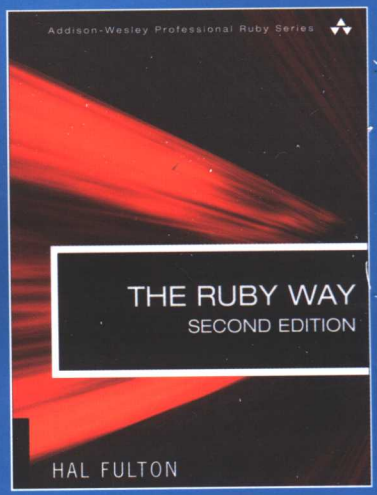
PEARSON
Addison Wesley

From Programmer,
For Programmer

- Ruby on Rails之父David Heinemeier Hansson倾力推荐
- Ruby创始人松本行弘倾情作序
- Ruby编程专家透彻阐述Ruby之道

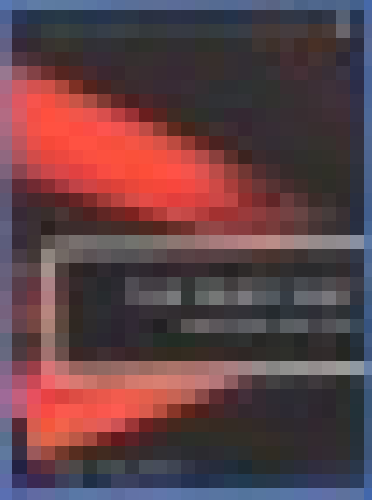


The Ruby Way（第二版）中文版

〔美〕Hal Fulton 著
陈秋萍 赵子鹏 译

人民邮电出版社
POSTS & TELECOM PRESS

From Programmer To Programmer



The Ruby Way

（第二版）

中文版

From Programmer, For Programmer

The Ruby Way (第二版) 中文版

〔美〕Hal Fulton 著
陈秋萍 赵子鹏 译

人民邮电出版社
北京

图书在版编目（CIP）数据

The Ruby Way（第二版）中文版 /（美）富尔顿（Fulton,H.）著；陈秋萍，赵子鹏译. —北京：人民邮电出版社，2007.11
ISBN 978-7-115-16669-2

Ⅰ. T… Ⅱ. ①富…②陈…③赵… Ⅲ. 软件开发 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2007）第 122956 号

版 权 声 明

The Ruby Way（第二版）中文版

◆ 著 [美] Hal Fulton
译 陈秋萍 赵子鹏
责任编辑 李 际

◆ 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 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
河北省三河市海波印务有限公司印刷
新华书店总店北京发行所经销

◆ 开本：800 × 1000 1/16
印张：34.75
字数：821 千字 2007 年 11 月第 1 版
印数：1–4 000 册 2007 年 11 月河北第 1 次印刷

著作权合同登记号 图字：01-2007-0983 号
ISBN 978-7-115-16669-2/TP

定价：79.00 元

读者服务热线：(010)67132705 印装质量热线：(010)67129223

内容提要

Ruby 是一种面向对象的敏捷语言，借鉴了 LISP、Smalltalk、Perl、CLU 和其他语言的出色特性。在本书第 1 版面世后的 5 年内，Ruby 得以日益流行。

本书采用“如何解决问题”的方式阐述 Ruby 编程，涵盖了以下内容：Ruby 术语和基本原理；数字、字符串等低级数据类型的操作；正则表达式；国际化和 Ruby 消息目录；散列、数组及栈、树和图等其他数据结构的操作；I/O、文件和对象持久化；Ruby 特有的 OOP 技术及动态编程；Tk、GTK+、Fox 和 Qt 等 Ruby 图形用户界面；使用 Ruby 线程实现轻量级多任务；使用 Ruby 编写脚本和进行系统管理；使用图像文件、PDF、YAML、XML、RSS 和 Atom；Ruby 代码的测试、调试、性能分析和打包；低级网络编程和客户/服务器交互；Rails、Nitro、Wee、IOWA 等 Web 开发工具；使用分布式 Ruby、Rinda 和 Ring。书中包含 400 多个按主题分类的示例，每个示例都回答了“如何使用 Ruby 来完成”的问题。首先对要完成的任务进行了描述，并讨论技术方面的约束条件，然后循序渐进地阐述一种优秀的解决方案，并辅以说明和解释以帮助读者理解。

作者 Hal Fulton 以旁征博引、引人入胜而又清晰易懂的写作手法，全面而详细地阐述了 Ruby，让读者感到犹如有一位知识渊博的老师在身边，并渴望将其掌握的知识悉数传授给你。本书可作为中高级程序员深入了解 Ruby 以及使用它来解决实际问题的参考手册。

作者简介

Hal Fulton 拥有密西西比大学的两个计算机学位。在社区大学里教授计算机课程 4 年后，他因工作需要（主要受聘于奥斯汀的 IBM）搬到了德克萨斯州奥斯汀。他拥有 15 年使用各种 UNIX 系统（包括 AIX、Solaris 和 Linux）的经验。他首次接触 Ruby 是在 1999 年，从 2001 年开始他就着手编写本书的第 1 版，这是第二本用英语编写的 Ruby 图书。他参加了六次 Ruby 大会，并在其中的四次中发表过演讲，包括在德国卡尔斯鲁厄召开的首届欧洲 Ruby 大会。目前，他在德克萨斯奥斯汀的 Broadwing Communications 工作，主要从事大型数据仓库和电信应用方面的工作。他每天都在使用 C++和 Oracle，当然还有 Ruby。

Hal 仍活跃在 Ruby 邮件列表和 IRC 频道中，且正在开发几个 Ruby 项目。他是 ACM 和 IEEE 计算机协会的成员。在生活中，他喜欢音乐、阅读、写作、艺术和摄影。他还是 Mars 社团的成员和天文爱好者，平生的愿望是进行一次太空旅行。

序　言

第 2 版序言

古代中国人，尤其是哲学家，认为世界和万物之后还隐藏着什么。它不可言传，不可解释，也无法用具体言语来形容。在中国和日本，这被称为“道”。与道相关的有柔道、剑道、空手道和合气道。这些不仅是武术，它们还包含着一种哲学和一种生活方式。

同样，Ruby 编程语言也有自己的哲学和思维方式。它启迪人们以不同的方式思考，让编程人员更享受其工作。这并不是因为 Ruby 来自日本，而是因为编程是人类（至少是有些人）的一个重要工作，而使用 Ruby 可以使工作更轻松。

“道”是很难形容的。我能够感受到它，但未曾尝试过用语言来解释它。这太难了，即使是用我的母语——日语。但 Hal Fulton 这样做了，且第一次（本书的第 1 版）就做得很不错。由于得到了 Ruby 社区许多人的帮助，他的第二次尝试（即本书第 2 版）更出色。随着 Ruby 受欢迎程度的提高（一个原因是 Ruby on Rails），理解程序员的编程效率秘诀显得更重要。希望本书可以帮助读者成为高效的程序员。

编程愉快！

松本行弘
2006 年 8 月于日本

第 1 版序言

我最早接触到计算机是在 20 世纪 80 年代初，不久之后我就对编程语言感兴趣。从那以后我就成了一个语言极客（geek）。我想原因是编程语言是表达人类思想的方式。编程语言本质上是面向人类的。

但是，编程语言越来越面向机器，许多语言是为计算机方便而设计的。

随着计算机变得越来越强大也越来越便宜，这种情况正在逐渐改变。以结构化语言为例，机器并不关心程序是否是结构化的，而只是逐位地执行。结构化编程不是面向机器的，而是面向人类的，面向对象编程也如此。

现在的语言设计应关注人类了。

1993 年，我与一位同事讨论了脚本语言的强大功能及其未来。我认为将来的编程应像脚本一样——面向人类。

但我对现有的语言（如 Perl 和 Python）并不满意。我要的是比 Perl 功能更强大的、比 Python 更面向人类的一种语言。我不能找到这样的理想语言，因此决定自己创造一种。

Ruby 不是最简单的语言，但人类思维并不像它的自然状态那样简单，它同时喜欢简单性和复杂性。它不能处理太多复杂的事物，也不能处理太多简单的事物，而需要平衡。

因此，在设计面向人类的语言 Ruby 时，我遵循了“最小惊讶”原则。我认为那些不那么使我感到意外的是好的。因此，使用 Ruby 编程时，我感觉到的是一种很自然的感觉，甚至是愉快的感觉。自 Ruby 于 1995 年首次发布以来，全球的很多程序员和我一样，认为 Ruby 编程是一件愉快的事情。

我非常感激 Ruby 社区里的人们，他们是 Ruby 成功的关键。

我也很感谢本书的作者 Hal E. Fulton，他提出“Ruby 之道”来帮助人们。

本书阐述了 Ruby 的哲学，对我和 Ruby 社区的思想进行了提炼。我不知道 Hal 怎么能够理解我的思想并揭示 Ruby 之道的秘诀。我从来没有见过他，希望不久可以见到他。

希望本书和 Ruby 都能使你的编程工作更有趣、更愉快！

松本行弘

2001 年 9 月于日本

前　言

道可道，非常道。

——老子，《道德经》

本书名为 *The Ruby Way*（《Ruby 之道》），这需要解释一下。

作者的目标是尽力使本书符合 Ruby 哲学，这也是其他撰稿人的目标。成功的荣誉应与他们分享，但对任何错误的指责将由作者承担。

当然，作者无法准确地阐述 Ruby 的精神，这主要应由松本行弘来说，但即使是他也难以用语言来描述。

总之，这是一本书，而 Ruby 之道是 Ruby 语言创始人和整个 Ruby 社区管辖的领域，一本书难以描述清楚。

虽然如此，作者还是尝试在前言中诠释不可言传的 Ruby 之道，聪明的学生不应将其视为权威。

这是第 2 版，很多内容没变，也有很多内容有了变化。我们保留了前言的大部分内容，但读者应阅读下一节，其中对修订和新增的内容做了总结。

第 2 版简介

万物都在变化，Ruby 也不例外。作者于 2006 年 8 月撰写该前言时，本书第 1 版已出版将近 5 年，是该进行修订了。

本版进行了大量的修订并新增了大量的内容。原来第 4 章的内容被划分到 6 章中，其中的两章（“符号和范围”与“Ruby 的国际化”）是新增的，其他 4 章也新增了示例和说明。有关正则表达式的内容更多了，不仅介绍了经典正则表达式，还介绍了正则表达式引擎 Oniguruma。

第 8 章和第 9 章原来是一章，但新增内容后篇幅太多，因此将其分为两章。

同样，添加新内容后，将原来的第 9 章分为第 18～20 章，删除了原来的附录以留出篇幅介绍更多的内容。

另外，还新增了如下几章：

- 第 15 章介绍 XML、RSS、图像文件和 PDF 文件的创建等。
- 第 16 章介绍单元测试、性能分析、调试、代码覆盖率等主题。
- 第 17 章介绍 setup.rb 的用法和 RubyGems 的创建等。
- 第 21 章从用户的角度介绍 Ruby 编辑器和 IDE、ri 工具以及 RubyGems。
- 第 22 章概述了主要网站、邮件列表、新闻组、会议和 IRC 频道等。

从很大程度上说，本书的每章都是“全新”的。每章都经过修订，做几百处的小修订和几十处的重大修订。删除了过时或不那么重要的内容，根据 Ruby 本身的变化修订了内容，每章都新增了示例和说明。

读者可能想知道在原有章节中新增了哪些内容。一些新增的内容包括前面说过的 Oniguruma、数学运算库和类（如 BigDecimal、mathn 和 matrix）以及新类（如 Set 和 DateTime）。

第 10 章新增了有关 readpartial、非阻断 I/O、StringIO 类等方面的内容，还新增了有关 CSV、YAML 和 KirbyBase 的内容。在该章的数据库部分，新增了有关数据库 Oracle、SQLite 和 DBI 以及对象关系映射（Object-Relational Mappers，ORM）的内容。

第 11 章介绍了新出现的 Ruby 元素，如 initialize_copy、const_get、const_missing 和 define_method，还讨论了委托（delegation）和转交（forwarding）技术。

第 12 章的内容必须修订（尤其是有关 GTK 和 Fox 的两节）。有关 QtRuby 的一节是全新的。

第 14 章讨论了 Windows one-click 安装程序和几个类似的包，并改进了示例代码。

第 18 章新增了介绍电子邮件附件以及如何同 IMAP 服务器交互的两节，还介绍了 Open URI 库。

第 19 章介绍了 Ruby on Rails、Nitro、Wee、IOWA 和其他 Web 工具，还介绍了 WEBrick 与 Mongrel。

第 20 章新增了讨论 Rinda 和 Ruby 元组空间实现以及与它们紧密相关的 Ring 的内容。

所有这些新增内容都是必不可少的吗？答案是肯定的。

本书英文版是用英语撰写的第二本有关 Ruby 的图书（第一本是 Dave Thomas 和 Andy Hunt 合著的 *Programming Ruby*）。本书经过精心设计，是 *Programming Ruby* 的补充而非重复，这是本书受欢迎的重要原因。

作者着手编写本书第 1 版时，还没有 Ruby 国际大会，RubyForge、ruby-doc.org 和 rubygarden.org 也未面世。除 Ruby 主网站外，Web 上基本没有 Ruby 资料，Ruby Application Archive 也只包含几百项。

那时，几乎没有有关 Ruby 的出版物（无论是在线的还是印刷的），每次刊登有关 Ruby 的文章都引起注意，并在邮件列表上公布和讨论。

众多常用的 Ruby 工具和库那时还没有出现。没有 RDoc；没有解析 XML 的 REXML，数学运算库也远没有现在丰富；缺乏对数据库的支持，也没有 ODBC。Tk 是用得最多的 GUI

工具包，最常见的 Web 开发方式是使用低级 CGI 库。

没有 Windows one-click 安装程序。Windows 用户通常使用 Cygwin 或基于 mingw 的编译器。

甚至没有最原始的 RubyGems 系统，查找、安装库和应用程序通常完全以手工方式使用 tar 和 make 进行。

那时没人听说过 Ruby on Rails，也没人使用术语 duck typing，没有用于 Ruby 的 YAML，也没有 Rake。

当时使用的是 Ruby1.6.4，已觉得它很酷，但 Ruby1.8.5（作者当前使用的版本）更酷。

Ruby 语法有些变化，但没有什么好说的。这些修改大多属于“边界状态”，比以前更合理。Ruby 的一个独特之处是，括号是可选的。在 98%的时间里，程序员不会注意到这种差异，但如果注意到了，将发现现在比以前顺畅和一致。

一些核心方法的语义也发生了变化，同样，这些大多是小变化。例如，以前 Dir#chdir 不接受代码块，但近年来能够接受了。

有些核心方法已淘汰或重命名。方法 class 不再有别名 type（因为在 Ruby 中通常不讨论对象的类型）；方法 intern 现在变成了更友好的 to_sym；Array#indices 现为 Array#values_at。

另外，还新增了一些核心方法，如 Enumerable #inject、Enumerable#zip 和 IO#readpartial。原来的 futils 库现为 fileutils，它有自己的模块命名空间 FileUtils，而不像以前那样将方法添加到 File 类中。

还有很多其他的变化，但这些修订都是经过谨慎考虑的。Ruby 依然是 Ruby，Ruby 的很多优点源于其精心而缓慢的变化，这些修订是松本行弘和其他开发人员的智慧结晶。

当前有大量有关 Ruby 的图书和文章，基于 Web 的教程和文档资源俯拾皆是。

新工具和库不断涌现，其中最常见的是 Web 框架和工具、博客工具、标记工具和对象—关系映射（ORM），当然还有数据库、GUI、数值计算、Web 服务、图像操作和源代码控制方面的工具和库。

Ruby 编辑器得到广泛而完善的支持，还有很有用且成熟的 IDE（其功能有一部分与 GUI 生成器重叠）。

不可否认，Ruby 社区也在不断壮大和变化。今天的 Ruby 决非小语言，美国宇航局、美国国家海洋和大气局、摩托罗拉和其他众多大型公司、机构都使用它，它被用于处理图形、数据库、数值分析和 Web 开发等诸多领域。总之，Ruby 已成为主流。

修订本书是令人享受的工作，希望读者能够从中获益。

如何使用本书

也许你无法通过阅读本书学会 Ruby，因为其中介绍性或指南性内容较少。如果读者是 Ruby 新手，也许应首先阅读其他图书。

尽管如此，程序员都有顽强的毅力，因此也可能能够通过阅读本书学会 Ruby，第 1 章确

实包含简介和教程。

第 1 章还包含一个完整的“起步”列表（要确保它最新很困难）。该列表的用处可能因读者而异，因为每个人对何为直观的理解不同。

本书主要旨在解答“如何做”的问题，因此读者可以根据兴趣选读其中的内容。如果每位读者都自始至终阅读每一页，作者将深感荣幸，但不希望如此。作者更希望读者通过浏览目录查找需要的技巧或感兴趣的内容。

第 1 版面世后，作者同很多读者有过交流，发现他们确实是从头读到尾。另外，有多位读者指出，他正是通过阅读本书学会 Ruby 的。可见一切皆有可能。

本书有些内容可能看起来很基本，因为读者的背景和经验各异，在有些人看来显而易见的东西在其他人看来未必如此。作者尽可能确保本书内容全面，同时尽量使其篇幅合理（这显然是个很有挑战性的目标）。

本书可视为“反向参考手册”，读者不应根据名称来查找方法和类，而应根据功能或用途来查找。例如，String 类有多个处理大小写的方法：capitalize、upcase、casecmp、downcase 和 swapcase。在参考手册中，很可能按字母顺序列出它们，但本书将它们放在一起。

当然，为确保完整性，作者有时也采用参考手册的做法。在很多情况下，作者将辅以比参考书更不寻常或更多元化的示例。

作者尽量追求较高的代码—注释率，从第 1 章看，这个目标达到了。作者可能变得唠唠叨叨，但程序员总是希望看到代码。

有些示例是虚构的，对此表示歉意。脱离现实问题说明技术或原理很困难，但任务越复杂越高级，作者越尽力使用实际解决方案来说明。因此，介绍字符串拼接时，可能使用包含 foo、bar 等平淡无奇的代码段，但介绍 XML 的解析等主题时，通常使用更有意义、更真实的代码段。

必须指出，本书有两三个独特之处。第一个独特之处是，避免使用类似 Perl 的“丑陋”全局变量，如$_等。Ruby 中有这些全局变量，它们的效果不错，大多数 Ruby 程序员在日常工作中都使用它们。但几乎在所有情况下都可避免使用它们，本书的大部分示例都是这样做的。

第二个独特之处是，即使独立表达式没有副作用也避免使用它们。Ruby 是面向表达式的，这很好，作者尽可能利用这一特点，但在代码段中，作者不编写这样的表达式，即其返回的值没有被使用。例如，表达式“abc” + “def”可用于说明字符串拼接，但作者将编写这样的代码：str = “abc” + “def”。这看似多余，但更自然，尤其是对关心函数是否为无效的 C 程序员（或以过程和函数方式思考的 Pascal 程序员）来说。

第三个独特之处是，作者不喜欢使用#来表示实例方法。很多 Ruby 程序员认为，使用“instance method crypt of class String”表示 String#crypt 显得啰嗦，但作者认为这样不会让人感到困惑（事实上，作者正慢慢转向后一种用法，因为这种用法显然不会消失）。

作者尽量包含指向外部资源的“指针”。虽然时间和空间不允许本书包罗万象，但作者希望通过告诉读者到哪里可以找到相关资料来弥补这种缺陷。网上的 Ruby Application Archive

可能是最主要的资源，本书将多次提到它。

通常需要事先说明代码使用的字体以及如何区分代码和正文，但作者不想侮辱读者的智商，因为读者以前肯定阅读过其他计算机书籍。

需要指出的是，本书大约10%的内容是他人撰写的，这还不包括编辑和排版工作。应该阅读本书（及每本书）的致谢部分，但大多数读者都跳过。现在就去阅读吧，它就像蔬菜一样对你有好处。

本书的源代码

本书所有重要的代码段都放在一个压缩文件中供读者下载，该文件可从网站www.awprofessional.com或作者的网站（www.rubyhacker.com）下载。

我们提供了该压缩文件的.tgz格式和.zip格式，其中的文件通常采用这样的命名约定：对于程序清单，使用其编号命名，如listing14-1.rb；对于较短的代码段，将使用页码和一个可选的字母命名，如p260a.rb和p260b.rb。很短或不能脱离上下文运行的代码段通常不包含在该压缩文件中。

何谓“Ruby之道”

那么，究竟什么是Ruby之道？作者认为这包含两方面：一是Ruby的设计理念；一是Ruby的使用哲学。显然，Ruby的设计和使用是相关的——无论是在硬件还是软件方面，否则为什么会有人类工程学呢？正是由于希望有人使用手柄，才在制造设备时安装一个。

Ruby有一种难以名状的品质，它不仅出现在该语言的语法和语义中，还出现在使用这种语言编写的程序中。然而，一旦人们想去辨别这种品质时，它就变得模糊起来。

显然，Ruby不仅是一种创建软件的工具，它本身也是软件。为什么Ruby程序要采用与解释器不同的法则？毕竟Ruby是高度动态和可扩展的。这两个层次不同可能是有理由的，也许是为了迁就现实世界的不便；但通常而言，思考过程可以且应该相同。Ruby可以使用Ruby来实现，虽然编写本书时还不是这样。

通常我们不会考虑“道”的起源，但它有两个重要含义。一方面，它表示方法或手段；另一方面，它也可以表示道路或路径。显然这两种含义是相互联系的，作者认为“Ruby之道”同时包含这两种含义。

因此，我们谈论的是一种思维过程，也是要遵循的路径。即使最伟大的软件大师也不能自称已臻完美，而只是在通往完美的路上。这样的道路可能不只一条，但这里作者只能谈论其中的一条。

有句名言：“功能决定形式。”（Form follows function。）传统上，这句名言是正确的，但

弗兰克·洛伊德·赖特曾说过："功能决定形式常被误解，形式和功能在精神上应为一体。"

赖特说的是什么意思呢？作者认为这一真理不是从书上可以学到的，而要在实践中才能体会。

赖特在其他地方用更容易理解的方式表述过该名言。他是一位简单性的倡导者，曾经说过："建筑师最有用的工具，是草稿板上的橡皮擦和工地的撬棒。"

Ruby 的优点之一就是简单性。在简单性方面，作者可再引述其他思想家的话。安东尼·圣·德克旭贝里说过："完美指的不是不能添加，而是不能再减少。"

但 Ruby 是一种复杂的语言，怎么能说它简单呢？

如果我们对宇宙有更深入的了解，将可能发现"复杂性守恒定律"——复杂性像熵一样干扰我们的生活，我们不能避免它而只能重新分配它。

这就是关键所在。我们不能避免复杂性，但可将其隐藏到看不到的地方，这就是"黑箱"原理。黑箱执行复杂的任务，但从外部看，它仍拥有简单性。

用爱因斯坦的话来结束引用吧："一切应尽量简单，但不能更简单。"

因此，从程序员（不包括维护解释器的程序员）的角度看，Ruby 体现的是简单性，还可以看到折衷的能力。在现实世界中，必须稍微让步。例如，Ruby 中的每个实体应是真正的对象，但有些值（如整数）存储为直接值。计算机科学的学生应熟悉折衷，数十年来我们就在设计的优雅与实现的可行性之间进行折衷。实际上，我们也在简单性与其他目标之间进行折衷。

拉里·沃尔关于 Perl 的阐述是正确的："使用小语言表达式，其含义很丰富；使用大语言表达式，其含义很小。"英语也如此。生物学家恩斯特·海克尔之所以能用"Ontogeny recapitulates phylogeny"这 3 个单词来概括胚胎重演律，是因为他以高度具体的含义运用了这 3 个单词。我们允许语言的内在复杂性，是因为它能够使我们的个体话语更简单。

这一原则可用这样的方式表达：10 行代码能解决的问题，就不要用 200 行。

作者认为"简洁即好"是理所当然的。短的程序段占据程序员的大脑空间较少，这样容易掌握整段代码，且编写代码时引入的错误将更少。

当然，必须记住爱因斯坦关于简单的警告。如果过于追求简单，最终写出的代码将非常混乱。信息理论告诉我们，经过压缩的数据在统计上类似于随机噪声。只要阅读一下编写糟糕的 C 或 APL 代码或正则表达式，就会对这个真理有切身体会。"简单，但不要过于简单"，这就是关键所在。拥抱简洁，但不牺牲可读性。

简洁和可读性都好，这一点人人皆知。但有一个根本原因，它是如此基本以至于我们有时会忘记，这就是计算机是为人类而存在的，而不是人类为计算机而存在。

在过去几乎是相反的。计算机的价格昂贵且非常耗电。人们对待计算机就像对待神一样，而程序员就像卑微的仆人一样。

当计算机变得越来越小、越来越便宜时，高级语言也越来越普及。从计算机的角度看，高级语言的效率低，而从人类的角度看，它们却是高效的。Ruby 是这种思想的产物。实际上有人将 Ruby 称为 VHLL（Very High-Level Language，超高级语言），虽然这个术语的定义不是很明确，但用在这里再合适不过了。

计算机应是仆人而不是主人，正如松本行弘所言，聪明的仆人应该只要短短几条命令就可执行复杂的任务。整个计算机科学的历史都如此，从机器语言发展到汇编语言，再到高级语言。

这里讨论的是从面向机器的观念向面向人类的观念转变。作者认为，Ruby 是面向人类编程的典范。

先切换一下话题。20 世纪 80 年代有一本很精彩的小书叫做《编程之道》（*The Tao of Programming*，杰弗里·詹姆斯著）。书中几乎每一行都是名言，这里只重复一句："程序应遵循最小惊讶法则（Law of Least Surprise）。什么是'最小惊讶'法则？很简单，就是程序对用户的响应尽可能不使用户感到惊讶。"（当然，语言解释器的用户是程序员。）

不知道这个术语是不是詹姆斯杜撰的，我在他的这本书中首次见到该术语。这是一个著名的原则，在 Ruby 社区中经常被引用，但通常称它为"最小惊讶原则"（Principle of Least Surprise），缩写为 POLS（作者更喜欢缩写 LOLA）。

不管怎么叫，这条规则都成立，它已成为贯穿整个 Ruby 语言发展过程的指南，也是库和用户界面的开发指南。

当然，唯一的问题是，使不同的人感到惊讶的东西不同，对象或方法的行为应是什么样的没有统一看法。人们可以努力达成一致，使设计决策合理，而每个人可训练自己的直觉。

松本行弘曾表示，"最小惊讶"应针对设计师（即松本行弘）。你越是和他想的一样，Ruby 就越不会使你感到惊讶。可以肯定，对大多数来说，模仿松本行弘不是件坏事。

无论系统的构造多么合理，直觉都需要训练。每种编程语言本身是一个世界，有自己的一套假设，人类语言也一样。作者在学习德语时，知道所有名词都是大写的，只有 deutsch 不是。我向教授抱怨：这毕竟是语言的名称啊，不是吗？他微笑着说："顺其自然就好。"

他教会我的是让德语成为德语。推而广之，对从其他语言转向 Ruby 的人来说也如此：让 Ruby 成为 Ruby。不要指望它是 Perl，因为它不是；也不要指望它是 LISP 或 Smalltalk。另一方面，Ruby 有这三种语言都有的要素。开始按照期望来做，但发现并不是期望的那样时，就顺其自然（除非松本行弘认为确实需要修订）。

现在每个程序员都知道正交原则（称为正交完备性原则更准确）。假设有一对轴，一条轴表示可比较的语言实体集，另一条轴表示属性或功能。"正交"通常指由这两条轴定义的空间是"满"的，可以合理地利用。

Ruby 之道的一部分就是要追求这种正交。数组在一定程度上与散列类似，因此对它们执行的操作也应类似。进入两者不同的领域后，就达到正交的极限。

松本行弘说过，"自然"应优先于正交。但要完全理解什么是自然、什么不是，需要进行一定的思考和编程。

Ruby 力图使程序员感到友好。例如，有很多方法名有同义词或别名，size 和 length 都返回数组包含的元素数，indexes 和 indices 指的是同一个方法。有人认为这会造成困扰或起反作用，但作者认为这是良好的设计。

Ruby 力图实现一致性和规律性。这没有什么神秘之处，在生活各个方面，我们经常要求

事物规则、类似。比较困难的是学习何时打破这条原则。

比如，Ruby 习惯在断言式方法后加一个问号（?）。这是个好主意，它使代码更清晰，使命名空间更容易管理。但感叹号的用法比较有争议的，它用于具有“破坏性”或“危险性”的方法，这些方法会修改接受方。之所以引起争议，是因为并非所有具有破坏性的方法都有这样的标记。那么是不是应该一致呢？

不，其实不应该。有些方法修改接受方（如 Array 的方法 replace 和 concat），而有些方法是给类属性赋值的 writer，不应在属性名或等号后加上感叹号。有些方法（如 read）修改接受方的状态，这种情况很常见，不能用这种方式标记。如果每个破坏性方法的名称后都加上感叹号，程序很快将变成多级营销公司的销售手册。

现在是否感到了一种似乎所有规则都会被打破的对立势力？这里引用富尔顿第二定律：每条规则都有例外，富尔顿第二定律除外（不错，这是个小笑话）。

Ruby 既不是愚蠢地追求一致性，也不拘泥于一套简单规则。事实上，Ruby 之道的一部分就在于它不是僵化死板的。松本行弘曾说过，在语言设计上，应“随心而行”。

Ruby 哲学的另一个方面是：不要害怕运行时的改变，不要害怕动态的东西。世界是动态的，为什么编程语言就应该是静态的呢？Ruby 是世界上动态程度最高的语言之一。

作者还想指出另一点：不要成为性能问题的奴隶。当性能无法接受时，这种问题必须解决，但通常不应首先考虑性能。如果效率并非至关重要，就应将优雅放在效率之前，然而，编写可能以无法预测的方式使用的库时，一开始就要考虑性能。

我在 Ruby 中看到的是不同设计目标之间的平衡，它反映了物理学中 N 体问题的复杂交互，它可能是根据 Alexander Calder 汽车设计的。也许这种交互、这种和谐体现了 Ruby 哲学。程序员知道他们的技巧不只是科学和技术，也是艺术。我不敢说计算机科学中有什么精神方面的东西，但你我之间肯定有（如果没有看过 Robert Pirsig 的 *Zen and the Art of Motorcycle Maintenance*，建议读一读）。

Ruby 源于人类创造有用而优美的事物的欲望，对作者来说，这就是 Ruby 之道的精髓。

目　录