MusicPlayer项目说明文档

> 同济大学2019级软件学院数据结构课荣誉班项目

项目信息

- **项目名称：用Qt实现一个轻量级的音乐播放器**
- **指导老师：张颖老师**
- **小组成员：1951576　沈星宇**
  1953288　张思源
- **贡献比例：沈星宇：100%**
  张思源：100%
- **项目github地址：**

  https://github.com/Sherlock-White/2020-TJ-program-MusicPlayer
- **参考博客：**

  https://blog.csdn.net/Kingsman_T/article/details/103879947

项目开发文档

- 开发环境：Qt5 （Qt Creator 4.13.3 + Qt 5.9.9 MinGW 32bit）
- 支持平台：windows
- 完成时间：2020.12.6 - 2020.12.31

1 功能分析

基础功能：

- [x] 支持添加本地文件（支持格式：.mp3 、.flac 、.mpga ）并播放
- [x] 支持解析歌词以滚动形式展示
- [x] 支持解析专辑封面
- [x] 支持自定义主页面背景
- [x] 支持歌单管理（”我喜欢“+”其他自定义歌单“）
- [x] 支持多种歌曲循环模式
- [x] 支持“下一曲、下一曲”
- [x] 支持“暂停、播放”

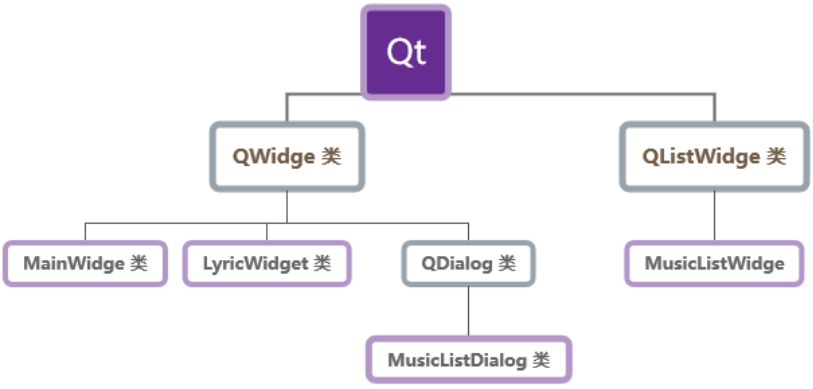
细节功能：

- [x] 鼠标悬浮在按钮上时有提示信息
- [x] 任务栏右击进行任务选择
- [x] 主页面通过”about“来展示项目基本信息

2 类结构设计

项目中用到的类如下：

- Music 类
- MusicList 类
- MusicListWidget 类
- MusicListDialog 类
- LyricWidget 类
- MainWidget 类

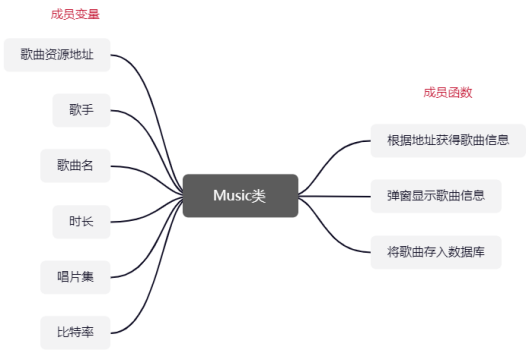


解释：灰色边框的类是Qt提供的可继承的类，紫色边框的为本项目从原有类中派生出来的类，详细的接口、成员与功能在2.2阐述。

2.2 各个类详细展示

2.2.1 Music 类（歌曲类）

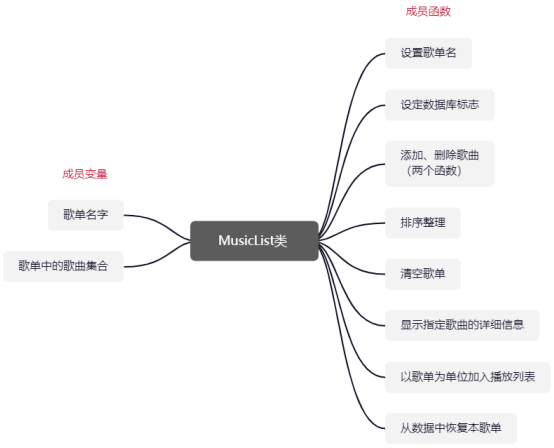
> 用来储存歌曲的信息，包含数据库的存取和歌曲解析两大功能
>
> 其中最为重要的是歌曲的 url 信息（即：歌曲文件所在的路径），播放的时候需要根据 url 来将歌曲文件加入播放器。还记录下的一些其他的歌曲信息，可见于下图。




```
class Music
{
private:
    QUrl url;                   //歌曲资源地址
    QString author;             //歌手
    QString title;              //歌曲名
    qint64 duration;            //时长
    QString albumTitle;         //唱片集
    int audioBitRate;           //比特率
    void refreshInfo();         //根据歌曲url获得歌曲信息
    friend class MusicCompare;  //用于歌曲排序的函数
public:
    Music(){}
    Music(QUrl iurl);
    QUrl getUrl() const {return url;
    QString getInfo() const;    //返回歌曲的信息
    void detail();              //弹窗显示歌曲信息
    void insertSQL(const QString& name);
                                //存入数据库
    QString getLyricFile();     //根据文件名来获取歌词路径
    friend class MusicList;
};
```

2.2.2 MusicList 类（歌单类）

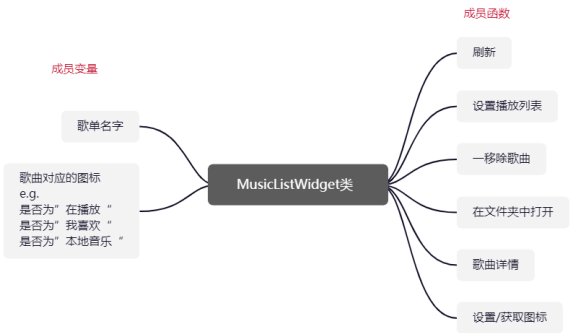
> 歌曲列表（即歌单），是用容器vector将music类的对象存储起来，由此可以进行一些具体的、针对多首歌曲的操作




```cpp
class MusicList
{

    QString name;            //歌单名

    vector<Music> music;    //所存储的歌曲


    bool sql_flag=true;     //控制是否需要与数据库交互
                            //e.g.删除播放列表的歌曲时不需要更新数据库
    friend class MainWidget;//设置友元关系，便于加入主页面中
public:
    MusicList(){}
    MusicList(const QList<QUrl>& urls,QString iname="");

    void setName(const QString& iname){name=iname;}
                                        //设定歌单名
    QString getName(){ return name; }   //设定数据库标志
    void setSQL(bool on){ sql_flag=on; }    //从url添加歌曲
    void addMusic(const QList<QUrl>& urls);
    void addMusic(const Music& iMusic);     //添加一首歌曲
    Music getMusic(int pos);                //获取指定位置的歌曲
    void addToPlayList(QMediaPlaylist *playlist);
                                        //将本歌单加入播放列表
    void addToListWidget(MusicListWidget *listWidget);
                                        //歌单可视化
    void removeMusic(int pos);          //移除指定的歌曲
    void showInExplorer(int pos);       //在文件夹中打开
    void detail(int pos);               //显示指定歌曲详细信息
    void remove_SQL_all();              //数据库中移除全部本歌单的歌曲
    void insert_SQL_all();              //将歌单中的歌曲全部写入数据库
    void read_fromSQL();                //从数据中恢复本歌单
    void sort_by(COMPARE key);          //将本列表中的歌曲排序
    void neaten();                      //整理歌单：将歌单中的重复歌曲去掉并排序
    void clear();                       //清空本歌单
};
```

2.2.3 MusicListWidget 类（歌单窗口）

> 以列表形式展示歌曲时，会用到 Qt 中的 QListWidget 组件，但直接使用 QListWidget 组件有一个弊端——因为没有结合 MusicList 这个数据结构，所以不太方便管理。
>
> 因此，MusicListWidget 继承了 QListWidget，它在类中包含 MusicList 类，这样可以将 MusicList 中的歌曲信息方便地可视化展示出来，并且可以方便地对展示出来的歌曲进行管理。




```cpp
class MusicListWidget:public QListWidget
{
    Q_OBJECT
    MusicList musicList;            //当前歌曲列表（存储的是歌曲信息）
    QIcon icon=QIcon(":/image/image/image/music.png");
                                    //当前展示列表项使用的图标
public:
    MusicListWidget(QWidget *parent = Q_NULLPTR);
    void refresh();                 //刷新显示（当musicList有所变化时，需要调用）
    void setMusicList(const MusicList& music);
                                    //设置歌曲列表
    void setMusicList_playing(const MusicList& music);
                                    //设置播放列表
    void removeMusic();             //移除歌曲
    void showInExplorer();          //在文件夹中打开
    void detail();                  //歌曲详情
    void setIcon(QIcon iicon){ icon=iicon; }
                                    //设置/获取图标Icon
    QIcon getIcon(){ return icon; }
    friend class MainWidget;        //建立友元关系，便于在主页面中插入
};
```

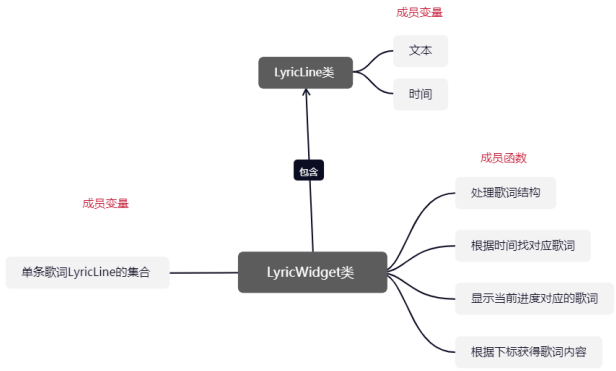
2.2.4 MusicListDialog 类（”添加至歌单“时弹出的对话框）

> 歌单的创立需要用户自行选择歌曲加入歌单，此时将弹出一个窗口，供用户从”本地音乐“中选取一个或多个歌曲进入对应的歌单，继承于 QDialog 类。
>
> 这里使用自定义对话框主要为了：
>
> ①便于独立使用一个.ui 文件进行界面设计；
>
> ②便于获取歌曲数据（即本地音乐有哪些歌曲），以及返回选择结果。


```cpp
class MusicListWidget:public QListWidget
{
    Q_OBJECT
    MusicList musicList;             //当前歌曲列表（存储的是歌曲信息）
    QIcon icon=QIcon(":/image/image/image/music.png");
                                     //当前展示列表项使用的图标
public:
    MusicListWidget(QWidget *parent = Q_NULLPTR);
    void refresh();                  //刷新显示（当musicList有所变化时，需要调用）
    void setMusicList(const MusicList& music);
                                     //设置歌曲列表
    void setMusicList_playing(const MusicList& music);
                                     //设置播放列表
    void removeMusic();              //移除歌曲
    void showInExplorer();           //在文件夹中打开
    void detail();                   //歌曲详情
    void setIcon(QIcon iicon){ icon=iicon; }
                                     //设置/获取图标Icon
    QIcon getIcon(){ return icon; }
    friend class MainWidget;         //建立友元关系，便于在主页面中插入
};
```

2.2.5 LyricWidget 类（歌词显示的窗口）

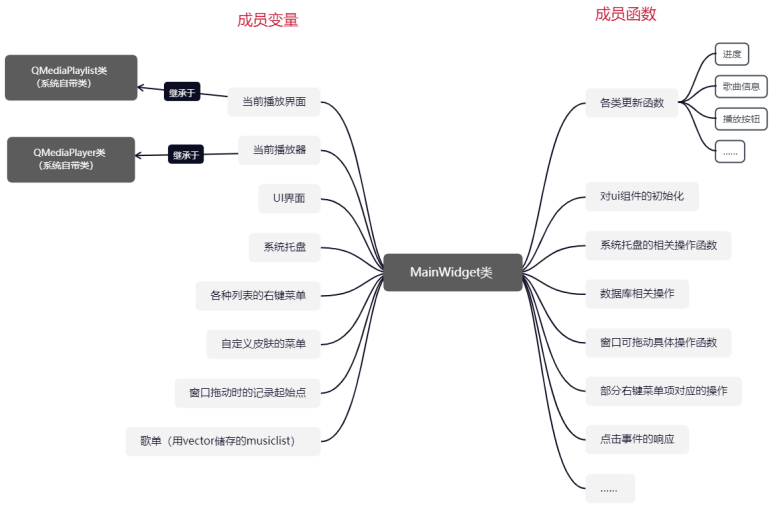
> 继承 QWidget，一个矩形区域，用来专门显示歌词的窗口，嵌入至主页面中，建立在”单条歌词（LyricLine类）“的基础上包含有用于显示歌词的几个文本框（QLabel）。这里使用自定义的 Widget，也是便于独立使用一个.ui 文件进行界面设计。采用在主窗口中直接显示歌词的文本框。使得主窗口组件模块化，更加有条理。




```cpp
class LyricLine
{
public:
    qint64 time;
    QString text;
    LyricLine(qint64 time, QString text):time(time), text(text){}
};

class LyricWidget : public QWidget
{
    Q_OBJECT
    vector<LyricLine> lines;         //储存所有歌词
public:
    explicit LyricWidget(QWidget *parent = nullptr);
    ~LyricWidget();
    bool process(QString filePath); //将歌词文件的内容处理为歌词结构的QList
    int getIndex(qint64 position);  //根据时间找到对应位置的歌词
    void show(qint64 position);     //显示当前播放进度的歌词
    QString getLyricText(int index);//根据下标获得歌词内容
    void clear();                   //清空歌词Label
private:
    Ui::LyricWidget *ui;
};
```

2.2.6 MainWidget 类（主窗口）

> 主窗口，用户直接交互的一个窗口组件，负责初始化、整个播放器各模块的协调与关联、响应用户交互事件等等工作，以此实现对应的功能。主要使用.ui在Qt Designer里完成界面设计



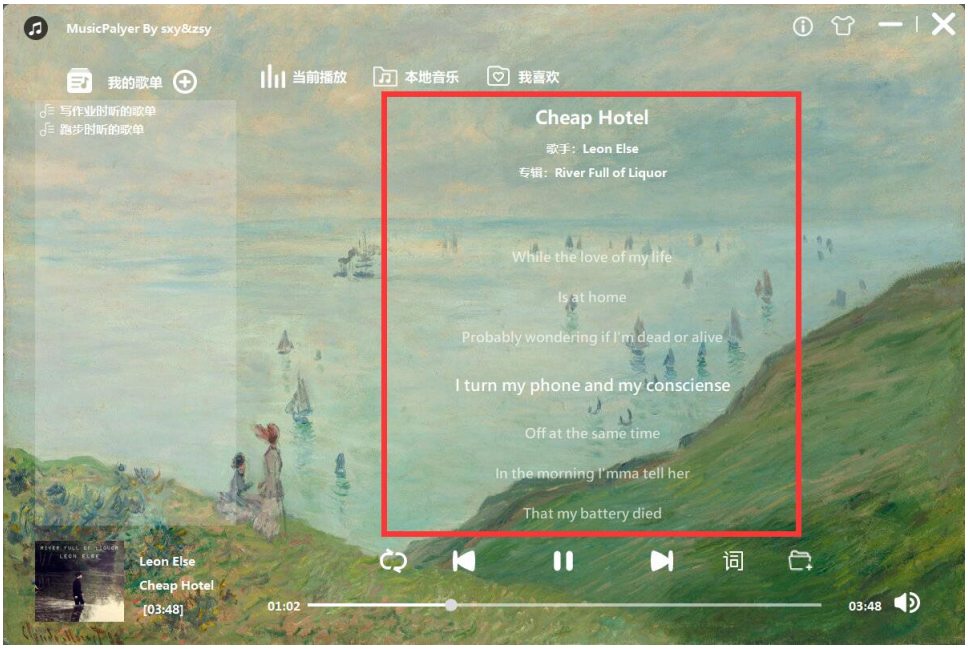
```cpp
class MainWidget : public QWidget
{
    Q_OBJECT
public:
    explicit MainWidget(QWidget *parent = nullptr);
    ~MainWidget() override;
private:
    Ui::Widget *ui;
    void paintEvent(QPaintEvent *event) override;
    void init_UI();                     //UI组件额外的一些处理
    QMediaPlayer *player;               //当前播放器
    QMediaPlaylist *playlist;           //当前播放列表
    void init_play();                   //初始化一些成员变量以及connect连接
    void quitMusicPlayer();             //退出应用
    void init_systemTrayIcon();         //系统托盘初始化
    void init_sqlite();                 //数据库初始化
    void init_settings();               //配置初始化（配置文件读取）
    void init_musicList();              //"本地音乐"、"我喜欢"等歌单的初始化
    vector<MusicList> musiclist;        //歌单
    int musiclist_index=-1;             //用于标识现在展示的是哪个歌单
    void namelist_refresh();            //更新展示歌单名字的listwidget
    void musicListWidget_refresh(); //用于更新展示歌单内容的listwidget

    /*右键菜单*/

    void init_actions();                //菜单项的初始化
    QMenu *menu_playlist;               //"当前播放"列表的右键菜单
    QMenu *menu_locallist;              //"本地音乐"列表的右键菜单
    QMenu *menu_favorlist;              //"我喜欢"列表的右键菜单
    QMenu *menu_namelist;               //"歌单名列表"的右键菜单
    QMenu *menu_musiclist;              //"歌单展示列表"的右键菜单
    QMenu *menu_changeSkin;             //更换皮肤的菜单
protected:
    QPoint offset;                      //窗口拖动时记录的起始点
    /*重写Widget的一些方法*/
    //实现窗口可拖动
    void mousePressEvent(QMouseEvent *event) override;
    void mouseMoveEvent(QMouseEvent *event) override;
    void mouseReleaseEvent(QMouseEvent *event) override;
    void closeEvent(QCloseEvent *event) override;
                                        //关闭时不退出，而是到系统托盘
    void dragEnterEvent(QDragEnterEvent* event) override;
                                        //拖拽文件进入
    void dropEvent(QDropEvent* event) override;

```

```cpp
private slots:
    /*部分右键菜单项对应的操作（即对应QAction连接的槽函数）*/
    void playlist_removeMusic();    //当前播放列表-右键菜单 移除歌曲
    void play_to_favor();           //从当前播放添加到我喜欢
    void local_to_favor();          //从本地音乐添加到我喜欢
    void local_to_playlist();       //从本地音乐添加到当前播放列表
    void favor_to_playlist();       //从我喜欢添加到当前播放列表
    void namelist_delete();         //移除歌单
    void musiclist_removeMusic();   //从歌单展示列表移除歌曲
    void musiclist_to_favor();      //从当前歌单添加到我喜欢
    void musiclist_to_playlist();   //从当前歌单添加到正在播放
    void background_to_default();   //换到默认背景
    void background_setting();      //自定义背景

    /*一些点击事件的响应（使用.ui中的部件"转到槽"自动生成）*/
    void on_btnCurMusic_clicked();
    void on_btnLocalMusic_clicked();
    void on_btnFavorMusic_clicked();
    void on_btnQuit_clicked();
    void on_btnMin_clicked();
    void on_btnPlay_clicked();
    void on_btnNext_clicked();
    void on_btnPre_clicked();
    void on_btnPlayMode_clicked();
    void on_btnAdd_clicked();
    void on_btnVolume_clicked();
    void on_volumeSlider_valueChanged(int value);
    void on_btnAddMusicList_clicked();
    void on_playListWidget_doubleClicked(const QModelIndex &index);
    void on_localMusicWidget_doubleClicked(const QModelIndex &index);
    void on_favorMusicWidget_doubleClicked(const QModelIndex &index);
    void on_playListWidget_customContextMenuRequested(const QPoint &pos);
    void on_localMusicWidget_customContextMenuRequested(const QPoint &pos);
    void on_favorMusicWidget_customContextMenuRequested(const QPoint &pos);
    void on_nameListWidget_customContextMenuRequested(const QPoint &pos);
    void on_nameListWidget_doubleClicked(const QModelIndex &index);
    void on_btnSkin_clicked();
    void on_btnAddtoMusicList_clicked();
    void on_musicListWidget_doubleClicked(const QModelIndex &index);
    void on_musicListWidget_customContextMenuRequested(const QPoint &pos);
    void on_btnAddtoFavor_clicked();
    void on_btnNeaten_clicked();    //整理歌单按钮
    void on_btnNeaten_2_clicked();
    void on_btnNeaten_3_clicked();
    void on_btnTitle_clicked();
    void on_btnLyric_clicked();
```

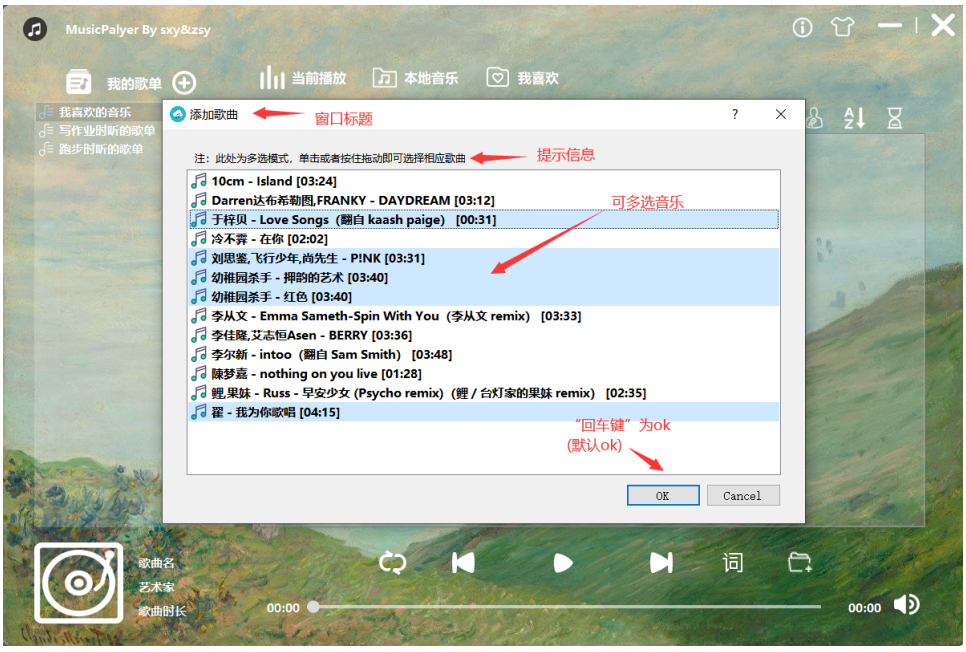
3 界面设计

3.1 主页面

3.2 歌词界面



3.3 ”添加音乐至歌单“界面



4 功能实现简述

5 亮点和小结

5.1 代码亮点简述

以下简述一下本项目的一些亮点（仅代表小组两人观点）：

1. ……
2. ……
3. ……

4.

5.2 项目小结/心得