Broadview
www.broadview.com.cn

Learning Concurrent Programming in Scala

Scala并发编程

本书由Scala语言的发明者Martin Odersky教授作序

Aleksandar Prokopec 著
苏宝龙 译

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Learning Concurrent Programming in Scala

Scala并发编程

本书由Scala语言的发明者Martin Odersky教授作序

Aleksandar Prokopec 著
苏宝龙 译

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

内 容 简 介

免费的性能午餐已经结束——Intel、AMD 和 Sparc、PowerPC 等主要的处理器生产厂商，从前使用的提高 CPU 性能的传统方法已经走到尽头，单核处理器的主频多年来一直踏步不前，当今主流的 CPU 制造技术是超线程和多核架构。面对计算机性能的挑战和硬件架构的改变，迷惘的软件开发者们应何去何从？本书为大家展示了一条光明的康庄大道！

本书由 Scala 语言的发明者，瑞士洛桑联邦理工大学教授 Martin Odersky 的爱徒 Aleksandar Prokopec 博士撰写。作者根据自己在 Scala 语言开发团队中的工作经验，全面地介绍了并发编程技术。这些并发编程技术包括：并发程序设计的基础知识和背景知识、常用并发实用组件（如线程池、原子变量和并发集合）的用法、Scala 专用并发框架（Future 和 Promise API）的用法、Scala 并行集合框架的用法、使用响应式扩展框架编写基于事件的程序和异步程序的方式、用于编写事务程序的 ScalaSTM 库以及 Actor 编程模型和 Akka 框架。作者在本书中列举的实例既介绍了理论知识又展示了实践方法，同时还能够开拓读者的编程思路。此外，作者还在每章末尾提供了大量编程习题，帮助读者测试和巩固知识。

版权贸易合同登记号　图字：01-2015-4562

图书在版编目（CIP）数据

Scala 并发编程 / 普罗科佩茨（Prokopec,A.）著；苏宝龙译. —北京：电子工业出版社，2015.10
书名原文：Learning Concurrent Programming in Scala
ISBN 978-7-121-27173-1
Ⅰ.①S… Ⅱ.①普… ②苏… Ⅲ.①JAVA 语言－程序设计 Ⅳ.①TP312
中国版本图书馆 CIP 数据核字(2015)第 224190 号

策划编辑：张春雨
责任编辑：张春雨
印　　刷：北京丰源印刷厂
装　　订：三河市皇庄路通装订厂
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：21.25　字数：476 千字
版　　次：2015 年 10 月第 1 版
印　　次：2015 年 10 月第 1 次印刷
定　　价：75.00 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。

译者序

在多核超线程技术没有成为主流处理器架构前，所有软件开发者和他们开发的应用程序都能够不花一分钱地享受硬件升级带来的性能。因为随着 CPU 主频的提高，应用程序的运行速度也会随之提高。这真是天下为数不多的免费午餐之一。

但这幸福的时代随着主流处理器架构的巨变随风而逝了。世界主流的处理器厂商都不再使用提高时钟频率（处理器的主频）和指令吞吐量等传统方式，提高 CPU 的性能了，处理器发展的新方向是多核！

硬件新时代的到来必然会导致软件新时代的到来。并发编程技术虽然从计算机出现的那一天起就存在，但从前它们仅是作为尖端的内核技术被束之高阁。而当前的软件新时代，就是以并发编程技术为主角的新时代。

免费的性能午餐结束后，任何应用程序开发者要想利用超线程多核架构彻底发挥自己应用程序的潜力，就必须掌握并发编程技术。

Scala 即可伸缩的语言（Scalable Language），是一种多范式的编程语言，它以无缝方式集成了面向对象编程和函数式编程的各种特性。

Scala 语言还有以下四个重要优点：

- Scala 语言与 Java 是兼容的——Scala 程序会被编译为 JVM 的字节码。Scala 不仅极度重用了 Java 类型，而且 Scala 代码和 Java 代码可以彼此相互调用。
- 简洁——Scala 程序一般都很短。较保守的估计是标准的 Scala 程序应该有 Java 写的同样的程序一半行数左右。
- 高层级——Scala 使你能够通过提升接口的抽象级别来管理复杂性。
- 静态类型——Scala 语言能够通过类型推断避免代码冗余，能够通过模式匹配及一些新的编写和组织类型的办法获得灵活性。

本书由 Scala 语言的发明者，瑞士洛桑联邦理工大学教授 Martin Odersky 的爱徒 Aleksandar Prokopec 博士撰写。Aleksandar Prokopec 博士本身就是 Scala 语言开发团队的成员，他为 Scala 编程语言的发展做出了许多贡献，开发了 Scala Parallel Collections 框架（一

个高级的 Scala 并行数据编程库），还参与了 Future、Promise 和 ScalaSTM 等抽象的开发工作。

本书行文简洁流畅深入浅出，既介绍了并发编程的基础知识和背景知识，也介绍了许多顶尖的并发编程技巧。本书还提供了大量的实践范例，使读者能够遵循从实践中来到实践中去的指导思想，切实做到知行合一。

本书还对一些著名的并发编程问题和工具做了详细介绍，如 Volatile 变量、ABA 问题以及不可变对象和 final 字段等。

此外，本书还在每一章的末尾附上了大量的编程练习题。这些练习题不仅可以帮助读者测试自己对该章内容的掌握情况，更为重要的是能够开拓读者的编程思路。使读者不仅能够掌握本书介绍的知识，又能够将这些知识灵活地应用到实际的编程工作之中。

翻译前沿计算机科学书籍的工作并不轻松，也不是单独一个人能够完成的。在此我要感谢电子工业出版社张春雨、田蕾等编辑对本书提供的帮助。此外，石浩、孙顾、徐颖、朱晶晶、沈骏杰、何志颖、许诗怡、马佳妮、尹晓婷、徐雯、郭昕、陆迎明和孙艳婷等也参与了本书的翻译工作。

因时间仓促，译者水平有限，本书的错漏之处欢迎广大读者朋友们批评指正。

序

并发和并行编程技术已经从内核程序设计和高性能计算等尖端领域延伸而出，发展成为每位程序员都必备的知识。当前并行和分布式计算系统已经变成了一种规范，大多数应用程序都使用并发处理方式，有些程序是为了提高性能，也有些程序是为了处理异步事件。

目前，大多数开发者都没有做好迎接这场变革的准备工作。也许他们在学校学习过传统的并发模型（基于线程和锁的），但是这类模型无法在确保可接受生产效率的前提下，以可靠的方式处理海量的并发工作。而且，线程和锁不仅难以使用还易于出错。开发者如果想要提高自己的水平，就必须使用等级更高且可组合性更高的并发抽象。

15 年前，我设计了 Scala 语言的前身 Funnel 语言。Funnel 是一种实验性编程语言，其核心中含有并发语义。这门语言将所有编程概念都表现为，以 functional Net 模型（join calculus 模型的面向对象版本）为基础的语法糖。尽管 join calculus 是非常优秀的理论，但经过一些实验后，我们认识到并发问题是多层面的，无法通过单一形式解决它。解决所有并发问题的万能灵药是不存在的；正确的解决方式是对症下药。你是否想要定义用于回应事件或数据流的异步计算？是否想要创建用于通过消息进行通信的独立实体？是否想要为可变数据定义事务？是否想要通过并行处理方式提高性能？Scala 语言提供了完成所有这些任务的抽象：Future 类、响应式事件流、Actor 类、事务内存和并行集合。

Scala 和这本书使我们走到了一起。当前有如此多的优秀并发抽象，通过编程语言以硬编码的方式实现它们好像意义不大。但我们通过 Scala 努力实现的目标，是能够在用户代码和库中更轻松地定义高级抽象的。通过这种工作模式，开发者可以定义用于处理各种并发程序设计层面的模块。所有这些模块都会建立在由主机系统提供的低等级内核上。现在回过头来看，这种工作模式确实取得了很好的效果。Scala 拥有当今功能最强、结构最简洁的并发编程库。本书将带你领略其中最重要的几个库的风采，使你了解这些库的擅长领域和使用它们的方式。

不是计算机专家就写不好计算机书籍。本书的作者 Aleksandar Prokopec 编写过多个最流行的 Scala 并发和并行编程库，他还发明过多个精妙的数据结构和算法。在推出这本书

的同时，他还创建了网上教程和权威参考站点。我相信本书能够成为所有 Scala 并发和并行程序开发者的良师益友。我期望这本书能够成为对并发领域感兴趣的人的书架藏品，因为他们都预见到了正在快速发展的并发计算领域的美好未来。

Martin Odersky

Scala 语言的发明者，瑞士洛桑联邦理工大学教授

作者简介

Aleksandar Prokopec 是一位软件开发者，同时也是并发和分布式编程技术研究者。他拥有克罗地亚萨格勒布大学电子工程和计算学院的计算机专业硕士学位，和瑞士洛桑联邦理工大学（EPFL）的计算机科学专业博士学位。作为 EPFL 博士助教和 Scala 语言开发团队成员，他积极为 Scala 编程语言做贡献，研究并发编程抽象、并行数据编程支持和 Scala 并发数据结构。他编写了 Scala Parallel Collections 框架，这是一个高级的 Scala 并行数据编程库。他还参加了多个 Scala 并发库开发小组，开发了 Future、Promise 和 ScalaSTM 等抽象。

致谢

首先，我要感谢本书的审稿人 Samira Tasharofi、Lukas Rytz、Dominik Gruntz、Michel Schinz、Zhen Li 和 Vladimir Kostyukov，他们为我提供了宝贵的反馈信息和建议。为了提高本书的质量，他们展示了高超的专业知识和难得的奉献精神。我还要感谢 Packt Publishing 出版社的编辑：Kevin Colaco、Sruthi Kutty、Kapil Hemnani、Vaibhav Pawar 和 Sebastian Rodrigues，他们为本书提供了许多帮助。能够与这些伙伴一起工作，我倍感荣幸！

不汇聚众人的智慧结晶，本书介绍的并发框架是不会面世的。许多人或直接或间接地为这些实用组件的开发工作做出了贡献。这些人是创建 Scala 并发处理模式的真正英雄，我衷心感谢他们对 Scala 并发编程支持功能所做的贡献。在此难以将他们的姓名都列举出来，但我会尽最大努力。如果漏掉了某位朋友，请你联系我，我会在本书的下一版中增补这个名单。

无须赘言，Scala 编程语言的发明者 Martin Odersky 教授的功劳首屈一指。在此我要特别感谢他与过去十多年来参与 EPFL 的 Scala 语言开发小组的人们。我还要感谢 Typesafe 公司的员工，他们一直在为使 Scala 成为一种通用语言而努力工作。

大多数 Scala 并发框架都在某种程度上以 Doug Lea 的工作成果为基础。他编写的 Fork/Join 架构，是 Akka 框架的 Actor 类、Scala Parallel Collections 框架以及 Future 和 Promise 库的基础；而且本书介绍的许多 JDK 并发数据结构也是由他开发的。许多 Scala 并发库都融入了他的建议。感谢 Doug Lea 所做的贡献。此外，我还要感谢为使 JVM 成为可靠的并发程序平台，而常年努力工作的 Java 并发专家们。我要特别感谢 Brian Goetz，他为本书的封面设计提供了灵感。

Scala 的 Future 和 Promise 库最初是由来自 EPFL 的 Philipp Haller、Heather Miller、Vojin Jovanović 和我、Akka 框架开发小组的 Viktor Klang 和 Roland Kuhn、Twitter 网站的 Marius Eriksen 一起设计的，而且还获得了 Havoc Pennington、Rich Dougherty、Jason Zaugg、Doug Lea 等人的帮助。

尽管我是 Scala Parallel Collections 框架的主要开发者，但是在编写这个库的过程中我

们仍然获得了许多人的帮助，其中包括：Phil Bagwell、Martin Odersky、Tiark Rompf、Doug Lea 和 Nathan Bronson。后来，Dmitry Petrashko 和我开始编写并行和标准集合操作的改进版本，该版本是通过 Scala 宏指令进行优化的。感谢 Eugene Burmako 和 Denys Shabalin 等人对 Scala 宏项目做出的贡献。

本书介绍的 Rx 框架是由 Erik Meijer、Wes Dyer 等 Rx 开发小组成员编写的。从该框架最初的.NET 实现代码开始，Rx 框架就为多种编程语言提供了接口，这些语言包括 Java、Scala、Groovy、JavaScript 和 PHP。Rx 框架本身也获得了广泛认可。感谢 Ben Christensen、Samuel Grütter、Shixiong Zhu、Donna Malayeri 等人对该框架所做的贡献。

Nathan Bronson 是 ScalaSTM 项目的最大功臣之一，ScalaSTM 的默认实现版本就是根据 Nathan Bronson 开发的 CCSTM 项目编写的。ScalaSTM API 是由多位 ScalaSTM 专家设计的，这些专家包括 Nathan Bronson、Jonas Bonér、Guy Korland、Krishna Sankar、Daniel Spiewak 和 Peter Veentjer。

设计 Scala Actor 库的灵感来自 Erlang 语言的 Actor 模型，最初该库是由 Philipp Haller 编写的。Jonas Bonér 从这个库获得灵感，并开始开发 Akka Actor 框架。开发 Akka 框架的功臣有多位，其中包括 Viktor Klang、Henrik Engström、Peter Vlugter、Roland Kuhn、Patrik Nordwall、Björn Antonsson、Rich Dougherty、Johannes Rudolph、Mathias Doenitz、Philipp Haller 等。

最后，我要感谢整个 Scala 社区对 Scala 语言所做的贡献，是他们使 Scala 成为了一门优秀的程序设计语言。

审稿人简介

Dominik Gruntz 拥有苏黎世理工学院的博士学位，并且从 2000 年起就担任瑞士西北应用科学大学的计算机科学专业教授。除了完成研究项目，他还教授并发编程课程。几年前，这门课程的目标还仅是向学生们展示并发程序的复杂性，和在普通程序中不适合使用并发技术的情况（这门课仅是普通的教学目标）。

随着 Java 和 Scala 高级并发框架的出现，这种情况已经改变，所有想要编写正确、可靠且高效的并发程序的程序员，都可以将本书视为极佳的资源。本书是理想的并发程序设计课程教材。

感谢 Packt Publishing 出版社提供机会，使我能够作为审稿人，参与本书的创作。

Zhen Li 自从上小学时接触到 Logo 语言后，她就对计算机科学产生了极大兴趣。从复旦大学获得软件工程专业学位和从爱尔兰都柏林大学获得计算机科学专业学位后，她到美国的乔治亚大学攻读博士学位并进行研究工作。她的研究方向是程序员学习行为背后的心理活动，专攻程序员理解并发程序时的心理学。她的目标是根据研究成果，开发出高效的软件工程方法和教学范式，帮助程序员掌握并发编程技术。

Zhen Li 拥有计算机专业本科课程的实践教学经验，这些课程包括系统和网络编程、建模和仿真及人机交互。她在计算机教学方面的主要贡献，是编写了多种编程语言和并发模式的教学大纲和课程，从而鼓励学生们积极地体会软件设计哲学，并全面地掌握并发编程技术。

Zhen Li 还拥有产业化创新工作经验。十多年来，她曾经在各种 IT 企业中工作过，这些企业包括 Oracle、Microsoft 和 Google 等。在这些知名公司中，她参与了最前沿产品、平台和核心企业基础架构，以及云业务技术的开发工作。

Zhen Li 对编程和教学工作充满热情。你可以通过 janeli@uga.edu 邮箱联系她。

Lukas Rytz 是 Typesafe 公司中 Scala 开发团队中的一名编译程序工程师。他在 2014 年从 EPFL 取得了博士学位，而他的博导就是 Scala 语言的发明者 Martin Odersky 教授。

Michel Schinz 是 EPFL 的一位讲师。

Samira Tasharofi 拥有伊利诺伊大学软件工程专业的博士学位。她的研究工作涉及多个领域，其中包括并发程序和特殊 Actor 程序测试、并行编程模式及鉴定基于组件的系统。

Samira Tasharofi 利用她丰富的实践经验进行研究工作，数年来她曾就职于多家 IT 公司，如 Microsoft 和 LinkedIn。她还担任过多本书籍的审稿人，其中包括 *Actors in Scala*、*Parallel Programming with Microsoft® .NET: Design Patterns for Decomposition and Coordination on Multicore Architectures*（*Patterns and Practices* 系列）和 *Parallel Programming with Microsoft Visual C++: Design Patterns for Decomposition and Coordination on Multicore Architectures*（*Patterns & Practices* 系列）。她还是多场软件工程大会和研讨会的技术研究论文审稿人，这些会议包括：ASE、AGERE、SPLASH、FSE 和 FSEN。她还担任过第四届 International Workshop on Programming based on Actors, Actors, Agents, and Decentralized Control 研讨会（AGERE 2014）和第六届 IPM International Conference on Fundamentals of Software Engineering 大会（FSEN 2015）的委员会委员。

感谢为我提供的机会，使我能够有幸为这本书出一份力。

献给 Sasha，她可能是读过本书的唯一一位物理化学博士。

前言

并发处理方式无处不在。随着多核处理器在消费者市场中崛起，并发编程技术也在软件开发行业内取得了霸主地位。并发编程技术以前就在程序或计算机系统中被用于实现异步计算，并且自成一个学术领域，而现在并发程序设计已经变成了开发人员普遍使用的软件开发技术。因此，高级并发框架和库正在以惊人的速度发展。近年来，并发计算领域呈现出了与文艺复兴类似的蓬勃发展景象。

随着现代编程语言和并发框架抽象等级的提高，了解和使用这些语言和框架的时机已经变为极为重要的知识。仅掌握经典并发和异步基元（如线程、锁和监控器）的使用方式，已经不足以应对当前的形势。高级并发框架既可以解决传统并发问题，也可以为特定任务做出调整。因此，高级并发框架逐渐占领了并发编程领域。

本书介绍 Scala 语言高级并发编程技术。本书不仅详细介绍了各种并发主题，还介绍了并发编程的基础理论。同时，本书也介绍了多种主要的现代并发框架、这些框架的语义细节和使用这些框架的方式。这样做的目的是展示重要的并发抽象，同时让你了解在实践中使用这些抽象的方式。

我们相信当你阅读了这本书后，既可以掌握坚实的并发编程理论基础，又能够获得编写正确、高效并发程序的实用技巧。这些技巧是通向现代并发编程专家之路的基石。

希望本书能够为你带来快乐。

本书的结构

本书的主要目标是帮助你掌握开发正确、高效并发程序的必备技巧。获得技巧的最佳方式是在实践中使用它。因此，获得编程技巧的最佳方式，是亲手编写程序。本书旨在通过一系列精心挑选的案例程序，全方位向你展示并发编程技术，从而使你掌握使用 Scala 语言编写并发程序的方式。这些案例程序包括从最简单的 Hello World 示例程序，到错综复杂的高级并发程序。

本书中程序的一个最大共同点是，既简洁又独立。选取这类程序的优点有两个。首先，你可以不受干扰地学习大多数案例程序。尽管我们建议你按照先后顺序阅读本书的各个章

节，但如果你想要先关注某个主题也不会有问题。其次，简洁性可以确保每个新概念都能够易于被掌握和理解。通过简单的示例程序，理解原子处理方式、内存争用和忙等待等问题要容易得多。但这不意味着这些示例程序，仅是为展示这些概念而专门编写的；尽管有些案例程序可能会被删去不相关的部分，但每个案例程序都是选自现实世界的真实案例。

我们强烈建议你在阅读本书的过程中，亲手编写和运行这些示例程序，而不应仅是消极地阅读它们。每个案例程序都会向你展示一个新概念，但是你只有亲手在实践中使用这些概念，才能彻底理解它们。通过亲手运行并发程序并观察其特殊效果获得的经验，比仅从纸上阅读它得到的经验要宝贵得多。因此，你应该下载 SBT，并在阅读本书前创建一个空白项目，这一点请参阅本书中介绍的详细步骤。本书选取的案例程序都比较短小，从而使你能够毫无压力地尝试它们。

本书每一章的末尾都有一组编程练习。这些练习专门用于测试你对该章内容的理解程度。我们建议你在阅读完一章内容后，至少做几道该章末尾的练习题。

在大多数情况中，我们会避免展示 API 方法和它们的确切签名。这样做的原因有多个。首先，你随时可以在网上学习 ScalaDoc 文档介绍的 API。本书如果重复这些内容会浪费一些篇幅。其次，软件永远都处于不断变化的状态中。尽管 Scala 并发框架的设计者们努力使这些 API 具有稳定性，但这些方法的名称和签名也可能会偶尔改变。本书介绍了改变可能性不高的许多重要并发工具的语义，使用这些工具足以编写并发程序。

本书的目标不是事无巨细地介绍 Scala 并发 API 的所有知识。本书旨在介绍最重要的并发编程概念。掌握了本书介绍的知识后，你不仅可以阅读网上文档介绍的补充资料，还可以明确寻找这些资料的方向。本书不是详细介绍每个方法确切语义的 API 参考文档，教授你从这些参考资料获取知识的方式才是本书的目的。当你阅读了本书的内容后，你不仅可以了解各种并发库不同的运行方式，还能获得编写并发程序的思路。

本书的内容

本书根据各个并发编程主题，划分为多个章节。本书的内容涵盖 Scala 运行时系统中的基础并发 API、较复杂的并发基元和高级并发抽象概要。

第 1 章介绍编写并发程序的原因和一些背景知识。本章也介绍了 Scala 编程语言的基础知识，以便使你能够顺畅地阅读本书后面介绍的内容。

第 2 章介绍并发程序设计的基础知识。本章介绍了使用线程的方式、防止以并发方式访问共享内存的方式，以及 Java 内存模型（JMM）。

第 3 章介绍常用的并发实用组件，如线程池、原子变量和并发集合。本章还着重介绍了使用并发集合与 Scala 语言功能进行交互的方式。本书将重点放在高等级的现代并发编程框架上；因此，虽然本章概述了传统的并发编程技巧，但并非将重点放在这个方面。

第 4 章是第一个介绍 Scala 专用并发框架的章节。本章介绍了 Future 和 Promise API，以及在编写异步程序时正确使用这些 API 的方式。

第 5 章介绍 Scala 并行集合框架。本章介绍了在条件允许的情况下，通过并行方式处理集合操作的方式，以及通过这种方式提高性能的手段。

第 6 章介绍使用响应式扩展框架编写基于事件的程序和异步程序的方式。本章会介绍事件流操作与集合操作的对应关系、在线程之间传递事件的方式和使用事件流设计响应式用户界面的方式。

第 7 章介绍用于编写事务程序的 ScalaSTM 库，使用该库可以获得更加安全、更加直观的共享内存编程模型。本章会介绍使用可伸缩的内存事务防止多个线程以并发方式访问共享内存的方式，还会介绍减少死锁和竞态条件的方式。

第 8 章介绍 Actor 编程模型和 Akka 框架。本章会介绍编写在多台计算机上运行的、以透明方式传递消息的分布式程序的方式。

第 9 章总结了前面各章介绍过的多种并发库。本章会介绍根据指定问题选择正确并发抽象的方式，还会介绍当设计较大的并发应用程序时，将多种并发抽象组合到一起的方式。

我们建议你根据先后顺序阅读各个章节，但这并非严格要求。如果你已经很好地掌握了第 2 章介绍的内容，就可以直接学习大多数章节。对其他章节介绍的知识依赖程度较高的是第 9 章，因为这一章是对前面介绍的各个主题的总结。

阅读本书前需要具备的知识

本节介绍阅读本书之前，你需要具备的知识。本节介绍了安装 Java Development Kit（Java 开发工具集）的方式，它是用于运行 Scala 程序的开发环境；还介绍了使用 Simple Build Tool（SBT）项目构建工具运行各种示例程序的方式。

阅读本书前无须具备 IDE 知识。你可以根据自己的喜好选择编写代码的编辑器，如 Vim、Emacs、Sublime Text、Eclipse、IntelliJ IDEA 和 Notepad++等文本编辑器。

安装 JDK

Scala 程序不会被直接编译为机器码，因此它们无法像可执行程序那样在各种硬件平台上直接运行。Scala 编译器生成的是一种中间代码——Java 字节码。要运行这种中间代码，你的计算机必须安装 Java 虚拟机（JVM）软件。本节介绍下载和安装 Java Development Kit 的方式，该开发环境集成了 Java 虚拟机和其他有用的工具。

软件市场上有多个 JDK 版本。我们建议你使用 Oracle JDK 分发版本。你可通过下列步骤下载和安装这个版本的 Java Development Kit：

1. 在你的浏览器中打开 URL：`www.oracle.com/technetwork/java/javase/downloads/index.html`。

2. 如果你无法打开这个 URL，可以在搜索引擎中搜索关键字 JDK Download。

3. 一旦你找到 Oracle 网站中的 Java SE 下载链接，应下载与你当前使用的操作系统（如 32 位或 64 位的 Windows、Linux 或者 Mac OS X）对应的 JDK 7 版本。

4. 如果你使用 Windows，那么只需运行安装程序即可。如果你使用 Mac OS X，则需要打开 `dmg` 存档才能安装 JDK。如果你使用 Linux，则需要将 `dmg` 存档解压到 XYZ 目录，并在 PATH 变量中添加 `bin` 子目录：

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

5. 现在你就能够在命令行界面中运行 `java` 和 `javac` 命令了。输入 `javac` 命令，查看该命令是否可用（本书没有介绍配置这个命令的方式，但你可以通过运行它，查明它是否可用）：

```
javac
```

这样你的操作系统就安装好了 JDK。要验证这一点，只需执行 `javac` 命令，如第 5 步骤所示。

安装和使用 SBT

SBT 是一种用于构建 Scala 项目的命令行工具。它的作用是编译 Scala 代码、管理依赖关系、进行持续编辑和测试、开发等。本书通篇都会使用 SBT，管理项目的依赖关系和运行示例程序。

通过下列步骤可以安装 SBT：