

ORACLE  
PRESS



Java核心技术系列

HZ BOOKS  
华章IT

涵盖Java 17 新特性

Core Java, Volume I : Fundamentals, Twelfth Edition

# Java

## 核心技術

卷I 开

(原书第12版)

[美] 凯·S·霍斯特曼 (Cay S. Horstmann) 著  
林琪 苏钰涵 译

ORACLE



机械工业出版社  
China Machine Press

# Java 核 心 技 术

卷I 开发基础 (原书第12版)

ORACLE  
PRESS

Core Java, Volume I : Fundamentals  
Twelfth Edition

本书经全面修订，以涵盖Java 17的新特性。新版延续之前版本的优良传统，利用清晰明了的示例加以解释，着力让读者在充分理解Java语言和Java类库的基础上，灵活应用Java提供的高级特性，具体包括面向对象程序设计、反射与代理、接口与内部类、异常处理、泛型程序设计、集合框架、事件监听器模型、图形用户界面设计和并发。整本书不仅可以让读者深入了解设计和实现Java应用程序涉及的所有基础知识和Java特性，还会帮助读者掌握开发Java程序所需的基本技能。

要想了解有关Java 17的高级特性，包括企业特性、模块系统、网络、安全和高级UI编程，请期待《Java核心技术 卷II 高级特性（原书第12版）》。

## 本书特点

- 除JDK和Java IDE外，详细介绍如何使用JShell工具快速、方便地尝试Java代码。
- 利用丰富示例全面介绍Java的基本语法。
- 详细解释Java语言封装机制，并提供面向对象程序设计(OOP)建议。
- 利用示例讲解Java继承的设计技巧。
- 通过清晰明了的介绍，帮助读者充分理解并有效使用相对复杂的反射。
- 利用接口和lambda表达式，帮助读者提升Java面向对象编程能力。
- 全面介绍Java异常处理，并提供实用的调试技巧。
- 重点强调泛型程序设计和强类型机制，避免不安全的强制类型转换。
- 帮助读者有效使用Java平台的集合框架和预建标准集合。
- 提供GUI程序设计和Swing GUI工具包使用指南，指导读者创建跨平台的图形用户界面。
- 全面介绍Java并发和多线程编程所需的工具。

P Pearson  
[www.pearson.com](http://www.pearson.com)



上架指导：计算机 / 程序设计

ISBN 978-7-111-70641-0



9 787111 706410

定价：149.00元

投稿热线：(010) 88379604  
读者信箱：[hzjsj@hzbook.com](mailto:hzjsj@hzbook.com)  
客服电话：(010) 88361066 88379833 68326294

华章网站：[www.hzbook.com](http://www.hzbook.com)  
网上购书：[www.china-pub.com](http://www.china-pub.com)  
数字阅读：[www.hzmedia.com.cn](http://www.hzmedia.com.cn)

Core Java, Volume I : Fundamentals, Twelfth Edition

# Java

## 核心 技术

### 卷I 开发基础 (原书第12版)

[美] 凯·S·霍斯特曼 (Cay S. Horstmann) 著  
林琪 苏钰涵 译



机械工业出版社  
China Machine Press

## 图书在版编目 (CIP) 数据

Java 核心技术：原书第 12 版·卷 I，开发基础 / (美) 凯·S·霍斯特曼 (Cay S. Horstmann) 著；林琪，苏钰涵译。-- 北京：机械工业出版社，2022.5  
(Java 核心技术系列)

书名原文：Core Java, Volume I: Fundamentals, 12e  
ISBN 978-7-111-70641-0

I. ① J… II. ① 凯… ② 林… ③ 苏… III. ① JAVA 语言 - 程序设计 IV. ① TP312.8

中国版本图书馆 CIP 数据核字 (2022) 第 068396 号

北京市版权局著作权合同登记 图字：01-2022-1597 号。

Authorized translation from the English language edition, entitled *Core Java, Volume I: Fundamentals, Twelfth Edition*, ISBN: 9780137673629, by Cay S. Horstmann, published by Pearson Education, Inc.. Copyright © 2022 Pearson Education Inc., Portions copyright. 1996-2013 Oracle and/or its affiliates.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Chinese simplified language edition published by China Machine Press, Copyright © 2022.

本书中文简体字版由 Pearson Education (培生教育出版集团) 授权机械工业出版社在中国大陆地区（不包括香港、澳门特别行政区及台湾地区）独家出版发行。未经出版者书面许可，不得以任何方式抄袭、复制或节录本书中的任何部分。

本书封底贴有 Pearson Education (培生教育出版集团) 激光防伪标签，无标签者不得销售。

## Java 核心技术 卷 I 开发基础 (原书第 12 版)

出版发行：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：王 颖 冯秀冰

责任校对：马荣敏

印 刷：北京诚信伟业印刷有限公司

版 次：2022 年 6 月第 1 版第 1 次印刷

开 本：186mm×240mm 1/16

印 张：43

书 号：ISBN 978-7-111-70641-0

定 价：149.00 元

客服电话：(010) 88361066 88379833 68326294

投稿热线：(010) 88379604

华章网站：[www.hzbook.com](http://www.hzbook.com)

读者信箱：[hzjsj@hzbook.com](mailto:hzjsj@hzbook.com)

版权所有·侵权必究

封底无防伪标均为盗版

## 译者序

书写 Java 传奇的 Sun 公司曾经堪称“日不落”帝国，但服务器市场的萎缩让这个声名赫赫的庞大帝国从蓬勃走向没落。在 2009 年被 Oracle 公司收购之后，Sun 公司逐渐淡出了人们的视线，而与此同时，我们也在很长一段时间内没能看到 Java 当初活跃的身影。

Java 就这样退出历史舞台了吗？当然不是！Sun 公司从 2006 年 12 月发布 Java 6 后，经过 5 年多的不懈努力，2011 年 7 月底发布了 Java 7 正式版。3 年后，被冠名为“跳票王”的 Oracle 公司终于发布了 Java 8 的正式版。又是 3 年后，Java 9 发布。从 2018 年开始，为了更快地引入新特性，每 6 个月就会发布一个 Java 版本，目前最新的长期支持版本是 Java 17。

值得一提的是，伴随着 Java 的成长，《Java 核心技术》也从第 1 版到第 11 版一路走来，得到了广大 Java 程序设计人员的青睐，成为一本畅销不衰的 Java 经典图书。2022 年，针对 Java 17，《Java 核心技术》第 12 版问世。这一版涵盖了 Java 17 的最新特性，相应调整了部分内容结构，同时延续之前版本的优良传统，利用清晰明了的示例加以解释，并提供了全部示例代码，以便读者学习和灵活应用。它将续写从前的辉煌，使人们能及时跟上 Java 前进的步伐。

本书由林琪、苏钰涵翻译。书中文字与内容力求忠实原书，不过由于译者水平有限，译文肯定有不当之处，敬请批评指正。

译者

2022 年 4 月于北京

# 前　　言

## 致读者

1995年年底，Java语言在Internet舞台一亮相便名声大噪。Java技术承诺成为连接用户与信息的万能胶，而不论这些信息来自Web服务器、数据库、信息提供商，还是任何其他可以想象的渠道。事实上，就兑现这个承诺而言，Java具有独特的优势和地位。它是一种完全可信赖的程序设计语言，并得到了广泛认可。其固有的可靠性与安全特性不仅令Java程序员放心，也令使用Java程序的用户放心。Java内建了对网络编程、数据库连接和并发等高级程序设计任务的支持。

1995年以来，已经发布了Java开发工具包（Java Development Kit）的12个主要版本，在过去的25年中，应用程序编程接口（API）也从200个类扩展到超过4000个类。现在这些API覆盖了用户界面构建、数据库管理、国际化、安全性以及XML处理等各个不同的领域。

你手上的这本书是《Java核心技术》第12版的卷I。《Java核心技术》的每个版本都紧随Java开发工具包的最新版本，并进行全面修订，以涵盖Java的最新特性。这一版经过更新，将反映Java 17的特性。

与本书以前的版本一样，这一版仍然将读者群定位为那些打算将Java应用到实际项目中的程序员。这里假设读者是具有程序设计语言（除Java之外）坚实背景知识的程序员，而且不希望书中充斥着玩具式的示例（诸如，烤面包机、动物园的动物或神经质的跳动文本）。这些绝对不会在这本书中出现。本书的目标是让读者充分理解Java语言及Java类库，而不是让读者产生误解。

本书提供大量示例代码来演示Java的几乎每一个语言特性和类库特性。这里有意使用简单的示例程序以突出重点，不过，大部分示例都不是虚构的，也没有偷工减料。在编写代码时，这些示例可以作为很好的起点。

我们假定读者愿意（甚至渴望）学习Java提供的所有高级特性。例如，本书将详细介绍以下内容：

- 面向对象程序设计
- 反射与代理
- 接口与内部类
- 异常处理
- 泛型程序设计
- 集合框架
- 事件监听器模型

- 图形用户界面设计
- 并发

随着 Java 类库的爆炸式增长，只用一卷无法涵盖程序员需要了解的所有 Java 特性。因此，我们决定将本书分为两卷。卷 I（本书）集中介绍 Java 语言的基本概念以及用户界面程序设计的基础知识。卷 II（高级特性）进一步介绍企业特性以及高级的用户界面程序设计，其中详细讨论以下内容：

- 流 API
- 文件处理与正则表达式
- 数据库
- XML 处理
- 注解
- 国际化
- 网络编程
- 高级 GUI 组件
- 高级图形
- 原生方法

写书时难免出现错误和不准确的地方。我们很想知道有哪些错误，不过，也希望同一个问题只被告知一次。我们在 <http://horstmann.com/corejava> 中给出了一个常见问题和 bug 修正列表。在勘误页的最后（建议先阅读勘误页）附有一个表单，可以用来报告 bug 和提出改进意见。如果我们没能回答每一个问题或者没有及时回复，请不要失望。我们确实会阅读所有电子邮件，而且非常感谢你的建议，这会使本书后续版本更清晰、更全面。

## 关于本书

第 1 章概述 Java 与其他程序设计语言不同的功能，解释这种语言的设计初衷，以及在哪些方面达到了预期。然后，简要叙述 Java 的历史，介绍 Java 是如何诞生和演进的。

第 2 章介绍如何下载和安装 JDK 以及本书的程序示例，然后指导读者编译和运行一个控制台应用和一个图形应用。你将了解如何使用 JDK、Java IDE 和 JShell 工具。

第 3 章开始讨论 Java 语言。这一章会介绍一些基础知识，包括变量、循环和简单的函数。对于 C 或 C++ 程序员来说，学习这一章的内容会感觉一帆风顺，因为这些语言特性的语法基本上与 C 语言相同。如果你没有 C 语言背景，但使用过其他程序设计语言（如 Visual Basic），可能需要仔细阅读这一章。

面向对象程序设计（Object-Oriented Programming, OOP）是当今程序设计的主流，而 Java 是一种面向对象程序设计语言。第 4 章将介绍面向对象两大基石中的第一个概念——封装，以及 Java 语言实现封装的机制，即类与方法。除了 Java 语言规则之外，还对如何实现完善的 OOP 给出了建议。最后，会介绍奇妙的 javadoc 工具，它能将代码注释转换为一组包含超链接的网页。熟悉 C++ 的程序员可以快速浏览这一章，而对于没有面向对象程序设计背

景的程序员，在进一步学习 Java 之前应当先花一些时间了解 OOP 的有关概念。

类和封装只是 OOP 的一部分，第 5 章将介绍另一部分——继承（inheritance）。继承允许利用现有的类，并根据需要进行修改。这是 Java 程序设计中的一个基础技术。Java 中的继承机制与 C++ 的继承机制十分相似。重申一次，C++ 程序员可以只关注这两种语言的不同之处。

第 6 章介绍如何使用 Java 的接口（interface）。接口允许你超越第 5 章中的简单继承模型。掌握接口会让你充分获得 Java 面向对象程序设计方法的强大能力。介绍接口之后，我们将转而介绍 lambda 表达式（lambda expression），这是一种简洁的表示方法，用来表示可以在以后某个时间点执行的代码块。接下来还会讲解 Java 的一个有用的技术特性——内部类（inner class）。

第 7 章讨论异常处理（exception handling），这是 Java 处理异常情况的一种健壮机制，用于处理正常程序可能出现意外的情况。异常提供了一种将正常处理代码与错误处理代码分开的有效手段。当然，即使通过处理所有异常条件来强化程序，程序仍然有可能不按预期的方式工作。这一章的最后一节将给出一组实用的调试技巧。

第 8 章概要介绍泛型程序设计。泛型程序设计可以让程序更可读、更安全。我们会展示如何使用强类型机制，而舍弃不好看也不安全的强制类型转换，以及如何处理与 Java 老版本兼容所带来的复杂问题。

第 9 章讨论的是 Java 平台的集合框架。如果希望收集多个对象，并在以后获取这些对象，就应当使用最适用的集合，而不是简单地把这些元素放在一个数组中。这一章会介绍如何充分利用预建的标准集合。

第 10 章介绍 GUI 程序设计，展示如何创建窗口、如何在窗口中绘图、如何利用几何图形绘图、如何采用多种字体格式化文本，以及如何显示图像。接下来介绍如何编写代码来响应事件，如鼠标单击事件或按键事件。

第 11 章详细讨论 Swing GUI 工具包。Swing 工具包允许你建立跨平台的图形用户界面。还将介绍各种按钮、文本组件、边框、滑动条、列表框、菜单以及对话框的有关内容。不过，一些更高级的组件会在卷 II 中讨论。

第 12 章是本书的最后一章，这一章将讨论并发，即编写并行执行的任务。当前，大多数处理器都有多个内核，而且我们希望这些内核都保持忙碌，所以并发是 Java 技术的一个重要且令人振奋的应用。

附录列出了 Java 语言的保留字。

## 约定

与很多计算机图书一样，本书使用等宽字体（monospace type）表示计算机代码。

 **注释：**“注释”信息会用这样的图标标志。

 **提示：**“提示”信息会用这样的图标标志。

 **警告：**对于可能出现的危险，我们用这样的图标做出警示。

 **C++ 注释：**在本书中有许多用来解释 Java 与 C++ 之间差别的 C++ 注释。没有 C++ 背景或者不擅长 C++ 编程并把它当作一场噩梦不愿再想起的程序员可以跳过这些注释。

Java 提供了一个庞大的程序设计库，即应用程序编程接口（API）。第一次使用 API 调用时，我们会在那一节末尾给出它的概要描述。这些描述不太正式，但我们希望它们能够比官方的联机 API 文档提供更多信息。类、接口或方法名后面的编号是引入该特性的 JDK 版本号，如下例所示：

 **应用程序编程接口 9**

本书英文版网站上提供了书中一些程序的源代码，这些程序在书中会以程序清单的形式给出，例如：

**程序清单 1-1 InputTest/InputTest.java**

**示例代码**

本书英文版网站 <http://horstmann.com/corejava> 提供了书中的所有示例代码。有关安装 Java 开发工具包和示例代码的详细信息请参看第 2 章。

# 致 谢

写一本书需要投入大量的精力，升级一本书也不像想象的那样轻松，尤其是 Java 技术一直在持续不断地更新。出版一本书会让很多人耗费大量心血，在此衷心地感谢《Java 核心技术》团队的每一位成员。

Pearson 公司的许多人提供了非常有价值的帮助，却甘愿做幕后英雄。在此，希望大家都能够知道我对他们辛勤工作的感谢。与以往一样，我要衷心感谢本书编辑 Greg Doench，从本书的写作到出版，他一直在给予指导，同时感谢那些不知姓名的为本书做出贡献的人们。非常感谢 Julie Nahil 在图书制作方面给予的支持，感谢 Dmitry Kirsanov 和 Alina Kirsanova 完成手稿的编辑与排版工作。还要感谢早期版本中我的合作者 Gary Cornell，他已经投身到其他行业。

感谢早期版本的许多读者，他们指出了很多错误，并给出了很多有建设性的改进意见。我还要特别感谢优秀的审校团队，他们极其仔细地审阅了我的手稿，避免了很多令人尴尬的错误。

本书及早期版本的审校人员包括：Chuck Allison（尤他谷大学）、Lance Andersen（Oracle）、Gail Anderson（Anderson Software Group）、Paul Anderson（Anderson Software Group）、Alec Beaton（IBM）、Cliff Berg、Andrew Binstock（Oracle）、Joshua Bloch、David Brown、Corky Cartwright、Frank Cohen（PushToTest）、Chris Crane（devXsolution）、Dr. Nicholas J. De Lillo（曼哈顿学院）、Rakesh Dhoopar（Oracle）、David Geary（Clarity Training）、Jim Gish（Oracle）、Brian Goetz（Oracle）、Angela Gordon、Dan Gordon（Electric Cloud）、Rob Gordon、John Gray（哈特福德大学）、Cameron Gregory（olabs.com）、Marty Hall（coreservlets.com 公司）、Vincent Hardy（Adobe Systems）、Dan Harkey（圣何塞州立大学）、William Higgins（IBM）、Marc Hoffmann（mtrail）、Vladimir Ivanovic（PointBase）、Jerry Jackson（CA Technologies）、Heinz Kabutz（Java Specialists）、Stepan V. Kalinin（I-Teco/Servionica LTD）、Tim Kimmet（Walmart）、Chris Laffra、Charlie Lai（Apple）、Angelika Langer、Jeff Langr（Langr Software Solutions）、Doug Langston、Hang Lau（麦吉尔大学）、Mark Lawrence、Doug Lea（SUNY Oswego）、Gregory Longshore、Bob Lynch（Lynch Associates）、Philip Milne（顾问）、Mark Morrissey（俄勒冈州研究生院）、Mahesh Neelakanta（佛罗里达大西洋大学）、José Paumard（Oracle）、Hao Pham、Paul Philon、Blake Ragsdell、Stuart Reges（亚利桑那大学）、Simon Ritter（Azul Systems）、Rich Rosen（Interactive Data Corporation）、Peter Sanders（法国尼斯索菲亚安提波利斯大学）、Dr. Paul Sanghera（圣何塞州立大学布鲁克斯学院）、Paul Sevinc（Teamup AG）、Devang Shah（Sun Microsystems）、Yoshiki Shibata、Bradley A. Smith、Steven Stelting（Oracle）、Christopher Taylor、Luke Taylor（Valtech）、George Thiruvathukal、Kim Topley（StreamingEdge）、Janet Traub、Paul Tyma（顾问）、Peter van der Linden、Christian Ullenboom、Burt Walsh、Dan Xu（Oracle）和 John Zavgren（Oracle）。

Cay Horstmann  
2021 年 10 月于德国柏林

# 目 录

译者序	
前言	
致谢	
第 1 章 Java 程序设计概述 ..... 1	
1.1 Java 程序设计平台 ..... 1	
1.2 Java 白皮书的关键术语 ..... 2	
1.2.1 简单性 ..... 2	
1.2.2 面向对象 ..... 3	
1.2.3 分布式 ..... 3	
1.2.4 健壮性 ..... 3	
1.2.5 安全性 ..... 3	
1.2.6 体系结构中立 ..... 4	
1.2.7 可移植性 ..... 4	
1.2.8 解释性 ..... 5	
1.2.9 高性能 ..... 5	
1.2.10 多线程 ..... 5	
1.2.11 动态性 ..... 6	
1.3 Java applet 与 Internet ..... 6	
1.4 Java 发展简史 ..... 7	
1.5 关于 Java 的常见误解 ..... 10	
第 2 章 Java 编程环境 ..... 12	
2.1 安装 Java 开发工具包 ..... 12	
2.1.1 下载 JDK ..... 12	
2.1.2 设置 JDK ..... 13	
2.1.3 安装源文件和文档 ..... 15	
2.2 使用命令行工具 ..... 15	
2.3 使用集成开发环境 ..... 19	
2.4 JShell ..... 22	
第 3 章 Java 的基本程序设计结构 ..... 25	
3.1 一个简单的 Java 程序 ..... 25	
3.2 注释 ..... 28	
3.3 数据类型 ..... 28	
3.3.1 整型 ..... 29	
3.3.2 浮点类型 ..... 30	
3.3.3 char 类型 ..... 31	
3.3.4 Unicode 和 char 类型 ..... 31	
3.3.5 boolean 类型 ..... 32	
3.4 变量与常量 ..... 33	
3.4.1 声明变量 ..... 33	
3.4.2 初始化变量 ..... 34	
3.4.3 常量 ..... 35	
3.4.4 枚举类型 ..... 35	
3.5 运算符 ..... 36	
3.5.1 算术运算符 ..... 36	
3.5.2 数学函数与常量 ..... 36	
3.5.3 数值类型之间的转换 ..... 38	
3.5.4 强制类型转换 ..... 39	
3.5.5 赋值 ..... 39	
3.5.6 自增与自减运算符 ..... 40	
3.5.7 关系和 boolean 运算符 ..... 40	
3.5.8 条件运算符 ..... 41	
3.5.9 switch 表达式 ..... 41	
3.5.10 位运算符 ..... 42	
3.5.11 括号与运算符级别 ..... 43	
3.6 字符串 ..... 44	
3.6.1 子串 ..... 44	
3.6.2 拼接 ..... 44	
3.6.3 字符串不可变 ..... 45	
3.6.4 检测字符串是否相等 ..... 46	
3.6.5 空串与 Null 串 ..... 47	
3.6.6 码点与代码单元 ..... 47	

3.6.7 String API	49	4.3.1 Employee 类	104
3.6.8 阅读联机 API 文档	51	4.3.2 使用多个源文件	106
3.6.9 构建字符串	53	4.3.3 剖析 Employee 类	107
3.6.10 文本块	54	4.3.4 从构造器开始	108
3.7 输入与输出	56	4.3.5 用 var 声明局部变量	109
3.7.1 读取输入	56	4.3.6 使用 null 引用	109
3.7.2 格式化输出	58	4.3.7 隐式参数与显式参数	110
3.7.3 文件输入与输出	60	4.3.8 封装的优点	111
3.8 控制流程	62	4.3.9 基于类的访问权限	113
3.8.1 块作用域	62	4.3.10 私有方法	114
3.8.2 条件语句	63	4.3.11 final 实例字段	114
3.8.3 循环	64	4.4 静态字段与静态方法	115
3.8.4 确定性循环	68	4.4.1 静态字段	115
3.8.5 多重选择：switch 语句	71	4.4.2 静态常量	116
3.8.6 中断控制流程的语句	75	4.4.3 静态方法	116
3.9 大数	77	4.4.4 工厂方法	117
3.10 数组	79	4.4.5 main 方法	118
3.10.1 声明数组	79	4.5 方法参数	120
3.10.2 访问数组元素	81	4.6 对象构造	125
3.10.3 for each 循环	81	4.6.1 重载	126
3.10.4 数组拷贝	82	4.6.2 默认字段初始化	126
3.10.5 命令行参数	83	4.6.3 无参数的构造器	126
3.10.6 数组排序	84	4.6.4 显式字段初始化	127
3.10.7 多维数组	86	4.6.5 参数名	128
3.10.8 不规则数组	89	4.6.6 调用另一个构造器	129
第 4 章 对象与类	92	4.6.7 初始化块	129
4.1 面向对象程序设计概述	92	4.6.8 对象析构与 finalize 方法	133
4.1.1 类	93	4.7 记录	134
4.1.2 对象	94	4.7.1 记录概念	134
4.1.3 识别类	94	4.7.2 构造器：标准、自定义和简洁	136
4.1.4 类之间的关系	95	4.8 包	138
4.2 使用预定义类	96	4.8.1 包名	138
4.2.1 对象与对象变量	96	4.8.2 类的导入	138
4.2.2 Java 类库中的 LocalDate 类	99	4.8.3 静态导入	140
4.2.3 更改器方法与访问器方法	101	4.8.4 在包中增加类	140
4.3 自定义类	104		

4.8.5 包访问 .....	143	5.3 泛型数组列表 .....	186
4.8.6 类路径 .....	144	5.3.1 声明数组列表 .....	187
4.8.7 设置类路径 .....	146	5.3.2 访问数组列表元素 .....	189
4.9 JAR 文件 .....	146	5.3.3 类型化与原始数组列表的兼容性 .....	192
4.9.1 创建 JAR 文件 .....	146	5.4 对象包装器与自动装箱 .....	193
4.9.2 清单文件 .....	147	5.5 参数个数可变的方法 .....	196
4.9.3 可执行 JAR 文件 .....	148	5.6 抽象类 .....	197
4.9.4 多版本 JAR 文件 .....	149	5.7 枚举类 .....	201
4.9.5 关于命令行选项的说明 .....	150	5.8 密封类 .....	203
4.10 文档注释 .....	151	5.9 反射 .....	208
4.10.1 注释的插入 .....	151	5.9.1 Class 类 .....	209
4.10.2 类注释 .....	152	5.9.2 声明异常入门 .....	211
4.10.3 方法注释 .....	152	5.9.3 资源 .....	212
4.10.4 字段注释 .....	153	5.9.4 利用反射分析类的能力 .....	213
4.10.5 通用注释 .....	153	5.9.5 使用反射在运行时分析对象 .....	220
4.10.6 包注释 .....	154	5.9.6 使用反射编写泛型数组代码 .....	224
4.10.7 注释提取 .....	154	5.9.7 调用任意方法和构造器 .....	227
4.11 类设计技巧 .....	155	5.10 继承的设计技巧 .....	231
第 5 章 继承 .....	158	第 6 章 接口、lambda 表达式与内部类 .....	233
5.1 类、超类和子类 .....	158	6.1 接口 .....	233
5.1.1 定义子类 .....	158	6.1.1 接口的概念 .....	233
5.1.2 覆盖方法 .....	160	6.1.2 接口的属性 .....	239
5.1.3 子类构造器 .....	161	6.1.3 接口与抽象类 .....	240
5.1.4 继承层次结构 .....	165	6.1.4 静态和私有方法 .....	241
5.1.5 多态 .....	165	6.1.5 默认方法 .....	241
5.1.6 理解方法调用 .....	166	6.1.6 解决默认方法冲突 .....	242
5.1.7 阻止继承：final 类和方法 .....	169	6.1.7 接口与回调 .....	244
5.1.8 强制类型转换 .....	170	6.1.8 Comparator 接口 .....	246
5.1.9 instanceof 模式匹配 .....	172	6.1.9 对象克隆 .....	247
5.1.10 受保护访问 .....	173	6.2 lambda 表达式 .....	252
5.2 Object：所有类的超类 .....	174	6.2.1 为什么引入 lambda 表达式 .....	253
5.2.1 Object 类型的变量 .....	174	6.2.2 lambda 表达式的语法 .....	254
5.2.2 equals 方法 .....	175		
5.2.3 相等测试与继承 .....	176		
5.2.4 hashCode 方法 .....	179		
5.2.5 toString 方法 .....	181		

6.2.3 函数式接口 .....	256	7.4.2 启用和禁用断言 .....	313
6.2.4 方法引用 .....	258	7.4.3 使用断言完成参数检查 .....	313
6.2.5 构造器引用 .....	260	7.4.4 使用断言提供假设文档 .....	314
6.2.6 变量作用域 .....	261	7.5 日志 .....	315
6.2.7 处理 lambda 表达式 .....	263	7.5.1 基本日志 .....	316
6.2.8 再谈 Comparator .....	266	7.5.2 高级日志 .....	316
6.3 内部类 .....	267	7.5.3 修改日志管理器配置 .....	318
6.3.1 使用内部类访问对象状态 .....	267	7.5.4 本地化 .....	320
6.3.2 内部类的特殊语法规则 .....	270	7.5.5 处理器 .....	321
6.3.3 内部类是否有用、必要和 安全 .....	271	7.5.6 过滤器 .....	324
6.3.4 局部内部类 .....	273	7.5.7 格式化器 .....	324
6.3.5 由外部方法访问变量 .....	274	7.5.8 日志技巧 .....	324
6.3.6 匿名内部类 .....	275	7.6 调试技巧 .....	332
6.3.7 静态内部类 .....	278	第 8 章 泛型程序设计 .....	337
6.4 服务加载器 .....	281	8.1 为什么要使用泛型程序设计 .....	337
6.5 代理 .....	283	8.1.1 类型参数的好处 .....	337
6.5.1 何时使用代理 .....	284	8.1.2 谁想成为泛型程序员 .....	338
6.5.2 创建代理对象 .....	284	8.2 定义简单泛型类 .....	339
6.5.3 代理类的特性 .....	288	8.3 泛型方法 .....	341
第 7 章 异常、断言和日志 .....	290	8.4 类型变量的限定 .....	342
7.1 处理错误 .....	290	8.5 泛型代码和虚拟机 .....	344
7.1.1 异常分类 .....	291	8.5.1 类型擦除 .....	344
7.1.2 声明检查型异常 .....	293	8.5.2 转换泛型表达式 .....	346
7.1.3 如何抛出异常 .....	295	8.5.3 转换泛型方法 .....	346
7.1.4 创建异常类 .....	296	8.5.4 调用遗留代码 .....	348
7.2 捕获异常 .....	297	8.6 限制与局限性 .....	349
7.2.1 捕获异常概述 .....	297	8.6.1 不能用基本类型实例化类型 参数 .....	349
7.2.2 捕获多个异常 .....	299	8.6.2 运行时类型查询只适用于 原始类型 .....	349
7.2.3 再次抛出异常与异常链 .....	300	8.6.3 不能创建参数化类型的数组 ..	349
7.2.4 finally 子句 .....	301	8.6.4 Varargs 警告 .....	350
7.2.5 try-with-Resources 语句 .....	303	8.6.5 不能实例化类型变量 .....	351
7.2.6 分析栈轨迹元素 .....	305	8.6.6 不能构造泛型数组 .....	352
7.3 使用异常的技巧 .....	308	8.6.7 泛型类的静态上下文中类型 变量无效 .....	353
7.4 使用断言 .....	311		
7.4.1 断言的概念 .....	312		

8.6.8 不能抛出或捕获泛型类的实例	354	9.4.5 链接散列集与映射	412
8.6.9 可以取消对检查型异常的检查	354	9.4.6 枚举集与映射	413
8.6.10 注意擦除后的冲突	356	9.4.7 标识散列映射	413
8.7 泛型类型的继承规则	357	9.5 副本与视图	415
8.8 通配符类型	359	9.5.1 小集合	415
8.8.1 通配符概念	359	9.5.2 不可修改的副本和视图	417
8.8.2 通配符的超类型限定	360	9.5.3 子范围	418
8.8.3 无限定通配符	363	9.5.4 检查型视图	419
8.8.4 通配符捕获	363	9.5.5 同步视图	419
8.9 反射和泛型	365	9.5.6 关于可选操作的说明	420
8.9.1 泛型 Class 类	365	9.6 算法	423
8.9.2 使用 Class<T> 参数进行类型匹配	366	9.6.1 为什么使用泛型算法	423
8.9.3 虚拟机中的泛型类型信息	367	9.6.2 排序与混排	424
8.9.4 类型字面量	370	9.6.3 二分查找	427
第 9 章 集合	376	9.6.4 简单算法	428
9.1 Java 集合框架	376	9.6.5 批操作	429
9.1.1 集合接口与实现分离	376	9.6.6 集合与数组的转换	430
9.1.2 Collection 接口	379	9.6.7 编写自己的算法	431
9.1.3 迭代器	379	9.7 遗留的集合	432
9.1.4 泛型实用方法	382	9.7.1 Hashtable 类	432
9.2 集合框架中的接口	384	9.7.2 枚举	432
9.3 具体集合	386	9.7.3 属性映射	433
9.3.1 链表	387	9.7.4 栈	436
9.3.2 数组列表	395	9.7.5 位集	437
9.3.3 散列集	396	第 10 章 图形用户界面程序设计	441
9.3.4 树集	399	10.1 Java 用户界面工具包简史	441
9.3.5 队列与双端队列	403	10.2 显示窗体	442
9.3.6 优先队列	404	10.2.1 创建窗体	442
9.4 映射	405	10.2.2 窗体属性	444
9.4.1 基本映射操作	405	10.3 在组件中显示信息	448
9.4.2 更新映射条目	408	10.3.1 处理 2D 图形	452
9.4.3 映射视图	409	10.3.2 使用颜色	458
9.4.4 弱散列映射	411	10.3.3 使用字体	459

10.4.2 实例：处理按钮点击事件	467	11.6.1 网格包布局	535
10.4.3 简洁地指定监听器	471	11.6.2 定制布局管理器	543
10.4.4 适配器类	472	11.7 对话框	547
10.4.5 动作	473	11.7.1 选项对话框	547
10.4.6 鼠标事件	478	11.7.2 创建对话框	551
10.4.7 AWT 事件继承层次结构	482	11.7.3 数据交换	554
10.5 首选项 API	485	11.7.4 文件对话框	560
<b>第 11 章 Swing 用户界面组件</b>	<b>491</b>	<b>第 12 章 并发</b>	<b>567</b>
11.1 Swing 和模型 – 视图 – 控制器		12.1 什么是线程	567
设计模式	491	12.2 线程状态	572
11.2 布局管理概述	495	12.2.1 新建线程	572
11.2.1 布局管理器	495	12.2.2 可运行线程	572
11.2.2 边框布局	497	12.2.3 阻塞和等待线程	573
11.2.3 网格布局	498	12.2.4 终止线程	573
11.3 文本输入	499	12.3 线程属性	575
11.3.1 文本域	499	12.3.1 中断线程	575
11.3.2 标签和标签组件	501	12.3.2 守护线程	578
11.3.3 密码域	502	12.3.3 线程名	578
11.3.4 文本区	502	12.3.4 未捕获异常的处理器	578
11.3.5 滚动窗格	503	12.3.5 线程优先级	579
11.4 选择组件	505	12.4 同步	580
11.4.1 复选框	506	12.4.1 竞态条件的一个例子	580
11.4.2 单选按钮	508	12.4.2 竞态条件详解	582
11.4.3 边框	511	12.4.3 锁对象	584
11.4.4 组合框	513	12.4.4 条件对象	587
11.4.5 滑动条	516	12.4.5 synchronized 关键字	591
11.5 菜单	522	12.4.6 同步块	595
11.5.1 菜单构建	522	12.4.7 监视器概念	597
11.5.2 菜单项中的图标	524	12.4.8 volatile 字段	597
11.5.3 复选框和单选按钮菜单项	525	12.4.9 final 变量	598
11.5.4 弹出菜单	526	12.4.10 原子性	599
11.5.5 键盘助记符和加速器	527	12.4.11 死锁	600
11.5.6 启用和禁用菜单项	528	12.4.12 为什么废弃 stop 和 suspend 方法	603
11.5.7 工具栏	532	12.4.13 按需初始化	604
11.5.8 工具提示	534	12.4.14 线程局部变量	605
11.6 复杂的布局管理	534		

12.5 线程安全的集合 .....	606
12.5.1 阻塞队列 .....	607
12.5.2 高效的映射、集和队列 .....	612
12.5.3 映射条目的原子更新 .....	614
12.5.4 并发散列映射的批操作 .....	617
12.5.5 并发集视图 .....	618
12.5.6 写时拷贝数组 .....	619
12.5.7 并行数组算法 .....	619
12.5.8 较早的线程安全集合 .....	620
12.6 任务和线程池 .....	621
12.6.1 Callable 与 Future .....	621
12.6.2 执行器 .....	622
12.6.3 控制任务组 .....	625
12.6.4 fork-join 框架 .....	630
12.7 异步计算 .....	632
12.7.1 可完成 Future .....	632
12.7.2 组合可完成 Future .....	634
12.7.3 用户界面回调中的长时间 运行任务 .....	639
12.8 进程 .....	646
12.8.1 建立进程 .....	646
12.8.2 运行进程 .....	647
12.8.3 进程句柄 .....	648
附录 Java 关键字 .....	652

# 第1章 Java 程序设计概述

- ▲ Java 程序设计平台
- ▲ Java 白皮书的关键术语
- ▲ Java applet 与 Internet

- ▲ Java 发展简史
- ▲ 关于 Java 的常见误解

1996 年 Java 第一次发布就引起了人们的极大兴趣。关注 Java 的人士不仅限于计算机出版界，还有诸如《纽约时报》《华盛顿邮报》和《商业周刊》这样的主流媒体。Java 是第一个也是唯一一个在 National Public Radio 上占用了 10 分钟时间进行介绍的程序设计语言，并且还得到了 100 000 000 美元的风险投资基金。这些基金全部用来支持用这种特别的计算机语言开发的产品。你可能想了解 Java 语言的发展，这一章就会带你简单地重温这段历史。

## 1.1 Java 程序设计平台

在本书的第 1 版中，我和合著者 Gary Cornell 是这样描述 Java 的：

“作为一种计算机语言，Java 的广告词确实有点夸大其词。当然，Java 的确是一种优秀的程序设计语言。作为一个名副其实的程序设计人员，使用 Java 无疑是一个比较好的选择。我们认为：Java 本来有潜力成为一种卓越的程序设计语言，但可能有些为时过晚。一旦一种语言得到广泛应用，与现存代码尴尬的兼容性问题就摆在了人们的面前。”

关于这段文字，我们的编辑受到 Sun 公司某高层人士的严厉批评（Sun 是最早开发 Java 的公司）。Java 有许多非常优秀的语言特性，本章稍后会详细地讨论这些特性。但它确实也有缺点，由于兼容性需求，新增的一些特性就没有原有的特性那么精巧。

但是，正像我们在第 1 版中所说的，Java 并不只是一个语言。在此之前出现的那么多种语言都没有引起那么大的轰动。Java 是一个完整的平台，有一个庞大的库，其中包含了大量可重用的代码，还有一个提供诸如安全性、跨操作系统的可移植性以及自动垃圾收集等服务的执行环境。

作为一名程序设计人员，你可能希望能有这样一种语言，既要有令人舒适的语法，也要有易于理解的语义（C++ 就不是这样的语言）。Java 完全满足这些要求，另外还有很多其他优秀语言也能满足要求。不过尽管有些语言提供了可移植性、垃圾收集等特性，但它们没有提供一个丰富的库。如果你想要酷炫的绘图功能、网络连接或数据库存取特性，就必须自己动手编写代码。而 Java 一应俱全，它具备所有这些特性，是一种功能齐全的出色语言和一个高质量的执行环境，同时有一个庞大的库。正是因为 Java 集多种优势于一身，所以对广大程

序设计人员有着不可抗拒的吸引力。

## 1.2 Java 白皮书的关键术语

Java 的设计者编写了一个颇有影响力的白皮书，来解释设计初衷以及完成的情况，他们还发布了一个简短的摘要。这个摘要按以下 11 个关键术语进行组织：

- |           |         |
|-----------|---------|
| 1) 简单性    | 7) 可移植性 |
| 2) 面向对象   | 8) 解释性  |
| 3) 分布式    | 9) 高性能  |
| 4) 健壮性    | 10) 多线程 |
| 5) 安全性    | 11) 动态性 |
| 6) 体系结构中立 |         |

在后面的小节中，我们将提供一个小结，给出白皮书中的相关说明（这是 Java 设计者对各个关键术语的描述），另外我还会根据使用 Java 当前版本的经验，给出对这些术语的理解。

**注释：**白皮书可以在 [www.oracle.com/technetwork/java/langenv-140151.html](http://www.oracle.com/technetwork/java/langenv-140151.html) 上找到。关于 11 个关键术语的概述请参见 <http://horstmann.com/corejava/java-an-overview/7Gosling.pdf>。

### 1.2.1 简单性

我们希望构建一个无须深奥的专业训练就可以进行编程的系统，并且要符合当今的标准惯例。因此，尽管我们发现 C++ 不太适用，但在设计 Java 的时候还是尽可能地接近 C++，以使系统更易于理解。Java 剔除了 C++ 中许多很少使用、难以理解、容易混淆的特性。在我们看来，这些特性带来的问题远远多于它们的好处。

的确，Java 语法是 C++ 语法的一个“纯净”版本。这里没有头文件、指针运算（甚至没有指针语法）、结构、联合、操作符重载、虚基类等（请参阅本书各个章节给出的 C++ 注释，其中比较详细地解释了 Java 与 C++ 之间的区别）。不过，Java 设计者并没有试图修正 C++ 中所有不适当的特性。例如，switch 语句的语法在 Java 中就没有改变。如果你了解 C++，会发现可以轻而易举地转换到 Java 语法。

Java 发布时，实际上 C++ 并不是最常用的程序设计语言。很多开发人员都在使用 Visual Basic 和它的拖放式编程环境。这些开发人员并不觉得 Java 简单。很多年之后 Java 开发环境才迎头赶上。如今，Java 开发环境已经远远超越了大多数其他编程语言的开发环境。

“简单”的另一面是“小”。Java 的目标之一是支持开发能够在小型机器上独立运行的软件。基本的解释器和类支持大约仅为 40KB，再加上基础的标准类库和线程支持（基本上是一个自包含的微内核），大约需要增加 175KB。

在当时，这是一个了不起的成就。当然，由于不断的扩展，类库已经相当庞大了。现在

还有一些带有较小类库的独立版本，这些版本适用于嵌入式设备和智能卡。

### 1.2.2 面向对象

简单地讲，面向对象设计是一种程序设计技术。它将重点放在数据（即对象）和对象的接口上。用木匠打一个比方：一个“面向对象的”木匠主要关注的是所制作的椅子，其次才是使用的工具；一个“非面向对象的”木匠主要考虑的则是使用的工具。在本质上，Java的面向对象能力与C++是一样的。

开发Java时面向对象技术已经相当成熟。Java的面向对象特性与C++旗鼓相当。Java与C++的主要不同点在于多重继承，在Java中，取而代之的是更简单的接口概念。与C++相比，Java提供了更丰富的运行时自省功能（有关内容将在第5章中讨论）。

### 1.2.3 分布式

Java有一个丰富的例程库，用于处理HTTP和FTP之类的TCP/IP协议。Java应用程序能够通过URL打开和访问网上的对象，其便捷程度就好像访问本地文件一样。

如今，这一点被认为是理所当然的，不过在1995年主要还是从C++或Visual Basic程序连接Web服务器。

### 1.2.4 健壮性

Java的设计目标之一是要让用Java编写的程序具有多方面的可靠性。Java非常强调进行早期的问题检测、后期的动态（运行时）检测，以及消除容易出错的情况……Java与C/C++最大的不同在于Java采用的指针模型可以消除重写内存和损坏数据的可能性。

Java编译器能够检测许多其他语言中仅在运行时才能够检测出来的问题。至于第二点，对于曾经花费几个小时来检查由于指针bug而引起内存冲突的人来说，一定很喜欢Java的这一特性。

### 1.2.5 安全性

Java要适用于网络/分布式环境。为了实现这个目标，安全性颇受重视。使用Java可以构建防病毒、防篡改的系统。

从一开始，Java就设计成能够防范各种攻击，其中包括：

- 运行时堆栈溢出，这是蠕虫和病毒常用的攻击手段。
- 破坏自己的进程空间之外的内存。
- 未经授权读写文件。

起初，Java对下载代码的态度是“尽管来吧！”不可信代码在沙箱环境中执行，在这里它不会影响主系统。用户可以确信不会发生不好的事情，因为Java代码不论来自哪里，都不

能逃离这个沙箱。

不过，Java 的安全模型很复杂。Java 开发包（Java Development Kit, JDK）的第一版发布之后不久，普林斯顿大学的一些安全专家就发现一些小 bug 会允许不可信的代码攻击主系统。

最初安全 bug 可以快速修复。遗憾的是，经过一段时间之后，黑客已经很擅长找出安全体系结构实现中的小漏洞。Sun 公司以及之后的 Oracle 公司为不断修复 bug 经历了一段很是艰难的日子。

遭遇多次高调攻击之后，浏览器开发商和 Oracle 公司变得越来越谨慎。有一段时间，远程代码必须有数字签名。如今，通过浏览器交付 Java 应用已经是很遥远的记忆。

**注释：**现在看来，尽管 Java 安全模型没有原先预想的那么成功，但 Java 在那个时代确实相当超前。微软公司提出了一种与之竞争的代码交付机制，称为 ActiveX，其安全性完全依赖于数字签名。显然这是不够的，因为微软公司的产品的任何用户都可以证实，一些知名开发商的程序确实会崩溃并对系统产生危害。

### 1.2.6 体系结构中立

编译器生成一个体系结构中立的目标文件格式，这是一种编译型代码，这些编译型代码可以在很多处理器上运行（只要它们有 Java 运行时系统）。Java 编译器通过生成与特定计算机体系结构无关的字节码指令来实现这一特性。精心设计的字节码不仅可以很容易地在任何机器上解释执行，而且可以很容易地动态转换为原生机器代码。

当时，为“虚拟机”生成代码并不是一个新思路，诸如 Lisp、Smalltalk 和 Pascal 等编程语言多年前就已经采用了这种技术。

当然，解释虚拟机指令肯定比全速运行机器指令慢很多。不过，虚拟机有一个选项，可以将执行最频繁的字节码序列转换成机器码，这一过程称为即时编译（just-in-time compilation）。

Java 虚拟机还有其他一些优点。它可以检查指令序列的行为，从而增强安全性。

### 1.2.7 可移植性

与 C 和 C++ 不同，Java 规范中没有“依赖具体实现”的地方。基本数据类型的大小以及有关运算的行为都是明确的。

例如，Java 中的 int 总是 32 位整数，而在 C/C++ 中，int 可能是 16 位整数、32 位整数，也可能是编译器开发商指定的任何其他大小。唯一的限制是，int 类型的字节数不能低于 short int，并且不能高于 long int。在 Java 中，数值类型有固定的字节数，这消除了代码移植时一个令人头痛的主要问题。二进制数据以固定的格式进行存储和传输，消除了有关字节顺序的困扰。字符串则采用标准的 Unicode 格式存储。

作为系统组成部分的类库定义了可移植的接口。例如，有一个抽象 Window 类，并给出了面向 UNIX、Windows 和 Macintosh 环境的不同实现。

选择 Window 类作为例子可能并不太合适。凡是尝试过的人都知道，要编写一个在 Windows、Macintosh 和 10 种不同风格的 UNIX 上看起来都不错的程序是多么困难。Java 1.0 就尝试着做了这么一个壮举，发布了一个简单的工具包，为多个不同平台提供了常用的用户界面元素。遗憾的是，尽管花费了大量的心血，结果却不尽如人意，这个库并不能在不同系统上都提供让人接受的结果。原先的用户界面工具包已经重写，而且后来又再次重写，跨平台的可移植性仍然是个问题。

不过，除了与用户界面有关的部分外，所有其他 Java 库确实能很好地支持平台独立性。你可以处理文件、正则表达式、XML、日期和时间、数据库、网络连接、线程等，而不用操心底层操作系统。不仅程序是可移植的，Java API 往往也比原生 API 质量更高。

### 1.2.8 解释性

Java 解释器可以在任何移植了解释器的机器上直接执行 Java 字节码。由于链接（linking）是一个增量式的轻量级过程，所以，开发过程也会更加快捷，更具有探索性。

这看上去很不错。用过 Lisp、Smalltalk、Visual Basic、Python、R 或 Scala 的人都知道“快捷而且具有探索性”的开发过程是怎样的。你可以做些尝试，然后立即就能看到结果。在 Java 发展的前 20 年里，开发环境并没有把重点放在这种体验上。直到 Java 9 才提供了 jshell 工具来支持快捷而且具有探索性的编程。

### 1.2.9 高性能

尽管解释型字节码的性能通常已经足够让人满意，但在有些场合下还需要更高的性能。字节码可以（在运行时）动态转换为面向运行这个应用的特定 CPU 的机器码。

使用 Java 的头几年，许多用户不同意“性能已经足够让人满意”的说法。不过，现在的即时编译器已经非常出色，可以与传统编译器相媲美，而且在某些情况下甚至超越了传统编译器，原因是它们有更多的可用信息。例如，即时编译器可以监控哪些代码频繁执行，并优化这些代码以提高速度。更为复杂的优化是消除函数调用（即“内联”）。即时编译器知道已经加载了哪些类。基于当前加载的类集合，如果一个特定的函数不会被覆盖，就可以使用内联。必要时，以后还可以撤销这种优化。

### 1.2.10 多线程

多线程可以带来更好的交互响应和实时行为。

如今，我们非常关注并发性，因为摩尔定律即将走到尽头。我们不再追求更快的处理器，而是着眼于获得更多的处理器，而且要让它们保持繁忙。不过，可以看到，大多数编程

语言对于这个问题并没有显示出足够的重视。

Java 在当时很超前。它是第一个支持并发程序设计的主流语言。从白皮书中可以看到，它的出发点稍有些不同。当时，多核处理器还很神秘，而 Web 编程才刚刚起步，处理器要花很长时间等待服务器的响应，需要并发程序设计来确保用户界面没有“冻住”。

并发程序设计绝非易事，不过 Java 在这方面表现很出色，可以很好地管理这个工作。

### 1.2.11 动态性

从很多方面来看，Java 与 C 或 C++ 相比更具有动态性。Java 设计为能够适应不断演进的环境。库可以自由地添加新方法和实例变量，而对客户端没有任何影响。在 Java 中找出运行时类型信息十分简单。

需要为正在运行的程序增加代码时，动态性将是一个非常重要的特性。一个很好的例子是：在浏览器中运行从 Internet 下载的代码。如果使用 C 或 C++，这确实难度很大，不过 Java 设计者很清楚动态语言可以很容易地让一个正在运行的程序实现演进。最终，他们将这一特性引入到这个主流程序设计语言中。

**注释：**Java 成功地推出后不久，微软就发布了一个叫作 J++ 的产品，它与 Java 有几乎相同的编程语言和虚拟机。现在，微软不再支持 J++，取而代之的是另一个名为 C# 的语言。C# 与 Java 有很多相似之处，不过在一个不同的虚拟机上运行。本书不准备介绍 J++ 或 C# 语言。

## 1.3 Java applet 与 Internet

这里的想法很简单：用户从 Internet 下载 Java 字节码，并在自己的机器上运行。在网页中运行的 Java 程序称为 applet。要使用 applet，只需要一个启用 Java 的 Web 浏览器，它会为你执行字节码。不需要安装任何软件。只要你访问包含 applet 的网页，都会得到这个程序的最新版本。最重要的是，归功于虚拟机的安全性，我们不必担心来自恶意代码的攻击。

在网页中插入一个 applet 就如同在网页中嵌入一幅图片。applet 会成为页面的一部分。文本环绕在 applet 占据的空间周围。关键是，这个图片是活动的（alive）。它会对用户命令做出响应，改变外观，在显示它的计算机和提供它的计算机之间交换数据。

图 1-1 展示了 Jmol applet，它会显示分子结构。可以利用鼠标旋转和放大各个分子，从而更好地理解分子结构。在发明 applet 的时代，用网页是无法实现这种直接的操作的，那时只有基本的 JavaScript 而没有 HTML 画布。

applet 首次出现时，人们欣喜若狂。许多人相信 applet 的魅力会让 Java 迅速流行起来。然而，初期的兴奋很快就变成了沮丧。不同版本的 Netscape 与 Internet Explorer 运行不同版本的 Java，其中有些早已过时。这种糟糕的情况导致更加难以利用 Java 的最新版本开发 applet。实际上，为了在浏览器中得到动态效果，Adobe 的 Flash 技术变得相当流行。后来，Java 受到严重安全问题的困扰时，浏览器也放弃了对 applet 的支持。当然，Flash 的命运也

好不到哪里去。

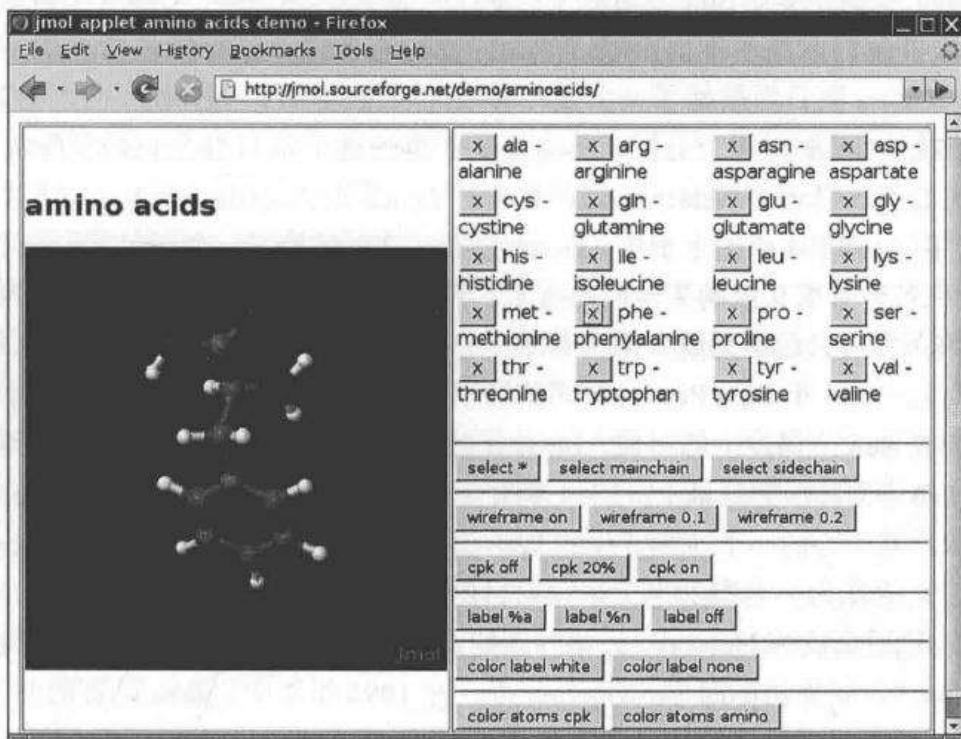


图 1-1 Jmol applet

## 1.4 Java 发展简史

本节将介绍 Java 的发展简史。这些内容来自很多已发布的资料（最重要的是 *SunWorld* 的在线杂志 1995 年 7 月刊上对 Java 创始人的专访）。

Java 的历史要追溯到 1991 年。由 Patrick Naughton 和 James Gosling（一个全能的计算机奇才，Sun 公司会士）带领的 Sun 公司的一个工程师小组想要设计一种小型的计算机语言，希望用于有线电视转换盒之类的消费设备。由于这些消费设备的处理能力和内存都很有限，所以这个语言必须非常小，而且要能够生成很紧凑的代码。另外，由于不同的厂商会选择不同的中央处理器（CPU），因此很重要的一点是这种语言不应与任何特定的体系结构绑定。这个项目被命名为“Green”。

代码短小、紧凑且与平台无关，这些要求促使开发团队设计出一个可移植的语言，可以为虚拟机生成中间代码。

Sun 公司的人都有 UNIX 的应用背景。因此，所开发的语言以 C++ 为基础，而不是 Lisp、Smalltalk 或 Pascal。不过，正如 Gosling 在专访中谈到：“毕竟，语言只是实现目标的工具，而不是目标本身。”Gosling 把这种语言称为“Oak”（这么起名大概是因为他非常喜欢他在 Sun 公司的办公室窗外的一棵橡树）。Sun 公司的人后来发现，已经有另外一个计算机语言取名为 Oak，于是，他们将这个语言改名为 Java。事实证明这是一个很有灵感的选择。

1992 年, Green 项目发布了它的第一个产品, 称之为 “\*7”。这个产品可以提供非常智能的远程控制。遗憾的是, Sun 公司对生产这个产品并不感兴趣, Green 项目组的人员必须找到其他方法将他们的技术推向市场。然而, 也没有任何一家标准消费品电子公司对此感兴趣。于是, Green 项目组投标了一个设计有线电视盒的项目, 它能提供视频点播等新型有线服务, 但他们没能拿到这个合同(有趣的是, 得到这个项目的公司的领导人恰恰是开创 Netscape 公司的 Jim Clark。Netscape 公司后来对 Java 的成功做出了很大贡献)。

在 1993 年以及 1994 年的上半年, Green 项目(这时候换了一个新名字——“First Person 公司”)一直在苦苦寻求买家购买他们的技术。然而, 一个也没有找到(Patrick Naughton——项目组的创始人之一, 也是完成大部分营销工作的人, 声称为了销售这项技术, 累计飞行了 300 000 英里<sup>①</sup>)。1994 年 First Person 公司解散了。

当这一切在 Sun 公司发生的时候, Internet 的万维网也在日渐发展壮大。万维网的关键是浏览器把超文本页面转换到屏幕上。1994 年大多数人都在使用 Mosaic, 这是 1993 年出自伊利诺伊大学超级计算中心的一个非商业化的 Web 浏览器(Mosaic 的一部分是由 Marc Andreessen 编写的。当时, 他作为一名参加半工半读项目的本科生, 编写了这个软件, 每小时的薪水只有 6.85 美元。他后来成为 Netscape 公司的创始人之一和技术总监, 可谓名利双收)。

在接受 *SunWorld* 采访的时候, Gosling 说, 在 1994 年年中, Java 语言的开发者意识到:“我们能建立一个相当酷的浏览器。在客户 / 服务器主流框架中, 浏览器恰好需要我们已经完成的一些工作: 体系结构中立、实时、可靠、安全——这些问题在工作站环境并不太重要, 所以, 我们决定开发浏览器。”

实际的浏览器是由 Patrick Naughton 和 Jonathan Payne 开发的, 并演变为 HotJava 浏览器。HotJava 浏览器采用 Java 编写, 以炫耀 Java 语言超强的能力。这个浏览器能够在网页中执行内嵌的 Java 代码。这一“技术证明”在 1995 年 5 月 23 日的 SunWorld’95 大会上展示, 引发了人们对 Java 的狂热追逐并延续至今。

1996 年年初, Sun 公司发布了 Java 的第 1 个版本。人们很快地意识到 Java 1.0 不能用来完成真正的应用开发。的确, 可以使用 Java 1.0 实现在画布上随机跳动的“神经质文本”applet, 但它没有提供打印功能。坦率地说, Java 1.0 的确没有为其黄金时期的到来做好准备。后来的 Java 1.1 弥补了大多明显的缺陷, 大大改进了反射能力, 并为 GUI 编程增加了新的事件处理模型。不过它仍然有很大的局限性。

1998 年 JavaOne 会议的头号新闻是即将发布 Java 1.2 版。这个版本将早期玩具式的 GUI 和图形工具包代之以复杂而且可伸缩的工具包。在 1998 年 12 月 Java 1.2 发布仅 3 天之后, Sun 公司市场部将它改名为更加吸引人的“Java 2 标准版软件开发包 1.2 版”。

除了“标准版”(Standard Edition)之外, Sun 公司还推出了另外两个版本: 一个是用于手机等嵌入式设备的“微型版”(Micro Edition); 另一个是用于服务器端处理的“企业版”(Enterprise Edition)。本书主要介绍标准版。

---

<sup>①</sup> 1 英里约为 1609 米。——编辑注

标准版的 1.3 和 1.4 版本对最初的 Java 2 版本做出了增量式的改进，提供了不断扩展的标准类库，提高了性能，当然，还修正了一些 bug。在此期间，原先对 Java applet 和客户端应用的炒作逐渐消退，但 Java 成了服务器端应用的首选平台。

5.0 版是自 1.1 版以来第一个对 Java 语言做出重大改进的版本（这一版本原来定为 1.5 版，但在 2004 年的 JavaOne 会议之后，版本号直接升至 5.0）。经过多年的研究，这个版本添加了泛型类型（generic type，大致相当于 C++ 的模板），其挑战性在于添加这一特性而不需要对虚拟机做任何修改。另外，受到 C# 的启发，还增加了几个很有用的语言特性：“for each” 循环、自动装箱和注解。

6 版（没有后缀 .0）于 2006 年年底发布。同样，这个版本没有对语言方面再进行修改，而是做了另外一些性能改进，并增强了类库。

随着数据中心越来越依赖于商业硬件而不是专用服务器，Sun 公司终于陷入困境，于 2009 年被 Oracle 公司收购。Java 的开发停滞了很长一段时间。直到 2011 年 Oracle 公司发布了 Java 的一个新版本——Java 7，其中只做了一些简单的改进。

2014 年，Java 8 终于发布，在近 20 年中这个版本的改变最大。Java 8 包含了一种“函数式”编程方式，可以很容易地表述能并发执行的计算。所有编程语言都必须与时俱进，Java 在这方面显示出了非凡的能力。

Java 9 的主要特性要一直追溯到 2008 年。那时，Java 平台的首席工程师 Mark Reinhold 开始着力解析这个庞大的 Java 平台。为此引入了模块（module），模块是提供一个特定功能的自包含代码单元。设计和实现一个适用于 Java 平台的模块系统前后用了 11 年，而它是否也适用于 Java 应用和类库还有待观察。Java 9 于 2017 年发布，它还提供了另外一些吸引人的特性，我们将在本书中介绍这些特性。

从 2018 年开始，每 6 个月就会发布一个 Java 版本，以支持更快地引入新特性。每过一段时间，会把某个版本（如 Java 11 和 Java 17）指定为长期支持版本。中间版本提供了一种试验新特性的机制。

表 1-1 展示了 Java 语言及类库的演进。可以看到，API 的规模有了惊人的增长。

表 1-1 Java 语言及类库的演进

版本	年份	新语言特性	类与接口的数量
1.0	1996	语言本身	211
1.1	1997	内部类	477
1.2	1998	strictfp 修饰符	1 524
1.3	2000	无	1 840
1.4	2002	断言	2 723
5.0	2004	泛型类、“for each” 循环、可变参数、自动装箱、元数据、枚举、静态导入	3 279
6	2006	无	3 793
7	2011	基于字符串的 Switch 语句、菱形运算符、二进制字面量、异常处理增强	4 024

(续)

版 本	年 份	新语言特性	类与接口的数量
8	2014	Lambda 表达式、包含默认方法的接口、流和日期 / 时间库	4 240
9	2017	模块、其他的语言和类库增强	6 005
11	2018	局部变量类型推导 (var)、HTTP 客户端、移除 Java FX、JNLP、Java EE 重叠模块和 CORBA	4 410
17	2021	Switch 表达式、文本块、instanceof 模式匹配、记录、密封类	4 859

## 1.5 关于 Java 的常见误解

在结束本章之前，我们将列出关于 Java 的一些常见误解，同时给出解释。

### 1. Java 是 HTML 的扩展。

Java 是一种程序设计语言，HTML 是一种描述网页结构的方式。除了用于在网页上放置 Java applet 的 HTML 扩展之外，两者没有任何共同之处。

### 2. 我使用 XML，所以不需要 Java。

Java 是一种程序设计语言，XML 是一种描述数据的方式。可以使用任何一种程序设计语言处理 XML 数据，而 Java API 对 XML 处理提供了很好的支持。此外，许多重要的 XML 工具都是用 Java 实现的。有关的更多信息请参见卷 II。

### 3. Java 是一种非常容易学习的程序设计语言。

像 Java 这种功能强大的语言大多都不太容易学习。首先，必须将编写玩具式程序的轻松与开发实际项目的艰难区分开来。另外，需要注意的是：本书只用了 7 章讨论 Java 语言，其余几章和卷 II 都在介绍如何使用 Java 类库来具体应用 Java 语言。Java 类库包含数千个类和接口，还有数万个函数。好在，你不需要知道其中的每一个类或函数，不过，要想用 Java 解决实际问题，还是需要了解不少内容的。

### 4. Java 将成为适用于所有平台的通用编程语言。

从理论上讲，这是完全有可能的。但在实际中，某些领域其他语言有更出色的表现，比如，Objective C 和后来的 Swift 在 iOS 设备上就有着无可取代的地位。浏览器中的处理几乎完全由 JavaScript 掌控。Windows 程序通常都用 C++ 或 C# 编写。Java 在服务器端编程和跨平台客户端应用领域则很有优势。

### 5. Java 只不过是另外一种程序设计语言。

Java 是一种很好的程序设计语言，很多程序设计人员喜欢 Java 胜过 C、C++ 和 C#。有几百种很好的程序设计语言没有广泛流行，而有明显缺陷的语言（如 C++ 和 Visual Basic）却大行其道。

这是为什么呢？程序设计语言的成功更多地取决于其支持系统（support system）的能力，而不是语法的精巧性。人们主要关注的是：是否提供了有用、便捷和标准的库来实现所需要的特性？是否有工具开发商建立了强大的编程和调试环境？语言和工具集是否与计算基础架

构的其他部分有效整合？Java的成功缘于其类库能够让人们轻松地完成原本有一定难度的工作，例如网络连接、Web应用和并发。Java减少了指针错误，这是一个额外的好处，因此使用Java编程的效率更高。但这些并不是Java成功的全部原因。

#### 6. Java是专用的，应该避免使用。

最初创建Java时，Sun公司为发布者和最终用户提供了免费许可。尽管Sun公司对Java拥有最终的控制权，不过在语言版本的不断发展和新库的设计过程中还涉及很多其他公司。虚拟机和类库的源代码可以免费获得，不过仅限于查看，而不能修改和再发布。Java是“闭源的，不过可以很好地使用”。

这种状况在2007年发生了巨大变化，Sun公司宣布Java未来的版本将在General Public License(GPL)下发布(Linux也使用同样的开放源代码许可)。Oracle公司一直致力于保持Java开源。目前有多个开源Java实现提供商，分别提供不同级别的承诺和支持。

#### 7. Java是解释性的，因此对于关键应用速度太慢了。

早期的Java确实是解释性的。现在Java虚拟机使用了即时编译器，因此用Java编写的“热点”代码运行速度与C++相差无几，有些情况下甚至更快。

#### 8. 所有的Java程序都在网页中运行。

有一段时间，Java applet在Web浏览器中运行。如今，Java程序是运行在Web浏览器之外的独立应用。实际上，大多数Java程序都在服务器上运行，为网页生成代码或者计算业务逻辑。

#### 9. Java程序存在重大安全风险。

对于早期的Java，有过关于Java安全系统失效的报道，曾经引起过公众关注。研究人员努力找出Java的漏洞，并质疑applet安全模型的强度和复杂度，将这视为一种挑战。人们很快就解决了所发现的技术问题。后来又发现了更严重的漏洞，而Sun公司以及后来的Oracle公司反应却很迟缓。浏览器制造商禁用了Java applet支持。促成applet的安全管理器体系结构现在已经过时。如今，Java应用与其他应用同样安全。由于虚拟机提供的保护，Java应用比用C或C++编写的应用要安全得多。

#### 10. JavaScript是Java的简易版。

JavaScript是一种可以在网页中使用的脚本语言，它由Netscape发明，最初的名字是LiveScript。JavaScript的语法让人想到Java，因为名字也有些相像，但除此之外，两者并无任何关系。尤其是，Java是强类型的，编译器能捕获类型滥用导致的很多错误。而在JavaScript中，只有当程序运行时才能发现这些错误，所以消除错误要费劲得多。

#### 11. 使用Java时，可以用廉价的“Internet设备”取代桌面计算机。

Java刚刚发布的时候，一些人打赌这肯定会发生。一些公司已经生产出支持Java的网络计算机原型，不过用户还不打算放弃功能强大且方便的桌面计算机，而去使用没有本地存储而且功能有限的网络计算机。当然，如今世界已经发生改变，对于大多数最终用户，常用的平台往往是手机或平板电脑。这些设备大多使用Android(安卓)平台。学习Java编程对Android编程也很有帮助。

# 第2章 Java 编程环境

- ▲ 安装 Java 开发工具包
- ▲ 使用命令行工具

- ▲ 使用集成开发环境
- ▲ JShell

这一章主要介绍如何安装 Java 开发工具包 (JDK)，以及如何编译和运行 Java 程序。可以在终端窗口中键入命令来运行 JDK 工具。不过，很多程序员更喜欢使用集成开发环境。你会了解如何使用一个免费的开发环境编译和运行 Java 程序。一旦掌握了本章的技术，并选定开发工具，就可以继续学习第 3 章，开始研究 Java 程序设计语言。

## 2.1 安装 Java 开发工具包

原先，Oracle 公司会提供最新、最完备的 Java 开发工具包 (JDK) 版本。如今，很多不同公司（包括 Microsoft、Amazon、Red Hat 和 Azul）都提供了最新的 OpenJDK 构建版本，有些公司的许可条件比 Oracle 公司更宽松。我写这一章时，最喜欢访问的网站是 <https://adoptium.net>，这个网站由开发商、开发人员和用户组共同组成的一个社区运营，为 Linux、Mac OS 和 Windows 提供了免费的构建版本。

### 2.1.1 下载 JDK

可以从 <https://adoptium.net> 下载 Java 开发工具包，或者也可以从 Oracle 公司网站 [www.oracle.com/technetwork/java/javase/downloads](http://www.oracle.com/technetwork/java/javase/downloads) 或其他提供商下载。要使用 Java SE 17 (LTS) JDK。表 2-1 总结了在下载网站上可能遇到的缩略语和术语。

表 2-1 Java 术语

术语名	缩 写	解 释
Java Development Kit (Java 开发工具包)	JDK	编写 Java 程序的程序员使用的软件
Java Runtime Environment (Java 运行时环境)	JRE	用来运行 Java 程序的软件，不带开发工具。这不是你想要的
Standard Edition (标准版)	SE	用于桌面或简单服务器应用的 Java 平台。这是你想要的
Micro Edition (微型版)	ME	用于小型设备的 Java 平台
OpenJDK	—	Java SE 的一个免费开源实现
Hotspot	—	Oracle 开发的“即时”编译器。如果要你选择，请选择这个选项
OpenJ9	—	IBM 开发的另一个“即时”编译器
Long Term Support (长期支持版本)	LTS	很多年都支持的一个版本，这不同于每 6 个月发布的那些展示新特性的版本。要选择最新的 LTS 版本

### 2.1.2 设置 JDK

下载 JDK 之后，需要安装这个开发工具包并确定要安装在哪里，后面还会需要这个信息。

- 在 Windows 上，启动安装程序，会询问你要把 JDK 安装到哪里。最好不要接受默认位置（这个路径名中有空格），如 c:\Program Files\Java\jdk-17.0.x。取出路径名中的 Program Files 部分就可以了。
- 在 Mac 上，运行安装程序。这会把软件安装到 /Library/Java/JavaVirtualMachines/jdk-17.0.x.jdk/Contents/Home。可以用 Finder 找到这个目录。
- 在 Linux 上，只需要把 .tar.gz 文件解压缩到你选择的某个位置，如你的主目录或者 /opt。如果从 RPM 文件安装，则要仔细确认安装到 /usr/java/jdk-17.0.x 中。

在本书中，安装目录用 *jdk* 表示。例如，涉及 *jdk* 的 bin 目录时，是指 /opt/jdk-17.0.4/bin 或 c:\Java\jdk-17.0.4\bin 目录。

可以如下测试安装是否成功。打开一个终端窗口，键入

```
javac --version
```

然后按回车键。应该能看到显示以下信息：

```
javac 17.0.4
```

如果得到诸如“javac: command not found”(javac：命令未找到)或“The name specified is not recognized as an internal or external command, operable program or batch file”(指定名不是一个内部或外部命令、可执行程序或批文件)的信息，则需要仔细检查安装。

在 Windows 或 Linux 上安装 JDK 时，还需要完成另外一个步骤：将 *jdk* 的 bin 目录添加到可执行路径中——可执行路径是操作系统查找可执行文件时所遍历的目录列表。

- 在 Linux 中，需要在 ~/.bashrc 或 ~/.bash\_profile 文件的最后增加这样一行：

```
export PATH=~/jdk/bin:$PATH
```

一定要使用正确的 JDK 路径，如 /opt/jdk-17.0.4。

- 在 Windows 10 中，在 Windows Settings (Windows 设置) 的搜索栏中键入 environment (环境)，选择 Edit environment variables for your account (编辑账户的环境变量，参见图 2-1)。会出现一个 Environment Variables (环境变量) 对话框。(它可能隐藏在 Windows 设置对话框后面。如果实在找不到，可以同时按住 Windows 和 R 键打开 Run (运行) 对话框，从这个对话框运行 sysdm.cpl，然后选择 Advanced (高级) 标签页，再单击 Environment Variables (环境变量) 按钮。) 在 User Variables (用户变量) 列表中找到并选择一个名为 Path 的变量。单击 Edit (编辑) 按钮，再单击 New (新建) 按钮，增加一个变量，值为 *jdk* 的 bin 目录 (参见图 2-2)。

保存所做的设置。之后新打开的所有命令提示窗口都会有正确的路径。

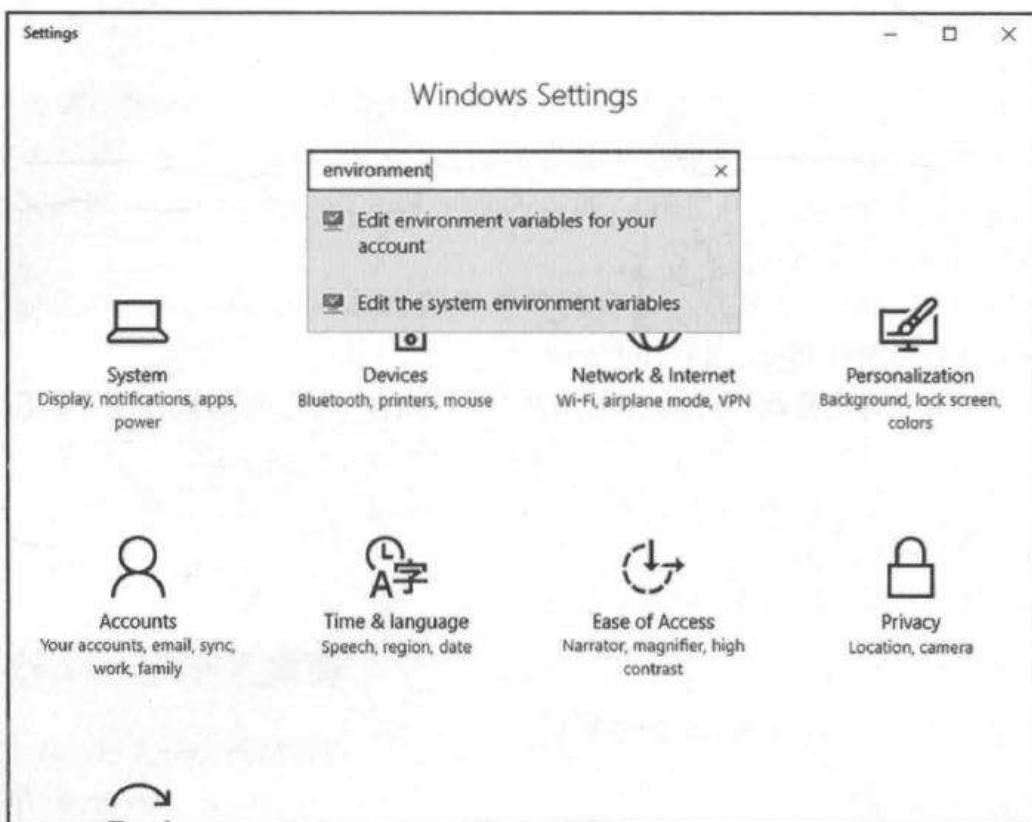


图 2-1 在 Windows 10 中设置系统属性

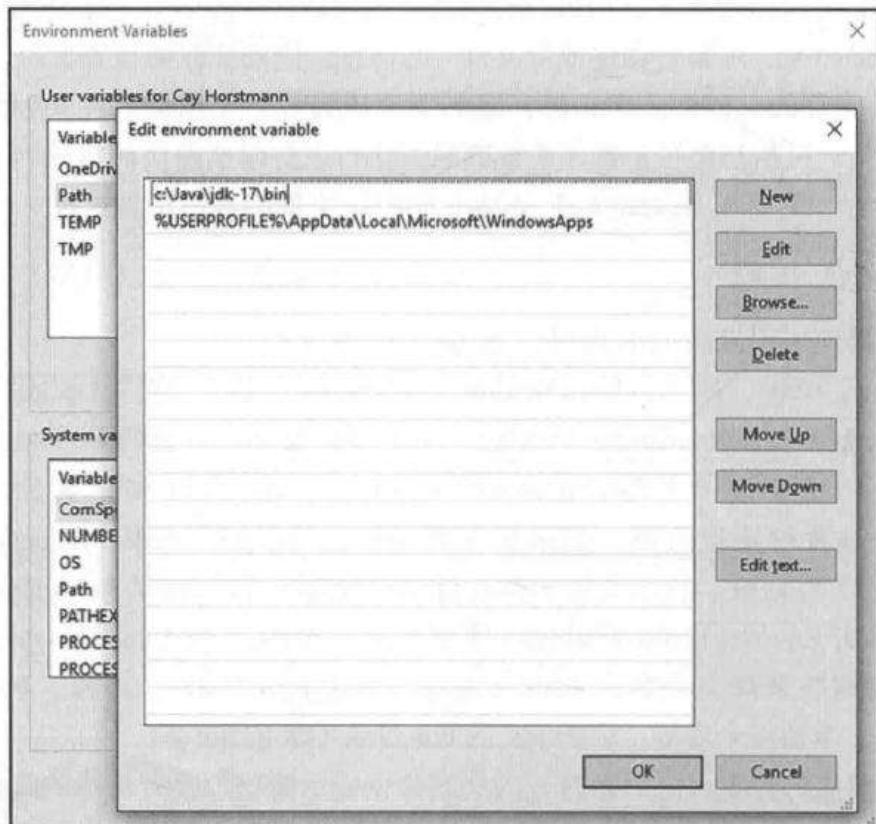


图 2-2 在 Windows 10 中设置 Path 环境变量

### 2.1.3 安装源文件和文档

类库源文件在 JDK 中以压缩文件 lib/src.zip 的形式发布，解压缩这个文件来得到源代码。为此只需完成以下步骤：

1. 确保 JDK 已经安装，而且 jdk/bin 目录在可执行路径中。
2. 在主目录中创建一个目录 javasrc。如果愿意，可以从一个终端窗口创建这个目录。  
mkdir javasrc
3. 在 jdk/lib 目录下找到文件 src.zip。
4. 将 src.zip 文件解压缩到 javasrc 目录。在一个终端窗口中，可以执行以下命令：

```
cd javasrc  
jar xvf jdk/lib/src.zip  
cd ..
```



**提示：**src.zip 文件中包含了所有公共类库的源代码。要想获得更多的源代码（例如编译器、虚拟机、原生方法以及私有辅助类的源代码），请访问网站 <http://openjdk.java.net>。

文档包含在独立于 JDK 的一个压缩文件中。可以直接从网站 [www.oracle.com/technetwork/java/javase/downloads](http://www.oracle.com/technetwork/java/javase/downloads) 下载文档。步骤如下：

1. 下载文档压缩文件。这个文件名为 jdk-17.0.x\_doc-all.zip。
2. 解压缩这个文件，将 doc 目录重命名为一个更有描述性的名字，如 javadoc。如果愿意，可以从命令行完成这个工作：

```
jar xvf Downloads/jdk-17.0.x_doc-all.zip  
mv docs jdk-17-docs
```

3. 在浏览器中导航到 jdk-17-docs/index.html，将这个页面增加到书签。

还要安装本书的程序示例。可以从 <http://horstmann.com/corejava> 下载示例。这些程序打包在一个 zip 文件 corejava.zip 中。可以将程序解压缩到主目录。它们会放在目录 corejava 中。如果愿意，可以从命令行完成这个工作：

```
jar xvf Downloads/corejava.zip
```

## 2.2 使用命令行工具

如果以前有过使用 Microsoft Visual Studio 等开发环境编程的经验，你可能会习惯于开发系统有一个内置的文本编辑器、用于编译和启动程序的菜单以及一个调试工具。JDK 完全没有这些功能。所有工作都要在终端窗口中通过键入命令来完成。这看起来很麻烦，不过确实是一个基本技能。第一次安装 Java 时，希望在安装开发环境之前先检查 Java 的安装是否正确。另外，通过执行这些基本步骤，可以更好地理解决发环境在后台的工作。

不过，掌握了编译和运行 Java 程序的基本步骤之后，你可能就会希望使用专业的开发环境。下一节会介绍如何使用开发环境。

首先介绍比较难的方法：从命令行编译并运行 Java 程序。

1. 打开一个终端窗口。
2. 进入 corejava/vlch02/Welcome 目录（corejava 是安装本书示例源代码的目录，请参见 2.1.3 节的解释）。
3. 键入下面的命令：

```
javac Welcome.java
java Welcome
```

然后，将会在终端窗口中看到图 2-3 所示的输出。

```
Terminal
$ cd corejava/vlch02/Welcome
~/corejava/vlch02/Welcome$ javac Welcome.java
~/corejava/vlch02/Welcome$ java Welcome
Welcome to Core Java!
~/corejava/vlch02/Welcome$
```

图 2-3 编译并运行 Welcome.java

祝贺你！你已经编译并运行了第一个 Java 程序。

那么，刚才都发生了什么？javac 程序是一个 Java 编译器，它将文件 Welcome.java 编译成 Welcome.class。java 程序启动 Java 虚拟机，虚拟机执行编译器编译到类文件中的字节码。

Welcome 程序非常简单，它只是向终端输出了一条消息。你可能想查看这个程序中的代码，如程序清单 2-1 所示。下一章中将解释它是如何工作的。

#### 程序清单 2-1 Welcome/Welcome.java

```
1 /**
2  * This program displays a greeting for the reader.
3  * @version 1.30 2014-02-27
4  * @author Cay Horstmann
5 */
6 public class Welcome
7 {
8     public static void main(String[] args)
9     {
10         String greeting = "Welcome to Core Java!";
11         System.out.println(greeting);
12         for (int i = 0; i < greeting.length(); i++)
13             System.out.print "=";
14         System.out.println();
15     }
16 }
```

在使用集成开发环境的年代，许多程序员对于在终端窗口中运行程序已经很生疏了。常常会出现很多错误，最后得到令人沮丧的结果。

一定要注意以下几点：

- 如果手动输入源程序，一定要确保正确地输入大小写。例如，类名为 `Welcome`，而不是 `welcome` 或 `WELCOME`。
- 编译器需要一个文件名 (`Welcome.java`)，而运行程序时，只需要指定类名 (`Welcome`)，不要带扩展名 `.java` 或 `.class`。
- 如果看到诸如“Bad command or file name”或“javac:command not found”之类的消息，就要返回去仔细检查安装是否有问题，特别是可执行路径的设置。
- 如果 `javac` 报告了一个错误，指出无法找到 `Welcome.java`，就应该检查目录中是否存在这个文件。

在 Linux 环境下，检查 `Welcome.java` 是否正确地以大写字母开头。

在 Windows 环境下，使用命令 `dir`，而不要使用图形化资源管理器工具。有些文本编辑器（特别是 Notepad）会在每个文件名后面添加扩展名 `.txt`。如果使用 Notepad 编辑 `Welcome.java`，实际上会把它保存为 `Welcome.java.txt`。如果采用默认的 Windows 设置，资源管理器会与 Notepad “勾结”，隐藏 `.txt` 扩展名，因为这属于“已知的文件类型”。对于这种情况，需要使用命令 `ren` 重新命名这个文件，或是另存一次，在文件名两边加双引号，如：“`"Welcome.java"`”。

- 运行程序之后，如果收到关于 `java.lang.NoClassDefFoundError` 的错误消息，就应该仔细检查出问题的类名。

如果收到关于 `welcome` (`w` 为小写) 的错误消息，就应该重新执行命令：`java Welcome` (`W` 为大写)。记住，Java 区分大小写。

如果收到有关 `Welcome/java` 的错误信息，这说明你错误地键入了 `java Welcome.java`，应该重新执行命令 `java Welcome`。

- 如果键入 `java Welcome`，而虚拟机没有找到 `Welcome` 类，就应该检查是否有人设置了系统的 `CLASSPATH` 环境变量（不提倡全局设置这个变量，不过 Windows 中有些比较差的软件安装程序确实会这样做）。可以像设置 `Path` 环境变量一样设置 `CLASSPATH`，不过这里将删除这个设置。

 提示：在 <http://docs.oracle.com/javase/tutorial/getStarted/cupojava/> 上有一个很好的教程，其中更详细地介绍了初学者容易犯的一些错误。

 注释：如果只有一个源文件，可以不执行 `javac` 命令。这个特性是为了支持 shell 脚本（以“shebang”行 `#!/path/to/java` 开头的脚本），可能也用于简单的学生程序。一旦程序变得更复杂，就需要使用 `javac` 命令了。

这个 `Welcome` 程序没有太多意思。接下来再来尝试一个图形化应用。这个程序是一个简单的图像文件查看器，可以加载和显示一个图像。与前面一样，从命令行编译和运行这个程序。

1. 打开一个终端窗口。
2. 切换到目录 corejava/v1ch02/ImageViewer。
3. 输入以下命令：

```
javac ImageViewer.java
java ImageViewer
```

会弹出一个新的程序窗口（ImageViewer 应用）。现在选择 File → Open，找到一个要打开的图像文件。（这个目录下有两个示例文件。）然后会显示这个文件（参见图 2-4）。要关闭这个程序，可以单击标题栏上的关闭钮，或者从菜单选择 File → Exit。

可以简单浏览这个源代码（程序清单 2-2）。这个程序比第一个程序长多了，但只要想一想用 C 或 C++ 编写类似功能的应用程序所需要的代码量，就不会觉得它复杂了。当然，如今编写带图形用户界面的桌面应用并不常见，不过，如果你感兴趣，可在第 10 章学习更多详细内容。

#### 程序清单 2-2 ImageViewer/ImageViewer.java

```
1 import java.awt.*;
2 import java.io.*;
3 import javax.swing.*;
4
5 /**
6  * A program for viewing images.
7  * @version 1.31 2018-04-10
8  * @author Cay Horstmann
9 */
10 public class ImageViewer
11 {
12     public static void main(String[] args)
13     {
14         EventQueue.invokeLater(() ->
15         {
16             var frame = new ImageViewerFrame();
17             frame.setTitle("ImageViewer");
18             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19             frame.setVisible(true);
20         });
21     }
22 }
23
24 /**
25  * A frame with a label to show an image.
26 */
27 class ImageViewerFrame extends JFrame
```

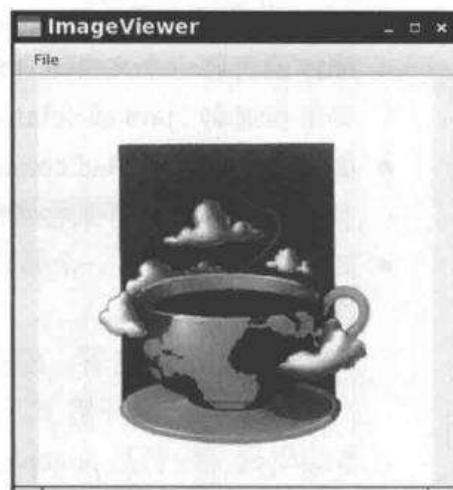


图 2-4 运行 ImageViewer 应用

```
28 {
29     private static final int DEFAULT_WIDTH = 300;
30     private static final int DEFAULT_HEIGHT = 400;
31
32     public ImageViewerFrame()
33     {
34         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
35
36         // use a label to display the images
37         var label = new JLabel();
38         add(label);
39
40         // set up the file chooser
41         var chooser = new JFileChooser();
42         chooser.setCurrentDirectory(new File("."));
43
44         // set up the menu bar
45         var menuBar = new JMenuBar();
46         setJMenuBar(menuBar);
47
48         var menu = new JMenu("File");
49         menuBar.add(menu);
50
51         var openItem = new JMenuItem("Open");
52         menu.add(openItem);
53         openItem.addActionListener(event ->
54         {
55             // show file chooser dialog
56             int result = chooser.showOpenDialog(null);
57
58             // if file selected, set it as icon of the label
59             if (result == JFileChooser.APPROVE_OPTION)
60             {
61                 String name = chooser.getSelectedFile().getPath();
62                 label.setIcon(new ImageIcon(name));
63             }
64         });
65
66         var exitItem = new JMenuItem("Exit");
67         menu.add(exitItem);
68         exitItem.addActionListener(event -> System.exit(0));
69     }
70 }
```

## 2.3 使用集成开发环境

上一节中，你已经了解了如何从命令行编译和运行一个 Java 程序。这是一个很有用的排错技能，不过对于大多数日常工作来说，还是应当使用集成开发环境。这些环境非常强大，也很方便，不使用这些集成开发环境简直有些不合情理。我们可以免费得到一些很棒的开发环境，如 Eclipse、IntelliJ IDEA 和 NetBeans。这一章中，我们将学习如何从 Eclipse 起步。

当然，如果你喜欢其他开发环境，学习本书时也完全可以使用你喜欢的环境。

首先从网站 <http://eclipse.org/downloads> 下载 Eclipse。Eclipse 提供了面向 Linux、Mac OS X 和 Windows 的版本。运行安装程序，并选择“Eclipse IDE for Java Developers”。

下面是用 Eclipse 编写程序的一般步骤：

1. 启动 Eclipse 之后，从菜单选择 File → New → Project。

2. 从向导对话框中选择“Java Project”（如图 2-5 所示）。

3. 单击 Next 按钮，不选中“Use default location”复选框。单击 Browse 导航到 corejava/v1ch02/Welcome 目录（见图 2-6）。

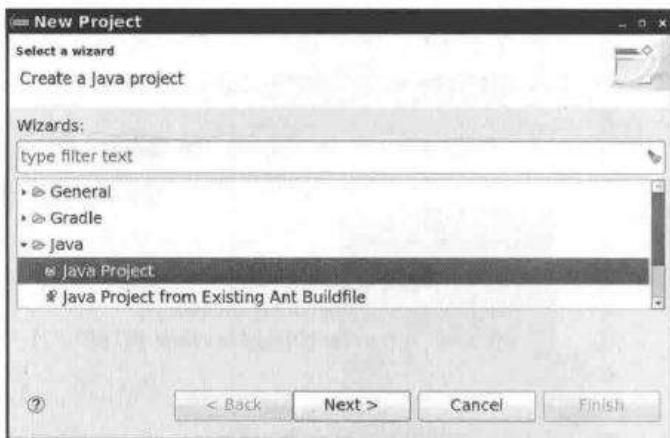


图 2-5 Eclipse 中的 New Project (新建项目) 对话框

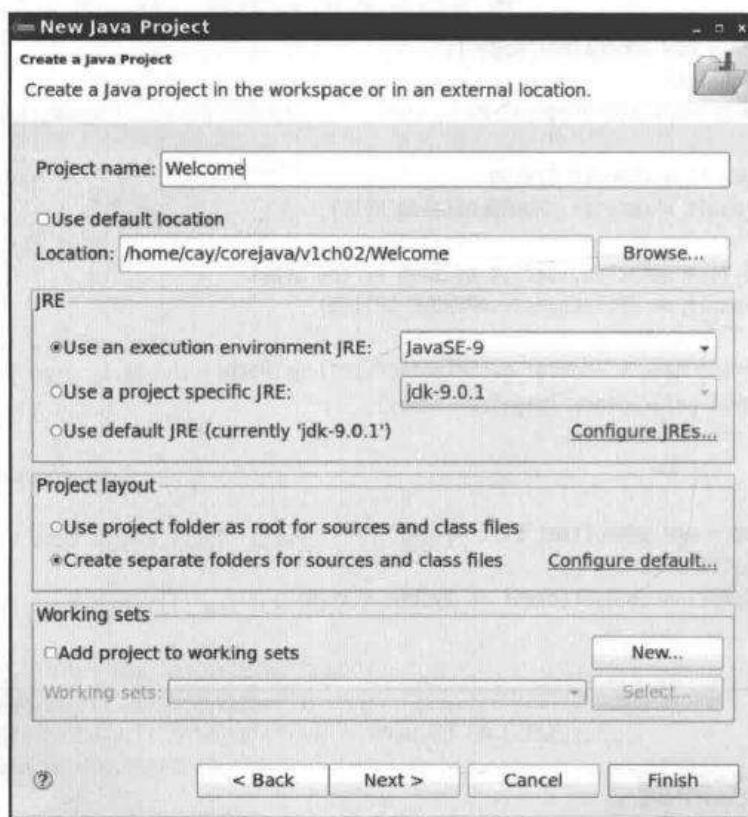


图 2-6 配置 Eclipse 项目

4. 单击 Finish 按钮。这样就创建了这个项目。

5. 单击项目左边窗格中的三角，直到找到 Welcome.java 并双击这个文件。现在应该会看到

一个包含程序代码的窗格（如图 2-7 所示）。

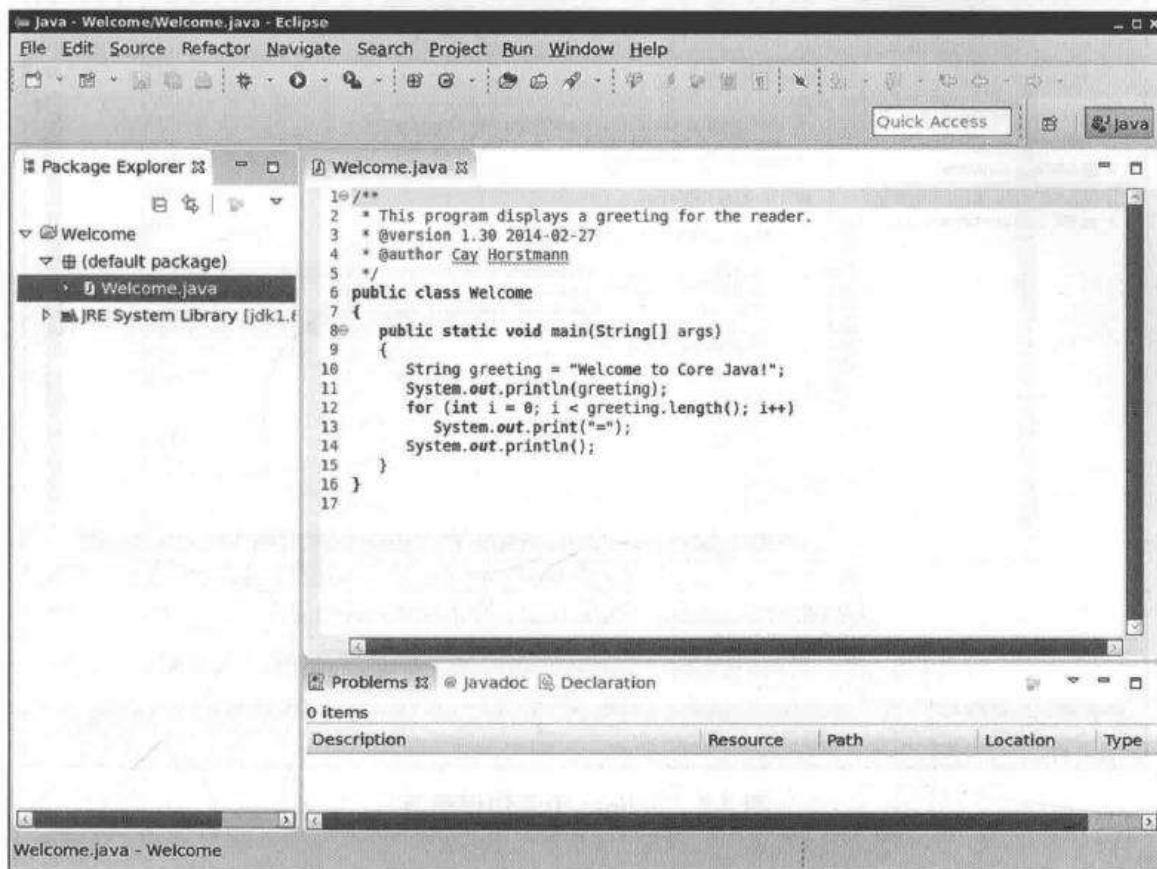


图 2-7 使用 Eclipse 编辑源文件

6. 用鼠标右键单击左侧窗格中的项目名 (Welcome)，选择 Run → Run As → Java Application。程序输出会显示在控制台窗格中。

之前我们假定这个程序没有输入错误或 bug (毕竟，这里只有几行代码)。不过为了说明问题，假设代码中意外包含一个录入错误 (或者甚至一个语法错误)。试着修改原来的程序，例如，故意将 String 的大小写弄错：

```
string greeting = "Welcome to Core Java!";
```

注意图 2-8 中 string 下面的波浪线。单击源代码下面的 Problems 标签页，展开小三角，直到看到一个错误消息指出有一个未知的 string 类型 (见图 2-8)。单击这个错误消息。光标会移到编辑窗格中相应的代码行，可以在这里纠正错误。利用这个特性可以快速地修正错误。



**提示：**通常，Eclipse 错误报告会伴随一个灯泡图标。单击灯泡图标可以得到修正这个错误的一组建议方案。

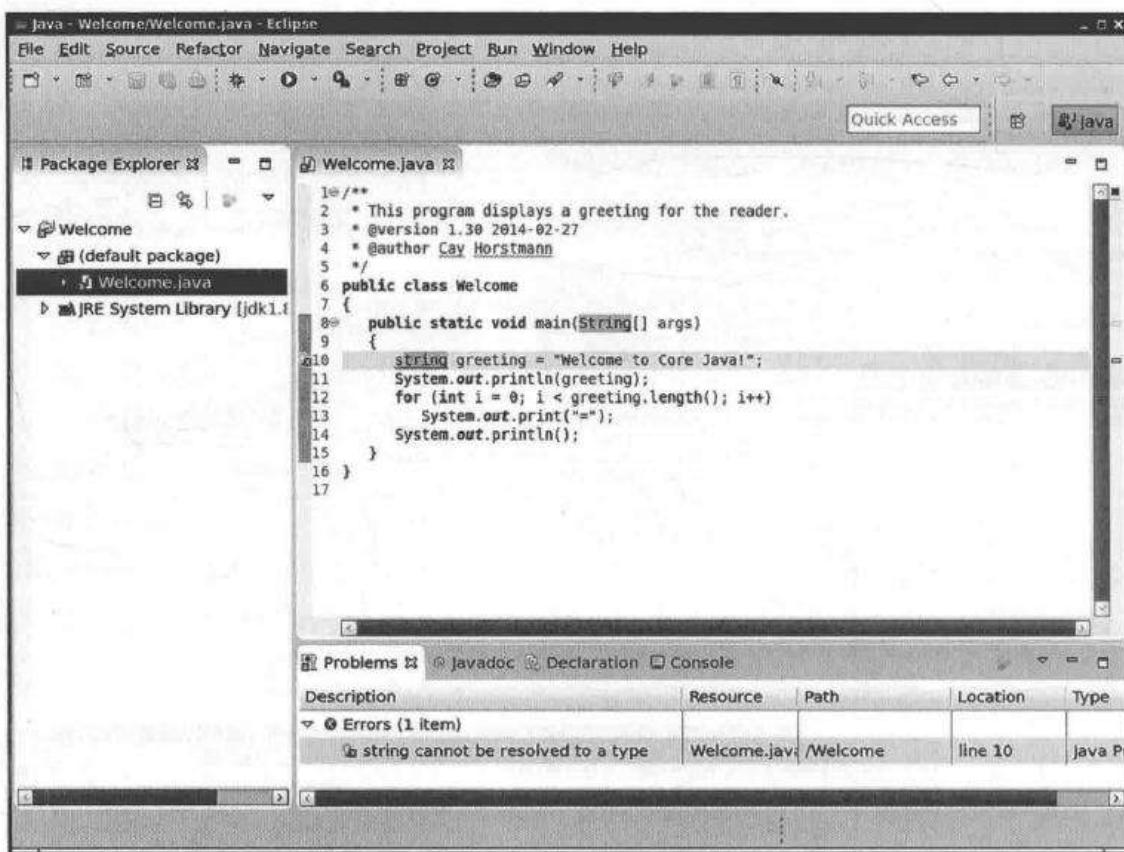


图 2-8 Eclipse 中的错误消息

## 2.4 JShell

上一节中，你已经看到了如何编译和运行一个 Java 程序。Java 9 引入了另一种使用 Java 的方法。JShell 程序提供了一个“读取 – 评估 – 打印循环”（Read-Evaluate-Print Loop, REPL）。键入一个 Java 表达式，JShell 会评估输入，打印结果，并等待下一个输入。

要启动 JShell，只需要在终端窗口中键入 `jshell`（参见图 2-9）。

JShell 首先显示一个问候语，后面是一个提示符：

```
| Welcome to JShell -- Version 17.0.4
| For an introduction type: /help intro
```

```
jshell>
```

现在键入一个表达式，如下：

```
"Core Java".length()
```

JShell 会回应一个结果——在这里就是字符串“Core Java”中的字符个数。

```
$1 ==> 9
```

注意，你并没有键入 `System.out.println`。JShell 会自动打印你输入的每一个表达式的值。

```

Terminal ~$ 
~$ jshell
| Welcome to JShell -- Version 17
| For an introduction type: /help intro

jshell> "Core Java".length()
$1 ==> 9

jshell> 5 * $1 - 3
$2 ==> 42

jshell> int answer = 6 * 7
answer ==> 42

jshell> Math.
E           IEEEremainder(   PI           abs(
absExact(    acos(        addExact(    asin(
atan(        atan2(       cbrt(       ceil(
class        copySign(    cos(         cosh(
decrementExact( exp(        expm1(      floor(
floorDiv(    floorMod(   fma(        getExponent(
hypot(       incrementExact( log(        log10(
log1p(       max(        min(        multiplyExact(
multiplyFull( multiplyHigh( negateExact( nextAfter(
nextDown(    nextUp(     pow(        random(
rint(        round(      scalb(      signum(
sin(         sinh(       sqrt(        subtractExact(
tan(         tanh(       toDegrees( ToIntExact(
toRadians(   ulp(        toRadians(  )
jshell> Math.

```

图 2-9 运行 JShell

输出中的 \$1 表示这个结果可以用于进一步的计算。例如，如果你键入：

5 \* \$1 - 3

就会得到：

\$2 ==> 42

如果需要多次使用一个变量，可以给它指定一个容易记忆的名字。一定要遵循 Java 语法（我们将在第 3 章介绍语法），指定类型，然后指定变量名。例如，

```
jshell> int answer = 6 * 7
answer ==> 42
```

另一个有用的特性是“tab 补全”。如果键入：

Math.

然后再按一次 Tab 键。你会得到用 Math 类调用的所有方法的一个列表：

```
jshell> Math.
E           IEEEremainder(   PI           abs(
absExact(    acos(        addExact(    asin(
atan(        atan2(       cbrt(       ceil(
class        copySign(    cos(         cosh(
decrementExact( exp(        expm1(      floor(
floorDiv(    floorMod(   fma(        getExponent(
hypot(       incrementExact( log(        log10(
log1p(       max(        min(        multiplyExact(
multiplyFull( multiplyHigh( negateExact( nextAfter(
nextDown(    nextUp(     pow(        random(
rint(        round(      scalb(      signum(
sin(         sinh(       sqrt(        subtractExact(
tan(         tanh(       toDegrees( ToIntExact(
toRadians(   ulp(        toRadians(  )
```

现在键入 `l`, 然后再按一次 Tab 键。方法名会补全为 `log`, 现在你会得到一个比较小的列表:

```
jshell> Math.log  
log( log10( log1p(
```

接下来你可以手动填入其余的代码:

```
jshell> Math.log10(0.001)  
$3 ==> -3.0
```

要重复运行一个命令, 可以连续按↑键, 直到看到想要重新运行或编辑的命令行。可以用←和→键移动命令行中的光标位置, 然后增加或删除字符。编辑完命令后再按回车键。例如, 把命令行中的 `0.001` 替换为 `1000`, 然后按回车键:

```
jshell> Math.log10(1000)  
$4 ==> 3.0
```

JShell 会让 Java 语言和类库的学习变得轻松而有趣, 它不要求你启动一个庞大的开发环境, 不会让你再为 `public static void main` 而困扰。

在本章中, 我们学习了编译和运行 Java 程序的机制。现在可以进入第 3 章开始学习 Java 语言了。

# 第3章 Java 的基本程序设计结构

- ▲ 一个简单的 Java 程序
- ▲ 注释
- ▲ 数据类型
- ▲ 变量与常量
- ▲ 运算符

- ▲ 字符串
- ▲ 输入与输出
- ▲ 控制流程
- ▲ 大数
- ▲ 数组

现在，你应该已经成功地安装了 JDK，并且能够执行第 2 章中的示例程序。下面开始介绍程序设计。本章主要介绍如何在 Java 中实现基本程序设计概念（如数据类型、分支以及循环）。

## 3.1 一个简单的 Java 程序

下面仔细分析一个最简单的 Java 程序，它只是向控制台打印一个消息：

```
public class FirstSample
{
    public static void main(String[] args)
    {
        System.out.println("We will not use 'Hello, World!'");
    }
}
```

这个程序虽然很简单，但这些内容在所有 Java 应用中都会出现，因此还是值得花一些时间来研究的。首先，Java 区分大小写。如果出现了大小写拼写错误（例如，将 main 拼写成 Main），程序将无法运行。

下面逐行地查看这段源代码。关键字 `public` 称为访问修饰符（access modifier），这些修饰符用于控制程序的其他部分对这段代码的访问级别。在第 5 章中将会更详细地介绍访问修饰符的有关内容。关键字 `class` 表明 Java 程序中的全部内容都包含在类中。下一章你会更多地了解 Java 类，不过现在只需要将类看作是程序逻辑的一个容器，定义了应用程序的行为。正如第 1 章所述，类是所有 Java 应用的构建模块。Java 程序中的所有内容都必须放在类中。

关键字 `class` 后面紧跟类名。Java 中定义类名的规则很宽松。类名必须以字母开头，后面可以跟字母和数字的任意组合。长度基本上没有限制。但是不能使用 Java 保留字（例如，`public` 或 `class`）作为类名（保留字列表请参见附录）。

标准命名约定为：类名是以大写字母开头的名词（类名 `FirstSample` 就使用了这个命名约定）。如果名字由多个单词组成，每个单词的第一个字母都应该大写。这种在一个单词

中间使用大写字母的方式有时称为骆驼命名法 (camel case)。以其自身为例，应该写为 CamelCase。

源代码的文件名必须与公共类的类名相同，并用.java 作为扩展名。因此，存储这个代码时，文件名必须为 FirstSample.java（再次提醒大家注意，大小写非常重要，千万不能写成 firstsample.java）。

如果已经正确地命名文件，并且源代码中没有任何录入错误，在编译这个源代码之后，会得到一个包含这个类字节码的文件。Java 编译器将这个字节码文件自动地命名为 FirstSample.class，并存储在源文件所在的同一个目录下。最后，使用下面这个命令运行这个程序：

```
java FirstSample
```

（请记住，不要加.class 扩展名。）程序执行之后，控制台上将会显示“*We will not use 'Hello,World'!*”。

当使用以下命令

```
java ClassName
```

运行一个已编译的程序时，Java 虚拟机总是从指定类中 main 方法的代码开始执行（这里的“方法”就是 Java 中对“函数”的叫法），因此为了能够执行代码，类的源代码中必须包含一个 main 方法。当然，也可以将你自己的方法添加到类中，并从 main 方法调用这些方法（第 4 章将介绍如何编写你自己的方法）。

**注释：**根据 Java 语言规范，main 方法必须声明为 public（Java 语言规范是描述 Java 语言的官方文档。可以从网站 <http://docs.oracle.com/javase/specs> 阅读或下载）。

不过，即使 main 方法没有声明为 public，有些版本的 Java 解释器也会执行 Java 程序。有个程序员报告了这个 bug。如果感兴趣，可以访问 <https://bugs.openjdk.java.net/browse/JDK-4252539> 查看这个 bug。1999 年，这个 bug 被标记为“关闭，不予修复”（Closed, Will not be fixed）。Sun 公司的一个工程师解释说：Java 虚拟机规范并没有强制要求 main 方法一定是 public，并且“修复这个 bug 有可能带来其他的隐患”。好在，这个问题最终得到了解决。在 Java 1.4 及以后的版本中，Java 解释器强制要求 main 方法必须是 public。

当然，让质量保证工程师对 bug 报告做出决定不仅让人生疑，也让他们自己很头疼，因为他们的工作量很大，而且他们对 Java 的所有细节也未必了解得很清楚。不过，Sun 公司在 Java 开源很久以前就把 bug 报告及其解决方案放在网站上让所有人监督检查，这是一个非常了不起的举措。

注意源代码中的大括号 {}。在 Java 中，像在 C/C++ 中一样，用大括号划分程序的各个部分（通常称为块）。Java 中任何方法的代码都必须以“{”开始，用“}”结束。

大括号的使用风格曾经引发过许多无意义的争论。我们的习惯是把匹配的大括号对齐。不过，由于 Java 编译器会忽略空白符，所以你可以选用自己喜欢的任何大括号风格。

我们暂且不考虑关键字 `static void`，只把它们当作编译 Java 程序必要的部分就行了。在学习完第 4 章后，这些部分的作用就会揭晓。现在需要记住的重点是：每个 Java 应用都必须有一个 `main` 方法，其声明格式如下所示：

```
public class ClassName
{
    public static void main(String[] args)
    {
        program statements
    }
}
```

**C++ 注释：**作为一名 C++ 程序员，你一定知道类是什么。Java 的类与 C++ 的类很相似，但有些差异还是会使人感到困惑。例如，Java 中的所有函数都是某个类的方法（标准术语将其称为方法，而不是成员函数）。因此，Java 中的 `main` 方法必须有一个外壳（shell）类。你可能对 C++ 中的静态成员函数（static member function）也很熟悉。它们是类中定义的成员函数，而且不对对象进行操作。Java 中的 `main` 方法总是静态的。最后，与 C/C++ 一样，关键字 `void` 表示这个方法不返回值，但与 C/C++ 不同的是，`main` 方法不会为操作系统返回一个“退出码”。如果 `main` 方法正常退出，那么 Java 程序的退出码为 0，表示成功地运行了程序。如果要以其他退出码终止程序，则需要使用 `System.exit` 方法。

来看以下代码片段：

```
{
    System.out.println("We will not use 'Hello, World!'");
}
```

一对大括号表示方法体的开始与结束，这个方法中只包含一条语句。与大多数程序设计语言一样，可以将 Java 语句看成是这个语言中的句子。在 Java 中，每个语句必须用分号结束。特别需要说明，回车不是语句的结束标志，因此，如果需要，一条语句可以跨多行。

这个 `main` 方法体中只包含一条语句，其功能是将一个文本行输出到控制台。

在这里，我们使用 `System.out` 对象并调用了它的 `println` 方法。注意，点号（.）用于调用方法。Java 使用的通用语法是

*object.method(parameters)*

这等价于一个函数调用。

在这个示例中，`println` 方法接收一个字符串参数。这个方法将这个字符串参数显示在控制台上。然后，终止这个输出行，所以每次调用 `println` 都会在新的一行上显示输出。需要注意一点，Java 与 C/C++ 一样，都使用双引号界定字符串。（本章稍后会介绍更多有关字符串的知识。）

与其他程序设计语言中的函数一样，Java 中的方法可以没有参数，也可以有一个或多个参数（有的程序员把参数叫作实参（argument））。即使一个方法没有参数，也需要使用空括号。例如，不带参数的 `println` 方法只打印一个空行。可以使用下面的语句来调用：

```
System.out.println();
```

**注释：**System.out 还有一个 print 方法，它不在输出之后增加换行符。例如，System.out.print("Hello") 打印 "Hello" 之后不换行，下一个输出将紧跟在字母 "o" 之后。

## 3.2 注释

与大多数程序设计语言一样，Java 中的注释不会出现在可执行程序中。因此，可以在源程序中根据需要添加任意多的注释，而不必担心代码膨胀。在 Java 中，有 3 种标记注释的方式。最常用的方式是使用 //。使用这种方式时，从 // 开始到本行结尾都是注释。

```
System.out.println("We will not use 'Hello, World!'"); // is this too cute?
```

当需要更长的注释时，可以在每一行注释的前面加 //，或者也可以使用 /\* 和 \*/ 注释界定符将一段比较长的注释括起来。

最后，第 3 种注释可以用来自动生成文档。这种注释以 /\*\* 开始，以 \*/ 结束。在程序清单 3-1 中可以看到这种注释。有关这种注释以及自动生成文档的更多内容请参见第 4 章。

程序清单 3-1 FirstSample/FirstSample.java

```
1 /**
2  * This is the first sample program in Core Java Chapter 3
3  * @version 1.01 1997-03-22
4  * @author Gary Cornell
5 */
6 public class FirstSample
7 {
8     public static void main(String[] args)
9     {
10         System.out.println("We will not use 'Hello, World!'");
11     }
12 }
```

**警告：**在 Java 中，/\* \*/ 注释不能嵌套。也就是说，不能简单地把代码用 /\* 和 \*/ 括起来作为注释，因为这段代码本身可能包含一个 \*/ 界定符。

## 3.3 数据类型

Java 是一种强类型语言。这就意味着必须为每一个变量声明一个类型。在 Java 中，一共有 8 种基本类型（primitive type），其中有 4 种整型、2 种浮点类型、1 种字符类型 char（用于表示 Unicode 编码的代码单元，请参见 3.3.3 节“char 类型”）和 1 种用于表示真值的 boolean 类型。

**注释：**Java 有一个能够表示任意精度的算术包，所谓的“大数”（big number）是 Java 对象，而不是一个基本 Java 类型。本章稍后将会详细地介绍如何使用大数。

### 3.3.1 整型

整型用于表示没有小数部分的数，可以是负数。Java 提供了 4 种整型，如表 3-1 所示。

表 3-1 Java 整型

类 型	存储需求	取值范围
int	4 字节	-2 147 483 648 ~ 2 147 483 647 (略高于 20 亿)
short	2 字节	-32 768 ~ 32 767
long	8 字节	-9 223 372 036 854 775 808 ~ 9 223 372 036 854 775 807
byte	1 字节	-128 ~ 127

在通常情况下，int 类型最常用。但如果想要表示整个地球的居住人口，就需要使用 long 类型了。byte 和 short 类型主要用于特定的应用场合，例如，底层的文件处理或者存储空间有限时的大数组。

在 Java 中，整型的范围与运行 Java 代码的机器无关。这就解决了软件从一个平台移植到另一个平台时（或者甚至在同一个平台中不同操作系统之间移植时）让程序员头疼的主要问题。与此相反，C 和 C++ 程序会针对不同的处理器选择最高效的整型，这样一来，一个在 32 位处理器上运行得很好的 C 程序在 16 位系统上运行时可能会发生整数溢出。由于 Java 程序必须保证在所有机器上都能够得到相同的运行结果，所以各种数据类型的取值范围是固定的。

长整型数值有一个后缀 L 或 l（如 4000000000L）。十六进制数值有一个前缀 0x 或 0X（如 0xCAFE）。八进制有一个前缀 0（例如，010 对应十进制中的 8）。显然，八进制表示法比较容易混淆，所以很少有程序员使用八进制常数。

加上前缀 0b 或 0B 还可以写二进制数。例如，0b1001 就是 9。另外，可以为数字字面量加下画线，如用 1\_000\_000（或 0b1111\_0100\_0010\_0100\_0000）表示 100 万。这些下画线只是为了让人更易读。Java 编译器会去除这些下画线。

**C++ 注释：**在 C 和 C++ 中，int 和 long 等类型的大小与目标平台相关。在 8086 这样的 16 位处理器上，整数占 2 字节；不过，在 32 位处理器上（比如 Pentium 或 SPARC），整数则为 4 字节。类似地，在 32 位处理器上 long 值为 4 字节，在 64 位处理器上则为 8 字节。由于存在这些差别，这给编写跨平台程序带来了很大难度。在 Java 中，所有数值类型的大小都与平台无关。

注意，Java 没有无符号（unsigned）形式的 int、long、short 或 byte 类型。

**注释：**如果使用不可能为负的整数值而且确实需要额外的一位（bit），也可以把有符号整数值解释为无符号数，但是要非常仔细。例如，一个 byte 值 b 可以不表示 -128 ~ 127 的范围，如果你想表示 0 ~ 255 的范围，也可以存储在一个 byte 中。基于二进制算术运算的性质，只要不溢出，加法、减法和乘法都能正常计算。但对于其他运算，需要调用 Byte.toUnsignedInt(b) 来得到一个 0 ~ 255 的 int 值，然后处理这个整数值，再把它转换回 byte。Integer 和 Long 类都提供了处理无符号除法和求余数的方法。

### 3.3.2 浮点类型

浮点类型用于表示有小数部分的数值。在 Java 中有两种浮点类型，如表 3-2 所示。

表 3-2 浮点类型

类 型	存储需求	取值范围
float	4 字节	大约 $\pm 3.402\ 823\ 47 \times 10^{38}$ (6 ~ 7 位有效数字)
double	8 字节	大约 $\pm 1.797\ 693\ 134\ 862\ 315\ 70 \times 10^{308}$ (15 位有效数字)

`double` 表示这种类型的数值精度是 `float` 类型的两倍（有人称之为双精度数（double-precision））。很多情况下，`float` 类型的精度（6 ~ 7 位有效数字）都不能满足需求。实际上，只有很少的情况适合使用 `float` 类型，例如，所使用的库需要单精度数，或者需要存储大量单精度数时。

`float` 类型的数值有一个后缀 F 或 f（例如，`3.14F`）。没有后缀 F 的浮点数值（如 `3.14`）总是默认为 `double` 类型。可选地，也可以在 `double` 数值后面添加后缀 D 或 d（例如，`3.14D`）。

**注释：**可以使用十六进制表示浮点数字面量。例如， $0.125 = 2^{-3}$  可以写为 `0x1.0p-3`。在十六进制表示法中，使用 p 表示指数，而不是 e。（e 是一个十六进制数位。）注意，尾数采用十六进制，指数采用十进制。指数的基数是 2，而不是 10。

所有的浮点数计算都遵循 IEEE 754 规范。具体来说，有 3 个特殊的浮点数值表示溢出和出错情况：

- 正无穷大
- 负无穷大
- NaN (不是一个数)

例如，一个正整数除以 0 的结果为正无穷大。计算 `0/0` 或者负数的平方根结果为 NaN。

**注释：**常量 `Double.POSITIVE_INFINITY`、`Double.NEGATIVE_INFINITY` 和 `Double.NaN`（以及相应的 `Float` 类型常量）分别表示这三个特殊的值，但在实际中很少用到。特别要说明的是，不能如下检测一个特定结果是否等于 `Double.NaN`：

```
if (x == Double.NaN) // is never true
```

所有 NaN 的值都认为是不相同的。不过，可以使用 `Double.isNaN` 方法来判断：

```
if (Double.isNaN(x)) // check whether x is "not a number"
```

**警告：**浮点数值不适用于无法接受舍入误差的金融计算。例如，命令 `System.out.println(2.0-1.1)` 将打印出 `0.8999999999999999`，而不是我们期望的 `0.9`。这种舍入误差的主要原因是浮点数值采用二进制表示，而在二进制系统中无法精确地表示分数 `1/10`。这就好像十进制无法精确地表示分数 `1/3` 一样。如果需要精确的数值计算，不允许有舍入误差，则应该使用 `BigDecimal` 类，本章稍后将介绍这个类。

### 3.3.3 char 类型

char 类型原本用于表示单个字符。不过，现在情况已经有所变化。如今，有些 Unicode 字符可以用一个 char 值描述，另外一些 Unicode 字符则需要两个 char 值。有关的详细信息请阅读下一节。

char 类型的字面量值要用单引号括起来。例如：'A' 是编码值为 65 的字符常量。它与 "A" 不同，"A" 是包含一个字符的字符串。char 类型的值可以表示为十六进制值，其范围从 \u0000 ~ \uFFFF。例如，\u2122 表示商标符号 (™)，\u03C0 表示希腊字母 π。

除了转义序列 \u 之外，还有一些用于表示特殊字符的转义序列，请见表 3-3。可以在加引号的字符字面量或字符串中使用这些转义序列。例如，'\u2122' 或 "Hello\n"。转义序列 \u 还可以在加引号字符常量或字符串之外使用（而其他所有转义序列不可以）。例如：

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

就是完全合法的，\u005B 和 \u005D 分别是 [ 和 ] 的编码。

表 3-3 特殊字符的转义序列

转义序列	名称	Unicode 值	转义序列	名称	Unicode 值
\b	退格	\u0008	\'	单引号	\u0027
\t	制表	\u0009	\\"	反斜线	\u005c
\n	换行	\u000a	\s	空格。在文本块中用来 保留末尾空白符	\u0020
\r	回车	\u000d	\newline	只在文本块中使用：连 接这一行和下一行	—
\f	换页	\u000c	—	—	—
\"	双引号	\u0022	—	—	—

◆ 警告：Unicode 转义序列会在解析代码之前处理。例如，"\u0022+\u0022" 并不是一个由引号 (U+0022) 包围加号构成的字符串。实际上，\u0022 会在解析之前转换为 "，这会得到 ""+"，也就是一个空串。

更隐秘地，一定要当心注释中的 \u。以下注释

```
// \u000a is a newline
```

会产生一个语法错误，因为读程序时 \u000a 会替换为一个换行符。类似地，下面这个注释

```
// look inside c:\users
```

也会产生一个语法错误，因为 \u 后面并没有跟着 4 位十六进制数。

### 3.3.4 Unicode 和 char 类型

要想弄清 char 类型，就必须了解 Unicode 编码机制，它打破了传统字符编码机制的限制。在 Unicode 出现之前，已经有许多种不同的标准：美国的 ASCII、西欧语言的 ISO 8859-

1、俄罗斯的 KOI-8、中国的 GB 18030 和 BIG-5 等。这样就产生了下面两个问题：一个是对一个特定的代码值，在不同的编码机制中可能对应不同的字母；二是采用大字符集的语言其编码长度有可能不同。例如，有些常用的字符采用单字节编码，而另外一些字符则需要两个或多个字节编码。

设计 Unicode 编码的目的就是要解决这些问题。在 20 世纪 80 年代开始启动统一工作时，人们认为两字节的代码宽度足以对世界上各种语言的所有字符进行编码，并有足够的空间留给未来扩展，当时所有人都这么想。在 1991 年发布了 Unicode 1.0，当时仅占用 65 536 个代码值中不到一半的部分。设计 Java 时决定采用 16 位的 Unicode 字符集，这比使用 8 位字符集的其他程序设计语言有了很大的改进。

遗憾的是，经过一段时间后，不可避免的事情发生了。Unicode 字符超过了 65 536 个，其主要原因是增加了汉语、日语和韩语中的大量表意文字。现在，16 位的 `char` 类型已经不足以描述所有 Unicode 字符了。

下面利用一些专用术语来解释 Java 语言是如何解决这个问题的。码点（code point）是指与一个编码表中的某个字符对应的代码值。在 Unicode 标准中，码点采用十六进制书写，并加上前缀 `U+`，例如 `U+0041` 就是拉丁字母 A 的码点。Unicode 的码点可以分成 17 个代码平面（code plane）。第一个代码平面称为基本多语言平面（basic multilingual plane），包括码点从 `U+0000` 到 `U+FFFF` 的“经典” Unicode 代码；其余的 16 个平面的码点从 `U+10000` 到 `U+10FFFF`，包含各种辅助字符（supplementary character）。

UTF-16 编码采用不同长度的代码表示所有 Unicode 码点。在基本多语言平面中，每个字符用 16 位表示，通常称为代码单元（code unit）；而辅助字符编码为一对连续的代码单元。采用这种编码对表示的每个值都属于基本多语言平面中未用的 2048 个值范围，通常称为替代区域（surrogate area）（`U+D800 ~ U+DBFF` 用于第一个代码单元，`U+DC00 ~ U+DFFF` 用于第二个代码单元）。这样设计十分巧妙，因为我们可以很快知道一个代码单元是一个字符的编码，还是一个辅助字符的第一或第二部分。例如，∅ 是八元数集 (<http://math.ucr.edu/home/baez/octonions>) 的数学符号，码点为 `U+1D546`，编码为两个代码单元 `U+D835` 和 `U+DD46`。（关于编码算法的具体描述见 <https://tools.ietf.org/html/rfc2781>。）

在 Java 中，`char` 类型描述了采用 UTF-16 编码的一个代码单元。

强烈建议不要在程序中使用 `char` 类型，除非确实需要处理 UTF-16 代码单元。最好将字符串作为抽象数据类型来处理（有关这方面的内容将在 3.6 节讨论）。

### 3.3.5 boolean 类型

`boolean`（布尔）类型有两个值：`false` 和 `true`，用来判定逻辑条件。整型值和布尔值之间不能进行相互转换。

 C++ 注释：在 C++ 中，数值甚至指针可以代替布尔值。值 0 相当于布尔值 `false`，非 0 值相当于布尔值 `true`。在 Java 中则不是这样。因此，Java 程序员不会遇到以下麻烦：

```
if (x = 0) // oops... meant x == 0
```

在 C++ 中这个测试可以编译运行，其结果总是 false。而在 Java 中，这个测试将不能通过编译，其原因是整数表达式  $x=0$  不能转换为布尔值。

## 3.4 变量与常量

与所有程序设计语言一样，Java 也使用变量来存储值。常量就是值不变的变量。在下面几节中，你会了解如何声明变量和常量。

### 3.4.1 声明变量

在 Java 中，每个变量都有一个类型 (type)。声明一个变量时，先指定变量的类型，然后是变量名。这里给出一些声明变量的示例：

```
double salary;
int vacationDays;
long earthPopulation;
boolean done;
```

注意每个声明都以分号结束。由于声明是一个完整的 Java 语句，而所有 Java 语句都以分号结束，所以这里的分号是必须的。

作为变量名（以及其他名字）的标识符由字母、数字、货币符号以及“标点连接符”组成。第一个字符不能是数字。

'+' 和 '©' 之类的符号不能出现在变量名中，空格也不行。字母区分大小写：main 和 Main 是不同的标识符。标识符的长度基本上没有限制。

与大多数程序设计语言相比，Java 中“字母”“数字”和“货币符号”的范围更大。字母是指一种语言中表示字母的任何 Unicode 字符。例如，德国用户可以在变量名中使用字母 'ä'；讲希腊语的人可以使用 π。类似地，数字包括 '0'~'9' 和表示一位数字的任何 Unicode 字符。货币符号为 \$、€、¥ 等。标点连接符包括下画线字符 \_、“波浪线” \_\_ 以及其他一些符号。实际上大多数程序员都总是使用 A~Z、a~z、0~9 和下画线 \_。

 提示：如果想要知道标识符中可以使用哪些 Unicode 字符，可以使用 Character 类的 isJavaIdentifierStart 和 isJavaIdentifierPart 方法来检查。

 提示：尽管 \$ 是一个合法的标识符字符，但不要在你自己的代码中使用这个字符。它只用于 Java 编译器或其他工具生成的名字。

另外，不能使用 Java 关键字作为变量名。

在 Java 9 中，单下画线 \_ 是一个保留字。将来的版本可能使用 \_ 作为通配符。

可以在一行中声明多个变量：

```
int i, j; // both are integers
```