南京大学课程设计报告

高级程序设计项目报告（三）

音乐播放器 MyMediaPlayer

学　　号：181840070

院　　系：计算机科学与技术系

姓　　名：葛睿芃

提交日期：2019/12/31

目录

目录

一、项目概述

1.1 选题背景

音乐播放器是现今人们最常用的软件之一，它不仅能够播放音乐，还能对用户设置的歌单列表进行一系列读写管理操作。本项目“小蓝鲸播放器”仿制了一个简化版本的音乐播放器。

音乐播放器中，不仅包含了对歌单的查找、读取、排序等对于内存的操作，也包含读取存储在本地的用户歌单、将歌单保存在本地，以及用户播放本地文件歌曲等对于磁盘的操作。

本项目要求实现一个 GUI 项目的程序，由于之前从来没有过 GUI 项目编程的经历，于是我选择了一个相对简单的 Qt 来设计程序，现学现做。在项目设计上，在歌单操作等底层设计采用了最基础的 C++语法，除 QString 和 QFile 外，没有使用任何与 Qt 相关的 API 和特性。关于歌单和歌曲播放等操作的主要接口则定义在 MainWidget.cpp 的 MianWidget 类的成员函数中，所有 cpp 文件代码总和为 1364 行。

1.2 需求分析

本题要求实现一个简化版本的 GUI 音乐播放器 MyMediaPlayer，能够完成以下的功能。

- 实现对于歌曲标题、歌手、评分、播放量、时长等关键字的实现。
- 能够实现对标题、歌手关键字的信息修改和搜索功能。
- 能够根据本地文件构建歌单，并将信息保存在本地磁盘文件中。
- 能够实现对歌单的增、删、改、移动、根据关键字排序。
- 能够根据用户对歌单的操作，实现对本地音乐文件的播放。
- 能够实现音乐的播放、暂停、上下切换、播放模式（循环、单曲、随机）的切换，定位并修改歌曲位置、音量等功能。
- 实现“缩略播放窗口”的功能。
- 友好的用户界面。

二、界面展示

本项目的报告在介绍实现之前，首先展示一下设计成果。本项目最终的主界面如下：

整个界面分三个部分，分别为左侧菜单栏，右侧歌单部分，和下面的音乐播放控件部分。下面将介绍本项目可以实现的所有操作：

2.1 播放控件操作展示

在下面的播放控件中，中间的歌曲进度条可以实时追踪当前正在播放歌曲的位置，并显示在进度条上，并且拖动进度条也可以用来修改歌曲定位。这个进度条是用 Qt 控件 HorizontalSlider 制作的。右侧的 Label 可以用于显示歌曲时长和当前歌曲定位。进度条上面的 Label 会显示当前正在播放的歌曲名和歌手名。

歌曲进度条右侧的音量控制条也是用 HorizontalSlider 做的，用于实时改变歌曲播放的音量。

左侧的三个按钮用于播放/暂停和上下曲切换，它们本身是 PushButton，在被点击的时候可以执行相应的操作。

音量右侧的按钮可以切换“列表循环”“单曲循环”和“随机播放”的播放模式，这部分实现主要是用 Qt 中 QMediaPlayList 类的 PlayBackMode 参数控制，但是在歌单被修改的特殊情况下会有所不同，具体请参考本报告的设计思路部分。

2.2 歌单操作展示

2.2.1 删除、移动歌曲

在程序运行开始，会从本地文件中读取歌单信息，生成歌单，歌单本质上式 QTableView 控件，用户可以选中歌单中的一行（一首歌），通过**双击**播放歌曲（此时正在播放的歌曲会被强制中断）。被播放的歌曲会显示【正在播放】字样在歌单中使用右键呼出菜单，可以进行“修改信息”，“删除歌曲”，“上移”和“下移”的操作。在移动或操作过程后，歌曲的编号（id）会相应的发生变化。用户不可以对正在播放的歌曲作修改信息或删除操作，但是可以对其上下移动。删除歌曲操作会相应地删除本地歌曲文件

下图中展示了移动歌曲操作前后的界面情况

9	SpiderDance	TobyFox	6	3.0	01:46	评分
10	RISE【正在播放】	The_Glitch_Mob,Mako,…	6	4.0	03:12	评分
11	以梦为马	于济洲	5	5.0	04:06	评分
12	恋爱循环	花泽香菜	6	0.0	04:13	评分
13	HandClap	Fitz_&_The_Tantrums	6	0.0	03:13	评分

下图中展示了删除歌曲前后的界面情况

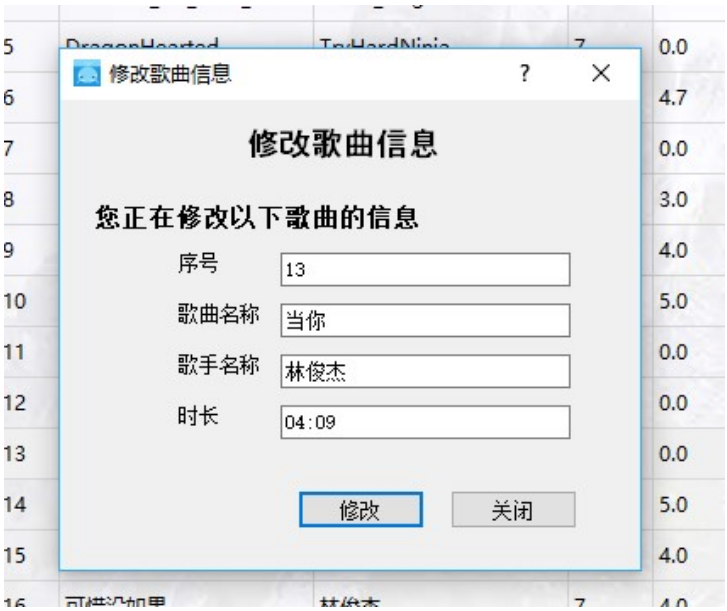
序号	歌曲名	歌手	播放量	评分	时长	
1	Hop	Azis	126	4.8	03:07	评分
2	ASGORE	TobyFox	48	3.7	02:36	评分
3	Revenge		20	4.4	03:40	评分
4	CheerS		23	4.6	04:02	评分
5	Zombies_On_Your_Law		12	4.0	02:39	评分
6	DragonHearted	TryHardNinja	7	0.0	04:56	评分

修改信息
删除歌曲
上移
下移

序号	歌曲名	歌手	播放量	评分	时长	
1	Hop	Azis	126	4.8	03:07	评分
2	Revenge	TryHardNinja	20	4.4	03:40	评分
3	CheerS	ClairS	23	4.6	04:02	评分
4	Zombies_On_Your_Lawn	Laura_Shigihara	12	4.0	02:39	评分
5	DragonHearted	TryHardNinja	7	0.0	04:56	评分

2.2.2 修改歌曲信息

在修改歌曲信息部分，我们创建了一个新的对话框类 ModifyDialog 类，在点击修改信息按钮时，呼出对话框并让用户修改歌曲信息，用户只能对歌曲名和歌手信息进行修改，不能修改其他信息。信息修改完成后，本地音乐文件名也会被相应修改。



2.2.3 评分系统

单击歌单中歌曲右侧的**“评分”按钮**，呼出评分对话框 ScoreDialog，可以实现对歌曲评分，用户评分可以修改当前歌曲评分并实时显示在歌单中。

2.3 菜单操作展示

左侧的菜单栏可以实现以下操作：

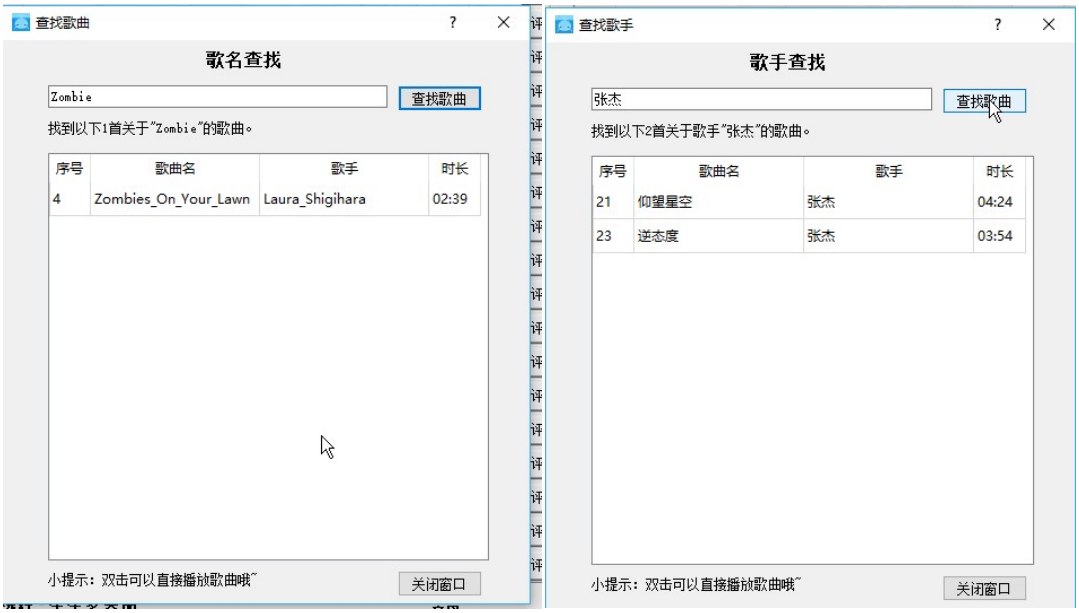
2.3.1 添加本地歌曲

点击“**添加歌曲**”**按钮**，可以将本地音乐文件添加到歌单中，添加歌曲的时候会要求用户对歌曲信息进行设置（修改默认信息），同时检查用户设置歌曲信息是否有冲突或符合规范。用户添加完毕后，将自动播放该歌曲。

2.3.2 查找功能展示

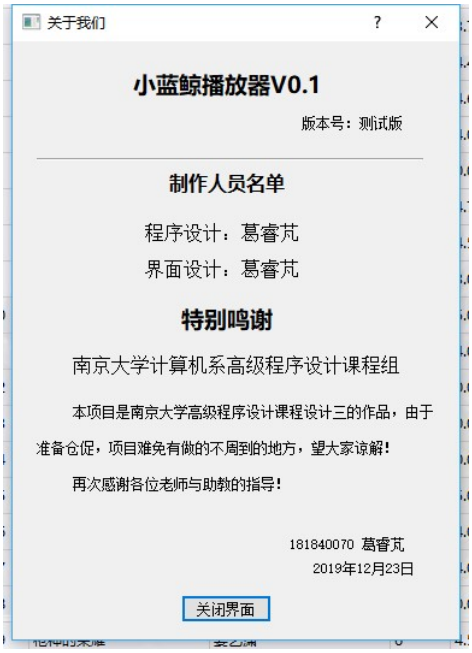
播放器可以支持按照歌曲名和歌手名模糊查找歌曲，以歌曲查找为例，点击“**歌曲查找**”**按钮**，输入关键字查找，下面的表格中会显示满足查找要求的所有歌曲，在表格中**双击**歌曲可以播放歌曲。

按歌曲和歌手查找窗口展示如下：



2.3.3 关于我们

这是一个彩蛋窗口，展示制作人员和鸣谢，没有什么功能上的实际价值。



2.4 缩略窗口

我们在听音乐的时候，往往不希望这么大的一个音乐播放器窗口永远置于顶端，但是每次我们想要暂停或者切歌、调整播放器音量的时候总要切出窗口，这样总会显得很麻烦。很多音乐播放器都带有了“缩略窗口”的功能，缩略窗口往往很小，不会遮挡当前计算机的工作界面，而且还可以让用户实现对歌单的简单操作（暂停、按顺序切歌、改变音量和播放方式等）。

“小蓝鲸播放器”也制作了一个简单的缩略窗口，由 Widget 窗口改造，禁用了最大化和关闭按钮，并设置窗口始终前端显示。这样用户可以在工作的时候，将缩略窗口放置在角落中，既不会遮挡工作界面，又能实时暂停、切歌，非常方便。

点击小蓝鲸 logo 下面的“↙”标识的按钮，即可进入缩略状态，缩略窗口和主窗口的显示是通过信号槽同步的，缩略窗口状态下点击“□”标识的按钮返回主窗口。

三、设计思路

本项目是我写的第一个 GUI 程序，用了 Qt Designer 的相应功能，所以我在设计这个项目的时候，基本上处于“摸着石头过河”的状态，做一步学一步，怎么调用 API 和信号槽对我来说都是一个比较陌生的工作。加之各种 DDL 接踵而至，真正花在本项目上的编程时间其实不是很多，所以项目相比来说就比较简陋。

其实在本项目上还是有很多能够拓展的地方的，日后有机会我也会考虑扩展一下我的播放器功能。现在在这部分，我会将我构建此项目的心路历程和设计思路详尽地叙述出来。

本项目不是很复杂，所以我就不按照“数据结构+算法”的逻辑来阐释我的设计了，我就单纯从从前到后的设计顺序介绍我的实现。

3.1 底层设计——歌单管理系统

不考虑歌曲播放的因素，本项目在底层实现上，就是一个能够管理各个歌曲

信息的歌单管理系统。说白了还是造一个数据库。

底层设计上，除了我运用了 QString 类代替 string 类和 QFile 文件操作以外，没有用任何 Qt 特性。

首先我们考虑歌曲的信息：对于一个歌曲来说，最重要的信息有：标题、歌手、歌曲时长。除此以外，题目中还要求了播放量、评分的信息，我们也加进去。

然后，对于一个歌曲的操作，除了评分，主要还是获取其私有成员的数值，来为定位歌曲文件提供参数，另外我们对歌曲是否正在被播放设置了一个 bool 标记。

考虑所有数据结构与操作，我们定义歌曲类 SongInfo 如下：

```
1.  class SongInfo{
2.      int id;
3.      QString title;
4.      QString singer;
5.      QString filename;
6.      double star;
7.      int scoreNum;
8.      int playedNum;
9.      bool songStatus; //(0: not playing; 1: playing)
10.     qint64 minute;
11.     qint64 second;
12.     QString display;
13. public:
14.     SongInfo(int id, QString path, QString title, QString singer);//create new
15.     SongInfo(int id, QString title, QString singer, double star, int scoreNum, int playe
        dNum, qint64 minute, qint64 second);//load existing
16.     //other functions, including obtaining private variables.
17. }
```

说明：id 是歌曲的标识，在歌单中基本上都是通过 id 定位歌曲，id 是会根据歌单内容动态变化的。

minute 和 second 变量是存储歌曲时长的变量，定义为 qint64 类型是为了和获取歌曲时长的库函数 qint64 QMediaPlayer::Duration() 兼容，具体会在后面介绍。

另外，filename 存储的是歌曲原文件的拷贝在项目目录下的文件名，本项目支持用户从本地导入歌曲，导入后，我们会根据歌曲标题、歌手，创建一个项目目录下的文件拷贝，拷贝是按照一定命名规则将标题和歌手串起来的格式，因此

添加文件中，我们会检查歌手和标题是否有完全重合的，重合会添加失败。之所以不按照歌曲 id 来命名文件拷贝，是因为歌单会动态变化，id 修改后，可能会有许多歌曲的文件都要重命名，这样非常复杂。同样，由于歌曲文件拷贝的文件名涉及标题和歌手，我们会对用户输入的歌曲信息进行格式检查，一些不应该出现在文件名中的符号和空白符 SPACE TAB \ / : * ? " < > | 都会被替换成下划线'_'。

有了歌曲信息了，类似的，我们就可以定义歌单信息类 SongList ，其中一个成员变量就是歌曲的数组 vector <SongInfo> list，同时我们还有一个指针指向当前正在播放的歌曲。

成员函数方面，除了基础的增、删、改操作，还有歌曲上移、下移，按照关键字模糊搜索歌曲的功能。另外还有获取歌曲指针的辅助函数等，SongList 类具体定义如下：

```
1.  class SongList{
2.      int num;
3.      QString sysfile;
4.      vector <SongInfo> list;
5.      SongInfo* current;
6.  public:
7.      SongList():num(0),sysfile("\0"),current(nullptr){list.clear();}
8.      SongList(QString sysfile);
9.      bool add_to_list(QString path, QString title, QString singer);
10.     bool remove_from_list(int id);
11.     void save_to_file();
12.     vector<SongInfo> searchBySinger(QString singer);  //search by singer name
13.     vector<SongInfo> searchByName(QString name);      //seach by song name
14.     bool sift_up(int id); //adjust position of the song in the songlist
15.     bool sift_down(int id);
16.     bool rename(int id, QString title, QString singer);
17.     //other functions
18.     ~SongList(){list.clear();}
19. };
```

这些定义只有对歌曲文件的操作，不涉及真正的歌曲播放等操作。

在关于歌单操作方面，值得注意的是，我们运用 vector<SongInfo>::erase 或者 vertor<SongInfo>::insert 函数实现插入删除时，**歌曲编号 id 会动态变化**，保证

歌单在每次操作过程后，歌曲编号永远是从 1 开始，后面一首歌比前面一首歌编号多 1 的情况。

底层的实现相对比较简单，这是因为在上层检查异常情况的时候，我们已经都提前把可能出现的问题都考虑在内，尽量保证调用函数的合法性。因此程序鲁棒性方面的考虑主要是在界面层面上实现的。

3.2 主窗口设计——歌单的显示与歌曲的播放

主窗口是本项目设计的一大重点，歌单显示、歌曲操作控件等全部由这一部分定义。

主窗口类 MainWidget 类，继承于 Qt 标准窗口类 QWidget，其中成员变量和成员函数主要分为三类，一为和歌单相关，二为与歌曲播放相关，三为与其他窗口交互的信号与槽，我们将依次对其进行介绍。

注：底层设计的 SongList 类对象 songlist 是其一成员变量，请注意与 QTableView 类对象 songList 区分。

3.2.1 主窗口歌单操作

利用 Qt Designer 设计师界面，我们利用 Qt 库中 QTableView 类对象 songList 来显示歌单，QTableView 可以用于设置并显示一个表格模型，并实行一系列对表格的操作。表格模型是通过 QStandardItemModel 类的动态对象 model 来设置的，它是 MainWidget 类中的一个成员变量，在构造函数中动态分配空间，用于根据底层 vector 向量中的数据来设置歌单信息。

MainWidget 中的成员函数 void showTable(); 就是用于显示歌单的，它会在歌单被修改后调用，来刷新歌单，这个函数的主要操作就是读取底层 SongList 类中的歌单向量，设置 model 信息，函数的主要操作如下所示：

```
1.  for(vector<SongInfo>::iterator it = song.begin();it!=song.end();it++){
2.      int id = it->Id();
3.      QString title = it->Title();
4.      QString singer = it->Singer();
5.      int played = it->PlayedNum();
6.      double star = it->Star();
7.      qint64 min = it->Minute();
```

```
        qint64 sec = it->Second();
        QString name = singer;
        name += " - ";
        name += title;
        if(name==current_playing){
            title+="【正在播放】";
        }
        QStandardItem* item1 = new QStandardItem;
        item1->setData(QVariant(id),Qt::EditRole);
        model->setItem(i,0,item1);
        model->setItem(i,1,new QStandardItem(title));
        model->setItem(i,2,new QStandardItem(singer));
        QStandardItem* item2 = new QStandardItem;
        item2->setData(QVariant(played),Qt::EditRole);
        model->setItem(i,3,item2);
        model->setItem(i,4,new QStandardItem(QString::number(star,10,1)));
        model->setItem(i,5,new QStandardItem(dur_trans(min,sec)));
        model->setItem(i,6,new QStandardItem("normal"));
        QPushButton* button = new QPushButton(tr("评分"));
        button->setProperty("id",id);
        connect(button, SIGNAL(clicked()), this, SLOT(onScoreBtnClicked()));
        ui->songList->setIndexWidget(model->index(i, 6), button);

        i++;
    }
```

至于排序，运用 QTableView 本身的库函数 setSortingEnabled(bool); 即可，关于歌曲的上移和下移，调用底层函数 bool SongList::sift_down(int id); 和 bool SongList::sift_up(int id); 即可，这里底层函数会对 id 的合法性进行处理，程序正常运行的情况下，不会产生非法的函数调用的。删除也是直接调用底层函数 bool SongList::remove_from_list(int id)。实现上下移动和删除的时候，歌单这个时候就已经发生了变化了，但歌曲仍然处于播放状态，这样实施列表循环播放的时候可能会出现顺序混乱，于是在这种情况下，我们会用一个私有成员变量 bool modified 来标记歌单是否被修改，同时用 QString backup 标记正在播放歌曲的信息，在切换歌曲信号发出时，我们会调用相应的槽函数来作修正，具体会在播放控件部分讲。

重命名修改信息我们另外定义了窗口类，这个也放在后面讲。

上下移动、重命名和删除时，用户操作都是通过**鼠标右键点击**相应歌曲时，选择菜单项目的方式实现的，菜单的实现运用了 Qt 库中 QMenu 类对象 tableMenu，

作为 MainWidget 的一个私有成员变量，将其关联到 QTableView SongList 的右键操作上。对于每一个项目，我们都通过相应的槽函数将其关联到一个菜单项中：

```
1.    tableMenu.addAction("修改信息",this,SLOT(onModify()));
2.    tableMenu.addAction("删除歌曲",this,SLOT(onRemove()));
3.    tableMenu.addAction("上移",this,SLOT(siftUp()));
4.    tableMenu.addAction("下移",this,SLOT(siftDown()));
5.  //in construcor of MainWidget
```

这些槽函数就相应地调用了底层设计的函数。

【Qt 的信号与槽机制】

到此为止，我们还没有明显地用到 Qt 最经典的信号与槽的机制，但是接下来的操作就是我们用到信号与槽机制的第一个例子，那就是**双击**歌曲列表播放歌曲。

Qt 信号槽机制是 Qt 编程的一个非常重要的特点，程序员可以设置在用户对某一个对象做出一个动作，比如点击按钮，键盘敲下一个字符，拖动了进度条等等，甚至是在不经意中调用了一个函数的情况下，可以发出一个信号 (signal)，信号本质上是一个没有返回值的类成员函数，它没有具体定义，但是可以传递参数，发射信号的时候，使用 emit signal(arg)的方式就可以发射一个信号。当信号被关联了一个槽 (slot) 函数时，槽函数会接收到这个信号，以及这个信号所带的参数，并根据参数执行一系列操作（相当于自动调用函数）。槽函数也是没有返回值的函数，它的形参一定与被关联的信号是类型一致的。

信号与槽的机制本质上就是函数调用，但是可以实现跨类的函数联动，这也是 Qt 比较方便的一个部分。

信号与槽的关联有两种方式，第一是槽函数按照特定的命名规则来命名，可以自动接收信号。比如 void on_playBtn_clicked(); 当按钮“playBtn”被按下的时候，槽函数会自动调用。第二种是手动关联，利用 connect 将信号和槽函数联系在一起，当信号被发射的时候，槽函数就会被自动调用，具体关联语法可以查阅 Qt 手册。

这里**双击歌曲信息**播放歌曲的操作用的是第一种方式，按照特定的命名方式，定义槽函数 void MainWidget::on_songList_doubleClicked(const QModelIndex

&index)，传入的参数是 model 中的某一项 item，函数实现只需要读取 item 中歌曲编号的信息，播放歌曲就行了。具体关键实现如下：

```
1.  void MainWidget::on_songList_doubleClicked(const QModelIndex &index)
2.  {
3.    int row = index.row();
4.    QAbstractItemModel *modessl = ui->songList->model();
5.    QModelIndex indextemp = modessl->index(row,0);
6.    QVariant datatemp = modessl->data(indextemp);
7.    if(modified){
8.       modified = false;
9.       player->stop();
10.      playlist->clear();
11.      loadPlaylist();
12.   }
13.   qDebug()<<"datatemp is "<<datatemp;
14.   playSong(datatemp.toInt());
15. }
```

说明：函数 void loadPlaylist(); 是更新 QMediaPlayList 对象的操作，这涉及到之前提到的歌单被修改后不能准确定位歌曲的情况，类似的操作在其他函数中也被用到了。函数 void playSong(int id); 是根据根据编号播放歌曲的函数。

3.2.2 播放控件相关操作

主界面的下方是歌曲播放相关操作的控件，这些控件都是用来对歌曲播放进行操作的。

对于播放歌曲，我们利用了 Qt 库中的 QMediaPlayer 类和 QMediaPlayList 类。这两个类的动态对象 player 和 playlist 都被定义为 MainWidget 的私有成员变量，在构造函数中被动态分配空间。

其中，playlist 用于加载底层 vector 歌单的信息，具体来说，就是根据歌曲文件名 filename，定位音乐文件，构造实体歌单，这部分操作是在成员函数 void MainWidget::loadPlaylist(); 中定义的，其中调用了 Qt 的库函数 QMediaPlayList::addMedia(const QList<QMediaContent> &items)

关键操作如下：

```
1.  void MainWidget::loadPlaylist(){
2.    for(int i=1;i<=songlist.Size();i++){
```

```
        SongInfo* song = songlist.getSong(i);
        QString file;
        file = song->Filename();
        playlist->addMedia(QUrl::fromLocalFile(file));
    }
}
```

那么，player 的作用就是播放功能了，它可以利用 Qt 库函数 void setPlaylist(QMediaPlaylist *playlist) 来设置播放歌单，在成员函数 void playSong(int id) 中，可以根据歌曲编号 id 定位歌曲并播放，具体定位过程如下：

```
bool MainWidget::playSong(int id){ //id is index of song (from 1 to Size();)
    id--;
    qDebug()<<"now index is "<<id<<" currentindex is "<<playlist->currentIndex();
    if(id==playlist->currentIndex()){
        //do something
    }
    if(id>=songlist.Size()||id<0){
        QMessageBox::warning(this,tr("播放错误"),tr("歌曲定位错误"));
        return false;
    }
    player->stop();
    playlist->setCurrentIndex(id);
    player->play();
    ui->playBtn->setProperty("id","pause");
    ui->playBtn->setStyleSheet("border-image: url(:/new/prefix1/pause.jpg);");
    songlist.playSong(songlist.getSong(id+1)->Filename());
    current_playing = songlist.getSong(id+1)->displayTitle();
    return true;
}
```

说明：由于传入的参数 id 是从 1 开始的，但是 playlist 歌曲编号是从 0 开始的，所以一开始 id—是为了正确获取歌曲编号。

QString current_playing 是 MainWidget 类成员变量，记录当前播放歌曲的标题语言歌手信息，具体字符串显示就是进度条上方的 Label 显示的结果。比如“林俊杰 – 当你”，与文件名有所不同。

这里对 i 的合法性也进行了检查，如果不合法的参数传递，会调用 warning 窗口做出错误提示。warning 窗口在这个项目中多次使用，用于考虑函数调用中可能存在的多种问题，可见程序的鲁棒性还是可以滴~。

我们可以看出，切歌操作也很容易，调用 Qt 库中的 QMediaPlayList::setCurrentIndex(int index) 即可。

这两个函数就是播放音乐最主要的函数，它们在函数中不断地被调用。调用它们的函数主要是一些槽函数，而这些槽函数接受的信号发送者，就是来自主界面下方的控件了。

主界面下方的控件有许多按钮，它们都是 QPushButton 对象，有的被样式表贴了个图，当单击这些按钮就会发射特定的信号，比如播放/暂停、切歌、切换列表播放模式等。还有两个 QHorizontalSlider 对象用于指示歌曲位置和音量，它们被拖动的时候就会发射相应信号，而指示歌曲位置的滑条还会根据歌曲播放进度的变化，在槽函数中改变滑条的位置。槽函数中还会完成相应的操作，调用上面讲到的两个函数，以及在显示上做出一些改变（主要是改变歌单和 Laebl 标签）。

在 MainWidget 类中定义的所有槽函数如下：

```
1.  private slots:
2.     //songlist operations
3.     void siftUp();
4.     void onRemove();
5.     void onModify();
6.     void siftDown();
7.     //operations to call new dialogs or widgtes
8.     void onScoreBtnClicked();
9.     void on_pushButton_clicked();
10.    void on_titleSearchButton_clicked();
11.    void on_singerSearchButton_clicked();
12.    void back(){this->show();}
13.    void on_pushButton_2_clicked();
14.    void on_pushButton_3_clicked();
15.    void score_end(ScoreDialog* );
16.    void search_end(SearchDialog*, int);
17.    void add_end(AddDialog*,bool,QString,QString,QString);
18.    void modify_end(int,QString,QString);
19.    void end_thank();
20.    //operations related to media player
21.    void on_songList_doubleClicked(const QModelIndex &index);
22.    void ChangeSlider(qint64 val);
23.    void updateDuration(qint64 duration);
24.    void on_volumnSlider_sliderMoved(int value);
25.    void on_SongSlider_valueChanged(int position);
26.    void on_SongSlider_sliderReleased();
27.    void on_playBtn_clicked();
28.    void on_cirType_clicked();
```

```
29.    void changeSong(int position);
30.    void on_nextBtn_clicked();
31.    void on_lastBtn_clicked();
```

最后一组函数是对歌曲播放状态的槽函数定义，其中包括了一系列歌曲和实体歌单操作，很多都是由控件自动关联的。关于各个控件的具体含义，我就不在报告中详细阐明了，有兴趣的朋友可以查看我的源代码。

3.3 附加窗口设计——功能多样化

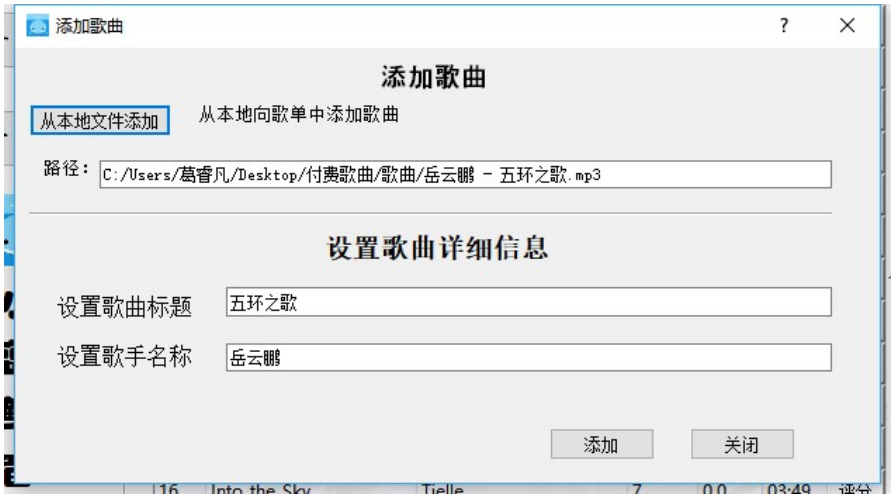
本项目定义了 7 个与窗口有关的类，它们都继承于 QWidget 和 QDialog 类，除了主窗口类 MainWidget 外，还有许多对话框窗口的定义，它们往往是对应于不同的播放器功能而设计的。

在 MainWidget 槽函数中，有许多槽函数（第二组）是对应于与其他窗口交互的，在下面的报告中，我将对于我所设计的所有窗口作个简要介绍，同时说明它们与主窗口是怎么联系在一起的。

3.3.1 添加歌曲窗口

类名称：AddDialog（继承于 QDialog）

添加歌曲窗口是在用户点击主窗口按钮“添加歌曲”时动态创建，不属于主窗口的一个私有成员。在添加歌曲窗口中，用户可以选择本地歌曲文件，并设置歌曲标题与歌手信息，点击“确认”按钮后，发送信号 add_end 至主窗口，传递歌曲文件位置和歌曲信息等参数，并隐藏该窗口。



用户在本地文件添加之前，无法点击“添加”按钮，歌曲的详细信息界面也不会显示，用户输入的信息不会在 AddDialog 中检查合法性，此窗口的作用仅为传递信息，所有鲁棒性检查的工作全部在主窗口中完成。

涉及 AddDialog 的创建、信号、槽关键代码如下所示：

```
1.  void MainWidget::on_pushButton_clicked()
2.  {
3.      AddDialog* adddia = new AddDialog(this);
4.      connect(adddia,SIGNAL(add_end(AddDialog*,bool,QString,QString,QString)),
    this,SLOT(add_end(AddDialog*,bool,QString,QString,QString)));
5.      adddia->setWindowTitle("添加歌曲");
6.      adddia->show();
7.  }
8.  void MainWidget::add_end(AddDialog* adddia, bool valid, QString path, QString tit
    le, QString singer){
9.      if(!valid){
10.         adddia->hide();
11.         return;
12.     }
13.     // do validity check
14.     if(!songlist.add_to_list(path,title,singer)){
15.         QMessageBox::warning(this,tr("添加失败"),tr("添加失败，歌单内已经有相应
    的歌曲了，尝试换个标题吧。"),QMessageBox::Abort);
16.         return;
17.     }
18.     songlist.save_to_file();
19.     QString file = //.....
20.     playlist->addMedia(QUrl::fromLocalFile(file));
21.     playSong(songlist.Size());
22.     adddia->hide();
23.     return;
```

```
24. }
```

3.3.2 查找歌曲/歌手窗口

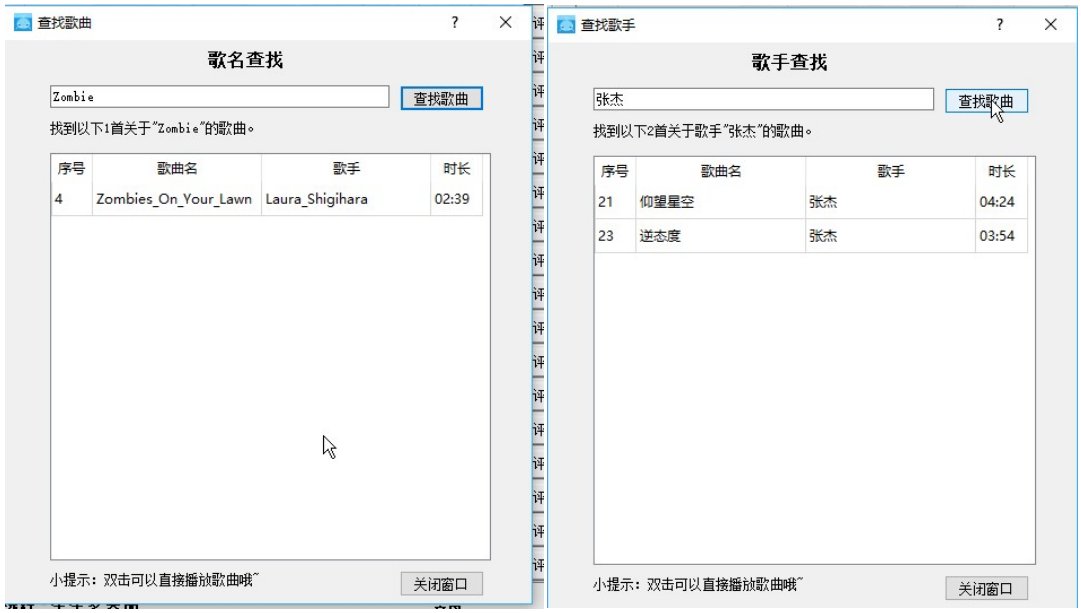
类名称：SearchDialog（继承于 QDialog）

查找歌曲或歌手窗口是在用户点击主窗口按钮“查找歌曲”或“查找歌手”时动态创建，不属于主窗口的一个私有成员。用户点击“查找歌曲”和点击“查找歌手”时，会向 SearchDialog 构造函数发送查找模式类型（1 代表歌名查找、2 代表歌手查找，取决于用户操作），构造函数根据查找类型参数类型准备标签和相应的底层查找函数，另外歌单对象也被作为一个私有成员变量，用于查找歌曲。用户输入关键词后，在歌单中进行逐个字符串模式匹配，满足条件的会显示在表格中，双击表格中的歌曲可以直接播放，并关闭搜索窗口。当然，程序也对输入关键字的合法性（主要是不能查找到信息时的提示）进行检查。

涉及 SearchDialog 的创建、信号、槽关键代码如下所示：

```
1.  void MainWidget::on_titleSearchButton_clicked()
2.  {
3.      SearchDialog* searchdia = new SearchDialog(this,&songlist,1);
4.      connect(searchdia,SIGNAL(search_end(SearchDialog*,int)),this,
    SLOT(search_end(SearchDialog*,int)));
5.      searchdia->setWindowTitle("查找歌曲");
6.      searchdia->show();
7.  }
8.
9.  void MainWidget::on_singerSearchButton_clicked()
10. {
11.     SearchDialog* searchdia = new SearchDialog(this,&songlist,2);
12.     connect(searchdia,SIGNAL(search_end(SearchDialog*,int)),this,
    SLOT(search_end(SearchDialog*,int)));
13.     searchdia->setWindowTitle("查找歌手");
14.     searchdia->show();
15. }
16. void MainWidget::search_end(SearchDialog* sd, int id){
17.     qDebug()<<"called search_end";
18.     if(id == -1) return;
19.     else{
20.         playSong(id);
21.     }
```

```
22.     sd->hide();
23.     return;
24. }
```



3.3.3 修改信息窗口

类名称 ModifyDialog（继承于 QDialog）

在主窗口界面鼠标右击某一首歌，菜单选择“修改信息”时，修改信息窗口被动态创建，并将歌曲信息作为参数传入 ModifyDialog 的构造函数中。

与添加歌曲相同，用户在修改信息中输入的信息会直接发送给主窗口，且在主窗口检查输入合法性并提示。用户只能对正在不在播放的歌曲修改信息，也只能修改歌手和歌曲标题的信息。

涉及 ModifyDialog 的创建、信号、槽关键代码如下所示：

```
1.  void MainWidget::onModify(){
2.      int row = ui->songList->currentIndex().row();
3.      QAbstractItemModel *modessl = ui->songList->model();
4.      QModelIndex indextemp = modessl->index(row,0);
5.      QVariant datatemp = modessl->data(indextemp);
6.      if(songlist.getSong(datatemp.toInt())->displayTitle()==current_playing){
7.          QMessageBox::warning(this,tr("修改失败"),tr("歌曲正在播放，无法修改信息。
    "));
8.          return;
9.      }
10.     backup = current_playing;
```

```
    modified = true;
    ModifyDialog* modifydia = new ModifyDialog(this,songlist.getSong(datatemp.toInt()));
    modifydia->setWindowTitle("修改歌曲信息");
    connect(modifydia,SIGNAL(modify_end(int, QString, QString)),this,SLOT(modify_end(int,QString,QString)));
    modifydia->show();
}

void MainWidget::modify_end(int id ,QString title ,QString singer){
    if(songlist.rename(id,title,singer)){
        QMessageBox::warning(this,tr("修改信息"),tr("修改成功！"));
        showTable();
        return;
    }
    else{
        QMessageBox::warning(this,tr("修改信息"),tr("修改失败，请检查关键字是否冲突。"));
        return;
    }
}
```

3.3.4 评分系统窗口

类名称 ScoreDialog（继承于 QDialog）

用户在歌单列表点击相应的“评分”按钮，可以对歌曲评分，此时 ScoreDialog 对象将被动态创建。构造函数将歌曲的信息作为参数传入，获取用户的评分数值后，调用底层的歌曲评分函数进行评分，确认后发送评分信号给主窗口，主窗口

根据评分信号传来的参数更新分数和歌单信息。

与其他窗口不同，由于构造函数传入了歌曲指针，因此在本窗口类中，底层函数就已经被调用了。

涉及 ScoreDialog 的创建、信号、槽关键代码如下所示：

```
1.  void MainWidget::onScoreBtnClicked(){
2.      QPushButton *button = (QPushButton *)sender();
3.      int id = button->property("id").toInt();
4.      ScoreDialog* scoredia = new ScoreDialog(this,songlist.getSong(id));
5.      connect(scoredia,SIGNAL(score_end(ScoreDialog* )),this,SLOT(score_end(Score
    Dialog* )));
6.      scoredia->setWindowTitle("评分系统");
7.      scoredia->show();
8.  }
9.  void ScoreDialog::on_okbtn_clicked()
10. {
11.     song->score_song(score);
12.     this->hide();
13.     QMessageBox::warning(this,tr("歌曲评分"),tr("成功提交！"));
14.     emit score_end(this);
15. }
16. void MainWidget::score_end(ScoreDialog* sd){
17.     sd->hide();
18.     showTable();
19. }
```

3.3.5 缩略窗口

类名称：MinimizeWidget（继承于 QWidget）

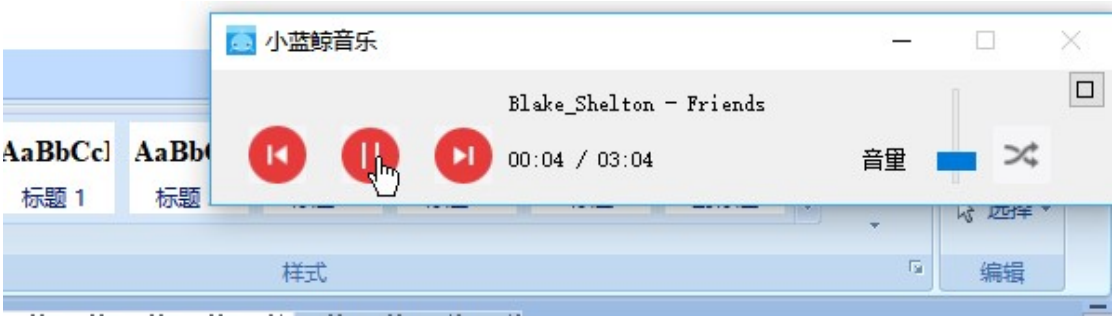
缩略窗口不仅继承的父类与其他窗口不同，而且他的动态对象是在主窗口的构造函数中一开始就被创建，且作为主窗口的一个私有成员变量。缩略窗口与主窗口之间，其实无时无刻不在互相发送信号，以使得空间和显示之间能够同步。

具体说来，在 MinimizeWidget 中点击控件，会发送信号到主窗口执行相应的操作，在主窗口播放歌曲的状态发生变化时，也会向 MinimizeWidget 发送信号，改变缩略窗口的显示，以此来获得主从窗口的显示和操作一致性。

主窗口与缩略窗口之间信号与槽的关联在主窗口的构造函数中被定义，具体如下所示：

```
1.  connect(this,SIGNAL(volumnChanged(int)),miniwidget,SLOT(changeVolumn(int)));
2.  connect(this,SIGNAL(changeLab(QString)),miniwidget,SLOT(changeLab(QString)));
3.  connect(this,SIGNAL(playBtn_clicked()),miniwidget,SLOT(playBtn()));
4.  connect(this,SIGNAL(playBtn_pause()),miniwidget,SLOT(playBtn_pause()));
5.  connect(this,SIGNAL(cirType_clicked()),miniwidget,SLOT(cirType()));
6.  connect(this,SIGNAL(setTitle(QString)),miniwidget,SLOT(changeTitle(QString)));
7.  connect(miniwidget,SIGNAL(playBtn_clicked()),this,SLOT(on_playBtn_clicked()));
8.  connect(miniwidget,SIGNAL(LastBtn_clicked()),this,SLOT(on_lastBtn_clicked()));
9.  connect(miniwidget,SIGNAL(NextBtn_clicked()),this,SLOT(on_nextBtn_clicked()));
10. connect(miniwidget,SIGNAL(volumnMoved(int)),this,SLOT(on_volumnSlider_sliderMo
    ved(int)));
11. connect(miniwidget,SIGNAL(cirType_clicked()),this,SLOT(on_cirType_clicked()));
12. connect(miniwidget,SIGNAL(back()),this,SLOT(back()));
```

除了 back 信号是为了切回主窗口外，所有的信号和槽都是为了使得主窗口和缩略窗口同步显示数据和操作，避免不必要的 bug。



3.3.6 其他窗口

主要就是鸣谢窗口，类名称 ThankDialog（继承于 QDialog）

四、代码框架

4.1 项目编写环境

本项目编写环境是 Windows10 下 Qt Creator4.9.8 集成环境。

4.2 项目文件目录

在提交的项目文件 MyMediaPlayer 中，包含以下文件夹/文件

文件夹 codes：所有的源代码、资源文件

文件夹 makefile：Qt 构建项目时自动生成的文件

文件夹 music：所有音乐文件的拷贝

文件夹 sysfile：存储歌单信息的本地文件

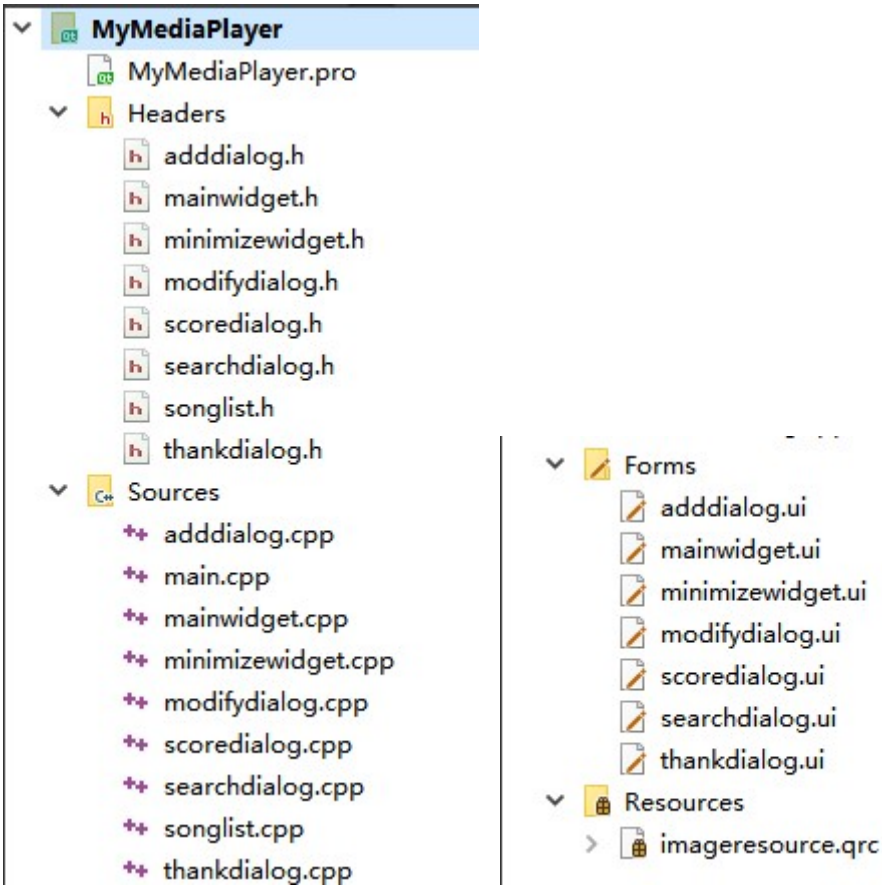
可执行文件 MyMediaPlayer.exe：程序可执行文件入口

README.md：用户手册和鸣谢说明

GitHub 项目开源地址：https://github.com/EricPro-NJU/MyMusicPlayer

4.3 项目模块划分

本项目的模块划分如下图所示：



其中：main.cpp 为主函数入口，songlist.h/songlist.cpp 为底层歌单实现，其余的七个类均为设计师窗口类，拥有.h 头文件，.cpp 实现文件和.ui 设计师文件。

imageresource 中存放了该项目所用到了（和没用到的）所有图片资源，这些资源可以在/codes/resources 中查看到。

五、后续思考

由于第一次编写 GUI 程序，有一点点生疏，所以代码和程序都比较简陋，回顾整个项目构建的过程，我们可以从项目编写过程中出现的问题和后续项目的扩写提出见解。

5.1 编写项目中遇到的问题和反思

问题：在项目中，我们在设计其他窗口类（比如：AddDialog）时，我们采取了动态分配空间的方式（用 new），但是没有任意一个地方使用了 delete 去释放，这是否会构成内存泄漏的问题？

答：确实有这样的问题。在之前的设计中，我们的确有在相应的主窗口槽函数中，运用其传来的窗口对象地址参数，去使用 delete 释放空间，但是却出现了 Segmentation Fault，没有 Qt 编程经验的我并没有找到答案，只是猜测发送信号过后，动态窗口中的函数还没有调用结束，导致函数从信号返回的时候，该窗口已经不存在了，所以出现的错误。目前我还没有比较好的解决方法，只是去掉了 delete 操作避免应用程序陷入异常。

问题：在实现歌单循环播放的时候，在接触到 QMediaPlayList 类之前，我本来想过直接通过操作 QMediaPlayer 类对象来实现切歌操作，但是出现了 Segmentation Fault，这个问题是如何产生的呢？

答：之前的设计中，我们是通过追踪歌曲位置 position 和 duration 进行比较，当 position==duration 的时候进行切歌操作，此状态使用了信号关联槽的方式实现，但是出现异常。实际 Debug 后发现是多次调用了这个函数导致的指针错误。后来发现，QMediaPlayer 的行为其实非常的诡异：在 play 之前，不能正确获取歌曲信息，这样才导致错误的发生。

在翻烂帮助文档，以及查遍 CSDN 社区大量博客之后，我终于发现 QMediaPlayList 类这个好东西，我们的问题也就得到了解决。

由此我们得出结论：发现问题要想方设法自己解决问题！用好各种帮助文档和网上资源。特别是对于我这个 GUI 新手。

5.2 代码的后续扩写与处理

这个代码其实有很多可以后续加工的地方，我想了几个没有来得及实现的点，使得其可以更加接近实际应用中的音乐播放器，比如：

- 背景切换，根据实时播放的歌曲，读取其专辑图，切换播放器背景。
- 多重歌单，可以实现新建歌单，往歌单里添加歌曲的操作。

- 滚动歌词，根据歌词文件代码，实现歌词的滚动播放，甚至桌面歌词。
- ……

另外，由于本项目底层实现用的 Qt 特性不是很多，因此把它改造成控制台版本也是不用修改太多底层实现的。

六、后记

这是高级程序设计最后一次课程设计了，综合一学期的课程设计体验，我发现我的编程实力与一些 dalao 还是有所差距的。但是熟能生巧，我相信多次的练习一定会给我带来许多不一样的收获。

最后，真诚感谢高级程序设计课程组所有助教和老师的悉心指导！

181840070 葛睿芃

2019/12/31