

第 3 篇

高级篇


- » 第 14 章 高级事件处理
- » 第 15 章 多线程
- » 第 16 章 网络通信
- » 第 17 章 JDBC 操作数据库
- » 第 18 章 Swing 高级组件
- » 第 19 章 高级布局管理器
- » 第 20 章 AWT 绘图技术

本篇介绍了高级事件处理、多线程、网络通信、JDBC 操作数据库、Swing 高级组件、高级布局管理器、AWT 绘图技术等内容。学习完本篇后，能够开发高级的桌面应用程序、多媒体程序和打印程序等。

[illegible]

第14章

高级事件处理

( 视频讲解：30 分钟)

本章将讲解一些常用高级事件的处理方法，包括键盘事件、鼠标事件、窗体事件、选项事件和表格模型事件，通过捕获这些事件并对其进行处理，可以更进一步控制程序的流程，保证每一步操作的合法性，实现一些更人性化的性能。例如，通过捕获键盘事件验证输入数据的合法性，通过捕获表格模型事件实现自动计算表格某一列的和等。

通过阅读本章，您可以：

- » 学会处理键盘事件
- » 学会处理鼠标事件
- » 学会处理窗体焦点变化、状态变化等事件
- » 学会处理选项事件
- » 学会处理表格模型事件

第14章
高级事件处理
PDG

14.1 键盘事件

 视频讲解：光盘\TM\14\键盘事件.exe

当向文本框中输入内容时，将发出键盘事件（KeyEvent）。KeyEvent 类负责捕获键盘事件，可以通过为组件添加实现了 KeyListener 接口的监听器类来处理相应的键盘事件。

KeyListener 接口共有 3 个抽象方法，分别在发生击键事件、按键被按下和释放时触发。KeyListener 接口的具体定义如下：

```
public interface KeyListener extends EventListener {
    public void keyTyped(KeyEvent e);           //发生击键事件时触发
    public void keyPressed(KeyEvent e);         //按键被按下时触发
    public void keyReleased(KeyEvent e);        //按键被释放时触发
}
```

在每个抽象方法中均传入了 KeyEvent 类的对象，KeyEvent 类中比较常用的方法如表 14.1 所示。

表 14.1 KeyEvent 类中的常用方法

方 法	功 能 简 介
getSource()	用来获得触发此次事件的组件对象，返回值为 Object 类型
getKeyChar()	用来获得与此事件中的键相关联的字符
getKeyCode()	用来获得与此事件中的键相关联的整数 keyCode
getKeyText(int keyCode)	用来获得描述 keyCode 的标签，如 A、F1 和 HOME 等
isActionKey()	用来查看此事件中的键是否为“动作”键
isControlDown()	用来查看 Ctrl 键在此次事件中是否被按下，当返回 true 时表示被按下
isAltDown()	用来查看 Alt 键在此次事件中是否被按下，当返回 true 时表示被按下
isShiftDown()	用来查看 Shift 键在此次事件中是否被按下，当返回 true 时表示被按下

 **技巧**

在 KeyEvent 类中以 VK_ 开头的静态常量代表各个按键的 keyCode，可以通过这些静态常量判断事件中的按键，以及获得按键的标签。

【例 14.1】 演示捕获和处理键盘事件的方法，尤其是键盘事件监听器接口 KeyListener 中各个方法的使用方法。（实例位置：光盘\TM\14\1）

```
final JLabel label = new JLabel();
label.setText("备注：");
getContentPane().add(label, BorderLayout.WEST);
final JScrollPane scrollPane = new JScrollPane();
getContentPane().add(scrollPane, BorderLayout.CENTER);
JTextArea textArea = new JTextArea();
textArea.addKeyListener(new KeyListener() {
    public void keyPressed(KeyEvent e) {           //按键被按下时触发
```

```

String keyText = KeyEvent.getKeyText(e.getKeyCode()); //获得描述 keyCode 的标签
if (e.isActionKey()) { //判断按下的是否为动作键
    System.out.println("您按下的是动作键 " + keyText + " ");
} else {
    System.out.println("您按下的是非动作键 " + keyText + " ");
    int keyCode = e.getKeyCode(); //获得与此事件中的键相关联的字符
    switch (keyCode) {
        case KeyEvent.VK_CONTROL: //判断按下的是否为 Ctrl 键
            System.out.println("Ctrl 键被按下");
            break;
        case KeyEvent.VK_ALT: //判断按下的是否为 Alt 键
            System.out.println("Alt 键被按下");
            break;
        case KeyEvent.VK_SHIFT: //判断按下的是否为 Shift 键
            System.out.println("Shift 键被按下");
            break;
    }
    System.out.println();
}
}

public void keyTyped(KeyEvent e) { //发生击键事件时触发
    System.out.println("此次输入的是 " + e.getKeyChar() + " "); //获得输入的字符
}

public void keyReleased(KeyEvent e) { //按键被释放时触发
    String keyText = KeyEvent.getKeyText(e.getKeyCode()); //获得描述 keyCode 的标签
    System.out.println("您释放的是 " + keyText + " 键");
    System.out.println();
}

});
textArea.setLineWrap(true);
textArea.setRows(3);
textArea.setColumns(15);
scrollPane.setViewPortView(textArea);

```

运行本实例，首先输入小写字母“m”，然后输入一个空格，接下来输入大写字母“M”，再按 Shift 键，最后按 F5 键，运行结果如图 14.1 所示。

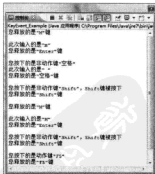


图 14.1 键盘事件

14.2 鼠标事件

视频讲解：光盘\TM\14\鼠标事件.exe

所有组件都能发出鼠标事件（MouseEvent），MouseEvent 类负责捕获鼠标事件，可以通过为组件添加实现了 MouseListener 接口的监听器类来处理相应的鼠标事件。

MouseListener 接口共有 5 个抽象方法，分别在光标移入或移出组件时、鼠标按键被按下或释放时和发生单击事件时触发。所谓单击事件，就是按键被按下并释放。需要注意的是，如果按键是在移出组件之后才被释放，则不会触发单击事件。MouseListener 接口的具体定义如下：

```
public interface MouseListener extends EventListener {
    public void mouseEntered(MouseEvent e);           //光标移入组件时触发
    public void mousePressed(MouseEvent e);           //鼠标按键被按下时触发
    public void mouseReleased(MouseEvent e);          //鼠标按键被释放时触发
    public void mouseClicked(MouseEvent e);          //发生单击事件时触发
    public void mouseExited(MouseEvent e);           //光标移出组件时触发
}
```

在每个抽象方法中均传入了 MouseEvent 类的对象，MouseEvent 类中比较常用的方法如表 14.2 所示。

表 14.2 MouseEvent 类中的常用方法

方 法	功 能 简 介
getSource()	用来获得触发此次事件的组件对象，返回值为 Object 类型
getButton()	用来获得代表触发此次按下、释放或单击事件的按键的 int 型值
getClickCount()	用来获得单击按键的次数

当需要判断触发此次事件的按键时，可以通过表 14.3 中的静态常量判断由 getButton()方法返回的 int 型值代表的键。

表 14.3 MouseEvent 类中代表鼠标按键的静态常量

静 态 常 量	常 量 值	代 表 的 键
BUTTON1	1	代表鼠标左键
BUTTON2	2	代表鼠标滚轮
BUTTON3	3	代表鼠标右键

【例 14.2】 演示捕获和处理鼠标事件的方法，尤其是鼠标事件监听器接口 MouseListener 中各个方法的使用方法。（实例位置：光盘\TM\14\2）

```
final JLabel label = new JLabel();
label.addMouseListener(new MouseListener() {
    public void mouseEntered(MouseEvent e) {           //光标移入组件时触发
        System.out.println("光标移入组件");
    }
}
```

```

public void mousePressed(MouseEvent e) { //鼠标按键被按下时触发
    System.out.print("鼠标按键被按下, ");
    int i = e.getButton(); //通过该值可以判断按下的是哪个键
    if (i == MouseEvent.BUTTON1)
        System.out.println("按下的是鼠标左键");
    if (i == MouseEvent.BUTTON2)
        System.out.println("按下的是鼠标滚轮");
    if (i == MouseEvent.BUTTON3)
        System.out.println("按下的是鼠标右键");
}

public void mouseReleased(MouseEvent e) { //鼠标按键被释放时触发
    System.out.print("鼠标按键被释放, ");
    int i = e.getButton(); //通过该值可以判断释放的是哪个键
    if (i == MouseEvent.BUTTON1)
        System.out.println("释放的是鼠标左键");
    if (i == MouseEvent.BUTTON2)
        System.out.println("释放的是鼠标滚轮");
    if (i == MouseEvent.BUTTON3)
        System.out.println("释放的是鼠标右键");
}

public void mouseClicked(MouseEvent e) { //发生单击事件时触发
    System.out.print("单击了鼠标按键, ");
    int i = e.getButton(); //通过该值可以判断单击的是哪个键
    if (i == MouseEvent.BUTTON1)
        System.out.print("单击的是鼠标左键, ");
    if (i == MouseEvent.BUTTON2)
        System.out.print("单击的是鼠标滚轮, ");
    if (i == MouseEvent.BUTTON3)
        System.out.print("单击的是鼠标右键, ");
    int clickCount = e.getClickCount();
    System.out.println("单击次数为" + clickCount + "下");
}

public void mouseExited(MouseEvent e) { //光标移出组件时触发
    System.out.println("光标移出组件");
}
}

```

运行本实例, 首先将光标移入窗体, 然后单击鼠标左键, 接着双击鼠标右键, 最后将光标移出窗体, 运行结果如图 14.2 所示。



图 14.2 鼠标事件



●注意

从图 14.2 中可以发现，当双击鼠标时，第一次单击鼠标将触发一次单击事件。

14.3 窗体事件

视频讲解：光盘\TM\lx\14\窗体事件.exe

在捕获窗体事件（WindowEvent）时，可以通过 3 个事件监听器接口来实现，分别为 WindowFocusListener、WindowStateListener 和 WindowListener。本节将深入学习这 3 种事件监听器的使用方法，主要是各自捕获的事件类型和各个抽象方法的触发条件。

14.3.1 捕获窗体焦点变化事件

需要捕获窗体焦点发生变化的事件时，即窗体获得或失去焦点的事件时，可以通过实现了 WindowFocusListener 接口的事件监听器完成。WindowFocusListener 接口的具体定义如下：

```
public interface WindowFocusListener extends EventListener {
    public void windowGainedFocus(WindowEvent e);           //窗体获得焦点时触发
    public void windowLostFocus(WindowEvent e);            //窗体失去焦点时触发
}
```

通过捕获窗体获得或失去焦点的事件，可以进行一些相关操作，例如当窗体重新获得焦点时，令所有组件均恢复为默认设置。

【例 14.3】 演示捕获和处理窗体焦点变化事件的方法，尤其是窗体焦点事件监听器接口 WindowFocusListener 中各个方法的使用方法。（实例位置：光盘\TM\lx\14\3）

```
public class WindowFocusListener_Example extends JFrame {
    public static void main(String args[]) {
        WindowFocusListener_Example frame = new WindowFocusListener_Example();
        frame.setVisible(true);
    }
    public WindowFocusListener_Example() {
        super();
        addWindowFocusListener(new MyWindowFocusListener()); //为窗体添加焦点事件监听器
        setTitle("捕获窗体焦点事件");
        setBounds(100, 100, 500, 375);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
    private class MyWindowFocusListener implements WindowFocusListener {
        public void windowGainedFocus(WindowEvent e) { //窗口获得焦点时触发
            System.out.println("窗口获得了焦点！");
        }
    }
}
```



```

    }
    public void windowLostFocus(WindowEvent e) { //窗口失去焦点时触发
        System.out.println("窗口失去了焦点!");
    }
}

```

运行结果如图 14.3 所示。



图 14.3 捕获窗体焦点事件监听器

14.3.2 捕获窗体状态变化事件

需要捕获窗体状态发生变化的事件时，即窗体由正常化变为图标化、由最大化变为正常化等事件时，可以通过实现 WindowStateListener 接口的事件监听器来完成。WindowStateListener 接口的具体定义如下：

```

public interface WindowStateListener extends EventListener {
    public void windowStateChanged(WindowEvent e); //窗体状态发生变化时触发
}

```

在抽象方法 windowStateChanged() 中传入了 WindowEvent 类的对象。WindowEvent 类中有如下两个常用方法，用来获得窗体的状态，它们均返回一个代表窗体状态的 int 型值。

- ☒ getNewState(): 用来获得窗体现在的状态。
- ☒ getOldState(): 用来获得窗体以前的状态。

可以通过 Frame 类中的静态常量判断返回的 int 型值具体代表什么状态，这些静态常量如表 14.4 所示。

表 14.4 Frame 类中代表窗体状态的静态常量

静态常量	常量值	代表的键
NORMAL	0	代表窗体处于“正常化”状态
ICONIFIED	1	代表窗体处于“图标化”状态
MAXIMIZED_BOTH	6	代表窗体处于“最大化”状态

【例 14.4】 演示捕获和处理窗体状态变化事件的方法，尤其是窗体状态变化事件监听器接口 WindowStateListener 中各个方法的使用方法。（实例位置：光盘\TM\sl\14\4）

```

public class WindowStateListener_Example extends JFrame {
    public static void main(String args[]) {
        WindowStateListener_Example frame = new WindowStateListener_Example();
        frame.setVisible(true);
    }
    public WindowStateListener_Example() {
        super();
        addWindowStateListener(new MyWindowStateListener()); //为窗体添加状态事件监听器
        setTitle("捕获窗体状态事件");
        setBounds(100, 100, 500, 375);
    }
}

```

```

        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
    private class MyWindowStateListener implements WindowStateListener {
        public void windowStateChanged(WindowEvent e) {
            int oldState = e.getOldState();
            int newState = e.getNewState();
            String from = "";
            String to = "";
            switch (oldState) {
                case Frame.NORMAL:
                    from = "正常化";
                    break;
                case Frame.MAXIMIZED_BOTH:
                    from = "最大化";
                    break;
                default:
                    from = "图标化";
            }
            switch (newState) {
                case Frame.NORMAL:
                    to = "正常化";
                    break;
                case Frame.MAXIMIZED_BOTH:
                    to = "最大化";
                    break;
                default:
                    to = "图标化";
            }
            System.out.println(from + "——>" + to);
        }
    }
}

```

//获得窗体以前的状态
 //获得窗体现在的状态
 //标识窗体以前状态的中文字符串
 //标识窗体现在状态的中文字符串
 //判断窗体以前的状态
 //窗体处于正常化
 //窗体处于最大化
 //窗体处于图标化
 //判断窗体现在的状态
 //窗体处于正常化
 //窗体处于最大化
 //窗体处于图标化

运行本实例，首先将窗体图标化后再恢复正常化，然后将窗体最大化后再图标化，最后将窗体最大化后再恢复正常化，在控制台将得到如图 14.4 所示的信息。

14.3.3 捕获其他窗体事件

需要捕获其他与窗体有关的事件时，例如捕获窗体被打开、将要被关闭、已经被关闭等事件时，可以通过实现了 WindowListener 接口的事件监听器完成。WindowListener 接口的具体定义如下：

```

public interface WindowListener extends EventListener {
    public void windowActivated(WindowEvent e);
    public void windowOpened(WindowEvent e);
    public void windowIconified(WindowEvent e);
}

```

//窗体被激活时触发
 //窗体被打开时触发
 //窗体被图标化时触发



图 14.4 捕获窗体状态变化事件

```

public void windowDeiconified(WindowEvent e);           // 窗体被非图标化时触发
public void windowClosing(WindowEvent e);              // 窗体将要被关闭时触发
public void windowDeactivated(WindowEvent e);          // 窗体不再处于激活状态时触发
public void windowClosed(WindowEvent e);               // 窗体已经被关闭时触发
}

```

通过捕获窗体将要被关闭等事件，可以进行一些相关操作，例如当窗体将要被关闭时，询问是否保存未保存的设置等。

【例 14.5】 演示捕获和处理其他窗体事件的方法，尤其是事件监听器接口 WindowListener 中各个方法的使用方法。（实例位置：光盘\TM\sl\14\5）

```

public class WindowListener_Example extends JFrame {
    public static void main(String args[]) {
        WindowListener_Example frame = new WindowListener_Example();
        frame.setVisible(true);
    }
    public WindowListener_Example() {
        super();
        addWindowListener(new MyWindowListener()); // 为窗体添加其他事件监听器
        setTitle("捕获其他窗体事件");
        setBounds(100, 100, 500, 375);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
    private class MyWindowListener implements WindowListener {
        public void windowActivated(WindowEvent e) { // 窗体被激活时触发
            System.out.println("窗口被激活！");
        }
        public void windowOpened(WindowEvent e) { // 窗体被打开时触发
            System.out.println("窗口被打开！");
        }
        public void windowIconified(WindowEvent e) { // 窗体被图标化时触发
            System.out.println("窗口被图标化！");
        }
        public void windowDeiconified(WindowEvent e) { // 窗体被非图标化时触发
            System.out.println("窗口被非图标化！");
        }
        public void windowClosing(WindowEvent e) { // 窗体将要被关闭时触发
            System.out.println("窗口将要被关闭！");
        }
        public void windowDeactivated(WindowEvent e) { // 窗体不再处于激活状态时触发
            System.out.println("窗口不再处于激活状态！");
        }
        public void windowClosed(WindowEvent e) { // 窗体已经被关闭时触发
            System.out.println("窗口已经被关闭！");
        }
    }
}

```

运行结果如图 14.5 所示。

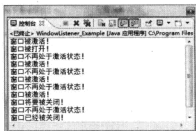


图 14.5 捕获其他窗体事件

14.4 选项事件

视频讲解：光盘\TM\14\14选项事件.exe

当修改下拉菜单中的选中项时，将发出选项事件（ItemEvent）。ItemEvent 类负责捕获选项事件，可以通过为组件添加实现了 ItemListener 接口的监听器类来处理相应的选项事件。

ItemListener 接口只有一个抽象方法，在修改一次下拉菜单选中项的过程中，该方法将被触发两次，一次是由取消原来选中项的选中状态触发的，另一次是由选中新选项触发的。ItemListener 接口的具体定义如下：

```
public interface ItemListener extends EventListener {
    void itemStateChanged(ItemEvent e);
}
```

在抽象方法 itemStateChanged() 中传入了 ItemEvent 类的对象。ItemEvent 类中有如下两个常用方法。

(1) getItem()

用来获得触发此次事件的选项，该方法的返回值为 Object 型。

(2) getStateChange()

用来获得此次事件的类型，即是由取消原来选中项的选中状态触发的，还是由选中新选项触发的。getStateChange() 方法将返回一个 int 型值，可以通过 ItemEvent 类中的如下静态常量判断此次事件的具体类型。

- ☒ SELECTED：如果返回值等于该静态常量，说明此次事件是由选中新选项触发的。
- ☒ DESELECTED：如果返回值等于该静态常量，说明此次事件是由取消原来选中项的选中状态触发的。

通过捕获选项事件，可以进行一些相关操作，如同步处理其他下拉菜单的可选项。

【例 14.6】 演示捕获和处理选项事件的方法，尤其是事件监听器接口 ItemListener 中各个方法的使用方法。（实例位置：光盘\TM\14\146）

```
JComboBox comboBox = new JComboBox(); //创建一个下拉菜单
for (int i = 1; i < 6; i++) {           //通过循环添加选项
    comboBox.addItem("选项" + i);
}
```

```

}
comboBox.addItemListener(new ItemListener() {                                //添加选项事件监听器
    public void itemStateChanged(ItemEvent e) {
        int stateChange = e.getStateChange();                                //获得事件类型
        String item = e.getItem().toString();                                //获得触发此次事件的选项
        if (stateChange == ItemEvent.SELECTED) {                            //查看是否由选中选项触发
            System.out.println("此次事件由 选中 选项 "" + item + "" 触发!");
        } else if (stateChange == ItemEvent.DESELECTED) {                  //查看是否由取消选中选项触发
            System.out.println("此次事件由 取消选中 选项 "" + item + "" 触发!");
        } else {                                                            //由其他原因触发
            System.out.println("此次事件由其他原因触发!");
        }
    }
});

```

运行结果如图 14.6 所示。

首先将选中项由“选项 1”改为“选项 2”；然后将选中项由“选项 2”改为“选项 2”；最后将选中项由“选项 2”改为“选项 5”，将得到如图 14.7 所示的信息。



图 14.6 运行结果



图 14.7 选项事件



注意

当选中项未发生变化时，并不会触发选项事件，例如在将选中项由“选项 2”改为“选项 2”时，在控制台并未输出信息。

14.5 表格模型事件



视频讲解：光盘\TM\14\表格模型事件.exe

当向表格模型中添加行时，或者是修改或删除表格模型中的现有行时，将发出表格模型事件（TableModelEvent）。TableModelEvent 类负责捕获表格模型事件，可以通过为组件添加实现了 TableModelListener 接口的监听器类来处理相应的表格模型事件。

TableModelListener 接口只有一个抽象方法，当向表格模型中添加行时，或者是修改或删除表格模型中的现有行时，该方法将被触发。TableModelListener 接口的具体定义如下：

```

public interface TableModelListener extends java.util.EventListener {
    public void tableChanged(TableModelEvent e);
}

```

在抽象方法 `tableChanged()` 中传入了 `TableModelEvent` 类的对象，`TableModelEvent` 类中比较常用的方法如表 14.5 所示。

表 14.5 `TableModelEvent` 类中的常用方法

方 法	功 能 简 介
<code>getType()</code>	获得此次事件的类型
<code>getFirstRow()</code>	获得触发此次事件的表格行的最小索引值
<code>getLastRow()</code>	获得触发此次事件的表格行的最大索引值
<code>getColumn()</code>	如果事件类型为 <code>UPDATE</code> ，获得触发此次事件的表格列的索引值；否则将返回 -1

`getType()` 方法将返回一个 `int` 型值，可以通过 `TableModelEvent` 类中的如下静态常量判断此次事件的具体类型。

- ☑ `INSERT`：如果返回值等于该静态常量，说明此次事件是由插入行触发的。
- ☑ `UPDATE`：如果返回值等于该静态常量，说明此次事件是由修改行触发的。
- ☑ `DELETE`：如果返回值等于该静态常量，说明此次事件是由删除行触发的。

通过捕获表格模型事件，可以进行一些相关操作，如自动计算表格某一列的总和。

【例 14.7】 演示捕获和处理表格模型事件的方法，尤其是事件监听器接口 `TableModelEvent` 中各个方法的使用方法。（实例位置：光盘\TM\sl\14\7）

```
public class TableModelEvent_Example extends JFrame {
    private JTable table;           //声明一个表格对象
    private DefaultTableModel tableModel; //声明一个表格模型对象
    private JTextField aTextField;
    private JTextField bTextField;
    public static void main(String args[]) {
        TableModelEvent_Example frame = new TableModelEvent_Example();
        frame.setVisible(true);
    }
    public TableModelEvent_Example() {
        super();
        setTitle("表格模型事件示例");
        setBounds(100, 100, 500, 375);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final JScrollPane scrollPane = new JScrollPane();
        getContentPane().add(scrollPane, BorderLayout.CENTER);
        String[] columnNames = { "A", "B" };
        String[][] rowValues = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" }, { "A4", "B4" } };
        //创建表格模型对象
        tableModel = new DefaultTableModel(rowValues, columnNames);
        //为表格模型添加事件监听器
        tableModel.addTableModelListener(new TableModelListener() {
            public void tableChanged(TableModelEvent e) {
                int type = e.getType(); //获得事件的类型
                int row = e.getFirstRow() + 1; //获得触发此次事件的表格行索引
                int column = e.getColumn() + 1; //获得触发此次事件的表格列索引
                if (type == TableModelEvent.INSERT) { //判断是否由插入行触发
```

```

        System.out.print("此次事件由 插入 行触发，");
        System.out.println("此次插入的是第 " + row + " 行！");
    } else if (type == TableModelEvent.UPDATE) { //判断是否由修改行触发
        System.out.print("此次事件由 修改 行触发，");
        System.out.println("此次修改的是第 " + row + " 行第 " + column + " 列！");
    } else if (type == TableModelEvent.DELETE) { //判断是否由删除行触发
        System.out.print("此次事件由 删除 行触发，");
        System.out.println("此次删除的是第 " + row + " 行！");
    } else {
        System.out.println("此次事件由 其他原因 触发！");
    }
}

});
table = new JTable(tableModel); //利用表格模型对象创建表格对象
scrollPane.setViewportView(table);
final JPanel panel = new JPanel();
getContentPane().add(panel, BorderLayout.SOUTH);
final JLabel aLabel = new JLabel("A: ");
panel.add(aLabel);
aTextField = new JTextField(15);
panel.add(aTextField);
final JLabel bLabel = new JLabel("B: ");
panel.add(bLabel);
bTextField = new JTextField(15);
panel.add(bTextField);
final JButton addButton = new JButton("添加");
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String[] rowValues = { aTextField.getText(), aTextField.getText() };
        tableModel.addRow(rowValues); //向表格模型中添加一行
        aTextField.setText(null);
        bTextField.setText(null);
    }
});
panel.add(addButton);
final JButton delButton = new JButton("删除");
delButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int[] selectedRows = table.getSelectedRows(); //获得表格中的选中行
        for (int row = 0; row < selectedRows.length; row++) {
            //从表格模型中移除表格中的选中行
            tableModel.removeRow(selectedRows[row] - row);
        }
    }
});
panel.add(delButton);
}
}
}

```

运行结果如图 14.8 所示。

首先向表格中添加一行；然后依次双击值为 A2 和 B4 的单元格对其进行修改；最后选中表格的第 3 行和第 5 行，单击下面的“删除”按钮删除这两行，输出信息如图 14.9 所示。



图 14.8 表格模型事件示例



图 14.9 测试后控制台信息

14.6 经典范例

14.6.1 经典范例 1: 模拟相机拍摄

 视频讲解: 光盘\TM\14\模拟相机拍摄.exe

在使用数码相机拍摄时，用户可以通过屏幕上提供的聚焦图标来选择拍摄的位置。本范例将实现类似功能，用户可以通过按方向键来完成移动镜头的操作，例如按→键可以让镜头向右移动。运行结果如图 14.10 所示。(实例位置: 光盘\TM\14\8)

(1) 在 Eclipse 中创建 JFrame 窗体, 名为 KeyMove Background, 对于图片, 使用标签来保存。将保存背景图片的标签设置成窗体的背景, 将保存镜头图片的标签应用到玻璃面板中。代码如下:



图 14.10 运行结果

```
public KeyMoveBackground() {
    super();
    setResizable(false);
    getContentPane().setLayout(null);
    setTitle("方向键移动背景");
    setBounds(100, 100, 500, 375);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    label = new JLabel();
    ImageIcon icon = new ImageIcon(getClass().getResource("background.jpg"));
    ImageIcon glassimg = new ImageIcon(getClass().getResource("glass.png"));
    label.setIcon(icon);
    label.setSize(icon.getIconWidth(), icon.getIconHeight());
    label.setLocation(-icon.getIconWidth() / 3, -icon.getIconHeight() / 3);
    addKeyListener(new KeyAdapter() {
        public void keyPressed(final KeyEvent e) {
            do_label_keyPressed(e);
        }
    });
}
```

//禁止调整窗体大小
//设置空布局
//设置窗体标题
//设置窗体位置 and 大小
//设置窗体退出时操作
//创建标签控件
//设置标签使用背景图像
//使标签与图像同步大小
//设置标签默认位置
//窗体添加按键事件监听适配器
//调用事件处理方法


```

getContentPane().add(label);           //添加背景标签到窗体
JLabel glassLabel = new JLabel(glassImg); //创建取景框标签
JPanel glassPane = new JPanel(new BorderLayout());
glassPane.add(glassLabel, BorderLayout.CENTER); //添加取景框标签到玻璃面板
glassPane.setOpaque(false);             //使面板透明
setGlassPane(glassPane);              //设置窗体使用玻璃面板
getGlassPane().setVisible(true);       //显示玻璃面板
}

```

(2) 监听用户按键盘上方向键事件，通过移动背景图片标签来表现移动镜头的效果。每次移动 3 个像素，例如按→键可以让保存背景图片的标签向左移动 3 像素。代码如下：

```

protected void do_label_keyPressed(final KeyEvent e) {
    int code = e.getKeyCode();           //获取按键代码
    Point location = label.getLocation(); //获取标签控件位置
    int step = 3;                         //移动速度
    switch (code) {
        case KeyEvent.VK_RIGHT:          //如果按键代码是右方向键
            if (location.x > (getWidth() - label.getWidth())) //在不出屏幕情况下
                label.setLocation(location.x - step, location.y); //向左移动标签
            break;
        case KeyEvent.VK_LEFT:           //如果按键代码是左方向键
            if (location.x < 0)            //在不出屏幕情况下
                label.setLocation(location.x + step, location.y); //向右移动标签
            break;
        case KeyEvent.VK_DOWN:           //如果按键代码是下方向键
            if (location.y > (getHeight() - label.getHeight())) //在不出屏幕情况下
                label.setLocation(location.x, location.y - step); //向上移动标签
            break;
        case KeyEvent.VK_UP:             //如果按键代码是上方向键
            if (location.y < 0)            //在不出屏幕情况下
                label.setLocation(location.x, location.y + step); //向下移动标签
            break;
        default:                          //其他情况
            break;
    }
}

```

14.6.2 经典范例 2：打地鼠游戏

 视频讲解：光盘\TM\14\打地鼠游戏.exe

打地鼠是一款非常简单易学的游戏，玩家使用鼠标不断地单击图片上的地鼠图标即可。本范例将完成这个游戏，其核心思想是监听用户单击鼠标事件，如果单击的位置有地鼠（一个包含图片的标签），则去掉标签上的图片。运行结果如图 14.11 所示。（实例位置：光盘\TM\14\9）



图 14.11 运行结果

(1) 在 Eclipse 中创建 JFrame 窗体，名为 Shrewmouse，在窗体中增加 7 个标签，一个用来显示游戏的背景图片，另外 6 个用来显示地鼠图片。监听标签上鼠标单击事件，如果该标签上有图标，则将其删除。代码如下：

```
public Shrewmouse() {
    super();
    setResizable(false); //禁止调整窗体大小
    getContentPane().setLayout(null); //窗体不使用布局管理器
    setTitle("简易打地鼠游戏"); //设置窗体标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗体关闭属性
    //初始化背景图片对象
    ImageIcon img = new ImageIcon(getClass().getResource("background.jpg"));
    ImageIcon imgMouse = new ImageIcon(getClass().getResource("mouse.png")); //初始化地鼠图片对象
    JLabels[] mouses = new JLabel[6]; //创建显示地鼠的标签数组
    for (int i = 0; i < 6; i++) { //遍历数组
        mouses[i] = new JLabel(); //初始化每一个数组元素
        //设置标签与地鼠图片相同大小
        mouses[i].setSize(imgMouse.getWidth(), imgMouse.getHeight());
        //为标签添加鼠标事件监听适配器
        mouses[i].addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                Object source = e.getSource();
                if (source instanceof JLabel) {
                    JLabel mouse = (JLabel) source;
                    mouse.setIcon(null); //取消标签图标
                }
            }
        });
        getContentPane().add(mouses[i]); //添加显示地鼠的标签到窗体
    }
}
```

```

mouses[0].setLocation(253, 300); //设置每个标签的位置
mouses[1].setLocation(333, 250); //设置每个标签的位置
mouses[2].setLocation(388, 296); //设置每个标签的位置
mouses[3].setLocation(362, 364); //设置每个标签的位置
mouses[4].setLocation(189, 353); //设置每个标签的位置
mouses[5].setLocation(240, 409); //设置每个标签的位置
final JLabel backLabel = new JLabel(); //创建显示背景的标签
backLabel.setBounds(0, 0, img.getIconWidth(), img.getIconHeight()); //设置标签与背景图片相同大小
setBounds(100, 100, img.getIconWidth(), img.getIconHeight() + 30); //设置窗体近似背景图片大小
backLabel.setIcon(img); //添加背景到标签
getContentPane().add(backLabel); //添加背景标签到窗体
}

```

(2) 编写多线程的核心方法，它可以每隔一秒钟随机让一个没有显示图片的标签显示地鼠图片。代码如下：

```

public void run() {
    while (true) { //使用无限循环
        try {
            Thread.sleep(1000); //使线程休眠 1 秒
            int index = (int) (Math.random() * 6); //生成随机的地鼠索引
            if (mouses[index].getIcon() == null) //如果地鼠标签没有设置图片
                mouses[index].setIcon(imgMouse); //为该标签添加地鼠图片
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

14.7 本章小结

通过本章的学习，相信读者已经可以熟练地处理一些高级事件，包括键盘事件、鼠标事件、窗体事件、选项事件和表格模型事件。至此，已经掌握了 5 种事件的处理方法，通过配合使用这 5 种事件监听器，可以充分控制各种组件，例如通过为文本框组件同时添加焦点事件监听器和键盘事件监听器，可以有效地控制文本框中输入的内容。

14.8 实战练习

1. 设计一个通过捕获文本框的键盘事件实现只允许输入数字的文本框。(答案位置：光盘\TM\sl\14\10)
2. 设计一个通过捕获表格模型事件实现自动计算表格某一数值列的和。(答案位置：光盘\TM\sl\14\11)

