1. 读取图片

方法一: 通过 java.awt.Toolkit 工具类来读取本地、网络 或 内存中 的 图片（支持 GIF、JPEG 或 PNG）

Image image = Toolkit.getDefaultToolkit().getImage(String filename);

Image image = Toolkit.getDefaultToolkit().getImage(URL url);

Image image = Toolkit.getDefaultToolkit().createImage(byte[] imageData);

方法二: 通过 javax.imageio.ImageIO 工具类读取本地、网络 或 内存中 的 图片（BufferedImage 继承自 Image）

BufferedImage bufImage = ImageIO.read(File input);

BufferedImage bufImage = ImageIO.read(URL input);

BufferedImage bufImage = ImageIO.read(InputStream input);

// 创建空图片

BufferedImage bufImage = new BufferedImage(int width, int height, int imageType);

PS: 推荐使用第二种读取图片的方式，读取后获得的 BufferedImage 有更丰富的 API 对图片进行相关操作。

2. BufferedImage 常用方法

BufferedImage 表示读取到内存中一张图片，首先把一张本地或远程图片读取为 BufferedImage 实例，然后使用其 API 进行相应的操作。

图片的宽高:

int getWidth();

int getHeight();

// 或

int getWidth(ImageObserver observer);

int getHeight(ImageObserver observer);

图片像素操作:

// 设置图片坐标 (x, y) 处的 RGB 像素值

void setRGB(int x, int y, int rgb);

// 获取图片坐标 (x, y) 处的 RGB 像素值

int getRGB(int x, int y);

/\*\*

\* 区域批量设置像素, 点(x, y)的像素值:

\* pixel = rgbArray[offset + (y - startY) \* scansize + (x - startX)];

\*

\* 参数说明:

\* startX: 设置区域起点横坐标

\* startY: 设置区域起点纵坐标

\* w: 设置区域的横向宽度（列数）

\* h: 设置区域的纵向宽度（行数）

\* rgbArray: 取值的像素数组, 数组长度必须大于 offset + h \* scansize

\* offset: rgbArray 数组的偏移量, 即读取数据时从该数组头部忽略跳过的长度

\* scansize: rgbArray 数组的扫描行间隔, 即每一行所需要占用数组内的数据长度, 即为设置区域的每一行设置数据时,

\* 都先从数组中读取 scansize 长度的数据, 如果 scansize 大于列数(w), 则后面多读取的数据将被忽略。

\*

\* PS: 如果想数组中的每一个值都能用上, 通常 offset==0, w==scansize, rgbArray.length==w\*h

\*/

void setRGB(int startX, int startY, int w, int h, int[] rgbArray, int offset, int scansize);

/\*\*

\* 获取图片某区域的所有像素值存入 rgbArray 数组中并返回,

\* 如果 rgbArray 为 null, 则内部会新建一个长度为 offset + h \* scansize 的数组,

\* 其他参数参考上面 setRGB(...) 方法的参数说明。

\*/

int[] getRGB(int startX, int startY, int w, int h, int[] rgbArray, int offset, int scansize);

图片裁剪:

/\*\*

\* 裁剪图片, 参数说明:

\* x: 裁剪起点横坐标

\* y: 裁剪起点纵坐标

\* w: 需要裁剪的宽度

\* h: 需要裁剪的高度

\*/

BufferedImage getSubimage (int x, int y, int w, int h);

图片缩放:

/\*\*

\* 图片缩放, 参数说明:

\* width: 缩放后的宽度

\* height: 缩放后的高度

\* hints: 指示用于图像重新取样的算法类型的标志

\*

\* hints 参数取值为以下之一（Image 类中的常量）:

\* SCALE\_AREA\_AVERAGING: 使用 Area Averaging 图像缩放算法;

\* SCALE\_DEFAULT: 使用默认的图像缩放算法;

\* SCALE\_FAST: 选择一种图像缩放算法，在这种缩放算法中，缩放速度比缩放平滑度具有更高的优先级;

\* SCALE\_REPLICATE: 使用 ReplicateScaleFilter 类中包含的图像缩放算法;

\* SCALE\_SMOOTH: 选择图像平滑度比缩放速度具有更高优先级的图像缩放算法。

\*

\* PS: 获取到的结果是 Image, 如果想转换为 BufferedImage, 可以创建一个相同宽高的空的 BufferedImage,

\* 然后把获取到的 Image 绘制到 BufferedImage 上。

\*/

Image getScaledInstance(int width, int height, int hints);

在图片上绘制几何图形、文字、图片（贴图）:

// 创建图片的画布

Graphics2D createGraphics();

// 获取图片的画布, 此方法实际上内部调用了 createGraphics() 方法, Graphics2D 继承自 Graphics

Graphics getGraphics();

/\*

\* 获取到 Graphics2D 对象后, 调用它的绘图方法，便可在原图片上绘制几何图形、文字、图片（贴图）等。

\* 获取到的画布是新创建的, 使用完后需要调用 Graphics 的 dispose() 方法销毁。

\*/

Graphics2D 的使用，具体的绘制方法，参考: Java绘图: 使用Graphics类绘制线段、矩形、椭圆/圆弧/扇形、图片、文本

3. 保存图片

把内存中的 BufferedImage 持久化保存到本地磁盘（使用 javax.imageio.ImageIO 工具类）:

// 先读取图片

BufferedImage bufImage = ImageIO.read(...);

/\*

\* 对 bufImage 进行相应的操作

\*/

/\*\*

\* 保存 bufImage, 参数说明:

\* im: bufImage 本身, BufferedImage 实现了 RenderedImage 接口

\* formatName: 保存的图片格式的名称, 支持 "JPEG"、"PNG"、"GIF"、"BMP" 等格式

\* output: 结果输出位置

\*/

ImageIO.write(RenderedImage im, String formatName, File output);

ImageIO.write(RenderedImage im, String formatName, OutputStream output);

4. 代码实例

把一张图片进行水平镜像，即依次把图片每一行的第1列像素点和最后1列对调，把第2列和倒数第2列对调，以此类推。

package com.xiets.imagedemo;

import javax.imageio.ImageIO;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

public class Main {

public static void main(String[] args) throws IOException {

// 读取图片

BufferedImage bufImage = ImageIO.read(new File("aa.jpg"));

// 获取图片的宽高

final int width = bufImage.getWidth();

final int height = bufImage.getHeight();

// 读取出图片的所有像素

int[] rgbs = bufImage.getRGB(0, 0, width, height, null, 0, width);

// 对图片的像素矩阵进行水平镜像

for (int row = 0; row < height; row++) {

for (int col = 0; col < width / 2; col++) {

int temp = rgbs[row \* width + col];

rgbs[row \* width + col] = rgbs[row \* width + (width - 1 - col)];

rgbs[row \* width + (width - 1 - col)] = temp;

}

}

// 把水平镜像后的像素矩阵设置回 bufImage

bufImage.setRGB(0, 0, width, height, rgbs, 0, width);

// 把修改过的 bufImage 保存到本地

ImageIO.write(bufImage, "JPEG", new File("bb.jpg"));

}

}

查看结果，下面左图是变换前的图片aa.jpg，右边是变换后的图片bb.jpg: