Java
程序
设计
丛书

Java GUI 程序设计

肖 刚 等 编著

清 华 大 学 出 版 社

(京) 新登字 158 号

内 容 简 介

本书深入地介绍了各种 Java GUI 程序设计技术，详细讨论了 AWT 构件类和布局管理器的使用、图形图像处理技术、简单动画生成、高级事件处理技术、JFC-Swing 接口编程，以及各种特殊技巧。

本书内容由浅入深，讲解详细，并附有大量的实例程序，既可作为初学 Java GUI 程序设计的入门指导书，也可作为 Java 高级程序员的技术参考书。

图书在版编目(CIP)数据

Java GUI 程序设计/肖刚等编著 . - 北京：清华大学出版社，1998.11
(Java 程序设计丛书)
ISBN 7-302-03117-7

Ⅰ.J… Ⅱ.肖… Ⅲ.Java 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(98)第 29781 号

出版者：清华大学出版社(北京清华大学校内，邮编 100084)
http://www.tup.tsinghua.edu.cn
印刷者：北京市清华园胶印厂
发行者：新华书店总店北京科技发行所
开 本：787 × 1092 1/16 印张：13 字数：307 千字
版 次：1998 年 11 月第 1 版 1998 年 11 月第 1 次印刷
书 号：ISBN 7 - 302 - 03117 - 7/TP • 1691
印 数：0001 ~ 6000
定 价：17.50 元

前　言

广义地说，用户界面(UI)是程序同用户之间的所有交互的总称，它不仅包括用户所见到的部分，而且包括用户所听到的部分和所感觉到的部分。Java语言提供了丰富的用户界面功能支持，其中图形用户界面(GUI)是Java UI的核心，也基本上是所有Java应用程序设计的一个重要基础。本书正是一本详细介绍各种Java GUI程序设计技术及技巧的教程。

本书分为三个部分。第一部分介绍基本的Java GUI程序设计技术，对Java GUI程序设计基本原理、AWT构件类和布局管理器进行了详细的论述；第二部分讲述基本的图形、图像处理功能以及简单动画的生成；第三部分介绍各种高级Java GUI程序设计技术，包括高级的事件处理技术、新的JFC-Swing接口编程，以及各种特殊功能，如桌面颜色控制、打印等。

本书是面向编程人员的Java GUI程序设计书，要求读者已经了解了Java的基础知识和GUI程序设计的基础知识。读者在阅读本书的同时，参阅相应的JDK文档，并上机实习，将会得到更好的学习效果。本书既可以作为系统学习Java GUI程序设计之用，也可供高级程序人员的Java GUI程序设计参考。

本书是集体创作的结晶，除了封面上署名的主要编著者外，莫倩、李子木、黄光奇、李磊、周会平、宋辉、王锋、王炜、任伟、王经亮等都参加了本书的编著工作。

Java是一个迅速发展的技术，尤其是其GUI设计部分变化很快，书中难免存在不足之处，敬请读者予以批评指正。

作　者

1998年3月

目　录

第一部分　Java GUI 程序设计基础

第二部分 图形和图像处理

第三部分 高级 Java GUI 程序设计

第1部分

Java GUI程序设计基础

本部分介绍基本的 Java GUI 程序设计技术，这一部分是基于 AWT 1.0 的。因为 AWT 1.0 仍然在大量使用，而且它是理解 AWT 1.1 和 JFC-Swing 中高级技术的基础，所以这里首先介绍它。本部分分为 3 章：第 1 章讲述编写 Java GUI 程序的基本原理，并对一个简单的例子程序进行彻底的剖析；第 2 章详细介绍各种 AWT 组件类及其使用方法；第 3 章讲述各种 AWT 布局管理器的使用技术。

第 1 章　Java GUI 程序设计基本原理

Java 给 Internet 带来的第一个好处就是使得 WWW 的页面变得生动起来，使它不再局限于僵硬的文本和静态图形显示，而这首先得益于 Java 提供的丰富灵活的 GUI 设计支持。可以说，Java GUI 程序设计是 Java 程序设计的基础，十分重要。JDK 每推出一个新的版本，都新增许多 GUI 程序设计技术和功能，所以 Java GUI 程序设计技术发展很快，新老技术变化很大。但有两点是保持不变的：一点是 Java GUI 程序设计技术保持了较好的兼容性，新老技术可以同时使用，而且从技术思想上具有连贯性；另一点是 Java GUI 程序设计技术还将向着更加灵活高效、兼容性更好、平台独立性更高的方向不断发展。

本章首先讲述采用 Java AWT 1.0 类库编写 Java GUI 程序的基本原理，这是最基本的 Java GUI 程序设计技术，也是其他高级 Java GUI 程序设计技术的基础。本章的目的是使读者通过本章的学习能够理解 Java GUI 程序设计的基本原理，为后续学习各种专门技术（如组件的使用、布局管理器功能、图形图像处理以及更高级的 GUI 设计技术等）打下理论基础，并通过最后一节的实例对 Java GUI 程序设计有一个感性的认识。本章分为 5 节，1.1节到 1.4 节讲述 Java GUI 程序设计的基本原理、Java AWT 类库的构成等；1.5 节通过对一个简单例子程序的分析介绍 Java GUI 程序的基本结构，并对事件处理等技术进行更深入的讨论。

1.1　AWT 组件类

图 1.1 表示了所有 AWT 组件类（Component）的类层次结构。除了与菜单有关的组件类以外，所有组件类都是 AWT Component 类的子类，菜单类则是 AWT MenuComponent 类的子类。

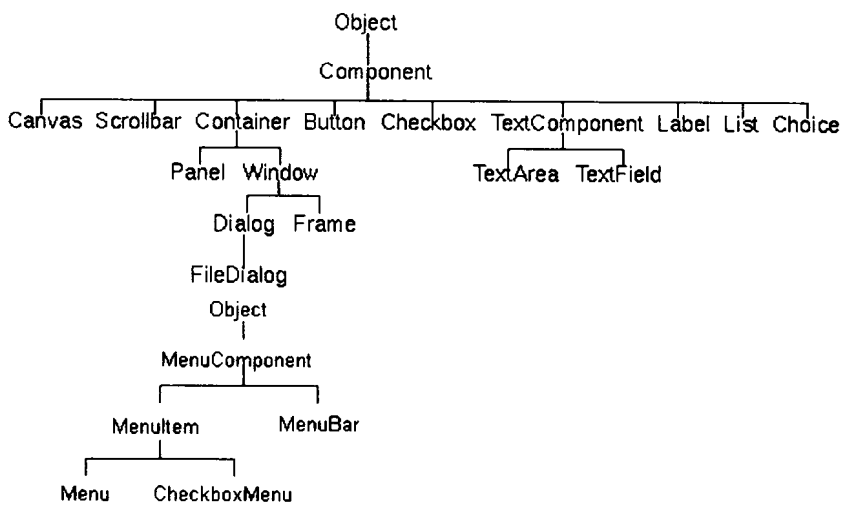

图 1.1　AWT 组件类层次

在实际应用中，通常按功能把 AWT 组件类分为以下几类：

- 基本控件：按钮(Button)类、复选钮(Checkbox)类、选择列表(Choice)类、列表(List)类、菜单(Menu)类和文本域(TextField)类。按钮、复选钮、选择列表、列表、菜单和文本域类提供基本的控件。它们是最常用的向 Java 程序发送命令的方法。当用户激活某个控件时——如按下一个按钮或在文本域中输入回车——它就会产生一个事件(ACTION_EVENT)，而包含此控件的对象通过实现 action( )方法就可以响应这个事件。
- 取得用户输入的其他方法：滚动条(Scrollbar)类和文本区(TextArea)类。除了使用基本控件外，Java 程序还可以使用滚动条和文本区类得到用户输入。滚动条类同时用于实现滑动器和滚动条两种功能，文本区类提供一个可以编辑的多行文本区。
- 生成定制组件：画布类。画布类可以用来生成定制组件。使用画布类，可以在屏幕上绘制图形并实现各种事件处理。
- 标签：标签用来显示一行不可选择的文本。
- 容器：窗口及底板。AWT 提供两类容器：窗口子类和底板子类。窗口子类包括对话框(Dialog)类、文件对话框(FileDialog)类和独立窗口(Frame)类，是提供承载组件的容器。独立窗口类生成通用的非模态窗口，对话框类生成模态窗口。底板将组件组织进一个已存在的窗口区域。

图 1.2 显示了一个包括有各种 AWT 组件类的 applet(小程序)的运行，读者可以从中看到各种 AWT 组件的显示特性。

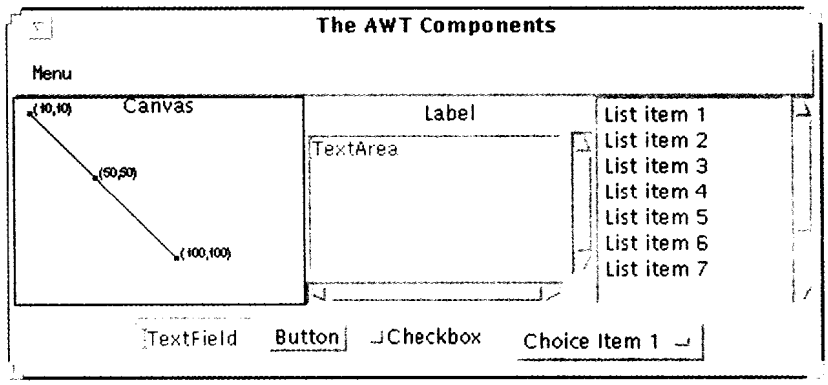

图 1.2 AWT 组件类显示示例

1.2 其他 AWT 类

AWT 中不仅包括各种组件类，而且还包括其他一些关于界面显示和事件处理的类，这些类主要包括：

- 布局管理器类。它用于控制容器中的组件在屏幕上显示的大小和位置。

- 表示尺寸和形状的类。尺寸类如 Dimension 类指定矩形区域面积;形状类如点类、矩形类、多面体类。
- 颜色类。它用于表示和处理颜色,定义了常用的颜色常量,如 Color.black。通常情况下,颜色的表示使用 RGB 格式,但也可以使用 HSB 格式。
- 图像类。它提供了表示图像数据的方法。applet 可以使用 getImage()方法获取 GIF 和 JPEG 图像,非 applet 程序则使用另一个类:ToolKit 获取图像。
- Font 和 FontMetrics 类。它用于控制程序中文本的显示。Font 类可以获取基本的字体信息,并生成表示各种字体的对象。使用 FontMetrics 对象可以得到详细的特定字体的尺寸信息。可以使用组件和图形的 setfont( )方法设置组件使用的字体。
- 图形类和事件类。它们是 AWT 界面显示和事件处理系统的核心。图形类表示了一个显示现场——不使用图形对象,程序就无法绘制屏幕。事件对象处理用户的动作,例如按下鼠标键。

1.3 组件体系结构

AWT 的设计目标是实现与平台无关的 API,但同时保持与组件平台相关的显示特性。例如,AWT 只有一个按钮 API,但按钮在 Macintosh 机器和 Windows 95 的微机上就有不同的显示形式。这个目标是通过同时提供平台无关性的组件类和平台相关的 peer(对等组件)来实现的。换句话说,每个 AWT 组件类有一个等价的 peer 类,每个组件对象有一个 peer 对象控制它的显示特性。

图 1.3 说明了一个典型的 AWT 组件(按钮)是如何映射到它的 peer 的。图中 ButtonPeer 是一个平台相关的类,它实现了 java.awt.peer 的 Buttonpeers 接口,Java.awt ToolKit 类在 Windows NT/95 ButtonPeer、Macintosh ButtonPeer 和 Motif ButtonPeer 中选择一个类作为 ButtonPeer 的具体实现。

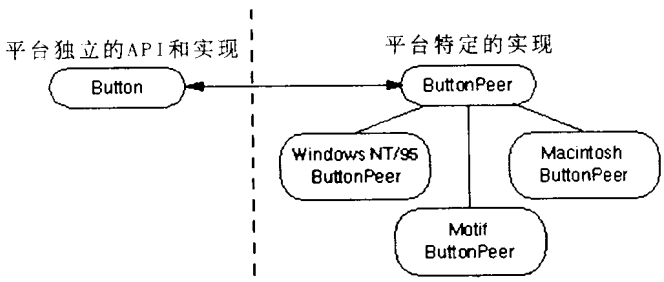

图 1.3 AWT 类到 peer 的映射

1.3.1 peer 的生成

peer 只在其相应的组件对象显示时才首次生成。当增加一个组件到一个不可见的容器时,这个容器是没有 peer 的,只有在此容器第一次显示时,它的 peer 和它包含的所有组件的 peer 才生成。

然而，当增加一个组件到一个可见的容器时，则需要由程序显式地告诉 AWT 生成一个组件的 peer，这可以通过调用 validate()方法来实现。程序可以直接在加入的容器上调用 validate()方法，但通常在容器中调用更好，因为在容器中调用 validate()方法可以引起一个连锁反应，从而简化了编程。

1.3.2 peer 的事件处理

peer 会响应用户的输入事件并相应地改变用户界面组件的显示特性。例如，当用户按下一个按钮时，peer 就会响应这个鼠标事件，改变该按钮的显示特性同时发送一个动作事件给相应的按钮对象。

理论上，peer 处于事件处理链的末尾。当产生一个事件时(如按键)，事件所对应的组件首先处理此事件，然后(若组件事件处理句柄返回 false)组件的容器处理此事件，等等，直到组件层次中的所有组件都处理过此事件(并且事件处理句柄都返回 false)后，peer 才处理此事件。

对于一些基本事件，如按键和按鼠标键，peer 有时会产生更高级的事件——动作(action)、焦点变换、窗口最小化等等。这些高级事件也被传递到相应的组件去处理。

1.4 AWT 显示机制

在 Java applet 中，AWT 对象是在 WWW 浏览器中被调用的。当一个 applet 对象被浏览器实例化时，它将被调整成 HTML 文档中 < APP > 标记所设定的大小，并嵌入到 WWW 页面的内部。

当需要为 applet 建立用户界面时，可以直接实例化必要的用户界面组件，并把它们加入到 applet 中去。这时，applet 被作为容器使用。为了使屏幕组件能以合适的大小显示在容器内适当的位置上，编程人员必须控制用户界面中各组件的布局。AWT 为此提供了专门的布局类。

使用布局类来设计组件的外观有两个好处：首先，组件的显示往往同具体平台相关，利用布局类来管理布局就可以实现平台的独立性；其次，利用布局类来管理组件，还可以保证容器大小发生变化时，组件仍然能正确显示。

总之，Java 的 AWT 类库提供了多个被称为组件的用户界面对象。注意容器是一种可以容纳其他组件的特殊组件。如果容器中包含有组件，它将利用一个布局类来组织这些组件，并控制组件的显示大小和位置。

独立应用程序中 AWT 类库的显示机制与 applet 中的基本相同。

1.5 Java GUI 程序结构及示例

本节通过对一个 Java GUI 程序例子的分析，讲述 Java GUI 程序的基本结构，使读者在进入后续章节前能对 Java GUI 程序设计在总体上有个更加深入和全面的了解。

1.5.1 例子

此程序进行长度的单位换算:公制(米)和美国制(英尺)的换算。当用户在标有 Metric System 的文本域中输入了一个长度数据(单位米),按下回车键或单击 Centimeters 按钮后,程序就在下面的文本域中显示单位换算后的长度值(单位英尺);反之,当用户在标有 U.S. System 的文本域中输入了一个长度数据(单位英尺),按下回车键或单击 Inches 按钮后,程序就在上面的文本域中显示单位换算后的长度值(单位米)。图 1.4 是程序运行结果显示。

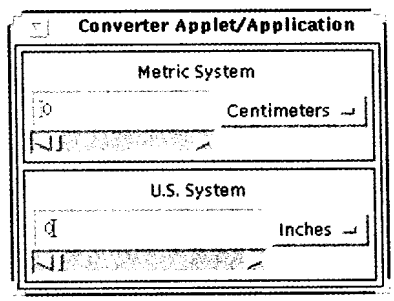

图 1.4 例子程序运行显示

1.5.2 程序源代码

程序的源代码见程序清单 1.1,读者可以通过阅读此代码进一步理解本章所讲述的基本原理。

程序清单 1.1

```
import java.awt.*;
import java.util.*;
import java.applet.Applet;

public class Converter extends Applet {
    ConversionPanel metricPanel, usaPanel;
    Unit[] metricDistances = new Unit[3];
    Unit[] usaDistances = new Unit[4];

    /**
     * 生成 ConversionPanels(一个用于公制,另一个用于美国制)
     */
    public void init() {
      //使用含两行的 GridLayout
      //填充区为五个像素
      setLayout(new GridLayout(2,0,5,5));

      //生成公制的 Unit 对象并用它们初始化一个 ConversionPanel 对象
      metricDistances[0] = new Unit("Centimeters", 0.01);
       metricDistances[1] = new Unit("Meters", 1.0);
       metricDistances[2] = new Unit("Kilometers", 1000.0);
        metricPanel = new ConversionPanel(this, "Metric System",
```

```
        metricDistances);

      //生成美国制的 Unit 对象并用它们初始化一个 ConversionPanel 对象
      usaDistances[0] = new Unit("Inches", 0.0254);
      usaDistances[1] = new Unit("Feet", 0.305);
      usaDistances[2] = new Unit("Yards", 0.914);
      usaDistances[3] = new Unit("Miles", 1613.0);
      usaPanel = new ConversionPanel(this, "U.S. System", usaDistances);

      //把两个 ConversionPanels 加到 Converter 对象中
      add(metricPanel);
      add(usaPanel);

    //调用 validate()方法
      validate();
    }
    /**
     * 进行从公制到美国制的转换或相反的转换并更新相应的 ConversionPanel 对象
     */
    void convert(ConversionPanel from) {
      ConversionPanel to;

      if (from == metricPanel)
            to = usaPanel;
      else
            to = metricPanel;
      double multiplier = from.getMultiplier() / to.getMultiplier();
      to.setValue(from.getValue() * multiplier);
    }

    public void paint(Graphics g) {
      Dimension d = size();
      g.drawRect(0,0, d.width - 1, d.height - 1);
    }

    public Insets insets() {
      return new Insets(5,5,5,5);
    }

    public static void main(String[] args) {
      MainFrame f = new MainFrame("Converter Applet/Application");

      Converter converter = new Converter();

      converter.init();

      f.add("Center", converter);
      f.pack();
      f.show();
    }
}

class ConversionPanel extends Panel {
```