HZ BOOKS 华章IT

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 8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

Java 核心技术系列

Pearson

Java 核心技术 卷II

高级特性（原书第10版）

Core Java Volume II—Advanced Features
(10th Edition)

[美] 凯 S. 霍斯特曼（Cay S. Horstmann）著
陈昊鹏 译

机械工业出版社
China Machine Press

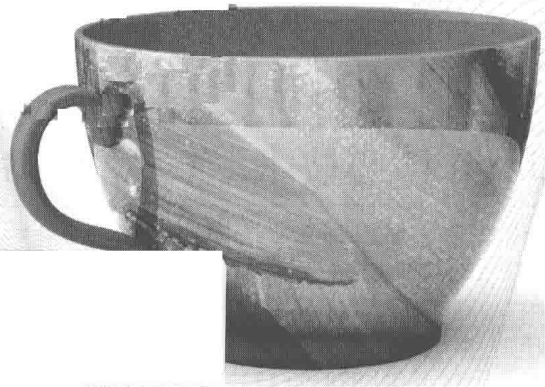
Java 核心技术系列

Java 核心技术 卷II

高级特性（原书第10版）

Core Java Volume II—Advanced Features
(10th Edition)

[美] 凯 S. 霍斯特曼（Cay S. Horstmann）著
陈昊鹏 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Ⅱ　高级特性（原书第 10 版）/（美）凯 S. 霍斯特曼（Cay S. Horstmann）著；陈昊鹏译 . —北京：机械工业出版社，2017.6
（Java 核心技术系列）
书名原文：Core Java Volume II—Advanced Features (10th Edition)

ISBN 978-7-111-57331-9

I. J…　II. ①凯…　②陈…　III. JAVA 语言 – 程序设计　IV. TP312.8

中国版本图书馆 CIP 数据核字（2017）第 142782 号

本书版权登记号：图字：01-2017-1400

Java 核心技术　卷Ⅱ　高级特性（原书第 10 版）

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：关　敏　　责任校对：殷　虹
印　　刷：北京文昌阁彩色印刷有限责任公司　　版　　次：2017 年 9 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：51
书　　号：ISBN 978-7-111-57331-9　　定　　价：139.00 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）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

译 者 序

《Java 核心技术 卷Ⅱ 高级特性（原书第 10 版）》中文版又要呈现在广大读者的面前了！这是我翻译的本书的第 4 个版本，细心一看，才发现距离最早的第 7 版已经过去了将近 12 年，岁月神偷悄然改变着我的音容相貌，也让 Java 语言不断地完善演化，发生了脱胎换骨般的变化。

随着 Java 语言的更新，本书的内容也进行了大幅度的调整，新增了 Java SE 8 中的流库，以及日期和时间 API 的内容，调整掉了 JavaBean 和 RMI 等内容，使得本书的内容既反映了 Java 语言的新变化，又显得更加紧凑，达到了与时俱进的目的。

本书实际上并不适合 Java 初学者，它更适合有一定 Java 编程基础的程序员，因为具备一定的基础知识才能更好地理解本书的内容，这也是卷Ⅱ被称为“高级特性”的原因。通过阅读本书，你会了解到高级特性的细节，它们涉及复杂系统的各个方面，是开发更好、更快、更安全和更易维护的系统所必不可少的语言特性。

在这一版的翻译工作中，我对原文没有变化的部分也进行了仔细修订，尽量修改了其中的错误和翻译不通顺的语句。令人汗颜的是，虽然已经修订过 3 版了，在这一版中还是发现了不少错误，在此我向所有之前版本的读者道歉，也恳请读者对这一版中的谬误提出批评。

最后，祝大家通过阅读本书不但能够提升 Java 编程能力，更能够加深对 Java 编程语言的理解和认识。让我们共同学习，永远在路上！

陈昊鹏

前　言

致读者

本书是按照 Java SE 8 完全更新后的《Java 核心技术　卷Ⅱ　高级特性（原书第 10 版）》。卷Ⅰ主要介绍了 Java 语言的一些关键特性；而本卷主要介绍编程人员进行专业软件开发时需要了解的高级主题。因此，与本书卷Ⅰ和之前的版本一样，我们仍将本书定位于用 Java 技术进行实际项目开发的编程人员。

编写任何一本书籍都难免会有一些错误或不准确的地方。我们非常乐意听到读者的意见。当然，我们更希望对本书问题的报告只听到一次。为此，我们创建了一个 FAQ、bug 修正以及应急方案的网站 http:// horstmann.com/corejava。你可以在 bug 报告网页（该网页的目的是鼓励读者阅读以前的报告）的末尾处添加 bug 报告，以此来发布 bug 和问题并给出建议，以便我们改进本书将来版本的质量。

内容提要

本书中的章节大部分是相互独立的。你可以研究自己最感兴趣的主题，并可以按照任意顺序阅读这些章节。

在**第 1 章**中，你将学习 Java 8 的流库，它带来了现代风格的数据处理机制，即只需指定想要的结果，而无须详细描述应该如何获得该结果。这使得流库可以专注于优化的计算策略，对于优化并发计算来说，这显得特别有利。

第 2 章的主题是输入输出处理。在 Java 中，所有 I/O 都是通过输入 / 输出流来处理的。这些流（不要与第 1 章的那些流混淆了）使你可以按照统一的方式来处理与各种数据源之间的通信，例如文件、网络连接或内存块。我们对各种读入器和写出器类进行了详细的讨论，它们使得对 Unicode 的处理变得很容易。我们还展示了如何使用对象序列化机制从而使保存和加载对象变得容易而方便，及其背后的原理。然后，我们讨论了正则表达式和操作文件与路径。

第 3 章介绍 XML，介绍怎样解析 XML 文件，怎样生成 XML 以及怎样使用 XSL 转换。在一个实用示例中，我们将展示怎样在 XML 中指定 Swing 窗体的布局。我们还讨论了 XPath API，它使得“在 XML 的干草堆中寻找绣花针”变得更加容易。

第 4 章介绍网络 API。Java 使复杂的网络编程工作变得很容易实现。我们将介绍怎样创建连接到服务器上，怎样实现你自己的服务器，以及怎样创建 HTTP 连接。

第 5 章介绍数据库编程，重点讲解 JDBC，即 Java 数据库连接 API，这是用于将 Java 程

序与关系数据库进行连接的 API。我们将介绍怎样通过使用 JDBC API 的核心子集，编写能够处理实际的数据库日常操作事务的实用程序。（如果要完整介绍 JDBC API 的功能，可能需要编写一本像本书一样厚的书才行。）最后我们简要介绍了层次数据库，探讨了一下 JNDI（Java 命名及目录接口）以及 LDAP（轻量级目录访问协议）。

Java 对于处理日期和时间的类库做出过两次设计，而在 Java 8 中做出的第三次设计则极富魅力。在**第 6 章**，你将学习如何使用新的日期和时间库来处理日历和时区的复杂性。

第 7 章讨论了一个我们认为其重要性将会不断提升的特性——国际化。Java 编程语言是少数几种一开始就被设计为可以处理 Unicode 的语言之一，不过 Java 平台的国际化支持则走得更加深远。因此，你可以对 Java 应用程序进行国际化，使得它们不仅可以跨平台，而且还可以跨越国界。例如，我们会展示怎样编写一个使用英语、德语和汉语的退休金计算器。

第 8 章讨论了三种处理代码的技术。脚本机制和编译器 API 允许程序去调用使用诸如 JavaScript 或 Groovy 之类的脚本语言编写的代码，并且允许程序去编译 Java 代码。可以使用注解向 Java 程序中添加任意信息（有时称为元数据）。我们将展示注解处理器怎样在源码级别或者在类文件级别上收集这些注解，以及怎样运用这些注解来影响运行时的类行为。注解只有在工具的支持下才有用，因此，我们希望我们的讨论能够帮助你根据需要选择有用的注解处理工具。

第 9 章继续介绍 Java 安全模型。Java 平台一开始就是基于安全而设计的，该章会带你深入内部，查看这种设计是怎样实现的。我们将展示怎样编写用于特殊应用的类加载器以及安全管理器。然后介绍允许使用消息、代码签名、授权以及认证和加密等重要特性的安全 API。最后，我们用一个使用 AES 和 RSA 加密算法的示例进行了总结。

第 10 章涵盖了没有纳入卷 I 的所有 Swing 知识，尤其是重要但很复杂的树形构件和表格构件。随后我们介绍了编辑面板的基本用法、“多文档”界面的 Java 实现、在多线程程序中用到的进度指示器，以及诸如闪屏和支持系统托盘这样的“桌面集成特性”。我们仍着重介绍在实际编程中可能遇到的最为有用的构件，因为对 Swing 类库进行百科全书般的介绍可能会占据好几卷书的篇幅，并且只有专门的分类学家才感兴趣。

第 11 章介绍 Java 2D API，你可以用它来创建实际的图形和特殊的效果。该章还介绍了抽象窗口操作工具包（AWT）的一些高级特性，这部分内容看起来过于专业，不适合在卷 I 中介绍。虽然如此，这些技术还是应该成为每一个编程人员工具包的一部分。这些特性包括打印和用于剪切粘贴及拖放的 API。

第 12 章介绍本地方法，这个功能可以让你调用为微软 Windows API 这样的特殊机制而编写的各种方法。很显然，这种特性具有争议性：使用本地方法，那么 Java 平台的跨平台特性将会随之消失。虽然如此，每个为特定平台编写 Java 应用程序的专业开发人员都需要了解这些技术。有时，当你与不支持 Java 平台的设备或服务进行交互时，为了你的目标平台，你可能需要求助于操作系统 API。我们将通过展示如何从某个 Java 程序访问 Windows 注册表 API 来阐明这一点。

所有章节都按照最新版本的 Java 进行了修订，过时的材料都删除了，Java SE 8 的新 API

也都详细地进行了讨论。

约定

我们使用等宽字体表示计算机代码，这种格式在众多的计算机书籍中极为常见。各种图标的含义如下：

注意：需要引起注意的地方。

提示：有用的提示。

警告：关于缺陷或危险情况的警告信息。

C++ 注意：本书中有许多这类提示，用于解释 Java 程序设计语言和 C++ 语言之间的不同。如果你对这部分不感兴趣，可以跳过。

Java 平台配备有大量的编程类库或者应用编程接口（API）。当第一次使用某个 API 时，我们在每一节的末尾都添加了一个简短的描述。这些描述可能有点不太规范，但是比官方在线 API 文档更具指导性。接口的名字都是斜体的，就像许多官方文档一样。类、接口或方法名后面的数字是 JDK 的版本，表示在该版本中才引入了这些特性。

Application Programming Interface 1.2

本书示例代码以程序清单的形式列举了出来，例如：

程序清单 1-1　ScriptTest.java

可以从网站 http://horstmann.com/corejava ㊀下载示例代码。

致谢

写书总是需要付出极大的努力，而重写也并不像看上去那么容易，特别是在 Java 技术方面，要跟上其飞快的发展速度，更是如此。一本书的面世需要众多有奉献精神的人共同努力，我非常荣幸地在此向整个《Java 核心技术》团队致谢。

Prentice Hall 出版社的许多人都提供了颇有价值的帮助，但是他们甘愿居于幕后。我希望他们都能够知道我是多么感谢他们付出的努力。与以往一样，我要热切地感谢我的编辑，Prentice Hall 出版社的 Greg Doench，他对本书从编写到出版进行全程掌舵，并使我可以十分幸福地根本意识不到幕后那些人的存在。我还非常感谢 Julie Nahil 在撰写上的支持，以及感谢 Dmitry Kirsanov 和 Alina Kirsanova 对手稿的编辑和排版。

我非常感谢找到了很多令人尴尬的错误并提出了许多颇具创见性的建议的早先版本的读

㊀ 也可登录华章网站（http://www.hzbook.com）下载相关代码。——编辑注

者。我特别要感谢十分出色的评审团队，他们用令人惊异的眼睛仔细浏览了所有原稿，并将我从许多令人尴尬的错误中拯救了出来。

这一版及以前版本是由以下人员评审的：Chuck Allison（特约编辑，《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（曼哈顿学院）、Rakesh Dhoopar（Oracle）、Robert Evans（资深教师，约翰·霍普金斯大学应用物理实验室）、David Geary（Sabreware）、Jin Gish(oracle) Brian Goetz（Oracle）、Angela Gordon、Dan Gordon、Rob Gordon、John Gray（Hartford 大学）、Cameron Gregory（olabs.com）、Steve Haines、Marty Hall（约翰·霍普金斯大学应用物理实验室）、Vincent Hardy、Dan Harkey（圣何塞州立大学）、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 大学）、Mark Lawrence、Doug Lea（SUNY Oswego）、Gregory Longshore、Bob Lynch（Lynch Associates）、Philip Milne（顾问）、Mark Morrissey（俄勒冈研究生院）、Mahesh Neelakanta（佛罗里达大西洋大学）、Hao Pham、Paul Philion、Blake Ragsdell、Ylber Ramadani（Ryerson 大学）、Stuart Reges（亚利桑那大学）、Simon Ritter、Rich Rosen（Interactive Data Corporation）、Peter Sanders（ESSI 大学，Nice, France）、Dr. Paul Sanghera（圣何塞州立大学和布鲁克学院）、Paul Sevinc（Teamup AG）、Yoshiki Shabata、Devang Shah、Richard Slywczak（NASA/Glenn 研究中心）、Bradley A. Smith、Steven Stelting、Christopher Taylor、Luke Taylor（Valtech）、George Thiruvathukal、Kim Topley（《Core JFC，Second Edition》的作者）、Janet Traub、Paul Tyma（顾问）、Christian Ullenboom、Peter van der Linden、Burt Walsh、Joe Wang(Oracle) 和 Dan Xu(Oracle)。

Cay Horstmann

2016 年 9 月于加州旧金山

目　录

第 1 章　Java SE 8 的流库

- ▲ 从迭代到流的操作
- ▲ 流的创建
- ▲ filter、map 和 flatMap 方法
- ▲ 抽取子流和连接流
- ▲ 其他的流转换
- ▲ 简单约简
- ▲ Optional 类型
- ▲ 收集结果
- ▲ 收集到映射表中
- ▲ 群组和分区
- ▲ 下游收集器
- ▲ 约简操作
- ▲ 基本类型流
- ▲ 并行流

流提供了一种让我们可以在比集合更高的概念级别上指定计算的数据视图。通过使用流，我们可以说明想要完成什么任务，而不是说明如何去实现它。我们将操作的调度留给具体实现去解决。例如，假设我们想要计算某个属性的平均值，那么我们就可以指定数据源和该属性，然后，流库就可以对计算进行优化，例如，使用多线程来计算总和与个数，并将结果合并。

在本章中，你将会学习如何使用 Java 的流库，它是在 Java SE 8 中引入的，用来以“做什么而非怎么做”的方式处理集合。

1.1　从迭代到流的操作

在处理集合时，我们通常会迭代遍历它的元素，并在每个元素上执行某项操作。例如，假设我们想要对某本书中的所有长单词进行计数。首先，将所有单词放到一个列表中：

```
String contents = new String(Files.readAllBytes(
   Paths.get("alice.txt")), StandardCharsets.UTF_8); // Read file into string
List<String> words = Arrays.asList(contents.split("\\PL+"));
   // Split into words; nonletters are delimiters
```

现在，我们可以迭代它了：

```
long count = 0;
for (String w : words)
{
   if (w.length() > 12) count++;
}
```

在使用流时，相同的操作看起来像下面这样：

```
long count = words.stream()
   .filter(w -> w.length() > 12)
   .count();
```

流的版本比循环版本要更易于阅读，因为我们不必扫描整个代码去查找过滤和计数操作，方法名就可以直接告诉我们其代码意欲何为。而且，循环需要非常详细地指定操作的顺序，而流却能够以其想要的任何方式来调度这些操作，只要结果是正确的即可。

仅将 stream 修改为 parallelStream 就可以让流库以并行方式来执行过滤和计数。

```
long count = words.parallelStream()
   .filter(w -> w.length() > 12)
   .count();
```

流遵循了“做什么而非怎么做”的原则。在流的示例中，我们描述了需要做什么：获取长单词，并对它们计数。我们没有指定该操作应该以什么顺序或者在哪个线程中执行。相比之下，本节开头处的循环要确切地指定计算应该如何工作，因此也就丧失了进行优化的机会。

流表面上看起来和集合很类似，都可以让我们转换和获取数据。但是，它们之间存在着显著的差异：

1. 流并不存储其元素。这些元素可能存储在底层的集合中，或者是按需生成的。

2. 流的操作不会修改其数据源。例如，filter 方法不会从新的流中移除元素，而是会生成一个新的流，其中不包含被过滤掉的元素。

3. 流的操作是尽可能惰性执行的。这意味着直至需要其结果时，操作才会执行。例如，如果我们只想查找前 5 个长单词而不是所有长单词，那么 filter 方法就会在匹配到第 5 个单词后停止过滤。因此，我们甚至可以操作无限流。

让我们再来看看这个示例。stream 和 parallelStream 方法会产生一个用于 words 列表的 stream。filter 方法会返回另一个流，其中只包含长度大于 12 的单词。count 方法会将这个流化简为一个结果。

这个工作流是操作流时的典型流程。我们建立了一个包含三个阶段的操作管道：

1. 创建一个流。

2. 指定将初始流转换为其他流的中间操作，可能包含多个步骤。

3. 应用终止操作，从而产生结果。这个操作会强制执行之前的惰性操作。从此之后，这个流就再也不能用了。

在程序清单 1-1 中的示例中，流是用 stream 或 parallelStream 方法创建的。filter 方法对其进行转换，而 count 方法是终止操作。

程序清单 1-1 streams/CountLongWords.java

```
1  package streams;
2
3  import java.io.IOException;
4  import java.nio.charset.StandardCharsets;
5  import java.nio.file.Files;
6  import java.nio.file.Paths;
7  import java.util.Arrays;
8  import java.util.List;
9
```

```
10 public class CountLongWords
11 {
12    public static void main(String[] args) throws IOException
13    {
14       String contents = new String(Files.readAllBytes(
15             Paths.get("../gutenberg/alice30.txt")), StandardCharsets.UTF_8);
16       List<String> words = Arrays.asList(contents.split("\\PL+"));
17
18       long count = 0;
19       for (String w : words)
20       {
21          if (w.length() > 12) count++;
22       }
23       System.out.println(count);
24
25       count = words.stream().filter(w -> w.length() > 12).count();
26       System.out.println(count);
27
28       count = words.parallelStream().filter(w -> w.length() > 12).count();
29       System.out.println(count);
30    }
31 }
```

在下一节中，你将会看到如何创建流。后续的三个小节将处理流的转换。再后面的五个小节将讨论终止操作。

API　java.util.stream.Stream<T> 8

- `Stream<T> filter(Predicate<? super T> p)`
  产生一个流，其中包含当前流中满足 P 的所有元素。
- `long count()`
  产生当前流中元素的数量。这是一个终止操作。

API　java.util.Collection<E> 1.2

- `default Stream<E> stream()`
- `default Stream<E> parallelStream()`
  产生当前集合中所有元素的顺序流或并行流。

1.2　流的创建

你已经看到了可以用 `Collection` 接口的 `stream` 方法将任何集合转换为一个流。如果你有一个数组，那么可以使用静态的 `Stream.of` 方法。

```
Stream<String> words = Stream.of(contents.split("\\PL+"));
   // split returns a String[] array
```

`of` 方法具有可变长参数，因此我们可以构建具有任意数量引元的流：

```
Stream<String> song = Stream.of("gently", "down", "the", "stream");
```

使用 `Array.stream(array, from, to)` 可以从数组中位于 `from`（包括）和 `to`（不包括）的元素中创建一个流。

为了创建不包含任何元素的流，可以使用静态的 `Stream.empty` 方法：

```
Stream<String> silence = Stream.empty();
   // Generic type <String> is inferred; same as Stream.<String>empty()
```

`Stream` 接口有两个用于创建无限流的静态方法。`generate` 方法会接受一个不包含任何引元的函数（或者从技术上讲，是一个 Supplier<T> 接口的对象）。无论何时，只要需要一个流类型的值，该函数就会被调用以产生一个这样的值。我们可以像下面这样获得一个常量值的流：

```
Stream<String> echos = Stream.generate(() -> "Echo");
```

或者像下面这样获取一个随机数的流：

```
Stream<Double> randoms = Stream.generate(Math::random);
```

为了产生无限序列，例如 0 1 2 3 …，可以使用 `iterate` 方法。它会接受一个“种子”值，以及一个函数（从技术上讲，是一个 UnaryOperation<T>），并且会反复地将该函数应用到之前的结果上。例如，

```
Stream<BigInteger> integers
   = Stream.iterate(BigInteger.ZERO, n -> n.add(BigInteger.ONE));
```

该序列中的第一个元素是种子 `BigInteger.ZERO`，第二个元素是 `f(seed)`，即 1（作为大整数），下一个元素是 `f(f(seed))`，即 2，后续以此类推。

注意：Java API 中有大量方法都可以产生流。例如，`Pattern` 类有一个 `splitAsStream` 方法，它会按照某个正则表达式来分割一个 `CharSequence` 对象。可以使用下面的语句来将一个字符串分割为一个个的单词：

```
Stream<String> words = Pattern.compile("\\PL+").splitAsStream(contents);
```

静态的 `Files.lines` 方法会返回一个包含了文件中所有行的 `Stream`：

```
try (Stream<String> lines = Files.lines(path))
{
   Process lines
}
```

程序清单 1-2 中的示例程序展示了创建流的各种方式。

程序清单 1-2 streams/CreatingStreams.java

```
1 package streams;
2
3 import java.io.IOException;
4 import java.math.BigInteger;
5 import java.nio.charset.StandardCharsets;
6 import java.nio.file.Files;
```