第17章 综合实例之文件查看器

本章提供一个综合应用各种 Qt Quick 主题的实例：文件查看器。它支持查看文本文件、图片，播放音乐、视频。浏览文本时可以设置文字颜色、背景色以及字号大小。而且，它是以 ApplicationWindow 为基础构建的，也算对一直以来备受鄙人冷落的 ApplicationWindow 的一点补偿哈。

本实例将会用到下列特性：

- ApplicationWindow
- MenuBar
- ToolBar、ToolButton
- Action
- StatusBar
- MediaPlayer
- Image
- XMLHttpRequest
- ColorDialog
- FileDialog
- TextArea
- 动态创建 QML 组件
- 多界面切换

笔者只在 Windows 平台测试过文件查看器，Android 平台还请有兴趣的朋友自己实测。

先看点图吧。

17.1 文件查看器的运行效果

图 17-1 是初始运行效果图。

图 17-2 是菜单效果图。

图 17-1 文件查看器初始运行效果

图 17-2 文件查看器的文件菜单

我在代码中通过共享 Action，让菜单项和工具栏按钮保持一致的逻辑。这也是使用 ApplicationWindow 要遵循的一个原则。

图 17-3 是浏览文本文件的效果图。

图 17-4 是看图片的效果图。

图 17-3 使用文件查看器浏览文本文件

图 17-4 使用文件查看器浏览图片

图 17-5 是看视频的效果图。

图 17-6 是听音乐的效果图。

图 17-5　使用文件查看器播放视频

图 17-6　使用文件查看器听音乐

好啦，现在应该对文件查看器有了感性认识了。接下来我们就对这个实例用到的新主题做详细讲解。

17.2　再论 ApplicationWindow

我们在 4.2.2 节第一次约会 ApplicationWindow，它很羞涩，低垂臻首，一句多余的话也没有。这次我们要敞开了聊聊。

ApplicationWindow，类似于 Qt C++中的 QMainWindow，请对照着理解。图 17-7 是我结合本节实例画的一张示意图。

如图 17-7 所示，最上面是标题栏，有应用图标和应用的名字。

紧接着就是菜单栏了，我标注出来了，它对应 ApplicationWindow 的 menuBar 属性，menuBar 属性的类型为 MenuBar，咱后面讲。

图 17-7 ApplicationWindow 示意图

菜单栏下面是工具栏，对应 ApplicationWindow 的 toolBar 属性，类型为 Item，不过一般我们使用 ToolBar 对象为其赋值。

工具栏一般是菜单栏的子集，在设计时将常用的功能放在工具栏上，操作方便，因此实现上也经常采用共享 Action 对象的方式来保证菜单与工具栏的一致性。

在图 17-7 所示界面的最下方，我标注出来的部分，是状态栏。对应 ApplicationWindow 的 statusBar 属性，类型为 Item。你可以将任意的 Item 赋值给它，可以随心所欲地构建妖娆多姿的状态栏。也可以像下面代码所示的那样寒碜：

```
ApplicationWindow{
    statusBar: Text {
        text: "status bar";
        color: "blue";
    }
}
```

界面的中间部分，我标注为 ContentItem 的，对应于 QMainWindow 的 centralWidget，也是我们常说的工作区，ApplicationWindow 有一个 contentItem 属性，可以设置这个区域的最大、最小、建议尺寸。那到底中间这个区域对应的控件是谁？怎么设置？哇哈哈，就是那个，你在 ApplicationWindow 对象内定义了却又不给人指定归属的 Item 对象。例如：

```
ApplicationWindow{
    ...
    Rectangle{
        id: centralView;
        anchors.fill: parent;
        color: "red";
    }
}
```

好啦，现在我们来看看使用 ApplicationWindow 的典型代码结构：

```
ApplicationWindow{
    ...
    menuBar: MenuBar {
        ...
    }

    toolBar: ToolBar {
```

```
        ...
    }

    Rectangle{
        id: centralView;
        anchors.fill: parent;
        color: "red";
    }

    statusBar: Text {
        text: "status bar";
        color: "blue";
    }
}
```

ApplicatonWindow 会自动计算 menuBar、toolBar、statusBar 三个属性指定的组件大小，合理布局，剩下没人要的区域，就把同样没人要的那个 Item 放进去，作为 ContentItem。

接下来让我们看看组成 ApplicationWindow 的两大要素：MenuBar 和 ToolBar。

17.3 MenuBar

MenuBar 对象就负责菜单条那块区域了，它干两件事：

- 维护一个 Menu 列表（menus 属性）。
- 绘制菜单栏背景色（style 属性）。

style 属性用来绘制菜单条的样子，比如背景了、菜单的效果了。暂且不讲。

menus 属性类型为 list<Menu>，是个列表属性，也是默认属性。比如可以这样定义一个菜单条：

```
MenuBar {
    Menu {
        title: "File"
        ...
    }

    Menu {
        title: "Edit"
        ...
    }
}
```

现在该看 MenuBar 管理的那些 Menu 对象到底是什么东西了。

17.3.1 Menu

图 17-2 是点击“文件”菜单后的效果图。Menu 就代表“文件”以及点击它弹出来的那个菜单列表。

Menu 的列表属性 items 是默认属性，指向 Menu 的子菜单项，它的类型是 list<Object>。一般我们使用 MenuItem 来作为叶子菜单项（不可再展开）。

title 属性是 string 类型，你看到的“文件”、“设置”等字样就是它指定的。

有这两个属性，Menu 就可以开始干活了。来看文件查看器的“帮助”菜单对应的代码：

```
Menu {
    title: "帮助";
    MenuItem{
        text: "关于";
        onTriggered: root.showAbout();
    }
    MenuItem{
        text: "访问作者博客";
        onTriggered: Qt.openUrlExternally("http://blog.csdn.net/foruok");
    }
}
```

又出新东西了：MenuItem。

17.3.2 MenuItem

Menu 的孩子啊，最受待见的就是 MenuItem 了。

MenuItem 代表一个具体的菜单项，点一下就干一件事情的那个角色。它是叶子节点，不可展开了。你可以实现 onTriggered 信号处理器响应用户对它的选择。

MenuItem 的 text 属性指定菜单文字，iconSource 属性指定菜单图标。

现在可以理解上一节的代码片段了。

MenuItem 还有 checkable、visible、type 等属性，请参考 Qt 帮助了解详情吧。

Action 属性很强大哦，MenuItem 的 text、iconSource、trigger 信号等实现的效果，都可以通过 Action 来实现，这两种方式是等同的。

17.3.3 Action

Action 类有 text、iconSource 等属性，还有 toggled、triggered 两个信号。MenuItem 里有对应的属性，不必多说了。

使用 Action 的一大好处是，你可以给它指定一个 id（比如叫 open），然后在为 ToolBar 添加“打开”按钮时，指定 ToolButton 对象的 action 属性为之前定义的 id 为 open 的那个 action，这样工具栏的按钮就和菜单关联起来了，步调一致哦。

好啦，总结一下，实现菜单栏的典型代码结构是这样子的：

```
MenuBar {
    Menu{
        title:"文件";
        MenuItem{
            text:"打开";
            iconSource: "res/ic_open.png";
            onTriggered:{
                ...
            }

        }
        MenuItem{
            action: Action {
                text:"保存";
                iconSource: "res/ic_save.png";
                onTriggered:{
```

```
                ...
            }
        }
    }
}
```

如你所见，我使用了两种构造 MenuItem 的方式，一种用到了 Action，一种没有。

17.4 ToolBar

ToolBar 就是工具栏对应的类，它只有一个属性——contentItem，类型为 Item。一般我们可以将一个 Row 或者 RowLayout 对象赋值给 contentItem，而 Row 或 RowLayout 则管理一组 ToolButton 来作为工具栏上的按钮。

ToolButton

ToolButton 是 Button 的派生类，专为 ToolBar 而生，一般情况下定义 ToolButton 对象时只需要指定其 iconSource 属性即可。例如：

```
ToolButton {
    iconSource: "res/ic_open.png";
}
```

还有一种方式是将一个已定义好的 Action 对象关联到 ToolButton 对象上。例如：

```
ToolButton{
    action: openAction;
}
```

我们的文件查看器大量使用了这种方式，它的好处不言而喻，ToolButton 会使用 Action 定义的 iconSource 或 iconName 作为其图标，而且 Action 定义的信号处理器也会在合适的时候被调用。

好啦，看看文件查看器的工具栏对应的代码：

```
toolBar: ToolBar{
    RowLayout {
        ToolButton{
            action: textAction;
        }
        ToolButton{
            action: imageAction;
        }
        ToolButton{
            action: videoAction;
        }
        ToolButton{
            action: audioAction;
        }
        ToolButton{
            action: textColorAction;
        }
        ToolButton {
            action: backgroundColorAction;
        }
        ToolButton {
            action: fontSizeAddAction;
```

```
            }
            ToolButton {
                action: fontSizeMinusAction;
            }
        }
    }
```

这里需要说明一点，当你在 main()函数中使用 QGuiApplication 时，Qt Quick 会使用自己实现的 MenuBar、ToolBar，效果和宿主环境的不太一样（使用 QApplication 时会优先使用宿主环境的实现）。看图 17-8，对比下图 17-2，看看有什么不同吧。

找茬了找茬了……菜单项的图标没显示出来；工具栏按钮比较大，与图标尺寸一样。还有呢……

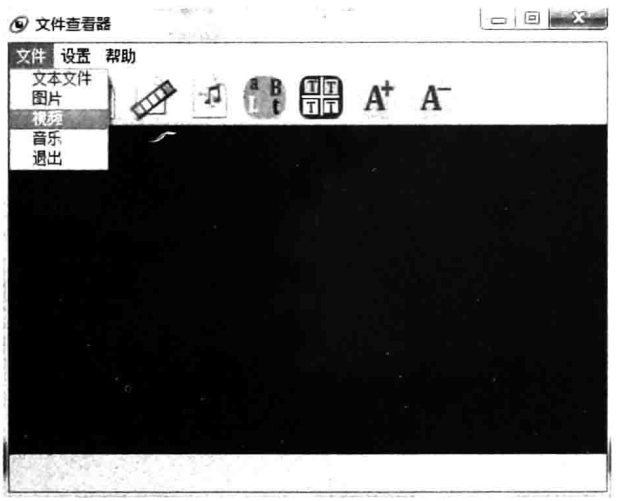

图 17-8　使用 Qt Quick 自带的 MenuBar 和 ToolBar

17.5　使用 XMLHttpRequest 加载本地文件

在 15.2 节我们介绍了 XMLHttpRequest 对象，主要用它来完成网络请求。而实际上，它也可以加载本地文件。QML 里没有提供文件操作的类库，这也算是加载本地文件的一种不得已的办法。文件查看器就是这么干的。

不过使用 XMLHttpRequest 加载本地文件有个显而易见的缺点：因为不是实际的网络请求，它无法从 HTTP 头部中获取正确的文件编码格式，统一使用 UTF-8 来处理文本，这样的话，那些非 UTF-8 格式的文本文件，如果包含中文，则会显示为乱码哈。

17.6　使用标准对话框

Qt Quick 提供了一系列的标准对话框，如 FileDialog、ColorDialog、MessageDialog、FontDialog 等，FileDialog 在前面讲过了，这里介绍一下我们用到的 ColorDialog 和 MessageDialog。

需要说明的是，Qt Quick 会首先使用平台相关的标准对话框，如果不可用，就使用自己

实现的版本，比如在 Windows 下，两者的样子就大大不同。图 17-9 是 Windows 7 上的颜色选择对话框效果图。

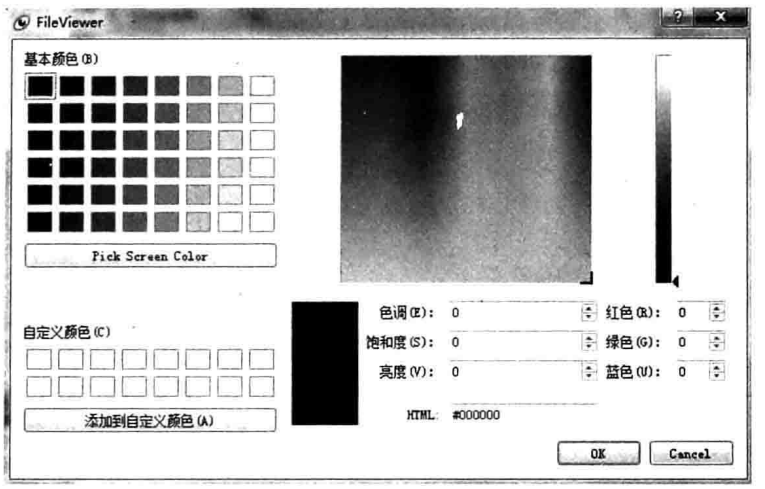

图 17-9　文件查看器调用 Windows 7 颜色选择对话框

图 17-10 是 Qt Quick 实现的效果图。

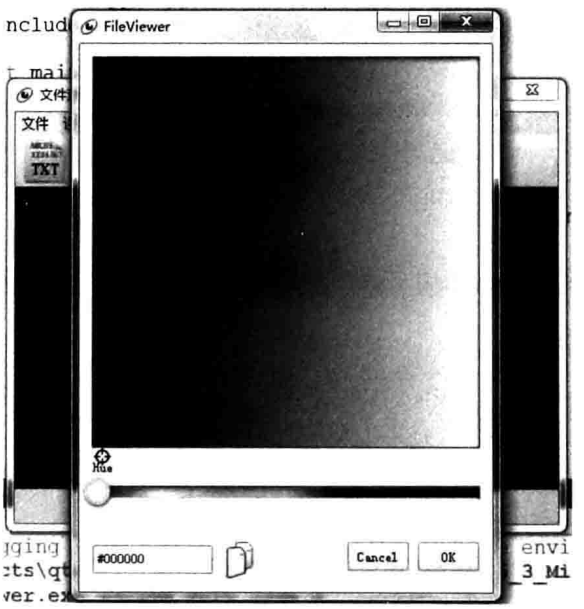

图 17-10　文件查看器使用 Qt Quick 本身的颜色选择对话框

17.6.1　ColorDialog

ColorDialog 用来选择一个颜色。

color 属性保存用户选择的颜色，在 onAccepted 信号处理器中读取它就对了。还有个 currentColor 属性，保存用户正在选的、按确定之前的颜色，可以用它跟踪用户的选择。

visible 属性，默认值为 false。设置为 true 就可以显示颜色对话框，等同于调用 open()方法。

title 属性设置颜色对话框的标题，你看图 17-10，都是因为我在代码里没有设置，它使用了英文字样“FileViewer”。

modality 属性设置模态策略。close()方法用于关闭对话框。

我们的实例中是这么用颜色对话框的：

```
function selectColor(func){
   if(colorDlg == null){
      colorDlg = Qt.createQmlObject(
                'import QtQuick 2.2;import
                 QtQuick.Dialogs 1.1;ColorDialog{}',
                 root, "colorDlg");
      colorDlg.accepted.connect(func);
      colorDlg.accepted.connect(onColorDlgClosed);
      colorDlg.rejected.connect(onColorDlgClosed);
      colorDlg.visible = true;
   }
}

function onColorDlgClosed(){
   colorDlg.destroy();
   colorDlg = null;
}

function onTextColorSelected(){
   root.textColor = colorDlg.color;
}

function onTextBackgroundColorSelected(){
   root.textBackgroundColor = colorDlg.color;
}
```

我使用 Qt.createQmlObject()动态地创建颜色对话框，连接 accepted 信号到 onTextColorSelected 或 onTextBackgroundColorSelected 方法上，具体是哪个，由用户点击的菜单项或工具按钮决定。

我还将 ColorDialog 的 accepted 与 rejected 两个信号连接到 onColorDlgClosed 方法上，以便销毁动态创建的对象。

17.6.2 MessageDialog

MessageDialog 用来显示一个弹出消息框，我使用它显示“关于”信息。

title 属性设置消息框的标题。

text 设置要显示的概要文字。而 detailedText 设置详细文字，一般是隐藏的，对话框上有个“显示详情”按钮，点击才会把 detailedText 指定的文本显示出来。

icon 设置提示图标，可以是 StandardIcon.Question、StandardIcon. Information、StandardIcon. Warning、StandardIcon.Critical、StandardIcon. NoIcon 这 5 个枚举值中的一个。

standardButtons 属性设置消息框显示的按钮，比如 StandardButton.Ok、StandardButton. Cancel、StandardButton.Yes 等，完整列表请参考 Qt 帮助。每个按钮都对应一个 Role，会触发对应的信号，比如 Ok 会触发 accepted 信号，而 Cancel 会触发 rejected 信号。

文件查看器使用 MessageDialog 的代码如下：

```
function showAbout(){
   if(aboutDlg == null){
```

```
            aboutDlg = Qt.createQmlObject(
                    'import QtQuick 2.2;import QtQuick.Dialogs 1.1;MessageDialog{icon:
StandardIcon.Information;title: "关于";\ntext: "仅仅是个示例撒";
\nstandardButtons:StandardButton.Ok;}'
                    , root, "aboutDlg");
            aboutDlg.accepted.connect(onAboutDlgClosed);
            aboutDlg.rejected.connect(onAboutDlgClosed);
            aboutDlg.visible = true;
        }

    }

    function onAboutDlgClosed(){
        aboutDlg.destroy();
        aboutDlg = null;
    }
```

与使用颜色对话框类似，不再赘述。

17.7　源码分析

分解动作讲解完毕，该看代码了。

17.7.1　QML 代码

所有功能都在一个 QML 文档中完成，先看它吧，450 行代码，都在这里了。

```
import QtQuick 2.2
import QtQuick.Window 2.1
import QtQuick.Controls 1.2
import QtQuick.Controls.Styles 1.2
import QtQuick.Layouts 1.1
import QtQuick.Dialogs 1.1
import QtMultimedia 5.0

ApplicationWindow {
    visible: true
    width: 480
    height: 360;
    color: "black";
    title: "文件查看器";
    id: root;
    property var aboutDlg: null;
    property var colorDlg: null;
    property color textColor: "green";
    property color textBackgroundColor: "black";

    menuBar: MenuBar{
        Menu {
            title: "文件";
            MenuItem{
                iconSource: "res/txtFile.png";
                action: Action{
                    id: textAction;
                    iconSource: "res/txtFile.png";
                    text: "文本文件";
                    onTriggered: {
                        fileDialog.selectedNameFilter = fileDialog.nameFilters[0];
```

```
                fileDialog.open();
            }
            tooltip: "打开 txt 等文本文件";
        }
    }
    MenuItem{
        action: Action {
            id: imageAction;
            text: "图片";
            iconSource: "res/imageFile.png";
            onTriggered: {
                fileDialog.selectedNameFilter = fileDialog.nameFilters[1];
                fileDialog.open();
            }
            tooltip: "打开 jpg 等格式的图片";
        }
    }
    MenuItem{
        action: Action {
            id: videoAction;
            iconSource: "res/videoFile.png";
            text: "视频";
            onTriggered: {
                fileDialog.selectedNameFilter = fileDialog.nameFilters[2];
                fileDialog.open();
            }
            tooltip: "打开 TS、MKV、MP4 等格式的文件";
        }
    }
    MenuItem{
        action: Action {
            id: audioAction;
            iconSource: "res/audioFile.png";
            text: "音乐";
            onTriggered: {
                fileDialog.selectedNameFilter = fileDialog.nameFilters[3];
                fileDialog.open();
            }
            tooltip: "打开 mp3、wma 等格式的文件";
        }
    }
    MenuItem{
        text: "退出";
        onTriggered: Qt.quit();
    }
}
Menu {
    title: "设置";
    MenuItem {
        action: Action {
            id: textColorAction;
            iconSource: "res/ic_textcolor.png";
            text: "文字颜色";
            onTriggered:
              root.selectColor(root.onTextColorSelected);
        }
```

```
        }
        MenuItem {
            action: Action{
                id: backgroundColorAction;
                iconSource: "res/ic_bkgndcolor.png";
                text: "文字背景色";
                onTriggered: root.selectColor(
                      root.onTextBackgroundColorSelected);
            }
        }
        MenuItem {
            action: Action{
                id: fontSizeAddAction;
                iconSource: "res/ic_fontsize2.png";
                text: "增大字体";
                onTriggered: textView.font.pointSize += 1;
            }
        }
        MenuItem {
            action: Action{
                id: fontSizeMinusAction;
                iconSource: "res/ic_fontsize1.png";
                text: "减小字体";
                onTriggered: textView.font.pointSize -= 1;
            }
        }
    }
    Menu {
        title: "帮助";
        MenuItem{
            text: "关于";
            onTriggered: root.showAbout();
        }
        MenuItem{
            text: "访问作者博客";
            onTriggered: Qt.openUrlExternally("http://blog.csdn.net/foruok");
        }
    }
}

toolBar: ToolBar{
    RowLayout {
        ToolButton{
            action: textAction;
        }
        ToolButton{
            action: imageAction;
        }
        ToolButton{
            action: videoAction;
        }
        ToolButton{
            action: audioAction;
        }
        ToolButton{
            action: textColorAction;
        }
        ToolButton {
            action: backgroundColorAction;
        }
        ToolButton {
```

```
            action: fontSizeAddAction;
        }
        ToolButton {
            action: fontSizeMinusAction;
        }
    }
}

statusBar: Rectangle {
    color: "lightgray";
    implicitHeight: 30;
    width: parent.width;
    property alias text: status.text;
    Text {
        id: status;
        anchors.fill: parent;
        anchors.margins: 4;
        font.pointSize: 12;
    }
}

Item {
    id: centralView;
    anchors.fill: parent;
    visible: true;
    property var current: null;
    BusyIndicator {
        id: busy;
        anchors.centerIn: parent;
        running: false;
        z: 3;
    }
    Image {
        id: imageViewer;
        anchors.fill: parent;
        visible: false;
        asynchronous: true;
        fillMode: Image.PreserveAspectFit;
        onStatusChanged: {
            if (status === Image.Loading) {
                centralView.busy.running = true;
            }
            else if(status === Image.Ready){
                centralView.busy.running = false;
            }
            else if(status === Image.Error){
                centralView.busy.running = false;
                centralView.statusBar.text = "图片无法显示";
            }
        }
    }

    TextArea {
        id: textView;
        anchors.fill: parent;
        readOnly: true;
        visible: false;
        wrapMode: TextEdit.WordWrap;
        font.pointSize: 12;
        style: TextAreaStyle{
            backgroundColor: root.textBackgroundColor;
            textColor: root.textColor;
            selectionColor: "steelblue";
            selectedTextColor: "#a00000";
        }
```

```
    property var xmlhttp: null;
    function onReadyStateChanged(){
        if(xmlhttp.readyState == 4){
            text = xmlhttp.responseText;
            xmlhttp.abort();
        }
    }

    function loadText(fileUrl){
        if(xmlhttp == null){
            xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = onReadyStateChanged;
        }
        if(xmlhttp.readyState == 0){
            xmlhttp.open("GET", fileUrl);
            xmlhttp.send(null);
        }
    }
}

VideoOutput {
    id: videoOutput;
    anchors.fill: parent;
    visible: false;
    source: player;
    onVisibleChanged: {
        playerState.visible = visible;
        if(visible == false){
            player.stop();
        }
    }
    MouseArea {
        anchors.fill: parent;
        onClicked: {
            switch(player.playbackState){
            case MediaPlayer.PausedState:
            case MediaPlayer.StoppedState:
                player.play();
                break;
            case MediaPlayer.PlayingState:
                player.pause();
                break;
            }
        }
    }
}

Rectangle {
    id: playerState;
    color: "gray";
    radius: 16;
    opacity: 0.8;
    visible: false;
    z: 2;
    implicitHeight: 80;
    implicitWidth: 200;
    anchors.horizontalCenter: parent.horizontalCenter;
    anchors.bottom: parent.bottom;
    anchors.bottomMargin: 16;
    Column {
        anchors.fill: parent;
        anchors.leftMargin: 12;
        anchors.rightMargin: 12;
        anchors.topMargin: 6;
        anchors.bottomMargin: 6;
```

```
            spacing: 4;
            Text {
                id: state;
                font.pointSize: 14;
                color: "blue";
            }
            Text {
                id: progress;
                font.pointSize: 12;
                color: "white";
            }
        }
    }

    MediaPlayer {
        id: player;

        property var utilDate: new Date();
        function msecs2String(msecs){
            utilDate.setTime(msecs);
            return Qt.formatTime(utilDate, "mm:ss");
        }
        property var sDuration;

        onPositionChanged: {
            progress.text = msecs2String(position) + sDuration;
        }
        onDurationChanged: {
            sDuration = " / " + msecs2String(duration);
        }
        onPlaybackStateChanged: {
            switch(playbackState){
            case MediaPlayer.PlayingState:
                state.text = "播放中";
                break;
            case MediaPlayer.PausedState:
                state.text = "已暂停";
                break;
            case MediaPlayer.StoppedState:
                state.text = "停止";
                break;
            }
        }
        onStatusChanged: {
            switch(status){
            case MediaPlayer.Loading:
            case MediaPlayer.Buffering:
                busy.running = true;
                break;
            case MediaPlayer.InvalidMedia:
                root.statusBar.text = "无法播放";
            case MediaPlayer.Buffered:
            case MediaPlayer.Loaded:
                busy.running = false;
                break;
            }
        }
    }
}


function processFile(fileUrl, ext){
    var i = 0;
    for(; i < fileDialog.nameFilters.length; i++){
```

```
            if(fileDialog.nameFilters[i].search(ext) != -1) break;
        }
        switch(i){
        case 0:
            if(centralView.current != textView){
                if(centralView.current != null){
                    centralView.current.visible = false;
                }
                textView.visible = true;
                centralView.current = textView;
            }
            textView.loadText(fileUrl);
            break;
        case 1:
            if(centralView.current != imageViewer){
                if(centralView.current != null){
                    centralView.current.visible = false;
                }
                imageViewer.visible = true;
                centralView.current = imageViewer;
            }
            imageViewer.source = fileUrl;
            break;
        case 2:
        case 3:
            if(centralView.current != videoOutput){
                if(centralView.current != null){
                    centralView.current.visible = false;
                }
                videoOutput.visible = true;
                centralView.current = videoOutput;
            }
            player.source = fileUrl;
            player.play();
            break;
        default:
            statusBar.text = "抱歉，处理不了";
            break;
        }
    }

    function showAbout(){
        if(aboutDlg == null){
            aboutDlg = Qt.createQmlObject(
                    'import QtQuick 2.2;import QtQuick.Dialogs 1.1;MessageDialog{icon:
StandardIcon.Information;title: "关于";\ntext: "仅仅是个示例撒";
\nstandardButtons:StandardButton.Ok;}'
                    , root, "aboutDlg");
            aboutDlg.accepted.connect(onAboutDlgClosed);
            aboutDlg.rejected.connect(onAboutDlgClosed);
            aboutDlg.visible = true;
        }

    }

    function selectColor(func){
        if(colorDlg == null){
            colorDlg = Qt.createQmlObject(
                    'import QtQuick 2.2;import
                      QtQuick.Dialogs 1.1;ColorDialog{}',
                    root, "colorDlg");
            colorDlg.accepted.connect(func);
            colorDlg.accepted.connect(onColorDlgClosed);
            colorDlg.rejected.connect(onColorDlgClosed);
            colorDlg.visible = true;
```

```
        }
    }

    function onAboutDlgClosed(){
        aboutDlg.destroy();
        aboutDlg = null;
    }

    function onColorDlgClosed(){
        colorDlg.destroy();
        colorDlg = null;
    }

    function onTextColorSelected(){
        root.textColor = colorDlg.color;
    }

    function onTextBackgroundColorSelected(){
        root.textBackgroundColor = colorDlg.color;
    }

    FileDialog {
        id: fileDialog;
        title: qsTr("Please choose an image file");
        nameFilters: [
            "Text Files (*.txt *.ini *.log *.c *.h
                         *.java *.cpp *.html *.xml)",
            "Image Files (*.jpg *.png *.gif *.bmp *.ico)",
            "Video Files (*.ts *.mp4 *.avi *.flv *.mkv *.3gp)",
            "Audio Files (*.mp3 *.ogg *.wav *.wma *.ape *.ra)",
            "*.*"
        ];
        onAccepted: {
            var filepath = new String(fileUrl);
            if(Qt.platform.os == "windows"){
                root.statusBar.text = filepath.slice(8);
            }else{
                root.statusBar.text = filepath.slice(7);
            }
            var dot = filepath.lastIndexOf(".");
            var sep = filepath.lastIndexOf("/");
            if(dot > sep){
                var ext = filepath.substring(dot);
                root.processFile(fileUrl, ext.toLowerCase());
            }else{
                root.statusBar.text = "Not Supported!";
            }
        }
    }
}
```

现在还没讲到的代码，就只有不同类型文件的识别与显示这部分了。

（1）不同文件的选择与识别

这个其实蛮简单的。我在 fileDialog 中设置了 4 个不同类型的 nameFilter，对应文本、图片、视频、音频，当相应的菜单项被选中后，切换 fileDialog 的 selectedNameFilter，这样打开文件对话框就会只列出对应的文件供君选择。

而实际上用户还可以在打开的文件对话框内切换名字过滤器，可能从文本文件菜单进去，而实际上选择了视频文件。所以呢，在 fileDialog 的 onAccepted 中，又调用 processFile()，根据文件扩展名做了一次识别，然后就桥归桥路归路喽。

（2）界面的切换

我定义了一个 id 为 centralView 的 Item 对象，没爹没娘就像神笔马良，一切神奇的事情都发生在它身上了。

centralView 有 4 个可见的孩子：用于显示图片的 Image 对象 imageViewer，用于显示文本的 TextArea 对象 textView，用于显示视频的 VideoOutput 对象 videoOutput，显示简单播放信息的 Rectangle 对象 playerState。

实现了 videoOutput 的 onVisibleChanged 信号处理器，将 playerState 的 visible 属性设置成与 videoOutput 一样。而 playerState 的 Z 序大，所以它总在上面。另外，还在 onVisibleChanged 信号处理器内当 videoOutput 不可见时调用 player.stop()来停止播放。

所以实际上有三个互斥的 Item，一开始 visible 都为 false。当用户选择了文件后，根据文件后缀来区分文件类型，然后把对应的 Item 的 visible 属性设置为 true。

17.7.2　C++代码

其实，我只对模板生成的 C++代码改动了三处：包含 QIcon 和 QApplication 对应的头文件，将 QGuiApplication 换成 QApplication，设置应用图标。main.cpp 如下：

```
#include <QApplication>
#include <QQmlApplicationEngine>
#include <QIcon>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    app.setWindowIcon(QIcon(":/res/eye.png"));

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:///main.qml")));

    return app.exec();
}
```

17.7.3　项目文件

FileViewer.pro 也做了一些改动，为 QT 变量添加了 widgets、network、multimedia 模块。如下：

```
TEMPLATE = app
QT += qml quick network multimedia widgets
SOURCES += main.cpp
RESOURCES += qml.qrc
QML_IMPORT_PATH =
include(deployment.pri)
HEADERS +=
```