# 个人报告

何淂劲3130100676

## 主要工作

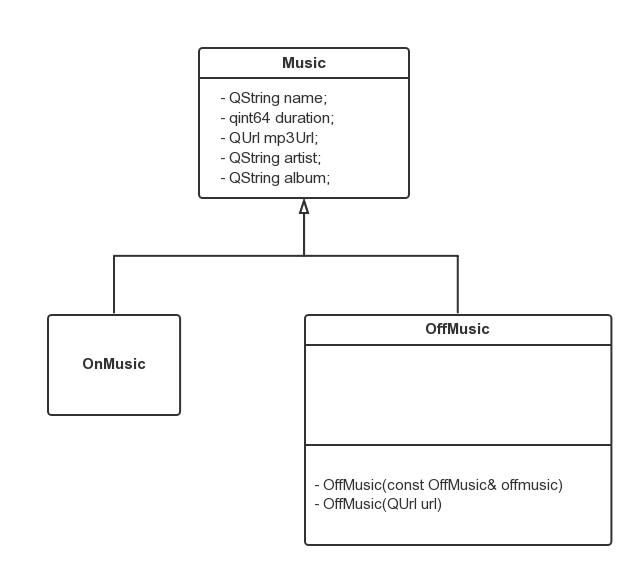
1. 撰写需求分析。
2. 读取音乐文件时对文件内容的解析。编写继承Music类的OffMusic类。撰写OffMusic.h。
3. 离线记录歌单的功能。使用json文件格式存储歌单。撰写PlaylistRecord.h
4. 完成src/commander/implementation中各个函数实现相关内容的填写。
5. 参与Player类部分功能的撰写，并与汪克杰同学一起完成播放器底层功能的debug工作。

接下来详细介绍1，2两点，第3点就是和ViewModel层进行交流并填写相应的函数，第4点交由主要负责Player类的汪克杰同学阐述。

OffMusic类

## 实现原理

Music类拥有数个需要提供给用户的信息，具体表现为交给protected属性，包括音乐名称，歌曲时长，歌曲路径，歌手名称以及专辑名称。而OffMusic类继承于Music类，自然继承了以上的protected属性。不同的是以url为参数进行构造，在构造函数中使用第三方库Taglib解析出需要的以上信息。



图表 1 OffMusic UML图

## 实现过程

1. 首先要下载Taglib的源代码，按照其中INSTALL文件的指示使用CMake和Make进行编译。为了以后方便他人对整个工程进行编译，我选择了编译出了对应的静态库libtag.a。使用Qt Creator添加工程需要用到的libtag.a，并且将第三方库的include文件添加到整个工程的include路径之中。
2. 撰写相关代码:

#ifndef OFFMUSIC

#define OFFMUSIC

#include "music.h"

#include "3rdpart/taglib/include/fileref.h"

//继承Music类

class OffMusic : public Music

{

public:

OffMusic() {}

OffMusic(const OffMusic& offmusic){

name = offmusic.name;

duration = offmusic.duration;

mp3Url = offmusic.mp3Url;

artist = offmusic.artist;

album = offmusic.album;

}

//使用url构造OffMusic类

OffMusic(QUrl url){

//先用url读入文件构造Taglib自己的文件解析类FileRef

TagLib::FileRef ref(url.path().toLatin1().data());

//检测是否成功

if(!ref.tag())

{

qDebug() << "NULL";

}

//调用FileRef的成员函数tag()获取标签，进而读取各项歌曲信息

name = QString(ref.tag()->*title*().toCString());

duration = ref.audioProperties()->*length*()\*1000;

mp3Url = url;

artist = QString(ref.tag()->*artist*().toCString());

album = QString(ref.tag()->*album*().toCString());

//如果使用taglib获取不了歌曲名称，则使用url中最后一段获取歌曲名称

if(name.isEmpty())

{

QString qstr = url.fileName();

QStringList list = qstr.split(".");

if(!list.isEmpty())

name = list.first();

else

name = "UNKNOWN";

}

}

virtual ~*OffMusic*(){}

};

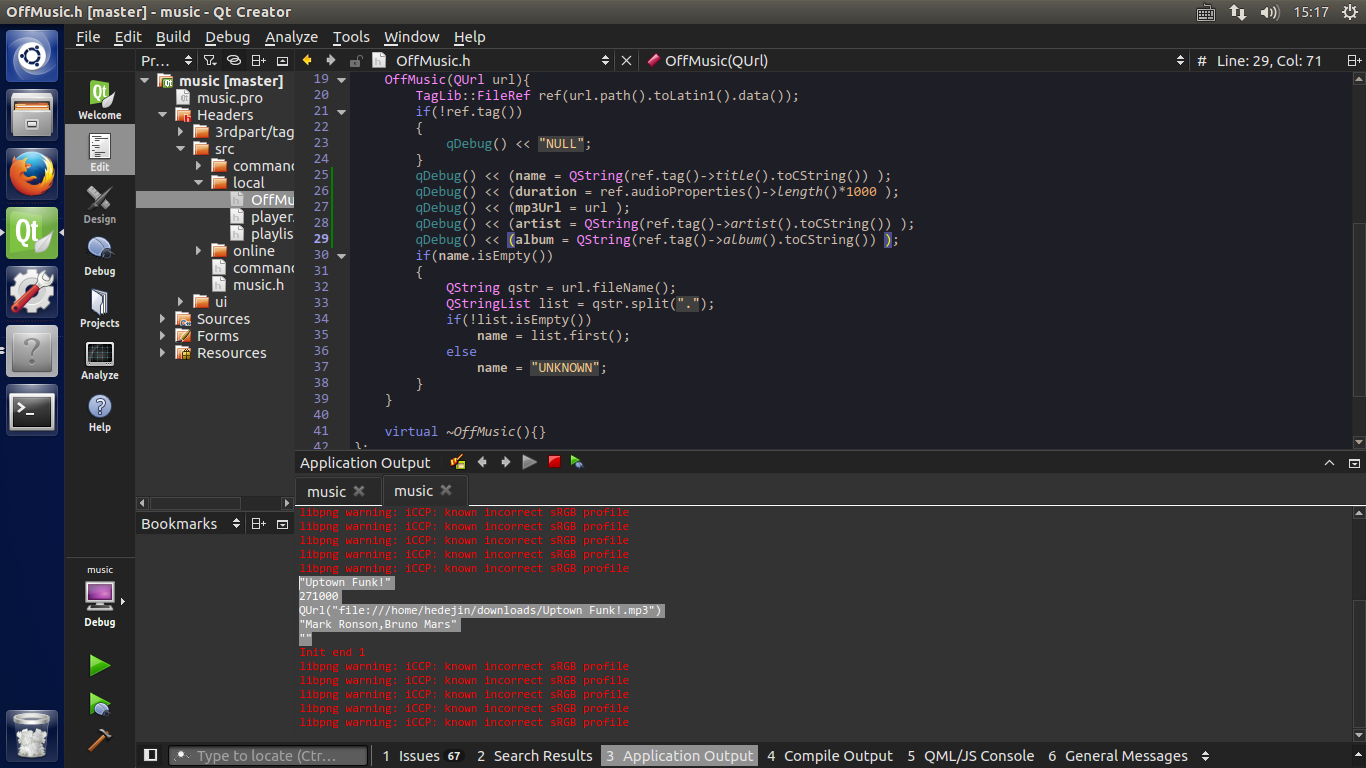
#endif

1. 编译时会提示需要源文件id3v2frame.cpp才能进行编译，再把需要的id3v2frame.cpp从Taglib源代码中找出，并添加到工程之中，qmake之后即可完成编译。

## 实现效果

当传入一个QUrl参数时，可以构造出一个含有歌曲信息的OffMusic对象。

## 测试记录



从下方程序输出的debug()信息可以看到，在OffMusic构造的过程中成功读出并记录了歌曲名称、歌曲时长、url、专辑名称等信息。

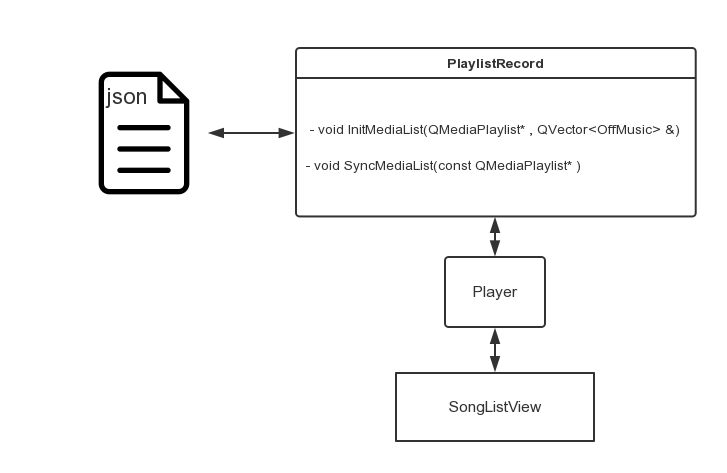
PlaylistRecord.h

## 实现原理

PlayerlistRecord.h中有两个函数，一个是void SyncMediaList(const QMediaPlaylist\* )，另外一个是void InitMediaList(QMediaPlaylist\* , QVector<OffMusic> &)。

每次程序结束，在MainWindow的析构函数中调用void SyncMediaList(const QMediaPlaylist\* )，将此时的QMediaPlaylist中的url列表转化为QJsonArray然后进一步转化为json文件存储下来。

在程序开始MainWindow初始化的时候调用void InitMediaList(QMediaPlaylist\* , QVector<OffMusic> &)。该函数将当前目录下的PlaylistRecord.json读入，并转化为QmeidaPlaylist中的ur表单，之后可以再由View部分读出QMediaPlaylist中的内容，显示出上一次退出程序时的歌单。



图表 2 PlaylistRecord UML图

## 实现过程

代码如下：

#ifndef PLAYLISTRECORD

#define PLAYLISTRECORD

#include <QJsonDocument>

#include <QJsonArray>

#include <QFile>

#include <QMediaPlaylist>

#include <QException>

#include "OffMusic.h"

//当程序结束调用同步失败时，抛出SyncPlaylistFail异常

class SyncPlaylistFail : public QException

{

public:

void *raise*() const { throw \*this; }

SyncPlaylistFail \**clone*() const { return new SyncPlaylistFail(\*this); }

};

//从json文件中读取url，载入QMediaPlaylist中

inline void InitMediaList(QMediaPlaylist\* playlist, QVector<OffMusic> &qvec)

{

//打开PlaylistRecord.json文件并转化为QJsonDocument

QFile file("PlaylistRecord.json");

QJsonDocument json;

if(file.*open*(QIODevice::ReadOnly | QIODevice::Text))

json = QJsonDocument::fromBinaryData(file.readAll());

//从QjsonDocument中获取QJsonArray，遍历得到上次存入的所有url

QJsonArray jsonarr = json.array();

for(int index = 0; index < jsonarr.size(); index++ )

{

QUrl url(jsonarr[index].toString());

//同时构造OffMusic返回给上层，给用户显示歌曲信息

OffMusic omus(url);

qvec.append(omus);

playlist->addMedia(url);

}

qDebug() <<"Init end"<< playlist->mediaCount();

}

//遍历QMediaPlaylist将url存储为json文件

inline void SyncMediaList(const QMediaPlaylist\* playlist)

{

QJsonArray jsonarr;

QJsonDocument json;

//遍历QMediaPlaylist，将url放入QJsonArray中

int cnt = playlist->mediaCount();

for(int index = 0; index < cnt; index++)

{

QUrl url(playlist->media(index).canonicalUrl());

jsonarr.append(url.toString());

}

//将QJsonArray转化为QJsonDocument

json.setArray(jsonarr);

//将QJsonDocumemt转化为QByteArray并写入PlaylistRecord.json文件

QByteArray byteArr = json.toBinaryData();

QFile file("PlaylistRecord.json");

if(file.*open*(QIODevice::WriteOnly | QIODevice::Text))

{

file.write(byteArr);

}

else

{

throw SyncPlaylistFail();

}

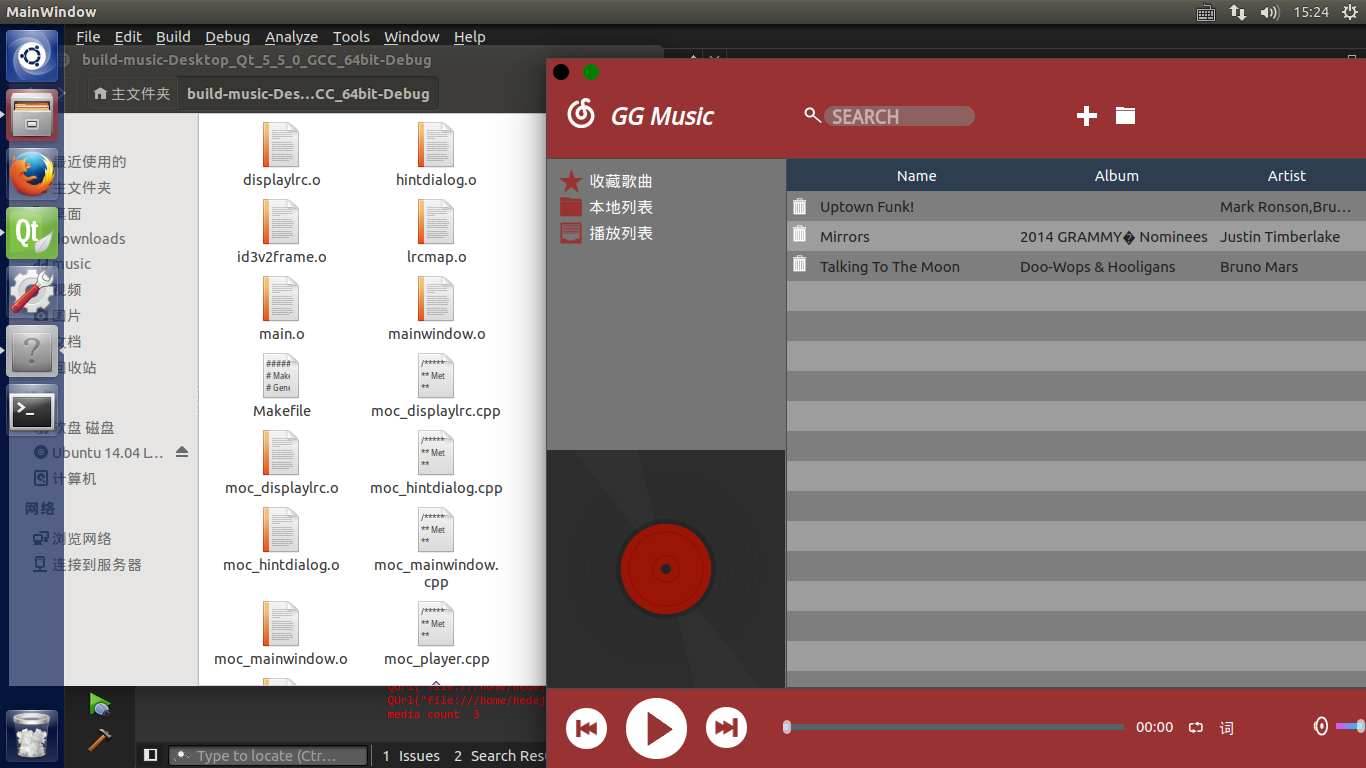
}

#endif // PLAYLISTRECORD

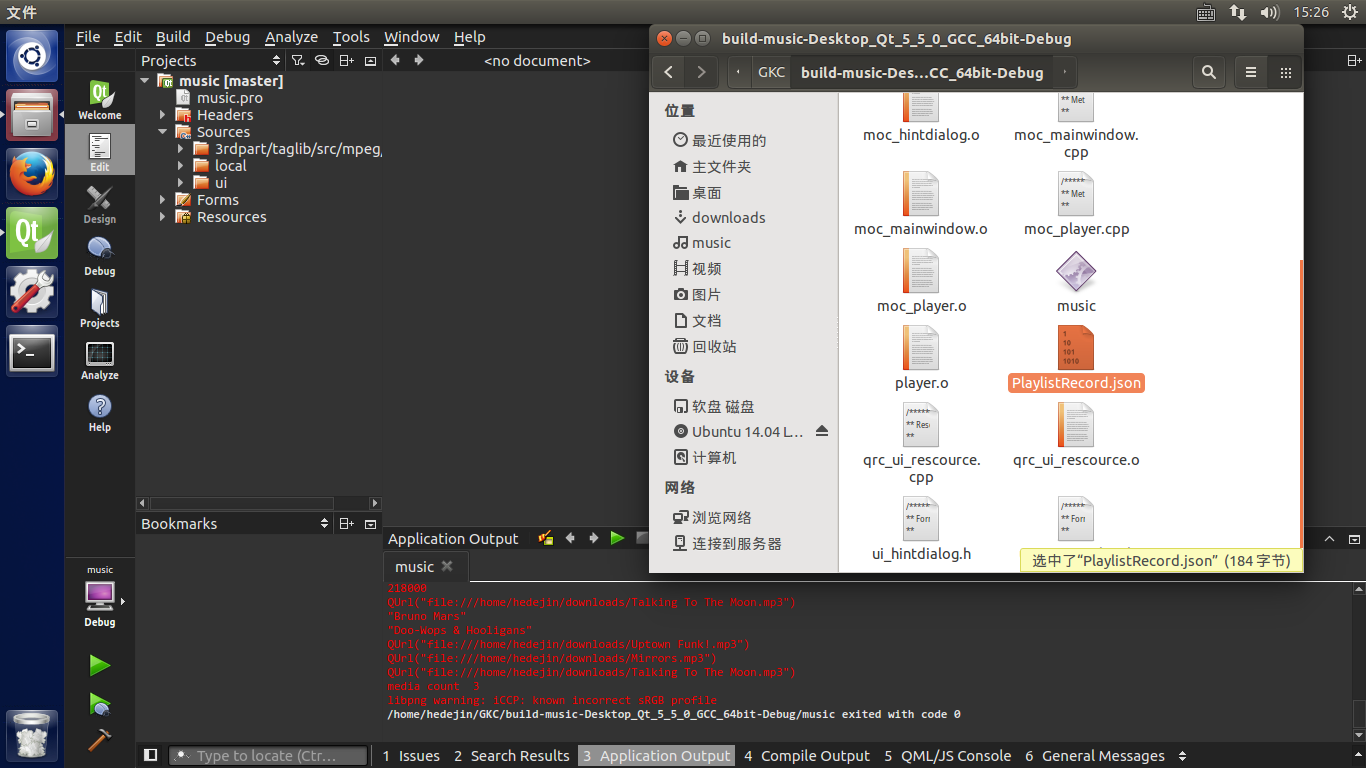
## 实现效果

每次点击窗口左上角的的退出标识进行安全退出，下一次打开程序时可以看到一模一样的歌单出现在本地列表之中。

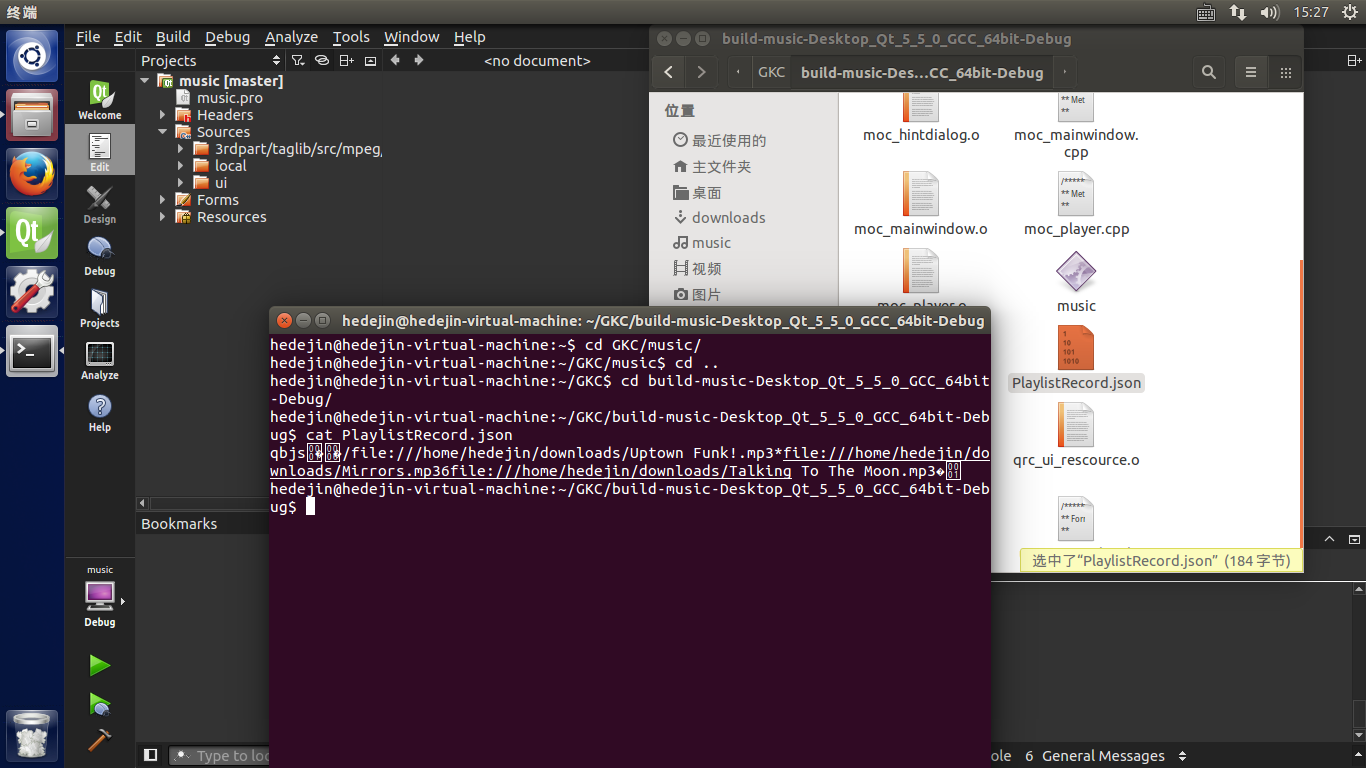
## 测试记录



假设程序第一次运行，对应的文件夹中不存在对应的PlaylistRecord.json文件。在添加三首歌后点击按钮退出程序。



则可以观察到生成的json文件。使用命令行可以观察到内部的部分信息。



再次打开程序。可以观察到三首歌已经被加载到本地列表之中。

