经 典 原 版 书 库

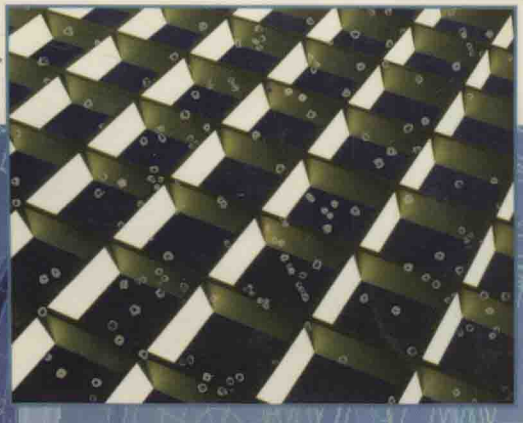
设计模式精解

（英文版 · 第2版）

DESIGN PATTERNS EXPLAINED

A New Perspective on Object-Oriented Design

SECOND EDITION



ALAN SHALLOWAY
JAMES R. TROTT

（美）Alan Shalloway
James R. Trott 著

机械工业出版社
China Machine Press

设计模式精解

（英文版 · 第2版）

Design Patterns Explained

(Second Edition)

A New Perspective on Object-Oriented Design

本书的最大特点之一是作者采用类比而不是编程实例的方式将概念解释得非常清楚。我正在做一套关于OOP和软件开发的音频产品，这种讲述概念的方式给予我很大的启发。

——Bruce Eckel

希望那些仅基本了解面向对象编程和设计的读者，在完全接触设计模式之前，能够发现这本有用的书。本书是对现有的设计模式教材的补充，并可以在入门级教材（如《UML Distilled》）和更高级模式著作之间充当一个很好的衔接。

——James Noble

本书是模式领域最简洁、最清晰、最实用的著作，阐述了模式如何使整个开发过程变得更加容易，解释了面向对象设计的关键原则，以及各种特定模式的概念和优势。通过采用许多最新的Java示例，本书精确地向程序员和架构师展示出如何使用模式来更有效地设计、开发和交付软件。通过分析Java示例，本书提示了为什么、为什么不以及如何应用模式，而且解释了模式的实现。

以畅销的第1版为基础，作者对本版进行了彻底更新，以反映新的软件设计趋势、模式和实现技术。根据广大读者的反馈，作者在第2版中加深了全书概念的阐述，并重新组织了全书内容，使其更易于理解。本书首先概述了模式的基础知识，以及面向对象分析和设计在当代软件开发中的重要性。随后，使用易懂的示例代码阐明了许多当今最有用的模式，包括它们的基础概念、优点、权衡取舍、实现技术以及需要避免的缺陷。另外，许多模式都附有UML图。

本书假定读者没有模式方面的经验，因此是学习模式的理想的第一本书，对于GoF的经典名著《设计模式》，本书也是一个很好的补充。本书适用于学习面向对象设计和设计模式的学生、程序员以及从事软件开发的人士。

第2版的新增和修订内容

- 开始“用模式的方法思考”的更好方式。
- 使用极限编程和其他方法，设计模式如何使敏捷开发更加便利。
- 如何使用共同性和可变性分析来设计应用程序架构。
- 在模式驱动的开发过程中进行测试的关键作用。
- 如何使用工厂来更有效地例示和管理对象。
- 对象池（Object-Pool）模式——一种未被GoF标识的新模式。
- 每章最后新增思考题/练习题。

作者简介

Alan Shalloway Net Objectives（一家从事面向对象业务咨询/培训的公司）的创始人、CEO和首席顾问，具有20多年的从业经验，并经常受邀在重要的软件开发会议（包括SD Expro、Java One、OOP和OOPSLA）上担任演讲人。他拥有麻省理工学院计算机科学硕士学位。

James R. Trott 目前是美国西北太平洋地区一家大型金融机构的高级顾问。20多年来，他使用面向对象和基于模式的分析技术在知识管理和知识工程领域积累了丰富的经验。他拥有应用数学科学硕士、工商管理硕士和跨文化研究文科硕士学位。

PEARSON Addison Wesley

www.PearsonEd.com

上架指导：计算机/软件工程

ISBN 7-111-17569-7

9 787111 175698

封面设计：吴 刚

HZ BOOKS 华章图书

PEARSON Education

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

投稿热线：(010) 88379604

购书热线：(010) 68995259, 68995264

读者信箱：hzjsj@hzbook.com

ISBN 7-111-17569-7/TP · 4495

定价：55.00元

HZ BOOKS

PEARSON
Addison Wesley

设计模式精解

Design Patterns Explained

A New Perspective on Object-Oriented Design (Second Edition)

(美) Alan Shalloway
James R. Trott 著

英文版
第2版

业出版社
China Press

经 典 原 版 书 库

设计模式精解

（英文版 · 第2版）

Design Patterns Explained

A New Perspective on Object-Oriented Design

(Second Edition)

(美) Alan Shalloway
James R. Trott 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2005-4840

图书在版编目（CIP）数据

设计模式精解（英文版·第2版）/（美）沙洛韦（Shalloway, A.）等著. -北京：机械工业出版社，2006.1

（经典原版书库）

书名原文：Design Patterns Explained: A New Perspective on Object-Oriented Design, Second Edition

ISBN 7-111-17569-7

Ⅰ.设…　Ⅱ.沙…　Ⅲ.面向对象语言-程序设计-英文　Ⅳ.TP312

中国版本图书馆CIP数据核字（2005）第119024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：迟振春

北京诚信伟业印刷有限公司印刷·新华书店北京发行所发行

2006年1月第1版第1次印刷

718mm × 1020mm 1/16·29.25印张

印数：0 001-3 000册

定价：55.00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

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权

威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

To Leigh, Bryan, Lisa, Michael, and Steven
for their love, support,
encouragement, and sacrifice.

—Alan Shalloway

To Jill, Erika, Lorien, Mikaela, and Geneva,
the roses in the garden of my life.
sola gloria Dei

—James R. Trott

Preface

Should You Buy the Second Edition If You Already Own the First?

The answer, of course, is yes! Here's why.

Since the first edition was written, we have learned so much more about design patterns, including the following:

- How to use commonality and variability analysis to design application architectures
- How design patterns relate to and actually facilitate eXtreme programming (XP) and agile development.
- How testing is a first principle of quality coding.
- Why the use of factories to instantiate and manage objects is critical
- Which set of patterns are essential for students to help them learn how to think in patterns

This book covers all of these topics. We have deepened and clarified what we had before and have added some new content that you will find very helpful, including the following:

- Chapter 15: Commonality and Variability Analysis
- Chapter 20: Lessons from Design Patterns: Factories
- Chapter 21: The Object-Pool Pattern (a pattern not covered by the Gang of Four)
- Chapter 22: Factories Summarized

We have changed the order in which we present some of the patterns. This sequence is more helpful for the students in our courses as they learn the ideas behind patterns.

We have touched every chapter, incorporating the feedback we have received from our many readers over these past three years.

And, to help students, we have created study questions for each chapter (with answers on the book's companion Web site).

We can honestly say this is one of the few second editions that is definitely worth buying — even if you have the first one.

We would love to hear what you think.

—Alan and Jim

Design patterns and object-oriented programming. They hold such promise to make your life as a software designer and developer easier. Their terminology is bandied about every day in the technical and even the popular press. It can be hard to learn them, however, to become proficient with them, to understand what is really going on.

Perhaps you have been using an object-oriented or object-based language for years. Have you learned that the true power of objects is not inheritance, but is in "encapsulating behaviors"? Perhaps you are curious about design patterns and have found the literature a bit too esoteric and high-falutin. If so, this book is for you.

It is based on years of teaching this material to software developers, both experienced and new to object orientation. It is based upon the belief—and our experience—that when you understand the basic principles and motivations that underlie these concepts, why they are doing what they do, your learning curve will be incredibly shorter. And in our discussion of design patterns, you will understand the

true mindset of object orientation, which is a necessity before you can become proficient.

As you read this book, you will gain a solid understanding of 12 core design patterns and a pattern used in analysis. You will learn that design patterns do not exist in isolation, but work in concert with other design patterns to help you create more robust applications. You will gain enough of a foundation that you will be able to read the design pattern literature, if you want to, and possibly discover patterns on your own. Most importantly, you will be better equipped to create flexible and complete software that is easier to maintain.

Although the 12 patterns we teach here are not all of the patterns you should learn, an understanding of these will enable you to learn the others on your own more easily. Instead of giving you more patterns than you need to get started, we have included pattern-related issues that will be more useful.

From Object Orientation to Patterns to True Object Orientation

In many ways, this book is a retelling of my personal experience learning design patterns. This started with learning the patterns themselves and then learning the principles behind them. I expanded this understanding into the realms of analysis and testing as well as learning how patterns relate to agile coding methods. This second edition of this book includes many additional insights I have had since publication of the first edition. Prior to studying design patterns, I considered myself to be reasonably expert in object-oriented analysis and design. My track record had included several fairly impressive designs and implementations in many industries. I knew C++ and was beginning to learn Java. The objects in my code were well-formed and tightly encapsulated. I could design excellent data abstractions for inheritance hierarchies. I thought I knew object orientation.

Now, looking back, I see that I really did not understand the full capabilities of object-oriented design, even though I was doing things the way most experts advised. It wasn't until I began to learn design patterns that my object-oriented design abilities expanded and deepened. Knowing design patterns has made me a better designer, even when I don't use these patterns directly.

I began studying design patterns in 1996. I was a C++/object-oriented design mentor at a large aerospace company in the Northwest. Several people asked me to lead a design pattern study group. That's where I met my coauthor, Jim Trott. In the study group, several interesting things happened. First, I grew fascinated with design patterns. I loved being able to compare my designs with the designs of others who had more experience than I. And second, I discovered that I was not taking full advantage of designing to interfaces and that I didn't always concern myself with seeing whether I could have an object use another object without knowing the used object's type. I also noticed that beginners in object-oriented design—those who would normally be deemed as learning design patterns too early—were benefiting as much from the study group as the experts were. The patterns presented examples of excellent object-oriented designs and illustrated basic object-oriented principles, which helped to mature their designs more quickly. By the end of the study sessions, I was convinced that design patterns were the greatest thing to happen to software design since the invention of object-oriented design.

When I looked at my work at the time, however, I saw that I was not incorporating *any* design patterns into my code. Or, at least, not consciously. Later, after learning patterns, I realized I had incorporated many design patterns into my code just out of being a good coder. However, now that I understand patterns better, I am able to use them better.

I just figured I didn't know enough design patterns yet and needed to learn more. At the time, I only knew about six of them. Then I had an epiphany. I was working as a mentor in object-oriented design

for a project and was asked to create the project's high-level design. The leader of the project was extremely sharp, but was fairly new to object-oriented design.

The problem itself wasn't that difficult, but it required a great deal of attention to make sure the code was going to be easy to maintain. Literally, after about two minutes of looking at the problem, I had developed a design based on my normal approach of data abstraction. Unfortunately, it was also clear to me this was not going to be a good design. Data abstraction alone had failed me. I had to find something better.

Two hours later, after applying every design technique I knew, I was no better off. My design was essentially the same. What was most frustrating was that I knew there was a better design. I just couldn't see it. Ironically, I also knew of four design patterns that "lived" in my problem, but I couldn't see how to use them. Here I was—a supposed expert in object-oriented design—baffled by a simple problem!

Feeling very frustrated, I took a break and started walking down the hall to clear my head, telling myself I would not think of the problem for at least 10 minutes. Well, 30 seconds later, I was thinking about it again! But I had gotten an insight that changed my view of design patterns: rather than using patterns as individual items, I should use the design patterns together.

Patterns are supposed to be sewn together to solve a problem.

I had heard this before, but hadn't really understood it. Because patterns in software have been introduced as *design* patterns, I had always labored under the assumption that they had mostly to do with design. My thoughts were that in the design world, the patterns came as pretty much well-formed relationships between classes. Then I read Christopher Alexander's amazing book, *The Timeless Way of Building* (Oxford University Press, 1979). I learned that patterns existed at all levels—analysis, design, and implementation. Alexander discusses

using patterns to help in the understanding of the problem domain (even in describing it), not just using them to create the design after the problem domain is understood.

My mistake had been in trying to create the classes in my problem domain and then stitch them together to make a final system, a process that Alexander calls a particularly bad idea. I had never asked whether I had the right classes because they just seemed so right, so obvious; they were the classes that immediately came to mind as I started my analysis, the "nouns" in the description of the system that we had been taught to look for. But I had struggled trying to piece them together.

When I stepped back and used design patterns and Alexander's approach to guide me in the creation of my classes, a far superior solution unfolded in only a matter of minutes. It was a good design, and we put it into production. I was excited—excited to have designed a good solution and excited about the power of design patterns. It was then that I started incorporating design patterns into my development work and my teaching.

I began to discover that programmers who were new to object-oriented design could learn design patterns, and in doing so, develop a basic set of object-oriented design skills. It was true for me, and it was true for the students whom I was teaching.

Imagine my surprise! The design pattern books I had been reading and the design pattern experts I had been talking to were saying that you really needed to have a good grounding in object-oriented design before embarking on a study of design patterns. Nevertheless, I saw, with my own eyes, students who learned object-oriented design concurrently with design patterns learned object-oriented design faster than those just studying object-oriented design. They even seemed to learn design patterns at almost the same rate as experienced object-oriented practitioners.

I began to use design patterns as a basis for my teaching. I began to call my classes *Pattern-Oriented Design: Design Patterns from Analysis to Implementation.*

I wanted my students to understand these patterns and began to discover that using an exploratory approach was the best way to foster this understanding. For instance, I found that it was better to present the Bridge pattern by presenting a problem and then have my students try to design a solution to the problem using a few guiding principles and strategies that I had found were present in most of the patterns. In their exploration, the students discovered the solution—essentially the Bridge pattern—and remembered it.

Design Patterns and Agile/XP

The guiding principles and strategies underlying design patterns seem very clear to me now. Certainly, they are stated in the "Gang of Four's" design patterns book, but too succinctly to be of value to me when I first read it. I believe the Gang of Four were writing for the Smalltalk community, which was very grounded in these principles and therefore needed little background. It took me a long time to understand them because of limitations in my own understanding of the object-oriented paradigm. It was only after integrating in my own mind the work of the Gang of Four with Alexander's work, Jim Coplien's work on commonality and variability analysis, and Martin Fowler's work in methodologies and analysis patterns that these principles became clear enough to me so that I was able to talk about them to others. It helped that I was making my livelihood explaining things to others—so I couldn't get away with making assumptions as easily as I could when I was just doing things for myself.

Since the first edition of this book appeared, I have been doing a considerable amount of agile development and have become very grounded in eXtreme Programming (XP) coding practices, test-driven development (TDD), and Scrum. Initially, I had a difficult

time reconciling design patterns with XP and TDD. However, I quickly realized that both have great value and both are grounded in the same principles (although they take different design approaches). In fact, in our agile software development boot camps, we make it clear that design patterns, used properly, are strong enablers of agile development.

Throughout this book, I discuss many of the ways design patterns relate to agile management and coding practices. If you are unfamiliar with XP, TDD, or Scrum, do not be too concerned about these comments. However, if this is the case, I suggest the next book you read be about one of these topics.

In any event, I found that these guiding principles and strategies could be used to "derive" several of the design patterns. By "derive a design pattern," I mean that if I looked at a problem that might be solved by a design pattern, I could use the guiding principles and strategies that I learned from patterns to come up with the solution expressed in a pattern. I made it clear to my students that we weren't really coming up with design patterns this way. Instead, I was just illustrating one possible thought process that the people who came up with the original solutions, those that were eventually classified as design patterns, might have used.

My abilities to explain these few, but powerful, principles and strategies improved. As they did, I found that it became more useful to explain an increasing number of the Gang of Four patterns. In fact, I use these principles and strategies to explain virtually all the patterns I discuss in my design patterns course.

I found that I was using these principles in my own designs both with and without patterns. This didn't surprise me. If using these strategies resulted in a design equivalent to a design pattern when I knew the pattern was present, that meant they were giving me a way to derive excellent designs (because patterns are excellent designs by