第8章 图形用户界面编程

内容概要

图形界面作为用户与程序交互的接口，是软件开发中一项非常重要的工作。随着用户需求的日益提高，如今的应用软件必须做到界面友好、功能强大且使用简单。本章主要介绍Java图形用户界面设计的相关基础知识，包括容器、基本组件、布局管理器、事件处理机制、菜单、表格和树等内容。通过本章的学习，读者会掌握图形用户界面的设计与实现。

学习目标

- 了解AWT和Swing之间的关系
- 掌握布局管理器的使用方法
- 理解GUI中的事件处理机制
- 掌握常用容器的使用方法
- 了解基本组件的使用方法
- 了解菜单、表格和树的用法

8.1 GUI编程概述

图形用户界面就是界面元素的有机合成。这些元素不仅在外观上相互关联，在内在上也具有逻辑关系，通过相互作用、消息传递完成对用户操作的响应。

设计和实现图形用户界面，主要包含两项内容。

（1）创建图形界面中需要的元素，进行相应的布局。

（2）定义界面元素对用户交互事件的响应以及对事件的处理。

Java中的图形用户界面是通过Java的GUI（Graphics User Interface，图形用户接口）实现的。无论采用JavaSE、JavaEE还是JavaME，GUI都是其中的关键部分。

在Java中为了方便图形用户界面的设计与实现，专门设计了类库来满足各种各样的图形界面元素和用户交互事件，该类库即为抽象窗口工具箱（Abstract Window Toolkit，AWT）。AWT是1995年随Java的发布而提出的。但随着Java的发展，AWT已经不能满足用户的需求，Sun公司于1997年在JavaOne大会上提出并在1998年5月发布的JFC（Java Foundation Class）中包含了一个新的Java窗口开发包Swing。

8.1.1 AWT与Swing的关系

AWT是早期随Java一起发布的，其目的是为程序员创建图形用户界面提供支持，其中不仅提供了基本的组件，还提供了丰富的事件处理接口。Swing是继AWT之后Sun公司推出的GUI工具包。它是建立在AWT基础上的，AWT是Swing的大厦基石。虽然AWT中提供的控件数量有限，远没有Swing中的丰富，但Swing的出现并不是为了替代AWT，而是为用户提供了更丰富的开发选择。Swing中使用的事件处理机制就是AWT提供的，所以Swing是对AWT的扩展，而且二者还存在密切的合作关系。

AWT组件定义在Java.awt包中，而Swing组件则定义在javax.swing包中。AWT和Swing包含了部分对应的组件。例如，标签和按钮，在java.awt包中分别用Label和Button表示，而在javax.swing包中则分别用JLabel和JButton表示，多数Swing组件以字母J开头。

Swing组件与AWT组件最大的不同是，Swing组件在实现时不包含任何本地代码。因此Swing组件可以不受硬件平台的限制，而且具有更多的功能。所以，在进行图形用户界面设计时，建议读者使用Swing组件。

与AWT相比，Swing组件显示出强大的优势，具体表现如下。

（1）丰富的组件类型

Swing提供了非常丰富的标准组件。基于良好的可扩展性，除了标准组件，Swing还提供了大量的第三方组件。

（2）更好的组件API模型支持

Swing遵循MVC模式，这是一种非常成功的设计模式，API成熟并设计良好。经过多年演化，Swing组件API变得越来越强大，灵活并且可扩展。

（3）标准的GUI库

Swing是JRE中的标准库，且与平台无关，用户不用担心平台兼容性。

（4）性能更稳定

Swing包中的组件是纯Java实现的，不会有兼容性问题。Swing在每个平台上都有同样的性能，不会有明显的性能差异。

8.1.2 GUI元素的分类

Java中构成图形用户界面的各种元素和成分可以粗略分为三类：容器、控制组件和用户自定义成分。

（1）容器

容器是用来组织或容纳其他界面成分和元素的组件。一个容器可以包含许多组件，它本身也可以作为一个组件。一般来说，一个应用程序的图形用户界面对应一个复杂的容器，例如一个窗口。这个容器内部包含许多界面成分和元素，其中某些界面元素本身也可能是一个容器，这个容器进一步包含它的界面成分和元素。以此类推就构成了一个复杂的图形界面系统。

容器是Java中的类，如框架（JFrame）、面板（JPanel）及滚动面板（JScrollPanel）等。容器的引入有利于分解复杂的图形用户界面。当界面比较复杂、功能较多时，可能需要多个容器进行嵌套才能实现。

（2）控制组件

与容器不同，控制组件是图形用户界面的最小单位，里面不包含其他的成分。控制组件的作用是完成与用户的交互，包括接收用户的命令，接收用户输入的文本或选择，向用户显示一段文本或图形等。

某种程度上，控制组件是图形用户界面标准化的结果，常用的控制组件有选择类的单选按钮、复选按钮、下拉列表，有文字处理类的文本框、文本区域，有命令类的按钮、菜单等。使用控制组件，通常需要如下的步骤。

①创建某控制组件类的对象，指定其大小等属性。

②使用某种布局策略，将该控制组件对象加入某个容器中的指定位置。

③将该组件对象注册所能产生的事件对应的事件监听程序，重载事件处理方法，实现利用该组件对象与用户交互的功能。

（3）用户自定义成分

除了上述的标准图形界面元素外，编程人员还可以根据用户的需要，使用各种字型、字体和色彩设计一些几何图形、标志图案等，它们被称为用户自定义成分。用户自定义成分通常只能起到装饰、美化的作用，而不能响应用户的动作，也不具有交互功能。

8.2 常用容器类

Java的图形用户界面由组件构成，如命令按钮、文本框等。这些组件必须放到容器中才能被用户使用，可以通过调用容器类的add()方法把相关组件添加到容器中。

8.2.1 顶层容器类（JFrame）

JFrame是Java应用程序的图形用户界面容器，是一个带有标题行和控制按钮(最小化、恢复/最大化、关闭)的独立窗口。

JFrame类包含支持任何通用窗口特性的基本功能，如最小化窗口、移动窗口、重新设定窗口大小等。JFrame容器作为最底层容器，不能被其他容器所包含，但可以被其他容器创建并弹出成为独立的容器。JFrame类的继承关系如图8-1所示。

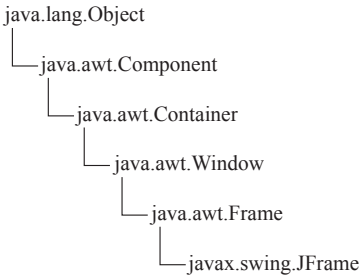


图 8-1 JFrame 类的继承关系图

JFrame类常用两种构造方法。

- **JFrame()**：构造一个初始时不可见的新窗体。
- **JFrame(String title)**：创建一个标题为title的JFrame对象。

还可以使用专门的方法getTitle( )和setTitle(String)获取或指定JFrame的标题。

创建窗体有两种方式。

- 直接编写代码调用JFrame类的构造器。这种方法适合使用简单窗体的情况。
- 继承JFrame类，在继承的类中编写代码对窗体进行详细地刻画。这种方式适合窗体比较复杂的情况。

下面通过一个案例来演示JFrame类的使用方法。

【例8-1】顶层容器JFrame类的使用。

功能实现：继承JFrame类，创建一个空白窗口，标题设置为“JFrame窗口演示”，背景色设置为红色。

```
import javax.swing.*;
import java.awt.*;
public class JFrameDemo extends JFrame{
    public  JFrameDemo(){
        this.setTitle("JFrame 窗口演示 "); // 设置窗体标题
        Container container = this.getContentPane(); // 获取当前窗体的 Container 对象
        container.setBackground(Color.red);// 设置窗体背景色为红色
        this.setVisible(true);  // 设置窗体可见
        this.setSize(350, 200); // 设置窗体大小
    }
    public static void main(String[] args) {
        new JFrameDemo(); // 创建窗体
```

```
    }
}
```

程序运行结果如图8-2所示。

图 8-2 例 8-1 的运行结果

注意事项 （1）JFrame类构造器创建的窗体不可见，需要在代码中使用setVisible(true)方法使其可见；（2）JFrame类构造器创建的窗体默认的尺寸为0×0像素，默认的位置坐标为[0,0]，因此开发中不仅要将窗体设置为可见，而且还要使用setSize(int x,int y)方法设置JFrame容器的大小；（3）利用JFrame默认的Container对象可以设置窗体颜色或添加组件。

通过调用JFrame的getContentPane()方法获得其默认的Container对象。该方法的返回类型为java.awt.Container，仍然为一个容器。然后可以将组件添加到Container中，例如：

```
Container contentPane=this.getContentPane();
contentPane.add(button); // button 为一命令按钮
```

上面两条语句可以合并为一条：

```
this.getContentPane().add(button);
```

8.2.2 中间容器类（JPanel）

面板（JPanel）是一种用途广泛的容器，但是与顶层容器不同的是面板不能独立存在，必须被添加到其他容器内部。可以将其他控件放在面板中以组织一个子界面，面板也可以嵌套，由此可以设计出复杂的图形用户界面。

JPanel是无边框的，不能被移动、放大、缩小或关闭的容器。它支持双缓冲功能，在处理动画上较少发生画面闪烁的情况。JPanel类继承自javax.swing.JComponent类，使用时首先创建该类的对象，再设置组件在面板上的排列方式，最后将所需组件加入面板中。

JPanel类的常用构造方法如下。

- **public JPanel()**：使用默认的FlowLayout方式创建具有双缓冲的JPanel对象。
- **public JPanel(FlowLayoutManager layout)**：在构建对象时指定布局格式。

下面通过一个案例来演示JPanel的使用方法。

【例8-2】面板类JPanel的使用。

功能实现：创建面板、设置面板背景色、添加按钮到面板、把面板添加到窗体。

```
import javax.swing.*;
import java.awt.*;
public class JPanelDemo extends JFrame {
    public JPanelDemo () {
        this.setTitle("JPanel 面板演示 ");
        Container container = this.getContentPane(); // 获取窗体 Container 对象
        JPanel panel = new JPanel();  // 创建一个面板对象
        panel.setBackground(Color.RED); // 设置背景颜色
        JButton bt = new JButton("Press me"); // 创建命令按钮对象
        panel.add(bt); // 把按钮添加到面板
        container.add(panel, BorderLayout.SOUTH); // 添加面板到窗体的下方
        this.setVisible(true); // 设置窗体可见
        this.setSize(350, 200); // 设置窗体大小
    }
    public static void main(String[] args) {
        new JPanelDemo ();
    }
}
```

程序运行结果如图8-3所示。

图 8-3　例 8-2 的运行结果

8.2.3　中间容器类（JScrollPane）

JScrollPane类也是一个中间容器，称为滚动面板。与JPanel类不同的是，JScrollPane类带有滚动条，而且只能向滚动面板添加一个组件。如果需要将多个组件放置到滚动面板，通常先将这些组件添加到一个JPanel面板，然后把这个JPanel面板添加到滚动面板。JScrollPane类常用的构造方法如下。

①JScrollPane()：创建一个空的JScrollPane类，需要时水平和垂直滚动条都可显示。

②JScrollPane(Component view)：创建一个显示指定组件内容的JScrollPane类，只要组件的内容超过视图大小就会显示水平和垂直滚动条。

③JScrollPane(int vsbPolicy,int hsbPolicy)：创建一个具有指定滚动条策略的空JScrollPane类。

常用的成员方法如下。

①public void setHorizontalScrollBarPolicy(int policy)：确定水平滚动条何时显示在滚动窗格上。选项如下。

```
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED 设置水平滚动条只在需要时显示，
默认策略
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER 水平滚动条永远不显示
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS 水平滚动条一直显示
```

②public void setVerticalScrollBarPolicy(int policy)：确定垂直滚动条何时显示在滚动窗格上。合法值如下。

```
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED 设置垂直滚动条只在需要时显示，
默认策略
ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER  垂直滚动条永远不显示
ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS 垂直滚动条一直显示
```

③public void setViewportView(Component view)：创建一个视口并设置其视图。不直接为JScrollPane类构造方法提供视图的应用程序应使用此方法，指定显示在滚动窗格的滚动组件子集。下面通过一个案例演示JScrollPane类的使用。

【例8-3】JScrollPane类的应用。

功能实现：添加5个按钮到JScrollPane容器中，并把JScrollPane容器添加到窗体的中间区域。当窗口的大小变化时，可以通过单击滚动条浏览被隐藏的组件。

```
import javax.swing.*;
import java.awt.*;
public class JScrollPaneDemo extends JFrame {
    JPanel p;
    JScrollPane scrollpane;
    private Container container;
    public JScrollPaneDemo () {
        this.setTitle("JScrollPane 演示实例 "); // 设置标题
        container = this.getContentPane();
        scrollpane = new JScrollPane(); // 创建 JScrollPane 类的对象
        // 设置水平滚动条的显示策略为“一直显示”
scrollpane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_
ALWAYS);
        p = new JPanel();                  // 创建面板
        p.add(new JButton(" 按钮 1"));       // 创建并添加命令按钮到面板
        p.add(new JButton(" 按钮 2"));
        p.add(new JButton(" 按钮 3"));
        p.add(new JButton(" 按钮 4"));
        p.add(new JButton(" 按钮 5"));
        scrollpane.setViewportView(p); // 设置滚动面板视图
        container.add(scrollpane);      // 把滚动面板添加到窗体中部
        this.setVisible(true);
```

```
        this.setSize(300, 200);
    }
    public static void main(String[] args) {
        new JScrollPaneDemo ();
    }
}
```

程序运行结果如图8-4所示。

图 8-4　例 8-3 的运行结果

8.3 布局管理器

布局管理器是一种用于控制组件在容器中排列和布局的工具。它可以根据容器的大小和组件的特性，自动调整组件的位置和大小，以实现灵活的界面布局。每个容器都有一个默认的布局管理器，开发者可以通过setLayout()方法改变容器的布局管理器。

Java提供多种布局管理器，每种布局管理器都有不同的特点和适用场景。本节介绍五种常用的布局管理器类，分别是FlowLayout（流式布局）、BorderLayout（边界布局）、GridLayout（网格布局）、CardLayout（卡片布局）和BoxLayOut（盒式布局）。

8.3.1 FlowLayout

FlowLayout（流式布局）类的布局策略是将容器看成一个行集，容器中的组件按照加入的先后顺序从左向右排列。当一行排满之后就转到下一行，行高由一行中最高的组件决定。FlowLayout类是所有JPanel的默认布局管理器。

FlowLayout类定义在java.awt包中，它有三种构造方法。

①FlowLayout()：创建一个使用居中对齐的FlowLayout类实例。

②FlowLayout(int align)：创建一个指定对齐方式的FlowLayout类实例。

③FlowLayout(int align，int hgap，int vgap)：创建一个既指定对齐方式又指定组件间隔的FlowLayout类的对象。

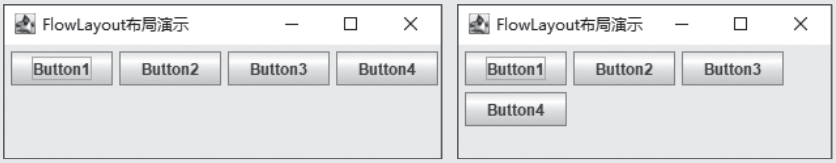
其中对齐方式align的可取值有FlowLayout.LEFT（左对齐）、FlowLayout.RIGHT（右对齐）、FlowLayout.CENTER（居中对齐）三种形式。例如new FlowLayout(FlowLayout. LEFT)就表示创建一个对齐方式为左对齐的FlowLayout类实例。下面通过一个案例演示FlowLayout类的使用。

【例8-4】FlowLayout布局管理器的使用。

功能实现：创建窗体，并以FlowLayout作为布局管理器，然后在窗体上放置4个命令按钮。

```
import javax.swing.*;
import java.awt.*;
public class FlowLayoutDemo extends JFrame {
    private JButton button1, button2, button3, button4; // 声明 4 个命令按钮对象
    public FlowLayoutDemo() {
        this.setTitle("FlowLayout 布局演示 "); // 设置标题
        Container container = this.getContentPane();// 获得 Container 对象
        // 设置为 FlowLayout 的布局，JFrame 默认的布局为 BorderLayout
        container.setLayout(new FlowLayout(FlowLayout.LEFT));
        // 创建一个标准命令按钮，按钮上的标签提示信息由构造方法中的参数指定
        button1 = new JButton("Button1") ;
        button2 = new JButton("Button2");
        button3 = new JButton("Button3");
        button4 = new JButton("Button4");
// 将组件添加到内容窗格，组件的大小和位置由 FlowLayout 布局管理器来控制
        container.add(button1);
        container.add(button2);
        container.add(button3);
        container.add(button4);
        this.setVisible(true);  // 使窗体显示出来
        this.setSize(300, 200); // 设置窗体大小
    }
    public static void main(String[] args) {
        new FlowLayoutDemo();
    }
}
```

注意事项 如果拖动窗口，并改变窗口大小，窗口中的组件位置会随之改变。程序运行结果如图8-5所示。



（a）按钮组件显示在同一行　　（b）按钮组件被放置到两行

图 8-5　例 8-4 的运行结果

8.3.2 BorderLayout

BorderLayout（边界布局）是顶层容器JFrame的默认布局管理器。它把容器被为东、西、南、北、中五个区域，这五个区域分别用字符串常量BorderLayout.EAST、BorderLayout.WEST、

BorderLayout.SOUTH、BorderLayout.NORTH、BorderLayout.CENTER表示。容器的每个区域只能放一个组件，每加入一个组件时都应该指明把这个组件放到哪个区域。

BorderLayout定义在java.awt包中，它有两种构造方法。

①BorderLayout()：创建一个各组件间水平、垂直间隔为0的BorderLayout实例。

②BorderLayout(int hgap, int vgap)：创建一个各组件间水平间隔为hgap、垂直间隔为vgap的BorderLayout实例。

在BorderLayout布局管理器的管理下，组件通过add( )方法加入容器中指定的区域。如果在add()方法中没有指定将组件放到哪个区域，那么它将会默认地被放置在Center区域。

在BorderLayout布局管理器的管理下，容器的每个区域只能加入一个组件。如果试图向某个区域加入多个组件，可以在这个区域放置一个中间容器JPanel或者JScrollPane组件，然后将所需的多个组件放到中间容器中，再把中间容器加入指定的区域，实现复杂的布局。示例代码如下。

```
JFrame f=new JFrame(“欢迎使用 BorderLayout 布局”);
JButton bt1=new JButton(“button1”);
JButton bt2=new JButton(“button2”);
JPanel p=new JPanel();
p.add(bt1);
p.add(bt2);
f.getContentPane().add(p, BorderLayout.SOUTH) ;
```

以上语句实现了将两个按钮bt1和bt2同时放到窗口的南部区域。

对于东、西、南、北四个边界区域，若某个区域没有被使用，则Center区域会扩展并占据这个区域的位置。如果四个边界区域都没有使用，那么Center区域将占据整个窗口。

下面通过一个案例演示BorderLayout布局管理器的使用方法。

【例8-5】BorderLayout布局管理器的使用。

功能实现：创建窗体并以BorderLayout的布局放置7个命令按钮。

```
import javax.swing.*;
import java.awt.*;
public class BorderLayoutDemo extends JFrame {
    // 声明 7 个命令按钮对象
    private JButton button1, button2, button3, button4, button5, button6, button7;
    public BorderLayoutDemo() {
        this.setTitle(" 欢迎使用 BorderLayout 布局 "); // 设置标题
        // 获取 Container 对象，并采用默认布局管理器 BorderLayout
        Container container = this.getContentPane();
        // 创建 7 个标准命令按钮，按钮上的标签由构造方法中的参数指定
        button1 = new JButton("ButtonA");
        button2 = new JButton("ButtonB");
        button3 = new JButton("ButtonC");
```

```
        button4 = new JButton("ButtonD");
        button5 = new JButton("ButtonE");
        button6 = new JButton("ButtonF");
        button7 = new JButton("ButtonG");
        JPanel p = new JPanel(); // 创建一个中间容器
        container.add(button1, BorderLayout.SOUTH); // button1 被放置到南部区域
        container.add(button2, BorderLayout.NORTH); // button2 被放置到北部区域
        container.add(button3, "East"); // button3 被放置到东部区域
        container.add("West", button4); // button4 被放置到西部区域
        p.add(button5);
        p.add(button6);
        p.add(button7); // 把 button5、 button6、 button7 放到中间容器中
        container.add(p); // 把中间容器放到中间区域中
        this.setVisible(true);
        this.setSize(600, 450);
    }
    public static void main(String[] args) {
        new BorderLayoutDemo();
    }
}
```

程序运行结果如图8-6所示。

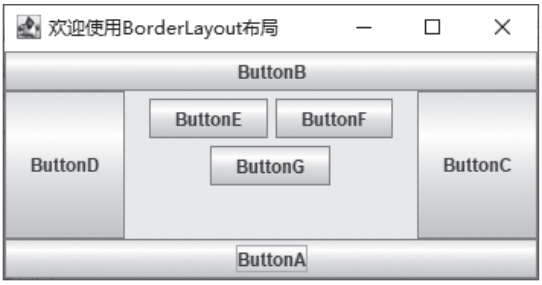


图 8-6 例 8-5 的运行结果

注意事项 按钮被放置到不同区域。如果改变窗口的大小，由于中间区域JPanel采用FlowLayout的布局管理，组件的布局会随之改变，其他区域不变。

8.3.3 GridLayout

如果界面上需要放置的组件比较多，且这些组件的大小又基本一致，如计算器、遥控器的面板，那么使用GridLayout布局管理器是最佳的选择。GridLayout是一种网格式的布局管理器。它将容器空间划分成若干行乘若干列的网格，而每个组件按添加的顺序从左到右、从上到下占据这些网格，每个组件占据一格。

GridLayout定义在java.awt包中，有三种构造方法，分别如下。

①GridLayout()：按默认（1行1列）方式创建一个GridLayout布局。

②GridLayout(int rows,int cols)：创建一个具有rows行、cols列的GridLayout布局。

③GridLayout(int rows,int cols,int hgap,int vgap)：按指定的行数rows、列数cols、水平间隔hgap和垂直间隔vgap创建一个GridLayout布局。

构造方法中的参数rows和cols两者有一个可以为0，但是不能同时为0。当容器增加组件时，容器自动向0的那个方向增长。

下面通过一个案例演示GridLayout布局管理器的使用方法。

【例8-6】GridLayout布局管理器的使用。

功能实现：创建窗体并以GridLayout的布局管理6个命令按钮。

```
import javax.swing.*;
import java.awt.*;
public class GridLayoutDemo extends JFrame {
    private JButton button1, button2, button3, button4, button5, button6;
    // 声明按钮对象
    public GridLayoutDemo() {
        this.setTitle(" 欢迎使用 GridLayout 布局管理器 "); // 设置标题
        Container container = this.getContentPane(); // 获得 Container 对象
        container.setLayout(new GridLayout(2, 3)); // 设置 2 行 3 列的布局管理器
        // 创建按钮对象，按钮上的标签由构造方法中的参数指定
        button1 = new JButton("ButtonA");
        button2 = new JButton("ButtonB");
        button3 = new JButton("ButtonC");
        button4 = new JButton("ButtonD");
        button5 = new JButton("ButtonE");
        button6 = new JButton("ButtonF");
        // 按放置的先后顺序，把命令按钮放置到内容窗格的不同区域
        container.add(button1);
        container.add(button2);
        container.add(button3);
        container.add(button4);
        container.add(button5);
        container.add(button6);
        this.setVisible(true);
        this.setSize(350, 200);
    }
    public static void main(String[] args) {
        new GridLayoutDemo();
    }
}
```

程序运行结果如图8-7所示。

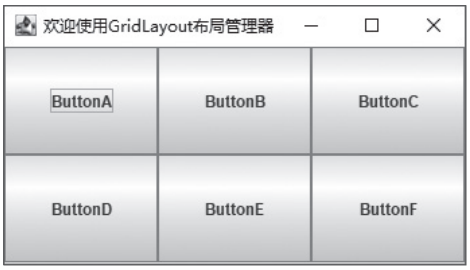


图 8-7 例 8-6 的运行结果

注意事项 组件放入容器中的次序决定了它占据的位置。当容器的大小发生改变时，GridLayout所管理的组件的相对位置不会发生变化，但组件的大小会随之变化。

8.3.4 CardLayout

CardLayout（卡片布局）位于java.awt包中。它将每个组件看成一张卡片，如同扑克牌一样将组件堆叠起来，但是只能看到最上面的一个组件，这个被显示的组件占据所有的容器空间。用户可以通过CardLayout类提供的方法切换空间中显示的卡片。例如，使用first（Container container）方法显示container中的第一个对象，last（Container container）显示container中的最后一个对象，next（Container container）显示下一个对象，previous（Container container）显示上一个对象。

CardLayout类有两个构造方法，分别是CardLayout( )和CardLayout（int hgap,int vgap）。前者使用默认（间隔为0）方式创建一个CardLayout类对象；后者创建指定水平间隔和垂直间隔的CardLayout类对象。下面通过一个案例演示CardLayout布局管理器的使用方法。

【例8-7】CardLayout布局管理器的使用。

功能实现：窗口使用CardLayout的布局管理，向其中加入3张卡片，每张卡片都是一个命令按钮对象。

```
import javax.swing.*;
import java.awt.*;
public class CardLayoutDemo extends JFrame {
    private JButton bt1, bt2, bt3;
    Container container;
    CardLayout myCard;
    public CardLayoutDemo() {
        this.setTitle(" 欢迎使用 CardLayout 布局管理器 ");
        container = this.getContentPane();
        myCard = new CardLayout();    // 创建 CardLayout 布局管理器对象
        container.setLayout(myCard); // 设置布局管理器
        // 创建 3 个 JButton 对象
        bt1 = new JButton("ButtonA");
        bt2 = new JButton("ButtonB");
        bt3 = new JButton("ButtonC");
```

```
        //将每个 JButton 对象作为一张卡片加入窗口
        container.add(bt1);
        container.add(bt2);
        container.add(bt3);
        this.setVisible(true);
        this.setSize(300, 200);
    }

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

程序执行结果如图8-8所示。

图 8-8　例 8-7 的运行结果

注意事项 在容器中只能看到最上面的一个组件。如果要切换显示其他组件，需要调用CardLayout提供的方法。

8.3.5 BoxLayout

BoxLayout位于javax.swing包中。它将容器中的组件按水平方向排成一行，或者垂直方向排成一列。当组件排成一行时，每个组件可以有不同的宽度；当排成一列时，每个组件可以有不同的高度。

BoxLayout类的常用构造方法是BoxLayout(Container target,int axis)，参数target是容器对象，表示要为哪个容器设置此布局管理器；axis指明target中组件的排列方式，其值包括表示水平排列的BoxLayout.X_AXIS和表示垂直排列的BoxLayout.Y_AXIS。下面通过一个案例演示BoxLayout布局管理器的使用方法。

【例8-8】BoxLayout布局管理器的使用。

功能实现：创建窗口，窗口使用BorderLayout的布局管理；在窗口中添加两个JPanel，二者的布局管理器分别采用水平排列和垂直排列的BoxLayout，再向这两个JPanel容器中分别加入三个命令按钮组件，并把这两个JPanel容器添加到窗口的北部和中部。

```
import javax.swing.*;
import java.awt.*;
```

```
public class BoxLayoutDemo extends JFrame {
    private JButton button1, button2, button3, button4, button5, button6;
    Container container;
    public BoxLayoutDemo() {
        this.setTitle(" 欢迎使用 BoxLayout 布局管理器 ");
        container = this.getContentPane(); // 获取 Container 对象
        container.setLayout(new BorderLayout()); // 设置布局
        // 声明中间容器 px 并设置布局为水平的 BoxLayout
        JPanel px = new JPanel();
        px.setLayout(new BoxLayout(px, BoxLayout.X_AXIS));
        // 创建命令按钮
        button1 = new JButton("ButtonA");
        button2 = new JButton("ButtonB");
        button3 = new JButton("ButtonC");
        // 把按钮放到中间容器 px 中
        px.add(button1);
        px.add(button2);
        px.add(button3);
        // 把中间容器 px 放到北部区域
        container.add(px, BorderLayout.NORTH);
        // 声明中间容器 py, 并设置布局为垂直的 BoxLayout
        JPanel py = new JPanel();
        py.setLayout(new BoxLayout(py, BoxLayout.Y_AXIS));
        button4 = new JButton("ButtonD");
        button5 = new JButton("ButtonE");
        button6 = new JButton("ButtonF");
        // 把按钮放到中间容器 py 中
        py.add(button4);
        py.add(button5);
        py.add(button6);
        // 把中间容器 py 放置到中间区域
        container.add(py, BorderLayout.CENTER);
        this.setVisible(true); // 显示窗口
        this.setSize(600, 450);// 设置窗口大小
    }
    public static void main(String[] args) {
        new BoxLayoutDemo();
    }
}
```

程序执行结果如图8-9所示。

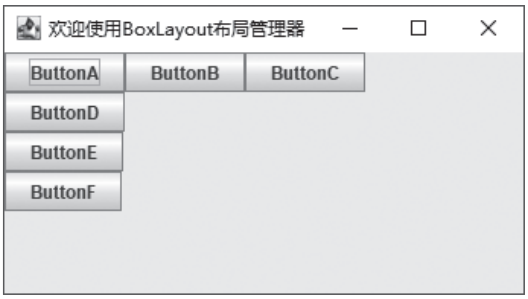


图 8-9　例 8-8 的运行结果

8.4 常用基本组件

在图形用户界面设计中使用最频繁的就是一些常用的基本组件，例如标签、按钮、文本框等组件。本节对这些常用的基本组件进行简单介绍。

8.4.1 标签（JLabel）

JLable组件被称为标签。它是一个静态组件，也是基本组件中最简单的一种组件。每个标签用一个标签类的对象表示，可以显示一行静态文本和图标。标签只起信息说明的作用，而不接收用户的输入，也无事件响应。其常用构造方法如下。

①JLabel()：构造一个既不显示文本信息也不显示图标的空标签。

②Label(String text)：构造一个显示文本信息的标签。

③JLabel(String text, int horizontalAlignment)：构造一个显示文本信息的标签。

④JLabel(String text, Icon icon, int horizontalAlignment)：构造一个同时显示文本信息和图标的标签。

构造方法中的参数text代表标签的文本提示信息，Icon icon代表标签的显示图标，int horizontalAlignment代表水平对齐方式。水平对齐方式的取值可以是JLabel .LEFT、JLabel .CENTER等常量，默认情况下标签的内容居中显示。

创建完标签对象，可以通过成员方法setHorizontalAlignment(int alignment)更改对齐方式。通过getIcon()和setIcon(Icon icon)方法获取标签的图标和修改标签上的图标。通过getText()和setText(String text)方法获取标签的文本提示信息和修改标签的文本内容。下面通过一个案例演示JLable的使用方法。

【例8-9】JLable的使用。

功能实现：创建窗口并在窗口添加两个JLable，一个仅显示文本信息，另一个既显示文本信息又显示图标。

```
import javax.swing.*;
import java.awt.*;
public class JLableDemo extends JFrame {
    private JLabel lb1,lb2;
```

```
    public JLableDemo() {
        this.setTitle("JLable 示例"); // 设置标题
        Container container = this.getContentPane();// 获得 Container 对象
        // 容器布局设置为 FlowLayout 布局
        container.setLayout(new FlowLayout());
        // 创建两个标签
        lb1 = new JLabel(" 第一个标签 "); // 只有文本信息
        // 既有文本信息又有图标
        lb2 = new JLabel(" 第二个标签 ",new ImageIcon("save.png"),JLabel.LEFT);
        // 将标签添加到容器
        container.add(lb1);
        container.add(lb2);
        this.setVisible(true);  // 使窗体显示出来
        this.setSize(300, 200); // 设置窗体大小
    }
    public static void main(String[] args) {
        new JLableDemo();
    }
}
```

程序执行结果如图8-10所示。

图 8-10　例 8-9 的运行结果

8.4.2　文本组件

文本组件是用于显示信息和提供用户输入文本信息的主要工具，在Swing包中提供了文本框（JTextField）、文本域（JTextArea）、口令输入域（JPasswordField）等多种文本组件。它们都有一个共同的基类JTextComponent。在JTextComponent类中定义的主要方法如表8-1所示，主要实现对文本进行选择、编辑等操作。

表8-1　JTextComponent类常用成员方法

成员方法	功能说明
getText()	从文本组件中提取所有文本内容
getText(int offs, int len)	从文本组件中提取指定范围的文本内容
getSelectedText()	从文本组件中提取被选中的文本内容
selectAll()	在文本组件中选中所有文本内容

（续表）

成员方法	功能说明
setEditable(boolean b)	设置为可编辑或不可编辑状态
setText(String t)	设置文本组件中的文本内容
replaceSelection(String content)	用给定字符串所表示的新内容替换当前选定的内容

1. JTextField

JTextField被称为文本框。它是一个单行文本输入框，可以输出任何基于文本的信息，也可以接收用户输入的信息。

（1）JTextField常用的构造方法

①JTextField()：创建一个空的文本框，一般作为输入框。

②JTextField(int columns)：构造一个具有指定列数的空文本框，一般用于显示长度或者输入字符的长度受到限制的情况下。

③JTextField(String text)：构造一个显示指定字符的文本框，一般作为输出框。

④JTextField(String text, int columns)：构造一个具有指定列数，并显示指定初始字符串的文本域 。

（2）JTextField组件常用的成员方法

①setFont(Font f)：设置字体。

②setActionCommand(String com)：设置动作事件使用的命令字符串。

③setHorizontalAlignment(int alig)：设置文本的水平对齐方式。

下面通过一个案例演示JTextField的使用方法。

【例8-10】JTextField的使用。

功能实现：创建窗口并在窗口添加一个JLable和一个JTextField，JLable用于显示提示信息，JTextField用于接收用户输入的信息。

```
import java.awt.*;
import javax.swing.*;
public class JTextFieldDemo extends JFrame{
    private JLabel lb1;
    private JTextField t1;
    private Container container;
    public JTextFieldDemo() {
        this.setTitle("JTextField 示例 ");// 设置窗体标题
        container = this.getContentPane(); // 获取 Container 对象
        container.setLayout(new FlowLayout()); // 设置容器布局管理
        lb1 = new JLabel(" 请输入一个整数 ");// 创建标签对象，字符串为提示信息
        t1 = new JTextField(10); // 创建输入文本框，最多显示 10 个字符
        // 将组件添加到窗口
        container.add(lb1);
```

```
        container.add(t1);
        this.setSize(300, 100);// 设置窗口大小
        this.setVisible(true);// 设置窗体的可见性
    }
    public static void main(String[] arg) {
        new JTextFieldDemo();
    }
}
```

程序执行结果如图8-11所示。

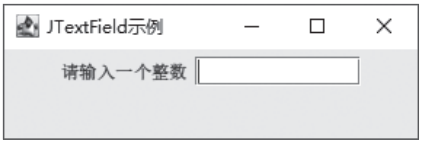


图 8-11　例 8-10 的运行结果

2. JTextArea

JTextArea被称为文本域。它与文本框的主要区别是文本框只能输入/输出一行文本，而文本域可以输入/输出多行文本。JTextArea本身不带滚动条，构造对象时可以设定区域的行、列数。由于文本域通常显示的内容比较多，超出指定的范围不方便浏览，因此一般将其放入滚动窗格JScrollPane中。

（1）常用的构造方法

①JTextArea()：构造一个空的文本域。

②JTextArea(String text)：构造显示初始字符串信息的文本域。

③JTextArea(int rows, int columns)：构造具有指定行和列的空的文本域。

④JTextArea(String text,int rows,int columns)：构造具有指定文本、行和列的文本域。

（2）JTextArea组件常用的成员方法

①insert(String str, int pos)：将指定文本插入指定位置。

②append(String str)：将给定文本追加到文档结尾。

③replaceRange(String str,int start,int end)：用给定的新文本替换从指示的起始位置到结尾位置的文本。

④setLineWrap(boolean wrap)：设置文本域是否自动换行，默认为false。

3. JPasswordField

JPasswordField组件实现一个密码框，用来接收用户输入的单行文本信息。在密码框中不显示用户输入的真实信息，而是通过显示一个指定的回显字符作为占位符。新创建密码框的默认回显字符为“*”，可以调用方法进行修改。

（1）JPasswordField的常用构造方法

①JPasswordField()：构造一个空的密码框。

②JPasswordField(String text)：构造一个显示初始字符串信息的密码框。

③JPasswordField(int columns)：构造一个具有指定长度的空密码框。

（2）JPasswordField的常用成员方法

①setEchoChar(char c)：设置密码框的回显字符。

②char[] getPassword()：返回此密码框中所包含的文本。

③char getEchoChar()：获得密码框的回显字符。

例如下面代码片段，判断输入密码框中的密码是否与给定密码相等。

```
JPasswordField pwf=new JPasswordField(6); // 可以接收 6 个字符的密码框
pwf.setEchoChar('*'); // 设置回显字符
getContentPane().add(lb1);
getContentPane().add(pwf); // 添加到内容窗格中
 ……
    char[] psword=pwf.getPassword(); // 得到密码框中输入的文本
    String s=new String(psword); // 把字符数组转换为字符串
    if(s.equals("123456")) // 比较字符串的值是否相等
        System.out.println(" 密码正确！ ");
```

8.4.3 按钮组件

按钮是图形用户界面最常用、最基本组件，经常用到的按钮有JButton、JCheckBox、JRadioButton等，这些按钮类均是AbstractButton类的子类或者间接子类。AbstractButton中定义了各种按钮所共有的一些方法。AbstractButton类常用的成员方法有以下几个。

①Icon getIcon()和setIcon(Icon icon)：获得和修改按钮图标。

②String getText()和setText(String text)：获取和修改按钮文本信息。

③setEnabled(boolean b)：启用或禁用按钮。

④setHorizontalAlignment(int alignment)：设置图标和文本的水平对齐方式。

按钮类之间的继承关系如图8-12所示，下面分别对几个常用的按钮类进行简单介绍。

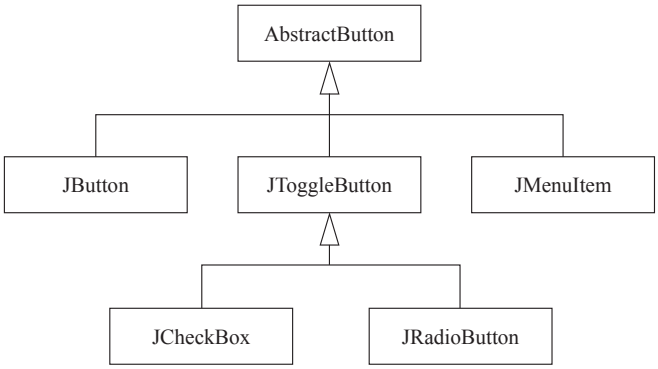


图 8-12　按钮组件类之间的继承关系图

1. JButton

JButton是最常用、最简单的按钮，可分为有无标签和图标几种情况。

JButton类常用的构造方法如下。

①JButton()：创建一个无文本也无图标的按钮。

②JButton(String text)：创建一个具有文本提示信息但没有图标的按钮。

③JButton(Icon icon)：创建一个具有图标、但没有文本提示信息的按钮。

④JButton(String text，Icon icon)：创建一个既有文本提示信息又有图标的按钮。

创建按钮对象的示例代码如下。

```
JButton bt=new JButton("保存",new ImageIcon("save.png"));
```

2. JCheckBox

JCheckBox组件被称为复选框。它提供选中/未选中两种状态，并且可以同时选定多个。用户单击复选框就会改变该复选框原来的状态。

JCheckBox组件类的常用构造方法如下。

①JCheckBox()：构造一个无标签的复选框。

②JCheckBox(String text)：构造一个具有提示信息的复选框。

③JCheckBox(String text,boolean selected)：创建具有文本的复选框，并指定其最初是否处于选定状态。

创建复选框组件对象，可以通过JCheckBox类提供的成员方法设定复选框的属性。如通过setText(String text)方法设定文本提示信息，通过setSelected(boolean b)方法设定复选框的状态，通过isSelected()方法获取按钮当前的状态。

3. JRadioButton

JRadioButton组件被称为选项按钮。在Java中，JRadioButton组件与JCheckBox组件功能完全一样，只是图形不同，复选框为方形图标，选项按钮为圆形图标。

如果要实现多选一的功能，需要利用javax.swing.ButtonGroup类实现。这个类是一个不可见的组件，表示一组单选按钮之间互斥的逻辑关系，实现诸如JRadioButton等组件的多选一功能。

下面通过一个案例演示这几种按钮类的使用方法。

【例8-11】按钮类的使用。

功能实现：创建窗口并在窗口分别添加JButton、JCheckButton和JRadioButton三种不同的按钮。

```
import java.awt.*;
import javax.swing.*;
public class ButtonDemo extends JFrame{
    public ButtonDemo() {
        this.setTitle("三种按钮使用示例");
        //设置布局管理器
        this.setLayout(new FlowLayout());
        //创建3个面板对象
        JPanel p1 = new JPanel();
```

```
        JPanel p2 = new JPanel();
        JPanel p3 = new JPanel();
        //创建3个复选框对象
        JCheckBox cb1 = new JCheckBox("复选框1");
        JCheckBox cb2 = new JCheckBox("复选框2");
        JCheckBox cb3 = new JCheckBox("复选框3");
        //添加组件到面板p1
        p1.add(cb1);
        p1.add(cb2);
        p1.add(cb3);
        //创建3个单选钮对象
        JRadioButton rb1 = new JRadioButton("单选钮1");
        JRadioButton rb2 = new JRadioButton("单选钮2");
        JRadioButton rb3 = new JRadioButton("单选钮3");
        //创建按钮组对象
        ButtonGroup gp1 = new ButtonGroup();
        //把单选钮对象添加到按钮组中，实现单选钮的多选一功能
        gp1.add(rb1);
        gp1.add(rb2);
        gp1.add(rb3);
        //把单选钮添加到面板p2
        p2.add(rb1);
        p2.add(rb2);
        p2.add(rb3);
        //创建两个普通按钮对象
        JButton bt1 = new JButton("按钮1");
        JButton bt2 = new JButton("按钮2");
        //添加组件到面板p3
        p3.add(bt1);
        p3.add(bt2);
        //把面板添加到窗体
        this.add(p1);
        this.add(p2);
        this.add(p3);
        this.setSize(300, 200);
        this.setVisible(true);
    }
    public static void main(String args[]) {
        ButtonDemo tsb = new ButtonDemo ();

    }
}
```

程序执行结果如图8-13所示。

图 8-13 例 8-11 的运行结果

8.4.4 下拉列表框（JComboBox）

JComboBox被称为组合框或者下拉列表框。用户可以从下拉列表中选择相应的选项作为自己的选择，但只能选择一个选项。

JComboBox有两种形式：不可编辑的和可编辑的。对不可编辑的JComboBox，用户只能在现有的选项中进行选择；而可编辑的JComboBox，用户既可以在现有选项中选择，也可以输入新的内容作为自己的选择。

（1）JComboBox常用的构造方法

①JComboBox()：创建一个没有任何可选项的组合框。

②JCombBox(Object[ ] items)：根据Object数组创建组合框，Object数组的元素即为组合框中的可选项。

例如，创建一个具有3个可选项的下拉列表框，核心代码如下。

```
String contentList={“学士”, “硕士”, “博士”};
JComboBox jcb=new JComboBox(contentList);
```

创建下拉列表框对象后，可以通过该类的成员方法对其属性进行查询或修改。

（2）JComboBox类常用成员方法

①void addItem(Object anObject)：为选项列表添加选项。

②Object getItemAt(int index)：返回指定索引处的列表项。

③int getItemCount()：返回列表中的项数。

④int getSelectedIndex()：返回列表中与给定项匹配的第一个选项。

⑤Object getSelectedItem()：返回当前所选项。

⑥void removeAllItems()：从项列表中移除所有选项。

⑦removeItem(Object anObject)：从项列表中移除指定的选项。

⑧removeItemAt(int anIndex)：移除指定位置anIndex处的选项。

⑨setEditable(boolean aFlag)：设置JComboBox是否可编辑。

下面通过一个案例演示JComboBox的使用方法。

【例8-12】JComboBox的使用。

功能实现：创建窗口，并在窗口添加一个下拉列表框对象，供用户选择学历。

```
import java.awt.*;
import javax.swing.*;
public class JComboBoxDemo extends JFrame{
    public JComboBoxDemo() {
        this.setTitle("JcomboBox使用示例");
        //设置布局管理器
        this.setLayout(new FlowLayout());
        //创建标签板对象
        JLabel lb1 = new JLabel("学历");
        //准备加入下拉列表框中的选项
        String[] s1={"初中","高中","大专","本科", "研究生"};
        //创建下拉列表框对象
        JComboBox jcb1=new JComboBox(s1);
        //把组件添加到窗体
        this.add(lb1);
        this.add(jcb1);

        this.setSize(300, 200);
        this.setVisible(true);
    }

    public static void main(String args[]) {
        JComboBoxDemo tsb = new JComboBoxDemo ();

    }
}
```

程序执行结果如图8-14所示。

图 8-14　例 8-12 的运行结果

8.4.5　列表框（JList）

JList又称为列表框。它会显示一组选项供用户选择，用户可以从中选择一个或多个选项。JList组件与JComboBox组件的最大区别是JComboBox组件一次只能选择一项，而JList组件一次可以选择一项或多项。选择多项时可以是连续区间选择（按住Shift键进行选择），也可以是不连

续选择(按住Ctrl键进行选择)。

（1）JList常用的构造方法

①JList()：构造一个空列表。

②JList(Object[] listData)：构造一个列表，列表的可选项由对象数组listData指定。

③JList(Vector listData)：构造一个列表，列表的可选项由Vector型参数dataModel指定。

（2）JList类常用的成员方法

①int getSelectedIndex( )：返回所选的第一个索引；如果没有选择项，则返回-1。

②void setSelectionBackground(Color c)：设置所选单元的背景色。

③void setSelection Foreground(Color c)：设置所选单元的前景色。

④void setVisibleRowCount(int num)：设置不使用滚动条可以在列表中显示的首选行数。

⑤void setSelectionMode(int selectionMode)：确定允许单项选择还是多项选择。

⑥void setListData(Object[] listData)：根据一个object数组构造列表。

下面通过一个案例演示JList的使用方法。

【例8-13】JList的使用。

功能实现：创建窗口，并在窗口添加一个列表框，供用户选择个人爱好，用户可以选择多个选项。

```
import java.awt.*;
import javax.swing.*;
public class JListDemo extends JFrame{
    public JListDemo() {
        this.setTitle("JList使用示例");
        this.setLayout(null); // 不使用布局管理器
        JLabel lb1 = new JLabel("个人爱好"); // 创建标签板对象
        lb1.setBounds(10, 10, 60, 15);     // 设置标签位置和大小
        // 准备加入列表框中的选项
        String[] s1={"读书","跑步","游泳","滑雪", "举重", "购物", "上网"};
        JList list1=new JList(s1);// 创建列表框对象
        JScrollPane sp1=new JScrollPane();// 创建滚动面板
        sp1.setViewportView(list1);     // 设置滚动面板视口
        sp1.setBounds(70, 5, 70, 100); // 设置面板位置和大小
        // 把组件添加到窗体
        this.getContentPane().add(lb1);
        this.getContentPane().add(sp1);
        this.setSize(300, 200);
        this.setVisible(true);
    }
    public static void main(String args[]) {
        JListDemo tsb = new JListDemo ();
    }
}
```

程序执行结果如图8-15所示。

图 8-15　例 8-13 的运行结果

8.5 Java的GUI事件处理

设计和实现图形用户界面的工作主要有两个：一是创建组成界面的各种元素，并指定它们的属性和位置关系，形成完整的图形用户界面的物理外观；二是定义图形用户界面的事件和各界面元素对不同事件的响应，从而实现图形用户界面与用户的交互功能。图形用户界面的事件驱动机制，可根据产生的事件来决定执行相应的程序段。

8.5.1 事件处理的基本过程

Java采用委托事件模型来处理事件。委托事件模型的特点是将事件的处理委托给独立的对象，而不是组件本身，从而将用户界面与程序逻辑分开。整个“委托事件模型”由产生事件的对象（事件源）、事件对象及监听者对象之间的关系所组成。

每当用户在组件上进行某种操作时，事件处理系统会将与该事件相关的信息封装在一个“事件对象”中。例如，单击命令按钮时会生成一个代表此事件的ActionEvent事件类对象。用户操作不同，事件类对象也会不同。然后将该事件对象传递给监听者对象。监听者对象根据该事件对象内的信息确定适当的处理方式。每类事件对应一个监听程序接口，规定接收并处理该类事件的方法的规范。如ActionEvent事件对应ActionListener接口。该接口只有一个方法，即actionPerformed()，当出现ActionEvent事件时，该方法会被调用。

为了接收并处理某类用户事件，必须在程序代码中向产生事件的对象注册相应的事件处理程序，即事件的监听程序（Listener）。它是实现了对应监听程序接口的一个类。当事件产生时，产生事件的对象主动通知监听者对象，监听者对象根据产生该事件的对象来决定处理事件的方法。例如，为了处理命令按钮上的ActionEvent事件，需要定义一个实现ActionListener接口的监听程序类。每个组件都有若干个形如add×××Listener(×××Listener listener)的方法。通过这类方法，可以为组件注册事件监听程序。例如在JButton类中的方法：public void addAciton Listener(AcitonListener listener)，该方法可以为JButton组件注册ActionEvent事件监听程序，方法的参数是一个实现了ActionListener接口的类的实例。Java的事件处理过程如图8-16所示。

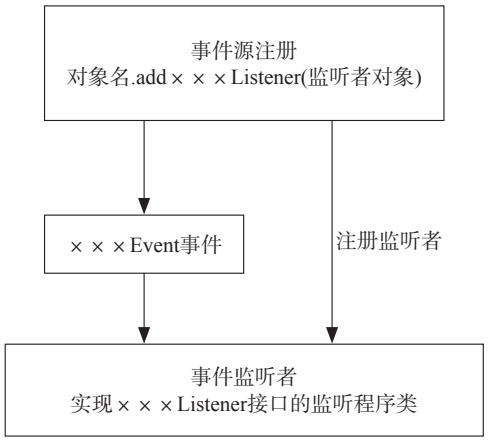


图 8-16 事件处理模型示意图

下面通过一些具体的案例演示ActionEvent事件的处理过程。

【例8-14】ActionEvent事件的处理过程。

功能实现：使用内部类作为事件监听器类，监听ActionEvent事件并进行处理。

创建包含一个命令按钮的窗口，并为该命令按钮注册一个内部类ButtonEventHandle对象，作为ActionEvent事件的监听器对象。内部类ButtonEventHandle实现了ActionEvent事件对应的ActionListener接口。在该类的actionPerformed方法中给出了处理ActionEvent事件的代码。当单击命令按钮时，ActionEvent事件被触发，该方法中的代码被执行。

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*; // ActionListener 接口和事件类处于 event 包中，需导入该包
public class ActionEventDemo1 extends JFrame {
    private JButton button1;
    private Container container;
    public TestEvent() {
        this.setTitle(" 事件处理演示程序 ");
        container = this.getContentPane();
        container.setLayout(new FlowLayout());
        // 创建标准命令按钮，按钮上的标签由构造方法中的参数指定
        button1 = new JButton(" 测试事件 ");
        // button1 为事件源，为事件注册监听者，监听者必须实现该事件对应的接口
        button1.addActionListener(new ButtonEventHandle());
        container.add(button1); // 把命令按钮添加到内容窗格
        this.setVisible(true);
        this.setSize(300, 100);
    }
    // 该类为内部类，作为事件监听程序类，该类必须实现事件对应的接口
    class ButtonEventHandle implements ActionListener {
        // 当触发 ActionEvent 事件时，执行 actionPerformed() 方法
```

```
        public void actionPerformed(ActionEvent e) {
            System.out.println("命令按钮被单击");
        }
    }
    public static void main(String[] args) {
        new ActionEventDemo1();
    }
}
```

程序执行结果如图8-17所示。

图 8-17　例 8-14 的运行结果

当单击命令按钮时，控制台将显示字符串"命令按钮被单击"。

本例中的事件监听器类被定义为一个内部类ButtonEventHandle，也可以使用组件所在的类作为监听器类，方法就是让组件所在的类直接实现监听器接口即可。

【例8-15】使用组件所在的类作为事件监听器类。

功能实现：创建一个包含一个命令按钮和一个文本框的窗口。当单击命令按钮时，把文本框的内容显示到控制台。由于要使用组件所在的类作为监听器类，该类必须实现ActionListener接口。

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class ActionEventDemo2 extends JFrame implements ActionListener{
    //组件所在类作为事件监听器类，该类必须实现对应的 ActionListener 接口
    private JTextField textField1; // 文本框
    private JButton button1;    //按钮
    private Container container;
    public ActionEventDemo2 () {
        this.setTitle("事件处理演示程序 2");
        container = this.getContentPane();
        container.setLayout(new FlowLayout());
        //创建文本框对象
        textField1 = new JTextField(20);
        //创建命令按钮对象
        button1 = new JButton("确定");
        //注册监听器对象
        button1.addActionListener(this);
```

```
        //在窗口上添加组件
        container.add(textField1);
        container.add(button1);
        //设置窗口可见状态和大小
        this.setVisible(true);
        this.setSize(360, 150);
    }
    //实现 ActionListener 接口中的方法
    public void actionPerformed(ActionEvent e) {
        //获取文本框的内容
        String s1 = textField1.getText();
        //输出到控制台
        System.out.println(s1);
    }
    public static void main(String[] args) {
        new ActionEventDemo2 ();
    }
}
```

程序执行结果如图8-18所示。

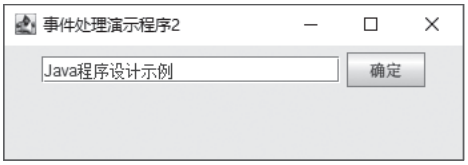


图 8-18 例 8-15 的运行结果

当用户单击图8-18中的“确定”按钮时，控制台将输出文本框中的内容“Java程序设计示例”。

也可以用匿名内部类对象作为事件监听器对象，具体示例如例8-16所示。

【例8-16】使用匿名内部类作为事件监听类。

功能实现：使用匿名内部类作为事件监听类。创建一个包含两个文本框的窗口，用户在第一个文本框中输入一个正整数，然后按回车键。程序自动计算该数的阶乘值，并把计算结果显示到第二个文本框。

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class ActionEventDemo3 extends JFrame{
    //组件所在类作为事件监听器类，该类必须实现对应的 ActionListener 接口
    private JTextField textField1; //文本框 1
    private JTextField textField2; //文本框 2
    private Container container;
```

```
    public ActionEventDemo3 () {
        this.setTitle("事件处理演示程序 3");
        container = this.getContentPane();
        container.setLayout(new FlowLayout());
        //创建标签用于显示提示信息
        JLabel lb1 = new JLabel("输入一个正整数 :");
        JLabel lb2 = new JLabel("该数的阶乘值为 :");
        //创建文本框对象
        textField1 = new JTextField(20);
        textField2 = new JTextField(20);
        //注册监听器对象，该对象为匿名内部类对象
        textField1.addActionListener(new ActionListener(){
            //匿名内部作为事件监听器类
            //该类必须实现事件对应的 ActionListener 接口中的方法
            public void actionPerformed(ActionEvent e) {
                //获取文本框 1 中的内容
                String s1 = textField1.getText();
                //把字符串转化为整数
                int n = Integer.parseInt(s1);
                //计算阶乘
                long f=1;
                for(int i=1;i<=n;i++) {
                    f *= i;
                }
                //把整数转化为字符串
                String s2 = String.valueOf(f);
                //在文本框 2 中显示计算结果
                textField2.setText(s2);
            }
        });

        //在窗口上添加组件
        container.add(lb1);
        container.add(textField1);
        container.add(lb2);
        container.add(textField2);
        //设置窗口属性
        this.setVisible(true);
        this.setSize(360, 150);
    }

    public static void main(String[] args) {
```

```
        new ActionEventDemo3 ();
    }
}
```

程序执行结果如图8-19所示。

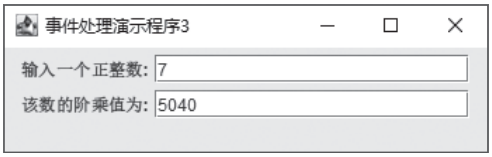


图 8-19　例 8-16 的运行结果

大部分事件监听器只是临时使用一次，所以使用匿名内部类形式的事件监听器更合适。实际上，这种形式是目前是最广泛的事件监听器形式。上面的程序代码就是使用匿名内部类创建事件监听器。使用匿名内部类作为监听器，唯一的缺点就是匿名内部类的语法不易掌握。如果读者的Java语言基本功扎实，对匿名内部类的语法掌握得较好，通常建议使用匿名内部类作为监听器。

8.5.2　常用的事件类及其监听器类

前面介绍了图形用户界面中事件处理的一般机制，其中只涉及ActionEvent事件类。由于不同事件源上发生的事件种类不同，不同的事件有不同的监听器处理。所以在java.awt.event包和javax.swing.event包中还定义了很多其他事件类。每个事件类都有一个对应的监听器接口，监听器接口中声明了若干个抽象的事件处理方法。事件的监听器类需要实现相应的监听器接口。

1. AWT 中的常用事件类及其监听器接口

java.util.EventObject类是所有事件对象的基础父类，所有事件都是由它派生出来的。AWT的相关事件继承于java.awt.AWTEvent类，这些AWT事件分为两类：低级事件和高级事件。低级事件是指基于组件和容器的事件，如鼠标的进入、单击、拖放，或组件的窗口开关等。低级事件主要包括ComponentEvent、ContainerEvent、WindowEvent、FocusEvent、KeyEvent、MouseEvent等。

高级事件是基于语义的事件。它可以不和特定的动作相关联，而依赖于触发此事件的类，如在JTextField中按Enter键会触发ActionEvent事件，滑动滚动条会触发AdjustmentEvent事件，或是选中项目列表的某一条就会触发ItemEvent事件。高级事件主要包括ActionEvent、AdjustmentEvent、ItemEvent、TextEvent等。

表8-2中是常用的AWT事件类及相应的监听器接口，共10类事件，11个接口。

表8-2　常用的AWT事件及其相应的监听器接口

事件类别	描述信息	接口名	方法
ActionEvent	激活组件	ActionListener	actionPerformed(ActionEvent e)
ItemEvent	选择了某些项目	ItemListener	itemStateChanged(ItemEvent e)

（续表）

事件类别	描述信息	接口名	方法
MouseEvent	鼠标移动	MouseMotionListener	mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)
	鼠标单击等	MouseListener	mousePressed(MouseEvent e) mouseReleased(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mouseClicked(MouseEvent e)
KeyEvent	键盘输入	KeyListener	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)
FocusEvent	组件收到或失去焦点	FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
AdjustmentEvent	移动滚动条等组件	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)
ComponentEvent	对象移动、缩放、显示、隐藏等	ComponentListener	componentMoved(ComponentEvent e) componentHidden(ComponentEvent e) componentResized(ComponentEvent e) componentShown(ComponentEvent e)
WindowEvent	窗口收到窗口级事件	WindowListener	windowClosing(WindowEvent e) windowOpened(WindowEvent e) windowIconified(WindowEvent e) windowDeiconified(WindowEvent e) windowClosed(WindowEvent e) windowActivated(WindowEvent e) windowDeactivated(WindowEvent e)
ContainerEvent	容器中增加、删除组件	ContainerListener	componentAdded(ContainerEvent e) componentRemoved(ContainerEvent e)
TextEvent	文本字段或文本区发生改变	TextListener	textValueChanged(TextEvent e)

2. Swing 中的常用事件类及其监听器接口

在javax.swing.event包中也定义了一些事件类，包括AncestorEvent、CaretEvent、DocumentEvent等。表8-3中列出常用的Swing事件类及其相应的监听器接口。

表8-3　常用的Swing事件类及其相应的监听器接口

事件类别	描述信息	接口名	方法
AncestorEvent	报告给子组件	AncestorListener	ancestorAdded(AncestorEvent event) ancestorRemoved(AncestorEvent event) ancestorMoved(AncestorEvent event)

（续表）

事件类别	描述信息	接口名	方法
CaretEvent	文本插入符已发生更改	CaretListener	caretUpdate(CaretEvent e)
ChangeEvent	事件源状态发生更改	ChangeListener	stateChanged(ChangeEvent e)
DocumentEvent	文档更改	DocumentListener	insertUpdate(DocumentEvent e) removeUpdate(DocumentEvent e) changedUpdate(DocumentEvent e)
UndoableEditEvent	撤销操作	UndoableEditListener	undoableEditHappened(UndoableEditEvent e)
ListSelectionEvent	选择值发生更改	ListSelectionListener	valueChanged(ListSelectionEvent e)
ListDataEvent	列表内容更改	ListDataListener	intervalAdded(ListDataEvent e) contentsChanged(ListDataEvent e) intervalRemoved(ListDataEvent e)
TableModelEvent	表模型发生更改	TableModelListener	tableChanged(TableModelEvent e)
MenuEvent	菜单事件	MenuListener	menuSelected(MenuEvent e) menuDeselected(MenuEvent e) menuCanceled(MenuEvent e)
TreeExpansionEvent	树扩展或折叠某一节点	TreeExpansionListener	treeExpanded(TreeExpansionEvent event) tree Collapsed(TreeExpansionEvent event)
TreeModelEvent	树模型更改	TreeModelListener	treeNodesChanged(TreeModelEvent e) treeNodesInserted(TreeModelEvent e) treeNodesRemoved(TreeModelEvent e) treeStructureChanged(TreeModelEvent e)
TreeSelectionEvent	树模型选择发生更改	TreeSelectionListener	valueChanged(TreeSelectionEvent e)

所有的事件类都继承自EventObject类。在该类中定义了一个重要的方法getSource()。该方法的功能是从事件对象获取触发该事件的事件源，为编写事件处理的代码提供方便。该方法的接口为public Object getSource()，无论事件源是何种具体类型，返回的都是Object类型的引用。开发人员需要自己编写代码进行引用的强制类型转换。

AWT的组件类和Swing组件类提供注册和注销监听器的方法。注册监听器的方法为public void add×××Listener (<ListenerType> listener)，如果不需要对该事件监听处理，可以把事件源的监听器注销，public void remove×××Listener (<ListenerType> listener)。

8.6 多监听程序与事件适配器

为了实现事件的处理需要实现对应监听器接口，而在接口中往往包含很多抽象方法，为了实现接口就需要实现接口中所有的抽象方法。然而在很多情况下，用户往往只关心其中的某一个或者某几个方法，为了简化编程可以考虑使用适配器（Adapter）类。

8.6.1 窗口事件的处理

大部分GUI应用程序都需要使用窗体作为最外层的容器。可以说窗体是组建GUI应用程序的基础，应用中需要使用的其他控件都是直接或间接放在窗体中的。

如果窗体关闭时需要执行自定义的代码，可以利用窗口事件WindowEvent对窗体进行操作，包括关闭窗体、窗体失去焦点、获得焦点、最小化等。WindowsEvent类包含的窗口事件如表8.1所示。

WindowEvent类的主要方法有getWindow()和getSource()。这两个方法的区别：getWindow()方法返回引发当前WindowEvent事件的具体窗口，返回值是具体的Window对象；getSource()方法返回相同的事件引用，其返回值的类型为Object。下面通过一个案例说明窗口事件的使用。

【例8-17】窗口事件的使用。

功能实现：创建两个窗口，对窗口事件进行测试。

```
import java.awt.*;
import javax.swing.*;
import javax.swing.JFrame;
import java.awt.event.*;
public class windowEventDemo {
    JFrame f1, f2;
    public static void main(String[] arg) {
        new windowEventDemo();
    }
    public windowEventDemo() {
        // 创建两个 JFrame 对象
        f1 = new JFrame(" 第一个窗口事件测试 ");
        f2 = new JFrame(" 第二个窗口事件测试 ");
        Container cp = f1.getContentPane();
        f1.setSize(300, 200); // 设置窗口大小
        f1.show(); // 设置窗口为可见
        f2.show();
        // 注册窗口事件监听程序，两个事件源自同一个监听者，WinLis 为内部类
        f1.addWindowListener(new WinLis());
        f2.addWindowListener(new WinLis());
    }
```

```
    class WinLis implements WindowListener{
        public void windowOpened(WindowEvent e) {
            // 窗口打开时调用
            System.out.println(" 窗口被打开 ");
        }
        public void windowActivated(WindowEvent e) {
            // 该方法暂时不用，代码为空
        }
        public void windowDeactivated(WindowEvent e) {
            // 将窗口设置成非活动窗口
            if (e.getSource() == f1)
                System.out.println(" 第一个窗口失去焦点 ");
            else
                System.out.println(" 第二个窗口失去焦点 ");
            }
            public void windowClosing(WindowEvent e) {
                // 把退出窗口时要执行的语句写在本方法中
                System.exit(0);
            }
            public void windowIconified(WindowEvent e) { // 窗口图标化时调用
                if (e.getSource() == f1)
                    System.out.println(" 第一个窗口被最小化 ");
                else
                System.out.println(" 第二个窗口被最小化 ");
            }
            public void windowDeiconified(WindowEvent e) {
            }// 窗口非图标化时调用
            public void windowClosed(WindowEvent e) {
            }// 窗口关闭时调用
    }
}
```

程序执行结果如图8-20所示。

图 8-20 例 8-17 的运行结果

当用户单击第2个窗口时，第1个窗口失去焦点；反之则第1个窗口获得焦点，第2个窗口失去焦点。控制台显示用户操作引起焦点转移的过程如图8-21所示。

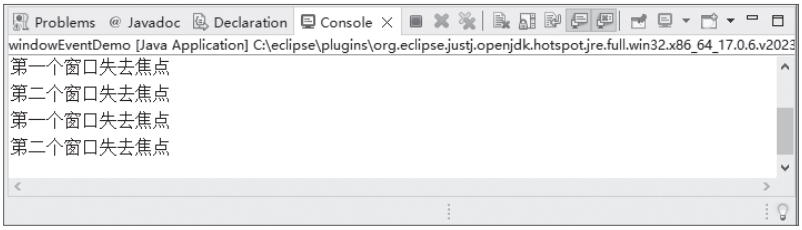


图 8-21　用户操作的执行结果

注意事项 接口中有多个抽象方法时，如果某个方法不需要处理，也要以空方法体的形式给出方法的实现。例如，本例中的windowActivated()方法就是以空方法体进行的实现。

8.6.2　事件适配器

从上例的窗口事件可以看出，为了进行事件处理需要创建实现对应接口的类。而在这些接口中往往声明很多抽象方法，为了实现这些接口需要给出这些方法的所有实现。如WindowListener接口中定义7个抽象方法，在实现接口的类中必须同时实现这7个方法。然而，多数情况是用户往往只关心其中的某一个或者某几个方法，为了简化编程可以使用适配器（Adapter）类。

具有两个以上方法的监听器接口均对应一个XXXAdapter类，提供该接口中每个方法的默认实现。在实际开发中，在编写监听器时不再直接实现监听接口，而是继承适配器类，并重写实际需要的事件处理方法，这样可避免编写大量不必要的代码。表8-4是一些常用的适配器类。

表8-4　Java中常用的适配器类

适配器类	实现的接口
ComponentAdapter	ComponentListener，EventListener
ContainerAdapter	ContainerListener, EventListener
FocusAdapter	FocusListener, EventListener
KeyAdapter	KeyListener, EventListener
MouseAdapter	MouseListener, EventListener
MouseMotionAdapter	MouseMotionListener, EventListener
WindowAdapter	WindowFocusListener,WindowListener,WindowStateListener, EventListener

表中所给的适配器都在java.awt.event包中。Java是单继承，一个类继承了适配器就不能再继承其他类了。因此在使用适配器开发监听程序时经常使用匿名类或内部类来实现。

8.6.3　键盘事件的处理

键盘操作也是最常用的用户交互方式，Java提供KeyEvent类来捕获键盘事件。处理KeyEvent事件的监听器对象可以是实现KeyListener接口的类对象，或者是继承KeyAdapter类的类对象。在

KeyListener接口中包括如下三个事件。

①public void keyPressed(KeyEvent e)：代表键盘按键被按下的事件。

②public void keyReleased(KeyEvent e)：代表键盘按键被放开的事件。

③public void keyTyped(KeyEvent e)：代表按键被敲击的事件。

KeyEvent类中的常用如下方法。

①char getKeyChar()：返回引发键盘事件的按键对应的Unicode字符。如果这个按键没有Unicode字符与之对应，则返回KeyEvent类的一个静态常量KeyEvent.CHAR-UNDEFINED。

②String getKeyText()：返回引发键盘事件的按键的文本内容。

③int getKeyCode()：返回与此事件中的键相关联的整数keyCode。

下面通过一个案例说明键盘事件的使用。

【例8-18】键盘事件的使用。

功能实现：把所敲击的按键上的键符显示在窗口上，当按下Esc键时退出程序。

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class KeyEventDemo extends JFrame {
    // 标签对象用于显示提示信息
    private JLabel showInf;
    private Container container;
    public KeyEventDemo() {
        container = this.getContentPane();
        container.setLayout(new BorderLayout()); // 设置布局管理器
        showInf = new JLabel();// 创建标签对象
        container.add(showInf, BorderLayout.NORTH); // 把标签放到窗口的北部
        this.addKeyListener(new keyLis()); // 注册键盘事件监听程序 keyLis() 为内部类
        // 注册窗口事件监听程序，监听器以匿名内部类的形式进行
        this.addWindowListener(new WindowAdapter() {// 匿名内部类开始
                public void windowClosing(WindowEvent e) {
                    // 把退出窗口的语句写在本方法中
                    System.exit(0);
                } // 窗口关闭
            });// 匿名类结束
        this.setSize(300, 200); // 设置窗口大小
        this.setVisible(true); // 设置窗口为可见
    }
    class keyLis extends KeyAdapter { /* 内部类开始 */
        public void keyTyped(KeyEvent e) {
            // 获取键盘键入的字符
            char c = e.getKeyChar();
```

```
            //设置标签上的显示信息
            showInf.setText("你按下的键盘键是" + c + "");
        }
        public void keyPressed(KeyEvent e) {
            if (e.getKeyCode() == 27) //如果按下Esc键，则退出程序
                System.exit(0);
        }
    } /* 内部类结束 */
    public static void main(String[] arg) {
        new KeyEventDemo();
    }
}
```

程序执行结果如图8-22所示。

图 8-22　例 8-18 的程序执行结果

注意事项 在本例中，对键盘事件的处理，采用的是内部类keyLis作为键盘事件的监听程序。该类是KeyAdapter类的子类，只对键盘按下和键盘敲击两种事件给出处理，同时也对窗口事件进行处理。由于windowListener接口中有7类事件，而这里只需要对窗口关闭事件进行处理即可，所以采用匿名内部类作为窗口事件的监听器。该例子对窗口注册了多个不同类型的监听器，可以对不同类型的事件进行处理。

8.6.3　鼠标事件的处理

在图形用户界面中，鼠标主要用来进行选择、切换或绘画。当用户用鼠标进行交互操作时，会产生鼠标事件MouseEvent。所有的组件都可以产生鼠标事件。可以通过实现MouseListener接口和MouseMotionListener接口的类，或者是继承MouseAdapter的子类来处理相应的鼠标事件。

与Mouse有关的事件可分为两类。一类是MouseListener接口，主要针对鼠标的按键与位置作检测，共提供如下5个事件的处理方法。

①public void mouseClicked (MouseEvent e)：鼠标单击事件。

②public void mouseEntered (MouseEvent e)：鼠标进入事件。

③public void mousePressed (MouseEvent)：鼠标按下事件。

④public void mouseReleased (MouseEvent)：鼠标释放事件。

⑤public void mouseExited (MouseEvent)：鼠标离开事件。

另一类是MouseMotionListener接口，主要针对鼠标的坐标与拖动操作做处理，处理方法有如下两个。

public void mouseDragged(MouseEvent)：鼠标拖动事件。

public void mouseMoved(MouseEvent)：鼠标移动事件。

MouseEvent类还提供获取发生鼠标事件坐标及单击次数的成员方法，MouseEvent类中的常用方法如下。

①Point getPoint()：返回Point对象，包含鼠标事件发生的坐标点。

②int getClickCount()：返回与此事件关联的鼠标单击次数。

③int getX()：返回鼠标事件x坐标。

④int getY()：返回鼠标事件y坐标。

⑤int getButton()：返回哪个鼠标按键更改了状态。

下面通过一个案例说明鼠标事件的使用。

【例8-19】键盘事件的使用。

功能实现：检测鼠标的坐标位置，并在窗口的文本框中显示，同时还显示鼠标的按键操作。

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
// 当前类作为 MouseEvent 事件的监听者，该类需要实现对应的接口
public class MouseEventDemo extends JFrame implements MouseListener {
    private JLabel showX, showY, showSatus; // 显示提示信息的标签
    private JTextField t1, t2; // 用于显示鼠标 x、y 坐标的文本框
    private Container container;
    public MouseEventDemo() {
        container = this.getContentPane();// 获取内容窗格
        container.setLayout(new FlowLayout()); // 设置布局格式
        showX = new JLabel("X 坐标 ");// 创建标签对象，字符串为提示信息
        showY = new JLabel("Y 坐标 ");// 创建标签对象，字符串为提示信息
        showSatus = new JLabel();// 创建标签初始为空，用于显示鼠标的状态信息
        // 创建显示信息的文本，用于显示鼠标坐标的值，最多显示 10 个字符
        t1 = new JTextField(10);
        t2 = new JTextField(10);
        // 把组件顺次放入窗口的内容窗格
        container.add(showX);
        container.add(t1);
        container.add(showY);
        container.add(t2);
        container.add(showSatus);
        /* 为本窗口注册鼠标事件监听程序为当前类，mouseEventDemo 必须实现 MouseListener
接口或者继承 MouseAdapter 类 */
        this.addMouseListener(this);
```

```
        // 为窗口注册 MouseMotionEvent 监听程序，为 MouseMotionAdapter 类的子类
        this.addMouseMotionListener(new mouseMotionLis());
        // 注册窗口事件监听程序，监听器以匿名类的形式进行
        this.addWindowListener(new WindowAdapter() {// 匿名内部类开始
            public void windowClosing(WindowEvent e) {
            // 把退出窗口的语句写在本方法中
                System.exit(0);
            } // 窗口关闭
            });// 匿名内部类结束
        this.setSize(300, 150); // 设置窗口大小
        this.setVisible(true); // 设置窗口可见
    }
    /* 内部类开始作为 MouseMotionEvent 的事件监听者 */
    class mouseMotionLis extends MouseMotionAdapter {
        public void mouseMoved(MouseEvent e) {
            int x = e.getX(); // 获取鼠标的 x 坐标
            int y = e.getY(); // 获取鼠标的 y 坐标
            t1.setText(String.valueOf(x)); // 设置文本框的提示信息
            t2.setText(String.valueOf(y));
        }
        public void mouseDragged(MouseEvent e) {
            showSatus.setText("拖动鼠标"); // 设置标签的提示信息
        }
    } /* 内部类结束 */
    // 以下方法是 mouseListener 接口中事件的实现对鼠标的按键与位置作检测
    public void mouseClicked(MouseEvent e) {
        showSatus.setText("单击鼠标" + e.getClickCount() + "次");
    } // 获取鼠标单击次数
    public void mousePressed(MouseEvent e) {
        showSatus.setText("鼠标按下");
    }
    public void mouseEntered(MouseEvent e) {
        showSatus.setText("鼠标进入窗口");
    }
    public void mouseExited(MouseEvent e) {
        showSatus.setText("鼠标不在窗口");
    }
    public void mouseReleased(MouseEvent e) {
        showSatus.setText("鼠标释放");
    }
    public static void main(String[] arg) {
```

```
            new MouseEventDemo();// 创建窗口对象
        }
    }
```

程序执行结果如图8-23所示。

图 8-23 例 8-19 的执行结果

注意事项 在本例中，程序自动检测鼠标的拖动以及进入和离开窗口的情况，并在窗口上显示。为一个组件注册了多个监听器。对于MouseEvent事件，采用组件所在类实现监听器接口的方式作为事件的监听者，对于鼠标的移动和拖动的处理，采用内部类继承适配器的方式来实现。对于关闭窗口的事件，采用了匿名类来处理。

8.7 菜单

菜单在GUI应用程序中有着非常重要的作用。通过菜单用户可以非常方便地访问应用程序的各个功能，是软件中必备的组件之一，利用菜单可以将程序功能模块化。菜单通常依附于JFrame，主要包括JMenuBar、JMenu、JMenuItem三个组件。

8.7.1 菜单概述

Swing包中提供了多种菜单组件，它们的继承关系如图8-24所示。通过菜单组件可以创建多种样式的菜单，如下拉式、快捷键式及弹出式菜单等。本章主要介绍下拉式菜单和弹出式菜单的定义与使用。

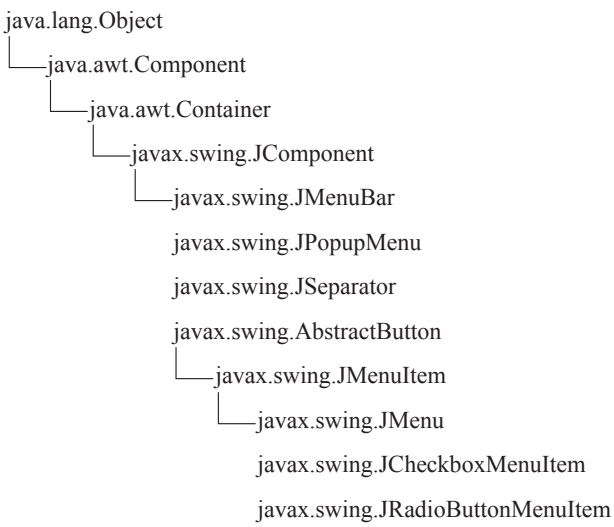


图 8-24 菜单组件的继承关系

8.7.2 下拉式菜单

下拉式菜单是最常用的菜单，用来包容一组菜单项和子菜单。多个菜单放在菜单栏上，构成系统菜单。

1. 菜单栏（JMenuBar）

菜单栏是窗口中的主菜单，只用来管理菜单，不参与交互式操作。Java应用程序中的菜单都包含在一个菜单栏对象之中。

JMenuBar(菜单栏)只有一个构造方法JMenuBar()。顶层容器类如JFrame都有一个setMenuBar(JMenuBar menu)方法，通过该方法可以把菜单栏添加到窗口上。

创建菜单栏并把菜单添加到窗口可以采用如下代码片段。

```
JMenuBar menuBar = new JMenuBar ();     // 创建菜单栏
JFrame frame = new JFrame(" 菜单示例 " ); // 创建窗口
frame. setMenuBar(menuBar);             // 把菜单栏添加到窗口
```

2. 菜单（JMenu）

菜单是用来存放和整合菜单项(JMenuItem)的组件。菜单可以是单一层次的结构，也可以是一个多层次的结构。具体使用何种形式的结构则取决于界面设计上的需要。

（1）JMenu常用的构造方法

①JMenu()：创建一个空标签的JMenu对象。

②JMenu(String text)：使用指定的标签创建一个JMenu对象。

③JMenu(String text，Boolean b)：使用指定的标签创建一个JMenu对象，并给出此菜单是否具有下拉式的属性。

创建菜单并把菜单添加到菜单栏，可以采用如下代码片段。

```
JMenu fileMenu = new JMenu(" 文件 (F)");  // 创建菜单
JMenu helpMenu = new JMenu(" 帮助 (H)");  // 创建菜单
menuBar.add(fileMenu);    // 把菜单添加到菜单栏
menuBar.add(helpMenu);    // 把菜单添加到菜单栏
```

（2）JMenu常用的成员方法

①getItem(int pos)：得到指定位置的JMenuItem。

②getItemCount()：得到菜单项数目包括分隔符。

③insert()和remove()：插入菜单项或者移除某个菜单项。

④addSeparator()和insertSeparator(int index)：在某个菜单项间加入分隔线。

3. 菜单项（JMenuItem）

菜单项是菜单系统中最基本的组件，继承自AbstractButton类，所以也可以把菜单项看作一个按钮。它支持许多按钮的功能，例如，加入图标(Icon)，以及在菜单中选择某一项时会触发ActionEvent事件等。

（1）JMenuItem常用的构造方法

①JMenuItem(String text)：创建一个具有文本提示信息的菜单项。

②JMenuItem(Icon icon)：创建一个具有图标的菜单项。

③JMenuItem(String text，Icon icon)：创建一个既有文本又有图标的菜单项。

④JMenuItem(String text，int mnemonic)：创建一个指定文本和键盘快捷的菜单项。

（2）菜单项常用的成员方法

①void setEnabled(boolean b)：启用或禁用菜单项。

②void setAccelerator(KeyStroke keyStroke)：设置加速键。它能直接调用菜单项的操作监听器，而不必显示菜单的层次结构。

③void setMnemonic(char mnemonic)：设置快捷键。

设置菜单项的加速键，可以采用如下的代码片段。

```
// 通过构造方法设置加速键
JMenuItem fNew = new JMenuItem("新建(N)",KeyEvent.VK_N);
// 通过调用 setMnemonic() 设置加速键
JMenuItem fOpen = new JMenuItem("打开(O)...");
fOpen.setMnemonic(KeyEvent.VK_O);
```

制作一个可用的菜单系统，一般需要经过下面的几个步骤。

步骤01 创建一个JMenuBar对象，并将其添加到JFrame对象。

步骤02 创建JMenu对象。

步骤03 创建JMenuItem对象，并将其添加到JMenu对象中。

步骤04 把JMenu对象添加到JMenuBar中。

上面的步骤主要是为了创建菜单的结构。如果要使用菜单所指出的功能，必须为菜单项注册监听器，并在监听器中提供事件处理程序。下面通过一个案例演示菜单的使用方法。

【例8-20】菜单的使用。

功能实现：创建下拉式菜单，当用户单击“退出”菜单项时，退出系统；单击其他菜单项时，在控制台输出与该菜单项有关的提示信息。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class MenuDemo extends JFrame implements ActionListener{
    JMenuItem save;    // 保存菜单项
    JMenuItem exit;    // 退出菜单项
    JMenuItem cut;     // 剪切菜单项
    JMenuItem copy;   // 复制菜单项
    JMenuItem paste;   // 粘贴菜单项
    JMenuItem about;   // 关于菜单项
```

```
public MenuDemo() {
    this.setTitle("菜单使用演示程序");  //设置标题
    Container container = this.getContentPane();  //获取默认的内容窗格
    container.setLayout(new BorderLayout());    //设置布局格式
    JMenuBar menuBar = new JMenuBar();      //创建菜单栏
    buildMainMenu(menuBar);                 //调用创建菜单的方法
    this.setJMenuBar(menuBar);               //把菜单栏添加到窗口
    this.setVisible(true);    //显示窗口
    this.setSize(300, 200);  //设置窗口大小
}
protected void buildMainMenu(JMenuBar menuBar) {
    //文件菜单
    JMenu fileMenu = new JMenu("文件(F)");
    //菜单项
    save = new JMenuItem("保存");
    exit = new JMenuItem("退出");
    //注册监听器
    save.addActionListener(this);
    exit.addActionListener(this);
    //把菜单项添加到菜单
    fileMenu.add(save);
    fileMenu.add(exit);
    //把菜单添加到菜单栏
    menuBar.add(fileMenu);
    //编辑菜单
    JMenu editMenu = new JMenu("编辑(E)");
    //菜单项
    cut = new JMenuItem("剪切");
    copy = new JMenuItem("复制");
    paste = new JMenuItem("粘贴");
    //注册监听器
    cut.addActionListener(this);
    copy.addActionListener(this);
    paste.addActionListener(this);
    //把菜单项添加到菜单
    editMenu.add(cut);
    editMenu.add(copy);
    editMenu.add(paste);
    //把菜单添加到菜单栏
    menuBar.add(editMenu);
    //帮助菜单
    JMenu helpMenu = new JMenu("帮助(H)");
```

```
        // 菜单项
        about = new JMenuItem("关于");
        // 注册监听器
        about.addActionListener(this);
        // 把菜单项添加到菜单
        helpMenu.add(about);
        // 把菜单添加到菜单栏
        menuBar.add(helpMenu);
    }
    // 单击菜单事件处理程序
    public void actionPerformed(ActionEvent e) {
        // 用户单击"退出"菜单项时，退出系统
        if(e.getSource()==exit) {
            System.exit(0);
        }
        // 用户单击其他菜单项时，在控制台输出提示信息
        if(e.getSource()==save) {
            System.out.println("用户单击的是 " +save.getText()+"菜单项");
        }else if(e.getSource()==cut) {
            System.out.println("用户单击的是 " +cut.getText()+"菜单项");
        }else if(e.getSource()==copy) {
            System.out.println("用户单击的是 " +copy.getText()+"菜单项");
        }else if(e.getSource()==paste) {
            System.out.println("用户单击的是 " +paste.getText()+"菜单项");
        }else if(e.getSource()==about) {
            System.out.println("用户单击的是 " +about.getText()+"菜单项");
        }
    }
    // 主方法
    public static void main(String[] args) {
        new MenuDemo();
    }
}
```

程序执行结果如图8-25所示。

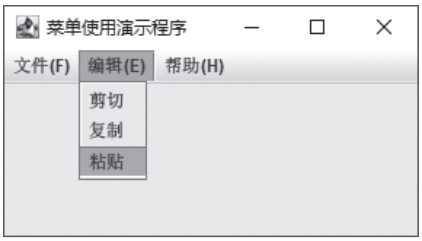


图 8-25 例 8-20 的运行结果

单击“粘贴”菜单项，控制台输出信息：“用户单击的是粘贴菜单项”。

单击“退出”菜单项，关闭窗口，退出系统。

8.7.3 弹出式菜单

弹出式菜单（JPopupMenu）是一种特殊菜单，可以根据需要显示在指定的位置。弹出式菜单有两种构造方法。

①public JPopupMenu()：创建一个没有名称的弹出式菜单。

②public JPopupMenu(String label)：构建一个有指定名称的弹出式菜单。

在弹出式菜单中可以像下拉式菜单一样加入菜单或者菜单项。在显示弹出式菜单时，必须调用show(Component invoker, int x,int y)方法。在该方法中需要一个组件作参数，该组件的位置将作为显示弹出式菜单的参考原点。同样可以像下拉式菜单一样为菜单项进行事件注册，对用户的交互作出响应。下面通过一个案例演示弹出式菜单的使用方法。

【例8-21】弹出式菜单的使用。

功能实现：创建窗口和弹出式菜单，当鼠标右键单击窗口时，弹出式菜单显示到鼠标单击的位置。

```
import javax.swing.*;
import java.awt.event.*;
public class JPopupMenuDemo extends JFrame {
    JPopupMenu popMenu;
    public JPopupMenuDemo() {
        this.setTitle(" 弹出式菜单演示示例 ");
        popMenu = new JPopupMenu();
        // 创建 4 个菜单项，并添加到弹出式菜单上
        JMenuItem save = new JMenuItem("Save");
        JMenuItem cut = new JMenuItem("Cut");
        JMenuItem copy = new JMenuItem("Copy");
        JMenuItem exit = new JMenuItem("Exit");
        popMenu.add(save);
        popMenu.add(cut);
        popMenu.add(copy);
        // 添加分隔线
        popMenu.addSeparator();
        popMenu.add(exit);
        this.addMouseListener(new mouseLis());
        this.setVisible(true);
        this.setSize(300, 200);
    }
    // 监听器类
    class mouseLis extends MouseAdapter {
```

```
        public void mouseClicked(MouseEvent e) {
            if (e.getButton() == e.BUTTON3) // 判断单击的是否是鼠标右键
            popMenu.show(e.getComponent(), e.getX(), e.getY()); // 在当前位置显示
        }
    }
    public static void main(String[] args) {
        new JPopupMenuDemo();
    }
}
```

程序执行结果如图8-26所示。

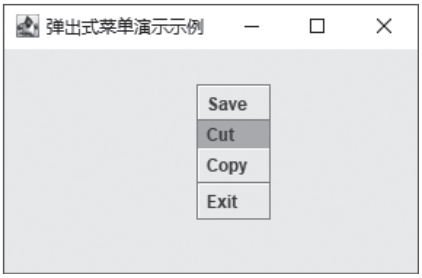


图 8-26　例 8-21 的运行结果

8.8 表格

表格（JTable）是图形用户界面设计中使用频率较高的一个高级组件，为显示大块数据提供了一种简单的机制，可以用于数据的生成和编辑。

（1）JTable常用的构造方法

①JTable ()：构造一个默认的表格。

②JTable(int numRows, int numColumns)：使用默认模式构造指定行和列的表格。

③JTable(Object[][] rowData, Object[] columnNames)：构造一个columnNames作为列名，显示二维数组YOWData中的数据的表格。

④JTable(Vector rowData, Vector columnNames)：构造columnNames作为列名，rowData中数据作为输入来源的表格。

（2）JTable类常用的成员方法

①void addColumn(TableColumn aColumn)：将列追加到列数组的结尾。

②int getColumnCount()：返回表格中的列数。

③int getRowCount()：返回此表格中的行数。

④void moveColumn(int column, int targetColumn)：移动列到目标位置。

⑤void removeColumn(TableColumn aColumn)：从表格的列数组中移除一列。

⑥void selectAll()：选择表中的所有行、列和单元格。

⑦Object getValueAt(int row, int column)：返回指定单元格的值。

⑧setValueAt(Object aValue, int row, int column)：设置表格指定单元格值。

下面通过一个案例演示JTable的使用方法。

【例8-22】JTable的使用。

功能实现：创建一个表格用于显示学生基本信息，程序运行结果如图8-27所示。

```
import java.awt.*;
import javax.swing.*;
public class JTableDemo extends JFrame {
    JTable stuTable;
    JTableDemo() {
        this.setTitle("JTable 演示程序 ");
        // 表格标题栏中的数据存放到一维数组
        String []colNames= {" 学号 "," 姓名 "," 年龄 "," 专业 "};
        // 表格中的数据存放到二维数组
        String [][]datas= {{"20140101"," 张三 ","19"," 网络工程 "},
                           {"20140102"," 李四 ","21"," 计算机应用 "},
                           {"20140103"," 王五 ","20"," 软件工程 "},
                           {"20140104"," 马六 ","21"," 人工智能 "}
                           };
        // 创建表格
        stuTable = new JTable(datas, colNames);
        // 设置首选的可滚动视口大小
        stuTable.setPreferredScrollableViewportSize(new Dimension(0, 120));
        // 创建滚动面板
        JScrollPane jsp = new JScrollPane();
        // 把表格添加到视口
        jsp.setViewportView(stuTable); // 放置到滚动面板
        // 设置提示信息
        jsp.setBorder(BorderFactory.createTitledBorder(" 学生信息 "));
        // 把滚动面板添加到窗口
        this.add(jsp);
        this.setSize(390, 200);
        this.setVisible(true);
    }

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

程序执行结果如图8-27所示。

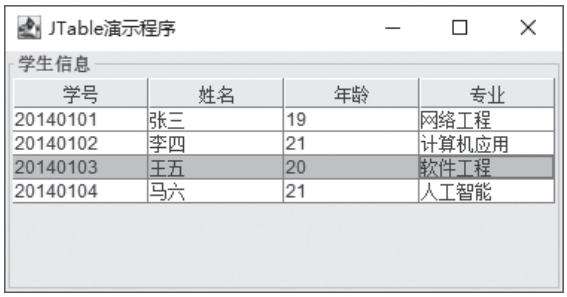


图 8-27 例 8-22 的运行结果

表格（JTable）是Swing包中最复杂的组件之一。在本书中只对它进行简单介绍，如果读者需要深入学习，可以参考Java API或者联机帮助。

8.9 树

JTree组件简称为树形组件，可以显示具有层次结构的数据。树形组件中的数据表现形式为一层套一层，结构清晰明了。用户可以方便地了解数据之间的层次关系，从而很容易地找到相关数据。例如Windows系统的文件管理器就是一个典型的树形层次结构。

一个JTree（树）对象并没有包含实际的数据，只提供数据的一个视图。树对象通过查询它的数据模型获得数据。树对象垂直显示它的数据，树中显示的每一行包含一项数据，称为节点。每棵树有一个根节点，其他所有节点是它的子孙。默认情况下，树只显示根节点，但是可以设置默认显示方式。一个节点可以拥有孩子也可以没有任何子孙。那些可以拥有孩子的节点被称为“分支节点”，而不能拥有孩子的节点为“叶子节点”。分支节点可以有任意多个孩子。通常，用户可以通过单击展开或者折叠分支节点，使得它的孩子可见或不可见。默认情况下，除了根节点以外的所有分支节点呈现折叠状态。

（1）JTree常用的构造方法

①JTree()：建立一棵带有示例模型的JTree。

②JTree(Hashtable<?,?> value)：返回从HashTable创建的JTree，不显示根。

③JTree(Object[] value)：返回JTree，指定数组的每个元素作为不被显示的新根节点的子节点。

④JTree(TreeModel newModel)：返回JTree的一个实例，使用指定的数据模型，显示根节点。

⑤Tree(TreeNode root)：返回JTree，指定TreeNode作为其根，显示根节点。

⑥JTree(TreeNode root,Boolean asksAllowsChildren)：返回JTree，指定TreeNode作为其根。它用指定的方式显示根节点，并确定节点是否为叶节点。

⑦JTree(Vector<?> value)：返回JTree，指定Vector的每个元素作为不被显示的新根节点的子节点。

下面通过一个简单的案例演示JTree的应用。

【例8-23】JTree的应用。

功能实现：使用简单TreeNode模型创建一个树，当用户单击“添加节点”按钮时，为树添加一个分支。

```
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
public class JTreeDemo extends JFrame {
    static int i = 0;
    // DefaultMutableTreeNode 是树结构中的通用节点
    DefaultMutableTreeNode root;
    DefaultMutableTreeNode child;
    DefaultMutableTreeNode chosen;
    JTree tree;
    DefaultTreeModel model; // 使用 TreeNodes 的简单树数据模型
    String[][] data = {
            { "财务部", "财务管理", "成本核算" },
            { "总经办", "档案管理", "行政事务"},
            { "工程部", "项目管理", "质检部"}
            };
    JTreeDemo() {
        this.setTitle("JTree 演示程序");
        Container contentPane = this.getContentPane();
        JPanel jPanel1 = new JPanel(new BorderLayout());
        // 创建根节点
        root = new DefaultMutableTreeNode("公司");
        // 建立以 root 为根的树
        tree = new JTree(root);
        // 将树添加至滚动窗格，同时将滚动窗格添加到 jPanel1 面板
        jPanel1.add(new JScrollPane(tree),BorderLayout.CENTER);
        // 返回提供数据的 TreeModel
        model = (DefaultTreeModel)tree.getModel();
        // 创建按钮
        JButton jButton1 = new JButton("添加节点");
        // 注册监听器
        jButton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // 实现添加节点的功能
                if (i < data.length) {
                    // 使用内部类 Branch 的方法创建子节点 child
                    child = new Branch(data[i++]).node();
                    // 返回当前选择的第一个节点中的最后一个路径组件
                    chosen = (DefaultMutableTreeNode)
                    tree.getLastSelectedPathComponent();
                    // 如果返回值为 null，则令 chosen=root
                    if (chosen == null) {
```

```
                    chosen = root;
                }
                // 如果返回值不是 null，则在父节点 chosen 的子节点中的
                // index 位置插入子节点 child
                model.insertNodeInto(child, chosen, 0);
            }
        }
    });

    jButton1.setBackground(Color.blue);
    jButton1.setForeground(Color.white);
    JPanel jPanel2 = new JPanel();
    jPanel2.add(jButton1);
    jPanel1.add(jPanel2, BorderLayout.SOUTH);
    contentPane.add(jPanel1);
    this.setSize(300, 400);
    this.setLocation(400, 400);
    this.setVisible(true);
}

// 内部类 Branch 是一个工具类，用来获取一个字符串数组
// 并为第一个字符串建立一个 DefaultMutableTreeNode 作为根
// 数组中其余的字符串作为叶子
class Branch {
    DefaultMutableTreeNode r;
    // 构造方法
    public Branch(String[] data) {
        r = new DefaultMutableTreeNode(data[0]);
        for (int i = 1; i < data.length; i++) {
            r.add(new DefaultMutableTreeNode(data[i]));
        }
    }
    // 返回分支的根节点
    public DefaultMutableTreeNode node() {
        return r;
    }
}
// 主方法
public static void main(String args[]) {
    new JTreeDemo();
}
}
```

程序执行结果如图8-28所示。

观察运行结果可以发现，此时树只有一个根节点（公司）。选中根节点，然后单击“添加节点”按钮后，再双击根节点，就可以发现根节点下面已经添加了一个分支（财务部），如图8-29所示。

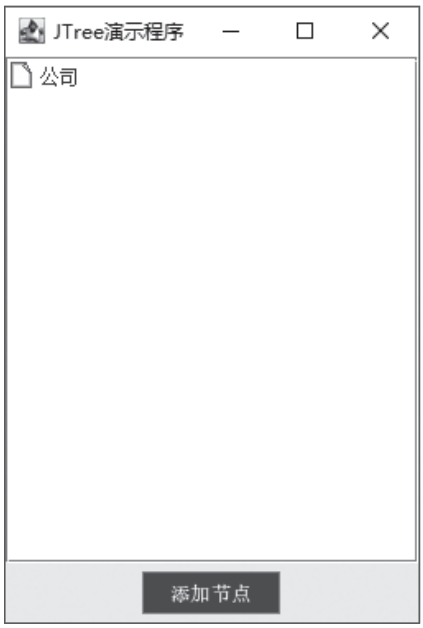


图 8-28　例 8-23 的运行结果

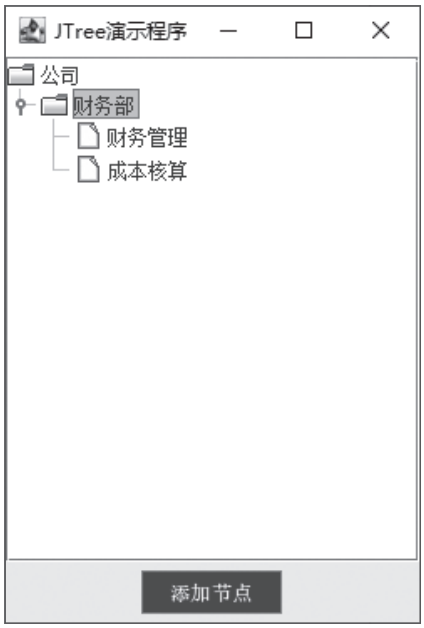


图 8-29　动态创建树的结果

按照同样的操作方法，可以为根节点添加第二个和第三个分支。这样就实现了动态创建一个树的功能。在实际应用中，树的节点可以动态从数据库获取。还可以为树添加相应的事件监听程序，每当用户选择不同的节点时，程序可以做出相应的处理。但由于篇幅所限，在此不再赘述，读者可以参阅Java API进行深入的学习。

8.10 本章小结

本章首先介绍了AWT和Swing之间的关系并对图形用户界面的元素进行了分类；接着对顶层容器类JFrame和中间容器类JPanel以及JScrollPane的定义和使用进行了较为详尽的介绍；然后对布局管理器的特点和用法进行了阐述；最后介绍了常用组件、Java事件处理机制和一些常用的高级组件。通过本章的学习，读者能够掌握图形用户界面的设计与实现，以及对用户操作的响应和处理。

8.11 课后练习

练习1：创建一个包含多个基本组件的窗口，要求其运行结果如图8-30所示。

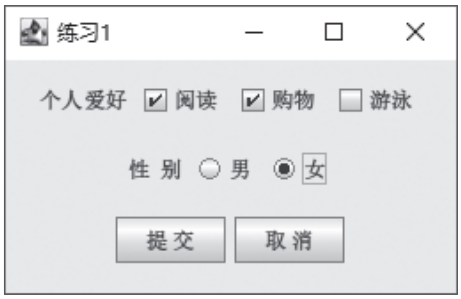


图 8-30 练习 1 的结果

练习2：创建一个窗口，使用Box和BorderLayout布局管理器管理组件，并在窗口上放置6个命令按钮，要求其运行结果如图8-31所示。

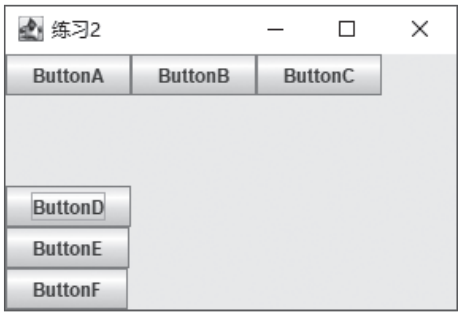


图 8-31 练习 2 的结果

练习3：创建用户登录窗口，要求其运行结果如图8-32所示。具体功能：当用户单击“登录”按钮时，如果输入的账号为admin，密码为123456，则在控制台显示“登录成功”的提示信息，否则显示“用户名或密码不正确”；当用户单击“退出”按钮时，关闭窗口。

图 8-32 练习 3 的结果