第3章 Java图形处理和Java 2D

Java 语言的类库提供了丰富的绘图方法，其中大部分对图形、文本、图像的操作方法都定义在 Graphics 类中，Graphics 类是 java.awt 程序包的一部分。本章介绍的内容包括了颜色、字体处理、基本图形绘制方法、文本处理以及 Java 2D 中 Graphics2D 提供的基本图形绘制和图形特殊效果处理等方面。

3.1 Java 图形坐标系统和图形上下文

要将图形在屏幕上绘制出来，必须有一个精确的图形坐标系统来给该图形定位。与大多数其他计算机图形系统所采用的二维坐标系统一样，Java 的坐标原点(0,0)位于屏幕的左上角，坐标度量以像素为单位，水平向右为 X 轴的正方向，竖直向下为 Y 轴的正方向，每个坐标点的值表示屏幕上的一个像素点的位置，所有坐标点的值都取整数，如图 3-1 所示。这种坐标系统与传统坐标系统(如图 3-2 所示)有所不同。

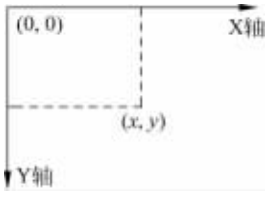

图 3-1 Java 坐标系

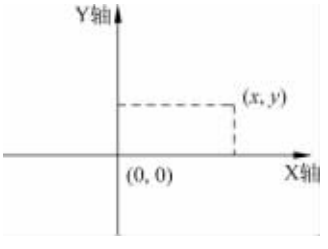

图 3-2 传统坐标系统

在屏幕上绘制图形时，所有输出都是通过一个图形上下文(graphics context)来产生的。图形上下文有时也称为图形环境，指允许用户在屏幕上绘制图形的信息，它由 Graphics 类封装，可以通过 Component 类的 getGraphics()方法返回获得。图形上下文表示一个绘制图层，如组件的显示区、打印机上的一页或一个屏幕外图像缓冲区。它提供了绘制 3 种图形对象(形状、文本和图像)的方法。

在 Java 中，可以通过 Graphics 的对象对图形上下文进行管理，Graphics 类定义了多种绘图函数，用户可以通过其提供的函数实现不同的图形绘制和处理。

在游戏编程开发中，常常在组件的 paint()方法内获得 java.awt 包中的 Graphics 类的对象，然后调用 Graphics 类中相应的绘制函数来实现输出。paint()方法是 java.awt.Component

类(所有窗口对象的基类)所提供的一个方法,当系统需要重新绘制组件时,将调用该方法。paint()方法只有一个参数,该参数是 Graphics 类的实例。下面给出一个实例。

```
public void paint(Graphics g)
{    Color myColor =  new Color(255, 0, 0);
     g.setColor(myColor);
     g.drawString("这是 Java 中的带颜色的文字串", 100,100) ;
     g.drawRect( 10,10,100 ,100) ;
}
```

组件绘制的时机:

(1) 组件外形发生变化时,如窗口的大小、位置、图标化等显示区域更新时,AWT 自动从高层直到叶结点组件相应地调用各组件的 paint()方法,但这可能有一个迟后感。

(2) 程序员也可直接调用某一个组件的 repaint()或 paint(),以立即更新外观(如在添加新的显示内容后)。

注意:如果要求保留上次的输出结果时可以调用 paint(),而不要求保留上次的输出结果只希望用户能看到最新的输出结果时可以调用 repaint()。

3.2 Color 类

可以使用 java.awt.Color 类为绘制的图形设置颜色。Color 类使用了标准 RGB(standard RGB,sRGB)颜色空间来表示颜色值。颜色由红(R)、绿(G)、蓝(B)三原色构成,每种原色的强度用一个 byte 值表示,每种原色取值为 0(最暗)~255(最亮),可以根据这 3 种颜色值的不同组合,显示不同的颜色效果,如(0,0,0)表示黑色,(255,255,255)表示白色。

在 Java 中 Color 类定义了 13 种颜色常量供用户使用,它们分别为 Color.black、Color.blue、Color. cyan、Color. darkGray、Color. gray、Color. green、Color. lightGray、Color.magenta、Color.orange、Color.pink、Color.red、Color.white 和 Color.yellow。从 JDK1.4 开始,也可以使用 Color 类中定义的新常量,它们和上述颜色常量一一对应,分别为 Color.BLACK、Color. BLUE、Color. CYAN、Color. DARK _ GRAY、Color. GRAY、Color.GREEN、Color. LIGHT _ GRAY、Color. MAGENTA、Color. ORANGE、Color. PINK、Color.RED、Color.WHITE 和 Color.YELLOW。

除此之外,用户也可以通过 Color 类提供的构造方法 Color(int *r*,int *g*,int *b*)创建自己需要的颜色。该构造方法通过指定红、绿、蓝 3 种颜色的值来创建一个新的颜色,参数 *r*、*g*、*b* 的取值范围为 0~255。如:

```
Color color = new Color(255,0,255);
```

一旦用户生成了自己需要的颜色,就可以通过 java.awt.Component 类中的 setBackground(Color c)和 setForeground(Color c)方法来设置组件的背景色和前景色,也可以使用该颜色作为当前的绘图颜色。

3.3 Font 类和 FontMetrics 类

3.3.1 Font 类

可以使用 java.awt.Font 类创建字体对象。Java 提供了物理字体和逻辑字体两种字体。AWT 定义了 5 种逻辑字体，分别为 SansSerif、Serif、Monospaced、Dialog 和 DialogInpu。

Font 类的构造方法为：

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

其中参数 name 为字体名，可以设置为系统上可用的任一字体，如 SansSerif、Serif、Monospaced、Dialog 或 DialogInput 等；参数 style 为字型，可以设置为 Font.PLAIN、Font.BOLD、Font.ITALIC 或 Font.BOLD + Font.ITALIC 等；参数 size 为字号，其取值为正整数。如：

```
Font font = new Font("Serif",Font.ITALIC,10);
```

如果需要找到系统上的所有可用字体，可以通过创建 java.awt.GraphicsEnviroment 类的静态方法 getLocalGraphicsEnviroment()的实例，调用 GetAllFonts()方法来获得系统的所有可用字体，或通过 getAvailableFontFamilyName()方法来取得可用字体的名字。如在生成可用的字体对象后，可以通过 java.awt.Component 类中的 setFont(Font f)方法设置组件的字体。

【例 3-1】 在控制台下输出系统所有的可用字体。程序运行结果如图 3-3 所示。

```
//ShowAvaliableFont.java
import java.awt.*;
public class ShowAvaliableFont {
     public static void main(String[] args) {
        GraphicsEnvironment e =
              GraphicsEnvironment.getLocalGraphicsEnvironment();
        String[] fontNames = e.getAvailableFontFamilyNames(); //获得可用字体名称
        int j = 0;
        for(int i = 0; i < fontNames.length; i++)
        {
            System.out.printf("%25s",fontNames[i]);
            j++;
            if(j % 3 == 0) System.out.println();
        }
    }
}
```

图 3-3 例 3.1 运行结果

3.3.2 FontMetrics 类

使用 drawString(String s,int x,int y)方法可以指定在框架的(x,y)位置开始显示字符串,但是如果想在框架的中央显示字符串,需要使用 FontMetrics 类。FontMetrics 类是一个抽象类,要使用 FontMetrics 对象,可以通过调用 Graphics 类中的 getFontMetrics()方法。FontMetrics 定义字体的度量,给出了关于在特定的组件上描绘特定字体的信息。这些字体信息包括了 ascent(上升量)、descent(下降量)、leading(前导宽度)和 height(高度)。其中 leading 用于描述两行文本间的间距,如图 3-4 所示。

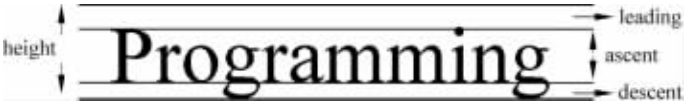

图 3-4 字体信息示意图

FontMetrics 类提供了下面几种方法用于获取 ascent、descent、leading 和 height:

- int getAscent(); //取得由当前 FontMetrics 对象描述的字体的 ascent 值
- int getDescent(); //取得由当前 FontMetrics 对象描述的字体的 descent 值
- int getLeading(); //取得由当前 FontMetrics 对象描述的字体的 leading 值
- int getHeight(); //取得使用当前字体的一行文本的标准高度

【例 3-2】 在框架中央位置显示字符串"Java Programming",并将字体设置为 Serif、粗斜体、大小为 30,颜色为红色,而将框架背景设置为淡灰色。程序源代码见 FontMetricsDemo.java,程序运行结果如图 3-5 所示。

```
//FontMetricsDemo.java
import java.awt.*;
import javax.swing.JFrame;
public class FontMetricsDemo extends JFrame{
    public FontMetricsDemo() {
    super();
    setTitle("FontMetrics Demo");
    setSize(300,200);
    setVisible(true);
    }
    public void paint(Graphics g) {
```

```
        Font font = new Font("Serif",Font.BOLD + Font.ITALIC,30);   //建立字体
        g.setFont(font);                                            //设置当前使用字体
        setBackground(Color.LIGHT_GRAY);                            //设置框架的背景颜色
        g.setColor(Color.RED);
        FontMetrics f = g.getFontMetrics();              //建立 FontMetrics 对象
        int width = f.stringWidth("Java Programming");   //取得字符串的宽度
        int ascent = f.getAscent();                      //取得当前使用字体的 ascent 值
        int descent = f.getDescent();                    //取得当前使用字体的 descent 值
        int x = (getWidth() - width)/2;
        int y = (getHeight() + ascent)/2;
        g.drawString("Java Programming",x,y);
    }
    public static void main(String[] args) {
        FontMetricsDemo fmd = new FontMetricsDemo();
        fmd.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

图 3-5　例 3.2 运行结果

3.4　常用的绘图方法

3.4.1　绘制直线

在 Java 中可以使用下面方法绘制一条直线：

```
drawLine(int x1,int y1,int x2,int y2);
```

其中参数 $x1$、$y1$、$x2$、$y2$ 分别表示该直线的起点($x1$,$y1$)和终点($x2$,$y2$)的坐标值。

3.4.2　绘制矩形

Java 中提供了绘制空心矩形(只绘制矩形的轮廓)和填充矩形的方法,分别针对普通直角矩形、圆角矩形和三维矩形有不同的绘制方法。

1. 普通直角矩形

可以使用下面方法绘制普通直角矩形的轮廓：

```
drawRect(int x,int y,int width,int height);
```

若需要绘制一个有填充颜色的普通直角矩形，可以使用下面方法：

```
fillRect(int x,int y,int width,int height);
```

这两种方法的参数含义相同，x、y 分别表示矩形左上角的 x 坐标和 y 坐标，width、height 分别表示矩形的宽和高。

2. 圆角矩形

可以使用下面方法绘制圆角矩形的轮廓：

```
drawRoundRect(int x,int y,int width,int height,int arcWidth,int arcHeight);
```

若需要绘制一个有填充颜色的圆角矩形，可以使用下面方法：

```
fillRoundRect(int x,int y,int width,int height,int arcWidth,int arcHeight);
```

这两种方法的参数含义相同，x、y 分别表示矩形左上角的 x 坐标和 y 坐标，width、height 分别表示矩形的宽和高，参数 arcWidth 和 arcHeight 分别表示圆角弧的水平直径和竖直直径，如图 3-6 所示。

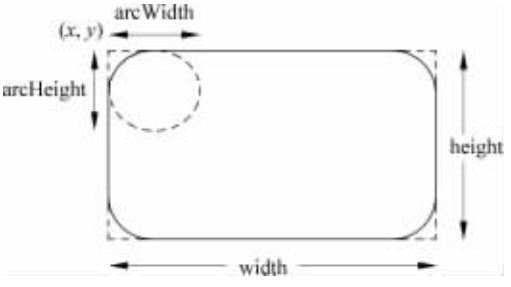

图 3-6 绘制圆角矩形示意图

3. 三维矩形

可以使用下面方法绘制三维矩形的轮廓：

```
draw3DRect(int x,int y,int width,int height,boolean raised);
```

若需要绘制一个有填充颜色的三维矩形，可以使用下面方法：

```
fill3DRect(int x,int y,int width,int height,boolean raised);
```

这两种方法的参数含义相同，x、y 分别表示矩形左上角的 x 坐标和 y 坐标，width、height 分别表示矩形的宽和高，raised 为真(True)表示矩形从表面凸起，raised 为假(False)表示矩形从表面凹进。

3.4.3 绘制椭圆

可以使用下面方法绘制空心椭圆：

```
drawOval(int x,int y,int width,int height);
```

若需要绘制一个有填充颜色的椭圆，可以使用下面方法：

```
fillOval(int x,int y,int width,int height);
```

这两种方法的参数含义相同，x、y 分别表示该椭圆外接矩形左上角的 x 坐标和 y 坐标，width、height 分别表示外接矩形的宽和高，如图 3-7 所示。如果设置外接矩形为正方形，即 width 和 height 相等，则可以绘制圆。

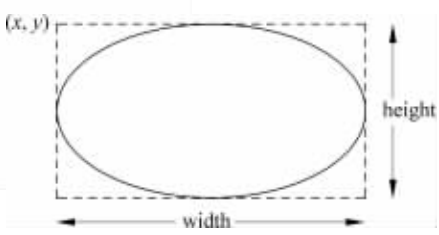

图 3-7　绘制椭圆示意图

【例 3-3】 在框架中绘制直线、矩形和椭圆。程序源代码见 DrawImageDemo.java，程序运行结果如图 3-8 所示。

```
import java.awt.*;
import javax.swing.*;
public class DrawImageDemo extends JFrame{
    public DrawImageDemo() {
    super();
    setTitle("Draw Line Rectangle Ellipse");
    setSize(300,300);
    setVisible(true);
    }
    public void paint(Graphics g) {
         //绘制直线、空心矩形和空心椭圆
        g.setColor(Color.red);
        g.drawRect(10,30,getWidth()/2-50,getHeight()/2-50);
        g.drawOval(10,30,getWidth()/2-50,getHeight()/2-50);
        g.drawLine(10,30,5+getWidth()/2-50,30+getHeight()/2-50);

         //绘制填充色为淡灰色的 3D 矩形、圆角矩形和椭圆
        g.setColor(Color.LIGHT_GRAY);
        g.fill3DRect(10,180,getWidth()/2-50,getHeight()/2-50,true);
        g.fillRoundRect(130,30,getWidth()/2-50,getHeight()/2-50,30,40);
        g.fillOval(130,180,getWidth()/2,getHeight()/2-50);
    }
    public static void main(String[] args) {
        DrawImageDemo d=new DrawImageDemo();
        d.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

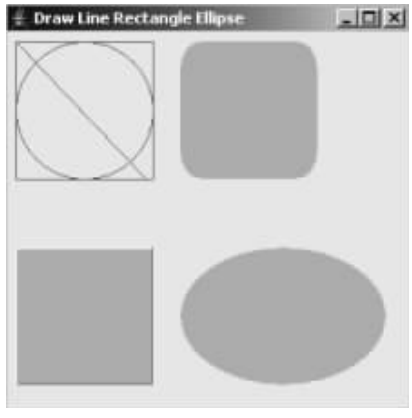

图 3-8　例 3.3 运行结果

3.4.4 绘制弧形

弧形可以看作椭圆的一部分，所以它的绘制也是根据其外接矩形进行的。通过drawArc()方法和fillArc()方法可以分别绘制弧线和扇形。这两种方法为：

```
drawArc(int x,int y,int width,int height,int startAngle,int arcAngle);
fillArc(int x,int y,int width,int height,int startAngle,int arcAngle);
```

其中 x、y、width、height 参数的含义和 drawOval()方法的参数含义相同，参数 startAngle 表示该弧的起始角度，参数 arcAngle 表示生成角度(从 startAngle 开始转了多少度)，且水平向右方向表示 0 度，从 0 度开始沿逆时针方向旋转为正角，如图 3-9 所示。

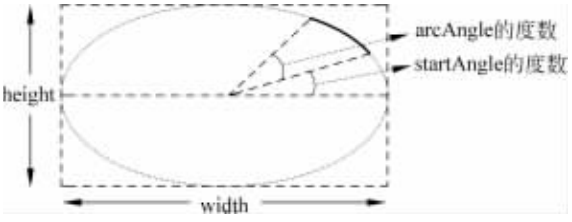

图 3-9 绘制弧形示意图

3.4.5 绘制多边形和折线段

1. 绘制多边形

使用 drawPolygon()方法和 fillPolygon()方法可以分别绘制多边形的外框轮廓和填充多边形。

```
drawPolygon(int[] xPoints,int[] yPoints,int nPoints);
fillPolygon(int[] xPoints,int[] yPoints,int nPoints);
```

其中多边形的顶点是由数组 xPoints 和 yPoints 中对应下标的相应元素组成的坐标来指定，数组 xPoints 存储所有顶点的 x 坐标，数组 yPoints 存储所有顶点的 y 坐标，参数 nPoints 指定多边形的顶点个数。drawPolygon()方法在绘制多边形时并不自动关闭多边形的最后一条边，而仅是一段开放的折线。所以，若想画封闭的边框型多边形，不要忘了在数组的尾部再添上一个起始点的坐标。

除此以外，drawPolygon(Polygon p)方法和 fillPolygon(Polygon p)方法也可以用来绘制多边形。这两种方法的参数是一个 Polygon 类的对象。

若想在多边形上增加一个顶点，可以使用 addPoint(int x,int y)方法。因此可以通过先创建一个空的 Polygon 对象，再重复调用 addPoint()方法将所有多边形的顶点加入到创建的 Polygon 对象中，然后通过调用 drawPolygon(Polygon p)方法或 fillPolygon(Polygon p)的方式绘制多边形。

2. 绘制折线段

可以使用 drawPolygonline()方法绘制折线段：

```
drawPolygonline(int[] xPoints,int[] yPoints,int nPoints);
```

其中数组 xPoints 存储所有顶点的 x 坐标，数组 yPoints 存储所有顶点的 y 坐标，nPoints 指定折线段顶点的个数。

【例 3-4】 在 JPanel 中绘制扇形和星形。程序源代码见 SimpleDraw.java，程序运行结果如图 3-10 所示。

```
import java.awt.*;
import javax.swing.*;
public class SimpleDraw {
    public static void main(String[] args) {
        DrawFrame frame = new DrawFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
class DrawFrame extends JFrame {
    public DrawFrame() {
        setTitle("简单图形绘制");
        setSize(300,300);
        DrawPanel panel = new DrawPanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
    }
}
class DrawPanel extends JPanel {
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        int x1 = 50,y1 = 50,x2 = 50,y2 = 150;
        int radius = 100;                    //半径
        int startAngle = -90;                //起始角度
        int arcAngle = 180;                  //弧的角度
        g.drawLine(x1,y1,x2,y2);             //画线
        g.drawArc(x1 - radius/2,y1,radius,radius,startAngle,arcAngle);
        Polygon p = new Polygon();
        x1 += 150; y1 += 50; radius /= 2;
        for (int i = 0; i < 6; i++)
            p.addPoint((int)(x1 + radius * Math.cos(i * 2 * Math.PI / 6)),
           (int)(y1 + radius * Math.sin(i * 2 * Math.PI / 6)));
        g.drawPolygon(p);                    //画六边形
    }
}
```

图 3-10　例 3.4 运行结果

3.4.6 清除绘制的图形

可以使用 clearRect()方法清除绘制的图形：

```
clearRect(int x,int y,int width,int height);
```

用背景色填充指定矩形以达到清除该矩形的效果，也就是说，当一个 Graphics 对象使用该方法时，相当于在使用一个“橡皮擦”。参数 x、y 是被清除矩形的左上角的坐标；另外两个参数是被清除矩形的宽和高。

3.5 Java 2D 简介

3.5.1 Java 2D API

Java 2D API(Application Programming Interface)增强了抽象窗口工具包(AWT)的图形、文本和图像功能，可以创建高级图形库，开发更为强大的用户接口和新型的 Java 应用程序。Java 2D API 对 AWT 进行了扩展，提供了更加灵活、功能更全面的绘制包，使其支持更多的图形绘制操作。

Java 2D 是 Java 核心类库的一部分，它包含的包有：

- java.awt
- java.awt.image
- java.awt.color
- java.awt.font
- java.awt.geom
- java.awt.print
- java.awt.image.renderable
- com.sun.image.codec.jpeg

其中 java.awt 包中包含一般的或比原有类增强的 Java 2D API 类和接口；java.awt.image 和 java.awt.image.renderable 包中包含用于图像定义和绘制的类和接口；java.awt.color 包中包含用于颜色空间定义和颜色监视的类和接口；java.awt.font 包中包含用于文本布局和字体定义的类和接口；java.awt.geom 包中包含所有与几何图形定义相关的类和接口；java.awt.print 包中包含用于打印所有基于 Java 2D 的文本、图形和图像的类和接口。

3.5.2 Graphics2D 简介

Graphics2D 扩展了 java.awt.Graphics 包，使得对形状、文本和图像的控制更加完善。Graphics2D 对象保存了大量用来确定如何绘制图形的信息，其中大部分都包含在一个 Graphics2D 对象的 6 个属性之中，这 6 个属性分别如下。

(1) 绘制(paint)：该属性确定所绘制线条的颜色以及填充图形的颜色和图案等。用户可以通过 setPaint(Paint p)方法进行该属性值的设置。

(2) 画笔(stroke)：该属性可以确定线条的类型以及粗细，还有线段端点的形状。用户可以通过 setStroke(Stroke s)方法进行该属性值的设置。

(3) 字体(font)：该属性可以确定所显示字符串的字体。用户可以通过 setFont(Font f)方法进行该属性值的设置。

(4) 转换(transform)：该属性确定了图形绘制过程中要应用的转换方法，通过指定转换方法可将所画内容进行平移、旋转和缩放。用户可以通过 setTransform()方法进行该属性值的设置。

(5) 剪切(clip)：该属性定义了组件上某区域的边界。用户可以通过 setClip(Clip c)方法进行该属性值的设置。

(6) 合成(composite)：该属性定义了如何绘制重叠的几何图形，使用合成规则可以确定重叠区域的显示效果。用户可以通过 setComposite(Composite c)方法来设置该属性的值。

一般情况下，我们使用 Griphics2D 对象的方法进行图形的绘制工作，Griphics2D 对象的常用方法如下。

(1) abstract void clip(Shape s)：将当前 Clip 与指定 Shape 的内部区域相交，并将 Clip 设置为所得的交集。

(2) abstract void draw(Shape s)：使用当前 Graphics2D 上下文的设置勾画 Shape 的轮廓。

(3) abstract void drawImage(BufferedImage img, BufferedImageOp op, int x, int y)：呈现使用 BufferedImageOp 过滤的 BufferedImage 应用的呈现属性，包括 Clip、Transform 和 Composite 属性。

(4) abstract boolean drawImage(Image img, AffineTransform xform, ImageObserver obs)：呈现一个图像，在绘制前进行从图像空间到用户空间的转换。

(5) abstract void drawString(String s,float x,float y)：使用 Graphics2D 上下文中的当前文本属性状态呈现由指定 String 指定的文本。

(6) abstract void drawString(String str, int x, int y)：使用 Graphics2D 上下文中的当前文本属性状态呈现指定的 String 的文本。

(7) abstract void fill(Shape s)：使用 Graphics2D 上下文的设置，填充 Shape 的内部区域。

(8) abstract Color getBackground()：返回用于清除区域的背景色。

(9) abstract Composite getComposite()：返回 Graphics2D 上下文中的当前 Composite。

(10) abstract Paint getPaint()：返回 Graphics2D 上下文中的当前 Paint。

(11) abstract Stroke getStroke()：返回 Graphics2D 上下文中的当前 Stroke。

(12) abstract boolean hit(Rectangle rect, Shape s, boolean onStroke)：检查指定的 Shape 是否与设备空间中的指定 Rectangle 相交。

(13) abstract void rotate(double theta)：将当前的 Graphics2D Transform 与旋转转换连接。

(14) abstract void rotate(double theta, double x, double y)：将当前的 Graphics2D Transform 与平移后的旋转转换连接。

(15) abstract void scale(double sx, double sy)：将当前的 Graphics2D Transform 与可缩放转换连接。

(16) abstract void setBackground(Color color)：设置 Graphics2D 上下文的背景色。

(17) abstract void setComposite(Composite comp)：为 Graphics2D 上下文设置 Composite Composite 用于所有绘制方法中，如 drawImage、drawString、draw 和 fill，它指定新的像素如何在呈现过程中与图形设备上的现有像素组合。

(18) abstract void setPaint(Paint paint)：为 Graphics2D 上下文设置 Paint 属性。

(19) abstract void setStroke(Stroke s)：为 Graphics2D 上下文设置 Stroke。

(20) abstract void setTransform(AffineTransform Tx)：重写 Graphics2D 上下文中的 Transform。

(21) abstract void shear(double shx, double shy)：将当前的 Graphics2D Transform 与剪裁转换连接。

(22) abstract void translate(double tx, double ty)：将当前的 Graphics2D Transform 与平移转换连接。

(23) abstract void translate(int x, int y)：将 Graphics2D 上下文的原点平移到当前坐标系统中的点 (x,y)。

3.5.3 Graphics2D 的图形绘制

Graphics2D 是 Graphics 类的子类，也是一个抽象类，不能实例化 Graphics2D 对象，为了使用 Graphics2D，可以通过 Graphics 对象传递一个组件的绘制方法给 Graphics2D 对象。方法如下面代码段所示：

```
public void paint(Graphics g){
    Graphics2D g2 = (Graphics 2D)g;
    …
}
```

Java 2D API 提供了几种定义点、直线、曲线、矩形和椭圆等常用几何对象的类，这些新几何类是 java.awt.geom 包的组成部分，包括如 Point2D、Line2D、Arc2D、Rectangle2D 和 Ellipse2D 等。每个类都有单精度和双精度两种像素定义方式，如 Point2D.double 和 Point2D.float、Line2D.double 和 Lint2D.float 等，用这些类可以很容易地绘制基本的二维图形对象。

【例 3-5】 使用 Graphics 2D 绘制直线、矩形和椭圆。程序运行结果如图 3-11 所示。

```
//Graphics2DDemo.java
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;
public class Graphics2DDemo extends JFrame{
    public Graphics2DDemo() {
    super();
    setTitle("Draw 2D Shape Demo");
```

```
        setSize(300,200);
        setVisible(true);
        }
        public void paint(Graphics g) {
            //建立 Graphics2D 对象
            Graphics2D g2 = (Graphics2D) g ;
            //建立 Line2D 对象
            Line2D l = new Line2D.Double(50,50,200,50);
            g2.draw(l);
            //建立 Rectangle2D 对象
            Rectangle2D r = new Rectangle2D.Float(30,80,100,100);
            Color c = new Color(10,20,255);
            g2.setColor(c);
            g2.draw(r);

            //建立 Ellipse2D 对象
            Ellipse2D e = new Ellipse2D.Double(150,80,100,100);
            g2.setColor(Color.GRAY);
            g2.fill(e);
        }
        public static void main(String[] args) {
            Graphics2DDemo g = new Graphics2DDemo();
            g.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
```

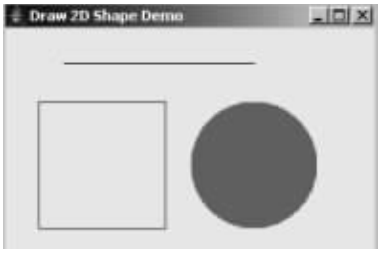

图 3-11　例 3.5 运行结果

3.5.4　Graphics2D 的属性设置

Graphics2D 通过设置对象的属性来确定如何绘制图形的信息，在前面已经介绍了 Graphics 2D 对象的 6 种属性，包括 paint、stroke、font、transform、clip 和 composite。下面将介绍如何在 Graphics 2D 的图形上下文中设置常用的属性 paint、stroke 和 composite。

(1) paint 用于填充绘制图形的颜色或图案，在 Java 2D API 中提供了两种 paint 属性的填充方式：GradientPaint 和 TexturePaint。GradientPaint 则定义在两种颜色间渐变的填充方式，而 TexturePaint 是利用重复图像片段的方式定义一种纹理填充方式。可以用 setPaint()方法设置定义好的填充方式来并将其应用于绘制图形中。

GradientPaint 类提供了下面的构造方法来建立颜色渐变方式：

① GradientPaint(float x1,float y1,Color color1,float x2,float y2,Color color2)；

② GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2, boolean cyclic)；

③ GradientPaint(Point2D p1,Color color1,Point2D p2,Color color2)；

④ GradientPaint(Point2D p1,Color color1,Point2D p2,Color color2,boolean cyclic)；

其中构造方法①和②中参数 $x1$、$y1$ 指定颜色渐变的起点坐标，$x2$、$y2$ 指定颜色渐变的终点坐标，填充颜色从 color1 渐变至 color2。构造方法②中参数 cyclic 为 True 时，填充方式和构造方法①定义的相同，若 cyclic 为 False，则填充方式为非周期性渐变。构造方法③的和构造方法①类似，构造方法④的和构造方法②类似，只是在构造方法③和④中使用了 Point2D 对象来指定填充颜色渐变的起始(p1)和结束(p2)位置。

TexturePaint 类的构造方法为：

```
TexturePaint(BufferedImage txtr,Rectangle2D anchor).
```

其中 txtr 用来定义一个单位的填充图像的材质，anchor 用来复制材质。

【例 3-6】 使用 GradientPaint 渐变填充方式和 TexturePaint 纹理填充方式绘制图形。程序源代码见 PaintDemo.java，程序运行结果如图 3-12 所示，使用纹理填充方式绘制图形的填充单元图像如图 3-13 所示。

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
public class PaintDemo extends JFrame{
    public PaintDemo() {
    super();
    setTitle("Paint Demo");
    setSize(300,200);
    setVisible(true);
    }
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;

        //建立两个 Point2D 对象 p1、p2,将它们作为颜色渐变的起点和终点
        Point2D p1 = new Point2D.Double(70,70);
        Point2D p2 = new Point2D.Double(90,90);
        GradientPaint gp = new GradientPaint(p1,Color.white,p2,Color.black,true);
        Ellipse2D e1 = new Ellipse2D.Double(30,60,90,90);
        g2.setPaint(gp);                                    //设置画笔绘制方式
        g2.fill(e1);

        //建立用于填充图形的一个单位的材质
     BufferedImage bi = new BufferedImage(10,10,BufferedImage.TYPE_INT_RGB);
        Graphics2D bg = bi.createGraphics();
```

```
            bg.setColor(Color.white);
            bg.fillRect(0,0,10,10);
            bg.setColor(Color.red);
            bg.fillRect(0,0,5,5);
            bg.fillRect(5,5,5,5);
        Rectangle2D r1 = new Rectangle2D.Double(0,0,10,10);        //设置复制填充材质的区域
            Ellipse2D e2 = new Ellipse2D.Double(150,60,90,90);     //创建需要绘制的图形
            TexturePaint tp = new TexturePaint(bi,r1);             //设置填充方式
            g2.setPaint(tp);                                       //设置绘制方式
            g2.fill(e2);
        }
        public static void main(String[] args) {
            PaintDemo p = new PaintDemo();
            p.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
```

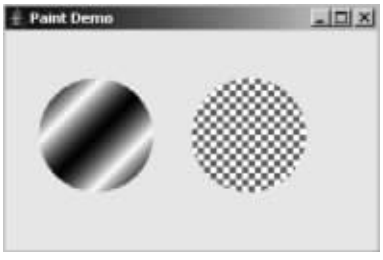

图 3-12　例 3.6 运行结果

图 3-13　用于填充图形的单元图像

(2) stroke 用于在绘制图形的轮廓时确定线条的形状和粗细，通常使用 BasicStroke 对象来定义、通过 setStroke()方法设定 stroke 的属性值。BasicStroke 定义的特性包括线条宽度、笔形样式、线段连接样式和短划线图案等。使用 stroke 属性设定图形轮廓的绘制方式时，首先调用 setStroke()方法设定轮廓的绘制方式，然后使用 setPaint()方法定义画笔如何绘制该图形，最后使用 draw()方法来绘制该图形。

在 BasicStroke 中定义了一组基本的简单图形轮廓的直线绘制和点划线绘制方式，它提供了 3 种绘制粗线的末端样式：CAP_BUTT、CAP_ROUND 和 CAP_SQUARE；以及 3 种线段连接样式：JOIN_BEVEL、JOIN_MITER 和 JOIN_ROUND。

BasicStroke 类提供了下面的构造方法来建立画笔的绘制方式：

① BasicStoke ()；

② BasicStoke (float width)；

③ BasicStoke (float width,int cap,int join)；

④ BasicStoke (float width,int cap,int join,float miterlimit)；

⑤ BasicStoke (float width,int cap,int join,float miterlimit,float[] dash,float dash_phase)；

其中 width 表示轮廓线的宽度；cap 表示轮廓线末端的样式；join 表示相交线段的连接样式；milterlimit 表示在 JOIN_MITER 模式下若相交的尖形末端大于 minterlimit，则超过

部分被削去；dash 为虚线的样式，数组内的值为短划线和空白间距的值；dash_phase 为虚线样式数组的起始索引。

【例 3-7】 使用 BasicStroke 类设定画笔绘制样式。程序源代码见 StrokeDemo.java，程序运行结果如图 3-14 所示。

```
//StrokeDemo.java
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;
public class StrokeDemo extends JFrame{
    public StrokeDemo() {
    super();
    setTitle("Stroke Demo");
    setSize(300,200);
    setVisible(true);
    }
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        Line2D l = new Line2D.Double(30,50,100,80);
        //创建一个 BasicStroke 对象来设置直线的绘制方式
        Stroke stroke = new BasicStroke(10.0f,
                BasicStroke.CAP_ROUND,BasicStroke.JOIN_BEVEL);
        g2.setStroke(stroke);
        g2.draw(l);

        Ellipse2D e = new Ellipse2D.Double(150,50,90,90);
        //创建一个 BasicStroke 对象来设置椭圆的绘制方式
        stroke = new BasicStroke(8,BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_BEVEL,0,new float[] { 10,5 },0);
        g2.setStroke(stroke);
        g2.draw(e);
        Rectangle2D r = new Rectangle2D.Double(30,100,80,80);
        //创建一个 BasicStroke 对象来设置矩形的绘制方式
        stroke = new BasicStroke(10,BasicStroke.CAP_SQUARE,
                BasicStroke.JOIN_ROUND,0);
        g2.setStroke(stroke);
        g2.draw(r);
    }
    public static void main(String[] args) {
        StrokeDemo s = new StrokeDemo();
        s.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

(3) composite 用于定义绘制重叠图形的绘制方式。当绘制多个图像时，遇到图像重叠的情况，需要确定重叠部分的颜色显示方式，重叠区域的像素颜色决定了该部分图像的透明程度。在 composite 的定义中，最常用的就是 AlphaComposite，可以通过 setComposite()方法来将 AlphaComposite 对象添加到 Graphics2D 上下文中，设置图像重叠部分的复合样式。

AlphaComposite 类中定义了多种新颜色与已有颜色的复合规则，如 SRC_OVER 表示混合时新颜色（源色）应覆盖在已有颜色（目标色）之上；DST_OUT 表示混合时去除已有颜色；SRC_OUT 表示混合时去除新颜色；DST _OVER 表示混合时用已有颜色覆盖新颜色。在设置混合颜色的同时，还可以设置颜色的透明度 alpha 值，它用百分比表示在颜色重叠时当前颜色的透明度，alpha 值从 0.0（完全透明）～1.0（完全不透明）。

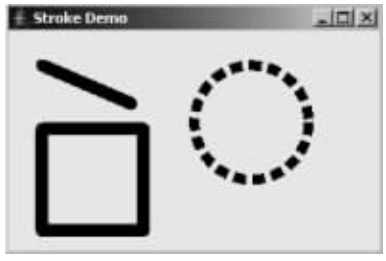

图 3-14　例 3.7 运行结果

3.5.5　路径类

在 java.awt.geom 包中定义了几何图形类，包括点、直线、矩形、圆、椭圆、多边形等。该包中各类的层次结构如下：

```
| - java.lang.Object
        | - java.awt.geom.AffineTransform
        | - java.awt.geom.Area
        | - java.awt.geom.CubicCurve2D
                | - java.awt.geom.CubicCurve2D.Double
                | - java.awt.geom.CubicCurve2D.Float
        | - java.awt.geom.Dimension2D
        | - java.awt.geom.FlatteningPathIterator
        | - java.awt.geom.Line2D
                | - java.awt.geom.Line2D.Double
                | - java.awt.geom.Line2D.Float
        | - java.awt.geom.Path2D
                | - java.awt.geom.Path2D.Double
                | - java.awt.geom.Path2D.Float
                        | - java.awt.geom.GeneralPath
        | - java.awt.geom.Point2D
                | - java.awt.geom.Point2D.Double
                | - java.awt.geom.Point2D.Float
                | - java.awt.geom.QuadCurve2D
                        | - java.awt.geom.QuadCurve2D.Double
                        | - java.awt.geom.QuadCurve2D.Float
        | - java.awt.geom.RectangularShape
                | - java.awt.geom.Arc2D
                        | - java.awt.geom.Arc2D.Double
                        | - java.awt.geom.Arc2D.Float
                | - java.awt.geom.Ellipse2D
                        | - java.awt.geom.Ellipse2D.Double
                        | - java.awt.geom.Ellipse2D.Float
                | - java.awt.geom.Rectangle2D
```

路径类用于构造直线、二次曲线和三次曲线的几何路径。它可以包含多个子路径。如上面的类层次结构所描述，Path2D 是基类（它是一个抽象类）；Path2D. Double 和 Path2D. Float 是其子类，它们分别以不同精度的坐标定义几何路径；GeneralPath 在 1.5 及其以前的版本中，它是一个独立的最终类，在 1.6 版本中进行了调整与划分，其功能由 Path2D 替代，为了其兼容性，把它划为 Path2D. Float 派生的最终类。下边以 GeneralPath 类为例介绍一下路径类的功能与应用。

1. 构造方法

构造路径对象的方法如下。

（1）**GeneralPath**(int rule)：以 rule 指定缠绕规则构建对象。缠绕规则确定路径内部的方式，缠绕规则有两种：Path2D. WIND_EVEN_ODD 用于确定路径内部的奇偶（even-odd）缠绕规则；Path2D. WIND_NON_ZERO 用于确定路径内部的非零（non-zero）缠绕规则。

（2）**GeneralPath**()：以默认的缠绕规则 Path2D. WIND_NON_ZERO 构建对象。

（3）**GeneralPath**(int rule, int initialCapacity)：以 rule 指定缠绕规则和 initialCapacity 指定的容量（以存储路径坐标）构建对象。

（4）**GeneralPath**(Shape s)：以 Shape 对象 s 构建对象。

2. 常用方法

路径对象的常用方法如下。

（1）void **append**(Shape s, boolean connect)：将指定 Shape 对象的几何形状追加到路径中，也许使用一条线段将新几何形状连接到现有的路径段。如果 connect 为 true 并且路径非空，则被追加的 Shape 几何形状的初始 moveTo 操作将被转换为 lineTo 操作。

（2）void **closePath**()：回到初始点使之形成闭合的路径。

（3）boolean **contains**(double x, double y)：测试指定坐标是否在当前绘制边界内。

（4）void **curveTo**(float x1, float y1, float x2, float y2, float x3, float y3)：将 3 个新点定义的曲线段添加到路径中。

（5）Rectangle2D **getBounds2D**()：获得路径的边界框。

（6）Point2D **getCurrentPoint**()：获得当前添加到路径的坐标。

（7）int **getWindingRule**()：获得缠绕规则。

（8）void **lineTo**(float x, float y)：绘制一条从当前坐标到(x,y)指定坐标的直线，将(x,y)坐标添加到路径中。

（9）void **moveTo**(float x, float y)：从当前坐标位置移动到(x,y)指定位置，将(x,y)添加到路径中。

（10）void **quadTo**(float x1, float y1, float x2, float y2)：将两个新点定义的曲线段添加到路径中。

（11）void **reset**()：将路径重置为空。

（12）void **setWindingRule**(int rule)：设置缠绕规则。

（13）void **transform**(AffineTransform at)：使用指定的 AffineTransform 变换此路径

的几何形状。

【例 3-8】 使用 GeneralPath 类绘制一个正三角形和一个倒三角形。运行效果如图 3-15 所示。

图 3-15 使用 GeneralPath 类创建正三角形和倒三角形路径对象

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.JFrame;
public class AddPathExample extends JFrame{
    public AddPathExample() {
        super();
        setTitle("GeneralPath Demo");
        setSize(200,200);
        setVisible(true);
        }
        public void paint(Graphics g) {
            Graphics2D g2 = (Graphics2D) g;
            GeneralPath myPath, myPath2;
            // myArray 包含正三角形的所有顶点
            Point[] myArray =
             {
                 new Point(130,130), new Point(160,160),
                 new Point(100,160), new Point(130,130)
             };
            myPath = new GeneralPath();                          //建立 GeneralPath 对象
            myPath.moveTo(myArray[0].x, myArray[0].y);           //起始顶点
            for(int i=1; i<myArray.length; i++)                  //创建正三角形的路径
                myPath.lineTo(myArray[i].x, myArray[i].y);       //绘制直线
            g2.draw(myPath);                                     //画正三角形
            // myArray2 包含倒三角形的所有顶点
            Point[] myArray2 =
             {
                 new Point(130,130),    new Point(100,100),
                 new Point(160,100),    new Point(130,130)
             };
            myPath2 = new GeneralPath();                         //建立 GeneralPath 对象
            myPath2.moveTo(myArray2[0].x, myArray2[0].y);        //起始顶点
            for(int i=1; i<myArray2.length; i++)                 //创建倒三角形的路径
                myPath2.lineTo(myArray2[i].x, myArray2[i].y);    //绘制直线
            g2.draw(myPath2);                                    //画倒三角形
```

```
        }
        public static void main(String[] args) {
            AddPathExample gp = new AddPathExample();
          gp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
```

GeneralPath 还可以包含多个图形，其中每一个图形被称为子路径(figures)。如果在 GeneralPath 对象中使用了 closePath 函数，在这之后对路径所添加的线条都将构成一个子路径。

3.5.6 平移、缩放或旋转图形

需要平移、缩放或旋转一个图形，可以使用 AffineTransform 类来实现。

(1) 首先使用 AffineTransform 类创建一个对象：

```
AffineTransform  trans = new AffineTransform();
```

对象 trans 具有最常用的 3 个方法来实现对图形的变换操作：

translate(double a,double b)：将图形在 X 轴方向移动 a 个单位像素，Y 轴方向移动 b 个像素单位。a 是正值时向右移动，是负值时向左移动；b 是正值时向下移动，是负值时向上移动。

scale(double a,double b)：将图形在 X 轴方向缩放 a 倍，Y 轴方向缩放 b 倍。

rotate(double number,double x,double y)：将图形沿顺时针或逆时针以(x,y)为轴点旋转 number 个弧度。

(2) 进行需要的变换，比如要把一个矩形绕点(100,100)顺时针旋转 60 度，那么就要先做好准备：

```
trans.rotate(60.0 * 3.1415927/180,100,100);
```

(3) 把 Graphics 对象，比如 g_2d，设置为具有 trans 功能的“画笔”：

```
g_2d.setTransform(trans);
```

假如 rect 是一个矩形对象，那么 g_2d.draw(rect)画的就是旋转后的矩形。