microsystems

Java 2 图形设计

卷 I: AWT

Graphic Java 1.2 Mastering the JFC Volume I: AWT
(3rd edition)

(美) David M. Geary 著 马欣民 等译

机械工业出版社
Prentice Hall

软件开发技术丛书

Java 2 图形设计

卷Ⅰ：AWT

（美）David M. Geary 著

马欣民 等译

王 英 校

机械工业出版社
China Machine Press

本书详细介绍 Java 中的 AWT（抽象窗口工具集）。主要内容包括同位体、轻量构件、剪贴板与数据传输、颜色、无鼠标操作、打印、布局管理器、国际化、串行化、滚动性、字体与字号以及 AWT 中的各种构件。

本书版权登记号：图字：01－1999－2352

图书在版编目（CIP）数据

Java 2 图形设计：卷Ⅰ，AWT/（美）吉瑞（Geary，D. M.）著；马欣民等译. --北京：机械工业出版社，2000.1
（软件开发技术丛书）
ISBN 7－111－07745－8
书名原文：Graphic Java 1.2 Mastering the JFC Volume Ⅰ：AWT，3rd edition.

Ⅰ. J... Ⅱ. ①吉 ... ②马 ... Ⅲ. JAVA 语言－程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（1999）第 55437 号

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）
责任编辑：陈剑匡
北京第二外国语学院印刷厂印刷 · 新华书店北京发行所发行
2000 年 1 月第 1 版第 1 次印刷
787mm × 1092mm 1/16 41 印张
印数 0 001－5000 册
定价：79.00 元（附光盘）

前　言

Java 语言可以说是到目前为止推广最快的一种计算机语言，自从 1990 年 SUN 公司开始设计，到现在在全球流行，也只不过是短短的几年时间。Java 语言是从 C + + 中衍生而来，它继承了 C + + 的大量语言成分，同时抛弃了 C + + 中冗余的和容易引起问题的成分，Java 集面向对象、与平台无关、稳固性、安全性、多线程性等多种特征于一身，是一种适合于分布式计算的新型面向对象程序设计语言。

JDK 2 与以前版本的 JDK 相比，在开发图形用户界面、图形 applet 以及应用程序等各个方面都提供了更加强大的支持。在 JDK 2 中，用户完全可以利用 Java 基本类完成整个开发过程。Java 基本类由 4 个 API 组成，分别为 AWT、Swing、辅助 API 以及 2D API。

其中，抽象窗口工具集（AWT）是 Java 最基本的用户界面工具集，它不仅提供标签、按钮以及滚动条等基本构件集，而且还提供授权事件模型、布局管理器、数据传输支持以及轻量构件支持等大量图形构件的基础结构。

Swing 是基于 AWT 基本结构创建的二级用户界面工具集。与 AWT 相比，Swing 提供了更加丰富的构件集。Swing 共提供 40 个构件，是 AWT 构件集的 4 倍。另外，Swing 中的标签以及按钮等构件设计可用于取代 AWT 中的相应构件，而树与表等高级构件则可以提供 AWT 不支持的其他功能。

辅助 API 允许用户开发伤残者也可以顺利使用的 Java applet 以及应用程序，而且还是用于移动用户界面开发的强有力的可选工具。例如：如果 Java 导航系统可以按照声音方式（而不是按照地图显示方式）提醒司机当前车辆所在的位置等信息，则可以极大程度上方便司机用户的使用。

2D API 包含许多重要的对 AWT 图形模型的改进，例如：颜色、形状、图像以及文本管理等。

本卷的目的

JDK 2 中提供了 Swing 构件集，其中包含了一个完整的轻量构件集，可用于替代 AWT 中的重量构件。由此，用户普遍会产生 Swing 是 AWT 替代物的误解，而实际的情形并不是这样的。实际上，Swing 是在 AWT 之上创建的，而且每个 Swing 构件都是一个真正的 AWT 构件。因此，为了准确掌握 Swing 构件集的工作方式，读者首先必须掌握 AWT 构件的行为及其工作原理。

本卷正是基于这样的背景而产生的，其最重要的目的是帮助用户掌握 AWT，它将详细介绍 AWT 的各种基础概念与高级概念。当然，本书不仅提供了每个 AWT 构件的详尽描述，而且还提供了 AWT 的一些技巧，并确实做到面面俱到。

本卷的内容

通过对本卷的阅读，读者不仅可以全面掌握 AWT 的设计思想，而且还可以掌握如何才能使这种设计思想在应用程序开发过程中发挥最佳效应。

本卷主要介绍 AWT 中的基础概念与高级概念，而没有介绍 AWT 软件包中包含的其他 JFC

API，即 2D API。如果需要了解 2D API 的详细内容，读者可参阅本套书中的第 3 卷。以下是本卷中涉及的主要内容：

同位体

读者可了解到 AWT 中的同位体结构以及同位体方法的利弊。

剪贴板与数据传输

介绍了 AWT 的数据传输模型，以及如何使用系统剪贴板与本地剪贴板等工具实现数据传输操作。虽然 AWT 构件只对字符串的传输提供直接的剪贴板操作支持，但是利用 AWT 构件也可以实现其他类型数据的传输操作，因此我们将以图形与自定义构件的传输为例，介绍利用剪贴板实现其他类型数据传输的方法。

轻量构件

介绍如何创建自定义轻量构件、如何利用双缓冲容器技术拖曳自定义构件，以及如何在演示区域内实现自定义轻量构件的动画效果等内容。通过相应内容的学习，读者不仅可以掌握双缓冲的工作机制，以及为什么应该将轻量构件放置在双缓冲容器内等问题，而且还可以了解将轻量构件放置在容器中时常容易犯的错误，以及轻量容器为什么必须手工实现与布局管理器的匹配等内容。

布局管理器

详尽介绍布局管理器的相关内容，主要包括：GridBagLayout 以及如何创建自定义布局管理器等。同时，还将介绍如何强制容器安排其包含的构件及其必要性。

国际化与串行化

介绍图形用户界面的国际化方法，以及 AWT 构件及其事件监听者的串行化方法。

滚动性

介绍利用滚动框构件滚动容器内任意数量构件的方法、滚动框类的限制以及无同位体滚动框架存在的必要性等内容。同时，我们还将讨论利用图形 Java 工具集创建无同位体滚动框架的方法，用户在实际应用过程中，可根据实际环境的需求更改该程序使其满足需要。

其他内容

讨论事件授权模型、颜色、无鼠标操作以及打印等内容，另外还将详细介绍滚动条、菜单（包括弹出式菜单）、图像管理、图形、字体与字号、对话框以及文本框等构件。

读者对象

本书适用于利用 Java 开发面向对象程序的用户。由于关于 Java 语言细节及其如何与 VB、C、C++ 等语言组合使用的书籍相当多，因此本书不讲解 Java 语言的细节问题。

互联网信息资源

与 Java 相关的在线资源很多。如果需要访问在线帮助信息，可访问 Sun 公司的主页：

http：//java.sun.com/

如果希望查阅 Java 新闻信息，则可访问 Java 新闻组地址：

comp.lang.java

另外，互联网上还有供 Java 爱好者交换意见、提出问题以及解决方案的邮件列表，如果需要查看这些邮件列表，可访问网址：

http：//java.sun.com/mail.html

从这些新闻组和网站，读者可以浏览其他许多与 Java 相关的资源、帮助信息、FAQ（常见问题及解答）以及在线杂志等信息。

另外，我们还特别设置了本书的更新信息网站，利用该网站同时还可以查阅 SunSoft 出版社的其他 Java 系列丛书的信息。该网站的地址为：

http：//www.sun.com/books/books/Geary/Geary.html

本书英文原书书号为 ISBN 0－13－079666－2，当你访问以上网站时可能会用到这个书号。

最后，如果需要设计精美的 Java 图片，则可以访问网站：

http：//www.pixelsight.com：80/PS/pixelsite/pixelsite.html

本书约定

为了进一步规范本书，我们在本书中使用了一系列的代码约定，如下所示：

约　定	示　例
类名称中每个单词的首字母大写	public class LineOfText
方法名称中第一个单词的首字母小写，其余单词的首字母大写	public int getLength（ ）
变量名中第一个单词的首字母小写，其余单词的首字母大写	private int length private int bufferLength
static 变量由下划线开头	protected static int _ defaultSize = 2;

注意　本书中提到方法和函数时一般都不带参数，但是当需要讨论这些参数的实际含义时，则将包含相应的参数信息。

参与本书翻译的人员有：马欣明、王兴军、孙克伟、朱家义、陈波、石鹏飞、石丽、于峰、张展等。他们都是多年从事 Java 编译器研究的人员，都十分精通和熟悉 Java 语言。

注：Java 2 以前称为 Java1.2，实际上这两个版本是完全一样的。本书中，我们统一采用“Java 2”这个提法。

目　录

第三部分　事件和布局管理器

第 8 章　基于继承的事件处理

在本章中，我们要讲述的是初始的 AWT 事件模型。在新出的 1.1 版本的 JDK 中，AWT 对事件模型有了很大的改进。我们在本书中保留这一章是因为 AWT 仍然支持旧版本的事件模型。但是，在以后的发行的 AWT 版本中，将不再支持旧版本的事件模型，所以，如果你对旧事件模型的有关信息知道得非常清楚，你可以忽略本章而直接进入下一章。在下一章中，我们要讨论新事件模型。在本章的最后，我们增加了一节，用来讲述旧事件模型的缺点和如何使用新事件模型对它们进行修改。在本书以后的版本中，这一章的内容将会被删除掉。

现代图形用户接口一般是事件驱动，这就意味着它们将花费大量的空闲时间，用来等待事件的出现。当事件发生时，事件处理程序将对事件做出反应。在 AWT 中 applet 程序和应用程序是不同的，所以在本章中，我们介绍 AWT 构件中的事件处理，并提供几点说明，包括何时覆盖事件处理方法和以及何时处理事件或将之传递。在本章中，我们还介绍几个 applet 程序实例，对事件处理技术进行说明，如检测鼠标的双击、管理动作事件、产生事件并向自定义构件传递等。

8.1　旧 AWT 事件模型

和几乎所有的面向对象用户接口工具包一样，AWT 属于事件驱动。事实上，因为 AWT 使用了同位体方法，所以基础的事件子系统都是使用本地平台来运行 applet 程序或应用程序。例如，如果在 Motif 窗口中运行 applet 程序，则它使用的是 X Window 事件处理系统。然而，我们讨论的焦点并不是上面所说的本地事件系统，而是我们对事件做出反应的 AWT 层。

要注意本章讨论的是 AWT 1.0 和 AWT 1.02 版本中的原先的、基于继承的事件模型。当前的 AWT 版本[⊖]含有一种新的事件模型，该事件模型以基于授权代替基于继承并改善了旧事件模型。我们建议你使用新的事件模型代替旧事件模型。事件模型是向后兼容的，因此如果你以前接触过原先版本的事件模型，你可以忽略本章。

关于授权事件模型，我们将在下面的一章中进行讨论。如果一定要使用旧的、基于继承的事件模型，那么本章内容是很适合于你的；如果你能使用授权事件模型，那么你就可以忽略本章，直接进入下一章的学习。

8.1.1　覆盖事件处理方法

从本质上来讲，使用旧 AWT 事件模型处理事件是相当简单的：当事件在构件中出现时，构件中的一个方法被调用。如果你想让构件响应事件，你可以简单地覆盖适当的方法。例如，调整构件大小时，你可以调用构件中的 void setSize（int w，int h）方法；如果你想在调整大小

⊖ 1.1 以后的版本。

事件发生时做一些动作，你应该覆盖 setSize（）并处理其事务。在 AWT 中，有很多 setSize（）这样的构件方法，可以用来响应事件。在下面的表 8-1 中，我们列出了这类方法中的一部分。

表 8-1　构件中使用的驱动事件方法

方　法	事　件
常覆盖的构件方法	
void setSize（int w，int h）	构件已经被调整大小
void setBounds（int x，int y，int w，int h）	构件被调整大小并/或移动
void paint（Graphics g）	构件需要被画出
void update（Graphics g）	默认情况下，清除并重绘
void addNotify（）	创建同位体
不常覆盖的构件方法	
void setLocation（int x，int y）	将构件移动到 x，y 位置
void layout（）	构件需要被布置
void validate（）	如果非法，则布置构件

在表 8-1 中，第一部分中所列举的方法是我们在开发自定义构件时经常覆盖的方法。而在第二部分所列举的方法则针对一些不常见的事件，例如需要布置构件——请参见后面的第 10 章“构件、容器和布局管理器”中相关的内容。几乎不需要覆盖和这些事件相关的方法。当然，通过覆盖在表 8-1 中所列出的方法，你可以改写构件的默认方法。你将会经常看到在增加改进之前，调用超类版本的方法对事件进行处理：

```
public addNotify () {  //Default version creates the peer
    super.addNotify ();    //Peer created by this call
}
```

8.1.2　被传播的事件

在上面，我们讨论的事件发生在构件中，它们在发生事件的构件中被处理（或忽略）。这些事件仅仅涉及到它们所发生的构件。

另一类事件被称为被传播的事件，无论被传播的事件什么时候发生在构件中，构件的 handleEvent（Event）方法被调用。构件的 handleEvent（Event）方法可以选择是向其容器传播事件还是完全在其本身中处理事件。

注意被传播的事件和在表 8-1 中列出的事件之间的差别。被传播的事件一般是通过构件的 handleEvent 方法来处理的，一个 java.awt.Event 被传递给这个方法。被传播的事件可以传播给构件的容器；假如这样的话，容器的 handleEvent 方法被调用，并且 java.awt.Event 被向前传递。

因为被传播的事件涉及一个 java.awt.Event，所以它们一般是引用 AWT 事件。在本章的剩余部分，我们仅仅讨论和这种事件相关联的部分。

8.1.3　事件类型常量

每个 java.awt.Event 都有一个对应的 id 域，该 id 域的值指示已经发生的事件的类型。通常，构件的 handleEvent 方法做的第一件事是检测事件的 id 域来判断事件的类型。例如，一个和鼠标事件相关的 handleEvent 方法可能像下面这样：

```
public boolean handleEvent (Event event) {
    if (event.id = = Event.MOUSE _ DOWN) {
        //react to mouse down
        return true; //event fully handled, do not propagate
    }
    else if (event.id = = Event.MOUSE _ UP) {
        //react to mouse up
        return true; //event fully handled, do not propagate
    }
    //let superclass handle event and decide to propagate
    return super.handleEvent (event);
}
```

注意在上面所列举的例子中，如果事件是鼠标按下或释放事件，则其返回值是 ture，表示我们已经完全完成处理事件，并不希望事件传播给构件的容器。如果事件不是鼠标按下或释放事件，将使用超类对事件进行处理。如果返回的值是 false，则表示事件将直接被传播给构件的容器，并且不给超类机会处理事件。所以，如果覆盖的 handleEvent 方法没有完全完成事件，最好返回 super.handleEvent (event)，而不要直接将事件传播给构件的容器。

在下面的表 8-2 中，完整的列出了 java. awt.Event 类型常数。

表 8-2 java.awt.Event 常数

事 件 常 数
窗口事件常数
WINDOW _ DESTROY
WINDOW _ EXPOSE
WINDOW _ ICONIFY
WINDOW _ DEICONIFY
WINDOW _ MOVED
键盘事件常数
KEY _ PRESS
KEY _ RELEASE
KEY _ ACTION
KEY _ ACTION _ RELEASE
鼠标事件常数
MOUSE _ DOWN, MOUSE _ UP
MOUSE _ MOVE
MOUSE _ ENTER, MOUSE _ EXIT
MOUSE _ DRAG
滚动条事件常数
SCROLL _ LINE _ UP, SCROLL _ LINE _ DOWN
SCROLL _ PAGE _ UP, SCROLL _ PAGE _ DOWN
SCROLL _ ABSOLUTE

（续）

事　件　常　数
列表事件常数
LIST _ SELECT，LIST _ DESELECT
动作事件常数
ACTION _ EVENT
LOAD _ FILE
SAVE _ FILE
GOT _ FOCUS，LOST _ FOCUS

AWT 技巧：避免从 handleEvent 方法中返回 false 值

从 handleEvent 方法中返回 false 会把事件传播给构件的容器，而不让构件的超类有机会处理事件。所以避免从 handleEvent 方法中返回 false 是一个好的思想。如果类没有完全完成处理某个事件，那么，它将给它的超类一个机会来处理事件并由超类决定是否将事件传播给构件的容器。即使你知道你的超类对某个类型的事件不感兴趣，也应这么做，因为超类可能在未来会被改变以处理这种事件。

被传播的事件处理方法

在覆盖的 handleEvent 方法中，检查事件的 id 域并判断出事件的类型，这是相当难看的，而且需要保存在表 8-2 中列出的所有事件常数，表 8-2 中列出的常数并不都是面向对象的[⊖]。因此，java.awt.Component 实现了一个 handleEvent 方法用于分析事件的类型，并调用在表 8-3 中列出的一个辅助方法。表 8-3 中列出了可以供 java.awt.Component.handleEvent（）调用的方法。

表 8-3　被传播的事件的处理方法

方　法	事　件
boolean action（Event，Object）	动作事件
boolean mouseUp（Event，int，int）	释放鼠标
boolean mouseDown（Event，int，int）	按下鼠标
boolean mouseDrag（Event，int，int）	拖动鼠标
boolean mouseMove（Event，int，int）	移动鼠标
boolean mouseEnter（Event，int，int）	鼠标进入构件
boolean mouseExit（Event，int，int）	鼠标离开构件
boolean keyUp（Event，int）	按键释放
boolean keyDown（Event，int）	按键被按下
boolean gotFocus（Event，Object）	构件得到焦点
boolean lostFocus（Event，Object）	构件失去焦点

当处理被传播的事件时，你可以选择下面给出的两个方法中的一个：

⊖ 请参见 Meyer、Bertrand 所著的“Object Oriented Software Construction”的 10.2.2 节。