到媒体流 MediaStream 并实例一个MediaRecorder对象

3. controlMediaRecorder

功能：此函数在点击事件后触发，用于判断录音状态，控制录音过程

实现：判断当前MediaRecorder状态是否在休息，若在休息就将之前录音清空，开始录音并请求帧；若正在工作则暂停录音，返回解析结果。

4. MediaStreamSourceNode.connect

入口参数：需要连接的 `AudioNode`

返回值：返回入口参数所表示的 `AudioNode` 对象的引用

功能：将一个节点的输出连接到一个指定目标，

实现：这里将我们分析时候要用到的`MediaStreamSourceNode`结点和分析时候用到的`analyser`结点连接起来，便于之后的解析。

4.2.5 audioFile 模块

1. XMLHttpRequest.open

入口参数：

①要使用的 HTTP 方法

②发送请求的 URL

返回值：一个`XMLHttpRequest`对象

功能：初始化一个新的读入文件的请求

实现：用GET方法打开对应的url文件

2. XMLHttpRequest.response

入口参数：无

返回值：请求数据

功能：对相应请求数据类型做出一个响应，保存到返回值中

实现：先设置`responseType`类型为`arraybuffer`类型，调用该函数，返回值存入`audioData`数组中。

3. AudioContext.decodeAudioData

入口参数：

①将会被解码的音频数据

②成功解码后被调用的回调函数。

③可选的错误回调函数。

返回值：一个解码后的对象

功能：将一个音频数据进行解码，解码后返回AudioBuffer对象

实现： 将XMLHttpRequest.response得到的音频文件数据作为被解码的音频数据读入，成功解码后会返回AudioBuffer对象，并且调用playFun函数，对AudioBuffer的各个属性进行初始化。

4. AudioBufferSourceNode.connect

同MediaStreamSourceNode.connect，不再赘述。

4.2.6 analysis 模块

1. requestAnimationFrame

入口参数：一个希望执行的回调函数

返回值：请求 ID，是回调列表中唯一的标识

功能：更新动画，并且在下次重绘之前调用指定的回调函数更新动画

实现：将更新帧的动画放入回调函数中，每次回调函数中会再次调用requestAnimationFrame方法，直到解析完成，flag标识变为1为止，退出帧的递归。

2. freqToNote

入口参数：存放音频数据的数组

返回值：生成的简谱数据

功能：将音频的时域数据变换为频域数据，根据频域数据来找对应音符的音高，最终返回生成带有简谱数据的数组。

实现：首先调用getBytesFrequencyData方法，将音频频域数据复制到传入的Uint8Array数组，调用getMaxRange函数对该数组进行最大振幅频率的获取，该最大振幅频率是由频率步长和各个区间内频率强度决定的，之后将该最大振幅频率和88音高标准频率数组传入binarySearch函数，二分查找得到对应音符数组下标，之后用哈希表查找的方式在keyList中找到对应的音高，写入简谱数组并以简谱形式传入前端进行展示。（此处主要说明调用函数的关系，具体实现方法可见3.4关键算法设计）

4.3 测试计划和测试用例

4.3.1 常用的软件测试方法

根据利用的被测对象信息的不同，可以将软件测试方法分为：黑盒测试、灰盒测试、白盒测试。

1.白盒测试

概念：是依据被测软件分析程序内部构造，并根据内部构造分析用例，来对内部控制流程进行测试，可完全不顾程序的整体功能实现情况，即已知软件产品的内部实现过程，可以通过测试证明每种内部操作是否符合设计规格的要求，所有内部成分是否已经过检查。

思想：白盒测试又被称为玻璃盒测试、透明盒测试、开放盒测试、结构化测试、逻辑驱动测试、是基于程序结构的逻辑驱动测试。

测试对象： 函数、算法与数据结构。

目的：

（1）一般在测试前期进行，通过达到一定的逻辑覆盖率指标，使得软件内部逻辑控制结构上的问题能基本得到消除；

（2）保证内部结构达到一定的覆盖程度，能够给予软件代码质量更大的保证；

（3）白盒测试发现问题后，解决问题的成本较低。

常用技术：

（1）静态分析：包括控制流分析、数据流分析、信息流分析

（2）动态分析：逻辑覆盖测试（分支测试、路径测试等）、程序插装等，逻辑覆盖测试根据覆盖的对象不同，可以分为：语句覆盖、判定（分支）覆盖、条件覆盖、判定条件覆盖、条件组合覆盖、路径覆盖。程序插装指调试程序时，在程序中插入一些打印语句，程序执行时打印出我们关心的信息，通过这些信息了解执行过程中程序的一些动态行为。

2.黑盒测试

概念：把测试对象看成是一个黑盒，只考虑其整体特性，不考虑其内部具体实现过程。即已知产品的需求规格，但不知其内部实现，可以进行测试证明每个需求是否实现。

思想：基于规格的测试，测试类型都来源于质量模型。

测试对象：系统、子系统、模块、子模块、函数等。

常用的黑盒测试方法：等价类划分法、边界值分析法、因果图分析法、判定表法、状态迁移法等。

目的：减少测试时的测试用例数，用尽量少的测试用例完成测试，发现更多的问题。

3.灰盒测试

定义：如果即利用被测对象的整体特性信息，又利用被测对象的内部具体实现信息，采用得就是灰盒测试方法。两种信息占得比例不同，相应的灰度就不同。

适用对象：一般集成测试采用灰盒测试方法。

4.3.2 测试用例

由于web应用的特点，我们难以对各个函数进行逐一测试，因此我们采用黑盒测试的方法，去测试每个页面是否能够正确实现预期的功能。

由于此web应用由六个模块构成，因此对这六个模块均进行一定的测试。

Index模块主要功能是负责主页的渲染和对其他页面的跳转，因此测试时只需考虑用户点击主页的其他模块对应按钮式是否能正确跳转并实现页面的渲染。

测试用例以及测试结果如表4-1所示：

表 4-1 Index 模块测试

测试用例	预期结果	实际结果截图
用户点击我的收藏按钮	页面跳转到 collection.html	

用户点击历史记录	页面跳转到 histRecord.html	
用户点击上传文件	页面跳转到 analysis.html	
用户点击录音	页面跳转到 mic.html	

analysis模块主要实现功能为实现用户上传本地音频文件并对其解析生成简

谱，显示到页面对应内容框，或选择保存简谱为txt文件到本地。因此测试内容分为三个部分，一是用户点击选择文件按钮后是否能正常弹出windows资源管理器；二是用户上传后的音频文件能否被正确解析并显示生成的简谱内容；三是用户选择保存简谱生成结果为文件时可以将简谱内容下载到本地。测试用例以及测试结果如表4-2所示：

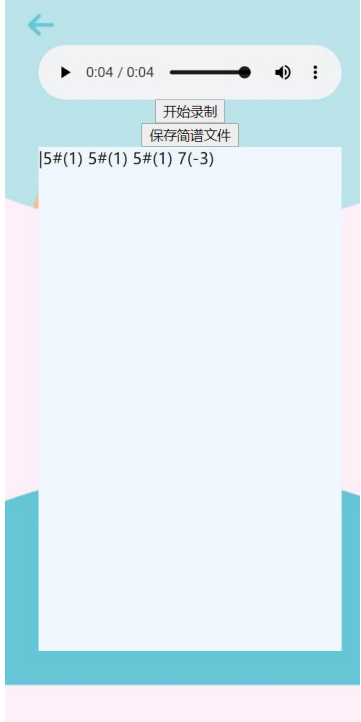
表 4-2 analysis 模块测试

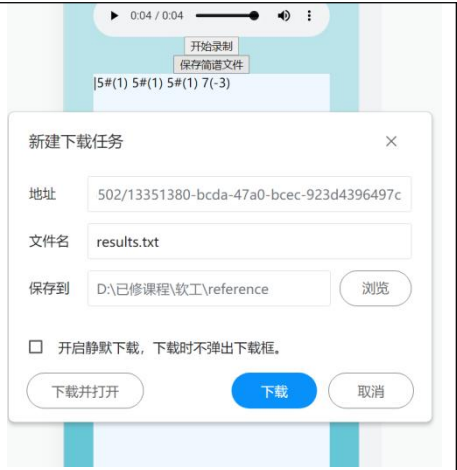
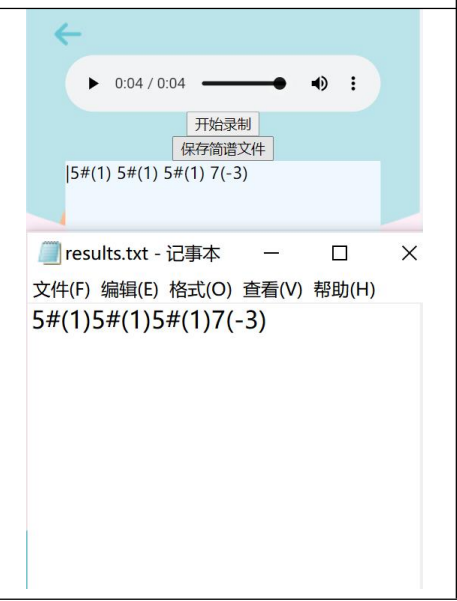
测试用例	预期结果	实际结果截图
用户点击选择文件按钮	弹出 WINDOWS 资源管理器	
用户点击选择音频文件后	在页面对应区实时显示简谱生成的结果,结束时弹出提示框	

		127.0.0.1:5502 显示 简谱生成好了 <div>确定</div>
用户点击 create file 按钮	浏览器弹出下载提示框，默认文件名为 result.txt	<div>新建下载任务 ×</div> <div>地址 I:5502/b8bfd311-3f7e-4f1c-9f39-9d8269d38f2f</div> <div>文件名 results.txt</div> <div>保存到 D:\已修课程\软工\reference <div>浏览</div></div> <div><input type="checkbox"/> 开启静默下载，下载时不弹出下载框。</div> <div><div>下载并打开</div><div>下载</div><div>取消</div></div> <div><div>选择文件 night.wav</div><div>保存</div><div>Create file</div></div>

Mic录音模块实现的功能是通过mic采集音频并解析生成简谱，保存简谱结果到本地文件，测试用例分为三个部分：一是录音功能是否正常，录音后能否生成简谱结果；二是能否保存简谱结果到本地文件；三是文件内容是否以实际结果一致。测试用例以及测试结果如表4-3所示：

表 4-3 Mic 录音模块测试

测试用例	预期结果	实际结果截图
用户点击开始录制按钮，一段时间后点击结束录制	结束录制后网页自动对录制的音频进行解析，并将生成的简谱显示到页面对应内容框。	

用户点击保存简谱文件按钮	浏览器弹出下载框，文件名默认为result.txt	
用记事本打开保存的简谱文件	文件内容与网页实际显示的解析结果一致	

4.4 结果分析

由上述各测试结果来看，软件的各个页面都能够正确实现预期的功能，可以正确实现mic采集录音，对录音进行频谱分析并输出简谱文件，各页面设置的功能按钮能够正常发挥作用。APP实现了设计目标，即能够实现简单的录音及简谱解析功能并提供图形化展示界面，同时web应用的实现方式保证了系统的易用性和系统的轻量化。

5 总结

5.1 用户反馈

当第一代听曲记谱的 APP 发布后，向周围的好友推广了我们的小程序。这些用户在一段时间的使用后，为我们反馈了一些使用截图，如图 5-1 所示。



图 5-1 用户使用截图

另外，我们也收到了来自这些用户的反馈，我们选取了两条用户反馈信息。如图 5-2 和图 5-3 所示。

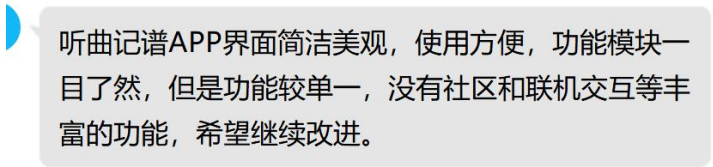


图 5-2 用户反馈 1

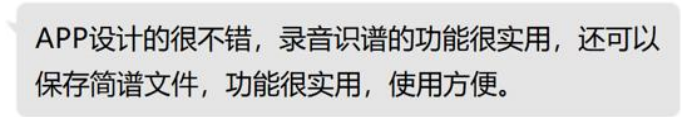


图 5-3 用户反馈 2

在整合用户反馈后，我们进行了规整，我们的 app 主要有以下优点：

1. 界面简洁，UI 设计美观。
2. 功能实用，使用方便，录音和选择上传文件的功能齐全。
3. 体量小，不占内存，运行快速，体验很好。

同时我们也通过用户使用后的反馈发现了我们的 app 需要改进和完善的地方：后续可以尝试加入社区模块，用户可以在社区讨论和分享内容，形成一定的用户群体和营造良好的社区氛围以促进我们的 app 有更好的发展前景。此外软件的 ui 设计还可以继续完善。

根据以上用户的反馈，我们对界面进行了优化调整。并考虑在之后的版本上线社交功能。

5.2 全文总结

何梦杰 总结

在软件工程的整个项目中，我作为三人小组的组长，和我的其他两名组员讨论了软件工程实现的过程，根据各个同学的能力合理划分了任务，通过我们共同的努力，可能完成度不是很高，但还是最终完成了听曲记谱 APP 的项目。

我在小组里主要负责系统设计，UML 图的绘制，系统逻辑与后台实现，各类算法的实现，以及代码测试等工作，同时也负责对项目文档以及最终报告的整理与编写。具体工作如下：

- （1）完成了选题、项目可行性的分析，完善了 NABCD 模型。
- （2）完成了系统任务在组员之间的分配。
- （3）绘制了 UML 图，对系统进行了需求分析。
- （4）完成了原型的部分设计。
- （5）初步设计了各个模块实现流程。
- （6）参与编码 js 部分，使用 Javascript 语言编写了系统的解析算法逻辑、文件上传逻辑、录音逻辑以及简谱保存逻辑。
- （7）使用 Cordova 工具对项目进行打包。
- （8）对代码运行效果进行了测试，在不满意的地方进行修改。
- （9）完成了项目文档的编写。
- （10）完成了项目报告的整理与编写。

常梦成 总结

在本次项目开发中，我主要负责前端页面 UI 设计和界面渲染的实现。以 html+css 为主要程序语言，主要负责各个页面模块的渲染呈现和页面跳转逻辑关系设计，同时最后负责进行用户反馈收集和整合，反馈给团队，进行后续的版本迭代。具体工作如下：

在前端页面设计时对原型系统进行了调整；

完成了各页面的跳转逻辑设计与实现。

完成了各个页面的 UI 设计和交互设计与实现。

完成了各个页面的组件布局和 css 渲染。

完成了用户反馈信息的收集和整合。

后期项目迭代时对 UI 进行调整。

徐海林 总结

（1）与小组成员一起完成了项目背景和可行性的调查和分析，初步编写了 NABCD 模型。

（2）完成原型系统的设计，完成了原始的界面设计和跳转逻辑设计。

（3）完成了各页面的初始参考 UI 设计和交互引导设计。

（4）与组员配合，完成最终的实现

6 体会

何梦杰同学，分工：需求分析，原型部分设计，系统模块设计，系统实现，js 代码编写，代码测试，文档与报告撰写。

体会：

此次软件工程项目是我第一次实际参与到的大型软件开发工程，遇到了不少困难，但也有不小的收获。由于之前没有软件开发的经历，可能在各个方面都比较缺乏经验，再加上我又担任了这次软件工程小组的组长，因此我感觉到完成过程比较吃力并且压力有些大。但是换个角度来看，虽然过程困难重重，但有些问题也铸就了我的成长，不仅锻炼了我的设计能力，编码能力，还让我理解了一个软件开发的全流程，激发了我学习技术的动力。

在项目的一开始，我们对选题问题就开始了很长一段时间纠结，怕题目太简单没有挑战，同时又怕题目太难做不出来，综合考虑，我们才选择了听曲记谱这一项目作为我们软件工程的选题。由于以前都是一个人写代码，初次接触这种多人协作的项目，对 Git 这种代码管理工具比较陌生，更不知道如何与组员们进行对接。我通过上网查资料的方式学习了 Git 的几种基本用法，了解了代码 push 的一般步骤，这为我们后面代码管理打下了基础。

项目开发之前需要做一些需求分析与系统设计，我结合老师上课所授和具体的项目，针对具体系统做了 UML 用例图、顺序图、E-R 图、类图的绘制，根据分析到的用户需求对系统功能做了初步的规划。我原本想的是能够支持不同乐器的识别，但是由于我们采用开发的语言为 html+css+js，在考虑之下还是只是选择了简单的钢琴曲进行支持识别。在系统算法的设计过程中也是困难重重，一开始我想的是能不能运用 python 中机器学习方法对音频进行学习得到简谱，后来发现无法掌握 python 与前端语言之间的接口连接，于是我换了一个能够用 js 实现的解析算法，即利用傅里叶变换来得到频域数据，根据数据特征得到最终简谱。我还结合数据库课堂的学习内容设计了一个用户-简谱的简易管理系统，并且设计出了其中运行的逻辑。

在项目的实际开发阶段，我是初次使用 JavaScript 语言编写程序，其中各种语法和方法我都不太了解，但是好在有其他高级语言的基础，我在学习 JavaScript 这门新语言时能够借鉴其他如 C++、Java 语言中对象、类的概念，同时通过查看

官方文档的方式来学习和音频处理有关的相关函数方法。利用这些方法，我一步步从零开始构建了一个音频解析系统，从一开始音频的采集，音频帧的选取，到傅里叶窗口的概念理解，解析时特征频率选取的过程，我边做边学完成了系统的部分实现。在这之中我也出过不少 bug，但我也是通过上网搜索和询问大牛同学的方式解决了。在逻辑的编写过程中，我也对钢琴 88 音的理解更加深刻了，知道了钢琴分为多个音高声部大调，每个大调含有 7 个音，这还丰富了我除软件开发之外的乐理知识。

理想中的设计和现实还是有一定差异，一开始计划把 web 用 Cordova 打包移植到移动端，但测试后发现部分功能因为设备的差异而出现异常。功能方面虽然设计了很多内容，但因为第一次接触这种项目，自己技术的匮乏，再加上课程给的实现时间实在太有限，很多功能如用户管理等内容还有待完善。不过我还是感觉有不少收获，首先是收获到了团队的力量，我知道了一个软件的开发绝不仅仅是一个人的事情，而是由大家一起共同完成的，因为软件开发一定是多阶段的，再者就是锻炼了我的领导能力，让我学会了任务的合理分配，团队间的协调分配可以帮助每个成员扬长避短，最关键的还是我掌握了 git 的基本操作，学习了一门新语言 js，同时也了解了打包工具的使用，增强了我的编码能力，激发了我的技术热情。

总而言之，我在本次项目中收获良多，感谢同组成员对我的支持，感谢老师对我们进行软件开发流程的指导，我在这次项目中提升了不少，同时也积累了一份可贵的软件开发经验。

常梦成同学，分工：前端页面 UI 设计与实现，

体会：

我在这次软件项目中主要负责前端界面实现，由于以前只接触过一些前端的基础知识，对于前端的各种框架和接口都不甚了解，且是第一次参与这样的规模的软件项目实现，所以遇到了很多以前从未遇到的困难，最后实现的页面效果不是很协调，但这也算是从无到有的突破了。

从一开始和室友讨论选题，到确定选题后开始确定项目的系统结构，主要用到哪些技术和程序语言，然后一步步查找各种开源项目，技术文档，不断学习摸索，最终实现了构想的功能实现策略。一开始我们计划以 java 为主要语言编写

安卓平台的手机 APP，但由于一直没找到相关的突破口，且对 android studio 平台的软件开发没有任何经验，最后决定用已经有所了解的 WEB 页面开发技术进行软件开发，也通过互联网了解到可以通过 node 的打包工具 cordova 将 web 应用打包为可以在 android 平台安装运行的 APK。确定技术方向后我们就开始了软件的实现，在参照原型系统进行 web 页面设计时，我尽可能通过所学的知识去设计 UI 并实现 CSS 渲染，中间也是对页面的每个组件的渲染改了很长时间。对于页面的交互功能按钮，我在与队友不断沟通的情况下最终实现了各个功能函数的调用，并最终达到了理想的效果。作为我们的第一个软件项目，我们在得到用户反馈后也知道了项目设计实现的诸多不足和需要完善之处，通过后续不断完善页面和项目功能，最终实现的项目也是收到了用户的好评和赞赏。

总之在花费了我大量时间和精力，和室友的充分沟通协作的情况下，我还是感觉小有成就，学习到了很多技术，增加了项目经验，了解的软件设计与开发的主要工作流程和相关的专业知识，收获满满。

徐海林同学，分工：原型系统设计，页面交互和跳转逻辑，UI

体会：随着我们设计实现的软件的成功运行，这次的软件工程课程就告一段落了，但是这门课带给我的远不止如此。软件工程课程是我第一次接触和参与软件项目的开发，过程中遇到了很多的困难。

在项目开发初期的选题阶段，不同选题的内容差异大，难度不一，给我们造成了不小的困惑，由于人数的限制，我们既要选择一个体量适中的软件项目进行开发，也要考虑的软件实际开发出来以后的实用性和易用性。好在经过老师在课堂上的讲解，我学习到了如用例图、UML 图等一系列系统工具，通过这些工具，我们可以了解到软件的层次结构、模块之间的配合调用等具体动作，这对于我们选题和评估起到了很重要的作用。

在原型系统设计中，我也遇到了不少的困难，体会到了设计的不易。在我看来，软件界面设计对于用户使用一款软件显得尤为重要。美观简洁的界面，自然的引导，科学的按键设置都是决定一款软件质量的因素之一。页面的具体设计也同样如此，要在保证功能的前提下尽量考虑美观。所以在设计原型系统的过程中，我通过墨刀工具的素材搜索功能，找到了很多优质的素材并进行了使用。在页面的跳转逻辑设计的过程中，我也走了很多弯路，一个页面的前一个页面是哪一个，

后一个页面又该跳转在哪，怎么样设计才显得科学，方便用户的使用。比如一开始我只在主页面设置了一个选项按钮，但是在后来的实际操作中发现，当用户解析完一个简谱后，有可能有查看其他简谱的需求，这时如果按照之前的设计思路，用户只能逐级点击返回按钮到达主页面来找到选项按钮查看我的收藏和历史记录，这样的设计显然是不科学的，用户使用起来会有不必要的操作。于是我在简谱展示页面也增加了一个选项按钮，这样就可以直接在此查看历史记录和我的收藏。

最后，这次的软件工程项目是一个团队任务，在这里感谢两位队友的帮助和共同努力，我也明白了团队合作配合的重要性。万事开头难，我相信这次的软件开发经历会成为日后工作生活的宝贵经验。

附录

```

1.录制解析功能 mic.js
//钢琴各个音的频率数组
const noteArr = [27.5, 29.135, 30.868, 32.703, 34.648, 36.708, 38.891, 41.203,
43.654, 46.249, 48.999, 51.913,
    55, 58.27, 61.735, 65.406, 69.296, 73.416, 77.782, 82.407, 87.307, 92.499,
97.999, 103.826,
    110, 116.541, 123.471, 130.813, 138.591, 146.832, 155.563, 164.814,
174.614, 184.997, 195.998, 207.652,
    220, 233.082, 246.942, 261.626, 277.183, 293.665, 311.127, 329.629,
349.228, 369.994, 391.995, 415.305,
    440, 466.164, 493.883, 523.251, 554.365, 587.33, 622.254, 659.255,
698.456, 739.989, 783.991, 830.609,
    880, 932.328, 987.767, 1046.502, 1108.731, 1174.659, 1244.598, 1318.52,
1396.913, 1479.978, 1567.982, 1661.219,
    1760, 1864.655, 1975.533, 2093.004, 2217.461, 2349.318, 2489.016,
2637.02, 2793.826, 2959.955, 3135.437, 3322.437,
    3520, 3729.31, 3951.066, 4186.009]
//这里将钢琴中央 C 大调定为 1
const keyList = ['6(-4)', '6#(-4)', '7(-4)', '1(-3)', '1#(-3)', '2(-3)', '2#(-3)', '3(-3)',
'4(-3)', '4#(-3)', '5(-3)', '5#(-3)',
    '6(-3)', '6#(-3)', '7(-3)', '1(-2)', '1#(-2)', '2(-2)', '2#(-2)', '3(-2)', '4(-2)', '4#(-2)',
'5(-2)', '5#(-2)',
    '6(-2)', '6#(-2)', '7(-2)', '1(-1)', '1#(-1)', '2(-1)', '2#(-1)', '3(-1)', '4(-1)', '4#(-1)',
'5(-1)', '5#(-1)',
    '6(-1)', '6#(-1)', '7(-1)', '1', '1#', '2', '2#', '3', '4', '4#', '5', '5#',
'6', '6#', '7', '1(1)', '1#(1)', '2(1)', '2#(1)', '3(1)', '4(1)', '4#(1)', '5(1)', '5#(1)',
'6(1)', '6#(1)', '7(1)', '1(2)', '1#(2)', '2(2)', '2#(2)', '3(2)', '4(2)', '4#(2)', '5(2)',
'5#(2)',
    '6(2)', '6#(2)', '7(2)', '1(3)', '1#(3)', '2(3)', '2#(3)', '3(3)', '4(3)', '4#(3)', '5(3)',
'5#(3)',
    '6(3)', '6#(3)', '7(3)', '1(4)', '1#(4)', '2(4)', '2#(4)', '3(4)', '4(4)', '4#(4)', '5(4)',
'5#(4)']

// 获取录音按钮
const recordBtn = document.querySelector('#record')
// 获取 audio
const audio = document.querySelector('#audio')

const audioStream = {
    status: false, // 录音状态
    streams: [], // 用于存储录音 stream
    blob: null, // stream 转换成的 blob
}

// MediaRecorder 实例
let mediaRecorder = null

```

```

var finalNoteArray = new Array();
var audioCtx = new (window.AudioContext || window.webkitAudioContext)();
const analyser = audioCtx.createAnalyser();

// 给录音按钮添加点击事件
recordBtn.addEventListener('click', async (e) => {
    // 判断录音状态
    if (audioStream.status) {
        controlMediaRecorder()
    } else {
        // 判断 mediaRecorder 是否存在
        mediaRecorder ? controlMediaRecorder() : getpermission()
    }
})

// 获取权限
function getpermission() {
    // audio 音频 | video 视频
    const constraints = { audio: true, video: false }
    navigator.mediaDevices.getUserMedia(constraints).then((MediaStream) =>
{
    // 获取成功 得到媒体流 MediaStream 并实例一个 MediaRecorder
对象

    mediaRecorder = new MediaRecorder(MediaStream);
    mediaRecorder.addEventListener('dataavailable', onDataavailable)
    mediaRecorder.addEventListener('stop', onStop)

    const source = audioCtx.createMediaStreamSource(MediaStream);
    source.connect(analyser);

    controlMediaRecorder()
}).catch((error) => {
    // 获取失败
    console.error(error);
})
}

var arr = new Uint8Array(analyser.frequencyBinCount); // 用于存放音频数据的
数组，其长度是 fftsize 的一半
var requestAnimationFrame = window.requestAnimationFrame ||
    window.webkitrequestAnimationFrame ||
    window.mozrequestAnimationFrame; // 兼容

let i = 1; // 记录 requestAnimationFrame 的执行次数（屏幕刷新次数）
let time = 60 // 每 time 帧转换一次音
function fn() {
    if ((i % time) == 0) { // 计时器触发时，执行功能函数
        freqToNote(arr)
    }
}

```

```

    }
    i++;
    if (mediaRecorder.state === 'recording') { //若正在录音则请求帧
        requestAnimationFrame(fn);
    }
}

// 控制 MediaRecorder
function controlMediaRecorder() {
    // 判断录音状态
    // inactive 休息 | recording 录音中 | paused 暂停
    if (mediaRecorder.state === 'inactive') {
        // 开始录制将之前的录音清空
        audioStream.streams = []
        // 释放内存
        if (audioStream.blob) {
            URL.revokeObjectURL(audioStream.blob)
            audioStream.blob = null
        }

        mediaRecorder.start(1000)
        finalNoteArray = new Array();
        requestAnimationFrame(fn); //请求帧
        recordBtn.innerText = '结束录制'
        audioStream.status = true
        console.log("开始录制---");
    } else {

        mediaRecorder.stop()
        //一次录音解析简谱结果在 finalNoteArray 中，在此处可以进行提取
        console.log(finalNoteArray)
        Display(finalNoteArray)
        recordBtn.innerText = '开始录制'
        audioStream.status = false
        console.log("结束录制---");
    }
}

// 监听 stop 事件
function onStop() {
    // 将 audioStream.streams 转换为地址
    audioStream.blob = new Blob(audioStream.streams, { type:
audioStream.streams[0].type })
    audio.src = URL.createObjectURL(audioStream.blob)
}

```



```

/**
 * 监听录音 dataavailable 事件
 * 触发条件
 * 1. 媒体流结束时
 * 2. 调用 stop()
 * 3. 调用 requestData()
 * 4. 调用 start(timeslice) 每隔 timeslice 毫秒触发一次 dataavailable 事件
 */
function onDataavailable(event) {
    // event.data blob 对象
    audioStream.streams.push(event.data)
}

function getMaxRange(arr) {
    //求幅度最大频率区间
    var maxx = arr[0]
    var index = 0
    for (var i = 0; i < arr.length - 1; i++) {
        if (maxx < arr[i + 1]) {
            maxx = arr[i + 1]
            index = i + 1
        }
    }
    var steplen = audioCtx.sampleRate / 2048
    var minindex = steplen * index
    var maxindex = steplen * (index + 1)
    var avgindex = (minindex + maxindex) / 2
    var value = new Array(minindex, maxindex)
    return avgindex
}

function freqToNote(arr) {
    //由频率数组得到乐谱
    analyser.getByteFrequencyData(arr);// 将音频频域数据复制到传入的
    Uint8Array 数组
    var maxEnergyFreq = getMaxRange(arr) // 获取最大振幅频率
    console.log("maxEnergyFreq:"+maxEnergyFreq)
    var noteIndex = normalSearch(noteArr, maxEnergyFreq) // 获取对应音符
    下标
    console.log("noteIndex:"+noteIndex)
    if (noteIndex != 0) {
        console.log("keyList[noteIndex]:" + keyList[noteIndex]) // 打印对应
        音符下标
        finalNoteArray.push(keyList[noteIndex])
    }
}

```

console.log("finalNoteArray:"+finalNoteArray) //一次录音解析简谱
结果在 finalNoteArray 中

```

    }
}

function normalSearch(arr, sel) {
    var i = 0
    for (i = 0; i < arr.length; i++) {
        if (sel < arr[i]) return i
    }
    return i
}

```

```

function binarySearch(arr, sel) {
    //首先确定首、尾下标
    var low = 0;
    var high = arr.length - 1;
    while (low <= high) { //只要查找区间起始点和结束点中间还有值(要包
        括两值相同的情况), 我们就继续进行查找
        var mid = (low + high) / 2; //确定中间值下标
        if (sel == arr[mid]) { //如果查找值等于中间值
            return mid //则这个 mid 值, 就是查找到的数组下标
        } else if (sel < arr[mid]) { //如果查找值小于中间值
            high = mid - 1; //则在左半部分查找, 需要重新确认区间 high 的
            位置
        } else { //否则查找值大于中间值
            low = mid + 1 //则在右半部分查找, 需要重新确认区间 low 的
            位置
        }
    }
    low = low - 1
    return low //查找完都没有查找到, 就退出
}

```

```

function download(text, name, type) {
    var a = document.getElementById("a");
    var file = new Blob([text], { type: type });
    a.href = URL.createObjectURL(file);
    a.download = name;
    a.dispatchEvent(new MouseEvent('click', { 'bubbles': false, 'cancelable':
true }));
}

```

```

function downLoadDataToLoc() {
    var blob = new Blob(finalNoteArray, { type: 'text/plain' })
    // 创建一个 blob 的对象, 把 Json 转化为字符串作为我们的值
    if ("msSaveOrOpenBlob" in navigator) {

```

```

        // 这个判断要不要都行，如果是 IE 浏览器，使用的是这个，
        window.navigator.msSaveOrOpenBlob(blob, "results.txt");
    } else {      // 不是 IE 浏览器使用的下面的
        var url = window.URL.createObjectURL(blob)
        // 上面这个是创建一个 blob 的对象链接，
        var link = document.createElement('a')
        // 创建一个链接元素，是属于 a 标签的链接元素，所以括号里才
是 a，

```

```

        link.href = url;
        // 把上面获得的 blob 的对象链接赋值给新创建的这个 a 链接
        link.setAttribute('download', "results.txt")
        // 设置下载的属性（所以使用的是 download），这个是 a 标签的
一个属性
        // 后面的是文件名字，可以更改
        link.click();
        // 使用 js 点击这个链接
    }
}

```

```

function Display(arr) {
    console.log("display");
    const rel = document.getElementsByClassName("content")[0];
    let Rstr = "";
    let i=0
    for (i = 0; i < arr.length; i++) {
        var str = arr[i]
        console.log(str)
        if (i % 4 == 0) {
            Rstr += '|';
        }
        if (i % 8 == 0 && i != 0) {
            Rstr += '\n+';
        }
        Rstr += str + ' ';
    }
    // while(i%4!=0){
    //     Rstr += '0' + ' ';
    //     i++
    // }
    // Rstr += '|';
    rel.innerText = Rstr;
    console.log(Rstr)
}

```

2.文件上传解析功能 audioFile.js

```

var flag = 0;
var finalNoteArray = new Array();
var audioCtx = new (window.AudioContext || window.webkitAudioContext)();

var AudioBufferSourceNode = audioCtx.createBufferSource();

document.getElementById("loadfile").onchange = function () {
    var file = this.files[0];
    var fr = new FileReader();

    fr.onload = function (e) {
        audioCtx.decodeAudioData(
            e.target.result,
            function (buffer) {
                playFun(buffer); // 解码后返回的 AudioBuffer 对象作为播放函数的
参数传入
            },
            function (err) {
                console.log(err);
            }
        );
    };
    fr.readAsArrayBuffer(file);
};

function getData() {
    var request = new XMLHttpRequest();
    request.open('GET', url, true);
    request.responseType = 'arraybuffer'; // 设置数据类型为 arraybuffer
    request.onload = function () {
        var audioData = request.response;
        audioCtx.decodeAudioData(audioData, function (buffer) {
            playFun(buffer);
        },
        function (e) { "Error with decoding audio data" + e.err });
    }
    request.send();
}

function playFun(buffer) {
    AudioBufferSourceNode.buffer = buffer; // AudioBuffer 数据赋值给 buffer 属
性
    //AudioBufferSourceNode.connect(audioCtx.destination); // 如果只是播放音
频，这边就直接将 AudioBufferSourceNode 连接到 AudioDestinationNode
    AudioBufferSourceNode.connect(AnalyserNode); // 实现播放后，需要将
bufferSourceNode 连接到 AnalyserNode，才能通过 AnalyserNode 获取后面所需的
数据
    AudioBufferSourceNode.loop = false; // 循环播放，默认为 false
    AudioBufferSourceNode.start(0); // 开始播放音频

```

```

    AudioBufferSourceNode.onended = function () {
        //播放完成
        flag = 1;
        console.log("播放完成");
        alert("简谱生成好了");
        window.cancelAnimationFrame(myReq); //跳走
    };
}

var AnalyserNode = audioCtx.createAnalyser();

var arr = new Uint8Array(AnalyserNode.frequencyBinCount); //用于存放音频
数据的数组，其长度是 fftsize 的一半

requestAnimationFrame =
    window.requestAnimationFrame ||
    window.webkitrequestAnimationFrame ||
    window.mozrequestAnimationFrame; //兼容
let i = 1; // 记录 requestAnimationFrame 的执行次数（屏幕刷新次数）
let time = 60; // 每 time 帧转换一次音
function fn() {
    //    setInterval(freqToNote(arr),5000)
    if (i % time == 0) {
        //计时器触发时，执行功能函数
        freqToNote(arr);
    }
    i++;
    if (flag == 0) {
        requestAnimationFrame(fn);
    }
}
var myReq = requestAnimationFrame(fn);

function getMaxRange(arr) {
    //求幅度最大频率区间
    var maxx = arr[0];
    var index = 0;
    for (var i = 0; i < arr.length - 1; i++) {
        if (maxx < arr[i + 1]) {
            maxx = arr[i + 1];
            index = i + 1;
        }
    }
    var steplen = audioCtx.sampleRate / 2048;
    var minindex = steplen * index;
    var maxindex = steplen * (index + 1);
    var avgindex = (minindex + maxindex) / 2;
    var value = new Array(minindex, maxindex);
    return avgindex;
}

```

```

    }

    function normalSearch(arr, sel) {
        var i = 0;
        for (i = 0; i < arr.length; i++) {
            if (sel < arr[i]) return i;
        }
        return i;
    }

    function binarySearch(arr, sel) {
        //首先确定首、尾下标
        var low = 0;
        var high = arr.length - 1;
        while (low <= high) {
            //只要查找区间起始点和结束点中间还有值(要包括两值相同的情况),我们就继续进行查找
            var mid = (low + high) / 2; //确定中间值下标
            if (sel == arr[mid]) {
                //如果查找值等于中间值
                return mid; //则这个 mid 值, 就是查找到的数组下标
            } else if (sel < arr[mid]) {
                //如果查找值小于中间值
                high = mid - 1; //则在左半部分查找, 需要重新确认区间 high 的位置
            } else {
                //否则查找值大于中间值
                low = mid + 1; //则在右半部分查找, 需要重新确认区间 low 的位置
            }
        }
        low = low - 1;
        return low; //查找完都没有查找到, 就退出
    }

    function freqToNote(arr) {
        //由频率数组得到乐谱
        AnalyserNode.getBytesFrequencyData(arr); // 将音频频域数据复制到传入的
        Uint8Array 数组
        var maxEnergyFreq = getMaxRange(arr); // 获取最大振幅频率
        var noteIndex = normalSearch(noteArr, maxEnergyFreq); // 获取对应音符下
        标
        if (noteIndex != 0) {
            console.log(keyList[noteIndex]); // 打印对应音符下标
            finalNoteArray.push(keyList[noteIndex]);
            console.log(finalNoteArray);
            Display(finalNoteArray);
        }
    }
}

```

```

//隔一段时间采样*

//写文件
function download(text, name, type) {
    var a = document.getElementById("a");
    var file = new Blob([text], { type: type });
    a.href = URL.createObjectURL(file);
    a.download = name;
    a.dispatchEvent(new MouseEvent('click', { 'bubbles': false, 'cancelable':
true }));
}

function downLoadDataToLoc() {
    var blob = new Blob(finalNoteArray, { type: 'text/plain' })
    // 创建一个 blob 的对象, 把 Json 转化为字符串作为我们的值
    if ("msSaveOrOpenBlob" in navigator) {
        // 这个判断要不要都行, 如果是 IE 浏览器, 使用的是这个,
        window.navigator.msSaveOrOpenBlob(blob, "results.txt");
    } else {    // 不是 IE 浏览器使用的下面的
        var url = window.URL.createObjectURL(blob)
        // 上面这个是创建一个 blob 的对象连链接,
        var link = document.createElement('a')
        // 创建一个链接元素, 是属于 a 标签的链接元素, 所以括号里才
是 a,

        link.href = url;
        // 把上面获得的 blob 的对象链接赋值给新创建的这个 a 链接
        link.setAttribute('download', "results.txt")
        // 设置下载的属性 (所以使用的是 download), 这个是 a 标签的
一个属性

        // 后面的是文件名字, 可以更改
        link.click();
        // 使用 js 点击这个链接
    }
}

function Display(arr) {
    console.log("display");
    const rel = document.getElementsByClassName("content")[0];
    let Rstr = "";
    let i=0
    for (i = 0; i < arr.length; i++) {
        var str = arr[i]
        console.log(str)
        if (i % 4 == 0) {
            Rstr += '|';
        }
        if (i % 8 == 0&& i!=0) {

```

```

        Rstr += '\n'+'|';
    }
    Rstr += str + ' ';

}
// while(i%4!=0){
//     Rstr += '0' + ' ';
//     i++
// }
// Rstr += '|';
rel.innerText = Rstr;
}

```

3.跳转界面功能 Return.js

```

function Return() {
    window.location.href = "./index.html";
}
function JieXi(){
    window.location.href = "./audioFile.html";
}
function LuYin(){
    window.location.href = "./mic.html";
}
function HistRecord(){
    window.location.href = "./histRecord.html";
}
function collection(){
    window.location.href = "./collection.html";
}

```