

程序员书库

初学者的入门宝典，程序员的百科全书



CD-ROM

10小时多媒体视频讲解

#### 本书特色

- ※ 起点低，即使没有任何编程经验，也能通过本书掌握Java
- ※ 避免大段理论讲解，而是通过大量实例进行讲解，有很强的实践性
- ※ 对代码进行了详细注释，阅读起来非常容易，没有任何障碍
- ※ 通过现实中的事物类比Java中的概念，使读者可以很容易理解
- ※ 重点讲解Java语言的基础知识和应用，并对一些设计模式也有所介绍
- ※ 全书提供190个实例和2个综合案例，非常实用

# Java

## 从入门到精通

高宏静 等编著



化学工业出版社

## 第 13 章 Swing 用户界面设计

AWT 存在缺少剪贴板、打印支持等缺陷，甚至没有弹出式菜单和滚动窗口等，因此 Swing 的产生也就成为必然。Swing 是纯 Java 实现的轻量级（light-weight）组件，它不依赖系统的支持。本章主要讨论 Swing 组件基本的使用方法和使用 Swing 组件创建用户界面的初步方法。

### 13.1 Swing 基础

Swing 元素比 AWT 元素具有更好的屏幕显示性能。Swing 用 100% 纯 Java 实现，所以 Swing 具有 Java 的跨平台性。Swing 不是真正使用原生平台提供设备，而是仅仅在模仿，因此可以在任何平台上使用 Swing 图形用户界面组件。Swing 绝大部分组件都是轻量级组件，它不像重量级组件那样必须在它们自己本地窗口中绘制，而是在它们所在的重量级窗口中绘制。

注意：AWT 组件具有平台相关性，它是系统对等类的实现；而 Swing 组件在不同平台具有一致性的表现，另外还可以提供本地系统不支持的一些特征，因此 Swing 比 AWT 的组件实用性更强。Swing 采用了 MVC（Model-View-Controller，即模型-视图-控制）设计模式。

#### 13.1.1 Swing 的类层次结构

在 javax.swing 包中，有两种类型的组件：顶层容器（JFrame、JApplet、JDialog 和 JWindow）和轻量级组件。Swing 轻量级组件都是由 AWT 的 Container 类直接或间接派生而来。

```
java.awt.Component
+ java.awt.Container
+ java.awt.Window
+ java.awt.Frame-javax.swing.JFrame
+ javax.swing.JDialog
+ javax.swing.JWindow
+ java.awt.Applet-javax.swing.JApplet
+ javax.swing.Box
+ javax.swing.JComponent
```

Swing 包是 JFC（Java Foundation Classes）的一部分，它由许多包组成，如表 13-1 所示。

表 13-1 Swing 包组成内容

包	描述
Com.sun.swing.plaf.motif	实现 Motif 界面样式代表类
Com.sun.java.swing.plaf.windows	实现 Windows 界面样式的代表类
javax.swing	Swing 组件和使用工具

续表

包	描述
<code>javax.swing.border</code>	Swing 轻量组件的边框
<code>javax.swing.colorchooser</code>	<code>JColorChooser</code> 的支持类/接口
<code>javax.swing.event</code>	事件和侦听器类
<code>javax.swing.filechooser</code>	<code>JFileChooser</code> 的支持类/接口
<code>javax.swing.pending</code>	未完全实现的 Swing 组件
<code>javax.swing.plaf</code>	抽象类，定义 UI 代表的行为
<code>javax.swing.plaf.basic</code>	实现所有标准界面样式公共基类
<code>javax.swing.plaf.metal</code>	实现 <code>Metal</code> 界面样式代表类
<code>javax.swing.table</code>	<code>JTable</code> 组件
<code>javax.swing.text</code>	支持文档的显示和编辑
<code>javax.swing.text.html</code>	支持显示和编辑 <code>HTML</code> 文档
<code>javax.swing.text.html.parser</code>	<code>Html</code> 文档的分析器
<code>javax.swing.text.rtf</code>	支持显示和编辑 <code>RTF</code> 文件
<code>javax.swing.tree</code>	<code>JTree</code> 组件的支持类
<code>javax.swing.undo</code>	支持取消操作

`javax.swing` 包是 Swing 提供的最大包，它大约包含 100 个类和 25 个接口，并且绝大部分 Swing 组件都包含在 `swing` 包中（`JtableHeader`、`JtextComponent` 除外，分别在 `swing.table` 和 `swing.text` 包中）。`javax.swing.event` 包中定义了事件和事件处理类，这与 `java.awt.event` 包类似，主要包括事件类和监听器接口、事件适配器。

- ❑ `javax.swing.pending` 包主要是一些没有完全实现的组件。
- ❑ `javax.swing.table` 包主要是 `Jtable` 类的支持类。
- ❑ `javax.swing.tree` 包同样也是 `Jtree` 类的支持类。
- ❑ `javax.swing.text`、`swing.text.html`、`swing.text.html.parser` 和 `swing.text.rtf` 包都是与文档显示和编辑相关的包。

### 13.1.2 Swing 的特点

在正式学习 Swing 之前，先来看一下 Swing 的特点，这也是和 AWT 的不同之处。Swing 的特点包括以下几点。

（1）组件的多样化。Swing 的基础是 AWT，它提供了比 AWT 更多的图形界面组件。Swing 组件的类名都是以字母“J”开头，除 AWT 具有的基本组件外（按钮 `JButton`、标签 `JLabel`、复选框 `JCheckBox`、菜单 `JMenu`），还增加了许多比较复杂的高级组件，例如表格（`JTable`）、树（`JTree`）。

（2）MVC 模式。Swing 比较突出的特点在于 MVC 模型的普遍实用性。MVC 模型主要包括 3 部分结构：模型、视图和控制器。模型（`Model`）用于保存所用到的数据；视图（`View`）用于显示数据内容；控制器（`Controller`）用于处理用户和模块交互事件。当模型的数据改变，

会通知与之相关的视图，视图通过控制器指定相应机制。为了简化工作，Swing 组件将视图和控制器合二为一。MVC 设计思想是图形用户界面设计比较通用，这样可以使内容本身和显示方式分类，使数据显示更加灵活多变。

(3) 可存取性支持。为了实现可存取性的支持，所有 Swing 组件实现了 Accessible 接口，这样一些辅助功能（屏幕阅读器）也可以得到实现。

(4) 支持键盘操作。JComponent 类的 registerKeyboardAction() 方法可以实现用键盘操作代替用户鼠标操作，相当于传统意义的热键操作。

▼ (5) 设置边框。Swing 组件可以设置一个或者多个边框，并且提供多种类型的边框供用户选择，甚至用户可以建立组合边框或自己设计边框。另外空白边框可以使组件变大，协助布局管理器布局。

(6) 使用图标 (Icon)。Swing 组件另外一个更加丰富的功能就是许多组件可以通过图标修饰自己。

### 13.1.3 Swing 程序结构简介

Swing 的程序设计一般可按照以下流程进行。

- (1) 通过 import 引入 swing 包。
- (2) 设置 GUI 的“外观界面风格”。
- (3) 创建顶层容器。
- (4) 创建按钮和标签等组件。
- (5) 将组件添加到顶层容器。
- (6) 在组件周围添加边界。
- (7) 进行事件处理。

下面是第一个 Swing 程序。

// 文件：程序 13.1	SwingApplication.java	描述：SwingApplication 演示
// 导入需要使用的包和类		
import javax.swing.*; // 引入 Swing 包名		
import java.awt.*;		
import java.awt.event.*;		
public class SwingApplication {		
public static void main(String[] args) {		
try {		// try 语句块，监视该段程序
// 设置窗口风格		
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());		
} catch (Exception e) {		// 捕获异常
e.printStackTrace();		// 异常信息输出
}		
JFrame frame = new JFrame("Swing 应用程序");		// 创建，并初始化顶层容器并
Container c = frame.getContentPane();		// 获取面板容器
JPanel pane = new JPanel();		// 创建，并初始化面板 panel
c.add(pane);		// 将面板添加到窗口
pane.setLayout(new FlowLayout());		// 设置布局管理器 FlowLayout
final JLabel label = new JLabel();		// 创建，并初始化标签 label

```

        JButton button = new JButton("Click me!");           //创建, 并初始化 button
        pane.add(label);                                     //向容器中添加组件 label
        pane.add(button);                                   //向容器中添加组件 button
        //添加事件处理
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText("北京欢迎您! ");              //设置 label 内容
            }
        });
        //窗口设置结束, 开始显示
        frame.addWindowListener(new WindowAdapter() {
            //匿名类用于注册监听器
            public void windowClosing(WindowEvent e) {
                System.exit(0);                              //程序退出
            }
        });
        frame.setSize(300,240);                             //设置窗口大小
        frame.setVisible(true);                             //显示窗口
    }
}

```

编写完程序后, 使用 `javac` 命令编译该文件产生 `class` 文件, 然后使用 `java` 命令运行该 `class` 文件, 运行结果如图 13-1 和图 13-2 所示。

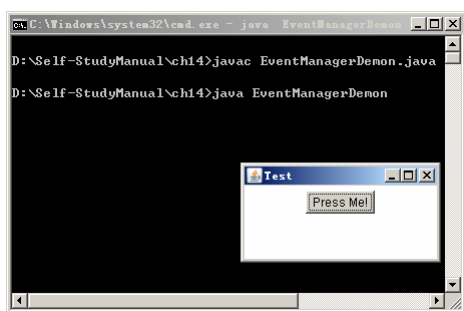


图 13-1 SwingApplication.java 运行结果一

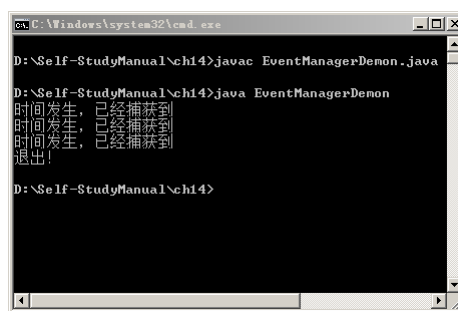


图 13-2 SwingApplication.java 运行结果二

## 13.2 Swing 组件分类和基本规则

Swing 不仅使用轻量级组件代替 AWT 的重量级组件, 而且还增加了许多丰富的功能。例如 Swing 的按钮和标签等组件可以图形化 (即可以使用图标), Swing 中的组件与 AWT 对应的组件名前面加了一个“J”。本节主要讲述组件的分类以及组件的基本使用规则。

### 13.2.1 组件的分类

Jcomponent 是一个抽象类, 主要用于定义所有子类的通用方法。层次关系如下。

```

java.lang.Object
+--java.awt.Component

```

```

+---java.awt.Container
    +---javax.swing.JComponent

```

JComponent 类派生于 Container 类。并不是 Swing 的所有组件都继承了 JComponent 类，凡是派生于 Container 类的组件都可以作为容器使用。Swing 组件从功能上可分为顶层容器、中间容器、特殊容器、基本控件、信息显示组件和编辑信息组件。

- ❑ 顶层容器：JFrame、JApplet、JDialog、JWindow。
- ❑ 中间容器：JPanel、JScrollPane、JSplitPane、JToolBar。
- ❑ 特殊容器：GUI 中特殊作用的中间层，例如 JInternalFrame、JLayeredPane、JRootPane。
- ❑ 基本控件：人机交互的基本组件，例如 JButton、JComboBox、JList、JMenu、JSlider、JTextField。
- ❑ 信息显示组件：组件仅仅为显示信息，但不能编辑，例如 JLabel、JProgressBar、ToolTip。
- ❑ 编辑信息组件：向用户显示可被编辑信息的组件，例如 JColorChooser、JFileChooser、JTable、JTextArea。

另外，JComponent 类的一些特殊功能包括边框设置、双缓冲区、提示信息、键盘导航和支持布局。

- ❑ 边框设置：使用 setBorder() 方法设置组件外围边框，如果不设置边框就会为组件的外围留出空白。
- ❑ 双缓冲区：为了改善组件的显示效果，采用双缓冲技术。JComponent 组件默认是双缓冲的，不必要自己写代码，可以通过 setDoubleBuffered(false) 关闭双缓冲区。
- ❑ 提示信息：setToolTipText() 方法可为组件设置提示信息，为用户提供帮助。
- ❑ 键盘导航：registerKeyboardAction() 方法可以实现键盘代替鼠标操作。
- ❑ 支持布局：用户可以设置组件最大、最小和设置对齐参数值等方法，指定布局管理器的约束条件。

### 13.2.2 使用 Swing 的基本规则

与 AWT 组件不同，Swing 不能直接在顶层容器中添加组件。Swing 组件必须添加到与顶层容器相关的内容面板上，内容面板是一个普通的轻量级组件，还要避免使用非 Swing 轻量级组件。在顶层容器 JFrame 对象中添加组件有以下两种方式。

(1) 用 getContentPane() 方法获得容器的内容面板，直接添加组件，格式如下。

```

Container c = frame.getContentPane(); //获取窗口内容面板
JPanel pane = new JPanel();           //创建面板
c.add(pane);                          //在容器中添加面板

```

(2) 建立一个中间容器对象 (JPanel 或 JDesktopPane)，将组件添加到中间容器对象内，然后通过 setContentPane() 方法将该容器设置为顶层容器 frame 的内容面板。

```

JPanel pane = new JPanel();           //创建面板对象
pane.add(new JButton("OK"));          //给面板添加按钮
frame.setContentPane(pane);           //将面板 pane 设置为窗口内容面板

```

## 13.3 轻量容器

JFrame 的内部轻量容器构成结构为：最底层是根面板 JRootPane，上一层是玻璃面板 glassPane（一个 JPanel 对象）和分层面板 layeredPane（一个 JLayeredPane 对象），而分层面板 layeredPane 又由内容面板 contentPane（一个 JPanel）和菜单条 menuBar（JMenuBar）构成。用户添加组件都是在内容面板上，背景图片只能添加到分层面板。玻璃面板是完全透明的，缺省值为不可见，覆盖了内容面板是为了接受鼠标事件和组件绘图功能提供方便。JRootPane 的结构如图 13-3 所示。

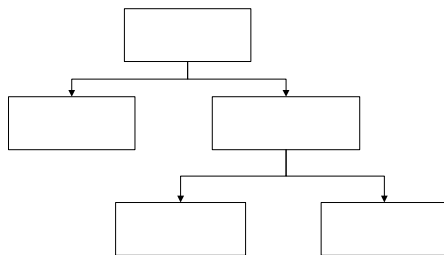


图 13-3 JRootPane 结构

### 13.3.1 根面板

根面板（JRootPane）是由内容面板、菜单条和玻璃面板组成，它可以获得和设置相关面板。

```

java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--javax.swing.JComponent
        +--javax.swing.JRootPane
  
```

JRootPane

根面板提供的方法有以下几种。

- ❑ Container getContentPane(): 获得内容面板。
- ❑ void setContentPane(Container): 设置内容面板。
- ❑ JMenuBar getMenuBar(): 获得活动菜单条。
- ❑ void setMenuBar(JMenuBar): 设置菜单条。
- ❑ JLayeredPane getLayeredPane(): 获得分层面板。
- ❑ void setLayeredPane(JLayeredPane): 设置分层面板。
- ❑ Component getGlassPane(): 获得玻璃面板。
- ❑ void setGlassPane(Component): 设置玻璃面板。

glassPane

contentPan

### 13.3.2 面板

面板（JPanel）是轻量容器，其用法与 Panel 基本相同，用于容纳其他组件。JScrollPane、JSplitPane 和 JInternalFrame 都属于常用的中间容器。面板的默认布局管理器是顺序布局管理器（FlowLayout）。

```
java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--javax.swing.JComponent
        +--javax.swing.JPanel
```

### 13.3.3 分层面板

Swing 有两种类型的分层面板：JLayeredPane 和 JDesktopPane。JDesktopPane 是 JLayeredPane 的派生类，用于创建多文档和虚拟桌面程序的容器。用户可以创建内部框架（JInternalFrame）对象并将其添加到 JDesktopPane。它们的继承关系如下。

```
java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--javax.swing.JComponent
        +--javax.swing.JLayeredPane
          +--javax.swing.JDesktopPane
```

### 13.3.4 滚动窗口

滚动窗口（JScrollPane）是管理视口（JViewport）、可选水平或垂直滚动条的轻量级容器。滚动窗口基本上是由 JScrollBar、一个 JViewport 之间的连接组成。Jviewport 是用于查看底层组件的一个工具，滚动条沿着组件的水平或者垂直方法移动视口，同时显示组件上的内容。JScrollPane 继承关系如下。

```
java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--javax.swing.JComponent
        +--javax.swing.JScrollPane
```

### 13.3.5 分隔板

分隔板（JSplitPane）用于提供可拆分的窗口，但只能分隔成两个容器。它可支持水平和垂直两种拆分类型，还可以带滚动条。分隔板的继承关系如下。

```
java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--javax.swing.JComponent
```



## +--javax.swing.JSplitPane

常用的方法有以下几种。

- ❑ `void addImpl(Component comp, Object constraints, int index)`: 将组件添加到次分隔窗口。
- ❑ `void setTopComponent(Component comp)`: 将组件设置到分隔条上面或者左边。
- ❑ `void setDividerLocation(double proportionalLocation)`: 设置分隔条的位置。
- ❑ `void setDividerSize(int newSize)`: 设置分隔条的大小。

### 13.3.6 选项板

选项板 (`JTabbedPane`) 提供允许用户通过单击具有标题和图标的选项卡, 在组件之间切换的组件。通过 `addTab` 和 `insertTab` 方法添加选项卡、组件到 `TabbedPane` 对象中。选项卡可以通过对应位置的索引表示, 第一个索引值为 0。

```
java.lang.Object
+--java.awt.Component
    +--java.awt.Container
        +--javax.swing.JComponent
            +--javax.swing.JTabbedPane
```

常用的方法有以下几种。

- ❑ `add(String title, Component component)`: 添加一个带标题的组件。
- ❑ `addChangeListener(ChangeListener l)`: 将一个 `ChangeListener` 注册到选项卡窗口中。
- ❑ `addTab(String title, Icon icon, Component component, String tip)`: 添加指定标题、图标的组件, 任意一个参数都可以为 `null`。
- ❑ `insertTab(String title, Icon icon, Component component, String tip, int index)`: 在索引 `index` 插入一个 `component`, 该组件指定标题、图标和提示。

### 13.3.7 工具栏

工具栏 (`JToolBar`) 是显示常用工具组件的容器, 可以将工具栏拖动出来成为一个独立的工具条。工具栏的继承关系如下。

```
java.lang.Object
+--java.awt.Component
    +--java.awt.Container
        +--javax.swing.JComponent
            +--javax.swing.JToolBar
```

常用的方法有以下几种。

- ❑ `JToolBar(String name)`: 构造方法。
- ❑ `addImpl(Component comp, Object constraints, int index)`: 在工具栏中添加新的 `JButton`。
- ❑ `getComponentIndex(Component c)`: 返回组件的序号。

- ❑ `getComponentAtIndex(int i)`: 返回序号为 `i` 的组件。

## 13.4 Swing 组件

Swing 的组件与 AWT 组件相似，但又为每一个组件增添了新的方法，并提供了更多的高级组件。所以本节 Swing 的基本组件选取了几个比较典型的组件进行详细讲解。

### 13.4.1 按钮

Swing 中，所有类型的按钮都是 `javax.swing.AbstractButton` 类的子类。用户使用 Swing 按钮可以显示图像，将整个按钮设置为窗口缺省图标，并且可将多个图像指定给一个按钮，来处理鼠标在按钮上的事件。JButton 类的继承关系如下。

```
java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--javax.swing.JComponent
        +--javax.swing.AbstractButton
          +--javax.swing.JButton
```

常用的构造方法有以下几种。

- ❑ `JButton(Icon icon)`: 按钮上显示图标。
- ❑ `JButton(String text)`: 按钮上显示字符。
- ❑ `JButton(String text, Icon icon)`: 按钮上既显示图标又显示字符。

下面是一个使用 JButton 的例子。

```
// 文件: 程序 13.2    ButtonIcon.java    描述: JButton 演示
//导入需要使用的包和类
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class ButtonIcon extends JPanel implements ActionListener{
    public static void main(String[] args) {
        JFrame jf = new JFrame();           //创建，并初始化窗口对象 jf
        ButtonIcon bi = new ButtonIcon();    //创建，并初始化图标对象 bi
        jf.getContentPane().add(bi);         //获取面板容器，并添加按钮
        jf.setSize(400, 130);               //设置窗口大小
        jf.setVisible(true);                //显示窗口
        jf.addWindowListener(new WindowAdapter() {
            //匿名类用于注册监听器
            public void windowClosing(WindowEvent e) {
                System.exit(0);}            //程序退出
        });
    }
}
//声明 ButtonIcon 类构造方法
public ButtonIcon() {
```

```

super();                                //调用父类构造方法
button = new JButton[3];                 //初始化 JButton 数组
imagemcon = new Imagemcon[3];            //初始化 Imagemcon 数组
//保存三个图标
imagemcon[0] = new Imagemcon("images\\first.gif"); //初始化图标 imagemcon[0]
imagemcon[1] = new Imagemcon("images\\second.gif"); //初始化图标 imagemcon[1]
imagemcon[2] = new Imagemcon("images\\third.gif"); //初始化图标 imagemcon[2]
button[0] = new JButton(imagemcon[0]);    //初始化 button[0]
button[1] = new JButton(imagemcon[1]);    //初始化 button[1]
button[2] = new JButton(imagemcon[2]);    //初始化 button[2]
add(button[0]);                          //添加按钮 button[0]
add(button[1]);                          //添加按钮 button[1]
add(button[2]);                          //添加按钮 button[2]
button[0].addActionListener(this);        //为按钮 button[0]添加监听器 ActionListener
button[1].addActionListener(this);        //为按钮 button[1]添加监听器 ActionListener
button[2].addActionListener(this);        //为按钮 button[2]添加监听器 ActionListener
}
public void actionPerformed(ActionEvent e) {
    Imagemcon imagemcon;                 //创建 Imagemcon 对象
    if( (JButton)e.getSource() == button[0]) //判断事件源是否是 button[0]
    {
        for(int i = 0; i < 3; i++)
            button[i].setIcon(imagemcon[(i+1)%3]); //为 button[i]设置图标
        //更换图标
        imagemcon = imagemcon[0];
        imagemcon[0] = imagemcon[1];
        imagemcon[1] = imagemcon[2];
        imagemcon[2] = imagemcon;
        imagemcon = null;                 //设置 imagemcon 为 null
    }else if((JButton)e.getSource() == button[2]) //判断事件源是否是 button[2]
    {
        for(int i = 2; i >= 0; i--)
            button[i].setIcon(imagemcon[(i-1+3)%3]); //为 button[i]设置图标
        //更换图标
        imagemcon = imagemcon[2];         //将 imagemcon[2]引用指向 imagemcon
        imagemcon[2] = imagemcon[1];       //将 imagemcon[1]引用指向 imagemcon[2]
        imagemcon[1] = imagemcon[0];       //将 imagemcon[0]引用指向 imagemcon[1]
        imagemcon[0] = imagemcon;          //将 imagemcon 引用指向 imagemcon[0]
        imagemcon = null;                 //设置 imagemcon 为 null
    }
}
JButton button[ ];                       //声明 JButton 类型数组 button
Imagemcon imagemcon[ ];                  //声明 Imagemcon 类型数组 imagemcon
}

```



图 13-4 ButtonIcon.java 运行结果

编写完程序后,使用 javac 命令编译该文件产生 class 文件,然后使用 java 命令运行该 class 文件,运行结果如图 13-4 所示。

程序 13.2 中,首先创建了 3 个按钮对象,并为第一个和第三个按钮添加监听器接口处理鼠标单击事件。当鼠标单击第一个按钮时,3 个图标左方向循环,当鼠标单击

击第三个按钮时，3 个图标右方向循环。

### 13.4.2 复选框

复选框（JCheckBox）可以满足在多个选项中选择一个或者多个选项，它比 AWT 的复选框具有更多的优点，例如可以显示图片。该类是 javax.swing.JToggleButton 的子类，其继承关系如下。

```
java.lang.Object
+--java.awt.Component
  +--java.awt.Container
    +--javax.swing.JComponent
      +--javax.swing.AbstractButton
        +--javax.swing.JToggleButton
          +--javax.swing.JCheckBox
```

复选框的构造方法有以下几种。

- ❑ JCheckBox(): 创建一个无文本、无图标、未被选定的复选框。
- ❑ JCheckBox(Action a): 创建一个复选框，属性由参数 Action 提供。
- ❑ JCheckBox(Icon icon): 创建一个有图标、但未被选定的复选框。
- ❑ JCheckBox(Icon icon, boolean selected): 创建一个有图标，并且指定是否被选定的复选框。
- ❑ JCheckBox(String text): 创建一个有文本，但未被选定的复选框。
- ❑ JCheckBox(String text, boolean selected): 创建一个有文本，并且指定是否被选定的复选框。
- ❑ JCheckBox(String text, Icon icon): 创建一个指定文本和图标，但未被选定的复选框。
- ❑ JCheckBox(String text, Icon icon, boolean selected): 创建一个指定文本和图标，并指定是否被选定的复选框。

下面是一个使用复选框的例子。

```
// 文件：程序 13.3    JCheckBoxDemo.java    描述：JCheckBox 演示
//导入需要使用的包和类
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class JCheckBoxDemo extends JPanel implements ActionListener{
    public static void main(String[ ] args) {
        JFrame jf = new JFrame();           //创建，并初始化 JFrame 对象 jf
        JCheckBoxDemo bi = new JCheckBoxDemo(); //创建，并初始化 JCheckBoxDemo 对象 bi
        jf.getContentPane().add(bi);         //获取窗口面板，并添加面板 bi
        jf.setSize(500, 230);               //设置窗口大小
        jf.setVisible(true);               //显示窗口
        jf.addWindowListener(new WindowAdapter() {
            //匿名类用于注册监听器
            public void windowClosing(WindowEvent e) {
```

```

        System.exit(0);}                                //程序退出

    });
    jf.pack();                                          //调整窗口大小，适应内部组件的大小和布局
}

//声明 JCheckBoxDemo 类的构造方法
public JCheckBoxDemo() {
    super();                                          //调用父类的构造方法
    this.setLayout(new GridLayout(2,1));           //设置窗口的布局管理器 GridLayout
    button = new JCheckBox[3];                       //初始化数组 button
    imagelcon = new Imgelcon[6];                   //初始化数组 imagelcon
    ftf = new JTextField (20);                     //初始化单行文本区 ftf
    panel = new JPanel();                          //初始化 panel
    add(fft);                                       //在窗口中添加组件 ftf
    //初始化图标数组 imagelcon
    imagelcon[0] = new Imgelcon("images\\Cat1.jpg"); //初始化数组元素 imagelcon[0]
    imagelcon[1] = new Imgelcon("images\\Rabbit1.jpg"); //初始化数组元素 imagelcon[1]
    imagelcon[2] = new Imgelcon("images\\Dog1.jpg"); //初始化数组元素 imagelcon[2]
    imagelcon[3] = new Imgelcon("images\\Cat.jpg"); //初始化数组元素 imagelcon[3]
    imagelcon[4] = new Imgelcon("images\\Rabbit.jpg"); //初始化数组元素 imagelcon[4]
    imagelcon[5] = new Imgelcon("images\\Dog.jpg"); //初始化数组元素 imagelcon[5]
    //初始化按钮数组
    button[0] = new JCheckBox("猫",imagelcon[0]);    //初始化数组元素 button[0]
    button[1] = new JCheckBox("兔",imagelcon[1]);    //初始化数组元素 button[1]
    button[2] = new JCheckBox("狗",imagelcon[2]);    //初始化数组元素 button[2]
    //将按钮数组的各个按钮添加在面板 panel 中
    panel.add(button[0]);                          //将按钮 button[0]添加到面板 panel
    panel.add(button[1]);                          //将按钮 button[1]添加到面板 panel
    panel.add(button[2]);                          //将按钮 button[2]添加到面板 panel
    add(panel);                                    //将面板添加在窗口中
    button[0].addActionListener(this);             //为按钮 button[0]添加 ActionListener 监听器
    button[1].addActionListener(this);             //为按钮 button[1]添加 ActionListener 监听器
    button[2].addActionListener(this);             //为按钮 button[2]添加 ActionListener 监听器
}

public void actionPerformed(ActionEvent e) {
    String select = "";                             //创建，并初始化字符串对象
    for(int i =0; i < button.length; i++)
    {
        if( button[i].isSelected())                //判断选择的对象
        {
            select = select + (i == 0?"猫":i==1?"兔":"狗"); //设置字符串的值 select
            button[i].setIcon(imagelcon[i+3]);        //设置按钮图标
        }
        else
        {
            button[i].setIcon(imagelcon[i]);          //设置按钮图标
        }
    }
    ftf.setText(select);                            //设置文本框文本内容
}

JCheckBox button[ ];                               //声明 JCheckBox 对象数组域
Imgelcon imagelcon[ ];                             //声明 Imgelcon 对象数组域
JTextField ftf;                                     //声明 JTextField 对象数组域
JPanel panel;                                       //声明 JPanel 对象数组域

```

}

编写完程序后,使用 `javac` 命令编译该文件产生 `class` 文件,然后使用 `java` 命令运行该 `class` 文件,运行结果如图 13-5 所示。

程序 13.3 中,主要展示了复选框可以显示图标的特性。该程序先创建,并初始化 `ImageIcon` 对象数组,然后通过图标创建并初始化 `JCheckBox` 对象数组,为多选框对象添加了 `ActionListener` 监听器,监听器方法 `actionPerformed` 根据是否选择该复选框来显示不同的图标效果。

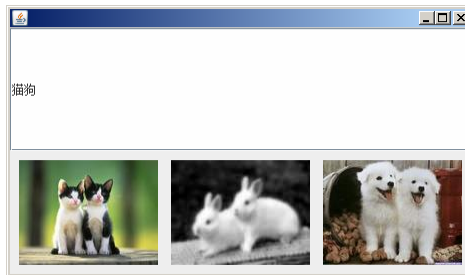


图 13-5 JCheckBoxDemo.java 运行结果

### 13.4.3 单选框

单选框 (`JRadioButton`) 与 AWT 中的复选框组功能类似,其继承关系如下。

```
java.lang.Object
+--java.awt.Component
  +--java.awt.Container
    +--javax.swing.JComponent
      +--javax.swing.AbstractButton
        +--javax.swing.JToggleButton
          +--javax.swing.JRadioButton
```

`JRadioButton` 与 `ButtonGroup` 配合使用时创建一组按钮,并且一次只能选择其中的一个按钮,通过 `add()` 方法将 `JRadioButton` 对象添加到该组中。

注意: 使用 `ButtonGroup` 对象进行分组是逻辑分组而不是物理分组。创建一组按钮通常需要创建一个 `JPanel` 或者类似容器,并将按钮添加到容器中。

`JRadioButton` 的构造方法有以下几种。

- ❑ `JRadioButton()`: 创建一个未指定图标和文本,并且未被选定的单选按钮。
- ❑ `JRadioButton(Action a)`: 创建一个属性来自 `Action` 的单选按钮。
- ❑ `JRadioButton(Icon icon)`: 创建一个指定图标但未指定文本,未被选定的单选按钮。
- ❑ `JRadioButton(Icon icon, boolean selected)`: 创建一个指定图像和状态,但无文本的单选按钮。
- ❑ `JRadioButton(String text)`: 创建一个指定文本,但无图标未被选择的单选按钮。
- ❑ `JRadioButton(String text, boolean selected)`: 创建一个执行文本和选择状态的单选按钮。
- ❑ `JRadioButton(String text, Icon icon)`: 创建一个指定文本和图标,但未被选择的单选按钮。
- ❑ `JRadioButton(String text, Icon icon, boolean selected)`: 创建一个具有指定的文本、图像和选择状态的单选按钮。

下面是一个使用 `JRadioButton` 的例子。

```
// 文件: 程序 13.4    JRadioButtonDemo.java    描述: JRadioButton 演示
//导入需要使用的包和类
import javax.swing.*;
import java.awt.*;
```

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class JRadioButtonDemo extends JPanel implements ActionListener{
    public static void main(String[ ] args) {
        JFrame jf = new JFrame();           //创建，并初始化 JFrame 对象 jf
        JRadioButtonDemo bi = new JRadioButtonDemo(); //创建并初始化 JRadioButtonDemo
对象 bi
        jf.getContentPane().add(bi);        //获取内容面板，并添加组件 bi
        jf.setSize(500, 230);               //设置窗口大小
        jf.setVisible(true);                //显示窗口
        jf.addWindowListener(new WindowAdapter() {
            //匿名类用于注册监听器
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);              //退出程序
            }
        });
        jf.pack();                           //调整窗口的大小
    }
//声明 JRadioButtonDemo 类构造方法
public JRadioButtonDemo() {
    super(); //调用父类构造方法
    this.setLayout(new GridLayout(2,1));    //设置窗口布局管理器 GridLayout
    button = new JRadioButton[2];           //初始化 JRadioButton 数组
    imagelcon = new Imgelcon[4];            //初始化 Imgelcon 数组
    ftf = new JTextField (20);              //初始化 JTextField 对象 ftf
    panel = new JPanel();                   //初始化 JPanel 对象 panel
    bg = new ButtonGroup();                 //初始化 ButtonGroup 对象 bg
    add(fft);                              //添加单行文本区到窗口
    //初始化图标数组 imagelcon
    imagelcon[0] = new Imgelcon("images\\male0.gif"); //初始化图标数组元素 imagelcon[0]
    imagelcon[1] = new Imgelcon("images\\female0.gif"); //初始化图标数组元素 imagelcon[1]
    imagelcon[2] = new Imgelcon("images\\male.gif"); //初始化图标数组元素 imagelcon[2]
    imagelcon[3] = new Imgelcon("images\\female.gif"); //初始化图标数组元素 imagelcon[3]
    //初始化单选按钮数组 button
    button[0] = new JRadioButton(imagelcon[0]); //初始化按钮数组元素 button[0]
    button[1] = new JRadioButton(imagelcon[1]); //初始化按钮数组元素 button[1]
    panel.add(button[0]);                    //将按钮 button[0]添加到面板 panel
    panel.add(button[1]);                   //将按钮 button[1]添加到面板 panel
    //使用 bg 创建一组单选按钮
    bg.add(button[0]);                      //在 bg 中添加 button[0]
    bg.add(button[1]);                      //在 bg 中添加 button[1]
    add(panel);                             //在窗口中添加面板 panel
    button[0].addActionListener(this);      //为 button[0]按钮添加监听器 ActionListener
    button[1].addActionListener(this);      //为 button[1]按钮添加监听器 ActionListener
}
public void actionPerformed(ActionEvent e)
{
    Imgelcon imagelconc;                   //创建 Imgelcon 对象
    String select = "";                    //创建字符串变量 select
    for(int i =0; i < button.length; i++)
    {

```

```

        if( button[i].isSelected())
        {
            select = select + (i == 0?"男":"女"); //为 select 字符串赋值
            button[i].setIcon(imagelcon[i+2]); //设置按钮图标
        }else
        {
            button[i].setIcon(imagelcon[i]); //设置按钮图标
        }
    }
    ftf.setText(select); //设置 ftf 文本内容
}
JRadioButton button[ ]; //声明 JRadioButton 类型数组
Imagelcon imagelcon[ ]; //声明 Imagelcon 类型数组
ButtonGroup bg; //声明 ButtonGroup 类型 bg
JTextField ftf; //声明 JTextField 类型 ftf
JPanel panel; //声明 JPanel 类型 panel
}

```

编写完程序后，使用 `javac` 命令编译该文件产生 `class` 文件，然后使用 `java` 命令运行该 `class` 文件，运行结果如图 13-6 所示。

程序 13.4 与程序 13.3 类似，都是首先创建图标对象数组，然后通过图标创建 `JRadioButton` 对象，并为每一个按钮添加 `ActionListener` 监听器实现 `actionPerformed` 方法。根据选择的内容，图标显示不同的效果。

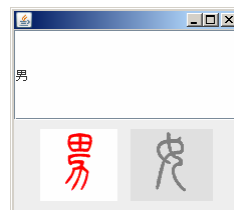


图 13-6 JRadioButtonDemo.java 运行结果

#### 13.4.4 组合框

组合框 (`JComboBox`) 是将按钮、可编辑字段以及下拉菜单组合的组件。用户可以从下拉列表中选择不同的值，如果组合框处于可编辑状态，用户可以键入值。组合框的构造方法有以下几种。

- ❑ `JComboBox()`: 创建一个没有数据选项的组合框。
- ❑ `JComboBox(ComboBoxModel aModel)`: 创建一个数据来源于 `ComboBoxModel` 的组合框。
- ❑ `JComboBox(Object[] items)`: 创建一个指定数组元素作为选项的组合框。
- ❑ `JComboBox(Vector<?> items)`: 创建一个指定 `Vector` 中元素的组合框。

#### 13.4.5 进程条

进程条 (`JProgressBar`) 是以图形化的方式来描述任务进度的组件。在任务完成过程中，进度条显示该任务完成的百分比，百分比通常用可视化的矩形表示，该矩形开始为空，随着任务的执行，组件被填满。进程条的继承关系如下。

```
java.lang.Object
```



```

+--java.awt.Component
  +--java.awt.Container
    +--javax.swing.JComponent
      +--javax.swing.JProgressBar

```

进程条的构建方法有以下几种。

- ❑ `JProgressBar()`: 创建一个显示矩形, 但不显示进度字符串的水平进度条。
  - ❑ `JProgressBar(BoundedRangeModel newModel)`: 创建一个指定进度条数据模型的水平进度条。
  - ❑ `JProgressBar(int orient)`: 创建一个指定方向 (`VERTICAL` 或 `HORIZONTAL`) 的进度条。
  - ❑ `JProgressBar(int min, int max)`: 创建一个指定最小值和最大值的水平进度条。
  - ❑ `JProgressBar(int orient, int min, int max)`: 创建一个指定方向、最小值和最大值的进度条。
- 下面是一个使用 `JProgressBar` 的例子。

```

// 文件: 程序 13.5      JProgressBarDemo.java      描述: JProgressBar 演示
//导入需要使用的包和类
import javax.swing.*;
public class JProgressBarDemo extends JFrame
{
    JProgressBar progress=new JProgressBar();//创建, 并初始化 JProgressBar 类型 progress
    int count=0;                               //声明整型变量 count
    //创建内部类 Task, 设置滚动条的值
    class Task extends java.util.TimerTask
    {
        //覆盖 TimerTask 的 run 方法
        public void run()
        {
            progress.setValue(count++);          //执行的时候为 JProgressBar 赋值
        }
    }
    //声明 JProgressBarDemo 类的构造方法
    public JProgressBarDemo()
    {
        progress.setStringPainted(true);          //指定显示百分比
        this.getContentPane().add(progress,"North");//获取内容面板, 并添加进度条 progress
        Task task=new Task();                      //创建, 并初始化 Task 对象
        java.util.Timer timer=new java.util.Timer(); //创建, 并初始化 Timer 对象 timer
        timer.schedule(task,100,100);              //安排指定时间执行, 重复执行任务
        this.setSize(500,100);                     //设置窗口的大小
        this.setVisible(true);                      //显示窗口
        //设置窗口关闭窗口, 自动隐藏并释放窗口
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void excute()
    {
    }
    public static void main(String[ ] args)
    {
        JProgressBarDemo p=new JProgressBarDemo();//创建, 并初始化 JProgressBarDemo
        对象 p
    }
}

```

编写完程序后,使用 `javac` 命令编译该文件产生 `class` 文件,然后使用 `java` 命令运行该 `class` 文件,运行结果如图 13-7 所示。

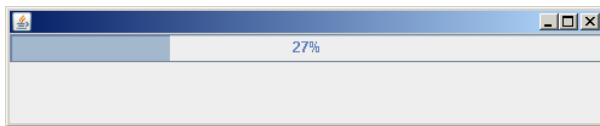


图 13-7 JProgressBarDemo.java 运行结果

程序 13.5 中,先创建了 `JProgressBar` 对象 `progress`,程序通过 `TimerTask` 派生类 `Task` 完成 `progress` 的动态显示,创建 `Timer` 对象 `timer`,通过 `timer` 安排执行任务 `Task` 对象。

### 13.4.6 表格

表格 (`JTable`) 是 `Swing` 新增加的组件,主要是为了将数据以表格的形式显示。通常用数据模型类的对象来保存数据,数据模型类派生于 `AbstractTableModel` 类,并且必须重写抽象模型类的几个方法,例如, `getColumnCount`、`getRowCount`、`getColumnName`、`getValueAt`。因为表格会从这个数据模型的对象中自动获取数据,数据模型类的对象负责表格大小(行/列)、数据填写、表格单元更新等与表格有关的属性和操作。表格类的继承关系如下。

```
java.lang.Object
+--java.awt.Component
+--java.awt.Container
+--javax.swing.JComponent
+--javax.swing.JTable
```

表格类的构造方法有以下几种。

- `JTable()`: 使用系统默认模型创建一个 `JTable` 实例。
- `JTable(int numRows,int numColumns)`: 创建一个使用 `DefaultTableModel` 指定行、列的空表格。
- `JTable(Object[][] rowData,Object[][] columnNames)`: 创建一个显示二维数据的表格。
- `JTable(TableModel dm)`: 创建一个指定数据模式和默认字段模式的 `JTable` 实例。
- `JTable(TableModel dm,TableColumnModel cm)`: 创建一个指定数据模式和字段模式的 `JTable` 实例。
- `JTable(TableModel dm,TableColumnModel cm,ListSelectionModel sm)`: 创建一个指定数据模式、字段模式与选择模式的 `JTable` 实例。
- `JTable(Vector rowData,Vector columnNames)`: 创建一个以 `Vector` 为数据源,并显示行名称的 `JTable` 实例。

下面是一个 `JTable` 应用的例子。

```
// 文件: 程序 13.6    TableDemo.java    描述: JTable 演示
//导入需要使用的包和类
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
```

```

import javax.swing.JOptionPane;
import java.awt.*;
import java.awt.event.*;
public class TableDemo extends JFrame {
    //实现构造方法
    public TableDemo() {
        super("JTable 演示程序"); //调用父类构造方法
        TableModel model = new TableModel(); //model 存放表格的数据
        JTable table = new JTable(model); //表格 table 从 model 中获取数据源
        table.setPreferredScrollableViewportSize(new Dimension(400, 200)); //设置窗口的大小
        JScrollPane scrollPane = new JScrollPane(table); //声明一个滚动条面板
        getContentPane().add(scrollPane, BorderLayout.CENTER); //滚动条面板加入窗口
        addWindowListener(new WindowAdapter() { //注册窗口监听器
            public void windowClosing(WindowEvent e) {
                System.exit(0); //退出程序
            }
        });
    }
    class TableModel extends AbstractTableModel {
        //将列名保存与字符串数组 columnNames 中
        final String[] columnNames = {"姓名", "学号", "年龄", "籍贯", "党员"};
        //将表格数据保存到 Object 数组
        final Object[][] data = {
            {"赵大元", "S050107001", new Integer(23), "北京", new Boolean(false)},
            {"李小乐", "S050107002", new Integer(22), "天津", new Boolean(true)},
            {"孙 月", "S050107003", new Integer(23), "上海", new Boolean(false)},
            {"周元元", "S050107004", new Integer(23), "重庆", new Boolean(true)},
            {"刘 涛", "S050107005", new Integer(24), "浙江", new Boolean(false)},
            {"杨 光", "S050107006", new Integer(25), "江西", new Boolean(true)},
        };
        //获得列的数目
        public int getColumnCount() {
            return columnNames.length; //返回列的数目
        }
        //获得行的数目
        public int getRowCount() {
            return data.length; //返回行的数目
        }
        //获得某列的名字
        public String getColumnName(int col) {
            return columnNames[col]; //返回指定列的名字
        }
        //获得某行某列的数据
        public Object getValueAt(int row, int col) {
            return data[row][col]; //返回指定行、列的数据
        }
        //获得指定列的类型
        public Class getColumnClass(int c) {
            return getValueAt(0, c).getClass(); //返回指定列的类型
        }
        //使得表格具有可编辑性
        public boolean isCellEditable(int row, int col) {
            if (col < 2) { //当 col 小于 2
                return false; //返回 false
            }
        }
    }
}

```

```

    } else {
        return true;
    }
}
//使得该表格可以修改数据
public void setValueAt(Object value, int row, int col) {
    if (data[0][col] instanceof Integer && !(value instanceof Integer)) //判断数据类型
    {
        try {
            data[row][col] = new Integer(value.toString()); //设置 data[row][col]的值
            fireTableCellUpdated(row, col); //更新指定行, 列数据
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(TableDemo.this, //显示报错对话框
                "The \"" + getColumnName(col)
                + "\" 本列只接受整型数据");
        }
    }
    else {
        data[row][col] = value; //设置 data[row][col]的值
        fireTableCellUpdated(row, col); //更新指定行, 列数据
    }
}
}
}
public static void main(String[] args) {
    TableDemo frame = new TableDemo(); //创建, 并初始化 TableDemo 对象 frame
    frame.pack(); //调整窗口
    frame.setVisible(true); //显示窗口
}
}

```

编写完程序后,使用 javac 命令编译该文件产生 class 文件,然后使用 java 命令运行该 class 文件,运行结果如图 13-8 所示。

程序 13.6 中,实现了继承 AbstractTableModel 的 MyTableModel 类,该类声明字符串数组 columnNames 用于存放列的名称,声明二维 Object 类型数组 data 用于保存数据。getColumnClass 方法用于将最后一列的信息以复选框形式表现,实现 isCellEditable 可以使表格具有可编辑功能,setValueAt 方法用于修改表格中的某项数据。

姓名	学号	年龄	籍贯	党员
赵大元	S050107001	23	北京	<input type="checkbox"/>
李小东	S050107002	22	天津	<input checked="" type="checkbox"/>
孙月	S050107003	23	上海	<input type="checkbox"/>
周元元	S050107004	23	重庆	<input checked="" type="checkbox"/>
刘清	S050107005	24	浙江	<input checked="" type="checkbox"/>
杨光	S050107006	25	江西	<input checked="" type="checkbox"/>

图 13-8 TableDemo.java 运行结果

### 13.4.7 树

树(JTree)中特定的节点可以由 TreePath 标识或由其显示行标识。当展开某一个节点的所有祖先时,将显示该节点,折叠节点是隐藏位于折叠祖先下面的节点。可查看节点的父类节点都是可以展开的,但是它们可以显示也可以不显示。显示节点必须是可查看的并且位于显示区域。

与显示相关的方法如下。

```
isRootVisible() //返回树的根节点是否显示
```

```

setRootVisible()      //设置是否显示树的根节点
scrollPathToVisible() //确保展开所有的路径
scrollRowToVisible()  //按行滚动标识的条目，直到显示出来
getVisibleRowCount()  //返回显示区域的显示行的数目
setVisibleRowCount()  //设置显示区域中显示行的数目

```

与可查看相关的方法如下。

```

isVisible()      //返回当前路径查看标识
makeVisible()    //设置当前路径的查看标识

```

树的构造方法有以下几种。

- ❑ `JTree()`: 创建一个空节点的树。
- ❑ `JTree(Hashtable<?,?> value)`: 创建一个由 `Hashtable` 元素构成节点的 `JTree`，它不显示根。
- ❑ `JTree(Object[] value)`: 创建一个指定数组的元素作为不给显示的新的根节点的子节点的树实例。
- ❑ `JTree(TreeModel newModel)`: 创建一个使用指定数据模型，并显示根节点的树实例。
- ❑ `JTree(TreeNode root)`: 创建一个指定 `TreeNode` 作为其根，并显示根节点的树实例。
- ❑ `JTree(TreeNode root, boolean asksAllowsChildren)`: 创建一个指定 `TreeNode` 作为其根，并指定根节点的显示方式的树实例。

下面是一个使用 `JTree` 的实例。

```

// 文件: 程序 13.7    JTreeDemo.java        描述: JTree 演示
//导入需要使用的包和类
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.*;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.DefaultTreeModel;
public class JTreeDemo implements ActionListener, TreeModelListener
{
    public static void main (String[] args)
    {
        new JTreeDemo();                //创建 JTreeDemo 对象
    }
    public JTreeDemo()
    {
        DefaultTreeModel model = new DefaultTreeModel (MakeRootTree()); //创建一个树模型
        tree = new JTree (model);      //再用树模型作为 JTree 的参数构一个树
        tree.setEditable(true);        //设置树为可编辑状态
        tree.addMouseListener(new MouseHandle()); //为树添加监听器 MouseHandle
        treemodel=(DefaultTreeModel)tree.getModel(); //返回正在提供数据的 TreeModel
        treemodel.addTreeModelListener(this); //添加监听器 TreeModelEvent
        JFrame frame =new JFrame("JTree 应用演示");//创建，并初始化 JFrame 对象 frame
    }
}

```

对象

```

Container contentPane = frame.getContentPane(); //返回此窗体的 contentPane 对象
JScrollPane scrollPane = new JScrollPane(tree); //创建一个显示 scrollPane 的 JScrollPane

contentPane.add(scrollPane, "Center"); //在内容面板中添加 scrollPane 于 Center 位置
InitPanel(); //调用 InitPanel 方法
contentPane.add(panel, "South"); //为 contentPane 添加面板 panel 于 South 位置
//设置窗口关闭窗口，自动隐藏并释放窗口
frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
frame.setSize(250,300); //设置窗口大小
frame.setVisible(true); //显示窗口
}
//声明，实现 MakeRootTree 方法
public DefaultMutableTreeNode MakeRootTree()
{
    //创建，并初始化树数据结构中的通用节点
    DefaultMutableTreeNode root = new DefaultMutableTreeNode ("Root");
    DefaultMutableTreeNode child1 = new DefaultMutableTreeNode ("第一层");
    DefaultMutableTreeNode child11 = new DefaultMutableTreeNode ("第二层");
    DefaultMutableTreeNode child111 = new DefaultMutableTreeNode ("第三层");
    root.add (child1); //为 root 节点添加子节点 Child1
    child1.add (child11); //为 Child1 节点添加子节点 Child11
    child11.add (child111); //为 Child11 节点添加子节点 Child111
    return root; //返回根节点
}
public void InitPanel()
{
    panel = new JPanel(); //初始化面板 panel
    addSibling = new JButton("添元素"); //初始化按钮对象 addSibling
    addChild = new JButton("添节点"); //初始化按钮对象 addChild
    delete = new JButton("删除点"); //初始化按钮对象 delete
    addSibling.addActionListener(this); //为按钮 addSibling 添加监听器
    addChild.addActionListener(this); //为按钮 addChild 添加监听器
    delete.addActionListener(this); //为按钮 delete 添加监听器
    panel.add(addSibling); //panel 中添加 addSibling 按钮
    panel.add(addChild); //panel 中添加 addChild 按钮
    panel.add(delete); //panel 中添加 delete 按钮
}
public void actionPerformed(ActionEvent event) {
    //获取选择的树形节点
    DefaultMutableTreeNode selectedNode =
        (DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
    if (selectedNode == null) //判读选择的节点是否为 null
        return;
    if (event.getSource().equals(delete)) { //判断事件源是否是 delete
        if (selectedNode.getParent() != null) //判断所选节点的父节点是否为 null
            treemodel.removeNodeFromParent(selectedNode); //移除所选节点
        return;
    }
    //创建新的节点
    DefaultMutableTreeNode newNode = new DefaultMutableTreeNode("New");
    if (event.getSource().equals(addSibling)) { //判断事件源是否是添加兄弟节点
        //获取被选节点的父节点
        DefaultMutableTreeNode parent = (DefaultMutableTreeNode) selectedNode.getParent();
        if (parent != null) {

```

```

        int selectedIndex = parent.getIndex(selectedNode); //在父节点中获取选择节点索引
        //将新建节点插入父节点下面的 selectedIndex + 1 索引处
        treemodel.insertNodeInto(newNode, parent, selectedIndex + 1);
    }
    } else if (event.getSource().equals(addChild)) { //判断事件源是否是添加子节点
        //将新建节点插入父节点下面的 selectedIndex.getChildCount()索引处
        treemodel.insertNodeInto(newNode, selectedNode, selectedNode.getChildCount());
    }
    //显示新的节点
    TreeNode[] nodes = treemodel.getPathToRoot(newNode); //向上构建节点的父节点
    TreePath path = new TreePath(nodes); //根据 TreeNode 的数组构造路径
    tree.scrollPathToVisible(path); //显示新节点
}

public void treeNodesChanged(TreeModelEvent event) { //覆盖接口方法 treeNodesChanged
    TreePath treePath=event.getTreePath(); //返回已更改节点的父节点
    //返回指定索引位置的路径组件
    DefaultMutableTreeNode
node=(DefaultMutableTreeNode)treePath.getLastPathComponent();
    try{
        int[] index=event.getChildIndices(); //返回子索引的值
        node=(DefaultMutableTreeNode)node.getChildAt(index[0]);//返回子节点数组 index[0]的子
节点
    }catch(NullPointerException exc){ //捕获异常
        System.out.println(nodeName+"更改数据为:"+String.valueOf(node.getUserObject())); //输出字符串信息
    }
    public void treeNodesInserted(TreeModelEvent arg0) { //覆盖接口方法 treeNodesInserted
        System.out.println("插入节点"); //输出字符串信息
    }
    public void treeNodesRemoved(TreeModelEvent arg0) { //覆盖接口方法 treeNodesRemoved
        System.out.println("删除节点"); //输出字符串信息
    }
    public void treeStructureChanged(TreeModelEvent arg0) { //覆盖接口方法
treeStructureChanged
        System.out.println("结构改变"); //输出字符串信息
    }
    class MouseHandle extends MouseAdapter{ //实现鼠标适配器的派生类 MouseHandle
        public void mousePressed(MouseEvent e){
            try{
                JTree tree=(JTree)e.getSource(); //获取发生事件的对象
                int rowLocation=tree.getRowForLocation(e.getX(),e.getY()); //返回指定位置
处节点路径
                if (rowLocation < 0 ) return; //若 rowLocation 小于 0, 返回
                TreePath treepath=tree.getPathForRow(rowLocation);//返回指定行的路径
                //返回指定索引位置的路径组件
                TreeNode treenode=(TreeNode)treepath.getLastPathComponent();
                nodeName = treenode.toString(); //节点转化为字符串信息
            }catch(NullPointerException ne) //捕获异常
            {
                ne.printStackTrace(); //输出异常信息
            }
        }
    }
}
private JTree tree = null; //声明 JTree 类型域 tree

```

```

private DefaultTreeModel treemodel = null;           //声明 DefaultTreeModel 类型域 treemodel
private JPanel panel;                               //声明 JPanel 类型域 panel
private String nodeName = null;                     //声明 String 类型域 nodeName
private JButton addSibling;                          //声明 JButton 类型域 addSibling
private JButton addChild;                           //声明 JButton 类型域 addChild
private JButton delete;                             //声明 JButton 类型域 delete
}

```

编写完程序后，使用 `javac` 命令编译该文件产生 `class` 文件，然后使用 `java` 命令运行该 `class` 文件，运行结果如图 13-9 所示。

程序 13.7 是一个比较复杂的程序，通过 `MakeRootTree` 方法创建一棵树，再通过该树创建 `DefaultTreeModel` 对象实例 `model`，然后通过 `model` 创建树实例 `tree`，为 `tree` 添加鼠标监听器 `MouseListener`，通过 `tree` 的 `getModel` 方法获取模型 `treemodel`，为 `treemodel` 添加 `TreeModelListener` 监听器。

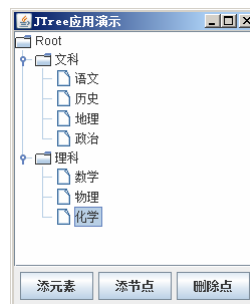


图 13-9 JTreeDemo.java 运行结果

## 13.5 盒布局管理器

为了容器中的组件实现平台无关的自动排列，`Swing` 继续保持了 `AWT` 组件布局原理。`Swing` 也采用了布局管理器来管理 `Swing` 组件大小、位置布局，但是在 `AWT` 基础上显示风格有所改进。

`Swing` 也有顶层容器，但不能将组件直接添加到顶层容器中，而是添加到内容面板上。另外 `Swing` 新增了一个盒布局管理器（`BoxLayout`），下面将讨论盒布局管理器的使用。

盒布局管理器是允许水平（`x` 轴）或者垂直（`y` 轴）顺序布局依次添加的组件的布局管理器。盒布局管理器通过参数指定布局类型。缺省情况下，组件在纵轴方向上居中对齐。设置布局管理器的方法如下。

```
pane.setLayout(new BoxLayout(pane,int axis));
```

`axis` 的取值有以下几种。

- ❑ `X_AXIS`：水平方法布局组件。
- ❑ `Y_AXIS`：垂直方法布局组件。
- ❑ `LINE_AXIS`：组件的布局是根据容器的 `ComponentOrientation` 属性，按照一行文字的排列方式放置组件。对于垂直方法的排列顺序，组件总是先上后下放置。
- ❑ `PAGE_AXIS`：组件的布局是根据容器的 `ComponentOrientation` 属性，按照一页中的文本行排列方式布置组件。对于垂直方法的排列顺序，组件总是先上后下放置。

下面是一个盒布局管理器应用的例子。

```

// 文件：程序 13.8      BoxLayoutDemo.java      描述：BoxLayout 演示
//导入需要使用的包和类
import java.awt.*;
import javax.swing.*;
public class BoxLayoutDemo {

```



```

public static void main(String[ ] args) {
    JFrame f=new JFrame("BoxLayout 演示程序");           //创建，并初始化窗口 f
    f.setBounds(80,60,300,230);                          //移动组件并调整大小
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE); //窗口关闭，自动隐藏并释
放该窗体
    int SIZE=3;                                           //声明，并赋值整型变量 SIZE
    Container c =f.getContentPane();                    //获取窗口内容面板
    c.setLayout(new BorderLayout(30,30));                //设置布局管理器 BorderLayout
    Box boxes[ ] = new Box[4];                          //创建，并初始化 Box 数组
    //初始化数组
    boxes[0]=Box.createHorizontalBox();                  //创建一个从左到右显示 Box
    boxes[1]=Box.createVerticalBox();                    //创建一个从上到下显示 Box
    boxes[2]=Box.createHorizontalBox();                  //创建一个从左到右显示 Box
    boxes[3]=Box.createVerticalBox();                    //创建一个从上到下显示 Box
    //在 boxes[0]对象中添加按钮组件
    for(int i=0;i<SIZE;i++)                             //循环添加按钮对象 boxes[0]
    boxes[0].add(new JButton("boxes[0]["+i+"]"));
    //在 boxes[1]对象中添加按钮组件
    for (int i=0;i<SIZE;i++)                             //循环添加按钮对象 boxes[1]
    boxes[1].add(new JButton("boxes[1]["+i+"]"));
    //在 boxes[2]对象中添加按钮组件
    for (int i=0;i<SIZE;i++)                             //循环添加按钮对象 boxes[2]
    boxes[2].add(new JButton("boxes[2]["+i+"]"));
    //在 boxes[3]对象中添加按钮组件
    for (int i=0;i<SIZE;i++)                             //循环添加按钮对象于 boxes[3]
    boxes[3].add(new JButton("boxes[3]["+i+"]"));
    c.add(boxes[0],BorderLayout.NORTH); //在内容面板中添加 boxes[0]在 NORTH 位置
    c.add(boxes[1],BorderLayout.EAST);  //在内容面板中添加 boxes[1]在 EAST 位置
    c.add(boxes[2],BorderLayout.SOUTH); //在内容面板中添加 boxes[2]在 SOUTH 位置
    c.add(boxes[3],BorderLayout.WEST);  //在内容面板中添加 boxes[3]在 WEST 位置
    f.setVisible(true);                               //显示窗口
}
}

```



图 13-10 BoxLayoutDemo.java 运行结果

编写完程序后，使用 `javac` 命令编译该文件产生 `class` 文件，然后使用 `java` 命令运行该 `class` 文件，运行结果如图 13-10 所示。

程序 13.8 中主要使用了 `BoxLayout` 布局管理器，该程序创建并初始化 `Box` 数组，为每一个元素添加 3 个按钮对象，然后分别将 `Box` 数组元素添加于窗口的不同位置。

## 13.6 案例——Swing 版 NoteBook

在上一章中，已经编写了一个基于 AWT 的 `NoteBook` 程序。本章学习了 `Swing` 图形设计，所以本节通过 `Swing` 实现记事本程序，读者通过比较、思考会更容易掌握基础知识。

### 13.6.1 域和构造方法

记事本类的域主要包括菜单栏 `JMenuBar`、主文本输入区 `JTextArea`、菜单 `JMenu`、菜单项 `JMenuItem` 以及文件对话框 `FileDialog` 等。`NoteBook` 的构造方法完成对域的初始化、组件位置安排和为组件添加事件监听器。

```
JMenu fileMenu = new JMenu("文件");           //创建，并初始化菜单对象 fileMenu
JMenu editMenu = new JMenu("编辑");           //创建，并初始化菜单对象 editMenu
JMenuItem newItem = new JMenuItem("新建");     //创建，并初始化菜单项对象 newItem
JMenuItem openItem = new JMenuItem("打开");    //创建，并初始化菜单项对象 openItem
JMenuItem saveItem = new JMenuItem("保存");     //创建，并初始化菜单项对象 saveItem
JMenuItem saveAsItem = new JMenuItem("另存");  //创建，并初始化菜单项对象 saveAsItem
JMenuItem exitItem = new JMenuItem("退出");    //创建，并初始化菜单项对象 exitItem
JMenuItem selectItem = new JMenuItem("全选");  //创建，并初始化菜单项对象 selectItem
JMenuItem copyItem = new JMenuItem("复制");    //创建，并初始化菜单项对象 copyItem
JMenuItem cutItem = new JMenuItem("剪切");     //创建，并初始化菜单项对象 cutItem
JMenuItem pasteItem = new JMenuItem("粘贴");   //创建，并初始化菜单项对象 pasteItem
//创建，并初始化打开文件对话框
private FileDialog openFileDialog = new FileDialog(this, "Open File", FileDialog.LOAD);
//创建，并初始化保存文件对话框
private FileDialog saveAsFileDialog = new FileDialog(this, "Save File As", FileDialog.SAVE);
String fileName = "NoName.txt";               //设置默认的文件名
final JTextArea textArea = new JTextArea();    //创建，并初始化文本区
JMenuBar menuBar = new JMenuBar();             //创建，并初始化菜单栏
String title = "ERROR MESSAGE";               //创建，并初始化字符串对象
int type = JOptionPane.ERROR_MESSAGE;         //创建，并初始化整型变量
public NoteBook() {
    setSize(600, 400);                        //设置窗口大小
    setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE); //设置关闭窗口时，释放窗口资源
    JPanel panel = new JPanel();               //创建，并初始化面板对象 panel
    panel.setLayout(new GridLayout(1,1));      //设置布局管理器 GridLayout
    panel.add(new JScrollPane(textArea));      //为文本区添加滚动条
    getContentPane().add(panel);              //获取内容面板，并给其添加面板
    setJMenuBar(menuBar);                     //为窗口设置菜单栏
    menuBar.add(fileMenu);                    //将文件菜单添加于菜单栏
    menuBar.add(editMenu);                    //将编辑菜单添加于菜单栏
    fileMenu.add(newItem);                     //将“新建”菜单项添加于“文件”菜单
    fileMenu.add(openItem);                   //将“打开”菜单项添加于“文件”菜单
    fileMenu.addSeparator();                  //添加分隔条与“文件”菜单
    fileMenu.add(saveItem);                   //将“保存”菜单项添加于“文件”菜单
    fileMenu.add(saveAsItem);                 //将“另存”菜单项添加于“文件”菜单
    fileMenu.addSeparator();                  //分隔条添加于“文件”菜单
    fileMenu.add(exitItem);                   //将“退出”菜单项添加于“文件”菜单
    editMenu.add(selectItem);                 //将“选择”菜单项添加于“编辑”菜单
    editMenu.addSeparator();                  //将分隔符添加于“编辑”菜单
    editMenu.add(copyItem);                   //将“复制”菜单项添加于“编辑”菜单
    editMenu.add(cutItem);                    //将“剪切”菜单项添加于“编辑”菜单
    editMenu.add(pasteItem);                  //将“粘贴”菜单项添加于“编辑”菜单
    newItem.addActionListener(this);          //newItem 菜单项注册事件监听器
    newItem.setMnemonic('N');                 //为 newItem 菜单项设置按钮“N”助记符
}
```

```

newItem.setAccelerator( KeyStroke.getKeyStroke('N',java.awt.Event.CTRL_MASK,true)); // 设置加速器
openItem.addActionListener(this); //openItem 菜单项注册事件监听器
openItem.setMnemonic('O'); //为 newItem 菜单项设置按钮“O”助记符
openItem.setAccelerator(KeyStroke.getKeyStroke('O',java.awt.Event.CTRL_MASK,true)); // 设置加速器
saveItem.addActionListener(this); //newItem 菜单项注册事件监听器
saveItem.setMnemonic('S'); //为 newItem 菜单项设置按钮“S”助记符
saveItem.setAccelerator(KeyStroke.getKeyStroke('S',java.awt.Event.CTRL_MASK,true)); // 设置加速器
saveAsItem.addActionListener(this); //newItem 菜单项注册事件监听器
saveAsItem.setMnemonic('T'); //为 newItem 菜单项设置按钮“T”助记符
saveAsItem.setAccelerator(KeyStroke.getKeyStroke('T',java.awt.Event.CTRL_MASK,true)); // 设置加速器
exitItem.addActionListener(this); //newItem 菜单项注册事件监听器
exitItem.setMnemonic('Q'); //为 newItem 菜单项设置按钮“Q”助记符
exitItem.setAccelerator(KeyStroke.getKeyStroke('Q',java.awt.Event.CTRL_MASK,true)); // 设置加速器
selectItem.addActionListener(this); //newItem 菜单项注册事件监听器
selectItem.setMnemonic('A'); //为 newItem 菜单项设置按钮“A”助记符
selectItem.setAccelerator(KeyStroke.getKeyStroke('A',java.awt.Event.CTRL_MASK,true)); // 设置加速器
copyItem.addActionListener(this); //newItem 菜单项注册事件监听器
copyItem.setMnemonic('C'); //为 newItem 菜单项设置按钮“C”助记符
copyItem.setAccelerator(KeyStroke.getKeyStroke('C',java.awt.Event.CTRL_MASK,true)); // 设置加速器
cutItem.addActionListener(this); //newItem 菜单项注册事件监听器
cutItem.setMnemonic('X'); //为 newItem 菜单项设置按钮“X”助记符
cutItem.setAccelerator(KeyStroke.getKeyStroke('X',java.awt.Event.CTRL_MASK,true)); // 设置加速器
pasteItem.addActionListener(this); //newItem 菜单项注册事件监听器
pasteItem.setMnemonic('P'); //为 newItem 菜单项设置按钮“P”助记符
pasteItem.setAccelerator(KeyStroke.getKeyStroke('P',java.awt.Event.CTRL_MASK,true)); // 设置加速器
setVisible(true); //显示窗口
}

```

构造方法主要作用就是初始化、布局组件对象。该构造方法先设置窗口的大小，为窗口设置关闭窗口的默认操作，设置布局管理器为 `GridLayout`，获取窗口的容器面板，为窗口设置菜单栏 `menuBar`，在菜单栏中添加菜单，在菜单中添加菜单项，最后为菜单项添加事件监听器，并为其设置快捷键。

### 13.6.2 事件处理方法

当用户为某个组件添加了一个事件监听器，必须定义特定的事件处理方法。当相应的事件发生时，就会调用该事件处理方法。

```

public void actionPerformed(ActionEvent e) {
    Object eventSource = e.getSource();
    if(eventSource == newItem){ //发生的事件源为 newItem 时

```

```

        textArea.setText(""); //新建时清空文本区
    } else if(eventSource == openItem) { //发生的事件源为 openItem 时
        readFile(); //调用 readFile 方法
    } else if(eventSource == saveItem) { //发生的事件源为 saveItem 时
        fileName = "NoName.txt"; //设置文件名为 NoName.txt
        writeFile(fileName); //调用 writeFile
    } else if(eventSource == saveAsItem) { //发生的事件源为 saveAsItem 时
        fileName=null;
        writeFile(fileName); //调用 writeFile
    }else if(eventSource == selectItem){ //发生的事件源为 selectItem 时
        textArea.selectAll(); //全选文本区文本内容
    }else if(eventSource == copyItem){ //发生的事件源为 copyItem 时
        textArea.copy(); //完成文本拷贝操作
    }else if(eventSource == cutItem){ //发生的事件源为 cutItem 时
        textArea.cut(); //完成文本剪切操作
    }else if(eventSource == pasteItem){ //发生的事件源为 pasteItem 时
        textArea.paste(); //完成文本粘贴
    }else if(eventSource == exitItem){ //发生的事件源为 exitItem 时
        System.exit(0); //退出程序
    }
}

```

由于这个案例为菜单项添加了 ActionListener 接口，所以必须定义方法 actionPerformed。actionPerformed 方法的参数为 ActionEvent 的对象 e，调用 e.getSource()方法返回事件源，程序将会根据事件源做出相应的处理。

### 13.6.3 文件操作

通过上面两节已经将记事本的界面和界面中组件的事件定义好了，记事本应该具有一个最基本的文件方面的操作，例如打开文件、读文件等。代码如下。

```

//读文件
public void readFile(){
    JFileChooser openfile=new JFileChooser(); //创建，并初始化 JFileChooser 对象
    openfile
    openfile.setDialogTitle("打开文件"); //设置对话框的标题
    openfile.setApproveButtonText("打开"); //设置 ApproveButton 的文本
    openfile.showOpenDialog(this); //弹出一个打开对话框
    File file=openfile.getSelectedFile(); //返回选中的文件
    if (file != null) //若 file 为 null
    {
        FileInputStream inputfile=null; //创建文件输出流对象
        String message="文件不能找到"; //初始化字符串对象 message
        try{
            inputfile=new FileInputStream(file); //通过文件，初始化 inputfile 对象
        }catch(FileNotFoundException fe) //捕获异常
        {
            JOptionPane.showMessageDialog(this,message,title,type); //显示对话框
        }
        int readbytes; //声明整数类型变量 readbytes
        String message1="读文件发生错误"; //创建，并初始化字符串对象 message1
    }
}

```

```

        try{
            while((readbytes=inputfile.read())!=-1) //判断是否读完文件
            {
                textArea.append(String.valueOf((char)readbytes)); //追加到文本区
            }
        }
        catch(IOException ioe) //捕获异常
        {
            JOptionPane.showMessageDialog(this,message1,title,type); //显示对话框
        }
        String closemessage="关闭文件发生错误"; //创建,并初始化字符串对象 closemessage
        try{ //try 语句块, 检查是否发生异常
            inputfile.close(); //关闭输入文件流
        }
        catch(IOException ioe) //捕获异常
        {
            JOptionPane.showMessageDialog(this,closemessage,title,type); //显示对话框
        }
    }
}

//写文件
public void writeFile(String fileName){
    File filesa; //创建 File 对象
    String messagef="文件未找到"; //创建,并初始化字符串对象 messagef
    FileOutputStream outputfile=null; //创建 FileOutputStream 对象
    if(fileName == null)
    {
        JFileChooser savefile=new JFileChooser(); //创建,并初始化文件选择对象
        savefile.setApproveButtonText("保存"); //设置 ApproveButton 文本内容
        savefile.setDialogTitle("保存文件"); //设置对话框标题
        savefile.showSaveDialog(this); //显示保存对话框
        filesa = savefile.getSelectedFile(); //返回选择的文件
        if (filesa == null) return;
    }else
    {
        filesa = new File(fileName); //创建新的文件
    }
    try{
        outputfile=new FileOutputStream(filesa); //创建,并初始化文件输出流
    }
    catch(FileNotFoundException fe) //捕获异常
    {
        JOptionPane.showMessageDialog(this,messagef,title,type); //显示报错对话框
    }
    String filecontent=textArea.getText(); //获取文本区文本内容
    String wrmessage="下文件错误";
    try
    {
        outputfile.write(filecontent.getBytes()); //进行写文件操作
    }
    catch(IOException ioe)
    {
        JOptionPane.showMessageDialog(this,wrmessage,title,type); //显示信息对话框
    }
    String cmessage="关闭流发生错误"; //创建,并初始化字符串对象

```

```

    try{
        outputfile.close();                //关闭流
    }
    catch(IOException ioe)                //捕获异常
    {
        JOptionPane.showMessageDialog(this,cmessage,title,type);    //显示信息对话框
    }
}

```

#### 13.6.4 主方法

本程序的主方法先实例化一个 `NoteBook` 对象，代码如下。

```

public static void main(String[] args) {
    NoteBook NoteBook1 = new NoteBook();
}

```

编写完程序后，使用 `javac` 命令编译该文件产生 `class` 文件，然后使用 `java` 命令运行该 `class` 文件，运行结果如图 13-11 所示。

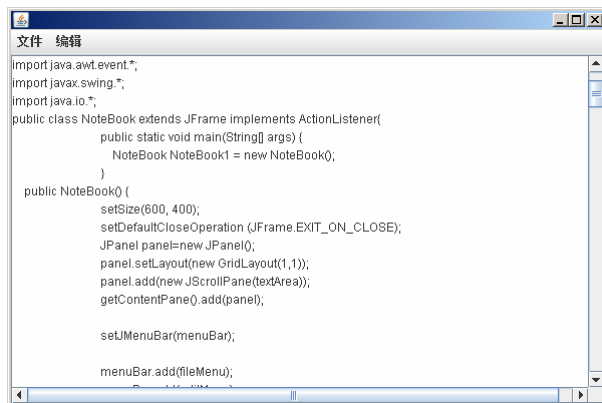


图 13-11 `NoteBook.java` 运行结果

程序 13.10 中，`NoteBook` 类继承了 `JFrame` 窗口类，并实现 `ActionListener` 接口的 `actionPerformed` 方法。该程序中的绝大部分功能主要体现于 `JTextArea` 类对象 `textArea`，`JTextArea` 提供文本的一些操作，包括 `selectAll`、`copy`、`cut`、`paste` 方法。另外还实现了两个重要的方法，包括 `readFile` 用于读取文件，`writeFile` 用于保存文件。

## 13.7 小结

本章主要讨论了 `Swing` 的一些新内容，不仅介绍了创建 `Swing` 程序的基本步骤，而且还具体通过实例演示了 `Swing` 组件的使用。由于 `Swing` 提供庞大而复杂的类库，如果想熟练地掌握和应用 `Swing` 组件，还必须利用 `API` 的帮助，逐步摸索规律，掌握方法。