# 8 媒体播放器

## ~/QtPlayer/Src/MainWindow.cpp

```cpp
#include <QMessageBox>
#include <QTime>
#include <QKeyEvent>
#include <string>
#include <vector>
using namespace std;
#include "MainWindow.h"
#include "ui_MainWindow.h"

// 构造函数
MainWindow::MainWindow(QWidget *parent, bool embed)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
    , embed(embed)
    , vlcInstance(NULL)
    , vlcMediaPlayer(NULL)
    , vlcEventManager(NULL)
{
    ui->setupUi(this);

    // 设置VLC插件路径环境变量
#ifndef Q_OS_WINDOWS
    if (setenv("VLC_PLUGIN_PATH", "/usr/lib/vlc/plugins", 0) == -1)
    {
        QMessageBox(QMessageBox::Critical, windowTitle(),
            "无法设置VLC插件路径环境变量！", QMessageBox::Ok, this).exec();
        return;
    }
#endif // Q_OS_WINDOWS

    // 创建VLC实例
    if (!(vlcInstance = libvlc_new(0, NULL)))
    {
        QMessageBox(QMessageBox::Critical, windowTitle(),
            "无法创建VLC实例！", QMessageBox::Ok, this).exec();
        return;
    }

    // 创建VLC媒体播放器
    if (!(vlcMediaPlayer = libvlc_media_player_new(vlcInstance)))
    {
        QMessageBox(QMessageBox::Critical, windowTitle(),
            "无法创建VLC媒体播放器！", QMessageBox::Ok, this).exec();
        return;
    }

    // 若内嵌视频...
    if (embed)
    {
```

```
        // 将视频框控件设置为VLC媒体播放器的输出窗口
#ifdef Q_OS_WINDOWS
        libvlc_media_player_set_hwnd(
            vlcMediaPlayer, reinterpret_cast<void*>(ui->frmVideo->winId()));
#else
        libvlc_media_player_set_xwindow(vlcMediaPlayer, ui->frmVideo->winId());
#endif // Q_OS_WINDOWS

        // 为视频框控件安装事件过滤器，以截获鼠标双击事件，切换全屏和窗口模式
        ui->frmVideo->installEventFilter(this);
    }
    else // 否则...
        // 销毁屏幕框控件(及其子控件----视频框控件)
        delete ui->frmScreen;

    // 获取VLC事件管理器
    if (!(vlcEventManager = libvlc_media_player_event_manager(vlcMediaPlayer)))
    {
        QMessageBox(QMessageBox::Critical, windowTitle(),
            "无法获取VLC事件管理器！", QMessageBox::Ok, this).exec();
        return;
    }

    // 设置VLC事件回调

    // 响应媒体长度改变事件----设置进度滑块控件
    libvlc_event_attach(
        vlcEventManager, libvlc_MediaPlayerLengthChanged,
        onMediaPlayerLengthChanged, this);
#ifdef Q_OS_WINDOWS
    // 若内嵌视频...
    if (embed)
        // 响应视频输出启动事件----枚举视频框控件的所有子窗口并禁用之
        // 在Windows上若不禁用视频框控件的子窗口，即VLC视频输出窗口，
        // 鼠标停留其上将导致界面崩溃
        libvlc_event_attach(
            vlcEventManager, libvlc_MediaPlayerVout,
            onMediaPlayerVout, this);
#endif // Q_OS_WINDOWS
    // 响应时间改变事件----同步进度滑块控件和进度标签控件
    libvlc_event_attach(
        vlcEventManager, libvlc_MediaPlayerTimeChanged,
        onMediaPlayerTimeChanged, this);
    // 响应终点到达事件----停止播放
    libvlc_event_attach(
        vlcEventManager, libvlc_MediaPlayerEndReached,
        onMediaPlayerEndReached, this);

    // 初始化界面
    ui->editURL->setEnabled(true);
    ui->btnPlay->setEnabled(true);

    // 初始化音量
```

```
    libvlc_audio_set_volume(vlcMediaPlayer, ui->sliderVolume->value());
}

// 析构函数
MainWindow::~MainWindow()
{
    // 销毁VLC媒体播放器
    if (vlcMediaPlayer)
    {
        libvlc_media_player_release(vlcMediaPlayer);
        vlcMediaPlayer = NULL;
    }

    // 销毁VLC实例
    if (vlcInstance)
    {
        libvlc_release(vlcInstance);
        vlcInstance = NULL;
    }

    delete ui;
}

// 响应媒体长度改变事件的静态处理函数
void MainWindow::onMediaPlayerLengthChanged(
    libvlc_event_t const* event, void* data)
{
    // 调用普通处理函数
    reinterpret_cast<MainWindow*>(data)->onMediaPlayerLengthChanged(
        event->u.media_player_length_changed.new_length);
}

// 响应媒体长度改变事件的普通处理函数
void MainWindow::onMediaPlayerLengthChanged(libvlc_time_t length)
{
    // 将媒体长度毫秒值设置为进度滑块控件的最大值
    ui->sliderProgress->setMaximum(length);
    // 将媒体长度毫秒值的十分之一设置为进度滑块控件的页步距
    ui->sliderProgress->setPageStep(length / 10);
    // 将媒体长度毫秒值的百分之一设置为进度滑块控件的刻度间隔
    ui->sliderProgress->setTickInterval(length / 100);
    // 将媒体长度毫秒值保存到成员变量中以备后用
    vlcMediaLength = length;
}

#ifdef Q_OS_WINDOWS

// 子窗口枚举回调函数
BOOL CALLBACK MainWindow::onEnumVLCWindow(HWND hwnd, LPARAM)
{
    // 禁用该子窗口，即VLC视频输出窗口，避免鼠标停留引发界面崩溃
    EnableWindow(hwnd, FALSE);
    return TRUE;
}

// 响应视频输出启动事件的静态处理函数
```

```
159 void MainWindow::onMediaPlayerVout(
160     libvlc_event_t const*, void* data)
161 {
162     // 调用普通处理函数
163     reinterpret_cast<MainWindow*>(data)->onMediaPlayerVout();
164 }
165 
166 // 响应视频输出启动事件的普通处理函数
167 void MainWindow::onMediaPlayerVout(void)
168 {
169     // 枚举视频框控件的所有子窗口，即VLC视频输出窗口
170     // 每个子窗口的句柄将作为参数，传递给枚举回调函数
171     EnumChildWindows((HWND)ui->frmVideo->winId(), onEnumVLCWindow, 0);
172 }
173 
174 #endif // Q_OS_WINDOWS
175 
176 // 响应时间改变事件的静态处理函数
177 void MainWindow::onMediaPlayerTimeChanged(
178     libvlc_event_t const* event, void* data)
179 {
180     // 调用普通处理函数
181     reinterpret_cast<MainWindow*>(data)->onMediaPlayerTimeChanged(
182         event->u.media_player_time_changed.new_time);
183 }
184 
185 // 响应时间改变事件的普通处理函数
186 void MainWindow::onMediaPlayerTimeChanged(libvlc_time_t time)
187 {
188     // 根据媒体时间毫秒值同步进度滑块控件
189     ui->sliderProgress->setValue(time);
190     // 根据媒体时间毫秒值同步进度标签控件
191     ui->labProgress->setText(
192         QTime::fromMSecsSinceStartOfDay(time).toString("hh:mm:ss.zzz") +
    "/" +
193 
     QTime::fromMSecsSinceStartOfDay(vlcMediaLength).toString("hh:mm:ss.zzz"));
194 }
195 
196 // 响应终点到达事件的静态处理函数
197 void MainWindow::onMediaPlayerEndReached(
198     libvlc_event_t const*, void* data)
199 {
200     // 调用普通处理函数
201     reinterpret_cast<MainWindow*>(data)->onMediaPlayerEndReached();
202 }
203 
204 // 响应终点到达事件的普通处理函数
205 void MainWindow::onMediaPlayerEndReached(void)
206 {
207     // 向停止按钮发射点击信号
208     emit ui->btnStop->click();
209 }
210 
211 // 显示窗口时被调用的虚函数
212 void MainWindow::showEvent(QShowEvent*)
```

```
{
    // 若非内嵌视频...
    if (!embed)
        // 将窗口收缩至最小
        resize(minimumSize());
}

// 视频框控件有事件时被调用的虚函数
bool MainWindow::eventFilter(QObject* obj, QEvent* event)
{
    // 若视频框控件发生鼠标双击事件...
    if (obj == ui->frmVideo && event->type() ==
QEvent::MouseButtonDblClick)
    {
        static bool     fullscreen = false; // 全屏模式
#ifdef Q_OS_WINDOWS
        static int      minHeight;          // 最小高度
        static QSize    winSize;            // 窗口大小
#else
        static QMargins cntMargins;         // 中心布局的边缘大小
        static int      frmStyle;           // 屏幕框控件的边框风格
#endif // Q_OS_WINDOWS

        if (fullscreen) // 全屏->窗口
        {
#ifdef Q_OS_WINDOWS
            // 调整界面
            setMinimumHeight(minHeight);
            resize(winSize);
            ui->frmScreen->show();
            // 退出全屏
            ui->frmVideo->setWindowFlags(Qt::SubWindow);
            ui->frmVideo->showNormal();
#else
            // 调整界面
            ui->labFullscreen->show();
            ui->frmControl->show();
            ui->layoutCentral->setContentsMargins(cntMargins);
            ui->frmScreen->setFrameStyle(frmStyle);
            // 退出全屏
            showNormal();
#endif // Q_OS_WINDOWS
            // 恢复光标
            ui->frmVideo->unsetCursor();
            // 激活窗口
            qApp->setActiveWindow(this);
        }
        else // 窗口->全屏
        {
            // 隐藏光标
            ui->frmVideo->setCursor(Qt::BlankCursor);
#ifdef Q_OS_WINDOWS
            // 进入全屏
            ui->frmVideo->setWindowFlags(Qt::Window);
            ui->frmVideo->showFullScreen();
            // 调整界面
```

```
            ui->frmScreen->hide();
            minHeight = minimumHeight();
            winSize = size();
            setMinimumHeight(
                ui->layoutCentral->contentsMargins().top() +
                ui->labFullscreen->size().height() +
                ui->layoutCentral->spacing() +
                ui->frmControl->size().height() +
                ui->layoutCentral->contentsMargins().bottom());
            resize(size().width(), minimumHeight());
#else
            // 进入全屏
            showFullScreen();
            // 调整界面
            ui->labFullscreen->hide();
            ui->frmControl->hide();
            cntMargins = ui->layoutCentral->contentsMargins();
            ui->layoutCentral->setContentsMargins(QMargins());
            frmStyle = ui->frmScreen->frameStyle();
            ui->frmScreen->setFrameStyle(QFrame::NoFrame);
#endif // Q_OS_WINDOWS
        }

        fullscreen = !fullscreen;
        return true;
    }

    return QMainWindow::eventFilter(obj, event);
}

// 响应进度滑块值改变信号的槽函数(拖拽滑块或点击滑轨)
void MainWindow::on_sliderProgress_valueChanged(int value)
{
    // 在响应时间改变事件的处理函数中，根据媒体时间的毫秒值调整进度
    // 滑块的位置，也会引发值改变信号，此处若不加判断势必构成死循环
    if (libvlc_media_player_get_time(vlcMediaPlayer) != value)
        // 定位VLC媒体
        libvlc_media_player_set_time(vlcMediaPlayer, value);
}

// 响应播放按钮点击信号的槽函数
void MainWindow::on_btnPlay_clicked()
{
    string url = ui->editURL->text().toStdString();

    vector<string> protocols;
    protocols.push_back("http");
    protocols.push_back("https");
    protocols.push_back("ftp");
    protocols.push_back("rstp");

    vector<string>::const_iterator it;
    for (it = protocols.begin(); it != protocols.end(); ++it)
        if (!url.compare(0, it->size(), *it))
            break;

```

```
    // 若网络媒体...
    if (it != protocols.end())
    {
        // 创建VLC媒体
        if (!(vlcMedia = libvlc_media_new_location(vlcInstance,
url.c_str())))
        {
            QMessageBox(QMessageBox::Critical, windowTitle(),
                "无法创建VLC媒体！", QMessageBox::Ok, this).exec();
            return;
        }
    }
    else // 否则，即本地媒体...
    {
        // 创建VLC媒体
        if (!(vlcMedia = libvlc_media_new_path(vlcInstance, url.c_str())))
        {
            QMessageBox(QMessageBox::Critical, windowTitle(),
                "无法创建VLC媒体！", QMessageBox::Ok, this).exec();
            return;
        }
    }

    // 将VLC媒体设置到VLC媒体播放器中
    libvlc_media_player_set_media(vlcMediaPlayer, vlcMedia);

    // 播放VLC媒体
    if (libvlc_media_player_play(vlcMediaPlayer) == -1)
    {
        QMessageBox(QMessageBox::Critical, windowTitle(),
            "无法播放VLC媒体！", QMessageBox::Ok, this).exec();
        return;
    }

    // 若内嵌视频...
    if (embed)
        // 禁止视频框控件更新显示，避免闪烁
        ui->frmVideo->setUpdatesEnabled(false);

    // 调整界面
    ui->sliderProgress->setEnabled(true);
    ui->btnPlay->setEnabled(false);
    ui->btnPause->setEnabled(true);
    ui->btnStop->setEnabled(true);
    ui->btnFastForward->setEnabled(true);
    ui->btnFastBackward->setEnabled(true);
}

// 响应暂停按钮点击信号的槽函数
void MainWindow::on_btnPause_clicked()
{
    if (libvlc_media_player_is_playing(vlcMediaPlayer))
    {
        // 暂停VLC媒体
        libvlc_media_player_pause(vlcMediaPlayer);
        ui->btnPause->setText("继续");
```

```
379     }
380     else
381     {
382         // 播放VLC媒体
383         libvlc_media_player_play(vlcMediaPlayer);
384         ui->btnPause->setText("暂停");
385     }
386 }
387 
388 // 响应停止按钮点击信号的槽函数
389 void MainWindow::on_btnStop_clicked()
390 {
391     // 停止VLC媒体
392     libvlc_media_player_stop(vlcMediaPlayer);
393 
394     // 若内嵌视频...
395     if (embed)
396         // 激活视频框控件更新显示
397         ui->frmVideo->setUpdatesEnabled(true);
398 
399     // 调整界面
400     ui->sliderProgress->setValue(0);
401     ui->sliderProgress->setEnabled(false);
402     ui->labProgress->setText("00:00:00.000/00:00:00.000");
403     ui->btnPlay->setEnabled(true);
404     ui->btnPause->setEnabled(false);
405     ui->btnPause->setText("暂停");
406     ui->btnStop->setEnabled(false);
407     ui->btnFastForward->setEnabled(false);
408     ui->btnFastBackward->setEnabled(false);
409 }
410 
411 // 响应快进按钮点击信号的槽函数
412 void MainWindow::on_btnFastForward_clicked()
413 {
414     // 向前步进媒体长度的百分之一
415     libvlc_time_t time = libvlc_media_player_get_time(vlcMediaPlayer) +
416         vlcMediaLength / 100;
417 
418     // 定位VLC媒体
419     libvlc_media_player_set_time(vlcMediaPlayer,
420         time < vlcMediaLength ? time : vlcMediaLength);
421 }
422 
423 // 响应快退按钮点击信号的槽函数
424 void MainWindow::on_btnFastBackward_clicked()
425 {
426     // 向后步进媒体长度的百分之一
427     libvlc_time_t time = libvlc_media_player_get_time(vlcMediaPlayer) -
428         vlcMediaLength / 100;
429 
430     // 定位VLC媒体
431     libvlc_media_player_set_time(vlcMediaPlayer,
432         time > 0 ? time : 0);
433 }
434 
```

```cpp
// 响应音量滑块移动信号的槽函数(拖拽滑块)
void MainWindow::on_sliderVolume_sliderMoved(int position)
{
    // 调整音量
    libvlc_audio_set_volume(vlcMediaPlayer, position);
    // 设置静音
    libvlc_audio_set_mute(vlcMediaPlayer, !position);
}

// 响应音量滑块值改变信号的槽函数(点击滑轨)
void MainWindow::on_sliderVolume_valueChanged(int value)
{
    if (libvlc_audio_get_volume(vlcMediaPlayer) != value)
    {
        // 调整音量
        libvlc_audio_set_volume(vlcMediaPlayer, value);
        // 设置静音
        libvlc_audio_set_mute(vlcMediaPlayer, !value);
    }
}

// 响应关于按钮点击信号的槽函数
void MainWindow::on_btnAbout_clicked()
{
    // 显示版本
    QMessageBox(QMessageBox::Information, windowTitle(),
        "基于Qt和libVLC的流媒体播放器\n\n"
        "版本：1.0\n\n"
        "版权所有 (C) 达内科技, 2020",
        QMessageBox::Ok, this).exec();
}

// 响应退出按钮点击信号的槽函数
void MainWindow::on_btnQuit_clicked()
{
    // 关闭窗口
    close();
}
```