第 12 章

多媒体应用

Qt 5 使用了全新的 Qt Multimedia 模块实现多媒体应用，Qt 4 中用来实现多媒体功能的 Phonon 模块已经被移除。新的 Qt Multimedia 模块提供了丰富的接口，可以轻松地使用平台的多媒体功能，例如进行媒体播放、使用相机和收音机等。这里分别提供了一组 QML 类型和一组 C++类来处理多媒体内容，本章只讲解 QML 中音视频播放的实现。

12.1 多媒体模块介绍

QML 中多媒体模块可以实现的功能、对应的示例程序以及需要使用的 QML 类型如表 12-1 所列。

表 12-1 多媒体功能及相关 QML 类型

功 能	示 例	QML 类型
播放编码音频(MP3、AAC 等)	player	Audio, MediaPlayer
播放视频	player, qmlvideo, qmlvideofx	MediaPlayer, VideoOutput, Video
处理视频	qmlvideofx	MediaPlayer, VideoOutput
收听录音机	declarative-radio	Radio, RadioData
访问相机取景器	camera, declarative-camera	Camera, VideoOutput
处理取景器		Camera, VideoOutput
拍摄照片	camera, declarative-camera	Camera
拍摄视频	camera, declarative-camera	Camera
3D 声源		Audio Engine

Qt 的多媒体接口建立在底层平台的多媒体框架之上，这就意味着对于各种编解码器的支持依赖于使用的平台。如果要访问一些平台相关的设置，或者将 Qt 多媒体接口移植到新的平台，可以参考 Qt 帮助中的 Multimedia Backend Development 文档。

要使用多媒体模块的内容，需要在 QML 文件中使用如下导入语句：

```
import QtMultimedia 5.0
```

关于本节的内容，可以在 Qt 帮助中查看 Qt Multimedia 关键字。

12.2 播放音频

12.2.1 播放压缩音频

Audio 类型提供了众多属性、函数和信号，用于播放压缩音频。要播放一个音频文件，最简单的方式是指定其 source 属性，然后设置 autoPlay 属性为 true，这样程序在运行后就可以自动播放音频了。

下面来看一个例子(项目源码路径：src\12\12-1\myaudio)：

```
import QtQuick 2.2
import QtMultimedia 5.0
Item {
    Audio {
        id: playMusic; source: "music.mp3"
        autoPlay: true
    }
}
```

Audio 类型可以通过 play()、pause()和 stop()等函数进行播放、暂停和停止等操作。当进行完这些操作时会发射相应的 playing()、paused()和 stopped()信号。使用 duration 属性可以获取音频的持续时间，单位是毫秒；使用 position 属性可以获取当前的播放位置；使用 seek()函数可以将播放位置设置为指定的值。使用 volume 属性可以获取或者设置音量的大小，取值范围是 0.0～1.0，默认值是 1.0。通过 loops 属性可以设置播放的循环次数：当设置为 0 或 1 时只会播放一次，当设置为 Audio.Infinite 时会无限循环播放，其默认值为 1。playbackRate 属性可以用来设置播放速率，设置的值为默认播放速率的倍数，默认值为 1.0。使用 metaData 分组属性可以获取媒体相关的信息，比如专辑的艺术家 metaData.albumArtist、专辑标题 metaData.albumTitle 等。Audio 类型的 status 属性保存了媒体加载的状态，可以取如下值：

- NoMedia：还没有设置媒体文件；
- Loading：媒体文件当前正在被加载；
- Loaded：媒体文件已经被加载；
- Buffering：指定的媒体文件是缓冲数据；
- Stalled：媒体文件时缓冲数据，播放已经被中断；
- Buffered：媒体包含缓冲数据；
- EndOfMedia：媒体文件已经播放到了末尾；
- InvalidMedia：媒体文件不能被播放；
- UnknownStatus：未知状态。

下面来看一个例子(项目源码路径：src\12\12-2\myaudio)：

```
import QtQuick 2.2
import QtMultimedia 5.0
import QtQuick.Controls 1.1
import QtQuick.Layouts 1.1
ApplicationWindow {
    id: window; width: 300; height: 300
    toolBar: ToolBar {
        RowLayout {
            anchors.fill: parent
            ToolButton { text: qsTr("播放"); onClicked: audio.play() }
            ToolButton { text: qsTr("暂停"); onClicked: audio.pause() }
            ToolButton { text: qsTr("停止"); onClicked: audio.stop() }
        }
    }
    ColumnLayout {
        spacing: 10
        Text { text: qsTr("进度:") }
        Slider { maximumValue: audio.duration; value: audio.position;
                 onValueChanged: audio.seek(value) }
        Text { text: qsTr("音量:") }
        Slider { maximumValue: 1.0; value: audio.volume;
                 onValueChanged: audio.volume = value }
        Text { text: qsTr("循环次数:") }
        SpinBox { value: 1; onValueChanged: audio.loops = value }
        Text { text: qsTr("播放速度:") }
        SpinBox { value: 1.0; decimals: 1; stepSize: 0.1
                  onValueChanged: audio.playbackRate = value }
    }
    Audio { id: audio; source: "music.mp3" }
}
```

这里在工具栏添加了控制播放的几个按钮，然后对播放进度、音量大小、循环次数和播放速度进行了设置。运行效果如图 12－1 所示。

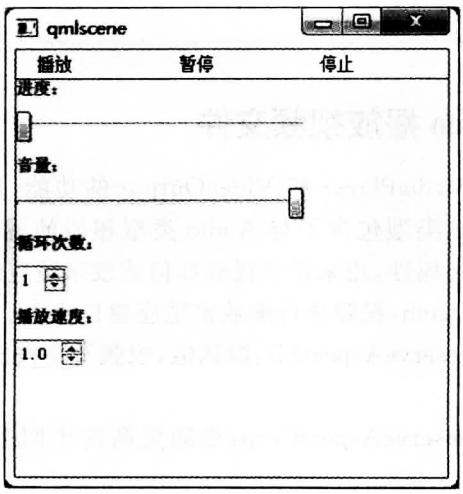

图 12－1　Audio 示例运行效果

12.2.2 播放未压缩音频

SoundEffect 类型允许使用低延迟的方式播放未压缩的音频文件(通常的 WAV 文件),适合于用作响应用户操作的"反馈"声音,例如虚拟键盘的声音、弹出对话框或游戏的声音。如果不需要低延迟,那么可以考虑使用 MediaPlayer 或者 Audio 类型,因为这些类型支持更多的媒体格式并且消耗更少的资源。

一般 SoundEffect 类型的声音都是会被多次使用的。它允许提前进行解析并且准备完毕,在需要的时候只需要触发即可。这在 QML 中实现起来很容易,可以声明一个 SoundEffect 实例,然后在任意位置引用它。

下面来看一个例子(项目源码路径:src\12\12-3\mysoundeffect):

```
import QtQuick 2.2
import QtMultimedia 5.0
Text {
    text: "Click Me!";
    font.pointSize: 24;
    width: 150; height: 50;
    SoundEffect {
        id: playSound
        source: "soundeffect.wav"
    }
    MouseArea {
        id: playArea; anchors.fill: parent
        onPressed: { playSound.play() }
    }
}
```

因为 SoundEffect 需要稍多资源实现低延迟播放,一些平台可能会限制同时播放 SoundEffect 的数量。

12.3 播放视频

12.3.1 使用 Video 播放视频文件

Video 类型结合了 MediaPlayer 和 VideoOutput 的功能,这是为了方便直接播放视频而设置的一个类。该类型包含了与 Audio 类型相似的属性和函数。另外,Video 类型还包含一个 fillMode 属性,用来定义视频如何缩放来适应窗口,可以取以下值:

- VideoOutput. Stretch:视频进行缩放来适应窗口大小;
- VideoOutput. PreserveAspectFit:默认值,视频宽高按比例进行缩放,不会进行裁剪;
- VideoOutput. PreserveAspectCrop:视频宽高按比例进行缩放,在必要时会进行裁剪。

下面来看一个例子(项目源码路径:src\12\12-4\myvideo):

```
import QtQuick 2.2
import QtMultimedia 5.0
Video {
    id: video; width : 800; height : 600
    source: "video.wmv"
    MouseArea {
        anchors.fill: parent
        onClicked: video.play()
    }
    focus: true
    Keys.onSpacePressed: video.playbackState
                         == MediaPlayer.PlayingState
                        ? video.pause() : video.play()
    Keys.onLeftPressed: video.seek(video.position - 5000)
    Keys.onRightPressed: video.seek(video.position + 5000)
}
```

这里使用空格来控制视频的播放和暂停，使用左右方向键来实现播放的快进和快退。运行效果如图 12 - 2 所示。

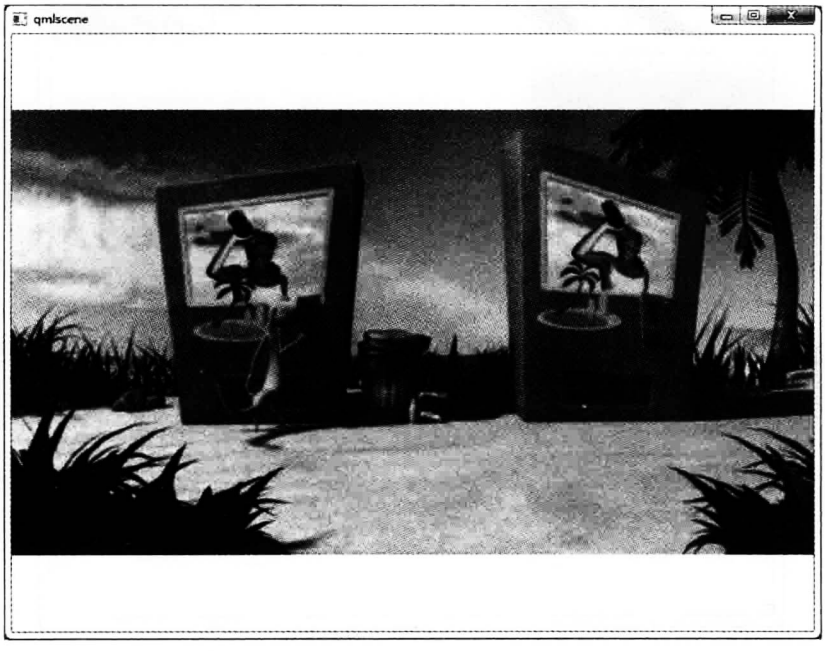

图 12 - 2　Video 示例运行效果

12.3.2　对视频使用图形效果

第 8 章讲到的图形效果可以应用到 Video 上，从而使视频产生某种特殊效果。下面来看一个例子(项目源码路径：src\12\12-5\myvideo)：

```
import QtQuick 2.2
import QtMultimedia 5.0
```

```
import QtGraphicalEffects 1.0
Item {
    width: 450; height: 350
    Video {
        id: video; y:50; width: 400; height: 300
        source: "video.wmv"
        MouseArea {
            anchors.fill: parent
            onClicked: video.play()
        }
    }
    FastBlur {
        width: 150; height: 100;
        source: video; radius: 32
    }
}
```

这里使用了快速模糊 FastBlur 类型，将 video 作为源对视频进行了渲染，运行效果如图 12-3 所示。

图 12-3　在视频上使用快速模糊效果

前面的例子中，原视频和模糊后效果是同时显示的。可以在 Video 中设置 visible 为 false，就只显示使用图形效果后的视频了。例如下面的例子中使用了阈值遮罩，可以设置视频的显示区域。（项目源码路径：src\12\12-6\myvideo。）

```
import QtQuick 2.2
import QtMultimedia 5.0
import QtGraphicalEffects 1.0
Item {
```

```
    width: 300; height: 300
    Video {
        id: video;  width: 300; height: 300
        source: "video.wmv"
        fillMode: VideoOutput.Stretch
        visible: false
    }
    MouseArea {
        anchors.fill: parent
        onClicked: video.play()
    }
    Image {
        id: mask; source: "mask.png"
        sourceSize: Qt.size(parent.width, parent.height)
        smooth: true; visible: false
    }
    ThresholdMask {
        anchors.fill: parent
        source: video; maskSource: mask
        threshold: 0.45; spread: 0.2
    }
}
```

运行效果如图 12-4 所示。

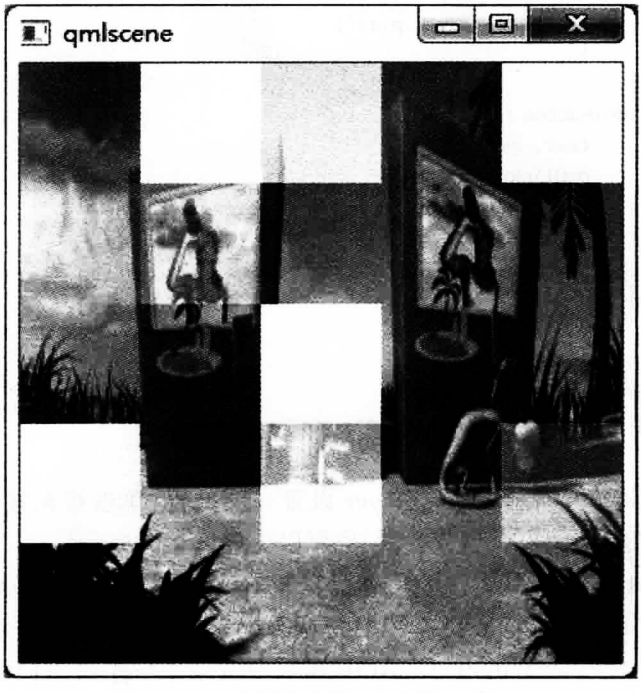

图 12-4　在视频上使用阈值遮罩效果

12.4 媒体播放器(MediaPlayer)

12.4.1 播放音视频

MediaPlayer 类型包含了与 Audio 类型几乎相同的属性、函数和信号,它自身可以直接播放音频内容(和 Audio 一样),也可以结合 VideoOutput 类型来渲染视频。

下面来看一个例子(项目源码路径:src\12\12-7\mymediaplayer):

```
import QtQuick 2.2
import QtMultimedia 5.0
import QtQuick.Controls 1.1
import QtQuick.Layouts 1.1
ApplicationWindow {
    id: window; width: 300; height: 300
    toolBar: ToolBar {
        RowLayout {
            anchors.fill: parent
            ToolButton {
                text: qsTr("音频")
                onClicked: {
                    mediaplayer.source = "music.mp3"
                    mediaplayer.play()
                }
            }
            ToolButton {
                text: qsTr("视频")
                onClicked: {
                    mediaplayer.source = "video.wmv"
                    mediaplayer.play()
                }
            }
        }
    }
    MediaPlayer { id: mediaplayer }
    VideoOutput { anchors.fill: parent; source: mediaplayer }
}
```

这里单击音频按钮时给 MediaPlayer 设置音频文件,单击视频按钮时为其设置视频文件。可见,该类型既可以播放音频,也可以播放视频。

VideoOutput 类型使用平台后端进行视频的渲染,它需要 MediaPlayer 为其提供视频帧。该类型工作的后端支持 QVideoRendererControl 或 QVideoWindowControl。如果后端只支持 QVideoWindowControl,视频会渲染到一个覆盖的窗口上,该窗口会层叠在 QtQuick 窗口之上,并且一些特性也不会被支持,例如:旋转等转换操作、在 VideoOutput 项目之上放置其他 QtQuick 项目。因为大多数后端都支持 QVideoRendererControl,所以一般不会出现这些限制。

12.4.2　使用 Windows 平台附加功能

下面设计一个具有 Windows 7 界面效果的音乐播放器程序，这需要使用到 Qt Windows Extras 模块。Windows 附加功能模块包含了一些 Windows 系统特有的功能，例如可以使用 Windows 7 中的 Aero Peek、跳转列表、在任务栏按钮上显示进度指示器、缩略图工具栏等。使用该模块需要在 QML 文件中使用如下导入语句：

```
import QtWinExtras 1.0
```

对于本节内容，可以在 Qt 帮助中通过索引 Qt Windows Extras Overview 关键字查看。

下面来看一个例子(项目源码路径：src\12\12-8\mymediaplayer)。新建 Qt Quick Application 应用程序，项目名称为 myquickplayer，组件集选择 Qt Quick 2.2。完成后在.pro 文件中添加如下一行代码：

```
RC_ICONS = images/quickplayer.ico
```

这样便指定了 quickplayer.ico 为程序的图标。然后在项目文件列表中右击 qml.qrc，选择 Open in Editor，然后将源码目录中的 images 目录中的图片添加到资源文件中，然后保存该文件。下面打开 main.cpp 文件，修改如下：

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QWindow>
int main(int argc, char * argv[])
{
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:///main.qml")));
    QObject * root = engine.rootObjects().value(0);
    if (QWindow * window = qobject_cast<QWindow * >(root))
        window->show();
    return app.exec();
}
```

这里获取了 QML 引擎的根对象，也就是 main.qml 的 Window，然后转换为 QWindow 类对象并进行了显示。其目的是显示 QML 指定的窗口，而不是一般的 Widget 程序窗口。

下面打开 main.qml，首先将其内容修改如下：

```
import QtQuick 2.2
import QtQuick.Window 2.1
import QtQuick.Layouts 1.1
import QtQuick.Controls 1.2
import QtQuick.Dialogs 1.1
import QtMultimedia 5.0
import QtWinExtras 1.0 as Win
Window {
    id: window
```

```
title: qsTr("My Quick Player")
width: 300; height: 60
minimumWidth: row.implicitWidth + 18
minimumHeight: column.implicitHeight + 18
MediaPlayer {
    id: mediaPlayer
    autoPlay: true
    readonly property string title:
     !! metaData.author && !! metaData.title
     ? qsTr("%1 - %2").arg(metaData.author).arg(metaData.title)
     : metaData.author || metaData.title || source
}
ColumnLayout {
    id: column
    anchors.margins: 9; anchors.fill: parent
    Label {
        id: infoLabel;
        elide: Qt.ElideLeft
        verticalAlignment: Qt.AlignVCenter
        text: mediaPlayer.errorString || mediaPlayer.title
        Layout.minimumHeight: infoLabel.implicitHeight
        Layout.fillWidth: true; Layout.fillHeight: true
    }
    RowLayout {
        id: row
        Button {
            id: openButton; text: qsTr("...")
            Layout.preferredWidth: openButton.implicitHeight
            onClicked: fileDialog.open()
            FileDialog {
                id: fileDialog
                title: qsTr("Open file")
                nameFilters: [qsTr("MP3 files (*.mp3)"),
                    qsTr("All files (*.*)")]
                onAccepted: mediaPlayer.source
                                = fileDialog.fileUrl
            }
        }
        Button {
            id: playButton
            enabled: mediaPlayer.hasAudio
            Layout.preferredWidth: playButton.implicitHeight
            iconSource: mediaPlayer.playbackState
                        === MediaPlayer.PlayingState
                         ? "qrc:/images/pause-16.png"
                         : "qrc:/images/play-16.png"
            onClicked: mediaPlayer.playbackState
                        === MediaPlayer.PlayingState
                    ? mediaPlayer.pause() : mediaPlayer.play()
        }
        Slider {
```

```
                id: positionSlider
                Layout.fillWidth: true
                maximumValue: mediaPlayer.duration
                property bool sync: false
                onValueChanged: {
                    if (! sync)
                        mediaPlayer.seek(value)
                }
                Connections {
                    target: mediaPlayer
                    onPositionChanged: {
                        positionSlider.sync = true
                        positionSlider.value = mediaPlayer.position
                        positionSlider.sync = false
                    }
                }
            }
            Label {
                id: positionLabel
                readonly property int minutes:
                 Math.floor(mediaPlayer.position / 60000)
                readonly property int seconds:
                 Math.round((mediaPlayer.position % 60000) / 1000)
                text: Qt.formatTime(new Date(0, 0, 0, 0, minutes,
                                             seconds), qsTr("mm:ss"))
            }
        }
    }
}
```

这里使用一些控件和 MediaPlayer 类型创建了一个简单的播放器界面，如图 12-5 所示。首先创建了一个 MediaPlayer 实例，用来播放音频，其中自定义了一个 title 属性，用来显示一些信息。下面使用 ColumnLayout 设置一个列布局，包含了一个用于显示信息的 Label 和一个 RowLayout。在 RowLayout 中，包含了两个按钮 Button，一个滑块 Slider 和一个用于显示播放时间的 Label。第一个 Button 用于弹出一个文件对话框，在文件对话框中设置了只显示 mp3 类型的文件；第二个 Button 用于控制音频的播放，根据播放器状态显示播放或暂停图标；Slider 用于显示和控制播放进度，因为其中使用 Connections 类型设置了关联，所以拖动滑块可以定位音频播放位置。最后的 Label 显示了音频的播放时间，其中 Math.floor()用于获取指定值的整数部分，Math.round()可以对指定值进行四舍五入来获取整数部分。

现在运行程序，已经可以实现播放器功能了，下面来为其实现玻璃界面效果。

1. DwmFeatures

使用 DwmFeatures 类型可以扩展玻璃边框到客户端区域，还可以控制 Aero Peek 和 Flip3D 的行为。该类型的 topGlassMargin 等属性可以设置边界玻璃边框的边距，就是从窗口的边界开始向内有多大区域使用玻璃边框效果。

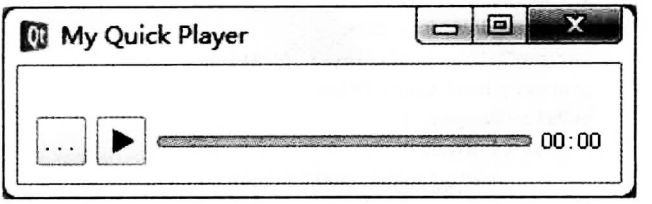

图 12-5　播放器界面设计效果图

下面在前面例子的 Window 根对象中添加如下代码：

```
Win.DwmFeatures {
    id: dwm
    topGlassMargin: -1; leftGlassMargin: -1
    rightGlassMargin: -1; bottomGlassMargin: -1
}
```

这里将 4 个边距都设置为了－1，这样整个窗口都会使用玻璃效果。运行程序效果如图 12-6 所示。

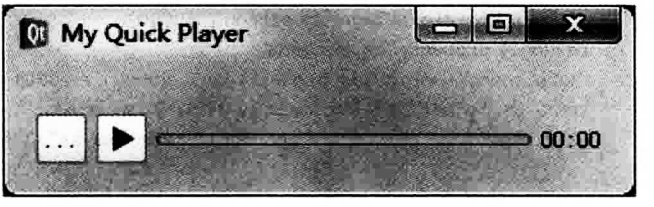

图 12-6　设置玻璃效果后的界面

2. TaskbarButton

TaskbarButton 类型可以在任务栏按钮上覆盖一个图标以及显示一个进度指示器。图标可以显示应用程序的状态变化，指示器可以显示一个耗时的任务的进度。其中图标可以使用 overlay.iconSource 属性指定；进度指示器需要使用 progress 分组属性来设置，包括当前值 progress.value、最小值 progress.minimum、最大值 progress.maximum、是否可见 progress.visible、是否暂停 progress.paused、是否停止 progress.stopped。

下面在前面例子的 Window 根对象中添加如下代码：

```
Win.TaskbarButton {
    id: taskbar
    progress.value: mediaPlayer.position
    progress.maximum: mediaPlayer.duration
    progress.visible: mediaPlayer.hasAudio
    progress.paused: mediaPlayer.playbackState
                     === MediaPlayer.PausedState
    overlay.iconSource: mediaPlayer.playbackState
                        === MediaPlayer.PlayingState
                        ? "qrc:/images/play-32.png" :
```

```
                mediaPlayer.playbackState
                === MediaPlayer.PausedState ?
                    "qrc:/images/pause - 32.png"
                  : "qrc:/images/stop - 32.png"
}
```

现在运行程序，然后播放一个音频文件，任务栏的按钮效果如图 12 - 7 所示。

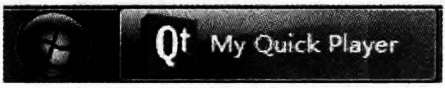

图 12 - 7　设置任务栏按钮效果

3. ThumbnailToolBar 和 ThumbnailToolButton

ThumbnailToolBar 类型允许应用程序在窗口缩略图上嵌入一个工具栏，当鼠标悬停在任务栏图标上时，它提供了一种快速访问窗口命令的方式，不需要用户恢复或者激活窗口。

ThumbnailToolButton 提供了工具栏上的工具按钮，可以使用 iconSource 属性来设置图标，使用 tooltip 属性设置提示，使用 visible 属性设置按钮是否可见，单击后会发射 clicked()信号。

下面在前面例子的 Window 根对象中添加如下代码：

```
Win.ThumbnailToolBar {
    id: thumbbar
    Win.ThumbnailToolButton {
        tooltip: qsTr("Rewind")
        iconSource: "qrc:/images/backward - 32.png"
        enabled: mediaPlayer.position > 0
        onClicked: mediaPlayer.seek(mediaPlayer.position
                                    - mediaPlayer.duration / 10)
    }
    Win.ThumbnailToolButton {
        tooltip: mediaPlayer.playbackState
                 === MediaPlayer.PlayingState
                 ? qsTr("Pause") : qsTr("Play")
        iconSource: mediaPlayer.playbackState
                    === MediaPlayer.PlayingState
            ? "qrc:/images/pause - 32.png" : "qrc:/images/play - 32.png"
        enabled: mediaPlayer.hasAudio
        onClicked: mediaPlayer.playbackState
                   === MediaPlayer.PlayingState
                   ? mediaPlayer.pause() : mediaPlayer.play()
    }
    Win.ThumbnailToolButton {
        tooltip: qsTr("Fast forward")
        iconSource: "qrc:/images/forward - 32.png"
        enabled: mediaPlayer.position < mediaPlayer.duration
        onClicked: mediaPlayer.seek(mediaPlayer.position
```

```
                                    + mediaPlayer.duration / 10)
    }
}
```

这里添加了 3 个工具按钮，运行效果如图 12-8 所示。

图 12-8　设置缩略图工具栏效果

12.5　小　结

本章简单介绍了 Qt 5 中全新 Qt Multimedia 模块的一些更新和音视频的简单处理操作，最后还讲解了 Windows 平台附加功能的一些内容。在 QML 中可以很容易地将其他控件或图形动画效果与音视频播放进行整合，从而实现一些特殊效果。