热点透视

Java 技术在二维图象处理方面的应用

徐 鹏　王克宏

（清华大学计算机系，北京　100084）

摘　要　本文旨在分析 Java 技术在图象处理方面的功能和具体编程接口的设计，并针对 Java 2D API 中基本图形的绘制、文字的设置和图象成像等具体功能进行介绍，同时提供了丰富的实例。

关键字　Java，二维成像模型，所见即所得

随着 Java 技术的不断完善和发展，Java 2D API 应运而生。Java 2D 应用编程接口 API 是建立在通过 java.awt 包定义的图形和图象处理类基础上的，它在保持与现有程序兼容的同时，还扩展了很多功能。开发人员可以通过使用 Java 2D API 很方便地在 Java 应用程序和 Applet 程序中组合高质量的二维图形、文字和图象。

Java 2D API 提供了一个二维成像模型，通过这个模型来勾画图形、文字和图象，同时还可以对这些内容进行颜色调整、坐标空间变换以及图形、图象合成等操作。通过 Java 2D API，您可以将显示和打印操作中使用的成像模型统一起来，这样就为用户提供了真正的所见即所得 WYSIWYG（What You See Is What You Get）的设计模式。

1　Java 2D API 介绍

1.1　Java 2D API 的基本原理

Java 2D API 可以处理各种类型的文字和图象，并提供了一种统一的机制来进行诸如旋转和比例放大等变换操作，以及对可扩展字体和颜色的支持。通过使用 Java 2D API，开发人员可以通过与二维图形状态相关的一个完善的属性集合来控制如何构成图形。您可以指定图形的特性，例如边线宽度、连接类型、填充的颜色和文字。

Java 2D API 定义了 2 种坐标空间，分别是用户坐标空间和设备坐标空间。设备坐标空间的源点位于左上角，x 轴的正方向向右，而 y 轴的正方向向下，如图 1 所示。

程序设计人员在独立于设备的用户坐标空间中对所有的图形对象进行描述，而一个二维图形对象

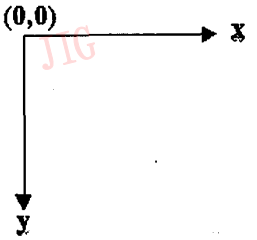

图 1

在最终设备上显示时需要进行从图形对象的用户坐标空间到设备坐标空间坐标的转换。在缺省状态下，系统将坐标空间转换成缺省的用户坐标空间，而这种坐标空间与设备坐标空间是完全相同的。

1.2　绘制过程

Java 2D API 使用了通过 java.awt 包定义的图象绘制模型。在这个模型中，每个组件对象实现了一个绘画方法，并可以在需要时被自动激活。假设您具有一个完成绘制一个红色矩形操作的组件，为了使用 java.awt 绘制这个矩形，您可以这样实现 Component.paint 方法：

```
public void paint(Graphics g) {
g.setColor(Color.red);
g.fillRect(300, 300, 200, 100);
```

这个程序展示的基本绘制过程如下：

（1）通过调用一个 Graphics 属性方法（例如 setColor）来指定图形的显示属性；

（2）定义绘制图形的形状，例如矩形；

（3）使用 Graphics 对象并通过调用一个 Graphics 绘制方法（例如 fillRect）绘制出图形；当您使用 Java 2D API 中的功能时，基本的绘制过程是相同的。而 Java 2D API 提供了一些额外的功能用

于指定绘制的风格，定义更加复杂的图形以及控制图形的绘制过程。为了使用这些功能，您必须实现 paint 方法，并构造一个 Graphics2D 对象，例如：

```
Graphics2D g2d = (Graphics2D) g;
```

下面实例展示了应当如何通过使用 Java 2D API 来绘制出一个红色矩形：

```
public void paint(Graphics g) {
// 引入 Graphics 参数，这样就可以调用 Graphics2D 函数了
Graphics2D g2d = (Graphics2D) g;
// 指定属性
g2d.setColor(Color.red);
// 定义形状(使用奇偶规则)
GeneralPath rect
new GeneralPath(GeneralPath.EVEN_ODD);
path.moveTo(300.0f, 400.0f); // 左下角
path.lineTo(500.0f, 400.0f); // 右下角
path.lineTo(500.0f, 300.0f); // 右上角
path.lineTo(300.0f, 300.0f); // 左上角
path.closePath(); // 封闭矩形
// 绘制图形
g2d.fill(rect);
}
```

除了在绘制过程中使用了 Java 2D API 中的类 GeneralPath 来定义矩形以外，整个程序与使用 java.awt 来实现相同功能的程序并没有太大的差异。

Java 2D API 使您可以通过使用这种相同的机制来实现更加复杂的绘制操作。然而，为了简化生成标准图形(例如矩形、椭圆、曲线等)的过程，Java 2D API 提供了除 GeneralPath 以外的许多 Shape 子类。例如，您可以使用一个 Rectangle2D.Double 代替 GeneralPath，来定义前面实例中的那个矩形，例如：

```
Rectangle2D.Double rect =
new Rectangle2D.Double(300, 300, 200, 100);
```

而这个矩形与使用 GeneralPath 定义的矩形并不是完全相同的。在第一个实例中建构的 GeneralPath 对象的起始点位于(300, 400)，这是矩形左下角的坐标点；而 Rectangle2D.Double 对象的起始点位于(300, 300)，这是矩形左上角的坐标点。当您构造一个 Rectangle2D.Double 对象时，如果提供的长度和宽度参数为负数，那么将生成一个大小为(0,0)的空矩形。由于坐标空间中的原点位于左上角，因此您就不能够使用 Rectangle2D.Double 来生成一个原点位于(300, 400)的矩形。

为了使用 Java 2D API 中的类来完成图形的绘制工作，您必须完成如下操作：

(1) 指定显示属性除了支持以基本颜色填充之外，您还可以通过 Java 2D API 中的类使用图案和文字对图形进行填充。为了指定复杂的填充方式，您需要使用方法 setPaint( )；

(2) 定义一个图形、文字字符串或者图象：Java 2D API 可以对图形、文字和图象进行相同的处理，如旋转、缩放、倾斜处理并使用相应的方法进行合成。在前面的实例中定义了一个简单的矩形。Shape 接口定义了一系列方法用于描绘出几何路径。Java 2D API 提供了一些定义简单图形(例如矩形、弧形和椭圆等)的 Shape 实例。其中 GereralPath 就实现了 Shape 接口，它可以被用于通过组合直线和二阶、三阶 Bezier 曲线来构成任意复杂的图形。GeneralPath 还利用一个参数来指定弯曲规则。弯曲规则被用于判断当路径分成各个独立的部分而又出现交叉现象时，一个点是否位于图形的内部。程序员可以为一个 GeneralPath 对象指定 2 种不同的弯曲规则，分别是奇偶弯曲规则或者非零弯曲规则。前面实例中使用的就是奇偶弯曲规则；

(3) 为了实现图形、文字或者图象，您可以调用一个 Graphics2D 实现方法。在前面的实例中，程序通过使用 fill 方法实现了一个矩形。在绘制一个 Java 2D API 对象之前，必须进行与 Graphics2D 对象相关的坐标系转换处理操作，将一个点或者一条路径转换成一个新的点或路径。在构造一个 Graphics2D 对象时，所生成的缺省转换对象将图形转换到设备坐标空间中。为了获得旋转、缩放等效果，系统将转换对象并将其应用到 Graphics2D 对象上。在 Java 2D API 中实现的转换方法是 AffineTransform，它将完成一些线性转换操作。

Java 2D API 的强大功能体现在其管理复杂的绘制操作的能力上。在前面的实例中我们绘制了一个矩形，在这里我们又绘制了第 2 个矩形，这个矩形将覆盖第一个矩形的一部分，并逆时针旋转 45°。第 2 个矩形用蓝色填充，同时具有 50%的透明度，这样在浏览程序运行结果时仍旧可以看见第 1 个矩形。通过利用 Java 2D API，程序员可以很方便地在源程序的基础上添加第 2 个矩形：

```
public void paint(Graphics g) {
Graphics2D g2d = (Graphics2D) g;
g2d.setColor(Color.red);
GeneralPath rect =
```

```
new GeneralPath(GeneralPath.EVEN_ODD);
path.moveTo(0.0f, 0.0f); // 左下角
path.lineTo(200.0f, 0.0f); // 右下角
path.lineTo(200.0f, -100.0f); // 右上角
path.lineTo(0.0f, -100.0f); // 左上角
path.closePath(); // 封闭矩形
AffineTransform at = new AffineTransform();
at.setToTranslation(300.0, 400.0);
g2d.transform(at);
g2d.fill(rect);
// 增加第 2 个矩形
g2d.setColor(Color.blue); // 定义颜色
AlphaComposite comp = AlphaComposite.getInstance(AlphaComposite.SRC_OVER,0.5f);
g2d.setComposite(comp); //设置组合方式
// Rotate about the origin -45 deg in radians
at.setToRotation(-Math.PI/4.0));
g2d.transform(at);
g2d.fill(rect);
}
```

2 个矩形绘制过程的共性包括：

(1) 用一个 GeneralPath 对象对矩形进行定义；

(2) 通过调用 setColor 指定颜色，蓝色矩形通过调用 setComposite 方法指定 50%的透明度；

(3) 在绘制矩形之前应用坐标系转换程序(Graphics2d.transform 被用于将 2 个矩形定位于(300，400) 点，同时将第 2 个矩形逆时针旋转 45°)；

(4) 通过调用 fill( )绘制出矩形；

这个实例遵循基本的绘制过程来完成定义新对象的颜色属性、进行坐标系转换和显示等操作。

(5) 定义颜色：为了绘制第 2 个矩形，同时使其具备 50%的透明度，首先需要设置颜色：

```
g2d.setColor(Color.blue); //定义颜色
```

(6) 定义合成操作：还需要对新颜色与现有颜色的混合方式进行定义。为了完成这项操作，需要生成一个 AlphaComposite 对象。一个 AlphaCompoite 对象定义了一个合成操作，所谓合成操作就是指定各种颜色的搭配方式。在这个实例中，我们希望将第 2 个矩形设置为具有 50%透明度，同时其颜色要覆盖在现有颜色之上。因此，在生成 AlphaComposite 对象时可指定 SRC_OVER 操作，并将代表透明度的参数值设定为 0.5。为了使用这个新的 Composite 对象，需调用 Graphics2D.setComposite 方法：

```
AlphaComposite comp = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 5f);
g2d.setComposite(comp); //设置组合方式
```

(7) 定义旋转角度：通过构造一个 AffineTransform 对象并调用其中的方法 setToRotation( )可以指定图形旋转的角度，并利用前面构造的 Graphics2D 对象调用方法 transform( )来实现坐标系的转换操作；

```
at.setToRotation(-Math.PI/4.0));
g2d.transform(at);
```

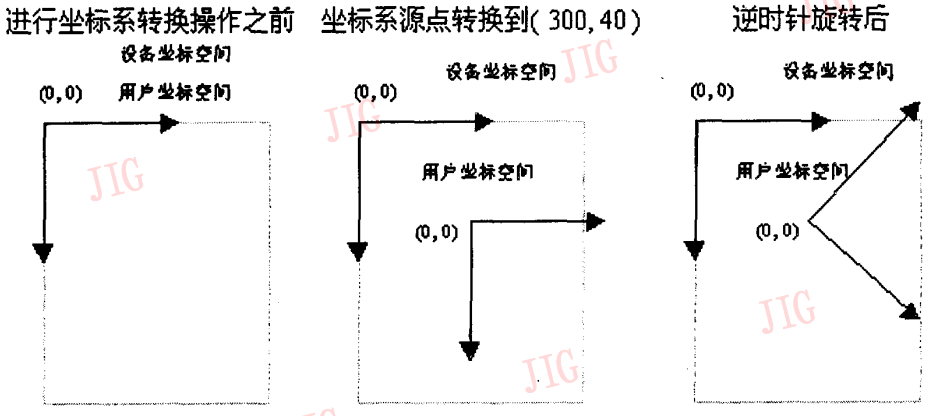

图 2　坐标空间的转换过程

(8) 与第 1 个矩形类似，调用方法 g2b.fill(path)绘制出这个蓝色矩形；

Graphics2D 对象通过使用指定的转换对象对路径进行转换。最终的结果得到了一个旋转后的矩形，它使用了指定的颜色，这种颜色与特定图象的颜色相混合。程序结果如图 3 所示。

对于经过复杂的坐标系变换的图形来说，要想监测用户在图形的哪个位置上进行了鼠标单击操作

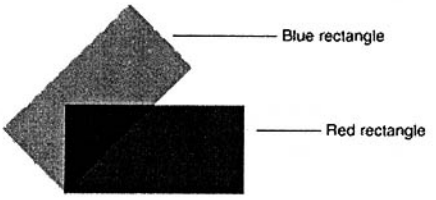

图3 激活 pain 方法所绘制出的图象

图4 使用文字作为一个剪贴路径

是比较复杂的。Java 2D API 通过提供一个 Graphics2D 方法简化了这个任务。此方法采用一个 Rectangle 对象和一个 Shape 对象作参数,并返回布尔值表明矩形内部是否有点位于图形路径之上。

1.3 文字

Java 2D API 提供了文字处理功能,从简单地使用字体到提供专业水准的字符布局和字体功能。Java 2D API 强化的 Font 类与现有的 java.awt.Font 类相比,提供了更加丰富的字体控件。Java 2D API 中的 Font 类将取代 java.awt.Font。

为了绘制文字,您同样可以使用路径法来完成这个过程。不同之处是这里并不是使用一个 GeneralPath 对象来定义路径的形状,可以生成一个 Font 对象并通过调用方法 Graphics2D.drawString( )来绘制出文字。例如,为了在通过上面的实例程序绘制出的 2 个矩形之上再绘制出一个逆时针旋转 45°的大写字母"J",您可以在 paint( )方法中加入如下的代码:

```
// 设置字体为 Helvetica-BoldOblique,大小为 200 点
Font myFont = new Font("Helvetica-BoldOblique",
Font.PLAIN, 200);
    // 设置字母"J"的颜色为绿色
    // 已经完成了旋转处理
    g2d.setColor(Color.green);
    // 字母 J 不透明显示
    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1.0f));
    // 绘制 J
    g2d.drawString("J", 0f, 20f);
```

因为 Java 2D API 中的 Font 类提供了一个 getGlyphOutline 方法来返回字符路径,因此您可以使用一个文字字符串作为一个剪贴路径。例如,您可以在绘制 2 个矩形时通过使用相应的字符路径作为一个剪贴路径,从而绘制出矩形中贯穿一个经过旋转后的字母"J"的部分,如图 4 所示。

为了实现这个功能,您需要通过调用方法 getGlyphOutline( )来获得字符的外形轮廓,此方法的返回值是一个 Shape 实例。之后您可以将 Shape 对象作为由 Graphics2D 方法定义的 setClip( )的参数。在图 4 中,文字的外围轮廓被包上了黑边。正象您在实例中所看到的那样,在 Java 2D API 中,文字就象其它图形对象一样可以进行坐标系转换,并可以作为剪贴路径来使用。您可以调用方法 Graphics2D.hitString( )来进行命中检测操作。

1.4 图象

Java 2D API 在 java.awt 和 java.awt.image 包中用于处理图象的诸多类的基础上又添加了一些新类,其中包括 BufferedImage、ComponentcolorModel 和 ColorSpace,这样就进一步完善了图象处理功能。

这些类为高级 Java 程序员提供了有关图象的更多控件。通过使用 Java 2D API 中的图形处理类,您可以在特定的彩色空间而不是 RGB 空间中生成图象。Java 2D API 图象处理类还使您能够精确地指定象素如何在内存中进行分配。与所有其它图形元素类似,图象可以在绘制过程中通过一个与 Graphics2D 对象相关的转换操作来进行转换。这就意味着可以对图象进行放缩、旋转倾斜或者类似文字和路径那样的转换操作。然而,图象并没有使用当前的颜色,而是包含了自己的颜色信息(图象具有嵌入式模板来解释象素数据)。

程序可以直接显示一个图象。在获得一个图象(可以通过一个 URL 对象来获取图象)后,您可以调用 Graphics2D.drawImage 来绘制图象,并可以同时完成相应的转换操作:

```
Image image = applet.getImage(url);
AffineTransform at = new AffineTransform();
at.rotate(Math.PI/4.0);
g2d.drawImage(image, at, this);
```

图象可以针对每个象素携带透明度信息。这种信息被称为 Alpha 通道，它通常是与当前的 Composite 对象结合起来使用的，用于将原图象与现有的绘制过程结合起来。

图 5 中包含了 3 个图象，它们分别具有不同的透明度信息，而每个图象都是在一个蓝色矩形的基础上显示的。在这个实例中，假定构造了一个 AlphaComposite 对象，并使用了 SRC_OVER 来完成合成操作。

图 5　具有不同透明度的图象

在第 1 个图象中，对狗身体部分的所有象素采用不透明处理，或者对背景中的所有象素采用透明处理都可以达到当前的效果。您可以在很多网页上看到利用这种效果处理的图象。第 1 个图象对狗身体部分采用了非全透明处理。而第 3 个图象则对狗面部区域采用了全透明值，而随着狗身体中其它部位离面部距离的增加而增大透明度的取值。

Java 2D API 对 Java 语言的抽象窗口工具集进行了扩展，提供了一个标准的、跨平台的接口用于处理复杂的图形、文字和图象。通过利用 Java 2D API 中的类，您可以在您的应用程序和 Applet 程序中合成高质量的二维图形、文字和图象。

我们可以将 Java 2D API 的功能做如下概括：

(1) 实现了与设备和分辨率相独立的高质量的图形；

(2) 强化了对字体和文本处理功能的支持；

(3) 提供了一个单一的、完善的显示模型；

2　显　示

“显示”也就是在一个输出设备上绘制一个图形对象的过程。在显示过程中一些必需的操作包括定义、生成和显示一个对象。当您使用 Java 2D API 时，显示过程通过 Graphics2D 对象及其状态属性来控制。Graphics2D 扩展了 java.awt.Graphics 类并提供了在显示图形、文字和图象功能方面更完善的控件。Graphics2D 定义了一些用于绘制对象和状态属性的方法来显示过程控制。通过这些状态属性，您可以实现下列操作：

(1) 设置一个剪贴路径用来限制被显示的区域；

(2) 定义用于填充文字或图形的颜色和图案；

(3) 变换边框宽度；

(3) 改变边线的连接方式；

(4) 指定多个图形对象的组合方式；

2.1　显示过程

显示过程可以划分为以下 4 个阶段：

(1) 被显示的图形对象通过使用 Graphics2D 对象中的转换操作，转换到设备空间。这就决定了图形对象应当在哪里被显示。完成这项操作的方式独立于被显示的图形对象的类型；

(2) 当显示一个图形时，其 PathIterator 对象被用于提取在图形路径中的元素。如果要勾画出一个路径，那么 Graphics2D 对象中的 Stroke 对象被用于将路径转换为一个勾画路径。通过使用与 Graph-

ics2D 相关的转换操作，图形将被转换到设备坐标空间；

(3) 当显示一个图象时，其在用户坐标空间中的外围边框将通过使用与 Graphics2D 相关的转换操作转换到设备坐标空间中；

(4) 当前的剪贴路径被用于对显示操作产生牵制作用。剪贴路径可以是任何形状的，它可以通过一个 Shape 对象来进行描述。当程序调用 setClip 或者 clip 方法时，系统将通过使用相应的转换操作将剪贴内容转换到设备空间中；

系统通过使用下面的标准来判断所要显示的颜色：

(1) 对于图象来说，颜色从图象数据中获取；

(2) 对于文字和路径来说，当前在 Graphics2D 中的 Paint 对象或者 Color 对象提供了有关颜色的信息；

(3) 程序通过使用当前的 Composite 对象将当前的颜色应用到最终的显示设备中；

当在光栅图形显示设备上显示图形时，这些图形的边缘会出现缺口。弧线和对角线由于它们是通过使用光栅网格中的象素来勾勒形状的，因此在实现时会出现缺口和楼梯状的外观。图 6 展示了在一个低分辨率的设备上绘制出的文字，您可以很明显地看出文字中存在缺口和楼梯状现象。

图 6 在低分辨率的设备上显示的文字

在 Java 2D API 中提供了反模糊技术，它主要是用于对特定对象的边缘进行平滑处理。这项技术的实施需要借助于其它的计算资源，并可能影响到显示速度。在使用 Java 2D API 的过程中，您可以调用 setRen－deringHints 来指定是希望尽可能快速地显示出对象，还是希望在显示对象时提供尽可能高的显示质量，但并不是所有的平台都支持对显示模式进行修改。

2.2 转换 2D 对象

Java 2D API 提供了一个转换类—AffineTransform，它的功能是完成二维对象的放缩、旋转和剪切等操作。

Java 2D API 中的一种变换方式就是均匀变换，这也就是对原始图形进行一次线性转换，通常是将直线转换为直线，经平行线转换为平行线，但是在非平行线之间进行转换时可能改变点之间的距离和角度的大小。均匀变换基于下面这个二维矩阵：

$$\begin{bmatrix} a & c & t_x \\ b & d & t_y \end{bmatrix}$$

此处 $x' = ax + cy + tx, y' = bx + dy + ty$

为了使用 AffineTransform 类，您不需要直接对转换矩阵进行交互。您只需要简单地按照一定的序列来调用方法，以实现旋转等转换操作。与 Graphics2D 对象相关的转换操作可以对您从用户空间向设备空间绘制的所有图形、文字和图象进行转换。Graphics2D 实现了一个 drawImage，利用一个 AffineTransform 对象实例来作为一个参数。这样您就可以在绘制一个图象的同时对图象进行其它转换处理。类似地，您可以将一个 AffineTransform 实例应用到一个 Font 对象，以便生成一个新的 Font 对象用以绘制文字。您可以使用转换方法将一次转换操作与当前的 Graphics2D 转换结合起来，这样它就成为了一个转换操作流水线的一部分。最后指定的转换操作将被首先应用到图象上。

2.3 生成新图形

您可以通过实现一个 Shape 接口来生成一个定义新图形的类。例如，您可以实现一个 Shape 接口用以通过几组点来形成一个多边形。这个 PolygonPath 类可以定义一个新方法 addPoint( )。为了建立一个 PolaygonPath 对象，客户程序将会重复调用 addPoint。一旦建立了多边形，那么它就可以作为一个参数传递给 draw、setClip 或者其它需要一个 Shape 对象的方法了。

PolygonPath 类必需实现下述 Shape 接口方法：

```
contains( );
getBounds( );
getBounds2D( );
getPathIterator( );
intersects( );
```

2.4 勾画路径

Java 2D API 提供了一个基本的 Stroke 类，其中包含了一些特性，例如线宽、线段端点类型、连接部分类型以及虚线类型。线段端点类型包括：CAP_

BUTT(切线类型)、CAP_ROUND(圆形)和 CAP_SUQARED(矩形)。而不同直线连接处的类型包括：JOIN_BEVEL、JOIN_MITER 和 JOIN_ROUND。在图 7 的第 1 个图象中使用的连接类型为 JOIN_MITER；而第 2 个图象则使用的连接类型为 JOIN_ROUND，线段端点类型为圆形，并设置了虚线模式。

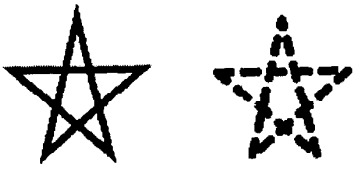

图 7　具有不同属性的 2 个图象

2.5　丰富的填充效果

利用 Java 2D API，您不但能够指定填充颜色，而且还可以对填充色进行由浅入深变化的处理或者进行网纹处理。为了实现这个功能，Java 2D API 定义了一个新的 Graphics2D 方法 setPaint( )和一个新的接口 Paint，并通过 GradientPaint 和 TexturePaint 实现了 Paint 接口。这样就大大节省了设计人员在生成复杂填充效果上所花费的时间。

图 8　丰富的填充效果

从理论上讲，所有的绘制操作都是通过一个 Paint 对象来完成的。一个 Color 对象是 Paint 对象的一个简单的类型，而 setColor( )正是 setPaint 的一个特殊的实例，它为您建立了一个利用单一颜色进行绘图的 Paint 对象。(因为 Color 实现了 Paint 接口并是另一类型的 Paint 对象，因此您可以将一个 Color 对象传递给 setPaint( )方法。)一旦您调用了 setPaint( )方法，那么包括文字和图形在内所有的内容都将通过使用特定的 Paint 方法来完成绘制。一个 Paint 对象必需指定绘制图形中每个象素所使用的颜色。从理论上讲，Java 2D API 对构成一个图形的象素进行判断，并要求 Paint 对象为每个象素设置颜色值。之后它将颜色值转换成特定输出设备的象素值。这无疑是一个烦琐的过程，且很难进行优化处理。为了对这个过程进行流水处理，Java 2D API 采取了成批处理象素的方法。一批数据可能是一个特定扫描行中的象素，也可能是一个象素块。这种批处理过程是按照下面的步骤来实现的：

(1) 通过 Paint 对象生成一个 PaintContext 对象，这个对象存储了有关当前显示操作和产生颜色所必需的信息。CreateContext 方法采用图形对象的边框作为参数，而在 ColorModel 中生成颜色，同时转换操作被用于将用户空间映射到设备空间。但是并非所有的 Paint 对象都能够支持一个 ColorModel；

(2) getColorModel( )仅仅被调用一次，但是 getTile( )在流水处理过程中需要被重复调用。之后这一信息被传递到流水处理过程的下一阶段，通过使用当前的 Composite 对象来绘制所生成的颜色；

2.6　图象合成

我们在前面简单讨论了基本的合成操作，并介绍了实现 Composite 接口的 AlphaComposite 类。这个类对很多不同的组合风格提供了支持。这个类的一些具体实例对描绘如何将一种新颜色与一种现有颜色进行组合的规则进行了具体化。

在 AlphaComposite 类中最常使用的一个组合规则就是 SRC_OVER。当应用 AlphaComposite. SRC—OVER 后表明新颜色应当覆盖在现有颜色基础之上。程序中可以生成具有特定 alpha 值的一些 AlphaComposite 对象用于增加所绘制的对象的透明度。这个特殊的 alpha 值于当前图形对象的 alpha 值结合在一起。这个结合后的 alpha 值表明现有颜色中多大比例能够透过新颜色显示出来，这也就是新颜色透明度的一个衡量标准。任何现有的颜色在 alpha＝1.0 时，即新颜色为不透明颜色时都不能够显示出来；而如果新颜色为透明状态，即 alpha＝0.0 时，则所有现存颜色都可以透过它显示出来。

下面的实例展示了您应当如何使用第 2 个 AlphaComposite 对象以使矩形部分透明。为了得到图 9 所示的效果，我们通过使用缺省的组合对象来绘制文字，这样文字就可以完全不透明了。之后，我们使用了一个 alpha 为 0.5 的 AphaComposite. SRC_OVER 对象来增加 2 个覆盖矩形的透明度：

```
public void paint(Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    Font myfont =new Font ("Helvetica-Bold",Font.
```

```
    PLAIN, 40);
StyledString myString1=
    new StyledString("Compositing", myfont);
    g2d.drawString(myString1,270f,280f);
    StyledString myString2=
    new StyledString("is Fun!", myfont);
    g2d.drawString(myString2,320f,340f);
    AlphaComposite comp=AlphaComposite.getInstance
    (AlphaComposite.SRC_OVER, 0.5f);
    g2d.setComposite(comp);
    g2d.setColor(Color.red);
    GeneralPath path=new GeneralPath
    (GeneralPath.EVEN_ODD);
    path.moveTo(0.0f, 0.0f); // 左下角
    path.lineTo(200.0f, 0.0f); // 右下角
    path.lineTo(200.0f, -100.0f); // 右上角
    path.lineTo(0.0f, -100.0f); // 左上角
    path.closePath(); // 封闭矩形
    AffineTransform at = new AffineTransform();
    at.setToTranslation(300.0, 400.0);
    g2d.transform(at);
    g2d.fill(path);
    g2d.setColor(Color.blue); // 定义颜色
// 旋转大约-45度
    at.setToRotation(-Math.PI/4.0));
    g2d.transform(at);
    g2d.fill(path);
}
```

图9 图象合成

程序员可以通过实现 Composite 和 CompositeContext 接口来生成一种全新的图象合成操作。一个 Composite 对象提供了一个控制状态并进行合成操作的 CompositeContext 对象。一个 Composite 对象中可以生成多个 CompositeContext 对象，这样就在一个多线程的环境中保持了一些相互独立的状态。

(未完待续)

VTEL 公司的专业服务体系在为客户设计和完成解决方案上保持世界领先水平

在当今飞速发展的通讯技术领域，“领先”的含义是多层面的。VTEL 美国视讯公司对领先地位的理解是指掌握最新的数字技术，在全球范围内提供咨询、设计和系统应用的服务。VTEL 公司在宾州费城的专业服务部在设计、管理、完成和支持数字视讯的解决方案等方面，在业界居领先地位。

VTEL 公司总裁兼首席执行官 Jerry Benson 说：“VTEL 的专业服务部通过在项目管理，培训和工程等方面为客户提供产品和服务，填补了我们从桌面到大型房间系统产品的空白。专业服务部不仅能安装 VTEL 的产品，还有能力安装竞争对手的产品。在这一点上其它公司是达不到这种服务水平的。”

VTEL 的技术服务体系是从 1995 年兼并的 Peirce-phelps 合成通讯系统集团(Integrated Communications Systems-ICS)发展而来的。VTEL 在这方面有 30 年的经验及 2 000 余次的视音频和会议电视系统安装经历。

通用电器公司(GE)工业控制系统部的系统设计技术联系人 Jemes Williams 说：“我们之所以选择 VTEL 专业服务部，是由于他们能预测我们的需求以及执行这种解决方案时的创造性。”

Williams 进一步说：“通讯对 GE 满足全球的客户需要而言至关重要。在 Duluth，我们所要求的就是为这一目的服务的技术和解决方案。在这一点上 VTEL 的专业服务部恰恰能满足我们的需要。