

《程序设计（上机）》报告

（ 2024/25 学年 第 一 学期）

题 目：**简单音乐播放器**

**这是我重修这门课的报告，大二时只觉得实验课痛苦，大三时才意识到项目经验的不足。程序很简陋，甚至没有用到数据库和网络编程相关的知识，但拿个95分还是绰绰有余的。**

**比较重要的点在报告里面讲得也很清楚了，希望同学们可以好好学习这些代码，不要全篇抄袭，争取灵活运用QT环境，祝各位学有所成。**

**简单音乐播放器**

**一、课题内容和要求**

音乐播放器系统是一种可以通过读取歌曲文件实现歌曲播放的各种功能的多媒体控制系统。通过该系统，可以实现音乐的播放、暂停、继续播放、停止功能；可以调节音量、任意选择播放曲目；在播放方式上，可以实现顺序播放、随机播放、单曲循环。用户可以将喜爱的音乐收藏进个人收藏夹，并且可以向收藏夹中增加曲目，也可以删除收藏曲目。

实现读取歌单文件浏览歌曲信息的浏览功能；选择相应的歌名，播放该歌曲，并可暂停、继续、停止等的点歌功能；浏览歌单，根据歌单形成收藏夹文件，并能对文件进行增加、删除、查找功能的管理收藏夹功能；采用友好的字符界面，实现一个功能控制菜单；每次操作都从该菜单选择，在菜单中可进行多次操作；歌单编号不得重复；可以实现歌曲的上一首、下一首功能；可以实现播放模式的选择（顺序播放、随机播放、单曲循环等），歌曲播完后会自动根据播放模式来自动播放歌单中的歌曲；上一首、下一首功能的实现，并且保证该功能受播放模式的影响；对歌曲音量的显示和调节；对歌曲进度的显示和调节。

使用QT开发工具，实现彩色或图形操作界面。

**二、需求分析**

简单音乐播放器的功能框架图如图1所示。

图示

描述已自动生成

图1 简单音乐播放器功能框架图

（1）上/下一曲功能：在每次操作中，都会更新播放器的歌曲源，并更新 UI 中歌曲列表的当前行，同时更新显示的歌曲封面或其他相关信息。。

（2）收藏功能：该功能实现了音乐播放器的收藏与取消收藏功能，根据当前歌曲的索引将其添加或移除收藏列表，并更新显示。

（3）播放模式切换：该功能实现了音乐播放器的播放模式切换，依次在顺序循环、单曲循环和随机播放模式之间切换，并更新图标和循环设置。

（4）显示专辑图片：该功能根据当前歌曲索引更新音乐封面图片显示。

（5）暂停/播放功能：该功能实现了音乐播放器的播放和暂停切换，根据当前播放状态决定播放、暂停或继续播放操作，并更新暂停按钮的图标。

（6）音量与进度调节：该功能实现了通过滑动播放进度条来调整音乐的播放位置，将播放位置设置为当前进度条的位置。音量大小调节同理。

（7）双击点歌功能：该功能实现了在收藏列表和歌曲列表中双击歌曲时，获取对应的歌曲索引并播放，同时更新播放器状态和封面图像。

**三、概要设计**

1 主要存储结构，如图2所示。



图2私有成员变量

2 主要函数流程

（1）上/下一曲功能：上一曲，如果当前没有选中的歌曲，则退出函数。根据不同的播放模式（由stage决定），执行不同的逻辑：模式0（正常顺序播放）：通过递减curindex来播放上一首歌曲，使用%确保curindex不会小于0。模式 1（未定义行为）：不做任何操作。模式2（历史记录模式）：删除当前歌曲的历史记录，若没有历史记录，则默认为播放第一首歌。根据历史记录播放上一首歌曲。播放选中的歌曲并更新UI显示。下一曲，如果当前没有选中的歌曲，则退出函数。根据不同的播放模式（由stage决定），执行不同的逻辑：模式0（正常顺序播放）：通过递增curindex来播放下一首歌曲，使用%确保curindex不超过列表的大小。模式1（未定义行为）：不做任何操作。模式2（随机播放）：随机选择一首歌曲播放。播放选中的歌曲并更新 UI 显示。如图2所示：

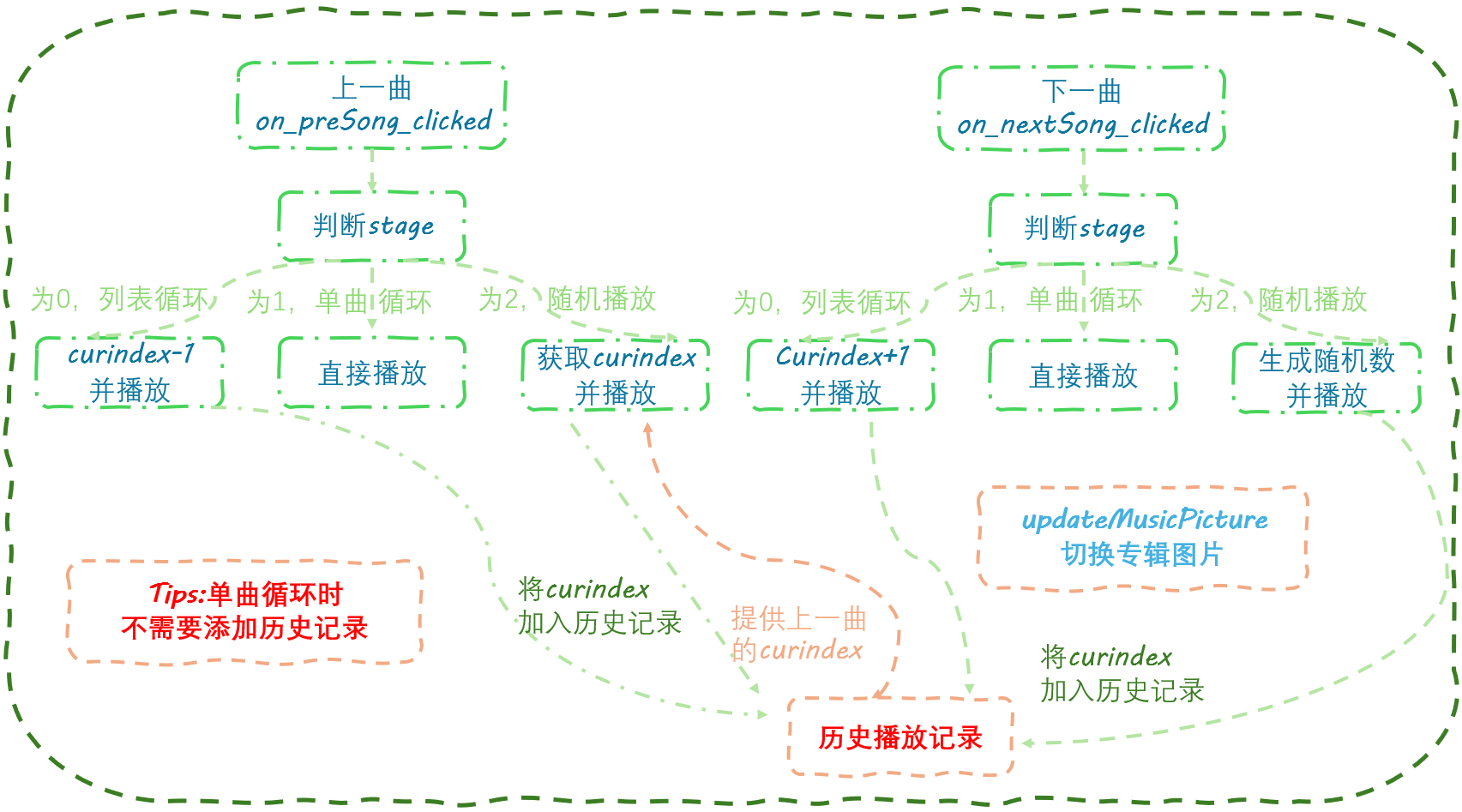


图2 上/下一曲功能示意图

这里给出实现上一曲功能的源码，如图3所示：



图3 上一曲功能源码

给出下一曲功能源码，如图4所示：



图4 下一曲功能原码

歌曲自动的下一曲逻辑与此类似，先看流程图，如图5所示：

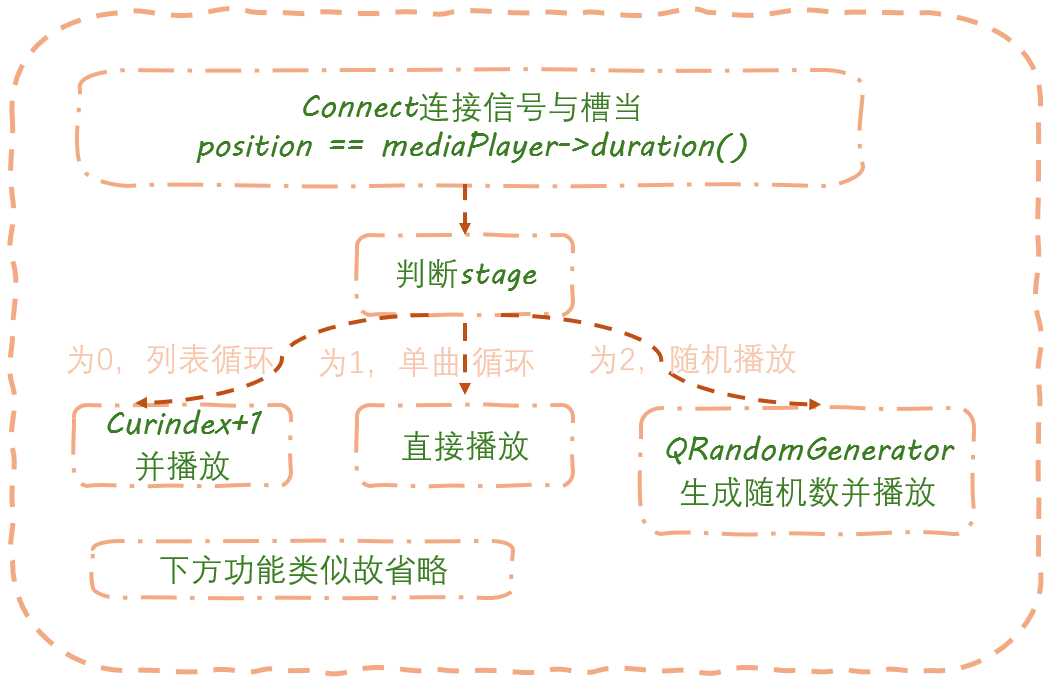


图5 自然播放下一曲功能示意图

源码如图6：



图6：自然播放下一曲源码

（2）播放/暂停功能，流程图如图7所示

图示

描述已自动生成

图7 播放/暂停功能

源码如图8所示：

文本

描述已自动生成

图8 播放/暂停功能源码

（3）收藏功能，核心是两个if判断，先判断索引是否越界，再判断vecbool的状态是收藏还是未收藏，如图9所示：

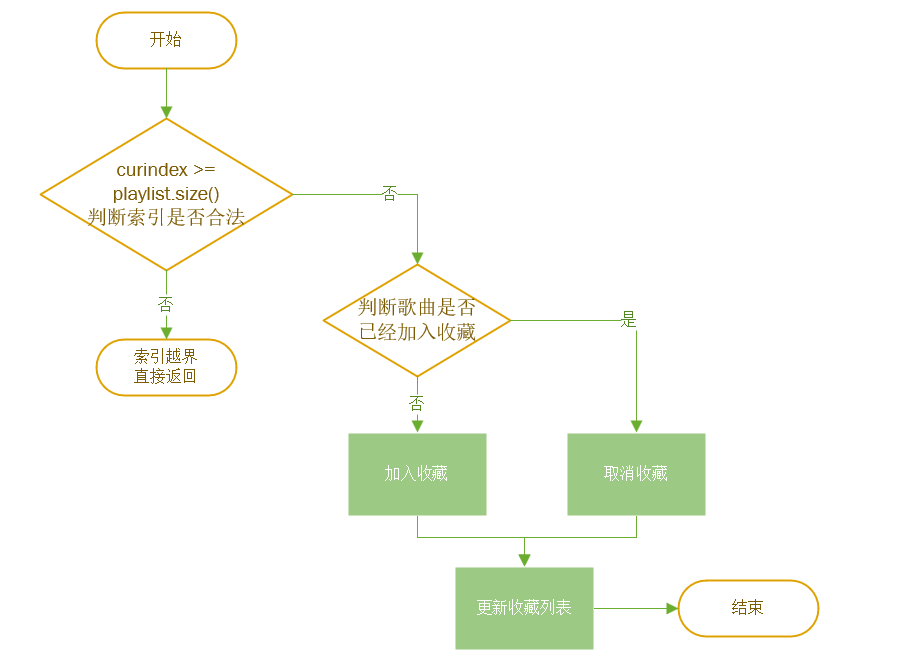


图9 收藏功能流程图

（4）播放模式切换：点击QPushButton按钮，模式按照列表循环（stage=0）、单曲循环（stage=1）、随机播放（stage=2）的顺序播放。流程图如图10所示：

图示

描述已自动生成

图10 播放模式切换流程图

（5）双击点播功能：点播功能包括歌曲列表的点播和收藏列表的点播，以歌曲列表点播为例，流程图如图11所示：



图11 列表点播流程图

这里获取索引的方式是curindex=index.row()，其中index是信号传来的变量，交给on\_songList\_doubleClicked()这个槽函数处理，充分体现了QT的信号与槽思想，为了更好理解函数，这里给出源代码的可视化截图。

列表点播源代码如下：



图12 列表点播源代码

（6）点击音量按钮调出/隐藏音量调节滑块，示意图如图13：



图13 音量调节滑块换入换出示意图

源代码如图14：

文本

低可信度描述已自动生成

图14 音量调节滑块换入换出源码

这里同时给出其流程图，如图15：

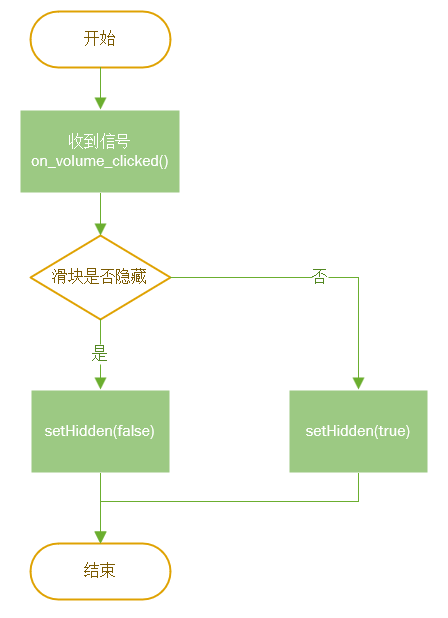


图15 调节音量滑块流程图

**四、源程序代码**

本项目的文件基本结构和作用如图16所示：



图16 项目文件基本结构和作用

项目主要的代码是框架自动生成的，这里给出MusicPlayer.cpp文件的源码，我在这里实现了Musicplayer类中声明的函数。

#include "MusicPlayer.h"

#include "ui\_MusicPlayer.h"

#include <QDebug>

#include <QFileDialog>

#include <QDir>

#include <QMediaPlayer>

#include <QAudioOutput>

#include <QString>

#include <QUrl>

#include <QDirIterator>

#include <QRandomGenerator>

#include <QList>

#include <QSharedPointer>

MusicPlayer::MusicPlayer(QWidget \*parent)

: QWidget(parent)

, ui(new Ui::MusicPlayer)

{

ui->setupUi(this);

ui->verticalSlider->setHidden(true);

//使用 QScopedPointer自动管理mediaPlayer和audioOutput

mediaPlayer.reset(new QMediaPlayer(this)); //使用reset并将新的QMediaPlayer实例绑定到mediaPlayer指针上

audioOutput.reset(new QAudioOutput(this));

mediaPlayer->setAudioOutput(audioOutput.data()); //将 audioOutput传给mediaPlayer

//connect(sender, signal发出的信号, receiver, slot槽函数)

//展示播放文件总时长

//使用lambda表达式获取duration

connect(mediaPlayer.get(),&QMediaPlayer::durationChanged,this,[=](qint64 duration){

ui->totallabel->setText(QString("%1:%2").arg(duration/1000/60,2,10,QChar('0')).arg(duration/1000%60,2,10,QChar('0')));

//给播放进度条滑块设置总时长

ui->playpaceSlider->setRange(0,duration);

});

//展示播放文件当前时长

connect(mediaPlayer.get(),&QMediaPlayer::positionChanged,this,[=](qint64 position){

ui->curlabel->setText(QString("%1:%2").arg(position/1000/60,2,10,QChar('0')).arg(position/1000%60,2,10,QChar('0')));

//给播放进度条滑块设置当前进度

ui->playpaceSlider->setValue(position);

//自动下一曲

if (position == mediaPlayer->duration()) {

if (ui->songList->currentRow() == -1) {

return;

}

switch (\*stage) {

case 0: { //顺序循环

curindex = (++curindex) % playlist.size(); //下一首

ui->songList->setCurrentRow(curindex);//选中对应歌曲，但不播放，仅ui变化

playNextSong(curindex);//加入历史记录

mediaPlayer->setSource(playlist[curindex]);

mediaPlayer->play();

break;

}

case 1: { //单曲循环

//mediaPlayer->play(); //播放当前歌曲，下面的-1已经无限循环了

break;

}

case 2: { //随机播放

//生成一个介于0和playlist.size()-1之间的随机整数

curindex = QRandomGenerator::global()->bounded(playlist.size());

ui->songList->setCurrentRow(curindex);

playNextSong(curindex);

mediaPlayer->setSource(playlist[curindex]);

mediaPlayer->play();

break;

}

}

}

});

connect(mediaPlayer.get(), &QMediaPlayer::mediaStatusChanged, this, &MusicPlayer::onMediaStatusChanged);

// 设置初始音量

audioOutput->setVolume(0.5);

// 指定默认的音乐文件夹路径

QString defaultPath = "D:/MusicPlayer/MusicPlayer/resource/resourcemusic";

// 加载音乐文件

loadMusicFiles(defaultPath);

//加载收藏夹列表，设置为空

for (int i = 0; i < playlist.size(); ++i) {

vecbool.append(0); // 动态添加元素

}

history.clear();

}

//加入历史记录，保证随机播放时回溯

void MusicPlayer::playNextSong(int curindex)

{

// 如果历史记录超过最大长度，删除最旧的记录

if (history.size() >= maxHistorySize) {

history.removeFirst(); // 删除最旧的歌曲

}

history.append(curindex); // 将当前歌曲索引添加到历史记录

}

MusicPlayer::~MusicPlayer()

{

delete ui;

}

//加载播放列表

void MusicPlayer::loadMusicFiles(const QString &path)

{

//检查路径是否有效

if (path.isEmpty()) {

return;

}

//遍历指定文件夹中的音乐文件

QDirIterator it(path, {"\*.mp3", "\*.wav", "\*.ogg"}, QDir::Files); // 只遍历文件

while (it.hasNext()) {

it.next(); //移动到下一个文件

QFileInfo info = it.fileInfo(); //获取文件信息

//将完整路径转为 QUrl 并添加到播放列表

//获取文件的标准化路径，为了播放歌曲

playlist.append(QUrl::fromLocalFile(info.canonicalFilePath()));

}

//更新UI中的播放列表

QStringList songNames;

for (const QUrl &url : playlist) {

//为了显示歌曲名

songNames.append(url.fileName()); //获取文件名

}

ui->songList->addItems(songNames); //更新歌曲列表显示

}

//播放和暂停

void MusicPlayer::on\_pause\_clicked()

{

QModelIndex currentIndex = ui->songList->selectionModel()->currentIndex();

int currentRow = currentIndex.row(); //获取当前选中的行号

//无播放文件时不响应

if (currentRow == -1) {

return;

}

//获取选中的音乐文件路径

curindex = currentRow; //更新当前索引

QUrl selectedSong = playlist[curindex]; //获取选中的歌曲 URL

//不同状态对应不同的操作

switch (mediaPlayer->playbackState()) {

case QMediaPlayer::PlaybackState::StoppedState: {

//设置音频源并播放

mediaPlayer->setSource(selectedSong);

mediaPlayer->play();

ui->pause->setStyleSheet("icon: url(:/resource/playing.png);");

break;

}

case QMediaPlayer::PlaybackState::PausedState: {

//继续播放

mediaPlayer->play();

ui->pause->setStyleSheet("icon: url(:/resource/playing.png);");

break;

}

case QMediaPlayer::PlaybackState::PlayingState: {

//暂停播放

mediaPlayer->pause();

ui->pause->setStyleSheet("icon: url(:/resource/pause.png);");

break;

}

}

}

//上一曲

void MusicPlayer::on\_preSong\_clicked()

{

if(ui->songList->currentRow()==-1){

return;

}

switch (\*stage) {

case 0:{

curindex=(--curindex+playlist.size())%playlist.size();

playNextSong(curindex);

break;

}

case 1:{

break;

}

case 2:{

history.removeLast();//删除当前歌曲的历史记录

if (history.isEmpty()) {

history.append(0); //如果没有历史记录，默认第一首歌

}

curindex = history.takeLast();

playNextSong(curindex);

break;

}

}

ui->songList->setCurrentRow(curindex);

mediaPlayer->setSource(playlist[curindex]);

mediaPlayer->play();

updateMusicPicture(curindex);

}

//下一曲

void MusicPlayer::on\_nextSong\_clicked()

{

if(ui->songList->currentRow()==-1){

return;

}

switch (\*stage) {

case 0:{

curindex=(++curindex)%playlist.size();

playNextSong(curindex);

break;

}

case 1:{

break;

}

case 2:{

curindex = QRandomGenerator::global()->bounded(playlist.size());

playNextSong(curindex);

break;

}

}

ui->songList->setCurrentRow(curindex);

mediaPlayer->setSource(playlist[curindex]);

mediaPlayer->play();

updateMusicPicture(curindex);

}

//双击列表播放

void MusicPlayer::on\_songList\_doubleClicked(const QModelIndex &index)

{

curindex=index.row();

playNextSong(curindex);

mediaPlayer->setSource(playlist[curindex]);

updateMusicPicture(curindex);

mediaPlayer->play();

ui->pause->setStyleSheet("icon: url(:/resource/playing.png);");

}

//随播放进度条滑块移动改变播放进度

void MusicPlayer::on\_playpaceSlider\_sliderMoved()

{

mediaPlayer->setPosition(ui->playpaceSlider->sliderPosition());

}

//点击音量按钮调出/隐藏音量调节滑块

void MusicPlayer::on\_volume\_clicked()

{

if(ui->verticalSlider->isHidden()){

ui->verticalSlider->setHidden(false);

}

else{

ui->verticalSlider->setHidden(true);

}

}

//随音量滑块移动调节音量

void MusicPlayer::on\_verticalSlider\_valueChanged(int value)

{

audioOutput->setVolume(float(value)/100);

}

//模式切换

void MusicPlayer::on\_modelstage\_clicked()

{

switch (\*stage) {

case 0: { //顺序循环

\*stage = 1; //切换到单曲循环

ui->modelstage->setStyleSheet("icon: url(:/resource/songrepeat.png);"); // 单曲循环图标

mediaPlayer->setLoops(-1); //设置为单曲循环

break;

}

case 1: { // 单曲循环

\*stage = 2; //切换到随机播放

ui->modelstage->setStyleSheet("icon: url(:/resource/listrandom.png);"); // 随机播放图标

mediaPlayer->setLoops(1); //设置为无限循环

break;

}

case 2: { //随机播放

\*stage = 0; //切换到顺序循环

ui->modelstage->setStyleSheet("icon: url(:/resource/listrepeat.png);"); // 顺序循环图标

mediaPlayer->setLoops(1); //设置为顺序播放一次

break;

}

}

}

//收藏和取消功能

void MusicPlayer::on\_love\_clicked()

{

//判断当前索引是否合法

if (curindex >= playlist.size()) {

return; //索引越界，直接返回

}

// 加入收藏

if (vecbool[curindex] == 0) {

vecbool[curindex] = 1; // 将当前索引位置的元素设置为喜欢

//如果当前歌曲没有在 loveNames 中，则加入

if (!loveNames.contains(playlist[curindex].fileName())) {

loveNames.append(playlist[curindex].fileName()); //插入 playlist 对应索引的文件名

veclove.append(curindex); //将当前索引添加到收藏列表

}

} else { //取消收藏

vecbool[curindex] = 0; //将当前索引位置的元素取消喜欢

//倒序遍历 loveNames 列表，防止删除时引起索引问题

for (int i = loveNames.size() - 1; i >= 0; --i) {

if (loveNames[i] == playlist[curindex].fileName()) {

loveNames.removeAt(i); //从列表中移除该歌曲

veclove.removeAt(i); //从收藏列表中删除

break; //只删除第一次出现的匹配项

}

}

}

//更新 favoritesList 显示

ui->favoritesList->clear(); //清空原来的列表

ui->favoritesList->addItems(loveNames); //一次性添加更新后的列表

}

//双击收藏列表播放

void MusicPlayer::on\_favoritesList\_doubleClicked(const QModelIndex &index)

{

curindex = veclove[index.row()]; //获取收藏列表中的歌曲索引

playNextSong(curindex);

mediaPlayer->setSource(playlist[curindex]);

mediaPlayer->play();

updateMusicPicture(curindex);

ui->pause->setStyleSheet("icon: url(:/resource/playing.png);");

}

//音乐自然播放完毕时触发信号

void MusicPlayer::onMediaStatusChanged(QMediaPlayer::MediaStatus status) {

if (status == QMediaPlayer::EndOfMedia) {

updateMusicPicture(curindex);

}

}

void MusicPlayer::updateMusicPicture(int nowindex)

{

//根据 curindex 的值动态构建图片路径

QString imagePath = QString(":/resource/musicpicture/%1.png").arg(nowindex);

//qDebug() << "Image path: " << imagePath;

//设置图片路径到 musicPicture 控件

ui->musicPicture->setStyleSheet(QString("image: url(%1);").arg(imagePath));

}

**五、测试数据及其结果分析**

（1）收藏功能测试：测试歌曲的是否可以正常收藏和取消，收藏列表是否可以点播，测试逻辑如图17所示：

图形用户界面

描述已自动生成

图17 收藏功能测试图

收藏功能用到了两个容器存放数据。一个是vecbool容器，判断该歌曲是否被收藏。另外一个是veclove容器，存放收藏列表行索引对应的歌曲列表行索引，这样做的目的是实现取消收藏和歌曲点播功能。

同时，收藏列表的每次点播都会加入历史记录，这为下面随机播放模式下的上一曲功能提供了可回溯的历史记录。

（2）测试上一曲是否可以根据不同播放模式改变功能。单曲循环的上一曲功能最为简单，无需做任何操作，直接break。列表循环的上一曲需要对索引进行减一的操作，并判断是否越界，这里用一行代码实现，如图18：

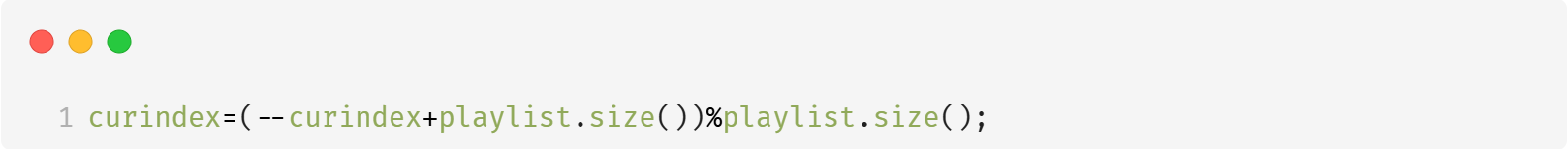


图18 改变上一曲索引源码

随机播放模式下的上一曲功能最为复杂，首先需要对每次播放的歌曲都加入历史记录，最后再通过搜索记录回溯。具体测试逻辑如图19：



图19 上一曲测试流程图

多模式混合的上一曲测试如图20所示：

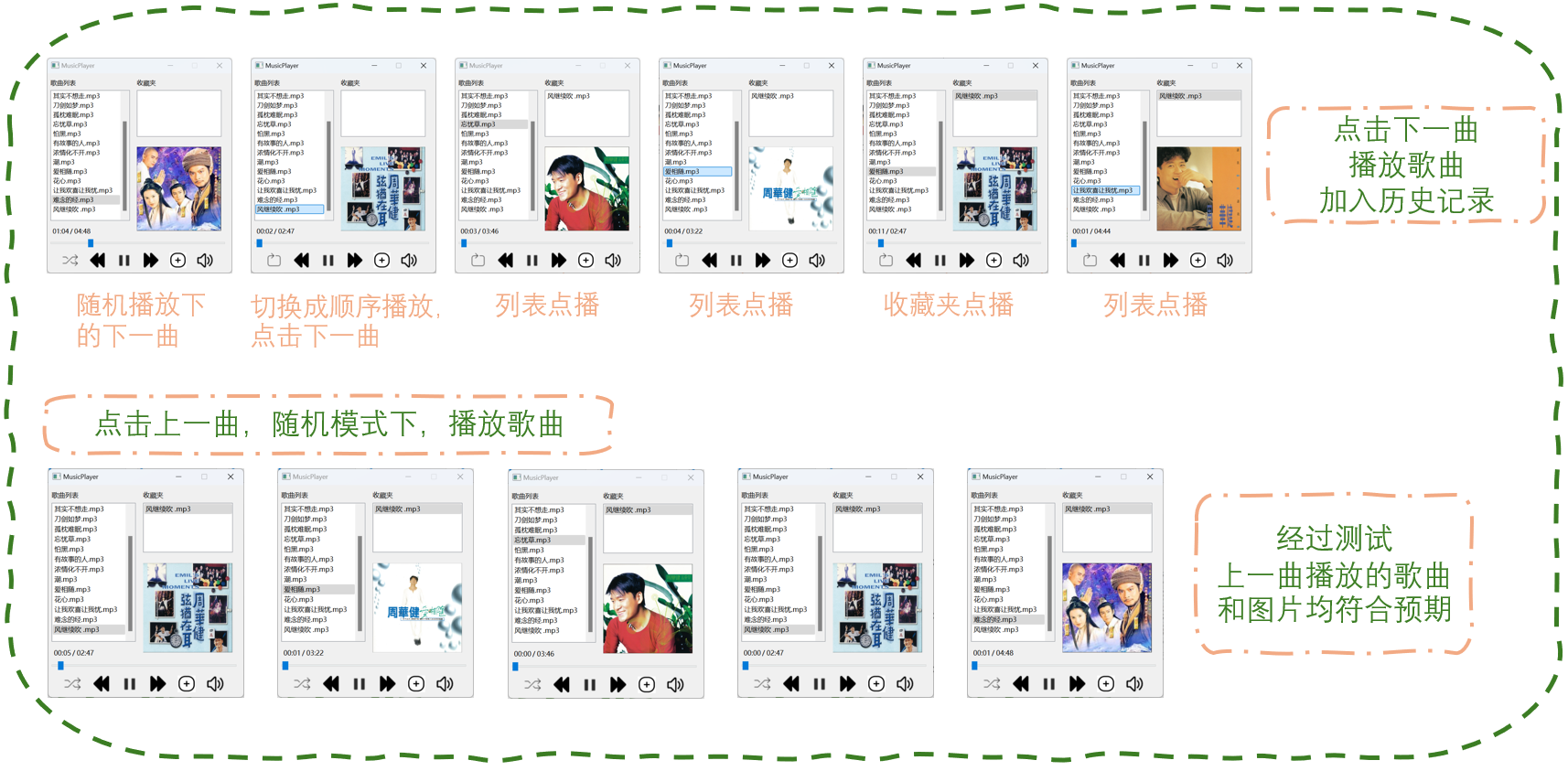


图20 多模式混合的上一曲测试

不难发现，测试结果完美符合预期。

**六、调试过程中的问题**

（1）对象忘记释放造成内存泄漏。

解决办法：用智能指针管理对象的生命周期。源码如图21所示：

图21 智能指针部分源码

QScopedPointer 会在对象超出作用域时自动释放其所管理的对象。因此，它避免了显式的delete操作。从而简化内存管理并降低出错的风险。

（2）索引越界。

在上一曲和下一曲的实现代码中，分别对curindex进行了减一和加一的操作，这可能会导致索引越界。

所以要在操作后对索引模最大值，具体代码如图22所示：



图22 上一曲和下一曲求模源码

一行代码完成减一/加一操作和取模的两个功能。

（3）切歌后专辑图片不更换。

分析源码，如图23所示：

文本

中度可信度描述已自动生成

图23 切换专辑图片源码

可以发现，这里槽函数等待的信号是mediaStatusChanged，也就是歌曲状态发生改变才会触发。所以当歌曲自然播放完毕会更换专辑图片，而人为切歌不会。因此需要在每一次人为切歌时都调用updateMusicPicture函数，保证专辑图片成功更换。

（4）随机模式下的上一曲功能不正常，重复当前歌曲。

分析源码，如图24所示：

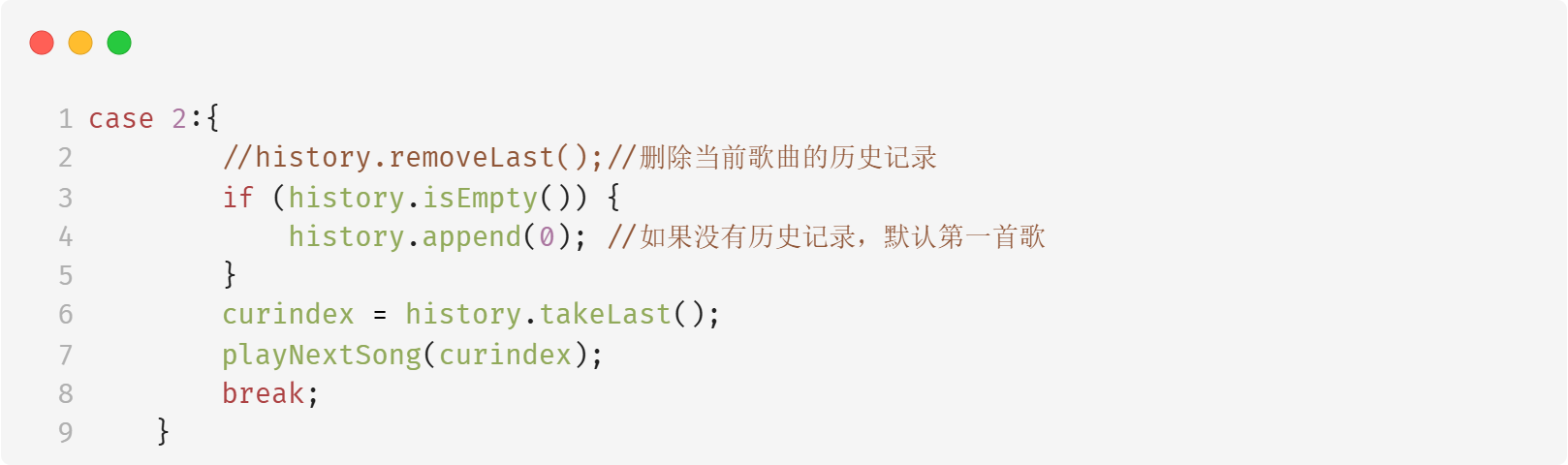


图24 随机模式上一曲源码

发现播放歌曲会将当前歌曲加入历史记录，所以上一曲时先要删除当前播放歌曲的索引。也就是图中被注释掉的代码history.removeLast。

为了防止history为空，导致程序崩溃。所以我在播放上一曲前先判断，如果为空，则播放歌曲列表的第一首歌。

（5）播放时间显示有误。播放时间不是两位数，不整齐。

规定显示两位数，十进制，不足部分用字符“0”补齐，源码如图25所示：

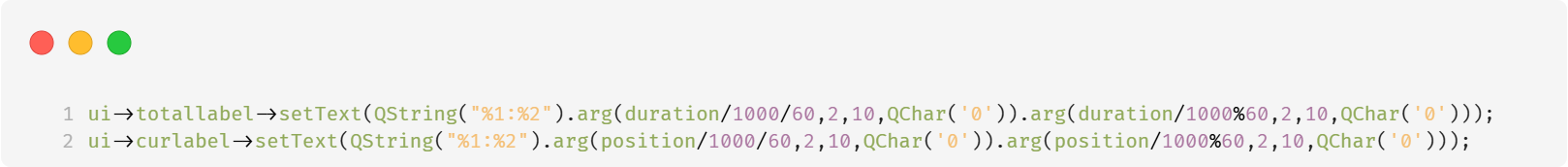


图25 补“0”源码

duration的值为毫秒，除以1000再对60取模即秒，对除60即求得分钟。

（6）删除收藏列表的索引时，其后续索引出现遗漏。

可以采取倒序遍历，因为删除的索引不影响前面索引的序号，源码如图26所示：

图片包含 散点图

描述已自动生成

图26 倒叙遍历源码

（7）显示的专辑图片与预期不符。

通过debug功能打印日志，阅读日志发现收藏功能的索引有误。具体代码如图27所示：

日程表

低可信度描述已自动生成

图27 打印日志源代码

**七、课程总结**

在本次课程的程序设计过程中，作为团队唯一成员，我深刻感受到了程序开发的复杂性与挑战性。整个开发过程不仅是对编程技术的考验，也是对项目管理、问题解决能力以及个人综合素质的提升。通过这一项目，我在实践中获得了宝贵的经验，积累了许多教训，并且对如何有效进行程序设计和开发有了更深刻的理解。

首先，明确需求和功能模块的划分是项目成功的关键。在开发之前，我花费了大量时间进行需求分析，确定了程序的基本功能：音乐文件加载、播放控制、音量调节、播放模式切换、历史记录和收藏夹等功能。对于每个功能模块，我都进行了详细的设计，确保各个功能之间相互独立、解耦，使得后期的开发和调试工作更加顺利。

在开发过程中，我深刻体会到良好的编码规范和代码结构的重要性。通过清晰的代码逻辑和适当的注释，能够帮助我们在后期的维护和优化中减少许多不必要的麻烦。例如，使用 `QScopedPointer` 来管理 `mediaPlayer` 和 `audioOutput` 的生命周期，避免了内存泄漏的问题；通过 `connect` 来连接信号和槽，确保了 UI 的及时更新和功能的实时响应。项目中充分使用了 Qt 框架提供的控件和特性，合理利用了信号与槽机制，大大简化了事件的处理。此外，调试和测试的过程中，我也获得了不少宝贵的经验。在面对 bug 时，特别是播放进度条与音量调节不兼容的问题时，我们采用了分阶段调试的策略，逐步排查各个模块，最终定位到问题所在。

通过本次项目开发，我深刻认识到程序设计并不仅仅是编写代码这么简单。它涉及到的知识面非常广泛，既包括数据结构、算法等基础知识，也包括开发工具的使用、界面设计、用户体验等方面。一个看似简单的音乐播放器，实际涉及到了文件管理、音频处理、UI 更新、事件响应等多个层面，只有将这些方面有机地结合在一起，才能够实现预期的效果。

在项目实施过程中，我也发现了一些不足之处。例如，尽管我在程序设计中尽量做到了模块化和结构化，但在一些细节设计上仍然存在可以改进的空间。比如，UI 的交互设计可以更加细致，播放模式切换的逻辑可以更直观，音量控制方面也可以加入更多的个性化设置。这些都提醒我们，在实际的项目开发中，必须时刻关注用户体验和细节优化。

总之，通过本次程序设计课程，我积累了丰富的实践经验，提升了自己的技术水平，明确了今后学习和工作的方向。程序设计不仅仅是实现功能，更是对问题的思考与创新。我期待在未来的项目中，能够更加注重创新和优化，不断提高自身的编程能力和综合素质，为软件开发领域做出更多贡献。