MANNING

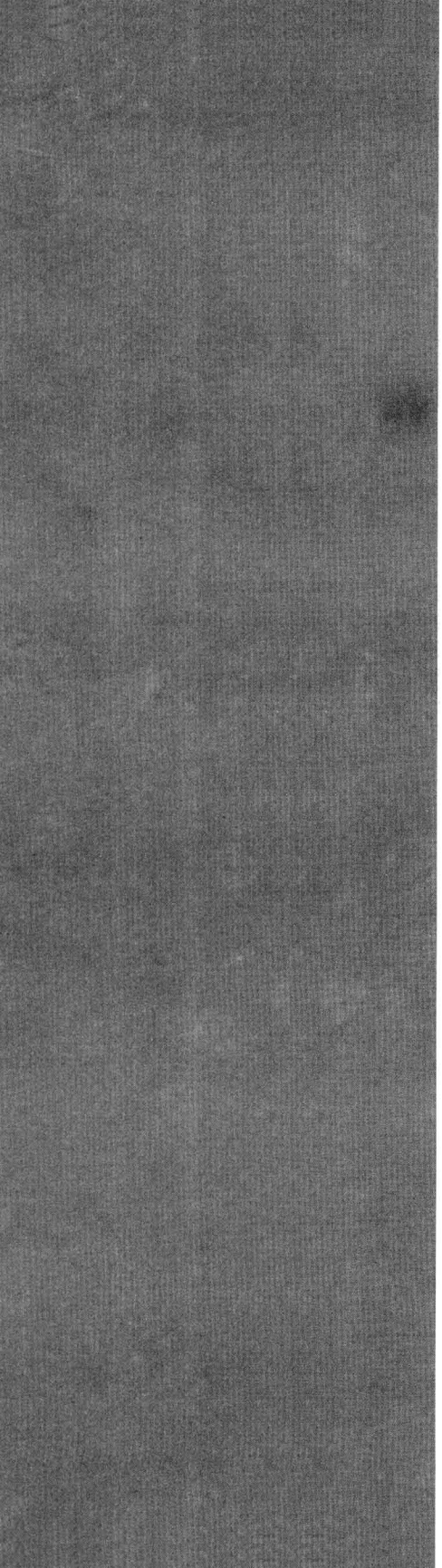
知名程序员 Steve Yegge 作序推荐

Clojure 编程乐趣

THE Joy OF Clojure

〔美〕 Michael Fogus
Chris Houser 著
郑晔 译

人民邮电出版社
POSTS & TELECOM PRESS



MANNING

Clojure 编程乐趣

THE Joy OF Clojure

〔美〕Michael Fogus
Chris Houser 著
郑晔 译

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

Clojure编程乐趣 / (美) 福格斯 (Fogus, M.), (美) 豪泽 (Houser, C.) 著 ; 郑晔译. -- 北京 : 人民邮电出版社, 2013.11
ISBN 978-7-115-31949-4

Ⅰ. ①C… Ⅱ. ①福… ②豪… ③郑… Ⅲ. ①程序语言－语言设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2013)第099948号

版权声明

◆ 著　[美] Michael Fogus　Chris Houser
译　郑　晔
责任编辑　陈冀康
责任印制　程彦红　杨林杰

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：21.5
字数：448 千字　2013 年 11 月第 1 版
印数：1－3 000 册　2013 年 11 月河北第 1 次印刷

著作权合同登记号　图字：01-2012-6450 号

定价：59.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

内容提要

Clojure 是一门 Lisp 方言。它通过函数式编程技术，直接支持并发软件开发，得到众多开发人员的欢迎。

本书并非 Clojure 初学指南，也不是一本 Clojure 的编程操作手册，而是通过对 Clojure 详尽地探究，教授函数式的程序设计方式，帮助读者理解和体会 Clojure 编程的乐趣，进而开发出优美的软件。

全书分为 5 个部分共 13 章。第 1 部分是基础，包括第 1 章到第 3 章，从 Clojure 背后的思想开始，介绍了 Clojure 的基础知识，并带领读者初步尝试 Clojure 编程。第 2 部分包括第 4 章和第 5 章，介绍了 Clojure 的各种数据类型。第 3 部分是第 6 章和第 7 章，介绍了函数式编程的特性。第 4 部分包括第 8 章到第 11 章，分别介绍了宏、组合数据域代码、Clojure 对 Java 的调用，以及并发编程等较为高级的话题。第 5 部分为第 12 章和第 13 章，探讨了 Clojure 的性能问题及其带给我们的思考。

本书适合有一定基础的 Clojure 程序员阅读，进而掌握函数编程的思考方法和程序设计方法，也可以作为读者学习函数式编程的参考资料。

序

本书作者选择了一种极具野心且颇为进取的方式教授 Clojure。当听闻有人进行“疾风式”教学，你会做何感想？喔，感觉就像有人马上就要被吹走一样……我只是说，这不是通常理解的疾风。本书根本没打算成为程序设计的第一本书，即便是第一本 Clojure 书也不合适。作者假设你是个无畏的家伙，重要的是，你还配备了搜索引擎。浏览书中例子时，手边最好有 Google。在这场 Clojure 旋风之旅中，作者带着我们飞快地领略了函数式编程和工业程序设计的经典基础，偶尔会让人觉得这简直是场五级热带风暴。你会学得飞快！

我们的产业，甚至整个的程序设计社区，都是时尚驱动的，以至于从纽约到巴黎高级服装设计师都局促不安。我们臣服于时尚。时尚决定着学校里教授怎样的程序设计语言，语言雇主招什么样的人，书架上摆什么书。天真的局外人或许以为语言的质量多少会有点影响，至少有那么一点点，但在现实世界里，时尚压倒一切。

所以，突然有一门 Lisp 方言流行起来，没有人会比我更为惊讶了。Clojure 仅仅面世三年[1]，却以数十年间前所未见的速度赢得关注。它甚至还没来得及有个“杀手级应用”，就像浏览器将 JavaScript 推到了闪光灯下，Rails 促进了 Ruby 那样。或者说，也许 Clojure 的杀手级应用就是 JVM 本身。所有人对 Java 语言都忍无可忍，但有一点却可以理解，我们并不打算放弃在 Java 虚拟机及其能力上的投资：程序库、配置、监控，以及所有各种完全有效的理由，都支持我们继续用下去。

对于使用 JVM 或是.NET 的我们而言，Clojure 感觉就像一个小奇迹。它的确是一门不可思议的高质量语言，实际上，我已经开始认为它是我见过的最好的程序设计语言了——然而不知怎么它就流行起来了。这简直是个魔法！它重燃了我对这个行业未来生产力整体提升的希望。或许，我们只是想摆脱困境，回到每个项目都像全新启动一样，没有遗留系统，如同 Java 的往日荣光一般。

[1] 译注：Clojure 诞生于 2008 年，而本书英文版出版于 2011 年。

在 Clojure 对生产环境的支持上，还有许多问题悬而未决，特别是相关的工具链。对于一门新语言，这是很正常的，也在预期之中。但是，Clojrue 让我们看到了希望，如此优美实用的设计原则，似乎每个人都会为之雀跃。我的确如此！自从 15 年前 Java 降临，我已许久未曾体会到新语言带来的乐趣了。有许多语言觊觎 JVM 的王座，承诺将 Java 平台带至前所未有的新境界。时至今日，没有一种语言能将表达性、工业强度、性能同简单的乐趣正确地融合在一起。

在我看来，也许正是 Clojure 中“乐趣”的部分使之流行起来。

从某种意义上说，我认为所有这些都无可避免。Lisp——直接以树形式编写代码的记法——这种理念已经是一次又一次得到了时间的验证。人们尝试过各种疯狂的做法：用 XML 格式，不透明的二进制，甚至用笨拙的代码生成器编写代码。但这种人造的“拜占庭帝国”总会年久失修，或为自身所累而坍塌崩溃，然而 Lisp 却历经岁月，依然简单、优雅、纯净。我们需要以一种现代的方式回到这条路上来。Rich Hickey 做到了，他用 Clojure 带我们回来了。

本书或许只是让 Clojure 有趣起来，对您如此，对我们也如此！

STEVE YEGGE
GOOGLE
steve-yegge.blogspot.com
《A Programmer's Rantings》作者

前言

想要完全理解 Clojure，就应该品味一番 Paul Graham 的文章《拒绝平庸》，它让人有机会一窥其公司 Viaweb 在 1998 年被雅虎收购之前的内部状态。虽然纵览创业文化很有趣，但这篇文章真正令人难以忘怀的部分是，Viaweb 怎样用 Lisp 程序设计语言赢得竞争优势。一门五十多岁的程序设计语言怎么能为 Viaweb 带来超越其竞争对手的优势呢？它的竞争对手肯定用的是更现代的企业级技术。这里无意重复文章中的确切内容，Graham 确实给出了一个令人信服的例子，充分展现了 Lisp 在促进更加敏捷程序设计环境方面的能力。

Clojure 是一门 Lisp 方言，通过函数式编程技术，直接支持并发软件开发。它类似于《拒绝平庸》中所描述的 Lisp，提供了一个有益于敏捷性的环境。Clojure 以一种许多流行的程序设计语言无法复制的方式促进了敏捷性的发展。许多程序设计语言受困于下列事情的全部或大多数：

- 啰嗦。
- 无可避免的样板代码（boilerplate）。
- 漫长的思考-编码-反馈循环。
- 偶然复杂性。
- 难于扩展。
- 缺乏对关键程序设计范式的支持。

相比之下，Clojure 将能力和实践融合在一起，缩短了开发周期。但是，Clojure 的益处并没有止步于其敏捷性——正如一篇文章所强烈地召唤一般，《多核已成新热点（Multicore is the new hot topic）》（Mache Creeger in ACM Queue, vol. 3, no. 7）。

虽然多核处理器概念本身并不新，但其重要性正日益赢得更多的关注。时至今日，并发和并行程序设计已然无法回避，而凭借不断加快的处理器浪潮赢得更好性能的往日荣光已经一去不回。好吧，它正逐步放缓，直至停止，Clojure 恰逢其时的出现，给了我们很大的帮助。

Clojure 将函数式编程与宿主共生以一种独特的方式融合在一起，这种宿主共生是对宿主平台的拥抱和直接的支持，这里宿主平台指的是 Java 虚拟机。此外，Clojure 简化有时甚至消除了需要协调的状态改变所包含的复杂性，这也将 Clojure 定位成一种重要的、勇往直前的语言。最终，所有的软件开发人员都会将处理这些问题视为理所当然，对于 Clojure 的研习、理解以及最终的运用都是征服这些问题的必经之路。从软件事务性内存，到惰性，再到不变性，本书将引导你理解 Clojure 这些主题背后的“为什么”，当然，还包括了“怎么做”。

我们愿做你的向导，帮你深入理解 Clojure 的乐趣，因为我们相信，这种艺术终将成为新时代软件开发的序曲。

致谢

本书作者要共同感谢 Rich Hickey，Clojure 之父，带给世人其深思熟虑之作，推动了语言设计进一步发展。没有他的辛勤工作、投入及视野，本书便不复存在。

我们还要感谢年轻的 Clojure 社区里那些充满智慧的人，包括但不限于 Stuart Halloway、David Edgar Liebke、Christophe Grand、Chas Emerick、Meikel Brandmeyer、Brian Carper、Bradford Cross、Sean Devlin、Tom Faulhaber、Stephen Gilardi、Phil Hagelberg、Konrad Hinsen、George Jahad、David Miller、David Nolen、Laurent Petit 和 Stuart Sierra。之后，我们要感谢一些早期阅读者，他们经过深思熟虑，给出了周密的反馈，包括 Jürgen Hötzel、Robert “Uncle Bob” Martin、Grant Michaels、Mangala Sadhu Sangeet Singh Khalsa 和 Sam Aaron。最后，我们要特别感谢 Steve Yegge 同意为本书写序，许多年来，他一直激励着我们。

在本书编写的不同阶段，Manning 都会把草稿发出去做评审，我们感谢下列评审者的无价建议：Art Gittleman、Stuart Caborn、Jeff Sapp、Josh Heyer、Dave Pawson、Andrew Oswald、Federico Tomassetti、Matt Revelle、Rob Friesel、David Liebke、Pratik Patel、Phil Hagelberg、Rich Hickey、Andy Dingley、Baishampayan Ghose、Chas Emerick、John D'Emic 和 Philipp K. Janert。

还要感谢 Manning 团队给予的指导和支持，从发行人 Marjan Bace 开始、副发行人 Michael Stephens、我们的开发编辑 Susan Harkins 以及生产团队 Nicholas Chase、Benjamin Berg、Katie Tennant、Dottie Marsico 和 Mary Piergies。还要再次感谢 Christophe Grand，在生产期间给予脚本最终的技术审校。

FOGUS

我要感谢我美丽的太太 Yuki，在写作本书期间，她给予了我无限的耐心。没有她，我恐怕无法坚持下来。我还亏欠 Chris Houser 许多，我的合作者和朋友，他教会了我许多 Clojure 的东西，我从未想到可以这么用。我要感谢 Larry Albright 博士，他把 Lisp 介绍给我，稍后，Russel E. Kacher 博士，他激发了我对学习、好奇心以及沉思的激情。此

外，我要感谢 National Capital Area Clojure Users Group 的组织者——Matthew Courtney、Russ Olsen 和 Gray Herter，他们提供了一个地方给 DC 地区的其他人讨论 Clojure。最后，我要感谢我的儿子 Keita 和 Shota，他们教会我爱的真正含义，并不总是为了我个人。

CHOUSER

感谢我的父母，感谢你们的爱与支持——你们的探索精神开启了我的奇妙冒险之旅。我的哥哥 Bill，感谢你最早把我带入计算机的世界，领悟程序设计的乐趣与挑战。我的爱人 Heather，你始终如一的鼓励，贯穿了本书从始至终的创作历程。我的朋友和合作者 Michael Fogus，感谢你绝妙的灵感以及令人叹为观止的知识宽度，才有诸位手上的这本书。

关于本书

为何学习 Clojure?

你与莎士比亚之间仅有的差别在于习语的数量——而非词汇量的多寡。

—Alan Perlis

本书酝酿之际，我们的第一直觉是将 Clojure 与其宿主语言 Java 做一个全方位的比较。经过深入反思，我们得到的结论是，做好了最多算是狡猾，搞不好则是灾难。诚然，一些比较无可避免，但 Java 与 Clojure 有着很大的不同，为了说明一个而试图歪曲另一个，对二者都有失公允。因此，我们决定采用一种更好的方式，专注于编写代码本身的“Clojure 之道”。

当我们熟悉了一门程序设计语言，这门语言的惯用法和构造就会定义我们思考以及解决程序设计任务的方式。因此，面对一门全新的语言时，我们很自然地就会在精神上将新语言映射为我们熟悉的旧语言。但是，我们恳求你，请将所有的包袱丢在身后；无论你来自 Java、Lisp、Scheme、C#或是 Befunge，我们都请你铭记，Clojure 有其自身的语言，请遵循它自己的一套惯用法。你会发现，Clojure 与你已然熟知的语言之间在概念上有着一些联系，但请千万不要假设类似的东西就是完全一样的。

我们会努力工作，引导你了解 Clojure 用于构建思维模型的特性和语义，这样才能更有效地使用语言。本书的大多数例子都设计成可以在 Clojure 的交互式程序设计环境中运行，通常称为读取——求值——打印循环（Read-Eval-Print Loop），或是 REPL，这是一个极其强大的环境，用于实验和做快速原型。

当你读完本书，用 Clojure 之道思考以及解决问题将成为你的另一片舒适区。如果我们成功了，那么，你不仅可以成为一个更好的 Clojure 程序员，还能够以别样的视角看待程序设计语言的选择——无论是 Java、C#、Python、Ruby 还是 Haskell。重新评估一些我们已经认为是理所当然的主题，对于个人成长而言，是不可或缺的。

谁该阅读本书？

路是走出来的。

—Franz Kafka

本书并非 Clojure 初学指南。虽然我们的确提供了一些入门指导，但我们起步极快，没有在搭建可运行的 Clojure 环境上花费太多精力。此外，本书讨论的并非实现细节，而是语义细节。这也不是一本 Clojure 的“cookbook”，取而代之的是，通过对 Clojure 详尽的探究，为创建优美的软件提供素材。我们常常会解释这些素材如何整合，为什么它们搭配极佳，但这里没有全面的系统秘籍。我们的例子直接处理了手头的一些东西，有时还会把一些东西留给你去扩展，进一步丰富自己的知识。别指望我们将一门全面的课程装到一本书里，无论对你我，还是对 Clojure，这都是不可能的。通常，一本语言的书要花掉一半的篇幅介绍“真实世界”的情况，这与语言本身完全没有关系，希望我们能够规避这个陷阱。我们有一种强烈的感觉，如果能够告诉你语言背后的“为什么”，那你就能做好准备，将这些知识应用于真实世界的问题。简而言之，如果你找的是一本对新手负责的书，告诉你如何迁移既有代码库，连接 NoSQL 数据库以及探索其他“真实世界”的主题，那我们推荐 Amit Rathore 的《Clojure in Action》(Manning, 2011)。

总而言之，我们确实提供了一份语言简介，我们认为，如果你是愿意花时间理解 Clojure 的人，这本书就是为你准备的。此外，如果已经有了 Lisp 程序设计的背景，那你会觉得许多介绍材料看起来都很熟悉，所以，这本书对你而言是理想的。虽然绝非完美，但对于解决实际的程序设计问题，Clojure 拥有一套很好的特性组合，能够放入一致的系统中，解决程序设计中的问题。Clojure 鼓励我们思考问题的方式可能不同于我们习惯的方式，需要花些精力才能“得到”。但是，一旦跨过了这个门槛，我们或许能体会到一种愉悦，在本书里，我们就是想帮你到达那里。这是令人兴奋的时刻，我们希望你会同意，Clojure 会是带我们驶入未来所不可或缺的一种工具。

路线图

我们要带你上路了。也许之前你已然开启了自己的 Clojure 探索之旅。也许你是个 Java 或 Lisp 老手，第一次接触 Clojure。也许你来自完全不同的背景。无论如何，我们在对你说。本书自诩为写给冒险者，它需要我们丢开自己的包袱，带着开放的心态了解其中的主题。在很多方面，Clojure 会改变我们看待程序设计的方式，在其他一些方面，它会冲刷掉我们预先形成的一些理念。关于软件如何设计与实现，这门语言有很多要说的，本书将逐一触及这些主题。

根基

几乎每一种程序设计语言都有一些被认为是根本性的东西。偶尔，一种语言发明出

来会动摇软件产业的根基，驱散一些广为人知的既有的关于“良好软件实践”的概念。那些根本性的语言总能将一些全新的方式引入软件开发，缓和那个时代一些困难的问题，如果不能完全消除的话。任何一份根本性语言的列表都无可避免的会引发某些语言支持者的愤慨，在他们看来，其喜好的语言不应被忽略。但是，我们愿意承担此风险，所以，仅将下列程序设计语言归为此类。

根本性的程序设计语言

年份	语言	发明者	趣味阅读
1957	Fortran	John Backus	John Backus, “The History of Fortran I, II, and III,” IEEE Annals of the History of Computing 20, no. 4 (1998)
1958	Lisp	John McCarthy	Richard P. Gabriel 和 Guy L. Steele Jr., “The Evolution of Lisp” (1992), www.dreamsongs.com/Files/HOPL2-Uncut.pdf
1959	COBOL	由委员会设计	Edsger Dijkstra, “EWD 498: How Do We Tell Truths That Might Hurt?”, 出自 Selected Writings on Computing: A Personal Perspective (New York: Springer-Verlag, 1982)
1968	Smalltalk	Alan Kay	Adele Goldberg, Smalltalk-80: The Language and Its Implementation (Reading, MA: Addison-Wesley, 1983)
1972	C	Dennis Ritchie	Brian W. Kernighan 和 Dennis M. Ritchie, The C Programming Language (Englewood Cliffs, NJ: Prentice Hall, 1988)
1972	Prolog	Alain Colmerauer	Ivan Bratko, PROLOG: Programming for Artificial Intelligence (New York: Addison-Wesley, 2000)
1975	Scheme	Guy Steele 和 Gerald Sussman	Guy Steele 和 Gerald Sussman, the “Lambda Papers,” mng.bz/sU33
1983	C++	Bjarne Stroustrup	Bjarne Stroustrup, The Design and Evolution of C++ (Reading, MA: Addison-Wesley, 1994)
1986	Erlang	爱立信公司	Joe Armstrong, “A History of Erlang,” Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (2007)
1987	Perl	Larry Wall	Larry Wall, Tom Christiansen 和 Jon Orwant, Programming Perl (Cambridge, MA: O’Reilly, 2000)
1990	Haskell	Simon Peyton Jones	Miran Lipovac˘a, “Learn You a Haskell for Great Good!” http://learnyouahaskell.com/
1995	Java	Sun 微系统公司	David Bank, “The Java Saga,” Wired 3.12 (1995)
2007	Clojure?	Rich Hickey	你在读的这本

无论喜欢与否，少有争论的一点是，所列语言极大地影响了软件构造的方式。Clojure 是否应位列其中尚待观察，但是，Clojure 确实从诸多根本性的语言中借鉴了许多，其他有影响力的程序设计语言也让它获益良多。

第 1 章开启了我们的旅程，介绍了 Clojure 蕴含的一些核心概念。此章完结之时，我们应该可以很好地理解这些概念。沿途之中，我们展示了一些说明性的代码样例，突显了一些概念（还有些不错的图）。第 1 章所包含的多数内容均可以视为“Clojure 哲学”，因此，如果你想知道 Clojure 受什么启发以及由什么组成，这章就是为你准备的。

第 2 章快速地介绍了 Clojure 特定的特性和语法。

第 3 章讨论了一些不易归类的通用 Clojure 程序设计的惯用法。从真值和风格，到打包和 nil 的考量，第 3 章就是个大杂烩。所有的主题本身都很重要，从许多方面来看，这些内容是理解大部分 Clojure 惯用源码的起点。

数据类型

第 4 章讨论了标量数据类型，大多数程序员对这个话题相对熟悉，但还有一些重点需要注意，源于 Clojure 一些有趣的特性，这是寄宿于 Java 虚拟机的函数式程序设计语言所固有的。阅读本书的 Java 程序员会关注数学精度（4.1 节），Lisp 程序员则会关注 Lisp-1 vs. Lisp-2（4.4 节）。Clojure 里还包含了实用的正则表达式，并将其作为一等语法元素（4.5 节），程序员们会对此心存感激的。最后，长期的 Clojure 程序员也许会发现，关于有理数和关键字（分别在 4.2 和 4.3 节）的讨论，对这些貌似无辜的类型给出了全新的见解。

无论背景如何，第 4 章都会提供一些关键信息，帮助我们理解 Clojure 那些未受重视的标量类型的本性。

第 5 章涵盖了 Clojure 全新的持久化数据结构；任何希望深入了解它们的人都能够从中获得启迪。持久化数据结构位于 Clojure 程序设计哲学的核心，必须理解方能完全掌握 Clojure 设计决策的含义。我们只会简要触及这些持久化结构的实现细节，因为相对而言，理解为什么使用以及如何使用这些结构会更重要一些。

函数式编程

第 6 章会让人们对不变性、持久化和惰性有个大致的了解。我们会探索 Clojure 在支持并发程序设计中的关键元素：不变性。类似地，我们还会看到，有了不变性，许多与需要协调状态改变的问题都消失殆尽了。我们还会探索 Clojure 利用惰性降低内存占用以及加速执行时间的方式。最后，我们会谈及不变性和惰性的相互作用。如果你来自那些对修改不加限制且拥有严格求值表达式的语言，初涉之下，第 6 章或许是一种令人费解的体验。但这种令人费解会带来某种启迪，我们可能以前所未有的视角审视我们最喜欢的程序设计语言。

第 7 章全面展示了 Clojure 式的函数式编程。如果有函数式编程的背景，那你会熟悉本章的很多内容，虽然 Clojure 会呈现出其独特的一些东西。但类似于每一种被授予“函数式”称号的程序设计语言，Clojure 的实现给我们提供了一个不同的视角，让我们有机会审视自己之前的经验。如果你完全不熟悉函数式编程的技术，第 7 章可能是令人费解的。以对象层次结构和命令式程序设计技术为核心的语言里，函数式编程的概念犹如异类。但我们相信，由于 Clojure 的决策源自函数式范型的编程模型，它应该是个正确的方向，我们希望你也会赞同。

大规模设计

任何规模的应用都可以把 Clojure 当做主要的语言，第 8 章关于宏的讨论会改变我

们对于开发软件的想法。作为一种 Lisp，Clojure 也拥抱了宏，我们会带你经历理解宏的过程，让你意识到，能力越大，责任越大。

在第 9 章里，我们会带你领略 Clojure 内建的对“代码和数据”进行组合及关联的机制。从命名空间到多重方法到类型和协议，我们会逐一解释 Clojure 如何促进大规模应用的设计和实现。

Clojure 是一种共生的程序设计语言，这意味着，它要运行于宿主环境之上。目前选择的宿主是 Java 虚拟机，但未来，Clojure 可能会变成跨宿主平台。无论如何，Clojure 都会提供一流的函数和宏，用于与宿主平台的直接交互。

在第 10 章里，我们会讨论 Clojure 与其宿主互操作的方式，自始至终关注于 JVM。

Clojure 与生俱来对程序状态就有着完善的管理，简化了并发程序设计，这些内容会在第 11 章看到。Clojure 的状态模型简单而强大，缓和了这种复杂任务所包含的大多数头疼问题，我们会逐个为你展示如何以及为什么使用。此外，我们还会强调一些并非由 Clojure 直接解决的问题，比如，如何识别和降低对 Clojure 引用类型所保护元素的需要。

杂项考量

本书最后一部分讨论了同样重要的话题：透过 Clojure 哲学的视角看待我们应用的设计和开发。在第 12 章里，我们会讨论改善单线程应用性能的一些方式。Clojure 提供了许多机制改善性能，我们会逐一深入，包括其用法及适用范围。作为本书的总结，在第 13 章里，我们强调了在某些偏离开发行为的方面，Clojure 改变了我们思考的方式，比如定义自己的应用领域语言、测试、错误处理和调试。

代码约定

本书的源码都采用直白而实用的方式进行了格式化。文本里内联列出的源码，比如 (:lemonade :fugu)，都采用等宽字体并加粗。在代码块里列出的代码片段距左边有一些偏移，采用等宽字体并加粗以突显出来：

```
(def population {::zombies 2700 ::humans 9})
(def per-capita (/ (population ::zombies) (population ::humans)))
(println per-capita "zombies for every human!")
```

如果源码片段表示一个表达式的结果，那么结果会有个前缀——“;=>”。这种特殊的序列有三重目的：

- 有助于将结果从代码表达式中突显出来。
- 它表示一个 Clojure 注释。
- 因为如此，可以轻松地将整个代码块从本书的 EBook 或 PDF 版本中复制出来，粘贴到 Clojure REPL 里运行：

```
(def population {::zombies 2700 ::humans 9})
(/ (population ::zombies) (population ::humans))
;=> 300
```

此外，在 REPL 里，如果预期的显示不是返回值（比如表达式或打印输出），那么，实际的返回值前面会有个先导的“;”：

```
(println population)
; {:user/zombies 2700, :user/humans 9}
;=> nil
```

在上面的例子里，显示为{:user/zombies 2700, :user/humans 9}的 map 就是个打印值，而 nil 表示 println 函数的返回值。如果表达式后面没有显示返回值，那么我们可以认为就这个例子而言，要么是 nil，要么是忽略了。

阅读 Clojure 代码 阅读 Clojure 代码，如果从左向右阅略读，只要注意重要部分的上下文（defn、binding、let 等）即可。而由内而外阅读，则要仔细注意每个表达式返回的东西，以传给紧邻的外部函数。在阅读最内部的表达式时，记住整个外部上下文会让事情简单许多。

无论是内联，还是代码块，所有格式化过的代码都是为了让键入或粘贴同写进 Clojure 源码或 REPL 的完全一样。一般来说，Clojure 的提示符 user>没有显示，因为它会导致复制/粘贴的失败。最后要说的一点是，我们有时会用省略号...表示略去的结果或打印输出。

在许多列表里还有一些代码标记，强调了一些重要概念。某些情况下，还会有一些用以解释的数字编号链接跟在列表后面。

获取 Clojure

如果你目前还没装 Clojure，那么，我推荐你使用 David Edgar Liebke 创建的 Clojure REPL 包（Cljr），位于 http://joyofclojure.com/cljr，其安装按如下指令进行。

先决条件

- Java 1.6 及以上版本
- 互联网连接

指令

在操作系统的控制台里运行如下命令：

```
java -jar cljr-installer.jar
```

如果你下载的 Cljr 包是以.zip 文件为扩展名，运行方式如下：

```
java -jar cljr-installer.jar.zip
```

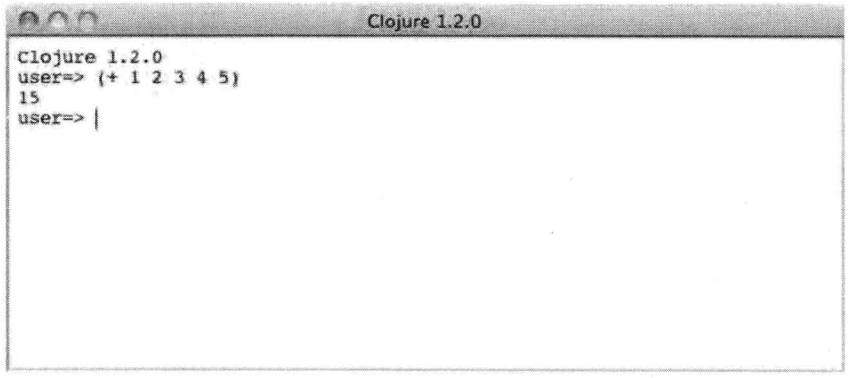
这样，你会看到 Cljr 安装和包下载的进度输出。完成之后，你会看到类似于下面的指令，提示你运行 Cljr：

```
Cljr has been successfully installed. Add $HOME/.cljr/bin to your PATH:
    $ export PATH=$HOME/.cljr/bin:$PATH
Run 'cljr help' for a list of available commands.
```

遵循所示命令，即可运行 Cljr。

REPL

Cljr 包会运行 1.2.0 版的 Clojure REPL（读取/求值/打印循环，Read/Eval/Print Loop）——这正是本书对应的版本。启动 Cljr 程序之后，你会看到如下图所示的窗口。



Cljr REPL 类似于标准的 Clojure REPL，只是额外增加了一些便于使用的特性，详情参见 http://github.com/fogus/cljr。

本书并不假定你用了 Cljr，但无论你的个人 REPL 设置是怎样的，应该都可以——只要运行的是 Clojure 1.2 版。

下载示例代码

本书所有的可运行源码都可以从出版商的网站下载，www.manning.com/TheJoyofClojure。

作者在线（Author Online）

购买本书的同时，你也就可以免费访问 Manning Publications 的一个私有 Web 论坛，可以在上面发表评论，咨询技术问题，得到作者和其他用户的帮助。要访问和订阅这个论坛，请到 www.manning.com/TheJoyofClojure。注册之后，就可以从这个页面获得论坛访问信息，了解得到怎样的帮助，以及论坛的管理规则。

Manning 对读者的承诺是，提供一个场所，让读者之间以及读者和作者之间进行一