移动与嵌入式开发技术

利用Android多媒体API创建动态移动应用程序

Android多媒体开发高级编程

——为智能手机和平板电脑开发图形、音乐、视频和富媒体应用

Pro Android Media: Developing Graphics, Music, Video and Rich Media Apps for Smartphones and Tablets

（美）Shawn Van Every 著
巢文涵 译

Apress®
清华大学出版社

移动与嵌入式开发技术

Android 多媒体开发高级编程

——为智能手机和平板电脑开发图形、音乐、视频和富媒体应用

(美) Shawn Van Every 著

巢文涵 译

清华大学出版社

北 京

Shawn Van Every
Pro Android Media: Developing Graphics, Music, Video and Rich Media Apps for Smartphones and Tablets
EISBN：978-1-4302-3267-4
Original English language edition published by Apress, 2855 Telegraph Avenue, #600, Berkeley, CA 94705 USA. Copyright © 2009 by Apress L.P. Simplified Chinese-language edition copyright © 2011 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Apress 出版公司授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号　图字：01-2011-2258

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

Android 多媒体开发高级编程——为智能手机和平板电脑开发图形、音乐、视频和富媒体应用/(美)艾佛瑞(Every, S. V.) 著；巢文涵 译. —北京：清华大学出版社，2012.2
(移动与嵌入式开发技术)
书名原文：Pro Android Media: Developing Graphics, Music, Video and Rich Media Apps for Smartphones and Tablets
ISBN 978-7-302-27889-4

Ⅰ. A…　Ⅱ. ①艾… ②巢…　Ⅲ. 移动终端—应用程序—程序设计　Ⅳ. TN929.53

中国版本图书馆 CIP 数据核字(2012)第 008697 号

责任编辑：王　军　张立浩
装帧设计：牛艳敏
责任校对：成凤进
责任印制：杨　艳
出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　　地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
　　社 总 机：010-62770175　　**邮　　购**：010-62786544
　　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　　质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京市密东印刷有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：19　　**字　　数**：462 千字
版　　次：2012 年 2 月第 1 版　　**印　　次**：2012 年 2 月第 1 次印刷
印　　数：1～3500
定　　价：48.00 元

产品编号：040482-01

作者简介

Shawn Van Every 是一位资深的移动和流媒体顾问，他帮助公司更好地利用与音频和视频相关的新兴技术，主要是开发移动和流媒体应用程序。他的客户范围从 19 Entertainment、MoMA 和迪斯尼(Disney)，到 Morgan Stanley、雷曼兄弟(Lehman Brothers)和纽约大学医学院(NYU Media School)，同时还包括无数的创业公司和其他小型客户。

此外，Shawn 是纽约大学交互式电信计划(Interactive Telecommunications Program)中通信领域内的一位兼职助理教授。他的教学范围很广泛，包括参与性和社会媒体、编程、移动技术以及交互式电话等课程。他于 2008 年获得了大卫佩恩卡特(David Payne Carter)卓越教学奖。

他在许多会议和技术展示中演示、展示和介绍了其工作内容(包括 O'Reilly 的新兴电话，O'Reilly 的新兴技术、ACM Multimedia、Vloggercon 以及 Strong Angle II)。他是 Open Media Developers Summit，Beyond Broadcast(开放媒体开发者峰会，超越广播)(2006)以及 iPhoneDevCamp NYC(NYC iPhone 开发夏令营)的联合组织者。

Shawn 在纽约布法罗市的 SUNY 大学获得多媒体研究(Media Study)方向的学士学位，并在纽约大学获得交互式电信方向的硕士学位。

技术编辑简介

Steve Bull 自从加入 Paul Allen 的 Interval Research 公司(位于加利福尼亚州的帕洛阿尔托市)之后一直从事编码工作和操纵移动设备。Bull 是一位资深的混合媒体技术艺术家和企业家，在过去 9 年中，他创建了许多特定位置的叙事型游戏，这些游戏利用了手机的社会性、技术性和创造性。可以通过 www.stevebull.org 与 Steve 取得联系。

Wallace Jackson 是一位经验丰富的多媒体制作人员和 i3D 程序员，主要面向 Acrobat3D PDF、Android 移动应用程序、iTV 设计、JavaFX 和 JavaTV。自 Atari ST1040 和 AMIGA 3000 以来，他一直在设计富媒体；并且自从 20 年前 *Multimedia Producer* 杂志诞生以来，他一直为顶级的多媒体出版物撰写关于新媒体内容开发的文章。可以通过 www.wallacejackson.com 与 Wallace 取得联系。

此为试读，需要完整PDF请访问：www.ertongbook.com

致　谢

本书的灵感来自于我在纽约大学的教学工作。非常感谢曾经鼓励过我的教师、工作人员以及学生，他们构建了纽约大学的交互式电信计划，同时提供了无止境的灵感来源。感谢 Red Burns 创建、推进以及改进 ITP。感谢 Dan O'Sullivan 不断向我挑战。感谢 Tom Igoe 和 Dan Shiffma 不断鼓励我可以完成本书。感谢 Rob Ryan 和 Marianne Petite 提供的所有支持。感谢所有其他和我一起共事过的教师、工作人员。感谢我所有现在和过去的学生，他们使我意识到讲授并看到项目逐步变得鲜活是多么伟大的奖赏；特别要感谢 Nisma Zaman，他提供了非常宝贵的早期反馈。

如果不是 Apress 出版社专业且很有天分的工作人员，本书不会如此顺利出版。感谢 Steve Anglin、Matthew Moodie、Corbin Collins、Mary Ann Fugate、Adam Heath、Anne Collette 和其他 Apress 工作人员的非凡努力。

非常感谢 Steve Bull 和 Wallace Jackson，作为技术审稿者，他们测试了每一个代码片段并填补了我错过的空白。你们的贡献非常宝贵！

如果没有 Android 的开发人员，那么本书也不会被撰写。感谢他们，特别是来自 Google 的 Dave Sparks，他提供了一些非常宝贵的事实检测和问题解答。

对于所有鼓励我的朋友和家人，我要衷心谢谢你们。

最后，如果没有我可爱的妻子 Karen Van Every 的支持，那么本书当然也不会存在。谢谢你！

前　言

在移动电话本身和其已经成为的所有事物当中，有一个明显的趋势是它们提供的多媒体生产和消费功能在不断地增长。这一趋势从20世纪90年代末具备摄像功能的手机出现开始，在过去几年随着人气激增的智能手机而戏剧性地兴起。在多媒体功能方面，今天的手机同时是照相机、相册、摄像机、电影播放器、音乐播放器、听写机，且可能具备更多功能。

特别是，Android在SDK中具有非常丰富的功能，本书将试图通过讨论和实例对该SDK进行介绍，从而使您能够着手开发下一代多媒体应用程序。本书所讲解的示例不仅介绍了如何显示和播放多媒体，而且还允许您利用摄像头、麦克风以及视频捕获功能。本书大致由3个部分组成：前4章将处理图像；接下来的4章处理音频；而最后4章将介绍视频，以及利用Web服务来查找和共享多媒体。

由于为介绍功能而开发的应用程序所需完成的工作量在不断地增加，因此随着本书的介绍，所展示的示例将越来越具有挑战性。不管怎样，如果对Android应用程序的开发有一点熟悉，读者就应该能够跳转到任何章节，利用讨论及示例代码创建一个利用当前展示功能的应用程序。

示例通常采取扩展了Activity的完整类的形式，用于在SDK版本4(Android 1.6)或更高版本上运行。示例还会包含XML布局文件的内容，而且在许多情况下包含AndroidManifest.xml文件的内容。本书假设您将使用带ADT插件(0.9.9或更新版本)的Eclipse(Galileo或更新版本)，并使用Android SDK(r7或更新版本)。因为本书主要是面向音频和视频，所以建议您在手机(运行Android 1.6或更新版本)而非仿真程序上运行示例，因为在许多情况下示例在仿真器中不能正常运行。

我很期待能看到多媒体应用程序在移动设备上的未来。希望能通过这本书帮助您创建并定义这一美好未来。期待看到您实际开发的Android多媒体应用程序。

把所有这些都先搁在一边，让我们开始学习本书吧！

目　录

第1章

Android 图像概述

本章将介绍有关 Android 上图像捕获和存储的基础知识。首先将探索 Android 所提供的内置功能，然后在后续章节中更多地介绍定制软件。内置的图像捕获与存储功能为 Android 上的所有媒体功能提供了一个很好的切入点，为我们在以后的章节中处理音频和视频奠定了基础。

考虑到这一点，我们将首先介绍如何利用内置的 Camera(摄像头)应用程序，然后介绍如何利用 MediaStore——内置的媒体和元数据存储机制。接着，将研究如何减少内存的使用量以及如何利用 EXIF——在消费类电子产品和图像处理软件世界中用于共享元数据的标准。

1.1 使用内置的 Camera 应用程序捕获图像

随着移动电话迅速成为移动计算机，它们在许多方面已经取代了各种各样的消费类电子产品。最早添加到移动电话上且和电话无关的硬件功能之一是摄像头。现在，似乎很难想象有人会购买一部不包含摄像头功能的移动电话。当然，基于 Android 的电话也不例外；从一开始，Android SDK 就支持访问电话内置的硬件摄像头来捕获图像。

在 Android 上，完成许多事情的最便捷方式是通过使用意图(intent)来利用该设备上的某个现有软件。意图是 Android 的核心组件，在文档中将它解释为一个“将要执行的操作的描述”。在实践中，意图用于触发其他应用程序来完成某件事情，或者在单个应用程序的活动之间进行切换。

所有带有合适硬件(摄像头)的原版 Android 设备都会附带 Camera 应用程序。Camera 应用程序包含一个意图过滤器(intent filter)，它使得开发人员能够提供与 Camera 应用程序同等的图像捕获能力，而不必构建他们自己的定制捕获例程。

意图过滤器是程序员用于指定其应用程序能够提供某个特定功能的一种方法。在应用程序的 AndroidManifest.xml 文件中指定一个意图过滤器，将会告诉 Android，这个应用程

此为试读，需要完整PDF请访问：www.ertongbook.com

序(尤其是包含意图过滤器的活动)将根据指令执行指定的任务。

Camera 应用程序在其清单文件中指定了以下意图过滤器。这里显示的意图过滤器包含在“Camera”活动标记内。

```
<intent-filter>
     <action android:name="android.media.action.IMAGE_CAPTURE"/>
     <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

为了通过一个意图利用 Camera 应用程序，我们所要做的仅仅是必须构造一个将由上述过滤器捕获的意图。

```
Intent i = new Intent("android.media.action.IMAGE_CAPTURE");
```

在实践中，我们可能不希望直接使用动作字符串创建意图。在这种情况下，可以指定 MediaStore 类中的常量 ACTION_IMAGE_CAPTURE。应该使用常量而非字符串本身的原因在于，如果该字符串发生了改变(当然常量也可能会不断地改变)，那么使用常量将使得我们的调用比之前使用字符串更有利于未来的变化。

```
Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
startActivity(i);
```

在一个基本的 Android 活动中使用这种意图，将导致默认的 Camera 应用程序以静止图片模式(still picture mode)启动，如图 1-1 所示。

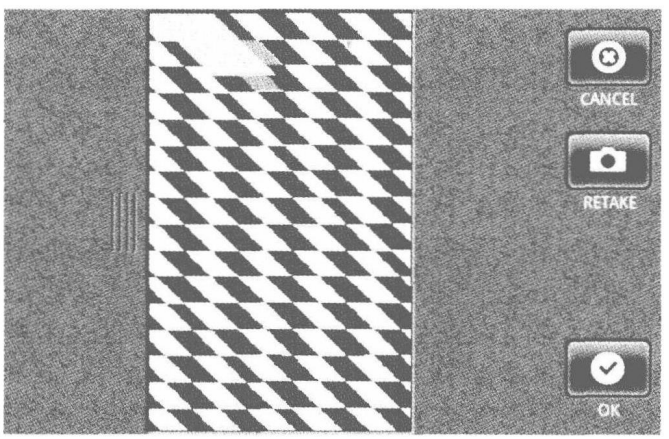

图 1-1　通过意图调用的内置 Camera 应用程序在模拟器中的运行结果

1.1.1　从 Camera 应用程序返回数据

当然，在捕获一张图片时，如果 Camera 应用程序没有将图片返回给调用活动，那么简单地使用内置的 Camera 应用程序捕获图像将不具有真正的作用。而为了使得它真正有用，可以将活动中的 startActivity 方法替换为 startActivityForResult 方法。使用该方法将允许我

们访问从 Camera 应用程序中返回的数据，它恰好是用户以位图(Bitmap)形式捕获的图像。

以下是一个基本的示例。

```
package com.apress.proandroidmedia.ch1.cameraintent;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.widget.ImageView;

public class CameraIntent extends Activity {

    final static int CAMERA_RESULT = 0;

    ImageView imv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Intent i = new Intent(android.provider
                                    .MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(i, CAMERA_RESULT);
    }

    protected void onActivityResult(int requestCode, int resultCode,
         Intent intent) {
        super.onActivityResult(requestCode, resultCode, intent);

        if (resultCode == RESULT_OK)
        {
            Get Bundle extras = intent.getExtras();
            Bitmap bmp = (Bitmap) extras.get("data");


            imv = (ImageView) findViewById(R.id.ReturnedImageView);
            imv.setImageBitmap(bmp);
        }
    }
}
```

它需要在项目的 layout/main.xml 文件中添加如下内容：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    >
    <ImageView android:id="@+id/ReturnedImageView"android:layout_width=
      "wrap_content"android:layout_height="wrap_content"></ImageView>
</LinearLayout>
```

为了完成上述示例，以下是 AndroidManifest.xml 文件的内容。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:versionCode="1"
      android:versionName="1.0"package="com.apress.proandroidmedia.ch1
                                                        .cameraintent">
   <application android:icon="@drawable/icon" android:label=
        "@string/app_name">
        <activity android:name=".CameraIntent"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
   </application>
   <uses-sdk android:minSdkVersion="4" />
</manifest>
```

在此示例中，Camera 应用程序在一个通过意图传递的附加值(extra)中返回图像，而该意图将在 onActivityResult 方法中传递给主调活动。附加值的名称为“data”，它包含一个 Bitmap 对象，需要从泛型对象将它强制转换过来。

```
//从意图中获取附加值
Bundle extras = intent.getExtras();

//从附加值中获取返回的图像
Bitmap bmp = (Bitmap) extras.get("data");
```

在我们的布局 XML (layout/main.xml)文件中，有一个 ImageView 对象。ImageView 是泛型视图的扩展，其支持图像的显示。由于我们有一个带有指定 ReturnedImageView 编号(id)的 ImageView 对象，因此需要在活动中获得它的引用，并通过 setImageBitmap 方法将它的 Bitmap 对象设置为返回的图像。这将使得应用程序用户能够查看这幅捕获的图像。

为了获得 ImageView 对象的引用，使用在 Activity 类中指定的标准方法 findViewById。该方法使得我们能够以编程方式引用在布局 XML 文件中指定的元素，我们正在通过将元素 id 传递给 setContentView 来使用该布局 XML 文件。上述示例在 XML 中以如下方式指定 ImageView 对象：

```
<ImageView android:id="@+id/ReturnedImageView" android:layout_width=
"wrap_content" android:layout_height="wrap_content"></ImageView>
```

为了引用 ImageView 并通知它显示来自 Camera 的 Bitmap 对象，使用以下代码。

```
imv = (ImageView) findViewById(R.id.ReturnedImageView);
imv.setImageBitmap(bmp);
```

当运行这个示例时，您可能会注意到结果图像很小(在我的手机上，它的宽为 121 像素，高为 162 像素。其他设备会具有不同的默认大小)。这不是一个 bug——相反，它是经过精心设计的。当通过一个意图触发时，Camera 应用程序不会将全尺寸的图像返回给主调活动。通常，这样做需要大量的内存，而移动设备一般会在内存方面受限。相反，Camera 应用程序将在返回的意图中返回一幅很小的缩略图，如图 1-2 所示。

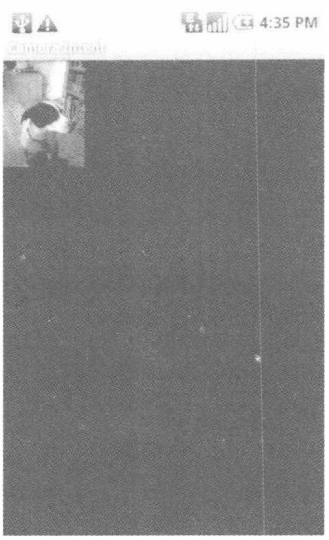

图 1-2　在 ImageView 中显示的 121×162 像素的结果图像

1.1.2　捕获更大的图像

为了绕过大小限制，从 Android 1.5 开始，在大多数设备上可以将一个附加值传递给触发 Camera 应用程序的意图。这个附加值的名称在 MediaStore 类中指定，它是一个常量，称为 EXTRA_OUTPUT。这个附加值(采用名-值对的形式)将以 URI 的方式指示 Camera 应用程序您想要将捕获的图像保存在什么位置。

以下代码片段指示 Camera 应用程序应该将图像保存到设备的 SD 卡上，文件名为 myfavoritepicture.jpg。

```
String imageFilePath = Environment.getExternalStorageDirectory()
       .getAbsolutePath() + "/myfavoritepicture.jpg";
File imageFile = new File(imageFilePath);
Uri imageFileUri = Uri.fromFile(imageFile);

Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
i.putExtra(android.provider.MediaStore.EXTRA_OUTPUT, imageFileUri);
startActivityForResult(i, CAMERA_RESULT);
```

注意：上述为图像文件创建 URI 的代码片段可以简化为下列形式：

```
imageFileUri = Uri.parse("file:///sdcard/myfavoritepicture.jpg");
```

然而在实践中，使用以上所示的方法将会使得代码更加具有设备独立性，并且对于 SD 卡的命名约定或本地文件系统的 URI 语法变化具有更好的适应性。

1.1.3 显示大图像

加载并显示一幅图像对内存使用情况具有显著的影响。例如，HTC G1 电话带有一个 320 万像素的摄像头。320 万像素的摄像头通常会捕获 2048×1536 像素的图像。显示如此大小的 32 位图像将需要超过 100 663kb 或大约 13MB 的内存。虽然我们的应用程序不一定会因此而耗尽内存，但是这肯定会使得内存更加容易耗尽。

Android 提供了一个名为 BitmapFactory 的实用程序类，该程序类提供了一系列的静态方法，允许通过各种来源加载 Bitmap 图像。针对我们的需求，将从文件加载图像，并在最初的活动中显示它。幸运的是，BitmapFactory 中的可用方法将会调用 BitmapFactory.Options 类，这使得我们能够定义如何将 Bitmap 读入内存。具体而言，当加载图像时，可以设置 BitmapFactory 应该使用的采样大小。在 BitmapFactory.Options 中指定 inSampleSize 参数，这将表明一旦加载时结果 Bitmap 图像所占的比例。例如，在这里将 inSampleSize 设置为 8，这会产生一幅大小是原始图像大小 1/8 的图像。

```
BitmapFactory.Options bmpFactoryOptions = new BitmapFactory.Options();
bmpFactoryOptions.inSampleSize = 8;
Bitmap bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);
imv.setImageBitmap(bmp);
```

这是一种快速加载大图像的方法，但是没有真正考虑图像的原始大小，也没有考虑屏幕的大小。最好能够将图像缩放到刚好适合屏幕。

下面的代码片段演示了如何使用显示维度来确定在加载图像时应该发生的减采样量。当使用这些方法时，应确保该图像尽可能多地填充显示范围。但如果该图像只是要在任何一个维度中显示 100 个像素，那么应该使用这个值而不是显示维度，可以通过如下方式获得该值。

```
Display currentDisplay = getWindowManager().getDefaultDisplay();
int dw = currentDisplay.getWidth();
int dh = currentDisplay.getHeight();
```

为了确定图像的所有尺寸(用于计算)，我们使用了 BitmapFactory 和 BitmapFactory. Options，并将 BitmapFactory.Options.inJustDecodeBounds 变量设置为 true。这将通知 BitmapFactory 类只须返回该图像的范围，而无须尝试解码图像本身。当使用此方法时，BitmapFactory. Options.outHeight 和 BitmapFactory.Options.outWidth 变量将会被赋值。

```
// 加载图像的尺寸而不是图像本身
BitmapFactory.Options bmpFactoryOptions = new BitmapFactory.Options();
bmpFactoryOptions.inJustDecodeBounds = true;
Bitmap bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);

int heightRatio = (int)Math.ceil(bmpFactoryOptions.outHeight/(float)dh);
int widthRatio = (int)Math.ceil(bmpFactoryOptions.outWidth/(float)dw);

Log.v("HEIGHTRATIO",""+heightRatio);
Log.v("WIDTHRATIO",""+widthRatio);
```

简单地将图像的尺寸除以显示的尺寸将获得显示的比率。然后，可以选择是否使用高度比率或宽度比率，这取决于它们当中谁更大。只须将这个比率作为 BitmapFactory.Options.inSampleSize 变量，这将产生一幅应该加载到内存中的图像，其尺寸接近于我们在这种情况下所需要的尺寸，也接近于显示本身的尺寸。

```
// 如果两个比率都大于 1，
// 那么图像的一条边将大于屏幕
if (heightRatio > 1 && widthRatio > 1)
{
    if (heightRatio > widthRatio)
    {
        // 若高度比率更大，则根据它缩放
        bmpFactoryOptions.inSampleSize = heightRatio;
    }
    else
    {
        // 若宽度比率更大，则根据它缩放
        bmpFactoryOptions.inSampleSize = widthRatio;
    }
}

//对它进行真正的解码
bmpFactoryOptions.inJustDecodeBounds = false;
bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);
```

下面是通过一个意图使用内置摄像头并显示结果图片的完整示例代码。图 1-3 显示了一幅由此示例生成的屏幕大小的结果图像。

```
package com.apress.proandroidmedia.ch1.sizedcameraintent;

import java.io.File;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
```


此为试读，需要完整PDF请访问：www.ertongbook.com