


# 第20章

---

## AWT 绘图技术

(  视频讲解：33 分钟 )

要开发高级的应用程序，就必须适当掌握图像处理技术。它是程序开发不可缺少的技术，使用该技术可以为程序提供数据统计、图表分析等功能，提高程序的交互能力。本章将向读者介绍绘图技术的基本知识以及图像处理。

通过阅读本章，您可以：

- » 了解 Java 绘制图形
- » 了解 Java 绘图颜色与笔画属性
- » 掌握 Java 绘制文本
- » 掌握 Java 图片处理



## 20.1 绘制图形

 视频讲解：光盘\TM\1\20\绘制图形.exe

绘图是高级程序设计非常重要的技术，例如应用程序需要绘制闪屏图片、绘制背景图片、绘制组件外观，Web 程序可以绘制统计图、绘制数据库存储的图片资源等。正所谓“一图胜千言”，使用图片能够更好地表达程序运行结果、进行细致的数据分析与保存等。本节将介绍 Java 语言程序设计的绘图类 Graphics 与 Graphics2D。

### 20.1.1 Graphics

Graphics 类是所有图形上下文的抽象基类，它允许应用程序在组件以及闭屏图像上进行绘制。Graphics 类封装了 Java 支持的基本绘图操作所需的状态信息，主要包括颜色、字体、画笔、文本、图像等。

Graphics 类提供了绘图常用的方法，利用这些方法可以实现直线、矩形、多边形、椭圆、圆弧等形状和文本、图片的绘制操作。另外，在执行这些操作之前，还可以使用相应的方法，设置绘图的颜色、字体等状态属性。

Java 可以分别使用 Graphics 和 Graphics2D 绘制图形，Graphics 类使用不同的方法实现不同图形的绘制，例如 drawLine()方法用于绘制直线、drawRect()方法用于绘制矩形、drawOval()方法用于绘制椭圆图形等。

**【例 20.1】** 在项目中创建 DrawCircle 类，使该类继承 JFrame 类成为窗体组件，在类中创建继承 JPanel 类的 DrawPanel 内部类，并重写 paint()方法，实现绘制 5 个圆形组成的图案。（实例位置：光盘\TM\1\20\1）

```
package com.lzw;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class DrawCircle extends JFrame {
    private final int OVAL_WIDTH = 80;           //圆形的宽
    private final int OVAL_HEIGHT = 80;         //圆形的高
    public DrawCircle(){
        super();
        initialize();                             //调用初始化方法
    }
    //初始化方法
    private void initialize() {
        this.setSize(300, 200);                 //设置窗体大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗体关闭模式
        setContentPane(new DrawPanel());        //设置窗体面板为绘图面板对象
        this.setTitle("绘图实例");              //设置窗体标题
    }
}
```

```

    }
    //主方法
    public static void main(String[] args) {
        new DrawCircle ().setVisible(true);           //创建窗体
    }
    //创建绘图面板
    class DrawPanel extends JPanel {
        public void paint(Graphics g) {
            super.paint(g);
            g.drawOval(10, 10, OVAL_WIDTH, OVAL_HEIGHT); //绘制第 1 个圆形
            g.drawOval(80, 10, OVAL_WIDTH, OVAL_HEIGHT); //绘制第 2 个圆形
            g.drawOval(150, 10, OVAL_WIDTH, OVAL_HEIGHT); //绘制第 3 个圆形
            g.drawOval(50, 70, OVAL_WIDTH, OVAL_HEIGHT); //绘制第 4 个圆形
            g.drawOval(120, 70, OVAL_WIDTH, OVAL_HEIGHT); //绘制第 5 个圆形
        }
    }
}

```

运行结果如图 20.1 所示。

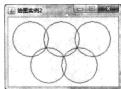


图 20.1 绘制圆形图的窗体

Graphics 类常用的图形绘制方法如表 20.1 所示。

表 20.1 Graphics 类常用的图形绘制方法

方 法	说 明	举 例	绘 图 效 果
<code>drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	弧形	<code>drawArc(100,100,100,50,270,200);</code>	
<code>drawLine(int x1, int y1, int x2, int y2)</code>	直线	<code>drawLine(10,10,50,10);</code> <code>drawLine(30,10,30,40);</code>	
<code>drawOval(int x, int y, int width, int height)</code>	椭圆	<code>drawOval(10,10,50,30);</code>	
<code>drawPolygon(int[] xPoints, int[] yPoints, int nPoints)</code>	多边形	<code>int[] xs={10,50,10,50};</code> <code>int[] ys={10,10,50,50};</code> <code>drawPolygon(xs, ys, 4);</code>	
<code>drawPolyline(int[] xPoints, int[] yPoints, int nPoints)</code>	多边形	<code>int[] xs={10,50,10,50};</code> <code>int[] ys={10,10,50,50};</code> <code>drawPolyline(xs, ys, 4);</code>	
<code>drawRect(int x, int y, int width, int height)</code>	矩形	<code>drawRect(10,10,100,50);</code>	
<code>drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	圆角矩形	<code>drawRoundRect(10, 10, 50, 30, 10, 10);</code>	

续表

方 法	说 明	举 例	绘 图 效 果
<code>fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	实心弧形	<code>fillArc(100,100,50,30,270,200);</code>	
<code>fillOval(int x, int y, int width, int height)</code>	实心椭圆	<code>fillOval(10,10,50,30);</code>	
<code>fillPolygon(int[] xPoints, int[] yPoints, int nPoints)</code>	实心多边形	<code>int[] xs={10,50,10,50}; int[] ys={10,10,50,50}; fillPolygon(xs, ys, 4);</code>	
<code>fillRect(int x, int y, int width, int height)</code>	实心矩形	<code>fillRect(10, 10, 50, 30);</code>	
<code>fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	实心圆角矩形	<code>g.fillRoundRect(10, 10, 50, 30,10,10);</code>	

## 20.1.2 Graphics2D

使用 `Graphics` 类可以完成简单的图形绘制任务，但是它所实现的功能非常有限，如无法改变线条的粗细、不能对图片使用旋转和模糊等过滤效果。

`Graphics2D` 继承 `Graphics` 类，实现了功能更加强大的绘图操作的集合。由于 `Graphics2D` 类是 `Graphics` 类的扩展，也是推荐使用的 Java 绘图类，所以本章主要介绍如何使用 `Graphics2D` 类实现 Java 绘图。

 说明

`Graphics2D` 是推荐使用的绘图类，但是程序设计中提供的绘图对象大多是 `Graphics` 类的实例对象，这时应该使用强制类型转换将其转换为 `Graphics2D` 类型。例如：

```
public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;           //强制类型转换为 Graphics2D 类型
    g2.....
}
```

`Graphics2D` 是继承 `Graphics` 类编写的，它包含了 `Graphics` 类的绘图方法并添加了更强的功能，是推荐使用的绘图类。`Graphics2D` 可以分别使用不同的类来表示不同的形状，如 `Line2D`、`Rectangle2D` 等。

要绘制指定形状的图形，需要先创建并初始化该图形类的对象，这些图形类必须是 `Shape` 接口的实现类，然后使用 `Graphics2D` 类的 `draw()` 方法绘制该图形对象或者使用 `fill()` 方法填充该图形对象。语格式如下：

```
draw(Shape form)
```

或者

```
fill(Shape form)
```

其中，`form` 表示实现 `Shape` 接口的对象。

java.awt.geom 包中提供了如下一些常用的图形类，这些图形类都实现了 Shape 接口。

- ☒ Arc2D
- ☒ CubicCurve2D
- ☒ Ellipse2D
- ☒ Line2D
- ☒ Point2D
- ☒ QuadCurve2D
- ☒ Rectangle2D
- ☒ RoundRectangle2D



### 注意

各图形类都是抽象类型的。在不同图形类中有 Double 和 Float 两个实现类，这两个实现类以不同精度构建图形对象。为方便计算，在程序开发中经常使用 Double 类的实例对象进行图形绘制，但是如果程序中要使用成千上万个图形，则建议使用 Float 类的实例对象进行绘制，这样会节省内存空间。

**【例 20.2】** 在窗体的实现类中创建图形类的对象，然后使用 Graphics2D 类绘制和填充这些图形。  
(实例位置：光盘\TM\sl\20\2)

```
package com.lzw;
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class DrawFrame extends JFrame {
    public DrawFrame() {
        super();
        initialize(); //调用初始化方法
    }
    //初始化方法
    private void initialize() {
        this.setSize(300, 200); //设置窗体大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗体关闭模式
        add(new CanvasPanel()); //设置窗体面板为绘图面板对象
        this.setTitle("绘图实例 2"); //设置窗体标题
    }
    public static void main(String[] args) {
        new DrawFrame().setVisible(true);
    }
    class CanvasPanel extends JPanel {
        public void paint(Graphics g) {
            super.paint(g);
            Graphics2D g2 = (Graphics2D) g;
            Shape[] shapes = new Shape[4]; //声明图形数组
            shapes[0] = new Ellipse2D.Double(5, 5, 100, 100); //创建圆形对象
            shapes[1] = new Rectangle2D.Double(110, 5, 100, 100); //创建矩形对象
            shapes[2] = new Rectangle2D.Double(15, 15, 80, 80); //创建圆形对象
        }
    }
}
```

```

        shapes[3] = new Ellipse2D.Double(120, 15, 80, 80); //创建矩形对象
        for (Shape shape : shapes) { //遍历图形数组
            Rectangle2D bounds = shape.getBounds2D();
            if (bounds.getWidth() == 80)
                g2.fill(shape); //填充图形
            else
                g2.draw(shape); //绘制图形
        }
    }
}

```

运行结果如图 20.2 所示。

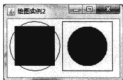


图 20.2 运行结果

### 20.1.3 范例 1：绘制指定角度的填充扇形

在 Java 中绘制指定角度的填充扇形，编写 Java 程序并运行，在窗体上将输出填充扇形。运行结果如图 20.3 所示。（实例位置：光盘\TM\sl\20\3）

（1）在项目中创建一个继承 JFrame 类的 DrawSectorFrame 窗体类。

（2）在 DrawSectorFrame 窗体类中创建内部面板类 DrawSectorPanel，并重写 JComponent 类的 paint() 方法，在该方法中使用 Graphics 类的 fillArc() 方法绘制填充扇形。

（3）将内部面板类 DrawSectorPanel 的实例添加到窗体类 DrawSectorFrame 的内容面板上，用于在窗体上显示绘制的填充扇形。代码如下：



图 20.3 绘制指定角度的填充扇形

```

class DrawSectorPanel extends JPanel { //创建内部面板类
    public void paint(Graphics g) { //重写 paint()方法
        g.fillArc(40, 20, 80, 80, 0, 150); //绘制填充扇形
        g.fillArc(140, 20, 80, 80, 180, -150); //绘制填充扇形
        g.fillArc(40, 40, 80, 80, 0, -110); //绘制填充扇形
        g.fillArc(140, 40, 80, 80, 180, 110); //绘制填充扇形
    }
}

```

### 20.1.4 范例 2：绘制多边形

在 Java 中绘制多边形，编写 Java 程序并运行，在窗体上将输出多边形。运行结果如图 20.4 所示。（实例位置：光盘\TM\sl\20\4）

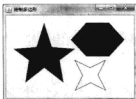


图 20.4 绘制多边形

(1) 在项目中创建一个继承 JFrame 类的 DrawPolygonFrame 窗体类。

(2) 在 DrawPolygonFrame 窗体类中创建内部面板类 DrawPolygonPanel, 并重写 JComponent 类的 paint() 方法, 在该方法中使用 Graphics 类的 drawPolygon() 和 fillPolygon() 方法绘制多边形。

(3) 将内部面板类 DrawPolygonPanel 的实例添加到窗体类 DrawPolygonFrame 的内容面板上, 用于在窗体上显示绘制的多边形。代码如下:

```
class DrawPolygonPanel extends JPanel {
    public void paint(Graphics g) {
        int[] x1 = { 100, 120, 180, 140, 150, 100, 50, 60, 20, 80 };
        int[] y1 = { 20, 85, 90, 120, 180, 140, 180, 120, 90, 85 };
        int n1 = 10;
        g.fillPolygon(x1, y1, n1);
        int[] x2 = { 210, 270, 310, 270, 210, 170 };
        int[] y2 = { 20, 20, 65, 110, 110, 65 };
        int n2 = 6;
        g.fillPolygon(x2, y2, n2);
        int[] x3 = { 180, 220, 260, 240, 260, 220, 180, 200 };
        int[] y3 = { 120, 140, 120, 160, 200, 180, 200, 160 };
        int n3 = 8;
        g.drawPolygon(x3, y3, n3);
    }
}
```

//创建内部面板类  
//重写 paint() 方法  
//多边形的横坐标  
//多边形的纵坐标  
//多边形的边数  
//绘制多边形  
//多边形的横坐标  
//多边形的纵坐标  
//多边形的边数  
//绘制实心多边形  
//多边形的横坐标  
//多边形的纵坐标  
//多边形的边数  
//绘制多边形

## 20.2 绘图颜色与笔画属性

 视频讲解: 光盘\TM\lx\20\绘图颜色与笔画属性.exe

Java 语言使用 Color 类封装颜色的各种属性, 并对颜色进行管理。另外, 在绘制图形时还可以指定线的粗细、虚线还是实线等笔画属性。

### 20.2.1 设置颜色

使用 Color 类可以创建任何颜色的对象, 而不用担心不同平台对该颜色支持与否, 因为 Java 以跨

平台和与硬件无关的方式支持颜色管理。创建 Color 对象的构造方法如下:

```
Color col = new Color(int r, int g, int b)
```

或者

```
Color col = new Color(int rgb)
```

- ☑ rgb: 颜色值, 该值是红、绿、蓝三原色的总和。
- ☑ r: 该参数是三原色中红色的取值。
- ☑ g: 该参数是三原色中绿色的取值。
- ☑ b: 该参数是三原色中蓝色的取值。

Color 类定义了常用色彩的常量值, 如表 20.2 所示。这些常量都是静态的 Color 对象, 可以直接使用这些常量值定义颜色对象。

表 20.2 常用的 Color 常量

常 量 名	颜 色 值
Color BLACK	黑色
Color BLUE	蓝色
Color CYAN	青色
Color DARK_GRAY	深灰色
Color GRAY	灰色
Color GREEN	绿色
Color LIGHT_GRAY	浅灰色
Color MAGENTA	洋红色
Color ORANGE	桔黄色
Color PINK	粉红色
Color RED	红色
Color WHITE	白色
Color YELLOW	黄色

绘图类可以使用 setColor() 方法设置当前颜色, 语法格式如下:

```
setColor(Color color);
```

color: Color 对象, 代表一个颜色值, 如红色、黄色或者默认的黑色。

**【例 20.3】** 设置当前绘图颜色为红色。

```
public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2 = (Graphics2D) g;
    g.setColor(Color.RED);
    .....
}
```



**说明**

设置绘图颜色以后, 再进行绘图或者绘制文本, 都会采用该颜色作为前景色; 如果想再绘制其他颜色的图形或文本, 需要在此调用 `setColor()` 方法设置其他颜色。

## 20.2.2 笔画属性

在默认情况下, `Graphics` 绘图类使用的笔画属性是粗细为 1 个像素的正方形, 而 `Java2D` 的 `Graphics2D` 类可以调用 `setStroke()` 方法设置笔画的属性, 例如改变线条的粗细、使用实线还是虚线、定义线段端点的形状和风格等, 语法格式如下:

```
setStroke(Stroke stroke)
```

`stroke`: `Stroke` 接口的实现类。

`setStroke()` 方法必须接受一个 `Stroke` 接口的实现类作参数, `java.awt` 包中提供了 `BasicStroke` 类, 它实现了 `Stroke` 接口, 并且通过不同的构造方法创建笔画属性不同的对象, 这些构造方法如下:

- ☑ `BasicStroke()`
- ☑ `BasicStroke(float width)`
- ☑ `BasicStroke(float width, int cap, int join)`
- ☑ `BasicStroke(float width, int cap, int join, float miterlimit)`
- ☑ `BasicStroke(float width, int cap, int join, float miterlimit, float[] dash, float dash_phase)`

这些构造方法中的参数说明如表 20.3 所示。

表 20.3 参数说明

参 数 名	说 明
<code>width</code>	笔画宽度, 此宽度必须大于或等于 0.0f。如果将宽度设置为 0.0f, 则将笔画设置为当前设备的默认宽度
<code>cap</code>	线段端点的装饰
<code>join</code>	应用在路径线段交汇处的装饰
<code>miterlimit</code>	斜接处的剪裁限制。该参数值必须大于或等于 1.0f
<code>dash</code>	表示虚线模式的数组
<code>dash_phase</code>	开始虚线模式的偏移量

`cap` 参数可以使用 `CAP_BUTT`、`CAP_ROUND` 和 `CAP_SQUARE` 常量, 这 3 个常量对线段端点的装饰效果如图 20.5 所示。

`join` 参数用于修饰线段交汇效果, 可以使用 `JOIN_BEVEL`、`JOIN_MITER` 和 `JOIN_ROUND` 常量, 这 3 个常量对线段交汇的修饰效果如图 20.6 所示。



图 20.5 cap 参数对线段端点的装饰效果



图 20.6 join 参数修饰线段交汇的效果

### 20.2.3 范例 3：为图形填充渐变色

在 Java 中绘制图形时，为图形填充渐变色，编写 Java 程序并运行，在窗体上将输出图形渐变色。运行结果如图 20.7 所示。（实例位置：光盘\TM\sl\20\5）

（1）在项目中创建一个继承 JFrame 类的 FillGradientFrame 窗体类。

（2）在 FillGradientFrame 窗体类中创建内部面板类 FillGradientPanel，并重写 JComponent 类的 paint() 方法，在该方法中使用 Graphics2D 类的 setPaint() 方法设置封装了渐变色的对象，该对象是通过 GradientPaint 类创建的。

（3）将内部面板类 FillGradientPanel 的实例添加到窗体类 FillGradientFrame 的内容面板上，用于在窗体上显示填充了渐变颜色后的图形。代码如下：

```
class FillGradientPanel extends JPanel {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        //创建矩形对象
        Rectangle2D.Float rect = new Rectangle2D.Float(20, 20, 280, 140);
        //创建循环渐变的 GradientPaint 对象
        GradientPaint paint = new GradientPaint(20,20,Color.BLUE,100,80,Color.RED,true);
        g2.setPaint(paint);
        g2.fill(rect);
    }
}
```

//创建内部面板类  
//重写 paint() 方法  
//获得 Graphics2D 对象  
  
  
  
  
//设置渐变  
//绘制矩形



图 20.7 为图形填充渐变色的效果

### 20.2.4 范例 4：设置笔画的粗细

在 Java 中绘制图形时，设置笔画的粗细，编写 Java 程序并运行，在窗体上将输出不同粗细的笔画。运行结果如图 20.8 所示。（实例位置：光盘\TM\sl\20\6）

（1）在项目中创建一个继承 JFrame 类的 StrokeWidthFrame 窗体类。

（2）在 StrokeWidthFrame 窗体类中创建内部面板类 ChangeStrokeWidthPanel，并重写 JComponent 类的 paint() 方法，在该方法中使用 BasicStroke 类创建笔画对象，并使用 Graphics2D 类的 setStroke() 方法设置笔画的粗细。

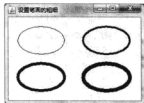


图 20.8 设置笔画粗细的效果

(3) 将内部面板类 `ChangeStrokeWidthPanel` 的实例添加到窗体类 `StrokeWidthFrame` 的内容面板上, 用于在窗体上显示设置笔画粗细后的图形。代码如下:

```
class ChangeStrokeWidthPanel extends JPanel {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        BasicStroke stroke = new BasicStroke(1);
        g2.setStroke(stroke);
        Ellipse2D.Float ellipse = new Ellipse2D.Float(20,20,100,60);
        g2.draw(ellipse);
        stroke = new BasicStroke(4);
        g2.setStroke(stroke);
        ellipse = new Ellipse2D.Float(160,20,100,60);
        g2.draw(ellipse);
        stroke = new BasicStroke(6);
        g2.setStroke(stroke);
        ellipse = new Ellipse2D.Float(20,100,100,60);
        g2.draw(ellipse);
        stroke = new BasicStroke(8);
        g2.setStroke(stroke);
        ellipse = new Ellipse2D.Float(160,100,100,60);
        g2.draw(ellipse);
    }
}
```

//创建内部面板类  
//重写 paint()方法  
//获得 Graphics2D 对象  
//创建宽度为 1 的笔画对象  
//设置笔画对象  
//创建椭圆对象  
//绘制椭圆  
//创建宽度为 4 的笔画对象  
//设置笔画对象  
//创建椭圆对象  
//绘制椭圆  
//创建宽度为 6 的笔画对象  
//设置笔画对象  
//创建椭圆对象  
//绘制椭圆  
//创建宽度为 8 的笔画对象  
//设置笔画对象  
//创建椭圆对象  
//绘制椭圆

## 20.3 绘制文本

 视频讲解: 光盘\TM\lx\20\绘制文本.exe

Java 绘图类也可以绘制文本内容, 在绘制文本之前可以设置使用的字体、大小等。本节将介绍如何绘制文本以及设置文本的字体。

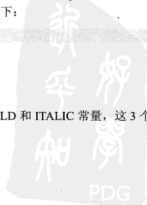
### 20.3.1 设置字体

Java 使用 `Font` 类封装了字体的大小、样式等属性, 该类在 `java.awt` 包中定义, 其构造方法可以指定字体的名称、大小和样式 3 个属性。语法格式如下:

```
Font(String name, int style, int size)
```

- ☒ name: 字体的名称。
- ☒ style: 字体的样式。
- ☒ size: 字体的大小。

其中字体样式可以使用 `Font` 类的 `PLAIN`、`BOLD` 和 `ITALIC` 常量, 这 3 个字体样式常量的效果如图 20.9 所示。



普通样式	PLAIN
粗体样式	BOLD
斜体样式	ITALIC
斜体粗合斜体样式	ITALIC BOLD

图 20.9 字体样式

设置绘图类的字体可以使用绘图类的 `setFont()` 方法。设置字体以后在图形上下文中绘制的所有文字都使用该字体，除非再次设置其他字体。语法格式如下：

```
setFont(Font font)
```

font: Font 类的字体对象。

## 20.3.2 显示文字

Graphics2D 类提供了 `drawString()` 方法，使用该方法可以实现图形上下文的文本绘制，从而实现在图片上显示文字的功能。语法格式如下：

```
drawString(String str, int x, int y);
```

或者

```
drawString(String str, float x, float y)
```

- ☒ str: 要绘制的文本字符串。
- ☒ x: 绘制字符串的水平起始位置。
- ☒ y: 绘制字符串的垂直起始位置。

这两个方法唯一不同的就是方法使用的 `x` 和 `y` 参数的类型不同。

**【例 20.4】** 绘制一个矩形图，在矩形图的中间显示文本，文本的内容是当前时间。（实例位置：光盘\TM\sl\20\7）

```
package com.lzw;
import java.awt.*;
import java.awt.geom.Rectangle2D;
import java.util.Date;
import javax.swing.JFrame;

public class DrawString extends JFrame {
    private Shape rect;           //矩形对象
    private Font font;           //字体对象
    private Date date;           //当前日期对象

    public DrawString() {
        rect = new Rectangle2D.Double(10,10,200,80);
        font = new Font("宋体",Font.BOLD,16);
        date = new Date();
        this.setSize(230, 140);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

//设置窗体大小  
//设置窗体关闭模式

```

        add(new CanvasPanel());                                //设置窗体面板为绘图面板对象
        this.setTitle("绘图文本");                            //设置窗体标题
    }
    public static void main(String[] args) {
        new DrawString().setVisible(true);
    }
    class CanvasPanel extends Canvas {
        public void paint(Graphics g) {
            super.paint(g);
            Graphics2D g2 = (Graphics2D) g;
            g2.setColor(Color.CYAN);                          //设置当前绘图颜色
            g2.fill(rect);                                       //填充矩形
            g2.setColor(Color.BLUE);                            //设置当前绘图颜色
            g2.setFont(font);                                    //设置字体
            g2.drawString("现在是", 20, 30);                    //绘制文本
            g2.drawString(String.format("%t", date), 50, 60);  //绘制时间文本
        }
    }
}

```

运行结果如图 20.10 所示。

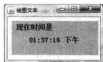


图 20.10 在窗体中绘制文本

### 20.3.3 范例 5: 设置文本的字体

在 Java 中绘制文本时, 如何设置文本的字体, 其中包括字体名称、字体大小和字体样式。编写 Java 程序并运行, 运行结果如图 20.11 所示。(实例位置: 光盘\TM\sl\20\8)

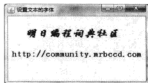


图 20.11 设置文本字体的效果

- (1) 在项目中创建一个继承 JFrame 类的 TextFontFrame 窗体类。
- (2) 在 TextFontFrame 窗体类中创建内部面板类 ChangeTextFontPanel, 并重写 JComponent 类的 paint() 方法, 在该方法中使用 Font 类创建字体对象, 并使用 Graphics 类的 setFont() 方法设置文本的字体。

(3) 将内部面板类 ChangeTextFontPanel 的实例添加到窗体类 TextFontFrame 的内容面板上, 用于在窗体上显示指定字体后的文本。代码如下:

```

class ChangeTextFontPanel extends JPanel {
    public void paint(Graphics g) {
        String value = "明日编程词典社区";
        int x = 40;
        int y = 50;
        Font font = new Font("华文行楷", Font.BOLD + Font.ITALIC, 26); //创建字体对象
        g.setFont(font);
        g.drawString(value, x, y);
    }
}

```

//创建内部面板类  
 //重写 paint() 方法  
 //文本位置的横坐标  
 //文本位置的纵坐标  
 //创建字体对象  
 //设置字体  
 //绘制文本

```

value = "http://community.mrbccod.com";
x = 10;
y = 100;
font = new Font("宋体", Font.BOLD, 20);
g.setFont(font);
g.drawString(value, x, y);
    }
}

```

//文本位置的横坐标  
//文本位置的纵坐标  
//创建字体对象  
//设置字体  
//绘制文本

## 20.3.4 范例 6：设置文本的图形和颜色

本范例演示在 Java 中绘制文本和图形时，如何设置文本和图形的颜色。运行结果如图 20.12 所示。（实例位置：光盘\TM\sl\20\9）

（1）在项目中创建一个继承 JFrame 类的 TextAndShapeColorFrame 窗体类。

（2）在 TextAndShapeColorFrame 窗体类中创建内部面板类 TextAndShapeColorPanel，并重写 JComponent 类的 paint()方法，在该方法中使用 Color 类创建颜色对象，并使用 Graphics 类的 setColor()方法设置文本和图形的颜色。

（3）将内部面板类 TextAndShapeColorPanel 的实例添加到窗体类 TextAndShapeColorFrame 的内部面板上，用于在窗体上显示设置颜色后的文本和图形。代码如下：



图 20.12 设置文本和图形颜色的效果

```

class TextAndShapeColorPanel extends JPanel {
    public void paint(Graphics g) {
        String value = "只要努力——";
        int x = 60;
        int y = 60;
        Color color = new Color(255,0,0);
        g.setColor(color);
        g.drawString(value, x, y);
        value = "一切皆有可能";
        x = 140;
        y = 100;
        color = new Color(0,0,255);
        g.setColor(color);
        g.drawString(value, x, y);
        color = Color.ORANGE;
        g.setColor(color);
        g.drawRoundRect(40,30,200,100,40,30);
        g.drawRoundRect(45,35,190,90,36,26);
    }
}

```

//创建内部面板类  
//重写 paint()方法  
//文本位置的横坐标  
//文本位置的纵坐标  
//创建颜色对象  
//设置颜色  
//绘制文本  
//文本位置的横坐标  
//文本位置的纵坐标  
//创建颜色对象  
//设置颜色  
//绘制文本  
//通过 Color 类的字段获得颜色对象  
//设置颜色  
//绘制圆角矩形  
//绘制圆角矩形

## 20.4 图片处理

 视频讲解：光盘\TM\lx\20\图片处理.exe

### 20.4.1 绘制图片

绘图类不仅可以绘制图形和文本，还可以使用 `drawImage()` 方法将图片资源显示到绘图上下文中，而且可以实现各种特效处理，如图片的缩放、翻转等。本节主要介绍如何显示图片，语法格式如下：

```
drawImage(Image img, int x, int y, ImageObserver observer)
```

该方法将 `img` 图片显示在 `x`、`y` 指定的位置上，方法中涉及的参数说明如表 20.4 所示。

表 20.4 参数说明

参 数 名	说 明
<code>img</code>	要显示的图片对象
<code>x</code>	水平位置
<code>y</code>	垂直位置
<code>observer</code>	要通知的图像观察者

该方法的使用与绘制文本的 `drawString()` 方法类似，唯一不同的是 `drawImage()` 方法需要指定通知的图像观察者。

**【例 20.5】** 在整个窗体中显示图片，图片的大小保持不变。（实例位置：光盘\TM\sl\20\10）

```
package com.lzw;
import java.awt.*;
import java.net.URL;
import javax.swing.JFrame;
public class DrawImage extends JFrame {
    Image img;
    public DrawImage() {
        URL imgUrl = DrawImage.class.getResource("img.jpg"); //获取图片资源的路径
        img=Toolkit.getDefaultToolkit().getImage(imgUrl); //获取图片资源
        this.setSize(440, 300); //设置窗体大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗体关闭模式
        add(new CanvasPanel()); //设置窗体面板为绘图面板对象
        this.setTitle("绘制图片"); //设置窗体标题
    }
    public static void main(String[] args) {
        new DrawImage().setVisible(true);
    }
}
class CanvasPanel extends Canvas {
```

```

        public void paint(Graphics g) {
            super.paint(g);
            Graphics2D g2 = (Graphics2D) g;
            g2.drawImage(img, 0, 0, this);
        }
    }
}
//显示图片

```

运行结果如图 20.13 所示。

开发高级的桌面应用程序，必须掌握一些图像处理与动画制作的技术，例如在程序中显示统计图、销售趋势图、动态按钮等。本节将在 Java 绘图的基础上讲解图像处理技术。



图 20.13 显示图片的窗体

## 20.4.2 放大与缩小

在讲解绘制图片时，使用了 `drawImage()` 方法将图片以原始大小显示在窗体中，想要实现图片的放大与缩小则需要使用它的重载方法。语法格式如下：

```
drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)
```

该方法将 `img` 图片显示在 `x`、`y` 指定的位置上，并指定图片的宽度和高度属性。方法中涉及的参数说明如表 20.5 所示。

表 20.5 参数说明

参 数 名	说 明
<code>img</code>	要显示的图片对象
<code>x</code>	水平位置
<code>y</code>	垂直位置
<code>width</code>	图片的新宽度属性
<code>height</code>	图片的新高度属性
<code>observer</code>	要通知的图像观察者

**【例 20.6】** 在窗体中显示原始大小的图片，然后通过两个按钮的单击事件，分别显示该图片缩小与放大后的效果。（实例位置：光盘\TM\sl\20\11）

```

package com.lzw;
import java.awt.*;
import java.net.*;
import javax.swing.*;
public class ImageZoom extends JFrame {
    Image img;
    ...//省略部分成员变量
    public ImageZoom() {
        initialize();
    }
}
//调用初始化方法

```



```

    }
    //界面初始化方法
    private void Initialize() {
        URL imgUrl = ImageZoom.class.getResource("img.jpg");           //获取图片资源的路径
        img = Toolkit.getDefaultToolkit().getImage(imgUrl);             //获取图片资源
        canvas = new MyCanvas();
        this.setBounds(100, 100, 800, 600);                             //设置窗口大小和位置
        this.setContentPane(getContentPane());                         //设置内容面板
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                 //设置窗体关闭模式
        this.setTitle("绘制图片");                                       //设置窗体标题
    }
    ...//省略布局方法的代码
    //获取滑块组件
    private JSlider getJSlider() {
        if (jSlider == null) {
            jSlider = new JSlider();                                     //创建滑块组件
            jSlider.setMaximum(1000);                                   //设置滑块最大取值
            jSlider.setValue(100);                                       //设置滑块最小取值
            jSlider.setMinimum(1);                                       //设置滑块当前值
            //添加滑块改变事件
            jSlider.addChangeListener(new javax.swing.event.ChangeListener() {
                public void stateChanged(javax.swing.event.ChangeEvent e) {
                    canvas.repaint();                                    //重新绘制画板内容
                }
            });
        }
        return jSlider;
    }
    //主方法
    public static void main(String[] args) {
        new ImageZoom().setVisible(true);
    }
    //画板类
    class MyCanvas extends Canvas {
        public void paint(Graphics g) {
            int newW = 0, newH = 0;
            imgWidth = img.getWidth(this);                             //获取图片宽度
            imgHeight = img.getHeight(this);                            //获取图片高度
            float value = jSlider.getValue();                           //滑块组件的取值
            newW = (int) (imgWidth * value / 100);                     //计算图片放大后的宽度
            newH = (int) (imgHeight * value / 100);                     //计算图片放大后的高度
            g.drawImage(img, 0, 0, newW, newH, this);                   //绘制指定大小的图片
        }
    }
}

```

运行结果如图 20.14 所示。

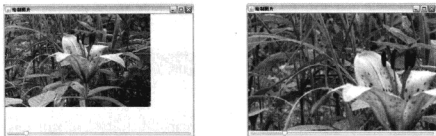


图 20.14 图像缩放效果

## 说明

repaint()方法将调用 paint()方法，实现组件或画板的重画功能，类似于界面刷新。

### 20.4.3 图片翻转

图像的翻转需要使用 drawImage()方法的另一个重载方法。语法格式如下：

```
drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)
```

此方法总是用非缩放的图像来呈现缩放的矩形，并且动态地执行所需的缩放。此操作不使用缓存的缩放图像。执行图像从源到目标的缩放：源矩形的第一个坐标被映射到目标矩形的第一个坐标，第二个源坐标被映射到第二个目标坐标。按需要缩放和翻转子图像以保持这些映射关系。方法中涉及的参数说明如表 20.6 所示。

表 20.6 参数说明

参 数 名	说 明
img	要绘制的指定图像
dx1	目标矩形第一个坐标的 x 位置
dy1	目标矩形第一个坐标的 y 位置
dx2	目标矩形第二个坐标的 x 位置
dy2	目标矩形第二个坐标的 y 位置
sx1	源矩形第一个坐标的 x 位置
sy1	源矩形第一个坐标的 y 位置
sx2	源矩形第二个坐标的 x 位置
sy2	源矩形第二个坐标的 y 位置
observer	要通知的图像观察者

【例 20.7】 在窗体界面中绘制图像的翻转效果。程序中 drawImage()方法使用的参数名称与语法中介绍的相同，MyCanvas 类只是在 paint()方法中按照参数顺序执行 drawImage()方法，图片的翻转由控制按钮变换参数值，然后执行 MyCanvas 类的 repaint()方法实现。（实例位置：光盘\TM\sl\20\12）

```

package com.lzw;
import java.awt.*;
import java.net.URL;
import javax.swing.*;

public class PartImage extends JFrame {
    private Image img;
    private int dx1,dy1,dx2,dy2;
    private int sx1,sy1,sx2,sy2;
    ...//省略部分代码
    private MyCanvas canvasPanel = null;
    public PartImage() {
        dx2=sx2=300;           //初始化图像大小
        dy2=sy2=200;           //调用初始化方法
        initialize();
    }
    ...//省略部分代码
    //获取“水平翻转”按钮
    private JButton getJButton() {
        if (JButton == null) {
            JButton = new JButton();
            JButton.setText("水平翻转");
            JButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    sx1=Math.abs(sx1-300);           //改变源矩形两个坐标的 x 位置
                    sx2=Math.abs(sx2-300);
                    canvasPanel.repaint();
                }
            });
        }
        return JButton;
    }
    //获取“垂直翻转”按钮
    private JButton getJButton1() {
        if (JButton1 == null) {
            JButton1 = new JButton();
            JButton1.setText("垂直翻转");
            JButton1.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    sy1=Math.abs(sy1-200);           //改变源矩形两个坐标的 y 位置
                    sy2=Math.abs(sy2-200);
                    canvasPanel.repaint();
                }
            });
        }
        return JButton1;
    }
    ...//省略部分代码
    //画板
    class MyCanvas extends JPanel {
        public void paint(Graphics g) {

```

```

        g.drawImage(img, dx1, dy1, dx2, dy2, sx1, sy1, sx2, sy2, this); //绘制指定大小的图片
    }
}

```

运行结果如图 20.15 所示。



图 20.15 源图、水平翻转和垂直翻转的效果

## 20.4.4 图片旋转

图像的旋转需要调用 Graphics2D 类的 rotate() 方法, 该方法将根据指定的弧度旋转图像。语法格式如下:

```
rotate(double theta)
```

theta: 旋转的弧度。

**说明**

该方法只接受旋转的弧度作参数, 可以使用 Math 类的 toRadians() 方法将角度转换为弧度。toRadians() 方法接受角度值作参数, 返回值是转换完毕的弧度值。

**【例 20.8】** 在主窗体中绘制 3 个旋转后的图像, 每个图像旋转角度值为 5。(实例位置: 光盘\TM\sl\20\13)

```

package com.lzw;
import java.awt.*;
import java.net.URL;
import javax.swing.*;

public class RotatImage extends JFrame {
    private Image img;
    private MyCanvas canvasPanel = null;
    public RotatImage() {
        initialize(); //调用初始化方法
    }
    //界面初始化方法
    private void initialize() {
        URL imgUrl = RotatImage.class.getResource("cow.jpg"); //获取图片资源的路径
        img = Toolkit.getDefaultToolkit().getImage(imgUrl); //获取图片资源
    }
}

```

```

        canvasPanel = new MyCanvas();
        this.setBounds(100, 100, 400, 350);
        add(canvasPanel);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("图片旋转");
    }
    //主方法
    public static void main(String[] args) {
        new RotateImage().setVisible(true);
    }
    //画板
    class MyCanvas extends JPanel {
        public void paint(Graphics g) {
            Graphics2D g2=(Graphics2D) g;
            g2.rotate(Math.toRadians(5));
            g2.drawImage(img, 70, 10, 300, 200, this);
            g2.rotate(Math.toRadians(5));
            g2.drawImage(img, 70, 10, 300, 200, this);
            g2.rotate(Math.toRadians(5));
            g2.drawImage(img, 70, 10, 300, 200, this);
        }
    }
}

```

//设置窗体大小和位置  
//设置窗体关闭模式  
//设置窗体标题  
  
//绘制指定大小的图片  
//绘制指定大小的图片  
//绘制指定大小的图片

运行结果如图 20.16 所示。



图 20.16 图像旋转效果

## 20.4.5 图片倾斜

可以使用 Graphics2D 类提供的 shear()方法设置绘图的倾斜方向，从而实现使图像倾斜的效果。语法格式如下：

```
shear(double shx, double shy)
```

- ☒ shx: 水平方向的倾斜量。
- ☒ shy: 垂直方向的倾斜量。

**【例 20.9】** 在窗体上绘制图像，使图像在水平方向实现倾斜效果。（实例位置：光盘\TM\sl\20\14）

```
package com.lzw;
import java.awt.*;
import java.net.URL;
import javax.swing.*;

public class TiltImage extends JFrame {
    private Image img;
    private MyCanvas canvasPanel = null;
    public TiltImage() {
        initialize(); //调用初始化方法
    }
    //界面初始化方法
    private void initialize() {
        URL imgUrl = TiltImage.class.getResource("cow.jpg"); //获取图片资源的路径
        img = Toolkit.getDefaultToolkit().getImage(imgUrl); //获取图片资源
        canvasPanel = new MyCanvas();
        this.setBounds(100, 100, 400, 300); //设置窗体大小和位置
        add(canvasPanel);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗体关闭模式
        this.setTitle("图片倾斜"); //设置窗体标题
    }
    //主方法
    public static void main(String[] args) {
        new TiltImage().setVisible(true);
    }
    //画板
    class MyCanvas extends JPanel {
        public void paint(Graphics g) {
            Graphics2D g2=(Graphics2D) g;
            g2.shear(0.3, 0);
            g2.drawImage(img, 0, 0, 300, 200, this); //绘制指定大小的图片
        }
    }
}
```

运行结果如图 20.17 所示。



图 20.17 水平倾斜的图片效果

### 20.4.6 范例 7：图形的交运算

在 Java 中如何实现图形的交运算，即保留两个图形的交集。编写 Java 程序并运行，将在窗体上显示进行交运算后的图形。运行结果如图 20.18 所示。（实例位置：

光盘\TM\sl\20\15）

（1）在项目中创建一个继承 JFrame 类的 IntersectOperationFrame 窗体类。

（2）在 IntersectOperationFrame 窗体类中创建内部面板类 IntersectOperationPanel，并重写 JComponent 类的 paint() 方法，在该方法中实现图形的交运算。

（3）将内部面板类 IntersectOperationPanel 的实例添加到窗体类 IntersectOperationFrame 的内容面板上，用于在窗体上显示图形进行交运算后的效果。代码如下：

```
class IntersectOperationPanel extends JPanel {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        Rectangle2D.Float rect = new Rectangle2D.Float(30, 30, 160, 120);
        Ellipse2D.Float ellipse = new Ellipse2D.Float(20, 30, 180, 180);
        Area area1 = new Area(rect);
        Area area2 = new Area(ellipse);
        area1.intersect(area2);
        g2.draw(area1);
        Ellipse2D.Float ellipse1 = new Ellipse2D.Float(190, 20, 100, 140);
        Ellipse2D.Float ellipse2 = new Ellipse2D.Float(240, 20, 100, 140);
        Area area3 = new Area(ellipse1);
        Area area4 = new Area(ellipse2);
        area3.intersect(area4);
        g2.fill(area3);
    }
}
```

//创建内部面板类  
//重写 paint()方法  
//获得 Graphics2D 对象  
//创建矩形对象  
//创建圆对象  
//创建区域矩形  
//创建区域圆  
//两个区域图形交运算  
//绘制交运算的区域图形  
//创建椭圆对象  
//创建区域椭圆  
//两个区域椭圆交运算  
//绘制交运算的区域椭圆

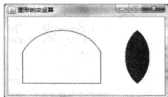


图 20.18 图形进行交运算的效果

### 20.4.7 范例 8：图形的异或运算

在 Java 中如何实现图形的异或运算，即两个图形去除交集后剩下的部分。编写 Java 程序并运行，将在窗体上显示进行异或运算后的图形。运行结果如图 20.19 所示。（实例位置：光盘\TM\sl\20\16）

（1）在项目中创建一个继承 JFrame 类的 ExclusiveOrOperationFrame 窗体类。

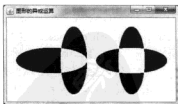


图 20.19 图形进行异或运算的效果

(2) 在 ExclusiveOrOperationFrame 窗体类中创建内部面板类 ExclusiveOrOperationPanel, 并重写 JComponent 类的 paint() 方法, 在该方法中实现图形的异或运算。

(3) 将内部面板类 ExclusiveOrOperationPanel 的实例添加到窗体类 ExclusiveOrOperationFrame 的内容面板上, 用于在窗体上显示图形进行异或运算后的效果。代码如下:

```
class ExclusiveOrOperationPanel extends JPanel {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        Ellipse2D.Float ellipse1 = new Ellipse2D.Float(20, 70, 160, 60);
        Ellipse2D.Float ellipse2 = new Ellipse2D.Float(120, 20, 60, 160);
        Area area1 = new Area(ellipse1);
        Area area2 = new Area(ellipse2);
        area1.exclusiveOr(area2);
        g2.fill(area1);
        Ellipse2D.Float ellipse3 = new Ellipse2D.Float(200, 70, 160, 60);
        Ellipse2D.Float ellipse4 = new Ellipse2D.Float(250, 20, 60, 160);
        Area area3 = new Area(ellipse3);
        Area area4 = new Area(ellipse4);
        area3.exclusiveOr(area4);
        g2.fill(area3);
    }
}
```

//创建内部面板类  
//重写 paint()方法  
//获得 Graphics2D 对象  
//创建椭圆对象  
//创建椭圆对象  
//创建区域椭圆  
//创建区域椭圆  
//两个区域椭圆进行异或运算  
//绘制异或运算后的区域椭圆  
//创建椭圆对象  
//创建椭圆对象  
//创建区域椭圆  
//创建区域椭圆  
//两个区域椭圆进行异或运算  
//绘制异或运算后的区域椭圆

## 20.5 经典范例

### 20.5.1 经典范例 1: 绘制花瓣

 视频讲解: 光盘\TM\lx\20\绘制花瓣.exe

本范例使用坐标轴平移和图形旋转等技术绘制花瓣。编写 Java 程序并运行, 将在窗体上绘制花瓣。运行结果如图 20.20 所示。(实例位置: 光盘\TM\sl\20\17)

(1) 在项目中创建一个继承 JFrame 类的 DrawFlowerFrame 窗体类。

(2) 在 DrawFlowerFrame 窗体类中创建内部面板类 DrawFlowerPanel, 并重写 JComponent 类的 paint() 方法, 在该方法中实现花瓣的绘制。

(3) 将内部面板类 DrawFlowerPanel 的实例添加到窗体类 DrawFlowerFrame 的内容面板上, 用于在窗体上显示绘制的花瓣。代码如下:

```
class DrawFlowerPanel extends JPanel {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
    }
}
```

//创建内部面板类  
//重写 paint()方法  
//获得 Graphics2D 对象



图 20.20 绘制花瓣



```

//平移坐标轴
g2.translate(drawFlowerPanel.getWidth() / 2, drawFlowerPanel.getHeight() / 2);
//绘制绿色花瓣
Ellipse2D.Float ellipse = new Ellipse2D.Float(30, 0, 70, 20);           //创建椭圆对象
Color color = new Color(0,255,0);                                       //创建颜色对象
g2.setColor(color);                                                       //指定颜色
g2.fill(ellipse);                                                         //绘制椭圆
int i=0;
while (i<8){
    g2.rotate(30);                                                         //旋转画布
    g2.fill(ellipse);                                                     //绘制椭圆
    i++;
}
//绘制红色花瓣
ellipse = new Ellipse2D.Float(20, 0, 60, 15);                           //创建椭圆对象
color = new Color(255,0,0);                                               //创建颜色对象
g2.setColor(color);                                                       //指定颜色
g2.fill(ellipse);                                                         //绘制椭圆
i=0;
while (i<15){
    g2.rotate(75);                                                         //旋转画布
    g2.fill(ellipse);                                                     //绘制椭圆
    i++;
}
//绘制黄色花瓣
ellipse = new Ellipse2D.Float(10, 0, 50, 15);                           //创建椭圆对象
color = new Color(255,255,0);                                             //创建颜色对象
g2.setColor(color);                                                       //指定颜色
g2.fill(ellipse);                                                         //绘制椭圆
i=0;
while (i<8){
    g2.rotate(30);                                                         //旋转画布
    g2.fill(ellipse);                                                     //绘制椭圆
    i++;
}
//绘制红色中心点
color = new Color(255, 0, 0);                                             //创建颜色对象
g2.setColor(color);                                                       //指定颜色
ellipse = new Ellipse2D.Float(-10, -10, 20, 20);                       //创建椭圆对象
g2.fill(ellipse);                                                         //绘制椭圆
}
}

```

## 20.5.2 经典范例 2：绘制艺术图案

 视频讲解：光盘\TM1x\20\绘制艺术图案.exe

本范例使用坐标轴平移、图形旋转和获得随机数等技术绘制艺术图案。编写 Java 程序并运行，将在窗体上绘制艺术图案。运行结果如图 20.21 所示。（实例位置：光盘\TM1x\20\18）

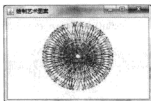


图 20.21 艺术图案

- (1) 在项目中创建一个继承 `JFrame` 类的 `ArtDesignFrame` 窗体类。
- (2) 在 `ArtDesignFrame` 窗体类中创建内部面板类 `ArtDesignPanel`，并重写 `JComponent` 类的 `paint()` 方法，在该方法中实现艺术图案的绘制。
- (3) 将内部面板类 `ArtDesignPanel` 的实例添加到窗体类 `ArtDesignFrame` 的内容面板上，用于在窗体上显示艺术图案。代码如下：

```

class ArtDesignPanel extends JPanel {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        Ellipse2D.Float ellipse = new Ellipse2D.Float(-80, 5, 160, 10);
        Random random = new Random();
        g2.translate(160, 90);
        int R = random.nextInt(256);
        int G = random.nextInt(256);
        int B = random.nextInt(256);
        Color color = new Color(R,G,B);
        g2.setColor(color);
        g2.draw(ellipse);
        int i=0;
        while (i<100){
            R = random.nextInt(256);
            G = random.nextInt(256);
            B = random.nextInt(256);
            color = new Color(R,G,B);
            g2.setColor(color);
            g2.rotate(10);
            g2.draw(ellipse);
            i++;
        }
    }
}

```

//创建内部面板类  
 //重写 paint()方法  
 //获得 Graphics2D 对象  
 //创建椭圆对象  
 //创建随机数对象  
 //平移坐标轴  
 //随机产生颜色的 R 值  
 //随机产生颜色的 G 值  
 //随机产生颜色的 B 值  
 //创建颜色对象  
 //指定颜色  
 //绘制椭圆  
  
 //随机产生颜色的 R 值  
 //随机产生颜色的 G 值  
 //随机产生颜色的 B 值  
 //创建新的颜色对象  
 //指定颜色  
 //旋转画布  
 //绘制椭圆

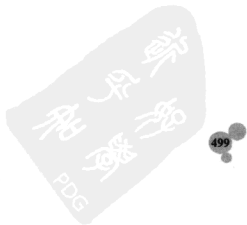
## 20.6 本章小结

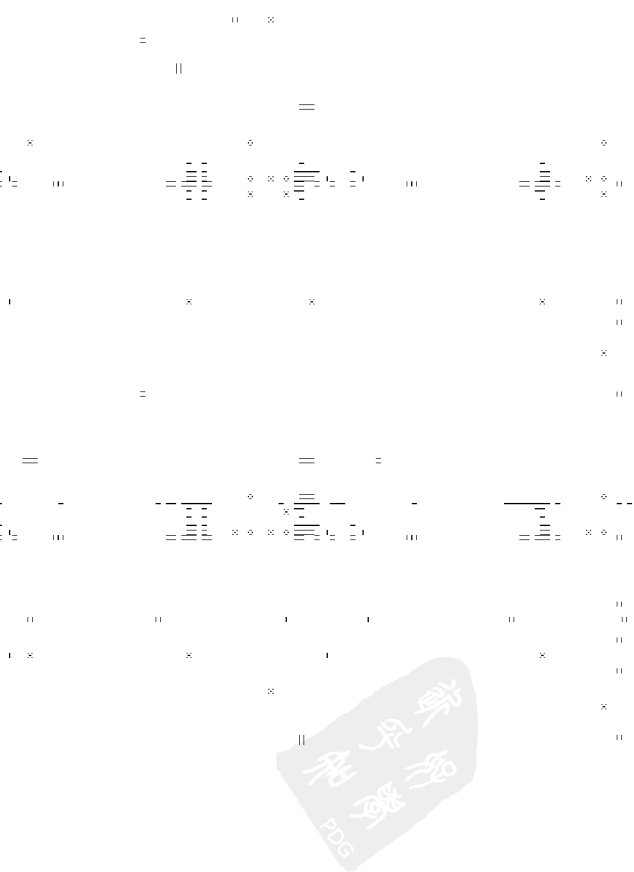
本章主要讲解了 Java 的绘图技术，是 `java.awt` 包所提供的功能。其中，有关绘图和图像处理技术的介绍比较详细，主要包括基本图形绘制、设置绘图颜色与笔画属性、绘制文本、绘制图片以及图像

的缩放、翻转、倾斜、旋转等图像处理技术。通过对本章的学习，读者应该能够掌握基本绘图技术和图像处理技术，在日后程序开发时，可以使用本章讲解的知识编写统计图表等功能。

## 20.7 实战练习

1. 创建一个主窗体，在窗体上分别绘制矩形、三角形、圆形和椭圆形。（答案位置：光盘\TM\sl\20\19）
2. 使用不同的颜色、不同的笔画属性绘制五环图形，并在五环图下显示年、月、日，文字要求使用宋体，大小为 14。（答案位置：光盘\TM\sl\20\20）
3. 尝试利用综合线程技术编写动画程序。（答案位置：光盘\TM\sl\20\21）





# 第 4 篇

## 项目篇

### 第 21 章 酒店管理系统

本篇通过开发一个完整的酒店管理系统，运用软件工程的设计思想，让读者学习如何进行 Java 项目的实践开发。书中按照编写项目设计思路→数据库设计→公共模块设计→主窗体设计→用户登录窗口设计→开台签单工作区设计→结账工作区设计→后台管理工作区设计的过程，带领读者一步一步亲身体验开发项目的全过程。

