

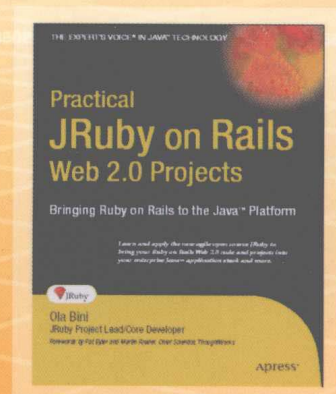
Practical JRuby on Rails Web 2.0 Projects

JRuby实战

[瑞典] Ola Bini 著

丁雪丰 译

- JRuby核心开发人员力作
- 通过4个实战项目全面介绍JRuby
- 完美结合Java和RoR的强大功能



人民邮电出版社
POSTS & TELECOM PRESS

图灵程序设计丛书 Java 系列

Practical JRuby on Rails Web 2.0 Projects

JRuby 实战

[瑞典] Ola Bini 著

丁雪丰 译

人民邮电出版社
北京

图书在版编目 (CIP) 数据

JRuby 实战 / (瑞典) 宾尼 (Bini, O.) 著; 丁雪丰译.
北京: 人民邮电出版社, 2008.8

(图灵程序设计丛书)

书名原文: Practical JRuby on Rails Web 2.0 Projects

ISBN 978-7-115-18375-0

I. J… II. ①宾…②丁… III. ①JAVA 语言—程序设计
②计算机网络—程序设计 IV. TP312 TP393.09

中国版本图书馆 CIP 数据核字 (2008) 第 091945 号

内 容 提 要

本书通过 4 个由浅入深的项目, 结合 Rails 向读者全面介绍了 JRuby。内容包括: 如何在 Ruby 中调用 Java 代码, 如何使用 Java 库, 如何实现并访问 EJB, 如何操作 JMS, 如何在 Java 中调用由 Ruby 实现的 Java 类和接口等。同时, 书中给出的代码都很有实用价值, 只需稍做加工就能放进真正的项目中发挥作用。

本书适合 Web 开发人员阅读和参考。

图灵程序设计丛书

JRuby 实战

-
- ◆ 著 [瑞典] Ola Bini
 - 译 丁雪丰
 - 责任编辑 杨 爽
 - ◆ 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>
北京艺辉印刷有限公司印刷
 - ◆ 开本: 800×1000 1/16
印张: 17.75
字数: 419 千字 2008 年 8 月第 1 版
印数: 1—3 500 册 2008 年 8 月北京第 1 次印刷
著作权合同登记号 图字: 01-2008-2675 号

ISBN 978-7-115-18375-0/TP

定价: 45.00 元

读者服务热线: (010)88593802 印装质量热线: (010)67129223

反盗版热线: (010)67171154

版 权 声 明

Original English language edition, entitled *Practical JRuby on Rails Web 2.0 Projects* by Ola Bini, published by Apress L.P., 2560 Ninth Street, Suite 219, Berkeley, CA 94710 USA.

Copyright © 2007 by Ola Bini. Simplified Chinese-language edition copyright © 2008 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由 Apress L.P.授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

译者序

记得以前在和朋友聊天时讲到自己最近在用Ruby写东西，他们大多会露出好奇的表情，问我什么是Ruby。可现在，这种情况应该不会出现了，Ruby on Rails早已成了人们讨论的热点，它也把Ruby带到了聚光灯下，越来越多的人开始使用Ruby，甚至爱上了这门语言。

如果你是一个Java开发者，想在实际项目中尝试Ruby和Rails时肯定会有这样的问题：我有很多遗留的Java资源，比如EJB和Java库，是把它们用到新项目里，还是放弃它们？有些问题用Java的解决方案显然更好，能不能结合Ruby和Java呢？来试试JRuby吧，你会如获至宝，谁说鱼与熊掌不可兼得？如果你是一个Ruby开发者，那最好也来看看JRuby，你会发现Java的世界一样很精彩。比方说，目前Rails缺少高可用性的解决方案，这是很多大流量网站不敢贸然使用它的主要原因，而采用JRuby方式来部署，可以利用很多成熟的Java解决方案，解决它们的后顾之忧。

本书的作者Ola Bini通过4个由浅入深的项目，结合Rails向读者介绍了JRuby的很多东西，例如，如何在Ruby中调用Java代码，如何使用Java库，如何实现并访问EJB，如何操作JMS，如何在Java中调用由Ruby实现的Java类和接口等，通过学习本书我们甚至可以用JMX来管理Rails应用程序，很神奇吧！本书中给出的代码都很有实用价值，只需稍做加工就能放进真正的项目中发挥作用。要是你急需可以上手使用的东西，那本书就再合适不过了。

本书最后的附录中整理了Ruby和JRuby的常用语法和一些其他的内容，虽然比不上完整的参考手册，但这些内容应付日常事务应该绰绰有余了。

大学时的翻译老师在课上曾多次叮嘱我们，翻译时要忠于原文，表达出作者真实的想法，在此基础上再追求信、达、雅。不过碍于时间，加之本人水平有限，译文中还有很多不足之处，希望各位读者不吝指正。

丁雪丰

2008年3月于上海

序

"Hey, you got your Ruby in my Java!"

"You got your Java on my Ruby!"

我并不是想说JRuby和好时的花生黄油杯巧克力一样美味，但看着Ola和他的同事们为JRuby工作确实是一种享受。他们将一个不完整的Ruby环境（和子社区）变成了可以用多种方式来推动整个Ruby社区进步的东西。

以前，Tim Bray还在抱怨Ruby世界里没有好的开发工具（比如强大的IDE、重构浏览器等）。当时我说Ruby中没有这些工具是因为Ruby社区不用它们也照样能进行开发。后来，JRuby出现了，NetBeans和Eclipse开始着手开发Tim渴望已久的Ruby工具。当这一切发生时，我想了很多，Java开发者也有类似Tim那样的对工具的渴求，JRuby将他们带入了Ruby的世界。

JRuby团队在将他们的实现交付测试时，总是悄悄地给别的开发团队透露一些信息。这些天里，经常可以看到JRuby的开发者挂在Rubinius IRC频道上（freenode上的#rubinius），YARV的开发者也常出现在Rubinius和JRuby的IRC上。不同开发团队间的讨论很引人入胜。你也许会想象出这么一个场景，说不定JRuby还帮助了Microsoft公司的IronRuby呢。

有了运行在JVM上的JRuby后，Ruby就能更好地出现在某些企业级环境中了。在日常工作中，当我还在Ruby上奋斗时，JRuby已经让Ruby解决方案超越了其本身的范畴。很快，JRuby就将成为快速的Ruby脚本架构和大型Java应用程序间的桥梁。这是我一直向往的。

JRuby也在改变着Java/JVM的世界，它的成功让JVM成为运行Groovy、Jython以及与其它同类的语言的理想场所。Sun公司的内部传来了新的、更动态的声音，提出了一些通用的需求，在未来的几年里这会带来巨大的变化。

无论是一个初涉Ruby的Java高手，还是才接触Java不久的Ruby人，本书都能带领你穿越新语言间的灰色地带。希望Ola的这些努力能让你知道不仅JRuby本身很棒，而且Java和Ruby结合后也非常棒。

本书将带你进入一个正在茁壮成长的社区，每个新的JRuby用户都可能带来一些变化。希望你在阅读本书后能接过火炬，寻求（并共享）新的方法，用JRuby来改善你的生活。

祝你的JRuby之旅一路顺风！希望很快能在JRuby的IRC频道和邮件列表中见到你的身影。

Pat Eyler

资深Ruby专家

序 二

过去的几年里，随着Ruby on Rails的崛起，Web应用程序开发的世界受到了巨大的冲击。很多Java世界中的名人（或是不停地叫嚣的人）都成了Ruby on Rails的拥护者，甚至宣称要告别Java世界。

我使用Ruby已经有很多年了，是它的忠实拥护者。比起那些主流的大括号语言，它更关注于简洁的语法，这让我能更清晰地表达我的意图。它完全面向对象，拥有像闭包这样的强大语言特性。值得一提的是它还为元编程和创建域专用语言提供了很多工具。有了这些特性的支持，Rails成为了极富影响力的Web框架。

Rails出现后，我和许多试用过它的同事进行了讨论，他们都是使用过不同Java和.NET Web平台的人，其中的绝大多数人认为Rails极大地提高了他们的工作效率。我不想用“生产率提高了50%”这样严谨的话语，因为软件的生产力本身就不是能轻易测量的。抛开它，我们仍然可以说Rails是一项好技术，因为它获得了广泛的认可。

迄今为止，大多数的图书和文章都关注于在C语言实现中使用Ruby on Rails。而本书则有所不同，它让相同的Rails工作在另一个平台上——Java。JRuby正努力打造一个运行于Java JVM之上的高效Ruby实现，我认为无论对Ruby还是Java，它都会成为一个重要的项目。对Ruby开发者而言，它提供了一个为人所熟知的部署平台，在大企业里更是如此。那扇曾对Ruby关上的大门，在我们讨论到Java部署时又再一次被打开了。

对Java社区而言，JRuby提供了一个体验强大语言和框架的机会，与此同时还能继续使用Java那些杰出的库，在工作中结合Ruby和Java。我看到了JVM的未来，它能够使用多种语言，而且语言之间可以清晰地进行互操作，因此你能为特定的项目选择合适的语言。JRuby是在这一方向上迈出的重要一步，它为JVM带来的不仅是一门语言，更是一个重要的框架。Rails搬进了一个充满咖啡因的新家里，本书可以帮助你更好地理解它。

Martin Fowler

ThoughtWorks首席科学家，《重构》一书作者

目 录

第1章 引言.....1	3.2.2 ActionMailer.....24
1.1 背景.....2	3.2.3 ActionWebService.....24
1.1.1 Ruby简史.....2	3.2.4 ActiveResource.....25
1.1.2 Rails简史.....3	3.3 Rails辅助脚本.....25
1.1.3 JRuby简史.....3	3.3.1 about.....25
1.2 为什么选择JRuby on Rails.....4	3.3.2 breakpointer.....25
1.3 全书概述.....5	3.3.3 console.....25
1.4 小结.....7	3.3.4 destroy.....26
第2章 准备工作.....8	3.3.5 generate.....26
2.1 安装JRuby.....8	3.3.6 plugin.....26
2.1.1 Java.....8	3.3.7 runner.....26
2.1.2 二进制JRuby.....9	3.3.8 server.....26
2.1.3 从源代码安装.....9	3.4 测试.....28
2.1.4 测试安装.....10	3.5 插件.....30
2.2 RubyGems.....11	3.5.1 Act As Taggable.....30
2.2.1 Rake.....12	3.5.2 CAS过滤器.....30
2.2.2 Rails.....13	3.5.3 全球化插件.....31
2.2.3 AR-JDBC.....13	3.5.4 Rails引擎.....31
2.2.4 BlueCloth和RedCloth.....13	3.6 小结.....31
2.2.5 Facets.....14	第4章 商店管理.....32
2.2.6 Mongrel.....14	4.1 创建一个新的Rails应用程序.....32
2.2.7 Mongrel JCluster.....14	4.2 运行Mongrel.....35
2.3 安装数据库.....14	4.3 第一个模型.....35
2.4 小结.....16	4.3.1 ProductType.....35
项目1 商店 (Shoplet)	4.3.2 Product.....37
第3章 Rails入门.....18	4.3.3 ProductCategory.....38
3.1 Rails应用程序的结构.....18	4.3.4 运行迁移.....40
3.1.1 模型.....19	4.3.5 验证.....40
3.1.2 控制器.....20	4.4 产品单元测试.....41
3.1.3 视图.....21	4.5 为产品建立scaffold.....44
3.2 Rails的其他部分.....24	4.5.1 Ajax.....48
3.2.1 ActiveSupport.....24	4.5.2 让界面更漂亮.....50
	4.6 更多模型.....52

4.6.1 用户管理	55	7.4.4 布局	111
4.6.2 订单处理	56	7.4.5 文章	115
4.7 添加身份验证	59	7.5 安全	118
4.8 功能测试	61	7.6 小结	119
4.9 小结	64	第8章 内容呈现	120
第5章 数据库驱动的网站	65	8.1 XML内容呈现	120
5.1 浏览产品	65	8.1.1 Ruby XML	121
5.2 添加购物车	69	8.1.2 Java DOM解析	123
5.2.1 查看购物车	70	8.1.3 Java SAX解析	124
5.2.2 结账	71	8.1.4 Java DOM和XSLT	126
5.3 验证与测试	73	8.1.5 其他Java API	127
5.4 ActiveRecord和JDBC	77	8.2 其他呈现内容的途径	127
5.4.1 支持的数据库	78	8.2.1 RedCloth (Textile)	127
5.4.2 如何支持新的数据库	81	8.2.2 BlueCloth (Markdown)	129
5.5 小结	82	8.2.3 ERb	130
项目2 内容管理系统 (CoMpoSe)		8.2.4 YAML	130
第6章 Java集成	84	8.2.5 其他解决方案	132
6.1 使用Java资源	84	8.3 完成CoMpoSe	132
6.1.1 类	85	8.3.1 呈现引擎	132
6.1.2 基本类型	88	8.3.2 内容	136
6.1.3 数组	88	8.3.3 预览	137
6.2 扩展Java	89	8.4 小结	139
6.2.1 接口	90	项目3 管理系统 (BigBrother)	
6.2.2 类	91	第9章 JRuby与EJB	142
6.3 Java集合类	92	9.1 序列数据库	143
6.4 陷阱	94	9.2 JRuby序列引擎	145
6.5 在Java中使用Ruby	94	9.3 JRuby bean封装器	149
6.5.1 JRuby运行时	95	9.4 小结	153
6.5.2 BSF	96	第10章 基于EJB的Rails应用程序	155
6.5.3 JSR223——Java Scripting	97	10.1 重温EJB客户端	156
6.6 小结	97	10.2 创建应用程序	157
第7章 一个Rails的CMS	98	10.3 创建一个小的序列支持库	161
7.1 数据库	98	10.4 序列控制器及相关视图	162
7.2 模型	102	10.5 服务器端JMX	165
7.3 布局	103	10.6 管理Rails的简单JMX	166
7.4 管理界面	106	10.7 小结	168
7.4.1 用户	106	第11章 部署	170
7.4.2 路径	107	11.1 部署Ruby on Rails	170
7.4.3 样式	109	11.1.1 WEBrick	171

11.1.2	CGI	171	13.2.9	出借图书实例	199
11.1.3	FastCGI	171	13.2.10	归还图书实例	199
11.1.4	Mongrel	172	13.2.11	检索	199
11.1.5	Mongrel集群	172	13.3	ActiveMessaging	200
11.2	部署JRuby on Rails	172	13.4	JRuby和MDB	201
11.2.1	WEBrick	173	13.5	与遗留系统交互的库	202
11.2.2	CGI	173	13.6	Rails间的通信	211
11.2.3	Mongrel	173	13.7	小结	215
11.2.4	GoldSpike (Rails Integration)	173	第14章	LibLib Rails 应用程序	217
11.2.5	Grizzly	173	14.1	数据库	217
11.2.6	Rails-asyncweb	174	14.2	部署多个Rails实例	219
11.2.7	Retty	174	14.3	创建模型	221
11.3	部署JRuby on Rails的最佳实践	174	14.4	视图和控制器	221
11.3.1	JVM Mongrel集群	174	14.4.1	布局	222
11.3.2	用Java制作WAR	177	14.4.2	检索图书信息	225
11.4	小结	182	14.4.3	身份验证	229
项目4	图书馆系统 (LibLib)		14.4.4	借阅者和图书管理员	233
第12章	JRuby与Web服务	184	14.4.5	从Amazon.com导入数据	238
12.1	LibLib系统	184	14.5	小结	239
12.2	Amazon Web Services	185	第15章	尾声: 下一步该做什么	241
12.3	SOAP4R	186	15.1	JRuby-extras	241
12.3.1	动态生成	186	15.1.1	为JRuby-extras做贡献	241
12.3.2	使用桩代码	187	15.1.2	当前的项目	241
12.4	Java中的SOAP	188	15.2	为JRuby做贡献	244
12.4.1	动态生成	189	15.3	潜在项目	244
12.4.2	使用桩代码	190	15.3.1	使用Lucene进行数据库索引	244
12.5	创建小型图书支持库	191	15.3.2	用Hibernate替换ActiveRecord	245
12.6	小结	194	15.3.3	创建新的ActiveRecord-JDBC 适配器	245
第13章	JRuby与面向消息系统	195	15.4	小结	247
13.1	什么是MOM	195	附录A	Java程序员眼中的Ruby	249
13.2	遗留系统	197	A.1	核心Ruby	249
13.2.1	添加图书馆	197	A1.1	命名	249
13.2.2	移除图书馆	198	A.1.2	核心类型	250
13.2.3	获取图书馆名称	198	A.2	类和模块	254
13.2.4	添加图书介绍	198	A.2.1	定义方法	255
13.2.5	移除图书介绍	198	A.2.2	包含和扩展	256
13.2.6	获取图书介绍	198	A.2.3	单例类	257
13.2.7	添加图书实例	199	A.3	块	257
13.2.8	移除图书实例	199			

A.4 元编程.....	259	C.1.6 Rails-core邮件列表.....	270
A.4.1 自省.....	259	C.1.7 松本行弘“Matz”的博客.....	270
A.4.2 send.....	260	C.1.8 O'Reilly Ruby.....	270
A.4.3 method_missing、const_missing.....	260	C.1.9 RubyInside.....	270
A.4.4 define_method.....	261	C.1.10 On Ruby.....	270
A.4.5 Class.new和Module.new.....	261	C.1.11 Loud Thinking.....	270
A.4.6 eval及相关内容.....	261	C.1.12 Riding Rails.....	270
A.5 to_proc的技巧.....	262	C.1.13 Eigenclass.....	270
附录 B JRuby 参考.....	263	C.1.14 Polishing Ruby.....	271
B.1 类和接口.....	263	C.1.15 Programming Ruby, Second Edition.....	271
B.1.1 引用一个Java类或接口.....	263	C.1.16 The Ruby Way, Second Edition.....	271
B.1.2 使用类.....	264	C.1.17 Agile Web Development with Rails, Second Edition.....	271
B.1.3 扩展和实现.....	264	C.2 JRuby.....	271
B.2 基本类型数组.....	265	C.2.1 JRuby主页.....	271
B.3 对Java类的扩展.....	266	C.2.2 JRuby Dev邮件列表.....	271
B.3.1 java.lang.Runnable.....	266	C.2.3 JRuby User邮件列表.....	271
B.3.2 java.util.Map.....	266	C.2.4 #jruby IRC频道.....	272
B.3.3 java.lang.Comparable.....	266	C.2.5 JRuby-extras项目.....	272
B.3.4 java.util.Collection.....	266	C.2.6 JRuby JIRA.....	272
B.3.5 java.util.List.....	267	C.2.7 JRubyInside.....	272
B.4 JRuby模块.....	267	C.2.8 Headius.....	272
B.4.1 runtime.....	267	C.2.9 Tom's Ruminations.....	272
B.4.2 parse.....	267	C.2.10 Ola Bini.....	272
B.4.3 compile.....	267	C.2.11 Nick Sieger.....	272
B.4.4 reference.....	268	C.3 其他.....	273
B.5 require.....	268	C.3.1 MySQL.....	273
附录 C 资源.....	269	C.3.2 ActiveMessaging.....	273
C.1 Ruby和Rails.....	269	C.3.3 Hitta.....	273
C.1.1 Ruby程序设计语言.....	269	C.3.4 Ferret.....	273
C.1.2 Ruby-talk邮件列表.....	269	C.3.5 GlassFish.....	273
C.1.3 Ruby-core邮件列表.....	269		
C.1.4 Ruby on Rails.....	269		
C.1.5 Rails-talk邮件列表.....	270		

JRuby on Rails是一项令人兴奋的技术。如果你正捧着这本书，那么说明你已经意识到了这点。你可能在Ruby和Rails方面都没有什么经验，也可能已经使用过它们，想知道为什么JRuby on Rails如此有魅力。不管怎么样，我希望本书可以教给你一些它所涉及技术的相关内容，介绍些看待问题的新方法，并在一种语言不能完美解决某个问题时，帮助你发现结合了多种语言的解决方案。

我使用Java已经很长时间了，不过从来没有真正喜欢过它。一直以来，我都是一个讨厌编程语言的人，不断地尝试新的语言，就好像我女友试新鞋一样。我知道外面有什么，Java并非针对所有现实问题的最终解决方案。不过，在大多数工作时间里，Java仍是我用来实现系统的主要语言。而在业余时间，我会使用些其他语言。大约3年半前，我发现了Ruby。具体的细节已经不太记得了，我开始用Ruby，并且很喜欢它。它将Lisp元编程能力中的有用部分与Smalltalk的敏感性（sensitivity）和整洁性（cleanliness）相结合，同时还兼具Perl的实用性。

我花了两年多的时间说服老板使用Ruby。这一切除了归功于我说服力提高，真正的原因是Rails的兴起。当我们刚决定要启用Rails时，面临的情况是要在资源和时间比较紧张的状况下，建立一个有简单数据库支持的Web应用程序。最终，我们说服了所有人用Rails来做这个项目，这无疑是一个胜利。从那时起，越来越多的项目使用Rails进行开发，现在大约有大约一半的项目使用Ruby on Rails而不是Java。

尽管如此，我还是感觉有些不对劲。虽然我喜欢Ruby和Rails，但在有些场合下我觉得它们还是应付不了。在Java里，我总是觉得受限于语言本身。而在Ruby中，情况正好相反：语言很棒，但平台和环境却不那么尽如人意。大多数情况下这是因为Ruby和Rails是相对新生的事物，但有部分则是因为Java代码的一些特性——一些让Java代码更健壮、更高效特性。

从那时起，我开始寻找能将Ruby中我最喜欢的特性和Java平台中的某些优点结合起来的途径。其中，我在Java上的不同Lisp实现上花了点时间，最终成了Jatha项目（一个Common Lisp实现的提交者）。但所有的Lisp实现都有一个共同的问题：它们不像Ruby那样拥有一个杀手级的应用程序。我很喜欢Jatha，但它背后没有庞大的社区，没有足够的库支持。

因此，我继续寻找，直到在2005年秋天发现了JRuby。那时我真的很想使用JRuby，并做点贡献，可惜我没有时间。3个月后，Charles O. Nutter和Thomas Enebo做出了不少成果，很明显，JRuby在未来的某天一定可以运行Rails。鉴于这一点，我开始帮着做点小的东西，之后又做了些JRuby

运行主流应用程序所必需的重要扩展。YAML (YAML Ain't Markup Language) 是其中关键的一个, 它让我们能够专心地关注于RubyGems和Rails的支持。就在那个时候我意识到Rails和JRuby在一起能有多么强大。现在, 也就是1年后, 我们已经可以成功运行几乎所有的纯Ruby应用程序了。Rails应用程序通常能完美地运行, 同时这些应用程序也可以使用Java的各种功能。它们可以在保持Ruby语言所有特性的同时充分利用Java的优势。

看起来我最终找到了能够使用的解决方案了。本书描述了结合Ruby和Java你能实现什么, 建立了几个实用的Rails应用程序, 并用了几个工具来部署它们。这些工具是你在用常规的Ruby实现时没办法使用的。

本章会着重介绍Ruby、Rails和JRuby的背景, 它们是从哪里来的, 简单地讲述它们是什么。我会更深入地说明为什么JRuby on Rails是如此理想的组合。最后概述本书的其余部分, 这样你就可以知道后面等待着你的的是什么。

Ruby、JRuby和Rails都是令人兴奋的技术。我很喜欢使用这些技术, 在过去的18个月里参与JRuby是我做过的最明智的选择。我希望能通过本书与你分享我的热情(以及我这么做的理由), 在读完本书后你也能感受到它。对我而言, 使用Ruby与使用Java是有天壤之别的。如果你是一位Java程序员, 那么也许你正在怀疑这种说法。然而, JRuby的好处就是它可以充当保护伞的角色。你可以享受Ruby带来的乐趣, 在需要的时候也能使用Java。

下面就让我们从简单介绍本书所涉及的技术开始吧。

1.1 背景

本书关注于以下4种技术: Ruby、Rails、JRuby和Java。之所以在此加上Java, 是因为它是JRuby与Ruby的不同之处。实际上, 书中不会出现太多的Java代码。你能感觉到Java的存在, 但它不会很明显。在这里, Java更重要的是作为一个平台, 让其他技术可以在其上运行。

这里假设你已经拥有了足够的Java知识。我会快速地从发展历史的角度来介绍Ruby、Rails和JRuby, 其中不会包括语言和API信息, 你可以分别通过附录A和第3章来了解Ruby和Rails。

Ruby和JRuby诞生的时间远比大家想象的久远。Ruby和Java差不多, JRuby也有6年多了。而它们的重要性在最近才渐渐为人所知, 一个重要的原因就是Ruby有了Rails这样的杀手级应用程序, 在那之前JRuby还没有被当做通用解释器来使用。

1.1.1 Ruby 简史

1993年松本行弘“Matz”创造了Ruby, 1995年首次公开发布。其主要的实现是一个用C语言写的解释器, 在需要区分Ruby语言和Ruby实现时, 这个解释器一般被称为MRI (Matz Ruby Implementation)。

Matz反复声明设计Ruby是为了提高程序员的效率, 给他们带来乐趣, 这点从Ruby代码中就可可见一斑。Matz还强调Ruby竭力遵循最少惊奇的原则, 意思是Ruby语言对有经验的使用者来说应该是没有多少容易混淆的地方的。它的一个额外好处就是使得基本的Ruby很容易被掌握。这个语言是为人而不是机器设计的, 意识到这点尤为重要。语言中包含很多效率低下的部分, 由于这

些功能对程序员很有价值，所以不能因为它们很难实现就将其排除在外。

如果你对计算机语言的术语比较感兴趣，那么你也可能会对以下这些Ruby的特性——反射、动态类型和强类型、面向对象以及垃圾回收感兴趣，Ruby还支持很多有趣的语言特性，例如Continuation、绿色线程^①（green thread）、协同例程（coroutine）、迭代器、生成器、闭包和元编程。它结合了Perl、Smalltalk、Python、Lisp、Dylan和CLU的主要特性。

1.1.2 Rails 简史

Ruby on Rails是一个Web应用程序框架，于2004年首次发布。它最早是从37signals公司的Basecamp应用程序中提取出来的。该框架的主要创始人是David Heinemeier Hansson（大家一般称他为DHH）。它自发布后就广受关注，吸引了大批追随者。现在看来，Rails可以被视作Ruby的一个杀手级应用程序，在它的帮助下Ruby开始流行起来了。

作为框架，Rails实在是不含有什么新的东西，让它如此出众的原因是它在一些核心哲学思想的指导下，以一种高效的途径结合了几种使用模式和库实现。其中一条就是“不要重复自己”（Don't Repeat Yourself, DRY）——信息应该被放在一个单独的、明显的地方。更重要的是“约定优于配置”（Convention over Configuration），这意味着你的应用程序只要遵循Rails的惯例，就可以极大程度地减少配置和编码量。它广泛地使用了Ruby的元编程特性，使得用Rails来做Web开发成为一件令人愉快的事情。

Rails提供了由代码生成器创建的scaffold和skeleton代码，它们可以加速应用程序的开发。通常情况下，你只需安装好Rails，几分钟后就可以运行一个简单的CRUD应用程序（带新增、读取、更新、删除功能的应用程序）了。它让你有机会使用一种不同的、更敏捷的结构来开发系统，因为能快速获得反馈，通常客户可以在开始时就参与进来。

在第3章里你会了解到更多Rails的信息，该章内容会告诉你在用Rails开发前需要知道的所有东西。

1.1.3 JRuby 简史

JRuby最早是Ruby 1.6 C代码的移植，诞生于2001年，创始人是Jan Arne Petersen，在很长的一段时间里它只支持1.6版的语义。在1.8版发布后，维护者在两年的时间里逐步地将1.8版的功能引入其中。最终的转折点是在2006年初，项目领导人Thomas Enebo和Charles O. Nutter将目标定位在完全兼容Ruby上，最终的考验是要在不做修改的情况下运行Rails。为了实现这一目标，大量的时间被用在创建更好的测试套件和重写大部分系统上。

很明显，要实现这一目标，直接移植C代码不是一个好方案。Java和C的执行模型有着很大的不同，这意味着，当Java里有更好的解决方案时复制C的结构就很困难，而且在使用类似MRI的执行模型时这么做性能很差。

在将几个重要的扩展移植到Java上后（YAML、Zlib和其他一些Ruby系统的重要部分），

^① 也称作用户线程（user thread）。——译者注

RubyGems和Rails可以开始运行了。虽然还存在问题，但对它们的支持在迅速改进。当Tom和Charles在2006年9月被Sun Microsystems公司雇用，全职进行JRuby的开发后，一切就变得更好了。

由于JRuby有数量众多的测试，这使得对代码的大量重构和对内部（internal）的改变成为可能，只要测试可以运行，解释器就是可用的。JRuby团队还从其他项目里引入测试套件，其中比较值得注意的是常规Ruby实现测试[这些测试大多来自于Rubinius（另一个Ruby实现）]和几个应用程序套件。JRuby有一个持续集成服务器，上面运行着完整的Rails、RubyGems和Rake测试。

在这段时间里，核心开发者们在寻找一种将Ruby代码编译为Java字节码的方法。2007年6月1.0版发布时，默认的运行时系统工作在混合模式下，对方法进行JIT（Just In Time，实时）编译。虽然那时编译器还不完整，只能处理一半的Ruby语法，但它却带来了不小的性能提升。1.0版发布时，JRuby的性能已经接近MRI了，有些地方靠得近些，有些地方则稍微慢些。

JRuby和MRI有几个显著的区别。首先是线程模型不同，JRuby使用真实的操作系统线程，而MRI则使用绿色线程（由Ruby解释器实现，运行在同一进程内）。

JRuby不支持Continuation，要在一个运行在像Java这样的虚拟机上的系统里实现Continuation很困难。另一个原因是没有办法将Java的集成特性和Continuation整合起来。如果你对这个问题很感兴趣，在JRuby-dev和Ruby-core的邮件列表上有很多相关讨论。

JRuby在文件系统的操作方面也不太理想，不过大多数情况下MRI在Windows系统的文件操作方面也不怎么样。

Ruby计划在下一个大版本中支持名为YARV（Yet Another Ruby VM）的新执行模型。对Rubinius执行的支持也在讨论中。

1.2 为什么选择 JRuby on Rails

现在，你知道为什么说Ruby和Rails是有趣且令人兴奋的技术了吧，剩下的就是解释一下JRuby on Rails有何与众不同之处，以至于需要为它写一本书。我在Ruby的介绍里并没有提到MRI存在一些问题。大多数问题是由语言的灵活性造成的，还有就是Matz总是关注于语言本身，而不是它的实现。

第1个问题就是性能。很多情况下，Ruby的速度已经够快了，而且工作得很好。但另一方面，在性能测试中，Ruby一般会排在所有语言的末尾。一种常见的思路是当在Ruby中遇到了性能问题时，就用C语言来实现关键部分。因为喜欢Ruby语言，所以我想使用它。我不想使用C语言，尤其不想在应用程序的关键部分使用C语言。事实上，那些关键部分应该是我从Ruby中获益最多的地方。但是这一点现在还不能完全得到保证。

JRuby通过十分关注性能来改善上述问题。正如大家所见，1.0版里并没有做太多性能方面的工作。要衡量总体性能比较困难，不过大多数情况下JRuby 1.0会比Ruby 1.8.6慢大约1.5~2倍，原因在于1.0版的重点是兼容性。但是，在解释器和编译器的工作中，核心团队已将重点放在提高性能上了。所以，有理由相信JRuby会比现在快，甚至比C语句的实现还要快。

第2个问题是当前的Ruby实现在Unicode和UTF-8的支持方面不够稳定。在2007年要创建一个连接因特网的应用程序，你需要在语言层面上拥有快速、可靠、无所不在的Unicode支持。没有

它,你注定会迷失方向。MRI通过一个叫做KCode的东西在一定程度上支持Unicode。但是很多字符串方法没有用KCode,而且它还存在不少问题。应用程序开发者只能通过建立一些库来弥补这些不足,Rails里有一个称作Multibyte的东西。

因为JRuby运行在Java平台上,从技术上讲,你能获得Java对Unicode的所有支持。目前你需要使用真正的Java String来处理Unicode,不过为JRuby添加更好的语言级支持才是当务之急。在那里处理Unicode会比较容易,因为它天生就支持Unicode。第一步要做的就是实现一个本地Multibyte后端和其他类似的库。当你读到这里时,这些应该已经实现了。

由于Ruby使用绿色线程,它永远不可能利用超过一个的处理器内核,这一事实在某些应用程序里是无法接受的,而在另一些里则会造成不便。JRuby使Ruby线程基于真实的操作系统线程之上,从而解决了这一问题。这造成了它与MRI的一些不兼容,不过大多数人认为这很值。

以上这些是使用JRuby代替Ruby的主要原因,同样也是在该平台上运行Rails的主要原因。然而,考虑到Rails,Java平台为它提供了一些更有趣的机会和能力。首先,Rails的开发通常被认为是件令人愉快的事情,不过Rails的部署就不是了(虽然有不少工具可以帮助部署Rails);其次,Rails的开发没有Java那样的成熟度。所以在JRuby中部署Rails应用程序就成为了一个卖点。(在第11章里你会看到怎么来做这件事。)

很多情况下,一个应用程序需要用到很多库来实现它的功能。Ruby诞生已经很久了,不过它的库的历史就不能和Java平台的比了。有时你不得不编写自己的库,因为Ruby还没有这方面的库。在Java里,这一现象已经不会发生了,通常有很多库可供你使用,有的还提供商业支持。所以在开发新的应用程序时使用JRuby on Rails很有用,能够在应用程序内部使用Java库来实现某些功能也很有帮助。JRuby可以使这一工作变得非常简单。

还有很多原因促使你使用JRuby on Rails,本书将涉及其中的绝大部分。

1.3 全书概述

本书分为4个部分,有3个附录。为了帮助你了解本书是如何安排的,我将在这里快速地介绍一下各章。如果你需要了解关于某个主题的特定信息,可以跳过此部分。请记住,每个部分里的大多数章都使用同一个项目,也就是说可以在之前的章中找到一些重要的上下文。

书中的4个项目相对独立,但它们都依赖于你在之前的章中学到的知识。

第1章 引言

这是本书的介绍部分,给出了书中涉及技术的信息,以及为什么你该对它们感兴趣,另外还给出了本书的概述。你正在读的就是这章。

第2章 准备工作

本章旨在帮助你加快步伐,告诉你如何安装书中用到的每一个东西,包括所有需要的RubyGems,还对它们做了简单的介绍,教你如何使用gem命令来执行基础任务。

项目1 商店(Shoplet)

商店应用程序是你建立的第一个Rails项目,如果你之前用MRI开发过Rails项目,它们之间不会有太大区别。最大的不同就是这个系统后台使用JDBC(Java Database Connectivity)。

第3章 Rails入门

本章以不涉及代码的方式介绍了Rails, 描述了它包含的部分, 告诉你在进行Rails开发时需要知道的东西。

第4章 商店管理

这里建立了Shoplet应用程序的前半部分。在这一过程中介绍了很多Rails的实用小细节。

第5章 数据库驱动力的商店

本章完成了Shoplet应用程序, 随后关注了JRuby on Rails支持的数据库。

项目2 内容管理系统 (CoMpoSe)

第二个应用程序并不比第一个大多少, 不同的是它大量使用了某些Java的库来处理XML (Extensible Markup Language, 可扩展标记语言) 和内容呈现。

第6章 Java集成

本章中我们会专门看一下JRuby中Java集成特性的语法和用法。

第7章 一个Rails的CMS

使用第一个项目中学到的内容, 我们着手建立CMS应用程序所需要的大部分Rails代码, 而呈现部分先用桩代码来代替。

第8章 内容呈现

本章中我们将用第6章讲到的Java集成特性来实现所有的内容呈现功能, 完成CMS应用程序, 同时再看一些可替代的方法。

项目3 管理系统 (BigBrother)

BigBrother系统建立在独立的Rails前端和企业级后端基础上。它还有一些特性使该系统可以用JMX (Java Management Extensions) 来管理。

第9章 JRuby与EJB

让我们来看看如何在一个J2EE Enterprise Bean中使用JRuby, 用Ruby来实现这个bean的功能。

第10章 基于EJB的Rails应用程序

通过实现一个Rails前端我们完成了绝大部分的BigBrother应用程序, 这个前端可以与一个Enterprise JavaBean相交互, 并用JMX来自行管理。

第11章 部署

本章详细介绍了JRuby on Rails应用程序的部署选项, 常规的Rails部署是如何工作的, 以及JRuby如何让情况变得更好。

项目4 图书馆系统 (LibLib)

最后一个项目是一个分布式的图书馆系统, 它共享了遗留系统中的中央数据库。该应用程序使用消息服务来与别的应用程序实例进行交互, 也包括遗留系统。

第12章 JRuby与Web服务

本章将介绍JRuby中调用Web服务时可用的选项, 并实现一个在Amazon.com查找图书的库。

第13章 JRuby与面向消息系统

我们将深入研究面向消息中间件, 实现一个使用JRuby和JMS的系统。本章会为JMS交互建