# 图形用户界面

## AWT与Swing

通过图形用户界面(Graphics User Interface,GUI)，用户和程序之间可以方便地进行交互。Java早期在进行图形用户界面设计时，使用java.awt(Abstrac Window Toolkit)抽象窗口工具包中提供的类，比如利用Button创建按钮组件、TextField创建文本框组件等。当用java.awt包中的组件类创建组件时，都有一个相应的本地组件在为它工作(称为它的同位体)，AWT组件的设计原理是把与显示组件有关的许多工作和处理组件事件的工作交给相应的本地组件。因此我们把有同位体的组件称为重量组件，基于重量组件的GUI设计有很多不足之处，比如程序的外观在不同的平台上可能有所不同，而且重量组件的类型也不能满足GUI设计的需要，例如，不可能把一副图像添加到AWT按钮上，因为AWT按钮的外观绘制是由同位体来完成的，而同位体可能是用C++编写的，它的行为是不能被Java扩展的。另外，使用AWT进行GUI设计可能会消耗大量的系统资源。

在JDK1.2推出之后，Java提供了javax.swing包，该包提供了功能更为强大的用来设计GUI的类。javax.swing包中的类创建的组件称作Swing组件，其中大部分是轻量组件，没有同位体。Swing的轻组件在设计上和AWT完全不同，轻组件把与显示组件有关的许多处理组件事件的工作交给相应的用户界面(User Interface,UI)代表来完成，这些UI代表是用Java语言编写的类，这些类被增加到Java的运行环境中，因此组件的外观不依赖平台，不仅在不同平台上的外观是相同的，而且较重量组件而言有更高的性能。

然而，Swing并没有完全替换掉AWT，而是构建在AWT之上，除了包含AWT的替代组件外，还包含树和表等新组件。Swing和AWT包中部分类的关系如图10‑1所示，其中以J开头的类为Swing包中的类，无J开头的类为AWT包中的类。

在学习GUI编程时，必须很好地理解掌握两个概念：组件和容器。

1、组件

凡是能够以图形化的方式显示在屏幕上，并能与用户进行交互的对象都称为组件。

Java把Component类的子类或间接子类创建的对象称为一个组件。

2、容器

容器是一些实际上含有多组控件或其他容器的屏幕窗口。

Java把Container的子类或间接子类创建的对象称为一个容器。

容器本身也是一个组件，因此可以把一个容器添加到另一个容器中实现容器的嵌套。

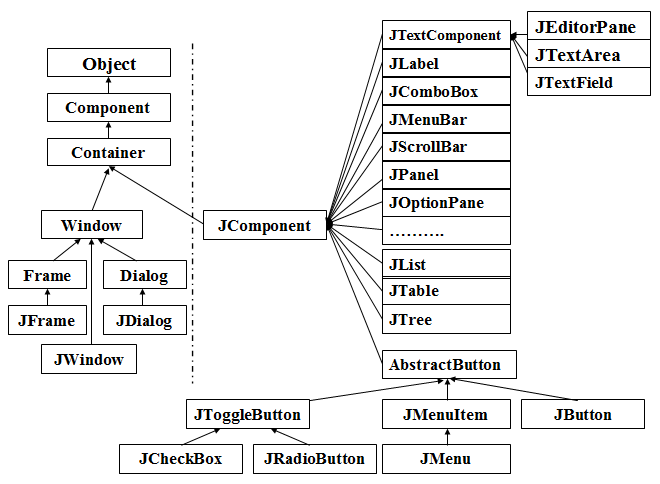


图10‑1 Swing和AWT包中部分类之间的关系

## Swing组件

Swing组件分为如下三种基本类型：

(1) 底层容器

底层容器直接和操作系统交互，可以被直接显示、绘制在操作系统所控制的平台上，包括JFrame、JDialog、JApplet、JWindow。它们是重量组件。

(2) 中间容器

中间容器是轻量组件，置于底层容器中，包括JPanel、Box、JScrollPane、JSplitPane、JLayeredPane等。

(3) 基本组件

实现人机交互的组件，包括JTextField、JTextArea、JButton、JLabel、JCheckBox、JRadioButton、JComboBox、JPasswordField等。

底层容器可以显示在屏幕上，非底层容器必须处于底层容器中才可以被显示出来。

### 底层容器

**1、JFrame**

JFrame类是Container类的间接子类，可以创建一个带有标题栏和控制按钮(最小化、恢复/最大化、关闭)的窗口。窗口也是一个容器，可以向窗口添加组件。

JFrame的常用方法如下：

(1) JFrame() 创建一个无标题的窗口。

(2) JFrame(String s) 创建标题为s的窗口。

(3) void setVisible(boolean b) 设置窗口是否可见，窗口默认是不可见的。

(4) void setTitle(String s) 设置标题。

(5) void setBounds(int a,int b,int width,int height) 设置窗口出现在屏幕上时的初始位置为(a,b)，即距屏幕左边a个像素、距屏幕上方b个像素；窗口的宽是width，高是height。

(6) void setSize(int width,int height) 设置窗口的大小，窗口在屏幕出现的默认位置是(0,0)。

(7) void setDefaultCloseOperation(int operation) 设置单击窗体右上角的关闭图标后，程序会做出怎样的处理，参数operation取值如下：

* EXIT\_ON\_CLOSE结束窗口所在的应用程序。
* DISPOSE\_ON\_CLOSE隐藏当前窗口，并释放窗体占有的其他资源。
* HIDE\_ON\_CLOSE 隐藏当前窗口。
* DO\_NOTHING\_ONCLOSE什么也不做。

这些参数都是javax.swing.JFrame类中的静态常量，需要导入，具体如下：

import static javax.swing.JFrame.\*;

(8) void dispose() 撤销当前窗口，并释放当前窗口所使用的资源。

例题 10.1

import javax.swing.\*;

import static javax.swing.JFrame.\*;

public class MyJFrame{

public static void main(String args[]){

JFrame myJFrame=new JFrame("我的窗口");

myJFrame.setBounds(100,100,300,150);

myJFrame.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

myJFrame.setVisible(true);

}

}

运行结果如图10‑2所示。

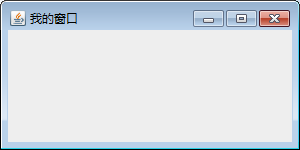


图10‑2 创建窗口

使用JFrame的构造方法创建一个标题为“我的窗口”的窗口，通过setBounds方法设置窗口的位置和大小，通过setDefaultCloseOperation方法设置窗口右上角关闭按钮的作用为关闭窗口并结束应用程序，通过setVisible方法设置窗口可见。

注意：一般setVisible方法写在程序的最后，保证窗口内所有组件都绘制完成后，被设置成可见。

创建窗口还可以通过创建JFrame类的子类实例来实现。

例题 10.2

通过创建JFrame子类的方式实现如图10‑2所示界面。

import javax.swing.\*;

import static javax.swing.JFrame.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class MyJFrameDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("我的窗口",100,100,300,150);

myJFrame.setVisible(true);

}

}

创建JFrame类的子类窗口MyJFrame类，并使用它创建窗口实例。子类窗口MyJFrame类的构造方法通过调用从父类继承而来的setTitle()、setBounds()、setDefaultCloseOperation()方法，完成窗口的属性设置。

课堂练习 10.1

指出下面程序的运行结果。

import javax.swing.\*;

import static javax.swing.JFrame.\*;

class MyJFrame1 extends JFrame{

MyJFrame1(String s,int x,int y,int w,int h){

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);

}

}

class MyJFrame2 extends JFrame{

MyJFrame2(String s,int x,int y,int w,int h){

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class Test{

public static void main(String args[]){

MyJFrame1 myJFrame1=new MyJFrame1("我的窗口",100,100,300,150);

myJFrame1.setVisible(true);

MyJFrame2 myJFrame2=new MyJFrame2("我的窗口",400,100,300,150);

myJFrame2.setVisible(true);

}

}

**2、JDialog**

JDialog通常用来设计具有依赖关系的窗口，分为有模式和无模式两种。有模式对话框，会堵塞其他线程的执行；无模式对话框，不堵塞其他线程。具体参见10.5对话框JDialog。

**3、JApplet**

JApplet用来创建小应用程序，在浏览器中运行。目前已经很少使用。

**4、JWindow**

JWindow是一个不带标题栏和控制按钮的窗口，使用较少。

### 中间容器

1、JPanel面板

通常使用JPanel创建面板，向面板中添加组件，然后将这个面板添加到其他容器中。

2、Box盒子

盒子容器就像盒子一样，被划分成一个个小格子。可以通过Box调用静态方法createHorizontalBox()创建一个横向盒子，或者调用静态方法createVerticalBox()创建一个纵向盒子。如果想控制盒中每个小格之间的距离，可以通过Box调用静态方法createHorizontalStrut(int width)创建一个不可见的固定宽度的支撑组件，Box调用静态方法createVerticalStrut(int height)创建一个不可见的固定高度的支撑组件。一个横向盒子，可以在组件之间插入固定宽度的支撑组件；一个纵向盒子，可以在组件之间插入固定高度的支撑组件。

3、JScrollPane滚动窗格

把组件放到一个滚动窗格中，通过滚动条操作组件。例如JTextArea不自带滚动条，因此可以把JTextArea组件放到滚动窗格中，间接实现滚动条。

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

4、JSplitPane拆分窗格

按照水平和垂直两种方式进行拆分，成为两个独立的容器，其拆分线可以移动。构造方法如下：

JSplitPane(int a,Component b,Component c)

参数a取值HORIZONTAL\_SPLIT、VERTICAL\_SPLIT，以决定是水平还是垂直拆分，后两个参数为决定要放置的组件。

5、JLayeredPane分层窗格

分层窗格共分为5层，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DEFAULT\_LAYER、PALETTE\_LAYER、MODAL\_LAYER、POPUP\_LAYER、DRAG\_LAYER。添加到DEFAULT\_LAYER层的组件，如果和其他层的组件发生重叠时，将被其他组件遮挡。当用户用鼠标移动某一组件时，该组件被自动设为DRAG\_LAYER层，显示在最上层。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。

分层窗口调用：

void setLayer(Component com,int layer)

可以重新设置组件com所在的层，调用：

int getLayer(Component com)

可以获取组件com所在的层数。

### 基本组件

1、JTextField文本框

允许用户在文本框中输入单行文本。

2、JTextArea文本区

允许用户在文本区输入多行文本。

3、JButton按钮

允许用户单击按钮。

4、JLabel标签

为用户提供信息提示。

5、JCheckBox复选框

为用户提供多项选择，通常以矩形框形式表现。

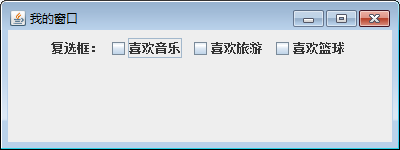


图10‑3 复选框

6、JRadioButton单选按钮

为用户提供单项选择，通常以圆圈形式表现。

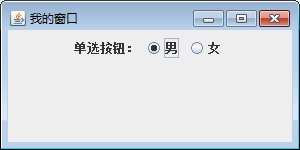


图10‑4 单选按钮

7、JComboBox下拉列表

为用户提供单项选择，用户可以在下拉列表中看到第一个选项和它旁边的箭头按钮，当用户单击箭头按钮时，选项列表打开。

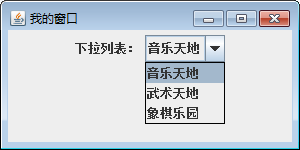


图10‑5 下拉列表框

8、JPasswordField密码框

允许用户在密码框中输入单行密码，默认回显字符为“\*”。

### 添加组件

1、添加按钮。

例题 10.3

import javax.swing.\*;

import static javax.swing.JFrame.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class JButtonDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("我的窗口",100,100,200,100);

JButton button=new JButton("按钮");

myJFrame.add(button);

myJFrame.setVisible(true);

}

}

上例在JFrame的实例myJFrame中添加了一个JButton实例button，结果如图10‑6所示。

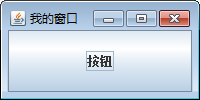


图10‑6 添加按钮

其实在JDK1.4或之前的版本中，是不能直接将JButton实例button添加到底层容器JFrame实例frame中的。语句

frame.add(button);

应为frame.getContentPane().add(button)。但是从JDK1.5版本开始，Java重写了add()、remove()和setLayout()这三个方法，可以由frame直接调用，等价于由frame.getContentPane()调用。

那么为什么不能直接将button添加到frame中去呢，这需要了解一下JFrame的层次结构，如图10‑7所示。

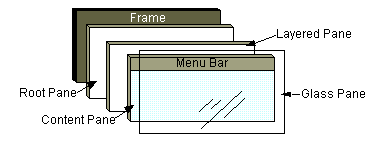


图10‑7 JFrame层次结构

(1) Glass Pane玻璃面板：默认是不可见的，可接收键盘或鼠标的响应事件等。

(2) Content Pane内容面板：一个可见的容器，除了菜单条外，用户所能看见的窗口中的组件都被添加在内容面板中。

(3) Menu Bar菜单条：可选组件。

(4) Layered Pane分层面板：定位内容面板、菜单条和Z方向上添加的其他组件。

(5) Root Pane底层面板：管理悬浮其上的Layered Pane、Menu Bar、Content Pane、Glass Pane。

由此可知，在JFrame的窗体中显示的内容本质上都被添加在内容面板中，只有add()、remove()和setLayout()这三个方法可以在写法上省略getContentPane()的调用，其他方法还是需要的。

课堂练习 10.2

编写程序实现如图10‑9所示的用户界面。

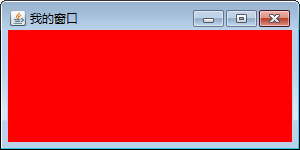


图10‑8 红色窗口

## 布局管理器

例题 10.3在JFrame实例中添加了一个按钮JButton实例button，但是我们注意到这个按钮跟我们常见的按钮有点不同，似乎太大了，占据了整个JFrame空间。这与底层容器JFrame的布局管理有关。

当把组件添加到容器中时，如果希望控制组件在容器中的位置，可以使用方法：

setLayout(布局对象);

设置自己的布局。常用的布局类有FlowLayout、BorderLayout、CardLayout、GridLayout、BoxLayout。如果不采用这些布局类，可以先设置布局对象为null，然后手动安排每个组件的位置。这是很麻烦的事情。因此一般绘制图形用户界面都需要设置布局对象。

### FlowLayout布局类

FlowLayout布局类是java.awt包中的布局类，是JPanel的默认布局。FlowLayout布局将窗口中的组件按照加入的先后顺序从左向右排列，一行排满之后就转入下一行继续从左至右排列。对于添加到使用FlowLayout布局的容器中的组件，组件调用setSize(int x,int y)设置的大小无效。



图10‑11 FlowLayout布局

FlowLayout布局对象调用setAlignment(int align)方法可以重新设置布局的对齐方式，其中参数align可以取值FlowLayout.LEFT, FlowLayout.CENTER, FlowLayout.RIGHT，默认取值CENTER。FlowLayout布局对象调用setHgap(int hgap)方法和setVgap(int vgap)可以重新设置水平间隙和垂直间隙，默认均为5个像素。

假设一个容器con需要设置为FlowLayout布局格式，对窗口中的组件按照FlowLayout布局安排位置。那么，con可以调用语句：

FlowLayout flow=new FlowLayout();

con.setLayout(flow);

或者

con.setLayout(new FlowLayout());

那么，对例题 10.3进行修改。

例题 10.5

import javax.swing.\*;

import static javax.swing.JFrame.\*;

***import java.awt.\*;***

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class JButtonDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("我的窗口",100,100,200,100);

***myJFrame.setLayout(new FlowLayout());***

JButton button=new JButton("按钮");

myJFrame.add(button);

myJFrame.setVisible(true);

}

}

结果如图10‑13所示。

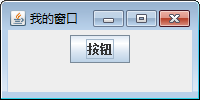


图10‑12 设置FlowLayout布局

这样的按钮看起来比较正常。但是我们也注意到整个JFrame中只有1个按钮，窗口明显是设置大了。其实可以在添加完组件后，使用语句

myJFrame.pack();

让窗体自动调整大小到刚好可以容纳要显示的内容，如图10‑13所示。



图10‑13 pack()之后

课堂练习 10.3

编写程序实现如图10‑11所示界面。

### BorderLayout布局类

BorderLayout布局类是java.awt包中的布局类，是JFrame、JDialog的默认布局。BorderLayout布局将窗口中的组件按照指定的空间位置(东、南、西、北、中)进行排定。

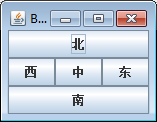


图10‑14 BorderLayout布局

假设一个容器con需要设置为BorderLayout布局格式，对窗口中的组件按照BorderLayout布局安排位置。那么，con可以调用语句：

BorderLayout border=new BorderLayout();

con.setLayout(border);

或者

con.setLayout(new BorderLayout());

而且，con在使用add()方法添加组件时，需要指定空间位置，取值为BorderLayout.CENTER、BorderLayout.NORTH、BorderLayout.SOUTH、BorderLayout.WEST、BorderLayout.EAST。如：

con.add(new JButton("北"),BorderLayout.NORTH);

在con的北部添加一个按钮“北”。

每个区域只能放置一个组件，如果向某个已经放置了组件的区域再放置一个组件，那么先前的组件将被后者替换掉。使用BorderLayout布局的容器最多能添加5个组件，如果需要加入更多组件，必须使用容器的嵌套或者改用其他的布局策略。

如果东西南北四个边界区域都没有使用，那么中部区域将会占据整个窗口。因此，例题 10.3添加的按钮填充了整个窗口。

例题 10.6

实现如图10‑14所示的界面。

import javax.swing.\*;

import static javax.swing.JFrame.\*;

import java.awt.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class BorderLayoutDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("BorderLayout Demo",100,100,300,300);

myJFrame.setLayout(new BorderLayout());

JButton button1=new JButton("东");

JButton button2=new JButton("南");

JButton button3=new JButton("西");

JButton button4=new JButton("北");

JButton button5=new JButton("中");

myJFrame.add(button1,BorderLayout.EAST);

myJFrame.add(button2,BorderLayout.SOUTH);

myJFrame.add(button3,BorderLayout.WEST);

myJFrame.add(button4,BorderLayout.NORTH);

myJFrame.add(button5,BorderLayout.CENTER);

myJFrame.pack();

myJFrame.setVisible(true);

}

}

### CardLayout布局类

CardLayout布局类是java.awt包中的布局类。CardLayout布局策略是将加入容器的组件层叠在容器中，最先加入容器的是第一张(在最上面)，依次向下排列。CardLayout布局的特点是，同一时刻容器中只能从组件中选一个出来显示，就像叠“扑克牌”，每次只能显示其中的一张，这个被显示的组件将占据所有的容器空间。

假设一个容器con需要设置为CardLayout布局格式，对窗口中的组件按照CardLayout布局安排位置。那么，con可以调用语句：

CardLayout card=new CardLayout ();

con.setLayout(card);

或者

con.setLayout(new CardLayout ());

容器可以调用add(String s,Component b)将组件b加入容器，并同时设置组件的代号s。组件的代号是一个字符串，和组件的名字没有必然联系。但是，不同的组件代码必须互不相同。最先加入con的是第一张，依次排序。

创建的布局card用CardLayout类提供的show()方法，显示容器con中组件代号为s的组件：card.show(con,s);

也可以按组件加入容器的顺序显示组件：card.first(con)显示con中第一个组件；card.last(con)显示con中最后一个组件；card.next(con)显示当前正在被显示的组件的下一个组件；card.previous(con)显示当前正在被显示的组件的前一个组件。

例题 10.7

实现如图10‑15所示的界面。

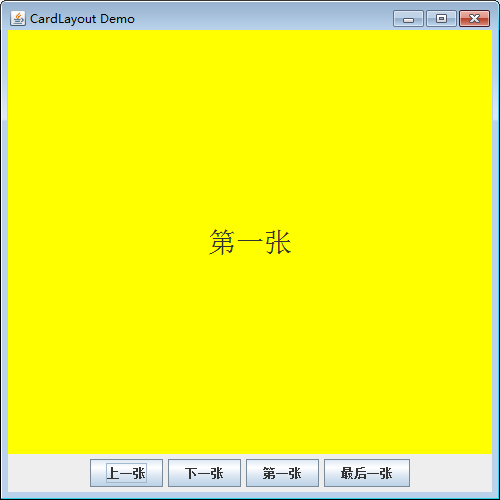


图10‑15 CardLayout布局

import javax.swing.\*;

import static javax.swing.JFrame.\*;

import java.awt.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class CardLayoutDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("CardLayout Demo",100,100,500,500);

JPanel panel1=new JPanel();

panel1.setLayout(new CardLayout());

panel1.setBackground(Color.yellow);

JPanel panel2=new JPanel();

myJFrame.add(panel1,BorderLayout.CENTER);

myJFrame.add(panel2,BorderLayout.SOUTH);

String[] names = {"第一张" , "第二张" , "第三张" , "第四张" , "第五张"};

JLabel label;

for (int i = 0 ; i < names.length ; i++){

label=new JLabel(names[i],SwingConstants.CENTER);

label.setFont(new Font("宋体",Font.PLAIN,28));

panel1.add(names[i],label);

}

JButton button1=new JButton("上一张");

JButton button2=new JButton("下一张");

JButton button3=new JButton("第一张");

JButton button4=new JButton("最后一张");

panel2.add(button1);

panel2.add(button2);

panel2.add(button3);

panel2.add(button4);

myJFrame.setVisible(true);

}

}

### GridLayout布局类

GridLayout布局类是java.awt包中的布局类。GridLayout布局策略是把容器划分成若干行乘若干列的网格区域，组件就位于这些划分出来的小格中。

假设一个容器con需要设置为3\*2的GridLayout布局格式，对窗口中的组件按照从左到右，从上到下的顺序排列。那么，con可以调用语句：

GridLayout grid=new GridLayout (3,2);

con.setLayout(grid);

或者

con.setLayout(new GridLayout(3,2));

例题 10.8

实现如图10‑16所示的界面。

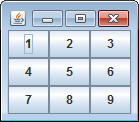


图10‑16 GridLayout布局

import javax.swing.\*;

import static javax.swing.JFrame.\*;

import java.awt.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class GridLayoutDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("GridLayout Demo",100,100,500,500);

myJFrame.setLayout(new GridLayout(3,3));

JButton button[]=new JButton[9];

for(int i=0;i<9;i++){

button[i]=new JButton(String.valueOf(i+1));

myJFrame.add(button[i]);

}

myJFrame.pack();

myJFrame.setVisible(true);

}

}

## 事件处理机制

### 事件处理

容器和组件完成了图形用户界面的绘制，下面的任务就是通过事件处理实现组件对用户操作的响应，即单击按钮、在文本框中按回车、在下拉列表框中选择一个条目、选择一个菜单等等操作后，程序需要作出什么样的处理。

下面从一个例子开始介绍事件处理机制。

例题 10.11

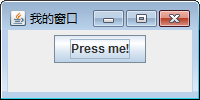


图10‑20 按钮事件处理

import javax.swing.\*;

import static javax.swing.JFrame.\*;

import java.awt.event.\*;

import java.awt.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setLayout(new FlowLayout());

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class ActionDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("我的窗口",100,100,200,100);

JButton button=new JButton("Press me!");

myJFrame.add(button);

JButtonHandler listener=new JButtonHandler();

button.addActionListener(listener);

myJFrame.setVisible(true);

}

}

class JButtonHandler implements ActionListener{

public void actionPerformed(ActionEvent e){

System.out.println("Action occurred");

}

}

程序运行后，点击按钮“Press me!”，在屏幕上输出“Action occurred”。

对上例进行分析：

(1) 事件源

产生事件的对象称为事件源，上例中按钮为事件源。

(2) 事件类型

* 文本框、按钮、菜单项、密码框和单选按钮都可以触发ActionEvent事件。
* 选择框、下拉列表都可以触发ItemEvent事件。
* 文本区内容发生变化可以触发DocumentEvent事件。
* 任何组件上都可以触发MouseEvent事件。
* 任何组件上都可以触发FocusEvent事件。
* 当按下、释放或敲击键盘上一个键时可以触发KeyEvent事件。

上例中按钮被点击，触发ActionEvent事件。

(3) 监听器

为了对事件源进行监听，需要一个监听器对象，它专门负责监听事件源的事件，并对发生的事件作出处理。上例中listener为监听器对象。

(4) 处理事件的接口

Java规定：为了让监视器能对事件源发生的事件进行处理，监听器对象所属的类必须声明实现相应的接口，那么当事件源发生事件时，监听器就自动调用被类重写的某个接口方法。

* 如果发生ActionEvent事件，监听器对象类必须声明实现ActionListener接口。
* 如果发生ItemEvent事件，监听器对象类必须声明实现ItemListener接口。
* 如果发生DocumentEvent事件，监听器对象类必须声明实现DocumentListener接口。
* 如果发生MouseEvent事件，监听器对象类必须声明实现MouseListener接口。
* 如果发生FocusEvent事件，监听器对象类必须声明实现FocusListener接口。
* 如果发生KeyEvent事件，监听器对象类必须声明实现KeyListener接口。

监听器对象类声明实现某个接口，就必须在类体中重写接口中所有方法。上例中监听器对象类JButtonHandler实现ActionListener接口，并重写其中的actionPerformed方法。

当按钮被点击，触发ActionEvent事件时，监听器对象执行actionPerformed方法，对发生的事件进行处理，此时产生的ActionEvent事件对象就会传递给该方法的参数e。

ActionEvent事件对象可以调用如下方法获取事件的基本信息，包括：

* Object getSource()获取ActionEvent事件的事件源对象的引用，即将事件源上转型为Object对象，并返回这个上转型对象的引用。
* String getActionCommand()获取ActionEvent事件发生时，和该事件相关的一个命令字符串。如文本框发生ActionEvent事件时，与事件相关的命令字符串就是文本框中的文本字符串。

(5) 程序执行过程

由于按钮“Press me!”为事件源，其上通过addActionListener方法添加了监听器对象listener。

因此，当按钮被点击时，产生了ActionEvent事件，这时其上的监听器对象listener监听到该事件，并执行监听器对象类中的actionPerformed方法，在屏幕上输出“Action occurred”。

我们发现，上例中点击按钮后在命令行输出结果，这似乎不符合GUI设计的理念，用户希望在窗口的某个组件，如文本区看到输出结果。于是，对上例进行修改。

例题 10.12

import javax.swing.\*;

import static javax.swing.JFrame.\*;

import java.awt.event.\*;

import java.awt.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setLayout(new FlowLayout());

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class ActionGUIDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("我的窗口",100,100,400,300);

JButton button=new JButton("Press me!");

myJFrame.add(button);

**JTextArea textArea=new JTextArea(9,30);**

**myJFrame.add(new JScrollPane(textArea));**

JButtonHandler listener=new JButtonHandler();

button.addActionListener(listener);

myJFrame.setVisible(true);

}

}

class JButtonHandler implements ActionListener{

public void actionPerformed(ActionEvent e){

System.out.println("Action occurred");

}

}

我们发现，虽然在窗口中添加了文本区，但是监听器对象监听到ActionEvent事件后，执行的actionPerformed方法并不能获取到文本区。因此还不能在文本区输出结果。

如何在actionPerformed方法中引用到文本区对象呢？有多种方法来解决该问题。其中一个比较好的方法就是，将与事件有关的组件，如文本区，组合到监听器对象类中去。继续对上例进行修改。

例题 10.13

import javax.swing.\*;

import static javax.swing.JFrame.\*;

import java.awt.event.\*;

import java.awt.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setLayout(new FlowLayout());

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class ActionGUIDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("我的窗口",100,100,400,300);

JButton button=new JButton("Press me!");

myJFrame.add(button);

JTextArea textArea=new JTextArea(9,30);

myJFrame.add(new JScrollPane(textArea));

**JButtonHandler listener=new JButtonHandler();**

**listener.setJTextArea(textArea);**

**button.addActionListener(listener);**

myJFrame.setVisible(true);

}

}

class JButtonHandler implements ActionListener{

JTextArea textshow;

**public void setJTextArea(JTextArea area){**

**textshow=area;**

**}**

public void actionPerformed(ActionEvent e){

textshow.append("Action occurred\n");

}

}

执行程序，效果如图10‑21所示。

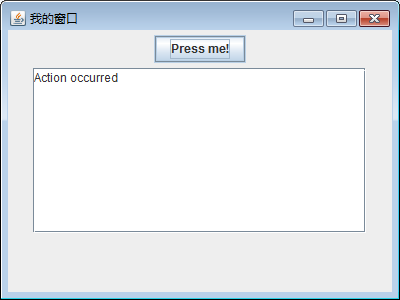


图10‑21 改进的按钮事件处理

上例中点击按钮“Press me!”后，在文本区输出了“Action occurred”，符合GUI设计的理念。

通过将与事件处理有关的组件组合到监听器对象类中，从而使得监听器对象可以引用组件对象。上例中通过设置setJTextArea方法将文本区textArea组合到监听器对象类JButtonHandler中，这样，actionPerformed方法就可以引用文本区textArea了，并在文本区textArea中追加文本。

另外，在上例中只有一个按钮“Press me!”，如果有多个按钮，则需要判断是哪个按钮触发了ActionEvent事件，因此进一步修改上例。

例题 10.14

import javax.swing.\*;

import static javax.swing.JFrame.\*;

import java.awt.event.\*;

import java.awt.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setLayout(new FlowLayout());

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class ActionGUIDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("我的窗口",100,100,400,300);

JButton button=new JButton("Press me!");

myJFrame.add(button);

JTextArea textArea=new JTextArea(9,30);

myJFrame.add(new JScrollPane(textArea));

JButtonHandler listener=new JButtonHandler();

listener.setJTextArea(textArea);

**listener.setJButton(button);**

button.addActionListener(listener);

myJFrame.setVisible(true);

}

}

class JButtonHandler implements ActionListener{

JButton button;

JTextArea textshow;

public void setJTextArea(JTextArea area){

textshow=area;

}

**public void setJButton(JButton b){**

**button=b;**

**}**

public void actionPerformed(ActionEvent e){

**if(e.getSource().equals(button)){**

textshow.append("Action occurred\n");

}

}

}

在actionPerformed方法中，首先判断ActionEvent事件源是不是“Press me!”，如果是，则在文本区textshow中追加输出结果。

### 其他事件类型

上一节曾经介绍过事件的类型，包括：

* ActionEvent事件
* ItemEvent事件
* DocumentEvent事件
* MouseEvent事件
* FocusEvent事件
* KeyEvent事件

**1、ItemEvent事件**

(1) 事件源

选择框、下拉列表框都可以触发ItemEvent事件。

选择框提供两种状态，一种是选中，另一种是未选中。对于注册了监听器的选择框，当用户的操作使得选择框从未选中状态变成选中状态或从选中状态变成未选中状态时就触发ItemEvent事件。

对于注册了监听器的下拉列表框，如果用户选中下拉列表中的某个选项，就会触发ItemEvent事件。

(2) 注册监听器

事件源通过使用addItemListener方法，注册监听器对象。

(3) 监听器对象类

监听器对象类需实现ItemListener接口，并实现其中的1个方法：

public void itemStateChanged(ItemEvent e)

事件源产生ItemEvent事件后，监听器对象执行itemStateChanged方法对事件进行处理，此时产生的ItemEvent事件对象会传递给该方法的参数e。

ItemEvent事件对象除了可以使用getSource方法获取事件源外，还可以使用getSelectedItem()方法获取当前所选项。

例题 10.15

下拉列表中的选项是当前目录下Java文件的名字，用户选择下拉列表的选项后，监听器负责在文本区显示文件的内容。程序运行效果如图10‑22所示。

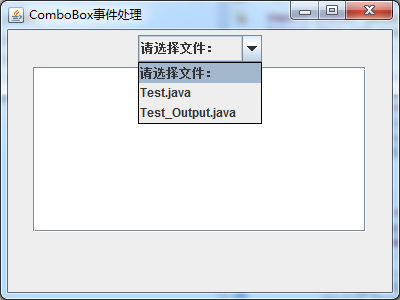


图10‑22 ComboBox事件处理

import javax.swing.\*;

import static javax.swing.JFrame.\*;

import java.awt.event.\*;

import java.awt.\*;

import java.io.\*;

class MyJFrame extends JFrame{

MyJFrame(String s,int x,int y,int w,int h){

setLayout(new FlowLayout());

setTitle(s);

setBounds(x,y,w,h);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

public class ItemEventDemo{

public static void main(String args[]){

MyJFrame myJFrame=new MyJFrame("ComboBox事件处理",100,100,400,300);

File dir=new File(".");

FileAccept fileAccept=new FileAccept();

fileAccept.setExtendName("java");

String[] fileName=dir.list(fileAccept);

String itemList[]=new String[fileName.length+1];

itemList[0]="请选择文件：";

for(int i=0;i<fileName.length;i++){

itemList[i+1]=fileName[i];

}

JComboBox jcb=new JComboBox(itemList);

myJFrame.add(jcb);

JTextArea textArea=new JTextArea(9,30);

myJFrame.add(new JScrollPane(textArea));

JComboBoxHandler listener=new JComboBoxHandler();

listener.setJTextArea(textArea);

listener.setJComboBox(jcb);

jcb.addItemListener(listener);

myJFrame.setVisible(true);

}

}

class FileAccept implements FilenameFilter{

private String extendName;

public void setExtendName(String s){

extendName="."+s;

}

public boolean accept(File dir,String name){

return name.endsWith(extendName);

}

}

class JComboBoxHandler implements ItemListener{

JComboBox choice;

JTextArea textShow;

public void setJComboBox(JComboBox jcb) {

choice = jcb;

}

public void setJTextArea(JTextArea area) {

textShow = area;

}

public void itemStateChanged(ItemEvent e) {

textShow.setText(null);

String fileName=choice.getSelectedItem().toString();

try(FileReader inOne = new FileReader(fileName);

BufferedReader inTwo = new BufferedReader(inOne)){

String s=null;

while((s=inTwo.readLine())!=null) {

textShow.append(s+"\n");

}

}catch(Exception ee) {

textShow.append(ee.toString());

}

}

}

### 事件处理机制

1、授权模式

Java的事件处理是基于授权模式的，组件将事件的处理委托给监听器来处理，组件只负责绘制图形，监听器对象负责处理事件。

2、接口回调

Java语言使用接口回调技术实现处理事件的过程。

addXXXListener(XXXListener listener)

方法中的参数是接口，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

从方法绑定角度看，Java将某种事件的处理绑定到对应的接口，即绑定到接口中的方法，也就是说，当事件源触发事件后，监听器准确知道去调用哪个方法。

4、保持松耦合

监听器和事件源应该保持一种松耦合关系，也就是说尽量让事件源所在的类和监听器是组合关系。也就是说，当事件源触发事件后，系统知道某个方法会被执行，但无需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(作为监听器)都可以调用这个方法来处理事件。