The Pragmatic Programmers

·新·锐·编·程·语·言·集·萃·

Programming Clojure

Second Edition

Clojure程序设计

【美】Stuart Halloway· Aaron Bedra 著

温瑞云 译

- Clojure编程经典之作
- Clojure之父作序推荐

人民邮电出版社
POSTS & TELECOM PRESS

The Pragmatic Programmers

·新·锐·

Programming Clojure

Second Edition

Clojure程序设计

【美】Stuart Halloway
Aaron Bedra 著

温瑞云 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Clojure程序设计 / (美) 哈罗威 (Halloway, S.), (美) 拜卓 (Bedra, A.) 著 ; 温瑞云译. -- 北京 : 人民邮电出版社, 2013.4
ISBN 978-7-115-30847-4

Ⅰ. ①C… Ⅱ. ①哈… ②拜… ③温… Ⅲ. ①程序设计 Ⅳ. ①TP311.1

中国版本图书馆CIP数据核字(2013)第010005号

版 权 声 明

Clojure 程序设计

◆ 著 [美] Stuart Halloway Aaron Bedra
译 温瑞云
责任编辑 陈冀康

◆ 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
北京天宇星印刷厂印刷

◆ 开本：800× 1000 1/16
印张：17.5
字数：330 千字 2013 年 4 月第 1 版
印数：1–3 000 册 2013 年 4 月北京第 1 次印刷

著作权合同登记号 图字：01-2012-4614 号

ISBN 978-7-115-30847-4

定价：49.00 元

读者服务热线：(010)67132692 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

内容提要

Clojure 是一种基于 Java 虚拟机的动态编程语言。它功能强大，支持函数式编程，简化了并发编程，并且能调用 Java 程序。正是这些优点，使其赢得了众多程序员的青睐。

本书是介绍 Clojure 编程语言和程序设计的经典之作。第 2 版针对 Clojure 1.3 进行了全面的更新。全书共包括 10 章，分别介绍了 Clojure 概览和基本特性、序列和函数式编程、并发编程模型、协议和数据类型、宏、多重方法，以及 Clojure 对 Java 的调用。最后提供了一个完整了解 Clojure 应用开发全过程的实例。

本书适合有不同语言背景而想要学习和了解 Clojure 编程的程序员阅读。函数式程序员、Java 和 C#程序员、Lisp 程序员，以及 Perl、Python 和 Ruby 程序员，都能够通过阅读本书得到不同程度的收获。

献给 Craig Bedra，我的父亲和导师。

是你让我懂得了通过探索进行学习的价值，

并且让我明白诸如魔法一类的事情是不存在的。

——Aaron

致谢

有许多人为本书那些精彩的部分做出了贡献。残留的问题和错误完全是由我们造成的。

感谢 Relevance 和 Clojure/core 这两个了不起的团队，是你们创造性的氛围，让好点子得以茁壮成长。

感谢 Clojure 邮件列表[①]里友善的人们，是你们给予了我们莫大的帮助和鼓励。

感谢 Pragmatic Bookshelf 的所有人。尤其是我们的编辑 Michael Swaine，为我们有些狂妄的交付时间表提供了很好的建议。感谢 Dave Thomas 和 Andy Hunt，你们创建了一个有趣的技术书籍编写平台，它令书籍作者热情满怀。

感谢所有向本书勘误表页面[②]投递建议的人们。

感谢技术评审们的意见和建议，你们是 Kevin Beam、Ola Bini、Sean Corfield、Fred Daoud、Steven Huwig、Tibor Simic、David Sletten、Venkat Subramaniam 和 Stefan Turalski。

特别要感谢 David Liebke，是你编写了第 6 章“协议和数据类型”的最初版本。如果不是因为你用全新的思路提供了一份奇妙的指南，这本书不会成为现在的样子。

感谢 Rich Hickey，是你创造了 Clojure 这种杰出的编程语言，并培育出围绕着它的社区。

感谢我的妻子 Joey，我的女儿 Hattie、Harper 和 Mabel Faire。是你们让我每天都如沐阳光。

——Stuart

感谢我的妻子，Erin，是你给予了我无尽的爱与鼓励。

——Aaron

① http://groups.google.com/group/clojure。

② http://www.pragprog.com/titles/shcloj2/errata。

第1版序

我们正在被复杂性淹没。其中绝大部分是偶然复杂性——并非源自问题本身，而是源自于我们为了解决问题而采用的方法。面向对象编程看起来容易，但生产出来的程序，往往会成为一张由可变对象交织而成的复杂巨网。单单是因为调用了一个对象上的方法，就可能会引起遍及整个对象关系图的一连串变化。在这种情况下，想要理解何时将会发生何事，事物是如何进入某种状态的，或是为修正某个 bug 而试图让对象回到特定状态，都变得极为复杂。而且一旦混入了并发问题，事情很快就会变得无法收拾。我们对程序使用了仿制对象（mock objects）和测试套件（test suites），但依然收效甚微。这不得不让我们对手头的工具和编程模型产生质疑。

函数式编程提供了别的选择。通过强调纯函数——传入和返回的都是不可变值（immutable value），副作用的存在成为了特例，而不再是常态。在多核架构中，我们面临的并发问题日益增加。因此，这种特性只会变得更加重要。Clojure 的设计目标，就是要让函数式编程更加平易近人，且兼备商业软件开发者所需的实用性。首先要能运行在像 Java 虚拟机这样可靠的基础设施之上，还要能支持现有客户在 Java 框架及库方面的投资。Clojure 非常清楚满足上述两点的必要性，以及这么做能带来的巨大实用性。

这是一种面向专业开发者（Stuart 自己就是）的编程语言。Stuart 的这本书之所以让人如此激动，源于他对 Clojure 的把握。很明显，他对 Clojure 瞄准的痛处有着充分的体验，这也增强了本书务实的态度。阅读本书，就像是一次充满激情的旅行，以实际应用为基础，逐步介绍 Clojure 的关键特性——有可能是一些全新的概念。我希望它能鼓舞你运用 Clojure 去编写软件，然后，回过头来你会说：“我不仅仅完成了工作，而且没想到采用的方法竟然如此健壮和简单，最棒的是，编写 Clojure 代码实在是太有乐趣了！”。

——Rich Hickey

Clojure 之父

第 2 版序

自本书第 1 版发行之后，很多东西发生了改变。是的，Clojure 语言本身得以增强，例如协议（protocol）和记录（record）的引入。然而，最有意义的是，Clojure 已应用于各种不同的领域。人们正在使用 Clojure 建立新系统，分析大规模数据集，以及处理通信、金融、互联网和数据库方面的工作。同时，一个大型的、乐于助人的社区已经围绕着 Clojure 成长起来。随之而来，涌现出了大量的程序库。这些库的出现格外令人激动，不仅仅在于它们提供的功能设施；还在于它们当中最优秀的那些库，欣然接受了 Clojure 倡导的方法和机制，从而在简洁性和互操作性方面达到了全新的高度。

Stuart 和 Aaron 确保在第 2 版中覆盖了语言的新增功能，并尝试着借助一些来自社区的库，来演示这些功能是如何运作的。本书仍然提供了令人振奋的 Clojure 简介，我希望它能激发你加入我们的社区，并最终对 Clojure 生态系统做出贡献。

——Rich Hickey

Clojure 之父

前 言

Clojure 是一种基于 Java 虚拟机（JVM，Java Virtual Machine）的动态编程语言（dynamic programming language）。它具有以下引人注目的特性。

- Clojure 非常优雅。
  摒弃了杂乱累赘的语法束缚，Clojure 干净、仔细的设计使你在编写代码时总能立刻切入问题的本质。

- Clojure 是 Lisp 的再度崛起。
  Clojure 从 Lisp 继承了强大的力量，却未受到 Lisp 历史的束缚。

- Clojure 是一种函数式语言（functional language）。
  作为一门函数式语言，Clojure 的数据结构具有不可变性（immutable），且大多数函数没有副作用（side effect）[①]。因此，编写正确的程序更加容易，也能更轻松地将小程序组合成一个大家伙。

- Clojure 简化了并发编程。
  很多其他语言围绕同步锁（locking）建立的并发模型，难以驾驭。为此，Clojure 提供了数个锁机制的替代方案：软事务内存（Software Transactional Memory，STM）、代理（agent）[②]、原子（atom）和动态变量（dynamic variable）。

- Clojure 与 Java 彼此亲密无间。
  在 Clojure 中调用 Java 代码，无需任何中间的转换层，直接而且快速。

- 不同于许多其他流行的动态语言，Clojure 运行飞快。
  Clojure 的实现利用了现代 Java 虚拟机上的众多优化技术。

尽管许多其他语言也包含了上述诸多特性中的一部分，但与它们相比，Clojure 仍

---

① 译注：初次接触函数式编程的读者可能对何为“副作用”略感疑惑。所谓副作用，就是指某个函数执行期间导致系统状态发生了变化。我们常见的大多数语言都有副作用。而典型的函数式语言没有赋值语句，也就不会对包括全局变量、函数参数或局部变量在内的各种系统状态造成影响。

② 译注：本书中后续章节中还将出现另外一个含义的代理（Proxy），但绝大多数情况下，根据上下文即可分辨文中出现的“代理”应该是 Agent 还是 Proxy，在可能会引起误解的地方，译者会加以注明。

显得魅力非凡。上述任何一个特性，都极为强大和有趣。Clojure 的迷人之处在于，将这些特性以非常干净的方式融合在了一起，且做到彼此协作无间。本书的第 1 章“启航”，将介绍以上这些特性及更多的内容。

谁应该阅读本书

Clojure 是一种强大的通用型（general-purpose）编程语言。如果你是一名经验老到的程序员，具备类似 C#、Java、Python 或者 Ruby 这样的现代编程语言的开发经验，并正在寻找更为强大、更加优雅的编程语言，那么本书是为你量身定做的。

Clojure 构建于 Java 虚拟机之上，并且运行飞快。如果你是一名对表现力丰富的动态语言馋涎已久，但却因为对性能问题的担忧而裹足不前的 Java 程序员，那么本书将引起你特别的兴趣。

Clojure 有助于重新定义，一种通用型编程语言应该包含哪些特性。如果你使用 Lisp，或使用一种诸如 Haskell 这样的函数式语言，又或者正编写明显存在并发的程序，那你一定会享受 Clojure 的一切。Clojure 融合了来自 Lisp、函数式编程和并发编程领域的理念，使得初次接触这些概念的程序员，一切触手可及。

Clojure 是本轮编程语言形态大规模演化现象的一部分。诸如 Erlang、F#、Haskell 和 Scala 这样的语言，由于它们支持函数式编程，或是由于它们的并发模型，最近都得到了格外的关注。作为这些语言的忠实信徒，你也一定会从 Clojure 当中找到众多共通之处。

本书主要内容

第 1 章，启航。本章将展示作为一门通用型语言，Clojure 的优雅特质及其函数式风格，以及独特的并发模型如何令其独一无二。阅读完本章，你还将能够轻松完成 Clojure 的安装，并学会如何使用 REPL 进行交互式开发。

第 2 章，探索 Clojure。在这里，我们将对 Clojure 的核心构造进行一次广度优先的概览。完成本章的阅读后，你将能顺畅地阅读大多数常规的 Clojure 代码。

接下来的两章将讨论函数式编程。第 3 章，一切皆序列，将展示 Clojure 如何使用强大的序列隐喻，统一了所有的数据形态。

第 4 章，函数式编程。本章将向你展示如何编写与序列库代码风格相同的函数式程序。

第 5 章，状态。本章我们将深入 Clojure 的并发编程模型。探讨 Clojure 中用于处理并发问题的 4 种强大模型。此外还有来自 Java 并发库中的精华内容一并奉上。

第 6 章，协议和数据类型。本章将逐个介绍在 Clojure 中的记录（record）、类型（type）和协议（protocol）。这些概念自 Clojure 1.2.0 版本首次引入后，在 Clojure 1.3.0 版本中得到了进一步增强。

第 7 章，宏。本章将不加掩饰地炫耀这一来自 Lisp 中的标志性特性——宏（Macros）。你将看到它如何利用“Clojure 代码本身就是数据”这一特质，提供了在其他非 Lisp 语系中极难甚至无法实现的非凡的元编程能力。

第 8 章，多重方法。本章将讨论 Clojure 解决多态问题的众多方法中的一种。多态，通常意味着“获取第一个参数的类型，并据此调度到相应的方法”。Clojure 的多重方法，使你可以更进一步，选择适用于所有参数的任意函数来进行调度。

第 9 章，极尽 Java 之所能。在本章中，你将看到如何从 Clojure 中调用 Java，以及从 Java 中调用 Clojure。你还将看到如何让 Clojure 疯狂运转，获得原生 Java 级别的性能。

最后，第 10 章，搭建应用。本章提供了一个可以让你完整了解 Clojure 应用开发全过程的视角。在这里，你将从头开始创建一个应用，并深入解决问题的方方面面，同时，还会考虑关于简单和质量的话题。你将借助一组有用的 Clojure 库，生产并发布一个 Web 应用。

附录，编辑器。这里列出了可供你选择的 Clojure 代码编辑器列表，并分别提供链接指向它们各自的安装说明。

如何阅读本书

所有读者都应该按顺序阅读最初的两章。请特别关注 1.1 节，这里提供了 Clojure 具备哪些优势的概述，里面的内容你一定会感兴趣。

持续的试验。Clojure 提供了一个可以让你立即获取反馈的交互式环境。请阅读 1.2.1 小节，以获得更多的信息。

读完最初的两章，你就可以随意翻阅了。但如果你打算开始阅读第 5 章，那么，确保你已经读过了第 3 章。顺序阅读这几章，将引导你从理解 Clojure 的不可变数据结构开始，一直到能够利用 Clojure 强大的并发模型，编写正确的并发程序。

当你开始接触后面各章中那些较长的代码示例时，请确保你使用的编辑器能为你提供 Clojure 代码自动缩进功能。附录“编辑器”列举了编写 Clojure 代码的通常选择。如果可能，请尝试使用支持括号匹配功能的编辑器，例如 Emacs 的 paredit 模式或者安装了 CounterClockWise 插件的 Eclipse。这些编辑功能将为你顺利学习 Clojure 编程提供巨大的帮助。

致函数式程序员

- Clojure 的函数式编程之道，在于将理论的纯粹之美，与 Clojure 需要运行在当前 Java 虚拟机之上的现实做出了完美的平衡。倘若仔细地阅读了第 4 章“函数式编程”，你将了解到 Clojure 与诸如 Haskell 这样的学院派语言之间存在的风格差异。
- Clojure 的并发模型（第 5 章），提供了数个直截了当的途径，用于处理并发世界中副作用和状态的问题。这也使得广大读者可以深入地体验函数式编程之魅力。

致 Java 和 C#程序员

- 请认真阅读第 2 章，Clojure 只有很少的语法规则（相比 Java 或 C#而言），所以我们能很快地熟悉它们。
- 请特别留意第 7 章，Java 或 C#背景的程序员将会发现，这部分是 Clojure 与他们所熟悉的语言之间的最大不同。

致 Lisp 程序员

- 第 2 章中的一些内容你可能已经很熟悉了，但无论如何，还是应该读一下这一章。Clojure 从 Lisp 中承袭了众多关键特性，但它也在一些地方打破了 Lisp 传统，这里将会讨论这些内容。
- 请密切关注第 4 章中的惰性序列。
- 为你的 Emacs 装备一个“clojure-mode”吧，这将为你享受后面章节中的代码

示例提供很大便利。

致 Perl、Python 和 Ruby 程序员

- 仔细阅读第 5 章，在 Clojure 中，进程内并行计算是一个非常重要的话题。
- 拥抱宏吧（第 7 章）。但请不要寄予太大的期望，能将你所用语言中的元编程风格轻松套用到 Clojure 宏中。请牢记，Clojure 的宏更为强大，并且，它是在代码读取期间被执行的，而非在运行期执行。

编写体例

以下编写体例将从始至终地贯穿于本书之中。

代码示例采用以下字体。

```
(+ 2 2)
```

为区别代码示例及其执行结果，我们会在执行结果前放置一个箭头（->）。

```
(+ 2 2)
-> 4
```

同样，控制台的输出也不容易与示例代码和结果区别开来，因此，我们会在控制台的输出前放置一个管道（|）符。

```
(println "hello")
| hello
-> nil
```

当首次引入某个 Clojure 形式（form），我们需要说明其语法时，将采用下述表示法。

```
(example-fn required-arg)
(example-fn optional-arg?)
(example-fn zero-or-more-arg*)
(example-fn one-or-more-arg+)
(example-fn & collection-of-variable-args)
```

这是一种非正式的语法，采用?、*、+和&符号，用于说明不同的参数传递模式。

Clojure 的代码是以程序库的形式进行组织的。如果本书某段示例代码所依赖

的库没有包含在 Clojure 语言核心中，我们将用 Clojure 的 use 或 require 对此加以说明。

```
(use '[lib-name :only (var-names+)])
(require '[lib-name :as alias])
```

此处使用 use 引入仅出现在列表 var-names 中的名称。使用 require 则创建一个库别名，使得每个引入函数的来源更加明晰。例如，来自于 clojure.java.io 库中的常用函数 file。

```
(use '[clojure.java.io :only (file)])
(file "hello.txt")
-> #<File hello.txt>
```

或使用基于 require 的版本。

```
(require '[clojure.java.io :as io])
(io/file "hello.txt")
-> #<File hello.txt>
```

事实上，如果成功调用了 use，Clojure 会返回 nil。但为使本书更加简洁，这个输出在示例清单中省略了。

在阅读本书期间，你将在名为 REPL 的 Clojure 交互式环境中输入代码。REPL 的控制台提示符形如下。

```
user=>
```

提示符中的 user 表明了你当前所在的 Clojure 名字空间。在本书大多数的例子中，当前位于哪个名字空间无足轻重。在这种情况下，我们将其省略，采用下述更简洁的语法表示在 REPL 中发生的一切。

```
(+ 2 2)        ; 没有命名空间提示的输入行
-> 4           ; 返回值
```

少数情况下，当位于哪个名字空间非常重要时，我们将采用如下语法。

```
user=> (+ 2 2)     ; 有命名空间提示的输入行
-> 4               ; 返回值
```

Web 资源及反馈

本书的英文官方主页[①]位于 Pragmatic Bookshelf 站点。在这里你可以订购本书的纸质版或是电子版，并且下载本书的示例代码。同样，你也可以将你的反馈提交至勘误表[②]或是直接发表至本书论坛[③]。

下载示例源码

你可以在下列任意位置找到本书的示例源码。

- 本书主页[④]上有链接指向官方发布的源码。同时，每次本书发布新版时，源码也将得到更新。
- 处于实时更新的本书 git 源码仓库[⑤]。这里有最新、最棒，且有时甚至强过书中所示的源码。

除非另行说明，示例文件都分别放在 examples 目录中。

贯穿于本书，示例源码的文件名列于源码清单起始位置，并采用灰色背景加以区别。例如，下面的源码清单来自于 src/examples/preface.clj。

```
src/examples/preface.clj
(println "hello")
```

如果你正在阅读的是本书的 PDF 版本，你可以直接点击文件名下载对应的源码清单文件。

有示例源码在手，你就可以准备启航了。首先，我们将领略究竟是怎样的特性组合，使得 Clojure 如此的独一无二。

① http://www.pragprog.com/titles/shcloj2/programming-clojure。
② http://www.pragprog.com/titles/shcloj2/errata。
③ http://forums.pragprog.com/forums/207。
④ http://www.pragprog.com/titles/shcloj2。
⑤ http://github.com/stuarthalloway/programming-clojure。

目 录