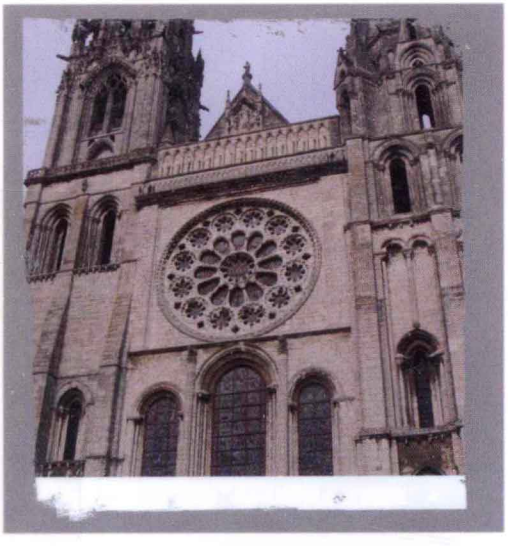
TURING 图灵程序设计丛书

Pattern-Oriented Software Architecture Volume 1
A System of Patterns

面向模式的软件架构
模式系统



卷1

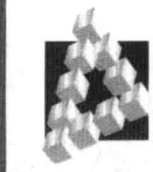
- POSA系列开山之作
- Jolt大奖图书

[德] Frank Buschmann
[德] Regine Meunier
[德] Hans Rohnert 著
[瑞士] Peter Sommerlad
[德] Michael Stal

袁国忠 译

人民邮电出版社
POSTS & TELECOM PRESS

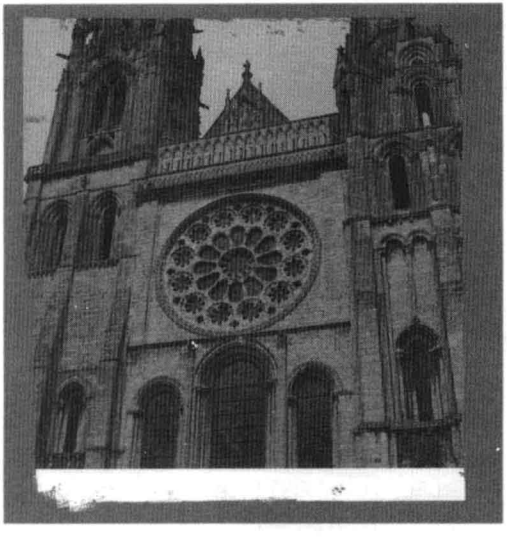
TURING 图灵程序设计丛书



Pattern-Oriented Software Architecture Volume 1
A System of Patterns

面向模式的软件架构
模式系统

卷1



[德] Frank Buschmann
[德] Regine Meunier
[德] Hans Rohnert 著
[瑞士] Peter Sommerlad
[德] Michael Stal
袁国忠 译

人民邮电出版社
北京

图书在版编目（CIP）数据

面向模式的软件架构. 第1卷, 模式系统 / (德) 布施曼 (Buschmann, F.) 等著 ; 袁国忠译. -- 北京 : 人民邮电出版社, 2013.11
(图灵程序设计丛书)
书名原文: Pattern-oriented software architecture volume 1 : a system of patterns
ISBN 978-7-115-33215-8

Ⅰ. ①面… Ⅱ. ①布… ②袁… Ⅲ. ①软件设计 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2013)第234654号

内 容 提 要

面向模式的软件架构系列丛书被公认为程序员必读经典。本书是该系列丛书的第 1 卷，涵盖模式系统的方方面面。

本书分 8 章，第 1 章系统介绍模式的概念，讨论描述模式的原则；第 2 ～ 4 章讲解模式编目，分别阐述了架构模式、设计模式和成例；第 5 章揭示如何将模式组织成模式系统及其重要性；第 6 章探讨将模式融入软件架构的方法；第 7 章概述模式的历史、相关著作及模式界；第 8 章展望模式未来的发展方向。本书最后还给出了表示法、术语表、参考文献和索引，方便读者阅读及进阶。

本书适合软件架构师、设计师和开发人员阅读，对计算机专业的学生也大有裨益。

◆ 著 [德] Frank Buschmann [德] Regine Meunier [德] Hans Rohnert [瑞士] Peter Sommerlad [德] Michael Stal
译 袁国忠
责任编辑 刘美英
责任印制 焦志炜

◆ 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 电子邮件 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：19.5
字数：474千字 2013年11月第1版
印数：1－3 000册 2013年11月北京第1次印刷
著作权合同登记号 图字：01-2012-1948号

定价：69.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

版权声明

献　　词

谨以此书献给 Martina。

——Frank Buschmann

谨以此书献给 Michael、Anja 和 Sandro。

——Regine Meunier

谨以此书献给 Ute。

——Hans Rohnert

谨以此书献给 Andrea。

——Peter Sommerlad

谨以此书献给 Gisela、Macho 和 Merlin。

——Michael Stal

关 于 本 书

本书探讨软件架构模式（简称“模式”）。近年来模式始终吸引着大量眼球，关于模式的研讨会、研究班、论坛、学术论文和专著层出不穷，模式界甚至召开了专门会议。这些有关模式的讨论如火如荼，让这个主题看起来处于浪潮的巅峰。

模式为何如此激动人心呢？可能是因为模式被认为是草根阶层所依赖的训练有素的设计师和软件工程师集体经验的结晶。对于很多反复出现的设计问题，专家都已找到解决方案，而模式以易于理解且规范的格式记录了这些经过实践检验的解决方案。

我们希望本书让软件开发新手和专家都能受益。对于新手，本书让他们无需积累多年经验，就能像专家一样应对中等规模的项目；对于专家，本书有助于他们设计具有指定特征且复杂而庞大的软件系统，并学到其他专家的经验。无论是新手还是专家，当你面对特定的设计问题时，本书都将助你找到久经考验的解决方案和替代方案。

本书既可作为教材，又可作为参考指南。它帮助软件开发人员以全新方式思考软件架构，同时针对反复出现的特定问题提供了大量解决方法。用作软件工程课程的教材时，本书将让学员以全新视角审视大型软件系统设计；作为参考手册时，本书介绍的技巧可立即付诸实践。书中包含大量实际使用这些模式的指南以及应满足的约束条件。

采用规范格式记录设计智慧的理念可追溯到 Christopher Alexander[①]，他率先在建筑领域提出了模式的概念。在其著作《建筑的永恒之道》中，Christopher Alexander 阐述了如何将模式用于房屋建造及社区和城市规划。该著作的基本主题是，居住场所的设计不仅要实用、时尚，还需给人以舒适和慰藉。设计优异的建筑展示了其固有品质，这些品质易于意会，却难以言传。简言之，这样的建筑拥有“无名的质”。

软件工程领域采用这种方法的早期实验过度依赖于 Alexander 的风格，但最近软件界一直致力于寻找更适合软件设计的风格。大家尝试了多种模式描述格式，但并未形成统一意见。

---

① Christopher Alexander 是执业建筑师和城市规划师，还是加州大学伯克利分校的建筑学教授兼环境结构中心主任。他提出了一种建筑理论，就是将建造和规划建立在诠释和使用模式的基础之上。牛津大学出版社出版了一系列图书，介绍了这种理论本身、模式、开展的试验，以及这种方法遭到的批评。

虽然我们为寻找描述模式的良方做了巨大努力，但提出“模式风格”理论并非本书的主要目标，这绝不是我们编写模式专著的初衷。1991 年，我们以简单易懂的方式记录了第一批模式。虽然记录模式的风格得到了逐步改善，但我们很快就发现模式之间不是彼此孤立的，而是存在千丝万缕的联系。我们之所以编写专著，而不是发表一系列论文并在每篇论文中阐述一个模式，这是其中的一个动力。编写专著的缺点在于出版前需要很长的酝酿时间。这一点几十年前就尽人皆知，但拿出良好模式描述格式需要的时间之长，还是令我们大吃了一惊。

另外四位作者也有类似的经历。1994 年秋天，Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides 的开创性著作《设计模式：可复用面向对象软件的基础》出版，作者们被称为“四人组”（GoF）。虽然那时设计模式的概念已不再新鲜，但该著作推出了第一个面向对象设计模式编目，并对这些模式做了详尽描述。

与 GoF 采用的方法相比，我们的方法稍有不同，虽然有很多类似之处，也有些重叠的地方。GoF 的著作专注于设计模式，而本书介绍的模式涉及多个抽象层次：从高层的*架构模式*到设计模式，再到低层的*成例*（idiom）。我们还专注与面向对象无关的问题，并试图在模式描述方法中融入最新的洞见。我们的总体目标是帮助读者在软件架构这个更大的背景下利用模式。我们将这种方法称为*面向模式的软件架构*。我们讨论了*模式系统*，即不仅将模式收集到一个无所不包的容器中，还根据合适的标准对它们进行分组。GoF 的著作开了模式分类之先河，将模式分为创建型、结构型和行为型。我们则试图更上一层楼，根据更细致的标准进行分类，如交互性系统、适应性系统、工作组织、通信和访问控制。

我们鼓励使用者与同事分享该模式系统。通过分享模式，可建立通用的设计问题词汇表，让日益庞大的模式界成员更有效地识别、指出和讨论问题和解决方案。提高设计系统的速度是使用模式的一大重要原因。

我们无意建立一个完备的模式系统。现有的模式如此之多，不可能将它们都囊括到一本书中，而且随着技术的发展，还可能出现新的模式。我们希望读者根据自己的需求，对这个模式系统进行拓展、修改和定制，可以补充遗漏的模式，忽略不需要的模式，修改现有模式。

无论你对本书的风格和内容有任何看法、意见或改进建议，请务必告诉我们，不要有任何顾虑。我们还欢迎你谈谈使用这些模式的经验。你可以给 John Wiley & Sons 写信，由他们转交给我们，也可以向 patterns@mchp.siemens.de 发送电子邮件。

我们在互联网上对本书大部分模式的初始版本展开了讨论。这样做并非想给本书做免费宣传，也不是要推广这些模式，而旨在支持一种新的出版潮流，即刊印前将内容呈现在相关人员面前，这对各方都有益。我们很喜欢这样的讨论，并对参与讨论的所有人员表示感谢。然而，这并不意味着对本书的讨论已经结束，原来的邮件列表还有效，欢迎读者参与。有关订阅指南，请参阅模式主页，其网址为 http://www.hillside.net/patterns/。

这个网站也是最重要的信息源，涉及模式的方方面面，如已出版和即将出版的图书、模式会议、模式论文等。

组织结构

第 1 章系统地介绍了模式的概念，讨论了描述模式的原则。第 2～4 章介绍了一个模式编目。

*架构模式*是最高层的模式，旨在提供系统架构的整体骨架。第 2 章介绍了 8 个架构模式，它们来自不同的应用领域。

第 3 章介绍了 8 个*设计模式*，确定软件系统的整体结构后，这些模式可用于解决常见的问题，如组织组件以应对复杂性、将工作负荷分配给多个组件以及组织组件间通信。

第 4 章是该模式编目的第三部分，也是最后一部分，探讨了*成例*——语言特定的模式。然而，这一章主要谈论他人的成果，而非记录我们的发现，且只介绍了一个成例。为什么不描述我们发现的一系列成例呢？原因很简单：适用于 C++和 Smalltalk 等语言的成例太多了，因此我们没有重述这些模式，而选择了指出它们的来源。

第 5 章阐述了将模式组织成*模式系统*的重要性。模式作者和使用者都将受益于模式系统，因为它便于查找适合当前情形的模式、填补模式集合中存在的空白、理解模式之间的关系、完善模式系统等。

第 6 章讨论了如何将模式融入软件架构中。具体地说，讨论了我们对模式架构及其基本原则的认识，并演示了模式如何支持这些原则。

第 7 章概述了模式的历史、相关著作和模式界。第 8 章展望了模式的未来发展方向。

最后是几个附录，包括表示法、术语表、参考文献和模式索引。

致谢

这里要感谢的人很多，他们都为本书的出版提供了这样或那样的帮助。我们发自内心地想感谢他们。

感谢 Joelle Coutaz、Wilhelm Gruber、Claus Jäkel、Doug Lea、Oscar Nierstrasz、Laurence Nigay、Frances Paulisch、Wolfgang Pree、Uwe Steinmüller、John Vlissides 和 Walter Zimmer，感谢他们讨论并审校了本书的早期版本。感谢伊利诺伊大学厄巴纳—香槟分校的 Ralph Johnson 及其架构读书小组的成员 John Brant、Michael Chung、Brian Foote、Don Roberts 和 Joseph Yoder，感谢他们一丝不苟地审阅本书大部分模式描述，并提出了很多宝贵意见和改进建议。我们还要感谢 Hillside Group 给予的支持和鼓励。

感谢所有帮助改进各个模式的人员。描述每个模式结束时，我们会专辟一节列出要感谢的人。

特别感谢 James Coplien、Joseph Davison、Neil Harrison 和 Douglas Schmidt 详细审阅了全部内容，这有助于本书的最终定稿和润色。

暑期学生 Marina Seidl 和 Martin Botzler 与我们一起完成了一些早期的艰难实验。还要特别感谢德国慕尼黑西门子研究院软件工程实验室的 Franz Kapsner 和 Hartmut Raffler，感谢他们在管理方面给予的帮助和支持。

Francis Glassborow 和 Steve Rickaby 帮助我们提高了英语写作能力，并消除了最糟糕的“德式英语”——这可不轻松。

最后，感谢本书的编辑 Gaynor Redvers-Mutton 以及 John Wiley & Sons 出版社的全体同仁，是他们让本书得以在如此短的时间内出版。

导　读

阅读顺序

本书的组织方式适合从头到尾地阅读，如果你要自己决定阅读顺序，请参阅下面的提示。

第 1 章深入阐释了软件架构模式，为阅读后续各章奠定了基础，应首先阅读。以什么样的顺序阅读各个模式由你决定。要掌握模式背后的重要理念，只需阅读“背景”、“问题”和“解决方案”部分。大量的交叉引用将帮助你了解模式之间的关系。

建议模式方面的新手先阅读基本而简单的模式，即那些易于理解、出现在大量结构良好的软件系统中的模式，如：

- 架构模式 Pipes and Filters（参见 2.2.2 节）；
- 设计模式 Proxy（参见 3.4 节）；
- 设计模式 Forwarder-Receiver（参见 3.6.1 节）。

遇到设计方面的问题可利用本书寻找解决方案：以第 5 章的模式系统概述为指南，挑选出可能提供了解决方案的模式，再阅读它们的详细描述。

其他各章（第 6 ~ 8 章）可按任何顺序阅读，但对大多数读者来说，按本书给定的顺序阅读最合适。

本书主要模式及其实践功能

结构分解

设计模式 Whole-Part（参见 3.2 节）有助于将组件聚合成语义整体。

工作组织

设计模式 Master-Slave（参见 3.3 节）有助于改善容错性以及计算的并行性和准确度。一个

主组件将工作分配给多个相同的从组件，并根据这些从组件返回的结果计算最终结果。

访问控制

设计模式 Proxy（参见 3.4 节）让客户端与代表而非组件本身通信。引入这样的代理可达成很多目的，如提高效率，简化访问以及禁止未经授权的访问。

管理

设计模式 Command Processor（参见 3.5.1 节）将服务的请求和执行分开，并提供额外的服务，如存储请求对象以便以后能够撤销请求。

设计模式 View Handler（参见 3.5.2 节）有助于管理软件系统提供的所有视图、协调视图间依赖关系以及统筹视图更新。

在 C++中，成例[Cope92] Counted Pointer（参见 4.4 节）简化了动态分配的共享对象的内存管理。

通信

设计模式 Forwarder-Receiver（参见 3.6.1 节）利用对等交互模型让软件系统能够透明地进行进程间通信。它通过引入转发者和接收者将对等体与底层通信机制解耦。

设计模式 Client-Dispatcher-Server（参见 3.6.2 节）在客户端和服务器之间添加了一个中间层——分派器组件。它利用名称服务提供了位置透明性，并隐藏了在客户端和服务器之间建立通信连接的细节。

设计模式 Publisher-Subscriber（参见 3.6.3 节）有助于让相互协作的组件的状态保持同步。

从混沌到有序

架构模式 Layers（参见 2.2.1 节）有助于将应用程序划分为多组子任务，其中每组子任务都位于特定抽象层。

架构模式 Pipes and Filters（参见 2.2.2 节）提供的结构适合处理数据流的系统。

架构模式 Blackboard（参见 2.2.3 节）对还未找到明确解决策略的问题很有帮助。多个专业子系统通过集思广议，获得可能的部分解或近似解。

分布式系统

架构模式 Broker（参见 2.3 节）可用于设计这样的分布式软件系统，即包含通过远程服务调用交互的组件。

交互式系统

MVC 架构模式 Model-View-Controller（参见 2.4.1 节）将交互式应用程序划分为三种组件：核心功能组件、表示组件和控制组件，并使用变更传播机制确保这三部分一致。

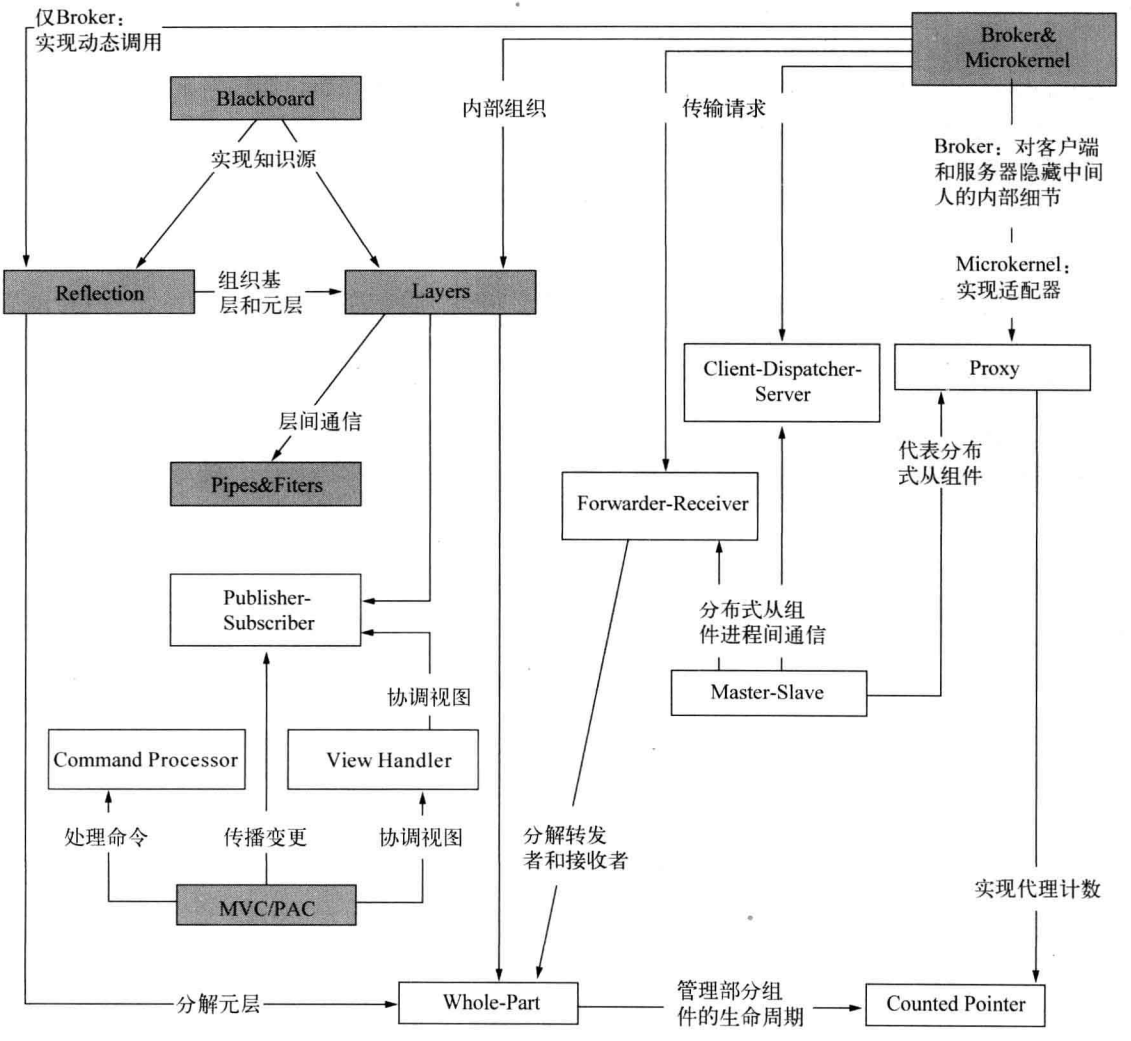
PAC 架构模式 Presentation-Abstraction-Control（参见 2.4.2 节）定义了一种适用于交互式软件系统的结构：由相互协作的智能体组成的层次结构。每个智能体都负责应用程序功能的特定方面，并包含三个组件：表示组件、抽象组件和控制组件，这将智能体的人机交互方面同功能核心和通信方面分离了。

可适应系统

架构模式 Microkernel（参见 2.5.1 节）适用于必须能够适应需求不断变化的系统，它将最基本的功能核心与扩展的功能和随客户而异的部分分离。微核还充当插座，用于插入扩展及协调它们之间的协作。

架构模式 Reflection（参见 2.5.2 节）提供了一种动态修改软件系统的结构和行为的机制。它支持对基本方面（如类型结构和函数调用机制）进行修改。

主要模式功能示意图


仅Broker：
实现动态调用
Broker&
Microkernel
Blackboard
内部组织
传输请求
实现知识源
Broker：对客户端
和服务器隐藏中间
人的内部细节
Microkernel：
实现适配器
Reflection
组织基
层和元层
Layers
Client-Dispatcher-
Server
Proxy
层间通信
代表分布
式从组件
Pipes&Fiters
Forwarder-Receiver
Publisher-
Subscriber
分布式从组
件进程间通信
协调视图
Master-Slave
Command Processor
View Handler
处理命令
传播变更
协调视图
分解转发
者和接收者
实现代理计数
MVC/PAC
分解元层
Whole-Part
管理部分组
件的生命周期
Counted Pointer

目　录

第 1 章 模　式

很久很久以前，一组随机排列的原子正飘过虚无的太空，它们因受到重创以不同寻常的模式组合在一起。这些组合模式很快学会了自我复制（这正是它们不同寻常之处）并继续飘移，给它们经过的每个星球都带来了巨大的麻烦。这就是宇宙中生命的起源。

——道格拉斯·亚当斯，《银河系漫游指南》

模式有助于你利用训练有素的软件工程师的集体经验，它们记录了软件开发领域已得到充分证明的既有经验，可帮助推广良好的设计实践。每个模式都阐述了一个在软件系统设计和实现过程中反复出现的问题。利用模式可打造出具有特定特征的软件架构。

软件架构模式是什么呢？模式对软件开发有何帮助呢？本章将深入阐述这两点。

1.1　模式是什么

面对特定问题时，专家很少去寻找与既有解决方案截然不同的新方案，而通常会想起一个以前解决过的类似问题，并将其解决方案的精髓用于解决这个新问题。在建筑[Ale79]、经济学[Etz64]和软件工程 [BJ94] 等众多领域，这种“专家行为”（即“问题—解决方案”的思考方式）已司空见惯。这是一种自然而然的方式，可用于应对任何问题或社交场合[NS72]。

对于这样的问题—解决方案，下面是一个绝佳而直观的例子，它来自建筑领域。

示例　窗户位置[AIS77]

大家都喜欢靠窗户的座位，喜欢在巨大飘窗和低矮窗台前的舒适座椅上落座。在没有这种位置的房间中，你很难感觉舒适或完全放松。

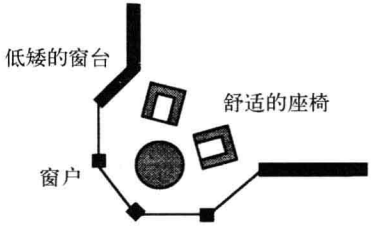
如果房间没有这样的窗户，人就会受到两种作用力的折磨：

(1) 既想舒适地坐下来；

(2) 又想待在光线充足的地方。

显然，如果舒适的地方（房间内你最想落座的地方）远离窗户，这种矛盾就无法解决。

因此，对于你白天需要长时间待在其中的每个房间，至少得有一个可在旁边舒适落座的窗户。



从特定问题—解决方案中提炼出通用的因素便可得到模式：这些问题—解决方案通常是一系列熟悉的问题和解决方案，其中每对问题—解决方案都呈现出相同的模式[Joh94]。在著作《建筑的永恒之道》中，建筑师Christopher Alexander对模式做了如下定义[Ale79]（247页）。

> 每个模式都是一条由三部分组成的规则，诠释了特定背景、问题和解决方案之间的关系。
>
> 作为现实世界的一个元素，模式阐述了特定背景、该背景下反复出现的一系列作用力以及消解这些作用力的空间配置。
>
> 作为一个语言元素，模式提供了指南，指导如何在相关背景下反复利用这种空间配置，以消解一系列给定的作用力。
>
> 简而言之，模式既是现实世界中的一件作品，又是如何及何时创作该作品的规则。模式既是流程又是作品：既描述了一件具有生命力的作品，又阐述了该作品的创作流程。

我们发现，存在众多的软件架构模式。这些模式是软件工程专家依靠实践经验摸索出来的，并被用来开发具有特定特征的应用程序。软件工程专家利用模式有效而妥善地解决设计问题。详细讨论这一点之前，先来看一个著名的例子。

示例　Model-View-Controller模式（参见2.4.1节）

来看看开发带人机界面的软件时如何利用这个模式。

用户界面需求容易变化。例如，添加应用程序功能时，必须修改菜单以便能够访问新功能，还可能需要针对特定客户调整用户界面。系统可能需要移植到另一个平台，而该平台采用的“外观”（feel and look）标准完全不同。即便是升级到新的窗口系统版本，也可能需要修改代码。总之，如果系统的使用寿命很长，可能经常需要修改用户界面。

设计灵活的系统时，让用户界面与功能核心紧密地交织在一起将付出高昂的代价，且容易出错。这样做的后果是，可能需要开发和维护多个大不相同的软件系统——每种用户界面实现一个，