Broadview®
www.broadview.com.cn 博文视点

综合进阶向导

Scala 编程

Programming in Scala

黄海旭　高宇翔　译

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

Scala编程

Programming in Scala

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

内 容 介 绍

本书介绍了一种新的编程语言，它把面向对象和函数式编程概念有机地结合为整体，从而形成一种完整统一、语义丰富的新思维体系。本书循序渐进，由浅入深，经作者精心组织、仔细编排，将语言中的各种概念自然地铺陈在字里行间。除此之外，本书还包含了大量富有针对性和趣味性的示例，它们除了提供对语言各个方面的具体演示之外，还从侧面说明了如何将函数式编程的理念切实并广泛地应用到面向对象编程中。本书面向的读者是有一定编程经验的开发人员，他们希望能够开拓眼界，并致力于提高在软件开发各方面的技能。

版权贸易合同登记号　图字：01-2010-4536

图书在版编目（CIP）数据

Scala编程 / 奥德斯基（Odersky, M.），莱斯彭（Spoon, L.），凡纳斯（Venners, B.）著；
黄海旭，高宇翔译. —北京：电子工业出版社，2010.11
书名原文：Programming in Scala
ISBN 978-7-121-12119-7
Ⅰ. ①S… Ⅱ. ①奥… ②莱… ③凡… ④黄… ⑤高…
Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312
中国版本图书馆CIP数据核字（2010）第212228号

策划编辑：卢鹓翔
责任编辑：杨绣国
印　　刷：北京天宇星印刷厂
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：787×1092　1/16　**印张：**33　**字数：**782千字
印　　次：2010年12月第1次印刷
定　　价：89.00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

最初接触 Scala 是在 2008 年的 9 月。当时刚刚换了公司，工作上出现了一段“空档期”，因此非常想找一些有技术含量的事情来做。而编程语言的设计开发，或者更确切地说只是对于“银弹”的幻想，这始终是我的爱好之一。那时总会感觉自己应该为这么多年的开发工作总结一些什么，让它们变成之后工作的基础，或者即使没有那么基础也可以是一种参考，即便连参考都不算，至少也是一种思考和总结吧！

然后我重新审视了自己曾做过的项目，分析了项目中使用过的编程语言（Delphi / C++ / C# / Java），总结其中的优缺点，想象着如果是我做的话会有什么样的改善，甚至开始着手策划语言的语法结构，希望能够自圆其说，从只言片语中建立起完整的逻辑体系。在这样的过程中，我时常会发现自己钻进了死胡同，似乎不断地修改只是在不断地重复，甚至唯一的改进也只是“语法糖”而已。没有发现与其他语言间的标志性差异，不能建立独有的思维方式。

于是我在互联网上寻找有没有什么现成的、别出一格的新观点或想法，能够启发我步出自己的圈子，带给我一种“啊哈，原来是这样！”的感觉。不过真没想到，很快，我就找到了。而且是这么彻底，甚至让我打消了“从轮子造起”的念头。

不得不说，Scala 语言的确具有很新颖的创意。这在之前所用的语言中是不多见的。

相信大多数商业软件的程序员都是从 Fortran、Pascal、C/C++/Java 或 COBOL 语言一路学起的。本书中，这一大类的语言被称为是指令式语言，意指以变量和过程语句构造指令的方式对需要实现的目的进行编码的语言。与之相对的是被称为函数式语言的另一个大类，这种语言把程序的目的转换为函数和函数应用，是以函数计算这种不同的视角来处理一般问题的。

直到现在，Lisp、Scheme 或者 Haskell 这样的函数式语言也仅仅还是“小众”语言。这在 TIOBE 语言排行榜上也可见一斑。上榜的前 20 位几乎被面向对象的或者面向过程的指令式语言占满。尤其是前三甲则几乎始终被 Java、C、和 C++所把持。反观函数式语言，Lisp 在第 13 位，是最靠前的了，Scheme 在第 25 位，Haskell 在第 40 位左右。但并不是说函数式语言对于指令式语言就不具有可比性，相反，在很多地方函数式语言都体现了它们独到的优势，如：代码的简洁性、值的不可改变、函数的无副作用、值和函数替换的指称透明等。

但函数式语言与指令式语言总显得有点儿“格格不入”。一个被指令式语言锻炼出来的头脑很难理解函数式语言的精髓，已经用惯了循环语句的方式解决问题，想要把它转换为递归语句也不是很容易的事情。莫非，函数式语言始终只能作为学术派的思维游戏，而不能适应于现代商业软件开发的需

要，因此无缘商业软件开发浪潮的洗礼吗？

所以说，Scala 语言的确具有很新颖的创意。因为它能很巧妙地把指令式语言与函数式语言结合在一起。用 Martin Odersky 的话说，就是：指令式语言与函数式语言本就是一枚硬币的两面，它们不应是互相排斥、非此即彼的，而是可以互相协调，共同发挥出更大力量的。

Scala 语言还从许多其他的语言中汲取了长处，构造出了自己完整而优雅的思想体系。从语言的各个方面来看，似乎都存在一些有别于其他语言的特点，但从语言的整体性来看，它的每个特点都不是孤立的，而是作为一个有机的整体自然而然生发出来的功能。你可以发现，Scala 语言中的基本概念并不多，甚至与其他的语言相比，原生数据类型和语法结构还“显得”颇为不足。但 Scala 具有强大的自发展能力，能够构建出“乱真”的自定义语法库，这使得 Scala 语言成为了一种可以依赖于有限的构件创造出几乎无限可能的语言。

（写到这里我眼前总能立刻浮现出 DNA 结构的四个基本构成部分，谁能想到几乎所有复杂生命体的程序都仅仅使用 4 个单词就能够拼装出来呢？）

不过本书还不止是一部编程语言的教材，它还是一部编程方法论的说明。由于本书三位作者同时也是 Scala 语言的设计和开发者，所以想要了解个中秘密真是再也找不到更好的透露者了。通过本书你不但能够学习到这种语言的语义语法结构、应用场景、用法、实例，还能够发现语言作者在这些结构之后的考量、动机、权衡和折中，从中可以领悟到许多编程专家的思维模式，学习到他们处理问题的方法，从而开拓你的眼界，让你不仅在软件代码开发方面，而且还在软件的架构设计思想上面有新的感悟。

我必须承认，在接触 Scala 之前，我只是一个普通的 Java 开发人员，是完全“指令式”的软件从业者。是 Scala 语言开始让我领略到了函数式编程思想的奥妙。因此，借此机会想向语言的创造者表示自己的敬意，是你们非凡的工作给了我另一种全新的思考问题的方式，同时也“断送”了我“制造轮子”的梦想（笑）。

本书的翻译工作分为两部分，其中，前 27 章由本人完成，第 28 至 33 章由高宇翔翻译。

另外，还要感谢电子工业出版社不遗余力地引进国外优秀 IT 类书籍，感谢博文视点的徐定翔编辑在本书筹划阶段给予我参与的机会；感谢卢鹓翔编辑为译者在本书翻译阶段提供的大力支持；感谢刘唯一编辑和杨绣国编辑为本书的统稿和排版不辞辛劳的工作。翻译本书的过程中，译者虽已尽最大努力确保专业术语的统一和准确，也尽最大努力将原作者的意境用朴实的中文展现给读者，但囿于个人的水平，书中的问题和疏漏之处在所难免，敬请读者朋友给予批评指正。

黄海旭

2010 年 10 月于上海浦东

致 Nastaran - M.O.

致 Fay - L.S.

致 Siew - B.V.

序

Martin Odersky 设计的 Pizza 语言曾经震惊了 Java 世界。尽管 Pizza 语言本身并不流行，但它巧妙地把面向对象和函数型语言两种风格融合在了一起，形成了自然而又强有力的组合。Pizza 语言的设计成为了 Java 泛型的基础，Martin 的 GJ（Generic Java）编译器从 Java 1.3 开始成为了 Sun 公司的标准编译器（尽管关闭了泛型）。我有幸维护这个编译器多年，因此对 Martin 设计与实现语言的能力有非常直接的体会。

那时候我们还在 Sun 公司，尝试用一些零打碎敲的、针对特定问题的解决方案来扩展语言，比如 for-each 循环、枚举、自动装包，以简化程序的开发，而 Martin 则继续着他在更强大的正交语言原语方面的工作，帮助程序员用库来提供解决方案。

近年来，静态类型语言受到了冲击。Java 的广泛应用暴露了静态语言编程会导致大量固定写法的弊病。常见的看法是我们应避免静态类型从而消除这种代码，于是人们对动态语言如 Python，Ruby 和 Groovy 的兴趣开始增加。这种看法被 Martin 最近的构想，Scala 的出现打破。

Scala 是一种类型优雅的语言：它是静态类型的，但仅在需要的地方显式定义类型。Scala 从面向对象和函数式语言两方面获得了强大的特性，然后用全新的理念把它们完美地整合成一体。它的语法是如此的轻量级，而原语（primitive）又如此富有表达力，以至于根本可以认为 API 的使用不会产生语法开销。我们可以在标准库中，如拆分器、组合器和执行器中发现例子。而这也说明 Scala 是支持内嵌的特定领域语言。

Scala 会成为下一个伟大的语言吗？只有时间可以证明一切。我相信 Martin Odersky 的小组绝对有这样的能力和水平做到这一点。不过有一件事是确定无疑的：Scala 语言建立了衡量未来语言的新标准。

Neal Gafter
圣约瑟，加利福尼亚
2008 年 9 月 3 日

致谢

许多人持续关注本书及其相关资料，在这里表示感谢。

Scala 语言本身已经成为许多人努力的集合。版本 1.0 的设计和实现得到了 Philippe Altherr、Vincent Cremet、Gilles Dubochet、Burak Emir、Stéphane Micheloud、Nikolay Mihaylov、Michel Schinz、Erik Stenman 和 Matthias Zenger 等人的帮助。Iulian Dragos、Gilles Dubochet、Philipp Haller、Sean McDirmid、Ingo Maier 和 Adriaan Moors 参与了第二版和当前版语言及工具开发的工作。

Gilad Bracha、Craig Chambers、Erik Ernst、Matthias Felleisen、Shriram Krishnamurti、Gary Leavens、Sebastian Maneth、Erik Meijer、David Pollak、Jon Pretty、Klaus Ostermann、Didier Rémy、Vijay Saraswat、Don Syme、Mads Torgersen、PhilipWadler、Jamie Webb 和 JohnWilliams 通过生动和启发性的讨论，或者对此篇文稿早期版本的评注，热情地与我们分享了他们的观点，从而使语言的设计得以成型。Scala 语言电子邮件列表的建设者们同样提供了非常有用的反馈信息来帮助我们改善语言及相关工具。

George Berger 为本书更为流畅的制作过程和 Web 体验做出了巨大的贡献。令人欣慰的是这个项目没有变成一个技术大杂烩。

许多人给我们的早期版本提供了反馈信息。在这里感谢 Eric Armstrong、George Berger、Gilad Bracha、William Cook、Bruce Eckel、Stéphane Micheloud、Todd Millstein、David Pollak、Frank Sommers、Philip Wadler 和 Matthias Zenger。同样感谢硅谷模式组成员他们大有助益的审校：Dave Astels、Tracy Bialik、John Brewer、Andrew Chase、Bradford Cross、Raoul Duke、John P. Eurich、Steven Ganz、Phil Goodwin、Ralph Jocham、Yan-Fa Li、Tao Ma、Jeffery Miller、Suresh Pai、Russ Rufer、Dave W. Smith、Scott Turnquest、Walter Vannini、Darlene Wallach 和 Jonathan Andrew Wolter。我们同样还要感谢 Dewayne Johnson 和 Kim Leedy 在封面设计上的帮助，还有 Frank Sommers 在索引上的工作。

我们要特别感谢所有那些曾经提供给我们建设性意见的读者。你们的建议对我们把本书做得更好提供了非常大的帮助。我们没办法印出所有参与者的名字，但以下是在 eBook 预印刷阶段通过点击建议链接提供了至少五条评论的读者名单，首先以最高评论数排序，然后是字母顺序，感谢这些人：David Biesack、Donn Stephan、Mats Henricson、Rob Dickens、Blair Zajac、Tony Sloane、Nigel Harrison、Javier Diaz Soto、William Heelan、Justin Forder、Gregor Purdy、Colin Perkins、Bjarte S. Karlsen、Ervin Varga、Eric Willigers、Mark Hayes、Martin Elwin、Calum MacLean、Jonathan Wolter、Les Pruszynski、Seth Tisue、Andrei Formiga、Dmitry Grigoriev、George Berger、Howard Lovatt、John P. Eurich、Marius Scurtescu、Jeff Ervin、Jamie Webb、Kurt Zoglmann、Dean Wampler、Nikolaj Lindberg、Peter McLain、Arkadiusz Stryjski、Shanky Surana、Craig Bordelon、Alexandre Patry、Filip Moens、Fred Janon、Jeff Heon、Boris

Lorbeer、Jim Menard、Tim Azzopardi、Thomas Jung、Walter Chang、Jeroen Dijkmeijer、Casey Bowman、Martin Smith、Richard Dallaway、Antony Stubbs、Lars Westergren、Maarten Hazewinkel、Matt Russell、Remigiusz Michalowski、Andrew Tolopko、Curtis Stanford、Joshua Cough、Zemian Deng、Christopher Rodrigues Macias、Juan Miguel Garcia Lopez、Michel Schinz、PeterMoore、Randolph Kahle、Vladimir Kelman、Daniel Gronau、Dirk Detering、Hiroaki Nakamura、Ole Hougaard、Bhaskar Maddala、David Bernard、DerekMahar、George Kollias、Kristian Nordal、Normen Mueller、Rafael Ferreira、Binil Thomas、John Nilsson、Jorge Ortiz、Marcus Schulte、Vadim Gerassimov、Cameron Taggart、 Jon-Anders Teigen、Silvestre Zabala、Will McQueen、还有 Sam Owen。

最后，Bill 还要感谢 Gary Cornell、Greg Doench、Andy Hunt、Mike Leonard、Tyler Ortman、Bill Pollock、Dave Thomas 和 Adam Wright 对本书出版方面提供的观点和建议。

简介

本书是 Scala 编程语言的教程。写给那些直接参与 Scala 开发的人群。我们的目标是通过阅读此书，你能够学会一切所需，成为多产的 Scala 程序员。本书中所有的例子都能在 Scala 版本 2.7.2 下面编译通过。

谁应该阅读此书

本书的主要目标读者是那些想要学习使用 Scala 编程的程序员。如果你打算用 Scala 做下一个软件项目，那么本书是为你准备的。而且，本书还会传授一些新概念以满足那些希望拓展视野的程序员们。比如说，如果你是 Java 程序员，阅读本书将使你领略从函数型编程到高级面向对象思想的许多概念。我们相信学习 Scala 及它隐含的理念，通常能使你成为更好的程序员。

本书假设你已经有了基本的编程知识。当然 Scala 也可以很好地作为首次学习的编程语言，但这不是学习如何编程的书。

从另一方面来说，阅读本书并不需要特定的编程语言知识。尽管大多数人在 Java 平台上使用 Scala 语言，但本书并不设定你具备 Java 基础知识。然而，我们希望读者能够熟悉 Java，这样我们可以在某些时候通过比较 Scala 和 Java 来帮助这些读者明白其中的差别。

如何使用本书

本书的主要目的是作为教材，所以推荐的阅读方式是按照章节的次序，从头到尾完成阅读。我们尽力一次介绍一个话题，并且仅用已经介绍过的话题来说明新的话题。因此，如果你跳到后面想先睹为快，就可能会发现有些东西用了不太明白的概念来解释。我们认为按照章节的顺序阅读，这种一步一个脚印的方式将引导你顺利地获得 Scala 开发的技能。

如果你发现一个不懂的术语，请一定查找一下术语表和索引。许多读者会略过书中的某些部分，这不是问题，术语表和索引有助于你返回到你略过的某些东西。

读完一遍之后，本书还可以作为语言参考书。Scala 语言有正式的定义，但是语言的定义是以可读性变差为代价换取精确性好的文档。尽管本书并未涵盖 Scala 的所有细节，但在你更好地掌控 Scala 编程之前，它已经足够作为一本通信易懂的语言参考书了。

如何学习 Scala

简单地通读本书，你将学到 Scala 的许多东西。但如果再稍微多付出一点努力，就能更快更全面地

了解 Scala。

首先，你可以充分利用本书中包含的许多编程例子。尝试自己输入能强迫你的大脑思考每一行代码。而尝试各种各样的变化能让它们变得更有趣也能让你确信已真正明白它们如何工作。

其次，与多个在线论坛保持联系。采用这种方式，你和其他 Scala 爱好者就能够互相帮助。你还可以访问更多的电子邮件列表、讨论论坛、聊天室、维基百科和多个特别为 Scala 准备的文档资料更新点。花些时间来查找包含你需要的信息的地方。这样，花更少的时间在小问题上，就能花更多的时间在更深入和更重要的地方。

最后，一旦你已经掌握了部分内容，请把它用在你自己的编程项目上。从草案开始开发一个小程序，或大一点儿程序的附加部分。仅仅看书不能学到更多东西。

印刷体变化

首次使用的术语（term），将使用括号演示原来的英文词语。小代码例子，如 `x+1`，将用等宽字体演示在文档段落中。大段的代码例子将放在等宽字体的段落中演示：

```
def hello() {
  println("Hello, world!")
}
```

在演示交互式 shell 的时候，shell 的回应将演示为深红色字体。

```
scala> 3 + 4
res0: Int = 7
```

内容概要

- 第 1 章，“可伸展的语言”，给出了 Scala 的设计，和它后面的理由，历史的概要。
- 第 2 章，“Scala 的入门初探”，展示给你如何使用 Scala 完成若干种基本编程任务，而不牵涉过多关于如何工作的细节。本章的目的是让你的手指开始敲击并执行 Scala 代码。
- 第 3 章，“Scala 的入门再探”，演示更多的基本编程任务来帮助你更快速地上手 Scala。本章之后，你将能够开始在简单的脚本任务中使用 Scala。
- 第 4 章，“类和对象”，通过描述面向对象语言的基本建设模块和如何编译及运行 Scala 程序的教程开始有深度地覆盖 Scala 语言。
- 第 5 章，“基本类型和操作”，覆盖了 Scala 的基本类型，它们的字面量，可执行的操作，优先级和关联性是如何工作的，还有什么是富包装器。
- 第 6 章，“函数式对象”，进入了 Scala 面向对象特征的更深层次，使用函数式（即不可变）有理数作为例子。

- 第 7 章，“内建控制结构”，演示了如何使用 Scala 的内建控制结构，如 `if`，`while`，`for`，`try` 和 `match`。
- 第 8 章，“函数和闭包”，深度讨论了函数式语言的基础建设模块，函数。
- 第 9 章，“控制抽象”，演示了如何通过定义你自己的控制抽象来增强 Scala 的基本控制结构。
- 第 10 章，“组合与继承”，讨论了更多 Scala 对面向对象编程的支持。这个话题并不像在第 4 章中那样基础，但它们在实践中经常出现。
- 第 11 章，“Scala 的层级”，解释了 Scala 的继承层级并讨论了其全体方法及底层类型。
- 第 12 章，“特质”（trait），演示了 Scala 在混入组成（mixin composition）中的机制。本章演示了特质如何工作，描述了通常的用法，还解释了为什么特质改善了传统的多继承。
- 第 13 章，“包和引用”，讨论了大项目编程中的事务，包括顶层包，引用语句，还有访问控制修饰符如，`protected` 和 `private`。
- 第 14 章，“断言和单元测试”，演示了 Scala 的断言机制并大致学习了各种可以为 Scala 编写测试的工具。
- 第 15 章，“样本类和模式匹配”，介绍了样本类和模式匹配，这对你在编写正规的非封装的数据结构时用到的工具，尤其对树型递归数据很有用。
- 第 16 章，“使用列表”，详细解释了列表。它或许是在 Scala 程序中最常用到的数据结构。
- 第 17 章，“集合类型”，演示了如何使用基础的 Scala 集合类型，如：.列表、数组、元组集（tuple）及映射表。
- 第 18 章，“有状态的对象”，解释了什么是有状态（即可变）的对象，Scala 提供的语法层面表达它们的术语。本章包括了一个在离散事件模拟上的案例研究，用来演示一些有状态对象的动作。
- 第 19 章，“类型参数化”，用具体的例子：纯函数队列类的设计，解释了第 13 章介绍过的一些信息隐藏技术。本章建立了关于各种类型参数的描述，以及它如何与信息隐藏实现交互。
- 第 20 章，“抽象成员”，描述了所有 Scala 支持的抽象成员。能够声明为抽象的不仅是方法，还包括字段和类型。
- 第 21 章，“隐式转换和参数”，描述了这两个特性有助于程序员忽略掉源码中那些能由编译器推导出来的繁琐的细节的特性。
- 第 22 章，“实现列表”，描述了 `List` 类的实现。弄明白在 Scala 里面列表如何工作是很重要的，而且，实现本身展示了若干 Scala 特性的应用。
- 第 23 章，“重访 For 表达式”，解释了 `for` 表达式是如何翻译成对 `map`、`flatMap`、`filter`

和 `foreach` 的访问。

- 第 24 章（Extractors），“抽取器”，展示了如何使用模式匹配任何类，而不仅仅是用例类。
- 第 25 章，“注解”，演示了如何通过注解使用语言的扩展部分。本章示范了若干标准注解，也示范了如何建立你自己的注解。
- 第 26 章，“使用 XML”，演示了在 Scala 中如何处理 XML。包括 XML 的创建、解析，以及解析之后的处理等一系列惯用方式。
- 第 27 章，“使用对象的模块化编程”，演示了说 Scala 的对象已足够丰富的原因，从而消除了分离式模块系统的使用需求。
- 第 28 章，“对象相等性”，指出若干在编写 `equals` 方法时要考虑的情况。说明了若干应避免的误区。
- 第 29 章，“结合 Scala 和 Java”，描述了若干在同一个项目中捆绑使用 Java 和 Scala 时会碰到的状况，以及建议的解决方法。
- 第 30 章，“Actor 和并发”，展示如何使用 Scala 的 actor 并发库。尽管你使用 Java 平台的同步原语和来自于 Scala 程序的库，但 actor 能帮你避免死锁和资源竞争这些影响着传统并发的问题。
- 第 31 章，“连结符解析”，演示了如何使用 Scala 的解析器连结符库来创建解析器。
- 第 32 章，“GUI 编程”，展示了使用 Scala 库简化基于 Swing 的 GUI 编程的快速旅程。
- 第 33 章，“`SCells` 试算表”，通过展示一个完整的试算表的实现，集中演示了 Scala 的一切。

资源

在 Scala 的主网站，*http://www.scala-lang.org*，你能找到 Scala 最近的发布版和文档、社区资源的链接。Scala 资源链接更全的页面，请访问本书网站：*http://booksites.artima.com/programming_in_scala*。与本书其他读者交流，请访问：*http://www.artima.com/forums/forum.jsp?forum=282*。

源码

你可以从本书的网站下载包含本书源码的 ZIP 文件，它是以 Apache 2.0 开源许可方式发布的：*http://booksites.artima.com/programming_in_scala*。

勘误

尽管本书已复审检查多次，仍不可避免错误的发生。要查阅本书的勘误列表，请访问：*http://booksites.artima.com/programming_in_scala/errata*。如果你发现错误，也请在上述网址报告，这样我们可以确保在本书将来的印刷或发行版中修正它。

目录概览

目录