HZ BOOKS
华章IT

CSDN博客专家撰写，多年工作与实践研究的沉淀，帮助读者跨越从理论到实践的鸿沟

系统全面地介绍图像处理各种方法与常见应用场景的编程实现，揭示Java Swing图形与图像编程基本API的使用技巧，逐级深入以实际应用来加深对各知识点编程实践的理解

Java
核心技术
系列

Java
数字图像处理
编程技巧与应用实践

Digital Image Process in Java

贾志刚 著

机械工业出版社
China Machine Press

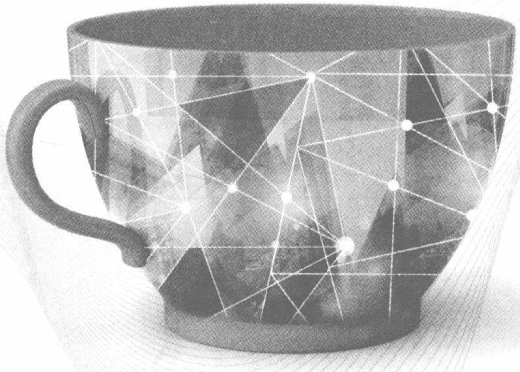
Java 核心技术系列

Java 数字图像处理

编程技巧与应用实践

Digital image process in Java

贾志刚 著



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Java 数字图像处理：编程技巧与应用实践 / 贾志刚著 . —北京：机械工业出版社，2015.11
（Java 核心技术系列）

ISBN 978-7-111-51946-1

I. J… II. 贾… III. JAVA 语言－程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2015）第 256250 号

Java 数字图像处理：编程技巧与应用实践

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：佘 洁 杨绣国 责任校对：殷 虹
印 刷：北京文昌阁彩色印刷有限责任公司 版 次：2016 年 1 月第 1 版第 1 次印刷
开 本：186mm×240mm 1/16 印 张：21.75
书 号：ISBN 978-7-111-51946-1 定 价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzit@hzbook.com

华章IT
HZBOOKS | Information Technology

Preface 前　言

为什么要写这本书

我对图像处理的认识最初来自于读软件工程专业时做毕业设计论文的需要，毕业论文做完以后，我便把所有关于图像处理的知识扔到了一边。2011 年的一天有位朋友问了我几个简单的图像处理方面的问题，在解答问题的过程中我发现自己对图像处理的热情一直都在燃烧，从那一刻起我决定重新学习图像处理。这之后，我把以前买的几本图像处理的书都读了一遍，同时还坚持通过写博客来督促自己加深理解，随着学习的不断深入，对图像处理的认知也在不断加深，我越来越渴望自己能实现那些书中提到的图像处理手段与方法，于是便开始不断尝试，在经过了各种“坑”与无助之后，我终于编程实现了学习过的每一种图像处理方法。这个过程十分痛苦，因为我深刻感受到了图像处理在理论与实践之间的细微差异，而这些细微差异往往会导致处理结果与理论预期相差很大。

可能提到图像处理，很多人马上就会想到相关书籍中各种复杂的数学公式与矩阵计算，然后就会说我数学不好学不了这个，早早地就把自己给否定了。那些数学公式的确让人望而生畏，但是只要仔细探究一番，就会发现它在图像处理的应用上远远没有看上去那么复杂，甚至可以说十分简单，这是本人学习图像处理时得到的最大一个心得体会，正如一句俗语说的：“世上无难事，只怕有心人”。

正是因为自己在学习过程中经历了痛苦，所以我想写一本不一样的图像处理入门图书，内容不再是冰冷的数学公式与文字描述，而是基于理论的实践步骤和细节详解，是一个个可以直接运行的代码实现，书中没有大量的数学公式，有的只是数学知识的巧妙运用。我希望通过分享自己学习过程中的体会与编程实践经验，帮助更多人在学习图像处理的道路上少走弯路，早日进入图像处理的科学殿堂。

在国内，程序员写书早已经不是什么新鲜事物，但是我可以肯定地说，本书是国内第一本由奋斗在编码一线的码农写的图像处理入门图书。它不是当下流行的视觉图像处理库的应用介绍，而是图像处理基础知识和理论的学习与实践，正如一句西方科技谚语所说的那样，“在理论上，理论与实践是一致的，在实践上，它们是不一致的”。当前关于图像处理的书大

多数都是重理论而轻实践，但图像处理在理论与实践编程之间是存在轻微差异的，而这就成了很多初学者无法逾越的鸿沟。本书就是要拟合理论与实践之间的鸿沟，帮助读者架起从理论到实践的大桥。

作为工作超过十年的程序员写的第一本书，本书也是我个人职业生涯的一个新起点，它鞭策与勉励自己不断努力上进，除了对图像处理的兴趣外，这一年多写书的动力更多的是毅力与帮助后来者的初衷。只要本书能为国内图像处理专业知识的普及与应用实践略尽绵薄之力，那辛苦也就值了。

读者对象

本书适合以下人群阅读：

- 从事图像处理的工作人员
- 学习图像处理的爱好者
- 希望提升自我的中高级程序员
- 计算机专业高年级本科生或研究生
- 开设图像处理相关课程的大专院校学生
- 从事 Java 应用的开发者

如何阅读本书

本书分为两大部分，其中第一部为前三章，主要介绍 Java Swing 编程的基础知识。第二部分是本书的核心内容，系统全面地介绍图像处理的各种方法与常见应用场景编程实现。如果你已经对 Java 语言和 Java Swing 有基本的认识，可以跳过前三章，直接从第 4 章开始阅读本书。同时本书注重实践，所以请务必阅读给出的源代码并运行它，这样才能更好地理解所讲的知识。

第一部分为基础篇，简单地介绍了 Java Swing 图形与图像编程基本 API 使用技巧，以及相关实践编程，帮助读者了解图像接口在 Java 语言中的基础知识，并熟悉像素的读写与操作。

第二部分为实践与应用编程，从最基础的像素操作开始，通过实践编程讲解图像处理过程中各种基本像素运算、混合、图像插值、直方图获取与图像搜索、图像卷积、边缘提取、二值图像分析与特征提取等知识，最后通过剖析一个流行的图像油画转换算法编程实践来结束本书。

附录为本书相关数学知识简单参考。其他参考资料索引可在我的 Github 上找到。

此外，本书的源文件可到 www.hzbook.com 上通过搜索本书下载，或者到 github 上下载。

勘误和支持

由于作者的水平有限，编写的时间也很仓促，书中难免会出现一些错误或不准确的地方，恳请读者批评指正。本书配套源代码已上传到 github 上，访问地址为：https://github.com/

gloomyfish/mybook-java-imageprocess，如果有读者想直接提交勘误之后的代码，请先邮件联系本人，同意以后即可提交，同时本人也会根据读者反馈修改更新源代码。如果你有更多的宝贵意见，也欢迎发送邮件至我的邮箱 bfnh1998@hotmail.com，我很期待能够听到你们的真挚反馈。

致谢

首先要感谢图像处理学科那些伟大的先行者，是他们创立了这个影响力巨大的学科。其次要感谢 CSDN 博客频道，在 CSDN 我结识了很多良师益友，他们直言不讳地指出了我博客文章中的很多不妥之处与需要改进的地方，特别是 Trent、jichen324、qiwenmingshiwo、FDHGVH2461、cr459464757、wust 小吴、xiaowei_cqu，以及这个仓促写就的名单之外的更多朋友，感谢你们的宝贵建议。

感谢机械工业出版社华章公司的编辑杨绣国老师，你的一封电子约稿邮件促成了本书，也帮助我实现了写一本注重实践的图像处理入门图书的梦想；感谢你的耐心，在这一年多时间里你不厌其烦地回答我在写作过程中一个又一个问题；感谢你的魄力和远见，始终支持我的写作，你的鼓励和帮助引导我顺利完成全部书稿。

最后一定要感谢我的父母，感谢你们将我培养成人；感谢我的妻子在我写书的这一年多时间让我从家务中解脱，给我支持与鼓励。

谨以此书，献给我最亲爱的两个孩子，以及众多热爱图像处理的朋友们。

贾志刚

中国，苏州，2015 年 9 月

目　录 *Contents*

第 1 章 Chapter 1

Java Graphics 及其 API 简介

在开始本书内容之前，笔者假设你已经有了面向对象语言编程的基本概念，了解 Java 语言的基本语法与特征，原因在于本书的所有源代码都是基于 Java 语言实现的，而且是基于 Java 开发环境运行与演示所有图像处理算法的。本书第 1 章到第 3 章是为了帮助读者了解与掌握 Java 图形与 GUI 编程的基本知识与概念而写的。本章主要介绍 Java GUI 编程中基本的图形知识，针对 GUI 编程，Java 语言提供了两套几乎并行的 API，分别是 Swing 与 AWT。早期的 Java GUI 编程中主要使用 AWT 的相关组件，但是 AWT 的功能并不是十分强大，而且严重依赖本地接口。于是在 Java 1.3 及后续版本中引入了 Swing 工具实现 GUI 编程，Swing 中的组件大多数都是基于纯 Java 语言实现的，而不是通过本地组件实现的，所以它们是轻量级的 GUI 组件，同时 Swing 对图形与图像的支持操作也有很大的提高与增强。如何区分 AWT 组件与 Swing 组件？一个简单而且相当直观的方法是看 Class 的名称，Swing 的组件大多数带有大写的前缀字母 J。

Graphics 作为 Java 的图形引擎绘制接口，几何形状、文字、图像的绘制都必须通过它完成，此外，Graphics 还支持绘制过程的控制，可以设置画笔颜色、纹理、颜色填充方法、合成与裁剪路径及各种 Stroke 与 Fill 的属性等。用户程序通常都是通过 Graphics 来访问绘制引擎，从而实现各种图形与图像绘制的，因此可以说 Graphics 是 Swing 中最重要的接口对象。好吧，下面让我们一起揭开 Graphics 的神秘面纱。

1.1 什么是 Java 图形设备 Graphics

简单地说 Graphics 是 Java 图形绘制引擎的访问接口，只有通过它才可以访问到 Java GUI

的图形绘制引擎，实现图形的绘制与绘制过程的控制。

1.1.1 Graphics 概述

Grahpics 的功能大致可以分为两类，第一类是通过 Draw 或 Fill 方法来实现各种图形的绘制与填充，第二类是设置各种绘制属性，最简单的包括设置字体、颜色、填充方法等。此外，在 Java 2D 中 Graphics 还可以被转型为 Graphics2D 对象，从而提供更高精度的图形绘制，设置更多绘制属性来控制绘制过程。

1.1.2 Graphics 图形设备的获取、使用和销毁

在 Java Swing 中正确获取 Graphics 对象的方法有两种。

第一种是从 BufferedImage 对象实例中获取，其代码如下：

```
bufferedImage.createGraphics() // bufferedImage为对象实例
```

第二种方法是通过重载 Swing 组件的 paintComponent（Graphics g）或 paint（Graphics g）方法来实现，个人推荐采用重载 paintComponent（Graphics g）方法来实现，因为重载 paint（Graphics g）是 AWT 时代遗留下来的产物，是一个重量级绘制重载，通常用于 Canvas 对象的重载绘制。

除了以上两种推荐的做法以外，笔者经常还看到直接通过 Swing 组件的 getGraphics 去获取 Graphics 对象的，这样做的坏处是一旦该组件没有被显示，所对应的 Graphics 对象将返回 NULL。而且这种做法常会导致一些意想不到的错误，所以应该尽量避免这么做。下面提供一个这么做导致错误的代码示例，如下：

```
JButton okBtn = new JButton("OK");
okBtn.getGraphics().drawRect(0,0,20,20); // NullPointerException
```

在获取了 Java 图形设备对象 Graphics 之后，就可以调用它的绘制方法来实现图形绘制与填充了。简单的示例代码如下：

```
public void paintComponent(Graphics g)
{
    g.setColor(Color.BLUE);
    g.drawRect(10, 10, 50, 50);
}
```

上述代码将会绘制一个蓝色边框的矩形，其中“10，10”表示矩形开始绘制的左上角位置，“50，50”分别代表矩形的长度与高度。

在使用完 Graphics 对象以后，请记得一定要销毁图形设备对象，可通过调用方法 dispose() 来释放图形绘制时所使用的任何资源。特别是当图形设备是从你自己的 BufferedImage 对象中创建出来的时候，记得使用完以后一定要调用 dispose() 方法来释放资源。假设没有调用 dispose()，一般情况下 Java 的 GC 也会自动调用来释放资源，但还是强烈建议在绘

制完成以后显式调用 dispose() 方法来确保被使用的资源得到及时释放而不是依赖 Java GC。原因在于当你使用的 Graphics 来自 BufferedImage 对象时，Graphics 对象不会被自动销毁，而依赖 GC 调用来清理与释放资源并不能保证及时释放，可能导致程序堆内存过度消耗产生 OOM（Out of Memory）问题。

1.1.3　Java Swing Graphics2D 的重要属性

Graphics 可以向下转型为 Graphics2D 对象，可以通过设置绘制属性来实现对图形绘制质量的控制。其接受对象为 RenderingHints 的枚举类型，通过方法 setRenderingHint（RenderingHints.key, RenderingHints.value）来实现，一般常用的 Key 与 Value 有如下形式。

控制图形边缘反锯齿时，RenderingHints.KEY_ANTIALIASING 的值为：

- RenderingHints.VALUE_ANTIALIAS_ON 表示支持边缘反锯齿。
- RenderingHints.VALUE_ANTIALIAS_OFF 表示不支持边缘反锯齿。

控制文字或文本边缘反锯齿时，RenderingHints.KEY_TEXT_ANTIALIASING 的值为：

- RenderingHints.VALUE_ TEXT_ANTIALIAS_ON 表示支持文本边缘反锯齿。
- RenderingHints.VALUE_ TEXT_ANTIALIAS_OFF 表示不支持文本边缘反锯齿。

控制图像的插值方法时，KEY_INTERPOLATION 的值为：

- RenderingHints.VALUE_INTERPOLATION_BICUBIC 表示使用双立方插值方法。
- RenderingHints.VALUE_ INTERPOLATION_BILINEAR 表示使用双线性插值方法。
- RenderingHints.VALUE_ INTERPOLATION_NEAREST_NEIGHBOR 表示使用临近点插值方法。

控制绘制方法时，KEY_RENDERING 的值为：

- RenderingHints.VALUE_ RENDER_QUALITY 表示支持绘制质量优先。
- RenderingHints.VALUE_ RENDER_SPEED 表示支持绘制速度优先。

控制绘制过程是否支持抖动时，KEY_DITHERING 的值为：

- RenderingHints.VALUE_ DITHER_DISABLE 表示不支持抖动。
- RenderingHints.VALUE_ DITHER_ENABLE 表示支持抖动。

更多的绘制属性控制可以参考官方文档，需要强调的是，由于 Java 的跨平台属性导致并不是所有的 RenderingHints 设置都会起作用，因此有些属性可能只有在某些特定的平台才支持。但是最常见的图形与文本的反锯齿功能几乎所有的操作系统平台都支持！

1.2　Java 2D API

当 Graphics 向下转型为 Grahpics2D 时，Java 2D 的图形绘制引擎得以访问，一个功能更加丰富的图形库呈现在读者眼前，它就是 Java 2D API。如果你问笔者 Java 2D 与 Swing 有何关系，可以很认真地说，二者毫无瓜葛，Java 通过引入 Swing、Java 2D 与 Java 3D，极大地

丰富了 Java 的图形功能，使应用程序接口更加完善，为各种可能的图形开发提供了可靠保证与全面支持，从而也使得学习 Java 图形方面的知识时不再那么无助了。下面来看一下 Java 2D 对图形支持与改进都包括了哪些：

- 为显示设备与打印机提供统一的绘制引擎。
- 一个广泛的几何形状支持。
- 文档打印支持。
- 可控制的绘制质量。
- 增强的色彩支持。
- 文字、形状、图像绘制检测。

1.2.1 基本的 Java 2D 图形绘制

Java 2D 图形绘制支持的图形形状如图 1-1 所示。

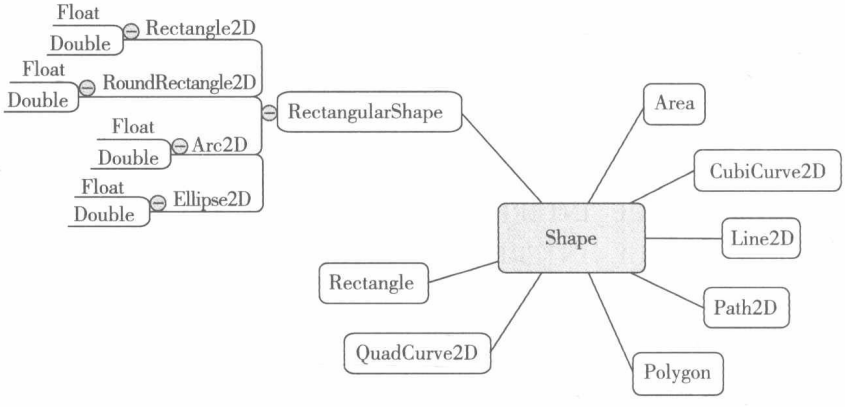


图 1-1 Java2D 形状绘制支持接口

Java 2D 图形绘制最常见的是将绘制代码放在 paintComponent（Graphics g）方法中，显示时 Swing 会首先调用 paint() 方法。该方法会调用下面的三个方法：

- paintComponent（Graphics g）
- paintBorder（Graphics g）
- paintChildren（Graphics g）

在绝大多数情况下，图形绘制只需要重载 paintComponent() 方法来实现。一个基本图形绘制代码如下：

```
public void paintComponent(Graphics g)
{
    Graphics2D g2d = (Graphics2D)g;
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);              //反锯齿
```

```
    g2d.setPaint(Color.BLUE);              // 设置画笔颜色
    g2d.drawRect(10, 10, 50, 50);          // 绘制矩形
    g2d.dispose();                         // 释放资源
}
```

1.2.2　使用 Java 2D 实现太极图形绘制

太极在中国源远流长，黑白相间的太极图案已经是一个文化标志，这里将使用 Java 2D 的图形绘制技术实现太极图案的绘制。如果仔细观察太极图案，就会发现它是非常精准的黑白对称图案。可通过设置画笔颜色来实现黑白颜色控制，利用 Java 2D Area 对图形布尔操作的支持实现太极图形绘制。Java 2D Area 对图形 Shape 对象进行支持的四种布尔操作如下。

- Union（加操作）：保留两个几何形状及其重叠部分。
- Subtraction（减操作）：从第一个几何形状减去与第二个几何形状重叠的部分。
- Intersection（可以看成与操作）：只保留两个几何形状重叠的部分。
- Exclusion-or（XOR 异或操作）：保留两个几何形状不重叠的部分。

这四种操作的示意图如图 1-2 所示。

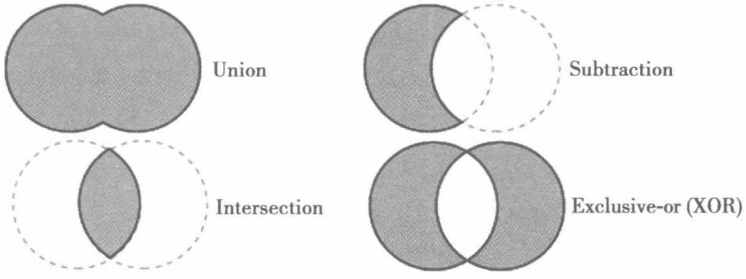


图 1-2　Java 2D 支持的四种几何操作

实现太极图案的相关代码如下：

```
protected void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
    // R = 150
    Shape lefthalfCirle = new Ellipse2D.Double(10,10, 300,300);
    // R = 150
    Shape righthalfCircle = new Ellipse2D.Double(10,10, 300,300);
    // R/2 = 75
    Shape innerCircle1 = new Ellipse2D.Double(85,10, 150,150);
    // R = 150
    Shape innerCircle2 = new Ellipse2D.Double(85,160, 150,150);
    Shape rectangel1 = new Rectangle2D.Double(160, 10, 150, 300);
    Shape rectangel2 = new Rectangle2D.Double(10, 10, 150, 300);

    Area left = new Area(lefthalfCirle);
```

```
        Area right = new Area(righthalfCircle);

        Area area11 = new Area(rectangel1);
        Area area22 = new Area(rectangel2);

        left.subtract(area11);
        right.subtract(area22);

        Area inner1 = new Area(innerCircle1);
        Area inner2 = new Area(innerCircle2);

        left.add(inner1);
        right.add(inner2);

        // trick is here !!!
        right.subtract(inner1);


        // create minor circle here!!!  // ++ 60, R = 150
        Shape minorWhiteCircle = new Ellipse2D.Double(150,70, 20,20);
        Shape innerBlackCircle = new Ellipse2D.Double(150,230, 20,20);

        // draw two big frame shape here...
        g2.setPaint(Color.WHITE);
        g2.fill(left);
        g2.setPaint(Color.BLACK);
        g2.fill(right);

        // draw minor circle here!!!
        g2.fill(minorWhiteCircle);
        g2.setPaint(Color.WHITE);
        g2.fill(innerBlackCircle);
    }
```

运行源文件中第 1 章中的完整代码可以看到一个标准的太极图案。

注意 书中所有完整的源代码均已打包上传至 www.hzbook.com 和 github ⊖，下载后按章节索引即可找到相应的代码，强烈建议运行每个源代码实例，将源代码看成本书的一部分。

1.3 用 Java Swing 绘制自定义的 JPanel

Swing 的 JPanel 组件是 GUI 编程中最重要的面板组件，可以通过重写 JPanel 中 paintComponent 方法实现对 JPanel 面板组件的背景颜色的调整或添加背景图片，进而实现自定义

⊖ https://github.com/gloomyfish/mybook-java-imageprocess。

版本的面板（JPanel）组件。只要完成如下几步就可以实现一个简单自定义 JPanel 面板的绘制。

1）实现对 JPanel 面板的继承，代码如下：

```
public class CustomJPanel extends JPanel
{
    // 更多代码
}
```

2）完成对 paintComponent（Graphics g）对象的重载，代码如下：

```
Protected void paintComponent(Grahpics g)
{
    // 绘制代码
}
```

3）访问 Graphics 绘制引擎，设置画笔颜色并完成绘制，在 Java 2D 中 paint 支持三种不同的画笔颜色填充策略，它们分别是：

❑ 单一颜色填充，如 Color.BLUE、Color.RED 等。代码如下：

```
// 单一颜色背景填充
g2.setPaint(Color.BLUE);
```

❑ 线性渐变颜色填充（GradientPaint），可以细分为水平与竖直方向。代码如下：

```
// 水平方向线性渐变颜色填充
Color sencondColor = new Color(99, 153, 255);
GradientPaint hLinePaint = new GradientPaint(0, 0, Color.BLACK,
        this.getWidth(), 0,sencondColor);
g2.setPaint(hLinePaint);

// 竖直方向线性渐变颜色填充
Color controlColor = new Color(99, 153, 255);
GradientPaint vLinePaint = new GradientPaint(0, 0, Color.BLACK,
        0, getHeight() ,controlColor);
g2.setPaint(vLinePaint);
```

❑ 圆周径向渐变颜色填充（RadialGradientPaint），支持两种以上的颜色渐变。代码如下：

```
// 圆周径向渐变颜色填充
float cx = this.getWidth() / 2;
float cy = this.getHeight() / 2;
float radius = Math.min(cx, cy);
float[] fractions = new float[]{0.1f, 0.5f, 1.0f};
Color[] colors = new Color[]{Color.RED, Color.GREEN,
                Color.BLUE};
// cx, cy表示圆周的中心点距离
// radius 表示半径长度，
// fractions表示色彩渐变关键帧位置，每个值取值在0～1之间
// colors 表示颜色数组
RadialGradientPaint rgp = new RadialGradientPaint(cx, cy,
    radius, fractions, colors, CycleMethod.NO_CYCLE);
g2.setPaint(rgp);
```