Java 1.1 Developer's Handbook

Java 高级开发指南

〔美〕 Philip Heller Simon Roberts
Peter Seymour Tom McGinn 著

邱仲潘 等译

SYBEX

PHEI

電子工業出版社
Publishing House of Electronics Industry
URL:http://www.phei.co.cn

Java 1.1 Developer's Handbook

Java高级开发指南

〔美〕Philip Heller　Simon Roberts　著
Peter Seymour　Tom McGinn

邱仲潘　等译

電子工業出版社

Publishing House of Electronics Industry

内 容 提 要

这是由几位具有丰富教学经验的Java教员编写的一本Java高级开发指南。

全书共分四大部分。第一部分：基础篇，介绍Java环境、小程序和应用程序、构件、布置管理器和移植性等问题；第二部分：高级问题，介绍图形、线程、动画、文件和流、联网、数据库访问、分布式对象和内容协议处理器等问题；第三部分：新API，介绍JDK V1.1中的新功能；最后部分为三个附录，分别介绍了核心Java类的轻量级API清单，Javadoc，FAQ（常见问题）的答案。

本书适用于计算机软件开发人员及相关人员。

书　　名：**Java高级开发指南**

著　　者：〔美〕Philip Heller　Simon Roberts　Peter Seymour　Tom McGinn

译　　者：邱仲潘等

责任编辑：张盛华

印 刷 者：北京顺义颖华印刷厂

装 订 者：三河金马印装有限公司

出版发行：电子工业出版社出版、发行

北京市海淀区万寿路173信箱　邮编：100036　发行部电话：68279077

北京市海淀区万寿路甲15号南小楼三层　邮编：100036　发行部电话：68215345

URL:http://www.phei.co.cn

经　　销：各地新华书店经销

开　　本：787×1092 1/16　印张：41.125　字数：1070 千字

版　　次：1997年10月第1版　　1997年10月第1次印刷

书　　号：ISBN 7-5053-4264-9/TP・1933

定　　价：63.00 元

著作权合同登记号　图字：01-97-0622

凡购买电子工业出版社的图书，如有缺页、倒页、脱页者，本社发行部负责调换

致　　谢

感谢Ally Bailey、Cindy Lewis、Carol Stille、Georgianna Meagher、Mike Bridwell、Ray Moore和Heather Young博士的支持，感谢Tim Russell、Russel Taylor、Devon Tuck、Tim Lindholm和Urs Eberle丰富的知识和耐心的解释，感谢Jennifer Sullivan将我们几位拉在一起并编辑了某些章节。

感谢图书策划人Suzanne Rotondo、副主编Krista Reid-McLaughin、主编Laura Arendal和编辑Maureen Adams与Steve Gilmartin对我们唠叨的耐心和对本书认真的编辑。感谢技术编辑John Zukowsri将我们的手稿进行技术处理，感谢生产主管Robin Kibby的校读和组织技巧。感谢电子出版专家Franz Baumhackl迅速而娴熟的排样。

特别感谢Emily Roberts提供的精彩图形（见光盘文件javadevhdbk\ch06\Emily.gif）。

前　言

在前言中，我们想把作者、读者和本书拉到一起来。

首先作个自我介绍，我们一个住在西海岸、两个住在东海岸、一个住在英格兰。我们不仅居住的地域分布很广，接触的面也很广。我们的职业是Java教员，我们向上千人介绍过Sun公司的Java类。

我们不仅讲学，而且询问学生学习Java的用意，答案常常出人意外。我们说，同学们好，请翻开教材第一页。学生会说，这些我们已经知道了，但怎么通过TCP/IP将小程序连接到数据库呢？上练习课时，我们说，请做第32页的实验。学生会说，我上次已经做过了，可我编写自己的布置管理器时却行不通，为什么呢？

换句话说，我们接触了Java的实际应用。我们伴着上千名Java学员进行了学习和体验。他们就会告诉我们。

我们用手电筒、鸡毛掸子、API页面和源码探索了Java的各个边边角角。本书将把我们见到的东西告诉你。

下面轮到你们自我介绍了，编写本书时，我们心中的读者是谁呢？

我们假设你已经读过第一本Java书籍或用过第一个Java类，已经知道一些理论，知道Java语言的语法，知道对象，知道大部分软件包的主要基础，已经学会在API文档中查找类和方法说明，已经编写小程序和应用程序，已经对Java的一般问题有所了解，准备进入边边角角的问题了。我们要提供的正是这时所要的手电筒和鸡毛掸子。

这是一本深入内幕的书籍，没有关于Internet历史的介绍，没有关于对象多态的说明，James Gosling（创立Java的人）的大名也不再提及。

编写本书时，我们遵循了几个原则。第一，每一章先分析相关背景材料，然后进入正题。如果你已经对基础有所了解，可以跳过第一或第二节。

使用样本程序码时，我们想尽办法使它切合正题。讲到线程时，样本码是用于说明线程的，而不是卖弄我们的技巧的。阅读长长的源码比较费劲，所以我们尽可能将码段缩短，但在实际编程中，为了使程序完整，有时需要很长的程序。本书的程序码力求简短，但同时又保证其功能的完整性。

所有源码和类文件都放在光盘中。我们尽可能将样本码设计成小程序，并包括简单的Web页面，便于读者运行。有时这个办法行不通，有时则需要用应用程序来执行（特别是在与网络和文件系统相关时）。

本书引进一种新的样本程序，称为Labs（实验室）。本书不介绍Labs的源码（但光盘中有提供）。这些labs的学习不是通过阅读源码，而是通过执行小程序进行。例如，第4章布置管理器部分有个Lab叫做GridBagLab。为了直观地掌握GridBag布置管理器，唯一的办法是建立许多布置，混合各种栅格包限制数值的组合。其中一种生成多个组合的办法是编写大量程序码。但是，使用GridBagLab，只要使用用户接口选择限制值，单击取一个钮，再看看结果即可。这样，就可以把Labs作为教学工具，在很短的时间内得到直观的体验。

先就介绍这些，很高兴与你们认识。下面简单介绍一下本书的背景细节。

1. 光盘

本书及配套光盘中的程序码用JDK（Java开发工具库）V1.1的beta工版编译过。光盘中有个目录叫javadevhdbk，其中每个子目录包含本书一章的源码，它的应用程序可以通过命令行执行，而小程序则要通过浏览器执行。每个小程序有个简单的html文件。文件名类似于小程序的子类名。由于商业化浏览器技术尚未支持JDK1.1，建议用小程序浏览器浏览小程序。这些应用程序和小程序可以在任何支持JDK1.1的平台上运行。

2. 本书的组织

本书分成四个部分。第一部分，基础篇（1~5章），介绍Java环境、小程序和应用程序、构件、布置管理器和移植性等问题。尽管这些问题是基本问题，但介绍得有一定深度和广度。第二部分，高级问题（6~14章），介绍图形、线程、动画、文件和流、联网、数据库访问、分布式对象和内容/协议处理器等问题。第三部分，新API（15~17章），介绍JDK（Java开发工具库）V1.1中的新功能。Sun系统公司将这些新功能放进了应用程序编程接口中。本书介绍的API包括JavaBeans、电子商业和小服务。本书最后部分是三个附录：第一个附录是所有核心Java类的轻量级API清单；第二个附录介绍Javadoc；第三个附录介绍FAQ（常见问题）的答案。

3. 规则

本书通过各种规则的使用，更合理地表达各种信息。通篇用下面所示的提示、说明和警告以引起读者对特定问题的注意。

提示：这是提示，包含具体编程技巧。

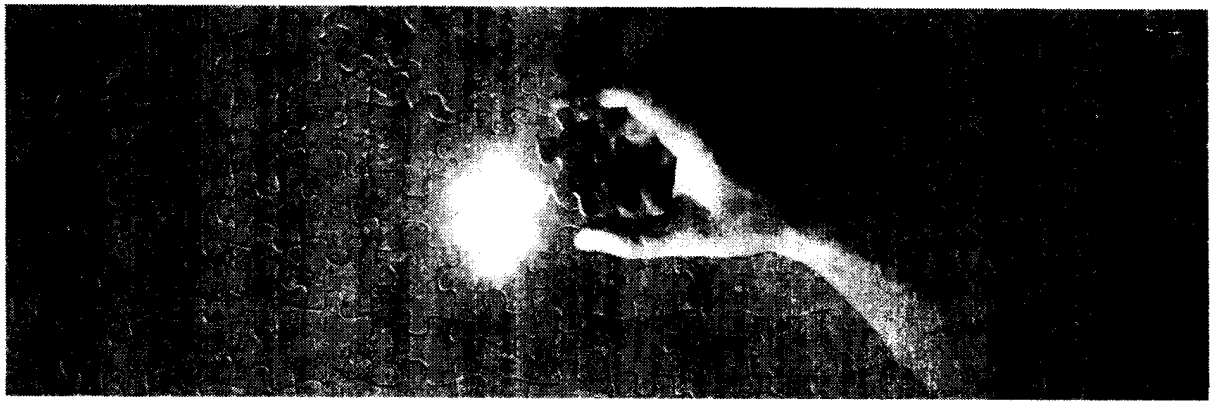
说明：这是说明，包含重要问题旁白。

警告：这是警告，引起对错误、设计疏忽和其它问题的注意。

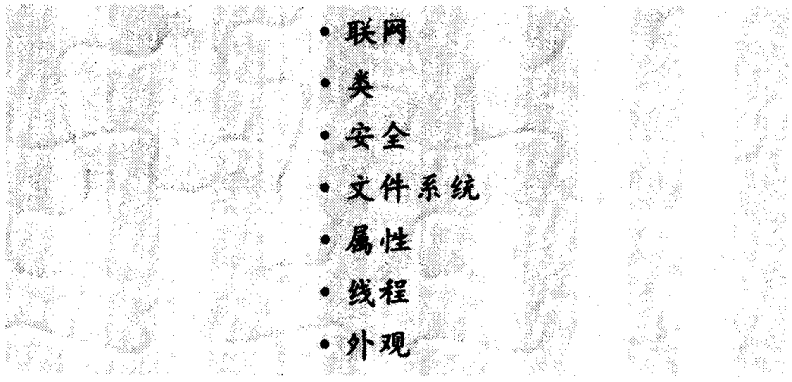
这些规则用于使阅读本书更加方便。我们对本书的出版深感高兴和激动，希望你的学习富于成果和趣味。

第一部分 基 础 篇

这几章深入介绍Java编程环境的基本问题和技巧。也许你对这些内容已有所知，但这里将提供前所未有的细节。例如，你无疑用过构件和布置管理器，但你未必建立过自己的管理器。



第1章 Java环境



Java程序运行在真实平台上实现的假想机中。Java程序的运行要靠环境的大量支持。本书首先介绍Java环境问题：联网、类、安全、文件系统、属性、线程和外观。编程人员充分理解这些材料有助于开发程序和理解Java开发工具库（JDK）的工作。

1.1 联网

Java环境中最明显的构件是网络。小程序是在Internet或Intranets上活动的。小程序和应用程序通过类的启动在网络上用UDP或TCP/IP协议进行通信。Java的类装入器是用于帮助网络操作的。Java的安全机制是用于防止来自网络上的攻击的。小程序中隐含的许多限制都是为了防止不良网络行为的。为了建立健全的网络小程序和应用程序，一定要充分理解Java联网类提供的功能和Java安全模型所隐含的限制。第10章联网部分介绍Java的联网功能。

1.2 类

每一行可执行Java码都属于一个或另一个类定义。Java用类表示一切，如钮、小程序、URL、文件，甚至类本身。本节介绍类的装入及类装入在Java安全机制中所起的作用。

1.2.1 类的装入

考虑下列最简单的小程序：

```
public class SimplestApplet extends java.applet.Applet
{
}
```

这个程序码绝对是什么也不干的，但研究这个什么也不干的过程却很有意义。首先，作为小程序，应当在HTML页面中进行介绍，才能让浏览器下载和执行：

```
...
<APPLET CODE=SimplestApplet.class WIDTH=200 HEIGHT=150>
</APPlET>
...
```

用户浏览这个页面时，浏览器的HTML翻译器对其进行分析。APPLET标志表明要装入Java类。

缺省情况下（没有CODEBASE标志时），浏览器寻找与HTML页面同一目录中的小程序，即寻找文件SimplestApplet.class，如果找到该文件，则将其装入浏览器中。

这时小程序定义文件进行全面安全检查，通过字节码验证器检查类定义是否符合安全标准。例如，每个操作码都要有效，并且带有正确数量的有效变元，必须遵循访问限制，不能有上溢和下溢的操作数堆栈。

说明： Java编译器将Java源码转变成独立于平台的字节码，可以将其想象成JVM（Java虚拟机）的机器语言。和实际硬件的机器语言一样，字节码也是一系列指令。每个指令包括一个操作码和一些操作数。

一旦字节码验证器验证合格，浏览器中的Java运行环境必须建立SimplestApplet的内部表达。由于每个Java类（除Object）都是从其它类扩展而来的，所以建立内部表达要两个步骤：

- 建立上级类的表达（如果还没有）。这个过程可以递归（除非上级类是Object）。
- 子类所属的新数据和功能的表达。

第二步很容易：类装入器只要从网络上装入SimplestApplet类即可。第一步要表达上级类Applet。假设浏览器的Java运行环境中还没有Applet类的表达，则必须装入这个类。但Applet来自Panel类的扩展，Panel来自Container，而Container又来自Component，所以这些类都要装入。那么，Java是从何处寻找要装入的类呢？Java用下列搜索方法寻找类的定义：

- 搜索Java运行系统自己的类定义集。
- 如果没找到，则按ClassPath指示的路径搜索本地文件系统中的其它位置。
- 如果还没找到，则搜索远程Web服务器。

下面会对这些步骤进一步介绍。

一、核心Java API

每个Java运行环境都有自己的标准Java类备份。有些厂家把这些文件直接放在上面，有些厂家则把这些文件放在比较隐蔽的地方。这些文件通常是压合到一起的，但不进行压缩。无论其形式如何变化，每个运行环境都知道如何找到自己的系统类。

这个方法确保常用类文件随手可得。例如，每个Java程序都需要Object的定义，每个带GUI的Java程序都需要Component的定义。有些类虽然用得较少（如Line NumberInputStream），但一旦需要，一定要保证装入正确的标准版本类。

二、Classpath类

如果系统仓库中找不到所要的类文件，则运行时搜索本地机上的其它位置。如果设置了环境变量CLASS PATH，则根据其中列出的顺序搜索所列的所有目录。CLASSPATH是

一列路径名，在DOS平台上用分号隔开，在其它机器上用冒号隔开。如果没有设置CLASSPAHT，则只搜索当前工作目录。

下列命令在Unix机上设置CLASSPATH:

```
Setenv CLASSPATH /w/x/java/classes:/y/z/morejava/classes
```

下列命令在Dos机上做同样的工作：

```
SET CLASSPATH=C:\w\x\java\classes; C:\y\z\morejava\classes
```

三、远程类

如果本地机上找不到小程序所要的类，则下一步要搜索Web服务器（如果本地机上找不到应用程序所要的类，则搜索失败，因为没有Web服务器）。

缺省情况下，类文件应放在浏览器找到Web页面的同一目录中。这可以用CODEBASE标志覆盖。关于CODEBASE标志和软件包位置的细节，请看第5章。Java用这个三步搜索法达到安全和时间效率。

这个三步搜索法保证比本地类更不安全和更费下载时间的远程类只在本地版本得不到时才采用。

1.2.2 静态初始化

类装入并通过字节码验证器检查后，类声明的任何静态变量都分配存储空间。如果静态声明中包括初始化语句，则也在此时完成初始化工作。例如，如果类声明为Static int rev=6，则分配int的空间并初始化为6。

Java编译器允许类中存在静态码段，即码中没有方法名、参数和返回值，只有一段静态程序码。例如，类定义可能如下：

```
Class MyClass extends MyOtherClass
{
   int                  i, j;
   static double        d = 123.456;

   static
   {
      System.out.println ("MyClass was just loaded.");
      d = d*2.1;
   }
      ...
{
```

类定义中可以有多个静态码段。静态变量初始化之后，按类定义中的顺序执行静态码段。和静态方法一样，静态码段只能引用所在类中的静态实例变量。

静态码段样子古怪，难于阅读，许多编程人员对其不熟悉。不管是否需要，静态码段总是执行，这就增加了维护的风险，所以除非确有必要，否则别用静态码段。静态码段的优势是运行得较早，最常见的用途是在包含本地方法的类中。本地方法动态调用装入的库，而在本地调整用发生之前将库完全装入对性能极为有利。为此，带本地方法的类中具有静态码

段，启动System.loadLibrary()调用。

1.3 安全性

Java有许多保护机制，使客户机免遭恶意小程序的袭击。前面介绍过字节码验证器，它能防止Java运行环境装入有害类文件。一个好的Java编译器不会产生不符合字节码验证器标准的操作码，但字节码也可能是敌人炮制的，这就可以用字节码验证器来判别。

Java小程序对客户机资源的访问比应用程序要少。这不是小程序的天性，而是浏览器的限制。每天浏览器都构造SecurityManager类的实例。进行某个操作之前（如文件或接插访问），包装这个操作的各个类必须取得安全管理器的许可权限。安全管理器提供或拒绝对各种功能的访问，其中文件系统和网络访问是其所控制的两个主要功能。

人们常说“小程序不能从本地或远程文件系统读取或写入”，这个说法不确切。实际情况是几乎所有支持Java的浏览器都有个安全管理器，它不让小程序进行本地或远程访问。这个限制是浏览器厂家作出的业务决定，而不是小程序内在的限制。但对于允许和不允许哪种小程序的访问则有一个普遍认可的看法。大多数现有浏览器（包括所有Netscape Navigator产品）都实现了安全管理器。这些安全管理器不让小程序本地或远程读取。不让小程序作为TCP/IP服务器，只在服务器提供小程序本身时让小程序作为TCP/IP客户机。

除了字节码验证器和安全管理器外，上面介绍的类装入机制是另一道防线，它防止一种称为伪装（spoofing）的攻击。伪装就是建立与标准类同名的类，希望冒名顶替。最可能被伪装的类是控制敏感资源访问的类。在Java中，首当其冲的是SecurityManager类。可以看出，类装入算法能防止SecurityManager和其它标准JDK类的假冒。

敌人建立一个小程序作为诱饵，并把它放在服务器上的Web页面中。敌人还建立假冒的SecurityManager.class文件，提供对任何地方的读取和写入权限。小程序、假冒SecurityManager.class文件和Web页面都放在服务器上的同一目录中。敌人希望这个假SecurityManager被下载和用于验证。

但这个攻击不会成功，因为这个假冒类不会被装入。浏览器要装入SecurityManager类时，首先寻找自己仓库中的类。SecurityManager马上找到，所以不需要进一步搜索，假冒版本无人问津。

Java安全说来话长，附录C回答了许多常见问题。

1.4 文件系统

小程序不能访问本地文件系统，但应用程序则能够访问。各种不同路径名和权限规则产生了单平台程序中没有的问题。Unix用前斜杠作为路径分隔符，支持多权限；Windows 95用反斜杠，要求盘号和绝对路径名，支持较少权限。

java.io.File类包装了本地规则的访问，允许Java应用程序以独立于平台的方式使用文件系统。路径分隔符和权限问题隐藏在编程人员的视线后边。FileDialog类向用户提供了独立于平台的文件选择对话。第9章文件I/O和流部分详细介绍Java和文件系统。

1.5 属性

Java程序无法从本地机上读取环境变量。这是因为并不是所有平台都支持环境变量。Java提供了另一种指定环境变量的方法，称为属性。

Java属性类似于X资源。小程序从浏览器提供的表中读取属性和属性值，而应用程序从文件或命令行取得属性和属性值。由于安全原因，小程序对本地机属性的访问受到限制。第5章可移植性问题中将进一步介绍属性的使用和限制。

1.6 线程

由于Java环境是多线程的，编程人员可以用Java的线程建立多线程应用程序和小程序。

有些Java版本用基础机器的线程支持机制。而有些则从头开始建立自己的线程支持机制。不同平台有不同的线程支持，结果其线程表现也各不相同。主要的差别在于有的版本是先占性的，有的版本则不是。编写成功的可移植Java线程程序有赖于对不同模型的了解，并要能够建立在各种情况下均很健全的程序。第7章线程部分将涉及这个问题。

1.7 外观

Java运行在不断扩大的各种平台上，不同平台上的小程序和应用程序不一定有一致的外观和感觉。事实上，Java的意图恰恰相反：Motif平台上运行程序时要有Motif的外观和感觉，而Windows平台上运行程序时要有Windows的外观和感觉，等等。

java.awt软件包中的类包装了这种独立于平台的“变色龙”方法。为了编写成功的图形用户接口，通常要对整个机制的实现方法有一定了解。第3章“建立专用构件”，第4章“布置管理器”和第6章“图形”部分从开发健全程序码的角度介绍了Java平台独立性的各个方面。现在先介绍几个细节。

1.7.1 颜色

Java颜色模型支持24位颜色和8位alpha（不透明度）。这是一个模型，目前运行Java的机器很少有支持这么多颜色的硬件。运行系统要设法请求的颜色转变为可能的颜色。

有三种颜色映射的方法。第一种是采用高级计算机，实际支持所有颜色。第二种是将请求颜色映射到最接近的实际颜色。这种情况下，许多不同的红、绿、蓝颜色组合映射到基础平台中的同一种颜色。第三种方法是用配色图案蒙骗肉眼。

光盘上的小程序TrueColor显示了Java颜色在读者机器上的绘制方法。这个小程序显示3个256×256图表的矩形。矩形左边是不加蓝色时用红和绿的不同强度配成的各种颜色，红色从上到下由零到最大，绿色从左到右从零到最大；中间矩形是绿和蓝的混合，右边矩形是红和蓝的混合。这个小程序的源码也在光盘上，其中用到了第6章要介绍的类。读者不妨运行这个小程序，看看它在你的机器上如何绘制颜色。

1.7.2 字体

和颜色一样，字体也随平台的不同而不同。Java端口支持Helvetica、Times-Roman、Courier和Dialog字体，但可以将其随意映射为任何实际字体。

Java用java.awt.Font类包装字体行为和映射。Font的构造器建立对象的实例，并请求基础视窗系统建立字体本身。字体通常都缓存在视窗系统内，所以同一字体的多次构造也无妨大雅。例如，如果小程序多次调用下列构造器：

```
Fon bigfont = new Font (Helvetica", Font.ITALIC, 55);
```

首次调用时，建立Font的一个实例，这是在瞬间完成的。此外，请求视窗系统生成55点斜体Helvetice，这很费时，特别是在X平台上。下次调用该构造器时，建立新对象，对视窗系统的请求立即返回，因为55点斜体Helvetica已经现成。

Font类有取得实例的族、样式和字号的方法，但通常用处不大。返回的并不是视窗系统的实际族、样式和字号，而只是传递到Font构造器的请求值。

第5章“可移植性问题”部分将详细介绍字体问题。

1.7.3 构件布置

GUI构件在不同平台间更是大有不同。这就产生了构件布置问题。OK钮在一个平台上可能是50个图素宽，而在另一个平台上则为56个图素。程序要考虑每个可能的钮尺寸（以及滚动条大小、文本字段大小，等等）才能生成优美的布置。有时，构件出人意料地大，会把屏幕区搞得乱七八糟。

Java利用平台的外观感觉解决这个问题，而不建立自己的外观和感觉。Java将准确布置的工作包装在不同的布置管理器类中。编程人员不必指明构件的准确位置，只要确定布置的政策和实现这个政策的布置管理器结构组合。第4章布置管理器中详细介绍这些结构。

1.7.4 构件和同级件

Java构件（例如钮）只是个对象，是某个类的实例。java.awt软件包隐藏了一个复杂的机制，能启动构件在本地机的视窗系统上表示构件本身。java.awt.Toolkit类是Button之类的构件子类与基础视窗系统之间的中介。类是抽象的，将Java移植到新平台的部分工作就是生成目标机器上的Toolkit类的相应子类。

建立工具库时，Java检查属性awt.toolkit的值，这个值是本地机的Toolkit子类名。此后本地视窗系统上建立构件、图形、字体等的所有活动都是通过工具库调用进行的。

举钮为例。将钮加入容器时，工具库要构造实现ButtonPeer接口的某个类的实例。ButtonPeer就是一组本地调用，调用特定平台的awt库。awt库与本地视窗系统交互作用。每个平台有自己的ButtonPeer本地版本。在Motif系统中，Toolkit的Motif子类建立ButtonPeer的Motif实现器，它与Motif版本的awt库交互作用，而这个awt库又与本地机上的X服务器交互作用用。同样，在Windows平台上，Toolkit的Windows子类建立ButtonPeer的Windows实现器，它与Windows版本的awt库交互作用，而这个awt库又与本地机上的Windows 95交互作用。

编写好的Java程序码并不要求知道同级件机制的所有细节，但知道这些细节会有所帮助。

知道一点同级件机制有助于了解Java操作方式背后的原因。一定要知道，在同级体水平上的修改和生成子类是将Java移植到新平台的主要工作。小程序和应用程序编程人员在更高一层上用java.awt.Component的各种子类进行工作。即使在这个水平上，API文档和Java源码中也经常提到同级件。

第3章“建立定制构件”中介绍如何不经同级件水平的编码而建立类似标准构件的专用构件。

1.8 小结

现代计算机程序运行在丰富多彩的环境中，具有图形用户接口、杂合网络、全球资讯网、安全威胁和各种其它特性。这些环境特性是现代计算机行业的一大景观。Java正是在这种环境中出现的。成功的Java编程需要了解如何利用Java提供的工具与现代计算环境进行通信。

每个Java运行环境有自己的标准JDK类的备份。有些厂家将这些文件放在一目了然的地方，而有些厂家则放在比较隐蔽的地方。文件通常压合在一起，但不压缩。无论其如何千变万化，每个运行环境都知道如何寻找自己的系统类。

第2章　小程序和应用程序基础

- 小程序
- 应用程序
- repaint()、update()和paint()循环

本书不是个入门书，本书要求读者已经对小程序和应用程序略有所知。本章复习这两种Java程序的基础，并介绍一些高级信息，确立本书余下部分的基调。

2.1　小程序

每个小程序都是java.applet类的子类，所以每个小程序都从Applet及其类java.awt.Component、java.awt.Container和java.awt.Panel中继承了大量的功能。图2.1显示了小程序的继承结构。

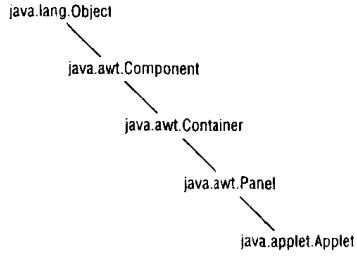


图2.1　Applet类结构

本节对小程序的介绍比下节对应用程序的介绍篇幅长很多，因为小程序继承内容很丰富，因此比应用程序有更多的内容值得一提。

2.1.1　小程序与HTML

小程序在浏览器中显示，这使它与应用程序有所不同。浏览器分析HTML文件，并在遇到<APPLET>标志时构造和管理小程序。

<APPLET>标志的格式如下：

```
<APPLETCODE=AppletClass.class WIDTH=width HEIGHT=height [optional tags] >
[ <PARAM NAME=ParamName VALUE=ParamValue> ]
[ alternate HTML ]
</APPLET>
```

方括号之内的内容是可选的，可以有多个<PARAM>标志。

可选标志列出如下，使用时必须放在第一组三角括号中。

ALIGN=*alignment* 指定小程序在HTML页面上对齐，可能的值为Left、right、top、texttop、middle、absmiddle、baseline、bottom和absbottom。这些值的作用和<IMG>标志中的一样。

Alt=*alternate Message* 指定HTML页面上显示的其它信息。这是用于能分析<APPLET>标志但不支持Java的浏览器，和用于放弃其它HTML。

ARCHIVE=*archive* 指定Java类的档案目录。

CODEBASE=*URL* 指定到URL指定的目录中寻找小程序字节码文件。这个目录不一定在HTML文件所在的机器上。如果没有这个标志，则编码基础目录为HTML文件所在的目录。

NAME=*appletName* 指定小程序名。同一页面的其它小程序可以用这个名称与这个小程序通信。这种方法会在下面介绍。

HSPACE=*spacing* 指定小程序左右边的空图素个数。

VSPACE=*spacing* 指定小程序上下边的空图素个数。

MAYSCRIPT=*boolean* 指定小程序是否与JavaScript码交互作用。这个标志只用于Netscape Navigator浏览器。boolean参数为true或false。

用<PARAM>标志可以将任意参数／数值对传递到小程序中。这个数据出现在自己的三角括号集合中，其格式如下：

```
<PARAM NAME=ParamName VALUE=ParamValue>
```

小程序可以调用下列方法读取任何参数值：

```
String getParameter (String ParamName) ;
```

如果ParamName在小程序页面的<PARAM>标志中有出现，则这个方法将参数返回为字串，如果paramName不在小程序页面中，则返回数值为Null。

说明： <Param>标志对定期改变的Web页面特别有用。例如，假设用小程序画一个地区天气分布图，并按最近24小时的平均气温进行颜色编码，则小程序通过<param>标志接收原始气温数据。一个批命令每天读取文件中的原始数据，并发表当前日期的最新HTML页面。批命令写入HTML页面的<Param>标志部分时，插入原始温度数据，结果是稳定的小程序产生动态的Web页面。

2.1.2 小程序生命周期

小程序生命周期从浏览器访问小程序的Web页面时开始。这个过程的第一部分在上一章“类的装入”部分介绍。这一节从小程序装入类之后讲起。

装入类之后，浏览器建立这个小程序类的实例，这是通过调用小程序构造器完成的。从这以后，这个小程序实例是被动的，不会主动地干任何事，只是响应浏览器的方法调用。

构造小程序之后，浏览器在屏幕视区中分配小程序的空间。这时发生一件重要事实：小程序得到一个同级件（peer）。同级件在第6章“图形”部分介绍。简单地说，同级件就是构件向本地的特定平台基础视窗系统的连接。Java构件的功能大都是利用本地视窗系统达到的。

在生命周期的下一步中，浏览器向小程序发出几种启动方法调用。每种方法都是从超

类java.applet.Applet继承的。继承的版本只是个空壳，什么也不干。编程人员可以改写这些方法，使小程序具有所要的特性。一定要说明的是，调用这些方法时，小程序已经完全构件并得到了同级件。

一、init( )

浏览器对小程序的第一个调用是init()。这是唯一只调用一次的起动方法，小程序应当在这里完成一次性初始化工作。init()中完成的通常功能包括：

- 建立小程序GUI
- 读取<param>标志的数值
- 装入外部文件中的屏外图形
- 装入外部文件中的声频片断

实例变量可以在init( )中初始化，也可以在声明行中初始化，使一切都在一个地方初始化。

说明： 编译器保证将数字实例变量初始化为零（zero）、字符为“”、逻辑变量为false，对象和数组引用变为null。这样，符合这些初始值的实例变量即不需再作显式初始化。这种初始化不发生在自动（方法内）变量中。

二、start( )和stop( )

浏览器调用的下一种方法是start()。这个名称有点含糊，因为没有指明要启动什么。Start()方法和Start()方法相伴使用。调用第一个Start()方法之后，用户浏览小程序以外时，浏览器调用Stop()，而用户浏览小程序时又再次调用Start()。这个概念有点含糊，不同浏览器在不同情况下进行调用。一般来说，Stop()在浏览器显示新Web页面或图标化时调用，Start()在浏览器返回小程序页面或展开图标时调用。

也许有些读者不知道，页面退到后面时小程序并不去分配。目的是以后可以重访页面而不必再次装入小程序，且小程序退出时的相同状态而不进行初始化。为此，浏览器缓存小程序。

一般来说，小程序在Stop()方法中暂停和Start()方法中恢复的活动有三项：

动画　将CPU时钟浪费在看不到的动画上是毫无益处的。

声音　播放与所看页面无关的声音会把用户搞糊涂。任何图标化的程序都不应播放声音。

一般后台线程处理　线程进行与当前页面不同的页面处理，从而会占用当前页面的可用资源。

三、Paint(Graphics g)

这个方法和start()一样，在启动时调用并在此后根据浏览器的需要调用。Paint()方法接受Graphics类的一个实例。Graphics类的每个实例针对屏幕构件区的一个图形或屏外的图形。传入Paint()的实例按小程序的图素数绘图。

初始调用之后，浏览器要小程序绘制图形时调用Paint()，产生这种调用的条件随浏览器的不同而不同，最常见的原因是损坏后的修复。浏览器的小程序部分被另一窗口覆盖之后再次显示时，有些地方的有些图素要重新绘制，才能给人三维桌面的感觉。