Java领域最有影响力和价值的著作之一，与《Java编程思想》齐名，10余年全球畅销不衰，广受好评

根据Java SE 7全面更新，系统全面讲解Java语言的核心概念、语法、重要特性和开发方法，包含大量案例，实践性强

PEARSON

HZ BOOKS 华章科技

Java 核心技术 系列

Java 核心技术 卷 I

基础知识（原书第9版）

Core Java Volume I——Fundamentals
(Ninth Edition)

Cay S. Horstmann　Gary Cornell　著
周立新　陈波　叶乃文　邝劲筠　杜永萍　译

机械工业出版社
China Machine Press

Java 核心技术系列

Java 核心技术

卷 I 基础知识

(原书第 9 版)

Core Java Volume I——Fundamentals (9th Edition)

(美) Cay S. Horstmann, Gary Cornell 著

周立新 陈 波 叶乃文 邝劲筠 杜永萍 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　卷 I　基础知识（原书第 9 版）/（美）霍斯特曼（Horstmann, C. S.），科内尔（Cornell, G.）著；周立新等译 . —北京：机械工业出版社，2013.11
（Java 核心技术系列）
书名原文：Core Java Volume I——Fundamentals (Ninth Edition)

ISBN 978-7-111-44514-2

Ⅰ. J…　Ⅱ. ①霍…　②科…　③周…　Ⅲ. JAVA 语言 – 程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2013）第 251592 号

Java 领域最有影响力和价值的著作之一，由拥有 20 多年教学与研究经验的资深 Java 技术专家撰写（获 Jolt 大奖），与《 Java 编程思想》齐名，10 余年全球畅销不衰，广受好评。第 9 版根据 Java SE 7 全面更新，同时修正了第 8 版中的不足，系统全面讲解了 Java 语言的核心概念、语法、重要特性和开发方法，包含大量案例，实践性强。

本书共 14 章。第 1 章概述 Java 语言与其他程序设计语言不同的性能；第 2 章讲解如何下载和安装 JDK 及本书的程序示例；第 3 章介绍变量、循环和简单的函数；第 4 章讲解类和封装；第 5 章介绍继承；第 6 章解释接口和内部类；第 7 章概述图形用户界面程序设计知识；第 8 章讨论 AWT 的事件模型；第 9 章探讨 Swing GUI 工具箱；第 10 章讲解如何部署自己的应用程序或 applet ；第 11 章讨论异常处理；第 12 章概要介绍泛型程序设计；第 13 章讲解 Java 平台的集合框架；第 14 章介绍多线程。本书最后还有一个附录，其中列出了 Java 语言的保留字。

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：关　敏
　　印刷
2014 年 1 月第 1 版第 1 次印刷
186mm × 240mm • 45 印张
标准书号：ISBN 978-7-111-44514-2
定　　价：119.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

译 者 序

书写 Java 传奇的 Sun Microsystems 曾经堪称“日不落”帝国，但服务器市场的萎缩却让这个声名赫赫的庞大帝国从蓬勃走向落寞。在 2009 年被 Oracle 收购之后，Sun 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！ 2011 年 Oracle 终于发布了 Java 的新版本，这就是 Java 7。相较于之前的版本，尽管这一版的改动不大，但让广大 Java 程序员看到了希望，有了前进的动力。

值得一提的是，2009 年之前，伴随着 Java 的成长，《 Java 核心技术》也从第 1 版一直更新到第 8 版，得到了广大 Java 程序设计人员的青睐，成了一本畅销不衰的 Java 经典图书。经过几年的蛰伏，为 Java 7 打造的《 Java 核心技术》第 9 版终于问世，第 9 版对上一版做了全面修订和更新，以反映 Java 7 增补、删改的内容。它将续写从前的辉煌，使人们能及时跟上 Java 前进的脚步。

本书由周立新、陈波等主译，程芳、刘晓兵、张练达、陈峰、江健、谢连宝、张雷生、杨健康、张莹参与了全书的修改整理，并完善了关键部分的翻译。全体人员共同完成了本书的翻译工作。特别需要说明的是，按照出版社的要求，这一版的翻译是在第 8 版中文版的基础上完成的，因此尤其要感谢第 8 版的译者叶乃文、邝劲筠和杜永萍，他们的辛勤工作为新版本的翻译奠定了很好的基础。

书中文字与内容力求忠实于原著，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

2013 年 10 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。其原因在于它将有望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，还是任何其他渠道。事实上，就发展前景而言，Java 的地位是独一无二的。它是一种完全可信赖的程序设计语言，得到了除微软之外的所有厂家的认可。其固有的可靠性与安全性不仅令 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、多线程等高级程序设计任务的支持。

1995 年以来，已经发布了 Java 开发工具箱（Java Development Kit）的 8 个主要版本。在过去的 17 年中，应用程序编程接口（API）已经从 200 个类扩展到超过 3000 个类，并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。

本书是《Java 核心技术》第 9 版的卷 I。自《Java 核心技术》出版以来，每个新版本都尽可能快地跟上 Java 开发工具箱发展的步伐，而且每一版都重新改写了部分内容，以便适应 Java 的最新特性。在这一版中，已经反映了 Java 标准版（Java SE 7）的特性。

与前几版一样，本版仍然将读者群定位在那些打算将 Java 应用到实际工程项目中的程序设计人员。本书假设读者是一名具有程序设计语言（除 Java 之外）坚实背景知识的程序设计人员，并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）。这些内容绝对不会在本书中出现。本书的目标是让读者充分理解书中介绍的 Java 语言及 Java 类库的相关特性，而不会产生任何误解。

在本书中，我们选用大量的示例代码演示所讨论的每一个语言特性和类库特性。我们有意使用简单的示例程序以突出重点，然而，其中的大部分既不是赝品也没有偷工减料。它们将成为读者自己编写代码的良好开端。

我们假定读者愿意（甚至渴望）学习 Java 提供的所有高级特性。本书将详细介绍下列内容：

- 面向对象程序设计
- 反射与代理
- 接口与内部类
- 事件监听器模型
- 使用 Swing UI 工具箱进行图形用户界面设计
- 异常处理
- 泛型程序设计
- 集合框架
- 并行操作

随着 Java 类库的爆炸式增长，一本书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图形用户界面程

序设计的基础知识。卷Ⅱ——高级特性，涉及企业特性以及高级的用户界面程序设计。其中详细讨论下列内容：

- 文件与流
- 分布式对象
- 本地方法
- XML 处理
- 网络编程
- 高级图形
- 数据库
- 高级 GUI 组件
- 国际化
- JavaBeans
- 注释

在编写本书的过程中，难免出现错误和不准确之处。我们很想知道这些错误，当然，也希望同一个问题只被告知一次。我们在网页 http://horstmann.com/corejava 中以列表的形式给出了常见的问题、bug 修正和解决方法。在勘误页（建议先阅读一遍）最后附有用来报告 bug 并提出修改意见的表单。如果我们不能回答每一个问题或没有及时回复，请不要失望。我们会认真地阅读所有的来信，感谢您的建议使本书后续的版本更清晰、更有指导价值。

关于本书

第 1 章概述 Java 与其他程序设计语言不同的性能。解释这种语言的设计初衷，以及在哪些方面达到了预期的效果。然后，简要叙述 Java 诞生和发展的历史。

第 2 章详细地论述如何下载和安装 JDK 以及本书的程序示例。然后，通过编译和运行三个典型的 Java 程序（一个控制台应用、一个图形应用、一个 applet），指导读者使用简易的 JDK、可启用 Java 的文本编辑器以及一个 Java IDE。

第 3 章开始讨论 Java 语言。这一章涉及的基础知识有变量、循环以及简单的函数。对于 C 或 C++ 程序员来说，学习这一章的内容将会感觉一帆风顺，因为这些语言特性的语法本质上与 C 语言相同。对于没有 C 语言程序设计背景，但使用过其他程序设计语言（如 Visual Basic）的程序员来说，仔细地阅读这一章是非常必要的。

面向对象程序设计（Object-Oriented Programming, OOP）是当今程序设计的主流，而 Java 是一种完全面向对象的语言。第 4 章将介绍面向对象两个基本成分中最重要的——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，还对如何正确地进行 OOP 设计给出了忠告。最后，介绍奇妙的 javadoc 工具，它将代码注释转换为超链接的网页。熟悉 C++ 的程序员可以快速地浏览这一章，而没有面向对象程序设计背景的程序员，应在进一步学习 Java 之前花一些时间了解 OOP 的有关概念。

类与封装仅仅是 OOP 中的一部分，第 5 章将介绍另一部分——继承。继承使程序员可以使用现有的类，并根据需要进行修改。这是 Java 程序设计中的基础。Java 中的继承机制与 C++ 的继承机制十分相似。C++ 程序员只需关注两种语言的不同之处即可。

第 6 章展示如何使用 Java 的接口。接口可以让你的理解超越第 5 章的简单继承模型。掌握接口的使用将可以获得 Java 的完全的面向对象程序设计的能力。本章还将介绍 Java 的一个有用的技术特性——内部类。内部类可以使代码更清晰、更简洁。

第 7 章开始细致地讨论应用程序设计。每一个 Java 程序员都应该了解一些图形用户界面程序设计的知识，本卷包含了其中的基本内容部分。本章将展示如何制作窗口、如何在窗口中绘图、如何用几何图形作画、如何用多种字体格式化文本以及如何显示图像。

第 8 章详细讨论 AWT（Abstract Window Toolkit ）的事件模型。我们将介绍如何编写代码来响应鼠标点击或按键等事件。同时，还将介绍如何处理基本的 GUI 元素，比如：按钮和面板。

第 9 章详细讨论 Swing GUI 工具箱。Swing 工具箱允许建立一个跨平台的图形用户界面。本章将介绍如何建立各种各样的按钮、文本组件、边框、滑块、列表框、菜单以及对话框等。一些更高级的组件将在卷 II 中讨论。

第 10 章阐述如何部署自己编写的应用程序或 applet。在这里将描述如何将应用程序打包到 JAR 文件中，以及如何使用 Java 的 Web Start 和 applet 机制在 Internet 上发布应用程序。最后，将解释 Java 程序部署之后如何存储和检索配置信息。

第 11 章讨论异常处理，即 Java 的健壮机制，它用于处理调试好的程序可能出现意外的情况。异常提供了一种将正常的处理代码与错误处理代码分开的有效手段。当然，即使程序包含处理所有异常情况的功能，依然有可能无法按照预计的方式工作。这一章的后半部分将给出大量的实用调试技巧。最后，将指导你完成一个完整的示例调试过程。

第 12 章概要介绍泛型程序设计，这是 Java SE 5.0 的一项重要改进。泛型程序设计使得程序拥有更好的可读性和安全性。在这里，将展示如何使用强类型机制，而舍弃不安全的强制类型转换，以及如何处理与旧版本 Java 兼容而带来的复杂问题。

第 13 章介绍 Java 平台的集合框架。当需要将大量对象收集到一起，并在以后要对它们进行检索时，可能会想要使用集合，这是目前最为合适的做法，它取代了将这些元素放置在数组中的做法。本章将介绍如何使用预先建立好的标准集合。

第 14 章是本书的最后一章。在这一章中将介绍多线程，这是一种可以让程序任务并行执行的特性（线程是程序中的控制流），并阐述如何建立线程、如何处理线程的同步问题。从 Java SE 5.0 开始，多线程有了很大的改进，本章将介绍所有这些新的机制。

附录列出了 Java 语言的保留字。

约定

本书使用以下图标表示特殊内容。

注释：“注释”信息会用这样的“注释”图标标识。

提示：“提示”信息会用这样的“提示”图标标识。

警告：对于可能出现的危险，我们用一个“警告”图标做出警示。

C++ 注释：在本书中有许多用来解释 Java 与 C++ 之间不同的 C++ 注释。对于没有 C++ 程序设计背景，或者不擅长 C++ 程序设计、把它当做一场噩梦不愿再想起的程序员来说，可以跳过这些注释。

API 应用程序编程接口

Java 带有一个很大的程序设计库，即应用程序编程接口。第一次使用 API 调用时，将会在该节的结尾给出一个概要描述。这些描述十分通俗易懂，希望能够比联机 API 文档提供更多的信息。类、接口或方法名后面的编号是介绍该特性的 JDK 版本号。

程序（源代码见本书网站）以如下形式给出：

程序清单 1-1　inputTest/InputTest.java

示例代码

本书的网站 http://horstmann.conl/corejava 以压缩的形式提供了书中的所有示例代码。可以用熟悉的解压缩程序或者用 Java 开发工具箱中的 jar 实用程序解压这个文件。有关安装 Java 开发工具箱和示例程序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，改写一本书也并不像想象的那样轻松，尤其是 Java 一直在持续不断地更新。编著一本书让很多人耗费了很多心血，在此衷心地感谢《Java 核心技术》编写小组的每一位成员。

Prentice Hall 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，我希望每一位都能够知道我对他们努力的感恩。与以往一样，我要真诚地感谢我的编辑，Prentice Hall 公司的 Greg Doench，从本书的写作到出版一直给予了指导，同时感谢那些不知其姓名的为本书做出贡献的幕后人士。非常感谢 Julie Nahil 在图书制作方面给予的支持，还要感谢 Dmitry Kirsanov 和 Alina Kirsanova 完成手稿的编辑和排版工作。我还要感谢早期版本中我的合作者，Gary Cornell，他已经转向其他的事业。

感谢早期版本的许多读者，他们指出了许多令人尴尬的错误并给出了许多具有建设性的修改意见。我还要特别感谢本书优秀的审阅小组，他们仔细地审阅我的手稿，使本书减少了许多错误。

本书及早期版本的审阅专家包括：Chuck Allison（UtahValley 大学）、Lance Andersen（Oracle）、Alec Beaton（IBM）、Cliff Berg、Joshua Bloch、David Brown、Corky Cartwright、Frank Cohen（PushToTest）、Chris Crane（devXsolution）、Dr. Nicholas J. De Lillo（Manhatta 学院）、Rakesh Dhoopar（Oracle）、David Geary（Clarity Training）、Jim Gish（Oracle）、Brian Goetz（Oracle）、Angela Gordon、Dan Gordon（Electric Cloud）、Rob Gordon、John Cray（Hartford 大 学）、Cameron Gregory（olabs.com）、Marty Hall（coreservlets.com、Inc.）、Vincent Hardy（Adobe Systems）、Dan Harkey（San Jose 州立大学）、William Higgins（IBM）、Vladimir Ivanovic（PointBase）、Jerry Jackson（CA Technologies）、Tim Kimmet（Walmart）、Chris Laffra、Charlie Lai（Apple）、Angelika Langer、Doug Langston、Hang Lau（McGill 大学）、Mark Lawrence、Doug Lea（SUNY Oswego）、Gregory Longshore、Bob Lynch（Lynch Associates）、Philip Milne（顾问）、Mark Morrissey（Oregon 研究院）、Mahesh Neelakanta（Florida Atlantic 大学）、Hao Pham、Paul Philion、Blake Ragsdell、Stuart Reges（Arizona 大学）、Rich Rosen（Interactive Data Corporation）、Peter Sanders（法国尼斯 ESSI 大学），Paul Sanghera 博士（San Jose 州立大学和 Brooks 学院）、Paul Sevinc（Teamup AG）、Devang Shah（Sun Microsystems）、Bradley A. Smith、Steven Stelting（Oracle）、Christopher Taylor、Luke Taylor（Valtech）、George Thiruvathukal、Kim Topley（StreamingEdge）、Janet Traub、Paul Tymal（顾问）、Peter van der Linden（Motorola Mobile Devices）、Burt Walsh、Dan Xu（Oracle) 和 John Zavgren（Oracle).

Cay Horstmann

2012 年 9 月于加州旧金山

目　录

第 1 章　Java 程序设计概述

- ▲ Java 程序设计平台
- ▲ Java 发展简史
- ▲ Java “白皮书”的关键术语
- ▲ 关于 Java 的常见误解
- ▲ Java applet 与 Internet

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》、《商业周刊》这样的主流媒体。Java 是第一种也是唯一一种在 National Public Radio 上占用了 10 分钟时间来进行介绍的程序设计语言，并且还得到了 $100 000 000 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。重温那些令人兴奋的日子是很有意思的。本章将简要地介绍一下 Java 语言的发展历史。

1.1　Java 程序设计平台

本书的第 1 版是这样描写 Java 的：“作为一种计算机语言，Java 的广告词确实有点夸大其辞。然而，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 无疑是一个好的选择。有人认为：Java 将有望成为一种最优秀的程序设计语言，但还需要一个相当长的发展时期。一旦一种语言应用于某个领域，与现存代码的相容性问题就摆在了人们的面前。”

我们的编辑手中有许多这样的广告词。这是 Sun 公司高层的某位不愿透露姓名的人士提供的。然而，现在看起来，当初的这些预测还是有一定准确性的。Java 有许多非常优秀的语言特性，本章稍后将会详细地讨论这些特性。由于相容性这个严峻的问题确实存在于现实中，所以，或多或少地还是有一些“累赘”被加到语言中，这就导致 Java 并不如想象中的那么完美无瑕。

但是，正像我们在第 1 版中已经指出的那样，Java 并不只是一种语言。在此之前出现的那么多种语言也没有能够引起那么大的轰动。Java 是一个完整的平台，有一个庞大的库，其中包含了很多可重用的代码和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，常常希望能够有一种语言，它具有令人赏心悦目的语法和易于理解的语义（C++ 不是这样的）。与许多其他的优秀语言一样，Java 恰恰满足了这些要求。有些语言提供了可移植性、垃圾收集等，但是，没有提供一个大型的库。如果想要有奇特的绘图功能、网络连接功能和数据库存取功能就必须自己动手编写代码。Java 这种功能齐全的出

色语言，具有高质量的执行环境以及庞大的库。正是因为它集多种优势于一身，所以对广大的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 的设计者已经编写了颇有影响力的“白皮书”，用来解释设计的初衷以及完成的情况，并且发布了一个简短的摘要。这个摘要用下面 11 个关键术语进行组织：

1）简单性
2）面向对象
3）网络技能（Network-Savvy）
4）健壮性
5）安全性
6）体系结构中立
7）可移植性
8）解释型
9）高性能
10）多线程
11）动态性

本节将论述下列主要内容：

- 给出白皮书中对每个关键术语的概述，这是 Java 设计者对相关术语的论述。
- 凭借 Java 当前版本的使用经验，给出对这些术语的理解。

注释：写这本书时，白皮书可以在 www.oracle.com/technetwork/java/langenv-140151.html. 上找到。对于 11 个关键术语的论述请参看 http://labs.oracle.com/features/tenyeares/volcd/papers/7Gosling.pdf。

1.2.1 简单性

人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，尽管人们发现 C++ 不太适用，但在设计 Java 的时候还是尽可能地接近 C++，以便系统更易于理解。Java 剔除了 C++ 中许多很少使用、难以理解、易混淆的特性。在目前看来，这些特性带来的麻烦远远多于其带来的好处。

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，那里比较详细地解释了 Java 与 C++ 之间的区别）。然而，设计者并没有试图清除 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。如果知道 C++ 就会发现可以轻而易举地将其转换成 Java。

如果已经习惯于使用可视化的编程环境（例如 Visual Basic），你就不会觉得 Java 简单了。Java 有许多奇怪的语法（尽管掌握其要领并不需要很长时间），更重要的是，使用 Java 需要自己编写大量的程序。Visual Basic 的魅力在于它的可视化设计环境几乎自动地为应用程序提供了大量的基础结构。而使用 Java 实现同样的功能却需要手工编制代码，通常代码量还相当大。然而，已经有一些支持“拖放”风格程序开发的第三方开发环境。

简单的另一个方面是小。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。基本的解释器以及类支持大约仅为 40KB；再加上基础的标准类库和对线程的支持（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。现在有一个独立的具有较小类库的 Java 微型版（Java Micro Edition）用于嵌入式设备。

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”木匠始终关注的是所制作的椅子，第二位才是所使用的工具；一个“非面向对象的”木匠首先考虑的是所用的工具。在本质上，Java 的面向对象能力与 C++ 是一样的。

在过去的 40 年里，面向对象已经证明了自身的价值，一种现代的程序设计语言不使用面向对象技术简直让人难以置信。的确，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继承，在 Java 中，取而代之的是简单的接口概念，以及 Java 的元类（metaclass）模型（有关这部分内容将在第 5 章中讨论）。

注释：如果没有使用面向对象程序设计语言的经验，你一定要仔细阅读第 4 章～第 6 章。这些章节解释了什么是面向对象程序设计以及在编程实现复杂的项目时为什么比传统的像 C 或 Basic 这样的面向过程的语言更加有效。

1.2.3 网络技能

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

人们已经看到 Java 的网络能力强大且易于使用。任何曾经试图使用其他语言进行网络编程的人都会惊呼 Java 竟然把类似打开 socket 连接这类繁重的任务都变得如此简单（在本书的卷 II 中介绍网络连接）。另外，远程方法调用机制使得分布式对象之间可以进行通信（也将在卷 II 中介绍）。

1.2.4 健壮性

Java 的设计目标之一在于使得 Java 编写的程序具有多方面的可靠性。Java 投入了大量的精力进行早期的问题检测、后期动态的（运行时）检测，并消除了有出错倾向的状态……Java 和 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。

这个特性非常有用。Java 编译器能够检测许多在其他语言中仅在运行时刻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。

如果曾经只使用过 Visual Basic 这类没有显式指针的语言，你就会感觉这么说似乎有些小题大做了。然而，C 程序员就没有这样幸运了。他们需要利用指针存取字符串、数组、对象，甚至文件。在 Visual Basic 中，根本不必使用指针访问这些实体，也不必关心有关内存分配的问题。另一方面，在没有指针的语言中，许多数据结构很难实现。Java 具有双方的优势。它不需要使用指针构造诸如字符串、数组这样的结构。如果必要的话，它也能够具有指针的能力，如链表。Java 绝对是安全的，其原因是永远不会存取一个“坏的”指针，造成内存分配的错误，也不必防范内存泄漏。

1.2.5 安全性

Java 适用于网络 / 分布式环境。为了达到这个目标，在安全方面投入了很大精力。使用 Java 可以构建防病毒、防篡改的系统。

本书的第 1 版曾经说过：“永远不要把话说绝！”事实证明这是正确的。在 Java 开发工具箱第 1 版启用后不久，普林斯顿大学的一些安全专家就发现了在 JDK1.0 中的某些安全特性方面存在着一些非常隐蔽的 bug。Sun Microsystems 大力支持对 Java 的安全性的研究，制定了供人们使用的规范，实现了虚拟机和安全库，并迅速地处理了所有已知的安全 bug。在任何情况下，蒙骗 Java 的安全机制都是十分困难的。现在，发现 bug 的技术越来越强，数目越来越少。

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。如，蠕虫等病毒常用的攻击手段。
- 在自己的处理空间之外破坏内存。
- 未经授权读写文件。

许多安全特性相继不断地加入到 Java 中。自从 Java 1.1 问世以来，Java 就有了数字签名类（digitally signed class）的概念（请参看卷 II）。通过数字签名类，可以确定类的作者。如果信任这个类的作者，这个类就可以在你的机器上拥有更多的权限。

注释：来自微软的基于 ActiveX 技术的竞争代码传输机制，其安全性完全依赖于数字签名。这显然是不够的，因为微软自身产品的任何用户都可以证实，来自知名提供商的程序会崩溃并对系统产生危害。Java 的安全机制比 ActiveX 要强得多，因为它是在应用程序运行时加以控制并制止恶意性破坏的。

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，这是一种编译过的代码，只要有 Java 运行时系统，就可以在许多处理器上运行。Java 编译器通过生成与特定的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，而且还可以迅速地翻译成本地机器的代码。

这并不是什么新的思路。40 多年以前，Niklaus Wirth 实现的原始 Pascal 以及 UCSD Pascal 系统都使用了这种技术。

当然，解释字节码肯定会比全速运行机器指令慢很多。所以说，这是不是一个好的思路还很难讲！然而，虚拟机有一个选项，可以将使用最频繁的字节码序列翻译成机器码，这一过程被称为即时编译。这一策略已经证明十分有效，致使微软的 .NET 平台也依赖于虚拟机。

虚拟机还有一些其他的优点。虚拟机可以检测指令序列的行为，以增强其安全性。有些程序还可以快速地生成字节码，并动态地增强所运行程序的处理能力。

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算法都做了明确的说明。

例如，Java 中的 int 永远为 32 位的整数，而在 C/C++ 中，int 可能是 16 位整数、32 位整数，也可能是编译器提供商指定的其他大小。唯一的限制只是 int 类型的大小不能低于 short int，并且不能高于 long int。在 Java 中，数据类型具有固定的大小，这消除了代码移植时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，消除了字节顺序的困扰。字符串是用标准的 Unicode 格式存储的。

作为系统组成部分的类库，定义了可移植的接口。例如，有一个抽象的 Window 类给出了在 UNIX、Windows 和 Macintosh 环境下的不同实现。

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有多么困难。Java 1.0 就尝试着做了这么一个壮举，发布了一个将常用的用户界面元素映射到不同平台上的简单工具箱。遗憾的是，花费了大量的心血，却构建了一个在各个平台上都难以让人接受的库（而且，在不同平台的图形实现中有不同的 bug）。不过，这毕竟是个开端。对于许多应用问题来说，可移植性比华而不实的用户界面更加重要；而且这些应用程序从 Java 的早期版本中获益匪浅。现在，用户界面工具箱已经完全重写了，不再依赖于主机的用户界面。现在的 Java 版本比早期版本更加稳定，更加吸引人。

1.2.8 解释型

Java 解释器可以在任何移植了解释器的机器上执行 Java 字节码。由于链接是一个增量式且轻量级的过程，所以，开发过程也变得更加快捷，更加具有探索性。

增量式链接有其优势，但给开发过程带来的好处显然是言过其实了。事实上，早期的 Java 开发工具的速度相当慢。现在，使用即时编译器将字节码翻译成机器码。

1.2.9 高性能

尽管对解释后的字节码性能已经比较满意，但在有些场合下还需要更加高效的性能。字节码可以（在运行时刻）快速地翻译成运行这个应用程序的特定 CPU 的机器码。

使用 Java 的头几年，许多用户不同意这样的看法：性能就是“适用性更强”。然而，现在的即时编译器已经非常出色，以至于成了传统编译器的竞争对手。在某些情况下，甚至超越了传统编译器，其原因是它们含有更多的可用信息。例如，即时编译器可以监控经常执行哪些代码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“内嵌”）。即时编译器知道哪些类已经加载。如果基于当前加载的类集，且特定的函数不被覆盖的话就可以内嵌。必要时，还可以撤销优化。

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。

如果曾经使用过其他语言编写多线程的应用程序，就会对 Java 多线程处理的便捷性惊叹不已。只要操作系统支持，Java 中的线程就可以利用多个处理器。在底层，主流平台的线程实现机制各不相同，Java 并没有花费太大的力气对此实现平台无关性。在不同的机器上，只是调用多线程的代码完全相同；Java 把多线程的实现交给了底层的操作系统或线程库来完成。尽管如此，多线程编译的简单性是 Java 成为颇具魅力的服务器端开发语言的主要原因之一。

1.2.11 动态性

从各种角度看，Java 与 C 或 C++ 相比更加具有动态性。它能够适应不断发展的环境。库中可以自由地添加新方法和实例变量，而对客户端却没有任何影响。在 Java 中找出运行时类型信息十分简单。

当需要将某些代码添加到正在运行的程序中时，动态性将是一个非常重要的特性。一个很好的例子是：从 Internet 上下载代码，然后在浏览器上运行。在 Java 1.0 中，不能直接获得运行时的类型信息，而 Java 的当前版本允许程序员知道对象的结构和行为。这对于必须在运行时分析对象的系统来说非常有用。这些系统有：Java GUI 构建器、智能调试器、可插拔组件以及对象数据库。

注释：Java 成功地推出后不久，微软就发布了一个叫做 J++ 的产品，它与 Java 有相同的编程语言以及虚拟机。现在，微软不再支持 J++，取而代之的是另一种被称为 C# 的语言。C# 与 Java 有很多相似之处，然而使用的却是完全不同的虚拟机。甚至还有一种 J# 语言可将 J++ 的应用迁移到使用 C# 的虚拟机上。本书不准备介绍 J++、C# 或 J# 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 Java 程序称为 applet。为了使用 applet，需要启用 Java 的 Web 浏览器执行字节码。由于 Sun 公司负责发放 Java 源代码的许可证，并坚持不允许对语言和基本类库的结构做出任何修改，因此，Java 的 applet 应该可以运行在任何启用 Java 浏览器上，并且无论何时访问包含

applet 的网页，都会得到程序的最终版本。最重要的是，要感谢虚拟机的安全机制，它让我们不必再担心来自恶意代码的攻击。

用户下载一个 applet 就如同在网页中嵌入一幅图片。applet 成了页面的一部分。文本环绕着 applet 所占据的空间周围。关键一点是图片是活动的。它可以对用户命令做出响应，改变外观，在运行它的计算机与提供它的计算机之间传递数据。

图 1-1 展示了一个很好的动态网页的示例。Jmol applet 显示了分子结构，这将需要相当复杂的计算。在这个网页中，可以利用鼠标进行旋转，调整焦距等操作，以便更加细致地理解分子结构。用静态网页将无法实现这种直接的操作，而 applet 却可以达到此目的（可以在 http://jmol.sourceforge.net 上找到这个 applet）。

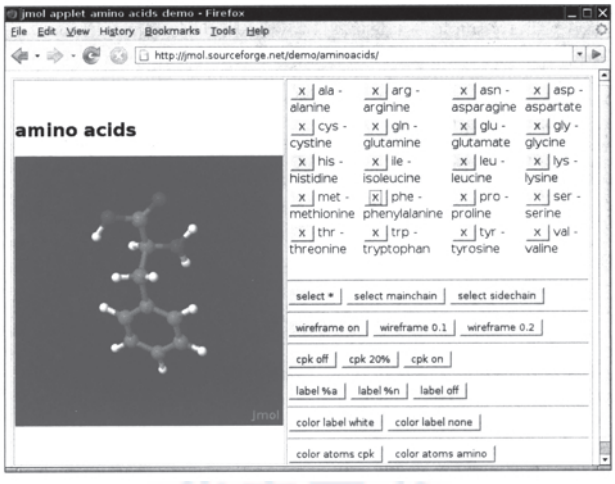

图 1-1 Jmol applet

当 applet 首次出现时，人们欣喜若狂。许多人相信 applet 的魅力将会导致 Java 迅速地流行起来。然而，初期的兴奋很快就淡化了。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于利用 Java 的最新版本开发 applet。今天，当需要在浏览器中显示动态效果时，大多数网页都直接使用 JavaScript 或 Flash。另外，Java 已经成为用来开发服务器端应用程序的最流行的语言，使用这些服务器端应用程序可以产生网页、运行后端逻辑。

1.4 Java 发展简史

本节将介绍 Java 的发展简史。这些参考资料来源于多方面的出版物（最重要的是 SunWorld 的在线杂志 1995 年 7 月刊上对 Java 创建者的专访）。

Java 的历史要追溯到 1991 年，由 Patrick Naughton 及其伙伴 James Gosling（一个全能的计算机奇才）带领的 Sun 公司的工程师小组想要设计一种小型的计算机语言，主要用于像

有线电视转换盒这类的消费设备。由于这些消费设备的处理能力和内存都很有限，所以语言必须非常小且能够生成非常紧凑的代码。另外，由于不同的厂商会选择不同的中央处理器（CPU），因此这种语言的关键是不能与任何特定的体系结构捆绑在一起。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联想起很早以前的一种模型，某些 Pascal 的实现曾经在早期的 PC 上尝试过这种模型。以 Pascal 的发明者 Niklaus Wirth 为先驱，率先设计出一种为假想的机器生成中间代码的可移植语言（假想的机器称为虚拟机——Java 虚拟机即 JVM 的命名由此而来）。这种中间代码可以应用于所有已经正确安装解释器的机器上。Green 项目工程师也使用了虚拟机，从而解决了课题中的主要问题。

不过，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 Pascal。特别是这种语言是面向对象的，而不是面向过程的。就像 Gosling 在专访中谈到的：“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”（这么起名的原因大概是因为他非常喜欢自己办公室外的橡树）。Sun 公司的人后来发现 Oak 是一种已有的计算机语言的名字，于是，将其改名为 Java。事实证明这是一个极好的选择。

1992 年，Green 项目发布了它的第一个产品，称之为“*7”。这个产品具有非常智能的远程控制（其能力相当于长 6 英寸、宽 4 英寸、高 4 英寸的 SPARC 工作站）。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出其他的方法来将他们的技术推向市场。然而，没有一个标准消费品公司对此感兴趣。于是，Green 项目组竞标了一个提供视频点播等新型服务的有线电视盒的项目，但没有成功（有趣的是，得到这个项目的公司的领导恰恰是开创 Netscape 公司的 Jim Clark。Netscape 公司后来对 Java 的成功给予了很大的帮助）。

Green 项目（这时换了一个新名字——“First Person 公司”）花费了 1993 年一整年以及 1994 年的上半年，一直在苦苦寻求其技术的买家。然而，一个也没有找到（Patrick Naughton，项目组的创立人之一，也是完成此项目大多数市场工作的人，声称为了销售这项技术，累计飞行了 300 000 英里）。1994 年 First Person 公司解散了。

当这一切在 Sun 公司中继续进行的时候，Internet 的万维网也日渐发展壮大。Web 的关键是把超文本页面转换到屏幕上的浏览器。1994 年大多数人都在使用 Mosaic，这是一个 1993 年出自伊利诺斯大学超级计算中心的非商业化的 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。他后来成了 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 年中期，Java 语言的开发者意识到：“我们能够建立一个最酷的浏览器。我们已经拥有在客户机 / 服务器主流模型中所需要的体系结构中立、实时、可靠、安全——这些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。为了炫耀 Java 语言超强的能力，HotJava 浏览器采用 Java 编写。然而，设计者也非常清

楚现在所谓的 applet 的威力，因此他们让 HotJava 浏览器具有执行网页中内嵌代码的能力。这一“技术印证”在 1995 年 5 月 23 日的 SunWorld 上得到展示，同时引发了人们延续至今的对 Java 的狂热追逐。

1996 年年初，Sun 发布了 Java 的第 1 个版本。人们很快地意识到 Java1.0 不能用来进行真正的应用开发。的确，可以使用 Java 1.0 来实现在画布上随机跳动的神经质的文本 applet，但它却没有提供打印功能。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 Java1.1 弥补了其中的大多明显的缺陷，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。不过它仍然具有很大的局限性。

1998 年 JavaOne 会议的头号新闻是即将发布 Java 1.2 版。这个版本取代了早期玩具式的 GUI，并且它的图形工具箱更加精细而具有可伸缩性，更加接近“一次编写，随处运行”的承诺。在 1998 年 12 月 Java 1.2 发布三天之后，Sun 公司市场部将其名称改为更加吸引人的“Java 2 标准版软件开发工具箱 1.2 版”。

除了“标准版”之外，Sun 还推出了两个其他的版本：一个是用于手机等嵌入式设备的“微型版”；另一个是用于服务器端处理的“企业版”。本书主要讲述标准版。

标准版的 1.3 和 1.4 版本对最初的 Java 2 版本做出了某些改进，扩展了标准类库，提高系统性能。当然，还修正了一些 bug。在此期间，Java applet 采用低调姿态，并淡化了客户端的应用，但 Java 却成为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（这一版本原来被命名为 1.5 版，在 2004 年的 JavaOne 会议之后，版本数字升至 5.0）。经历了多年的研究，这个版本添加了泛型类型（generic type）（类似于 C++ 的模板），其挑战性在于添加这一特性并没有对虚拟机做出任何修改。另外，还有几个受 C# 启发的很有用的语言特性：“for each”循环、自动装箱和元数据。

版本 6（没有后缀 .0）于 2006 年年末发布。同样，这个版本没有对语言方面再进行改进。但是，改进了其他性能，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 Microsystems 终于沦陷，于 2009 年被 Oracle 收购。Java 的开发停滞了很长一段时间。直到 2011 年 Oracle 发布了 Java 的一个新版本，Java 7，其中只做了一些简单的改进，而决定将重要的改进推迟到 Java 8，该版本将在 2013 年发布。

表 1-1 展示了 Java 语言以及类库的发展状况。可以看到，应用程序接口（API）的规模发生了惊人的变化。

表 1-1 Java 语言的发展状况

版　本	年　份	语言新特性	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	无	1524
1.3	2000	无	1840
1.4	2002	断言	2723

（续）

版 本	年 份	语言新特性	类与接口的数量
5.0	2004	泛型类型、“for each”循环、可变元参数、自动装箱、元数据、枚举、静态导入	3279
6	2006	无	3793
7	2011	基于字符串的 switch、变形操作符、二进制字面量、异常处理改进	4024

1.5 关于 Java 的常见误解

在结束本章之前，我们列出了一些关于 Java 的常见误解，同时给出了解释。

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）使用 XML，就不需要 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，许多重要的第三方 XML 工具采用 Java 编写。有关这方面更加详细的信息请参看卷 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都不太容易学习。首先，必须将编写玩具式程序的轻松和开发实际项目的艰难区分开来。需要注意的是：本书只用了 4 章讨论 Java 语言。在两卷中，其他的章节介绍如何使用 Java 类库将 Java 语言应用到实际中去。Java 类库包含了数千种类和接口与几万个方法。幸运的是，并不需要知道它们中的每一个，然而，要想 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性编程语言。

从理论上讲，这是完全有可能的。的确，除了微软之外的每一个厂商都希望如此。然而，有很多在桌面计算机上已经工作良好的应用程序，在其他设备上或浏览器中或许不能正常地工作。同时，在编写这些应用程序时，利用了相应处理器的速度和本地的用户接口库，而且它们已经移植到所有重要的平台上。这类应用程序包括：字处理程序、图片编辑器以及 Web 浏览器。它们通常是用 C 或 C++ 编写的，采用 Java 语言重新编写似乎对最终的用户不会带来什么特别的好处。

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++ 或 C#。有上百种好的程序设计语言没有广泛地流行，而带有明显缺陷的语言，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撑系统的能力，而不是优美的语

法。人们主要关注：是否提供了易于实现某些功能的易用、便捷和标准的库？是否拥有强大的程序设计能力与调试工具？语言和工具是否能够与计算机的其他基础结构整合在一起？Java 的成功源于其类库能够让人们轻松地完成原本有一定难度的事情。例如：联网和多线程。Java 减少了指针错误，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）现在有了 C#，Java 过时了。

C# 借鉴了 Java 许多好的思想，例如：清晰的语言结构、虚拟机和垃圾收集。无论怎样，C# 还是省去了一些好的特性，其中最重要的是安全性和平台无关性。如果确定使用 Windows，C# 就更有意义。但是，从求职广告判定，Java 仍然是大多数开发者选择的语言。

7）Java 是专用的，应该避免使用。

Sun Microsystems 负责将 Java 的许可发放给销售者以及最终用户。尽管 Sun 公司通过 Java Community Process 最终控制着 Java，但他们同时与许多其他的公司联手一起进行着语言修订版的开发及新类库的设计。虚拟机和类库的源代码都可以免费获取，但是，只能查阅，不能修改，也不能再发布。因此，Java 是“闭源，但运转良好”。

这种状况在 2007 年发生了戏剧性的变化，Sun 声称 Java 未来的版本将在 General Public License 下可用。Linux 使用的是同一个开放源代码许可。Oracle 一直致力于保持 Java 开源。只有一点美中不足——专利。根据 GPL，任何人都可以得到专利许可，允许其使用和修改 Java，不过仅限于桌面和服务器平台。如果你想在嵌入式系统中使用 Java，就需要另外一个不同的许可，这很可能需要付费。不过，这些专利在未来十年就会到期，那时 Java 就完全免费了。

8）Java 是解释型的，因此对于关键的应用程序速度太慢了。

早期的 Java 是解释型的。现在除了像手机这样的“微型”平台之外，Java 虚拟机使用了即时编译器，因此采用 Java 编写的“热点”代码其运行速度与 C++ 相差无几。

Java 有一些 C++ 没有的额外开销。虚拟机的启动时间要慢一些，并且 Java GUI 要比本地的 GUI 慢一些，这是因为它们采用了与平台无关的绘图方式。

对于 Java 比 C++ 慢，人们已经抱怨很多年了。但是，今天的计算机速度远比人们发出抱怨的时候快了很多。一个较慢的 Java 程序与几年前相当快的 C++ 程序相比还要快一些。就这一点来说，那些抱怨听起来有点像狐狸抱怨葡萄酸，有些人已经转过来攻击 Java 用户界面不够漂亮而不再攻击速度慢了。

9）所有的 Java 程序都是在网页中运行的。

所有的 Java applet 都是在网页浏览器中运行的。这也恰恰是 applet 的定义，即一种在网页中运行的 Java 程序。然而，大多数 Java 程序是运行在 Web 浏览器之外的独立应用程序。实际上，很多 Java 程序都在 Web 服务器上运行并生成用于网页的代码。

10）Java 程序是主要的安全风险。

早期的 Java，有过关于安全系统失效的报道，曾经一度引起公众哗然。大多数安全问题都存在于 Java 的特定浏览器中。研究人员将这视为一种挑战，即努力找出 Java 的漏洞，对 applet 安全模型的强度和复杂度发起挑战。随后，人们很快就解决了引发问题的所有技术因

素。据我们所知，任何实用系统都有安全危机。想想看：毫不夸张地说，有数百万种病毒攻击着 Windows 的可执行文件和 Word 宏，这给系统造成了巨大的损害，但却很少有人批评被攻击平台的脆弱。同样，Internet Explorer 中的 ActiveX 机制始终作为被攻击的目标，但由于阻止这种攻击非常简单，所以人们也就懒得将它们公布于众了。

有些系统程序员在公司的浏览器中禁用 Java，却允许其用户下载可执行文件、ActiveX 控件和 Word 文档。这是多么荒唐可笑啊！事实上这会带来更大的风险。尽管距离 Java 的诞生已经 15 年之久，但与其他常用的执行平台相比，还是 Java 安全得多。

11）JavaScript 是 Java 的简易版。

JavaScript 是一种在网页中使用的脚本语言，它是由 Netscape 发明的，原来的名字叫做 LiveScript。JavaScript 的语法类似 Java，除此之外，两者无任何关系。当然，名字有些相像。JavaScript 的一个子集已经标准化为 ECMA-262。与 Java applet 相比，JavaScript 更紧密地与浏览器集成在一起。特别是 JavaScript 程序可以修改正在显示的文档，而 applet 只能在有限的区域内控制外观。

12）使用 Java 可以用价值 500 美元的 Internet 设备取代电脑。

当 Java 刚刚发布的时候，一些人打赌：肯定会有这样的好事情发生。从本书的第 1 版开始，我们就已经认定“家庭用户将会放弃功能强大且便利的桌面系统，而使用没有本地存储的网络设备”是一种荒谬的想法。我们发现基于 Java 的网络计算机，对利用“零管理”降低计算机所有者的商业成本是一种很好的选择。即便如此，这种好事也没有发生。新一代的平板电脑并没有使用 Java。

第 2 章　Java 程序设计环境

▲ 安装 Java 开发工具箱　　▲ 使用集成开发环境
▲ 选择开发环境　　▲ 运行图形化应用程序
▲ 使用命令行工具　　▲ 建立并运行 applet

本章主要介绍如何安装 Java 开发工具箱（JDK）以及如何编译和运行各种类型的程序：控制台程序、图形化应用程序以及 applet 应用程序。运行 JDK 的方法是在 shell 窗口中键入命令行。然而，很多程序员更喜欢使用集成开发环境。为此，将在稍后介绍如何使用免费的开发环境编译和运行 Java 程序。尽管学起来很容易，但集成开发环境需要吞噬大量资源，在编写小型程序时会给人带来烦恼。作为折中方案，再介绍一下如何调用 Java 编译器并运行 Java 程序的文本编辑器。一旦掌握了本章的技术，并选定了自己的开发工具，就可以学习第 3 章，开始研究 Java 程序设计语言。

2.1　安装 Java 开发工具箱

Oracle 公司为 Linux、Mac OS X、Solaris 和 Windows 提供了 Java 开发工具箱（JDK）的最新、最完整的版本。用于很多其他平台的版本仍处于各种不同的开发状态中，不过，这些版本都是由相应平台的开发商授权并分发的。

2.1.1　下载 JDK

要想下载 Java 开发工具箱，必须访问 Oracle 网站，地址：www.oracle.com/technetwork/java/javase/downloads，并且在得到所需的软件之前必须弄清楚大量的专业术语。请看表 2-1。

表 2-1　Java 术语

术 语 名	缩　写	解　释
Java Development Kit	JDK	编写 Java 程序的程序员使用的软件
Java Runtime Environment	JRE	运行 Java 程序的用户使用的软件
Standard Edition	SE	用于桌面或简单的服务器应用的 Java 平台
Enterprise Edition	EE	用于复杂的服务器应用的 Java 平台
Micro Edition	ME	用于手机和其他小型设备的 Java 平台
Java 2	J2	一个过时的术语，用于描述 1998 年 ~2006 年之间的 Java 版本
Software Development Kit	SDK	一个过时的术语，用于描述 1998 年 ~2006 年之间的 JDK
Update	u	Oracle 的术语，用于发布修改的 bug
NetBeans	—	Oracle 的集成开发环境

JDK 是 Java Development Kit 的缩写。有点混乱的地方是：工具箱的版本 1.2~ 版本 1.4 被称为 Java SDK（Software Development Kit）。在某些场合下，还可以看到这些过时的术语。另外，还有 Java 运行时环境（JRE），它包含虚拟机但不包含编译器。这并不是开发者所想要的环境，而是专门为不需要编译器的用户而设计。

还有，随处可见的 Java SE，相对于 Java EE（Enterprise Edition）和 Java ME（Micro Edition），它是 Java 的标准版。

Java 2 这种提法始于 1998 年。当时 Sun 公司的销售人员感觉通过增加小数点后面的数值改变版本号并没有反映出 JDK 1.2 的重大改进。但是，由于在发布之后才意识到这个问题，所以决定将开发工具箱的版本号仍然沿用 1.2，接下来的版本就是 1.3、1.4 和 5.0。但是，Java 平台被重新命名为 Java 2。因此，就有了 Java 2 Standard Edition Software Development Kit 的 5.0 版，即 J2SE SDK 5.0。

对于工程师来说，所有这一切都可能会引起困惑，这正是没有将其投入市场的原因。2006 年，日趋完善的 Java SE 开始流行。无意义的 Java 2 被遗弃，Java 当前的标准版本被称为 Java SE 6。偶尔还会看到使用 1.5 版本和 1.6 版本，但这些只是 5.0 版本和 6 版本的同义词。

最后，当 Oracle 为解决一些紧急问题做出了某些微小的版本改变时，将其称为更新。例如：对 Java SE 7 开发包做出的第一次更新，正式称为 JDK 7u1，内部的版本号为 1.7.0_01。更新不需要安装在前一个版本上，它将包含整个 JDK 的最新版本。

Oracle 公司曾经制作过将 Java 开发工具箱和集成开发环境捆绑在一起的产品。其中的集成开发环境，在不同时期被命名为不同的名字，例如，Forte、Sun ONE Studio、Sun Java Studio 和 NetBeans。我们无法知道每个人在登录下载网站时，市场正在热销什么。这里，建议大家只安装 Java 开发工具箱。如果最终决定使用 Oracle 的集成开发环境，只需要从 http://netbeans.org 下载。

注释：安装过程提供了包含 JDK 版本号（如 jdk1.7.0）的默认的安装路径。这似乎有些烦人，但是，应该重视版本号，它会给安装新版 JDK 的测试带来便利。

在 Windows 环境下，强烈建议不要接受带空格的默认路径名，如：c:\ProgramFiles\jdk1.7.0。应该将 Program Files 部分删掉。

在本书中，使用的安装路径是 jdk。例如：当引用 jdk/bin 目录时，意味着引用的是 /usr/ local/jdk1.7.0/bin 或 c:\ jdk1.7.0\bin。

2.1.2 设置执行路径

在完成了 JDK 的安装之后，还需要执行另外一个步骤：把 jdk/bin 目录添加到执行路径中。所谓执行路径是指操作系统搜索本地可执行文件的目录列表。对于不同的操作系统，这个步骤的操作过程有所不同。

- 在 UNIX（包括 Linux、Mac OS X 和 Solaris）环境下，编辑执行路径的过程与所使用的 shell 有关。如果使用的是 Bourne Again shell（Linux 的默认选择），在 ~/.bashrc 或

~/.bash_profile 文件的末尾就要添加：

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

- 在 Windows 下，以管理员身份登录。启动 Control Panel，切换到 Classic View，并选择 System 图标。在 Windows XP 中，立即会看到 System Properties 对话框。在 Vista 和 Windows 7 中，需要选择 Advanced 系统设置（如图 2-1 所示）。在 System Properties 对话框中，点击 Advanced 标签，然后点击 Environment 按钮。滚动 System Variables 窗口直到找到变量名 path 为止。点击 Edit 按钮（如图 2-2 所示）。将 jdk\bin 目录添加到路径的开始处，用分号将新条目隔开，如下所示：

```
jdk\bin;other stuff
```

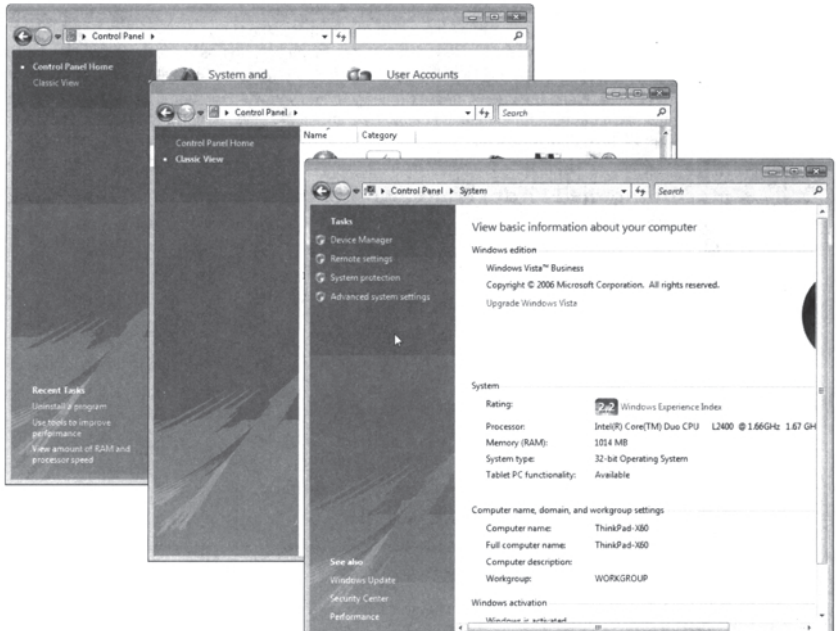

图 2-1 在 Windows Vista 中启动的 System Properties 对话框

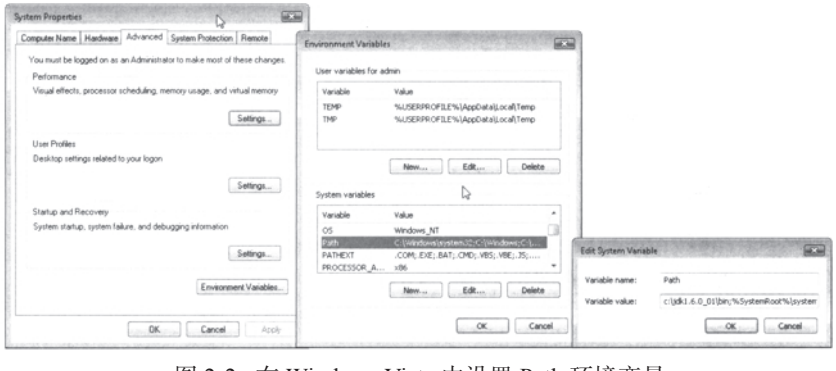

图 2-2 在 Windows Vista 中设置 Path 环境变量

要注意，需要将 *jdk* 替换为你的 Java 安装的实际路径，如 c:\jdk1.7.0_02。如果忽视我们的建议，想要保留 Program Files 目录，则要把整个路径用双引号引起来，如：“ c:\Program Files\jdk 1.7.0_02\bin”;other stuff。

将这个设置保存起来，新打开的任何控制台窗口都会有正确的路径。

下面是测试上述设置是否正确的方法：打开一个 shell 窗口，键入：

```
javac -version
```

然后，按 ENTER 键。应该能够看到下面的显示信息：

```
javac 1.7.0_02
```

如果看到的是“ java:command not found”、或“ The name specified is not recognized as an internel or external command, operable program or batch file”，则需要回到前面，重新检查整个安装过程。

注释：在 Windows 中，按照下面的命令打开 shell 窗口。如果使用 Windows XP 环境，那么在开始菜单中选择 Run 选项，并键入 cmd。在 Vista 和 Windows 7 中，只在开始菜单中的 Start Search 中输入 cmd，再按下 ENTER 键就会出现一个 shell 窗口。

如果以前没有接触过这些内容，建议学习一下有关命令行的基础教程。例如 http://www.horstmann.com/bigj/help/windows/tutorial. html。

2.1.3 安装库源文件和文档

库源文件在 JDK 中以一个压缩文件 src.zip 的形式发布，必须将其解压缩后才能够访问源代码。这里强烈地建议按照下面所述的步骤进行操作。很简单：

1）确保 JDK 已经安装，并且 jdk/bin 目录在执行路径中。

2）打开 shell 窗口。

3）进入 jdk 目录（例如：cd /usr/local/jdk1.7.0 或 cd c:\jdk1.7.0 ）。

4）建立一个子目录 src

```
mkdir src
cd src
```

5）执行命令：

```
jar xvf ../src.zip
```

（或者在 Windows 中执行 jar xvf ..\src.zip。）

提示：在 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、本地方法以及私有辅助类），请访问网站：http://jdk7.java.net。

文档包含在一个压缩文件中，它是一个独立于 JDK 的压缩文件。可以直接从网站 http://www.oracle.com/technetwork/java/javase/downloads 下载获得这个文档。操作步骤如下：

1）确认 JDK 已经安装，并且 jdk/bin 目录在执行路径上。

2）下载文档压缩文件并将其存放在 jdk 目录下。这个文件名为 jdk-*version*-apidocs.zip，

其中的 *version* 表示版本号，例如 7。

3）打开一个 shell 窗口。

4）进入 jdk 目录。

5）执行命令：

```
jar xvf jdk-version-apidocs.zip
```

其中 *version* 是相应的版本号。

2.1.4 安装本书中的示例

读者可以安装本书中的程序示例。这些程序可以从 http://horstmann.com/corejava 下载，它们都打包在 corejava.zip 文件中。应该将它们解压到一个单独的文件夹中，建议将文件夹命名为 CoreJavaBook。需要执行下列步骤：

1）确保 JDK 已经安装，并且 jdk/bin 目录在执行路径中。

2）建立目录 CoreJavaBook。

3）将 corejava.zip 下载到这个目录下。

4）打开一个 shell 窗口。

5）进入 CoreJavaBook 目录。

6）执行命令：

```
jar xvf corejava.zip
```

2.1.5 导航 Java 目录

在学习 Java 的过程中，经常需要查看 Java 源文件。当然，也会频繁地使用类库文档。图 2-3 显示了 JDK 目录树。

目录结构	描　述
jdk	（名字可能不同，例如：jdk1.7.0_02）
├─*bin*	编译器和工具
├─*demo*	演示
├─*docs*	HTML 格式的类库文档（解压 j2sdk*version*-doc.zip 之后）
├─*include*	用于编译本地方法的文件（参看卷 II）
├─*jre*	Java 运行环境文件
├─*lib*	类库文件
└─*src*	类库源文件（解压 src.zip 之后）

图 2-3　JDK 目录树

就学习 Java 而言，docs 和 src 是两个最有用的子目录。docs 目录包含了 HTML 格式的类库文档，可以使用任何浏览器（如：Firefox）查看这些文档。

提示：在浏览器中设置一个指向 docs/api/index.html 的书签。使用 Java 平台时经常需要

查看这一页的内容。

src 目录包含了 Java 类库中公共部分的源代码。当对 Java 熟悉到一定程度时，可能会感到本书以及联机文档已经无法提供所需的信息了。那时，应该深入研究 Java 的源代码。请放心，只要深入地研究源代码就一定会搞清楚类库函数的真正功能。例如，如果对 System 类的内部工作感到好奇，可以查看 src/java/lang/System.java。

2.2 选择开发环境

如果在此之前使用 Microsoft Visual Studio 编写过程序，那么就会习惯于带有内嵌的文本编辑器、用于编译和运行程序的菜单，以及配有集成调试器的开发环境。基本的 JDK 全然没有这些功能。利用它完成每一项任务时都要在 shell 窗口中键入命令。这些听起来很麻烦，但只是一个基本的技能。第一次安装 Java 时，可能希望在安装开发环境之前检测一下安装的 Java 是否正确。此外，还可以通过执行一些基本的操作步骤，加深对开发环境幕后工作的理解。

然而，在掌握了编译和运行 Java 程序的基本步骤之后，你就想要使用专业的开发环境了。在过去的 10 年中，这些环境变得功能非常强大，操作也非常方便，以至于不选用它们将是极其不理智的。使用 Eclipse 和 NetBeans 这两个免费的开发环境是很好的选择。尽管 NetBeans 发展的速度比较快，但在本章中，还是打算介绍如何使用 Eclipse，这是因为 Eclipse 要比 NetBeans 更加灵活一些。当然，如果倾向于使用其他的开发环境，同样也可以完成本书的所有程序。

过去，推荐使用文本编辑器编写简单的程序，如：Emacs、JEdit 或者 TextPad。现在不会再这样推荐了，因为集成开发环境非常快捷、方便。

总之，应当了解如何使用基本的 JDK 工具，这样才会感觉使用集成开发环境是一种享受。

2.3 使用命令行工具

首先介绍较难的一种方法：通过命令行编译并运行 Java 程序。

1）打开一个 shell 窗口。

2）进入 CoreJavaBook/v1ch02/Welcome 目录（CoreJavaBook 是安装本书示例源代码的目录，请参看“安装本书中的示例”一节）。

3）键入下面的命令：

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

然后，将会在 shell 窗口中看到图 2-4 所示的输出。

祝贺你！已经编译并运行了第一个 Java 程序。

那么，刚才都进行了哪些操作呢？ javac 程序是一个 Java 编译器。它将文件 Welcom.java 编译成 Welcom.class，并发送到 Java 虚拟机。虚拟机执行编译器放在 class 文件中的字节码。

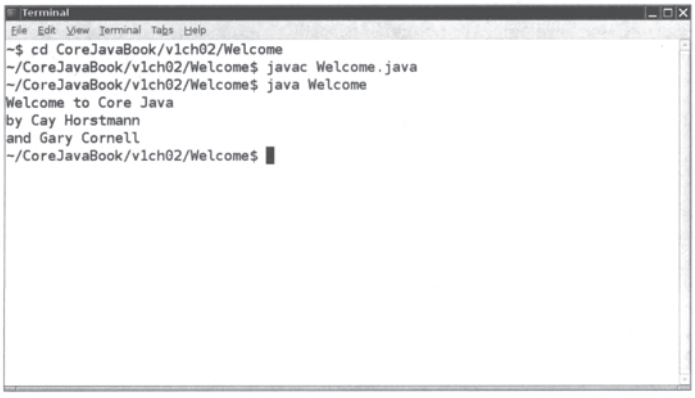

图 2-4 编译并运行 Welcome.java

注释：如果得到了下面这行语句的错误消息

```
for (String g : greeting)
```

就可能是使用了 Java 编译器的旧版本。如果使用的是 Java 旧版本，则需要将上面的循环语句改写成下列形式：

```
for (int i = 0; i < greeting.length; i++)
   System.out.println(greeting[i]);
```

Welcome 程序非常简单。其功能只是向控制台输出了一条消息。你可能想查看程序清单 2-1 的程序代码。（在下一章中，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 from the authors.
3   * @version 1.20 2004-02-28
4   * @author Cay Horstmann
5   */
6  public class Welcome
7  {
8     public static void main(String[] args)
9     {
10       String[] greeting = new String[3];
11       greeting[0] = "Welcome to Core Java";
12       greeting[1] = "by Cay Horstmann";
13       greeting[2] = "and Gary Cornell";
14
15       for (String g : greeting)
16          System.out.println(g);
17    }
18 }
```

疑难解答提示

在使用可视化开发环境的年代，许多程序员对于在 shell 窗口中运行程序已经很生疏了。

常常会出现很多错误，最终导致令人沮丧的结果。

一定要注意以下几点：

- 如果手工地输入源程序，一定要注意大小写。尤其是类名为 Welcome，而不是 welcome 或 WELCOME。
- 编译时需要提供一个文件名（Welcome.java），而运行时，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 这类消息，就要返回去检查一下安装是否出现了问题，特别是执行路径的设置。
- 如果 javac 报告了一个错误“ cannot read ： Welcome.java”，就应该检查一下目录中是否存在这个文件。

在 UNIX 环境下，检查 Welcome.java 是否以正确的大写字母开头。在 Windows 环境下，使用 shell 命令 dir，而不要使用图形浏览器工具。有些文本编辑器（特别是 Notepad）在每个文件名后面要添加扩展名 .txt。如果使用 Notepad 编辑 Welcome.java 就会存为 Welcome.java.txt。对于默认的 Windows 设置，浏览器与 Notepad 都隐含 .txt 的扩展名，这是因为这个扩展名属于“已知的文件类型”。此时，需要重新命名这个文件，使用 shell 命令 ren，或是另存一次，给文件名加一对双引号，如：“Welcome.java”。

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

  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，就是错误地键入了 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，就应该检查一下是否系统的 CLASSPATH 环境变量被别人更改了（将这个变量设置为全局并不是一个提倡的做法，然而，Windows 中有些编写很差的软件安装程序就是这样做的）。可以在当前的 shell 窗口中键入下列命令，临时地取消 CLASSPATH 环境变量的设置：

  ```
  set CLASSPATH=
  ```

  这个命令应用于使用 C shell 的 Windows 和 UNIX/Linux 环境下。在使用 Bourne/bash shell 的 UNIX/Linux 环境下需要使用：

  ```
  export CLASSPATH=
  ```

提示：在 http://docs.oracle.com/javase/tutorial/getStarted/cupojava/ 上有一个很好的教程。其中提到了初学者经常容易犯的一些错误。

2.4 使用集成开发环境

本节将介绍如何使用 Eclipse 编译一个程序。Eclipse 是一个可以从网站 http://eclipse.org

上免费下载得到的集成开发环境。Eclipse 是采用 java 编写的，然而，由于所使用的是非标准视窗类库，所以，不如 Java 那样具有可移植性。尽管如此，这个开发环境已经拥有可以应用在 Linux、Mac OS X、Solaris 以及 Windows 的版本了。

还有一些使用比较普遍的 IDE，但就目前而言，Eclipse 是最通用的。使用步骤如下所示：

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）。这些是 Eclipse 3.2 的屏幕画面。如果使用的 Eclipse 版本稍有差异，不必介意。

3）点击 Next 按钮，键入工程名 Welcome，并输入包含 Welcome.java 的完整路径名。如图 2-6 所示。

图 2-5 Eclipse 中的“New Project”对话框

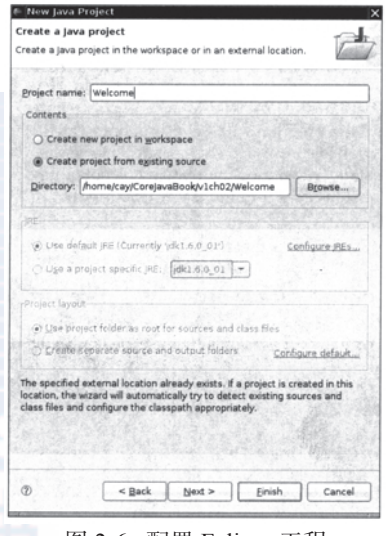

图 2-6 配置 Eclipse 工程

4）确保选中标有 Create project form existing source 的选项。

5）点击 Finish 按钮。这个工程已经创建完成了。

6）点击在工程窗口左边窗格中的三角，并打开它。然后，点击 Default package 旁边的三角，再双击 Welcome.java。现在应该看到带有程序代码的窗口了（如图 2-7 所示）。

7）用鼠标右键点击最左侧窗格中的工程名（Welcome），选择 Run → Run As → Java Application。可以看到：这个窗口底部出现了一个输出窗口。在这个输出窗口中显示了程序的输出结构（如图 2-8 所示）。

定位编译错误

可以肯定，这个程序没有出现输入错误或 bug（毕竟，这段代码只有几行）。为了说明问题，假定在代码中不小心出现了录入错误（或语法错误）。试着将原来的程序修改一下，让它包含一些录入错误，例如，将 String 的大小写弄错：

```
String[] greeting = new string[3];
```

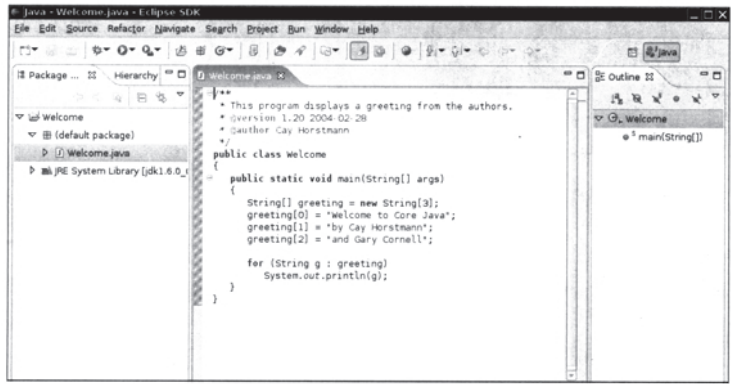

图 2-7 使用 Eclipse 编辑源文件

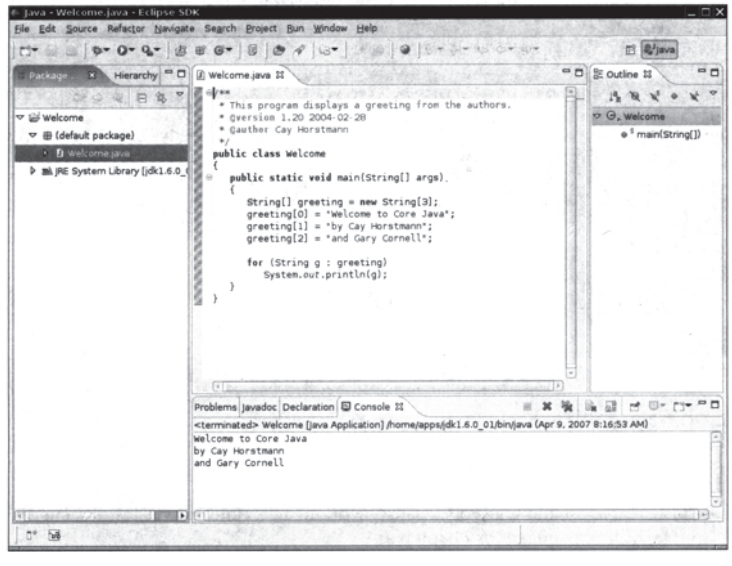

图 2-8 在 Eclipse 中运行程序

现在，重新编译这个程序，就会收到一条错误消息，其中指出了有一个不能识别的 string 类型（如图 2-9 所示）。点击这个错误消息，可以看到光标马上移到了编辑窗口中相应的行上，在这里将错误纠正过来。使用这种方式可以快速地纠正错误。

提示：通常，Eclipse 错误报告会伴有一个灯泡的图标。点击这个图标可以得到一个建议解决这个错误的方案列表。

通过这些介绍，你应该对集成环境已经有些认识了。我们将在第 11 章讨论 Eclipse 调试工具。

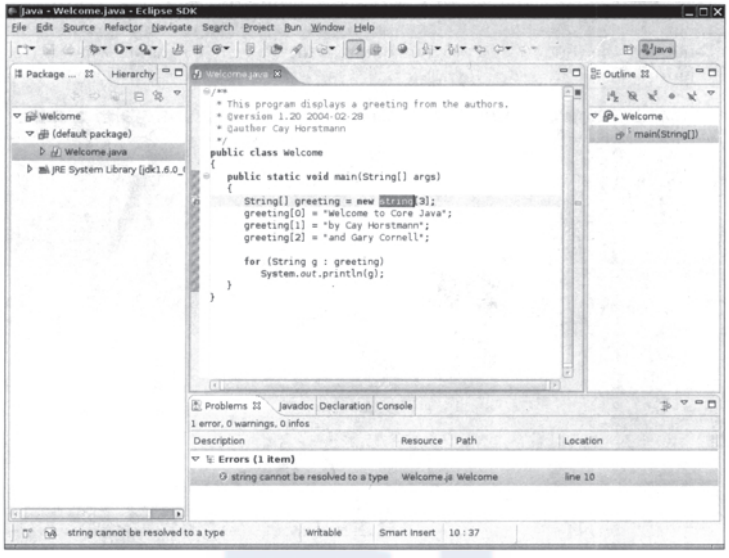

图 2-9 Eclipse 的错误消息

2.5 运行图形化应用程序

Welcome 程序并不会引起人们的兴奋。接下来，给出一个图形化应用程序。这个程序是一个简单的图像文件查看器（viewer），它可以加载并显示一个图像。首先，由命令行编译并运行这个程序。

1）打开一个 shell 窗口。

2）进入 CoreJavaBook/v1ch02/ImageViewer。

3）输入：

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

运行后将弹出一个标题栏为 ImageViewer 的新程序窗口（如图 2-10 所示）。

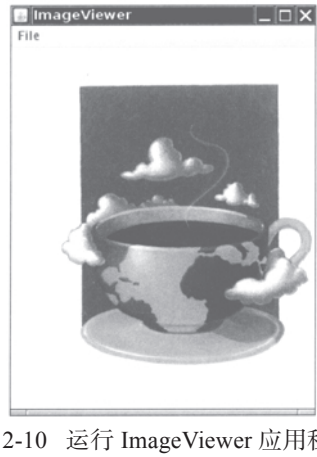

图 2-10 运行 ImageViewer 应用程序

现在，选择 File → Open，然后找到一个图像文件并打开它（我们在同一个目录下提供了两个示例文件）。要想关闭这一程序，只需要点击标题栏中的关闭按钮或者从菜单中选择 File → Exit。

下面快速地浏览一下源代码。这个程序比第一个程序要长很多，但是只要想一想用 C 或 C++ 编写同样功能的应用程序所需要的代码量，就不会感到它太复杂了。当然，在 Visual Basic 中，编写（或者更确切地说“拖放”

出）这个程序相当简单。JDK 没有可视化的界面构造器，所以必须通过编写代码完成这一切工作，如程序清单 2-2 所示。本书将在第 7 章～第 9 章介绍编写图形化应用程序的内容。

程序清单 2-2 ImageViewer/ImageViewer.java

```
import java.awt.EventQueue;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

/**
 * A program for viewing images.
 * @version 1.22 2007-05-21
 * @author Cay Horstmann
 */
public class ImageViewer
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               JFrame frame = new ImageViewerFrame();
               frame.setTitle("ImageViewer");
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}

/**
 * A frame with a label to show an image.
 */
class ImageViewerFrame extends JFrame
{
   private JLabel label;
   private JFileChooser chooser;
   private static final int DEFAULT_WIDTH = 300;
   private static final int DEFAULT_HEIGHT = 400;
   public ImageViewerFrame()
   {
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // use a label to display the images
      label = new JLabel();
      add(label);

      // set up the file chooser
      chooser = new JFileChooser();
      chooser.setCurrentDirectory(new File("."));

      // set up the menu bar
```

```
30        JMenuBar menuBar = new JMenuBar();
31        setJMenuBar(menuBar);
32
33        JMenu menu = new JMenu("File");
34        menuBar.add(menu);
35
36        JMenuItem openItem = new JMenuItem("Open");
37        menu.add(openItem);
38        openItem.addActionListener(new ActionListener()
39           {
60              public void actionPerformed(ActionEvent event)
61              {
62                 // show file chooser dialog
63                 int result = chooser.showOpenDialog(null);
64
65                 // if file selected, set it as icon of the label
66                 if (result == JFileChooser.APPROVE_OPTION)
67                 {
68                    String name = chooser.getSelectedFile().getPath();
69                    label.setIcon(new ImageIcon(name));
70                 }
71              }
72           });
73
74        JMenuItem exitItem = new JMenuItem("Exit");
75        menu.add(exitItem);
76        exitItem.addActionListener(new ActionListener()
77           {
78              public void actionPerformed(ActionEvent event)
79              {
80                 System.exit(0);
81              }
82           });
83     }
84  }
```

2.6 建立并运行 applet

本书给出的前两个程序是 Java 应用程序。它们与所有本地程序一样，可以独立地运行。然而，正如第 1 章提到的，有关 Java 的大量宣传都在炫耀 Java 能够在浏览器中运行 applet 的能力。下面介绍一下如何利用命令行建立并运行 applet。然后，利用 JDK 自带的 applet 查看器加载 applet。最后，在 Web 浏览器中显示。

首先，打开 shell 窗口并将目录转到 CoreJavaBook/v1ch02/WelcomeApplet，然后，输入下面的命令：

```
javac WelcomeApplet.java
appletviewer WelcomeApplet.html
```

图 2-11 显示了在 applet 查看器窗口中显示的内容。

第一条命令是大家已经非常熟悉的调用 Java 编译器的命令。它将 WelcomeApplet.java 源

文件编译成字节码文件 WelcomeApplet.class。

图 2-11 在 applet 查看器窗口中显示的 WelcomeApplet

不过这一次不要运行 Java 程序，而调用 appletviewer 程序。这是 JDK 自带的一个特殊工具，它可以帮助人们快速地测试 applet。这里需要向这个程序提供一个 HTML 文件名，而不是 Java 类文件名。WelcomeApplet.html 的内容请参看下面的程序清单 2-3。

程序清单 2-3 WelcomeApplet/WelcomeApplet.html

```
<html>
   <head>
      <title>WelcomeApplet</title>
   </head>
   <body>
      <hr/>
      <p>
         This applet is from the book
         <a href="http://www.horstmann.com/corejava.html">Core Java</a>
         by <em>Cay Horstmann</em> and <em>Gary Cornell</em>.
      </p>
      <applet code="WelcomeApplet.class" width="400" height="200">
         <param name="greeting" value ="Welcome to Core Java!"/>
      </applet>
      <hr/>
      <p><a href="WelcomeApplet.java">The source.</a></p>
   </body>
</html>
```

当然，applet 就是要在浏览器中查看。遗憾的是，很多浏览器默认情况下并没有启用 Java 支持。http://java.com/en/download/help/enable_browser.xml 介绍了如何配置一些最常用的浏览器，使它们支持 Java。一旦完成浏览器的配置，可以在浏览器中试着加载 applet。

1）启动浏览器。

2）选择 File → Open File（或等效的操作）。

3）进入 CoreJavaBook/v1ch02/WelcomeApplet 目录。在文件对话框中，将会看到 WelcomeApplet.html 文件。加载这个文件。

4）浏览器将加载 applet，并显示文字。显示效果基本上如图 2-12 所示。

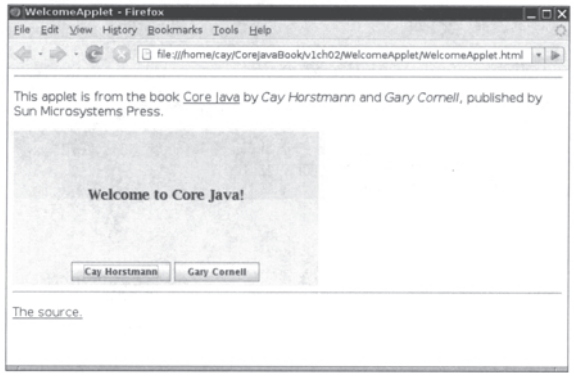

图 2-12 在浏览器中运行 WelcomeApplet

读者可能会发现这个应用程序是活动的，并且可以与 Internet 进行交互。点击 Cay Horstmann 按钮，applet 会让浏览器显示 Cay 的网页。点击 Gray Cornell 按钮，applet 会让浏览器弹出一个邮件窗口，其中包含已经填写好的 Gray 的邮件地址。

需要注意的是，在查看器中的两个按钮并不起作用。applet 查看器没有能力发送邮件或显示一个网页，因此会忽略这里的操作请求。applet 查看器适于用来单独地测试 applet，但是，最终需要将 applet 放到浏览器中，以便检测与浏览器以及 Internet 的交互情况。

提示：也可以从集成开发环境内部运行 applet。在 Eclipse 中，使用 Run → Run as → Java Applet 菜单选项。

最后，在程序清单 2-4 中给出了这个 applet 的代码。现在，只要阅读一下就可以了。在第 10 章中，还会再次介绍 applet 的编写。

程序清单 2-4 WelcomeApplet/WelcomeApplet.java

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.net.*;
4  import javax.swing.*;
5  /**
6   * This applet displays a greeting from the authors.
7   * @version 1.22 2007-04-08
8   * @author Cay Horstmann
9   */
10 public class WelcomeApplet extends JApplet
11 {
12    public void init()
13    {
14       EventQueue.invokeLater(new Runnable()
15          {
16             public void run()
17             {
18                setLayout(new BorderLayout());
19
```

```
            JLabel label = new JLabel(getParameter("greeting"), SwingConstants.CENTER);
            label.setFont(new Font("Serif", Font.BOLD, 18));
            add(label, BorderLayout.CENTER);

            JPanel panel = new JPanel();

            JButton cayButton = new JButton("Cay Horstmann");
            cayButton.addActionListener(makeAction("http://www.horstmann.com"));
            panel.add(cayButton);

            JButton garyButton = new JButton("Gary Cornell");
            garyButton.addActionListener(makeAction("mailto:gary_cornell@apress.com"));
            panel.add(garyButton);

            add(panel, BorderLayout.SOUTH);
         }
      });
   }

   private ActionListener makeAction(final String urlString)
   {
      return new ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               try
               {
                  getAppletContext().showDocument(new URL(urlString));
               }
               catch (MalformedURLException e)
               {
                  e.printStackTrace();
               }
            }
         };
   }
}
```

在本章中，我们学习了有关编译和运行 Java 程序的机制。现在可以转到第 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 应用程序
- ▲ 注释
- ▲ 数据类型
- ▲ 变量
- ▲ 运算符
- ▲ 字符串
- ▲ 输入输出
- ▲ 控制流
- ▲ 大数值
- ▲ 数组

现在，假定已经成功地安装了 JDK，并且能够运行第 2 章中给出的示例程序。我们从现在开始将介绍 Java 应用程序设计。本章主要讲述程序设计相关的基本概念（如数据类型、分支以及循环）在 Java 中的实现方式。

非常遗憾，需要告诫大家，使用 Java 编写 GUI 应用程序并不是一件很容易的事情，编程者需要掌握很多相关的知识才能够创建窗口、添加文本框和按钮等。介绍基于 GUI 的 Java 应用程序设计技术与本章将要介绍的程序设计基本概念相差甚远，因此本章给出的所有示例都是为了说明一些相关概念而设计的“玩具式”程序，它们仅仅通过 shell 窗口输入输出。

最后需要说明，对于一个具有 C++ 编程经验的程序员来说，本章的内容只需要浏览一下，应该重点阅读分布在正文中的 C/C++ 注释。对于具有使用 Visual Basic 等其他编程背景的程序员来说，可能会发现其中的绝大多数概念都很熟悉，但是在语法上有比较大的差异，因此，需要非常仔细地阅读本章的内容。

3.1　一个简单的 Java 应用程序

下面看一个最简单的 Java 应用程序，它只发送一条消息到控制台窗口中：

```
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，但所有的 Java 应用程序都具有这种结构，还是值得花一些时间研究一下。首先，Java 对大小写敏感。如果出现了大小写拼写错误（例如，将 main 拼写成 Main），那程序将无法运行。

下面逐行地查看一下这段源代码。关键字 public 称为访问修饰符（access modifier），它用于控制程序的其他部分对这段代码的访问级别。在第 5 章中将会更加详细地介绍访问修饰符的具体内容。关键字 class 表明 Java 程序中的全部内容都包含在类中。这里，只需要将类作

为一个加载程序逻辑的容器，程序逻辑定义了应用程序的行为。在第 4 章中将会用大量的篇幅介绍 Java 类。正如第 1 章所述，类是构建所有 Java 应用程序和 applet 的构建块。Java 应用程序中的全部内容都必须放置在类中。

关键字 class 后面紧跟类名。Java 中定义类名的规则很宽松。名字必须以字母开头，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（保留字列表请参看附录）。

从类名 FirstSample 可以看出，标准的命名规范为：类名是以大写字母开头的名词。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称为骆驼命名法。以其自身为例，应该写成 CamelCase）。

源代码的文件名必须与公共类的名字相同，并用 .java 作为扩展名。因此，存储这段源代码的文件名必须为 FirstSample.java（再次提醒大家注意，大小写是非常重要的，千万不能写成 firstsample.java）。

如果已经正确地命名了这个文件，并且源代码中没有任何录入错误，在编译这段源代码之后就会得到一个包含这个类字节码的文件。Java 编译器将字节码文件自动地命名为 FirstSample.class，并与源文件存储在同一个目录下。最后，使用下面这行命令运行这个程序：

```
java FirstSample
```

（请记住，不要添加 .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

```
java ClassName
```

运行编译程序时，Java 虚拟机将从指定类中的 main 方法开始执行（这里的“方法”就是 Java 中所说的“函数”），因此为了代码能够执行，在类的源文件中必须包含一个 main 方法。当然，也可以将用户自定义的方法添加到类中，并且在 main 方法中调用它们（第 4 章将讲述如何自定义方法）。

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. 上阅读或下载）。

不过，当 main 方法不是 public 时，有些版本的 Java 解释器也可以执行 Java 应用程序。有个程序员报告了这个 bug。如果感兴趣的话，可以在网站 http://bugs.sun.com/bugdatabase/ index.jsp 上输入 bug 号码 4252539 查看一下。这个 bug 被标明“关闭，不予修复。”Sun 公司的工程师解释说：Java 虚拟机规范（在 http://docs.oracle.com/javase/specs/jvms/se7/html）并没有要求 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 SE 1.4 及以后的版本中将强制 main 方法是 public 的。

从上面这段话可以发现一个问题的两个方面。一方面让质量保证工程师判断在 bug 报告中是否存在问题是一件很头痛的事情，这是因为其工作量很大，并且工程师对 Java

的所有细节也未必了解得很清楚。另一方面，Sun 公司把 bug 报告及其解决方案放到网站上让所有人监督检查，这是一种非常了不起的举动。“bug 展示”对程序员来说是一种十分有用的资源，甚至程序员可以对感兴趣的 bug 进行“投票”。得票多的 bug 在下一个将要发布的 JDK 版本中得到解决的可能性就大。

需要注意源代码中的括号 { }。在 Java 中，像在 C/C++ 中一样，用花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用“{”开始，用“}”结束。

花括号的使用风格曾经引发过许多无谓的争论。我们的习惯是把匹配的花括号上下对齐。不过，由于空白符会被 Java 编译器忽略，所以可以选用自己喜欢的任意风格。在下面讲述各种循环语句时，我们还会详细地介绍花括号的使用。

我们暂且不去理睬关键字 static void，而仅把它们当作编译 Java 应用程序必要的部分就行了。在学习完第 4 章后，这些内容的作用就会揭晓。现在需要记住：每个 Java 应用程序都必须有一个 main 方法，其格式如下所示：

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

C++ 注释：作为一名 C++ 程序员，一定知道类的概念。Java 的类与 C++ 的类很相似，但还是有些差异会使人感到困惑。例如，Java 中的所有函数都属于某个类的方法（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类。读者有可能对 C++ 中的静态成员函数（static member functions）十分熟悉。这些成员函数定义在类的内部，并且不对对象进行操作。Java 中的 main 方法必须是静态的。最后，与 C/C++ 一样，关键字 void 表示这个方法没有返回值，所不同的是 main 方法没有给操作系统返回“退出代码”。如果 main 方法正常退出，那么 Java 应用程序的退出代码为 0，表示成功地运行了程序。如果希望在终止程序时返回其他的代码，那就需要调用 System.exit 方法。

接下来，研究一下这段代码：

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

一对花括号表示方法体的开始与结束，在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 语句看成是这种语言的句子。在 Java 中，每个句子必须用分号结束。特别需要说明，回车不是语句的结束标志，因此，如果需要可以将一条语句写在多行上。

在上面这个 main 方法体中只包含了一条语句，其功能是：将一个文本行输出到控制台上。

在这里，使用了 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)

这等价于函数调用。

在这个示例中，调用了 println 方法并传递给它一个字符串参数。这个方法将传递给它的字符串参数显示在控制台上。然后，终止这个输出行，以便每次调用 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，都采用双引号分隔字符串。本章稍后将会详细地讲解有关字符串的知识。

与其他程序设计语言一样，在 Java 的方法中，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做实参）。对于一个方法，即使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。使用下面的语句：

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

注释：System.out 还有一个 print 方法，它在输出之后不换行。例如，System.out.print（“Hello”）打印“Hello”之后不换行，后面的输出紧跟在字符“o”之后。

3.2 注释

与大多数程序设计语言一样，Java 中的注释也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，而不必担心可执行代码会膨胀。在 Java 中，有三种书写注释的方式。最常用的方式是使用 //，其注释内容从 // 开始到本行结尾。

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

当需要长篇的注释时，既可以在每行的注释前面标记 //，也可以使用 /* 和 */ 将一段比较长的注释括起来。

第三种注释可以用来自动地生成文档。这种注释以 /** 开始，以 */ 结束。请参见程序清单 3-1。有关这种注释的详细内容和自动生成文档的具体方法请参见第 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，如果代码本身包含了一个 */，就不能用 /* 和 */ 将注释括起来。

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种类型。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 种用于表示 Unicode 编码的字符单元的字符类型 char（请参见论述 char 类型的章节）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，通常称为“大数值”（big number）。虽然被称为大数值，但它并不是一种新的 Java 类型，而是一个 Java 对象。本章稍后将会详细地介绍它的用法。

3.3.1 整型

整型用于表示没有小数部分的数值，它允许是负数。Java 提供了 4 种整型，具体内容如表 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（正好超过 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。但如果表示星球上的居住人数，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需要控制占用存储空间量的大数组。

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，或者在同一个平台中的不同操作系统之间进行移植给程序员带来的诸多问题。与此相反，C 和 C++ 程序需要针对不同的处理器选择最为有效的整型，这样就有可能造成一个在 32 位处理器上运行很好的 C 程序在 16 位系统上运行却发生整数溢出。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，所以每一种数据类型的取值范围必须固定。

长整型数值有一个后缀 L（如 4000000000L）。十六进制数值有一个前缀 0x（如 0xCAFE）。八进制有一个前缀 0，例如，010 对应八进制中的 8。很显然，八进制表示法比较容易混淆，所以建议最好不要使用八进制常数。

从 Java 7 开始，加上前缀 0b 就可以写二进制数。例如，0b1001 就是 9。另外，同样是从 Java 7 开始，还可以为数字字面量加下划线，如用 1_000_000（或 0b1111_0100_0010_0100_0000）表示一百万。这些下划线只是为了让人更易读。Java 编译器会去除这些下划线。

C++ 注释：在 C 和 C++ 中，int 表示的整型与目标平台相关。在 8086 这样的 16 位处理器上整型数值占 2 字节；不过，在 32 位处理器（比如 Pentium 或 SPARC）上，整型数值则为 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，这对编写跨平台程序带来了很大难度。在 Java 中，所有的数值类型所占据的字节数量与平台无关。

注意，Java 没有任何无符号类型（unsigned）。

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，具体内容如表 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 ±3.402 823 47E + 38F（有效位数为 6 ~ 7 位）
double	8 字节	大约 ±1.797 693 134 862 315 70E + 308（有效位数为 15 位）

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值）。绝大部分应用程序都采用 double 类型。在很多情况下，float 类型的精度很难满足需求。例如，用 7 位有效数字足以精确地表示普通雇员的年薪，但表示公司总裁的年薪可能就不够用了。实际上，只有很少的情况适合使用 float 类型，例如，需要快速地处理单精度数据，或者需要存储大量数据。

float 类型的数值有一个后缀 F（例如，3.14F）。没有后缀 F 的浮点数值（如 3.14）默认为 double 类型。当然，也可以在浮点数值后面添加后缀 D（例如，3.14D）。

注释：在 JDK 5.0 中，可以使用十六进制表示浮点数值。例如，0.125 可以表示成 0x1.0p-3。在十六进制表示法中，使用 p 表示指数，而不是 e。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。

所有的浮点数值计算都遵循 IEEE 754 规范。下面是用于表示溢出和出错情况的三个特殊的浮点数值：

- 正无穷大
- 负无穷大
- NaN（不是一个数字）

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（与相应的 Float 类型的常量一样）分别表示这三个特殊的值，但在实际应用中很少遇到。特别要说明的是，不能这样检测一个特定值是否等于 Double.NaN：

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

所有“非数值”的值都认为是不相同的。然而，可以使用 Double.isNaN 方法：

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

警告：浮点数值不适用于禁止出现舍入误差的金融计算中。例如，命令 System.out.println（2.0–1.1）将打印出 0.8999999999999999，而不是人们想象的 0.9。其主要原因是浮点数值采用二进制系统表示，而在二进制系统中无法精确的表示分数 1/10。这就好像十进制无法精确地表示 1/3 一样。如果需要在数值计算中不含有任何舍入误差，就应该使用 BigDecimal 类，本章稍后将介绍这个类。

3.3.3 char 类型

char 类型用于表示单个字符。通常用来表示字符常量。例如：'A' 是编码为 65 所对应的字符常量。与 "A" 不同，"A" 是一个包含字符 A 的字符串。Unicode 编码单元可以表示为十六进制值，其范围从 \u0000 到 \Uffff。例如：\u2122 表示注册符号（™），\u03C0 表示希腊字母 π。

除了可以采用转义序列符 \u 表示 Unicode 代码单元的编码之外，还有一些用于表示特殊字符的转义序列符，请参看表 3-3。所有这些转义序列符都可以出现在字符常量或字符串的引号内。例如，'\u2122' 或 "Hello\n"。转义序列符 \u 还可以出现在字符常量或字符串的引号之外（而其他所有转义序列不可以）。例如：

```
public static void main(String\u005B\u005D args)
```

这种形式完全符合语法规则，\u005B 和 \u005D 是 [和] 的编码。

表 3-3 特殊字符的转义序列符

转义序列	名　称	Unicode 值	转义序列	名　称	Unicode 值
\b	退格	\u0008	\”	双引号	\u0022
\t	制表	\u0009	\’	单引号	\u0027
\n	换行	\u000a	\\	反斜杠	\u005c
\r	回车	\u000d			

要想弄清 char 类型，就必须了解 Unicode 编码表。Unicode 打破了传统字符编码方法的限制。在 Unicode 出现之前，已经有许多种不同的标准：美国的 ASCII、西欧语言中的 ISO 8859-1、俄国的 KOI-8、中国的 GB 18030 和 BIG-5 等。这样就产生了下面两个问题：一个是对于任意给定的代码值，在不同的编码方案下有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，而另一些字符则需

要两个或更多个字节。

设计 Unicode 编码的目的就是要解决这些问题。在 20 世纪 80 年代开始启动设计工作时，人们认为两个字节的代码宽度足以能够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的扩展。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。在设计 Java 时决定采用 16 位的 Unicode 字符集，这样会比使用 8 位字符集的程序设计语言有很大的改进。

十分遗憾，经过一段时间，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大量的汉语、日语和韩国语言中的表意文字。现在，16 位的 char 类型已经不能满足描述所有 Unicode 字符的需要了。

下面利用一些专用术语解释一下 Java 语言解决这个问题的基本方法。从 JDK 5.0 开始。*代码点*（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，代码点采用十六进制书写，并加上前缀 U+，例如 U+0041 就是字母 A 的代码点。Unicode 的代码点可以分成 17 个*代码级别*（code plane）。第一个代码级别称为*基本的多语言级别*（basic multilingual plane），代码点从 U+0000 到 U+FFFF，其中包括了经典的 Unicode 代码；其余的 16 个附加级别，代码点从 U+10000 到 U+10FFFF，其中包括了一些*辅助字符*（supplementary character）。

UTF-16 编码采用不同长度的编码表示所有 Unicode 代码点。在基本的多语言级别中，每个字符用 16 位表示，通常被称为*代码单元*（code unit）；而辅助字符采用一对连续的代码单元进行编码。这样构成的编码值一定落入基本的多语言级别中空闲的 2048 字节内，通常被称为*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，我们可以从中迅速地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，对于整数集合的数学符号$\mathbb{Z}$，它的代码点是 U+1D56B，并且是用两个代码单元 U+D835 和 U+DD6B 编码的（有关编码算法的描述请参看 http://en.wikipe-dia.org/wiki/UTF-16 ）。

在 Java 中，char 类型用 UTF-16 编码描述一个代码单元。

我们强烈建议不要在程序中使用 char 类型，除非确实需要对 UTF-16 代码单元进行操作。最好将需要处理的字符串用抽象数据类型表示（有关这方面的内容将在 3.6 节讨论）。

3.3.4 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，数值或指针可以代替 boolean 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 应用程序员不会遇到下述麻烦：

```
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

在 Java 中，每一个变量属于一种类型（type）。在声明变量时，变量所属的类型位于变量名之前。这里列举一些声明变量的示例：

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

可以看到，每个声明以分号结束。由于声明是一条完整的语句，所以必须以分号结束。

变量名必须是一个以字母开头的由字母或数字构成的序列。需要注意，与大多数程序设计语言相比，Java 中“字母”和“数字”的范围要大。字母包括 'A' ~ 'Z'、'a' ~ 'z'、'_'、'$' 或在某种语言中代表字母的任何 Unicode 字符。例如，德国的用户可以在变量名中使用字母‘ä’；希腊人可以用 π。同样，数字包括 '0' ~ '9' 和在某种语言中代表数字的任何 Unicode 字符。但 '+' 和 '©' 这样的符号不能出现在变量名中，空格也不行。变量名中所有的字符都是有意义的，并且大小写敏感。变量名的长度没有限制。

提示：如果想要知道哪些 Unicode 字符属于 Java 中的“字母”，可以使用 Character 类的 isJavaIdentifierStart 和 isJavaIdentifierPart 方法进行检测。

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

另外，不能将变量名命名为 Java 保留字（请参看附录中的保留字列表）。

可以在一行中声明多个变量：

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

不过，不提倡使用这种风格。逐一声明每一个变量可以提高程序的可读性。

注释：如前所述，变量名对大小写敏感，例如，hireday 和 hireDay 是两个不同的变量名。在对两个不同的变量进行命名时，最好不要只存在大小写上的差异。不过，在有些时候，确实很难给变量取一个好的名字。于是，许多程序员将变量名命名为类型名，例如：

```
Box box; // "Box" is the type and "box" is the variable name
```

还有一些程序员更加喜欢在变量名前加上前缀“a”：

```
Box aBox;
```

3.4.1 变量初始化

声明一个变量之后，必须用赋值语句对变量进行显式初始化，千万不要使用未被初始化的变量。例如，Java 编译器认为下面语句序列是错误的：

```
int vacationDays;
System.out.println(vacationDays); // ERROR--variable not initialized
```

要想对一个已经声明过的变量进行赋值，就需要将变量名放在等号（=）左侧，相应取值的 Java 表达式放在等号的右侧。

```
int vacationDays;
vacationDays = 12;
```

也可以将变量的声明和初始化放在同一行中。例如：

```
int vacationDays = 12;
```

最后，在 Java 中可以将声明放在代码中的任何地方。例如，下列代码的书写形式在 Java 中是完全合法的：

```
double salary = 65000.0;
System.out.println(salary);
int vacationDays = 12; // OK to declare a variable here
```

在 Java 中，变量的声明尽可能地靠近变量第一次使用的地方，这是一种良好的程序编写风格。

C++ 注释： C 和 C++ 区分变量的声明与定义。例如：

```
int i = 10;
```

是一个定义，而

```
extern int i;
```

是一个声明。在 Java 中，不区分变量的声明与定义。

3.4.2 常量

在 Java 中，利用关键字 final 指示常量。例如：

```
public class Constants
{
   public static void main(String[] args)
   {
      final double CM_PER_INCH = 2.54;
      double paperWidth = 8.5;
      double paperHeight = 11;
      System.out.println("Paper size in centimeters: "
         + paperWidth * CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);
   }
}
```

关键字 final 表示这个变量只能被赋值一次。一旦被赋值之后，就不能够再更改了。习惯上，常量名使用全大写。

在 Java 中，经常希望某个常量可以在一个类中的多个方法中使用，通常将这些常量称为类常量。可以使用关键字 static final 设置一个类常量。下面是使用类常量的示例：

```
public class Constants2
{
    public static final double CM_PER_INCH = 2.54;

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

```
    {
       double paperWidth = 8.5;
       double paperHeight = 11;
       System.out.println("Paper size in centimeters: "
          + paperWidth * CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);
    }
}
```

需要注意，类常量的定义位于 main 方法的外部。因此，在同一个类的其他方法中也可以使用这个常量。而且，如果一个常量被声明为 public，那么其他类的方法也可以使用这个常量。在这个示例中，Constants2.CM_PER-INCH 就是这样一个常量。

C++ 注释：const 是 Java 保留的关键字，但目前并没有使用。在 Java 中，必须使用 final 定义常量。

3.5 运算符

在 Java 中，使用算术运算符＋、－、*、/ 表示加、减、乘、除运算。当参与 / 运算的两个操作数都是整数时，表示整数除法；否则，表示浮点除法。整数的求余操作（有时称为取模）用 % 表示。例如，15/2 等于 7，15%2 等于 1，15.0/2 等于 7.5。

需要注意，整数被 0 除将会产生一个异常，而浮点数被 0 除将会得到无穷大或 NaN 结果。

可以在赋值语句中采用一种简化的格式书写二元算术运算符。

例如，

```
x += 4;
```

等价于

```
x = x + 4;
```

（通常，将运算符放在赋值号的左侧，如 *= 或 %=。）

注释：可移植性是 Java 语言的设计目标之一。无论在哪个虚拟机上运行，同一运算应该得到同样的结果。对于浮点数的算术运算，实现这样的可移植性是相当困难的。double 类型使用 64 位存储一个 double 数值，而有些处理器使用 80 位浮点寄存器。这些寄存器增加了中间过程的计算精度。例如，下列运算：

```
double w = x * y / z;
```

很多 Intel 处理器计算 x * y，并且将结果存储在 80 位的寄存器中，再除以 z 并将结果截断为 64 位。这样可以得到一个更加精确的计算结果，并且还能够避免产生指数溢出。但是，这个结果可能与始终在 64 位机器上计算的结果不一样。因此，Java 虚拟机的最初规范规定所有的中间计算都必须进行截断。这种行为遭到了数值计算团体的反对。截断计算不仅可能导致溢出，而且由于截断操作需要消耗时间，所以在计算速度上要比精确计算慢。为此，Java 程序设计语言承认了最优性能与理想结果之间存在的冲突，并给予了改进。在默认情况下，虚拟机设计者允许将中间计算结果采用扩展的精度。但是，

对于使用 strictfp 关键字标记的方法必须使用严格的浮点计算来产生理想的结果。例如，可以把 main 方法标记为

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

于是，在 main 方法中的所有指令都将使用严格的浮点计算。如果将一个类标记为 strictfp，这个类中的所有方法都要使用严格的浮点计算。

实际的计算方式将取决于 Intel 处理器。在默认情况下，中间结果允许使用扩展的指数，但不允许使用扩展的尾数（Intel 芯片在截断尾数时并不损失性能）。因此，这两种方式的区别仅仅在于采用默认的方式不会产生溢出，而采用严格的计算有可能产生溢出。

如果没有仔细阅读这个注释，也没有什么关系。对大多数程序来说，浮点溢出不属于大问题。在本书中，将不使用 strictfp 关键字。

3.5.1 自增运算符与自减运算符

当然，程序员都知道加 1、减 1 是数值变量最常见的操作。在 Java 中，借鉴了 C 和 C++ 的实现方式，也使用了自增、自减运算符：n++ 将变量 n 的当前值加 1；n-- 将 n 的值减 1。例如：

```
int n = 12;
n++;
```

n 的值将变为 13。因为这些运算符改变了变量的值，所以它的操作数不能是数值。例如，4++ 就是一条非法的语句。

实际上，这两个运算符有两种形式。上面介绍的是运算符放在操作数后面的“后缀”形式，还有一种“前缀”形式，++n。两种方式都是对变量值加 1。但在表达式中，这两种形式就有区别了。前缀方式先进行加 1 运算；后缀方式则使用变量原来的值。

```
int m = 7;
int n = 7;
int a = 2 * ++m; // now a is 16, m is 8
int b = 2 * n++; // now b is 14, n is 8
```

我们建议不要在其他表达式的内部使用 ++，这样编写的代码很容易令人困惑，并会产生烦人的 bug。

（当然，++ 运算符作为 C++ 语言名称的一部分，也引发了有关程序设计语言的第一个笑话。C++ 的反对者认为这种语言的名称也存在着 bug，他们说：“因为只有对它改进之后，我们才有可能使用它，所以它的名字应该命名为 ++C。”）

3.5.2 关系运算符与 boolean 运算符

Java 包含各种关系运算符。其中，使用两个等号 = = 检测是否相等。例如，3 = = 7 的值为 false。

使用 != 检测是否不相等。例如，3 != 7 的值为 true。

另外，经常使用的运算符还有 <（小于）、>（大于）、<=（小于等于）和 >=（大于等于）。

Java 沿用了 C++ 的习惯，用 && 表示逻辑“与”、用 || 表示逻辑“或”。从 != 运算符很容易看出，! 表示逻辑“非”。&& 和 || 是按照“短路”方式求值的。如果第一个操作数已经能够确定表达式的值，第二个操作数就不必计算了。如果用 && 对两个表达式进行计算：

expression$_1$ && *expression*$_2$

并且第一个表达式值为 false，结果不可能为真。因此，第二个表达式的值就没有必要计算了。这种方式可以避免一些错误的发生。例如，表达式：

```
x != 0 && 1 / x > x + y // no division by 0
```

当 x 为 0 时，不会计算第二部分。因此，若 x 为 0，1/x 不被计算，也不会出现除以 0 的错误。

与之类似，对于 expression1 || expression2，当第一个表达式为 true 时，结果自动为 true，不必再计算第二部分。

最后，Java 支持三元操作符 ?:。在很多时候，这个操作符非常有用。表达式

condition ? *expression*$_1$: *expression*$_2$

当条件 condition 为真时计算第 1 个表达式，否则计算第 2 个表达式。例如：

```
x < y ? x : y
```

返回 x 和 y 中较小的那个值。

3.5.3 位运算符

在处理整型数值时，可以直接对组成整型数值的各个位进行操作。这意味着可以使用屏蔽技术获得整数中的各个位。位运算符包括：

& ("与")、| ("或")、^ ("异或")、~ ("非")

这些运算符在位模式下工作。例如，如果 n 是一个整型变量，并且用二进制表示的 n 从右数第 4 位为 1，那么

```
int fourthBitFromRight = (n & 0b1000) / 0b1000;
```

返回 1；否则返回 0。通过运用 2 的幂次方的 & 运算可以将其他位屏蔽掉，而只保留其中的某一位。

注释： & 和 | 运算符应用于布尔值，得到的结果也是布尔值。这两个运算符与 && 和 || 的运算非常类似，只是不按“短路”方式计算。即在得到计算结果之前，一定要计算两个操作数的值。

另外，“>>”和“<<”运算符将二进制位进行右移或左移操作。当需要建立位模式屏蔽某些位时，使用这两个运算符十分方便：

```
int fourthBitFromRight = (n & (1 << 3)) >> 3;
```

最后，>>> 运算符将用 0 填充高位；>> 运算符用符号位填充高位。没有 <<< 运算符。

警告： 对移位运算符右侧的参数需要进行模 32 的运算（除非左边的操作数是 long 类型，在这种情况下需对右侧操作数模 64）。例如，1 << 35 与 1 << 3 或 8 是相同的。

C++ 注释：在 C 或 C++ 中无法确定 >> 操作执行的是算术移位（扩展符号位），还是逻辑移位（高位填 0）。在执行中将会选择效率较高的一种。这就是说，在 C/C++ 中，>> 运算符实际上只是为非负数定义的。Java 消除了这种含糊性。

3.5.4 数学函数与常量

在 Math 类中，包含了各种各样的数学函数。在编写不同类别的程序时，可能需要的函数也不同。

要想计算一个数值的平方根，可以使用 sqrt 方法：

```
double x = 4;
double y = Math.sqrt(x);
System.out.println(y); // prints 2.0
```

注释：println 方法和 sqrt 方法存在微小的差异。println 方法操作一个定义在 System 类中的 System.out 对象。但是，Math 类中的 sqrt 方法处理的不是对象，这样的方法被称为静态方法。有关静态方法的详细内容请参看第 4 章。

在 Java 中，没有幂运算，因此需要借助于 Math 类的 pow 方法。语句：

```
double y = Math.pow(x, a);
```

将 y 的值设置为 x 的 a 次幂（x^a）。pow 方法有两个 double 类型的参数，其返回结果也为 double 类型。

Math 类提供了一些常用的三角函数：

```
Math.sin
Math.cos
Math.tan
Math.atan
Math.atan2
```

还有指数函数以及它的反函数——自然对数以及以 10 为底的对数：

```
Math.exp
Math.log
Math.log10
```

最后，Java 还提供了两个用于表示 π 和 e 常量的近似值：

```
Math.PI
Math.E
```

提示：不必在数学方法名和常量名前添加前缀“Math.”，只要在源文件的顶部加上下面这行代码就可以了。

```
import static java.lang.Math.*;
```

例如：

```
System.out.println("The square root of \u03C0 is " + sqrt(PI));
```

在第 4 章中将讨论静态导入。

注释：在 Math 类中，为了达到最快的性能，所有的方法都使用计算机浮点单元中的例程。如果得到一个完全可预测的结果比运行速度更重要的话，那么就应该使用 StrictMath 类。它使用“自由发布的 Math 库”（fdlibm）实现算法，以确保在所有平台上得到相同的结果。有关这些算法的源代码请参看 www.netlib.org/fdlibm（当 fdlibm 为一个函数提供了多个定义时，StrictMath 类就会遵循 IEEE 754 版本，它的名字将以“e”开头）。

3.5.5 数值类型之间的转换

在程序运行时，经常需要将一种数值类型转换为另一种数值类型。图 3-1 给出了数值类型之间的合法转换。

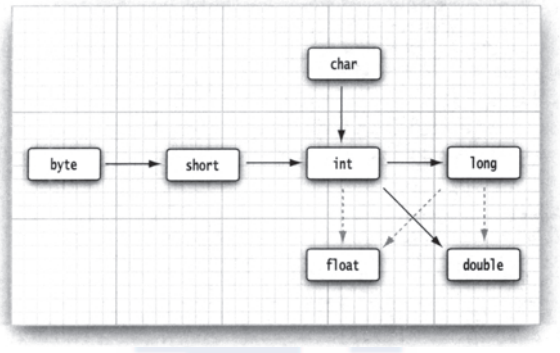

图 3-1 数值类型之间的合法转换

在图 3-1 中有 6 个实心箭头，表示无信息丢失的转换；有 3 个虚箭头，表示可能有精度损失的转换。例如，123 456 789 是一个大整数，它所包含的位数比 float 类型所能够表达的位数多。当将这个整型数值转换为 float 类型时，将会得到同样大小的结果，但却失去了一定的精度。

```
int n = 123456789;
float f = n; // f is 1.23456792E8
```

当使用上面两个数值进行二元操作时（例如 n + f，n 是整数，f 是浮点数），先要将两个操作数转换为同一种类型，然后再进行计算。

- 如果两个操作数中有一个是 double 类型，另一个操作数就会转换为 double 类型。
- 否则，如果其中一个操作数是 float 类型，另一个操作数将会转换为 float 类型。
- 否则，如果其中一个操作数是 long 类型，另一个操作数将会转换为 long 类型。
- 否则，两个操作数都将被转换为 int 类型。

3.5.6 强制类型转换

在上一小节中看到，在必要的时候，int 类型的值将会自动地转换为 double 类型。但另一方面，有时也需要将 double 转换成 int。在 Java 中，允许进行这种数值之间的类型转换。

当然，有可能会丢失一些信息。在这种情况下，需要通过强制类型转换（cast）实现这个操作。强制类型转换的语法格式是在圆括号中给出想要转换的目标类型，后面紧跟待转换的变量名。例如：

```
double x = 9.997;
int nx = (int) x;
```

这样，变量 nx 的值为 9。强制类型转换通过截断小数部分将浮点值转换为整型。

如果想对浮点数进行舍入运算，以便得到最接近的整数（在很多情况下，希望使用这种操作方式），那就需要使用 Math.round 方法：

```
double x = 9.997;
int nx = (int) Math.round(x);
```

现在，变量 nx 的值为 10。当调用 round 的时候，仍然需要使用强制类型转换（int）。其原因是 round 方法返回的结果为 long 类型，由于存在信息丢失的可能性，所以只有使用显式的强制类型转换才能够将 long 类型转换成 int 类型。

警告：如果试图将一个数值从一种类型强制转换为另一种类型，而又超出了目标类型的表示范围，结果就会截断成一个完全不同的值。例如，(byte) 300 的实际值为 44。

C++ 注释：不要在 boolean 类型与任何数值类型之间进行强制类型转换，这样可以防止发生错误。只有极少数的情况才需要将布尔类型转换为数值类型，这时可以使用条件表达式 b？ 1:0。

3.5.7 括号与运算符级别

表 3-4 给出了运算符的优先级。如果不使用圆括号，就按照给出的运算符优先级次序进行计算。同一个级别的运算符按照从左到右的次序进行计算（除了表中给出的右结合运算符外。）例如，由于 && 的优先级比 || 的优先级高，所以表达式

```
a && b || c
```

等价于

```
(a && b) || c
```

又因为 += 是右结合运算符，所以表达式

```
a += b += c
```

等价于

```
a += (b += c)
```

也就是将 b += c 的结果（加上 c 之后的 b）加到 a 上。

C++ 注释：与 C 或 C++ 不同，Java 不使用逗号运算符。不过，可以在 for 语句中使用逗号分隔表达式列表。

表 3-4 运算符优先级

运 算 符	结 合 性
[ ] . ( ) (方法调用)	从左向右
! ~ ++ -- + (一元运算) - (一元运算) ( ) (强制类型转换) new	从右向左
*/ %	从左向右
+ –	从左向右
<< >> >>>	从左向右
< <= > >= instanceof	从左向右
= = !=	从左向右
&	从左向右
^	从左向右
|	从左向右
&&	从左向右
||	从左向右
?:	从右向左
= += –= *= /= %= &= |= ^= <<= >>= >>>=	从右向左

3.5.8 枚举类型

有时候，变量的取值只在一个有限的集合内。例如：销售的服装或比萨饼只有小、中、大和超大这四种尺寸。当然，可以将这些尺寸分别编码为 1、2、3、4 或 S、M、L、X。但这样存在着一定的隐患。在变量中很可能保存的是一个错误的值（如 0 或 m）。

针对这种情况，可以自定义枚举类型。枚举类型包括有限个命名的值。例如，

```
enum Size { SMALL, MEDIUM, LARGE, EXTRA_LARGE };
```

现在，可以声明这种类型的变量：

```
Size s = Size.MEDIUM;
```

Size 类型的变量只能存储这个类型声明中给定的某个枚举值，或者 null 值，null 表示这个变量没有设置任何值。

有关枚举类型的详细内容将在第 5 章介绍。

3.6 字符串

从概念上讲，Java 字符串就是 Unicode 字符序列。例如，串“ Java\u2122”由 5 个 Unicode 字符 J、a、v、a 和 ™。Java 没有内置的字符串类型，而是在标准 Java 类库中提供了一个预定义类，很自然地叫做 String。每个用双引号括起来的字符串都是 String 类的一个实例：

```
String e = ""; // an empty string
String greeting = "Hello";
```

3.6.1 子串

String类的substring方法可以从一个较大的字符串提取出一个子串。例如：

```
String greeting = "Hello";
String s = greeting.substring(0, 3);
```

创建了一个由字符“Hel”组成的字符串。

substring方法的第二个参数是不想复制的第一个位置。这里要复制位置为0、1和2（从0到2，包括0和2）的字符。在substring中从0开始计数，直到3为止，但不包含3。

substring的工作方式有一个优点：容易计算子串的长度。字符串s.substring(a, b)的长度为b–a。例如，子串“Hel”的长度为3 — 0 = 3。

3.6.2 拼接

与绝大多数的程序设计语言一样，Java语言允许使用+号连接（拼接）两个字符串。

```
String expletive = "Expletive";
String PG13 = "deleted";
String message = expletive + PG13;
```

上述代码将“Expletivedeleted”赋给变量message（注意，单词之间没有空格，+号按照给定的次序将两个字符串拼接起来）。

当将一个字符串与一个非字符串的值进行拼接时，后者被转换成字符串（在第5章中可以看到，任何一个Java对象都可以转换成字符串）。例如：

```
int age = 13;
String rating = "PG" + age;
```

rating设置为“PG13”。

这种特性通常用在输出语句中。例如：

```
System.out.println("The answer is " + answer);
```

这是一条合法的语句，并且将会打印出所希望的结果（因为单词is后面加了一个空格，输出时也会加上这个空格）。

3.6.3 不可变字符串

String类没有提供用于修改字符串的方法。如果希望将greeting的内容修改为“Help!”，不能直接地将greeting的最后两个位置的字符修改为‘p’和‘!’。这对于C程序员来说，将会感到无从下手。如何修改这个字符串呢？在Java中实现这项操作非常容易。首先提取需要的字符，然后再拼接上替换的字符串：

```
greeting = greeting.substring(0, 3) + "p!";
```

上面这条语句将greeting当前值修改为“Help！”。

由于不能修改Java字符串中的字符，所以在Java文档中将String类对象称为不可变字符串，如同数字3永远是数字3一样，字符串“Hello”永远包含字符H、e、l、l和o的代

码单元序列，而不能修改其中的任何一个字符。当然，可以修改字符串变量 greeting，让它引用另外一个字符串，这就如同可以将存放 3 的数值变量改成存放 4 一样。

这样做是否会降低运行效率呢？看起来好像修改一个代码单元要比创建一个新字符串更加简洁。答案是：也对，也不对。的确，通过拼接“ Hel”和“ p!”来创建一个新字符串的效率确实不高。但是，不可变字符串却有一个优点：编译器可以让字符串共享。

为了弄清具体的工作方式，可以想象将各种字符串存放在公共的存储池中。字符串变量指向存储池中相应的位置。如果复制一个字符串变量，原始字符串与复制的字符串共享相同的字符。

总而言之，Java 的设计者认为共享带来的高效率远远胜过于提取、拼接字符串所带来的低效率。查看一下程序会发现：很少需要修改字符串，而是往往需要对字符串进行比较（有一种例外情况，将源自于文件或键盘的单个字符或较短的字符串汇集成字符串。为此，Java 提供了一个独立的类，在 3.6.8 节中将详细介绍）。

C++ **C++ 注释：**在 C 程序员第一次接触 Java 字符串的时候，常常会感到迷惑，因为他们总将字符串认为是字符型数组：

```
char greeting[] = "Hello";
```

这种认识是错误的，Java 字符串更加像 char* 指针，

```
char* greeting = "Hello";
```

当采用另一个字符串替换 greeting 的时候，Java 代码主要进行下列操作：

```
char* temp = malloc(6);
strncpy(temp, greeting, 3);
strncpy(temp + 3, "p!", 3);
greeting = temp;
```

的确，现在 greeting 指向字符串“ Help!”。即使一名最顽固的 C 程序员也得承认 Java 语法要比一连串的 strncpy 调用舒适得多。然而，如果将 greeting 赋予另外一个值又会怎样呢？

```
greeting = "Howdy";
```

这样做会不会产生内存遗漏呢？毕竟，原始字符串放置在堆中。十分幸运，Java 将自动地进行垃圾回收。如果一块内存不再使用了，系统最终会将其回收。

对于一名使用 ANSI C++ 定义的 string 类的 C++ 程序员，会感觉使用 Java 的 String 类型更为舒适。C++ string 对象也自动地进行内存的分配与回收。内存管理是通过构造器、赋值操作和析构器显式执行的。然而，C++ 字符串是可修改的，也就是说，可以修改字符串中的单个字符。

3.6.4 检测字符串是否相等

可以使用 equals 方法检测两个字符串是否相等。对于表达式：

```
s.equals(t)
```

如果字符串 s 与字符串 t 相等，则返回 true ；否则，返回 false。需要注意，s 与 t 可以是字符

串变量，也可以是字符串常量。例如，下列表达式是合法的：

```
"Hello".equals(greeting)
```

要想检测两个字符串是否相等，而不区分大小写，可以使用 equalsIgnoreCase 方法。

```
"Hello".equalsIgnoreCase("hello")
```

一定不能使用 == 运算符检测两个字符串是否相等！这个运算符只能够确定两个字符串是否放置在同一个位置上。当然，如果字符串放置在同一个位置上，它们必然相等。但是，完全有可能将内容相同的多个字符串的拷贝放置在不同的位置上。

```
String greeting = "Hello"; //initialize greeting to a string
if (greeting == "Hello") . . .
   // probably true
if (greeting.substring(0, 3) == "Hel") . . .
   // probably false
```

如果虚拟机始终将相同的字符串共享，就可以使用 == 运算符检测是否相等。但实际上只有字符串常量是共享的，而 + 或 substring 等操作产生的结果并不是共享的。因此，千万不要使用 == 运算符测试字符串的相等性，以免在程序中出现糟糕的 bug。从表面上看，这种 bug 很像随机产生的间歇性错误。

C++ 注释：对于习惯使用 C++ 的 string 类的人来说，在进行相等性检测的时候一定要特别小心。C++ 的 string 类重载了 == 运算符以便检测字符串内容的相等性。可惜 Java 没有采用这种方式，它的字符串“看起来、感觉起来”与数值一样，但进行相等性测试时，其操作方式又类似于指针。语言的设计者本应该像对 + 那样也进行特殊处理，即重定义 == 运算符。当然，每一种语言都会存在一些不太一致的地方。

C 程序员从不使用 == 对字符串进行比较，而使用 strcmp 函数。Java 的 compareTo 方法与 strcmp 完全类似，因此，可以这样使用：

```
if (greeting.compareTo("Hello") == 0) . . .
```

不过，使用 equals 看起来更为清晰。

3.6.5 空串与 Null 串

空串 "" 是长度为 0 的字符串。可以调用以下代码检查一个字符串是否为空：

```
if (str.length() == 0)
```

或

```
if (str.equals(""))
```

空串是一个 Java 对象，有自己的串长度（0）和内容（空）。不过，String 变量还可以存放一个特殊的值，名为 null，这表示目前没有任何对象与该变量关联（关于 null 的更多信息请参见第 4 章）。要检查一个字符串是否为 null，要使用以下条件：

```
if (str == null)
```

有时要检查一个字符串既不是 null 也不为空串，这种情况下就需要使用以下条件：

```
if (str != null && str.length() != 0)
```

首先要检查 str 不为 null。在第 4 章会看到，如果在一个 null 值上调用方法，会出现错误。

3.6.6 代码点与代码单元

Java 字符串由 char 序列组成。从 3.3.3 节“ char 类型”已经看到，char 数据类型是一个采用 UTF-16 编码表示 Unicode 代码点的代码单元。大多数的常用 Unicode 字符使用一个代码单元就可以表示，而辅助字符需要一对代码单元表示。

length 方法将返回采用 UTF-16 编码表示的给定字符串所需要的代码单元数量。例如：

```
String greeting = "Hello";
int n = greeting.length(); // is 5.
```

要想得到实际的长度，即代码点数量，可以调用：

```
int cpCount = greeting.codePointCount(0, greeting.length());
```

调用 s.charAt(n) 将返回位置 n 的代码单元，n 介于 0 ~ s.length()-1 之间。例如：

```
char first = greeting.charAt(0); // first is 'H'
char last = greeting.charAt(4); // last is 'o'
```

要想得到第 i 个代码点，应该使用下列语句

```
int index = greeting.offsetByCodePoints(0, i);
int cp = greeting.codePointAt(index);
```

注释： 类似于 C 和 C++，Java 对字符串中的代码单元和代码点从 0 开始计数。

为什么会对代码单元如此大惊小怪？请考虑下列语句：

```
𝕫 is the set of integers
```

使用 UTF-16 编码表示 𝕫 需要两个代码单元。调用

```
char ch = sentence.charAt(1)
```

返回的不是空格，而是第二个代码单元 𝕫。𝕫 为了避免这种情况的发生，请不要使用 char 类型。这太低级了。

如果想要遍历一个字符串，并且依次查看每一个代码点，可以使用下列语句：

```
int cp = sentence.codePointAt(i);
if (Character.isSupplementaryCodePoint(cp)) i += 2;
else i++;
```

可以使用下列语句实现回退操作：

```
i--;
if (Character.isSurrogate(sentence.charAt(i))) i--;
int cp = sentence.codePointAt(i);
```

3.6.7 字符串 API

Java 中的 String 类包含了 50 多个方法。令人惊讶的是绝大多数都很有用，可以设想使用的频繁非常高。下面的 API 注释汇总了一部分最常用的方法。

注释：可以发现，本书中给出的 API 注释会有助于理解 Java 应用程序编程接口（API）。每一个 API 的注释都以形如 java.lang.String 的类名开始。java.lang 包的重要性将在第 4 章给出解释。类名之后是一个或多个方法的名字、解释和参数描述。

在这里，一般不列出某个类的所有方法，而是选择一些最常用的方法，并以简洁的方式给予描述。完整的方法列表请参看联机文档（请参看 3.6.8 节）。

这里还列出了所给类的版本号。如果某个方法是在这个版本之后添加的，就会给出一个单独的版本号。

API java.lang.string 1.0

- `char charAt (int index)`
  返回给定位置的代码单元。除非对底层的代码单元感兴趣，否则不需要调用这个方法。
- `int codePointAt(int index) 5.0`
  返回从给定位置开始或结束的代码点。
- `int offsetByCodePoints(int startIndex, int cpCount) 5.0`
  返回从 startIndex 代码点开始，位移 cpCount 后的代码点索引。
- `int compareTo(String other)`
  按照字典顺序，如果字符串位于 other 之前，返回一个负数；如果字符串位于 other 之后，返回一个正数；如果两个字符串相等，返回 0。
- `boolean endsWith(String suffix)`
  如果字符串以 suffix 结尾，返回 true。
- `boolean equals(Object other)`
  如果字符串与 other 相等，返回 true。
- `boolean equalsIgnoreCase(String other)`
  如果字符串与 other 相等（忽略大小写），返回 true。
- `int indexOf(String str)`
- `int indexOf(String str, int fromIndex)`
- `int indexOf(int cp)`
- `int indexOf(int cp, int fromIndex)`
  返回与字符串 str 或代码点 cp 匹配的第一个子串的开始位置。这个位置从索引 0 或 fromIndex 开始计算。如果在原始串中不存在 str，返回 –1。
- `int lastIndexOf(String str)`
- `int lastIndexOf(String str, int fromIndex)`
- `int lastindexOf(int cp)`
- `int lastindexOf(int cp, int fromIndex)`
  返回与字符串 str 或代码点 cp 匹配的最后一个子串的开始位置。这个位置从原始串尾端或 fromIndex 开始计算。

- `int length( )`
  返回字符串的长度。
- `int codePointCount(int startIndex, int endIndex) 5.0`
  返回 startIndex 和 endIndex — 1 之间的代码点数量。没有配成对的代用字符将计入代码点。
- `String replace(CharSequence oldString,CharSequence newString)`
  返回一个新字符串。这个字符串用 newString 代替原始字符串中所有的 oldString。可以用 String 或 StringBuilder 对象作为 CharSequence 参数。
- `boolean startsWith(String prefix)`
  如果字符串以 preffix 字符串开始，返回 true。
- `String substring(int beginIndex)`
- `String substring(int beginIndex, int endIndex)`
  返回一个新字符串。这个字符串包含原始字符串中从 beginIndex 到串尾或 endIndex–1 的所有代码单元。
- `String toLowerCase( )`
  返回一个新字符串。这个字符串将原始字符串中的所有大写字母改成了小写字母。
- `String toUpperCase( )`
  返回一个新字符串。这个字符串将原始字符串中的所有小写字母改成了大写字母。
- `String trim( )`
  返回一个新字符串。这个字符串将删除了原始字符串头部和尾部的空格。

3.6.8 阅读联机 API 文档

正如前面所看到的，String 类包含许多方法。而且，在标准库中有几千个类，方法数量更加惊人。要想记住所有的类和方法是一件不太不可能的事情。因此，学会使用在线 API 文档十分重要，从中可以查阅到标准类库中的所有类和方法。API 文档是 JDK 的一部分，它是 HTML 格式的。让浏览器指向安装 JDK 的 docs/api/index.html 子目录，就可以看到如图 3-2 所示的屏幕。

可以看到，屏幕被分成三个窗框。在左上方的小窗框中显示了可使用的所有包。在它下面稍大的窗框中列出了所有的类。点击任何一个类名之后，这个类的 API 文档就会显示在右侧的大窗框中（请参看图 3-3）。例如，要获得有关 String 类方法的更多信息，可以滚动第二个窗框，直到看见 String 链接为止，然后点击这个链接。

接下来，滚动右面的窗框，直到看见按字母顺序排列的所有方法为止（请参看图 3-4）。点击任何一个方法名便可以查看这个方法的详细描述（参见图 3-5）。例如，如果点击 compareToIgnoreCase 链接，就会看到 compareToIgnoreCase 方法的描述。

提示：马上在浏览器中将 docs/api/index.html 页面做一个书签。

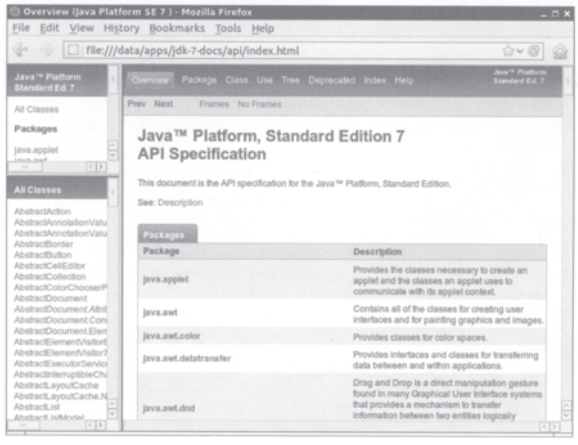

图 3-2　API 文档的三个窗格

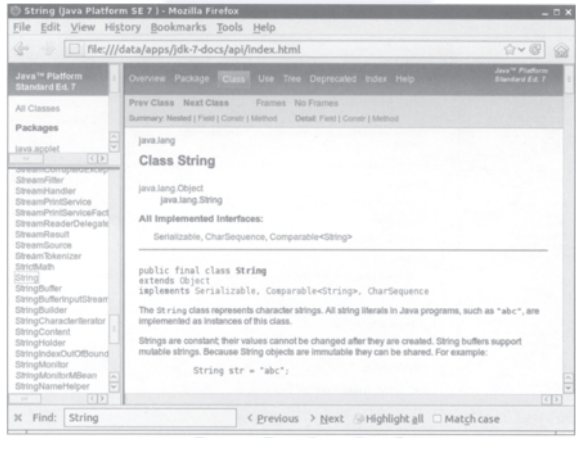

图 3-3　String 类的描述

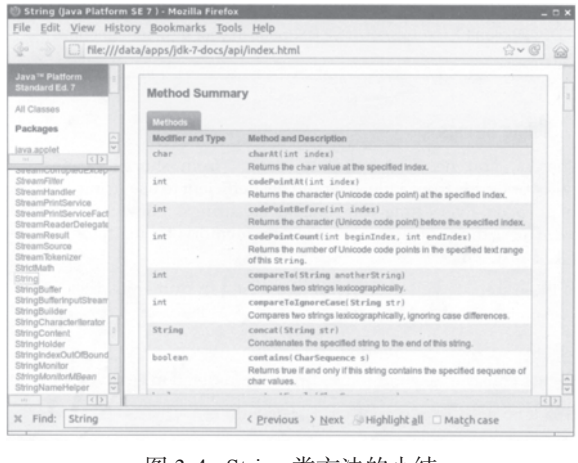

图 3-4　String 类方法的小结

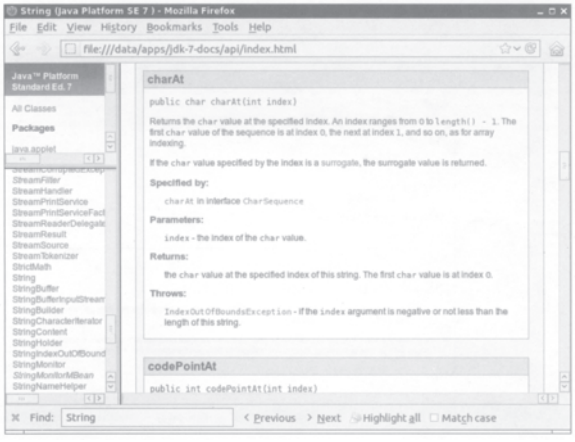

图 3-5 String 方法的详细描述

3.6.9 构建字符串

有些时候，需要由较短的字符串构建字符串，例如，按键或来自文件中的单词。采用字符串连接的方式达到此目的效率比较低。每次连接字符串，都会构建一个新的 String 对象，既耗时，又浪费空间。使用 StringBuilder 类就可以避免这个问题的发生。

如果需要用许多小段的字符串构建一个字符串，那么应该按照下列步骤进行。首先，构建一个空的字符串构建器：

```
StringBuilder builder = new StringBuilder();
```

（有关构造器和 new 操作符的内容将在第 4 章详细介绍。）

当每次需要添加一部分内容时，就调用 append 方法。

```
builder.append(ch); // appends a single character
builder.append(str); // appends a string
```

在需要构建字符串时就调用 toString 方法，将可以得到一个 String 对象，其中包含了构建器中的字符序列。

```
String completedString = builder.toString();
```

注释：在 JDK5.0 中引入 StringBuilder 类。这个类的前身是 StringBuffer，其效率稍有些低，但允许采用多线程的方式执行添加或删除字符的操作。如果所有字符串在一个单线程中编辑（通常都是这样），则应该用 StringBuilder 替代它。这两个类的 API 是相同的。

下面的 API 注释包含了 StringBuilder 类中的重要方法。

API java.lang.StringBuilder 5.0

- `StringBuilder()`

  构造一个空的字符串构建器。

- `int length()`
  返回构建器或缓冲器中的代码单元数量。
- `StringBuilder append(String str)`
  追加一个字符串并返回 this。
- `StringBuilder append(char c)`
  追加一个代码单元并返回 this。
- `StringBuilder appendCodePoint(int cp)`
  追加一个代码点，并将其转换为一个或两个代码单元并返回 this。
- `void setCharAt(int i,char c)`
  将第 i 个代码单元设置为 c。
- `StringBuilder insert(int offset,String str)`
  在 offset 位置插入一个字符串并返回 this。
- `StringBuilder insert(int offset,Char c)`
  在 offset 位置插入一个代码单元并返回 this。
- `StringBuilder delete(int startIndex,int endIndex)`
  删除偏移量从 startIndex 到— endIndex — 1 的代码单元并返回 this。
- `String toString()`
  返回一个与构建器或缓冲器内容相同的字符串。

3.7 输入输出

为了增加后面示例程序的趣味性，需要程序能够接收输入，并以适当的格式输出。当然，现代的程序都使用 GUI 收集用户的输入，然而，编写这种界面的程序需要使用较多的工具与技术，目前还不具备这些条件。主要原因是需要熟悉 Java 程序设计语言，因此只要有简单的用于输入输出的控制台就可以了。第 7 章 ~ 第 9 章将详细地介绍 GUI 程序设计。

3.7.1 读取输入

前面已经看到，打印输出到“标准输出流”（即控制台窗口）是一件非常容易的事情，只要调用 System.out.println 即可。然而，读取“标准输入流”System.in 就没有那么简单了。要想通过控制台进行输入，首先需要构造一个 Scanner 对象，并与“标准输入流”System.in 关联。

```
Scanner in = new Scanner(System.in);
```

（构造器和 new 操作符将在第 4 章中详细地介绍。）

现在，就可以使用 Scanner 类的各种方法实现输入操作了。例如，nextLine 方法将输入一行。

```
System.out.print("What is your name? ");
String name = in.nextLine();
```

在这里，使用 nextLine 方法是因为在输入行中有可能包含空格。要想读取一个单词（以空白符作为分隔符），就调用

```
String firstName = in.next();
```

要想读取一个整数，就调用 nextInt 方法。

```
System.out.print("How old are you? ");
int age = in.nextInt();
```

与此类似，要想读取下一个浮点数，就调用 nextDouble 方法。

在程序清单 3-2 的程序中，询问用户姓名和年龄，然后打印一条如下格式的消息：

```
Hello, Cay. Next year, you'll be 52
```

最后，在程序的最开始添加上一行：

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

Scanner 类定义在 java.util 包中。当使用的类不是定义在基本 java.lang 包中时，一定要使用 import 指示字将相应的包加载进来。有关包与 import 指示字的详细描述请参看第 4 章。

程序清单 3-2 InputTest/InputTest.java

```
1  import java.util.*;
2
3  /**
4   * This program demonstrates console input.
5   * @version 1.10 2004-02-10
6   * @author Cay Horstmann
7   */
8  public class InputTest
9  {
10    public static void main(String[] args)
11    {
12       Scanner in = new Scanner(System.in);
13
14       // get first input
15       System.out.print("What is your name? ");
16       String name = in.nextLine();
17
18       // get second input
19       System.out.print("How old are you? ");
20       int age = in.nextInt();
21
22       // display output on console
23       System.out.println("Hello, " + name + ". Next year, you'll be " + (age + 1));
24    }
25 }
```


注释：因为输入是可见的，所以 Scanner 类不适用于从控制台读取密码。Java SE 6 特别引入了 Console 类实现这个目的。要想读取一个密码，可以采用下列代码：

```
Console cons = System.console();
String username = cons.readLine("User name: ");
char[] passwd = cons.readPassword("Password: ");
```

为了安全起见，返回的密码存放在一维字符数组中，而不是字符串中。在对密码进行处理之后，应该马上用一个填充值覆盖数组元素（数组处理将在本章稍后介绍）。

采用 Console 对象处理输入不如采用 Scanner 方便。每次只能读取一行输入，而没有能够读取一个单词或一个数值的方法。

API java.util.Scanner 5.0

- `Scanner (InputStream in)`
  用给定的输入流创建一个 Scanner 对象。
- `String nextLine( )`
  读取输入的下一行内容。
- `String next( )`
  读取输入的下一个单词（以空格作为分隔符）。
- `int nextInt( )`
- `double nextDouble( )`
  读取并转换下一个表示整数或浮点数的字符序列。
- `boolean hasNext( )`
  检测输入中是否还有其他单词。
- `boolean hasNextInt( )`
- `boolean hasNextDouble( )`
  检测是否还有表示整数或浮点数的下一个字符序列。

API java.lang.System 1.0

- `static Console console( ) 6`
  如果有可能进行交互操作，就通过控制台窗口为交互的用户返回一个 Console 对象，否则返回 null。对于任何一个通过控制台窗口启动的程序，都可使用 Console 对象。否则，其可用性将与所使用的系统有关。

API java.io.Console 6

- `static char[] readPassword(String prompt, Object...args)`
- `static String readLine(String prompt, Object...args)`
  显示字符串 prompt 并且读取用户输入，直到输入行结束。args 参数可以用来提供输入格式。有关这部分内容将在下一节中介绍。

3.7.2 格式化输出

可以使用 System.out.print(x) 将数值 x 输出到控制台上。这条命令将以 x 对应的数据类型所允许的最大非 0 数字位数打印输出 x。例如：

```
double x = 10000.0 / 3.0;
System.out.print(x);
```

打印

```
3333.3333333333335
```

如果希望显示美元、美分等符号，则有可能会出现问题。

在早期的 Java 版本中，格式化数值曾引起过一些争议。庆幸的是，Java SE 5.0 沿用了 C 语言库函数中的 printf 方法。例如，调用

```
System.out.printf("%8.2f", x);
```

可以用 8 个字符的宽度和小数点后两个字符的精度打印 x。也就是说，打印输出一个空格和 7 个字符，如下所示：

```
3333.33
```

在 printf 中，可以使用多个参数，例如：

```
System.out.printf("Hello, %s. Next year, you'll be %d", name, age);
```

每一个以 % 字符开始的格式说明符都用相应的参数替换。格式说明符尾部的转换符将指示被格式化的数值类型：f 表示浮点数，s 表示字符串，d 表示十进制整数。表 3-5 列出了所有转换符。

表 3-5 用于 printf 的转换符

转换符	类 型	举 例	转换符	类 型	举 例
d	十进制整数	159	s	字符串	Hello
x	十六进制整数	9f	c	字符	H
o	八进制整数	237	b	布尔	True
f	定点浮点数	15.9	h	散列码	42628b2
e	指数浮点数	1.59e+01	tx	日期时间	见表 3-7
g	通用浮点数	—	%	百分号	%
a	十六进制浮点数	0x1.fccdp3	n	与平台有关的行分隔符	—

另外，还可以给出控制格式化输出的各种标志。表 3-6 列出了所有的标志。例如，逗号标志增加了分组的分隔符。即

```
System.out.printf("%,.2f", 10000.0 / 3.0);
```

打印

```
3,333.33
```

可以使用多个标志，例如，“%, (.2f”使用分组的分隔符并将负数括在括号内。

表 3-6 用于 printf 的标志

标 志	目 的	举 例
+	打印正数和负数的符号	+3333.33
空格	在正数之前添加空格	| 3333.33|

（续）

标 志	目 的	举 例
0	数字前面补 0	003333.33
-	左对齐	|3333.33 |
(	将负数括在括号内	(3333.33)
,	添加分组分隔符	3,333.33
#（对于 f 格式）	包含小数点	3,333.
#（对于 x 或 0 格式）	添加前缀 0x 或 0	0xcafe
$	给定被格式化的参数索引。例如，%1$d，%1$x 将以十进制和十六进制格式打印第 1 个参数	159 9F
<	格式化前面说明的数值。例如，%d%<x 以十进制和十六进制打印同一个数值	159 9F

注释：可以使用 s 转换符格式化任意的对象。对于任意实现了 Formattable 接口的对象都将调用 formatTo 方法；否则将调用 toString 方法，它可以将对象转换为字符串。在第 5 章中将讨论 toString 方法，在第 6 章中将讨论接口。

可以使用静态的 String.format 方法创建一个格式化的字符串，而不打印输出：

```
String message = String.format("Hello, %s. Next year, you'll be %d", name, age);
```

尽管在第 4 章之前，没有对 Date 类型进行过详细地描述，但基于完整性的考虑，还是简略地介绍一下 printf 方法中日期与时间的格式化选项。在这里，使用以 t 开始，以表 3-7 中任意字母结束的两个字母格式。例如，

```
System.out.printf("%tc", new Date());
```

这条语句将用下面的格式打印当前的日期和时间：

```
Mon Feb 09 18:05:19 PST 2004
```

表 3-7 日期和时间的转换符

转 换 符	类 型	举 例
c	完整的日期和时间	Mon Feb 09 18:05:19 PST 2004
F	ISO 8601 日期	2004-02-09
D	美国格式的日期（月 / 日 / 年）	02/09/2004
T	24 小时时间	18:05:19
r	12 小时时间	06:05:19 pm
R	24 小时时间没有秒	18:05
Y	4 位数字的年（前面补 0）	2004
y	年的后两位数字（前面补 0）	04
C	年的前两位数字（前面补 0）	20
B	月的完整拼写	February

（续）

转 换 符	类　型	举　例
b 或 h	月的缩写	Feb
m	两位数字的月（前面补 0）	02
d	两位数字的日（前面补 0）	09
e	两位数字的月（前面不补 0）	9
A	星期几的完整拼写	Monday
a	星期几的缩写	Mon
j	三位数的年中的日子（前面补 0），在 001 到 366 之间	069
H	两位数字的小时（前面补 0），在 0 到 23 之间	18
k	两位数字的小时（前面不补 0），在 0 到 23 之间	18
I	两位数字的小时（前面补 0），在 0 到 12 之间	06
l	两位数字的小时（前面不补 0），在 0 到 12 之间	6
M	两位数字的分钟（前面补 0）	05
S	两位数字的秒（前面补 0）	19
L	三位数字的毫秒（前面补 0）	047
N	九位数字的毫微秒（前面补 0）	047000000
P	上午或下午的大写标志	PM
p	上午或下午的小写标志	pm
z	从 GMT 起，RFC822 数字位移	− 0800
Z	时区	PST
s	从格林威治时间 1970-01-01 00:00:00 起的秒数	1078884319
Q	从格林威治时间 1970-01-01 00:00:00 起的毫秒数	1078884319047

从表 3-7 可以看到，某些格式只给出了指定日期的部分信息。例如，只有日期或月份。如果需要多次对日期操作才能实现对每一部分进行格式化的目的就太笨拙了。为此，可以采用一个格式化的字符串指出要被格式化的参数索引。索引必须紧跟在 % 后面，并以 $ 终止。例如，

```
System.out.printf("%1$s %2$tB %2$te, %2$tY", "Due date:", new Date());
```

打印

```
Due date: February 9, 2004
```

还可以选择使用 < 标志。它指示前面格式说明中的参数将被再次使用。也就是说，下列语句将产生与前面语句同样的输出结果：

```
System.out.printf("%s %tB %<te, %<tY", "Due date:", new Date());
```

提示：参数索引值从 1 开始，而不是从 0 开始，%1$... 对第 1 个参数格式化。这就避免了与 0 标志混淆。

现在，已经了解了 printf 方法的所有特性。图 3-6 给出了格式说明符的语法图。

注释：许多格式化规则是本地环境特有的。例如，在德国，组分隔符是句号而不是逗号，Monday 被格式化为 Montag。在卷 II 第 5 章中将介绍如何控制应用程序与地域有关的行为。

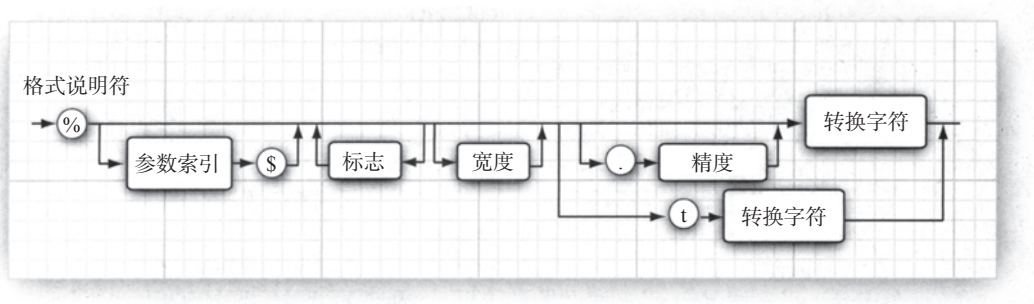

图 3-6 格式说明符语法

3.7.3 文件输入与输出

要想对文件进行读取，就需要一个用 File 对象构造一个 Scanner 对象，如下所示：

```
Scanner in = new Scanner(Paths.get("myfile.txt"));
```

如果文件名中包含反斜杠符号，就要记住在每个反斜杠之前再加一个额外的反斜杠："c:\\mydirectory\\myfile.txt"。

现在，就可以利用前面介绍的任何一个 Scanner 方法对文件进行读取。

要想写入文件，就需要构造一个 PrintWriter 对象。在构造器中，只需要提供文件名：

```
PrintWriter out = new PrintWriter("myfile.txt");
```

如果文件不存在，创建该文件。可以像输出到 System.out 一样使用 print、println 以及 printf 命令。

警告：可以构造一个带有字符串参数的 Scanner，但这个 Scanner 将字符串解释为数据，而不是文件名。例如，如果调用：

```
Scanner in = new Scanner("myfile.txt"); // ERROR?
```

这个 scanner 会将参数作为包含 10 个字符的数据：'m'，'y'，'f' 等。在这个示例中所显示的并不是人们所期望的效果。

注释：当指定一个相对文件名时，例如，"myfile.txt"，"mydirectory/myfile.txt" 或 "../myfile.txt"，文件位于 Java 虚拟机启动路径的相对位置。如果在命令行方式下用下列命令启动程序：

```
java MyProg
```

启动路径就是命令解释器的当前路径。然而，如果使用集成开发环境，那么启动路径将由 IDE 控制。可以使用下面的调用方式找到路径的位置：

```
String dir = System.getProperty("user.dir");
```

如果觉得定位文件比较烦恼，则可以考虑使用绝对路径，例如："c:\\mydirectory\\myfile.txt" 或者 "/home/me/mydirectory/myfile.txt"。

正如读者所看到的，访问文件与使用 System.in 和 System.out 一样容易。要记住一点：如果用一个不存在的文件构造一个 Scanner，或者用一个不能被创建的文件名构造一个

PrintWriter，那么就会发生异常。Java 编译器认为这些异常比“被零整除”异常更严重。在第 11 章中，将会学习各种处理异常的方式。现在，应该告知编译器：已经知道有可能出现“找不到文件”的异常。这需要在 main 方法中用 throws 子句标记，如下所示：

```
public static void main(String[] args) throws FileNotFoundException
{
   Scanner in = new Scanner(Paths.get("myfile.txt"));
   . . .
}
```

现在读者已经学习了如何读写包含文本数据的文件。对于更加高级的技术，例如，处理不同的字符编码、处理二进制数据、读取目录以及编写压缩文件，请参看卷 II 第 1 章。

注释：当采用命令行方式启动一个程序时，可以利用重定向将任意文件捆绑到 System.in 和 System.out:

```
java MyProg < myfile.txt > output.txt
```

这样，就不必担心处理 FileNotFoundException 异常了。

API java.util.Scanner 5.0

- `Scanner(File f)`
  构造一个从给定文件读取数据的 Scanner。
- `Scanner(String data)`
  构造一个从给定字符串读取数据的 Scanner。

API java.io.PrintWriter 1.1

- `PrintWriter(String fileName)`
  构造一个将数据写入文件的 PrintWriter。文件名由参数指定。

API java.nio.file.Paths 7

- `static Path get(String pathname)`
  根据给定的路径名构造一个 Path。

3.8 控制流程

与任何程序设计语言一样，Java 使用条件语句和循环结构确定控制流程。本节先讨论条件语句，然后讨论循环语句，最后介绍看似有些笨重的 switch 语句，当需要对某个表达式的多个值进行检测时，可以使用 switch 语句。

C++ 注释：Java 的控制流程结构与 C 和 C++ 的控制流程结构一样，只有很少的例外情况。没有 goto 语句，但 break 语句可以带标签，可以利用它实现从内层循环跳出的目的（这种情况 C 语言采用 goto 语句实现）。另外，还有一种变形的 for 循环，在 C 或 C++ 中没有这类循环。它有点类似于 C# 中的 foreach 循环。

3.8.1 块作用域

在深入学习控制结构之前，需要了解块（block）的概念。

块（即复合语句）是指由一对花括号括起来的若干条简单的Java语句。块确定了变量的作用域。一个块可以嵌套在另一个块中。下面就是在main方法块中嵌套另一个语句块的示例。

```
public static void main(String[] args)
{
   int n;
   . . .
   {
      int k;
      . . .
   } // k is only defined up to here
}
```

但是，不能在嵌套的两个块中声明同名的变量。例如，下面的代码就有错误，而无法通过编译：

```
public static void main(String[] args)
{
   int n;
   . . .
   {
      int k;
      int n; // ERROR--can't redefine n in inner block
      . . .
   }
}
```

C++ **C++ 注释：**在C++中，可以在嵌套的块中重定义一个变量。在内层定义的变量会覆盖在外层定义的变量。这样，有可能会导致程序设计错误，因此在Java中不允许这样做。

3.8.2 条件语句

在Java中，条件语句的格式为

```
if (condition) statement
```

这里的条件必须用括号括起来。

与绝大多数程序设计语言一样，Java常常希望在某个条件为真时执行多条语句。在这种情况下，应该使用块语句（block statement），格式为

```
{
  statement1
  statement2
  . . .
}
```

例如：

```
if (yourSales >= target)
{
```

```
    performance = "Satisfactory";
    bonus = 100;
}
```

当 yourSales 大于或等于 target 时，将执行括号中的所有语句（请参看图 3-7）。

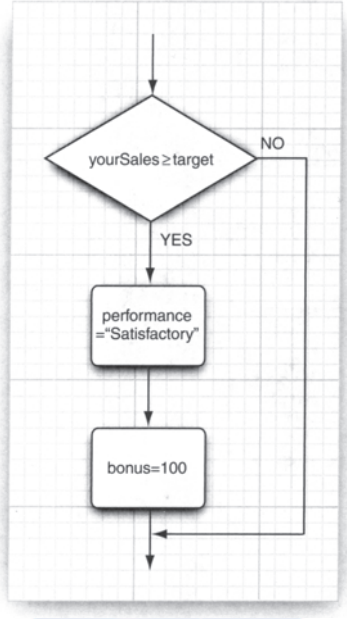

图 3-7 if 语句的流程图

注释：使用块（有时称为复合语句）可以在 Java 程序结构中原本只能放置一条简单语句的地方放置多条语句。

在 Java 中，比较常见的条件语句格式如下所示（请参看图 3-8）：

if (*condition*) *statement*$_1$ else *statement*$_2$

例如：

```
if (yourSales >= target)
{
   performance = "Satisfactory";
   bonus = 100 + 0.01 * (yourSales - target);
}
else
{
   performance = "Unsatisfactory";
   bonus = 0;
}
```

其中 else 部分是可选的。else 子句与最邻近的 if 构成一组。因此，在语句

```
if (x <= 0) if (x == 0) sign = 0; else sign = -1;
```

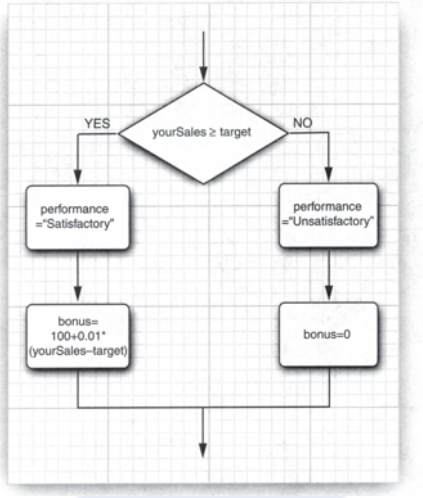

图 3-8 if/else 语句的流程图

中 else 与第 2 个 if 配对。当然，用一对括号将会使这段代码更加清晰：

```
if (x <= 0) { if (x == 0) sign = 0; else sign = -1; }
```

重复地交替出现 if...else if... 是一种很常见的情况（请参看图 3-9）。例如：

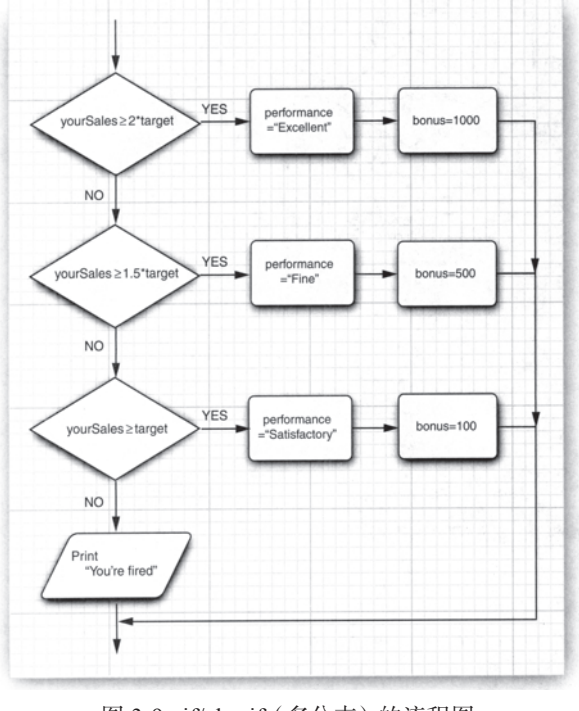

图 3-9 if/else if（多分支）的流程图

```
if (yourSales >= 2 * target)
{
   performance = "Excellent";
   bonus = 1000;
}
else if (yourSales >= 1.5 * target)
{
   performance = "Fine";
   bonus = 500;
}
else if (yourSales >= target)
{
   performance = "Satisfactory";
   bonus = 100;
}
else
{
   System.out.println("You're fired");
}
```

3.8.3 循环

当条件为 true 时，while 循环执行一条语句（也可以是一个语句块）。常用的格式为

```
while (condition) statement
```

如果开始循环条件的值就为 false，则 while 循环体一次也不执行（请参看图 3-10）。

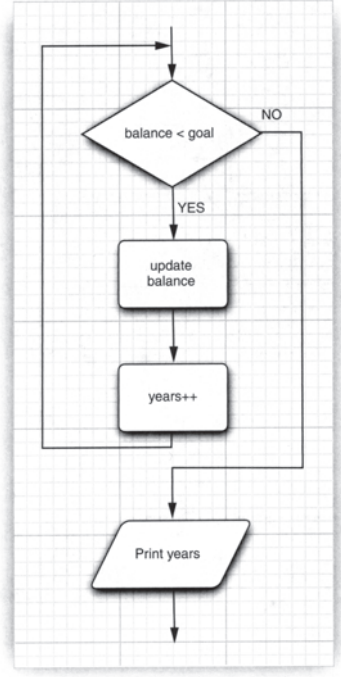

图 3-10 while 语句的流程图

程序清单 3-3 中的程序将计算需要多长时间才能够存储一定数量的退休金，假定每年存入相同数量的金额，而且利率是固定的。

在这个示例中，增加了一个计数器，并在循环体中更新当前的累积数量，直到总值超过目标值为止。

```
while (balance < goal)
{
   balance += payment;
   double interest = balance * interestRate / 100;
   balance += interest;
   years++;
}
System.out.println(years + " years.");
```

（千万不要使用这个程序安排退休计划。这里忽略了通货膨胀和所期望的生活水准。）

while 循环语句首先检测循环条件。因此，循环体中的代码有可能不被执行。如果希望循环体至少执行一次，则应该将检测条件放在最后。使用 do/while 循环语句可以实现这种操作方式。它的语法格式为：

```
do statement while (condition);
```

这种循环语句先执行语句（通常是一个语句块），再检测循环条件；然后重复语句，再检测循环条件，以此类推。在程序清单 3-4 中，首先计算退休账户中的余额，然后再询问是否打算退休：

```
do
{
   balance += payment;
   double interest = balance * interestRate / 100;
   balance += interest;
   year++;
   // print current balance
   . . .
   // ask if ready to retire and get input
   . . .
}
while (input.equals("N"));
```

只要用户回答“N”，循环就重复执行（见图 3-11）。这是一个需要至少执行一次的循环的很好示例，因为用户必须先看到余额才能知道是否满足退休所用。

程序清单 3-3 Retirement/Retirement.java

```
1  import java.util.*;
2
3  /**
4   * This program demonstrates a <code>while</code> loop.
5   * @version 1.20 2004-02-10
6   * @author Cay Horstmann
7   */
8  public class Retirement
9  {
```

```
10    public static void main(String[] args)
11    {
12       // read inputs
13       Scanner in = new Scanner(System.in);
14
15       System.out.print("How much money do you need to retire? ");
16       double goal = in.nextDouble();
17
18       System.out.print("How much money will you contribute every year? ");
19       double payment = in.nextDouble();
20
21       System.out.print("Interest rate in %: ");
22       double interestRate = in.nextDouble();
23
24       double balance = 0;
25       int years = 0;
26
27       // update account balance while goal isn't reached
28       while (balance < goal)
29       {
30          // add this year's payment and interest
31          balance += payment;
32          double interest = balance * interestRate / 100;
33          balance += interest;
34          years++;
35       }
36
37       System.out.println("You can retire in " + years + " years.");
38    }
39 }
```

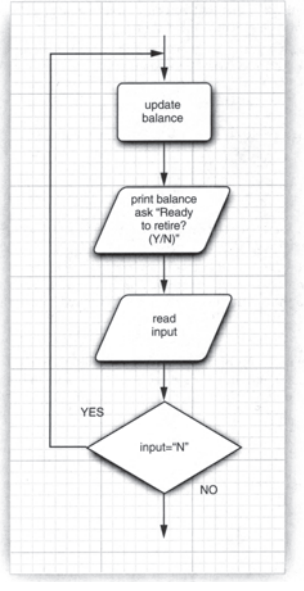

图 3-11 do/while 语句的流程图

程序清单 3-4 Retirement2/Retirement2.java

```
1  import java.util.*;
2
3  /**
4   * This program demonstrates a <code>do/while</code> loop.
5   * @version 1.20 2004-02-10
6   * @author Cay Horstmann
7   */
8  public class Retirement2
9  {
10    public static void main(String[] args)
11    {
12       Scanner in = new Scanner(System.in);
13
14       System.out.print("How much money will you contribute every year? ");
15       double payment = in.nextDouble();
16
17       System.out.print("Interest rate in %: ");
18       double interestRate = in.nextDouble();
19
20       double balance = 0;
21       int year = 0;
22
23       String input;
24
25       // update account balance while user isn't ready to retire
26       do
27       {
28          // add this year's payment and interest
29          balance += payment;
30          double interest = balance * interestRate / 100;
31          balance += interest;
32
33          year++;
34
35          // print current balance
36          System.out.printf("After year %d, your balance is %,.2f%n", year, balance);
37
38          // ask if ready to retire and get input
39          System.out.print("Ready to retire? (Y/N) ");
40          input = in.next();
41       }
42       while (input.equals("N"));
43    }
44 }
```

3.8.4 确定循环

for 循环语句是支持迭代的一种通用结构，利用每次迭代之后更新的计数器或类似的变量来控制迭代次数。如图 3-12 所示，下面的程序将数字 1 ~ 10 输出到屏幕上。

```
for (int i = 1; i <= 10; i++)
   System.out.println(i);
```

for 语句的第 1 部分通常用于对计数器初始化；第 2 部分给出每次新一轮循环执行前要检测的循环条件；第 3 部分指示如何更新计数器。

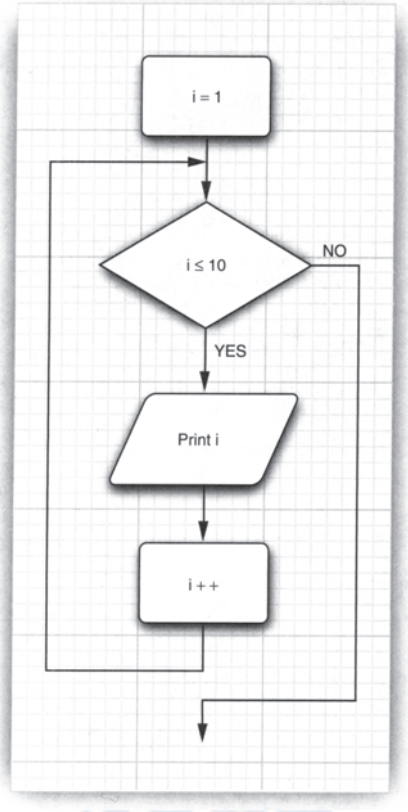

图 3-12 for 语句的流程图

与 C++ 一样，尽管 Java 允许在 for 循环的各个部分放置任何表达式，但有一条不成文的规则：for 语句的 3 个部分应该对同一个计数器变量进行初始化、检测和更新。若不遵守这一规则，编写的循环常常晦涩难懂。

即使遵守了这条规则，也还有可能出现很多问题。例如，下面这个倒计数的循环：

```
for (int i = 10; i > 0; i--)
   System.out.println("Counting down . . . " + i);
System.out.println("Blastoff!");
```

警告： 在循环中，检测两个浮点数是否相等需要格外小心。下面的 for 循环

```
for (double x = 0; x != 10; x += 0.1) . . .
```

可能永远不会结束。由于舍入的误差，最终可能得不到精确值。例如，在上面的循环中，因为 0.1 无法精确地用二进制表示，所以，x 将从 9.999 999 999 999 98 跳到 10.099 999 999 999 98。

当在 for 语句的第 1 部分中声明了一个变量之后，这个变量的作用域就为 for 循环的整个循环体。

```
for (int i = 1; i <= 10; i++)
{
   . . .
}
// i no longer defined here
```

特别指出，如果在 for 语句内部定义一个变量，这个变量就不能在循环体之外使用。因此，如果希望在 for 循环体之外使用循环计数器的最终值，就要确保这个变量在循环语句的前面且在外部声明！

```
int i;
for (i = 1; i <= 10; i++)
{
   . . .
}
// i is still defined here
```

另一方面，可以在各自独立的不同 for 循环中定义同名的变量：

```
for (int i = 1; i <= 10; i++)
{
   . . .
}
. . .
for (int i = 11; i <= 20; i++) // OK to define another variable named i
{
   . . .
}
```

for 循环语句只不过是 while 循环的一种简化形式。例如，

```
for (int i = 10; i > 0; i--)
   System.out.println("Counting down . . . " + i);
```

可以重写为：

```
int i = 10;
while (i > 0)
{
   System.out.println("Counting down . . . " + i);
   i--;
}
```

程序清单 3-5 给出了一个应用 for 循环的典型示例。这个程序用来计算抽奖中奖的概率。例如，如果必须从 1 ~ 50 之间的数字中取 6 个数字来抽奖，那么会有 $(50 \times 49 \times 48 \times 47 \times 46 \times 45)/(1 \times 2 \times 3 \times 4 \times 5 \times 6)$ 种可能的结果，所以中奖的几率是 1/15 890 700。祝你好运！

一般情况下，如果从 n 个数字中抽取 k 个数字，就可以使用下列公式得到结果。

$$\frac{n \times (n-1) \times (n-2) \times \cdots \times (n-k+1)}{1 \times 2 \times 3 \times \cdots \times k}$$

下面的 for 循环语句计算了上面这个公式的值：

```
int lotteryOdds = 1;
for (int i = 1; i <= k; i++)
   lotteryOdds = lotteryOdds * (n - i + 1) / i;
```

注释：3.10.1 节将会介绍“通用 for 循环”(又称为 for each 循环)，这是 Java SE 5.0 新增加的一种循环结构。

程序清单 3-5 LotteryOdds/LotteryOdds.java

```
1  import java.util.*;
2  /**
3   * This program demonstrates a <code>for</code> loop.
4   * @version 1.20 2004-02-10
5   * @author Cay Horstmann
6   */
7  public class LotteryOdds
8  {
9     public static void main(String[] args)
10    {
11       Scanner in = new Scanner(System.in);
12
13       System.out.print("How many numbers do you need to draw? ");
14       int k = in.nextInt();
15
16       System.out.print("What is the highest number you can draw? ");
17       int n = in.nextInt();
18       /*
19        * compute binomial coefficient n*(n-1)*(n-2)*...*(n-k+1)/(1*2*3*...*k)
20        */
21       int lotteryOdds = 1;
22       for (int i = 1; i <= k; i++)
23          lotteryOdds = lotteryOdds * (n - i + 1) / i;
24
25       System.out.println("Your odds are 1 in " + lotteryOdds + ". Good luck!");
26    }
27 }
```

3.8.5 多重选择：switch 语句

在处理多个选项时，使用 if/else 结构显得有些笨拙。Java 有一个与 C/C++ 完全一样的 switch 语句。

例如，如果建立一个如图 3-13 所示的包含 4 个选项的菜单系统，就应该使用下列代码：

```
Scanner in = new Scanner(System.in);
System.out.print("Select an option (1, 2, 3, 4) ");
int choice = in.nextInt();
switch (choice)
{
   case 1:
      . . .
      break;
   case 2:
```

```
        . . .
        break;
    case 3:
        . . .
        break;
    case 4:
        . . .
        break;
    default:
        // bad input
        . . .
        break;
}
```

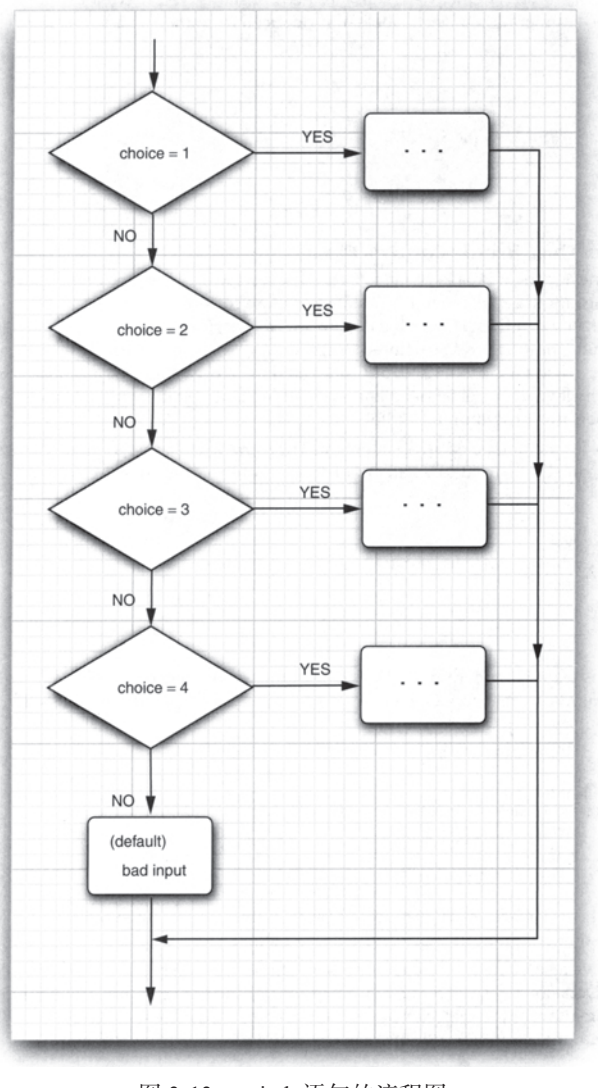

图 3-13 switch 语句的流程图

switch 语句将从与选项值相匹配的 case 标签处开始执行直到遇到 break 语句，或者执行到 switch 语句的结束处为止。如果没有相匹配的 case 标签，而有 default 子句，就执行这个子句。

警告：有可能触发多个 case 分支。如果在 case 分支语句的末尾没有 break 语句，那么就会接着执行下一个 case 分支语句。这种情况相当危险，常常会引发错误。为此，我们在程序中从不使用 switch 语句。

如果你比我们喜欢 switch 语句，编译代码时可以考虑加上 -Xlint:fallthrough 选项，如下所示：

```
javac -Xlint:fallthrough Test.java
```

这样一来，如果某个分支最后缺少一个 break 语句，编译器就会给出一个警告消息。

如果你确实正是想使用这种“直通式”（fallthrough）行为，可以为其外围方法加一个标注 @SuppressWarnings("fallthrough")。这样就不会对这个方法生成警告了。（标注是为编译器或处理 Java 源文件或类文件的工具提供信息的一种机制。我们将在卷 II 的第 13 章详细讨论标注。）

case 标签可以是：

- 类型为 char、byte、short 或 int（或其包装器类 Character、Byte、Short 和 Intege，这些包装器类将在第 4 章介绍）的常量表达式。
- 枚举常量。
- 从 Java SE 7 开始，case 标签还可以是字符串字面量。

例如：

```
String input = . . .;
switch (input.toLowerCase())
{
   case "yes": // OK since Java SE 7
      . . .
      break;
   . . .
}
```

当在 switch 语句中使用枚举常量时，不必在每个标签中指明枚举名，可以由 switch 的表达式值确定。例如：

```
Size sz = . . .;
switch (sz)
{
   case SMALL: // no need to use Size.SMALL
      . . .
      break;
   . . .
}
```

3.8.6 中断控制流程语句

尽管 Java 的设计者将 goto 作为保留字，但实际上并没有打算在语言中使用它。通常，使用 goto 语句被认为是一种拙劣的程序设计风格。当然，也有一些程序员认为反对 goto 的呼声似乎有些过分（例如，Donald Knuth 就曾编著过一篇名为《 Structured Programming with goto statements 》的著名文章）。这篇文章说：无限制地使用 goto 语句确实是导致错误的根源，但在有些情况下，偶尔使用 goto 跳出循环还是有益处的。Java 设计者同意这种看法，甚至在 Java 语言中增加了一条带标签的 break，以此来支持这种程序设计风格。

下面首先看一下不带标签的 break 语句。与用于退出 switch 语句的 break 语句一样，它也可以用于退出循环语句。例如，

```
while (years <= 100)
{
   balance += payment;
   double interest = balance * interestRate / 100;
   balance += interest;
   if (balance >= goal) break;
   years++;
}
```

在循环开始时，如果 years > 100，或者在循环体中 balance ≥ goal，则退出循环语句。当然，也可以在不使用 break 的情况下计算 years 的值，如下所示：

```
while (years <= 100 && balance < goal)
{
   balance += payment;
   double interest = balance * interestRate / 100;
   balance += interest;
   if (balance < goal)
      years++;
}
```

但是需要注意，在这个版本中，检测了两次 balance < goal。为了避免重复检测，有些程序员更加偏爱使用 break 语句。

与 C++ 不同，Java 还提供了一种带标签的 break 语句，用于跳出多重嵌套的循环语句。有时候，在嵌套很深的循环语句中会发生一些不可预料的事情。此时可能更加希望跳到嵌套的所有循环语句之外。通过添加一些额外的条件判断实现各层循环的检测很不方便。

这里有一个示例说明了 break 语句的工作状态。请注意，标签必须放在希望跳出的最外层循环之前，并且必须紧跟一个冒号。

```
Scanner in = new Scanner(System.in);
int n;
read_data:
while (. . .) // this loop statement is tagged with the label
{
   . . .
   for (. . .) // this inner loop is not labeled
   {
      System.out.print("Enter a number >= 0: ");
```

```
            n = in.nextInt();
            if (n < 0) // should never happen-can't go on
               break read_data;
               // break out of read_data loop
            . . .
         }
      }
      // this statement is executed immediately after the labeled break
      if (n < 0) // check for bad situation
      {
         // deal with bad situation
      }
      else
      {
         // carry out normal processing
      }
```

如果输入有误，通过执行带标签的 break 跳转到带标签的语句块末尾。对于任何使用 break 语句的代码都需要检测循环是正常结束，还是由 break 跳出。

注释：事实上，可以将标签应用到任何语句中，甚至可以应用到 if 语句或者块语句中，如下所示：

```
label:
{
   . . .
   if (condition) break label; // exits block
   . . .
}
// jumps here when the break statement executes
```

因此，如果希望使用一条 goto 语句，并将一个标签放在想要跳到的语句块之前，就可以使用 break 语句！当然，并不提倡使用这种方式。另外需要注意，只能跳出语句块，而不能跳入语句块。

最后，还有一个 continue 语句。与 break 语句一样，它将中断正常的控制流程。continue 语句将控制转移到最内层循环的首部。例如：

```
Scanner in = new Scanner(System.in);
while (sum < goal)
{
   System.out.print("Enter a number: ");
   n = in.nextInt();
   if (n < 0) continue;
   sum += n; // not executed if n < 0
}
```

如果 n<0，则 continue 语句越过了当前循环体的剩余部分，立刻跳到循环首部。

如果将 continue 语句用于 for 循环中，就可以跳到 for 循环的“更新”部分。例如，下面这个循环：

```
for (count = 1; count <= 100; count++)
{
   System.out.print("Enter a number, -1 to quit: ");
```

```
      n = in.nextInt();
      if (n < 0) continue;
      sum += n; // not executed if n < 0
   }
```

如果 n<0，则 continue 语句跳到 count++ 语句。

还有一种带标签的 continue 语句，将跳到与标签匹配的循环首部。

提示：许多程序员容易混淆 break 和 continue 语句。这些语句完全是可选的，即不使用它们也可以表达同样的逻辑含义。在本书中，将不使用 break 和 continue。

3.9 大数值

如果基本的整数和浮点数精度不能够满足需求，那么可以使用 java.math 包中的两个很有用的类：BigInteger 和 BigDecimal。这两个类可以处理包含任意长度数字序列的数值。BigInteger 类实现了任意精度的整数运算，BigDecimal 实现了任意精度的浮点数运算。

使用静态的 valueOf 方法可以将普通的数值转换为大数值：

```
BigInteger a = BigInteger.valueOf(100);
```

遗憾的是，不能使用人们熟悉的算术运算符（如：+ 和 *）处理大数值。而需要使用大数值类中的 add 和 multiply 方法。

```
BigInteger c = a.add(b); // c = a + b
BigInteger d = c.multiply(b.add(BigInteger.valueOf(2))); // d = c * (b + 2)
```

C++ 注释：与 C++ 不同，Java 没有提供运算符重载功能。程序员无法重定义 + 和 * 运算符，使其应用于 BigInteger 类的 add 和 multiply 运算。Java 语言的设计者确实为字符串的连接重载了 + 运算符，但没有重载其他的运算符，也没有给 Java 程序员在自己的类中重载运算符的机会。

程序清单 3-6 是对程序清单 3-5 中彩概率程序的改进，使其可以采用大数值进行运算。假设你被邀请参加抽奖活动，并从 490 个可能的数值中抽取 60 个，这个程序将会得到中彩概率 1/716395843461995557415116222540092933411717612789263493493351013459481104668848。祝你好运！

在程序清单 3-5 中，用于计算的语句是

```
lotteryOdds = lotteryOdds * (n - i + 1) / i;
```

如果使用大数值，则相应的语句为：

```
lotteryOdds = lotteryOdds.multiply(BigInteger.valueOf(n - i + 1)).divide(BigInteger.valueOf(i));
```

程序清单 3-6 BigIntegerTest/BigIntegerTest.java

```
1 import java.math.*;
2 import java.util.*;
3
4 /**
5  * This program uses big numbers to compute the odds of winning the grand prize in a lottery.
```

```
 * @version 1.20 2004-02-10
 * @author Cay Horstmann
 */
public class BigIntegerTest
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.print("How many numbers do you need to draw? ");
      int k = in.nextInt();

      System.out.print("What is the highest number you can draw? ");
      int n = in.nextInt();

      /*
       * compute binomial coefficient n*(n-1)*(n-2)*...*(n-k+1)/(1*2*3*...*k)
       */

      BigInteger lotteryOdds = BigInteger.valueOf(1);

      for (int i = 1; i <= k; i++)
         lotteryOdds = lotteryOdds.multiply(BigInteger.valueOf(n - i + 1)).divide(
               BigInteger.valueOf(i));

      System.out.println("Your odds are 1 in " + lotteryOdds + ". Good luck!");
   }
}
```

API API java.math.BigInteger 1.1

- `BigInteger add(BigInteger other)`
- `BigInteger subtract(BigInteger other)`
- `BigInteger multiply(BigInteger other)`
- `BigInteger divide(BigInteger other)`
- `BigInteger mod(BigInteger other)`

  返回这个大整数和另一个大整数 other 的和、差、积、商以及余数。
- `int compareTo(BigInteger other)`

  如果这个大整数与另一个大整数 other 相等，返回 0；如果这个大整数小于另一个大整数 other，返回负数；否则，返回正数。
- `static BigInteger valueOf(long x)`

  返回值等于 x 的大整数。

API java.math.BigInteger 1.1

- `BigDecimal add(BigDecimal other)`
- `BigDecimal subtract(BigDecimal other)`

- `BigDecimal multiply(BigDecimal other)`
- `BigDecimal divide(BigDecimal other RoundingMode mode) 5.0`

  返回这个大实数与另一个大实数other的和、差、积、商。要想计算商，必须给出舍入方式（rounding mode）。RoundingMode.HALF_UP是在学校中学习的四舍五入方式（即，数值0到4舍去，数值5到9进位）。它适用于常规的计算。有关其他的舍入方式请参看API文档。
- `int compareTo(BigDecimal other)`

  如果这个大实数与另一个大实数相等，返回0；如果这个大实数小于另一个大实数，返回负数；否则，返回正数。
- `static BigDecimal valueOf(long x)`
- `static BigDecimal valueOf(long x,int scale)`

  返回值为x或$x / 10^{scale}$的一个大实数。

3.10 数组

数组是一种数据结构，用来存储同一类型值的集合。通过一个整型下标可以访问数组中的每一个值。例如，如果a是一个整型数组，a[i]就是数组中下标为i的整数。

在声明数组变量时，需要指出数组类型（数据元素类型紧跟[]）和数组变量的名字。下面声明了整型数组a：

```
int[] a;
```

这条语句只声明了变量a，并没有将a初始化为一个真正的数组。应该使用new运算符创建数组。

```
int[] a = new int[100];
```

这条语句创建了一个可以存储100个整数的数组。数组长度不要求是常量：new int[n]会创建一个长度为n的数组。

注释：可以使用下面两种形式声明数组

```
int[] a;
```

或

```
int[] a;
```

大多数Java应用程序员喜欢使用第一种风格，因为它将类型int[]（整型数组）与变量名分开了。

这个数组的下标从0 ~ 99（不是1 ~ 100）。一旦创建了数组，就可以给数组元素赋值。例如，使用一个循环：

```
int[] a = new int[100];
for (int i = 0; i < 100; i++)
   a[i] = i; // fills the array with numbers 0 to 99
```

创建一个数字数组时，所有元素都初始化为 0。boolean 数组的元素会初始化为 false。对象数组的元素则初始化为一个特殊值 null，这表示这些元素（还）未存放任何对象。初学者对此可能有些不解。例如，

```
String[] names = new String[10];
```

会创建一个包含 10 个字符串的数组，所有字符串都为 null。如果希望这个数组包含空串，可以为元素指定空串：

```
for (int i = 0; i < 10; i++) names[i] = "";
```

警告：如果创建了一个 100 个元素的数组，并且试图访问元素 a[100]（或任何在 0 ~ 99 之外的下标），程序就会引发“array index out of bounds”异常而终止执行。

要想获得数组中的元素个数，可以使用 array.length。例如，

```
for (int i = 0; i < a.length; i++)
   System.out.println(a[i]);
```

一旦创建了数组，就不能再改变它的大小（尽管可以改变每一个数组元素）。如果经常需要在运行过程中扩展数组的大小，就应该使用另一种数据结构——数组列表（array list）有关数组列标的详细内容请参看第 5 章。

3.10.1 for each 循环

Java 有一种功能很强的循环结构，可以用来依次处理数组中的每个元素（其他类型的元素集合亦可）而不必为指定下标值而分心。

这种增强的 for 循环的语句格式为：

```
for (variable : collection) statement
```

定义一个变量用于暂存集合中的每一个元素，并执行相应的语句（当然，也可以是语句块）。collection 这一集合表达式必须是一个数组或者是一个实现了 Iterable 接口的类对象（例如 ArrayList）。有关数组列表的内容将在第 5 章中讨论，有关 Iterable 接口的内容将在卷 II 的第 2 章中讨论。

例如，

```
for (int element : a)
   System.out.println(element);
```

打印数组 a 的每一个元素，一个元素占一行。

这个循环应该读作“循环 a 中的每一个元素”（for each element in a）。Java 语言的设计者认为应该使用诸如 foreach、in 这样的关键字，但这种循环语句并不是最初就包含在 Java 语言中的，而是后来添加进去的，并且没有人打算废除已经包含同名（例如 System.in）方法或变量的旧代码。

当然，使用传统的 for 循环也可以获得同样的效果：

```
for (int i = 0; i < a.length; i++)
   System.out.println(a[i]);
```

但是，for each 循环语句显得更加简洁、更不易出错（不必为下标的起始值和终止值而操心）。

注释： for each 循环语句的循环变量将会遍历数组中的每个元素，而不需要使用下标值。

如果需要处理一个集合中的所有元素，for each 循环语句对传统循环语句所进行的改进更是叫人称赞不已。然而，在很多场合下，还是需要使用传统的 for 循环。例如，如果不希望遍历集合中的每个元素，或者在循环内部需要使用下标值等。

提示： 有个更加简单的方式打印数组中的所有值，即利用 Arrays 类的 toString 方法。调用 Arrays.toString(a)，返回一个包含数组元素的字符串，这些元素被放置在括号内，并用逗号分隔，例如，“[2,3,5,7,11,13]”。要想打印数组，可以调用

```
System.out.println(Arrays.toString(a));
```

3.10.2 数组初始化以及匿名数组

在 Java 中，提供了一种创建数组对象并同时赋予初始值的简化书写形式。下面是一个例子：

```
int[] smallPrimes = { 2, 3, 5, 7, 11, 13 };
```

请注意，在使用这种语句时，不需要调用 new。

甚至还可以初始化一个匿名的数组：

```
new int[] { 17, 19, 23, 29, 31, 37 }
```

这种表示法将创建一个新数组并利用括号中提供的值进行初始化，数组的大小就是初始值的个数。使用这种语法形式可以在不创建新变量的情况下重新初始化一个数组。例如：

```
smallPrimes = new int[] { 17, 19, 23, 29, 31, 37 };
```

这是下列语句的简写形式：

```
int[] anonymous = { 17, 19, 23, 29, 31, 37 };
smallPrimes = anonymous;
```

注释： 在 Java 中，允许数组长度为 0。在编写一个结果为数组的方法时，如果碰巧结果为空，则这种语法形式就显得非常有用。此时可以创建一个长度为 0 的数组：

```
new elementType[0]
```

注意，数组长度为 0 与 null 不同。

3.10.3 数组拷贝

在 Java 中，允许将一个数组变量拷贝给另一个数组变量。这时，两个变量将引用同一个数组：

```
int[] luckyNumbers = smallPrimes;
luckyNumbers[5] = 12; // now smallPrimes[5] is also 12
```

图 3-14 显示了拷贝的结果。如果希望将一个数组的所有值拷贝到一个新的数组中去，就要使用 Arrays 类的 copyTo 方法：

```
int[] copiedLuckyNumbers = Arrays.copyOf(luckyNumbers, luckyNumbers.length);
```

第 2 个参数是新数组的长度。这个方法通常用来增加数组的大小：

```
luckyNumbers = Arrays.copyOf(luckyNumbers, 2 * luckyNumbers.length);
```

如果数组元素是数值型，那么多余的元素将被赋值为 0；如果数组元素是布尔型，则将赋值为 false。相反，如果长度小于原始数组的长度，则只拷贝最前面的数据元素。

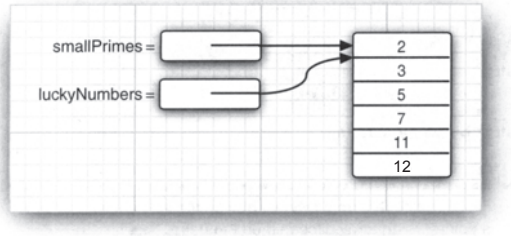

图 3-14 拷贝一个数组变量

C++ 注释：Java 数组与 C++ 数组在堆栈上有很大不同，但基本上与分配在堆（heap）上的数组指针一样。也就是说，

```
int[] a = new int[100]; // Java
```

不同于

```
int a[100]; // C++
```

而等同于

```
int* a = new int[100]; // C++
```

Java 中的 [ ] 运算符被预定义为检查数组边界，而且没有指针运算，即不能通过 a 加 1 得到数组的下一个元素。

3.10.4 命令行参数

前面已经看到多个使用 Java 数组的示例。每一个 Java 应用程序都有一个带 String arg[] 参数的 main 方法。这个参数表明 main 方法将接收一个字符串数组，也就是命令行参数。

例如，看一看下面这个程序：

```
public class Message
{
   public static void main(String[] args)
   {
      if (args[0].equals("-h"))
         System.out.print("Hello,");
      else if (args[0].equals("-g"))
         System.out.print("Goodbye,");
      // print the other command-line arguments
      for (int i = 1; i < args.length; i++)
         System.out.print(" " + args[i]);
      System.out.println("!");
   }
}
```

如果使用下面这种形式运行这个程序：

```
java Message -g cruel world
```

args 数组将包含下列内容：

```
args[0]: "-g"
args[1]: "cruel"
args[2]: "world"
```

这个程序将显示下列信息：

```
Goodbye, cruel world!
```

C++ **C++ 注释：** 在 Java 应用程序的 main 方法中，程序名并没有存储在 args 数组中。例如，当使用下列命令运行程序时

```
java Message -h world
```

args[0] 是“— h”，而不是“Message”或“java”。

3.10.5 数组排序

要想对数值型数组进行排序，可以使用 Arrays 类中的 sort 方法：

```
int[] a = new int[10000];
. . .
Arrays.sort(a)
```

这个方法使用了优化的快速排序算法。快速排序算法对于大多数数据集合来说都是效率比较高的。Array 类还提供了几个使用很便捷的方法，在稍后的 API 注释中将介绍它们。

程序清单 3-7 中的程序用到了数组，它产生一个抽彩游戏中的随机数值组合。假如抽彩是从 49 个数值中抽取 6 个，那么程序可能的输出结果为：

```
Bet the following combination. It'll make you rich!
  4
  7
  8
  19
  30
  44
```

要想选择这样一个随机的数值集合，就要首先将数值 1，2，…，n 存入数组 numbers 中：

```
int[] numbers = new int[n];
for (int i = 0; i < numbers.length; i++)
  numbers[i] = i + 1;
```

而用第二个数组存放抽取出来的数值：

```
int[] result = new int[k];
```

现在，就可以开始抽取 k 个数值了。Math.random 方法将返回一个 0 到 1 之间（包含 0、不包含 1）的随机浮点数。用 n 乘以这个浮点数，就可以得到从 0 到 n — 1 之间的一个随机数。

```
int r = (int) (Math.random() * n);
```

下面将 result 的第 i 个元素设置为 numbers[r] 存放的数值，最初是 r+1。但正如所看到的，numbers 数组的内容在每一次抽取之后都会发生变化。

```
result[i] = numbers[r];
```

现在，必须确保不会再次抽取到那个数值，因为所有抽彩的数值必须不相同。因此，这里用数组中的最后一个数值改写 number[r]，并将 n 减 1。

```
numbers[r] = numbers[n - 1];
n--;
```

关键在于每次抽取的都是下标，而不是实际的值。下标指向包含尚未抽取过的数组元素。

在抽取了 k 个数值之后，就可以对 result 数组进行排序了，这样可以让输出效果更加清晰：

```
Arrays.sort(result);
for (int r : result)
   System.out.println(r);
```

程序清单 3-7　LotteryDrawing/LotteryDrawing.java

```
1  import java.util.*;
2
3  /**
4   * This program demonstrates array manipulation.
5   * @version 1.20 2004-02-10
6   * @author Cay Horstmann
7   */
8  public class LotteryDrawing
9  {
10    public static void main(String[] args)
11    {
12       Scanner in = new Scanner(System.in);
13
14       System.out.print("How many numbers do you need to draw? ");
15       int k = in.nextInt();
16
17       System.out.print("What is the highest number you can draw? ");
18       int n = in.nextInt();
19
20       // fill an array with numbers 1 2 3 . . . n
21       int[] numbers = new int[n];
22       for (int i = 0; i < numbers.length; i++)
23          numbers[i] = i + 1;
24
25       // draw k numbers and put them into a second array
26       int[] result = new int[k];
27       for (int i = 0; i < result.length; i++)
28       {
29          // make a random index between 0 and n - 1
30          int r = (int) (Math.random() * n);
31
32          // pick the element at the random location
33          result[i] = numbers[r];
34
35          // move the last element into the random location
```

```
36            numbers[r] = numbers[n - 1];
37            n--;
38         }
39
40         // print the sorted array
41         Arrays.sort(result);
42         System.out.println("Bet the following combination. It'll make you rich!");
43         for (int r : result)
44            System.out.println(r);
45      }
46   }
```

API java.util.Arrays 1.2

- `static String toString(type[] a)  5.0`

  返回包含 a 中数据元素的字符串，这些数据元素被放在括号内，并用逗号分隔。

  参数：a　　类型为 int、long、short、char、byte、boolean、float 或 double 的数组。

- `static type copyOf(type[] a, int length) 6`
- `static type copyOf(type[] a, int start, int end) 6`

  返回与 a 类型相同的一个数组，其长度为 length 或者 end-start，数组元素为 a 的值。

  参数：a　　类型为 int、long、short、char、byte、boolean、float 或 double 的数组。

  start　　起始下标（包含这个值）。

  end　　终止下标（不包含这个值）。这个值可能大于 a.length。在这种情况下，结果为 0 或 false。

  length　　拷贝的数据元素长度。如果 length 值大于 a.length，结果为 0 或 false；否则，数组中只有前面 length 个数据元素的拷贝值。

- `static void sort(type[] a)`

  采用优化的快速排序算法对数组进行排序。

  参数：a　　类型为 int、long、short、char、byte、boolean、float 或 double 的数组。

- `static int binarySearch(type[] a, type v)`
- `static int binarySearch(type[] a, int start, int end, type v)  6`

  采用二分搜索算法查找值 v。如果查找成功，则返回相应的下标值；否则，返回一个负数值 r。— r — 1 是为保持 a 有序 v 应插入的位置。

  参数：a　　类型为 int、long、short、char、byte、boolean、float 或 double 的有序数组。

  start　　起始下标（包含这个值）。

  end　　终止下标（不包含这个值）。

  v　　同 a 的数据元素类型相同的值。

- `static void fill(type[] a, type v)`

  将数组的所有数据元素值设置为 v。

参数：a　　类型为 int、long、short、char、byte、boolean、float 或 double 的数组。

v　　与 a 数据元素类型相同的一个值。

- `static boolean equals(type[] a, type[] b)`

如果两个数组大小相同，并且下标相同的元素都对应相等，返回 true。

参数：a、b　　类型为 int、long、short、char、byte、boolean、float 或 double 的两个数组。

3.10.6 多维数组

多维数组将使用多个下标访问数组元素，它适用于表示表格或更加复杂的排列形式。这一节的内容可以先跳过，等到需要使用这种存储机制时再返回来学习。

假设需要建立一个数值表，用来显示在不同利率下投资 $10,000 会增长多少，利息每年兑现，而且又被用于投资（见表 3-8）。

表 3-8 不同利率下的投资增长情况

10%	11%	12%	13%	14%	15%
10 000.00	10 000.00	10 000.00	10 000.00	10 000.00	10 000.00
11 000.00	11 100.00	11 200.00	11 300.00	11 400.00	11 500.00
12 100.00	12 321.00	12 544.00	12 769.00	12 996.00	13 225.00
13 310.00	13 676.31	14 049.28	14 428.97	14 815.44	15 208.75
14 641 00	15 180.70	15 735.19	16 304.74	16 889.60	17 490.06
16 105.10	16 850.58	17 623.42	18 424 .35	19 254.15	20 113.57
17 715.61	18 704.15	19 738.23	20 819.52	21 949.73	23 130.61
19 487.17	20 761.60	22 106.81	23 526.05	25 022.69	26 600.20
21 435.89	23 045.38	24 759.63	26 584.44	28 525.86	30 590.23
23 579.48	25 580.37	27 730.79	30 040.42	32 519.49	35 178.76

可以使用一个二维数组（也称为矩阵）存储这些信息。这个数组被命名为 balance。

在 Java 中，声明一个二维数组相当简单。例如：

```
double[][] balances;
```

与一维数组一样，在调用 new 对多维数组进行初始化之前不能使用它。在这里可以这样初始化：

```
balances=new double[NYEARS][NRATES];
```

另外，如果知道数组元素，就可以不调用 new，而直接使用简化的书写形式对多维数组进行初始化。例如：

```
int[][] magicSquare =
   {
      {16, 3, 2, 13},
      {5, 10, 11, 8},
      {9, 6, 7, 12},
      {4, 15, 14, 1}
   };
```

一旦数组被初始化，就可以利用两个方括号访问每个元素，例如，balances[i][j]。

在示例程序中用到了一个存储利率的一维数组 interest 与一个存储余额的二维数组 balances。一维用于表示年，另一维用于表示利率，最初使用初始余额来初始化这个数组的第一行：

```
for (int j = 0; j < balances[0].length; j++)
   balances[0][j] = 10000;
```

然后，按照下列方式计算其他行：

```
for (int i = 1; i < balances.length; i++)
{
   for (int j = 0; j < balances[i].length; j++)
   {
      double oldBalance = balances[i - 1][j];
      double interest = . . .;
      balances[i][j] = oldBalance + interest;
   }
}
```

程序清单 3-8 给出了完整的程序。

注释：for each 循环语句不能自动处理二维数组的每一个元素。它是按照行，也就是一维数组处理的。要想访问二维数组 a 的所有元素，需要使用两个嵌套的循环，如下所示：

```
for (double[] row : a)
   for (double value : row)
      do something with value
```

提示：要想快速地打印一个二维数组的数据元素列表，可以调用：

```
System.out.println(Arrays.deepToString(a));
```

输出格式为：

```
[[16, 3, 2, 13], [5, 10, 11, 8], [9, 6, 7, 12], [4, 15, 14, 1]]
```

程序清单 3-8 CompoundInterest/CompoundInterest.java

```
1  /**
2   * This program shows how to store tabular data in a 2D array.
3   * @version 1.40 2004-02-10
4   * @author Cay Horstmann
5   */
6  public class CompoundInterest
7  {
8     public static void main(String[] args)
9     {
10       final double STARTRATE = 10;
11       final int NRATES = 6;
12       final int NYEARS = 10;
13
14       // set interest rates to 10 . . . 15%
15       double[] interestRate = new double[NRATES];
16       for (int j = 0; j < interestRate.length; j++)
17          interestRate[j] = (STARTRATE + j) / 100.0;
```

```
18
19       double[][] balances = new double[NYEARS][NRATES];
20
21       // set initial balances to 10000
22       for (int j = 0; j < balances[0].length; j++)
23          balances[0][j] = 10000;
24
25       // compute interest for future years
26       for (int i = 1; i < balances.length; i++)
27       {
28          for (int j = 0; j < balances[i].length; j++)
29          {
30             // get last year's balances from previous row
31             double oldBalance = balances[i - 1][j];
32
33             // compute interest
34             double interest = oldBalance * interestRate[j];
35
36             // compute this year's balances
37             balances[i][j] = oldBalance + interest;
38          }
39       }
40
41       // print one row of interest rates
42       for (int j = 0; j < interestRate.length; j++)
43          System.out.printf("%9.0f%%", 100 * interestRate[j]);
44
45       System.out.println();
46       // print balance table
47       for (double[] row : balances)
48       {
49          // print table row
50          for (double b : row)
51             System.out.printf("%10.2f", b);
52
53          System.out.println();
54       }
55    }
56 }
```

3.10.7 不规则数组

到目前为止，读者所看到的数组与其他程序设计语言中提供的数组没有多大区别。但实际存在着一些细微的差异，而这正是 Java 的优势所在：Java 实际上没有多维数组，只有一维数组。多维数组被解释为“数组的数组。”

例如，在前面的示例中，balances 数组实际上是一个包含 10 个元素的数组，而每个元素又是一个由 6 个浮点数组成的数组（请参看图 3-15 ）。

表达式 balances[i] 引用第 i 个子数组，也就是二维表的第 i 行。它本身也是一个数组，balances[i][j] 引用这个数组的第 j 项。

由于可以单独地存取数组的某一行，所以可以让两行交换。

```
double[] temp = balances[i];
balances[i] = balances[i + 1];
balances[i + 1] = temp;
```

还可以方便地构造一个“不规则”数组，即数组的每一行有不同的长度。下面是一个典型的示例。在这个示例中，创建一个数组，第 i 行第 j 列将存放“从 i 个数值中抽取 j 个数值”产生的结果。

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5 10 10  5 1
1  6 15 20 15 6 1
```

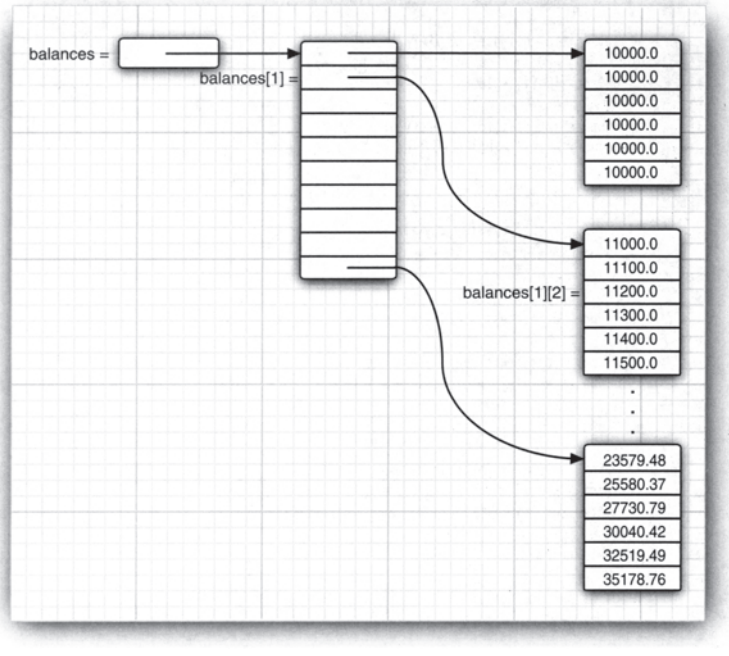

图 3-15 一个二维数组

由于 j 不可能大于 i，所以矩阵是三角形的。第 i 行有 i + 1 个元素（允许抽取 0 个元素，也是一种选择）。要想创建一个不规则的数组，首先需要分配一个具有所含行数的数组。

```
int[][] odds = new int[NMAX + 1][];
```

接下来，分配这些行。

```
for (int n = 0; n <= NMAX; n++)
   odds[n] = new int[n + 1];
```

在分配了数组之后，假定没有超出边界，就可以采用通常的方式访问其中的元素了。

```
for (int n = 0; n < odds.length; n++)
   for (int k = 0; k < odds[n].length; k++)
   {
      // compute lotteryOdds
      . . .
      odds[n][k] = lotteryOdds;
   }
```

程序清单 3-9 给出了完整的程序。

C++ 注释： 在 C++ 中，Java 的声明

```
double[][] balances = new double[10][6]; // Java
```

不同于

```
double balances[10][6]; // C++
```

也不同于

```
double (*balances)[6] = new double[10][6]; // C++
```

而是分配了一个包含 10 个指针的数组：

```
double** balances = new double*[10]; // C++
```

然后，指针数组的每一个元素被填充了一个包含 6 个数字的数组：

```
for (i = 0; i < 10; i++)
   balances[i] = new double[6];
```

庆幸的是，当创建 new double[10][6] 时，这个循环将自动地执行。当需要不规则的数组时，只能单独地创建行数组。

程序清单 3-9 LotteryArray/LotteryArray.java

```
/**
 * This program demonstrates a triangular array.
 * @version 1.20 2004-02-10
 * @author Cay Horstmann
 */
public class LotteryArray
{
   public static void main(String[] args)
   {
      final int NMAX = 10;

      // allocate triangular array

      int[][] odds = new int[NMAX + 1][];
      for (int n = 0; n <= NMAX; n++)
         odds[n] = new int[n + 1];

      // fill triangular array

      for (int n = 0; n < odds.length; n++)
         for (int k = 0; k < odds[n].length; k++)
         {
            /*
```

```
24              * compute binomial coefficient n*(n-1)*(n-2)*...*(n-k+1)/(1*2*3*...*k)
25              */
26             int lotteryOdds = 1;
27             for (int i = 1; i <= k; i++)
28                lotteryOdds = lotteryOdds * (n - i + 1) / i;
29
30             odds[n][k] = lotteryOdds;
31          }
32
33       // print triangular array
34       for (int[] row : odds)
35       {
36          for (int odd : row)
37             System.out.printf("%4d", odd);
38          System.out.println();
39       }
40    }
41 }
```

现在，已经看到了Java语言的基本程序结构，下一章节将介绍Java中的面向对象的程序设计。