

Java程序设计

第9章 组件及事件处理

导读

□ 主要内容

- ◆ Java Swing概述
- ◆ 窗口
- ◆ 常用组件与布局
- ◆ 处理事件
- ◆ 使用MVC结构
- ◆ 对话框
- ◆ 发布GUI程序

□ 重点和难点

- ◆ 重点：Swing包中的各种组件，各种布局和事件处理器的应用
- ◆ 难点：各种事件处理器的使用

9.2.2 菜单条、菜单、菜单项

① 菜单条

② 菜单

③ 菜单项



图 9.3 带菜单的窗口

9.2.2 菜单条、菜单、菜单项

① 菜单条

JMenuBar负责创建菜单条

setJMenuBar(JMenuBar bar);该方法将菜单条添加到窗口的顶端。

菜单条JMenuBar类构造方法：**JMenuBar();**

JMenuBar Mbar = new JMenuBar()



9.2.2 菜单条、菜单、菜单项

② 菜单

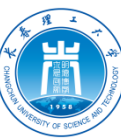
JMenu负责创建菜单：**JMenu()** ; **JMenu(String s)** ; **JMenu m=new JMenu();**

常用方法：

public void add(JMenuItem item) 向菜单增加由参数item指定的菜单项

public JMenuItem getItem(int n) 得到指定索引处的菜单选项。

public int getItemCount() 得到菜单选项的数目。





9.2.2 菜单条、菜单、菜单项

③ 菜单项

JMenuItem负责创建菜单项,**JMenuItem**类的主要方法有以下几种:

JMenuItem(String s) 构造有标题的菜单项。

JMenuItem(String text, Icon icon) 构造有标题和图标的菜单项

public void setAccelerator(KeyStroke keyStroke)为菜单项设置快捷键



9.2.2 菜单条、菜单、菜单项

```
JMenuBar jmb = new JMenuBar();
JMenu jm1 = new JMenu("文件"); JMenu jm2 = new JMenu("帮助");
JMenuItem jmi1 = new JMenuItem("新建"); JMenuItem jmi3 = new JMenuItem("退
jmi3.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, InputEvent.CTRL_MASK
jmi3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        System.exit(0); } });
JMenuItem jmi4 = new JMenuItem("帮助"); JMenuItem jmi5 = new JMenuItem("版
jmi5.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        JOptionPane.showConfirmDialog(null, "版本: 1.0.1"); } });
jmi5.setAccelerator(KeyStroke.getKeyStroke('A'));
jm1.add(jmi1); jm1.add(jmi2); jm1.addSeparator();
jm1.add(jmi3); jm2.add(jmi4); jm2.add(jmi5);
jmb.add(jm1); jmb.add(jm2); jf.setJMenuBar(jmb);
```



9.3 常用组件与布局

- 可以使用JComponent的子类JTextField创建各种组件。利用组件可以完成应用程序与用户的交互及事件处理等。
- 也可以在命令行窗口反编译组件即时查看组件所具有的属性及常用方法，例如：

C:\>javap javax.swing.JComponent

- 也可以查看类库帮助文档.例如下载Java类库帮助文档：
jdk-6-doc.zip。

9.3.1 常用组件

1. **文本框**：由JComponent子类JTextField创建文本框
2. **文本区**：JTextArea创建文本区
3. **按钮**：JButton类用来创建按钮
4. **标签**：JLabel类用来创建标签
5. **选择框**：JCheckBox类用来创建选择框
6. **单选按钮**：JRadioButton类用来创建单项选择框
7. **下拉列表**：JComboBox类用来创建下拉列表
8. **密码框**：JPasswordField创建密码框。



9.3.1 常用组件

1. 文本框：由JComponent的子类JTextField创建文本框。

➤ 构造函数：JTextField() ; JTextField(int columns) ;

➤ 常用方法：

☆ public String getText();

☆ public void setText(String t);





9.3.1 常用组件

2. 文本区：由JComponent的子类JTextArea创建文本区。

- 构造函数： JTextArea() ;JTextArea(int rows, int columns)
- 常用方法： public String getText();
- public void setText(String t);





9.3.1 常用组件

3. 按钮：由JComponent的子类JButton类用来创建按钮。

- 构造函数： JButton() ; JButton(String text) ;
- 常用方法：
- public void addActionListener(ActionListener l);





9.3.1 常用组件

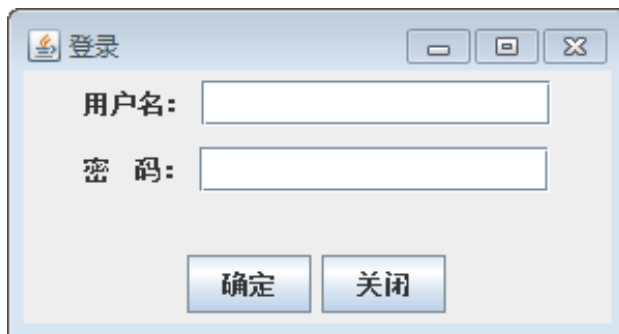
4. 标签：由JComponent的子类JLabel类用来创建标签。

- 构造函数：JLabel() ； JLabel(String text) ；
JLabel(Icon image) ；
- 常用方法：public String getText();
- public void setText(String t);



9.3.1 常用组件

练习



A screenshot of a standard Windows-style login dialog box. The title bar at the top is light blue and contains the text '登录' (Login) next to a small icon, followed by standard minimize, maximize, and close buttons. The main area has a light gray background. It contains two text input fields: the first is preceded by the label '用户名:' (Username) and the second by '密 码:' (Password). Below these fields are two buttons: '确定' (OK) on the left and '关闭' (Close) on the right.



9.3.1 常用组件

// 创建底层窗体

```
JFrame win = new JFrame("登录");  
win.setSize(300, 160);           // 设置窗体大小  
win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
win.setLocationRelativeTo(null);
```

// 创建中间层容器

```
JPanel pnlWin = new JPanel();  
JPanel pnlTop = new JPanel();  
JPanel pnlClient = new JPanel();  
JPanel pnlBottom = new JPanel();
```





9.3.1 常用组件

// 创建两个布局

```
BorderLayout bl = new BorderLayout();
```

```
FlowLayout fl = new FlowLayout();
```

// 将容器层层添加

```
win.setContentPane(pnlWin);
```

```
pnlWin.setLayout(bl);
```

```
pnlWin.add(pnlTop, BorderLayout.NORTH);
```

```
pnlWin.add(pnlBottom, BorderLayout.SOUTH);
```

```
pnlWin.add(pnlClient, BorderLayout.CENTER);
```

// 为不同容器设置不同的布局

```
pnlTop.setLayout(fl);
```

```
pnlClient.setLayout(fl);
```

```
pnlBottom.setLayout(fl);
```





9.3.1 常用组件

// 创建Label标签

```
JLabel lblName = new JLabel("用户名: ");
```

```
JLabel lblPwd = new JLabel("密 码: ");
```

// 创建文本框

```
JTextField txtName = new JTextField(15);
```

```
JPasswordField txtPwd = new JPasswordField(15);
```

```
txtPwd.setEchoChar('*');
```

// 创建按钮

```
JButton btnConfirm = new JButton("确定");
```

```
JButton btnClose = new JButton("关闭");
```





9.3.1 常用组件

```
pnlTop.add(lblName);
```

```
pnlTop.add(txtName);
```

```
pnlClient.add(lblPwd);
```

```
pnlClient.add(txtPwd);
```

```
pnlBottom.add(btnConfirm);
```

```
pnlBottom.add(btnClose);
```

```
//win.pack();
```

```
win.setVisible(true);
```

9.3.1 常用组件

5.选择框： 由JComponent的子类JCheckBox类用来创建选择框

- 构造函数： JCheckBox() ;JCheckBox(String text) ;
- 常用方法： public void addItemListener(ItemListener l)
- public void addActionListener(ActionListener l)

ItemListener接口中：

Object getItemSelectable()相当于getSource();

public int getStateChange();

- 被选中SELECTED/DESELECTED



9.3.1 常用组件

6. 单选按钮：由JComponent的子类JRadioButton类用来创建单项选择框。

➤ 构造函数： JRadioButton() ；

JRadioButton(String text) ；

➤ 常用方法： public void
addItemListener(ItemListener l)



9.3.1 常用组件

7. 下拉列表：由JComponent的子类JComboBox类用来创建下拉列表。

- 构造函数： JComboBox() ; JComboBox(Object[] items)
- 常用方法： `public void addItemListener(ItemListener l)`
- `public Object getSelectedItem();`
- `public int getSelectedIndex()`



9.3.1 常用组件

8.密码框：由JComponent的子类JPasswordField创建密码框。

- 构造函数：JPasswordField() ；
JPasswordField(int columns)
- 常用方法：public String getText();
- public void setText(String t);
- public void setEchoChar(char c)使用该方法重新设置回显字符。
- public char[] getPassword()方法可以返回实际的密码



9.3.1 常用组件

例子3(Example9_3.java , ComponentInWindow.java)包含有上面提到的常用组件。



图 9.4 常用组件



9.3.2 常用容器

□ JComponent 是 Container 的子类，因此 JComponent 子类创建的组件也都是容器。容器经常用来添加组件。**JFrame 是底层容器**，本节提到的容器被习惯地称做**中间容器**，中间容器必须被添加到底层容器中才能发挥作用。



9.3.2 常用容器

1. JPanel 面板:

构造方法: JPanel() ; 如: `JPanel p=new JPanel();`

常用方法: `public void add();`

使用JPanel创建面板, 再向这个面板添加组件, 然后把这个面板添加到其它容器中。JPanel面板的默认布局是FlowLayout布局。

9.3.2 常用容器

2. JTabbedPane选项卡窗格

可以使用 JTabbedPane 容器作为中间容器。当用户向 JTabbedPane 容器添加一个组件时，JTabbedPane 容器就会自动为该组件指定对应的一个选项卡，即让一个**选项卡对应一个组件**。各个选项卡对应的组件层叠式放入 JTabbedPane 容器，*当用户单击选项卡时，JTabbedPane 容器将显示该选项卡对应的组件。*选项卡默认地在 JTabbedPane 容器的顶部，从左向右依次排列。JTabbedPane 容器可以使用：
add(String text, Component c);
方法将组件 c 添加到 JTabbedPane 容器中，并指定和组件 c 对应的选项卡的文本提示是 text。

9.3.2 常用容器

3. 滚动窗格JScrollPane

滚动窗格只可以添加一个组件，可以把一个组件放到一个滚动窗格中，然后通过滚动条来观看该组件。JTextArea不自带滚动条，因此就需要把文本区放到一个滚动窗格中。例如，

```
JScrollPane scroll = new JScrollPane(new JTextArea());
```

9.3.2 常用容器

4. 拆分窗格JSplitPane

拆分窗格就是被分成两部分的容器。拆分窗格有两种类型：水平拆分和垂直拆分。拆分线可以垂直移动。

JSplitPane的两个常用的构造方法：

JSplitPane(int a, Component b, Component c)

参数a取JSplitPane的静态常量 HORIZONTAL_SPLIT或 VERTICAL_SPLIT，以决定是水平还是垂直拆分。后两个参数决定要放置的组件。

JSplitPane(int a, boolean b, Component c, Component d)

参数a取JSplitPane的静态常量HORIZONTAL_SPLIT或 VERTICAL_SPLIT，以决定是水平还是垂直拆分，参数b决定当拆分线移动时，组件是否连续变化（true是连续）。

9.3.2 常用容器

5. JLayeredPane分层窗格

如果添加到容器中的组件经常需要处理重叠问题，就可以考虑将组件添加到分层窗格。分层窗格分成5个层，分层窗格使用

`add(Jcomponent com, int layer);`

添加组件com，并指定com所在的层，其中参数layer取值JLayeredPane类中的类常量：

DEFAULT_LAYER、PALETTE_LAYER、MODAL_LAYER、POPUP_LAYER、DRAG_LAYER。

添加到同一层上的组件，如果发生重叠，后添加的会遮挡先添加的组件。分层窗格调用

`public void setLayer(Component c,int layer)`可以重新设置组件c所在的层，调用

`public int getLayer(Component c)`可以获取组件c所在的层数。

9.4 处理事件

- 学习组件除了要熟悉组件的属性和功能外，一个更重要的方面是学习怎样处理组件上发生的界面事件。当用户在文本框中键入文本后按回车键、单击按钮、在一个下拉式列表中选择一个条目等操作时，都发生界面事件。
- 程序有时需对发生的事件作出反应，来实现特定的任务，例如，用户单击一个名字叫“确定”或名字叫“取消”的按钮，程序可能需要作出不同的处理。

9.4.1 事件处理模式

1. **事件源**：能够产生事件的对象都可以成为事件源
2. **监视器**：事件源通过调用相应的方法将某个对象注册为自己的监视器。对于文本框，这个方法是：
`addActionListener(监视器);`

java语言中监视器都是使用**接口**来实现的。事件源注册监视器之后，相应的操作就会导致相应的事件的发生，并通知监视器，监视器就会作出相应的处理

3. **处理事件的接口**：监视器负责处理事件源发生的事件。监视器是一个对象，为了处理事件源发生的事件，监视器这个对象会**自动调用接口**中一个方法来处理事件

Java规定：为了让监视器这个对象能对事件源发生的事件进行处理，创建该监视器对象的**类必须声明实现相应的接口**，那么当事件源发生事件时，监视器就自动调用被类重写的某个接口方法(如图9.7)

9.4.1 事件处理模式

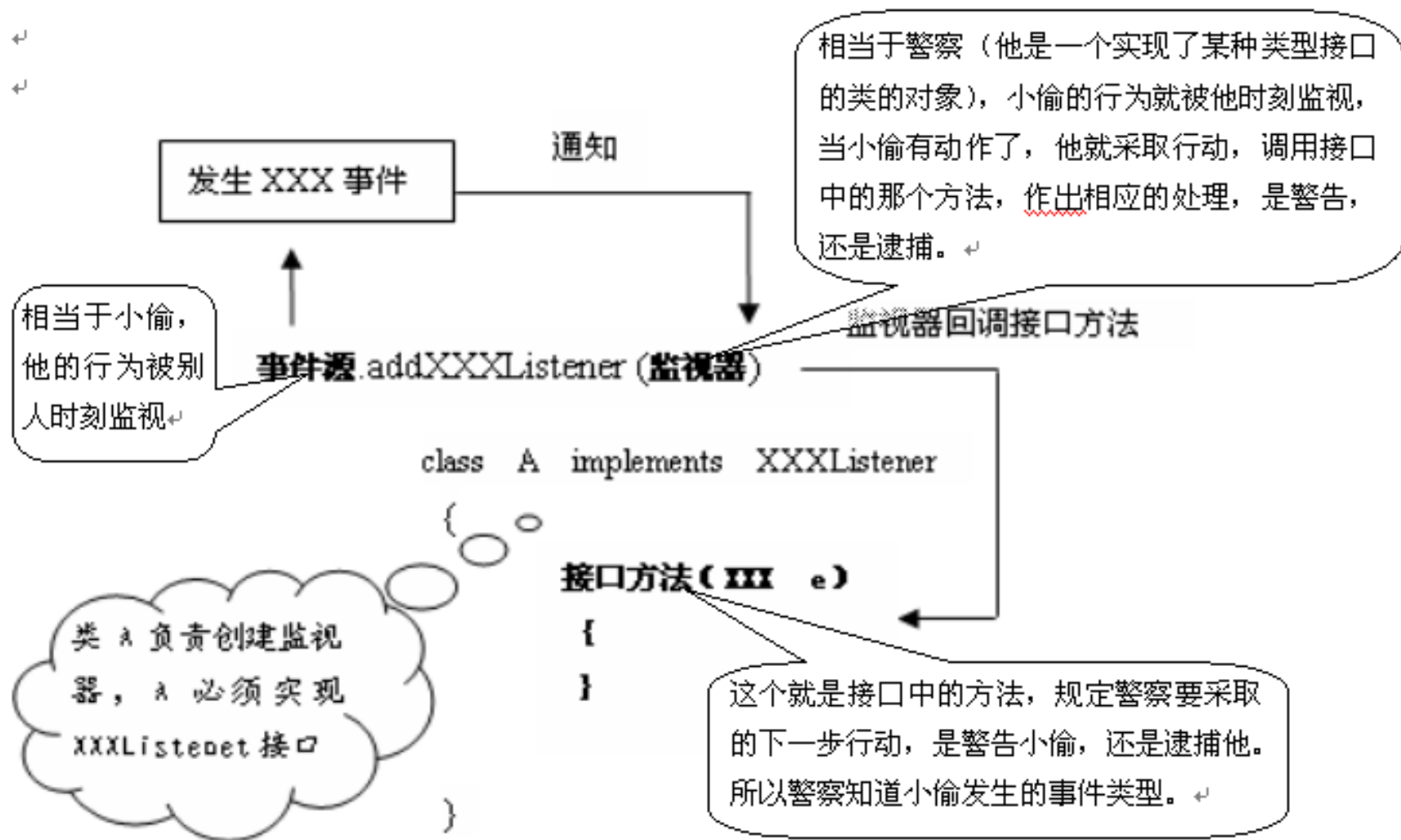


图 处理事件示意图

9.4.2 ActionEvent事件

1.ActionEvent事件源：

文本框、按钮、菜单项、密码框和单选按钮都可以触发ActionEvent事件，即都可以成为ActionEvent事件的事件源

2.注册监视器: ActionListener接口作为监视器

事件源.addActionListener(ActionListener listen)

将实现ActionListener接口的类的实例注册为事件源的监视器。



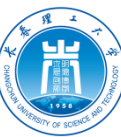
9.4.2 ActionEvent事件

3.ActionListener接口：重写接口中的方法作为事件处理。

ActionListener接口在java.awt.event包中,该接口中只有一个方法:

public void actionPerformed(ActionEvent e)

事件源触发 ActionEvent 事件后，监视器将发现触发的 ActionEvent事件，然后调用接口中的这个方法对发生的事件作出处理。ActionEvent类事先创建的事件对象就会传递给该方法的参数e





9.4.2 ActionEvent事件

4.ActionEvent类中的方法:下面的方法能够获取发生动作的事件源。

public Object getSource()

调用该方法可以获取发生ActionEvent事件的事件源对象的引用

public String getActionCommand()

调用该方法可以获取发生ActionEvent事件时，和该事件相关的一个命令字符串。



事件处理步骤1：确定事件源及监听器类型



- 由于我们想要处理按钮的点击事件，因此，按钮便是事件源；
- 监听器类型是**ActionListener**。

事件处理步骤2：实现监听器接口

- 编写类来实现监听器接口，并重写其抽象方法，如：

```
class MyListener implements ActionListener {  
    public void actionPerformed(ActionEvent ae) {  
        .....  
    }  
}
```

- 事实上，我们重写的这个抽象方法就是事件处理函数。也就是说，当事件发生时，这个方法将自动调用，其中的代码将被执行；
- 但是，为了方便成员间的相互访问，我们一般采用**内部类**的方式来实现监听器



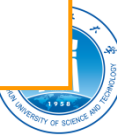
事件处理步骤2：实现监听器接口（代码）

```
public class EventDemo {  
    /*采用内部类的方式实现监听器接口*/  
    private class MyListener implements ActionListener {  
        //实现接口中的抽象方法，事件发生时，自动调用此方法*/  
        public void actionPerformed(ActionEvent ae) {  
            lblMsg.setText("我被点击了! "); //设置标签中的文本  
        }  
    }  
  
    public static void main(String[] args) {  
        .....  
        btnClick.addActionListener(new MyListener());  
        .....  
    }  
}
```



事件处理步骤2：实现监听器接口（代码）

```
public class EventDemo extends JFrame {  
    private JLabel lblMsg;  
    private JButton btnClick;  
    public EventDemo() { //构造方法，代码略  
        ..... }  
    /*采用内部类的方式实现监听器接口*/  
    private class MyListener implements ActionListener {  
        /*实现接口中的抽象方法，事件发生时，将自动调用此方法*/  
        public void actionPerformed(ActionEvent ae) {  
            lblMsg.setText("我被点击了！"); //设置标签中的文本  
        }  
    }  
    public static void main(String[] args) {  
        new EventDemo();  
    } }  
}
```



事件处理步骤3：事件源注册监听器

□ 最后，我们要将事件源注册到监听器，也就是说，必须委派监听器去监听事件源所发生的事件；

□ 每种类型的事件都有其自己的注册方法，一般形式为：

`void addXxxListener(XxxListener listener);`

这里的Xxx指代具体的事件类型，而listener则是相应类型的监听器实例；

□ 一般会采用如下形式将事件源注册到监听器：

事件源.addXxxListener(监听器实例);

`btnClick.addActionListener(new MyListener());`



事件处理步骤3：事件源注册监听器

```
public class EventDemo extends JFrame {  
    private JLabel lblMsg;  
    private JButton btnClick;  
    public EventDemo() { //构造方法，代码略  
        .....  
        btnClick.addActionListener(new MyListener()); //将事件源注册到监听器  
        .....  
    }  
    /*内部类实现监听器接口*/  
    private class MyListener implements ActionListener {  
        public void actionPerformed(ActionEvent ae) { //实现事件处理函数  
            lblMsg.setText("我被点击了! ");  
        }  
    }  
    public static void main(String[] args) {  
        new EventDemo();  
    }  
}
```



9.4.2 ActionEvent事件

例子 6 ([Example9_6.java](#) , [WindowActionEvent.java](#) , [ReaderListen.java](#))处理文本框上触发的ActionEvent事件。在文本框text中输入字符串回车，监视器负责计算字符串的长度，并在命令行窗口显示字符串的长度。例子6程序运行效果如图9.8和9.9。

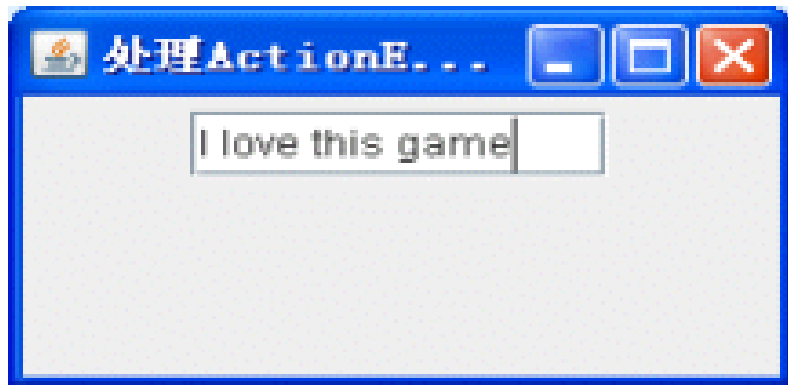


图 9.8 事件源触发事件

```
we:2  
你好:2  
how are you:11  
今天是国庆日:6  
今天是国庆日:6  
I love this game:16
```

图 9.9 监视器负责处理事件



```
txt.addActionListener(new MyRead());
```

```
public class MyRead implements ActionListener {  
    public void actionPerformed(ActionEvent arg0) {  
        String str = arg0.getActionCommand();  
        System.out.println(str + " : " + str.length());  
    }  
}
```



9.4.2 ActionEvent事件

例子 7 ([Example9_7.java](#) , [WindowActionEvent.java](#) , [MyCommandListener.java](#) , [PoliceListen.java](#))

例子 7 (中的 监视器 : PoliceListen 与 例子 6 中的 ReaderListen 略有不同, PoliceListen 类实现了 ActionListener 接口的子接口 MyCommandListener (我们自己写的一个接口) 。

当用户在文本框中输入字符串回车或单击按钮 (按钮可以触发 ActionEvent 事件, 当按钮获得监视器之后, 如果激活按钮, 比如用鼠标单击按钮或按钮获得焦点时按下空格键, 就可以触发 ActionEvent 事件), PoliceListen 监视器将字符串的长度显示在一个文本区中。运行效果如图 9.10。



图 9.10 处理 ActionEvent 事件

```
public class MyListener implements MyCommand {  
    JTextField txtInput; JTextArea txtShow;  
    public void actionPerformed(ActionEvent arg0) {  
        String str = txtInput.getText();  
        txtShow.append(str + "的长度：" + str.length() + "\n");  
    }  
    public void setJTextField(JTextField tf) { this.txtInput = tf; }  
    public void setJTextArea(JTextArea ta) { this.txtShow = ta; } }
```

```
public interface MyCommand extends ActionListener {  
    public void setJTextField(JTextField tf);  
    public void setJTextArea(JTextArea ta); }
```

```
MyCommand ml = new MyListener();  
ml.setJTextArea(ta); ml.setJTextField(txt);  
txt.addActionListener(ml); btn.addActionListener(ml);
```

9.4.3 ItemEvent事件

1. ItemEvent事件源 :选择框、下拉列表都可以触发ItemEvent事件。

2. 注册监视器:ItemListener接口作为监视器

事件源.addItemListener(ItemListener listen)

将实现ItemListener接口的类的实例注册为事件源的监视器。

3. ItemListener接口:重写方法作为事件处理

ItemListener接口在java.awt.event包中,该接口中只有一个方法

public void itemStateChanged(ItemEvent e)

事件源触发ItemEvent事件后, 监视器将发现触发的ItemEvent事件, 然后调用接口中的itemStateChanged(ItemEvent e)方法对发生的事件作出处理。ItemEvent类事先创建的事件对象就会传递给该方法的参数e。

4. ItemEvent类中的方法 :

getSource()方法返回发生Itemevent事件的事件源外

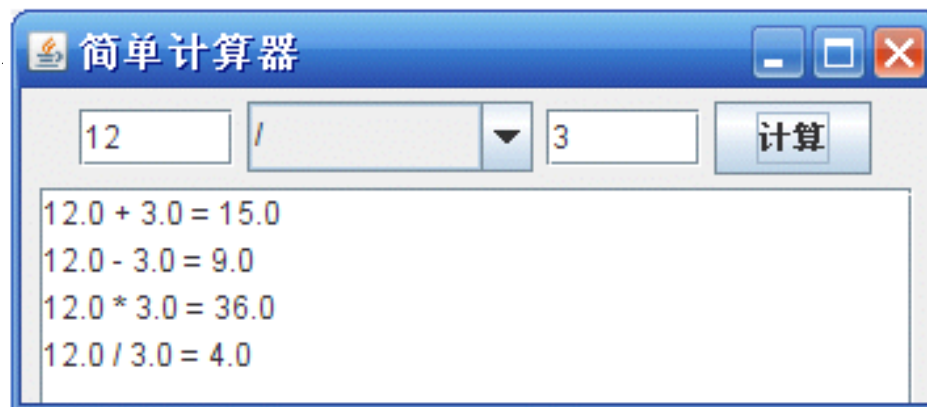
getItemSelectable()方法返回发生Itemevent事件的事件源。

9.4.3 ItemEvent事件

例子8([Example9_8.java](#), [WindowOperation.java](#),
[OperatorListener.java](#), [ComputerListener.java](#))

例子8(是简单的**计算器** (程序运行效果如图9.11) , 实现如下功能:

- (1)用户在窗口 (WindowOperation类负责创建) 中的**两个文本框**中输入参与运算的两个操作数。
- (2)用户**在下拉列选择**运算符**触发ItemEvent事件**, ItemEvent事件的监视器operator (OperatorListener类负责创建) **获得运算符**, 并将运算符传递给ActionEvent事件的监视器computer
- (3)用户单击按钮**触发ActionEvent事件**, 监视器computer (ComputerrListener类负责创



9.4.4 DocumentEvent事件

1. DocumentEvent 事件源：文本区所维护的文档能触发 DocumentEvent事件

2. 注册监视器：DocumentListener作为监视器

`addDocumentListener(DocumentListener listen)`

将实现DocumentListener接口的类的实例注册为事件源的监视器

3. DocumentListener接口：重写接口中的方法作为事件处理

DocumentListener接口在javax.swing.event包中，该接口中有三个方法：

`public void changedUpdate(DocumentEvent e)`

`public void removeUpdate(DocumentEvent e)`

`public void insertUpdate(DocumentEvent e)`

事件源触发DocumentEvent事件后，监视器将发现触发的DocumentEvent事件，然后调用接口中的相应方法对发生的事件作出处理。

9.4.4 DocumentEvent事件

例子9

(Example9 9.java,

WindowDocument.java,

TextListener.java,

HandleListener.java) 将用户
在一个文本区输入的单词按字典
序排序后放入另一个文本区。

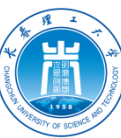


9.12 处理 DocumentEvent 事件。

```
JTextArea.getDocument().addDocumentListener(ml);
```



```
public class MyListener implements DocumentListener {  
    JTextArea ta1, ta2;  
    public void SetInputText(JTextArea ta) { ta1 = ta; }  
    public void SetShowText(JTextArea ta) { ta2 = ta; }  
    public void changedUpdate(DocumentEvent arg0) {  
        String str = ta1.getText();  
        String regex = "[\\s\\d\\p{Punct}]+";  
        String words[] = str.split(regex);  
        Arrays.sort(words);  
        ta2.setText(null);  
        for (String s : words) {  
            ta2.append(s + ","); } }  
}
```



9.4.5 MouseEvent事件_1

任何组件上都可以发生鼠标事件，如：鼠标进入组件、退出组件、在组件上方单击鼠标、拖动鼠标等都触发鼠标事件，即导致MouseEvent类自动创建一个事件对象。

1. 使用MouseListener接口可处理以下5种操作触发的鼠标事件

在事件源上按下鼠标键、在事件源上释放鼠标键、在事件源上击鼠标键、鼠标进入事件源、鼠标退出事件源。

➤注册监视器:MouseListener接口作为监视器

事件源.addMouseListener(MouseListener listener)。



9.4.5 MouseEvent事件_1

➤ **MouseListener接口**中有如下方法：重写方法作为事件源的处理

mousePressed(MouseEvent) 负责处理在组件上按下鼠标键触发的鼠标事件

mouseReleased(MouseEvent) 负责处理在组件上释放鼠标键触发的鼠标事件

mouseEntered(MouseEvent) 负责处理鼠标进入组件触发的鼠标事件

mouseExited(MouseEvent) 负责处理鼠标离开组件触发的鼠标事件

mouseClicked(MouseEvent) 负责处理在组件上单击鼠标键触发的鼠标事件

➤ **MouseEvent** 中有下列几个重要的方法：

getX() 获取鼠标指针在事件源坐标系中的x-坐标。

getY() 获取鼠标指针在事件源坐标系中的y-坐标。

getModifiers() 获取鼠标的左键或右键。

getClickCount() 获取鼠标被单击的次数。

getSource() 获取发生鼠标事件的事件源。



9.4.5 MouseEvent事件_1

例子10 Example9_10.java

WindowMouse.java

MousePolice.java

分别监视按钮、文本框和窗口上的鼠标事件，当发生鼠标事件时，获取鼠标指针的坐标值，注意，事件源的坐标系的左上角是原点。

9.4.5 MouseEvent事件_3

2. 使用**MouseMotion**Listener接口可以处理以下两种操作触发的鼠标事件，在事件源上拖动鼠标、在事件源上移动鼠标。

➤ 事件源注册监视器的方法是

addMouseMotionListener(MouseMotionListener listener)。

➤ **MouseMotionListener**接口中有如下方法：

mouseDragged(MouseEvent)负责处理拖动鼠标触发的鼠标事件

mouseMoved(MouseEvent)负责处理移动鼠标触发的鼠标事件

例子11 Example9_11.java

WindowMove.java

LP.java 使用坐标变换来实现组件的拖动

9.4.6 焦点事件

1. **焦点事件源** :组件可以触发焦点事件。

2. **注册监视器**:组件可以使用

`addFocusListener(FocusListener listener)`

注册焦点事件监视器。

3. **FocusListener接口** :

创建监视器的类必须要实现FocusListener接口，该接口有两个方法：

`public void focusGained(FocusEvent e)`

`public void focusLost(FocusEvent e)`

当发生FocusEvent事件时，监视器调用类实现的接口中的相应方法。

4. **组件也可调用**

`public boolean requestFocusInWindow()`

方法可以获得输入焦点。

9.4.7 键盘事件

一个组件处于激活状态时，敲击键盘上一个键就导致这个组件触发键盘事件

1.某个组件使用**addKeyListener**方法注册监视器

2.**接口KeyListener**中有如下方法：

`public void keyPressed(KeyEvent e),`

`public void keyTyped(KeyEvent e),`

`public void KeyReleased(KeyEvent e)`

3.**相关方法：**

`public int getKeyCode()`判断哪个键被按下、敲击或释放,返回一个键码值。

`getKeyChar()`判断哪个键被按下、敲击或释放，返回键上的字符。

9.4.7 键盘事件

例子12 Example9_12.java

Win.java

Police.java通过处理键盘事件来实现软件序列号的输入。



图 9.13 输入序列号

9.4.8 窗口事件

□ JFrame及子类创建的窗口可以调用

setDefaultCloseOperation(int operation);

- 方法设置窗口的关闭方式（如前面各个例子所示）。
- 但是**setDefaultCloseOperation**方式可能不能满足程序的需要，比如，用户单击窗口上的关闭图标时，可能程序需要提示用户是否需要保存窗口中的有关数据到磁盘等。



9.4.8 窗口事件

1.WindowListener接口.

(1) `public void windowActivated(WindowEvent e)` 当窗口从非激活状态到激活时，窗口的监视器调用该方法。

(2) `public void windowDeactivated(WindowEvent e)` 当窗口激活状态到非激活状态时，窗口的监视器调用该方法。

(3) `public void windowClosing(WindowEvent e)` 当窗口正在被关闭时，窗口的监视器调用该方法。

(4) `public void windowClosed(WindowEvent e)` 当窗口关闭后，窗口的监视器调用该方法。



9.4.8 窗口事件

1.WindowListener接口.

(5) `public void windowIconified(WindowEvent e)` 当窗口图标化时，窗口的监视器调用该方法。

(6) `public void windowDeiconified(WindowEvent e)` 当窗口撤消图标化时，窗口的监视器调用该方法。

(7) `public void windowOpened(WindowEvent e)` 当窗口打开时，窗口的监视器调用该方法。


9.4.8 窗口事件

2. WindowAdapter适配器

□ 每个具有不止一个方法的AWT监听器接口都有一个实现了它的所有方法，但却不做任何工作的适配器类。

- ComponentAdapter
- ContainerAdapter
- FocusAdapter
- KeyAdapter
- MouseAdapter
- MouseMotionAdapter
- WindowAdapter

例子13(Example9_13.java)
使用适配器做监视器，只处理窗口关闭事件，因此只需重写windowClosing方法即可。



9.4.8 窗口事件

2. WindowAdapter适配器

□ **适配器**可以代替接口来处理事件，当Java提供处理事件的接口中多于一个方法时，Java相应地就提供一个适配器类，比如WindowAdapter类。**适配器已经实现了相应的接口**，例如WindowAdapter类实现了WindowListener接口。因此，可以使用WindowAdapte的子类创建的对象做监视器，在子类中重写所需要的接口方法即可



9.4.9 匿名类实例或窗口做监视器

匿名类的方便之处是匿名类的外嵌类的成员变量在匿名类中仍然有效，当发生事件时，监视器就比较容易操作事件源所在的外嵌类中的成员。当事件的处理比较简单、系统也不复杂时，使用匿名类做监视器是一个不错的选择。

让事件源所在的类的实例作为监视器，能让事件的处理比较方便，这是因为，监视器可以方便的操作事件源所在的类中的其他成员。当事件的处理比较简单，系统也不复杂时，让事件源所在的类的实例作为监视器是一个不错

例子14([Example9_14.java](#))是一个猜数字



图 9.14 猜数字

9.4.10 事件总结

1.授权模式

Java的事件处理是基于授权模式，即事件源调用方法将某个对象注册为自己的监视器。

2.接口回调

`addXXXListener(XXXListener listener)`方法中的参数是一个接口，`listener`可以引用任何实现了该接口的类所创建的对象，当事件源发生事件时，接口`listener`立刻回调被类实现的接口中的某个方法。

3.方法绑定

当事件源触发事件发生后，监视器准确知道去调用哪个方法。

4.保持松耦合

当事件源触发事件发生后，系统知道某个方法会被执行，但无须关心到底是哪个对象去调用了这个方法，因为任何实现接口的类的实例(做为监视器)都可以调用这个方法来处理事件。

9.5 使用MVC结构

- 模型-视图-控制器 (**Model-View-Controller**)，简称为MVC
- MVC是一种先进的设计结构，其目的是以会话形式提供方便的GUI支持
- MVC是一种通过三个不同部分构造一个软件或组件的理想办法：
 - **模型(model)** 用于存储数据的对象。
 - **视图(view)** 为模型提供数据显示的对象。
 - **控制器(controller)**处理用户的交互操作，对于用户的操作作出响应，让模型和视图进行必要的交互，即通过视图修改、获取模型中的数据；当模型中的数据变化时，让视图更新显示。

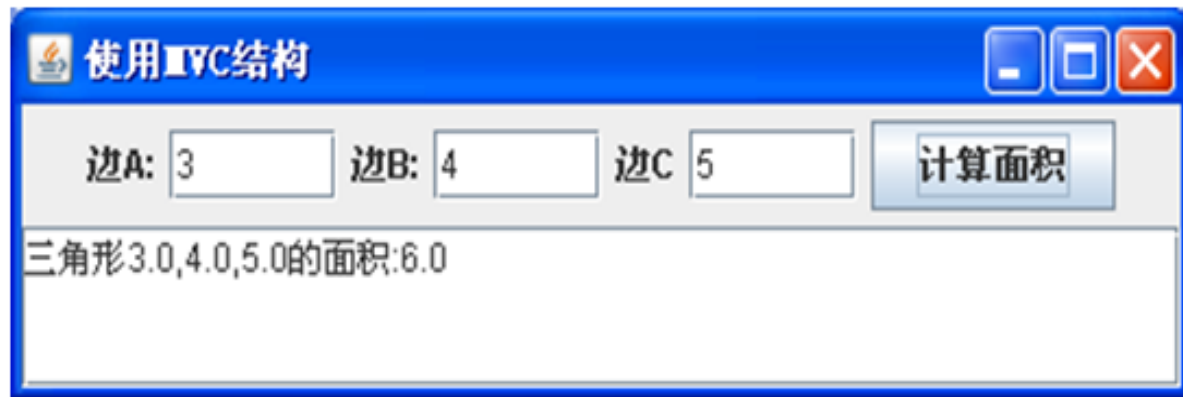
9.5 使用MVC结构

例子15([Example9_15.java](#), [WindowTriangle.java](#), [Triangle.java](#))

首先编写一个封装三角形的类（模型）

然后再编写一个窗口。要求窗口使用三文本框和一个文本区为三角形对象中的数据提供视图，其中三个文本框用来显示和更新三角形对象的三个边的长度；文本区对象用来显示三角形的面积。

窗口中有一个按钮（控制器），用户单击该按钮后，程序用3个文本框中的数据分别作为三角形的三个边的长度，并将计算出的三角形的面积显示在文本区中。程序运行效果如图9.15。



9.6 对话框

- **JDialog类和JFrame都是Window的子类，二者的实例都是底层容器。**
- **JDialog类创建的对话框必须要依赖于某个窗口。**
- **对话框分为无模式和有模式两种。**
 - **有模式的对话框：**当这个对话框处于激活状态时，只让程序响应对话框内部的事件，而且将堵塞其它线程的执行，用户不能再激活对话框所在程序中的其它窗口，直到该对话框消失不可见。
 - **无模式对话框：**当这个对话框处于激活状态时，能再激活其它窗口，也不堵塞其它线程的执行。

9.6.1 消息对话框

消息对话框是有模式对话框，进行一个重要的操作动作之前，最好能弹出一个消息对话框。可以用 `javax.swing` 包中的 `JOptionPane` 类的静态方法：

```
public static void showMessageDialog(  
    Component parentComponent,  
    String message,  
    String title,  
    int messageType)
```

创建一个消息对话框。

例子 16 ([Example9_16.java](#) , [WindowMess.java](#))，要求用户在文本框中只能输入英文字母，当输入非英文字符时，弹出消息对话框。



图 9.16 消息对话框

9.6.2 输入对话框

输入对话框含有供用户输入文本的文本框、一个确认和取消按钮，是有模式对话框。

可以用javax.swing包中的JOptionPane类的静态方法：

```
public static String showInputDialog(Component parentComponent,  
                                     Object message,  
                                     String title,  
                                     int messageType)
```

创建一个输入对话框。

例 子 1 7 ([Example11_17.java](#) , [WindowInput.java](#)), 用户在单击按钮弹出输入对话框，用户在输入对话框中输入若干个数字



图 9.17 输入对话框

9.6.3 确认对话框

确认对话框是有模式对话框。

可以用javax.swing包中的JOptionPane类的静态方法：

```
public static int showConfirmDialog(Component parentComponent,  
                                   Object message,  
                                   String title,  
                                   int optionType)
```

得到一个确认对话框。

例子18([Example9_18.java](#) , [WindowEnter.java](#)), 用户在文本框中输入帐户名称, 按回车后, 将弹出一个确认对话框。

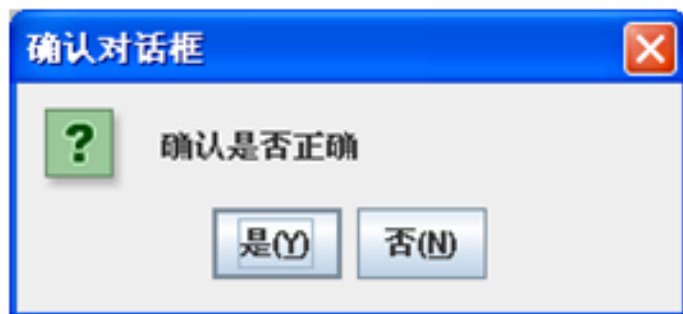


图 9.18 确认对话框

9.6.4 颜色对话框

可以用javax.swing包中的JColorChooser类的静态方法：
`public static Color showDialog (Component component,
String title,
Color initialColor)`

创建一个有模式的颜色对话框。

例子19([Example9_19.java](#) , [WindowColor.java](#)), 当用户单击按钮时, 弹出一个颜色对话框



图 9.19 颜色对话框

9.6.5 自定义对话框

- ❑ 创建对话框与创建窗口类似，通过建立JDialog的子类来建立一个对话框类，然后这个类的一个实例，即这个子类创建的一个对象，
- ❑ 构造对话框的2个常用构造方法
 - **JDialog()** 构造一个无标题的初始不可见的对话框，对话框依赖一个默认的不可见的窗口，该窗口由Java运行环境提供。
 - **JDialog(JFrame owner)** 构造一个无标题的初始不可见的无模式的对话框，owner是对话框所依赖的窗口，如果owner取null，对话框依赖一个默认的不可见的窗口，该窗口由Java运行环境提供。

例 子 2 0 ([Example9_20.java](#) , [MyWindow.java](#) , [MyDialog.java](#))



图 9.20 自定义对话框



```
public class MyDialog extends JDialog {  
    JTextField txtTitle; String title;  
    MyDialog(JFrame f, String s) {  
        super(f,s); txtTitle = new JTextField(10);  
        txtTitle.addActionListener(new ActionListener(){  
            public void actionPerformed(ActionEvent arg0) {  
                title = txtTitle.getText(); setVisible(false); } } );  
        setLayout(new FlowLayout());  
        add(new JLabel("输入新标题")); add(txtTitle);  
        setBounds(60, 60, 100, 100);  
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE); }  
  
    public String getTitle() { return title; } }
```



9.7 树组件与表格组件

9.7.1 树组件

1. DefaultMutableTreeNode节点

DefaultMutableTreeNode类的两个常用的构造方法是：

DefaultMutableTreeNode(Object userObject)

DefaultMutableTreeNode(Object userObject,boolean allowChildren)

2. 树上的TreeSelectionEvent事件使用

addTreeSelectionListener(TreeSelectionListener listener)

方法获得一个监视器。

例子21 Example9_21.java,

TreeWin.java, Goods.java

节点中存放的对象由Goods类（描述商品）创建，当用户选中节点时，窗口中的文本区显示节点中存放的对象的有关信息。



图 9.21 左侧是树组件

9.7.2 表格组件

表格组件以行和列的形式显示数据，允许对表格中的数据进行编辑。表格的模型功能强大、灵活并易于执行。表格是最复杂的组件，对于初学者，这里只介绍默认的表格模型。

JTable有7个构造方法，这里介绍常用的三个。

JTable() 创建默认的表格模型。

JTable(int a,int b) 创建a行,b列的默认模型表格

JTable (Object data[][],Object columnName[]) 创建默认表格模型对象，并且显示由data指定的二维数组的值，其列名由数组columnName指定。通过对表格中的数据进行编辑，可以修改表格中二维数组data中对应的数据。在表格中输入或修改数据后，需按回车或用鼠标单击表格的单元格确定所输入或修改的结果。当表格需要刷新

例子22([Example9_22.java](#))

是一个成绩单录入程序，客户通过一个表格的单元格输入学生的数学和英语成绩。



姓名	英语成绩	数学成绩	总成绩
张三	77	66	143.0
李四	88	100	188.0
孙强	56	99	155.0
姓名	0	0	0.0

计算每人总成绩

图 9.22 表格

9.8 按钮绑定到键盘

按钮绑定到键盘的步骤如下：

(1) 获取输入映射。 按钮首先调用

```
public final InputMap getInputMap(int condition)
```

方法返回一个InputMap对象。

(2) 绑定按钮的键盘操作。 步骤（1）返回的输入映射首先调用方法

```
public void put(KeyStroke keyStroke, Object actionMapKey)
```

将敲击键盘上的某键指定为按钮的键盘操作。

(3) 为按钮的键盘操作指定监视器。 按钮调用方法

```
public final ActionMap getActionMap()
```

返回一个ActionMap对象：

```
ActionMap actionmap = button.getActionMap();
```

然后，该对象actionmap调用方法：

```
public void put(Object key, Action action)
```

为按钮的键盘操作指定监视器

9.8 按钮绑定到键盘

例子23 Example9_23.java

BindButtonWindow.java

用鼠标单击按钮或敲击键盘的'A'键，程序将移动按钮

9.9 打印组件

步骤如下：

1. 获取Toolkit对象

让组件调用getToolkit()方法返回系统提供的Toolkit对象。

2. 获得PrintJob对象

Toolkit对象调用getPrintJob()方法返回一个PrintJob对象。

3. 获取Graphics对象

PrintJob对象使用getGraphics()方法获得一个Graphics对象。

4. 打印组件

步骤3获取的Graphics对象是g，组件调用

paintAll(g)

将打印出该组件及其子组件。如果

paint(g)

将打印出该组件本身，但不打印子组件。 [例子24](#)

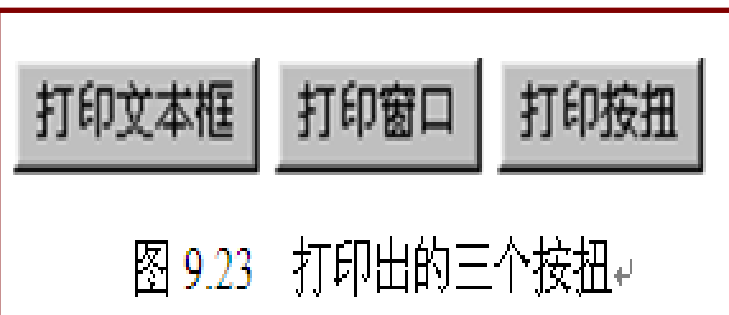


图 9.23 打印出的三个按钮

9. 10 发布GUI程序

- 可以使用jar.exe把一些文件压缩成一个JAR文件，来发布我们的应用程序。
- 生成一个Jar文件的步骤如下：
 1. 首先用文本编辑器（比如Windows下的记事本）编写一个清单件，
扩展名为.mf，如Mymoon.mf。
 2. 生成JAR文件
- 如果目录test下的字节码文件刚好是应用程序需要的全部字节码文件，也可以如下生成JAR文件：
`D:\test\jar cfm Tom.jar Mymoon.mf *.class`
- 可以将Tom.jar文件复制到任何一个安装了java运行环境的计算机上，只要用鼠标双击该文件就可以运行该java应用程序了。

9.11 应用举例

例子 25 华容道 ([Example9_25.java](#) , [Person.java](#) , [Hua_Rong_Road.java](#))通过键盘和鼠标事件来实现曹操、关羽等人物的移动, 如图9.24所示。



图 9.24 华容道