数据结构课程（荣誉班）

加分项目说明文档

——音乐播放器

组员姓名：沈 星 宇

张 思 源

指导教师：张 颖

学 院：软件 学院

同济大学
TONGJI UNIVERSITY

目录（可跳转）

项目开发文档.....5

1 功能分析.....5

2 类结构设计.....6

2.1 各个类之间的关系.....6

2.2 各个类详细展示.....7

2.2.1 Music 类（歌曲类）.....7

2.2.2 MusicList 类（歌单类）.....8

2.2.3 MusicListWidget 类（歌单窗口）.....9

2.2.4 MusicListDialog 类（”添加至歌单“时弹出的对话框）.....10

2.2.5 LyricWidget 类（歌词显示的窗口）.....11

2.2.6 MainWidget 类（主窗口）.....12

3 界面设计.....13

3.1 主页面.....13

3.2 歌词界面.....13

3.3 ”添加音乐至歌单“界面.....14

3.3 ”系统托盘“显示界面.....14

4 功能实现简述.....15

4.1 可执行文件.exe 的打包.....15

4.2 歌曲信息的获取.....15

4.3 专辑图片显示的实现.....16

4.4 滚动歌词的显示.....16

4.5 自定义部件的使用.....17

4.6 QSS 的使用.....17

4.7 系统托盘的实现与联动 18
4.8 项目数据库的实现 19
4.9 歌单之间的互相添加 19
4.10 自定义背景图片 20
5 亮点和小结 21
5.1 代码亮点简述 21
5.2 项目小结/心得 21

MusicPlayer 项目说明文档

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：

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

细节功能：

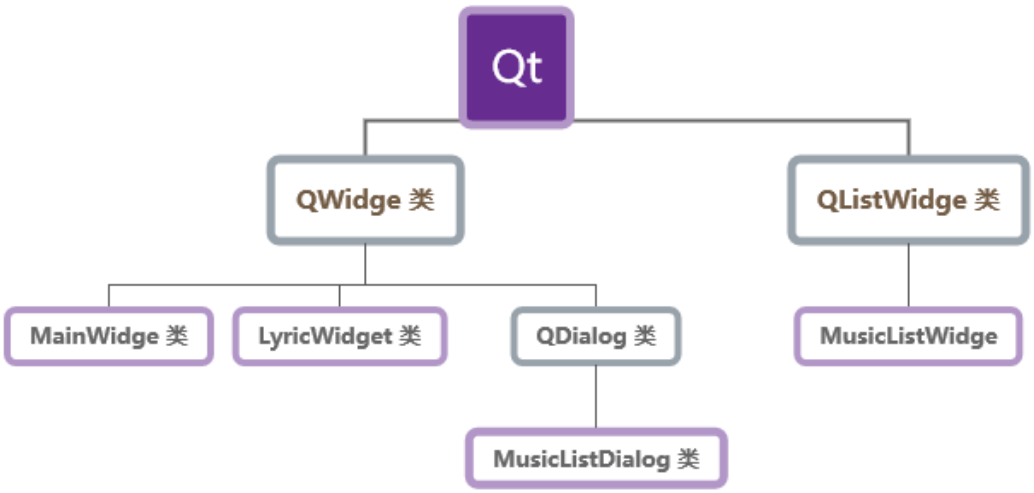
- 鼠标悬浮在按钮上时有提示信息
- 任务栏右击进行任务选择
- 主页面通过”about“来展示项目基本信息

2 类结构设计

2.1 各个类之间的关系

项目中用到的类如下：

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



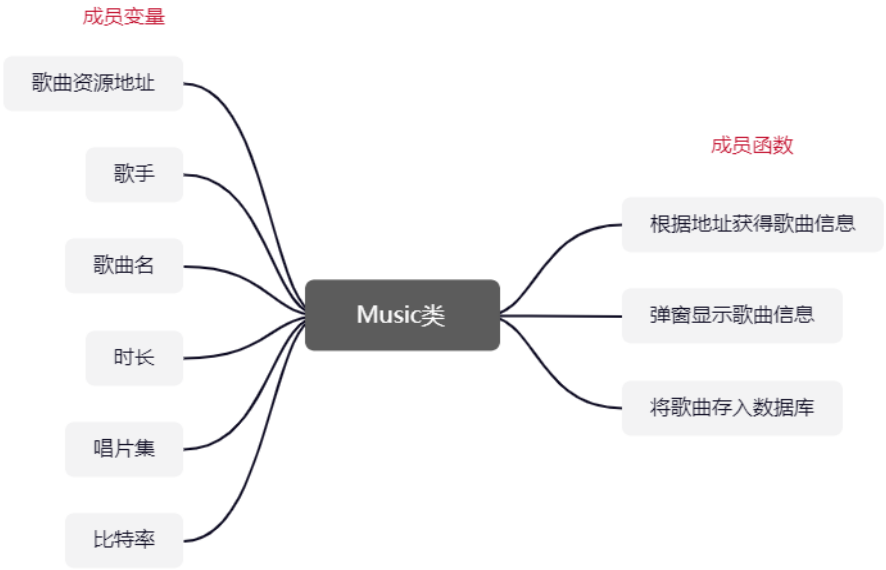
解释：灰色边框的类是 Qt 提供的可继承的类，紫色边框的为本项目从原有类中派生出来的类，详细的接口、成员与功能在 2.2 阐述。

2.2 各个类详细展示

2.2.1 Music 类（歌曲类）

功能：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

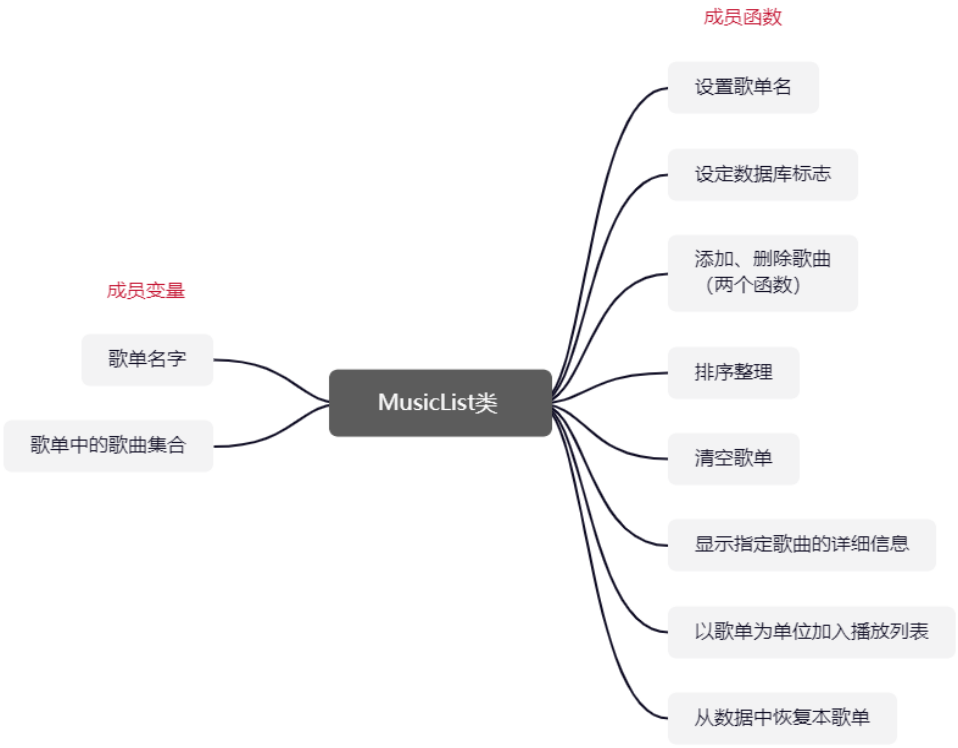
简述：其中最为重要的是歌曲的 url 信息（即：歌曲文件所在的路径），播放的时候需要根据 url 来将歌曲文件加入播放器。还记录下的一些其他的歌曲信息，可见于下图。




```
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）

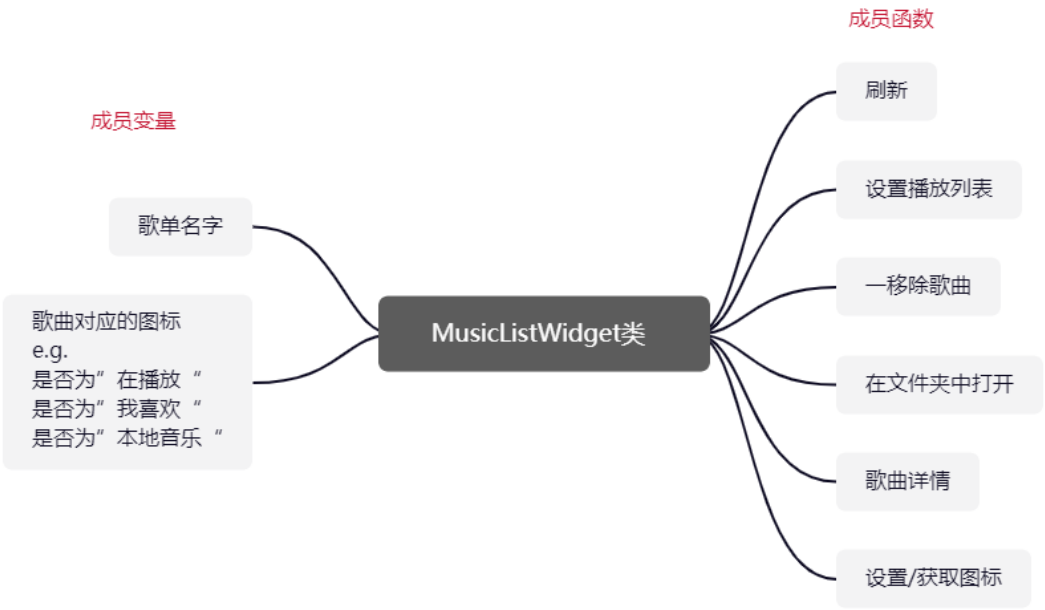
功能：歌曲列表（即歌单），是用容器 vector 将 music 类的对象存储起来，由此可以进行一些具体的、针对多首歌曲的操作




```
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）

功能简述：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。




```
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）

功能：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。

简述：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；

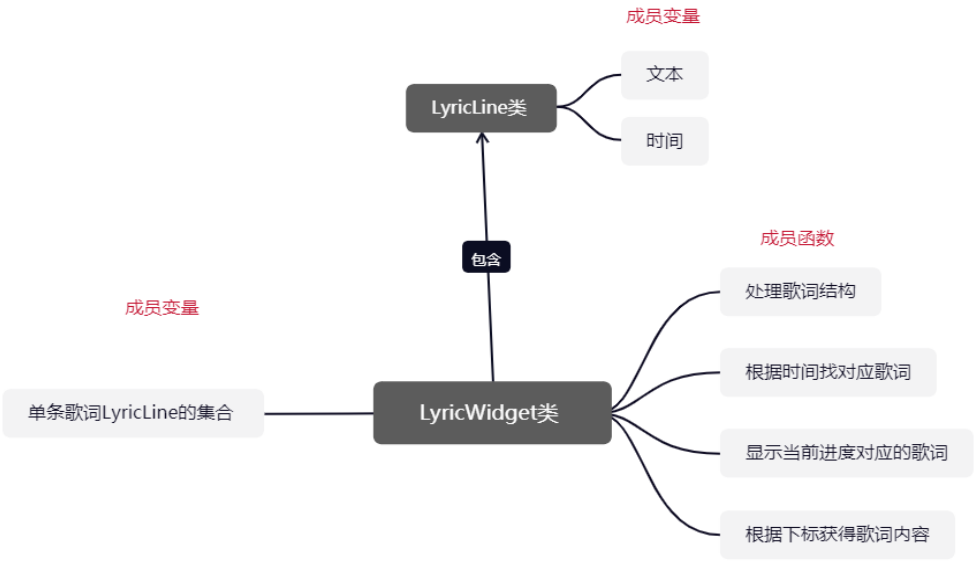
②便于获取歌曲数据（即本地音乐有哪些歌曲），以及返回选择结果。


```
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）

功能：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）。

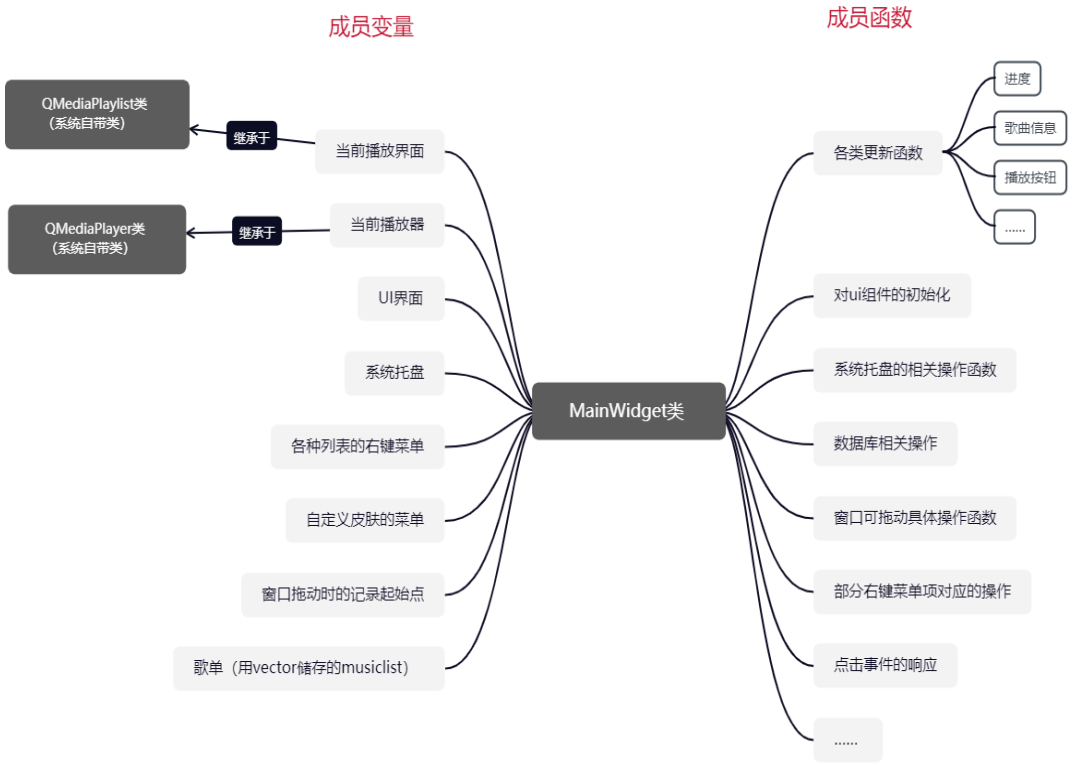
简述：这里使用自定义的 Widget，也是便于独立使用一个.ui 文件进行界面设计。采用在主窗口中直接显示歌词的文本框。使得主窗口组件模块化，更加有条理。




```
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）

功能：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



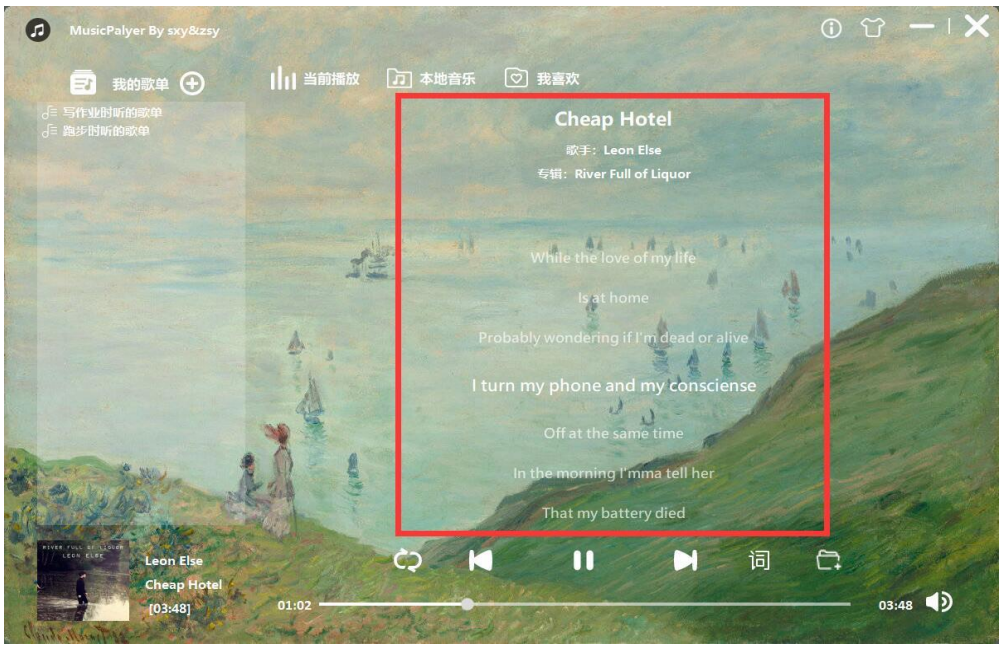

```
class MainWidget : public QWidget
{
    Q_OBJECT
public:
    explicit MainWidget(QWidget *parent = nullptr);
    ~MainWidget() override;
private:
    Ui::Widget *ui;
    void paintEvent(QPaintEvent *event) override;
    void init_UI();                     //UI组件额外的一些处理
    QMediaPlayer *player;               //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
    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
    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listwidg

    /*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;

```

```
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();
```

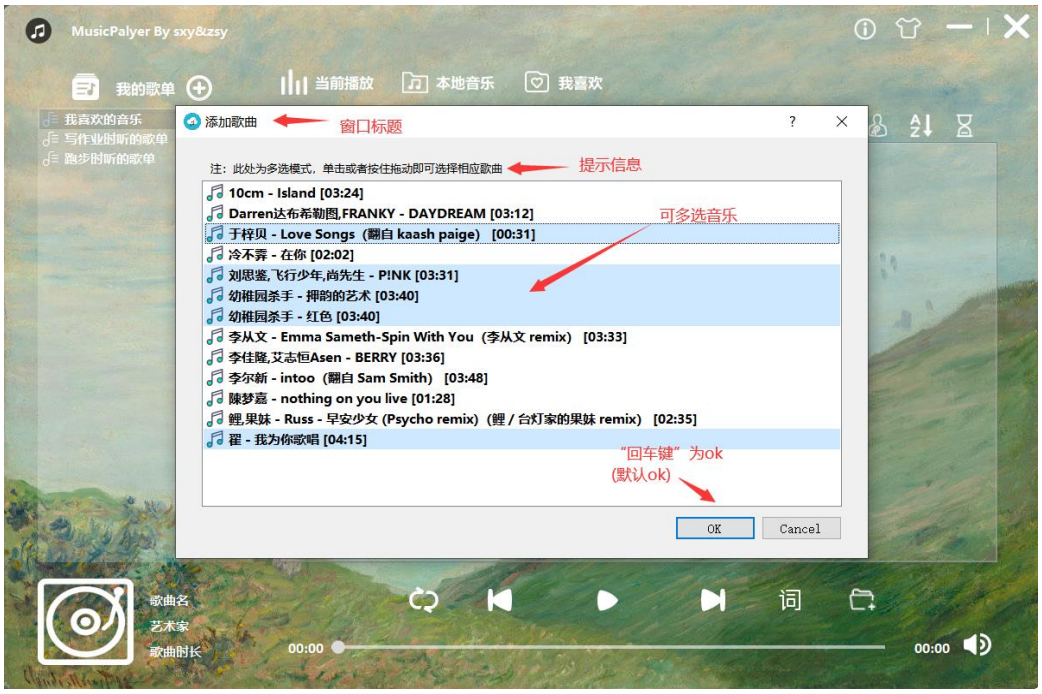
3 界面设计

3.1 主页面

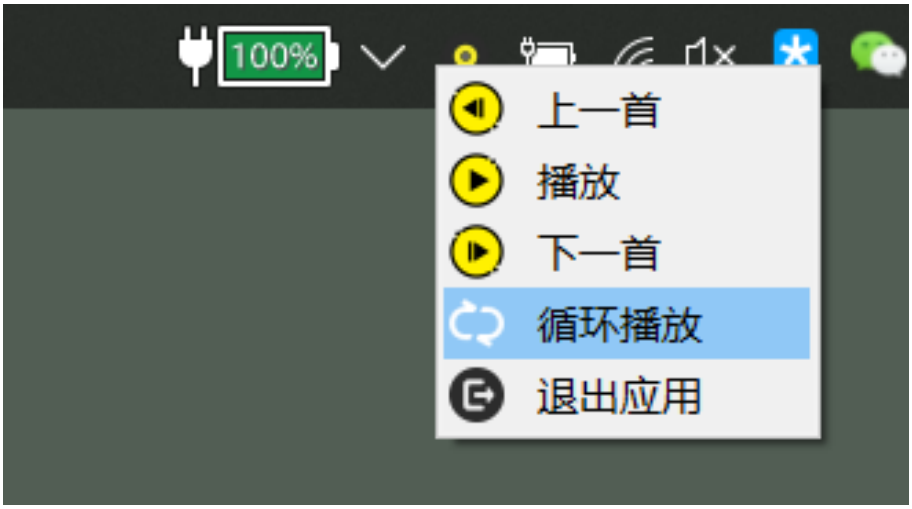
3.2 歌词界面



3.3 ”添加音乐至歌单“界面



3.3 ”系统托盘“显示界面



4 功能实现简述

4.1 可执行文件.exe 的打包

1. 可执行程序为单个.exe 文件。使用“Desktop Qt 5.9.8 MinGW 32bit”构建套件在 Release 模式下生成，然后使用 Enigma Virtual Box 软件将动态 DLL 库文件及其他相关文件打包生成了单个的一个.exe 文件。

2. 第一次运行后，会在同目录下产生两个文件：

a) Music.db 文件。SQLite 本地数据库文件，存储已添加到播放器的歌曲的信息；

b) LightMusicPlayer.ini 文件。保存用户设置信息，即如果设置了自定义背景图片，会在此配置文件中记录下图片文件的路径。

4.2 歌曲信息的获取

歌曲信息的获取通过歌曲的 url 地址得到（url 含义见下图）


统一资源定位符 [编辑]

维基百科，自由的百科全书

统一资源定位符（英语：Uniform Resource Locator，缩写：**URL**；或称**统一资源定位器**、**定位地址**、**URL地址**[1]，俗称**网页地址**或简称**网址**）是因特网上标准的资源的地址（Address），如同在网络上的门牌。它最初是由蒂姆·伯纳斯-李发明用来作为万维网的地址，现在它已经被万维网联盟编制为因特网标准RFC 1738。

代码展示：

```
extern QString formatTime(qint64 timeMilliSeconds);
void Music::refreshInfo()
{
    QMediaPlayer tempPlayer;
    tempPlayer.setMedia(url);
    //元数据的解析需要时间，所以这里需要循环等待（但同时需要保持Qt事件处理机制在运行）
    while(!tempPlayer.isMetaDataAvailable()){
        QCoreApplication::processEvents();
    }
    QStringList list=tempPlayer.availableMetaData();//调试时查看有哪些元数据可用
    if(tempPlayer.isMetaDataAvailable()){
        //歌曲信息
        author = tempPlayer.metaData(QStringLiteral("Author")).toStringList().join(",");
        //author = tempPlayer.metaData(QStringLiteral("Author")).toString(); //查手册发现，这里返回的是StringList
        //author=tempPlayer.metaData(QStringLiteral("ContributingArtist")).toStringList().join(","); //另一种元数据
        title = tempPlayer.metaData(QStringLiteral("Title")).toString();
        albumTitle = tempPlayer.metaData(QStringLiteral("AlbumTitle")).toString();
        audioBitRate = tempPlayer.metaData(QStringLiteral("AudioBitRate")).toInt();
        duration=tempPlayer.duration();
    }
}
```

歌唱者
歌曲名
专辑名称
比特率
歌曲长度

4.3 专辑图片显示的实现

平时从 QQ 音乐等播放器上下载的 MP3 歌曲文件，在 Windows 资源管理器中查看的时候，会有图片显示（专辑图）。所以猜想，歌曲的图片信息也是存储在 MP3 文件中，那 Qt 的库函数可以解析出来吗？最终，查阅手册找到了方法。只需要在解析歌曲文件时候，获取名为" ThumbnailImage" （缩略图）的元数据即可。

```
//封面图片（应获取"ThumbnailImage" From: https://www.zhihu.com/question/36859497）
        QImage picImage= player->metaData(QStringLiteral("ThumbnailImage")).value<QImage>();
        if(picImage.isNull()) picImage=QImage(":/image/image/image/non-music.png");
        ui->coverLabel->setPixmap(QPixmap::fromImage(picImage));
        ui->coverLabel->setScaledContents(true);
```

4.4 滚动歌词的显示

这里使用了 Qt 中的正则表达式来解析标准的 LRC 歌词文件，主要是需要识别出歌词的时间点，如下图。然后在播放的时候，根据播放进度，比较时间点的先后，在对应的 QLabel 上展示出相应位置的歌词。 **(参考博客：**https://www.cnblogs.com/wanghuixi/p/9540694.html **)**

1. 某歌曲的 LRC 文件截图：

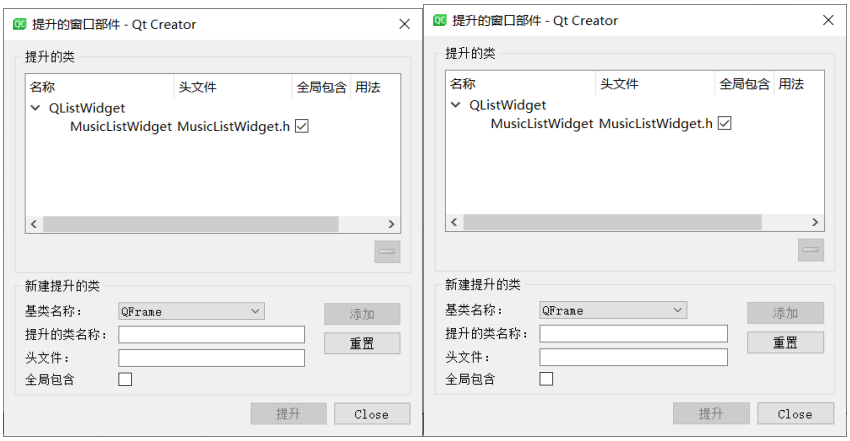
2. 对应的项目中代码（定位以[d:d.d]字符串开头的字符串）：

```
    //先清空歌词
    lines.clear();

     //用来查找时间标签的正则表达式
    const QRegExp rx("\\[(\\d+):(\\d+(\\.\\d+)?)\\]");

    // 步骤1
    int pos = rx.indexIn(content);
```

4.5 自定义部件的使用

主窗口的界面设计主要使用.ui 文件完成，而主窗口中的部分组件使用的是自定义的部件（比如 MusicListWidget 和 LyricWidget），但是 Qt Designer 左侧可拖拽出的部件都是 Qt 中的自带部件，那么如何使用自定义部件在 Qt Designer 中完成设计呢?

这里可以使用“部件提升”功能，在右侧的“对象与类”窗口，右键，选择“提升为”，即可将某个 Qt 中的标准部件，修改为使用自定义的部件（该部件必须继承自对应的标准部件）。



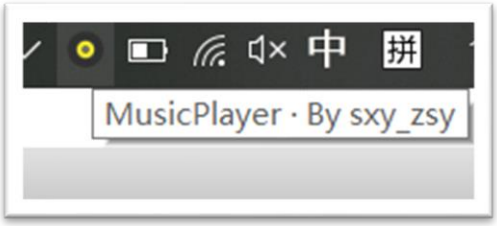
4.6 QSS 的使用

QSS 称为 Qt Style Sheets 也就是 Qt 样式表，它是 Qt 提供的一种用来自定义控件外观的机制。QSS 大量参考了 CSS 的内容，只不过 QSS 的功能比 CSS 要弱很多，体现在选择器要少，可以使用的 QSS 属性也要少很多，并且并不是所有的属性都可以用在 Qt 的所有控件上。

（参考博客：https://www.cnblogs.com/wangqiguo/p/4960776.html#_label6）

在程序当中，为了更加贴近于我们日常使用的音乐播放器的界面，使用了 CSS，例如主页面针对歌曲“上/下一曲、播放/暂停”等都使用了 CSS

4.7 系统托盘的实现与联动



1.鼠标悬浮于系统托盘时，显示项目名称和作者

2. 鼠标单击右键，显示系统托盘菜单栏。

实现“上一首”“播放”“下一首”和“播放模式”“退出应用”几项功能

代码实现：

```
605 void MainWidget::init_systemTrayIcon()
606 {
607     mySystemTray=new QSystemTrayIcon(this);
608     mySystemTray->setIcon(QIcon(":/image/image/image/systemTrayIcon.png"));
609     mySystemTray->setToolTip(u8"MusicPlayer · By sxy_zsy");   显示项目信息
610     connect(mySystemTray,&QSystemTrayIcon::activated,this,&MainWidget::systemTrayIcon_activated);
611     //添加菜单项
612     QAction *action_systemTray_pre = new QAction(QIcon(":/image/image/image/pre2.png"), u8"上一首");
613     connect(action_systemTray_pre, &QAction::triggered, this, &MainWidget::on_btnPre_clicked);
614     action_systemTray_play = new QAction(QIcon(":/image/image/image/play2.png"), u8"播放");
615     connect(action_systemTray_play, &QAction::triggered, this, &MainWidget::on_btnPlay_clicked);
616     QAction *action_systemTray_next = new QAction(QIcon(":/image/image/image/next2.png"), u8"下一首");
617     connect(action_systemTray_next, &QAction::triggered, this, &MainWidget::on_btnNext_clicked);
618     action_systemTray_playmode = new QAction(QIcon(":/image/image/image/loop2.png"), u8"循环播放");
619     connect(action_systemTray_playmode, &QAction::triggered, this, &MainWidget::on_btnPlayMode_clicked);
620     QAction *action_systemTray_quit = new QAction(QIcon(":/image/image/image/exit.png"), u8"退出应用");
621     connect(action_systemTray_quit, &QAction::triggered, this, &MainWidget::quitMusicPlayer);
622
623     QMenu *pContextMenu = new QMenu(this);
624     pContextMenu->addAction(action_systemTray_pre);
625     pContextMenu->addAction(action_systemTray_play);
626     pContextMenu->addAction(action_systemTray_next);
627     pContextMenu->addAction(action_systemTray_playmode);
628     pContextMenu->addAction(action_systemTray_quit);
629     mySystemTray->setContextMenu(pContextMenu);
630     mySystemTray->show();
631 }
632
```

为图标设置图片

上一曲
播放
下一曲
播放模式
退出应用

4.8 项目数据库的实现

项目数据库采用 Qt 提供的 **QSqlDatabase 类**

```
181 void MainWidget::init_sqlite()
182 {
183     QSqlDatabase database;
184     if (QSqlDatabase::contains("qt_sql_default_connection"))
185     {
186         database = QSqlDatabase::database("qt_sql_default_connection");
187     }
188     else
189     {
190         database = QSqlDatabase::addDatabase("QSQLITE");
191         database.setDatabaseName("Music.db");
192         database.setUserName("sxyzsy");
193         database.setPassword("123456");
194         if (!database.open())
195         {
196             QMessageBox::critical(this,"无法打开数据库文件：Music.db",database.lastError().databaseText());
197             exit(-1);
198         }
199     }
```

创建一个数据库实例

设置数据库名字

设置用户名称 sxyzsy（沈星宇，张思源）

设置数据库密码

4.9 歌单之间的互相添加

1. 总览：

```
343
344 void MainWidget::play_to_favor() {...}
350
351 void MainWidget::local_to_favor() {...}
357
358 void MainWidget::local_to_playlist() {...}
367
368 void MainWidget::favor_to_playlist() {...}
377
```

从“正在播放”添加至“我喜欢”

从“本地音乐”添加至“我喜欢”

从”本地音乐“添加至“正在播放”

从“我喜欢”添加至“正在播放”

2. 以“从‘正在播放’添加至‘我喜欢’”为例：

```
344 void MainWidget::play_to_favor()
345 {
346     int pos=ui->playListWidget->currentRow();
347     ui->favorMusicWidget->musicList.addMusic(ui->playListWidget->musicList.getMusic(pos));
348     ui->favorMusicWidget->refresh();
349 }
```

从ui界面中获取当前音乐所在的行数作为其地址

添加音乐至“我喜欢”

刷新“我喜欢”列表

4.10 自定义背景图片

本项目支持用户对主页面的背景图片进行个性化选择，通过弹出文件对话框以供读者选取自己喜欢的图片作为背景图片，也可随时换回默认背景

核心代码展示：

```
void MainWidget::background_setting()
{
    //从默认图片位置打开文件选择框
    QString fileName=QFileDialog::getOpenFileName(this,("选择自定义背景图片"),QStandardPaths::standardLocations(QStandardPaths::F
    if(!fileName.isEmpty())
    {
        QImage testImage(fileName);
        if(!testImage.isNull()){
            QSettings mysettings("./LightMusicPlayer.ini",QSettings::IniFormat);
            mysettings.setIniCodec("UTF8");
            mysettings.setValue("background/image-url",fileName);
            setStyleSheet(QString("QWidget#Widget{"
                                  "border-radius:10px;"
                                  "border-image: url(%1);}").arg(fileName));
        }
    }
}
```

调出文件对话框（默认从“图片”打开）

如果图片不为空
将其设置为当前背景图片

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. 项目实现了音乐播放器的一些基本功能，无论是播放音乐，还是歌曲的播放模式，抑或是歌单的管理，都满足了用户的基本需求，是一个功能较为完善的轻量级“音乐播放器”
2. 项目的设计界面，结合 QSS 以及自定义控件，完成了与我们平时使用的音乐播放器十分接近的交互界面，体验感较好
3. 项目注重细节设计，主页面上的大部分按钮均设置了提示信息，鼠标悬浮在上面也有对应的显示（比如部分按钮会在鼠标预选中时变颜色）
4. 整个项目的思路较为清晰，类划分明确
5. 项目的说明文档配备有大量详细而形象的思维导图、代码截图以及相应的文字说明，便于用户理解项目。

5.2 项目小结/心得

本项目作为数据结构课的加分项，允许在网上寻找相关案例，于是我们便找到了一位南大的“唐同学”的博客，在全面理解了原项目的构建思路及代码含义后，进行了一些功能的完善与修改，期间收获很多，一方面是看到别人的优秀代码后反思自己的不足，另一方面对自己能力也确实有了很大提升。

小组两人这次都是第一次接触 Qt 这个平台，相比于大一的程序设计课使用的“cocos-2dx”，Qt 的使用感更胜一筹，因为有了 Qt Designer 的可视化的界面设计平台，能够通过很直观的方式达到自己想要的效果，比起使用代码布局来，不知方便了多少。

最后，也是很重要的一点就是，通过这次项目，小组二人对于数据结构的掌握更深入了一步，虽然“对象树”与我们课程中所学的“树”不完全是一回事，但对于继承等操作，却有着异曲同工之妙，强化了我们对“树”这种重要数据结构的认识……

收获很多，在此就不再继续列举了，总之，我们从这次项目中对自己的理论认知、项目构建能力，甚至是问题检索能力，都有着很大提升，希望以后也能抓住这类机会，积极锻炼自己。