大学计算机基础教育规划教材

Java 语言应用开发基础

柳西玲　许　斌　等编著

清华大学出版社
北　京

内 容 简 介

本书是以 Java 2 技术为背景的 Java 应用开发技术基础教程。全书分为 8 章，内容包括图形用户界面的编程基础、Java 数据库连接设计、Java 应用开发平台的基础知识、XML 基础知识、Servlet 与 JSP 技术、JavaBean 技术、Web Services 的基础知识和面向服务的体系结构等。本书强调基本概念、基本技术和基本方法的阐述，注重理论联系实际。书中列举许多实例，每章都有练习题，利于读者提高实际解决问题的能力。附录给出功能较全面的 NetBeans IDE 的 GUI 开发应用说明。

本书的主要读者对象为非计算机专业的本科学生，也可作为各大专院校的选修课程教材或 Java 编程爱好者的参考书。对于 Java 语言不熟悉的读者，可先学习清华大学出版社出版的《Java 语言程序设计基础》一书。

图书在版编目(CIP)数据

Java 语言应用开发基础/柳西玲等编著. —北京：清华大学出版社，2006.10
(大学计算机基础教育规划教材)
ISBN 7-302-13481-2

Ⅰ. J…　Ⅱ. 柳…　Ⅲ. JAVA 语言—程序设计—高等学校—教材　Ⅳ. TP312

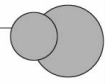
中国版本图书馆 CIP 数据核字(2006)第 083345 号

出 版 者：清华大学出版社
　　http://www.tup.com.cn
　　社 总 机：010-62770175
组稿编辑：张　民
文稿编辑：顾　冰
印 刷 者：北京世界知识印刷厂
装 订 者：三河市李旗庄少明装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　印张：19　字数：445 千字
版　　次：2006 年 10 月第 1 版　2006 年 10 月第1 次印刷
书　　号：ISBN 7-302-13481-2/TP・8457
印　　数：1～4000
定　　价：25.00 元

地　　址：北京清华大学学研大厦
邮　　编：100084
客户服务：010-62776969

第1章

图形用户界面的编程基础



1.1 概述

当今，所有的应用软件都要求有友好的用户界面，图形用户界面自然是首选。Java编程语言是一种跨平台的编程语言，在编写图形用户界面方面，也要支持跨平台功能。为此，Java提供了强大而丰富的图形界面工具包。Java的图形用户界面技术经历了两个发展阶段，分别通过提供AWT包和Swing包来体现，而且功能越来越强大，界面设计越来越美观。

无论是采用AWT包还是Swing包，Java图形用户界面的编写普遍采用构件化的思想来进行。AWT和Swing本身提供的也是许多标准的构件和容器。在进行界面设计的时候，关键要掌握好3个设计要领：首先是界面中构件的放置，即构件的布局；其次是考虑构件如何去响应用户的操作；第三是考虑每种构件的显示效果。因此，在利用AWT和Swing工具包时，当然需要掌握AWT和Swing中每个构件的显示效果和响应用户操作的方法。

1.2 用AWT对图形用户界面的编程

抽象窗口工具箱(abstract window toolkit，AWT)是Java提供的建立图形用户界面(graphic user interface，GUI)的开发包，AWT可用于Java的applet和application中。它支持图形用户界面编程的主要功能包括：用户界面构件；事件处理机制；图形和图像工具(形状、颜色和字体类)；布局管理器，它可以进行灵活的窗口布局而独立于窗口的尺寸和屏幕分辨率；数据传送类，可以通过本地平台的剪贴板来进行剪切和粘贴。

1.2.1 java.awt包

java.awt包中提供了GUI设计所使用的类和接口，可从图1.1中看到其中主要的类之间的关系。

java.awt包在使用上涉及3个主要的概念，每个概念对应着一个类：

- 构件类(Component)　它是java.awt包的核心。是一个抽象类，其他相关构件都从它派生而来。

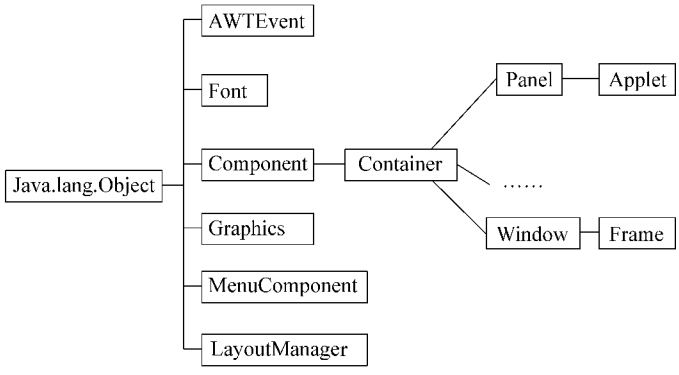


图 1.1 java.awt 包中的类和接口

- 容器类(Container) 由 Component 类派生而来,用来管理构件。
- 布局管理器类(LayoutManager) 用于确定容器内构件的布局。

1.2.2 构件和容器

构件类是 AWT 的最基本组成部分,这里的构件是一个图形化显示在屏幕上并能与用户进行交互的对象,例如 1 个按钮、1 个对话框、1 个图标等。应注意的是:构件本身并不能独立地显示,必须将构件放在一定的容器中才可以显示出来。

类 java.awt.Component 是许多构件类的父类,在编程过程中采用的都是 Component 类的子类;但是 Component 类中也封装了构件通用的方法和属性,如图形的构件对象、大小、显示位置、前景色和背景色、边界、可见性等,因此许多构件类也就继承了 Component 类的成员方法和成员变量。Component 类的部分重要的成员方法包括:

- getComponentAt(int x, int y) 获得坐标(x,y)上的构件对象
- getFont( ) 获得构件的字体
- getForeground( ) 获得构件的前景色
- getName( ) 获得构件的名字
- getSize( ) 获得构件的大小
- paint(Graphics g) 绘制构件
- repaint( ) 重新绘制构件
- update( ) 刷新构件
- setVisible(Boolean b) 设置构件是否可见
- setSize(Dimension d) 设置构件的大小
- setName(String name) 设置构件的名字

容器类是 Component 的子类,因此容器本身也是一个构件,具有构件的所有性质,另外还具有放置其他构件和容器的功能。一个容器可以容纳多个构件作为一个整体来使用。容器类可以想像成 1 个存储东西的盒子,构件是盒子中的东西,编写图形界面的过程,就像把东西放置到盒子的过程。为实现跨平台的特性和获得动态的布局效果,AWT

的容器内构件的大小和位置由“布局管理器”负责管理，而不采用绝对坐标的方式，使编程者可摆脱一个个设置构件的繁琐劳动。不同的布局管理器使用不同算法和策略，容器可以通过选择不同的布局管理器来决定布局，AWT 中的布局管理器包括：FlowLayout，BorderLayout，GridLayout，CardLayout 和 GridBagLayout。此外，在复杂的图形用户界面设计中，为了使布局更加易于管理，具有简洁的整体风格，一个包含了多个构件的容器本身也可作为一个构件加到另一个容器中去，容器中再添加容器，这样就形成了容器的嵌套。可获得良好的动态布局效果。因此，在实际编程过程，更多地采用容器类 Container 派生的子类。容器可简化图形界面的设计，用整体结构来布局界面。实际上，布局管理器是管理构件在容器中的布局工具，LayoutManager 本身是一个接口，编程中使用的是实现了该接口的类。每个容器都有一个布局管理器，当容器需要对构件进行定位或确定其大小时，会调用它所对应的布局管理器。这种利用布局管理器来管理构件在容器中的布局，而不直接设置构件位置和大小的方法，保证生成的图形用户界面具有良好的平台无关性。

在程序中安排构件的位置和大小时，应该注意以下两点：

- 由于各个构件的大小和位置是容器中的布局管理器负责，因此编程者不能直接用 Java 语言提供的 setLocation( )，setSize( )，setBounds( )等方法来设置构件属性，否则会被布局管理器所覆盖。
- 如果编程者确实需要亲自设置构件大小或位置，则应使用 setLayout(null)方法先取消该容器的布局管理器，然后再调用 setLocation( )，setSize( )，setBounds( )等方法。

1.2.3　常用容器

容器有 3 种类型：Window，Panel，ScrollPane。它们的子类中常用的有 Frame，Panel，Applet。所有的构件都可以通过 add( )方法向容器中添加构件。这里列举例 1.1 和例 1.2 分别说明 Frame 和 Panel 类的使用。

例 1.1　FirstFrame.java

```
import java.awt.*;
public class FirstFrame extends Frame{
  public static void main(String args[]){
    FirstFrame fr = new FirstFrame("帧窗口容器!");    //构造方法
    fr.setSize(100,100);                    //设置 Frame 的大小，默认为(0,0)
    fr.setBackground(Color.yellow);         //设置 Frame 的背景为黄色，默认值为白色
    fr.setVisible(true);                    //设置 Frame 为可见，默认为不可见
  }
  public FirstFrame (String str){
    super(str);                             //调用父类的构造方法
  }
}
```

程序运行结果如图 1.2 所示。

图 1.2　FirstFrame.java 运行结果

该例子中的容器是一个帧窗口，窗口中并没有放置其他构件；生成一个帧窗口通常是用 Window 的子类 Frame 来进行实例化，而不是直接用 Window 类。Frame 的外观就像平常在 Windows 系统下见到的窗口，有标题、边框、菜单、大小等等。每个 Frame 的对象实例化以后，都是没有大小和不可见的，因此必须调用 setSize( ) 来设置大小，调用 setVisible(true) 来设置该窗口为可见的。另外，AWT 在实际的运行过程中是调用所在平台的图形系统，因此同样一段 AWT 程序在不同的操作系统平台下运行所看到的图形系统是不一样的。例如在 Windows 下运行，则显示的窗口是 Windows 系统风格的窗口；而在 UNIX 下运行时，则显示的是 UNIX 风格的窗口。

需要注意的是：该例子只是生成了一个窗口，但是并不能响应用户的操作，即使是单击窗口右上角的关闭按钮，也不能关闭窗口。需要添加相应的代码才能够关闭窗口，但是可以在控制台命令行下用 Ctrl+C 键来终止程序的运行。

例 1.2　PanelInFrame.java

```
import java.awt.*;
public class PanelInFrame extends Frame{
    public static void main(String args[]){
        PanelInFrame fr = new PanelInFrame("在帧中显示面板");
        Panel pan=new Panel( );
        fr.setSize(200,200);
        fr.setBackground(Color.yellow);      //框架 fr 的背景颜色设置为黄色
        fr.setLayout(null);                  //取消布局管理器
        pan.setSize(100,100);
        pan.setBackground(Color.green);      //设置面板 pan 的背景颜色为绿色
        fr.add(pan);                         //用 add 方法把面板 pan 添加到框架 fr 中
        fr.setVisible(true);
        }
public PanelInFrame(String str){
        super(str);
    }
}
```

程序运行结果如图 1.3 所示。

图 1.3　PanelInFrame.java 运行结果

注意：该例在控制台的命令行下用 Ctrl＋C 键来终止程序的运行。

Panel 通常也称为面板，Panel 是一种透明的容器，既没有标题，也没有边框，就像一块透明的玻璃。与 Frame 不同，它不能作为最外层的容器单独存在，它首先必须作为一个构件放置到其他容器中，然后在把它当作容器，把其他构件放到它里面。Panel 的作用就是充分利用它既是构件又是容器的特点，在后面的例子中会看到。

1.2.4　容器布局管理

Java 为了实现跨平台的特性并且获得动态的布局效果，Java 将容器内的所有构件安排给一个“布局管理器”负责管理，如排列顺序，构件的大小、位置，当窗口移动或调整大小后构件如何变化等功能授权给对应的容器布局管理器来管理，不同的布局管理器使用不同算法和策略，容器可以通过选择不同的布局管理器来决定布局。

1. FlowLayout 布局管理器

FlowLayout 是 Panel 和 Applet 的默认布局管理器。构件在容器中的放置是从上到下、从左到右进行，如果容器足够宽，第一个构件先添加到容器中第一行的最左边，后续的构件依次添加到上一个构件的右边，如果当前行已放置不下该构件，则放置到下一行的最左边。在例 1.3 中有两个构件，分别是按钮“确认”和“取消确认”，它们在第一行中从左到右依次排列；而构件的大小采用的是它们的最佳尺寸，注意到两个按钮的宽度是不一样的，因为每个按钮显示的字符串的长度不一样，但是每个按钮都能够恰好把字符串显示出来。FlowLayout 的构造方法主要有下面几种：

- FlowLayout(参数 1，参数 2，参数 3)　参数 1 表示构件的对齐方式，指构件在这一行中的位置是居中对齐、居右对齐还是居左对齐，参数 2 是构件之间的横向间隔，参数 3 是构件之间的纵向间隔，单位是像素。
- FlowLayout(参数 1)　参数 1 是表示构件的对齐方式，指构件在这一行中的位置是居中对齐、居右对齐还是居左对齐，横向间隔和纵向间隔都是默认值 5 个像素。
- FlowLayout( )　默认的对齐方式居中对齐，横向间隔和纵向间隔都是默认值 5 个像素。

下面例 1.3 是设计有两个按钮的帧窗口。

例 1.3 TwoButtons.java

```
import java.awt.*;
public class TwoButtons{
    private Frame f;
    private Button b1;
    private Button b2;
      public static void main(String args[]){
        TwoButtons two= new TwoButtons();
        two.go();
      }
      public void go(){
        f = new Frame("FlowLayout");
        f.setLayout(new FlowLayout());    //设置布局管理器为 FlowLayout
        b1 = new Button("确认");          //按钮上显示字符"确认"
        b2 = new Button("取消确认");
        f.add(b1);
        f.add(b2);
        //紧凑排列,其作用相当于 setSize(),即让窗口小到只包容 b1,b2 两个按钮
        f.pack();
        f.setVisible(true);
    }
}
```

程序运行结果如图 1.4 所示。

图 1.4 TwoButtons.java **运行结果**

注意：该例在控制台的命令行下用 Ctrl+C 键来终止程序的运行。

当容器的大小发生变化时，用 FlowLayout 管理的构件会发生变化，其变化规律是：构件的大小不变，但是相对位置会发生变化。如上例改为有 3 个按钮都处于同一行，但是如果把该窗口变窄，窄到不能把有 3 个按钮都处于同一行时，它会自动把按钮折到下一行。如例 1.4 所示。FlowLayout 的变化规律是：构件的大小不变，但是相对位置会发生变化。

例 1.4 ThreeButtons.java

```
import java.awt.*;
public class ThreeButtons{
  public static void main(String args[]){
    Frame f = new Frame();
    f.setLayout(new FlowLayout());
    Button button1 = new Button("确认");
```

```
        Button button2 = new Button("打开");
        Button button3 = new Button("关闭");
        f.add(button1);
        f.add(button2);
        f.add(button3);
        f.setSize(50,100);
        f.setVisible(true);
    }
}
```

程序运行结果如图 1.5 所示。

图 1.5　ThreeButtons.java 运行结果

注意：该例在控制台的命令行下用 Ctrl+C 键来终止程序的运行。

2. BorderLayout 布局管理器

BorderLayout **是** Window,Frame **和** Dialog **的默认布局管理器。**BorderLayout **布局管理器把容器分成 5 个区域：北、南、东、西和中，每个区域只能放置一个构件。如果容器采用** BorderLayout **进行布局管理，在用** add( )**方法添加构件的时候，必须注明添加到哪个位置。在使用** BorderLayout **的时候，如果容器的大小发生变化，其变化规律为：构件的相对位置不变，大小发生变化。例如容器变高了，则北、南区域不变，而西、中、东区域变高；如果容器变宽了，西、东区域不变，北、中、南区域变宽。不一定所有的区域都有构件，如果四周的区域没有构件，则由中区域去补充，但是如果中区域没有构件，则保持空白，其效果如图 1.6 所示。**

(a) 北区没有构件　　(b) 中区没有构件

图 1.6　BorderLayout 进行布局管理的区域特征

例 1.5　FiveButtons.java

```
import java.awt.*;
public class FiveButtons{
public static void main(String args[]){
    Frame f = new Frame("BorderLayout");
    f.setLayout(new BorderLayout( ));
    f.add("北", new Button("North"));   //第一个参数表示把按钮添加到容器的北区域
```

```
        f.add("南", new Button("South"));   //第一个参数表示把按钮添加到容器的南区域
        f.add("东", new Button("East"));    //第一个参数表示把按钮添加到容器的东区域
        f.add("西", new Button("West"));    //第一个参数表示把按钮添加到容器的西区域
        f.add("中", new Button("Center"));  //第一个参数表示把按钮添加到容器的中区域
        f.setSize(100,100);
        f.setVisible(true);
    }
}
```

运行结果如图 1.7 所示。

图 1.7　FiveButtons.java 运行结果

注意：该例在控制台的命令行下用 Ctrl+C 键来终止程序的运行。

3. GridLayout 布局管理器

GridLayout 布局管理器使容器中各个构件呈网格状布局，平均占据容器的空间。即使容器的大小发生变化，每个构件还是平均占据容器的空间。构件在往容器中放置的时候，是按照从上到下、从左到右的规律进行的。

例 1.6　ButtonGrid.java

```
import java.awt.*;
public class ButtonGrid {
    public static void main(String args[]){
        Frame f = new Frame("GridLayout");
        f.setLayout(new GridLayout(3,2));      //容器平均分成 3 行 2 列共 6 格
        f.add(new Button("1"));                //添加到第一行的第一格
        f.add(new Button("2"));                //添加到第一行的下一格
        f.add(new Button("3"));                //添加到第二行的第一格
        f.add(new Button("4"));                //添加到第二行的下一格
        f.add(new Button("5"));                //添加到第三行的第一格
        f.add(new Button("6"));                //添加到第三行的下一格
        f.setSize(100,100);
        f.setVisible(true);
    }
}
```

程序运行结果如图 1.8 所示。

图 1.8　ButtonGrid.java 运行结果

注意：该例在控制台的命令行下用 Ctrl＋C 键来终止程序的运行。

4. CardLayout 布局管理器

CardLayout **布局管理器能够帮助用户处理两个以至更多的成员共享同一显示空间，它把容器分成许多层，每层的显示空间占据整个容器的大小，但是每层只允许放置一个构件，当然每层都可以利用 Panel 来实现复杂的用户界面。例 1.7 中的牌布局管理器(CardLayout)就像一副叠得整整齐齐的扑克牌一样，有 54 张牌，但是只能看见最上面的一张牌，每一张牌就相当于牌布局管理器中的每一层。它实现了 LayoutManager2 接口。**

构造方法如下：

CardLayout( ); CardLayout(int hgap,int vgap)

其他的方法有：

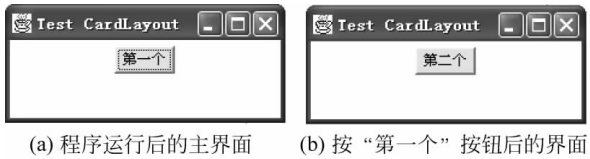
- first(Container parent)　**显示第一张卡片。**
- last(Container parent)　**显示最后一张卡片。**
- next(Container parent)　**显示下一张卡片。**
- previous(Container parent)　**显示前一张卡片。**
- show(Container parent,String name)　**显示指定名称的卡片。**

例 1.7　TestCardLayout.java

```
import java.awt.*;
import java.awt.event.*;          //事件处理机制,1.3 节的内容中将介绍
public class TestCardLayout implements ActionListener{
  private Panel p1,p2,p3;
  private Button b1,b2,b3;
  private Frame f;
  private CardLayout CLayout=new CardLayout( );   //定义 CardLayout 布局管理器
  public void create( ){
    b1=new Button("第一个");
    b2=new Button("第二个");
    b3=new Button("第三个");
    p1=new Panel( );
    p2=new Panel( );
    p3=new Panel( );
    f=new Frame("Test CardLayout");
    p1.add(b1);
    b1.addActionListener(this);      //为按钮注册监听器
    p2.add(b2);
```

```
        b2.addActionListener(this);
        p3.add(b3);
        b3.addActionListener(this);
        f.setLayout(CLayout);
        f.add(p1,"第一层");
        f.add(p2,"第二层");
        f.add(p3,"第三层");
        f.setSize(200,100);
        f.setVisible(true);
    }
    public static void main(String args[]){
        TestCardLayout tc=new TestCardLayout();
        tc.create();
    }
    public void actionPerformed(ActionEvent e){
        CLayout.next(f);                    //当按钮被单击时,实现显示下一张卡片的功能
    }
}
```

程序运行结果如图 1.9 所示。



(a) 程序运行后的主界面　　(b) 按“第一个”按钮后的界面

(c) 按“第二个”按钮后的界面

图 1.9　TestCardLayout.java 运行结果

注意：该例在控制台的命令行下用 Ctrl+C 键来终止程序的运行。

5. 容器的嵌套

在复杂的图形用户界面设计中，为了使布局更加易于管理，具有简洁的整体风格，一个包含了多个构件的容器本身也可以作为一个构件加到另一个容器中去，容器中再添加容器，这样就形成了容器的嵌套。下面是一个容器嵌套的例子。

例 1.8　TestNesting.java

```
import java.awt.*;
public class TestNesting {
    private Frame f;
    private Panel p;
```

```
    private Button bw,bc;
    private Button bfile,bhelp;
    private TextArea ta;
    public void create( ){
      f=new Frame("容器嵌套");
      ta=new TextArea("文本范围");
      bw=new Button("西");
      bc=new Button("工作空间区域");
      f.add(bw,"West");
      f.add(bc,"Center");
      p=new Panel( );
      p.setLayout(new BorderLayout( ));
      f.add(p,"North");          //面板 p 作为一个构件添加到窗口 f 中
      bfile=new Button("文件");
      bhelp=new Button("帮助");
      p.add(bfile,"North");      //按钮 bfile 被添加到面板 p
      p.add(bhelp,"West");       //按钮 bhelp 被添加到面板 p
      p.add(ta,"Center");        //文本区域 ta 被添加到面板 p
      f.pack( );
      f.setVisible(true);
    }
    public static void main(String args[ ]){
      TestNesting tn=new TestNesting( );
      tn.create( );
    }
}
```

程序运行结果如图 1.10 所示。

图 1.10　TestNesting.java 运行结果

注意：该例在控制台的命令行下用 Ctrl+C 键来终止程序的运行。

通过以上实例，看到布局管理器的作用。虽然，从后面的章节中得知，许多布局管理器已在 Swing 容器中隐藏起来，只在创建 JPanel 或 content pane 上添加构件时，如对默认布局不满意可以考虑布局管理器的使用。各种布局管理器都有自己的优势，归纳起来使用的原则是：

- 如希望在尽可能大的范围内显示一个构件，应使用 BorderLayout 或 GridBagLayout。用 BorderLayout 必须将构件放在正中间。用 GridBagLayout 对构件要求设置：fill＝GridBagContaints. BOTH。另外，还可用 BoxLayout，将构件声明为最大尺寸。
- 如想在一行中显示多个尺寸不改的构件，最好使用 JPanel 将这些构件分为一组，再用另外一个 JPanel 默认的 FlowLayout 或 BoxLayout 管理器。
- 如想把构件显示成几行和列，用 GridBagLayout 最方便。
- 如想在一行（或一列）中显示多个构件，并对构件之间设置不同间隔或不同对齐方式，用 BoxLayout 最容易实现。
- 如想显示的构件较多，排列也复杂时，可用 GridBagLayout。

1.3 AWT 事件处理机制

1.2 节中的主要内容是如何放置各种构件，使图形界面更加丰富多彩，但是还不能响应用户的任何操作，所有实例都不能自动终止程序运行。要能够让图形界面接收用户的操作，就必须给各个构件加上事件处理机制。比如上述几节中的程序例子还不能在运行过程中进行窗口的关闭，就是因为程序中没有处理窗口事件。而本节的内容将解决这些问题。

在事件处理的过程中，主要涉及 3 类对象：

- 事件（event） 用户对界面操作在 Java 语言上的描述，以类的形式出现，例如键盘操作对应的事件类是 KeyEvent。
- 事件源（event source） 事件发生的场所，通常就是各个构件，例如按钮 Button。
- 事件处理器（event handler） 接收事件对象并对其进行处理的对象。

如用户用鼠标单击了按钮对象，则该按钮就是事件源，而 Java 运行时系统会生成 ActionEvent 类的对象 actionE，该对象中描述了该单击事件发生时的一些信息，然后，事件处理者对象将接收由 Java 运行时系统传递过来的事件对象 actionE 并进行相应的处理。

由于同一个事件源上可能发生多种事件，因此 Java 采取了授权处理机制（delegation model），如图 1.11 所示。事件源可以把在其自身所有可能发生的事件分别授权给不同的事件处理器来处理。如在画布对象 Canvas 上既可能发生鼠标事件，也可能发生键盘事件，该 Canvas 对象就可以授权给事件处理器去处理鼠标事件，也同时可授权给事件处理器去处理键盘事件。事件处理器也可称为监听器，因为它负责监听事件源上所有发生的事件，一旦事件类型与自己所负责处理的事件类型一致，就马上进行处理。授权机制把事件的处理委托给外部的处理实体，实现事件源和监听器分开的目的。事件处理器（监听器）通常是一个类，该类如果要能够处理某种类型的事件，就必须实现与该事件类型相对应的接口。在例 1.9 中，类 ButtonHandler 之所以能够处理 ActionEvent 事件，是因为它实现了与 ActionEvent 事件对应的接口 ActionListener。每个事件类都有一个与之相对

应的接口。

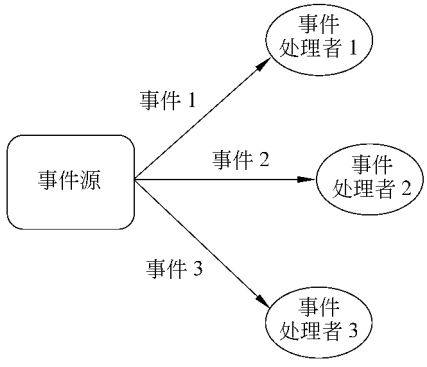


图 1.11　授权处理机制

例 1.9　TestButton.java

```
import java.awt.*;
import java.awt.event.*;
  public class TestButton {
    public static void main(String args[]){
    Frame f = new Frame("TestButton");
    Button b = new Button("确认");
    /*注册监听器进行授权,该方法的参数是事件处理器对象,要处理的事件类型可以
    从方法名中看出,本方法要授权处理的是 ActionEvent 事件,因为方法名是
    addActionListener*/
    b.addActionListener(new ButtonHandler());
    f.setLayout(new FlowLayout());          //设置布局管理器
    f.add(b);
    f.setSize(200,100);
    f.setVisible(true);
    }
  }
//实现接口 ActionListener 才能充当事件 ActionEvent 的处理器
class ButtonHandler implements ActionListener {
  //系统产生的 ActionEvent 事件对象被当作参数传递给该方法
  public void actionPerformed(ActionEvent e){
    //本接口只有一个方法,因此事件发生时,系统会自动调用本方法
    System.out.println("Action occurred");
  }
}
```

程序运行结果如图 1.12 所示。

图 1.12　TestButton.java 运行结果

该例子的执行情况是，当单击按钮后，就会在命令行中输出 Action occurred。

注意：该例在控制台的命令行下用 Ctrl+C 键来终止程序的运行。

在上面这个例子中，事件源就是按钮；事件就是 ActionEvent 对象，而这种事件的产生原因在于用户单击了该事件源按钮；而事件处理者就是 ButtonHandler 对象，事件发生以后，Java 的运行时系统会自动调用 ButtonHandler 对象的 actionPerformed( )方法进行处理，而且事件 ActionEvent 对象将被当作参数传递给 actionPerformed( )方法，在 actionPerformed( )方法可以通过读取 ActionEvent 对象的相关信息来得到事件发生时候的情况。

使用授权处理模型进行事件处理的一般方法归纳如下：

- 对于某种类型的事件×××Event，要想接收并处理这类事件，必须定义相应的事件处理类，该类需要实现与该事件相对应的接口×××Listener；
- 事件源实例化以后，必须进行授权，注册该类事件的监听器，使用 add×××Listener(×××Listener) 方法来注册监听器。

1.3.1　事件类

1. java.lang.Object 类

java.lang.Object 类的层次如下：

```
java.lang.Object
    |
    +--java.util.EventObject
```

java.util.EventObject 类是所有事件对象的基础父类，所有事件都是由它派生而来，如图 1.13 所示。

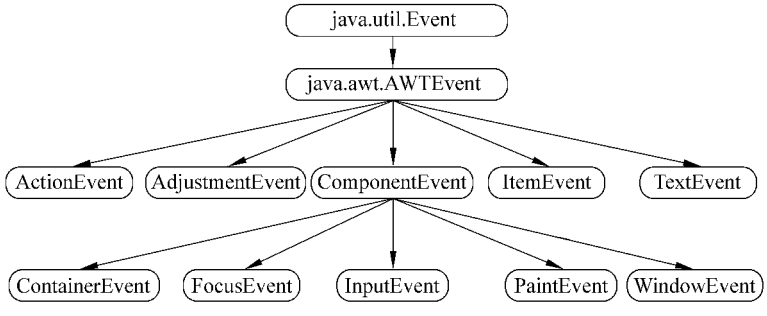


图 1.13　AWT 事件类的层次图

```
public class EventObject implements java.io.Serialization {
   protected EventObject(Object source);
   public Object getSource( );
   public String toString( );
}
```

EventObject 类继承于 java.lang.Object 类，并实现了串行化接口。通过 getSource( ) 方法可以得到事件源对象。

2. java.awt.AWTEvent

与 AWT 有关的所有事件类都由 java.awt.AWTEvent 类派生，它也是 EventObject 类的子类。这些 AWT 事件分为两大类：低级事件和高级事件。低级事件是指基于构件和容器的事件，当一个构件上发生事件(如鼠标的进入、单击、拖放等)或构件的窗口开关等都属于低级事件。高级事件是基于语意的事件，它可以不和特定的动作相关联，而依赖于触发此事件的类，如在 TextField 中按 Enter 键，或是选中项目列表的某一选项就会触发 ActionEvent 事件。

(1) 低级事件

- ComponentEvent(构件事件)　构件尺寸的变化，移动。
- ContainerEvent(容器事件)　构件增加，移动。
- WindowEvent(窗口事件)　关闭窗口，窗口闭合，图标化。
- FocusEvent(焦点事件)　焦点的获得和丢失。
- KeyEvent(键盘事件)　键按下、释放。
- MouseEvent(鼠标事件)　鼠标单击，移动。

(2) 高级事件(语义事件)

- ActionEvent(动作事件)　按钮按下，TextField 中按 Enter 键。
- AdjustmentEvent(调节事件)　在滚动条上移动滑块以调节数值。
- ItemEvent(项目事件)　选择项目，不选择“项目改变”。
- TextEvent(文本事件)　文本对象改变。

1.3.2　事件监听器

每类事件都有对应的事件监听器，监听器是接口，根据动作来定义方法。例如，与键盘事件 KeyEvent 相对应的接口是：

```
public interface KeyListener extends EventListener {
     public void keyPressed(KeyEvent ev);
     public void keyReleased(KeyEvent ev);
     public void keyTyped(KeyEvent ev);
}
```

注意到在本接口中有 3 个方法，那么 Java 运行时系统何时调用哪个方法？其实根据这 3 个方法的方法名就能够知道应该是什么时候调用哪个方法执行了。当键盘刚按下去