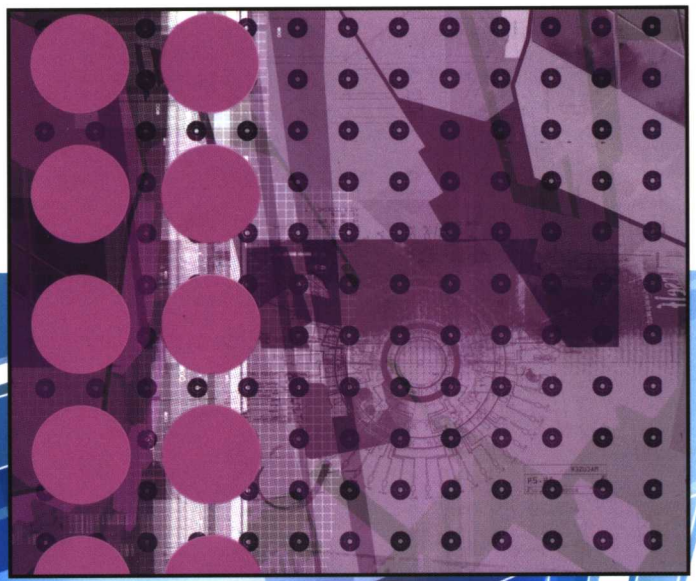
PEARSON
Addison Wesley

程序设计的模式语言·卷4

Pattern Languages of Program Design 4



Neil Harrison
Brian Foote 著
Hans Rohnert
北京SPIN 译

清华大学出版社

程序设计的模式语言·卷4

Neil Harrison　Brian Foote　Hans Rohnert　著

北京 SPIN　译

清华大学出版社

图书在版编目(CIP)数据

程序设计的模式语言. 卷 4/（美）哈里森（Harrison, N.），（美）福特（Foote, B.），（美）罗耐特（Rohnert, H.）著；北京 SPIN 译. —北京：清华大学出版社，2004.11
书名原文：Pattern Languages of Program Design 4
ISBN 7-302-09726-7

I. 程… II. ①哈… ②福… ③罗… ④北… III. 程序设计语言学 IV. TP311.1

中国版本图书馆 CIP 数据核字（2004）第 105330 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社总机：010-62770175　　**客户服务**：010-62776969
责任编辑：常晓波
封面设计：立日新
印 装 者：北京鑫海金澳胶印有限公司
发 行 者：新华书店总店北京发行所
开　　本：185×230　**印张**：39.75　**字数**：889 千字
版　　次：2004 年 11 月第 1 版　2004 年 11 月第 1 次印刷
书　　号：ISBN 7-302-09726-7/TP・6724
印　　数：1~3000
定　　价：64.00 元

序　言

关于 Phish 和 Phugues

时间：1621 年；地点：普利茅斯（Plymouth），此处最后变为马萨诸塞州（Massachusetts）。1620 年 11 月抵达普利茅斯的一群英格兰定居者要开始种植农作物了。不久以后，一位名叫 Squanto 的印第安人前来拜访。他显然是一位园艺爱好者，他提议给这些定居者教授种植技巧。首先是将鱼放入土里使土地肥沃，这种技巧加上其他一些新大陆（New World）的种植技巧带来了巨大的丰收。定居者顺利度过了随之而来的冬季，这在很大程度上要感谢 Squanto 和他明智的建议。

毫无疑问，这是一个虚假的故事。但是现代园艺家肯定了 Squanto 关于鱼在农业中的作用的见解。实际上，你可在许多园艺商店买到鱼肥。我们的农业祖先可能不具有植物生理学的深厚知识，但是他们知道什么有用，而且还把经验一代一代传下来。

一百多年以后，在半个地球远的地方，即现在的德国，一位多才多艺的大师展现了他的才华。传说 Johann Sebastian Bach 拜访普鲁士国王 Frederick，寒暄过后，国王要 Bach 为他演奏。Bach 在风琴旁坐下，即兴创作了一首 5 个乐章的赋格曲，自身也是作曲家的国王对 Bach 钦佩不已。

Bach 的作品是巴洛克音乐的经典。他把本领传给了后代——他的后代中的许多人也变成了重要的音乐家和作曲家。

知识代代相传。语言就是生动的例子，经久不衰，从口传单词和概念演变而来，从一个人传给另一个人。这方面的一个例子是英语，其源头可追溯到许多语言，包括德语、希腊语和拉丁语。这里有一个问题：英语的多方面继承带来了一个显然会使人混乱的拼写规则的大杂烩。比如，在“physics”中，我们把字母 ph 拼作“f”音；而在“fish”和“fugue”中，则把“f”拼作“f”音。对于这种不规范性有其历史和语言学的原因，不过这些对我们没有造成影响。我们只是记住了，它是“fugue”而不是“phugue”。

在软件中，我们没有数个世纪的历史可以依靠。我们没有从先辈那里学会农业，也不能追根溯源到中世纪 Chaucer 时代的英语。但我们存在的时间已足够我们学会一些事情了。其中最重要的一条就是：*我们必须分享我们学到的*。如果我们都把知识占为己有，像林鼠那样把其存储起来，那么我们所从事的领域肯定会停滞不前，我们注定要使同行和继任者重复我们犯过的错误。慢慢地，重新发明取代了创新。进步减慢了，领域变得愈加贫瘠。

模式是知识的导管，可以用来获取和传送经过时间证明的实践。模式不仅仅是技巧或

者看起来很随意的拼写规则，它们可以传递理解。它们不仅仅告诉你 what 和 how，还告诉你 why 和 when，这是它们的真正力量所在。

我们能从模式那里合理地期望得到什么？也许它们会帮助其他人开发出比我们所开发的更好的软件；也许它们会允许人们把他们做的工作建立在我们所做工作的基础上。我们肯定希望模式能帮助其他人避免我们遇到过的陷阱。

不过我们可以期望更多。计算机技术的广泛传播对社会施加了巨大的影响，从而必须负责任地运用它们。虽然模式不会强制你负责任地做事，但它们能减轻重新发明的痛苦，把你解放出来以考虑更高层次的目的。最终，模式使得每个人（软件用户和开发人员）的生活更加美好。

无疑，目标有高有低。我们希望“程序设计的模式语言”系列的第 4 卷能实现这些目标。

致谢

现在要对为本书作过贡献的人表示感谢。如果不是很多人一起努力，这么厚的图书是不会出现的。我们非常感谢他们的工作。我们特别感谢为本书提供必需稿件的作者，他们为我们解决了很多疑难问题。我们并不仅仅是指可以在本书中找到的作者，还包括那些其作品没有采纳的大约 60%的提交者。极高质量的提交稿件尽管加大了我们工作的难度，但是它确保了本书的质量。

我们聘用了大量的审稿人帮助筛选提交稿件。我们把我们的理智归功于他们：Francis Anderson, Brad Appleton, Jorge Arjona, Owen Astrannen, Ken Auer, Jeff Barcalow, Kent Beck, Mike Beedle, Steve Berczuk, Manish Bhatt, Rosana Braga, John Brant Kyle Brown, Jose Burgos, Frank Buschmann, Andy Carlson, Ian Chai, Alistair Cockburn, Jens Coldewey, James Coplien, Ward Cunningham, David Cymbala, Fonda Daniels, Dennis DeBruler, Michel DeChamplain, David Delano, Dwight Deugo, Paul Dyson, Philip Eskelin, Javier Galve, Julio Garcia, Alejandra Garrido, John Goodsen, Robert Hanmer, Kevlin Henney, Robert Hirschfeld, Ralph Johnson, Wolfgang Keller, Elizabeth Kendall, Norm Kerth, Charles Knutson, Frederick Koh, Philippe Lalanda, Manfred Lange, Doug Lea, Mary Lynn Manns, Klaus•Marquardt, Paulo Masiero, Skip McCormick, Regine Meunier, Oscar Nierstrasz, James Noble, Alan O'Callaghan, Don Olson, William Opdyke, Dorina Petriu, Irfan Pyarali, Andreas Rausch, Dirk Riehle, Linda Rising, António Rito Silva, Don Roberts, Gustavo Rossi, Cecilia Rubira, Andreas Rüping, Doug Schmidt, Ari Schoenfeld, Dietmar Schütz, Christa Schwanninger, Joe Seda, Peter Sommerlad, Michael Stal, Paul Taylor, Jenifer Tidwell, Dwayne Towell, David Ungar.

我们特别感谢本系列丛书的主编 John Vlissides。他给了我们鼓励和必要的鞭策。Neil

要特别感谢他的两位合作编辑，和你们一起工作是一种快乐。

最后，我们特别感谢支持我们一路走来的我们的家人、朋友和合作者。我们希望在你们读到这里时，我们将重新做回快乐的自己。

Neil Harrison，丹佛，科罗拉多州
Brian Foote，乌尔班纳，伊利诺伊州
Hans Rohnert，慕尼黑，德国

前言（一）

如果你是一位软件开发人员，可能会有太多的事情要做，而你也许不知道如何将它们做全了。可能你对模式有所耳闻，因为在各种各样的消息源（比如计算机期刊和华尔街日报）中提及过模式。你可能也会认为模式能帮助你快速找到问题的答案。不过，如果你在寻找特定问题的快速解决方案，本书中的很多内容并不适合你。但如果你是在寻找一问题的经过证明的解决方案，请继续阅读。

编写计算机软件与许多其他学科不同。软件设计师创建抽象的结构，甚至在用户界面中也只能看到他们的一小部分工作。硬件设计师能触摸到他们的创造物，但是软件设计师无法触摸或看到他们所创建的成果，他们只能看到他们的创造物是如何运行的。你可以观察罗马高架渠的拱，并弄明白如何建造一个拱。但是不能观察计算机系统，并弄明白它是如何工作的。眼睛看不到很多错综复杂的情况。本书所介绍的模式是一些作者的成果，这些作者注意到了软件系统内部的规律性和重复出现的结构，他们会解释这些规律和结构，以帮助你避免犯错误。

本书中的模式是 PLoP（Pattern Languages of Programming，程序设计的模式语言）会议的成果。与会者说，PLoP 会议是他们所参加过的最好的技术性会议之一，原因之一是 PLoP 把实践者聚集在一起讨论问题的解决方案。它们也给模式作者提供了一个支持平台来分享他们最新的创造性工作，PLoP 就是关于经验分享的。

PLoP 的主要目标是改进模式文学和模式写作的质量。在 PLoP '97 上，我们营造了一种支持作者创造性的氛围。除了传统的作者研讨会（Writers' Workshops）之外，也举行写作研讨会，给作者培养他们创造性的机会。在 PLoP '98 上，我们没有按照传统的标准给模式提交稿件归类。我们想要看看研讨会参与者是否能发现他们模式之间的联系，并带着构建新联系的想法离开大会。有一些人已经这样做了，并且出现了许多正在进行中的受本次大会启发产生的协作（比如组件设计和配置管理模式）。

模式作者从 Christopher Alexander 身上得到灵感。Christopher Alexander 是一位建筑师，他试图从其他事务中捕获使建筑漂亮的“无名的性质”（Quality Without a Name）。软件开发人员从 Christopher Alexander 的工作中得到的重要成果不是与建筑相关的事务，而是通过美化来改进质量的方法的实践价值。Donald Knuth 在其 1974 年图灵奖（Turning Award）演说中说道：

“我感到，当我们为程序做准备时，经历就好比是写诗或编曲；当我们阅读他人的程序时，我们能够承认其中一些程序是真正的艺术品。有些程序很优雅，有些很精美，有些闪烁着智慧的灵光。我认为，编写出宏伟的、高贵的、真正华丽的程序是可能的。”

在你认为这个目标过于理想化或者抽象之前，请记住，模式是关于经过证明的解决方案的，而不是关于新鲜和独特的解决方案的。在 Alexander 的自传 *Christopher Alexander: the Search for a New Paradigm in Architecture* 中，Stephen Grabow 描写了 Alexander 对简单时髦建筑的厌恶。Alexander 的模式描述“显而易见的事物”，这由于我们沉迷于时髦和潮流中而被我们中的大多数人所忽略了。尽管实际情况是新的和大胆的建筑物通常不适合于居住和工作，但这些建筑物常常获得奖项。在软件行业也是如此。较之更加陈旧的经过证明的想法，某些软件开发文化更青睐新鲜的未经证明的想法，其重点常放在发明上。但是优美的技巧并不总是产生更好的程序。除非你的问题与众不同（从正确的尺度来看，与众不同的问题是很罕见的），不然新颖并不总是好的。你常常会发现，你所碰到的优雅的、效率高的解决方案实际上之前已经被许多其他方案使用过了，尽管可能是在孤立的状态下被使用。模式是这样一种尝试：它试图捕获那些经过长时间观察的显而易见的解决方案。

从某种意义上说，软件的发展就好比是传统医学。不幸的是，今天的计算机编程是许多许多代之前的医学：个人试图解决问题，发现不想要的副作用，做个人记录以便后代能从他们的经验中获益。模式试图将记录传播到更广泛的群体中，而不仅仅是传给合作者。它们是关于事情的故事——我们所看到的从不久的以前一直到现在起作用的事情。它们是记录，我们可以把记录传给后面的设计师以便他们能避免同样的一些错误。PLoP 上的研讨会支持分享和确认经验的过程。作者提交他们推荐作为通用解决方案的模式。研讨会和其他论坛的参与者添加他们的故事，要么确认要么否认这些方案是模式。模式作者在系统内部寻找创建结构的底层结构和过程，然后把他们的观察记录下来，以便其他人能分享他们的经验。由于本书中的论文所涉猎的过程的本质，你能确信，它们不是一些具有创造性的人的想法。本书和更加传统的学术作品的区别在于，在本书中没有任何东西是新的，它可能只是个别读者还没有发现的事物。

模式封装了多年的程序开发、观察和经验。找到解决方案很简单，但要找到正确的解决方案，就需要理解问题和影响问题的强制条件。书中的内容会帮助你做出困难的决定。它以某种方式让你接触到一些经过证明的问题的解决方案，你能从那些初看上去差别不大的若干方法中找到你想要的方案。这些模式能帮助你，但是它们不会将方案交给你。在某种意义上，模式的价值和威力把被它们接受为有用事物的方式掩盖了。*Design Patterns*（《设计模式》）[Gamma+1995]被认为是一本有用的书，它描述了每个对象开发人员应该熟悉的思想。*Design Patterns* 捕获了关键习语，后面紧跟关于它们的适用性的合理建议。这是一本每个开发人员都应拥有的书，不过，它只涉及了模式是关于什么的表面知识。

从某种意义上说，模式是关于对美丽和优雅的追求。模式也是关于遵循其他成功系统，快速构建可靠系统的需要。通过提供给你机会学习其他人的经验，模式将你从重复的任务中解放出来，允许你集中精力于创新和发明。模式作者来自开发过程的各个方面，从管理人员和软件架构师到普通开发人员。每一个与软件系统和开发有关的人都有能力观察和记录那些运行良好的重复出现的主题。

通常人们（特别是软件领域人士）喜欢归类。许多系统会比最初看到的有更多的共性。当浏览本书的目录寻找答案时，不要仅仅因为标题不符合你对问题的直接感觉，而把特定章节当作介绍性内容放弃。如果你是在用 Smalltalk 编程，尽管你可能不会马上用到 Coplien 的 C++模式，但是如果你不得不设计很普通的 I/O 时，你会发现 Hanmer 和 Stymfal 的第 23 章中的模式会很有用。本书中的章节按多种方式分类，你这里所看到的类别是各种可能性的最好折中。本书中的模式的作者共享他们的智慧和创造性，以帮助你从他们在各种各样系统中所看到的事物中获益。有些人写作的内容可用于帮助新手学会如何用正确的方法做事，或者至少是帮助他们尽可能减少错误。有些人通过收集跨领域的问题/解决方案空间的模式语言，来全面记载领域。我们有关模式的工作只是开了个头。《程序设计的模式语言》系列以及其他一些工作是捕获软件开发专家经验的尝试，组织这种专家经验以使其可用的工作正在进行中。也许作为读者的你能帮助找到这些模式作者、审稿人和编辑所没有发现的联系和应用。

当连接到其他模式时，模式就显示出了它的威力。这是人们所称为“背景”的模式，即在一个“更高层次”上存在的模式。使用组织内部的特定工具、过程和语言，从体系结构构建软件系统。本书中的模式涵盖了所有这些方面，并且每件作品都独自成文。阅读时，注意模式与作者所关注的领域之外的模式可能的应用之间的联系。

本书中的思想可能对你而言不全是新的。如果你阅读到其中某一章，会心地发出“我知道这个！”的感慨，那你就承认了作者的工作。如果你在本卷中没有发现任何新的内容，那你可能是一位专家。事实上，没有一个软件开发人员是无所不知的。本书和本系列中前面的书籍是构建经过证明的技术的文献体系过程的一部分，你可以使用这种体系，在其基础上构建你自己的工作和乐趣。

我们希望你会发现本书是有用的、有乐趣的，甚至是有启发性的。

Steve Berczuk

Bob Hanmer

Steve Berczuk 是 NetSuite Development Corporation 公司的软件工程师，也是 PLoP '98 的程序主席。

Bob Hanmer 是 Lucent Technologies 的知名技术人士，也是 PLoP '97 的程序主席。

参考文献

[Gamma+1995]E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

[Kernighan+1999]B. Kernighan and R. Pike. Finding Performance Improvements. IEEE Software, 16(2), 1999.

[Petzinger1999]J. Petzinger Jr. The Frontlines: To Get Machines to Talk to Each Other Two Men Write Human Language. The Wall Street Journal, April 14, 1999.

前言（二）

Ralph Johnson 和 Ward Cunningham 以“创立 PLoP 用于创建新文学” 开始本系列的第一本书，这已经是 6 年前的事情了。目前，若干有关模式的书籍在计算机文学的核心体系中占据了应有的地位。模式已经改变了我们构建软件的思考方式以及我们的工作方式。建立项目团队时，我们遵循组织模式；分析领域时，我们应用分析模式组。我们中的很多人进行设计时，如果桌上没有著名的 *Gang of Four* 这本书，会感到心里空荡荡的。从 OOPSLA 上的研讨会开始，模式在不同的时间影响了软件产业以及计算机科学。

那么什么因素造就了一种好的和我们不断寻找的模式呢？Jim Doble 和 Gerard Meszaros 在本系列前面的卷[Doble+1998]中对好的模式和模式语言的元素做了精彩的描述。而该书描述的是现象而不是起因，它描述了艺术家所要的技巧而不是艺术本身。读者经常谈到他们阅读一种好的模式时所得到的“温暖模糊的感觉”，或者谈到模式是如何与他们自己的经历产生“共鸣”的。许多计算机专业人士已经忘了倾听这些感觉，但是模式作者（和读者）正在学会接受它们。

有 3 种元素有助于造就一个优秀的模式作者：经验、努力工作和态度。一种好的模式讨论了带来解决方案的所有强制条件和伴随它的所有陷阱和不足。它展示了作者和所有那些协助编写模式的人的经验，还展示了投入于创建模式的无数心血——文字、结构和流程。但最重要的是，它展示了吸引读者的愿望。好的模式写出来是为了读的，其整个结构的创立是用于引导读者通读文章，给他们这种模糊感觉以及这种与文献和作者的共鸣。它的目的是在引导读者更深入地理解问题和其解决方案的同时吸引他们。

以人为本的态度是一种好模式的灵魂。它是模式团体的最深厚基础。在过去，模式的发展好比是一个宗教派别（尽管这有所争议），二者都是基于价值体系且拥有哲学基础。模式提倡软件中的人性。也许这可以解释对模式文学的广泛兴趣和痴迷，因为与软件系统开发和使用相关的人士逐渐认识到，构建软件是一种极其人性化的活动。

本卷中所有的模式都在 PLoP 大会上改进过。PLoP 以及后来的 EuroPLoP 和 ChiliPLoP 创立用于培育模式和模式作者，鼓励作者写下他们下意识应用的模式并鼓励他们为 Johnson 和 Cunningham 的“新文学”做出贡献。我们相信，PLoP 是一种与众不同的会议，不同之处在于我们关注软件开发的人的活动中的人。会议做了巨大努力来营造一种舒适和欢迎的氛围。它是仅有的带有小憩时间的会议。EuroPLoP 限制与会人数，并不是因为它是排外的，而是为了确保小规模带来的亲密感和凝聚力。所有的 PLoP 会议都提供了许多社交活动和服务（从协作游戏到为那些不希望家人因自己的职业而受影响的人照顾孩子），这些活动和服务有助于建立一个圈子，高度集中而富有效率的工作氛围也由此产生。

作者研讨会是工作的重心，你在本书中所发现的所有模式都至少经历过一场研讨会。研讨会提供了一种支持和安全的环境，在其中作者能从同辈的其他与会作者那儿得到反馈。许多第一次参加 PLoP 会议的人认为他们到这里是来讲解他们的文章的，但是结果发现自己只是不被允许发言的人。你必须学会倾听你的读者，而不是希望他们能倾听你。也许这就是使得模式如此成功的原因。模式是由在自身领域中有专业经验的普通人写出来的，这些人想要提高他们的作品质量，也想帮助其他人提高作品质量。

Jens Coldewey

Paul Dyson

Jens Coldewey 是德国慕尼黑的独立顾问，也是 EuroPLoP'99 的程序主席。

Paul Dyson 是 Big Blue Steel Tiger 有限公司的经济技术部门的主任，也是 EuroPLoP'99 的程序主席。

参考文献

[Doble+1998]J. Doble and G. Meszaros. A Pattern Language for Pattern Writing. In R. Martin, F. Buschmann, and D. Riehle (eds), Pattern Languages of Program Design 3. Reading, MA: Addison-Wesley, 1998.

目　录

第 1 部分　基本面向对象模式 …… 1

第 1 章　Abstract Class（抽象类） …… 4
第 2 章　Role Object（角色对象） …… 13
第 3 章　Essence（本质） …… 28
第 4 章　Object Recursion（对象递归） …… 36
第 5 章　Prototype-Based Object System（基于原型的对象系统） …… 46
第 6 章　Basic Relationship（基本关系）模式 …… 63

第 2 部分　面向对象基础结构（Object-Oriented Infrastructure）模式 …… 79

第 7 章　Abstract Session（抽象会话）：一种对象结构模式 …… 81
第 8 章　Object Synchronizer（对象同步器） …… 94
第 9 章　Proactor（主动反应） …… 112

第 3 部分　编程策略 …… 139

第 10 章　C++语言模式 …… 141
第 11 章　Smalltalk 体系结构模式 …… 172
第 12 章　存储器维护协会的 High-Level and Process（高级和处理）模式：
管理有限的存储器模式 …… 193

第 4 部分　时间 …… 209

第 13 章　Temporal（时间）模式 …… 210
第 14 章　History（历史）模式集合 …… 229

第 5 部分　安全性 …… 262

第 15 章　支持应用程序安全性的体系结构模式 …… 264
第 16 章　Tropyc：面向对象的加密软件的模式语言 …… 299

第 6 部分　面向领域的模式 …… 333

第 17 章　使用查询对象创建报表 …… 335

第 18 章 特征抽取：用于信息检索的模式 …… 350
第 19 章 有限状态机模式 …… 368

第 7 部分 人机交互模式 …… 394

第 20 章 设计可导航信息空间的模式 …… 396
第 21 章 创建可重用的多媒体人工制品 …… 412
第 22 章 显示维护：一种模式语言 …… 437
第 23 章 一种输入和输出模式语言：从远程通信中学到的经验 …… 449

第 8 部分 评审 …… 480

第 24 章 确定支持者：用于程序委员会的组织模式语言 …… 482
第 25 章 用于作者研讨会的模式语言 …… 499

第 9 部分 管理软件 …… 521

第 26 章 客户交互模式 …… 523
第 27 章 有能力的、有生产力的、满意的：一些用于保护生产人员的组织模式 …… 549
第 28 章 SCRUM：用于超生产力软件开发的模式语言 …… 574
第 29 章 大泥球 …… 587

第 1 部分　基本面向对象模式

可以说，此故事在 1987 年开始于 Disney World（迪斯尼世界）附近。你可能会问是什么故事？就是关于对象和模式错综复杂交织的故事。该故事开始于在 Orlando 举行的 OOPSLA'87 研讨会，在会上，Ward Cunningham 和 Kent Beck 将离经叛道的建筑师 Christopher Alexander 关于模式和模式语言的概念引入计算中，提出为什么我们在设计软件时不能使用类似 that 和 what 的事物呢？

快速回到 10 年前，Alexander 和同事所著的 *A Pattern Language* 一书出版，书中包含了 253 种不同“模式”的集合，这些模式是应用于建筑上而不是软件的。它们描述了针对出现在城镇选址及不同类型公共场所和私人住宅的设计和建造中出现的问题提出的解决办法，同时该著作也是对 Alexander 及其同事所生活年代的正统的钢铁玻璃的现代建筑的回应。A Pattern Language 所收集的智慧并非来自现代主义理念的简易书籍，而是来自上万年的被验证过的真实的人类建筑经验。每种模式描述了一个可能翻来覆去出现的问题，并且提供了一种可使用无数次而决不重样的解决方案。

无数次？决不重样？传统上，软件开发人员用一个方案两次就很困难了，更别说上百万次了。向后看，模式在日新月异的面向对象编程领域找到一席之地就毫不令人惊讶了。对象看来是模式的天然中介物，它被证明好比是建筑的粘结剂。

因而，对于大多数人来说，故事实际上源于 1994 后期，以 *Gang of Four* 的 *Design Patterns*：*Elements of Reusable Object-Oriented Software*[Gamma+1995]为标志。该书探索模式和对象间的联系，描述了面向对象设计中递归式的主旨，以及对象如何能成为表达和联系设计及构建这些设计的构建块的中介。

我们也从这里开始我们的故事吧。

第 1 章：Abstract Class（抽象类），作者：Bobby Woolf。

可以认为 Abstract Class（抽象类）模式是 Gang-of-Four 摘要。抽象类是几乎所有模式的基础的一部分。作为接口，抽象类在以显示类型化语言实现时可以是基础元素。甚至在动态类型化语言，抽象类既可以用作实际的接口，也可以用作模板类。抽象类是面向对象框架搭建的生成器。事实上，Gang of Four 认识到了抽象类的重要性，但是担心使它们成为显式模式会导致蠕虫，因为所有在一个由其他模式构建的模式中是隐式的问题会被带入一个已经很复杂的故事中。他们推论出，让后人筛选出来比正面应付当时的层次关系要好得多。从后来的观点看，这可能是明智的选择。Bobby 的贡献最终使得这种无所不在的结构进入已发表的模式的殿堂。

第2章：Role Object（角色对象），作者：Dirk Bäumer、Dirk Riehle、Wolf Siberski和Martina Wulf。

随着对象的成熟，对于不同的用户或者系统的不同部分来说，考虑同一主题事务的不同方面是很正常的。Role Object模式显示了如何结合使用多个对象，以使得不同角色、方面或者视图附着到一个未工作的对象上。该对象的核心身份（core identity）由一个对象表示，而用动态Role Object来为各种各样暂时的关系建模。例如，在你的Reader角色中对系统建模可能实现isBored（），而一个Employee角色可能实现salary（），一个核心Person对象则可能实现name（）。传统的面向对象语言并不直接支持Roles，但是动态协作对象可以使用Role Object模式实现它们。对象和模式之间的这种整合效果有助于解释它们的巨大威力和持久魅力。

第3章：Essence（本质），作者：Andy Carlson。

模式和协作对象的威力的又一个例子可在Essence模式中看到。Essence是一种创造性的模式，它确保在创建一个对象之前，会完全初始化该对象所有重要的强制性字段。它用一对协作对象来确保满足限制要求。这种模式再一次演示了对象如何协同工作，以克服单个对象的缺点和编程语言的局限性。

第4章：Object Recursion（对象递归），作者：Bobby Woolf。

Object Recursion模式填补了模式文学中的另一项空白。它升华了有经验的面向对象程序员多年来用于真实的模式捕集器的神圣的递归式、多态的、分布解决的策略。同样地，它捕获位于相关的Gang-of-Four模式（如Composite、Chain-of-Responsibility和Decorator）之间或超越这些模式的意图。

第5章：Prototype-Based Object System（基于原型的对象系统），作者：James Noble。

有时，你可能会对底层的编程语言提供的能力感到不满意，或者你可能希望将你系统的威力和能力提供给你的用户。Object System（对象系统）模式显示了如何通过从对象本身构建自己的对象系统来实现这个目标。变量、单元、方法，甚至对象本身都由这种对象系统中的对象表示。有时，诸如Interpreter、Visitor、Type Object和Comand之类的设计模式全都看起来像是沿着构建特定于领域的语言的道路前进。在构建一个完全Object System时可以采用这种方法得出其逻辑结论。对象适合于构建程序，也适合于构建编程语言。

当然，当你构建自己的Object System时，可以用你自己的方式构建对象。你可以添加诸如保护或者约束满意度的能力到变量或用于查询类或类型对象的特殊方法上。

第6章：Basic Relationship（基本关系）模式，作者：James Noble。

James对本部分的第二个贡献是研究了对象如何用于更好地建模和表示不同类型的对象-对象关系。这些理念中的很多迄今为止只存在于纸上及模型符号和工具的像素卡通世界中。James的模式展示了如何构建结构，以实现这些对象之外的理念。

第一种模式，Relationship as Attribute，描述面向对象程序员的第一种倾向为其他模式搭建了施展的舞台。这种倾向就是用字段或者实例变量实现简单的、一对一的、单向

（one-way）关系。当关系变得更复杂时，可把这些问题作为独特的 Relationship Object（关系对象），这样更好处理。面向对象程序员常常把 Collection Objects（集合对象）用于一对多（one-to-many）关系。需要把更多精力花在存储值部分的情况，比如协调 Observers，可使用 Active Value 模式。对象不直接支持复杂的双向关系，但可以用成对出现的单向关系模拟出这种效果，在 Mutual Friends 模式中描述了这种方法。

自从 Gang-of-Four 图书出现，我们的模式文集已成长起来，它包含软件体系结构、过程、和多种领域，这一点读者可在翻阅本卷时看到。然而在很多方面，由 Gang of Four 开启的丰富的面向对象设计宝矿仍然是我们的谋生之计，说我们已经完全吃透了这个领域还为时过早，但我们确实填补了一些空白。

参考文献

[Alexander+1997] C. Alexander. A Pattern Language: Towns/Buildings/Construction. New York: Oxford University Press, 1977.

[Gamma+1995] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

第 1 章　Abstract Class（抽象类）

Bobby Woolf

分类

类行为

目标

在暂缓传递特定实现细节给子类时，为类的层次结构定义接口和通用实现。Abstract Class 在允许子类关注其实现的区别时，还提供了对多态类的支持。

也称为

Template Class（模板类）[Woolf1997]，Base Class（基类）[Auer1995]

动机

考虑执行简单算术运算的需要，几乎每个应用程序都需要使用诸如整数和浮点数之类的数字来进行加、减、乘、除这样的算术运算。

执行这些简单数学运算的一个显然的办法是让 CPU 来做。所有现代的 CPU 都有内置的指令，用整数和浮点数执行算术运算，这是进行这种计算最有效率的方式。

问题在于并不是所有数值量都能完全以 CPU 的整数和浮点数表示。整数有有限的范围；浮点数精度有限，在十进制和二进制之间转换时会丧失精度。

健壮的面向对象系统的数字框架应随时充分利用 CPU 的效率。然而，为了使得系统更加健壮，框架也应能随时克服 CPU 的局限性。它应能表示实际上无限范围的数，无论是无穷大的数还是无穷小的数。它应能以完整精度表示十进制数，至少应能表示出小数位的特殊数字。它应能在不丧失精度的情况下进行简单的算术运算，甚至能够先简化复杂公式然后再对其进行计算。

一个健壮的数字库应用不同的类来实现这些目标：Integer 和 Float 用于 CPU 数字；LargePositiveInteger 和 LargeNegativeInteger 用于超大整数值；FixedPoint 用于完整精度；