Groove

Release v1.2.4

zhiyiYo

Aug 29, 2023

用户指南

本文档是 Groove 项目的说明及帮助文档，包含用户指南和开发者指南两部分。

CHAPTER

ONE

快速上手

Groove 使用 Python3 进行开发，基于 PyQt5 构建 GUI，在使用之前需要根据操作系统下载相应的解码器。

1.1 安装

1.1.1 Win32

安装包

1. 下载并安装 LAV Filters
2. 从 Release 页面下载 `Groove_v*.*.*_x64_setup.exe`
3. 右键并以管理员身份运行 `Groove_v*.*.*_x64_setup.exe`，根据安装向导完成 Groove 的安装
4. 开启你的音乐之旅 ~~

免安装版

1. 下载并安装 LAV Filters
2. 从 Release 页面下载 `Groove_v*.*.*_windows_x64.zip`
3. 解压 `Groove_v*.*.*_windows_x64.zip`
4. 在解压出来的 `Groove` 文件夹中，找到并双击运行 **Groove.exe**
5. 开启你的音乐之旅 ~~

1.1.2 Linux

1. 安装 GStreamer
2. 从 Release 页面下载 `Groove_v*.*.*_linux_x64.zip`
3. 解压 `Groove_v*.*.*_linux_x64.zip`
4. 在解压出来的 `Groove` 文件夹中，找到并双击运行 **Groove** 可执行文件
5. 开启你的音乐之旅 ~~

1.2 基本使用

- 播放本地音乐
- 搜索、播放和下载在线音乐
- 创建和管理个人播放列表
- 查看和编辑歌曲元数据
- 观看和下载在线 MV

CHAPTER

TWO

音频格式

目前 Groove 音乐支持以下格式的音频，并根据文件的后缀名过滤掉不受支持的文件：

CHAPTER

THREE

播放逻辑

3.1 播放模式

`QMediaPlaylist` 支持五种播放模式:

Groove 音乐支持除了 `CurrentItemOnce` 外的所有播放模式。

3.2 按钮组合

Groove 音乐的播放栏上有两个按钮用来控制播放逻辑，分别是**随机播放按钮**和**循环模式按钮**。

随机播放按钮有两种状态: 选中和 未选中，循环模式按钮有三种状态: 顺序播放、列表循环和 单曲循环。两种按钮的状态组合与播放器播放模式的对应关系如下表所示：

当播放模式为 `CurrentItemInLoop` 时，无论随机播放按钮是否被选中，点击下一首按钮时都会按顺序选中并播放正在播放列表中的下一首歌曲。

CHAPTER

FOUR

快捷键

4.1 全局

全局快捷键在 Groove 音乐不处于活跃状态时（比如最小化到托盘）仍可用：

4.2 局部

局部快捷键只在 Groove 音乐处于活跃状态时（位于所有桌面应用的顶部）可用：

CHAPTER

FIVE

歌词文件

Groove 音乐支持 `lrc` 格式和 `json` 格式的歌词文件。

5.1 lrc 格式

歌词格式为 **[mm:ss.xx]**，其中 **mm** 为分钟，**ss** 为秒，**xx** 为百分之一秒，更多关于 `lrc` 格式的信息可以参见维基百科。下面是一个例子：

```
[ti:Lyric Demo]
[ar:zhiyiYo]
[au:Written by zhiyiYo, 2022]
[al:Groove - Vol. 2 - Melody]

[00:12.00]zhiyiYo - Lyric Demo
[00:15.30]hello
[00:15.30] 你好 # 重复时间标签来添加翻译
[01:02.30]world
[01:04.29] 我家硝子真卡哇伊 ⍰
```

5.2 json 格式

歌词格式为 **“seconds” :[” orginal lyric”]** 或者 **“seconds” :[” orginal lyric” , “translation lyric”]**。如下所示：

```
{
    "1.86": [
        " 微熱 - Aiko "
    ],
    "3.7": [
        " 词：aiko"
    ],
```

(continues on next page)

(continued from previous page)

```
    "6.49": [
        " 曲: aiko"
    ],
    "28.22": [
        " 今夜も必ず連絡するね",
        " 今夜也一定会和我联系"
    ],
    "34.36": [
        " 昼も夜も抱きしめて",
        " 又能相拥一夜"
    ],
}
```

CHAPTER

SIX

常见问题

6.1 为什么窗口拖动的时候会出现卡顿现象？

由于界面使用了亚克力窗口特效，在某些版本的 Win10 上会出现这个问题。有三种解决方案:

- 更新 Win10 到最新版本，比如 Win11.
- 取消复选框的选中 **高级系统设置–> 性能–> 拖动时显示窗口内容**.
- 在设置界面禁用亚克力效果.

6.2 为什么运行的时候 GStreamer 报错：Warning: “No decoder available for type …”？

可以尝试 `sudo apt-get install gstreamer1.0-libav` 来解决该问题，Ubuntu 20.04 亲测有效。

6.3 支持哪些格式的歌词文件呀？

目前支持 `.lrc` 和 `.json` 格式的歌词文件，更多信息请参见 歌词文件格式说明。

CHAPTER

SEVEN

快速上手

7.1 搭建开发环境

1. 创建虚拟环境:

```
conda create -n Groove python=3.8
conda activate Groove
pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple
```

2. 下载解码器:
   - 对于 Win32，安装 LAV Filters
   - 对于 Linux，安装 GStreamer
3. 打开 Groove 音乐:

```
cd app
conda activate Groove
python Groove.py
```

7.2 VSCode 配置文件

这里提供几个使用 VSCode 开发时会用到的配置文件。

7.2.1 launch.json

launch.json 用来调试 Groove 音乐，需要在 VSCode 中将 Python 解释器切换为 Groove 虚拟环境下的解释器才能保证环境不出问题。

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": " 调试当前文件",
            "type": "python",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal",
            "justMyCode": true,
            "cwd": "${fileDirname}"
        },
        {
            "name": " 调试 Groove",
            "type": "python",
            "request": "launch",
            "program": "${workspaceFolder}/app/Groove.py",
            "console": "integratedTerminal",
            "justMyCode": true,
        }
    ]
}
```

7.2.2 tasks.json

tasks.json 用来配置任务，一个任务对应着一条或者多条命令，这里总共配置了三个任务：Run Groove、Compile qrc 和 Compile qrc and run Groove：

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "Run Groove",
            "detail": " 运行 Groove 音乐",
            "type": "shell",
            "command": "D:/Anaconda/envs/Groove/python.exe",
            "args": ["Groove.py"],
            "problemMatcher": "$gcc",
```

(continues on next page)

(continued from previous page)

```
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "options": {
                "cwd": "${workspaceFolder}/app"
            }
        },
        {
            "label": "Compile qrc",
            "detail": " 编译 qrc 文件",
            "type": "shell",
            "command": "pyrcc5",
            "args": [
                "-o",
                "../common/resource.py",
                "resource.qrc",
            ],
            "options": {
                "cwd": "${workspaceFolder}/app/resource"
            },
            "problemMatcher": "$gcc",
            "group": {
                "kind": "build",
                "isDefault": true
            }
        },
        {
            "label": "Compile qrc and run Groove",
            "detail": " 编译 qrc 并运行 Groove 音乐",
            "type": "shell",
            "command": "D:/Anaconda/envs/Groove/python.exe",
            "args": ["Groove.py"],
            "problemMatcher": "$gcc",
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "dependsOn": [
                "Compile qrc"
            ],
            "options": {
                "cwd": "${workspaceFolder}/app"
```

(continues on next page)

(continued from previous page)

```
                }
            },
        ]
}
```

7.3 注意事项

1. 资源文件发生变更之后需要使用 `pyrcc5` 重新编译 `resource.qrc` 文件，生成的 `resource.py` 文件放在 `common` 文件夹下面

CHAPTER

EIGHT

软件架构

8.1 主要模块

8.2 界面结构

8.2.1 主界面

8.2.2 选择模式界面

Groove 音乐中大多数界面的结构如下图所示，由 `view` 和 `SelectionModeBar` 组成：

由于 `SelectionModeBar` 种类多样，代码中使用工厂模式来创建 `SelectionModeBar`，这样可以增强代码的可拓展性：

选择模式界面的类图如下所示，`SelectionModeInterface` 的子类使用 `setView(view)` 方法更换视图为专辑卡视图、歌曲列表部件、歌手卡视图或者播放列表卡视图，这些视图都实现了 `SelectionModeViewBase` 的两个抽象方法：

以专辑卡视图为例，`AlbumCardViewBase` 通过 `AlbumCardFactory` 创建各种类型的专辑卡，由于专辑卡视图有网格布局和水平布局两种，所以相应地有 `GridAlbumCardView` 和 `HorizonAlbumCardView` 子类：

CHAPTER

NINE

开发规范

9.1 命名规范

- **包名**和**文件名**使用蛇形命名法
- **类**使用大驼峰命名法
- **函数**和**变量**使用小驼峰命名法，与 Qt 保持一致

9.2 项目结构

9.2.1 app

所有与图形界面相关的代码都放在此文件夹下，具体结构如下：

- **common** 文件夹：包含所有文件共享的函数和类
- **components** 文件夹：包含所有窗口共享的组件，比如按钮、菜单和对话框
- **View** 文件夹：包含各个界面，比如我的音乐界面、正在播放界面和主界面
- **resource** 文件夹：包含图标和样式表等资源文件
- **config** 文件夹：包含配置文件 `config.json`
- **cache** 文件夹：包含缓存的图片、数据库和日志

9.2.2 tests

用于存放测试用例，修改代码后应该再次运行测试用例。

9.2.3 docs

用于存放项目文档，使用说明可以参见 《Sphinx + Read the Docs 从懵逼到入门》

CHAPTER

TEN

数据库

Groove 音乐使用 sqlite 数据库进行歌曲信息、专辑信息和播放列表信息等数据的管理。

10.1 各个模块

10.1.1 entity

实体类模块，每个实体类实例用于保存一条数据表记录。

10.1.2 dao

数据库访问操作模块，`DaoBase` 作为基类封装了基本的数据库操作方法，使得子类无需编写重复的 SQL 语句就能操作数据库。

10.1.3 service

业务模块，业务类使用 `Dao` 类来操作数据库

10.1.4 controller

控制器模块，控制器类使用 `Service` 类来操作数据库，外部使用 `controller` 提供的接口来访问数据库。

10.2 数据表

10.2.1 tbl_song_info

歌曲信息表，对应 `SongInfo` 实体类：

其中，`file` 字段有两种格式：

- 本地音乐路径，比如：`D:/Music/aiko - 二人.mp3`
- 在线音乐 URL
  - 虚假 URL，比如：http://kuwo/song/2333，播放时会被转换为真实 URL
  - 真实 URL

10.2.2 tbl_album_info

专辑信息表，对应 `AlbumInfo` 实体类：

10.2.3 tbl_singer_info

歌手信息表，对应 `SingerInfo` 实体类：

10.2.4 tbl_playlist

自定义播放列表，对应 `Playlist` 实体类：

仔细想想，`singer`、`album` 和 `count` 字段不应该保存到数据表中，而是在查询的时候填入实体类实例，罢了罢了，软件开发第一原则，程序能跑就行 ~~

10.2.5 tbl_song_playlist

自定义播放列表和歌曲信息中间表，对应 `SongPlaylist` 实体类：

10.2.6 tbl_recent_play

最近播放表，对应 `RecentPlay` 实体类：

CHAPTER

ELEVEN

事件总线

在 Qt 中可以使用信号和槽机制很方便地实现部件之间的通信，考虑下面这样的场景：

如果想要点击任意一个专辑卡并通知主界面跳转到专辑界面，一种实现方式如上图所示：点击任意一个蓝色方框所示的专辑卡，发出 `switchToAlbumInterfaceSig` 信号给父级部件专辑卡视图，因为专辑卡视图有许多个分组，比如上图中为 `aiko` 分组，可能还有 *柳井爱子* 分组，那么这些视图都应该将 `switchToAlbumInterfaceSig` 转发给父级窗口我的音乐界面，我的音乐界面再转发给主界面，从而实现界面跳转。

可以看到上面这种做法很麻烦，专辑卡上拥有 `switchToAlbumInterfaceSig` 属性就算了，还要连累父级专辑卡视图以及祖父级我的音乐界面也拥有这个属性才能实现信号的转发。有没有一种方式可以省掉中间的转发过程，从而一步到位通知主界面呢？这就需要使用下面所介绍的全局事件总线思想（这里不区分信号总线和事件总线两种叫法）。

11.1 Vue 中的全局事件总线

在 vue 中要实现任意组件间通信，可以在 `Vue.prototype` 上添加一个全局事件总线 `$bus` 属性，当组件 A 想要给组件 B 发送一些数据时，只需要在 A 中 `this.$bus.$emit(事件名，数据)` 发送数据，在 B 中 `this.$bus.$on(事件名，回调)` 就能通过总线收到数据，而无需借助其他组件的转发。将事件名视为信号，回调视为槽函数，那么这个过程和 Qt 的信号和槽机制神似。

11.2 Qt 中的全局事件总线

仿照上述过程，我们来定义一个全局事件总线类，并使用单例模式保证只能实例化出一个对象：

```
# coding:utf-8
from PyQt5.QtCore import QObject, pyqtSignal


class SignalBus(QObject):
```

(continues on next page)

(continued from previous page)

```
    """ 全局事件总线 """

    switchToAlbumInterfaceSig = pyqtSignal(str)

    def __new__(cls, *args, **kwargs):
        if not hasattr(cls, '_instance'):
            cls._instance = super(SignalBus, cls).__new__(cls, *args, **kwargs)

        return cls._instance

bus = SignalBus()
```

回到最初的那个例子，现在我们只需导入 `bus` 对象，点击 `aiko　の詩。专辑卡时` 发送 `bus.switchToAlbumInterfaceSig.emit('aiko　-　aiko　の詩。')` 来发送切换到专辑界面的信号，然后在主界面中 `bus.switchToAlbumInterfaceSig.connect(self.switchToAlbumInterface)` 即可，这样就省去了信号的转发流程，代码会简洁许多。

CHAPTER

TWELVE

踩过的坑

- 不能直接给图片添加 .jpg 后缀名，会导致 QPixmap 无法识别
- 网格布局的行和列只能增加不能减少，但是可以改变没有用到的行或者列的宽度
- 要想改变旧布局，只需在一个总的布局中添加后来想要移除的布局就行，比如 all_h_layout.addLayout(gridLayout)，后面要改变的时候只需 removeItem(gridLayout)
- deleteLater() 释放内存
- 滚动条最好手动设置最小高度，不然可能太小而看不见
- m4a 不存在某个键时需要先手动创建一个空列表，再将值添加到列表中
- 如果 widget 是自定义类要设置背景颜色首先要添加一句：

```
self.setAttribute(Qt::WA_StyledBackground, true)
self.setStyleSheet("background-color: rgb(255, 255, 255)")
```

- 给主窗口设置磨砂效果，然后留下一部分完全透明的给子部件，这样看起来就好像子部件也打开了磨砂效果
- 如果需要指定无边框窗体，但是又需要保留操作系统的边框特性，可以自由拉伸边框，可以使用 setWindowFlags(Qt::CustomizeWindowHint);
- 使用 raise_() 函数可以使子窗口置顶
- event.pos() 返回的是事件相对小部件自己的位置
- 可以通过设置最外层的布局 self.all_h_layout.setSizeConstraint(QLayout.SetFixedSize) 来自动调整大小
- 要在小部件上使用磨砂效果只需将其设置成独立窗体，比如 Qt.window、Qt.popup
- 可以通过设置已有的属性来直接改变小部件的状态，比如 label.setProperty('text', str) 可以将实例的 text 设置为想要的 str, 而且还可以在一个小部件上设置多个自定义的属性
- setContentsMargins(int, int, int, int) 的顺序为 left、top、right、bottom
- 使用 self.window() 可以直接获取顶层对象

- 当把小部件添加到 groupBox 中时，groupBox 会变成父级
- 要想动态更新 `QListWidget` 的 Item 的尺寸只需重写 `resizeEvent()` 的时候 `item.setSizeHint(QSize())`
- 可以在样式表中用 `background:transparent` 来替代 `setAttribute(Qt.WA_TranslucentBackground)`
- 画图用 `drawPixmap()` 别用 `drawRect()`,要写字的时候不能 `painter.setPen(Qt.NoPen)`
- 文件夹的最后一个字符绝对不能是 /
- 如果出现主界面卡顿，可以通过信号提前结束此时进行的槽函数，将信号连到另一个槽函数来处理
- `font-weight = 500` 时会变为好康的楷体
- `self.pos().x()` 和 `self.x()` 得到的结果相同，代表窗体标题栏左上角的全局坐标，`self.geometry().x()` 得到的是客户区的全局坐标
- 弹出窗口的 qss 不起作用时可以手动 `setStyle(QAppliction.style())`
- 使用 `adjustSize()` 自动调整窗口尺寸
- `QListWidget` 使用 `setViewportMargins()` 设置内边距
- 当滚动条背景出现花纹时记得将

```
QScrollBar::add-page:vertical,
QScrollBar::sub-page:vertical {
    background: none;
}
```

- 处于省电模式下运行会卡顿
- lambda 表达式在执行的时候才会去寻找变量，开循环将按钮的 `clicked` 信号连接到 lambda 函数需要写成:

```
bt.clicked.connect(lambda checked, x=x: slotFunc(x))
```

具体操作参见 https://www.cnblogs.com/liuq/p/6073855.html

CHAPTER

THIRTEEN

关于

Groove 音乐开发自 2020 年 5 月 1 日，刚开始只是个本地音乐播放器，随着自身知识储备的增加，逐渐加入了联网功能，支持在线音乐、歌词和 MV。在开发这个项目的过程中自己也学到了很多知识，包括但不仅限于：

- PyQt 界面开发
- 数据库
- 图像处理
- 设计模式
- 爬虫

由于作者能力有限，软件难免有些缺陷，大家在使用过程中有遇到任何问题欢迎提 issue，如果喜欢本项目的话也可以点个 ⭐ 以示支持。

> **Warning:** Groove 音乐仅供学习使用，任何人不得将其用于商业及其他非法用途，否则后果自负。

13.1 感谢以下项目

- zhiyiYo/PyQt-Frameless-Window：一个基于 PyQt5 的跨平台无边框窗口，支持 Win32、Linux 和 MacOS
- zhiyiYo/PyQt-Fluent-Widgets：一个 Windows Fluent 风格的组件库
- jsmolka/egg-player：一个 Groove 音乐风格的播放器，这个项目的代码写的非常优雅，作者甚至手撕线程池，对本项目的影响非常大，墙裂推荐大家阅读其代码，一定会有不小的收获