Broadview
www.broadview.com.cn

PEARSON

写给大忙人看的 Java SE 8

[美] Cay S. Horstmann 著
张若飞 译

Java SE 8
for the Really Impatient

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

写给大忙人看的
Java SE 8

[美] Cay S. Horstmann 著
张若飞 译

Java SE 8
for the Really Impatient

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书向 Java 开发人员言简意赅地介绍了 Java 8 的许多新特性（以及 Java 7 中许多未被关注的特性），本书延续了《快学 Scala》“不废话”的风格。

本书共分为 9 章。第 1 章讲述了 lambda 表达式的全部语法；第 2 章给出了流的完整概述；第 3 章给出了使用 lambda 表达式设计库的有效技巧；第 4 章介绍了 JavaFX；第 5 章详细介绍了 Java 新增的日期/时间 API；第 6 章介绍了原子计数器、并发哈希映射、并行数组操作等特性中的改进；第 7 章介绍了如何在 Java 虚拟机上执行 JavaScript，以及如何与 Java 代码进行交互操作；第 8 章描述了 Java 8 中其他一些不起眼但很实用的特性；第 9 章则关注于 Java 7 中改进的异常处理，以及其他一些你可能会忽略掉的 API。

本书适合所有 Java 程序员、软件设计师、架构师及软件开发爱好者阅读。对于想要快速了解 Java SE 8 新特性的 Java 工程师来说，本书是一本不可多得的枕边读物。

Authorized translation from the English language edition, entitled JAVA SE 8 FOR THE REALLY IMPATIENT, 1E, 9780321927767 by CAY S. HORSTMANN, published by Pearson Education, Inc., publishing as Addison-Wesley Professional, Copyright © 2014 Pearson Education, Inc.

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

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD., and PUBLISHING HOUSE OF ELECTRONICS INDUSTRY Copyright © 2015.

本书简体中文版专有出版权由 Pearson Education 培生教育出版亚洲有限公司授予电子工业出版社。未经出版者预先书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书简体中文版贴有 Pearson Education 培生教育出版集团激光防伪标签，无标签者不得销售。

版权贸易合同登记号　图字：01-2014-4716

图书在版编目（CIP）数据

写给大忙人看的 Java SE 8 /（美）霍斯曼（Horstmann,C.S.）著；张若飞译. —北京：电子工业出版社，2015.1
书名原文：Java SE 8 for the really impatient
ISBN 978-7-121-22728-8
Ⅰ. ①写… Ⅱ. ①霍… ②张… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312
中国版本图书馆 CIP 数据核字(2014)第 254597 号

策划编辑：张春雨
责任编辑：徐津平
印　　刷：北京中新伟业印刷有限公司
装　　订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：787×980　1/16　印张：15　字数：275 千字
版　　次：2015 年 1 月第 1 版
印　　次：2015 年 1 月第 1 次印刷
定　　价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

致与我合作二十年的编辑——Greg Doench，
我由衷地敬佩他的耐心、热情与准确的判断力。

前　言

本书向 Java 开发人员言简意赅地介绍了 Java 8 的许多特性，以及 Java 7 中许多没有被关注的特性。

本书延续了《快学 Scala》（*Scala for the Impatient*）“不废话”的风格（我在那本书中第一次尝试了这种写作风格）。在那本书中，我希望快速带读者切入正题，避免烦琐枯燥地罗列各种技术规范之间的优劣。我对知识点进行了分类整理，将它们组织为一个个模块，方便你快速获取所需的内容。这种方式在 Scala 社区取得了巨大的成功，因此我再次在本书中使用了这种方式。

自从有了 Java 8，Java 语言和库就仿佛获得了新生。lambda 表达式可以允许开发人员编写简洁的“计算片段”，并将它们传递给其他的代码。接收代码可以选择在合适的时候来执行“计算片段”。这对于构建第三方库有着深远的影响。

尤其要指出一点，它彻底改变了集合的使用方式。我们不需要再指定计算结果的过程（从起始遍历到结尾，如果某个元素满足某个条件，就根据它计算一个值，然后将值添加到总和中），只需指定想要什么样的结果（给我所有满足条件的元素的总和）。这样，代码就可以重新对计算排序——例如，来充分利用并行的优势。或者，如果你只希望匹配前 100 个元素，那么你不必再维护一个计数器，程序可以自动停止计算。

Java 8 中全新的 Stream API 实现了这一想法。第 1 章介绍了 lambda 表达式的全部语法，第 2 章给出了流的完整概述，第 3 章则给出了一些如何用 lambda 表达式设计库的有效技巧。

有了 Java 8 之后，由于 Swing 现在已经处于“维护”阶段，客户端应用程序的开发

试读结束：需要全本请在线购买：　www.ertongbook.com

人员需要转到 JavaFX API。第 4 章会向需要进行图形化编程的开发人员介绍 JavaFX——这在一图胜过万语千言时尤为有效。

在经过多年的等待之后，开发人员终于能够用上设计良好的日期/时间库了。第 5 章会详细介绍 `java.time` API。

Java 的每个版本都会对并发 API 进行增强，Java 8 也不例外。在第 6 章中，你将会了解原子计数器、并发哈希映射、并行数组操作及可完成的 Future 等方面的改进。

Java 8 内置了一个高质量的 JavaScript 引擎——Nashorn。在第 7 章中，你将会看到如何在 Java 虚拟机上执行 JavaScript，以及如何与 Java 代码进行交互操作。

第 8 章集中介绍了 Java 8 中其他一些不起眼但很实用的特性。同样，第 9 章关注于 Java 7 中改进的异常处理、处理文件和目录的“新 I/O” API，以及其他一些你可能会忽略的 API。

我要一如既往地感谢我的编辑 Greg Doench，他想出了这个点子，即为有经验的开发人员编写一本言简意赅的书，帮助他们跟上 Java 8 的步伐。Dmitry Kirsanov 和 Alina Kirsanova 又一次以令人惊讶的速度和对细节的关注，将 XHTML 手稿转化为了这本引人注目的书籍。由衷地感谢本书的审校人员，他们指出了许多低级的错误，并给出了精彩的改进建议。这些审校人员是：Gail Anderson、Paul Anderson、James Denvir、Trisha Gee、Brian Goetz（尤其感谢他细致入微的审校）、Marty Hall、Angelika Langer、Mark Lawrence、Stuart Marks、Attila Szegedi 和 Jim Weave。

我希望你能喜欢这本简明扼要介绍 Java 8 新特性的书，希望它能帮助你成为一个更成功的 Java 开发人员。如果你发现了本书的任何错误，或者有任何改进建议，请访问 http://horstmann.com/java8 并留言。在该页面上，你还可以下载包含书中所有代码示例在内的归档文件。

Cay S. Horstmann

2013 年于旧金山

关于作者

Cay S. Horstmann 不仅是 *Scala for the Impatient*（Addison-Wesley 于 2012 年出版）一书的作者，而且是 *Core Java™, Volumes I and II, Ninth Edition*（Prentice Hall 于 2013 年出版）一书的主要作者，他还编写了一系列针对专业编程人员和计算机专业学生的书籍。他是圣荷西州立大学计算机科学专业的一名教授，也是一位 Java 拥护者。

目　录

试读结束：需要全本请在线购买： www.ertongbook.com

1

Java 作为一门面向对象的编程语言诞生于 20 世纪 90 年代，在当时，面向对象编程是软件开发的主流模式。在面向对象编程出现之前，也曾诞生过像 Lisp 和 Scheme 这样的函数式编程语言，但它们只活跃于学术圈中。最近，由于在并发和事件驱动（或者称“互动”）编程中的优势，函数式编程又逐渐变得重要起来。这并不意味着面向对象编程不好，相反，最终的趋势是将面向对象编程和函数式编程结合起来。即使你对并发等功能不感兴趣，函数式编程也会给你带来帮助。例如，如果语言有了非常方便的函数表达式语法，集合 API 就会变得异常强大。

Java 8 主要是在原来面向对象的基础上增加了函数式编程的能力。在本章中，你将学习基本的语法。下一章将会向你介绍如何利用这些语法来使用 Java 集合类，第 3 章将介绍如何构建自己的函数式 API。

本章的要点包括：

- 一个 lambda 表达式是一个带有参数的代码块。
- 当你想要代码块在以后某个时间点执行时，可以使用 lambda 表达式。
- lambda 表达式可以被转换为函数式接口。
- lambda 表达式可以在闭包作用域中有效地访问 final 变量。
- 方法和构造器引用可以引用方法或构造器，但无须调用它们。
- 你现在可以向接口添加默认（default）和静态（static）方法来提供具体的实现。

- 你必须解决接口中多个默认方法之间的冲突。

1.1 为什么要使用 lambda 表达式

“lambda 表达式”是一段可以传递的代码，因此它可以被执行一次或多次。在学习语法（甚至包括一些奇怪的术语）之前，我们先回顾一下之前在 Java 中一直使用的相似的代码块。

当我们要在另一个独立线程中执行一些逻辑时，通常会将代码放在一个实现 Runnable 接口的类的 run 方法中，如下所示：

```
class Worker implements Runnable {
  public void run() {
    for (int i = 0; i < 1000; i++)
      doWork();
  }
  ...
}
```

然后，当我们希望执行这段代码时，会构造一个 Worker 类的实例。然后将该实例提交到一个线程池中，或者简单点，直接启动一个新的线程：

```
Worker w = new Worker();
new Thread(w).start();
```

这段代码的关键在于，run 方法中包含了你希望在另一个线程中执行的代码。

我们考虑一下用一个自定义的比较器来进行排序。如果你希望按照字符串的长度而不是默认的字典顺序来排序，那么你可以将一个 Comparator 对象传递给 sort 方法：

```
class LengthComparator implements Comparator<String> {
  public int compare(String first, String second) {
    return Integer.compare(first.length(), second.length());
  }
}
Arrays.sort(strings, new LengthComparator());
```

3

sort 方法会一直调用 compare 方法，对顺序不对的元素进行重新排序，直到数组完全有序为止。你给 sort 方法传递了一段需要比较元素的代码片段，而该代码会被整合到排序逻辑中，而你可能并不关心如何在那里实现。

注意：如果 x=y，那么 Integer.compare(x,y)会返回 0；如果 x<y，则它会返回一个负数；而如果 x>y，则它会返回一个正数。这个静态方法已经被添加到 Java 7 中（请参考第 9 章）。还要注意，不应该使用 x-y 来比较 x 和 y 的大小，因为对于大的、符号相反的操作数，这种计算有可能会产生溢出。

按钮回调是另外一个会延迟执行的例子。你将回调操作放到一个实现了监听器接口的类的某个方法中，然后构造一个实例，并将实例注册到按钮上。在这种情况下，许多开发人员都会使用“匿名类的匿名实例”的方法：

```
button.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
      System.out.println("Thanks for clicking!");
    }
});
```

这里的关键是代码处于 handle 方法中。该代码会在按钮被点击时执行。

注意：由于 Java 8 将 JavaFX 作为 Swing GUI 的下一任继承者，我会在这些示例中使用 JavaFX（请参考第 4 章来了解更多关于 JavaFX 的信息）。当然，细节并不重要，因为不管是 Swing、JavaFX 还是 Android，你都需要为按钮添加一些代码，以使它们在按钮被点击时可以执行。

在所有三个例子中，你会看到相同的方式。一段代码会被传递给其他调用者——也许是一个线程池、一个排序方法，或者是一个按钮。这段代码会在稍后被调用。

到现在为止，在 Java 中向其他代码传递一段代码并不是很容易。你不可能将代码块到处传递。由于 Java 是一个面向对象的语言，因此你不得不构建一个属于某个类的对象，由它的某个方法来包含所需的代码。

在其他一些语言中可以直接使用代码块。在很长一段时间里，Java 设计者们都拒绝加入这一特性。毕竟，Java 的一大优势在于它的简单和一致性。如果一个语言包含了所有可以略微简化代码的特性，那么它就会变得不可维护。但是，在其他那些语言中，并不只是产生线程或者注册按钮点击事件的代码变得更简单了，它们大量的 API 都是更简单、更一致、更强大的。虽然我们已经通过类、对象的方式在 Java 中实现了相似的 API

和功能，但是这些 API 使用起来并不让人感到轻松和愉快。

最近的一段时间，争论的问题已经不再是 Java 是否要变成一门函数式编程语言，而是如何实现这种改变了。在设计出一个适合 Java 的解决办法之前已经进行了多年的实验。在下一节中，你将会看到如何在 Java 8 中使用代码块。

1.2 lambda 表达式的语法

还以上一节中的排序为例。我们传递代码来检查某个字符串的长度是否小于另一个字符串的长度，如下所示：

```
Integer.compare(first.length(), second.length())
```

`first` 和 `second` 是什么呢？它们都是字符串。Java 是一个强类型的语言，因此我们必须同时指定类型，如下：

```
(String first, String second)
  -> Integer.compare(first.length(), second.length())
```

这就是你见到的第一个“lambda 表达式”。这个表达式不仅是一个简单的代码块，还指定了必须传递给代码的所有变量。

为什么要叫这个名字呢？许多年前，在计算机出现之前，有位名叫 Alonzo Church 的逻辑学家，他想要证明什么样的数学函数是可以有效计算的。（奇怪的是，当时已经存在了许多已知的函数，但是没有人知道怎样去计算它们的值。）他使用希腊字母的 lambda（λ）来标记参数。如果他懂 Java API 的话，他应该会写下如下代码：

```
λfirst.λsecond.Integer.compare(first.length(), second.length())
```

注意：为什么使用字母 λ？难道 Church 没有其他拉丁字母可用了吗？事实上，经典的《数学原理》中使用“^”符号表示自由变量，这启发 Church 使用大写的 lambda“Λ”来表示参数。但是最终，他选择换回到小写版本。于是从那时起，带有参数变量的表达式都被称为 lambda 表达式。

5

你已经见到了 Java 中 lambda 表达式的格式：参数、箭头 ->，以及一个表达式。如果负责计算的代码无法用一个表达式表示，那么可以用编写方法的方式来编写：即用 `{}` 包裹代码并明确使用 `return` 语句，例如：

```
(String first, String second) -> {
```

```
    if (first.length() < second.length()) return -1;
    else if (first.length() > second.length()) return 1;
    else return 0;
}
```

如果 lambda 表达式没有参数，你仍可以提供一对空的小括号，如同不含参数的方法那样：

```
() -> { for (int i = 0; i < 1000; i++) doWork(); }
```

如果一个 lambda 表达式的参数类型是可以被推导的，那么就可以省略它们的类型，例如：

```
Comparator<String> comp
   = (first, second)// 同(String first, String second) 一样
     -> Integer.compare(first.length(), second.length());
```

这里，编译器会推导出 first 和 second 必须是字符串，因为 lambda 表达式被赋给了一个字符串比较器（我们将会在下一节详细讲解该赋值过程）。

如果某个方法只含有一个参数，并且该参数的类型可以被推导出来，你甚至可以省略小括号：

```
EventHandler<ActionEvent> listener = event ->
   System.out.println("Thanks for clicking!");
     // 无须 (event) -> 或 (ActionEvent event) ->
```

注意：你可以像对待方法参数一样向 lambda 表达式的参数添加注解或者 final 修饰符，如下。

```
(final String name) -> ...
(@NonNull String name) -> ...
```

永远不需要为一个 lambda 表达式执行返回类型，它总是会从上下文中被推导出来。例如，表达式

```
(String first, String second) -> Integer.compare(first.length(), second.length())
```

可以被使用在期望结果类型为 int 的上下文中。

6

注意：在 lambda 表达式中，只在某些分支中返回值（其他分支没有返回值）是不合法的。例如，(int x) -> { if (x >= 0) return 1; }是不合法的。

试读结束：需要全本请在线购买：www.ertongbook.com