PEARSON

JAVA™ 核心技术

Core Java Volume Ⅱ — Advanced Features, Ninth Edition

卷Ⅱ：高级特性 上

（第9版·英文版）

[美] Cay S. Horstmann　Gary Cornell 著

人民邮电出版社
POSTS & TELECOM PRESS

JAVA 核心技术

Core Java Volume Ⅱ — Advanced Features, Ninth Edition

卷Ⅱ：高级特性 上

（第9版·英文版）

[美] Cay S. Horstmann　Gary Cornell 著

人 民 邮 电 出 版 社

北 京

图书在版编目（CIP）数据

Java核心技术 = Core java : 第9版. 第2卷, 高级特性 : 英文 / (美) 霍斯特曼 (Horstmann, C. S) , (美) 康奈尔 (Cornell, G.) 著. -- 2版. -- 北京 : 人民邮电出版社, 2015.5
ISBN 978-7-115-38038-8

Ⅰ. ①J… Ⅱ. ①霍… ②康… Ⅲ. ①JAVA语言－程序设计－英文 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2014)第300548号

内 容 提 要

本书是经典的《Java 核心技术 卷 II：高级特性》的最新版，针对 Java SE 7 平台进行了全面更新，涵盖了 Java 语言的高级特性，并兼顾 Java SE 7 引入的高级用户界面程序设计和企业特性。

本书是 Java 技术权威指南，全面覆盖 Java 技术的高级主题，包括输入输出流、XML、网络 API、数据库编程、高级 Swing、Java 2D API、JavaBeans 组件、安全、脚本、编译和注解处理、分布式对象等，同时涉及本地化、国际化以及 Java SE 7 的内容。本书对 Java 技术的阐述精确到位，叙述方式深入浅出，并包含大量程序示例，从而帮助读者充分理解 Java 语言以及 Java 类库的相关高级特性。

本书适合想将 Java 应用于实际项目的软件开发人员、高等院校教师和学生参考阅读。

◆ 著 [美] Cay S. Horstmann Gary Cornell
责任编辑 杨海玲
责任印制 张佳莹 焦志炜
◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷
◆ 开本：800×1000 1/16
印张：71.5
字数：1370 千字 2015 年 5 月第 2 版
印数：2501－4 500 册 2015 年 5 月北京第 1 次印刷
著作权合同登记号 图字：01-2013-3135 号

定价：119.00 元（上、下册）

读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315

版权声明

前言[①]

致读者

现在你手中的这本书是《Java 核心技术》一书第 9 版的卷 II，其内容已经按照 Java SE 7 进行了全面更新。卷 I 介绍的是 Java 语言的基本特性，本卷介绍的是程序员进行专业软件开发时所需了解的更高级的主题。因此，与卷 I 以及本书之前的版本一样，本书的读者定位仍是那些希望将 Java 技术应用于实际项目的程序员。

请注意，如果你是经验丰富的程序员，能够灵活运用内部类和泛型等高级语言特性，那么你并不需要通过阅读卷 I 来领会卷 II。虽然本书会看情况适当地参考引用卷 I 的部分章节（当然，我们希望你能够购买或者已经购买了卷 I），但是你也可以在任何一本综合介绍 Java 平台的书中找到所需的背景资料。

最后需要说明的是，编写任何书籍都难免会有错误和不准确的地方。如果你在本书中发现了错误和不准确的地方，非常希望你能通知我们。当然，我们希望每个问题只被提交一次。我们为此特意创建了一个网站 http://horstmann.com/corejava，用于支持 FAQ、更正错误和解决方案。在错误提交页面上，我们希望你先阅读已经提交的错误，因此我们把错误提交表格放在了页面末尾，你可以使用该表格来提交错误或问题，并为改进本书将来的版本提供建议。

本书内容

本书的大部分章都是相互独立的。你可以钻研任何最感兴趣的主题，以任意顺序阅读这些章。

第 1 章的主题是输入输出（I/O）处理。Java 中所有的 I/O 都是通过所谓的“流”（stream）来处理的。我们可以利用流按照统一的方式来处理各种数据源之间的通信，如文件、网络连接或内存块。这一章会详细介绍读者（reader）和写者（writer）类，它们使处理 Unicode 变得更加容易。

① 本前言由臧秀涛翻译。

对象序列化机制使对象的保存和加载变得既方便又简单，我们将会介绍其背后的原理。这一章还介绍正则表达式库和 Java SE 7 的 NIO2 库，其中 NIO2 库使很多常用操作（例如，读取一个文件中所有的行）变得非常方便。

第 2 章介绍 XML。这一章会向读者展示如何解析 XML 文件，如何生成 XML，以及如何使用 XSL 转换。作为一个有用的例子，我们将展示如何在 XML 中指定 Swing 窗体的布局。这一章还讨论 XPath API，在 XML 中找东西就像海底捞针一般，而利用 XPath 就容易多了。

第 3 章介绍网络 API。Java 使复杂的网络编程变得非常容易。这一章将向读者展示如何创建连接到服务器的网络连接，如何实现自己的服务器，以及如何创建 HTTP 连接。

第 4 章介绍数据库编程。这一章主要关注 JDBC（Java 数据库连接）API，它们用于将 Java 程序连接到关系数据库。这一章将展示如何使用 JDBC API 的一个核心子集，并编写实用的程序来处理现实中琐碎的数据库问题（要完整地介绍 JDBC API，还需要这么厚的一本书才行）。最后，这一章简要地介绍一下分层数据库，并探讨 JNDI（Java 命名和目录接口）和 LDAP（轻量级目录访问协议）。

第 5 章探讨的是国际化，我们认为该特性将会越来越重要。Java 是一开始设计时就支持 Unicode 处理的少数编程语言之一，不过 Java 平台的国际化支持更具有前瞻性。因此，你可以对 Java 应用程序进行国际化，使其不仅能够跨平台，还能够跨越国界。作为例子，我们会展示如何编写一个支持英语、德语和汉语的退休金计算器程序。

第 6 章包含的是所有没有纳入卷 I 的 Swing 知识，尤其是重要但很复杂的树（tree）和表（table）等组件。这一章会展示编辑器面板的基本用法，“多文档”界面的 Java 实现，用在多线程程序中的进度指示器，以及诸如闪现屏幕和支持系统托盘等的“桌面集成功能”。再次强调一下，我们主要讲解的是读者在实际编程中很可能会遇到的最有用的构件，因为全面介绍整个 Swing 类库可能会需要几卷书的篇幅，并且只有专业人士才会对此感兴趣。

第 7 章介绍 Java 2D API，可以使用它们来创建实际图形和特殊效果。这一章还会介绍抽象窗口工具包（AWT）的一些高级特性，这些特性放入卷 I 中可能会显得过于高级，尽管如此，它们还是应该成为每个程序员工具箱中的一部分。这些特性包括打印和用于剪贴和拖放的 API。

第 8 章解释的是读者需要了解的 Java 平台的组件 API——JavaBean。这

一章会展示如何编写自己的 Bean，其他程序员可以在集成构建环境中操作这些 Bean。最后，我们将展示如何使用 JavaBean 的持久化用适于长期存储（这与对象序列化不同）的格式保存数据。

第 9 章讨论 Java 的安全模型。Java 在一开始设计时就考虑了安全性，那这种设计是如何实现的呢？本章将带你深入其中。对于有特殊用途的应用，我们会为你展示如何编写自己的类加载器和安全管理器。然后，我们介绍安全 API，它支持消息与代码的签名、授权和认证以及加密等重要功能。最后，这一章会提供几个使用了 AES 和 RSA 加密算法的例子。

第 10 章介绍分布式对象。这一章会详细地介绍 RMI（远程方法调用），可以利用该 API 来使用分布在多台机器上的 Java 对象。

第 11 章讨论三种处理代码的技术。借助脚本 API 和编译器 API，程序可以调用诸如 JavaScript 或 Groovy 等脚本语言编写的代码，也可以编译 Java 代码。我们可以利用注解（annotation）将任意信息（有时称为元信息）添加到 Java 程序中。这一章会向读者展示注解处理器如何在源码级或类文件级收集这些注解，以及如何使用注解来影响运行时类的行为。注解只有在工具的支持下才有用，希望本章的讨论对你有所帮助，使你能够根据需要选择有用的注解处理工具。

第 12 章介绍本地方法，可以利用本地方法调用为特定机器编写的方法，如 Microsoft Windows API。很明显，这个特性是有争议的：一旦使用了本地方法，Java 的跨平台特性就会消失。尽管如此，为特定平台编写 Java 应用程序时，每个严谨的程序员都需要了解这些技术。用户有时需要与 Java 并不支持的设备或服务进行交互，这就要求助于目标平台的操作系统 API 了。为了说明这一点，这一章会向读者展示在一个 Java 程序中如何访问 Windows 注册表 API。

一如既往，我们按照 Java 的最新版本完整地修订了所有章节，删除了过时的资料，并且详细介绍了 Java SE 7 的新 API。

约定

我们使用等宽字体表示计算机代码，这在很多计算机书籍中都很常见。

注意（NOTE）：注意会使用这样的“注意”图标进行标记。

提示（TIP）：提示会使用这样的“提示”图标进行标记。

警告（CAUTION）：如果存在危险，我们会使用“警告”图标警告你。

C++注意（C++NOTE）：用于解释 Java 和 C++之间的不同。如果对 C++不感兴趣，可以跳过这些内容。

Java 提供了一个大型的编程库，即应用编程接口（API）。在第一次使用某个 API 时，我们会在该节末尾添加一个简短的描述，这些描述不那么正式，但希望它们能比官方的联机 API 文档更具指导性。接口名用斜体表示。类名、接口名或方法名后的数字表示引入该特性的 JDK 版本。

`Application Programming Interface 1.2`

程序的源代码以代码清单的形式列出。例如：

Listing 1.1 `InputTest/InputTest.java`

本书的配套代码可以从 http://horstmann.com/corejava 下载。

Acknowledgments

Writing a book is always a monumental effort, and rewriting doesn't seem to be much easier, especially with such a rapid rate of change in Java technology. Making a book a reality takes many dedicated people, and it is my great pleasure to acknowledge the contributions of the entire Core Java team.

A large number of individuals at Prentice Hall provided valuable assistance, but they managed to stay behind the scenes. I'd like them all to know how much I appreciate their efforts. As always, my warm thanks go to my editor, Greg Doench, for steering the book through the writing and production process, and for allowing me to be blissfully unaware of the existence of all those folks behind the scenes. I am very grateful to Julie Nahil for production support, and to Dmitry Kirsanov and Alina Kirsanova for copyediting and typesetting the manuscript.

Thanks to the many readers of earlier editions who reported embarrassing errors and made lots of thoughtful suggestions for improvement. I am particularly grateful to the excellent reviewing team that went over the manuscript with an amazing eye for detail and saved me from many more embarrassing errors.

Reviewers of this and earlier editions include Chuck Allison (Contributing Editor, C/C++ Users Journal), Lance Anderson (Oracle), Alec Beaton (PointBase, Inc.), Cliff Berg (iSavvix Corporation), Joshua Bloch, David Brown, Corky Cartwright, Frank Cohen (PushToTest), Chris Crane (devXsolution), Dr. Nicholas J. De Lillo (Manhattan College), Rakesh Dhoopar (Oracle), Robert Evans (Senior Staff, The Johns Hopkins University Applied Physics Lab), David Geary (Sabreware), Jim Gish (Oracle), Brian Goetz (Principal Consultant, Quiotix Corp.), Angela Gordon, Dan Gordon, Rob Gordon, John Gray (University of Hartford), Cameron Gregory (olabs.com), Marty Hall (The Johns Hopkins University Applied Physics Lab), Vincent Hardy, Dan Harkey (San Jose State University), William Higgins (IBM), Vladimir Ivanovic (PointBase), Jerry Jackson (ChannelPoint Software), Tim Kimmet (Preview Systems), Chris Laffra, Charlie Lai, Angelika Langer, Doug Langston, Hang Lau (McGill University), Mark Lawrence, Doug Lea (SUNY Oswego), Gregory Longshore, Bob Lynch (Lynch Associates), Philip Milne (consultant), Mark Morrissey (The Oregon Graduate Institute), Mahesh Neelakanta (Florida Atlantic University), Hao Pham, Paul Philion, Blake Ragsdell, Ylber Ramadani (Ryerson University), Stuart Reges (University of Arizona), Rich Rosen (Interactive Data Corporation), Peter Sanders

(ESSI University, Nice, France), Dr. Paul Sanghera (San Jose State University and Brooks College), Paul Sevinc (Teamup AG), Devang Shah, Richard Slywczak (NASA/Glenn Research Center), Bradley A. Smith, Steven Stelting, Christopher Taylor, Luke Taylor (Valtech), George Thiruvathukal, Kim Topley (author of *Core JFC, Second Edition*), Janet Traub, Paul Tyma (consultant), Peter van der Linden, Burt Walsh, Joe Wang (Oracle), and Dan Xu (Oracle).

Cay Horstmann
San Francisco, California
December 2012

Contents

CHAPTER 1

Streams and Files

In this chapter:

In this chapter, we will cover the Java Application Programming Interfaces (APIs) for input and output. You will learn how to access files and directories and how to read and write data in binary and text format. This chapter also shows you the object serialization mechanism that lets you store objects as easily as you can store text or numeric data. Next, we will turn to several improvements that were made in the "new I/O" package `java.nio`, introduced in Java SE 1.4, and the "new new I/O" enhancements of Java 7. We finish the chapter with a discussion of regular expressions, even though they are not actually related to streams and files. We couldn't find a better place to handle that topic, and apparently neither could the Java team—the regular expression API specification was attached to the specification request for the "new I/O" features.

1.1 Streams

In the Java API, an object from which we can read a sequence of bytes is called an *input stream*. An object to which we can write a sequence of bytes is called an *output stream*. These sources and destinations of byte sequences can be—and often are—files, but they can also be network connections and even blocks of memory. The abstract classes `InputStream` and `OutputStream` form the basis for a hierarchy of input/output (I/O) classes.

Byte-oriented streams are inconvenient for processing information stored in Unicode (recall that Unicode uses multiple bytes per character). Therefore, a separate hierarchy provides classes for processing Unicode characters that inherit from the abstract `Reader` and `Writer` classes. These classes have read and write operations that are based on two-byte Unicode code units rather than on single-byte characters.

1.1.1 Reading and Writing Bytes

The `InputStream` class has an abstract method:

```
abstract int read()
```

This method reads one byte and returns the byte that was read, or `-1` if it encounters the end of the input source. The designer of a concrete input stream class overrides this method to provide useful functionality. For example, in the `FileInputStream` class, this method reads one byte from a file. `System.in` is a predefined object of a subclass of `InputStream` that allows you to read information from the keyboard.

The `InputStream` class also has nonabstract methods to read an array of bytes or to skip a number of bytes. These methods call the abstract `read` method, so subclasses need to override only one method.

Similarly, the `OutputStream` class defines the abstract method

```
abstract void write(int b)
```

which writes one byte to an output location.

Both the `read` and `write` methods *block* until the byte is actually read or written. This means that if the stream cannot immediately be accessed (usually because of a busy network connection), the current thread blocks. This gives other threads the chance to do useful work while the method is waiting for the stream to become available again.

The `available` method lets you check the number of bytes that are currently available for reading. This means a fragment like the following is unlikely to block: