Java 8 编程入门官方教程

（第6版）

[美] Herbert Schildt 著

王楚燕 鱼 静 译

清华大学出版社

北 京

Herbert Schildt
Java : A Beginner's Guide，Sixth Edition
EISBN：978-0-07-180925-2

北京市版权局著作权合同登记号　图字：01-2014-7593

图书在版编目(CIP)数据

Java 8 编程入门官方教程：第 6 版/(美) 施密特(Schildt, H.) 著；王楚燕，鱼静 译. —北京：清华大学出版社，2015
书名原文：Java : A Beginner's Guide，Sixth Edition
ISBN 978-7-302-38738-1

Ⅰ. ①J… Ⅱ. ①施… ②王… ③鱼… Ⅲ. ①JAVA 语言－程序设计－教材 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2014)第 284272 号

责任编辑：王　军　李维杰
封面设计：牛艳敏
责任校对：成凤进
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**39　**字　　数：**1023 千字
版　　次：2015 年 1 月第 1 版　　**印　　次：**2015 年 1 月第 1 次印刷
印　　数：1～3000
定　　价：69.80 元

产品编号：

译 者 序

Java 从最初发布到现在，已经快 20 年了。如今，Java 依然是众多开发人员的首选，这自然与其不断更新、紧跟时代有关。这是一门语言生命力旺盛的体现。

Java 发展到今天，最新的版本就是本书要介绍的 Java SE 8。历代版本中，有的对上一版做了重大更新，有的只是修修补补。Java 8 属于前一类，其最突出的地方就是引入了 lambda 表达式。lambda 表达式产生的影响深远：不但改变了概念化的编程方式，而且改变了 Java 代码的编写方式。可以预见，Java 开发人员思考问题的现有方式和编码方法必然会受到冲击。但冲击的后果，就是当开发人员选择在合适的地方使用 lambda 表达式后，工作效率会有明显提升。其他一些新特性还包括接口方法的默认实现，以及终将取代 Swing 的 JavaFX 框架，这些全都足以使开发人员欢欣鼓舞。

可见，Java 并没有沉溺于过去的荣耀、固步自封，而是竭力为开发人员考虑，通过版本的更新，使开发人员有机会利用最新、最有价值的思想和理念提高开发效率。所以选择学习 Java 无疑是正确的决定。那么，要如何学习 Java 呢？

对于入门来说，选择一本教程是最方便、最实际的方法。但是，Java 是一门极受欢迎的语言，市面上介绍 Java 的图书让人眼花缭乱，难道让新手每本都看？事实上，读者既没有必要每本都看，也不会有那样的时间每本都看。那么，选择什么样的图书呢？答案很简单：选择一本优秀的教程。优秀的教程往往由浅入深进行介绍，并且重点突出，内容全面，示例代码丰富且实用性强。作者穿插在特性介绍中的语言，也常常反映出作者丰富的经验和深刻的见解，对于初学者形成正确的认识和良好的编程习惯极有帮助。

本书就是这样一本优秀的教程，用凝练的语言传达最有价值的信息，在 Amazon 网站上

获得了非常好的评价，销量也非常不错。作者 Herbert Schildt 可谓大名鼎鼎，他有几十年的编程经验，也是 Java 语言方面的权威，曾撰写过众多关于 Java 的经典图书。所以本书的质量应该不言而喻。当然，有的作者功力深厚，在写书时不由讲得很深奥，所以作者知名不代表书就一定适合初学者。但是对于本书，作者提到了前提就是读者根本没有编程背景，所以完全从良师的角度，循循善诱，帮助读者从最根本的概念开始，一步一步地学习庞大的 Java 平台。书中除了介绍入门技巧之外，还以“自测题”、“编程练习”和“专家问答”的形式给出了很多实用性的习题和问题，这些都是作者智慧和经验的结晶，仔细研究这些习题和问题会让读者受益匪浅。

这次翻译技术类书籍，最大的体会就是，翻译和自己阅读外文技术书籍差别非常大。虽然阅读外文技术类书籍没有什么障碍，但开始翻译工作之后才真切体会到，想要准确精炼地表达原作者的意思，绝不是自己读懂那么简单。况且本书凝聚了 Java 大师的独到见解，即便专门从事 Java 开发，也没有完全准确地理解并表达所有内容的自信。为了尽可能保证翻译内容的准确完整性，我在翻译过程中，曾多次就不确定的部分征求产品开发团队同事的意见。在此我想对团队的全体同事表达我最真挚的谢意。

在这里要感谢清华大学出版社的编辑，她们为本书的翻译投入了巨大的热情并付出了很多心血。没有你们的帮助和鼓励，本书不可能顺利付梓。本书全部章节由王楚燕 、鱼静翻译，参与翻译活动的还有孔祥亮、陈跃华、杜思明、熊晓磊、曹汉鸣、陶晓云、王通、方峻、李小凤、曹晓松、蒋晓冬、邱培强、洪妍、李亮辉、高娟妮、曹小震、陈笑。在此一并感谢！对于这本经典之作，译者本着“诚惶诚恐”的态度，在翻译过程中力求“信、达、雅”，但由于译者水平有限，翻译工作中可能会有不准确的内容，如果读者在阅读过程中所发现的失误和遗漏之处，希望能够多多包涵，并欢迎批评指正。

作 者 简 介

畅销书作家 Herbert Schildt 是 Java 语言的权威，他撰写的程序设计图书大约有三十多本，在全世界销售了数百万册，并被翻译成了多种语言。他撰写了众多关于 Java 的图书，包括 *Java: The Complete Reference*、*Herb Schildt's Java Programming Cookbook* 和 *Swing: A Beginner's Guide*，还撰写了一些有关 C、C++和C#方面的书籍。虽然他对计算机的方方面面都很感兴趣，但主要关注点是计算机语言，包括编译器、解释器和机器人控制语言。他对语言的标准化也有浓厚的兴趣。Schildt 获得了伊利诺伊大学的学士学位。他的咨询中心的电话号码为(217)586-4683。他的网站为 www.HerbSchildt.com。

技术编辑简介

Danny Coward 博士在 Java 平台的各个版本上都工作过。他将 Java servlet 的定义引入到 Java EE 平台的第一个版本中，将 Web 服务引入 Java ME 平台，并提出了 Java SE 7 的战略和规划。他开发了 Java FX 技术，最近，他还为 Java EE 7 标准设计了 Java WebSocket API。从用 Java 编写代码到与行业专家设计 API，到他作为 Java Community Process Committee 主管的多年来，他对 Java 技术的多个方面都具有独到的见解。另外，他还是 *Java WebSocket Programming* 一书的作者，并且有一本关于 Java EE 的书也即将出版。Coward 博士拥有牛津大学数学系的学士、硕士和博士学位。

前　言

本书旨在帮助你学习 Java 程序设计的基础知识，采用循序渐进的教学方法，安排了许多示例、自测题和编程练习。本书不需要读者具备编程经验，而是从最基础的知识，从如何编译并运行一个 Java 程序开始讲起。然后讨论了构成 Java 语言核心的关键字、功能和结构。还介绍了 Java 的一些最重要的高级功能，如多线程编程和泛型。此外本书还介绍了 Swing 基础和 JavaFX。学完本书后，读者将会牢固地掌握 Java 编程精髓。

值得说明的是，本书只是学习 Java 的起点。Java 不仅仅是一些定义语言的元素，还包括扩展的库和工具来帮助开发程序。要想成为顶尖的 Java 程序员，就必须掌握这些领域。读者在学习完本书之后，就有了足够的知识来继续学习 Java 的其他方面。

0.1　Java 的发展历程

只有少数几种编程语言对程序设计带来过根本性的影响。其中，Java 的影响由于迅速和广泛而格外突出。可以毫不夸张地说，1995 年 Sun 公司发布的 Java 1.0 给计算机程序设计领域带来了一场变革。这场变革迅速地把 Web 转变成了一个高度交互的环境，也给计算机语言的设计设置了一个新标准。

多年以来，Java 不断地发展、演化和修订。和其他语言加入新功能的动作迟缓不同，Java 一直站在计算机程序设计语言的前沿，部分原因是其变革的文化，部分原因是它所面对的变化。Java 已经做过或大或小的多次升级。

第一次主要的升级是 Java 1.1 版，这次升级比较大，加入了很多新的库元素，修订了处理事件的方式，重新配置了 1.0 版本的库中的许多功能。

第二个主要的版本是 Java 2，它代表 Java 的第二代，标志着 Java 的“现代化”的到来。

Java 2 第一个发布的版本号是 1.2。Java 2 在第一次发布时使用 1.2 版本号看上去有些奇怪。原因在于，该号码最初指 Java 库的内部版本号，后来就泛指整个版本号了。Java 2 被 Sun 重新包装为 J2SE(Java 2 Platform Standard Edition)，并且开始把版本号应用于该产品。

Java 的下一次升级是 J2SE 1.3，它是 Java 2 版本首次较大的升级。它增强了一些已有的功能，并且紧凑了开发环境。J2SE 1.4 进一步增强了 Java。该版本包括一些重要的新功能，如链式异常、基于通道的 I/O 以及 assert 关键字。

Java 的下一版本是 J2SE 5，它是 Java 的第二次变革。以前的几次 Java 升级提供的改进虽然重要，但都是增量式的，而 J2SE 5 却从该语言的作用域、功能和范围等方面提供了根本性的改进。为了帮助理解 J2SE 5 的修改程度，下面列出了 J2SE 5 中的一些主要的新功能：

- 泛型
- 自动装箱/自动拆箱
- 枚举
- 增强型 for-each 形式的 for 循环
- 可变长度实参(varargs)
- 静态导入
- 注解(annotation)

这些条目都是重要的升级，每一个条目都代表了 Java 语言的一处重要改进。其中，泛型、增强型 for 循环和可变长度实参引入了新的语法元素；自动装箱和自动拆箱修改了语法规则；注释增加了一种全新的编程注释方法。

这些新功能的重要性反映在使用的版本号“5”上。从版本号的变化方式看，这一版本的 Java 应该是 1.5。由于新功能和变革如此之多，常规的版本号升级(从 1.4 到 1.5)已无法标识实际的变化，因此 Sun 决定使用版本号 5，以强调发生了重要改进。因此，当前的版本叫做 J2SE 5，开发工具包叫做 JDK 5。但是，为了保持和以前的一致性，Sun 决定使用 1.5 作为内部版本号，也叫做开发版本号。J2SE 5 中的“5”叫做产品版本号。

之后发布的 Java 版本是 Java SE 6，Sun 再次决定修改 Java 平台的名称，把“2”从版本号中删除了。因此，Java 平台现在的名称是 Java SE，官方产品名称是 Java Platform Standard Edition 6，对应的 Java 开发工具包叫做 JDK 6。和 J2SE 5 一样，Java SE 6 中的“6”是指产品的版本号，内部的开发版本号是 1.6。

Java SE 6 建立在 J2SE 5 的基础之上，做了进一步的增强和改进。Java SE 6 并没有对 Java 语言本身添加较大的功能，而是增强了 API 库，添加了多个新包，改进了运行时环境。它在漫长的生命周期(Java 术语)内经历了一些更新，添加了一些升级功能。总之，Java SE 6 进一步巩固了 J2SE 5 建立的领先地位。

Java 的下一版本是 Java SE 7，对应的 Java 开发工具包叫做 JDK 7，内部版本号是 1.7。Java SE 7 是 Oracle 收购 Sun Microsystems 之后发布的第一个主版本。Java SE 7 包含许多新功能，对语言和 API 库做了许多增强。Java SE 7 添加的最重要的功能是在 Project Coin 中开发的那些功能。Project Coin 的目的是确保把对 Java 语言所做的很多小改动包含到 JDK 7 中，其中包括：

- 现在 String 可以控制 switch 语句。
- 二进制整型字面值。

- 在数值字面值中使用下划线。
- 新增一种叫做 try-with-resources 的 try 语句，支持自动资源管理。
- 构造泛型实例时，通过菱形运算符使用类型推断。
- 增强了异常处理，可以使用单个catch捕获两个或更多个异常(多重捕获)，并且对重新抛出的异常可以进行更好的类型检查。

可以看到，虽然 Project Coin 中的功能被视为小改动，但是“小”这个词实在不能体现它们所带来的好处。特别是，try-with-resources 语句对大量代码的编写方式会产生深远的影响。

0.2　Java SE 8

Java 的最新版本是 Java SE 8，对应的开发工具包是 JDK 8，内部的开发版本号是 1.8。JDK 8 表示这是对 Java 语言的一次重大升级，因为本次升级包含了一种意义深远的新语言功能：lambda 表达式。lambda 表达式的影响深远：不但改变了概念化的编程方式，而且改变了 Java 代码的编写方式。使用 lambda 表达式，可以简化并减少创建某个结构所需的源代码量。另外，使用 lambda 表达式还可以将新的运算符->和一种新的语法元素引入到 Java 语言中。lambda 表达式有助于确保 Java 成为用户所期望的充满活力且敏捷的语言。

除了 lambda 表达式，JDK 8 中还新增了一些其他重要的功能。例如，从 JDK 8 开始，通过接口可以为指定的方法定义默认实现。JDK 8 也捆绑了对 JavaFX、Java 的新 GUI 框架的支持。期待 JavaFX 不久便能够在几乎所有的 Java 应用程序中扮演重要的角色，并且最终代替用于大多数基于 GUI 项目的 Swing。总之，Java SE 8 这一主要版本扩展了 Java 语言的功能，并且改变了 Java 代码的编写方式，带来的影响足够深远，在几年后将体现出来。本书中的内容已更新至 Java SE 8 版本，包括一些新的功能、更新和其他内容。

0.3　本书的组织结构

本书采用教程式的组织结构，每一章都建立在前面的基础之上。本书共分 17 章，每一章讨论一个有关 Java 的主题。本书的特色就在于包含了许多便于读者学习的特色内容。

- 关键技能与概念：每一章都首先介绍一些该章中要介绍的重要技能。
- 自测题：每一章都有自测题，测试读者学习到的知识。答案在附录 A 中提供。
- 专家问答：每一章中都穿插一些“专家问答”，以一问一答的形式介绍补充知识和要点。
- 编程练习：每一章中都包含一或两个编程练习，以帮助读者将学习到的知识应用到实践中去。很多这样的练习都是实际的示例，读者可以将其用作自己的程序的起点。

0.4　本书不需要读者具备编程经验

本书假定读者没有任何编程经验。如果读者没有编程经验，阅读本书是正确的选择。如果读者有过一些编程经验，在阅读本书时可以加快速度。但是要记住，Java 在几个重要的地方与其他一些流行的计算机语言不同，所以不要急于下结论。因此，即使读者是经验丰富的程序员，也仍然建议仔细阅读本书。

0.5 本书需要的软件环境

要想编译和运行本书提供的所有程序，需要获得 Oracle 提供的最新版本的 Java Development Kit (JDK)。在撰写本书时，最新版本为 JDK 8，这是 Java SE 8 使用的 JDK 版本。本书在第 1 章介绍如何获得 Java JDK。

如果读者使用早期版本的 Java，也仍然可以阅读本书，只是无法编译和运行使用了 Java 新功能的程序。

0.6 不要忘记 Web 上的代码

本书所有示例和编程项目的源代码都可以免费从 Web 网址 www.oraclepressbooks.com 和 www.tupwk.com.cn/download 获得。

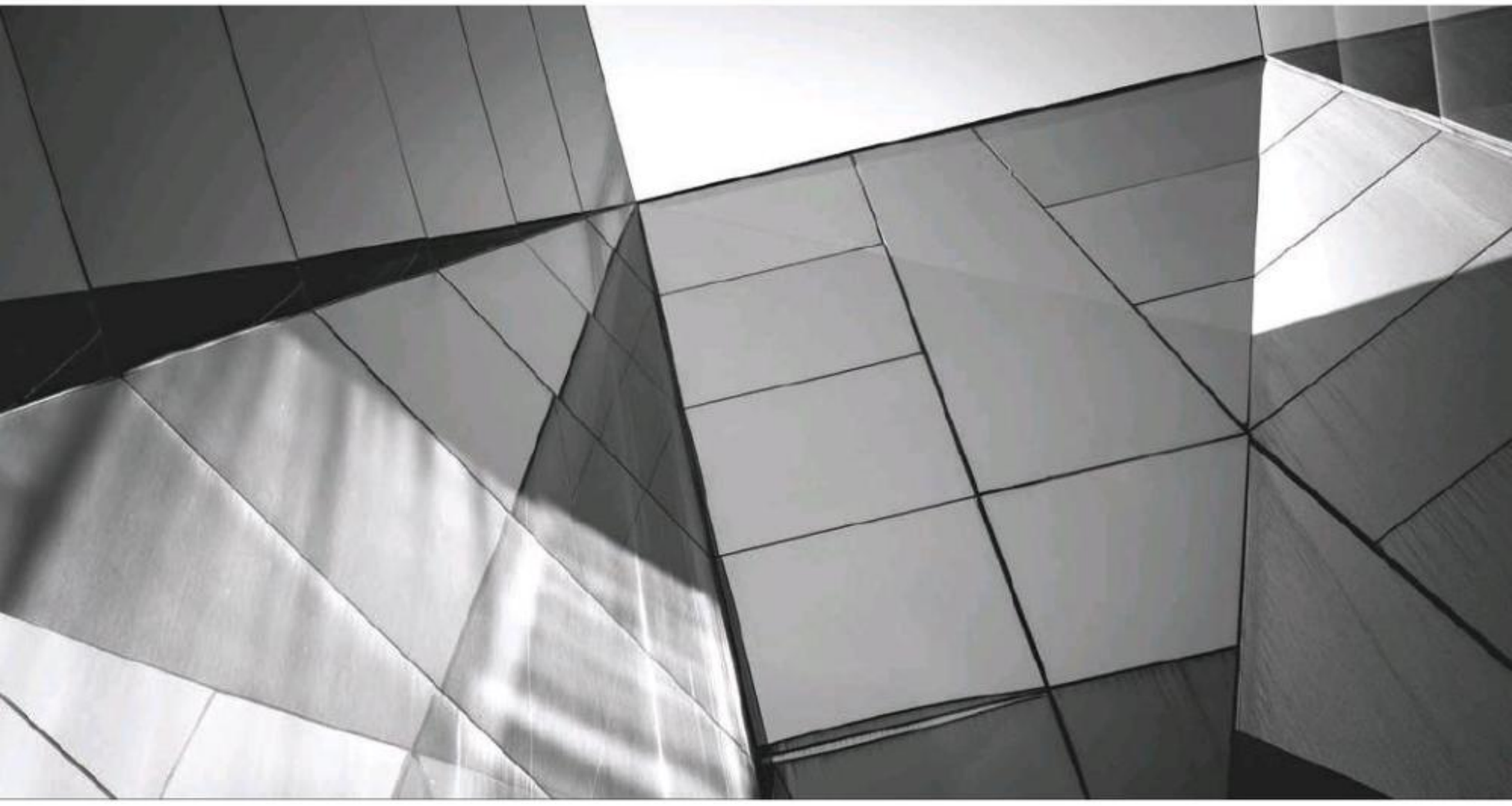
0.7 特别感谢

特别感谢本书的技术编辑 Danny Coward。他对本书提出了许多宝贵的建议和意见，对此十分感谢。

本书是引导读者进入 Herb Schildt 系列编程图书的大门，对下面的一些书你也会感兴趣：

- Java 8 编程参考官方教程(第 9 版)
- 新手学 JavaFX

目　录



第 1 章

Java 基础

关键技能与概念

- 了解 Java 的历史和基本原理
- 理解 Java 对 Internet 的贡献
- 理解字节码的重要性
- 了解 Java 的术语
- 理解面向对象程序设计的基本原理
- 创建、编译和运行一个简单的 Java 程序
- 使用变量
- 使用 if 和 for 控制语句

- 创建代码块
- 理解如何定位、缩进和终止语句
- 了解 Java 关键字
- 理解 Java 标识符的规则

Internet 和 World Wide Web 的兴起从根本上改变了计算的处理方式。在 Web 出现之前，网络空间还是由孤立的 PC 统治。而今天，几乎所有的 PC 都连接到了 Internet。最初，Internet 本身只是用于提供一种共享文件和信息的捷径，但是今天，它已经演变成浩瀚的分布式计算空间。这些改变催生了一种新的编程方法，这就是 Java。

Java 是一门卓越的 Internet 语言，不仅如此，它还使程序设计发生了变革，改变了我们考虑程序的形式与功能的方式。今天，要成为职业的程序员就意味着要具备使用 Java 编程的能力，它就是这么重要。在本书的课程中，你将学习到掌握 Java 的必备技能。

本章旨在向你介绍 Java，包括它的历史、设计原理和一些最重要的特性。目前，学习程序设计语言最大的难点是语言的各部分之间不是相互孤立的，而是相互关联的。这种相互关联性在 Java 中尤为突出。事实上，只讨论 Java 的一个方面，而不涉及其他部分是非常困难的。为了帮助克服这一困难，本章对 Java 的几个特性进行了简单概述，其中包括 Java 程序的基本形式、一些基本的控制结构和运算符。对于这些内容我们并不进行深入讨论，而只是关注一下 Java 程序共有的一些概念。

1.1 Java 的起源

驱使计算机语言革新的因素有两个：程序设计技术的改进和计算环境的改变。Java 也不例外。在大量继承 C 和 C++的基础之上，Java 还增加了反映当前程序设计技术状态的功能与精华。针对在线环境的蓬勃发展，Java 为高度的分布式体系结构提供了流水线程序设计的功能。

Java 是 1991 年由 Sun Microsystems 公司的 James Gosling、Patrick Naughton、Chris Warth、Ed Frank 和 Mike Sheridan 共同构想的成果。这门语言最初名为“Oak”，于 1995 年更名为“Java”。多少有些让人吃惊的是，设计 Java 的最初动力并不是源于 Internet，而是为了开发一种独立于平台的语言，使其能够用于创建内嵌于不同家电设备(如烤箱、微波炉和遥控器)的软件。你可能已猜到，不同类型的 CPU 都可以用作控制器。麻烦在于当时多数的计算机语言都是针对特定的目标而设计的，例如 C++。

虽然任何类型的 CPU 或许都能编译 C++程序，然而这需要 CPU 有完整的 C++编译器。而开发编译器的成本很高，并且很耗时。为了找到更好的解决方法，Gosling 和其他人尝试开发一种可移植的跨平台语言，使该语言生成的代码可以在不同环境下的不同 CPU 上运行。这一努力最终导致 Java 的诞生。

大概就在即将设计出 Java 细节的时候，另一个对 Java 的成型有更重要影响的因素出现了。第二个动力就是 World Wide Web。如果 Web 没有在 Java 即将成型的时候问世，那么 Java 可能会成为对消费类电子产品的程序设计而言有用但却晦涩的语言。然而随着 Web 的出现，以及 Web 对可移植程序的需求，Java 被推到了计算机语言设计的前台。

大多数程序员在工作不久就了解到可移植程序既令人期待，也让人难以捉摸。虽然在有了程序设计学科时就有了对创建高效可移植(平台独立)程序的需要，但还是让位于其他一些更为迫切的问题。Internet 和 Web 的出现使原有的可移植性问题重新摆上了桌面。因为，Internet 毕竟是由许多类型的计算机、操作系统和 CPU 组成的多样化的分布式空间。

曾经恼人心绪，却没那么重要的问题也就成为要亟待解决的问题。到 1993 年，Java 设计团队的成员发现，在创建嵌入式控制器时经常遇到的可移植性问题同样也出现在创建的 Internet 代码中。了解到这一点以后，Java 的重点从消费类电子产品转移到了 Internet 程序设计。因此，尽管开发独立于体系结构的程序设计语言的初衷提供了起初的星星之火，然而却是 Internet 最终促成了 Java 的燎原之势。

1.1.1　Java 与 C 和 C++的关系

Java 与 C 和 C++直接相关。Java 继承了 C 的语法，Java 的对象模型是从 C++改编而来的。Java 与 C 和 C++的关系之所以重要，是出于以下几个原因：

首先，许多程序员都熟悉 C/C++语法。这样对于他们而言，学习 Java 就简单了。同样，Java 程序员学习 C/C++也是很简单的。

其次，Java 设计者并没有重复工作。相反，他们进一步对已经成功的程序设计范式进行了提炼。现代程序设计始于 C，而后过渡到 C++，现在则是 Java。通过大量的继承和进一步的构建，Java 提供了强大的、可以更好利用已有成果的、逻辑一致的程序设计环境，并且增加了在线环境需求的新功能。然而，最重要的一点或许在于，由于它们的相似性，C、C++和 Java 为专业程序员定义了统一的概念架构。程序员从其中一种语言转为另一种语言时，不会遇到太大的困难。

C 和 C++的核心设计原理之一就是程序员拥有控制权。Java 也继承了这一原理。除了 Internet 环境施加的约束以外，Java 为程序员提供了完全的控制权。如果程序编写得出色，就会体现出来；而如果较糟糕，也会体现出来。换句话说，Java 并不是一种教学式语言，而是为专业程序员准备的语言。

Java 还有与 C 和 C++共有的特性：都由真正的程序员设计、测试和修改，与设计者的需求和经验紧密结合。因此，再没有比这更好的方法来创建如此一流的专业程序设计语言了。

因为 Java 与 C++的相似性，特别是它们对面向对象程序设计的支持，所以有些程序员可能会将 Java 简单地看做“C++的 Internet 版”。然而，这种观点是错误的。因为 Java 在实际应用以及基本原理上与 C++有着显著的区别。尽管 Java 受到 C++的影响，但它绝不是 C++的增强版。例如，Java 不提供对 C++的向上或向下兼容。当然，Java 与 C++的相似是十分明显的，如果你是一名 C++程序员，那么在使用 Java 时会有驾轻就熟的感觉。另外，Java 不是为替代 C++而设计的，而是为了解决一系列特定问题而设计的。C++则是用来解决另一个不同系列的问题。两者将在未来共存。

1.1.2　Java 与 C#的关系

在 Java 问世以后没几年，Microsoft 开发出了 C#语言。C#与 Java 密切相关。事实上，C#的许多功能都是直接从 Java 改编而来的。Java 和 C#共享相同的 C++语法风格，都支持分布式程序设计，使用相同的对象模型。它们之间当然也有不同之处，但就整体感觉而言，两者

极为相似。这就意味着，如果已经了解了 C#，那么学习 Java 就很简单；反之，如果将来要学的是 C#，那么现在学到的有关 Java 的知识也会对你将来有所帮助。

鉴于 Java 与 C#两者的相似性，自然有人要问，“C#会替代 Java 吗？”答案是否定的。Java 和 C#是对两种不同类型计算环境的优化，正如 C++会和 Java 长期共存一样，C#和 Java 也会长期共存。

1.2 Java 对 Internet 的贡献

Internet 帮助 Java 走到了程序设计的前台，而 Java 也对 Internet 产生了深远的影响。除了从整体上简化了 Web 编程任务之外，Java 还创造了一种全新的网络程序类型——applet，applet 改变了在线世界对于内容的看法。Java 还解决了与 Internet 相关的棘手问题：可移植性和安全性。下面分别进行详述讨论。

1.2.1 Java applet

applet 是一种特殊的 Java 程序，用于在 Internet 上传输，由兼容 Java 的 Web 浏览器自动执行。而且可以根据需要下载 applet，无须和用户进一步交互。如果用户点击一个包含 applet 的链接，applet 就会在浏览器自动下载并运行。它们通常用来显示服务器提供的数据，处理用户输入，或者提供在本地而非服务器上执行的简单功能(如贷款计算器)。也就是说，applet 支持把一些功能从服务器转移到客户端。

applet 的产生改变了 Internet 的编程方式，因为它使得对象可以在网络空间自由地移动。一般而言，有两种主要的对象类别可以在服务器和客户端之间传递：被动信息和动态的活动程序。例如，当读取电子邮件时，就是在查看被动数据。即使是在下载程序，程序的代码在执行之前也是被动的数据。与此不同的是，applet 是动态的、自执行程序。这样的程序是客户端计算机上的活动代理，但是由服务器初始化。

尽管人们很需要动态网络程序，但是它们也在安全性与可移植性领域带来了严重问题。很明显，要在客户端计算机上自动下载并且执行的程序必须保证不会带来危害。它还要能够在各种不同环境和不同操作系统中运行。Java 高效完美地解决了该问题。下面将逐一详细介绍。

1.2.2 安全性

正如你可能意识到的，每次下载一个“普通”程序时都可能会感染病毒、“木马程序”或其他有害代码。问题的核心在于恶意代码获得了对系统资源未授权的访问，因此可能会带来危害。例如，病毒程序可能会通过搜索计算机的本地文件系统获取私人信息，如信用卡号、银行账户余额及密码。Java 为了让 applet 安全地在客户端计算机上下载并执行，必须防止 applet 发动类似的攻击。

Java 实现这种保护功能的方法是将 applet 限制于 Java 执行环境中，不允许它访问计算机的其他部分(稍后，你会看到这是如何实现的)。很多人认为 Java 最重要的创新就是能够在下载 applet 时确信对客户计算机无害，没有安全隐患。

1.2.3　可移植性

可移植性是 Internet 要考虑的主要问题之一，因为与 Internet 连接的计算机和操作系统有多种类型。如果 Java 程序要运行在与 Internet 连接的任何计算机上，就需要某种机制确保程序能够在不同的系统中执行。例如，对于 applet 而言，同样的 applet 必须能够由各种与 Internet 连接的不同的 CPU、操作系统和浏览器下载和执行。对不同计算机采用不同版本的 applet 是一种不可行的做法。相同的代码必须能够在所有的计算机上工作，因此需要有某种机制来生成可移植的可执行代码。正如你稍后会看到的，确保安全性的机制同时也有助于确保创建可移植的代码。

1.3　Java 的魔法：字节码

Java 能够同时解决前面提到的安全性问题和可移植问题的关键在于，Java 编译器的编译结果不是可执行代码，而是字节码(bytecode)。字节码是一系列设计用来由名为 Java 虚拟机(Java Virtual Machine，JVM)的 Java 运行时系统执行的高度优化的指令。确切地讲，初始的 Java 虚拟机是一个字节码解释器。这可能会让你有些吃惊。因为如你所知，出于性能考虑，多数现代语言是用来被编译，而不是被解释的。然而，Java 程序由 JVM 执行这一事实帮助解决了与基于 Web 的程序相关的主要问题。下面就是原因所在。

将 Java 程序解释成字节码会使不同环境下的程序运行都变得十分轻松，因为只需要对每个平台实现 Java 虚拟机。一旦给定系统有了运行时包，那么在它上面就可以运行任何 Java 程序。切记，尽管平台之间的 Java 虚拟机不尽相同，但是它们都可以理解相同的 Java 字节码。如果 Java 程序被编译成了本机代码，那么一个程序就要为与 Internet 相连的每种 CPU 准备一种不同的版本。显然，这不是一种可行的解决方案。因此，由 JVM 执行字节码是创建真正可移植程序的最简单的方法。

Java 程序由 JVM 执行这一事实也使其更加安全。因为每一个 Java 程序的执行都处于 JVM 的控制之下，JVM 可以包含程序，防止程序在系统外产生副作用。此外，Java 语言中的一些限制也增强了安全性。

当一个程序被解释时，它的总体运行速度要比该程序被编译为可执行代码时的执行速度慢许多。然而，对于 Java，两者的区别却并不是很明显。因为字节码已被高度优化，所以使用字节码会使 JVM 执行程序的速度比你想象的快许多。

尽管 Java 被设计为解释型的语言，但这在技术上并不妨碍 Java 的字节码也可以迅速编译为本机代码。基于这一原因，Sun 在 Java 初始版本发布后不久就提供了其 HotSpot 技术。HotSpot 提供了一个 JIT(Just In Time)字节码编译器。当 JIT 成为 JVM 的一部分后，它可以根据逐条命令将字节码实时转换为可执行代码。因为 Java 执行的各种检查只有在运行时才进行，所以不可能将整个 Java 程序一次性编译为可执行代码，而是在执行期间需要时 JIT 才编译代码，理解这一点是非常重要的。而且，并不是所有的字节码序列都被编译，只有那些能够从编译受益的字节码才会被编译。其余的代码会被简单地解释。尽管如此，JIT 方法也使得性能有了显著提升。因为 JVM 依然控制着执行环境，所以甚至是在对字节码进行动态编译时，可移植性与安全性也可以保证。

专家解答

问：我听说有一种特殊的 Java 程序叫做 servlet，它是什么？

答： servlet 是一种在服务器上执行的小程序。就像 applet 动态地扩展了 Web 浏览器的功能一样，servlet 动态地扩展了 Web 服务器的功能。理解 applet 的作用有助于理解 servlet 的作用，它们分别作用在客户端和服务器上。Java 的最初版本发布后不久，就很清楚地表现出在服务器端也能胜任。结果就产生了 servlet。servlet 的出现意味着 Java 同时占领了客户端和服务器端。尽管创建 servlet 的任务超出了本书的讨论范围，不过在读者以后的 Java 编程工作中，还是需要去学习的。笔者撰写的《Java 完全参考手册》(由清华大学出版社引进并出版)一书详细介绍了 servlet。

1.4 Java 的主要术语

不了解 Java 的术语就无法对 Java 进行完整的概述。尽管促使 Java 产生成为必然的根本原因在于安全性和可移植性，但是一些其他因素对于 Java 语言的最后形成也起到了重要的作用。表 1-1 所示的术语汇总了 Java 设计团队所考虑的关键因素。

表 1-1 Java 的主要术语

术　语	说　明
简单(Simple)	Java 有一系列简洁、统一的功能，使其易于学习和使用
安全(Secure)	Java 提供了创建 Internet 应用程序的安全方法
可移植(Portable)	Java 程序可以在任何具有 Java 运行时系统的环境中执行
面向对象(Object-Oriented)	Java 代表了现代的面向对象编程理念
健壮(Robust)	Java 通过进行严格的输入和执行运行时错误检查提倡无错程序设计
多线程(Multithreaded)	Java 提供对多线程程序设计的集成支持
体系结构中立(Architecture-Neutral)	Java 并不局限于特定的计算机或操作系统体系结构
解释型(Interpreted)	通过使用 Java 字节码，Java 支持跨平台代码
高性能 (High Performance)	Java 字节码的执行速度被高度优化
分布式(Distributed)	Java 被特意设计用于在 Internet 的分布式环境中使用
动态(Dynamic)	Java 程序带有大量在运行时，用于检查和解决对象访问的运行时类型信息

专家解答

问： 为解决可移植性与安全性问题，为什么需要创建一种像 Java 这样的新的计算机语言？难道不能修改像 C++这样的语言吗？换言之，难道不能创建一个可以输出字节码的 C++编译器吗？

答： 尽管 C++编译器可以生成类似于字节码而非可执行文件的东西，但是 C++的一些特性却不支持创建 Internet 程序，其中一个最重要的特性就是 C++对指针的支持。指针(pointer)是存储在内存中的一些对象的地址。如果使用指针，那就可能访问程序以外的资源，这样就破坏了安全性。Java 不支持指针，也就不存在这个问题。

1.5　面向对象程序设计

Java 的核心是面向对象程序设计(Object-Oriented Programming, OOP)。面向对象方法论与 Java 是密不可分的，而 Java 所有的程序至少在某种程度上都是面向对象的。因为 OOP 对 Java 的重要性，所以在开始编写一个哪怕是很简单的 Java 程序之前，理解 OOP 的基本原理都是非常有用的。在本书最后，将解释如何将这些概念应用到实践中。

OOP 是一种功能强大的程序设计方法。从计算机诞生以来，为适应程序不断增加的复杂度，程序设计方法论也发生了巨大的变化。例如，在计算机最初被发明时，程序设计是通过使用计算机面板输入二进制机器指令来完成的。只要程序仅限于几百条指令，这种方法就是可以接受的。随着程序的增长，汇编语言被发明了，这样程序员就可以使用代表机器指令的符号表示法来处理大型的、复杂的程序。随着程序的继续增长，高级语言的引入为程序员提供了更多的工具，这些工具可使他们处理更复杂的程序。当然，第一个广泛使用的语言是 FORTRAN。尽管 FORTRAN 是人们迈出的颇具影响的第一步，但是它很难设计出清晰、简洁且易懂的程序。

20 世纪 60 年代诞生了结构化程序设计方法，C 和 Pascal 这样的语言鼓励使用这种方法。结构化语言的使用使得编写中等复杂程度的程序变得相当轻松。结构化语言的特点是支持孤立的子例程、局部变量、丰富的控制结构和不使用 GOTO 语句。尽管结构化语言是一个功能强大的工具，但是在项目很大时仍然显得有些捉襟见肘。

考虑一下：程序设计发展的每个里程碑，技术和工具都是为了使程序员处理日渐复杂的程序而创建的。在这条道路上的每一步，新的方法都吸收了过去方法的精华而不断前进。OOP 出现之前，许多项目已经接近(超过)结构化方法工作的极限。于是，为了冲破这一束缚，就创建了面向对象方法。

面向对象程序设计采纳了结构化程序设计的思想精华，并且用一些新的概念与之结合。这样的结果就是一种新的程序组织方法的产生。广义上讲，一个程序可以用下面两种方法来组织：一种是围绕代码(发生了什么)，一种是围绕数据(谁受了影响)。如果仅使用结构化程序设计技术，那么程序通常围绕代码来组织。这种方法可以被认为是“代码作用于数据”。

面向对象程序则以另一种方式工作。它们以“数据控制访问代码”为主要原则，围绕数据来组织程序。在面向对象语言中，需要定义数据和作用于数据的例程。这样，数据类型可以精确地定义出哪种类型的操作可以应用于该数据。

为了支持面向对象程序设计的原理，所有 OOP 语言，包括 Java 在内，都有三个特性：封装(encapsulation)、多态性(polymorphism)和继承(inheritance)。下面，我们对此一一学习。

1.5.1 封装

封装是一种将代码与它所处理的数据结合起来，而不被外界干扰滥用的程序设计机制。在面向对象语言中，代码和数据可以使用创建自包含的黑盒(black box)的方式捆绑在一起。盒子中包含了所有必需的数据和代码。当代码和数据以这种方式链接在一起时，就创建了对象。换言之，对象是支持封装的。

在对象中，代码或数据，或者两者对该对象都可以是私有的(private)或公有的(public)。私有代码或数据仅被对象的其他部分知晓或访问，即私有代码或数据不能被该对象以外的任何程序部分所访问。当代码或数据是公有的时，虽然它们是定义在对象中的，但程序的其他部分也可以对其进行访问。通常，对象的公有部分用于为对象的私有元素提供控制接口。

Java 的基本封装单元是类(class)。虽然本书后面将会详尽地介绍类，但是下面对类的简述也会对你有所帮助。类定义了对象的形式，指定了数据和操作数据的代码。Java 使用类规范来构造对象。对象是类的实例。因此，类在本质上是指定如何构建对象的一系列规定。

组成类的代码或数据称为类的成员(member)。具体而言，类定义的数据称为成员变量(member variable)或实例变量(instance variable)。处理这些数据的代码则称为成员方法(member method)或简称为方法(method)。方法是子例程在 Java 中的术语。如果熟悉 C/C++，那么知道 Java 程序员所称的“方法”就是 C/C++程序员所称的“函数”会有所帮助。

1.5.2 多态性

多态性(polymorphism)是一种允许使用一个接口来访问一类动作的特性。特定的动作由不同情况的具体本质而定。汽车的方向盘就是一个简单的多态性示例。无论实际的方向控制机制是什么类型的，方向盘(也就是接口)都是一样的。也就是说，无论汽车是手动操纵、电力操纵还是齿轮操纵，方向盘使用起来都是一样的。因此，只要知道如何操作方向盘，就可以驾驶任何类型的汽车。

同样的原理也可以应用于程序设计。考虑一下堆栈(先进后出)，你的程序可能需要三个不同类型的堆栈：一个用于处理整型值，另一个用于处理浮点型值，还有一个用于处理字符。在这个示例中，尽管堆栈存储的数据类型是不同的，但是实现各个堆栈的算法都是一致的。在非面向对象的语言中，需要创建三个不同的堆栈例程，每个例程使用不同的名称。然而，在 Java 中，由于多态性的使用，可以创建一个基本的堆栈例程来为这三种特定的情况服务。这样，只要知道如何使用一个堆栈，就能够使用所有的堆栈。

更普遍的是，多态性的概念常常被表述为“单接口，多方法”。这就意味着为一组相关的活动设计一个泛型接口是可能的。多态性通过允许同一接口指定一类动作减少了程序的复杂度。编译器的工作就是选择适用于各个情况的特定动作(也就是方法)。而程序员则无须手动进行这样的选择。你要做的仅仅是记住，以及利用这个统一的接口。

1.5.3　继承

继承是一个对象获得另一个对象的属性的过程。继承之所以重要，是因为它支持层次结构类的概念。思考一下就会发现，许多知识都是通过层次结构(即从上至下)方式进行管理的。例如，红色美味的苹果是苹果类的一部分，而苹果又是水果类的一部分，水果类则是食物类的一部分。即食物类具有的某些特性(可食用、有营养等)也适用于它的子类——水果。除了这些特性以外，水果类还具有与其他食物不同的特性(多汁、味甜等)。苹果类则定义了属于苹果的特性(生长在树上、非热带水果等)。而红色味美的苹果继承了前面所有类的属性，还会定义自己特有的属性。

如果没有使用层次结构，那么对象就不得不明确定义出自己的所有特性。如果使用继承，那么对象就只用定义使自己在类中与众不同的属性就可以了，至于基本的属性，可以从自己的父类继承。因此，正是继承机制使得对象能够成为更为一般的类的特定实例。

1.6　获得 Java 开发工具包

既然已经解释了 Java 的理论基础，现在就应该开始编写 Java 程序了。然而在编译并运行这些程序之前，必须在计算机上安装一个 Java 开发包(Java Development Kit，JDK)。JDK 可以从 Oracle 免费获取(本书编写时，JDK 的最新版本是 JDK 8，Java SE 8 用的就是这个版本(SE 表示标准版本)。由于 JDK 8 包含许多以前版本不支持的新功能，因此读者在编译和运行本书的程序时，推荐使用 JDK 8 或更高版本。如果使用早期的版本，包含新特性的程序就不能通过编译。

JDK 可以从 www.oracle.com/technetwork/java/javase/downloads/index.html 免费下载。找到下载页，按照对计算机类型的指示下载安装即可。安装完 JDK 以后，就可以编译并运行程序了。JDK 支持两个主要程序。第一个是 Java 的编译器 javac。第二个是标准 Java 解释器 java，也称为 application launcher。

另外一点是 JDK 运行在命令提示环境中，且使用命令行工具。既不是窗口式的应用程序，也不是集成开发环境(Integrated Development Environment, SIDE)。

注意：

除了 JDK 提供的基本命令行工具以外，Java 程序员还可以使用一些高质量的 IDE，例如 NetBeans 和 Eclipse。在开发和部署商业应用程序时，IDE 十分有用。一般来说，如果愿意，也可以使用 IDE 来编译和运行本书中的程序。但是，本书中关于编译和运行 Java 程序的说明只针对 JDK 命令行工具。原因很简单。首先，所有读者都可以使用 JDK。其次，关于 JDK 的使用说明对所有读者都是一样的。最后，对于本书提供的简单程序，使用 JDK 命令行工具通常是最简单的方法。如果选择使用某个 IDE，就需要遵循该 IDE 的说明。因为不同 IDE 之间存在一些差别，所以不存在通用的指导说明。

专家解答

问： 你说面向对象程序设计是一种管理大型程序的有效方法。但是，它似乎会增加小型程序的潜在开销。既然你说所有的 Java 程序在一定程度上都是面向对象的，那么这会对小型程序造成不利影响吗？

答： 不会的。正如你所看到的，对于小型程序，Java 的面向对象特性几乎是透明的。尽管 Java 的确遵循严格的对象模型，但是你对它的使用程度可以有很宽的选择范围。对于小型程序而言，它们的面向对象性几乎是觉察不到的。而当你的程序增长时，就会轻松地用到更多的面向对象特性。

1.7 第一个简单的程序

让我们从编译并运行下面这个简单程序开始：

```
/*
  This is a simple Java program.

  Call this file Example.java.
*/
class Example {
  // A Java program begins with a call to main().
  public static void main(String args[]) {
    System.out.println("Java drives the Web.");
  }
}
```

你要遵循以下三个步骤：

(1) 输入程序。

(2) 编译程序。

(3) 运行程序。

1.7.1 输入程序

本书的所有程序都可从 McGraw-Hill Education 的网站获得：www.oraclepressbooks.com。然而，也可以尝试亲手输入程序。在本例中，必须自己使用文本编辑器来输入程序，而不能使用字处理程序。字处理程序通常会将格式信息与文本一同存储，而这些格式信息会使 Java 编译器不知所措。如果使用 Windows 平台，可以使用 WordPad 或其他任何喜欢的程序编辑器。

对于多数计算机语言而言，存储程序源代码的文件的名称是任意的，然而 Java 却不是这样。你要了解的关于 Java 的第一点就是源文件的命名是极为重要的。对于本例，源文件的名称应该为 Example.java。下面我们来看一下为什么要这样做。

在 Java 中，源文件的正式名称是编译单元(compilation unit)。它是一个包含一个或多个

类定义的文本文件(现在我们使用只包含一个类的源文件)。Java 编译器要求源文件使用.java 为文件扩展名。查看程序即可发现，程序定义的类的名称也是 Example。这并不是巧合。在 Java 中，所有的代码都必须驻留于一个类中。根据规则，类名应该与存储程序的文件名称相符，而且应该确保文件名的大小写与类名相符。这样做是因为 Java 区分大小写。此时文件名与类名的一致规则看似有些武断，然而正是这样的规则使得程序的维护与组织更为轻松了。

1.7.2　编译程序

为了编译 Example 程序，执行编译器 javac，这需要在命令行指定源文件的名称，如下所示：

```
javac Example.java
```

编译器 javac 创建一个包含程序字节码的名为 Example.class 的文件。切记，字节码不是可执行代码。字节码必须由 Java 虚拟机来执行。因此，javac 输出的代码是不可以直接执行的。

要真正运行程序，必须使用 Java 解释器 java。为此，需要将类名 Example 作为一个命令行实参来传递，如下所示：

```
java Example
```

当程序运行时，输出如下所示：

```
Java drives the Web.
```

编译 Java 源代码时，每一个类都放入文件名与该类类名相同的输出文件中，并以.class 为扩展名。这就是使 Java 源文件的名称与它们所包含的类名一致的原因。源文件的名称会与.class 文件名相匹配。当执行如前所示的 Java 解释器时，实际上要指定希望解释器执行的类名。解释器会自动寻找一个与该类名相同，且以.class 为扩展名的文件。如果它找到文件，就会执行包含在里面的特定类的代码。

注意

假设你正确安装了 JDK。如果在尝试编译程序时，计算机找不到 javac，就需要指定命令行工具的路径。例如，在 Windows 中，这意味着需要在 PATH 环境变量定义的路径中添加命令行工具的路径。例如，如果在 Program Files 目录中安装 JDK 8，那么命令行工具的路径将是 C:\Program Files\Java\jdk1.8.0\bin(当然，你需要找到自己计算机上的 Java 路径，该路径可能不同于此处所示的路径。JDK 的具体版本也会有所变化)。因为在不同的操作系统中，设置路径的过程不尽相同，所以需要参考所用操作系统的文档来设置路径。

1.7.3　逐行分析第一个程序

尽管程序 Example.java 非常短，但是它却包含了所有 Java 程序共有的几个特点。下面，我们仔细研究一下程序的各个部分。

程序以下面几行开始：

```
/*
  This is a simple Java program.

  Call this file Example.java.
*/
```

这是一个注释(comment)。与其他多数程序设计语言一样，Java 允许在程序源代码中输入注释。编译器会忽略注释的内容。而注释可以向任何阅读程序源代码的人员描述或解释程序的操作。本例中，注释描述了程序，并且提醒你源文件应该以 Example.java 命名。当然，在实际的应用程序中，注释一般用来解释程序的某些部分如何工作，或对特定的功能进行解释。

Java 支持三种形式的注释。在这个例子中，位于程序最上面的是多行注释(multiline comment)。这种类型的注释必须以“/*”开始，以“*/”结尾。这两个注释符号中间的任何内容都将被编译器忽略。顾名思义，多行注释可以有若干行。

程序的下一行代码如下所示：

```
class Example {
```

该行代码使用关键字 class 声明创建一个新类。如前所述，类是 Java 的基本封装单元。Example 是类的名称。类的定义以左花括号“{”开始，以右花括号“}”结束。两个花括号间的元素是类的成员。此时不必过于担心类的细节，只要知道 Java 中所有的程序活动发生在一个类中即可。这也就是 Java 程序都(至少有一点)具有面向对象性的原因之一。

如下所示，程序的下一行是一个单行注释(single-line comment)。

```
// A Java program begins with a call to main().
```

这是 Java 支持的第二种注释方式。单行注释以“//”开始，以行末为结尾。作为一项基本规则，程序员使用多行注释进行较长的描述，用单行注释进行简要的逐行描述。

下一行代码如下所示：

```
public static void main (String args[]) {
```

本行是 main()方法的开始。如前所述，在 Java 中，子例程称为方法(method)。正如它之前的注释所提示的那样，程序从这一行开始执行。所有 Java 应用程序的执行都是以调用 main()开始的。虽然对于本行各部分的意思现在不能一一尽述，因为这需要深入理解其他几个 Java 特性，但是由于本书的许多示例都用到了这行代码，所以我们现在对其进行简要的介绍。

关键字 public 是一个访问修饰符(access modifier)。访问修饰符用以决定程序其他部分如何访问类的成员。当类成员前面有 public 时，该成员就可以被声明它的类以外的代码访问(与 public 相反的是 private，它用于防止类以外的代码使用成员)。本例中，main()必须被声明为 public，因为它要在程序开始时被它的类以外的代码调用。关键字 static 允许 main()在类的对象被创建之前调用。这一点是必需的，因为 JVM 要在任何对象被创建之前调用 main()。关键字 void 只告知编译器 main()不返回值。正如你所看到的，方法也可以返回值。如果这些看起来有点让人头昏的话，不必担心，后面各章对所有这些概念都将详细讨论。

如前所述，main()是在 Java 应用程序开始时调用的方法。需要传递给方法的任何信息都将被方法名后面一对圆括号中指定的变量所接收。这些变量称为形参(parameter)。如果给定

的方法不需要形参，那么还需要包括一对空的圆括号。main()中只有一个形参 String args[ ]，它用来声明一个名为 args 的形参。这是一个 String 类型的对象数组(数组是相似对象的集合)。String 类型的对象用于存储字符序列。本例中，args 接收执行程序时出现的任何命令行实参。这个程序没有用到这一信息，但是本书后面的其他程序则会用到。

本行的最后一个字符是“{”。这是 main()的主体开始的标志。方法中的所有代码都包含在方法的左花括号与右花括号之间。

下一行代码如下所示，注意它出现在 main()内：

```
System.out.println("Java drives the Web.");
```

本行输出字符串“Java drives the Web.”，而且在屏幕上显示字符串后另起一行。输出实际上是由内置的 println()方法完成的。本例中，println()显示传递给它的字符串。如你所见，println( )也可以用于显示其他类型的信息。本行以 System.out 开始。虽然此时详细解释 System 还有些复杂，但是简单讲，System 是一个预定义类，它提供对系统的访问，而 out 是与控制台相连的输出流。因此，System.out 是一个封装控制台输出的对象。Java 使用对象来定义控制台输出这一事实是其面向对象本质的又一佐证。

你可能已经猜想到，控制台输出(和输入)在实际的 Java 程序中并不常用。因为多数现代计算机环境是窗口化、图形化的，所以控制台 I/O 多用于简单的工具程序和演示程序，以及服务器端代码。本书后面，你会学习使用 Java 产生输出的其他方法，但是现在，我们还要继续使用控制台 I/O 方法。

注意 println( )语句以分号结束。Java 中的所有语句都要以分号结束。程序中的其他行不以分号结尾是因为从技术上讲它们不是语句。

程序中的第一个“}”是用来结束 main()的，而最后一个“}”是用来结束 Example 类定义的。

最后提醒一点：Java 是区分大小写的。忘记这一点会给你带来很大的麻烦。例如，如果你偶尔将 main 输成了 Main，或者将 println 输成了 PrintLn，那么前面的程序就不正确了。而且，尽管 Java 编译器会编译不包含 main()方法的类，但它却无法执行它们。因此，如果你输错了 main，编译器虽然还会编译程序，但是 Java 解释器会报告一个错误，因为它找不到 main()方法。

1.8　处理语法错误

如果还没有输入、编译和运行前面的程序，那么请现在完成这些工作。正如你从以前的程序设计经验了解到的，向计算机输入代码时会很容易输入一些不正确的内容。幸运的是，如果向程序输入了不正确的内容，那么编译器会在编译时报告语法错误消息。无论输入的是什么，Java 编译器都会尝试理解源代码。出于这一原因，被报告的错误并不总是反映实际引起问题的地方。例如，在前面的程序中，在 main()方法后没有输入左花括号会导致编译器报告下列两条错误消息：

```
Example.java:8: ';' expected
  public static void main(String args[])
                                      ^
Example.java:11: class, interface, or enum expected
}
^
```

很明显，第一条错误消息是完全错误的，因为缺少的不是分号而是花括号。

这里讨论的关键是当程序包含语法错误时，并不必一定要从字面上理解编译器提供的消息，因为这些消息可能会误导你。需要推测错误消息以求找出真正的问题。另外，应该看看程序中被标记行之前的几行代码，有时报告错误的位置却是真正发生错误位置的后面几行。

1.9 第二个简单程序

对于程序设计语言而言，可能再没有任何结构比为变量赋值更为重要了。变量(variable)是可以被赋值的已命名内存位置。而且，变量的值在程序的执行过程中可以被修改。即变量的内容是可改动的，而不是固定的。下面的程序创建了两个变量：var1 和 var2。

```
/*
  This demonstrates a variable.

  Call this file Example2.java.
*/
class Example2 {
  public static void main(String args[]) {
    int var1; // this declares a variable      <-- 声明变量
    int var2; // this declares another variable

    var1 = 1024; // this assigns 1024 to var1  <-- 为变量赋值

    System.out.println("var1 contains " + var1);

    var2 = var1 / 2;

    System.out.print("var2 contains var1 / 2: ");
    System.out.println(var2);
  }
}
```

运行程序时，输出如下所示：

```
var1 contains 1024
var2 contains var1 / 2: 512
```

这个程序引入了几个新概念。第一个是声明整型变量 var1 的语句：

```
int var1; // this declares a variable
```

本书试读到此结束啦！

[Java 8编程入门官方教程]

作者： (美) 施密特 (Schildt,H.) , 著

出版社：清华大学出版社
