MANNING

来自资深实践者的深刻洞见　Scala 之父作序推荐

深入理解 Scala

Scala IN DEPTH

〔美〕Joshua D. Suereth　著
杨云　译

人民邮电出版社
POSTS & TELECOM PRESS

MANNING

深入理解 Scala

〔美〕Joshua D. Suereth 著
杨云 译

人民邮电出版社
北京

图书在版编目（CIP）数据

深入理解Scala / （美）苏瑞茨（Suereth, J. D.）著；
杨云译. -- 北京 : 人民邮电出版社, 2015.1
ISBN 978-7-115-36554-5

Ⅰ. ①深… Ⅱ. ①苏… ②杨… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2014)第251097号

内 容 提 要

Scala 是一种多范式的编程语言，它既支持面向对象编程，也支持函数式编程的各种特性。

本书深入探讨了 Scala 里几个较为复杂的领域，包括类型系统的高阶内容、隐式转换、特质的组合技巧、集合、Actor、函数式编程的范畴论等，而且不是干巴巴地讲述语言和库的概念。本书充满各种实用的建议和最佳实践，可以来帮助读者学习怎样把 Scala 里较少被掌握的部分应用到工作中。

本书不是 Scala 的入门级教程，而是适合有经验的 Scala 程序员向专家水平进阶的参考读物。本书适合想要了解 Scala 语言的底层机制和技术细节的读者阅读参考。

◆ 著 [美] Joshua D. Suereth
译 杨 云
责任编辑 陈冀康
责任印制 张佳莹 彭志环
◆ 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：18
字数：385 千字 2015 年 1 月第 1 版
印数：1 – 3 000 册 2015 年 1 月北京第 1 次印刷
著作权合同登记号 图字：01-2012-4605 号

定价：59.00 元

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

前言

Joshua Suereth 似乎是我所知的最全面的程序员之一，熟悉各种编程语言和技术。他是高性能系统、构建工具、类型理论以及其他很多领域的专家。同时他在教学方面也极有才能。这些特质的结合使《深入理解 Scala》成为一本与众不同的书。

这本书深入探讨了 Scala 里几个较为复杂的领域，包括类型系统的高阶内容、隐式转换、特质的组合技巧、集合、Actor、函数式编程的范畴论等，而且不是干巴巴地讲述语言和库的概念。这本书里充满各种实用的建议和最佳实践来帮助读者学习怎样把 Scala 里较少被掌握的部分应用到工作中。书中的解释和例子展现出 Joshua 用 Scala 构建大型可伸缩系统的丰富经验。

《深入理解 Scala》不是一本入门级的书，而是有经验的 Scala 程序员向专家水平进阶的参考读物。书中所教的都是在构建灵活且类型安全的库时非常好用的技巧。其中很多都是“隐藏在民间”的技巧，第一次在本书中落在纸面上。

我还特别为另一件事而高兴：这本书填补了一个空白——它把正式的 Scala 语言规范中的关键部分解释给了不是专门研究语言的程序员们。Scala 是少数几个有正式的语言规范的编程语言之一。语言规范主要包含高度程序化的文本和数学公式，所以不是所有人都愿意一读。Joshua 的书在解释语言规范里的概念时做到了既权威又可理解。

Martin Odersky
Scala 语言的创始人
首席程序员
RESEARCH GROUP，EPFL

关于本书

《深入理解 Scala》是一本使用 Scala 的实用指导书，深入地探讨了一些必要的主题。这本书撷取了入门书籍忽略的主题，使读者可以写出符合 Scala 惯用法的代码，理解使用高级语言特性时的取舍。尤其是，这本书详细讲解了 Scala 的隐式转换和类型系统，然后才讨论怎样使用它们来极大地简化开发。本书提倡“混合风格”的 Scala 编程，结合使用不同编程范式以达成更大的目标。

谁应当阅读本书

本书适合新或中等水平的 Scala 开发人员来提升他们的开发技能。这本书在覆盖 Scala 中一些非常高阶的概念的同时，也尽量考虑了学习 Scala 的新手的需求。

本书的读者应当学过 Java 或别的面向对象语言。具有 Scala 经验可有助于读者阅读本书，但这种经验并不是必需的。本书覆盖 Scala2.7.x 到 Scala2.9.x 版本。

路线图

本书从讨论 Scala 的“禅”，也就是 Scala 的设计哲学开始——Scala 是把多种概念混合，达成 1+1 大于 2 的效果。 特别讨论了三组对立的概念：静态类型与表达力，函数式编程与面向对象编程，强大的语言特性与极简的 Java 集成。

第 2 章讨论 Scala 的核心规则。这些规则是每个 Scala 程序员都应当了解并在日常开发中使用的。这一章的内容适用每个 Scala 程序员，覆盖了那些使 Scala 如此卓越的基本内容。

第 3 章是关于编码风格及相关问题的插话。Scala 带来了一些新东西，任何 Scala 编码风格指导都应当反映这些内容。有些来自流行的编程语言如 Ruby 和 Java 的常用规则实际上会妨碍你写出好的 Scala 代码。

第 4 章覆盖了 Scala 的 mixin 继承带来的面向对象设计的新问题。每个 Scala 程序员都感兴趣的一个主题是早期初始化，这个主题在别的书里很少谈到。

谈完面向对象后，本书接着讨论隐式转换系统。在第 5 章，我们不是简单地讨论最佳实践，而是深入理解 Scala 的隐式转换机制。这章对于所有想写出有表达力的库和代码的 Scala 程序员都是必读章节。

第 6 章专注于 Scala 的类型系统。这章讨论 Scala 里“类型”的各种表现形式，以及如何利用类型系统来强制约束。这章接着进入到对高阶类型的讨论，然后深入到存在类型的讨论作为结束。

第 7 章讨论 Scala 语言最高阶的运用模式——类型系统和隐式转换的交叉使用。这种交叉使用带来了很多有趣而强大的抽象，典型的就是类型类模式。

讨论完 Scala 最高阶的部分后，在第 8 章，我们讨论 Scala 的集合库。内容包括 Scala 集合库的设计和性能，以及如何利用强大的类型机制。

第 9 章开始讨论 Scala 的 Actor。Actor 是一种并发机制，当正确使用时，可以提供极大的吞吐量和并行性。这一章从讨论基于 Actor 的设计入手，以展示 Akka actors 库默认提供的最佳实践作为结束。

第 10 章内容覆盖 Java 与 Scala 的集成。虽然 Scala 与 Java 的兼容性好于 JVM 上的大部分其他语言，但两者还是存在一些特性上的不匹配。这些不匹配的角落里容易发生 Scala-Java 集成问题，这章提供了几个简单规则来帮助避免这些问题。

第 11 章拿来了范畴论里的概念并使之实用化。在纯函数式编程里，很多来自范畴论的概念已经被应用到代码里。这些概念有点类似面向对象的设计模式，但是要抽象得多。虽然这些概念的名字挺吓人——这在数学上很常见——但它们其实极有实用价值。不讨论这些抽象概念就没法完整讲述函数式编程，本书尽力使这些概念更现实（不那么抽象）。

代码下载和约定

为了使代码显示区别于正文，本书中的所有代码都用等宽字体显示。很多代码清单里有一些用来指出关键点的注解。我已经尽量通过增加换行和使用缩进来调整代码格式，使注解能够在页面空间里显示完整，但偶尔还会有一些很长的语句不得不用换行连接符号。

书中所有例子的源代码可以在 www.manning.com/ScalainDepth 和 https://github.com/jsuereth/scala-in-depth-source 获取。要运行代码示例，读者需要安装 Scala 以及（可选的）SBT 构建工具。

全书包含很多代码示例，较长的代码有明显的清单标题，较短的代码直接显示在文本行之间。

作者在线

购买本书的同时，您获得了免费访问 Manning 出版社运营的私有网络论坛的权利，您可以在那里对本书做评论、询问技术问题、向作者和其他用户寻求帮助。要访问和订阅论坛，请在 Web 浏览器里打开地址 www.manning.com/ScalainDepth。这个页面提供了在注

册后如何使用论坛，有哪些可用的帮助，以及论坛的指导规则等信息。

Manning 对读者的承诺仅是提供一个能够让读者之间、读者和作者之间进行有意义的对话的场所，并不承诺作者在论坛上的投入度。作者在论坛上的投入是义务的（无偿的）。我们建议您向作者提出一些有挑战的问题，以免他失去兴趣。

只要书还在印刷，作者在线论坛以及发生过的讨论将保持可访问。

作者简介

Josh Suereth 是 Typesafe 公司（Scala 背后的公司）的一名高级软件工程师。从 2007 年了解到 Scala 这门美丽的语言后，他就成了 Scala 的狂热分子。他在 2004 年开始了软件开发者的职业生涯，一开始先在 C++、STL 和 Boost 上磨砺技能。当时 Java 正在狂热传播，他的兴趣迁移到 Web 部署的基于 Java 的解决方案来帮助健康部门发现疾病的爆发。

他在 2007 年将 Scala 引入公司的代码库，然后迅速染上了 Scala 狂热症，他对 Scala IDE、maven-scala-plugin 和 Scala 本身都做出了贡献。今日，Josh 已经是好几个开源 Scala 项目的作者，包括 Scala 自动化资源管理库和 PGP sbt 插件，他还是 Scala 生态系统的一些关键组件的贡献者，比如说 maven-scala-plugin。他现在就职于 Typesafe 公司，做包括从构建 MSI 到侦测性能问题等各种事情。

Josh 定期地通过文章和演讲分享他的专业知识。他喜欢在海滩边散步还有喝黑啤。

致谢

在这本书出版的过程中得到了很多人的帮助。我将尽量全部列出，但我确信帮助我的人太多，实非我的小小脑袋能够全部记住。这本书让我知道我有非常多高水准的朋友、同事，还有家庭。

最应当感谢的是我的妻子和孩子，他们不得不忍受一个一直躲在角落里写书，该搭手帮忙的时候也不出来的丈夫和父亲。没有任何作者能够在没有家庭支持的情况下写一本书，我也不例外。

接着我要感谢 Manning 出版社和工作人员为使我成为一个真正的作者所做的事。他们不仅做了审阅、排版的工作，还帮助我提高有助于清晰沟通的写作技巧。对整个出版团队，我感激不尽，尤其要感谢 Katherine Osborne 容忍我的不断拖稿，宾夕法尼亚州的荷兰语式的语句和经常的拼写错误。Katherine 非常注意搜集对本书的读者之声，那些读过 MEAP（早期发行版）的读者应该能注意到本书的进步。

下一个应该感谢的群体是帮助我提高技术材料和文字写作的 Scala 专家和非专家们。在我写本书的差不多同时期，Tim Perret 正在为 Manning 出版社写《Lift in Action》。和 Tim 的讨论既有益又激励。对我不幸的是他先写完了他那本书。Justin Wick 是本书很多内容的审阅者和协作者，并最终帮助我使这本书所针对的读者范围比我开始时所想的大得多。他同时也是付印前最后一个审阅手稿和代码的人。Adriaan Moor，一如既往地在讨论类型系统和隐式解析时指出我所有的错误，使讨论既实用又正确。Eric Weinberg 是我的老同事，帮助提供了本书中给非 Scala 程序员的指导。Viktor Klang 审阅了“Actors”章节（和整本书）并提供了改进建议。也要感谢 Martin Odersky 的支持，感谢他给本书写序。感谢 Josh Cough，他是一个能在需要的时候跟我激荡创意的家伙，还有 Peter Simany，感谢他用电子邮件发给我对于整本书的非常详尽、细致、完整、极好的评审意见。

Manning 还联系了以下这些评审者，他们在不同阶段阅读了本书的手稿，我想在这里感谢他们无价的洞见和评论：John C. Tyler、Orhan Alkan、Michael Nash、John Griffin、

Jeroen Benckhuijsen、David Biesack、Lutz Hankewitz、Oleksandr Alesinskyy、Cheryl Jerozal、Edmon Begoli、Ramnivas Laddad、Marco Ughetti、Marcus Kazmierczak、Ted Neward、Eric Weinberg、Dave Pawson、Patrick Steger、Paul Stusiak、Mark Thomas、David Dossot、Tariq Ahmed、Ken McDonald、Mark Needham 和 James Hatheway。

最后，我要感谢所有 MEAP 版的评审者，他们给予我非常有价值的反馈，感谢他们的支持和本书付印前所得到的审阅意见。他们不得不忍受大量的拼写错误，使这本书由粗糙的初稿转变为最终的版本。

自序

2010 年秋，Manning 出版社的 Michael Stephens 联系我写一本关于 Scala 的书。当时我就职于一家主营虚拟化/安全方面的小创业公司，期间我学习并在工作代码中应用了 Scala 语言。在 Michael 和我的第一次会谈中我们讨论了 Scala 的生态系统和什么样的书对社区最有价值。

我认为 Scala 需要一本“实用 Scala”这样的书来指导那些 Scala 新人。Scala 是一种美丽的语言，但它一下子带给程序员很多新概念。我曾看着 Scala 社区慢慢地识别出最佳实践，慢慢地发展出完全属于“Scala”的编码风格，但我一直不确定我是不是写这本书的合适人选。当各种条件逐渐齐备——我对这个主题充满激情，有足够的自由时间来做研究，有社区牛人的支持——于是我决定写这本书。

在写作过程中我学到了很多，写这本书耗时如此之久的原因之一在于 Scala 的不断进化和不断涌现的新最佳实践。另一个原因则是我意识到自己的知识在 Scala 的某些领域非常不足。我想在这里告诉所有有志写书的作者，写书的过程能够让你成为专家。 在开始写书前你可能觉得自己本来就是专家，但我要告诉你真正的专业技能是在教导他人，是在尽力把复杂的概念清晰地解释给你的读者的过程中伴随着血、汗和泪成长起来的。

如果没有一直支持我的妻子、伟大的出版社、了不起的 Scala 程序员社区和愿意阅读我不同阶段的手稿，指出拼写错误，给出改进建议的读者，我绝不可能完成写这本书的旅程。感谢你们让这本书远超我独自一人能够达到的水平。

目录

第1章 Scala——一种混合式编程语言

本章包括的内容：

- 简要介绍 Scala 语言
- 剖析 Scala 语言的设计思想

Scala 是一种将其他编程语言中的多种技巧融合为一的语言。Scala 尝试跨越多种不同类型的语言，给开发者提供面向对象编程、函数式编程、富有表达力的语法、静态强类型和丰富的泛型等特性，而且全部架设于 Java 虚拟机之上。因此开发者使用 Scala 时可以继续使用原本熟悉的某种编程特性，但要发挥 Scala 的强大能力则需要结合使用这些有时候相互抵触的概念和特性，建立一种平衡的和谐。Scala 对开发者的真正解放之处在于让开发者可以随意使用最适合手头上的问题的编程范式。如果当前的任务更适合用命令式的设计实现，没什么规定禁止你写命令式的代码，如果函数式编程和不可变性（immutability）更符合需要，那程序员也可以尽管用。更重要的是，面对有多种不同需求的问题领域（problem domain），你可以在一个解决方案的不同部分采用不同的编程方法。

1.1 Scala 的设计哲学

为了理解 Scala 的哲学，我们需要理解产生 Scala 的环境：Java 生态圈 Java（TM）语言在 1995 年左右进入计算机科学领域，产生了巨大的影响。Java 和运行 Java 的虚

拟机开始慢慢地变革了我们的编程方法。在那时候，C++正如日中天，开发者正在从纯 C 风格的编程转而开始学习如何有效地使用面向对象编程方法。尽管 C++有很多的优点，但它也有一些痛点，比如难以分发库（distributing libraries）以及其面向对象实现的复杂度。

Java 语言通过提供限制了部分能力的面向对象特性和使用 Java 虚拟机，同时解决了这两个痛点。Java 虚拟机（JVM）允许代码在一个平台上编写和编译后，几乎不费多大劲就能分发到其他的平台上。尽管跨平台问题并没有就此消失，但是跨平台编程的成本极大地降低了。

随着时间的推移，JVM 的执行效率越来越高，同时 Java 社区不断成长。HotSpot（TM）优化技术被发明出来，这样就可以先探测运行环境再进行针对性的代码优化。虽然这使得 JVM 启动速度变慢，但是之后的运行时性能则变得很适合于运行服务器之类的应用。尽管最初并非为企业服务器领域设计的，JVM 开始在此领域大行其道。于是人们开始尝试简化企业服务器应用开发，Enterprise Java Beans（TM）和较新的 Spring Application Framework（TM）出现，帮助程序员更好地利用 JVM 的能力。Java 社区发生了爆炸式的成长，创造出成百万的易于使用的库。“只要你能想到的，基本都能找到 Java 库”成为一个职场口号。Java 语言持续地缓慢进化，努力维持住其社区。

与此同时，部分开发者开始延展他们的羽翼，触及了 Java 本身设计上的局限之处。Java 简化了一些（编程元素），而社区中的部分成员需要增加一些复杂的，但是可控的元素。第二波创造 JVM 语言的浪潮掀起并持续至今。Groovy、JRuby、Clojure 和 Scala 等语言开始将 Java 程序员带入一个新时代。我们不再局限于一种语言，而是可以有多种选择，每一种语言都有不同的优点和弱点。Scala 是其中较为流行的一种。

Scala 语言创造者 Martin Odersky 是 javac 编译器的作者，也是他将泛型引入 Java 语言中。Scala 语言衍生自 Funnel 语言。Funnel 语言尝试将函数式编程和 Petri 网结合起来，而 Scala 的预期目标则是将面向对象、函数式编程和强大的类型系统结合起来，同时仍然要能写出优雅、简洁的代码。将以上多种概念混合的目的是创造出一种既能让程序员真正用起来，同时又能用来研究新的编程范式的语言。事实上它取得了巨大的成功——它作为一种可行的有竞争力的语言已经开始被产业界采用。

要想掌握 Scala，你需要理解多种混合在一起的概念。Scala 试图将以下三组对立的思想融合到一种语言中。

- 函数式编程和面向对象编程。
- 富有表达力的语法和静态类型。
- 高级的语言特性同时保持与 Java 的高度集成。

我们来看一下这些特性是如何融合在一起的。先从函数式编程和面向对象编程概念开始。

1.2 当函数式编程遇见面向对象

函数式编程和面向对象编程是软件开发的两种不同途径。函数式编程并非什么新概念，在现代开发者的开发工具箱里也绝非是什么天外来客。我们将通过 Java 生态圈里的例子来展示这一点，主要来看 Spring Application framework 和 Google Collections 库。这两个库都在 Java 的面向对象基础上融合了函数式的概念，而如果我们把它们翻译成 Scala，则会优雅得多。在深入之前，我们需要先理解面向对象编程和函数式编程这两个术语的含义。

面向对象编程是一种自顶向下的程序设计方法。用面向对象方法构造软件时，我们将代码以名词（对象）做切割，每个对象有某种形式的标识符（self/this）、行为（方法）、和状态（成员变量）。识别出名词并且定义出它们的行为后，再定义出名词之间的交互。实现交互时存在一个问题，就是这些交互必须放在其中一个对象中（而不能独立存在）。现代面向对象设计倾向于定义出“服务类”，将操作多个领域对象的方法集合放在里面。这些服务类，虽然也是对象，但通常不具有独立状态，也没有与它们所操作的对象无关的独立行为。

函数式编程方法通过组合和应用函数来构造软件。函数式编程倾向于将软件分解为其需要执行的行为或操作，而且通常采用自底向上的方法。函数式编程中的函数概念具有一定的数学上的含义，纯粹是对输入进行操作，产生结果。所有变量都被认为是不可变的。函数式编程中对不变性的强调有助于编写并发程序。函数式编程试图将副作用推迟到尽可能晚。从某种意义上说，消除副作用使得对程序进行推理（reasoning）变得较为容易。函数式编程还提供了非常强大的对事物进行抽象和组合的能力。

表 1.1 面向对象和函数式编程的一般特点

面向对象编程	函数式编程
对象的组合（名词）	函数的组合（动词）
封装的有状态的交互（Encapsulated stateful interaction）	推迟副作用
迭代算法	递归算法和 Continuations
命令流	延迟计算
-	模式匹配

函数式编程和面向对象编程从不同的视角看待软件。这种视角上的差异使得它们非常互补。面向对象可以处理名词而函数式编程能够处理动词。其实近年来很多 Java 程序员已经开始转向这一策略（分离名词和动词）。EJB 规范将软件切分为用来容纳行为的 Session bean 和用来为系统中的名词建模的 Entity bean。无状态 Session bean 看上去就更像是函数式代码的集合了（尽管欠缺了很多函数式代码有

用的特性）。

这种朝函数式风格方向的推动远不止 EJB 规范。Spring 框架的模板类（Template classes）就是一种非常函数式的风格，而 Google Collections 库在设计上就非常的函数式。我们先来看一下这些通用的 Java 库，然后看看 Scala 的函数式和混合面向对象编程能怎样增强这些 API。

1.2.1 重新发现函数式概念

很多现代 API 设计时都融入了函数式编程的好东西而又不称自己是函数式编程。对于 Java 来说，像 Google Collections 和 Spring 应用框架以 Java 库的形式使 Java 程序员也能接触到流行的函数式编程概念。Scala 更进一步，将函数式编程直接融合到了语言里。我们来将流行的 Spring 框架中的 JdbcTemplate 类简单地翻译成 Scala，看看它在 Scala 下会是什么样子。

清单 1.1 Spring 的 JdbcTemplate 类上的查询方法

```
public interface JdbcTemplate {
  List query(PreparedStatementCreator psc,                    ❶查询对象列表
             RowMapper rowMapper)
  ...
}
```

现在，来直译一下，我们把接口转换为有相同方法的特质（trait）。

清单 1.2 查询方法的 Scala 直译

```
trait JdbcTemplate {
  def query(psc : PreparedStatementCreator,
                  rowMapper : RowMapper) : List[_]
}
```

简单的直译也很有意思，不过它还是非常的 Java。我们现在来深挖一下，特别看看 PreparedStatementCreator 和 RowMapper 接口。

清单 1.3 PreparedStatementCreator 接口

```
public interface PreparedStatementCreator {
  PreparedStatement createPreparedStatement(Connection con)   ❶定义的唯一一个
    throws SQLException;                                        方法
}
```

PreparedStatementCreator 接口只有一个方法。这个方法接受 JDBC 连接，返回 PreparedStatement. RowMapper 接口看上去也差不多。