第十二章 多媒体

多媒体（Multimedia）是多种媒体的综合，一般包括文本，声音和图像等多种媒体形式。在计算机系统中，多媒体指组合两种或两种以上媒体的一种人机交互式信息交流和传播媒体。使用的媒体包括文字、图片、照片、声音、动画和影片，以及程式所提供的互动功能。

Qt 的多媒体模块提供了音频、视频、录音、摄像头拍照和录像等功能。本章将介绍 Qt 多媒体的功能和使用。

12.1 Qt 多媒体简介

Qt 从 4.4 版本开始提供的一套多媒体框架，提供多媒体回放的功能。在 Qt 4.6 中实现多媒体播放图形界面主要依赖 phonon 框架。phonon 最初是 一个 源于 KDE 的项目，为使用音频和视频的应用程序开发提供的一个框架。应用程序不用去管多媒体播放是通过什么实现的（如 gstreamer、xine），只需调用相应的接口就行，但这中间需要一个中转，被称为 backend。Qt 也是通过 phonon 来实现跨平台的多媒体播放。

从 Qt5 开始，Qt 就弃用了 phonon，直接使用 Qt Multimedia 模块。我们可以 Qt Multimedia 模块来提供的类实现跨平台的多媒体播放了。使用 Qt Multimedia 就不需要中转了，但是底层还是需要多媒体插件实现的。Qt 只是提供多媒体接口，播放多媒体实际上是通过多媒体插件实现的，我们不需要管这些插件是什么，Qt 在不同平台使用的多媒体插件不同。本章将会介绍如何在 Windows 和 Linux 安装多媒体插件，Mac 系统不考虑，编者条件有限！

Qt 多媒体模块提供了很多类，主要有 QMediaPlayer，QSound、QSoundEffect、QAudioOutput、QAudioInput、QAudioRecorder、QVideoWidget 等等。类太多了不一一作解释，可以直接复制名字到 Qt 的帮助文档里查看改解释。可以从名称大概了解它们是什么意思，具体类的使用直接看本章的例子。

想要在 Qt 里使用使用 Qt 多媒体模块，需要在 pro 项目文件里添加如下语句。

```
QT          += multimedia
```

注意：Qt 中的音乐播放器与视频播放器需要在 Ubuntu 里安装媒体解码器才能实现播放。

- Ubuntu16 / Ubuntu18，需要安装以下插件。播放音乐需要安装 Gst 解码插件。需要在终端输入如下指令，注意不要复制错误了，下面指令已经在 Ubuntu16/Ubuntu18 测试成功，如果读者 Ubuntu 没有配置网络这与源服务器，这些导致安装不成功与本教程无关，确实需要读者好好打下 Ubuntu 操作的基础了！

```
sudo apt-get install gstreamer1.0-plugins-base gstreamer1.0-plugins-bad
gstreamer1.0-plugins-good gstreamer1.0-plugins-ugly gstreamer1.0-pulseaudio
gstreamer1.0-libav
```

- Windows 需要安装如 LAVFilters 解码器，只需要百度 LAVFilters，找到 LAVFilters 官网下载此软件即可，当然本教程的资料会提供一份 LAVFilters 的安装包。点击页脚下方的程序下载链接跳转到下载本教程所有资料下载地址处，在顶层目录下。

12.2 音效文件播放

播放音效文件，比如简短的提示音（按键音等），可以使用 Qt 的 QSoundEffect 和 QSound 类来播放。

Qt 的 QSoundEffect 和 QSound 类主要区别是 QSound（异步方式播放）只能播放本地的 WAV 音效文件（WAV 音效文件是 PC 机上最为流行的声音文件格式，但其文件尺寸较大，多用于存储简短的声音片段，具有低延时性，不失真的特点），QSoundEffect 不仅可以播放网络文件，也可以播放本地音效文件，播放网络的文件一般使用到 QUrl 链接。

12.2.1 应用实例

本例目的：了解 QSound 类的使用。

例 13_button_sound，按钮音效测试（难度：一般）。项目路径为 Qt/2/13_button_sound。本例大体流程，通过点击一个按钮，然后使用 QSound 来播放音效文件，模仿按键按下的声音。

项目文件 13_button_sound.pro 文件第一行添加的代码部分如下。

13_button_sound.pro 编程后的代码

```
1  QT       += core gui multimedia
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
```

```
12
13  # You can also make your code fail to compile if it uses deprecated
APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      main.cpp \
20      mainwindow.cpp
21
22  HEADERS += \
23      mainwindow.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/$${TARGET}/bin
27  else: unix:!android: target.path = /opt/$${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
29
30  RESOURCES += \
31      src.qrc
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   13_button_sound
   * @brief         mainwindow.h
   * @author        Deng Zhimao
   * @email         1252699831@qq.com
   * @net           www.openedv.com
```

```
    * @date          2021-04-20

**********************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QSound>
6   #include <QPushButton>
7
8   class MainWindow : public QMainWindow
9   {
10      Q_OBJECT
11
12  public:
13      MainWindow(QWidget *parent = nullptr);
14      ~MainWindow();
15
16  private:
17      /* 按钮 */
18      QPushButton *pushButton;
19
20  private slots:
21      /* 按钮点击槽函数 */
22      void pushButtonClicked();
23
24  };
25  #endif // MAINWINDOW_H
26
```

头文件里主要是声明界面使用的一个按钮，及按钮槽函数。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*********************************************************************
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   13_button_sound
   * @brief        mainwindow.cpp
   * @author       Deng Zhimao
   * @email        1252699831@qq.com
   * @net          www.openedv.com
   * @date         2021-04-20
**********************************************************************/
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 设置主窗体的位置与大小 */
    this->setGeometry(0, 0, 800, 480);

    /* 实例化按钮 */
    pushButton = new QPushButton(this);

    /* 设置按钮文本 */
    pushButton->setText("按钮音效测试");

    /* 设置按钮的位置与大小 */
    pushButton->setGeometry(340, 220, 120, 40);

    /* 信号槽连接 */
    connect(pushButton, SIGNAL(clicked()),
            this, SLOT(pushButtonClicked()));
}

```

```
23  MainWindow::~MainWindow()
24  {
25  }
26
27  void MainWindow::pushButtonClicked()
28  {
29      /* 异步的方式播放 */
30      QSound::play(":/audio/bell.wav");
31  }
```

第 30 行，直接使用 QSound 的静态函数 play()播放，这种播放方式是异步的，可以多次点击按钮连续听到点击的声音。

12.2.1 程序运行效果

单击按钮后，可以听到播放 1 秒左右的叮咚声，用此方法来模拟单击按钮声音效果。

12.3 音乐播放器

QMediaPlayer 类是一个高级媒体播放类。它可以用来播放歌曲、电影和网络广播等内容。一般用于播放 mp3 和 mp4 等等媒体文件。QMediaPlayer 类常常与 QMediaPlaylist 类一起使用。可以很轻松的设计一个自己喜欢的音乐播放器与视频播放器。

QMediaPlayer 提供了很多信号，我们可以使用这些信号来完成音乐播放器的一系列操作，比如媒体状态改变的信号 stateChanged(QMediaPlayer::State state)，判断这个 state 的状态就可以知道什么时候媒体暂停、播放、停止了。Qt 在媒体播放类已经提供了很多功能函数给我们使用，像直接使用 play()函数就可以实现音乐文件的播放，前提我们需要知道媒体文件的路径。pause()函数可以直接暂停媒体播放等等，这些都可以在 Qt 帮助文档里查看 QMediaPlayer 类的使用方法就可以知道。不再一一列出。

12.3.1 应用实例

本例设计一个比较好看的音乐播放器，界面是编者模仿网上的一个音乐播放器的界面，并非编者原创界面，只是编者用 Qt 实现了网上的一个好看的音乐播放器界面。其中本例有些功能并没有完善，比如播放模式、没有加音量控制等。这些可以由读者自由完善，比较简单。

本例目的：音乐播放器的设计与使用。

例 14_musicplayer，音乐播放器（难度：中等）。项目路径为 Qt/2/14_musicplayer。注意本例有用到 qss 样式文件，关于如何添加资源文件与 qss 文件请参考 7.1.3 小节。音乐播放器的功能这些都为大家所熟知，不用编者介绍了。

项目文件 14_musicplayer.pro 文件第一行添加的代码部分如下。

14_musicplayer.pro 编程后的代码

```
1  QT       += core gui multimedia
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated
APIs.
14 # In order to do so, uncomment the following line.
```

```
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      main.cpp \
20      mainwindow.cpp
21
22  HEADERS += \
23      mainwindow.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/$${TARGET}/bin
27  else: unix:!android: target.path = /opt/$${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
29
30  RESOURCES += \
31      res.qrc
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   14_musicplayer
   * @brief         mainwindow.h
   * @author        Deng Zhimao
   * @email         1252699831@qq.com
   * @net           www.openedv.com
   * @date          2021-04-20

*****************************************************************/
1   #ifndef MAINWINDOW_H
```

```
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QMediaPlayer>
6   #include <QMediaPlaylist>
7   #include <QPushButton>
8   #include <QSlider>
9   #include <QVBoxLayout>
10  #include <QHBoxLayout>
11  #include <QListWidget>
12  #include <QLabel>
13  #include <QSpacerItem>
14  #include <QDebug>
15
16  /* 媒体信息结构体 */
17  struct MediaObjectInfo {
18      /* 用于保存歌曲文件名 */
19      QString fileName;
20      /* 用于保存歌曲文件路径 */
21      QString filePath;
22  };
23
24  class MainWindow : public QMainWindow
25  {
26      Q_OBJECT
27
28  public:
29      MainWindow(QWidget *parent = nullptr);
30      ~MainWindow();
31
32  private:
33      /* 媒体播放器，用于播放音乐 */
34      QMediaPlayer *musicPlayer;
```

```
35
36      /* 媒体列表 */
37      QMediaPlaylist *mediaPlaylist;
38
39      /* 音乐列表 */
40      QListWidget *listWidget;
41
42      /* 播放进度条 */
43      QSlider *durationSlider;
44
45      /* 音乐播放器按钮 */
46      QPushButton *pushButton[7];
47
48      /* 垂直布局 */
49      QVBoxLayout *vBoxLayout[3];
50
51      /* 水平布局 */
52      QHBoxLayout *hBoxLayout[4];
53
54      /* 垂直容器 */
55      QWidget *vWidget[3];
56
57      /* 水平容器 */
58      QWidget *hWidget[4];
59
60      /* 标签文本 */
61      QLabel *label[4];
62
63      /* 用于遮罩 */
64      QWidget *listMask;
65
66      /* 音乐布局函数 */
67      void musicLayout();
```

```
68
69      /* 主窗体大小重设大小函数重写 */
70      void resizeEvent(QResizeEvent *event);
71
72      /* 媒体信息存储 */
73      QVector<MediaObjectInfo> mediaObjectInfo;
74
75      /* 扫描歌曲 */
76      void scanSongs();
77
78      /* 媒体播放器类初始化 */
79      void mediaPlayerInit();
80
81  private slots:
82      /* 播放按钮点击 */
83      void btn_play_clicked();
84
85      /* 下一曲按钮点击*/
86      void btn_next_clicked();
87
88      /* 上一曲按钮点击 */
89      void btn_previous_clicked();
90
91      /* 媒体状态改变 */
92      void mediaPlayerStateChanged(QMediaPlayer::State);
93
94      /* 列表单击 */
95      void listWidgetCliked(QListWidgetItem*);
96
97      /* 媒体列表项改变 */
98      void mediaPlaylistCurrentIndexChanged(int);
99
100     /* 媒体总长度改变 */
```

```
101     void musicPlayerDurationChanged(qint64);
102
103     /* 媒体播放位置改变 */
104     void mediaPlayerPositionChanged(qint64);
105
106     /* 播放进度条松开 */
107     void durationSliderReleased();
108 };
109 #endif // MAINWINDOW_H
```

头文件里主要是声明界面所使用的元素及一些槽函数。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   14_musicplayer
   * @brief         mainwindow.cpp
   * @author        Deng Zhimao
   * @email         1252699831@qq.com
   * @net           www.openedv.com
   * @date          2021-04-20

*****************************************************************/
1   #include "mainwindow.h"
2   #include <QCoreApplication>
3   #include <QFileInfoList>
4   #include <QDir>
5
6   MainWindow::MainWindow(QWidget *parent)
7       : QMainWindow(parent)
8   {
9       /* 布局初始化 */
10      musicLayout();
```

```
11
12      /* 媒体播放器初始化 */
13      mediaPlayerInit();
14
15      /* 扫描歌曲 */
16      scanSongs();
17
18      /* 按钮信号槽连接 */
19      connect(pushButton[0], SIGNAL(clicked()),
20              this, SLOT(btn_previous_clicked()));
21      connect(pushButton[1], SIGNAL(clicked()),
22              this, SLOT(btn_play_clicked()));
23      connect(pushButton[2], SIGNAL(clicked()),
24              this, SLOT(btn_next_clicked()));
25
26      /* 媒体信号槽连接 */
27      connect(musicPlayer,
28              SIGNAL(stateChanged(QMediaPlayer::State)),
29              this,
30              SLOT(mediaPlayerStateChanged(QMediaPlayer::State)));
31      connect(mediaPlaylist,
32              SIGNAL(currentIndexChanged(int)),
33              this,
34              SLOT(mediaPlaylistCurrentIndexChanged(int)));
35      connect(musicPlayer, SIGNAL(durationChanged(qint64)),
36              this,
37              SLOT(musicPlayerDurationChanged(qint64)));
38      connect(musicPlayer,
39              SIGNAL(positionChanged(qint64)),
40              this,
41              SLOT(mediaPlayerPositionChanged(qint64)));
42
43      /* 列表信号槽连接 */
```

```
44      connect(listWidget, SIGNAL(itemClicked(QListWidgetItem*)),
45              this, SLOT(listWidgetCliked(QListWidgetItem*)));
46
47      /* slider 信号槽连接 */
48      connect(durationSlider, SIGNAL(sliderReleased()),
49              this, SLOT(durationSliderReleased()));
50
51      /* 失去焦点 */
52      this->setFocus();
53  }
54
55  void MainWindow::musicLayout()
56  {
57      /* 设置位置与大小,这里固定为 800, 480 */
58      this->setGeometry(0, 0, 800, 480);
59      QPalette pal;
60
61      /* 按钮 */
62      for (int i = 0; i < 7; i++)
63          pushButton[i] = new QPushButton();
64
65      /* 标签 */
66      for (int i = 0; i < 4; i++)
67          label[i] = new QLabel();
68
69      for (int i = 0; i < 3; i++) {
70          /* 垂直容器 */
71          vWidget[i] = new QWidget();
72          vWidget[i]->setAutoFillBackground(true);
73          /* 垂直布局 */
74          vBoxLayout[i] = new QVBoxLayout();
75      }
76
```

```
77      for (int i = 0; i < 4; i++) {
78          /* 水平容器 */
79          hWidget[i] = new QWidget();
80          hWidget[i]->setAutoFillBackground(true);
81          /* 水平布局 */
82          hBoxLayout[i] = new QHBoxLayout();
83      }
84
85      /* 播放进度条 */
86      durationSlider = new QSlider(Qt::Horizontal);
87      durationSlider->setMinimumSize(300, 15);
88      durationSlider->setMaximumHeight(15);
89      durationSlider->setObjectName("durationSlider");
90
91      /* 音乐列表 */
92      listWidget = new QListWidget();
93      listWidget->setObjectName("listWidget");
94      listWidget->resize(310, 265);
95      listWidget->setVerticalScrollBarPolicy(
96                  Qt::ScrollBarAlwaysOff);
97      listWidget->setHorizontalScrollBarPolicy(
98                  Qt::ScrollBarAlwaysOff);
99
100     /* 列表遮罩 */
101     listMask = new QWidget(listWidget);
102     listMask->setMinimumSize(310, 50);
103     listMask->setMinimumHeight(50);
104     listMask->setObjectName("listMask");
105     listMask->setGeometry(0,
106                           listWidget->height() - 50,
107                           310,
108                           50);
109
```

```
110     /* 设置对象名称 */
111     pushButton[0]->setObjectName("btn_previous");
112     pushButton[1]->setObjectName("btn_play");
113     pushButton[2]->setObjectName("btn_next");
114     pushButton[3]->setObjectName("btn_favorite");
115     pushButton[4]->setObjectName("btn_mode");
116     pushButton[5]->setObjectName("btn_menu");
117     pushButton[6]->setObjectName("btn_volume");
118
119     /* 设置按钮属性 */
120     pushButton[1]->setCheckable(true);
121     pushButton[3]->setCheckable(true);
122
123     /* H0 布局 */
124     vWidget[0]->setMinimumSize(310, 480);
125     vWidget[0]->setMaximumWidth(310);
126     vWidget[1]->setMinimumSize(320, 480);
127     QSpacerItem *hSpacer0 = new
128           QSpacerItem(70, 480,
129                       QSizePolicy::Minimum,
130                       QSizePolicy::Maximum);
131
132     QSpacerItem *hSpacer1 = new
133           QSpacerItem(65, 480,
134                       QSizePolicy::Minimum,
135                       QSizePolicy::Maximum);
136
137     QSpacerItem *hSpacer2 = new
138           QSpacerItem(60, 480,
139                       QSizePolicy::Minimum,
140                       QSizePolicy::Maximum);
141
142     hBoxLayout[0]->addSpacerItem(hSpacer0);
```

```
143     hBoxLayout[0]->addWidget(vWidget[0]);
144     hBoxLayout[0]->addSpacerItem(hSpacer1);
145     hBoxLayout[0]->addWidget(vWidget[1]);
146     hBoxLayout[0]->addSpacerItem(hSpacer2);
147     hBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
148
149     hWidget[0]->setLayout(hBoxLayout[0]);
150     setCentralWidget(hWidget[0]);
151
152     /* V0 布局 */
153     listWidget->setMinimumSize(310, 265);
154     hWidget[1]->setMinimumSize(310, 80);
155     hWidget[1]->setMaximumHeight(80);
156     label[0]->setMinimumSize(310, 95);
157     label[0]->setMaximumHeight(95);
158     QSpacerItem *vSpacer0 = new
159           QSpacerItem(310, 10,
160                       QSizePolicy::Minimum,
161                       QSizePolicy::Maximum);
162     QSpacerItem *vSpacer1 = new
163           QSpacerItem(310, 30,
164                       QSizePolicy::Minimum,
165                       QSizePolicy::Minimum);
166     vBoxLayout[0]->addWidget(label[0]);
167     vBoxLayout[0]->addWidget(listWidget);
168     vBoxLayout[0]->addSpacerItem(vSpacer0);
169     vBoxLayout[0]->addWidget(hWidget[1]);
170     vBoxLayout[0]->addSpacerItem(vSpacer1);
171     vBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
172
173     vWidget[0]->setLayout(vBoxLayout[0]);
174
175     /* H1 布局 */
```

```
176     for (int i = 0; i < 3; i++) {
177         pushButton[i]->setMinimumSize(80, 80);
178     }
179     QSpacerItem *hSpacer3 = new
180             QSpacerItem(40, 80,
181                         QSizePolicy::Expanding,
182                         QSizePolicy::Expanding);
183     QSpacerItem *hSpacer4 = new
184             QSpacerItem(40, 80,
185                         QSizePolicy::Expanding,
186                         QSizePolicy::Expanding);
187     hBoxLayout[1]->addWidget(pushButton[0]);
188     hBoxLayout[1]->addSpacerItem(hSpacer3);
189     hBoxLayout[1]->addWidget(pushButton[1]);
190     hBoxLayout[1]->addSpacerItem(hSpacer4);
191     hBoxLayout[1]->addWidget(pushButton[2]);
192     hBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
193
194     hWidget[1]->setLayout(hBoxLayout[1]);
195
196     /* V1 布局 */
197     QSpacerItem *vSpacer2 = new
198             QSpacerItem(320, 40,
199                         QSizePolicy::Minimum,
200                         QSizePolicy::Maximum);
201     QSpacerItem *vSpacer3 = new
202             QSpacerItem(320, 20,
203                         QSizePolicy::Minimum,
204                         QSizePolicy::Maximum);
205     QSpacerItem *vSpacer4 = new
206             QSpacerItem(320, 30,
207                         QSizePolicy::Minimum,
208                         QSizePolicy::Minimum);
```

```
209     label[1]->setMinimumSize(320, 320);
210     QImage Image;
211     Image.load(":/images/cd.png");
212     QPixmap pixmap = QPixmap::fromImage(Image);
213     int with = 320;
214     int height = 320;
215     QPixmap fitpixmap =
216           pixmap.scaled(with, height,
217                         Qt::IgnoreAspectRatio,
218                         Qt::SmoothTransformation);
219     label[1]->setPixmap(fitpixmap);
220     label[1]->setAlignment(Qt::AlignCenter);
221     vWidget[2]->setMinimumSize(300, 80);
222     vWidget[2]->setMaximumHeight(80);
223     vBoxLayout[1]->addSpacerItem(vSpacer2);
224     vBoxLayout[1]->addWidget(label[1]);
225     vBoxLayout[1]->addSpacerItem(vSpacer3);
226     vBoxLayout[1]->addWidget(durationSlider);
227     vBoxLayout[1]->addWidget(vWidget[2]);
228     vBoxLayout[1]->addSpacerItem(vSpacer4);
229     vBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
230
231     vWidget[1]->setLayout(vBoxLayout[1]);
232
233     /* V2 布局 */
234     QSpacerItem *vSpacer5 = new
235           QSpacerItem(300, 10,
236                       QSizePolicy::Minimum,
237                       QSizePolicy::Maximum);
238     hWidget[2]->setMinimumSize(320, 20);
239     hWidget[3]->setMinimumSize(320, 60);
240     vBoxLayout[2]->addWidget(hWidget[2]);
241     vBoxLayout[2]->addSpacerItem(vSpacer5);
```

```
242     vBoxLayout[2]->addWidget(hWidget[3]);
243     vBoxLayout[2]->setContentsMargins(0, 0, 0, 0);
244
245     vWidget[2]->setLayout(vBoxLayout[2]);
246
247     /* H2 布局 */
248     label[2]->setText("00:00");
249     label[3]->setText("00:00");
250     QFont font;
251
252     font.setPixelSize(10);
253
254     /* 设置标签文本 */
255     label[0]->setText("Q Music, Enjoy it! ");
256     label[2]->setText("00:00");
257     label[3]->setText("00:00");
258     label[2]->setSizePolicy(QSizePolicy::Expanding,
259                             QSizePolicy::Expanding);
260     label[3]->setSizePolicy(QSizePolicy::Expanding,
261                             QSizePolicy::Expanding);
262     label[3]->setAlignment(Qt::AlignRight);
263     label[2]->setAlignment(Qt::AlignLeft);
264     label[2]->setFont(font);
265     label[3]->setFont(font);
266
267     pal.setColor(QPalette::WindowText, Qt::white);
268     label[0]->setPalette(pal);
269     label[2]->setPalette(pal);
270     label[3]->setPalette(pal);
271
272     hBoxLayout[2]->addWidget(label[2]);
273     hBoxLayout[2]->addWidget(label[3]);
274
```

```
275     hBoxLayout[2]->setContentsMargins(0, 0, 0, 0);
276     hWidget[2]->setLayout(hBoxLayout[2]);
277
278     /* H3 布局 */
279     QSpacerItem *hSpacer5 = new
280           QSpacerItem(0, 60,
281                       QSizePolicy::Minimum,
282                       QSizePolicy::Maximum);
283     QSpacerItem *hSpacer6 = new
284           QSpacerItem(80, 60,
285                       QSizePolicy::Maximum,
286                       QSizePolicy::Maximum);
287     QSpacerItem *hSpacer7 = new
288           QSpacerItem(80, 60,
289                       QSizePolicy::Maximum,
290                       QSizePolicy::Maximum);
291     QSpacerItem *hSpacer8 = new
292           QSpacerItem(80, 60,
293                       QSizePolicy::Maximum,
294                       QSizePolicy::Maximum);
295     QSpacerItem *hSpacer9 = new
296           QSpacerItem(0, 60,
297                       QSizePolicy::Minimum,
298                       QSizePolicy::Maximum);
299
300     for (int i = 3; i < 7; i++) {
301        pushButton[i]->setMinimumSize(25, 25);
302        pushButton[i]->setMaximumSize(25, 25);
303     }
304
305     hBoxLayout[3]->addSpacerItem(hSpacer5);
306     hBoxLayout[3]->addWidget(pushButton[3]);
307     hBoxLayout[3]->addSpacerItem(hSpacer6);
```

```
308     hBoxLayout[3]->addWidget(pushButton[4]);
309     hBoxLayout[3]->addSpacerItem(hSpacer7);
310     hBoxLayout[3]->addWidget(pushButton[5]);
311     hBoxLayout[3]->addSpacerItem(hSpacer8);
312     hBoxLayout[3]->addWidget(pushButton[6]);
313     hBoxLayout[3]->addSpacerItem(hSpacer9);
314     hBoxLayout[3]->setContentsMargins(0, 0, 0, 0);
315     hBoxLayout[3]->setAlignment(Qt::AlignHCenter);
316 
317     hWidget[3]->setLayout(hBoxLayout[3]);
318 
319     //hWidget[0]->setStyleSheet("background-color:red");
320     //hWidget[1]->setStyleSheet("background-color:#ff5599");
321     //hWidget[2]->setStyleSheet("background-color:#ff55ff");
322     //hWidget[3]->setStyleSheet("background-color:black");
323     //vWidget[0]->setStyleSheet("background-color:#555555");
324     //vWidget[1]->setStyleSheet("background-color:green");
325     //vWidget[2]->setStyleSheet("background-color:gray");
326 
327 }
328 
329 MainWindow::~MainWindow()
330 {
331 }
332 
333 void MainWindow::btn_play_clicked()
334 {
335     int state = musicPlayer->state();
336 
337     switch (state) {
338     case QMediaPlayer::StoppedState:
339         /* 媒体播放 */
340         musicPlayer->play();
```

```
341         break;
342 
343     case QMediaPlayer::PlayingState:
344         /* 媒体暂停 */
345         musicPlayer->pause();
346         break;
347 
348     case QMediaPlayer::PausedState:
349         musicPlayer->play();
350         break;
351     }
352 }
353 
354 void MainWindow::btn_next_clicked()
355 {
356     musicPlayer->stop();
357     int count = mediaPlaylist->mediaCount();
358     if (0 == count)
359         return;
360 
361     /* 列表下一个 */
362     mediaPlaylist->next();
363     musicPlayer->play();
364 }
365 
366 void MainWindow::btn_previous_clicked()
367 {
368     musicPlayer->stop();
369     int count = mediaPlaylist->mediaCount();
370     if (0 == count)
371         return;
372 
373     /* 列表上一个 */
```

```
374     mediaPlaylist->previous();
375     musicPlayer->play();
376 }
377
378 void MainWindow::mediaPlayerStateChanged(
379         QMediaPlayer::State
380         state)
381 {
382     switch (state) {
383     case QMediaPlayer::StoppedState:
384         pushButton[1]->setChecked(false);
385         break;
386
387     case QMediaPlayer::PlayingState:
388         pushButton[1]->setChecked(true);
389         break;
390
391     case QMediaPlayer::PausedState:
392         pushButton[1]->setChecked(false);
393         break;
394     }
395 }
396
397 void MainWindow::listWidgetCliked(QListWidgetItem *item)
398 {
399     musicPlayer->stop();
400     mediaPlaylist->setCurrentIndex(listWidget->row(item));
401     musicPlayer->play();
402 }
403
404 void MainWindow::mediaPlaylistCurrentIndexChanged(
405         int index)
406 {
```

```
407     if (-1 == index)
408         return;
409 
410     /* 设置列表正在播放的项 */
411     listWidget->setCurrentRow(index);
412 }
413 
414 void MainWindow::musicPlayerDurationChanged(
415         qint64 duration)
416 {
417     durationSlider->setRange(0, duration / 1000);
418     int second  = duration / 1000;
419     int minute = second / 60;
420     second %= 60;
421 
422     QString mediaDuration;
423     mediaDuration.clear();
424 
425     if (minute >= 10)
426         mediaDuration = QString::number(minute, 10);
427     else
428         mediaDuration = "0" + QString::number(minute, 10);
429 
430     if (second >= 10)
431         mediaDuration = mediaDuration
432                 + ":" + QString::number(second, 10);
433     else
434         mediaDuration = mediaDuration
435                 + ":0" + QString::number(second, 10);
436 
437     /* 显示媒体总长度时间 */
438     label[3]->setText(mediaDuration);
439 }
```

```
440
441 void MainWindow::mediaPlayerPositionChanged(
442         qint64 position)
443 {
444     if (!durationSlider->isSliderDown())
445         durationSlider->setValue(position/1000);
446
447     int second  = position / 1000;
448     int minute = second / 60;
449     second %= 60;
450
451     QString mediaPosition;
452     mediaPosition.clear();
453
454     if (minute >= 10)
455         mediaPosition = QString::number(minute, 10);
456     else
457         mediaPosition = "0" + QString::number(minute, 10);
458
459     if (second >= 10)
460         mediaPosition = mediaPosition
461                 + ":" + QString::number(second, 10);
462     else
463         mediaPosition = mediaPosition
464                 + ":0" + QString::number(second, 10);
465
466     /* 显示现在播放的时间 */
467     label[2]->setText(mediaPosition);
468 }
469
470 void MainWindow::resizeEvent(QResizeEvent *event)
471 {
472     Q_UNUSED(event);
```

```
473     listMask->setGeometry(0,
474                           listWidget->height() - 50,
475                           310,
476                           50);
477 }
478
479 void MainWindow::durationSliderReleased()
480 {
481     /* 设置媒体播放的位置 */
482     musicPlayer->setPosition(durationSlider->value() * 1000);
483 }
484
485 void MainWindow::scanSongs()
486 {
487     QDir dir(QCoreApplication::applicationDirPath()
488              + "/myMusic");
489     QDir dirbsolutePath(dir.absolutePath());
490     /* 如果目录存在 */
491     if (dirbsolutePath.exists()) {
492         /* 定义过滤器 */
493         QStringList filter;
494         /* 包含所有.mp3 后缀的文件 */
495         filter << "*.mp3";
496         /* 获取该目录下的所有文件 */
497         QFileInfoList files =
498                 dirbsolutePath.entryInfoList(filter, QDir::Files);
499         /* 遍历 */
500         for (int i = 0; i < files.count(); i++) {
501             MediaObjectInfo info;
502             /* 使用 utf-8 编码 */
503             QString fileName = QString::fromUtf8(files.at(i)
504                                                  .fileName()
505                                                  .replace(".mp3", "")
```

```
506                                             .toUtf8()
507                                             .data());
508             info.fileName = fileName + "\n"
509                     + fileName.split("-").at(1);
510             info.filePath = QString::fromUtf8(files.at(i)
511                                               .filePath()
512                                               .toUtf8()
513                                               .data());
514             /* 媒体列表添加歌曲 */
515             if (mediaPlaylist->addMedia(
516                     QUrl::fromLocalFile(info.filePath))) {
517                 /* 添加到容器数组里储存 */
518                 mediaObjectInfo.append(info);
519                 /* 添加歌曲名字至列表 */
520                 listWidget->addItem(info.fileName);
521             } else {
522                 qDebug()<<
523                           mediaPlaylist->errorString()
524                           .toUtf8().data()
525                         << endl;
526                 qDebug()<< "  Error number:"
527                          << mediaPlaylist->error()
528                          << endl;
529             }
530         }
531     }
532 }
533
534 void MainWindow::mediaPlayerInit()
535 {
536     musicPlayer = new QMediaPlayer(this);
537     mediaPlaylist = new QMediaPlaylist(this);
538     /* 确保列表是空的 */
```

```
539     mediaPlaylist->clear();
540     /* 设置音乐播放器的列表为 mediaPlaylist */
541     musicPlayer->setPlaylist(mediaPlaylist);
542     /* 设置播放模式，Loop 是列循环 */
543     mediaPlaylist->setPlaybackMode(QMediaPlaylist::Loop);
544 }
```

第 10 行，布局初始化，第一步我们先建立好界面，确定好布局再实现功能，一般流程都这样，布局 msuicLayout()的内容比较多，也不难，但是比较复杂，如果我们有第七章的基础，看这种布局是没有难度的，这里就不多解释了。没有好看的布局也能完成本例。如果您喜欢这种布局方法，您需要多花点时间去研究如何布局才好看，这些没有固定的方法，完全是一个人的审美感。

第 13 行，媒体先初始化，初始化媒体播放器与媒体播放列表。

第 16 行，扫描歌曲，初始化本地歌曲，从固定的文件夹里找歌曲文件，这里编者设计从固定的文件夹里找歌曲文件。也有些读者可能会说可以自由选择文件夹吗？答案是可以的！使用 QFileDilog 类打开选择歌曲目录或者文件即可！但是在嵌入式里，一般是初始化界里面就已经有歌曲在界面里的了，无需用户再去打开，打开歌曲这种操作是极少会使用的！

第 26 至 47 行，信号槽连接，这里使用了各种各样的信号。这里有必要说明一下，编者设计了单击歌曲列表就能播放歌曲了。如果在电脑上可能有些播放器软件会双击才能播放歌曲，在嵌入式的触摸屏里，只有单击！没有双击，没有用户会双击歌曲的列表的。这些特殊的地方我们需要在嵌入式里考虑！

第 333~376 行，点击播放按钮，上一曲，下一曲，Qt 的媒体类已经提供了 previous()，next()，stop()，play()，pause()这些方法直接可以使用。除了实现好看的界面之外，这部分内容也是本例实现播放器重要的部分！

其他部分是实现播放状态的切换及扩进度条的显示进度处理，请参阅源码理解。

main.cpp 内容如下，主要是加载 qss 样式文件。没有什么可讲解。

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4   #include <QFile>
5
6   int main(int argc, char *argv[])
7   {
8       QApplication a(argc, argv);
```

```
    /* 指定文件 */
    QFile file(":/style.qss");

    /* 判断文件是否存在 */
    if (file.exists() ) {
        /* 以只读的方式打开 */
        file.open(QFile::ReadOnly);
        /* 以字符串的方式保存读出的结果 */
        QString styleSheet = QLatin1String(file.readAll());
        /* 设置全局样式 */
        qApp->setStyleSheet(styleSheet);
        /* 关闭文件 */
        file.close();
    }

    MainWindow w;
    w.show();
    return a.exec();
}
```

style.qss 样式文件如下。素材已经在源码处提供。注意下面的 style.qss 不能有注释！

```
QWidget {
background: "#25242a"
}

QWidget#listMask {
border-image: url(:/images/mask.png);
background-color: transparent;
}

QListWidget#listWidget {
color:white;
font-size: 15px;
border:none;
```

```
}

QListWidget#listWidget:item:active {
background: transparent;
}

QListWidget#listWidget:item {
background: transparent;
height:60;
}

QListWidget#listWidget:item:selected {
color:#5edcf3;
background: transparent;
}

QListWidget#listWidget:item:hover {
background: transparent;
color:#5edcf3;
border:none;
}

QPushButton#btn_play {
border-image:url(:/images/btn_play1.png);
}

QPushButton#btn_play:hover {
border-image:url(:/images/btn_play2.png);
}

QPushButton#btn_play:checked {
border-image:url(:/images/btn_pause1.png);
}
```

```
47
48  QPushButton#btn_play:checked:hover {
49  border-image:url(:/images/btn_pause2.png);
50  }
51
52  QPushButton#btn_previous {
53  border-image:url(:/images/btn_previous1.png);
54  }
55
56  QPushButton#btn_previous:hover {
57  border-image:url(:/images/btn_previous2.png);
58  }
59
60  QPushButton#btn_next {
61  border-image:url(:/images/btn_next1.png);
62  }
63
64  QPushButton#btn_next:hover {
65  border-image:url(:/images/btn_next2.png);
66  }
67
68  QPushButton#btn_favorite {
69  border-image:url(:/images/btn_favorite_no.png);
70  }
71
72  QPushButton#btn_favorite:checked {
73  border-image:url(:/images/btn_favorite_yes.png);
74  }
75
76  QPushButton#btn_menu {
77  border-image:url(:/images/btn_menu1.png);
78  }
79
```

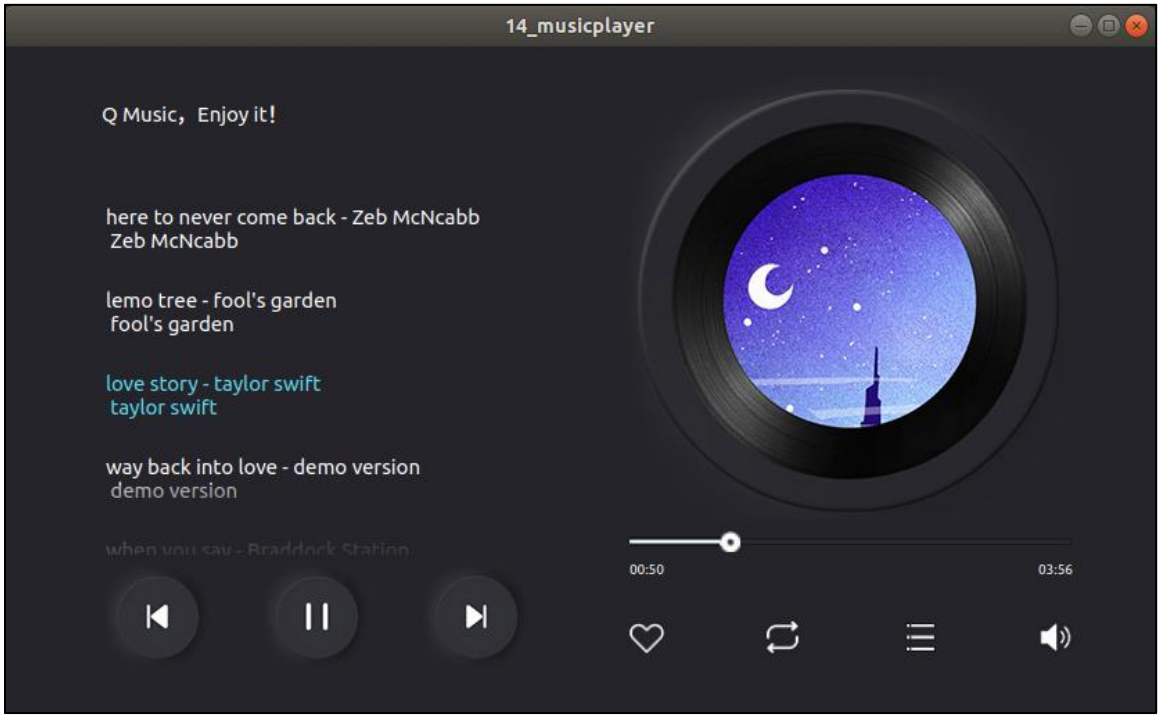
```
80  QPushButton#btn_menu:hover {
81  border-image:url(:/images/btn_menu2.png);
82  }
83
84  QPushButton#btn_mode {
85  border-image:url(:/images/btn_listcircle1.png);
86  }
87
88  QPushButton#btn_mode:hover {
89  border-image:url(:/images/btn_listcircle2.png);
90  }
91
92  QPushButton#btn_mode {
93  border-image:url(:/images/btn_listcircle1.png);
94  }
95
96  QPushButton#btn_mode:hover {
97  border-image:url(:/images/btn_listcircle2.png);
98  }
99
100 QPushButton#btn_volume {
101 border-image:url(:/images/btn_volume1.png);
102 }
103
104 QPushButton#btn_volume:hover {
105 border-image:url(:/images/btn_volume2.png);
106 }
107
108 QSlider#durationSlider:handle:horizontal {
109 border-image:url(:/images/handle.png);
110 }
111
112 QSlider#durationSlider:sub-page:horizontal {
```

```
113 border-image:url(:/images/sub-page.png);
114 }
```

12.3.2 程序运行效果

先点击构建项目，项目构建完成后，再将本例的 myMusic 歌曲文件夹拷贝到可执行程序的文件夹同一级目录下，也就是 build-14_musicplayer-Desktop_Qt_5_12_9_GCC_64bit-Debug 目录下。再点击运行，就出现歌曲在列表里，如下图，点击播放即可播放歌曲，上一曲，下一曲也可以用。注意右下角的某些按钮功能，在本例没有继续去实现，比如音量控制，可以直接加一个垂直方向的滑条，然后控制媒体的软件音量即可。留给读者自由发挥，可以基于本例去开发，就当读者练习吧。本例的界面开发编者还是比较满意的，前面的界面都比较普通，编者个人 Qt 开发重要的是界面与稳定性，功能是次要的！因为功能都不难实现。

注意歌曲格式应严格为歌名 + “-” + 歌手名称.mp3，例如江南 - 林俊杰.mp3。中间的“-”是英文字符的短横杠，这样的目的是能够将歌曲名称和歌手分开显示到界面上。



12.4 视频播放器

与音乐播放器一样使用 QMediaPlayer 类，不同的是需要使用 setVideoOutput(QVideoWidget*)设置一个视频输出窗口，好让视频在此窗口显示，其他步骤基本都一样。

12.4.1 应用实例

本例设计一个比较好看且简洁的视频播放器，界面是编者原创界面。

本例目的：视频播放器的设计与使用。

例 15_videoplayer，视频播放器（难度：中等）。项目路径为 Qt/2/15_videoplayer。注意本例有用到 qss 样式文件，关于如何添加资源文件与 qss 文件请参考 7.1.3 小节。音乐播放器的功能这些都为大家所熟知，不用编者介绍了。

项目文件 15_videoplayer.pro 文件第一行添加的代码部分如下。

15_videoplayer.pro 编程后的代码

```
1  QT       += core gui multimedia multimediawidgets
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated
APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0
17
18 SOURCES += \
19     main.cpp \
20     mainwindow.cpp
21
22 HEADERS += \
```

```
23      mainwindow.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/$${TARGET}/bin
27  else: unix:!android: target.path = /opt/$${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
29
30  RESOURCES += \
31      res.qrc
```

在头文件“mainwindow.h”具体代码如下。

mainwindow.h 编程后的代码

```
/*****************************************************************
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   15_videoplayer
   * @brief         mainwindow.h
   * @author        Deng Zhimao
   * @email         1252699831@qq.com
   * @net           www.openedv.com
   * @date          2021-04-27

*****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QMediaPlayer>
6   #include <QMediaPlaylist>
7   #include <QPushButton>
8   #include <QSlider>
9   #include <QVBoxLayout>
10  #include <QHBoxLayout>
11  #include <QListWidget>
```

```
12  #include <QLabel>
13  #include <QSpacerItem>
14  #include <QVideoWidget>
15  #include <QDebug>
16
17  /* 媒体信息结构体 */
18  struct MediaObjectInfo {
19      /* 用于保存视频文件名 */
20      QString fileName;
21      /* 用于保存视频文件路径 */
22      QString filePath;
23  };
24
25  class MainWindow : public QMainWindow
26  {
27      Q_OBJECT
28
29  public:
30      MainWindow(QWidget *parent = nullptr);
31      ~MainWindow();
32
33  private:
34      /* 媒体播放器，用于播放视频 */
35      QMediaPlayer *videoPlayer;
36
37      /* 媒体列表 */
38      QMediaPlaylist *mediaPlaylist;
39
40      /* 视频显示窗口 */
41      QVideoWidget *videoWidget;
42
43      /* 视频列表 */
44      QListWidget *listWidget;
```

```
45
46      /* 播放进度条 */
47      QSlider *durationSlider;
48
49      /* 音量条 */
50      QSlider *volumeSlider;
51
52      /* 视频播放器按钮 */
53      QPushButton *pushButton[5];
54
55      /* 水平布局 */
56      QHBoxLayout *hBoxLayout[3];
57
58      /* 水平容器 */
59      QWidget *hWidget[3];
60
61      /* 标签文本 */
62      QLabel *label[2];
63
64      /* 垂直容器 */
65      QWidget *vWidget[2];
66
67      /* 垂直界面 */
68      QVBoxLayout *vBoxLayout[2];
69
70      /* 视频布局函数 */
71      void videoLayout();
72
73      /* 主窗体大小重设大小函数重写 */
74      void resizeEvent(QResizeEvent *event);
75
76      /* 媒体信息存储 */
77      QVector<MediaObjectInfo> mediaObjectInfo;
```

```
78
79      /* 扫描本地视频文件 */
80      void scanVideoFiles();
81
82      /* 媒体初始化 */
83      void mediaPlayerInit();
84  private slots:
85      /* 播放按钮点击 */
86      void btn_play_clicked();
87
88      /* 下一个视频按钮点击 */
89      void btn_next_clicked();
90
91      /* 音量加 */
92      void btn_volmeup_clicked();
93
94      /* 音量减 */
95      void btn_volmedown_clicked();
96
97      /* 全屏 */
98      void btn_fullscreen_clicked();
99
100     /* 媒体状态改变 */
101     void mediaPlayerStateChanged(QMediaPlayer::State);
102
103     /* 列表单击 */
104     void listWidgetCliked(QListWidgetItem*);
105
106     /* 媒体列表项改变 */
107     void mediaPlaylistCurrentIndexChanged(int);
108
109     /* 媒体总长度改变 */
110     void musicPlayerDurationChanged(qint64);
```

```
111
112     /* 媒体播放位置改变 */
113     void mediaPlayerPositionChanged(qint64);
114
115     /* 播放进度条松开 */
116     void durationSliderReleased();
117
118     /* 音量条松开 */
119     void volumeSliderReleased();
120 };
121 #endif // MAINWINDOW_H
```

头文件里主要是声明界面所使用的元素及一些槽函数。

在源文件“mainwindow.cpp”具体代码如下。

mainwindow.cpp 编程后的代码

```
/*****************************************************************
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   15_videoplayer
   * @brief         mainwindow.cpp
   * @author        Deng Zhimao
   * @email         1252699831@qq.com
   * @net           www.openedv.com
   * @date          2021-04-27

*****************************************************************/
1   #include "mainwindow.h"
2   #include <QCoreApplication>
3   #include <QFileInfoList>
4   #include <QDir>
5
6   MainWindow::MainWindow(QWidget *parent)
7       : QMainWindow(parent)
8   {
```

```
9       /* 视频播放器布局初始化 */
10      videoLayout();
11
12      /* 媒体初始化 */
13      mediaPlayerInit();
14
15      /* 扫描本地视频 */
16      scanVideoFiles();
17
18      /* 设置按钮的属性 */
19      pushButton[0]->setCheckable(true);
20      pushButton[4]->setCheckable(true);
21
22      /* 按钮连接信号槽 */
23      connect(pushButton[0], SIGNAL(clicked()),
24              this, SLOT(btn_play_clicked()));
25      connect(pushButton[1], SIGNAL(clicked()),
26              this, SLOT(btn_next_clicked()));
27      connect(pushButton[2], SIGNAL(clicked()),
28              this, SLOT(btn_volmedown_clicked()));
29      connect(pushButton[3], SIGNAL(clicked()),
30              this, SLOT(btn_volmeup_clicked()));
31      connect(pushButton[4], SIGNAL(clicked()),
32              this, SLOT(btn_fullscreen_clicked()));
33
34      /* 列表连接信号槽 */
35      connect(listWidget, SIGNAL(itemClicked(QListWidgetItem*)),
36              this, SLOT(listWidgetCliked(QListWidgetItem*)));
37
38      /* 媒体连接信号槽 */
39      connect(videoPlayer,
40              SIGNAL(stateChanged(QMediaPlayer::State)),
41              this,
```

```
42              SLOT(mediaPlayerStateChanged(QMediaPlayer::State)));
43      connect(mediaPlaylist,
44              SIGNAL(currentIndexChanged(int)),
45              this,
46              SLOT(mediaPlaylistCurrentIndexChanged(int)));
47      connect(videoPlayer, SIGNAL(durationChanged(qint64)),
48              this,
49              SLOT(musicPlayerDurationChanged(qint64)));
50      connect(videoPlayer,
51              SIGNAL(positionChanged(qint64)),
52              this,
53              SLOT(mediaPlayerPositionChanged(qint64)));
54
55      /* slider 信号槽连接 */
56      connect(durationSlider, SIGNAL(sliderReleased()),
57              this, SLOT(durationSliderReleased()));
58      connect(volumeSlider, SIGNAL(sliderReleased()),
59              this, SLOT(volumeSliderReleased()));
60  }
61
62  MainWindow::~MainWindow()
63  {
64  }
65
66  void MainWindow::videoLayout()
67  {
68      /* 设置位置与大小,这里固定为 800, 480 */
69      this->setGeometry(0, 0, 800, 480);
70      //    this->setMinimumSize(800, 480);
71      //    this->setMaximumSize(800, 480);
72      QPalette pal;
73      pal.setColor(QPalette::WindowText, Qt::white);
74
```

```
75      for (int i = 0; i < 3; i++) {
76          /* 水平容器 */
77          hWidget[i] = new QWidget();
78          hWidget[i]->setAutoFillBackground(true);
79          /* 水平布局 */
80          hBoxLayout[i] = new QHBoxLayout();
81      }
82
83      for (int i = 0; i < 2; i++) {
84          /* 垂直容器 */
85          vWidget[i] = new QWidget();
86          vWidget[i]->setAutoFillBackground(true);
87          /* 垂直布局 */
88          vBoxLayout[i] = new QVBoxLayout();
89      }
90
91      for (int i = 0; i < 2; i++) {
92          label[i] = new QLabel();
93      }
94
95      for (int i = 0; i < 5; i++) {
96          pushButton[i] = new QPushButton();
97          pushButton[i]->setMaximumSize(44, 44);
98          pushButton[i]->setMinimumSize(44, 44);
99      }
100
101     /* 设置 */
102     vWidget[0]->setObjectName("vWidget0");
103     vWidget[1]->setObjectName("vWidget1");
104     hWidget[1]->setObjectName("hWidget1");
105     hWidget[2]->setObjectName("hWidget2");
106     pushButton[0]->setObjectName("btn_play");
107     pushButton[1]->setObjectName("btn_next");
```

```
108     pushButton[2]->setObjectName("btn_volumedown");
109     pushButton[3]->setObjectName("btn_volumeup");
110     pushButton[4]->setObjectName("btn_screen");
111
112     QFont font;
113
114     font.setPixelSize(18);
115     label[0]->setFont(font);
116     label[1]->setFont(font);
117
118     pal.setColor(QPalette::WindowText, Qt::white);
119     label[0]->setPalette(pal);
120     label[1]->setPalette(pal);
121
122     label[0]->setText("00:00");
123     label[1]->setText("/00:00");
124
125     durationSlider = new QSlider(Qt::Horizontal);
126     durationSlider->setMaximumHeight(15);
127     durationSlider->setObjectName("durationSlider");
128
129     volumeSlider = new QSlider(Qt::Horizontal);
130     volumeSlider->setRange(0, 100);
131     volumeSlider->setMaximumWidth(80);
132     volumeSlider->setObjectName("volumeSlider");
133     volumeSlider->setValue(50);
134
135     listWidget = new QListWidget();
136     listWidget->setObjectName("listWidget");
137     listWidget->setVerticalScrollBarPolicy(
138               Qt::ScrollBarAlwaysOff);
139     listWidget->setHorizontalScrollBarPolicy(
140               Qt::ScrollBarAlwaysOff);
```

```
141     //listWidget->setFocusPolicy(Qt::NoFocus);
142     videoWidget = new QVideoWidget();
143     videoWidget->setStyleSheet("border-image: none;"
144                                "background: transparent;"
145                                "border:none");
146
147     /* H0 布局 */
148     vWidget[0]->setMinimumSize(300, 480);
149     vWidget[0]->setMaximumWidth(300);
150     videoWidget->setMinimumSize(500, 480);
151
152     hBoxLayout[0]->addWidget(videoWidget);
153     hBoxLayout[0]->addWidget(vWidget[0]);
154
155     hWidget[0]->setLayout(hBoxLayout[0]);
156     hBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
157
158     setCentralWidget(hWidget[0]);
159
160     /* V0 布局 */
161     QSpacerItem *vSpacer0 = new
162           QSpacerItem(0, 80,
163                       QSizePolicy::Minimum,
164                       QSizePolicy::Maximum);
165     vBoxLayout[0]->addWidget(listWidget);
166     vBoxLayout[0]->addSpacerItem(vSpacer0);
167     vBoxLayout[0]->setContentsMargins(0, 0, 0, 0);
168
169     vWidget[0]->setLayout(vBoxLayout[0]);
170
171     /* V1 布局 */
172     /* 底板部件布局 */
173     hWidget[1]->setMaximumHeight(15);
```

```
174     hWidget[2]->setMinimumHeight(65);
175     vBoxLayout[1]->addWidget(hWidget[1]);
176     vBoxLayout[1]->addWidget(hWidget[2]);
177     vBoxLayout[1]->setAlignment(Qt::AlignCenter);
178
179     vWidget[1]->setLayout(vBoxLayout[1]);
180     vWidget[1]->setParent(this);
181     vWidget[1]->setGeometry(0, this->height() - 80, this->width(),
80);
182     vBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
183     /* 位于最上层 */
184     vWidget[1]->raise();
185
186     /* H1 布局 */
187     hBoxLayout[1]->addWidget(durationSlider);
188     hBoxLayout[1]->setContentsMargins(0, 0, 0, 0);
189     hWidget[1]->setLayout(hBoxLayout[1]);
190
191     /* H2 布局 */
192     QSpacerItem *hSpacer0 = new
193           QSpacerItem(300, 80,
194                       QSizePolicy::Expanding,
195                       QSizePolicy::Maximum);
196
197     hBoxLayout[2]->addSpacing(20);
198     hBoxLayout[2]->addWidget(pushButton[0]);
199     hBoxLayout[2]->addSpacing(10);
200     hBoxLayout[2]->addWidget(pushButton[1]);
201     hBoxLayout[2]->addSpacing(10);
202     hBoxLayout[2]->addWidget(pushButton[2]);
203     hBoxLayout[2]->addWidget(volumeSlider);
204     hBoxLayout[2]->addWidget(pushButton[3]);
205     hBoxLayout[2]->addWidget(label[0]);
```

```
206     hBoxLayout[2]->addWidget(label[1]);
207     hBoxLayout[2]->addSpacerItem(hSpacer0);
208     hBoxLayout[2]->addWidget(pushButton[4]);
209     hBoxLayout[2]->addSpacing(20);
210     hBoxLayout[2]->setContentsMargins(0, 0, 0, 0);
211     hBoxLayout[2]->setAlignment(Qt::AlignLeft | Qt::AlignTop);
212 
213     hWidget[2]->setLayout(hBoxLayout[2]);
214 }
215 
216 void MainWindow::mediaPlayerInit()
217 {
218     videoPlayer = new QMediaPlayer(this);
219     mediaPlaylist = new QMediaPlaylist(this);
220     /* 确保列表是空的 */
221     mediaPlaylist->clear();
222     /* 设置视频播放器的列表为 mediaPlaylist */
223     videoPlayer->setPlaylist(mediaPlaylist);
224     /* 设置视频输出窗口 */
225     videoPlayer->setVideoOutput(videoWidget);
226     /* 设置播放模式，Loop 是列循环 */
227     mediaPlaylist->setPlaybackMode(QMediaPlaylist::Loop);
228     /* 设置默认软件音量为 50% */
229     videoPlayer->setVolume(50);
230 }
231 
232 void MainWindow::resizeEvent(QResizeEvent *event)
233 {
234     Q_UNUSED(event);
235     vWidget[1]->setGeometry(0, this->height() - 80, this->width(),
80);
236 }
237 
```

```
238 void MainWindow::btn_play_clicked()
239 {
240     int state = videoPlayer->state();
241     switch (state) {
242     case QMediaPlayer::StoppedState:
243         /* 媒体播放 */
244         videoPlayer->play();
245         break;
246 
247     case QMediaPlayer::PlayingState:
248         /* 媒体暂停 */
249         videoPlayer->pause();
250         break;
251 
252     case QMediaPlayer::PausedState:
253         /* 设置视频输出窗口 */
254         videoPlayer->play();
255         break;
256     }
257 }
258 
259 void MainWindow::btn_next_clicked()
260 {
261     videoPlayer->stop();
262     int count = mediaPlaylist->mediaCount();
263     if (0 == count)
264         return;
265 
266     /* 列表下一个 */
267     mediaPlaylist->next();
268     videoPlayer->play();
269 }
270 
```

```
271 void MainWindow::btn_volmeup_clicked()
272 {
273     /* 点击每次音量+5 */
274     volumeSlider->setValue(volumeSlider->value() + 5);
275     videoPlayer->setVolume(volumeSlider->value());
276 }
277
278 void MainWindow::btn_fullscreen_clicked()
279 {
280     /* 全屏/非全屏操作 */
281     vWidget[0]->setVisible(!pushButton[4]->isChecked());
282 }
283
284 void MainWindow::btn_volmedown_clicked()
285 {
286     /* 点击每次音量-5 */
287     volumeSlider->setValue(volumeSlider->value() - 5);
288     videoPlayer->setVolume(volumeSlider->value());
289 }
290
291 void MainWindow::mediaPlayerStateChanged(
292         QMediaPlayer::State
293         state)
294 {
295     switch (state) {
296     case QMediaPlayer::StoppedState:
297         pushButton[0]->setChecked(false);
298         break;
299
300     case QMediaPlayer::PlayingState:
301         pushButton[0]->setChecked(true);
302         break;
303
```

```
304     case QMediaPlayer::PausedState:
305         pushButton[0]->setChecked(false);
306         break;
307     }
308 }
309 
310 void MainWindow::listWidgetCliked(QListWidgetItem *item)
311 {
312     videoPlayer->stop();
313     mediaPlaylist->setCurrentIndex(listWidget->row(item));
314     videoPlayer->play();
315 }
316 
317 void MainWindow::mediaPlaylistCurrentIndexChanged(
318         int index)
319 {
320     if (-1 == index)
321         return;
322 
323     /* 设置列表正在播放的项 */
324     listWidget->setCurrentRow(index);
325 }
326 
327 void MainWindow::musicPlayerDurationChanged(
328         qint64 duration)
329 {
330     durationSlider->setRange(0, duration / 1000);
331     int second  = duration / 1000;
332     int minute = second / 60;
333     second %= 60;
334 
335     QString mediaDuration;
336     mediaDuration.clear();
```

```
337
338     if (minute >= 10)
339         mediaDuration = QString::number(minute, 10);
340     else
341         mediaDuration = "0" + QString::number(minute, 10);
342
343     if (second >= 10)
344         mediaDuration = mediaDuration
345                 + ":" + QString::number(second, 10);
346     else
347         mediaDuration = mediaDuration
348                 + ":0" + QString::number(second, 10);
349
350     /* 显示媒体总长度时间 */
351     label[1]->setText("/" + mediaDuration);
352 }
353
354 void MainWindow::mediaPlayerPositionChanged(
355         qint64 position)
356 {
357     if (!durationSlider->isSliderDown())
358         durationSlider->setValue(position / 1000);
359
360     int second  = position / 1000;
361     int minute = second / 60;
362     second %= 60;
363
364     QString mediaPosition;
365     mediaPosition.clear();
366
367     if (minute >= 10)
368         mediaPosition = QString::number(minute, 10);
369     else
```

```
370         mediaPosition = "0" + QString::number(minute, 10);
371 
372     if (second >= 10)
373         mediaPosition = mediaPosition
374                 + ":" + QString::number(second, 10);
375     else
376         mediaPosition = mediaPosition
377                 + ":0" + QString::number(second, 10);
378 
379     /* 显示现在播放的时间 */
380     label[0]->setText(mediaPosition);
381 }
382 
383 void MainWindow::durationSliderReleased()
384 {
385     /* 设置媒体播放的位置 */
386     videoPlayer->setPosition(durationSlider->value() * 1000);
387 }
388 
389 void MainWindow::volumeSliderReleased()
390 {
391     /* 设置音量 */
392     videoPlayer->setVolume(volumeSlider->value());
393 }
394 
395 void MainWindow::scanVideoFiles()
396 {
397     QDir dir(QCoreApplication::applicationDirPath()
398             + "/myVideo");
399     QDir dirbsolutePath(dir.absolutePath());
400     /* 如果目录存在 */
401     if (dirbsolutePath.exists()) {
402         /* 定义过滤器 */
```

```
403         QStringList filter;
404         /* 包含所有 xx 后缀的文件 */
405         filter << "*.mp4" << "*.mkv" << "*.wmv" << "*.avi";
406         /* 获取该目录下的所有文件 */
407         QFileInfoList files =
408                 dirbsolutePath.entryInfoList(filter, QDir::Files);
409         /* 遍历 */
410         for (int i = 0; i < files.count(); i++) {
411             MediaObjectInfo info;
412             /* 使用 utf-8 编码 */
413             info.fileName = QString::fromUtf8(files.at(i)
414                                               .fileName()
415                                               .toUtf8()
416                                               .data());
417             info.filePath = QString::fromUtf8(files.at(i)
418                                               .filePath()
419                                               .toUtf8()
420                                               .data());
421             /* 媒体列表添加视频 */
422             if (mediaPlaylist->addMedia(
423                         QUrl::fromLocalFile(info.filePath))) {
424                 /* 添加到容器数组里储存 */
425                 mediaObjectInfo.append(info);
426                 /* 添加视频名字至列表 */
427                 listWidget->addItem(info.fileName);
428             } else {
429                 qDebug()<<
430                           mediaPlaylist->errorString()
431                           .toUtf8().data()
432                         << endl;
433                 qDebug()<< "  Error number:"
434                         << mediaPlaylist->error()
435                         << endl;
```

```
436             }
437         }
438     }
439 }
```

与上一小节音乐播放器的一样，在构造函数里布局初始化，然后执行扫描本地视频文件。之后就是一些信号槽的连接，基本上就是这么一个流程了。

第 395~439 行，扫本地目录的视频文件，通过过滤文件名的后缀，将视频文件名添加至媒体列表里，就可以点击播放了，需要更多的格式的视频文件，可以自己尝试修改需要过滤的文件名。

main.cpp 内容如下，主要是加载 qss 样式文件。没有什么可讲解。

```
1   #include "mainwindow.h"
2
3   #include <QApplication>
4   #include <QFile>
5
6   int main(int argc, char *argv[])
7   {
8       QApplication a(argc, argv);
9       /* 指定文件 */
10      QFile file(":/style.qss");
11
12      /* 判断文件是否存在 */
13      if (file.exists() ) {
14          /* 以只读的方式打开 */
15          file.open(QFile::ReadOnly);
16          /* 以字符串的方式保存读出的结果 */
17          QString styleSheet = QLatin1String(file.readAll());
18          /* 设置全局样式 */
19          qApp->setStyleSheet(styleSheet);
20          /* 关闭文件 */
21          file.close();
22      }
23
```

```
    MainWindow w;
    w.show();
    return a.exec();
}
```

style.qss 样式文件如下。素材已经在源码处提供。注意下面的 style.qss 不能有注释！

```
QWidget {
border-image:url(:/images/bg.png);
}

QLabel {
border-image:none;
}

QWidget#hWidget1 {
border-image:none;
background:transparent;
}

QWidget#hWidget2 {
border-image:none;
background:transparent;
}

QWidget#vWidget1 {
border-image:url(:/images/mask.png);
background:#24252a;
}

QWidget#vWidget0 {
border-image:none;
}

QListWidget#listWidget {
```

```
color:white;
font-size: 15px;
border:none;
background: "#20ffffff";
border-image:none;
}

QListWidgetlistWidget:item:active {
background: transparent;
}

QListWidget#listWidget:item {
background: transparent;
height:60;
}

QListWidget#listWidget:item:selected {
color:#5edcf3;
background: transparent;
}

QListWidget#listWidget:item:hover {
background: transparent;
color:#5edcf3;
border:none;
}


QPushButton#btn_play {
border-image:url(:/icons/btn_play1.png);
}

QPushButton#btn_play:hover {
```

```
62  border-image:url(:/icons/btn_play2.png);
63  }
64
65  QPushButton#btn_play:checked {
66  border-image:url(:/icons/btn_pause1.png);
67  }
68
69  QPushButton#btn_play:checked:hover {
70  border-image:url(:/icons/btn_pause2.png);
71  }
72
73  QPushButton#btn_next {
74  border-image:url(:/icons/btn_next1.png);
75  }
76
77  QPushButton#btn_next:hover {
78  border-image:url(:/icons/btn_next2.png);
79  }
80
81  QPushButton#btn_volumedown {
82  border-image:url(:/icons/btn_volumedown1.png);
83  }
84
85  QPushButton#btn_volumedown:hover {
86  border-image:url(:/icons/btn_volumedown2.png);
87  }
88
89  QPushButton#btn_volumeup {
90  border-image:url(:/icons/btn_volumeup1.png);
91  }
92
93  QPushButton#btn_volumeup:hover {
94  border-image:url(:/icons/btn_volumeup2.png);
```

```
95  }
96
97  QSlider#durationSlider:handle:horizontal {
98  border-image:url(:/icons/handle.png);
99  }
100
101 QSlider#durationSlider {
102 border-image:none;
103 }
104
105 QSlider#durationSlider:add-page:horizontal {
106 border-image:url(:/images/add_page.png);
107 }
108
109 QSlider#volumeSlider {
110 border-image:none;
111 }
112 QSlider#volumeSlider:handle:horizontal {
113 border-image:url(:/icons/handle.png);
114 }
115
116 QSlider#volumeSlider:handle:horizontal {
117 background:transparent;
118 }
119
120 QSlider#volumeSlider:add-page:horizontal {
121 border-image:url(:/images/add_page.png);
122 }
123
124 QPushButton#btn_screen {
125 border-image:url(:/icons/btn_fullscreen1.png);
126 }
127
```

```
128 QPushButton#btn_screen:hover {
129 border-image:url(:/icons/btn_fullscreen2.png);
130 }
131
132 QPushButton#btn_screen:checked {
133 border-image:url(:/icons/btn_screen1.png);
134 }
135
136 QPushButton#btn_screen:checked:hover {
137 border-image:url(:/icons/btn_screen2.png);
138 }
```

12.4.2 程序运行效果

先点击构建项目，项目构建完成后，再将本例的 myVideo 视频文件夹拷贝到可执行程序的文件夹同一级目录下，也就是 build-15_videoplayer-Desktop_Qt_5_12_9_GCC_64bit-Debug 目录下。再重新运行，就出现视频文件在列表里，如下图，点击播放即可播放视频。

默认列表，未播放前。

开始播放，未全屏状态。

全屏状态。

12.5 录音

Qt 提供了 QAudioRecorder 类录制音频，继承于 QmediaRecorder 类，音频输入可以使用 QAudioRecorder 或者 QAudioInput 类实现。QAudioRecorder 是高级的类，输入的音频数据

直接保存为一个音频文件。而 QAudioInput 则是低层次的实现，从类的名称可以知道它是与输入输出流有关的，它可以将音频录制的数据写入一个流设备。本小节将介绍使用 QAudioRecorder 录制音频并保存成一个 mp3 文件。并使用 QAudioProbe 类探测音频数据缓冲区里的实时音量大小设计一个实用的录音应用程序。

12.5.1 应用实例

本例设计一个实用的录音界面，界面是编者原创界面。**本例适用于正点原子 ALPHA 开发板，已经测试。**Windows 与 Ubuntu 下请读者使用 Qt 官方的 audiorecorder 例子自行测试，Windows 系统上的声卡设置比较复杂，不详解，编者只确保正点原子 I.MX6U ALPHA 开发板正常运行此应用程序。Mini 板没有声卡，请使用 USB 声卡插到正点原子 I.MX6U 开发板进行自行测试!!!

本例目的：录音程序的设计与使用。

例 16_audiorecorder，录音程序（难度：难）。项目路径为 Qt/2/16_audiorecorder。注意本例有用到 qss 样式文件，关于如何添加资源文件与 qss 文件请参考 7.1.3 小节。本例设计一个录音程序，录音功能部分直接参考 Qt 官方的 audiorecorder 例程，界面设计由编者设计。在 Qt 官方的 audiorecorder 例程里（自行在 Qt 官方的 audiorecorder 例程里打开查看），我们可以看到官方的例程录音设置都是通过 QComboBox 来选择的，当然这个只是一个 Qt 官方的例子，有很大的参考性。如果运用到实际项目里我们需要做一定的修改。如果是面向用户，我们就不需要暴露那么信息给用户，同时也可以避免用户操作失败等等。所以编者参考 Qt 官方的例程重新设计了一个录音例程。源码如下，界面效果如 12.5.2 小节。

项目文件 16_audiorecorder 文件第一行添加的代码部分如下。

16_audiorecorder.pro 编程后的代码

```
1  QT       += core gui multimedia
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
```

```
12
13  # You can also make your code fail to compile if it uses deprecated
APIs.
14  # In order to do so, uncomment the following line.
15  # You can also select to disable deprecated APIs only up to a certain
version of Qt.
16  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0
17
18  SOURCES += \
19      main.cpp \
20      audiorecorder.cpp
21
22  HEADERS += \
23      audiorecorder.h
24
25  # Default rules for deployment.
26  qnx: target.path = /tmp/$${TARGET}/bin
27  else: unix:!android: target.path = /opt/$${TARGET}/bin
28  !isEmpty(target.path): INSTALLS += target
29
30  RESOURCES += \
31      res.qrc
```

在头文件“audiorecorder.h”具体代码如下。

audiorecorder.h 编程后的代码

```
/*****************************************************************
   Copyright (C) 2017 The Qt Company Ltd.
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   16_audiorecorder
   * @brief         audiorecorder.h
   * @author        Deng Zhimao
   * @email         1252699831@qq.com
```

```
    * @net           www.openedv.com
    * @date          2021-05-10

**************************************************************************/
1   #ifndef AUDIORECORDER_H
2   #define AUDIORECORDER_H
3
4   #include <QMainWindow>
5   #include <QListWidget>
6   #include <QVBoxLayout>
7   #include <QPushButton>
8   #include <QLabel>
9   #include <QAudioRecorder>
10  #include <QAudioProbe>
11  #include <QAudioBuffer>
12  #include <QMediaPlaylist>
13  #include <QMediaPlayer>
14  #include <QProgressBar>
15
16  /* 媒体信息结构体 */
17  struct MediaObjectInfo {
18      /* 用于保存视频文件名 */
19      QString fileName;
20      /* 用于保存视频文件路径 */
21      QString filePath;
22  };
23
24  class AudioRecorder : public QMainWindow
25  {
26      Q_OBJECT
27
28  public:
29      AudioRecorder(QWidget *parent = nullptr);
```

```
30      ~AudioRecorder();
31
32  private:
33      /* 布局初始化 */
34      void layoutInit();
35
36      /* 主 Widget*/
37      QWidget *mainWidget;
38
39      /* 录音列表 */
40      QListWidget *listWidget;
41
42      /* 底部的 Widget,用于存放按钮 */
43      QWidget *bottomWidget;
44
45      /* 中间的显示录制时长的 Widget 容器 */
46      QWidget *centerWidget;
47
48      /* 垂直布局 */
49      QVBoxLayout *vBoxLayout;
50
51      /* 录音 Level 布局 */
52      QHBoxLayout *levelHBoxLayout;
53
54      /* 水平布局 */
55      QHBoxLayout *hBoxLayout;
56
57      /* 录音按钮 */
58      QPushButton *recorderBt;
59
60      /* 上一首按钮 */
61      QPushButton *previousBt;
62
```

```
63      /* 下一首按钮 */
64      QPushButton *nextBt;
65
66      /* 删除按钮 */
67      QPushButton *removeBt;
68
69      /* 录音类 */
70      QAudioRecorder *m_audioRecorder = nullptr;
71
72      /* 用于探测缓冲区的 level */
73      QAudioProbe *m_probe = nullptr;
74
75      /* 扫描录音文件 */
76      void scanRecordFiles();
77
78      /* 录音设置容器，保存录音设备的可用信息，
79       * 本例使用默认的信息，即可录音 */
80      QList<QVariant>devicesVar;
81      QList<QVariant>codecsVar;
82      QList<QVariant>containersVar;
83      QList<QVariant>sampleRateVar;
84      QList<QVariant>channelsVar;
85      QList<QVariant>qualityVar;
86      QList<QVariant>bitratesVar;
87
88      /* 媒体播放器，用于播放视频 */
89      QMediaPlayer *recorderPlayer;
90
91      /* 媒体列表 */
92      QMediaPlaylist *mediaPlaylist;
93
94      /* 录音媒体信息存储 */
95      QVector<MediaObjectInfo> mediaObjectInfo;
```

```
96
97      /* 用于显示录音时长 */
98      QLabel *countLabel;
99
100     /* 用于显示录音 level，最多四通道 */
101     QProgressBar *progressBar[4];
102
103     /* 清空录音 level */
104     void clearAudioLevels();
105
106 private slots:
107     /* 点击录音按钮槽函数 */
108     void recorderBtClicked();
109
110     /* 播放列表点击 */
111     void listWidgetCliked(QListWidgetItem*);
112
113     /* 当前媒体状态改变 */
114     void mediaPlayerStateChanged(QMediaPlayer::State);
115
116     /* 媒体列表改变 */
117     void mediaPlaylistCurrentIndexChanged(int);
118
119     /* 当前列表项改变 */
120     void listWidgetCurrentItemChange(QListWidgetItem*,
121                                      QListWidgetItem*);
122
123     /* 上一首按钮点击 */
124     void previousBtClicked();
125
126     /* 下一首按钮点击 */
127     void nextBtClicked();
128
```

```
129     /* 删除按钮点击 */
130     void removeBtClicked();
131
132     /* 更新录音时长 */
133     void updateProgress(qint64);
134
135     /* 在列表里显示播放时间 */
136     void recorderPlayerPositionChanged(qint64);
137
138     /* 更新录音 level */
139     void processBuffer(const QAudioBuffer&);
140 };
141 #endif // AUDIORECORDER_H
```

头文件里主要是声明界面所使用的元素及一些槽函数。重点是 QAudioRecorder 与 QAudioProbe 对象的声明。

在源文件“audiorecorder.cpp”具体代码如下。

audiorecorder.cpp 编程后的代码

```
/*****************************************************************
   Copyright (C) 2017 The Qt Company Ltd.
   Copyright © Deng Zhimao Co., Ltd. 1990-2021. All rights reserved.
   * @projectName   16_audiorecorder
   * @brief        audiorecorder.cpp
   * @author       Deng Zhimao
   * @email        1252699831@qq.com
   * @net          www.openedv.com
   * @date         2021-05-10

*****************************************************************/
1   #include "audiorecorder.h"
2   #include <QDebug>
3   #include <QUrl>
4   #include <QDateTime>
```

```
5   #include <QDir>
6   #include <QCoreApplication>
7
8   static qreal getPeakValue(const QAudioFormat &format);
9   static QVector<qreal> getBufferLevels(const QAudioBuffer &buffer);
10
11  template <class T>
12  static QVector<qreal> getBufferLevels(const T *buffer, int frames,
int channels);
13
14  AudioRecorder::AudioRecorder(QWidget *parent)
15      : QMainWindow(parent)
16  {
17      /* 初始化布局 */
18      layoutInit();
19
20      /* 录制音频的类 */
21      m_audioRecorder = new QAudioRecorder(this);
22
23      /* 用于探测缓冲区的数据 */
24      m_probe = new QAudioProbe(this);
25
26      /* 信号槽连接，更新录音 level 显示 */
27      connect(m_probe, &QAudioProbe::audioBufferProbed,
28              this, &AudioRecorder::processBuffer);
29
30      /* 设置探测的对象 */
31      m_probe->setSource(m_audioRecorder);
32
33      /* 播放器 */
34      recorderPlayer = new QMediaPlayer(this);
35
36      /* 播放列表 */
```

```
37      mediaPlaylist = new QMediaPlaylist(this);
38
39      recorderPlayer->setPlaylist(mediaPlaylist);
40
41      /* 设置播放模式，默认是列表播放 */
42
mediaPlaylist->setPlaybackMode(QMediaPlaylist::CurrentItemOnce);
43
44      /* 扫描本地声卡设备 */
45      devicesVar.append(QVariant(QString()));
46      for (auto &device: m_audioRecorder->audioInputs()) {
47          devicesVar.append(QVariant(device));
48          //qDebug()<<"本地声卡设备："<<device<<endl;
49      }
50
51      /* 音频编码 */
52      codecsVar.append(QVariant(QString()));
53      for (auto &codecName: m_audioRecorder->supportedAudioCodecs()) {
54          codecsVar.append(QVariant(codecName));
55          //qDebug()<<"音频编码："<<codecName<<endl;
56      }
57
58      /* 容器/支持的格式 */
59      containersVar.append(QVariant(QString()));
60      for (auto &containerName:
m_audioRecorder->supportedContainers()) {
61          containersVar.append(QVariant(containerName));
62          //qDebug()<<"支持的格式："<<containerName<<endl;
63      }
64
65      /* 采样率 */
66      sampleRateVar.append(QVariant(0));
```

```
67      for (int sampleRate:
m_audioRecorder->supportedAudioSampleRates()) {
68          sampleRateVar.append(QVariant(sampleRate));
69          //qDebug()<<"采样率: "<<sampleRate<<endl;
70      }
71
72      /* 通道 */
73      channelsVar.append(QVariant(-1));
74      channelsVar.append(QVariant(1));
75      channelsVar.append(QVariant(2));
76      channelsVar.append(QVariant(4));
77
78      /* 质量 */
79      qualityVar.append(QVariant(int(QMultimedia::LowQuality)));
80      qualityVar.append(QVariant(int(QMultimedia::NormalQuality)));
81      qualityVar.append(QVariant(int(QMultimedia::HighQuality)));
82
83      /* 比特率 */
84      bitratesVar.append(QVariant(0));
85      bitratesVar.append(QVariant(32000));
86      bitratesVar.append(QVariant(64000));
87      bitratesVar.append(QVariant(96000));
88      bitratesVar.append(QVariant(128000));
89
90      /* 初始化时扫描已经录制的录音 mp3 文件 */
91      scanRecordFiles();
92
93      /* 录音类信号槽连接 */
94      connect(m_audioRecorder, &QAudioRecorder::durationChanged,
95              this, &AudioRecorder::updateProgress);
96
97      /* 列表信号槽连接 */
98      connect(listWidget, SIGNAL(itemClicked(QListWidgetItem*)),
```

```
99            this, SLOT(listWidgetCliked(QListWidgetItem*)));
100     connect(listWidget,
SIGNAL(currentItemChanged(QListWidgetItem*,
101                                               QListWidgetItem*)),
102             this,
SLOT(listWidgetCurrentItemChange(QListWidgetItem*,
103                                                QListWidgetItem*)));
104
105     /* 媒体连接信号槽 */
106     connect(recorderPlayer,
107             SIGNAL(stateChanged(QMediaPlayer::State)),
108             this,
109             SLOT(mediaPlayerStateChanged(QMediaPlayer::State)));
110     connect(mediaPlaylist,
111             SIGNAL(currentIndexChanged(int)),
112             this,
113             SLOT(mediaPlaylistCurrentIndexChanged(int)));
114     connect(recorderPlayer, SIGNAL(positionChanged(qint64)),
115             this,
116             SLOT(recorderPlayerPositionChanged(qint64)));
117
118     /* 按钮 */
119     connect(recorderBt, SIGNAL(clicked()), this,
SLOT(recorderBtClicked()));
120     connect(nextBt, SIGNAL(clicked()), this,
SLOT(nextBtClicked()));
121     connect(previousBt, SIGNAL(clicked()), this,
SLOT(previousBtClicked()));
122     connect(removeBt, SIGNAL(clicked()), this,
SLOT(removeBtClicked()));
123 }
124
125 AudioRecorder::~AudioRecorder()
```

```
126 {
127 }
128
129 void AudioRecorder::layoutInit()
130 {
131     this->setGeometry(0, 0, 800, 480);
132
133     mainWidget = new QWidget();
134     setCentralWidget(mainWidget);
135
136     vBoxLayout = new QVBoxLayout();
137     bottomWidget = new QWidget();
138     listWidget = new QListWidget();
139     listWidget->setFocusPolicy(Qt::NoFocus);
140     listWidget->setVerticalScrollBarPolicy(
141                 Qt::ScrollBarAlwaysOff);
142     listWidget->setHorizontalScrollBarPolicy(
143                 Qt::ScrollBarAlwaysOff);
144
145     /* 垂直布局 */
146     vBoxLayout->addWidget(listWidget);
147     vBoxLayout->addWidget(bottomWidget);
148     vBoxLayout->setContentsMargins(0, 0, 0, 0);
149     mainWidget->setLayout(vBoxLayout);
150
151     bottomWidget->setMinimumHeight(80);
152     bottomWidget->setMaximumHeight(80);
153     bottomWidget->setStyleSheet("background:#cccccc");
154
155     /* 水平布局 */
156     hBoxLayout = new QHBoxLayout();
157
158     /* 按钮，录音、上一首、下一首、删除项按钮 */
```

```
159     recorderBt = new QPushButton();
160     previousBt = new QPushButton();
161     nextBt = new QPushButton();
162     removeBt = new QPushButton();
163 
164     recorderBt->setCheckable(true);
165     recorderBt->setObjectName("recorderBt");
166     recorderBt->setFocusPolicy(Qt::NoFocus);
167     recorderBt->setMaximumSize(60, 60);
168     recorderBt->setMinimumSize(60, 60);
169 
170     hBoxLayout->setContentsMargins(0, 0, 0, 0);
171 
172     bottomWidget->setLayout(hBoxLayout);
173     hBoxLayout->addWidget(recorderBt);
174     hBoxLayout->addWidget(previousBt);
175     hBoxLayout->addWidget(nextBt);
176     hBoxLayout->addWidget(removeBt);
177 
178     nextBt->setMaximumSize(50, 50);
179     removeBt->setMaximumSize(50, 50);
180     previousBt->setMaximumSize(50, 50);
181 
182     previousBt->setObjectName("previousBt");
183     removeBt->setObjectName("removeBt");
184     nextBt->setObjectName("nextBt");
185 
186     previousBt->setFocusPolicy(Qt::NoFocus);
187     removeBt->setFocusPolicy(Qt::NoFocus);
188     nextBt->setFocusPolicy(Qt::NoFocus);
189 
190     /* 显示录音时长与录音 Level */
191     centerWidget = new QWidget(this);
```

```
192     centerWidget->setGeometry(width()/ 2 - 150,
193                               height() /2 - 100,
194                               300,
195                               200);
196     centerWidget->setStyleSheet("QWidget { background:#8823242a;"
197                                 "border-radius:10px}");
198     countLabel = new QLabel(centerWidget);
199     countLabel->setGeometry(0,
200                             0,
201                             300,
202                             50);
203     countLabel->setStyleSheet("QLabel {font-size:
30px;color:#eeeeee;"
204                               "font: blod;background:transparent}");
205     countLabel->setAlignment(Qt::AlignCenter);
206     levelHBoxLayout = new QHBoxLayout();
207
208     for (int i = 0; i < 4; i++) {
209         progressBar[i] = new QProgressBar();
210         progressBar[i]->setOrientation(Qt::Vertical);
211         progressBar[i]->setRange(0, 100);
212         progressBar[i]->setVisible(false);
213         progressBar[i]->setMaximumWidth(centralWidget()->width());
214         levelHBoxLayout->addWidget(progressBar[i]);
215         levelHBoxLayout->setContentsMargins(5, 50, 5, 5);
216         progressBar[i]->setStyleSheet("QWidget
{ background:#22eeeeee;"
217                                       "border-radius:0px}");
218     }
219     centerWidget->setLayout(levelHBoxLayout);
220     centerWidget->hide();
221     countLabel->raise();
222
```

```
223
224 }
225
226 void AudioRecorder::recorderBtClicked()
227 {
228     /* 录音前停止正在播放的媒体 */
229     if (recorderPlayer->state() != QMediaPlayer::StoppedState)
230         recorderPlayer->stop();
231     /* 如果录音已经停止，则开始录音 */
232     if (m_audioRecorder->state() == QMediaRecorder::StoppedState) {
233         /* 设置默认的录音设备 */
234
m_audioRecorder->setAudioInput(devicesVar.at(0).toString());
235
236         /* 下面的是录音设置，都是选择默认,可根据录音可用项，自行修改 */
237         QAudioEncoderSettings settings;
238         settings.setCodec(codecsVar.at(0).toString());
239         settings.setSampleRate(sampleRateVar[0].toInt());
240         settings.setBitRate(bitratesVar[0].toInt());
241         settings.setChannelCount(channelsVar[0].toInt());
242         settings.setQuality(QMultimedia::EncodingQuality(
243                                 qualityVar[0].toInt()));
244         /* 以恒定的质量录制，可选恒定的比特率 */
245
settings.setEncodingMode(QMultimedia::ConstantQualityEncoding);
246         QString container = containersVar.at(0).toString();
247         m_audioRecorder->setEncodingSettings(settings,
248                                              QVideoEncoderSettings(),
249                                              container);
250         m_audioRecorder->setOutputLocation(
251                   QUrl::fromLocalFile(tr("./Sounds/%1.mp3")
252                                       .arg(QDateTime::currentDateTime(
)
```

```
253                                         .toString()));
254         /* 开始录音 */
255         m_audioRecorder->record();
256         /* 显示录制时长标签 */
257         countLabel->clear();
258         centerWidget->show();
259     } else {
260         /* 停止录音 */
261         m_audioRecorder->stop();
262         /* 重设录音 level */
263         clearAudioLevels();
264         /* 隐藏录制时长标签 */
265         centerWidget->hide();
266         /* 重新扫描录音文件 */
267         scanRecordFiles();
268     }
269 }
270
271 void AudioRecorder::scanRecordFiles()
272 {
273     mediaPlaylist->clear();
274     listWidget->clear();
275     mediaObjectInfo.clear();
276     /* 录音文件保存在当前 Sounds 文件夹下 */
277     QDir dir(QCoreApplication::applicationDirPath()
278             + "/Sounds");
279     QDir dirbsolutePath(dir.absolutePath());
280
281     /* 如果文件夹不存在，则创建一个 */
282     if(!dirbsolutePath.exists())
283         dirbsolutePath.mkdir(dirbsolutePath.absolutePath());
284
285     /* 定义过滤器 */
```

```
286     QStringList filter;
287     /* 包含所有 xx 后缀的文件 */
288     filter<<"*.mp3";
289     /* 获取该目录下的所有文件 */
290     QFileInfoList files =
291             dirbsolutePath.entryInfoList(filter, QDir::Files);
292     /* 遍历 */
293     for (int i = 0; i < files.count(); i++) {
294         MediaObjectInfo info;
295         /* 使用 utf-8 编码 */
296         info.fileName = QString::fromUtf8(files.at(i)
297                                           .fileName()
298                                           .toUtf8()
299                                           .data());
300         info.filePath = QString::fromUtf8(files.at(i)
301                                           .filePath()
302                                           .toUtf8()
303                                           .data());
304         /* 媒体列表添加音频 */
305         if (mediaPlaylist->addMedia(
306                     QUrl::fromLocalFile(info.filePath))) {
307             /* 添加到容器数组里储存 */
308             mediaObjectInfo.append(info);
309             /* 添加音频名字至列表 */
310             listWidget->addItem(
311                         new
QListWidgetItem(QIcon(":/icons/play.png"),
312                                             info.fileName));
313         } else {
314             qDebug()<<
315                       mediaPlaylist->errorString()
316                       .toUtf8().data()
317                   << endl;
```

```
318             qDebug()<< "  Error number:"
319                     << mediaPlaylist->error()
320                     << endl;
321         }
322     }
323 }
324
325 void AudioRecorder::listWidgetCliked(QListWidgetItem *item)
326 {
327     /* item->setIcon 为设置列表里的图标状态 */
328     for (int i = 0; i < listWidget->count(); i++) {
329         listWidget->item(i)->setIcon(QIcon(":/icons/play.png"));
330     }
331
332     if (recorderPlayer->state() != QMediaPlayer::PlayingState) {
333         recorderPlayer->play();
334         item->setIcon(QIcon(":/icons/pause.png"));
335     } else {
336         recorderPlayer->pause();
337         item->setIcon(QIcon(":/icons/play.png"));
338     }
339 }
340
341 void AudioRecorder::listWidgetCurrentItemChange(
342         QListWidgetItem *currentItem,
343         QListWidgetItem *previousItem)
344 {
345     if (mediaPlaylist->mediaCount() == 0)
346         return;
347
348     if (listWidget->row(previousItem) != -1)
349         previousItem->setText(mediaObjectInfo
350                               .at(listWidget->row(previousItem))
```

```
351                             .fileName);
352
353     /* 先暂停播放媒体 */
354     if (recorderPlayer->state() == QMediaPlayer::PlayingState)
355         recorderPlayer->pause();
356
357     /* 设置当前媒体 */
358     mediaPlaylist->
359             setCurrentIndex(listWidget->row(currentItem));
360 }
361
362 void AudioRecorder::mediaPlayerStateChanged(
363         QMediaPlayer::State
364         state)
365 {
366     for (int i = 0; i < listWidget->count(); i++) {
367         listWidget->item(i)
368                 ->setIcon(QIcon(":/icons/play.png"));
369     }
370
371     /* 获取当前项，根据当前媒体的状态，然后设置不同的图标 */
372     if (mediaPlaylist->currentIndex() == -1)
373         return;
374     QListWidgetItem *item = listWidget->item(
375                 mediaPlaylist->currentIndex());
376
377     switch (state) {
378     case QMediaPlayer::PausedState:
379     case QMediaPlayer::PlayingState:
380         item->setIcon(QIcon(":/icons/pause.png"));
381         break;
382     case QMediaPlayer::StoppedState:
383         item->setIcon(QIcon(":/icons/play.png"));
```

```
384         break;
385     }
386 }
387 
388 void AudioRecorder::mediaPlaylistCurrentIndexChanged(
389         int index)
390 {
391     if (-1 == index)
392         return;
393 }
394 
395 void AudioRecorder::previousBtClicked()
396 {
397     /* 上一首操作 */
398     recorderPlayer->stop();
399     int count = listWidget->count();
400     if (0 == count)
401         return;
402     if (listWidget->currentRow() == -1)
403         listWidget->setCurrentRow(0);
404     else {
405         if (listWidget->currentRow() - 1 != -1)
406             listWidget->setCurrentRow(
407                     listWidget->currentRow() - 1);
408         else
409             listWidget->setCurrentRow(listWidget->count() - 1);
410     }
411     mediaPlaylist->setCurrentIndex(listWidget->currentRow());
412     recorderPlayer->play();
413 }
414 
415 void AudioRecorder::nextBtClicked()
416 {
```

```
417     /* 下一首操作 */
418     recorderPlayer->stop();
419
420     /* 获取列表的总数目 */
421     int count = listWidget->count();
422
423     /* 如果列表的总数目为 0 则返回 */
424     if (0 == count)
425         return;
426
427     if (listWidget->currentRow() == -1)
428         listWidget->setCurrentRow(0);
429     else {
430         if (listWidget->currentRow() + 1 < listWidget->count())
431             listWidget->setCurrentRow(
432                         listWidget->currentRow() + 1);
433         else
434             listWidget->setCurrentRow(0);
435     }
436     mediaPlaylist->setCurrentIndex(listWidget->currentRow());
437     recorderPlayer->play();
438 }
439
440 void AudioRecorder::removeBtClicked()
441 {
442     int index = listWidget->currentRow();
443     if (index == -1)
444         return;
445
446     /* 移除媒体的项 */
447     mediaPlaylist->removeMedia(index);
448
449     /* 指向要删除的文件 */
```

```
450     QFile file(mediaObjectInfo.at(index).filePath);
451
452     /* 移除录音文件 */
453     file.remove();
454
455     /* 删除列表选中的项 */
456     listWidget->takeItem(index);
457
458     /* 删除后设置当前项为删除项的前一个 */
459     if (index - 1 != -1)
460         listWidget->setCurrentRow(index - 1);
461 }
462
463 void AudioRecorder::updateProgress(qint64 duration)
464 {
465     if (m_audioRecorder->error()
466             != QMediaRecorder::NoError)
467         return;
468
469     /* 显示录制时长 */
470     countLabel->setText(tr("已录制 %1 s")
471                         .arg(duration / 1000));
472 }
473
474 void AudioRecorder::recorderPlayerPositionChanged(
475         qint64 position)
476 {
477     /* 格式化时间 */
478     int p_second  = position / 1000;
479     int p_minute = p_second / 60;
480     p_second %= 60;
481
482     QString mediaPosition;
```

```
483     mediaPosition.clear();
484
485     if (p_minute >= 10)
486         mediaPosition = QString::number(p_minute, 10);
487     else
488         mediaPosition = "0" + QString::number(p_minute, 10);
489
490     if (p_second >= 10)
491         mediaPosition = mediaPosition
492                 + ":" + QString::number(p_second, 10);
493     else
494         mediaPosition = mediaPosition
495                 + ":0" + QString::number(p_second, 10);
496
497
498     int d_second  =  recorderPlayer->duration() / 1000;
499     int d_minute = d_second / 60;
500     d_second %= 60;
501
502     QString mediaDuration;
503     mediaDuration.clear();
504
505     if (d_minute >= 10)
506         mediaDuration = QString::number(d_minute, 10);
507     else
508         mediaDuration = "0" + QString::number(d_minute, 10);
509
510     if (d_second >= 10)
511         mediaDuration = mediaDuration
512                 + ":" + QString::number(d_second, 10);
513     else
514         mediaDuration = mediaDuration
515                 + ":0" + QString::number(d_second, 10);
```

```
516
517     QString fileNmae = mediaObjectInfo
518             .at(listWidget->currentRow()).fileName + "\t";
519     /* 显示媒体总长度时间与播放的当前位置 */
520     listWidget->currentItem()->setText(fileNmae
521                                        + mediaPosition
522                                        +"/" + mediaDuration);
523 }
524
525 void AudioRecorder::clearAudioLevels()
526 {
527     for (int i = 0; i < 4; i++)
528         progressBar[i]->setValue(0);
529 }
530
531 // This function returns the maximum possible sample value for a given
audio format
532 qreal getPeakValue(const QAudioFormat& format)
533 {
534     // Note: Only the most common sample formats are supported
535     if (!format.isValid())
536         return qreal(0);
537
538     if (format.codec() != "audio/pcm")
539         return qreal(0);
540
541     switch (format.sampleType()) {
542     case QAudioFormat::Unknown:
543         break;
544     case QAudioFormat::Float:
545         if (format.sampleSize() != 32) // other sample formats are not
supported
546             return qreal(0);
```

```
547         return qreal(1.00003);
548     case QAudioFormat::SignedInt:
549         if (format.sampleSize() == 32)
550             return qreal(INT_MAX);
551         if (format.sampleSize() == 16)
552             return qreal(SHRT_MAX);
553         if (format.sampleSize() == 8)
554             return qreal(CHAR_MAX);
555         break;
556     case QAudioFormat::UnSignedInt:
557         if (format.sampleSize() == 32)
558             return qreal(UINT_MAX);
559         if (format.sampleSize() == 16)
560             return qreal(USHRT_MAX);
561         if (format.sampleSize() == 8)
562             return qreal(UCHAR_MAX);
563         break;
564     }
565
566     return qreal(0);
567 }
568
569 // returns the audio level for each channel
570 QVector<qreal> getBufferLevels(const QAudioBuffer& buffer)
571 {
572     QVector<qreal> values;
573
574     if (!buffer.format().isValid() || buffer.format().byteOrder() !=
QAudioFormat::LittleEndian)
575         return values;
576
577     if (buffer.format().codec() != "audio/pcm")
578         return values;
```

```
579
580     int channelCount = buffer.format().channelCount();
581     values.fill(0, channelCount);
582     qreal peak_value = getPeakValue(buffer.format());
583     if (qFuzzyCompare(peak_value, qreal(0)))
584         return values;
585
586     switch (buffer.format().sampleType()) {
587     case QAudioFormat::Unknown:
588     case QAudioFormat::UnSignedInt:
589         if (buffer.format().sampleSize() == 32)
590             values = getBufferLevels(buffer.constData<quint32>(),
buffer.frameCount(), channelCount);
591         if (buffer.format().sampleSize() == 16)
592             values = getBufferLevels(buffer.constData<quint16>(),
buffer.frameCount(), channelCount);
593         if (buffer.format().sampleSize() == 8)
594             values = getBufferLevels(buffer.constData<quint8>(),
buffer.frameCount(), channelCount);
595         for (int i = 0; i < values.size(); ++i)
596             values[i] = qAbs(values.at(i) - peak_value / 2) /
(peak_value / 2);
597         break;
598     case QAudioFormat::Float:
599         if (buffer.format().sampleSize() == 32) {
600             values = getBufferLevels(buffer.constData<float>(),
buffer.frameCount(), channelCount);
601             for (int i = 0; i < values.size(); ++i)
602                 values[i] /= peak_value;
603         }
604         break;
605     case QAudioFormat::SignedInt:
606         if (buffer.format().sampleSize() == 32)
```

```
607             values = getBufferLevels(buffer.constData<qint32>(),
buffer.frameCount(), channelCount);
608         if (buffer.format().sampleSize() == 16)
609             values = getBufferLevels(buffer.constData<qint16>(),
buffer.frameCount(), channelCount);
610         if (buffer.format().sampleSize() == 8)
611             values = getBufferLevels(buffer.constData<qint8>(),
buffer.frameCount(), channelCount);
612         for (int i = 0; i < values.size(); ++i)
613             values[i] /= peak_value;
614         break;
615     }
616
617     return values;
618 }
619
620 template <class T>
621 QVector<qreal> getBufferLevels(const T *buffer, int frames, int
channels)
622 {
623     QVector<qreal> max_values;
624     max_values.fill(0, channels);
625
626     for (int i = 0; i < frames; ++i) {
627         for (int j = 0; j < channels; ++j) {
628             qreal value = qAbs(qreal(buffer[i * channels + j]));
629             if (value > max_values.at(j))
630                 max_values.replace(j, value);
631         }
632     }
633
634     return max_values;
635 }
```

```
636
637 void AudioRecorder::processBuffer(const QAudioBuffer& buffer)
638 {
639     /* 根据通道数目需要显示 count 个 level */
640     int count = buffer.format().channelCount();
641     for (int i = 0; i < 4; i++) {
642         if (i < count)
643             progressBar[i]->setVisible(true);
644         else
645             progressBar[i]->setVisible(false);
646     }
647
648     /* 设置 level 的值 */
649     QVector<qreal> levels = getBufferLevels(buffer);
650     for (int i = 0; i < levels.count(); ++i)
651         progressBar[i]->setValue(levels.at(i) * 100);
652 }
```

布局与播放录音部分的代码编者不再解释，与音乐播放器和视频播放器的原理一样。只是换了个样式而已。

第 21 行，初始化录音类对象，m_audioRecorder。录音的功能全靠这个类了，要完成录音的工作，我们只需要关注这个类。剩下的都是其他功能的实现。

第 44~88 行，本例将录音设置的参数，参数可以从 Qt 提供的 API 里如 supportedAudioCodecs（）表示可用支持的编码方式，详细请参考代码，全部使用 QVariant 容器来储存。这样可以不必要暴露太多接口给用户修改，以免出错。

第 222~269，是录音功能的重要代码，一般是通过 QAudioEncoderSettings 来设置输入音频设置，主要是**编码格式、采样率、通道数、音频质量**等设置（音频格式 Qt 提供了如 setCodes（）等方式可以直接设置，音频相关知识不探讨，我们只需要知道这个流程即可。）。这些参考 Qt 官方的 audiorecorder 例程，使用默认的设置，也就是 Default 项，Qt 自动选择系统音频输入设备输入，同时自动确定底层的采样参数等。在 Linux 里，Qt 扫描声卡的设备项很多，让用户选择可能会导致出错。所以编者测试使用默认的设置，即可录音，无需用户自行选择和修改。同时设置了录音保存的文件名为 xx.mp3。根据系统时间命名录音文件，如果不指定录音文件名，在 Linux 下一般保存为 clip_0001.mov，clip_0002.mov 等录音文件（mov 格式为苹果操作系统常用音视频格式）。Windows 一般保存为 clip_0001.wav 格式文件。

设置完成录音项后，使用 QAudioRecorder 的 record()、pause()和 stop()函数即可完成录音。本例没有使用 pause()也就是录音暂停，根据情景无需录音暂停。record()和 stop()是开始录音和录音停止。

第 23~28 行，QAudioProbe 类型的对象用于探测缓冲区的数据。setSource()是指定探测的对象。

第 525~652 行，这部分代码是直接拷贝 Qt 官方的代码进行修改，代码比较复杂（闲时自行理解），我们只需要知道它是从缓冲区里获取通道数与音频的实时音量。

整个代码主要完成录音的功能是 QAudioRecorder、QAudioEncoderSettings 和 QAudioProbe 类。其他代码可以没有也可以完成本例录音的功能。QAudioRecorder 负责录音，QAudioProbe 类负责获取通道数，与输入的实时音量。只要掌握了这两个类，设计一个好看的录音应用界面不在话下。

main.cpp 内容如下，主要是加载 qss 样式文件。

```
1   #include "audiorecorder.h"
2
3   #include <QApplication>
4   #include <QFile>
5
6   int main(int argc, char *argv[])
7   {
8       QApplication a(argc, argv);
9       /* 指定文件 */
10      QFile file(":/style.qss");
11
12      /* 判断文件是否存在 */
13      if (file.exists() ) {
14          /* 以只读的方式打开 */
15          file.open(QFile::ReadOnly);
16          /* 以字符串的方式保存读出的结果 */
17          QString styleSheet = QLatin1String(file.readAll());
18          /* 设置全局样式 */
19          qApp->setStyleSheet(styleSheet);
20          /* 关闭文件 */
21          file.close();
```

```
    }

    AudioRecorder w;
    w.show();
    return a.exec();
}
```

style.qss 样式文件如下。素材已经在源码处提供。注意下面的 style.qss 不能有注释！

```
QTabBar:tab {
height:0; width:0;
}

QWidget {
background:#e6e6e6;
}

QListWidget {
border:none;
}

QPushButton#recorderBt {
border-image:url(:/icons/recorder_stop1.png);
background:transparent;
}

QPushButton#recorderBt:hover {
border-image:url(:/icons/recorder_stop2.png);
}

QPushButton#recorderBt:checked {
border-image:url(:/icons/recorder_start1.png);
}

QPushButton#recorderBt:checked:hover {
```

```
27  border-image:url(:/icons/recorder_start2.png);
28  }
29
30  QListWidget {
31  color:black;
32  font-size: 20px;
33  border:none;
34  icon-size:40px;
35  }
36
37  QListWidget:item:active {
38  background: transparent;
39  }
40
41  QListWidget:item {
42  background: transparent;
43  height:60;
44  }
45
46  QListWidget:item:selected {
47  color:red;
48  background: transparent;
49  }
50
51  QListWidget:item:hover {
52  background: transparent;
53  color:red;
54  border:none;
55  }
56
57  QPushButton#nextBt {
58  border-image:url(:/icons/btn_next1.png);
59  }
```

```
60
61  QPushButton#nextBt:hover {
62  border-image:url(:/icons/btn_next2.png);
63  }
64
65  QPushButton#previousBt {
66  border-image:url(:/icons/btn_previous1.png);
67  }
68
69  QPushButton#previousBt:hover {
70  border-image:url(:/icons/btn_previous2.png);
71  }
72
73  QPushButton#removeBt {
74  border-image:url(:/icons/remove1.png);
75  }
76
77  QPushButton#removeBt:hover {
78  border-image:url(:/icons/remove2.png);
79  }
80
81  QProgressBar::chunk {
82  background-color: #f6f6f6;
83  height: 8px;
84  margin: 0.5px;
85  border-radius:0px;
86  }
87
88  QProgressBar {
89  color:transparent;
90  }
```

12.5.2 程序运行效果

本例适用于正点原子 I.MX6U ALPHA 开发板！请使用正点原子 **I.MX6U 的出厂系统**进行测试！

请使用正点原子的 I.MX6U 的出厂时的系统测试！

请使用正点原子的 I.MX6U 的出厂时的系统测试！

请使用正点原子的 I.MX6U 的出厂时的系统测试！

重要的事情是说三遍！

开始录音前，需要根据正点原子 I.MX6U 用户快速体验手册，第 3.15 小节进行测试板子的录音功能。确保能正常录音，再交叉编译此 Qt 应用程序到开发板上运行。如何交叉编译 Qt 应用程序到开发板，请看【正点原子】I.MX6U 出厂系统 Qt 交叉编译环境搭建 V1.x 版本。

在正点原子 I.MX6U 开发板上运行此录音程序，需要先配置是麦克风（板子上的麦头）或者是 Line_in 输入方式。

如果是麦头录音，则在板子上运行开启麦头录音的脚本。

```
/home/root/shell/audio/mic_in_config.sh
```

如果是 Line_in 的方式录音，请使用一条 3.5mm 的两头公头的音频线，一头对板子上 Line_in 接口。另一头连接手机或者电脑的音频输出设备。手机或者电脑开始播放音乐，音乐尽量调高！执行下面的脚本，开启板子以 line_in 的方式录音。

```
/home/root/shell/audio/line_in_config.sh
```

点击左下角的录音按钮开始录音（Ubuntu 上模拟效果图）。可以看到两个通道的实时音量柱状图，用于显示实时音量输入的大小。

开如播放录音。（Ubuntu 上模拟效果图）。