

```
    }
    public static void main( String[] args ) {
        launchApp( new ImageCreator() );
    }
}
```

## 10.3 执行颜色插值

颜色插值（color interpolation）是从一种颜色到另一种颜色的混合。此过程的第一部分是线性插值，并且由于公式是线性的，因此数学上并不复杂。

这个示例涉及很多有趣的概念。你可以把颜色混合背后的数学，应用于需要在一个距离之间插值的任何数值。使用这一技术来捕获单个的像素值，使你能够执行定制图像插值。此外，学习如何手动执行一种图像效果，而不依赖于隐藏的算法，这总是很有趣的事情。

给定任意两个点，如图 10.4 所示，线条上的任何 x 值对应一个 y 值，如图 10.5 所示。

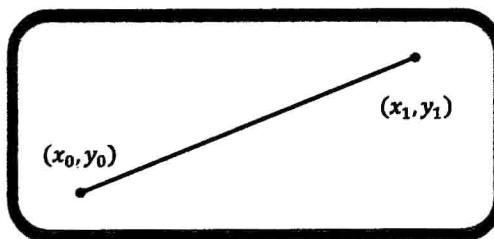


图 10.4 线段

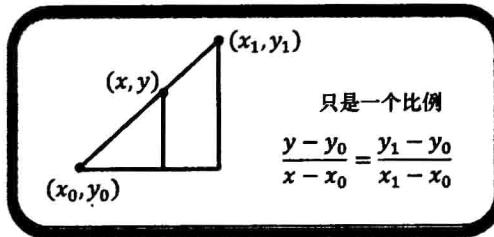


图 10.5 只是一个比例



使用如下的比例乘积求出 y:

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0}$$

如果让  $x_0 = 0$  且  $x_1 = 1$ :

$$y = y_0 + (x - 0)(y_1 - y_0 / 1 - 0)$$

$$y = y_0 + x(y_1 - y_0 / 1)$$

$$y = y_0 + x(y_1 - y_0)$$

这就是所谓的直线参数方程:

$$P = P_0 + t(P_1 - P_0)$$

这是有意义的，因为这个方程式在从  $P_0$  到  $P_1$  的线段之间，插入了一个点。

然而，混合颜色要更加复杂一些。给定一个起始颜色和一个结束颜色，以及两个点之间的像素距离，颜色就可以混合了，但是每个颜色成分必须各自混合，并且在每个阶段组合以创建特定的颜色。例如，从红色到绿色进行混合，每个颜色成分必须插值。

如图 10.6 所示，红色从 255 到 0 混合，绿色从 0 到 255 混合，蓝色没有任何操作。

当使用插值方程式混合颜色时，未知的值  $y$  是任何特定的点的颜色。要混合一个单个的颜色成分，例如，红色从 255 到 0 跨越 20 个像素，代码应该如下所示：

```
int y0 = 255; int y1 = 0;
int x0 = 0; int x1 = 20;
for( int x = x0; x < x1; ++x ) {
    int y = y0 + (x - x0) * (y1 - y0) / (x1 - x0);
    System.out.println(y);
}
```

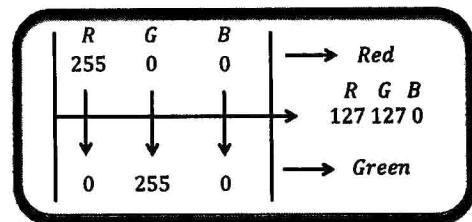


图 10.6 颜色插值

创建单个颜色并在颜色之间混合，有助于在 `BufferedImage` 中设置单个的像素值。如下的代码用来获取对一个 `BufferedImage` 像素数据的访问。

```
BufferedImage img = new BufferedImage( 400, 400,
    BufferedImage.TYPE_INT_ARGB );
WritableRaster raster = img.getRaster();
```

```
DataBuffer dataBuffer = raster.getDataBuffer();
DataBufferInt data = (DataBufferInt) dataBuffer;
int[] pixels = data.getData();
```

注意，这里强制转型为一个 DataBufferInt。如果图像的格式不是 bytes 或 shorts，相应的类型必须强制转型为正确的缓冲，才能获取像素。

ColorInterpolationExample 如图 10.7 所示，位于 javagames.images 包中，它在一个方形中混合颜色。



图 10.7 颜色插值示例

左边的颜色是从红色混合到蓝色，右边的颜色是从绿色混合到黑色。水平方向的颜色

在顶部是从红色混合到绿色，而在底部是从蓝色混合到黑色。如图 10.8 所示，在每一帧，都用 clear[] 数组清理所有像素，从而创建 BufferedImage，这个数组中只包含设置为 0 的像素。initialize()方法创建缓冲图像，获得了数组形式的原始像素值，然后，创建值全部为 0 的一个数组副本，用来清理每一帧的像素。

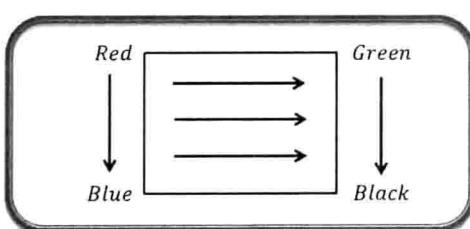


图 10.8 颜色插值方形

在每一帧中，updateObjects()方法通过从起始颜色到结束颜色的插值，来设置图像中



的每个像素值。如下的代码用来清理像素数组：

```
System.arraycopy( clear, 0, pixels, 0, pixels.length );
```

尽管图像是具有宽度和高度的一个方块，但像素数组只有一个维度。如下的代码使用宽度和高度来遍历单个的数组。

```
for( int row = 0; row < height; ++row ) {
    for( int col = 0; col < width; ++col ) {
        pixels[ row * width + col ] = 0;
    }
}
```

ColorInterpolationExample 的代码如下所示：

```
package javagames.images;
import java.awt.Graphics;
import java.awt.image.*;
import javagames.util.SimpleFramework;

public class ColorInterpolationExample extends SimpleFramework {
    private BufferedImage img;
    private int[] pixels;
    private int[] clear;
    public ColorInterpolationExample() {
        appWidth = 640;
        appHeight = 640;
        appSleep = 0L;
        appTitle = "Color Interpolation Example";
    }
    @Override
    protected void initialize() {
        super.initialize();
        img = new BufferedImage( 400, 400, BufferedImage.TYPE_INT_ARGB );
        // get pixels
        WritableRaster raster = img.getRaster();
        DataBuffer dataBuffer = raster.getDataBuffer();
        DataBufferInt data = (DataBufferInt) dataBuffer;
        pixels = data.getData();
        clear = new int[ pixels.length ];
    }
    @Override
    protected void updateObjects( float delta ) {
        super.updateObjects( delta );
    }
}
```



```
    createColorSquare();
}

private void createColorSquare() {
    int w = img.getWidth();
    float w0 = 0.0f;
    float w1 = w - 1.0f;
    int h = img.getHeight();
    float h0 = 0.0f;
    float h1 = h - 1.0f;
    System.arraycopy( clear, 0, pixels, 0, pixels.length );
    // Top-Left
    float tlr = 255.0f;
    float tlg = 0.0f;
    float tlb = 0.0f;
    // Bottom-Left
    float blr = 0.0f;
    float blg = 0.0f;
    float blb = 255.0f;
    // Top-Right
    float trr = 0.0f;
    float trg = 255.0f;
    float trb = 0.0f;
    // Bottom-Right
    float brr = 0.0f;
    float brg = 0.0f;
    float brb = 0.0f;
    float h1h0 = h1 - h0;
    float w1w0 = w1 - w0;
    for( int row = 0; row < h; ++row ) {
        // left pixel
        int lr = (int)(tlr + (row - h0) * (blr - tlr) / h1h0);
        int lg = (int)(tlg + (row - h0) * (blg - tlg) / h1h0);
        int lb = (int)(tlb + (row - h0) * (blb - tlb) / h1h0);
        // right pixel
        int rr = (int)(trr + (row - h0) * (brr - trr) / h1h0);
        int rg = (int)(trg + (row - h0) * (brg - trg) / h1h0);
        int rb = (int)(trb + (row - h0) * (brb - trb) / h1h0);
        for( int col = 0; col < w; ++col ) {
            int r = (int)(lr + (col - w0) * (rr - lr) / w1w0);
            int g = (int)(lg + (col - w0) * (rg - lg) / w1w0);
            int b = (int)(lb + (col - w0) * (rb - lb) / w1w0);
            int index = row * w + col;
            pixels[ index ] = 0xFF << 24 | r << 16 | g << 8 | b;
        }
    }
}
```



```
        }
    }
}

@Override
protected void render( Graphics g ) {
    super.render( g );
    int xPos = (canvas.getWidth() - img.getWidth()) / 2;
    int yPos = (canvas.getHeight() - img.getHeight()) / 2;
    g.drawImage( img, xPos, yPos, null );
}
public static void main( String[] args ) {
    launchApp( new ColorInterpolationExample() );
}
}
```

## 10.4 使用 VolatileImage 提高速度

BufferedImage 不是唯一可用的图像类型。VolatileImage 是另一种图像类型，比使用 BufferedImage 要快很多。由于 VolatileImage 的内容可能在任何时候丢失（其名字就是由此而来的），因此使用它会更加困难；但是，根据应用程序所要求的速度，它可能是一个可选方案。

当图像位于显卡的内存之中时，该内存对于操作系统中的任何程序都是全局的。例如，在 Windows 上，用户可以使用 **Ctrl+Alt+Delete** 来查看运行的程序，或者按下 **Alt+Tab** 从一个全屏应用程序切换回桌面。当这些操作中的任何一个发生时，任何非恒定图像都可能会丢失。即便切换回游戏而没有重新加载丢失的图像，这也会让垃圾数据显示到屏幕上。

使用一个 VolatileImage 的第一个不同之处在于，它不能由一个构造方法创建。如下的代码用于创建 VolatileImage。

```
GraphicsEnvironment ge =
    GraphicsEnvironment.getLocalGraphicsEnvironment();
GraphicsDevice gd = ge.getDefaultScreenDevice();
GraphicsConfiguration gc = gd.getDefaultConfiguration();
VolatileImage vi = gc.createCompatibleVolatileImage( width, height );
```

一旦图像可用后，如下的模式将用来进行渲染：

```
GraphicsConfiguration gc;
VolatileImage vi;
```



```
Graphics g;
do {
    int returnCode = vi.validate( gc );
    if( returnCode == VolatileImage.IMAGE_RESTORED ) {
        // Contents need to be restored
        renderVolatileImage();
    } else if( returnCode == VolatileImage.IMAGE_INCOMPATIBLE ) {
        // incompatible GraphicsConfig
        createVolatileImage();
        renderVolatileImage();
    }
    g.drawImage( vi, 0, 0, null );
} while( vi.contentsLost() );
```

所有额外的工作都是值得的，因为 `VolatileImage` 可能是硬件加速的。一个缺点是，单个的像素无法作为 `BufferedImage` 的数组而使用。

`ImageSpeedTest` 如图 10.9 所示，位于 `javagames.images` 包中，它使用了 4 个不同的方法把图像绘制到屏幕上。`BufferedImage` 和 `VolatileImage` 都进行了绘制，一种状态只创建图像一次，并且在每一帧绘制它；另一种状态在每一帧创建图像然后将其绘制到屏幕。当图像只创建一次时，这两种图形类型之间确实没什么区别。原因在于，`BufferStrategy` 在其后台使用的是一个 `VolatileImage`，这隐藏了复杂性并且考虑到了硬件加速。最大的速度差距，产生于在每一帧中创建图像。创建 `VolatileImage` 可以更快一些，因此，如果游戏需要在每一帧中重新绘制图像的话，`VolatileImage` 是一种实现方式。如果只在启动时加载图像，则使用一个 `BufferedImage` 更容易，并且它将保存在内存中直到程序结束。

`initialize()` 方法创建了一个 `BufferedImage` 和一个 `VolatileImage`。`renderToBufferedImage()` 和 `renderToVolatileImage()` 使用相同的代码来渲染图像。`processInput()` 方法通过 B 键，在 `BufferedImage` 和 `VolatileImage` 之间切换；并且使用 R 键在只渲染一次和渲染每一帧之间切换。`render()` 方法使用 `realtime` 和 `bufferedImage` 变量来确定应该使用 `VolatileImage` 还是 `BufferedImage`，以及在将图像绘制到屏幕之前是否要渲染它。

`renderToVolatileImage()` 方法绘制已经渲染的图像。`renderToVolatileImageEveryFrame()` 方法在将图像绘制到屏幕之前，也重新绘制它。正如你所见到的，在每一帧渲染 `VolatileImage` 可能比使用 `BufferedImage` 更快。

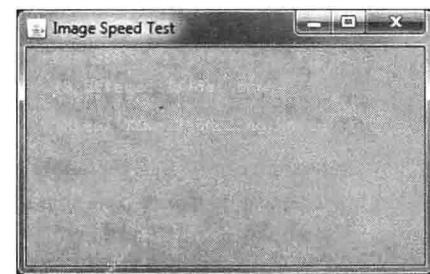


图 10.9 图像速度测试



```
package javagames.images;

import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.image.*;
import java.util.Random;
import javagames.util.SimpleFramework;

public class ImageSpeedTest extends SimpleFramework {
    private Random rand = new Random();
    private GraphicsConfiguration gc;
    private BufferedImage bi;
    private VolatileImage vi;

    private boolean realtime = true;
    private boolean bufferedImage = true;
    public ImageSpeedTest() {
        appWidth = 640;
        appHeight = 640;
        appSleep = 0L;
        appTitle = "Image Speed Test";
    }
    @Override
    protected void initialize() {
        super.initialize();
        GraphicsEnvironment ge = GraphicsEnvironment
            .getLocalGraphicsEnvironment();
        GraphicsDevice gd = ge.getDefaultScreenDevice();
        gc = gd.getDefaultConfiguration();
        bi = gc.createCompatibleImage( appWidth, appHeight );
        createVolatileImage();
        renderToBufferedImage();
    }
    @Override
    protected void processInput( float delta ) {
        super.processInput( delta );
        if( keyboard.keyDownOnce( KeyEvent.VK_B ) ) {
            bufferedImage = !bufferedImage;
        }
        if( keyboard.keyDownOnce( KeyEvent.VK_R ) ) {
            realtime = !realtime;
        }
    }
}
```



## 10.4 使用 VolatileImage 提高速度



```
private void createVolatileImage() {
    if( vi != null ) {
        vi.flush();
        vi = null;
    }
    vi = gc.createCompatibleVolatileImage(getWidth(), getHeight());
}

@Override
protected void render( Graphics g ) {
    if( bufferedImage ) {
        renderToBufferedImage( g );
    } else if( realtime ) {
        renderToVolatileImageEveryFrame( g );
    } else {
        renderToVolatileImage( g );
    }
    super.render( g );
    // spit out help
    g.drawString( "(B)uffered Image: " + bufferedImage, 20, 35 );
    g.drawString( "(R)eal Time Rendering: " + realtime, 20, 65 );
}

private void renderToVolatileImage( Graphics g ) {
    do {
        int returnCode = vi.validate( gc );
        if( returnCode == VolatileImage.IMAGE_RESTORED ) {
            // Contents need to be restored
            renderVolatileImage();
        } else if( returnCode == VolatileImage.IMAGE_INCOMPATIBLE ) {
            // incompatible GraphicsConfig
            createVolatileImage();
            renderVolatileImage();
        }
        g.drawImage( vi, 0, 0, null );
    } while( vi.contentsLost() );
}

private void renderToVolatileImageEveryFrame( Graphics g ) {
    do {
        int returnCode = vi.validate( gc );
        if( returnCode == VolatileImage.IMAGE_INCOMPATIBLE ) {
            // incompatible GraphicsConfig
            createVolatileImage();
        }
    }
```



```
    renderVolatileImage();
    g.drawImage( vi, 0, 0, null );
} while( vi.contentsLost() );
}

protected void renderVolatileImage() {
    Graphics2D g2d = vi.createGraphics();
    g2d.setColor( new Color( rand.nextInt() ) );
    g2d.fillRect( 0, 0, getWidth(), getHeight() );
    g2d.dispose();
}

private void renderToBufferedImage( Graphics g ) {
    if( realtime ) {
        renderToBufferedImage();
    }
    g.drawImage( bi, 0, 0, null );
}

private void renderToBufferedImage() {
    Graphics2D g2d = bi.createGraphics();
    g2d.setColor( new Color( rand.nextInt() ) );
    g2d.fillRect( 0, 0, getWidth(), getHeight() );
    g2d.dispose();
}

public static void main( String[] args ) {
    launchApp( new ImageSpeedTest() );
}
}
```

## 10.5 创建透明图像

TransparentImageExample 如图 10.10 所示，位于 javagames.images 包中，它创建了带有透明像素的一幅图像。在此之前，还没有任何图像带有透明的像素。对于任何类型的、非方形的游戏精灵（精灵是一个通用性的术语，表示绘制到屏幕上的任何图像文件），围绕边缘的透明像素都是必要的，因为这使得没有围绕游戏对象的边框。

如果所需要做的只是创建带有透明背景的一幅图像，那也太容易了。一个新创建的 BufferedImage，带有一个支持透明度的类型，其默认值为 0，这意味着整幅图像都是透明的。

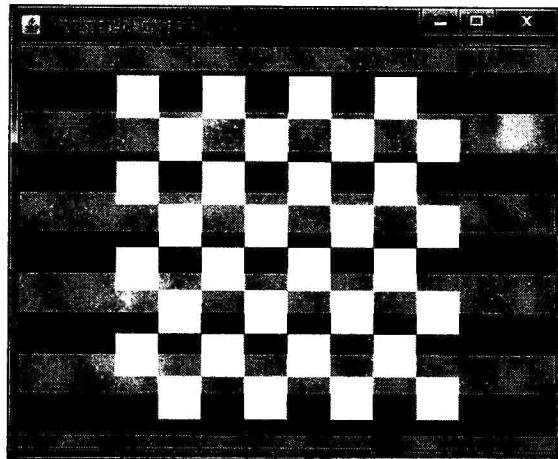


图 10.10 透明图像示例

```
BufferedImage img =  
    new BufferedImage( 256, 256, BufferedImage.TYPE_INT_ARGB );
```

前面代码段中创建的图像，包括 alpha 值，默认为一个透明的背景。TransparentImage Example 代码在 initialize()方法中创建了一个新的 BufferedImage。即便每个方形都绘制成为白色，其他的方形还是透明的。

updateObjects()方法将绘制区域分为 5 个部分。其中的每一个部分，都通过用一个亮灰色的条覆盖住另一条的一半，从而创建一种阴影效果。这些条使用增量值来移动，以使得这些条移动向屏幕的底部。一旦移动值比条的高度还要高，将会重置它。

render()方法将高度分成块，并且从屏幕顶端的一条开始，渲染每一条，因此，这些条从屏幕的顶端开始进入屏幕。最后，透明的图像绘制到移动的条之上。由于棋盘图像的背景是透明的，因此可以看到图像下面的背景。

```
package javagames.images;  
import java.awt.*;  
import java.awt.image.BufferedImage;  
import javagames.util.SimpleFramework;  
  
public class TransparentImageExample extends SimpleFramework {  
    private BufferedImage img;  
    private float shift;  
    public TransparentImageExample() {  
        appWidth = 400;  
        appHeight = 300;  
        appSleep = 10L;
```



```
appTitle = "Transparent Image Example";
appBackground = Color.DARK_GRAY;
}
@Override
protected void initialize() {
    super.initialize();
    img = new BufferedImage( 256, 256, BufferedImage.TYPE_INT_ARGB );
    Graphics2D g2d = img.createGraphics();
    int w = 8;
    int h = 8;
    int dx = img.getWidth() / w;
    int dy = img.getHeight() / h;
    for( int i = 0; i < w; ++i ) {
        for( int j = 0; j < h; ++j ) {
            if( (i+j) % 2 == 0 ) {
                g2d.setColor( Color.WHITE );
                g2d.fillRect( i*dx, j*dy, dx, dy );
            }
        }
    }
    g2d.dispose();
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    int ribbonHeight = canvas.getHeight() / 5;
    shift += delta * ribbonHeight;
    if( shift > ribbonHeight ) {
        shift -= ribbonHeight;
    }
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    // draw shifting background
    int hx = canvas.getHeight() / 5;
    g.setColor( Color.LIGHT_GRAY );
    for( int i = -1; i < 5; ++i ) {
        g.fillRect( 0, (int)shift + hx * i, canvas.getWidth(), hx / 2 );
    }
    int x = ( canvas.getWidth() - img.getWidth() ) / 2;
    int y = ( canvas.getHeight() - img.getHeight() ) / 2;
```



```
        g.drawImage( img, x, y, null );
    }
    public static void main( String[] args ) {
        launchApp( new TransparentImageExample() );
    }
}
```

## 10.6 使用 alpha 合成的规则

在前面的示例中，透明的图像绘制到了背景的上面，而下面的背景是可以看到的。但这并非唯一的选择。当源图像和目标图像中的像素具有某种透明度的时候，有很多方法来混合它们。AlphaComposite 包括了很多规则。



Java Docs( 参见 Class AlphaComposite, 2013, <http://docs.oracle.com/javase/7/docs/api/java.awt/AlphaComposite.html> ) 很好地介绍了相关的数学知识。“Compositing Digital Images”这篇论文介绍得甚至更为详细 (参见 T. Porter and T. Duff, “Compositing Digital Images,” SIGGRAPH, 1984, pp. 253–259; <http://keithp.com/~keithp/porterduff/p253-porter.pdf>)

不同的混合操作包括以下内容。

- SRC——将源色复制到目标色。
- DEST——目标色未修改。
- SRC\_IN——只复制在目标色中的源色。
- DEST\_IN——只复制在源色中的目标色。
- SRC\_OUT——目标色只被外部的源色所替代。
- DEST\_OUT——只复制目标色的外部部分。
- SRC\_OVER——源色复制到目标色之上。
- DEST\_OVER——目标色复制到源色之上。
- SRC\_ATOP——目标色内部的源色，复制到目标色之上。
- DEST\_ATOP——源色内部的目标色，复制到源色之上。
- XOR——复制源色和目标色彼此之外的那部分。
- CLEAR——目标色和目标 alpha 值都被清除。

并不是每个源图像和目标图像都有一个 alpha 值, Java 添加了一个额外的 alpha 组合可应用于源 alpha。

比较复杂的事情之一是理解这一点: 对使用一个图形对象绘制的形状应用 AlphaComposite 规则, 与绘制一幅完整的图像是不同的。当使用形状的时候, 只有形状内部的像素遵从组合规则, 而图像则会使用整个区域。



当初次尝试获取与 Porter 和 Duff 的论文一致的 alpha 组合结果时, 我无法得到相同的结果, 因为我使用了一个三角形的形状而非复制的完整图像。作为一个练习, 请以使用各种形状修改该示例, 并且注意区别。

理解这些不同选项如何工作的最好的方法是, 使用它们并看看结果。尽管阅读方程式从理论的角度来看是不错的, 但是, 一旦你看到了结果, 方程式就变得更加直观了。图 10.11 所示的 AlphaCompositeExample, 位于 javagames.images 包中, 其大多数代码都与显示和修改所有不同的选项有关。使用 AlphaComposite 的代码位于 createImages()方法中。

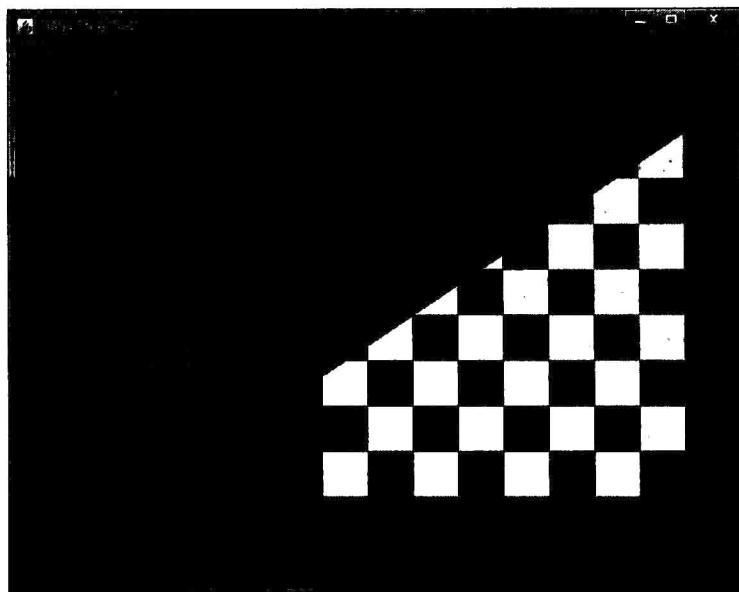


图 10.11 Alpha 合成示例

第一步是使用 AlphaComposite.CLEAR 规则从源图像中清除像素, 以便整个图像只有透明的像素。尽管在每一帧创建一个全新的图像也有效, 但清除像素更快。

```
Graphics2D g2d = sourceImage.createGraphics();
g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.CLEAR));
g2d.fillRect(0, 0, sourceImage.getWidth(), sourceImage.getHeight());
```



接下来，使用默认的组合规则 `AlphaComposite.SRC_OVER` 来绘制黄色的三角形。注意，该颜色针对 `alpha` 值使用 `srcAlpha` 变量。

```
g2d.setComposite(  
    AlphaComposite.getInstance(AlphaComposite.SRC_OVER));  
Polygon p = new Polygon();  
p.addPoint(0, 0);  
p.addPoint(sourceImage.getWidth(), 0);  
p.addPoint(  
    sourceImage.getWidth(), (int)(sourceImage.getHeight() / 1.5));  
g2d.setColor(new Color(1.0f, 1.0f, 0.0f, srcAlpha));  
g2d.fill(p);
```

接下来，使用同样的过程来清除目标图像。

```
g2d = destinationImage.createGraphics();  
g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.CLEAR));  
g2d.fillRect(  
    0, 0, destinationImage.getWidth(), destinationImage.getHeight());
```

接下来，在相对的一边绘制一个蓝色的三角形。注意，该颜色使用了 `dstAlpha` 值。

```
g2d.setComposite(  
    AlphaComposite.getInstance(AlphaComposite.SRC_OVER));  
p = new Polygon();  
p.addPoint(0, 0);  
p.addPoint(destinationImage.getWidth(), 0);  
p.addPoint(0, (int)(destinationImage.getHeight() / 1.5));  
g2d.setColor(new Color(0.0f, 0.0f, 1.0f, dstAlpha));  
g2d.fill(p);
```

接下来，源图像绘制到了目标图像的上面。用户所选择的 `AlphaComposite` 规则，就是在这里应用于两个图像之上。

```
int rule = compositeRule[compositeIndex];  
g2d.setComposite(AlphaComposite.getInstance(rule, extAlpha));  
g2d.drawImage(sourceImage, 0, 0, null);
```

最后一步是用一个棋盘重新绘制背景图像，清除最后一帧的结果，然后在上面绘制 `alpha` 组合图像。使用棋盘对于看到混合效果是有帮助的。

```
g2d = sprite.createGraphics();  
int dx = (sprite.getWidth()) / 8;  
int dy = (sprite.getHeight()) / 8;  
for(int i = 0; i < 8; ++i) {
```



```
    for( int j = 0; j < 8; ++j ) {
        g2d.setColor( (i+j) % 2 == 0 ? Color.BLACK : Color.WHITE );
        g2d.fillRect( i*dx, j*dy, dx, dy );
    }
}
g2d.drawImage( destinationImage, 0, 0, null );
```

processInput()方法使用箭头键选择不同的组合规则。A 和 Q 键控制着源 alpha 值，S 和 W 键控制着目标 alpha 值，D 和 E 键控制着额外的 alpha 值。

render()绘制了不同的组合规则，并且将当前选择的规则颜色设置为红色。源 alpha、目标 alpha 和额外的 alpha 值都使用其键值绘制，这些键值用于它们的自增或自减。最后，渲染精灵图像，以显示出 alpha 组合的结果。

```
package javagames.images;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import javagames.util.SimpleFramework;

public class AlphaCompositeExample extends SimpleFramework {
    private String[] compositeName = {
        "SRC",
        "DST",
        "SRC_IN",
        "DST_IN",
        "SRC_OUT",
        "DST_OUT",
        "SRC_OVER",
        "DST_OVER",
        "SRC_ATOP",
        "DST_ATOP",
        "XOR",
        "CLEAR",
    };
    private int[] compositeRule = {
        AlphaComposite.SRC,
        AlphaComposite.DST,
        AlphaComposite.SRC_IN,
        AlphaComposite.DST_IN,
        AlphaComposite.SRC_OUT,
        AlphaComposite.DST_OUT,
        AlphaComposite.SRC_OVER,
```



```
AlphaComposite.DST_OVER,
AlphaComposite.SRC_ATOP,
AlphaComposite.DST_ATOP,
AlphaComposite.XOR,
AlphaComposite.CLEAR,
};

private int compositeIndex;
private float srcAlpha;
private float dstAlpha;
private float extAlpha;
private BufferedImage sprite;
private BufferedImage sourceImage;
private BufferedImage destinationImage;
public AlphaCompositeExample() {
    appBackground = Color.DARK_GRAY;
    appWidth = 640;
    appHeight = 480;
    appSleep = 10L;
    appTitle = "Alpha Composite Example";
}
@Override
protected void initialize() {
    super.initialize();
    srcAlpha = 1.0f;
    dstAlpha = 1.0f;
    extAlpha = 1.0f;
    destinationImage =
        new BufferedImage( 320, 320, BufferedImage.TYPE_INT_ARGB );
    sourceImage =
        new BufferedImage( 320, 320, BufferedImage.TYPE_INT_ARGB );
    sprite = new BufferedImage( 320, 320, BufferedImage.TYPE_INT_ARGB );
    createImages();
}
private void createImages() {
    // source image
    Graphics2D g2d = sourceImage.createGraphics();
    g2d.setComposite( AlphaComposite.getInstance( AlphaComposite.CLEAR ) );
    g2d.fillRect( 0, 0, sourceImage.getWidth(), sourceImage.getHeight() );
    g2d.setComposite(
        AlphaComposite.getInstance( AlphaComposite.SRC_OVER ) );
    Polygon p = new Polygon();
    p.addPoint( 0, 0 );
    p.addPoint( sourceImage.getWidth(), 0 );
    p.addPoint(
```



```
        sourceImage.getWidth(), (int)(sourceImage.getHeight() / 1.5));
g2d.setColor( new Color( 1.0f, 1.0f, 0.0f, srcAlpha ) );
g2d.fill( p );
g2d.dispose();
// destination image
g2d = destinationImage.createGraphics();
g2d.setComposite( AlphaComposite.getInstance( AlphaComposite.CLEAR ) );
g2d.fillRect(
    0, 0, destinationImage.getWidth(), destinationImage.getHeight() );
g2d.setComposite(
    AlphaComposite.getInstance( AlphaComposite.SRC_OVER ) );
p = new Polygon();
p.addPoint( 0, 0 );
p.addPoint( destinationImage.getWidth(), 0 );
p.addPoint( 0, (int)(destinationImage.getHeight() / 1.5) );
g2d.setColor( new Color( 0.0f, 0.0f, 1.0f, dstAlpha ) );
g2d.fill( p );
int rule = compositeRule[ compositeIndex ];
g2d.setComposite(AlphaComposite.getInstance( rule, extAlpha ) );
g2d.drawImage( sourceImage, 0, 0, null );
g2d.dispose();
// checkerboard background
g2d = sprite.createGraphics();
int dx = (sprite.getWidth()) / 8;
int dy = (sprite.getHeight()) / 8;
for( int i = 0; i < 8; ++i ) {
    for( int j = 0; j < 8; ++j ) {
        g2d.setColor( (i+j) % 2 == 0 ? Color.BLACK : Color.WHITE );
        g2d.fillRect( i*dx, j*dy, dx, dy );
    }
}
g2d.drawImage( destinationImage, 0, 0, null );
g2d.dispose();
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    if( keyboard_KeyDownOnce( KeyEvent.VK_UP ) ) {
        compositeIndex--;
        if( compositeIndex < 0 ) {
            compositeIndex = compositeRule.length - 1;
        }
    }
    if( keyboard_KeyDownOnce( KeyEvent.VK_DOWN ) ) {
```



```
compositeIndex++;
if( compositeIndex > compositeRule.length - 1 ) {
    compositeIndex = 0;
}
}
if( keyboard.keyDown( KeyEvent.VK_A ) ) {
    srcAlpha = dec( srcAlpha, delta );
}
if( keyboard.keyDown( KeyEvent.VK_Q ) ) {
    srcAlpha = inc( srcAlpha, delta );
}
if( keyboard.keyDown( KeyEvent.VK_S ) ) {
    dstAlpha = dec( dstAlpha, delta );
}
if( keyboard.keyDown( KeyEvent.VK_W ) ) {
    dstAlpha = inc( dstAlpha, delta );
}
if( keyboard.keyDown( KeyEvent.VK_D ) ) {
    extAlpha = dec( extAlpha, delta );
}
if( keyboard.keyDown( KeyEvent.VK_E ) ) {
    extAlpha = inc( extAlpha, delta );
}
createImages();
}
private float inc( float val, float delta ) {
    val += 0.5f * delta;
    if( val > 1.0f ) {
        val = 1.0f;
    }
    return val;
}
private float dec( float val, float delta ) {
    val -= 0.5f * delta;
    if( val < 0.0f ) {
        val = 0.0f;
    }
    return val;
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    Graphics2D g2d = (Graphics2D)g;
    int xPos = 20;
```



```
int yPos = 35;
g2d.drawString( "", xPos, yPos );
yPos += 15;
g2d.drawString( "UP/DOWN Arrow to select", xPos, yPos );
yPos += 15;
g2d.drawString( "", xPos, yPos );
yPos += 15;
for( int i = 0; i < compositeName.length; ++i ) {
    if( i == compositeIndex ) {
        g2d.setColor( Color.RED );
    } else {
        g2d.setColor( Color.GREEN );
    }
    g2d.drawString( compositeName[i], xPos, yPos );
    yPos += 15;
}
g2d.drawString( "", xPos, yPos );
yPos += 15;
g2d.setColor( Color.GREEN );
g2d.drawString(
    String.format( "Q | A : SRC_ALPHA=%.4f", srcAlpha ), xPos, yPos );
yPos += 15;
g2d.drawString(
    String.format( "W | S : DST_ALPHA=%.4f", dstAlpha ), xPos, yPos );
yPos += 15;
g2d.drawString(
    String.format( "E | D : EXT_ALPHA=%.4f", extAlpha ), xPos, yPos );
yPos += 15;
int x = (canvas.getWidth() - destinationImage.getWidth() - 50);
int y = (canvas.getHeight() - destinationImage.getHeight()) / 2;
g2d.drawImage( sprite, x, y, null );
}
public static void main( String[] args ) {
    launchApp( new AlphaCompositeExample() );
}
}
```

## 10.7 绘制精灵

到现在，本章已经介绍了加载、保存、创建和使用缓冲图像。尽管理解 BufferedImage

如何用于从头创建图形很重要，但制作游戏最常用的方法是，加载和绘制精灵，而精灵是通过一个绘图程序创建并保存为一个文件的。

为了进行概念测试，我们将使用前面示例中所使用的相同的棋盘，而不是真的创建精灵并加载它们以供使用。这个图像不仅容易创建，而且还将通过绘制旋转图像来强调某些问题。

就像前面的大多数示例一样，这个示例的代码在本节后面给出，以允许你了解所有不同设置的组合。在屏幕上绘制精灵时，总是要在速度和质量之前求得平衡。较好的图像，看上去有更好的体验，但如果帧速率变得太慢，游戏有多好看都无济于事。为了求得平衡，你需要查看所有不同的属性以及绘制精灵的方法。

能够影响一幅图像的速度和质量的第一个属性是抗锯齿渲染提示。当直线旋转得看上去显得模糊时，就会发生图像的锯齿现象。抗锯齿是将像素与背景混合，以使其平滑的一个过程。但是，这个过程需要消耗时间，而且会降低渲染的速度。

```
Key: RenderingHints.KEY_ANTIALIASING
Antialiasing On: RenderingHints.VALUE_ANTIALIAS_ON
Antialiasing Off: RenderingHints.VALUE_ANTIALIAS_OFF
```

能够影响到渲染图像的下一个属性是插值算法。这些算法控制了当图像旋转或缩放时，图像的颜色如何进行插值。Java 所使用的 3 个主要的插值方法如下。

- 最近邻法——找到最接近的值并使用它，不用考虑任何其他周围的值。
- 双线性插值——类似于颜色插值代码示例中使用的插值法。
- 双三次插值——类似于双线性插值法，但是使用 16 个邻近值而不是双线性插值所使用的 4 个。

最近邻法是最快的，而双三次插值的质量最好。

```
Key: RenderingHints.KEY_INTERPOLATION
Nearest Neighbor: RenderingHints.VALUE_INTERPOLATION_NEAREST_NEIGHBOR
Bilinear: RenderingHints.VALUE_INTERPOLATION_BILINEAR
Bicubic: RenderingHints.VALUE_INTERPOLATION_BICUBIC
```

这些属性和将图像旋绕到屏幕的不同方法组合起来，产生了良莠不齐、快慢不一的各种结果。渲染图像的 3 个方法是 `Affine Transform`、`Affine Transform Op` 和 `Texture Paint`。不管哪个方法，它们都以 `AffineTransform` 为起点。



还记得吧，`AffineTransform` 用一个列主序的矩阵来表示，因此，该矩阵的连接是从右向左的。参阅本书第 3 章中行主序矩阵和列主序矩阵部分以进行回顾。



Final = 4<sup>th</sup> \* 3<sup>rd</sup> \* 2<sup>nd</sup> \* 1<sup>st</sup>

给定一个 `AffineTransform`, 第一种方法使用 `Graphics2D` 对象的 `drawImage` 方法。

```
AffineTransform transform = ...;
g2d.drawImage( sourceImage, transform, null );
```

第二种方法使用 `AffineTransformOp` 类。首先, 使用 `AffineTransform` 创建 `AffineTransformOp`, 并且实现插值算法类型。然后, 使用过滤器方法变换图像。

```
AffineTransform transform = ...;
AffineTransformOp op =
    new AffineTransformOp( transform, AffineTransformOp.TYPE_BILINEAR );
BufferedImage transformedImage = op.filter( sourceImage, null );
g2d.drawImage( transformedImage, 0, 0, null );
```

第三种方法使用类来绘制精灵。由于该方法通常直接将变换设置为图形对象, 因此当完成渲染时必须清除它。

```
AffineTransform transform = ...;
g2d.setTransform( transform );
Rectangle2D anchor = new Rectangle2D.Float(
    0, 0, sprite.getWidth(), sprite.getHeight() );
TexturePaint paint = new TexturePaint( sprite, anchor );
g2d.setPaint( paint );
g2d.fillRect( 0, 0, sprite.getWidth(), sprite.getHeight() );
// reset!!!
g2d.setTransform( new AffineTransform() );
```

尽管有些方法比其他方法好, 但 `AffineTransform` 和 `TexturePaint` 方法在渲染图像边缘时, 都遇到了麻烦。`AffineTransform` 使用一个双线性插值, 在渲染图像的内部的时候是正确的, 但是无法渲染边缘。`TexturePaint` 方法通过折返纹理来插值边缘, 这会导致诸如将图像从一端折返到另一端的问题。

修复 `TexturePaint` 和 `AffineTransform` 的解决方案, 都是将围绕图像的边缘像素保留为透明的。`AffineTransform` 混合了清除像素, 因此, 不平滑的边缘变得可见了;`TexturePaint` 折返了不可见的像素, 也不会导致视觉效果上的问题。

使用该示例程序, 我们测试了所有不同的属性和渲染方法, 以确定哪一个方法能够以最快的速度产生最好的结果。在我们的测试机器上, 两个最好的方法如下。

- 关闭 Antialiasing, Bicubic Interpolation, Texture Paint 带有一个透明边框的渲染。

- 打开 Antialiasing, Bilinear Interpolation, Affine Transform 带有透明边框的渲染。

FlyingSpritesExample 如图 10.12 所示, 位于 javagames.images 包中, 它允许测试不同的属性和渲染方法的速度和质量。initialize()方法为 4 幅图像创建位置、速率、角度和旋转。设置默认的属性并创建测试图像。

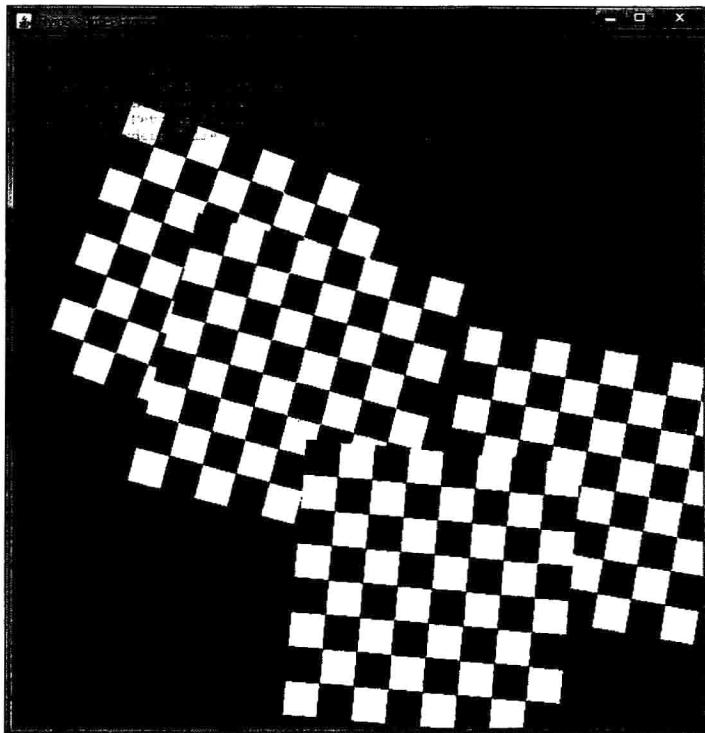


图 10.12 飞翔精灵示例

在渲染属性改变时, createSprite() 方法用来重新创建棋盘图像。该图像在 createCheckerboard() 方法中创建, addTransparentBorder() 方法将围绕图像放置 4 像素的一个边框; drawGreenBorder() 方法添加一个绿色的边框以使得图像可见。

processInput() 方法开关所有不同的属性和渲染方法。A 键开关抗锯齿属性。I 键遍历不同的插值算法。T 键开关围绕图像的透明边框。R 键在 3 种不同的渲染方法之间切换, G 键开关绿色边框。根据需要重新创建精灵。

updateObject() 方法更新 4 个对象中的每一个的位置和旋转。render() 方法显示常用信息以及每个属性的当前状态。设置了抗锯齿提示和插值渲染提示, 并且使用所选择的渲染方法来绘制 4 幅图像。



setAntialiasing()和setInterpolation()方法配置渲染提示。createTransform()围绕每个对象的中心旋转它们，然后平移它以穿过屏幕。createTransformOp()根据所选的插值类来进行变换操作。

doAffineTransform()、doAffineTransformOp()和doTexturePaint()方法使用前面介绍的不用的方法来渲染图像。

```
package javagames.images;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.geom.*;
import java.awt.image.*;
import javagames.util.*;

public class FlyingSpritesExample extends SimpleFramework {
    private static final int IMG_WIDTH = 256;
    private static final int IMG_HEIGHT = 256;
    private enum Interpolation {
        NearestNeighbor,
        BiLinear,
        BiCubic;
    }
    private enum RotationMethod {
        AffineTransform,
        AffineTransformOp,
        TexturePaint;
    }
    private boolean antialiased;
    private boolean transparent;
    private boolean greenBorder;
    private Interpolation interpolation;
    private RotationMethod rotationMethod;
    private BufferedImage sprite;
    private Vector2f[] positions;
    private float[] angles;
    private Vector2f[] velocities;
    private float[] rotations;
    public FlyingSpritesExample() {
        appWidth = 640;
        appHeight = 640;
        appSleep = 0L;
        appTitle = "Flying Sprites Example";
        appBackground = Color.DARK_GRAY;
```



```
}

@Override
protected void initialize() {
    super.initialize();
    positions = new Vector2f[] {
        new Vector2f( -0.15f, 0.3f ),
        new Vector2f( 0.15f, 0.0f ),
        new Vector2f( 0.25f, -0.3f ),
        new Vector2f( -0.25f, -0.6f ),
    };
    velocities = new Vector2f[] {
        new Vector2f( -0.04f, 0.0f ),
        new Vector2f( -0.05f, 0.0f ),
        new Vector2f( 0.06f, 0.0f ),
        new Vector2f( 0.07f, -0.0f ),
    };
    angles = new float[] {
        (float)Math.toRadians( 0 ),
        (float)Math.toRadians( 0 ),
        (float)Math.toRadians( 0 ),
        (float)Math.toRadians( 0 ),
    };
    rotations = new float[] {
        1.0f, 0.75f, 0.5f, 0.25f
    };
    antialiased = false;
    transparent = false;
    greenBorder = false;
    interpolation = Interpolation.NearestNeighbor;
    rotationMethod = RotationMethod.AffineTransform;
    createSprite();
}

@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    if( keyboard.keyDownOnce( KeyEvent.VK_A ) ) {
        antialiased = !antialiased;
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_I ) ) {
        Interpolation[] values = Interpolation.values();
        int index = (interpolation.ordinal() + 1) % values.length;
        interpolation = values[ index ];
    }
}
```



```
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_T ) ) {
        transparent = !transparent;
        createSprite();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_R ) ) {
        RotationMethod[] methods = RotationMethod.values();
        int index = (rotationMethod.ordinal() + 1) % methods.length;
        rotationMethod = methods[ index ];
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_G ) ) {
        greenBorder = !greenBorder;
        createSprite();
    }
}
private void createSprite() {
    createCheckerboard();
    if( transparent ) {
        addTransparentBorder();
    }
    if( greenBorder ) {
        drawGreenBorder();
    }
}
private void createCheckerboard() {
    sprite =
        new BufferedImage( IMG_WIDTH, IMG_HEIGHT,
                           BufferedImage.TYPE_INT_ARGB );
    Graphics2D g2d = sprite.createGraphics();
    int dx = IMG_WIDTH / 8;
    int dy = IMG_HEIGHT / 8;
    for( int i = 0; i < 8; ++i ) {
        for( int j = 0; j < 8; ++j ) {
            if( (i+j) % 2 == 0 ) {
                g2d.setColor( Color.WHITE );
            } else {
                g2d.setColor( Color.BLACK );
            }
            g2d.fillRect( i*dx, j*dy, dx, dy );
        }
    }
    g2d.dispose();
}
```



```
private void addTransparentBorder() {
    int borderWidth = IMG_WIDTH + 8;
    int borderHeight = IMG_HEIGHT + 8;
    BufferedImage newSprite =
        new BufferedImage( borderWidth, borderHeight,
                           BufferedImage.TYPE_INT_ARGB );
    Graphics2D g2d = newSprite.createGraphics();
    g2d.drawImage( sprite, 4, 4, null );
    g2d.dispose();
    sprite = newSprite;
}
private void drawGreenBorder() {
    Graphics2D g2d = sprite.createGraphics();
    g2d.setColor( Color.GREEN );
    g2d.drawRect( 0, 0, sprite.getWidth() - 1, sprite.getHeight() - 1 );
    g2d.dispose();
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    for( int i = 0; i < positions.length; ++i ) {
        positions[i] = positions[i].add( velocities[i].mul( delta ) );
        if( positions[i].x >= 1.0f ) {
            positions[i].x = -1.0f;
        } else if( positions[i].x <= -1.0f ) {
            positions[i].x = 1.0f;
        }
        if( positions[i].y <= -1.0f ) {
            positions[i].y = 1.0f;
        } else if( positions[i].y >= 1.0f ) {
            positions[i].y = -1.0f;
        }
        angles[i] += rotations[i] * delta;
    }
}
@Override
protected void render( Graphics g ) {
    Graphics2D g2d = (Graphics2D)g;
    setAntialiasing( g2d );
    setInterpolation( g2d );
    switch( rotationMethod ) {
        case AffineTransform: doAffineTransform( g2d ); break;
        case AffineTransformOp: doAffineTransformOp( g2d ); break;
    }
}
```



```
        case TexturePaint: doTexturePaint( g2d ); break;
    }
    super.render( g );
    g.drawString( "(A)ntialiased: " + antialiased, 20, 35 );
    g.drawString( "(I)nterpolation: " + interpolation, 20, 50 );
    g.drawString( "(T)ransparent Border: " + transparent, 20, 65 );
    g.drawString( "(R)otation Method: " + rotationMethod, 20, 80 );
    g.drawString( "(G)reen Border: " + greenBorder, 20, 95 );
}
private void setAntialiasing( Graphics2D g2d ) {
    if( antialiased ) {
        g2d.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON
        );
    } else {
        g2d.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_OFF
        );
    }
}
private void setInterpolation( Graphics2D g2d ) {
    if( interpolation == Interpolation.NearestNeighbor ) {
        g2d.setRenderingHint(
            RenderingHints.KEY_INTERPOLATION,
            RenderingHints.VALUE_INTERPOLATION_NEAREST_NEIGHBOR
        );
    } else if( interpolation == Interpolation.BiLinear ) {
        g2d.setRenderingHint(
            RenderingHints.KEY_INTERPOLATION,
            RenderingHints.VALUE_INTERPOLATION_BILINEAR
        );
    } else if( interpolation == Interpolation.BiCubic ) {
        g2d.setRenderingHint(
            RenderingHints.KEY_INTERPOLATION,
            RenderingHints.VALUE_INTERPOLATION_BICUBIC
        );
    }
}
private AffineTransform createTransform( Vector2f position, float angle ) {
    Vector2f screen = getViewportTransform().mul( position );
    AffineTransform transform =

```



```
AffineTransform.getTranslateInstance( screen.x, screen.y );
transform.rotate( angle );
transform.translate( -sprite.getWidth() / 2, -sprite.getHeight() / 2 );
return transform;
}
private void doAffineTransform( Graphics2D g2d ) {
for( int i = 0; i < positions.length; ++i ) {
AffineTransform tranform =
createTransform( positions[i], angles[i] );
g2d.drawImage( sprite, tranform, null );
}
}
private AffineTransformOp createTransformOp(
Vector2f position, float angle ) {
AffineTransform transform = createTransform( position, angle );
if( interpolation == Interpolation.NearestNeighbor ) {
return new AffineTransformOp(
transform, AffineTransformOp.TYPE_NEAREST_NEIGHBOR );
} else if( interpolation == Interpolation.BiLinear ) {
return new AffineTransformOp(
transform, AffineTransformOp.TYPE_BILINEAR );
} else { // interpolation == Interpolation.BiCubic
return new AffineTransformOp(
transform, AffineTransformOp.TYPE_BICUBIC );
}
}
private void doAffineTransformOp( Graphics2D g2d ) {
for( int i = 0; i < positions.length; ++i ) {
AffineTransformOp op = createTransformOp( positions[i], angles[i] );
g2d.drawImage( op.filter( sprite, null ), 0, 0, null );
}
}
private void doTexturePaint( Graphics2D g2d ) {
for( int i = 0; i < positions.length; ++i ) {
Rectangle2D anchor =
new Rectangle2D.Float(
0, 0, sprite.getWidth(), sprite.getHeight() );
TexturePaint paint = new TexturePaint( sprite, anchor );
g2d.setPaint( paint );
AffineTransform transform =
createTransform( positions[i], angles[i] );
g2d.setTransform( transform );
g2d.fillRect( 0, 0, sprite.getWidth(), sprite.getHeight() );
}
```



```
    }
    // very important!!!
    g2d.setTransform( new AffineTransform() );
}
public static void main( String[] args ) {
    launchApp( new FlyingSpritesExample() );
}
}
```

## 10.8 探索不同的缩放算法

旋转图像不仅用于需要平衡速度和质量的地方。根据图像的最初大小缩放它们，如果操作不当的话，也会引发问题。由于有很多不同的方法来缩放图像，搞清楚使用哪种方法对游戏来说足够快，这也是很难的。缩放图像的第一种方法是，使用如下的代码：

```
BufferedImage imgToScale;
Image scaledImg = imgToScale.getScaledInstance(
    200, 200, Image.SCALE_AREA_AVERAGING
);
```

该方法中，只有两个渲染提示可用：

```
Image.SCALE_AREA_AVERAGING
Image.SCALE_REPLICATE
```

缩放图像的第二种方法是使用 `Graphics` 对象：

```
BufferedImage toScale;
BufferedImage destImg;
Graphics2D g2d = destImg.createGraphics();
g2d.setRenderingHint( key, value );
g2d.drawImage(
    toScale, 0, 0, destImg.getWidth(), destImg.getHeight(), null );
g2d.dispose();
```

这两种方法都有问题，当初次缩放图像时，问题还不明显。第一种方法使用 `getScaledInstance()`，这很慢，并且只返回一个 `Image` 而不是一个 `BufferedImage`。采用第二种方法，当把图像缩放到小于其原来的一半的时候，不管使用何种插值方法，效果都很差。

第三种可选的方法是，使用第二种方法逐步缩小图像，每次都只是缩放到原来大小的



一半，直到图像缩放到合适的大小。尽管这种方法较慢，但它通常还是比 `getScaledInstance()` 方法快，质量上的损失也很小。

```
BufferedImage ret = sprite;
int targetWidth = sprite.getWidth() / 4;
int targetHeight = sprite.getHeight() / 4;
int w = sprite.getWidth();
int h = sprite.getHeight();
do {
    w = w / 2;
    if( w < targetWidth ) {
        w = targetWidth;
    }
    h = h / 2;
    if( h < targetHeight ) {
        h = targetHeight;
    }
    BufferedImage tmp =
        new BufferedImage( w, h, BufferedImage.TYPE_INT_ARGB );
    Graphics2D g2d = tmp.createGraphics();
    g2d.setRenderingHint( RenderingHints.KEY_INTERPOLATION, hintValue );
    g2d.drawImage( ret, 0, 0, w, h, null );
    g2d.dispose();
    ret = tmp;
} while( w != targetWidth || h != targetHeight );
```

在试图测量缩放图像所需的时间的时候，会遇到的一个问题：`getScaledInstance()` 方法真的很快，但是当在游戏框架代码中缩放图像进行测试时，帧速率并不是根据度量的渲染时间进行匹配的。度量的时间与帧速率时间不一致的原因在于，`getScaledInstance()` 返回的图像是异步加载的。这也是避免使用该方法的另一个原因。要迫使返回的图像对象完成渲染，我们像下面这样使用 `ImageConsumer`：

```
private void generateAveragedInstance() {
    averaged = sprite.getScaledInstance(
        sprite.getWidth() / 4, sprite.getHeight() / 4,
        Image.SCALE_AREA_AVERAGING
    );
    averaged.getSource().startProduction( getConsumer() );
}
private ImageConsumer getConsumer() {
    return new ImageConsumer() {
        public void setProperties( Hashtable<?, ?> props ) { }
```

```
public void setPixels( int x, int y, int w, int h, ColorModel model,
    int[] pixels, int off, int scansize ) { }
public void setPixels( int x, int y, int w, int h, ColorModel model,
    byte[] pixels, int off, int scansize ) { }
public void setHints( int hintflags ) { }
public void setDimensions( int width, int height ) { }
public void setColorModel( ColorModel model ) { }
public void imageComplete( int status ) { }
};

}
```

ScaleImageExample 如图 10.13 所示，位于 javagames.images 包中，它包含了很多设置代码，用于度量图像缩放的 6 种不同方法的渲染时间，并且创建测试的图像。用不同的方法将一幅图像缩放到较小的尺寸，这可以很好地看到质量和速度上的差异。感谢 Chris Campbell 的精彩文章以及示例图像的思路。

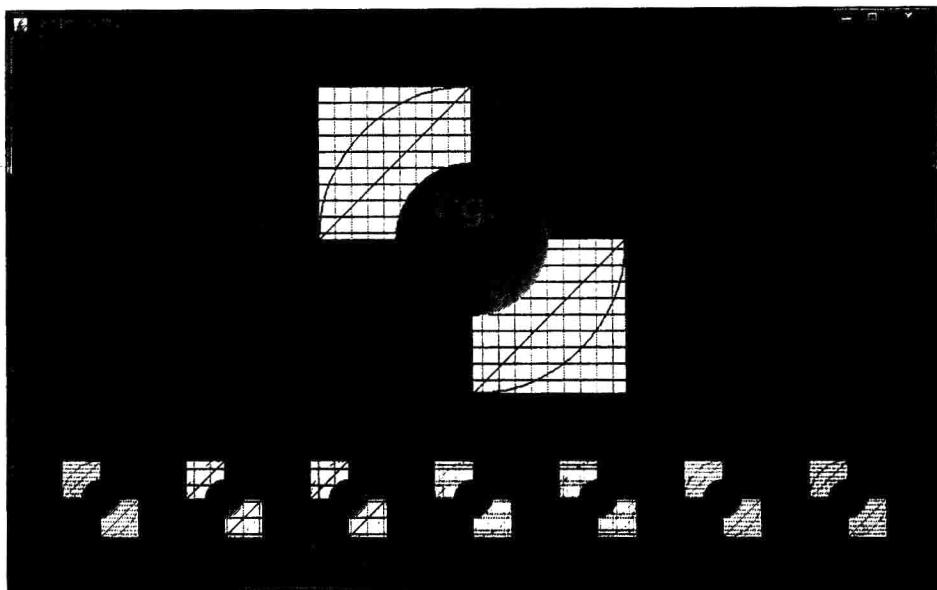


图 10.13 缩放图像示例



要了解详细信息，参见 Campbell, Chris, “The Perils of Image.getScaledInstance(),”  
2007.  
<https://today.java.net/pub/a/today/2007/04/03/perils-of-image-getscaledinstance.html>.

方法创建了每一幅图像的 100 个副本，测量渲染它们所需的时间，然后将该时间除以 100 得到一个平均渲染时间。这些时间并不完全精确，并且将随着每次程序的运行而变化，



但是它们足以用来判定哪一个方法比其他方法更快或者更慢。

```
long start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    generateAveragedInstance();
}
long end = System.nanoTime();
averagedSpeed = (end - start) / 1.0E6; // convert to milliseconds
averagedSpeed /= 100;
```

`createTestImage()`方法负责创建在示例中使用的测试图像。线条间距专门用来突出显示某些缩放方法的问题所在。这在缩放结果中很明显，其中漏掉了一些垂直或水平的线条。

`render()`方法绘制了最初的图像和所有缩放的版本，带上了标签描述渲染提示以及缩放图像所有用的毫秒数。`generateAveragedInstance()`和 `generateNearestNeighbor()`方法使用 `Image` 类中 `getScaledInstance()` 方法来缩放图像。`scaleWithGraphics()` 方法接受一个渲染提示，并且使用 `Graphics` 对象来缩放图像。最后，`scaleDownImage()` 方法逐步实现缩放的结果。



`scaledDownImage()`方法并不是缩放图像的通用方法。缩放大小直接编码为四分之一( $\frac{1}{4}$ )，并且如果图像缩放到一个更大的比例，`do/while` 循环将不再工作。下一节将会介绍这些。

```
package javagames.images;
import java.awt.*;
import java.awt.image.*;
import java.util.Hashtable;
import javagames.util.SimpleFramework;

public class ScaleImageExample extends SimpleFramework {
    private static final int IMG_WIDTH = 320;
    private static final int IMG_HEIGHT = 320;
    private BufferedImage sprite;
    private Image averaged;
    private double averagedSpeed;
    private Image nearestNeighbor;
    private double nearestSpeed;
    private BufferedImage nearest2;
    private double nearest2Speed;
    private BufferedImage bilinear;
    private double bilinearSpeed;
    private BufferedImage bicubic;
    private double bicubicSpeed;
    private BufferedImage stepDownBilinear;
```



```
private double stepDownBilinearSpeed;
private BufferedImage stepDownBicubic;
private double stepDownBicubicSpeed;
public ScaleImageExample() {
    appWidth = 960;
    appHeight = 570;
    appBackground = Color.DARK_GRAY;
    appSleep = 1L;
    appTitle = "Scale Image Example";
}
@Override
protected void initialize() {
    super.initialize();
    System.out.println( "Creating test image..." );
    createTestImage();
    System.out.println( "Generating Averaged Image" );
    long start = System.nanoTime();
    for( int i = 0; i < 100; ++i ) {
        generateAveragedInstance();
    }
    long end = System.nanoTime();
    averagedSpeed = (end - start) / 1.0E6;
    averagedSpeed /= 100;
    System.out.println( "Generating Nearest Neighbor Image" );
    start = System.nanoTime();
    for( int i = 0; i < 100; ++i ) {
        generateNearestNeighbor();
    }
    end = System.nanoTime();
    nearestSpeed = (end - start) / 1.0E6;
    nearestSpeed /= 100;
    System.out.println( "Generating Nearest Neighbor 2" );
    start = System.nanoTime();
    for( int i = 0; i < 100; ++i ) {
        nearest2 = scaleWithGraphics(
            RenderingHints.VALUE_INTERPOLATION_NEAREST_NEIGHBOR );
    }
    end = System.nanoTime();
    nearest2Speed = (end - start) / 1.0E6;
    nearest2Speed /= 100;
    System.out.println( "Generating Bilinear" );
    start = System.nanoTime();
    for( int i = 0; i < 100; ++i ) {
```



```
bilinear = scaleWithGraphics(
    RenderingHints.VALUE_INTERPOLATION_BILINEAR );
}

end = System.nanoTime();
bilinearSpeed = (end - start) / 1.0E6;
bilinearSpeed /= 100;
System.out.println( "Generating Bicubic" );
start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    bicubic = scaleWithGraphics(
        RenderingHints.VALUE_INTERPOLATION_BICUBIC );
}
end = System.nanoTime();
bicubicSpeed = (end - start) / 1.0E6;
bicubicSpeed /= 100;
System.out.println( "Generating Step Down Bilinear" );
start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    stepDownBilinear = scaleDownImage(
        RenderingHints.VALUE_INTERPOLATION_BILINEAR );
}
end = System.nanoTime();
stepDownBilinearSpeed = (end - start) / 1.0E6;
stepDownBilinearSpeed /= 100;
System.out.println( "Generating Step Down Bicubic" );
start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    stepDownBicubic = scaleDownImage(
        RenderingHints.VALUE_INTERPOLATION_BICUBIC );
}
end = System.nanoTime();
stepDownBicubicSpeed = (end - start) / 1.0E6;
stepDownBicubicSpeed /= 100;
}

private void createTestImage() {
    sprite = new BufferedImage(
        IMG_WIDTH, IMG_HEIGHT, BufferedImage.TYPE_INT_ARGB );
    Graphics2D g2d = sprite.createGraphics();
    // draw checker board
    g2d.setColor( Color.WHITE );
    g2d.fillRect( 0, 0, IMG_WIDTH / 2, IMG_HEIGHT / 2 );
    g2d.fillRect( IMG_WIDTH / 2, IMG_HEIGHT / 2, IMG_WIDTH, IMG_HEIGHT );
    g2d.setColor( Color.BLACK );
```



```
g2d.fillRect( IMG_WIDTH / 2, 0, IMG_WIDTH , IMG_HEIGHT / 2 );
g2d.fillRect( 0, IMG_HEIGHT / 2, IMG_WIDTH / 2, IMG_HEIGHT );
// draw red diamond
g2d.setColor( Color.RED );
g2d.drawLine( 0, sprite.getHeight() / 2, sprite.getWidth() / 2, 0 );
g2d.drawLine(
    sprite.getWidth() / 2, 0, sprite.getWidth(), sprite.getHeight() / 2
);
g2d.drawLine(
    sprite.getWidth(), sprite.getHeight() / 2, sprite.getWidth() / 2,
    sprite.getHeight()
);
g2d.drawLine(
    sprite.getWidth() / 2, sprite.getHeight(), 0, sprite.getHeight() / 2
);
// draw circle
g2d.drawOval( 0, 0, sprite.getWidth(), sprite.getHeight() );
// draw hash lines
g2d.setColor( Color.GREEN );
int dx = sprite.getWidth() / 18;
for( int i = 0; i < sprite.getWidth(); i += dx ) {
    g2d.drawLine( i, 0, i, sprite.getHeight() );
}
g2d.setColor( Color.BLUE );
dx = sprite.getHeight() / 18;
for( int i = 0; i < sprite.getHeight(); i += dx ) {
    g2d.drawLine( 0, i, sprite.getWidth(), i );
}
// gradient circle
float x1 = sprite.getWidth() / 4;
float x2 = sprite.getWidth() * 3 / 4;
float y1 = sprite.getHeight() / 4;
float y2 = sprite.getHeight() * 3 / 4;
GradientPaint gp =
    new GradientPaint( x1, y1, Color.BLACK, x2, y2, Color.WHITE );
g2d.setPaint( gp );
g2d.fillOval(
    sprite.getWidth() / 4, sprite.getHeight() / 4,
    sprite.getWidth() / 2, sprite.getHeight() / 2
);
g2d.setFont( new Font( "Arial", Font.BOLD, 42 ) );
g2d.setRenderingHint(
    RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON
```



```
};

g2d.setColor( Color.LIGHT_GRAY );
g2d.drawString(
    "Pg.1", sprite.getWidth()/2-40, sprite.getHeight()/2-20 );
g2d.setColor( Color.DARK_GRAY );
g2d.drawString(
    "Pg.2", sprite.getWidth()/2-40, sprite.getHeight()/2+40 );
g2d.dispose();
}

@Override
protected void processInput( float delta ) {
    super.processInput( delta );
}

@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
}

@Override
protected void render( Graphics g ) {
    super.render( g );
    // Test Image
    g.drawImage(
        sprite, (canvas.getWidth() - sprite.getWidth()) / 2, 50, null );
    int sw = averaged.getWidth( null );
    int sh = averaged.getHeight( null );
    int pos = canvas.getHeight() - sh - 50;
    int textPos = pos + sh;
    // Averaged Image
    int imgPos = (sw + 50) * 0 + 50;
    g.drawImage( averaged, imgPos, pos, null );
    String time = String.format( "%.4f ms", averagedSpeed );
    g.drawString( "Area Avg", 50, textPos + 15 );
    g.drawString( time, 50, textPos + 30 );
    // Nearest Image
    imgPos = (sw + 50) * 1 + 50;
    g.drawImage( nearestNeighbor, imgPos, pos, null );
    time = String.format( "%.4f ms", nearestSpeed );
    g.drawString( "Nearest", imgPos, textPos + 15 );
    g.drawString( time, imgPos, textPos + 30 );
    // Nearest2 Image
    imgPos = (sw + 50) * 2 + 50;
    g.drawImage( nearest2, imgPos, pos, null );
    time = String.format( "%.4f ms", nearest2Speed );
```



```
g.drawString( "Nearest 2", imgPos, textPos + 15 );
g.drawString( time, imgPos, textPos + 30 );
// Bilinear Image
imgPos = (sw + 50) * 3 + 50;
g.drawImage( bilinear, imgPos, pos, null );
time = String.format( "%.4f ms", bilinearSpeed );
g.drawString( "Bilinear", imgPos, textPos + 15 );
g.drawString( time, imgPos, textPos + 30 );
// Bicubic Image
imgPos = (sw + 50) * 4 + 50;
g.drawImage( bicubic, imgPos, pos, null );
time = String.format( "%.4f ms", bicubicSpeed );
g.drawString( "Bicubic", imgPos, textPos + 15 );
g.drawString( time, imgPos, textPos + 30 );
// Step Down Bilinear Image
imgPos = (sw + 50) * 5 + 50;
g.drawImage( stepDownBilinear, imgPos, pos, null );
time = String.format( "%.4f ms", stepDownBilinearSpeed );
g.drawString( "Bilin-Step", imgPos, textPos + 15 );
g.drawString( time, imgPos, textPos + 30 );
// Step Down Bicubic Image
imgPos = (sw + 50) * 6 + 50;
g.drawImage( stepDownBicubic, imgPos, pos, null );
time = String.format( "%.4f ms", stepDownBicubicSpeed );
g.drawString( "Bicub-Step", imgPos, textPos + 15 );
g.drawString( time, imgPos, textPos + 30 );
}
private void generateAveragedInstance() {
    averaged = sprite.getScaledInstance(
        sprite.getWidth() / 4, sprite.getHeight() / 4,
        Image.SCALE_AREA_AVERAGING
    );
    averaged.getSource().startProduction( getConsumer() );
}
private void generateNearestNeighbor() {
    nearestNeighbor = sprite.getScaledInstance(
        sprite.getWidth() / 4, sprite.getHeight() / 4, Image.SCALE_REPLICATE
    );
    nearestNeighbor.getSource().startProduction( getConsumer() );
}
private BufferedImage scaleWithGraphics( Object hintValue ) {
    BufferedImage image = new BufferedImage(
        sprite.getWidth() / 4, sprite.getHeight() / 4,
```



```
        BufferedImage.TYPE_INT_ARGB
);
Graphics2D g2d = image.createGraphics();
g2d.setRenderingHint( RenderingHints.KEY_INTERPOLATION, hintValue );
g2d.drawImage(
    sprite, 0, 0, image.getWidth(), image.getHeight(), null );
g2d.dispose();
return image;
}

private BufferedImage scaleDownImage( Object hintValue ) {
    BufferedImage ret = sprite;
    int targetWidth = sprite.getWidth() / 4;
    int targetHeight = sprite.getHeight() / 4;
    int w = sprite.getWidth();
    int h = sprite.getHeight();
    do {
        w = w / 2;
        if( w < targetWidth ) {
            w = targetWidth;
        }
        h = h / 2;
        if( h < targetHeight ) {
            h = targetHeight;
        }
    }
    BufferedImage tmp =
        new BufferedImage( w, h, BufferedImage.TYPE_INT_ARGB );
    Graphics2D g2d = tmp.createGraphics();
    g2d.setRenderingHint( RenderingHints.KEY_INTERPOLATION, hintValue );
    g2d.drawImage( ret, 0, 0, w, h, null );
    g2d.dispose();
    ret = tmp;
} while( w != targetWidth || h != targetHeight );
return ret;
}

private ImageConsumer getConsumer() {
    return new ImageConsumer() {
        public void setProperties( Hashtable<?, ?> props ) { }
        public void setPixels( int x, int y, int w, int h, ColorModel model,
            int[] pixels, int off, int scansize ) { }
        public void setPixels( int x, int y, int w, int h, ColorModel model,
            byte[] pixels, int off, int scansize ) { }
        public void setHints( int hintflags ) { }
        public void setDimensions( int width, int height ) { }
    };
}
```

```
public void setColorModel( ColorModel model ) { }
public void imageComplete( int status ) { }
};

}

public static void main( String[] args ) {
    launchApp( new ScaleImageExample() );
}
}
```

ScaleUpImageExample 如图 10.14 所示，位于 javagames.images 包中，它就像前面的示例一样，只不过生成了一个小的图像并将其放大到一个较大的尺寸。注意，在缩小时产生较好结果的一些方法，当用来放大图像时，并不一定具有同样的表现。

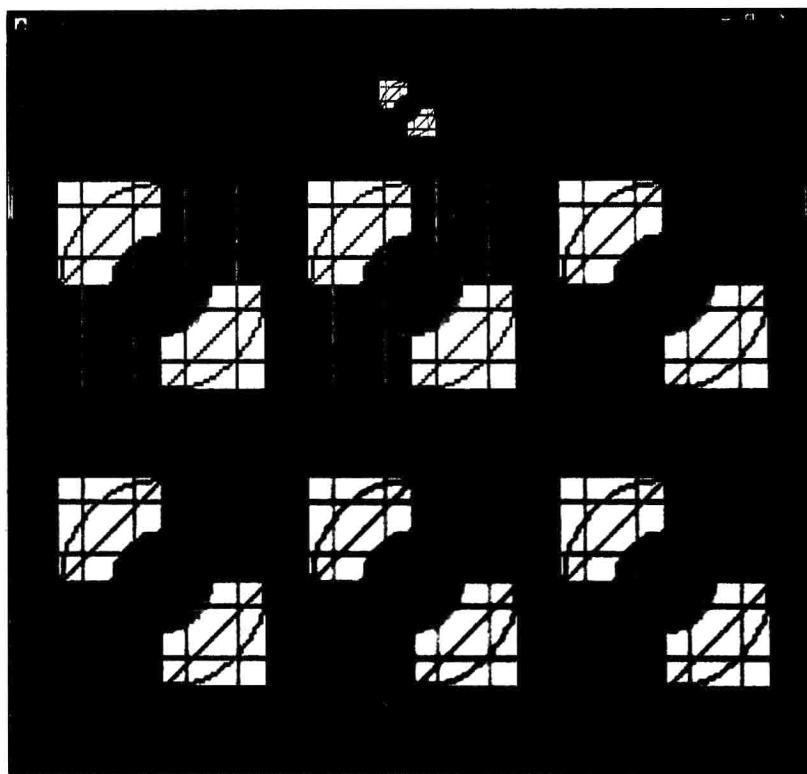


图 10.14 放大图像示例

双线性分步缩小方法对于缩小图像来说很好用，但是，要放大图像的话，我选择常规的双线性方法。但这完全取决于具体情况和图像。如果较慢的方法只是在开始的时候执行并且不会再次执行，那么，在加载资源时牺牲速度是值得的。



```
package javagames.images;
import java.awt.*;
import java.awt.image.*;
import java.util.Hashtable;
import javagames.util.SimpleFramework;

public class ScaleUpImageExample extends SimpleFramework {
    private static final int IMG_WIDTH = 64;
    private static final int IMG_HEIGHT = 64;
    private BufferedImage sprite;
    private Image averaged;
    private double averagedSpeed;
    private BufferedImage nearestNeighbor;
    private double nearestNeighborSpeed;
    private BufferedImage bilinear;
    private double bilinearSpeed;
    private BufferedImage bicubic;
    private double bicubicSpeed;
    private BufferedImage stepDownBilinear;
    private double stepDownBilinearSpeed;
    private BufferedImage stepDownBicubic;
    private double stepDownBicubicSpeed;
    public ScaleUpImageExample() {
        appWidth = 900;
        appHeight = 830;
        appBackground = Color.DARK_GRAY;
        appSleep = 1L;
        appTitle = "Scale Up Image Example";
    }
    @Override
    protected void initialize() {
        super.initialize();
        System.out.println( "Creating test image..." );
        createTestImage();
        System.out.println( "Generating Averaged Image" );
        long start = System.nanoTime();
        for( int i = 0; i < 100; ++i ) {
            generateAveragedInstance();
        }
        long end = System.nanoTime();
        averagedSpeed = (end - start) / 1.0E6;
        averagedSpeed /= 100;
        System.out.println( "Generating Nearest Neighbor" );
    }
}
```



```
start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    nearestNeighbor =
        scaleWithGraphics(
            RenderingHints.VALUE_INTERPOLATION_NEAREST_NEIGHBOR );
}
end = System.nanoTime();
nearestNeighborSpeed = (end - start) / 1.0E6;
nearestNeighborSpeed /= 100;
System.out.println( "Generating Bilinear" );
start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    bilinear = scaleWithGraphics(
        RenderingHints.VALUE_INTERPOLATION_BILINEAR );
}
end = System.nanoTime();
bilinearSpeed = (end - start) / 1.0E6;
bilinearSpeed /= 100;
System.out.println( "Generating Bicubic" );
start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    bicubic = scaleWithGraphics(
        RenderingHints.VALUE_INTERPOLATION_BICUBIC );
}
end = System.nanoTime();
bicubicSpeed = (end - start) / 1.0E6;
bicubicSpeed /= 100;
System.out.println( "Generating Step Down Bilinear" );
start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    stepDownBilinear = scaleUpImage(
        RenderingHints.VALUE_INTERPOLATION_BILINEAR );
}
end = System.nanoTime();
stepDownBilinearSpeed = (end - start) / 1.0E6;
stepDownBilinearSpeed /= 100;
System.out.println( "Generating Step Down Bicubic" );
start = System.nanoTime();
for( int i = 0; i < 100; ++i ) {
    stepDownBicubic = scaleUpImage(
        RenderingHints.VALUE_INTERPOLATION_BICUBIC );
}
end = System.nanoTime();
```





```
stepDownBicubicSpeed = (end - start) / 1.0E6;
stepDownBicubicSpeed /= 100;
}
private void createTestImage() {
    sprite = new BufferedImage(
        IMG_WIDTH, IMG_HEIGHT, BufferedImage.TYPE_INT_ARGB );
    Graphics2D g2d = sprite.createGraphics();
    // draw checker board
    g2d.setColor( Color.WHITE );
    g2d.fillRect( 0, 0, IMG_WIDTH / 2, IMG_HEIGHT / 2 );
    g2d.fillRect( IMG_WIDTH / 2, IMG_HEIGHT / 2, IMG_WIDTH, IMG_HEIGHT );
    g2d.setColor( Color.BLACK );
    g2d.fillRect( IMG_WIDTH / 2, 0, IMG_WIDTH , IMG_HEIGHT / 2 );
    g2d.fillRect( 0, IMG_HEIGHT / 2, IMG_WIDTH / 2, IMG_HEIGHT );
    // gradient circle
    float x1 = sprite.getWidth() / 4;
    float x2 = sprite.getWidth() * 3 / 4;
    float y1 = sprite.getHeight() / 4;
    float y2 = sprite.getHeight() * 3 / 4;
    GradientPaint gp =
        new GradientPaint( x1, y1, Color.BLACK, x2, y2, Color.WHITE );
    g2d.setPaint( gp );
    g2d.fillOval(
        sprite.getWidth() / 4, sprite.getHeight() / 4,
        sprite.getWidth() / 2, sprite.getHeight() / 2
    );
    // draw hash lines
    g2d.setColor( Color.GREEN );
    int dx = sprite.getWidth() / 4;
    for( int i = 0; i < sprite.getWidth(); i += dx ) {
        g2d.drawLine( i + 7, 0, i + 7, sprite.getHeight() );
    }
    g2d.setColor( Color.BLUE );
    dx = sprite.getHeight() / 4;
    for( int i = 0; i < sprite.getHeight(); i += dx ) {
        g2d.drawLine( 0, i + 7, sprite.getWidth(), i + 7 );
    }
    // draw red diamond
    g2d.setColor( Color.RED );
    g2d.drawLine( 0, sprite.getHeight() / 2, sprite.getWidth() / 2, 0 );
    g2d.drawLine( sprite.getWidth() / 2, 0,
        sprite.getWidth(), sprite.getHeight() / 2 );
    g2d.drawLine( sprite.getWidth(), sprite.getHeight() / 2,
```



```
        sprite.getWidth() / 2, sprite.getHeight() );
g2d.drawLine( sprite.getWidth() / 2, sprite.getHeight(), 0,
    sprite.getHeight() / 2 );
// draw circle
g2d.drawOval( 0, 0, sprite.getWidth(), sprite.getHeight() );
g2d.dispose();
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    // Test Image
    g.drawImage( sprite,
        (canvas.getWidth() - sprite.getWidth()) / 2, 50, null );
    int sw = averaged.getWidth( null );
    int sh = averaged.getHeight( null );
    int pos = sprite.getHeight() + 100;
    int textPos = pos + sh;
    // Averaged Image
    int imgPos = (sw + 50) * 0 + 50;
    g.drawImage( averaged, imgPos, pos, null );
    String time = String.format( "%.4f ms", averagedSpeed );
    g.drawString( "Area Avg", 50, textPos + 15 );
    g.drawString( time, 50, textPos + 30 );
    // Nearest Image
    imgPos = (sw + 50) * 1 + 50;
    g.drawImage( nearestNeighbor, imgPos, pos, null );
    time = String.format( "%.4f ms", nearestNeighborSpeed );
    g.drawString( "Nearest", imgPos, textPos + 15 );
    g.drawString( time, imgPos, textPos + 30 );
    // Bilinear Image
    imgPos = (sw + 50) * 2 + 50;
    g.drawImage( bilinear, imgPos, pos, null );
    time = String.format( "%.4f ms", bilinearSpeed );
    g.drawString( "Bilinear", imgPos, textPos + 15 );
    g.drawString( time, imgPos, textPos + 30 );
    pos += nearestNeighbor.getHeight() + 100;
    textPos = pos + nearestNeighbor.getHeight();
    // Bicubic Image
    imgPos = (sw + 50) * 0 + 50;
    g.drawImage( bicubic, imgPos, pos, null );
    time = String.format( "%.4f ms", bicubicSpeed );
    g.drawString( "Bicubic", imgPos, textPos + 15 );
    g.drawString( time, imgPos, textPos + 30 );
```



```
// Step Down Bilinear Image
imgPos = (sw + 50) * 1 + 50;
g.drawImage( stepDownBilinear, imgPos, pos, null );
time = String.format( "%.4f ms", stepDownBilinearSpeed );
g.drawString( "Bilin-Step", imgPos, textPos + 15 );
g.drawString( time, imgPos, textPos + 30 );
// Step Down Bicubic Image
imgPos = (sw + 50) * 2 + 50;
g.drawImage( stepDownBicubic, imgPos, pos, null );
time = String.format( "%.4f ms", stepDownBicubicSpeed );
g.drawString( "Bicub-Step", imgPos, textPos + 15 );
g.drawString( time, imgPos, textPos + 30 );
}
private void generateAveragedInstance() {
    averaged = sprite.getScaledInstance( (int)(sprite.getWidth() * 3.7),
        (int)(sprite.getHeight() * 3.7), Image.SCALE_AREA_AVERAGING );
    averaged.getSource().startProduction( getConsumer() );
}
private BufferedImage scaleWithGraphics( Object hintValue ) {
    BufferedImage image = new BufferedImage(
        (int)(sprite.getWidth() * 3.7),
        (int)(sprite.getHeight() * 3.7), BufferedImage.TYPE_INT_ARGB );
    Graphics2D g2d = image.createGraphics();
    g2d.setRenderingHint( RenderingHints.KEY_INTERPOLATION, hintValue );
    g2d.drawImage(
        sprite, 0, 0, image.getWidth(), image.getHeight(), null );
    g2d.dispose();
    return image;
}
private BufferedImage scaleUpImage( Object hintValue ) {
    BufferedImage ret = sprite;
    int targetWidth = (int)(sprite.getWidth() * 3.7);
    int targetHeight = (int)(sprite.getHeight() * 3.7);
    int w = sprite.getWidth();
    int h = sprite.getHeight();
    do {
        w = (int)(w * 1.5);
        if(' w > targetWidth ) {
            w = targetWidth;
        }
        h = (int)(h * 1.5);
        if( h > targetHeight ) {
            h = targetHeight;
        }
    } while( w < targetWidth || h < targetHeight );
    if( w > targetWidth ) {
        w = targetWidth;
    }
    if( h > targetHeight ) {
        h = targetHeight;
    }
    ret = scaleWithGraphics( hintValue );
}
```

```
        }
        BufferedImage tmp =
            new BufferedImage( w, h, BufferedImage.TYPE_INT_ARGB );
        Graphics2D g2d = tmp.createGraphics();
        g2d.setRenderingHint( RenderingHints.KEY_INTERPOLATION, hintValue );
        g2d.drawImage( ret, 0, 0, w, h, null );
        g2d.dispose();
        ret = tmp;
    } while( w != targetWidth || h != targetHeight );
    return ret;
}
private ImageConsumer getConsumer() {
    return new ImageConsumer() {
        public void setProperties( Hashtable<?, ?> props ) { }
        public void setPixels( int x, int y, int w, int h, ColorModel model,
            int[] pixels, int off, int scansize ) { }
        public void setPixels( int x, int y, int w, int h, ColorModel model,
            byte[] pixels, int off, int scansize ) { }
        public void setHints( int hintflags ) { }
        public void setDimensions( int width, int height ) { }
        public void setColorModel( ColorModel model ) { }
        public void imageComplete( int status ) { }
    };
}
public static void main( String[] args ) {
    launchApp( new ScaleUpImageExample() );
}
}
```

## 10.9 资源与延伸阅读

Campbell, Chris, "Java 2D Trickery: Antialiased Image Transforms," 2007, [https://weblogs.java.net/blog/campbell/archive/2007/03/java\\_2d\\_tricker.html](https://weblogs.java.net/blog/campbell/archive/2007/03/java_2d_tricker.html).

Haase, Chet, "BufferedImage as Good as Butter," 2003, [https://weblogs.java.net/blog/chet/archive/2003/08/bufferedimage\\_a.html](https://weblogs.java.net/blog/chet/archive/2003/08/bufferedimage_a.html).

Haase, Chet, "BufferedImage as Good as Butter, Part II," 2003, [https://weblogs.java.net/blog/chet/archive/2003/08/bufferedimage\\_a\\_1.html](https://weblogs.java.net/blog/chet/archive/2003/08/bufferedimage_a_1.html).



Haase, Chet, “VolatileImage: Now You See it, Now You Don’t,” 2003, [https://weblogs.java.net/blog/chet/archive/2003/09/volatileimage\\_n.html](https://weblogs.java.net/blog/chet/archive/2003/09/volatileimage_n.html).

Haase, Chet, “VolatileImage Q&A,” 2003, [https://weblogs.java.net/blog/chet/archive/2003/09/volatileimage\\_q.html](https://weblogs.java.net/blog/chet/archive/2003/09/volatileimage_q.html).

Haase, Chet, “ImageIO: Just Another Example of Better Living by Doing it Yourself,” 2004, [https://weblogs.java.net/blog/chet/archive/2004/07/imageio\\_just\\_an.html](https://weblogs.java.net/blog/chet/archive/2004/07/imageio_just_an.html).

Haase, Chet, “ToolkitBufferedVolatileManagedImage Strategies,” 2004, <https://weblogs.java.net/blog/chet/archive/2004/08/toolkitbuffered.html>.



# 第 11 章

## 文本

我们第一次尝试将文本显示到屏幕上，是为了显示帧速率，这很容易。但是，你必须要注意到，所绘制的所有文本都是直接编码的值，就像下面的代码一样：

```
// draw directions on top of bouncing balls
super.render( g );
g.drawString( "Ball Count: " + ballCount, 20, 35 );
g.drawString( "Press [SPACE] to launch.", 20, 50 );
g.drawString( "Press Up arrow to increase ball count", 20, 65 );
g.drawString( "Press Down arrow to decrease ball count", 20, 80 );
```

如果字体大小变了，那么对高度直接编码将不再有效。但是，将文本放在一个特定位  
置会比看上去难度更大。

### 11.1 理解 Java 字体

要在 Java 中创建新字体，需要给构造方法传递 3 个参数：字体名称、样式及其大小。

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

字体名称是表示字体名的一个字符串，例如，Courier New。可以从 `GraphicsEnvironment` 对象获取可用字体的一个列表。

```
public static void main( String[] args ) {
    GraphicsEnvironment ge =
        GraphicsEnvironment.getLocalGraphicsEnvironment();
    String[] fontFamilies = ge.getAvailableFontFamilyNames();
    for( String fontFamily : fontFamilies ) {
        System.out.println( fontFamily );
    }
}
```



字体大小是字体的点大小，例如，12 表示 12 点的字体。样式可以是普通、粗体、斜体或者粗体且斜体，可以用 OR | 运算符将粗体和斜体属性组合起来。

```
Font plain = new Font("Name", Font.PLAIN, 12 );
Font bold = new Font("Name", Font.BOLD, 12 );
Font italic = new Font("Name", Font.ITALIC, 12 );
Font boldItalic = new Font("Name", Font.BOLD | Font.ITALIC, 12 );
```

记住，在得到当前字体的宽度和高度之前，必须创建 font 对象并且在 Graphics 对象中设置它。

图 11.1 所示的 BoxedTextProblem 示例，其代码位于 javagames.text 包中，它尝试围绕一些文本绘制一个矩形，从而展示了将文本和图像绘制到屏幕上的区别。

```
package javagames.text;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import javagames.util.SimpleFramework;

public class BoxedTextProblem extends SimpleFramework {
    public BoxedTextProblem() {
        appWidth = 640;
        appHeight = 640;
        appSleep = 10L;
        appTitle = "Boxed Text Problem";
        appBackgroundColor = Color.WHITE;
        appFPSColor = Color.BLACK;
    }
    @Override
    protected void initialize() {
        super.initialize();
    }
    @Override
    protected void render( Graphics g ) {
        super.render( g );
        // Box this text...
        g.setColor( Color.BLACK );
        String box = "great Java, now what?";
        Font font = new Font( "Arial", Font.PLAIN, 24 );
        g.setFont( font );
        g.drawString( box, 20, 50 );
        g.setColor( Color.RED );
```



```
g.drawRect( 20, 50, 200, 20 );  
}  
public static void main( String[] args ) {  
    launchApp( new BoxedTextProblem() );  
}  
}  
}
```

矩形的宽度只是一个猜测，暂时先忽略这一点。即便矩形和文本都按照其左上角位于(20, 50)来绘制，结果也并不相似，如图 11.1 所示。

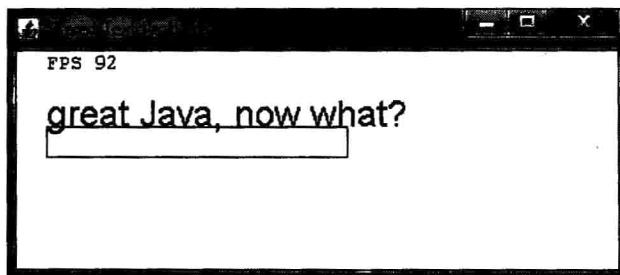


图 11.1 带边框的文本问题

问题在于，对待文本与图像不一样。图像是从左上角开始绘制的，文本是从基线开始绘制的，并且，渲染字符串的高度和宽度取决于所选择的字体类型、样式和大小。

基线穿越字体的底部。然而，一些字母可能延伸到基线以下。Ascent (上升) 值是从基线到字体顶部的距离。Descent (下沉) 值是字母延伸到基线以下的距离。Leading (行距) 值是为了让字母不要彼此碰到，两行之间的距离。这 3 个值加起来，构成了字母的总高度，如图 11.2 所示。

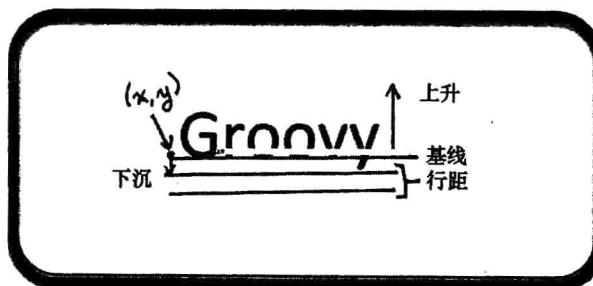


图 11.2 文本度量

$$\text{Height} = \text{ascent} + \text{descent} + \text{leading}$$

和 Java 中的大多数问题一样，有多种方式来解决这个问题。





BoxedTextSolution 代码如图 11.3 所示，位于 javagames.text 包中，它展示了围绕一些文本放置一个边框而不用猜测值所需要的步骤。第一步是从 Graphics 对象获取 FontMetrics。

```
FontMetrics fm = g.getFontMetrics();
```

由于一些字体对于每个字母来说宽度都不同，因此我们需要字符串值来确定该宽度。

```
int width = fm.stringWidth( str );
```

现在，字符串的宽度可用，下一步是绘制文本，以使得左上角不会放在基线上，而是保持在左上角。绘制的起始点在基线，加上上升值，也就是文本在 y 坐标上提高的距离，将文本放到正确的位置。

```
g.drawString( str, x, y + fm.getAscent() );
```

现在，使用左上角位置绘制了一条线，这是因为文本根据上升值发生了移动。剩下的就是计算字体的高度。

```
int height = fm.getAscent() + fm.getDescent() + fm.getLeading();
```

有了左上角的位置、高度、宽度，以及绘制文本的正确位置，围绕文本放置一个边框就容易了。BoxedTextSolution 通过一系列必要的方法调用，放置了一个围绕文本的边框。正确的带边框的文本，基本上把游戏文本放到了需要的位置。

```
package javagames.text;
import java.awt.*;
import javagames.util.SimpleFramework;
public class BoxedTextSolution extends SimpleFramework {
    public BoxedTextSolution() {
        appWidth = 640;
        appHeight = 640;
        appSleep = 10L;
        appTitle = "Boxed Text Solution";
        appBackground = Color.WHITE;
        appFPSColor = Color.BLACK;
    }
    @Override
    protected void initialize() {
```

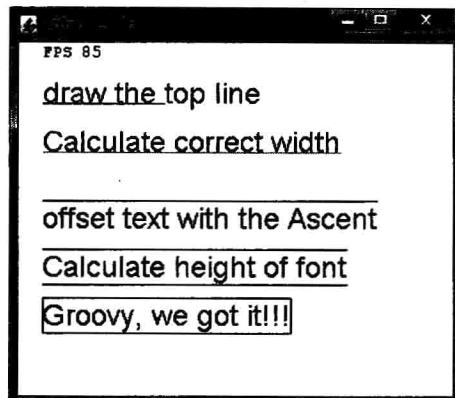


图 11.3 带边框的文本解决方案



```
super.initialize();
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    // Set the font...
    Font font = new Font( "Arial", Font.PLAIN, 24 );
    g.setFont( font );
    FontMetrics fm = g.getFontMetrics();
    int x = 20;
    int y = 50;
    // Draw the top...
    String str = "draw the top line";
    g.setColor( Color.DARK_GRAY );
    g.drawString( str, x, y );
    int width = 100;
    g.setColor( Color.RED );
    g.drawLine( x, y, x + width, y );
    // Calculate the string width...
    y += 40;
    str = "Calculate correct width";
    g.setColor( Color.DARK_GRAY );
    g.drawString( str, x, y );
    width = fm.stringWidth( str );
    g.setColor( Color.GREEN );
    g.drawLine( x, y, x + width, y );
    // Use Ascent to offset y
    y += 40;
    g.setColor( Color.DARK_GRAY );
    str = "offset text with the Ascent";
    g.drawString( str, x, y + fm.getAscent() );
    width = fm.stringWidth( str );
    g.setColor( Color.BLUE );
    g.drawLine( x, y, x + width, y );
    // Ascent+Descent+Leading=Height
    y += 40;
    g.setColor( Color.DARK_GRAY );
    str = "Calculate height of font";
    g.drawString( str, x, y + fm.getAscent() );
    width = fm.stringWidth( str );
    g.setColor( Color.BLUE );
    g.drawLine( x, y, x + width, y );
    int height = fm.getAscent() + fm.getDescent() + fm.getLeading();
```



```
g.drawLine( x, y + height, x + width, y + height );
// Box the text
y += 40;
g.setColor( Color.DARK_GRAY );
str = "Groovy, we got it!!!";
g.drawString( str, x, y + fm.getAscent() );
width = fm.stringWidth( str );
g.setColor( Color.BLUE );
height = fm.getAscent() + fm.getDescent() + fm.getLeading();
g.drawRect( x, y, width, height );
}
public static void main( String[] args ) {
    launchApp( new BoxedTextSolution() );
}
}
```

## 11.2 制作绘制字符串工具

现在可以根据字体大小来确定文本的放置，我们可以给位于 `javagames.util` 的 `Utility` 类添加一些工具方法。给定一个起始位置和一个字符串的数组，工具方法把文本绘制到屏幕上，带有正确的空白，并保持起始位置在左上角。

```
public class Utility {
    // previous code here
    public static int drawString( Graphics g, int x, int y, String str ) {
        return drawString( g, x, y, new String[]{ str } );
    }
    public static int drawString(
        Graphics g, int x, int y, List<String> str ) {
        return drawString( g, x, y, str.toArray( new String[0] ) );
    }
    public static int drawString(
        Graphics g, int x, int y, String... str ) {
        FontMetrics fm = g.getFontMetrics();
        int height = fm.getAscent() + fm.getDescent() + fm.getLeading();
        for( String s : str ) {
            g.drawString( s, x, y + fm.getAscent() );
            y += height;
        }
    }
}
```



```
        }
        return y;
    }
}
```

注意，一旦绘制了文本，这些新方法将返回新的 y 位置。这允许更多的文本附加到之前绘制的文本之后。使用新的工具方法，如下的代码将添加到位于 `javagames.util` 包中的 `SimpleFramework` 中。

```
public class SimpleFramework extends JFrame implements Runnable {
    // attributes
    protected int textPos = 0; // this is new!
    public SimpleFramework() {
    }
    protected void render( Graphics g ) {
        g.setFont( appFont );
        g.setColor( appFPSColor );
        frameRate.calculate();
        textPos =
            Utility.drawString( g, 20, 0, frameRate.getFrameRate() );
            //new
    }
    // rest of the code
}
```

现在你可以使用 `Utility` 方法绘制文本，帧速率的 y 位置将会是左上角，而不管字体有多大。之前，较大的字体可能导致帧速率显示在屏幕之外。

`UtilityDrawStringExample` 如图 11.4 所示，位于 `javagames.text` 包中，它使用对 `SimpleFramework` 的新的更新以及 `Utility` 类来展示用新的方法绘制文本。`processInput()` 方法处理向上和向下箭头的按下事件，把字体大小调大或调小以测试新的方法。



Java 有两个不同的抗锯齿方法，一个用于图像，另一个单独用于文本。  
在绘制文本时，要确保打开基于文本的抗锯齿渲染提示。

`render()` 方法首先打开文本抗锯齿。然后，它使用 `textPos` 变量以多种不同的方式绘制文本：一个使用单个的值，一个使用字符串的一个数组，一个使用字符串的一个列表，还有一个使用多个字符串值。由于新的方法会计算字体的高度，因此这些字符串总是正确地空开。



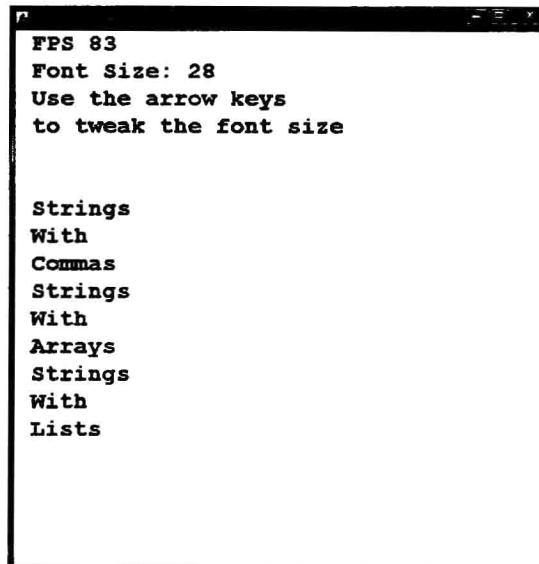


图 11.4 Utility 绘制字符串示例

```
package javagames.text;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.Vector;
import javagames.util.*;

public class UtilityDrawStringExample extends SimpleFramework {
    public UtilityDrawStringExample() {
        appFont = new Font( "Courier New", Font.BOLD, 48 );
        appWidth = 640;
        appHeight = 640;
        appSleep = 10L;
        appTitle = "Utility Draw String Example";
        appBackground = Color.WHITE;
        appFPSColor = Color.BLACK;
    }
    @Override
    protected void processInput( float delta ) {
        super.processInput( delta );
        if( keyboard_KeyDownOnce( KeyEvent.VK_UP ) ) {
            int fontSize = appFont.getSize();
            appFont = new Font(
                appFont.getFamily(), appFont.getStyle(), fontSize + 2 );
        }
    }
}
```



```
if( keyboard.keyDownOnce( KeyEvent.VK_DOWN ) ) {
    int fontSize = appFont.getSize();
    appFont = new Font(
        appFont.getFamily(), appFont.getStyle(), fontSize - 2 );
}
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    Graphics2D g2d = (Graphics2D)g;
    g2d.setRenderingHint(
        RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON
    );
    textPos = Utility.drawString( g2d, 20, textPos,
        "Font Size: " + g2d.getFont().getSize()
    );
    textPos = Utility.drawString( g2d, 20, textPos,
        "Use the arrow keys",
        "to tweak the font size",
        ""
    );
    g2d.setColor( Color.WHITE );
    textPos = Utility.drawString( g2d, 20, textPos, "Single String" );
    g2d.setColor( Color.BLUE );
    textPos = Utility.drawString( g2d, 20, textPos,
        "Strings",
        "With",
        "Commas"
    );
    g2d.setColor( Color.DARK_GRAY );
    String[] array = new String[] {
        "Strings",
        "With",
        "Arrays",
    };
    textPos = Utility.drawString( g2d, 20, textPos, array );
    g2d.setColor( Color.RED );
    Vector<String> list = new Vector<String>();
    list.add( "Strings" );
    list.add( "With" );
    list.add( "Lists" );
    textPos = Utility.drawString( g2d, 20, textPos, list );
```



```
    }
    public static void main( String[] args ) {
        launchApp( new UtilityDrawStringExample() );
    }
}
```

## 11.3 使用文本度量进行布局

FontMetrics 类并非获取字体相关信息的唯一方法。Java 还有其他的类和方法可以提供有关字体对象的可用信息。

FontMetrics 对象提供了一个 LineMetrics 对象，它拥有额外的信息，例如一种字体的中央基线。

```
float dy = fm.getLineMetrics(
    str, graphics).getBaselineOffsets()[ Font.CENTER_BASELINE ];
```

TextLayout 对象甚至更有用。可以使用一个 Font、一个 FontRenderContext 和一个 String 构建该对象。

```
Graphics2D g2d;
Font font;
String str;
FontRenderContext frc = g2d.getFontRenderContext();
TextLayout tl = new TextLayout( str, font, frc );
```

文本布局对象之所以有用，是因为它以浮点值而不是整数的形式提供了很多信息，因此，可以尽可能精确地放置。TextLayout 对象有如下的值可供使用。

- Ascent
- Descent
- Leading
- Baseline
- Roman baseline
- Center baseline
- Hanging baseline
- Advance

- Bounds
- Visible bounds

TextMetricsExample 如图 11.5 所示, 位于 javagames.text 包中, 它展示了使用与字体相关的各种可用信息。

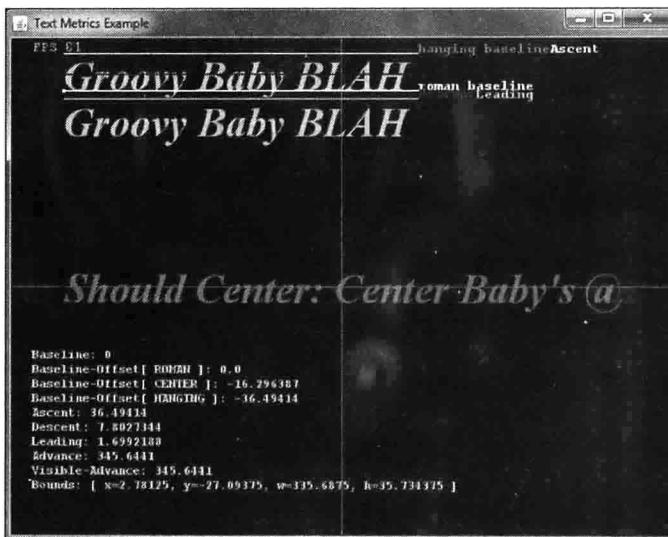


图 11.5 文本度量示例

该示例展示了有关字体的很多信息。这些值也在应用程序的底部列出。注意, 一些字体值是负数。负值要和基线相加, 而不是相减。

```
package javagames.text;

import java.awt.*;
import java.awt.font.*;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import javagames.util.*;

public class TextMetricsExample extends SimpleFramework {
    public TextMetricsExample() {
        appWidth = 640;
        appHeight = 480;
        appSleep = 10L;
        appTitle = "Text Metrics Example";
    }
}
```



```
@Override
protected void initialize() {
    super.initialize();
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    Graphics2D g2d = (Graphics2D)g;
    g2d.setRenderingHint( RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON );
    Font font = new Font( "Times New Roman", Font.BOLD | Font.ITALIC, 40 );
    g2d.setFont( font );
    g2d.setColor( Color.GREEN );
    String str = "Groovy Baby BLAH";
    int x = 50;
    int y = 50;
    g2d.drawString( str, x, y );
    // Text Layout gives floating point values
    FontRenderContext frc = g2d.getFontRenderContext();
    TextLayout tl = new TextLayout( str, font, frc );
    // draw another line, should be at
    // y + ascent + decent + leading
    int newY =
        y + (int)( tl.getAscent() + tl.getDescent() + tl.getLeading() );
    g2d.drawString( str, x, newY );
    // draw centered Text
    // first lets draw the center of the window...
    g2d.setColor( Color.GRAY );
    int sw = canvas.getWidth();
    int sh = canvas.getHeight();
    int cx = sw / 2;
    int cy = sh / 2;
    g2d.drawLine( 0, cy, sw, cy );
    g2d.drawLine( cx, 0, cx, sh );
    String center = "Should Center: Center Baby's @";
    // to calculate the x, need the width...
    int stringWidth = g2d.getFontMetrics().stringWidth( center );
    float dy = g2d.getFontMetrics().getLineMetrics( center, g2d )
        .getBaselineOffsets()[ Font.CENTER_BASELINE ];
    g2d.drawString( center, cx - stringWidth / 2, cy - dy );
    // draw the pixel where we are drawing the text...
    g2d.setColor( Color.WHITE );
    g2d.fillRect( x - 1, y - 1, 3, 3 );
```



```
ArrayList<String> console = new ArrayList<String>();
console.add( "Baseline: " + tl.getBaseline() );
float[] baselineOffsets = tl.getBaselineOffsets();
console.add( "Baseline-Offset[ ROMAN ]: "
    + baselineOffsets[ Font.ROMAN_BASELINE ] );
console.add( "Baseline-Offset[ CENTER ]: "
    + baselineOffsets[ Font.CENTER_BASELINE ] );
console.add( "Baseline-Offset[ HANGING ]: "
    + baselineOffsets[ Font.HANGING_BASELINE ] );
console.add( "Ascent: " + tl.getAscent() );
console.add( "Descent: " + tl.getDescent() );
console.add( "Leading: " + tl.getLeading() );
console.add( "Advance: " + tl.getAdvance() );
console.add( "Visible-Advance: " + tl.getVisibleAdvance() );
console.add( "Bounds: " + toString( tl.getBounds() ) );
Font propFont = new Font( "Courier New", Font.BOLD, 14 );
g2d.setFont( propFont );
int xLeft = x;
int xRight = xLeft + (int)tl.getVisibleAdvance();
// draw baseline
g2d.setColor( Color.WHITE );
int baselineY = y + (int)baselineOffsets[ Font.ROMAN_BASELINE ];
g2d.drawLine( xLeft, baselineY, xRight, baselineY );
g2d.drawString( "roman baseline", xRight, baselineY );
// draw center
g2d.setColor( Color.BLUE );
int centerY = y + (int)baselineOffsets[ Font.CENTER_BASELINE ];
g2d.drawLine( xLeft, centerY, xRight, centerY );
g2d.drawString( "center baseline", xRight, centerY );
// draw hanging
g2d.setColor( Color.GRAY );
int hangingY = y + (int)baselineOffsets[ Font.HANGING_BASELINE ];
g2d.drawLine( xLeft, hangingY, xRight, hangingY );
g2d.drawString( "hanging baseline", xRight, hangingY );
// draw Ascent
g2d.setColor( Color.YELLOW );
int propY = y - (int)tl.getAscent();
g2d.drawLine( xLeft, propY, xRight, propY );
TextLayout temp = new TextLayout( "hanging baseline", propFont,
    g2d.getFontRenderContext() );
g2d.drawString( "Ascent", xRight + temp.getVisibleAdvance(), propY );
// draw Descent
g2d.setColor( Color.RED );
```





```
propY = y + (int)tl.getDescent();
g2d.drawLine( xLeft, propY, xRight, propY );
g2d.drawString( "Descent", xRight, propY );
// draw Leading
g2d.setColor( Color.GREEN );
propY = y + (int)tl.getDescent() + (int)tl.getLeading();
g2d.drawLine( xLeft, propY, xRight, propY );
temp = new TextLayout(
    "Descent", propFont, g2d.getFontRenderContext() );
g2d.drawString( "Leading", xRight + temp.getVisibleAdvance(), propY );
// draw console output...
g2d.setColor( Color.LIGHT_GRAY );
g2d.setFont( new Font( "Courier New", Font.BOLD, 12 ) );
Utility.drawString( g2d, 20, 300, console );
}
private String toString( Rectangle2D r ) {
    return "[ x=" + r.getX() + ", y=" + r.getY() + ", w=" + r.getWidth()
        + ", h=" + r.getHeight() + " ]";
}
public static void main( String[] args ) {
    launchApp( new TextMetricsExample() );
}
}
```

TextLayoutExample 如图 11.6 所示，位于 javagames.text 包中，它使用字体信息来绘制一个复古风格的高分输入界面。initialize()方法负责计算每个字母的宽度，以确定最大宽度。由于图形对象在 initialize()方法中还不可用，因此引发了一个问题。有两种方法解决这个问题。一种方法是从 JFrame 类获取字体度量对象。

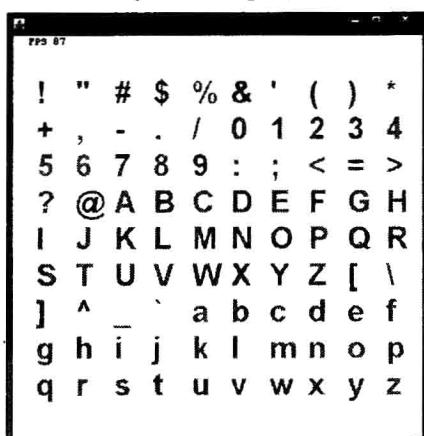


图 11.6 文本布局示例

```
Font font = new Font(...);
FontMetrics fm = this.getFontMetrics(font);
```

第二种方法是通过给 JFrame 一个 Font 对象，从而创建一个 FontMetrics 对象。然而，这两种解决方案都需要访问一个 JFrame 对象。如果当前类不能访问一个 JFrame，那么获取 FontMetrics 以进行字体计

算的另一种方法是，创建一个图像并使用其 Graphic 对象。

```
BufferedImage img = new
BufferedImage( 1, 1, BufferedImage.TYPE_INT_ARGB );
```



```
Graphics2D g = img.createGraphics();
FontMetrics fontMetrics = g.getFontMetrics( font );
g.dispose();
```

尽管需要编写一些代码来从头创建一个新的对象，但如果 `JFrame` 类不可用的话，这是一种可选的方法。

```
package javagames.text;

import java.awt.*;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import javagames.util.SimpleFramework;

public class TextLayoutExample extends SimpleFramework {
    private Font font;
    private int maxWidth;
    public TextLayoutExample() {
        appWidth = 640;
        appHeight = 640;
        appSleep = 10L;
        appTitle = "Text Layout Example";
        appBackground = Color.WHITE;
        appFPSColor = Color.BLACK;
    }
    @Override
    protected void initialize() {
        super.initialize();
        font = new Font( "Arial", Font.BOLD, 40 );
        FontMetrics fm = getFontMetrics( font );
        maxWidth = Integer.MIN_VALUE;
        for( int i = (int)'!'; i < (int)'z'; ++i ) {
            String letter = " " + (char)i;
            maxWidth = Math.max( maxWidth, fm.stringWidth( letter ) );
        }
        // another way
        BufferedImage img =
            new BufferedImage( 1, 1, BufferedImage.TYPE_INT_ARGB );
        Graphics2D g = img.createGraphics();
        FontMetrics fontMetrics = g.getFontMetrics( font );
        g.dispose();
    }
    @Override
```





```
protected void render( Graphics g ) {
    super.render( g );
    Graphics2D g2d = (Graphics2D)g;
    g2d.setRenderingHint( RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON );
    g2d.setColor( Color.GREEN );
    g2d.setFont( font );
    FontMetrics fm = g2d.getFontMetrics();
    int height = fm.getAscent() + fm.getDescent() + fm.getLeading();
    int x = 20;
    int y = 50;
    y += fm.getAscent();
    int count = 0;
    for( int i = (int)'!' ; i <= (int)'z' ; ++i ) {
        String letter = " " + (char)i;
        g2d.drawString( letter, x, y );
        x += maxWidth;
        count++;
        if( count % 10 == 0 ) {
            y += height;
            x = 20;
        }
    }
}
public static void main( String[] args ) {
    launchApp( new TextLayoutExample() );
}
```

ConsoleOverlayExample 如图 11.7 所示，位于 javagames.text 包中，它将文本放置与透

明度组合起来，创建了一个可以滚动并隐藏自己的文本控制台。initialize()方法创建了 Font 并且计算了字体的高度。滚动的文本在 Vector 对象中创建。文本的第一行复制到了底部以用于滚动。控制台设置为文本的高度加上一条额外的线条，但是，由于复制了顶部的线条，因此实际上有两条额外的线条。

processInput()方法负责两件事情。第一，当鼠标位于控制台的边界之中时，它负责监控鼠标以及更新透明度。其二，无论何时当空格键按下时，它开始滚动文本；用 B 键来开关滚动文本的边框，并且用 H 键显示或隐藏控制台。

updateObjects()方法负责处理控制台状态。首先用一个灰色的背景填充控制台，然后，将文本绘制到背景之上。再然后，将滚动框绘制到文本之上，或者绘制绿色的矩形以显示滚动框的位置。

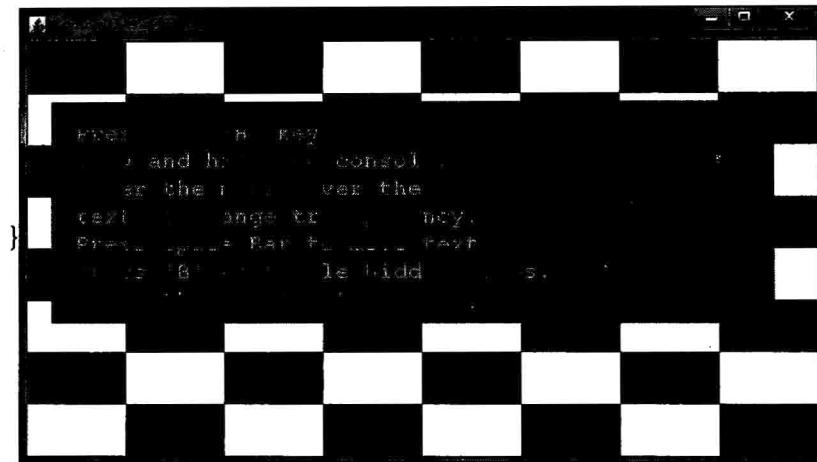


图 11.7 控制台重叠示例

隐藏值变量是从 0 到 1 的一个浮点数，其中 1 是完全隐藏，0 是完全可见。根据控制台是否显示或隐藏，使用增量时间值来更新 hidden 值，以缓慢地显示或隐藏控制台。如果 hidden 值还不是 0，那么，使用 AlphaComposite 对象以一定的比例清理控制台的每一边。这将允许控制台在任何背景之上变得可见。



第 10 章介绍过 AlphaComposite 类和透明度。如果不理解 ConsoleOverlayExample 中的任何代码的话，请回顾一下该章。

render()方法创建了棋盘背景，设置了 AlphaComposite，并且在背景图像上覆盖文本控制台。

```
package javagames.text;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Vector;
import javagames.util.*;

public class ConsoleOverlayExample extends SimpleFramework {
    private BufferedImage console;
    private int xConsole;
    private int yConsole;
    private float alpha;
    private Font consoleFont;
```



```
private int fontHeight;
private Vector<String> text;
private float currY;
private boolean boxes;
private boolean hide;
private float hidden;
public ConsoleOverlayExample() {
    appWidth = 640;
    appHeight = 640;
    appSleep = 10L;
    appTitle = "Console Overlay Example";
}
@Override
protected void initialize() {
    super.initialize();
    consoleFont = new Font( "Courier New", Font.BOLD, 20 );
    FontMetrics fm = getFontMetrics( consoleFont );
    fontHeight = fm.getAscent() + fm.getDescent() + fm.getLeading();
    currY = 0;
    text = new Vector<String>();
    text.add( "Press the 'H' key to" );
    text.add( "show and hide the console." );
    text.add( "Hover the mouse over the" );
    text.add( "text to change transparency." );
    text.add( "Press Space Bar to move text." );
    text.add( "Press 'B' to toggle hidden boxes." );
    text.add( 0, text.lastElement() );
    int consoleHeight = fontHeight * (text.size() + 1);
    console = new BufferedImage(
        canvas.getWidth() - 40,
        consoleHeight,
        BufferedImage.TYPE_INT_ARGB
    );
    xConsole = 20;
    yConsole = 50;
    hide = false;
    hidden = 1.0f;
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    Point pos = mouse.getPosition();
    int minX = xConsole;
```



```
int minY = yConsole;
int maxX = minX + console.getWidth();
int maxY = minY + console.getHeight();
if( pos.x > minX && pos.x < maxX &&
    pos.y > minY && pos.y < maxY ) {
    alpha = 1.0f;
} else {
    alpha = 0.75f;
}
if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
    text.remove( 0 );
    text.add( text.get( 0 ) );
    currY = fontHeight;
}
if( keyboard.keyDownOnce( KeyEvent.VK_B ) ) {
    boxes = !boxes;
}
if( keyboard.keyDownOnce( KeyEvent.VK_H ) ) {
    hide = !hide;
}
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    if( hide && hidden == 1.0f ) {
        return; // don't bother
    }
    if( currY > 0 ) {
        currY -= delta* fontHeight;
    }
    Graphics2D g = console.createGraphics();
    g.setRenderingHint(
        RenderingHints.KEY_TEXT_ANTIALIASING,
        RenderingHints.VALUE_TEXT_ANTIALIAS_ON
    );
    g.setColor( Color.DARK_GRAY );
    g.fillRect( 0, 0, console.getWidth(), console.getHeight() );
    g.setColor( Color.LIGHT_GRAY );
    g.setFont( consoleFont );
    int x = 20;
    int y = (int)currY;
    Utility.drawString( g, x, y, text );
    if( boxes ) {
```



```
g.setColor( Color.DARK_GRAY );
g.fillRect( 0, 0, console.getWidth(), fontHeight );
g.fillRect(
    0, fontHeight * text.size(), console.getWidth(), fontHeight );
} else {
    g.setColor( Color.GREEN );
    g.drawRect( 0, 0, console.getWidth() - 1, fontHeight - 1 );
    g.drawRect(
        0, fontHeight * text.size(),
        console.getWidth() - 1, fontHeight - 1
    );
}
if( hide && hidden < 1.0f ) {
    hidden += delta ;
    if( hidden > 1.0f ) hidden = 1.0f;
} else if( !hide && hidden > 0.0f ) {
    hidden -= delta;
    if( hidden < 0.0f ) hidden = 0.0f;
}
if( hidden > 0.0f ) {
    g.setComposite(
        AlphaComposite.getInstance( AlphaComposite.CLEAR ) );
    // clear left
    int xHide = (int)(console.getWidth() * hidden * 0.5f);
    g.fillRect( 0, 0, xHide, console.getHeight() );
    // clear right
    g.fillRect(
        console.getWidth() - xHide, 0,
        console.getWidth(), console.getHeight()
    );
    // clear top
    int yHide = (int)(console.getHeight() * hidden * 0.5f);
    g.fillRect( 0, 0, console.getWidth(), yHide );
    // clear bottom
    g.fillRect(
        0, console.getHeight() - yHide,
        console.getWidth(), console.getHeight()
    );
}
g.dispose();
}
@Override
protected void render( Graphics g ) {
```



```
super.render( g );
Graphics2D g2d = (Graphics2D)g;
int dx = (canvas.getWidth()) / 8;
int dy = (canvas.getHeight()) / 8;
for( int i = 0; i < 8; ++i ) {
    for( int j = 0; j < 8; ++j ) {
        g2d.setColor( (i+j) % 2 == 0 ? Color.BLACK : Color.WHITE );
        g2d.fillRect( i*dx, j*dy, dx, dy );
    }
}
g2d.setComposite(
    AlphaComposite.getInstance( AlphaComposite.SRC_OVER,alpha ) );
g2d.drawImage( console, xConsole, yConsole, null );
}
public static void main( String[] args ) {
    launchApp( new ConsoleOverlayExample() );
}
}
```

## 11.4 支持线程安全的键盘输入

这时候，输入某些文本并让其可用是很不错的。然而，`KeyboardInput` 类有两个问题需要修复。第一个问题是 `KeyListener` 接口对于键盘输入事件不做任何事情。这个问题比较容易解决。不容易解决的是，如果帧速率太慢的话，检查输入的轮询方法可能会漏掉某些按键事件。

例如，如果帧速率是每 5 秒钟只轮询一次，那么重复的按键事件将会漏掉。在前面的某些示例中，你可能留意到了，帧速率已经下降到足够漏掉某些按键事件。在输入文本时，这是不能接受的。如果连单个按键都会漏掉，这对于用户来说太明显而且会令人失望。

解决方案是捕获所有的事件，并在游戏循环中遍历它们，从而不管帧速率是多少，都不会漏掉事件。`SafeKeyboardInput` 类包含一个内部类 `Event`，并且具有如下类型的一个 `EventType enum`。

- `EventType.PRESSED`
- `EventType.RELEASED`
- `EventType.TYPED`



`EventType` 对象组合了键盘事件对象，并且将其保存到 `Event` 类中。两个事件列表，`gameThread` 和 `eventThread` 用来捕获并遍历这些键盘事件。在每一个键盘监听器方法中，事件被添加到事件线程列表。这确保了没有事件会被漏掉。

`poll()`方法也简化了。`gameThread` 和 `eventThread` 列表互换，将任何的事件传入到游戏循环中，并且在游戏循环处理之前的键盘事件的同时继续进行事件。由于事件线程集合是在事件线程和游戏循环中使用的，因此访问 `eventThread` 集合的所有方法都是异步的。

当列表中的所有事件都已经处理了，并且该方法返回 `false` 之后，调用 `processInput()` 方法。在每一个事件都处理之后，检查键的状态以发现按键事件。

使用这种方法还有一个问题。如果在轮询时按下了 5 个键，那么，下一次轮询状态时，将会有 5 个事件。如果第一个事件是按下空格键，那么，该键将设置为按下次。由于没有更多的空格键事件，因此对于所有 5 个事件来说，空格键状态将是始终保持相同，看上去似乎空格键按下去了 5 次。这并不是我们想要的结果。更新了 `keyDown` 和 `keyDownOnce` 方法，以检查当前事件键的代码，从而不会处理额外的事件。最后，添加了 `getKeyTyped()` 方法，以返回按下的键，它用于在屏幕上显示输入的文本。

```
package javagames.util;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.LinkedList;
public class SafeKeyboardInput implements KeyListener {
    enum EventType {
        PRESSED,
        RELEASED,
        TYPED;
    }
    class Event {
        KeyEvent event;
        EventType type;
        public Event( KeyEvent event, EventType type ) {
            this.event = event;
            this.type = type;
        }
    }
    private LinkedList<Event> eventThread = new LinkedList<Event>();
    private LinkedList<Event> gameThread = new LinkedList<Event>();
    private Event event = null;
    private int[] polled;

    public SafeKeyboardInput() {
```



```
polled = new int[ 256 ];
}

public boolean keyDown( int keyCode ) {
    return keyCode == event.event.getKeyCode() && polled[ keyCode ] > 0;
}
public boolean keyDownOnce( int keyCode ) {
    return keyCode == event.event.getKeyCode() && polled[ keyCode ] == 1;
}
public boolean processEvent() {
    event = gameThread.poll();
    if( event != null ) {
        int keyCode = event.event.getKeyCode();
        if( keyCode >= 0 && keyCode < polled.length ) {
            if( event.type == EventType.PRESSED ) {
                polled[keyCode]++;
            } else if( event.type == EventType.RELEASED ) {
                polled[keyCode] = 0;
            }
        }
    }
    return event != null;
}
public Character getKeyTyped() {
    if( event.type != EventType.TYPED ) {
        return null;
    } else {
        return event.event.getKeyChar();
    }
}
public synchronized void poll() {
    LinkedList<Event> swap = eventThread;
    eventThread = gameThread;
    gameThread = swap;
}
public synchronized void keyPressed( KeyEvent e ) {
    eventThread.add( new Event( e, EventType.PRESSED ) );
}
public synchronized void keyReleased( KeyEvent e ) {
    eventThread.add( new Event( e, EventType.RELEASED ) );
}
public synchronized void keyTyped( KeyEvent e ) {
    eventThread.add( new Event( e, EventType.TYPED ) );
}
```



```
}
```

**SafeKeyboardInputExample** 如图 11.8 所示，位于 `javagames.text` 包中，使用新的输入类把文本输出到屏幕。为了简化测试，向 `javagames.util` 包添加了一个新的框架类 `SafeKeyboardFramework`，它使用新的 `SafeKeyboardInput` 类。由于新的键盘输入过程不同，因此如果直接在其他框架中替换输入类，将需要重构之前的所有示例。

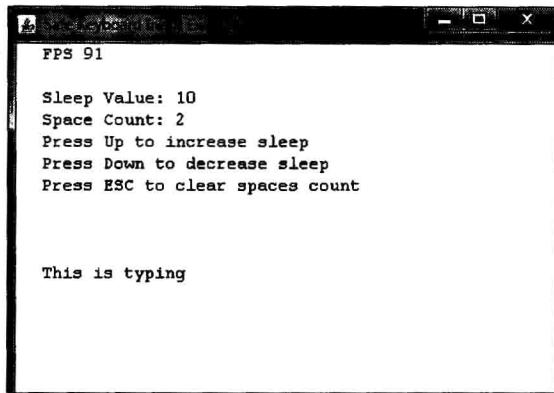


图 11.8 安全的键盘输入

`SafeKeyboardFramework` 框架与 `SimpleFramework` 完全相同，只不过用新的类替换键盘输入框架。

```

public class SafeKeyboardFramework extends JFrame implements Runnable {
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    protected FrameRate frameRate;
    protected Canvas canvas;
    protected RelativeMouseInput mouse;
    protected SafeKeyboardInput keyboard;
    //... rest of the class
}

```

使用了新的框架的任何类，其 `processInput()` 方法都需要更新，以使用新的方法来处理输入。应将事件处理包含到一个 `while` 循环中以免漏掉了事件，从而确保用新的输入处理更新了所有代码。

```

@Override
protected void processInput( float delta ) {
    super.processInput( delta );
}

```



```
    while( keyboard.processEvent() ) {
        if( keyboard.keyDownOnce( KeyEvent.VK_UP ) ) {
            appSleep += Math.min( appSleep * 2, 1000L );
        }
        if( keyboard.keyDownOnce( KeyEvent.VK_DOWN ) ) {
            appSleep -= Math.min( appSleep / 2, 1000L );
        }
        if( keyboard.keyDownOnce( KeyEvent.VK_ESCAPE ) ) {
            spacesCount = 0;
        }
        if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
            spacesCount++;
        }
        processTypedChar();
    }
}
```

向上箭头和向下箭头键调整帧速率以测试是否漏掉了按键事件。计算空格键按下的次数，并且 Escape 键会重置空格键计数。processTypedChar()方法处理输入的键，所有输入字符都在这个方法中处理，包括那些不需要显示的键，例如空格和回车，也需要处理它们。addCharater()和 removeCharacter()方法负责更新输入到屏幕上的文本。updateObject()方法负责控制文本末尾闪烁的光标。注意，在 processTypedChar()方法的末尾，重置光标闪烁。这避免了在输入文本时光标还在闪烁。

```
drawCursor = true;
blink = 0.0f;
```

render()方法绘制常用的指令，以及空格键按下的次数。这个计数用来验证没有按键被漏掉。输入的文本，使用 Utility 类绘制于指令的下方。最后，绘制闪烁的光标。

```
package javagames.text;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import javagames.util.*;

public class SafeKeyboardInputExample extends SafeKeyboardFramework {
    private int spacesCount;
    private float blink;
    private boolean drawCursor;
    private ArrayList<String> strings = new ArrayList<String>();
    public SafeKeyboardInputExample() {
```



```
appSleep = 10L;
appTitle = "Safe Keyboard Input Example";
strings.add( "" );
}

@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    while( keyboard.processEvent() ) {
        if( keyboard.keyDownOnce( KeyEvent.VK_UP ) ) {
            appSleep += Math.min( appSleep * 2, 1000L );
        }
        if( keyboard.keyDownOnce( KeyEvent.VK_DOWN ) ) {
            appSleep -= Math.min( appSleep / 2, 1000L );
        }
        if( keyboard.keyDownOnce( KeyEvent.VK_ESCAPE ) ) {
            spacesCount = 0;
        }
        if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
            spacesCount++;
        }
        processTypedChar();
    }
}
private void processTypedChar() {
    Character typedChar = keyboard.getKeyTyped();
    if( typedChar != null ) {
        if( Character.isISOControl( typedChar ) ) {
            if( KeyEvent.VK_BACK_SPACE == typedChar ) {
                removeCharacter();
            }
            if( KeyEvent.VK_ENTER == typedChar ) {
                strings.add( "" );
            }
        } else {
            addCharacter( typedChar );
        }
        drawCursor = true;
        blink = 0.0f;
    }
}
private void addCharacter( Character c ) {
    strings.add( strings.remove( strings.size() - 1 ) + c );
}
```



```
}

private void removeCharacter() {
    String line = strings.remove( strings.size() - 1 );
    if( !line.isEmpty() ) {
        strings.add( line.substring( 0, line.length() - 1 ) );
    }
    if( strings.isEmpty() ) {
        strings.add( "" );
    }
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    blink += delta;
    if( blink > 0.5f ) {
        blink -= 0.5f;
        drawCursor = !drawCursor;
    }
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    textPos = Utility.drawString( g, 20, textPos,
        "",
        "Sleep Value: " + appSleep,
        "Space Count: " + spacesCount,
        "Press Up to increase sleep",
        "Press Down to decrease sleep",
        "Press ESC to clear spaces count",
        "",
        "",
        ""
    );
    textPos = Utility.drawString( g, 20, textPos, strings );
    if( drawCursor ) {
        FontMetrics fm = g.getFontMetrics();
        int height = fm.getAscent() + fm.getDescent() + fm.getLeading();
        int y = textPos - height;
        int x = 20 + fm.stringWidth( strings.get( strings.size() - 1 ) );
        g.drawString( "_", x, y + fm.getAscent() );
    }
}
```



```
public static void main( String[] args ) {  
    launchApp( new SafeKeyboardInputExample() );  
}  
}
```

## 11.5 资源和延伸阅读

“Working with Text APIs,” 1995, <http://docs.oracle.com/javase/tutorial/2d/text/index.html>.

“Fonts and Text Layout,” 2011, <http://docs.oracle.com/javase/6/docs/technotes/guides/2d/spec/j2d-fonts.html#wp73059>.



# 第 12 章

## 线程

线程增加了软件的复杂性。它们非常复杂并且容易出错。Bug、死锁以及竞争条件可能难以发现，而日志或按步执行代码的简单操作，可能改变计时而使得 bug 不再发生。正如有一种说法，线程可以解决任何问题，除了线程太多的问题之外。尽管宁愿不使用线程，因为线程如此复杂；但是在有些情况下，线程是最好的解决方案。



当程序的两个部分等待对方完成的时候，就会发生死锁。例如，如果线程 A 等待线程 B 完成任务，而线程 B 又等待线程 A 完成任务，两个线程都将永远等待下去。

当两个线程同时更新一种资源的时候，竞争条件就发生了。如果资源没有得到保护，使得一次只有一个线程能够访问它，那么两个线程将会试图同时更新变量。这两个线程会为更新数据而竞争，并且，根据哪一个先更新，结果可能会产生难以发现的 bug。

没有办法在一章的篇幅中教授理解线程所需的所有知识。关于线程这个主题，已经有一整本的书了，并且其中的一些书也并没有介绍所有的内容。因此，如果你还没有线程编程的经验，请尽快查阅本章末尾给出的资源。

其实从本书第 1 章开始，你已经使用线程创建了一个定制的游戏循环，它与 Swing 的事件分派线程分开运行。然而，Java 中还有很多更多的类可用，它们使得多线程编程更为容易。

### 12.1 使用线程实现 Callable 任务

有一个 ExecutorService 接口可以处理 Callable 任务并返回 Future 对象。有两个确实很



不错的功能，使得 Callable 对象与 Runnable 对象之间有了差异，Callable 对象能够返回一个结果，并且它们能够捕获并重新抛出其他线程中的异常。这都是通过使用 Future 对象而成为可能的。 Callable 接口如下所示：

```
public interface Callable<V> {  
    V call() throws Exception;  
}
```

ExecutorService 包含了一个 submit 方法，它执行一次 Callable 任务并且返回一个 Future 对象。

```
public interface ExecutorService extends Executor {  
    <T> Future<T> submit( Callable<T> task );  
}
```

从执行器返回的 Future 对象有一个 get()方法，它将会阻塞直到 Callable 任务完成。它将返回从 Callable 方法所返回的结果。如果 call()方法抛出一个异常，该异常将会重新抛出，以便软件能够处理当前线程中的错误。

CallableTaskExample 位于 javagames.threads 包中，它是一个简单的程序，展示了使用 ExecutorService 执行 Callable 任务。它还展示了使用 Future 对象来获取任务的结果和异常。 CallableTaskExample 实现了 Callable<Boolean> 接口。注意， call()方法返回了一个 Boolean 结果。该任务睡眠了几个毫秒，返回 true 或 false，甚至当随机睡眠时间为 0 的时候抛出一个异常。 main()方法使用 Executors 类来创建一个新的缓存线程池：

```
ExecutorService exec = Executors.newCachedThreadPool();
```

这个缓存的线程池为每个任务创建了一个新的线程，但是，在销毁这些线程之前，先保留它们一会儿。如果提交了其他的任务，将复用缓存的线程。注意这是如何做到的：Future 对象的 get()方法没有返回，直到 Callable 任务完成；它重新抛出了 Callable 任务所抛出的任何异常。

该线程池使用 shutdown()方法来关闭。它使用 awaitTermination()方法阻塞，直到线程池中的所有线程都完成。

```
package javagames.threads;  
import java.util.Random;  
import java.util.concurrent.*;  
  
public class CallableTaskExample implements Callable<Boolean> {  
    @Override
```



```
public Boolean call() throws Exception {
    // simulate some stupid long task and maybe fail...
    Random rand = new Random();
    int seconds = rand.nextInt( 6 );
    if( seconds == 0 ) {
        // pretend there was an error
        throw new RuntimeException( "I love the new thread stuff!!! :)" );
    }
    try {
        Thread.sleep( seconds * 100 );
    } catch( InterruptedException e ) { }
    // even = true, odd = false
    return seconds % 2 == 0;
}
public static void main( String[] args ) {
    ExecutorService exec = Executors.newCachedThreadPool();
    try {
        for( int i = 0; i < 50; ++i ) {
            try {
                Future<Boolean> result =
                    exec.submit( new CallableTaskExample() );
                Boolean success = result.get();
                System.out.println( "Result: " + success );
            } catch( ExecutionException ex ) {
                Throwable throwable = ex.getCause();
                System.out.println( "Error: " + throwable.getMessage() );
            } catch( InterruptedException e ) {
                System.out.println( "Awesome! Thread was canceled" );
                e.printStackTrace();
            }
        }
    } finally {
        try {
            exec.shutdown();
            exec.awaitTermination( 10, TimeUnit.SECONDS );
            System.out.println( "Threadpool Shutdown :)" );
        } catch( InterruptedException e ) {
            // at this point, just give up...
            e.printStackTrace();
            System.exit( -1 );
        }
    }
}
```



```
}
```

## 12.2 使用线程加载文件

FileLoadingExample示例如图12.1所示，位于javagames.threads包中，它使用 Callable、Future 和 ExecutorService 对象来模拟加载一堆的文件。有3个不同的线程池，可以用来模拟加载100个文件：一个是单个的线程执行器，一个拥有最大32个线程，最后一个是无限制缓存的线程池。

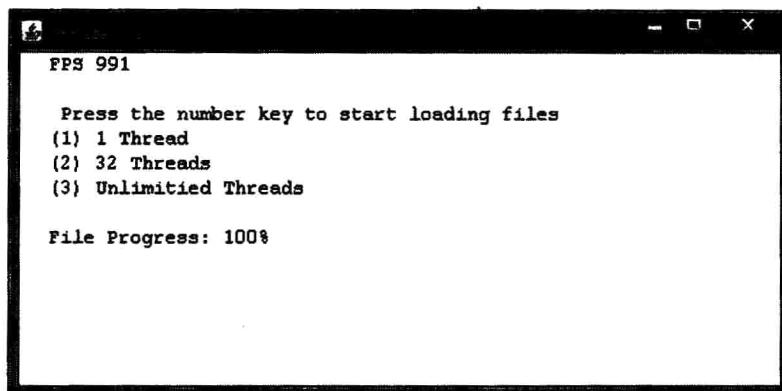


图 12.1 文件加载示例

initialize()方法创建了所有这3个线程池，并且创建了100个 callable 任务来模拟加载100个文件。processInput()方法将每个任务提交给不同的线程池。如果你按下1，将使用单个的线程池。2键使用最大32个线程的线程池。3键使用无限制的缓存线程池。

updateObject()方法遍历 Future 任务的一个列表，并且移除任何已经完成的任务。render()方法显示了常用的指令和任务完成的百分比。在应用程序关闭之前，terminate()方法关闭每个线程池。

尽管线程可能让一些任务更为复杂，但这里给出适合使用线程的一个任务的示例。

```
package javagames.threads;

import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.util.*;
```



```
import java.util.concurrent.*;
import javagames.util.*;

public class FileLoadingExample extends SimpleFramework {
    private static final int NUMBER_OF_FILES = 100;
    private ExecutorService singleThread;
    private ExecutorService thirtyTwoThreads;
    private ExecutorService unlimitedThreads;

    private boolean loading = false;
    private List<Callable<Boolean>> fileTasks;
    private List<Future<Boolean>> fileResults;
    public FileLoadingExample() {
        appWidth = 640;
        appHeight = 640;
        appSleep = 1L;
        appTitle = "File Loading Example";
        appBackground = Color.WHITE;
        appFPSColor = Color.BLACK;
    }
    @Override
    protected void initialize() {
        super.initialize();
        singleThread = Executors.newSingleThreadExecutor();
        thirtyTwoThreads = Executors.newFixedThreadPool( 32 );
        unlimitedThreads = Executors.newCachedThreadPool();
        fileTasks = new ArrayList<Callable<Boolean>>();
        for( int i = 0; i < NUMBER_OF_FILES; ++i ) {
            final int taskNumber = i;
            fileTasks.add( new Callable<Boolean>() {
                @Override
                public Boolean call() throws Exception {
                    try {
                        // pretend to load a file
                        // just sleep a little
                        Thread.sleep( new Random().nextInt(750) );
                        System.out.println( "Task: " + taskNumber );
                    } catch( InterruptedException ex ) { }
                    return Boolean.TRUE;
                }
            });
        }
        fileResults = new ArrayList<Future<Boolean>>();
    }
}
```



```
    }

@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    if( keyboard.keyDownOnce( KeyEvent.VK_1 ) ) {
        if( !loading ) {
            for( Callable<Boolean> task : fileTasks ) {
                fileResults.add( singleThread.submit( task ) );
            }
        }
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_2 ) ) {
        if( !loading ) {
            for( Callable<Boolean> task : fileTasks ) {
                fileResults.add( thirtyTwoThreads.submit( task ) );
            }
        }
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_3 ) ) {
        if( !loading ) {
            for( Callable<Boolean> task : fileTasks ) {
                fileResults.add( unlimitedThreads.submit( task ) );
            }
        }
    }
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    Iterator<Future<Boolean>> it = fileResults.iterator();
    while( it.hasNext() ) {
        Future<Boolean> next = it.next();
        if( next.isDone() ) {
            try {
                if( next.get() ) {
                    it.remove();
                }
            } catch( ExecutionException ex ) {
                ex.printStackTrace();
            } catch( InterruptedException ex ) {
                ex.printStackTrace();
            }
        }
    }
    loading = !fileResults.isEmpty();
```



```
    }
    @Override
    protected void render( Graphics g ) {
        super.render( g );
        textPos = Utility.drawString( g, 20, textPos,
            "",
            " Press the number key to start loading files",
            "(1) 1 Thread",
            "(2) 32 Threads",
            "(3) Unlimited Threads",
            ""
        );
        double percentComplete =
            (NUMBER_OF_FILES - fileResults.size()) / (double)NUMBER_OF_FILES;
        String fileProgress =
            String.format( "File Progress: %.0f%%", 100.0 * percentComplete );
        textPos = Utility.drawString( g, 20, textPos, fileProgress );
    }
    @Override
    protected void terminate() {
        super.terminate();
        shutdownExecutor( singleThread );
        shutdownExecutor( thirtyTwoThreads );
        shutdownExecutor( unlimitedThreads );
    }
    private void shutdownExecutor( ExecutorService exec ) {
        try {
            exec.shutdown();
            exec.awaitTermination( 10, TimeUnit.SECONDS );
            System.out.println( "Executor Shutdown!!!" );
        } catch( InterruptedException e ) { }
    }
    public static void main( String[] args ) {
        launchApp( new FileLoadingExample() );
    }
}
```



## 12.3 使用 FakeHardware 类测试

接下来，我们将创建一个 `FakeHardware` 类，位于 `javagames.threads` 包中，它假装使用



某些硬件。不要被其名称给混淆了。任何需要花很长时间完成的任务，例如，真正的长文件的加载、复杂的人工智能的计算，或者网络通信，都可以从这种非阻塞的类设计中得到益处。在下一章中，我们需要用相同的思路让声音类工作，因此，这里先按下不表。如果你还不完全理解这些示例，下一章中还有更多这样的示例。

**FakeHardware** 类是一个非阻塞类。如下方法中的每一个都立即返回，稍后在其他线程中执行任务。

- turnOn()
- turnoff()
- start()
- stop()

当非阻塞方法完成时，**FakeHardwareListener** 回调用来通知感兴趣的对象。

```
package javagames.threads;
import javagames.threads.FakeHardware.FakeHardwareEvent;
public interface FakeHardwareListener {
    public void event( FakeHardware source, FakeHardwareEvent event );
}
```

**FakeHardwareEvent** 是具有 4 个事件的一个 enum 类型：

```
public enum FakeHardwareEvent {
    START,
    STOP,
    ON,
    OFF;
}
```

**FakeHardware** 类包含了用 `Collections.synchronizedList()` 方法创建的监听器。这个监听器列表对于添加和删除项是线程安全的。只要在一个 `synchronized` 语句块中保护了列表迭代，就可以安全地在多线程中使用它。

`fireEvent()` 方法把完成的事件通知所有的监听器：

```
private void fireEvent( FakeHardwareEvent event ) {
    synchronized( listeners ) {
        for( FakeHardwareListener listener : listeners ) {
            listener.event( this, event );
        }
    }
}
```



`setStart()`方法有一个问题。一旦事件发送了，它调用 `runTask()`方法。该方法不能同步地完成，或者说在任务运行时，没有其他的同步语句块可以调用。然而，由于编写方法代码的方式，它不是线程安全的。如果在同步语句块之后但是在下一次 Boolean 检查之前，硬件停止了，那么任务不会运行，并且也不会发送 STOP 事件。这是一个有意的 bug，并且下一章用声音类模拟了一个类似的 bug。即便当类不是线程安全的，我们也必须以线程安全的方式使用该类，而不是修正代码。

注意所有的非阻塞方法是如何产生一个新的线程来完成任务然后立即返回的。任务实际上在另一个线程中发生，并且随后触发了监听器以通知该任务完成了。

```
package javagames.threads;
import java.util.*;

/*
 * This will simulate some fake hardware.
 */
public class FakeHardware {
    private static final int SLEEP_MIN = 100;
    private static final int SLEEP_MAX = 500;
    public enum FakeHardwareEvent {
        START,
        STOP,
        ON,
        OFF;
    }
    private volatile boolean on = false;
    private volatile boolean running = false;
    private String name;
    private List<FakeHardwareListener> listeners =
        Collections.synchronizedList(new ArrayList<FakeHardwareListener>());
    public FakeHardware( String name ) {
        this.name = name;
    }
    public boolean addListener( FakeHardwareListener listener ) {
        return listeners.add( listener );
    }
    public boolean isOn() {
        return on;
    }
    public boolean isRunning() {
        return running;
    }
}
```



```
    }
    private void sleep() {
        int rand = new Random().nextInt( SLEEP_MAX - SLEEP_MIN + 1 );
        sleep( rand + SLEEP_MIN );
    }
    private void sleep( int ms ) {
        try {
            Thread.sleep( ms );
        } catch( InterruptedException ex ) { }
    }
    public void turnOn() {
        new Thread( new Runnable() {
            public void run() {
                sleep();
                setOn();
            }
        }).start();
    }
    public void turnOff() {
        new Thread( new Runnable() {
            public void run() {
                sleep();
                setOff();
            }
        }).start();
    }
    public void start( final int timeMS, final int slices ) {
        new Thread( new Runnable() {
            public void run() {
                sleep();
                setStart( timeMS, slices );
            }
        }).start();
    }
    public void stop() {
        new Thread( new Runnable() {
            public void run() {
                sleep();
                setStop();
            }
        }).start();
    }
    private synchronized void setOn() {
```



```
if( !on ) {
    on = true;
    fireEvent( FakeHardwareEvent.ON );
}
}

private synchronized void setOff() {
    if( on ) {
        setStop();
        on = false;
        fireEvent( FakeHardwareEvent.OFF );
    }
}
/*
 * There is a problem with this method.
 * If the lock running is set to false after the lock
 * is released but before the next if statement,
 * the task will never run...
 *
 * Let's pretend this Hardware driver doesn't work well,
 * even though that NEVER happens :)
 */
private void setStart( int timeMS, int slices ) {
    synchronized( this ) {
        if( on && !running ) {
            running = true;
            fireEvent( FakeHardwareEvent.START );
        }
    }
    if( running ) {
        runTask( timeMS, slices );
        running = false;
        fireEvent( FakeHardwareEvent.STOP );
    }
}
private synchronized void setStop() {
    if( running ) {
        running = false;
        // don't send the event
        // not actually done yet :)
    }
}
private void runTask( int timeMS, int slices ) {
    int sleep = timeMS / slices;
```



```
for( int i = 0; i < slices; ++i ) {
    if( !running ) {
        return;
    }
    System.out.println( name + "[" + (i+1) + "/" + slices + "]" );
    sleep( sleep );
}
}
private void fireEvent( FakeHardwareEvent event ) {
    synchronized( listeners ) {
        for( FakeHardwareListener listener : listeners ) {
            listener.event( this, event );
        }
    }
}
}
```

## 12.4 使用等待/通知方法

**FakeHardware** 类是一个很好的例子，它使用线程创建一个非阻塞的类并使用回调监听器来通知事件完成。但是，有时候需要创建非阻塞类的一个阻塞版本。在这种情况下，`wait()` 和 `notify()` 工作得很好。

每个对象都有一个 `wait()` 方法，只要在该对象上保留一个同步的锁，则它将会阻塞直到被通知苏醒。

```
synchronized( object ) {
    while( conditionNotTrue() ) {
        object.wait();
    }
}
```

`wait()` 方法将会阻塞，直到另一个线程获得了对象锁并且通知所有的线程等待该对象。有必要使用一个 `while` 循环来检查状态，因为在某些情况下，`wait()` 方法可能苏醒，但没有任何类唤醒等待的线程。仅仅因为 `wait()` 方法返回了，并不意味着条件确保为真。要唤醒等待的线程，应使用 `notifyAll()` 方法：

```
synchronized( object ) {
    object.notifyAll();
}
```



这些都有效，因为当调用 `wait()` 方法的时候，锁释放了，只有这时候才允许其他相关的类获取锁。但是，请求一个非阻塞的类执行某些操作并且等待回调触发时，可能会用一个阻塞的版本来包装一个非阻塞的类。`WaitNotifyExample` 位于 `javagames.threads` 包中，它是使用 `FakeHardware` 类的一个简单程序，它组合了 `wait()` 和 `notifyAll()` 方法，直到每个操作都完成。

```
package javagames.threads;
import javagames.threads.FakeHardware.FakeHardwareEvent;

public class WaitNotifyExample implements FakeHardwareListener {
    public WaitNotifyExample() {
    }
    public void runTest() throws Exception {
        FakeHardware hardware = new FakeHardware( "name" );
        hardware.addListener( this );
        synchronized( this ) {
            hardware.turnOn();
            while( !hardware.isOn() ) {
                wait();
            }
        }
        System.out.println( "Hardware is on!" );
        synchronized( this ) {
            hardware.start( 1000, 4 );
            while( !hardware.isRunning() ) {
                wait();
            }
        }
        System.out.println( "Hardware is running" );
        synchronized( this ) {
            while( hardware.isRunning() ) {
                wait();
            }
        }
        System.out.println( "Hardware has stopped!" );
        synchronized( this ) {
            hardware.turnOff();
            while( hardware.isOn() ) {
                wait();
            }
        }
    }
}
```



```
@Override  
public synchronized void event(   
    FakeHardware source, FakeHardwareEvent event ) {  
    System.out.println( "Got Event: " + event );  
    notifyAll();  
}  
public static void main( String[] args ) throws Exception {  
    new WaitNotifyExample().runTest();  
}  
}
```

## 12.5 在游戏循环中使用线程

`BlockingHardware` 类位于 `javagames.threads` 包中，这是用一个阻塞版本包装 `FakeHardware` 类的一个类。然而，它使用 `Lock` 和 `Condition` 类，而不是使用 `wait/notify` 方法，这两个类是并发库的一部分。学习这些类是很好的，因为它们提供了一些 `wait/notify` 方法所没有的功能。注意，锁对象提供了用于通知的条件对象。这些对象像下面这样创建：

```
Lock lock = new ReentrantLock();  
Condition cond = lock.newCondition();
```

要等待一个对象，可以调用条件的 `await()` 方法。`Lock` 类既有 `lock()` 方法也有 `unlock()` 方法，它们分别用来获取和释放锁。使用了一个 `try/finally` 语句块来确保这个锁总是释放的。

```
lock.lock();  
try {  
    while( conditionNotTrue() ) {  
        cond.await();  
    }  
} finally {  
    lock.unlock();  
}
```

要通知等待线程，使用 `signalAll()` 方法：

```
lock.lock();  
try {  
    cond.signalAll();  
} finally {  
    lock.unlock();  
}
```



此时，你可能会问为什么任何人都使用这些类，而不是使用所有对象都可用的、更简单的 wait/notify 方法。原因在于，Lock/Condition 对象有很多方法以允许更大的灵活性。例如，wait/notify 模式使用一个同步语句块。

```
synchronized( object ) {  
    // do stuff here  
}
```

即便永久等待，这段代码也将阻塞，直到它获得锁，然而，Lock 对象有其他方法可以用来获取锁，如下所示。

- tryLock()——该方法将试图获取锁，并且如果获得的话，返回 true。然而，如果没有获得锁，它也不会阻塞，返回 false。
- tryLock( long, TimeUnit )——该方法接受一个超时值，并且在达到一定时间后，放弃获取锁的尝试。
- lockInterruptibly()——wait/notify 方法中的 synchronized 语句块并不会对线程中断做出响应。这个方法阻塞，直到获得了锁，但是，如果 Lock 对象中断的话，将返回一个异常。

此外，Condition 对象有一个超时值可供 await() 方法以及如下的方法使用。

- awaitUninterruptibly()——该方法将等待通知，并且不会对线程中断做出响应。

另一项功能是多锁类型。可以以公平策略创建一个 ReentrantLock，从而确保等待时间最长的线程能够获得锁。还有一个 ReadWriteLock 维护了两个不同的锁，从而尽管一次只有一个线程可以写对象，但有多个线程可以读取它。

BlockingHardwareListener 接口位于 javagames.threads 包中，当硬件任务完成时，使用它来进行事件通知。

```
package javagames.threads;  
  
public interface BlockingHardwareListener {  
    public void taskFinished();  
}
```

BlockingHardware 类维护了监听器的一个列表，并且当硬件任务完成时通知监听器。BlockingHardware 还维护了自己的状态变量，以防止 FakeHardware 类的状态与实际不符。

turnOn()、turnoff()、start() 和 stop() 方法都使用 Lock/Condition 对象阻塞。这使得 FakeHardware 类在游戏循环这样的单线程循环中很容易使用，并且确保竞争条件和死锁 bug 的可能性最小。



```
package javagames.threads;
import java.util.*;
import java.util.concurrent.locks.*;
import javagames.threads.FakeHardwareEvent;

public class BlockingHardware {
    private final Lock lock = new ReentrantLock();
    private final Condition cond = lock.newCondition();
    private volatile boolean on = false;
    private volatile boolean started = false;
    private FakeHardware hardware;
    private List<BlockingHardwareListener> listeners =
        Collections.synchronizedList(
            new ArrayList<BlockingHardwareListener>() );
    public BlockingHardware( String name ) {
        hardware = new FakeHardware( name );
        hardware.addListener( new FakeHardwareListener() {
            @Override
            public void event( FakeHardware source, FakeHardwareEvent event ) {
                handleHardwareEvent( source, event );
            }
        });
    }
    public boolean addListener( BlockingHardwareListener listener ) {
        return listeners.add( listener );
    }
    public void start( int ms, int slices ) {
        lock.lock();
        try {
            hardware.start( ms, slices );
            while( !started ) {
                cond.await();
            }
            System.out.println( "It's Started" );
        } catch( InterruptedException e ) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
    public void stop() {
        lock.lock();
        try {
```



```
hardware.stop();
while( started ) {
    cond.await();
}
} catch( InterruptedException ex ) {
    ex.printStackTrace();
} finally {
    lock.unlock();
}
}

public void turnOn() {
    lock.lock();
    try {
        hardware.turnOn();
        while( !on ) {
            cond.await();
        }
        System.out.println( "Turned on" );
    } catch( InterruptedException ex ) {
        ex.printStackTrace();
    } finally {
        lock.unlock();
    }
}
}

public void turnOff() {
    lock.lock();
    try {
        hardware.turnOff();
        while( on ) {
            cond.await();
        }
        System.out.println( "Turned off" );
    } catch( InterruptedException ex ) {
        ex.printStackTrace();
    } finally {
        lock.unlock();
    }
}
}

protected void handleHardwareEvent(
    FakeHardware source, FakeHardwareEvent event ) {
    boolean wasStarted = started;
    lock.lock();
    try {
```



```
        if( event == FakeHardwareEvent.ON ) {
            on = true;
        } else if( event == FakeHardwareEvent.OFF ) {
            on = false;
        } else if( event == FakeHardwareEvent.START ) {
            started = true;
        } else if( event == FakeHardwareEvent.STOP ) {
            started = false;
        }
        cond.signalAll();
    } finally {
        lock.unlock();
    }
    if( wasStarted && !started ) {
        fireTaskFinished();
    }
}
private void fireTaskFinished() {
    synchronized( listeners ) {
        for( BlockingHardwareListener listener : listeners ) {
            listener.taskFinished();
        }
    }
}
```

你已经看到了创建拥有非阻塞方法的一个类，这个方法在事件完成后通知监听器，并且你看到了以 `wait/notify` 机制用一个阻塞版本来包装一个非阻塞类。但是，尽管这个阻塞类等待直到代码完成，该方法还是要花费半秒钟。这种类无法用于主游戏循环中，因为它所花费的时间太长。这个思路适用于需要花费较长时间运行的任何代码。例如，可能 AI 部分需要数秒钟来计算下一次移动，或者大量的向导需要时间来计算策略。

不管代码做什么事情，在主游戏循环中调用这种长时间任务而不导致游戏严重暂停是不可能的。把长时间任务放到另一个线程中，但是，仍然保持这段代码容易在游戏循环中使用，要做到这一点，一种简单的方法是使用一个 `BlockingQueue`。`BlockingQueue` 是一个线程安全的类，它提供了方法从队列获取项并将项放入到队列中。尽管有很多不同的类型，具有很多不同的功能，但我们需要的只是一个简单的 `LinkedBlockingQueue`，它把项作为一个链表存储。该队列有两个重要的方法。

#### ■ void BlockingQueue<E>.put( E e )



## ■ E BlockingQueue<E>.get()

这两个方法允许 BlockingQueue 在线程之间传递消息。传递给 put()方法的项添加到了队列中，并且只要还没有达到队列的大小，该方法就立即返回。它不会等待某个其他的线程来获取值。get()方法将阻塞，直到有某些内容可用。很容易想象到这段代码使用相同的 wait/notify 方法，来等待某些内容放置到队列中，然后再让方法返回。

使用队列传递消息还有另一个有趣的方面。BlockingQueue 示例使用如下的消息：

```
enum Message {  
    MESSAGE_ONE,  
    MESSAGE_TWO,  
    MESSAGE_THREE,  
    POISON_PILL; // Quit :)  
}
```

由于 get()方法将阻塞，直到有一条消息等待，因此关闭该线程的一种方法是传递一条特殊的 POISON\_PILL 消息，以便当队列关闭线程的时候，消费者能够识别。

BlockingQueueExample 创建了一个 Producer 和 Consumer，它们都实现了一个 Callable 接口。注意，Producer 使用一个 Void 对象作为返回类型。

```
class Producer implements Callable<Void> {  
    public Void call() throws Exception {  
        return null;  
    }  
}
```

Void 类型用于那些不返回任何内容的 Callable 对象。通过这种方式，仍然会调用 Future.get()并等待 Callable 完成，并且如果有问题的话，抛出一个异常。

runTest()方法创建了 Producer 和 Consumer 任务，并且将它们提交到线程池。然后，该方法调用消费者的 get()方法，它等到 Producer 完成之后才会返回，发送 POISION\_PILL 消息，并且关闭线程。它最后停止线程池并返回。

给 Producer 一个消息计数和一个睡眠时间后，它随机地生成消息（不包括 POISION\_PILL）并发送消息，直到达到给定的数目，在每条消息之间会有睡眠。一旦 Producer 完成了，它把 POISION\_PILL 消息放到队列中并退出。

```
package javagames.threads;  
import java.util.Random;  
import java.util.concurrent.*;  
  
public class BlockingQueueExample {
```



```
class Producer implements Callable<Void> {
    private Random rand = new Random();
    private int numberOfMessages;
    private int sleep;
    private Producer( int numberOfMessages, int sleep ) {
        this.numberOfMessages = numberOfMessages;
        this.sleep = sleep;
    }
    @Override
    public Void call() throws Exception {
        Message[] messages = Message.values();
        for( int i = 0; i < numberOfMessages; ++i ) {
            try {
                // don't include last message (POISON)
                int index = rand.nextInt( messages.length - 2 ); // 0 - 2
                queue.put( messages[ index ] );
                System.out.println(
                    "PUT(" + (i+1) + ") " + messages[ index ]
                );
                sleep( sleep );
            } catch( InterruptedException ex ) { }
        }
        // All done. Shut her down...
        queue.put( messages[ messages.length - 1 ] );
        return null;
    }
}
class Consumer implements Callable<Integer> {
    private int messageCount = 0;
    @Override
    public Integer call() throws Exception {
        while( true ) {
            // take will block forever unless we do something
            Message msg = queue.take();
            messageCount++;
            System.out.println( "Received: " + msg );
            if( msg == Message.POISON_PILL ) {
                break;
            }
        }
        return new Integer( messageCount );
    }
}
```



```
enum Message {
    MESSAGE_ONE,
    MESSAGE_TWO,
    MESSAGE_THREE,
    POISON_PILL; // Quit :)
}

private ExecutorService exec;
private BlockingQueue<Message> queue;

public BlockingQueueExample() {
    exec = Executors.newCachedThreadPool();
    queue = new LinkedBlockingQueue<BlockingQueueExample.Message>();
}

public void runTest() {
    int numberOfMessages = 100;
    int sleep = 100;
    System.out.println( "Messages Sent: " + numberOfMessages );
    exec.submit( new Producer( numberOfMessages, sleep ) );
    // sleep a little
    sleep( 2000 ); // two seconds..
    try {
        // Start the consumer much later, but that's ok!
        Future<Integer> consumer = exec.submit( new Consumer() );
        try {
            System.out.println( "Messages Processed: " + consumer.get() );
        } catch( ExecutionException ex ) {
        } catch( InterruptedException ex ) { }
    } finally {
        try {
            exec.shutdown();
            exec.awaitTermination( 10, TimeUnit.SECONDS );
            System.out.println( "Threadpool Shutdown :" );
        } catch( InterruptedException e ) {
            // at this point, just give up...
            e.printStackTrace();
            System.exit( -1 );
        }
    }
}

protected void sleep( int ms ) {
    try {
        Thread.sleep( ms );
    } catch( InterruptedException ex ) { }
```

```
    }
    public static void main( String[] args ) {
        new BlockingQueueExample().runTest();
    }
}
```

## 12.6 使用状态机

在游戏循环中使用新的 `BlockingHardware` 类的一种方式是，使用状态机。状态机的思路将会在 13 章中使用，因此，如果现在不理解的话，后面会有更多的示例。状态机是根据接受到的事件来执行某种操作的一种简单的方式。经典的状态机示例是一扇门，它具有打开、关闭和锁上的状态。解锁的状态与关闭相同，因此，不包括该状态。事件可能是（例如）一个钥匙事件和一个手事件。如果一个手事件用于一扇关闭的门上，门将会转换为打开状态。手事件用于一扇开着的门上，它将会转换为关闭状态。钥匙事件用于一扇关闭的门上，它将转换为锁上的状态。手事件用于锁着的门上，或者钥匙事件用于一扇开着的门上，将不会有任何效果，如图 12.2 所示。

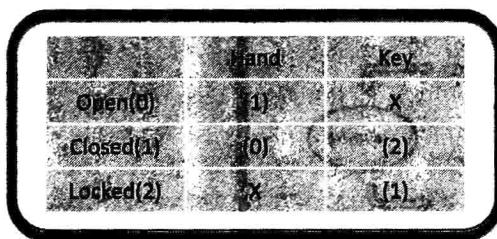


图 12.2 门状态机

初次尝试用 `FakeHardware` 类对一个状态机建模，如图 12.3 所示。

但是，这不仅复杂，而且落后。`On` 和 `Off` 应该是状态，而不是事件。但是，如果你试图在事件发送给 `FakeHardwareListener` 之后建模它，情况可能变得真的复杂了，这是因为你必须用 `BlockingHardware` 类包装 `FakeHardware`，这种转换可以按照顺序执行任务，诸如打开硬件并开始任务，如图 12.4 所示。

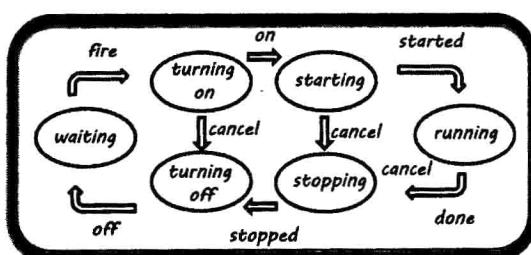


图 12.3 复杂的状态

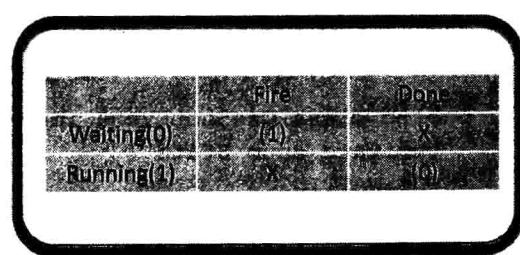


图 12.4 简单状态



如果该状态等待并接收到一个触发事件，则任务开始运行。当任务完成后——要么由于完成，要么由于被取消——状态都会返回到等待。如果描述得不够清楚，那么用一个示例来帮助说明。

## 12.7 OneShotEvent 类

OneShotEvent 如图 12.5 所示，位于 javagames.threads 包中，它是你的第一个状态机。它触发一个事件运行一次，然后等待另一次触发事件。如果事件在运行过程中取消了，它也会停止。`initialize()`方法创建了一个阻塞的队列，用于将事件传递给消费者线程，这会处理事件并转换状态机。`run()`方法一次处理队列中的一个事件。`shutdown()`方法通过传递 `DONE` 事件来停止消费者线程。

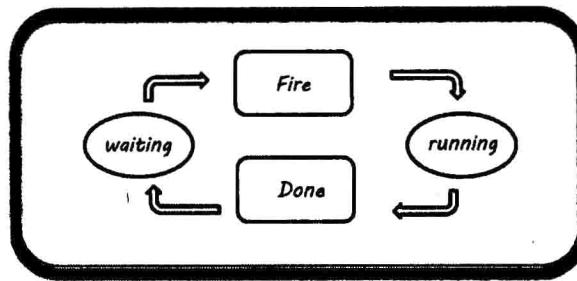


图 12.5 One Shot Event

添加了一个 `BlockingHardwareListener`，以监听完成的任务，并且发送了一个 `DONE` 事件。`processEvent()`方法根据接收到的事件来转换状态，根据请求打开或关闭硬件。

```
package javagames.threads;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

public class OneShotEvent implements Runnable {
    private enum State {
        WAITING,
        RUNNING;
    };
    private enum Event {
        FIRE,
        DONE
    };
}
```



```
DONE;
}
private BlockingQueue<Event> queue;
private BlockingHardware hardware;
private State currentState;
private Thread consumer;
private int ms;
private int slices;
public OneShotEvent( int ms, int slices ) {
    this.ms = ms;
    this.slices = slices;
}
public void initialize() {
    hardware = new BlockingHardware( "oneshot" );
    hardware.addListener( getListener() );
    queue = new LinkedBlockingQueue<Event>();
    currentState = State.WAITING; // default state
    // startup the consumer thread
    consumer = new Thread( this );
    consumer.start();
}
public void fire() {
    try {
        queue.put( Event.FIRE );
    } catch( InterruptedException e ) { }
}
public void done() {
    try {
        queue.put( Event.DONE );
    } catch( InterruptedException e ) { }
}
public void shutDown() {
    Thread temp = consumer;
    consumer = null;
    try {
        // send event to wake up consumer
        // and/or stop.
        queue.put( Event.DONE );
        temp.join( 10000L );
        System.out.println( "OneShot shutdown!!!" );
    } catch( InterruptedException ex ) { }
}
@Override
```



```
public void run() {
    while( Thread.currentThread() == consumer ) {
        try {
            processEvent( queue.take() );
        } catch( InterruptedException e ) { }
    }
}
private void processEvent( Event event ) throws InterruptedException {
    System.out.println( "Got " + event + " event" );
    if( currentState == State.WAITING ) {
        if( event == Event.FIRE ) {
            hardware.turnOn();
            hardware.start( ms, slices );
            currentState = State.RUNNING;
        }
    } else if( currentState == State.RUNNING ) {
        if( event == Event.DONE ) {
            hardware.turnOff();
            currentState = State.WAITING;
        }
    }
}
private BlockingHardwareListener getListener() {
    return new BlockingHardwareListener() {
        @Override
        public void taskFinished() {
            try {
                queue.put( Event.DONE );
            } catch( InterruptedException e ) {}
        }
    };
}
```

## 12.8 LoopEvent 类

LoopEvent 如图 12.6 所示，位于 javagames.threads 包中，它几乎和 OneShotEvent 相同。不同的是，当循环事件结束时，它重新启动并且继续循环，直到接收到一个 DONE 事件。

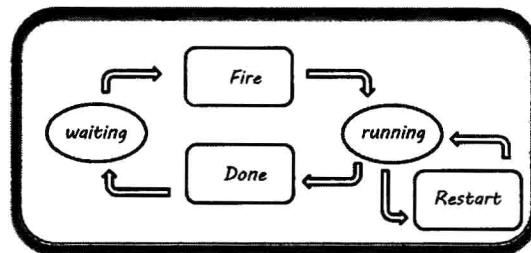


图 12.6 循环事件

如果在事件任务已经运行时，向它发送了多个 fire() 事件，这些事件将会直接被忽略。

```
package javagames.threads;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
public class LoopEvent implements Runnable {
    private enum State {
        WAITING,
        RUNNING;
    };
    private enum Event {
        FIRE,
        RESTART,
        DONE;
    }
    private BlockingQueue<Event> queue;
    private BlockingHardware hardware;
    private State currentState;
    private Thread consumer;
    private int ms;
    private int slices;
    public LoopEvent( int ms, int slices ) {
        this.ms = ms;
        this.slices = slices;
    }
    public void initialize() {
        hardware = new BlockingHardware( "looper" );
        hardware.addListener( getListener() );
        queue = new LinkedBlockingQueue<Event>();
        currentState = State.WAITING; // default state
        // startup the consumer thread
        consumer = new Thread( this );
        consumer.start();
    }
}
```



```
}

public void fire() {
    try {
        queue.put( Event.FIRE );
    } catch( InterruptedException e ) { }
}

public void done() {
    try {
        queue.put( Event.DONE );
    } catch( InterruptedException e ) { }
}

public void shutDown() {
    Thread temp = consumer;
    consumer = null;
    try {
        // send event to wake up consumer
        // and/or stop.
        queue.put( Event.DONE );
        temp.join( 10000L );
        System.out.println( "Loop shutdown!!!" );
    } catch( InterruptedException ex ) { }
}

@Override
public void run() {
    while( Thread.currentThread() == consumer ) {
        try {
            processEvent( queue.take() );
        } catch( InterruptedException e ) { }
    }
}

private void processEvent( Event event ) throws InterruptedException {
    System.out.println( "Got " + event + " event" );
    if( currentState == State.WAITING ) {
        if( event == Event.FIRE ) {
            hardware.turnOn();
            hardware.start( ms, slices );
            currentState = State.RUNNING;
        }
    } else if( currentState == State.RUNNING ) {
        if( event == Event.RESTART ) {
            hardware.start( ms, slices );
            currentState = State.RUNNING;
        }
    }
}
```



```
if( event == Event.DONE ) {
    hardware.stop();
    hardware.turnOff();
    currentState = State.WAITING;
}
}
private BlockingHardwareListener getListener() {
    return new BlockingHardwareListener() {
        @Override
        public void taskFinished() {
            try {
                queue.put( Event.RESTART );
            } catch( InterruptedException e ) {}
        }
    };
}
}
```

## 12.9 RestartEvent 类

RestartEvent 如图 12.7 所示，位于 `javagames.threads` 包中，它要更加复杂一些。如果 `RestartEvent` 处在等待状态，一个 FIRE 事件将启动任务。

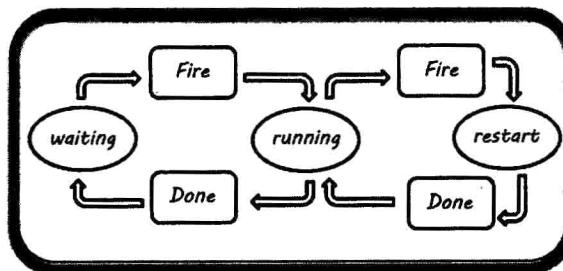


图 12.7 重新启动事件

当 `RestartEvent` 完成的时候，它将会停止。如果在 `RestartEvent` 运行时，另一个 FIRE 事件发送给了它，`RestartEvent` 将重新启动（其名称由此而来）。重新启动状态是必须的，这样当任务完成时，状态机将知道要重新启动任务了。如果任务完成了并且状态是



RUNNING，状态会变换为 WAITING。如果任务完成了，并且状态是 RESTART，它会重新启动任务并且状态设置会变为 RUNNING，而不是停止任务。

```
package javagames.threads;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
public class RestartEvent implements Runnable {
    private enum State {
        WAITING,
        RESTART,
        RUNNING;
    };
    private enum Event {
        FIRE,
        DONE;
    }
    private BlockingQueue<Event> queue;
    private BlockingHardware hardware;
    private State currentState;
    private Thread consumer;
    private int ms;
    private int slices;
    public RestartEvent( int ms, int slices ) {
        this.ms = ms;
        this.slices = slices;
    }
    public void initialize() {
        hardware = new BlockingHardware( "restart" );
        hardware.addListener( getListener() );
        queue = new LinkedBlockingQueue<Event>();
        currentState = State.WAITING; // default state
        // start up the consumer thread
        consumer = new Thread( this );
        consumer.start();
    }
    public void fire() {
        try {
            queue.put( Event.FIRE );
        } catch( InterruptedException e ) { }
    }
    public void shutDown() {
        Thread temp = consumer;
        consumer = null;
```



```
try {
    // send event to wake up consumer
    // and/or stop.
    queue.put( Event.DONE );
    temp.join( 10000L );
    System.out.println( "Restart shutdown!!!" );
} catch( InterruptedException ex ) { }

}

@Override
public void run() {
    while( Thread.currentThread() == consumer ) {
        try {
            processEvent( queue.take() );
        } catch( InterruptedException e ) { }
    }
}

private void processEvent( Event event ) throws InterruptedException {
    System.out.println( "Got " + event + " event" );
    if( currentState == State.WAITING ) {
        if( event == Event.FIRE ) {
            hardware.turnOn();
            hardware.start( ms, slices );
            currentState = State.RUNNING;
        }
    } else if( currentState == State.RUNNING ) {
        if( event == Event.FIRE ) {
            hardware.stop();
            currentState = State.RESTART;
        }
        if( event == Event.DONE ) {
            hardware.turnOff();
            currentState = State.WAITING;
        }
    } else if( currentState == State.RESTART ) {
        if( event == Event.DONE ) {
            hardware.start( ms, slices );
            currentState = State.RUNNING;
        }
    }
}

private BlockingHardwareListener getListener() {
    return new BlockingHardwareListener() {
        @Override
```



```
    public void taskFinished() {  
        try {  
            queue.put( Event.DONE );  
        } catch( InterruptedException e ) {}  
    }  
};  
}  
}
```

## 12.10 多线程事件示例

MultiThreadEventExample 如图 12.8 所示，位于 javagames.threads 包中，它是一个简单的游戏循环，使用 3 种不同的状态类。

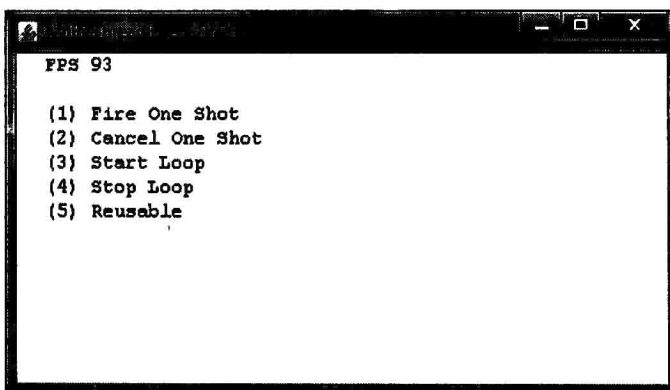


图 12.8 多线程事件示例

每个状态机都在其自己的线程中运行，使用阻塞队列沟通来自游戏线程的事件。每个事件包含一个 BlockingHardware 类，它包含了 FakeHardware 类，并且等待甚至在更多线程中运行的任务。尽管游戏循环代码很简单，但这里还是有很多事情发生。花时间来理解这个示例中的工具是值得的。

```
package javagames.threads;  
  
import java.awt.Graphics;  
import java.awt.event.KeyEvent;  
  
import javagames.util.SimpleFramework;  
import javagames.util.Utility;
```



```
public class MultiThreadEventExample extends SimpleFramework {  
    private OneShotEvent oneShot;  
    private LoopEvent loop;  
    private RestartEvent restart;  
    public MultiThreadEventExample() {  
        appWidth = 640;  
        appHeight = 640;  
        appSleep = 10L;  
        appTitle = "Multi-Thread Event Example";  
        appBackground = Color.WHITE;  
        appFPSColor = Color.BLACK;  
    }  
    @Override  
    protected void initialize() {  
        super.initialize();  
        oneShot = new OneShotEvent( 5000, 10 );  
        oneShot.initialize();  
        loop = new LoopEvent( 1000, 4 );  
        loop.initialize();  
        restart = new RestartEvent( 5000, 10 );  
        restart.initialize();  
    }  
    @Override  
    protected void processInput( float delta ) {  
        super.processInput( delta );  
        if( keyboard.keyDownOnce( KeyEvent.VK_1 ) ) {  
            oneShot.fire();  
        }  
        if( keyboard.keyDownOnce( KeyEvent.VK_2 ) ) {  
            oneShot.done();  
        }  
        if( keyboard.keyDownOnce( KeyEvent.VK_3 ) ) {  
            loop.fire();  
        }  
        if( keyboard.keyDownOnce( KeyEvent.VK_4 ) ) {  
            loop.done();  
        }  
        if( keyboard.keyDownOnce( KeyEvent.VK_5 ) ) {  
            restart.fire();  
        }  
    }  
    @Override
```



```
protected void render( Graphics g ) {
    super.render( g );
    textPos = Utility.drawString( g, 20, textPos,
        "",
        "(1) Fire One Shot",
        "(2) Cancel One Shot",
        "(3) Start Loop",
        "(4) Stop Loop",
        "(5) Reusable"
    );
}
@Override
protected void terminate() {
    super.terminate();
    oneShot.shutDown();
    loop.shutDown();
    restart.shutDown();
}
public static void main( String[] args ) {
    launchApp( new MultiThreadEventExample() );
}
}
```

本章介绍了很多基础知识。在第 13 章中，所有这些概念都用来为 Java 声音库创建一个包装 API。这里所介绍的所有思路，包括产生线程来处理逻辑，然后将那些调用包装到一个阻塞库中，以便它们可以在单线程的游戏循环中使用，在第 13 章也会详细地展示。可能需要花很长的时间才能熟悉多线程编程，因此，如果你还没有掌握，也不要担心。下一章应该会有所帮助（并且别忘了，查看下面那些资源）。

## 12.11 资源和延伸阅读

Goetz, Brian, *Java Concurrency in Practice*, Addison-Wesley, Upper Saddle River, 2006.

Oaks, Scott, Henry Wong, *Java Threads*, 3<sup>rd</sup> Edition, O'Reilly, Sebastopol, CA, 2004.



# 第 13 章

## 声音

Java Sound API 是较底层的库，用于播放及录制声音及 MIDI 文件。尽管声音库涉及方方面面，但本章只专注于播放\*.wav 文件。我们将要使用的这个库的部分，其结构如图 13.1 所示。

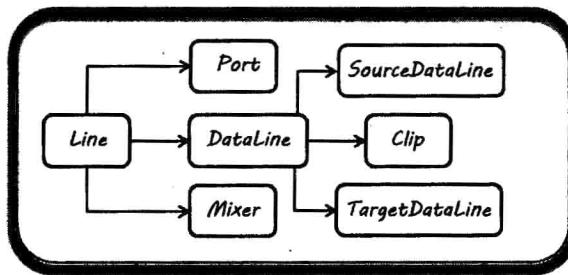


图 13.1 Sound API

所有的类都从基类 **Line** 继承。抽象类分为 3 个不同的子类。一个 **Port** 对象表示物理的声音输入和输出，例如扬声器、耳机插孔、麦克风等。**Mixer** 对象拥有输入输出行，并且可以混合输入数据，并创建新的输出数据。这个类是根据录音工作室中的声音混合器而建模的。最后是 **DataLine**，它表示传递给其他类的一些音频数据。

**DataLine** 包含 3 个子类。这是该 API 令人混淆的地方，因此，这里先不介绍。有一个 **SourceDataLine**、一个 **TargetDataLine** 和一个 **Clip**。通常，源是输入，目标是输出，但是，由于数据行是以混合器为中心的，因此名称也似乎反过来了，参见图 13.2。

要把数据写入到混合器，数据写出到源中，并且从目标读入（实际上是，从源读出，写入到目标中）。再读一次。为什么这么描述，我也不知道，但却很容易令人混淆。

- **SourceDataLine = OutputStream**
- **TargetDataLine = InputStream**

好消息是 **SourceDataLine** 拥有唯一的 **write()** 方法，并且 **TargetDataLine** 拥有唯一的 **read()**



方法，因此，要用错它们是不可能的。只记住其名称的意义不大。Clip 类似乎也不太对。

当数据行将数据流出到其他行或者从其他行流入数据时，Clip 类是 SourceDataLine（输出）的一个包装类，而 SourceDataLine 用于完全适合内存的较小的声音文件。因此，尽管一个 Clip 对象应该包含一个 SourceDataLine，但它不是 SourceDataLine，这就是该 API 的组织方式。

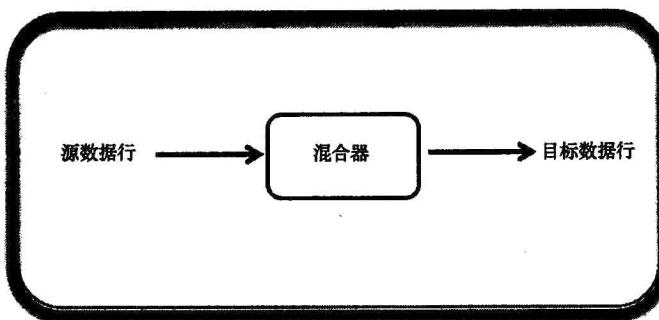


图 13.2 输入/输出问题

这个 API 还通过使用 LineListener 接口，设计为非阻塞的 API。大多数声音 API 调用立即返回，当任务完成时在另一个线程上发送一个更新事件。

```
public void LineListener.update( LineEvent )
```

LineEvent.Type 拥有如下的事件类型：

- OPEN
- CLOSE
- START
- STOP

这些事件的实际起因是根据环境而生成的。例如，当一个 SourceDataLine 停止流数据时或者当 stop()方法调用时，发送一个 STOP 事件。添加一个监听器的代码如下所示：

```
Clip clip = magicalJavaCode();
clip.addLineListener( new LineListener() {
    @Override
    public void update( LineEvent event ) {
        // TODO handle line event
    }
});
```



现在还不用担心创建一个 Clip 的代码。我们将稍后介绍。如下的方法用来播放声音。

- `open()`——用于打开一个数据行。由于最大有 32 个行可用，因此每行必须在使用之前打开。
- `start()`——调用从一行上开始，这一行允许数据发送到由另一端处理的行上。例如，把数据写到一行，不会播放数据，除非这一行已经开始了。
- `stop()`——数据行有两个内部缓冲：当前播放的数据帧，以及等待播放的一个缓冲。当数据行停止时，当前播放的数据完成，并且没有新的数据播放。然而，中间缓冲中留下的任何数据仍然在那里。再次调用 `start()`，将继续从数据停止的位置开始播放。
- `drain()`——这个方法阻塞，直到所有可以播放的缓冲数据都已经播放了，并且没有数据留下。该方法用于在停止声音之前播放所有的声音数据。调用 `stop()`方法而不先使用 `drain()`，将会立即切断声音，并且将数据留在缓冲中。
- `flush()`——该方法从声音缓冲中删除任何保留的数据。如果声音已经停止但数据仍然保留，`flush()`方法将清除缓冲。如果在停止声音之后剩下了数据，并且没有使用该方法，当声音重新启动之后，最后一点声音仍然会播放。
- `close()`——该方法释放声音行，并且使其可以重用。

## 13.1 操作声音文件

声音 API 只处理几种格式：`*.wav`、`*.aiff` 和`*.au`。如果没有第三方软件的话，将不支持更多其他常用的格式。

声音文件有几个属性很重要。每个文件包含了一些采样。每个采样或帧都是某个特定时间点的声音。声音文件有一个帧速率，或者采样率，以赫兹为单位。例如，采样率为 44 100HZ，每秒钟将会有 44 100 个采样。另一个重要的属性是每个采样的位数。一个 8 位的声音文件，按照 8 位数据（或 1 个字节）来存储帧，而 16 位的声音文件按照两个字节来存储声音。通道的数目也很重要。每一帧中的每个通道，只有一个数据采样。这意味着，一个 16 位带有两个通道（立体声）的声音，每帧拥有 16 位+16 位（或者说 32 位）的数据。

$$\text{frame size in bytes} = \frac{\text{bits per frame}}{8} * \text{number of channels}$$



稍后，当需要创建流数据的缓冲大小时，声音数据的格式以及计算每一帧的字节数就显得很重要的。现在，只需要知道，对于每一种不同的声音文件类型，所有这些属性都是不同的。要播放一个音频文件，需要一个 `AudioInputStream`。就像大多数的 `InputStream` 对象一样，这些流对象的问题在于，它们只能够使用一次。要为一个游戏重复地使用声音，需要从文件提取原始声音字节。它们可能包含在一个 `ByteArrayInputStream` 中，并且用于创建一个 `AudioInputStream`，而不用每次播放声音时从一个 `InputStream` 流出数据。

```
InputStream soundFile =
    ResourceLoader.load( Object.class, "filepath", "respath" );
byte[] rawBytes = null;
try {
    BufferedInputStream buf = new BufferedInputStream( soundFile );
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    int read;
    while( (read = buf.read()) != -1 ) {
        out.write( read );
    }
    soundFile.close();
    rawBytes = out.toByteArray();
} catch( IOException ex ) {
    ex.printStackTrace();
}

ByteArrayInputStream audioStream =
    new ByteArrayInputStream( rawBytes );
AudioInputStream audioInputStream
    = AudioSystem.getAudioInputStream( audioStream );
```

正如你在前面的代码示例中所见到的，`AudioSystem` 是声音 API 所提供的一个辅助类。一旦一个 `AudioInputStream` 可用了，就可以打开一个剪辑并且准备播放声音。

```
InputStream soundFile = loadSoundFile();
byte[] rawBytes = readBytes( soundFile );
ByteArrayInputStream in = new ByteArrayInputStream( rawBytes );
AudioInputStream audioInputStream =
    AudioSystem.getAudioInputStream( in );
Clip clip = AudioSystem.getClip();
clip.open( audioInputStream );
```

有些声音太大了，无法放入到 `Clip` 缓冲中。非常大的声音，如环境噪音或音乐，需要使用一个 `SourceDataLine` 输出行。由于这个行不能打开音频文件，所以需要打开该行时需要音频格式。



```
InputStream soundFile = loadSoundFile();
byte[] rawBytes = readBytes( soundFile );
ByteArrayInputStream in = new ByteArrayInputStream( rawBytes );
AudioInputStream audioInputStream =
    AudioSystem.getAudioInputStream( in );
AudioFormat audioFormat = audioInputStream.getFormat();
DataLine.Info info = new DataLine.Info(
    SourceDataLine.class,
    audioFormat,
    AudioSystem.NOT_SPECIFIED
);
SourceDataLine dataLine = (SourceDataLine)AudioSystem.getLine( info );
dataLine.open( audioFormat );
```

注意，使用 `AudioFormat` 对象而不是音频文件来打开 `SourceDataLine`。如果没有传入格式而打开了该行，则会使用用户系统的默认行来打开它，而该格式可能与声音文件不兼容。

## 13.2 使用声音库的问题

现在，在开始创建下一个美妙的音乐游戏之前，声音库需要和游戏循环整合，这正是麻烦开始的地方。声音库中的一些方法会在另一个线程中发送通知事件，但实际上事件还需要花些时间才能真正地完成，因此，很容易不正确地使用该库。让我们来看一个简单的循环，它开始并停止一个剪辑声音。

```
for( int i = 0; i < 10; ++i ) {
    clip.start();
    while( !clip.isActive() ) {
        Thread.sleep(100);
    }
    clip.stop();
    clip.flush();
    clip setFramePosition(0);
    clip.start();
    clip.drain();
}
```

这段代码开始一个剪辑，然后睡眠以等待该剪辑激活。此时，剪辑会重新启动。首先



它会停止，缓冲中剩下的数据也会被清空，帧位置设置回到开始，然后剪辑开始。最后，剪辑用完，因此在下一次循环迭代之前，将播放整个声音。

在我的计算机上，使用最新的 Java 6.0 SDK 时，这个循环在几次迭代之后锁了起来。为什么会锁起来我并不清楚，但是应该与在剪辑重新启动之前的 drain() 方法阻塞有关。不管怎样，没有抛出错误，也没有迹象表明出错了。

使用声音库的另一个问题是，Java 6.0 和 Java 7.0 之间存在差异。尽管前面的例子，在 Java 6.0 中发生了锁定，但 Java 7.0 已经替代了软件混合器，并且其行为也不同了。在测试示例时，注意 Java 6.0 和 Java 7.0 之间的差异。

使用该库的试验，揭示了发送给 LineListener 接口的事件，例如 START 和 STOP，必须在对该库采取任何其他操作之前接受。因此，尽管很多的方法立即返回，但软件必须阻塞或轮询，等待事件然后才能继续。这使得在多线程的 Swing 应用程序中，该库很容易使用，而在单线程游戏循环中，它却非常难以使用。在上一章中介绍的 wait/notify 线程符号，通过阻塞非阻塞的 API 以使得声音库不会死锁，从而解决了这个问题。

PlayingClipsExample 示例，位于 javagames.sound 包中，它展示了使用第 12 章所介绍的 wait/notify 机制，播放声音而不等待、锁定声音 API 以及在阻塞响应的同时播放声音。该示例代码实现了 LineListener 接口，以便当接收到事件的时候，状态存储到 open 和 started 变量中，并且等待通知线程以唤醒。

readBytes() 方法把原始声音文件字节读入到一个数组中以易于复用。runTestWithoutWaiting()方法（可能会死锁）展示了使用声音 API 的错误方式。由于它在继续之前没有等待 LineListener 事件，因此它可能会死锁，耗尽一个不会播放的声音。runWithWaiting()展示了使用 wait/notify 机制来等待 LineListener 事件，然后再继续进行。

最后，main()方法运行两个测试。即便你能够运行两个测试而不会死锁，但其他人可能会有一台太快或太慢的机器（就像我一样），而没有人会喜欢声音停止播放的情况。

```
package javagames.sound;
import java.io.*;
import javagames.util.ResourceLoader;
import javax.sound.sampled.*;

public class PlayingClipsExample implements LineListener {
    private volatile boolean open = false;
    private volatile boolean started = false;
    public byte[] readBytes( InputStream in ) {
        try {
            BufferedInputStream buf = new BufferedInputStream( in );

```



```
ByteArrayOutputStream out = new ByteArrayOutputStream();
int read;
while( (read = buf.read()) != -1 ) {
    out.write( read );
}
in.close();
return out.toByteArray();
} catch( IOException ex ) {
    ex.printStackTrace();
return null;
}
}

public void runTestWithoutWaiting() throws Exception {
System.out.println( "runTestWithoutWaiting()" );
Clip clip = AudioSystem.getClip();
clip.addLineListener( this );
InputStream resource = ResourceLoader.load(
    PlayingClipsExample.class,
    "res/assets/sound/WEAPON_scifi_fire_02.wav",
    "notneeded"
);
byte[] rawBytes = readBytes( resource );
ByteArrayInputStream in = new ByteArrayInputStream( rawBytes );
AudioInputStream audioInputStream =
    AudioSystem.getAudioInputStream( in );
clip.open( audioInputStream );
for( int i = 0; i < 10; ++i ) {
    clip.start();
    while( !clip.isActive() ) {
        Thread.sleep(100);
    }
    clip.stop();
    clip.flush();
    clip.setFramePosition(0);
    clip.start();
    clip.drain();
}
clip.close();
}

public void runTestWithWaiting() throws Exception {
System.out.println( "runTestWithWaiting()" );
Clip clip = AudioSystem.getClip();
clip.addLineListener( this );
```



```
InputStream resource = ResourceLoader.load(
    PlayingClipsExample.class,
    "res/assets/sound/WEAPON_scifi_fire_02.wav",
    "notneeded"
);
byte[] rawBytes = readBytes( resource );
ByteArrayInputStream in = new ByteArrayInputStream( rawBytes );
in = new ByteArrayInputStream( rawBytes );
AudioInputStream audioInputStream =
    AudioSystem.getAudioInputStream( in );
synchronized( this ) {
    clip.open( audioInputStream );
    while( !open ) {
        wait();
    }
}
for( int i = 0; i < 10; ++i ) {
    clip.setFramePosition( 0 );
    synchronized( this ) {
        clip.start();
        while( !started ) {
            wait();
        }
    }
    clip.drain();
    synchronized( this ) {
        clip.stop();
        while( started ) {
            wait();
        }
    }
}
synchronized( this ) {
    clip.close();
    while( open ) {
        wait();
    }
}
@Override
public synchronized void update( LineEvent lineEvent ) {
    System.out.println( "Got Event: " + lineEvent.getType() );
    LineEvent.Type type = lineEvent.getType();
```



```
if( type == LineEvent.Type.OPEN ) {
    open = true;
} else if( type == LineEvent.Type.START ) {
    started = true;
} else if( type == LineEvent.Type.STOP ) {
    started = false;
} else if( type == LineEvent.Type.CLOSE ) {
    open = false;
}
notifyAll();
}
public static void main( String[] args ) throws Exception {
    PlayingClipsExample lineListenerExample = new PlayingClipsExample();
    lineListenerExample.runTestWithWaiting();
    lineListenerExample.runTestWithoutWaiting();
}
}
```

## 13.3 开发阻塞音频类

你打算开发一组简单的类，它们可以用来将声音添加到游戏循环中。第一个问题是，创建阻塞类，它在返回之前等待声音库事件。这里使用一个阻塞类包装一个非阻塞类的技术，即第 12 章中介绍的 `wait/notify` 方法。

`BlockingAudioListener` 接口位于 `javagames.sound` 包中，当一个声音完成播放时，它用于通知监听器。

```
package javagames.sound;
public interface BlockingAudioListener {
    public void audioFinished();
}
```

`SoundException` 类位于同样的包中，用做一个定制的异常，它扩展了 `RuntimeException`，以便不需要特别捕获声音 API 异常（有很多这样的异常）。

```
package javagames.sound;
public class SoundException extends RuntimeException {
    public SoundException( String message ) {
        super( message );
    }
}
```



```
    }
    public SoundException( String message, Throwable cause ) {
        super( message, cause );
    }
}
```

AudioStream 是用于阻塞音频流的一个基类，它包装了一个 Clip 和 SourceDataLine。它使用 Lock 和 Condition 变量来监控 LineListener，而不是使用 wait/notify。它还管理 BlockingAudioListener 接口的监听器，并且提供方法来开始、停止、打开、关闭、重新开始以及循环声音。LineListener 包含的代码，可以在声音完成时触发事件，并且根据条件变量标记所有的线程等待。

```
package javagames.sound;

import java.util.*;
import java.util.concurrent.locks.*;
import javax.sound.sampled.*;
import javax.sound.sampled.LineEvent.*;

public abstract class AudioStream implements LineListener {
    public static final int LOOP_CONTINUOUSLY = -1;
    protected final Lock lock = new ReentrantLock();
    protected final Condition cond = lock.newCondition();
    protected volatile boolean open = false;
    protected volatile boolean started = false;
    protected byte[] soundData;
    private List<BlockingAudioListener> listeners = Collections
        .synchronizedList( new ArrayList<BlockingAudioListener>() );
    public AudioStream( byte[] soundData ) {
        this.soundData = soundData;
    }
    public abstract void open();
    public abstract void close();
    public abstract void start();
    public abstract void loop( int count );
    public abstract void restart();
    public abstract void stop();
    public boolean addListener( BlockingAudioListener listener ) {
        return listeners.add( listener );
    }
    protected void fireTaskFinished() {
        synchronized( listeners ) {
            for( BlockingAudioListener listener : listeners ) {
```



```
        listener.audioFinished();
    }
}
}

@Override
public void update( LineEvent event ) {
    boolean wasStarted = started;
    lock.lock();
    try {
        if( event.getType() == Type.OPEN ) {
            open = true;
        } else if( event.getType() == Type.CLOSE ) {
            open = false;
        } else if( event.getType() == Type.START ) {
            started = true;
        } else if( event.getType() == Type.STOP ) {
            started = false;
        }
        cond.signalAll();
    } finally {
        lock.unlock();
    }
    if( wasStarted && !started ) {
        fireTaskFinished();
    }
}
}
```

## 13.4 用阻塞的 Clip 类

BlockingClip 类扩展了 AudioStream 基类，并且提供了如下抽象方法的实现。

- open
- close
- start
- stop
- restart



## ■ loop

当 Clip 对象执行任务时，必须阻塞的每一个方法具有如下的签名：

```
lock.lock();
try {
    // use clip
    while( not finished ) {
        cond.await();
    }
} finally {
    lock.unlock();
}
```

正如第 12 章所介绍的，在请求剪辑执行某些动作之前，应先获取锁，然后等待一个响应，并总是在一个 try/finally 语句块中解锁。这确保了不管方法中发生什么，该锁都会释放。

open()方法创建了 AudioInputStream，从 AudioSystem 获取一个 Clip，将其自身添加为 LineListener，打开该剪辑并且阻塞，直到 LineListener 接口接受 LineEvent.OPEN 事件。注意所有的异常，UnsupportedAudioFileException、LineUnavailableExcetion 和 IOException，都用定制的声音异常包装了起来。这使得该类更容易使用。

start()方法使用 flush()清除了任何剩余数据的剪辑，将帧的位置设置为 0，开始该 Clip，并且在返回之前等待 LineEvent.START 事件。

loop()方法类似于 start()方法，但是它使用 Clip.loop()方法传入一个循环计数。使用 AudioStream.LOOP\_CONTINUOUSLY 标志，将允许声音永久循环。

stop()方法停止 Clip，并且等待 LineEvent.STOP 事件。它不需要先耗尽该行。Clip 类将播放，直到声音完成然后调用 BlockingAudio.audioFinished()方法。

restart()方法有点复杂。它覆盖了 fireTaskFinished()方法，挂起该事件，以便当声音停止并再次开始时，和或接收到 LineEvent.STOP 事件时，fireTaskFinished()不会错误地通知监听器任务已经停止，而实际上任务只是重新启动了。

close()方法关闭 Clip，并且等待 LineEvent.CLOSE 事件，然后才返回。

```
package javagames.sound;
import java.io.*;
import javax.sound.sampled.*;
public class BlockingClip extends AudioStream {
    private Clip clip;
    private boolean restart;
```



```
public BlockingClip( byte[] soundData ) {
    super( soundData );
}
/*
 * This guy could throw a bunch of exceptions.
 * We're going to wrap them all in a custom exception
 * handler that is a RuntimeException so we don't
 * have to catch and throw all these exceptions.
*/
@Override
public void open() {
    lock.lock();
    try {
        ByteArrayInputStream in = new ByteArrayInputStream( soundData );
        AudioInputStream ais = AudioSystem.getAudioInputStream( in );
        clip = AudioSystem.getClip();
        clip.addLineListener( this );
        clip.open( ais );
        while( !open ) {
            cond.await();
        }
        System.out.println( "open" );
    } catch( UnsupportedAudioFileException ex ) {
        throw new SoundException( ex.getMessage(), ex );
    } catch( LineUnavailableException ex ) {
        throw new SoundException( ex.getMessage(), ex );
    } catch( IOException ex ) {
        throw new SoundException( ex.getMessage(), ex );
    } catch( InterruptedException ex ) {
        ex.printStackTrace();
    } finally {
        lock.unlock();
    }
}
@Override
public void start() {
    lock.lock();
    try {
        clip.flush();
        clip setFramePosition( 0 );
        clip.start();
        while( !started ) {
            cond.await();
        }
    } catch( LineUnavailableException ex ) {
        throw new SoundException( ex.getMessage(), ex );
    }
}
```



```
        }
        System.out.println( "It's Started" );
    } catch( InterruptedException e ) {
        e.printStackTrace();
    } finally {
        lock.unlock();
    }
}
@Override
public void loop( int count ) {
    lock.lock();
    try {
        clip.flush();
        clip.setFramePosition( 0 );
        clip.loop( count );
        while( !started ) {
            cond.await();
        }
        System.out.println( "It's Started" );
    } catch( InterruptedException e ) {
        e.printStackTrace();
    } finally {
        lock.unlock();
    }
}
@Override
public void restart() {
    restart = true;
    stop();
    restart = false;
    start();
}
@Override
protected void fireTaskFinished() {
    if( !restart ) {
        super.fireTaskFinished();
    }
}
@Override
public void stop() {
    lock.lock();
    try {
        clip.stop();
    }
```



```
        while( started ) {
            cond.await();
        }
    } catch( InterruptedException ex ) {
    ex.printStackTrace();
} finally {
    lock.unlock();
}
}

@Override
public void close() {
    lock.lock();
    try {
        clip.close();
        while( open ) {
            cond.await();
        }
        clip = null;
        System.out.println( "Turned off" );
    } catch( InterruptedException ex ) {
    ex.printStackTrace();
} finally {
    lock.unlock();
}
}
}
```

## 13.5 使用 AudioDataLine 类

对于声音库需要做的下一件事情是，创建一个 `BlockingDataLine` 类，以便可以手动流播放声音。记住，Java 6.0 SDK 和 Java 7.0 SDK 存在行为差异。尽管在 6.0 中，`Clip` 类不能加载大文件，但是在 7.0 中，可以加载和播放大文件。然而，尽管较小的剪辑在 6.0 中用得很好，但在 7.0 中使用它们则可能导致较短的声音无法工作。因此，不管你支持哪个版本，手动流播放声音的能力都是需要的。

`AudioDataLine` 类位于 `javagames.sound` 包中，类似于 `Clip` 类，但是它可以流化较大的声音文件。这种方式播放声音的问题之一是，声音数据需要以字节块为单位写入 `SourceDataLine` 中。要做到这一点，必须从 `AudioInputStream` 提取实际的音频数据。原始



声音文件不能播放，或者文件头信息将解释为声音数据。我尝试过，听起来不是很好。有很多杂音和爆音。

可以在帧中，从 `AudioInputStream` 中提取声音数据。一次只读取一个字节的数据是不可能的，除非每一帧只有一个字节的长度。

```
InputStream soundFile = loadSoundFile();
byte[] rawBytes = readBytes( soundFile );
ByteArrayInputStream in = new ByteArrayInputStream( rawBytes );
AudioInputStream audioInputStream =
    AudioSystem.getAudioInputStream( in );
AudioFormat audioFormat = audioInputStream.getFormat();
ByteArrayOutputStream out = new ByteArrayOutputStream();
int chunk = (int)audioFormat.getFrameSize();
byte[] buf = new byte[ chunk ];
while( audioInputStream.read( buf ) != -1 ) {
    out.write( buf );
}
audioInputStream.close();
byte[] soundBytes = out.toByteArray();
```

从头开始将数据流入到 `SourceDataLine` 的另一个问题是缓冲的大小。和 `Clip` 不同，写入到 `SourceDataLine` 的数据应该在声音停止之前耗尽。为了得到更快的响应，缓冲必须较小；一个 50 毫秒的缓冲似乎工作得很好。由于声音有不同的采样大小、频道以及位采样大小，因此计算一个 50 毫秒的缓冲字节大小需要做一些工作。

`computeBufferSize()`方法使用毫秒、采样大小、采样位数以及通道数为单位的缓冲长度，来创建字节大小正确的一个缓冲。

$$\text{frame size} = \frac{\text{sample rate Hz}}{1000} * \text{milliseconds}$$

通过将采样速率转换为毫秒，可以计算出一个 50 毫秒的采样的帧数。将其除以 1000，然后与想要的毫秒数相乘，从而做到这一点。

$$\text{bytes per frame} = \frac{\text{sample bits}}{1000} * \text{channels}$$

通过将采样位数转换为字节，从而计算出每一帧的字节大小。将采样位数除以 8，然后将其与通道数计算得到的字节数相乘，从而做到这一点。

$$\text{buffer size} = \text{frame size} * \text{bytes per frame}$$

`initialize()`方法创建 `AudioInputStream`，获取打开 `SourceDataLine` 所需的 `AudioFormat`，



计算 50 毫秒的缓冲大小，并且从 `AudioInputStream` 中提取声音数据。`readSoundData()` 方法负责提取音频数据。`computeBufferSize()` 方法计算一个 50 毫秒采样的正确的缓冲大小。还要注意，检查缓冲以确保它可以除以 1000，并且，如果采样大小不能平均地除开的话，还需要进行调整。

```
double temp = milliseconds;
double frames = sampleRate * temp / 1000.0;
while( frames != Math.floor( frames ) ) {
    temp++;
    frames = sampleRate * temp / 1000.0;
}
```

`open()`方法需要一个 `SoundDataLine`。注意，计算的缓冲大小传递给了 `open()`方法，覆盖了默认的大小。这个类还维持了 `LineListener` 对象的一个列表。由于在获取行之后，需要添加监听器，因此当行打开之后，添加 `LineListener` 对象。

`start()`和`loop()`方法清空了任何旧的数据行，开始了该行，然后触发了另一个线程来遍历所提取的音频数据，并一次向数据行写出一个缓冲。

`stop()`方法通过将引用设置为 `null`，从而停止了写线程。注意，该方法并不会在数据行上调用 `stop()`。停止数据行，将会阻止当前缓冲的数据继续播放，导致偶然出现爆音或杂音。任何写入的声音数据，在行停止之前都需要耗尽，这就是为什么需要较小的缓冲大小。否则的话，声音会在停止时间之后还播放很长一段时间，这会导致延迟。

`reset()`方法直接设置一个标志，告诉 `write` 方法从头开始重新启动流数据。

`run()`方法是最为复杂的，它流化了较小的数据块，支持循环和重新启动，然后继续确保循环没有停止。`run()`方法继续写数据，直到循环值为 0。当循环值设置为 -1 时，将永远循环。如果循环停止或者循环计数达到了 0，该行耗尽了，那么停止，并且线程退出。如果声音重新启动了，那么循环计数递增，从而当声音重启时，循环计数保持相同。

```
package javagames.sound;
import java.io.*;
import java.util.*;
import javax.sound.sampled.*;

public class AudioDataLine implements Runnable {
    private static final int BUFFER_SIZE_MS = 50;
    private List<LineListener> listeners =
        Collections.synchronizedList( new ArrayList<LineListener>() );
    private Thread writer;
    private AudioFormat audioFormat;
```



```
private SourceDataLine dataLine;
private byte[] rawData;
private byte[] soundData;
private int bufferSize;
private int loopCount;
private volatile boolean restart = false;
public AudioDataLine( byte[] rawData ) {
    this.rawData = rawData;
}
public void initialize() {
    try {
        ByteArrayInputStream in = new ByteArrayInputStream( rawData );
        AudioInputStream ais = AudioSystem.getAudioInputStream( in );
        audioFormat = ais.getFormat();
        bufferSize = computeBufferSize( BUFFER_SIZE_MS );
        soundData = readSoundData( ais );
    } catch( UnsupportedAudioFileException ex ) {
        throw new SoundException( ex.getMessage(), ex );
    } catch( IOException ex ) {
        throw new SoundException( ex.getMessage(), ex );
    }
}
private byte[] readSoundData( AudioInputStream ais ) {
    try {
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        long chunk = audioFormat.getFrameSize();
        byte[] buf = new byte[ (int)chunk ];
        while( ais.read( buf ) != -1 ) {
            out.write( buf );
        }
        ais.close();
        return out.toByteArray();
    } catch( IOException ex ) {
        ex.printStackTrace();
        return null;
    }
}
public void addLineListener( LineListener listener ) {
    listeners.add( listener );
}
public void open() {
    try {
        DataLine.Info info =
```



```
    new DataLine.Info( SourceDataLine.class, audioFormat,
                      AudioSystem.NOT_SPECIFIED );
    dataLine = (SourceDataLine)AudioSystem.getLine( info );
    synchronized( listeners ) {
        for( LineListener listener : listeners ) {
            dataLine.addLineListener( listener );
        }
    }
    dataLine.open( audioFormat, bufferSize );
} catch( LineUnavailableException ex ) {
    throw new SoundException( ex.getMessage(), ex );
}
}

private int computeBufferSize( int milliseconds ) {
    double sampleRate = audioFormat.getSampleRate();
    double bitSize = audioFormat.getSampleSizeInBits();
    double channels = audioFormat.getChannels();
    System.out.println( "Sample Rate: " + sampleRate );
    System.out.println( "Bit Size: " + bitSize );
    System.out.println( "Channels: " + channels );
    System.out.println( "Milliseconds: " + milliseconds );
    if( bitSize == AudioSystem.NOT_SPECIFIED ||
        sampleRate == AudioSystem.NOT_SPECIFIED ||
        channels == AudioSystem.NOT_SPECIFIED ) {
        System.out.println( "BufferSize: " + -1 );
        return -1;
    } else {
        double temp = milliseconds;
        double frames = sampleRate * temp / 1000.0;
        while( frames != Math.floor( frames ) ) {
            temp++;
            frames = sampleRate * temp / 1000.0;
        }
        double bytesPerFrame = bitSize / 8.0;
        double size = (int)(frames * bytesPerFrame * channels);
        System.out.println( "BufferSize: " + size );
        return (int)size;
    }
}

public void close() {
    dataLine.close();
}

public void start() {
```



```
loopCount = 0;
dataLine.flush();
dataLine.start();
writer = new Thread( this );
writer.start();
}
public void reset() {
    restart = true;
}
public void loop( int count ) {
    loopCount = count;
    dataLine.flush();
    dataLine.start();
    writer = new Thread( this );
    writer.start();
}
public void stop() {
    if( writer != null ) {
        Thread temp = writer;
        writer = null;
        try {
            temp.join( 10000 );
        } catch( InterruptedException ex ) { }
    }
}
public Line getLine() {
    return dataLine;
}
@Override
public void run() {
    System.out.println( "write stream" );
    try {
        while( true ) {
            int written = 0;
            int length =
                bufferSize == -1 ? dataLine.getBufferSize() : bufferSize;
            while( written < soundData.length ) {
                if( Thread.currentThread() != writer ) {
                    System.out.println( "Stream canceled" );
                    loopCount = 0;
                    break; // stop writing data
                } else if( restart ) {
                    restart = false;
                }
            }
        }
    }
}
```



```
        System.out.println( "Stream canceled" );
        if( loopCount != AudioStream.LOOP_CONTINUOUSLY ) {
            loopCount++;
        }
        break; // stop writing data
    }
    int bytesLeft = soundData.length - written;
    int toWrite = bytesLeft > length * 2 ? length : bytesLeft;
    written += dataLine.write( soundData, written, toWrite );
}
if( loopCount == 0 ) {
    break;
} else if( loopCount != AudioStream.LOOP_CONTINUOUSLY ) {
    loopCount--;
}
}
} catch( Exception e ) {
    e.printStackTrace();
} finally {
    System.out.println( "Stream finished" );
    dataLine.drain();
    dataLine.stop();
}
}
}
```

## 13.6 BlockingDataLine 类

AudioDataStream 包装了一个 SourceDataLine，并且提供了和 Clip 相似的类，现在可以创建一个 BlockingDataLine，它在返回之前等待声音事件。这为较小和较大的声音文件创建了阻塞类。

BlockingDataLine 使用与 BlockClip 相同的结构。

```
lock.lock();
try {
    // use clip
    while( not finished ) {
        cond.await();
```



```
    }
} finally {
    Lock.unlock();
}
```

open()方法创建了一个新的 AudioDataLine，添加了一个 LineListener，打开了流，并且阻塞直到其完成。start()方法启动了流，并且等待直到数据开始流化然后才返回。loop()方法做同样的事情。

restart()方法转发了对 AudioDataLine 的请求。stop()方法停止了流，并且在返回之前等待 stop()事件。close()方法关闭流。

```
package javagames.sound;

public class BlockingDataLine extends AudioStream {
    private AudioDataLine stream;
    public BlockingDataLine( byte[] soundData ) {
        super( soundData );
    }
    @Override
    public void open() {
        lock.lock();
        try {
            stream = new AudioDataLine( soundData );
            stream.initialize();
            stream.addLineListener( this );
            stream.open();
            while( !open ) {
                cond.await();
            }
            System.out.println( "open" );
        } catch( InterruptedException ex ) {
            ex.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
    @Override
    public void start() {
        lock.lock();
        try {
            stream.start();
            while( !started ) {
```



```
        cond.await();
    }
    System.out.println( "started" );
} catch( InterruptedException ex ) {
    ex.printStackTrace();
} finally {
    lock.unlock();
}
}
@Override
public void loop( int count ) {
    lock.lock();
    try {
        stream.loop( count );
        while( !started ) {
            cond.await();
        }
        System.out.println( "started" );
    } catch( InterruptedException ex ) {
        ex.printStackTrace();
    } finally {
        lock.unlock();
    }
}
@Override
public void restart() {
    stream.reset();
}
@Override
public void stop() {
    lock.lock();
    try {
        stream.stop();
        while( started ) {
            cond.await();
        }
        System.out.println( "stopped" );
    } catch( InterruptedException ex ) {
        ex.printStackTrace();
    } finally {
        lock.unlock();
    }
}
```



```
@Override
public void close() {
    lock.lock();
    try {
        stream.close();
        while( open ) {
            cond.await();
        }
        System.out.println( "closed" );
    } catch( InterruptedException ex ) {
        ex.printStackTrace();
    } finally {
        lock.unlock();
    }
}
}
```

## 13.7 创建一个 SoundEvent 类

好在这个问题在第 12 章中介绍过了。通过使用阻塞队列方法，你可以创建一个 SoundEvent 类，它在另一个线程中运行游戏循环，并且在一个单个的线程队列中处理声音事件。参考第 12 章的 BlockingQueueExample，以了解使用阻塞队列进行线程间通信的详细说明。

```
package javagames.sound;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

public class SoundEvent implements Runnable {
    public static final String SHUT_DOWN = "shutdown";
    protected AudioStream audio;
    protected BlockingQueue<String> queue;
    private Thread consumer;
    public SoundEvent( AudioStream audio ) {
        this.audio = audio;
    }
    public void initialize() {
        audio.addListener( getListener() );
    }
}
```



```
queue = new LinkedBlockingQueue<String>();
consumer = new Thread( this );
consumer.start();
}
public void put( String event ) {
    try {
        queue.put( event );
    } catch( InterruptedException e ) { }
}
public void shutDown() {
    Thread temp = consumer;
    consumer = null;
    try {
        // send event to wake up consumer
        // and/or stop.
        queue.put( SHUT_DOWN );
        temp.join( 10000L );
        System.out.println( "Event shutdown!!!!" );
    } catch( InterruptedException ex ) { }
}
@Override
public void run() {
    while( Thread.currentThread() == consumer ) {
        try {
            processEvent( queue.take() );
        } catch( InterruptedException e ) { }
    }
}
protected void processEvent( String event )
throws InterruptedException { }
protected void onAudioFinished() { }
private BlockingAudioListener getListener() {
    return new BlockingAudioListener() {
        @Override
        public void audioFinished() {
            onAudioFinished();
        }
    };
}
}
```



## 13.8 使用 OneShotEvent 类

OneShotEvent 如图 13.3 所示,位于 javagames.sound 包中,它有 WAITING 和 RUNNING 两个状态,以及 FIRE 和 DONE 两个事件。

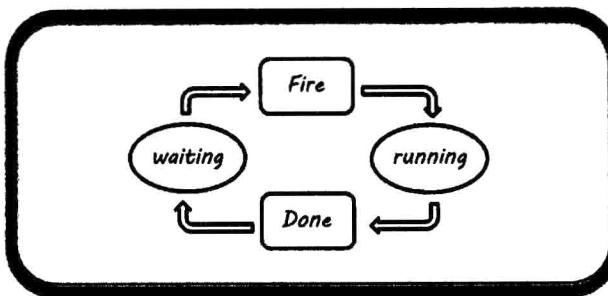


图 13.3 One Shot 事件

如果 OneShotEvent 在等待,那么触发事件开始播放声音。在播放声音时,触发事件什么也不做。DONE 事件将停止声音,或者当声音完成播放时将会停止。这个状态机的概念,在第 12 章中详细介绍过。

```
package javagames.sound;

public class OneShotEvent extends SoundEvent {
    public static final String STATE_WAITING = "waiting";
    public static final String STATE_RUNNING = "running";
    public static final String EVENT_FIRE = "fire";
    public static final String EVENT_DONE = "done";
    private String currentState;
    public OneShotEvent( AudioStream audio ) {
        super( audio );
        currentState = STATE_WAITING;
    }
    public void fire() {
        put( EVENT_FIRE );
    }
    public void done() {
        put( EVENT_DONE );
    }
}
```



```
}

protected void processEvent( String event ) throws InterruptedException {
    System.out.println( "Got " + event + " event" );
    if( currentState == STATE_WAITING ) {
        if( event == EVENT_FIRE ) {
            audio.open();
            audio.start();
            currentState = STATE_RUNNING;
        }
    } else if( currentState == STATE_RUNNING ) {
        if( event == EVENT_DONE ) {
            audio.stop();
            audio.close();
            currentState = STATE_WAITING;
        }
    }
}
@Override
protected void onAudioFinished() {
    put( EVENT_DONE );
}
}
```

## 13.9 使用 LoopEvent 类

LoopEvent 如图 13.4 所示，位于 javagames.sound 包中，它与 OneShotEvent 很相似。

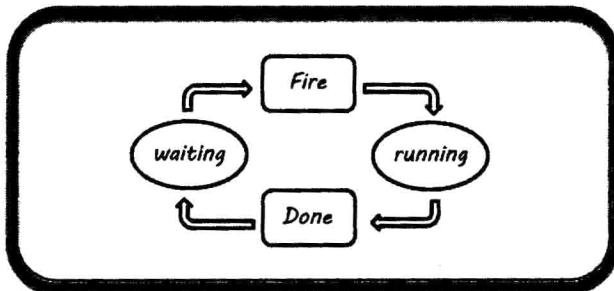


图 13.4 LoopEvent

唯一不同在于，声音将保持重复，直到调用 DONE 方法。该事件自身不会停止。



```
package javagames.sound;

public class LoopEvent extends SoundEvent {
    public static final String STATE_WAITING = "waiting";
    public static final String STATE_RUNNING = "running";
    public static final String EVENT_FIRE = "fire";
    public static final String EVENT_DONE = "done";
    private String currentState;
    public LoopEvent( AudioStream audio ) {
        super( audio );
        currentState = STATE_WAITING;
    }
    public void fire() {
        put( EVENT_FIRE );
    }
    public void done() {
        put( EVENT_DONE );
    }
    protected void processEvent( String event ) throws InterruptedException {
        System.out.println( "Got " + event + " event" );
        if( currentState == STATE_WAITING ) {
            if( event == EVENT_FIRE ) {
                audio.open();
                audio.loop( AudioStream.LOOP_CONTINUOUSLY );
                currentState = STATE_RUNNING;
            }
        } else if( currentState == STATE_RUNNING ) {
            if( event == EVENT_DONE ) {
                audio.stop();
                audio.close();
                currentState = STATE_WAITING;
            }
        }
    }
}
```

## 13.10 使用 RestartEvent 类

RestartEvent 如图 13.5 所示，它只播放声音一次，但是，当声音播放的时候，另一个



触发事件将会重新启动声音。

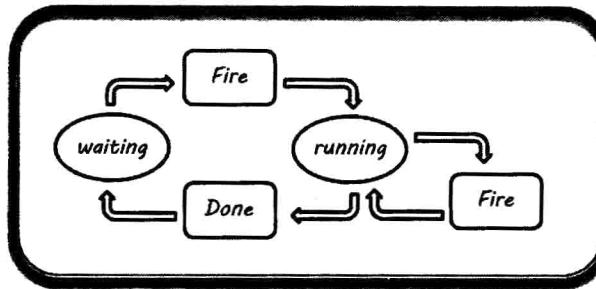


图 13.5 Restart Event

```

package javagames.sound;

public class RestartEvent extends SoundEvent {
    public static final String STATE_WAITING = "waiting";
    public static final String STATE_RUNNING = "running";
    public static final String EVENT_FIRE = "fire";
    public static final String EVENT_DONE = "done";
    private String currentState;
    public RestartEvent( AudioStream stream ) {
        super( stream );
        currentState = STATE_WAITING;
    }
    public void fire() {
        put( EVENT_FIRE );
    }
    protected void processEvent( String event ) throws InterruptedException {
        System.out.println( "Got " + event + " event" );
        if( currentState == STATE_WAITING ) {
            if( event == EVENT_FIRE ) {
                currentState = STATE_RUNNING;
                audio.open();
                audio.start();
            }
        } else if( currentState == STATE_RUNNING ) {
            if( event == EVENT_FIRE ) {
                audio.restart();
            }
            if( event == EVENT_DONE ) {
                currentState = STATE_WAITING;
                audio.close();
            }
        }
    }
}
  
```



```
        }
    }

    @Override
    protected void onAudioFinished() {
        put( EVENT_DONE );
    }
}
```

这 3 个事件 ( `oneshot`、`loop` 和 `restart` ) 既可以针对小的数据，也可以针对大的数据，提供计算机游戏所需的很多声音。 `SoundPlayerExample` 代码如图 13.6 所示，它位于 `javagames.sound` 包中，展示了使用新的声音类把声音添加到游戏循环中。

`initialize()` 方法加载了两个声音：一个武器爆炸的声音和一个下雨的声音。`readBytes()` 方法从文件中提取了原始的声音字节。`loadWaveFile()` 方法创建了所有的事件对象。对于较短的剪辑，该事件使用一个 `BlockingClip`。对于较大的数据，使用 `BlockingDataLine`。这个示例对两种类型的声音都使用一个较小的声音进行测试。`shutDownClips()` 方法在返回之前停止了每一个声音事件。

`processInput()` 方法总是处理按键事件。`F1` 和 `F2` 键加载两种不同的声音。数字 `1` 键启动 `one-shot` 剪辑，而数字 `2` 键停止 `one-shot` 剪辑。数字 `3` 键启动剪辑循环，数字 `4` 键停止它。数字 `5` 键触发重新启动剪辑。数字 `6`、`7`、`8`、`9` 和 `0` 键对于数据行流做同样的事情。`render()` 方法显示常用的指令。

现在，我们很少使用的是 `terminate()` 方法（我一开始认为这个方法并不需要）。在程序关闭的时候，该方法关闭所有的事件线程。

```
package javagames.sound;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.io.*;
import javagames.util.*;

public class SoundPlayerExample extends SimpleFramework {
    private OneShotEvent oneShotClip;
    private LoopEvent loopClip;
```

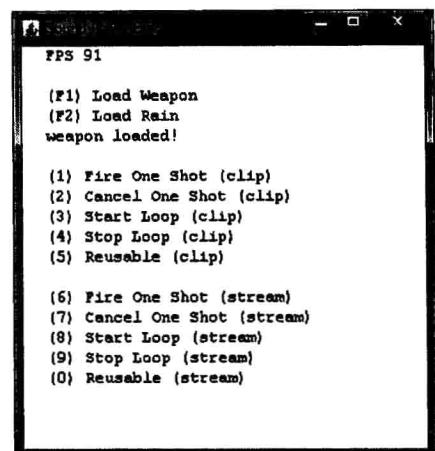


图 13.6 声音播放器示例



```
private RestartEvent restartClip;
private OneShotEvent oneShotStream;
private LoopEvent loopStream;
private RestartEvent restartStream;
private byte[] weaponBytes;
private byte[] rainBytes;
private String loaded;
public SoundPlayerExample() {
    appWidth = 340;
    appHeight = 340;
    appSleep = 10L;
    appTitle = "Sound Player Example";
    appBackground = Color.WHITE;
    appFPSColor = Color.BLACK;
}
@Override
protected void initialize() {
    super.initialize();
    InputStream in = ResourceLoader.load(
        SoundPlayerExample.class,
        "./res/assets/sound/WEAPON_scifi_fire_02.wav",
        "asdf"
    );
    weaponBytes = readBytes( in );
    in = ResourceLoader.load(
        SoundPlayerExample.class,
        "./res/assets/sound/WEATHER_rain_medium_5k.wav",
        "asdf"
    );
    rainBytes = readBytes( in );
    loadWaveFile( weaponBytes );
    loaded = "weapon";
}
private byte[] readBytes( InputStream in ) {
    try {
        BufferedInputStream buf = new BufferedInputStream( in );
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        int read;
        while( (read = buf.read()) != -1 ) {
            out.write( read );
        }
        in.close();
        return out.toByteArray();
    }
}
```



```
        } catch( IOException ex ) {
            ex.printStackTrace();
            return null;
        }
    }
    private void loadWaveFile( byte[] rawData ) {
        shutDownClips();
        oneShotClip = new OneShotEvent( new BlockingClip( rawData ) );
        oneShotClip.initialize();
        loopClip = new LoopEvent( new BlockingClip( rawData ) );
        loopClip.initialize();
        restartClip = new RestartEvent( new BlockingClip( rawData ) );
        restartClip.initialize();
        oneShotStream = new OneShotEvent( new BlockingDataLine( rawData ) );
        oneShotStream.initialize();
        loopStream = new LoopEvent( new BlockingDataLine( rawData ) );
        loopStream.initialize();
        restartStream = new RestartEvent( new BlockingDataLine( rawData ) );
        restartStream.initialize();
    }
    private void shutDownClips() {
        if( oneShotClip != null ) oneShotClip.shutDown();
        if( loopClip != null ) loopClip.shutDown();
        if( restartClip != null ) restartClip.shutDown();
        if( oneShotStream != null ) oneShotStream.shutDown();
        if( loopStream != null ) loopStream.shutDown();
        if( restartStream != null ) restartStream.shutDown();
    }
    @Override
    protected void processInput( float delta ) {
        super.processInput( delta );
        if( keyboard_KeyDownOnce( KeyEvent.VK_F1 ) ) {
            loadWaveFile( weaponBytes );
            loaded = "weapon";
        }
        if( keyboard_KeyDownOnce( KeyEvent.VK_F2 ) ) {
            loadWaveFile( rainBytes );
            loaded = "rain";
        }
        if( keyboard_KeyDownOnce( KeyEvent.VK_1 ) ) {
            oneShotClip.fire();
        }
        if( keyboard_KeyDownOnce( KeyEvent.VK_2 ) ) {
```



```
        oneShotClip.done();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_3 ) ) {
        loopClip.fire();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_4 ) ) {
        loopClip.done();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_5 ) ) {
        restartClip.fire();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_6 ) ) {
        oneShotStream.fire();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_7 ) ) {
        oneShotStream.done();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_8 ) ) {
        loopStream.fire();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_9 ) ) {
        loopStream.done();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_0 ) ) {
        restartStream.fire();
    }
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    textPos = Utility.drawString( g, 20, textPos,
        "",
        "(F1) Load Weapon",
        "(F2) Load Rain",
        loaded + " loaded!",
        "",
        "(1) Fire One Shot (clip)",
        "(2) Cancel One Shot (clip)",
        "(3) Start Loop (clip)",
    );
}
```



```
    " (4) Stop Loop (clip)",
    " (5) Reusable (clip)",
    "",
    " (6) Fire One Shot (stream)",
    " (7) Cancel One Shot (stream)",
    " (8) Start Loop (stream)",
    " (9) Stop Loop (stream)",
    " (0) Reusable (stream)"
);
}

@Override
protected void terminate() {
    super.terminate();
    shutDownClips();
}
public static void main( String[] args ) {
    launchApp( new SoundPlayerExample() );
}
}
```

## 13.11 添加声音控件

几乎就要完成了。剩下的唯一事情是添加声音控件。有如下的控件可用。

```
Boolean: MUTE
        APPLY_REVERB
Enum: REVERB
Float: AUX_RETURN
      AUX_SEND
      BALANCE
      MASTER_GAIN
      PAN
      REVERB_RETURN
      REVERB_SEND
      SAMPLE_RATE
      VOLUME
```

并不是所有的行都支持所有控件，因此，在使用控件之前，检查其是否可用是很重要的。如下的代码检查 `MASTER_GAIN` 空间，并获知该控件是否可用。



```
if( line.isControlSupported( FloatControl.Type.MASTER_GAIN ) ) {  
    FloatControl gainControl =  
        (FloatControl)line.getControl( FloatControl.Type.MASTER_GAIN );  
}
```

MASTER\_GAIN 控件将调整音量，而 PAN 控件将确定哪个扬声器的声音可以听到。如果声音不是单声道声音，而是一个立体声，使用 BALANCE 控件。更新后的 AudioStream 代码添加了对 GAIN 和 PAN 控件的支持。

```
package javagames.sound;  
  
import java.util.*;  
import java.util.concurrent.locks.*;  
import javax.sound.sampled.*;  
import javax.sound.sampled.LineEvent.*;  
  
public abstract class AudioStream implements LineListener {  
    public static final int LOOP_CONTINUOUSLY = -1;  
    protected final Lock lock = new ReentrantLock();  
    protected final Condition cond = lock.newCondition();  
    protected volatile boolean open = false;  
    protected volatile boolean started = false;  
    // UPDATES  
    protected FloatControl gainControl;  
    protected FloatControl panControl;  
    // UPDATES  
    protected byte[] soundData;  
    private List<BlockingAudioListener> listeners =  
        Collections.synchronizedList(  
            new ArrayList<BlockingAudioListener>() );  
    public AudioStream( byte[] soundData ) {  
        this.soundData = soundData;  
    }  
    public abstract void open();  
    public abstract void close();  
    public abstract void start();  
    public abstract void loop( int count );  
    public abstract void restart();  
    public abstract void stop();  
    public boolean addListener( BlockingAudioListener listener ) {  
        return listeners.add( listener );  
    }  
    protected void fireTaskFinished() {
```



```
synchronized( listeners ) {
    for( BlockingAudioListener listener : listeners ) {
        listener.audioFinished();
    }
}
@Override
public void update( LineEvent event ) {
    boolean wasStarted = started;
    lock.lock();
    try {
        if( event.getType() == Type.OPEN ) {
            open = true;
        } else if( event.getType() == Type.CLOSE ) {
            open = false;
        } else if( event.getType() == Type.START ) {
            started = true;
        } else if( event.getType() == Type.STOP ) {
            started = false;
        }
        cond.signalAll();
    } finally {
        lock.unlock();
    }
    if( wasStarted && !started ) {
        fireTaskFinished();
    }
}
//UPDATES
public void clearControls() {
    gainControl = null;
    panControl = null;
}
public void createControls( Line line ) {
    if( line.isControlSupported( FloatControl.Type.MASTER_GAIN ) ) {
        gainControl =
            (FloatControl)line.getControl( FloatControl.Type.MASTER_GAIN );
    }
    if( line.isControlSupported( FloatControl.Type.PAN ) ) {
        panControl =
            (FloatControl)line.getControl( FloatControl.Type.PAN );
    }
}
```



```
    }
    public boolean hasGainControl() {
        return gainControl != null;
    }
    public void setGain( float fGain ) {
        if( hasGainControl() ) {
            gainControl.setValue( fGain );
        }
    }
    public float getGain() {
        return hasGainControl() ? gainControl.getValue() : 0.0f;
    }
    public float getMaximum() {
        return hasGainControl() ? gainControl.getMaximum() : 0.0f;
    }
    public float getMinimum() {
        return hasGainControl() ? gainControl.getMinimum() : 0.0f;
    }
    public boolean hasPanControl() {
        return panControl != null;
    }
    public float getPrecision() {
        return hasPanControl() ? panControl.getPrecision() : 0.0f;
    }
    public float getPan() {
        return hasPanControl() ? panControl.getValue() : 0.0f;
    }
    public void setPan( float pan ) {
        if( hasPanControl() ) {
            panControl.setValue( pan );
        }
    }
}
//UPDATES
```

BlockingClip 和 BlockingDataLine 类的 open() 和 close() 方法也有更新：

```
// add to both open() methods
while( !open ) {
    cond.await();
}
//UPDATE
```



```
createControls( clip );
//UPDATE
System.out.println( "open" );
// add to both close() methods
clip.close();
while( open ) {
    cond.await();
}
clip = null;
//UPDATE
clearControls();
//UPDATE
System.out.println( "Turned off" );
```

尽管有很多不同的控件，但它们大多数都是面向混合器的，并且有一些甚至不能工作。因此，尽管将它们都添加似乎很好，但使用专业的编辑软件编辑声音或者只是使用声音 API 播放它们，这样会更容易一些。参见本章末尾给出的关于声音编辑软件的相关信息。

SoundControlsExample 如图 13.7 所示，位于 javagames.sound 包中，它使用一个计算机蜂鸣声以及带有一个 Clip 和 SourceDataLine 的循环事件。数字键可以打开或关闭该循环，并且字母键在播放声音时可调整音量和平衡。

声音控件示例的另一个问题是，只有在使用 Java 6.0 SDK 时，PAN 控件才可用。Java 7.0 中对混合器的新的实现，不再支持 PAN 控件。如果使用 7.0 版本，示例的 PAN 功能将无法工作。

注意，PAN 空间有一个值范围 (-1, 1)，其中 -1 表示左扬声器，1 表示右扬声器，0 表示两个扬声器。精度决定了 PAN 控件所支持的小数位数。

```
package javagames.sound;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.io.*;
import javagames.util.*;
public class SoundControlsExample extends SimpleFramework {
```

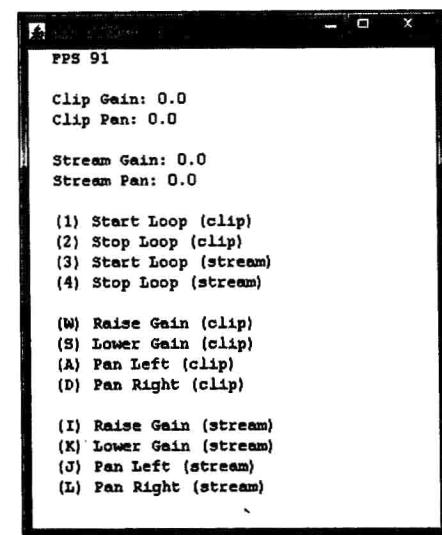


图 13.7 声音控件示例



### 13.11 添加声音



```
private BlockingClip clip;
private LoopEvent loopClip;
private BlockingDataLine dataLine;
private LoopEvent loopStream;
private byte[] rawSound;
public SoundControlsExample() {
    appWidth = 340;
    appHeight = 400;
    appSleep = 10L;
    appTitle = "Sound Controls Example";
    appBackground = Color.WHITE;
    appFPSColor = Color.BLACK;
}
@Override
protected void initialize() {
    super.initialize();
    InputStream in = ResourceLoader.load(
        SoundControlsExample.class,
        "./res/assets/sound/ELECTRONIC_computer_beep_09.wav",
        "asdf"
    );
    rawSound = readBytes( in );
    clip = new BlockingClip( rawSound );
    loopClip = new LoopEvent( clip );
    loopClip.initialize();
    dataLine = new BlockingDataLine( rawSound );
    loopStream = new LoopEvent( dataLine );
    loopStream.initialize();
}
private byte[] readBytes( InputStream in ) {
    try {
        BufferedInputStream buf = new BufferedInputStream( in );
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        int read;
        while( (read = buf.read()) != -1 ) {
            out.write( read );
        }
        in.close();
        return out.toByteArray();
    } catch( IOException ex ) {
        ex.printStackTrace();
        return null;
    }
}
```



```
        }
    }

private void shutDownClips() {
    if( loopClip != null ) loopClip.shutDown();
    if( loopStream != null ) loopStream.shutDown();
}

@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    if( keyboard.keyDownOnce( KeyEvent.VK_1 ) ) {
        loopClip.fire();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_2 ) ) {
        loopClip.done();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_3 ) ) {
        loopStream.fire();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_4 ) ) {
        loopStream.done();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_W ) ) {
        increaseGain( clip );
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_S ) ) {
        decreaseGain( clip );
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_A ) ) {
        panLeft( clip );
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_D ) ) {
        panRight( clip );
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_I ) ) {
        increaseGain( dataLine );
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_K ) ) {
        decreaseGain( dataLine );
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_J ) ) {
        panLeft( dataLine );
    }
}
```



```
if( keyboard.keyDownOnce( KeyEvent.VK_L ) ) {
    panRight( dataLine );
}
}

private void increaseGain( AudioStream audio ) {
    float current = audio.getGain();
    if( current < 10.0f ) {
        audio.setGain( current + 3.0f );
    }
}

private void decreaseGain( AudioStream audio ) {
    float current = audio.getGain();
    if( current > -20.0f ) {
        audio.setGain( current - 3.0f );
    }
}

private void panLeft( AudioStream audio ) {
    float current = audio.getPan();
    float precision = audio.getPrecision();
    audio.setPan( current - precision * 10.0f );
}

private void panRight( AudioStream audio ) {
    float current = audio.getPan();
    float precision = audio.getPrecision();
    audio.setPan( current + precision * 10.0f );
}

@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
}

@Override
protected void render( Graphics g ) {
    super.render( g );
    textPos = Utility.drawString( g, 20, textPos,
        "",
        "Clip Gain: " + clip.getGain(),
        "Clip Pan: " + clip.getPan(),
        "",
        "Stream Gain: " + dataLine.getGain(),
        "Stream Pan: " + dataLine.getPan(),
        "",
        "(1) Start Loop (clip)",
    );
}
```



```
        "(2) Stop Loop (clip)",
        "(3) Start Loop (stream)",
        "(4) Stop Loop (stream)",
        "",
        "(W) Raise Gain (clip)",
        "(S) Lower Gain (clip)",
        "(A) Pan Left (clip)",
        "(D) Pan Right (clip)",
        "",
        "(I) Raise Gain (stream)",
        "(K) Lower Gain (stream)",
        "(J) Pan Left (stream)",
        "(L) Pan Right (stream"
    );
}

@Override
protected void terminate() {
    super.terminate();
    shutDownClips();
}

public static void main( String[] args ) {
    launchApp( new SoundControlsExample() );
}
}
```



在处理声音时，Java 6.0 和 Java 7.0 之间有很多的不同。播放剪辑示例展示了 Java 6.0 和 Java 7.0 之间的不同行为。尽管较大的剪辑可以在 Java 7.0 中加载，并且播放较小的剪辑也会少些延迟，但有时候，Java 7.0 的小剪辑会停止工作。我没有找到这一问题的解决方案。作为一个折中方案，如果你使用 Java 6.0，那么所有较大的文件都需要是一个 `AudioDataLine` 类，并且较小的声音作为剪辑会工作得更好。如果你使用 Java 7.0，那么较小的剪辑和 `AudioDataLine` 对象一样好，并且较大的文件应该可以作为剪辑而加载。

遗憾的是，不使用第三方库的话，无法保证版本和操作系统之间的行为的一致性。如果需要声音可靠性，而当前声音库的行为又不够，请查看资源部分列出的带有 JOAL 库包装的 OpenAL。尽管 OpenAL 比较难以设置并开始工作，但它可以提供当前声音实现所没有的必需的功能。



## 13.12 资源和延伸阅读

“Sound,” 1995, <http://docs.oracle.com/javase/tutorial/sound/>.

Pfisterer, Matthias, Florian Bomers, “jsresources.org - Java Sound Resources,” 2005, <http://www.jsresources.org/>.

Audacity, which is a free, open source, cross-platform software for recording and editing sounds; see <http://audacity.sourceforge.net/>.

OpenAL, which is a cross-platform 3D audio API appropriate for use with gaming; see <http://connect.creativelabs.com/openal/default.aspx>.

JOAL, which is a reference implementation of the Java bindings for the OpenAL API; see <http://jogamp.org/joal/www/>.



# 第 14 章

## 用 ANT 进行开发

尽管本书的目标是提供工具以便从头开始开发游戏，而不使用任何第三方库，但使用命令行来构建和部署游戏实在是太困难了。能够从头做这些事情是很好，但我并不期望任何人这么做。

ANT 表示 Another Neat Tool，它用于提供独立于平台的构建脚本过程。构建脚本是用 XML 编写的，并且，尽管 ANT 易于扩展，但你所需要的大多数任务都已经提供了。

使用构建脚本很重要，因为它对于将一款游戏打包为一个易于运行的格式来说很节省时间。随着游戏的大小不断增加，要记住打包游戏所需的所有步骤变得很难。构建脚本不仅会记录打包软件所需的所有步骤，而且使得游戏能够在任何平台上构建。

### 14.1 安装 ANT 软件

不管是什操作系统，在安装 ANT 时，都需要记住几件事情。从 <http://ant.apache.org/> 下载所需的文件很容易，并且，下载文件不需要任何安装程序就可以运行。该文件可以放在任何位置。然而，需要设置如下的环境变量：

```
ANT_HOME=FOLDER_THAT_CONTAINS_BIN_AND_LIB  
JAVA_HOME=JDK_HOME_FOLDER  
Path=%Path%;ANT_HOME\bin
```

这些变量就是我在 Windows 7 机器上所使用的。如果你喜欢运行 Linux，如何设置这些变量，要取决于你的版本。在系统路径中添加 ANT\_HOME\bin 文件夹，将允许从命令行运行 ANT 工具。ANT 工具应该已经与最常用的 IDE 集成了，因此请放心使用那些 IDE 版本。

JAVA\_HOME 环境变量应该指向 JDK，以便 ANT 能够访问编译源代码所需的 Java 工



具。安装 ANT 的在线文档应该是最正确的，并且会及时更新，因此，建议你使用该文档作为安装指南。参见本章末尾的资源和延伸阅读以了解更多信息。

## 14.2 理解构建脚本的格式

构建脚本拥有几个标签。有一个 `project` 标签，它是任何构建脚本的根标签。该标签有一个名称和可选的 `basedir`，以及 `default` 目标属性。

```
<project name="name" basedir=". " default="default">  
</project>
```

还有 `target` 标签。一个构建脚本包含一个或多个 `target`，这是构建过程中的一个步骤。`target` 包含一个名称，一个说明，以及一个可选的 `depends` 属性，这个属性列出了这个 `target` 所依赖的所有 `target` 的列表，其中各个 `target` 用逗号隔开。在一个 `target` 开始运行之前，所有其依赖的 `target` 都要运行。依赖的 `target` 只运行一次，不管引用它们多少次。

```
<target name="target-name" description="target description"  
       depends="list, separated, with, commas">  
</target>
```

ANT 脚本还包括属性。属性有一个 `name`，以及一个 `Value` 或一个 `location`。如果使用了 `location` 属性，位置会转换为一个有效的文件路径。一旦定义了属性，就不能再覆盖它。不管一个属性定义了多少次，它都只拥有其第一次定义时设置的值。这也是在导入多个构建脚本时会有很多问题的原因。

```
<property name="name1" value="value1"/>  
<property name="name2" location="location/one"/>
```

属性很有用，因为可以用美元符号和花括号将其括起来，以引用它们。 `${property.name}`  在运行时展开，成为名称为 `property.name` 的属性的值。这种展开甚至在 `xml` 属性内部也会发生。ANT 使用相同的美元符号，提供了对所有 Java 系统属性的访问。 `${os.name}`  扩展后与 `System.getProperty("os.name")` 的值相同。

此外，ANT 还有一些内建的属性。

- `basedir`——构建脚本的绝对路径。`basedir` 属性在 `<project>` 标签中设置。
- `ant.file`——构建文件的绝对路径。



- `ant.project.name`——`project` 标签的 `name` 属性。

参见 ANT 站点的文档，以了解有关内建属性的更多信息。

下面是一个简单的 ANT 脚本，它只有一个 `target`，该 `target` 使用 `echo` 标签打印出 Hello World!：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="hello-world" default="default">
    <description>
        This is a simple Hello World script
    </description>
    <target name="default" description="target description">
        <echo>Hello World!</echo>
    </target>
</project>
```

要运行这个脚本，我们有两种选择。如果使用名称 `build.xml` 保存该脚本，那么，直接将命令提示符指向相同的目录并输入 `ant`，这将导致 ANT 扫描当前目录，加载名为 `build.xml` 的任何文件，并且执行默认的 `target`。

```
C:\Ant Tests>ant
Buildfile: C:\Ant Tests\build.xml
default:
    [echo] Hello World!
BUILD SUCCESSFUL
Total time: 0 seconds
```

如果该脚本不是名为 `build.xml`，则使用 `-f` 命令和 `-line` 标志，则可以通过传递文件名来运行任何脚本：

```
C:\Ant Tests>ant -f hello-world.xml
Buildfile: C:\Ant Tests\hello-world.xml
default:
    [echo] Hello World!
BUILD SUCCESSFUL
Total time: 0 seconds
```

`-version` 标志是确保 ANT 的当前版本在系统路径中的一种好方法：

```
C:\Ant Tests>ant -version
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
C:\Ant Tests>
```

`-help` 标志用来列出所有可用的命令。



```
C:\Ant Tests>ant -help
ant [options] [target [target2 [target3] ...]]
Options:
-help, -h          print this message
-projecthelp, -p   print project help information
-version           print the version information and exit
-diagnostics       print information that might be helpful to
                   diagnose or report problems.
-quiet, -q         be extra quiet
-verbose, -v        be extra verbose
... lots more flags ...
```

-p 标志列出项目帮助信息，它显示与构建脚本相关的所有信息。HelloWorld.xml 脚本只有一个 target，但是，项目的 help 标志的输出仍然有用，它显示所有的标记文档以及默认的 target。

```
C:\Ant Tests>ant -p
Buildfile: C:\Ant Tests\build.xml
    This is a simple Hello World script
Main targets:
default target description
Default target: default
```

最后，要运行特定的 target，可以在该命令的末尾传入 target 名称。或者用空格隔开的多个 target，则只运行那些非默认 target 的 target。这是只运行特定构建步骤的一种简易方式。

```
C:\Ant Tests>ant -f hello-world.xml default
Buildfile: C:\Ant Tests\hello-world.xml
default:
[echo] Hello World!
BUILD SUCCESSFUL
Total time: 0 seconds
```

## 14.3 学习常见 ANT 任务

ANT 有很多内建标记，可以用来执行构建和打包一个应用程序所需的大多数任务。  
<echo>标记把信息打印到控制台。



进行开发



```
<echo>This prints out to the console</echo>
```

要创建目录，使用`<mkdir>`标签。`<delete>`标签将删除目录。注意在属性值中如何使用 `${basedir}`语法。

```
<delete dir="${basedir}/lib"/>
<mkdir dir="${basedir}/lib"/>
<delete dir="${basedir}/bin"/>
<mkdir dir="${basedir}/bin"/>
```

可以使用`<copy>`标签来复制和重命名文件，并复制整个目录。

像下面这样复制和重命名单个文件：

```
<copy file=". /res/myfile.txt" tofile=". /lib/mycopy.txt"/>
```

如下标签将一个文件复制到一个不同的目录中：

```
<copy file="res/myfile.txt" todir="lib/some/other/dir"/>
```

要复制整个目录，使用如下标签：

```
<copy todir="lib/new/dir">
  <fileset dir="res"/>
</copy>
```

要避免复制某些文件，使用如下形式：

```
<copy todir="lib/new/dir">
  <fileset dir="res">
    <exclude name="**/*.java"/>
  </fileset>
</copy>
```

注意`**/*.java`语法。两个`**`告诉 ANT 搜索所有的目录和子目录。

可以使用如下的内容，允许 ANT 从一个构建脚本来编译 Java 代码：

```
<path id="common.classpath">
  <pathelement location="${basedir}/bin"/>
</path>
<javac
  srcdir="${basedir}/src"
  destdir="${basedir}/bin"
  classpathref="common.classpath"
  debug="on"
/>
```



使用 `classpath` 引用，以便可以在其他地方定义众多不同的库和代码文件。

如下的标记用来创建一个 JAR 文件：

```
<fileset id="jar.files" dir="bin">
    <exclude name="**/*Tests.class"/>
</fileset>

<jar destfile="lib/${ant.project.name}.jar">
    <fileset refid="jar.files"/>
    <manifest>
        <attribute name="Main-Class" value="javagames.ant.HelloWorld"/>
        <attribute name="Class-Path"
            value="jar1-name jar2-name directory-name/jar3-name"/>
    </manifest>
</jar>
```

注意，可以包含一个`<manifest>`标签，以指定 `main` 类和独立的库。`<fileset>`标签指定了在 JAR 文件中将放置哪个文件。`destfile` 属性是`<jar>`标签所创建的 JAR 文件的名称和位置。下面的内容可以用来标记一个 JAR 文件：

```
<delete file="signjar.keystore"/>
<genkey alias="${ant.project.name}"
    validity="999999"
    storepass="storepass"
    keystore="signjar.keystore">
    <dname>
        <param name="CN" value="Tim Wright"/>
    <param name="OU" value="Groovy Inc."/>
    <param name="O" value="Planet Earth"/>
    <param name="C" value="Milkyway"/>
    </dname>
</genkey>
<signjar jar="lib/${ant.project.name}.jar"
    keystore="signjar.keystore"
    alias="${ant.project.name}"
    storepass="storepass"/>
<delete file="signjar.keystore"/>
```

在任务的开头和结尾都删除了 `signjar.keystore`，以确保将其删除，即便在运行单个 JAR 任务的时候可能会有错误。`<genkey>` 标签创建了 `keystore`，并且 `<signjar>` 使用该 `keystore` 来签名 JAR 文件。这些标签运行 JDK 所带的命令行工具。

最后，如下的标签可以用来将文件、文件夹以及其他 ZIP 文件，如其他的 Java 库，



进行开发

添加到一个已经创建的 JAR 文件中。

```
<jar update="true" destfile="lib/${ant.project.name}.jar">
    <fileset dir="src"/>
    <fileset dir="res"/>
    <zipfileset src="res/gamelib.jar"/>
</jar>
```

当创建默认的脚本然后将其扩展，以向 JAR 文件添加额外的资源的时候，这个标签很方便。

## 14.4 构建一个可扩展的构建脚本

创建一个可扩展的脚本，它可以用做一个较长过程的起点，但是，这么做是值得的。

<import>标签用来在构建脚本中包含其他脚本。

```
<import file="toInclude.xml"/>
```

在所包含的脚本的顶层定义的任何属性，都不能被覆盖，因此，确保在包含任何脚本之前，先设置属性。

在通用的构建脚本中，将第三方库定义为属性，这可能是有帮助的。

```
<property name="junit.lib"
  location="${common.home}/../3rd/JUnit/junit-4.5.jar"/>
```

通用脚本中的每一个默认 target 都划分为两个 target。例如，clean target 定义如下：

```
<target name="clean" depends="project-clean">
    <echo>Deleting project directory structure...</echo>
    <delete dir="${common.bin}"/>
    <delete dir="${common.lib}"/>
    <delete dir="${common.docs}"/>
    <delete dir="${common.junit}"/>
</target>
<target name="project-clean"/>
```

注意，clean target 取决于空的 project-clean target。这个结构考虑到了默认的 target 将被替代，或者在一个定制的 target 之前或之后执行。

和属性不同，target 是可以覆盖的。具有相同名称的最后一个 target 将会采用，而属性



则使用给定的第一个值。如果 3 个不同的文件都定义了相同的 target，那么使用最后定义的 target。要调用一个之前定义的 target，可以使用项目名称来引用被覆盖的 target。例如，如果名为 common 的项目定义了一个名为 clean 的 target，那么总是可以使用名称 common.clean 来访问该 target。

要覆盖默认的 clean target，使用相同的名称定义一个新的 target。

```
<project name="build">
  <import file="common.xml"/>
  <target name="clean">
  </target>
</project>
```

前面的例子覆盖了 clean target，完全替代了它。如果你需要定义一个定制的 tag，它在默认的 clean target 之前运行但并不替代 clean target，那么要覆盖默认的 clean 标签所依赖的<project-clean>。

```
<project name="custom" default="clean">
  <include file="common.xml"/>
  <target name="project-clean">
    <echo>Runs before default</echo>
  </target>
</project>
```

前面的例子覆盖了 project-clean 标签，它将会在通用的 clean tag 之前执行。

前面所介绍的项目名称语法，可以用来引用默认的 clean 标签。这个功能可以用来将默认标签定义为一个覆盖标签的依赖标签，以便定制的标签在通用的标签之后运行。

```
<project name="custom">
  <import file="common.xml"/>
  <target name="clean" depends="common.clean">
    <echo>Runs after default</echo>
  </target>
</project>
```

这个例子覆盖了 clean target，但由于它依赖于 common.clean target，因此定制的 clean 将在默认的标签之后运行。

有 3 个选项，使得我们可以以多种方法来扩展一个脚本。

- 替代一个 target——用相同的名称创建一个新的 target。
- 在默认之后定制——使用 depends="projectname.target"语法。



进行开发

### ■ 在默认之前定制——覆盖 project-target 标签。

使用 ANT 覆盖属性是不可能的，因此，一种简单的技巧是在 target 中定义默认属性，而不是在构建脚本的开头定义。只要在调用 target 之前设置了属性，就可以覆盖默认的属性。

```
<target name="compile" depends="project-compile">
    <property name="compile.srmdir" value="${basedir}/src;${basedir}/test"/>
    <property name="compile.destdir" value="${basedir}/bin"/>
    <property name="compile.debug" value="on"/>
    <property name="compile.classpath" value="${common.classpath}"/>

    <echo>Compiling source code...</echo>
    <javac
        srmdir="${compile.srmdir}"
        destdir="${compile.destdir}"
        classpathref="${compile.classpath}"
        debug="${compile.debug}"
    />
</target>
```

target 中使用的所有属性，compile.srmdir、compile.destdir、compile.debug 和 compile.classpath，都不能等到 target 运行时才设置。如果构建脚本包含了通用脚本，该脚本要在调用任务之前定义这些属性中的一个，当设置 target 属性时，这些属性应该已经存在了。

如下是可扩展的构建脚本的一个示例，可以将其用做构建将来的游戏项目的起点。

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    This is the base script used for creating
    custom build scripts.
-->
<project name="common" basedir=".">
    <dirname property="common.home" file="${ant.file.common}"/>
    <property name="junit.lib"
        location="${common.home}/../3rd/JUnit/junit-4.5.jar"/>
    <property name="common.bin" value="${basedir}/bin"/>
    <property name="common.lib" value="${basedir}/lib"/>
    <property name="common.docs" value="${basedir}/docs"/>
    <property name="common.test" value="${basedir}/test"/>
    <property name="common.junit" value="${basedir}/junit"/>
    <property name="common.classpath" value="common.classpath"/>
```



```
<path id="common.classpath">
    <pathelement location="${common.bin}"/>
</path>

<!-- =====
     BUILD
===== -->
<target name="build" description="--> Builds the project"
        depends="project-build, clean, init, compile, jar"/>
<target name="project-build"/>
<!-- =====

INIT
===== -->
<target name="init" depends="project-init">
    <echo>Creating project directory structure...</echo>
    <mkdir dir="${common.bin}"/>
    <mkdir dir="${common.lib}"/>
    <mkdir dir="${common.docs}"/>
    <mkdir dir="${common.test}"/>
    <mkdir dir="${common.junit}"/>
</target>
<target name="project-init"/>
<!-- =====

CLEAN
===== -->
<target name="clean" depends="project-clean">
    <echo>Deleting project directory structure...</echo>
    <delete dir="${common.bin}"/>
    <delete dir="${common.lib}"/>
    <delete dir="${common.docs}"/>
    <delete dir="${common.junit}"/>
</target>
<target name="project-clean"/>

<!-- =====

COMPILE
===== -->
<target name="compile" depends="project-compile">
    <property name="compile.srmdir" value="${basedir}/src;${basedir}/test"/>
    <property name="compile.destdir" value="${basedir}/bin"/>
    <property name="compile.debug" value="on"/>
    <property name="compile.classpath" value="${common.classpath}"/>
```



```
<echo>Compiling source code...</echo>
<javac
    srcdir="${compile.srcdir}"
    destdir="${compile.destdir}"
    classpathref="${compile.classpath}"
    debug="${compile.debug}"
    />
</target>
<target name="project-compile"/>

<!-- =====
      JAR
===== -->
<target name="jar" depends="project-jar">
    <property name="jar.name" value="${ant.project.name}" />
    <property name="jar.mainclass" value="" />
    <property name="jar.classpath" value="" />
    <property name="jar.filesel" value="jar.filesel" />
    <property name="jar.destfile" value="${common.lib}/${jar.name}.jar" />
    <filesel id="jar.filesel" dir="${common.bin}">
        <exclude name="**/Test*.class" />
        <exclude name="**/*Tests.class" />
        <exclude name="**/*TestCase.class" />
    </filesel>
    <echo>Creating ${common.lib}${file.separator}${jar.name}.jar...</echo>
    <jar destfile="${jar.destfile}">
        <filesel refid="${jar.filesel}" />
    <manifest>
        <attribute name="Main-Class" value="${jar.mainclass}" />
        <attribute name="Class-Path" value="${jar.classpath}" />
    </manifest>
    </jar>
</target>
<target name="project-jar"/>

<!-- =====
      SIGNJAR
===== -->
<target name="signjar" depends="project-signjar">
    <property name="signjar.keystore"
        value="${basedir}/${ant.project.name}.keystore" />
    <property name="signjar.alias" value="${ant.project.name}" />
```



```

<property name="signjar.storepass" value="storepass"/>
<property name="signjar.validity" value="999999"/>
<property name="signjar.jar" value="${common.lib}/${jar.name}.jar"/>
<echo>Signing Jar File</echo>
<delete file="${signjar.keystore}"/>
<genkey alias="${signjar.alias}">
    validity="${signjar.validity}"
    storepass="${signjar.storepass}"
    keystore="${signjar.keystore}>
<dname>
    <param name="CN" value="Tim Wright"/>
    <param name="OU" value="Groovy Inc."/>
    <param name="O" value="Rio Rancho"/>
    <param name="C" value="US"/>
</dname>
</genkey>
<signjar jar="${signjar.jar}">
    keystore="${signjar.keystore}"
    alias="${signjar.alias}"
    storepass="${signjar.storepass}"/>
    <delete file="${signjar.keystore}"/>
</target>
<target name="project-signjar"/>
</project>

```

使用 common.xml 构建脚本很容易。CustomBuild.xml 文件的默认 target 设置为定制的默认 target。默认 target 依赖于 build 和 signjar 标签，它们在通用构建脚本中定义。注意，在导入通用构建脚本之前，定义了 common.classpath 属性。由于 common.classpath 在通用构建脚本的顶部定义，因此在包含通用脚本之前，必须先设置它。

即便在导入之前设置了 common.classpath 属性，还是可以在导入之后定义类路径，只要在使用它之前定义就可以了。定制的构建示例脚本是一个很好的起点。在本书最后的游戏示例中，我们将使用它来构建、打包游戏，并且在其最终配置中部署游戏。

```

<?xml version="1.0" encoding="UTF-8"?>
<project name="custom-build" default="default">
    <description>
        Extending the common.xml
    </description>
    <property name="common.classpath" value="custom.classpath"/>
    <import file="common.xml"/>
    <path id="custom.classpath">
        <pathelement location="${basedir}/bin"/>

```



```
<path element location=" ${junit.lib} "/>
</path>
<target name="default" depends="build, signjar"
       description="--> This builds my first real game"/>
</project>
```



## 14.5 资源和延伸阅读

“Apache ANT Manual,” 2013, <http://ant.apache.org/manual/index.html>.