

程序员书库

初学者的入门宝典，程序员的百科全书



CD-ROM

10小时多媒体视频讲解

#### 本书特色

- ※ 起点低，即使没有任何编程经验，也能通过本书掌握Java
- ※ 避免大段理论讲解，而是通过大量实例进行讲解，有很强的实践性
- ※ 对代码进行了详细注释，阅读起来非常容易，没有任何障碍
- ※ 通过现实中的事物类比Java中的概念，使读者可以很容易理解
- ※ 重点讲解Java语言的基础知识和应用，并对一些设计模式也有所介绍
- ※ 全书提供190个实例和2个综合案例，非常实用

# Java

## 从入门到精通

高宏静 等编著



化学工业出版社

## 第 11 章 图形编程

在进入本章之前，所有运行的应用程序都是命令行界面（非图形界面）。从本章开始，就不在受限于只能创建命令行应用程序，读者可以创建图形界面程序。图形编程内容主要包括 AWT（Abstract Windowing Toolkit，抽象窗口工具集）和 Swing 两个内容。AWT 是用来创建 Java 图形用户界面的基本工具，Java Swing 是 JFC（Java Foundation Classes）的一部分，它可以弥补 AWT 的一些不足。

### 11.1 AWT 简介

AWT 是抽象窗口工具集（Abstract Window Toolkit）的英文缩写，Java 的 AWT 类库内容相当丰富，一共有 60 多个类和接口，包括了创建 Java 图形界面程序的所有工具。利用 AWT 类库，程序员可以在 Applet 的显示区域创建标签、按钮、复选框、文本域以及其他丰富的用户界面元素，还可针对用户的行为做出相应响应。

AWT 是 JDK 的一部分，在开发图形应用程序和 Applet 小程序时，一般都要用到它。AWT 为用户界面程序提供所需要的组件，例如按钮、标签、复选框、下拉菜单、画布、文本输入区、列表等。此外，AWT 提供事件类、监听器类、图形处理工具、2D 图形等的支持。表 11-1 列出了 AWT 中的 Java 软件包。

表 11-1 AWT 中的 Java 软件包

AWT 软件包	描述	AWT 软件包	描述
java.awt	基本组件实用工具	java.awt.geom	2D API 几何软件包
java.awt.accessibility	辅助技术	java.awt.im	引入方法
java.awt.color	颜色和颜色空间	java.awt.image	图像处理工具包
java.awt.datatransfer	支持剪贴板和数据传输	java.awt.peer	同位体组件、界面包
java.awt.dnd	拖放	java.awt.print	支持打印 2D
java.awt.event	事件类和监听者	java.awt.swing	Swing 组件
java.awt.font	2D API 字体软件包	java.awt.test	测试 AWT 函数有限子集的独立 applet

Java 抽象窗口工具集有 4 个主要的类用于确定容器内组件的位置和形状，包括组件（Component）、容器（Container）、布局管理器（LayoutManager）和图形（Graphics）。

- ❑ Component（组件）：按钮、标签、菜单等组件的抽象基本类。
- ❑ Container（容器）：扩展组件的抽象基本类，例如 Panel、Applet、Window、Dialog 和 Frame 等是由 Container 演变的类，容器中可以包括多个组件。
- ❑ LayoutManager（布局管理器）：定义容器中组件的摆放位置和大小接口。Java 中定

义了几种默认的布局管理器。

- ❑ Graphics（图形）：组件内与图形处理相关的类，每个组件都包含一个图形类的对象。

## 11.2 组件和容器

在抽象窗口工具集中，组件、容器和布局管理器是图形编程的基础。本节主要讲述 AWT 组件的基本概念和种类，并详细介绍容器的概念、种类和层次结构，最后介绍 Frame 和 Panel 的类继承关系，并举例说明两个容器的使用。

### 11.2.1 组件

组件是 Java 图形用户界面程序设计的最基本组成部分，它是一个以图形方式显示的，并且可以与用户进行交互的界面组成元素，例如按钮、标签、单选框、多选框等。单独的一个组件不能显示出来，必须将组件添加到容器当中才能显示。表 11-2 列出了从 java.awt.Component 类演变产生的 AWT 组件。

表 11-2 AWT组件

组件	超类	描述
Button	Component	触发行为的文本按钮
Canvas	Component	绘制图形的画布
Checkbox	Component	可检验的布尔组件
Choice	Component	文本软件的弹出菜单
Dialog	Window	可模式化窗口
FileDialog	Dialog	选择文件的相关平台对话框
Frame	Window	具有标题栏和可选菜单的顶层窗口
Label	Component	显示字符串的组件
List	Component	文本输入的可滚动列表
Panel	Container	一般组件容器
Scrollbar	Component	滚动项目的 adjustable 组件
ScrollPane	Container	可滚动容器
Textarea	TextComponent	多行可滚动的文本框
TextComponent	Component	TextArea 和 TextField 的基本功能
TextField	TextComponent	输入文本的单行组件
Window	Container	没有标题的无边界窗口

AWT 的所有组件中，许多都是 java.awt.Component 的子类。作为父类的 Component 封装了所有组件最基本的属性和方法，例如组件对象的大小、显示位置、前景色、边界属性以及

可见性等，同时这些方法也被扩展到它的派生类组件中。

- ❑ **Component** `getComponentAt(int x, int y)`: 返回包含该位置(x,y)的组件或子组件。
- ❑ **Font** `getFont()`: 获取组件的字体。
- ❑ **Color** `getForeground()`: 获取组件的前景色。
- ❑ **String** `getName()`: 获取组件的名称。
- ❑ **Dimension** `getSize()`: 以 **Dimension** 对象的形式返回组件的大小。
- ❑ **void** `paint(Graphics g)`: 绘制该组件。
- ❑ **void** `repaint()`: 重绘组件。
- ❑ **void** `update(Graphics g)`: 更新该组件。
- ❑ **void** `setVisible(boolean b)`: 根据参数的值显示或隐藏此组件。
- ❑ **void** `setSize(int width, int height)`: 调整组件的大小，使其宽度和高度分别为 **width** 和 **height**。
- ❑ **void** `setName(String name)`: 将组件的名称设置为指定的字符串。
- ❑ **void** `setForeground(Color c)`: 设置组件的前景色。

### 11.2.2 容器

Container 也是一个派生于容器 Component 的抽象类, 因此它拥有组件的所有属性和方法。

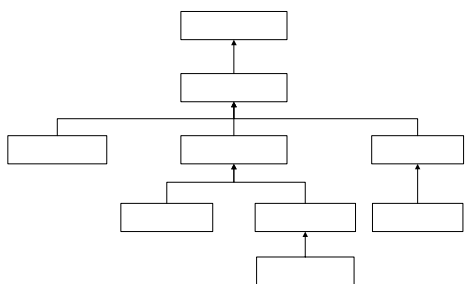


图 11-1 AWT 容器的层次结构

容器最主要的功能是存放其他的组件和容器。一个容器可以存放多个组件，它将相关的组件容纳到一个容器中形成一个整体。使用容器存放组件的技术可以简化组件显示安排。所有的容器都可以通过 `add()` 方法添加组件。

注意：Applet 类是 Panel 类的子类，因此所有的 Applet 都继承了包含组件的能力。

图 11-1 是 AWT 的容器层次结构图。其中两个最常用的容器是窗口（Frame）和面板（Panel）。

容器类常用的方法有以下几种。

- ❑ `void add(Component com)`: 将指定组件追加到此容器的尾部。
- ❑ `void add(Component, int index)`: 将指定组件添加到此容器的给定位置上。
- ❑ `void add(Component component, Object constraints)`: 将指定组件添加到此容器的尾部。
- ❑ `void add(String name, Component component)`: 将指定组件添加到此容器中。
- ❑ `void add(Component component, Object constraints, int index)`: 使用指定约束, 将指定组件添加到此容器指定索引所在的位置上。
- ❑ `void setLayout()` and `getLayout()`: 设置此容器的布局管理器。
- ❑ `void remove(Component comp)`: 从此容器中移除指定组件。
- ❑ `void remove(int index)`: 从此容器中移除 `index` 指定的组件。

❑ void removeAll(): 从此容器中移除所有组件。

### 11.2.3 窗口

窗口 (Frame) 是 Window 的子类, 它是顶级窗口容器, 可以添加组件、设置布局管理器、设置背景色等。

```
java.lang.Object
+--java.awt.Component
  +--java.awt.Container
    +--java.awt.Window
      +--java.awt.Frame
```

下面是一个窗口应用的例子。

```
// 文件: 程序 11.1    FirstFrame.java    描述: 第一个 Frame 演示
//导入使用的包和类
import java.awt.Color;
import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class FirstFrame extends Frame {
    public FirstFrame(String string) {
        super(string); //调用父类的构造方法
    }
    public static void main(String args[]){
        FirstFrame f = new FirstFrame("这是第一个 Frame");    //构造方法调用
        f.setSize(300, 200);    //设置 Frame 的大小, 缺省为(0,0);
        f.setBackground(Color.BLUE);    //设置 Frame 的背景, 缺省为红色
        f.setVisible(true);    //设置 Frame 为可见, 缺省为不可见
        //这里可以不理睬, 添加事件处理方法
        f.addWindowListener(f. new Mywindowadapter());//为窗口 f 添加监听器
    }
    //下面这段是为了实现窗口的关闭功能, 可以不管, 后面有详细内容
    class Mywindowadapter extends WindowAdapter
    {
        public void windowClosing(WindowEvent we){ //覆盖 WindowAdapter 方法
            System.exit(0);    //程序退出
        }
    };
}
```

编写完程序后, 使用 javac 命令编译该文件产生 class 文件, 然后使用 java 命令运行该 class 文件, 运行结果如图 11-2 所示。

程序 11.1 通过顶级窗口容器进行实例化, 用方法 setName 设置窗口名字, setBackground 设置背景色, 最后必须调用 setVisible 显示窗口。

通常情况下, 生成一个窗口要使用 Window 的派生类窗口实例化, 而非直接使用 Window 类。窗口的外观界面和通常情况下在 Windows 系统下的窗口相似, 可以设置标题名

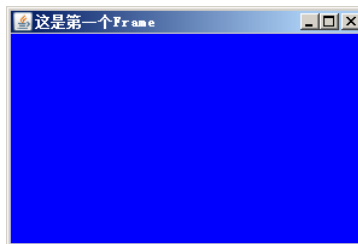


图 11-2 FirstFrame.java 运行结果

称、边框、菜单栏以及窗口大小等。窗口对象实例化后都是大小为零并且默认是不可见的，因此在程序中必须调用 `setSize()` 设置大小，调用 `setVisible(true)` 来设置该窗口为可见。

注意：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，显示的窗口是 Windows 风格的窗口；而在 UNIX 下运行时，显示的则是 UNIX 风格的窗口。

## 11.2.4 面板

面板（Panel）是容器的一个子类，它提供了建立应用程序的容器。可以在一个面板上进行图形处理，并把这个容器添加到其他容器中（例如 `Frame`、`Applet`）。后面会单独介绍 `Applet`（一种特殊的 `Panel`）。类的关系图如下所示。

```
java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--java.awt.Panel
```

下面是一个演示使用面板的例子。

```
// 文件：程序 11.2    FrameaddedPanel.java    描述：在 Frame 中添加 Panel
//导入相关包和类
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class FrameaddedPanel extends Frame{
    //声明 FrameaddedPanel 构造方法
    public FrameaddedPanel(String str){
        super(str);          //调用父类构造方法
    }
    public static void main(String args[] ){
        //创建，并初始化 FrameaddedPanel 对象 f
        FrameaddedPanel f = new FrameaddedPanel("在 Frame 中添加 Panel");
        Panel p=new Panel();    //创建，并初始化 Panel 对象 p
        f.setSize(400,300);      //设置窗口大小
        f.setBackground(Color.blue); //框架 fr 的背景颜色设置为蓝色
        f.setLayout(null);       //取消布局管理器
        p.setSize(200,200);      //设置面板 p 的大小
        p.setBackground(Color.red); //设置面板 p 的颜色为红色
        f.add(p);                //用 add 方法把面板 pan 添加到框架 fr 中
        f.setVisible(true);      //显示窗口 f
        //这里可以不理睬，添加事件处理方法
        f.addWindowListener(f.new Mywindowadapter());
    }
    //下面这段是为了实现窗口的关闭功能，可以不管，后面有详细内容
    class Mywindowadapter extends WindowAdapter
    {
        public void windowClosing(WindowEvent we){ //覆盖 WindowAdapter 的方法
            System.exit(0);          //程序终止
        }
    };
}
```

编写完程序后，使用 `javac` 命令编译该文件产生 `class` 文件，然后使用 `java` 命令运行该 `class` 文件，运行结果如图 11-3 所示。

程序 11.2 中，类 `FrameaddedPanel` 是窗口 `Frame` 的子类，`f` 是 `FrameaddedPanel` 的实例化对象。在 `f` 中添加一个面板对象，最后通过 `add()` 方法添加到窗口中。

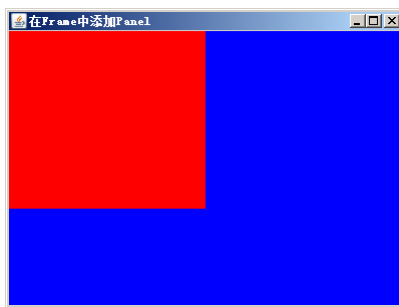


图 11-3 FirstFrame.java 运行结果

## 11.3 布局管理器

为了实现容器中跨平台的特性、组件的大小改变、位置转移等动态特性，Java 提供了布局管理器容器（`LayoutManager`）处理机制。布局管理器可以实现容器内部组件的排列顺序、大小、位置以及窗口大小变化。

### 11.3.1 布局管理器的分类

每一个容器中保存着一个布局管理器的引用，该布局管理器可以完成容器内组件的布局 and 整形。每发生一个可以引起容器重新布置内部组件的事件时，容器会自动调用布局管理器布置容器内部的组件。布局管理器有多个种类，不同的布局管理器使用不同算法和布局策略，并且容器可以选择不同的布局管理进行布局。AWT 提供了 5 种类型的布局管理器。

- ❑ `BorderLayout`（边界布局管理器）：该管理器将容器分东、南、西、北、中 5 个区域，当向容器添加组件时，必须指明 `BorderLayout` 将组件放置的区域。
- ❑ `CardLayout`（卡片布局管理器）：该布局管理器将加入到容器中的组件视为卡片栈，把每个组件放置在一个单独的卡片上，而每次只能看见一张卡片。
- ❑ `FlowLayout`（顺序布局管理器）：该布局管理器将组件从左到右、从上到下放置。
- ❑ `GridLayout`（网格布局管理器）：该布局管理器将容器分成相同尺寸的网格，将组件从左到右、从上到下放置在网格中。
- ❑ `GridBagLayout`（网络包布局管理器）：与网格布局管理器不同的是，一个组件不止占一个网格位置，因此在加入组件时，必须指明一个对应的参数。

### 11.3.2 顺序布局管理器

顺序布局管理器（`FlowLayout`）是 `Panel` 和 `Applet` 缺省的布局管理器。添加组件的放置规律是从上到下、从左到右，也就是说，添加组件时，先放置在第一行的左边，依次放满第一行，然后在开始放置第二行，依此类推。构造方法主要有以下几种。

- ❑ `FlowLayout(FlowLayout.RIGHT,20,40)`：第一个参数是组件的对齐模式，包括左右中 3 种；第二个参数是组件行间隔；第三个参数是组件列间隔，单位是像素。
- ❑ `FlowLayout(FlowLayout.LEFT)`：居左对齐，行间隔和列间隔默认为 5 个像素。

□ **FlowLayout():** 默认是居中对齐, 并且行、列间隔默认为 5 像素。  
下面是一个顺序布局管理器应用的例子。

```
// 文件: 程序 11.3    FlowLayoutDemo.java    描述: FlowLayout 顺序布局
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class FlowLayoutDemo{
    //声明 FlowLayoutDemo 构造方法
    public FlowLayoutDemo()
    {
        b1 = new Button("继续");    //初始化 Button 变量 b1
        b2 = new Button("取消");    //初始化 Button 变量 b2
        b3 = new Button("确定");    //初始化 Button 变量 b3
    }
    public static void main(String args[ ])
    {
        FlowLayoutDemo fl = new FlowLayoutDemo(); //创建, 并初始化 FlowLayoutDemo 对象 fl
        fl.show();                                //调用 show 方法
    }
    public void show()
    {
        f = new Frame("FlowLayout 顺序布局");    //初始化对象 f
        f.setSize(300, 240);                      //设置窗口 f 的大小
        //设置布局管理器为 FlowLayout
        f.setLayout(new FlowLayout(FlowLayout.CENTER,30,20));
        f.add(b1);                                //在窗口中添加按钮 b1
        f.add(b2);                                //在窗口中添加按钮 b2
        f.add(b3);                                //在窗口中添加按钮 b3
        //为窗口 f 添加 WindowListener 监听器
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {    //实现 windowClosing 方法
                f.setVisible(false);    //设置窗口 f 不可见
                f.dispose();            //释放窗口及其子组件的屏幕资源
                System.exit(0);        //退出程序
            }
        });
        //紧凑排列, 其作用相当于 setSize(), 即让窗口尽量小, 小到刚刚能够包容住 b1、b2 两个按钮
        //f.pack();
        f.setVisible(true);                //设置窗口 f 可视
    }
    private Frame f;                        //声明 Frame 类型数据域 f
    private Button b1,b2,b3;                //声明 Button 类型的数据域 b1, b2, b3
}
```

编写完程序后, 使用 `javac` 命令编译该文件产生 `class` 文件, 然后使用命令运行该 `class` 文件, 运行结果如图 11-4 所示。如果我们不想使用在上面程序中设置的窗口的大小, 想变再小一点, 小到刚刚能够包容住 `b1`、`b2`、`b3` 三个按钮, 我们可以将鼠标放到窗口的边缘, 待变成双箭头状, 按住鼠标左键不放, 拖拽到我们想要的效果即可, 如图 11-5 所示。

程序 11.3 中, 实例化三个按钮对象, 设置 `Frame` 的布局管理器为顺序布局。从运行结果可以看出, 三个按钮按照顺序添加在窗口中, 顺序是从左到右、从上到下。





图 11-4 FlowLayoutDemo.java 运行结果 1



图 11-5 FlowLayoutDemo.java 运行结果 2

如果改变窗口的宽，通过 **FlowLayout** 布局管理器管理的组件的放置位置会随之发生变化，其变化规律是：组件的大小不变，而组件的位置会根据容器的大小进行调节。如图 11-4 所示，3 个按钮都处于同一行，最后窗口变窄到在一行只能放置一个按钮，原来处于一行的按钮分别移动到第二行和第三行。可以看出程序中安排组件的位置和大小时，具有以下特点。

- ❑ 容器中组件的大小和位置都委托给布局管理器管理，程序员无法设置这些属性。如果已经设置布局管理器在容器中，使用 Java 语言提供的 `setLocation()`、`setSize()`、`setBounds()`等方法不会起到任何作用。
- ❑ 如果用户必须设置组件的大小和位置，必须设置容器布局管理器为空，方法为：`setLayout(null)`。

### 11.3.3 边界布局管理器

边界布局管理器（**BorderLayout**）是 **Window**、**Frame** 和 **Dialog** 的缺省布局管理器。边界布局管理器将容器分成 5 个区：北（**N**）、南（**S**）、西（**W**）、东（**E**）和中（**C**），每个区域只能放置一个组件。各个区域的位置安排如图 11-6 所示。

下面是一个演示边界布局管理器的例子。

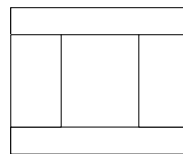


图 11-6 BorderLayout 布局

```
// 文件：程序 11.4    BorderLayoutDemo.java    描述：BorderLayout 管理器布局
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class BorderLayoutDemo{
    //声明 BorderLayoutDemo 构造方法
    public BorderLayoutDemo()
    {
        b1 = new Button("上北");           //初始化按钮 b1
        b2 = new Button("下南");           //初始化按钮 b2
        b3 = new Button("左西");           //初始化按钮 b3
        b4 = new Button("右东");           //初始化按钮 b4
        b5 = new Button("中间");           //初始化按钮 b5
    }
    public static void main(String args[ ])
}
```

```

{
    BorderLayoutDemo fl = new BorderLayoutDemo(); //创建，并初始化 BorderLayoutDemo 对象 fl
    fl.show(); //调用 show 方法
}
public void show()
{
    f = new Frame("BorderLayout 布局演示"); //创建，并初始化数据域 f
    f.setSize(400, 300); //设置窗口 f 的大小
    f.setLayout(new BorderLayout()); //设置布局管理器为 BorderLayout
    f.add(BorderLayout.NORTH, b1); //将 b5 添加到 NORTH 位置
    f.add(BorderLayout.SOUTH, b2); //将 b5 添加到 SOUTH 位置
    f.add(BorderLayout.WEST, b3); //将 b5 添加到 WEST 位置
    f.add(BorderLayout.EAST, b4); //将 b5 添加到 EAST 位置
    f.add(BorderLayout.CENTER, b5); //将 b5 添加到 CENTER 位置
    f.addWindowListener(new WindowAdapter(){ //添加监听器
        public void windowClosing(WindowEvent evt) { //实现 windowClosing 方法
            f.setVisible(false); //设置窗口 f 不可见
            f.dispose(); //释放窗口及其子组件的屏幕资源
            System.exit(0); //退出程序
        }
    });
    //紧凑排列，其作用相当于 setSize()，即让窗口尽量小，小到刚刚能够包容住 b1、b2 两个按钮
    f.pack();
    f.setVisible(true); //显示窗口 f
}
private Frame f; //声明 Frame 类型数据域 f
private Button b1,b2,b3,b4,b5; //声明 Button 类型数据域 b1,b2,b3,b4,b5
}

```

编写完程序后，使用 javac 命令编译该文件产生 class 文件，然后使用 java 命令运行该 class 文件，运行结果如图 11-7 所示。



图 11-7 BorderLayoutDemo.java 运行结果

从程序 11.4 可以看出，程序中放置了 5 个按钮，边界布局管理器可以分为 5 个区域，并且在每一个区域只能放置一个组件。

注意：在使用边界布局管理器时，如果容器大小发生变化，内部组件的变化规律如下：组件大小会变化，相对位置不变。另外，容器 5 个区域并没有要求必须添加组件，如果中间区域没有组件，则中间区域将会保留空白；如果四周的区域没有组件，中间区域将会补充。

### 11.3.4 网格布局管理器

网格布局管理器（GridLayout）使容器中各个组件呈网格状分布，并且每一个网格的大小一致。其构造方法有以下几种。

- ❑ public GridLayout(): 默认网格布局管理器只占据一行一列。
- ❑ public GridLayout(int row,int col): 创建指定行数和列数的网格布局管理器，组件分

配大小是平均的。但是行和列不能同时为零，其中一个为零时，只是表示所有的组件都放置于一行或者一列中。

- `public GridLayout(int row,int col,int horz,int vert)`: 创建指定行数和列数的网格布局管理器，组件分配大小是平均的。

下面是一个网格布局的例子。

```
// 文件: 程序 11.5    GridLayoutDemo.java    描述: GridLayout 管理器布局
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class GridLayoutDemo{
    public GridLayoutDemo()
    {
        b1 = new Button("[0][0]");    //初始化按钮 b1
        b2 = new Button("[0][1]");    //初始化按钮 b2
        b3 = new Button("[0][2]");    //初始化按钮 b3
        b4 = new Button("[1][0]");    //初始化按钮 b4
        b5 = new Button("[1][1]");    //初始化按钮 b5
        b6 = new Button("[1][2]");    //初始化按钮 b6
    }
    public static void main(String args[ ])
    {
        GridLayoutDemo fl = new GridLayoutDemo(); //创建，并初始化 GridLayoutDemo 对象 fl
        fl.show();                                //调用 show 方法
    }
    public void show()
    {
        f = new Frame("GridLayout 布局演示");    //初始化窗口 f
        f.setSize(400, 300);                    //设置 f 的大小
        //设置布局管理器为 GridLayout
        f.setLayout(new GridLayout(2,3));
        f.add(b1);                                //添加阵列中的[0][0]位置
        f.add(b2);                                //添加阵列中的[0][1]位置
        f.add(b3);                                //添加阵列中的[0][2]位置
        f.add(b4);                                //添加阵列中的[1][0]位置
        f.add(b5);                                //添加阵列中的[1][1]位置
        f.add(b6);                                //添加阵列中的[1][2]位置
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {    //实现 windowClosing 方法
                f.setVisible(false);                    //设置窗口 f 不可见
                f.dispose();                            //释放窗口及其子组件的屏幕资源
                System.exit(0);                        //退出程序
            }
        });
        f.setVisible(true);                        //显示窗口
    }
    private Frame f;                                //声明 Frame 类型数据域 f
    private Button b1,b2,b3,b4,b5,b6;            //声明 Button 类型数据域 b1,b2,b3,b4,b5,b6
}
```

编写完程序后,使用 `javac` 命令编译该文件产生 `class` 文件,然后使用 `java` 命令运行该 `class` 文件,运行结果如图 11-8 所示。



图 11-8 GridLayoutDemo.java 运行结果

程序 11.5 设置窗口的布局管理器为网格布局管理器，网格的行数是 2，列数为 3。可以在网格中添加 6 个组件，放置顺序是从左到右、从上到下。

### 11.3.5 卡片布局管理器

卡片布局管理器（CardLayout）将每一个组件视为一张卡片，一次只能看到一张卡片，容器充当卡片的堆栈，容器第一次显示的是第一次添加的组件。构造方法有以下几种。

- ❑ `public CardLayout()`: 创建一个新卡片的布局，水平间距和垂直间距都是 0。
- ❑ `public CardLayout(int hgap,int vgap)`: 创建一个具有指定水平间距和垂直间距的新卡片布局。

还有一些比较重要的方法，如下所示。

- ❑ `void first(Container parent)`: 翻转到容器的第一张卡片。
- ❑ `void next(Container parent)`: 翻转到指定容器的下一张卡片。
- ❑ `void last(Container parent)`: 翻转到容器的最后一张卡片。
- ❑ `void previous(Container parent)`: 翻转到指定容器的前一张卡片。

下面是一个卡片布局应用的例子。

```
//文件：程序 11.6    CardLayoutDemo.java           : CardLayout 管理器布局演示
import java.awt.*;
import java.awt.event.*;
public class CardLayoutDemo extends Frame implements MouseListener{
    //声明 CardLayoutDemo 带有字符串参数的构造方法
    public CardLayoutDemo(String string) {
        super(string);           //调用父类构造方法
        init();                  //调用方法 init()
    }
    public static void main(String args[ ]){
        new CardLayoutDemo("CardLayout1"); //创建 CardLayoutDemo 类型变量
    }
    public void init()
    {
        setLayout(new BorderLayout()); //设置窗口的布局管理器为 BorderLayout
        setSize(400,300);             //设置窗口的大小
        Panel p = new Panel();         //创建，并初始化面板 Panel 对象 p
    }
}
```

```

p.setLayout(new FlowLayout()); //设置面板 p 的布局管理器为 FlowLayout
first.addMouseListener(this); //为 first 按钮添加鼠标监听器
second.addMouseListener(this); //为 second 按钮添加鼠标监听器
third.addMouseListener(this); //为 third 按钮添加鼠标监听器
p.add(first); //在面板 p 中添加按钮 first
p.add(second); //在面板 p 中添加按钮 second
p.add(third); //在面板 p 中添加按钮 third
add("North", p); //在窗口中添加面板 p
cards.setLayout(cl); //设置 panel 为卡片布局器
cards.add("First card", new Button("第一页内容")); //在 cards 中添加按钮
cards.add("Second card", new Button("第二页内容")); //在 cards 中添加按钮
cards.add("Third card", new Button("第三页内容")); //在 cards 中添加按钮
add("Center", cards); //将 cards 添加到窗口的 Center 位置
//注册监听器，关闭功能
addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent evt) { //实现 windowClosing 方法
        f.setVisible(false); //设置窗口 f 不可见
        f.dispose(); //释放窗口及其子组件的屏幕资源
        System.exit(0); //退出程序
    }
});
setVisible(true); //显示窗口
}
//实现监听器方法
public void mouseClicked(MouseEvent evt) {
    if (evt.getSource() == first) { //当事件源为 first 时
        cl.first(cards); //翻转到第一张卡片
    }
    else if (evt.getSource() == second) { //当事件源为 second 时
        cl.first(cards); //翻转到第一张卡片
        cl.next(cards); //翻转到下一张卡片
    }
    else if (evt.getSource() == third) { //当事件源为 third 时
        cl.last(cards); //翻转到最后一张卡片
    }
}
public void mouseEntered(MouseEvent arg0) { //mouseEntered 为空方法
}
public void mouseExited(MouseEvent arg0) { //mouseExited 为空方法
}
public void mousePressed(MouseEvent arg0) { //mousePressed 为空方法
}
public void mouseReleased(MouseEvent arg0) { //mouseReleased 为空方法
}
private Button first = new Button("第一页"); //声明，并初始化按钮 first
private Button second = new Button("第二页"); //声明，并初始化按钮 second
private Button third = new Button("第三页"); //声明，并初始化按钮 third
private Panel cards = new Panel(); //声明，并初始化面板 cards
private CardLayout cl = new CardLayout(); //实例化一个卡片布局对象
}

```

编写完程序后，使用 javac 命令编译该文件产生 class 文件，然后使用 java 命令运行该 class 文件，运行结果如图 11-9 所示。



图 11-9 CardLayoutDemo.java 运行结果

程序 11.6 设置窗口的布局管理器为卡片布局管理器，并为窗口类实现 `MouseListener` 接口（下一章详细讲述相关内容），重写了 `mouseClicked` 方法，当单击按钮“第一页”显示第一个 `Button`，单击按钮“第二页”显示第二个 `Button`，单击按钮“第三页”显示最后一个 `Button`。

注意：在程序中，由于经常操作卡片之间的跳转，必须将卡片布局管理 `CardLayout` 实例化保留句柄，方便以后处理时使用。

### 11.3.6 网格包布局管理器

网格包布局管理器（`GridBagLayout`）是一个复杂的布局管理器，容器中的组件大小不求一致。通常使用网格包布局管理器要涉及到一个辅助类 `GridBagConstraints`，该类包含 `GridBagLayout` 类用来保存组件布局大小和位置的全部信息，其使用步骤如下。

- （1）创建一个网格包布局管理器的对象，并将其设置为当前容器的布局管理器。
- （2）创建一个 `GridBagConstraints` 对象。
- （3）通过 `GridBagConstraints` 为组件设置布局信息。
- （4）将组件添加到容器中。

`GridBagConstraints` 类的成员变量包括以下几种。

- ❑ `gridx`、`gridy`：指定包含组件的开始边、顶部的单元格，它们的默认值为 `RELATIVE`，该值指将组件添加到刚刚添加组件的右边和下边。`gridx`、`gridy` 应为非负值。
- ❑ `gridwidth`、`gridheight`：指定组件的单元格数，分别是行和列。它们的值应为非负，默认值为 1。
- ❑ `weightx`、`weighty`：指定分配额外的水平和垂直空间，它们的默认值为 0 并且为非负。
- ❑ `ipadx`、`ipady`：指定组件的内部填充宽度，即为组件的最小宽度、最小高度添加多大的空间，默认值为 0。
- ❑ `fill`：指定单元大于组件的情况下，组件如何填充此单元，缺省为组件大小不变。以下为静态数据成员列表，它们是 `fill` 变量的值。

<code>GridBagConstraints.NONE</code>	//组件大小不改变
<code>GridBagConstraints.HORIZONTAL</code>	//水平填充
<code>GridBagConstraints.VERTICAL</code>	//垂直填充
<code>GridBagConstraints.BOTH</code>	//填充全部区域

如果指定单元大于组件的情况下，如果不填充可以通过 `anchor` 指定组件在单元的位置，缺省值为中部。还可以是下面的静态成员，它们都是 `anchor` 的值。

<code>GridBagConstraints.CENTER</code>	//中间位置
<code>GridBagConstraints.NORTH</code>	//上北位置
<code>GridBagConstraints.EAST</code>	//右东位置
<code>GridBagConstraints.WEST</code>	//左西位置
<code>GridBagConstraints.SOUTH</code>	//下南位置

```
GridBagConstraints.NORTHEAST //东北位置
GridBagConstraints.SOUTHEAST //东南位置
GridBagConstraints.NORTHWEST //西北位置
GridBagConstraints.SOUTHWEST //西南位置
```

下面是一个使用 setConstraints()方法设置各组件布局的例子。

```
// 文件: 程序 11.7    GridBagLayoutDemo.java    GridBagLayout 管理器布局演示
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class GridBagLayoutDemo extends Frame{
    Label l1,l2,l3,l4;           //声明, 并初始化 Label 类型域 l1,l2,l3,l4
    TextField tf1,tf2,tf3;       //声明, 并初始化 TextField 类型域 tf1,tf2,tf3
    Button btn1,btn2;            //声明, 并初始化 Button 类型域 btn1,btn2
    CheckboxGroup cbg;           //声明, 并初始化 CheckboxGroup 类型域 cbg
    Checkbox cb1,cb2,cb3,cb4;     //声明, 并初始化 Checkbox 类型域 cb1,cb2,cb3,cb4
    GridBagLayout gb;            //声明, 并初始化 GridBagLayout 类型域 gb
    GridBagConstraints gbc;      //声明, 并初始化 GridBagConstraints 类型域 gbc
    public GridBagLayoutDemo(String title){
        super(title);           //调用父类构造方法
        l1 = new Label("用户名"); //初始化 l1
        l2 = new Label("密码");   //初始化 l2
        l3 = new Label("重复密码"); //初始化 l3
        l4 = new Label("获取途径"); //初始化 l4
        tf1 = new TextField(20);  //初始化 tf1
        tf2 = new TextField(20);  //初始化 tf2
        tf3 = new TextField(20);  //初始化 tf3
        gb=new GridBagLayout();   //初始化 gb
        setLayout(gb);            //设置窗口布局管理器 gb
        gbc=new GridBagConstraints(); //初始化网格容器
        Panel p = new Panel();     //创建, 并初始化面板 Panel
        cbg=new CheckboxGroup();   //初始化多选框组 CheckboxGroup
        cb1=new Checkbox("搜索",cbg,false); //初始化复选框 cb1
        cb2=new Checkbox("广告",cbg,false); //初始化复选框 cb2
        cb3=new Checkbox("朋友",cbg,false); //初始化复选框 cb3
        cb4=new Checkbox("其他",cbg,false); //初始化复选框 cb4
        p.add(cb1);                //在面板 p 中添加 cb1
        p.add(cb2);                //在面板 p 中添加 cb2
        p.add(cb3);                //在面板 p 中添加 cb3
        p.add(cb4);                //在面板 p 中添加 cb4
        btn1=new Button("提交");   //初始化按钮 btn1
        btn2=new Button("重置");   //初始化按钮 btn2
        Panel p1 = new Panel();    //创建, 并初始化面板 p1
        p1.add(btn1);              //在面板 p1 添加按钮 btn1
        p1.add(btn2);              //在面板 p1 添加按钮 btn2
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);    //程序退出
            }
        });
        gbc.fill=GridBagConstraints.HORIZONTAL; //设置 gbc 的 fill 域
        addComponent(l1, 0, 0, 1, 1); //添加 l1 标签
```

```

        addComponent(tf1,0, 2, 1, 4);    //添加 tf1 文本框
        addComponent(l2, 1, 0, 1, 1);    //添加 l2 标签
        addComponent(tf2,1, 2, 1, 4);    //添加 tf2 文本框
        addComponent(l3, 2, 0, 1, 1);    //添加 l3 标签
        addComponent(tf3,2, 2, 1, 4);    //添加 tf3 文本框
        addComponent(l4,4, 0, 1, 1);    //添加 l4 标签
        addComponent(p,4, 2, 1, 1);      //添加面板 p
        addComponent(p1,5, 2, 1, 5);     //添加面板 p1
    }
    //声明添加组件的方法
    public void addComponent(Component c,int row,int col, int nrow,int ncol){
        gbc.gridx=col;                    //设置组件显示区域的开始边单元格
        gbc.gridy=row;                    //设置组件显示区域的顶端单元格
        gbc.gridheight=nrow;              //设置组件显示区域一列的单元格数
        gbc.gridwidth=nrow;               //设置组件显示区域一行的单元格数
        gb.setConstraints(c,gbc);         //设置布局的约束条件
        add(c);                           //组件 c 添加到容器中
    }
    public static void main(String args[ ]){
        //创建, 并初始化 GridBagLayoutDemo 对象 mygb
        GridBagLayoutDemo mygb =new GridBagLayoutDemo("网格包布局管理器");
        mygb.setSize(300,200);           //设置窗口大小
        mygb.setVisible(true);           //显示窗口
    }
}

```

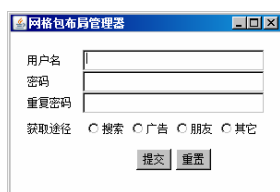


图 11-10 GridBagLayoutDemo.java 运行结果

编写完程序后, 使用 `javac` 命令编译该文件产生 `class` 文件, 然后使用 `java` 命令运行该 `class` 文件, 运行结果如图 11-10 所示。

从程序 11.7 可以看出, 网络包布局管理器是比较复杂的布局管理器, 也正是因为它的复杂性才决定了它的功能强大性。它通常需要与 `GridBagConstraints` 配合使用, 通过 `GridBagConstraints` 的对象来设置组件的布局信息。

### 11.3.7 容器的嵌套

在复杂的图形用户界面设计过程中, 将所有的组件一起添加到一个容器时, 图形界面的管理是具有一定的难度。容器中可以添加容器, 这就是容器的嵌套。容器的嵌套使复杂的图形用户界面的设计更加方便有效。下面是一个容器嵌套的例子。

```

// 文件: 程序 11.8    NestedContainer.java        嵌套容器演示
//导入需要使用的包和类
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class NestedContainer{
    public static void main(String args[ ]){
        new NestedContainer().show();           //创建 NestedContainer 实例, 并调用 show 方法
    }
}

```



```

}
//声明，并实现 show 方法
public void show(){
    f = new Frame("容器嵌套使用");           //初始化窗口对象 f
    bo= new Button("打开");                   //初始化按钮对象 bo
    bc= new Button("关闭");                   //初始化按钮对象 bc
    pn = new Panel();                         //初始化面板对象 pn
    pn.setBackground(Color.green);           //设置面板背景色
    pn.add(bo);                              //面板 pn 中添加按钮 bo
    pn.add(bc);                              //面板 pn 中添加按钮 bc
    f.add(pn,"North");                       //在窗口中添加面板 pn
    bw=new Button("左西");                   //初始化按钮对象 bw
    f.add(bw,"West");                       //将按钮对象 bw 添加到 West 位置
    p=new Panel();                          //初始化面板对象 p
    p.setBackground(Color.red);             //设置面板对象的背景色
    f.add(p,"Center");                      //在窗口 f 中添加 p 与 Center 位置
    f.setSize(400, 300);                    //设置 f 窗口的大小
    f.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent evt) {
            f.setVisible(false);             //设置窗口 f 不可见
            f.dispose();                    //释放窗口及其子组件的屏幕资源
            System.exit(0);                 //退出程序
        }
    });
    f.setVisible(true);                     //显示窗口 f
}
private Frame f;                          //声明窗口域 f
private Panel pn,p;                       //声明面板 Panel 域 pn, p
private Button bw;                         //声明面板 Button 域 bw
private Button bo,bc;                     //声明面板 Button 域 bo,bc
}

```

编写完程序后，使用 javac 命令编译该文件产生 class 文件，然后使用 java 命令运行该 class 文件，运行结果如图 11-11 所示。

程序 11.8 在窗口 f 中添加了两个容器 pn 和 p，因此窗口 f 的布局管理器与 pn、p 的布局管理器可以随便选择。

注意：如果采用无布局管理器，必须调用 setLayout(null)，使用 setLocation()、setSize()、setBounds() 等方法设置组件的布局。由于此方法与平台相关，所以不建议读者使用。

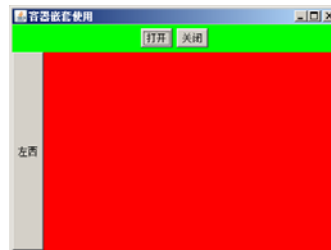


图 11-11 NestedContainer.java 运行结果

## 11.4 AWT 组件库

AWT 的组件库是图形界面设计的基本组成元素，主要包括按钮、标签、复选框、下拉菜单、画布、文本输入区、列表、滚动条、对话框、菜单等组件。本节主要讲述 AWT 组件

的应用，通过本节的学习，读者可以加深对 AWT 的理解，并且掌握各种常用组件的使用方法和技巧。

### 11.4.1 按钮、标签

按钮（Button）是经常使用的组件之一，其构造方法如下。

```
Button b = new Button("按钮名称");
```

前面讲述的内容中已经使用过按钮。当单击按钮后，会产生 `ActionEvent` 事件，如果应用程序需要执行基于按下按钮并释放的某个动作，则应该实现 `ActionListener` 接口监听，目的是为了通过调用按钮的 `addActionListener` 方法来接受此按钮的单击事件。

标签（Label）是一个可在容器中放置静态文本的组件，一个标签只显示一行文本，它能够通过应用程序更改，但不可直接修改。其构造方法如下。

```
Label label1 = new Label("文本内容")
```

下面是一个演示标签使用的例子。

```
// 文件：程序 11.9    LabelDemo.java           : Label 演示
//导入需要使用的包和类
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class LabelDemo{
    public static void main(String args[] )
    {
        //创建，并初始化窗口对象 f
        final Frame f = new Frame();
        f.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10)); //设置窗口 f 的布局管理器为
        FlowLayout
        Label label1 = new Label("北京欢迎您"); //创建，并初始化标签 label1
        Label label2 = new Label("2008 奥运会"); //创建，并初始化标签 label2
        f.add(label1);                               //窗口中添加 label1
        f.add(label2);                               //窗口中添加 label2
        f.setSize(200, 150);                         //设置窗口大小
        f.setVisible(true);                          //显示窗口 f
        //为窗口添加监听器 WindowListener
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                f.setVisible(false); //设置窗口 f 不可见
                f.dispose();        //释放窗口及其子组件的屏幕资源
                System.exit(0);      //退出程序
            }
        });
    }
}
```

编写完程序后，使用 `javac` 命令编译该文件产生 `class` 文件，然后使用 `java` 命令运行该 `class` 文件，运行结果如图 11-12 所示。

程序 11.9 中创建了两个标签对象 label1、label2，将两个标签添加于窗口中，再设置窗口大小，最后通过 setVisible 方法显示窗口。

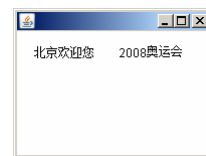


图 11-12 LabelDemo.java

运行结果

### 11.4.2 复选框、下拉式菜单

复选框（Checkbox）是一个可处于“on”或者“off”状态的组件，“on”和“off”状态的转变需要单击复选框，复选框旁边显示文本信息。复选框的主要方法如下。

```
add(new Checkbox("one",null,true));//添加复选框按钮，该项为默认选项
add(new Checkbox("two"));           //添加复选框按钮，显示文本 two
add(new Checkbox("three"));         //添加复选框按钮，显示文本 three
```

作为可选的复选框，如果使用 CheckboxGroup 类对象将复选框组合到一起作为单个对象控制之后，这一组复选框中的按钮最多只能有一个处于“on”状态。单击并打开一组的一个复选框会强制其他原来处于打开状态的复选框处于“off”状态。具体使用方法如下。

```
CheckboxGroup cbg = new CheckboxGroup();//创建 CheckboxGroup 对象 cbg
add(new Checkbox("one", cbg, true));    //添加复选框按钮为一组按钮，默认打开
add(new Checkbox("two", cbg, false));   //添加复选框按钮为一组按钮，默认关闭
add(new Checkbox("three", cbg, false)); //添加复选框按钮为一组按钮，默认关闭
```

当单击复选框时，产生 ItemEvent 事件，复选框可以使用 ItemListener 来监听 ItemEvent 事件，通过复选框的方法 getStateChange() 可以获得当前状态，方法 getItem() 获得复选框的字符串对象。

下拉式菜单（Choice）表示一个弹出式选择菜单组件，并且只能选择其中的一项。下拉式菜单的主要特点是能够存放多个选项并且可以节约空间，它与复选框一样使用 ItemListener 接口监听 ItemEvent 事件。

```
Choice colorChooser=new Choice();//创建，并初始化 Choice 对象 colorChooser
colorChooser.add("Green");        //添加一个菜单项 Green
colorChooser.add("Red");          //添加一个菜单项 Red
colorChooser.add("Blue");         //添加一个菜单项 Blue
```

### 11.4.3 画布

画布（Canvas）是处于屏幕上的一个空白矩形区域，在该区域可以绘图，也可以获得用户激发的事件。具体的程序中，通过创建画布的派生类，可以丰富画布的功能。通过调用画布的子类 paint() 方法，以实现需要的绘图功能。画布的绘图方法是 paint()，原型如下。

```
public void paint(Graphics g)
```

注意：由于画布的 paint() 方法默认操作是清除画布，所以重写此方法的应用程序不需要调用 super.paint(g)。

另外，绘图相关的方法还有：

```
public void update(Graphics g)
```

调用 `update()` 的目的是为了调用 `repaint()` 方法。执行过程是通过背景色填充画布的背景，然后再调用 `paint()` 方法重新绘制图像。

注意：`update()` 方法不同于 `paint` 方法，必须在重写的 `update()` 方法中第一行调用 `super.update(g)`，或者重新实现背景色填充和重绘两个功能。

画布组件可以监听鼠标事件和键盘事件。下面是一个创建画布的例子。

```
// 文件：程序 11.10 CanvasDemo.java      CanvasDemo 演示
//导入需要使用的包和类
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class CanvasDemo extends Panel {
    public static void main(String args[] )
    {
        new CanvasDemo();           //创建 CanvasDemo 实例
    }
    public CanvasDemo()
    {
        f = new Frame();             //初始化窗口对象 f
        mc = new MyCanvas();         //初始化画布对象
        mc.repaint(0,0,100,100);     //画布重绘
        add("Center",mc);           //添加画布到窗口 Center 位置
        f.add(mc);                   //在窗口 f 中添加画布对象
        f.setSize(300, 200);         //设置窗口的大小
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                f.setVisible(false); //设置窗口 f 不可见
                f.dispose();         //释放窗口及其子组件的屏幕资源
                System.exit(0);      //退出程序
            }
        });
        f.setVisible(true);          //显示窗口 f
    }
    private MyCanvas mc;
    private Frame f;
    class MyCanvas extends Canvas {
        //重载 paint 方法
        public void paint(Graphics g) {
            g.setColor(Color.red);   //设置绘图颜色
            g.drawRect(100, 40, 100, 100); //绘制矩形
            g.drawString("Canvas",100,50); //绘制字符串
        }
    }
}
```

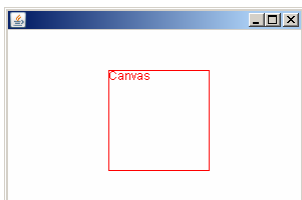


图 11-13 CanvasDemo.java 运行结果

编写完程序后，使用 `javac` 命令编译该文件产生 `class` 文件，然后使用 `java` 命令运行该 `class` 文件，运行结果如图 11-13 所示。

程序 11.10 中，`MyCanvas` 类派生于 `Canvas`，并重载 `paint` 方法，绘制字符串。先创建程序并初始化 `MyCanvas` 对象 `mc`，然后调用 `repaint` 方法重绘，最后将画布添加于

窗口中，设置窗口大小并显示。

#### 11.4.4 文本输入区、列表

单行文本输入区是可编辑的单行文本组件，一般用来输入内容比较少的文本信息，例如姓名、籍贯等。单行文本输入区的构造方法有 4 种类型，其使用方法如下。

- ❑ `tf1 = new TextField();` 空单行文本输入区。
- ❑ `tf2 = new TextField(10);` 指定单行文本输入区长度为 10 的空文本域。
- ❑ `tf3 = new TextField("好");` 带有初始文本内容的单行文本输入区。
- ❑ `tf4 = new TextField("好", 20);` 带有初始文本内容并具有指定长度为 20 的单行文本输入区。

每当用户在单行文本输入区中键值时，会产生一个或多个键事件，包括 `keyPressed`、`keyReleased` 或 `keyTyped`。键事件被传递给 `KeyListener` 或 `KeyAdapter` 对象，并且这些对象可以通过 `addKeyListener` 方法注册到组件。

当在单行文本输入文本结束，按下回车键将产生 `ActionEvent` 事件，通过 `ActionListener` 中的 `actionPerformed()` 方法对事件处理。调用 `setEditable(boolean)` 方法，单行文本输入区被设置为只读。文本输入区可以显示多行文本区域，与单行文本输入区相似，文本输入区也有 4 种构造方法创建文本输入区对象，但是如果要指定文本区域大小，必须指定行和列。

- ❑ `ta1 = new TextArea();` 一个空的文本输入区。
- ❑ `ta2 = new TextArea (5,10);` 指定文本输入区行为 5，列为 10 的空文本域。
- ❑ `ta3 = new TextArea ("好");` 带有初始文本内容的文本输入区。
- ❑ `ta4 = new TextArea("好!", 5, 30);` 带有初始内容的行数为 5，列数为 30 的文本输入区。
- ❑ `ta2.setEditable(false);` 可以用成员方法 `setEditable()` 决定用户是否可以对文本区的内容进行编辑。

文本输入区的成员方法 `getText()` 可获得文本区的内容。文本输入区的文本超过显示区域时，可以显示水平或垂直的滚动条。

列表框组件提供一个可滚动的文本列表，并且通过参数设置列表框是单选项或多选项。如果列表框中选项比较多，并且超过了列表框显示的范围时，它的右边会产生一个滚动条，用来浏览显示的选项。

```
List lst=new List(4,false); //显示 4 行，不允许多选
lst.add("语文");           //为类别添加选项"语文"
lst.add("数学");           //为类别添加选项"数学"
lst.add("英语");           //为类别添加选项"英语"
lst.add("物理");           //为类别添加选项"物理"
lst.add("化学");           //为类别添加选项"化学"
lst.add("历史");           //为类别添加选项"历史"
lst.add("地理");           //为类别添加选项"地理"
```

### 11.4.5 滚动条

滚动条 (Scrollbar) 是一个大家很熟悉的组件，在应用程序中可以调整线性值。滚动条可以设置可以选择的取值范围。创建滚动条时，必须指定的属性包括方向（垂直还是水平）、初始值（滚动块位置的值）、滑块的大小、最小值和最大值。

```
public Scrollbar(int orientation, int initialValue, int sizeOfSlider, int minValue, int maxValue);
```

下面是一个例子。

```
Scrollbar redSlider;                                //声明 Scrollbar 对象 redSlider
redSlider = new Scrollbar(Scrollbar.VERTICAL,0,1,0,255); //初始化 redSlider 对象
add(redSlider);                                     //将 redSlider 添加于容器中
```

当改变滚动条的值时，滚动条接受事件 AdjustmentEvent。如果需要监听 AdjustmentEvent 事件，必须实现 AdjustmentListener 接口，并且需要调用 addAdjustmentListener() 和 removeAdjustmentListener() 方法添加或删除侦听器。

用户可以通过鼠标、键盘的 <Page Up> 和 <Page Down> 键改变滚动条的位置。如果应用程序需要显示滑动块所在位置的值，用户需要添加文本域和相应的时间监听和处理方法。下面是一个演示滚动条使用的例子。

```
// 文件: 程序 11.11 ScrollbarDemo.java      Scrollbar 演示
//导入需要使用的包和类
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class ScrollbarDemo extends Frame {
    //声明 ScrollbarDemo 构造方法
    public ScrollbarDemo()
    {
        super();                //调用父类构造方法
        init();                  //调用 init 方法
    }
    public static void main(String args[ ])
    {
        new ScrollbarDemo();     //实例 ScrollbarDemo 对象
    }
    Scrollbar slider;            //声明滚动条 slider
    Label label;                 //声明标签 label
    public void init( ) {
        setLayout(new GridLayout(1, 3)); //设置窗口的布局管理器为 GridLayout
        slider = new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,100); //初始化滚动条对象 slider
        label = new Label("0~100");      //初始化标签对象 label
        label.setBackground(Color.cyan); //设置标签的背景色
        add(label);                     //在窗口中添加标签
        add(slider);                    //在窗口中添加滚动条
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                setVisible(false); //设置窗口 f 不可见
                dispose();         //释放窗口及其子组件的屏幕资源
                System.exit(0);     //退出程序
            }
        });
    }
}
```

```

    });
    setSize(300, 50);           //设置窗口的大小
    setVisible(true);          //显示窗口
}

```

编写完程序后,使用 javac 命令编译该文件产生 class 文件,然后使用 java 命令运行该 class 文件,运行结果如图 11-14 所示。



图 11-14 CanvasDemo.java 运行结果

程序 11.11 中,创建并初始化了一个滚动条对象和一个标签对象,然后将两个组件添加到窗口中,设置窗口的大小,最后显示窗口。

## 11.4.6 对话框、菜单

对话框是 Window 类的子类,它是一个带有标题和边界的顶层窗口。对话框最主要的特点是它依赖于其他窗口,它分为无模式和有模式两种。两种模式的区别在于有模式对话框会阻止将内容输入到应用程序中的其他一些顶层窗口中。

文件对话框显示一个对话框窗口,用户可以打开、选择并存储文件。它是一个模式对话框,当文件对话框显示后,其他应用程序被阻塞,直到用户选择文件或者取消操作。应用的具体方法如下。

```

FileDialog d=new FileDialog(ParentFr,"FileDialog"); //创建, 初始化对话框对象 d
d.setVisible(true);                               //显示对话框 d
String filename=d.getFile();                       //获取选取的文件 url

```

菜单是一个比较复杂的内容,通常有 3 个相关的类:Menu、MenuBar 和 MenuItem。类层次结构如图 11-15 所示。

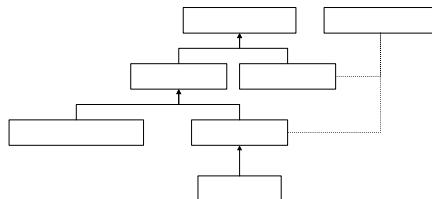


图 11-15 菜单的层次结构

### 1. 菜单栏

菜单无法直接添加在窗口中,也无法用布局管理器控制。菜单必须先添加到菜单栏中。为窗口添加菜单时,必须先将菜单栏添加到窗口中,作为菜单树的基础。

```

Frame fr = new Frame("MenuBar"); //创建, 并初始化窗口 fr
MenuBar mb = new MenuBar();       //创建, 并初始化菜单栏
fr.setMenuBar(mb);               //将菜单栏对象设置为窗口的菜单栏
fr.setSize(150,100);             //设置窗口大小
fr.setVisible(true);             //显示窗口

```

### 2. 菜单

菜单可以被添加到菜单栏或其他菜单对象中。

```

Frame fr = new Frame("MenuBar"); //创建, 并初始化窗口 fr
MenuBar mb = new MenuBar();       //创建, 并初始化菜单栏
fr.setMenuBar(mb);               //设置窗口的菜单栏
Menu m1 = new Menu("File");      //创建, 并初始化菜单 m1

```

```

Menu m2 = new Menu("Edit");      //创建，并初始化菜单 m2
Menu m3 = new Menu("Help");      //创建，并初始化菜单 m3
mb.add(m1);                      //将菜单添加于菜单栏中 m1
mb.add(m2);                      //将菜单添加于菜单栏中 m2
mb.setHelpMenu(m3);             //设置 m2 为帮助菜单
fr.setSize(200,200);            //设置窗口大小
fr.setVisible(true);            //显示窗口

```

### 3. 菜单项

菜单项是菜单树的“节点”，它一般被添加到菜单对象中。菜单中的所有项必须属于类 `MenuItem`。

```

Menu m1 = new Menu("File");      //创建，并初始化菜单
MenuItem mi1 = new MenuItem("Save"); //创建，并初始化菜单项 mi1
MenuItem mi2 = new MenuItem("Load"); //创建，并初始化菜单项 mi2
MenuItem mi3 = new MenuItem("Quit"); //创建，并初始化菜单项 mi3
m1.add(mi1);                    //将 mi1 菜单项添加于菜单 m1
m1.add(mi2);                    //将 mi2 菜单项添加于菜单 m1
m1.addSeparator();              //为菜单 m1 中添加分割线
m1.add(mi3);                    //将 mi3 菜单项添加于菜单 m1

```

通常没有必要在 `MenuBar` 和 `Menu` 中注册监听器，只需要为 `MenuItem` 添加监听器 `ActionListener` 完成事件处理。一个完整的 `Menu` 示例如下。

```

// 文件：程序 11.12  MenuDemo.java      Menu 演示
//导入需要使用的包和类
import java.awt.*;
import java.awt.event.*;
class MenuDemo extends Frame {
    PopupMenu pop;                //声明，并初始化弹出菜单 pop
    public MenuDemo(String string) {
        super(string);           //调用父类构造方法
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent evt) {
                setVisible(false); //设置窗口 f 不可见
                dispose();         //释放窗口及其子组件的屏幕资源
                System.exit(0);    //退出程序
            }
        });
        setSize(400,300);        //设置窗口大小
        Menu menu = new Menu("文件");//创建，并初始化菜单对象 menu
        menu.add("新建");         //为菜单对象添加“新建”菜单项
        menu.add("打开");         //为菜单对象添加“打开”菜单项
        menu.add("关闭");         //为菜单对象添加“关闭”菜单项
        menu.add("退出");         //为菜单对象添加“退出”菜单项
        Menu irons = new Menu("编辑");//创建，并初始化菜单对象 menu
        irons.add("重写");         //为菜单对象添加“重写”菜单项
        irons.add("复制");         //为菜单对象添加“复制”菜单项
        irons.add("删除");         //为菜单对象添加“删除”菜单项
        irons.add("代替");         //为菜单对象添加“代替”菜单项
        irons.add("查找");         //为菜单对象添加“查找”菜单项
        irons.addSeparator();      //为菜单添加分割线
        irons.add("取消");         //为菜单对象添加“取消”菜单项
    }
}

```



```

        irons.insert("粘贴", 2);           //为菜单对象添加“粘贴”菜单项
        MenuBar mb = new MenuBar();       //创建，并初始化菜单栏 mb
        mb.add(menu);                     //将 menu 菜单添加到菜单栏 mb
        mb.add(iron);                     //将 iron 菜单添加到菜单栏 mb
        setMenuBar(mb);                   //将菜单栏与窗口管理
        pop = new PopupMenu("menu");      //初始化浮动弹出菜单
        pop.add("新建");                  //在菜单中添加“新建”菜单项
        pop.add("粘贴");                  //在菜单中添加“粘贴”菜单项
        pop.add("取消");                  //在菜单中添加“取消”菜单项
        final TextArea p = new TextArea(100, 100); //声明，初始化 TextArea 对象 p
        p.setBackground(Color.blue);      //设置颜色
        p.add(pop);                       //为文本区域对象中添加浮动菜单
        //以下是事件处理，稍后介绍
        p.addMouseListener(new MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                if(evt.isPopupTrigger()) {
                    System.out.println("popup trigger");           //输出字符串信息
                    System.out.println(evt.getComponent());         //输出组件属性
                    System.out.println("" + evt.getX() + " " + evt.getY()); //输出坐标
                    pop.show(p, evt.getX(), evt.getY());           //显示浮动窗口
                }
            }
        });
        this.add(p, BorderLayout.CENTER); //将面板 p 添加到窗口
    }
    public static void main (String [ ] args) {
        new MenuDemo("菜单演示").setVisible(true); //创建 MenuDemo 实例，并显示窗口
    }
}

```

编写完程序后，使用 javac 命令编译该文件产生 class 文件，然后使用 java 命令运行该 class 文件，运行结果如图 11-16 所示。

程序 11.12 中，创建了一个窗口，并为窗口设置菜单。创建菜单的一般步骤是：先创建 Menu 类的对象 menu，然后通过 add 方法为菜单添加菜单项，这样菜单就创建成功。再将菜单对象通过 add 方法添加域 MenuBar 对象，通过窗口的 setMenuBar 将 MenuBar 对象设置为窗口菜单栏。创建浮动菜单栏是创建一个 PopupMenu 对象，并为之添加菜单项。

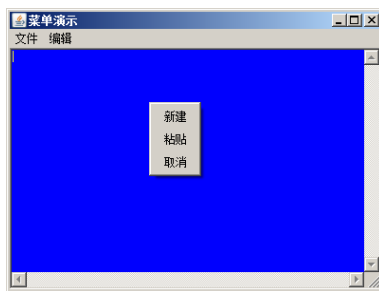


图 11-16 MenuDemo.java 运行结果

## 11.5 绘图

Java 中提供了 Graphics 类用于绘图和显示格式化文字的输出工具。绘图必须在一个窗口中进行，例如 Frame 或者 Applet 等。本节主要介绍如何设置绘图颜色，设置文字的字体、大小和颜色以及绘制图形的程序案例。

### 11.5.1 文字输出

如果使用者自己不设置，将使用默认字体、大小和颜色。若要改变默认属性，必须使用相关的方法改变 Graphics 对象的属性。设置窗口背景色的方法如下。

```
public void setBackground(Color c) //设置组件的背景色。
```

背景色对每一个组件的影响不相同，受背景色的影响而导致组件在不同的操作系统表现可能不同。设置 Graphics 的绘图色如下。

```
public abstract void setColor(Color c)
```

该方法将 Graphics 的绘图颜色设置为指定颜色。设置之后，如果没有修改，Graphics 的绘图的颜色就会一直保持。选择颜色有两种方法，一种是直接用 RGB 颜色值创建 Color 对象如下所示；另一种是用颜色常量，例如 Color.red, Color.green 等。

```
Color color = new Color(R,G,B)
```

字体设置需要使用 java.awt.Font 类创建一个对象，它常用的构造方法如下。

```
Font(字体名或逻辑字体名,字型,字号)
```

字体名是指与运行程序的操作系统相适应的字体外观名称或者字体系列名称。字型为 Font 类声明了样式静态常量有 4 种：Font.PLAIN、Font.BOLD、Font.ITALIC、Font.BOLD|ITALIC。下面是一个格式化文字输出的例子。

```
// 文件：程序 11.13 FontDemo.java 字体大小和颜色的演示
//导入需要使用的包和类
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class FontDemo extends Frame{
    //声明 FontDemo 构造方法
    public FontDemo(String str)
    {
        super(str); //调用父类构造方法
        //为窗口添加窗口监听器
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                setVisible(false); //设置窗口 f 不可见
                dispose(); //释放窗口及其子组件的屏幕资源
                System.exit(0); //退出程序
            }
        });
    }
    public static void main(String args[ ])
    {
        //创建，并初始化 FontDemo 对象 f
        FontDemo f = new FontDemo("字体大小和颜色");
        f.setBackground(Color.black); //设置窗口背景色
        f.setSize(400, 300); //设置窗口大小
        f.setVisible(true); //显示窗口
    }
}
```

```

//重载 paint 方法
public void paint (Graphics g){
    Font font = new Font("Arial", Font.BOLD, 18);           //创建字体类对象 font
    g.setColor(Color.red);                                   //设置字体颜色
    g.setFont(font);                                         //设置字体
    g.drawString("Font:" + font.getName() + font.getSize(),30, 50); //绘制字符串文本
    g.setFont(new Font("宋体", Font.BOLD, 36));             //设置字体
    g.drawString("2008 北京奥运会",20, 120);               //绘制字符串文本
    g.setColor(new Color(0,0,0));                           //设置演示
}
}

```

编写完程序后,使用 javac 命令编译该文件产生 class 文件,然后使用 java 命令运行该 class 文件,运行结果如图 11-17 所示。

在程序 11.13 中,子类 FontDemo 重载了 paint 方法。程序创建并初始化字体对象 font(重新组合),通过 setColor 方法设置字符串颜色,通过 setFont 方法设置字体类型,通过 drawString 方法绘制字符串。

注意:创建字体对象之后,在设置新字体 g.setFont(font) 后一直有效,直到再次设置调用 g.setFont(font) 方法设置字体。实际应用中,习惯先保存原来的颜色和字体设置,等用完后还原。



图 11-17 FontDemo.java 运行结果

## 11.5.2 图形绘制

Graphics 常用的绘图方法有以下几种。

- ❑ abstract void drawLine(int x1, int y1, int x2, int y2): 用图形上下文的当前颜色绘制在点(x1, y1)和(x2, y2)之间画一条线。
- ❑ abstract void drawRect(int x, int y, int width, int height): 绘制指定矩形左上坐标为(x,y), 并指定长宽的边框。
- ❑ abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight): 用图形上下文的当前颜色绘制圆角矩形的边框。
- ❑ abstract void fill3DRect(int x, int y, int width, int height, boolean raised): 用图形上下文的当前颜色填充的 3D 矩形。
- ❑ abstract void fillOval(int x, int y, int width, int height): 用图形上下文的当前颜色填充外接指定矩形框的椭圆。
- ❑ abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): 用图形上下文的当前颜色填充覆盖指定矩形的圆弧或椭圆弧。
- ❑ abstract void drawPolyline(int[] xPoints, int[] yPoints, int nPoints): 用图形上下文的当前颜色绘制由 x 和 y 坐标数组定义的一系列连接线。
- ❑ abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints): 用图形上下文的当前颜色绘制一个由 x 和 y 坐标数组定义的闭合多边形。
- ❑ void drawPolygon(Polygon p): 用图形上下文的当前颜色绘制由指定的 Polygon 对象

定义的多边形边框。

下面是一个 Graphics 绘图的例子。

```
// 文件: 程序 11.14 GraphicsDraw.java Graphics 绘图
//导入需要使用的包和类
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class GraphicsDraw extends Frame {
    public static void main(String args[ ])
    {
        GraphicsDraw f = new GraphicsDraw("Graphics 绘图"); //创建, 并初始化 GraphicsDraw
对象 f
        f.setSize(500, 300); //设置窗口 f 大小
        f.setVisible(true); //显示窗口

    }
    public void paint(Graphics g) {
        setXY(); //调用 setXY 方法
        Point p = new Point(cx / 2, cy / 2); //创建, 并初始化 Point 对象 p
        g.drawLine(0, 0, cx, cy); //绘制一条(0,0)(cx,cy)的线
        g.draw3DRect(cx/2, cy/2, cx/3, cy/3, false); //绘制 3D 高亮矩形
        Rectangle rect = new Rectangle((p.x - 40), (p.y - 40), 80, 40); //创建, 初始化 Rectangle
对象 rect
        int[] xP = {(p.x - 40), (p.x + 90), (p.x+200), (p.x - 40)}; //创建, 初始化 x 坐标数组
        int[] yP = {(p.y - 40), (p.y +140), (p.y + 60), (p.y-40)}; //创建, 初始化 y 坐标数组
        g.drawArc(rect.x, rect.y, rect.width, rect.height * 2, 270, 90); //绘制圆弧
        g.drawPolygon(xP, yP,3); //绘制多边形
    }

    //声明, 并实现方法, 用于设置 cx、cy
    public void setXY()
    {
        Dimension d = getSize(); //获取窗口的大小
        cx = d.width/2 ; cy = d.height/2; //去窗口中点
    }
    public GraphicsDraw (String str) {
        super(str); //调用父类构造方法
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                setVisible(false); //设置窗口 f 不可见
                dispose(); //释放窗口及其子组件的屏幕资源
                System.exit(0); //退出程序
            }
        });
    }
    private int cx; //声明 int 类型域 cx
    private int cy; //声明 int 类型域 cy
}
```

编写完程序后,使用 javac 命令编译该文件产生 class 文件,然后使用 java 命令运行该 class 文件,运行结果如图 11-18 所示。

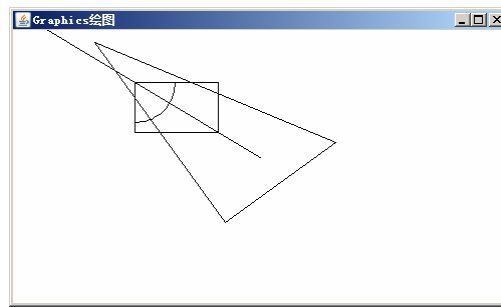


图 11-18 GraphicsDraw.java 运行结果

程序 11.14 中，重写了 `paint` 方法，通过 `drawLine` 方法绘制直线、使用 `draw3Drect` 方法绘制 3D 矩形、使用 `drawArc` 方法绘制弧形、使用 `drawPolygon` 方法绘制多边形。

## 11.6 小结

本章主要讲述了 Java 图形界面设计的基础知识，其中最重要的是常用的容器、组件以及布局管理器的使用；另外还具体介绍了多种组件，包括按钮、标签、复选框、下拉菜单、画布、滚动条等；最后还专门讨论了绘图的基本知识。本章的内容是学习图形设计的基础，读者必须掌握。