```
            }
        };
        // 为 commentItem 菜单项添加事件监听器
        commentItem.addActionListener(menuListener);
        exitItem.addActionListener(menuListener);
        // 为 file 菜单添加菜单项
        file.add(newItem);
        file.add(saveItem);
        file.add(exitItem);
        // 为 edit 菜单添加菜单项
        edit.add(autoWrap);
        // 使用 addSeparator 方法来添加菜单分隔线
        edit.addSeparator();
        edit.add(copyItem);
        edit.add(pasteItem);
        // 为 format 菜单添加菜单项
        format.add(commentItem);
        format.add(cancelItem);
        // 使用添加 new MenuItem("-") 的方式添加菜单分隔线
        edit.add(new MenuItem("-"));
        // 将 format 菜单组合到 edit 菜单中，从而形成二级菜单
        edit.add(format);
        // 将 file、edit 菜单添加到 mb 菜单条中
        mb.add(file);
        mb.add(edit);
        // 为 f 窗口设置菜单条
        f.setMenuBar(mb);
        // 以匿名内部类的形式来创建事件监听器对象
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        f.add(ta);
        f.pack();
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new SimpleMenu().init();
    }
}
```

上面程序中的菜单既有复选框菜单项和菜单分隔符，也有二级菜单，并为两个菜单项添加了快捷键，为 commentItem、exitItem 两个菜单项添加了事件监听器。运行该程序，并按“Ctrl+Shift+/”快捷键，将看到如图 11.25 所示的窗口。

提示：

AWT 的菜单组件不能创建图标菜单，如果希望创建带图标的菜单，则应该使用 Swing 的菜单组件：JMenuBar、JMenu、JMenuItem 和 JPopupMenu 组件。Swing 的菜单组件和 AWT 的菜单组件的用法基本相似，读者可参考本程序学习使用 Swing 的菜单组件。

11.6.2 右键菜单

右键菜单使用 PopupMenu 对象表示，创建右键菜单的步骤如下。

① 创建 PopupMenu 的实例。

② 创建多个 MenuItem 的多个实例，依次将这些实例加入 PopupMenu 中。

③ 将 PopupMenu 加入到目标组件中。

④ 为需要出现上下文菜单的组件编写鼠标监听器，当用户释放鼠标

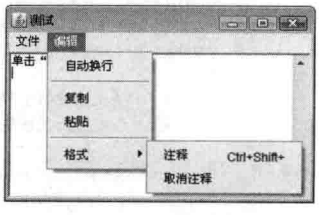


图 11.25　AWT 菜单示例

右键时弹出右键菜单。

下面程序创建了一个右键菜单，该右键菜单就是“借用”前面 SimpleMenu 中 edit 菜单下的所有菜单项。

程序清单：codes\11\11.6\PopupMenuTest.java

```
public class PopupMenuTest
{
    private TextArea ta = new TextArea(4 , 30);
    private Frame f = new Frame("测试");
    PopupMenu pop = new PopupMenu();
    CheckboxMenuItem autoWrap =
        new CheckboxMenuItem("自动换行");
    MenuItem copyItem = new MenuItem("复制");
    MenuItem pasteItem = new MenuItem("粘贴");
    Menu format = new Menu("格式");
    // 创建 commentItem 菜单项，指定使用“Ctrl+Shift+/”快捷键
    MenuItem commentItem = new MenuItem("注释" ,
        new MenuShortcut(KeyEvent.VK_SLASH , true));
    MenuItem cancelItem = new MenuItem("取消注释");
    public void init()
    {
        // 以 Lambda 表达式创建菜单事件监听器
        ActionListener menuListener = e ->
        {
            String cmd = e.getActionCommand();
            ta.append("单击“" + cmd + "”菜单" + "\n");
            if (cmd.equals("退出"))
            {
                System.exit(0);
            }
        };
        // 为 commentItem 菜单项添加事件监听
        commentItem.addActionListener(menuListener);
        // 为 pop 菜单添加菜单项
        pop.add(autoWrap);
        // 使用 addSeparator 方法来添加菜单分隔线
        pop.addSeparator();
        pop.add(copyItem);
        pop.add(pasteItem);
        // 为 format 菜单添加菜单项
        format.add(commentItem);
        format.add(cancelItem);
        // 使用添加 new MenuItem("-")的方式添加菜单分隔线
        pop.add(new MenuItem("-"));
        // 将 format 菜单组合到 pop 菜单中，从而形成二级菜单
        pop.add(format);
        final Panel p = new Panel();
        p.setPreferredSize(new Dimension(300, 160));
        // 向 p 窗口中添加 PopupMenu 对象
        p.add(pop);
        // 添加鼠标事件监听器
        p.addMouseListener(new MouseAdapter()
        {
            public void mouseReleased(MouseEvent e)
            {
                // 如果释放的是鼠标右键
                if (e.isPopupTrigger())
                {
                    pop.show(p , e.getX() , e.getY());
                }
            }
        });
        f.add(p);
        f.add(ta , BorderLayout.NORTH);
        // 以匿名内部类的形式来创建事件监听器对象
        f.addWindowListener(new WindowAdapter()
        {
```

```
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        f.pack();
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new PopupMenuTest().init();
    }
}
```

运行上面程序，会看到如图 11.26 所示的窗口。

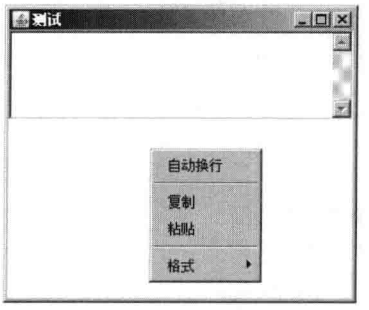
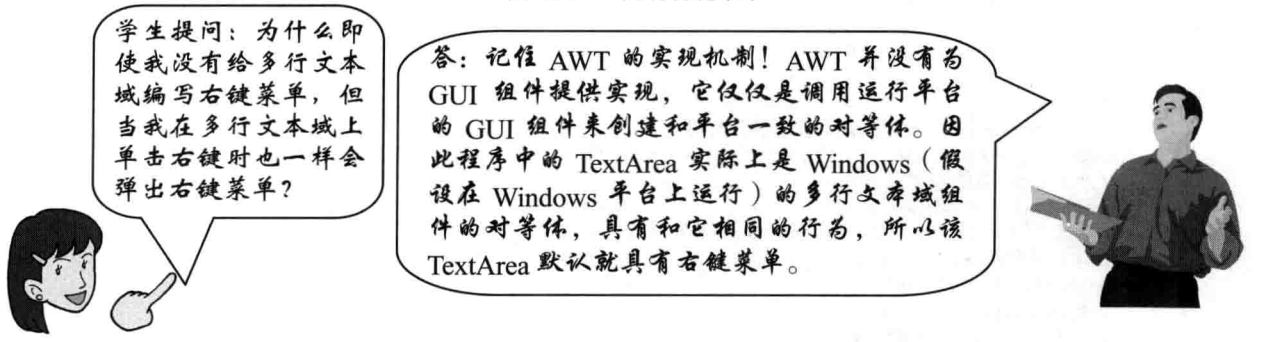


图 11.26　实现右键菜单



11.7　在 AWT 中绘图

很多程序如各种小游戏都需要在窗口中绘制各种图形，除此之外，即使在开发 Java EE 项目时，有时候也必须“动态”地向客户端生成各种图形、图表，比如图形验证码、统计图等，这都需要利用 AWT 的绘图功能。

11.7.1　画图的实现原理

在 Component 类里提供了和绘图有关的三个方法。

- paint(Graphics g)：绘制组件的外观。
- update(Graphics g)：调用 paint()方法，刷新组件外观。
- repaint()：调用 update()方法，刷新组件外观。

上面三个方法的调用关系为：repaint()方法调用 update()方法；update()方法调用 paint()方法。

Container 类中的 update()方法先以组件的背景色填充整个组件区域，然后调用 paint()方法重画组件。Container 类的 update()方法代码如下：

```
public void update(Graphics g) {
    if (isShowing()) {
        //以组件的背景色填充整个组件区域
        if (! (peer instanceof LightweightPeer)) {
```

```
            g.clearRect(0, 0, width, height);
        }
        paint(g);
    }
}
```

普通组件的 update()方法则直接调用 paint()方法。

```
public void update(Graphics g) {
    paint(g);
}
```

图 11.27 显示了 paint()、repaint()和 update()三个方法之间的调用关系。

从图 11.27 中可以看出，程序不应该主动调用组件的 paint()和 update()方法，这两个方法都由 AWT 系统负责调用。如果程序希望 AWT 系统重新绘制该组件，则调用该组件的 repaint()方法即可。而 paint()和 update()方法通常被重写。在通常情况下，程序通过重写 paint()方法实现在 AWT 组件上绘图。

重写 update()或 paint()方法时，该方法里包含了一个 Graphics 类型的参数，通过该 Graphics 参数就可以实现绘图功能。

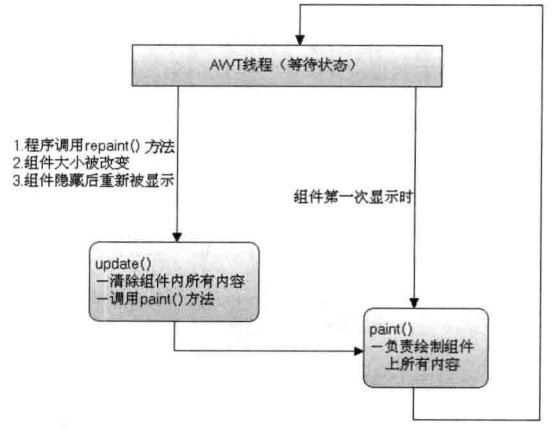


图 11.27 paint()、update()和 repaint()三个方法的调用关系

11.7.2 使用 Graphics 类

Graphics 是一个抽象的画笔对象，Graphics 可以在组件上绘制丰富多彩的几何图形和位图。Graphics 类提供了如下几个方法用于绘制几何图形和位图。

- drawLine()：绘制直线。
- drawString()：绘制字符串。
- drawRect()：绘制矩形。
- drawRoundRect()：绘制圆角矩形。
- drawOval()：绘制椭圆形状。
- drawPolygon()：绘制多边形边框。
- drawArc()：绘制一段圆弧（可能是椭圆的圆弧）。
- drawPolyline()：绘制折线。
- fillRect()：填充一个矩形区域。
- fillRoundRect()：填充一个圆角矩形区域。
- fillOval()：填充椭圆区域。
- fillPolygon()：填充一个多边形区域。
- fillArc()：填充圆弧和圆弧两个端点到中心连线所包围的区域。
- drawImage()：绘制位图。

除此之外，Graphics 还提供了 setColor()和 setFont()两个方法用于设置画笔的颜色和字体（仅当绘制字符串时有效），其中 setColor()方法需要传入一个 Color 参数，它可以使用 RGB、CMYK 等方式设置一个颜色；而 setFont()方法需要传入一个 Font 参数，Font 参数需要指定字体名、字体样式、字体大小三个属性。

提示：

实际上，不仅 Graphics 对象可以使用 setColor()和 setFont()方法来设置画笔的颜色和字体，AWT 普通组件也可以通过 Color()和 Font()方法来改变它的前景色和字体。除此之外，所有组件都有一个 setBackground()方法用于设置组件的背景色。

AWT专门提供一个Canvas类作为绘图的画布，程序可以通过创建Canvas的子类，并重写它的paint()方法来实现绘图。下面程序示范了一个简单的绘图程序。

程序清单：codes\11\11.7\SimpleDraw.java

```
public class SimpleDraw
{
    private final String RECT_SHAPE = "rect";
    private final String OVAL_SHAPE = "oval";
    private Frame f = new Frame("简单绘图");
    private Button rect = new Button("绘制矩形");
    private Button oval = new Button("绘制圆形");
    private MyCanvas drawArea = new MyCanvas();
    // 用于保存需要绘制什么图形的变量
    private String shape = "";
    public void init()
    {
        Panel p = new Panel();
        rect.addActionListener(e ->
        {
            // 设置 shape 变量为 RECT_SHAPE
            shape = RECT_SHAPE;
            // 重画 MyCanvas 对象，即调用它的 repait()方法
            drawArea.repaint();
        });
        oval.addActionListener(e ->
        {
            // 设置 shape 变量为 OVAL_SHAPE
            shape = OVAL_SHAPE;
            // 重画 MyCanvas 对象，即调用它的 repait()方法
            drawArea.repaint();
        });
        p.add(rect);
        p.add(oval);
        drawArea.setPreferredSize(new Dimension(250 , 180));
        f.add(drawArea);
        f.add(p , BorderLayout.SOUTH);
        f.pack();
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new SimpleDraw().init();
    }
    class MyCanvas extends Canvas
    {
        // 重写 Canvas 的 paint()方法，实现绘画
        public void paint(Graphics g)
        {
            Random rand = new Random();
            if (shape.equals(RECT_SHAPE))
            {
                // 设置画笔颜色
                g.setColor(new Color(220, 100, 80));
                // 随机地绘制一个矩形框
                g.drawRect( rand.nextInt(200)
                    , rand.nextInt(120) , 40 , 60);
            }
            if (shape.equals(OVAL_SHAPE))
            {
                // 设置画笔颜色
                g.setColor(new Color(80, 100, 200));
                // 随机地填充一个实心圆形
                g.fillOval( rand.nextInt(200)
                    , rand.nextInt(120) , 50 , 40);
            }
        }
    }
}
```

上面程序定义了一个 MyCanvas 类，它继承了 Canvas 类，重写了 Canvas 类的 paint()方法（上面程序中粗体字代码部分），该方法根据 shape 变量值随机地绘制矩形或填充椭圆区域。窗口中还定义了两个按钮，当用户单击任意一个按钮时，程序调用了 drawArea 对象的 repaint()方法，该方法导致画布重绘（即调用 drawArea 对象的 update()方法，该方法再调用 paint()方法）。

运行上面程序，单击“绘制圆形”按钮，将看到如图 11.28 所示的窗口。

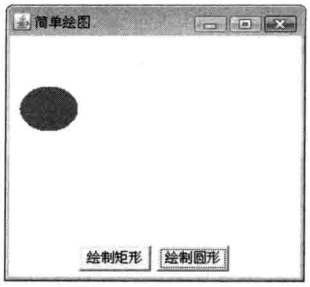


图 11.28 简单绘图

注意：

运行上面程序时，如果改变窗口大小，或者让该窗口隐藏后重新显示都会导致 drawArea 重新绘制形状——这是因为这些动作都会触发组件的 update()方法。

Java 也可用于开发一些动画。所谓动画，就是间隔一定的时间（通常小于 0.1 秒）重新绘制新的图像，两次绘制的图像之间差异较小，肉眼看起来就成了所谓的动画。为了实现间隔一定的时间就重新调用组件的 repaint()方法，可以借助于 Swing 提供的 Timer 类，Timer 类是一个定时器，它有如下一个构造器。

- Timer(int delay, ActionListener listener)：每间隔 delay 毫秒，系统自动触发 ActionListener 监听器里的事件处理器（actionPerformed()方法）。

下面程序示范了一个简单的弹球游戏，其中小球和球拍分别以圆形区域和矩形区域代替，小球开始以随机速度向下运动，遇到边框或球拍时小球反弹；球拍则由用户控制，当用户按下向左、向右键时，球拍将会向左、向右移动。

程序清单：codes\11\11.7\PinBall.java

```
public class PinBall
{
    // 桌面的宽度
    private final int TABLE_WIDTH = 300;
    // 桌面的高度
    private final int TABLE_HEIGHT = 400;
    // 球拍的垂直位置
    private final int RACKET_Y = 340;
    // 下面定义球拍的高度和宽度
    private final int RACKET_HEIGHT = 20;
    private final int RACKET_WIDTH = 60;
    // 小球的大小
    private final int BALL_SIZE = 16;
    private Frame f = new Frame("弹球游戏");
    Random rand = new Random();
    // 小球纵向的运行速度
    private int ySpeed = 10;
    // 返回一个-0.5~0.5 的比率，用于控制小球的运行方向
    private double xyRate = rand.nextDouble() - 0.5;
    // 小球横向的运行速度
    private int xSpeed = (int)(ySpeed * xyRate * 2);
    // ballX 和 ballY 代表小球的坐标
    private int ballX = rand.nextInt(200) + 20;
    private int ballY = rand.nextInt(10) + 20;
    // racketX 代表球拍的水平位置
    private int racketX = rand.nextInt(200);
    private MyCanvas tableArea = new MyCanvas();
    Timer timer;
    // 游戏是否结束的旗标
    private boolean isLose = false;
    public void init()
    {
        // 设置桌面区域的最佳大小
        tableArea.setPreferredSize(
```

```
            new Dimension(TABLE_WIDTH , TABLE_HEIGHT));
        f.add(tableArea);
        // 定义键盘监听器
        KeyAdapter keyProcessor = new KeyAdapter()
        {
            public void keyPressed(KeyEvent ke)
            {
                // 按下向左、向右键时，球拍水平坐标分别减少、增加
                if (ke.getKeyCode() == KeyEvent.VK_LEFT)
                {
                    if (racketX > 0)
                    racketX -= 10;
                }
                if (ke.getKeyCode() == KeyEvent.VK_RIGHT)
                {
                    if (racketX < TABLE_WIDTH - RACKET_WIDTH)
                    racketX += 10;
                }
            }
        };
        // 为窗口和 tableArea 对象分别添加键盘监听器
        f.addKeyListener(keyProcessor);
        tableArea.addKeyListener(keyProcessor);
        // 定义每 0.1 秒执行一次的事件监听器
        ActionListener taskPerformer = evt ->
        {
            // 如果小球碰到左边边框
            if (ballX  <= 0 || ballX >= TABLE_WIDTH - BALL_SIZE)
            {
                xSpeed = -xSpeed;
            }
            // 如果小球高度超出了球拍位置，且横向不在球拍范围之内，游戏结束
            if (ballY >= RACKET_Y - BALL_SIZE &&
                (ballX < racketX || ballX > racketX + RACKET_WIDTH))
            {
                timer.stop();
                // 设置游戏是否结束的旗标为 true
                isLose = true;
                tableArea.repaint();
            }
            // 如果小球位于球拍之内，且到达球拍位置，小球反弹
            else if (ballY  <= 0 ||
                (ballY >= RACKET_Y - BALL_SIZE
                    && ballX > racketX && ballX <= racketX + RACKET_WIDTH))
            {
                ySpeed = -ySpeed;
            }
            // 小球坐标增加
            ballY += ySpeed;
            ballX += xSpeed;
            tableArea.repaint();
        };      timer = new Timer(100, taskPerformer);
        timer.start();
        f.pack();
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new PinBall().init();
    }
    class MyCanvas extends Canvas
    {
        // 重写 Canvas 的 paint()方法，实现绘画
        public void paint(Graphics g)
        {
            // 如果游戏已经结束
            if (isLose)
            {
                g.setColor(new Color(255, 0, 0));
                g.setFont(new Font("Times" , Font.BOLD, 30));
```

```
                g.drawString("游戏已结束！" , 50 ,200);
            }
            // 如果游戏还未结束
            else
            {
                // 设置颜色，并绘制小球
                g.setColor(new Color(240, 240, 80));
                g.fillOval(ballX , ballY , BALL_SIZE, BALL_SIZE);
                // 设置颜色，并绘制球拍
                g.setColor(new Color(80, 80, 200));
                g.fillRect(racketX , RACKET_Y
                    , RACKET_WIDTH , RACKET_HEIGHT);
            }
        }
    }
}
```

运行上面程序，将看到一个简单的弹球游戏，运行效果如图 11.29 所示。

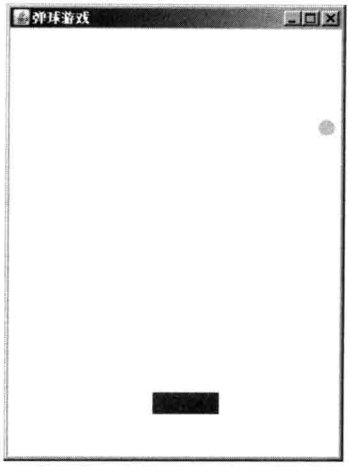


图 11.29 简单的弹球游戏

提示： 上面的弹球游戏还比较简陋，如果为该游戏增加位图背景，使用更逼真的小球位图代替小球，更逼真的球拍位图代替球拍，并在弹球桌面增加一些障碍物，整个弹球游戏将会更有趣味性。细心的读者可能会发现上面的游戏有轻微的闪烁，这是由于 AWT 组件的绘图没有采用双缓冲技术，当重写 paint()方法来绘制图形时，所有的图形都是直接绘制到 GUI 组件上的，所以多次重新调用 paint()方法进行绘制会发生闪烁现象。使用 Swing 组件就可避免这种闪烁，Swing 组件没有提供 Canvas 对应的组件，使用 Swing 的 Panel 组件作为画布即可。

11.8 处理位图

如果仅仅绘制一些简单的几何图形，程序的图形效果依然比较单调。AWT 也允许在组件上绘制位图，Graphics 提供了 drawImage 方法用于绘制位图，该方法需要一个 Image 参数——代表位图，通过该方法就可以绘制出指定的位图。

11.8.1 Image 抽象类和 BufferedImage 实现类

Image 类代表位图，但它是一个抽象类，无法直接创建 Image 对象，为此 Java 为它提供了一个 BufferedImage 子类，这个子类是一个可访问图像数据缓冲区的 Image 实现类。该类提供了一个简单的构造器，用于创建一个 BufferedImage 对象。

- BufferedImage(int width, int height, int imageType)：创建指定大小、指定图像类型的 BufferedImage 对象，其中 imageType 可以是 BufferedImage.TYPE_INT_RGB、BufferedImage. TYPE_BYTE_GRAY 等值。

除此之外，BufferedImage 还提供了一个 getGraphics()方法返回该对象的 Graphics 对象，从而允许通过该 Graphics 对象向 Image 中添加图形。

借助 BufferedImage 可以在 AWT 中实现缓冲技术——当需要向 GUI 组件上绘制图形时，不要直接绘制到该 GUI 组件上，而是先将图形绘制到 BufferedImage 对象中，然后再调用组件的 drawImage 方法一次性地将 BufferedImage 对象绘制到特定组件上。

下面程序通过 BufferedImage 类实现了图形缓冲，并实现了一个简单的手绘程序。

程序清单：codes\11\11.8\HandDraw.java

```
public class HandDraw
{
```

```
// 画图区的宽度
private final int AREA_WIDTH = 500;
// 画图区的高度
private final int AREA_HEIGHT = 400;
// 下面的 preX、preY 保存了上一次鼠标拖动事件的鼠标坐标
private int preX = -1;
private int preY = -1;
// 定义一个右键菜单用于设置画笔颜色
PopupMenu pop = new PopupMenu();
MenuItem redItem = new MenuItem("红色");
MenuItem greenItem = new MenuItem("绿色");
MenuItem blueItem = new MenuItem("蓝色");
// 定义一个 BufferedImage 对象
BufferedImage image = new BufferedImage(AREA_WIDTH
    , AREA_HEIGHT , BufferedImage.TYPE_INT_RGB);
// 获取 image 对象的 Graphics
Graphics g = image.getGraphics();
private Frame f = new Frame("简单手绘程序");
private DrawCanvas drawArea = new DrawCanvas();
// 用于保存画笔颜色
private Color foreColor = new Color(255, 0 ,0);
public void init()
{
    // 定义右键菜单的事件监听器
    ActionListener menuListener = e ->
    {
        if (e.getActionCommand().equals("绿色"))
        {
            foreColor = new Color(0 , 255 , 0);
        }
        if (e.getActionCommand().equals("红色"))
        {
            foreColor = new Color(255 , 0 , 0);
        }
        if (e.getActionCommand().equals("蓝色"))
        {
            foreColor = new Color(0 , 0 , 255);
        }
    };
    // 为三个菜单添加事件监听器
    redItem.addActionListener(menuListener);
    greenItem.addActionListener(menuListener);
    blueItem.addActionListener(menuListener);
    // 将菜单项组合成右键菜单
    pop.add(redItem);
    pop.add(greenItem);
    pop.add(blueItem);
    // 将右键菜单添加到 drawArea 对象中
    drawArea.add(pop);
    // 将 image 对象的背景色填充成白色
    g.fillRect(0 , 0 ,AREA_WIDTH , AREA_HEIGHT);
    drawArea.setPreferredSize(new Dimension(AREA_WIDTH , AREA_HEIGHT));
    // 监听鼠标移动动作
    drawArea.addMouseMotionListener(new MouseMotionAdapter()
    {
        // 实现按下鼠标键并拖动的事件处理器
        public void mouseDragged(MouseEvent e)
        {
            // 如果 preX 和 preY 大于 0
            if (preX > 0 && preY > 0)
            {
                // 设置当前颜色
                g.setColor(foreColor);
                // 绘制从上一次鼠标拖动事件点到本次鼠标拖动事件点的线段
                g.drawLine(preX , preY , e.getX() , e.getY());
            }
            // 将当前鼠标事件点的 X、Y 坐标保存起来
            preX = e.getX();
            preY = e.getY();
            // 重绘 drawArea 对象
```

```
                drawArea.repaint();
            }
        });
        // 监听鼠标事件
        drawArea.addMouseListener(new MouseAdapter()
        {
            // 实现鼠标键松开的事件处理器
            public void mouseReleased(MouseEvent e)
            {
                // 弹出右键菜单
                if (e.isPopupTrigger())
                {
                    pop.show(drawArea , e.getX() , e.getY());
                }
                // 松开鼠标键时，把上一次鼠标拖动事件的 X、Y 坐标设为-1
                preX = -1;
                preY = -1;
            }
        });
        f.add(drawArea);
        f.pack();
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new HandDraw().init();
    }
    class DrawCanvas extends Canvas
    {
        // 重写 Canvas 的 paint 方法，实现绘画
        public void paint(Graphics g)
        {
            // 将 image 绘制到该组件上
            g.drawImage(image , 0 , 0 , null);
        }
    }
}
```

实现手绘功能其实是一种假象：表面上看起来可以随鼠标移动自由画曲线，实际上依然利用 Graphics 的 drawLine()方法画直线，每条直线都是从上一次鼠标拖动事件发生点画到本次鼠标拖动事件发生点。当鼠标拖动时，两次鼠标拖动事件发生点的距离很小，多条极短的直线连接起来，肉眼看起来就是鼠标拖动的轨迹了。上面程序还增加了右键菜单来选择画笔颜色。

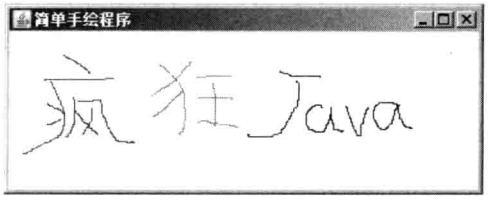


图 11.30 手绘窗口

运行上面程序，出现一个空白窗口，用户可以使用鼠标在该窗口上拖出任意的曲线，如图 11.30 所示。

提示：

上面程序进行手绘时只能选择红、绿、蓝三种颜色，不能调出像 Windows 的颜色选择对话框那种“专业”的颜色选择工具。实际上，Swing 提供了对颜色选择对话框的支持，如果结合 Swing 提供的颜色选择对话框，就可以选择任意的颜色进行画图，并可以提供一些按钮让用户选择绘制直线、折线、多边形等几何图形。如果为该程序分别建立多个 BufferedImage 对象，就可实现多图层效果（每个 BufferedImage 代表一个图层）。

11.8.2 使用 ImageIO 输入/输出位图

如果希望可以访问磁盘上的位图文件，例如 GIF、JPG 等格式的位图，则需要利用 ImageIO 工具类。ImageIO 利用 ImageReader 和 ImageWriter 读写图形文件，通常程序无须关心该类底层的细节，只需要利用该工具类来读写图形文件即可。

ImageIO 类并不支持读写全部格式的图形文件，程序可以通过 ImageIO 类的如下几个静态方法来访

问该类所支持读写的图形文件格式。

- static String[] getReaderFileSuffixes()：返回一个 String 数组，该数组列出 ImageIO 所有能读的图形文件的文件后缀。
- static String[] getReaderFormatNames()：返回一个 String 数组，该数组列出 ImageIO 所有能读的图形文件的非正式格式名称。
- static String[] getWriterFileSuffixes()：返回一个 String 数组，该数组列出 ImageIO 所有能写的图形文件的文件后缀。
- static String[] getWriterFormatNames()：返回一个 String 数组，该数组列出 ImageIO 所有能写的图形文件的非正式格式名称。

下面程序测试了 ImageIO 所支持读写的全部文件格式。

程序清单：codes\11\11.8\ImageIOTest.java

```
public class ImageIOTest
{
    public static void main(String[] args)
    {
        String[] readFormat = ImageIO.getReaderFormatNames();
        System.out.println("-----Image 能读的所有图形文件格式-----");
        for (String tmp : readFormat)
        {
            System.out.println(tmp);
        }
        String[] writeFormat = ImageIO.getWriterFormatNames();
        System.out.println("-----Image 能写的所有图形文件格式-----");
        for (String tmp : writeFormat)
        {
            System.out.println(tmp);
        }
    }
}
```

运行上面程序就可以看到 Java 所支持的图形文件格式，通过运行结果可以看出，AWT 并不支持 ico 等图标格式。因此，如果需要在 Java 程序中为按钮、菜单等指定图标，也不要使用 ico 格式的图标文件，而应该使用 JPG、GIF 等格式的图形文件。

ImageIO 类包含两个静态方法：read()和 write()，通过这两个方法即可完成对位图文件的读写，调用 wirte()方法输出图形文件时需要指定输出的图形格式，例如 GIF、JPEG 等。下面程序可以将一个原始位图缩小成另一个位图后输出。

程序清单：codes\11\11.8\ZoomImage.java

```
public class ZoomImage
{
    // 下面两个常量设置缩小后图片的大小
    private final int WIDTH = 80;
    private final int HEIGHT = 60;
    // 定义一个 BufferedImage 对象，用于保存缩小后的位图
    BufferedImage image = new BufferedImage(WIDTH , HEIGHT
        , BufferedImage. TYPE_INT_RGB);
    Graphics g = image.getGraphics();
    public void zoom()throws Exception
    {
        // 读取原始位图
        Image srcImage = ImageIO.read(new File("image/board.jpg"));
        // 将原始位图缩小后绘制到 image 对象中
        g.drawImage(srcImage , 0 , 0 , WIDTH , HEIGHT , null);
        // 将 image 对象输出到磁盘文件中
        ImageIO.write(image , "jpeg"
            , new File(System.currentTimeMillis() + ".jpg"));
    }
    public static void main(String[] args)throws Exception
    {
        new ZoomImage().zoom();
    }
}
```

上面程序中第一行粗体字代码从磁盘中读取一个位图文件，第二行粗体字代码则将原始位图按指定大小绘制到 image 对象中，第三行代码再将 image 对象输出，这就完成了位图的缩小（实际上不一定是缩小，程序总是将原始位图缩放到 WIDTH、HEIGHT 常量指定的大小）并输出。

提示：

上面程序总是使用 board.jpg 文件作为原始图片文件，总是缩放到 80 × 60 的尺寸，且总是以当前时间作为文件名来输出该文件，这是为了简化该程序。如果为该程序增加图形界面，允许用户选择需要缩放的原始图片文件和缩放后的目标文件名，并可以设置缩放后的尺寸，该程序将具有更好的实用性。对位图文件进行缩放是非常实用的功能，大部分 Web 应用都允许用户上传的图片，而 Web 应用则需要对用户上传的位图生成相应的缩略图，这就需要对位图进行缩放。

利用 ImageIO 读取磁盘上的位图，然后将这图绘制在 AWT 组件上，就可以做出更加丰富多彩的图形界面程序。

下面程序再次改写第 4 章的五子棋游戏，为该游戏增加图形用户界面，这种改写很简单，只需要改变如下两个地方即可。

- 原来是在控制台打印棋盘和棋子，现在改为使用位图在窗口中绘制棋盘和棋子。
- 原来是靠用户输入下棋坐标，现在改为当用户单击鼠标键时获取下棋坐标，此处需要将鼠标事件的 *X*、*Y* 坐标转换为棋盘数组的坐标。

程序清单：codes\11\11.8\Gobang.java

```
public class Gobang
{
    // 下面三个位图分别代表棋盘、黑子、白子
    BufferedImage table;
    BufferedImage black;
    BufferedImage white;
    // 当鼠标移动时的选择框
    BufferedImage selected;
    // 定义棋盘的大小
    private static int BOARD_SIZE = 15;
    // 定义棋盘宽、高多少个像素
    private final int TABLE_WIDTH = 535;
    private final int TABLE_HETGHT = 536;
    // 定义棋盘坐标的像素值和棋盘数组之间的比率
    private final int RATE = TABLE_WIDTH / BOARD_SIZE;
    // 定义棋盘坐标的像素值和棋盘数组之间的偏移距离
    private final int X_OFFSET = 5;
    private final int Y_OFFSET = 6;
    // 定义一个二维数组来充当棋盘
    private String[][] board = new String[BOARD_SIZE][BOARD_SIZE];
    // 五子棋游戏的窗口
    JFrame f = new JFrame("五子棋游戏");
    // 五子棋游戏棋盘对应的 Canvas 组件
    ChessBoard chessBoard = new ChessBoard();
    // 当前选中点的坐标
    private int selectedX = -1;
    private int selectedY = -1;
    public void init()throws Exception
    {
        table = ImageIO.read(new File("image/board.jpg"));
        black = ImageIO.read(new File("image/black.gif"));
        white = ImageIO.read(new File("image/white.gif"));
        selected = ImageIO.read(new File("image/selected.gif"));
        // 把每个元素赋为"╋"，"╋"代表没有棋子
        for (int i = 0 ; i < BOARD_SIZE ; i++)
        {
            for ( int j = 0 ; j < BOARD_SIZE ; j++)
            {
```

```
                board[i][j] = "╋";
            }
        }
        chessBoard.setPreferredSize(new Dimension(
            TABLE_WIDTH , TABLE_HETGHT));
        chessBoard.addMouseListener(new MouseAdapter()
        {
            public void mouseClicked(MouseEvent e)
            {
                // 将用户鼠标事件的坐标转换成棋子数组的坐标
                int xPos = (int)((e.getX() - X_OFFSET) / RATE);
                int yPos = (int)((e.getY() - Y_OFFSET ) / RATE);
                board[xPos][yPos] = "●";
                /*
                电脑随机生成两个整数，作为电脑下棋的坐标，赋给board数组
                还涉及：
                1.如果下棋的点已经有棋子，不能重复下棋
                2.每次下棋后，需要扫描谁赢了
                */
                chessBoard.repaint();
            }
            // 当鼠标退出棋盘区后，复位选中点坐标
            public void mouseExited(MouseEvent e)
            {
                selectedX = -1;
                selectedY = -1;
                chessBoard.repaint();
            }
        });
        chessBoard.addMouseMotionListener(new MouseMotionAdapter()
        {
            // 当鼠标移动时，改变选中点的坐标
            public void mouseMoved(MouseEvent e)
            {
                selectedX = (e.getX() - X_OFFSET) / RATE;
                selectedY = (e.getY() - Y_OFFSET) / RATE;
                chessBoard.repaint();
            }
        });
        f.add(chessBoard);
        f.pack();
        f.setVisible(true);
    }
    public static void main(String[] args)throws Exception
    {
        Gobang gb = new Gobang();
        gb.init();
    }
    class ChessBoard extends JPanel
    {
        // 重写JPanel的paint方法，实现绘画
        public void paint(Graphics g)
        {
            // 绘制五子棋棋盘
            g.drawImage(table , 0 , 0 , null);
            // 绘制选中点的红框
            if (selectedX >= 0 && selectedY >= 0)
                g.drawImage(selected , selectedX * RATE + X_OFFSET ,
            selectedY * RATE + Y_OFFSET, null);
            // 遍历数组，绘制棋子
            for (int i = 0 ; i < BOARD_SIZE ; i++)
            {
                for ( int j = 0 ; j < BOARD_SIZE ; j++)
                {
                    // 绘制黑棋
                    if (board[i][j].equals("●"))
                    {
                        g.drawImage(black , i * RATE + X_OFFSET
```

```
                        , j * RATE + Y_OFFSET, null);
                }
                // 绘制白棋
                if (board[i][j].equals("○"))
                {
                    g.drawImage(white, i * RATE  + X_OFFSET
                        , j * RATE  + Y_OFFSET, null);
                }
            }
        }
    }
}
```

上面程序中前面一段粗体字代码负责监听鼠标单击动作，负责把鼠标动作的坐标转换成棋盘数组的坐标，并将对应的数组元素赋值为“●”；后面一段粗体字代码则负责在窗口中绘制棋盘和棋子：先直接绘制棋盘位图，接着遍历棋盘数组，如果数组元素是“●”，则在对应点绘制黑棋，如果数组元素是“○”，则在对应点绘制白棋。

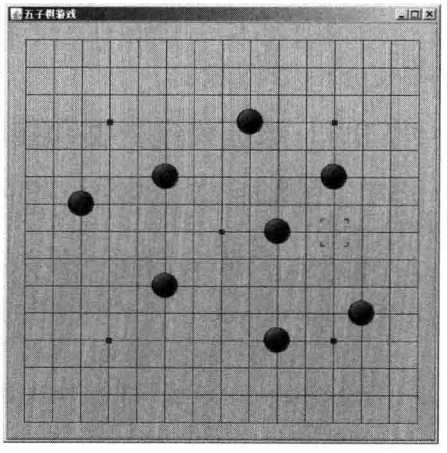


图 11.31 五子棋游戏界面

提示： 上面程序为了避免游戏时产生闪烁感，将棋盘所用的画图区改为继承 JPanel 类，游戏窗口改为使用 JFrame 类，这两个类都是 Swing 组件，Swing 组件的绘图功能提供了双缓冲技术，可以避免图像闪烁。

运行上面程序，会看到如图 11.31 所示的游戏界面。

上面游戏界面中还有一个红色选中框，提示用户鼠标所在的落棋点，这是通过监听鼠标移动事件实现的——当鼠标在游戏界面移动时，程序根据鼠标移动事件发生的坐标来绘制红色选中框。

提示： 上面程序中使用了字符串数组来保存下棋的状态，其实完全可以使用一个 byte[][]数组来保存下棋的状态；数组元素为 0 代表没有棋子；数组元素为 1 代表白棋；数组元素为 2 代表黑棋。上面的游戏程序已经接近完成了，读者只需要按上面思路就可完成这个五子棋游戏，如果能为电脑下棋增加一些智能就更好了。另外，其他小游戏如俄罗斯方块、贪食蛇、连连看、梭哈、斗地主等，只要按这种编程思路来开发都会变得非常简单。实际上，很多程序其实没有想象的那么难，读者只要认真阅读本书，认真完成每章后面的作业，一定可以成为专业的 Java 程序员。

11.9 剪贴板

当进行复制、剪切、粘贴等 Windows 操作时，也许读者从未想过这些操作的实现过程。实际上这是一个看似简单的过程：复制、剪切把一个程序中的数据放置到剪贴板中，而粘贴则读取剪贴板中的数据，并将该数据放入另一个程序中。

剪贴板的复制、剪切和粘贴的过程看似很简单，但实现起来则存在一些具体问题需要处理——假设从一个文字处理程序中复制文本，然后将这段文本复制到另一个文字处理程序中，肯定希望该文字能保持原来的风格，也就是说，剪贴板中必须保留文字原来的格式信息；如果只是将文字复制到纯文本域中，则可以无须包含文字原来的格式信息。除此之外，可能还希望将图像等其他对象复制到剪贴板中。为了处理这种复杂的剪贴板操作，数据提供者（复制、剪切内容的源程序）允许使用多种格式的剪贴板数据，而数据的使用者（粘贴内容的目标程序）则可以从多种格式中选择所需的格式。

提示：

因为 AWT 的实现依赖于底层运行平台的实现，因此 AWT 剪贴板在不同平台上所支持的传输的对象类型并不完全相同。其中 Microsoft、Macintosh 的剪贴板支持传输富格式文本、图像、纯文本等数据，而 X Window 的剪贴板功能则比较有限，它仅仅支持纯文本的剪切和粘贴。读者可以通过查看 JRE 的 jre/lib/flavormap.properties 文件来了解该平台支持哪些类型的对象可以在 Java 程序和系统剪贴板之间传递。

AWT 支持两种剪贴板：本地剪贴板和系统剪贴板。如果在同一个虚拟机的不同窗口之间进行数据传递，则使用 AWT 自己的本地剪贴板就可以了。本地剪贴板则与运行平台无关，可以传输任意格式的数据。如果需要在不同的虚拟机之间传递数据，或者需要在 Java 程序与第三方程序之间传递数据，那就需要使用系统剪贴板了。

11.9.1 数据传递的类和接口

AWT 中剪贴板相关操作的接口和类被放在 java.awt.datatransfer 包下，下面是该包下重要的接口和类的相关说明。

- Clipboard：代表一个剪贴板实例，这个剪贴板既可以是系统剪贴板，也可以是本地剪贴板。
- ClipboardOwner：剪贴板内容的所有者接口，当剪贴板内容的所有权被修改时，系统将会触发该所有者的 lostOwnership 事件处理器。
- Transferable：该接口的实例代表放进剪贴板中的传输对象。
- DataFlavor：用于表述剪贴板中的数据格式。
- StringSelection：Transferable 的实现类，用于传输文本字符串。
- FlavorListener：数据格式监听器接口。
- FlavorEvent：该类的实例封装了数据格式改变的事件。

11.9.2 传递文本

传递文本是最简单的情形，因为 AWT 已经提供了一个 StringSelection 用于传输文本字符串。将一段文本内容（字符串对象）放进剪贴板中的步骤如下。

① 创建一个 Clipboard 实例，既可以创建系统剪贴板，也可以创建本地剪贴板。创建系统剪贴板通过如下代码：

```
Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
```

创建本地剪贴板通过如下代码：

```
Clipboard clipboard = new Clipboard("cb");
```

② 将需要放入剪贴板中的字符串封装成 StringSelection 对象，如下代码所示：

```
StringSelection st = new StringSelection(targetStr);
```

③ 调用剪贴板对象的 setContents()方法将 StringSelection 放进剪贴板中，该方法需要两个参数，第一个参数是 Transferable 对象，代表放进剪贴板中的对象；第二个参数是 ClipboardOwner 对象，代表剪贴板数据的所有者，通常无须关心剪贴板数据的所有者，所以把第二个参数设为 null。

```
clipboard.setContents(st , null);
```

从剪贴板中取出数据则比较简单，调用 Clipboard 对象的 getData(DataFlavor flavor)方法即可取出剪贴板中指定格式的内容，如果指定 flavor 的数据不存在，该方法将引发 UnsupportedFlavorException 异常。为了避免出现异常，可以先调用 Clipboard 对象的 isDataFlavorAvailable(DataFlavor flavor)来判断指定 flavor 的数据是否存在。如下代码所示：

```
if (clipboard.isDataFlavorAvailable(DataFlavor.stringFlavor))
```

```
{
    String content = (String)clipboard.getData(DataFlavor.stringFlavor);
}
```

下面程序是一个利用系统剪贴板进行复制、粘贴的简单程序。

程序清单：codes\11\11.9\SimpleClipboard.java

```
public class SimpleClipboard
{
    private Frame f = new Frame("简单的剪贴板程序");
    // 获取系统剪贴板
    private Clipboard clipboard = Toolkit
        .getDefaultToolkit().getSystemClipboard();
    // 下面是创建本地剪贴板的代码
    // Clipboard clipboard = new Clipboard("cb");  // ①
    // 用于复制文本的文本框
    private TextArea jtaCopyTo = new TextArea(5,20);
    // 用于粘贴文本的文本框
    private TextArea jtaPaste = new TextArea(5,20);
    private Button btCopy = new Button("复制"); // 复制按钮
    private Button btPaste = new Button("粘贴"); // 粘贴按钮
    public void init()
    {
        Panel p = new Panel();
        p.add(btCopy);
        p.add(btPaste);
        btCopy.addActionListener(event ->
        {
            // 将一个多行文本域里的字符串封装成 StringSelection 对象
            StringSelection contents = new
                StringSelection(jtaCopyTo.getText());
            // 将 StringSelection 对象放入剪贴板
            clipboard.setContents(contents, null);
        });
        btPaste.addActionListener(event ->
        {
            // 如果剪贴板中包含 stringFlavor 内容
            if (clipboard.isDataFlavorAvailable(DataFlavor.stringFlavor))
            {
                try
                {
                    // 取出剪贴板中的 stringFlavor 内容
                    String content = (String)clipboard
                        .getData(DataFlavor.stringFlavor);
                    jtaPaste.append(content);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        });
        // 创建一个水平排列的 Box 容器
        Box box = new Box(BoxLayout.X_AXIS);
        // 将两个多行文本域放在 Box 容器中
        box.add(jtaCopyTo);
        box.add(jtaPaste);
        // 将按钮所在的 Panel、Box 容器添加到 Frame 窗口中
        f.add(p,BorderLayout.SOUTH);
        f.add(box,BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new SimpleClipboard().init();
    }
}
```

上面程序中“复制”按钮的事件监听器负责将第一个文本域的内容复制到系统剪贴板中，“粘贴”按钮的事件监听器则负责取出系统剪贴板中的 stringFlavor 内容，并将其添加到第二个文本域内。运行上面程序，将看到如图 11.32 所示的结果。

因为程序使用的是系统剪贴板，因此可以通过 Windows 的剪贴簿查看器来查看程序放入剪贴板中的内容。在 Windows 的“开始”菜单中运行“clipbrd”程序，将可以看到如图 11.33 所示的窗口。

提示：

Windows 7 系统已经删除了默认的剪贴板查看器，因此读者可以到 Windows XP 的 C:\windows\system32\目录下将 clipbrd.exe 文件复制过来。

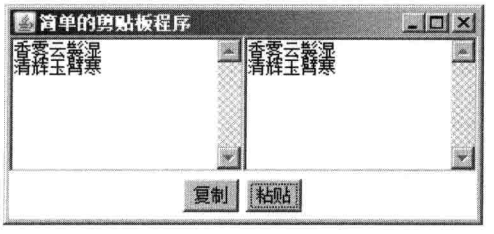


图 11.32　使用剪贴板复制、粘贴文本内容

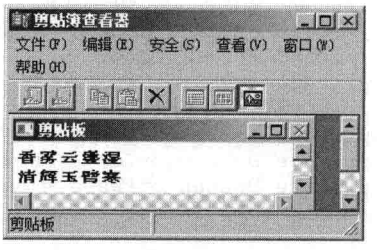


图 11.33　通过剪贴簿查看器查看剪贴板中的内容

11.9.3　使用系统剪贴板传递图像

前面已经介绍了，Transferable 接口代表可以放入剪贴板的传输对象，所以如果希望将图像放入剪贴板内，则必须提供一个 Transferable 接口的实现类，该实现类其实很简单，它封装一个 image 对象，并且向外表现为 imageFlavor 内容。

注意：

JDK 为 Transferable 接口仅提供了一个 StringSelection 实现类，用于封装字符串内容。但 JDK 在 DataFlavor 类中提供了一个 imageFlavor 常量，用于代表图像格式的 DataFlavor，并负责执行所有的复杂操作，以便进行 Java 图像和剪贴板图像的转换。

下面程序实现了一个 ImageSelection 类，该类实现了 Transferable 接口，并实现了该接口所包含的三个方法。

程序清单：codes\11\11.9\ImageSelection.java

```
public class ImageSelection implements Transferable
{
    private Image image;
    // 构造器，负责持有一个 Image 对象
    public ImageSelection(Image image)
    {
        this.image = image;
    }
    // 返回该 Transferable 对象所支持的所有 DataFlavor
    public DataFlavor[] getTransferDataFlavors()
    {
        return new DataFlavor[]{DataFlavor.imageFlavor};
    }
    // 取出该 Transferable 对象里实际的数据
    public Object getTransferData(DataFlavor flavor)
        throws UnsupportedFlavorException
    {
        if(flavor.equals(DataFlavor.imageFlavor))
        {
            return image;
        }
        else
```

```
        {
            throw new UnsupportedFlavorException(flavor);
        }
    }
    // 返回该 Transferable 对象是否支持指定的 DataFlavor
    public boolean isDataFlavorSupported(DataFlavor flavor)
    {
        return flavor.equals(DataFlavor.imageFlavor);
    }
}
```

有了 ImageSelection 封装类后，程序就可以将指定的 Image 对象包装成 ImageSelection 对象放入剪贴板中。下面程序对前面的 HandDraw 程序进行了改进，改进后的程序允许将用户手绘的图像复制到剪贴板中，也可以把剪贴板里的图像粘贴到该程序中。

程序清单：codes\11\11.9\CopyImage.java

```
public class CopyImage
{
    // 系统剪贴板
    private Clipboard clipboard = Toolkit
        .getDefaultToolkit().getSystemClipboard();
    // 使用 ArrayList 来保存所有粘贴进来的 Image——就是当成图层处理
    java.util.List<Image> imageList = new ArrayList<>();
    // 下面代码与前面 HandDraw 程序中控制绘图的代码一样，省略这部分代码
    ...
        f.add(drawArea);
        Panel p = new Panel();
        Button copy = new Button("复制");
        Button paste = new Button("粘贴");
        copy.addActionListener(event ->
        {
            // 将 image 对象封装成 ImageSelection 对象
            ImageSelection contents = new ImageSelection(image);
            // 将 ImageSelection 对象放入剪贴板
            clipboard.setContents(contents, null);
        });
        paste.addActionListener(event ->
        {
            // 如果剪贴板中包含 imageFlavor 内容
            if (clipboard.isDataFlavorAvailable(DataFlavor.imageFlavor))
            {
                try
                {
                    // 取出剪贴板中的 imageFlavor 内容，并将其添加到 List 集合中
                    imageList.add((Image)clipboard
                        .getData(DataFlavor.imageFlavor));
                    drawArea.repaint();
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        });
        p.add(copy);
        p.add(paste);
        f.add(p , BorderLayout.SOUTH);
        f.pack();
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new CopyImage().init();
    }
    class DrawCanvas extends Canvas
    {
        // 重写 Canvas 的 paint 方法，实现绘画
        public void paint(Graphics g)
```

```
            {
                // 将 image 绘制到该组件上
                g.drawImage(image , 0 , 0 , null);
                // 将 List 里的所有 Image 对象都绘制出来
                for (Image img : imageList)
                {
                    g.drawImage(img , 0 , 0 , null);
                }
            }
        }
    }
```

上面程序实现图像复制、粘贴的代码也很简单，就是程序中两段粗体字代码部分：第一段粗体字代码实现了图像复制功能，将 image 对象封装成 ImageSelection 对象，然后调用 Clipboard 的 setContents() 方法将该对象放入剪贴板中；第二段粗体字代码实现了图像粘贴功能，取出剪贴板中的 imageFlavor 内容，返回一个 Image 对象，将该 Image 对象添加到程序的 imageList 集合中。

上面程序中使用了“图层”的概念，使用 imageList 集合来保存所有粘贴到程序中的 Image——每个 Image 就是一个图层，重绘 Canvas 对象时需要绘制 imageList 集合中的每个 image 图像。运行上面程序，当用户在程序中绘制了一些图像后，单击“复制”按钮，将看到程序将该图像复制到了系统剪贴板中，如图 11.34 所示。

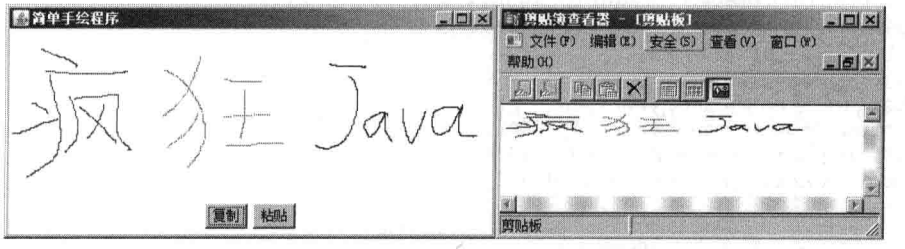


图 11.34　将 Java 程序中的图像放入系统剪贴板中

如果在其他程序中复制一块图像区域（由其他程序负责将图片放入系统剪贴板中），然后单击本程序中的“粘贴”按钮，就可以将该图像粘贴到本程序中。如图 11.35 所示，将其他程序中的图像复制到 Java 程序中。

图 11.35　将画图程序中的图像复制到 Java 程序中

11.9.4　使用本地剪贴板传递对象引用

本地剪贴板可以保存任何类型的 Java 对象，包括自定义类型的对象。为了将任意类型的 Java 对象保存到剪贴板中，DataFlavor 里提供了一个 javaJVMLocalObjectMimeType 的常量，该常量是一个 MIME 类型字符串：application/x-java-jvm-local-objectref，将 Java 对象放入本地剪贴板中必须使用该 MIME 类型。该 MIME 类型表示仅将对象引用复制到剪贴板中，对象引用只有在同一个虚拟机中才有效，所以只能使用本地剪贴板。创建本地剪贴板的代码如下：

```
Clipboard clipboard = new Clipboard("cp");
```

创建本地剪贴板时需要传入一个字符串，该字符串是剪贴板的名字，通过这种方式允许在一个程序中创建本地剪贴板，就可以实现像 Word 那种多次复制，选择剪贴板粘贴的功能。

注意：

本地剪贴板是 JVM 负责维护的内存区，因此本地剪贴板会随虚拟机的结束而销毁。因此一旦 Java 程序退出，本地剪贴板中的内容将会丢失。

Java 并没有提供封装对象引用的 Transferable 实现类，因此必须自己实现该接口。实现该接口与前面的 ImageSelection 基本相似，一样要实现该接口的三个方法，并持有某个对象的引用。看如下代码。

程序清单：codes\11\11.9\LocalObjectSelection.java

```
public class LocalObjectSelection implements Transferable
{
    // 持有一个对象的引用
    private Object obj;
    public LocalObjectSelection(Object obj)
    {
        this.obj = obj;
    }
    // 返回该 Transferable 对象支持的 DataFlavor
    public DataFlavor[] getTransferDataFlavors()
    {
        DataFlavor[] flavors = new DataFlavor[2];
        //获取被封装对象的类型
        Class clazz = obj.getClass();
        String mimeType = "application/x-java-jvm-local-objectref;"
            + "class=" + clazz.getName();
        try
        {
            flavors[0] = new DataFlavor(mimeType);
            flavors[1] = DataFlavor.stringFlavor;
            return flavors;
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
            return null;
        }
    }
    // 取出该 Transferable 对象封装的数据
    public Object getTransferData(DataFlavor flavor)
        throws UnsupportedFlavorException
    {
        if(!isDataFlavorSupported(flavor))
        {
            throw new UnsupportedFlavorException(flavor);
        }
        if (flavor.equals(DataFlavor.stringFlavor))
        {
            return obj.toString();
        }
        return obj;
    }
    public boolean isDataFlavorSupported(DataFlavor flavor)
    {
        return flavor.equals(DataFlavor.stringFlavor) ||
            flavor.getPrimaryType().equals("application")
            && flavor.getSubType().equals("x-java-jvm-local-objectref")
           && flavor.getRepresentationClass().isAssignableFrom(obj.getClass());
    }
}
```

上面程序创建了一个 DataFlavor 对象，用于表示本地 Person 对象引用的数据格式。创建 DataFlavor 对象可以使用如下构造器。

➢ DataFlavor(String mimeType)：根据 mimeType 字符串构造 DataFlavor。

程序使用上面构造器创建了 MIME 类型为 "application/x-java-jvm-local-objectref;class="+clazz.getName()的 DataFlavor 对象，它表示封装本地对象引用的数据格式。

有了上面的 LocalObjectSelection 封装类后，就可以使用该类来封装某个对象的引用，从而将该对象的引用放入本地剪贴板中。下面程序示范了如何将一个 Person 对象放入本地剪贴板中，以及从本地剪贴板中读取该 Person 对象。

程序清单：codes\11\11.9\CopyPerson.java

```
public class CopyPerson
{
    Frame f = new Frame("复制对象");
    Button copy = new Button("复制");
    Button paste = new Button("粘贴");
    TextField name = new TextField(15);
    TextField age = new TextField(15);
    TextArea ta = new TextArea(3 , 30);
    // 创建本地剪贴板
    Clipboard clipboard = new Clipboard("cp");
    public void init()
    {
        Panel p = new Panel();
        p.add(new Label("姓名"));
        p.add(name);
        p.add(new Label("年龄"));
        p.add(age);
        f.add(p , BorderLayout.NORTH);
        f.add(ta);
        Panel bp = new Panel();
        // 为“复制”按钮添加事件监听器
        copy.addActionListener(e -> copyPerson());
        // 为“粘贴”按钮添加事件监听器
        paste.addActionListener(e ->
        {
            try
            {
                readPerson();
            }
            catch (Exception ee)
            {
                ee.printStackTrace();
            }
        });        bp.add(copy);
        bp.add(paste);
        f.add(bp , BorderLayout.SOUTH);
        f.pack();
        f.setVisible(true);
    }
    public void copyPerson()
    {
        // 以 name、age 文本框的内容创建 Person 对象
        Person p = new Person(name.getText()
            , Integer.parseInt(age.getText()));
        // 将 Person 对象封装成 LocalObjectSelection 对象
        LocalObjectSelection ls = new LocalObjectSelection(p);
        // 将 LocalObjectSelection 对象放入本地剪贴板中
        clipboard.setContents(ls , null);
    }
    public void readPerson()throws Exception
    {
        // 创建保存 Person 对象引用的 DataFlavor 对象
        DataFlavor peronFlavor = new DataFlavor(
            "application/x-java-jvm-local-objectref;class=Person");
        // 取出本地剪贴板中的内容
        if (clipboard.isDataFlavorAvailable(DataFlavor.stringFlavor))
        {
            Person p = (Person)clipboard.getData(peronFlavor);
```

```
            ta.setText(p.toString());
        }
    }
    public static void main(String[] args)
    {
        new CopyPerson().init();
    }
}
```

上面程序中的两段粗体字代码实现了复制、粘贴对象的功能，这两段代码与前面复制、粘贴图像的代码并没有太大的区别，只是前面程序使用了 Java 本身提供的 Data.imageFlavor 数据格式，而此处必须自己创建一个 DataFlavor，用以表示封装 Person 引用的 DataFlavor。运行上面程序，在“姓名”文本框内随意输入一个字符串，在“年龄”文本框内输入年龄数字，然后单击“复制”按钮，就可以将根据两个文本框的内容创建的 Person 对象放入本地剪贴板中；单击“粘贴”按钮，就可以从本地剪贴板中读取刚刚放入的数据，如图 11.36 所示。

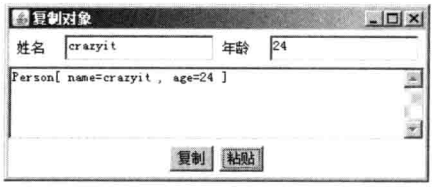


图 11.36 将本地对象复制到本地剪贴板中

上面程序中使用的 Person 类是一个普通的 Java 类，该 Person 类包含了 name 和 age 两个成员变量，并提供了一个包含两个参数的构造器，用于为这两个 Field 成员变量；并重写了 toString()方法，用于返回该 Person 对象的描述性信息。关于 Person 类代码可以参考 codes\11\11.9\CopyPerson.java 文件。

11.9.5 通过系统剪贴板传递 Java 对象

系统剪贴板不仅支持传输文本、图像的基本内容，而且支持传输序列化的 Java 对象和远程对象，复制到剪贴板中的序列化的 Java 对象和远程对象可以使用另一个 Java 程序（不在同一个虚拟机内的程序）来读取。DataFlavor 中提供了 javaSerializedObjectMimeType、javaRemoteObjectMimeType 两个字符串常量来表示序列化的 Java 对象和远程对象的 MIME 类型，这两种 MIME 类型提供了复制对象、读取对象所包含的复杂操作，程序只需创建对应的 Tranferable 实现类即可。

提示：

关于对象序列化请参考本书第 15 章的介绍——如果某个类是可序列化的，则该类的实例可以转换成二进制流，从而可以将该对象通过网络传输或保存到磁盘上。为了保证某个类是可序列化的，只要让该类实现 Serializable 接口即可。

下面程序实现了一个 SerialSelection 类，该类与前面的 ImageSelection、LocalObjectSelection 实现类相似，都需要实现 Tranferable 接口，实现该接口的三个方法，并持有一个可序列化的对象。

程序清单：codes\11\11.9\SerialSelection.java

```
public class SerialSelection implements Transferable
{
    // 持有一个可序列化的对象
    private Serializable obj;
    // 创建该类的对象时传入被持有的对象
    public SerialSelection(Serializable obj)
    {
        this.obj = obj;
    }
    public DataFlavor[] getTransferDataFlavors()
    {
        DataFlavor[] flavors = new DataFlavor[2];
        // 获取被封装对象的类型
        Class clazz = obj.getClass();
        try
        {
            flavors[0] = new DataFlavor(DataFlavor.javaSerializedObjectMimeType
                + ";class=" + clazz.getName());
            flavors[1] = DataFlavor.stringFlavor;
```

```
            return flavors;
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
            return null;
        }
    }
    public Object getTransferData(DataFlavor flavor)
        throws UnsupportedFlavorException
    {
        if(!isDataFlavorSupported(flavor))
        {
            throw new UnsupportedFlavorException(flavor);
        }
        if (flavor.equals(DataFlavor.stringFlavor))
        {
            return obj.toString();
        }
        return obj;
    }
    public boolean isDataFlavorSupported(DataFlavor flavor)
    {
        return flavor.equals(DataFlavor.stringFlavor) ||
            flavor.getPrimaryType().equals("application")
            && flavor.getSubType().equals("x-java-serialized-object")
            && flavor.getRepresentationClass().isAssignableFrom(obj.getClass());
    }
}
```

上面程序也创建了一个 DataFlavor 对象，该对象使用的 MIME 类型为"application/x-java-serialized-object;class=" + clazz.getName()，它表示封装可序列化的 Java 对象的数据格式。

有了上面的 SerialSelection 类后，程序就可以把一个可序列化的对象封装成 SerialSelection 对象，并将该对象放入系统剪贴板中，另一个 Java 程序也可以从系统剪贴板中读取该对象。下面复制、读取 Dog 对象的程序与前面的复制、粘贴 Person 对象的程序非常相似，只是该程序使用的是系统剪贴板，而不是本地剪贴板。

程序清单：codes\11\11.9\CopySerializable.java

```
public class CopySerializable
{
    Frame f = new Frame("复制对象");
    Button copy = new Button("复制");
    Button paste = new Button("粘贴");
    TextField name = new TextField(15);
    TextField age = new TextField(15);
    TextArea ta = new TextArea(3 , 30);
    // 创建系统剪贴板
    Clipboard clipboard = Toolkit.getDefaultToolkit()
        .getSystemClipboard();
    public void init()
    {
        Panel p = new Panel();
        p.add(new Label("姓名"));
        p.add(name);
        p.add(new Label("年龄"));
        p.add(age);
        f.add(p , BorderLayout.NORTH);
        f.add(ta);
        Panel bp = new Panel();
        copy.addActionListener(e -> copyDog());
        paste.addActionListener(e ->
        {
            try
            {
                readDog();
            }
```

```
            catch (Exception ee)
            {
                ee.printStackTrace();
            }
        });         bp.add(copy);
        bp.add(paste);
        f.add(bp , BorderLayout.SOUTH);
        f.pack();
        f.setVisible(true);
    }
    public void copyDog()
    {
        Dog d = new Dog(name.getText()
            , Integer.parseInt(age.getText()));
        // 把 dog 实例封装成 SerialSelection 对象
        SerialSelection ls =new SerialSelection(d);
        // 把 SerialSelection 对象放入系统剪贴板中
        clipboard.setContents(ls , null);
    }
    public void readDog()throws Exception
    {
        DataFlavor peronFlavor = new DataFlavor(DataFlavor
            .javaSerializedObjectMimeType + ";class=Dog");
        if (clipboard.isDataFlavorAvailable(DataFlavor.stringFlavor))
        {
            // 从系统剪贴板中读取数据
            Dog d = (Dog)clipboard.getData(peronFlavor);
            ta.setText(d.toString());
        }
    }
    public static void main(String[] args)
    {
        new CopySerializable().init();
    }
}
```

上面程序中的两段粗体字代码实现了复制、粘贴对象的功能，复制时将 Dog 对象封装成 SerialSelection 对象后放入剪贴板中；读取时先创建 application/x-java-serialized-object;class=Dog 类型的 DataFlavor，然后从剪贴板中读取对应格式的内容即可。运行上面程序，在“姓名”文本框内输入字符串，在“年龄”文本框内输入数字，单击“复制”按钮，即可将该 Dog 对象放入系统剪贴板中。

再次运行上面程序（即启动另一个虚拟机），单击窗口中的“粘贴”按钮，将可以看到系统剪贴板中的 Dog 对象被读取出来，启动系统剪贴板也可以看到被放入剪贴板内的 Dog 对象，如图 11.37 所示。

图 11.37 访问系统剪贴板中的 Dog 对象

上面的 Dog 类也非常简单，为了让该类是可序列化的，让该类实现 Serializable 接口即可。读者可以参考 codes\11\11.9\CopySerializable.java 文件来查看 Dog 类的代码。

11.10 拖放功能

拖放是非常常见的操作，人们经常会通过拖放操作来完成复制、剪切功能，但这种复制、剪切操作无须剪贴板支持，程序将数据从拖放源直接传递给拖放目标。这种通过拖放实现的复制、剪切效果也被称为复制、移动。

人们在拖放源中选中一项或多项元素，然后用鼠标将这些元素拖离它们的初始位置，当拖着这些元

素在拖放目标上松开鼠标按键时，拖放目标将会查询拖放源，进而访问到这些元素的相关信息，并会相应地启动一些动作。例如，从 Windows 资源管理器中把一个文件图标拖放到 WinPad 图标上，WinPad 将会打开该文件。如果在 Eclipse 中选中一段代码，然后将这段代码拖放到另一个位置，系统将会把这段代码从初始位置删除，并将这段代码放到拖放的目标位置。

除此之外，拖放操作还可以与三种键组合使用，用以完成特殊功能。

- 与 Ctrl 键组合使用：表示该拖放操作完成复制功能。例如，可以在 Eclipse 中通过拖放将一段代码剪切到另一个地方，如果在拖放过程中按住 Ctrl 键，系统将完成代码复制，而不是剪切。
- 与 Shift 键组合使用：表示该拖放操作完成移动功能。有些时候直接拖放默认就是进行复制，例如，从 Windows 资源管理器的一个路径将文件图标拖放到另一个路径，默认就是进行文件复制。此时可以结合 Shift 键来进行拖放操作，用以完成移动功能。
- 与 Ctrl、Shift 键组合使用：表示为目标对象建立快捷方式（在 UNIX 等平台上称为链接）。

在拖放操作中，数据从拖放源直接传递给拖放目标，因此拖放操作主要涉及两个对象：拖放源和拖放目标。AWT 已经提供了对拖放源和拖放目标的支持，分别由 DragSource 和 DropTarget 两个类来表示。下面将具体介绍如何在程序中建立拖放源和拖放目标。

实际上，拖放操作与前面介绍的剪贴板操作有一定的类似之处，它们之间的差别在于：拖放操作将数据从拖放源直接传递给拖放目标，而剪贴板操作则是先将数据传递到剪贴板上，然后再从剪贴板传递给目标。剪贴板操作中被传递的内容使用 Transferable 接口来封装，与此类似的是，拖放操作中被传递的内容也使用 Transferable 来封装；剪贴板操作中被传递的数据格式使用 DataFlavor 来表示，拖放操作中同样使用 DataFlavor 来表示被传递的数据格式。

11.10.1 拖放目标

在 GUI 界面中创建拖放目标非常简单，AWT 提供了 DropTarget 类来表示拖放目标，可以通过该类提供的如下构造器来创建一个拖放目标。

- DropTarget(Component c, int ops, DropTargetListener dtl)：将 c 组件创建成一个拖放目标，该拖放目标默认可接受 ops 值所指定的拖放操作。其中 DropTargetListener 是拖放操作的关键，它负责对拖放操作做出相应的响应。ops 可接受如下几个值。
  - DnDConstants.ACTION_COPY：表示“复制”操作的 int 值。
  - DnDConstants.ACTION_COPY_OR_MOVE：表示“复制”或“移动”操作的 int 值。
  - DnDConstants.ACTION_LINK：表示建立“快捷方式”操作的 int 值。
  - DnDConstants.ACTION_MOVE：表示“移动”操作的 int 值。
  - DnDConstants.ACTION_NONE：表示无任何操作的 int 值。

例如，下面代码将一个 JFrame 对象创建成拖放目标。

```
// 将当前窗口创建成拖放目标
new DropTarget(jf, DnDConstants.ACTION_COPY , new ImageDropTargetListener());
```

正如从上面代码中所看到的，创建拖放目标时需要传入一个 DropTargetListener 监听器，该监听器负责处理用户的拖放动作。该监听器里包含如下 5 个事件处理器。

- dragEnter(DropTargetDragEvent dtde)：当光标进入拖放目标时将触发 DropTargetListener 监听器的该方法。
- dragExit(DropTargetEvent dtde)：当光标移出拖放目标时将触发 DropTargetListener 监听器的该方法。
- dragOver(DropTargetDragEvent dtde)：当光标在拖放目标上移动时将触发 DropTargetListener 监听器的该方法。
- drop(DropTargetDropEvent dtde)：当用户在拖放目标上松开鼠标键，拖放结束时将触发 DropTargetListener 监听器的该方法。

- dropActionChanged(DropTargetDragEvent dtde)：当用户在拖放目标上改变了拖放操作，例如按下或松开了 Ctrl 等辅助键时将触发 DropTargetListener 监听器的该方法。

通常程序不想为上面每个方法提供响应，即不想重写 DropTargetListener 监听器的每个方法，只想重写我们关心的方法，可以通过继承 DropTargetAdapter 适配器来创建拖放监听器。下面程序利用拖放目标创建了一个简单的图片浏览工具，当用户把一个或多个图片文件拖入该窗口时，该窗口将会自动打开每个图片文件。

程序清单：codes\11\11.10\DropTargetTest.java

```
public class DropTargetTest
{
    final int DESKTOP_WIDTH = 480;
    final int DESKTOP_HEIGHT = 360;
    final int FRAME_DISTANCE = 30;
    JFrame jf = new JFrame("测试拖放目标——把图片文件拖入该窗口");
    // 定义一个虚拟桌面
    private JDesktopPane desktop = new JDesktopPane();
    // 保存下一个内部窗口的坐标点
    private int nextFrameX;
    private int nextFrameY;
    // 定义内部窗口为虚拟桌面的 1/2 大小
    private int width = DESKTOP_WIDTH / 2;
    private int height = DESKTOP_HEIGHT / 2;
    public void init()
    {
        desktop.setPreferredSize(new Dimension(DESKTOP_WIDTH
            , DESKTOP_HEIGHT));
        // 将当前窗口创建成拖放目标
        new DropTarget(jf, DnDConstants.ACTION_COPY
            , new ImageDropTargetListener());
        jf.add(desktop);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.pack();
        jf.setVisible(true);
    }
    class ImageDropTargetListener extends DropTargetAdapter
    {
        public void drop(DropTargetDropEvent event)
        {
            // 接受复制操作
            event.acceptDrop(DnDConstants.ACTION_COPY);
            // 获取拖放的内容
            Transferable transferable = event.getTransferable();
            DataFlavor[] flavors = transferable.getTransferDataFlavors();
            // 遍历拖放内容里的所有数据格式
            for (int i = 0; i < flavors.length; i++)
            {
                DataFlavor d = flavors[i];
                try
                {
                    // 如果拖放内容的数据格式是文件列表
                    if (d.equals(DataFlavor.javaFileListFlavor))
                    {
                        // 取出拖放操作里的文件列表
                        List fileList = (List)transferable
                            .getTransferData(d);
                        for (Object f : fileList)
                        {
                            // 显示每个文件
                            showImage((File)f , event);
                        }
                    }
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
```

```
                // 强制拖放操作结束，停止阻塞拖放目标
                event.dropComplete(true);    // ①
            }
        }
        // 显示每个文件的工具方法
        private void showImage(File f , DropTargetDropEvent event)
            throws IOException
        {
            Image image = ImageIO.read(f);
            if (image == null)
            {
                // 强制拖放操作结束，停止阻塞拖放目标
                event.dropComplete(true);    // ②
                JOptionPane.showInternalMessageDialog(desktop
                    , "系统不支持这种类型的文件");
                // 方法返回，不会继续操作
                return;
            }
            ImageIcon icon = new ImageIcon(image);
            // 创建内部窗口显示该图片
            JInternalFrame iframe = new JInternalFrame(f.getName()
                , true , true , true , true);
            JLabel imageLabel = new JLabel(icon);
            iframe.add(new JScrollPane(imageLabel));
            desktop.add(iframe);
            // 设置内部窗口的原始位置（内部窗口默认大小是 0×0，放在 0,0 位置）
            iframe.reshape(nextFrameX, nextFrameY, width, height);
            // 使该窗口可见，并尝试选中它
            iframe.show();
            // 计算下一个内部窗口的位置
            nextFrameX += FRAME_DISTANCE;
            nextFrameY += FRAME_DISTANCE;
            if (nextFrameX + width > desktop.getWidth())
                nextFrameX = 0;
            if (nextFrameY + height > desktop.getHeight())
                nextFrameY = 0;
        }
    }
    public static void main(String[] args)
    {
        new DropTargetTest().init();
    }
}
```

上面程序中粗体字代码部分创建了一个拖放目标，创建拖放目标很简单，关键是需要为该拖放目标编写事件监听器。上面程序中采用 ImageDropTargetListener 对象作为拖放目标的事件监听器，该监听器重写了 drop()方法，即当用户在拖放目标上松开鼠标按键时触发该方法。drop()方法里通过 DropTargetDropEvent 对象的 getTransferable()方法取出被拖放的内容，一旦获得被拖放的内容后，程序就可以对这些内容进行适当处理，本例中只处理被拖放格式是 DataFlavor.javaFileListFlavor（文件列表）的内容，处理方法是把所有的图片文件使用内部窗口显示出来。

运行该程序时，只要用户把图片文件拖入该窗口，程序就会使用内部窗口显示该图片。

注意：

上面程序中①②处的 event.dropComplete(true);代码用于强制结束拖放事件，释放拖放目标的阻塞，如果没有调用该方法，或者在弹出对话框之后调用该方法，将会导致拖放目标被阻塞。在对话框被处理之前，拖放目标窗口也不能获得焦点，这可能不是程序希望的效果，所以程序在弹出内部对话框之前强制结束本次拖放操作（因为文件格式不对），释放拖放目标的阻塞。

上面程序中只处理 DataFlavor.javaFileListFlavor 格式的拖放内容；除此之外，还可以处理文本格式的拖放内容，文本格式的拖放内容使用 DataFlavor.stringFlavor 格式来表示。

更复杂的情况是，可能被拖放的内容是带格式的内容，如 text/html 和 text/rtf 等。为了处理这种内容，需要选择合适的数据格式，如下代码所示：

```
// 如果被拖放的内容是 text/html 格式的输入流
if (d. isMimeTypeEqual("text/html") && d.getRepresentationClass()
    == InputStream.class)
{
    String charset = d.getParameter("charset");
    InputStreamReader reader = new InputStreamReader(
    transferable.getTransferData(d) , charset);
    // 使用 IO 流读取拖放操作的内容
    ...
}
```

关于如何使用 IO 流来处理被拖放的内容，读者需要参考本书第 15 章的内容。

11.10.2 拖放源

前面程序使用 DropTarget 创建了一个拖放目标，直接使用系统资源管理器作为拖放源。下面介绍如何在 Java 程序中创建拖放源，创建拖放源比创建拖放目标要复杂一些，因为程序需要把被拖放内容封装成 Transferable 对象。

创建拖放源的步骤如下。

① 调用 DragSource 的 getDefaultDragSource()方法获得与平台关联的 DragSource 对象。

② 调用 DragSource 对象的 createDefaultDragGestureRecognizer(Component c, int actions, DragGestureListener dgl)方法将指定组件转换成拖放源。其中 actions 用于指定该拖放源可接受哪些拖放操作，而 dgl 是一个拖放监听器，该监听器里只有一个方法：dragGestureRecognized()，当系统检测到用户开始拖放时将会触发该方法。

如下代码将会把一个 JLabel 对象转换为拖放源。

```
// 将 srcLabel 组件转换为拖放源
dragSource.createDefaultDragGestureRecognizer(srcLabel,
    DnDConstants.ACTION_COPY_OR_MOVE, new MyDragGestureListener()
```

③ 为第 2 步中的 DragGestureListener 监听器提供实现类，该实现类需要重写该接口里包含的 dragGestureRecognized()方法，该方法负责把拖放内容封装成 Transferable 对象。

下面程序示范了如何把一个 JLabel 转换成拖放源。

程序清单：codes\11\11.10\DragSourceTest.java

```
public class DragSourceTest
{
    JFrame jf = new JFrame("Swing 的拖放支持");
    JLabel srcLabel = new JLabel("Swing 的拖放支持.\n"
        +"将该文本域的内容拖入其他程序.\n");
    public void init()
    {
        DragSource dragSource = DragSource.getDefaultDragSource();
        // 将 srcLabel 转换成拖放源，它能接受复制、移动两种操作
        dragSource.createDefaultDragGestureRecognizer(srcLabel
            , DnDConstants.ACTION_COPY_OR_MOVE
            , event -> {
            // 将 JLabel 里的文本信息包装成 Transferable 对象
            String txt = srcLabel.getText();
            Transferable transferable = new StringSelection(txt);
            // 继续拖放操作，拖放过程中使用手状光标
            event.startDrag(Cursor.getPredefinedCursor(Cursor
                .HAND_CURSOR), transferable);
        });
        jf.add(new JScrollPane(srcLabel));
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.pack();
        jf.setVisible(true);
    }
```

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

上面程序中粗体字代码负责把一个 JLabel 组件创建成拖放源，创建拖放源时指定了一个 DragGestureListener 对象，该对象的 dragGestureRecognized()方法负责将 JLabel 上的文本转换成 Transferable 对象后继续拖放。

运行上面程序后，可以把程序窗口中 JLabel 标签的内容直接拖到 Eclipse 编辑窗口中，或者直接拖到 EditPlus 编辑窗口中。

除此之外，如果程序希望能精确监听光标在拖放源上的每个细节，则可以调用 DragGestureEvent 对象的 startDrag(Cursor dragCursor, Transferable transferable, DragSourceListener dsl)方法来继续拖放操作。该方法需要一个 DragSourceListener 监听器对象，该监听器对象里提供了如下几个方法。

- dragDropEnd(DragSourceDropEvent dsde)：当拖放操作已经完成时将会触发该方法。
- dragEnter(DragSourceDragEvent dsde)：当光标进入拖放源组件时将会触发该方法。
- dragExit(DragSourceEvent dse)：当光标离开拖放源组件时将会触发该方法。
- dragOver(DragSourceDragEvent dsde)：当光标在拖放源组件上移动时将会触发该方法。
- dropActionChanged(DragSourceDragEvent dsde)：当用户在拖放源组件上改变了拖放操作，例如按下或松开 Ctrl 等辅助键时将会触发该方法。

掌握了开发拖放源、拖放目标的方法之后，如果接下来在同一个应用程序中既包括拖放源，也包括拖放目标，这样即可在同一个 Java 程序的不同组件之间相互拖动内容。

11.11 本章小结

本章主要介绍了 Java AWT 编程的基本知识，虽然在实际开发中很少直接使用 AWT 组件来开发 GUI 应用，但本章所介绍的知识会作为 Swing GUI 编程的基础。实际上，AWT 编程的布局管理、事件机制、剪贴板内容依然适合 Swing GUI 编程，所以读者应好好掌握本章内容。

本章介绍了 Java GUI 界面编程以及 AWT 的基本概念，详细介绍了 AWT 容器和布局管理器。本章重点介绍了 Java GUI 编程的事件机制，详细描述了事件源、事件、事件监听器之间的运行机制，AWT 的事件机制也适合 Swing 的事件处理。除此之外，本章也大致介绍了 AWT 里的常用组件，如按钮、文本框、对话框、菜单等。本章还介绍了如何在 Java 程序中绘图，包括绘制各种基本几何图形和绘制位图，并通过简单的弹球游戏介绍了如何在 Java 程序中实现动画效果。

本章最后介绍了 Java 剪贴板的用法，通过使用剪贴板，可以让 Java 程序和操作系统进行数据交换，从而允许把 Java 程序的数据传入平台中的其他程序，也可以把其他程序中的数据传入 Java 程序。

➤➤ 本章练习

1. 开发图形界面计算器。
2. 开发桌面弹球游戏。
3. 开发 Windows 画图程序。
4. 开发图形界面五子棋。

CHAPTER

12

第12章 Swing 编程

本章要点

- Swing 编程基础
- Swing 组件的继承层次
- 常见 Swing 组件的用法
- 使用 JToolBar 创建工具条
- 颜色选择对话框和文件浏览对话框
- Swing 提供的特殊容器
- Swing 的简化拖放操作
- 使用 JLayer 装饰组件
- 开发透明的、不规则形状窗口
- 开发进度条
- 开发滑动条
- 使用 JTree 和 TreeModel 开发树
- 使用 JTable 和 TableModel 开发表格
- 使用 JTextPane 组件

使用 Swing 开发图形界面比 AWT 更加优秀，因为 Swing 是一种轻量级组件，它采用 100%的 Java 实现，不再依赖于本地平台的图形界面，所以可以在所有平台上保持相同的运行效果，对跨平台支持比较出色。

除此之外，Swing 提供了比 AWT 更多的图形界面组件，因此可以开发出更美观的图形界面。由于 AWT 需要调用底层平台的 GUI 实现，所以 AWT 只能使用各种平台上 GUI 组件的交集，这大大限制了 AWT 所支持的 GUI 组件。对 Swing 而言，几乎所有组件都采用纯 Java 实现，所以无须考虑底层平台是否支持该组件，因此 Swing 可以提供如 JTabbedPane、JDesktopPane、JInternalFrame 等特殊的容器，也可以提供像 JTree、JTable、JSpinner、JSlider 等特殊的 GUI 组件。

除此之外，Swing 组件都采用 MVC（Model-View-Controller，即模型－视图－控制器）设计模式，从而可以实现 GUI 组件的显示逻辑和数据逻辑的分离，允许程序员自定义 Render 来改变 GUI 组件的显示外观，提供更多的灵活性。

12.1 Swing 概述

前一章已经介绍过 AWT 和 Swing 的关系，因此不难知道：实际使用 Java 开发图形界面程序时，很少使用 AWT 组件，绝大部分时候都是用 Swing 组件开发的。Swing 是由 100%纯 Java 实现的，不再依赖于本地平台的 GUI，因此可以在所有平台上都保持相同的界面外观。独立于本地平台的 Swing 组件被称为轻量级组件；而依赖于本地平台的 AWT 组件被称为重量级组件。

由于 Swing 的所有组件完全采用 Java 实现，不再调用本地平台的 GUI，所以导致 Swing 图形界面的显示速度要比 AWT 图形界面的显示速度慢一些，但相对于快速发展的硬件设施而言，这种微小的速度差别无妨大碍。

使用 Swing 开发图形界面有如下几个优势。

- Swing 组件不再依赖于本地平台的 GUI，无须采用各种平台的 GUI 交集，因此 Swing 提供了大量图形界面组件，远远超出了 AWT 所提供的图形界面组件集。
- Swing 组件不再依赖于本地平台 GUI，因此不会产生与平台相关的 bug。
- Swing 组件在各种平台上运行时可以保证具有相同的图形界面外观。

Swing 提供的这些优势，让 Java 图形界面程序真正实现了“Write Once, Run Anywhere”的目标。

除此之外，Swing 还有如下两个特征。

- Swing 组件采用 MVC（Model-View-Controller，即模型－视图－控制器）设计模式，其中模型（Model）用于维护组件的各种状态，视图（View）是组件的可视化表现，控制器（Controller）用于控制对于各种事件、组件做出怎样的响应。当模型发生改变时，它会通知所有依赖它的视图，视图会根据模型数据来更新自己。Swing 使用 UI 代理来包装视图和控制器，还有另一个模型对象来维护该组件的状态。例如，按钮 JButton 有一个维护其状态信息的模型 ButtonModel 对象。Swing 组件的模型是自动设置的，因此一般都使用 JButton，而无须关心 ButtonModel 对象。因此，Swing 的 MVC 实现也被称为 Model-Delegate（模型－代理）。

提示： 对于一些简单的 Swing 组件通常无须关心它对应的 Model 对象，但对于一些高级的 Swing 组件，如 JTree、JTable 等需要维护复杂的数据，这些数据就是由该组件对应的 Model 来维护的。另外，通过创建 Model 类的子类或通过实现适当的接口，可以为组件建立自己的模型，然后用 setModel()方法把模型与组件关联起来。

- Swing 在不同的平台上表现一致，并且有能力提供本地平台不支持的显示外观。由于 Swing 组件采用 MVC 模式来维护各组件，所以当组件的外观被改变时，对组件的状态信息（由模型维

护）没有任何影响。因此，Swing 可以使用插拔式外观感觉（Pluggable Look And Feel，PLAF）来控制组件外观，使得 Swing 图形界面在同一个平台上运行时能拥有不同的外观，用户可以选择自己喜欢的外观。相比之下，在 AWT 图形界面中，由于控制组件外观的对等类与具体平台相关，因此 AWT 组件总是具有与本地平台相同的外观。

Swing 提供了多种独立于各种平台的 LAF（Look And Feel），默认是一种名为 Metal 的 LAF，这种 LAF 吸收了 Macintosh 平台的风格，因此显得比较漂亮。Java 7 则提供了一种名为 Nimbus 的 LAF，这种 LAF 更加漂亮。

为了获取当前 JRE 所支持的 LAF，可以借助于 UIManager 的 getInstalledLookAndFeels()方法，如下程序所示。

程序清单：codes\ 12\12.1\AllLookAndFeel.java

```
public class AllLookAndFeel
{
    public static void main(String[] args)
    {
        System.out.println("当前系统可用的所有 LAF:");
        for (UIManager.LookAndFeelInfo info :
            UIManager.getInstalledLookAndFeels())
        {
            System.out.println(info.getName()
                + "--->" + info);
        }
    }
}
```

提示：除了可以使用 Java 默认提供的数量不多的几种 LAF 之外，还有大量的 Java 爱好者提供了各种开源的 LAF，有兴趣的读者可以自行去下载、体验各种 LAF，使用不同的 LAF 可以让 Swing 应用程序更加美观。

12.2 Swing 基本组件的用法

前面已经提到，Swing 为所有的 AWT 组件提供了对应实现（除了 Canvas 组件之外，因为在 Swing 中无须继承 Canvas 组件），通常在 AWT 组件的组件名前添加“J”就变成了对应的 Swing 组件。

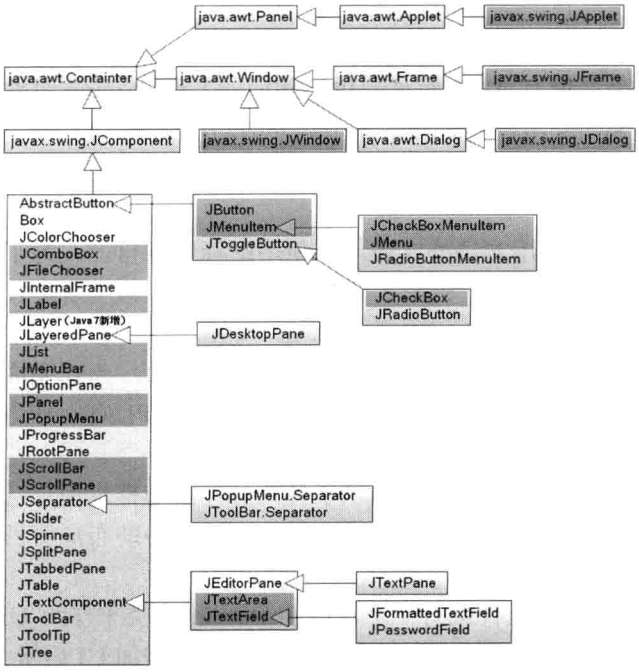


图 12.1 Swing 组件继承层次图

12.2.1 Java 7 的 Swing 组件层次

大部分 Swing 组件都是 JComponent 抽象类的直接或间接子类（并不是全部的 Swing 组件），JComponent 类定义了所有子类组件的通用方法，JComponent 类是 AWT 里 java.awt.Container 类的子类，这也是 AWT 和 Swing 的联系之一。绝大部分 Swing 组件类继承了 Container 类，所以 Swing 组件都可作为容器使用（JFrame 继承了 Frame 类）。图 12.1 显示了 Swing 组件继承层次图。

图 12.1 中绘制了 Swing 所提供的绝大部分组件，其中以灰色区域覆盖的组件可以找到与之对应的 AWT 组件；JWindow 与 AWT 中的 Window 相似，代表没有标题的窗口。读者不难发现这些 Swing 组件的类名和对应 AWT 组件的类型也基

本一致，只要在原来的 AWT 组件类型前添加“J”即可，但有如下几个例外。

- JComboBox：对应于 AWT 里的 Choice 组件，但比 Choice 组件功能更丰富。
- JFileChooser：对应于 AWT 里的 FileDialog 组件。
- JScrollBar：对应于 AWT 里的 Scrollbar 组件，注意两个组件类名中 b 字母的大小写差别。
- JCheckBox：对应于 AWT 里的 Checkbox 组件，注意两个组件类名中 b 字母的大小写差别。
- JCheckBoxMenuItem：对应于 AWT 里的 CheckboxMenuItem 组件，注意两个组件类名中 b 字母的大小写差别。

上面 JCheckBox 和 JCheckBoxMenuItem 与 Checkbox 和 CheckboxMenuItem 的差别主要是由早期 Java 命名不太规范造成的。

注意：

从图 12.1 中可以看出，Swing 中包含了 4 个组件直接继承了 AWT 组件，而不是从 JComponent 派生的，它们分别是：JFrame、JWindow、JDialog 和 JApplet，它们并不是轻量级组件，而是重量级组件（需要部分委托给运行平台上 GUI 组件的对等体）。

将 Swing 组件按功能来分，又可分为如下几类。

- 顶层容器：JFrame、JApplet、JDialog 和 JWindow。
- 中间容器：JPanel、JScrollPane、JSplitPane、JToolBar 等。
- 特殊容器：在用户界面上具有特殊作用的中间容器，如 JInternalFrame、JRootPane、JLayeredPane 和 JDestopPane 等。
- 基本组件：实现人机交互的组件，如 JButton、JComboBox、JList、JMenu、JSlider 等。
- 不可编辑信息的显示组件：向用户显示不可编辑信息的组件，如 JLabel、JProgressBar 和 JToolTip 等。
- 可编辑信息的显示组件：向用户显示能被编辑的格式化信息的组件，如 JTable、JTextArea 和 JTextField 等。
- 特殊对话框组件：可以直接产生特殊对话框的组件，如 JColorChooser 和 JFileChooser 等。

下面将会依次详细介绍各种 Swing 组件的用法。

12.2.2 AWT 组件的 Swing 实现

从图 12.1 中可以看出，Swing 为除了 Canvas 之外的所有 AWT 组件提供了相应的实现，Swing 组件比 AWT 组件的功能更加强大。相对于 AWT 组件，Swing 组件具有如下 4 个额外的功能。

- 可以为 Swing 组件设置提示信息。使用 setToolTipText()方法，为组件设置对用户有帮助的提示信息。
- 很多 Swing 组件如按钮、标签、菜单项等，除了使用文字外，还可以使用图标修饰自己。为了允许在 Swing 组件中使用图标，Swing 为 Icon 接口提供了一个实现类：ImageIcon，该实现类代表一个图像图标。
- 支持插拔式的外观风格。每个 JComponent 对象都有一个相应的 ComponentUI 对象，为它完成所有的绘画、事件处理、决定尺寸大小等工作。ComponentUI 对象依赖当前使用的 PLAF，使用 UIManager.setLookAndFeel()方法可以改变图形界面的外观风格。
- 支持设置边框。Swing 组件可以设置一个或多个边框。Swing 中提供了各式各样的边框供用户选用，也能建立组合边框或自己设计边框。一种空白边框可以用于增大组件，同时协助布局管理器对容器中的组件进行合理的布局。

每个 Swing 组件都有一个对应的 UI 类，例如 JButton 组件就有一个对应的 ButtonUI 类来作为 UI 代理。每个 Swing 组件的 UI 代理的类名总是将该 Swing 组件类名的 J 去掉，然后在后面添加 UI 后缀。

UI 代理类通常是一个抽象基类，不同的 PLAF 会有不同的 UI 代理实现类。Swing 类库中包含了几套 UI 代理，每套 UI 代理都几乎包含了所有 Swing 组件的 ComponentUI 实现，每套这样的实现都被称为一种 PLAF 实现。以 JButton 为例，其 UI 代理的继承层次如图 12.2 所示。

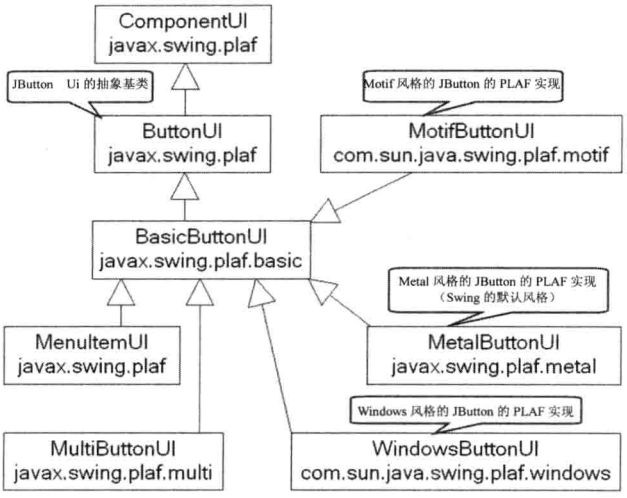


图 12.2 JButton UI 代理的继承层次

如果需要改变程序的外观风格，则可以使用如下代码。

```
try
{
    //设置使用 Windows 风格
    UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
    //通过更新 f 容器以及 f 容器里所有组件的 UI
    SwingUtilities.updateComponentTreeUI(f);
}
catch(Exception e)
{
    e.printStackTrace();
}
```

下面程序示范了使用 Swing 组件来创建窗口应用，该窗口里包含了菜单、右键菜单以及基本 AWT 组件的 Swing 实现。

程序清单：codes\12\12.2\SwingComponent.java

```
public class SwingComponent
{
    JFrame f = new JFrame("测试");
    // 定义一个按钮，并为之指定图标
    Icon okIcon = new ImageIcon("ico/ok.png");
    JButton ok = new JButton("确认" , okIcon);
    // 定义一个单选按钮，初始处于选中状态
    JRadioButton male = new JRadioButton("男" , true);
    // 定义一个单选按钮，初始处于没有选中状态
    JRadioButton female = new JRadioButton("女" , false);
    // 定义一个 ButtonGroup，用于将上面两个 JRadioButton 组合在一起
    ButtonGroup bg = new ButtonGroup();
    // 定义一个复选框，初始处于没有选中状态。
    JCheckBox married = new JCheckBox("是否已婚？" , false);
    String[] colors = new String[]{"红色" , "绿色" , "蓝色"};
    // 定义一个下拉选择框
    JComboBox<String> colorChooser = new JComboBox<>(colors);
    // 定义一个列表选择框
    JList<String> colorList = new JList<>(colors);
    // 定义一个 8 行、20 列的多行文本域
    JTextArea ta = new JTextArea(8, 20);
    // 定义一个 40 列的单行文本域
    JTextField name = new JTextField(40);
```

```
JMenuBar mb = new JMenuBar();
JMenu file = new JMenu("文件");
JMenu edit = new JMenu("编辑");
// 创建“新建”菜单项，并为之指定图标
Icon newIcon = new ImageIcon("ico/new.png");
JMenuItem newItem = new JMenuItem("新建" , newIcon);
// 创建“保存”菜单项，并为之指定图标
Icon saveIcon = new ImageIcon("ico/save.png");
JMenuItem saveItem = new JMenuItem("保存" , saveIcon);
// 创建“退出”菜单项，并为之指定图标
Icon exitIcon = new ImageIcon("ico/exit.png");
JMenuItem exitItem = new JMenuItem("退出" , exitIcon);
JCheckBoxMenuItem autoWrap = new JCheckBoxMenuItem("自动换行");
// 创建“复制”菜单项，并为之指定图标
JMenuItem copyItem = new JMenuItem("复制"
    , new ImageIcon("ico/copy.png"));
// 创建“粘贴”菜单项，并为之指定图标
JMenuItem pasteItem = new JMenuItem("粘贴"
    , new ImageIcon("ico/paste.png"));
JMenu format = new JMenu("格式");
JMenuItem commentItem = new JMenuItem("注释");
JMenuItem cancelItem = new JMenuItem("取消注释");
// 定义一个右键菜单用于设置程序风格
JPopupMenu pop = new JPopupMenu();
// 用于组合 3 个风格菜单项的 ButtonGroup
ButtonGroup flavorGroup = new ButtonGroup();
// 创建 5 个单选按钮，用于设定程序的外观风格
JRadioButtonMenuItem metalItem = new JRadioButtonMenuItem("Metal 风格" , true);
JRadioButtonMenuItem nimbusItem = new JRadioButtonMenuItem("Nimbus 风格");
JRadioButtonMenuItem windowsItem = new JRadioButtonMenuItem("Windows 风格");
JRadioButtonMenuItem classicItem = new JRadioButtonMenuItem("Windows 经典风格");
JRadioButtonMenuItem motifItem = new JRadioButtonMenuItem("Motif 风格");
// ------------------用于执行界面初始化的 init 方法----------------------
public void init()
{
    // 创建一个装载了文本框、按钮的 JPanel
    JPanel bottom = new JPanel();
    bottom.add(name);
    bottom.add(ok);
    f.add(bottom , BorderLayout.SOUTH);
    // 创建一个装载了下拉选择框、三个 JCheckBox 的 JPanel
    JPanel checkPanel = new JPanel();
    checkPanel.add(colorChooser);
    bg.add(male);
    bg.add(female);
    checkPanel.add(male);
    checkPanel.add(female);
    checkPanel.add(married);
    // 创建一个垂直排列组件的 Box，盛装多行文本域 JPanel
    Box topLeft = Box.createVerticalBox();
    // 使用 JScrollPane 作为普通组件的 JViewPort
    JScrollPane taJsp = new JScrollPane(ta);     // ⑤
    topLeft.add(taJsp);
    topLeft.add(checkPanel);
    // 创建一个水平排列组件的 Box，盛装 topLeft、colorList
    Box top = Box.createHorizontalBox();
    top.add(topLeft);
    top.add(colorList);
    // 将 top Box 容器添加到窗口的中间
    f.add(top);
    // -----------下面开始组合菜单，并为菜单添加监听器----------
    // 为 newItem 设置快捷键，设置快捷键时要使用大写字母
    newItem.setAccelerator(KeyStroke.getKeyStroke('N'
        , InputEvent.CTRL_MASK));    // ①
    newItem.addActionListener(e -> ta.append("用户单击了“新建”菜单\n"));
    // 为 file 菜单添加菜单项
    file.add(newItem);
    file.add(saveItem);
    file.add(exitItem);
    // 为 edit 菜单添加菜单项
```

```
        edit.add(autoWrap);
        // 使用 addSeparator 方法添加菜单分隔线
        edit.addSeparator();
        edit.add(copyItem);
        edit.add(pasteItem);
        // 为 commentItem 组件添加提示信息
        commentItem.setToolTipText("将程序代码注释起来！");
        // 为 format 菜单添加菜单项
        format.add(commentItem);
        format.add(cancelItem);
        // 使用添加 new JMenuItem("-")的方式不能添加菜单分隔符
        edit.add(new JMenuItem("-"));
        // 将 format 菜单组合到 edit 菜单中，从而形成二级菜单
        edit.add(format);
        // 将 file、edit 菜单添加到 mb 菜单条中
        mb.add(file);
        mb.add(edit);
        // 为 f 窗口设置菜单条
        f.setJMenuBar(mb);
        // ----------下面开始组合右键菜单，并安装右键菜单----------
        flavorGroup.add(metalItem);
        flavorGroup.add(nimbusItem);
        flavorGroup.add(windowsItem);
        flavorGroup.add(classicItem);
        flavorGroup.add(motifItem);
        pop.add(metalItem);
        pop.add(nimbusItem);
        pop.add(windowsItem);
        pop.add(classicItem);
        pop.add(motifItem);
        // 为 5 个风格菜单创建事件监听器
        ActionListener flavorListener = e -> {
            try
            {
                switch(e.getActionCommand())
                {
                    case "Metal 风格":
                        changeFlavor(1);
                        break;
                    case "Nimbus 风格":
                        changeFlavor(2);
                        break;
                    case "Windows 风格":
                        changeFlavor(3);
                        break;
                    case "Windows 经典风格":
                        changeFlavor(4);
                        break;
                    case "Motif 风格":
                        changeFlavor(5);
                        break;
                }
            }
            catch (Exception ee)
            {
                ee.printStackTrace();
            }
        };
        // 为 5 个风格菜单项添加事件监听器
        metalItem.addActionListener(flavorListener);
        nimbusItem.addActionListener(flavorListener);
        windowsItem.addActionListener(flavorListener);
        classicItem.addActionListener(flavorListener);
        motifItem.addActionListener(flavorListener);
        // 调用该方法即可设置右键菜单，无须使用事件机制
        ta.setComponentPopupMenu(pop);     // ④
        // 设置关闭窗口时，退出程序
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
```

```
        }
        // 定义一个方法，用于改变界面风格
        private void changeFlavor(int flavor)throws Exception
        {
            switch (flavor)
            {
                // 设置Metal风格
                case 1:
                    UIManager.setLookAndFeel(
                    "javax.swing.plaf.metal.MetalLookAndFeel");
                    break;
                // 设置Nimbus风格
                case 2:
                    UIManager.setLookAndFeel(
                    "javax.swing.plaf.nimbus.NimbusLookAndFeel");
                    break;
                // 设置Windows风格
                case 3:
                    UIManager.setLookAndFeel(
                    "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
                    break;
                // 设置Windows经典风格
                case 4:
                    UIManager.setLookAndFeel(
                    "com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel");
                    break;
                // 设置Motif风格
                case 5:
                    UIManager.setLookAndFeel(
                    "com.sun.java.swing.plaf.motif.MotifLookAndFeel");
                    break;
            }
            // 更新 f 窗口内顶级容器以及内部所有组件的 UI
            SwingUtilities.updateComponentTreeUI(f.getContentPane()); // ②
            // 更新 mb 菜单条以及内部所有组件的 UI
            SwingUtilities.updateComponentTreeUI(mb);
            // 更新 pop 右键菜单以及内部所有组件的 UI
            SwingUtilities.updateComponentTreeUI(pop);
        }
        public static void main(String[] args)
        {
            // 设置 Swing 窗口使用 Java 风格
            // JFrame.setDefaultLookAndFeelDecorated(true);   // ③
            new SwingComponent().init();
        }
    }
```

上面程序在创建按钮、菜单项时传入了一个ImageIcon对象，通过这种方式就可以创建带图标的按钮、菜单项。程序的init方法中的粗体字代码用于为comment菜单项添加提示信息。运行上面程序，并通过右键菜单选择“Nimbus LAF”，可以看到如图12.3所示的窗口。

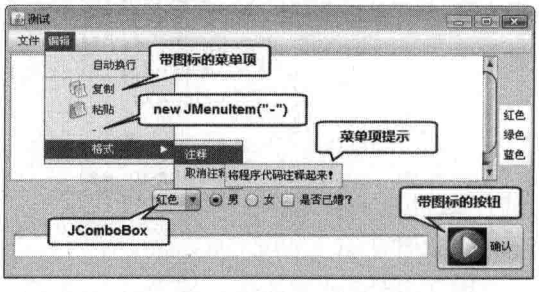


图12.3　Nimbus风格的Swing图形界面

从图12.3中可以看出，Swing菜单不允许使用add(new JMenuItem("-"))的方式来添加菜单分隔符，只能使用addSeparator()方法来添加菜单分隔符。

提示：Swing专门为菜单项、工具按钮之间的分隔符提供了一个JSeparator类，通常使用JMenu或者JPopupMenu的addSeparator()方法来创建并添加JSeparator对象，而不是直接使用JSeparator。实际上，JSeparator可以用在任何需要使用分隔符的地方。

上面程序为newItem菜单项增加了快捷键，为Swing菜单项指定快捷键与为AWT菜单项指定快捷

键的方式有所不同——创建 AWT 菜单对象时可以直接传入 KeyShortcut 对象为其指定快捷键；但为 Swing 菜单项指定快捷键时必须通过 setAccelerator(KeyStroke ks)方法来设置（如①处程序所示），其中 KeyStroke 代表一次击键动作，可以直接通过按键对应字母来指定该击键动作。

提示：

为菜单项指定快捷键时应该使用大写字母来代表按键，例如 KeyStroke.getKeyStroke ('N' , InputEvent.CTRL_MASK)代表 “Ctrl+N”，但 KeyStroke.getKeyStroke('n', InputEvent. CTRL_MASK)则不代表 “Ctrl+N”。

除此之外，上面程序中的大段粗体字代码所定义的 changeFlavor()方法用于改变程序外观风格，当用户单击多行文本域里的右键菜单时将会触发该方法，该方法设置 Swing 组件的外观风格后，再次调用 SwingUtilities 类的 updateComponentTreeUI()方法来更新指定容器，以及该容器内所有组件的 UI。注意此处更新的是 JFrame 对象 getContentPane()方法的返回值，而不是直接更新 JFrame 对象本身（如②处程序所示）。这是因为如果直接更新 JFrame 本身，将会导致 JFrame 也被更新，JFrame 是一个特殊的容器，JFrame 依然部分依赖于本地平台的图形组件。尤其是当取消③处代码的注释后，JFrame 将会使用 Java 风格的标题栏、边框，如果强制 JFrame 更新成 Windows 或 Motif 风格，则会导致该窗口失去标题栏和边框。如果通过右键菜单选择程序使用 Motif 风格，将看到如图 12.4 所示的窗口。

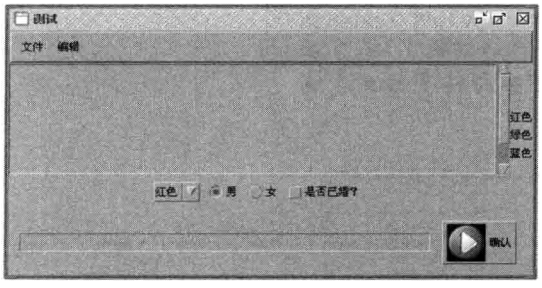


图 12.4 使用 Java 风格窗口标题、边框、Motif 显示风格的窗口

提示：

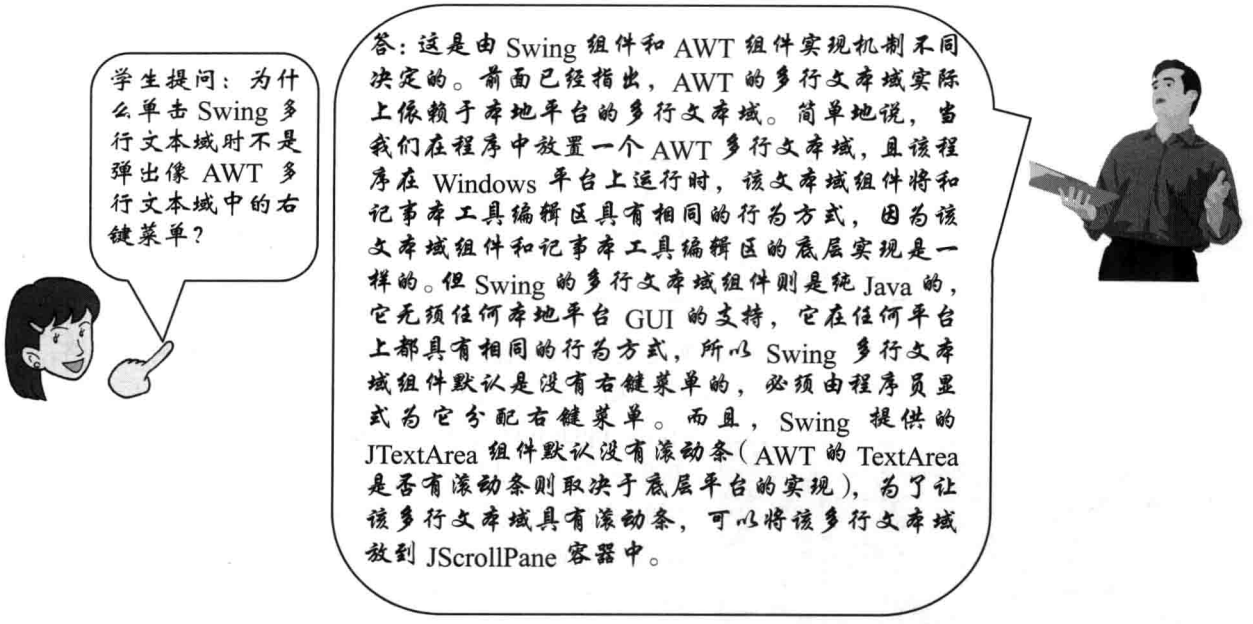
JFrame 提供了一个 getContentPane()方法，这个方法用于返回该 JFrame 的顶级容器(即 JRootPane 对象)，这个顶级容器会包含 JFrame 所显示的所有非菜单组件。可以这样理解：所有看似放在 JFrame 中的 Swing 组件，除菜单之外，其实都是放在 JFrame 对应的顶级容器中的，而 JFrame 容器里提供了 getContentPane()方法返回的顶级容器。在 Java 5 以前，Java 甚至不允许直接向 JFrame 中添加组件，必须先调用 JFrame 的 getContentPane()方法获得该窗口的顶级容器，然后将所有组件添加到该顶级容器中。从 Java 5 以后，Java 改写了 JFrame 的 add()和 setLayout()等方法，当程序调用 JFrame 的 add()和 setLayout()等方法时，实际上是对 JFrame 的顶级容器进行操作。

从程序中④处代码可以看出，为 Swing 组件添加右键菜单无须像 AWT 中那样烦琐，只需要简单地调用 setComponentPopupMenu()方法来设置右键菜单即可，无须编写事件监听器。由此可见，使用 Swing 组件编写图形界面程序更加简单。

除此之外，如果程序希望用户单击窗口右上角的 “×” 按钮时，程序退出，也无须使用事件机制，只要调用 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)方法即可，Swing 提供的这种方式也是为了简化界面编程。

JScrollPane 组件是一个特殊的组件，它不同于 JFrame、JPanel 等普通容器，它甚至不能指定自己的布局管理器，它主要用于为其他的 Swing 组件提供滚动条支持，JScrollPane 通常由普通的 Swing 组件，可选的垂直、水平滚动条以及可选的行、列标题组成。

简而言之，如果希望让 JTextArea、JTable 等组件能有滚动条支持，只要将该组件放入 JScrollPane 中，再将该 JScrollPane 容器添加到窗口中即可。关于 JScrollPane 的详细说明，读者可以参考 JScrollPane 的 API 文档。



提示：

JScrollPane对于JTable组件尤其重要，通常需要把JTable放在JScrollPane容器中才可以显示出JTable组件的标题栏。

12.2.3　为组件设置边框

可以调用JComponent提供的setBorder(Border b)方法为Swing组件设置边框，其中Border是Swing提供的一个接口，用于代表组件的边框。该接口有数量众多的实现类，如LineBorder、MatteBorder、BevelBorder等，这些Border实现类都提供了相应的构造器用于创建Border对象，一旦获取了Border对象之后，就可以调用JComponent的setBorder(Border b)方法为指定组件设置边框。

TitledBorder和CompoundBorder比较独特，其中TitledBorder的作用并不是为其他组件添加边框，而是为其他边框设置标题，当创建TitledBorder对象时，需要传入一个已经存在的Border对象，新创建的TitledBorder对象会为原有的Border对象添加标题；而CompoundBorder用于组合两个边框，因此创建CompoundBorder对象时需要传入两个Border对象，一个用作组件的内边框，一个用作组件的外边框。

除此之外，Swing还提供了一个BorderFactory静态工厂类，该类提供了大量的静态工厂方法用于返回Border实例，这些静态方法的参数与各Border实现类的构造器参数基本一致。

提示：

Border不仅提供了上面所提到的一些Border实现类，还提供了MetalBorders.oolBarBorder、MetalBorders.TextFieldBorder等Border实现类，这些实现类用作Swing组件的默认边框，程序中通常无须使用这些系统边框。

为Swing组件添加边框可按如下步骤进行。

① 使用BorderFactory或者XxxBorder创建XxxBorder实例。

② 调用Swing组件的setBorder(Border b)方法为该组件设置边框。

图12.5显示了系统可用边框之间的继承层次。

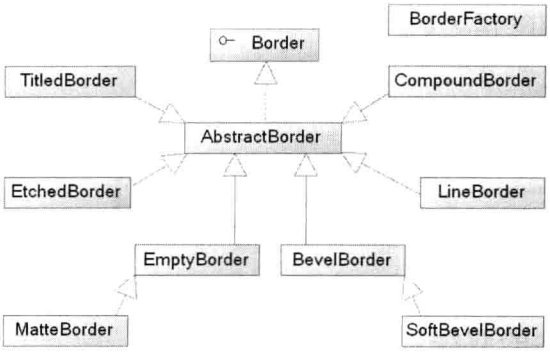


图 12.5 系统可用边框之间的继承层次

下面的例子程序示范了为 Panel 容器分别添加如图 12.5 所示的几种边框。

程序清单：codes\12\12.2\BorderTest.java

```
public class BorderTest
{
    private JFrame jf = new JFrame("测试边框");
    public void init()
    {
        jf.setLayout(new GridLayout(2, 4));
        // 使用静态工厂方法创建 BevelBorder
        Border bb = BorderFactory.createBevelBorder(
            BevelBorder.RAISED , Color.RED, Color.GREEN
            , Color.BLUE, Color.GRAY);
        jf.add(getPanelWithBorder(bb , "BevelBorder"));
        // 使用静态工厂方法创建 LineBorder
        Border lb = BorderFactory.createLineBorder(Color.ORANGE, 10);
        jf.add(getPanelWithBorder(lb , "LineBorder"));
        // 使用静态工厂方法创建 EmptyBorder，EmptyBorder 就是在组件四周留空
        Border eb = BorderFactory.createEmptyBorder(20, 5, 10, 30);
        jf.add(getPanelWithBorder(eb , "EmptyBorder"));
        // 使用静态工厂方法创建 EtchedBorder
        Border etb = BorderFactory.createEtchedBorder(EtchedBorder.RAISED,
        Color.RED, Color.GREEN);
        jf.add(getPanelWithBorder(etb , "EtchedBorder"));
        // 直接创建 TitledBorder，TitledBorder 就是为原有的边框增加标题
        TitledBorder tb = new TitledBorder(lb , "测试标题"
            , TitledBorder.LEFT , TitledBorder.BOTTOM
            , new Font("StSong" , Font.BOLD , 18), Color.BLUE);
        jf.add(getPanelWithBorder(tb , "TitledBorder"));
        // 直接创建 MatteBorder，MatteBorder 是 EmptyBorder 的子类，
        // 它可以指定留空区域的颜色或背景，此处是指定颜色
        MatteBorder mb = new MatteBorder(20, 5, 10, 30, Color.GREEN);
        jf.add(getPanelWithBorder(mb , "MatteBorder"));
        // 直接创建 CompoundBorder，CompoundBorder 将两个边框组合成新边框
        CompoundBorder cb = new CompoundBorder(new LineBorder(
            Color.RED, 8) , tb);
        jf.add(getPanelWithBorder(cb , "CompoundBorder"));
        jf.pack();
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new BorderTest().init();
    }
    public JPanel getPanelWithBorder(Border b , String BorderName)
    {
        JPanel p = new JPanel();
        p.add(new JLabel(BorderName));
        // 为 Panel 组件设置边框
        p.setBorder(b);
        return p;
    }
}
```

运行上面程序，会看到如图 12.6 所示的效果。

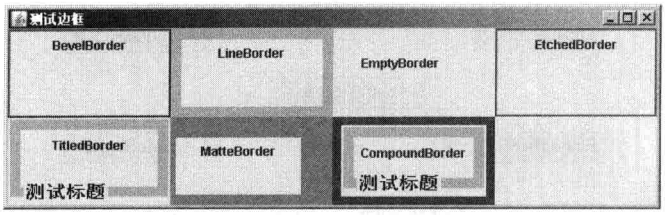


图 12.6　为 Swing 组件设置边框

12.2.4　Swing 组件的双缓冲和键盘驱动

除此之外，Swing 组件还有如下两个功能。

- 所有的 Swing 组件默认启用双缓冲绘图技术。
- 所有的 Swing 组件都提供了简单的键盘驱动。

Swing 组件默认启用双缓冲绘图技术，使用双缓冲技术能改进频繁重绘 GUI 组件的显示效果（避免闪烁现象）。JComponent 组件默认启用双缓冲，无须自己实现双缓冲。如果想关闭双缓冲，可以在组件上调用 setDoubleBuffered(false)方法。前一章介绍五子棋游戏时已经提到 Swing 组件的双缓冲技术，而且可以使用 JPanel 代替前一章所有示例程序中的 Canvas 画布组件，从而可以解决运行那些示例程序时的“闪烁”现象。

JComponent 类提供了 getInputMap()和 getActionMap()两个方法，其中 getInputMap()返回一个 InputMap 对象，该对象用于将 KeyStroke 对象（代表键盘或其他类似输入设备的一次输入事件）和名字关联；getActionMap()返回一个 ActionMap 对象，该对象用于将指定名字和 Action（Action 接口是 ActionListener 接口的子接口，可作为一个事件监听器使用）关联，从而可以允许用户通过键盘操作来替代鼠标驱动 GUI 上的 Swing 组件，相当于为 GUI 组件提供快捷键。典型用法如下：

```
// 把一次键盘事件和一个 aCommand 对象关联
component.getInputMap().put(aKeyStroke, aCommand);
// 将 aCommand 对象和 anAction 事件响应关联
component.getActionMap().put(aCommmand, anAction);
```

下面程序实现这样一个功能：用户在单行文本框内输入内容，当输入完成后，单击后面的“发送”按钮即可将文本框的内容添加到一个多行文本域中；或者输入完成后在文本框内按“Ctrl+Enter”键也可以将文本框的内容添加到一个多行文本域中。

程序清单：codes\12\12.2\BindKeyTest.java

```
public class BindKeyTest
{
    JFrame jf = new JFrame("测试键盘绑定");
    JTextArea jta = new JTextArea(5, 30);
    JButton jb = new JButton("发送");
    JTextField jtf = new JTextField(15);
    public void init()
    {
        jf.add(jta);
        JPanel jp = new JPanel();
        jp.add(jtf);
        jp.add(jb);
        jf.add(jp , BorderLayout.SOUTH);
        // 发送消息的 Action，Action 是 ActionListener 的子接口
        Action sendMsg = new AbstractAction()
        {
            public void actionPerformed(ActionEvent e)
            {
                jta.append(jtf.getText() + "\n");
                jtf.setText("");
            }
        };
        // 添加事件监听器
        jb.addActionListener(sendMsg);
```

```
        // 将Ctrl+Enter键和"send"关联
        jtf.getInputMap().put(KeyStroke.getKeyStroke('\n'
            , java.awt.event.InputEvent.CTRL_MASK) , "send");
        // 将"send"和sendMsg Action关联
        jtf.getActionMap().put("send", sendMsg);
        jf.pack();
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new BindKeyTest().init();
    }
}
```

上面程序中粗体字代码示范了如何利用键盘事件来驱动 Swing 组件，采用这种键盘事件机制，无须为 Swing 组件绑定键盘监听器，从而可以复用按钮单击事件的事件监听器，程序十分简洁。

12.2.5 使用 JToolBar 创建工具条

Swing 提供了 JToolBar 类来创建工具条，创建 JToolBar 对象时可以指定如下两个参数。

- name：该参数指定该工具条的名称。
- orientation：该参数指定该工具条的方向。

一旦创建了 JToolBar 对象之后，JToolBar 对象还有如下几个常用方法。

- JButton add(Action a)：通过 Action 对象为 JToolBar 添加对应的工具按钮。
- void addSeparator(Dimension size)：向工具条中添加指定大小的分隔符，Java 允许不指定 size 参数，则添加一个默认大小的分隔符。
- void setFloatable(boolean b)：设置该工具条是否可浮动，即该工具条是否可以拖动。
- void setMargin(Insets m)：设置工具条边框和工具按钮之间的页边距。
- void setOrientation(int o)：设置工具条的方向。
- void setRollover(boolean rollover)：设置此工具条的 rollover 状态。

上面的大多数方法都比较容易理解，比较难以理解的是 add(Action a)方法，系统如何为工具条添加 Action 对应的按钮呢？

Action 接口是 ActionListener 接口的子接口，它除了包含 ActionListener 接口的 actionPerformed()方法之外，还包含 name 和 icon 两个属性，其中 name 用于指定按钮或菜单项中的文本，而 icon 则用于指定按钮的图标或菜单项中的图标。也就是说，Action 不仅可作为事件监听器使用，而且可被转换成按钮或菜单项。

值得指出的是，Action 本身并不是按钮，也不是菜单项，只是当把 Action 对象添加到某些容器（也可直接使用 Action 来创建按钮），如菜单和工具栏中时，这些容器会为该 Action 对象创建对应的组件（菜单项和按钮）。也就是说，这些容器需要负责完成如下事情。

- 创建一个适用于该容器的组件（例如，在工具栏中创建一个工具按钮）。
- 从 Action 对象中获得对应的属性来设置该组件（例如，通过 name 来设置文本，通过 icon 来设置图标）。
- 检查 Action 对象的初始状态，确定它是否处于激活状态，并根据该 Action 的状态来决定其对应所有组件的行为。只有处于激活状态的 Action 所对应的 Swing 组件才可以响应用户动作。
- 通过 Action 对象为对应组件注册事件监听器，系统将为该 Action 所创建的所有组件注册同一个事件监听器（事件处理器就是 Action 对象里的 actionPerformed()方法）。

例如，程序中有一个菜单项、一个工具按钮，还有一个普通按钮都需要完成某个“复制”动作，程序就可以将该“复制”动作定义成 Action，并为之指定 name 和 icon 属性，然后通过该 Action 来创建菜单项、工具按钮和普通按钮，就可以让这三个组件具有相同的功能。另一个“粘贴”按钮也大致相似，而且“粘贴”组件默认不可用，只有当“复制” 组件被触发后，且剪贴板中有内容时才可用。

程序清单：codes\12\12.2\JToolBarTest.java

```
public class JToolBarTest
{
    JFrame jf = new JFrame("测试工具条");
    JTextArea jta = new JTextArea(6, 35);
```

```
JToolBar jtb = new JToolBar();
JMenuBar jmb = new JMenuBar();
JMenu edit = new JMenu("编辑");
// 获取系统剪贴板
Clipboard clipboard = Toolkit.getDefaultToolkit()
    .getSystemClipboard();
// 创建"粘贴"Action，该Action用于创建菜单项、工具按钮和普通按钮
Action pasteAction = new AbstractAction("粘贴"
    , new ImageIcon("ico/paste.png"))
{
    public void actionPerformed(ActionEvent e)
    {
        // 如果剪贴板中包含stringFlavor内容
        if (clipboard.isDataFlavorAvailable(DataFlavor.stringFlavor))
        {
            try
            {
                // 取出剪贴板中的stringFlavor内容
                String content = (String)clipboard.getData
                (DataFlavor.stringFlavor);
                // 将选中内容替换成剪贴板中的内容
                jta.replaceRange(content , jta.getSelectionStart()
                    , jta.getSelectionEnd());
            }
            catch (Exception ee)
            {
                ee.printStackTrace();
            }
        }
    }
};
// 创建"复制"Action
Action copyAction = new AbstractAction("复制"
    , new ImageIcon("ico/copy.png"))
{
    public void actionPerformed(ActionEvent e)
    {
        StringSelection contents = new StringSelection(
            jta.getSelectedText());
        // 将StringSelection对象放入剪贴板中
        clipboard.setContents(contents, null);
        // 如果剪贴板中包含stringFlavor内容
        if (clipboard.isDataFlavorAvailable(DataFlavor.stringFlavor))
        {
            // 将pasteAction激活
            pasteAction.setEnabled(true);
        }
    }
};
public void init()
{
    // pasteAction默认处于不激活状态
    pasteAction.setEnabled(false);   // ①
    jf.add(new JScrollPane(jta));
    // 以Action创建按钮，并将该按钮添加到Panel中
    JButton copyBn = new JButton(copyAction);
    JButton pasteBn = new JButton(pasteAction);
    JPanel jp = new JPanel();
    jp.add(copyBn);
    jp.add(pasteBn);
    jf.add(jp , BorderLayout.SOUTH);
    // 向工具条中添加Action对象，该对象将会转换成工具按钮
    jtb.add(copyAction);
    jtb.addSeparator();
    jtb.add(pasteAction);
    // 向菜单中添加Action对象，该对象将会转换成菜单项
    edit.add(copyAction);
    edit.add(pasteAction);
    // 将edit菜单添加到菜单条中
    jmb.add(edit);
```

```
        jf.setJMenuBar(jmb);
        // 设置工具条和工具按钮之间的页边距。
        jtb.setMargin(new Insets(20 ,10 , 5 , 30));    // ②
        // 向窗口中添加工具条
        jf.add(jtb , BorderLayout.NORTH);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.pack();
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new JToolBarTest().init();
    }
}
```

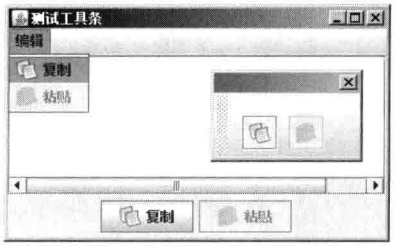


图 12.7 使用 Action 创建按钮、工具按钮和菜单项

上面程序中创建了 pasteAction、copyAction 两个 Action，然后根据这两个 Action 分别创建了按钮、工具按钮、菜单项组件（程序中粗体字代码部分），开始时 pasteAction 处于非激活状态，则该 Action 对应的按钮、工具按钮、菜单项都处于不可用状态。运行上面程序，会看到如图 12.7 所示的界面。

图 12.7 显示了工具条被拖动后的效果，这是因为工具条默认处于浮动状态。除此之外，程序中②号粗体字代码设置了工具条和工具按钮之间的页边距，所以可以看到工具条在工具按钮周围保留了一些空白区域。

12.2.6 使用 JFileChooser 和 Java 7 增强的 JColorChooser

JColorChooser 用于创建颜色选择器对话框，该类的用法非常简单，该类主要提供了如下两个静态方法。

- showDialog(Component component, String title, Color initialColor)：显示一个模式的颜色选择器对话框，该方法返回用户所选颜色。其中 component 指定该对话框的 parent 组件，而 title 指定该对话框的标题，大部分时候都使用该方法来让用户选择颜色。
- createDialog(Component c, String title, boolean modal, JColorChooser chooserPane, ActionListener okListener, ActionListener cancelListener)：该方法返回一个对话框，该对话框内包含指定的颜色选择器，该方法可以指定该对话框是模式的还是非模式的（通过 modal 参数指定），还可以指定该对话框内“确定”按钮的事件监听器（通过 okListener 参数指定）和“取消”按钮的事件监听器（通过 cancelListener 参数指定）。

Java 7 为 JColorChooser 增加了一个 HSV 标签页，允许用户通过 HSV 模式来选择颜色。

下面程序改写了前一章的 HandDraw 程序，改为使用 JPanel 作为绘图组件，而且使用 JColorChooser 来弹出颜色选择器对话框。

程序清单：codes\12\12.2\HandDraw.java

```
public class HandDraw
{
    // 画图区的宽度
    private final int AREA_WIDTH = 500;
    // 画图区的高度
    private final int AREA_HEIGHT = 400;
    // 下面的 preX、preY 保存了上一次鼠标拖动事件的鼠标坐标
    private int preX = -1;
    private int preY = -1;
    // 定义一个右键菜单用于设置画笔颜色
    JPopupMenu pop = new JPopupMenu();
    JMenuItem chooseColor = new JMenuItem("选择颜色");
    // 定义一个 BufferedImage 对象
    BufferedImage image = new BufferedImage(AREA_WIDTH
        , AREA_HEIGHT , BufferedImage.TYPE_INT_RGB);
```

```
        // 获取 image 对象的 Graphics
        Graphics g = image.getGraphics();
        private JFrame f = new JFrame("简单手绘程序");
        private DrawCanvas drawArea = new DrawCanvas();
        // 用于保存画笔颜色
        private Color foreColor = new Color(255, 0 ,0);
        public void init()
        {
            chooseColor.addActionListener(ae) -> {
                // 下面代码直接弹出一个模式的颜色选择对话框，并返回用户选择的颜色
                // foreColor = JColorChooser.showDialog(f
                //     , "选择画笔颜色" , foreColor);    // ①
                // 下面代码则弹出一个非模式的颜色选择对话框
                // 并可以分别为“确定”按钮、“取消”按钮指定事件监听器
                final JColorChooser colorPane = new JColorChooser(foreColor);
                JDialog jd = JColorChooser.createDialog(f , "选择画笔颜色"
                    , false, colorPane, e->foreColor = colorPane.getColor(), null);
                jd.setVisible(true);
            });
            // 将菜单项组合成右键菜单
            pop.add(chooseColor);
            // 将右键菜单添加到 drawArea 对象中
            drawArea.setComponentPopupMenu(pop);
            // 将 image 对象的背景色填充成白色
            g.fillRect(0 , 0 ,AREA_WIDTH , AREA_HEIGHT);
            drawArea.setPreferredSize(new Dimension(AREA_WIDTH , AREA_HEIGHT));
            // 监听鼠标移动动作
            drawArea.addMouseMotionListener(new MouseMotionAdapter()
            {
                // 实现按下鼠标键并拖动的事件处理器
                public void mouseDragged(MouseEvent e)
                {
                    // 如果 preX 和 preY 大于 0
                    if (preX > 0 && preY > 0)
                    {
                        // 设置当前颜色
                        g.setColor(foreColor);
                        // 绘制从上一次鼠标拖动事件点到本次鼠标拖动事件点的线段
                        g.drawLine(preX , preY , e.getX() , e.getY());
                    }
                    // 将当前鼠标事件点的 X、Y 坐标保存起来
                    preX = e.getX();
                    preY = e.getY();
                    // 重绘 drawArea 对象
                    drawArea.repaint();
                }
            });
            // 监听鼠标事件
            drawArea.addMouseListener(new MouseAdapter()
            {
                // 实现鼠标松开的事件处理器
                public void mouseReleased(MouseEvent e)
                {
                    // 松开鼠标键时，把上一次鼠标拖动事件的 X、Y 坐标设为-1
                    preX = -1;
                    preY = -1;
                }
            });
            f.add(drawArea);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.pack();
            f.setVisible(true);
        }
        public static void main(String[] args)
        {
            new HandDraw().init();
        }
        // 让画图区域继承 JPanel 类
        class DrawCanvas extends JPanel
        {
```

```
        // 重写 JPanel 的 paint 方法，实现绘画
        public void paint(Graphics g)
        {
            // 将 image 绘制到该组件上
            g.drawImage(image , 0 , 0 , null);
        }
    }
}
```

上面程序分别使用了两种方式来弹出颜色选择对话框，其中①号粗体字代码可弹出一个模式的颜色选择对话框，并直接返回用户选择的颜色。这种方式简单明了，编程简单。

如果程序有更多额外的需要，则使用程序下面的粗体字代码，弹出一个非模式的颜色选择对话框（允许程序设定），并为“确定”按钮指定了事件监听器，而“取消”按钮的事件监听器为 null（也可以为该按钮指定事件监听器）。Swing 的颜色选择对话框如图 12.8 所示。

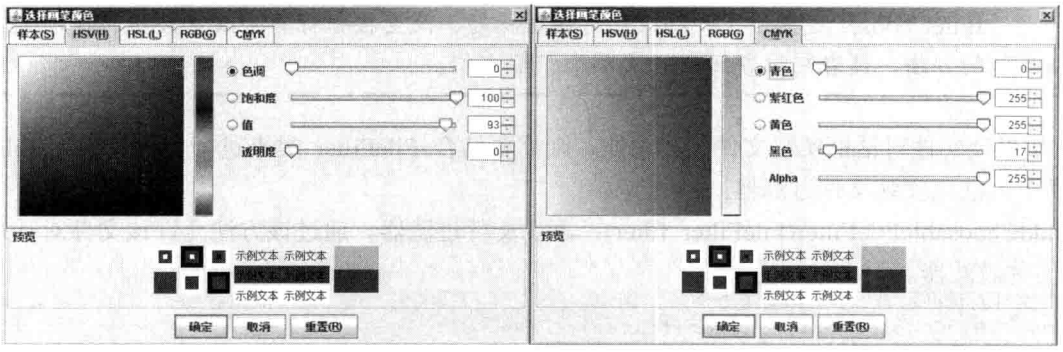


图 12.8 Swing 的颜色选择对话框

从图 12.8 中可以看出，Swing 的颜色选择对话框提供了 5 种方式来选择颜色，图中显示了 HSV 方式、CMYK 方式的颜色选择器，除此之外，该颜色选择器还可以使用 RGB、HSL 方式来选择颜色。

提示：

学习过本书第 1 版的读者应该知道，在 Java 6 时，JcolorChooser 只提供了三种颜色选择方式，图 12.8 中看到的 HSV、CMYK 两种颜色选择方式都是新增的。

JFileChooser 的功能与 AWT 中的 FileDialog 基本相似，也是用于生成“打开文件”、“保存文件”对话框；与 FileDialog 不同的是，JFileChooser 无须依赖于本地平台的 GUI，它由 100%纯 Java 实现，在所有平台上具有完全相同的行为，并可以在所有平台上具有相同的外观风格。

为了调用 JFileChooser 来打开一个文件对话框，必须先创建该对话框的实例，JFileChooser 提供了多个构造器来创建 JFileChooser 对象，它的构造器总共包含两个参数。

- currentDirectory：指定所创建文件对话框的当前路径，该参数既可以是一个 String 类型的路径，也可以是一个 File 对象所代表的路径。
- FileSystemView：用于指定基于该文件系统外观来创建文件对话框，如果没有指定该参数，则默认以当前文件系统外观创建文件对话框。

JFileChooser 并不是 JDialog 的子类，所以不能使用 setVisible(true)方法来显示该文件对话框，而是调用 showXxxDialog()方法来显示文件对话框。

使用 JFileChooser 来建立文件对话框并允许用户选择文件的步骤如下。

① 采用构造器创建一个 JFileChooser 对象，该 JFileChooser 对象无须指定 parent 组件，这意味着可以在多个窗口中共用该 JFileChooser 对象。创建 JFileChooser 对象时可以指定初始化路径，如下代码所示。

```
// 以当前路径创建文件选择器
JFileChooser chooser = new JFileChooser(".");
```

② 调用 JFileChooser 的一系列可选的方法对 JFileChooser 执行初始化操作。JFileChooser 大致有如

下几个常用方法。

- setSelectedFile/setSelectedFiles：指定该文件选择器默认选择的文件（也可以默认选择多个文件）。

```
// 默认选择当前路径下的123.jpg文件
chooser.setSelectedFile(new File("123.jpg"));
```

- setMultiSelectionEnabled(boolean b)：在默认情况下，该文件选择器只能选择一个文件，通过调用该方法可以设置允许选择多个文件（设置参数值为true即可）。
- setFileSelectionMode(int mode)：在默认情况下，该文件选择器只能选择文件，通过调用该方法可以设置允许选择文件、路径、文件与路径，设置参数值为：JFileChooser. FILES_ONLY、JFileChooser.DIRECTORIES_ONLY、JFileChooser.FILES_AND_DIRECTORIES。

```
// 设置既可选择文件，也可选择路径
chooser.setFileSelectionMode (JFileChooser.FILES_AND_DIRECTORIES);
```

提示：

JFileChooser还提供了一些改变对话框标题、改变按钮标签、改变按钮的提示文本等功能的方法，读者应该查阅API文档来了解它们。

3 如果让文件对话框实现文件过滤功能，则需要结合FileFilter类来进行文件过滤。JFileChooser提供了两个方法来安装文件过滤器。

- addChoosableFileFilter(FileFilter filter)：添加文件过滤器。通过该方法允许该文件对话框有多个文件过滤器。

```
// 为文件对话框添加一个文件过滤器
chooser.addChoosableFileFilter(filter);
```

- setFileFilter(FileFilter filter)：设置文件过滤器。一旦调用了该方法，将导致该文件对话框只有一个文件过滤器。

4 如果需要改变文件对话框中文件的视图外观，则可以结合FileView类来改变对话框中文件的视图外观。

5 调用showXxxDialog方法可以打开文件对话框，通常如下三个方法可用。

- int showDialog(Component parent, String approveButtonText)：弹出文件对话框，该对话框的标题、“同意”按钮的文本（默认是“保存”或“取消”按钮）由approveButtonText来指定。
- int showOpenDialog(Component parent)：弹出文件对话框，该对话框具有默认标题，“同意”按钮的文本是“打开”。
- int showSaveDialog(Component parent)：弹出文件对话框，该对话框具有默认标题，“同意”按钮的文本是“保存”。

当用户单击“同意”、“取消”按钮，或者直接关闭文件对话框时才可以关闭该文件对话框，关闭该对话框时返回一个int类型的值，分别是：JFileChooser.APPROVE_OPTION、JFileChooser.CANCEL_OPTION和JFileChooser. ERROR_OPTION。如果希望获得用户选择的文件，则通常应该先判断对话框的返回值是否为JFileChooser.APPROVE_OPTION，该选项表明用户单击了“打开”或者“保存”按钮。

6 JFileChooser提供了如下两个方法来获取用户选择的文件或文件集。

- File getSelectedFile()：返回用户选择的文件。
- File[] getSelectedFiles()：返回用户选择的多个文件。

按上面的步骤，就可以正常地创建一个“打开文件”、“保存文件”对话框，整个过程非常简单。如果要使用FileFilter类来进行文件过滤，或者使用FileView类来改变文件的视图风格，则有一点麻烦。

先看使用FileFilter类来进行文件过滤。Java在java.io包下提供了一个FileFilter接口，该接口主要用于作为File类的listFiles(FileFilter)方法的参数，也是一个进行文件过滤的接口。但此处需要使用位于javax.swing.filechooser包下的FileFilter抽象类，该抽象类包含两个抽象方法。

- boolean accept(File f)：判断该过滤器是否接受给定的文件，只有被该过滤器接受的文件才可以

在对应的文件对话框中显示出来。

- String getDescription()：返回该过滤器的描述性文本。

如果程序要使用 FileFilter 类进行文件过滤，则通常需要扩展该 FileFilter 类，并重写该类的两个抽象方法，重写 accept()方法时就可以指定自己的业务规则，指定该文件过滤器可以接受哪些文件。例如，如下代码：

```
public boolean accept(File f)
{
    // 如果该文件是路径，则接受该文件
    if (f.isDirectory()) return true;
    // 只接受以.gif 作为后缀的文件
    if (name.endsWith(".gif"))
    {
        return true;
    }
    return false
}
```

在默认情况下，JFileChooser 总会在文件对话框的“文件类型”下拉列表中增加“所有文件”选项，但可以调用 JFileChooser 的 setAcceptAllFileFilterUsed(false)来取消显示该选项。

FileView 类用于改变文件对话框中文件的视图风格，FileView 类也是一个抽象类，通常程序需要扩展该抽象类，并有选择性地重写它所包含的如下几个抽象方法。

- String getDescription(File f)：返回指定文件的描述。
- Icon getIcon(File f)：返回指定文件在 JFileChooser 对话框中的图标。
- String getName(File f)：返回指定文件的文件名。
- String getTypeDescription(File f)：返回指定文件所属文件类型的描述。
- Boolean isTraversable(File f)：当该文件是目录时，返回该目录是否是可遍历的。

与重写 FileFilter 抽象方法类似的是，重写这些方法实际上就是为文件选择器对话框指定自定义的外观风格。通常可以通过重写 getIcon()方法来改变文件对话框中的文件图标。

下面程序是一个简单的图片查看工具程序，该程序综合使用了上面所介绍的各知识点。

程序清单：codes\12\12.2\ImageViewer.java

```
public class ImageViewer
{
    // 定义图片预览组件的大小
    final int PREVIEW_SIZE = 100;
    JFrame jf = new JFrame("简单图片查看器");
    JMenuBar menuBar = new JMenuBar();
    // 该 label 用于显示图片
    JLabel label = new JLabel();
    // 以当前路径创建文件选择器
    JFileChooser chooser = new JFileChooser(".");
    JLabel accessory = new JLabel();
    // 定义文件过滤器
    ExtensionFileFilter filter = new ExtensionFileFilter();
    public void init()
    {
        // --------下面开始初始化 JFileChooser 的相关属性--------
        //  创建一个 FileFilter
        filter.addExtension("jpg");
        filter.addExtension("jpeg");
        filter.addExtension("gif");
        filter.addExtension("png");
        filter.setDescription("图片文件(*.jpg,*.jpeg,*.gif,*.png)");
        chooser.addChoosableFileFilter(filter);
        // 禁止“文件类型”下拉列表中显示“所有文件”选项
        chooser.setAcceptAllFileFilterUsed(false);    // ①
        // 为文件选择器指定自定义的 FileView 对象
        chooser.setFileView(new FileIconView(filter));
        // 为文件选择器指定一个预览图片的附件
        chooser.setAccessory(accessory);     // ②
```

```
        // 设置预览图片组件的大小和边框
        accessory.setPreferredSize(new Dimension(PREVIEW_SIZE, PREVIEW_SIZE));
        accessory.setBorder(BorderFactory.createEtchedBorder());
        // 用于检测被选择文件的改变事件
        chooser.addPropertyChangeListener(event -> {
            // JFileChooser 的被选文件已经发生了改变
            if (event.getPropertyName() ==
                JFileChooser.SELECTED_FILE_CHANGED_PROPERTY)
            {
                // 获取用户选择的新文件
                File f = (File) event.getNewValue();
                if (f == null)
                {
                    accessory.setIcon(null);
                    return;
                }
                // 将所选文件读入 ImageIcon 对象中
                ImageIcon icon = new ImageIcon(f.getPath());
                // 如果图像太大，则缩小它
                if(icon.getIconWidth() > PREVIEW_SIZE)
                {
                    icon = new ImageIcon(icon.getImage().getScaledInstance
                        (PREVIEW_SIZE, -1, Image.SCALE_DEFAULT));
                }
                // 改变 accessory Label 的图标
                accessory.setIcon(icon);
            }
        });
        // ------下面代码开始为该窗口安装菜单------
        JMenu menu = new JMenu("文件");
        menuBar.add(menu);
        JMenuItem openItem = new JMenuItem("打开");
        menu.add(openItem);
        // 单击 openItem 菜单项显示“打开文件”对话框
        openItem.addActionListener(event -> {
            // 设置文件对话框的当前路径
            // chooser.setCurrentDirectory(new File("."));
            // 显示文件对话框
            int result = chooser.showDialog(jf , "打开图片文件");
            // 如果用户选择了 APPROVE（同意）按钮，即打开，保存的等效按钮
            if(result == JFileChooser.APPROVE_OPTION)
            {
                String name = chooser.getSelectedFile().getPath();
                // 显示指定图片
                label.setIcon(new ImageIcon(name));
            }
        });
        JMenuItem exitItem = new JMenuItem("Exit");
        menu.add(exitItem);
        // 为退出菜单绑定事件监听器
        exitItem.addActionListener(event -> System.exit(0));
        jf.setJMenuBar(menuBar);
        // 添加用于显示图片的 JLabel 组件
        jf.add(new JScrollPane(label));
        jf.pack();
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new ImageViewer().init();
    }
}
// 创建 FileFilter 的子类，用以实现文件过滤功能
class ExtensionFileFilter extends FileFilter
{
    private String description;
    private ArrayList<String> extensions = new ArrayList<>();
    // 自定义方法，用于添加文件扩展名
    public void addExtension(String extension)
    {
```

```
            if (!extension.startsWith("."))
            {
                extension = "." + extension;
                extensions.add(extension.toLowerCase());
            }
        }
        // 用于设置该文件过滤器的描述文本
        public void setDescription(String aDescription)
        {
            description = aDescription;
        }
        // 继承 FileFilter 类必须实现的抽象方法，返回该文件过滤器的描述文本
        public String getDescription()
        {
            return description;
        }
        // 继承 FileFilter 类必须实现的抽象方法，判断该文件过滤器是否接受该文件
        public boolean accept(File f)
        {
            // 如果该文件是路径，则接受该文件
            if (f.isDirectory()) return true;
            // 将文件名转为小写（全部转为小写后比较，用于忽略文件名大小写）
            String name = f.getName().toLowerCase();
            // 遍历所有可接受的扩展名，如果扩展名相同，该文件就可接受
            for (String extension : extensions)
            {
                if (name.endsWith(extension))
                {
                    return true;
                }
            }
            return false;
        }
    }
    // 自定义一个 FileView 类，用于为指定类型的文件或文件夹设置图标
    class FileIconView extends FileView
    {
        private FileFilter filter;
        public FileIconView(FileFilter filter)
        {
            this.filter = filter;
        }
        // 重写该方法，为文件夹、文件设置图标
        public Icon getIcon(File f)
        {
            if (!f.isDirectory() && filter.accept(f))
            {
                return new ImageIcon("ico/pict.png");
            }
            else if (f.isDirectory())
            {
                // 获取所有根路径
                File[] fList = File.listRoots();
                for (File tmp : fList)
                {
                    // 如果该路径是根路径
                    if (tmp.equals(f))
                    {
                        return  new ImageIcon("ico/dsk.png");
                    }
                }
                return new ImageIcon("ico/folder.png");
            }
            // 使用默认图标
            else
            {
                return null;
            }
        }
    }
```

上面程序中第二段粗体字代码用于为“打开”菜单项指定事件监听器，当用户单击该菜单时，程序打开文件对话框，并将用户打开的图片文件使用 Label 在当前窗口显示出来。

第三段粗体字代码用于重写 FileFilter 类的 accept()方法，该方法根据文件后缀来决定是否接受该文件，其要求是当该文件的后缀等于该文件过滤器的 extensions 集合的某一项元素时，则该文件是可接受的。程序的①处代码禁用了 JFileChooser 中“所有文件”选项，从而让用户只能看到图片文件。

第四段粗体字代码用于重写 FileView 类的 getIcon()方法，该方法决定 JFileChooser 对话框中文件、文件夹的图标——图标文件就返回 pict.png 图标，根文件夹就返回 dsk.png 图标，而普通文件夹则返回 folder.png 图标。

运行上面程序，单击“打开”菜单项，将看到如图 12.9 所示的对话框。

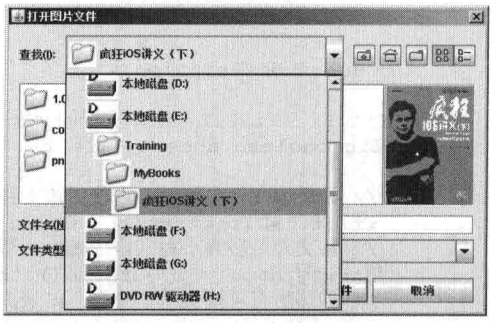


图 12.9　文件对话框

上面程序中的②处粗体字代码还用了 JFileChooser 类的 setAccessory(JComponent newAccessory)方法为该文件对话框指定附件，附件将会被显示在文件对话框的右上角，如图 12.9 所示。该附件可以是任何 Swing 组件（甚至可以使用容器），本程序中使用一个 JLabel 组件作为该附件组件，该 JLabel 用于显示用户所选图片文件的预览图片。该功能的实现很简单——当用户选择的图片发生改变时，以用户所选文件创建 ImageIcon，并将该 ImageIcon 设置成该 Label 的图标即可。

为了实现当用户选择图片发生改变时，附件组件的 icon 随之发生改变的功能，必须为 JFileChooser 添加事件监听器，该事件监听器负责监听该对话框中用户所选择文件的变化。JComponent 类中提供了一个 addPropertyChangeListener 方法，该方法可以为该 JFileChooser 添加一个属性监听器，用于监听用户选择文件的变化。程序中第一段粗体字代码实现了用户选择文件发生改变时的事件处理器。

12.2.7　使用 JOptionPane

通过 JOptionPane 可以非常方便地创建一些简单的对话框，Swing 已经为这些对话框添加了相应的组件，无须程序员手动添加组件。JOptionPane 提供了如下 4 个方法来创建对话框。

- showMessageDialog/showInternalMessageDialog：消息对话框，告知用户某事已发生，用户只能单击“确定”按钮，类似于 JavaScript 的 alert 函数。
- showConfirmDialog/showInternalConfirmDialog：确认对话框，向用户确认某个问题，用户可以选择 yes、no、cancel 等选项。类似于 JavaScript 的 comfirm 函数。该方法返回用户单击了哪个按钮。
- showInputDialog/showInternalInputDialog：输入对话框，提示要求输入某些信息，类似于 JavaScript 的 prompt 函数。该方法返回用户输入的字符串。
- showOptionDialog/showInternalOptionDialog：自定义选项对话框，允许使用自定义选项，可以取代 showConfirmDialog 所产生的对话框，只是用起来更复杂。

JOptionPane 产生的所有对话框都是模式的，在用户完成与对话框的交互之前，showXxxDialog 方法都将一直阻塞当前线程。

JOptionPane 所产生的对话框总是具有如图 12.10 所示的布局。

上面这些方法都提供了相应的 showInternalXxxDialog 版本，这种方法以 InternalFrame 的方式打开对话框。关于什么是 InternalFrame 方式，请参考下一节关于 JInternalFrame 的介绍。

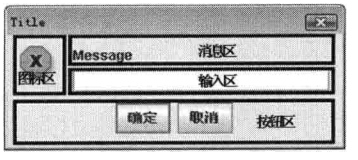


图 12.10　JoptionPane 产生的对话框的布局

下面就图 12.10 中所示的 4 个区域分别进行介绍。

（1）输入区

如果创建的对话框无须接收用户输入，则输入区不存

在。输入区组件可以是普通文本框组件，也可以是下拉列表框组件。

如果调用上面的 showInternalXxxDialog()方法时指定了一个数组类型的 selectionValues 参数，则输入区包含一个下拉列表框组件。

（2）图标区

左上角的图标会随创建的对话框所包含消息类型的不同而不同，JOptionPane 可以提供如下 5 种消息类型。

- ERROR_MESSAGE：错误消息，其图标是一个红色的 X 图标，如图 12.10 所示。
- INFORMATION_MESSAGE：普通消息，其默认图标是蓝色的感叹号。
- WARNING_MESSAGE：警告消息，其默认图标是黄色感叹号。
- QUESTION_MESSAGE：问题消息，其默认图标是绿色问号。
- PLAIN_MESSAGE：普通消息，没有默认图标。

实际上，JoptionPane 的所有 showXxxDialog()方法都可以提供一个可选的 icon 参数，用于指定该对话框的图标。

提示： 调用 showXxxDialog 方法时还可以指定一个可选的 title 参数，该参数指定所创建对话框的标题。

（3）消息区

不管是哪种对话框，其消息区总是存在的，消息区的内容通过 message 参数来指定，根据 message 参数的类型不同，消息区显示的内容也是不同的。该 message 参数可以是如下几种类型。

- String 类型：系统将该字符串对象包装成 JLabel 对象，然后显示在对话框中。
- Icon：该 Icon 被包装成 JLabel 后作为对话框的消息。
- Component：将该 Component 在对话框的消息区中显示出来。
- Object[]：对象数组被解释为在纵向排列的一系列 message 对象，每个 message 对象根据其实际类型又可以是字符串、图标、组件、对象数组等。
- 其他类型：系统调用该对象的 toString()方法返回一个字符串，并将该字符串对象包装成 JLabel 对象，然后显示在对话框中。

大部分时候对话框的消息区都是普通字符串，但使用 Component 作为消息区组件则更加灵活，因为该 Component 参数几乎可以是任何对象，从而可以让对话框的消息区包含任何内容。

提示： 如果用户希望消息区的普通字符串能换行，则可以使用“\n”字符来实现换行。

（4）按钮区

对话框底部的按钮区也是一定存在的，但所包含的按钮则会随对话框的类型、选项类型而改变。对于调用 showInputDialog()和 showMessageDialog()方法得到的对话框，底部总是包含“确定”和“取消”两个标准按钮。

对于 showConfirmDialog()所打开的确认对话框，则可以指定一个整数类型的 optionType 参数，该参数可以取如下几个值。

- DEFAULT_OPTION：按钮区只包含一个“确定”按钮。
- YES_NO_OPTION：按钮区包含“是”、“否”两个按钮。
- YES_NO_CANCEL_OPTION：按钮区包含“是”、“否”、“取消”三个按钮。
- OK_CANCEL_OPTION：按钮区包含“确定”、“取消”两个按钮。

如果使用 showOptionDialog 方法来创建选项对话框，则可以通过指定一个 Object[]类型的 options 参数来设置按钮区能使用的选项按钮。与前面的 message 参数类似的是，options 数组的数组元素可以是

如下几种类型。

- String 类型：使用该字符串来创建一个 JButton，并将其显示在按钮区。
- Icon：使用该 Icon 来创建一个 JButton，并将其显示在按钮区。
- Component：直接将该组件显示在按钮区。
- 其他类型：系统调用该对象的 toString()方法返回一个字符串，并使用该字符串来创建一个 JButton，并将其显示在按钮区。

当用户与对话框交互结束后，不同类型对话框的返回值如下。

- showMessageDialog：无返回值。
- showInputDialog：返回用户输入或选择的字符串。
- showConfirmDialog：返回一个整数代表用户选择的选项。
- showOptionDialog：返回一个整数代表用户选择的选项，如果用户选择第一项，则返回 0；如果选择第二项，则返回 1……依此类推。

对 showConfirmDialog 所产生的对话框，有如下几个返回值。

- YES_OPTION：用户单击了“是”按钮后返回。
- NO_OPTION：用户单击了“否”按钮后返回。
- CANCEL_OPTION：用户单击了“取消”按钮后返回。
- OK_OPTION：用户单击了“确定”按钮后返回。
- CLOSED_OPTION：用户单击了对话框右上角的“×”按钮后返回。

提示：

对于 showOptionDialog 方法所产生的对话框，也可能返回一个 CLOSED_OPTION 值，当用户单击了对话框右上角的“×”按钮后将返回该值。

下面程序允许使用 JOptionPane 来弹出各种对话框。

程序清单：codes\12\12.2\JOptionPaneTest.java

```
public class JOptionPaneTest
{
    JFrame jf = new JFrame("测试 JOptionPane");
    // 定义 6 个面板，分别用于定义对话框的几种选项
    private ButtonPanel messagePanel;
    private ButtonPanel messageTypePanel;
    private ButtonPanel msgPanel;
    private ButtonPanel confirmPanel;
    private ButtonPanel optionsPanel;
    private ButtonPanel inputPanel;
    private String messageString = "消息区内容";
    private Icon messageIcon = new ImageIcon("ico/heart.png");
    private Object messageObject = new Date();
    private Component messageComponent = new JButton("组件消息");
    private JButton msgBn = new JButton("消息对话框");
    private JButton confrimBn = new JButton("确认对话框");
    private JButton inputBn = new JButton("输入对话框");
    private JButton optionBn = new JButton("选项对话框");
    public void init()
    {
        JPanel top = new JPanel();
        top.setBorder(new TitledBorder(new EtchedBorder()
            , "对话框的通用选项" , TitledBorder.CENTER ,TitledBorder.TOP));
        top.setLayout(new GridLayout(1 , 2));
        // 消息类型 Panel，该 Panel 中的选项决定对话框的图标
        messageTypePanel = new ButtonPanel("选择消息的类型",
            new String[]{"ERROR_MESSAGE", "INFORMATION_MESSAGE"
                , "WARNING_MESSAGE", "QUESTION_MESSAGE",    "PLAIN_MESSAGE" });
        // 消息内容类型 Panel，该 Panel 中的选项决定对话框消息区的内容
        messagePanel = new ButtonPanel("选择消息内容的类型",
        new String[]{"字符串消息", "图标消息", "组件消息"
```

```
        , "普通对象消息" , "Object[]消息"});
    top.add(messageTypePanel);
    top.add(messagePanel);
    JPanel bottom = new JPanel();
    bottom.setBorder(new TitledBorder(new EtchedBorder()
        , "弹出不同的对话框" , TitledBorder.CENTER ,TitledBorder.TOP));
    bottom.setLayout(new GridLayout(1 , 4));
    // 创建用于弹出消息对话框的 Panel
    msgPanel = new ButtonPanel("消息对话框", null);
    msgBn.addActionListener(new ShowAction());
    msgPanel.add(msgBn);
    // 创建用于弹出确认对话框的 Panel
    confirmPanel = new ButtonPanel("确认对话框",
        new String[]{"DEFAULT_OPTION", "YES_NO_OPTION"
            , "YES_NO_CANCEL_OPTION","OK_CANCEL_OPTION"});
    confrimBn.addActionListener(new ShowAction());
    confirmPanel.add(confrimBn);
    // 创建用于弹出输入对话框的 Panel
    inputPanel = new ButtonPanel("输入对话框"
        , new String[]{"单行文本框","下拉列表选择框"});
    inputBn.addActionListener(new ShowAction());
    inputPanel.add(inputBn);
    // 创建用于弹出选项对话框的 Panel
    optionsPanel = new ButtonPanel("选项对话框"
        , new String[]{"字符串选项", "图标选项", "对象选项"});
    optionBn.addActionListener(new ShowAction());
    optionsPanel.add(optionBn);
    bottom.add(msgPanel);
    bottom.add(confirmPanel);
    bottom.add(inputPanel);
    bottom.add(optionsPanel);
    Box box = new Box(BoxLayout.Y_AXIS);
    box.add(top);
    box.add(bottom);
    jf.add(box);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.pack();
    jf.setVisible(true);
}
// 根据用户选择返回选项类型
private int getOptionType()
{
    switch(confirmPanel.getSelection())
    {
        case "DEFAULT_OPTION":
            return JOptionPane.DEFAULT_OPTION;
        case "YES_NO_OPTION":
            return JOptionPane.YES_NO_OPTION;
        case "YES_NO_CANCEL_OPTION":
            return JOptionPane.YES_NO_CANCEL_OPTION;
        default:
            return JOptionPane.OK_CANCEL_OPTION;
    }
}
// 根据用户选择返回消息
private Object getMessage()
{
    switch(messagePanel.getSelection())
    {
        case "字符串消息":
            return messageString;
        case "图标消息":
            return messageIcon;
        case "组件消息":
            return messageComponent;
        case "普通对象消息":
            return messageObject;
        default:
            return new Object[]{messageString , messageIcon
                , messageObject , messageComponent};
```

```
        }
    }
    // 根据用户选择返回消息类型（决定图标区的图标）
    private int getDialogType()
    {
        switch(messageTypePanel.getSelection())
        {
            case "ERROR_MESSAGE":
                return JOptionPane.ERROR_MESSAGE;
            case "INFORMATION_MESSAGE":
                return JOptionPane.INFORMATION_MESSAGE;
            case "WARNING_MESSAGE":
                return JOptionPane.WARNING_MESSAGE;
            case "QUESTION_MESSAGE":
                return JOptionPane.QUESTION_MESSAGE;
            default:
                return JOptionPane.PLAIN_MESSAGE;
        }
    }
    private Object[] getOptions()
    {
        switch(optionsPanel.getSelection())
        {
            case "字符串选项":
                return new String[]{"a" , "b" , "c" , "d"};
            case "图标选项":
                return new Icon[]{new ImageIcon("ico/1.gif")
                    , new ImageIcon("ico/2.gif")
                    , new ImageIcon("ico/3.gif")
                    , new ImageIcon("ico/4.gif")};
            default:
                return new Object[]{new Date() ,new Date() , new Date()};
        }
    }
    // 为各按钮定义事件监听器
    private class ShowAction implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {
            switch(event.getActionCommand())
            {
                case "确认对话框":
                    JOptionPane.showConfirmDialog(jf , getMessage()
                        ,"确认对话框", getOptionType(), getDialogType());
                    break;
                case "输入对话框":
                    if (inputPanel.getSelection().equals("单行文本框"))
                    {
                        JOptionPane.showInputDialog(jf, getMessage()
                            , "输入对话框", getDialogType());
                    }
                    else
                    {
                        JOptionPane.showInputDialog(jf, getMessage()
                            , "输入对话框", getDialogType() , null
                            , new String[]{"轻量级 Java EE 企业应用实战"
                            , "疯狂 Java 讲义"}, "疯狂 Java 讲义");
                    }
                    break;
                case "消息对话框":
                    JOptionPane.showMessageDialog(jf,getMessage()
                        ,"消息对话框", getDialogType());
                    break;
                case "选项对话框":
                    JOptionPane.showOptionDialog(jf , getMessage()
                        , "选项对话框", getOptionType() , getDialogType()
                        , null,getOptions(), "a");
                    break;
            }
        }
```

```
    }
    public static void main(String[] args)
    {
        new JOptionPaneTest().init();
    }
}
// 定义一个 JPanel 类扩展类，该类的对象包含多个纵向排列的
// JRadioButton 控件，且 Panel 扩展类可以指定一个字符串作为 TitledBorder
class ButtonPanel extends JPanel
{
    private ButtonGroup group;
    public ButtonPanel(String title, String[] options)
    {
        setBorder(BorderFactory.createTitledBorder(BorderFactory
            .createEtchedBorder(), title));
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        group = new ButtonGroup();
        for (int i = 0; options!= null && i < options.length; i++)
        {
            JRadioButton b = new JRadioButton(options[i]);
            b.setActionCommand(options[i]);
            add(b);
            group.add(b);
            b.setSelected(i == 0);
        }
    }
    // 定义一个方法，用于返回用户选择的选项
    public String getSelection()
    {
        return group.getSelection().getActionCommand();
    }
}
```

运行上面程序，会看到如图 12.11 所示的窗口。

图 12.11 已经非常清楚地显示了 JOptionPane 所支持的 4 种对话框，以及所有对话框的通用选项、每个对话框的特定选项。如果用户选择“INFORMATION_MESSAGE”、“图标消息”，然后打开“下拉列表选择框”的输入对话框，将打开如图 12.12 所示的对话框。

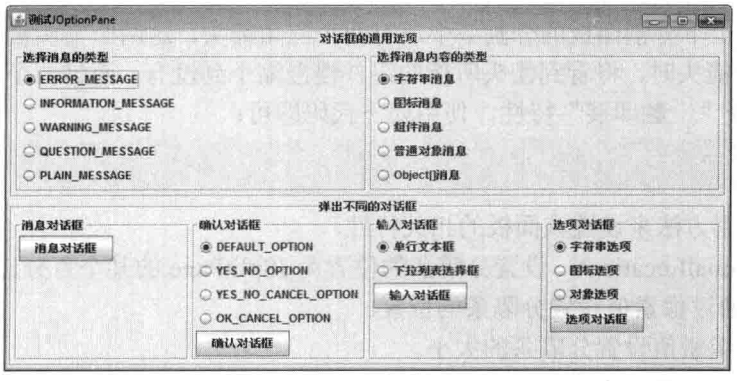


图 12.11 测试对话框的窗口

图 12.12 对话框实例

读者可以通过运行上面程序来查看 JOptionPane 所创建的各种对话框。

12.3 Swing 中的特殊容器

Swing 提供了一些具有特殊功能的容器，这些特殊容器可以用于创建一些更复杂的用户界面。下面将依次介绍这些特殊容器。

12.3.1 使用 JSplitPane

JSplitPane 用于创建一个分割面板，它可以将一个组件（通常是一个容器）分割成两个部分，并提

供一个分割条，用户可以拖动该分割条来调整两个部分的大小。图 12.13 显示了分割面板效果，图中所示的窗口先被分成左右两块，其中左边一块又被分为上下两块。

从图 12.13 中可以看出，分割面板的实质是一个特殊容器，该容器只能容纳两个组件，而且分割面板又分为上下分割、左右分割两种情形，所以创建分割面板的代码非常简单，如下代码所示。

```
new JSplitPane(方向, 左/上组件, 右/下组件)
```

除此之外，创建分割面板时可以指定一个 newContinuousLayout 参数，该参数指定该分割面板是否支持“连续布局”，如果分割面板支持连续布局，则用户拖动分割条时两边组件将会不断调整大小；如果不支持连续布局，则拖动分割条时两边组件不会调整大小，而是只看到一条虚拟的分割条在移动，如图 12.14 所示。

图 12.13　分割面板效果

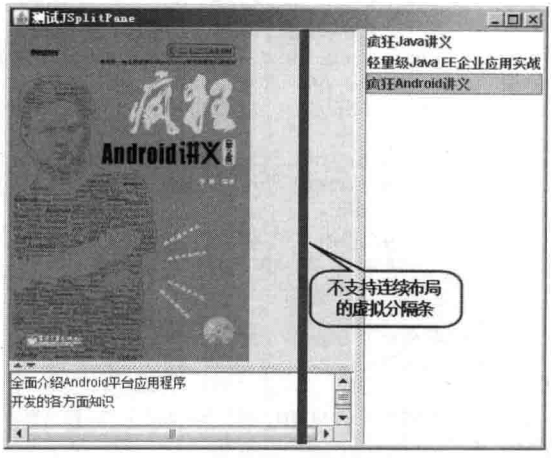


图 12.14　不支持连续布局的虚拟分割条

JSplitPane 默认关闭连续布局特性，因为使用连续布局需要不断重绘两边的组件，因此运行效率很低。如果需要打开指定 JSplitPane 面板的连续布局特性，则可以使用如下代码：

```
// 打开 JSplitPane 的连续布局特性
jsp.setContinuousLayout(true);
```

除此之外，正如图 12.13 中看到的，上下分割面板的分割条中还有两个三角箭头，这两个箭头被称为“一触即展”键，当用户单击某个三角箭头时，将看到箭头所指的组件慢慢缩小到没有，而另一个组件则扩大到占据整个面板。如果需要打开“一触即展”特性，使用如下代码即可：

```
// 打开“一触即展”特性
jsp.setOneTouchExpandable(true);
```

JSplitPane 分割面板还有如下几个可用方法来设置该面板的相关特性。

- setDividerLocation(double proportionalLocation)：设置分隔条的位置为 JSplitPane 的某个百分比。
- setDividerLocation(int location)：通过像素值设置分隔条的位置。
- setDividerSize(int newSize)：通过像素值设置分隔条的大小。
- setLeftComponent(Component comp)/setTopComponent(Component comp)：将指定组件放置到分割面板的左边或者上面。
- setRightComponent(Component comp)/setBottomComponent(Component comp)：将指定组件放置到分割面板的右边或者下面。

下面程序简单示范了 JSplitPane 的用法。

程序清单：codes\12\12.3\SplitPaneTest.java

```
public class SplitPaneTest
{
    Book[] books = new Book[]{
        new Book("疯狂 Java 讲义" , new ImageIcon("ico/java.png")
            , "国内关于 Java 编程最全面的图书\n 看得懂，学得会")
        , new Book("轻量级 Java EE 企业应用实战" , new ImageIcon("ico/ee.png")
            , "SSH 整合开发的经典图书，值得拥有")
```

```
        , new Book("疯狂 Android 讲义" , new ImageIcon("ico/android.png")
            , "全面介绍 Android 平台应用程序\n 开发的各方面知识")
    };
    JFrame jf = new JFrame("测试 JSplitPane");
    JList<Book> bookList = new JList<>(books);
    JLabel bookCover = new JLabel();
    JTextArea bookDesc = new JTextArea();
    public void init()
    {
        // 为三个组件设置最佳大小
        bookList.setPreferredSize(new Dimension(150, 300));
        bookCover.setPreferredSize(new Dimension(300, 150));
        bookDesc.setPreferredSize(new Dimension(300, 150));
        // 为下拉列表添加事件监听器
        bookList.addListSelectionListener(event {
            Book book = (Book)bookList.getSelectedValue();
            bookCover.setIcon(book.getIco());
            bookDesc.setText(book.getDesc());
        });
        // 创建一个垂直的分割面板，
        // 将 bookCover 放在上面，将 bookDesc 放在下面，支持连续布局
        JSplitPane left = new JSplitPane(JSplitPane.VERTICAL_SPLIT
            , true , bookCover, new JScrollPane(bookDesc));
        // 打开“一触即展”特性
        left.setOneTouchExpandable(true);
        // 下面代码设置分割条的大小
        // left.setDividerSize(50);
        // 设置该分割面板根据所包含组件的最佳大小来调整布局
        left.resetToPreferredSizes();
        // 创建一个水平的分割面板
        // 将 left 组件放在左边，将 bookList 组件放在右边
        JSplitPane content = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT
            , left, bookList);
        jf.add(content);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.pack();
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new SplitPaneTest().init();
    }
}
```

上面代码中粗体字代码创建了两个 JSplitPane，其中一个支持连续布局，另一个不支持连续布局。运行上面程序，将可看到如图 12.13 所示的界面。

12.3.2 使用 JTabbedPane

JTabbedPane 可以很方便地在窗口上放置多个标签页，每个标签页相当于获得了一个与外部容器具有相同大小的组件摆放区域。通过这种方式，就可以在一个容器里放置更多的组件，例如右击桌面上的“我的电脑”图标，在弹出的快捷菜单里单击“属性”菜单项，就可以看到一个“系统属性”对话框，这个对话框里包含了 7 个标签页。

如果需要使用 JTabbedPane 在窗口上创建标签页，则可以按如下步骤进行。

① 创建一个 JTabbedPane 对象，JTabbedPane 提供了几个重载的构造器，这些构造器里一共包含如下两个参数。

- tabPlacement：该参数指定标签页标题的放置位置，例如前面介绍的“系统属性”对话框里标签页的标题放在窗口顶部。Swing 支持将标签页标题放在窗口的 4 个方位：TOP（顶部）、LEFT（左边）、BOTTOM（下部）和 RIGHT（右边）。
- tabLayoutPolicy：指定标签页标题的布局策略。当窗口不足以在同一行摆放所有的标签页标题时，Swing 有两种处理方式——将标签页标题换行（JTabbedPane.WRAP_TAB_LAYOUT）排列，或者使用滚动条来控制标签页标题的显示（SCROLL_TAB_LAYOUT）。

提示：

即使创建 JTabbedPane 时没有指定这两个参数，程序也可以在后面改变 JTabbedPane 的这两个属性。例如，通过 setTabLayoutPolicy()方法改变标签页标题的布局策略；使用 setTabPlacement()方法设置标签页标题的放置位置。

例如，下面代码创建一个 JTabbedPane 对象，该 JTabbedPane 的标签页标题位于窗口左侧，当窗口的一行不能摆放所有的标签页标题时，JTabbedPane 将采用换行方式来排列标签页标题。

```
JTabbedPane tabPane = new JTabbedPane(JTabbedPane.LEFT
    , JTabbedPane.WRAP-TAB_LAYOUT);
```

② 调用 JTabbedPane 对象的 addTab()、insertTab()、setComponentAt()、removeTabAt()方法来增加、插入、修改和删除标签页。其中 addTab()方法总是在最前面增加标签页，而 insertTab()、setComponentAt()、removeTabAt()方法都可以使用一个 index 参数，表示在指定位置插入标签页，修改指定位置的标签页，删除指定位置的标签页。

添加标签页时可以指定该标签页的标题（title）、图标（icon），以及该 Tab 页面的组件（component）及提示信息（tip），这 4 个参数都可以是 null；如果某个参数是 null，则对应的内容为空。

不管使用增加、插入、修改哪种操作来改变 JTabbedPane 中的标签页，都是传入一个 Component 组件作为标签页。也就是说，如果希望在某个标签页内放置更多的组件，则必须先将这些组件放置到一个容器（例如 JPanel）里，然后将该容器设置为 JTabbedPane 指定位置的组件。

注意：

不要使用 JTabbedPane 的 add()方法来添加组件，该方法是 JTabbedPane 重写 Containner 容器中的 add()方法，如果使用该 add()方法来添加 Tab 页面，每次添加的标签页会直接覆盖原有的标签页。

③ 如果需要让某个标签页显示出来，则可以通过调用 JTabbedPane 的 setSelectedIndex()方法来实现。例如如下代码：

```
// 设置第三个 Tab 页面处于显示状态
tabPane.setSelectedIndex(2);
// 设置最后一个 Tab 页面处于显示状态
tabPane.setSelectedIndex(tabPanel.getTabCount() - 1);
```

④ 正如上面代码见到的，程序还可通过 JTabbedPane 提供的一系列方法来操作 JTabbedPane 的相关属性。例如，有如下几个常用方法。

- setDisabledIconAt(int index, Icon disabledIcon)：将指定位置的禁用图标设置为 icon，该图标也可以是 null，表示不使用禁用图标。
- setEnabledAt(int index, boolean enabled)：设置指定位置的标签页是否启用。
- setForegroundAt(int index, Color foreground)：设置指定位置标签页的前景色为 foreground。该颜色可以是 null，这时将使用该 JTabbedPane 的前景色作为此标签页的前景色。
- setIconAt(int index, Icon icon)：设置指定位置标签页的图标。
- setTitleAt(int index, String title)：设置指定位置标签页的标题为 title，该 title 可以是 null，这表明设置该标签页的标题为空。
- setToolTipTextAt(int index, String toolTipText)：设置指定位置标签页的提示文本。

实际上，Swing 也为这些 setter 方法提供了对应的 getter 方法，用于返回这些属性。

⑤ 如果程序需要监听用户单击标签页的事件，例如，当用户单击某个标签页时才载入该标签页的内容，则可以使用 ChangeListener 监听器来监听 JTabbedPane 对象。例如如下代码：

```
tabPane.addChangeListener(listener);
```

当用户单击标签页时，系统将把该事件封装成 ChangeEvent 对象，并作为参数来触发 ChangeListener 里的 stateChanged 事件处理器方法。

下面程序定义了具有5个标签页的JTabbedPane面板，该程序可以让用户选择标签布局策略、标签位置。

程序清单：codes\12\12.3\JTabbedPaneTest.java

```
public class JTabbedPaneTest
{
    JFrame jf = new JFrame("测试 Tab 页面");
    // 创建一个 Tab 页面的标签放在左边，采用换行布局策略的 JTabbedPane
    JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.LEFT
        , JTabbedPane.WRAP_TAB_LAYOUT);
    ImageIcon icon = new ImageIcon("ico/close.gif");
    String[] layouts = {"换行布局" , "滚动条布局"};
    String[] positions = {"左边" , "顶部" , "右边" , "底部"};
    Map<String , String> books = new LinkedHashMap<>();
    public void init()
    {
        books.put("疯狂 Java 讲义" , "java.png");
        books.put("轻量级 Java EE 企业应用实战" , "ee.png");
        books.put("疯狂 Ajax 讲义" , "ajax.png");
        books.put("疯狂 Android 讲义" , "android.png");
        books.put("经典 Java EE 企业应用实战" , "classic.png");
        String tip = "可看到本书的封面照片";
        // 向 JTabbedPane 中添加 5 个标签页，指定了标题、图标和提示
        // 但该标签页的组件为 null
        for (String bookName : books.keySet())
        {
            tabbedPane.addTab(bookName, icon, null , tip);
        }
        jf.add(tabbedPane, BorderLayout.CENTER);
        // 为 JTabbedPane 添加事件监听器
        tabbedPane.addChangeListener(event -> {
            // 如果被选择的组件依然是空
            if (tabbedPane.getSelectedComponent() == null)
            {
                // 获取所选标签页
                int n = tabbedPane.getSelectedIndex();
                // 为指定标签页加载内容
                loadTab(n);
            }
        });
        // 系统默认选择第一页，加载第一页内容
        loadTab(0);
        tabbedPane.setPreferredSize(new Dimension(500 , 300));
        // 增加控制标签布局、标签位置的单选按钮
        JPanel buttonPanel = new JPanel();
        ChangeAction action = new ChangeAction();
        buttonPanel.add(new ButtonPanel(action
            , "选择标签布局策略" , layouts));
        buttonPanel.add (new ButtonPanel(action
            , "选择标签位置" , positions));
        jf.add(buttonPanel, BorderLayout.SOUTH);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.pack();
        jf.setVisible(true);
    }
    // 为指定标签页加载内容
    private void loadTab(int n)
    {
        String title = tabbedPane.getTitleAt(n);
        // 根据标签页的标题获取对应的图书封面
        ImageIcon bookImage = new ImageIcon("ico/"
            + books.get(title));
        tabbedPane.setComponentAt(n , new JLabel(bookImage));
        // 改变标签页的图标
        tabbedPane.setIconAt(n, new ImageIcon("ico/open.gif"));
    }
    // 定义改变标签页的布局策略、放置位置的监听器
    class ChangeAction implements ActionListener
    {
```

```
        public void actionPerformed(ActionEvent event)
        {
            JRadioButton source = (JRadioButton)event.getSource();
            String selection = source.getActionCommand();
            // 设置标签页的标题布局策略
            if (selection.equals(layouts[0]))
            {
                tabbedPane.setTabLayoutPolicy(
                    JTabbedPane.WRAP_TAB_LAYOUT);
            }
            else if (selection.equals(layouts[1]))
            {
                tabbedPane.setTabLayoutPolicy(
                    JTabbedPane.SCROLL_TAB_LAYOUT);
            }
            // 设置标签页的标题放置位置
            else if (selection.equals(positions[0]))
            {
                tabbedPane.setTabPlacement(JTabbedPane.LEFT);
            }
            else if (selection.equals(positions[1]))
            {
                tabbedPane.setTabPlacement(JTabbedPane.TOP);
            }
            else if (selection.equals(positions[2]))
            {
                tabbedPane.setTabPlacement(JTabbedPane.RIGHT);
            }
            else if (selection.equals(positions[3]))
            {
                tabbedPane.setTabPlacement(JTabbedPane.BOTTOM);
            }
        }
    }
    public static void main(String[] args)
    {
        new JTabbedPaneTest().init();
    }
}
// 定义一个 JPanel 类扩展类，该类的对象包含多个纵向排列的 JRadioButton 控件
// 且 JPanel 扩展类可以指定一个字符串作为 TitledBorder
class ButtonPanel extends JPanel
{
    private ButtonGroup group;
    public ButtonPanel(JTabbedPaneTest.ChangeAction action
        , String title, String[] labels)
    {
        setBorder(BorderFactory.createTitledBorder(BorderFactory
            .createEtchedBorder(), title));
        setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
        group = new ButtonGroup();
        for (int i = 0; labels!= null && i < labels.length; i++)
        {
            JRadioButton b = new JRadioButton(labels[i]);
            b.setActionCommand(labels[i]);
            add(b);
            // 添加事件监听器
            b.addActionListener(action);
            group.add(b);
            b.setSelected(i == 0);
        }
    }
}
```

上面程序中的粗体字代码是操作 JTabbedPane 各种属性的代码，这些代码完成了向 JTabbedPane 中添加标签页、改变标签页图标等操作。程序运行后会看到如图 12.15 所示的标签页效果。

如果选择滚动条布局，并选择将标签放在底部，将看到如图 12.16 所示的标签页效果。

图 12.15 标签页效果一

图 12.16 标签页效果二

12.3.3 使用 JLayeredPane、JDesktopPane 和 JInternalFrame

JLayeredPane 是一个代表有层次深度的容器，它允许组件在需要时互相重叠。当向 JLayeredPane 容器中添加组件时，需要为该组件指定一个深度索引，其中层次索引较高的层里的组件位于其他层的组件之上。

JLayeredPane 还将容器的层次深度分成几个默认层，程序只是将组件放入相应的层，从而可以更容易地确保组件的正确重叠，无须为组件指定具体的深度索引。JLayeredPane 提供了如下几个默认层。

- DEFAULT_LAYER：大多数组件位于的标准层。这是最底层。
- PALETTE_LAYER：调色板层位于默认层之上。该层对于浮动工具栏和调色板很有用，因此可以位于其他组件之上。
- MODAL_LAYER：该层用于显示模式对话框。它们将出现在容器中所有工具栏、调色板或标准组件的上面。
- POPUP_LAYER：该层用于显示右键菜单，与对话框、工具提示和普通组件关联的弹出式窗口将出现在对应的对话框、工具提示和普通组件之上。
- DRAG_LAYER：该层用于放置拖放过程中的组件（关于拖放操作请看下一节内容），拖放操作中的组件位于所有组件之上。一旦拖放操作结束后，该组件将重新分配到其所属的正常层。

注意：

每一层都是一个不同的整数。可以在调用 add() 的过程中通过 Integer 参数指定该组件所在的层。也可以传入上面几个静态常量，它们分别等于 0，100，200，300，400 等值。

除此之外，也可以使用 JLayeredPane 的 moveToFront()、moveToBack()和 setPosition()方法在组件所在层中对其进行重定位，还可以使用 setLayer()方法更改该组件所属的层。

下面程序简单示范了 JLayeredPane 容器的用法。

程序清单：codes\12\12.3\JLayeredPaneTest.java

```
public class JLayeredPaneTest
{
    JFrame jf = new JFrame("测试 JLayeredPane");
    JLayeredPane layeredPane = new JLayeredPane();
    public void init()
    {
        // 向 layeredPane 中添加 3 个组件
        layeredPane.add(new ContentPanel(10 , 20 , "疯狂 Java 讲义"
            , "ico/java.png"), JLayeredPane.MODAL_LAYER);
        layeredPane.add(new ContentPanel(100 , 60 , "疯狂 Android 讲义"
            , "ico/android.png"), JLayeredPane.DEFAULT_LAYER);
        layeredPane.add(new ContentPanel(190 , 100
            , "轻量级 Java EE 企业应用实战", "ico/ee.png"), 4);
```

```
        layeredPane.setPreferredSize(new Dimension(400, 300));
        layeredPane.setVisible(true);
        jf.add(layeredPane);
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new JLayeredPaneTest().init();
    }
}
// 扩展了 JPanel 类，可以直接创建一个放在指定位置
// 且有指定标题、放置指定图标的 JPanel 对象
class ContentPanel extends JPanel
{
    public ContentPanel(int xPos , int yPos
        , String title , String ico)
    {
        setBorder(BorderFactory.createTitledBorder(
            BorderFactory.createEtchedBorder(), title));
        JLabel label = new JLabel(new ImageIcon(ico));
        add(label);
        setBounds(xPos , yPos , 160, 220);    // ①
    }
}
```

上面程序中粗体字代码向 JLayeredPane 中添加了三个 Panel 组件，每个 Panel 组件都必须显式设置大小和位置（程序中①处代码设置了 Panel 组件的大小和位置），否则该组件不能被显示出来。

运行上面程序，会看到如图 12.17 所示的运行效果。

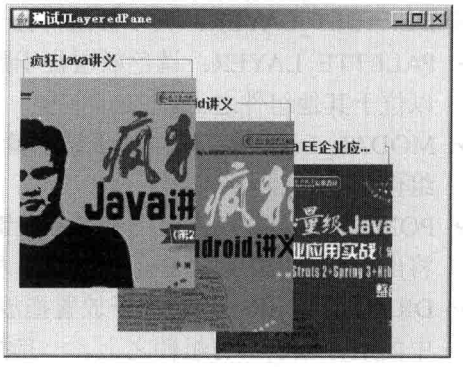


图 12.17　使用 JLayeredPane 的效果

注意：

向 JLayeredPane 中添加组件时，必须显式设置该组件的大小和位置，否则该组件不能显示出来。

JLayeredPane的子类JDesktopPane容器更加常用——很多应用程序都需要启动多个内部窗口来显示信息（典型的如 Eclipse、EditPlus 都使用了这种内部窗口来分别显示每个 Java 源文件），这些内部窗口都属于同一个外部窗口，当外部窗口最小化时，这些内部窗口都被隐藏起来。在 Windows 环境中，这种用户界面被称为多文档界面（Multiple Document Interface，MDI）。

使用 Swing 可以非常简单地创建出这种 MDI 界面，通常，内部窗口有自己的标题栏、标题、图标、三个窗口按钮，并允许拖动改变内部窗口的大小和位置，但内部窗口不能拖出外部窗口。

提示：

内部窗口与外部窗口表现方式上的唯一区别在于：外部窗口的桌面是实际运行平台的桌面，而内部窗口以外部窗口的指定容器作为桌面。就其实现机制来看，外部窗口和内部窗口则完全不同，外部窗口需要部分依赖于本地平台的 GUI 组件，属于重量级组件；而内部窗口则采用 100%的 Java 实现，属于轻量级组件。

JDesktopPane 需要和 JInternalFrame 结合使用，其中 JDesktopPane 代表一个虚拟桌面，而 JInternalFrame 则用于创建内部窗口。使用 JDesktopPane 和 JInternalFrame 创建内部窗口按如下步骤进行即可。

① 创建一个 JDesktopPane 对象。JDesktopPane 类仅提供了一个无参数的构造器，通过该构造器创建 JDesktopPane 对象，该对象代表一个虚拟桌面。

② 使用 JInternalFrame 创建一个内部窗口。创建内部窗口与创建 JFrame 窗口有一些区别，创建 JInternalFrame 对象时除了可以传入一个字符串作为该内部窗口的标题之外，还可以传入 4 个 boolean 值，用于指定该内部窗口是否允许改变窗口大小、关闭窗口、最大化窗口、最小化窗口。例如，下面代码可以创建一个内部窗口。

```
// 创建内部窗口
final JInternalFrame iframe = new JInternalFrame("新文档",
    true, // 可改变大小
    true, // 可关闭
    true, // 可最大化
    true); // 可最小化
```

③ 一旦获得了内部窗口之后，该窗口的用法和普通窗口的用法基本相似，一样可以指定该窗口的布局管理器，一样可以向窗口内添加组件、改变窗口图标等。关于操作内部窗口具体存在哪些方法，请参阅 JInternalFrame 类的 API 文档。

④ 将该内部窗口以合适大小、在合适位置显示出来。与普通窗口类似的是，该窗口默认大小是 0×0 像素，位于 0,0 位置（虚拟桌面的左上角处），并且默认处于隐藏状态，程序可以通过如下代码将内部窗口显示出来。

```
// 同时设置窗口的大小和位置
iframe.reshape(20, 20, 300, 400);
// 使该窗口可见，并尝试选中它
iframe.show();
```

⑤ 将内部窗口添加到 JDesktopPane 容器中，再将 JDesktopPane 容器添加到其他容器中。

注意：

外部窗口的 show()方法已经过时了，不再推荐使用。但内部窗口的 show()方法没有过时，该方法不仅可以让内部窗口显示出来，而且可以让该窗口处于选中状态。

注意：

JDesktopPane 不能独立存在，必须将 JDesktopPane 添加到其他顶级容器中才可以正常使用。

下面程序示范了如何使用 JDesktopPane 和 JInternalFrame 来创建 MDI 界面。

程序清单：codes\12\12.3\JInternalFrameTest.java

```
public class JInternalFrameTest
{
    final int DESKTOP_WIDTH = 480;
    final int DESKTOP_HEIGHT = 360;
    final int FRAME_DISTANCE = 30;
    JFrame jf = new JFrame("MDI 界面");
    // 定义一个虚拟桌面
    private MyJDesktopPane desktop = new MyJDesktopPane();
    // 保存下一个内部窗口的坐标点
    private int nextFrameX;
    private int nextFrameY;
    // 定义内部窗口为虚拟桌面的 1/2 大小
    private int width = DESKTOP_WIDTH / 2;
    private int height = DESKTOP_HEIGHT / 2;
    // 为主窗口定义两个菜单
    JMenu fileMenu = new JMenu("文件");
    JMenu windowMenu = new JMenu("窗口");
    // 定义 newAction 用于创建菜单和工具按钮
    Action newAction = new AbstractAction("新建"
        , new ImageIcon("ico/new.png"))
```

```
    {
        public void actionPerformed(ActionEvent event)
        {
            // 创建内部窗口
            final JInternalFrame iframe = new JInternalFrame("新文档",
                true, // 可改变大小
                true, // 可关闭
                true, // 可最大化
                true); // 可最小化
            iframe.add(new JScrollPane(new JTextArea(8, 40)));
            // 将内部窗口添加到虚拟桌面中
            desktop.add(iframe);
            // 设置内部窗口的原始位置（内部窗口默认大小是 0×0，放在 0,0 位置）
            iframe.reshape(nextFrameX, nextFrameY, width, height);
            // 使该窗口可见，并尝试选中它
            iframe.show();
            // 计算下一个内部窗口的位置
            nextFrameX += FRAME_DISTANCE;
            nextFrameY += FRAME_DISTANCE;
            if (nextFrameX + width > desktop.getWidth()) nextFrameX = 0;
            if (nextFrameY + height > desktop.getHeight()) nextFrameY = 0;
        }
    };
    // 定义 exitAction 用于创建菜单和工具按钮
    Action exitAction = new AbstractAction("退出"
        , new ImageIcon("ico/exit.png"))
    {
        public void actionPerformed(ActionEvent event)
        {
            System.exit(0);
        }
    };
    public void init()
    {
        // 为窗口安装菜单条和工具条
        JMenuBar menuBar = new JMenuBar();
        JToolBar toolBar = new JToolBar();
        jf.setJMenuBar(menuBar);
        menuBar.add(fileMenu);
        fileMenu.add(newAction);
        fileMenu.add(exitAction);
        toolBar.add(newAction);
        toolBar.add(exitAction);
        menuBar.add(windowMenu);
        JMenuItem nextItem = new JMenuItem("下一个");
        nextItem.addActionListener(event -> desktop.selectNextWindow());
        windowMenu.add(nextItem);
        JMenuItem cascadeItem = new JMenuItem("级联");
        cascadeItem.addActionListener(event ->
            // 级联显示窗口，内部窗口的大小是外部窗口的 0.75 倍
            desktop.cascadeWindows(FRAME_DISTANCE , 0.75));
        windowMenu.add(cascadeItem);
        JMenuItem tileItem = new JMenuItem("平铺");
        // 平铺显示所有内部窗口
        tileItem.addActionListener(event -> desktop.tileWindows());
        windowMenu.add(tileItem);
        final JCheckBoxMenuItem dragOutlineItem = new
            JCheckBoxMenuItem("仅显示拖动窗口的轮廓");
        dragOutlineItem.addActionListener(event ->
            // 根据该菜单项是否选择来决定采用哪种拖动模式
            desktop.setDragMode(dragOutlineItem.isSelected()
                ? JDesktopPane.OUTLINE_DRAG_MODE
                : JDesktopPane.LIVE_DRAG_MODE));    // ①
        windowMenu.add(dragOutlineItem);
        desktop.setPreferredSize(new Dimension(480, 360));
        // 将虚拟桌面添加到顶级 JFrame 容器中
        jf.add(desktop);
        jf.add(toolBar , BorderLayout.NORTH);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.pack();
```

```
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new JInternalFrameTest().init();
    }
}
class MyJDesktopPane extends JDesktopPane
{
    // 将所有的窗口以级联方式显示
    // 其中 offset 是两个窗口的位移距离
    // scale 是内部窗口与 JDesktopPane 的大小比例
    public void cascadeWindows(int offset , double scale)
    {
        // 定义级联显示窗口时内部窗口的大小
        int width = (int)(getWidth() * scale);
        int height = (int)(getHeight() * scale);
        // 用于保存级联窗口时每个窗口的位置
        int x = 0;
        int y = 0;
        for (JInternalFrame frame : getAllFrames())
        {
            try
            {
                // 取消内部窗口的最大化、最小化
                frame.setMaximum(false);
                frame.setIcon(false);
                // 把窗口重新放置在指定位置
                frame.reshape(x, y, width, height);
                x += offset;
                y += offset;
                // 如果到了虚拟桌面边界
                if (x + width > getWidth()) x = 0;
                if (y + height > getHeight()) y = 0;
            }
            catch (PropertyVetoException e)
            {}
        }
    }
    // 将所有窗口以平铺方式显示
    public void tileWindows()
    {
        // 统计所有窗口
        int frameCount = 0;
        for (JInternalFrame frame : getAllFrames())
        {
            frameCount++;
        }
        // 计算需要多少行、多少列才可以平铺所有窗口
        int rows = (int) Math.sqrt(frameCount);
        int cols = frameCount / rows;
        // 需要额外增加到其他列中的窗口
        int extra = frameCount % rows;
        // 计算平铺时内部窗口的大小
        int width = getWidth() / cols;
        int height = getHeight() / rows;
        // 用于保存平铺窗口时每个窗口在横向、纵向上的索引
        int x = 0;
        int y = 0;
        for (JInternalFrame frame : getAllFrames())
        {
            try
            {
                // 取消内部窗口的最大化、最小化
                frame.setMaximum(false);
                frame.setIcon(false);
                // 将窗口放在指定位置
                frame.reshape(x * width, y * height, width, height);
                y++;
                // 每排完一列窗口
```

```
                if (y == rows)
                {
                    // 开始排放下一列窗口
                    y = 0;
                    x++;
                    // 如果额外多出的窗口与剩下的列数相等
                    // 则后面所有列都需要多排列一个窗口
                    if (extra == cols - x)
                    {
                        rows++;
                        height = getHeight() / rows;
                    }
                }
            }
            catch (PropertyVetoException e)
            {}
        }
    }
    // 选中下一个非图标窗口
    public void selectNextWindow()
    {
        JInternalFrame[] frames = getAllFrames();
        for (int i = 0; i < frames.length; i++)
        {
            if (frames[i].isSelected())
            {
                // 找出下一个非最小化的窗口，尝试选中它
                // 如果选中失败，则继续尝试选中下一个窗口
                int next = (i + 1) % frames.length;
                while (next != i)
                {
                    // 如果该窗口不是处于最小化状态
                    if (!frames[next].isIcon())
                    {
                        try
                        {
                            frames[next].setSelected(true);
                            frames[next].toFront();
                            frames[i].toBack();
                            return;
                        }
                        catch (PropertyVetoException e)
                        {}
                    }
                    next = (next + 1) % frames.length;
                }
            }
        }
    }
}
```

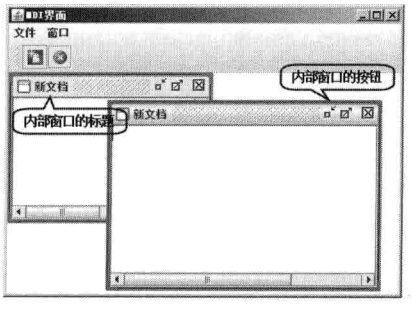


图 12.18　内部窗口效果

上面程序中粗体字代码示范了创建 JDesktopPane 虚拟桌面，创建 JInternatFrame 内部窗口，并将内部窗口添加到虚拟桌面中，最后将虚拟桌面添加到顶级 JFrame 容器中的过程。

运行上面程序，会看到如图 12.18 所示的内部窗口效果。

在默认情况下，当用户拖动窗口时，内部窗口会紧紧跟随用户鼠标的移动，这种操作会导致系统不断重绘虚拟桌面的内部窗口，从而引起性能下降。为了改变这种拖动模式，可以设置当用户拖动内部窗口时，虚拟桌面上仅绘出该内部窗口的轮廓。可以通过调用 JDesktopPane 的 setDragMode()方法来改变内部窗口的拖动模式，该方法接收如下两个参数值。

- JDesktopPane.OUTLINE_DRAG_MODE：拖动过程中仅显示内部窗口的轮廓。

➢ JDesktopPane.LIVE_DRAG_MODE：拖动过程中显示完整窗口，这是默认选项。

上面程序中①处代码允许用户根据 JCheckBoxMenuItem 的状态来决定窗口采用哪种拖动模式。

读者可能会发现，程序创建虚拟桌面时并不是直接创建 JDesktopPane 对象，而是先扩展了 JDesktopPane 类，为该类增加了如下三个方法。

➢ cascadeWindows()：级联显示所有的内部窗口。

➢ tileWindows()：平铺显示所有的内部窗口。

➢ selectNextWindow()：选中当前窗口的下一个窗口。

JDesktopPane 没有提供这三个方法，但这三个方法在 MDI 应用里又是如此常用，以至于开发者总需要自己来扩展 JDesktopPane 类，而不是直接使用该类。这是一个非常有趣的地方——Oracle 似乎认为这些方法太过简单，不屑为之，于是开发者只能自己实现，这给编程带来一些麻烦。

级联显示窗口其实很简单，先根据内部窗口与 JDesktopPane 的大小比例计算出每个内部窗口的大小，然后以此重新排列每个窗口，重排之前让相邻两个窗口在横向、纵向上产生一定的位移即可。

平铺显示窗口相对复杂一点，程序先计算需要几行、几列可以显示所有的窗口，如果还剩下多余（不能整除）的窗口，则依次分布到最后几列中。图 12.19 显示了平铺窗口的效果。

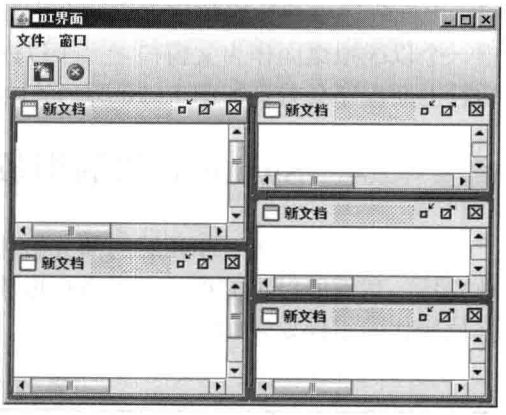


图 12.19　平铺显示所有的窗口效果

前面介绍 JOptionPane 时提到该类包含了多个重载的 showInternalXxxDialog()方法，这些方法用于弹出内部对话框，当使用该方法来弹出内部对话框时通常需要指定一个父组件，这个父组件既可以是虚拟桌面（JDesktopPane 对象），也可以是内部窗口（JInternalFrame 对象）。下面程序示范了如何弹出内部对话框。

程序清单：codes\12\12.3\InternalDialogTest.java

```
public class InternalDialogTest
{
    private JFrame jf = new JFrame("测试内部对话框");
    private JDesktopPane desktop = new JDesktopPane();
    private JButton internalBn = new JButton("内部窗口的对话框");
    private JButton deskBn = new JButton("虚拟桌面的对话框");
    // 定义一个内部窗口，该窗口可拖动，但不可最大化、最小化、关闭
    private JInternalFrame iframe = new JInternalFrame("内部窗口");
    public void init()
    {
        // 向内部窗口中添加组件
        iframe.add(new JScrollPane(new JTextArea(8, 40)));
        desktop.setPreferredSize(new Dimension(400, 300));
        // 把虚拟桌面添加到 JFrame 窗口中
        jf.add(desktop);
        // 设置内部窗口的大小、位置
        iframe.reshape(0 , 0 , 300 , 200);
        // 显示并选中内部窗口
        iframe.show();
        desktop.add(iframe);
        JPanel jp = new JPanel();
        deskBn.addActionListener(event ->
            // 弹出内部对话框，以虚拟桌面作为父组件
            JOptionPane.showInternalMessageDialog(desktop
                , "属于虚拟桌面的对话框"));
        internalBn.addActionListener(event ->
            // 弹出内部对话框，以内部窗口作为父组件
            JOptionPane.showInternalMessageDialog(iframe
                , "属于内部窗口的对话框"));
```

```
            jp.add(deskBn);
            jp.add(internalBn);
            jf.add(jp , BorderLayout.SOUTH);
            jf.pack();
            jf.setVisible(true);
        }
        public static void main(String[] args)
        {
            new InternalDialogTest().init();
        }
    }
```

上面程序中两行粗体字弹出两个内部对话框，这两个对话框一个以虚拟桌面作为父窗口，一个以内部窗口作为父组件。运行上面程序会看到如图 12.20 所示的内部窗口的对话框。

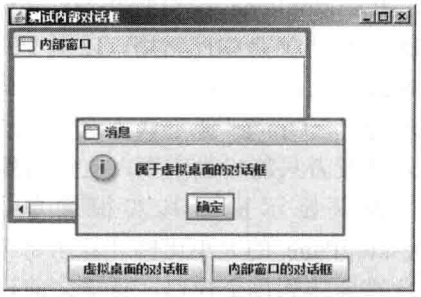


图 12.20　内部窗口的对话框

12.4　Swing 简化的拖放功能

从 JDK 1.4 开始，Swing 的部分组件已经提供了默认的拖放支持，从而能以更简单的方式进行拖放操作。Swing 中支持拖放操作的组件如表 12.1 所示。

表 12.1　支持拖放操作的 Swing 组件

Swing 组件	作为拖放源导出	作为拖放目标接收
JColorChooser	导出颜色对象的本地引用	可接收任何颜色
JFileChooser	导出文件列表	无
JList	导出所选择节点的 HTML 描述	无
JTable	导出所选中的行	无
JTree	导出所选择节点的 HTML 描述	无
JTextComponent	导出所选文本	接收文本，其子类 JTextArea 还可接收文件列表，负责将文件打开

在默认情况下，表 12.1 中的这些 Swing 组件都没有启动拖放支持，可以调用这些组件的 setDragEnabled (true)方法来启动拖放支持。下面程序示范了 Swing 提供的拖放支持。

程序清单：codes\12\12.4\SwingDndSupport.java

```
public class SwingDndSupport
{
    JFrame jf = new JFrame("Swing 的拖放支持");
    JTextArea srcTxt = new JTextArea(8 , 30);
    JTextField jtf = new JTextField(34);
    public void init()
    {
        srcTxt.append("Swing 的拖放支持.\n");
        srcTxt.append("将该文本域的内容拖入其他程序.\n");
        // 启动文本域和单行文本框的拖放支持
        srcTxt.setDragEnabled(true);
        jtf.setDragEnabled(true);
        jf.add(new JScrollPane(srcTxt));
        jf.add(jtf , BorderLayout.SOUTH);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.pack();
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new SwingDndSupport().init();
    }
}
```

上面程序中的两行粗体字代码负责开始多行文本域和单行文本框的拖放支持。运行上面程序，会看到如图 12.21 所示的界面。

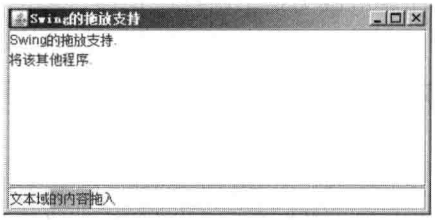


图 12.21 启用 Swing 组件的拖放功能

除此之外，Swing 还提供了一种非常特殊的类：TransferHandler，它可以直接将某个组件的指定属性设置成拖放目标，前提是该组件具有该属性的 setter 方法。例如，JTextArea 类提供了一个 setForeground(Color) 方法，这样即可利用 TransferHandler 将 foreground 定义成拖放目标。代码如下：

```
// 允许直接将一个 Color 对象拖入该 JTextArea 对象，并赋给它的 foreground 属性
txt.setTransferHandler(new TransferHandler("foreground"));
```

下面程序可以直接把颜色选择器面板中的颜色拖放到指定文本域中，用以改变指定文本域的前景色。

程序清单：codes\12\12.4\TransferHandlerTest.java

```
public class TransferHandlerTest
{
    private JFrame jf = new JFrame("测试 TransferHandler");
    JColorChooser chooser = new JColorChooser();
    JTextArea txt = new JTextArea("测试 TransferHandler\n"
        + "直接将上面颜色拖入以改变文本颜色");
    public void init()
    {
        // 启动颜色选择器面板和文本域的拖放功能
        chooser.setDragEnabled(true);
        txt.setDragEnabled(true);
        jf.add(chooser, BorderLayout.SOUTH);
        // 允许直接将一个 Color 对象拖入该 JTextArea 对象
        // 并赋给它的 foreground 属性
        txt.setTransferHandler(new TransferHandler("foreground"));
        jf.add(new JScrollPane(txt));
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.pack();
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new TransferHandlerTest().init();
    }
}
```

上面程序中的粗体字代码将 JTextArea 的 foreground 属性转换成拖放目标，它可以接收任何 Color 对象。而 JColorChooser 启动拖放功能后可以导出颜色对象的本地引用，从而可以直接将该颜色对象拖给 JTextArea 的 foreground 属性。运行上面程序，会看到如图 12.22 所示的界面。

从图 12.22 中可以看出，当用户把颜色选择器面板中预览区的颜色拖到上面多行文本域后，多行文本域的颜色也随之发生改变。

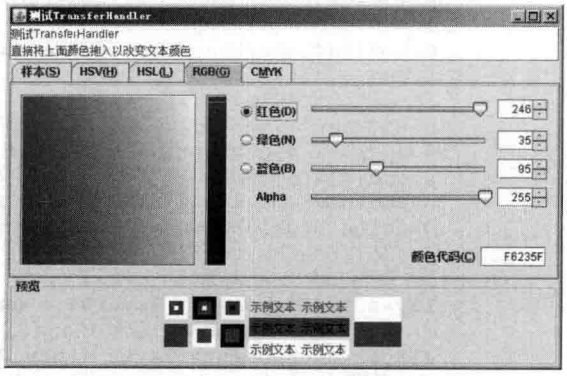


图 12.22 通过拖放操作改变文本域的前景色

12.5 Java 7 新增的 Swing 功能

Java 7 提供的重大更新就包括了对 Swing 的更新，对 Swing 的更新除了前面介绍的 Nimbus 外观、改进的 JColorChooser 组件之外，还有两个很有用的更新——JLayer 和创建不规则窗口。下面将会详细介绍这两个知识点。

12.5.1 使用 JLayer 装饰组件

JLayer 的功能是在指定组件上额外地添加一个装饰层，开发者可以在这个装饰层上进行任意绘制（直接重写 paint(Graphics g, JComponent c)方法），这

样就可以为指定组件添加任意装饰。

JLayer 一般总是要和 LayerUI 一起使用，而 LayerUI 用于被扩展，扩展 LayerUI 时重写它的 paint(Graphics g, JComponent c)方法，在该方法中绘制的内容会对指定组件进行装饰。

实际上，使用 JLayer 很简单，只要如下两行代码即可。

```
// 创建 LayerUI 对象
LayerUI<JComponent> layerUI = new XxxLayerUI();
// 使用 layerUI 来装饰指定的 JPanel 组件
JLayer<JComponent> layer = new JLayer<JComponent>(panel, layerUI);
```

上面程序中的 XxxLayerUI 就是开发者自己扩展的子类，这个子类会重写 paint(Graphics g, JComponent c)方法，重写该方法来完成“装饰层”的绘制。

上面第二行代码中的 panel 组件就是被装饰的组件，接下来把 layer 对象（layer 对象包含了被装饰对象和 LayerUi 对象）添加到指定容器中即可。

下面程序示范了使用 JLayer 为窗口添加一层“蒙版”的效果。

程序清单：codes\12\12.5\JLayerTest.java

```
class FirstLayerUI extends LayerUI<JComponent>
{
    public void paint(Graphics g, JComponent c)
    {
        super.paint(g, c);
        Graphics2D g2 = (Graphics2D) g.create();
        // 设置透明效果
        g2.setComposite(AlphaComposite.getInstance(
            AlphaComposite.SRC_OVER, .5f));
        // 使用渐变画笔绘图
        g2.setPaint(new GradientPaint(0 , 0 , Color.RED
            , 0 , c.getHeight() , Color.BLUE));
        // 绘制一个与被装饰组件具有相同大小的组件
        g2.fillRect(0, 0, c.getWidth(), c.getHeight());      // ①
        g2.dispose();
    }
}
public class JLayerTest
{
    public void init()
    {
        JFrame f = new JFrame("JLayer 测试");
        JPanel p = new JPanel();
        ButtonGroup group = new ButtonGroup();
        JRadioButton radioButton;
        // 创建 3 个 RadioButton，并将它们添加成一组
        p.add(radioButton = new JRadioButton("网购购买", true));
        group.add(radioButton);
        p.add(radioButton = new JRadioButton("书店购买"));
        group.add(radioButton);
        p.add(radioButton = new JRadioButton("图书馆借阅"));
        group.add(radioButton);
        // 添加 3 个 JCheckBox
        p.add(new JCheckBox("疯狂 Java 讲义"));
        p.add(new JCheckBox("疯狂 Android 讲义"));
        p.add(new JCheckBox("疯狂 Ajax 讲义"));
        p.add(new JCheckBox("轻量级 Java EE 企业应用"));
        JButton orderButton = new JButton("投票");
        p.add(orderButton);
        // 创建 LayerUI 对象
        LayerUI<JComponent> layerUI = new FirstLayerUI();      // ②
        // 使用 layerUI 来装饰指定的 JPanel 组件
        JLayer<JComponent> layer = new JLayer<JComponent>(p, layerUI);
        // 将装饰后的 JPanel 组件添加到容器中
        f.add(layer);
        f.setSize(300, 170);
        f.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        f.setVisible (true);
```

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

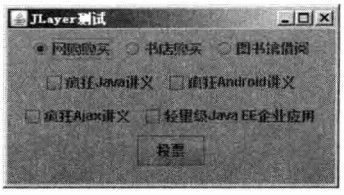


图 12.23 被装饰的 JPanel

上面程序中开发了一个 FirstLayerUI，它扩展了 LayerUI，重写 paint(Graphics g, JComponent c)方法时绘制了一个半透明的、与被装饰组件具有相同大小的矩形。接下来在 main 方法中使用这个 LayerUI 来装饰指定的 JPanel 组件，并把 JLayer 添加到 JFrame 容器中，这就达到了对 JPanel 进行包装的效果。运行该程序，可以看到如图 12.23 所示的效果。

由于开发者可以重写 paint(Graphics g, JComponent c)方法，因此获得对被装饰层的全部控制权——想怎么绘制，就怎么绘制！因此开发者可以“随心所欲”地对指定组件进行装饰。例如，下面提供的 LayerUI 则可以为被装饰组件增加“模糊”效果。程序如下（程序清单同上）。

```
class BlurLayerUI extends LayerUI<JComponent>
{
    private BufferedImage screenBlurImage;
    private BufferedImageOp operation;
    public BlurLayerUI()
    {
        float ninth = 1.0f / 9.0f;
        // 定义模糊参数
        float[] blurKernel = {
            ninth, ninth, ninth,
            ninth, ninth, ninth,
            ninth, ninth, ninth
        };
        // ConvolveOp 代表一个模糊处理，它将原图片的每一个像素与周围
        // 像素的颜色进行混合，从而计算出当前像素的颜色值
        operation = new ConvolveOp(
            new Kernel(3, 3, blurKernel),
            ConvolveOp.EDGE_NO_OP, null);
    }
    public void paint(Graphics g, JComponent c)
    {
        int w = c.getWidth();
        int h = c.getHeight();
        // 如果被装饰窗口大小为 0×0，直接返回
        if (w == 0 || h == 0)
            return;
        // 如果 screenBlurImage 没有初始化，或它的尺寸不对
        if (screenBlurImage == null
            || screenBlurImage.getWidth() != w
            || screenBlurImage.getHeight() != h)
        {
            // 重新创建新的 BufferdImage
            screenBlurImage = new BufferedImage(w
                , h , BufferedImage.TYPE_INT_RGB);
        }
        Graphics2D ig2 = screenBlurImage.createGraphics();
        // 把被装饰组件的界面绘制到当前 screenBlurImage 上
        ig2.setClip(g.getClip());
        super.paint(ig2, c);
        ig2.dispose();
        Graphics2D g2 = (Graphics2D)g;
        // 对 JLayer 装饰的组件进行模糊处理
        g2.drawImage(screenBlurImage, operation, 0, 0);
    }
}
```

上面程序扩展了 LayerUI，重写了 paint(Graphics g, JComponent c)方法，重写该方法时也是绘制

了一个与被装饰组件具有相同大小的矩形，只是这种绘制添加了模糊效果。

将 JLayerTest.java 中的 ②号粗体字代码改为使用 BlurLayerUI，再次运行该程序，将可以看到如图 12.24 所示的“毛玻璃”窗口。

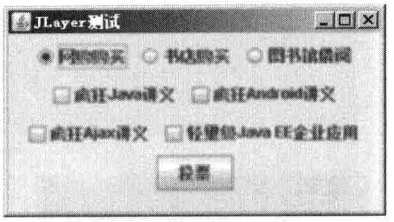


图 12.24　使用 JLayer 装饰的“毛玻璃”窗口

除此之外，开发者自定义的 LayerUI 还可以增加事件机制，这种事件机制能让装饰层响应用户动作，随着用户动作动态地改变 LayerUI 上的绘制效果。比如下面的 LayerUI 示例，程序通过响应鼠标事件，可以在窗口上增加“探照灯”效果。程序如下（程序清单同上）。

```
class SpotlightLayerUI extends LayerUI<JComponent>
{
    private boolean active;
    private int cx, cy;
    public void installUI(JComponent c)
    {
        super.installUI(c);
        JLayer layer = (JLayer)c;
        // 设置 JLayer 可以响应鼠标事件和鼠标动作事件
        layer.setLayerEventMask(AWTEvent.MOUSE_EVENT_MASK
            | AWTEvent.MOUSE_MOTION_EVENT_MASK);      // ①
    }
    public void uninstallUI(JComponent c)
    {
        JLayer layer = (JLayer)c;
        // 设置 JLayer 不响应任何事件
        layer.setLayerEventMask(0);
        super.uninstallUI(c);
    }
    public void paint(Graphics g, JComponent c)
    {
        Graphics2D g2 = (Graphics2D)g.create();
        super.paint (g2, c);
        // 如果处于激活状态
        if (active)
        {
            // 定义一个 cx、cy 位置的点
            Point2D center = new Point2D.Float(cx, cy);
            float radius = 72;
            float[] dist = {0.0f, 1.0f};
            Color[] colors = {Color.YELLOW , Color.BLACK};
            // 以 center 为中心、colors 为颜色数组创建环形渐变
            RadialGradientPaint p = new RadialGradientPaint(center
                , radius , dist , colors);
            g2.setPaint(p);
            // 设置渐变效果
            g2.setComposite(AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER, .6f));
            // 绘制矩形
            g2.fillRect(0, 0, c.getWidth(), c.getHeight());
        }
        g2.dispose();
    }
    // 处理鼠标事件的方法
    public void processMouseEvent(MouseEvent e, JLayer layer)
    {
        if (e.getID() == MouseEvent.MOUSE_ENTERED)
            active = true;
        if (e.getID() == MouseEvent.MOUSE_EXITED)
            active = false;
        layer.repaint();
    }
    // 处理鼠标动作事件的方法
    public void processMouseMotionEvent(MouseEvent e, JLayer layer)
    {
        Point p = SwingUtilities.convertPoint(
```

```
            e.getComponent(), e.getPoint(), layer);
        // 获取鼠标动作事件发生点的坐标
        cx = p.x;
        cy = p.y;
        layer.repaint();
    }
}
```

上面程序中重写了 LayerUI 的 installUI(JComponent c)方法，重写该方法时控制该组件能响应鼠标事件和鼠标动作事件，如粗体字代码所示。接下来程序重写了 processMouseMotionEvent()方法，该方法负责为 LayerUI 上的鼠标事件提供响应——当鼠标在界面上移动时，程序会改变 cx、cy 的坐标值，重写 paint(Graphics g, JComponent c)方法时会在 cx、cy 对应的点绘制一个环形渐变，这就可以充当“探照灯”效果了。将 JLayerTest.java 中的②号粗体字代码改为使用 SpotlightLayerUI，再次运行该程序，即可看到如图 12.25 所示的效果。

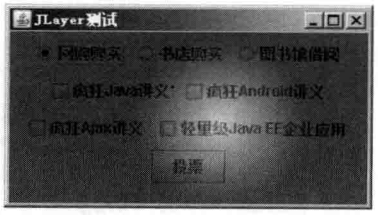


图 12.25 窗口上的“探照灯”效果

既然可以让 LayerUI 上的绘制效果响应鼠标动作，当然也可以在 LayerUI 上绘制“动画”——所谓动画，就是通过定时器控制 LayerUI 上绘制的图形动态地改变即可。

接下来重写的 LayerUI 使用了 Timer 来定时地改变 LayerUI 上的绘制，程序绘制了一个旋转中的“齿轮”，这个旋转的齿轮可以提醒用户“程序正在处理中”。

下面程序重写 LayerUI 时绘制了 12 条辐射状的线条，并通过 Timer 来不断地改变这 12 条线条的排列角度，这样就可以形成“转动的齿轮”了。程序提供的 WaitingLayerUI 类代码如下。

程序清单：codes\12\12.5\WaitingJLayerTest.java

```
class WaitingLayerUI extends LayerUI<JComponent>
{
    private boolean isRunning;
    private Timer timer;
    // 记录转过的角度
    private int angle;      // ①
    public void paint(Graphics g, JComponent c)
    {
        super.paint(g, c);
        int w = c.getWidth();
        int h = c.getHeight();
        // 已经停止运行，直接返回
        if (!isRunning)
            return;
        Graphics2D g2 = (Graphics2D)g.create();
        Composite urComposite = g2.getComposite();
        g2.setComposite(AlphaComposite.getInstance(
            AlphaComposite.SRC_OVER, .5f));
        // 填充矩形
        g2.fillRect(0, 0, w, h);
        g2.setComposite(urComposite);
        // -----下面代码开始绘制转动中的“齿轮”----
        // 计算得到宽、高中较小值的 1/5
        int s = Math.min(w , h) / 5;
        int cx = w / 2;
        int cy = h / 2;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING
            , RenderingHints.VALUE_ANTIALIAS_ON);
        // 设置笔触
        g2.setStroke( new BasicStroke(s / 2
            , BasicStroke.CAP_ROUND , BasicStroke.JOIN_ROUND));
        g2.setPaint(Color.BLUE);
        // 画笔绕被装饰组件的中心转过 angle 度
        g2.rotate(Math.PI * angle / 180, cx, cy);       // ②
        // 循环绘制 12 条线条，形成“齿轮”
        for (int i = 0; i < 12; i++)
        {
```

```
                float scale = (11.0f - (float)i) / 11.0f;
                g2.drawLine(cx + s, cy, cx + s * 2, cy);
                g2.rotate(-Math.PI / 6, cx, cy);
                g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER, scale));
            }
            g2.dispose();
        }
        // 控制等待（齿轮开始转动）的方法
        public void start()
        {
            // 如果已经在运行中，直接返回
            if (isRunning)
                return;
            isRunning = true;
            // 每隔 0.1 秒重绘一次
            timer = new Timer(100, e -> {
                if (isRunning)
                {
                    // 触发 applyPropertyChange()方法，让 JLayer 重绘
                    // 在这行代码中，后面两个参数没有意义
                    firePropertyChange("crazyitFlag", 0 , 1);
                    // 角度加 6
                    angle += 6;     // ③
                    // 到达 360 角度后再从 0 开始
                    if (angle >= 360)
                        angle = 0;
                }
            });
            timer.start();
        }
        // 控制停止等待（齿轮停止转动）的方法
        public void stop()
        {
            isRunning = false;
            // 最后通知 JLayer 重绘一次，清除曾经绘制的图形
            firePropertyChange("crazyitFlag", 0 , 1);
            timer.stop();
        }
        public void applyPropertyChange(PropertyChangeEvent pce
            , JLayer layer)
        {
            // 控制 JLayer 重绘
            if (pce.getPropertyName().equals("crazyitFlag"))
            {
                layer.repaint();
            }
        }
    }
```

上面程序中的①号粗体字代码定义了一个 angle 变量，它负责控制 12 条线条的旋转角度。程序使用 Timer 定时地改变 angle 变量的值（每隔 0.1 秒 angle 加 6），如③号粗体字代码所示。控制了 angle 角度之后，程序根据该 angle 角度绘制 12 条线条，如②号粗体字代码所示。

提供了 WaitingLayerUI 之后，接下来使用该 WaitingLayerUI 与使用前面的 UI 没有任何区别。不过程序需要通过特定事件来显示 WaitingLayerUI 的绘制（就是调用它的 start()方法），下面程序为按钮添加了事件监听器——当用户单击该按钮时，程序会调用 WaitingLayerUI 对象的 start()方法（程序清单同上）。

```
// 为 orderButton 绑定事件监听器：单击该按钮时，调用 layerUI 的 start()方法
orderButton.addActionListener(ae -> {
    layerUI.start();
    // 如果 stopper 定时器已停止，则启动它
    if (!stopper.isRunning())
    {
        stopper.start();
    }
});
```

除此之外，上面代码中还用到了 stopper 计时器，它会控制在一段时间（比如 4 秒）之后停止绘制 WaitingLayerUI，因此程序还通过如下代码进行控制（程序清单同上）。

```
// 设置 4 秒之后执行指定动作：调用 layerUI 的 stop()方法
final Timer stopper = new Timer(4000, ae -> layerUI.stop());
// 设置 stopper 定时器只触发一次
stopper.setRepeats(false);
```

再次运行该程序，可以看到如图 12.26 所示的“动画装饰”效果。

通过上面几个例子可以看出，Swing 提供的 JLayer 为窗口美化提供了无限可能性。只要你想做的，比如希望用户完成输入之后，立即在后面显示一个简单的提示按钮（钩表示输入正确，叉表示输入错误）……都可以通过 JLayer 绘制。

图 12.26 JLayer 生成的“动画装饰”效果

12.5.2 创建透明、不规则形状窗口

Java 7 为 Frame 提供了如下两个方法。

- setShape(Shape shape)：设置窗口的形状，可以将窗口设置成任意不规则的形状。
- setOpacity(float opacity)：设置窗口的透明度，可以将窗口设置成半透明的。当 opacity 为 1.0f 时，该窗口完全不透明。

这两个方法简单、易用，可以直接改变窗口的形状和透明度。除此之外，如果希望开发出渐变透明的窗口，则可以考虑使用一个渐变透明的 JPanel 来代替 JFrame 的 ContentPane；按照这种思路，还可以开发出有图片背景的窗口。

下面程序示范了如何开发出透明、不规则的窗口。

程序清单：codes\12\12.5\NonRegularWindow.java

```
public class NonRegularWindow extends JFrame
    implements ActionListener
{
    // 定义 3 个窗口
    JFrame transWin = new JFrame("透明窗口");
    JFrame gradientWin = new JFrame("渐变透明窗口");
    JFrame bgWin = new JFrame("背景图片窗口");
    JFrame shapeWin = new JFrame("椭圆窗口");
    public NonRegularWindow()
    {
        super("不规则窗口测试");
        setLayout(new FlowLayout());
        JButton transBn = new JButton("透明窗口");
        JButton gradientBn = new JButton("渐变透明窗口");
        JButton bgBn = new JButton("背景图片窗口");
        JButton shapeBn = new JButton("椭圆窗口");
        // 为 3 个按钮添加事件监听器
        transBn.addActionListener(this);
        gradientBn.addActionListener(this);
        bgBn.addActionListener(this);
        shapeBn.addActionListener(this);
        add(transBn);
        add(gradientBn);
        add(bgBn);
        add(shapeBn);
        //-------设置透明窗口-------
        transWin.setLayout(new GridBagLayout());
        transWin.setSize(300,200);
        transWin.add(new JButton("透明窗口里的简单按钮"));
        // 设置透明度为 0.65f，透明度为 1f 时完全不透明
        transWin.setOpacity(0.65f);
        //-------设置渐变透明的窗口-------
        gradientWin.setBackground(new Color(0,0,0,0));
        gradientWin.setSize(new Dimension(300,200));
```

```
// 使用一个 JPanel 对象作为渐变透明的背景
JPanel panel = new JPanel()
{
    protected void paintComponent(Graphics g)
    {
        if (g instanceof Graphics2D)
        {
            final int R = 240;
            final int G = 240;
            final int B = 240;
            // 创建一个渐变画笔
            Paint p = new GradientPaint(0.0f, 0.0f
                , new Color(R, G, B, 0)
                , 0.0f, getHeight()
                , new Color(R, G, B, 255) , true);
            Graphics2D g2d = (Graphics2D)g;
            g2d.setPaint(p);
            g2d.fillRect(0, 0, getWidth(), getHeight());
        }
    }
};
// 使用 JPanel 对象作为 JFrame 的 contentPane
gradientWin.setContentPane(panel);
panel.setLayout(new GridBagLayout());
gradientWin.add(new JButton("渐变透明窗口里的简单按钮"));
//-------设置有背景图片的窗口-------
bgWin.setBackground(new Color(0,0,0,0));
bgWin.setSize(new Dimension(300,200));
// 使用一个 JPanel 对象作为背景图片
JPanel bgPanel = new JPanel()
{
    protected void paintComponent(Graphics g)
    {
        try
        {
            Image bg = ImageIO.read(new File("images/java.png"));
            // 绘制一张图片作为背景
            g.drawImage(bg , 0 , 0 , getWidth() , getHeight() ,  null);
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
};
// 使用 JPanel 对象作为 JFrame 的 contentPane
bgWin.setContentPane(bgPanel);
bgPanel.setLayout(new GridBagLayout());
bgWin.add(new JButton("有背景图片窗口里的简单按钮"));
//-------设置椭圆形窗口-------
shapeWin.setLayout(new GridBagLayout());
shapeWin.setUndecorated(true);
shapeWin.setOpacity(0.7f);
// 通过为 shapeWin 添加监听器来设置窗口的形状
// 当 shapeWin 窗口的大小被改变时，程序动态设置该窗口的形状
shapeWin.addComponentListener(new ComponentAdapter()
{
    // 当窗口大小被改变时，椭圆的大小也会相应地改变
    public void componentResized(ComponentEvent e)
    {
        // 设置窗口的形状
        shapeWin.setShape(new Ellipse2D.Double(0 , 0
            , shapeWin.getWidth() , shapeWin.getHeight())); // ①
    }
});
shapeWin.setSize(300,200);
shapeWin.add(new JButton("椭圆形窗口里的简单按钮"));
//-------设置主程序的窗口-------
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
pack();
```

```
        setVisible(true);
    }
    public void actionPerformed(ActionEvent event)
    {
        switch(event.getActionCommand())
        {
            case "透明窗口":
                transWin.setVisible(true);
                break;
            case "渐变透明窗口":
                gradientWin.setVisible(true);
                break;
            case "背景图片窗口":
                bgWin.setVisible(true);
                break;
            case "椭圆窗口":
                shapeWin.setVisible(true);
                break;
        }
    }
    public static void main(String[] args)
    {
        JFrame.setDefaultLookAndFeelDecorated(true);
        new NonRegularWindow();
    }
}
```

上面程序中的粗体字代码就是设置透明窗口、渐变透明窗口、有背景图片窗口的关键代码；当需要开发不规则形状的窗口时，程序往往会为该窗口实现一个 ComponentListener，该监听器负责监听窗口大小发生改变的事件——当窗口大小发生改变时，程序调用窗口的 setShape()方法来控制窗口的形状，如①号粗体字代码所示。

运行上面程序，打开透明窗口和渐变透明窗口，效果如图 12.27 所示。

打开有背景图片窗口和椭圆窗口，效果如图 12.28 所示。

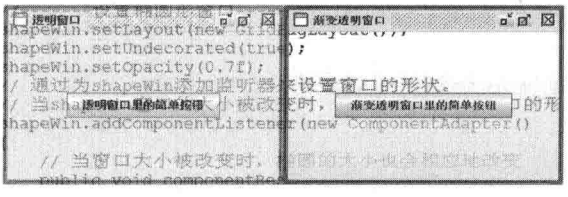


图 12.27 透明窗口和渐变透明窗口

图 12.28 有背景图片窗口和椭圆窗口

12.6 使用 JProgressBar、ProgressMonitor 和 BoundedRangeModel 创建进度条

进度条是图形界面中广泛使用的 GUI 组件，当复制一个较大的文件时，操作系统会显示一个进度条，用于标识复制操作完成的比例；当启动 Eclipse 等程序时，因为需要加载较多的资源，故而启动速度较慢，程序也会在启动过程中显示一个进度条，用以表示该软件启动完成的比例……

12.6.1 创建进度条

使用 JProgressBar 可以非常方便地创建进度条，使用 JProgressBar 创建进度条可按如下步骤进行。

① 创建一个 JProgressBar 对象，创建该对象时可以指定三个参数，用于设置进度条的排列方向（竖直和水平）、进度条的最大值和最小值。也可以在创建该对象时不传入任何参数，而是在后面程序中修改这三个属性。例如，如下代码创建了 JProgressBar 对象。

```
// 创建一条垂直进度条
JProgressBar bar = new JProgressBar(JProgressBar.VERTICAL );
```

② 调用该对象的常用方法设置进度条的普通属性。JProgressBar 除了提供设置排列方向、最大值、最小值的 setter 和 getter 方法之外，还提供了如下三个方法：

- setBorderPainted(boolean b)：设置该进度条是否使用边框。
- setIndeterminate(boolean newValue)：设置该进度条是否是进度不确定的进度条，如果指定一个进度条的进度不确定，将看到一个滑块在进度条中左右移动。
- setStringPainted(boolean newValue)：设置是否在进度条中显示完成百分比。

当然，JProgressBar 也为上面三个属性提供了 getter 方法，但这三个 getter 方法通常没有太大作用。

③ 当程序中工作进度改变时，调用 JProgressBar 对象的 setValue()方法。当进度条的完成进度发生改变时，程序还可以调用进度条对象的如下两个方法。

- double getPercentComplete()：返回进度条的完成百分比。
- String getString()：返回进度字符串的当前值。

下面程序示范了使用进度条的简单例子。

程序清单：codes\12\12.6\JProgressBarTest.java

```
public class JProgressBarTest
{
    JFrame frame = new JFrame("测试进度条");
    // 创建一条垂直进度条
    JProgressBar bar = new JProgressBar(JProgressBar.VERTICAL );
    JCheckBox indeterminate = new JCheckBox("不确定进度");
    JCheckBox noBorder = new JCheckBox("不绘制边框");
    public void init()
    {
        Box box = new Box(BoxLayout.Y_AXIS);
        box.add(indeterminate);
        box.add(noBorder);
        frame.setLayout(new FlowLayout());
        frame.add(box);
        // 把进度条添加到 JFrame 窗口中
        frame.add(bar);
        // 设置进度条的最大值和最小值
        bar.setMinimum(0);
        bar.setMaximum(100);
        // 设置在进度条中绘制完成百分比
        bar.setStringPainted(true);
        // 根据该选择框决定是否绘制进度条的边框
        noBorder.addActionListener(event ->
            bar.setBorderPainted(!noBorder.isSelected()));
        indeterminate.addActionListener(event -> {
            // 设置该进度条的进度是否确定
            bar.setIndeterminate(indeterminate.isSelected());
            bar.setStringPainted(!indeterminate.isSelected());
        });       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
        // 采用循环方式来不断改变进度条的完成进度
        for (int i = 0 ; i <= 100 ; i++)
        {
            // 改变进度条的完成进度
            bar.setValue(i);
            try
            {
                // 程序暂停 0.1 秒
                Thread.sleep(100);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args)
```

```
    {
        new JProgressBarTest().init();
    }
}
```

上面程序中的粗体字代码创建了一个垂直进度条，并通过方法来设置进度条的外观形式——是否包含边框，是否在进度条中显示完成百分比，并通过一个循环来不断改变进度条的 value 属性，该 value 将会自动转换成进度条的完成百分比。

图 12.29 使用进度条

运行该程序，将看到如图 12.29 所示的效果。

在上面程序中，在主程序中使用循环来改变进度条的 value 属性，即修改进度条的完成百分比，这是没有任何意义的事情。通常会希望用进度条去检测其他任务的完成情况，而不是在其他任务的执行过程中主动修改进度条的 value 属性，因为其他任务可能根本不知道进度条的存在。此时可以使用一个计时器来不断取得目标任务的完成情况，并根据其完成情况来修改进度条的 value 属性。下面程序改写了上面程序，使用一个 SimulatedTarget 来模拟一个耗时的任务。

程序清单：codes\12\12.6\JProgressBarTest2.java

```
public class JProgressBarTest2
{
    JFrame frame = new JFrame("测试进度条");
    // 创建一条垂直进度条
    JProgressBar bar = new JProgressBar(JProgressBar.VERTICAL);
    JCheckBox indeterminate = new JCheckBox("不确定进度");
    JCheckBox noBorder = new JCheckBox("不绘制边框");
    public void init()
    {
        Box box = new Box(BoxLayout.Y_AXIS);
        box.add(indeterminate);
        box.add(noBorder);
        frame.setLayout(new FlowLayout());
        frame.add(box);
        // 把进度条添加到 JFrame 窗口中
        frame.add(bar);
        // 设置在进度条中绘制完成百分比
        bar.setStringPainted(true);
        // 根据该选择框决定是否绘制进度条的边框
        noBorder.addActionListener(event ->
            bar.setBorderPainted(!noBorder.isSelected()));
        final SimulatedActivity target = new SimulatedActivity(1000);
        // 以启动一条线程的方式来执行一个耗时的任务
        new Thread(target).start();
        // 设置进度条的最大值和最小值
        bar.setMinimum(0);
        // 以总任务量作为进度条的最大值
        bar.setMaximum(target.getAmount());
        Timer timer = new Timer(300 , e -> bar.setValue(target.getCurrent()));
        timer.start();
        indeterminate.addActionListener(event -> {
            // 设置该进度条的进度是否确定
            bar.setIndeterminate(indeterminate.isSelected());
            bar.setStringPainted(!indeterminate.isSelected());
        });
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args)
    {
        new JProgressBarTest2().init();
    }
}
// 模拟一个耗时的任务
class SimulatedActivity implements Runnable
{
    // 任务的当前完成量
```

```
    private volatile int current;
    // 总任务量
    private int amount;
    public SimulatedActivity(int amount)
    {
        current = 0;
        this.amount = amount;
    }
    public int getAmount()
    {
        return amount;
    }
    public int getCurrent()
    {
        return current;
    }
    // run 方法代表不断完成任务的过程
    public void run()
    {
        while (current < amount)
        {
            try
            {
                Thread.sleep(50);
            }
            catch(InterruptedException e)
            {
            }
            current++;
        }
    }
}
```

上面程序的运行效果与前一个程序的运行效果大致相同，但这个程序中的JProgressBar就实用多了，它可以检测并显示 SimulatedTarget 的完成进度。

提示：

SimulatedActivity 类实现了 Runnable 接口，这是一个特殊的接口，实现该接口可以实现多线程功能。关于多线程的介绍请参考本书第 16 章内容。

Swing 组件大都将外观显示和内部数据分离，JProgressBar 也不例外，JProgressBar 组件有一个用于保存其状态数据的 Model 对象，这个对象由 BoundedRangeModel 对象表示，程序调用 JProgressBar 对象的 setValue()方法时，实际上是设置 BoundedRangeModel 对象的 value 属性。

程序可以修改 BoundedRangeModel 对象的 minimum 属性和 maximum 属性，当该 Model 对象的这两个属性被修改后，它所对应的 JProgressBar 对象的这两个属性也会随之修改，因为 JProgressBar 对象的所有状态数据都是保存在该 Model 对象中的。

程序监听 JProgressBar 完成比例的变化，也是通过为 BoundedRangeModel 提供监听器来实现的。BoundedRangeModel 提供了如下一个方法来添加监听器。

- addChangeListener(ChangeListener x)：用于监听 JProgressBar 完成比例的变化，每当 JProgressBar 的 value 属性被改变时，系统都会触发 ChangeListener 监听器的 stateChanged()方法。例如，下面代码为进度条的状态变化添加了一个监听器。

```
// JProgressBar 的完成比例发生变化时会触发该方法
bar.getModel().addChangeListener(ce -> {
    // 对进度变化进行合适处理
    ...
});
```

12.6.2 创建进度对话框

ProgressMonitor 的用法与 JProgressBar 的用法基本相似，只是 ProgressMonitor 可以直接创建一个进度对话框。ProgressMonitor 提供了如下构造器。

- ProgressMonitor(Component parentComponent, Object message, String note, int min, int max)：该构造器中的 parentComponent 参数用于设置该进度对话框的父组件，message 用于设置该进度对话框的描述信息，note 用于设置该进度对话框的提示文本，min 和 max 用于设置该对话框所包含进度条的最小值和最大值。

例如，如下代码创建了一个进度对话框。

```
final ProgressMonitor dialog = new ProgressMonitor(null ,"等待任务完成" ,
    "已完成: " , 0 , target.getAmount());
```

使用上面代码创建的进度对话框如图 12.30 所示。

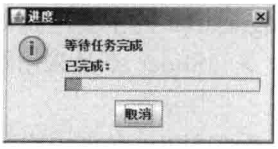


图 12.30 进度对话框

如图 12.30 所示，该对话框中包含了一个“取销”按钮，如果程序希望判断用户是否单击了该按钮，则可以通过 ProgressMonitor 的 isCanceled() 方法进行判断。

使用 ProgressMonitor 创建的对话框里包含的进度条是非常固定的，程序甚至不能设置该进度条是否包含边框（总是包含边框），不能设置进度不确定，不能改变进度条的方向（总是水平方向）。

与普通进度条类似的是，进度对话框也不能自动监视目标任务的完成进度，程序通过调用进度对话框的 setProgress()方法来改变进度条的完成比例（该方法类似于 JProgressBar 的 setValue()方法）。

下面程序同样采用前面的 SimulatedTarget 来模拟一个耗时的任务，并创建了一个进度对话框来监测该任务的完成百分比。

程序清单：codes\12\12.6\ProgressMonitorTest.java

```
public class ProgressMonitorTest
{
    Timer timer;
    public void init()
    {
        final SimulatedActivity target = new SimulatedActivity(1000);
        // 以启动一条线程的方式来执行一个耗时的任务
        final Thread targetThread = new Thread(target);
        targetThread.start();
        final ProgressMonitor dialog = new ProgressMonitor(null
            , "等待任务完成" , "已完成: " , 0 , target.getAmount());
        timer = new Timer(300 , e -> {
            // 以任务的当前完成量设置进度对话框的完成比例
            dialog.setProgress(target.getCurrent());
            // 如果用户单击了进度对话框中的“取消”按钮
            if (dialog.isCanceled())
            {
                // 停止计时器
                timer.stop();
                // 中断任务的执行线程
                targetThread.interrupt();    // ①
                // 系统退出
                System.exit(0);
            }
        });
        timer.start();
    }
    public static void main(String[] args)
    {
        new ProgressMonitorTest().init();
    }
}
```

上面程序中的粗体字代码创建了一个进度对话框，并创建了一个 Timer 计时器不断询问 SimulatedTarget 任务的完成比例，进而设置进度对话框里进度条的完成比例。而且该计时器还负责监听用户是否单击了进度对话框中的“取销”按钮，如果用户单击了该按钮，则中止执行 SimulatedTarget 任务的线程，并停止计时器，同时退出该程序。运行该程序，会看到如图 12.30 所示的对话框。

提示：

程序中①处代码用于中止线程的执行，读者可以参考第16章内容来理解这行代码。

12.7 使用JSlider和BoundedRangeModel创建滑动条

JSlider的用法和JProgressBar的用法非常相似，这一点可以从它们共享同一个Model类看出来。使用JSlider可以创建一个滑动条，这个滑动条同样有最小值、最大值和当前值等属性。JSlider与JprogressBar的主要区别如下。

- JSlider不是采用填充颜色的方式来表示该组件的当前值，而是采用滑块的位置来表示该组件的当前值。
- JSlider允许用户手动改变滑动条的当前值。
- JSlider允许为滑动条指定刻度值，这系列的刻度值既可以是连续的数字，也可以是自定义的刻度值，甚至可以是图标。

使用JSlider创建滑动条的步骤如下。

① 使用JSlider的构造器创建一个JSlider对象，JSlider有多个重载的构造器，但这些构造器总共可以接收如下4个参数。

- orientation：指定该滑动条的摆放方向，默认是水平摆放。可以接收JSlider.VERTICAL和JSlider.HORIZONTAL两个值。
- min：指定该滑动条的最小值，该属性值默认为0。
- max：指定该滑动条的最大值，该属性值默认是为100。
- value：指定该滑动条的当前值，该属性值默认是为50。

② 调用JSlider的如下方法来设置滑动条的外观样式。

- setExtent(int extent)：设置滑动条上的保留区，用户拖动滑块时不能超过保留区。例如，最大值为100的滑动条，如果设置保留区为20，则滑块最大只能拖动到80。
- setInverted(boolean b)：设置是否需要反转滑动条，滑动条的滑轨上刻度值默认从小到大、从左到右排列。如果该方法设置为true，则排列方向会反转过来。
- setLabelTable(Dictionary labels)：为该滑动条指定刻度标签。该方法的参数是Dictionary类型，它是一个古老的、抽象集合类，其子类是Hashtable。传入的Hashtable集合对象的key-value对为{ Integer value, java.swing.JComponent label }格式，刻度标签可以是任何组件。
- setMajorTickSpacing(int n)：设置主刻度标记的间隔。
- setMinorTickSpacing(int n)：设置次刻度标记的间隔。
- setPaintLabels(boolean b)：设置是否在滑块上绘制刻度标签。如果没有为该滑动条指定刻度标签，则默认绘制将刻度值的数值作为标签。
- setPaintTicks(boolean b)：　设置是否在滑块上绘制刻度标记。
- setPaintTrack(boolean b)：设置是否为滑块绘制滑轨。
- setSnapToTicks(boolean b)：设置滑块是否必须停在滑道的有刻度处。如果设置为true，则滑块只能停在有刻度处；如果用户没有将滑块拖到有刻度处，则系统自动将滑块定位到最近的刻度处。

③ 如果程序需要在用户拖动滑块时做出相应处理，则应为该JSlider对象添加事件监听器。JSlider提供了addChangeListener()方法来添加事件监听器，该监听器负责监听滑动值的变化。

④ 将JSlider对象添加到其他容器中显示出来。

下面程序示范了如何使用JSlider来创建滑动条。

程序清单：codes\12\12.7\JSliderTest.java

```
public class JSliderTest
```

```
{
    JFrame mainWin = new JFrame("滑动条示范");
    Box sliderBox = new Box(BoxLayout.Y_AXIS);
    JTextField showVal = new JTextField();
    ChangeListener listener;
    public void init()
    {
        // 定义一个监听器，用于监听所有的滑动条
        listener = event -> {
            // 取出滑动条的值，并在文本中显示出来
            JSlider source = (JSlider) event.getSource();
            showVal.setText("当前滑动条的值为："
                + source.getValue());
        };
        // -----------添加一个普通滑动条-----------
        JSlider slider = new JSlider();
        addSlider(slider, "普通滑动条");
        // -----------添加保留区为 30 的滑动条-----------
        slider = new JSlider();
        slider.setExtent(30);
        addSlider(slider, "保留区为 30");
        // ---添加带主、次刻度的滑动条，并设置其最大值、最小值---
        slider = new JSlider(30 , 200);
        // 设置绘制刻度
        slider.setPaintTicks(true);
        // 设置主、次刻度的间距
        slider.setMajorTickSpacing(20);
        slider.setMinorTickSpacing(5);
        addSlider(slider, "有刻度");
        // -----------添加滑块必须停在刻度处的滑动条-----------
        slider = new JSlider();
        // 设置滑块必须停在刻度处
        slider.setSnapToTicks(true);
        // 设置绘制刻度
        slider.setPaintTicks(true);
        // 设置主、次刻度的间距
        slider.setMajorTickSpacing(20);
        slider.setMinorTickSpacing(5);
        addSlider(slider, "滑块停在刻度处");
        // -----------添加没有滑轨的滑动条-----------
        slider = new JSlider();
        // 设置绘制刻度
        slider.setPaintTicks(true);
        // 设置主、次刻度的间距
        slider.setMajorTickSpacing(20);
        slider.setMinorTickSpacing(5);
        // 设置不绘制滑轨
        slider.setPaintTrack(false);
        addSlider(slider, "无滑轨");
        // -----------添加方向反转的滑动条-----------
        slider = new JSlider();
        // 设置绘制刻度
        slider.setPaintTicks(true);
        // 设置主、次刻度的间距
        slider.setMajorTickSpacing(20);
        slider.setMinorTickSpacing(5);
        // 设置方向反转
        slider.setInverted(true);
        addSlider(slider, "方向反转");
        // --------添加绘制默认刻度标签的滑动条--------
        slider = new JSlider();
        // 设置绘制刻度
        slider.setPaintTicks(true);
        // 设置主、次刻度的间距
        slider.setMajorTickSpacing(20);
        slider.setMinorTickSpacing(5);
        // 设置绘制刻度标签，默认绘制数值刻度标签
        slider.setPaintLabels(true);
        addSlider(slider, "数值刻度标签");
        // ------添加绘制 Label 类型的刻度标签的滑动条------
```

```
        slider = new JSlider();
        // 设置绘制刻度
        slider.setPaintTicks(true);
        // 设置主、次刻度的间距
        slider.setMajorTickSpacing(20);
        slider.setMinorTickSpacing(5);
        // 设置绘制刻度标签
        slider.setPaintLabels(true);
        Dictionary<Integer, Component> labelTable = new Hashtable<>();
        labelTable.put(0, new JLabel("A"));
        labelTable.put(20, new JLabel("B"));
        labelTable.put(40, new JLabel("C"));
        labelTable.put(60, new JLabel("D"));
        labelTable.put(80, new JLabel("E"));
        labelTable.put(100, new JLabel("F"));
        // 指定刻度标签，标签是 JLabel
        slider.setLabelTable(labelTable);
        addSlider(slider, "JLable 标签");
        // ------添加绘制 Label 类型的刻度标签的滑动条------
        slider = new JSlider();
        // 设置绘制刻度
        slider.setPaintTicks(true);
        // 设置主、次刻度的间距
        slider.setMajorTickSpacing(20);
        slider.setMinorTickSpacing(5);
        // 设置绘制刻度标签
        slider.setPaintLabels(true);
        labelTable = new Hashtable<Integer, Component>();
        labelTable.put(0, new JLabel(new ImageIcon("ico/0.GIF")));
        labelTable.put(20, new JLabel(new ImageIcon("ico/2.GIF")));
        labelTable.put(40, new JLabel(new ImageIcon("ico/4.GIF")));
        labelTable.put(60, new JLabel(new ImageIcon("ico/6.GIF")));
        labelTable.put(80, new JLabel(new ImageIcon("ico/8.GIF")));
        // 指定刻度标签，标签是 ImageIcon
        slider.setLabelTable(labelTable);
        addSlider(slider, "Icon 标签");
        mainWin.add(sliderBox, BorderLayout.CENTER);
        mainWin.add(showVal, BorderLayout.SOUTH);
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.pack();
        mainWin.setVisible(true);
    }
    // 定义一个方法，用于将滑动条添加到容器中
    public void addSlider(JSlider slider, String description)
    {
        slider.addChangeListener(listener);
        Box box = new Box(BoxLayout.X_AXIS);
        box.add(new JLabel(description + ": "));
        box.add(slider);
        sliderBox.add(box);
    }
    public static void main(String[] args)
    {
        new JSliderTest().init();
    }
}
```

上面程序向窗口中添加了多个滑动条，程序通过粗体字代码来控制不同滑动条的不同外观。运行上面程序，会看到如图 12.31 所示的各种滑动条的效果。

JSlider 也使用 BoundedRangeModel 作为保存其状态数据的 Model 对象，程序可以直接修改 Model 对象来改变滑动条的状态，但大部分时候程序无须使用该 Model 对象。JSlider 也提供了 addChange Listener() 方法来为滑动条添加监听器，无须像 JProgressBar 那样监听它所对应的 Model 对象。

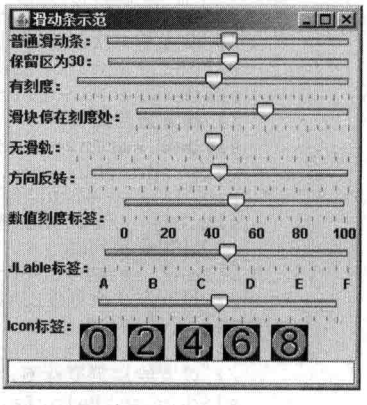


图 12.31　各种滑动条的效果

12.8 使用 JSpinner 和 SpinnerModel 创建微调控制器

JSpinner 组件是一个带有两个小箭头的文本框，这个文本框只能接收满足要求的数据，用户既可以通过两个小箭头调整该微调控制器的值，也可以直接在文本框内输入内容作为该微调控制器的值。当用户在该文本框内输入时，如果输入的内容不满足要求，系统将会拒绝用户输入。典型的 JSpinner 组件如图 12.32 所示。

图 12.32 微调控制器组件

JSpinner 组件常常需要和 SpinnerModel 结合使用，其中 JSpinner 组件控制该组件的外观表现，而 SpinnerModel 则控制该组件内部的状态数据。

JSpinner 组件的值可以是数值、日期和 List 中的值，Swing 为这三种类型的值提供了 SpinnerNumberModel、SpinnerDateModel 和 SpinnerListModel 三个 SpinnerModel 实现类；除此之外，JSpinner 组件的值还可以是任意序列，只要这个序列可以通过 previous()、next()获取值即可。在这种情况下，用户必须自行提供 SpinnerModel 实现类。

使用 JSpinner 组件非常简单，JSpinner 提供了如下两个构造器。

- JSpinner()：创建一个默认的微调控制器。
- JSpinner(SpinnerModel model)：使用指定的 SpinnerModel 来创建微调控制器。

采用第一个构造器创建的默认微调控制器只接收整数值，初始值是 0，最大值和最小值没有任何限制。每单击向下箭头或者向上箭头一次，该组件里的值分别减 1 或加 1。

使用 JSpinner 关键在于使用它对应的三个 SpinnerModel，下面依次介绍这三个 SpinnerModel。

- SpinnerNumberModel：这是最简单的 SpinnerModel，创建该 SpinnerModel 时可以指定 4 个参数：最大值、最小值、初始值、步长，其中步长控制单击上、下箭头时相邻两个值之间的差。这 4 个参数既可以是整数，也可以是浮点数。
- SpinnerDateModel：创建该 SpinnerModel 时可以指定 4 个参数：起始时间、结束时间、初始时间和时间差，其中时间差控制单击上、下箭头时相邻两个时间之间的差值。
- SpinnerListModel：创建该 SpinnerModel 只需要传入一个 List 或者一个数组作为序列值即可。该 List 的集合元素和数组元素可以是任意类型的对象，但由于 JSpinner 组件的文本框只能显示字符串，所以 JSpinner 显示每个对象 toString()方法的返回值。

提示：

从图 12.32 中可以看出，JSpinner 创建的微调控制器和 ComboBox 有点像（由 Swing 的 JComboBox 提供，ComboBox 既允许通过下拉列表框进行选择，也允许直接输入），区别在于 ComboBox 可以产生一个下拉列表框供用户选择，而 JSpinner 组件只能通过上、下箭头逐项选择。使用 ComboBox 通常必须明确指定下拉列表框中每一项的值，但使用 JSpinner 则只需给定一个范围，并指定步长即可；当然，使用 JSpinner 也可以明确给出每一项的值（就是对应使用 SpinnerListModel）。

为了控制 JSpinner 中值的显示格式，JSpinner 还提供了一个 setEditor()方法。Swing 提供了如下 3 个特殊的 Editor 来控制值的显示格式。

- JSpinner.DateEditor：控制 JSpinner 中日期值的显示格式。
- JSpinner.ListEditor：控制 JSpinner 中 List 项的显示格式。
- JSpinner.NumberEditor：控制 JSpinner 中数值的显示格式。

下面程序示范了几种使用 JSpinner 的情形。

程序清单：codes\12\12.8\JSpinnerTest.java

```
public class JSpinnerTest
{
```

```
final int SPINNER_NUM = 6;
JFrame mainWin = new JFrame("微调控制器示范");
Box spinnerBox = new Box(BoxLayout.Y_AXIS);
JSpinner[] spinners = new JSpinner[SPINNER_NUM];
JLabel[] valLabels = new JLabel[SPINNER_NUM];
JButton okBn = new JButton("确定");
public void init()
{
    for (int i = 0 ; i < SPINNER_NUM ; i++ )
    {
        valLabels[i] = new JLabel();
    }
    // -----------普通 JSpinner-----------
    spinners[0] = new JSpinner();
    addSpinner(spinners[0], "普通" , valLabels[0]);
    // -----------指定最小值、最大值、步长的 JSpinner-----------
    // 创建一个 SpinnerNumberModel 对象，指定最小值、最大值和步长
    SpinnerNumberModel numModel = new SpinnerNumberModel(
        3.4 , -1.1 , 4.3 , 0.1);
    spinners[1] = new JSpinner(numModel);
    addSpinner(spinners[1], "数值范围" , valLabels[1]);
    // -----------使用 SpinnerListModel 的 JSpinner------------
    String[] books = new String[]
    {
        "轻量级 Java EE 企业应用实战"
        , "疯狂 Java 讲义"
        , "疯狂 Ajax 讲义"
    };
    // 使用字符串数组创建 SpinnerListModel 对象
    SpinnerListModel bookModel = new SpinnerListModel(books);
    // 使用 SpinnerListModel 对象创建 JSpinner 对象
    spinners[2] = new JSpinner(bookModel);
    addSpinner(spinners[2], "字符串序列值" , valLabels[2]);
    // -----------使用序列值是 ImageIcon 的 JSpinner------------
    ArrayList<ImageIcon> icons = new ArrayList<>();
    icons.add(new ImageIcon("a.gif"));
    icons.add(new ImageIcon("b.gif"));
    // 使用 ImageIcon 数组创建 SpinnerListModel 对象
    SpinnerListModel iconModel = new SpinnerListModel(icons);
    // 使用 SpinnerListModel 对象创建 JSpinner 对象
    spinners[3] = new JSpinner(iconModel);
    addSpinner(spinners[3], "图标序列值" , valLabels[3]);
    // -----------使用 SpinnerDateModel 的 JSpinner------------
    // 分别获取起始时间、结束时间、初时时间
    Calendar cal = Calendar.getInstance();
    Date init = cal.getTime();
    cal.add(Calendar.DAY_OF_MONTH , -3);
    Date start = cal.getTime();
    cal.add(Calendar.DAY_OF_MONTH , 8);
    Date end = cal.getTime();
    // 创建一个 SpinnerDateModel 对象，指定最小时间、最大时间和初始时间
    SpinnerDateModel dateModel = new SpinnerDateModel(init
        , start , end , Calendar.HOUR_OF_DAY);
    // 以 SpinnerDateModel 对象创建 JSpinner
    spinners[4] = new JSpinner(dateModel);
    addSpinner(spinners[4], "时间范围" , valLabels[4]);
    // -----------使用 DateEditor 来格式化 JSpinner------------
    dateModel = new SpinnerDateModel();
    spinners[5] = new JSpinner(dateModel);
    // 创建一个 JSpinner.DateEditor 对象，用于对指定的 Spinner 进行格式化
    JSpinner.DateEditor editor = new JSpinner.DateEditor(
        spinners[5] , "公元 yyyy 年 MM 月 dd 日 HH 时");
    // 设置使用 JSpinner.DateEditor 对象进行格式化
    spinners[5].setEditor(editor);
    addSpinner(spinners[5], "使用 DateEditor" , valLabels[5]);
    // 为“确定”按钮添加一个事件监听器
    okBn.addActionListener(evt -> {
        // 取出每个微调控制器的值，并将该值用后面的 Label 标签显示出来
        for (int i = 0 ; i < SPINNER_NUM ; i++)
        {
```

```
                // 将微调控制器的值通过指定的 JLabel 显示出来
                valLabels[i].setText(spinners[i].getValue().toString());
            }
        });
        JPanel bnPanel = new JPanel();
        bnPanel.add(okBn);
        mainWin.add(spinnerBox, BorderLayout.CENTER);
        mainWin.add(bnPanel, BorderLayout.SOUTH);
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.pack();
        mainWin.setVisible(true);
    }
    // 定义一个方法，用于将滑动条添加到容器中
    public void addSpinner(JSpinner spinner
        , String description , JLabel valLabel)
    {
        Box box = new Box(BoxLayout.X_AXIS);
        JLabel desc = new JLabel(description + ": ");
        desc.setPreferredSize(new Dimension(100 , 30));
        box.add(desc);
        box.add(spinner);
        valLabel.setPreferredSize(new Dimension(180 , 30));
        box.add(valLabel);
        spinnerBox.add(box);
    }
    public static void main(String[] args)
    {
        new JSpinnerTest().init();
    }
}
```

上面程序创建了 6 个 JSpinner 对象，并将它们添加到窗口中显示出来，程序中的粗体字代码用于控制每个微调控制器的具体行为。

第一个 JSpinner 组件是一个默认的微调控制器，其初始值是 0，步长是 1，只能接收整数值。

第二个 JSpinner 通过 SpinnerNumberModel 来创建，指定了 JSpinner 的最小值为–1.1、最大值为 4.3、初始值为 3.4、步长为 0.1，所以用户单击该微调控制器的上、下箭头时，微调控制器的值之间的差值是 0.1，并只能处于–1.1~4.3 之间。

第三个 JSpinner 通过 SpinnerListModel 来创建的，创建 SpinnerListModel 对象时指定字符串数组作为多个序列值，所以当用户单击该微调控制器的上、下箭头时，微调控制器的值总是在该字符串数组之间选择。

第四个 JSpinner 也是通过 SpinnerListModel 来创建的，虽然传给 SpinnerListModel 对象的构造参数是集合元素为 ImageIcon 的 List 对象，但 JSpinner 只能显示字符串内容，所以它会把每个 ImageIcon 对象的 toString()方法返回值当成微调控制器的多个序列值。

第五个 JSpinner 通过 SpinnerDateModel 来创建，而且指定了最小时间、最大时间和初始时间，所以用户单击该微调控制器的上、下箭头时，微调控制器里的时间只能处于指定时间范围之间。这里需要注意的是，SpinnerDateModel 的第 4 个参数没有太大的作用，它不能控制两个相邻时间之间的差。当用户在 JSpinner 组件内选中该时间的指定时间域时，例如年份，则两个相邻时间的时间差就是 1 年。

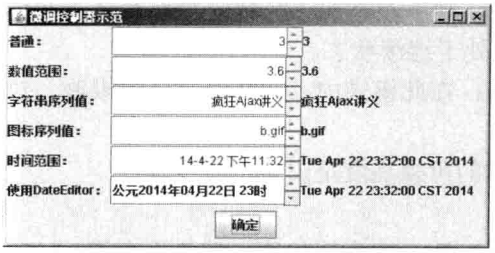


图 12.33 JSpinner 组件的用法示范

第六个 JSpinner 使用 JSpinner.DateEditor 来控制时间微调控制器里日期、时间的显示格式，创建 JSpinner.DateEditor 对象时需要传入一个日期时间格式字符串（dateFormatPattern），该参数用于控制日期、时间的显示格式，关于这个格式字符串的定义方式可以参考 SimpleDateFormat 类的介绍。本例程序中使用"公元 yyyy 年 MM 月 dd 日 HH 时"作为格式字符串。

运行上面程序，会看到如图 12.33 所示的窗口。

程序中还提供了一个“确定”按钮，当单击该按钮时，

系统会把每个微调控制器的值通过对应的 JLabel 标签显示出来，如图 12.33 所示。

12.9 使用 JList、JComboBox 创建列表框

无论从哪个角度来看，JList 和 JComboBox 都是极其相似的，它们都有一个列表框，只是 JComboBox 的列表框需要以下拉方式显示出来；JList 和 JComboBox 都可以通过调用 setRenderer()方法来改变列表项的表现形式。甚至维护这两个组件的 Model 都是相似的，JList 使用 ListModel，JComboBox 使用 ComboBoxModel，而 ComboBoxModel 是 ListModel 的子类。

12.9.1 简单列表框

如果仅仅希望创建一个简单的列表框（包括 JList 和 JComboBox），则直接使用它们的构造器即可，它们的构造器都可接收一个对象数组或元素类型任意的 Vector 作为参数，这个对象数组或元素类型任意的 Vector 里的所有元素将转换为列表框的列表项。

使用 JList 和 JComboBox 来创建简单列表框非常简单，只需要按如下步骤进行即可。

① 使用 JList 或者 JComboBox 的构造器创建一个列表框对象，创建 JList 或 JComboBox 时，应该传入一个 Vector 对象或者 Object[]数组作为构造器参数，其中使用 JComboBox 创建的列表框必须单击右边的向下箭头才会出现。

② 调用 JList 或 JComboBox 的各种方法来设置列表框的外观行为，其中 JList 可以调用如下几个常用的方法。

- addSelectionInterval(int anchor, int lead)：在已经选中列表项的基础上增加选中从 anchor 到 lead 索引范围内的所有列表项。
- setFixedCellHeight、setFixedCellWidth：设置每个列表项具有指定的高度和宽度。
- setLayoutOrientation(int layoutOrientation)：设置列表框的布局方向，该属性可以接收三个值，即 JList.HORIZONTAL_WRAP、JList.VERTICAL_WRAP 和 JList.VERTICAL（默认），用于指定当列表框长度不足以显示所有的列表项时，列表框如何排列所有的列表项。
- setSelectedIndex(int index)：设置默认选择哪一个列表项。
- setSelectedIndices(int[] indices)：设置默认选择哪一批列表项（多个）。
- setSelectedValue(Object anObject, boolean shouldScroll)：设置选中哪个列表项的值，第二个参数决定是否滚动到选中项。
- setSelectionBackground(Color selectionBackground)：设置选中项的背景色。
- setSelectionForeground(Color selectionForeground)：设置选中项的前景色。
- setSelectionInterval(int anchor, int lead)：设置选中从 anchor 到 lead 索引范围内的所有列表项。
- setSelectionMode(int selectionMode)：设置选中模式。支持如下 3 个值。
- ListSelectionModel.SINGLE_SELECTION：每次只能选择一个列表项。在这种模式中，setSelectionInterval 和 addSelectionInterval 是等效的。
- ListSelectionModel.SINGLE_INTERVAL_SELECTION：每次只能选择一个连续区域。在此模式中，如果需要添加的区域没有与已选择区域相邻或重叠，则不能添加该区域。简而言之，在这种模式下每次可以选择多个列表项，但多个列表项必须处于连续状态。
- ListSelectionModel.MULTIPLE_INTERVAL_SELECTION：在此模式中，选择没有任何限制。该模式是默认设置。
- setVisibleRowCount(int visibleRowCount)：设置该列表框的可视高度足以显示多少项。

JComboBox 则提供了如下几个常用方法。

- setEditable(boolean aFlag)：设置是否允许直接修改 JComboBox 文本框的值，默认不允许。
- setMaximumRowCount(int count)：设置下拉列表框的可视高度可以显示多少个列表项。

- setSelectedIndex(int anIndex)：根据索引设置默认选中哪一个列表项。
- setSelectedItem(Object anObject)：根据列表项的值设置默认选中哪一个列表项。

提示：

JComboBox 没有设置选择模式的方法，因为 JComboBox 最多只能选中一项，所以没有必要设置选择模式。

③ 如果需要监听列表框选择项的变化，则可以通过添加对应的监听器来实现。通常 JList 使用 addListSelectionListener()方法添加监听器，而 JComboBox 采用 addItemListener()方法添加监听器。

下面程序示范了 JList 和 JCombox 的用法，并允许用户通过单选按钮来控制 JList 的选项布局、选择模式，在用户选择图书之后，这些图书会在窗口下面的文本域里显示出来。

程序清单：codes\12\12.9\ListTest.java

```
public class ListTest
{
    private JFrame mainWin = new JFrame("测试列表框");
    String[] books = new String[]
    {
        "疯狂 Java 讲义"
        , "轻量级 Java EE 企业应用实战"
        , "疯狂 Android 讲义"
        , "疯狂 Ajax 讲义"
        , "经典 Java EE 企业应用实战"
    };
    // 用一个字符串数组来创建一个 JList 对象
    JList<String> bookList = new JList<>(books);
    JComboBox<String> bookSelector;
    // 定义布局选择按钮所在的面板
    JPanel layoutPanel = new JPanel();
    ButtonGroup layoutGroup = new ButtonGroup();
    // 定义选择模式按钮所在的面板
    JPanel selectModePanel = new JPanel();
    ButtonGroup selectModeGroup = new ButtonGroup();
    JTextArea favoriate = new JTextArea(4 , 40);
    public void init()
    {
        // 设置 JList 的可视高度可同时显示 3 个列表项
        bookList.setVisibleRowCount(3);
        // 默认选中第 3 项到第 5 项（第 1 项的索引是 0）
        bookList.setSelectionInterval(2, 4);
        addLayoutButton("纵向滚动", JList.VERTICAL);
        addLayoutButton("纵向换行", JList.VERTICAL_WRAP);
        addLayoutButton("横向换行", JList.HORIZONTAL_WRAP);
        addSelectModelButton("无限制", ListSelectionModel
            .MULTIPLE_INTERVAL_SELECTION);
        addSelectModelButton("单选", ListSelectionModel
            .SINGLE_SELECTION);
        addSelectModelButton("单范围", ListSelectionModel
            .SINGLE_INTERVAL_SELECTION);
        Box listBox = new Box(BoxLayout.Y_AXIS);
        // 将 JList 组件放在 JScrollPane 中，再将该 JScrollPane 添加到 listBox 容器中
        listBox.add(new JScrollPane(bookList));
        // 添加布局选择按钮面板、选择模式按钮面板
        listBox.add(layoutPanel);
        listBox.add(selectModePanel);
        // 为 JList 添加事件监听器
        bookList.addListSelectionListener(e -> {  // ①
            // 获取用户所选择的所有图书
            List<String> books = bookList.getSelectedValuesList();
            favoriate.setText("");
            for (String book : books )
            {
                favoriate.append(book + "\n");
            }
```

```
        });
        Vector<String> bookCollection = new Vector<>();
        bookCollection.add("疯狂 Java 讲义");
        bookCollection.add("轻量级 Java EE 企业应用实战");
        bookCollection.add("疯狂 Android 讲义");
        bookCollection.add("疯狂 Ajax 讲义");
        bookCollection.add("经典 Java EE 企业应用实战");
        // 用一个 Vector 对象来创建一个 JComboBox 对象
        bookSelector = new JComboBox<>(bookCollection);
        // 为 JComboBox 添加事件监听器
        bookSelector.addItemListener(e -> {  // ②
            // 获取 JComboBox 所选中的项
            Object book = bookSelector.getSelectedItem();
            favoriate.setText(book.toString());
        });
        // 设置可以直接编辑
        bookSelector.setEditable(true);
        // 设置下拉列表框的可视高度可同时显示 4 个列表项
        bookSelector.setMaximumRowCount(4);
        JPanel p = new JPanel();
        p.add(bookSelector);
        Box box = new Box(BoxLayout.X_AXIS);
        box.add(listBox);
        box.add(p);
        mainWin.add(box);
        JPanel favoriatePanel = new JPanel();
        favoriatePanel.setLayout(new BorderLayout());
        favoriatePanel.add(new JScrollPane(favoriate));
        favoriatePanel.add(new JLabel("您喜欢的图书：")
            , BorderLayout.NORTH);
        mainWin.add(favoriatePanel , BorderLayout.SOUTH);
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.pack();
        mainWin.setVisible(true);
    }
    private void addLayoutButton(String label, final int orientation)
    {
        layoutPanel.setBorder(new TitledBorder(new EtchedBorder()
            , "确定选项布局"));
        JRadioButton button = new JRadioButton(label);
        // 把该单选按钮添加到 layoutPanel 面板中
        layoutPanel.add(button);
        // 默认选中第一个按钮
        if(layoutGroup.getButtonCount() == 0)
            button.setSelected(true);
        layoutGroup.add(button);
        button.addActionListener(event ->
            // 改变列表框里列表项的布局方向
            bookList.setLayoutOrientation(orientation));
    }
    private void addSelectModelButton(String label, final int selectModel)
    {
        selectModePanel.setBorder(new TitledBorder(new EtchedBorder()
            , "确定选择模式"));
        JRadioButton button = new JRadioButton(label);
        // 把该单选按钮添加到 selectModePanel 面板中
        selectModePanel.add(button);
        // 默认选中第一个按钮
        if (selectModeGroup.getButtonCount() == 0)
        button.setSelected(true);
        selectModeGroup.add(button);
        button.addActionListener(event ->
            // 改变列表框里的选择模式
            bookList.setSelectionMode(selectModel));
    }
    public static void main(String[] args)
    {
        new ListTest().init();
    }
}
```

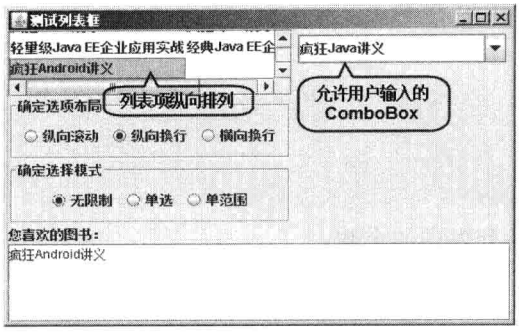


图 12.34 JList 和 JComboBox 的用法示范

上面程序中的粗体字代码实现了使用字符串数组创建一个 JList 对象，并通过调用一些方法来改变该 JList 的表现外观；使用 Vector 创建一个 JComboBox 对象，并通过调用一些方法来改变该 JComboBox 的表现外观。

程序中①②号粗体字代码为 JList 对象和 JComboBox 对象添加事件监听器，当用户改变两个列表框里的选择时，程序会把用户选择的图书显示在下面的文本域内。运行上面程序，会看到如图 12.34 所示的效果。

从图 12.34 中可以看出，因为 JComboBox 设置了 setEditable(true)，所以可以直接在该组件中输入用户自己喜欢的图书，当输入结束后，输入的图书名会直接显示在窗口下面的文本域内。

注意：

JList 默认没有滚动条，必须将其放在 JScrollPane 中才有滚动条，通常总是将 JList 放在 JScrollPane 中使用，所以程序中先将 JList 放到 JScrollPane 容器中，再将该 JScrollPane 添加到窗口中。要在 JList 中选中多个选项，可以使用 Ctrl 或 Shift 辅助键，按住 Ctrl 键才可以在原来选中的列表项基础上添加选中新的列表项；按 Shift 键可以选中连续区域的所有列表项。

12.9.2 不强制存储列表项的 ListModel 和 ComboBoxModel

正如前面提到的，Swing 的绝大部分组件都采用了 MVC 的设计模式，其中 JList 和 JComboBox 都只负责组件的外观显示，而组件底层的状态数据维护则由对应的 Model 负责。JList 对应的 Model 是 ListModel 接口，JComboBox 对应的 Model 是 ComboBoxModel 接口，这两个接口负责维护 JList 和 JComboBox 组件里的列表项。其中 ListModel 接口的代码如下：

```
public interface ListModel<E>
{
    // 返回列表项的数量
    int getSize();
    // 返回指定索引处的列表项
    E getElementAt(int index);
    // 为列表项添加一个监听器，当列表项发生变化时将触发该监听器
    void addListDataListener(ListDataListener l);
    // 删除列表项上的指定监听器
    void removeListDataListener(ListDataListener l);
}
```

从上面接口来看，这个 ListModel 不管 JList 里的所有列表项的存储形式，它甚至不强制存储所有的列表项，只要 ListModel 的实现类提供了 getSize()和 getElementAt()两个方法，JList 就可以根据该 ListModel 对象来生成列表框。

ComboBoxModel 继承了 ListModel，它添加了“选择项”的概念，选择项代表 JComboBox 显示区域内可见的列表项。ComboBoxModel 为“选择项”提供了两个方法，下面是 ComboBoxModel 接口的代码。

```
public interface ComboBoxModel<E> extends ListModel<E>
{
    // 设置选中“选择项”
    void setSelectedItem(Object anItem);
    // 获取“选择项”的值
    Object getSelectedItem();
}
```

因为 ListModel 不强制保存所有的列表项，因此可以为它创建一个实现类：NumberListModel，这个实现类只需要传入数字上限、数字下限和步长，程序就可以自动为之实现上面的 getSize()方法和

getElementAt()方法，从而允许直接使用一个数字范围来创建 JList 对象。

实现 getSize()方法的代码如下：

```
public int getSize()
{
    return (int)Math.floor(end.subtract(start)
        .divide(step).doubleValue()) + 1;
}
```

用“(上限–下限) ÷步长+1”即得到该 ListModel 中包含的列表项的个数。

注意：

程序使用 BigDecimal 变量来保存上限、下限和步长，而不是直接使用 double 变量来保存这三个属性，主要是为了实现对数值的精确计算，所以上面程序中的 end、start 和 step 都是 BigDecimal 类型的变量。

实现 getElementAt()方法也很简单，“下限+步长×索引”就是指定索引处的元素，该方法的具体实现请参考 ListModelTest.java。

下面程序为 ListModel 提供了 NumberListModel 实现类，并为 ComboBoxModel 提供了 NumberComboBoxModel 实现类，这两个实现类允许程序使用数值范围来创建 JList 和 JComboBox 对象。

程序清单：codes\12\12.9\ListModelTest.java

```
public class ListModelTest
{
    private JFrame mainWin = new JFrame("测试 ListModel");
    // 根据 NumberListModel 对象来创建一个 JList 对象
    private JList<BigDecimal> numScopeList = new JList<>(
        new NumberListModel(1 , 21 , 2));
    // 根据 NumberComboBoxModel 对象来创建 JComboBox 对象
    private JComboBox<BigDecimal> numScopeSelector = new JComboBox<>(
        new NumberComboBoxModel(0.1 , 1.2 , 0.1));
    private JTextField showVal = new JTextField(10);
    public void init()
    {
        // JList 的可视高度可同时显示 4 个列表项
        numScopeList.setVisibleRowCount(4);
        // 默认选中第 3 项到第 5 项（第 1 项的索引是 0）
        numScopeList.setSelectionInterval(2, 4);
        // 设置每个列表项具有指定的高度和宽度
        numScopeList.setFixedCellHeight(30);
        numScopeList.setFixedCellWidth(90);
        // 为 numScopeList 添加监听器
        numScopeList.addListSelectionListener(e -> {
            // 获取用户所选中的所有数字
            List<BigDecimal> nums = numScopeList.getSelectedValuesList();
            showVal.setText("");
            // 把用户选中的数字添加到单行文本框中
            for (BigDecimal num : nums )
            {
                showVal.setText(showVal.getText()
                    + num.toString() + ", ");
            }
        });
        // 设置列表项的可视高度可显示 5 个列表项
        numScopeSelector.setMaximumRowCount(5);
        Box box = new Box(BoxLayout.X_AXIS);
        box.add(new JScrollPane(numScopeList));
        JPanel p = new JPanel();
        p.add(numScopeSelector);
        box.add(p);
        // 为 numScopeSelector 添加监听器
        numScopeSelector.addItemListener(e -> {
            // 获取 JComboBox 中选中的数字
```

```
            Object num = numScopeSelector.getSelectedItem();
            showVal.setText(num.toString());
        });
        JPanel bottom = new JPanel();
        bottom.add(new JLabel("您选择的值是："));
        bottom.add(showVal);
        mainWin.add(box);
        mainWin.add(bottom , BorderLayout.SOUTH);
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.pack();
        mainWin.setVisible(true);
    }
    public static void main(String[] args)
    {
        new ListModelTest().init();
    }
}
class NumberListModel extends AbstractListModel<BigDecimal>
{
    protected BigDecimal start;
    protected BigDecimal end;
    protected BigDecimal step;
    public NumberListModel(double start
        , double end , double step)
    {
        this.start = BigDecimal.valueOf(start);
        this.end = BigDecimal.valueOf(end);
        this.step = BigDecimal.valueOf(step);
    }
    // 返回列表项的个数
    public int getSize()
    {
        return (int)Math.floor(end.subtract(start)
            .divide(step).doubleValue()) + 1;
    }
    // 返回指定索引处的列表项
    public BigDecimal getElementAt(int index)
    {
        return BigDecimal.valueOf(index)
            .multiply(step).add(start);
    }
}
class NumberComboBoxModel extends NumberListModel
    implements ComboBoxModel<BigDecimal>
{
    // 用于保存用户选中项的索引
    private int selectId = 0;
    public NumberComboBoxModel(double start
        , double end , double step)
    {
        super(start , end , step);
    }
    // 设置选中“选择项”
    public void setSelectedItem(Object anItem)
    {
        if (anItem instanceof BigDecimal)
        {
            BigDecimal target = (BigDecimal)anItem;
            // 根据选中的值来修改选中项的索引
            selectId = target.subtract(super.start)
                .divide(step).intValue();
        }
    }
    // 获取“选择项”的值
    public BigDecimal getSelectedItem()
    {
        // 根据选中项的索引来取得选中项
        return BigDecimal.valueOf(selectId)
```

```
                .multiply(step).add(start);
        }
    }
```

上面程序中的粗体字代码分别使用 NumberListModel 和 NumberComboBoxModel 创建了一个 JList 和 JComboBox 对象，创建这两个列表框时无须指定每个列表项，只需给出数值的上限、下限和步长即可。运行上面程序，会看到如图 12.35 所示的窗口。

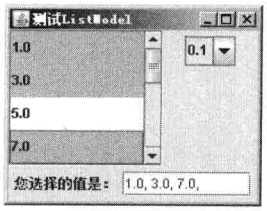


图 12.35　根据数值范围创建的 JList 和 JComboBox

12.9.3　强制存储列表项的 DefaultListModel 和 DefaultComboBoxModel

前面只是介绍了如何创建 JList、JComboBox 对象，当调用 JList 和 JComboBox 构造器时传入数组或 Vector 作为参数，这些数组元素或集合元素将会作为列表项。当使用 JList 或 JComboBox 时常常还需要动态地增加、删除列表项。

对于 JComboBox 类，它提供了如下几个方法来增加、插入和删除列表项。

- addItem(E anObject)：向 JComboBox 中的添加一个列表项。
- insertItemAt(E anObject, int index)：向 JComboBox 的指定索引处插入一个列表项。
- removeAllItems()：删除 JComboBox 中的所有列表项。
- removeItem(E anObject)：删除 JComboBox 中的指定列表项。
- removeItemAt(int anIndex)：删除 JComboBox 指定索引处的列表项。

提示：

上面这些方法的参数类型是 E，这是由于 Java 7 为 JComboBox、JList、ListModel 都增加了泛型支持，这些接口都有形如 JComboBox<E>、JList<E>、ListModel<E>的泛型声明，因此它们里面的方法可使用 E 作为参数或返回值的类型。

通过这些方法就可以增加、插入和删除 JComboBox 中的列表项，但 JList 并没有提供这些类似的方法。实际上，对于直接通过数组或 Vector 创建的 JList 对象，则很难向该 JList 中添加或删除列表项。如果需要创建一个可以增加、删除列表项的 JList 对象，则应该在创建 JList 时显式使用 DefaultListModel 作为构造参数。因为 DefaultListModel 作为 JList 的 Model，它负责维护 JList 组件的所有列表数据，所以可以通过向 DefaultListModel 中添加、删除元素来实现向 JList 对象中增加、删除列表项。DefaultListModel 提供了如下几个方法来添加、删除元素。

- add(int index, E element)：在该 ListModel 的指定位置处插入指定元素。
- addElement(E obj)：将指定元素添加到该 ListModel 的末尾。
- insertElementAt(E obj, int index)：在该 ListModel 的指定位置处插入指定元素。
- Object remove(int index)：删除该 ListModel 中指定位置处的元素。
- removeAllElements()：删除该 ListModel 中的所有元素，并将其的大小设置为零。
- removeElement(E obj)：删除该 ListModel 中第一个与参数匹配的元素。
- removeElementAt(int index)：删除该 ListModel 中指定索引处的元素。
- removeRange(int fromIndex, int toIndex)：删除该 ListModel 中指定范围内的所有元素。
- set(int index, E element)：将该 ListModel 指定索引处的元素替换成指定元素。
- setElementAt(E obj, int index)：将该 ListModel 指定索引处的元素替换成指定元素。

上面这些方法有些功能是重复的，这是由于 Java 的历史原因造成的。如果通过 DefaultListModel 来创建 JList 组件，则就可以通过调用上面的这些方法来添加、删除 DefaultListModel 中的元素，从而实现对 JList 里列表项的增加、删除。下面程序示范了如何向 JList 中添加、删除列表项。

程序清单：codes\12\12.9\DefaultListModelTest.java

```
public class DefaultListModelTest
```

```
{
    private JFrame mainWin = new JFrame("测试 DefaultListModel");
    // 定义一个 JList 对象
    private JList<String> bookList;
    // 定义一个 DefaultListModel 对象
    private DefaultListModel<String> bookModel
        = new DefaultListModel<>();
    private JTextField bookName = new JTextField(20);
    private JButton removeBn = new JButton("删除选中图书") ;
    private JButton addBn = new JButton("添加指定图书");
    public void init()
    {
        // 向 bookModel 中添加元素
        bookModel.addElement("疯狂 Java 讲义");
        bookModel.addElement("轻量级 Java EE 企业应用实战");
        bookModel.addElement("疯狂 Android 讲义");
        bookModel.addElement("疯狂 Ajax 讲义");
        bookModel.addElement("经典 Java EE 企业应用实战");
        // 根据 DefaultListModel 对象创建一个 JList 对象
        bookList = new JList<>(bookModel);
        // 设置最大可视高度
        bookList.setVisibleRowCount(4);
        // 只能单选
        bookList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        // 为添加按钮添加事件监听器
        addBn.addActionListener(evt -> {
            // 当 bookName 文本框的内容不为空时
            if (!bookName.getText().trim().equals(""))
            {
                // 向 bookModel 中添加一个元素
                // 系统会自动向 JList 中添加对应的列表项
                bookModel.addElement(bookName.getText());
            }
        });
        // 为删除按钮添加事件监听器
        removeBn.addActionListener(evt -> {
            // 如果用户已经选中一项
            if (bookList.getSelectedIndex() >= 0)
            {
                // 从 bookModel 中删除指定索引处的元素
                // 系统会自动删除 JList 对应的列表项
                bookModel.removeElementAt(bookList.getSelectedIndex());
            }
        });
        JPanel p = new JPanel();
        p.add(bookName);
        p.add(addBn);
        p.add(removeBn);
        // 添加 bookList 组件
        mainWin.add(new JScrollPane(bookList));
        // 将 p 面板添加到窗口中
        mainWin.add(p , BorderLayout.SOUTH);
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.pack();
        mainWin.setVisible(true);
    }
    public static void main(String[] args)
    {
        new DefaultListModelTest().init();
    }
}
```

上面程序中的粗体字代码通过一个 DefaultListModel 创建了一个 JList 对象，然后在两个按钮的事件监听器中分别向 DefaultListModel 对象中添加、删除元素，从而实现了向 JList 对象中添加、删除列表项。运行上面程序，会看到如图 12.36 所示的窗口。

图 12.36 向 JList 中添加、删除列表项

DefaultListModel 和 DefaultComboBoxModel 是两个强制保存所有列表项的 Model 类，它们使用 Vector 来保存所有的列表项。从 DefaultListModelTest 程序中可以看出，DefaultListModel 的用法和 Vector 的用法非常相似。实际上，DefaultListModel 和 DefaultComboBoxModel 从功能上来看，与一个 Vector 并没有太大的区别。如果要创建列表项可变的 JList 组件，使用 DefaultListModel 作为构造参数即可，读者可以把 DefaultListModel 当成一个特殊的 Vector；创建列表项可变的 JComboBox 组件，当然也可以显式使用 DefaultComboBoxModel 作为参数，但这并不是必需的，因为 JComboBox 默认使用 DefaultComboBoxModel 作为对应的 model 对象。

➤➤ 12.9.4　使用 ListCellRenderer 改变列表项外观

前面程序中的 JList 和 JComboBox 采用的都是简单的字符串列表项，实际上，JList 和 JComboBox 还可以支持图标列表项，如果在创建 JList 或 JComboBox 时传入图标数组，则创建的 JList 和 JComboBox 的列表项就是图标。

如果希望列表项是更复杂的组件，例如，希望像 QQ 程序那样每个列表项既有图标，也有字符串，那么可以通过调用 JList 或 JComboBox 的 setCellRenderer(ListCellRenderer cr)方法来实现，该方法需要接收一个 ListCellRenderer 对象，该对象代表一个列表项绘制器。

ListCellRenderer 是一个接口，该接口里包含一个方法：

```
public Component getListCellRendererComponent(JList list, Object value
    , int index, bolean isSelected, boolean cellHasFocus)
```

上面的 getListCellRendererComponent()方法返回一个 Component 组件，该组件就代表了 JList 或 JComboBox 的每个列表项。

注意：

自定义绘制 JList 和 JComboBox 的列表项所用的方法相同，所用的列表项绘制器也相同，故本节以 JList 为例。

ListCellRenderer 只是一个接口，它并未强制指定列表项绘制器属于哪种组件，因此可扩展任何组件来实现 ListCellRenderer 接口。通常采用扩展其他容器（如 JPanel）的方式来实现列表项绘制器，实现列表项绘制器时可通过重写 paintComponent()的方法来改变单元格的外观行为。例如下面程序，重写

paintComponent()方法时先绘制好友图像，再绘制好友名字。

程序清单：codes\12\12.9\ListRenderingTest.java

```
public class ListRenderingTest
{
    private JFrame mainWin = new JFrame("好友列表");
    private String[] friends = new String[]
    {
        "李清照",
        "苏格拉底",
        "李白",
        "弄玉",
        "虎头"
    };
    // 定义一个 JList 对象
    private JList friendsList = new JList(friends);
    public void init()
    {
        // 设置该 JList 使用 ImageCellRenderer 作为列表项绘制器
        friendsList.setCellRenderer(new ImageCellRenderer());
        mainWin.add(new JScrollPane(friendsList));
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.pack();
        mainWin.setVisible(true);
    }
    public static void main(String[] args)
    {
        new ListRenderingTest().init();
    }
}
class ImageCellRenderer extends JPanel
    implements ListCellRenderer
{
    private ImageIcon icon;
    private String name;
    // 定义绘制单元格时的背景色
    private Color background;
    // 定义绘制单元格时的前景色
    private Color foreground;
    public Component getListCellRendererComponent(JList list
        , Object value , int index
        , boolean isSelected , boolean cellHasFocus)
    {
        icon = new ImageIcon("ico/" + value + ".gif");
        name = value.toString();
        background = isSelected ? list.getSelectionBackground()
            : list.getBackground();
        foreground = isSelected ? list.getSelectionForeground()
            : list.getForeground();
        // 返回该 JPanel 对象作为列表项绘制器
        return this;
    }
    // 重写 paintComponent()方法，改变 JPanel 的外观
    public void paintComponent(Graphics g)
    {
        int imageWidth = icon.getImage().getWidth(null);
        int imageHeight = icon.getImage().getHeight(null);
        g.setColor(background);
        g.fillRect(0, 0, getWidth(), getHeight());
        g.setColor(foreground);
        // 绘制好友图标
        g.drawImage(icon.getImage() , getWidth() / 2
            - imageWidth / 2 , 10 , null);
        g.setFont(new Font("SansSerif" , Font.BOLD , 18));
        // 绘制好友用户名
        g.drawString(name, getWidth() / 2
            - name.length() * 10 , imageHeight + 30 );
    }
```

```
        // 通过该方法来设置该 ImageCellRenderer 的最佳大小
        public Dimension getPreferredSize()
        {
            return new Dimension(60, 80);
        }
    }
```

上面程序中的粗体字代码显式指定了该 JList 对象使用 ImageCellRenderer 作为列表项绘制器，ImageCellRenderer 重写了 paintComponent()方法来绘制单元格内容。除此之外，ImageCellRenderer 还重写了 getPreferredSize()方法，该方法返回一个 Dimension 对象，用于描述该列表项绘制器的最佳大小。运行上面程序，会看到如图 12.37 所示的窗口。

图 12.37　使用 ListCellRenderer 绘制列表项

通过使用自定义的列表项绘制器，可以让 JList 和 JComboBox 的列表项是任意组件，并且可以在该组件上任意添加内容。

12.10　使用 JTree 和 TreeModel 创建树

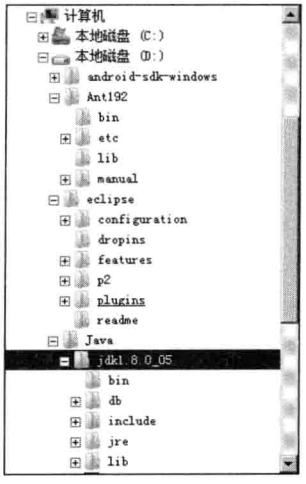


图 12.38　Windows 资源管理器目录树

树也是图形用户界面中使用非常广泛的 GUI 组件，例如使用 Windows 资源管理器时，将看到如图 12.38 所示的目录树。

如图 12.38 所示的树，代表计算机世界里的树，它从自然界实际的树抽象而来。计算机世界里的树是由一系列具有严格父子关系的节点组成的，每个节点既可以是其上一级节点的子节点，也可以是其下一级节点的父节点，因此同一个节点既可以是父节点，也可以是子节点（类似于一个人，他既是他儿子的父亲，又是他父亲的儿子）。

如果按节点是否包含子节点来分，节点分为如下两种。

- 普通节点：包含子节点的节点。
- 叶子节点：没有子节点的节点，因此叶子节点不可作为父节点。

如果按节点是否具有唯一的父节点来分，节点又可分为如下两种。

- 根节点：没有父节点的节点，根节点不可作为子节点。
- 普通节点：具有唯一父节点的节点。

一棵树只能有一个根节点，如果一棵树有了多个根节点，那它就不是一棵树了，而是多棵树的集合，有时也被称为森林。图 12.39 显示了计算机世界里树的一些专业术语。

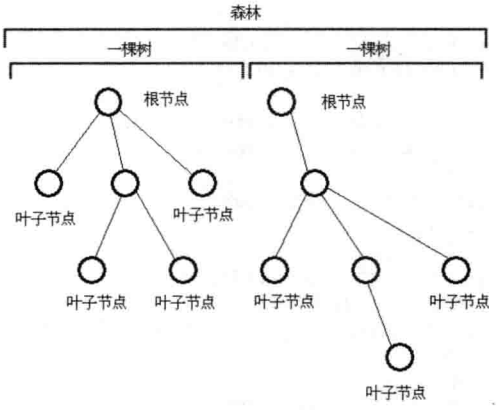


图 12.39　计算机世界里树的示意图

使用 Swing 里的 Jtree、TreeModel 及其相关的辅助类可以很轻松地开发出计算机世界里的树，如图 12.39 所示。

12.10.1 创建树

Swing 使用 JTree 对象来代表一棵树（实际上，JTree 可以代表森林，因为在使用 JTree 创建树时可以传入多个根节点），JTree 树中节点可以使用 TreePath 来标识，该对象封装了当前节点及其所有的父节点。必须指出，节点及其所有的父节点才能唯一地标识一个节点；也可以使用行数来标识，如图 12.39 所示，显示区域的每一行都标识一个节点。

当一个节点具有子节点时，该节点有两种状态。

- 展开状态：当父节点处于展开状态时，其子节点是可见的。
- 折叠状态：当父节点处于折叠状态时，其子节点都是不可见的。

如果某个节点是可见的，则该节点的父节点（包括直接的、间接的父节点）都必须处于展开状态，只要有任意一个父节点处于折叠状态，该节点就是不可见的。

如果希望创建一棵树，则直接使用 JTree 的构造器创建 JTree 对象即可。JTree 提供了如下几个常用构造器。

- JTree(TreeModel newModel)：使用指定的数据模型创建 JTree 对象，它默认显示根节点。
- JTree(TreeNode root)：使用 root 作为根节点创建 JTree 对象，它默认显示根节点。
- JTree(TreeNode root, boolean asksAllowsChildren)：使用 root 作为根节点创建 JTree 对象，它默认显示根节点。asksAllowsChildren 参数控制怎样的节点才算叶子节点，如果该参数为 true，则只有当程序使用 setAllowsChildren(false)显式设置某个节点不允许添加子节点时（以后也不会拥有子节点），该节点才会被 JTree 当成叶子节点；如果该参数为 false，则只要某个节点当时没有子节点（不管以后是否拥有子节点），该节点都会被 JTree 当成叶子节点。

上面的第一个构造器需要显式传入一个 TreeModel 对象，Swing 为 TreeModel 提供了一个 DefaultTreeModel 实现类，通常可先创建 DefaultTreeModel 对象，然后利用 DefaultTreeModel 来创建 JTree，但通过 DefaultTreeModel 的 API 文档会发现，创建 DefaultTreeModel 对象依然需要传入根节点，所以直接通过根节点创建 JTree 更加简洁。

为了利用根节点来创建 JTree，程序需要创建一个 TreeNode 对象。TreeNode 是一个接口，该接口有一个 MutableTreeNode 子接口，Swing 为该接口提供了默认的实现类：DefaultMutableTreeNode，程序可以通过 DefaultMutableTreeNode 来为树创建节点，并通过 DefaultMutableTreeNode 提供的 add()方法建立各节点之间的父子关系，然后调用 JTree 的 JTree(TreeNode root)构造器来创建一棵树。

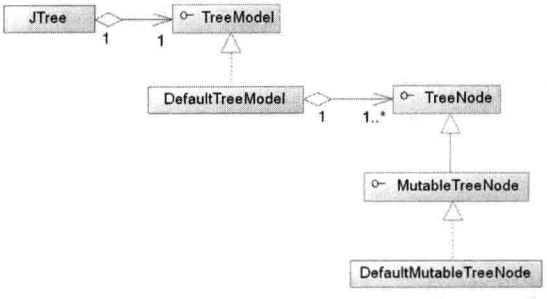


图 12.40 JTree 相关类的关系

图 12.40 显示了 JTree 相关类的关系，从该图可以看出 DefaultTreeModel 是 TreeModel 的默认实现类，当程序通过 TreeNode 类创建 JTree 时，其状态数据实际上由 DefaultTreeModel 对象维护，因为创建 JTree 时传入的 TreeNode 对象，实际上传给了 DefaultTreeModel 对象。

提示： DefaultTreeModel 也提供了 DefaultTreeModel(TreeNode root)构造器，用于接收一个 TreeNode 根节点来创建一个默认的 TreeModel 对象；当程序中通过传入一个根节点来创建 JTree 对象时，实际上是将该节点传入对应的 DefaultTreeModel 对象，并使用该 DefaultTreeModel 对象来创建 JTree 对象。

下面程序创建了一棵最简单的 Swing 树。

程序清单：codes\12\12.10\SimpleJTree.java

```
public class SimpleJTree
{
    JFrame jf = new JFrame("简单树");
    JTree tree;
    DefaultMutableTreeNode root;
    DefaultMutableTreeNode guangdong;
    DefaultMutableTreeNode guangxi;
    DefaultMutableTreeNode foshan;
    DefaultMutableTreeNode shantou;
    DefaultMutableTreeNode guilin;
    DefaultMutableTreeNode nanning;
    public void init()
    {
        // 依次创建树中的所有节点
        root = new DefaultMutableTreeNode("中国");
        guangdong = new DefaultMutableTreeNode("广东");
        guangxi = new DefaultMutableTreeNode("广西");
        foshan = new DefaultMutableTreeNode("佛山");
        shantou = new DefaultMutableTreeNode("汕头");
        guilin = new DefaultMutableTreeNode("桂林");
        nanning = new DefaultMutableTreeNode("南宁");
        // 通过 add()方法建立树节点之间的父子关系
        guangdong.add(foshan);
        guangdong.add(shantou);
        guangxi.add(guilin);
        guangxi.add(nanning);
        root.add(guangdong);
        root.add(guangxi);
        // 以根节点创建树
        tree = new JTree(root);   //①
        jf.add(new JScrollPane(tree));
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new SimpleJTree().init();
    }
}
```

上面程序中的粗体字代码创建了一系列的DefaultMutableTreeNode对象，并通过 add()方法为这些节点建立了相应的父子关系。程序中①号粗体字代码则以一个根节点创建了一个 JTree 对象。当程序把 JTree 对象添加到其他容器中后，JTree 就会在该容器中绘制出一棵 Swing 树。运行上面程序，会看到如图 12.41 所示的窗口。

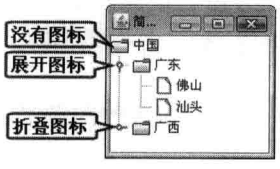


图 12.41　Swing 树的效果

从图 12.41 中可以看出，Swing 树的默认风格是使用一个特殊图标来表示节点的展开、折叠，而不是使用我们熟悉的“+”、“-”图标来表示节点的展开、折叠。如果希望使用“+”、“-”图标来表示节点的展开、折叠，则可以考虑使用 Windows 风格。

从图 12.41 中可以看出，Swing 树默认使用连接线来连接所有节点，程序可以使用如下代码来强制 JTree 不显示节点之间的连接线。

```
// 没有连接线
tree.putClientProperty("JTree.lineStyle" , "None");
```

或者使用如下代码来强制节点之间只有水平分隔线。

```
// 水平分隔线
tree.putClientProperty("JTree.lineStyle" , "Horizontal");
```

图 12.41 中显示的根节点前没有绘制表示节点展开、折叠的特殊图标，如果希望根节点也绘制表示节点展开、折叠的特殊图标，则使用如下代码。

```
// 设置是否显示根节点的“展开、折叠”图标，默认是 false
tree.setShowsRootHandles(true);
```

JTree 甚至允许把整个根节点都隐藏起来，可以通过如下代码来隐藏根节点。

```
// 设置根节点是否可见，默认是 true
tree.setRootVisible(false);
```

DefaultMutableTreeNode 是 JTree 默认的树节点，该类提供了大量的方法来访问树中的节点，包括遍历该节点的所有子节点的两个方法。DefaultMutableTreeNode 提供了深度优先遍历、广度优先遍历两种方法。

- Enumeration breadthFirstEnumeration()/preorderEnumeration()：按广度优先的顺序遍历以此节点为根的子树，并返回所有节点组成的枚举对象。
- Enumeration depthFirstEnumeration()/postorderEnumeration()：按深度优先的顺序遍历以此节点为根的子树，并返回所有节点组成的枚举对象。

提示：关于树的深度优先和广度优先遍历算法已经不属于本书的介绍范围，读者可以参考《疯狂 Java 程序员的基本修养》学习有关树的更详细内容。

除此之外，DefaultMutableTreeNode 也提供了大量的方法来获取指定节点的兄弟节点、父节点、子节点等，常用的有如下几个方法。

- DefaultMutableTreeNode getNextSibling()：返回此节点的下一个兄弟节点。
- TreeNode getParent()：返回此节点的父节点。如果此节点没有父节点，则返回 null。
- TreeNode[] getPath()：返回从根节点到达此节点的所有节点组成的数组。
- DefaultMutableTreeNode getPreviousSibling()：返回此节点的上一个兄弟节点。
- TreeNode getRoot()：返回包含此节点的树的根节点。
- TreeNode getSharedAncestor(DefaultMutableTreeNode aNode)：返回此节点和 aNode 最近的共同祖先。
- int getSiblingCount()：返回此节点的兄弟节点数。
- boolean isLeaf()：返回该节点是否是叶子节点。
- boolean isNodeAncestor(TreeNode anotherNode)：判断 anotherNode 是否是当前节点的祖先节点（包括父节点）。
- boolean isNodeChild(TreeNode aNode)：如果 aNode 是此节点的子节点，则返回 true。
- boolean isNodeDescendant(DefaultMutableTreeNode anotherNode)：如果 anotherNode 是此节点的后代，包括是此节点本身、此节点的子节点或此节点的子节点的后代，都将返回 true。
- boolean isNodeRelated(DefaultMutableTreeNode aNode)：当 aNode 和当前节点位于同一棵树中时返回 true。
- boolean isNodeSibling(TreeNode anotherNode)：返回 anotherNode 是否是当前节点的兄弟节点。
- boolean isRoot()：返回当前节点是否是根节点。
- Enumeration pathFromAncestorEnumeration(TreeNode ancestor)：返回从指定祖先节点到当前节点的所有节点组成的枚举对象。

12.10.2 拖动、编辑树节点

JTree 生成的树默认是不可编辑的，不可以添加、删除节点，也不可以改变节点数据；如果想让某个 JTree 对象变成可编辑状态，则可以调用 JTree 的 setEditable(boolean b)方法，传入 true 即可把这棵树变成可编辑的树（可以添加、删除节点，也可以改变节点数据）。

一旦将 JTree 对象设置成可编辑状态后，程序就可以为指定节点添加子节点、兄弟节点，也可以修改、删除指定节点。

前面简单提到过，JTree 处理节点有两种方式：一种是根据 TreePath；另一种是根据节点的行号，所有 JTree 显示的节点都有一个唯一的行号（从 0 开始）。只有那些被显示出来的节点才有行号，这就带来一个潜在的问题——如果该节点之前的节点被展开、折叠或增加、删除后，那么该节点的行号就会发生变化，因此通过行号来识别节点可能有一些不确定的地方；相反，使用 TreePath 来识别节点则会更加稳定。

可以使用文件系统来类比 JTree，从图 12.38 中可以看出，实际上所有的文件系统都采用树状结构，其中 Windows 的文件系统是森林，因为 Windows 包含 C、D 等多个根路径，而 UNIX、Linux 的文件系统是一棵树，只有一个根路径。如果直接给出 abc 文件夹（类似于 JTree 中的节点），系统不能准确地定位该路径；如果给出 D:\xyz\abc，系统就可以准确地定位到该路径，这个 D:\xyz\abc 实际上由三个文件夹组成：D:、xyz、abc，其中 D:是该路径的根路径。类似地，TreePath 也采用这种方式来唯一地标识节点。

TreePath 保持着从根节点到指定节点的所有节点，TreePath 由一系列节点组成，而不是单独的一个节点。JTree 的很多方法都用于返回一个 TreePath 对象，当程序得到一个 TreePath 后，可能只需要获取最后一个节点，则可以调用 TreePath 的 getLastPathComponent()方法。例如需要获得 JTree 中被选定的节点，则可以通过如下两行代码来实现。

```
// 获取选中节点所在的 TreePath
TreePath path = tree.getSelectionPath();
// 获取指定 TreePath 的最后一个节点
TreeNode target = (TreeNode)path.getLastPathComponent();
```

又因为 JTree 经常需要查询被选中的节点，所以 JTree 提供了一个 getLastSelectedPathComponent()方法来获取选中的节点。比如采用下面代码也可以获取选中的节点。

```
// 获取选中的节点
TreeNode target = (TreeNode) tree.getLastSelectedPathComponent();
```

可能有读者对上面这行代码感到奇怪，getLastSelectedPathComponent()方法返回的不是 TreeNode 吗？getLastSelectedPathComponent()方法返回的不一定是 TreeNode，该方法的返回值是 Object。因为 Swing 把 JTree 设计得非常复杂，JTree 把所有的状态数据都交给 TreeModel 管理，而 JTree 本身并没有与 TreeNode 发生关联（从图 12.40 可以看出这一点），只是因为 DefaultTreeModel 需要 TreeNode 而已，如果开发者自己提供一个 TreeModel 实现类，这个 TreeModel 实现类完全可以与 TreeNode 没有任何关系。当然，对于大部分 Swing 开发者而言，无须理会 JTree 的这些过于复杂的设计。

如果已经有了从根节点到当前节点的一系列节点所组成的节点数组，也可以通过 TreePath 提供的构造器将这些节点转换成 TreePath 对象，如下代码所示。

```
// 将一个节点数组转换成 TreePath 对象
TreePath tp = new TreePath(nodes);
```

获取了选中的节点之后，即可通过 DefaultTreeModel（它是 Swing 为 TreeModel 提供的唯一一个实现类）提供的一系列方法来插入、删除节点。DefaultTreeModel 类有一个非常优秀的设计，当使用 DefaultTreeModel 插入、删除节点后，该 DefaultTreeModel 会自动通知对应的 JTree 重绘所有节点，用户可以立即看到程序所做的修改。

也可以直接通过 TreeNode 提供的方法来添加、删除和修改节点，但通过 TreeNode 改变节点时，程序必须显式调用 JTree 的 updateUI()通知 JTree 重绘所有节点，让用户看到程序所做的修改。

下面程序实现了增加、修改和删除节点的功能，并允许用户通过拖动将一个节点变成另一个节点的子节点。

程序清单：codes\12\12.10\EditJTree.java

```
public class EditJTree
{
    JFrame jf;
    JTree tree;
    // 上面 JTree 对象对应的 model
```

```
DefaultTreeModel model;
// 定义几个初始节点
DefaultMutableTreeNode root = new DefaultMutableTreeNode("中国");
DefaultMutableTreeNode guangdong = new DefaultMutableTreeNode("广东");
DefaultMutableTreeNode guangxi = new DefaultMutableTreeNode("广西");
DefaultMutableTreeNode foshan = new DefaultMutableTreeNode("佛山");
DefaultMutableTreeNode shantou = new DefaultMutableTreeNode("汕头");
DefaultMutableTreeNode guilin = new DefaultMutableTreeNode("桂林");
DefaultMutableTreeNode nanning = new DefaultMutableTreeNode("南宁");
// 定义需要被拖动的 TreePath
TreePath movePath;
JButton addSiblingButton = new JButton("添加兄弟节点");
JButton addChildButton = new JButton("添加子节点");
JButton deleteButton = new JButton("删除节点");
JButton editButton = new JButton("编辑当前节点");
public void init()
{
    guangdong.add(foshan);
    guangdong.add(shantou);
    guangxi.add(guilin);
    guangxi.add(nanning);
    root.add(guangdong);
    root.add(guangxi);
    jf = new JFrame("可编辑节点的树");
    tree = new JTree(root);
    // 获取 JTree 对应的 TreeModel 对象
    model = (DefaultTreeModel)tree.getModel();
    // 设置 JTree 可编辑
    tree.setEditable(true);
    MouseListener ml = new MouseAdapter()
    {
        // 按下鼠标时获得被拖动的节点
        public void mousePressed(MouseEvent e)
        {
            // 如果需要唯一确定某个节点，则必须通过 TreePath 来获取
            TreePath tp = tree.getPathForLocation(
                e.getX() , e.getY());
            if (tp != null)
            {
                movePath = tp;
            }
        }
        // 松开鼠标时获得需要拖到哪个父节点
        public void mouseReleased(MouseEvent e)
        {
            // 根据松开鼠标时的 TreePath 来获取 TreePath
            TreePath tp = tree.getPathForLocation(
                e.getX(), e.getY());
            if (tp != null && movePath != null)
            {
                // 阻止向子节点拖动
                if (movePath.isDescendant(tp) && movePath != tp)
                {
                    JOptionPane.showMessageDialog(jf,
                        "目标节点是被移动节点的子节点，无法移动！",
                        "非法操作", JOptionPane.ERROR_MESSAGE );
                    return;
                }
                // 不是向子节点移动，鼠标按下、松开的也不是同一个节点
                else if (movePath != tp)
                {
                    // add 方法先将该节点从原父节点下删除，再添加到新父节点下
                    ((DefaultMutableTreeNode)tp.getLastPathComponent())
                        .add((DefaultMutableTreeNode)movePath
                        .getLastPathComponent());
                    movePath = null;
                    tree.updateUI();
                }
            }
        }
```

```
};
// 为 JTree 添加鼠标监听器
tree.addMouseListener(ml);
JPanel panel = new JPanel();
// 实现添加兄弟节点的监听器
addSiblingButton.addActionListener(event -> {
    // 获取选中的节点
    DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode)
        tree.getLastSelectedPathComponent();
    // 如果节点为空，则直接返回
    if (selectedNode == null) return;
    // 获取该选中节点的父节点
    DefaultMutableTreeNode parent = (DefaultMutableTreeNode)
        selectedNode.getParent();
    // 如果父节点为空，则直接返回
    if (parent == null) return;
    // 创建一个新节点
    DefaultMutableTreeNode newNode = new
        DefaultMutableTreeNode("新节点");
    // 获取选中节点的选中索引
    int selectedIndex = parent.getIndex(selectedNode);
    // 在选中位置插入新节点
    model.insertNodeInto(newNode, parent, selectedIndex + 1);
    // --------下面代码实现显示新节点（自动展开父节点）-------
    // 获取从根节点到新节点的所有节点
    TreeNode[] nodes = model.getPathToRoot(newNode);
    // 使用指定的节点数组来创建 TreePath
    TreePath path = new TreePath(nodes);
    // 显示指定的 TreePath
    tree.scrollPathToVisible(path);
});
panel.add(addSiblingButton);
// 实现添加子节点的监听器
addChildButton.addActionListener(event -> {
    // 获取选中的节点
    DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode)
        tree.getLastSelectedPathComponent();
    // 如果节点为空，则直接返回
    if (selectedNode == null) return;
    // 创建一个新节点
    DefaultMutableTreeNode newNode = new
        DefaultMutableTreeNode("新节点");
    // 通过 model 来添加新节点，则无须调用 JTree 的 updateUI 方法
    // model.insertNodeInto(newNode, selectedNode
    //     , selectedNode.getChildCount());
    // 通过节点添加新节点，则需要调用 tree 的 updateUI 方法
    selectedNode.add(newNode);
    // --------下面代码实现显示新节点（自动展开父节点）-------
    TreeNode[] nodes = model.getPathToRoot(newNode);
    TreePath path = new TreePath(nodes);
    tree.scrollPathToVisible(path);
    tree.updateUI();
});
panel.add(addChildButton);
// 实现删除节点的监听器
deleteButton.addActionListener(event -> {
    DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode)
        tree.getLastSelectedPathComponent();
    if (selectedNode != null && selectedNode.getParent() != null)
    {
        // 删除指定节点
        model.removeNodeFromParent(selectedNode);
    }
});
panel.add(deleteButton);
// 实现编辑节点的监听器
editButton.addActionListener(event -> {
    TreePath selectedPath = tree.getSelectionPath();
    if (selectedPath != null)
    {
        // 编辑选中的节点
```

```
                tree.startEditingAtPath(selectedPath);
            }
        });
        panel.add(editButton);
        jf.add(new JScrollPane(tree));
        jf.add(panel , BorderLayout.SOUTH);
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new EditJTree().init();
    }
}
```

上面程序中实现拖动节点也比较容易——当用户按下鼠标时获取鼠标事件发生位置的树节点，并把该节点赋给 movePath 变量；当用户松开鼠标时获取鼠标事件发生位置的树节点，作为目标节点需要拖到的父节点，把 movePath 从原来的节点中删除，添加到新的父节点中即可（TreeNode 的 add()方法可以同时完成这两个操作）。程序中的粗体字代码是实现整个程序的关键代码，读者可以结合程序运行效果来研究该代码。运行上面程序，会看到如图 12.42 所示的效果。

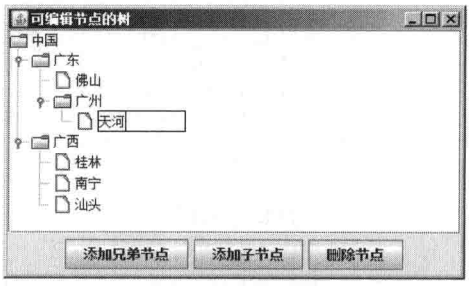


图 12.42 可以拖动、添加、删除节点的 Swing 树

选中图 12.42 中的某个节点并双击，或者单击“编辑当前节点”按钮，就可以进入该节点的编辑状态，系统启动默认的单元格编辑器来编辑该节点，JTree 的单元格编辑器与 JTable 的单元格编辑器都实现了相同的 CellEditor 接口。本书将在下一节与 JTable 一起介绍如何定制节点编辑器。

12.10.3 监听节点事件

JTree 专门提供了一个 TreeSelectionModel 对象来保存该 JTree 选中状态的信息。也就是说，JTree 组件背后隐藏了两个 model 对象，其中 TreeModel 用于保存该 JTree 的所有节点数据，而 TreeSelectionModel 用于保存该 JTree 的所有选中状态的信息。

提示：

对于大部分开发者而言，无须关心 TreeSelectionModel 的存在，程序可以通过 JTree 提供的 getSelectionPath()方法和 getSelectionPaths()方法来获取该 JTree 被选中的 TreePath，但实际上这两个方法底层实现依然依赖于 TreeSelectionModel，只是普通开发者一般无须关心这些底层细节而已。

程序可以改变 JTree 的选择模式，但必须先获取该 JTree 对应的 TreeSelectionModel 对象，再调用该对象的 setSelectionMode()方法来设置该 JTree 的选择模式。该方法支持如下三个参数。

- TreeSelectionModel.CONTINUOUS_TREE_SELECTION：可以连续选中多个 TreePath。
- TreeSelectionModel.DISCONTINUOUS_TREE_SELECTION：该选项对于选择没有任何限制。
- TreeSelectionModel.SINGLE_TREE_SELECTION：每次只能选择一个 TreePath。

与 JList 操作类似，按下 Ctrl 辅助键，用于添加选中多个 JTree 节点；按下 Shift 辅助键，用于选择连续区域里的所有 JTree 节点。

JTree 提供了如下两个常用的添加监听器的方法。

- addTreeExpansionListener(TreeExpansionListener tel)：添加树节点展开/折叠事件的监听器。
- addTreeSelectionListener(TreeSelectionListener tsl)：添加树节点选择事件的监听器。

下面程序设置 JTree 只能选择单个 TreePath，并为节点选择事件添加事件监听器。

程序清单：codes\12\12.10\SelectJTree.java

```
public class SelectJTree
{
    JFrame jf = new JFrame("监听树的选择事件");
    JTree tree;
    // 定义几个初始节点
    DefaultMutableTreeNode root = new DefaultMutableTreeNode("中国");
    DefaultMutableTreeNode guangdong = new DefaultMutableTreeNode("广东");
    DefaultMutableTreeNode guangxi = new DefaultMutableTreeNode("广西");
    DefaultMutableTreeNode foshan = new DefaultMutableTreeNode("佛山");
    DefaultMutableTreeNode shantou = new DefaultMutableTreeNode("汕头");
    DefaultMutableTreeNode guilin = new DefaultMutableTreeNode("桂林");
    DefaultMutableTreeNode nanning = new DefaultMutableTreeNode("南宁");
    JTextArea eventTxt = new JTextArea(5 , 20);
    public void init()
    {
        // 通过 add()方法建立树节点之间的父子关系
        guangdong.add(foshan);
        guangdong.add(shantou);
        guangxi.add(guilin);
        guangxi.add(nanning);
        root.add(guangdong);
        root.add(guangxi);
        // 以根节点创建树
        tree = new JTree(root);
        // 设置只能选择一个 TreePath
        tree.getSelectionModel().setSelectionMode(
            TreeSelectionModel.SINGLE_TREE_SELECTION);
        // 添加监听树节点选择事件的监听器
        // 当 JTree 中被选择节点发生改变时，将触发该方法
        tree.addTreeSelectionListener(e -> {
            if (e.getOldLeadSelectionPath() != null)
                eventTxt.append("原选中的节点路径："
                + e.getOldLeadSelectionPath().toString() + "\n");
            eventTxt.append("新选中的节点路径："
                + e.getNewLeadSelectionPath().toString() + "\n");
        });        // 设置是否显示根节点的展开/折叠图标，默认是 false
        tree.setShowsRootHandles(true);
        // 设置根节点是否可见，默认是 true
        tree.setRootVisible(true);
        Box box = new Box(BoxLayout.X_AXIS);
        box.add(new JScrollPane(tree));
        box.add(new JScrollPane(eventTxt));
        jf.add(box);
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new SelectJTree().init();
    }
}
```

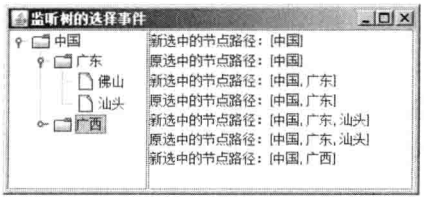


图 12.43　监听树的选择事件

上面程序中的第一行粗体字代码设置了该 JTree 对象采用 SINGLE_TREE_SELECTION 选择模式，即每次只能选中该 JTree 的一个 TreePath。第二段粗体字代码为该 JTree 添加了一个节点选择事件的监听器，当该 JTree 中被选择节点发生改变时，该监听器就会被触发。运行上面程序，会看到如图 12.43 所示的效果。

注意：

不要通过监听鼠标事件来监听所选节点的变化，因为 JTree 中节点的选择完全可以通过键盘来操作，不通过鼠标单击亦可。

12.10.4 使用 DefaultTreeCellRenderer 改变节点外观

对比图 12.38 和图 12.41 所示的两棵树，不难发现图 12.38 所示的树更美观，因为图 12.38 所示的树节点的图标非常丰富，而图 12.41 所示的树节点的图标太过于单一。

实际上，JTree 也可以改变树节点的外观，包括改变节点的图标、字体等，甚至可以自由绘制节点外观。为了改变树节点的外观，可以通过为树指定自己的 CellRenderer 来实现，JTree 默认使用 DefaultTreeCellRenderer 来绘制每个节点。通过查看 API 文档可以发现：DefaultTreeCellRenderer 是 JLabel 的子类，该 JLabel 包含了该节点的图标和文本。

改变树节点的外观样式，可以有如下三种方式。

- 使用 DefaultTreeCellRenderer 直接改变节点的外观，这种方式可以改变整棵树所有节点的字体、颜色和图标。
- 为 JTree 指定 DefaultTreeCellRenderer 的扩展类对象作为 JTree 的节点绘制器，该绘制器负责为不同节点使用不同的字体、颜色和图标。通常使用这种方式来改变节点的外观。
- 为 JTree 指定一个实现 TreeCellRenderer 接口的节点绘制器，该绘制器可以为不同的节点自由绘制任意内容，这是最复杂但最灵活的节点绘制器。

第一种方式最简单，但灵活性最差，因为它会改变整棵树所有节点的外观。在这种情况下，Jtree 的所有节点依然使用相同的图标，相当于整体替换了 Jtree 中节点的所有默认图标。用户指定的节点图标未必就比 JTree 默认的图标美观。

DefaultTreeCellRenderer 提供了如下几个方法来修改节点的外观。

- setBackgroundNonSelectionColor(Color newColor)：设置用于非选定节点的背景颜色。
- setBackgroundSelectionColor(Color newColor)：设置节点在选中状态下的背景颜色。
- setBorderSelectionColor(Color newColor)：设置选中状态下节点的边框颜色。
- setClosedIcon(Icon newIcon)：设置处于折叠状态下非叶子节点的图标。
- setFont(Font font)：设置节点文本的字体。
- setLeafIcon(Icon newIcon)：设置叶子节点的图标。
- setOpenIcon(Icon newIcon)：设置处于展开状态下非叶子节点的图标。
- setTextNonSelectionColor(Color newColor)：设置绘制非选中状态下节点文本的颜色。
- setTextSelectionColor(Color newColor)：设置绘制选中状态下节点文本的颜色。

下面程序直接使用 DefaultTreeCellRenderer 来改变树节点的外观。

程序清单：codes\12\12.10\ChangeAllCellRender.java

```
public class ChangeAllCellRender
{
    JFrame jf = new JFrame("改变所有节点的外观");
    JTree tree;
    // 定义几个初始节点
    DefaultMutableTreeNode root = new DefaultMutableTreeNode("中国");
    DefaultMutableTreeNode guangdong = new DefaultMutableTreeNode("广东");
    DefaultMutableTreeNode guangxi = new DefaultMutableTreeNode("广西");
    DefaultMutableTreeNode foshan = new DefaultMutableTreeNode("佛山");
    DefaultMutableTreeNode shantou = new DefaultMutableTreeNode("汕头");
    DefaultMutableTreeNode guilin = new DefaultMutableTreeNode("桂林");
    DefaultMutableTreeNode nanning = new DefaultMutableTreeNode("南宁");
    public void init()
    {
        // 通过 add()方法建立树节点之间的父子关系
        guangdong.add(foshan);
        guangdong.add(shantou);
        guangxi.add(guilin);
        guangxi.add(nanning);
        root.add(guangdong);
        root.add(guangxi);
        // 以根节点创建树
```

```
        tree = new JTree(root);
        // 创建一个 DefaultTreeCellRenderer 对象
        DefaultTreeCellRenderer cellRender = new DefaultTreeCellRenderer();
        // 设置非选定节点的背景颜色
        cellRender.setBackgroundNonSelectionColor(new
            Color(220 , 220 , 220));
        // 设置节点在选中状态下的背景颜色
        cellRender.setBackgroundSelectionColor(new Color(140 , 140, 140));
        // 设置选中状态下节点的边框颜色
        cellRender.setBorderSelectionColor(Color.BLACK);
        // 设置处于折叠状态下非叶子节点的图标
        cellRender.setClosedIcon(new ImageIcon("icon/close.gif"));
        // 设置节点文本的字体
        cellRender.setFont(new Font("SansSerif" , Font.BOLD , 16));
        // 设置叶子节点的图标
        cellRender.setLeafIcon(new ImageIcon("icon/leaf.png"));
        // 设置处于展开状态下非叶子节点的图标
        cellRender.setOpenIcon(new ImageIcon("icon/open.gif"));
        // 设置绘制非选中状态下节点文本的颜色
        cellRender.setTextNonSelectionColor(new Color(255 , 0 , 0));
        // 设置绘制选中状态下节点文本的颜色
        cellRender.setTextSelectionColor(new Color(0 , 0 , 255));
        tree.setCellRenderer(cellRender);
        // 设置是否显示根节点的展开/折叠图标，默认是 false
        tree.setShowsRootHandles(true);
        // 设置节点是否可见，默认是 true
        tree.setRootVisible(true);
        jf.add(new JScrollPane(tree));
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new ChangeAllCellRender().init();
    }
}
```

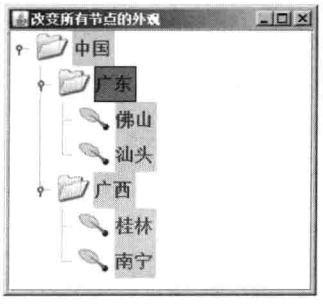


图 12.44　直接使用 DefaultTreeCellRenderer 改变所有节点的外观效果

上面程序中的粗体字代码创建了一个 DefaultTreeCellRenderer 对象，并通过该对象改变了 Jtree 中所有节点的字体、颜色和图标。运行上面程序，会看到如图 12.44 所示的效果。

从图 12.44 中可以看出，Jtree 中的所有节点全部被改变了，相当于完全替代了 Jtree 中所有节点的默认图标、字体和颜色。但所有的叶子节点依然保持相同的外观，所有的非叶子节点也保持相同的外观。这种改变依然不能满足更复杂的需求，例如，如果需要不同类型的节点呈现出不同的外观，则不能直接使用 DefaultTreeCellRenderer 来改变节点的外观，可以采用扩展 DefaultTreeCellRenderer 的方式来实现该需求。

提示：

不要试图通过 TreeCellRenderer 来改变表示节点展开/折叠的图标，因为该图标是由 Metal 风格决定的。如果需要改变该图标，则可以考虑改变该 JTree 的外观风格。

12.10.5　扩展 DefaultTreeCellRenderer 改变节点外观

DefaultTreeCellRenderer 实现类实现了 TreeCellRenderer 接口，该接口里只有一个用于绘制节点内容的方法：getTreeCellRendererComponent()，该方法负责绘制 JTree 节点。如果读者还记得前面介绍的绘制 JList 的列表项外观的内容，应该对该方法非常熟悉——与 ListCellRenderer 接口类似的是，getTreeCellRendererComponent()方法返回一个 Component 对象，该对象就是 JTree 的节点组件。

DefaultTreeCellRenderer 类继承了 JLabel，实现 getTreeCellRendererComponent()方法时返回 this，即返回一个特殊的 JLabel 对象。如果需要根据节点内容来改变节点的外观，则可以再次扩展 DefaultTreeCellRenderer 类，并再次重写它提供的 getTreeCellRendererComponent()方法。

下面程序模拟了一个数据库对象导航树，程序可以根据节点的类型来绘制节点的图标。在本程序中为了给每个节点指定节点类型，程序不再使用 String 作为节点数据，而是使用 NodeData 来封装节点数据，并重写了 NodeData 的 toString()方法。

注意：

使用 Object 类型的对象来创建 TreeNode 对象时，DefaultTreeCellRenderer 默认使用该对象的 toString()方法返回的字符串作为该节点的标签。

程序清单：codes\12\12.10\ExtendsDefaultTreeCellRenderer.java

```
public class ExtendsDefaultTreeCellRenderer
{
    JFrame jf = new JFrame("根据节点类型定义图标");
    JTree tree;
    // 定义几个初始节点
    DefaultMutableTreeNode root = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.ROOT , "数据库导航"));
    DefaultMutableTreeNode salaryDb = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.DATABASE , "公司工资数据库"));
    DefaultMutableTreeNode customerDb = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.DATABASE , "公司客户数据库"));
    // 定义 salaryDb 的两个子节点
    DefaultMutableTreeNode employee = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.TABLE , "员工表"));
    DefaultMutableTreeNode attend = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.TABLE , "考勤表"));
    // 定义 customerDb 的一个子节点
    DefaultMutableTreeNode contact = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.TABLE , "联系方式表"));
    // 定义 employee 的三个子节点
    DefaultMutableTreeNode id = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.INDEX , "员工 ID"));
    DefaultMutableTreeNode name = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.COLUMN , "姓名"));
    DefaultMutableTreeNode gender = new DefaultMutableTreeNode(
        new NodeData(DBObjectType.COLUMN , "性别"));
    public void init()
    {
        // 通过 add()方法建立树节点之间的父子关系
        root.add(salaryDb);
        root.add(customerDb);
        salaryDb.add(employee);
        salaryDb.add(attend);
        customerDb.add(contact);
        employee.add(id);
        employee.add(name);
        employee.add(gender);
        // 以根节点创建树
        tree = new JTree(root);
        // 设置该 JTree 使用自定义的节点绘制器
        tree.setCellRenderer(new MyRenderer());
        // 设置是否显示根节点的展开/折叠图标，默认是 false
        tree.setShowsRootHandles(true);
        // 设置节点是否可见，默认是 true
        tree.setRootVisible(true);
        try
        {
            // 设置使用 Windows 风格外观
            UIManager.setLookAndFeel("com.sun.java.swing.plaf."
                + "windows.WindowsLookAndFeel");
        }
```

```
            catch (Exception ex){}
            // 更新 JTree 的 UI 外观
            SwingUtilities.updateComponentTreeUI(tree);
            jf.add(new JScrollPane(tree));
            jf.pack();
            jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            jf.setVisible(true);
        }
        public static void main(String[] args)
        {
            new ExtendsDefaultTreeCellRenderer().init();
        }
    }
    // 定义一个 NodeData 类，用于封装节点数据
    class NodeData
    {
        public int nodeType;
        public String nodeData;
        public NodeData(int nodeType , String nodeData)
        {
            this.nodeType = nodeType;
            this.nodeData = nodeData;
        }
        public String toString()
        {
            return nodeData;
        }
    }
    // 定义一个接口，该接口里包含数据库对象类型的常量
    interface DBObjectType
    {
        int ROOT = 0;
        int DATABASE = 1;
        int TABLE = 2;
        int COLUMN = 3;
        int INDEX = 4;
    }
    class MyRenderer extends DefaultTreeCellRenderer
    {
        // 初始化 5 个图标
        ImageIcon rootIcon = new ImageIcon("icon/root.gif");
        ImageIcon databaseIcon = new ImageIcon("icon/database.gif");
        ImageIcon tableIcon = new ImageIcon("icon/table.gif");
        ImageIcon columnIcon = new ImageIcon("icon/column.gif");
        ImageIcon indexIcon = new ImageIcon("icon/index.gif");
        public Component getTreeCellRendererComponent(JTree tree
            , Object value , boolean sel , boolean expanded
            , boolean leaf , int row , boolean hasFocus)
        {
            // 执行父类默认的节点绘制操作
            super.getTreeCellRendererComponent(tree , value
                , sel, expanded , leaf , row , hasFocus);
            DefaultMutableTreeNode node = (DefaultMutableTreeNode)value;
            NodeData data = (NodeData)node.getUserObject();
            // 根据数据节点里的 nodeType 数据决定节点图标
            ImageIcon icon = null;
            switch(data.nodeType)
            {
                case DBObjectType.ROOT:
                    icon = rootIcon;
                    break;
                case DBObjectType.DATABASE:
                    icon = databaseIcon;
                    break;
                case DBObjectType.TABLE:
                    icon = tableIcon;
                    break;
                case DBObjectType.COLUMN:
                    icon = columnIcon;
                    break;
```

```
            case DBObjectType.INDEX:
                icon = indexIcon;
                break;
        }
        // 改变图标
        this.setIcon(icon);
        return this;
    }
}
```

图 12.45 根据节点类型绘制节点图标

程序中的粗体字代码强制 JTree 使用自定义的节点绘制器：MyRenderer，该节点绘制器继承了 DefaultTreeCellRenderer 类，并重写了 getTreeCellRendererComponent()方法。该节点绘制器重写该节点时根据节点的 nodeType 属性改变其图标。运行上面程序，会看到如图 12.45 所示的效果。

从图 12.45 中可以看出，JTree 中表示节点展开、折叠的图标已经改为了“+”和“-”，这是因为本程序强制 JTree 使用了 Windows 风格。

12.10.6 实现 TreeCellRenderer 改变节点外观

这种方式是最灵活的方式，程序实现 TreeCellRenderer 接口时同样需要实现 getTreeCellRendererComponent()方法，该方法可以返回任意类型的组件，该组件将作为 JTree 的节点。通过这种方式可以最大程度地改变 JTree 的节点外观。

与前面实现 ListCellRenderer 接口类似的是，本实例程序同样通过扩展 JPanel 来实现 TreeCellRenderer，实现 TreeCellRenderer 的方式与前面实现 ListCellRenderer 的方式基本相似，所以读者将会看到一个完全不同的 JTree。

程序清单：codes\12\12.10\CustomTreeNode.java

```
public class CustomTreeNode
{
    JFrame jf = new JFrame("定制树的节点");
    JTree tree;
    // 定义几个初始节点
    DefaultMutableTreeNode friends = new DefaultMutableTreeNode("我的好友");
    DefaultMutableTreeNode qingzhao = new DefaultMutableTreeNode("李清照");
    DefaultMutableTreeNode suge = new DefaultMutableTreeNode("苏格拉底");
    DefaultMutableTreeNode libai = new DefaultMutableTreeNode("李白");
    DefaultMutableTreeNode nongyu = new DefaultMutableTreeNode("弄玉");
    DefaultMutableTreeNode hutou = new DefaultMutableTreeNode("虎头");
    public void init()
    {
        // 通过 add()方法建立树节点之间的父子关系
        friends.add(qingzhao);
        friends.add(suge);
        friends.add(libai);
        friends.add(nongyu);
        friends.add(hutou);
        // 以根节点创建树
        tree = new JTree(friends);
        // 设置是否显示根节点的展开/折叠图标，默认是 false
        tree.setShowsRootHandles(true);
        // 设置节点是否可见，默认是 true
        tree.setRootVisible(true);
        // 设置使用定制的节点绘制器
        tree.setCellRenderer(new ImageCellRenderer());
        jf.add(new JScrollPane(tree));
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
```

```
    public static void main(String[] args)
    {
        new CustomTreeNode().init();
    }
}
// 实现自己的节点绘制器
class ImageCellRenderer extends JPanel implements TreeCellRenderer
{
    private ImageIcon icon;
    private String name;
    // 定义绘制单元格时的背景色
    private Color background;
    // 定义绘制单元格时的前景色
    private Color foreground;
    public Component getTreeCellRendererComponent(JTree tree
        , Object value , boolean sel , boolean expanded
        , boolean leaf , int row , boolean hasFocus)
    {
        icon = new ImageIcon("icon/" + value + ".gif");
        name = value.toString();
        background = hasFocus ? new Color(140 , 200 ,235)
            : new Color(255 , 255 , 255);
        foreground = hasFocus ? new Color(255 , 255 ,3)
            : new Color(0 , 0 , 0);
        // 返回该 JPanel 对象作为单元格绘制器
        return this;
    }
    // 重写 paintComponent 方法，改变 JPanel 的外观
    public void paintComponent(Graphics g)
    {
        int imageWidth = icon.getImage().getWidth(null);
        int imageHeight = icon.getImage().getHeight(null);
        g.setColor(background);
        g.fillRect(0 , 0 , getWidth() , getHeight());
        g.setColor(foreground);
        // 绘制好友图标
        g.drawImage(icon.getImage() , getWidth() / 2
            - imageWidth / 2 , 10 , null);
        g.setFont(new Font("SansSerif" , Font.BOLD , 18));
        // 绘制好友用户名
        g.drawString(name, getWidth() / 2
            - name.length() * 10 , imageHeight + 30 );
    }
    // 通过该方法来设置该 ImageCellRenderer 的最佳大小
    public Dimension getPreferredSize()
    {
        return new Dimension(80, 80);
    }
}
```

上面程序中的粗体字代码设置 JTree 对象使用定制的节点绘制器：ImageCellRenderer，该节点绘制器实现了 TreeCellRenderer 接口的 getTreeCellRendererComponent()方法，该方法返回 this，也就是一个特殊的 JPanel 对象，这个特殊的 JPanel 重写了 paintComponent()方法，重新绘制了 JPanel 的外观——根据节点数据来绘制图标和文本。运行上面程序，会看到如图 12.46 所示的树。

这看上去似乎不太像一棵树，但可从每个节点前的连接线、表示节点的展开/折叠的图标中看出这依然是一棵树。

图 12.46　自行定制树节点的外观

12.11　使用 JTable 和 TableModel 创建表格

表格也是 GUI 程序中常用的组件，表格是一个由多行、多列组成的二维显示区。Swing 的 JTable 以及相关类提供了这种表格支持，通

过使用 JTable 以及相关类，程序既可以使用简单的代码创建出表格来显示二维数据，也可以开发出功能丰富的表格，还可以为表格定制各种显示外观、编辑特性。

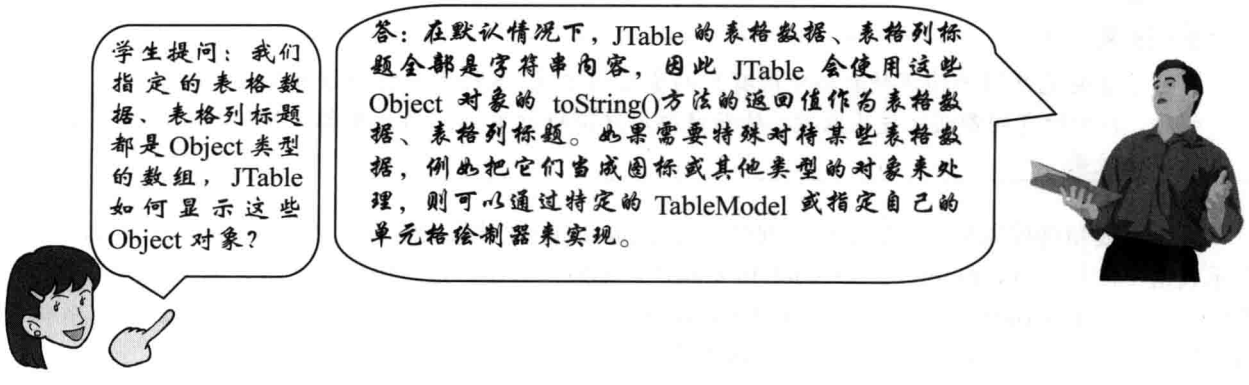
12.11.1 创建表格

使用 JTable 来创建表格是非常容易的事情，JTable 可以把一个二维数据包装成一个表格，这个二维数据既可以是一个二维数组，也可以是集合元素为 Vector 的 Vector 对象（Vector 里包含 Vector 形成二维数据）。除此之外，为了给该表格的每一列指定列标题，还需要传入一个一维数据作为列标题，这个一维数据既可以是一维数组，也可以是 Vector 对象。下面程序使用二维数组和一维数组来创建一个简单表格。

程序清单：codes\12\12.11\SimpleTable.java

```
public class SimpleTable
{
    JFrame jf = new JFrame("简单表格");
    JTable table;
    // 定义二维数组作为表格数据
    Object[][] tableData =
    {
        new Object[]{"李清照" , 29 , "女"},
        new Object[]{"苏格拉底", 56 , "男"},
        new Object[]{"李白", 35 , "男"},
        new Object[]{"弄玉", 18 , "女"},
        new Object[]{"虎头" , 2 , "男"}
    };
    // 定义一维数据作为列标题
    Object[] columnTitle = {"姓名" , "年龄" , "性别"};
    public void init()
    {
        // 以二维数组和一维数组来创建一个 JTable 对象
        table = new JTable(tableData , columnTitle);
        // 将 JTable 对象放在 JScrollPane 中
        // 并将该 JScrollPane 放在窗口中显示出来
        jf.add(new JScrollPane(table));
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new SimpleTable().init();
    }
}
```

上面程序中的粗体字代码创建了两个 Object 数组，第一个二维数组作为 JTable 的数据，第二个一维数组作为 JTable 的列标题。创建二维数组时利用了 JDK 1.5 提供的自动装箱功能——虽然直接指定的数组元素是 int 类型的整数，但系统会将它包装成 Integer 对象。



在默认情况下，JTable 的所有单元格、列标题显示的全部是字符串内容。除此之外，通常应该将 JTable

对象放在 JScrollPane 容器中，由 JScrollPane 为 JTable 提供 ViewPort。

通常总是会把 JTable 对象放在 JScrollPane 中显示，使用 JScrollPane 来包装 JTable 不仅可以为 JTable 增加滚动条，而且可以让 JTable 的列标题显示出来；如果不把 JTable 放在 JScrollPane 中显示，JTable 默认不会显示列标题。

运行上面程序，会看到如图 12.47 所示的简单表格。

虽然生成如图 12.47 所示表格的代码非常简单，但这个表格已经表现出丰富的功能。该表格具有如下几个功能。

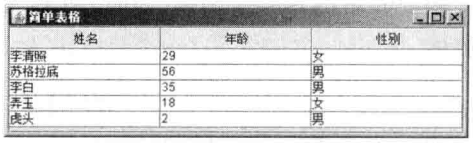


图 12.47　简单表格

- 当表格高度不足以显示所有的数据行时，该表格会自动显示滚动条。
- 当把鼠标移动到两列之间的分界符时，鼠标形状会变成可调整大小的形状，表明用户可以自由调整表格列的大小。
- 当在表格列上按下鼠标并拖动时，可以将表格的整列拖动到其他位置。
- 当单击某一个单元格时，系统会自动选中该单元格所在的行。
- 当双击某一个单元格时，系统会自动进入该单元格的修改状态。

运行 SimpleTable.java 程序，当拖动两列分界线来调整某列的列宽时，将看到该列后面的所有列的列宽都会发生相应的改变，但该列前面的所有列的列宽都不会发生改变，整个表格的宽度不会发生改变。

JTable 提供了一个 setAutoResizeMode()方法来控制这种调整方式，该方法可以接收如下几个值。

- JTable.AUTO_RESIZE_OFF：关闭 JTable 的自动调整功能，当调整某一列的宽度时，其他列的宽度不会发生改变，只有表格的宽度会随之改变。
- JTable.AUTO_RESIZE_NEXT_COLUMN：只调整下一列的宽度，其他列及表格的宽度不会发生改变。
- JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS：平均调整当前列后面所有列的宽度，当前列的前面所有列及表格的宽度都不会发生变化，这是默认的调整方式。
- JTable.AUTO_RESIZE_LAST_COLUMN：只调整最后一列的宽度，其他列及表格的宽度不会发生改变。
- JTable.AUTO_RESIZE_ALL_COLUMNS：平均调整表格中所有列的宽度，表格的宽度不会发生改变。

JTable 默认采用平均调整当前列后面所有列的宽度的方式，这种方式允许用户从左到右依次调整每一列的宽度，以达到最好的显示效果。

尽量避免使用平均调整表格中所有列的宽度的方式，这种方式将会导致用户调整某一列时，其余所有列都随之发生改变，从而使得用户很难把每一列的宽度都调整到具有最好的显示效果。

如果需要精确控制每一列的宽度，则可通过 TableColumn 对象来实现。JTable 使用 TableColumn 来表示表格中的每一列，JTable 中表格列的所有属性，如最佳宽度、是否可调整宽度、最小和最大宽度等都保存在该 TableColumn 中。此外，TableColumn 还允许为该列指定特定的单元格绘制器和单元格编辑器（这些内容将在后面讲解）。TableColumn 具有如下方法。

- setMaxWidth(int maxWidth)：设置该列的最大宽度。如果指定的 maxWidth 小于该列的最小宽度，

则 maxWidth 被设置成最小宽度。

- setMinWidth(int minWidth)：设置该列的最小宽度。
- setPreferredWidth(int preferredWidth)：设置该列的最佳宽度。
- setResizable(boolean isResizable)：设置是否可以调整该列的宽度。
- sizeWidthToFit()：调整该列的宽度，以适合其标题单元格的宽度。

在默认情况下，当用户单击 JTable 的任意一个单元格时，系统默认会选中该单元格所在行的整行，也就是说，JTable 表格默认的选择单元是行。当然也可通过 JTable 提供的 setRowSelectionAllowed()方法来改变这种设置，如果为该方法传入 false 参数，则可以关闭这种每次选择一行的方式。

除此之外，JTable 还提供了一个 setColumnSelectionAllowed()方法，该方法用于控制选择单元是否是列，如果为该方法传入 true 参数，则当用户单击某个单元格时，系统会选中该单元格所在的列。

如果同时调用 setColumnSelectionAllowed(true)和 setRowSelectionAllowed(true)方法，则该表格的选择单元是单元格。实际上，同时调用这两个方法相当于调用 setCellSelectionEnabled(true)方法。与此相反，如果调用 setCellSelectionEnabled(false)方法，则相当于同时调用 setColumnSelectionAllowed(false)和 setRowSelectionAllowed (false)方法，即用户无法选中该表格的任何地方。

与 JList、JTree 类似的是，JTable 使用了一个 ListSelectionModel 表示该表格的选择状态，程序可以通过 ListSelectionModel 来控制 JTable 的选择模式。JTable 的选择模式有如下三种。

- ListSelectionModel.MULTIPLE_INTERVAL_SELECTION：没有任何限制，可以选择表格中任何表格单元，这是默认的选择模式。通过 Shift 和 Ctrl 辅助键的帮助可以选择多个表格单元。
- ListSelectionModel.SINGLE_INTERVAL_SELECTION：选择单个连续区域，该选项可以选择多个表格单元，但多个表格单元之间必须是连续的。通过 Shift 辅助键的帮助来选择连续区域。
- ListSelectionModel.SINGLE_SELECTION：只能选择单个表格单元。

程序通常通过如下代码来改变 JTable 的选择模式。

```
// 设置该表格只能选中单个表格单元
table.getSelectionModel().setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

注意：

保存 JTable 选择状态的 model 类就是 ListSelectionModel，这并不是笔误。

下面程序示范了如何控制每列的宽度、控制表格的宽度调整模式、改变表格的选择单元和表格的选择模式。

程序清单：codes\12\12.11\AdjustingWidth.java

```
public class AdjustingWidth
{
    JFrame jf = new JFrame("调整表格列宽");
    JMenuBar menuBar = new JMenuBar();
    JMenu adjustModeMenu = new JMenu("调整方式");
    JMenu selectUnitMenu = new JMenu("选择单元");
    JMenu selectModeMenu = new JMenu("选择方式");
    // 定义5个单选框按钮，用以控制表格的宽度调整方式
    JRadioButtonMenuItem[] adjustModesItem = new JRadioButtonMenuItem[5];
    // 定义3个单选框按钮，用以控制表格的选择方式
    JRadioButtonMenuItem[] selectModesItem = new JRadioButtonMenuItem[3];
    JCheckBoxMenuItem rowsItem = new JCheckBoxMenuItem("选择行");
    JCheckBoxMenuItem columnsItem = new JCheckBoxMenuItem("选择列");
    JCheckBoxMenuItem cellsItem = new JCheckBoxMenuItem("选择单元格");
    ButtonGroup adjustBg = new ButtonGroup();
    ButtonGroup selectBg = new ButtonGroup();
    // 定义一个 int 类型的数组，用于保存表格所有的宽度调整方式
    int[] adjustModes = new int[]{
        JTable.AUTO_RESIZE_OFF
        , JTable.AUTO_RESIZE_NEXT_COLUMN
        , JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS
```

```
        , JTable.AUTO_RESIZE_LAST_COLUMN
        , JTable.AUTO_RESIZE_ALL_COLUMNS
    };
    int[] selectModes = new int[]{
        ListSelectionModel.MULTIPLE_INTERVAL_SELECTION
        , ListSelectionModel.SINGLE_INTERVAL_SELECTION
        , ListSelectionModel.SINGLE_SELECTION
    };
    JTable table;
    // 定义二维数组作为表格数据
    Object[][] tableData =
    {
        new Object[]{"李清照" , 29 , "女"},
        new Object[]{"苏格拉底", 56 , "男"},
        new Object[]{"李白", 35 , "男"},
        new Object[]{"弄玉", 18 , "女"},
        new Object[]{"虎头" , 2 , "男"}
    };
    // 定义一维数据作为列标题
    Object[] columnTitle = {"姓名" , "年龄" , "性别"};
    public void init()
    {
        // 以二维数组和一维数组来创建一个 JTable 对象
        table = new JTable(tableData , columnTitle);
        // -----------为窗口安装设置表格调整方式的菜单-----------
        adjustModesItem[0] = new JRadioButtonMenuItem("只调整表格");
        adjustModesItem[1] = new JRadioButtonMenuItem("只调整下一列");
        adjustModesItem[2] = new JRadioButtonMenuItem("平均调整余下列");
        adjustModesItem[3] = new JRadioButtonMenuItem("只调整最后一列");
        adjustModesItem[4] = new JRadioButtonMenuItem("平均调整所有列");
        menuBar.add(adjustModeMenu);
        for (int i = 0; i < adjustModesItem.length ; i++)
        {
            // 默认选中第三个菜单项，即对应表格默认的宽度调整方式
            if (i == 2)
            {
                adjustModesItem[i].setSelected(true);
            }
            adjustBg.add(adjustModesItem[i]);
            adjustModeMenu.add(adjustModesItem[i]);
            final int index = i;
            // 为设置调整方式的菜单项添加监听器
            adjustModesItem[i].addActionListener(evt -> {
                // 如果当前菜单项处于选中状态，表格使用对应的调整方式
                if (adjustModesItem[index].isSelected())
                {
                    table.setAutoResizeMode(adjustModes[index]); // ①
                }
            });
        }
        // -----------为窗口安装设置表格选择方式的菜单-----------
        selectModesItem[0] = new JRadioButtonMenuItem("无限制");
        selectModesItem[1] = new JRadioButtonMenuItem("单独的连续区");
        selectModesItem[2] = new JRadioButtonMenuItem("单选");
        menuBar.add(selectModeMenu);
        for (int i = 0; i < selectModesItem.length ; i++)
        {
            // 默认选中第一个菜单项，即对应表格默认的选择方式
            if (i == 0)
            {
            selectModesItem[i].setSelected(true);
            }
            selectBg.add(selectModesItem[i]);
            selectModeMenu.add(selectModesItem[i]);
            final int index = i;
            // 为设置选择方式的菜单项添加监听器
            selectModesItem[i].addActionListener(evt -> {
                // 如果当前菜单项处于选中状态，表格使用对应的选择方式
                if (selectModesItem[index].isSelected())
                {
```

```
                table.getSelectionModel().setSelectionMode
                    (selectModes[index]);     // ②
            }
        });
    }
    menuBar.add(selectUnitMenu);
    // -----为窗口安装设置表格选择单元的菜单-----
    rowsItem.setSelected(table.getRowSelectionAllowed());
    columnsItem.setSelected(table.getColumnSelectionAllowed());
    cellsItem.setSelected(table.getCellSelectionEnabled());
    rowsItem.addActionListener(event -> {
        table.clearSelection();
        // 如果该菜单项处于选中状态，设置表格的选择单元是行
        table.setRowSelectionAllowed(rowsItem.isSelected());
        // 如果选择行、选择列同时被选中，其实质是选择单元格
        cellsItem.setSelected(table.getCellSelectionEnabled());
    });
    selectUnitMenu.add(rowsItem);
    columnsItem.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            table.clearSelection();
            // 如果该菜单项处于选中状态，设置表格的选择单元是列
            table.setColumnSelectionAllowed(columnsItem.isSelected());
            // 如果选择行、选择列同时被选中，其实质是选择单元格
            cellsItem.setSelected(table.getCellSelectionEnabled());
        }
    });
    selectUnitMenu.add(columnsItem);
    cellsItem.addActionListener(event -> {
        table.clearSelection();
        // 如果该菜单项处于选中状态，设置表格的选择单元是单元格
        table.setCellSelectionEnabled(cellsItem.isSelected());
        // 该选项的改变会同时影响选择行、选择列两个菜单
        rowsItem.setSelected(table.getRowSelectionAllowed());
        columnsItem.setSelected(table.getColumnSelectionAllowed());
    });
    selectUnitMenu.add(cellsItem);
    jf.setJMenuBar(menuBar);
    // 分别获取表格的三个表格列，并设置三列的最小宽、最佳宽度和最大宽度
    TableColumn nameColumn = table.getColumn(columnTitle[0]);
    nameColumn.setMinWidth(40);
    TableColumn ageColumn = table.getColumn(columnTitle[1]);
    ageColumn.setPreferredWidth(50);
    TableColumn genderColumn = table.getColumn(columnTitle[2]);
    genderColumn.setMaxWidth(50);
    // 将 JTable 对象放在 JScrollPane 中，并将该 JScrollPane 放在窗口中显示出来
    jf.add(new JScrollPane(table));
    jf.pack();
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setVisible(true);
}
public static void main(String[] args)
{
    new AdjustingWidth().init();
}
}
```

上面程序中的①号粗体字代码根据单选钮菜单来设置表格的宽度调整方式，②号粗体字代码根据单选钮菜单来设置表格的选择模式，最后一段粗体字代码通过 JTable 的 getColumn()方法获取指定列，并分别设置三列的最佳、最大、最小宽度。如果选中“只调整表格”菜单项，并把第一列宽度拖大，将看到如图 12.48 所示的界面。

上面程序中还有三段粗体字代码，分别用于为三个复选框菜单添加监听器，根据复选框菜单的选中状态来决定表格的选择单元。如果程序采用 JTable 默认的选择模式（无限制的选择模式），并设置表格的选择单元是单元格，则可看到如图 12.49 所示的界面。

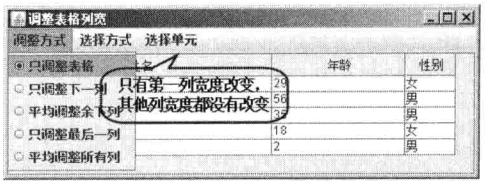


图 12.48　采用只调整表格宽度的方式

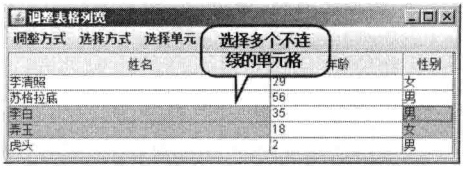


图 12.49　选择多个不连续的单元格

12.11.2　TableModel 和监听器

与 JList、JTree 类似的是，JTable 采用了 TableModel 来保存表格中的所有状态数据；与 ListModel 类似的是，TableModel 也不强制保存该表格显示的数据。虽然在前面程序中看到的是直接利用一个二维数组来创建 JTable 对象，但也可以通过 TableModel 对象来创建表格。如果需要利用 TableModel 来创建表格对象，则可以利用 Swing 提供的 AbstractTableModel 抽象类，该抽象类已经实现了 TableModel 接口里的大部分方法，程序只需要为该抽象类实现如下三个抽象方法即可。

- getColumnCount()：返回该 TableModel 对象的列数量。
- getRowCount()：返回该 TableModel 对象的行数量。
- getValueAt()：返回指定行、指定列的单元格值。

重写这三个方法后只是告诉 JTable 生成该表格所需的基本信息，如果想指定 JTable 生成表格的列名，还需要重写 getColumnName(int c)方法，该方法返回一个字符串，该字符串将作为第 c+1 列的列名。

在默认情况下，AbstractTableModel 的 boolean isCellEditable(int rowIndex, int columnIndex)方法返回 false，表明该表格的单元格处于不可编辑状态，如果想让用户直接修改单元格的内容，则需要重写该方法，并让该方法返回 true。重写该方法后，只实现了界面上单元格的可编辑，如果需要控制实际的编辑操作，还需要重写该类的 setValueAt(Object aValue, int rowIndex, int columnIndex)方法。

关于 TableModel 的典型应用就是用于封装 JDBC 编程里的 ResultSet，程序可以利用 TableModel 来封装数据库查询得到的结果集，然后使用 JTable 把该结果集显示出来。还可以允许用户直接编辑表格的单元格，当用户编辑完成后，程序将用户所做的修改写入数据库。下面程序简单实现了这种功能——当用户选择了指定的数据表后，程序将显示该数据表中的全部数据，用户可以直接在该表格内修改数据表的记录。

程序清单：codes\12\12.11\TableModelTest.java

```
public class TableModelTest
{
    JFrame jf = new JFrame("数据表管理工具");
    private JScrollPane scrollPane;
    private ResultSetTableModel model;
    // 用于装载数据表的 JComboBox
    private JComboBox<String> tableNames = new JComboBox<>();
    private JTextArea changeMsg = new JTextArea(4, 80);
    private ResultSet rs;
    private Connection conn;
    private Statement stmt;
    public void init()
    {
        // 为 JComboBox 添加事件监听器，当用户选择某个数据表时，触发该方法
        tableNames.addActionListener(event -> {
            try
            {
                // 如果装载 JTable 的 JScrollPane 不为空
                if (scrollPane != null)
                {
                    // 从主窗口中删除表格
                    jf.remove(scrollPane);
                }
                // 从 JComboBox 中取出用户试图管理的数据表的表名
                String tableName = (String) tableNames.getSelectedItem();
```

```
            // 如果结果集不为空，则关闭结果集
            if (rs != null)
            {
                rs.close();
            }
            String query = "select * from " + tableName;
            // 查询用户选择的数据表
            rs = stmt.executeQuery(query);
            // 使用查询到的 ResultSet 创建 TableModel 对象
            model = new ResultSetTableModel(rs);
            // 为 TableModel 添加监听器，监听用户的修改
            model.addTableModelListener(evt -> {
                int row = evt.getFirstRow();
                int column = evt.getColumn();
                changeMsg.append("修改的列:" + column
                    + ",修改的行:" + row + "修改后的值:"
                    + model.getValueAt(row , column));
            });
            // 使用 TableModel 创建 JTable，并将对应表格添加到窗口中
            JTable table = new JTable(model);
            scrollPane = new JScrollPane(table);
            jf.add(scrollPane, BorderLayout.CENTER);
            jf.validate();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    });
    JPanel p = new JPanel();
    p.add(tableNames);
    jf.add(p, BorderLayout.NORTH);
    jf.add(new JScrollPane(changeMsg), BorderLayout.SOUTH);
    try
    {
        // 获取数据库连接
        conn = getConnection();
        // 获取数据库的 MetaData 对象
        DatabaseMetaData meta = conn.getMetaData();
        // 创建 Statement
        stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE
            , ResultSet.CONCUR_UPDATABLE);
        // 查询当前数据库的全部数据表
        ResultSet tables = meta.getTables(null, null, null
            , new String[] { "TABLE" });
        // 将全部数据表添加到 JComboBox 中
        while (tables.next())
        {
            tableNames.addItem(tables.getString(3));
        }
        tables.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    jf.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent event)
        {
            try
            {
                if (conn != null) conn.close();
            }
            catch (SQLException e)
            {
```

```
                e.printStackTrace();
            }
        }
    });
    jf.pack();
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setVisible(true);
}
private static Connection getConnection()
    throws SQLException, IOException , ClassNotFoundException
{
    // 通过加载 conn.ini 文件来获取数据库连接的详细信息
    Properties props = new Properties();
    FileInputStream in = new FileInputStream("conn.ini");
    props.load(in);
    in.close();
    String drivers = props.getProperty("jdbc.drivers");
    String url = props.getProperty("jdbc.url");
    String username = props.getProperty("jdbc.username");
    String password = props.getProperty("jdbc.password");
    // 加载数据库驱动
    Class.forName(drivers);
    // 取得数据库连接
    return DriverManager.getConnection(url, username, password);
}
public static void main(String[] args)
{
    new TableModelTest().init();
}
}
// 扩展 AbstractTableModel，用于将一个 ResultSet 包装成 TableModel
class ResultSetTableModel extends AbstractTableModel   // ①
{
    private ResultSet rs;
    private ResultSetMetaData rsmd;
    // 构造器，初始化 rs 和 rsmd 两个属性
    public ResultSetTableModel(ResultSet aResultSet)
    {
        rs = aResultSet;
        try
        {
            rsmd = rs.getMetaData();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
    // 重写 getColumnName 方法，用于为该 TableModel 设置列名
    public String getColumnName(int c)
    {
        try
        {
            return rsmd.getColumnName(c + 1);
        }
        catch (SQLException e)
        {
            e.printStackTrace();
            return "";
        }
    }
    // 重写 getColumnCount 方法，用于设置该 TableModel 的列数
    public int getColumnCount()
    {
        try
        {
            return rsmd.getColumnCount();
        }
        catch (SQLException e)
        {
```

```
            e.printStackTrace();
            return 0;
        }
    }
    // 重写 getValueAt 方法，用于设置该 TableModel 指定单元格的值
    public Object getValueAt(int r, int c)
    {
        try
        {
            rs.absolute(r + 1);
            return rs.getObject(c + 1);
        }
        catch(SQLException e)
        {
            e.printStackTrace();
            return null;
        }
    }
    // 重写 getRowCount 方法，用于设置该 TableModel 的行数
    public int getRowCount()
    {
        try
        {
            rs.last();
            return rs.getRow();
        }
        catch(SQLException e)
        {
            e.printStackTrace();
            return 0;
        }
    }
    // 重写 isCellEditable 返回 true，让每个单元格可编辑
    public boolean isCellEditable(int rowIndex, int columnIndex)
    {
        return true;
    }
    // 重写 setValueAt()方法，当用户编辑单元格时，将会触发该方法
    public void setValueAt(Object aValue , int row,int column)
    {
        try
        {
            // 结果集定位到对应的行数
            rs.absolute(row + 1);
            // 修改单元格对应的值
            rs.updateObject(column + 1 , aValue);
            // 提交修改
            rs.updateRow();
            // 触发单元格的修改事件
            fireTableCellUpdated(row, column);
        }
        catch (SQLException evt)
        {
            evt.printStackTrace();
        }
    }
}
```

上面程序的关键在于①号粗体字代码所扩展的 ResultSetTableModel 类，该类继承了 AbstractTableModel 父类，根据其 ResultSet 来重写 getColumnCount()、getRowCount()和 getValueAt()三个方法，从而允许该表格可以将该 ResultSet 里的所有记录显示出来。除此之外，该扩展类还重写了 isCellEditable()和 setValueAt()两个方法——重写前一个方法实现允许用户编辑单元格的功能，重写后一个方法实现当用户编辑单元格时将所做的修改同步到数据库的功能。

程序中的粗体字代码使用 ResultSet 创建了一个 TableModel 对象，并为该 TableModel 添加事件监听器，然后把该 TableModel 使用 JTable 显示出来。当用户修改该 JTable 对应表格里单元格的内容时，该监听器会检测到这种修改，并将这种修改信息通过下面的文本域显示出来。

提示：

上面程序大量使用了 JDBC 编程中的 JDBC 连接数据库、获取可更新的结果集、ResultSetMetaData、DatabaseMetaData 等知识，读者可能一时难以读懂，可以参考本书第 13 章的内容来阅读本程序。该程序的运行需要底层数据库的支持，所以读者应按第 13 章的内容正常安装 MySQL 数据库，并将 codes\12\12.11\路径下的 mysql.sql 脚本导入数据库，修改 conn.ini 文件中的数据库连接信息才可运行该程序。使用 JDBC 连接数据库还需要加载 JDBC 驱动，所以本章为运行该程序提供了一个 run.cmd 批处理文件，读者可以通过该文件来运行该程序。不要直接运行该程序，否则可能出现 java.lang.ClassNotFoundException: com.mysql.jdbc.Driver 异常。

运行上面程序，会看到如图 12.50 所示的界面。

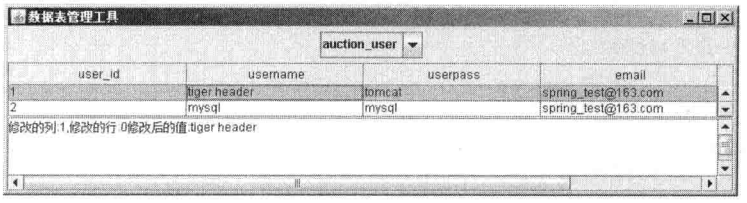


图 12.50　使用 JTable 管理数据表记录

从图 12.50 中可以看出，当修改指定单元格的记录时，添加在 TableModel 上的监听器就会被触发。当修改 JTable 单元格里的内容时，底层数据表里的记录也会做出相应的改变。

不仅用户可以扩展 AbstractTableModel 抽象类，Swing 本身也为 AbstractTableModel 提供了一个 DefaultTableModel 实现类，程序可以通过使用 DefaultTableModel 实现类来创建 JTable 对象。通过 DefaultTableModel 对象创建 JTable 对象后，就可以调用它提供的方法来添加数据行、插入数据行、删除数据行和移动数据行。DefaultTableModel 提供了如下几个方法来控制数据行操作。

- addColumn()：该方法用于为 TableModel 增加一列，该方法有三个重载的版本，实际上该方法只是将原来隐藏的数据列显示出来。
- addRow()：该方法用于为 TableModel 增加一行，该方法有两个重载的版本。
- insertRow()：该方法用于在 TableModel 的指定位置插入一行，该方法有两个重载的版本。
- removeRow(int row)：该方法用于删除 TableModel 中的指定行。
- moveRow(int start, int end, int to)：该方法用于移动 TableModel 中指定范围的数据行。

通过 DefaultTableModel 提供的这样几个方法，程序就可以动态地改变表格里的数据行。

提示：

Swing 为 TableModel 提供了两个实现类，其中一个是 DefaultTableModel，另一个是 JTable 的匿名内部类。如果直接使用二维数组来创建 JTable 对象，维护该 JTable 状态信息的 model 对象就是 JTable 匿名内部类的实例；当使用 Vector 来创建 JTable 对象时，维护该 JTable 状态信息的 model 对象就是 DefaultTableModel 实例。

12.11.3　TableColumnModel 和监听器

JTable 使用 TableColumnModel 来保存该表格所有数据列的状态数据，如果程序需要访问 JTable 的所有列状态信息，则可以通过获取该 JTable 的 TableColumnModel 来实现。TableColumnModel 提供了如下几个方法来增加、删除和移动数据列。

- addColumn(TableColumn aColumn)：该方法用于为 TableModel 添加一列。该方法主要用于将原来隐藏的数据列显示出来。
- moveColumn(int columnIndex, int newIndex)：该方法用于将指定列移动到其他位置。

- removeColumn(TableColumn column)：该方法用于从 TableModel 中删除指定列。实际上，该方法并未真正删除指定列，只是将该列在 TableColumnModel 中隐藏起来，使之不可见。

注意：

当调用 removeColumn()删除指定列之后，调用 TableColumnModel 的 getColumnCount()方法也会看到返回的列数减少了，看起来很像真正删除了该列。但使用 setValueAt()方法为该列设置值时，依然可以设置成功，这表明这些列依然是存在的。

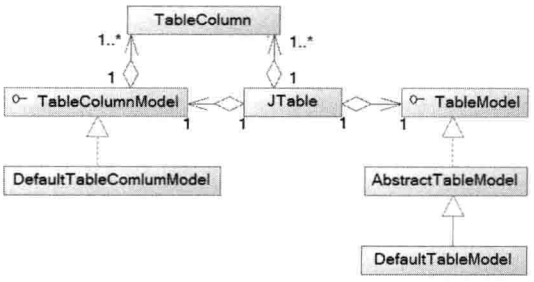


图 12.51 JTable 及其主要辅助类之间的关系

实际上，JTable 也提供了对应的方法来增加、删除和移动数据列，不过 JTable 的这些方法实际上还是需要委托给它所对应的 TableColumnModel 来完成。图 12.51 显示了 JTable 及其主要辅助类之间的关系。

下面程序示范了如何通过 DefaultTableModel 和 TableColumnModel 动态地改变表格的行、列。

程序清单：codes\12\12.11\DefaultTableModelTest.java

```
public class DefaultTableModelTest
{
    JFrame mainWin = new JFrame("管理数据行、数据列");
    final int COLUMN_COUNT = 5;
    DefaultTableModel model;
    JTable table;
    // 用于保存被隐藏列的 List 集合
    ArrayList<TableColumn> hiddenColumns = new ArrayList<>();
    public void init()
    {
        model = new DefaultTableModel(COLUMN_COUNT ,COLUMN_COUNT);
        for (int i = 0; i < COLUMN_COUNT ; i++ )
        {
            for (int j = 0; j < COLUMN_COUNT ; j++ )
            {
                model.setValueAt("老单元格值 " + i + " " + j , i , j);
            }
        }
        table = new JTable(model);
        mainWin.add(new JScrollPane(table), BorderLayout.CENTER);
        // 为窗口安装菜单
        JMenuBar menuBar = new JMenuBar();
        mainWin.setJMenuBar(menuBar);
        JMenu tableMenu = new JMenu("管理");
        menuBar.add(tableMenu);
        JMenuItem hideColumnsItem = new JMenuItem("隐藏选中列");
        hideColumnsItem.addActionListener(event -> {
            // 获取所有选中列的索引
            int[] selected = table.getSelectedColumns();
            TableColumnModel columnModel = table.getColumnModel();
            // 依次把每一个选中的列隐藏起来，并使用 List 保存这些列
            for (int i = selected.length - 1; i >= 0; i--)
            {
                TableColumn column = columnModel.getColumn(selected[i]);
                // 隐藏指定列
                table.removeColumn(column);
                // 把隐藏的列保存起来，确保以后可以显示出来
                hiddenColumns.add(column);
            }
        });
        tableMenu.add(hideColumnsItem);
        JMenuItem showColumnsItem = new JMenuItem("显示隐藏列");
        showColumnsItem.addActionListener(event -> {
            // 把所有隐藏的列依次显示出来
```

```
            for (TableColumn tc : hiddenColumns)
            {
                // 依次把所有隐藏的列显示出来
                table.addColumn(tc);
            }
            // 清空保存隐藏列的 List 集合
            hiddenColumns.clear();
        });
        tableMenu.add(showColumnsItem);
        JMenuItem addColumnItem = new JMenuItem("插入选中列");
        addColumnItem.addActionListener(event -> {
            // 获取所有选中列的索引
            int[] selected = table.getSelectedColumns();
            TableColumnModel columnModel = table.getColumnModel();
            // 依次把选中的列添加到 JTable 之后
            for (int i = selected.length - 1; i >= 0; i--)
            {
                TableColumn column = columnModel
                    .getColumn(selected[i]);
                table.addColumn(column);
            }
        });
        tableMenu.add(addColumnItem);
        JMenuItem addRowItem = new JMenuItem("增加行");
        addRowItem.addActionListener(event -> {
            // 创建一个 String 数组作为新增行的内容
            String[] newCells = new String[COLUMN_COUNT];
            for (int i = 0; i < newCells.length; i++)
            {
                newCells[i] = "新单元格值 " + model.getRowCount()
                    + " " + i;
            }
            // 向 TableModel 中新增一行
            model.addRow(newCells);
        });
        tableMenu.add(addRowItem);
        JMenuItem removeRowsItem = new  JMenuItem("删除选中行");
        removeRowsItem.addActionListener(event -> {
            // 获取所有选中行
            int[] selected = table.getSelectedRows();
            // 依次删除所有选中行
            for (int i = selected.length - 1; i >= 0; i--)
            {
                model.removeRow(selected[i]);
            }
        });
        tableMenu.add(removeRowsItem);
        mainWin.pack();
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.setVisible(true);
    }
    public static void main(String[] args)
    {
        new DefaultTableModelTest().init();
    }
}
```

上面程序中的粗体字代码部分就是程序控制隐藏列、显示隐藏列、增加数据行和删除数据行的代码。除此之外，程序还实现了一个功能：当用户选中某个数据列之后，还可以将该数据列添加到该表格的后面——但不要忘记了 add()方法的功能，它只是将已有的数据列显示出来，并不是真正添加数据列。运行上面程序，会看到如图 12.52 所示的界面。

从图 12.52 中可以看出，虽然程序新增了一列，但新增列的列名依然是 B，如果修改新增列内的单元格的值时，看到原来的 B 列的值也随之改变，由此可见，addColumn()方法只是将原有的列显示出来而已。程序还允许新增数据行，当执行 addRows()方法时需要传入数组或 Vector 参数，该参数里包含的多个数值将作为新增行的数据。

图 12.52　新增数据行、数据列的效果

如果程序需要监听 JTable 里列状态的改变，例如监听列的增加、删除、移动等改变，则必须使用该 JTable 所对应的 TableColumnModel 对象，该对象提供了一个 addColumnModelListener()方法来添加监听器，该监听器接口里包含如下几个方法。

- columnAdded(TableColumnModelEvent e)：当向 TableColumnModel 里添加数据列时将会触发该方法。
- columnMarginChanged(ChangeEvent e)：当由于页面距（Margin）的改变引起列状态改变时将会触发该方法。
- columnMoved(TableColumnModelEvent e)：当移动 TableColumnModel 里的数据列时将会触发该方法。
- columnRemoved(TableColumnModelEvent e)：当删除 TableColumnModel 里的数据列时将会触发该方法。
- columnSelectionChanged(ListSelectionEvent e)：当改变表格的选择模式时将会触发该方法。

但表格的数据列通常需要程序来控制增加、删除，用户操作通常无法直接为表格增加、删除数据列，所以使用监听器来监听 TableColumnModel 改变的情况比较少见。

12.11.4　实现排序

使用 JTable 实现的表格并没有实现根据指定列排序的功能，但开发者可以利用 AbstractTableModel 类来实现该功能。由于 TableModel 不强制要求保存表格里的数据，只要 TableModel 实现了 getValueAt()、getColumnCount()和 getRowCount()三个方法，JTable 就可以根据该 TableModel 生成表格。因此可以创建一个 SortableTableModel 实现类，它可以将原 TableModel 包装起来，并实现根据指定列排序的功能。

程序创建的 SortableTableModel 实现类会对原 TableModel 进行包装，但它实际上并不保存任何数据，它会把所有的方法实现委托给原 TableModel 完成。SortableTableModel 仅保存原 TableModel 里每行的行索引，当程序对 SortableTableModel 的指定列排序时，实际上仅仅对 SortableTableModel 里的行索引进行排序——这样造成的结果是：SortableTableModel 里的数据行的行索引与原 TableModel 里数据行的行索引不一致，所以对于 TableModel 的那些涉及行索引的方法都需要进行相应的转换。下面程序实现了 SortableTableModel 类，并使用该类来实现对表格根据指定列排序的功能。

程序清单：codes\12\12.11\SortTable.java

```
public class SortTable
{
    JFrame jf = new JFrame("可按列排序的表格");
    // 定义二维数组作为表格数据
    Object[][] tableData =
    {
        new Object[]{"李清照" , 29 , "女"},
        new Object[]{"苏格拉底", 56 , "男"},
        new Object[]{"李白", 35 , "男"},
        new Object[]{"弄玉", 18 , "女"},
        new Object[]{"虎头" , 2 , "男"}
    };
    // 定义一维数据作为列标题
    Object[] columnTitle = {"姓名" , "年龄" , "性别"};
    // 以二维数组和一维数组来创建一个 JTable 对象
    JTable table = new JTable(tableData , columnTitle);
```

```
        // 将原表格里的 model 包装成新的 SortTableModel 对象
        SortableTableModel sorterModel = new SortableTableModel(
            table.getModel());
        public void init()
        {
            // 使用包装后的 SortableTableModel 对象作为 JTable 的 model 对象
            table.setModel(sorterModel);
            // 为每列的列头增加鼠标监听器
            table.getTableHeader().addMouseListener(new MouseAdapter()
            {
                public void mouseClicked(MouseEvent event)    // ①
                {
                    // 如果单击次数小于 2，即不是双击，直接返回
                    if (event.getClickCount() < 2)
                    {
                        return;
                    }
                    // 找出鼠标双击事件所在的列索引
                    int tableColumn = table.columnAtPoint(event.getPoint());
                    // 将 JTable 中的列索引转换成对应 TableModel 中的列索引
                    int modelColumn = table.convertColumnIndexToModel(tableColumn);
                    // 根据指定列进行排序
                    sorterModel.sort(modelColumn);
                }
            });
            // 将 JTable 对象放在 JScrollPane 中，并将该 JScrollPane 显示出来
            jf.add(new JScrollPane(table));
            jf.pack();
            jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            jf.setVisible(true);
        }
        public static void main(String[] args)
        {
            new SortTable().init();
        }
    }
    class SortableTableModel extends AbstractTableModel
    {
        private TableModel model;
        private int sortColumn;
        private Row[] rows;
        // 将一个已经存在的 TableModel 对象包装成 SortableTableModel 对象
        public SortableTableModel(TableModel m)
        {
            // 将被封装的 TableModel 传入
            model = m;
            rows = new Row[model.getRowCount()];
            // 将原 TableModel 中每行记录的索引使用 Row 数组保存起来
            for (int i = 0; i < rows.length; i++)
            {
                rows[i] = new Row(i);
            }
        }
        // 实现根据指定列进行排序
        public void sort(int c)
        {
            sortColumn = c;
            java.util.Arrays.sort(rows);
            fireTableDataChanged();
        }
        // 下面三个方法需要访问 model 中的数据，所以涉及本 model 中数据
        // 和被包装 model 数据中的索引转换，程序使用 rows 数组完成这种转换
        public Object getValueAt(int r, int c)
        {
            return model.getValueAt(rows[r].index, c);
        }
        public boolean isCellEditable(int r, int c)
        {
            return model.isCellEditable(rows[r].index, c);
        }
```

```
    public void setValueAt(Object aValue, int r, int c)
    {
        model.setValueAt(aValue, rows[r].index, c);
    }
    // 下面方法的实现把该 model 的方法委托给原封装的 model 来实现
    public int getRowCount()
    {
        return model.getRowCount();
    }
    public int getColumnCount()
    {
        return model.getColumnCount();
    }
    public String getColumnName(int c)
    {
        return model.getColumnName(c);
    }
    public Class getColumnClass(int c)
    {
        return model.getColumnClass(c);
    }
    // 定义一个 Row 类，该类用于封装 JTable 中的一行
    // 实际上它并不封装行数据，它只封装行索引
    private class Row implements Comparable<Row>
    {
        // 该 index 保存着被封装 Model 里每行记录的行索引
        public int index;
        public Row(int index)
        {
            this.index = index;
        }
        // 实现两行之间的大小比较
        public int compareTo(Row other)
        {
            Object a = model.getValueAt(index, sortColumn);
            Object b = model.getValueAt(other.index, sortColumn);
            if (a instanceof Comparable)
            {
                return ((Comparable)a).compareTo(b);
            }
            else
            {
                return a.toString().compareTo(b.toString());
            }
        }
    }
}
```

上面程序是在 SimpleTable 程序的基础上改变而来的，改变的部分就是增加了两行粗体字代码和①号粗体字代码块。其中粗体字代码负责把原 JTable 的 model 对象包装成 SortableTableModel 实例，并设置原 JTable 使用 SortableTableModel 实例作为对应的 model 对象；而①号粗体字代码部分则用于为该表格的列头增加鼠标监听器：当用鼠标双击指定列时，SortableTableModel 对象根据指定列进行排序。

注意：

程序中还使用了 convertColumnIndexToModel()方法把 JTable 中的列索引转换成 TableModel 中的列索引。这是因为 JTable 中的列允许用户随意拖动，因此可能造成 JTable 中的列索引与 TableModel 中的列索引不一致。

运行上面程序，并双击“年龄”列头，将看到如图 12.53 所示的排序效果。

实际上，上面程序的关键在于 SortableTableModel 类，该类使用 rows[]数组来保存原 TableModel 里的行索引。

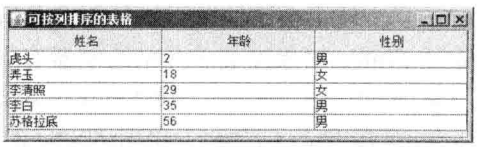


图 12.53 根据“年龄”列排序的效果

为了让程序可以对 rows[]数组元素根据指定列排序，程序使用了 Row 类来封装行索引，并实现了 compareTo()方法，该方法实现了根据指定列来比较两行大小的功能，从而允许程序根据指定列对 rows[]数组元素进行排序。

12.11.5 绘制单元格内容

前面看到的所有表格的单元格内容都是字符串，实际上表格的单元格内容也可以是更复杂的内容。JTable 使用 TableCellRenderer 绘制单元格，Swing 为该接口提供了一个实现类：DefaultTableCellRenderer，该单元格绘制器可以绘制如下三种类型的单元格值（根据其 TableModel 的 getColumnClass()方法来决定该单元格值的类型）。

- Icon：默认的单元格绘制器会把该类型的单元格值绘制成该 Icon 对象所代表的图标。
- Boolean：默认的单元格绘制器会把该类型的单元格值绘制成复选按钮。
- Object：默认的单元格绘制器在单元格内绘制出该对象的 toString()方法返回的字符串。

在默认情况下，如果程序直接使用二维数组或 Vector 来创建 JTable，程序将会使用 JTable 的匿名内部类或 DefaultTableModel 充当该表格的 model 对象，这两个 TableModel 的 getColumnClass()方法的返回值都是 Object。这意味着，即使该二维数组里值的类型是 Icon，但由于两个默认的 TableModel 实现类的 getColumnClass()方法总是返回 Object，这将导致默认的单元格绘制器把 Icon 值当成 Object 值处理——只是绘制出其 toString()方法返回的字符串。

为了让默认的单元格绘制器可以将 Icon 类型的值绘制成图标，把 Boolean 类型的值绘制成复选框，创建 JTable 时所使用的 TableModel 绝不能采用默认的 TableModel，必须采用扩展后的 TableModel 类，如下所示。

```
// 定义一个 DefaultTableModel 类的子类
class ExtendedTableModel extends DefaultTableModel
{
    ...
    // 重写 getColumnClass 方法，根据每列的第一个值来返回每列真实的数据类型
    public Class getColumnClass(int c)
    {
        return getValueAt(0 , c).getClass();
    }
}
```

提供了上面的 ExtendedTableModel 类之后，程序应该先创建 ExtendedTableModel 对象，再利用该对象来创建 JTable，这样就可以保证 JTable 的 model 对象的 getColumnClass()方法会返回每列真实的数据类型，默认的单元格绘制器就会将 Icon 类型的单元格值绘制成图标，将 Boolean 类型的单元格值绘制成复选框。

如果希望程序采用自己定制的单元格绘制器，则必须实现自己的单元格绘制器，单元格绘制器必须实现 TableCellRenderer 接口。与前面的 TreeCellRenderer 接口完全相似，该接口里也只包含一个 getTableCellRendererComponent()方法，该方法返回的 Component 将会作为指定单元格绘制的组件。

提示：

Swing 提供了一致的编程模型，不管是 JList、JTree 还是 JTable，它们所使用的单元格绘制器都有一致的编程模型，分别需要扩展 ListCellRenderer、TreeCellRenderer 或 TableCellRenderer，扩展这三个基类时都需要重写 getXxxCellRendererComponent()方法，该方法的返回值将作为被绘制的组件。

一旦实现了自己的单元格绘制器之后，还必须将该单元格绘制器安装到指定的 JTable 对象上，为指定的 JTable 对象安装单元格绘制器有如下两种方式。

- 局部方式（列级）：调用 TableColumn 的 setCellRenderer()方法为指定列安装指定的单元格绘制器。
- 全局方式（表级）：调用 JTable 的 setDefaultRenderer()方法为指定的 JTable 对象安装单元格绘制器。setDefaultRenderer()方法需要传入两个参数，即列类型和单元格绘制器，表明指定类型的数

据列才会使用该单元格绘制器。

注意：

当某一列既符合全局绘制器的规则，又符合局部绘制器的规则时，局部绘制器将会负责绘制该单元格，全局绘制器不会产生任何作用。除此之外，TableColumn还包含了一个setHeaderRenderer()方法，该方法可以为指定列的列头安装单元格绘制器。

下面程序提供了一个ExtendedTableModel类，该类扩展了DefaultTableModel，重写了父类的getColumnClass()方法，该方法根据每列的第一个值来决定该列的数据类型；下面程序还提供了一个定制的单元格绘制器，它使用图标来形象地表明每个好友的性别。

程序清单：codes\12\12.11\TableCellRendererTest.java

```
public class TableCellRendererTest
{
    JFrame jf = new JFrame("使用单元格绘制器");
    JTable table;
    // 定义二维数组作为表格数据
    Object[][] tableData =
    {
        new Object[]{"李清照" , 29 , "女"
            , new ImageIcon("icon/3.gif") , true},
        new Object[]{"苏格拉底", 56 , "男"
            , new ImageIcon("icon/1.gif") , false},
        new Object[]{"李白", 35 , "男"
            , new ImageIcon("icon/4.gif") , true},
        new Object[]{"弄玉", 18 , "女"
            , new ImageIcon("icon/2.gif") , true},
        new Object[]{"虎头" , 2 , "男"
            , new ImageIcon("icon/5.gif") , false}
    };
    // 定义一维数据作为列标题
    String[] columnTitle = {"姓名" , "年龄" , "性别"
        , "主头像" , "是否中国人"};
    public void init()
    {
        // 以二维数组和一维数组来创建一个 ExtendedTableModel 对象
        ExtendedTableModel model = new ExtendedTableModel(columnTitle
            , tableData);
        // 以 ExtendedTableModel 来创建 JTable
        table = new JTable( model);
        table.setRowSelectionAllowed(false);
        table.setRowHeight(40);
        // 获取第三列
        TableColumn lastColumn = table.getColumnModel().getColumn(2);
        // 对第三列采用自定义的单元格绘制器
        lastColumn.setCellRenderer(new GenderTableCellRenderer());
        // 将 JTable 对象放在 JScrollPane 中，并将该 JScrollPane 显示出来
        jf.add(new JScrollPane(table));
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new TableCellRendererTest().init();
    }
}
class ExtendedTableModel extends DefaultTableModel
{
    // 重新提供一个构造器，该构造器的实现委托给 DefaultTableModel 父类
    public ExtendedTableModel(String[] columnNames , Object[][] cells)
    {
        super(cells , columnNames);
    }
```

```
    // 重写 getColumnClass 方法，根据每列的第一个值来返回其真实的数据类型
    public Class getColumnClass(int c)
    {
        return getValueAt(0 , c).getClass();
    }
}
// 定义自定义的单元格绘制器
class GenderTableCellRenderer extends JPanel
    implements TableCellRenderer
{
    private String cellValue;
    // 定义图标的宽度和高度
    final int ICON_WIDTH = 23;
    final int ICON_HEIGHT = 21;
    public Component getTableCellRendererComponent(JTable table
        , Object value , boolean isSelected , boolean hasFocus
        , int row , int column)
    {
        cellValue = (String)value;
        // 设置选中状态下绘制边框
        if (hasFocus)
        {
            setBorder(UIManager.getBorder("Table.focusCellHighlightBorder"));
        }
        else
        {
            setBorder(null);
        }
        return this;
    }
    // 重写 paint()方法，负责绘制该单元格内容
    public void paint(Graphics g)
    {
        // 如果表格值为"男"或"male"，则绘制一个男性图标
        if (cellValue.equalsIgnoreCase("男")
            || cellValue.equalsIgnoreCase("male"))
        {
            drawImage(g , new ImageIcon("icon/male.gif").getImage());
        }
        // 如果表格值为"女"或"female"，则绘制一个女性图标
        if (cellValue.equalsIgnoreCase("女")
            || cellValue.equalsIgnoreCase("female"))
        {
            drawImage(g , new ImageIcon("icon/female.gif").getImage());
        }
    }
    // 绘制图标的方法
    private void drawImage(Graphics g , Image image)
    {
        g.drawImage(image, (getWidth() - ICON_WIDTH ) / 2
            , (getHeight() - ICON_HEIGHT) / 2 , null);
    }
}
```

上面程序中没有直接使用二维数组和一维数组来创建 JTable 对象，而是采用 ExtendedTableModel 对象来创建 JTable 对象（如第一段粗体字代码所示）。ExtendedTableModel 类重写了父类的 getColumnClass()方法，该方法将会根据每列实际的值来返回该列的类型（如第二段粗体字代码所示）。

程序提供了一个 GenderTableCellRenderer 类，该类实现了 TableCellRenderer 接口，可以作为单元格绘制器使用。该类继承了 JPanel 容器，重写 getTableCellRendererComponent()方法时返回 this，这表明它会使用 JPanel 对象作为单元格绘制器。

提示：

读者可以将 ExtendedTableModel 补充得更加完整——主要是将 DefaultTableModel 中的几个构造器重新暴露出来，以后程序中可以使用 ExtendedTableModel 类作为 JTable 的 model 类，这样创建的 JTable 就可以将 Icon 列、Boolean 列绘制成图标和复选框。

运行上面程序，会看到如图 12.54 所示的效果。

图 12.54　重写 getColumnClass()方法和定制单元格绘制器

12.11.6　编辑单元格内容

如果用户双击 JTable 表格的指定单元格，系统将会开始编辑该单元格的内容。在默认情况下，系统会使用文本框来编辑该单元格的内容，包括如图 12.54 所示表格的图标单元格。与此类似的是，如果用户双击 JTree 的节点，默认也会采用文本框来编辑节点的内容。

但如果单元格内容不是文字内容，而是如图 12.54 所示的图形类型时，用户当然不希望使用文本编辑器来编辑该单元格的内容，因为这种编辑方式非常不直观，用户体验相当差。为了避免这种情况，可以实现自己的单元格编辑器，从而可以给用户提供更好的操作界面。

实现 JTable 的单元格编辑器应该实现 TableCellEditor 接口，实现 JTree 的节点编辑器需要实现 TreeCellEditor 接口，这两个接口有非常紧密的联系。它们有一个共同的父接口：CellEditor；而且它们有一个共同的实现类：DefaultCellEditor。关于 TableCellEditor 和 TreeCellEditor 两个接口及其实现类之间的关系如图 12.55 所示。

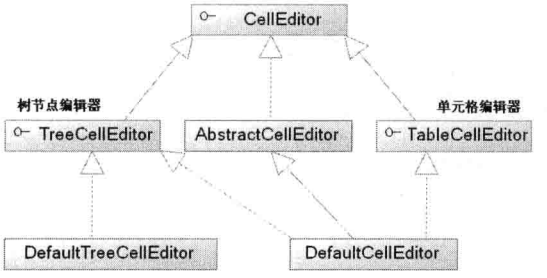


图 12.55　TableCellEditor 和 TreeCellEditor 的关系

从图 12.55 中可以看出，Swing 为 TableCellEditor 提供了 DefaultCellEditor 实现类（也可作为 TreeCellEditor 的实现类），DefaultCellEditor 类有三个构造器，它们分别使用文本框、复选框和 JComboBox 作为单元格编辑器，其中使用文本框编辑器是最常见的情形，如果单元格的值是 Boolean 类型，则系统默认使用复选框编辑器（如图 12.54 中最右边一列所示），这两种情形都是前面见过的情形。如果想指定某列使用 JComboBox 作为单元格编辑器，则需要显式创建 JComboBox 实例，然后以此实例来创建 DefaultCellEditor 编辑器。

实现 TableCellEditor 接口可以开发自己的单元格编辑器，但这种做法比较烦琐；通常会使用扩展 DefaultCellEditor 类的方式，这种方式比较简单。TableCellEditor 接口定义了一个 getTableCellEditorComponent()方法，该方法返回一个 Component 对象，该对象就是该单元格的编辑器。

一旦实现了自己的单元格编辑器，就可以为 JTable 对象安装该单元格编辑器，与安装单元格绘制器类似，安装单元格编辑器也有两种方式。

- 局部方式（列级）：为特定列指定单元格编辑器，通过调用 TableColumn 的 setCellEditor()方法为该列安装单元格编辑器。
- 全局方式（表级）：调用 JTable 的 setDefaultEditor()方法为该表格安装默认的单元格编辑器。该方法需要两个参数，即列类型和单元格编辑器，这两个参数表明对于指定类型的数据列使用该单元格编辑器。

与单元格绘制器相似的是，如果有一列同时满足列级单元格编辑器和表级单元格编辑器的要求，系统将采用列级单元格编辑器。

下面程序实现了一个 ImageCellEditor 编辑器，该编辑器由一个不可直接编辑的文本框和一个按钮组成，当用户单击该按钮时，该编辑器弹出一个文件选择器，方便用户选择图标文件。除此之外，下面程序还创建了一个基于 JComboBox 的 DefaultCellEditor 类，该编辑器允许用户通过下拉列表来选择图标。

程序清单：codes\12\12.11\TableCellEditorTest.java

```
public class TableCellEditorTest
{
    JFrame jf = new JFrame("使用单元格编辑器");
```

```
    JTable table;
    // 定义二维数组作为表格数据
    Object[][] tableData =
    {
        new Object[]{"李清照" , 29 , "女" , new ImageIcon("icon/3.gif")
            , new ImageIcon("icon/3.gif") , true},
        new Object[]{"苏格拉底", 56 , "男" , new ImageIcon("icon/1.gif")
            , new ImageIcon("icon/1.gif") , false},
        new Object[]{"李白", 35 , "男" , new ImageIcon("icon/4.gif")
            , new ImageIcon("icon/4.gif") , true},
        new Object[]{"弄玉", 18 , "女" , new ImageIcon("icon/2.gif")
            , new ImageIcon("icon/2.gif") , true},
        new Object[]{"虎头" , 2 , "男" , new ImageIcon("icon/5.gif")
            , new ImageIcon("icon/5.gif") , false}
    };
    // 定义一维数据作为列标题
    String[] columnTitle = {"姓名" , "年龄" , "性别" , "主头像"
        , "次头像" , "是否中国人"};
    public void init()
    {
        // 以二维数组和一维数组来创建一个 ExtendedTableModel 对象
        ExtendedTableModel model = new ExtendedTableModel(
            columnTitle , tableData);
        // 以 ExtendedTableModel 来创建 JTable
        table = new JTable(model);
        table.setRowSelectionAllowed(false);
        table.setRowHeight(40);
        // 为该表格指定默认的编辑器
        table.setDefaultEditor(ImageIcon.class, new ImageCellEditor());
        // 获取第 5 列
        TableColumn lastColumn = table.getColumnModel().getColumn(4);
        // 创建 JComboBox 对象，并添加多个图标列表项
        JComboBox<ImageIcon> editCombo = new JComboBox<>();
        for (int i = 1; i <= 10; i++)
        {
            editCombo.addItem(new ImageIcon("icon/" + i + ".gif"));
        }
        // 设置第 5 列使用基于 JComboBox 的 DefaultCellEditor
        lastColumn.setCellEditor(new DefaultCellEditor(editCombo));
        // 将 JTable 对象放在 JScrollPane 中，并将该 JScrollPane 放在窗口中显示出来
        jf.add(new JScrollPane(table));
        jf.pack();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    public static void main(String[] args)
    {
        new TableCellEditorTest().init();
    }
}
class ExtendedTableModel extends DefaultTableModel
{
    // 重新提供一个构造器，该构造器的实现委托给 DefaultTableModel 父类
    public ExtendedTableModel(String[] columnNames , Object[][] cells)
    {
        super(cells , columnNames);
    }
    // 重写 getColumnClass 方法，根据每列的第一个值返回该列真实的数据类型
    public Class getColumnClass(int c)
    {
        return getValueAt(0 , c).getClass();
    }
}
// 扩展 DefaultCellEditor 来实现 TableCellEditor 类
class ImageCellEditor extends DefaultCellEditor
{
    // 定义文件选择器
    private JFileChooser fDialog = new JFileChooser(); ;
    private JTextField field = new JTextField(15);
    private JButton button = new JButton("...");
```

```
    public ImageCellEditor()
    {
        // 因为 DefaultCellEditor 没有无参数的构造器
        // 所以这里显式调用父类有参数的构造器
        super(new JTextField());
        initEditor();
    }
    private void initEditor()
    {
        field.setEditable(false);
        // 为按钮添加监听器，当用户单击该按钮时
        // 系统将出现一个文件选择器让用户选择图标文件
        button.addActionListener(e -> browse());
        // 为文件选择器安装文件过滤器
        fDialog.addChoosableFileFilter(new FileFilter()
        {
            public boolean accept(File f)
            {
                if (f.isDirectory())
                {
                    return true;
                }
                String extension = Utils.getExtension(f);
                if (extension != null)
                {
                    if (extension.equals(Utils.tiff)
                        || extension.equals(Utils.tif)
                        || extension.equals(Utils.gif)
                        || extension.equals(Utils.jpeg)
                        || extension.equals(Utils.jpg)
                        || extension.equals(Utils.png))
                    {
                        return true;
                    }
                    else
                    {
                        return false;
                    }
                }
                return false;
            }
            public String getDescription()
            {
                return "有效的图片文件";
            }
        });
        fDialog.setAcceptAllFileFilterUsed(false);
    }
    // 重写 TableCellEditor 接口的 getTableCellEditorComponent 方法
    // 该方法返回单元格编辑器，该编辑器是一个 JPanel
    // 该容器包含一个文本框和一个按钮
    public Component getTableCellEditorComponent(JTable table
        , Object value , boolean isSelected , int row , int column)  // ①
    {
        this.button.setPreferredSize(new Dimension(20, 20));
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        field.setText(value.toString());
        panel.add(this.field, BorderLayout.CENTER);
        panel.add(this.button, BorderLayout.EAST);
        return panel;
    }
    public Object getCellEditorValue()
    {
        return new ImageIcon(field.getText());
    }
    private void browse()
    {
        // 设置、打开文件选择器
        fDialog.setCurrentDirectory(new File("icon"));
```

```
            int result = fDialog.showOpenDialog(null);
            // 如果单击了文件选择器的“取消”按钮
            if (result == JFileChooser.CANCEL_OPTION)
            {
                // 取消编辑
                super.cancelCellEditing();
                return;
            }
            // 如果单击了文件选择器的“确定”按钮
            else
            {
                // 设置 field 的内容
                field.setText("icon/" + fDialog.getSelectedFile().getName());
            }
        }
    }
    class Utils
    {
        public final static String jpeg = "jpeg";
        public final static String jpg = "jpg";
        public final static String gif = "gif";
        public final static String tiff = "tiff";
        public final static String tif = "tif";
        public final static String png = "png";
        // 获取文件扩展名的方法
        public static String getExtension(File f)
        {
            String ext = null;
            String s = f.getName();
            int i = s.lastIndexOf('.');
            if (i > 0 &&  i < s.length() - 1)
            {
                ext = s.substring(i + 1).toLowerCase();
            }
            return ext;
        }
    }
```

上面程序中实现了一个 ImageCellEditor 编辑器，程序中的粗体字代码将该单元格编辑器注册成 ImageIcon 类型的单元格编辑器，如果某一列的数据类型是 ImageIcon，则默认使用该单元格编辑器。ImageCellEditor 扩展了 DefaultCellEditor 基类，重写 getTableCellEditorComponent()方法返回一个 JPanel，该 JPanel 里包含一个文本框和一个按钮。

除此之外，程序中的粗体字代码还为最后一列安装了一个基于 JComboBox 的 DefaultCellEditor。

运行上面程序，双击倒数第 3 列的任意单元格，开始编辑该单元格，将看到如图 12.56 所示的窗口。

双击第 5 列的任意单元格，开始编辑该单元格，将看到如图 12.57 所示的窗口。

图 12.56　自定义单元格编辑器

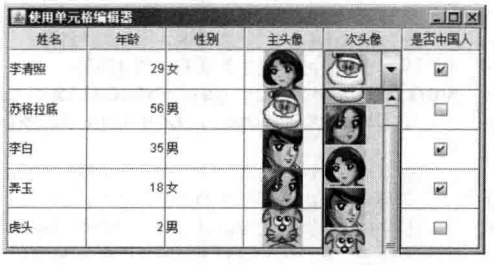


图 12.57　基于 JComboBox 的 DefaultCellEditor

通过图 12.56 和图 12.57 可以看出，如果单元格的值需要从多个枚举值之中选择，则使用 DefaultCellEditor 即可。使用自定义的单元格编辑器则非常灵活，可以取得单元格编辑器的全部控制权。

12.12　使用 JFormattedTextField 和 JTextPane 创建格式文本

Swing 使用 JTextComponent 作为所有文本输入组件的父类，从图 12.1 中可以看出，Swing 为该类提供了三个子类：JTextArea、JTextField 和 JEditorPane，并为 JEditorPane 提供了一个 JTextPane 子类，

JEditorPane 和 JTextPane 是两个典型的格式文本编辑器，也是本节介绍的重点。JTextArea 和 JTextField 是两个常见的文本组件，比较简单，本节不会再次介绍它们。

JTextField 派生了两个子类：JPasswordField 和 JFormattedTextField，它们代表密码输入框和格式化文本输入框。

与其他的 Swing 组件类似，所有的文本输入组件也遵循了 MVC 的设计模式，即每个文本输入组件都有对应的 model 来保存其状态数据；与其他的 Swing 组件不同的是，文本输入组件的 model 接口不是 XxxModel 接口，而是 Document 接口，Document 既包括有格式的文本，也包括无格式的文本。不同的文本输入组件对应的 Document 不同。

12.12.1 监听 Document 的变化

如果希望检测到任何文本输入组件里所输入内容的变化，则可以通过监听该组件对应的 Document 来实现。JTextComponent 类里提供了一个 getDocument()方法，该方法用于获取所有文本输入组件对应的 Document 对象。

Document 提供了一个 addDocumentListener()方法来为 Document 添加监听器，该监听器必须实现 DocumentListener 接口，该接口里提供了如下三个方法。

- changedUpdate(DocumentEvent e)：当 Document 里的属性或属性集发生了变化时触发该方法。
- insertUpdate(DocumentEvent e)：当向 Document 中插入文本时触发该方法。
- removeUpdate(DocumentEvent e)：当从 Document 中删除文本时触发该方法。

对于上面的三个方法而言，如果仅需要检测文本的变化，则无须实现第一个方法。但 Swing 并没有为 DocumentListener 接口提供适配器（难道是 Oracle 的疏忽），所以程序依然要为第一个方法提供空实现。

除此之外，还可以为文件输入组件添加一个撤销监听器，这样就允许用户撤销以前的修改。添加撤销监听器的方法是 addUndoableEditListener()，该方法需要接收一个 UndoableEditListener 监听器，该监听器里包含了 undoableEditHappened()方法，当文档里发生了可撤销的编辑操作时将会触发该方法。

下面程序示范了如何为一个普通文本域的 Document 添加监听器，当用户在目标文本域里输入、删除文本时，程序会显示出用户所做的修改。该文本域还支持撤销操作，当用户按“Ctrl+Z”键时，该文本域会撤销用户刚刚输入的内容。

程序清单：codes\12\12.12\MonitorText.java

```
public class MonitorText
{
    JFrame mainWin = new JFrame("监听 Document 对象");
    JTextArea target = new JTextArea(4, 35);
    JTextArea msg = new JTextArea(5, 35);
    JLabel label = new JLabel("文本域的修改信息");
    Document doc = target.getDocument();
    // 保存撤销操作的 List 对象
    LinkedList<UndoableEdit> undoList = new LinkedList<>();
    // 最多允许撤销多少次
    final int UNDO_COUNT = 20;
    public void init()
    {
        msg.setEditable(false);
        // 添加 DocumentListener
        doc.addDocumentListener(new DocumentListener()
        {
            // 当 Document 的属性或属性集发生了变化时触发该方法
            public void changedUpdate(DocumentEvent e){}
            // 当向 Document 中插入文本时触发该方法
            public void insertUpdate(DocumentEvent e)
            {
                int offset = e.getOffset();
                int len = e.getLength();
                // 取得插入事件的位置
                msg.append("插入文本的长度：" + len + "\n");
```

```
            msg.append("插入文本的起始位置：" + offset + "\n");
            try
            {
                msg.append("插入文本内容："
                    + doc.getText(offset, len) + "\n");
            }
            catch (BadLocationException evt)
            {
                evt.printStackTrace();
            }
        }
        // 当从 Document 中删除文本时触发该方法
        public void removeUpdate(DocumentEvent e)
        {
            int offset = e.getOffset();
            int len = e.getLength();
            // 取得插入事件的位置
            msg.append("删除文本的长度：" + len + "\n");
            msg.append("删除文本的起始位置：" + offset + "\n");
        }
    });
    // 添加可撤销操作的监听器
    doc.addUndoableEditListener(e -> {
        // 每次发生可撤销操作时都会触发该代码块      // ①
        UndoableEdit edit = e.getEdit();
        if (edit.canUndo() && undoList.size() < UNDO_COUNT)
        {
            // 将撤销操作装入 List 内
            undoList.add(edit);
        }
        // 已经达到了最大撤销次数
        else if (edit.canUndo() && undoList.size() >= UNDO_COUNT)
        {
            // 弹出第一个撤销操作
            undoList.pop();
            // 将撤销操作装入 List 内
            undoList.add(edit);
        }
    });
    // 为“Ctrl+Z”按键添加监听器
    target.addKeyListener(new KeyAdapter()
    {
        public void keyTyped(KeyEvent e)    // ②
        {
            // 如果按键是“Ctrl + Z”
            if (e.getKeyChar() == 26)
            {
                if (undoList.size() > 0)
                {
                    // 移出最后一个可撤销操作，并取消该操作
                    undoList.removeLast().undo();
                }
            }
        }
    });
    Box box = new Box(BoxLayout.Y_AXIS);
    box.add(new JScrollPane(target));
    JPanel panel = new JPanel();
    panel.add(label);
    box.add(panel);
    box.add(new JScrollPane(msg));
    mainWin.add(box);
    mainWin.pack();
    mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    mainWin.setVisible(true);
}
public static void main(String[] args) throws Exception
{
```

```
        new MonitorText().init();
    }
}
```

上面程序中的两段粗体字代码实现了 Document 中插入文本、删除文本的事件处理器，当用户向 Document 中插入文本、删除文本时，程序将会把这些修改信息添加到下面的一个文本域里。

程序中①号粗体字代码是可撤销操作的事件处理器，当用户在该文本域内进行可撤销操作时，这段代码将会被触发，这段代码把用户刚刚进行的可撤销操作以 List 保存起来，以便在合适的时候撤销用户所做的修改。

程序中②号粗体字代码主要用于为“Ctrl+Z”按键添加按键监听器，当用户按下“Ctrl+Z”键时，程序从保存可撤销操作的 List 中取出最后一个可撤销操作，并撤销该操作的修改。

运行上面程序，会看到如图 12.58 所示的运行结果。

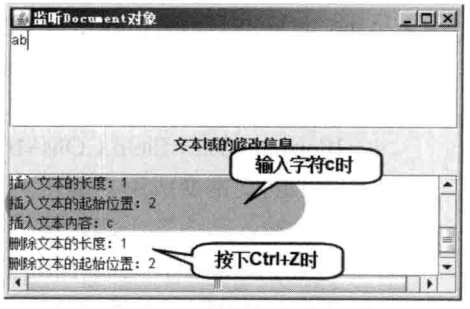


图 12.58 为 Document 添加监听器

12.12.2 使用 JPasswordField

JPasswordField 是 JTextField 的一个子类，它是 Swing 的 MVC 设计的产品——JPasswordField 和 JTextField 的各种特征几乎完全一样，只是当用户向 JPasswordField 输入内容时，JPasswordField 并不会显示出用户输入的内容，而是以 echo 字符（通常是星号和黑点）来代替用户输入的所有字符。

JPasswordField 和 JTextField 的用法几乎完全一样，连构造器的个数和参数都完全一样。但是 JPasswordField 多了一个 setEchoChar(Char ch)方法，该方法用于设置该密码框的 echo 字符——当用户在密码输入框内输入时，每个字符都会使用该 echo 字符代替。

除此之外，JPasswordField 重写了 JTextComponent 的 getText()方法，并且不再推荐使用 getText()方法返回字符串密码框的字符串，因为 getText()方法所返回的字符串会一直停留在虚拟机中，直到垃圾回收，这可能导致存在一些安全隐患，所以 JPasswordField 提供了一个 getPassword()方法，该方法返回一个字符数组，而不是返回字符串，从而提供了更好的安全机制。

注意：

当程序使用完 getPassword()方法返回的字符数组后，应该立即清空该字符数组的内容，以防该数组泄露密码信息。

12.12.3 使用 JFormattedTextField

在有些情况下，程序不希望用户在输入框内随意地输入，例如，程序需要用户输入一个有效的时间，或者需要用户输入一个有效的物品价格，如果用户输入不合理，程序应该阻止用户输入。对于这种需求，通常的做法是为该文本框添加失去焦点的监听器，再添加回车按键的监听器，当该文本框失去焦点时，或者该用户在该文本框内按回车键时，就检测用户输入是否合法。这种做法基本可以解决该问题，但编程比较烦琐！Swing 提供的 JFormattedTextField 可以更优雅地解决该问题。

使用 JFormattedTextField 与使用普通文本行有一个区别——它需要指定一个文本格式，只有当用户的输入满足该格式时，JFormattedTextField 才会接收用户输入。JFormattedTextField 可以使用如下两种类型的格式。

- JFormattedTextField.AbstractFormatter：该内部类有一个子类 DefaultFormatter，而 DefaultFormatter 又有一个非常实用的 MaskFormatter 子类，允许程序以掩码的形式指定文本格式。
- Format：主要由 DateFormat 和 NumberFormat 两个格式器组成，这两个格式器可以指定 JFormattedTextField 所能接收的格式字符串。

创建 JFormattedTextField 对象时可以传入上面任意一个格式器，成功地创建了 JFormattedTextField

对象之后，JFormattedTextField 对象的用法和普通 TextField 的用法基本相似，一样可以调用 setColumns()来设置该文本框的宽度，调用 setFont()来设置该文本框内的字体等。除此之外，JFormattedTextField 还包含如下三个特殊方法。

- Object getValue()：获取该格式化文本框里的值。
- void setValue(Object obj)：设置该格式化文本框的初始值。
- void setFocusLostBehavior(int behavior)：设置该格式化文本框失去焦点时的行为，该方法可以接收如下 4 个值。
  - JFormattedTextField.COMMIT：如果用户输入的内容满足格式器的要求，则该格式化文本框显示的文本变成用户输入的内容，调用 getValue()方法返回的是该文本框内显示的内容；如果用户输入的内容不满足格式器的要求，则该格式化文本框显示的依然是用户输入的内容，但调用 getValue()方法返回的不是该文本框内显示的内容，而是上一个满足要求的值。
  - JFormattedTextField.COMMIT_OR_REVERT：这是默认值。如果用户输入的内容满足格式器的要求，则该格式化文本框显示的文本、getValue()方法返回的都是用户输入的内容；如果用户输入的内容不满足格式器的要求，则该格式化文本框显示的文本、getValue()方法返回的都是上一个满足要求的值。
  - JFormattedTextField.PERSIST：不管用户输入的内容是否满足格式器的要求，该格式化文本框都显示用户输入的内容，getValue()方法返回的都是上一个满足要求的值。
  - JFormattedTextField.REVERT：不管用户输入的内容是否满足格式器的要求，该格式化文本框显示的内容、getValue()方法返回的都是上一个满足要求的值。在这种情况下，不管用户输入什么内容对该文本框都没有任何影响。

上面三个方法中获取格式化文本框内容的方法返回 Object 类型，而不是返回 String 类型；与之对应的是，设置格式化文本框初始值的方法需要传入 Object 类型参数，而不是 String 类型参数，这都是因为格式化文本框会将文本框内容转换成指定格式对应的对象，而不再是普通字符串。

DefaultFormatter 是一个功能非常强大的格式器，它可以格式化任何类的实例，只要该类包含一个带一个字符串参数的构造器，并提供对应的 toString()方法（该方法的返回值就是传入给构造器字符串参数的值）即可。

例如，URL 类包含一个 URL(String spec)构造器，且 URL 对象的 toString()方法恰好返回刚刚传入的 spec 参数，因此可以使用 DefaultFormatter 来格式化 URL 对象。当格式化文本框失去焦点时，该格式器就会调用带一个字符串参数的构造器来创建新的对象，如果构造器抛出了异常，即表明用户输入无效。

注意：

DefaultFormatter 格式器默认采用改写方式来处理用户输入，即当用户在格式化文本框内输入时，每输入一个字符就会替换文本框内原来的一个字符。如果想关闭这种改写方式，采用插入方式，则可通过调用它的 setOverwriteMode(false)方法来实现。

MaskFormatter 格式器的功能有点类似于正则表达式，它要求用户在格式化文本框内输入的内容必须匹配一定的掩码格式。例如，若要匹配广州地区的电话号码，则可采用 020-########的格式，这个掩码字符串和正则表达式有一定的区别，因为该掩码字符串只支持如下通配符。

- #：代表任何有效数字。
- '：转义字符，用于转义具有特殊格式的字符。例如，若想匹配#，则应该写成'#。
- U：任何字符，将所有小写字母映射为大写。
- L：任何字符，将所有大写字母映射为小写。
- A：任何字符或数字。
- ?：任何字符。

- *：可以匹配任何内容。
- H：任何十六进制字符（0~9、a~f 或 A~F）。

值得指出的是，格式化文本框内的字符串总是和掩码具有相同的格式，连长度也完全相同。如果用户删除了格式化文本框内的字符，这些被删除的字符将由占位符替代。默认使用空格作为占位符，当然也可以调用 MaskFormatter 的 setPlaceholderCharacter()方法来设置该格式器的占位符。例如如下代码：

```
formatter.setPlaceholderCharacter('□');
```

下面程序示范了关于 JFormattedTextField 的简单用法。

程序清单：codes\12\12.12\JFormattedTextFieldTest.java

```
public class JFormattedTextFieldTest
{
    private JFrame mainWin = new JFrame("测试格式化文本框");
    private JButton okButton = new JButton("确定");
    // 定义用于添加格式化文本框的容器
    private JPanel mainPanel = new JPanel();
    JFormattedTextField[] fields = new JFormattedTextField[6];
    String[] behaviorLabels = new String[]
    {
        "COMMIT",
        "COMMIT_OR_REVERT",
        "PERSIST",
        "REVERT"
    };
    int[] behaviors = new int[]
    {
        JFormattedTextField.COMMIT,
        JFormattedTextField.COMMIT_OR_REVERT,
        JFormattedTextField.PERSIST,
        JFormattedTextField.REVERT
    };
    ButtonGroup bg = new ButtonGroup();
    public void init()
    {
        // 添加按钮
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(okButton);
        mainPanel.setLayout(new GridLayout(0, 3));
        mainWin.add(mainPanel, BorderLayout.CENTER);
        // 使用 NumberFormat 的 integerInstance 创建一个 JformattedTextField 对象
        fields[0] = new JFormattedTextField(NumberFormat
            .getIntegerInstance());
        // 设置初始值
        fields[0].setValue(100);
        addRow("整数格式文本框 :", fields[0]);
        // 使用 NumberFormat 的 currencyInstance 创建一个 JFormattedTextField 对象
        fields[1] = new JFormattedTextField(NumberFormat
            .getCurrencyInstance());
        fields[1].setValue(100.0);
        addRow("货币格式文本框:", fields[1]);
        // 使用默认的日期格式创建一个 JFormattedTextField 对象
        fields[2] = new JFormattedTextField(DateFormat.getDateInstance());
        fields[2].setValue(new Date());
        addRow("默认的日期格式器:", fields[2]);
        // 使用 SHORT 类型的日期格式创建一个 JFormattedTextField 对象
        // 且要求采用严格日期格式
        DateFormat format = DateFormat.getDateInstance(DateFormat.SHORT);
        // 要求采用严格的日期格式语法
        format.setLenient(false);
        fields[3] = new JFormattedTextField(format);
        fields[3].setValue(new Date());
        addRow("SHORT 类型的日期格式器（语法严格）:", fields[3]);
        try
        {
            // 创建默认的 DefaultFormatter 对象
            DefaultFormatter formatter = new DefaultFormatter();
```

```
            // 关闭 overwrite 状态
            formatter.setOverwriteMode(false);
            fields[4] = new JFormattedTextField(formatter);
            // 使用 DefaultFormatter 来格式化 URL
            fields[4].setValue(new URL("http://www.crazyit.org"));
            addRow("URL:", fields[4]);
        }
        catch (MalformedURLException e)
        {
            e.printStackTrace();
        }
        try
        {
            MaskFormatter formatter = new MaskFormatter("020-########");
            // 设置占位符
            formatter.setPlaceholderCharacter('□');
            fields[5] = new JFormattedTextField(formatter);
            // 设置初始值
            fields[5].setValue("020-28309378");
            addRow("电话号码：", fields[5]);
        }
        catch (ParseException ex)
        {
            ex.printStackTrace();
        }

        JPanel focusLostPanel = new JPanel();
        // 采用循环方式加入失去焦点行为的单选按钮
        for (int i = 0; i < behaviorLabels.length ; i++ )
        {
            final int index = i;
            final JRadioButton radio = new JRadioButton(behaviorLabels[i]);
            // 默认选中第二个单选按钮
            if (i == 1)
            {
                radio.setSelected(true);
            }
            focusLostPanel.add(radio);
            bg.add(radio);
            // 为所有的单选按钮添加事件监听器
            radio.addActionListener(e -> {
                // 如果当前该单选按钮处于选中状态
                if (radio.isSelected())
                {
                    // 设置所有的格式化文本框失去焦点的行为
                    for (int j = 0 ; j < fields.length ; j++)
                    {
                      fields[j].setFocusLostBehavior(behaviors[index]);
                    }
                }
            });
        }
        focusLostPanel.setBorder(new TitledBorder(new EtchedBorder(),
            "请选择焦点失去后的行为"));
        JPanel p = new JPanel();
        p.setLayout(new BorderLayout());
        p.add(focusLostPanel , BorderLayout.NORTH);
        p.add(buttonPanel , BorderLayout.SOUTH);

        mainWin.add(p , BorderLayout.SOUTH);
        mainWin.pack();
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.setVisible(true);
    }
    // 定义添加一行格式化文本框的方法
    private void addRow(String labelText, final JFormattedTextField field)
    {
        mainPanel.add(new JLabel(labelText));
        mainPanel.add(field);
        final JLabel valueLabel = new JLabel();
```

```
        mainPanel.add(valueLabel);
        // 为“确定”按钮添加事件监听器
        // 当用户单击“确定”按钮时，文本框后显示文本框的值
        okButton.addActionListener(event -> {
            Object value = field.getValue();
            // 输出格式化文本框的值
            valueLabel.setText(value.toString());
        });
    }
    public static void main(String[] args)
    {
        new JFormattedTextFieldTest().init();
    }
}
```

上面程序添加了 6 个格式化文本框，其中两个是基于 NumberFormat 生成的整数格式器、货币格式器，两个是基于 DateFormat 生成的日期格式器，一个是使用 DefaultFormatter 创建的 URL 格式器，最后一个是使用 MaskFormatter 创建的掩码格式器，程序中的粗体字代码是创建这些格式器的关键代码。

除此之外，程序还添加了 4 个单选按钮，用于控制这些格式化文本框失去焦点后的行为。运行上面程序，并选中“COMMIT”行为，将看到如图 12.59 所示的界面。

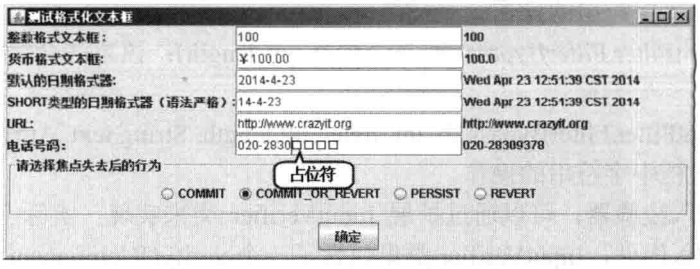


图 12.59 COMMIT 行为下的格式化文本框

从图 12.59 中可以看出，虽然用户向格式化文本框内输入的内容与该文本框所要求的格式不符，但该文本框依然显示了用户输入的内容，只是后面显示该文本框的 getValue()方法返回值时看到的依然是 100，即上一个符合格式的值。

大部分时候，使用基于 Format 的格式器，DefaultFormatter 和 MaskFormatter 已经能满足绝大部分要求；但对于一些特殊的要求，则可以采用扩展 DefaultFormatter 的方式来定义自己的格式器。定义自己的格式器通常需要重写如下两个方法。

- Object stringToValue(String string)：根据格式化文本框内的字符串来创建符合指定格式的对象。
- String valueToString(Object value)：将符合格式的对象转换成文本框中显示的字符串。

例如，若需要创建一个只能接收 IP 地址的格式化文本框，则可以创建一个自定义的格式化文本框，因为 IP 地址是由 4 个 0~255 之间的整数表示的，所以程序采用长度为 4 的 byte[]数组来保存 IP 地址。程序可以采用如下方法将用户输入的字符串转换成 byte[]数组。

```
public Object stringToValue(String text) throws ParseException
{
    // 将格式化文本框内的字符串以点号（.）分成 4 节
    String[] nums = text.split("\\.");
    if (nums.length != 4)
    {
        throw new ParseException("IP 地址必须是 4 个整数", 0);
    }
    byte[] a = new byte[4];
    for (int i = 0; i < 4; i++)
    {
        int b = 0;
        try
        {
            b = Integer.parseInt(nums[i]);
        }
        catch (NumberFormatException e)
```

```
            {
                throw new ParseException("IP地址必须是整数", 0);
            }
            if (b < 0 || b >= 256)
            {
                throw new ParseException("IP地址值只能在0~255之间", 0);
            }
            a[i] = (byte) b;
        }
        return a;
    }
```

除此之外，Swing还提供了如下两种机制来保证用户输入的有效性。

- 输入过滤：输入过滤机制允许程序拦截用户的插入、替换、删除等操作，并改变用户所做的修改。
- 输入校验：输入验证机制允许用户离开输入组件时，验证机制自动触发——如果用户输入不符合要求，校验器强制用户重新输入。

输入过滤器需要继承DocumentFilter类，程序可以重写该类的如下三个方法来拦截用户的插入、删除和替换等操作。

- insertString(DocumentFilter.FilterBypass fb, int offset, String string, AttributeSet attr)：该方法会拦截用户向文档中插入字符串的操作。
- remove(DocumentFilter.FilterBypass fb, int offset, int length)：该方法会拦截用户从文档中删除字符串的操作。
- replace(DocumentFilter.FilterBypass fb, int offset, int length, String text, AttributeSet attrs)：该方法会拦截用户替换文档中字符串的操作。

为了创建自己的输入校验器，可以通过扩展InputVerifier类来实现。实际上，InputVerifier输入校验器可以绑定到任何输入组件，InputVerifier类里包含了一个verify(JComponent component)方法，当用户在该输入组件内输入完成，且该组件失去焦点时，该方法被调用——如果该方法返回false，即表明用户输入无效，该输入组件将自动得到焦点。也就是说，如果某个输入组件绑定了InputVerifier，则用户必须为该组件输入有效内容，否则用户无法离开该组件。

注意：

有一种情况例外，如果输入焦点离开了带InputVerifier输入校验器的组件后，立即单击某个按钮，则该按钮的事件监听器将会在焦点重新回到原组件之前被触发。

下面程序示范了如何为格式化文本框添加输入过滤器、输入校验器，程序还自定义了一个IP地址格式器，该IP地址格式器扩展了DefaultFormatter格式器。

程序清单：codes\12\12.12\JFormattedTextFieldTest2.java

```
public class JFormattedTextFieldTest2
{
    private JFrame mainWin = new JFrame("测试格式化文本框");
    private JButton okButton = new JButton("确定");
    // 定义用于添加格式化文本框的容器
    private JPanel mainPanel = new JPanel();
    public void init()
    {
        // 添加按钮
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(okButton);
        mainPanel.setLayout(new GridLayout(0, 3));
        mainWin.add(mainPanel, BorderLayout.CENTER);
        JFormattedTextField intField0 = new JFormattedTextField(
            new InternationalFormatter(NumberFormat.getIntegerInstance())
            {
                protected DocumentFilter getDocumentFilter()
                {
                    return new NumberFilter();
```

```
                }
            });
        intField0.setValue(100);
        addRow("只接受数字的文本框", intField0);
        JFormattedTextField intField1 = new JFormattedTextField
            (NumberFormat.getIntegerInstance());
        intField1.setValue(100);
        // 添加输入校验器
        intField1.setInputVerifier(new FormattedTextFieldVerifier());
        addRow("带输入校验器的文本框", intField1);
        // 创建自定义格式器对象
        IPAddressFormatter ipFormatter = new IPAddressFormatter();
        ipFormatter.setOverwriteMode(false);
        // 以自定义格式器对象创建格式化文本框
        JFormattedTextField ipField = new JFormattedTextField(ipFormatter);
        ipField.setValue(new byte[]{(byte)192, (byte)168, 4, 1});
        addRow("IP地址格式", ipField);
        mainWin.add(buttonPanel , BorderLayout.SOUTH);
        mainWin.pack();
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.setVisible(true);
    }
    // 定义添加一行格式化文本框的方法
    private void addRow(String labelText, final JFormattedTextField field)
    {
        mainPanel.add(new JLabel(labelText));
        mainPanel.add(field);
        final JLabel valueLabel = new JLabel();
        mainPanel.add(valueLabel);
        // 为“确定”按钮添加事件监听器
        // 当用户单击“确定”按钮时，文本框后显示文本框内的值
        okButton.addActionListener(event -> {
            Object value = field.getValue();
            // 如果该值是数组，则使用Arrays的toString()方法输出数组
            if (value.getClass().isArray())
            {
                StringBuilder builder = new StringBuilder();
                builder.append('{');
                for (int i = 0; i < Array.getLength(value); i++)
                {
                    if (i > 0)
                        builder.append(',');
                    builder.append(Array.get(value, i).toString());
                }
                builder.append('}');
                valueLabel.setText(builder.toString());
            }
            else
            {
                // 输出格式化文本框的值
                valueLabel.setText(value.toString());
            }
        });
    }
    public static void main(String[] args)
    {
        new JFormattedTextFieldTest2().init();
    }
}
// 输入校验器
class FormattedTextFieldVerifier extends InputVerifier
{
    // 当输入组件失去焦点时，该方法被触发
    public boolean verify(JComponent component)
    {
        JFormattedTextField field = (JFormattedTextField)component;
        // 返回用户输入是否有效
        return field.isEditValid();
    }
}
```

```
// 数字过滤器
class NumberFilter extends DocumentFilter
{
    public void insertString(FilterBypass fb , int offset
        , String string , AttributeSet attr)throws BadLocationException
    {
        StringBuilder builder = new StringBuilder(string);
        // 过滤用户输入的所有字符
        filterInt(builder);
        super.insertString(fb, offset, builder.toString(), attr);
    }
    public void replace(FilterBypass fb , int offset , int length
        , String string , AttributeSet attr)throws BadLocationException
    {
        if (string != null)
        {
            StringBuilder builder = new StringBuilder(string);
            // 过滤用户替换的所有字符
            filterInt(builder);
            string = builder.toString();
        }
        super.replace(fb, offset, length, string, attr);
    }
    // 过滤整数字符，把所有非0~9的字符全部删除
    private void filterInt(StringBuilder builder)
    {
        for (int i = builder.length() - 1; i >= 0; i--)
        {
            int cp = builder.codePointAt(i);
            if (cp > '9' || cp < '0')
            {
                builder.deleteCharAt(i);
            }
        }
    }
}
class IPAddressFormatter extends DefaultFormatter
{
    public String valueToString(Object value)
        throws ParseException
    {
        if (!(value instanceof byte[]))
        {
            throw new ParseException("该IP地址的值只能是字节数组", 0);
        }
        byte[] a = (byte[])value;
        if (a.length != 4)
        {
            throw new ParseException("IP地址必须是4个整数", 0);
        }
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < 4; i++)
        {
            int b = a[i];
            if (b < 0) b += 256;
                builder.append(String.valueOf(b));
            if (i < 3) builder.append('.');
        }
        return builder.toString();
    }
    public Object stringToValue(String text) throws ParseException
    {
        // 将格式化文本框内的字符串以点号（.）分成4节
        String[] nums = text.split("\\.");
        if (nums.length != 4)
        {
            throw new ParseException("IP地址必须是4个整数", 0);
        }
        byte[] a = new byte[4];
        for (int i = 0; i < 4; i++)
```

```
        {
            int b = 0;
            try
            {
                b = Integer.parseInt(nums[i]);
            }
            catch (NumberFormatException e)
            {
                throw new ParseException("IP 地址必须是整数", 0);
            }
            if (b < 0 || b >= 256)
            {
                throw new ParseException("IP 地址值只能在 0~255 之间", 0);
            }
            a[i] = (byte) b;
        }
        return a;
    }
}
```

运行上面程序，会看到窗口中出现三个格式化文本框，其中第一个格式化文本框只能输入数字，其他字符无法输入到该文本框内；第二个格式化文本框有输入校验器，只有当用户输入的内容符合该文本框的要求时，用户才可以离开该文本框；第三个格式化文本框的格式器是自定义的格式器，它要求用户输入的内容是一个合法的 IP 地址。

12.12.4 使用 JEditorPane

Swing 提供了一个 JEditorPane 类，该类可以编辑各种文本内容，包括有格式的文本。在默认情况下，JEditorPane 支持如下三种文本内容。

- text/plain：纯文本，当 JEditorPane 无法识别给定内容的类型时，使用这种文本格式。在这种模式下，文本框的内容是带换行符的无格式文本。
- text/html：HTML 文本格式。该文本组件仅支持 HTML 3.2 格式，因此对互联网上复杂的网页支持非常有限。
- text/rtf：RTF（富文本格式）文本格式。实际上，它对 RTF 的支持非常有限。

通过上面介绍不难看出，其实 JEditorPane 类的用途非常有限，使用 JEditorPane 作为纯文本的编辑器，还不如使用 JTextArea；如果使用 JEditorPane 来支持 RTF 文本格式，但它对这种文本格式的支持又相当有限；JEditorPane 唯一可能的用途就是显示自己的 HTML 文档，前提是这份 HTML 文档比较简单，只包含 HTML 3.2 或更早的元素。

JEditorPane 组件支持三种方法来加载文本内容。

- 使用 setText()方法直接设置 JEditorPane 的文本内容。
- 使用 read()方法从输入流中读取 JEditorPane 的文本内容。
- 使用 setPage()方法来设置 JEditorPane 从哪个 URL 处读取文本内容。在这种情况下，将根据该 URL 来确定内容类型。

在默认状态下，使用 JEditorPane 装载的文本内容是可编辑的，即使装载互联网上的网页也是如此，可以使用 JEditorPane 的 setEditable(false)方法阻止用户编辑该 JEditorPane 里的内容。

当使用 JEditorPane 打开 HTML 页面时，该页面的超链接是活动的，用户可以单击超链接。如果程序想监听用户单击超链接的事件，则必须使用 addHyperlinkListener()方法为 JEditorPane 添加一个 HyperlinkListener 监听器。

从目前的功能来看，JEditorPane 确实没有太大的实用价值，所以本书不打算给出此类的用法示例，有兴趣的读者可以参考光盘 codes\12\12.12\路径下的 JEditorPaneTest.java 来学习该类的用法。相比之下，该类的子类 JTextPane 则功能丰富多了，下面详细介绍 JTextPane 类的用法。

12.12.5 使用 JTextPane

使用 EditPlus、Eclipse 等工具时会发现，当在这些工具中输入代码时，如果输入的单词是程序关键

字、类名等，则这些关键字将会自动变色。使用 JTextPane 组件，就可以开发出这种带有语法高亮的编辑器。

JTextPane 使用 StyledDocument 作为它的 model 对象，而 StyleDocument 允许对文档的不同段落分别设置不同的颜色、字体属性。Document 使用 Element 来表示文档中的组成部分，Element 可以表示章（chapter）、段落（paragraph）等，在普通文档中，Element 也可以表示一行。为了设置 StyledDocument 中文字的字体、颜色，Swing 提供了 AttributeSet 接口来表示文档字体、颜色等属性。

Swing 为 StyledDocument 提供了 DefaultStyledDocument 实现类，该实现类就是 JTextPane 的 model 实现类；为 AttributeSet 接口提供了 MutableAttributeSet 子接口，并为该接口提供了 SimpleAttributeSet 实现类，程序通过这些接口和实现类就可以很好地控制 JTextPane 中文字的字体和颜色。

StyledDocument 提供了如下一个方法来设置文档中局部文字的字体、颜色。

- setParagraphAttributes(int offset, int length, AttributeSet s, boolean replace)：设置文档中从 offset 开始，长度为 length 处的文字使用 s 属性（控制字体、颜色等），最后一个参数控制新属性是替换原有属性，还是将新属性累加到原有属性上。

AttributeSet 的常用实现类是 MutableAttributeSet，为了给 MutableAttributeSet 对象设置字体、颜色等属性，Swing 提供了 StyleConstants 工具类，该工具类里大致包含了如下常用的静态方法来设置 MutableAttributeSet 里的字体、颜色等。

- setAlignment(MutableAttributeSet a, int align)：设置文本对齐方式。
- setBackground(MutableAttributeSet a, Color fg)：设置背景色。
- setBold(MutableAttributeSet a, boolean b)：设置是否使用粗体字。
- setFirstLineIndent(MutableAttributeSet a, float i)：设置首行缩进的大小。
- setFontFamily(MutableAttributeSet a, String fam)：设置字体。
- setFontSize(MutableAttributeSet a, int s)：设置字体大小。
- setForeground(MutableAttributeSet a, Color fg)：设置字体前景色。
- setItalic(MutableAttributeSet a, boolean b)：设置是否采用斜体字。
- setLeftIndent(MutableAttributeSet a, float i)：设置左边缩进大小。
- setLineSpacing(MutableAttributeSet a, float i)：设置行间距。
- setRightIndent(MutableAttributeSet a, float i)：设置右边缩进大小。
- setStrikeThrough(MutableAttributeSet a, boolean b)：设置是否为文字添加删除线。
- setSubscript(MutableAttributeSet a, boolean b)：设置将指定文字设置成下标。
- setSuperscript(MutableAttributeSet a, boolean b)：设置将指定文字设置成上标。
- setUnderline(MutableAttributeSet a, boolean b)：设置是否为文字添加下画线。

提示：

上面这些方法用于控制文档中文字的外观样式，如果读者对这些外观样式不是太熟悉，则可以参考 Word 里设置“字体”属性的设置效果。

图 12.60 显示了 Document 及其相关实现类，以及相关辅助类的类关系图。

下面程序简单地定义了三个 SimpleAttributeSet 对象，并为这三个对象设置了对应的文字、颜色、字体等属性，并使用三个 SimpleAttributeSet 对象设置文档中三段文字的外观。

程序清单：codes\12\12.12\JTextPaneTest.java

```
public class JTextPaneTest
{
    JFrame mainWin = new JFrame("测试 JTextPane");
    JTextPane txt = new JTextPane();
    StyledDocument doc = txt.getStyledDocument();
    // 定义 3SimpleAttributeSet 对象
    SimpleAttributeSet android = new SimpleAttributeSet();
```

```
    SimpleAttributeSet java = new SimpleAttributeSet();
    SimpleAttributeSet javaee = new SimpleAttributeSet();
    public void init()
    {
        // 为 android 属性集设置颜色、字体大小、字体和下画线
        StyleConstants.setForeground(android, Color.RED);
        StyleConstants.setFontSize(android, 24);
        StyleConstants.setFontFamily(android, "Dialog");
        StyleConstants.setUnderline(android, true);
        // 为 java 属性集设置颜色、字体大小、字体和粗体字
        StyleConstants.setForeground(java, Color.BLUE);
        StyleConstants.setFontSize(java, 30);
        StyleConstants.setFontFamily(java, "Arial Black");
        StyleConstants.setBold(java, true);
        // 为 javaee 属性集设置颜色、字体大小、斜体字
        StyleConstants.setForeground(javaee, Color.GREEN);
        StyleConstants.setFontSize(javaee, 32);
        StyleConstants.setItalic(javaee, true);
        // 设置不允许编辑
        txt.setEditable(false);
        txt.setText("疯狂 Android 讲义\n"
            + "疯狂 Java 讲义\n" + "轻量级 Java EE 企业应用实战\n");
        // 分别为文档中 3 段文字设置不同的外观样式
        doc.setCharacterAttributes(0 , 12 , android, true);
        doc.setCharacterAttributes(12 , 12 , java, true);
        doc.setCharacterAttributes(24 , 30 , javaee, true);
        mainWin.add(new JScrollPane(txt), BorderLayout.CENTER);
        // 获取屏幕尺寸
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        int inset = 100;
        // 设置主窗口的大小
        mainWin.setBounds(inset, inset, screenSize.width - inset * 2
            , screenSize.height - inset * 2);
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.setVisible(true);
    }
    public static void main(String[] args)
    {
        new JTextPaneTest().init();
    }
}
```

上面程序其实很简单，程序中的第一段粗体字代码为三个 SimpleAttributeSet 对象设置了字体、字体大小、颜色等外观样式，第二段粗体字代码使用前面的三个 SimpleAttributeSet 对象来控制文档中三段文字的外观样式。运行上面程序，将看到如图 12.61 所示的界面。

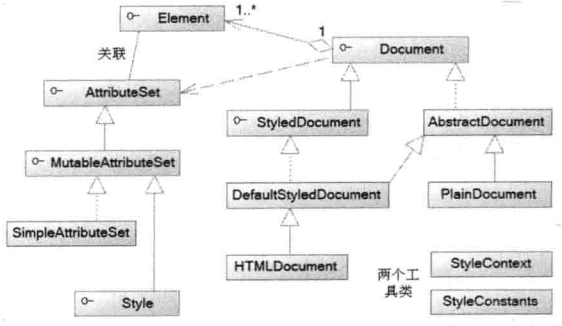


图 12.60 Document 及其相关实现类，以及相关辅助类的类关系图

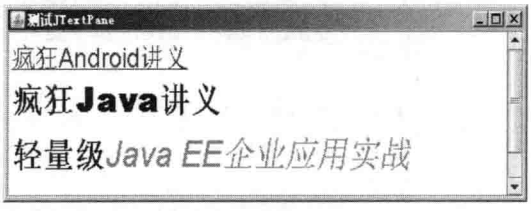


图 12.61 使用 JTextPane 的效果

从图 12.61 中可以看出，窗口中文字具有丰富的外观，而且还可以选中这些文字，表明它们依然是文字，而不是直接绘制上去的图形。

如果希望开发出类似于 EditPlus、Eclipse 等的代码编辑窗口，程序可以扩展 JTextPane 的子类，为该对象添加按键监听器和文档监听器。当文档内容被修改时，或者用户在该文档内进行击键动作时，程序负责分析该文档的内容，对特殊关键字设置字体颜色。

为了保证具有较好的性能，程序并不总是分析文档中的所有内容，而是只分析文档中被改变的部分，这个要求看似简单，只为文档添加文档监听器即可——当文档内容改变时分析被改变部分，并设置其中关键字的颜色。问题是：DocumentListener 监听器里的三个方法不能改变文档本身，所以程序还是必须通过监听按键事件来启动语法分析，DocumentListener 监听器中仅仅记录文档改变部分的位置和长度。

除此之外，程序还提供了一个 SyntaxFormatter 类根据语法文件来设置文档中的文字颜色。

程序清单：codes\12\12.12\MyTextPane.java

```
public class MyTextPane extends JTextPane
{
    protected StyledDocument doc;
    protected SyntaxFormatter formatter = new SyntaxFormatter("my.stx");
    // 定义该文档的普通文本的外观属性
    private SimpleAttributeSet normalAttr =
        formatter.getNormalAttributeSet();
    private SimpleAttributeSet quotAttr = new SimpleAttributeSet();
    // 保存文档改变的开始位置
    private int docChangeStart = 0;
    // 保存文档改变的长度
    private int docChangeLength = 0;
    public MyTextPane()
    {
        StyleConstants.setForeground(quotAttr
            , new Color(255, 0 , 255));
        StyleConstants.setFontSize(quotAttr, 16);
        this.doc = super.getStyledDocument();
        // 设置该文档的页边距
        this.setMargin(new Insets(3, 40, 0, 0));
        // 添加按键监听器，当按键松开时进行语法分析
        this.addKeyListener(new KeyAdapter()
        {
            public void keyReleased(KeyEvent ke)
            {
                syntaxParse();
            }
        });
        // 添加文档监听器
        doc.addDocumentListener(new DocumentListener()
        {
            // 当 Document 的属性或属性集发生了变化时触发该方法
            public void changedUpdate(DocumentEvent e){}
            // 当向 Document 中插入文本时触发该方法
            public void insertUpdate(DocumentEvent e)
            {
                docChangeStart = e.getOffset();
                docChangeLength = e.getLength();
            }
            // 当从 Document 中删除文本时触发该方法
            public void removeUpdate(DocumentEvent e){}
        });
    }
    public void syntaxParse()
    {
        try
        {
            // 获取文档的根元素，即文档内的全部内容
            Element root = doc.getDefaultRootElement();
            // 获取文档中光标插入符的位置
            int cursorPos = this.getCaretPosition();
            int line = root.getElementIndex(cursorPos);
            // 获取光标所在位置的行
            Element para = root.getElement(line);
            // 定义光标所在行的行头在文档中的位置
            int start = para.getStartOffset();
            // 让 start 等于 start 与 docChangeStart 中的较小值
            start = start > docChangeStart ? docChangeStart :start;
```

```
            // 定义被修改部分的长度
            int length = para.getEndOffset() - start;
            length = length < docChangeLength ? docChangeLength + 1
                : length;
            // 取出所有可能被修改的字符串
            String s = doc.getText(start, length);
            // 以空格、点号等作为分隔符
            String[] tokens = s.split("\\s+|\\.|\\(|\\)|\\{|\\}|\\[|\\]");
            // 定义当前分析单词在 s 字符串中的开始位置
            int curStart = 0;
            // 定义单词是否处于引号内
            boolean isQuot = false;
            for (String token : tokens)
            {
                // 找出当前分析单词在 s 字符串中的位置
                int tokenPos = s.indexOf(token , curStart);
                if (isQuot && (token.endsWith("\"") || token.endsWith("\'")))
                {
                    doc.setCharacterAttributes(start + tokenPos
                        , token.length(), quotAttr, false);
                    isQuot = false;
                }
                else if (isQuot && !(token.endsWith("\"")
                    || token.endsWith("\'")))
                {
                    doc.setCharacterAttributes(start + tokenPos
                        , token.length(), quotAttr, false);
                }
                else if ((token.startsWith("\"") || token.startsWith("\'"))
                    && (token.endsWith("\"") || token.endsWith("\'")))
                {
                    doc.setCharacterAttributes(start + tokenPos
                        , token.length(), quotAttr, false);
                }
                else if ((token.startsWith("\"") || token.startsWith("\'"))
                    && !(token.endsWith("\"") || token.endsWith("\'")))
                {
                    doc.setCharacterAttributes(start + tokenPos
                        , token.length(), quotAttr, false);
                    isQuot = true;
                }
                else
                {
                    // 使用格式器对当前单词设置颜色
                    formatter.setHighLight(doc , token , start + tokenPos
                        , token.length());
                }
                // 开始分析下一个单词
                curStart = tokenPos + token.length();
            }
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
    // 重画该组件，设置行号
    public void paint(Graphics g)
    {
        super.paint(g);
        Element root = doc.getDefaultRootElement();
        // 获得行号
        int line = root.getElementIndex(doc.getLength());
        // 设置颜色
        g.setColor(new Color(230, 230, 230));
        // 绘制显示行数的矩形框
        g.fillRect(0 , 0 , this.getMargin().left - 10 , getSize().height);
        // 设置行号的颜色
        g.setColor(new Color(40, 40, 40));
        // 每行绘制一个行号
```

```
            for (int count = 0, j = 1; count <= line; count++, j++)
            {
                g.drawString(String.valueOf(j), 3, (int)((count + 1)
                    * 1.535 * StyleConstants.getFontSize(normalAttr)));
            }
        }
        public static void main(String[] args)
        {
            JFrame frame = new JFrame("文本编辑器");
            // 使用 MyTextPane
            frame.getContentPane().add(new JScrollPane(new MyTextPane()));
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            final int inset = 50;
            Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
            frame.setBounds(inset, inset, screenSize.width - inset*2
                , screenSize.height - inset * 2);
            frame.setVisible(true);
        }
    }
    // 定义语法格式器
    class SyntaxFormatter
    {
        // 以一个 Map 保存关键字和颜色的对应关系
        private Map<SimpleAttributeSet , ArrayList<String>> attMap
            = new HashMap<>();
        // 定义文档的正常文本的外观属性
        SimpleAttributeSet normalAttr = new SimpleAttributeSet();
        public SyntaxFormatter(String syntaxFile)
        {
            // 设置正常文本的颜色、大小
            StyleConstants.setForeground(normalAttr, Color.BLACK);
            StyleConstants.setFontSize(normalAttr, 16);
            // 创建一个 Scanner 对象，负责根据语法文件加载颜色信息
            Scanner scaner = null;
            try
            {
                scaner = new Scanner(new File(syntaxFile));
            }
            catch (FileNotFoundException e)
            {
                throw new RuntimeException("丢失语法文件："
                    + e.getMessage());
            }
            int color = -1;
            ArrayList<String> keywords = new ArrayList<>();
            // 不断读取语法文件的内容行
            while(scaner.hasNextLine())
            {
                String line = scaner.nextLine();
                // 如果当前行以#开头
                if (line.startsWith("#"))
                {
                    if (keywords.size() > 0 && color > -1)
                    {
                        // 取出当前行的颜色值，并封装成 SimpleAttributeSet 对象
                        SimpleAttributeSet att = new SimpleAttributeSet();
                        StyleConstants.setForeground(att, new Color(color));
                        StyleConstants.setFontSize(att, 16);
                        // 将当前颜色和关键字 List 对应起来
                        attMap.put(att , keywords);
                    }
                    // 重新创建新的关键字 List，为下一个语法格式做准备
                    keywords = new ArrayList<>();
                    color = Integer.parseInt(line.substring(1) , 16);
                }
                else
                {
                    // 对于普通行，将每行内容添加到关键字 List 里
```

```
                if (line.trim().length() > 0)
                {
                    keywords.add(line.trim());
                }
            }
        }
        // 把所有的关键字和颜色对应起来
        if (keywords.size() > 0 && color > -1)
        {
            SimpleAttributeSet att = new SimpleAttributeSet();
            StyleConstants.setForeground(att, new Color(color));
            StyleConstants.setFontSize(att, 16);
            attMap.put(att , keywords);
        }
    }
    // 返回该格式器里正常文本的外观属性
    public SimpleAttributeSet getNormalAttributeSet()
    {
        return normalAttr;
    }
    // 设置语法高亮
    public void setHighLight(StyledDocument doc , String token
        , int start , int length)
    {
        // 保存当前单词对应的外观属性
        SimpleAttributeSet currentAttributeSet = null;
        outer :
        for (SimpleAttributeSet att : attMap.keySet())
        {
            // 取出当前颜色对应的所有关键字
            ArrayList<String> keywords = attMap.get(att);
            // 遍历所有关键字
            for (String keyword : keywords)
            {
                // 如果该关键字与当前单词相同
                if (keyword.equals(token))
                {
                    // 跳出循环，并设置当前单词对应的外观属性
                    currentAttributeSet = att;
                    break outer;
                }
            }
        }
        // 如果当前单词对应的外观属性不为空
        if (currentAttributeSet != null)
        {
            // 设置当前单词的颜色
            doc.setCharacterAttributes(start , length
                , currentAttributeSet , false);
        }
        // 否则使用普通外观来设置该单词
        else
        {
            doc.setCharacterAttributes(start , length , normalAttr , false);
        }
    }
}
```

上面程序中的粗体字代码负责分析当前单词与哪种颜色关键字匹配，并为这段文字设置字体颜色。其实这段程序为文档中的单词设置颜色并不难，难点在于找出每个单词与哪种关键字匹配，并要标识出该单词在文档中的位置，然后才可以为该单词设置颜色。

运行上面程序，会看到如图 12.62 所示的带语法高亮的文本编辑器。

上面程序已经完成了对不同类型的单词进行着色，所以会看到如图 12.62 示的运行界面。如果进行改进，则可以为上面的编辑器增加括号配对、代码折叠等功能，这些都可以通过 JTextPane 组件来完成。对于此文本编辑器，只要传入不同的语法文件，程序就可以为不同的源代码显示语法高亮。

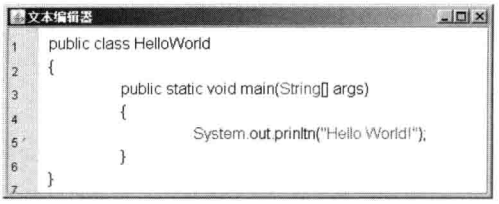


图 12.62　带语法高亮的文本编辑器

12.13　本章小结

本章与前一章内容的结合性非常强，本章主要介绍了以 AWT 为基础的 Swing 编程知识。本章简要介绍了 Swing 基本组件如对话框、按钮的用法，还详细介绍了 Swing 所提供的特殊容器。除此之外，本章重点介绍了 Swing 提供的特殊控件：JList、JComboBox、JSpinner、JSlider、JTable、JTree 等，介绍 JTable、JTree 时深入介绍了 Swing 的 MVC 实现机制，并通过提供自定义的 Render 来改变页面 JTable、JTree 的外观效果。

➤➤ 本章练习

1．设计俄罗斯方块游戏。

2．设计仿 ACDSee 的看图程序。

3．结合 JTree、JList、JSplitPane、JDesktopPane、JInternalFrame、JTextPane 等组件，开发仿 EditPlus 的文字编辑程序界面，可以暂时不提供文字保存、文字打开等功能。

CHAPTER

13

第13章 MySQL数据库与JDBC编程

本章要点

- 关系数据库和 SQL 语句
- DML 语句的语法
- DDL 语句的语法
- 简单查询语句的语法
- 多表连接查询
- 子查询
- JDBC 数据库编程步骤
- 执行 SQL 语句的三种方法
- 使用 PreparedStatement 执行 SQL 语句
- 使用 CallableStatement 调用存储过程
- 使用 ResultSetMetaData 分析结果集元数据
- 理解并掌握 RowSet、RowSetFactory
- 离线 RowSet
- 使用 RowSet 控制分页
- 使用 DatabaseMetaData 分析数据库元数据
- 事务的基础知识
- SQL 语句中的事务控制
- JDBC 编程中的事务控制

通过使用 JDBC，Java 程序可以非常方便地操作各种主流数据库，这是 Java 语言的巨大魅力所在。由于 Java 语言的跨平台特性，所以使用 JDBC 编写的程序不仅可以实现跨数据库，还可以跨平台，具有非常优秀的可移植性。

程序使用 JDBC API 以统一的方式来连接不同的数据库，然后通过 Statement 对象来执行标准的 SQL 语句，并可以获得 SQL 语句访问数据库的结果，因此掌握标准的 SQL 语句是学习 JDBC 编程的基础。本章将会简要介绍关系数据库理论基础，并以 MySQL 数据库为例来讲解标准的 SQL 语句的语法细节，包括基本查询语句、多表连接查询和子查询等。

本章将重点介绍 JDBC 连接数据库的详细步骤，并讲解使用 JDBC 执行 SQL 语句的各种方式，包括使用 CallableStatement 调用存储过程等。本章还会介绍 ResultSetMetaData、DatabaseMetaData 两个接口的用法。事务也是数据库编程中的重要概念，本章不仅会介绍标准 SQL 语句中的事务控制语句，而且会讲解如何利用 JDBC API 进行事务控制。

13.1 JDBC 基础

JDBC 的全称是 Java Database Connectivity，即 Java 数据库连接，它是一种可以执行 SQL 语句的 Java API。程序可通过 JDBC API 连接到关系数据库，并使用结构化查询语言（SQL，数据库标准的查询语言）来完成对数据库的查询、更新。

与其他数据库编程环境相比，JDBC 为数据库开发提供了标准的 API，所以使用 JDBC 开发的数据库应用可以跨平台运行，而且可以跨数据库（如果全部使用标准的 SQL）。也就是说，如果使用 JDBC 开发一个数据库应用，则该应用既可以在 Windows 平台上运行，也可以在 UNIX 等其他平台上运行；既可以使用 MySQL 数据库，也可以使用 Oracle 等数据库，而程序无须进行任何修改。

13.1.1 JDBC 简介

通过使用 JDBC，就可以使用同一种 API 访问不同的数据库系统。换言之，有了 JDBC API，就不必为访问 Oracle 数据库学习一组 API，为访问 DB2 数据库又学习一组 API……开发人员面向 JDBC API 编写应用程序，然后根据不同的数据库，使用不同的数据库驱动程序即可。

提示：

最早的时候，Sun 公司希望自己开发一组 Java API，程序员通过这组 Java API 即可操作所有的数据库系统，但后来 Sun 发现这个目标具有不可实现性——因为数据库系统太多了，而且各数据库系统的内部特性又各不相同。后来 Sun 就制定了一组标准的 API，它们只是接口，没有提供实现类——这些实现类由各数据库厂商提供实现，这些实现类就是驱动程序。而程序员使用 JDBC 时只要面向标准的 JDBC API 编程即可，当需要在数据库之间切换时，只要更换不同的实现类（即更换数据库驱动程序）就行，这是面向接口编程的典型应用。

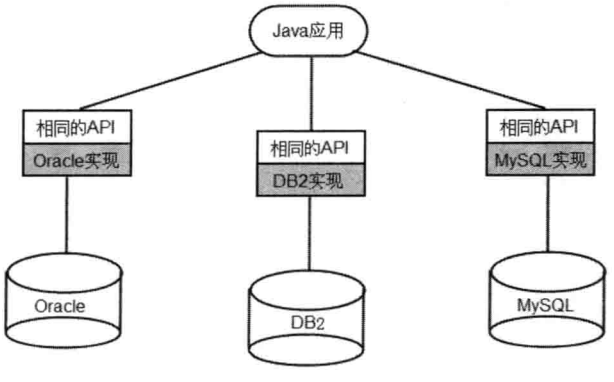


图 13.1 JDBC 驱动示意图

Java 语言的各种跨平台特性，都采用相似的结构，因为它们都需要让相同的程序在不同的平台上运行，所以都需要中间的转换程序（为了实现 Java 程序的跨平台性，Java 为不同的操作系统提供了不同的虚拟机）。同样，为了使 JDBC 程序可以跨平台，则需要不同的数据库厂商提供相应的驱动程序。图 13.1 显示了 JDBC 驱动示意图。

正是通过 JDBC 驱动的转换，才使得使用相同 JDBC API 编写的程序，在不同的数据库系统上运行良好。Sun 提供的 JDBC 可以完成以下三个基本工作。

- 建立与数据库的连接。
- 执行 SQL 语句。
- 获得 SQL 语句的执行结果。

通过 JDBC 的这三个功能，应用程序即可访问、操作数据库系统。

13.1.2 JDBC 驱动程序

数据库驱动程序是 JDBC 程序和数据库之间的转换层，数据库驱动程序负责将 JDBC 调用映射成特定的数据库调用。图 13.2 显示了 JDBC 示意图。

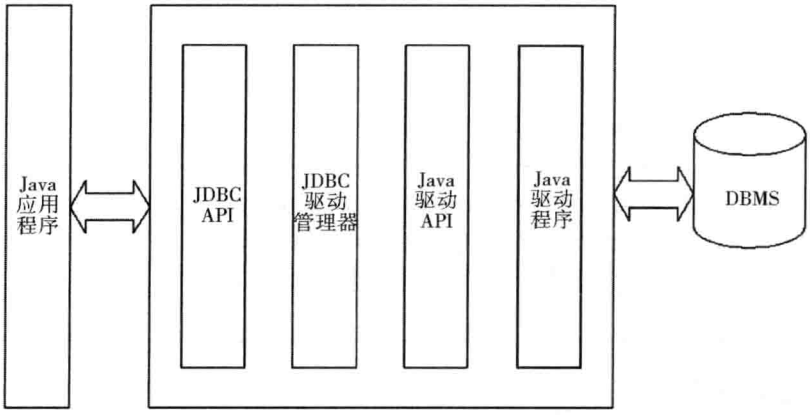


图 13.2 JDBC 访问示意图

大部分数据库系统，例如 Oracle 和 Sybase 等，都有相应的 JDBC 驱动程序，当需要连接某个特定的数据库时，必须有相应的数据库驱动程序。

提示：

还有一种名为 ODBC 的技术，其全称是 Open Database Connectivity，即开放数据库连接。ODBC 和 JDBC 很像，严格来说，应该是 JDBC 模仿了 ODBC 的设计。ODBC 也允许应用程序通过一组通用的 API 访问不同的数据库管理系统，从而使得基于 ODBC 的应用程序可以在不同的数据库之间切换。同样，ODBC 也需要各数据库厂商提供相应的驱动程序，而 ODBC 则负责管理这些驱动程序。

JDBC 驱动通常有如下 4 种类型。

- 第 1 种 JDBC 驱动：称为 JDBC-ODBC 桥，这种驱动是最早实现的 JDBC 驱动程序，主要目的是为了快速推广 JDBC。这种驱动将 JDBC API 映射到 ODBC API。这种方式在最新的 Java 8 中已经被删除了。
- 第 2 种 JDBC 驱动：直接将 JDBC API 映射成数据库特定的客户端 API。这种驱动包含特定数据库的本地代码，用于访问特定数据库的客户端。
- 第 3 种 JDBC 驱动：支持三层结构的 JDBC 访问方式，主要用于 Applet 阶段，通过 Applet 访问数据库。
- 第 4 种 JDBC 驱动：是纯 Java 的，直接与数据库实例交互。这种驱动是智能的，它知道数据库使用的底层协议。这种驱动是目前最流行的 JDBC 驱动。

注意：

早期为了让 Java 程序操作 Access 这种伪数据库，可能需要使用 JDBC-ODBC 桥，但 JDBC-ODBC 桥不适合在并发访问数据库的情况下使用，其固有的性能和扩展能力也非常有限，因此最新的 Java 8 删除了 JDBC-ODBC 桥驱动。基本上 Java 应用也很少使用 Access 这种伪数据库。

通常建议选择第 4 种 JDBC 驱动，这种驱动避开了本地代码，减少了应用开发的复杂性，也减少了

产生冲突和出错的可能。如果对性能有严格的要求，则可以考虑使用第 2 种 JDBC 驱动，但使用这种驱动，则势必增加编码和维护的困难。

相对于 ODBC 而言，JDBC 更加简单。总结起来，JDBC 比 ODBC 多了如下几个优势。

- ODBC 更复杂，ODBC 中有几个命令需要配置很多复杂的选项，而 JDBC 则采用简单、直观的方式来管理数据库连接。
- JDBC 比 ODBC 安全性更高，更易部署。

13.2 SQL 语法

SQL 语句是对所有关系数据库都通用的命令语句，而 JDBC API 只是执行 SQL 语句的工具，JDBC 允许对不同的平台、不同的数据库采用相同的编程接口来执行 SQL 语句。在开始 JDBC 编程之前必须掌握基本的 SQL 知识，本节将以 MySQL 数据库为例详细介绍 SQL 语法知识。

提示：

除了标准的 SQL 语句之外，所有的数据库都会在标准 SQL 语句基础上进行扩展，增加一些额外的功能，这些额外的功能属于特定的数据库系统，不能在所有的数据库系统上都通用。因此，如果想让数据库应用程序可以跨数据库运行，则应该尽量少用这些属于特定数据库的扩展。

13.2.1 安装数据库

对于基于 JDBC 的应用程序，如果使用标准的 SQL 语句进行数据库操作，则应用程序可以在所有的数据库之间切换，只要为程序提供不同的数据库驱动程序即可。从这个角度来看，我们可以使用任何一种数据库来学习 JDBC 编程。本章将以 MySQL 为例来介绍 JDBC 编程，因为 MySQL 数据库非常小巧，而且使用相当简单。

提示：

对初学者不推荐使用 Microsoft 的 SQL Server 作为 JDBC 应用的数据库，因为 Microsoft 为 SQL Server 提供的 JDBC 驱动偶尔会出现未知异常，这些异常会影响初学者学习的心情。

安装 MySQL 数据库与安装普通程序并没有太大的区别，关键是配置 MySQL 数据库时需要注意选择支持中文的编码集。下面简要介绍在 Windows 平台上下载和安装 MySQL 数据库系统的步骤。

① 登录 http://dev.mysql.com/downloads/mysql/站点，下载 MySQL 数据库的最新版本。本书成书之时，MySQL 数据库的最新稳定版本是 MySQL 5.6.17，建议下载该版本的 MySQL 安装文件。读者可根据自己所用的 Windows 平台选择下载相应的 MSI Installer 安装文件。

提示：

如果读者使用的是 64 位系统的电脑，则应该下载 Windows (x86, 64-bit), MSI Installer 项；如果读者使用 32 位系统的电脑，则下载 Windows (x86, 32-bit), MSI Installer 项。

② 下载完成后，得到一个 mysql-installer-community-5.6.17.0.msi 文件，双击该文件，开始安装 MySQL 数据库系统，安装 MySQL 数据库系统与安装普通的 Windows 软件没有太大差别。

③ 开始安装 MySQL 后，在出现的对话框中单击“Install MySQL Products”按钮，然后看到“License Information”界面，该界面要求用户必须接受该协议才能安装 MySQL 数据库系统。勾选该界面下方的“I accept the license terms”复选框，然后单击“Next”按钮。

④ 显示“Find lastest products”界面，通常安装的总是最新下载的 MySQL，因此没必要重新联网来获取最新的 MySQL 版本。勾选该界面下方的“Skip the check for updates（not recommended）”复选框，然后单击“Next”按钮，显示如图 13.3 所示的安装选项对话框。

⑤ 勾选“Custom”单选钮，并设置 MySQL 数据库及数据文件的安装路径，本书选择将 MySQL

数据库和数据文件都安装在 D 盘下。单击“Next”按钮，将显示选择安装组件对话框，选择安装 MySQL 服务器和文档（Applications 和 MySQL Connectors 通常用不到，无须安装），如图 13.4 所示。

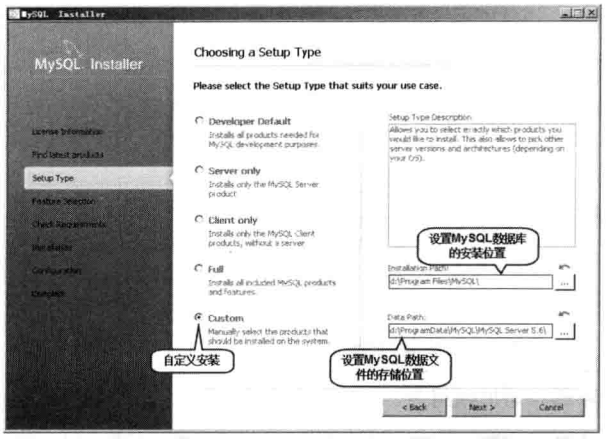


图 13.3　选择自定义安装并设置 MySQL 数据库和数据文件的安装路径

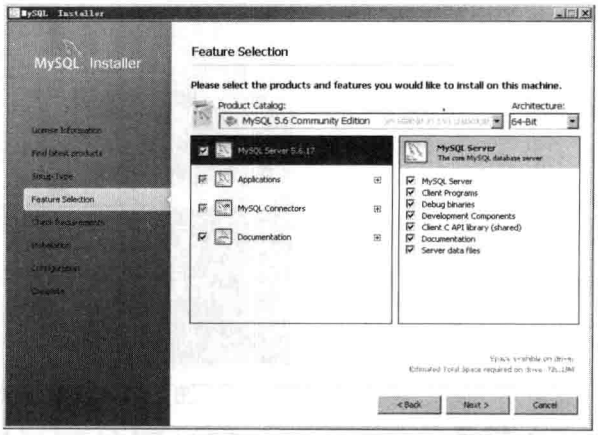


图 13.4　只安装 MySQL 服务器和文档

⑥ 单击“Next”按钮，MySQL Installer 会检查系统环境是否满足安装 MySQL 的要求。如果满足要求，则可以直接单击“Next”按钮开始安装；如果不符合条件，请根据 MySQL 提示先安装相应的系统组件，然后再重新安装 MySQL。开始安装 MySQL 数据库系统。

⑦ 成功安装 MySQL 数据库系统后，会看到如图 13.5 所示的成功安装对话框。

⑧ MySQL 数据库程序安装成功后，系统还要求配置 MySQL 数据库。单击如图 13.5 所示对话框中下方的“Next”按钮，开始配置 MySQL 数据库。在如图 13.6 所示的对话框中，勾选“Show Advanced Options”复选框，这样即可进行更详细的配置。

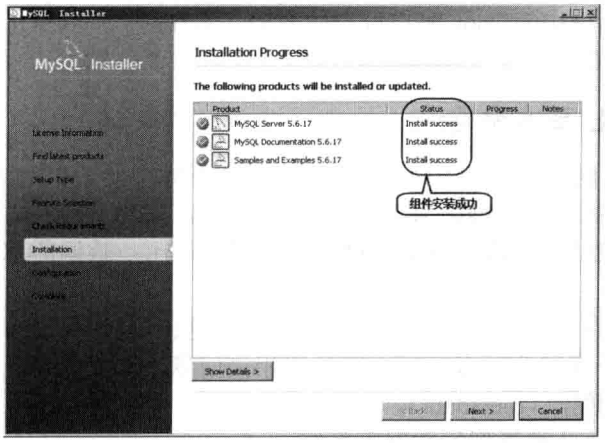


图 13.5　成功安装 MySQL 数据库

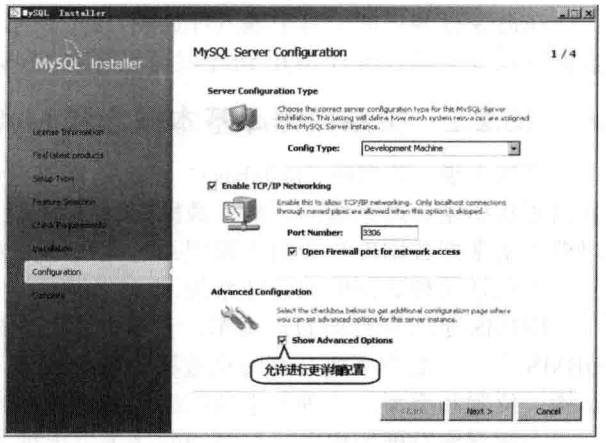


图 13.6　选择进行详细配置

⑨ 单击“Next”按钮，将出现如图 13.7 所示的对话框，允许用户设置 MySQL 的 root 账户密码，也允许添加更多的用户。

⑩ 如果需要为 MySQL 数据库添加更多的用户，则可单击“Add User”按钮进行添加。设置完成后单击“Next”按钮，在配置中将依次出现一系列对话框，但这些对话框对配置影响不大，直接单击“Next”按钮直至 MySQL 配置成功。

MySQL 可通过命令行客户端来管理 MySQL 数据库及数据库里的数据。经过上面 10 个步骤之后，应该在 Windows 的“开始”菜单中看到“MySQL”→“MySQL Server 5.6”→“MySQL 5.5 Command Line Client - Unicode”菜单项，单击该菜单项将启动 MySQL 的命令行客户端窗口，进入该窗口将会提示输入 root 账户密码。

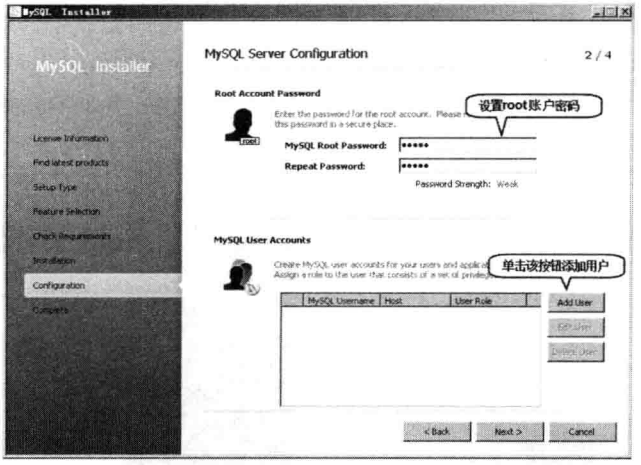


图 13.7　设置 root 账户密码和添加新用户

提示：

由于 MySQL 默认使用 UTF-8 字符串，因此应该通过“MySQL 5.5 Command Line Client - Unicode”菜单项启动命令行工具，该工具将会使用 UTF-8 字符集。

提示：

市面上有一个名为 SQLyog 的工具程序提供了较好的图形用户界面来管理 MySQL 数据库的数据。除此之外，MySQL 也提供了 MySQLAdministrator 工具来管理 MySQL 数据库。读者可以自行下载这两个工具，并使用这两个工具来管理 MySQL 数据库。但本书依然推荐读者使用命令行窗口，因为这种“恶劣”的工具会强制读者记住 SQL 命令的详细用法。

在命令行客户端工具中输入在如图 13.7 所示对话框中为 root 账户设定的密码，系统进入 MySQL 数据库系统，通过执行 SQL 命令就可以管理 MySQL 数据库系统了。

13.2.2　关系数据库基本概念和 MySQL 基本命令

严格来说，数据库（Database）仅仅是存放用户数据的地方。当用户访问、操作数据库中的数据时，就需要数据库管理系统的帮助。数据库管理系统的全称是 Database Management System，简称 DBMS。习惯上常常把数据库和数据库管理系统笼统地称为数据库，通常所说的数据库既包括存储用户数据的部分，也包括管理数据库的管理系统。

DBMS 是所有数据的知识库，它负责管理数据的存储、安全、一致性、并发、恢复和访问等操作。DBMS 有一个数据字典（有时也被称为系统表），用于存储它拥有的每个事务的相关信息，例如名字、结构、位置和类型，这种关于数据的数据也被称为元数据（metadata）。

在数据库发展历史中，按时间顺序主要出现了如下几种类型的数据库系统。

- 网状型数据库
- 层次型数据库
- 关系数据库
- 面向对象数据库

在上面 4 种数据库系统中，关系数据库是理论最成熟、应用最广泛的数据库。从 20 世纪 70 年代末开始，关系数据库理论逐渐成熟，随之涌现出大量商用的关系数据库。关系数据库理论经过 30 多年的发展已经相当完善，在大量数据的查找、排序操作上非常成熟且快速，并对数据库系统的并发、隔离有非常完善的解决方案。

面向对象数据库则是由面向对象编程语言催生的新型数据库，目前有些数据库系统如 Oracle 11g 等开始增加面向对象特性，但面向对象数据库还没有大规模地商业应用。

对于关系数据库而言，最基本的数据存储单元就是数据表，因此可以简单地把数据库想象成大量数

据表的集合（当然，数据库绝不仅由数据表组成）。

数据表是存储数据的逻辑单元，可以把数据表想象成由行和列组成的表格，其中每一行也被称为一条记录，每一列也被称为一个字段。为数据库建表时，通常需要指定该表包含多少列，每列的数据类型信息，无须指定该数据表包含多少行——因为数据库表的行是动态改变的，每行用于保存一条用户数据。除此之外，还应该为每个数据表指定一个特殊列，该特殊列的值可以唯一地标识此行的记录，则该特殊列被称为主键列。

MySQL 数据库的一个实例（Server Instance）可以同时包含多个数据库，MySQL 使用如下命令来查看当前实例下包含多少个数据库：

```
show databases;
```

注意：

MySQL 默认以分号作为每条命令的结束符，所以在每条 MySQL 命令结束后都应该输一个英文分号（;）。

如果用户需要创建新的数据库，则可以使用如下命令：

```
create database [IF NOT EXISTS] 数据库名;
```

如果用户需要删除指定数据库，则可以使用如下命令：

```
drop database 数据库名;
```

建立了数据库之后，如果想操作该数据库（例如为该数据库建表，在该数据库中执行查询等操作），则需要进入该数据库。进入指定数据库可以使用如下命令：

```
use 数据库名;
```

进入指定数据库后，如果需要查询该数据库下包含多少个数据表，则可以使用如下命令：

```
show tables;
```

如果想查看指定数据表的表结构（查看该表有多少列，每列的数据类型等信息），则可以使用如下命令：

```
desc 表名
```

图 13.8 显示了使用 MySQL 命令行客户端执行这些命令的效果。

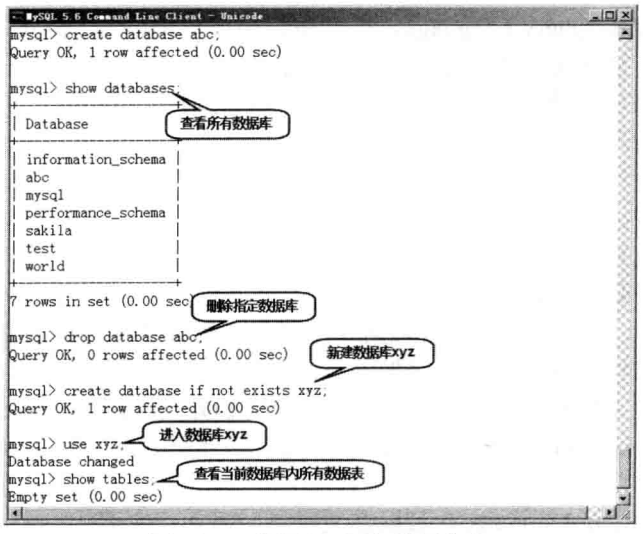


图 13.8　执行 MySQL 常用命令

正如在图 13.8 中看到的，MySQL 的命令行客户端依次执行了 show databases;、drop database abc; 等命令，如果将多条 MySQL 命令写在一份 SQL 脚本文件里，然后将这份 SQL 脚本的内容一次复制到该窗口里，将可以看到该命令行客户端一次性执行所有 SQL 命令的效果——这种一次性执行多条 SQL

命令的方式也被称为导入 SQL 脚本。

提示：

本章的大量程序需要相应数据库的支持，因为本章的大部分程序都会提供对应的 SQL 脚本，因此运行这些程序之前，应该先向 MySQL 数据库中导入这些 SQL 脚本。

MySQL 数据库安装成功后，在其安装目录下有一个 bin 路径（本书中该路径为 D:\Program Files\MySQL\MySQL Server 5.6\bin），该路径下包含一个 mysql 命令，该命令用于启动 MySQL 命令行客户端。执行 mysql 命令的语法如下：

```
mysql -p 密码 -u 用户名 -h 主机名 --default-character-set=utf8
```

执行上面命令可以连接远程主机的 MySQL 服务。为了保证有较好的安全性，执行上面命令时可以省略-p 后面的密码，执行该命令后系统会提示输入密码。

提示：

为了更方便地使用该命令，可以将该 MySQL 安装路径下的 bin 目录添加到系统 PATH 环境变量中。实际上，“开始”菜单中的“MySQL 5.5 Command Line Client - Unicode”菜单项就是一条 mysql 命令。

MySQL 数据库通常支持如下两种存储机制。

- MyISAM：这是 MySQL 早期默认的存储机制，对事务支持不够好。
- InnoDB：InnoDB 提供事务安全的存储机制。InnoDB 通过建立行级锁来保证事务完整性，并以 Oracle 风格的共享锁来处理 Select 语句。系统默认启动 InnoDB 存储机制，如果不想使用 InnoDB 表，则可以使用 skip-innodb 选项。

对比两种存储机制，不难发现 InnoDB 比 MyISAM 多了事务支持的功能，而事务支持是 Java EE 最重要的特性，因此通常推荐使用 InnoDB 存储机制。如果使用 5.0 以上版本的 MySQL 数据库系统，通常无须指定数据表的存储机制，因为系统默认使用 InnoDB 存储机制。如果需要在建表时显式指定存储机制，则可在标准建表语法的后面添加下面任意一句。

- ENGINE=MyISAM——强制使用 MyISAM 存储机制。
- ENGINE=InnoDB——强制使用 InnoDB 存储机制。

13.2.3 SQL 语句基础

SQL 的全称是 Structured Query Language，也就是结构化查询语言。SQL 是操作和检索关系数据库的标准语言，标准的 SQL 语句可用于操作任何关系数据库。

使用 SQL 语句，程序员和数据库管理员（DBA）可以完成如下任务。

- 在数据库中检索信息。
- 对数据库的信息进行更新。
- 改变数据库的结构。
- 更改系统的安全设置。
- 增加或回收用户对数据库、表的许可权限。

在上面 5 个任务中，一般程序员可以管理前 3 个任务，后面 2 个任务通常由 DBA 来完成。

标准的 SQL 语句通常可分为如下几种类型。

- 查询语句：主要由 select 关键字完成，查询语句是 SQL 语句中最复杂、功能最丰富的语句。
- DML（Data Manipulation Language，数据操作语言）语句：主要由 insert、update 和 delete 三个关键字完成。
- DDL（Data Definition Language，数据定义语言）语句：主要由 create、alter、drop 和 truncate 四个关键字完成。
- DCL（Data Control Language，数据控制语言）语句：主要由 grant 和 revoke 两个关键字完成。
- 事务控制语句：主要由 commit、rollback 和 savepoint 三个关键字完成。

SQL 语句的关键字不区分大小写，也就是说，create 和 CREATE 的作用完全一样。在上面 5 种 SQL

语句中，DCL 语句用于为数据库用户授权，或者回收指定用户的权限，通常无须程序员操作，所以本节不打算介绍任何关于 DCL 的知识。

在 SQL 命令中也可能需要使用标识符，标识符可用于定义表名、列名，也可用于定义变量等。这些标识符的命名规则如下。

- 标识符通常必须以字母开头。
- 标识符包括字母、数字和三个特殊字符（# _ $）。
- 不要使用当前数据库系统的关键字、保留字，通常建议使用多个单词连缀而成，单词之间以_分隔。
- 同一个模式下的对象不应该同名，这里的模式指的是外模式。

掌握了 SQL 的这些基础知识后，下面将分类介绍各种 SQL 语句。

注意：

truncate 是一个特殊的 DDL 语句，truncate 在很多数据库中都被归类为 DDL，它相当于先删除指定的数据表，然后再重建该数据表。如果使用 MySQL 的普通存储机制，truncate 确实是这样的。但如果使用 InnoDB 存储机制，则比较复杂，在 MySQL 5.0.3 之前，truncate 和 delete 完全一样；在 5.0.3 之后，truncate table 比 delete 效率高，但如果该表被外键约束所参照，则依然被映射成 delete 操作。当使用快速 truncate 时，该操作会重设自动增长计数器。在 5.0.13 之后，快速 truncate 总是可用，即比 delete 性能要好。关于 truncate 的用法，请参考本章后面内容。

13.2.4　DDL 语句

DDL 语句是操作数据库对象的语句，包括创建（create）、删除（drop）和修改（alter）数据库对象。

前面已经介绍过，最基本的数据库对象是数据表，数据表是存储数据的逻辑单元。但数据库里绝不仅包括数据表，数据库里可包含如表 13.1 所示的几种常见的数据库对象。

表 13.1　常见的数据库对象

对象名称	对应关键字	描　述
表	table	表是存储数据的逻辑单元，以行和列的形式存在；列就是字段，行就是记录
数据字典		就是系统表，存放数据库相关信息的表。系统表里的数据通常由数据库系统维护，程序员通常不应该手动修改系统表及系统表数据，只可查看系统表数据
约束	constraint	执行数据校验的规则，用于保证数据完整性的规则
视图	view	一个或者多个数据表里数据的逻辑显示。视图并不存储数据
索引	index	用于提高查询性能，相当于书的目录
函数	function	用于完成一次特定的计算，具有一个返回值
存储过程	procedure	用于完成一次完整的业务处理，没有返回值，但可通过传出参数将多个值传给调用环境
触发器	trigger	相当于一个事件监听器，当数据库发生特定事件后，触发器被触发，完成相应的处理

因为存在上面几种数据库对象，所以 create 后可以紧跟不同的关键字。例如，建表应使用 create table，建索引应使用 create index，建视图应使用 create view……drop 和 alter 后也需要添加类似的关键字来表示删除、修改哪种数据库对象。

提示：

因为函数、存储过程和触发器属于数据库编程内容，而且需要大量使用数据库特性，这已经超出了本书的介绍范围，故本章不打算介绍函数、存储过程和触发器编程。

1. 创建表的语法

标准的建表语句的语法如下：

```
create table [模式名.]表名
(
```

```
    # 可以有多个列定义
    columnName1 datatype [default expr] ,
    ...
)
```

上面语法中圆括号里可以包含多个列定义，每个列定义之间以英文逗号（,）隔开，最后一个列定义不需要使用英文逗号，而是直接以括号结束。

前面已经讲过，建立数据表只是建立表结构，就是指定该数据表有多少列，每列的数据类型，所以建表语句的重点就是圆括号里的列定义，列定义由列名、列类型和可选的默认值组成。

列定义有点类似于 Java 里的变量定义，与变量定义不同的是，列定义时将列名放在前面，列类型放在后面。如果要指定列的默认值，则使用 default 关键字，而不是使用等号（=）。

例如下面的建表语句：

```
create table test
(
    # 整型通常用 int
    test_id int,
    # 小数点数
    test_price decimal,
    # 普通长度文本，使用 default 指定默认值
    test_name varchar(255) default 'xxx',
    # 大文本类型
    test_desc text,
    # 图片
    test_img blob,
    test_date datetime
);
```

建表时需要指定每列的数据类型，不同数据库所支持的列类型不同，这需要查阅不同数据库的相关文档。MySQL 支持如表 13.2 所示的几种列类型。

表 13.2　MySQL 支持的列类型

列　类　型	说　明
tinyint/smallint/mediumint int(integer)/bigint	1 字节/2 字节/3 字节/4 字节/8 字节整数，又可分为有符号和无符号两种。这些整数类型的区别仅仅是表数范围不同
float/double	单精度、双精度浮点类型
decimal(dec)	精确小数类型，相对于 float 和 double 不会产生精度丢失的问题
date	日期类型，不能保存时间。把 java.util.Date 对象保存进 date 列时，时间部分将会丢失
time	时间类型，不能保存日期。把 java.util.Date 对象保存进 time 列时，日期部分将会丢失
datetime	日期、时间类型
timestamp	时间戳类型
year	年类型，仅仅保存时间的年份
char	定长字符串类型
varchar	可变长度字符串类型
binary	定长二进制字符串类型，它以二进制形式保存字符串
varbinary	可变长度的二进制字符串类型，它以二进制形式保存字符串
tinyblob/blob mediumblob/longblob	1 字节/2 字节/3 字节/4 字节的二进制大对象，可用于存储图片、音乐等二进制数据，分别可存储：255B/64KB/16MB/4GB 的大小
tinytext/text mediumtext/longtext	1 字节/2 字节/3 字节/4 字节的文本对象，可用于存储超长长度的字符串，分别可存储：255B/64KB/16MB/4GB 大小的文本
enum('value1','value2',...)	枚举类型，该列的值只能是 enum 后括号里多个值的其中之一
set('value1','value2',...)	集合类型，该列的值可以是 set 后括号里多个值的其中几个

上面是比较常见的建表语句，这种建表语句只是创建一个空表，该表里没有任何数据。如果使用子

查询建表语句，则可以在建表的同时插入数据。子查询建表语句的语法如下：

```
create table [模式名.]表名 [column[, column...]]
as subquery;
```

上面语法中新表的字段列表必须与子查询中的字段列表数量匹配，创建新表时的字段列表可以省略，如果省略了该字段列表，则新表的列名与选择结果完全相同。下面语句使用子查询来建表。

```
# 创建 hehe 数据表，该数据表和 user_inf 完全相同，数据也完全相同
create table hehe
as
select * from user_inf;
```

因为上面语句是利用子查询来建立数据表，所以执行该 SQL 语句要求数据库中已存在 user_inf 数据表（读者可向 test 数据库中导入 codes\13\13.2 目录下的 user_inf.sql 脚本后执行上面命令），否则程序将出现错误。

提示：

当数据表创建成功后，MySQL 使用 information_schema 数据库里的 TABLES 表来保存该数据库实例中的所有数据表，用户可通过查询 TABLES 表来获取该数据库的表信息。

2. 修改表结构的语法

修改表结构使用 alter table，修改表结构包括增加列定义、修改列定义、删除列、重命名列等操作。增加列定义的语法如下：

```
alter table 表名
add
(
    # 可以有多个列定义
    column_name1 datatype [default expr] ,
    ...
);
```

上面的语法格式中圆括号部分与建表语法的圆括号部分完全相同，只是此时圆括号里的列定义是追加到已有表的列定义后面。还有一点需要指出，如果只是新增一列，则可以省略圆括号，仅在 add 后紧跟一个列定义即可。为数据表增加字段的 SQL 语句如下：

```
# 为 hehe 数据表增加一个 hehe_id 字段，该字段的类型为 int
alter table hehe
add hehe_id int;
# 为 hehe 数据表增加 aaa、bbb 字段，两个字段的类型都为 varchar(255)
alter table hehe
add
(
    aaa varchar(255) default 'xxx',
    bbb varchar(255)
);
```

上面第二条 SQL 语句增加 aaa 字段时，为该字段指定默认值为'xxx'。值得指出的是，SQL 语句中的字符串值不是用双引号引起，而是用单引号引起的。

增加字段时需要注意：如果数据表中已有数据记录，除非给新增的列指定了默认值，否则新增的数据列不可指定非空约束，因为那些已有的记录在新增列上肯定是空（实际上，修改表结构很容易失败，只要新增的约束与已有数据冲突，修改就会失败）。

修改列定义的语法如下：

```
alter table 表名
modify column_name datatype [default expr] [first|after col_name];
```

上面语法中 first 或者 after col_name 指定需要将目标修改到指定位置。

从上面修改语法中可以看出，该修改语句每次只能修改一个列定义，如下代码所示：

```
# 将 hehe 表的 hehe_id 列修改成 varchar(255)类型
alter table hehe
```

```
modify hehe_id varchar(255);
# 将 hehe 表的 bbb 列修改成 int 类型
alter table hehe
modify bbb int;
```

从上面代码中不难看出，使用 SQL 修改数据表里列定义的语法和为数据表只增加一个列定义的语法几乎完全一样，关键是增加列定义使用 add 关键字，而修改列定义使用 modify 关键字。还有一点需要指出，add 新增的列名必须是原表中不存在的，而 modify 修改的列名必须是原表中已存在的。

注意：

虽然 MySQL 的一个 modify 命令不支持一次修改多个列定义，但其他数据库如 Oracle 支持一个 modify 命令修改多个列定义，一个 modify 命令修改多个列定义的语法和一个 add 命令增加多个列定义的语法非常相似，也需要使用圆括号把多个列定义括起来。如果需要让 MySQL 支持一次修改多个列定义，则可在 alter table 后使用多个 modify 命令。

如果数据表里已有数据记录，则修改列定义非常容易失败，因为有可能修改的列定义规则与原有的数据记录不符合。如果修改数据列的默认值，则只会对以后的插入操作有作用，对以前已经存在的数据不会有任何影响。

从数据表中删除列的语法比较简单：

```
alter table 表名
drop column_name
```

删除列只要在 drop 后紧跟需要删除的列名即可。例如：

```
# 删除 hehe 表中的 aaa 字段
alter table hehe
drop aaa;
```

从数据表中删除列定义通常总是可以成功，删除列定义时将从每行中删除该列的数据，并释放该列在数据块中占用的空间。所以删除大表中的字段时需要比较长的时间，因为还需要回收空间。

上面介绍的这些增加列、修改列和删除列的语法是标准的 SQL 语法，对所有的数据库都通用。除此之外，MySQL 还提供了两种特殊的语法：重命名数据表和完全改变列定义。

重命名数据表的语法格式如下：

```
alter table 表名
rename to 新表名
```

如下 SQL 语句用于将 hehe 表命名为 wawa：

```
# 将 hehe 数据表重命名为 wawa
alter table hehe
rename to wawa;
```

MySQL 为 alter table 提供了 change 选项，该选项可以改变列名。change 选项的语法如下：

```
alter table 表名
change old_column_name new_column_name type [default expr] [first|after col_name]
```

对比 change 和 modify 两个选项，不难发现：change 选项比 modify 选项多了一个列名，因为 change 选项可以改变列名，所以它需要两个列名。一般而言，如果不需要改变列名，使用 alter table 的 modify 选项即可，只有当需要修改列名时才会使用 change 选项。如下语句所示：

```
# 将 wawa 数据表的 bbb 字段重命名为 ddd
alter table wawa
change bbb ddd int;
```

3. 删除表的语法

删除表的语法格式如下：

```
drop table 表名;
```

如下 SQL 语句将会把数据库中已有的 wawa 数据表删除：

```
# 删除数据表
drop table wawa;
```

删除数据表的效果如下。

- 表结构被删除，表对象不再存在。
- 表里的所有数据也被删除。
- 该表所有相关的索引、约束也被删除。

4. truncate 表

对于大部分数据库而言，truncate 都被当成 DDL 处理，truncate 被称为“截断”某个表——它的作用是删除该表里的全部数据，但保留表结构。相对于 DML 里的 delete 命令而言，truncate 的速度要快得多，而且 truncate 不像 delete 可以删除指定的记录，truncate 只能一次性删除整个表的全部记录。truncate 命令的语法如下：

```
truncate 表名
```

MySQL 对 truncate 的处理比较特殊——如果使用非 InnoDB 存储机制，truncate 比 delete 速度要快；如果使用 InnoDB 存储机制，在 MySQL 5.0.3 之前，truncate 和 delete 完全一样，在 5.0.3 之后，truncate table 比 delete 效率高，但如果该表被外键约束所参照，truncate 又变为 delete 操作。在 5.0.13 之后，快速 truncate 总是可用，即比 delete 性能要好。

13.2.5 数据库约束

前面创建的数据表仅仅指定了一些列定义，这仅仅是数据表的基本功能。除此之外，所有的关系数据库都支持对数据表使用约束，通过约束可以更好地保证数据表里数据的完整性。约束是在表上强制执行的数据校验规则，约束主要用于保证数据库里数据的完整性。除此之外，当表中数据存在相互依赖性时，可以保护相关的数据不被删除。

大部分数据库支持下面 5 种完整性约束。

- NOT NULL：非空约束，指定某列不能为空。
- UNIQUE：唯一约束，指定某列或者几列组合不能重复。
- PRIMARY KEY：主键，指定该列的值可以唯一地标识该条记录。
- FOREIGN KEY：外键，指定该行记录从属于主表中的一条记录，主要用于保证参照完整性。
- CHECK：检查，指定一个布尔表达式，用于指定对应列的值必须满足该表达式。

虽然大部分数据库都支持上面 5 种约束，但 MySQL 不支持 CHECK 约束，虽然 MySQL 的 SQL 语句也可以使用 CHECK 约束，但这个 CHECK 约束不会有任何作用。

虽然约束的作用只是用于保证数据表里数据的完整性，但约束也是数据库对象，并被存储在系统表中，也拥有自己的名字。根据约束对数据列的限制，约束分为如下两类。

- 单列约束：每个约束只约束一列。
- 多列约束：每个约束可以约束多个数据列。

为数据表指定约束有如下两个时机。

- 建表的同时为相应的数据列指定约束。
- 建表后创建，以修改表的方式来增加约束。

大部分约束都可以采用列级约束语法或者表级约束语法。下面依次介绍 5 种约束的建立和删除（约束通常无法修改）。

提示：

MySQL 使用 information_schema 数据库里的 TABLE_CONSTRAINTS 表来保存该数据库实例中所有的约束信息，用户可以通过查询 TABLE_CONSTRAINTS 表来获取该数据库的约束信息。

1. NOT NULL 约束

非空约束用于确保指定列不允许为空，非空约束是比较特殊的约束，它只能作为列级约束使用，只能使用列级约束语法定义。这里要介绍一下 SQL 中的 null 值，SQL 中的 null 不区分大小写。SQL 中的 null 具有如下特征。

- 所有数据类型的值都可以是 null，包括 int、float、boolean 等数据类型。
- 与 Java 类似的是，空字符串不等于 null，0 也不等于 null。

如果需要在建表时为指定列指定非空约束，只要在列定义后增加 not null 即可。建表语句如下：

```
create table hehe
(
    # 建立了非空约束，这意味着 hehe_id 不可以为 null
    hehe_id int not null,
    # MySQL 的非空约束不能指定名字
    hehe_name varchar(255) default 'xyz' not null,
    # 下面列可以为空，默认就是可以为空
    hehe_gender varchar(2) null
);
```

除此之外，也可以在使用 alter table 修改表时增加或者删除非空约束，SQL 命令如下：

```
# 增加非空约束
alter table hehe
modify hehe_gender varchar(2) not null
# 取消非空约束
alter table hehe
modify hehe_name varchar(2) null;
# 取消非空约束，并指定默认值
alter table hehe
modify hehe_name varchar(255) default 'abc' null;
```

2. UNIQUE 约束

唯一约束用于保证指定列或指定列组合不允许出现重复值。虽然唯一约束的列不可以出现重复值，但可以出现多个 null 值（因为在数据库中 null 不等于 null）。

同一个表内可建多个唯一约束，唯一约束也可由多列组合而成。当为某列创建唯一约束时，MySQL 会为该列相应地创建唯一索引。如果不给唯一约束起名，该唯一约束默认与列名相同。

唯一约束既可以使用列级约束语法建立，也可以使用表级约束语法建立。如果需要为多列建组合约束，或者需要为唯一约束指定约束名，则只能用表级约束语法。

当建立唯一约束时，MySQL 在唯一约束所在列或列组合上建立对应的唯一索引。

使用列级约束语法建立唯一约束非常简单，只要简单地在列定义后增加 unique 关键字即可。SQL 语句如下：

```
# 建表时创建唯一约束，使用列级约束语法建立约束
create table unique_test
(
    # 建立了非空约束，这意味着 test_id 不可以为 null
    test_id int not null,
    # unique 就是唯一约束，使用列级约束语法建立唯一约束
    test_name varchar(255) unique
);
```

如果需要为多列组合建立唯一约束，或者想自行指定约束名，则需要使用表级约束语法。表级约束语法格式如下：

```
[constraint 约束名] 约束定义
```

上面的表级约束语法格式既可放在 create table 语句中与列定义并列，也可放在 alter table 语句中使用 add 关键字来添加约束。SQL 语句如下：

```
# 建表时创建唯一约束，使用表级约束语法建立约束
create table unique_test2
(
```

```
    # 建立了非空约束，这意味着 test_id 不可以为 null
    test_id int not null,
    test_name varchar(255),
    test_pass varchar(255),
    # 使用表级约束语法建立唯一约束
    unique (test_name),
    # 使用表级约束语法建立唯一约束，而且指定约束名
    constraint test2_uk unique(test_pass)
);
```

上面的建表语句为 test_name、test_pass 分别建立了唯一约束，这意味着这两列都不能出现重复值。除此之外，还可以为这两列组合建立唯一约束，SQL 语句如下：

```
# 建表时创建唯一约束，使用表级约束语法建立约束
create table unique_test3
(
    # 建立了非空约束，这意味着 test_id 不可以为 null
    test_id int not null,
    test_name varchar(255),
    test_pass varchar(255),
    # 使用表级约束语法建立唯一约束，指定两列组合不允许重复
    constraint test3_uk unique(test_name,test_pass)
);
```

对于上面的 unique_test2 和 unique_test3 两个表，都是对 test_name、test_pass 建立唯一约束，其中 unique_test2 要求 test_name、test_pass 都不能出现重复值，而 unique_test3 只要求 test_name、test_pass 两列值的组合不能重复。

也可以在修改表结构时使用 add 关键字来增加唯一约束，SQL 语句如下：

```
# 增加唯一约束
alter table unique_test3
add unique(test_name, test_pass);
```

还可以在修改表时使用 modify 关键字，为单列采用列级约束语法来增加唯一约束，代码如下：

```
# 为 unique_test3 表的 test_name 列增加唯一约束
alter table unique_test3
modify test_name varchar(255) unique;
```

对于大部分数据库而言，删除约束都是在 alter table 语句后使用“drop constraint 约束名”语法来完成的，但 MySQL 并不使用这种方式，而是使用“drop index 约束名”的方式来删除约束。例如如下 SQL 语句：

```
# 删除 unique_test3 表上的 test3_uk 唯一约束
alter table unique_test3
drop index test3_uk;
```

3. PRIMARY KEY 约束

主键约束相当于非空约束和唯一约束，即主键约束的列既不允许出现重复值，也不允许出现 null 值；如果对多列组合建立主键约束，则多列里包含的每一列都不能为空，但只要求这些列组合不能重复。主键列的值可用于唯一地标识表中的一条记录。

每一个表中最多允许有一个主键，但这个主键约束可由多个数据列组合而成，主键是表中能唯一确定一行记录的字段或字段组合。

建立主键约束时既可使用列级约束语法，也可使用表级约束语法。如果需要对多个字段建立组合主键约束，则只能使用表级约束语法。使用表级约束语法来建立约束时，可以为该约束指定约束名。但不管用户是否为该主键约束指定约束名，MySQL 总是将所有的主键约束命名为 PRIMARY。

提示：

MySQL 允许在建立主键约束时为该约束命名，但这个名字没有任何作用，这是为了保持与标准 SQL 的兼容性。大部分数据库都允许自行指定主键约束的名字，而且一旦指定了主键约束名，则该约束名就是用户指定的名字。

当创建主键约束时，MySQL 在主键约束所在列或列组合上建立对应的唯一索引。

创建主键约束的语法和创建唯一约束的语法非常像，一样允许使用列级约束语法为单独的数据列创建主键，如果需要为多列组合建立主键约束或者需要为主键约束命名，则应该使用表级约束语法来建立主键约束。与建立唯一约束不同的是，建立主键约束使用 primary key。

建表时创建主键约束，使用列级约束语法：

```
create table primary_test
(
    # 建立了主键约束
    test_id int primary key,
    test_name varchar(255)
);
```

建表时创建主键约束，使用表级约束语法：

```
create table primary_test2
(
    test_id int not null,
    test_name varchar(255),
    test_pass varchar(255),
    # 指定主键约束名为 test2_pk，对大部分数据库有效，但对 MySQL 无效
    # MySQL 数据库中该主键约束名依然是 PRIMARY
    constraint test2_pk primary key(test_id)
);
```

建表时创建主键约束，以多列建立组合主键，只能使用表级约束语法：

```
create table primary_test3
(
    test_name varchar(255),
    test_pass varchar(255),
    # 建立多列组合的主键约束
    primary key(test_name, test_pass)
);
```

如果需要删除指定表的主键约束，则在 alter table 语句后使用 drop primary key 子句即可。SQL 语句如下：

```
# 删除主键约束
alter table primary_test3
drop primary key;
```

如果需要为指定表增加主键约束，既可通过 modify 修改列定义来增加主键约束，这将采用列级约束语法来增加主键约束；也可通过 add 来增加主键约束，这将采用表级约束语法来增加主键约束。SQL 语句如下：

```
# 使用表级约束语法增加主键约束
alter table primary_test3
add primary key(test_name,test_pass);
```

如果只是为单独的数据列增加主键约束，则可使用 modify 修改列定义来实现，如下 SQL 语句所示：

```
# 使用列级约束语法增加主键约束
alter table primary_test3
modify test_name varchar(255) primary key;
```

注意：

不要连续执行上面两条 SQL 语句，因为上面两条 SQL 语句都是为 primary_test3 增加主键约束，而同一个表里最多只能有一个主键约束，所以连续执行上面两条 SQL 语句肯定出现错误。为了避免这个问题，可以在成功执行了第一条增加主键约束的 SQL 语句之后，先将 primary_test3 里的主键约束删除后再执行第二条增加主键约束的 SQL 语句。

很多数据库对主键列都支持一种自增长的特性——如果某个数据列的类型是整型，而且该列作为主键列，则可指定该列具有自增长功能。指定自增长功能通常用于设置逻辑主键列——该列的值没有任何

物理意义，仅仅用于标识每行记录。MySQL 使用 auto_increment 来设置自增长，SQL 语句如下：

```
create table primary_test4
(
    # 建立主键约束，使用自增长
    test_id int auto_increment primary key,
    test_name varchar(255),
    test_pass varchar(255)
);
```

一旦指定了某列具有自增长特性，则向该表插入记录时可不为该列指定值，该列的值由数据库系统自动生成。

4. FOREIGN KEY 约束

外键约束主要用于保证一个或两个数据表之间的参照完整性，外键是构建于一个表的两个字段或者两个表的两个字段之间的参照关系。外键确保了相关的两个字段的参照关系：子（从）表外键列的值必须在主表被参照列的值范围之内，或者为空（也可以通过非空约束来约束外键列不允许为空）。

当主表的记录被从表记录参照时，主表记录不允许被删除，必须先把从表里参照该记录的所有记录全部删除后，才可以删除主表的该记录。还有一种方式，删除主表记录时级联删除从表中所有参照该记录的从表记录。

从表外键参照的只能是主表主键列或者唯一键列，这样才可保证从表记录可以准确定位到被参照的主表记录。同一个表内可以拥有多个外键。

建立外键约束时，MySQL 也会为该列建立索引。

外键约束通常用于定义两个实体之间的一对多、一对一的关联关系。对于一对多的关联关系，通常在多的一端增加外键列，例如老师—学生（假设一个老师对应多个学生，但每个学生只有一个老师，这是典型的一对多的关联关系）。为了建立他们之间的关联关系，可以在学生表中增加一个外键列，该列中保存此条学生记录对应老师的主键。对于一对一的关联关系，则可选择任意一方来增加外键列，增加外键列的表被称为从表，只要为外键列增加唯一约束就可表示一对一的关联关系了。对于多对多的关联关系，则需要额外增加一个连接表来记录它们的关联关系。

建立外键约束同样可以采用列级约束语法和表级约束语法。如果仅对单独的数据列建立外键约束，则使用列级约束语法即可；如果需要对多列组合创建外键约束，或者需要为外键约束指定名字，则必须使用表级约束语法。

采用列级约束语法建立外键约束直接使用 references 关键字，references 指定该列参照哪个主表，以及参照主表的哪一列。如下 SQL 语句所示：

```
# 为了保证从表参照的主表存在，通常应该先建主表
create table teacher_table
(
    # auto_increment：代表数据库的自动编号策略，通常用作数据表的逻辑主键
    teacher_id int auto_increment,
    teacher_name varchar(255),
    primary key(teacher_id)
);
create table student_table
(
    # 为本表建立主键约束
    student_id int auto_increment primary key,
    student_name varchar(255),
    # 指定 java_teacher 参照到 teacher_table 的 teacher_id 列
    java_teacher int references teacher_table(teacher_id)
);
```

值得指出的是，虽然 MySQL 支持使用列级约束语法来建立外键约束，但这种列级约束语法建立的外键约束不会生效，MySQL 提供这种列级约束语法仅仅是为了和标准 SQL 保持良好的兼容性。因此，如果要使 MySQL 中的外键约束生效，则应使用表级约束语法。

```
# 为了保证从表参照的主表存在，通常应该先建主表
create table teacher_table1
(
```

```
    # auto_increment：代表数据库的自动编号策略，通常用作数据表的逻辑主键
    teacher_id int auto_increment,
    teacher_name varchar(255),
    primary key(teacher_id)
);
create table student_table1
(
    # 为本表建立主键约束
    student_id int auto_increment primary key,
    student_name varchar(255),
    # 指定 java_teacher 参照到 teacher_table1 的 teacher_id 列
    java_teacher int,
    foreign key(java_teacher) references teacher_table1(teacher_id)
);
```

如果使用表级约束语法，则需要使用 foreign key 来指定本表的外键列，并使用 references 来指定参照哪个主表，以及参照到主表的哪个数据列。使用表级约束语法可以为外键约束指定约束名，如果创建外键约束时没有指定约束名，则 MySQL 会为该外键约束命名为 table_name_ibfk_*n*，其中 table_name 是从表的表名，而 *n* 是从 1 开始的整数。

如果需要显式指定外键约束的名字，则可使用 constraint 来指定名字。如下 SQL 语句所示：

```
# 为了保证从表参照的主表存在，通常应该先建主表
create table teacher_table2
(
    # auto_increment：代表数据库的自动编号策略，通常用作数据表的逻辑主键
    teacher_id int auto_increment,
    teacher_name varchar(255),
    primary key(teacher_id)
);
create table student_table2
(
    # 为本表建立主键约束
    student_id int auto_increment primary key,
    student_name varchar(255),
    java_teacher int,
    # 使用表级约束语法建立外键约束，指定外键约束的约束名为 student_teacher_fk
    constraint student_teacher_fk foreign key(java_teacher) references
        teacher_table2(teacher_id)
);
```

如果需要建立多列组合的外键约束，则必须使用表级约束语法，如下 SQL 语句所示：

```
# 为了保证从表参照的主表存在，通常应该先建主表
create table teacher_table3
(
    teacher_name varchar(255),
    teacher_pass varchar(255),
    # 以两列建立组合主键
    primary key(teacher_name , teacher_pass)
);
create table student_table3
(
    # 为本表建立主键约束
    student_id int auto_increment primary key,
    student_name varchar(255),
    java_teacher_name varchar(255),
    java_teacher_pass varchar(255),
    # 使用表级约束语法建立外键约束，指定两列的联合外键
    foreign key(java_teacher_name , java_teacher_pass)
        references teacher_table3(teacher_name , teacher_pass)
);
```

删除外键约束的语法很简单，在 alter table 后增加“drop foreign key 约束名”子句即可。如下代码所示：

```
# 删除 student_table3 表上名为 student_table3_ibfk_1 的外键约束
alter table student_table3
drop foreign key student_table3_ibfk_1;
```

增加外键约束通常使用 add foreign key 命令。如下 SQL 语句所示：

```
# 修改 student_table3 数据表，增加外键约束
alter table student_table3
add foreign key(java_teacher_name , java_teacher_pass)
   references teacher_table3(teacher_name , teacher_pass);
```

值得指出的是，外键约束不仅可以参照其他表，而且可以参照自身，这种参照自身的情况通常被称为自关联。例如，使用一个表保存某个公司的所有员工记录，员工之间有部门经理和普通员工之分，部门经理和普通员工之间存在一对多的关联关系，但他们都是保存在同一个数据表里的记录，这就是典型的自关联。下面的 SQL 语句用于建立自关联的外键约束。

```
# 使用表级约束语法建立外约束键，且直接参照自身
create table foreign_test
(
   foreign_id int auto_increment primary key,
   foreign_name varchar(255),
   # 使用该表的 refer_id 参照到本表的 foreign_id 列
   refer_id int,
   foreign key(refer_id) references foreign_test(foreign_id)
);
```

如果想定义当删除主表记录时，从表记录也会随之删除，则需要在建立外键约束后添加 on delete cascade 或添加 on delete set null，第一种是删除主表记录时，把参照该主表记录的从表记录全部级联删除；第二种是指定当删除主表记录时，把参照该主表记录的从表记录的外键设为 null。如下 SQL 语句所示：

```
# 为了保证从表参照的主表存在，通常应该先建主表
create table teacher_table4
(
   # auto_increment：代表数据库的自动编号策略，通常用作数据表的逻辑主键
   teacher_id int auto_increment,
   teacher_name varchar(255),
   primary key(teacher_id)
);
create table student_table4
(
   # 为本表建立主键约束
   student_id int auto_increment primary key,
   student_name varchar(255),
   java_teacher int,
   # 使用表级约束语法建立外键约束，定义级联删除
   foreign key(java_teacher) references teacher_table4(teacher_id)
      on delete cascade # 也可用 on delete set null
);
```

5. CHECK 约束

当前版本的 MySQL 支持建表时指定 CHECK 约束，但这个 CHECK 约束不会有任何作用。建立 CHECK 约束的语法很简单，只要在建表的列定义后增加 check（逻辑表达式）即可。如下 SQL 语句所示：

```
create table check_test
(
   emp_id int auto_increment,
   emp_name varchar(255),
   emp_salary decimal,
   primary key(emp_id),
   # 建立 CHECK 约束
   check(emp_salary>0)
);
```

虽然上面的 SQL 语句建立的 check_test 表中有 CHECK 约束，CHECK 约束要求 emp_salary 大于 0，但这个要求实际上并不会起作用。

提示：

MySQL 作为一个开源、免费的数据库系统，对有些功能的支持确实不太好。如果读者确实希望 MySQL 创建的数据表有 CHECK 约束，甚至有更复杂的完整性约束，则可借助于 MySQL 的触发器机制。本书不会介绍 MySQL 的触发器内容，读者可参考其他相关书籍。

13.2.6 索引

索引是存放在模式（schema）中的一个数据库对象，虽然索引总是从属于数据表，但它也和数据表一样属于数据库对象。创建索引的唯一作用就是加速对表的查询，索引通过使用快速路径访问方法来快速定位数据，从而减少了磁盘的 I/O。

索引作为数据库对象，在数据字典中独立存放，但不能独立存在，必须属于某个表。

提示：

MySQL 使用 information_schema 数据库里的 STATISTICS 表来保存该数据库实例中的所有索引信息，用户可通过查询该表来获取该数据库的索引信息。

创建索引有两种方式。

- 自动：当在表上定义主键约束、唯一约束和外键约束时，系统会为该数据列自动创建对应的索引。
- 手动：用户可以通过 create index...语句来创建索引。

删除索引也有两种方式。

- 自动：数据表被删除时，该表上的索引自动被删除。
- 手动：用户可以通过 drop index...语句来删除指定数据表上的指定索引。

索引的作用类似于书的目录，几乎没有一本书没有目录，因此几乎没有一个表没有索引。一个表中可以有多个索引列，每个索引都可用于加速该列的查询速度。

正如书的目录总是根据书的知识点来建立一样——因为读者经常要根据知识点来查阅一本书。类似的，通常为经常需要查询的数据列建立索引，可以在一列或者多列上创建索引。创建索引的语法格式如下：

```
create index index_name
on table_name (column[, column]...);
```

下面的索引将会提高对 employees 表基于 last_name 字段的查询速度。

```
create index emp_last_name_idx
on employees(last_name);
```

也可同时对多列建立索引，如下 SQL 语句所示：

```
# 下面语句为 employees 的 first_name 和 last_name 两列同时建立索引
create index emp_last_name_idx2
on employees(first_name, last_name);
```

MySQL 中删除索引需要指定表，采用如下语法格式：

```
drop index 索引名 on 表名
```

如下 SQL 语句删除了 employees 表上的 emp_last_name_idx2 索引：

```
drop index emp_last_name_idx2
on employees
```

有些数据库删除索引时无须指定表名，因为它们要求建立索引时每个索引都有唯一的名字，所以无须指定表名，例如 Oracle 就采用这种策略。但 MySQL 只要求同一个表内的索引不能同名，所以删除索引时必须指定表名。

索引的好处是可以加速查询。但索引也有如下两个坏处。

- 与书的目录类似，当数据表中的记录被添加、删除、修改时，数据库系统需要维护索引，因此有一定的系统开销。

- 存储索引信息需要一定的磁盘空间。

13.2.7 视图

视图看上去非常像一个数据表，但它不是数据表，因为它并不能存储数据。视图只是一个或多个数据表中数据的逻辑显示。使用视图有如下几个好处。

- 可以限制对数据的访问。
- 可以使复杂的查询变得简单。
- 提供了数据的独立性。
- 提供了对相同数据的不同显示。

因为视图只是数据表中数据的逻辑显示——也就是一个查询结果，所以创建视图就是建立视图名和查询语句的关联。创建视图的语法如下：

```
create or replace view 视图名
as
subquery
```

从上面的语法可以看出，创建、修改视图都可使用上面语法。上面语法的含义是，如果该视图不存在，则创建视图；如果指定视图名的视图已经存在，则使用新视图替换原有视图。后面的 subquery 就是一个查询语句，这个查询可以非常复杂。

注意：

通过建立视图的语法规则不难看出，所谓视图的本质，其实就是一条被命名的 SQL 查询语句。

一旦建立了视图以后，使用该视图与使用数据表就没有什么区别了，但通常只是查询视图数据，不会修改视图里的数据，因为视图本身没有存储数据。

如下 SQL 语句就创建了一个简单的视图：

```
create or replace view view_test
as
select teacher_name , teacher_pass from teacher_table;
```

通常不推荐直接改变视图的数据，因为视图并不存储数据，它只是相当于一条命名的查询语句而已。为了强制不允许改变视图的数据，MySQL 允许在创建视图时使用 with check option 子句，使用该子句创建的视图不允许修改，如下所示：

```
create or replace view view_test
as
select teacher_name from teacher_table
# 指定不允许修改该视图的数据
with check option;
```

注意：

大部分数据库都采用 with check option 来强制不允许修改视图的数据，但 Oracle 采用 with read only 来强制不允许修改视图的数据。

删除视图使用如下语句：

```
drop view 视图名
```

如下 SQL 语句删除了前面刚刚创建的视图：

```
drop view view_test;
```

13.2.8 DML 语句语法

与 DDL 操作数据库对象不同，DML 主要操作数据表里的数据，使用 DML 可以完成如下三个任务。

- 插入新数据。
- 修改已有数据。
- 删除不需要的数据。

DML 语句由 insert into、update 和 delete from 三个命令组成。

1. insert into 语句

insert into 用于向指定数据表中插入记录。对于标准的 SQL 语句而言，每次只能插入一条记录。insert into 语句的语法格式如下：

```
insert into table_name [(column [, column...])]
values(value [, value...]);
```

执行插入操作时，表名后可以用括号列出所有需要插入值的列名，而 values 后用括号列出对应需要插入的值。

如果省略了表名后面的括号及括号里的列名列表，默认将为所有列都插入值，则需要为每一列都指定一个值。如果既不想在表名后列出列名，又不想为所有列都指定值，则可以为那些无法确定值的列分配 null。下面的 SQL 语句示范了如何向数据表中插入记录。

注意：

只有在数据库中已经成功创建了数据表之后，才可以向数据表中插入记录。下面的 SQL 语句以前面介绍外键约束时所创建的 teacher_table2 和 student_table2 为例来介绍数据插入操作。

在表名后使用括号列出所有需要插入值的列：

```
insert into teacher_table2(teacher_name)
values('xyz');
```

如果不想在表后用括号列出所有列，则需要为所有列指定值；如果某列的值不能确定，则为该列分配一个 null 值。

```
insert into teacher_table2
# 使用 null 代替主键列的值
values(null , 'abc');
```

经过两条插入语句后，可以看到 teacher_table2 表中的数据如图 13.9 所示。

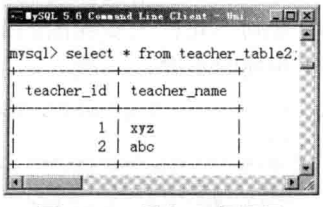


图 13.9　插入 2 条记录

从图 13.9 中看到 abc 记录的主键列的值是 2，而不是 SQL 语句插入的 null，因为该主键列是自增长的，系统会自动为该列分配值。

根据前面介绍的外键约束规则：外键列里的值必须是被参照列里已有的值，所以向从表中插入记录之前，通常应该先向主表中插入记录，否则从表记录的外键列只能为 null。现在主表 teacher_table2 中已有了 2 条记录，现在可以向从表 student_table2 中插入记录了，如下 SQL 语句所示：

```
insert into student_table2
# 当向外键列里插值时，外键列的值必须是被参照列里已有的值
values(null , '张三' , 2);
```

注意：

外键约束保证被参照的记录必须存在，但并不保证必须有被参照记录，即外键列可以为 null。如果想保证每条从表记录必须存在对应的主表记录，则应使用非空、外键两个约束。

在一些特别的情况下，可以使用带子查询的插入语句，带子查询的插入语句可以一次插入多条记录，如下 SQL 语句所示：

```
insert into student_table2(student_name)
# 使用子查询的值来插入
select teacher_name from teacher_table2;
```

正如上面的 SQL 语句所示，带子查询的插入语句甚至不要求查询数据的源表和插入数据的目的表是同一个表，它只要求选择出来的数据列和插入目的表的数据列个数相等、数据类型匹配即可。

MySQL 甚至提供了一种扩展的语法，通过这种扩展的语法也可以一次插入多条记录。MySQL 允许在 values 后使用多个括号包含多条记录，表示多条记录的多个括号之间以英文逗号（,）隔开。如下 SQL 语句所示：

```
insert into teacher_table2
# 同时插入多个值
values(null , "Yeeku"),
(null , "Sharfly");
```

2. update 语句

update 语句用于修改数据表的记录，每次可以修改多条记录，通过使用 where 子句限定修改哪些记录。where 子句是一个条件表达式，该条件表达式类似于 Java 语言的 if，只有符合该条件的记录才会被修改。没有 where 子句则意味着 where 表达式的值总是 true，即该表的所有记录都会被修改。update 语句的语法格式如下：

```
update table_name
set column1= value1[, column2 = value2] …
[WHERE condition];
```

使用 update 语句不仅可以一次修改多条记录，也可以一次修改多列。修改多列都是通过在 set 关键字后使用 column1=value1,column2=value2…来实现的，修改多列的值之间以英文逗号（,）隔开。

下面的 SQL 语句将会把 teacher_table2 表中所有记录的 teacher_name 列的值都改为'孙悟空'。

```
update teacher_table2
set teacher_name = '孙悟空';
```

也可以通过添加 where 条件来指定只修改特定记录，如下 SQL 语句所示：

```
# 只修改 teacher_id 大于 1 的记录
update teacher_table2
set teacher_name = '猪八戒'
where teacher_id > 1;
```

3. delete from 语句

delete from 语句用于删除指定数据表的记录。使用 delete from 语句删除时不需要指定列名，因为总是整行地删除。

使用 delete from 语句可以一次删除多行，删除哪些行采用 where 子句限定，只删除满足 where 条件的记录。没有 where 子句限定将会把表里的全部记录删除。

delete from 语句的语法格式如下：

```
delete from table_name
[WHERE condition];
```

如下 SQL 语句将会把 student_table2 表中的记录全部删除：

```
delete from student_table2;
```

也可以使用 where 条件来限定只删除指定记录，如下 SQL 语句所示：

```
delete from teacher_table2
where teacher_id > 2;
```

注意：

当主表记录被从表记录参照时，主表记录不能被删除，只有先将从表中参照主表记录的所有记录全部删除后，才可删除主表记录。还有一种情况，定义外键约束时定义了主表记录和从表记录之间的级联删除 on delete cascade，或者使用 on delete set null 用于指定当主表记录被删除时，从表中参照该记录的从表记录把外键列的值设为 null。

13.2.9 单表查询

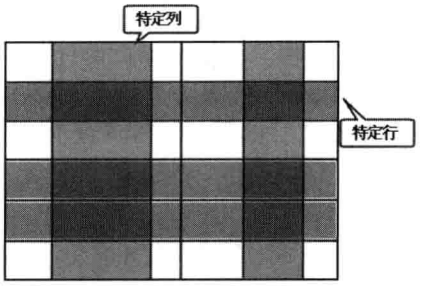


图 13.10　select 语句选择特定行、特定列的示意图

select 语句的功能就是查询数据。select 语句也是 SQL 语句中功能最丰富的语句，select 语句不仅可以执行单表查询，而且可以执行多表连接查询，还可以进行子查询，select 语句用于从一个或多个数据表中选出特定行、特定列的交集。select 语句最简单的功能如图 13.10 所示。

单表查询的 select 语句的语法格式如下：

```
select column1, column2 ...
from 数据源
[where condition]
```

上面语法格式中的数据源可以是表、视图等。从上面的语法格式中可以看出，select 后的列表用于确定选择哪些列，where 条件用于确定选择哪些行，只有满足 where 条件的记录才会被选择出来；如果没有 where 条件，则默认选出所有行。如果想选择出所有列，则可使用星号（*）代表所有列。

下面的 SQL 语句将会选择出 teacher_table 表中的所有行、所有列的数据。

```
select *
from teacher_table;
```

提示：

为了能看到查询的效果，必须准备数据表，并向数据表中插入一些数据，因此在运行本节的 select 语句之前，请先导入 codes\13\13.2\select_data.sql 文件中的 SQL 语句。

如果增加 where 条件，则只选择出符合 where 条件的记录。如下 SQL 语句将选择出 student_table 表中 java_teacher 值大于 3 的记录的 student_name 列的值。

```
select student_name
from student_table
where java_teacher > 3;
```

当使用 select 语句进行查询时，还可以在 select 语句中使用算术运算符（+、－、*、/），从而形成算术表达式。使用算术表达式的规则如下。

- 对数值型数据列、变量、常量可以使用算术运算符（+、－、*、/）创建表达式；
- 对日期型数据列、变量、常量可以使用部分算术运算符（+、－）创建表达式，两个日期之间可以进行减法运算，日期和数值之间可以进行加、减运算；
- 运算符不仅可以在列和常量、变量之间进行运算，也可以在两列之间进行运算。

不论从哪个角度来看，数据列都很像一个变量，只是这个变量的值具有指定的范围——逐行计算表中的每条记录时，数据列的值依次变化。因此能使用变量的地方，基本上都可以使用数据列。

下面的 select 语句中使用了算术运算符。

```
# 数据列实际上可当成一个变量
select teacher_id + 5
from teacher_table;
# 查询出 teacher_table 表中 teacher_id * 3 大于 4 的记录
select *
from teacher_table
where teacher_id * 3 > 4;
```

需要指出的是，select 后的不仅可以是数据列，也可以是表达式，还可以是变量、常量等。例如，如下语句也是正确的。

```
# 数据列实际上可当成一个变量
select 3*5, 20
from teacher_table;
```

SQL 语句中算术运算符的优先级与 Java 语言中的运算符优先级完全相同，乘法和除法的优先级高

于加法和减法，同级运算的顺序是从左到右，表达式中使用括号可强行改变优先级的运算顺序。

MySQL 中没有提供字符串连接运算符，即无法使用加号（+）将字符串常量、字符串变量或字符串列连接起来。MySQL 使用 concat 函数来进行字符串连接运算。

如下 SQL 语句所示：

```
# 选择出 teacher_name 和'xx'字符串连接后的结果
select concat(teacher_name ,'xx')
from teacher_table;
```

对于 MySQL 而言，如果在算术表达式中使用 null，将会导致整个算术表达式的返回值为 null；如果在字符串连接运算中出现 null，将会导致连接后的结果也是 null。如下 SQL 语句将会返回 null。

```
select concat(teacher_name , null)
from teacher_table;
```

对某些数据库而言，如果让字符串和 null 进行连接运算，它会把 null 当成空字符串处理。

如果不希望直接使用列名作为列标题，则可以为数据列或表达式起一个别名，为数据列或表达式起别名时，别名紧跟数据列，中间以空格隔开，或者使用 as 关键字隔开。如下 SQL 语句所示：

```
select teacher_id + 5 as MY_ID
from teacher_table;
```

执行此条 SQL 语句的效果如图 13.11 所示。

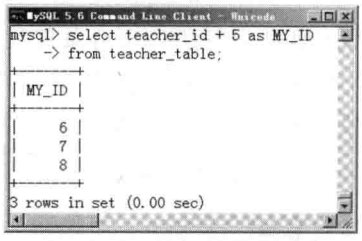


图 13.11 为数据列起别名

从图 13.11 中可以看出，为列起别名，可以改变列的标题头，用于表示计算结果的具体含义。如果列别名中使用特殊字符（例如空格），或者需要强制大小写敏感，都可以通过为别名添加双引号来实现。如下 SQL 语句所示：

```
# 可以为选出的列起别名，别名中包括单引号字符，所以把别名用双引号引起来
select teacher_id + 5  "MY'id"
from teacher_table;
```

如果需要选择多列，并为多列起别名，则列与列之间以逗号隔开，但列和列别名之间以空格隔开。如下 SQL 语句所示：

```
select teacher_id + 5 MY_ID , teacher_name 老师名
from teacher_table;
```

不仅可以为列或表达式起别名，也可以为表起别名，为表起别名的语法和为列或表达式起别名的语法完全一样，如下 SQL 语句所示：

```
select teacher_id + 5 MY_ID , teacher_name 老师名
# 为 teacher_table 起别名 t
from teacher_table t
```

前面已经提到，列名可以当成变量处理，所以运算符也可以在多列之间进行运算，如下 SQL 语句所示：

```
select teacher_id + 5 MY_ID , concat(teacher_name , teacher_id) teacher_name
from teacher_table
where teacher_id * 2 > 3;
```

甚至可以在 select、where 子句中都不出现列名，如下 SQL 语句所示：

```
select 5 + 4
from teacher_table
where 2 < 9;
```

这种情况比较特殊：where 语句后的条件表达式总是 true，所以会把 teacher_table 表中的每条记录都选择出来——但 SQL 语句没有选择任何列，仅仅选择了一个常量，所以 SQL 会把该常量当成一列，teacher_table 表中有多少条记录，该常量就出现多少次。运行上面的 SQL 语句，结果如图 13.12 所示。

对于选择常量的情形，指定数据表可能没有太大的意义，所以 MySQL 提供了一种扩展语法，允许 select 语句后没有 from 子句，即可写成如下形式：

```
select 5 + 4;
```

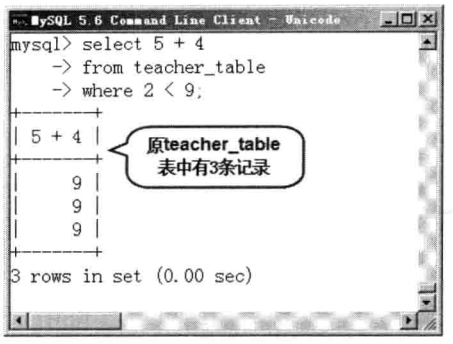


图 13.12　选择常量的结果

上面这种语句并不是标准 SQL 语句。例如，Oracle 就提供了一个名为 dual 的虚表（最新的 MySQL 数据库也支持 dual 虚表），它没有任何意义，仅仅相当于 from 后的占位符。如果选择常量，则可使用如下语句：

```
select 5+4 from dual;
```

select 默认会把所有符合条件的记录全部选出来，即使两行记录完全一样。如果想去除重复行，则可以使用 distinct 关键字从查询结果中清除重复行。比较下面两条 SQL 语句的执行结果：

```
# 选出所有记录，包括重复行
select student_name,java_teacher
from student_table;

# 去除重复行
select distinct student_name,java_teacher
from student_table;
```

注意：

使用 distinct 去除重复行时，distinct 紧跟 select 关键字。它的作用是去除后面字段组合的重复值，而不管对应记录在数据库里是否重复。例如，(1 , 'a' , 'b')和(2 , 'a' , 'b')两条记录在数据库里是不重复的，但如果仅选择后面两列，则 distinct 会认为两条记录重复。

前面已经看到了 where 子句的作用——可以控制只选择指定的行。因为 where 子句里包含的是一个条件表达式，所以可以使用>、>=、<、<=、=和<>等基本的比较运算符。SQL 中的比较运算符不仅可以比较数值之间的大小，也可以比较字符串、日期之间的大小。

注意：

SQL 中判断两个值是否相等的比较运算符是单等号，判断不相等的运算符是<>；SQL 中的赋值运算符不是等号，而是冒号等号（:=）。

除此之外，SQL 还支持如表 13.3 所示的特殊的比较运算符。

表 13.3　特殊的比较运算符

运　算　符	含　义
expr1 between expr2 and expr3	要求 expr1 >= expr2 并且 expr2<=expr3
expr1 in(expr2 , expr3 , expr4 , ...)	要求 expr1 等于后面括号里任意一个表达式的值
like	字符串匹配，like 后的字符串支持通配符
is null	要求指定值等于 null

下面的 SQL 语句选出 student_id 大于等于 2，且小于等于 4 的所有记录。

```
select * from student_table
where student_id between 2 and 4;
```

使用 between val1 and val2 必须保证 val1 小于 val2，否则将选不出任何记录。除此之外，between val1

and val2 中的两个值不仅可以是常量，也可以是变量，或者是列名也行。如下 SQL 语句选出 java_teacher 小于等于 2，student_id 大于等于 2 的所有记录。

```
select * from student_table
where 2 between java_teacher and student_id;
```

使用 in 比较运算符时，必须在 in 后的括号里列出一个或多个值，它要求指定列必须与 in 括号里任意一个值相等。如下 SQL 语句所示：

```
# 选出 student_id 为 2 或 4 的所有记录
select * from student_table
where student_id in(2, 4);
```

与之类似的是，in 括号里的值既可以是常量，也可以是变量或者列名，如下 SQL 语句所示：

```
# 选出 student_id、java_teacher 列的值为 2 的所有记录
select * from student_table
where 2 in(student_id, java_teacher);
```

like 运算符主要用于进行模糊查询，例如，若要查询名字以“孙”开头的所有记录，这就需要用到模糊查询，在模糊查询中需要使用 like 关键字。SQL 语句中可以使用两个通配符：下画线（_）和百分号（%），其中下画线可以代表一个任意的字符，百分号可以代表任意多个字符。如下 SQL 语句将查询出所有学生中名字以“孙”开头的学生。

```
select * from student_table
where student_name like '孙%';
```

下面的 SQL 语句将查询出名字为两个字符的所有学生。

```
select * from student_table
# 下面使用两个下画线代表两个字符
where student_name like '__';
```

在某些特殊的情况下，查询的条件里需要使用下画线或百分号，不希望 SQL 把下画线和百分号当成通配符使用，这就需要使用转义字符，MySQL 使用反斜线（\）作为转义字符，如下 SQL 语句所示：

```
# 选出所有名字以下画线开头的学生
select * from student_table
where student_name like '\_%';
```

标准 SQL 语句并没有提供反斜线（\）的转义字符，而是使用 escape 关键字显式进行转义。例如，为了实现上面功能需要使用如下 SQL 语句：

```
#在标准的 SQL 中选出所有名字以下画线开头的学生
select * from student_table
where student_name like '\_%' escape '\';
```

is null 用于判断某些值是否为空，判断是否为空不要用=null 来判断，因为 SQL 中 null=null 返回 null。如下 SQL 语句将选择出 student_table 表中 student_name 为 null 的所有记录。

```
select * from student_table
where student_name is null;
```

如果 where 子句后有多个条件需要组合，SQL 提供了 and 和 or 逻辑运算符来组合两个条件，并提供了 not 来对逻辑表达式求否。如下 SQL 语句将选出学生名字为 2 个字符，且 student_id 大于 3 的所有记录。

```
select * from student_table
# 使用 and 来组合多个条件
where student_name like '__' and student_id > 3;
```

下面的 SQL 语句将选出 student_table 表中姓名不以下画线开头的所有记录。

```
select * from student_table
# 使用 not 对 where 条件取否
where not student_name like '\_%';
```

当使用比较运算符、逻辑运算符来连接表达式时，必须注意这些运算符的优先级。SQL 中比较运

算符、逻辑运算符的优先级如表 13.4 所示。

表 13.4　SQL 中比较运算符、逻辑运算符的优先级

运　算　符	优先级（优先级小的优先）
所有的比较运算符	1
not	2
and	3
or	4

如果 SQL 代码需要改变优先级的默认顺序，则可以使用括号，括号的优先级比所有的运算符高。如下 SQL 语句使用括号来改变逻辑运算符的优先级。

```
select * from student_table
# 使用括号强制先计算 or 运算
where (student_id > 3 or student_name > '张')
   and java_teacher > 1;
```

执行查询后的查询结果默认按插入顺序排列；如果需要查询结果按某列值的大小进行排序，则可以使用 order by 子句。order by 子句的语法格式如下：

```
order by column_name1 [desc] , column_name2 …
```

进行排序时默认按升序排列，如果强制按降序排列，则需要在列后使用 desc 关键字（与之对应的是 asc 关键字，用不用该关键字的效果完全一样，因为默认是按升序排列）。

上面语法中设定排序列时可采用列名、列序号和列别名。如下 SQL 语句选出 student_table 表中的所有记录，选出后按 java_teacher 列的升序排列。

```
select * from student_table
order by java_teacher;
```

如果需要按多列排序，则每列的 asc、desc 必须单独设定。如果指定了多个排序列，则第一个排序列是首要排序列，只有当第一列中存在多个相同的值时，第二个排序列才会起作用。如下 SQL 语句先按 java_teacher 列的降序排列，当 java_teacher 列的值相同时按 student_name 列的升序排列。

```
select * from student_table
order by java_teacher desc , student_name;
```

13.2.10　数据库函数

正如前面看到的连接字符串使用的 concat 函数，每个数据库都会在标准的 SQL 基础上扩展一些函数，这些函数用于进行数据处理或复杂计算，它们通过对一组数据进行计算，得到最终需要的输出结果。函数一般都会有一个或者多个输入，这些输入被称为函数的参数，函数内部会对这些参数进行判断和计算，最终只有一个值作为返回值。函数可以出现在 SQL 语句的各个位置，比较常用的位置是 select 之后和 where 子句中。

根据函数对多行数据的处理方式，函数被分为单行函数和多行函数，单行函数对每行输入值单独计算，每行得到一个计算结果返回给用户；多行函数对多行输入值整体计算，最后只会得到一个结果。单行函数和多行函数的示意图如图 13.13 所示。

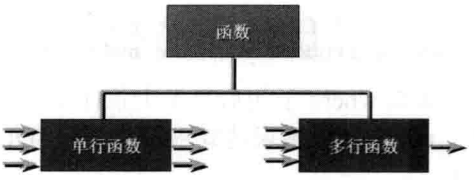


图 13.13　单行函数和多行函数的示意图

SQL 中的函数和 Java 语言中的方法有点相似，但 SQL 中的函数是独立的程序单元，也就是说，调用函数时无须使用任何类、对象作为调用者，而是直接执行函数。执行函数的语法如下：

```
function_name(arg1, arg2 ...)
```

多行函数也称为聚集函数、分组函数，主要用于完成一些统计功能，在大部分数据库中基本相同。但不同数据库中的单行函数差别非常大，MySQL 中的单行函数具有如下特征。

- 单行函数的参数可以是变量、常量或数据列。单行函数可以接收多个参数，但只返回一个值。

- 单行函数会对每行单独起作用，每行（可能包含多个参数）返回一个结果。
- 使用单行函数可以改变参数的数据类型。单行函数支持嵌套使用，即内层函数的返回值是外层函数的参数。

MySQL 的单行函数分类如图 13.14 所示。

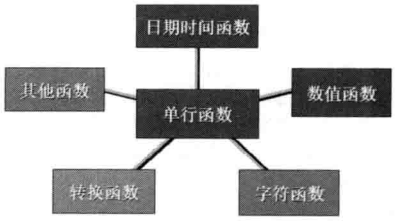


图 13.14　MySQL 的单行函数分类

MySQL 数据库的数据类型大致分为数值型、字符型和日期时间型，所以 MySQL 分别提供了对应的函数。转换函数主要负责完成类型转换，其他函数又大致分为如下几类。

- 位函数
- 流程控制函数
- 加密解密函数
- 信息函数

每个数据库都包含了大量的单行函数，这些函数的用法也存在一些差异，但有一点是相同的——每个数据库都会为一些常用的计算功能提供相应的函数，这些函数的函数名可能不同，用法可能有差异，但所有数据库提供的函数库所能完成的功能大致相似，读者可以参考各数据库系统的参考文档来学习这些函数的用法。下面通过一些例子来介绍 MySQL 单行函数的用法。

```
# 选出 teacher_table 表中 teacher_name 列的字符长度
select char_length(teacher_name)
from teacher_table;
# 计算 teacher_name 列的字符长度的 sin 值
select sin(char_length(teacher_name))
from teacher_table;
# 计算 1.57 的 sin 值，约等于 1
select sin(1.57);
# 为指定日期添加一定的时间
# 在这种用法下 interval 是关键字，需要一个数值，还需要一个单位
SELECT DATE_ADD('1998-01-02', interval 2 MONTH);
# 这种用法更简单
select ADDDATE('1998-01-02',3);
# 获取当前日期
select CURDATE();
# 获取当前时间
select curtime();
# 下面的 MD5 是 MD5 加密函数
select MD5('testing');
```

MySQL 提供了如下几个处理 null 的函数。

- ifnull(expr1,expr2)：如果 expr1 为 null，则返回 expr2，否则返回 expr1。
- nullif(expr1,expr2)：如果 erpr1 和 expr2 相等，则返回 null，否则返回 expr1。
- if(expr1,expr2,expr3)：有点类似于?:三目运算符，如果 expr1 为 true，不等于 0，且不等于 null，则返回 expr2，否则返回 expr3。
- isnull(expr1)：判断 expr1 是否为 null，如果为 null 则返回 true，否则返回 false。

```
# 如果 student_name 列为 null，则返回'没有名字'
select ifnull(student_name,'没有名字')
from student_table;
# 如果 student_name 列等于'张三'，则返回 null
select nullif(student_name,'张三')
from student_table;
# 如果 student_name 列为 null，则返回'没有名字'，否则返回'有名字'
select if(isnull(student_name),'没有名字', '有名字')
from student_table;
```

MySQL 还提供了一个 case 函数，该函数是一个流程控制函数。case 函数有两个用法，case 函数第一个用法的语法格式如下：

```
case value
when compare_value1 then result1
when compare_value2 then result2
```

```
...
else result
end
```

case 函数用 value 和后面的 compare_value1、compare_value2、…依次进行比较，如果 value 和指定的 compare_value1 相等，则返回对应的 result1，否则返回 else 后的 result。例如如下 SQL 语句：

```
# 如果 java_teacher 为 1，则返回'Java 老师'，为 2 返回'Ruby 老师'，否则返回'其他老师'
select student_name , case java_teacher
when 1 then 'Java 老师'
when 2 then 'Ruby 老师'
else '其他老师'
end
from student_table;
```

case 函数第二个用法的语法格式如下：

```
case
when condition1 then result1
when condition2 then result2
...
else result
end
```

在第二个用法中，condition1、condition2 都是一个返回 boolean 值的条件表达式，因此这种用法更加灵活。例如如下 SQL 语句：

```
# id 小于 3 的为初级班，3~6 的为中级班，其他的为高级班
select student_name,case
when student_id<=3 then '初级班'
when student_id<=6 then '中级班'
else '高级班'
end
from student_table;
```

虽然此处介绍了一些 MySQL 常用函数的简单用法，但通常不推荐在 Java 程序中使用特定数据库的函数，因为这将导致程序代码与特定数据库耦合；如果需要把该程序移植到其他数据库系统上时，可能需要打开源程序，重新修改 SQL 语句。

13.2.11 分组和组函数

组函数也就是前面提到的多行函数，组函数将一组记录作为整体计算，每组记录返回一个结果，而不是每条记录返回一个结果。常用的组函数有如下 5 个。

- avg([distinct|all]expr)：计算多行 expr 的平均值，其中，expr 可以是变量、常量或数据列，但其数据类型必须是数值型。还可以在变量、列前使用 distinct 或 all 关键字，如果使用 distinct，则表明不计算重复值；all 用和不用的效果完全一样，表明需要计算重复值。
- count({ *|[distinct|all]expr})：计算多行 expr 的总条数，其中，expr 可以是变量、常量或数据列，其数据类型可以是任意类型；用星号（*）表示统计该表内的记录行数；distinct 表示不计算重复值。
- max(expr)：计算多行 expr 的最大值，其中 expr 可以是变量、常量或数据列，其数据类型可以是任意类型。
- min(expr)：计算多行 expr 的最小值，其中 expr 可以是变量、常量或数据列，其数据类型可以是任意类型。
- sum([distinct|all]expr)：计算多行 expr 的总和，其中，expr 可以是变量、常量或数据列，但其数据类型必须是数值型；distinct 表示不计算重复值。

```
# 计算 student_table 表中的记录条数
select count(*)
from student_table;
# 计算 java_teacher 列总共有多少个值
select count(distinct java_teacher)
from student_table;
```

```
# 统计所有 student_id 的总和
select sum(student_id)
from student_table;
# 计算的结果是 20 * 记录的行数
select sum(20)
from student_table;
# 选出 student_table 表中 student_id 最大的值
select max(student_id)
from student_table;
# 选出 teacher_table 表中 teacher_id 最小的值
select min(teacher_id)
from teacher_table;
# 因为 sum 里的 expr 是常量 34，所以每行的值都相同
# 使用 distinct 强制不计算重复值，所以下面计算结果为 34
select sum(distinct 34)
from student_table;
# 使用 count 统计记录行数时，null 不会被计算在内
select count(student_name)
from student_table;
```

对于可能出现 null 的列，可以使用 ifnull 函数来处理该列。

```
# 计算 java_teacher 列所有记录的平均值
select avg(ifnull(java_teacher , 0))
from student_table;
```

值得指出的是，distinct 和*不同时使用，如下 SQL 语句有错误。

```
select count(distinct *)
from student_table;
```

在默认情况下，组函数会把所有记录当成一组，为了对记录进行显式分组，可以在 select 语句后使用 group by 子句，group by 子句后通常跟一个或多个列名，表明查询结果根据一列或多列进行分组——当一列或多列组合的值完全相同时，系统会把这些记录当成一组。如下 SQL 语句所示：

```
# count(*)将会对每组得到一个结果
select count(*)
from student_table
# 将 java_teacher 列值相同的记录当成一组
group by java_teacher;
```

如果对多列进行分组，则要求多列的值完全相同才会被当成一组。如下 SQL 语句所示：

```
select count(*)
from student_table
# 当 java_teacher、student_name 两列的值完全相同时才会被当成一组
group by java_teacher , student_name;
```

对于很多数据库而言，分组计算时有严格的规则——如果查询列表中使用了组函数，或者 select 语句中使用了 group by 分组子句，则要求出现在 select 列表中的字段，要么使用组函数包起来，要么必须出现在 group by 子句中。这条规则很容易理解，因为一旦使用了组函数或使用了 group by 子句，都将导致多条记录只有一条输出，系统无法确定输出多条记录中的哪一条记录。

对于 MySQL 来说，并没有上面的规则要求，如果某个数据列既没有出现在 group by 之后，也没有使用组函数包起来，则 MySQL 会输出该列的第一条记录的值。图 13.15 显示了 MySQL 的处理结果。

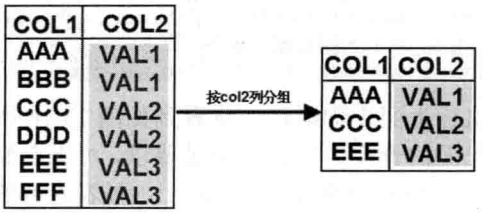


图 13.15 MySQL 处理不在 group by、组函数中的列

如果需要对分组进行过滤，则应该使用 having 子句，having 子句后面也是一个条件表达式，只有满足该条件表达式的分组才会被选出来。having 子句和 where 子句非常容易混淆，它们都有过滤功能，但它们有如下区别。

- 不能在 where 子句中过滤组，where 子句仅用于过滤行。过滤组必须使用 having 子句。
- 不能在 where 子句中使用组函数，having 子句才可使用组函数。

如下 SQL 语句所示：

```
select *
from student_table
group by java_teacher
# 对组进行过滤
having count(*) > 2;
```

13.2.12　多表连接查询

很多时候，需要选择的数据并不是来自一个表，而是来自多个数据表，这就需要使用多表连接查询。例如，对于上面的 student_table 和 teacher_table 两个数据表，如果希望查询出所有学生以及他的老师名字，这就需要从两个表中取数据。

多表连接查询有两种规范，较早的 SQL 92 规范支持如下几种多表连接查询。

- 等值连接。
- 非等值连接。
- 外连接。
- 广义笛卡儿积。

SQL 99 规范提供了可读性更好的多表连接语法，并提供了更多类型的连接查询。SQL 99 支持如下几种多表连接查询。

- 交叉连接。
- 自然连接。
- 使用 using 子句的连接。
- 使用 on 子句的连接。
- 全外连接或者左、右外连接。

1. SQL 92 的连接查询

SQL 92 的多表连接语法比较简洁，这种语法把多个数据表都放在 from 之后，多个表之间以逗号隔开；连接条件放在 where 之后，与查询条件之间用 and 逻辑运算符连接。如果连接条件要求两列值相等，则称为等值连接，否则称为非等值连接；如果没有任何连接条件，则称为广义笛卡儿积。SQL 92 中多表连接查询的语法格式如下：

```
select column1 , column2 ...
from table1, table2 ...
[where join_condition]
```

多表连接查询中可能出现两个或多个数据列具有相同的列名，则需要在这些同名列之间使用表名前缀或表别名前缀作为限制，避免系统混淆。

实际上，所有的列都可以增加表名前缀或表别名前缀。只是进行单表查询时，绝不可能出现同名列，所以系统不可能混淆，因此通常省略表名前缀。

如下 SQL 语句查询出所有学生的资料以及对应的老师姓名。

```
select s.* , teacher_name
# 指定多个数据表，并指定表别名
from student_table s , teacher_table t
# 使用 where 指定连接条件
where s.java_teacher = t.teacher_id;
```

执行上面查询语句，将看到如图 13.16 所示的结果。

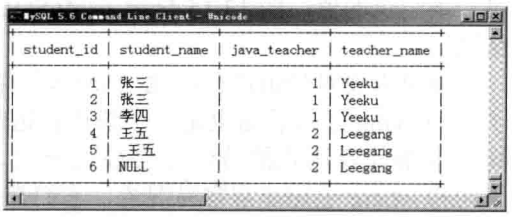


图 13.16　等值连接查询的结果

上面的查询结果正好满足要求，可以看到每个学生以及他对应的老师的名字。实际上，多表查询的过程可理解成一个嵌套循环，这个嵌套循环的伪码如下：

```
// 依次遍历 teacher_table 表中的每条记录
for t in teacher_table
{
    // 遍历 student_table 表中的每条记录
    for s in student_table
```

```
    {
        // 当满足连接条件时，输出两个表连接后的结果
        if (s.java_teacher = t.teacher_id)
            output s + t
    }
}
```

理解了上面的伪码之后，接下来即可很轻易地理解多表连接查询的运行机制。如果求广义笛卡儿积，则 where 子句后没有任何连接条件，相当于没有上面的 if 语句，广义笛卡儿积的结果会有 $n \times m$ 条记录。只要把 where 后的连接条件去掉，就可以得到广义笛卡儿积，如下 SQL 语句所示：

```
# 不使用连接条件，得到广义笛卡儿积
select s.* , teacher_name
# 指定多个数据表，并指定表别名
from student_table s , teacher_table t;
```

与此类似的是，非等值连接的执行结果可以使用上面的嵌套循环来计算，如下 SQL 语句所示：

```
select s.* , teacher_name
# 指定多个数据表，并指定表别名
from student_table s , teacher_table t
# 使用 where 指定连接条件，非等值连接
where s.java_teacher > t.teacher_id;
```

上面 SQL 语句的执行结果相当于 if 条件换成了 s.java_teacher > t.teacher_id。

如果还需要对记录进行过滤，则将过滤条件和连接条件使用 and 连接起来，如下 SQL 语句所示：

```
select s.* , teacher_name
# 指定多个数据表，并指定表别名
from student_table s , teacher_table t
# 使用 where 指定连接条件，并指定 student_name 列不能为 null
where s.java_teacher = t.teacher_id and student_name is not null;
```

虽然 MySQL 不支持 SQL 92 中的左外连接、右外连接，但本书还是有必要了解一下 SQL 92 中的左外连接和右外连接。SQL 92 中的外连接就是在连接条件的列名后增加括号包起来的外连接符（+或*，不同的数据库有一定的区别），当外连接符出现在左边时称为左外连接，出现在右边时则称为右外连接。如下 SQL 语句所示：

```
select s.* , teacher_name
from student_table s , teacher_table t
# 右外连接
where s.java_teacher = t.teacher_id(*);
```

外连接就是在外连接符所在的表中增加一个“万能行”，这行记录的所有数据都是 null，而且该行可以与另一个表中所有不满足条件的记录进行匹配，通过这种方式就可以把另一个表中的所有记录选出来，不管这些记录是否满足连接条件。

除此之外，还有一种自连接，正如前面介绍外键约束时提到的自关联，如果同一个表中的不同记录之间存在主、外键约束关联，例如把员工、经理保存在同一个表里，则需要使用自连接查询。

注意：

自连接只是连接的一种用法，并不是一种连接类型，不管是 SQL 92 还是 SQL 99 都可以使用自连接查询。自连接的本质就是把一个表当成两个表来用。

下面的 SQL 语句建立了一个自关联的数据表，并向表中插入了 4 条数据。

```
create table emp_table
(
    emp_id int auto_increment primary key,
    emp_name varchar(255),
    manager_id int,
    foreign key(manager_id) references emp_table(emp_id)
);
insert into emp_table
values(null , '唐僧' , null),
```

```
(null , '孙悟空' , 1),
(null , '猪八戒' , 1),
(null , '沙僧' , 1);
```

如果需要查询该数据表中的所有员工名，以及每个员工对应的经理名，则必须使用自连接查询。所谓自连接就是把一个表当成两个表来用，这就需要为一个表起两个别名，而且查询中用的所有数据列都要加表别名前缀，因为两个表的数据列完全一样。下面的自连接查询可以查询出所有的员工名，以及对应的经理名。

```
select emp.emp_id,emp.emp_name 员工名,mgr.emp_name 经理名
from emp_table emp,emp_table mgr
where emp.manager_id = mgr.emp_id;
```

2．SQL 99 的连接查询

SQL 99 的连接查询与 SQL 92 的连接查询原理基本相似，不同的是 SQL 99 连接查询的可读性更强——查询用的多个数据表显式使用 xxx join 连接，而不是直接依次排列在 from 之后，from 后只需要放一个数据表；连接条件不再放在 where 之后，而是提供了专门的连接条件子句。

- **交叉连接（cross join）**：交叉连接效果就是 SQL 92 中的广义笛卡儿积，所以交叉连接无须任何连接条件，如下 SQL 语句所示：

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# cross join 交叉连接，相当于广义笛卡儿积
cross join teacher_table t;
```

- **自然连接（natural join）**：自然连接表面上看起来也无须指定连接条件，但自然连接是有连接条件的，自然连接会以两个表中的同名列作为连接条件；如果两个表中没有同名列，则自然连接与交叉连接效果完全一样——因为没有连接条件。如下 SQL 语句所示：

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# natural join 自然连接使用两个表中的同名列作为连接条件
natural join teacher_table t;
```

- **using 子句连接**：using 子句可以指定一列或多列，用于显式指定两个表中的同名列作为连接条件。假设两个表中有超过一列的同名列，如果使用 natural join，则会把所有的同名列当成连接条件；使用 using 子句，就可显式指定使用哪些同名列作为连接条件。如下 SQL 语句所示：

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# join 连接另一个表
join teacher_table t
using(teacher_id);
```

运行上面语句将出现一个错误，因为 student_table 表中并不存在名为 teacher_id 的列。也就是说，如果使用 using 子句来指定连接条件，则两个表中必须有同名列，否则就会出现错误。

- **on 子句连接**：这是最常用的连接方式，SQL 99 语法的连接条件放在 on 子句中指定，而且每个 on 子句只指定一个连接条件。这意味着：如果需要进行 *N* 表连接，则需要有 *N*-1 个 join...on 对。如下 SQL 语句所示：

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# join 连接另一个表
join teacher_table t
# 使用 on 来指定连接条件
on s.java_teacher = t.teacher_id;
```

使用 on 子句的连接完全可以代替 SQL 92 中的等值连接、非等值连接，因为 on 子句的连接条

件除了等值条件之外，也可以是非等值条件。如下 SQL 语句就是 SQL 99 中的非等值连接。

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# join 连接另一个表
join teacher_table t
# 使用 on 来指定连接条件：非等值连接
on s.java_teacher > t.teacher_id;
```

- **左、右、全外连接**：这三种外连接分别使用 left [outer] join、right [outer] join 和 full [outer] join，这三种外连接的连接条件一样通过 on 子句来指定，既可以是等值连接条件，也可以是非等值连接条件。

下面使用右外连接，连接条件是非等值连接。

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# right join 右外连接另一个表
right join teacher_table t
# 使用 on 来指定连接条件，使用非等值连接
on s.java_teacher < t.teacher_id;
```

下面使用左外连接，连接条件是非等值连接。

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# left join 左外连接另一个表
left join teacher_table t
# 使用 on 来指定连接条件，使用非等值连接
on s.java_teacher > t.teacher_id;
```

运行上面两条外连接语句并查看它们的运行结果，不难发现 SQL 99 外连接与 SQL 92 外连接恰好相反，SQL99 左外连接将会把左边表中所有不满足连接条件的记录全部列出；SQL 99 右外连接将会把右边表中所有不满足连接条件的记录全部列出。

下面的 SQL 语句使用全外连接，连接条件是等值连接。

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# full join 全外连接另一个表
full join teacher_table t
# 使用 on 来指定连接条件，使用等值连接
on s.java_teacher = t.teacher_id;
```

SQL 99 的全外连接将会把两个表中所有不满足连接条件的记录全部列出。

注意：

运行上面查询语句时会出现错误，这是因为 MySQL 并不是全外连接。

13.2.13 子查询

子查询就是指在查询语句中嵌套另一个查询，子查询可以支持多层嵌套。对于一个普通的查询语句而言，子查询可以出现在两个位置。

- 出现在 from 语句后当成数据表，这种用法也被称为行内视图，因为该子查询的实质就是一个临时视图。
- 出现在 where 条件后作为过滤条件的值。

使用子查询时要注意如下几点。

- 子查询要用括号括起来。
- 把子查询当成数据表时（出现在 from 之后），可以为该子查询起别名，尤其是作为前缀来限定

数据列时，必须给子查询起别名。

- 把子查询当成过滤条件时，将子查询放在比较运算符的右边，这样可以增强查询的可读性。
- 把子查询当成过滤条件时，单行子查询使用单行运算符，多行子查询使用多行运算符。

对于把子查询当成数据表是完全把子查询当做数据表来用，只是把之前的表名变成子查询（也可为子查询起别名），其他部分与普通查询没有任何区别。下面的 SQL 语句示范了把子查询当成数据表的用法。

```
select *
# 把子查询当成数据表
from (select * from student_table) t
where t.java_teacher > 1;
```

把子查询当成数据表的用法更准确地说是当成视图，可以把上面的 SQL 语句理解成在执行查询时创建了一个临时视图，该视图名为 t，所以这种临时创建的视图也被称为行内视图。理解了这种子查询的实质后，不难知道这种子查询可以完全代替查询语句中的数据表，包括在多表连接查询中使用这种子查询。

还有一种情形：把子查询当成 where 条件中的值，如果子查询返回单行、单列值，则被当成一个标量值使用，也就可以使用单行记录比较运算符。例如如下 SQL 语句：

```
select *
from student_table
where java_teacher >
# 返回单行、单列的子查询可以当成标量值使用
(select teacher_id
from teacher_table
 where teacher_name='Yeeku');
```

上面查询语句中的子查询（粗体字部分）将返回一个单行、单列值（该值就是 1），如果把上面查询语句的括号部分换成 1，那么这条语句就再简单不过了——实际上，这就是这种子查询的实质，单行、单列子查询的返回值被当成标量值处理。

如果子查询返回多个值，则需要使用 in、any 和 all 等关键字，in 可以单独使用，与前面介绍比较运算符时所讲的 in 完全一样，此时可以把子查询返回的多个值当成一个值列表。如下 SQL 语句所示：

```
select *
from student_table
where student_id in
(select teacher_id
 from teacher_table);
```

上面查询语句中的子查询（粗体字部分）将返回多个值，这多个值将被当成一个值列表，只要 student_id 与该值列表中的任意一个值相等，就可以选出这条记录。

any 和 all 可以与>、>=、<、<=、<>、=等运算符结合使用，与 any 结合使用分别表示大于、大于等于、小于、小于等于、不等于、等于其中任意一个值；与 all 结合使用分别表示大于、大于等于、小于、小于等于、不等于、等于全部值。从上面介绍中可以看出，=any 的作用与 in 的作用相同。如下 SQL 语句使用=any 来代替上面的 in。

```
select *
from student_table
where student_id =
any(select teacher_id
 from teacher_table);
```

<ANY 只要小于值列表中的最大值即可，>ANY 只要大于值列表中的最小值即可。<All 要求小于值列表中的最小值，>All 要求大于值列表中的最大值。

下面的 SQL 语句选出 student_table 表中 student_id 大于 teacher_table 表中所有 teacher_id 的记录。

```
select *
from student_table
where student_id >
all(select teacher_id
 from teacher_table);
```

还有一种子查询可以返回多行、多列，此时 where 子句中应该有对应的数据列，并使用圆括号将多

个数据列组合起来。如下 SQL 语句所示：

```
select *
from student_table
where (student_id,student_name)
=any(select teacher_id, teacher_name
 from teacher_table);
```

13.2.14 集合运算

select 语句查询的结果是一个包含多条数据的结果集，类似于数学里的集合，可以进行交（intersect）、并（union）和差（minus）运算，select 查询得到的结果集也可能需要进行这三种运算。

为了对两个结果集进行集合运算，这两个结果集必须满足如下条件。

- 两个结果集所包含的数据列的数量必须相等。
- 两个结果集所包含的数据列的数据类型也必须一一对应。

1. union 运算

union 运算的语法格式如下：

```
select 语句 union select 语句
```

下面的 SQL 语句查询出所有教师的信息和主键小于 4 的学生信息。

```
# 查询结果集包含两列，第一列为 int 类型，第二列为 varchar 类型
select * from teacher_table
union
# 这个结果集的数据列必须与前一个结果集的数据列一一对应
select student_id , student_name from student_table;
```

2. minus 运算

minus 运算的语法格式如下：

```
select 语句 minus select 语句
```

上面的语法格式十分简单，不过很遗憾，MySQL 并不支持使用 minus 运算符，因此只能借助于子查询来“曲线”实现上面的 minus 运算。

假如想从所有学生记录中“减去”与老师记录的 ID 相同、姓名相同的记录，则可进行如下的 minus 运算：

```
select student_id , student_name from student_table
minus
# 两个结果集的数据列的数量相等，数据类型一一对应，可以进行 minus 运算
select teacher_id , teacher_name from teacher_table;
```

不过，MySQL 并不支持这种运算。但可以通过如下子查询来实现上面运算。

```
select student_id , student_name from student_table
where(student_id , student_name)
not in
(select teacher_id , teacher_name from teacher_table);
```

3. intersect 运算

intersect 运算的语法格式如下：

```
select 语句 intersect select 语句
```

上面的语法格式十分简单，不过很遗憾，MySQL 并不支持使用 intersect 运算符，因此只能借助于多表连接查询来“曲线”实现上面的 intersect 运算。

假如想找出学生记录中与老师记录中的 ID 相同、姓名相同的记录，则可进行如下的 intersect 运算：

```
select student_id , student_name from student_table
intersect
# 两个结果集的数据列的数量相等，数据类型一一对应，可以进行 intersect 运算
select teacher_id , teacher_name from teacher_table;
```

不过，MySQL 并不支持这种运算。但可以通过如下多表连接查询来实现上面运算。

```
select student_id , student_name from student_table
join
teacher_table
on(student_id=teacher_id and student_name=teacher_name);
```

需要指出的是，如果进行 intersect 运算的两个 select 子句中都包括了 where 条件，那么将 intersect 运算改写成多表连接查询后还需要将两个 where 条件进行 and 运算。假如有如下 intersect 运算的 SQL 语句：

```
select student_id , student_name from student_table where student_id<4
intersect
# 两个结果集的数据列的数量相等，数据类型一一对应，可以进行 intersect 运算
select teacher_id , teacher_name from teacher_table where teacher_name like '李%';
```

上面语句改写如下：

```
select student_id , student_name from student_table
join
teacher_table
on(student_id=teacher_id and student_name=teacher_name)
where student_id<4 and teacher_name like '李%';
```

13.3 JDBC 的典型用法

掌握了标准的 SQL 命令语法之后，就可以开始使用 JDBC 开发数据库应用了。

13.3.1 JDBC 4.2 常用接口和类简介

Java 8 支持 JDBC 4.2 标准，JDBC 4.2 在原有 JDBC 标准上增加了一些新特性。下面介绍这些 JDBC API 时会提到 Java 8 新增的功能。

- DriverManager：用于管理 JDBC 驱动的服务类。程序中使用该类的主要功能是获取 Connection 对象，该类包含如下方法。
  - public static synchronized Connection getConnection(String url, String user, String pass) throws SQLException：该方法获得 url 对应数据库的连接。
- Connection：代表数据库连接对象，每个 Connection 代表一个物理连接会话。要想访问数据库，必须先获得数据库连接。该接口的常用方法如下。
  - Statement createStatement() throws SQLExcetpion：该方法返回一个 Statement 对象。
  - PreparedStatement prepareStatement(String sql) throws SQLExcetpion：该方法返回预编译的 Statement 对象，即将 SQL 语句提交到数据库进行预编译。
  - CallableStatement prepareCall(String sql) throws SQLExcetpion：该方法返回 CallableStatement 对象，该对象用于调用存储过程。

上面三个方法都返回用于执行 SQL 语句的 Statement 对象，PreparedStatement、CallableStatement 是 Statement 的子类，只有获得了 Statement 之后才可执行 SQL 语句。

除此之外，Connection 还有如下几个用于控制事务的方法。

- Savepoint setSavepoint()：创建一个保存点。
- Savepoint setSavepoint(String name)：以指定名字来创建一个保存点。
- void setTransactionIsolation(int level)：设置事务的隔离级别。
- void rollback()：回滚事务。
- void rollback(Savepoint savepoint)：将事务回滚到指定的保存点。
- void setAutoCommit(boolean autoCommit)：关闭自动提交，打开事务。
- void commit()：提交事务。

Java 7 为 Connection 新增了 setSchema(String schema)、getSchema()两个方法，这两个方法用于控制该 Connection 访问的数据库 Schema。Java 7 还为 Connection 新增了 setNetworkTimeout(Executor executor,

int milliseconds)、getNetworkTimeout()两个方法来控制数据库连接的超时行为。

- Statement：用于执行 SQL 语句的工具接口。该对象既可用于执行 DDL、DCL 语句，也可用于执行 DML 语句，还可用于执行 SQL 查询。当执行 SQL 查询时，返回查询到的结果集。它的常用方法如下。
  - ResultSet executeQuery(String sql)throws SQLException：该方法用于执行查询语句，并返回查询结果对应的 ResultSet 对象。该方法只能用于执行查询语句。
  - int executeUpdate(String sql)throws SQLExcetion：该方法用于执行 DML 语句，并返回受影响的行数；该方法也可用于执行 DDL 语句，执行 DDL 语句将返回 0。
  - boolean execute(String sql)throws SQLException：该方法可执行任何 SQL 语句。如果执行后第一个结果为 ResultSet 对象，则返回 true；如果执行后第一个结果为受影响的行数或没有任何结果，则返回 false。

Java 7 为 Statement 新增了 closeOnCompletion()方法，如果 Statement 执行了该方法，则当所有依赖于该 Statement 的 ResultSet 关闭时，该 Statement 会自动关闭。Java 7 还为 Statement 提供了一个 isCloseOnCompletion()方法，该方法用于判断该 Statement 是否打开了“closeOnCompletion”。

Java 8 为 Statement 新增了多个重载的 executeLargeUpdate()方法，这些方法相当于增强版的 executeUpdate()方法，返回值类型为 long——也就是说，当 DML 语句影响的记录条数超过 Integer.MAX_VALUE 时，就应该使用 executeLargeUpdate()方法。

提示：

考虑到目前应用程序所处理的数据量越来越大，使用 executeLargeUpdate()方法具有更好的适应性。但遗憾的是，目前最新的 MySQL 驱动暂不支持该方法。

- PreparedStatement：预编译的 Statement 对象。PreparedStatement 是 Statement 的子接口，它允许数据库预编译 SQL 语句（这些 SQL 语句通常带有参数），以后每次只改变 SQL 命令的参数，避免数据库每次都需要编译 SQL 语句，因此性能更好。相对于 Statement 而言，使用 PreparedStatement 执行 SQL 语句时，无须再传入 SQL 语句，只要为预编译的 SQL 语句传入参数值即可。所以它比 Statement 多了如下方法。
  - void setXxx(int parameterIndex,Xxx value)：该方法根据传入参数值的类型不同，需要使用不同的方法。传入的值根据索引传给 SQL 语句中指定位置的参数。

注意：

PreparedStatement 同样有 executeUpdate()、executeQuery()和 execute()三个方法，只是这三个方法无须接收 SQL 字符串，因为 PreparedStatement 对象已经预编译了 SQL 命令，只要为这些命令传入参数即可。Java 8 还为 PreparedStatement 增加了不带参数的 executeLargeUpdate()方法——执行 DML 语句影响的记录条数可能超过 Integer.MAX_VALUE 时，就应该使用 executeLargeUpdate()方法。

- **ResultSet**：结果集对象。该对象包含访问查询结果的方法，ResultSet 可以通过列索引或列名获得列数据。它包含了如下常用方法来移动记录指针。
  - void close()：释放 ResultSet 对象。
  - boolean absolute(int row)：将结果集的记录指针移动到第 row 行，如果 row 是负数，则移动到倒数第 row 行。如果移动后的记录指针指向一条有效记录，则该方法返回 true。
  - void beforeFirst()：将 ResultSet 的记录指针定位到首行之前，这是 ResultSet 结果集记录指针的初始状态——记录指针的起始位置位于第一行之前。
  - boolean first()：将 ResultSet 的记录指针定位到首行。如果移动后的记录指针指向一条有效记录，则该方法返回 true。
  - boolean previous()：将 ResultSet 的记录指针定位到上一行。如果移动后的记录指针指向一条有效记录，则该方法返回 true。
  - boolean next()：将 ResultSet 的记录指针定位到下一行，如果移动后的记录指针指向一条有效

记录，则该方法返回 true。

- boolean last()：将 ResultSet 的记录指针定位到最后一行，如果移动后的记录指针指向一条有效记录，则该方法返回 true。
- void afterLast()：将 ResultSet 的记录指针定位到最后一行之后。

注意：

在 JDK 1.4 以前，采用默认方法创建的 Statement 所查询得到的 ResultSet 不支持 absolute()、previous()等移动记录指针的方法，它只支持 next()这个移动记录指针的方法，即 ResultSet 的记录指针只能向下移动，而且每次只能移动一格。从 Java 5.0 以后就避免了这个问题，程序采用默认方法创建的 Statement 所查询得到的 ResultSet 也支持 absolute()、previous()等方法。

当把记录指针移动到指定行之后，ResultSet 可通过 getXxx(int columnIndex)或 getXxx(String columnLabel)方法来获取当前行、指定列的值，前者根据列索引获取值，后者根据列名获取值。Java 7 新增了<T> T getObject(int columnIndex, Class<T> type)和<T> T getObject(String columnLabel, Class<T> type)两个泛型方法，它们可以获取任意类型的值。

13.3.2 JDBC 编程步骤

大致了解了 JDBC API 的相关接口和类之后，下面就可以进行 JDBC 编程了，JDBC 编程大致按如下步骤进行。

① 加载数据库驱动。通常使用 Class 类的 forName()静态方法来加载驱动。例如如下代码：

```
// 加载驱动
Class.forName(driverClass)
```

上面代码中的 driverClass 就是数据库驱动类所对应的字符串。例如，加载 MySQL 的驱动采用如下代码：

```
// 加载 MySQL 的驱动
Class.forName("com.mysql.jdbc.Driver");
```

而加载 Oracle 的驱动则采用如下代码：

```
// 加载 Oracle 的驱动
Class.forName("oracle.jdbc.driver.OracleDriver");
```

从上面代码中可以看出，加载驱动时并不是真正使用数据库的驱动类，只是使用数据库驱动类名的字符串而已。

② 通过 DriverManager 获取数据库连接。DriverManager 提供了如下方法：

```
// 获取数据库连接
DriverManager.getConnection(String url,String user,String pass);
```

当使用 DriverManager 获取数据库连接时，通常需要传入三个参数：数据库 URL、登录数据库的用户名和密码。这三个参数中用户名和密码通常由 DBA（数据库管理员）分配，而且该用户还应该具有相应的权限，才可以执行相应的 SQL 语句。

数据库 URL 通常遵循如下写法：

```
jdbc:subprotocol:other stuff
```

上面 URL 写法中的 jdbc 是固定的，而 subprotocol 指定连接到特定数据库的驱动，而后面的 other 和 stuff 也是不固定的——也没有较强的规律，不同数据库的 URL 写法可能存在较大差异。例如，MySQL 数据库的 URL 写法如下：

```
jdbc:mysql://hostname:port/databasename
```

Oracle 数据库的 URL 写法如下：

```
jdbc:oracle:thin:@hostname:port:databasename
```

提示：

如果想了解特定数据库的 URL 写法，请查阅该数据库 JDBC 驱动的文档。

③ 通过 Connection 对象创建 Statement 对象。Connection 创建 Statement 的方法有如下三个。

- createStatement()：创建基本的 Statement 对象。
- prepareStatement(String sql)：根据传入的 SQL 语句创建预编译的 Statement 对象。
- prepareCall(String sql)：根据传入的 SQL 语句创建 CallableStatement 对象。

④ 使用 Statement 执行 SQL 语句。所有的 Statement 都有如下三个方法来执行 SQL 语句。

- execute()：可以执行任何 SQL 语句，但比较麻烦。
- executeUpdate()：主要用于执行 DML 和 DDL 语句。执行 DML 语句返回受 SQL 语句影响的行数，执行 DDL 语句返回 0。
- executeQuery()：只能执行查询语句，执行后返回代表查询结果的 ResultSet 对象。

⑤ 操作结果集。如果执行的 SQL 语句是查询语句，则执行结果将返回一个 ResultSet 对象，该对象里保存了 SQL 语句查询的结果。程序可以通过操作该 ResultSet 对象来取出查询结果。ResultSet 对象主要提供了如下两类方法。

- next()、previous()、first()、last()、beforeFirst()、afterLast()、absolute()等移动记录指针的方法。
- getXxx()方法获取记录指针指向行、特定列的值。该方法既可使用列索引作为参数，也可使用列名作为参数。使用列索引作为参数性能更好，使用列名作为参数可读性更好。

ResultSet 实质是一个查询结果集，在逻辑结构上非常类似于一个表。图 13.17 显示了 ResultSet 的逻辑结构，以及操作 ResultSet 结果集并获取值的方法示意图。

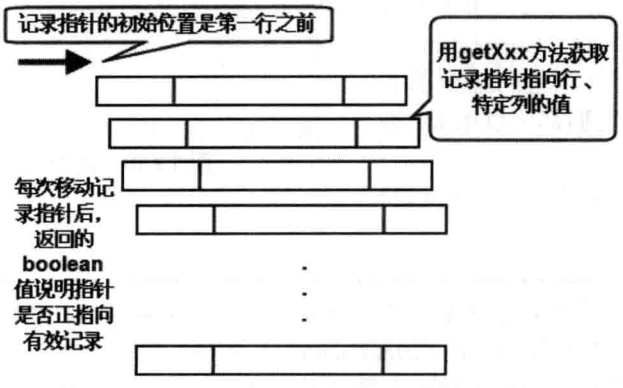


图 13.17　ResultSet 结果集示意图

⑥ 回收数据库资源，包括关闭 ResultSet、Statement 和 Connection 等资源。

下面程序简单示范了 JDBC 编程，并通过 ResultSet 获得结果集的过程。

程序清单：codes\13\13.3\ConnMySql.java

```
public class ConnMySql
{
    public static void main(String[] args) throws Exception
    {
        // 1.加载驱动，使用反射知识，现在记住这么写
        Class.forName("com.mysql.jdbc.Driver");
        try(
            // 2.使用 DriverManager 获取数据库连接
            // 其中返回的 Connection 就代表了 Java 程序和数据库的连接
            // 不同数据库的 URL 写法需要查驱动文档，用户名、密码由 DBA 分配
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/select_test"
                , "root" , "32147");
            // 3.使用 Connection 来创建一个 Statement 对象
            Statement stmt = conn.createStatement();
            // 4.执行 SQL 语句
            /*
            Statement 有三种执行 SQL 语句的方法：
            1. execute()可执行任何 SQL 语句—返回一个 boolean 值
              如果执行后第一个结果是 ResultSet，则返回 true，否则返回 false
            2. executeQuery()执行 select 语句 — 返回查询到的结果集
            3. executeUpdate()用于执行 DML 语句— 返回一个整数
              代表被 SQL 语句影响的记录条数
            */
            ResultSet rs = stmt.executeQuery("select s.* , teacher_name"
                + " from student_table s , teacher_table t"
                + " where t.teacher_id = s.java_teacher"))
        {
            // ResultSet 有一系列的 getXxx(列索引 | 列名)方法，用于获取记录指针
            // 指向行、特定列的值，不断地使用 next()将记录指针下移一行
            // 如果移动之后记录指针依然指向有效行，则 next()方法返回 true
            while(rs.next())
            {
                System.out.println(rs.getInt(1) + "\t"
                    + rs.getString(2) + "\t"
                    + rs.getString(3) + "\t"
                    + rs.getString(4));
            }
        }
    }
}
```

上面程序严格按 JDBC 访问数据库的步骤执行了一条多表连接查询语句，这条连接查询语句就是前面介绍 SQL 92 连接时所讲的连接查询语句。

与前面介绍的步骤略有区别的是，本程序采用了自动关闭资源的 try 语句来关闭各种数据库资源，Java 7 改写了 Connection、Statement、ResultSet 等接口，它们都继承了 AutoCloseable 接口，因此它们都可以由 try 语句来关闭。

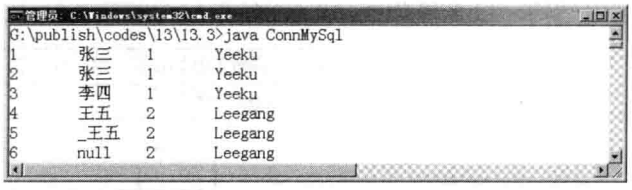


图 13.18　使用 JDBC 执行查询的结果

运行上面程序，会看到如图 13.18 所示的结果。

提示：

上面的运行结果也是基于前面所使用的 select_test 数据库，所以运行该程序之前应该先导入 codes\13\13.2 路径下的 select_data.sql 文件。除此之外，运行本程序时需要使用 MySQL 数据库驱动，该驱动 JAR 文件就是 codes\13\mysql-connector-java-5.1.30-bin.jar 文件，读者应该把该文件添加到系统的 CLASSPATH 环境变量里，或者直接使用 codes\13\13.3\路径下的 runConnMySql.cmd 来运行该程序，本章所有的程序都会提供这样一个对应的 cmd 批处理文件。

13.4 执行 SQL 语句的方式

前面介绍了 JDBC 执行查询等示例程序，实际上，JDBC 不仅可以执行查询，也可以执行 DDL、DML 等 SQL 语句，从而允许通过 JDBC 最大限度地控制数据库。

13.4.1 使用 Java 8 新增的 executeLargeUpdate 方法执行 DDL 和 DML 语句

Statement 提供了三个方法来执行 SQL 语句，前面已经介绍了使用 executeQuery()来执行查询语句，下面将介绍使用 executeLargeUpdate()（或 executeUpdate()）来执行 DDL 和 DML 语句。使用 Statement 执行 DDL 和 DML 语句的步骤与执行普通查询语句的步骤基本相似，区别在于执行了 DDL 语句后返回值为 0，执行了 DML 语句后返回值为受影响的记录条数。

下面程序示范了使用 executeUpdate()方法（此处暂未使用 executeLargeUpdate()方法是因为 MySQL 驱动暂不支持）创建数据表。该示例并没有直接把数据库连接信息写在程序里，而是使用一个 mysql.ini 文件（就是一个 properties 文件）来保存数据库连接信息，这是比较成熟的做法——当需要把应用程序从开发环境移植到生产环境时，无须修改源代码，只需要修改 mysql.ini 配置文件即可。

程序清单：codes\13\13.4\ExecuteDDL.java

```
public class ExecuteDDL
{
    private String driver;
    private String url;
    private String user;
    private String pass;
    public void initParam(String paramFile)
        throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
    }
    public void createTable(String sql)throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        try(
        // 获取数据库连接
        Connection conn = DriverManager.getConnection(url , user , pass);
        // 使用 Connection 来创建一个 Statement 对象
        Statement stmt = conn.createStatement())
        {
            // 执行 DDL 语句，创建数据表
            stmt.executeUpdate(sql);
        }
    }
    public static void main(String[] args) throws Exception
    {
        ExecuteDDL ed = new ExecuteDDL();
        ed.initParam("mysql.ini");
        ed.createTable("create table jdbc_test "
            + "( jdbc_id int auto_increment primary key, "
            + "jdbc_name varchar(255), "
            + "jdbc_desc text);");
        System.out.println("-----建表成功-----");
    }
}
```

运行上面程序，执行成功后会看到 select_test 数据库中添加了一个 jdbc_test 数据表，这表明 JDBC 执行 DDL 语句成功。

使用 executeUpdate()执行 DML 语句与执行 DDL 语句基本相似，区别是 executeUpdate()执行 DDL 语句后返回 0，而执行 DML 语句后返回受影响的记录条数。下面程序将会执行一条 insert 语句，这条 insert 语句会向刚刚建立的 jdbc_test 数据表中插入几条记录。因为使用了带子查询的 insert 语句，所以可以一次插入多条语句。

程序清单：codes\13\13.4\ExecuteDML.java

```
public class ExecuteDML
{
    private String driver;
    private String url;
    private String user;
    private String pass;
    public void initParam(String paramFile)
        throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
    }
    public int insertData(String sql)throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        try(
            // 获取数据库连接
            Connection conn = DriverManager.getConnection(url
                , user , pass);
            // 使用 Connection 来创建一个 Statement 对象
            Statement stmt = conn.createStatement())
        {
            // 执行 DML 语句，返回受影响的记录条数
            return stmt.executeUpdate(sql);
        }
    }
    public static void main(String[] args)throws Exception
    {
        ExecuteDML ed = new ExecuteDML();
        ed.initParam("mysql.ini");
        int result = ed.insertData("insert into jdbc_test(jdbc_name,jdbc_desc)"
            + "select s.student_name , t.teacher_name "
            + "from student_table s , teacher_table t "
            + "where s.java_teacher = t.teacher_id;");
        System.out.println("--系统中共有" + result + "条记录受影响--");
    }
}
```

运行上面程序，执行成功将会看到 jdbc_test 数据表中多了几条记录，而且在程序控制台会看到输出有几条记录受影响的信息。

13.4.2 使用 execute 方法执行 SQL 语句

Statement 的 execute()方法几乎可以执行任何 SQL 语句，但它执行 SQL 语句时比较麻烦，通常没有必要使用 execute()方法来执行 SQL 语句，使用 executeQuery()或 executeUpdate()方法更简单。但如果不清楚 SQL 语句的类型，则只能使用 execute()方法来执行该 SQL 语句了。

使用 execute()方法执行 SQL 语句的返回值只是 boolean 值，它表明执行该 SQL 语句是否返回了 ResultSet 对象。那么如何来获取执行 SQL 语句后得到的 ResultSet 对象呢？Statement 提供了如下两个方法来获取执行结果。

- getResultSet()：获取该 Statement 执行查询语句所返回的 ResultSet 对象。
- getUpdateCount()：获取该 Statement()执行 DML 语句所影响的记录行数。

下面程序示范了使用 Statement 的 execute()方法来执行任意的 SQL 语句，执行不同的 SQL 语句时产生不同的输出。

程序清单：codes\13\13.4\ExecuteSQL.java

```
public class ExecuteSQL
{
    private String driver;
    private String url;
    private String user;
    private String pass;
    public void initParam(String paramFile)throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
    }
    public void executeSql(String sql)throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        try(
            // 获取数据库连接
            Connection conn = DriverManager.getConnection(url
                , user , pass);
            // 使用 Connection 来创建一个 Statement 对象
            Statement stmt = conn.createStatement())
        {
            // 执行 SQL 语句，返回 boolean 值表示是否包含 ResultSet
            boolean hasResultSet = stmt.execute(sql);
            //如果执行后有 ResultSet 结果集
            if (hasResultSet)
            {
                try(
                    // 获取结果集
                    ResultSet rs = stmt.getResultSet())
                {
                    // ResultSetMetaData 是用于分析结果集的元数据接口
                    ResultSetMetaData rsmd = rs.getMetaData();
                    int columnCount = rsmd.getColumnCount();
                    // 迭代输出 ResultSet 对象
                    while (rs.next())
                    {
                        // 依次输出每列的值
                        for (int i = 0 ; i < columnCount ; i++ )
                        {
                            System.out.print(rs.getString(i + 1) + "\t");
                        }
                        System.out.print("\n");
                    }
                }
            }
            else
            {
                System.out.println("该 SQL 语句影响的记录有"
                    + stmt.getUpdateCount() + "条");
            }
        }
    }
    public static void main(String[] args) throws Exception
    {
        ExecuteSQL es = new ExecuteSQL();
        es.initParam("mysql.ini");
        System.out.println("------执行删除表的 DDL 语句-----");
```

```
            es.executeSql("drop table if exists my_test");
            System.out.println("------执行建表的 DDL 语句-----");
            es.executeSql("create table my_test"
                + "(test_id int auto_increment primary key, "
                + "test_name varchar(255))");
            System.out.println("------执行插入数据的 DML 语句-----");
            es.executeSql("insert into my_test(test_name) "
                + "select student_name from student_table");
            System.out.println("------执行查询数据的查询语句-----");
            es.executeSql("select * from my_test");
        }
    }
```

运行上面程序，会看到使用 Statement 的不同方法执行不同 SQL 语句的效果。执行 DDL 语句显示受影响的记录条数为 0；执行 DML 语句显示插入、修改或删除的记录条数；执行查询语句则可以输出查询结果。

提示：

上面程序获得 SQL 执行结果时没有根据各列的数据类型调用相应的 getXxx()方法，而是直接使用 getString()方法来取得值，这是可以的。ResultSet 的 getString()方法几乎可以获取除 Blob 之外的任意类型列的值，因为所有的数据类型都可以自动转换成字符串类型。

13.4.3 使用 PreparedStatement 执行 SQL 语句

如果经常需要反复执行一条结构相似的 SQL 语句，例如如下两条 SQL 语句：

```
insert into student_table values(null,'张三',1);
insert into student_table values(null,'李四',2);
```

对于这两条 SQL 语句而言，它们的结构基本相似，只是执行插入时插入的值不同而已。对于这种情况，可以使用带占位符（?）参数的 SQL 语句来代替它：

```
insert into student_table values(null,?,?);
```

但 Statement 执行 SQL 语句时不允许使用问号占位符参数，而且这个问号占位符参数必须获得值后才可以执行。为了满足这种功能，JDBC 提供了 PreparedStatement 接口，它是 Statement 接口的子接口，它可以预编译 SQL 语句，预编译后的 SQL 语句被存储在 PreparedStatement 对象中，然后可以使用该对象多次高效地执行该语句。简而言之，使用 PreparedStatement 比使用 Statement 的效率要高。

创建 PreparedStatement 对象使用 Connection 的 prepareStatement()方法，该方法需要传入一个 SQL 字符串，该 SQL 字符串可以包含占位符参数。如下代码所示：

```
// 创建一个 PreparedStatement 对象
pstmt = conn.prepareStatement("insert into student_table values(null,?,1)");
```

PreparedStatement 也提供了 execute()、executeUpdate()、executeQuery()三个方法来执行 SQL 语句，不过这三个方法无须参数，因为 PreparedStatement 已存储了预编译的 SQL 语句。

使用 PreparedStatement 预编译 SQL 语句时，该 SQL 语句可以带占位符参数，因此在执行 SQL 语句之前必须为这些参数传入参数值，PreparedStatement 提供了一系列的 setXxx(int index , Xxx value)方法来传入参数值。

提示：

如果程序很清楚 PreparedStatement 预编译 SQL 语句中各参数的类型，则使用相应的 setXxx()方法来传入参数即可；如果程序不清楚预编译 SQL 语句中各参数的类型，则可以使用 setObject()方法来传入参数，由 PreparedStatement 来负责类型转换。

下面程序示范了使用 Statement 和 PreparedStatement 分别插入 100 条记录的对比。使用 Statement 需要传入 100 条 SQL 语句，但使用 PreparedStatement 则只需要传入 1 条预编译的 SQL 语句，然后 100 次为该 PreparedStatement 的参数设值即可。

程序清单：codes\13\13.4\PreparedStatementTest.java

```
public class PreparedStatementTest
{
    private String driver;
    private String url;
    private String user;
    private String pass;
    public void initParam(String paramFile)throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
        // 加载驱动
        Class.forName(driver);
    }
    public void insertUseStatement()throws Exception
    {
        long start = System.currentTimeMillis();
        try(
            // 获取数据库连接
            Connection conn = DriverManager.getConnection(url
                , user , pass);
            // 使用 Connection 来创建一个 Statement 对象
            Statement stmt = conn.createStatement())
        {
            // 需要使用 100 条 SQL 语句来插入 100 条记录
            for (int i = 0; i < 100 ; i++ )
            {
                stmt.executeUpdate("insert into student_table values("
                    + " null ,'姓名" + i + "' , 1)");
            }
            System.out.println("使用 Statement 费时:"
                + (System.currentTimeMillis() - start));
        }
    }
    public void insertUsePrepare()throws Exception
    {
        long start = System.currentTimeMillis();
        try(
            // 获取数据库连接
            Connection conn = DriverManager.getConnection(url
                , user , pass);
            // 使用 Connection 来创建一个 PreparedStatement 对象
            PreparedStatement pstmt = conn.prepareStatement(
                "insert into student_table values(null,?,1)"))
        {
            // 100 次为 PreparedStatement 的参数设值，就可以插入 100 条记录
            for (int i = 0; i < 100 ; i++ )
            {
                pstmt.setString(1 , "姓名" + i);
                pstmt.executeUpdate();
            }
            System.out.println("使用 PreparedStatement 费时:"
                + (System.currentTimeMillis() - start));
        }
    }
    public static void main(String[] args) throws Exception
    {
        PreparedStatementTest pt = new PreparedStatementTest();
        pt.initParam("mysql.ini");
        pt.insertUseStatement();
        pt.insertUsePrepare();
    }
}
```

多次运行上面程序，可以发现使用 PreparedStatement 插入 100 条记录所用的时间比使用 Statement 插入 100 条记录所用的时间少，这表明 PreparedStatement 的执行效率比 Statement 的执行效率高。

除此之外，使用 PreparedStatement 还有一个优势——当 SQL 语句中要使用参数时，无须“拼接”SQL 字符串。而使用 Statement 则要“拼接”SQL 字符串，如上程序中粗体字代码所示，这是相当容易出现错误的——注意粗体字代码中的单引号，这是因为 SQL 语句中的字符串必须用单引号引起来。尤其是当 SQL 语句中有多个字符串参数时，“拼接”这条 SQL 语句时就更容易出错了。使用 PreparedStatement 则只需要使用问号占位符来代替这些参数即可，降低了编程复杂度。

使用 PreparedStatement 还有一个很好的作用——用于防止 SQL 注入。

提示：

SQL 注入是一个较常见的 Cracker 入侵方式，它利用 SQL 语句的漏洞来入侵。

下面以一个简单的登录窗口为例来介绍这种 SQL 注入的结果。下面登录窗口包含两个文本框，一个用于输入用户名，一个用于输入密码，系统根据用户输入与 jdbc_test 表里的记录进行匹配，如果找到相应记录则提示登录成功。

程序清单：codes\13\13.4\LoginFrame.java

```
public class LoginFrame
{
    private final String PROP_FILE = "mysql.ini";
    private String driver;
    // url 是数据库的服务地址
    private String url;
    private String user;
    private String pass;
    // 登录界面的 GUI 组件
    private JFrame jf = new JFrame("登录");
    private JTextField userField = new JTextField(20);
    private JTextField passField = new JTextField(20);
    private JButton loginButton = new JButton("登录");
    public void init()throws Exception
    {
        Properties connProp = new Properties();
        connProp.load(new FileInputStream(PROP_FILE));
        driver = connProp.getProperty("driver");
        url = connProp.getProperty("url");
        user = connProp.getProperty("user");
        pass = connProp.getProperty("pass");
        // 加载驱动
        Class.forName(driver);
        // 为登录按钮添加事件监听器
        loginButton.addActionListener(e -> {
            // 登录成功则显示“登录成功”
            if (validate(userField.getText(), passField.getText()))
            {
                JOptionPane.showMessageDialog(jf, "登录成功");
            }
            // 否则显示“登录失败”
            else
            {
                JOptionPane.showMessageDialog(jf, "登录失败");
            }
        });
        jf.add(userField , BorderLayout.NORTH);
        jf.add(passField);
        jf.add(loginButton , BorderLayout.SOUTH);
        jf.pack();
        jf.setVisible(true);
    }
    private boolean validate(String userName, String userPass)
    {
        // 执行查询的 SQL 语句
```

```
        String sql = "select * from jdbc_test "
            + "where jdbc_name='" + userName
            + "' and jdbc_desc='" + userPass + "'";
        System.out.println(sql);
        try(
            Connection conn = DriverManager.getConnection(url , user ,pass);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql))
        {
            // 如果查询的 ResultSet 里有超过一条的记录，则登录成功
            if (rs.next())
            {
                return true;
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return false;
    }
    public static void main(String[] args) throws Exception
    {
        new LoginFrame().init();
    }
}
```

运行上面程序，如果用户正常输入其用户名、密码当然没有问题，输入正确可以正常登录，输入错误将提示输入失败。但如果这个用户是一个 Cracker，他可以按图 13.19 所示来输入。

图 13.19 利用 SQL 注入

图 13.19 所示的输入明显不正确，但当单击“登录”按钮后也会显示“登录成功”对话框。可以在程序运行的后台看到如下 SQL 语句：

```
# 利用 SQL 注入后生成的 SQL 语句
select * from jdbc_test where jdbc_name='' or true or '' and jdbc_desc=''
```

看到这条 SQL 语句，读者应该不难明白为什么这样输入也可以显示“正常登录”对话框了，因为 Cracker 直接输入了 true，而 SQL 把这个 true 当成了直接量。

提示：

JDBC 编程本身并没有提供图形界面功能，它仅仅提供了数据库访问支持。如果希望 JDBC 程序有较好的图形用户界面，则需要结合前面介绍的 AWT 或 Swing 编程才可以做到。在 Web 编程中，数据库访问也是非常重要的基础知识。

如果把上面的 validate()方法换成使用 PreparedStatement 来执行验证，而不是直接使用 Statement。程序如下：

```
private boolean validate(String userName, String userPass)
{
    try(
        Connection conn = DriverManager.getConnection(url
            , user ,pass);
        PreparedStatement pstmt = conn.prepareStatement(
            "select * from jdbc_test where jdbc_name=? and jdbc_desc=?"))
    {
        pstmt.setString(1, userName);
        pstmt.setString(2, userPass);
        try(
            ResultSet rs = pstmt.executeQuery())
        {
            // 如果查询的 ResultSet 里有超过一条的记录，则登录成功
            if (rs.next())
            {
                return true;
            }
```

```
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return false;
    }
```

将上面的 validate()方法改为使用 PrepareStatement 来执行 SQL 语句之后，即使用户按图 13.19 所示输入，系统一样会显示“登录失败”对话框。

总体来看，使用 PreparedStatement 比使用 Statement 多了如下三个好处。

- PreparedStatement 预编译 SQL 语句，性能更好。
- PreparedStatement 无须“拼接”SQL 语句，编程更简单。
- PreparedStatement 可以防止 SQL 注入，安全性更好。

基于以上三点，通常推荐避免使用 Statement 来执行 SQL 语句，改为使用 PreparedStatement 执行 SQL 语句。

注意：

使用 PreparedStatement 执行带占位符参数的 SQL 语句时，SQL 语句中的占位符参数只能代替普通值，不要使用占位符参数代替表名、列名等数据库对象，更不要用占位符参数来代替 SQL 语句中的 insert、select 等关键字。

13.4.4 使用 CallableStatement 调用存储过程

下面的 SQL 语句可以在 MySQL 数据库中创建一个简单的存储过程。

```
delimiter //
create procedure add_pro(a int , b int, out sum int)
begin
set sum = a + b;
end;
//
```

上面的 SQL 语句将 MySQL 的语句结束符改为双斜线（//），这样就可以在创建存储过程中使用分号作为分隔符（MySQL 默认使用分号作为语句结束符）。上面程序创建了名为 add_pro 的存储过程，该存储过程包含三个参数：a、b 是传入参数，而 sum 使用 out 修饰，是传出参数。

提示：

关于存储过程的介绍请读者自行查阅相关书籍，本书仅介绍如何使用 JDBC 调用存储过程，并不会介绍创建存储过程的知识。

调用存储过程使用 CallableStatement，可以通过 Connection 的 prepareCall()方法来创建 CallableStatement 对象，创建该对象时需要传入调用存储过程的 SQL 语句。调用存储过程的 SQL 语句总是这种格式：{call 过程名(?,?,?...)}，其中的问号作为存储过程参数的占位符。例如，如下代码就创建了调用上面存储过程的 CallableStatement 对象。

```
// 使用 Connection 来创建一个 CallableStatement 对象
cstmt = conn.prepareCall("{call add_pro(?,?,?)}");
```

存储过程的参数既有传入参数，也有传出参数。所谓传入参数就是 Java 程序必须为这些参数传入值，可以通过 CallableStatement 的 setXxx()方法为传入参数设置值；所谓传出参数就是 Java 程序可以通过该参数获取存储过程里的值，CallableStatement 需要调用 registerOutParameter()方法来注册该参数。如下代码所示：

```
// 注册 CallableStatement 的第三个参数是 int 类型
cstmt.registerOutParameter(3, Types.INTEGER);
```

经过上面步骤之后，就可以调用 CallableStatement 的 execute()方法来执行存储过程了，执行结束后

通过 CallableStatement 对象的 getXxx(int index)方法来获取指定传出参数的值。下面程序示范了如何来调用该存储过程。

程序清单：codes\13\13.4\CallableStatementTest.java

```
public class CallableStatementTest
{
    private String driver;
    private String url;
    private String user;
    private String pass;
    public void initParam(String paramFile)throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
    }
    public void callProcedure()throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        try(
            // 获取数据库连接
            Connection conn = DriverManager.getConnection(url
                , user , pass);
            // 使用 Connection 来创建一个 CallableStatement 对象
            CallableStatement cstmt = conn.prepareCall(
                "{call add_pro(?,?,?)}"))
        {
            cstmt.setInt(1, 4);
            cstmt.setInt(2, 5);
            // 注册 CallableStatement 的第三个参数是 int 类型
            cstmt.registerOutParameter(3, Types.INTEGER);
            // 执行存储过程
            cstmt.execute();
            // 获取并输出存储过程传出参数的值
            System.out.println("执行结果是: " + cstmt.getInt(3));
        }
    }
    public static void main(String[] args) throws Exception
    {
        CallableStatementTest ct = new CallableStatementTest();
        ct.initParam("mysql.ini");
        ct.callProcedure();
    }
}
```

上面程序中的粗体字代码就是执行存储过程的关键代码，运行上面程序将会看到这个简单存储过程的执行结果，传入参数分别是 4、5，执行加法后传出总和 9。

13.5 管理结果集

JDBC 使用 ResultSet 来封装执行查询得到的查询结果，然后通过移动 ResultSet 的记录指针来取出结果集的内容。除此之外，JDBC 还允许通过 ResultSet 来更新记录，并提供了 ResultSetMetaData 来获得 ResultSet 对象的相关信息。

13.5.1 可滚动、可更新的结果集

前面提到，ResultSet 定位记录指针的方法有 absolute()、previous()等方法，但前面程序自始至终都只用了 next()方法来移动记录指针，实际上也可以使用 absolute()、previous()、last()等方法来移动记录

指针。可以使用 absolute()、previous()、afterLast()等方法自由移动记录指针的 ResultSet 被称为可滚动的结果集。

提示：

在 JDK 1.4 以前，默认打开的 ResultSet 是不可滚动的，必须在创建 Statement 或 PreparedStatement 时传入额外的参数。从 Java 5.0 以后，默认打开的 ResultSet 就是可滚动的，无须传入额外的参数。

以默认方式打开的 ResultSet 是不可更新的，如果希望创建可更新的 ResultSet，则必须在创建 Statement 或 PreparedStatement 时传入额外的参数。Connection 在创建 Statement 或 PreparedStatement 时还可额外传入如下两个参数。

- resultSetType：控制 ResultSet 的类型，该参数可以取如下三个值。
  - ResultSet.TYPE_FORWARD_ONLY：该常量控制记录指针只能向前移动。这是 JDK 1.4 以前的默认值。
  - ResultSet.TYPE_SCROLL_INSENSITIVE：该常量控制记录指针可以自由移动（可滚动结果集），但底层数据的改变不会影响 ResultSet 的内容。
  - ResultSet.TYPE_SCROLL_SENSITIVE：该常量控制记录指针可以自由移动（可滚动结果集），而且底层数据的改变会影响 ResultSet 的内容。

注意：

TYPE_SCROLL_INSENSITIVE、TYPE_SCROLL_SENSITIVE 两个常量的作用需要底层数据库驱动的支持，对于有些数据库驱动来说，这两个常量并没有太大的区别。

- resultSetConcurrency：控制 ResultSet 的并发类型，该参数可以接收如下两个值。
  - ResultSet.CONCUR_READ_ONLY：该常量指示 ResultSet 是只读的并发模式（默认）。
  - ResultSet.CONCUR_UPDATABLE：该常量指示 ResultSet 是可更新的并发模式。

下面代码通过这两个参数创建了一个 PreparedStatement 对象，由该对象生成的 ResultSet 对象将是可滚动、可更新的结果集。

```
// 使用 Connection 创建一个 PreparedStatement 对象
// 传入控制结果集可滚动、可更新的参数
pstmt = conn.prepareStatement(sql , ResultSet.TYPE_SCROLL_INSENSITIVE
    , ResultSet.CONCUR_UPDATABLE);
```

需要指出的是，可更新的结果集还需要满足如下两个条件。

- 所有数据都应该来自一个表。
- 选出的数据集必须包含主键列。

通过该 PreparedStatement 创建的 ResultSet 就是可滚动、可更新的，程序可调用 ResultSet 的 updateXxx(int columnIndex , Xxx value)方法来修改记录指针所指记录、特定列的值，最后调用 ResultSet 的 updateRow()方法来提交修改。

Java 8 为 ResultSet 添加了 updateObject(String columnLabel, Object x, SQLType targetSqlType)和 updateObject(int columnIndex, Object x, SQLType targetSqlType)两个默认方法，这两个方法可以直接用 Object 来修改记录指针所指记录、特定列的值，其中 SQLType 用于指定该数据列的类型。但目前最新的 MySQL 驱动暂不支持该方法。

下面程序示范了这种创建可滚动、可更新的结果集的方法。

程序清单：codes\13\13.5\ResultSetTest.java

```
public class ResultSetTest
{
    private String driver;
    private String url;
    private String user;
    private String pass;
```

```
    public void initParam(String paramFile)throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
    }
    public void query(String sql)throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        try(
            // 获取数据库连接
            Connection conn = DriverManager.getConnection(url, user , pass);
            // 使用 Connection 来创建一个 PreparedStatement 对象
            // 传入控制结果集可滚动、可更新的参数
            PreparedStatement pstmt = conn.prepareStatement(sql
                , ResultSet.TYPE_SCROLL_INSENSITIVE
                , ResultSet.CONCUR_UPDATABLE);
            ResultSet rs = pstmt.executeQuery())
        {
            rs.last();
            int rowCount = rs.getRow();
            for (int i = rowCount; i > 0 ; i-- )
            {
                rs.absolute(i);
                System.out.println(rs.getString(1) + "\t"
                    + rs.getString(2) + "\t" + rs.getString(3));
                // 修改记录指针所指记录、第 2 列的值
                rs.updateString(2 , "学生名" + i);
                // 提交修改
                rs.updateRow();
            }
        }
    }
    public static void main(String[] args) throws Exception
    {
        ResultSetTest rt = new ResultSetTest();
        rt.initParam("mysql.ini");
        rt.query("select * from student_table");
    }
}
```

上面程序中的粗体字代码示范了如何自由移动记录指针并更新记录指针所指的记录。运行上面程序，将会看到 student_table 表中的记录被倒过来输出了，因为是从最大记录行开始输出的。而且当程序运行结束后，student_table 表中所有记录的 student_name 列的值都被修改了。

注意：

如果要创建可更新的结果集，则使用查询语句查询的数据通常只能来自于一个数据表，而且查询结果集中的数据列必须包含主键列，否则将会引起更新失败。

13.5.2 处理 Blob 类型数据

Blob（Binary Long Object）是二进制长对象的意思，Blob 列通常用于存储大文件，典型的 Blob 内容是一张图片或一个声音文件，由于它们的特殊性，必须使用特殊的方式来存储。使用 Blob 列可以把图片、声音等文件的二进制数据保存在数据库里，并可以从数据库里恢复指定文件。

如果需要将图片插入数据库，显然不能直接通过普通的 SQL 语句来完成，因为有一个关键的问题——Blob 常量无法表示。所以将 Blob 数据插入数据库需要使用 PreparedStatement，该对象有一个方法：setBinaryStream(int parameterIndex, InputStream x)，该方法可以为指定参数传入二进制输入流，从而可以实现将 Blob 数据保存到数据库的功能。

当需要从 ResultSet 里取出 Blob 数据时，可以调用 ResultSet 的 getBlob(int columnIndex)方法，该方法将返回一个 Blob 对象，Blob 对象提供了 getBinaryStream()方法来获取该 Blob 数据的输入流，也可以使用 Blob 对象提供的 getBytes()方法直接取出该 Blob 对象封装的二进制数据。

为了把图片放入数据库，本程序先使用如下 SQL 语句来建立一个数据表。

```
create table img_table
(
    img_id int auto_increment primary key,
    img_name varchar(255),
    # 创建一个 mediumblob 类型的数据列，用于保存图片数据
    img_data mediumblob
);
```

提示：

上面 SQL 语句中的 img_data 列使用 mediumblob 类型，而不是 blob 类型。因为 MySQL 数据库里的 blob 类型最多只能存储 64KB 内容，这可能不够满足实际用途。所以使用 mediumblob 类型，该类型的数据列可以存储 16MB 内容。

下面程序可以实现图片“上传”——实际上就是将图片保存到数据库，并在右边的列表框中显示图片的名字，当用户双击列表框中的图片名时，左边窗口将显示该图片——实质就是根据选中的 ID 从数据库里查找图片，并将其显示出来。

程序清单：codes\13\13.5\BlobTest.java

```
public class BlobTest
{
    JFrame jf = new JFrame("图片管理程序");
    private static Connection conn;
    private static PreparedStatement insert;
    private static PreparedStatement query;
    private static PreparedStatement queryAll;
    // 定义一个 DefaultListModel 对象
    private DefaultListModel<ImageHolder> imageModel
        = new DefaultListModel<>();
    private JList<ImageHolder> imageList = new JList<>(imageModel);
    private JTextField filePath = new JTextField(26);
    private JButton browserBn = new JButton("...");
    private JButton uploadBn = new JButton("上传");
    private JLabel imageLabel = new JLabel();
    // 以当前路径创建文件选择器
    JFileChooser chooser = new JFileChooser(".");
    // 创建文件过滤器
    ExtensionFileFilter filter = new ExtensionFileFilter();
    static
    {
        try
        {
            Properties props = new Properties();
            props.load(new FileInputStream("mysql.ini"));
            String driver = props.getProperty("driver");
            String url = props.getProperty("url");
            String user = props.getProperty("user");
            String pass = props.getProperty("pass");
            Class.forName(driver);
            // 获取数据库连接
            conn = DriverManager.getConnection(url , user , pass);
            // 创建执行插入的 PreparedStatement 对象
            // 该对象执行插入后可以返回自动生成的主键
            insert = conn.prepareStatement("insert into img_table"
                + " values(null,?,?)" , Statement.RETURN_GENERATED_KEYS);
            // 创建两个 PreparedStatement 对象，用于查询指定图片，查询所有图片
            query = conn.prepareStatement("select img_data from img_table"
                + " where img_id=?");
            queryAll = conn.prepareStatement("select img_id, "
                + " img_name from img_table");
```

```
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    public void init()throws SQLException
    {
        // -------初始化文件选择器--------
        filter.addExtension("jpg");
        filter.addExtension("jpeg");
        filter.addExtension("gif");
        filter.addExtension("png");
        filter.setDescription("图片文件(*.jpg,*.jpeg,*.gif,*.png)");
        chooser.addChoosableFileFilter(filter);
        // 禁止“文件类型”下拉列表中显示“所有文件”选项
        chooser.setAcceptAllFileFilterUsed(false);
        // ---------初始化程序界面---------
        fillListModel();
        filePath.setEditable(false);
        // 只能单选
        imageList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JPanel jp = new JPanel();
        jp.add(filePath);
        jp.add(browserBn);
        browserBn.addActionListener(event -> {
            // 显示文件对话框
            int result = chooser.showDialog(jf , "浏览图片文件上传");
            // 如果用户选择了 APPROVE（赞同）按钮，即打开，保存等效按钮
            if(result == JFileChooser.APPROVE_OPTION)
            {
                filePath.setText(chooser.getSelectedFile().getPath());
            }
        });
        jp.add(uploadBn);
        uploadBn.addActionListener(avt -> {
            // 如果上传文件的文本框有内容
            if (filePath.getText().trim().length() > 0)
            {
                // 将指定文件保存到数据库
                upload(filePath.getText());
                // 清空文本框内容
                filePath.setText("");
            }
        });
        JPanel left = new JPanel();
        left.setLayout(new BorderLayout());
        left.add(new JScrollPane(imageLabel) , BorderLayout.CENTER);
        left.add(jp , BorderLayout.SOUTH);
        jf.add(left);
        imageList.setFixedCellWidth(160);
        jf.add(new JScrollPane(imageList) , BorderLayout.EAST);
        imageList.addMouseListener(new MouseAdapter()
        {
            public void mouseClicked(MouseEvent e)
            {
                // 如果鼠标双击
                if (e.getClickCount() >= 2)
                {
                    // 取出选中的 List 项
                    ImageHolder cur = (ImageHolder)imageList.
                    getSelectedValue();
                    try
                    {
                        // 显示选中项对应的 Image
                        showImage(cur.getId());
                    }
                    catch (SQLException sqle)
                    {
```

```
                    sqle.printStackTrace();
                }
            }
        }
    });
    jf.setSize(620, 400);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setVisible(true);
}
// ----------查找 img_table 填充 ListModel----------
public void fillListModel()throws SQLException
{

    try(
        // 执行查询
        ResultSet rs = queryAll.executeQuery())
    {
        // 先清除所有元素
        imageModel.clear();
        // 把查询的全部记录添加到 ListModel 中
        while (rs.next())
        {
            imageModel.addElement(new ImageHolder(rs.getInt(1)
                ,rs.getString(2)));
        }
    }
}
// ---------将指定图片放入数据库---------
public void upload(String fileName)
{
    // 截取文件名
    String imageName = fileName.substring(fileName.lastIndexOf('\\')
        + 1 , fileName.lastIndexOf('.'));
    File f = new File(fileName);
    try(
        InputStream is = new FileInputStream(f))
    {
        // 设置图片名参数
        insert.setString(1, imageName);
        // 设置二进制流参数
        insert.setBinaryStream(2, is , (int)f.length());
        int affect = insert.executeUpdate();
        if (affect == 1)
        {
            // 重新更新 ListModel，将会让 JList 显示最新的图片列表
            fillListModel();
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
// ---------根据图片 ID 来显示图片----------
public void showImage(int id)throws SQLException
{
    // 设置参数
    query.setInt(1, id);
    try(
        // 执行查询
        ResultSet rs = query.executeQuery())
    {
        if (rs.next())
        {
            // 取出 Blob 列
            Blob imgBlob = rs.getBlob(1);
            // 取出 Blob 列里的数据
            ImageIcon icon=new ImageIcon(imgBlob.getBytes(1L
                ,(int)imgBlob.length()));
```

```
                imageLabel.setIcon(icon);
            }
        }
    }
    public static void main(String[] args)throws SQLException
    {
        new BlobTest().init();
    }
}
// 创建 FileFilter 的子类，用以实现文件过滤功能
class ExtensionFileFilter extends FileFilter
{
    private String description = "";
    private ArrayList<String> extensions = new ArrayList<>();
    // 自定义方法，用于添加文件扩展名
    public void addExtension(String extension)
    {
        if (!extension.startsWith("."))
        {
            extension = "." + extension;
            extensions.add(extension.toLowerCase());
        }
    }
    // 用于设置该文件过滤器的描述文本
    public void setDescription(String aDescription)
    {
        description = aDescription;
    }
    // 继承 FileFilter 类必须实现的抽象方法，返回该文件过滤器的描述文本
    public String getDescription()
    {
        return description;
    }
    // 继承 FileFilter 类必须实现的抽象方法，判断该文件过滤器是否接受该文件
    public boolean accept(File f)
    {
        // 如果该文件是路径，接受该文件
        if (f.isDirectory()) return true;
        // 将文件名转为小写（全部转为小写后比较，用于忽略文件名大小写）
        String name = f.getName().toLowerCase();
        // 遍历所有可接受的扩展名，如果扩展名相同，该文件就可接受
        for (String extension : extensions)
        {
            if (name.endsWith(extension))
            {
                return true;
            }
        }
        return false;
    }
}
// 创建一个 ImageHolder 类，用于封装图片名、图片 ID
class ImageHolder
{
    // 封装图片的 ID
    private int id;
    // 封装图片的名字
    private String name;
    public ImageHolder(){}
    public ImageHolder(int id , String name)
    {
        this.id = id;
        this.name = name;
    }
    // id 的 setter 和 getter 方法
    public void setId(int id)
    {
        this.id = id;
    }
```

```
    public int getId()
    {
        return this.id;
    }
    // name 的 setter 和 getter 方法
    public void setName(String name)
    {
        this.name = name;
    }
    public String getName()
    {
        return this.name;
    }
    // 重写 toString()方法，返回图片名
    public String toString()
    {
        return name;
    }
}
```

上面程序中的第一段粗体字代码用于控制将一个图片文件保存到数据库，第二段粗体字代码用于控制将数据库里的图片数据显示出来。运行上面程序，并上传一些图片，会看到如图 13.20 所示的界面。

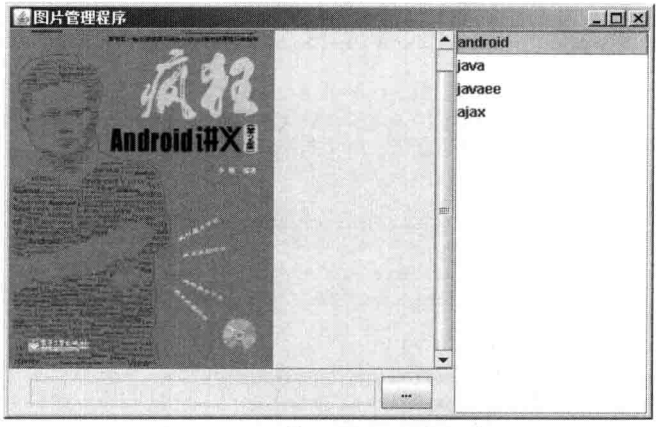


图 13.20　使用 Blob 保存图片

13.5.3　使用 ResultSetMetaData 分析结果集

当执行 SQL 查询后可以通过移动记录指针来遍历 ResultSet 的每条记录，但程序可能不清楚该 ResultSet 里包含哪些数据列，以及每个数据列的数据类型，那么可以通过 ResultSetMetaData 来获取关于 ResultSet 的描述信息。

提示：

MetaData 的意思是元数据，即描述其他数据的数据，因此 ResultSetMetaData 封装了描述 ResultSet 对象的数据；后面还要介绍的 DatabaseMetaData 则封装了描述 Database 的数据。

ResultSet 里包含一个 getMetaData()方法，该方法返回该 ResultSet 对应的 ResultSetMetaData 对象。一旦获得了 ResultSetMetaData 对象，就可通过 ResultSetMetaData 提供的大量方法来返回 ResultSet 的描述信息。常用的方法有如下三个。

- int getColumnCount()：返回该 ResultSet 的列数量。
- String getColumnName(int column)：返回指定索引的列名。
- int getColumnType(int column)：返回指定索引的列类型。

下面是一个简单的查询执行器，当用户在文本框内输入合法的查询语句并执行成功后，下面的表格将会显示查询结果。

程序清单：codes\13\13.5\QueryExecutor.java

```
public class QueryExecutor
{
    JFrame jf = new JFrame("查询执行器");
    private JScrollPane scrollPane;
    private JButton execBn = new JButton("查询");
    // 用于输入查询语句的文本框
    private JTextField sqlField = new JTextField(45);
    private static Connection conn;
    private static Statement stmt;
    // 采用静态初始化块来初始化 Connection、Statement 对象
    static
    {
```

```
        try
        {
            Properties props = new Properties();
            props.load(new FileInputStream("mysql.ini"));
            String drivers = props.getProperty("driver");
            String url = props.getProperty("url");
            String username = props.getProperty("user");
            String password = props.getProperty("pass");
            // 加载数据库驱动
            Class.forName(drivers);
            // 取得数据库连接
            conn = DriverManager.getConnection(url, username, password);
            stmt = conn.createStatement();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    // --------初始化界面的方法----------
    public void init()
    {
        JPanel top = new JPanel();
        top.add(new JLabel("输入查询语句："));
        top.add(sqlField);
        top.add(execBn);
        // 为执行按钮、单行文本框添加事件监听器
        execBn.addActionListener(new ExceListener());
        sqlField.addActionListener(new ExceListener());
        jf.add(top , BorderLayout.NORTH);
        jf.setSize(680, 480);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
    // 定义监听器
    class ExceListener implements ActionListener
    {
        public void actionPerformed(ActionEvent evt)
        {
            // 删除原来的 JTable(JTable 使用 scrollPane 来包装)
            if (scrollPane != null)
            {
                jf.remove(scrollPane);
            }
            try(
                // 根据用户输入的 SQL 执行查询
                ResultSet rs = stmt.executeQuery(sqlField.getText()))
            {
                // 取出 ResultSet 的 MetaData
                ResultSetMetaData rsmd = rs.getMetaData();
                Vector<String> columnNames =  new Vector<>();
                Vector<Vector<String>> data = new Vector<>();
                // 把 ResultSet 的所有列名添加到 Vector 里
                for (int i = 0 ; i < rsmd.getColumnCount(); i++ )
                {
                    columnNames.add(rsmd.getColumnName(i + 1));
                }
                // 把 ResultSet 的所有记录添加到 Vector 里
                while (rs.next())
                {
                    Vector<String> v = new Vector<>();
                    for (int i = 0 ; i < rsmd.getColumnCount(); i++ )
```

```
                {
                    v.add(rs.getString(i + 1));
                }
                data.add(v);
            }
            // 创建新的 JTable
            JTable table = new JTable(data , columnNames);
            scrollPane = new JScrollPane(table);
            // 添加新的 Table
            jf.add(scrollPane);
            // 更新主窗口
            jf.validate();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
public static void main(String[] args)
{
    new QueryExecutor().init();
}
}
```

上面程序中的粗体字代码就是根据 ResultSetMetaData 分析 ResultSet 的关键代码，使用 ResultSetMetaData 查询 ResultSet 包含多少列，并把所有数据列的列名添加到一个 Vector 里，然后把 ResultSet 里的所有数据添加到 Vector 里，并使用这两个 Vector 来创建新的 TableModel，再利用该 TableModel 生成一个新的 JTable，最后将该 JTable 显示出来。运行上面程序，会看到如图 13.21 所示的窗口。

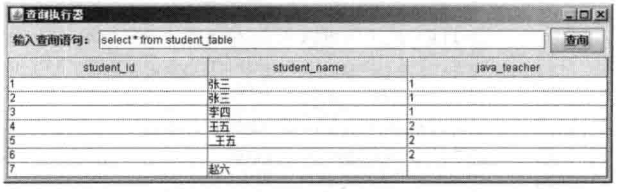


图 13.21　使用 ResultSetMetaData 分析 ResultSet

注意：

虽然 ResultSetMetaData 可以准确地分析出 ResultSet 里包含多少列，以及每列的列名、数据类型等，但使用 ResultSetMetaData 需要一定的系统开销，因此如果在编程过程中已经知道 ResultSet 里包含多少列，以及每列的列名、类型等信息，就没有必要使用 ResultSetMetaData 来分析该 ResultSet 对象了。

13.6　Java 7 的 RowSet 1.1

RowSet 接口继承了 ResultSet 接口，RowSet 接口下包含 JdbcRowSet、CachedRowSet、FilteredRowSet、JoinRowSet 和 WebRowSet 常用子接口。除了 JdbcRowSet 需要保持与数据库的连接之外，其余 4 个子接口都是离线的 RowSet，无须保持与数据库的连接。

与 ResultSet 相比，RowSet 默认是可滚动、可更新、可序列化的结果集，而且作为 JavaBean 使用，因此能方便地在网络上传输，用于同步两端的数据。对于离线 RowSet 而言，程序在创建 RowSet 时已把数据从底层数据库读取到了内存，因此可以充分利用计算机的内存，从而降低数据库服务器的负载，提高程序性能。

提示：

当年 C#提供了 DataSet，它可以把底层的数据读取到内存中进行离线操作，操作完成后再同步到底层数据源。Java 则提供了与此功能类似的 RowSet。

图 13.22 显示了 RowSet 规范的接口类图。

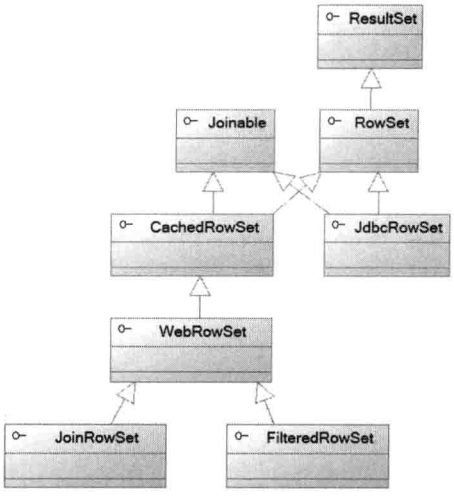


图 13.22 RowSet 规范的接口类图

在图 13.22 所示的各种接口中，CachedRowSet 及其子接口都代表了离线 RowSet，它们都不需要底层数据库连接。

13.6.1 Java 7 新增的 RowSetFactory 与 RowSet

在 Java 6.0 以前，RowSet 及其 5 个子接口都已经存在了，但在实际编程中的应用一直并不广泛，这是因为 Java 公开 API 并没有为 RowSet 及其各子接口提供实现类，而且也没有提供公开的方法来创建 RowSet 及其各子接口的实例。

实际上，Java 6.0 已经在 com.sun.rowset 包下提供了 JdbcRowSetImpl、CachedRowSetImpl、WebRowSetImpl、FilteredRowSetImpl 和 JoinRowSetImpl 五个实现类，它们代表了各种 RowSet 接口的实现类。

以 JdbcRowSet 为例，在 Java 6.0 及以前的版本中，如果程序需要使用 JdbcRowSet，则必须通过调用 JdbcRowSetImpl 的构造器来创建 JdbcRowSet 实例，JdbcRowSetImpl 提供了如下常用构造器。

- JdbcRowSetImpl()：创建一个默认的 JdbcRowSetImpl 对象。
- JdbcRowSetImpl(Connection conn)：以给定的 Connection 对象作为数据库连接来创建 JdbcRowSetImpl 对象。
- JdbcRowSetImpl(ResultSet rs)：创建一个包装 ResultSet 对象的 JdbcRowSetImpl 对象。

除此之外，RowSet 接口中定义了如下常用方法。

- setUrl(String url)：设置该 RowSet 要访问的数据库的 URL。
- setUsername(String name)：设置该 RowSet 要访问的数据库的用户名。
- setPassword(String password)：设置该 RowSet 要访问的数据库的密码。
- setCommand(String sql)：设置使用该 sql 语句的查询结果来装填该 RowSet。
- execute()：执行查询。
- populate(ResultSet rs)：让该 RowSet 直接包装给定的 ResultSet 对象。

通过 JdbcRowSet 的构造器和上面几个方法不难看出，为 JdbcRowSet 装填数据有如下两种方式。

- 创建 JdbcRowSetImpl 时，直接传入 ResultSet 对象。
- 使用 execute()方法来执行 SQL 查询，用查询返回的数据来装填 RowSet。

对于第二种方式来说，如果创建 JdbcRowSetImpl 时已经传入了 Connection 参数，则只要先调用 setCommand(String sql)指定 SQL 查询语句，接下来就可调用 execute()方法执行查询了。如果创建 JdbcRowSetImpl 时没有传入 Connection 参数，则先要为 JdbcRowSet 设置数据库的 URL、用户名、密码等连接信息。

下面程序通过 JdbcRowSetImpl 示范了使用 JdbcRowSet 的可滚动、可修改特性。

程序清单：codes\13\13.6\JdbcRowSetTest.java

```
public class JdbcRowSetTest
{
    private String driver;
    private String url;
    private String user;
    private String pass;
    public void initParam(String paramFile)throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
    }
    public void update(String sql)throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        try(
            // 获取数据库连接
            Connection conn = DriverManager.getConnection(url, user , pass);
            // 创建 JdbcRowSetImpl 对象
            JdbcRowSet jdbcRs = new JdbcRowSetImpl(conn))    // ①
        {
            // 设置 SQL 查询语句
            jdbcRs.setCommand(sql);
            // 执行查询
            jdbcRs.execute();
            jdbcRs.afterLast();
            // 向前滚动结果集
            while (jdbcRs.previous())
            {
                System.out.println(jdbcRs.getString(1)
                    + "\t" + jdbcRs.getString(2)
                    + "\t" + jdbcRs.getString(3));
                if (jdbcRs.getInt("student_id") == 3)
                {
                    // 修改指定记录行
                    jdbcRs.updateString("student_name", "孙悟空");
                    jdbcRs.updateRow();
                }
            }
        }
    }
    public static void main(String[] args)throws Exception
    {
        JdbcRowSetTest jt = new JdbcRowSetTest();
        jt.initParam("mysql.ini");
        jt.update("select * from student_table");
    }
}
```

上面程序中①号粗体字代码创建一个 JdbcRowSetImpl 实例，这就是一个 JdbcRowSet 对象。接下来的粗体字代码则示范了 JdbcRowSet 的可滚动、可修改的特性。

编译该程序，编译器将会在①号粗体字代码处发出警告：JdbcRowSetImpl 是内部专用 API，可能会在未来发行版中删除。这就是直接使用 JdbcRowSetImpl 的代价。

运行上面程序，一切正常。JdbcRowSet 是一个可滚动、可修改的结果集，因此底层数据表中相应的记录也被修改了。

需要说明的是，使用 JdbcRowSetImpl 除了编译器会发出警告之外，还有如下坏处。

- 程序直接与 JdbcRowSetImpl 实现类耦合，不利于后期的升级、扩展。
- JdbcRowSetImpl 实现类不是一个公开的 API，未来可能被删除。

正是因为上面两个原因，所以在 Java 6.0 时代，RowSet 并未得到广泛应用。Java 7 新增了 RowSetProvider 类和 RowSetFactory 接口，其中 RowSetProvider 负责创建 RowSetFactory，而 RowSetFactory 则提供了如下方法来创建 RowSet 实例。

- CachedRowSet createCachedRowSet()：创建一个默认的 CachedRowSet。
- FilteredRowSet createFilteredRowSet()：创建一个默认的 FilteredRowSet。
- JdbcRowSet createJdbcRowSet()：创建一个默认的 JdbcRowSet。
- JoinRowSet createJoinRowSet()：创建一个默认的 JoinRowSet。
- WebRowSet createWebRowSet()：创建一个默认的 WebRowSet。

通过使用 RowSetFactory，就可以把应用程序与 RowSet 实现类分离开，避免直接使用 JdbcRow SetImpl 等非公开的 API，也更有利于后期的升级、扩展。

下面程序使用 RowSetFactory 来创建 JdbcRowSet 实例。

程序清单：codes\13\13.6\RowSetFactoryTest.java

```
public class RowSetFactoryTest
{
    private String driver;
    private String url;
    private String user;
    private String pass;
    public void initParam(String paramFile)throws Exception
    {
        // 该方法的实现代码与前一个程序相同
    }
    public void update(String sql)throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        // 使用 RowSetProvider 创建 RowSetFactory
        RowSetFactory factory = RowSetProvider.newFactory();
        try(
            // 使用 RowSetFactory 创建默认的 JdbcRowSet 实例
            JdbcRowSet jdbcRs = factory.createJdbcRowSet())
        {
            // 设置必要的连接信息
            jdbcRs.setUrl(url);
            jdbcRs.setUsername(user);
            jdbcRs.setPassword(pass);
            // 设置 SQL 查询语句
            jdbcRs.setCommand(sql);
            // 执行查询
            jdbcRs.execute();
            // 其他部分与前一个程序完全相同
            ...
        }
    }
}
```

上面程序中的粗体字代码使用了 RowSetFactory 来创建 JdbcRowSet 对象，这就避免了与 JdbcRowSetImpl 实现类耦合。由于通过这种方式创建的 JdbcRowSet 还没有传入 Connection 参数，因此程序还需调用 setUrl()、setUsername()、setPassword()等方法来设置数据库连接信息。

编译、运行该程序，编译器不会发出任何警告，程序运行结果也一切正常。

13.6.2 离线 RowSet

在使用 ResultSet 的时代，程序查询得到 ResultSet 之后必须立即读取或处理它对应的记录，否则一旦 Connection 关闭，再去通过 ResultSet 读取记录就会引发异常。在这种模式下，JDBC 编程十分痛苦——假设应用程序架构被分为两层：数据访问层和视图显示层，当应用程序在数据访问层查询得到 ResultSet 之后，对 ResultSet 的处理有如下两种常见方式。

- 使用迭代访问 ResultSet 里的记录，并将这些记录转换成 Java Bean，再将多个 Java Bean 封装成一个 List 集合，也就是完成“ResultSet→Java Bean 集合”的转换。转换完成后可以关闭 Connection 等资源，然后将 Java Bean 集合传到视图显示层，视图显示层可以显示查询得到的数据。
- 直接将 ResultSet 传到视图显示层——这要求当视图显示层显示数据时，底层 Connection 必须一直处于打开状态，否则 ResultSet 无法读取记录。

第一种方式比较安全，但编程十分烦琐；第二种方式则需要 Connection 一直处于打开状态，这不仅不安全，而且对程序性能也有较大的影响。

通过使用离线 RowSet 可以十分“优雅”地处理上面的问题，离线 RowSet 会直接将底层数据读入内存中，封装成 RowSet 对象，而 RowSet 对象则完全可以当成 Java Bean 来使用。因此不仅安全，而且编程十分简单。CachedRowSet 是所有离线 RowSet 的父接口，因此下面以 CachedRowSet 为例进行介绍。看下面程序。

程序清单：codes\13\13.6\CachedRowSetTest.java

```
public class CachedRowSetTest
{
    private static String driver;
    private static String url;
    private static String user;
    private static String pass;
    public void initParam(String paramFile)throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
    }
    public CachedRowSet query(String sql)throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        // 获取数据库连接
        Connection conn = DriverManager.getConnection(url , user , pass);
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);
        // 使用 RowSetProvider 创建 RowSetFactory
        RowSetFactory factory = RowSetProvider.newFactory();
        // 创建默认的 CachedRowSet 实例
        CachedRowSet cachedRs = factory.createCachedRowSet();
        // 使用 ResultSet 装填 RowSet
        cachedRs.populate(rs);    // ①
        // 关闭资源
        rs.close();
        stmt.close();
        conn.close();
        return cachedRs;
    }
    public static void main(String[] args)throws Exception
    {
        CachedRowSetTest ct = new CachedRowSetTest();
        ct.initParam("mysql.ini");
        CachedRowSet rs = ct.query("select * from student_table");
        rs.afterLast();
        // 向前滚动结果集
        while (rs.previous())
        {
            System.out.println(rs.getString(1)
                + "\t" + rs.getString(2)
                + "\t" + rs.getString(3));
            if (rs.getInt("student_id") == 3)
            {
```

```
                // 修改指定记录行
                rs.updateString("student_name", "孙悟空");
                rs.updateRow();
            }
        }
        // 重新获取数据库连接
        Connection conn = DriverManager.getConnection(url
            , user , pass);
        conn.setAutoCommit(false);
        // 把对 RowSet 所做的修改同步到底层数据库
        rs.acceptChanges(conn);
    }
}
```

上面程序中的①号粗体字代码调用了 RowSet 的 populate(ResultSet rs)方法来包装给定的 ResultSet，接下来的粗体字代码关闭了 ResultSet、Statement、Connection 等数据库资源。如果程序直接返回 ResultSet，那么这个 ResultSet 无法使用——因为底层的 Connection 已经关闭；但程序返回的是 CachedRowSet，它是一个离线 RowSet，因此程序依然可以读取、修改 RowSet 中的记录。

运行该程序，可以看到在 Connection 关闭的情况下，程序依然可以读取、修改 RowSet 里的记录。为了将程序对离线 RowSet 所做的修改同步到底层数据库，程序在调用 RowSet 的 acceptChanges()方法时必须传入 Connection。

提示：

上面程序没有使用自动关闭资源的 try 语句来关闭 Connection 等数据库资源，这只是为了让读者更明确地看到 Connection 已被关闭。在实际项目中还是推荐使用自动关闭资源的 try 语句。

13.6.3 离线 RowSet 的查询分页

由于 CachedRowSet 会将数据记录直接装载到内存中，因此如果 SQL 查询返回的记录过大，CachedRowSet 将会占用大量的内存，在某些极端的情况下，它甚至会直接导致内存溢出。

为了解决该问题，CachedRowSet 提供了分页功能。所谓分页功能就是一次只装载 ResultSet 里的某几条记录，这样就可以避免 CachedRowSet 占用内存过大的问题。

CachedRowSet 提供了如下方法来控制分页。

- populate(ResultSet rs, int startRow)：使用给定的 ResultSet 装填 RowSet，从 ResultSet 的第 startRow 条记录开始装填。
- setPageSize(int pageSize)：设置 CachedRowSet 每次返回多少条记录。
- previousPage()：在底层 ResultSet 可用的情况下，让 CachedRowSet 读取上一页记录。
- nextPage()：在底层 ResultSet 可用的情况下，让 CachedRowSet 读取下一页记录。

下面程序示范了 CachedRowSet 的分页支持。

程序清单：codes\13\13.6\CachedRowSetPage.java

```
public class CachedRowSetPage
{
    private String driver;
    private String url;
    private String user;
    private String pass;
    public void initParam(String paramFile)throws Exception
    {
        // 使用 Properties 类来加载属性文件
        Properties props = new Properties();
        props.load(new FileInputStream(paramFile));
        driver = props.getProperty("driver");
        url = props.getProperty("url");
        user = props.getProperty("user");
        pass = props.getProperty("pass");
    }
    public CachedRowSet query(String sql , int pageSize
```

```
        , int page)throws Exception
    {
        // 加载驱动
        Class.forName(driver);
        try(
            // 获取数据库连接
            Connection conn = DriverManager.getConnection(url , user , pass);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql))
        {
            // 使用 RowSetProvider 创建 RowSetFactory
            RowSetFactory factory = RowSetProvider.newFactory();
            // 创建默认的 CachedRowSet 实例
            CachedRowSet cachedRs = factory.createCachedRowSet();
            // 设置每页显示 pageSize 条记录
            cachedRs.setPageSize(pageSize);
            // 使用 ResultSet 装填 RowSet，设置从第几条记录开始
            cachedRs.populate(rs , (page - 1) * pageSize + 1);
            return cachedRs;
        }
    }
    public static void main(String[] args)throws Exception
    {
        CachedRowSetPage cp = new CachedRowSetPage();
        cp.initParam("mysql.ini");
        CachedRowSet rs = cp.query("select * from student_table" , 3 , 2);  // ①
        // 向后滚动结果集
        while (rs.next())
        {
            System.out.println(rs.getString(1)
                + "\t" + rs.getString(2)
                + "\t" + rs.getString(3));
        }
    }
}
```

上面两行粗体字代码就是使用 CachedRowSet 实现分页的关键代码。程序中①号代码显示要查询第 2 页的记录，每页显示 3 条记录。运行上面程序，可以看到程序只会显示从第 4 行到第 6 行的记录，这就实现了分页。

13.7 事务处理

对于任何数据库应用而言，事务都是非常重要的，事务是保证底层数据完整的重要手段，没有事务支持的数据库应用，那将非常脆弱。

13.7.1 事务的概念和 MySQL 事务支持

事务是由一步或几步数据库操作序列组成的逻辑执行单元，这系列操作要么全部执行，要么全部放弃执行。程序和事务是两个不同的概念。一般而言，一段程序中可能包含多个事务。

事务具备 4 个特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持续性（Durability）。这 4 个特性也简称为 ACID 性。

- 原子性（Atomicity）：事务是应用中最小的执行单位，就如原子是自然界的最小颗粒，具有不可再分的特征一样，事务是应用中不可再分的最小逻辑执行体。
- 一致性（Consistency）：事务执行的结果，必须使数据库从一个一致性状态，变到另一个一致性状态。当数据库只包含事务成功提交的结果时，数据库处于一致性状态。如果系统运行发生中断，某个事务尚未完成而被迫中断，而该未完成的事务对数据库所做的修改已被写入数据库，此时，数据库就处于一种不正确的状态。比如银行在两个账户之间转账：从 A 账户向 B 账户转入 1000 元，系统先减少 A 账户的 1000 元，然后再为 B 账户增加 1000 元。如果全部执行成功，数据库处于一致性状态；如果仅执行完 A 账户金额的修改，而没有增加 B 账户的金额，则数据

库就处于不一致性状态；因此，一致性是通过原子性来保证的。

- 隔离性（Isolation）：各个事务的执行互不干扰，任意一个事务的内部操作对其他并发的事务都是隔离的。也就是说，并发执行的事务之间不能看到对方的中间状态，并发执行的事务之间不能互相影响。
- 持续性（Durability）：持续性也称为持久性（Persistence），指事务一旦提交，对数据所做的任何改变都要记录到永久存储器中，通常就是保存进物理数据库。

数据库的事务由下列语句组成。

- 一组 DML 语句，经过这组 DML 语句修改后的数据将保持较好的一致性。
- 一条 DDL 语句。
- 一条 DCL 语句。

DDL 和 DCL 语句最多只能有一条，因为 DDL 和 DCL 语句都会导致事务立即提交。

当事务所包含的全部数据库操作都成功执行后，应该提交（commit）事务，使这些修改永久生效。事务提交有两种方式：显式提交和自动提交。

- 显式提交：使用 commit。
- 自动提交：执行 DDL 或 DCL 语句，或者程序正常退出。

当事务所包含的任意一个数据库操作执行失败后，应该回滚（rollback）事务，使该事务中所做的修改全部失效。事务回滚有两种方式：显式回滚和自动回滚。

- 显式回滚：使用 rollback。
- 自动回滚：系统错误或者强行退出。

MySQL 默认关闭事务（即打开自动提交），在默认情况下，用户在 MySQL 控制台输入一条 DML 语句，这条 DML 语句将会立即保存到数据库里。为了开启 MySQL 的事务支持，可以显式调用如下命令：

```
SET AUTOCOMMIT = {0 | 1}  0 为关闭自动提交，即开启事务
```

提示：

自动提交和开启事务恰好相反，如果开启自动提交就是关闭事务；关闭自动提交就是开启事务。

一旦在 MySQL 的命令行窗口中输入 set autocommit=0 开启了事务，该命令行窗口里的所有 DML 语句都不会立即生效，上一个事务结束后第一条 DML 语句将开始一个新的事务，而后续执行的所有 SQL 语句都处于该事务中，除非显式使用 commit 来提交事务，或者正常退出，或者运行 DDL、DCL 语句导致事务隐式提交。当然，也可以使用 rollback 回滚来结束事务，使用 rollback 结束事务将导致本次事务中 DML 语句所做的修改全部失效。

提示：

一个 MySQL 命令行窗口代表一次连接 Session，在该窗口里设置 set autocommit=0，相当于关闭了该连接 Session 的自动提交，对其他连接不会有任何影响，也就是对其他 MySQL 命令行窗口不会有任何影响。

除此之外，如果不想关闭整个命令行窗口的自动提交，而只是想临时性地开始事务，则可以使用 MySQL 提供的 start transaction 或 begin 两个命令，它们都表示临时性地开始一次事务，处于 start transaction 或 begin 后的 DML 语句不会立即生效，除非使用 commit 显式提交事务，或者执行 DDL、DCL 语者来隐式提交事务。如下 SQL 代码将不会对数据库有任何影响。

```
# 临时开始事务
begin;
# 向 student_table 表中插入 3 条记录
insert into student_table
values(null , 'xx' , 1);
insert into student_table
values(null , 'yy' , 1);
```

```
insert into student_table
values(null , 'zz' , 1);
# 查询 student_table 表的记录
select * from student_table;      # ①
# 回滚事务
rollback;
# 再次查询
select * from student_table;      # ②
```

执行上面 SQL 语句中的第①条查询语句将会看到刚刚插入的 3 条记录，如果打开 MySQL 的其他命令行窗口将看不到这 3 条记录——这正体现了事务的隔离性。接着程序 rollback 了事务中的全部修改，执行第②条查询语句时将看到数据库又恢复到事务开始前的状态。

提交，不管是显式提交还是隐式提交，都会结束当前事务；回滚，不管是显式回滚还是隐式回滚，都会结束当前事务。

除此之外，MySQL 还提供了 savepoint 来设置事务的中间点，通过使用 savepoint 设置事务的中间点可以让事务回滚到指定中间点，而不是回滚全部事务。如下 SQL 语句设置了一个中间点：

```
savepoint a;
```

一旦设置了中间点后，就可以使用 rollback 回滚到指定中间点，回滚到指定中间点的代码如下：

```
rollback to a;
```

注意：

普通的提交、回滚都会结束当前事务，但回滚到指定中间点因为依然处于事务之中，所以不会结束当前事务。

13.7.2 JDBC 的事务支持

JDBC 连接也提供了事务支持，JDBC 连接的事务支持由 Connection 提供，Connection 默认打开自动提交，即关闭事务，在这种情况下，每条 SQL 语句一旦执行，便会立即提交到数据库，永久生效，无法对其进行回滚操作。

可以调用 Connection 的 setAutoCommit()方法来关闭自动提交，开启事务，如下代码所示：

```
// 关闭自动提交，开启事务
conn.setAutoCommit(false);
```

程序中还可调用 Connection 提供的 getAutoCommit()方法来返回该连接的自动提交模式。

一旦事务开始之后，程序可以像平常一样创建 Statement 对象，创建了 Statement 对象之后，可以执行任意多条 DML 语句，如下代码所示：

```
stmt.executeUpdate(...);
stmt.executeUpdate(...);
stmt.executeUpdate(...);
```

上面这些 SQL 语句虽然被执行了，但这些 SQL 语句所做的修改不会生效，因为事务还没有结束。如果所有的 SQL 语句都执行成功，程序可以调用 Connection 的 commit()方法来提交事务，如下代码所示：

```
// 提交事务
conn.commit();
```

如果任意一条 SQL 语句执行失败，则应该用 Connection 的 rollback()方法来回滚事务，如下代码所示：

```
// 回滚事务
conn.rollback();
```

提示：

实际上，当 Connection 遇到一个未处理的 SQLException 异常时，系统将会非正常退出，事务也会自动回滚。但如果程序捕获了该异常，则需要在异常处理块中显式地回滚事务。