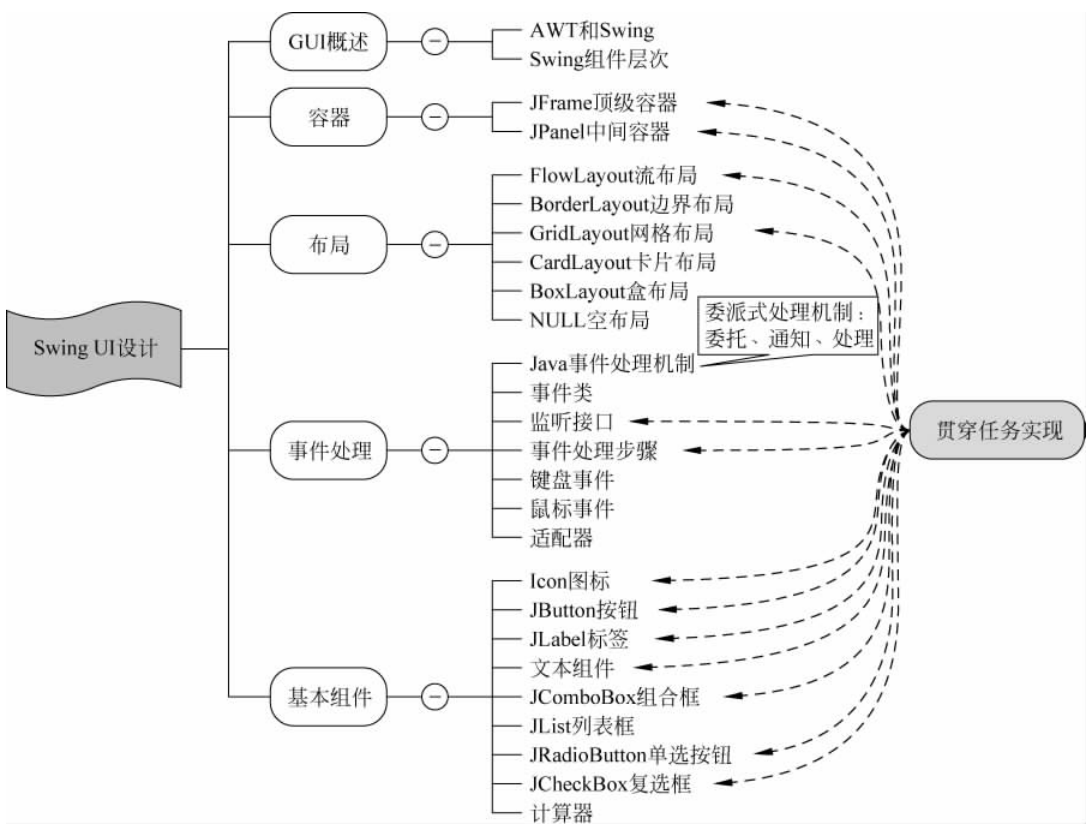
第3章 Swing UI设计

任务驱动

本章任务是完成“Q-DMS 数据挖掘”系统的 UI 设计及注册、登录功能：

- 【任务 3-1】 创建用户数据库表、用户实体类和用户业务逻辑类。
- 【任务 3-2】 创建用户注册窗口，并将用户注册信息保存到数据库。
- 【任务 3-3】 创建用户登录窗口，登录成功则进入系统主界面。

学习路线



知 识 点	Listen(听)	Know(懂)	Do(做)	Revise(复习)	Master(精通)
GUI 概述	★	★			
JFrame 和 JPanel 容器	★	★	★	★	★
布局	★	★	★		
事件处理	★	★	★	★	★
基本组件	★	★	★	★	★

3.1 GUI 概述

用户喜欢功能丰富、操作简单且直观的应用程序。为了提高用户体验度,使系统的交互操作更加交好,大多数应用程序都采用图形用户界面(Graphical User Interface,GUI)的形式。Java 中提供了 AWT(Abstract Windows Tookit,抽象窗口工具集)和 Swing 来实现 GUI 图形用户界面编程。

3.1.1 AWT 和 Swing

在 JDK 1.0 发布时,Sun 提供了一套基本的 GUI 类库,这套基本类库被称为 AWT。AWT 是 Sun 公司最早提供的 GUI 库,为 Java 程序提供基本的图形组件,实现一些基本的功能,并希望在所有平台下都能运行。

使用 AWT 提供的组件所构建的 GUI 应用程序具有以下几个问题:

- 界面功能有限,且不美观;
- 运行在不同的平台上,呈现不同的外观效果,为保证界面的一致和可预见性,程序员需要在不同平台下进行测试;
- 编程模式笨拙,且并非面向对象。

1996 年,Netscape 公司开发了一套工作方式完全不同的 GUI 库,称为 IFC(Internet Foundation Class)。IFC 除了窗口本身需要借助操作系统的窗口来实现,其他的组件都是绘制在空白窗口中。IFC 能够真正地实现各平台界面的一致性,Sun 公司与 Netscape 公司合作完善了这种方案,并创建了一套新的用户界面库,并命名为 Swing。Swing 组件完全采用 Java 语言编写,不再需要使用那些平台所用的复杂的 GUI 功能,因此,使用 Swing 构建的 GUI 应用程序在不同平台上运行时,所显示的外观效果完全相同。

AWT、Swing、2D API、辅助功能 API 以及拖放 API 共同组成了 JFC(Java Foundation Class,Java 基础类库),其中 Swing 全面替代了 Java 1.0 中的 AWT 组件,但保留了 Java 1.1 中的 AWT 事件模型。总体上 Swing 替代了绝大部分 AWT 组件,但并没有完全替代 AWT,

而是在 AWT 的基础之上，进行了有力的补充和加强。

使用 Swing 组件进行 GUI 编程的优势有以下几点：

- Swing 用户界面组件丰富，使用便捷；
- Swing 组件对底层平台的依赖少，与平台相关的 Bug 也很少；
- 能够保证不同平台上用户一致的感观效果。

3.1.2 Swing 组件层次

大部分 Swing 组件都是 JComponent 抽象类的直接或间接子类，在 JComponent 抽象类中定义了所有子类组件的通用方法。JComponent 类位于 javax. swing 包中，javax 包是一个 Java 扩展包。要有效地使用 GUI 组件，必须理解 javax. swing 和 java. awt 包中组件之间的继承层次，尤其是要理解 Component 类、Container 类和 JComponent 类，其中声明了大多数 Swing 组件的通用特性。Swing 中的 JComponent 类是 AWT 中 java. awt. Container 类的子类，也是 Swing 和 AWT 的联系之一。JComponent 类的继承层次如图 3-1 所示：JComponent 类是 Container 的子类；Container 类是 Component 类的子类；而 Component 类又是 Object 类的子类。

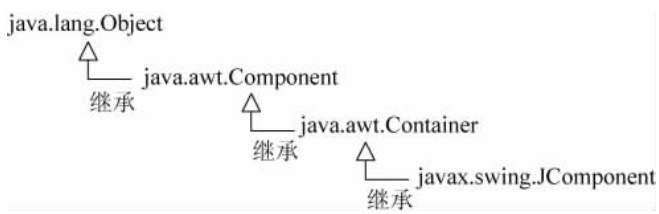


图 3-1 JComponent 类的继承层次

绝大部分的 Swing 组件位于 javax. swing 包中，且继承 Container 类。Swing 组件之间的继承层次如图 3-2 所示。

将 Swing 组件按功能进行划分，可以分为以下几类：

- 顶层容器——JFrame、JApplet、JDialog 和 JWindow；
- 中间容器——JPanel、JScrollPane、JSplitPane 和 JToolBar 等；
- 特殊容器——特殊作用的中间容器，如 JInternalFrame、JRootPane 和 JLayeredPane 等；
- 基本组件——与用户交互的组件，如 JButton、JComboBox、JList、JMenu 和 JSlider 等；
- 特殊对话框——直接产生特殊对话框组件，如 JColorChooser 和 JFileChooser 等；
- 不可编辑信息的显示组件——组件中的信息不可编辑，如 JLabel、JProgressBar 和 JToolTip 等；
- 可编辑信息的显示组件——组件中的信息可以编辑，如 JTextField、JTextArea 和 JTable 等。

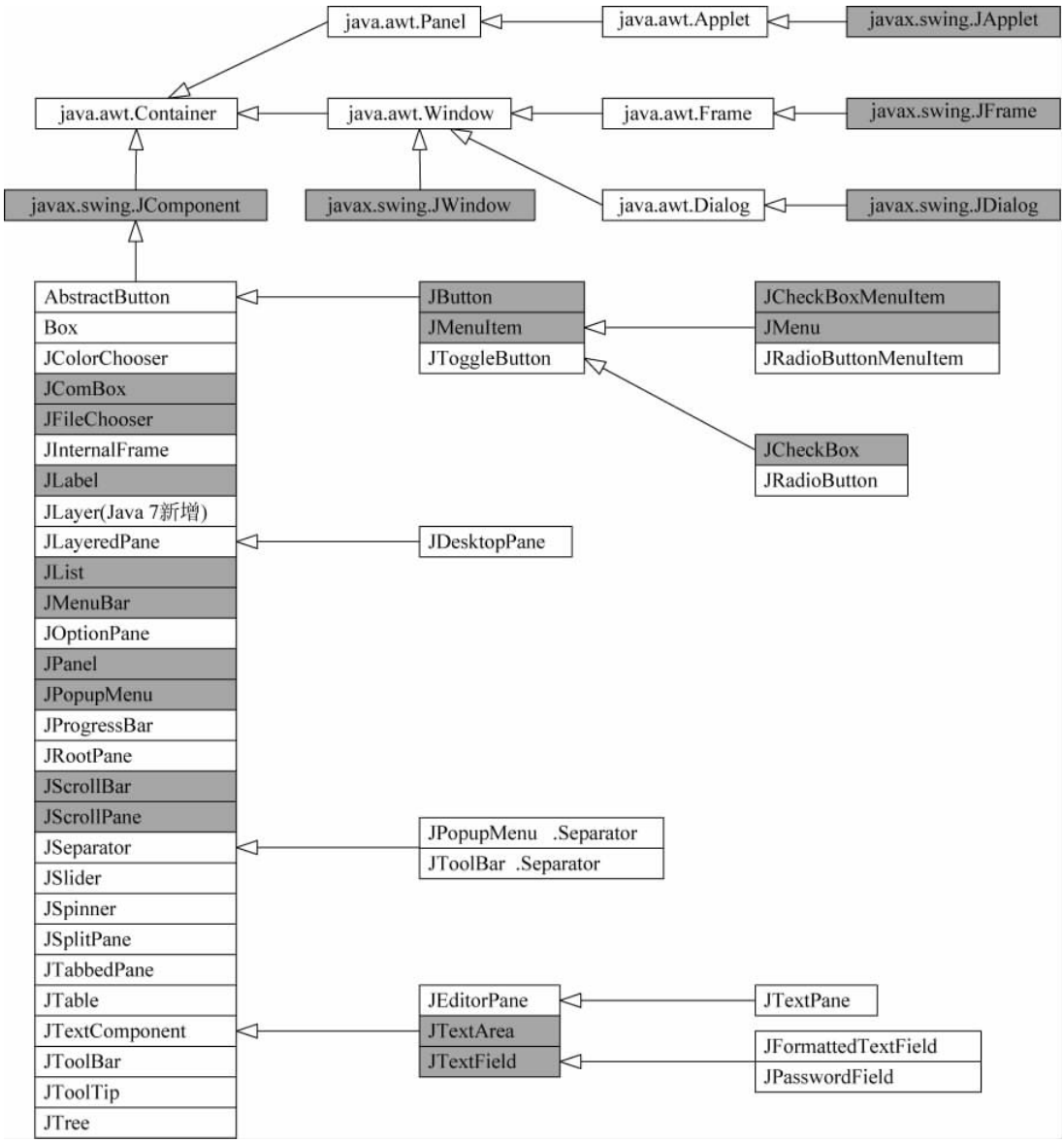


图 3-2 Swing 组件继承图

3.2 容器

容器可以用来存放其他组件，而容器的本身也是一个组件，属于 Component 的子类。容器具有组件的所有性质，同时还具备容纳其他组件的容器功能。

3.2.1 JFrame 顶级容器

JFrame(窗口框架)是可以独立存在的顶级窗口容器，能够包含其他子容器，但不能被其他容器所包含。JFrame 继承了 java.awt.Frame 类，该类的继承层次如图 3-3 所示。

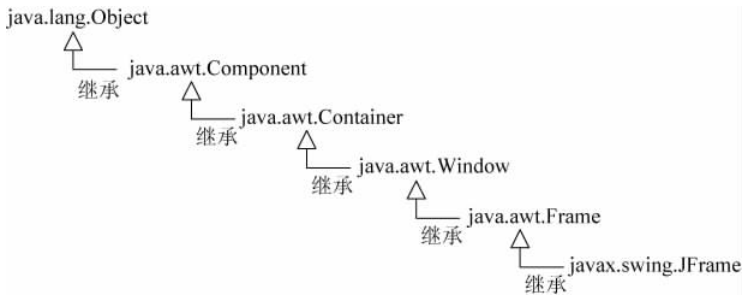


图 3-3　JFrame 类的继承层次

JFrame 类常用的构造方法有以下两种：

- JFrame()——不带参数的构造方法，该方法用于创建一个初始不可见的新窗体；
- JFrame(String title)——带一个字符串参数的构造方法，该方法用于创建一个初始不可见的新窗体，且窗口的标题由字符串参数指定。

JFrame 类常用的方法及功能如表 3-1 所示。

表 3-1　JFrame 的方法列表

方　　法	功 能 描 述
protected void frameInit()	在构造方法中调用该方法用来初始化窗体
public Component add(Component comp)	该方法从 Container 类中继承而来，用于向窗口中添加组件
public voidsetLocation(int x,int y)	该方法从 Component 类中继承而来，用于设置窗口的位置坐标(以像素为单位)
public void setSize(int width,int height)	该方法从 Window 类中继承而来，用于设置窗口的大小(以像素为单位)
public void setVisible(boolean b)	该方法从 Window 类中继承而来，用于设置是否可视，当参数为 true 则可视，false 则隐藏
public void setContentPane(Container contentPane)	设置容器面板
public void setIconImage(Image image)	设置窗体左上角的图标
public void setJMenuBar(JMenuBar menubar)	设置窗体的菜单栏
public void setDefaultCloseOperation(int operation)	设置用户对此窗体的默认关闭操作，该方法的参数是常量，必须是以下选项之一： • DO_NOTHING_ON_CLOSE——不执行任何操作； • HIDE_ON_CLOSE——自动隐藏该窗体； • DISPOSE_ON_CLOSE——自动隐藏并释放该窗体； • EXIT_ON_CLOSE——退出应用程序。
public void setTitle(String title)	该方法从 Frame 类中继承而来，用于设置窗口的标题

下述代码演示如何使用 JFrame 创建可视化窗体。

【代码 3-1】　JFrameDemo.java

```
package com.qst.chapter03;
import javax.swing.JFrame;
public class JFrameDemo extends JFrame {
```

```
    public JFrameDemo() {
        // 调用父类构造方法,并指定窗口标题
        super("QST 青软实训");
        // 设定窗口大小(宽度 400 像素,高度 300 像素)
        this.setSize(400, 300);
        // 设定窗口左上角坐标(X 轴 200 像素,Y 轴 100 像素)
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        // 实例化一个 JFrameDemo 对象
        new JFrameDemo();
    }
}
```

上述代码创建一个窗口继承 JFrame 类,并调用不同方法对该窗口进行设置,运行结果如图 3-4 所示。

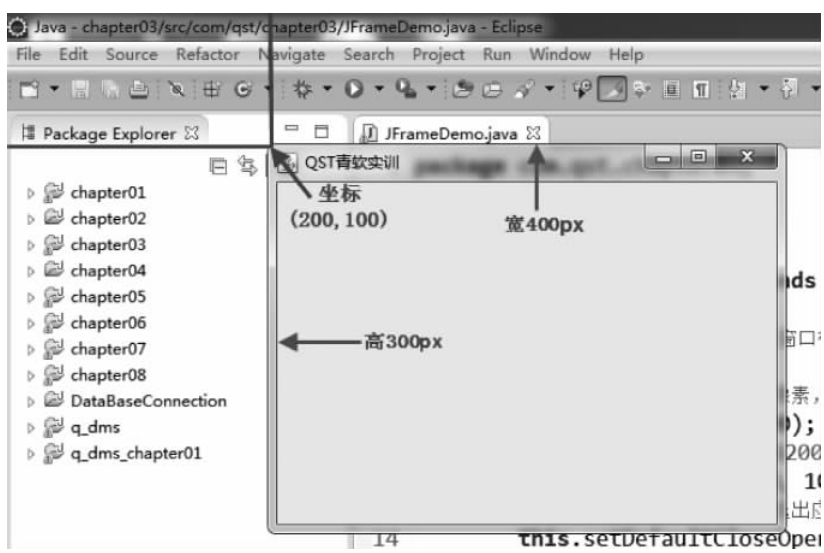


图 3-4 创建 JFrame 窗口

3.2.2 JPanel 中间容器

JPanel(面板)是一种中间容器,中间容器与顶级容器不同,不能独立存在,必须放在其他容器中。JPanel 中间容器的意义在于为其他组件提供空间。在使用 JPanel 时,通常先将其他组件添加到 JPanel 中间容器中,然后再将 JPanel 中间容器添加到 JFrame 顶级容器中。

JPanel 类继承 JComponent 类,其继承层次如图 3-5 所示。

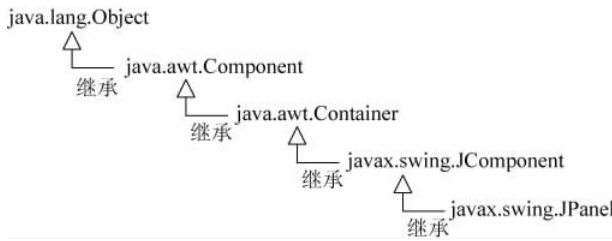


图 3-5　JPanel 类的继承层次

JPanel 类常用的构造方法有以下两种：

- JPanel ()——不带参数的构造方法，用于创建一个默认为流布局(FlowLayout)的面板；
- JPanel(LayoutManager layout)——带参数的构造方法，参数是一个布局管理器，用于创建一个指定布局的面板。

JPanel 类的常用的方法及功能如表 3-2 所示。

表 3-2　JPanel 的方法列表

方　　法	功 能 描 述
public Component add(Component comp)	该方法从 Container 类中继承而来，用于向面板容器中添加其他组件
public void setLayout(LayoutManager mgr)	该方法从 Container 类中继承而来，用于设置面板的布局方式

下述代码演示 JPanel 中间容器的使用。

【代码 3-2】　JPanelDemo. java

```
package com.qst.chapter03;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class JPanelDemo extends JFrame {
    // 声明一个面板对象
    private JPanel p;
    // 声明两个按钮对象
    private JButton btnOk, btnCancel;
    public JPanelDemo() {
        super("JPanelDemo");
        // 实例化面板对象 p(默认为流布局)
        p = new JPanel();
        // 实例化一个按钮对象,该按钮上的文本为"确认"
        btnOk = new JButton("确认");
        // 实例化一个按钮对象,该按钮上的文本为"取消"
        btnCancel = new JButton("取消");
        // 将按钮添加到面板中
        p.add(btnOk);
        p.add(btnCancel);
```

```
            // 将面板添加到窗体中
            this.add(p);
            // 设定窗口大小(宽度 400 像素,高度 300 像素)
            this.setSize(400, 300);
            // 设定窗口左上角坐标(X 轴 200 像素,Y 轴 100 像素)
            this.setLocation(200, 100);
            // 设定窗口默认关闭方式为退出应用程序
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // 设置窗口可视(显示)
            this.setVisible(true);
        }
        public static void main(String[] args) {
            new JPanelDemo();
        }
    }
```

上述代码先声明一个面板对象(p)和两个按钮对象(btnOk 和 btnCancel),在构造方法中先将面板容器和按钮对象实例化,再将两个按钮添加到面板容器中,最后将面板添加到窗口中。

注意

此处为了效果演示,先简单了解 JButton 按钮组件的使用,有关 JButton 按钮类的详细介绍见 3.5.1 节。

运行结果如图 3-6 所示。

图 3-6 JPanel 中间容器

3.3 布局

布局管理器用来管理组件在容器中的布局格式。当容器中容纳多个组件时,可以使用布局管理器对这些组件进行排列和分布。AWT 提供了 FlowLayout、BorderLayout、GridLayout、GridBagLayout 和 CardLayout 五个常用的布局管理器,Swing 还提供了 BoxLayout

布局管理器。本节详细介绍 FlowLayout、BorderLayout、GridLayout、CardLayout、BoxLayout 以及 NULL 空布局。

3.3.1 FlowLayout 流布局

FlowLayout 流布局是将容器中的组件按照从左到右的顺序，流动地排列和分布，直到上方的空间被占满，才移动到下一行，继续从左到右流动排列。

FlowLayout 类的构造方法有如下三个：

- FlowLayout()——不带参数的构造方法，使用默认对齐方式（中间对齐）和默认间距（水平、垂直间距都为 5 像素）创建一个新的流布局管理器；
- FlowLayout(int align)——带有对齐方式参数的构造方法，用于创建一个指定对齐方式、默认间距为 5 像素的流布局管理器；
- FlowLayout(int align,int hgap,int vgap)——带有对齐方式、水平间距和垂直间距参数的构造方法，用于创建一个指定对齐方式、水平和垂直间距的流布局管理器。

FlowLayout 类提供了三个静态常量，用于指明布局的对齐方式，如表 3-3 所示。

表 3-3 FlowLayout 类中的对齐方式静态常量

常　量	描　述
FlowLayout.LEFT	左对齐
FlowLayout.CENTER	居中对齐，默认对齐方式
FlowLayout.RIGHT	右对齐

注意

FlowLayout 是面板的默认布局，即当创建一个面板 JPanel 对象且没有设定其布局管理器时，默认使用流布局。

下述代码演示 FlowLayout 流布局的使用。

【代码 3-3】 FlowLayoutDemo.java

```
package com.qst.chapter03;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class FlowLayoutDemo extends JFrame {
    // 声明组件
    private JPanel p;
    private JButton btn1, btn2, btn3;
    public FlowLayoutDemo() {
        super("FlowLayout 流布局");
        // 创建面板
        p = new JPanel();
        // 创建一个流布局对象，对齐方式是左对齐，水平间距 10 像素，垂直间距 15 像素
        FlowLayout layout = new FlowLayout(FlowLayout.LEFT, 10, 15);
```

```
        // 设置面板的布局
        p.setLayout(layout);
        // 上面三行代码可以简化成下面一条语句
        // p = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 15));
        //创建按钮
        btn1 = new JButton("按钮 1");
        btn2 = new JButton("按钮 2");
        btn3 = new JButton("按钮 3");
        //将按钮添加到面板中
        p.add(btn1);
        p.add(btn2);
        p.add(btn3);
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(200, 200);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new FlowLayoutDemo();
    }
}
```

在上述代码中,在创建面板和流布局对象后,使用面板对象的 setLayout()方法为面板指定布局的方式,此过程使用了如下三条语句:

```
// 创建面板
p = new JPanel();
// 创建一个流布局对象,对齐方式是左对齐,水平间距 10 像素,垂直间距 15 像素
FlowLayout layout = new FlowLayout(FlowLayout.LEFT, 10, 15);
// 设置面板的布局
p.setLayout(layout);
```

实际上,可以在面板的构造方法中直接指定布局管理器对象。因此,上面三行代码可以简化成一条语句,形式如下:

```
p = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 15));
```

运行结果如图 3-7 所示。

注意

当容器采用 FlowLayout 流布局时,改变窗体的大小,可以发现各组件的位置会随着窗体的变化而变化,且各组件大小不变保持原来的“最合适”大小。

图 3-7　FlowLayout 布局效果

3.3.2　BorderLayout 边界布局

BorderLayout 边界布局允许将组件有选择地放置到容器的中部、北部、南部、东部、西部这五个区域，如图 3-8 所示。

BorderLayout 类的构造方法如下：

- BorderLayout()——不带参数的构造方法，用于创建一个无间距的边界布局管理器对象；
- BorderLayout(int hgap,int vgap)——带参数的构造方法，用于创建一个指定水平、垂直间距的边界布局管理器。

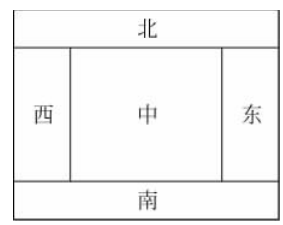


图 3-8　边界布局的五个区域

BorderLayout 类提供了五个静态常量，用于指明边界布局管理中的五个区域，如表 3-4 所示。

表 3-4　BorderLayout 类中的五个区域常量

常　　量	描　　述
BorderLayout.EAST	东部位置
BorderLayout.WEST	西部位置
BorderLayout.SOUTH	南部位置
BorderLayout.NORTH	北部位置
BorderLayout.CENTER	中央位置，该位置属于默认位置

一个容器使用 BorderLayout 边界布局后，当向容器中添加组件时，需要使用带两个参数的 add()方法，将指定组件添加到此容器的给定位置上。

【语法】

```
public Component add(Component comp,int index)
```

【示例】　将按钮添加到北部

```
p. add(btn, BorderLayout.NORTH);
```

当使用 BorderLayout 布局时，需要注意以下两点：

- 当向使用 BorderLayout 布局的容器中添加组件时，需要指定组件所放置的区域位

置，如果没有指定则默认放置到布局的中央位置；

- 通常一个区域位置只能添加一个组件，如果同一个区域中添加多个组件，则后放入的组件将会覆盖先放入的组件。

下述代码演示边界布局 BorderLayout 的使用。

【代码 3-4】 BorderLayoutDemo.java

```
package com.qst.chapter03;

import java.awt.BorderLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class BorderLayoutDemo extends JFrame {
    private JPanel p;
    private JButton btn1, btn2, btn3, btn4, btn5;
    public BorderLayoutDemo() {
        super("BorderLayout 边界布局");
        p = new JPanel();
        // 创建一个边界布局管理器对象,并将该布局设置到面板中
        p.setLayout(new BorderLayout());
        //创建按钮
        btn1 = new JButton("按钮 1");
        btn2 = new JButton("按钮 2");
        btn3 = new JButton("按钮 3");
        btn4 = new JButton("按钮 4");
        btn5 = new JButton("按钮 5");
        // 将按钮放置到面板中指定位置
        p.add(btn1, BorderLayout.EAST);
        p.add(btn2, BorderLayout.WEST);
        p.add(btn3, BorderLayout.SOUTH);
        p.add(btn4, BorderLayout.NORTH);
        p.add(btn5, BorderLayout.CENTER);
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(300, 200);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new BorderLayoutDemo();
    }
}
```

上述代码中，先创建了一个面板，并设置其布局方式为边界布局；再创建五个按钮，分别添加到面板的五个不同的区域。

运行结果如图 3-9 所示。

图 3-9　BorderLayout 布局效果

注意

BorderLayout 边界布局是窗体(JFrame)的默认布局。当容器采用边界布局时，改变窗体的大小，可以发现东西南北四个位置上的组件长度进行拉伸，而中间位置的组件进行扩展。

3.3.3　GridLayout 网格布局

GridLayout 网格布局就像表格一样，将容器按照行和列分割成单元格，每个单元格所占的区域大小都一样。当向使用 GridLayout 布局的容器中添加组件时，默认是按照从左到右、从上到下的顺序，依次将组件添加到每个网格中。与 FlowLayout 不同，放置在 GridLayout 布局中的各组件的大小由所处区域来决定，即每个组件将自动占满整个区域。

GridLayout 类提供了两个构造方法：

- GridLayout(int rows,int cols)——用于创建一个指定行数和列数的网格布局管理器；
- GridLayout(int rows,int cols,int hgap,int vgap)——用于创建一个指定行数、列数、水平间距和垂直间距的网格布局管理器。

下述代码用于演示网格布局 GridLayout 的使用。

【代码 3-5】　GridLayoutDemo.java

```
package com.qst.chapter03;

import java.awt.GridLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class GridLayoutDemo extends JFrame {
    private JPanel p;
```

```
    // 声明按钮数组
    private JButton[] btns;
    public GridLayoutDemo() {
        super("网格布局");
        // 创建一个 2 行 3 列的网格布局管理器对象,并将该布局设置到面板中
        p = new JPanel(new GridLayout(2, 3));
        // 实例化按钮数组的长度
        btns = new JButton[6];
        // 循环实例化数组中的每个按钮对象
        for (int i = 0; i < btns.length; i++) {
            btns[i] = new JButton("按钮 " + (i + 1));
        }
        // 循环将数组中的按钮添加到面板中
        for (int i = 0; i < btns.length; i++) {
            p.add(btns[i]);
        }
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(300, 200);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new GridLayoutDemo();
    }
}
```

上述代码使用“new GridLayout(2, 3)”创建一个 2 行 3 列的网格布局管理器对象,并定义了一个长度为 6 的按钮数组,然后使用了两次 for 循环:

- 第 1 个 for 循环分别实例化每个按钮对象;
- 第 2 个 for 循环将数组中的按钮添加到面板中。

运行结果如图 3-10 所示。

图 3-10 GridLayout 布局效果

注意

当容器采用网格布局时，改变窗体的大小，可以发现各组件的大小随着窗体的变化而均匀变化。

3.3.4 CardLayout 卡片布局

CardLayout 卡片布局将加入到容器中的组件看成一叠卡片，每次只能看见最上面的组件。因此，CardLayout 卡片布局是以时间而非空间来管理容器中的组件。

CardLayout 类提供了两个构造方法：

- CardLayout()——不带参数的构造方法，用于创建一个默认间距为 0 的新卡片布局管理器对象；
- CardLayout(int hgap,int vgap)——带参数的构造方法，用于创建一个指定水平和垂直间距的卡片布局管理器对象。

CardLayout 类中用于控制组件可见的 5 个常用方法如表 3-5 所示。

表 3-5 CardLayout 类中的显示方法

方 法	功 能 描 述
first(Container parent)	显示容器中的第一张卡片
last(Container parent)	显示容器中的最后一张卡片
previous(Container parent)	显示容器中当前卡片的上一张卡片
next(Container parent)	显示容器中当前卡片的下一张卡片
show(Container parent,String name)	显示容器中指定名称的卡片

一个容器使用 CardLayout 卡片布局后，当向容器中添加组件时，需要使用带两个参数的 add()方法，给组件指定一个名称并将其添加到容器中。

【语法】

```
public Component add(String name,Component comp)
```

【示例】 给按钮指定名称并添加到面板中

```
p. add("第 1 个",btn1);
```

下述代码演示 CardLayout 卡片布局的使用。

【代码 3-6】 CardLayoutDemo. java

```
package com.qst.chapter03;

import java.awt.CardLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class CardLayoutDemo extends JFrame {
```

```
    private JPanel p;
    // 声明按钮数组
    private JButton[ ] btns;
    // 声明卡片布局管理器
    private CardLayout cl;
    public CardLayoutDemo( ) {
        super("CardLayout 卡片布局");
        // 实例化卡片布局管理器对象
        cl = new CardLayout();
        // 实例面板对象,其布局为卡片布局
        p = new JPanel(cl);
        // 实例化按钮数组的长度
        btns = new JButton[6];
        // 循环实例化数组中的每个按钮对象
        for (int i = 0; i < btns.length; i++) {
            btns[i] = new JButton("按钮 " + (i + 1));
        }
        // 循环将数组中的按钮添加到面板中
        for (int i = 0; i < btns.length; i++) {
            p.add("第" + (i + 1) + "张", btns[i]);
        }
        // 将面板添加到窗体中
        this.add(p);
        // 显示最后一张卡片
        //cl.last(p);
        // 显示名称是"第 3 张"的卡片
        // cl.show(p, "第 3 张");
        // 设定窗口大小
        this.setSize(300, 200);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[ ] args) {
        new CardLayoutDemo();
    }
}
```

上述代码中使用 CardLayout 进行布局,通过 for 循环向容器中添加组件时 add()方法带两个参数,分别用来指明所添加的组件名称和组件对象。

运行结果如图 3-11 所示。

在代码中将已注释的行取消注释:

```
cl.last(p);
```

将显示 CardLayout 布局中的最后一张卡片，运行结果如图 3-12 所示。

图 3-11　CardLayout 显示第 1 张卡片的效果

图 3-12　CardLayout 显示最后一张卡片的效果

再将代码中已注释的行取消注释：

```
cl.show(p, "第 3 张");
```

将显示 CardLayout 布局中指定名称的卡片，运行结果如图 3-13 所示。

图 3-13　CardLayout 显示指定卡片的效果

注意

当容器采用卡片布局时，改变窗体的大小，可以发现卡片上的组件大小随着窗体的变化而变化，并始终占满整个窗口。

3.3.5　BoxLayout 盒布局

原来 AWT 中提供的 GridBagLayout 布局管理器虽然功能强大，但过于复杂，所以 Swing 引入了一个新的布局管理器——BoxLayout 盒布局。BoxLayout 盒布局保留了 GridBagLayout 的很多优点，使用起来也更加简单。

BoxLayout 可以在垂直和水平两个方向上摆放组件，构造方法如下：

```
BoxLayout(Container target, int axis)
```

其功能是创建一个基于 target 容器、沿给定轴放置组件的盒布局管理器对象。

BoxLayout 类提供的静态常量，用于指明组件的排列方向，如表 3-6 所示。

表 3-6 BoxLayout 类静态常量

常 量	描 述
BoxLayout. X_AXIS	X 轴(横向)排列
BoxLayout. Y_AXIS	Y 轴(纵向)排列
LINE_AXIS	根据容器的 ComponentOrientation 属性,按照文字在一行中的排列方式布置组件
PAGE_AXIS	根据容器的 ComponentOrientation 属性,按照文本行在一页中的排列方式布置组件

下述代码演示 BoxLayout 盒布局的使用。

【代码 3-7】 BoxLayoutDemo. java

```
package com.qst.chapter03;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class BoxLayoutDemo extends JFrame {
    private JPanel p;
    // 声明按钮数组
    private JButton[] btns;
    public BoxLayoutDemo() {
        super("BoxLayout 盒布局");
        // 实例面板对象
        p = new JPanel();
        // 设置面板的布局为盒布局
        p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));
        // 实例化按钮数组的长度
        btns = new JButton[6];
        // 循环实例化数组中的每个按钮对象
        for (int i = 0; i < btns.length; i++) {
            btns[i] = new JButton("按钮 " + (i + 1));
        }
        // 循环将数组中的按钮添加到面板中
        for (int i = 0; i < btns.length; i++) {
            p.add(btns[i]);
        }
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(300, 300);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new BoxLayoutDemo();
    }
}
```

上述代码创建并使用了 BoxLayout 盒布局，运行结果如图 3-14 所示。

图 3-14　BoxLayout 显示效果

3.3.6　NULL 空布局

在实际开发过程中，用户界面比较复杂，而且要求美观，单一使用 FlowLayout、BorderLayout、GridLayout、CardLayout 或 BoxLayout 布局都很难满足要求。此时，可以采用混合布局（多种布局管理器结合使用），或者直接采用 NULL 空布局。空布局是指容器不采用任何布局，而是通过每个组件的绝对定位进行布局。

使用 NULL 空布局的步骤如下：

（1）将容器中的布局管理器设置为 null（空），即容器不采用任何布局，例如：

```
//设置面板对象的布局为空
p. setLayout(null);
```

（2）调用 setBounds()设置组件的绝对位置坐标及大小，例如：

```
//设置按钮 x 轴坐标为 30,y 轴坐标为 60,宽度 40,高度 25(像素)
btn.setBounds(30,60,40,25);
```

或使用 setLocation()、setSize()分别设置组件的坐标和大小，例如：

```
//调用 setLocation()设置按钮的坐标
btn.setLocation(30,60);
//调用 setSize()设置按钮的大小
btn.setSize(40,25);
```

以上两种方式的效果是等价的。

（3）将组件添加到容器中

```
//将按钮添加到面板中
p. add(b);
```

下述代码用于演示 NULL 空布局的使用。

【代码 3-8】 NullDemo.java

```
package com.qst.chapter03;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class NULLLayoutDemo extends JFrame {
    private JPanel p;
    // 声明两个按钮对象
    private JButton btnOk, btnCancel;

    public NULLLayoutDemo() {
        super("NULL 空布局");
        p = new JPanel();
        // 设置面板布局为空
        p.setLayout(null);
        // 创建按钮
        btnOk = new JButton("确定");
        btnCancel = new JButton("取消");
        // 调用 setBounds()设置按钮的坐标和大小
        btnOk.setBounds(30, 60, 60, 25);
        // 调用 setLocation()设置按钮的坐标
        btnCancel.setLocation(100, 60);
        // 调用 setSize()设置按钮的大小
        btnCancel.setSize(60, 25);
        // 将按钮添加到面板中
        p.add(btnOk);
        p.add(btnCancel);
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(300, 300);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new NULLLayoutDemo();
    }
}
```

在上述代码中，两个按钮在面板中使用绝对位置进行定位：使用 setBounds()方法设置“确定”按钮的位置和大小，使用 setLocation()和 setSize()方法分别设置“取消”按钮的位置和大小。

运行结果如图 3-15 所示。

图 3-15　Null 空布局运行效果

注意

NULL 空布局一般用于组件之间位置相对固定，并且窗口不允许随便变换大小的情况，否则当窗口大小发生变化时，因所有组件都使用绝对位置定位，所以会产生组件整体“偏移”的情况。

3.4　事件处理

前面介绍了如何摆放组件，从而得到不同的图形界面，但这些界面还不能响应用户的任何操作。对于图形用户界面的应用程序，如果要实现用户界面的交互，必须通过事件处理。事件处理是指在事件驱动机制中，应用程序为响应事件而执行的一系列操作。事件驱动机制是指在图形界面应用程序中由用户操作而产生“事件”，例如单击鼠标或按下键盘的某个键。这种基于事件驱动机制的事件处理是目前实现与用户交互的最好方式。

3.4.1　Java 事件处理机制

在图形用户界面中，当用户使用鼠标单击按钮、在列表框进行选择或者单击窗口右上角的“×”关闭按钮时，都会触发一个相应的事件。

在 Java 事件处理体系结构中，主要涉及三种对象。

- 事件(Event)：在 Event 对象中封装了 GUI 组件所发生的特定事情，通常由用户的一次操作产生，而不是通过 new 运算符创建。事件包括键盘事件、鼠标事件等。Event 对象一般作为事件处理方法的参数，以便事件处理程序从中获取 GUI 组件上所发生的事件相关信息；
- 事件源(Event Source)：事件发生的场所，通常就是各个 GUI 组件，例如窗口、按钮、菜单等；
- 事件监听器(Event Listener)：负责监听事件源所产生的事件，并对事件做出响应处理。事件监听器对象需要实现监听接口 Listener 中所定义的事件处理方法；当事件

触发时，直接调用该事件对应的处理方法对此事件进行响应和处理。

用户操作产生一个事件后，事件将被事件源所绑定的事件监听器进行监听并捕获，随后事件监听器调用相应的事件处理方法进行处理。这种事件处理机制是一种委派式的事件处理方式，实现步骤如下：

(1) 委托——组件(事件源)将事件处理委托给特定的事件处理对象(事件监听器)；

(2) 通知——当组件(事件源)触发指定的事件时，就通知所委托的事件处理对象(事件监听器)；

(3) 处理——事件处理对象(事件监听器)调用相应的事件处理方法来处理该事件。

这种受委托处理事件的对象被称为“事件监听对象”，图形界面中的每个组件均可以针对特定的事件指定一个或多个事件监听对象，并由这些事件监听对象负责监听并处理事件。

Java 的事件处理机制如图 3-16 所示。

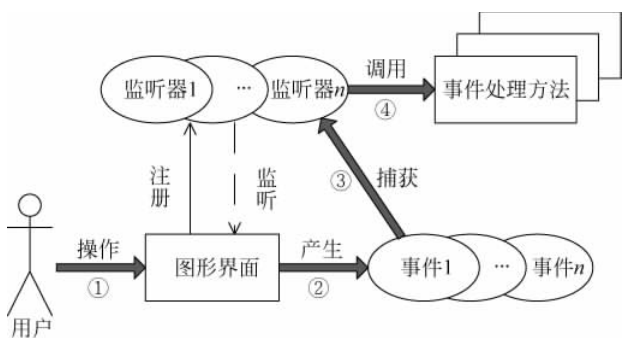


图 3-16 Java 事件处理机制

3.4.2 事件类

事件用于封装事件处理所必需的基本信息，包括事件源、事件信息等。AWT 中提供了丰富的事件类，用于封装不同组件上所发生的特定操作。所有 AWT 的事件类都是 AWTEvent 类的子类，而 AWTEvent 类又是 EventObject 类的子类。EventObject 类代表更广义的事件，其中包括 Swing 组件上所触发的事件、数据库连接所触发的事件。

AWTEvent 是所有 AWT 事件的根事件类，此类及其子类取代了原来的 java.awt.Event 类；AWTEvent 类的常用方法如表 3-7 所示。

表 3-7 AWTEvent 的常用方法列表

方 法	功 能 描 述
int getID()	返回事件类型
String paramString()	返回此 Event 状态的字符串，此方法仅在进行调试的时候使用
Object getSource()	从 EventObject 类继承的方法，用于返回事件源的对象

AWT 事件可以分为两大类：

- 低级事件——基于特定动作的事件，例如，鼠标进入、单击、拖放等动作，组件获得焦点、失去焦点时所触发的焦点事件；

- 高级事件——基于语义的事件，不与特定的动作关联，而依赖于触发此事件的类。例如，单击按钮和菜单、滑动滑动条、选中单选按钮等。

AWT事件包括低级事件类和高级事件类，常见的AWT事件类有MouseEven、ActionEvent、ItemEvent和WindowEvent等，具体如表3-8所示。

表3-8　AWT事件分类

分类	事件类	描述	事件源
低级事件	ComponentEvent	组件事件，当组件尺寸发生变化、位置发生移动、显示和隐藏状态发生改变时会触发该事件	所有组件
	ContainerEvent	容器事件，当往容器中添加组件、删除组件时会触发该事件	容器
	WindowEvent	窗口事件，当窗口状态发生改变，如打开、关闭、最大化、最小化窗口时会触发该事件	窗体
	FocusEvent	焦点事件，当组件获得或失去焦点时会触发该事件	能接受焦点的组件
	KeyEvent	键盘事件，当键盘按键被按下、松开、单击时会触发该事件	能接受焦点的组件
	MouseEvent	鼠标事件，当单击、按下、松开、移动鼠标时会触发该事件	所有组件
	PaintEvent	绘制事件，当GUI组件调用update/paint()方法来呈现自身时会触发该事件，该事件是一个特殊的事件类型，并非专用于事件处理模型	所有组件
高级事件	ActionEvent	动作事件，最常用的一个事件，当单击按钮、菜单时会触发该事件	按钮、列表、菜单项等
	AdjustmentEvent	调节事件，当调节滚动条时会触发该事件	滚动条
	ItemEvent	选项事件，当选择不同的选项时会触发该事件	列表、组合框等
	TextEvent	文本事件，当文本框或文本域中的文本发生变化时会触发该事件	文本框、文本域等

AWT事件类的继承层次关系如图3-17所示。

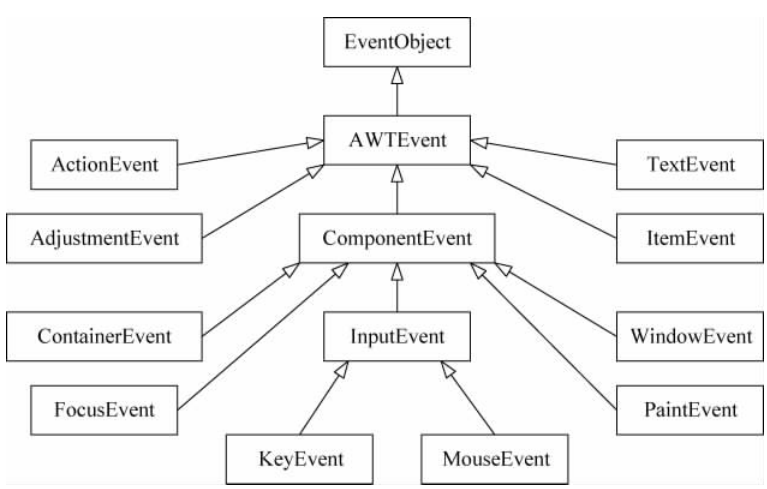


图3-17　AWT事件类继承层次

注意

开发过程中很少用到 EventObject、AWTEvent、InputEvent、PaintEvent 事件类，这些事件类通常作为事件的基类或系统内部类实现来使用。大多数事件类是 Swing 组件和 AWT 组件所共有的，这些事件类都定义在 java.awt.event 包中，而 javax.swing.event 包中的事件类是 Swing 组件所特有的事件类。

3.4.3 监听接口

对不同的事件需要使用不同的监听器进行监听，不同的监听器需要实现不同的监听接口。监听接口中定义了抽象的事件处理方法，这些方法能够针对不同的操作进行不同的处理。在程序中，通常使用监听类来实现监听接口中的事件处理方法。AWT 提供了大量的监听接口，用于实现不同类型的事件监听器，常用的监听接口及说明如表 3-9 所示。

表 3-9 常用的监听接口

监听接口	接口中声明的事件处理方法	功能描述
ActionListener	actionPerformed(ActionEvent e)	行为处理
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)	调节值改变
ItemListener	itemStateChanged(ItemEvent e)	选项值状态改变
FocusListener	focusGained(FocusEvent e)	获得聚焦
	focusLost(FocusEvent e)	失去聚焦
KeyListener	keyPressed(KeyEvent e)	按下键盘
	keyReleased(KeyEvent e)	松开键盘
	keyTyped(KeyEvent e)	敲击键盘
MouseListener	mouseClicked(MouseEvent e)	鼠标单击
	mouseEntered(MouseEvent e)	鼠标进入
	mouseExited(MouseEvent e)	鼠标退出
	mousePressed(MouseEvent e)	鼠标按下
	mouseReleased(MouseEvent e)	鼠标松开
MouseMotionListener	mouseDragged(MouseEvent e)	鼠标拖动
	mouseMoved(MouseEvent e)	鼠标移动
WindowListener	windowActivated(WindowEvent e)	窗体激活
	windowClosed(WindowEvent e)	窗体关闭以后
	windowClosing(WindowEvent e)	窗体正在关闭
	windowDeactivated(WindowEvent e)	窗体失去激活
	windowDeiconified(WindowEvent e)	窗体非最小化
	windowIconified(WindowEvent e)	窗体最小化
	windowOpened(WindowEvent e)	窗体打开后

监听接口与事件一样，通常都定义在 java.awt.event 包中，该包提供了不同类型的事件类和监听接口。

3.4.4 事件处理步骤

在 Java 程序中,实现事件处理需要以下三个步骤:

(1) 创建监听类,实现监听接口并重写监听接口中的事件处理方法;

(2) 创建监听对象,即实例化上一步中所创建的监听类的对象;

(3) 注册监听对象,调用组件的 addXXXListener()方法将监听对象注册到相应组件上,以便监听对事件源所触发的事件。

此处需要注意监听类、事件处理方法和监听对象之间的区别与联系。

- 监听类:是一个自定义的实现监听接口的类,监听类可以实现一个或多个监听接口。

【示例】 定义一个监听类实现 ActionListener 监听接口

```
class MyListener implements ActionListener{
    ...
}
```

- 事件处理方法:即监听接口中已经定义好的相应的事件处理方法,该方法是抽象方法,需要在创建监听类时重写接口中的事件处理方法,并将处理事件的业务代码放入到方法中。

【示例】 ActionListener 监听接口中的事件处理方法 actionPerformed()

```
class MyListener implements ActionListener {
        // 重写 ActionListener 接口中的事件处理方法 actionPerformed()
        public void actionPerformed(ActionEvent e) {
            ...
        }
}
```

- 监听对象:就是监听类的一个实例对象,该对象具有监听功能,前提是先将监听对象注册到事件源组件上,当操作该组件产生事件时,该事件将会被此监听对象捕获并调用相应的事件方法进行处理。

【示例】 创建监听对象并注册

```
//创建一个监听对象
MyListener listener = new MyListener();
// 注册监听
button.addActionListener(listener);
```

下述代码通过创建监听类来实现单击按钮改变面板的背景颜色。

【代码 3-9】 ChangeColor.java

```
package com.qst.chapter03;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class ChangeColor extends JFrame {
    JPanel p;
    JButton btnRed, btnGreen, btnYellow;
    public ChangeColor() {
        super("事件测试 - 改变颜色");
        // 创建组件
        p = new JPanel();
        btnRed = new JButton("红色");
        btnGreen = new JButton("绿色");
        btnYellow = new JButton("黄色");
        //2. 创建一个监听对象
        ButtonListener btnListener = new ButtonListener();
        //3. 注册监听
        btnRed.addActionListener(btnListener);
        btnGreen.addActionListener(btnListener);
        btnYellow.addActionListener(btnListener);
        // 将按钮添加到面板中
        p.add(btnRed);
        p.add(btnGreen);
        p.add(btnYellow);
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(300, 300);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    // 1. 创建扩展 ActionListener 的监听类
    // 该监听类是一个内部类,以便可以直接对外部类中的资源进行访问
    class ButtonListener implements ActionListener {
        // 重写 ActionListener 接口中的事件处理方法 actionPerformed()
        public void actionPerformed(ActionEvent e) {
            // 获取事件源
            Object source = e.getSource();
            // 判断事件源,进行相应的处理
            if (source == btnRed) {
                // 设置面板的背景颜色是红色 Color.red
                p.setBackground(Color.red);
            } else if (source == btnGreen) {
                // 设置面板的背景颜色是绿色 Color.green
                p.setBackground(Color.green);
            } else {
                // 设置面板的背景颜色是黄色 Color.yellow
                p.setBackground(Color.yellow);
            }
```

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

对于上述代码应注意以下几点：

- 定义的 ButtonListener 类是一个监听类，实现了 ActionListener 监听接口。另外，需要注意的是，ButtonListener 定义在 ColorChange 类体内，是一个内部类，以便可以直接访问其所在外部类的成员变量，如 btnRed、btnGreen、btnYello 等；
- actionPerformed()方法是动作事件的处理方法，该方法先获取事件源（用户操作的按钮），判断后再进行相应的处理；
- java.awt.Color 是一个基于标准 RGB 的颜色类，内部定义了一些颜色常量，例如 Color.red 或 Color.RED 都表示红色；
- 通过调用 setBackground()方法来设置面板的背景颜色；
- 在 ChangeColor()构造方法中创建一个 ButtonListener 类的对象 btnListener，该对象就是监听对象；然后调用按钮的 addActionListener()方法来注册监听对象，此时监听对象将监视着按钮是否被单击；当用户单击按钮时，监听对象 btnListener 会自动调用其 actionPerformed()方法进行处理；
- 一个监听对象可以对多个组件进行监听，此处监听对象 btnListener 监听了 btnRed、btnGreen 和 btnYello 三个按钮。

图 3-18　改变颜色

运行结果如图 3-18 所示，单击不同的按钮，观察面板颜色的变化。

通过上面改变面板颜色的案例，再一次验证了事件的处理机制，其事件产生及处理的过程如图 3-19 所示。

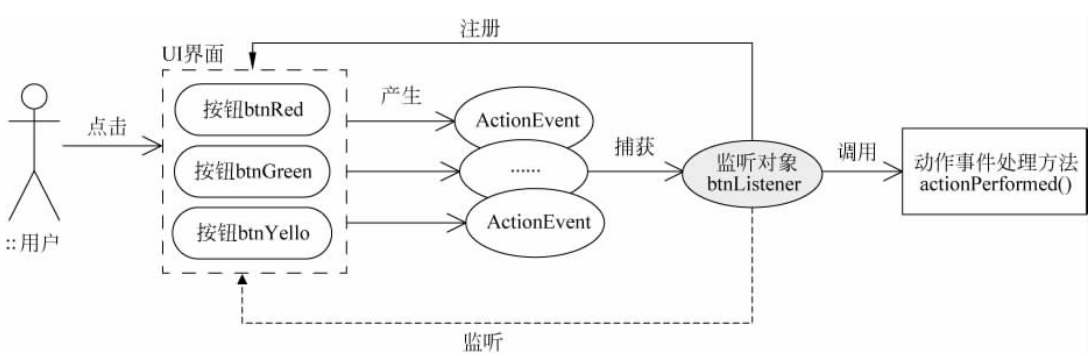


图 3-19　事件产生及处理过程

在事件处理的方式上，除了采用上面的这种通过定义事件监听器类的方式，还可以采用另外一种匿名的方式，即无须为事件监听器类进行命名，只需在注册的同时实现监听器接口及其方法即可。

下述代码通过匿名监听类的方式来实现单击按钮改变面板的背景颜色。

【代码 3-10】 ChangeColorAnonymous.java

```
package com.qst.chapter03;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class ChangeColorAnonymous extends JFrame {
    JPanel p;
    JButton btnRed, btnGreen, btnYellow;
    public ChangeColorAnonymous() {
        super("事件测试 - 改变颜色");
        // 创建组件
        p = new JPanel();
        btnRed = new JButton("红色");
        btnGreen = new JButton("绿色");
        btnYellow = new JButton("黄色");
        // 使用匿名监听类的方式注册监听
        btnRed.addActionListener(new ActionListener() {
            // 重写 ActionListener 接口中的事件处理方法 actionPerformed()
            public void actionPerformed(ActionEvent e) {
                // 设置面板的背景颜色是红色 Color.red
                p.setBackground(Color.red);
            }
        });
        btnGreen.addActionListener(new ActionListener() {
            // 重写 ActionListener 接口中的事件处理方法 actionPerformed()
            public void actionPerformed(ActionEvent e) {
                // 设置面板的背景颜色是绿色 Color.green
                p.setBackground(Color.green);
            }
        });
        btnYellow.addActionListener(new ActionListener() {
            // 重写 ActionListener 接口中的事件处理方法 actionPerformed()
            public void actionPerformed(ActionEvent e) {
                // 设置面板的背景颜色是黄色 Color.yellow
                p.setBackground(Color.yellow);
            }
        });
        // 将按钮添加到面板中
```

```
            p.add(btnRed);
            p.add(btnGreen);
            p.add(btnYellow);
            // 将面板添加到窗体中
            this.add(p);
            // 设定窗口大小
            this.setSize(300, 300);
            // 设定窗口左上角坐标
            this.setLocation(200, 100);
            // 设定窗口默认关闭方式为退出应用程序
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // 设置窗口可视(显示)
            this.setVisible(true);
        }
        public static void main(String[] args) {
            new ChangeColorAnonymous();
        }
    }
```

上述代码在给按钮注册监听对象时，直接 new 一个 ActionListener 对象，但因为接口不能直接实例化，所以在 new 一个对象的同时必须实现该接口中的 actionPerformed()事件处理方法。这种既没有给监听类进行命名，又没有给监听对象命名的方式，被称为匿名处理方式。

ChangeColorAnonymous.java 程序代码与 ChangeColor.java 程序代码所实现的功能效果是一致的，运行结果不再演示。

3.4.5 键盘事件

大多数窗体程序都是通过响应键盘事件来处理键盘输入。键盘事件 KeyEvent 的事件处理也是应用程序中经常用到的，例如用户敲击键盘的回车键下移光标位置等。

下述代码使用键盘方向键控制按钮移动。

【代码 3-11】 KeyEventDemo.java

```
package com.qst.chapter03;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class KeyEventDemo extends JFrame {
    private JPanel p;
    private JButton btnMove;
    public KeyEventDemo() {
        super("键盘事件 - 方向控制");
```

```
        // 创建组件
        p = new JPanel();
        btnMove = new JButton("走动");
        // 注册键盘监听
        btnMove.addKeyListener(new MyListener());
        // 将组件添加到面板中
        p.add(btnMove);
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(300, 300);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    // 定义一个监听类,实现键盘监听接口 KeyListener
    class MyListener implements KeyListener {
        // 敲击键盘的事件处理方法
        public void keyTyped(KeyEvent e) {
        }
        // 键盘按下的事件处理方法
        public void keyPressed(KeyEvent e) {
            // 获取按下键盘的码值
            int key = e.getKeyCode();
            // 获得按钮当前的 x,y 轴坐标
            int x = btnMove.getX();
            int y = btnMove.getY();
            if (key == KeyEvent.VK_RIGHT) {
                // 向右,x 轴坐标增加
                btnMove.setLocation(x + 5, y);
            } else if (key == KeyEvent.VK_LEFT) {
                // 向右,x 轴坐标减少
                btnMove.setLocation(x - 5, y);
            } else if (key == KeyEvent.VK_UP) {
                // 向右,y 轴坐标减少
                btnMove.setLocation(x, y - 5);
            } else if (key == KeyEvent.VK_DOWN) {
                // 向右,y 轴坐标增加
                btnMove.setLocation(x, y + 5);
            }
        }
        // 键盘松开的事件处理方法
        public void keyReleased(KeyEvent e) {
        }
    }
    public static void main(String[] args) {
        new KeyEventDemo();
    }
}
```

上述代码定义了一个监听内部类 MyListener，该类对 KeyListener 接口进行扩展，并重写 KeyListener 接口中的 keyTyped()、keyPressed()和 keyReleased()三个事件处理方法。需要注意的是，虽然只有 keyPressed()方法中有处理代码，其他两个方法没有处理代码，但这两个空的方法也必须保留。这是因为当一个非抽象类在实现一个接口时，该类必须重写所实现接口中的所有方法，哪怕有些方法不需要也必须重写；否则，该类就成为一个抽象类，不能被实例化。

KeyEvent 类是键盘事件类，该类对每个键盘按键都定义了一个整型常量与之对应，其中：

- KeyEvent. VK_UP 代表“↑”键；
- KeyEvent. VK_DOWN 代表“↓”键；
- KeyEvent. VK_RIGHT 代表方向键“→”键；
- KeyEvent. VK_LEFT 代表“←”键。

此外，KeyEvent 类还提供了以下两个常用的方法：

- intgetKeyCode()——获取键盘按键的码值，返回一个 int 类型的整数值，可以直接跟 KeyEvent 类中定义的整型常量进行比较，如 KeyEvent. VK_UP 等；
- chargetKeyChar()——获取键盘按键上的字符值，返回一个 char 类型的字符值，例如'A'、'M'等。

运行程序，使用键盘的方向键控制按钮移动，观察按钮位置的变化，结果如图 3-20 所示。

图 3-20　键盘方向键控制按钮移动

3.4.6　鼠标事件

鼠标事件 MouseEvent 有两个监听接口 MouseListener 和 MouseMotionListener，具体如下：

- MouseListener 监听接口专门用于监听鼠标的按下、松开、单击等操作；
- MouseMotionListener 监听接口则用于监听鼠标移动和拖动方面的操作。

下述代码演示鼠标拖动事件处理。

【代码 3-12】　MouseEventDemo. java

```
package com.qst.chapter03;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import javax.swing.JFrame;
```

```
import javax.swing.JPanel;
public class MouseEventDemo extends JFrame {
    private JPanel p;
    // 鼠标上一次的坐标
    int pre_x = -1, pre_y = -1;
    // 鼠标当前坐标
    int x, y;
    public MouseEventDemo() {
        super("画板");
        p = new JPanel();
        // 注册鼠标监听
        p.addMouseMotionListener(new PaintListener());
        p.addMouseListener(new ResetListenter());
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(400, 300);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    // 重写 JFrame 的 paint()方法,此方法用于在窗体中画图
    public void paint(Graphics g) {
        // 设置画笔的颜色
        g.setColor(Color.red);
        // 历史坐标>0
        if (pre_x > 0 && pre_y > 0) {
            // 绘制一条线段,从上一次鼠标拖动事件点到本次鼠标拖动事件点
            g.drawLine(pre_x, pre_y, x, y);
        }
        // 保存当前鼠标坐标,称为上一次的历史坐标
        pre_x = x;
        pre_y = y;
    }
    // 定义鼠标拖动监听类
    class PaintListener implements MouseMotionListener {
        // 鼠标移动的处理方法
        public void mouseMoved(MouseEvent e) {
        }
        // 鼠标拖动的处理方法,负责画画工作
        public void mouseDragged(MouseEvent e) {
            // 获取鼠标当前的坐标
            x = e.getX();
            y = e.getY();
            // 重画,repaint()触发 paint()
            MouseEventDemo.this.repaint();
        }
    }
```

```
        // 定义鼠标监听类
        class ResetListenter implements MouseListener {
            // 鼠标单击事件处理
            public void mouseClicked(MouseEvent e) {
            }
            // 鼠标按下事件处理
            public void mousePressed(MouseEvent e) {
                // 获取鼠标按键,判断是否是右键
                if (e.getButton() == MouseEvent.BUTTON3) {
                    // 重画面板(擦除原来的轨迹)
                    MouseEventDemo.this.p.repaint();
                }
            }
            // 鼠标松开事件处理,重置历史坐标
            public void mouseReleased(MouseEvent e) {
                // 鼠标松开时,将历史坐标重设为 -1(重置)
                pre_x = -1;
                pre_y = -1;
            }
            // 鼠标进入事件处理
            public void mouseEntered(MouseEvent e) {
            }
            // 鼠标退出事件处理
            public void mouseExited(MouseEvent e) {
            }
        }
        public static void main(String[] args) {
            new MouseEventDemo();
        }
    }
```

上述代码定义了两个监听类,这两个监听类都是内部类,其中:

- PaintListener 监听类实现了 MouseMotionListener 接口,并重写该接口中的 mouseDragged()事件处理方法,实现拖动鼠标可以画画的功能;
- ResetListenter 监听类实现了 MouseListener 接口,并重写该接口中的 mousePressed()和 mouseReleased()事件处理方法,实现按下鼠标右键能够擦除原来轨迹、松开鼠标按键能够重置历史坐标的功能。

调用 MouseEvent 类的 getButton()方法可以获取用户所单击的鼠标按键。此外,MouseEvent 类还提供了三个静态常量来标识鼠标的按键:

- MouseEvent.BUTTON1——鼠标左键;
- MouseEvent.BUTTON2——鼠标中间键(滑轮);
- MouseEvent.BUTTON3——鼠标右键。

上面这两个监听类需要实现监听接口中的所有方法,虽然只是使用了部分方法,但其他的方法依然需要重写,这种情况也可以采用适配器进行简化(参见 3.4.7 节的内容)。

当在窗体容器中画图时,需要重写窗体的 paint()方法,该方法的参数是 Graphics 类的对象。Graphics 类的对象 g 相当于一支画笔,能够设置其颜色,并提供了绘制直线、椭圆、图

像和多边形等功能。repaint()方法会自动触发 paint()方法,进行重画。

程序运行结果如图 3-21 所示。

图 3-21 鼠标画图

3.4.7 适配器

在实现监听接口时,虽然有些事件处理方法不是必需的,但也必须重写该接口中所有的方法。出于简化代码的目的,在 java.awt.event 包中还提供了一套抽象适配器类,分别来实现各事件处理的监听接口。通过继承适配器类,创建的监听类可以仅重写需要的事件处理方法,其他未用到的事件处理方法不需要重写,使代码更加简洁。

常见的适配器类如表 3-10 所示。

表 3-10 适配器类

适 配 器 类	实现的监听接口
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

下述代码通过使用适配器简化 3.4.6 节中实现画图功能的 MouseEventDemo 程序。

【代码 3-13】 MouseEventAdapterDemo.java

```
package com.qst.chapter03;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;

import javax.swing.JFrame;
import javax.swing.JPanel;
```

```
public class MouseEventAdapterDemo extends JFrame {
    private JPanel p;
    // 鼠标上一次的坐标
    int pre_x = -1, pre_y = -1;
    // 鼠标当前坐标
    int x, y;
    public MouseEventAdapterDemo() {
        super("画板");
        p = new JPanel();
        // 注册鼠标监听
        p.addMouseMotionListener(new PaintListener());
        p.addMouseListener(new ResetListenter());
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(400, 300);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    // 重写 JFrame 的 paint()方法,此方法用于在窗体中画图
    public void paint(Graphics g) {
        // 设置画笔的颜色
        g.setColor(Color.red);
        // 历史坐标> 0
        if (pre_x > 0 && pre_y > 0) {
            // 绘制一条线段,从上一次鼠标拖动事件点到本次鼠标拖动事件点
            g.drawLine(pre_x, pre_y, x, y);
        }
        // 保存当前鼠标坐标,称为上一次的历史坐标
        pre_x = x;
        pre_y = y;
    }
    // 定义鼠标拖动监听类
    class PaintListener extends MouseMotionAdapter{
        // 鼠标拖动的处理方法,负责画画工作
        public void mouseDragged(MouseEvent e) {
            // 获取鼠标当前的坐标
            x = e.getX();
            y = e.getY();
            // 重画,repaint()触发 paint()
            MouseEventAdapterDemo.this.repaint();
        }
    }
    // 定义鼠标监听类
    class ResetListenter extends MouseAdapter {
        // 鼠标按下事件处理
```

```
            public void mousePressed(MouseEvent e) {
                // 获取鼠标按键,判断是否是右键
                if (e.getButton() == MouseEvent.BUTTON3) {
                    // 重画面板(擦除原来的轨迹)
                    MouseEventAdapterDemo.this.p.repaint();
                }
            }
            // 鼠标松开事件处理,重置历史坐标
            public void mouseReleased(MouseEvent e) {
                // 鼠标松开时,将历史坐标重设为 -1(重置)
                pre_x = -1;
                pre_y = -1;
            }
        }
        public static void main(String[] args) {
            new MouseEventAdapterDemo();
        }
    }
```

上述代码定义的两个监听类直接继承了适配器类,只需重写所需的事件处理方法,简化了程序代码。运行结果一样,此处不再展示。

3.5 基本组件

GUI 图形界面是由一些基本的组件在布局管理器的统一控制下组合而成的,常用的基本组件包括图标、按钮、标签、文本组件、列表框、单选按钮、复选框和组合框等。

3.5.1 Icon 图标

Icon 是一个图标接口,用于加载图片。ImageIcon 类是 Icon 接口的一个实现类,用于加载指定的图片文件,通常加载的图片文件为 gif、jpg、png 等格式。

ImageIcon 类常用的构造方法如下:

- ImageIcon()——创建一个未初始化的图标对象;
- ImageIcon(Image image)——根据图像创建图标对象;
- ImageIcon(String filename)——根据指定的图片文件创建图标对象。

ImageIcon 类常用的方法如表 3-11 所示。

表 3-11 ImageIcon 类的常用方法

方 法	功 能 描 述
public int getIconWidth()	获得图标的宽度
public int getIconHeight()	获得图标的高度
public Image getImage()	返回此图标的 Image 图像对象
public void setImage(Image image)	设置图标所显示的 Image 图像
public void paintIcon(Component c,Graphics g,int x,int y)	绘制图标

在 Eclipse 项目中，当使用到图片文件时通常先将图片文件复制到自定的一个文件目录中，如图 3-22 所示，将图片文件复制到 images 目录下。

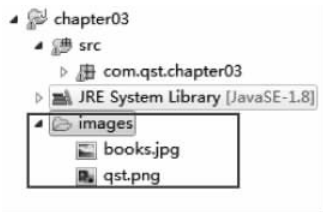


图 3-22　images 目录存放图片

下述代码演示 ImageIcon 的使用。

【代码 3-14】　ImageIconDemo.java

```
package com.qst.chapter03;

import java.awt.Graphics;

import javax.swing.ImageIcon;
import javax.swing.JFrame;

public class ImageIconDemo extends JFrame {
    public ImageIconDemo() {
        super("ImageIcon 图标");
        //创建 ImageIcon 图标
        ImageIcon qstIcon = new ImageIcon("images\\qst.png");
        //设置窗体的 Icon
        this.setIconImage(qstIcon.getImage());
        // 设定窗口大小(宽度 400 像素,高度 300 像素)
        this.setSize(400, 300);
        // 设定窗口左上角坐标(X 轴 200 像素,Y 轴 100 像素)
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public void paint(Graphics g) {
         //创建 ImageIcon 图标
        ImageIcon booksIcon  = new ImageIcon("images\\books.jpg");
        //在窗体中画图标
        g.drawImage(booksIcon.getImage(), 0, 20, this);
        //显示图标的宽度和高度
        g.drawString("宽:" + booksIcon.getIconWidth()
                + "px,高:" + booksIcon.getIconHeight() + "px", 20,210);
    }
    public static void main(String[] args) {
        new ImageIconDemo();
    }
}
```

在上述代码中，使用 ImageIcon 创建了两个图片对象，其中 qstIcon 对象用于设置窗口左上角的图标；而 booksIcon 对象则被绘制到 JFrame 窗口中。

注意

> 对 JFrame 窗口的 paint()方法进行重写，实现在窗口中绘制图片和字符串。图片文件名中包含路径，其中“\\”是转义字符，代表“\”。

运行结果如图 3-23 所示。

图 3-23 显示图标

3.5.2 JButton 按钮

JButton 类提供一个可接受单击操作的按钮功能，单击按钮会使其处于“下压”形状，松开后按钮会又恢复原状。在按钮中可以显示字符串、图标或两者同时显示。

JButton 类的构造方法如下：

- JButton(String str)——参数 str 是字符串，用于创建一个指定文本的按钮对象；
- JButton(Icon icon)——参数 icon 是一个图标对象，用于创建一个指定图标的按钮对象；
- JButton(String str,Icon icon)——该构造方法带有字符串和图标两个参数，用于创建一个指定文本和图标的按钮对象。

JButton 类常用的方法如表 3-12 所示。

表 3-12 JButton 类的常用方法

方 法	功 能 描 述
String getText()	获取按钮上的文本内容
void setText(String str)	设置按钮上的文本内容
void setIcon(Icon icon)	设置按钮上的图标

下述代码演示 JButton 类的使用。

【代码 3-15】 JButtonDemo.java

```
package com.qst.chapter03;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class JButtonDemo extends JFrame {
    // 声明组件
    private JPanel p;
    private JButton btnTxt, btnImg;
    private int num = 0;
    public JButtonDemo() {
        super("JButton 类");
        // 实例化面板对象 p(默认为流布局)
        p = new JPanel();
        // 实例化一个按钮对象,该按钮上显示文字
        btnTxt = new JButton("您单击了 0 次按钮!");
        // 实例化一个按钮对象,该按钮上显示图标
        btnImg = new JButton(new ImageIcon("images\\configure.png"));
        // 注册监听
        btnTxt.addActionListener(new ActionListener() {
            // 行为事件处理方法
            public void actionPerformed(ActionEvent e) {
                // 统计单击按钮的此书
                num++;
                // 改变按钮的文本
                btnTxt.setText("您单击了" + num + "次按钮!");
            }
        });
        btnImg.addMouseListener(new MouseAdapter() {
            // 鼠标按下事件处理,按下左右键加载不同的图片
            public void mousePressed(MouseEvent e) {
                // 获取鼠标按键,判断是否是左键
                if (e.getButton() == MouseEvent.BUTTON1) {
                    // 改变按钮的 Icon
                    btnImg.setIcon(new ImageIcon("images\\download.png"));
                }
                // 获取鼠标按键,判断是否是右键
                if (e.getButton() == MouseEvent.BUTTON3) {
                    // 改变按钮的 Icon
                    btnImg.setIcon(new ImageIcon("images\\configure.png"));
                }
            }
        });
```

```
            // 将按钮添加到面板中
            p.add(btnTxt);
            p.add(btnImg);
            // 将面板添加到窗体中
            this.add(p);
            // 设定窗口大小
            this.setSize(600, 640);
            // 设定窗口左上角坐标
            this.setLocation(200, 100);
            // 设定窗口默认关闭方式为退出应用程序
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // 设置窗口可视(显示)
            this.setVisible(true);
        }
        public static void main(String[] args) {
            new JButtonDemo();
        }
    }
```

上述代码定义了两个按钮：一个用于显示文本，另一个显示图片。两个按钮都添加了事件处理，当单击 btnTxt 按钮时，按钮上的文本会显示按钮被单击的次数；当在 btnImg 按钮上单击鼠标的左右按键时，会加载不同的图片。

运行结果如图 3-24 所示，操作两个按钮，注意文本和图片的变化。

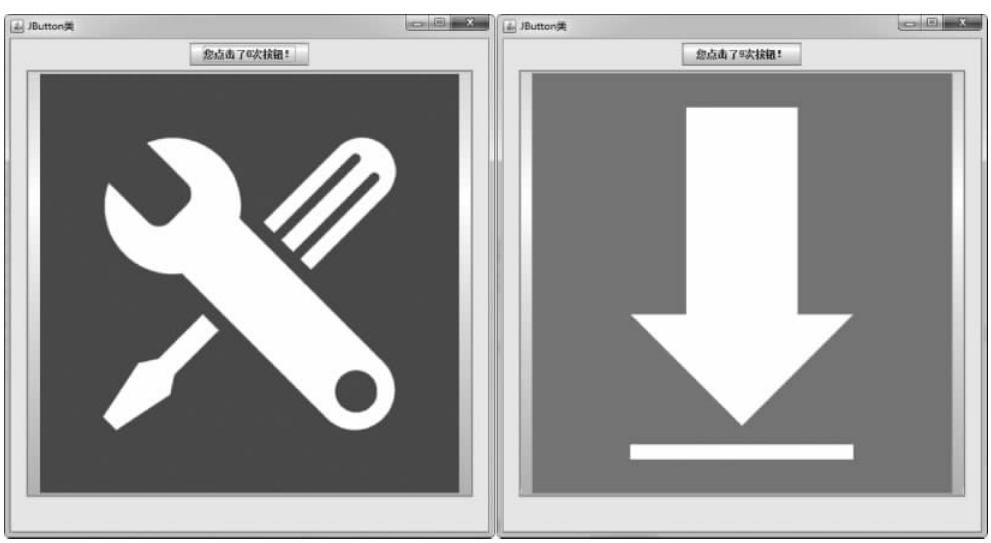


图 3-24 操作按钮

3.5.3 JLabel 标签

JLabel 标签具有标识和提示作用，可以显示文字或图标。标签没有边界，也不会响应用户操作，即单击标签是没有反应的。在 GUI 编程中，标签通常放在文本框、文本域、组合框等不带标签的组件前，对用户进行提示。

JLabel 类的构造方法如下：

- JLabel(String text)——用于创建一个指定文本的标签对象；
- JLabel(Icon icon)——用于创建一个指定图标的标签对象；
- JLabel(String text, Icon icon, int horizontalAlignment)——用于创建一个指定文本、图标和对齐方式的标签对象。

JLabel 类的常用的方法如表 3-13 所示。

表 3-13　JLabel 类的常用方法

方　　法	功 能 描 述
void setText(String txt)	设置标签中的文本内容
void setIcon(Icon icon)	设置标签中的图标
String getText()	获取标签中的文本内容

【代码 3-16】　JLabelDemo.java

```
package com.qst.chapter03;

import java.awt.GridLayout;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

public class JLabelDemo extends JFrame {
    // 声明组件
    private JPanel p;
    private JLabel lblTxt, lblImg, lblTxtImg;
    public JLabelDemo() {
        super("JLabel 类");
        // 实例化面板对象 p,面板布局是网格布局(3 行 1 列)
        p = new JPanel(new GridLayout(3, 1));
        // 实例化一个标签对象,显示文字
        lblTxt = new JLabel("这是一个文本标签");
        // 实例化一个标签对象,显示图标
        lblImg = new JLabel(new ImageIcon("images\\logo.png"));
        // 实例化一个标签对象,显示文本和标签
        lblTxtImg = new JLabel("商标", new ImageIcon("images\\logo.png"),
                SwingConstants.CENTER);
        // 将按钮添加到面板中
        p.add(lblTxt);
        p.add(lblImg);
        p.add(lblTxtImg);
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(400, 300);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
```

```
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new JLabelDemo();
    }
}
```

上述代码使用 JLabel 类的不同构造方法实例化了三个标签对象，分别用于显示文本标签、图片标签以及文本图片标签。运行结果如图 3-25 所示。

图 3-25 标签显示

3.5.4 文本组件

文本组件可以接收用户输入的文本内容。

Swing 常用的文本组件有以下三种：

- JTextField——文本框，该组件只能接收单行的文本输入；
- JTextArea——文本域，该组件可以接收多行的文本输入；
- JPasswordField——密码框，不显示原始字符，用于接收用户输入的密码。

JTextField 和 JTextArea 都属于 JTextComponent 类的子类。JTextComponent 是一个抽象类，该类定义了文本组件通用的方法及特性。

JTextField 类常用的构造方法及其他方法如表 3-14 所示。

表 3-14 JTextField 的方法列表

方 法	功 能 描 述
JTextField(int cols)	构造方法，用于创建一个内容是空的、指定长度的文本框
JTextField(String str)	构造方法，用于创建一个指定文本内容的文本框
JTextField(String s, int cols)	构造方法，用于创建一个指定文本内容的、指定长度的文本框
String getText()	获取文本框中用户输入的文本内容
void setText(String str)	设置文本框中的文本内容为指定字符串内容

JTextArea 文本域组件可以编辑多行多列文本，且具有换行能力。JTextArea 类常用的构造方法及其他方法如表 3-15 所示。

表 3-15　JTextArea 的方法列表

方　　法	功 能 说 明
JTextArea(int rows,int columns)	构造方法，用于创建一个内容是空的、指定行数及列数的文本域
JTextArea(String text)	构造方法，用于创建一个指定文本内容的文本域
JTextArea(String text, int rows, int columns)	构造方法，用于添加组件创建一个指定文本内容的、指定行数及列数的文本域
String getText()	获取文本域中用户输入的文本内容
void setText(String str)	设置文本域中的文本内容为指定字符串内容

JPasswordField 是 JTextField 类的子类，允许编辑单行文本，密码框用于接收用户输入的密码，但不显示原始字符，而是以特殊符号(掩码)形式显示。JPasswordField 类常用的构造方法及其他方法如表 3-16 所示。

表 3-16　JPasswordField 的方法列表

方　　法	功 能 说 明
JPasswordField (int cols)	构造方法，用于创建一个内容是空的、指定长度的密码框
JPasswordField (String str)	构造方法，用于创建一个指定密码信息的密码框
JPasswordField (String s, int cols)	构造方法，用于创建一个指定密码信息的、指定长度的密码框
char[] getPassword()	获取密码框中用户输入的密码，以字符型数组形式返回
setEchoChar(char c)	设置密码框中显示的字符为指定的字符

下述代码创建一个用户注册界面，以此对所学的文本组件进行练习。

【代码 3-17】　Login.java

```
package com.qst.chapter03;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextArea;
import javax.swing.JTextField;

//文本组件
public class TextComponentDemo extends JFrame {
    // 声明组件
    private JPanel p;
    private JLabel lblName, lblPwd, lblRePwd, lblAddress, lblMsg;
    // 声明一个文本框
```

```
private JTextField txtName;
// 声明两个密码框
private JPasswordField txtPwd, txtRePwd;
// 声明一个文本域
private JTextArea txtAddress;
private JButton btnReg, btnCancel;
public TextComponentDemo() {
    super("注册新用户");
    // 创建面板,面板的布局为 NULL
    p = new JPanel(null);
    // 实例化 5 个标签
    lblName = new JLabel("用户名");
    lblPwd = new JLabel("密 码");
    lblRePwd = new JLabel("确认密码");
    lblAddress = new JLabel("地 址");
    // 显示信息的标签
    lblMsg = new JLabel();
    // 设置标签的文字颜色是红色
    lblMsg.setForeground(Color.red);
    // 创建一个长度为 20 的文本框
    txtName = new JTextField(20);
    // 创建两个密码框,长度 20
    txtPwd = new JPasswordField(20);
    txtRePwd = new JPasswordField(20);
    // 设置密码框显示的字符为 *
    txtPwd.setEchoChar('*');
    txtRePwd.setEchoChar('*');
    // 创建一个文本域
    txtAddress = new JTextArea(20, 2);
    // 创建两个按钮
    btnReg = new JButton("注册");
    btnCancel = new JButton("清空");
    // 注册确定按钮的事件处理
    btnReg.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // 设置信息标签为空,清空原来历史信息
            lblMsg.setText("");
            // 获取用户输入的用户名
            String strName = txtName.getText();
            if (strName == null || strName.equals("")) {
                lblMsg.setText("用户名不能为空!");
                return;
            }
            // 获取用户输入的密码
            String strPwd = new String(txtPwd.getPassword());
            if (strPwd == null || strPwd.equals("")) {
                lblMsg.setText("密码不能为空!");
                return;
            }
            // 获取用户输入的确认密码
```

```
            String strRePwd = new String(txtRePwd.getPassword());
            if (strRePwd == null || strRePwd.equals("")) {
                lblMsg.setText("确认密码不能为空!");
                return;
            }
            // 判断确认密码是否跟密码相同
            if (!strRePwd.equals(strPwd)) {
                lblMsg.setText("确认密码与密码不同!");
                return;
            }
            // 获取用户输入的地址
            String strAddress = new String(txtAddress.getText());
            if (strAddress == null || strAddress.equals("")) {
                lblMsg.setText("地址不能为空!");
                return;
            }
            lblMsg.setText("注册成功!");
        }
    });
    // 取消按钮的事件处理
    btnCancel.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // 清空所有文本组件中的文本
            txtName.setText("");
            txtPwd.setText("");
            txtRePwd.setText("");
            txtAddress.setText("");
            // 设置信息标签为空
            lblMsg.setText("");
        }
    });
    // 定位所有组件
    lblName.setBounds(30, 30, 60, 25);
    txtName.setBounds(95, 30, 120, 25);
    lblPwd.setBounds(30, 60, 60, 25);
    txtPwd.setBounds(95, 60, 120, 25);
    lblRePwd.setBounds(30, 90, 60, 25);
    txtRePwd.setBounds(95, 90, 120, 25);
    lblAddress.setBounds(30, 120, 60, 25);
    txtAddress.setBounds(95, 120, 120, 50);
    lblMsg.setBounds(60, 185, 180, 25);
    btnReg.setBounds(60, 215, 60, 25);
    btnCancel.setBounds(125, 215, 60, 25);
    // 将组件添加到面中
    p.add(lblName);
    p.add(txtName);
    p.add(lblPwd);
    p.add(txtPwd);
    p.add(lblRePwd);
    p.add(txtRePwd);
```

```
            p.add(lblAddress);
            p.add(txtAddress);
            p.add(lblMsg);
            p.add(btnReg);
            p.add(btnCancel);
            // 将面板添加到窗体中
            this.add(p);
            // 设定窗口大小
            this.setSize(280, 300);
            // 设定窗口左上角坐标(X 轴 200 像素,Y 轴 100 像素)
            this.setLocation(200, 100);
            // 设定窗口默认关闭方式为退出应用程序
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // 设置窗口可视(显示)
            this.setVisible(true);
        }
        public static void main(String[] args) {
            new TextComponentDemo();
        }
    }
```

上述代码实现了用户注册界面,面板采用 NULL 空布局,因此需要对各组件进行定位。注册信息包括用户名、密码、确认密码和地址。另外,还定义了一个用于显示提示信息的标签 lblMsg 对象,为了让提示信息更醒目,通过 setForeground()方法来设置该标签的文字颜色为红色。“注册”和“清空”按钮都采用匿名方式注册了监听:当单击 btnReg 注册按钮时,获取用户输入的信息,并对信息进行验证;当单击 btnCancel 取消按钮时,清空所有文本组件中的文本内容和提示信息。

运行结果如图 3-26 所示,在文本控件中输入信息,单击按钮,观察提示信息。

图 3-26 注册窗口中的文本组件

3.5.5 JComboBox 组合框

JComboBox 组合框是一个文本框和下拉列表的组合,用户可以从下拉列表选项中选择一个选项。JComboBox 类常用的构造方法如下:

- JComboBox()——不带参数的构造方法,用于创建一个没有选项的组合框;
- JComboBox(Object[] listData)——构造方法的参数是对象数组,用于创建一个选项列表为对象数组中的元素的组合框;
- JComboBox(Vector<? > listData)——构造方法的参数是泛型向量,用于创建一个选项列表为向量集合中的元素的组合框。

JComboBox 类常用的方法及功能如表 3-17 所示。

表 3-17　JComboBox 类的常用方法

方　　法	功 能 说 明
intgetSelectedIndex()	获取用户选中的选项的下标(从 0 开始)
ObjectgetSelectedValue()	获得用户选中的选项值
void addItem(Object obj)	添加一个新的选项内容
void removeAllItems()	从项列表中移除所有项

下述代码演示组合框的使用。

【代码 3-18】　JComboBoxDemo. java

```
package com.qst.chapter03;

import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class JComboBoxDemo extends JFrame {
    private JPanel p;
    private JLabel lblProvince, lblCity;
    private JComboBox cmbProvince, cmbCity;
    public JComboBoxDemo() {
        super("组合框联动");
        p = new JPanel();
        lblProvince = new JLabel("省份");
        lblCity = new JLabel("城市");
        // 创建组合框,并使用字符串数组初始化其选项列表
        cmbProvince = new JComboBox(new String[] { "北京", "上海", "山东", "安徽" });
        // 创建一个没有选项的组合框
        cmbCity = new JComboBox();
        // 注册监听
        cmbProvince.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                // 获取用户选中的选项下标
                int i = cmbProvince.getSelectedIndex();
                // 清空组合框中的选项
                cmbCity.removeAllItems();
                // 根据用户选择的不同省份,组合框中添加不同的城市
                switch (i) {
                case 0:
                    cmbCity.addItem("北京");
                    break;
                case 1:
                    cmbCity.addItem("上海");
                    break;
```

```
                case 2:
                    cmbCity.addItem("济南");
                    cmbCity.addItem("青岛");
                    cmbCity.addItem("烟台");
                    cmbCity.addItem("潍坊");
                    cmbCity.addItem("威海");
                    break;
                case 3:
                    cmbCity.addItem("合肥");
                    cmbCity.addItem("芜湖");
                    cmbCity.addItem("淮北");
                    cmbCity.addItem("蚌埠");
                    break;
                }
            }
        });
        p.add(lblProvince);
        p.add(cmbProvince);
        p.add(lblCity);
        p.add(cmbCity);
        // 将面板添加到窗体中
        this.add(p);
        // 设定窗口大小
        this.setSize(300, 200);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new JComboBoxDemo();
    }
}
```

上述代码创建了两个组合框：第一个组合框是省份列表，第二个组合框是城市类别；根据用户选择省份的不同，城市列表相应的进行变化，即实现两个组合框之间的联动效果。调用 addItemListener()方法能够在组合框中注册选项监听，当选项改变时会触发选项事件处理。

运行结果如图 3-27 所示。

图 3-27 组合框联动

3.5.6 JList 列表框

JList 列表框中的选项以列表的形式都显示出来，用户在列表框中可以选择一个或多个选项(按住 Ctrl 键才能选中多个)。

JList 类常用的构造方法如下：

- JList()——不带参数的构造方法，用于创建一个没有选项的列表框；
- JList(Object[] listData)——构造方法的参数是对象数组，用于创建一个选项列表为对象数组中的元素的列表框；
- JList(Vector<? > listData)——构造方法的参数是泛型向量，用于创建一个选项列表为向量集合中的元素的列表框。

JList 类的常用的方法如表 3-18 所示。

表 3-18 JList 类的常用方法

方 法	功 能 描 述
intgetSelectedIndex()	获得选中选项的下标，此时用户选择一个选项
ObjectgetSelectedValue()	获得列表中用户选中的选项的值
Object[]getSelectedValues()	以对象数组的形式返回所有被选中选项的值
void setModel(ListModel m)	设置表示列表内容或列表“值”的模型
voidsetSelectionMode (int selectionMode)	设置列表的选择模式，三种选择模式： • ListSelectionModel. SINGLE_SELECTION 单选 • ListSelectionModel. SINGLE_INTERVAL_SELECTION 一次只能选择一个连续间隔 • ListSelectionModel. MULTIPLE_INTERVAL_SELECTION 多选(默认)

下述代码演示列表框的使用。

【代码 3-19】 JListDemo. java

```
package com.qst.chapter03;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.ListSelectionModel;

public class JListDemo extends JFrame {
    private JPanel p;
    private JList listLeft, listRight;
    private JButton btnOk, btnCancel;
    DefaultListModel model;
```

```
    public JListDemo() {
        super("JList 列表");
        this.setLayout(new GridLayout(1, 3));
        // 创建组件
        p = new JPanel(new GridLayout(2, 1));
        // 创建列表,并使用一个字符串数组初始化其选项列表
        listLeft = new JList(
                new String[] { "看书", "写字", "画画", "爬山", "跑步", "游泳" });
        // 设置列表的选择模式为单选
        listLeft.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        // 创建一个空的列表
        listRight = new JList();
        // 定义一个默认的列表值模型
        model = new DefaultListModel();
        // 设置列表的值模型
        listRight.setModel(model);
        btnOk = new JButton("---->");
        btnCancel = new JButton("<----");
        // 注册监听
        btnOk.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // 获取用户在左侧列表中选中的选项
                String strSelect = listLeft.getSelectedValue().toString();
                // 添加到右侧
                model.addElement(strSelect);
            }
        });
        btnCancel.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // 移除用户在右侧列表选中的选项
                model.remove(listRight.getSelectedIndex());
            }
        });
        // 将组件添加到容器中
        p.add(btnOk);
        p.add(btnCancel);
        this.add(listLeft);
        this.add(p);
        this.add(listRight);

        // 设定窗口大小
        this.setSize(300, 200);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new JListDemo();
    }
}
```

上述代码中创建了两个列表，其中在创建右侧的列表对象时，使用到了列表的值模型类DefaultListModel。DefaultListModel类用于创建一个默认的列表值模型，使用值模型可以对列表中的选项内容进行添加或删除操作：

- addElement()方法可以向列表中添加一个新的选项；
- remove()方法可以移除列表中选中的选项。

整个界面使用了嵌套的网格布局来组织所有的组件，运行结果如图3-28所示，操作左右两侧的列表，观察添加和删除操作的效果。

图3-28　添加和删除列表框中的选项

3.5.7　JRadioButton单选按钮

JRadioButton单选按钮可被选择或被取消选择。JRadioButton类常用的构造方法如下：

- JRadioButton(String str)——用于创建一个具有指定文本的单选按钮；
- JRadioButton(String str,boolean state)——创建一个具有指定文本和选择状态的单选按钮；当选择状态为true时，表示单选按钮被选中；状态为false时表示未被选中。

JRadioButton类常用的方法如表3-19所示。

表3-19　JRadioButton类常用的方法

方　法	功能描述
void setSelected(boolean state)	设置单选按钮的选中状态
boolean isSelected()	判断单选按钮是否被选中，返回一个布尔值

单选按钮一般成组出现，且需与ButtonGroup按钮组配合使用后，才能实现单选规则，即一次只能选择按钮组中的一个按钮。因此，使用单选按钮要经过以下两个步骤：

(1) 先实例化所有的JRadioButton单选按钮对象；

(2) 再创建一个ButtonGroup按钮组对象，并用其add()方法将所有的单选按钮添加到该组中，实现单选规则。

【示例】　创建单选按钮，并使用按钮组实现单选规则

```
// 创建单选按钮
JRadioButton rbMale = new JRadioButton("男", true);
JRadioButton rbFemale = new JRadioButton("女");
// 创建按钮组
```

```
ButtonGroup bg = new ButtonGroup();
// 将 rb1 和 rb2 两个单选按钮添加到按钮组中,这两个单选按钮只能选中其一
bg.add(rbMale);
bg.add(rbFemale);
```

注意

ButtonGroup 只是为了实现单选规则的逻辑分组,而不是物理上的分组,因此仍要将 JRadioButton 单选按钮对象添加到容器对象中。

3.5.8 JCheckBox 复选框

JCheckBox 复选框可以控制选项的开启或关闭;在复选框上单击时,可以改变复选框的状态,复选框可以被单独使用或作为一组使用。

JCheckBox 类的构造方法如下:

- JCheckBox(String str)——创建一个带文本的、最初未被选定的复选框;
- JCheckBox(String str, boolean state)——创建一个带文本的复选框,并指定其最初是否处于选定状态。

JCheckBox 类常用的方法如表 3-20 所示。

表 3-20 JCheckBox 类常用的方法

方 法	功 能 描 述
void setSelected(boolean state)	设置复选框的选中状态
boolean isSelected()	获得复选框是否被选中

下述代码演示单选按钮和复选框的使用。

【代码 3-20】 RadioCheckDemo.java

```
package com.qst.chapter03;

import java.awt.FlowLayout;
import java.awt.GridLayout;

import javax.swing.ButtonGroup;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;

public class RadioCheckDemo extends JFrame {
    private JPanel p1, p2;
    private JLabel lblSex, lblLike;
    private JRadioButton rbMale, rbFemale;
    private ButtonGroup bg;
```

```
    private JCheckBox ckbRead, ckbNet, ckbSwim, ckbTour;
    public RadioCheckDemo() {
        super("单选和复选");
        this.setLayout(new GridLayout(2, 1));
        p1 = new JPanel(new FlowLayout(FlowLayout.LEFT));
        p2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
        lblSex = new JLabel("性别:");
        lblLike = new JLabel("爱好:");
        // 创建单选按钮
        rbMale = new JRadioButton("男", true);
        rbFemale = new JRadioButton("女");
        // 创建按钮组
        bg = new ButtonGroup();
        // 将 rb1 和 rb2 两个单选按钮添加到按钮组中,这两个单选按钮只能选中其一
        bg.add(rbMale);
        bg.add(rbFemale);
        // 创建复选框
        ckbRead = new JCheckBox("阅读");
        ckbNet = new JCheckBox("上网");
        ckbSwim = new JCheckBox("游泳");
        ckbTour = new JCheckBox("旅游");

        // 性别相关的组件添加到 p1 子面板中
        p1.add(lblSex);
        p1.add(rbMale);
        p1.add(rbFemale);
        this.add(p1);

        // 爱好相关的组件添加到 p2 子面板中
        p2.add(lblLike);
        p2.add(ckbRead);
        p2.add(ckbNet);
        p2.add(ckbSwim);
        p2.add(ckbTour);
        this.add(p2);

        // 设定窗口大小
        this.setSize(300, 100);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new RadioCheckDemo();
    }
}
```

上述代码采用了网格和流布局相结合的混合布局，运行结果如图 3-29 所示。

图 3-29 单选按钮和复选框

3.5.9 计算器

下述代码实现一个计算器功能，用户操作按钮或键盘都能进行算术运算。

【代码 3-21】 Calculator.java

```
package com.qst.chapter03;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Calculator extends JFrame {
    // 声明一个文本栏控件,用于显示计算结果
    private JTextField txtResult;
    private JPanel p;
    // 定义一个字符串数组,将计算器中按钮的文字都放在该数组中
    private String name[] = { "7", "8", "9", "+", "4", "5", "6", "-", "1", "2",
            "3", "*", "0", ".", "=", "/" };
    // 声明一个按钮数组,该数组的长度以字符串数组的长度为准
    private JButton button[] = new JButton[name.length];
    // 定义一个存放计算结果的变量,初始为 0
    private double result = 0;
    // 存放最后一个操作符,初始为"="
    private String lastCommand = "=";
    // 标识是否是开始
    private boolean start = true;
    public Calculator() {
        super("计算器");
        // 实例化文本栏控件
        txtResult = new JTextField(20);
        // 设置文本框不是焦点状态
        txtResult.setFocusable(false);
```

```
        // 将文本栏控件放置在窗体框架的上方(北部)
        this.add(txtResult, BorderLayout.NORTH);
        // 实例化面板对象,同时设置此面板布局为 4 行 4 列的网格布局
        p = new JPanel(new GridLayout(4, 4));
        // 循环实例化按钮对象数组
        // 实例化按钮监听对象
        ButtonAction ba = new ButtonAction();
        // 实例化键盘监听对象
        KeyAction ka = new KeyAction();
        for (int i = 0; i < button.length; i++) {
            button[i] = new JButton(name[i]);
            // 注册监听
            button[i].addActionListener(ba);
            button[i].addKeyListener(ka);
            p.add(button[i]);
        }
        this.add(p, BorderLayout.CENTER);
        // 设定窗口大小
        this.setSize(200, 200);
        // 设定窗口左上角坐标
        this.setLocation(200, 100);
        // 设定窗口默认关闭方式为退出应用程序
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗口可视(显示)
        this.setVisible(true);
    }
    // 计算
    public void calculate(double x) {
        if (lastCommand.equals(" + "))
            result += x;
        else if (lastCommand.equals(" - "))
            result -= x;
        else if (lastCommand.equals(" * "))
            result * = x;
        else if (lastCommand.equals("/"))
            result / = x;
        else if (lastCommand.equals(" = "))
            result = x;
        // 将结果显示在文本栏
        txtResult.setText("" + result);
    }
    // 单击按钮监听
    private class ButtonAction implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            String input = e.getActionCommand();
            // 单击操作符号按钮
            if (input.equals(" + ") || input.equals(" - ") || input.equals(" * ")
                    || input.equals("/") || input.equals(" = ")) {
                if (start) {
                    if (input.equals(" - ")) {
```

```
                        txtResult.setText(input);
                        start = false;
                    } else
                        lastCommand = input;
                } else {
                    calculate(Double.parseDouble(txtResult.getText()));
                    lastCommand = input;
                    start = true;
                }
            } else {
                if (start) {
                    txtResult.setText("");
                    start = false;
                }
                txtResult.setText(txtResult.getText() + input);
            }
        }
    }
    // 键盘监听
    private class KeyAction extends KeyAdapter {
        public void keyTyped(KeyEvent e) {
            char key = e.getKeyChar();
            // 敲击的键盘是数字
            if (key == '0' || key == '1' || key == '2' || key == '3'
                    || key == '4' || key == '5' || key == '6' || key == '7'
                    || key == '8' || key == '9' || key == '9') {
                if (start) {
                    txtResult.setText("");
                    start = false;
                }
                txtResult.setText(txtResult.getText() + key);
            }
            // 敲击的键盘是操作符号
            else if (key == '+' || key == '-' || key == '*' || key == '/'
                    || key == '=') {
                if (start) {
                    if (key == '-') {
                        txtResult.setText(String.valueOf(key));
                        start = false;
                    } else
                        lastCommand = String.valueOf(key);
                } else {
                    calculate(Double.parseDouble(txtResult.getText()));
                    lastCommand = String.valueOf(key);
                    start = true;
                }
            }
        }
    }
    public static void main(String[] args) {
        new Calculator();
    }
}
```

上述代码不仅要对单击按钮的行为事件进行处理，还要对敲击键盘的键盘事件进行处理。因此，创建了两个监听类，分别监听单击按钮的行为事件和敲击键盘的事件。进行事件处理时“数字”和“操作符”需要分别处理，其中“-”有两种意思：“负”和“减”，当在算式开始时代表负号，否则是减法符号。

运行结果如图 3-30 所示，操作按钮或键盘检验运算器的计算效果。

图 3-30　计算器

3.6　贯穿任务实现

3.6.1　实现【任务 3-1】

下述内容实现 Q-DMS 贯穿项目中的【任务 3-1】创建用户数据库表、用户实体类和用户业务逻辑类，为后期实现用户的注册和登录功能打下基础。

1. 创建用户表

在 Oracle 数据库中创建用户信息表，具体的 SQL 代码如下所示。

【任务 3-1】　q_dms. sql

```
-- 创建用户表
create table USERDETAILS
(
  id           NUMBER primary key,
  username     VARCHAR2(50) not null,
  password     VARCHAR2(50),
  sex          NUMBER,
  hobby        VARCHAR2(500),
  address      VARCHAR2(50),
  degree       VARCHAR2(50)
);
-- 创建序列
create sequence SEQ_USER
start with 1001
increment by 1;
```

2. 编写用户实体类

编写用户实体类 User，以便封装用户数据，代码如下所示。

【任务 3-1】　User. java

```
package com.qst.dms.entity;

//用户实体
```

```
public class User {
    // 用户 id
    private int id;
    // 用户名
    private String username;
    // 密码
    private String password;
    // 性别
    private int sex;
    // 爱好
    private String hobby;
    // 地址
    private String address;
    // 学历
    private String degree;

    //...省略 getter/setter 方法
    public User() {
    }
    public User(String username, String password, int sex, String hobby,
            String address, String degree) {
        this.username = username;
        this.password = password;
        this.sex = sex;
        this.hobby = hobby;
        this.address = address;
        this.degree = degree;
    }
    public User(int id, String username, String password, int sex,
            String hobby, String address, String degree) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.sex = sex;
        this.hobby = hobby;
        this.address = address;
        this.degree = degree;
    }
}
```

3. 编写用户业务类

编写用户业务逻辑类 UserService,代码如下所示。

【任务 3-1】 UserService.java

```
package com.qst.dms.service;

import java.sql.ResultSet;
```

```
import com.qst.dms.db.DBUtil;
import com.qst.dms.entity.User;

public class UserService {
    // 根据用户名查询用户
    public User findUserByName(String userName) {
        DBUtil db = new DBUtil();
        User user = null;
        try {
            // 获取数据库链接
            db.getConnection();
            // 使用 PreparedStatement 发送 sql 语句
            String sql = "SELECT * FROM userdetails WHERE username = ?";
            // 设置参数
            Object[] param = new Object[] { userName };
            // 执行查询
            ResultSet rs = db.executeQuery(sql, param);
            if (rs.next()) {
                // 将结果集中的数据封装到对象中
                user = new User(rs.getInt(1), rs.getString(2), rs.getString(3),
                        rs.getInt(4), rs.getString(5), rs.getString(6),
                        rs.getString(7));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //释放数据库资源
            db.closeAll();
        }
        // 返回用户对象
        return user;
    }
    // 保存用户信息
    public boolean saveUser(User user) {
        // 定义一个布尔返回值,初始值为 false
        boolean r = false;
        DBUtil db = new DBUtil();
        try {
            // 获取数据库连接
            db.getConnection();
            // 使用 PreparedStatement 发送 sql 语句
            String sql = "INSERT INTO userdetails
                (id,username,password,sex,hobby,address,degree)
                VALUES (SEQ_USER.NEXTVAL,?,?,?,?,?,?)";
            // 设置参数
            Object[] param = new Object[] { user.getUsername(),
                    user.getPassword(), user.getSex(), user.getHobby(),
                    user.getAddress(), user.getDegree() };
            // 判断数据是否保存成功
            if (db.executeUpdate(sql, param) > 0) {
```

```
                // 保存成功,设置返回值为 true
                r = true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //释放数据库资源
            db.closeAll();
        }
        // 返回
        return r;
    }
}
```

在上述代码中实现了两个业务方法,其中 findUserByName()方法用于根据用户名查找指定的用户,而 saveUser()方法用于将用户的信息保存到数据库。

3.6.2 实现【任务 3-2】

下述内容实现 Q-DMS 贯穿项目中的【任务 3-2】创建用户注册窗口,并将用户注册信息保存到数据库。

编写用户注册窗口 RegistFrame,代码如下所示。

【任务 3-2】 RegistFrame.java

```
package com.qst.dms.ui;

import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JRadioButton;
import javax.swing.JTextArea;
import javax.swing.JTextField;

import com.qst.dms.entity.User;
import com.qst.dms.service.UserService;
//注册窗口
public class RegistFrame extends JFrame {
```

```
// 主面板
private JPanel p;
// 标签
private JLabel lblName, lblPwd, lblRePwd, lblSex, lblHobby, lblAdress,
        lblDegree;
// 用户名,文本框
private JTextField txtName;
// 密码和确认密码,密码框
private JPasswordField txtPwd, txtRePwd;
// 性别,单选按钮
private JRadioButton rbMale, rbFemale;
// 爱好,多选框
private JCheckBox ckbRead, ckbNet, ckbSwim, ckbTour;
// 地址,文本域
private JTextArea txtAdress;
// 学历,组合框
private JComboBox<String> cmbDegree;
// 确认和取消,按钮
private JButton btnOk, btnCancle;
// 注册的用户
private static User user;
// 用户业务类
private UserService userService;
// 构造方法
public RegistFrame() {
    super("用户注册");
    // 实例化用户业务类对象
    userService = new UserService();
    // 设置窗体的 icon
    ImageIcon icon = new ImageIcon("images\\dms.png");
    this.setIconImage(icon.getImage());
    // 设置面板布局,网格布局
    p = new JPanel(new GridLayout(8, 1));
    // 实例化组件
    lblName = new JLabel("用  户  名: ");
    lblPwd = new JLabel("密    码: ");
    lblRePwd = new JLabel("确认密码: ");
    lblSex = new JLabel("性    别: ");
    lblHobby = new JLabel("爱    好: ");
    lblAdress = new JLabel("地    址: ");
    lblDegree = new JLabel("学    历: ");
    txtName = new JTextField(16);
    txtPwd = new JPasswordField(16);
    txtRePwd = new JPasswordField(16);
    rbMale = new JRadioButton("男");
    rbFemale = new JRadioButton("女");
    // 性别的单选逻辑
    ButtonGroup bg = new ButtonGroup();
    bg.add(rbMale);
    bg.add(rbFemale);
```

```
ckbRead = new JCheckBox("阅读");
ckbNet = new JCheckBox("上网");
ckbSwim = new JCheckBox("游泳");
ckbTour = new JCheckBox("旅游");
txtAdress = new JTextArea(3, 20);
// 组合框显示的学历数组
String str[] = { "小学", "初中", "高中", "本科", "硕士", "博士" };
cmbDegree = new JComboBox<String>(str);
// 设置组合框可编辑
cmbDegree.setEditable(true);
btnOk = new JButton("确定");
// 注册监听器,监听确定按钮
btnOk.addActionListener(new RegisterListener());
btnCancle = new JButton("重置");
// 注册监听器,监听重置按钮
btnCancle.addActionListener(new ResetListener());
// 将组件分组放入面板,然后将小面板放入主面板
JPanel p1 = new JPanel(new FlowLayout(FlowLayout.LEFT));
p1.add(lblName);
p1.add(txtName);
p.add(p1);
JPanel p2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
p2.add(lblPwd);
p2.add(txtPwd);
p.add(p2);
JPanel p3 = new JPanel(new FlowLayout(FlowLayout.LEFT));
p3.add(lblRePwd);
p3.add(txtRePwd);
p.add(p3);
JPanel p4 = new JPanel(new FlowLayout(FlowLayout.LEFT));
p4.add(lblSex);
p4.add(rbMale);
p4.add(rbFemale);
p.add(p4);
JPanel p5 = new JPanel(new FlowLayout(FlowLayout.LEFT));
p5.add(lblHobby);
p5.add(ckbRead);
p5.add(ckbNet);
p5.add(ckbSwim);
p5.add(ckbTour);
p.add(p5);
JPanel p6 = new JPanel(new FlowLayout(FlowLayout.LEFT));
p6.add(lblAdress);
p6.add(txtAdress);
p.add(p6);
JPanel p7 = new JPanel(new FlowLayout(FlowLayout.LEFT));
p7.add(lblDegree);
p7.add(cmbDegree);
p.add(p7);
JPanel p8 = new JPanel(new FlowLayout(FlowLayout.CENTER));
```

```
        p8.add(btnOk);
        p8.add(btnCancle);
        p.add(p8);
        // 主面板放入窗体中
        this.add(p);
        // 设置窗体大小和位置居中
        this.setSize(310, 350);
        this.setLocationRelativeTo(null);
        // 设置窗体不可改变大小
        this.setResizable(false);
        // 设置窗体初始可见
        this.setVisible(true);
    }
    // 监听类,负责处理确认按钮的业务逻辑
    private class RegisterListener implements ActionListener {
        // 重写 actionPerFormed()方法,事件处理方法
        public void actionPerformed(ActionEvent e) {
            // 获取用户输入的数据
            String userName = txtName.getText().trim();
            String password = new String(txtPwd.getPassword());
            String rePassword = new String(txtRePwd.getPassword());
            // 将性别"男""女"对应转化为"1""0"
            int sex = Integer.parseInt(rbFemale.isSelected() ? "0" : "1");
            String hobby = (ckbRead.isSelected() ? "阅读" : "")
                    + (ckbNet.isSelected() ? "上网" : "")
                    + (ckbSwim.isSelected() ? "游泳" : "")
                    + (ckbTour.isSelected() ? "旅游" : "");
            String address = txtAdress.getText().trim();
            String degree = cmbDegree.getSelectedItem().toString().trim();
            // 判断两次输入密码是否一致
            if (password.equals(rePassword)) {
                // 将数据封装到对象中
                user = new User(userName, password, sex, hobby, address, degree);
                // 保存数据
                if (userService.saveUser(user)) {
                    // 输出提示信息
                    System.out.println("注册成功!");
                } else {
                    // 输出提示信息
                    System.out.println("注册失败!");
                }
            } else {
                // 输出提示信息
                System.out.println("两次输入的密码不一致!");
            }
        }
    }
    // 监听类,负责处理重置按钮
    public class ResetListener implements ActionListener {
        // 重写 actionPerFormed()方法,重置组件内容事件处理方法
```

```
        public void actionPerformed(ActionEvent e) {
            // 清空姓名、密码、确认密码内容
            txtName.setText("");
            txtPwd.setText("");
            txtRePwd.setText("");
            // 重置单选按钮为未选择
            rbMale.setSelected(false);
            rbFemale.setSelected(false);
            // 重置所有的复选按钮为未选择
            ckbRead.setSelected(false);
            ckbNet.setSelected(false);
            ckbSwim.setSelected(false);
            ckbTour.setSelected(false);
            // 清空地址栏
            txtAdress.setText("");
            // 重置组合框为未选择状态
            cmbDegree.setSelectedIndex(0);
        }
    }
    public static void main(String[] args) {
        new RegistFrame();
    }
```

在上述代码中，声明了用户对象 user 和用户业务类 userService，并在构造方法中先实例化 userService 业务对象。当用户单击“注册”按钮时，先获取界面中的信息，然后封装到 user 对象中，再调用 userService.saveUser()方法将 user 对象中的数据保存到数据库中。

运行结果如图 3-31 所示。

单击“注册”按钮，注册多个用户信息后，查看数据库中保存的数据，如图 3-32 所示。

图 3-31　注册界面

图 3-32　用户表

3.6.3　实现【任务 3-3】

下述内容实现 Q-DMS 贯穿项目中的【任务 3-3】创建用户登录窗口，登录成功则进入系统主界面。

1. 编写主窗口

主窗口 MainFrame 的代码如下所示。

【任务 3-3】 MainFrame.java

```
package com.qst.dms.ui;

import javax.swing.ImageIcon;
import javax.swing.JFrame;

//主窗口
public class MainFrame extends JFrame {
    // 构造方法
    public MainFrame() {
        // 调用父类的构造方法
        super("Q-DMS 系统客户端");
        // 设置窗体的 icon
        ImageIcon icon = new ImageIcon("images\\dms.png");
        this.setIconImage(icon.getImage());
        // 设置窗体初始可见
        this.setVisible(true);
        // 设置窗体大小
        this.setSize(600, 400);
        // 设置窗口在屏幕中央
        this.setLocationRelativeTo(null);
        // 设置默认的关闭按钮操作为退出程序
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

上述代码主要为了用户登录成功后可以进入主界面，仅为了显示而用，更具体的内容会在第 4 章的任务中进行迭代。

2. 编写用户登录窗口

编写用户登录窗口 LoginFrame，代码如下所示。

【任务 3-3】 LoginFrame.java

```
package com.qst.dms.ui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
```

```
import javax.swing.JTextField;

import com.qst.dms.entity.User;
import com.qst.dms.service.UserService;

//登录窗口
public class LoginFrame extends JFrame {
    // 主面板
    private JPanel p;
    // 标签
    private JLabel lblName, lblPwd;
    // 用户名,文本框
    private JTextField txtName;
    // 密码,密码框
    private JPasswordField txtPwd;
    // 确认、取消和注册,按钮
    private JButton btnOk, btnCancle, btnRegist;
    // 登录用户
    private static User user;
    // 用户业务类
    private UserService userService;
    // 构造方法
    public LoginFrame() {
        super("登录");
        // 实例化用户业务类对象
        userService = new UserService();
        // 设置窗体的 icon
        ImageIcon icon = new ImageIcon("images\\dms.png");
        this.setIconImage(icon.getImage());
        // 实例化组件
        p = new JPanel();
        // 使用 null 布局
        p.setLayout(null);
        lblName = new JLabel("用户名: ");
        lblPwd = new JLabel("密 码: ");
        txtName = new JTextField(20);
        txtPwd = new JPasswordField(20);
        txtPwd.setEchoChar('*');

        btnOk = new JButton("登录");
        btnOk.addActionListener(new LoginListener());

        btnCancle = new JButton("重置");
        btnCancle.addActionListener(new ResetListener());

        btnRegist = new JButton("注册");
        btnRegist.addActionListener(new RegistListener());

        lblName.setBounds(30, 30, 60, 25);
        lblPwd.setBounds(30, 60, 60, 25);
```

```
        txtName.setBounds(95, 30, 120, 25);
        txtPwd.setBounds(95, 60, 120, 25);
        btnOk.setBounds(30, 90, 60, 25);
        btnCancle.setBounds(95, 90, 60, 25);
        btnRegist.setBounds(160, 90, 60, 25);

        p.add(lblName);
        p.add(txtName);
        p.add(lblPwd);
        p.add(txtPwd);
        p.add(btnOk);
        p.add(btnCancle);
        p.add(btnRegist);

        // 主面板放入窗体中
        this.add(p);
        // 设置窗体大小和位置
        this.setSize(250, 170);
        // 设置窗口在屏幕中央
        this.setLocationRelativeTo(null);
        // 设置窗体不能改变大小
        this.setResizable(false);
        // 设置窗体的默认关闭按钮
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 设置窗体初始可见
        this.setVisible(true);
    }
    // 监听类,负责处理登录按钮
    public class LoginListener implements ActionListener {
        // 重写 actionPerFormed()方法,事件处理逻辑
        public void actionPerformed(ActionEvent e) {
            // 根据用户名查询用户
            user = userService.findUserByName(txtName.getText().trim());
            // 判断用户是否存在
            if (user != null) {
                // 判断输入的密码是否正确
                if (user.getPassword().equals(new String(txtPwd.getPassword()))) {
                    // 登录成功,隐藏登录窗口
                    LoginFrame.this.setVisible(false);
                    // 显示主窗口
                    new MainFrame();
                } else {
                    // 输出提示信息
                    System.out.println("密码错误!请重新输入!");
                    // 清空密码框
                    txtPwd.setText("");
                }
            } else {
                // 输出提示信息
                System.out.println("该用户不存在,请先注册!");
```

```
                }
            }
        }
        // 监听类,负责处理重置按钮
        public class ResetListener implements ActionListener {
            // 重写 actionPerFormed()方法,事件处理方法
            public void actionPerformed(ActionEvent e) {
                // 清空文本框
                txtName.setText("");
                txtPwd.setText("");
            }
        }
        // 监听类,负责处理注册按钮
        public class RegistListener implements ActionListener {
            // 重写 actionPerFormed()方法,事件处理方法
            public void actionPerformed(ActionEvent e) {
                // 创建注册窗口
                new RegistFrame();
            }
        }
        // 主程序,整个应用程序的入口
        public static void main(String[] args) {
            new LoginFrame();
        }
    }
```

在上述代码中,声明了用户对象 user 和用户业务类 userService,并在构造方法中先实例化 userService 业务对象。当用户单击“登录”按钮时,调用 userService.findUserByName()方法根据用户名查询数据库中的 user 对象,再判断用户输入的密码是否正确,如果正确,则进入主窗口中。当用户单击“注册”按钮时,则显示注册窗口。如此,登录窗口 LoginForm 可以作为整个项目的入口。

运行结果如图 3-33 所示。

图 3-33 登录窗口

登录成功后进入的主窗口如图 3-34 所示。

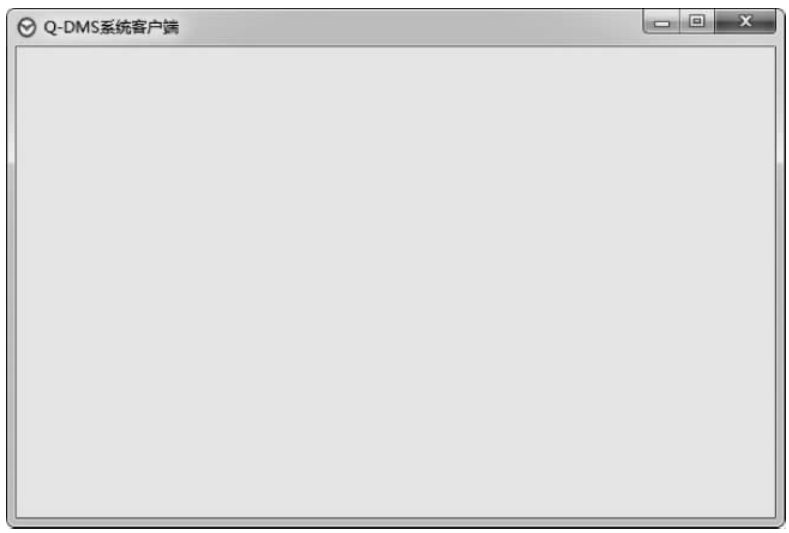


图 3-34　主窗口

本章总结

小结

- AWT(Abstract Windows Tookit,抽象窗口工具集)界面功能有限,在不同的平台上外观效果不同且不美观。
- Swing 用户界面组件丰富,在不同平台上外观效果一致。
- 大部分 Swing 组件都是 JComponent 抽象类的直接或间接子类。
- JFrame(窗口框架)是可以独立存在的顶级窗口容器,能够包含其他子容器,其默认布局为 BorderLayout。
- JPanel(面板)是一种中间容器,其默认布局为 FlowLayout。
- AWT 提供了 FlowLayout、BorderLayout、GridLayout、GridBagLayout 和 CardLayout 五个常用的布局管理器。
- Swing 提供了 BoxLayout 布局管理器。
- NULL 空布局是指容器不采用任何布局,而是通过每个组件的绝对定位进行布局。
- 事件(Event)由用户操作产生,而不是通过 new 运算符创建。
- 事件源(Event Source)是事件发生的场所。
- 事件监听器(Event Listener)负责监听事件源所产生的事件,并对事件做出响应处理。
- 事件处理机制是一种委派式的事件处理方式:委托、通知、处理。
- AWT 的事件类都是 AWTEvent 类的子类。
- 常见的 AWT 事件类有 MouseEven、ActionEvent、ItemEvent 和 WindowEvent 等。
- AWT 提供了大量的监听接口,用于实现不同类型的事件监听器。
- ActionListener、KeyListener、MouseListener 和 MouseMotionListener 都是常用的

监听接口。

- 适配器用于简化程序编码。
- Icon、JButton、JLabel、JTextField、JTextArea、JPasswordField、JComboBox、JList、JRadioButton 和 JCheckBox 都是常用的 GUI 基本组件。

Q&A

1. 问题：简述 Java 事件处理机制。

回答：事件处理机制是一种委派式的事件处理方式，实现步骤如下：

(1) 委托——组件(事件源)将事件处理委托给特定的事件处理对象(事件监听器)；

(2) 通知——当组件(事件源)触发指定的事件时，就通知所委托的事件处理对象(事件监听器)；

(3) 处理——事件处理对象(事件监听器)调用相应的事件处理方法来处理该事件。

2. 问题：简述 Java 程序中实现事件处理的具体步骤。

回答：在 Java 程序中，实现事件处理需要以下三个步骤：

(1) 创建监听类，实现监听接口并重写监听接口中的事件处理方法；

(2) 创建监听对象，即实例化上一步中所创建的监听类的对象；

(3) 注册监听对象，调用组件的 addXXXListener()方法将监听对象注册到相应组件上，以便监听对事件源所触发的事件。

3. 问题：简述使用 NULL 空布局的步骤。

回答：使用 NULL 空布局需要以下三步：

(1) 将容器中的布局管理器设置为 null(空)，即容器不采用任何布局；

(2) 调用 setBounds()设置组件的绝对位置坐标及大小，或使用 setLocation()、setSize()分别设置组件的坐标和大小；

(3) 将组件添加到容器中。

章节练习

习题

1. 下列关于 AWT 和 Swing 说法的中正确的是_____。

A. AWT 是 Sun 公司提供的一套基本的抽象窗口工具集

B. AWT 运行在不同的平台上，会呈现不同的外观效果

C. Swing 能够保证不同平台上用户一致的感观效果

D. Swing 中的 JComponent 类是 AWT 中 java.awt.Container 类的子类

2. 下列_____不是 Swing 中的顶层容器。

A. JFrame　　B. JApplet　　C. JDialog　　D. JPanel

E. JWindow

3. 布局管理器用来管理组件在容器中的布局格式，下列关于布局的说法中错误的

是_____。

A. FlowLayout 流布局是将容器中的组件按照从左到右的顺序，流动地排列和分布

B. CardLayout 网格布局就像表格一样，将容器按照行和列分割成单元格，每个单元格所占的区域大小都一样

C. BorderLayout 边界布局允许将组件有选择地放置到容器的中部、北部、南部、东部、西部这五个区域

D. 空布局是指容器不采用任何布局，而是通过每个组件的绝对定位进行布局

4. AWT 事件分为低级事件和高级事件，其中低级事件不包括_____。

A. ItemEvent　　B. WindowEvent

C. MouseEvent　　D. KeyEvent

5. AWT 事件分为低级事件和高级事件，其中高级事件不包括_____。

A. ActionEvent　　B. AdjustmentEvent

C. MouseEvent　　D. TextEvent

6. 下列关于 Icon 接口的说法中错误的是_____。

A. Icon 是一个图标接口，用于加载图片

B. ImageIcon 类是 Icon 接口的一个实现类

C. Icon 类型的对象可以作为 JButton 的构造方法的参数，在创建 JButton 对象的同时指定按钮对应的图标

D. Icon 类型的对象可以作为 JRadioButton 的构造方法的参数，在创建 JRadioButton 对象的同时指定按钮对应的图标

7. 下列组件中，不允许多选的是_____。

A. JComboBox　　B. JRadioButton

C. JCheckBox　　D. JList

上机

训练目标：GUI 界面编程。

培养能力	使用 Swing 组件创建 GUI 界面		
掌握程度	★★★★★	难度	难
代码行数	350	实施方式	编码强化
结束条件	独立编写，不出错。		

参考训练内容

(1) 创建一个学生信息录入界面，学生有学号、姓名、年龄、班级和成绩信息。

(2) 实现事件处理，单击“确定”按钮将学生信息封装到对象中并保存到数据库；单击“重置”按钮清空界面中用户输入的信息。