



Java: A Beginner's Guide, Eighth Edition

Java 11 官方入门教程 (第8版)

马上就能创建、编辑和运行Java程序！

[美] 赫伯特·希尔特(Herbert Schildt) 著
杜 静 敖富江 译



清华大学出版社

Java 11 官方入门教程

(第 8 版)

[美] 赫伯特 · 希尔特(Herbert Schildt) 著

杜静 敖富江 译

清华大学出版社

北京

Herbert Schildt

Java: A Beginner's Guide, Eighth Edition

EISBN: 9781260440218

Copyright © 2019 by McGraw-Hill Education.

All Rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including without limitation photocopying, recording, taping, or any database, information or retrieval system, without the prior written permission of the publisher.

This authorized Chinese translation edition is jointly published by McGraw-Hill Education and Tsinghua University Press Limited.

This edition is authorized for sale in the People's Republic of China only, excluding Hong Kong, Macao SAR and Taiwan.

Translation copyright © 2019 by McGraw-Hill Education and Tsinghua University Press Limited.

版权所有。未经出版人事先书面许可，对本出版物的任何部分不得以任何方式或途径复制或传播，包括但不限于复印、录制、录音，或通过任何数据库、信息或可检索的系统。

本授权中文简体字翻译版由麦格劳-希尔(亚洲)教育出版公司和清华大学出版社有限公司合作出版。此版本经授权仅限在中国大陆区域销售，不能销往中国香港、澳门特别行政区和中国台湾地区。

版权© 2019 由麦格劳-希尔(亚洲)教育出版公司与清华大学出版社有限公司所有。

北京市版权局著作权合同登记号 图字：01-2019-3436

本书封面贴有 McGraw-Hill Education 公司防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目(CIP)数据

Java 11 官方入门教程 / (美)赫伯特·希尔特(Herbert Schildt) 著；杜静，敖富江 译. —8 版. —北京：清华大学出版社，2019

书名原文：Java: A Beginner's Guide, Eighth Edition

ISBN 978-7-302-53605-5

I . ①J… II . ①赫… ②杜… ③敖… III. ①JAVA 语言—程序设计—教材 IV. ①TP312.8

中国版本图书馆 CIP 数据核字(2019)第 173922 号

责任编辑：王军 韩宏志

装帧设计：孔祥峰

责任校对：成凤进

责任印制：沈露

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

印 装 者：清华大学印刷厂

经 销：全国新华书店

开 本：190mm×260mm 印 张：33.25 字 数：1095 千字

版 次：2019 年 9 月第 1 版 印 次：2019 年 9 月第 1 次印刷

定 价：99.80 元

产品编号：083282-01

译 者 序

Java 在全球编程语言排行榜中稳居第二，Java 语言到底有哪些优势呢？

第一，Java 是一种跨平台语言，其主旨是“一次编写，到处运行”。第二，Java 语法比较简单，JVM 为开发者屏蔽了大量复杂细节，学过计算机编程的开发者都能快速上手。第三，Java 能力过硬，在多个领域的竞争力都非常强；Java 用途广泛，可用来开发传统的客户端软件和网站后台，也可用来开发如火如荼的 Android 应用和云计算平台，如服务器端编程、企业软件事务处理、大数据处理、分布式计算、移动开发、嵌入终端开发等。第四，Java 吸收了业内领先的工程实践，具备构建嵌入式设备乃至超大规模软件系统的能力。所有这些使 Java 得到软件和互联网公司的青睐。

时移世易，Java 正在改变，也必须改变。Java 改为每半年发布一次后，在合并关键特性等方面做得越来越好，各厂商和社区对 Java 的投入越来越大。最新发布的 JDK 11 虽然谈不上划时代的进步，但一定是 JDK 发展历程中的一个重要里程碑，升级 JDK 即可提升性能，获得基础能力的全面进步和突破，这一切无不说明，是时候开始评估和规划升级到 JDK 11 了。

JDK 11 中主要的新语言特性是支持局部变量类型推断以及在 lambda 表达式中使用 var，还向 Java 启动程序添加了一种执行模式，使其能直接执行简单的单文件程序。JDK 11 取消了对 applet 的支持，删除了 JavaFX，不再支持与部署相关的 Java Web Start。

本书采用循序渐进的教学方法，旨在帮助读者学习 Java 程序设计的基础知识。全书共分 16 章，每一章都重点讨论 Java 的一个方面，还安排了许多示例、自测题和编程练习。本书不要求读者具有编程经验；首先介绍基础知识，然后讨论构成 Java 语言核心的关键字、功能和结构，最后介绍 Java 的一些重要高级功能，如多线程编程、泛型、lambda 表达式、模块和 Swing。学完本书后，读者将领悟到 Java 编程精髓。

本书只是学习 Java 的起点。Java 还包括扩展的库和工具。要成为顶尖的 Java 程序员，就必须掌握这些知识。读者在学完本书后，就有了足够的知识来继续学习 Java 的其他方面。

本书内容清晰，详略得当，附有大量程序实例，包含 Java 语言基础语法以及一些高级特性，极具实用价值，是 Java 初学者和 Java 程序员的必备参考书，也是高等院校的绝佳 Java 语言教材。

这里要感谢清华大学出版社的编辑们，他们为本书的翻译投入了巨大热情，付出了很多心血。没有他们的帮助和鼓励，本书不可能顺利付梓。

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”，但鉴于译者水平有限，错误在所难免，如有任何意见和建议，请不吝指正。

译 者

作 者 简 介



Herbert Schildt 是权威的 Java 语言专家、畅销书作家。三十多年来，Herbert 撰写的程序设计图书在全球的销量达数百万册，并被翻译成多种语言。Herbert 已撰写大量关于 Java、C++、C 和 C# 编程语言的书籍和文章，包括《Java 9 编程参考官方大全(第 10 版)》、*Herb Schildt's Java Programming Cookbook*、*Introducing JavaFX 8 Programming* 和 *Swing: A Beginner's Guide*。

Herbert 对计算机的各个方面充满兴趣，其中投入精力最多的是计算机语言，尤其是计算机语言的标准化。Herbert 是 ANSI/ISO 委员会的成员，参与了 1989 年 C 语言的标准化、1998 年 C++ 语言的标准化，以及 2011 年 C++ 标准的更新。Herbert 拥有伊利诺伊大学的学士和硕士学位。他的个人网站为 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 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 JavaFX 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 主管这么多年来，他对 Java 技术的多个方面都有独到的见解。另外，他还是 *Java WebSocket Programming* 和 *Java EE: The Big Picture* 两本书的作者。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 Java 程序开始讲起。然后讨论构成 Java 语言核心的关键字、功能和结构。还介绍 Java 的一些最重要高级功能，如多线程编程、泛型、lambda 表达式和模块。此外，本书还介绍 Swing 基础。学完本书后，读者将牢固掌握 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，就必须掌握这些知识。读者在学完本书后，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。这场变革迅速将 Web 转变成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。

多年来，Java 不断发展、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，部分原因是其不断变革的文化，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。

第二个主要版本是 Java 2，它代表 Java 的第二代，标志着 Java “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。它增强了已有的功能，精简了开发环境。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改进。为帮助理解 J2SE 5 的修改程度，下面列出了 J2SE 5 中的一些主要新功能：

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

- 注解(annotation)

这些条目都是重要升级，每个条目都代表了 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；注解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)已无法标识变化的幅度，因此 Sun 决定使用版本号 5，以强调发生了重要改进。因此将这个版本称为 J2SE 5，将开发工具包称为 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，也称为开发版本号。J2SE 5 中的“5”称为产品版本号。

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。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 开发工具包称为 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。

接下来的版本是 Java SE 7，对应的 Java 开发工具包称为 JDK 7，内部版本号是 1.7。Java SE 7 是 Oracle 收购 Sun Microsystems 之后发布的一个主版本。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 可控制 switch 语句。
- 二进制整型字面值。
- 在数值字面值中使用下画线。
- 新增一种称为 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)，并且可以对重新抛出的异常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 语句会对大量代码的编写方式产生深远影响。

此后的版本是 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 表达式还可将新的运算符-> 和一种新的语法元素引入 Java 语言中。

除了 lambda 表达式，JDK 8 中还新增了其他一些重要功能。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。总之，Java SE 8 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。

再后的 Java 版本是 Java SE 9，对应的开发工具包是 JDK 9。JDK 9 表示这是对 Java 语言的一次重大升级，合并了对 Java 语言及其库的重大改进。主要的新功能是模块，它允许指定构成应用程序的代码之间的关系和依赖。模块还给 Java 的访问控制功能添加了另一种方式。包括模块导致一个新的语法元素、几个新的关键字和各种工具改进被添加到 Java 中。模块还对 API 库具有深远的影响，因为从 JDK 9 开始，库包现在组织为模块。

除了模块之外，JDK 9 还包括几个新功能。其中一个特别有趣的是 JShell，它是一个支持交互式程序体验和学习的工具(有关 Jshell 的简介，见附录 D)。另一个有趣的升级是支持私有接口方法。包含它们进一步增强了 JDK 8 对接口中默认方法的支持。JDK 9 给 javadoc 工具添加了搜索功能，还添加了一个新的标记@index 来支持它。与以前的版本一样，JDK 9 包含对 Java API 库的许多更新和改进。

作为一般规则，在任何 Java 版本中，都有令人瞩目的新功能。但 JDK 9 废弃了 Java 高度配置的一个方面：

applet。从 JDK 9 开始，applet 不再推荐在新项目中使用。如第 1 章所述，因为 applet 需要浏览器支持以及其他一些因素，JDK 9 废弃了整个 applet API。

Java 的下一个版本是 Java SE 10 (JDK 10)。然而，在发布它之前，Java 发布计划发生了重大变化。过去，主要发行版通常间隔两年或更长时间。然而，从 JDK 10 开始，发行版之间的时间明显缩短了。现在预计发布将严格按照基于时间的计划表进行，主要发布版本(现在称为功能发布版本)之间的预期时间只有 6 个月。因此，JDK 10 于 2018 年 3 月发布，也就是 JDK 9 发布 6 个月之后。这种更快的发布节奏使 Java 程序员能够快速获得新特性和改进。当一个新特性准备好时，它将成为下一个预定发行版的一部分，而不是等待两年或更长时间。

JDK 10 增加的主要新语言特性是支持本地变量类型推断。有了局部变量类型推断，现在可以在初始化器的类型中推断局部变量的类型，而不是显式指定其类型。为了支持这个新功能，将上下文敏感的标识符 var 添加到 Java 中，作为保留类型名。类型推断可以简化代码，因为如果可以从初始化器中推断变量的类型，就不需要指定多余的变量类型。在难以识别类型或无法显式指定类型的情况下，它还可以简化声明。局部变量类型推断已经成为当代编程环境的一个常见部分。它包含在 Java 中，帮助 Java 跟上语言设计不断发展的趋势。除了其他一些更改外，JDK 10 还重新定义了 Java 版本字符串，更改了版本号的含义，以便更好地与新的基于时间的发布计划保持一致。

在撰写本书时，Java 的最新版本是 Java SE 11 (JDK 11)。它于 2018 年 9 月发布，比 JDK 10 晚了 6 个月。JDK 11 中主要的新语言特性是支持在 lambda 表达式中使用 var。此外，还向 Java 启动程序添加了另一种执行模式，使其能够直接执行简单的单文件程序。JDK 11 还删除了一些特性。也许最有趣的是取消对 applet 的支持，这是因为 applet 的历史意义。回顾一下，applet 最初是由 JDK 9 禁用的。随着 JDK 11 的发布，applet 支持已经被移除。JDK 11 还删除了对另一种与部署相关的技术 Java Web Start 的支持。JDK 11 中还有一个引人注目的删除：JavaFX；这个 GUI 框架不再是 JDK 的一部分，而是成为一个独立的开源项目。因为 JDK 已经删除了这些特性，所以本书不讨论它们。

关于 Java 演化的另一要点是：从 2006 年开始，Java 的开源过程就开始了。今天，JDK 的开源实现是可用的。开源进一步促进了 Java 开发的动态性。归根结底，Java 的创新是安全的。Java 仍然是编程界所期待的充满活力、灵活的语言。

本书中的内容已通过 JDK 11 更新。然而，如前所述，Java 编程的历史是以动态变化为标志的。随着对 Java 的学习不断深入，用户将希望查看后续 Java 发行版的每个新特性。简单地说：Java 的演化还在继续！

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 16 章，每一章讨论 Java 的一个方面。本书的特色就在于包含许多便于读者学习的特色内容。

- **关键技能与概念：**每一章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。
- **专家解答：**每一章中都穿插一些“专家解答”，以一问一答的形式介绍补充知识和要点。
- **编程练习：**每一章中都包含一两道编程练习，以帮助读者将学到的知识应用到实践中。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 本书不需要读者具有编程经验

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。如果读者有一些编程经验，在阅读本书时可以加快速度。但要记住，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

要编译和运行本书提供的所有程序，需要获得最新版本的 Java Development Kit (JDK)。在撰写本书时，最新版本为 JDK 11，这是 Java SE 11 使用的 JDK 版本。本书第 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网址 www.oraclepressbooks.com 下载，也可以扫本书封底二维码下载。

0.7 特别感谢

特别感谢本书的技术编辑 Danny Coward。Danny 编辑过我写的多本书籍，他的见解和建议总是很有价值，也很受赞赏。

0.8 进一步学习

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

Java: The Complete Reference

Herb Schildt's Java Programming Cookbook

The Art of Java

Swing: a Beginner's Guide

Introducing JavaFX 8 Programming

目 录

第1章 Java基础	1
1.1 Java的历史和基本原则	2
1.1.1 Java的起源	2
1.1.2 Java与C和C++的关系	3
1.1.3 Java对Internet的贡献	3
1.1.4 Java的魔法：字节码	4
1.1.5 超越applet	5
1.1.6 更快速的发布时间表	6
1.1.7 Java的主要术语	6
1.2 面向对象程序设计	6
1.2.1 封装	7
1.2.2 多态性	8
1.2.3 继承	8
1.3 Java开发工具包	8
1.4 第一个简单的程序	9
1.4.1 输入程序	9
1.4.2 编译程序	10
1.4.3 逐行分析第一个程序	10
1.5 处理语法错误	12
1.6 第二个简单程序	12
1.7 另一种数据类型	14
1.8 两个控制语句	16
1.8.1 if语句	16
1.8.2 for循环语句	18
1.9 创建代码块	19
1.10 分号和定位	20
1.11 缩进原则	20
1.12 Java关键字	22
1.13 Java标识符	23
1.14 Java类库	23
1.15 自测题	23
第2章 数据类型与运算符	25
2.1 数据类型为什么重要	26
2.2 Java的基本类型	26
2.2.1 整数类型	26
2.2.2 浮点型	27
2.2.3 字符型	28
2.2.4 布尔类型	29
2.3 字面值	31
2.3.1 十六进制、八进制和二进制字面值	31
2.3.2 字符转义序列	32
2.3.3 字符串字面值	32
2.4 变量详解	33
2.4.1 初始化变量	33
2.4.2 动态初始化	33
2.5 变量的作用域和生命周期	34
2.6 运算符	36
2.7 算术运算符	36
2.8 关系运算符和逻辑运算符	37
2.9 短路逻辑运算符	39
2.10 赋值运算符	40
2.11 速记赋值	40
2.12 赋值中的类型转换	41
2.13 不兼容类型的强制转换	42
2.14 运算符的优先级	43
2.15 表达式	45
2.15.1 表达式中的类型转换	45
2.15.2 间距和圆括号	46
2.16 自测题	47
第3章 程序控制语句	49
3.1 从键盘输入字符	50
3.2 if语句	51
3.2.1 嵌套if语句	52
3.2.2 if-else-if阶梯状结构	53
3.3 switch语句	54
3.4 for循环	58

3.4.1 for 循环的一些变体	60	5.6.2 应用增强型 for 循环	123
3.4.2 缺失部分要素的 for 循环	61	5.7 字符串	123
3.4.3 无限循环	61	5.7.1 构造字符串	124
3.4.4 没有循环体的循环	62	5.7.2 操作字符串	124
3.4.5 在 for 循环内部声明循环控制变量	62	5.7.3 字符串数组	126
3.4.6 增强型 for 循环	63	5.7.4 字符串是不可变的	127
3.5 while 循环	63	5.7.5 使用 String 控制 switch 语句	128
3.6 do-while 循环	64	5.8 使用命令行实参	128
3.7 使用 break 语句退出循环	69	5.9 使用局部变量的类型推断功能	130
3.8 将 break 语句作为一种 goto 语句使用	70	5.9.1 引用类型的局部变量类型推断	131
3.9 使用 continue 语句	73	5.9.2 在 for 循环中使用局部变量类型推断	132
3.10 嵌套循环	77	5.9.3 var 的一些限制	133
3.11 自测题	78	5.10 位运算符	133
第 4 章 类、对象和方法	81	5.10.1 位运算符的与、或、异或和非	134
4.1 类的基础知识	82	5.10.2 移位运算符	137
4.1.1 类的基本形式	82	5.10.3 位运算符的赋值速记符	139
4.1.2 定义类	83	5.11 ?运算符	141
4.2 如何创建对象	85	5.12 自测题	143
4.3 引用变量和赋值	85	第 6 章 方法和类详解	145
4.4 方法	86	6.1 控制对类成员的访问	146
4.5 从方法返回值	88	6.2 向方法传递对象	150
4.6 返回值	89	6.3 返回对象	153
4.7 使用形参	90	6.4 方法重载	155
4.8 构造函数	98	6.5 重载构造函数	159
4.9 带形参的构造函数	99	6.6 递归	163
4.10 深入介绍 new 运算符	100	6.7 理解 static 关键字	165
4.11 垃圾回收	101	6.8 嵌套类和内部类	170
4.12 this 关键字	101	6.9 varargs	173
4.13 自测题	103	6.9.1 varargs 基础	173
第 5 章 其他数据类型与运算符	105	6.9.2 重载 varargs 方法	175
5.1 数组	106	6.9.3 varargs 和歧义	177
5.2 多维数组	110	6.10 自测题	178
5.2.1 二维数组	110	第 7 章 继承	179
5.2.2 不规则数组	111	7.1 继承的基础知识	180
5.2.3 三维或更多维的数组	112	7.2 成员访问与继承	182
5.2.4 初始化多维数组	112	7.3 构造函数和继承	184
5.3 另一种声明数组的语法	113	7.4 使用 super 调用超类构造函数	186
5.4 数组引用赋值	114	7.5 使用 super 访问超类成员	189
5.5 使用 length 成员	115	7.6 创建多级层次结构	192
5.6 for-each 形式的循环	119	7.7 何时调用构造函数	195
5.6.1 迭代多维数组	122	7.8 超类引用和子类对象	196

7.9 方法重写	200	9.7 抛出异常	249
7.10 重写的方法支持多态性	202	9.8 Throwable 详解	251
7.11 为何使用重写方法	203	9.9 使用 finally	252
7.12 使用抽象类	207	9.10 使用 throws 语句	254
7.13 使用 final	210	9.11 另外 3 种异常功能	255
7.13.1 使用 final 防止重写	210	9.12 Java 的内置异常	256
7.13.2 使用 final 防止继承	210	9.13 创建异常子类	258
7.13.3 对数据成员使用 final	211	9.14 自测题	262
7.14 Object 类	212	第 10 章 使用 I/O	265
7.15 自测题	213	10.1 基于流的 Java I/O	266
第 8 章 包和接口	215	10.2 字节流和字符流	266
8.1 包	216	10.3 字节流类	266
8.1.1 定义包	216	10.4 字符流类	267
8.1.2 寻找包和 CLASSPATH	217	10.5 预定义流	267
8.1.3 一个简短的包示例	217	10.6 使用字节流	268
8.2 包和成员访问	218	10.6.1 读取控制台输入	269
8.3 理解被保护的成员	220	10.6.2 写入控制台输出	269
8.4 导入包	222	10.7 使用字节流读写文件	270
8.5 Java 的类库位于包中	223	10.7.1 从文件输入	270
8.6 接口	223	10.7.2 写入文件	273
8.7 实现接口	224	10.8 自动关闭文件	275
8.8 使用接口引用	227	10.9 读写二进制数据	277
8.9 接口中的变量	233	10.10 随机访问文件	281
8.10 接口能够扩展	234	10.11 使用 Java 字符流	283
8.11 默认接口方法	235	10.11.1 使用字符流的控制台输入	284
8.11.1 默认方法的基础知识	235	10.11.2 使用字符流的控制台输出	286
8.11.2 默认方法的实际应用	236	10.12 使用字符流的文件 I/O	287
8.11.3 多继承问题	237	10.12.1 使用 FileWriter	287
8.12 在接口中使用静态方法	238	10.12.2 使用 FileReader	288
8.13 私有接口方法	239	10.13 使用 Java 的类型封装器转换数值	289
8.14 有关包和接口的最后思考	240	字符串	289
8.15 自测题	240	10.14 自测题	296
第 9 章 异常处理	241	第 11 章 多线程程序设计	299
9.1 异常的层次结构	242	11.1 多线程的基础知识	300
9.2 异常处理基础	242	11.2 Thread 类和 Runnable 接口	300
9.2.1 使用关键字 try 和 catch	242	11.3 创建一个线程	301
9.2.2 一个简单的异常示例	243	11.4 创建多个线程	309
9.3 未捕获异常的结果	245	11.5 确定线程何时结束	311
9.4 使用多个 catch 语句	247	11.6 线程的优先级	314
9.5 捕获子类异常	247	11.7 同步	316
9.6 try 代码块可以嵌套	248	11.8 使用同步方法	317

11.9 同步语句.....	319	13.14.3 泛型数组限制.....	382
11.10 使用 notify()、wait() 和 notifyAll() 的线程通信.....	321	13.14.4 泛型异常限制.....	383
11.11 线程的挂起、继续执行和停止.....	326	13.15 继续学习泛型.....	383
11.12 自测题.....	330	13.16 自测题.....	383
第 12 章 枚举、自动装箱、静态导入和注解	333	第 14 章 lambda 表达式和方法引用	385
12.1 枚举.....	334	14.1 lambda 表达式简介.....	386
12.2 Java 语言中的枚举是类类型.....	336	14.1.1 lambda 表达式的基础知识.....	386
12.3 values() 和 valueOf() 方法.....	336	14.1.2 函数式接口.....	387
12.4 构造函数、方法、实例变量和枚举.....	337	14.1.3 几个 lambda 表达式示例.....	389
12.5 枚举继承 enum.....	339	14.2 块 lambda 表达式.....	392
12.6 自动装箱.....	344	14.3 泛型函数式接口.....	393
12.7 类型封装器.....	344	14.4 lambda 表达式和变量捕获.....	398
12.8 自动装箱的基础知识.....	346	14.5 从 lambda 表达式中抛出异常.....	399
12.9 自动装箱和方法.....	347	14.6 方法引用.....	401
12.10 发生在表达式中的自动装箱/自动 拆箱.....	348	14.6.1 静态方法的方法引用.....	401
12.11 静态导入.....	349	14.6.2 实例方法的方法引用.....	402
12.12 注解(元数据).....	352	14.7 构造函数引用.....	406
12.13 自测题.....	354	14.8 预定义的函数式接口.....	408
第 13 章 泛型	355	14.9 自测题.....	409
13.1 泛型的基础知识.....	356	第 15 章 模块	411
13.2 简单的泛型示例.....	356	15.1 模块基础.....	412
13.2.1 泛型只能用于引用类型.....	359	15.1.1 简单的模块示例.....	413
13.2.2 泛型类型是否相同基于其类型实参.....	359	15.1.2 编译、运行第一个模块示例.....	416
13.2.3 带有两个类型形参的泛型类.....	360	15.1.3 requires 和 exports.....	417
13.2.4 泛型类的一般形式.....	361	15.2 java.base 和平台模块.....	417
13.3 受限类型.....	361	15.3 旧代码和未命名的模块.....	418
13.4 使用通配符实参.....	364	15.4 导出到特定的模块.....	419
13.5 受限通配符.....	366	15.5 使用 requires transitive	420
13.6 泛型方法.....	369	15.6 使用服务.....	423
13.7 泛型构造函数.....	370	15.6.1 服务和服务提供程序的基础知识	423
13.8 泛型接口.....	371	15.6.2 基于服务的关键字	424
13.9 原类型和遗留代码.....	377	15.6.3 基于模块的服务示例	424
13.10 使用菱形运算符进行类型推断.....	379	15.7 其他模块功能.....	430
13.11 局部变量类型推断和泛型	380	15.7.1 open 模块	430
13.12 擦除特性.....	380	15.7.2 opens 语句	430
13.13 歧义错误.....	380	15.7.3 requires static	430
13.14 一些泛型限制.....	381	15.8 继续模块的学习	431
13.14.1 类型形参不能实例化.....	381	15.9 自测题.....	431
13.14.2 对静态成员的限制.....	381	第 16 章 Swing 介绍	433
		16.1 Swing 的起源和设计原则	434

16.2 组件和容器.....	435	16.9 使用 JList.....	450
16.2.1 组件.....	435	16.10 使用匿名内部类或 lambda 表达式 来处理事件.....	458
16.2.2 容器.....	436	16.11 自测题.....	459
16.2.3 顶级容器窗格.....	436		
16.3 布局管理器.....	436	附录 A 自测题答案.....	461
16.4 第一个简单的 Swing 程序.....	437	附录 B 使用 Java 的文档注释.....	495
16.5 Swing 事件处理.....	440	附录 C 编译运行简单的单文件程序.....	503
16.5.1 事件.....	441	附录 D JShell 简介.....	505
16.5.2 事件源.....	441		
16.5.3 事件监听器.....	441		
16.5.4 事件类和监听器接口.....	441		
16.6 使用 JButton.....	442	附录 E 更多 Java 关键字.....	513
16.7 使用 JTextField.....	445		
16.8 使用 JCheckBox.....	448		



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

在计算领域，很少有技术具有 Java 这样的影响力。它在 Web 的早期创立，帮助构造了 Internet 的现代形式，包括客户端和服务器端。它的创新功能改进了编程的艺术和科学，在计算机语言设计方面设立了新标准。围绕着 Java 的前向思考文化确保它一直充满活力。在计算界经常进行快速、频繁的变化，简言之，Java 不仅是世界上最重要的计算机语言，也是改革编程的一种力量，在此过程中也改变了世界。

尽管 Java 是经常与 Internet 编程相关的语言，但它并不限于此。Java 是一门强大、功能全面、通用的编程语言。因此对于编程新手，Java 是一种绝佳的学习语言。今天，要成为职业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，你将学习必备的 Java 技能。

本章旨在介绍 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。为了帮助读者克服这一困难，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简单的运算符。对于这些内容我们并不进行深入讨论，只是关注一下 Java 程序共有的一些概念。

1.1 Java 的历史和基本原则

在充分理解 Java 的独特之处之前，有必要了解驱动其创建的动力、它所体现的编程哲学以及设计的关键概念。随着阅读本书的深入，你会发现 Java 的许多方面都是历史因素的直接或间接结果，这些因素塑造了这种语言。因此，要研究 Java，应该首先探索 Java 与更大的编程领域之间的关系。

1.1.1 Java 的起源

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旨在编译到机器码中，用于特定类型的 CPU，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、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。

曾经恼人却没那么重要的问题也就成为亟待解决的问题。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 Internet 的代码中。了解到这一点以后，Java 的重点从消费类电子产品转移到 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点燃了星星之火，然而 Internet 最终促成了 Java 的燎原之势。

1.1.2 Java 与 C 和 C++ 的关系

计算机语言的历史并不是一个孤立的事件，而是每种新语言都以这种或那种方式受以前语言的影响。在这方面，Java 也不例外。在继续之前，需要理解 Java 处于计算机语言家谱树的那个位置。

C 和 C++ 是与 Java 最接近的上一代语言。我们知道，C 和 C++ 是有史以来最重要的计算机语言，目前仍得到广泛使用。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，创建 Java 时，许多程序员都熟悉 C/C++ 语法。而 Java 使用类似的语法，所以 C/C++ 程序员学习 Java 相对容易。于是现有的程序员能够随意使用 Java，从而编程社团就很容易接受 Java。

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、逻辑一致的程序设计环境，可以更好利用已有的成果，并且增加了在线环境需要的新功能，改进了程序设计艺术。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。

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。

最后一点：尽管 Java 与 C++ 相似，尤其是它们都支持面向对象程序设计，但 Java 绝不是“C++ 的 Internet 版”，因为 Java 在实际应用以及基本原理上与 C++ 存在显著的区别。Java 也不是 C++ 的增强版。例如，Java 不提供对 C++ 的向上或向下兼容。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++ 则用来解决另一个不同系列的问题。两者将在未来共存。

1.1.3 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 产生了深远影响。首先，Java 的创建在总体上简化了 Internet 编程，它就像一个催化剂，吸引了大批程序员关注 Web。其次，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。最后，Java 还解决了与 Internet 相关的棘手问题：可移植性和安全性。

1. Java 简化了基于 Web 的编程

Java 在许多方面都简化了基于 Web 的编程，其中最重要的是 Java 能创建可移植的跨平台程序。同样重要的是 Java 对联网的支持。Java 库的易用功能允许程序员方便地编写访问和使用 Internet 的程序。Java 提供的机制还可以轻松地让程序传输到 Internet 上。尽管其细节超出了本书的范围，但应知道 Java 对联网的支持是它快速提升的一个关键因素。

2. Java applet

Java 问世时，它最重要的一个功能是 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。如果用户单击一个包含 applet 的链接，applet 就会在浏览器中自动下载并运行。它们通常都是小程序，用来显示服务器提供的数据，处理用户输入，或者提供简单功能（如贷款计算器）。applet 的关键功能是在本地执行，而不是在服务器上执行。实际上，applet 支持将一些功能从服务器转移到客户端。

applet 的产生很重要，因为它使对象可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数据。与此不同的是，applet 是动态的活动程序。这样的程序是客户端计算机上的活动代理，但由服务器初始化。

在 Java 的早期，applet 是 Java 编程的一个重要部分，它们体现了 Java 的强大和优势，给网页添加了令人激动

的功能，允许程序员探索 Java 的所有方面。尽管目前 applet 仍在使用，但随着时间的推移，它们将逐渐变得不那么重要，原因如前所述。从 JDK 9 开始，applet 被设置为过时，最终 JDK 11 会终止对 applet 的支持。

专家解答

问：什么是 C#，Java 与 C#的关系如何？

答：在 Java 问世以后没几年，Microsoft 开发出 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 的知识也会对你有所帮助。

3. 安全性

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。

用户可能意识到，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(稍后会讨论这是如何实现的)。能够在下载 applet 时确信对客户端计算机无害，是 Java 早期成功的主要因素。

4. 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在不同系统中执行。换言之，需要一种机制，使各种与 Internet 连接的不同的 CPU、操作系统和浏览器能够下载和执行同一个应用程序。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计算机上工作，因此需要某种机制来生成可移植的代码。稍后会提到，确保安全性的机制也有助于确保创建可移植的代码。

1.1.4 Java 的魔法：字节码

Java 能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)。字节码是一系列高度优化的指令，由名为 Java 虚拟机(Java Virtual Machine, JVM)的 Java 运行时系统执行。JVM 是 Java 运行时环境(Java Runtime Environment, JRE)的一部分。确切地讲，初始的 JRE 是一个字节码解释器。这可能让人吃惊，因为出于性能考虑，多数现代语言都编译为面向 CPU 的可执行代码。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 JRE(包括 Java 虚拟机)。一旦给定系统有了 JRE，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 JRE 不尽相同，但是它们都可以理解相同的 Java 字节码。如果把 Java 程序编译成本机代码，那么一个 Java 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 可以创建一种受限的执行环境，称为沙箱，它可以包含程序，防止对机器的不受限访问。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 执行程序的速度比想象的快许多。

尽管 Java 被设计为解释型语言，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，以提高性能。基于这一原因，Sun 在 Java 初始版本发布后不久就提供了 HotSpot JVM。HotSpot 提供了一个 JIT(Just In Time) 字节码编译器。在 JIT 成为 JVM 的一部分后，它可以根据逐条命令将选中的字节码部分实时转换为可执行代码。即在执行期间按照需要编译 JIT。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编译。其余代码会被简单地解释。尽管如此，JIT 方法也使性能有了显著提升。因为 JVM 依然控制着执行环境，所以甚至对字节码进行动态编译时，也可以保证可移植性与安全性。

另外，自从 JDK 9 以来，选中的 Java 环境也包含一个提前编译器，它可以先把字节码编译为本机代码，再由 JVM 执行，而不是由 JVM 就地编译执行。提前编译是一个特定的功能，没有代替之前介绍的 Java 传统方法。因为提前编译有高度专业化的特性，学习 Java 时不使用它，所以本书不进一步讨论它。

专家解答

问：我听说有一种特殊的 Java 程序叫作 servlet，它是什么？

答：servlet 是一种在服务器上执行的小程序。servlet 动态地扩展了 Web 服务器的功能。理解客户端应用程序的作用有助于理解 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。尽管进行 servlet 和服务器端编程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。

1.1.5 超越 applet

撰写本书时，距离 Java 初次发布已经二十多年了。这些年来，发生了许多变化。在 Java 诞生的时候，互联网是一个令人兴奋的创新；Web 浏览器正在迅速发展、完善；智能手机的现代形式还没有问世；计算机的普及还需要几年的时间。当然，Java 也在变革，其使用方式不断变化。也许没有什么比 applet 更好地说明了 Java 正在进行的演化。

如前所述，在 Java 的早期，applet 是 Java 编程的一个重要部分。它们不仅给网页添加了有趣内容，还是 Java 一个高度引人注目的部分，增加了 Java 的魅力。但是，applet 依赖于 Java 浏览器插件。因此 applet 要工作，浏览器必须支持它。最近，对 Java 浏览器插件的支持逐渐减少。简言之，没有浏览器支持，applet 就没有那么重要了。因此，从 JDK 9 开始，Java 对 applet 的支持被废弃。在 Java 语言中，被废弃意味着该功能仍可用，但标记为过时。被废弃的功能不应用于新代码。随着 JDK 11 的发布，由于对 applet 的支持被删除，applet 被逐步淘汰了。

值得注意的是，在 Java 问世几年后添加了 applet 的替代方案，称为 Java Web Start，支持从 Web 页面上动态下载应用程序。它是一种部署机制，对于不适合 applet 的大型 Java 应用程序尤其有用。applet 和 Java Web Start 应用程序的区别在于，Java Web Start 应用程序是独立运行的，而不是在浏览器中运行。因此，它很像一个“正常”的应用程序。但是，它要求主机系统上有一个支持 Java Web Start 的独立 JRE。从 JDK 11 开始，删除了对 Java Web Start 的支持。

考虑到 applet 和 Java Web Start 都不是现代版本 Java 的可行选项，那么应该使用什么机制来部署 Java 应用程序？在撰写本书时，部分答案是使用 JDK 9 添加的 jlink 工具。它可以创建一个完整的运行时映像，其中包括对程序的所有必要支持，包括 JRE。显然，部署是一个相当高级的主题，超出了本书的范围。幸运的是，使用本书不

需要担心部署问题，因为所有示例程序都直接运行在计算机上。它们不是通过 Internet 部署的。

1.1.6 更快速的发布时间表

最近在 Java 中有另一个重大变化，但它不涉及对语言或运行时环境的更改，而与 Java 版本的发布计划有关。过去，主要的 Java 版本通常相隔两年或更长时间才发布。然而，在 JDK 9 发布后，发布主要 Java 版本之间的时间间隔已经缩短了。今天，预计主要版本将严格按照基于时间的计划表发布，主要版本之间的预期时间间隔只有 6 个月。

每个主要的发行版(现在称为功能发行版)都包含在发行时已经准备好的功能。这种加快的发布节奏使 Java 程序员能够及时获得新的特性和增强。此外，它允许 Java 快速响应不断变化的编程环境的需求。简单地说，更快的发布计划对于 Java 程序员来说是一个非常积极的开发策略。

目前，功能发行版计划在每年的 3 月和 9 月发布。因此，JDK 10 于 2018 年 3 月发布，也就是 JDK 9 发布 6 个月之后。下一个版本(JDK 11)是在 2018 年 9 月发布的。同样，预计每 6 个月发布一个新版本。查阅 Java 文档，可以获得最新的发布进度信息。

在撰写本书时，已经出现了许多新的 Java 特性。由于更快的发布进度，很有可能在未来几年内将其中几个添加到 Java 中。请详细检查每 6 个月发布的信息和发布说明。对于 Java 程序员来说，这真令人激动！

1.1.7 Java 的主要术语

不了解 Java 的术语就无法对 Java 进行完整概述。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

1.2 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。本书后面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工具来处理更复杂的程序。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，但很难用它设计出清晰、简洁易懂的程序。

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，具有丰富的控制结构且不使用 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，许多项目已经接近甚至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。

面向对象程序设计吸收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。结果是产生一种新的程序组织方法。广义上讲，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)，另一种是围绕数据(谁受到影响)。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，数据类型精确地定义了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)。下面对此一一学习。

1.2.1 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，代码和数据可以通过创建自包含的黑盒(black box)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访问。当代码或数据是公有时，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。

1.2.2 多态性

多态性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)，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，就能使用所有的堆栈。

更普遍的是，多态性的概念常被表述为“单接口，多方法”。这就意味着可为一组相关的活动设计一个泛型接口。多态性允许使用同一接口指定一类动作，降低了程序的复杂度。编译器的工作就是选择适用于各种情况的特定动作(也就是方法)。程序员则无须手动进行这样的选择，只需要记住并利用这个统一的接口。

1.2.3 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)方式来管理的。例如，美味的红色苹果是苹果类的一部分，而苹果又是水果类的一部分，水果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)。而味美的红色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，对象就不得不明确定义出自己的所有特性。如果使用继承，那么对象只需要定义使自己在类中与众不同的属性，至于基本属性，可以从自己的父类继承。因此，正是继承机制使对象能够成为更一般的类的特定实例。

1.3 Java 开发工具包

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)。在撰写本书时，JDK 的当前版本是 JDK 11。这是 Java SE 11 的版本(SE 代表标准版)，也是本书所描述的版本。由于 JDK 11 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 11 或更高版本(请记住，由于 Java 加快了其发布进度，JDK 特性的发布预计间隔 6 个月。因此，不要对 JDK 的版本号更高感到惊讶)。但是，根据工作的环境，可能安装了较早的 JDK。如果是这种情况，包含新特性的程序就不能通过编译。

如果需要在计算机上安装 JDK，请注意，对于现代版本的 Java，可以下载 Oracle JDK 和开源 OpenJDK。通常，应该首先找到要使用的 JDK。例如，在撰写本书时，Oracle JDK 可以从 www.oracle.com/technetwork/java/javase/downloads/index.html 下载。同时，jdk.java.net 也提供了一个开源版本。接下来，下载选中的 JDK，按照说明将其安装到计算机上。安装 JDK 之后，就可以编译和运行程序了。

JDK 提供了两个主要程序。第一个是 Java 的编译器 javac。第二个是标准 Java 解释器 java，也称为应用程序启动器。另外，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, IDE)。

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 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。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，但是它的使用范围很广泛。对于小型程序而言，它们的面向对象性几乎是察觉不到的。而当程序增长时，就会轻松地用到更多面向对象特性。

1.4 第一个简单的程序

首先编译并运行下面这个简单程序：

```
/*
 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.");
    }
}
```

遵循以下三个步骤：

- (1) 输入程序。
- (2) 编译程序。
- (3) 运行程序。

1.4.1 输入程序

本书的所有程序都可从 www.oraclepressbooks.com 获得。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 或自己喜欢的其他任何程序编辑器。

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。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 作为文件扩展名。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，所有代码都必须驻留于一个类中。根据规则，主类名应该与存储程序的文件名相符，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程序的维护与组织更为轻松。而且，如本书后面所述，在某些情况下，必须这么做。

1.4.2 编译程序

为了编译 Example 程序，执行编译器 javac，这需要在命令行中指定源文件的名称，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，将每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，就会执行包含在特定类中的代码。

在继续之前，有一点很重要，即从 JDK 11 开始，Java 提供了一种方法，可以直接从源文件中运行某些类型的简单程序，而不必显式地调用 javac。附录 C 中描述了这种技术，它在某些情况下是有用的。出于本书的目的，假定使用的就是刚才描述的正常编译过程。

注意：

假设正确安装了 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 11，那么命令行工具的路径将是 C:\Program Files\Java\jdk-11\bin(当然，读者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。

1.4.3 逐行分析第一个程序

尽管程序 Example.java 非常短，却包含了所有 Java 程序共有的几个特点。下面仔细研究程序的各个部分。程序以下面几行开头：

```
/*
 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 是类名。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 程序都在某种程度上具有面向对象特点的原因之一。

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 “//” 开头，到行尾结束。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述，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。对于本行，各部分的意思现在不能一一详述，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。当类成员的前面有 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。如后面所述，方法也可以返回值。如果这些看起来令人费解，不必担心，后面各章将详细讨论所有这些概念。

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由方法名后面一对圆括号中指定的变量接收。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。如后面所述，`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。

容易猜出，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。在本书后面，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 中的许多语句都以分号结束。如后面所述，分号是 Java 语法中的一个重要部分。

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 区分大小写。忘记这一点会有很大麻烦。例如，如果不小心将 `main` 输成了 `Main`，或将 `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，却无法执行它们。因此，如果输错了 `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。

1.5 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。从以前的程序设计经验你了解到，向计算机输入代码时很容易输入一些不正确的內容。幸运的是，如果向程序输入了不正确的內容，那么编译器会在编译时报语法错误消息。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原因。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，并非一定要从字面上理解编译器提供的消息，因为这些消息可能有误导作用。需要推测错误消息以求找出问题的真正根源。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在真正发生错误位置的后面几行。

1.6 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 myVar1; // this declares a variable ← 声明变量
        int myVar2; // this declares another variable

        myVar1 = 1024; // this assigns 1024 to myVar1 ← 为变量赋值

        System.out.println("myVar1 contains " + myVar1);

        myVar2 = myVar1 / 2;

        System.out.print("myVar2 contains myVar1 / 2: ");
        System.out.println(myVar2);
    }
}

```

运行程序时，输出如下所示：

```

myVar1 contains 1024
myVar2 contains myVar1 / 2: 512

```

这个程序引入了几个新概念。第一个是声明整型变量 myVar1 的语句：

```
int myVar1; // this declares a variable
```

在 Java 中，所有变量都必须在使用前被声明，而且必须指定变量存储的值的类型，这称为变量的类型。本例中，myVar1 可存储整型值。在 Java 中，为声明整型变量，应该在变量名前添加关键字 int。因此，上面的语句声明了一个名为 myVar1 的 int 类型的变量。

下面一行声明的是第二个变量 myVar2：

```
int myVar2; // this declares another variable
```

注意，除了变量名不同以外，本行使用的格式与第一行一样。

一般来说，声明变量的语句格式如下：

```
type var-name;
```

这里，type 指定的是要声明的变量的类型，var-name 是变量名。除了 int，Java 还支持其他几种数据类型。

下面一行代码将值 1024 赋给 myVar1：

```
myVar1 = 1024; // this assigns 1024 to var1
```

在 Java 中，赋值运算符是一个等号，它将右侧的值复制到左侧的变量中。

下一行代码输出 myVar1 的值，并且前面加有字符串“myVar1 contains”：

```
System.out.println("myVar1 contains " + myVar1);
```

在这条语句中，加号会使 myVar1 的值紧跟字符串显示。这种方法可以被推广。使用“+”运算符，可在一条 println() 语句中将任意多个项链接在一起。

下面一行代码将 myVar1 的值除以 2 后赋给 myVar2：

```
myVar2 = myVar1 / 2;
```

本行将 myVar1 的值除以 2，然后存储到 myVar2 中。因此，在本行执行完毕后，myVar2 的值为 512。myVar1 的值则不发生变化。像其他许多计算机语言一样，Java 支持所有算术运算符，包括如表 1-2 所示的运算符在内。

表 1-2 Java 支持的算术运算符

运 算 符	说 明	运 算 符	说 明
+	加(Addition)	*	乘(Multiplication)
-	减(Subtraction)	/	除(Division)

以下是程序中接下来的两行：

```
System.out.print("myVar2 contains myVar1 / 2: ");
System.out.println(myVar2);
```

这里出现了两个新内容。首先是用于显示字符串"myVar2 contains myVar1 / 2:"的内置方法 print()。该字符串后面不再另起一行。这意味着当生成下一个输出时，它将出现在同一行中。除了在每次被调用后不再输出新行以外，print()方法与 println()十分相似。其次，注意在对 println()的调用中，使用的是 myVar2 变量本身。print()和 println()可用于输出任何 Java 内置类型的值。

在继续介绍之前，关于变量声明还有一点需要说明：只要用逗号将变量名分隔开，一条声明语句就可以声明两个或更多个变量。例如，myVar1 和 myVar2 可以这样声明：

```
int myVar1, myVar2; // both declared using one statement
```

1.7 另一种数据类型

前面的程序中使用了 int 类型的变量。然而，int 类型的变量只能存储整数。因此，当有小数出现时，就不能再使用该类型。例如，int 变量可以存储 18 这样的值，而不能存储 18.3 这样的值。幸好，除了 int，Java 还支持其他一些数据类型。为使用带有小数部分的数值，Java 定义了两种浮点类型：float 和 double，分别表示单精度值和双精度值。其中，double 是最常用的类型。

声明 double 类型的变量需要使用下面的语句：

```
double x;
```

这里，x 是变量名，类型为 double。因为 x 是浮点类型，所以它可以存储诸如 122.23、0.034 或 -19.0 这样的值。

为更好地理解 int 与 double 的区别，请看下面这个程序：

```
/*
 This program illustrates the differences
 between int and double.

 Call this file Example3.java.
 */
class Example3 {
    public static void main(String args[]) {
        int v; // this declares an int variable
        double x; // this declares a floating-point variable

        v = 10; // assign v the value 10
```

```

x = 10.0; // assign x the value 10.0

System.out.println("Original value of v: " + v);
System.out.println("Original value of x: " + x);
System.out.println(); // print a blank line ← 输出一个空行

// now, divide both by 4
v = v / 4;
x = x / 4;

System.out.println("v after division: " + v);
System.out.println("x after division: " + x);
}
}

```

程序的输出如下所示：

```

Original value of v: 10
Original value of x: 10.0

v after division: 2 ← 小数部分丢掉
x after division: 2.5 ← 小数部分保留

```

可以看出，当 v 除以 4 时，执行的是整除操作，输出为 2，丢掉了小数部分。但当 double 变量 x 除以 4 时，小数部分被保留下来，并且显示出了正确的值。

程序中还有一个新的地方需要注意，即为了输出一个空行，只需要调用一个没有实参的 println() 方法即可。

专家解答

问：为什么 Java 对于整数和浮点值有不同的数据类型？也就是说，为什么不对所有的数值使用同样的类型？

答：为编写出高效的程序，Java 提供了不同的数据类型。首先，整型运算比浮点型运算快。因此，如果不需要小数值，就不必使用 float 或 double 类型，以减少开销。其次，一种数据类型所需的内存空间可能比另一种要少；通过支持不同的类型，Java 可以更好地利用系统资源。最后，一些运算需要(至少可以得益于)使用特定的数据类型。总之，Java 支持的内置类型提供了最大的灵活性。

练习 1-1(GalToLit.java) 将加仑换算为升

尽管前面的几个程序说明了 Java 语言的几个重要特性，但是它们的用处并不是很大。即使现在你对 Java 了解不多，也依然可以学着创建一个实用程序。在本练习中，我们将创建一个将加仑(gallon)转换为升(liter)的程序。

该程序先声明两个 double 变量，一个用于存储加仑数，另一个用于存储转换为升以后的数。1 加仑等于 3.7854 升。因此，为将加仑转换为升，应将加仑值乘以 3.7854。程序会显示加仑数和对应的升数。

步骤如下：

- (1) 创建一个名为 GalToLit.java 的新文件。
- (2) 将下面的程序输入文件中：

```

/*
Try This 1-1

```

```
This program converts gallons to liters.

Call this program GalToLit.java.

*/
class GalToLit {
    public static void main(String args[]) {
        double gallons; // holds the number of gallons
        double liters; // holds conversion to liters

        gallons = 10; // start with 10 gallons

        liters = gallons * 3.7854; // convert to liters

        System.out.println(gallons + " gallons is " + liters + " liters.");
    }
}
```

(3) 使用下面的命令行编译程序:

```
javac GalToLit.java
```

(4) 使用下面的命令行运行程序:

```
java GalToLit
```

输出如下所示:

```
10.0 gallons is 37.854 liters.
```

(5) 如上所示，该程序将 10 加仑转换为升。通过赋予 `gallons` 不同的值，可以让程序将不同数量的加仑数转换为对应的升数。

1.8 两个控制语句

在方法内部，语句从上至下依次执行。然而，通过使用 Java 支持的不同控制语句可改变这一流程。虽然后面会详细介绍控制语句，但这里需要首先简要介绍一下两条控制语句，因为在编写示例程序时会用到它们。

1.8.1 if 语句

使用 Java 的条件语句 `if`，可以有选择地执行程序的某一部分。Java 的 `if` 语句与其他语言中的 `if` 语句非常相似。它根据某个条件是真或假来确定执行哪个程序流。`if` 语句的最简单形式如下：

```
if(condition) statement;
```

此处，`condition` 是一个 Boolean 表达式(计算结果是真或假的表达式)。如果 `condition` 为真，则执行语句。如果 `condition` 为假，则跳过语句。下面是一个示例：

```
if(10 < 11) System.out.println("10 is less than 11");
```

本例中，因为 10 比 11 小，所以条件表达式为真，执行 `println()`。考虑下面的语句：

```
if(10 < 9) System.out.println("this won't be displayed");
```

本例中，10 比 9 大，因此不会调用 `println()`。

Java 定义了可在条件表达式中使用的完整关系运算符，如表 1-3 所示。

表 1-3 Java 定义的在条件表达式中使用的关系运算符

运 算 符	含 义	运 算 符	含 义
<	小于	>=	大于或等于
<=	小于或等于	==	等于
>	大于	!=	不等于

注意等于是两个等号。

下面给出了一个演示 if 语句的程序：

```

/*
Demonstrate the if.

Call this file IfDemo.java.
*/
class IfDemo {
    public static void main(String args[]) {
        int a, b, c;

        a = 2;
        b = 3;

        if(a < b) System.out.println("a is less than b");
        // this won't display anything
        if(a == b) System.out.println("you won't see this");

        System.out.println();

        c = a - b; // c contains -1

        System.out.println("c contains -1");
        if(c >= 0) System.out.println("c is non-negative");
        if(c < 0) System.out.println("c is negative");

        System.out.println();

        c = b - a; // c now contains 1

        System.out.println("c contains 1");
        if(c >= 0) System.out.println("c is non-negative");
        if(c < 0) System.out.println("c is negative");

    }
}

```

程序的输出如下所示：

```

a is less than b

c contains -1
c is negative

```

```
c contains 1
c is non-negative
```

程序中还有一点要注意，下面这一行代码：

```
int a, b, c;
```

通过使用逗号分隔的列表声明了 3 个变量 a、b 和 c。如前所述，当需要两个或多个相同类型的变量时，只要将变量名用逗号分隔开，它们就可在一条语句中声明。

1.8.2 for 循环语句

通过创建循环(loop)，可以重复执行一段代码。只要需要执行重复的任务，就使用循环，因为它们比一遍遍地编写相同的语句更简单。Java 支持各种功能强大的循环结构。这里介绍的是 for 循环。下面是最简单形式的 for 循环：

```
for(initialization; condition; iteration) statement;
```

在 for 循环最常用的形式中，循环的 initialization(初始化)部分设定了一个循环控制变量的初始值。condition(条件)是测试循环控制变量的 Boolean 表达式。如果测试的结果是真，就执行 statement，for 循环将继续；如果为假，循环就要终止。iteration 表达式用于决定循环的每一次迭代完成之后控制变量如何变化。下面的程序演示了 for 循环的用法：

```
/*
Demonstrate the for loop.

Call this file ForDemo.java.
*/
class ForDemo {
    public static void main(String args[]) {
        int count;

        for(count = 0; count < 5; count = count+1) ←———— 循环迭代 5 次
            System.out.println("This is count: " + count);

        System.out.println("Done!");
    }
}
```

程序的输出如下所示：

```
This is count: 0
This is count: 1
This is count: 2
This is count: 3
This is count: 4
Done!
```

本例中，count 是循环控制变量，它在 for 循环的初始化部分被设为 0。在各次循环(包括第一次)开始时，执行条件测试 $count < 5$ 。如果测试的结果为真，那么执行 `println()` 语句，接着执行循环的迭代部分，将 count 递增 1。这一过程一直持续进行，直到条件测试为假，然后执行循环体之后的语句。有趣的一点是，在专业编写的 Java 程序中，几乎没有像上面那样编写循环的迭代部分，即很少看到下面所示的语句：

```
count = count + 1;
```

因为 Java 有一个可以更有效地执行这一操作的递增运算符。这个递增运算符就是++(即并列的两个加号)。递增运算符每次将操作数加 1。通过使用递增运算符，前面的语句可以这样写：

```
count++;
```

因此，上面程序中的 for 语句经常写为：

```
for(count = 0; count < 5; count++)
```

测试一下这个循环。正如你将看到的，循环的运行结果与原来是一样的。

Java 还提供了一个递减运算符--。该运算符使操作数减 1。

1.9 创建代码块

Java 的另一个关键元素是代码块(code block)。代码块是两条或多条语句的组合。这是通过将语句包含在左右花括号之间来实现的。代码块一旦创建，它就成为一个逻辑单元，凡是可以说使用单条语句的地方，就可以使用它。例如，代码块可以作为 Java 的 if 和 for 语句执行的目标代码。考虑下面的 if 语句：

```
if(w < h) { ← 代码块开始
    v = w * h;
    w = 0;
} ← 代码块结束
```

此外，如果 w 小于 h，就会执行代码块里的两条语句。因此，代码块中的两条语句就形成了一个逻辑单元，如果一条语句不能执行，那么另一条语句也无法执行。这里的要点就是，只需要将两条或多条语句在逻辑上链接在一起，就可以通过创建一个代码块来实现。代码块使许多算法的实现更清楚、更高效。

下面的程序使用代码块来防止除 0 错误：

```
/*
Demonstrate a block of code.

Call this file BlockDemo.java.
*/
class BlockDemo {
    public static void main(String args[]) {
        double i, j, d;

        i = 5;
        j = 10;

        // the target of this if is a block
        if(i != 0) {
            System.out.println("i does not equal zero");
            d = j / i;
            System.out.println("j / i is " + d);
        }
    }
}
```

程序的输出如下所示：

```
i does not equal zero
j / i is 2.0
```

本例中 if 语句的目标代码是一个代码块，而不是一条语句。如果 if 控制条件为真(正如本例所示)，将会执行代码块中的三条语句。尝试将 i 设为 0，观察结果。你将发现整个代码块会被跳过。

如本书后面所述，代码块还有其他特性和用法。然而，它们存在的主要原因还是创建逻辑相关的代码单元。

专家解答

问：代码块的使用会造成运行时效率降低吗？换句话说，Java 会执行 “{” 和 “}” 吗？

答：不会。代码块并不会增加任何开销。事实上，由于它们能够简化某些算法的编码，因此它们一般会加快速度，提高效率。另外，“{” 和 “}” 只存在于程序的源代码中，Java 根本不会执行 “{” 和 “}”。

1.10 分号和定位

Java 中，分号是一条语句的终止符，即每条语句都必须以分号结尾。它表明一个逻辑实体的结束。

如你所知，代码块是一组逻辑相关的语句，包含在左右花括号之间。代码块不以分号结束，而以代码块末尾的右花括号表示结束。

Java 不把行末作为结束符。出于这一原因，在某一行的哪个位置输入语句就无关紧要了。例如：

```
x = y;
y = y + 1;
System.out.println(x + " " + y);
```

对于 Java 而言，这几行代码与下面的代码是等效的：

```
x = y; y = y + 1; System.out.println(x + " " + y);
```

而且，一条语句的单个元素也可放在不同的行中。例如，下面的代码也是完全正确的：

```
System.out.println("This is a long line of output" +
    x + y + z +
    "more output");
```

将一个较长的行如此分隔，常用于增强程序的可读性，也有助于防止一行过长而发生换行。

1.11 缩进原则

注意在前面的示例中，对某些语句使用了缩进。Java 是一种形式自由的语言，因此在一行放置语句时，语句之间的相对位置无关紧要。但是，长期以来，已经形成了一种公认的、被人们接受的缩进形式。这种形式增强了程序的可读性。本书就遵循这种形式，并推荐你也这样做。使用这种形式时，应该在每个左花括号之后缩进一级，而在每个右花括号之后提前一级。对于个别语句还提倡其他缩进方式，这些内容将在后面介绍。

练习 1-2(GalToLitTable.java)

改进从加仑到升的转换程序

现在可以使用 for 循环、if 语句和代码块来改进在练习 1-1 中开发的从加仑到升的转换程序。新程序将打印 1 加仑到 100 加仑的转换表。每隔 10 加仑输出一个空行。为此使用变量 counter，用于统计输出行数。请特别留意它。

步骤:

(1) 创建名为 GalToLitTable.java 的新文件。

(2) 将下列程序输入该文件中:

```
/*
 Try This 1-2

 This program displays a conversion
 table of gallons to liters.

 Call this program "GalToLitTable.java".
*/
class GalToLitTable {
    public static void main(String args[]) {
        double gallons, liters;
        int counter;

        counter = 0; ← 将行计数器初始化为0
        for(gallons = 1; gallons <= 100; gallons++) {
            liters = gallons * 3.7854; // convert to liters
            System.out.println(gallons + " gallons is " +
                liters + " liters.");

            counter++; ← 每次循环迭代时递增行计数器
            // every 10th line, print a blank line
            if(counter == 10) { ← 如果计数器为 10, 输出一个空行
                System.out.println();
                counter = 0; // reset the line counter
            }
        }
    }
}
```

(3) 使用下面的命令行编译程序:

```
javac GalToLitTable.java
```

(4) 使用下面的命令行运行程序:

```
java GalToLitTable
```

下面是部分输出结果:

```
1.0 gallons is 3.7854 liters.
2.0 gallons is 7.5708 liters.
3.0 gallons is 11.356200000000001 liters.
4.0 gallons is 15.1416 liters.
5.0 gallons is 18.927 liters.
6.0 gallons is 22.712400000000002 liters.
7.0 gallons is 26.4978 liters.
8.0 gallons is 30.2832 liters.
9.0 gallons is 34.0686 liters.
10.0 gallons is 37.854 liters.

11.0 gallons is 41.6394 liters.
12.0 gallons is 45.424800000000005 liters.
```

```

13.0 gallons is 49.2102 liters.
14.0 gallons is 52.9956 liters.
15.0 gallons is 56.781 liters.
16.0 gallons is 60.5664 liters.
17.0 gallons is 64.3518 liters.
18.0 gallons is 68.1372 liters.
19.0 gallons is 71.9226 liters.
20.0 gallons is 75.708 liters.

21.0 gallons is 79.49340000000001 liters.
22.0 gallons is 83.2788 liters.
23.0 gallons is 87.0642 liters.
24.0 gallons is 90.84960000000001 liters.
25.0 gallons is 94.635 liters.
26.0 gallons is 98.4204 liters.
27.0 gallons is 102.2058 liters.
28.0 gallons is 105.9912 liters.
29.0 gallons is 109.7766 liters.
30.0 gallons is 113.562 liters.

```

1.12 Java 关键字

Java 语言目前定义了 61 个关键字(参见表 1-4)。这些关键字与运算符和分隔符的语法结合起来就构成 Java 语言的定义。这些关键字一般不能作为变量名、类名或方法名使用。这条规则的例外是 JDK 9 为支持模块(详见第 15 章)而添加的与上下文相关的新关键字。另外,从 JDK 9 开始,下画线本身也是关键字,以防止把它用作程序中某个对象的名称。

关键字保留了 const 和 goto,但不能使用。早期 Java 中保留了几个关键字以备后用。但是目前的 Java 规范只定义了如表 1-4 所示的关键字。

表 1-4 Java 关键字

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	enum	exports	extends
final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long
module	native	new	open	opens	package
private	protected	provides	public	requires	return
short	static	strictfp	super	switch	synchronized
this	throw	throws	to	transient	transitive
try	use	void	volatile	while	with
-					

除了这 61 个关键字之外,Java 还保留了 true、false 和 null,它们是 Java 定义的值,也不能作为变量名、类名或方法名使用。从 JDK 10 开始,var 这个单词就添加为与上下文相关的保留类型名(参见第 5 章以了解 var 的详细信息)。

1.13 Java 标识符

在 Java 中，标识符是给方法、变量或其他用户定义项指定的名称。标识符可以包含一个到若干个字符。变量名可以字母表中的任何字母、下画线或美元符号开头，后面可以是字母、数字、美元符号或下画线。下画线用于增加变量名的可读性，如 `line_count`。大写和小写是不同的，即对 Java 而言，`myVar` 和 `MyVar` 是两个名称。下面列出了几个合法的标识符：

Test	x	y2	MaxLoad
\$up	_top	my_myVar	sample23

切记不能以数字开头。因此，`12x` 就是无效的标识符。

不能使用任何 Java 关键字作为标识符，也不能将任何标准方法名用作标识符，例如 `println`。除了这两条限制以外，良好的编程习惯要求使用可以反映被命名项的含义或作用的标识符。

1.14 Java 类库

本章的示例程序用到了两个 Java 内置方法：`println()` 和 `print()`。这两个方法通过 `System.out` 访问。`System` 是一个自动引入程序中的 Java 预定义类。大体而言，Java 环境依赖于几个内置的类库，这些类库又包含许多为 I/O、字符串处理、网络和图形等提供支持的内置方法。标准类也提供对图形用户界面的支持。因此，Java 作为一个整体是 Java 语言本身及其标准类的结合。类库提供了 Java 所带的许多功能。的确，要成为 Java 程序员，需要学习使用标准 Java 类库。虽然本书通篇都会描述标准类库的不同元素和方法，但是对于 Java 类库，你还需要进一步学习。

1.15 自测题

1. 什么是字节码？它对 Java 的 Internet 程序设计为何十分重要？
2. 面向对象程序设计的三个主要原则是什么？
3. Java 程序从何处开始执行？
4. 什么是变量？
5. 下面哪几个变量名是无效的？
 - A. `count`
 - B. `$count`
 - C. `count27`
 - D. `67count`
6. 如何创建单行注释与多行注释？
7. 写出 `if` 语句和 `for` 循环的基本形式。
8. 如何创建代码块？
9. 月球重力为地球重力的 17%。编写一个程序来计算你在月球上的实际重力。
10. 改编练习 1-2，打印从英寸到米的转换表。转换 12 英尺，一英寸一英寸地转换；每 12 英寸输出一个空行（1 米约等于 39.37 英寸）。
11. 如果在输入程序时犯了输入错误，会导致什么类型的错误？
12. 语句在一行中的放置位置有限制吗？



第 2 章

数据类型与运算符

关键技能与概念

- 了解 Java 的基本类型
- 使用字面值
- 初始化变量
- 了解方法中变量的作用域原则
- 使用算术运算符
- 使用关系运算符和逻辑运算符
- 理解赋值运算符
- 使用赋值速记符
- 理解赋值中的类型转换
- 不兼容类型的强制转换
- 理解表达式中的类型转换