HZ BOOKS

Sun 公司核心技术丛书

Effective Java
Programming Language Guide

Effective Java
中文版

2002年度
Jolt大奖
BOOKS
software development 2002 product excellence award

(美) Joshua Bloch 著

潘爱民 译

Sun microsystems

机械工业出版社
China Machine Press

Sun公司核心技术丛书

Effective Java中文版

Effective Java Programming Language Guide

（美） Joshua Bloch 著

潘爱民 译

机械工业出版社
China Machine Press

本书介绍了在Java编程中57条极具实用价值的经验规则，这些经验规则涵盖了大多数开发人员每天所面临的问题的解决方案。通过对Java平台设计专家所使用的技术的全面描述，揭示了应该做什么、不应该做什么才能产生清晰、健壮和高效的代码。

本书中的每条规则都以简短、独立的小文章形式出现，这些小文章包含了详细而精确的建议，以及对语言中许多细微之处的深入分析，并通过例子代码加以进一步说明。贯穿全书的是通用的语言用法和设计模式，以及一些具有启发意义的技巧和技术。

本书版权登记号：图字：01-2001-4782

图书在版编目（CIP）数据

Effective Java中文版 /（美）布洛克（Bloch, J.）著；潘爱民译. -北京：机械工业出版社，2003.1

（Sun 公司核心技术丛书）

书名原文：Effective Java Programming Language Guide

ISBN 7-111-11385-3

Ⅰ. E… Ⅱ. ①布… ②潘… Ⅲ. JAVA语言－程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2002）第102866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：贾 梅

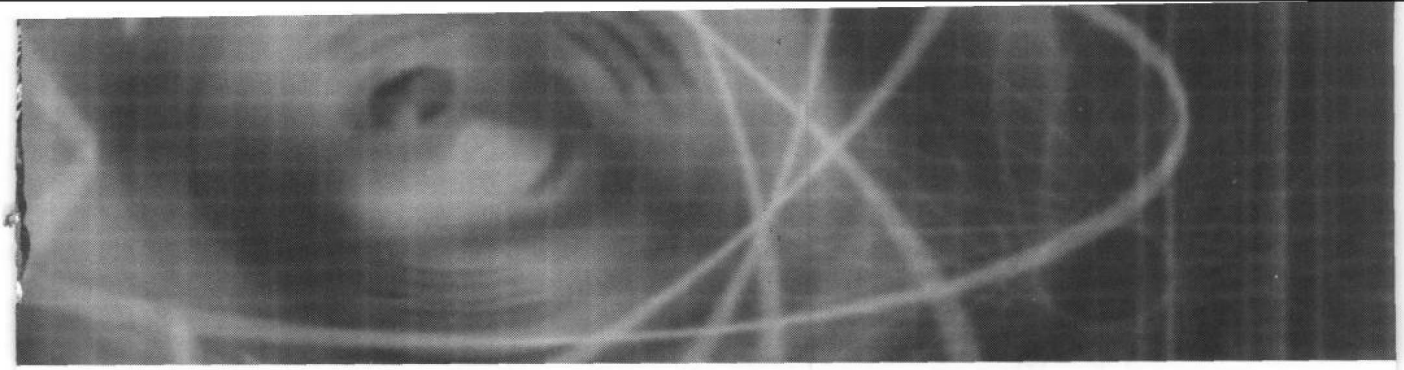
北京瑞德印刷有限公司印刷·新华书店北京发行所发行

2003 年 1 月第 1 版·2003 年 3 月第 2 次印刷

787mm×1092mm 1/16·14.75 印张

印数：5 001-8 000 册

定价：39.00 元



译者序

这是一本不起眼的小书，但是它里边介绍了不平凡的内容。

世界上每一件事情之所以会发生都有一定的原因，这本薄薄的小书能够获得第12届软件开发Jolt图书大奖（http://www.sdmagazine.com/），起初让我非常惊讶，但是当我接手重新翻译这本书的任务，仔细阅读之后，我知道它获得这个奖项是当之无愧的。

我曾经译写过多部软件开发类的图书，但是从来没有进入到Java世界中来，也没有做过Java平台上的开发工作，所以，翻译这本书主要靠在其他领域的经验和感觉，也算是一种胆大的尝试。尤其让我倍感压力的是，这本书已经有了第一次翻译，我能翻译得更好吗？不管怎么样，我尽力去做就是了。

这本书之于Java程序设计语言的意义，决不亚于Scott Meyers的《Effective C++》之于C++程序设计语言的意义。我之所以这么说，不仅仅是因为这本书借用了Scott Meyers的著书风格，更重要的是，它所介绍的内容都来自于第一线实践经验，书中的每一条规则都抓住了Java程序设计语言和平台库的关键之处。这些经验对于普通程序员而言，可以让他们迅速回归到Java语言所规定的正确道路上来；对于高级程序员而言，可以让他们把每天的使用经验上升到理论和规范的高度，使之适用于更广泛的场合。本书作者Joshua Bloch是Java Collections Framework的设计师，他还设计、指导和实现了其他一些可重用的类库。本书融入了作者多年的可重用组件设计经验，所以，您在阅读本书的时候，处处都能感受到作者的设计经历，包括经验和教训，也能感受到Java平台如何克服其原始的设计缺点，逐步走向完美和成熟。

毫无疑问，这本书不适合Java语言的初学者。它并没有指导您如何编写实用的Java程序，而是指导您在Java程序设计中需要遵循什么样的规则才能编写出高效、灵活、健壮、可重用的程序。作者把焦点集中在API的设计上，所以，如果您正在设计一个可重用的系统或者子系统，那么本书中的内容对您再合适不过了。

本书共包括57条程序设计建议，有的建议是几乎每一个熟练的Java程序设计人员都亲身经历过的，比如第2章中的一些通用方法，以及第3章中有关类和接口的设计规则；有的建议涉及到Java程序设计语言的一些高级用法，比如第5章介绍了如何代替C语言中的一些类型设施，第8章讨论了异常的一些用法；有的建议涉及到Java平台的底层系统，比如第9章讨论了Java平

台上设计多线程程序应该注意的事项。这些建议涵盖了Java程序设计的方方面面，可以这样说，本书是Java程序设计浓缩的精华，每一条都值得您细细阅读和品味。

在内容叙述上，本书也颇具特色：

- 在讲述每条规则的时候，使用了一些很有说服力的短小例子，这些例子大多来自于作者的亲身经历。
- 客观地指出了Java语言和平台库中的一些设计缺陷，以及容易被忽略的实现细节。从这个意义上讲，本书称得上是一本“Java语言和平台库的技术内幕”。
- 大量运用了模式和反模式。设计模式是人们对于普遍适用的设计方案的经验总结和提炼，反模式则是应该避免的设计方案。本书把Java语言和大量现有的设计模式有机地结合起来，同时也展现了一些在Java平台库成长过程中提炼出来的新模式。
- 严谨的参考引用信息。作者不仅在叙述每个条目的时候，提供了前后相关的参考信息，同时也引用到其他一些经典资料。作者在许多细节上着墨并不多，但是他都提供了有关的参考资料，从而既保证了叙述的完整性，也保持了本书“精、巧”的特色。

本书包括这么多高质量的程序设计建议，我相信每一个有经验的Java程序设计人员都会喜欢和赞同这些建议。但是，如果把这些设计建议应用到日常的Java程序设计中，一定会编写出高质量的代码来吗？应该会的，但是这将使Java程序设计工作非常复杂，并且编写出来的代码不简洁，也不直观，这当然不符合Java语言的设计思想。所以，您需要用正确的态度来学习和使用这些设计规则。如果您正在设计可重用组件库，那么几乎每一个条目都有助于您设计出更合理的API来，您无论花多少时间来研究这些设计规则都是值得的。如果您正在设计普通的Java应用，那么，这些设计规则将有助于您更好地利用Java平台库来完成开发任务，而且一旦应用程序出现问题，您可以快速地诊断出问题所在，并找到合理的解决方案。

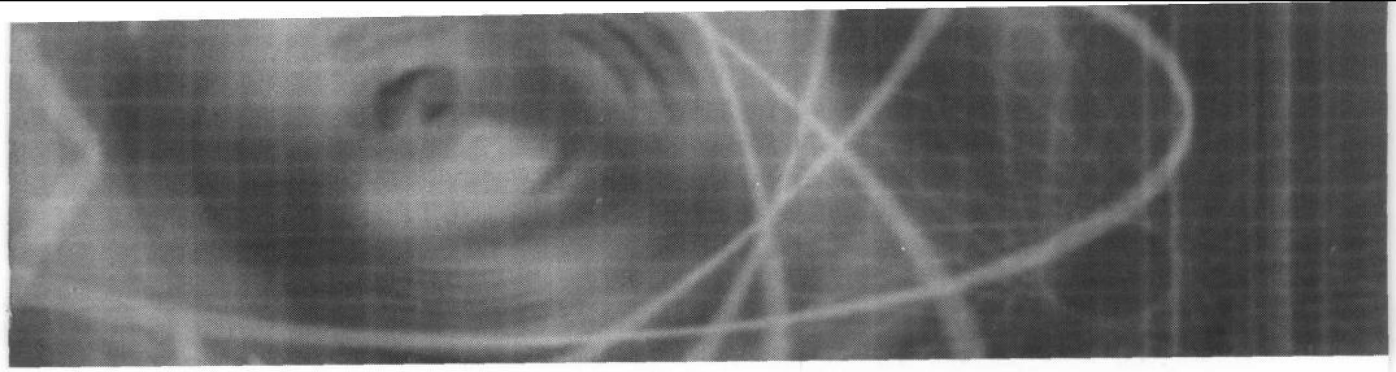
虽然这本书不适合Java语言的初学者，但是，如果您具有其他语言的程序设计经验，特别是C/C++语言的程序设计经验，那么本书对于您了解Java语言和平台库非常有帮助。而且，本书中的许多内容具有普遍适用性，并不局限于Java语言和平台，所以您一样可以从中学到有用的知识，甚至全面提升自己的程序设计能力。这是我翻译这本书过程中的切身体会。

由于本书内容深入，涉及面又比较广，加之我对于Java缺乏足够的实践经验，所以，翻译过程并不轻松，尽管对于一些疑难之处我查阅了有关的文档，特别是Java 2 平台的在线文档，但是，译文中错误在所难免，敬请读者谅解。为阅读方便，特在书后附上中英文术语对照。

这是一本好书，希望它不会辜负您的期待。

潘爱民

2002年10月于北京大学燕北园



序

如果有一个同事这样对你说，“我的配偶今天这个晚上在家里制造了一场不同寻常的晚餐，你愿意参加我们吗？”（Spouse of me this night today manufactures the unusual meal in a home. You will join?）这时候你脑子里会想到三件事情：第一，同事是在邀请你参加家庭晚宴；第二，英语不是这位同事的母语；第三，有一顿可口的晚餐在等着你。

如果曾经学习过第二种语言，并尝试在课堂之外使用这种语言，那么你该知道有三件事情是必须要掌握的：这门语言的结构如何（语法）、如何命名你想谈论的事物（词汇），以及如何用习惯和高效的方式来表达事情（用法）。在课堂上通常只是涉及到前面两点，而当你努力使对方明白你的意思的时候，你常常会发现当地人对你的表述忍俊不禁。

对于程序设计语言，也是如此。你需要理解语言的核心：它是面向算法的，还是面向函数的，或者是面向对象的？你需要知道词汇表：标准库提供了哪些数据结构、操作和功能设施？你还需要熟悉如何用习惯和高效的方式来构建代码。关于程序设计语言的书籍通常只是涉及到前面两点，或者只是蜻蜓点水般地介绍一下用法。也许原因在于，前面两点更加容易编写。语法和词汇是语言本身固有的特性，但是用法则反映了使用这门语言的群体的特征。

例如，Java程序设计语言是一门只支持单继承的面向对象程序设计语言，在每一个方法内部，它也支持命令方式的（面向语句的，statement-oriented）编码风格。Java库包括对图形显示、网络、分布式计算和安全性的支持。但是，如何把这门语言以最佳的方式用到实践中呢？

还有一点，程序与口头的句子以及大多数书籍和杂志不同，它是会随着时间而变化的。仅仅编写出能够有效地工作并且能够被别人理解的代码往往是不够的，我们还必须要把代码组织成易于修改的形式。针对一个任务T可能会有10种不同的编码方法，而在这10种方法中，有7种方法是笨拙的、低效的或者是难以理解的。而在剩下的3种编码方法中，哪一种会最接近该任务T的下一年度版本的代码呢？

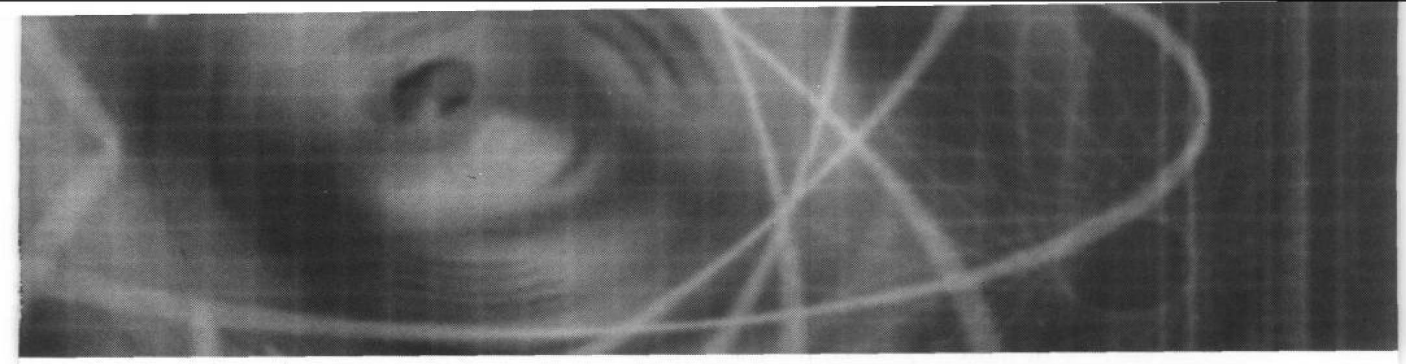
目前有大量的书籍可以供你学习Java程序设计语言的语法，包括《The Java Programming Language》[Arnold00]（作者Arnold、Gosling和Holmes，2000），以及《The Java Language

Specification》[JLS]（作者Gosling、Joy和Bracha）。同样地，关于Java库和API的书籍也有很多。

本书定位在你的第三个需要上：习惯和高效的用法。作者Joshua Bloch在Sun Microsystems公司多年来一直从事Java语言的扩展、实现和使用的工作；他还阅读了其他人的大量代码，包括我的代码。他在本书中提出了许多好的建议，他按照系统化的方式把这些建议组织起来，这些建议的宗旨在于如何更好地构造你的代码以便它们工作得更好，以便其他人也能够理解这些代码，以便将来对代码做修改和增强的时候不会头痛，甚至，你的程序因此而变得更加令人舒适、更加优美和雅致。

Guy L. Steele Jr.[⊖]
Burlington, Massachusetts
2001年4月

⊖ Guy Steele是Sun研究院的杰出工程师（Distinguished Engineer）。他是程序设计语言领域的世界级专家，Scheme语言的设计者之一，1988年ACM Grace Murray Hopper奖得主。其著名著作《C: A Reference Manual》即将由机械工业出版社出版。——译注



前　言

1996年，我打点行囊，西行来到了当时的JavaSoft，因为我很清楚那里将会出现奇迹。在这5年间，我担任Java平台库的设计师。我曾经设计、实现和维护过许多库，同时也担任其他一些库的技术顾问。伴随着Java平台的成熟和壮大，主持这些库的设计工作是一个人一生中难得的机会。毫不夸张地说，我有幸与一些当代最杰出的软件工程师一起工作。在这个过程中，我学到了许多关于Java程序设计语言的知识——它能够做什么，不能够做什么，如何最有效地使用这门语言和它的库。

本书是我的一次尝试，我希望与你分享我的经验，你可以因此而吸取我的经验，避免重复我的失败。本书中我借用了Scott Meyers的《Effective C++》[Meyers98]一书的格式，该书中包含有50个条目，每个条目给出了一条用于改进程序性能和设计方案的规则。我觉得这种格式非常有效，希望你也有这样的感觉。

在许多例子中，我冒昧地使用了Java平台库中的真实例子来说明相应的条目。在介绍那些做得不是很完美的工作时，我尽量使用我自己编写的代码，但是偶尔我也会使用其他同事的代码。尽管我尽力做得更好一点，但是如果我真的冒犯了他人，我在这里致以最诚挚的歉意。引用反面例子是出于协作的考虑，而不是要羞辱例子中的做法，我希望大家都能够从我们过去的错误经历中得到启发。

尽管本书并不只是针对可重用组件开发人员的，但是过去20多年来我编写此类组件的经历一定会影响到这本书。我很自然地会按照可导出API的方式来思考问题，而且我鼓励你也这样做。即使你并没有开发可重用的组件，但是这样的思考方法有助于你提高软件的质量。进一步来说，毫无意识地编写可重用组件的情形并不少见：你编写了一些很有用的代码，然后在同伴之间共享，不久之后你就有了很多用户。这时候，你就不能随心所欲地改变API了，并且如果你刚开始编写软件的时候在设计API上付出了较多的努力，那么这时你就会非常庆幸了。

我把焦点放在API的设计上，这对于那些热衷于新兴的轻量级软件开发方法学（比如Extreme Programming[Beck99]，中文译为“极限编程”，简称XP）的读者来说，也许会显得有点不太自然。这些方法学强调编写最简单的、能够工作的程序。如果你正在使用某种此类

的程序设计方法，那么你会发现，把焦点放在API设计上对于“*重构（refactoring）*”过程是多么有益。重构（refactoring）的基本目标是改进系统结构，以及避免代码重复。如果系统的组件没有设计良好的API，则要达到这样的目标是不可能的。

没有一门语言是完美的，但是有些语言非常优秀。我认为Java程序设计语言以及它的库非常有益于代码质量和效率的提高，并且使得编码工作成为一种乐趣。我希望本书能够抓住我的热情并传递给你，帮助你更有效地使用Java语言，工作更为愉快。

Joshua Bloch
Cupertino, California
2001年4月

请注意

本书索引所列页码，皆为英文原书页码。

“真希望10年前我就能拥有这本书。可能有人会认为我不需要任何关于Java的书籍，但是我确实需要这本书。”

——James Gosling，Java之父，
Sun 公司副总裁

“一本非常优秀的书，充满了各种关于使用Java程序设计语言和面向对象程序设计的好的建议。”

——Gilad Bracha，Sun公司计算机科学家，
《The Java™ Language Specification》
(Second Edition) 的作者之一

你正在寻找一本简明扼要地阐述Java精髓的书吗？你希望深入地理解Java程序设计语言吗？你希望编写出清晰、正确、健壮和可重用的代码吗？不用再找了，你手上这本书将会使你实现这些愿望，而且还能提供其他许多你意想不到的好处。

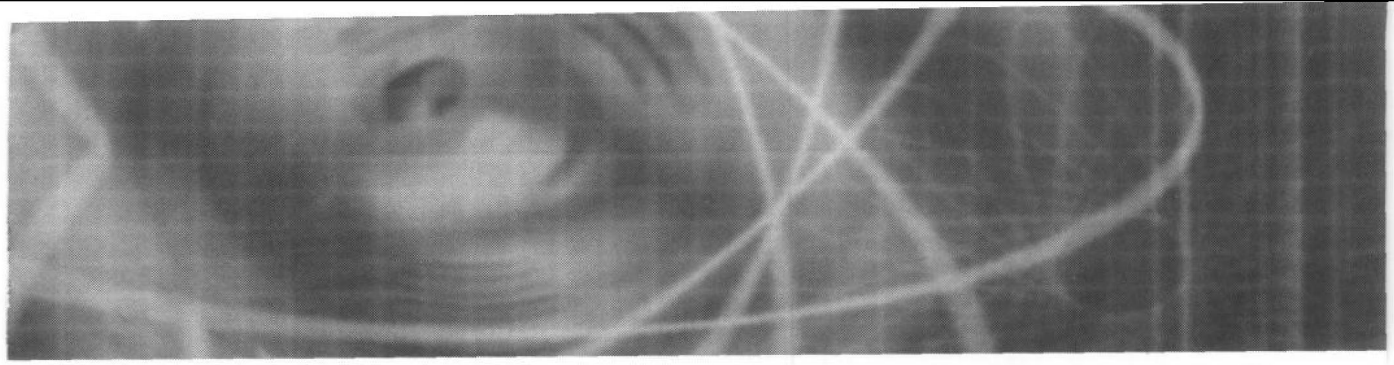
作者简介

Joshua Bloch 是Sun公司的高级工程师，也是“Java平台核心组”的设计师。他设计并实现了获奖的Java Collections Framework和java.math软件包，并且对Java平台的其他部分也做出了贡献。Joshua是许多技术文章和论文的作者，他的关于抽象数据对象复制的博士论文获得过“ACM杰出博士论文奖”提名。他拥有哥伦比亚大学的学士学位和卡耐基-梅隆大学的博士学位。

译者简介

潘爱民 浙江海宁人，现任职于北京大学计算机科学技术研究所，副研究员；研究方向为信息安全（包括网络安全和公钥技术）和软件开发（包括组件技术和模式）；主要著作有《COM原理与应用》等，译著有《Visual C++技术内幕》（第4版）、《COM本质论》和《C++ Primer中文版》等。

目 录



第1章

引　言

本书的目标是帮助你最有效地使用Java™程序设计语言，以及它的基本库：java.lang、java.util，某种程度上还包括java.io。本书也会不时地讨论到其他的库，但是没有涉及图形用户界面编程以及企业级API。

本书共包含57个条目，每一条都涵盖一个规则。这些规则反映了最有经验的程序员在实践中的一些有益的做法。这些条目以一种比较松散的方式组织成9章，每一章涉及软件设计的一个主要方面。对本书并不一定要从前到后逐页阅读，每一个条目都有一定程度的独立性。这些条目相互之间有交叉索引，你可以按照自己的思路来通读全书。

大多数条目都通过程序例子来说明每一条的内容。本书的一个突出的特点是，它包含了许多代码例子，这些例子说明了许多设计模式（*design pattern*）和习惯用法（idiom）。其中一些模式是老的，例如Singleton（见第2条）；其他一些模式则是新的，例如Finalizer Guardian（终结函数守卫者）（见第6条）和Defensive readResolve（保护性的readResolve方法）（见第57条）。书末有一个单独的索引表可用来快速地查找到这些设计模式和习惯用法。在需要参考设计模式领域标准参考书[Gamma 95]的地方，还为这些模式和习惯用法提供了交叉索引。

许多条目包含有一个或多个在实践中应该避免的程序例子。像这样的例子，有时候也被称为“反模式（*antipattern*）”，在注释中被清楚地标注为“//Never do this!”。对于每一种情况，该条目都解释了为什么此例不好，并提出了可选的解决方法。

本书并不是针对初学者的：本书假设读者已经熟悉Java程序设计语言。如果还没有，请考虑先参考一本好的Java入门书籍[Arnold00, Campione00]。本书的目标是适用于任何一个具有实际Java工作经验的程序员，对于高级程序员，它也应该能够提供很有价值的参考材料。

本书中大多数规则源于少数几条基本的原则。清晰性和简洁性是最为重要的：一个模块的用户永远也不应该被模块的行为所迷惑（那样就不清晰了）；模块要尽可能的小，但又不能太小［术语模块（*module*）在本书中的用法，是指任何可重用的软件组件，从单个方法，到

包含多个包的复杂系统都可以是一个模块]。代码应该被重用，而不是被拷贝。模块之间的相依性应该尽可能地降低到最小。错误应该尽早被检测出来，理想情况下是在编译时刻。

虽然本书中的规则不会百分之百地适用于任何时刻和任何场合，但是，它们确实刻画了绝大多数情况下最佳的程序设计行为。你不应该盲目地遵从这些规则，但是，你只应该在偶尔的情况下，有了充分的理由之后才打破这些规则。学习程序设计的艺术，如同大多数其他的学科一样，首先要学会基本的规则，然后才能知道什么时候可以打破这些规则。

本书中大部分内容都不是讨论性能的，而是关心如何编写出清晰、正确、可用、健壮、灵活和可维护的程序来。如果你能够做到这一点的话，那么要想获得你所需要的性能往往是相对比较简单的（见第37条）。有些条目确实讨论了性能问题，甚至有的还提供了性能指标。但是，在提及这些指标的时候，也会出现“在我的机器上”这样的话，所以，你最好把这些指标看做近似值。

有必要提及的是，我的机器是一台过时的家用电脑，主机为400MHz Pentium II，128M内存，在Microsoft Windows NT 4.0操作系统平台上运行Sun 1.3版本的Java 2 Standard Edition Software Development Kit（SDK）。该SDK包括Sun的Java HotSpot™ Client VM——这是一个专门供客户端使用的新一代Java虚拟机（JVM）。

在讨论到Java程序设计语言及其库的特性的时候，有时候必须要指明具体的发行版本。为简单起见，本书使用了工程版本号（engineering version number），而不是正式的发行号。表1-1列出了发行号与工程版本号之间的对应关系。

表1-1 Java平台的版本

正式发行号	工程版本号
JDK 1.1.x/JRE 1.1.x	1.1
Java 2 Platform, Standard Edition, v 1.2	1.2
Java 2 Platform, Standard Edition, v 1.3	1.3
Java 2 Platform, Standard Edition, v 1.4	1.4

虽然有些条目讨论到了1.4发行版本中引入的特性，但是程序例子中除了个别情况外，尽量避免使用这些特性。这些例子在发行版本1.3上已通过测试，大多数例子（不是所有的例子）无需修改就可以在发行版本1.2上运行。

尽管这些例子都很完整，但是它们更注重于可读性。它们直接使用了`java.util`和`java.io`包中的类。为了编译这些例子程序，你可能需要在程序中加上一行或者两行*import*语句：

```
import java.util.*;
import java.io.*;
```

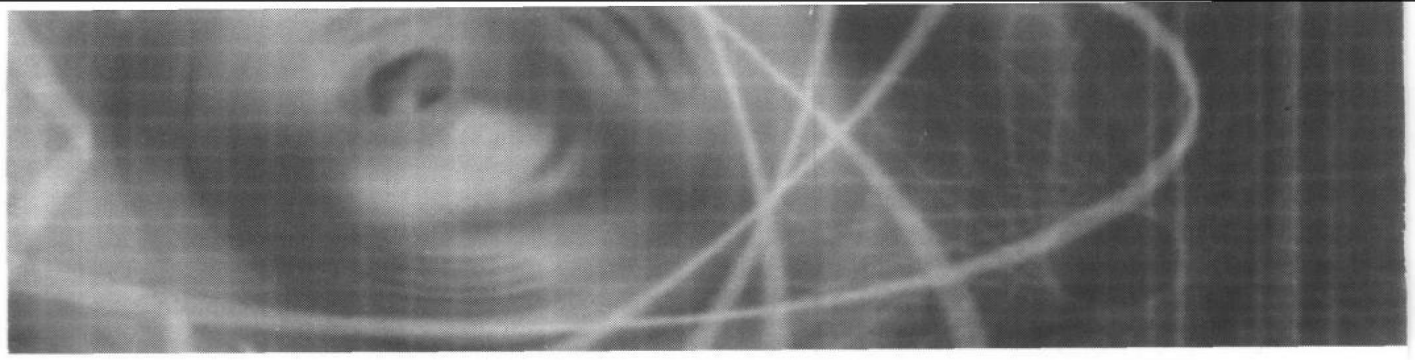
其他的代码示例中也有类似被省略的情况。本书的Web站点，http://java.sun.com/docs/books/effective，包含了每个例子的完整版本，你可以直接编译和运行这些例子。

在很大程度上，本书使用的技术术语与《The Java Language Specification, Second Edition》[JLS]相同。而有一些术语值得特别提出来。Java语言支持四种类型：接口（*interface*）、类（*class*）、数组（*array*）和原语类型（*primitive*）。前三种类型通常被称为引用类型（*reference type*），类的实例和数组是对象（*object*），而原语类型的值不是对象。一个类的成员（*member*）包括它的域（*field*）、方法（*method*）、成员类（*member class*）和成员接口（*member interface*）。一个方法的原型（*signature*）包括它的名字和所有形参的类型，方法原型不包括它的返回类型。

本书也使用了一些与《The Java Language Specification》不同的术语。与《The Java Language Specification》不同的是，本书把术语“继承（*inheritance*）”用做“子类化（*subclassing*）”的同义词。本书不再使用“接口继承”这种说法，而是简单地说，一个类实现（*implement*）了一个接口，或者一个接口扩展（*extend*）了另一个接口。为了描述“在没有指定访问级别的情况下所使用的访问级别”，本书使用了描述性的术语“包级私有（*package-private*）”，而不是如[JLS, 6.6.1]中所使用的技术性术语“默认访问（*default access*）级别”。

本书也使用了一些在《The Java Language Specification》中没有定义的技术术语。术语“导出的*API*（*exported API*）”，或者简单地说*API*，是指类、接口、构造函数（constructors）、成员和序列化形式（serialized form），程序员通过它们可以访问一个类、接口或包（术语*API*是*Application Programming Interface*的简写，这里之所以使用API而不用接口（*interface*），是为了不与Java语言中的interface类型相混淆）。使用API编写程序的程序员被称为该API的用户（*user*），在类的实现中使用了API的类被称为该API的客户（*client*）。 3

类、接口、构造函数、成员以及序列化形式被统称为*API元素*（*API element*）。导出的API包括所有可在定义该API的包之外访问的API元素。任何客户都可以使用这些API元素，而API的创建者负责支持这些API元素。无独有偶，Javadoc实用工具在它的默认操作模式下生成的文档中也正是这些元素。可以不严格地讲，一个包的导出API包括该包中每一个公有（public）类或者接口中所有公有的或者受保护的（protected）成员和构造函数。 4



第2章

创建和销毁对象

本章的主题是创建和销毁对象：什么时候、如何创建对象；什么时候、如何避免创建对象；如何保证对象能够适时地销毁；对象被销毁之前如何管理各种清理工作。

第1条：考虑用静态工厂方法代替构造函数

对于一个类，为了让客户获得它的一个实例，最通常的方法是提供一个公有的构造函数。实际上还有另外一种技术，尽管较少为人所知，但也应该成为每个程序员的工具箱中的一部分。类可以提供一个公有的静态工厂方法（*static factory method*），所谓静态工厂方法，实际上只是一个简单的静态方法，它返回的是类的一个实例。下面是一个来自Boolean类（原语类型boolean的包装类）的简单例子。其中静态工厂方法是1.4版新增的，它把一个boolean原语值转换为一个Boolean对象引用：

```
public static Boolean valueOf(boolean b) {
    return (b ? Boolean.TRUE : Boolean.FALSE);
}
```

类可以为它的客户提供一些静态工厂方法，来替代构造函数，或者同时也提供一些构造函数。用静态工厂方法来代替公有的构造函数，既有好处，也有不足之处。

静态工厂方法的一个好处是，与构造函数不同，静态工厂方法具有名字。如果一个构造函数的参数并没有确切地描述被返回的对象，那么选用适当名字的静态工厂可以使一个类更易于使用，并且相应的客户代码更易于阅读。例如，构造函数BigInteger（int，int，Random）返回的BigInteger可能是素数，然而，如果使用一个名为BigInteger.probablePrime的静态工厂方法，表达显然更为清楚。（静态工厂方法BigInteger.probablePrime最终已加入1.4版了。）

一个类只能有一个原型相同的构造函数。程序员通常知道该如何绕开这种限制，他可以提