PEARSON
Addison Wesley

Rails之道

THE RAILS WAY

[美] Obie Fernandez 著
Ruby on Rails中文社区 译

Ruby on Rails 经典参考用书

Jolt年度技术图书大奖获得者

深入讲解Ruby on Rails框架中的各种知识点和应用新特性
Rails各种API函数参考大全
Ruby on Rails中文社区倾情奉献
一本书，让你参透Ruby on Rails的世界

人民邮电出版社
POSTS & TELECOM PRESS

Rails之道
THE RAILS WAY

[美] Obie Fernandez 著
Ruby on Rails中文社区 译

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

Rails之道 / (美) 费尔南德斯 (Fernandez, O.) 著 ; Ruby on Rails中文社区译. -- 北京 : 人民邮电出版社, 2010.4
ISBN 978-7-115-22072-1

Ⅰ. ①R… Ⅱ. ①费… ②R… Ⅲ. ①计算机网络－程序设计 Ⅳ. ①TP393.09

中国版本图书馆CIP数据核字(2009)第244473号

Rails 之道

◆ 著 [美] Obie Fernandez
译 Ruby on Rails 中文社区
责任编辑 屈艳莲
执行编辑 汪 振

◆ 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
三河市海波印务有限公司印刷

◆ 开本：787×1092 1/16
印张：33
字数：818 千字 2010 年 4 月第 1 版
印数：1－2 000 册 2010 年 4 月河北第 1 次印刷

著作权合同登记号 图字：01-2008-5889 号
ISBN 978-7-115-22072-1

定价：89.00 元

读者服务热线：(010)67132705 印装质量热线：(010)67129223
反盗版热线：(010)67171154

内容提要

本书按照 Rails 的各个子系统进行组织编排，分别介绍了 Rails 的环境、初始过程、配置和日志记录，Rails 的分配器、控制器、页面生成和路由，REST、资源和 Rails，ActiveRecord 的基础、关联、验证和高级技巧，ActionView 的模板、缓存和帮助器，Ajax、Prototype 和 Scriptaculous 等 JavaScript 代码库和 RJS，Session 管理、用户登录和认证系统，XML 和 ActiveResource，后台处理和 ActionMaile，测试和 specs（包括 RSpec on Rails 和 Selenium），安装、管理、编写插件，Rails 的生产部署、配置和 Capistrano 等内容。

本书详细讨论了 Rails 的程序代码并通过分析 Rails 中的代码片段来深入解释它的功能，同时，本书部分章节也摘录了一些 API 文档中的内容，使读者能够快速地找到对应的 API 文档、相关的示例代码以及深入的解析说明。

本书是 Rails 的权威参考书，适合对 Rails 已经有一定了解的开发人员学习和使用。

序

Rails 不只是一个用于创建网络应用程序的编程代码框架，更是一个帮助思考网络应用程序的框架。Rails 并不是力图面面俱到而平淡无奇的软件，相反它牺牲了面面俱到的灵活性以换得解决“大多数人花费大多数时间去做的大多数事情”——便捷。对于软件的设计者们而言，Rails 就像一件紧身衣，它帮助你把全部精力投入到核心业务上去，并为你处理那些与业务无关的琐事。

要接受这种工作方式，不仅需要知道如何在 Rails 中做这些事情，而且更需要了解 Rails 为何如此为之。当理解这些“为什么”之后，你便能顺畅地使用这个程序框架进行工作而不是抗拒它。这并不意味着你必须接受某些选择，而是说你需要同意“采用约定”的这个原则。为了获得出众的开发效率，有些人可能不得不放弃一些工作中的个人癖好，开始学会放松并遵循 Rails 中的约定。

这是一本真正能够帮助你做到这些的书。不仅是因为本书是一部全面介绍 Rails 各功能的工具书，同时它为你开启一扇可以了解 Rails 的思想和灵魂的窗户——为什么我们会选择这些约定的方法来做事？又为什么我们不苟同一些已经被广泛使用的方式？本书还将讲述直接从帮助 Rails 成长的社区成员口中得知的 Rails 之所以能成为今天的 Rails 的故事。

学习如何在 Rails 中编写“Hello World”是一件每个人都能自己完成的易事，但是要了解和掌握 Rails 的全面形态可要困难得多。我衷心感谢 Obie 的工作，他使这本书成为读者学习和应用 Rails 的旅程中的好伙伴。愿你能喜欢此书。

——David Heinemeier Hansson
Ruby on Rails 的创造者

致　谢

首先，诚挚地感谢我的编辑，Debra Williams Cauley，因为她肯定了我的潜能，并且在过去一年半的时间里耐心地陪伴我度过了所有曲折，直到此书完成。她也是我遇到的最体贴最聪明的女性之一，我期待能和她建立长期的友谊，共同出版更多的作品。我在 Addison-Wesley 的团队也一直很耐心，为我提供了极大的帮助和鼓励，他们是：San Dee Phillips、Songlin Qiu、Mandie Frank、Marie MaKinley 和 Heather Fox。

当然，由于我专注于本书的工作，我的家庭承担了很多的压力，特别是在最后六个月，截稿压力越来越大的时候。Taylor、Lian 和 Desi：我深爱你们。谢谢你们给了我足够的时间和安静的环境去写作。我还要好好地感谢我长期的伴侣（也是我最喜欢的 Rails 开发人员）Desi MacAdam，这段时间她承担了几乎所有我应该承担的家庭责任。我还要感谢我的父亲 Marco，因为是他充满远见地建议我写作一整套 Ruby 系列的丛书而不仅仅是一本书。

凭一己之力当然不可能完成这样一本范围如此之广、内容如此之丰富的书——我要感谢许多的供稿人和审稿人。David Black 提供了有关路由器和控制器方面的资料，帮助我开始了这本书的写作。大师系列的作者 James Adam、Trotter Cashion 和 Matt Pelletier 分别就插件、ActiveRecord 和产品布局等内容提供了帮助。Matt Bauer 帮助我完成了关于 Ajax 和 XML 的章节，Jodi Showers 编写了关于 Capistrano 的章节。Pat Maddox 将我从一次写作瓶颈中解救了出来，并帮助我完成了关于 PRpec 的内容，而 David Chelimsky 则帮我进行了专业的审阅。Charles Brian Quinn 和 Patric Naik 就后台进程等内容提供了及时的建议和帮助。Diego Scatalini 则帮助审阅了后面的一些章节。

Francis Hwang 和 Sebastian Delmont 从本书一开始写作时就对本书的技术内容进行了准确的审阅。Wilson Bilkovich 和 Courtenay Gasking 除了提供了原创性的写作内容和本书工具栏的内容外，还做了大量工作以帮助我赶上最后的截稿日期。特别是 Wilson，他在时间安排方面的机智和诙谐，总是令我发笑，他甚至重写了全书的最后一章以加入自己在这方面的专业知识。其他宝贵的审稿人包括：Sam Aaron、Nola Stowe 和 Susan Potter。在最后几周里，我的新朋友和编程伙伴“云雀”Jon Larkowski 找出了其他审稿人忽略的一些问题。

“声名狼籍”的 Zed Shaw 在我写作本书的很长一段时间里都是我的室友，不断为我提供启发和动力。同时，还要大力感谢在 caboose 在线聊天频道的朋友们不断提供专业的观点和看法。其中一些人的名字上面已经提到了，另外还有（排名不分先后）：Josh Susser（hasmanyjosh）、Rick Olson（technoweenie）、Ezra Zygmuntovich（ezmobius）、Geoffrey Grosenbach（topfunky）、Robby Russel（robbyonrails）、Jeremy Hubert、Dave Fayram (kirin-dave)、Matt Lyon（mattly）、Joshua Sierles（corp）、Dan Peterson（danp）、Dave Astels（dastels）、Trevor Squires（protocol）、David Goodlad (dgoodlad)、Amy Hoy（eriberri）、Josh Goebel（Dreamer3）、Evan Phoenix（evan）、Ryan Davis (zenspider)、Michael Schubert、Cristi Balan (evilchelu)、Jamie van Dyke（fearoffish）、

Nic Williams（drnick）、Eric Hodel（drbrain）、James Cox (imajes)、Kevin Clark（kevinclark）、Thomas Fuchs（madrobby）、Manfred Stienstra (man-fred-s)、Pastie Paste Bor（pastie）、Evan Henshaw-Plath(ramble)、Rob Orsini(rorsini)、Adam Keys(therealadam)、John Athayde (boborishi)、Robert Bousequet, Bryan Helkamp(brynary)和 Chad Fowler。此外，David Heinemeier Hansson（nextangler）也一直在鼓励我，甚至帮我解决了一些非常棘手的问题。

我在 ThoughtWorks 的许多前同事也间接地促成了本书的完成。首先，Martin Fowler 不仅是我的榜样和灵感来源，也帮助我维护了与 Addison-Wesley 的关系，并且在项目前期我自己还毫无头绪的时候期提供了许多个人建议和支持。ThoughtWorks 的 CEO Roy Singham 在 2005 年初我将他介绍给 David Hansson 时就看出了 Rails 的潜力，此后他一直坚定地支持我推广 Ruby（他也在自己的圈子里进行了不少推广工作）。这些在项目初期是至关重要的，因为当时 ThoughtWork 的很多其他受人尊敬的人认为 Rails 不过是炒作，不会坚持多久。

我还需要感谢很多来自 ThoughtWorks 的工作人员。首先，我很骄傲能够教导 Jay Fields 有关 Ruby 的知识,并以他这个实例证明了青出于蓝而胜于蓝的道理。起初我并不喜欢 Ruby,而 Carlos Villela 是令我再次对它产生兴趣的原因之一。在 ThoughtWorks，我的搭档 Upendra Kanda 和我独自为 Rail 的第一个付费客户的项目工作了好几个月。非凡的电脑黑客 Michael Granger 在创意性使用 Ruby 方面教了我很多东西。Fred George 是我的导师和灵感来源。Steve Wilkins 和 Fern Schiff 是将 Rails 商业化的最初支持者。其他 ThoughtWorks 的 Rails 热心支持者在不同程度上提供了帮助，包括：Badri Janakiraman、Rohan Kini, Kartik C、Paul Hammant、Robin Gibson、Nick Drew, Srihari Srinivasan、Julian Boot, Jon Tirsen、Chris Stevenson、Alex Verkhovsky、Patrick Farley、Neal Ford、Ron Campbell、Julias Shaw、David Rice、Jeff Patton（赶快完成你的书！）、Kent Spillner、John Hume、Jake Scruggs 和 Stephen Chu。

我在 Rails 上卓有成效的工作离不开客户方面的“进步思想者”和“风险家”们，他们信任我和我的团队能够以还没有获得认可的技术提供方案。我无法一一列出他们的名字，但是其中一些相当突出：我要感谢 Deere 的 Hank Roark、Tom Horsley、Howard Todd、Jeff Elam 和 Carol Rinauro 提供的坚定支持和信任；我还要感谢 Barclays 的一位朋友 Robbert Brown 的远见卓识，他一直在鼓励我。还有，同 Rackspace 的 Dirk 和 Brett Elemendorf 一起工作实在是太棒了。

在完成此书的过程中，InfoQ.com 一直非常合作，也很理解时间的紧张。在此，特别感谢 Floyd Marinescu 和 Diana Plesa 的耐心，也要感谢我的 Ruby 写作团队，包括 Werner Schuster 和 Sebastien Auvray。

从 2007 年 1 月开始，我的朋友 Mark Smith 为我提供了和一组杰出人才在佛罗里达州阳光灿烂的大西洋海岸边共同开发 Web 2.0 项目的乐趣。First Street Live 的团队从始至终一直在大力支持我，他们是：Marian Phelan、Ryan Poland、Nick Strate、Joe Hunt、Clay Kromberg、Dena Freeman 和其他所有人（Mark，我爱你，我们永远都是朋友。感谢那有趣的工作环境、读书会和不间断的智力激荡。如果没有去那里为你工作，我无法如此成功地完成本书）。

最后但同样重要的是，我还要真诚感谢 PragDave Thomas。他曾经说过：“如果让 Obie 写一本关于 Rails 的书，那就像是让萨德侯爵写一本有关餐桌礼仪的书一样。”

作者简介

Obie Fernandez 是一位广为人知的技术行业领袖和独立咨询师。从 20 世纪 80 年代获得第一台 Commodore VIC-20 开始，他就一直在从事各种黑客工作。20 世纪 90 年代中期，他终于找到了自己的位置，成为第一代 Java 企业项目的编程师。他于 1998 年移居到美国乔治亚州亚特兰大市，并作为当地新兴企业 MediaOcean 的首席架构师而闻名。他还成立了 Extreme Programming（后改名为 Agile Atlanta）用户社团，并在该社团担任了几年的主席和组织人。2004 年，他又回归企业，为世界著名的咨询公司 ThoughtWorks 处理那些具有很大发展潜力的高风险项目。

从 2005 年初开始，他就通过博客和出版物推广 Ruby 和 Rails，并且在 Java 开源社区的老朋友之间获得到了一定认可（当然也有指责）。从那时起，他就定期参加一些业界活动和用户会议，有时也会为想要参与 Rails 开发的公司或组织提供一定培训。

目前，Obie 致力于开发和营销大规模的基于网路的应用程序。他几乎每天都会更新自己深受欢迎的科技博客 http://obiefernandez.com，就各种话题进行讨论。

前　言

在2004年末，我正在为一家大型美国汽车制造公司做咨询服务。当时和我一起工作的还有我的好朋友Aslak Hellesoy。我们所接手的那个项目很具挑战性，不仅面临困难的公司政治矛盾、技术障碍，而且工期很紧。这个项目的工期还不是一般意义上的紧张，在当时的项目情况下，如果我们每拖延一天完工，我们的客户就得被罚100万美元。因此压力可想而知！

在那个项目中我们曾面临一个具有争议的抉择，不过最后我们的团队决定将我们的连续整合系统构建在Aslak的一个业余项目DamageControl的基础上。DamageControl是老旧的CruiseControl服务器的Ruby版本，是由我们的雇主公司ThoughtWorks开发的。

问题是DamageControl当时还是一个半成品，而且和其他许多Ruby事物一样，它在Windows系统中兼容性不是很好。我已经记不得是什么原因，不过当时我们必须将它部署到一个陈旧的Windows 2000服务器上，而且那个服务器上还跑着一个StarTeam代码版本库（折腾！）。

Aslak需要帮助——在这之后的几个星期，我们没日没夜地进行结对编程的工作。对DamageControl的程序代码和Ruby的Win32程序代码库的C代码进行了大量改造。当时的我具有八年正规行业的Java开发经验，而且当时我特别喜欢使用IntelliJ IDE；但在那段时间，我对Ruby的痛恨不禁溢于言表。

当这个倍具压力的项目上线之后，我接手了另一个相对轻松的任务并转去ThoughtWorks的伦敦办公室工作。过了大概一个月的时间，Ruby再一次吸引了我的注意。当时我在很多朋友的博客上看到他们激动地讨论一个即将发布的网络程序框架，叫做Ruby on Rails。于是我决定再来试试Ruby，或许它并非那么糟糕吧。我很快就创新地为ThoughtWorks建立起一个供内部使用的社区网络系统。

在2005年初的那几周中，我的Rails第一次亲密接触让我完全转变了看法。经年累月以来我所学到的那些用于构建网络应用程序的经验和技巧已经全部被浓缩到了这么一个程序框架中，更何况这个框架是用我所见过的最优雅而简练的编程语言编写的。我对Java的兴趣戛然而止（虽然之后我还差不多用了一年的IntelliJ）。我开始热心地在我的博客上撰写有关Ruby和Rails的文章，并且不遗余力地在ThoughtWorks的里里外外宣传它们。结果，正如他们所说的，Ruby on Rails是件影响网络应用程序开发的大事记。

当我在2007年撰写这本书的时候，ThoughtWorks全球收入中的近一半来自于我所推荐的Rails的项目生意。他们特别成立了一个很大的部门专门从事基于Ruby的商业软件开发工作。他们的产品中包括CruiseControl.rb，它很荣幸地成为Ruby on Rails核心团队所选择的连续式整合服务系统。我猜CruiseControl.rb就是Aslak一直以来都想编写的系统。

Ruby on Rails

为什么有那么多深具企业经验的开发者爱上 Ruby on Rails 呢？在项目实施中，采用 Java 或者 Microsoft 技术的解决方案的复杂程度都难以让人接受。过度的复杂程度使个人失去对于项目的理解，同时大幅增加了团队间的沟通需求。此外，一味强调遵循设计模式加上对性能的执迷都使得使用那些平台的开发者很快失去了对于开发应用程序的兴趣。

Rails 社区不存在此类压力。DHH（David Heinemeier Hansson）选择了一个让他感觉舒畅的语言。Rails 的产生是因为他追求代码完美。类似这样的事情定下了 Rails 社区的基调。关于 Rails 的种种都充斥着主观成分。大家要么能“分享”这种愉悦要么便无法体会。只是选择了 Rails 和不选择的人之间没有恶意的寻衅，所有的只是温文尔雅的鼓励。

——Pat Maddox

Ruby 很美妙，编写 Ruby 代码更是件美妙的事。我认识每一个选择 Ruby 语言的人都告诉我他们的工作比以前轻松开心了很多。就冲着这一点，Ruby 和 Rails 正在不断动摇那些安于维持现状的程序员们，特别是在企业应用方面。在接触 Rails 之前，我习惯于从事那些要求反复而又和真实世界没什么关系的项目。我受够了在各种相似的代码框架中挑来选去进行整合，也受够了编写丑陋的代码。

相反，Ruby 是一门美妙动态的高级语言。Ruby 代码易读易写，因为它的代码类似人类语言的方式，更贴近于我们所要解决的问题。这种特别强的可读性带来了许多长期和短期的好处，因为程序代码迟早要被转入生产环境并且必须被维护人员彻底理解。

我的经验告诉我，和 Java 或者 C#比起来，处理相同的问题，使用 Ruby 来编写程序代码量要少得多。从程序的长期维护角度来看，较小的代码库更易于维护。更重要的是长期维护被广泛认为是成功的软件项目中最大的成本。此外当出现问题时，较小的代码库更便于迅速找出问题所在，甚至是在不借助花哨的调试工具的情况下就能解决。

Rails 的兴起和被主流软件世界的接受程度

和敏捷开发原则的出现一样，Rails 所注重的是我们作为程序开发者的需求，而不是作为软件工程师的需求，更不是作为计算机科学家的需求。Rails 通过竭力减少和程序开发无关的复杂性以便开发者紧紧把握住项目成功这个终极目标。使用 Rails 进行编程不仅让人享受到开发的乐趣，更增强人想要成功的信心！

Rails 提供非常完整的平台性的工具和技巧，从而鼓励我们将精力全部放在实现项目的商业价值上。Ruby 所秉持的“最少意外（Least Surprise）”原则已被完美地诠释在 Rails 那简单而优雅的设计之中。而且最好的是，Rails 是一个完全免费的程序，这意味着当遇到其他程序框架都无法胜任的问题时，你可以自由地在 Rails 的源代码中找到解决问题的答案。

David 曾偶尔表示出他对于 Rails 被主流软件世界的接受程度并不是特别感兴趣，因为那可能会伤害到早期用户所热爱的竞争势态。那些早期用户主要是从事网站设计和开发的个人和小团体，其中有很多来自于 PHP 世界。

被企业所采用

你尽可以称我为一个理想主义者，因为我始终相信即便是大型而保守的企业中的开发者，当他们有机会用到这样的工具时，也绝对会积极地希望让他们自己的动作变得更高效而且更具创新性。这就是为什么几年来越来越多的这类开发者转而投入 Rails 世界。

或许企业开发者最终将成为 Ruby on Rails 最热忱和最有发言权的用户，因为他们这个群体是在如今这种多变的行业环境下失去最多的人。一直以来他们都是大规模裁员时优先考虑的目标，同时又是那些（基于“项目定义比实现更重要”或者“代码实现应当是机械化的琐事”等假设上的）糟糕的外包工作失败时的替罪羊。

项目定义真比代码实现更重要吗？对大多数项目来说不是如此。代码实现仅仅是项目中无足轻重的部分吗？当然不是！软件开发中有许多显著的困难，特别是在以下企业环境中。

- 难以理解的早期系统。
- 高复杂性的业务问题，比如投资银行。
- 项目参与者和业务分析员对客户需求理解有偏颇。
- 管理人员因担心被削减年度预算而对提高效率的改变持反对意见。
- 积极破坏你的项目的最终用户。
- 公司政治！枪打出头鸟。

作为一个服务于世界 1000 强企业的商业顾问，近十年来我每天都生活在这样的环境中。最终我悟出了一个道理，原来另有更高妙的生存之道。那个高招不仅能够使你免受公司政治的影响，而且保证为你顺利地带来新的机遇。

这个高招就是成为一名出众的员工！具体而言就是工作效率。我的意思是使你在工作中具有其他人都比不上的效率，从而再也没人能把你当替罪羊。当然在没有把握的情况下去尝试这样做无疑于自毁前程。在此我所指的是通过培养高效和有效的工作方式来提高你的工作成果，因此即便是你团队中最顽固和最愤世嫉俗的人都不得不钦佩你的能力。也就是说经常花些时间不断完善你所开发的应用程序，让使用程序的最终用户越来越喜欢使用它。

一旦成为具有这般出众能力的员工，你便是团队中能够让客户满意的救星。客户满意了就会乖乖地及时付钱。这样你也不会在每年的公司人员调整中受到影响，因为公司管理人员都会觉得：“我们实在不能失去这样的人才。”

话说到此我得打住了。我并不介意在听了这些话后，你仍觉得我是一个怀疑论者。不过以上我所说的这些都不是凭空想象，我只是在描述掌握了 Ruby on Rails 之后的我的生活。我写这本书的目的就是希望 Ruby on Rails 也能成为你的（或许不用那么秘密的）秘密武器，让你在这个混乱的软件开发世界中游刃有余。

注重结果

在本书中我所能为你做的以及基于我自己的经验和行业知识所能为你指出的，就是如何使用 Ruby on Rails 在你的项目中高效地产生所预期的结果。这本书能把你武装起来，给你弹药，帮你在面临技术抉择时有的放矢，更帮你抵挡那些在项目中迟早会出现的质疑。不过话说回来，林子大了，什么鸟都有，在现实项目中，一招必杀只是一个美好的童话。因此在本

书中，我更将为你指明在哪些情况下选择 Rails 将可能会是一个错误。

在书中，我将逐个深入地分析 Rails 的组成部分，并且讨论如何在需要加强功能时扩展这些组件。Ruby 是一门超级灵活的语言，这意味着你可以找到无数的途径来自定义 Rails 的功能。正如你将看到的，Ruby 的原则是在你遇到问题时，让你能就近自由地找到解决它的最佳方法。

作为一本参考书，本书将完整地介绍 Rails API 以及对 Ruby on Rails 应用开发者有用的 Ruby 语言特性、设计方法、代码库和插件。

关于偏执的软件（Opinionated Software）

在继续深入之前，有一点我必须提醒你。Rails 之所以获得如此成功是因为它是一款偏执的软件，由偏执的程序员所编写。同理可知，这本书也是一本偏执的书，由偏执的作者所撰写。

以下是一些影响本书写作思路的观点。你当然无需全盘同意这些观点，只是请注意它们对于本书的影响。

- 项目成功的各种关键因素中，至关重要的是开发者的积极性和高工作效率。
- 保持积极性和高工作效率的最佳方式是注重实现产品的商业价值。
- “性能”的意思是在“给定资源的情况下执行得越快越好”。
- “可扩展性”的意思是“在所需的最多资源情况下以所需的最快速度执行”。
- 如果无法被扩展，那么性能只是空谈。
- 如果扩展的成本很低，那么从硬件中挤出性能永远都不是你的首要任务。
- 在这个行业中将可扩展性和开发工具挂钩是一个普遍性错误，此外大多数软件并没有极端的可扩展性需求。
- 软件的性能是和所选择的语言以及开发工具有关，因为越高级的语言越易读易写。一个普遍的共识是大多数软件中的性能问题源自于糟糕的程序代码。
- 约定优于配置是编写软件的更好方式。庞杂的 XML 配置文件必将被取缔！
- 代码的可携带性，也就是说可以将代码放到不同的硬件平台中运行，这并非特别重要的事情。
- 即使一种解决方案只能适用于某一平台，让问题得到妥贴的解决也是更重要的。如果你的项目最终失败了，那么可携带性只是空谈。
- 数据库的可携带性，即指让同一套程序代码在不同的关系型数据库上运行的能力，这更不是一件重要的事情，而且几乎不会被要求去实现。
- 界面是非常重要的，即便是小项目也不例外。如果你的程序“长相”难看，那么所有人都会假设它的代码质量也不会太好。
- 光靠技术来解决商业问题一般来说不是个好主意。然而，永远不能以此为借口而使用劣等的技术。
- 程序模组的通用化所带来的好处是有待商榷的。各种独立的项目都包含非常具有针对性的商业需求和宽泛的且各不相同的基础层需求，这会使得模块化的程序模组难以满足所有的实际需要。

呵呵，这么多意见。不过别担心，《Rails 之道》主要还是一本参考手册。更何况在 Rails 的介绍方面，目前也只有这一本参考手册。

关于本书

这本书不是一本关于 Ruby on Rails 的教程或者入门手册，而是为全职的 Rails 开发者的每日工作所服务的。在本书中我们钻研 Rails 的程序代码并通过分析 Rails 中的代码片段来深入解释它的功能。对于 Rails 有一定了解的读者可以直接以本书作为开始，辅之以网上丰富的资料，来完全掌握 Rails。而对于初学者而言，本书可能略显枯燥，或许其他一些科普性的出版物更为适用。

我是一个全职的 Rails 应用程序开发者，本书的其他参与者也都是。我们并非整体写书或者指导别人，虽然这正是我们现在所乐于做的。

我主要是为了我自己而开始写这本书，因为我讨厌使用在线文档，特别是需要不断翻查的 API 文档。由于 Rails 的 API 文档慷慨地开放版权（Rails 的其他一切都是），所以本书的一小部分章节中摘录了一些 API 文档中的内容。因此在本书出版时，相关的 API 文档有可能已经被扩充、调整或者加上额外的示例以及在实际应用中的经验总结。

希望你能和我一样享受到这本书带来的方便，因为我非常喜欢把类似这样的工具书放在案头，在书中加入注释，用书签和折角标识出正用到的章节。当我编写代码时，我希望能快速找到对应的 API 文档、深入的解析说明和相关的示例代码。

本书结构

我希望能够把这本书写成最好的 Rails 参考书，同时也努力将书中的内容以最自然的方式进行组合。所以经过周详的思考，我决定本书的结构根据 Rails 的每个子系统来进行编排，并在内容中穿插入详细的 API 文档。而每一个章节在具体写作时又各自略有不同。我认为如今对 Rails 的所有主题进行深析已经不是一本书能够容纳的了。

请相信我，要做到让所有人对本书的结构皆大欢喜可不是件容易的事，特别是当我意外地发现有几位读者纳闷本书竟然不是开篇先介绍 ActiveRecord。对我而言，Rails 最初只是一个网络程序框架，它其中的控制器和路由的实现才是最特别、强大又高效的功能。紧接其后的才是 ActiveRecord。

所以，这本书采用以下的结构。

- Rails 的环境、初始过程、配置和日志记录。
- Rails 的分配器、控制器、页面生成和路由。
- REST、资源和 Rails。
- ActiveRecord 的基础、关联、验证和高级技巧。
- ActionView 的模板、缓存和帮助器。
- Ajax、Prototype 和 Scriptaculous 三个 JavaScript 代码库和 RJS。
- Session 管理、用户登录和认证系统。
- XML 和 ActiveResource。
- 后台处理和 ActionMaile。
- 测试和 specs（包括 RSpec on Rails 和 Selenium）。
- 安装、管理、编写你自己的插件。
- Rails 的生产部署、配置和 Capistrano。

示例代码和列表

本书所选的示例代码对于所有的专业开发者来说应该都不会陌生。这些示例代码包括时间和成本跟踪、局部数据管理和博客程序。我不会浪费篇章来讨论和本书主题无直接关系的各种细节（比如这些示例程序在实现商业逻辑时的不同细节，或者评判他们的程序设计差异）。在这方面我追随我的老同事 Hal Fulton 和他所撰写的《The Ruby Way》（中文版由人民邮电出版社出版）的风格，大多数的代码片段都只是整篇代码中和主题相关的那一部分。其他不相关的部分都被代之以省略号。

在少数情况下，有那么几个示例代码比较长也比较重要，可能会在自己的程序中完整地用到它们。遇到这种情况，我都在示例代码之前特别加入标题，不过这类的示例代码并不多。当然所有这些示例代码无法拼凑出一个完整的程序系统，附录中那 30 页长的示例程序也不能。这些代码是用来给你在真正的工作中带去灵感。不过请记住，这些示例代码都是经过精简的，并不涵盖现实工作中的方方面面，比如控制器的示例代码中就没有翻页功能，因为那并不是我们所讨论的主要问题。

插件

当你发现自己正在编写的代码看上去像是一个插件，换句话说你所编写的代码和你的程序所面对的商业问题毫无关系时，那么你可能做了重复的工作。我希望当你产生这种感觉时这本书恰巧就在你的案头。对于正在做的工作，你几乎始终能够找到几个新的 Rails API 或者第三方插件来完完全全地完成这些工作。

本书的一个特点是在有可用的插件时，都尽可能给出充分地交代。我还特别介绍了一些我觉得高效的 Rails 开发工作不可或缺的插件。而且当第三方插件所提供的功能优于 Rails 的内建功能时，我们就直接介绍第三方插件（翻页插件就是一例）。

普通的开发者会发现使用了 Rails 之后自己的开发效率会翻倍，而且我看到很多专业的 Rails 开发者的效率也提高了很多。这主要是因为我们恪守“不重复自己（DIY）”的原则，以及由此原则直接衍生出的“不重复发明轮子（DRTW）”原则。重新实现一个已经被很好地实现了的功能虽然可能会让人觉得有所挑战，不过也实在是在浪费时间，更何况 Ruby 编程中还有更多的乐趣等你去发掘。

Ruby on Rails 实际上是由核心代码、官方插件和第三方插件所组成的庞大的系统。这个系统不断地以迅猛的速度在壮大，并且为你所要开发的最复杂的企业级网络应用程序提供所有的基础技术。我的目标就是为你提供足够的知识以免你重复发明轮子。

关于 David Heinemeier Hansson 亦称 DHH

2005 年初，Rails 还不像现在这样火爆时，我有幸同 Rails 的创造者——David Heinemeier Hansson 成为朋友。我和 David 的友谊是推动我撰写此书的一大原因。David 的许多观点对 Rails 社区具有深远的影响，这意味着在本书中，当我们在讨论 Rails 的本质和如何有效地使用它时，我会免不了引用一些 David 的看法。

David 有几次告诉我他讨厌大家给他起的这个“DHH”绰号用来代替他那个又长又难读

的名字。出于这个原因，在这本书中，凡是我引用他的话时，我一律使用“David”，而非“DHH”。但你在书中看到“David 说……”这样的语段时，我所指的那个 David 是独一无二的 David Heinemeier Hansson。

Rails 至今仍是一个相对较小的社区。所以当我引用核心团队或者其他几个 Rails 明星的话语时，我就直接使用他们的名字了。特别是 Rick Olson，他是核心团队的一员。他编写了大量非常有用的插件，我们会在书中一再提到。

目标

之前已经提过，我希望本书能够成为 Ruby on Rails 的头号工具书。我并不期望很多人会从头到尾一遍又一遍地来读这本书，除非他们想要扩充对于 Rails 程序框架的基础知识。不管怎样，我希望随着时间的推移，这本书能够让你作为一个应用软件开发者/程序员在日常工作中满怀自信地做出设计和实现的决定。在阅读了本书之后，你对 Rails 基本概念的掌握和所见到的 Rails 解决问题的示例应当让你在面对真实工作中的 Rails 问题时能够得心应手。

如果你在工作中从事程序架构或者开发领导的职位，那么这本书可能并不特别适合你。不过它能够让你在讨论采用 Ruby on Rails 的优缺点或根据你们的项目需要而扩展 Rails 时有的放矢。

最后，如果你是一个开发经理，那么你定会发现我撰写这本书时的务实角度，并会对本书中所介绍的测试和工具感兴趣。同时也希望通过这本书，你能了解为什么你手下的开发者会对 Ruby on Rails 如此憧憬。

前提条件

这本书假设读者具备以下知识。

- 了解 Ruby 基础语法和语言构造，比如代码块。
- 对面向对象编程和设计模式有充分的理解。
- 对关系型数据库有基本了解。
- 熟悉 Rails 程序是如何被开发出来的以及如何工作的。
- 对网络协议有基本了解，比如 HTTP 和 SMTP。
- 对 XML 文档和 Web Service 有大致理解。
- 熟悉事务处理概念，比如 ACID 属性。

我在“本书结构”一节中已经提过，这本书并非以容易的内容开头然后逐渐深入。某些章节的确是以基础知识和介绍性的内容开始的，然后介绍更高级的技巧。本书中的一些内容有经验的 Rails 开发者可能早已掌握，不过，我确信对于各种程度的读者，每章中都有新的知识和启迪。

需要的技术

一部最新型号的、具备 4GB 内存、运行 OSX10.4 的 Apple MacBookPro！当然，我只是开个玩笑。Linux 同样非常适用于 Rails 开发，Microsoft Windows 也可以。不过我不得不礼貌地说明我们特地不在本书中讨论在 Microsoft 平台上的 Rails 开发。据我所知，大多数的专业 Rails 程序都是在非 Microsoft 平台上开发和发布的。

目　录