Swing是目前**[Java](http://lib.csdn.net/base/java" \t "_blank" \o "Java 知识库)**中不可缺少的窗口工具组，是用户建立图形化用户界面（GUI）程序的强大工具。Java Swing组件自动产生各种事件来响应用户行为。如当用户点击按钮或选择菜单项目时，Swing组件会产生一个ActionEvent。Swing组件会产生许多事件，如ActionEvents,ChangeEvents,ItemEvents等，来响应用户的鼠标点击行为，列表框中值的改变，计时器的开始计时等行为。在Java Swing编程中，通过**注册监听器**，我们可以监听事件源产生的事件，从而在事件处理程序中处理我们所需要处理的用户行为。

Java Swing中处理各组件事件的一般步骤是：

1．  新建一个组件（如JButton）。

2．  将该组件添加到相应的面板（如JPanel）。

3．  注册监听器以监听事件源产生的事件（如通过ActionListener来响应用户点击按钮）。

4．  定义处理事件的方法（如在ActionListener中的actionPerformed中定义相应方法）。

以上步骤我们可以用多种方法实现。但人们通常用两种方法。第一种方法是只利用一个监听器以及多个if语句来决定是哪个组件产生的事件；第二种方法是使用多个内部类来响应不同组件产生的各种事件，其具体实现又细分为两种方式：一种是匿名内部类，一种是一般内部类。

为了说明如何使用上述三种方法实现事件的处理方法，我们建立一个简单的应用程序。该程序界面有两个按钮，当用户点击相应的按钮，就会弹出一个对话框显示相应的内容。通过这个简单的程序，你可以了解Swing中事情处理的机制，从而实现更多、更复杂的用户界面程序。

首先，我们利用单个监听器来实现该程序。我们定义一个名为Simple1的类来包括所有代码。所有的用户行为（如点击按钮）由一个监听器SimpleListenner中的actionPerformed方法来处理。以下是实现代码：

**package** org.leno.swing.demo2;

/\*

 \* Simple1.java - 处理事件的第一种方法

 \*    在这个例子中，利用一个ActionListener来监听事件源产生的事件

 \*    用一些if语句来决定是哪个事件源

 \*/

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** Simple1 {

**private** **static** JFrame *frame*; // 定义为静态变量以便main使用

**private** **static** JPanel *myPanel*; // 该面板用来放置按钮组件

**private** JButton button1; // 这里定义按钮组件

**private** JButton button2; // 以便让ActionListener使用

**public** Simple1() // 构造器, 建立图形界面

       {

              // 新建面板

*myPanel* = **new** JPanel();

              // 新建按钮

              button1 = **new** JButton("按钮1"); // 新建按钮1

              button2 = **new** JButton("按钮2");

              SimpleListener ourListener = **new** SimpleListener();

              // 建立一个actionlistener让两个按钮共享

              button1.addActionListener(ourListener);

              button2.addActionListener(ourListener);

*myPanel*.add(button1); // 添加按钮到面板

*myPanel*.add(button2);

       }

**private** **class** SimpleListener **implements** ActionListener {

              /\*

               \* 利用该内部类来监听所有事件源产生的事件 便于处理事件代码模块化

               \*/

**public** **void** actionPerformed(ActionEvent e) {

                     // 利用getActionCommand获得按钮名称

                     // 也可以利用getSource()来实现

                     // if (e.getSource() ==button1)

                     String buttonName = e.getActionCommand();

**if** (buttonName.equals("按钮1"))

                            JOptionPane.*showMessageDialog*(*frame*, "按钮1 被点击");

**else** **if** (buttonName.equals("按钮2"))

                            JOptionPane.*showMessageDialog*(*frame*, "按钮2 被点击");

**else**

                            JOptionPane.*showMessageDialog*(*frame*, "Unknown event");

              }

       }

**public** **static** **void** main(String s[]) {

              Simple1 gui = **new** Simple1(); // 新建Simple1组件

*frame* = **new** JFrame("Simple1"); // 新建JFrame

              // 处理关闭事件的通常方法

*frame*.addWindowListener(**new** WindowAdapter() {

**public** **void** windowClosing(WindowEvent e) {

                            System.*exit*(0);

                     }

              });

*frame*.getContentPane().add(*myPanel*);

*frame*.pack();

*frame*.setVisible(**true**);

       }

}

让我们来看看以上代码是按钮被点击，然后在对话框中显示相应的内容。如何工作的。在main方法中，我们定义了一个JFrame，然后将面板Jpanel添加到窗体中，该面板包括两个按钮。相应的变量Frame,button1,button2定义在程序的开头部分。在程序入口main方法中，首先新建Simple1组件，通过构造器建立用户GUI，定义一个面板Jpanle,，增加两个按钮，然后利用JButton.addActionListerner将两个按钮加入到一个活动监听器SimpleLister中，最后，两个按钮添加到面板。当GUI建立后，我们将面板添加到窗体并显示结果。当用户点击按钮时，程序调用actionPerformed方法，通过if语句来判断是哪一个

利用一个监听器来处理事件的缺点是，当程序比较复杂时，需要一大串的if 语句来实现，程序代码较难阅读与维护。当然，如果处理的事件较少，这种方式比较简单。

通过使用匿名内部类可以解决上述存在的问题。使用简单的匿名内部类作为addActionListener的参数即可。以下是实现代码：

**package** org.leno.swing.demo2;

/\*

 \* Simple2.java - 处理事件的第二种方法

 \*    在这个例子中，利用匿名内部类来监听每一个事件源产生的事件

 \*    避免使用一些if语句来决定是哪个事件源

 \*/

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** Simple2 {

**private** **static** JFrame *frame*; // 定义为静态变量以便main使用

**private** **static** JPanel *myPanel*; // 该面板用来放置按钮组件

**private** JButton button1; // 这里定义按钮组件

**private** JButton button2; // 以便让ActionListener使用

**public** Simple2() // 构造器, 建立图形界面

       {

              // 新建面板

*myPanel* = **new** JPanel();

              // 新建按钮

              button1 = **new** JButton("按钮1"); // 新建按钮1

              button2 = **new** JButton("按钮2");

              // 每一个事件源需要一个监听器

              // 定义一个匿名内部类来监听事件源产生的事件

              button1.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

                            JOptionPane.*showMessageDialog*(*frame*, "按钮1 被点击");

                     }

              });

              button2.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

                            JOptionPane.*showMessageDialog*(*frame*, "按钮2 被点击");

                     }

              });

*myPanel*.add(button1); // 添加按钮到面板

*myPanel*.add(button2);

       }

**public** **static** **void** main(String s[]) {

              Simple2 gui = **new** Simple2(); // 新建Simple2组件

*frame* = **new** JFrame("Simple2"); // 新建JFrame

              // 处理关闭事件的通常方法

*frame*.addWindowListener(**new** WindowAdapter() {

**public** **void** windowClosing(WindowEvent e) {

                            System.*exit*(0);

                     }

              });

*frame*.getContentPane().add(*myPanel*);

*frame*.pack();

*frame*.setVisible(**true**);

       }

}

使用匿名内部类同样存在许多另外的问题。首先，根据组件在代码中被定义的不同位置，类的定义以及处理事件的代码将分散在程序的各个部分，不是集中在一块，同样不便于阅读与维护。各事件的处理全部由嵌套的程序块组成，视觉上很难定位程序代码。如果事件处理程序比较复杂，内部类中的代码将变得很长，你将找不到相应的组件定义位置。最后，当工具栏、菜单栏目等需要处理同样的用户行为时，该方法将使代码更难维护。

我们使用一般的命名内部类可以解决以上许多问题。所有的事件处理方法都集中在一块，并且都具有有意义的名称，程序非常容易阅读与维护。单个的事件处理程序也可以被工具栏、菜单栏等重复使用，以下是实现代码：

**package** org.leno.swing.demo2;

/\*

 \* Simple3.java - 处理事件的第三种方法

 \*    For this example, we will use inner member classes to

 \*    在这个例子中，利用一般内部类来监听每个事件源产生的事件

 \*    该方法避免了第二种方法中由于使用匿名内部类而导致的代码混乱

 \*    便于集中处理事件代码

 \*    每一个Hander可以被工具栏或菜单多次使用

 \*/

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** Simple3 {

**private** **static** JFrame *frame*; // 定义为静态变量以便main使用

**private** **static** JPanel *myPanel*; // 该面板用来放置按钮组件

**private** JButton button1; // 这里定义按钮组件

**private** JButton button2; // 以便让ActionListener使用

       // 利用一般内部类来监听每一个事件源产生的事件如(button1, button2)

**private** **class** Button1Handler **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

                     JOptionPane.*showMessageDialog*(*frame*, "按钮1 被点击");

              }

       }

**private** **class** Button2Handler **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

                     JOptionPane.*showMessageDialog*(*frame*, "按钮2 被点击");

              }

       }

**public** Simple3() // 构造器, 建立图形界面

       {

              // 新建面板

*myPanel* = **new** JPanel();

              // 新建按钮

              button1 = **new** JButton("按钮1"); // 新建按钮1

              button2 = **new** JButton("按钮2");

              // 对每一个组件注册监听内部类

              button1.addActionListener(**new** Button1Handler());

              button2.addActionListener(**new** Button2Handler());

*myPanel*.add(button1); // 添加按钮到面板

*myPanel*.add(button2);

       }

**public** **static** **void** main(String s[]) {

              Simple3 gui = **new** Simple3(); // 新建Simple3组件

*frame* = **new** JFrame("Simple3"); // 新建JFrame

              // 处理关闭事件的通常方法

*frame*.addWindowListener(**new** WindowAdapter() {

**public** **void** windowClosing(WindowEvent e) {

                            System.*exit*(0);

                     }

              });

*frame*.getContentPane().add(*myPanel*);

*frame*.pack();

*frame*.setVisible(**true**);

       }

}

以上分析了在Java Swing中三种事件处理的方式，其中利用一般内部类来实现的方法，从代码书写、阅读、维护以及程序的可扩展性角度来看，最为值得推荐。