

前言

这一节实现一个简易的音乐播放器,其音乐播放的核心功能是采用 Qt 支持的 Phonon 框架,该框架在前一篇博文 [Qt 学习之路_13\(简易俄罗斯方块\)](#) 中已经使用过了,在俄罗斯方块中主要是用来设置背景音乐和消行的声音的。这里用这个框架同样也是用来播放,暂停等多媒体的各种控制功能,另外该框架可以自动获取音频文件的一些信息,这样我们在设计播放列表时可以获取这些信息,比如歌手名,专辑名,时长,文件名等等。程序中桌面歌词的实现是继承了 QLabel 类,然后使用 3 层文本显示,最上面一层采用渐进显示的方式来达到歌词播放的动态效果。

实验的参考资料为 <http://www.yafeilinux.com/> 网站上 yafei 作者提供的代码,本人只是看懂其源码然后自己敲了一遍,代码做了稍微的改变,其设计方法和技巧全是原创作者 yafei 的功劳。

开发环境: WindowsXP+Qt4.8.2+QtCreator2.5.1

实验说明

本实验没有使用 QtDesigner 来设计界面,其界面而是直接采用 c++ 代码来写的。下面分以下几个方面来介绍本实验的实现过程中应该注意的知识点:

播放界面设计部分:

因为主界面的设计是从 QWidget 类继承而来,但是本程序却没有使用界面设计工具来设计界面,而是直接使用 c++ 代码完成。

在界面设计时,首先一般是设置窗口的标题,尺寸,图标等。然后然后本程序时在主界面上面添加了 2 个工具栏和一个标题栏,这 3 个栏目构成了播放器的主界面,主界面采用的是垂直布局,即 QVBoxLayout. 2 个工具栏分别为 QAction, 里面可以使用 addAction () 方法直接插入 action 或者使用 addWidget() 方法插入 widget. 对 action 可以设置其快捷键,提示文本,图标,响应槽函数等。对于 widget 可以设置其显示内容,提示文本,尺寸属性,对其方式,如果外加网络连接,则也可以设置其是否链接到外部等。

在播放媒体文件时,媒体对象 MediaObject 会在指定的时间间隔发送 tick() 信号,这个时间间隔可以使用 setTrickInterval() 函数来进行设置。tick() 中的参数 time 指定了媒体对象在媒体流中的当前时间位置,单位是毫秒。程序中关联了这个信号,其主要目的是为了获得当前的播放时间。

可以直接调用媒体播放文件的 totalTime 方法实现统计媒体文件的总播放时长,单位为毫秒,然后可以将其转换保存在 QTime 对象中,直接使用 toString() 函数来指定其形式。

媒体对象的各种状态:

当创建了媒体对象后,它就会处于 LoadingState 状态,只有使用 createPath() 为其设置了 Path,再使用 setCurrentSource() 为其设置了当前媒体源以后,媒体对象才会进入 StoppedState 状态。如果在设置了媒体源之后立即调用了 play() 函数,那么媒体对象就不会进入 StoppedState 状态了,而是直接进入 PlayingState 状态。

每当媒体对象的状态发生改变时，就会自动发射 `stateChanged()` 信号,这里绑定信号后，就可以用这些状态来进行一些有关的设置。

播放列表：

程序中 `sources` 为打开的所以音频文件列表, `playlist` 为音乐播放列表表格对象。程序中并没有直接使用 `meidaObject` 对象来获取音频文件信息，而是创建了新的 `MedioObject` 类对象 `meta_information_resolver` 作为元数据的解析器。因为只有在 `LoadingState` 完成后才能获得元数据，所以可以先调用解析器的 `setCurrentSource()` 函数为其设置一个媒体源，然后关联它的 `stateChanged()` 信号，等其进入到 `StoppedState` 状态再进行元数据的解析。

桌面歌词：

程序中实现桌面歌词设计是类 `MyLrc`，继承 `QLabel` 类。桌面歌词的显示首先需要将部件的背景设置为透明色，然后重新实现其重绘事件处理函数来自定义文本的显示，这里可以使用渐变填充来实现多彩的文字。然后再使用定时器，在已经绘制的歌词上面再绘制一个不断变宽的相同的歌词来实现歌词的动态播放效果。因此程序中的歌词共绘制了 3 遍，第一遍是深黑色，在最底层；第 2 遍是渐变填充的歌词，为正常显示所用；第 3 次绘制的是用于遮罩用，实现动态效果。

歌词的解析都在 `resolve_lrc()` 函数中实现的，利用正则表达式来获取歌曲文件中的各种信息，一般的歌词文件以 `.lrc` 后缀结尾，歌词文件的格式如下所示：

```
1
2 [ti:当爱已成往事]
3 [ar:张国荣]
4 [al:宠爱]
5 [by:fyes]
6 [00:00.00]当爱已成往事 张国荣
7 [00:46.00]往事不要再提
8 [00:49.00]人生已多风雨
9 [00:53.00]纵然记忆抹不去
10 [00:55.00]爱与恨都还在心里
11 [01:00.00]真的要断了过去
12 [01:04.00]让明天好好继续
13 [01:08.00]你就不要再苦苦追问我的消息
14 [01:15.00]爱情它是个难题
15 [01:19.00]让人目眩神迷
16 [01:23.00]忘了痛或许可以
17 [01:25.00]忘了你却太不容易
18 [01:31.00]你不曾真的离去
19 [01:34.00]你始终在我心里
20 [01:38.00]我对你仍有爱意
21 [01:40.00]我对自己无能为力
22 [03:15.00][01:45.00]因为我仍有梦
23 [03:19.00][01:49.00]依然将你放在我心中
24 [03:23.00][01:53.00]总是容易被往事打动
25 [03:27.00][01:57.00]总是为了你心痛
26 [03:30.00][02:00.00]别留恋岁月中
27 [03:34.00][02:04.00]我无意的柔情万种
28 [03:38.00][02:08.00]不要问我是否再相逢
```

关于歌词的解析部分详见代码部分。

系统图标的设计：

一般的音乐播放器都会有一个系统托盘图标，这样就可以在播放歌曲的时候将主界面最小化到系统托盘图标了。Qt 中是通过 `QSystemTrayIcon` 类来实现系统托盘图标的，并且可以很容易在该图标上添加菜单，设置工具栏提示，显示消息和处理各种交互等。

知识点总结

Qt 知识点总结：

`QAction` 对象使用 `setText()` 方法时，如果在对象的构造函数中已经有了其文字显示，那么 `action` 上面显示的就是构造函数中的 `text` 文本。这里的 `setText` 文本有 2 个作用，第一个是如果该 `action` 对应到了菜单栏中，则菜单栏会自动将其显示出来；第二个时如果构造函数中没

有设置文本内容，则该 `action` 会显示 `setText()` 方法设置的内容，当然了，如果 `action` 设置了图标，该文本内容就被覆盖了，退化为文本提示了。

`cellClicked(int, int)` 信号是当表格中的一个 `cell` 单元被单击时发出的。它的两个参数分别为表格中 `cell` 的行号和列号。

可以使用 `frameGeometry()` 来获得程序中的主界面，然后该界面的定位函数可以获得与主界面的相对位置，比如说 `frameGeometry().bottomLeft()` 就是获得主界面的左下方的位置。

当自己定义了一个类，该类有对应的头文件和源文件。如果在第二个类的头文件中要使用到第一个类，则可以不用包含第一个类的头文件，直接用 `class` 关键字声明就可以了，在第二个类的源文件中则需要包含第一个类的头文件，因为这里需要使用第一个类对象的成员方法。

Qt 中正则表达式为类 `QRegExp`，正则表达式是指一个用来描述或者匹配一系列符合某个句法规则的字符串的单个字符串。比如说程序中的 `QRegExp rx("\\[\\d{2}:\\d{2}\\]\\.");` 其实就是表示歌词文件前面的格式，比如 `[00:05.54]`。表达式中的 `d{2}` 表示匹配 2 个数字。

Qt 中常见的类的继承总结：

如果需要设计界面，且需要菜单栏，工具栏，状态栏等，一般继承 `QMainWindow` 类。

如果需要界面，不需要菜单栏，工具栏，状态栏等，一般继承 `QDialog` 类。

如果需要使用自定义视图来画图形，则可以继承 `QAbstractItem` 类。

如果需要自己设计场景，比如游戏开发的时候，可以继承 `QGraphicsView` 类。

如果需要自己制作一个小图形视图，可以考虑继承 `QGraphicsObject` 类，当将这些小视图构成一个视图组时，该组的类可以继承 `QGraphicsItemGroup` 类和 `QObject` 类。

一般的界面设计也可以继承 `QWidget` 类。

一般的文本类可以继承 `QLabel`，比如本实验的桌面歌词类 `MyLrc`。

实验结果

该实验有打开播放文件，播放按钮，暂停按钮，选择上一首歌按钮，选择下一首歌按钮，显示播放列表，单击播放列表实现歌曲播放，动态显示桌面歌词，显示歌曲总时长和已播放时长，调节音乐音量，最小化到系统托盘等功能，其截图效果如下所示：



实验主要部分代码及注释（附录有工程 **code** 下载链接）：

mywidget.h:

```
#ifndef MYWIDGET_H
#define MYWIDGET_H

#include <QWidget>
#include <Phonon>
```

```

#include <QSystemTrayIcon>

class QLabel;
class MyPlaylist;
class MyLrc;

namespace Ui {
class MyWidget;
}

class MyWidget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::MyWidget *ui;
    void InitPlayer();
    Phonon::MediaObject *media_object;
    QAction *play_action;
    QAction *stop_action;
    QAction *skip_backward_action;
    QAction *skip_forward_action;
    QLabel *top_label;
    QLabel *time_label;

    MyPlaylist *playlist;
    Phonon::MediaObject *meta_information_resolver;
    QList<Phonon::MediaSource> sources;
    void change_action_state();

    MyLrc *lrc;
    QMap<qint64, QString> lrc_map;
    void resolve_lrc(const QString &source_file_name);

    QSystemTrayIcon *tray_icon;

private slots:
    void UpdateTime(qint64 time);
    void SetPaused();
    void SkipBackward();

```

```

void SkipForward();
void OpenFile();
void SetPlayListShown();
void SetLrcShown();

void StateChanged(Phonon::State new_state, Phonon::State old_state);
void SourceChanged(const Phonon::MediaSource &source);
void AboutToFinish();
void MetaStateChanged(Phonon::State new_state, Phonon::State old_state);
void TableClicked(int row);
void ClearSources();

void TrayIconActivated(QSystemTrayIcon::ActivationReason
activation_reason);

protected:
    void closeEvent(QCloseEvent *);

};

```

```
#endif // MYWIDGET_H
```



mywidget.cpp:



```

#include "mywidget.h"
#include "ui_mywidget.h"
#include "myplaylist.h"
#include "mylrc.h"
#include <QLabel>
#include <QToolBar>
#include <QVBoxLayout>
#include <QTime>
#include <QMessageBox>
#include <QFileInfo>
#include <QFileDialog>
#include <QDesktopServices>
#include <QTextCodec>
#include <QMenu>
#include <QCloseEvent>

MyWidget::MyWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::MyWidget)

```

```

{
    ui->setupUi(this);
    InitPlayer();
}

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

//初始化播放器
void MyWidget::InitPlayer()
{
    //设置窗口基本属性
    setWindowTitle(tr("MyPlayer 音乐播放器"));
    setWindowIcon(QIcon(":/images/icon.png")); //从资源文件中招图标
    setMinimumSize(320, 160);
    setMaximumSize(320, 160); //最大最小设置为一样，代表不改变播放器窗口的大小

    //创建媒体对象
    media_object = new Phonon::MediaObject(this);
    Phonon::AudioOutput *audio_output = new
Phonon::AudioOutput(Phonon::MusicCategory, this);
    Phonon::createPath(media_object, audio_output); //绑定源和接收器

    //关联媒体对象的 tick 信号来更新播放时间的显示
    connect(media_object, SIGNAL(tick(qint64)), this,
SLOT(UpdateTime(qint64)));

    //创建顶部标签
    top_label = new QLabel(tr("<a
href=\"http://www.cnblogs.com/tornadomeet/\">http://www.cnblogs.com/tornadom
eet/</a>"));
    top_label->setTextFormat(Qt::RichText);
    top_label->setOpenExternalLinks(true); //运行点击进入外部链接
    top_label->setAlignment(Qt::AlignCenter);

    //创建控制播放进度的滑块
    Phonon::SeekSlider *seek_slider = new Phonon::SeekSlider(media_object,
this);

    //设置显示时间的标签
    QToolBar *widget_bar = new QToolBar(this);
    time_label = new QLabel(tr("00:00/00:00"), this);

```



```

time_label->setToolTip(tr("当前时间/总时间"));
time_label->setAlignment(Qt::AlignCenter);
//QSizePolicy 类是描述水平和垂直修改大小策略的一种属性
time_label->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);//水
平方向上尺寸可扩展，水平方向已固定

//播放列表开启控制图标
QAction *PLAction = new QAction(tr("PL"), this);
PLAction->setShortcut(QKeySequence("F4")); //设置开启播放列表的快捷键为 F4
PLAction->setToolTip(tr("播放列表 (F4)"));
connect(PLAction, SIGNAL(triggered()), this, SLOT(SetPlayListShown())); //
链接触发信号

//桌面歌词显示开启控制图标
QAction *LRCAction = new QAction(tr("LRC"), this);
LRCAction->setShortcut(QKeySequence("F2")); //设置开启桌面歌词的播放列表快捷键为
F2
LRCAction->setToolTip(tr("桌面歌词 (F2)"));
connect(LRCAction, SIGNAL(triggered()), this, SLOT(SetLrcShown()));

//将上面 2 个 action 和 1 个 widget 添加到工具栏，默认的添加方式为水平方向添加
widget_bar->addAction(PLAction);
widget_bar->addSeparator();
widget_bar->addWidget(time_label);
widget_bar->addSeparator();
widget_bar->addAction(LRCAction);
widget_bar->addSeparator();

//设置播放动作
QToolBar *tool_bar = new QToolBar(this); //该构造函数没有写入文字
play_action = new QAction(this);
play_action->setIcon(QIcon(":/images/play.png"));
play_action->setText(tr("播放 (F5)"));
play_action->setShortcut(QKeySequence("F5")); //播放的快捷键位 F5
connect(play_action, SIGNAL(triggered()), this, SLOT(SetPaused()));

//设置停止动作
stop_action = new QAction(this);
stop_action->setIcon(QIcon(":/images/stop.png"));
stop_action->setText(tr("停止 (F6)"));
stop_action->setShortcut(QKeySequence("F6"));
connect(stop_action, SIGNAL(triggered()), this, SLOT(stop()));

//设置上一首动作

```

```

skip_backward_action = new QAction(this);
skip_backward_action->setIcon(QIcon(":/images/skipBackward.png"));
skip_backward_action->setText(tr("上一首(Ctrl+Left)"));
skip_backward_action->setShortcut(QKeySequence("Ctrl+Left"));
connect(skip_backward_action, SIGNAL(triggered()), this,
SLOT(SkipBackward()));

//设置下一首动作
skip_forward_action = new QAction(this);
skip_forward_action->setIcon(QIcon(":/images/skipForward.png"));
skip_forward_action->setText(tr("下一首(Ctrl+Right)"));
skip_forward_action->setShortcut(QKeySequence("Ctrl+Right"));
connect(skip_forward_action, SIGNAL(triggered()), this,
SLOT(SkipForward()));

//设置打开文件动作
QAction *open_action = new QAction(this);
open_action->setIcon(QIcon(":/images/open.png"));
open_action->setText(tr("播放文件(Ctrl+O)"));
open_action->setShortcut(QKeySequence("Ctrl+O"));
open_action->setEnabled(true);
connect(open_action, SIGNAL(triggered()), this, SLOT(OpenFile()));

//音乐控制部件
Phonon::VolumeSlider *volume_slider = new Phonon::VolumeSlider(audio_output,
this);
volume_slider->setSizePolicy(QSizePolicy::Maximum, QSizePolicy::Maximum);

//将以上部件添加到工具栏
tool_bar->addAction(play_action);
tool_bar->addSeparator();
tool_bar->addAction(stop_action);
tool_bar->addSeparator();
tool_bar->addAction(skip_backward_action);
tool_bar->addSeparator();
tool_bar->addAction(skip_forward_action);
tool_bar->addSeparator();
tool_bar->addWidget(volume_slider);
tool_bar->addSeparator();
tool_bar->addAction(open_action);

//创建主界面管理器
QVBoxLayout *main_layout = new QVBoxLayout;
main_layout->addWidget(top_label);

```

```

main_layout->addWidget(seek_slider);
main_layout->addWidget(widget_bar);
main_layout->addWidget(tool_bar);
setLayout(main_layout);

// //设置媒体部件音乐源
// media_object->setCurrentSource(Phonon::MediaSource("./music.mp3"));

//每当媒体对象的状态发生改变时,就会自动发射 stateChanged() 信号,这里绑定信号后,就可以
用这些状态来进行一些有关的设置
connect(media_object, SIGNAL(stateChanged(Phonon::State, Phonon::State)),
        this, SLOT(StateChanged(Phonon::State, Phonon::State)));

playlist = new MyPlaylist(this);
//cellClicked() 信号是当表格中的一个 cell 单元被单击时发出的。
connect(playlist, SIGNAL(cellClicked(int,int)), this,
SLOT(TableClicked(int)));
connect(playlist, SIGNAL(play_list_clean()), this, SLOT(ClearSources()));

meta_information_resolver = new Phonon::MediaObject(this);
Phonon::AudioOutput *meta_information_audio_output =
    new Phonon::AudioOutput(Phonon::MusicCategory, this);
Phonon::createPath(meta_information_resolver,
meta_information_audio_output);
connect(meta_information_resolver,
SIGNAL(stateChanged(Phonon::State, Phonon::State)),
        this, SLOT(MetaStateChanged(Phonon::State, Phonon::State)));
connect(media_object, SIGNAL(currentSourceChanged(Phonon::MediaSource)),
        this, SLOT(SourceChanged(Phonon::MediaSource)));
connect(media_object, SIGNAL(aboutToFinish()), media_object,
SLOT(AboutToFinish()));
play_action->setEnabled(false);
stop_action->setEnabled(false);
skip_forward_action->setEnabled(false);
skip_backward_action->setEnabled(false);
top_label->setFocus();

lrc = new MyLrc(this);

// 创建系统托盘图标
tray_icon = new QSystemTrayIcon(QIcon(":/images/icon.png"), this);
tray_icon->setToolTip(tr("简易音乐播放器"));
// 创建菜单,系统托盘图标后右击出现的菜单

```

```

QMenu *menu = new QMenu;
QList<QAction *> actions;
actions << play_action << stop_action << skip_backward_action <<
skip_forward_action;
menu->addActions(actions);
menu->addSeparator();
menu->addAction(PLAction);
menu->addAction(LRCAction);
menu->addSeparator();
menu->addAction(tr("退出"), qApp, SLOT(quit()));
tray_icon->setContextMenu(menu);
// 托盘图标被激活后进行处理
connect(tray_icon, SIGNAL(activated(QSystemTrayIcon::ActivationReason)),
        this, SLOT(TrayIconActivated(QSystemTrayIcon::ActivationReason)));
// 显示托盘图标
tray_icon->show();
}

// 根据媒体源列表内容和当前媒体源的位置来改变主界面图标的状态
void MyWidget::change_action_state()
{
    // 如果媒体源列表为空
    if (sources.count() == 0) {
        // 如果没有在播放歌曲，则播放和停止按钮都不可用
        // （因为可能歌曲正在播放时清除了播放列表）
        if (media_object->state() != Phonon::PlayingState &&
            media_object->state() != Phonon::PausedState) {
            play_action->setEnabled(false);
            stop_action->setEnabled(false);
        }
        skip_backward_action->setEnabled(false);
        skip_forward_action->setEnabled(false);
    }
    else { // 如果媒体源列表不为空
        play_action->setEnabled(true);
        stop_action->setEnabled(true);
        // 如果媒体源列表只有一行
        if (sources.count() == 1) {
            skip_backward_action->setEnabled(false);
            skip_forward_action->setEnabled(false);
        } else { // 如果媒体源列表有多行
            skip_backward_action->setEnabled(true);
            skip_forward_action->setEnabled(true);
        }
    }
}

```

```

        int index = playlist->currentRow();
        // 如果播放列表当前选中的行为第一行
        if (index == 0)
            skip_backward_action->setEnabled(false);
        // 如果播放列表当前选中的行为最后一行
        if (index + 1 == sources.count())
            skip_forward_action->setEnabled(false);
    }
}
}

```

// 解析 LRC 歌词，在 stateChanged() 函数的 Phonon::PlayingState 处和 aboutToFinish() 函数中调用了该函数

```

void MyWidget::resolve_lrc(const QString &source_file_name)
{
    lrc_map.clear();
    if(source_file_name.isEmpty())
        return;
    QString file_name = source_file_name;
    QString lrc_file_name = file_name.remove(file_name.right(3)) + "lrc"; //把音频文件的后缀改成 lrc 后缀

    // 打开歌词文件
    QFile file(lrc_file_name);
    if (!file.open(QIODevice::ReadOnly)) {

lrc->setText(QFileInfo(media_object->currentSource().fileName()).baseName()
            + tr(" --- 未找到歌词文件! "));
        return ;
    }
    // 设置字符串编码
    QTextCodec::setCodecForCStrings(QTextCodec::codecForLocale());
    QString all_text = QString(file.readAll());
    file.close();
    // 将歌词按行分解为歌词列表
    QStringList lines = all_text.split("\n");

    //这个是时间标签的格式[00:05.54]
    //正则表达式 d{2}表示匹配 2 个数字
    QRegExp rx("\\[\\d{2}:\\d{2}\\.\\d{2}\\]");
    foreach(QString oneline, lines) {
        QString temp = oneline;
    }
}

```

本

```
temp.replace(rx, ""); //用空字符串替换正则表达式中所匹配的地方,这样就获得了歌词文本

// 然后依次获取当前行中的所有时间标签,并分别与歌词文本存入 QMap 中
//indexIn() 为返回第一个匹配的位置,如果返回为-1,则表示没有匹配成功
//正常情况下 pos 后面应该对应的是歌词文件
int pos = rx.indexIn(online, 0);
while (pos != -1) { //表示匹配成功
    QString cap = rx.cap(0); //返回第 0 个表达式匹配的内容
    // 将时间标签转换为时间数值,以毫秒为单位
    QRegExp regexp;
    regexp.setPattern("\\d{2}(?=:)");
    regexp.indexIn(cap);
    int minute = regexp.cap(0).toInt();
    regexp.setPattern("\\d{2}(?=\\.)");
    regexp.indexIn(cap);
    int second = regexp.cap(0).toInt();
    regexp.setPattern("\\d{2}(?=\\])");
    regexp.indexIn(cap);
    int millisecond = regexp.cap(0).toInt();
    qint64 totalTime = minute * 60000 + second * 1000 + millisecond * 10;
    // 插入到 lrc_map 中
    lrc_map.insert(totalTime, temp);
    pos += rx.matchedLength();
    pos = rx.indexIn(online, pos); //匹配全部
}
}

// 如果 lrc_map 为空
if (lrc_map.isEmpty()) {

lrc->setText(QFileInfo(media_object->currentSource().fileName()).baseName()
            + tr(" --- 歌词文件内容错误!"));

    return;
}
}

void MyWidget::UpdateTime(qint64 time)
{
    qint64 total_time_value = media_object->totalTime(); //直接获取该音频文件的总时长参数,单位为毫秒
    //这 3 个参数分别代表了时,分,秒; 60000 毫秒为 1 分钟,所以分钟第二个参数是先除 6000,第 3 个参数是直接除 1s
    QTime total_time(0, (total_time_value/60000)%60,
                    (total_time_value/1000)%60);
```

```

    QTime current_time(0, (time/60000)%60, (time/1000)%60); //传进来的 time 参数代
    表了当前的时间
    QString str = current_time.toString("mm:ss") + "/" +
total_time.toString("mm:ss");
    time_label->setText(str);

    // 获取当期时间对应的歌词
    if(!lrc_map.isEmpty()) {
        // 获取当前时间在歌词中的前后两个时间点
        qint64 previous = 0;
        qint64 later = 0;
        //keys() 方法返回 lrc_map 列表
        foreach (qint64 value, lrc_map.keys()) {
            if (time >= value) {
                previous = value;
            } else {
                later = value;
                break;
            }
        }

        // 达到最后一行,将 later 设置为歌曲总时间的值
        if (later == 0)
            later = total_time_value;

        // 获取当前时间所对应的歌词内容
        QString current_lrc = lrc_map.value(previous);

        //      // 没有内容时
        //      if(current_lrc.length() < 2)
        //          current_lrc = tr("简易音乐播放器");

        // 如果是新的一行歌词,那么重新开始显示歌词遮罩
        if(current_lrc != lrc->text()) {
            lrc->setText(current_lrc);
            top_label->setText(current_lrc);
            qint64 interval_time = later - previous;
            lrc->start_lrc_mask(interval_time);
        }
    } else { // 如果没有歌词文件,则在顶部标签中显示歌曲标题
        top_label->setText(QFileInfo(media_object->
                                                                    currentSource().fileName()).baseName());
    }
}

```

```

void MyWidget::SetPaused()
{
    if(media_object->state() == Phonon::PlayingState) {
        media_object->pause();
    }
    else
        media_object->play();
}

//播放上一首歌曲
void MyWidget::SkipBackward()
{
    lrc->stop_lrc_mask();
    int index = sources.indexOf(media_object->currentSource());
    media_object->setCurrentSource(sources.at(index - 1));
    media_object->play();
}

//播放下一首歌曲
void MyWidget::SkipForward()
{
    lrc->stop_lrc_mask();
    int index = sources.indexOf(media_object->currentSource());
    media_object->setCurrentSource(sources.at(index + 1));
    media_object->play();
}

void MyWidget::OpenFile()
{
    //可以同时打开多个音频文件
    QStringList list = QFileDialog::getOpenFileNames(this, tr("打开音乐文件"),

QDesktopServices::storageLocation(QDesktopServices::MusicLocation));
    if(list.isEmpty())
        return;
    //获取当前媒体源列表的大小
    int index = sources.size();
    foreach(QString string, list) {
        Phonon::MediaSource source(string);
        sources.append(source);
    }
}

```



```

    }
    if(!sources.isEmpty()) {
        //如果媒体源列表不为空，则将新加入的第一个媒体源作为当前媒体源
        meta_information_resolver->setCurrentSource(sources.at(index));
    }
}

void MyWidget::SetPlayListShown()
{
    if(playlist->isHidden()) {
        playlist->move(frameGeometry().bottomLeft()); //显示在主界面的下方
        playlist->show();
    }
    else {
        playlist->hide();
    }
}

void MyWidget::SetLrcShown()
{
    if(lrc->isHidden())
        lrc->show();
    else
        lrc->hide();
}

void MyWidget::StateChanged(Phonon::State new_state, Phonon::State old_state)
{
    switch(new_state)
    {
        //当新状态时错误状态时，如果是致命错误则显示警告致命错误消息框，否则显示普通错误消息框
        case Phonon::ErrorState:
            if(media_object->errorType() == Phonon::FatalError) {
                QMessageBox::warning(this, tr("致命错误"),
media_object->errorString()); //显示其错误的内容
            }
            else {
                QMessageBox::warning(this, tr("错误"),
media_object->errorString()); //显示普通错误
            }
            break;
    }
}

```

```

//当新状态为播放状态时,更改一些状态的控件
case Phonon::PlayingState:
    stop_action->setEnabled(true);
    play_action->setIcon(QIcon(":/images/pause.png"));
    play_action->setText(tr("暂停(F5)"));
    //更改第一行的标签内容为播放文件的文件名。注意 baseName 是在 QFileInfo 的后面

top_label->setText(QFileInfo(media_object->currentSource().fileName()).baseName());

    resolve_lrc(media_object->currentSource().fileName());
    break;
case Phonon::StoppedState:
    stop_action->setEnabled(false);
    play_action->setIcon(QIcon(":/images/play.png"));
    //setText 函数实现的功能感觉和 setToolTip 一样,只是这里设置过了的文本如果该
action 对应到菜单栏,则会显示出来
    play_action->setText(tr("播放(F5)"));
    top_label->setText(tr("<a
href=\"http://www.cnblogs.com/tornadomeet/\">http://www.cnblogs.com/tornadomeet/</a>"));
    time_label->setText(tr("00:00/00:00"));
    lrc->stop_lrc_mask();
    lrc->setText(tr("简易音乐播放器"));
    break;
case Phonon::PausedState:
    stop_action->setEnabled(true);
    play_action->setIcon(QIcon(":/images/play.png"));
    play_action->setText(tr("播放(F5)"));

top_label->setText(QFileInfo(media_object->currentSource().fileName()).baseName() + tr(" 已暂停!"));
    // 如果该歌曲有歌词文件
    if (!lrc_map.isEmpty()) {
        lrc->stop_lrc_mask();
        lrc->setText(top_label->text());
    }
    break;
case Phonon::BufferingState:
    break;
default:
    ;
}
}

```

```

//该槽函数是当媒体源发生改变时，触发 currentSourceChanged() 信号, 从而执行该槽函数
//该函数完成的功能是选中所改变的媒体源那一行
void MyWidget::SourceChanged(const Phonon::MediaSource &source)
{
    int index = sources.indexOf(source);
    playlist->selectRow(index);
    change_action_state();
}

//当媒体播放快结束时，会发送 aboutToFinish() 信号，从而触发该槽函数
void MyWidget::AboutToFinish()
{
    int index = sources.indexOf(media_object->currentSource())+1;
    if(sources.size() > index) {
        media_object->enqueue(sources.at(index)); //将下一首歌曲添加到播放列表中
        media_object->seek(media_object->totalTime()); //跳到当前歌曲的最后
        lrc->stop_lrc_mask();
        resolve_lrc(sources.at(index).fileName());
    }
    else {
        media_object->stop(); //如果已经是打开音频文件的最后一首歌了，就直接停止
    }
}

void MyWidget::MetaStateChanged(Phonon::State new_state, Phonon::State
old_state)
{
    // 错误状态，则从媒体源列表中除去新添加的媒体源
    if(new_state == Phonon::ErrorState) {
        QMessageBox::warning(this, tr("打开文件时出错"),
meta_information_resolver->errorString());
        //takeLast() 为删除最后一行并将其返回
        while (!sources.isEmpty() &&
            !(sources.takeLast() ==
meta_information_resolver->currentSource()))
        {};//只留下最后一行
        return;
    }
    // 如果既不处于停止状态也不处于暂停状态，则直接返回
    if(new_state != Phonon::StoppedState && new_state != Phonon::PausedState)

```

```

        return;
    // 如果媒体源类型错误, 则直接返回
    if(meta_information_resolver->currentSource().type() ==
Phonon::MediaSource::Invalid)
        return;

    QMap<QString, QString> meta_data = meta_information_resolver->metaData(); //
获取媒体源中的源数据

    //获取文件标题信息
    QString title = meta_data.value("TITLE");
    //如果媒体元数据中没有标题信息, 则去该音频文件的文件名为该标题信息
    if(title == "") {
        QString str = meta_information_resolver->currentSource().fileName();
        title = QFileInfo(str).baseName();
    }
    QTableWidgetItem *title_item = new QTableWidgetItem(title);
    title_item->setFlags(title_item->flags() ^ Qt::ItemIsEditable);

    //获取艺术家信息
    QTableWidgetItem *artist_item = new
QTableWidgetItem(meta_data.value("ARTIST"));
    artist_item->setFlags(artist_item->flags() ^ Qt::ItemIsEditable);

    //获取总时间信息
    qint64 total_time = meta_information_resolver->totalTime();
    QTime time(0, (total_time/60000)%60, (total_time/10000)%60);
    QTableWidgetItem *time_item = new QTableWidgetItem(time.toString("mm:ss"));

    //插入播放列表
    int current_rows = playlist->rowCount(); //返回列表中的行数
    playlist->insertRow(current_rows);
    playlist->setItem(current_rows, 0, title_item);
    playlist->setItem(current_rows, 1, artist_item);
    playlist->setItem(current_rows, 2, time_item);

    //sources 为打开的所以音频文件列表, playlist 为音乐播放列表表格对象
    int index = sources.indexOf(meta_information_resolver->currentSource())+1;
    if(sources.size() > index) //没有解析完
        meta_information_resolver->setCurrentSource(sources.at(index));
    else {
        //没有被选中的行
        if(playlist->selectedItems().isEmpty()) {
            // 如果现在没有播放歌曲则设置第一个媒体源为媒体对象的当前媒体源

```

```

        // (因为可能正在播放歌曲时清空了播放列表, 然后又添加了新的列表)
        if(media_object->state() != Phonon::PlayingState &&
media_object->state() != Phonon::PausedState)
            media_object->setCurrentSource(sources.at(0));
        else {
            //如果正在播放歌曲, 则选中播放列表的第一个曲目, 并更改图标状态
            playlist->selectRow(0);
            change_action_state();
        }
    }
    else {
        // 如果播放列表中有选中的行, 那么直接更新图标状态
        change_action_state();
    }
}

}

void MyWidget::TableClicked(int row)
{
    bool was_palying = media_object->state() == Phonon::PlayingState;
    media_object->stop(); //停止当前播放的歌曲
    media_object->clearQueue(); //清楚播放队列

    //如果单就的播放列表行号比媒体源中列表行号还打, 则直接返回
    if(row >= sources.size())
        return;
    media_object->setCurrentSource(sources.at(row));
    if(was_palying) //如果选中前在播放歌曲, 那么选中后也继续播放歌曲
        media_object->play();
}

void MyWidget::ClearSources()
{
    sources.clear();
    change_action_state();
}

//系统托盘图标被激活

```

```


void MyWidget::TrayIconActivated(QSystemTrayIcon::ActivationReason
activation_reason)
{
    if(activation_reason == QSystemTrayIcon::Trigger)
        show();
}

//关闭事件处理函数
void MyWidget::closeEvent(QCloseEvent *event)
{
    if(isVisible()) {
        hide();//单击关闭时，软件并没有关闭起来，而是隐藏在系统图标上
        tray_icon->showMessage(tr("简易音乐播放器"), tr("单击我重新回到主界面"));//
        这个提示有点像软件更新提示
        event->ignore();//不发送关闭信号
    }
}

```



myplaylist.h:



```

#ifndef MYPLAYLIST_H
#define MYPLAYLIST_H

#include <QTableWidget>

class MyPlaylist : public QTableWidget
{
    Q_OBJECT
public:
    explicit MyPlaylist(QWidget *parent = 0);

signals:
    void play_list_clean();

public slots:

protected:
    void contextMenuEvent(QContextMenuEvent *);
    void closeEvent(QCloseEvent *);

private slots:
    void clear_play_list();

```

```
};
```

```
#endif // MYPLAYLIST_H
```



myplaylist.cpp:



```
#include "myplaylist.h"
```

```
#include <QContextMenuEvent>
```

```
#include <QMenu>
```

```
MyPlaylist::MyPlaylist(QWidget *parent) :
```

```
    QTableWidget(parent)
```

```
{
```

```
    setWindowTitle(tr("播放列表"));
```

```
    //设置为一个独立的窗口，且只有一个关闭按钮
```

```
    setWindowFlags(Qt::Window | Qt::WindowTitleHint);
```

```
    resize(400, 400);
```

```
    setMaximumWidth(400);
```

```
    setMinimumWidth(400); //固定窗口大小
```

```
    setRowCount(0); //初始的行数为 0
```

```
    setColumnCount(3); //初始的列数为 1
```

```
    //设置第一个标签
```

```
    QStringList list;
```

```
    list << tr("标题") << tr("歌手") << tr("长度");
```

```
    setHorizontalHeaderLabels(list);
```

```
    setSelectionMode(QAbstractItemView::SingleSelection); //设置只能选择单行
```

```
    setSelectionBehavior(QAbstractItemView::SelectRows);
```

```
    setShowGrid(false); //设置不显示网格
```

```
}
```

```
void MyPlaylist::clear_play_list()
```

```
{
```

```
    while(rowCount())
```

```
        removeRow(0);
```

```
    emit play_list_clean(); //删除完后，发送清空成功信号
```

```

}


void MyPlaylist::contextMenuEvent(QContextMenuEvent *event)
{
    QMenu menu;
    menu.addAction(tr("清空列表"), this, SLOT(clear_play_list())); //可以直接在这里
指定槽函数
    menu.exec(event->globalPos()); //返回鼠标指针的全局位置
}

void MyPlaylist::closeEvent(QCloseEvent *event)
{
    if(isVisible()) {
        hide();
        event->ignore(); //清零接收标志
    }
}

```



mylrc.h:



```

#ifndef MYLRC_H
#define MYLRC_H

#include <QLabel>
class QTimer;

class MyLrc : public QLabel
{
    Q_OBJECT
public:
    explicit MyLrc(QWidget *parent = 0);
    void start_lrc_mask(qint64 intervaltime);
    void stop_lrc_mask();
protected:
    void paintEvent(QPaintEvent *);
    void mousePressEvent(QMouseEvent *ev);
    void mouseMoveEvent(QMouseEvent *ev);
    void contextMenuEvent(QContextMenuEvent *ev);

signals:

```



```

public slots:

private slots:
    void timeout();

private:
    QLinearGradient linear_gradient;
    QLinearGradient mask_linear_gradient;
    QFont font;
    QTimer *timer;
    qreal lrc_mask_width;

    qreal lrc_mask_width_interval;
    QPoint offset;

};

#endif // MYLRC_H

```



mylrc.cpp:



```

#include "mylrc.h"
#include <QPainter>
#include <QTimer>
#include <QMouseEvent>
#include <QContextMenuEvent>
#include <QMenu>

MyLrc::MyLrc(QWidget *parent) :
    QLabel(parent)
{
    //FramelessWindowHint 为无边界的窗口
    setWindowFlags(Qt::Window | Qt::FramelessWindowHint);
    setAttribute(Qt::WA_TranslucentBackground);
    setText(tr("简易音乐播放器"));
    // 固定显示区域大小
    setMaximumSize(800, 60);
    setMinimumSize(800, 60);

    //歌词的线性渐变填充
    linear_gradient.setStart(0, 10); //填充的起点坐标

```

```

linear_gradient.setFinalStop(0, 40); //填充的终点坐标
//第一个参数终点坐标，相对于我们上面的区域而言，按照比例进行计算
linear_gradient.setColorAt(0.1, QColor(14, 179, 255));
linear_gradient.setColorAt(0.5, QColor(114, 232, 255));
linear_gradient.setColorAt(0.9, QColor(14, 179, 255));

// 遮罩的线性渐变填充
mask_linear_gradient.setStart(0, 10);
mask_linear_gradient.setFinalStop(0, 40);
mask_linear_gradient.setColorAt(0.1, QColor(222, 54, 4));
mask_linear_gradient.setColorAt(0.5, QColor(255, 72, 16));
mask_linear_gradient.setColorAt(0.9, QColor(222, 54, 4));

// 设置字体
font.setFamily("Times New Roman");
font.setBold(true);
font.setPointSize(30);

// 设置定时器
timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(timeout()));
lrc_mask_width = 0;
lrc_mask_width_interval = 0;
}

// 开启遮罩，需要指定当前歌词开始与结束之间的时间间隔
void MyLrc::start_lrc_mask(qint64 intervaltime)
{
    // 这里设置每隔 30 毫秒更新一次遮罩的宽度，因为如果更新太频繁
    // 会增加 CPU 占用率，而如果时间间隔太大，则动画效果就不流畅了
    qreal count = intervaltime / 30;
    // 获取遮罩每次需要增加的宽度，这里的 800 是部件的固定宽度
    lrc_mask_width_interval = 800 / count;
    lrc_mask_width = 0;
    timer->start(30);
}

void MyLrc::stop_lrc_mask()
{
    timer->stop();
    lrc_mask_width = 0;
    update();
}

```

```

void MyLrc::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    painter.setFont(font);

    // 先绘制底层文字，作为阴影，这样会使显示效果更加清晰，且更有质感
    painter.setPen(QColor(0, 0, 0, 200));
    painter.drawText(1, 1, 800, 60, Qt::AlignLeft, text()); //左对齐

    // 再在上面绘制渐变文字
    painter.setPen(QPen(linear_gradient, 0));
    painter.drawText(0, 0, 800, 60, Qt::AlignLeft, text());

    // 设置歌词遮罩
    painter.setPen(QPen(mask_linear_gradient, 0));
    painter.drawText(0, 0, lrc_mask_width, 60, Qt::AlignLeft, text());
}

//左击操作
void MyLrc::mousePressEvent(QMouseEvent *event)
{
    if (event->button() == Qt::LeftButton)
        offset = event->globalPos() - frameGeometry().topLeft();
}

void MyLrc::mouseMoveEvent(QMouseEvent *event)
{
    //移动鼠标到歌词上时，会显示手型
    //event->buttons() 返回鼠标点击的类型，分为左击，中击，右击
    //这里用与操作表示是左击
    if (event->buttons() & Qt::LeftButton) {
        setCursor(Qt::PointingHandCursor);
        //实现移动操作
        move(event->globalPos() - offset);
    }
}

//右击事件
void MyLrc::contextMenuEvent(QContextMenuEvent *event)
{

```

```

QMenu menu;

menu.addAction(tr("隐藏"), this, SLOT(hide()));
menu.exec(event->globalPos()); //globalPos() 为当前鼠标的位置坐标
}

void MyLrc::timeout()
{
    //每隔一段固定的时间笼罩的长度就增加一点
    lrc_mask_width += lrc_mask_width_interval;
    update(); //更新 widget，但是并不立即重绘，而是安排一个 Paint 事件，当返回主循环时由系统来重绘
}

```

main.cpp:

```

#include <QApplication>
#include <QTextCodec>
#include "mywidget.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QTextCodec::setCodecForTr(QTextCodec::codecForLocale());
    MyWidget w;
    w.show();

    return a.exec();
}

```

实验总结：

本次实验主要学习到了界面设计，Phonon 应用框架，文件解析，2D 绘图和系统图片等知识。

到此为止，《Qt 及 Qt Quick 开发实战精讲》中关于 Qt 部分的 5 个例子已初步学完了。从 6 月底接触 Qt 到现在，整个过程断断续续差不多 3 个月，感觉收获还是有一些，越来越喜欢 Qt 了。

参考资料:

<http://www.yafeilinux.com/>

附录: 实验工程 code 下载。