# 第八周实训报告

这周主要完成了对代码的一些重构，并且制作了一个音乐播放器。

音乐播放器主要能够实现以下功能：

1、添加歌曲到目录中，对目录中的歌曲进行去除以及对目录中的歌曲进行清空。

2、可以支持用户自由调节音量大小，音乐播放的进度。

3、用户可以根据自己的需求选择播放的模式：随机模式，单曲循环模式、列表循环模式。

以下介绍如何实现该音乐播放器。

## 音乐播放器的实现

### musicPlayer.h文件

头文件中对每个成员函数和数据成员的功能都进行了描述，这里不再重复赘述。

重要函数的实现会在下面的cpp文件中进行讲解。

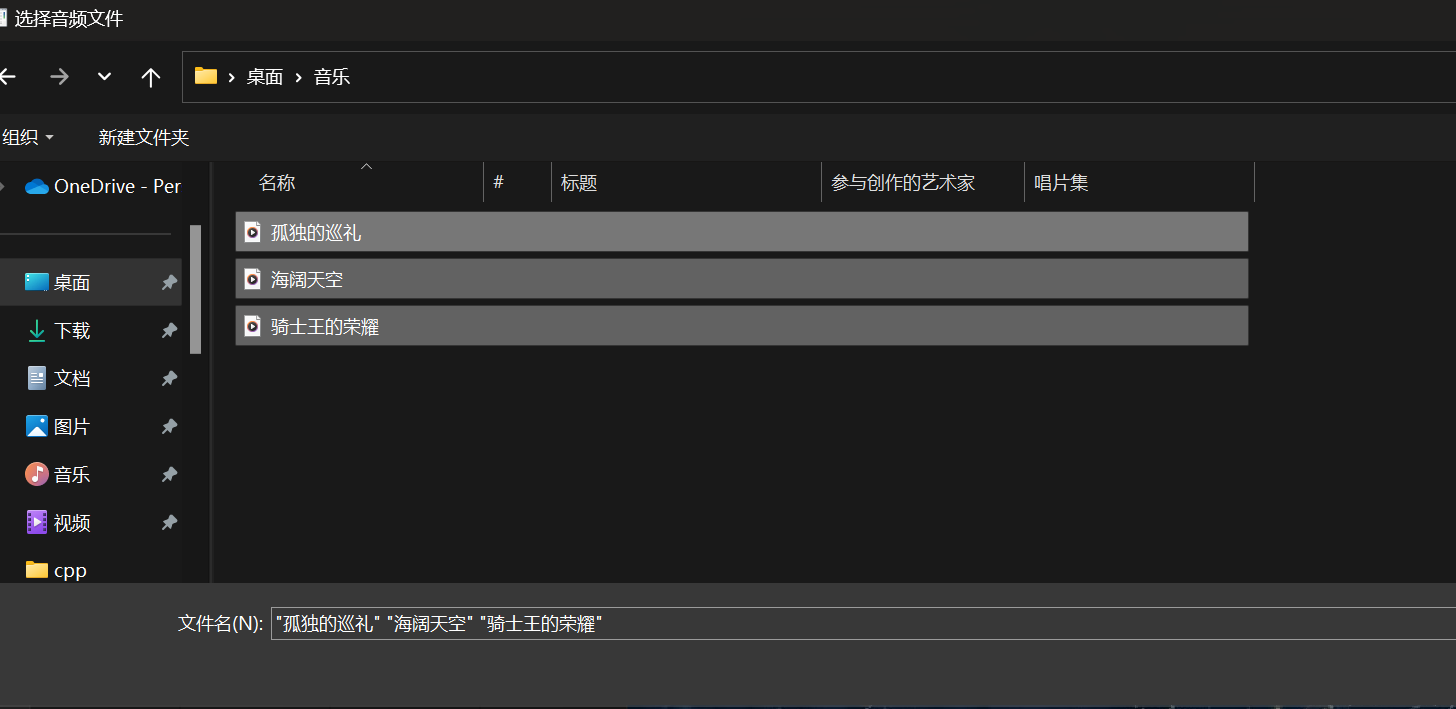
#ifndef MUSICPLAYER\_H   
#define MUSICPLAYER\_H   
   
#include <QWidget>   
#include <QPushButton>   
#include <QFileDialog>   
#include <QHBoxLayout>   
#include <QVBoxLayout>   
#include <QGridLayout>   
#include <QListWidget>   
#include <QSlider>   
#include <QSpacerItem>   
#include <QMediaPlayer>   
#include <QLabel>   
#include <QAudioOutput>   
class musicPlayer : public QWidget   
{   
 Q\_OBJECT   
public:   
 explicit musicPlayer(QWidget \*parent = nullptr);   
public slots:   
 void onFileButtonClicked(); //添加歌曲   
 void playMusic(); //播放音乐   
 void lastMusic(); //跳转到上一首歌曲   
 void nextMusic(); //跳转到下一首歌曲   
 void removeMusic(); //去除选定的歌曲   
 void clearMusic(); //清空歌曲的列表   
 void switchMusic(const QModelIndex &); //切换歌曲   
 void changeVolume(const float); //调节音量   
 void handleMediaStatusChanged(QMediaPlayer::MediaStatus status); //用于解决播放结束时，接下来应该如何播放   
 void setLoopMode(); //设置单曲循环模式   
 void setSortMode(); //设置顺序播放模式   
 void setRandomMode(); //设置随机播放模式   
 void setMode(); //连接播放模式的函数，用于设置播放的模式   
private:   
 QVBoxLayout \*main\_layout; //主布局   
 QHBoxLayout \*menu\_layout; //菜单布局：包括了：添加、去除、清空、播放模式   
 QHBoxLayout \*button\_layout; //按钮布局：包括了：上一首、继续播放、下一首   
 QListWidget \*listWidget; //播放列表：用于显示歌曲菜单   
 QSlider \*scheduleSlider; //播放的进度条   
 QSlider \*volumeSlider; //音量的进度条   
 QList<QUrl> playList; //播放列表：用于存储歌曲的路径   
 QAudioOutput \*m\_audioOutput; //管理音频的输出   
 QMediaPlayer \*m\_mediaPlayer; //管理音频的播放   
 QLabel \*current; //当前的音乐播放进度标签   
 QLabel \*total; //目前播放歌曲的总进度标签   
 QHBoxLayout \*musicSchedule; //用于描述播放进度的布局   
 int m\_currentIndex; //目前播放歌曲的编号   
   
 QPushButton \*addMusicBtn; //添加音乐按钮   
 QPushButton \*lastBtn; //上一首歌曲按钮   
 QPushButton \*nextBtn; //下一首歌曲按钮   
 QPushButton \*playBtn; //继续播放按钮   
 QPushButton \*removeBtn; //去除选中的按钮   
 QPushButton \*clearBtn; //清空歌曲的按钮   
 QPushButton \*modeBtn; //选择模式的按钮   
 bool sortMode; //顺序模式   
 bool loopMode; //单曲循环模式   
 bool randomMode; //随机模式   
};   
   
#endif // MUSICPLAYER\_H

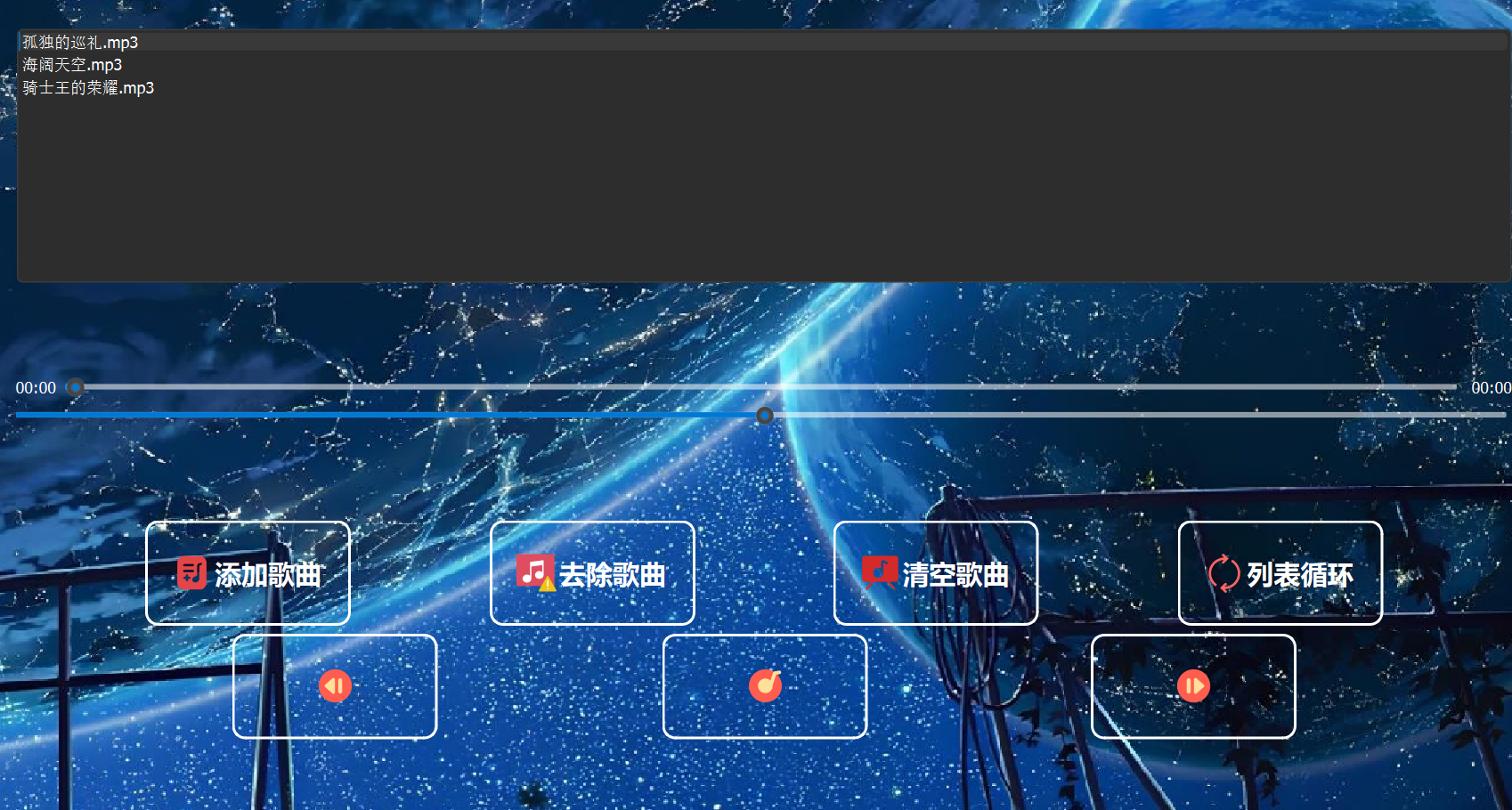
### musicPlayer.cpp文件的实现

首先介绍一下如何实现导入歌曲。

通过点击导入歌曲按钮，会打开文件资源管理器页面，用户通过点击一个或者多个选择，导入歌曲。

可以看到，这样就可以一次性导入多首歌曲。歌曲菜单页面也有更新。





void musicPlayer::onFileButtonClicked()   
{   
 QString path=QDir::homePath();//获取用户目录.   
 QString title= "选择音频文件";   
 QString filter="音频文件(\* .mp3 \*.wav \* .wma) ;;mp3文件(\* .mp3);;wav文件(\* .wav); ;wma文件(\*. .wma);;所有文件(\*.\*)";   
   
 QStringList fileList=QFileDialog::getOpenFileNames(this,title,path,filter);   
   
 if(fileList.empty())   
 {   
 return;   
 }   
   
 for(auto &item:fileList)   
 {   
 QString newFile=item;   
 playList.append(QUrl::fromLocalFile (newFile)); //添加文件   
 QFileInfo fileInfo(newFile);   
 listWidget->addItem(fileInfo.fileName()) ; //添加到界面文件列表   
 }   
 listWidget->setCurrentRow(0);   
}

这里主要是通过QFileDialog::getOpenFileNames获取符合要求的歌曲文件，并且导入到fileList中，并且对其进行转换，添加到视图中和playList中，并且默认当前选中的歌曲为导入的第一首歌曲。

void musicPlayer::playMusic()   
{   
 if(playList.empty())   
 {   
 return;   
 }   
 switch(m\_mediaPlayer->playbackState())   
 {   
 case QMediaPlayer::PlaybackState::StoppedState:   
 {   
 //1、获取当前获得的行号   
 m\_currentIndex=listWidget->currentRow();   
 //2、播放对应行号的音乐   
 m\_mediaPlayer->setSource(playList[m\_currentIndex]);   
 m\_mediaPlayer->play();   
   
 QIcon icon(":/icons/resource/icons/播放.png");   
 playBtn->setIcon(icon);   
 playBtn->setIconSize(QSize(40,40));   
 break;   
 }   
 case QMediaPlayer::PlaybackState::PlayingState:   
 {   
 //如果现在正在播放，则暂停音乐   
 m\_mediaPlayer->pause();   
 QIcon icon(":/icons/resource/icons/暂停.png");   
 playBtn->setIcon(icon);   
 playBtn->setIconSize(QSize(40,40));   
 break;   
 }   
 case QMediaPlayer::PlaybackState::PausedState:   
 {   
 //如果现在是暂停的，则继续播放音乐   
 m\_mediaPlayer->play();   
 QIcon icon(":/icons/resource/icons/播放.png");   
 playBtn->setIcon(icon);   
 playBtn->setIconSize(QSize(40,40));   
 break;   
 }   
 }   
}

接下来是播放音乐时的一些逻辑判断：如果没有播放歌曲，那么播放当前选中的歌曲。

如果当前的歌曲处于播放状态，那么点击播放按钮时，则会切换为暂停状态。

如果当前的歌曲处于暂停状态，那么点击播放按钮时，则会切换为播放状态.

void musicPlayer::lastMusic()   
{   
 if(playList.empty())   
 {   
 return;   
 }   
 m\_currentIndex=(m\_currentIndex-1+playList.size())%playList.size();   
 listWidget->setCurrentRow(m\_currentIndex);   
 m\_mediaPlayer->setSource(playList[m\_currentIndex]);   
 m\_mediaPlayer->play();   
}   
   
void musicPlayer::nextMusic()   
{   
 if(playList.empty())   
 {   
 return;   
 }   
 m\_currentIndex=(m\_currentIndex+1)%playList.size(); //防止下标越界   
 listWidget->setCurrentRow(m\_currentIndex);   
 m\_mediaPlayer->setSource(playList[m\_currentIndex]);   
 m\_mediaPlayer->play();   
}

以上分别是切换到上一首歌曲和下一首歌曲的实现函数，主要需要处理的是：没有导入歌曲时，随便点击按钮可能会导致程序崩溃。

另外还需要注意循环的问题（尤其是切换到上一首歌曲）。

void musicPlayer::switchMusic(const QModelIndex &index)   
{   
 if(playList.empty())   
 {   
 return;   
 }   
 m\_currentIndex=index.row();   
 listWidget->setCurrentRow(m\_currentIndex);   
 m\_mediaPlayer->setSource(playList[m\_currentIndex]);   
 m\_mediaPlayer->play();   
}   
   
//对应的槽函数为：   
connect(listWidget,&QListWidget::doubleClicked,this,&::musicPlayer::switchMusic);

通过双击，可以是切换歌曲到当前选中的歌曲进行播放。

void musicPlayer::removeMusic()   
{   
 if(playList.empty())   
 {   
 return;   
 }   
 int index=listWidget->currentRow();   
 QListWidgetItem \*item=listWidget->takeItem(index);   
 delete item;   
 playList.removeAt(index);   
}   
   
void musicPlayer::clearMusic()   
{   
 if(playList.empty())   
 {   
 return;   
 }   
 m\_mediaPlayer->stop();   
 playList.clear();   
 listWidget->clear();   
}

这两个函数实现的分别是去除歌曲和清空歌曲的功能。

//拖动播放进度条滑块   
connect(scheduleSlider,&QSlider::sliderMoved,m\_mediaPlayer,&QMediaPlayer::setPosition);   
   
//拖动音量进度条滑块   
volumeSlider->setRange(0,100);   
volumeSlider->setSliderPosition(50);   
connect(volumeSlider,&QSlider::sliderMoved,this,&musicPlayer::changeVolume);   
   
void musicPlayer::changeVolume(const float volume)   
{   
 //qInfo()<<"改变前的音量"<<m\_audioOutput->volume();   
 m\_audioOutput->setVolume(volume/100);   
 //qInfo()<<"改变后的音量"<<m\_audioOutput->volume();   
 m\_mediaPlayer->setAudioOutput(m\_audioOutput);   
}

这里实现的是进度条进行调整播放进程，以及调节音量的功能。

一开始不知道m\_audioOutput的音量范围是从0-1，一直实现不了调节功能，通过答应输出发现音量改变前后一直都是1，猜测并发现volume的有效范围是0-1，所以这里需要对传入的volume除以1000。

void musicPlayer::setLoopMode()   
{   
 loopMode=true;   
 randomMode=false;   
 sortMode=false;   
 modeBtn->setText("单曲循环");   
 QIcon icon(":/icons/resource/icons/单曲循环.png");   
 modeBtn->setIcon(icon);   
 modeBtn->setIconSize(QSize(30,30));   
}   
   
void musicPlayer::setRandomMode()   
{   
 randomMode=true;   
 loopMode=false;   
 sortMode=false;   
 modeBtn->setText("随机播放");   
 QIcon icon(":/icons/resource/icons/随机播放.png");   
 modeBtn->setIcon(icon);   
 modeBtn->setIconSize(QSize(30,30));   
}   
   
void musicPlayer::setSortMode()   
{   
 sortMode=true;   
 loopMode=false;   
 randomMode=false;   
 modeBtn->setText("列表循环");   
 QIcon icon(":/icons/resource/icons/列表循环.png");   
 modeBtn->setIcon(icon);   
 modeBtn->setIconSize(QSize(30,30));   
}   
   
void musicPlayer::setMode()   
{   
 if(loopMode && !sortMode && !randomMode)   
 {   
 setSortMode();   
 }   
 else if(!loopMode && sortMode && !randomMode)   
 {   
 setRandomMode();   
 }   
 else if(!loopMode && !sortMode && randomMode)   
 {   
 setLoopMode();   
 }   
}   
   
void musicPlayer::handleMediaStatusChanged(QMediaPlayer::MediaStatus status)   
{   
 if(status==QMediaPlayer::EndOfMedia)   
 {   
 if(loopMode)   
 {   
 m\_mediaPlayer->setPosition(0);   
 m\_mediaPlayer->play();   
 }   
 else if(sortMode)   
 {   
 m\_currentIndex=(m\_currentIndex+1)%playList.size();   
 m\_mediaPlayer->setSource(playList[m\_currentIndex]);   
 listWidget->setCurrentRow(m\_currentIndex);   
 m\_mediaPlayer->play();   
 }   
 else   
 {   
 m\_currentIndex=QRandomGenerator::global()->bounded(playList.size());   
 listWidget->setCurrentRow(m\_currentIndex);   
 m\_mediaPlayer->setSource(playList[m\_currentIndex]);   
 m\_mediaPlayer->play();   
 }   
 }   
}   
   
//对应的槽函数：歌曲播放结束时，选择的是哪一种播放模式   
connect(m\_mediaPlayer,&QMediaPlayer::mediaStatusChanged,this,&musicPlayer::handleMediaStatusChanged);   
   
//对应的槽函数，通过点击，切换到不同的播放模式   
connect(modeBtn,&QPushButton::clicked,this,&musicPlayer::setMode);

以上是设置播放模式：分别有三个函数实现setRandomMode，setLoopMode，setSortMode，通过函数setMode对三个状态分别进行设置。

当歌曲播放结束时，通过槽函数，音乐播放器会查看当前处于哪一种播放模式，并根据对应的播放模式进行播放。