第17章

Java 事件处理

JavaGUI 的事件，能够让我们真正完善程序的功能。本章将首先讲解事件的基本原理，然后讲解事件的开发流程，最后讲解几种最常见的事件的处理：ActionEvent、FocusEvent、KeyEvent、MouseEvent、WindowEvent，最后讲解用 Adapter 简化事件的开发。

本章术语

Event _____

Listener _____

ActionEvent _____

FocusEvent _____

KeyEvent _____

MouseEvent _____

WindowEvent _____

Adapter _____

17.1 认识事件处理

Note

17.1.1 什么是事件

在前面的程序中，我们可以在窗体上添加若干控件。比如，添加按钮，以下面的例子为例：

EventTest1.java

```
package event;
import javax.swing.*;
public class EventTest1 extends JFrame{
    private JButton btHello = new JButton("Hello");
    public EventTest1(){
        this.add(btHello);
        this.setSize(30,50);
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new EventTest1();
    }
}
```

运行，效果为：

图 17-1

界面上有一个按钮。但是，当点击按钮时，却没有任何反应。显然，真正丰富多彩的程序中，我们点击按钮，至少需要能做点事情。比如，点击按钮，在控制台上打印一个字符串：Hello。该功能如何实现呢？这就需要使用本章讲解的事件处理。

什么是事件？简单讲，事件是指用户为了交互而产生的键盘和鼠标动作。比如，点击按钮，就可以认为发出了一个“按钮点击事件”。

注意

1. 以上事件的定义，不太严谨，只是最直观的说法。实际上，事件不一定在用户交互时产生。比如，程序运行出了异常，也可以认为是一个事件。

2. 事件是有种类的。比如，按钮点击，是一种事件；鼠标在界面上移动，也是一种事件；等等。要处理某事件，首先必须搞清楚事件的种类。后面的篇幅我们有详细讲解。

17.1.2 事件处理代码的编写

Note

在了解事件处理之前，我们先举生活中的一个例子：

在生活中，也会出现很多出现“事件”的场合。比如，上课铃响了，小王听到了铃声，走进教室。

在这个事件中，上课铃响，相当于发出了一个事件，类似于上节的“点击按钮”，小王走进教室，相当于处理这个事件，类似于上节的“打印 Hello”。

实际上，这个看似简单的日常生活例子，其顺利执行的条件并不简单，至少需要以下条件：

1. 铃声必须由响铃的地方传到小王耳朵里，因此，铃声必须进行封装。
2. 小王必须长着耳朵，否则他听不到，铃声再响也是徒劳。
3. 小王必须执行“走进教室”这个动作，否则相当于事件没有处理。
4. 必须规定，上课铃响，让小王进教室。如果没有这个规定，小王如何知道要走进教室？

我们来进行类比，实际上，上面的几个条件可以解释如下：

1. 事件必须用一个对象封装。
2. 事件的处理者必须具有监听事件的能力。
3. 事件的处理者必须编写事件处理函数。
4. 必须将事件的发出者和事件的处理者对象绑定起来。

这四个步骤，就是我们编写事件代码的依据。这里，我们来实现“点击按钮，打印 Hello”。

1. 事件必须用一个对象封装。

点击按钮，系统自动将发出的事件封装在 java.awt.event.ActionEvent 对象内。

问答

问：如何知道某个事件封装在什么样的对象内？

答：由于事件是有种类的，因此，不同的事件封装在不同的对象内，在后面的章节中，我们进行了详细的总结。此处只要知道点击按钮，发出的事件封装在 java.awt.event.ActionEvent 对象内即可。

2. 事件的处理者必须具有监听事件的能力。

在 java 中，ActionEvent 是由 java.awt.event.ActionListener 监听的。

因此，此步骤中，我们需要编写一个事件处理类，来实现 ActionListener 接口。

```
class ButtonClickOpe implements ActionListener{
    //处理事件
}
```

3. 事件的处理者必须编写事件处理函数。

在 java 中，实现一个接口，必须将接口中的函数重写一遍，该函数就是事件处理函数。查看文档，可以找到 ActionListener 中定义的函数：

Note

void actionPerformed(ActionEvent e)

对其进行重写并编写事件处理代码：

```
class ButtonClickOpe implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        System.out.println("Hello");
    }
}
```

注意

actionPerformed 函数中，参数 e 就封装了发出的事件。通过参数 e 的 getSource() 方法，可以知道事件是由谁发出的。后面我们将会用到。

4. 必须将事件的发出者和事件的处理者对象绑定起来。

该步骤中，必须规定按钮点击，发出的事件由 ButtonClickOpe 对象处理。方法是调用按钮的如下函数：

public void addActionListener(ActionListener l)

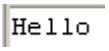
因此，整个代码就可以写成：

EventTest2.java

```
package event;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
public class EventTest2 extends JFrame{
    private JButton btHello = new JButton("按钮");
    public EventTest2(){
        this.add(btHello);
        //绑定
        btHello.addActionListener(new ButtonClickOpe());
        this.setSize(30,50);
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new EventTest2();
    }
}
class ButtonClickOpe implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        System.out.println("Hello");
```

```
    }
}
```

运行，点击按钮，控制台上打印：



说明事件成功处理。

从本例可以看出，事件处理，关键是要弄清要处理什么样的事件，其他按照上面的流程编程即可。

Note

阶段性作业

已知：在 JTextField 中输入回车，发出的也是 ActionEvent，请编写：

在 JFrame 上放置一个文本框，文本框中输入一个数字，回车，在控制台上打印该文本框中数值的平方。

17.1.3 另外几种编程风格

前一章的例子中，我们需要编写两个类，实际上，有很多方法可以进行简化。比如，我们可以使用匿名处理对象的方法实现前一节的例子：

EventTest3.java

```
package event;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
public class EventTest3 extends JFrame{
    private JButton btHello = new JButton("按钮");
    public EventTest3(){
        this.add(btHello);
        //绑定
        btHello.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                System.out.println("Hello");
            }
        });
        this.setSize(30,50);
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new EventTest3();
```

```
        }
    }
```

运行，点击按钮，打印“Hello”。不过，这种方法使用不多。

Note

由于一个 Java 类可以实现多个接口，我们通常用界面类直接实现接口的方法进行简化。本例中实现两个按钮，点击登录按钮，打印“登录”，点击退出按钮，程序退出。

EventTest4.java

```
package event;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
public class EventTest4 extends JFrame implements ActionListener{
    private JButton btLogin = new JButton("登录");
    private JButton btExit = new JButton("退出");
    public EventTest4(){
        this.setLayout(new FlowLayout());
        this.add(btLogin);
        this.add(btExit);
        //绑定
        btLogin.addActionListener(this);
        btExit.addActionListener(this);
        this.setSize(100,100);
        this.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==btLogin){
            System.out.println("登录");
        }else{
            System.exit(0);
        }
    }
    public static void main(String[] args) {
        new EventTest4();
    }
}
```

运行，效果如下：

图 17-2

点击登录按钮，控制台打印：

图 17-3

点击退出按钮，程序退出。

注意

此处使用 e.getSource()判断事件是由谁发出的。

17.2 处理 ActionEvent

17.2.1 什么情况发出 ActionEvent

在 java.awt.event 包中，ActionEvent 是最常用的一种事件。一般情况下，ActionEvent 适合于对某些控件的单击(也有特殊情况)。常见发出 ActionEvent 的场合如下：

1. JButton、JComboBox、JMenu、JMenuItem、JCheckBox、JRadioButton等控件的的单击。

2. javax.swing.Timer 发出的事件。

3. 在 JTextField 等控件上按下回车、JButton 等控件上按下空格(相当于点击效果)等。

等等。ActionEvent 用 ActionListener 监听。其编程方法，采用上节流程即可。

17.2.2 使用 ActionEvent 解决实际问题

以下案例中，界面上包含一个下拉列表框，选择界面颜色，当选择之后，能够将界面背景自动变成相应颜色。代码如下：

ActionEventTest1.java

```
package actionevent;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
```

Note

```
import java.awt.event.ActionListener;
import javax.swing.JComboBox;
import javax.swing.JFrame;
public class ActionEventTest1 extends JFrame implements ActionListener{
    private JComboBox cbColor = new JComboBox();
    public ActionEventTest1(){
        this.add(cbColor,BorderLayout.NORTH);
        cbColor.addItem("红");
        cbColor.addItem("绿");
        cbColor.addItem("蓝");
        cbColor.addActionListener(this);
        this.setSize(30,100);
        this.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        Object color = cbColor.getSelectedItem();
        if(color.equals("红")){
            this.getContentPane().setBackground(Color.red);
        }else if(color.equals("绿")){
            this.getContentPane().setBackground(Color.green);
        }else{
            this.getContentPane().setBackground(Color.blue);
        }
    }
    public static void main(String[] args) {
        new ActionEventTest1();
    }
}
```

运行，效果为：

图 17-4

选择某种颜色，可以将界面背景变成相应颜色：

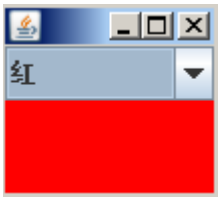


图 17-5

Note

注意

此代码中，我们改变的是 JFrame 的颜色，以变为红色为例，使用的代码段是：this.getContentPane().setBackground(Color.red);，改变颜色，必须得到 JFrame 上的 ContentPane，而不能直接使用 this.setBackground(Color.red);。

阶段性作业

1. 将上例中的下拉列表框改为单选按钮，选择颜色，能够将界面背景变成相应颜色。

2. 编写一个 JFrame 界面，上面含有一个菜单(JMenu)：打开文件。点击该菜单，出现文件选择框(JFileChooser)，选择一个文本文件，能够将文件内容显示在界面上的多行文本框(JTextArea)内。

3. 在 Swing 中，提供了一个定时器类 javax.swing.Timer，可以每隔一段时间执行一段代码。javax.swing.Timer 的构造函数为：

public Timer(int delay, ActionListener listener)

表示每隔一段时间(毫秒)，触发 ActionListener 内的处理代码。在实例化 Timer 对象之后，可以用 start()函数让其启动，可以用 stop()函数让其停止。

用 Timer 完成：界面上有一个按钮，从左边飞到右边。

4. 前面的章节中学习过，java.awt.SystemTray 可以向任务栏上添加一个托盘图标，托盘图标用 java.awt.TrayIcon 封装，但是将 TrayIcon 添加到任务栏上后，点击托盘图标，却没有任何反应。查询文档，实现：点击托盘图标，能够显示一个 JFrame 界面。

17.3 处理 FocusEvent

17.3.1 什么情况发出 FocusEvent

在 java.awt.event 包中，FocusEvent 也经常使用。一般情况下，FocusEvent 适合于对某些控件 Component 获得或失去输入焦点时，需要处理的场合。FocusEvent 用 java.awt.event.FocusListener 接口监听。该接口中有如下函数：

1. void focusGained(FocusEvent e)：组件获得焦点时调用。

2. void focusLost(FocusEvent e)：组件失去焦点时调用。

17.3.2 使用 FocusEvent 解决实际问题

以下案例中，界面上有一个文本框，要求，该文本框失去焦点时，内部显示：请您输入账号；当得到焦点时，该提示消失。代码如下：

Note

FocusEventTest1.java

```
package focusevent;
import java.awt.FlowLayout;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class FocusEventTest1 extends JFrame implements FocusListener{
    private JButton btOK = new JButton("确定");
    private JTextField tfAcc = new JTextField("请您输入账号",10);
    public FocusEventTest1(){
        this.setLayout(new FlowLayout());
        this.add(btOK);
        this.add(tfAcc);
        tfAcc.addFocusListener(this);//绑定
        this.setSize(200,80);
        this.setVisible(true);
    }
    public void focusGained(FocusEvent arg0) {
        tfAcc.setText("");
    }
    public void focusLost(FocusEvent arg0) {
        tfAcc.setText("请您输入账号");
    }
    public static void main(String[] args) {
        new FocusEventTest1();
    }
}
```

运行，效果为：

图 17-6

鼠标移动到文本框中，效果如下：

图 17-7

Note

阶段性作业

在界面上放 2 个按钮: 登录和退出，焦点到达某个按钮上，该按钮背景变为黄色，文字变为红色。如果失去焦点，就显示为一个普通按钮。

17.4 处理 KeyEvent

17.4.1 什么情况发出 KeyEvent

在 java.awt.event 包中，KeyEvent 也经常使用。一般情况下，KeyEvent 适合于在某个控件上进行键盘操作时，需要处理事件的场合。KeyEvent 用 java.awt.event.KeyListener 接口监听。该接口中有如下函数：

1. void keyTyped(KeyEvent e)：键入某个键时调用此方法。
2. void keyPressed(KeyEvent e)：按下某个键时调用此方法。
3. void keyReleased(KeyEvent e)：释放某个键时调用此方法。

17.4.2 使用 KeyEvent 解决实际问题

下面用一个程序进行测试，在一个 JFrame 上敲击按下键盘，释放，打印按键的内容。代码如下：

KeyEventTest1.java

```
package keyevent;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
public class KeyEventTest1 extends JFrame implements KeyListener{
    public KeyEventTest1(){
        this.addKeyListener(this);
        this.setSize(200,80);
        this.setVisible(true);
    }
```

Note

```
    public void keyPressed(KeyEvent e) {
        System.out.println(e.getKeyChar() + "按下");
    }
    public void keyReleased(KeyEvent e) {
        System.out.println(e.getKeyChar() + "释放");
    }
    public void keyTyped(KeyEvent e) {
        System.out.println(e.getKeyChar() + "敲击");
    }
    public static void main(String[] args) {
        new KeyEventTest1();
    }
}
```

运行，效果为：

图 17-8

如果按下键盘上的 a 键释放，控制台打印：

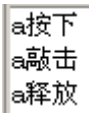


图 17-9

注意

1. 键盘事件，一定要在发出事件的控件已经获取焦点的情况下才能使用。比如，如果我们在 JFrame 上增加一个按钮，此时按钮获取了焦点，JFrame 的键盘事件就不会触发了，此时除非给按钮增加键盘事件。

2. KeyEvent 类中，封装了键的信息。主要函数有如下几个:

(1) public char getKeyChar():获取键的字符。

(2) public int getKeyCode(): 获取键对应的代码。代码可在文档 java.awt.event.KeyEvent 中查找，用静态变量表示，比如，左键对应的是: KeyEvent.VK_LEFT，左括弧对应的是: KeyEvent VK_LEFT_PARENTHESIS; 等等。

3. 键盘事件在游戏开发时经常用到，我们将在后面的篇幅中讲解。

阶段性作业

使用键盘事件完成: 界面上有一个含有卡通图标的 JLabel，可以通过键盘上的上下左右键控制其移动。

17.5 处理 MouseEvent

17.5.1 什么情况发出 MouseEvent

Note

在 java.awt.event 包中，MouseEvent 也经常使用。一般情况下，MouseEvent 在以下情况下发生：

1. 鼠标事件。

鼠标事件包括按下鼠标按键、释放鼠标按键、单击鼠标按键（按下并释放）、鼠标光标进入组件几何形状的未遮掩部分、鼠标光标离开组件几何形状的未遮掩部分。此时，MouseEvent 用 java.awt.event.MouseListener 接口监听。该接口中有如下函数：

(1) void mouseClicked(MouseEvent e)：鼠标按键在组件上单击（按下并释放）时调用。

(2) void mousePressed(MouseEvent e)：鼠标按键在组件上按下时调用。

(3) void mouseReleased(MouseEvent e)：鼠标按钮在组件上释放时调用。

(4) void mouseEntered(MouseEvent e)：鼠标进入到组件上时调用。

(5) void mouseExited(MouseEvent e)：鼠标离开组件时调用。

2. 鼠标移动事件

鼠标移动事件，包括移动鼠标和拖动鼠标。此时，MouseEvent 用 java.awt.event.MouseMotionListener 接口监听。该接口中有如下函数：

(1) void mouseDragged(MouseEvent e)：鼠标拖动时调用。

(2) void mouseMoved(MouseEvent e)：鼠标移动时调用。

17.5.2 使用 MouseEvent 解决实际问题

下面用一个程序进行鼠标事件测试，鼠标在 JFrame 上按下，将该处坐标设置为界面标题。代码如下：

MouseEventTest1.java

```
package mouseevent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.JFrame;
public class MouseEventTest1 extends JFrame implements MouseListener{
    public MouseEventTest1(){
        this.addMouseListener(this);
        this.setSize(300,100);
        this.setVisible(true);
    }
```

Note

```
    public void mouseClicked(MouseEvent e) {
        this.setTitle("鼠标点击：("+e.getX() + "," + e.getY() + ")");
    }
    public void mouseEntered(MouseEvent arg0) {}
    public void mouseExited(MouseEvent arg0) {}
    public void mousePressed(MouseEvent arg0) {}
    public void mouseReleased(MouseEvent arg0) {}
    public static void main(String[] args) {
        new MouseEventTest1();
    }
}
```

运行，效果为：

图 17-10

点击，界面标题变为：

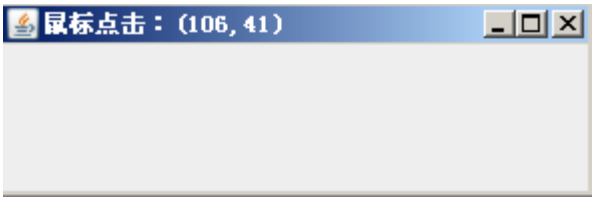


图 17-11

注意

1. 在本例中，MouseListener 接口中有 5 个函数，我们只用到一个：mouseClicked，其它函数是否可以不写呢？答案是不行，因为实现一个接口，必须将接口中的函数重写一遍。不用，也得写。不过该问题也可以通过其他手段得到解决，后面将会讲解。

2. MouseEvent 类中，封装了鼠标事件的信息。主要函数有如下几个：

(1) public int getClickCount()：返回鼠标单击次数。

(2) public int getX()和 public int getY()：返回鼠标光标在界面中的水平和垂直坐标。

其他的内容，大家可以参考文档。

3. 鼠标事件在画图软件开发时经常用到，我们将在后面的篇幅中讲解。

下面用一个程序进行测试鼠标移动事件，鼠标在 JFrame 上移动，当前坐标在界面上不断显示。代码如下：

MouseEventTest2.java

Note

```
package mouseevent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import javax.swing.JFrame;
public class MouseEventTest2 extends JFrame implements MouseMotionListener{
    public MouseEventTest2(){
        this.addMouseMotionListener(this);
        this.setSize(300,100);
        this.setVisible(true);
    }
    public void mouseDragged(MouseEvent arg0) {}
    public void mouseMoved(MouseEvent e) {
        this.setTitle("鼠标位置：("+e.getX() + "," + e.getY() + ")");
    }
    public static void main(String[] args) {
        new MouseEventTest2();
    }
}
```

运行，效果为：

图 17-12

鼠标移动，界面标题不断变化：

图 17-13

阶段性作业

使用鼠标事件完成：在界面空白处某个位置点击鼠标，能够在该位置放置一个含有卡通图标的 JLabel，如果在该 JLabel 内拖动鼠标，则可以将该 JLabel 拖到另一个位置释放。

17.6 处理 WindowEvent

Note

17.6.1 什么情况发出 WindowEvent

在 java.awt.event 包中，WindowEvent 也经常使用。一般情况下，WindowEvent 适合窗口状态改变，如打开、关闭、激活、停用、图标化或取消图标化时，需要处理事件的场合。WindowEvent 一般用 java.awt.event.WindowListener 接口监听。该接口中有如下函数：

1. void windowOpened(WindowEvent e)：窗口首次变为可见时调用。
2. void windowClosing(WindowEvent e)：用户试图从窗口的系统菜单中关闭窗口时调用。
3. void windowClosed(WindowEvent e)：因对窗口调用 dispose 而将其关闭时调用。
4. void windowIconified(WindowEvent e) ：窗口从正常状态变为最小化状态时调用。
5. void windowDeiconified(WindowEvent e)：窗口从最小化状态变为正常状态时调用。
6. void windowActivated(WindowEvent e)：将 Window 设置为活动 Window 时调用。
7. void windowDeactivated(WindowEvent e)：当 Window 不再是活动 Window 时调用。

17.6.2 使用 WindowEvent 解决实际问题

下面用一个程序进行测试，在一个按下窗口上的关闭键，询问用户是否关闭该窗口。代码如下：

WindowEventTest1.java

```
package windowevent;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
public class WindowEventTest1 extends JFrame implements WindowListener{
    public WindowEventTest1(){
        //设置关闭时默认不做任何事
        this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        this.addWindowListener(this);
        this.setSize(200,80);
        this.setVisible(true);
    }
```

```
    public void windowClosing(WindowEvent arg0) {
        int result = JOptionPane.showConfirmDialog(this, "您确认关闭吗?",
                "确认",JOptionPane.YES_NO_OPTION);
        if(result==JOptionPane.YES_OPTION){
            System.exit(0);
        }
    }
    public void windowActivated(WindowEvent arg0) {}
    public void windowClosed(WindowEvent arg0) {}
    public void windowDeactivated(WindowEvent arg0) {}
    public void windowDeiconified(WindowEvent arg0) {}
    public void windowIconified(WindowEvent arg0) {}
    public void windowOpened(WindowEvent arg0) {}
    public static void main(String[] args) {
        new WindowEventTest1();
    }
}
```

运行，效果为：

图 17-14

按下右上角的关闭按钮，显示确认框：

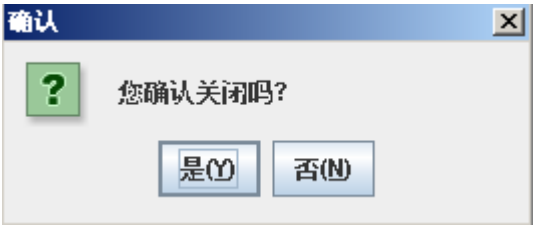


图 17-15

如果选择“是”，则关闭；如果选择“否”则不关闭。

注意

在本例中，一定要用：setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE); 设置窗口关闭时默认不做任何事，否则点击关闭按钮，界面都会关闭。

17.7 使用 Adapter 简化开发

Note

在前面的例子中，KeyEvent、MouseEvent、WindowEvent 的处理中，不约而同遇到了一个问题：Listener 接口中的函数个数较多，但是经常我们只用到一两个，由于实现一个接口，必须将接口中的函数重写一遍，因此，造成大量的空函数，用不着，不写又不行。

能否解决这个问题呢？我们知道，实现一个接口，必须将接口中的函数重写一遍，但是继承一个类，并不一定将类中的函数重写一遍，因此，java 中提供了相应的 Adapter 类来帮我们简化这个操作。

常见的 Adapter 类有：

1. KeyAdapter：内部函数和 KeyListener 基本相同。
2. MouseAdapter：内部函数和 MouseListener、MouseMotionListener 基本相同。
3. WindowAdapter：内部函数和 WindowListener 基本相同。

注意

在底层，这些 Adapter 已经实现了相应的 Listener 接口。

因此，我们编程时，就可以将事件响应的代码写在 Adapter 内。

比如，上节的 WindowEvent 的例子可以改为：

WindowAdapterTest1.java

```
package windowadapter;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
public class WindowAdapterTest1 extends JFrame    {
    public WindowAdapterTest1(){
        //设置关闭时不做任何事
        this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        this.addWindowListener(new WindowOpe());
        this.setSize(200,80);
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new WindowAdapterTest1();
    }

    class WindowOpe extends WindowAdapter{
```

```
        public void windowClosing(WindowEvent arg0) {
            int result = JOptionPane.showConfirmDialog(null, "您确认关闭吗?",
                    "确认",JOptionPane.YES_NO_OPTION);
            if(result==JOptionPane.YES_OPTION){
                System.exit(0);
            }
        }
    }
}
```

运行，效果和上节相同。

注意

此代码中，由于 Adapter 是一个类，而 Java 不支持多重继承，因此，我们不得不将事件处理代码写在另一个类 WindowOpe 类中。

阶段性作业

将前几节中，和 KeyEvent、MouseEvent 有关的程序改为用 Adapter 实现。

Note

本章知识体系

知识点	重要等级	难度等级
事件原理	★★★★	★★★★
事件开发流程	★★★★	★★★★
处理 ActionEvent	★★★	★★★
处理 FocusEvent	★★	★★
处理 KeyEvent	★★★	★★★
处理 MouseEvent	★★★	★★★
处理 WindowEvent	★★	★★
使用 Adapter 简化开发	★★	★★

第18章

实践指导 4

前面学习了 JavaGUI 开发、JavaGUI 布局和 Java 事件处理，这些内容在 Java 界面编程中，属于非常重要的内容。本章将利用一个用户管理系统的案例，来对这些内容进行复习。

术语复习

GUI _____

Layout _____

Event _____

Listener _____

JFrame _____

JDialog _____

ActionEvent _____

ActionListener _____

Component _____

18.1 用户管理系统功能简介

Note

在本章中，我们将制作一个模拟的用户管理系统。用户能够将自己的账号、密码、姓名、部门存入数据库，由于没有学习数据库操作，因此，我们将内容存入文件。

该系统由 4 个界面组成。系统运行，出现登录界面：

图 18-1

该界面出现在屏幕中间。在这个界面中：

1. 点击登录按钮，能够根据输入的账号密码进行登录；如果登录失败，能够提示；如果登录成功，提示登录成功之后，能够到达操作界面。

2. 点击注册按钮，登录界面消失，出现注册界面。

3. 点击退出按钮，程序退出。

注册界面如下：

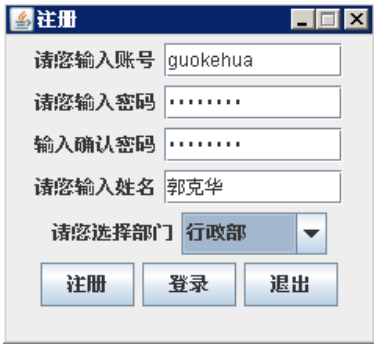


图 18-2

在这个界面中：

1. 点击注册按钮，能够根据输入的账号、密码、姓名、部门进行注册。两个密码必须相等，账号不能重复注册，部门选项如下：

图 18-3

2. 点击登录按钮，注册界面消失，出现登录界面。
3. 点击退出按钮，程序退出。

用户登录成功之后，出现操作界面，该界面效果如下：

图 18-4

在这个界面中：

1. 标题栏显示当前登录的账号。
2. 点击显示详细信息按钮，显示用户的详细信息：

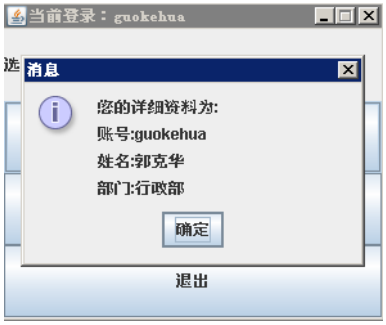


图 18-5

3. 点击退出按钮，程序退出。
4. 点击修改个人资料按钮，显示修改个人资料的对话框：

Note

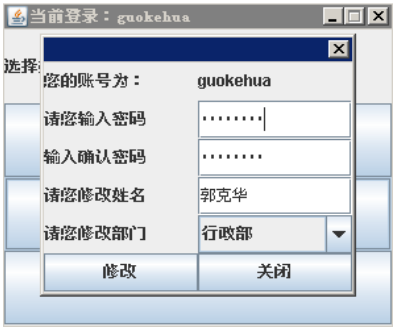


图 18-6

所有内容均初始化填入相应的控件。账号不可修改。

在这个界面中：

1. 点击修改按钮，能够修改用户信息。
2. 点击关闭按钮，能够关掉该界面。

18.2 关键技术

18.2.1 如何组织界面

在这个项目中，我们需要用到以下几个界面：登录界面，注册界面，操作界面和修改界面。很明显，这些界面各自有自己的控件和事件，四个界面应该分 4 个类，在各个类里面负责界面的界面元素和事件处理，是比较好的方法。

我们设计出来的类如下：

1.frame.LoginFrame：登录界面。

2.frame.RegisterFrame：注册界面。

3.frame.OperationFrame：操作界面。

4.frame.ModifyFrame：修改界面。

18.2.2 如何访问文件

但是，该项目有些特殊，主要是在好几个界面中都用到了文件操作，如果将文件操作的代码分散在多个界面类中，维护性较差。因此，这里有必要将文件操作的代码专门放在一个类中，让各个界面调用。

为了简化文件操作，我们将用户的信息用如下格式存储：

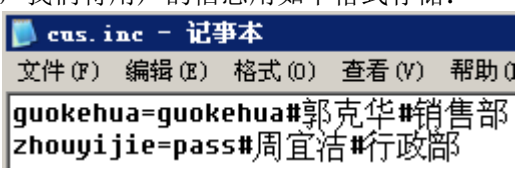


图 18-7

数据保存在 cus.inc 内，以“账号=密码#姓名#部门”的格式保存，便于用 Properties 类来读。

读文件的类设计如下：

util.FileOpe。负责读文件，将信息保存到文件。

因此，整个系统结构如图所示：

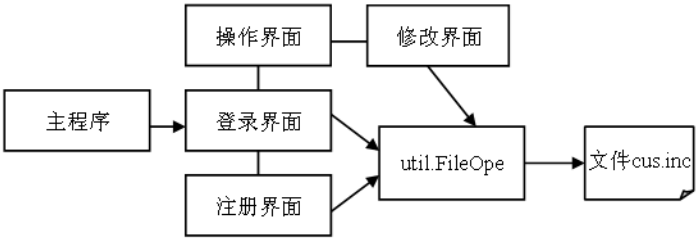


图 18-8

图 17-6 程序结构

18.2.3 如何保持状态

将项目划分为几个模块之后，模块之间的数据传递难度增大了。比如，在登录界面中，登录成功之后，系统就应该记住该用户的所有信息，否则到了操作界面，无法知道是谁在登录，到了修改界面，更无法显示其详细信息。

怎样保存其状态呢？有很多种方法，这里可以采用“静态变量法”。该方法就是将各个模块之间需要共享的数据保存在某个类的静态变量中，我们知道，静态变量一旦赋值，在另一个时刻访问，仍然是这个值，因此，可以用静态变量来传递数据。

我们设计的类如下：

util.Conf：内含 4 个静态成员：

1. public static String account;：保存登录用户的账号。
2. public static String password; ：保存登录用户的密码。
3. public static String name; ：保存登录用户的姓名。
4. public static String dept; ：保存登录用户的部门。

注意

在多线程的情况下，如果多个线程可能访问登录用户的数据，编程要十分谨慎，以免造成线程 A 将线程 B 的状态改掉的情况。不过，本项目中没有这个问题。

18.2.4 还有哪些公共功能

在本项目中，界面都要显示在屏幕中间，因此，我们可以编写一段公用代码来完成这个功能。该公用代码放在 util.GUIUtil 类中。

当然，还有一些资源文件，事先要建立好，比如，登录界面上的欢迎图片，数据文件 cus.inc 等。

Note

最终设计出来的项目结构如下：

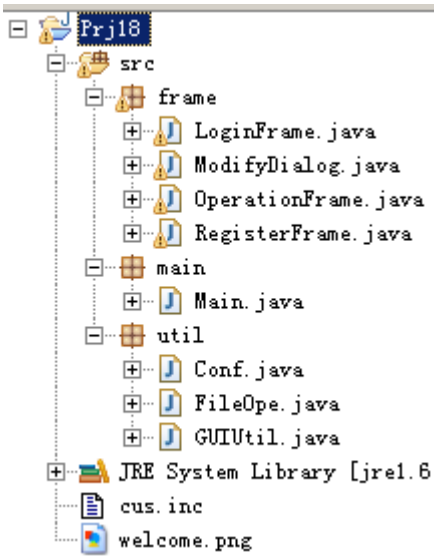


图 18-9

18.3 代码编写

18.3.1 编写 util 包中的类

首先是 Conf 类，比较简单：

Conf.java

```
package util;
public class Conf {
    public static String account;
    public static String password;
    public static String name;
    public static String dept;
}
```

然后是 FileOpe 类：

FileOpe.java

```
package util;
import java.io.FileReader;
import java.io.PrintStream;
import java.util.Properties;
import javax.swing.JOptionPane;
```

Note

```
public class FileOpe {
    private static String fileName = "cus.inc";
    private static Properties pps;
    static {
        pps = new Properties();
        FileReader reader = null;
        try{
            reader = new FileReader(fileName);
            pps.load(reader);
        }catch(Exception ex){
            JOptionPane.showMessageDialog(null, "文件操作异常");
            System.exit(0);
        }finally{
            try{
                reader.close();
            }catch(Exception ex){}
        }
    }
    private static void listInfo(){
        PrintStream ps = null;
        try{
            ps = new PrintStream(fileName);
            pps.list(ps);
        }catch(Exception ex){
            JOptionPane.showMessageDialog(null, "文件操作异常");
            System.exit(0);
        }finally{
            try{
                ps.close();
            }catch(Exception ex){}
        }
    }
    public static void getInfoByAccount(String account) {
        String cusInfo = pps.getProperty(account);
        if(cusInfo!=null){
            String[] infos = cusInfo.split("#");
            Conf.account = account;
```

Note

```
                Conf.password = infos[0];
                Conf.name = infos[1];
                Conf.dept = infos[2];
            }
        }
        public static void updateCustomer(String account,String password,
                                    String name,String dept) {
            pps.setProperty(account, password + "#" + name + "#" + dept);
            listInfo();
        }
    }
```

注意

本类中，静态代码负责载入 cus.inc 中的数据。

接下来是 FileOpe 类：

GUIUtil.java

```
package util;
import java.awt.Component;
import java.awt.GraphicsEnvironment;
import java.awt.Rectangle;
public class GUIUtil {
    public static void toCenter(Component comp){
        GraphicsEnvironment ge =
            GraphicsEnvironment.getLocalGraphicsEnvironment();
        Rectangle rec =
            ge.getDefaultScreenDevice().getDefaultConfiguration().getBounds();
        comp.setLocation(((int)rec.getWidth()-comp.getWidth())/2,
                        ((int)rec.getHeight()-comp.getHeight())/2);
    }
}
```

注意

1. 本类中，toCenter(Component comp)函数传入的参数不是 JFrame，而是其父类 Component，完全是为了扩大本函数的适用范围，让其适用于所有 Component 的子类。

2.本类中使用了界面居中的坐标计算方法，请读者仔细理解。

18.3.2 编写 frame 包中的类

首先是登录界面类：

LoginFrame.java

Note

```
package frame;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import util.Conf;
import util.FileOpe;
import util.GUIUtil;
public class LoginFrame extends JFrame implements ActionListener{
    /**********************定义各控件*****************************/
    private Icon welcomeIcon = new ImageIcon("welcome.png");
    private JLabel lbWelcome = new JLabel(welcomeIcon);
    private JLabel lbAccount = new JLabel("请您输入账号");
    private JTextField tfAccount = new JTextField(10);
    private JLabel lbPassword = new JLabel("请您输入密码");
    private JPasswordField pfPassword = new JPasswordField(10);
    private JButton btLogin = new JButton("登录");
    private JButton btRegister = new JButton("注册");
    private JButton btExit = new JButton("退出");
    public LoginFrame(){
        /************************界面初始化************************/
        super("登录");
        this.setLayout(new FlowLayout());
        this.add(lbWelcome);
        this.add(lbAccount);
        this.add(tfAccount);
        this.add(lbPassword);
        this.add(pfPassword);
        this.add(btLogin);
        this.add(btRegister);
```

Note

```
        this.add(btExit);
        this.setSize(240, 180);
        GUIUtil.toCenter(this);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.setVisible(true);
        /************************增加监听*************************/
        btLogin.addActionListener(this);
        btRegister.addActionListener(this);
        btExit.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==btLogin){
            String account = tfAccount.getText();
            String password = new String(pfPassword.getPassword());
            FileOpe.getInfoByAccount(account);
            if(Conf.account==null || !Conf.password.equals(password)){
                JOptionPane.showMessageDialog(this, "登录失败");
                return;
            }
            JOptionPane.showMessageDialog(this, "登录成功");
            this.dispose();
            new OperationFrame();
        }else if(e.getSource()==btRegister){
            this.dispose();
            new RegisterFrame();
        }else{
            JOptionPane.showMessageDialog(this, "谢谢光临");
            System.exit(0);
        }
    }
}
```

注意

本类中 this.dispose();表示让本界面消失，释放内存，但是程序不结束。System.exit(0);表示整个程序退出。

接下来是注册界面类：

RegisterFrame.java

Note

```
package frame;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import util.Conf;
import util.FileOpe;
import util.GUIUtil;
public class RegisterFrame extends JFrame implements ActionListener{
    /************************定义各控件*****************************/
    private JLabel lbAccount = new JLabel("请您输入账号");
    private JTextField tfAccount = new JTextField(10);
    private JLabel lbPassword1 = new JLabel("请您输入密码");
    private JPasswordField pfPassword1 = new JPasswordField(10);
    private JLabel lbPassword2 = new JLabel("输入确认密码");
    private JPasswordField pfPassword2 = new JPasswordField(10);
    private JLabel lbName = new JLabel("请您输入姓名");
    private JTextField tfName = new JTextField(10);
    private JLabel lbDept = new JLabel("请您选择部门");
    private JComboBox cbDept = new JComboBox();
    private JButton btRegister = new JButton("注册");
    private JButton btLogin = new JButton("登录");
    private JButton btExit = new JButton("退出");
    public RegisterFrame(){
        /************************界面初始化*************************/
        super("注册");
        this.setLayout(new FlowLayout());
        this.add(lbAccount);
        this.add(tfAccount);
        this.add(lbPassword1);
        this.add(pfPassword1);
```

Note

```
        this.add(lbPassword2);
        this.add(pfPassword2);
        this.add(lbName);
        this.add(tfName);
        this.add(lbDept);
        this.add(cbDept);
        cbDept.addItem("财务部");
        cbDept.addItem("行政部");
        cbDept.addItem("客户服务部");
        cbDept.addItem("销售部");
        this.add(btRegister);
        this.add(btLogin);
        this.add(btExit);
        this.setSize(240, 220);
        GUIUtil.toCenter(this);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.setVisible(true);
        /************************增加监听************************/
        btLogin.addActionListener(this);
        btRegister.addActionListener(this);
        btExit.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==btRegister){
            String password1 = new String(pfPassword1.getPassword());
            String password2 = new String(pfPassword2.getPassword());
            if(!password1.equals(password2)){
                JOptionPane.showMessageDialog(this, "两个密码不相同");
                return;
            }
            String account = tfAccount.getText();
            FileOpe.getInfoByAccount(account);
            if(Conf.account!=null){
                JOptionPane.showMessageDialog(this, "用户已经注册");
                return;
            }
```

Note

```
                String name = tfName.getText();
                String dept = (String)cbDept.getSelectedItem();
                FileOpe.updateCustomer(account, password1, name , dept);
                JOptionPane.showMessageDialog(this, "注册成功");
            }else if(e.getSource()==btLogin){
                this.dispose();
                new LoginFrame();
            }else{
                JOptionPane.showMessageDialog(this, "谢谢光临");
                System.exit(0);
            }
        }
    }
```

接下来是操作界面类：

OperationFrame.java

```
package frame;
import java.awt.GridLayout;
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.JOptionPane;
import util.Conf;
import util.GUIUtil;
public class OperationFrame extends JFrame implements ActionListener{
    /************************定义各控件*****************************/
    private String welcomeMsg = "选择如下操作:";
    private JLabel lbWelcome = new JLabel(welcomeMsg);
    private JButton btQuery = new JButton("显示详细信息");
    private JButton btModify = new JButton("修改个人资料");
    private JButton btExit = new JButton("退出");
    public OperationFrame(){
        /************************界面初始化*************************/
        super("当前登录：" + Conf.account);
        this.setLayout(new GridLayout(4,1));
        this.add(lbWelcome);
```

Note

```
        this.add(btQuery);
        this.add(btModify);
        this.add(btExit);
        this.setSize(300, 250);
        GUIUtil.toCenter(this);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.setVisible(true);
        /**********************增加监听**************************/
        btQuery.addActionListener(this);
        btModify.addActionListener(this);
        btExit.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==btQuery){
            String message = "您的详细资料为:\n";
            message += "账号:" + Conf.account + "\n";
            message += "姓名:" + Conf.name + "\n";
            message += "部门:" + Conf.dept + "\n";
            JOptionPane.showMessageDialog(this, message);
        }else if(e.getSource()==btModify){
            new ModifyDialog(this);
        }else{
            JOptionPane.showMessageDialog(this, "谢谢光临");
            System.exit(0);
        }
    }
}
```

最后是 ModifyDialog 类，注意，ModifyDialog 是个模态对话框。

ModifyDialog.java

```
package frame;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
```

Note

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import util.Conf;
import util.FileOpe;
import util.GUIUtil;
public class ModifyDialog extends JDialog implements ActionListener{
    /************************定义各控件*****************************/
    private JLabel lbMsg = new JLabel("您的账号为：");
    private JLabel lbAccount = new JLabel(Conf.account);
    private JLabel lbPassword1 = new JLabel("请您输入密码");
    private JPasswordField pfPassword1 = new JPasswordField(Conf.password,10);
    private JLabel lbPassword2 = new JLabel("输入确认密码");
    private JPasswordField pfPassword2 = new JPasswordField(Conf.password,10);
    private JLabel lbName = new JLabel("请您修改姓名");
    private JTextField tfName = new JTextField(Conf.name,10);
    private JLabel lbDept = new JLabel("请您修改部门");
    private JComboBox cbDept = new JComboBox();
    private JButton btModify = new JButton("修改");
    private JButton btExit = new JButton("关闭");
    public ModifyDialog(JFrame frm){
        /************************界面初始化************************/
        super(frm,true);
        this.setLayout(new GridLayout(6,2));
        this.add(lbMsg);
        this.add(lbAccount);
        this.add(lbPassword1);
        this.add(pfPassword1);
        this.add(lbPassword2);
        this.add(pfPassword2);
        this.add(lbName);
        this.add(tfName);
        this.add(lbDept);
        this.add(cbDept);
        cbDept.addItem("财务部");
```

Note

```
            cbDept.addItem("行政部");
            cbDept.addItem("客户服务部");
            cbDept.addItem("销售部");
            cbDept.setSelectedItem(Conf.dept);
            this.add(btModify);
            this.add(btExit);
            this.setSize(240, 200);
            GUIUtil.toCenter(this);
            this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            /************************增加监听*************************/
            btModify.addActionListener(this);
            btExit.addActionListener(this);
            this.setResizable(false);
            this.setVisible(true);
        }
        public void actionPerformed(ActionEvent e) {
            if(e.getSource()==btModify){
                String password1 = new String(pfPassword1.getPassword());
                String password2 = new String(pfPassword2.getPassword());
                if(!password1.equals(password2)){
                    JOptionPane.showMessageDialog(this, "两个密码不相同");
                    return;
                }
                String name = tfName.getText();
                String dept = (String)cbDept.getSelectedItem();
                //将新的值存入静态变量
                Conf.password = password1;
                Conf.name = name;
                Conf.dept = dept;
                FileOpe.updateCustomer(Conf.account, password1, name , dept);
                JOptionPane.showMessageDialog(this, "修改成功");
            }else{
                this.dispose();
            }
        }
    }
```

18.3.3 编写主函数所在的类

主函数所在的类，调用登录界面类：

Main.java

```
package main;
import frame.LoginFrame;
public class Main {
    public static void main(String[] args) {
            new LoginFrame();
    }
}
```

运行该类，则可以出现登录界面。

Note

18.4 思考题

本程序开发完毕，留下几个思考题，请大家思考：

1. 该程序中，需要用 Properties 类将整个文件读入进行处理，如果遇到文件较大的情况，会有什么问题？如何解决？

2. 将用户的登录信息用静态变量存储，在多线程情况下，有什么隐患？你能否举出一个例子？如何解决？