



经典的设计模式，灵活的语言实现

Ruby 设计模式

Design Patterns in Ruby



(美) Russ Olsen 著
谈熠 陈熙 译



机械工业出版社
China Machine Press

Ruby

设计模式

Design Patterns in Ruby



(美) Russ Olsen 著
谈熠 陈熙 译



机械工业出版社
China Machine Press

本书是一本关于设计模式方面的重点书籍。本书以通俗易懂的方式介绍了 Ruby 设计模式，主要包括 Ruby 概述、使用模板方法变换算法、使用策略替换算法、通过观察器保持协调、通过迭代器遍历集合、使用命令模式完成任务、使用适配器填补空隙、使用装饰器改善对象、单例、使用工厂模式挑选正确的类、通过生成器简化对象创建和使用解释器组建系统等内容。

本书适合程序员阅读，也可以作为 Ruby 语言的参考书。

Simplified Chinese edition copyright © 2008 by Pearson Education Asia Limited and China Machine Press.

Original English language title: Design Patterns in Ruby (ISBN 978-0-321-49045-2) by Russ Olsen, Copyright © 2008.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc.

本书封面贴有 Pearson Education (培生教育出版集团) 激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

本书法律顾问 北京市展达律师事务所

本书版权登记号：图字：01-2008-1861

图书在版编目 (CIP) 数据

Ruby 设计模式 / (美) 奥尔森 (Olsen, R.) 著; 谈熠, 陈熙译. —北京: 机械工业出版社, 2009. 1

(Ruby 和 Rails 技术系列)

书名原文: Design Patterns in Ruby

ISBN 978-7-111-25120-0

I. R… II. ①奥… ②谈… ③陈… III. 计算机网络—程序设计 IV. TP393.09

中国版本图书馆 CIP 数据核字 (2008) 第 144655 号

机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑: 王 玉

北京京北印刷有限公司印刷 · 新华书店北京发行所发行

2009 年 1 月第 1 版第 1 次印刷

186mm × 240mm · 17.25 印张

标准书号: ISBN 978-7-111-25120-0

定价: 45.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线: (010) 68326294

本书的赞誉

“本书记录了用于解决 Ruby 开发者常见问题的聪明方法。Russ Olsen 在这方面做得棒极了。他不仅收录了经典的设计模式，还扩充了只与 Ruby 相关的新模式。他清楚地介绍每个模式，使 Ruby 开发者能在日常工作中获得宝贵的经验。”

——Steve Metsker, Dominion Digital, Inc. 管理咨询顾问

“此书为‘四人组 (GoF)’所提出的一些重要设计模式提供了极佳的演示，而没有过多地诉诸技术性解释。作者以精准而易懂的风格进行了完整的描述。即使之前没有接触过设计模式的开发者，通过阅读此书，也能很快地、自信地在使用 Ruby 时运用设计模式。Olsen 干得很出色，他将一本关于经典的‘枯燥’主题的书写得有声有色并不乏幽默。”

——Peter Cooper

“在我关注设计模式 10 年之后，本书使我对这个主题重新燃起了兴趣。Russ 在四人组的基础上，挑选了 Ruby 中最有用的设计模式，并以一种直接和逻辑的方式介绍它们。本书提升了我使用 Ruby 的能力，并且鼓励我拂去覆盖在四人组的那本书上的灰尘。”

——Mike Stok

“对于来自静态类型的面向对象语言的程序员来说，本书给出了一个很好的途径，以学习如何在更加动态、灵活的语言（比如 Ruby）中运用设计模式。”

——Rob Sanheim, Ruby Ninja, Relevance

告

2008 年 8 月

译者序

随着计算机程序设计语言的不断发展和进步，动态语言已经得到开发领域的认可，并在软件行业的各个领域中获得长足的发展。今天，Ruby 已经成为世界上发展最快的程序设计语言之一。一个充满热情和创造力的社群围绕 Ruby，开展着各种激动人心的工作。这个社群无需豪言壮语，所有的工作都在扎扎实实地推进，人们被自己内心的力量驱动着，而这种力量来自于 Ruby 语言的强大和优雅。使用这门语言也是莫大的乐趣。

自 1994 年，四人组（GoF）的名著《Design Patterns: Elements of Reusable Object-Oriented Software》^①问世以来，设计模式便成为软件工业的圭臬。四人组所提出的设计模式总结归纳了传统软件开发的经验，为开发中所遇到的各种常见问题提供了解决之道。Ruby 语言的动态特征恰恰为这些设计模式的实现提供了一个轻盈而高效的平台。此外，Ruby 本身具有传统的静态类型语言所不具备的动态优越性，这些高级特性为 Ruby 语言创造了域指定语言、元编程和惯例优于配置等设计模式。传统的软件开发思路是：好的设计会产生好的软件。因此在实际开发之前，值得花时间做一个全面而细致的设计。而在阅读本书之后，你会发现 Ruby 使你的设计灵活地贯穿于开发之中。

本书由谈熠和陈熙合力翻译，在翻译中得到了陈理人、陈慧勤、陈少科和金爱珍等朋友的指导和帮助。借此机会，特别感谢华章公司引进此书，并将如此重要的一部书交托给我们来翻译，希望译文能不辱使命。我们还要感谢华章的编辑不辞辛劳的工作。

Ruby 和基于 Ruby 的著名程序框架 Rails 在国内的应用开始步入快速发展阶段。我们虽然有数年的使用 Ruby 的工作经验，然而 Ruby 和 Rails 社区中如泉水般喷涌的新技术始终鞭策着我们不断地学习和埋头奋进。限于译者水平，译稿中若存有这样那样的错误，恳请你吝指正。在 Ruby on Rails 中文网站的用户社区 <http://rubyonrails.cn/community/> 中，有不少国内使用 Ruby 和 Rails 的顶尖高手。如果你遇到了技术方面的问题，你可以在那里最快地找到解决之道。此外，也可以通过电子邮件联系我们：yi@rubyonrails.cn。

译者

2008 年 8 月

① 此书由机械工业出版社出版。——编辑注

序

《Design Patterns: Elements of Reusable Object-Oriented Software》的作者被读者亲切地称为“四人组 (GoF)”。该书是第一本作为设计模式参考的主流书籍。从 1995 年至今, 该书已经售出了 50 多万册。毫无疑问, 它影响了全球数以百万计的程序员的路和代码。我仍然清楚地记得在 20 世纪 90 年代末我第一次买那本书时的情景。部分热情源自向我推荐该书的同辈人。我将其视为我迈向一个成熟程序员的一步。我在几天内便翻遍了那本书, 并急切地构想着每一个模式的实际应用。

大家普遍认为设计模式的最大用处在于, 当程序员们进行关于开发的讨论时, 为他们提供一个能够清楚描绘设计决定的词汇库。特别是在结对编程 (极限编程和其他敏捷开发的基础) 的情况下, 设计是一个进行态的共享步骤。你能够轻松地跟与你结对的同事说: “我觉得需要在这里使用些策略”、或者“让我们给这个功能加一个观察器。”

在一些公司里, 考察对于设计模式知识的把握程度已经成为一种快速检阅程序员应聘者的途径。在那里通常会听到:

“你最喜欢的设计模式是什么?”

“嗯……工厂模式吧?”

“谢谢你前来面试, 门在那里。”

上面说到的“最喜欢”的模式是不是听起来有点奇怪? 我们钟爱的设计模式应当是对于给定的情况能对症下药的那个。缺乏经验的程序员在学习设计模式时容易犯的一个典型错误是, 把一种模式当作目标去实现, 而不是当作一种手段。为什么会有人为了好玩而去实现设计模式时又找不着头绪?

至少在静态类型的世界里, 实现设计模式需要解决一些技术难点。最好的情况是, 你使用一些忍者技巧显示你超群的编程能力。而在最糟糕的情况下, 你最后得到的是一堆从模板文件里七拼八凑而成的废物。所以, 设计模式至少对于像我这样的奇客 (Geek) 来说是一个有趣的题目。

那么, 用 Ruby 实现四人组所提出的设计模式困难吗? 不一定。对于入门者来说, 没有了静态类型的 Ruby 大大减少了实现程序的代码总量。Ruby 标准库也以单行包含的方式提供一些最常见的模式。而另外的一些也构建在 Ruby 语言自身中。比如, 四人组所说的 Command 对象本质上就是对一段用来做特殊事情的代码的包装, 即在某个时刻运行一段特定的代码。当然, 这是对 Ruby 中的 block 对象 (或 Proc) 的一个相当准确的描述。

Russ 自 2002 年起使用 Ruby 进行工作。他知道最有经验的 Ruby 用户早就通晓设计模式, 并且知道如何运用。所以, 他在本书中遇到的最大挑战, 就我所知, 是要以一个既能和专业 Ruby 程序员相关相通, 又能帮助初学者走近我们所热爱的这门语言的角度来撰写此书。我认为他成功了, 你也会这样认为。再用 Command 对象举例子: 它最基本的实现是简单地采用一个 block。当加入了状

态和一些行为后，实现就变得一点都不简单了。Russ 在此为我们提供了专用于 Ruby 的、被证明过的建议，而且“药到病除”。

本书的另一个优点就是，加入了 Russ 定义的专用于 Ruby 的新设计模式。Russ 对它们进行了详细的说明，其中包括我最喜欢的内部域指定语言。就像一个 Interpreter（解释器）模型的进化，我相信他对这个主题的论述，使本书出版界在本领域中的第一部有益的著作。

最后，我觉得本书会给刚开始在专业工作中使用 Ruby 的人和来自其他并不十分强调面向对象设计和模式的语言（比如 PHP）背景的人带来巨大的福音。在描述这些设计模式的过程中，Russ 捕捉了我们用来解决每天使用 Ruby 编程时都会遇到的障碍的精华——对于新手来说，这可谓是无价之宝。本书具有如此多的优点，我早就将它列入送给我的新同事和朋友的礼物清单中。

——Obie Fernandez 专业 Ruby 丛书编辑

前言

我曾经共事过的一个同事一直对我说，那些厚重的，关于设计模式的书足以证明编程语言的匮乏。他的意思是，因为设计模式是代码中的约定惯例，所以一个好的编程语言应当全面地整合这些设计模式，使它们消失在代码中。

举一个极端的例子，在 20 世纪 80 年代，我曾参与的一个项目是用 C 语言编写面向对象代码。是的，是 C 语言，而不是 C++。我们所采用的技巧是将每个“对象”映射到一个函数指针列表。然后通过查找表中的指针来调用相应函数从而操作我们的“对象”，从而模拟对于对象中方法的调用。这种方法比较笨拙而且烦琐，但是能够解决问题。如果对这个方法进行归纳总结，可能会称其为“面向对象”设计模式。当然，后来有了先进的 C++，以及后来的 Java，我们的面向对象模式便消失了。今天，我们不太会将面向对象看作一种设计模式——因为今天的它实在太简单。

但很多情况仍未变得简单。名副其实的“四人组”之书（《Design Patterns: Elements of Reusable Object-Oriented Software》，Gamma, Helm, Johnson 和 Vlissides 著）依旧是每个程序员的必修读物。然而对于该书所说的许多模型，在今天各类流行语言（Java 和 C++，和（或）C#）中的具体实现看上去多少有点像我那个 20 世纪 80 年代的过时手工对象系统。它太痛苦、太冗长、太容易出错。

Ruby 编程语言带着我们向我那位同事的理想迈进了一步。用 Ruby 实现设计模式是如此简单，以致有时候设计模式淡出到程序背后。由于以下原因，用 Ruby 构建模式变得更简单：

- Ruby 是一种动态类型语言。免去了静态类型，Ruby 极大地减少了大多数程序的代码量，包括用来实现设计模式的代码。
 - Ruby 内建代码闭包。闭包允许我们传递一段代码并指定其执行访问而不必辛苦地构建空而无效的整个类和对象。
 - Ruby 的类是真实的对象。因为类在 Ruby 中即另一个对象，我们可以像对待任何其他对象一样对 Ruby 类进行常规的运行时操作：我们可以创建全新的类。我们可以通过增加或删除方法来修改已存在的类。我们甚至可以克隆一个类，修改它的副本，而保持原类不变。
 - Ruby 具备一个优雅的代码重用系统。除了支持普通的类继承以外，Ruby 还允许我们定义混合（mixin）。那是一种简单但灵活的方法，使代码可以被若干个类共享。
- 以上这些因素使得 Ruby 代码变得可压缩。在 Ruby 中，像在 Java 和 C++ 中一样可以实现复杂的想法。不过只有通过 Ruby 才可能把实现的具体细节更有效地隐藏起来。正如你将在以后见到的那样，许多设计模式在传统静态语言中需要大量的模板代码，而在 Ruby 中只需要一两行即可。你可以简单地使用 `include Singleton` 命令将类变成一个单例（singleton）。你可以像实现继承那样简单地实现委托（delegate）。因为 Ruby 使每行代码能做更多有趣的事情，最后，只需要更少的代码来

完成工作。

这并不是简单地在键盘上偷懒，而是对 DRY（不重复你自己）原则的应用。我不认为现在还会有人对旧 C 语言中的面向对象模式的消亡而感到惋惜。它的确曾为我解决了问题，但我也不得不花时间为它工作。Ruby 提供了一个通向更高境界的阶梯，只需要工作一次，便可以将模式从代码中压缩出来。简而言之，Ruby 使你把注意力放在解决实际问题上而不是其他工作中。我希望本书能够帮助你了解如何达到那个境界。

本书的目标读者

简单地说，本书写给那些想知道如何用 Ruby 有效地开发软件的人。假设你熟悉面向对象编程，但对设计模式没有任何了解，你可以在阅读本书的过程中学会。

你也不需要掌握很多 Ruby 知识来阅读本书。你会在第 2 章中找到对于这门语言的快速入门。同时，我会对每个 Ruby 语言特定的环节进行详细解释。

本书内容的组织结构

本书分为三个部分。第一部分从整个设计模式发展的历史和背景的最简单纲要开始，涵盖重点的 Ruby 语言的快速入门。

第二部分是本书的主体，通过 Ruby 的角度来讲解四人组提供的设计模式。这个模式要解决怎样的问题？这个模式的传统实现方法（四人组给出的方法）在 Ruby 中是怎样的？传统实现方法在 Ruby 中是否有效？Ruby 是否提供了其他选择让实现变得更简单？

第三部分将讲述随着 Ruby 语言的推广和发展而出现的 3 种新设计模式。

有言在先

在重复那句我多年来一直默念的箴言之前，我无法在一本关于设计模式的书上签下我的名字：设计模式是一小盒封装好的用来解决常见编程问题的方案。在理想的情况下，当遇到一个相应的问题时，你应当激活对应的设计模式，然后就会解决问题。但第一步（等待相应问题的到来）经常困扰一些工程师。只有当你面对应当使用设计模式来解决的问题之后，才能说你正确地应用了设计模式。

在一些圈子中，对于设计模式的滥用已经玷污了设计模式的名声。我可能会争辩，用 Ruby 能比用其他语言更简单地写一个使用工厂方法的转换器，然后用它获得一个生成器的代理器。那个生成器随后创建一个命令去进行一个 2 加 2 的操作。Ruby 的确能让这些步骤的实现变得简单，即使在 Ruby 中，这样的设计没有任何意义。

你也不能将程序的构建看作一个对于现有设计模式简单拼接的组合。任何有趣的程序都有特殊之处，一些只用来完美地解决特定问题的代码。设计模式是要帮助你认识和解决在软件创建过程中重复产生的一般问题。使用设计模式的优势在于，让你能够迅速克服一些别人已经面对过的问题，

从而使你能调整难点——那些只有在你的程序中面对的问题。设计模式不是万能的灵丹妙药，无法帮你解决所有的设计问题。它们仅是一个技巧，一个帮助你构造程序的非常有用的技巧。

关于本书中的代码风格

用 Ruby 编程变得如此有趣的原因之一是，这门语言尽量不影响你进行程序设计——如果有多种方法可以用来描述事物的话，Ruby 通常支持所有这些方法：

```
# 方法之一

if (divisor == 0)
  puts 'Division by zero'
end

# 另一种

puts 'Division by zero' if (divisor == 0)

# 第三种

(divisor == 0) && puts 'Division by zero'
```

出于语法上的考虑，Ruby 也尽量不强调语法。只要表意清晰，Ruby 尽可能地让你省略不必要的东西。比如，在调用一个方法的时候，你可以省略参数列表两旁的括号：

```
puts('A fine way to call puts')
puts 'Another fine way to call puts'
```

当你定义一个函数的参数列表时，以及编写 if 语句的条件时，你甚至都可以省略括号。

```
def method_with_3_args a, b, c
  puts "Method 1 called with #{a} #{b} #{c}"
  if a == 0
    puts 'a is zero'
  end
end
```

所有这些快捷方式能够在编写真正的 Ruby 程序时带来方便。然而，当它们被大量使用的时候，通常会给初学者带来困惑。所以许多刚用 Ruby 的程序员可能对以下格式感到难以适应：

```
if file.eof?
  puts( 'Reached end of file' )
end
```

```
puts 'Reached end of file' if file.eof?
```

也可能

```
file.eof? || puts('Reached end of file')
```

由于本书主要讨论 Ruby 的深层力量和优雅，而不是纠缠语言的语法细节上，所以，我尽量保持平衡，一方面尽量让示例代码看上去像真正的 Ruby 代码，另一方面，确保它们对初学者来说是友善的。在具体的实践中，我采用了一些显而易见的快捷方式，同时避免使用更极端的窍门。这并非意味我不了解，或不赞同 Ruby 的语法快捷功能。我只是更想对使用 Ruby 的新手们尽量保持概念的优雅。当你忘我地投入这门语言之后，你会有大量的时间学习语法快捷功能。

致 谢

我一直以来都觉得引用名言“无人是孤岛”多少会让人感觉呆板。大多数岛屿都是水下山脉的顶峰，你所见到的那一小块绿色的部分是由水下看不见的连绵山体所支撑的。同样，在无人帮助的情况下，没人能完成任何一件事情。所以，每件事所得到的赞誉应当由我们的朋友、家人和同事们共享。我们都是跳出水面的岛屿，背后由帮助过我们，且不知名的人所支撑。离开了慷慨相助的人们，这本书绝不会开始，也不可能完成。

我要特别感谢我的好朋友 Bob Kiel。他至少告诉了我 17 827 次我应当写本书。同时也感谢 Xandy Johnson，谢谢他在我写作过程中的鼓励 and 大力帮助。

我还要向所有校阅过本书的人说声“谢谢”，他们包括 Mike Abner、Geoff Adams、Peter Cooper、Tom Corbin、Bill Higgins、Jason Long、Steve Metsker、Glenn Pruitt、Rob Sanheim、Mike Stok 和 Gary Winklosky。特别感谢 Andy Lynn 和 Arild Shirazi 对本书的早期书稿进行的大量审阅工作，也特别感谢 Rob Cross 查出“最后一个”错别字。

同样感谢 Heli Roosild，他是一个专业作者。他花时间阅读了我写的一些东西并说：“可以，就这样。”

还要感谢 Lara Wysong、Raina Chrobak 和 Christopher Guzikowski，以及 Addison-Wesley 的全体工作人员，特别是 Chris，他审读了我写的 900 字的博文文章和整本书。还要感谢 Jill Hobbs 用精准的眼光和更为精准的笔尖为本书进行了校对。

我想感谢 FGM 的朋友们，他们营造了一个求知的环境让写作本书成为可能。

还要感谢 Steve McMaster 和他在 SamSix 的同伴，感谢他们的支持和鼓励。

然后从个人角度，感谢我的妻子 Karen，谢谢她的鼓励和建议，并感谢她在数月中让我租用厨房餐桌的一角。同时也感谢我的儿子 Jackson，让我在书中不时地能讲讲他的故事。感谢 Diana Greenberg，一个和我们患难与共的朋友。

感谢我的兄弟 Charles，他树立了一个勇敢和坚定的榜样，使我每天都努力向他看齐。

最后，感谢我的姐姐 Dolores，她培养了我对阅读的兴趣。我清楚地记得那天她把从借来的那些快速读物中拉开，拖着我走进图书馆，带我到严肃读物的书架前。她拿下一本说：“这才是你应该读的书。”我还能记起那本书的样子。那本书书名是《The Rise and Fall of the Third Reich》（第三帝国的兴衰），William L. Shirer 所著，总共 1 264 页。我想我当时应该是 7 岁。

目 录

本书的赞誉	2.11 更多关于字符串	24
译者序	2.12 符号	27
序	2.13 数组	27
前言	2.14 散列	29
致谢	2.15 正则表达式	29
	2.16 自己的类	30
	2.17 获取一个实例变量	31
	2.18 对象问：我是谁	33
	2.19 继承、子类 and 超类	34
	2.20 参数选项	35
	2.21 模组	36
	2.22 异常	38
	2.23 线程	39
	2.24 管理分散的源文件	40
	2.25 本章回顾	41
第一部分 设计模式和 Ruby	第二部分 Ruby 中的模式	
第 1 章 使用设计模式创建更好的程序	第 3 章 使用模板方法变换算法	44
1.1 四人组	3.1 迎接现实中的挑战	45
1.2 模式的模式	3.2 将不变的部分独立出来	46
1.2.1 把变和不变的事物分开	3.3 探究模板方法模式	48
1.2.2 针对接口编程而不对实现编程	3.4 钩子方法	49
1.2.3 组合优先于继承	3.5 但是类型声明都去哪里了	51
1.2.4 委托、委托、委托	3.6 类型、安全和灵活性	52
1.2.5 你不会用到它	3.7 单元测试并非可有可无	53
1.3 23 种模式中的 14 种	3.8 使用和滥用模板方法模式	55
1.4 Ruby 中的设计模式	3.9 模板方法模式的实际应用	56
第 2 章 从 Ruby 起步	3.10 本章回顾	57
2.1 交互式 Ruby	第 4 章 使用策略替换算法	58
2.2 说 hello world	4.1 委托、委托、还是委托	58
2.3 变量		
2.4 Fixnums 和 Bignums		
2.5 浮点数		
2.6 这里没有原始类型		
2.7 但是有时没有对象		
2.8 true、false 和 nil		
2.9 判定、判定		
2.10 循环		

4.2 在策略和环境中共享数据	60	7.7 本章回顾	107
4.3 再说鸭子类型	62	第8章 使用命令模式完成任务	109
4.4 Proc 和代码块	63	8.1 子类大泛滥	109
4.5 快速而随性的策略对象	66	8.2 一个更简便的方法	110
4.6 使用和滥用策略模式	68	8.3 代码块即命令	111
4.7 策略模式的实际应用	68	8.4 用于记录操作的命令	112
4.8 本章回顾	69	8.5 使用命令取消一个操作	115
第5章 通过观察器保持协调	71	8.6 命令队列	117
5.1 随时待命	71	8.7 使用和滥用命令模式	118
5.2 更好的随时待命方法	73	8.8 命令模式的实际应用	119
5.3 提取可被观察能力支持的代码	75	8.8.1 ActiveRecord 数据移植	119
5.4 使用代码块作为观察器	78	8.8.2 Madeleine	119
5.5 观察器模式的异体	79	8.9 本章回顾	123
5.6 使用和滥用观察器模式	80	第9章 使用适配器填补空隙	124
5.7 观察器模式的实际应用	81	9.1 软件适配器	124
5.8 本章回顾	83	9.2 几乎错过	127
第6章 使用组合模式将各部分组成 整体	84	9.3 适配器的另一种选择	128
6.1 整体和部分	84	9.4 修改单个实例	129
6.2 创建组合	86	9.5 适配还是修改	131
6.3 使用运算符将组合模式打扮整齐	89	9.6 使用和滥用适配器模式	131
6.4 给予数组的组合类	90	9.7 适配器模式的实际应用	132
6.5 一种不方便的差异	91	9.8 本章回顾	132
6.6 给各层次指明方向	91	第10章 通过代理来到对象面前	134
6.7 使用和滥用组合模式	93	10.1 使用代理进行拯救	134
6.8 组合模式的实际应用	94	10.2 保护代理	136
6.9 本章回顾	96	10.3 远程代理	137
第7章 通过迭代器遍历集合	97	10.4 虚拟代理让人变懒	138
7.1 外部迭代器	97	10.5 取消代理的苦差事	140
7.2 内部迭代器	99	10.5.1 消息传递和方法	140
7.3 比较内部迭代器和外部迭代器	100	10.5.2 method_missing 方法	141
7.4 无可比拟的 Enumerable	101	10.5.3 发送消息	142
7.5 使用和滥用迭代器模式	103	10.5.4 无痛的代理	142
7.6 迭代器的实际应用	104	10.6 使用和滥用代理	145
		10.7 代理模式的实际应用	146
		10.8 本章回顾	147

第 11 章 使用装饰器改善对象	148	13.2 模板方法再露一手	176
11.1 装饰器: 丑陋代码的解药	148	13.3 参数化的工厂方法	178
11.2 正式装饰	153	13.4 类也是对象	181
11.3 减轻代理的郁闷	154	13.5 坏消息: 你的程序搞大了	182
11.4 实现装饰器模式的另一种动态方法	155	13.6 对象创建的组合	183
11.4.1 装饰方法	155	13.7 再论类就是对象	185
11.4.2 使用模组进行装饰	155	13.8 协调名字	186
11.5 使用和滥用装饰器模式	156	13.9 使用和滥用工厂模式	187
11.6 装饰器的实际应用	157	13.10 工厂模式的实际应用	188
11.7 本章回顾	158	13.11 本章回顾	189
第 12 章 使用单例确保仅有一个	159	第 14 章 通过生成器简化对象创建	190
12.1 一个对象, 全局访问	159	14.1 制造计算机	190
12.2 类变量和类方法	159	14.2 多态生成器	193
12.2.1 类变量	159	14.3 生成器能保证产生健全的对象	196
12.2.2 类方法	160	14.4 可重用的生成器	196
12.3 Ruby 单例应用的第一次尝试	162	14.5 使用魔术方法制作更好的生成器	197
12.3.1 管理单个实例	163	14.6 使用和滥用生成器模式	198
12.3.2 确保只有一个	163	14.7 生成器模式的实际应用	198
12.4 单例模组	164	14.8 本章回顾	199
12.5 勤性单例和惰性单例	165	第 15 章 使用解释器组建系统	200
12.6 其他单例模式的实现方法	165	15.1 用以完成任务的正确语言	200
12.6.1 使用全局变量作为单例	165	15.2 构建一个解释器	201
12.6.2 使用类作为单例	166	15.3 一个文件查找解释器	202
12.6.3 使用模组作为单例	167	15.3.1 查找所有文件	203
12.7 安全带还是拘束衣	168	15.3.2 根据文件名查找文件	203
12.8 使用和滥用单例模式	169	15.3.3 大文件和可写文件	204
12.8.1 实际上就是全局变量	169	15.3.4 加入 Not、And 和 Or 的更复杂的 检索	205
12.8.2 到底需要多少这样的单例	170	15.4 创建 AST	207
12.8.3 在需要知道的基础上的单例	170	15.4.1 一个简单的分析器	207
12.8.4 减轻测试的郁闷	172	15.4.2 没有分析器的解释器	209
12.9 单例模式的实际应用	173	15.4.3 让 XML 和 YAML 进行分析 工作	210
12.10 本章回顾	173	15.4.4 为更复杂的分析器使用 Racc	211
第 13 章 使用工厂模式挑选	174	15.4.5 让 Ruby 作分析	211
正确的类	174	15.5 使用和滥用解释器模式	211
13.1 一种不同的鸭子类型	174		

15.6 解释器模式的实际应用	212
15.7 本章回顾	213

第三部分 Ruby 的设计模式

第 16 章 采用域指定语言打开系统	216
16.1 特定语言的域	216
16.2 备份文件的 DSL	217
16.3 是数据文件，更是程序	217
16.4 构建 PackRat	218
16.5 将 DSL 合成一体	220
16.6 评估 PackRat	221
16.7 改进 PackRat	222
16.8 使用和滥用内部 DSL	224
16.9 内部 DSL 的实际应用	225
16.10 本章回顾	226

第 17 章 使用元编程创建自定义

对象	227
17.1 通过方法度身定制的对象和方法	227
17.2 通过模块自定义对象和模块	229
17.3 召唤出崭新的方法	230
17.4 到对象的内部看看	233

17.5 使用和滥用元编程	234
17.6 元编程的实际应用	235
17.7 本章回顾	238

第 18 章 惯例优于配置

18.1 一个优秀的用户界面——对于 开发者	240
18.1.1 预期需求	240
18.1.2 让他们说一次	240
18.1.3 提供一个模板	241
18.2 一个消息中转器	241
18.3 选择适配器	243
18.4 载人类	244
18.5 增加一些安全性	246
18.6 帮助用户开始使用	248
18.7 评估消息中转器	249
18.8 使用和滥用惯例优于配置模式	250
18.9 惯例优于配置的实际应用	250
18.10 本章回顾	250

第 19 章 总结

附录	254
----------	-----

第一部分

设计模式和Ruby