**设计文档**

**1 引言**

此文档为多平台音乐播放器的设计文档。

项目地址：https://github.com/Bowenduan/BowenMusic

参考资料：如表1所示

符号说明：如表2所示

表1 项目参考资料

|  |  |
| --- | --- |
| 资料名称 | 地址 |
| Qt for Python Documentation | https://doc.qt.io/qtforpython/api.html |
| Python **cmd** Documentation | https://docs.python.org/3/library/cmd.html |
| Python **mpv** | https://github.com/jaseg/python-mpv |
| mpv properties | https://mpv.io/manual/master/#properties |

表2 符号说明

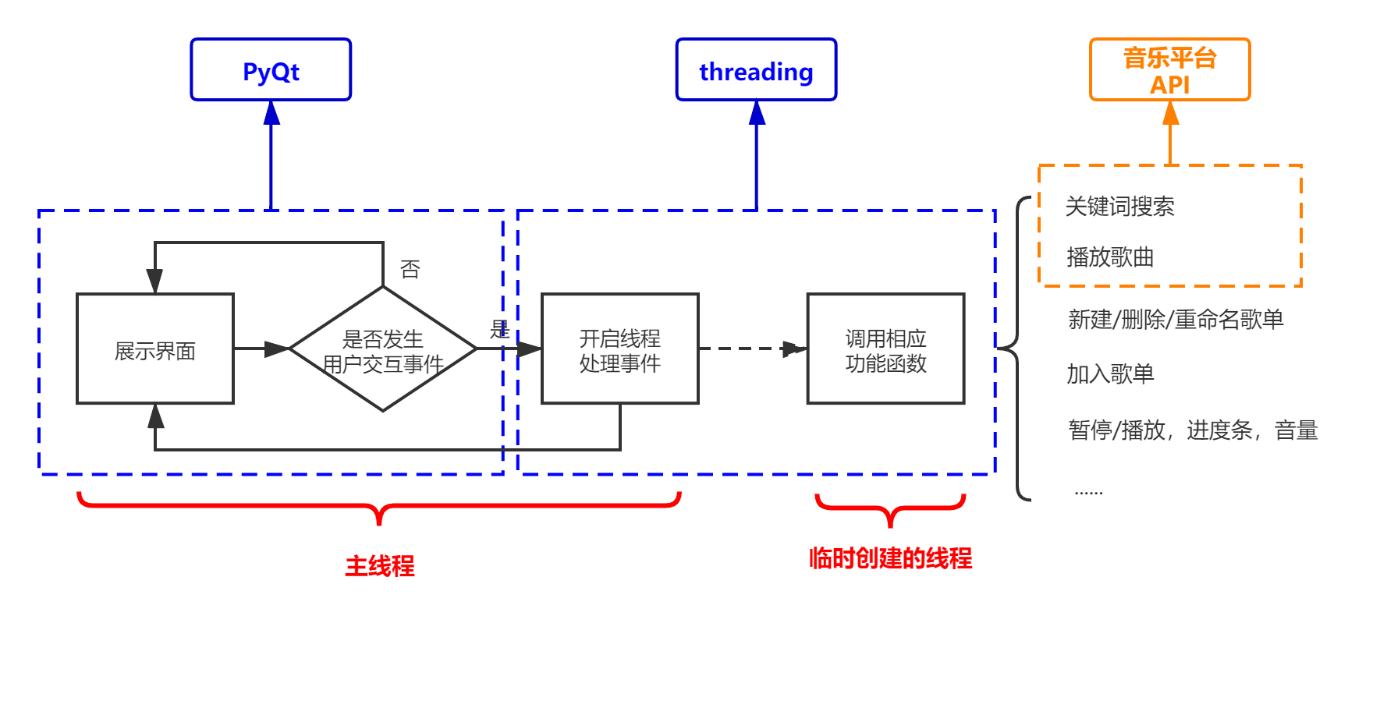
|  |  |
| --- | --- |
| 符号 | 说明 |
| # | 出现在伪代码中#后均为说明或注释内容 |
| ` ` | Python类型，如 `int` `str` 等 |

**2 概要设计**

**2.1 界面版概要设计**

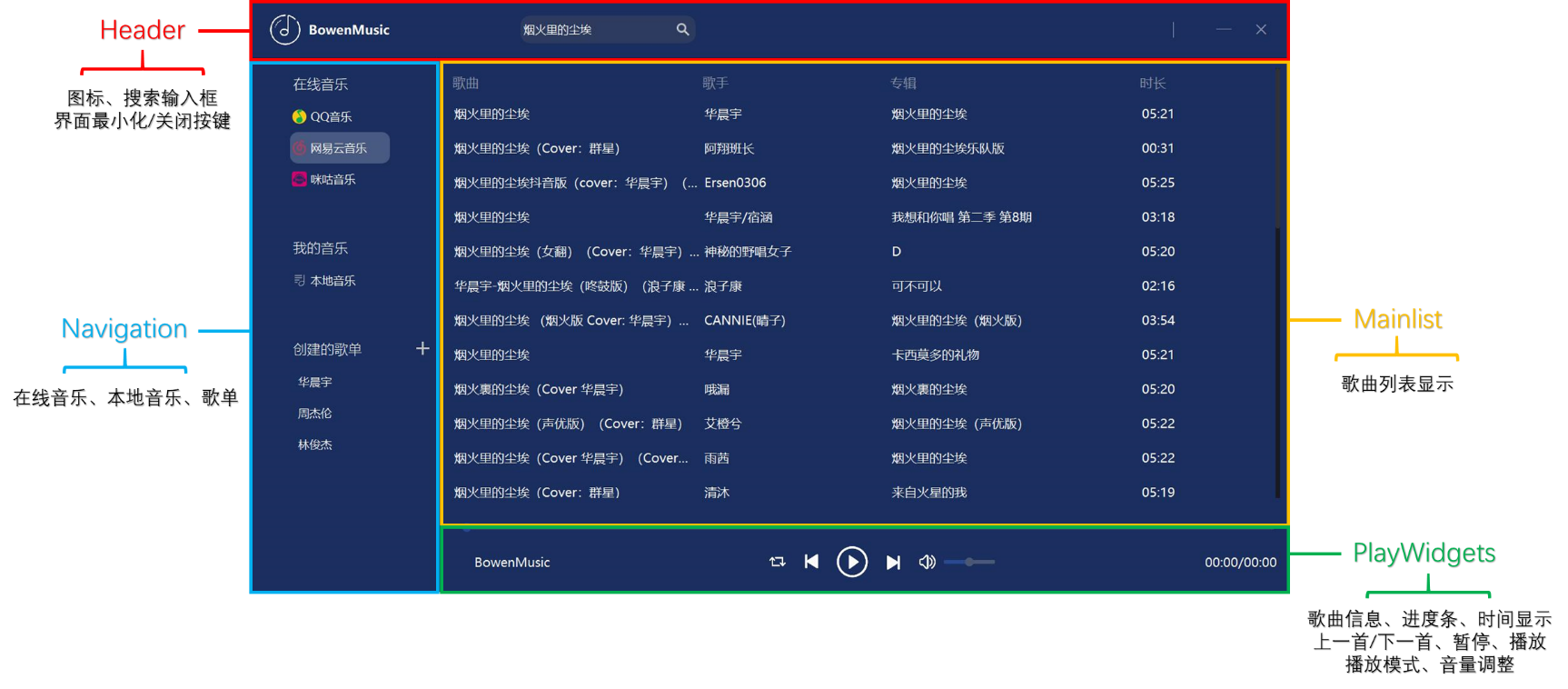
多平台音乐播放器的界面能够不间断地接受用户的交互操作，故该软件整体框架是一个主线程负责界面的展示与用户事件的接收，接收用户事件后再创建一个新的进程处理相应事件，从而后台功能模块的运行就不会影响界面的刷新。界面设计基于PyQt库，线程处理使用Python自带threading库，功能模块中关键词搜索和播放歌曲操作需要调用音乐平台API（在需求分析文档的可行性分析中有详细介绍）。界面版播放器的框架如图1所示。

图1 界面版播放器框架



播放器界面设计如图2所示，主要包含四个模块：Header, Navigation, Mainlist, PlayWidgets. Header位于播放器最上方，包含播放器图标，搜索输入框，界面最小化与关闭界面；Navigation位于播放器左方，包含在线音乐、本地音乐和歌单，作为导航；Mainlist为歌曲列表，根据Navigation中的选择来显示相应列表内容；PlayWidgets包含音乐播放相关功能，如暂停/播放，音量调整，进度条显示等。

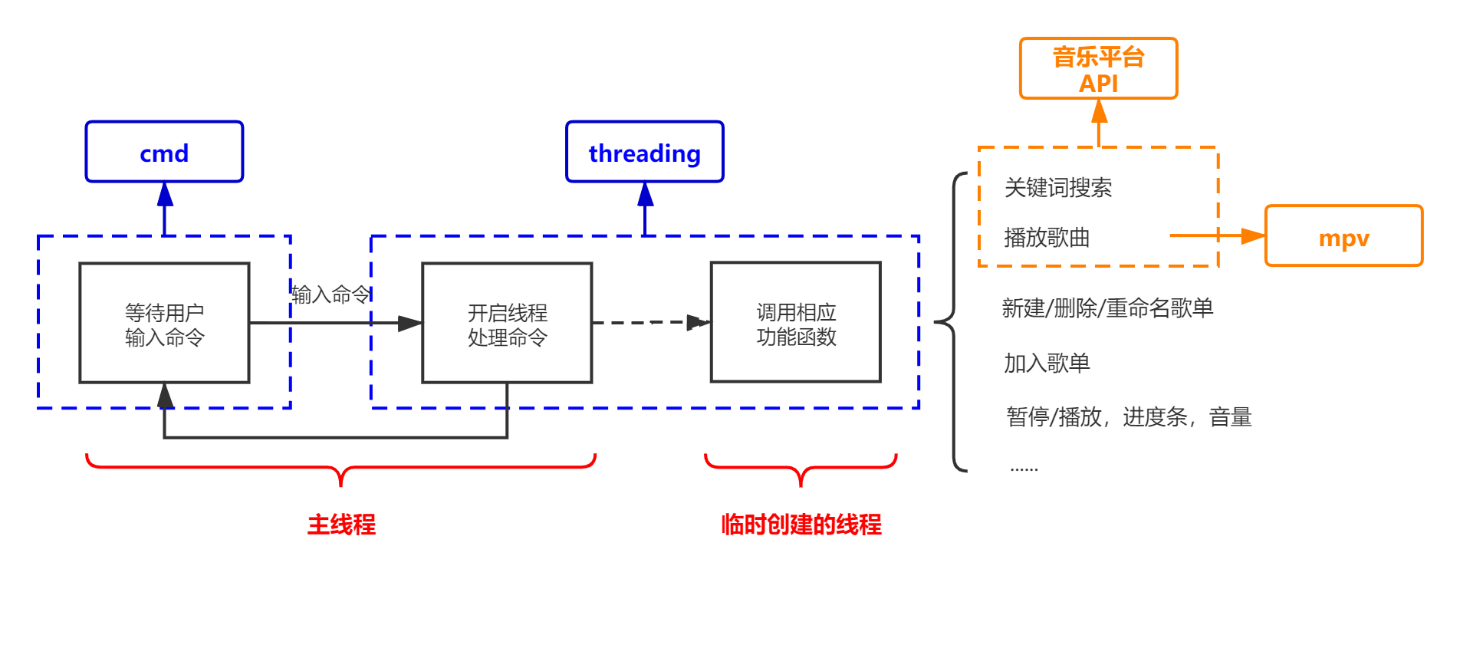
图2 播放器界面



**2.2 终端版概要设计**

终端版多平台音乐播放器设计整体框架与界面版相同，故同样设计一个主线程接受用户命令的输入，并根据该输入创建进程进行相应的处理。交互式终端基于Python自带cmd库，线程处理使用Python自带threading库，功能模块中关键词搜索和歌曲url获取调用音乐平台API，播放歌曲使用mpv对python接口。终端版播放器的框架如图3所示。

图3 终端版播放器框架



**3 重要数据结构和变量说明**

（1）描述一首歌的信息使用Python的字典结构，以下称为歌曲字典。由于不同平台获取到的歌曲信息有差异，如果没有某个键，值置为None.

`dict` - { ‘song\_name’ : # `str` 歌曲名称

‘song\_mid’ : # `str` 歌曲id

‘media\_mid’ : # `str` 播放媒体id, qq音乐独有

‘interval’ : # `int` 歌曲总时长（秒）

‘album\_name’ : # `str` 所属专辑名称

‘album\_mid’ : # `str` 所属专辑id

‘singer\_list’ : # `list` 歌手列表（可能由多个歌手）

‘url’ : # `str` 歌曲url }

**4 重要模块详细设计**

**4.1 音乐平台API**

根据需求分析文档中对QQ音乐、网易云音乐、咪咕音乐API的分析，使用Python的requests库构造请求实现音乐平台API。针对三个平台分别设计了三个类：QQMusicApi, NeteaseCloudMusicAPI, MiguMusicAPI，三个类提供的接口如下。

（1）

class QQMusicApi( ):

# 关键词搜索接口

def search(self, page, keyword):

# 参数: page `int` – 展示第几页结果

# keyword `str` – 关键词

# 返回: 歌曲列表 `list`，列表中每个元素是歌曲字典

# 歌曲url接口

def get\_url(self, song\_mid):

# 参数: song\_mid – 歌曲id，可由search函数获得

# 返回：歌曲url `str`.

（2）

class NeteaseCloudMusicAPI( ):

# 关键词搜索接口

def search(self, keyword, type):

# 参数：type `int` –

# 1 单曲，10 专辑，100 歌手，1000 歌单，1002 用户，1004 MV

# 1006 歌词，1009 电台

# keyword `str` - 关键词

# 返回：歌曲列表 `list`，列表中每个元素是歌曲字典

# 歌曲url接口

def get\_url(self, song\_id, br=128000):

# 参数：song\_id `str` - 歌曲id

# br `int` - 码率

# 返回：歌曲url `str`

（3）

class MiguMusicAPI( ):

# 关键词搜索接口

def search(self, keyword):

# 参数：keyword `str` - 关键词

# 返回：歌曲列表`list`，列表中每个元素是歌曲字典（包含url键）

# 因为咪咕搜索接口能直接获取歌曲url

**4.2 界面版播放器GUI设计**

界面包含的四个模块Header, Navigation, Mainlist, PlayWidgets分别以Python类实现。

（1）Header类

继承PyQt的QFrame类，包含的主要控件有关闭窗口按键(QPushButton)，窗口最小化按键(QPushButton)，软件图标(QLabel)，搜索输入框(QLineEdit)。采用QHBoxLayout()水平布局。

（2）Navigation类

继承PyQt的QScrollArea类，包含的主要控件有在线音乐列表(QListWidget)，包含qq音乐、网易云音乐、咪咕音乐，本地音乐列表(QListWidget)，歌单列表(QListWidget)。针对歌单列表设计了新增歌单的按钮(QPushButton)。采用QVBoxLayout()垂直布局。用户点击相应区域会在Mainlist显示相应歌曲列表。在歌单列表中还增加了右键菜单(QMenu)，可重命名/删除歌单。

（3）Mainlist类

继承PyQt的QScrollArea类，只包含一个控件：歌曲列表(QTreeWidget)。用户通过双击列表中的某首歌进行播放；该控件中还加入了右键菜单功能(QMenu)，可将歌曲加入歌单，也可进行播放。

（4）PlayWidgets类

继承PyQt的QFrame类，包含的主要控件有音乐播放器(QMediaPlayer)，播放进度条(QSlider)，音量调整(QSlider)，歌曲信息显示(QLabel)，歌曲播放时长显示(QLabel)，还有播放模式切换、上一首/下一首，播放/暂停，静音按钮，均由QPushButton实现。进度条与其他控件呈QVBoxLayout()垂直布局，其他控件之间呈QHBoxLayout()水平布局。按键对应事件可通过调用QMediaPlayer的函数实现。

此外，界面的样式设计通过在python代码中导入.qss脚本实现。

**4.3 终端版播放器交互式终端设计**

编写了MusicShell类，该类继承cmd库中的Cmd类。通过在类中定义do\_command( )函数，实现接收到用户输入command时执行的功能。MusicShell类详细设计如下。

class MusicShell(Cmd):

def do\_pl(self, arg):

# 展示已有歌单

# 用法 > pl # 展示所有歌单名称

# > pl [index] # 展示第index个歌单中的歌曲列表

def do\_cpl(self, arg):

# 创建新的歌单

# 用法 > cpl # 会提示输入歌单名

def do\_rpl(self, arg):

# 重命名歌单

# 用法 > rpl [index] # 修改第index个歌单的名称，会提示输入新名称

def do\_dpl(self, arg):

# 删除歌单

# 用法 > dpl [index] # 删除第index个歌单

def do\_search(self, keyword):

# 根据关键词搜索歌曲

# 用法 > search [keyword] # 用当前音乐平台API搜索，并展示结果

def do\_add(self, arg):

# 将搜索结果中某首歌加入歌单

# 执行条件: 此前已经执行过search操作

# 用法 > add [index1] [index2]

# index1是歌曲在上一次搜索结果中的索引

# index2 是想要加入的歌单的索引

def do\_play(self, arg):

# 播放歌曲

# 用法 > play # 播放第一个歌单

# > play -pl [index] # 播放第index个歌单

# > play -s [index] # 播放上次搜索结果中第index首歌

def do\_m(self, arg):

# 暂停/继续(mode)音乐

# 用法 > m # 若正在播放，会暂停；若已经暂停，会继续播放

def do\_i(self, arg):

# 显示当前正在播放歌曲的信息(info)

# 用法 > i # 显示歌名、歌手、播放时长

def do\_s(self, arg):

# 显示当前使用的API名称(source)

# 用法 > s # qq/netease/migu

def do\_cs(self, arg):

# 更改音乐源(API)

# 用法 > cs qq # 切换为QQMusicAPI

# > cs netease # 切换为NeteaseCloudMusicAPI

# > cs migu # 切换为MiguMusicAPI

def do\_quit(self, arg):

# 关闭终端播放器

# 用法 > quit/bye/exit

def do\_bye(self, arg): # 与do\_quit相同

def do\_exit(self, arg): # 与do\_quit相同

def default(self, arg):

# 提示命令不存在

**4.4 主要功能设计**

（1）关键词搜索

调用音乐API（详见4.1）的search函数即可，关键词作为参数传入，再将返回值以合适的方式展示（界面展示再Mainlist模块中，终端直接格式化print）。

伪代码：

print(api.search(keyword))

（2）创建歌单

若用户输入的新建歌单的名称已经存在，则返回；否则在本地的歌单目录新建json文件，文件名即歌单名。伪代码如下：

pl\_name = input()

if pl\_name + ‘.json’ in 本地歌单文件:

return

else:

new\_path = os.path.join(歌单目录, pl\_name+’.json’)

with open(new\_path, ‘w’) as f:

json.dump([ ], f, indent=1)

（3）加入歌单

将选中歌曲的歌曲字典存入对应歌单文件即可。伪代码如下：

song\_dict = 用户选中歌曲的歌曲字典

pl\_name = 用户选取的歌单名称

if pl\_name + ‘.json’ not in 本地歌单文件：

return

else:

path = os.path.join(歌单目录, pl\_name+’.json’)

with open(path, ‘r’) as f:

jsonlist = json.load(f)

jsonlist.append(song\_dict)

with open(path, ‘w’) as f:

f.dump(jsonlist, f, indent=1)

（4）重命名歌单

若用户输入新名称与旧名称相同，返回；否则修改歌单文件名即可。伪代码如下：

new\_name = 用户输入的新歌单名

pl\_name = 旧歌单名

if new\_name == pl\_name:

return

else:

new\_path = os.path.join(歌单目录, new\_name+’.json’)

old\_path = os.path.join(歌单目录, pl\_name+’.json’)

os.rename(old\_path, new\_path)

（5）删除歌单

若删除歌单存在，则删除；否则返回。伪代码如下：

pl\_name = 待删除歌单名称

if pl\_name+’.json’ not in 本地歌单文件:

return

else:

path = os.path.join(歌单目录, pl\_name+’.json’)

os.remove(path)

（6）播放歌曲

首先需要获取歌曲的url，再使用mpv库构造MPV播放器变量，开启播放线程。并设置一个threading.Event()变量用于两个线程之间的交互。伪代码如下：

# 主线程

song\_dict = 待播放歌曲的歌曲字典

threading.Thread(target=self.play\_song, args=(song\_dict,)).start() # 开启线程1

# 线程1

def play\_song(self, song\_dict):

根据song\_dict获取song\_id

调用音乐API中get\_url(song\_id)获取歌曲url # 获取url

self.player = mpv.MPV() # 构造MPV播放器

self.event = threading.Event() # event变量

threading.Thread(target=self.play\_base, args=(url,)).start() # 开启线程2

self.event.wait() # 等待线程2结束

self.player.terminate() # 析构MPV播放器

# 线程2

def play\_base(self, url):

self.player.play(url) # 根据url播放歌曲

self.player.wait\_for\_playback() # 持续播放歌曲直到播放完毕

self.event.set() # 使线程1继续执行

**5 源码文件说明**

- qss # 文件夹 - .qss样式脚本

- resouce # 文件夹 - GUI中使用的图标

- src # 文件夹 - 源码

--- api.py # 代码文件 - 三个音乐API的实现

--- component.py # 代码文件 - 自己设计的继承PyQt中类的组件

--- module.py # 代码文件 - GUI四个模块的实现

--- MusicCmd.py # 代码文件 - 终端版播放器运行代码

--- MusicDesktop.py # 代码文件 - 界面版播放器运行代码

--- intro.txt # txt文件 - 终端版播放器欢迎字符

- userdata # 文件夹 - 用户数据文件夹

--- music # 文件夹 – 存放下载的歌曲

--- playlist # 文件夹 – 歌单目录，存放歌单json文件