山西大学

课程设计报告

题　　目：基于 QT的音乐播放器

系　　别： 软件学院　班　　级： 1522 班

姓　名：贾梦洁　　学　　号：

同组组员：

设计时间：　2017　年 12 月 22 日----2017　年 12 月 30 日

公司名称：　上海杰普软件科技有限公司

目　录

第一章 引言

计算机技术的飞速发展大大提高了人们的工作效率，　尤其是互联网技术更是很大程度上丰富和方便了人们的生活。　近些年来，人们的生活水平也在不断提升，　在物质丰富的条件下，　人们开始在工作之余关注娱乐，期望在其它方面释放工作压力，　同时培养自己的兴趣爱好，随之而来的是人们对多媒体应用的关注，　许多电影播放平台、　音乐播放软件等逐渐深入人们的生活，并慢慢地成为人们生活重要组成部分。

目前，互联网上已经拥有大量的音乐播放软件，　这些软件是各个软件供应商的商品关键组成部分，　当前其实现技术较复杂，　在功能方面相当完备且强大，如何简单、高效、方便地设计实现一款小巧美观的音乐播放器软件成为人们关注的热点。　基于该问题，本文设计并实现了基于 Qt 的音乐播放软件，该软件能够便捷、高效地为用户展示音乐播放界面，　方便的进行歌曲播放和控制功能。　本文的工作分为软件界面设计和软件功能实现两部分，　其中，软件界面设计工作主要包括用户界面设计实现；各个功能模块实现工作主要包括后台程序编码设计模块化完成设计等内容。

该音乐播放软件以　Qt 开发平台中实现歌曲播放的相应功能为基本框架设计，通过　C++语言编程实现各个功能函数，软件设计方面采用模块化的软件设计思想实现，　具有友好的用户交互界面和高承载能力的运行稳定性。

第二章　软件分析与设计

2.1　软件需求分析

在设计实现音乐播放器软件的同时，　可以对目前存在的不同类型的音乐播放器进行广泛深入的研究，查看软件可能需要的需求内容。

因此，需要对具体问题进行具体分析，　深入挖掘其需要实现的系统功能，以方便后面对软件构架的设计工作。需求分析的过程，是开发人员对音乐播放器工作过程的认识与熟悉的过程，　也是对软件内部工作流程进行计算机建模的过程，　最终目的是通过需求分析了解用户需求实现的功能，　根据用户提出的需求设计好系统的概念模型，　对用户提出的需求进行计算机方法的描述，并建立相应配套的需求分析文档，设计好系统的具体实现方案。

在设计实现基于　Qt 的音乐播放器的同时，考虑到音乐播放器的实际工作环境，可以确定的是该软件应当拥有以下几个方面的特性：

（1）基本功能

随着电脑终端的扩大化，　PC 机上各种软件也不计其数。为了使用户体验及软件质量都达到一定的完备性，　我们需要保证一定的基础功能。基于人性化角度，开发设计基础功能，使用户可以流畅使用软件。在进行功能划分的过程当中，可以采用模块化的功能设计思想，对功能的划分尽可能的细致，做到不遗漏。例如，应当有播放模式的选择，音乐列表的选择，音量高低的调试等。

（2）稳定性

基于 QT的音乐播放器是一款娱乐软件，更需要保证软件的稳定性，以使用户体验达到最高。若播放器不稳定，则用户体验这一指标则会大幅下降。

（3）联通性

移动设备日益增长，微博、微信等交流平台在市场中也占据重要地位。为迎合这一现象， QT 音乐播放器必须进行一定的互联网联通性。可以进行音乐分享、转发评论等。这样可以实现软件之间的交流结合，有利于软件的推广。 同时也有利于通过大数据分析，得到听众最喜爱听的歌曲，符合大众口味。

2.2 开发环境

操作系统： Windows 10

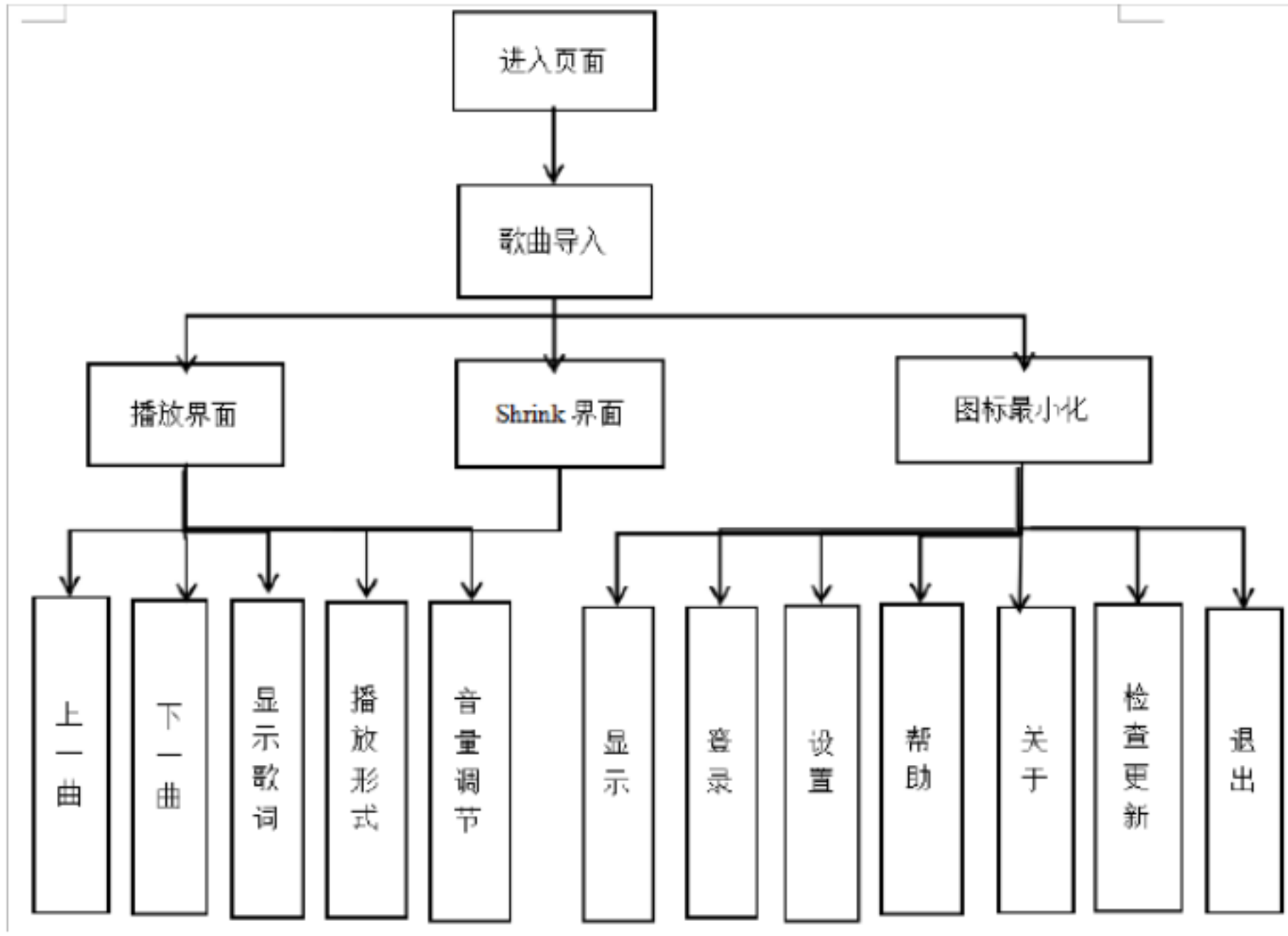
开发平台： Qt

使用语言： C++

2.3 软件概要设计

在软件概要设计阶段， 是基于整个系统需要实现的功能， 对物业管理信息系统的整体架构进行科学合理的设计， 使之有一定的规律可以遵循，不进行盲目的设计工作， 这对于后期的程序编码和系统的来说意义重大。 在对音乐播放器软件进行软件需求分析之后， 针对需求

分析的结果，对系统进行了整体的架构设计。



设计实现的音乐播放器软件，　从软件的架构来看，　主要包括两个功能部分：软件的歌曲播放部分和图标最小化功能实现部分。其中，歌曲播放部分主要实现该软件的主要功能，　即选择歌曲和播放歌曲；　另一部分则是最小化后软件的实现，主要包括显示、登录、设置、检查更新、退出。以上是对该音乐播放器软件的整体工作架构进行的设计，该步骤是后续进一步进行系统功能实现的基础。

第三章　软件功能实现

在设计实现基于　Qt 的音乐播放器软件的同时，　充分考虑了　Qt 的语言特性和开发环境，发挥了其在用户界面应用程序开发当中的优势，对音乐播放器软件进行了开发。　根据前文对该音乐播放器软件的

需求分析， 以及对软件的设计分析结果， 结合实际使用当中可能使用到的软件功能， 并采用了软件工程中模块化的开发思想， 完成了该音乐播放器软件的开发， 下面分别对软件中重要的功能模块的实现方法及实现效果进行详细的介绍和分析。

3.1 总体架构

通过在需求阶段对系统的总体功能的分析， 我们得到了这个音乐播放器软件的总体功能结构，它应包括四大基本功能模块。

（1）选择歌曲文件模块：用户可以方便的查询环境当中存在的歌曲列表信息，并找到满意歌曲的位置和具体的歌曲内容。

（2）歌曲播放模块：用户根据所选择的具体歌曲让该软件对歌曲进行播放，并且可以实时查看歌曲的播放状态。

（3）播放状态控制模块：用户可以查看成自己在歌曲播放过程中进度情况，并可以随时更改歌曲的播放进度信息。

（4）第二界面模块： 用户可以进入第二个更为简洁的界面进行播放、暂停、切换等操作。

3.2 软件功能流程

整个软件所设计的工作流程如下：

（1）开始。进入 Qt 界面开始运行软件。

（2）播放歌曲。该部分是音乐播放器软件最主要的功能部分，播放歌曲的功能实现需要相对复杂的函数调用来实现， 在软件开发的过程

当中有具体的实现代码。

（3）播放设置。该部分的实现是进一步提高用户的使用体验，方便用户根据其自身的需求对所播放的歌曲进行进度控制，　　可以暂停歌曲的播放，也可以继续歌曲的播放功能，同时也可以进行音量的调控，以及对歌曲播放的顺序进行调整，比如顺序播放、单曲循环等。

（4）第二界面。点击 shrink 按钮可跳转至第二界面，进行播放切换歌曲等操作，也可以进行最小化。

（5）结束。点击关闭软件，结束软件运行。

3.3 具体功能实现

本人完成了本次音乐播放器中的歌曲导入模块以及播放歌曲的初步工作，具体的实现过程如下。

3.3.1 歌曲导入

（1）.h 文件

歌曲导入部分的代码函数名称以及相应的函数声明完成在 settingwiget.h　　的头文件下，先对命名空间进行前向声明，对类内的对象和方法进行声明，具体代码如下：

```
#ifndef SETTINGWIGET_H
#define SETTINGWIGET_H

#include <QWidget>

namespace Ui {
class settingwiget;
```

```
}

class settingwiget : public QWidget
{
    Q_OBJECT

public:
    explicit settingwiget(QWidget *parent = 0);
    ~    settingwiget         ();

    void initSystemSetting(void);

    bool writeInit(QString path, QString user_key, QString
user_value);
    bool readInit(QString path, QString user_key, QString
&user_value);

private slots:
    void on_pb_save_clicked();

    void on_pb_cancle_clicked();

    void on_tb_lyricsPath_clicked();

    void on_tb_songsPath_clicked();

    void on_pb_save_destroyed();

private:
    Ui::settingwiget *ui;
};

#endif // SETTINGWIGET_H
```

（2）.cpp 文件

配置文件写入和读取的方法完成在 settingwiget.cpp 文件内，代码如下：

```
settingwiget::settingwiget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::settingwiget)
```

```
{
    ui->setupUi(this);

    SongsFile::m_songPath.clear();
    LyricsFile::m_lyricPath.clear();
}

settingwiget::~          settingwiget        ()
{
    delete ui;
}

bool    settingwiget::writeInit(QString                    path,    QString    user_key,
QString user_value)
{
    if(path.isEmpty() || user_key.isEmpty())
    {
        return false;
    }
    else
    {

        //          创建配置文件操作对象
        QSettings *config = new QSettings(path,
QSettings::IniFormat);


        //          将信息写入配置文件
        config->beginGroup("config");
        config->setValue(user_key, user_value);
        config->endGroup();

        return true;
    }
}

bool    settingwiget::readInit(QString                    path,    QString    user_key,
QString &user_value)
{
    user_value = QString("");
    if(path.isEmpty() || user_key.isEmpty())
    {
        return false;
    }
```

```
    else
    {
        //          创建配置文件操作对象
        QSettings *config = new QSettings(path,
QSettings::IniFormat);

        //          读取用户配置信息
        user_value = config->value(QString("config/") +
user_key).toString();

        return true;
    }
}
```

界面设计中按钮的具体功能也对应在 settingwiget.cpp 文件内，代码如下：

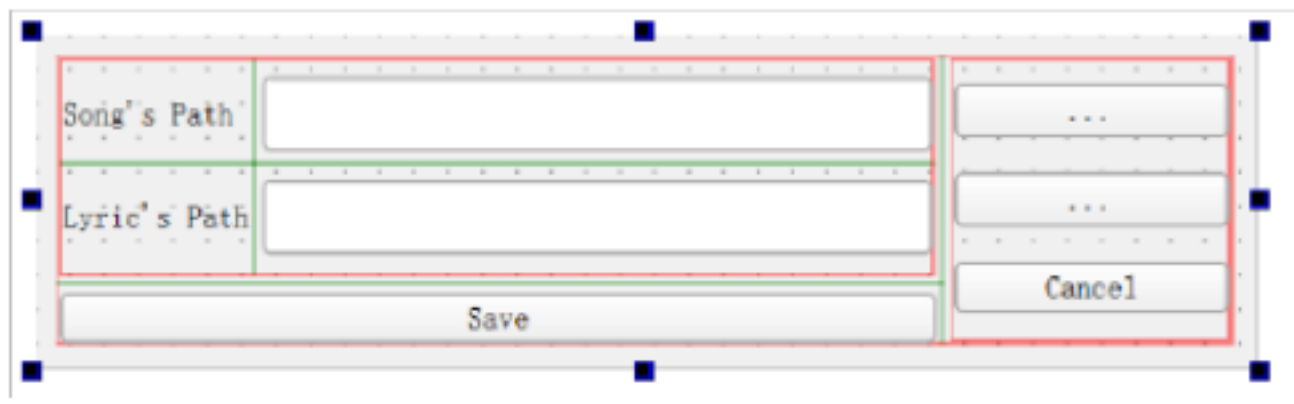
```
void settingwiget::on_tb_songsPath_clicked()
{
    SongsFile::m_songPath =
QFileDialog::getExistingDirectory();
    ui->le_songsPath->setText(SongsFile::m_songPath);
}

void settingwiget::on_tb_lyricsPath_clicked()
{
    LyricsFile::m_lyricPath =
QFileDialog::getExistingDirectory();
    ui->le_lyricsPath->setText(LyricsFile::m_lyricPath);
}

void settingwiget::on_pb_save_clicked()
{
    writeInit(QString("../user.ini"), "SONGPATH",
SongsFile::m_songPath);
    writeInit(QString("../user.ini"), "LYRICPATH",
LyricsFile::m_lyricPath);

    initSystemSetting();
    this->hide();
```

```
}

void settingwiget::on_pb_cancle_clicked()
{
    this->hide();
}

void settingwiget::initSystemSetting(void)
{
    readInit(QString("../user.ini"), "SONGPATH",
SongsFile::m_songPath);
    readInit(QString("../user.ini"), "LYRICPATH",
LyricsFile::m_lyricPath);

    ui->le_songsPath->setText(SongsFile::m_songPath);
    ui->le_lyricsPath->setText(LyricsFile::m_lyricPath);

}

void settingwiget::on_pb_save_destroyed()
{}
```

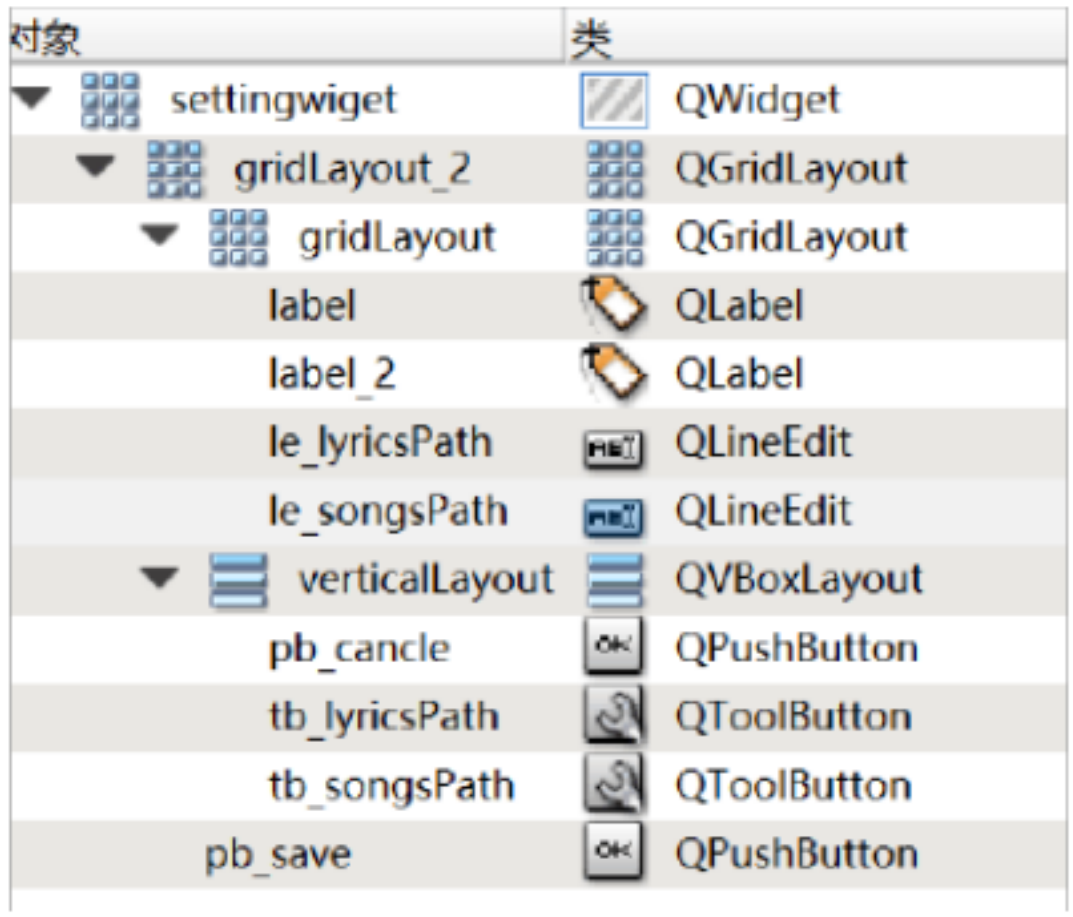
其中， cpp 文件内的头文件为：

```
#include "settingwiget.h"
#include "ui_settingwiget.h"

#include <QFileDialog>
#include <QSettings>
#include <QDebug>
#include "musicplayer.h"
#include "musicwiget.h"
#include "songsfile.h"
#include "lyricsfile.h"
```

导入歌曲的页面设计完成在界面设计的 settingwiget.ui 文件下，如下图所示：



具体的对象为：



3.3.2 歌曲播放的部分功能

（1）.h 文件

musicplayer.h 的头文件下，采用枚举类型对播放模式进行声明，写入信号和槽方便传值，对歌曲播放逻辑、歌曲列表获取逻辑、歌曲歌词获取逻辑的对象、方法进行声明，具体代码如下：

```
#ifndef MUSICPLAYER_H
#define MUSICPLAYER_H

#include <QObject>

#include <QMediaPlayer>
#include <QMediaPlaylist>
#include <QList>

#include "songsfile.h"
#include "lyricsfile.h"

#define TIME_MS_DURATION 1000
#define INIT_SYSTEM_VOLUME 50


enum PlaybackMode            //                播放模式
```

```
{
    CURRENT_ITEM_ONCE = 0,          //                  单曲播放
    CURRENT_ITEM_IN_LOOP = 1,       //                  单曲循环
    SEQUENTIAL = 2,                 //                  顺序播放
    LOOP = 3,                       //                  列表循环
    RANDOM = 4                      //                  随机播放
};

class MusicPlayer : public QObject
{
    Q_OBJECT

signals:
    void signalPositionChanged(QString updateTime);
    void signalDurationChanged(qint64 duration);

public slots:
    void slotPositionChanged(qint64 position);
    void slotDurationChanged(qint64 duration);

public:
    explicit MusicPlayer(QObject *parent = 0);

    const qint64 getTotalPlayerTime(void) const;
    void setTotalPlayerTime(const qint64 time);

    const qint64 getCurrentPlayerTime(void) const;
    void setCurrentPlayerTime(const qint64 time);


    ///         歌曲播放逻辑
    QMediaPlayer &currentMediaPlayer(void);

    ///         歌曲列表获取逻辑
    QMediaPlaylist &musicPlayList(void);
    QStringList &songsNameList(void);

    ///         歌曲歌词获取逻辑
```

```
    void getCurrentSongLyric(void);
    QList<qint64>&lyricIndexList(void);
    QStringList &lyricContentList(void);

private:
    ///         歌曲播放逻辑
    qint64 m_totalPlayerTime;
    qint64 m_currentPlayerTime;
    QMediaPlayer m_player;

    ///         歌曲列表获取逻辑
    SongsFile *m_songsFile;
    QMediaPlaylist m_musicPlayList;
    QStringList m_songsNameList;

    ///         歌曲歌词获取逻辑
    QList<qint64> m_lyricIndexList;
    QStringList m_lyricContentList;
};

#endif // MUSICPLAYER_H
```

（2）.cpp 文件

musicplayer.cpp 的头文件如下：

```
#include "musicplayer.h"

#include <QTime>
#include "musicwiget.h"
```

同时在 cpp 文件内写入了信号与槽传值连接的方法，代码如下：

```
MusicPlayer::MusicPlayer(QObject *parent) : QObject(parent)
{
    m_songsFile = new SongsFile(this);

m_songsFile->initSongsListAndSongsNameList(m_musicPlayLis
t, m_songsNameList);
```

```
    m_player.setPlaylist(&m_musicPlayList);
    m_player.setVolume(INIT_SYSTEM_VOLUME);
    connect(&m_player, SIGNAL(durationChanged(qint64)),
            this, SLOT(slotDurationChanged(qint64)));
    connect(&m_player, SIGNAL(positionChanged(qint64)),
            this, SLOT(slotPositionChanged(qint64)));
}
```

得到 siganl 后相应执行的槽函数（ position ），具体代码如下：

```
void MusicPlayer::slotPositionChanged(qint64 position)
{
    m_currentPlayerTime = position / TIME_MS_DURATION;

  //    歌曲进度条显示

    QTime currentTime((m_currentPlayerTime/3600)%60,
(m_currentPlayerTime/60)%60,
                      m_currentPlayerTime%60,
(m_currentPlayerTime*1000)%1000);
    QTime totalTime((m_totalPlayerTime/3600)%60,
(m_totalPlayerTime/60)%60,
                    m_totalPlayerTime%60,
(m_totalPlayerTime*1000)%1000);

    QString updateTime = currentTime.toString("mm:ss")
                       + "/"+ totalTime.toString("mm:ss");

    emit signalPositionChanged(updateTime);
}

void MusicPlayer::slotDurationChanged(qint64 duration)
{
    m_totalPlayerTime = duration / TIME_MS_DURATION;
//TIME_MS_DURATION 1000
    emit signalDurationChanged(m_totalPlayerTime);
}
```

以及对播放时间、播放列表等的声明，代码如下：

```
const qint64 MusicPlayer::getTotalPlayerTime(void) const
{
    return m_totalPlayerTime / TIME_MS_DURATION;
//INIT_SYSTEM_VOLUME 50
```

```
}
void MusicPlayer::setTotalPlayerTime(const qint64 time)
{
    m_totalPlayerTime = time;
}

const qint64 MusicPlayer::getCurrentPlayerTime(void) const
{
    return m_currentPlayerTime;
}
void MusicPlayer::setCurrentPlayerTime(const qint64 time)
{
    m_currentPlayerTime = time;
}

QMediaPlayer &MusicPlayer::currentMediaPlayer(void)
{
    return m_player;
}

QMediaPlaylist &MusicPlayer::musicPlayList(void)
{
    return m_musicPlayList;
}

QStringList &MusicPlayer::songsNameList(void)
{
    return m_songsNameList;
}

void MusicPlayer::getCurrentSongLyric(void)
{
    LyricsFile lyricsFile;
    m_lyricIndexList.clear();
    m_lyricContentList.clear();
    lyricsFile.getCurrentSongLyric(m_lyricIndexList,
m_lyricContentList);
}

QList<qint64>&MusicPlayer::lyricIndexList(void)//                    歌词索引列
表
{
    return m_lyricIndexList;
```

```
}

QStringList &MusicPlayer::lyricContentList(void)//                    歌词内容
列表
{
    return m_lyricContentList;
}
```

第四章　软件测试

软件实现截图如下：

（1）主界面

（2）第二界面

（3）最小化

第五章 总结

本次所设计的基于 Qt 的音乐播放器软件，在 Windows系统环境下可以成功的运行， 对于预期的功能目标已经基本实现， 其中包括歌曲的选择，歌曲打开，歌曲播放，歌曲暂停，显示歌曲播放的进度条等功能。该软件的开发是在 Qt Creator 的开发环境下完成的程序编写工作，该开发平台简单方便，操作快捷，可以直接在 Windows系统平台下安装运行， 完成代码的编写后， 只需要直接点击编译和运行即可使音乐播放器开始工作， 按用户的选择进行歌曲播放等功能， 具有一定的实用性。

通过本次毕业设计 ,我终于明白了“ 看一万行代码，不如动手写一行代码 ”这一句真理，对于工科类的学生来说，除了加强对对书本里的理论知识的学习之外， 更重要的是培养自己实践动手的能力。 这次毕业设计，让我以后面对困难时变得更有耐心，对我来说，这一精力都是在以后的生活和学习中的很宝贵的财富， 极大的影响我以后的成长和发展道路。

开发时间限制， 我们小组实现了系统的基本功能， 软件可以实现基本功能，但是界面尚不够美观，系统也不够完善，下一步，我们会

继续改进系统。 本次设计以及系统的实现让我们认识到了以前很多没有注意到的细节问题，让我学到了不少的新知识。