第14章 多媒体

我们经常要看视频，听音乐，在移动平台上偶尔还会自拍一下美颜，甚至还可能在微信聊天时按住说话……这些场景用到的多媒体功能，Qt Quick 中都支持，这章咱们就简略地过一下。

Qt 的多媒体模块，其实是个壳儿，封装了对其他多媒体框架的调用细节，向开发者提供平台无关的一致的 API。而实际上呢，在 Windows 系列平台上，它使用 DirectShow；在 Linux 系列平台上，它使用 GStreamer；在 Android 平台上，它对接 Android 的 MediaPlayer 框架，有兴趣的可以探秘这里：C:\Qt\Qt5.3.1\5.3\Src\qtmultimedia\src\plugins\android 目录，其中对 Android 多媒体框架的调用在 jar\src\org\qtproject\qt5\android 目录下，而 C++代码则在 src 和 videonode 目录下。如果你回到 android 目录的上一级，即 plugins 目录，就可以看到 gstreamer、directshow 等插件。

我无意研究 Qt Multimedia 模块对各个多媒体框架的封装细节，本章仅仅针对 QML 提供的 MediaPlayer 及相关辅助控制类来做一些简略的介绍。

14.1 MediaPlayer

MediaPlayer 是 QML 提供的核心多媒体类，可以播放音频、视频。

要使用 MediaPlayer，需要引入 QtMultimedia 模块，在 QML 文档的开始加入“import QtMultimedia 5.0”语句。QML 中的 MediaPlayer 是 Qt C++中的多媒体框架在 QML 环境中的代言人，假如以 Qt QuickApp 为模板创建项目，你还需要在 pro 文件中加入语句：QT += multimedia。

14.1.1 播放音乐

先看最简单的播放音乐的例子，simple_music.qml，就十来行代码：

```
import QtQuick 2.2
import QtMultimedia 5.0
Rectangle {
    width: 200;
    height: 100;
```

```
    MediaPlayer {
        autoPlay: true;
        source: "wangjie_game_and_dream.mp3";
    }
}
```

关于 MediaPlayer 的，就 4 行代码。在 QML 文档同目录下放置对应的 MP3 文件，执行“qmlscene simple_music.qml”命令，音乐就会响起来：

不要谈什么分离
我不会因为这样而哭泣
那只是昨夜的一场梦而已
……

我用的王杰的歌，《一场游戏一场梦》，很好听。

在这个简单的示例中，source 属性指定了要播放的文件，它的类型是 url，它能接受绝对路径、相对路径、有效的 http 链接。autoPlay 属性设置为 true，指示 MediaPlayer 对象创建后立即开始播放。

如果你觉得立马开始播放不太好，也可以在你认为合适的时候调用 MediaPlayer 的 play() 方法。

如果你想知道音乐的时长，访问 duration 属性，它是整型值，单位是毫秒，实现 onDurationChanged 信号处理器，可以取到时长。

如果你还想知道播放进度，可以访问 position 属性，它是整型值，单位是毫秒，实现一个 onPositionChanged 信号处理器，就可以实时显示进度。

调用 pause()方法暂停播放，调用 stop()方法停止播放。而播放状态变化时，会发出 playbackStateChanged()信号，在 onPlaybackStateChanged 信号处理器内，可以读取枚举类型的 playbackState 属性，它取 MediaPlayer.PlayingState、MediaPlayer.PausedState、MediaPlayer.StoppedState 三个值中的一个。

seekable 属性指示媒体是否支持 seek，当它为 true 时，你就可以调用 seek（offset）方法来定位播放了。参数 offset 是相对于当前位置的偏移量，单位是毫秒。注意，操作可能是异步的，当方法返回时 position 属性不一定会立即变成新的。

静音可以通过 muted 属性读取、设置，音量通过 volume 属性读取、设置。

好啦，常用操作就这么多，现在让我们来打造一个功能更丰富的简易音乐播放器。还记得“找图看实例”中我们实现的 FlatButton 吗？这里要用到它。另外还要找几个播放控制图标。simple_music_player.qml 的内容如下：

```
import QtQuick 2.2
import QtMultimedia 5.0
Rectangle {
    width: 320;
    height: 240;
    color: "black";

    property var utilDate: new Date();

    function msecs2String(msecs){
        utilDate.setTime(msecs);
        return Qt.formatTime(utilDate, "mm:ss");
    }
```

```
MediaPlayer {
    id: player;
    source: "wangjie_game_and_dream.mp3";
    onPositionChanged: {
        progress.text = msecs2String(position) + progress.sDuration;
    }
    onDurationChanged: {
        progress.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;
        }
    }
}

Row {
    id: controller;
    anchors.top: parent.verticalCenter;
    anchors.horizontalCenter: parent.horizontalCenter;
    anchors.topMargin: 4;
    spacing: 4;
    FlatButton {
        width: 50;
        height: 50;
        iconSource: "ic_rew.png";
        onClicked: if(player.seekable)player.seek(player.position - 5000);
    }
    FlatButton {
        width: 50;
        height: 50;
        iconSource: "ic_pause.png";
        onClicked: player.pause();
    }
    FlatButton {
        width: 50;
        height: 50;
        iconSource: "ic_play.png";
        onClicked: player.play();
    }
    FlatButton {
        width: 50;
        height: 50;
        iconSource: "ic_stop.png";
        onClicked: player.stop();
    }
    FlatButton {
        width: 50;
        height: 50;
        iconSource: "ic_ff.png";
        onClicked:if(player.seekable)player.seek(player.position + 5000);
    }
```

```
    }
    Text {
        id: progress;
        anchors.left: controller.left;
        anchors.bottom: controller.top;
        anchors.bottomMargin: 4;
        color: "white";
        font.pointSize: 12;
        property string sDuration;
    }
    Text {
        id: state;
        anchors.left: progress.left;
        anchors.bottom: progress.top;
        anchors.bottomMargin: 4;
        color: "white";
        font.pointSize: 12;
    }
}
```

使用 qmlscene 加载 simple_music_player.qml，效果如图 14-1 所示。

图 14-1　简易音乐播放器

这个版本的播放器依旧很简单，只能播放一个内置的文件，也没有可拖动的进度条，不过 MediaPlayer 的常用接口都用到了。界面上的那些事儿，播放列表那些事儿，都可以在此基础上加进来，再找个妹妹帮你做些漂亮的图片，你就可以打造出一个酷炫的音乐播放器了。

14.1.2　视频

播放视频比音乐稍稍复杂一些，需要使用 VideoOutput 元素与 MediaPlayer 配合。VideoOutput 用来渲染视频，也可以作为相机的取景器（预览窗口），最简单的用法是，你只需要将其 source 属性指向一个 MediaPlayer 对象即可。

simple_video.qml 演示如何播放一个本地视频：

```
import QtQuick 2.2
import QtMultimedia 5.0
Rectangle {
    width: 720;
    height: 480;

    MediaPlayer {
        id: player;
        source: "test.ts";
        onError: {
```

```
            console.log(errorString);
        }
    }

    VideoOutput {
        anchors.fill: parent;
        source: player;
    }

    MouseArea {
        anchors.fill: parent;
        onClicked: {
            player.play();
        }
    }
}
```

如你所见，视频的示例相比音频，仅仅多了一个VideoOutput对象，其source属性为player。

当用鼠标左键单击时，调用play()播放视频。视频文件我放在QML文档所在目录，你也可以使用本地路径，如“e:/temp/media/test.ts”，或者http链接，如“http://video.xxx.com/xxx.flv”。

如果你在Windows平台上使用qmlscene加载simple_video.qml，不一定能够播放哦，要确保你的系统安装了必需的DirectShow Filter才行。如果不能播放，可以尝试安装LAV Filters（请找度娘要下载地址）。LAV Filters是一组基于ffmpeg的DirectShow分离器和音视频解码器，支持绝大多数常见的音视频格式。

图14-2是播放效果图。

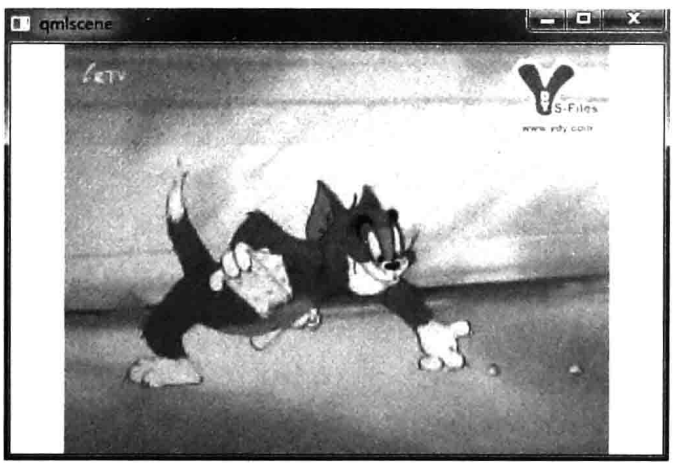

图14-2　QML播放视频

视频播放的控制，如暂停、停止、定位等，与音乐一样，不多说了。接下来我们看看如何获取多媒体的元信息。

14.1.3　多媒体元信息

何谓多媒体元信息？就是媒体以外、用来描述媒体的那些信息，比如一首歌，专辑、发行时间、艺术家、采样率等，就是元信息；又如一个视频，分辨率、编码格式、帧率等，就是元信息。

MediaPlayer对象有个分组属性metaData，它包含了方方面面的元信息，通过访问它，

你就可以知道多媒体的描述信息。不过呢，这些看上去很美的元信息，不一定可用哦，要看 MediaPlayer 使用的底层的播放服务是否能够提供这些。如果你需要这些信息，可以这么获取：

在 onStatusChanged 信号处理器中，读取 status 属性，当它的值为 MediaPlayer.Loaded 时，访问你想要的信息。就像下面这样：

```
MediaPlayer {
    id: player;
    source: "wangjie_game_and_dream.mp3";

    onStatusChanged: {
        switch(status){
        case MediaPlayer.Loaded:
            console.log(metaData.albumArtist,
                        metaData.albumTitle,
                        metaData.author, metaData.channelCount);
            break;
        }
    }
}
```

我用的 MP3 文件，只能取到 channelCount 这个信息，输出结果为 2，说明是双声道。

如果你想了解多媒体元信息的更多细节，请在 Qt 帮助中查看 MediaPlayer 的文档。

14.2　拍照

从 Qt SDK 5.2 开始，在移动平台（如 Android 手机）上可以使用 QML 来拍照和录制视频了。我们单讲拍照，录制视频且不去管它。

一般地，我们在使用手机拍照时都会在屏幕上预览图像，看看相机指向哪儿，看看焦点在哪儿，然后根据预览调整相机方向，调焦，调亮度，调闪光……如此反复，觉得过得去了，就按下快门，然后呢，咔哒一声响，大尺寸的图片被镜头捕获，辅以适当的图像处理优化后保存下来，最后你可以进入浏览模式查看大图。

这就是拍照的过程，在开发应用时，实际上可以拆分为三步：

① 配置相机。

② 设置取景器。

③ 调整参数，抓取图像。

我们就据此来对照着讲解 QML 提供的拍照类库。

注意：Qt 相机只能在 Android 3.0 及以上系统中使用。

14.2.1　配置 Camera

Camera 对象用来访问和控制系统的物理设备，完成拍照功能。它能控制闪光、曝光、聚焦等与拍照相关的各种设置。

digitalZoom 配置数字变焦；maximumDigitalZoom 保存相机支持的最大数字变焦倍数，如果是 1.0，则说明不支持数字变量。

opticalZoom 配置光学变焦（很少有手机提供这么高级的玩意儿）倍数；maximumOpticalZoom

保存相机支持的最大光学变焦倍数,值为 1.0 表示不支持光学变焦。

captureMode 属性是个枚举值，支持 Camera.CaptureViewfinder（仅预览）、Camera.CaptureStillImage（捕获静态图片）、Camera.CaptureVideo（录制视频）三种模式。

在 Qt 帮助的 Camera 手册中列出的主要属性也就这些了。要想控制其他选项，就必须请出 Camera 的重量级的小伙伴们了。先看下我从 Camera 对应的 C++类型的头文件里摘出来的几行源码：

```
Q_PROPERTY(QDeclarativeCameraCapture* imageCapture READ imageCapture CONSTANT)
Q_PROPERTY(QDeclarativeCameraRecorder* videoRecorder READ videoRecorder CONSTANT)
Q_PROPERTY(QDeclarativeCameraExposure* exposure READ exposure CONSTANT)
Q_PROPERTY(QDeclarativeCameraFlash* flash READ flash CONSTANT)
Q_PROPERTY(QDeclarativeCameraFocus* focus READ focus CONSTANT)
Q_PROPERTY(QDeclarativeCameraImageProcessing* imageProcessing READ imageProcessing
CONSTANT)
```

如你所见，Camera 其实还有几个属性：imageCapture、videoRecorder、exposure、flash、focus、imageProcessing。

（1）CameraFocus

focus 是 Camera 的一个属性，类型是 CameraFocus，主要用来控制聚焦和焦点模式。例如：

```
Camera {
    id: camera

    focus {
        focusMode: Camera.FocusMacro
        focusPointMode: Camera.FocusPointCustom
        customFocusPoint: Qt.point(0.2, 0.2) // Focus to top-left
    }
}
```

focusMode 属性是个枚举值，对应的枚举类型定义了 6 种聚焦方式，Camera.ManualFocus 代表手动聚焦，Camera.AutoFocus 代表自动聚焦，Camera.MacroFocus 代表微距聚焦，用于拍摄离相机很近的物体，比如正在采花的小蜜蜂。其他的几种聚焦类型都是 AF 或 MF 的锦上添花式补充，请参看 Qt 文档来了解详情。实际上手机不是每种聚焦方式都支持，CameraFocus 类的 isFocusModeSupported（mode）用来查询物理设备是否支持指定的聚焦方式。

focusPointMode 属性定义焦点模式，也是个枚举值，对应的枚举类型定义了 4 种焦点模式：Camera.FocusPointAuto（自动焦点）、Camera.FocusPointCenter（中心焦点）、Camera.FocusPointFaceDetection（面部识别）、Camera.FocusPointCustom（自定义焦点，使用 customFocusPoint 属性指定的点作为焦点）。在设置焦点模式前，最好也判断下设备是否支持，isFocusPointModeSupported（mode）方法可以帮助你。

（2）CameraExposure

CameraExposure 用来控制相机的曝光选项，你可以通过 Camera 的 exposure 属性完成这项复杂又晦涩的工作。

先说说曝光模式，对应的属性为 exposureMode，是枚举类型，可以取 Camera.ExposureAuto（自动）、Camera.ExposureManual（手动）、Camera.ExposureSports（运动）、Camera.ExposureLargeAperture（大光圈、近景）、Camera.ExposureSmallAperture（小光圈、远景）、Camera.ExposureBacklight（背光）、Camera.ExposureNight（夜间）、Camera.ExposurePortrait（人物）、Camera.ExposureBeach（海岸）等，完整的列表请参考 Qt 帮助。

CameraExposure 还可以控制快门速度，shutterSpeed 属性表示当前的快门速度，manualShutterSpeed 代表手动设置的快门速度，它们的单位是秒。setAutoShutterSpeed()可以打开自动快门模式。

再说说感光选项，iso 属性表示当前的感光系数，manualIso 保存手动设置的感光度，而 setAutoIsoSensitivity()可以打开自动感光模式。iso 值越大，感光越快，成像越模糊；iso 值越小，感光越慢，得到的图像越细腻。

最后说光圈吧，aperture 表示当前光圈大小，manualAperture 表示手动设置的光圈大小，而 setAutoAperture()可以打开自动模式。光圈一般以 F 加数字来表示，如 F3.5。

补充说明一个属性：exposureCompensation 表示曝光补偿。

照片的好坏与曝光量有关，也就是说，应该通多少的光线使镜头能够得到清晰的图像。曝光量又与通光时间（快门速度决定）、通光面积（光圈大小决定）有关，而快门速度又与光圈大小、感光速度有关……总之，相机的各种设置相互牵扯，很是复杂，一般人都是傻瓜式操作什么都自动化，比如我给媳妇拍照，要么是她笑面瘫了我还没调整好呢，要么是她还没摆好 POSE 我就说照好了……

CameraExposure 允许你控制曝光相关的各种选项，如果你想制作专业的拍照应用，一定要弄明白各种选项的含义，不但如此，还要拎得清如何组合不同的选项以获得最佳的拍照效果。

示例代码片段：

```
Camera{
    id: camera;

    exposure.manualAperture: 3.5;
    exposure.manualShutterSpeed: 0.1;
    exposure.exposureCompensation: -1.0;
    exposure.exposureMode: Camera.ExposureAuto;
    Component.onCompleted:{
        exposure.setAutoIsoSensitivity();
    }
}
```

（3）CameraFlash

flash 选项（CameraFlash 类型）是控制闪光的，主要的选项就一个：mode。mode 是枚举类型，可以取这些值：Camera.FlashOff（闪光灯关闭）、Camera.FlashOn（闪光灯打开）、Camera.FlashAuto（自动）、Camera.FlashRedEyeReduction（去红眼）……完整的选项请参考 Qt 帮助吧。

示例代码片段：

```
Camera {
    id: camera;

    flash.mode: Camera.FlashRedEyeReduction;
}
```

（4）CameraImageProcessing

很多相机固件支持对镜头捕获的图片做一些处理，如降噪、白平衡、锐化、对比度、饱和度等。Camera 的 imageProcessing 属性可以设置这些选项，当然前提是相机固件支持。

imageProcessing 属性的类型是 CameraImageProcessing，我们来看看它支持的选项。

contrast 属性用来设置对比度，在−1.0～1.0 之间取值，默认值是 0。

denoisingLevel 设置降噪级别，为−1.0 表示禁止降噪处理，为 0 表示应用默认的降噪处理，为 1.0 则告诉相机尽最大可能降噪。

sharpeningLevel 用于设置锐化级别，为−1.0 表示禁止锐化，为 0 表示应用默认的锐化级别，为 1.0 通知相机尽最大可能锐化。

whiteBalanceMode 是个枚举值（枚举类型属于类 CameraImageProcessing），设置白平衡模式，提供 WhiteBalanceManual（手动，在此种模式下 manualWhiteBalance 属性生效）、WhiteBalanceAuto（自动）、WhiteBalanceSunlight（日光）、WhiteBalanceCloudy（阴天）、WhiteBalanceShade（阴影）、WhiteBalanceTungsten（钨光、白炽灯、室内光）、WhiteBalanceFluorescent（荧光灯）、WhiteBalanceFlash（闪光）、WhiteBalanceSunset（日落）等白平衡模式。所谓白平衡，就是数码相机对白色物体的还原。当我们用肉眼观看这大千世界时，在不同的光线下，对相同的颜色的感觉基本是相同的，比如在早晨旭日初升时，我们看一个白色的物体，感觉到它是白的；而我们在夜晚昏暗的灯光下，看到的白色物体，感觉到它仍然是白的。这这这，人眼和大脑的适应性太强了！可是镜头只能客观地记录环境颜色，并且 CCD/CMOS 输出经常不平衡，就会导致成像后色彩失真，比如在日光灯下拍摄出来的样片就偏暗偏黄……因此就需要调整感光器件对各种颜色的感应强度，使色彩平衡。白色物体在不同光线下，人眼很容易确认其为白色，所以就将白色作为确认其他颜色是否平衡的标准，当白色能反映为白色时，其他颜色也就正常了，这就是白平衡的含义。

saturation 属性代表饱和度，取值在−1.0～1.0 之间，默认值为 0。

14.2.2 设置取景器 VideoOutput

所谓取景器，就是一个预览相机视频的窗口，和数码相机的液晶显示屏类似。像智能手机，一般只能使用数字的虚拟取景器（就是液晶屏，放到程序里就是一个特殊的显示视频的 Surface），而数码相机根据配置不同，则可以有光学取景器、TTL 取景器、电子取景器等专业取景器。

取景器让你看到焦点在哪里，图像是否发虚，角度是否合适等。

在 QML 里，VideoOutput 可以作为取景器使用，只需将其 source 属性设置为 Camera 对象即可。例如：

```
Rectangle {
    width: 960;
    height: 540;

    Camera {
        id: camera;
        captureMode: Camera.CaptureStillImage;
    }

    VideoOutput {
        source: camera;
        anchors.fill: parent;
    }
}
```

VideoOutput 的 fillMode 属性设置图像的填充模式，支持 VideoOutput.Stretch、VideoOutput.

PreserveAspectFit、VideoOutput.VideoOutput 三种模式，默认为 PreserveAspectFit。

autoOrientation 属性是从 QtMultimedia5.2 引入的，在拍照时，你设置它为 true，VideoOutput 会自动调整预览视频的方向，使得你看到的被摄物体方向正确；否则，哈哈，如果 orientation 也没设置，在 Android 手机上你就只能修炼歪脖神功了。这个属性的默认值为 false。

orientation 属性用于修正视频方向，整型值，它取正值（如 90）时图像会顺时针旋转，取负值（如-90）时图像会逆时针旋转。它也是用来修正拍照时取景器的预览图像与实际被摄物体之间的角度偏差的。当设置 autoOrientation 为 true 时，orientation 就不起作用了。

14.2.3　捕获静态图片

拍照的最后一步，就是你手指点一下，然后咔哒一声。当你手指点击界面上那个按钮时，代码中实际上调用了 CameraCapture 类的 capture()或 captureToLocation（location）方法来捕获图片。

Camera 的 imageCapture 属性是 CameraCapture 类的实例。CameraCapture 的只读属性 captureImagePath 说明了图片的保存位置；resolution 属性设置图片分辨率，如果你不设置，系统会帮你选一个合适的分辨率。

当照片拍完后，会触发 CameraCapture 的 imageCaptured（requestId, preview）信号，它的第二个参数是个字符串，指向小尺寸预览图像的位置，你可以用 Image 对象把它显示出来。当图片保存成功后，会触发 imageSaved（requestId, path）信号，其中 path 为字符串，是一个本地路径，不是 URL。如果捕获失败，会触发 captureFailed（requestId, message）信号，字符串类型的 message 参数给出了失败原因。

如果你直接调用 imageCapture.capture()或 captureToLocation()来拍照，可能得到的图像是虚的，因为它没有执行聚焦、寻找焦点这些过程。所以呢，我们一般要先调用 Camera 的 searchAndLock()方法，锁定动作完成后，lockStatus 属性会变化，实现 onLockStatusChanged 信号处理器就可以监听锁定结果，当 lockStatus 属性的值为 Camera.Locked 时说明锁定成功，此时再调用 imageCapture.capture()，得到的图像焦点就清晰了。

14.2.4　简单的拍照实例

笔者拍照时最擅长的是自动模式，拿着单反相机也只会用自动，什么光圈、曝光、白平衡等选项，对我只是个摆设。我设计的实例，simpleCamera，也没有太多的可选项，只支持调整数字放大、调节闪光灯以及少数几种曝光模式。拍照实在是一个高深的技术活儿啊，还要求灵性（这玩意儿我是没有），所以 simpleCamera 的目标仅仅是演示 Camera 的用法，提供的曝光选项设置可能得不到最好的成像效果。

项目创建请参考 2.3 节，另外请把 13.8.3 节提到的 FlatButton.qml 复制一份，放在项目目录下并加入到 qrc 中，还有很多个图标文件需要加入到 qrc 中。

为了适应项目需要，我对 FlatButton 做了一些修改，让图标在上，文字在下，居中对齐。

在我的笔记本电脑上运行 simpleCamera，效果如图 14-3 所示。

图 14-3　拍照模式预览效果

闪光灯按钮，点击会在自动、打开、关闭三种模式间切换，图标和文字都会变化。控制面板的最右侧是曝光模式按钮，支持自动、人物、夜间、运动、风景 5 种模式，点击可以在不同模式间切换。

点击控制面板中间的快门按钮，会捕获图像并跳转到图片查看模式，如图 14-4 所示。

图 14-4　图片查看模式

我在图片查看界面左下角放置了一个相机按钮，点击它可以回到拍照模式。

在写作本书的过程中，我换了酷派的 K1（米少就买国货吧），电信版，支持 4G，拍照实例在它上面做了测试。Android 系统版本为 4.3，满足最低 3.0 的要求。图 14-5 是预览界面。

看到了吧，我的预览视频方向正确。图 14-6 是图片查看效果图。

对于放大、缩小、闪光以及曝光这些选项，在我的 DELL 笔记本电脑上，都是无效的。而在我的酷派 K1 上，都是可以用的。你也可以在手机上试试，看哪些选项能用。

图 14-5　手机预览效果

图 14-6　手机上查看拍摄的图片

好啦，现在来看看 main.qml：

```
import QtQuick 2.2
import QtQuick.Window 2.1
import QtMultimedia 5.2
import QtQuick.Controls 1.1

Window {
    visible: true;
    width: 600;
    height: 400;
    color: "black";

    Camera {
        id: camera;
        captureMode: Camera.CaptureStillImage;

        focus {
            focusMode: Camera.FocusAuto;
            focusPointMode: Camera.FocusPointCenter;
        }

        imageProcessing {
            whiteBalanceMode: CameraImageProcessing.WhiteBalanceAuto;
        }
        flash.mode: Camera.FlashAuto;

        imageCapture {
            resolution: Qt.size(640, 480);
            onImageCaptured: {
                camera.stop();
                photoPreview.visible = true;
```

```
                actionBar.visible = false;
                viewfinder.visible = false;
                photoPreview.source = preview
            }
            onImageSaved: {
                console.log(path);
            }
        }
        onLockStatusChanged: {
            switch(lockStatus){
            case Camera.Locked:
                console.log("locked");
                imageCapture.captureToLocation("capture.jpg");
                unlock();
                break;
            case Camera.Searching:
                console.log("searching");
                break;
            case Camera.Unlocked:
                console.log("unlocked");
                break;
            }
        }
    }

    VideoOutput {
        id: viewfinder;
        source: camera;
        focus : visible;
        anchors.fill: parent;
        autoOrientation: true;
    }

    Image {
        id: photoPreview;
        anchors.fill: parent;
        visible: false;
        fillMode: Image.PreserveAspectFit;

        FlatButton {
            iconSource: "res/ic_launcher_camera.png";
            width: 76;
            height: 76;
            anchors.left: parent.left;
            anchors.bottom: parent.bottom;
            anchors.margins: 8;
            onClicked: {
                camera.start();
                actionBar.visible = true;
                viewfinder.visible = true;
                photoPreview.visible = false;
            }
        }
    }

    Image {
        id: actionBar;
        source: "res/control_bar.png";
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 8;
        anchors.horizontalCenter: parent.horizontalCenter;
        z: 1;
        FlatButton {
            id: shutter;
            anchors.centerIn: parent;
            iconSource: "res/ic_cam_shutter.png";
```

```
    width: 88;
    height: 88;
    iconWidth: 84;
    iconHeight: 84;
    onClicked: {
        camera.searchAndLock();
    }
}
FlatButton {
    id: zoomout;
    anchors.verticalCenter: shutter.verticalCenter;
    anchors.right: shutter.left;
    anchors.rightMargin: 4;
    width: 70;
    height: 70;
    iconWidth: 40;
    iconHeight: 40;
    text: "缩小";
    font.pointSize: 12;
    iconSource: "res/ic_zoom_out.png";
    onClicked: {
        if(camera.digitalZoom > 1){
            camera.digitalZoom -= 1;
        }
    }
}
FlatButton {
    id: zoomin;
    anchors.verticalCenter: shutter.verticalCenter;
    anchors.right: zoomout.left;
    anchors.rightMargin: 4;
    width: 70;
    height: 70;
    iconWidth: 40;
    iconHeight: 40;
    text: "放大";
    font.pointSize: 12;
    iconSource: "res/ic_zoom_in.png";
    onClicked: {
        if(camera.digitalZoom < camera.maximumDigitalZoom){
            camera.digitalZoom += 1;
        }
    }
}
FlatButton {
    id: currentFlash;
    anchors.verticalCenter: shutter.verticalCenter;
    anchors.left: shutter.right;
    anchors.leftMargin: 4;
    width: 70;
    height: 70;
    iconWidth: 40;
    iconHeight: 40;
    font.pointSize: 12;
    property var modeIcon: [
        "res/ic_menu_stat_flash_auto.png",
        "res/ic_menu_stat_flash.png",
        "res/ic_menu_stat_flash_off.png"
    ]
    property var modeDesc: [
        "自动", "打开", "关闭"
    ]
    property var flashMode: [
```

```
                Camera.FlashAuto, Camera.FlashOn, Camera.FlashOff
            ]
            property int mode: 0;
            text: modeDesc[mode];
            iconSource: modeIcon[mode];
            onClicked: {
                mode = (mode + 1)%3;
                camera.flash.mode = flashMode[mode];
            }
        }
        FlatButton {
            id: currentScene;
            anchors.verticalCenter: shutter.verticalCenter;
            anchors.left: currentFlash.right;
            anchors.leftMargin: 4;
            width: 70;
            height: 70;
            iconWidth: 40;
            iconHeight: 40;
            font.pointSize: 12;

            property var modeIcon: [
                "res/ic_menu_stat_auto.png",
                "res/ic_menu_stat_portrait.png",
                "res/ic_menu_stat_landscape.png",
                "res/ic_menu_stat_night.png",
                "res/ic_menu_stat_action.png"
            ]
            property var modeDesc: [
                "自动", "人物", "风景", "夜间", "运动"
            ]
            property var exposureMode: [
                Camera.ExposureAuto, Camera.ExposurePortrait,
                Camera.ExposureBeach, Camera.ExposureNight,
                Camera.ExposureSports
            ]
            property int mode: 0;
            text: modeDesc[mode];
            iconSource: modeIcon[mode];
            onClicked: {
                mode = (mode + 1)%5;
                camera.exposure.exposureMode = exposureMode[mode];
            }
        }
    }
}
```

代码 200 行，有点长，咱简短点儿说。

（1）界面切换

在预览模式下，取景器是充满窗口的，下方有控制面板，所有支持的操作都在上面放着。当点击快门按钮时，触发拍照，然后跳转到图片查看界面。当点击查看界面上的相机按钮时，会回到预览模式。

界面切换是通过 Item 的 visible 属性控制的。

（2）相机选项控制

定义 Camera 对象时，基本上各种选项都是自动的。

点击放大、缩小按钮，在 onClicked 信号处理器中更改 Camera 的 digitalZoom 属性来调整焦距。

在闪光灯按钮对应的对象内定义了 modeIcon、modeDesc、flashMode 三个类型为数组的属性，还有一个类型为 int 的属性，方便在点击按钮时循环切换按钮的图片和文字。在 onClicked 信号处理器内，我们变换 mode 属性，设置 camera.flash.mode 的值。

曝光选项按钮的实现与闪光灯按钮类似，只是它在 onClicked 中修改的是 camera.exposure.exposureMode 属性。

（3）取景器设置

我特意设置 VideoOutput 的 autoOrientation 为 true，保证你看到的预览视频与实际被摄物体方向一致。与此对应，必须引入 QtMultimedia 5.2。

另外，需要注意的就是，在 Android 平台上，企图通过设置 width、height 属性让 VideoOutput 对象龟缩一隅是难以得逞的，你只能接受它霸占整个屏幕，所以，乖乖地使用“anchors.fill: parent”这种设置吧；如果 VideoOutput 的父不是顶层对象，那还要一级一级地传递这种设置。

（4）拍照

快门按钮触发拍照动作，首先调用 camera.searchAndLock()，然后 onLockStatusChanged 信号处理器发现锁定成功时调用 camera.imageCapture.captureToLocation()进行图像捕获，最后我们为 imageCapture 实现了 onImageCaptured 信号处理器，将 preview 拿给 Image 对象去显示。