

<<研磨设计模式>>

图书基本信息

书名：<<研磨设计模式>>

13位ISBN编号：9787302239239

10位ISBN编号：7302239231

出版时间：2010-11

出版时间：清华大学出版社

作者：陈臣,王斌

页数：783

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

<<研磨设计模式>>

前言

创作背景软件开发越来越复杂，对软件设计的要求也越来越高，而软件设计和架构的入门功夫就是深入理解和掌握设计模式。

因此，设计模式的重要性不言而喻。

很多朋友认识到了设计模式的重要性，也看了很多的书籍和资料，但是，常听到这样的抱怨：“设计模式的书我看了不少，觉得都看懂了，就是不知道在实际开发中怎么运用这些设计模式”，从而认为设计模式是“看上去很美的花拳绣腿”。

其实不然，造成这种情况的原因就在于：这些朋友对设计模式的理解不到位，自己感觉懂了，其实还差很远，并没有“真正”理解和掌握设计模式。

市面上有不少设计模式方面的书籍，但对一般的学习者而言，要么是太深，看得云里雾里的，比如GoF的著作《设计模式——可复用面向对象软件的基础》，很经典，但是能吃透的人少；要么就是太浅，看了跟没看差不多，也就是介绍一下每个设计模式，告诉你这就是某某设计模式，虽然语言很生动但是实在没货，看完也不知道怎么运用，就像是带领大家摸到了设计模式的大门口，却不告诉你怎么进去一样，其根本原因还是讲得太浅，跟实际的应用有太大的差距。

对于所有想要真正理解和掌握设计模式的朋友，其实需要这样的书籍：理论全面、准确，难度适中；讲解深入浅出、浅显易懂；理论联系实际应用，对于晦涩的理论，应有相应的示例；示例最好来自实际应用，而不是来自虚拟的场景；示例最好相对完整，而不是片段代码，以利于学习和应用。

这也是本书写作的目的，希望能够帮助更多的朋友早日修成设计模式的正果。

经过多年的准备和一年的写作，以及各层次读者的多轮试读意见和建议汇总，最终成书，我们可以这样说：这是一本诚意十足的书，敬请您的评鉴！

本书的试读人员包括：从还没有参加工作的学生，一直到工作7年的人员；职务覆盖普通的程序员、项目经理、高级系统架构师、技术部的经理；两位作者本身从事开发工作的年限，一位超过10年，一位超过5年。

<<研磨设计模式>>

内容概要

本书完整覆盖GoF讲述的23个设计模式并加以细细研磨。

初级内容从基本讲起，包括每个模式的定义、功能、思路、结构、基本实现、运行调用顺序、基本应用示例等，让读者能系统、完整、准确地掌握每个模式，培养正确的“设计观”；中高级内容则深入探讨如何理解这些模式，包括模式中蕴涵什么样的设计思想，模式的本质是什么，模式如何结合实际应用，模式的优缺点以及与其他模式的关系等，以期让读者尽量去理解和掌握每个设计模式的精髓所在。

本书在内容上深入、技术上实用、和实际开发结合程度很高，书中大部分的示例程序都是从实际项目中简化而来，因此很多例子都可以直接拿到实际项目中使用。

如果你想要深入透彻地理解和掌握设计模式，并期望能真正把设计模式应用到项目中去，那么这是你不可错过的一本好书。

本书难度为初级到中级，适合于所有开发人员、设计人员或者即将成为开发人员的朋友。也可以作为高校学生深入学习设计模式的参考读物。

<<研磨设计模式>>

作者简介

陈臣：十年Java/JavaEE开发经验，高级系统架构师，功力深厚，技术精湛，精通Java/JavaEE相关技术和多种开源框架，尤其擅长系统分析和架构设计。

从事过专业的中间件研发，包括基于组件的Web页面框架、基于WFMC的工作流中间件、类似于Hibernate的ORM框架等等；参与或主持了多个中大型的企业级应用项目，拥有多年项目经理、技术部经理的管理经验。

个人博客：<http://www.javass.cn/javapeixunxyd/index.html>

王斌：从事Java/JavaEE开发五年，系统架构师，精通Ejb、Struts、Spring、Hibernate、iBatis等框架技术，擅长设计模式和Eclipse插件开发。

作为架构小组骨干，参与了国旅电子商务平台、南王酒庄等多个项目的开发，开发并维护有constance4j、myxstream、SimpleMapping等多个公司内部开源框架，深得多个项目组好评。

<<研磨设计模式>>

书籍目录

第1章 设计模式基础

- 1.1 设计模式是什么
 - 1.1.1 什么是模式
 - 1.1.2 设计模式的概念
 - 1.1.3 设计模式的理解
 - 1.1.4 设计模式的历史
- 1.2 设计模式有什么
 - 1.2.1 设计模式的组成
 - 1.2.2 设计模式的分类
- 1.3 设计模式的学习
 - 1.3.1 为什么要学习设计模式
 - 1.3.2 学习设计模式的层次
 - 1.3.3 如何学习设计模式
- 1.4 本书的组织方式
 - 1.4.1 本书所讲述的设计模式的提纲
 - 1.4.2 每个模式的讲述结构

第2章 简单工厂

- 2.1 场景问题
 - 2.1.1 接口回顾
 - 2.1.2 面向接口编程
 - 2.1.3 不用模式的解决方案
 - 2.1.4 有何问题
- 2.2 解决方案
 - 2.2.1 使用简单工厂来解决问题
 - 2.2.2 简单工厂的结构和说明
 - 2.2.3 简单工厂示例代码
 - 2.2.4 使用简单工厂重写示例
- 2.3 模式讲解
 - 2.3.1 典型疑问
 - 2.3.2 认识简单工厂
 - 2.3.3 简单工厂中方法的写法
 - 2.3.4 可配置的简单工厂
 - 2.3.5 简单工厂的优缺点
 - 2.3.6 思考简单工厂
 - 2.3.7 相关模式

第3章 外观模式

- 3.1 场景问题
 - 3.1.1 生活中的示例
 - 3.1.2 代码生成的应用
 - 3.1.3 不用模式的解决方案
 - 3.1.4 有何问题
- 3.2 解决方案
 - 3.2.1 使用外观模式来解决问题
 - 3.2.2 外观模式的结构和说明
 - 3.2.3 外观模式示例代码

<<研磨设计模式>>

3.2.4 使用外观模式重写示例

3.3 模式讲解

3.3.1 认识外观模式

3.3.2 外观模式的实现

3.3.3 外观模式的优缺点

3.3.4 思考外观模式

3.3.5 相关模式

第4章 适配器模式 (Adapter)

4.1 场景问题

4.1.1 装配电脑的例子

4.1.2 同时支持数据库和文件的日志管理

4.1.3 有何问题

4.2 解决方案

4.2.1 使适配器模式来解决问题

4.2.2 模式的结构和说明

4.2.3 适配器模式示例代码

4.2.4 使用适配器模式来实现示例

4.3 模式讲解

4.3.1 认识适配器模式

4.3.2 适配器模式的实现

4.3.3 双向适配器

4.3.4 对象适配器和类适配器

4.3.5 适配器模式的优缺点

4.3.6 思考适配器模式

4.3.7 相关模式

第5章 单例模式 (Singleton)

5.1 场景问题

5.1.1 读取配置文件的内容

5.1.2 不用模式的解决方案

5.1.3 有何问题

5.2 解决方案

5.2.1 使用单例模式来解决问题

5.2.2 单例模式的结构和说明

5.2.3 单例模式示例代码

5.2.4 使用单例模式重写示例

5.3 模式讲解

5.3.1 认识单例模式

5.3.2 懒汉式和饿汉式实现

5.3.3 延迟加载的思想

5.3.4 缓存的思想

5.3.5 Java中缓存的基本实现

5.3.6 利用缓存来实现单例模式

5.3.7 单例模式的优缺点

5.3.8 在Java中一种更好的单例实现方式

5.3.9 单例和枚举

5.3.10 思考单例模式

5.3.11 相关模式

<<研磨设计模式>>

第6章 工厂方法模式 (Factory Method)

6.1 场景问题

6.1.1 导出数据的应用框架

6.1.2 框架的基础知识

6.1.3 有何问题

6.2 解决方案

6.2.1 使用工厂方法模式来解决问题

6.2.2 工厂方法模式的结构和说明

6.2.3 工厂方法模式示例代码

6.2.4 使用工厂方法模式来实现示例

6.3 模式讲解

6.3.1 认识工厂方法模式

6.3.2 工厂方法模式与IoC/DI

6.3.3 平行的类层次结构

6.3.4 参数化工厂方法

6.3.5 工厂方法模式的优缺点

6.3.6 思考工厂方法模式

6.3.7 相关模式

第7章 抽象工厂模式 (Abstract Factory)

7.1 场景问题

7.1.1 选择组装电脑的配件

7.1.2 不用模式的解决方案

7.1.3 有何问题

7.2 解决方案

7.2.1 使用抽象工厂模式来解决问题

7.2.2 抽象工厂模式的结构和说明

7.2.3 抽象工厂模式示例代码

7.2.4 使用抽象工厂模式重写示例

7.3 模式讲解

7.3.1 认识抽象工厂模式

7.3.2 定义可扩展的工厂

7.3.3 抽象工厂模式和DAO

7.3.4 抽象工厂模式的优缺点

7.3.5 思考抽象工厂模式

7.3.6 相关模式

第8章 生成器模式 (Builder)

8.1 场景问题

8.1.1 继续导出数据的应用框架

8.1.2 不用模式的解决方案

8.1.3 有何问题

8.2 解决方案

8.2.1 使用生成器模式来解决问题

8.2.2 生成器模式的结构和说明

8.2.3 生成器模式示例代码

8.2.4 使用生成器模式重写示例

8.3 模式讲解

8.3.1 认识生成器模式

<<研磨设计模式>>

- 8.3.2 生成器模式的实现
- 8.3.3 使用生成器模式构建复杂对象
- 8.3.4 生成器模式的优点
- 8.3.5 思考生成器模式
- 8.3.6 相关模式

第9章 原型模式 (Prototype)

- 9.1 场景问题
 - 9.1.1 订单处理系统
 - 9.1.2 不用模式的解决方案
 - 9.1.3 有何问题
- 9.2 解决方案
 - 9.2.1 使用原型模式来解决问题
 - 9.2.2 原形模式的结构和说明
 - 9.2.3 原型模式示例代码
 - 9.2.4 使用原型模式重写示例
- 9.3 模式讲解
 - 9.3.1 认识原型模式
 - 9.3.2 Java中的克隆方法
 - 9.3.3 浅度克隆和深度克隆
 - 9.3.4 原型管理器
 - 9.3.5 原型模式的优缺点
 - 9.3.6 思考原型模式
 - 9.3.7 相关模式

第10章 中介者模式 (Mediator)

- 10.1 场景问题
 - 10.1.1 如果没有主板
 - 10.1.2 有何问题
 - 10.1.3 使用电脑来看电影
- 10.2 解决方案
 - 10.2.1 使用中介者模式来解决问题
 - 10.2.2 中介者模式的结构和说明
 - 10.2.3 中介者模式示例代码
 - 10.2.4 使用中介者模式来实现示例
- 10.3 模式讲解
 - 10.3.1 认识中介者模式
 - 10.3.2 广义中介者
 - 10.3.3 中介者模式的优缺点
 - 10.3.4 思考中介者模式
 - 10.3.5 相关模式

第11章 代理模式 (Proxy)

- 11.1 场景问题
 - 11.1.1 访问多条数据
 - 11.1.2 不用模式的解决方案
 - 11.1.3 有何问题
- 11.2 解决方案
 - 11.2.1 使用代理模式来解决问题
 - 11.2.2 代理模式的结构和说明

<<研磨设计模式>>

- 11.2.3 代理模式示例代码
- 11.2.4 使用代理模式重写示例
- 11.3 模式讲解
 - 11.3.1 认识代理模式
 - 11.3.2 保护代理
 - 11.3.3 Java中的代理
 - 11.3.4 代理模式的特点
 - 11.3.5 思考代理模式
 - 11.3.6 相关模式
- 第12章 观察者模式(Observer)
 - 12.1 场景问题
 - 12.1.1 订阅报纸的过程
 - 12.1.2 订阅报纸的问题
 - 12.2 解决方案
 - 12.2.1 使用观察者模式来解决问题
 - 12.2.2 观察者模式的结构和说明
 - 12.2.3 观察者模式示例代码
 - 12.2.4 使用观察者模式实现示例
 - 12.3 模式讲解
 - 12.3.1 认识观察者模式
 - 12.3.2 推模型和拉模型
 - 12.3.3 Java中的观察者模式
 - 12.3.4 观察者模式的优缺点
 - 12.3.5 思考观察者模式
 - 12.3.6 Swing中的观察者模式
 - 12.3.7 简单变形示例——区别对待观察者
 - 12.3.8 相关模式
- 第13章 命令模式(Command)
 - 13.1 场景问题
 - 13.1.1 如何开机
 - 13.1.2 与我何干
 - 13.1.3 有何问题
 - 13.2 解决方案
 - 13.2.1 使用命令模式来解决问题
 - 13.2.2 命令模式的结构和说明
 - 13.2.3 命令模式示例代码
 - 13.2.4 使用命令模式来实现示例
 - 13.3 模式解
 - 13.3.1 认识命令模式
 - 13.3.2 参数化配置
 - 13.3.3 可撤销的操作
 - 13.3.4 宏命令
 - 13.3.5 队列请求
 - 13.3.6 日志请求
 - 13.3.7 命令模式的优点
 - 13.3.8 思考命令模式
 - 13.3.9 退化的命令模式

<<研磨设计模式>>

13.3.10 相关模式

第14章 迭代器模式 (Iterator)

14.1 场景问题

14.1.1 工资表数据的整合

14.1.2 有何问题

14.2 解决方案

14.2.1 使用迭代器模式来解决问题

14.2.2 迭代器模式的结构和说明

14.2.3 迭代器模式示例代码

14.2.4 使用迭代器模式来实现示例

14.3 模式讲解

14.3.1 认识迭代器模式

14.3.2 使用Java的迭代器

14.3.3 带迭代策略的迭代器

14.3.4 双向迭代器

14.3.5 迭代器模式的优点

14.3.6 思考迭代器模式

14.3.7 翻页迭代

14.3.8 相关模式

第15章 组合模式(Composite)

15.1 场景问题

15.1.1 商品类别树

15.1.2 不用模式的解决方案

15.1.3 有何问题

15.2 解决方案

15.2.1 使用组合模式来解决问题

15.2.2 组合模式的结构和说明

15.2.3 组合模式示例代码

15.2.4 使用组合模式重写示例

15.3 模式讲解

15.3.1 认识组合模式

15.3.2 安全性和透明性

15.3.3 父组件引用

15.3.4 环状引用

15.3.5 组合模式的优缺点

15.3.6 思考组合模式

15.3.7 相关模式

第16章 模板方法模式(Template Method)

16.1 场景问题

16.1.1 登录控制

16.1.2 不用模式的解决方案

16.1.3 有何问题

16.2 解决方案

16.2.1 使用模板方法模式来解决问题

16.2.2 模板方法模式的结构和说明

16.2.3 模板方法模式示例代码

16.2.4 使用模板方法模式重写示例

<<研磨设计模式>>

16.3 模式讲解

- 16.3.1 认识模板方法模式
- 16.3.2 模板的写法
- 16.3.3 Java回调与模板方法模式
- 16.3.4 典型应用：排序
- 16.3.5 实现通用的增删改查
- 16.3.6 模板方法模式的优缺点
- 16.3.7 思考模板方法模式
- 16.3.8 相关模式

第17章 策略模式(Strategy)

17.1 场景问题

- 17.1.1 报价管理
- 17.1.2 不用模式的解决方案
- 17.1.3 有何问题

17.2 解决方案

- 17.2.1 使用策略模式来解决问题
- 17.2.2 策略模式的结构和说明
- 17.2.3 策略模式示例代码
- 17.2.4 使用策略模式重写示例

17.3 模式讲解

- 17.3.1 认识策略模式
- 17.3.2 Context和Strategy的关系
- 17.3.3 容错恢复机制
- 17.3.4 策略模式结合模板方法模式
- 17.3.5 策略模式的优缺点
- 17.3.6 思考策略模式
- 17.3.7 相关模式

第18章 状态模式 (State)

18.1 场景问题

- 18.1.1 实现在线投票
- 18.1.2 不用模式的解决方案
- 18.1.3 有何问题

18.2 解决方案

- 18.2.1 使用状态模式来解决问题
- 18.2.2 状态模式的结构和说明
- 18.2.3 状态模式示例代码
- 18.2.4 使用状态模式重写示例

18.3 模式讲解

- 18.3.1 认识状态模式
- 18.3.2 状态的维护和转换控制
- 18.3.3 使用数据库来维护状态
- 18.3.4 模拟 workflow
- 18.3.5 状态模式的优缺点
- 18.3.6 思考状态模式
- 18.3.7 相关模式

第19章 备忘录模式 (Memento)

19.1 场景问题

<<研磨设计模式>>

- 19.1.1 开发仿真系统
- 19.1.2 不用模式的解决方案
- 19.1.3 有何问题
- 19.2 解决方案
 - 19.2.1 使用备忘录模式来解决问题
 - 19.2.2 备忘录模式的结构和说明
 - 19.2.3 备忘录模式示例代码
 - 19.2.4 使用备忘录模式重写示例
- 19.3 模式讲解
 - 19.3.1 认识备忘录模式
 - 19.3.2 结合原型模式
 - 19.3.3 离线存储
 - 19.3.4 再次实现可撤销操作
 - 19.3.5 备忘录模式的优缺点
 - 19.3.6 思考备忘录模式
 - 19.3.7 相关模式
- 第20章 享元模式 (Flyweight)
 - 20.1 场景问题
 - 20.1.1 加入权限控制
 - 20.1.2 不使用模式的解决方案
 - 20.1.3 有何问题
 - 20.2 解决方案
 - 20.2.1 使用享元模式来解决问题
 - 20.2.2 享元模式的结构和说明
 - 20.2.3 享元模式示例代码
 - 20.2.4 使用享元模式重写示例
 - 20.3 模式讲解
 - 20.3.1 认识享元模式
 - 20.3.2 不需要共享的享元实现
 - 20.3.3 对享元对象的管理
 - 20.3.4 享元模式的优缺点
 - 20.3.5 思考享元模式
 - 20.3.6 相关模式
- 第21章 解释器模式 (Interpreter)
 - 21.1 场景问题
 - 21.1.1 读取配置文件
 - 21.1.2 不用模式的解决方案
 - 21.1.3 有何问题
 - 21.2 解决方案
 - 21.2.1 使用解释器模式来解决问题
 - 21.2.2 解释器模式的结构和说明
 - 21.2.3 解释器模式示例代码
 - 21.2.4 使用解释器模式重写示例
 - 21.3 模式讲解
 - 21.3.1 认识解释器模式
 - 21.3.2 读取多个元素或属性的值
 - 21.3.3 解析器

<<研磨设计模式>>

21.3.4 解释器模式的优缺点

21.3.5 思考解释器模式

21.3.6 相关模式

第22章 装饰模式 (Decorator)

22.1 场景问题

22.1.1 复杂的奖金计算

22.1.2 简化后的奖金计算体系

22.1.3 不用模式的解决方案

22.1.4 有何问题

22.2 解决方案

22.2.1 使用装饰模式来解决问题

22.2.2 装饰模式的结构和说明

22.2.3 装饰模式示例代码

22.2.4 使用装饰模式重写示例

22.3 模式讲解

22.3.1 认识装饰模式

22.3.2 Java中的装饰模式应用

22.3.3 装饰模式和AOP

22.3.4 装饰模式的优缺点

22.3.5 思考装饰模式

22.3.6 相关模式

第23章 职责链模式 (Chain of Responsibility)

23.1 场景问题

23.1.1 申请聚餐费用

23.1.2 不用模式的解决方案

23.1.3 有何问题

23.2 解决方案

23.2.1 使用职责链模式来解决问题

23.2.2 职责链模式的结构和说明

23.2.3 职责链模式示例代码

23.2.4 使用职责链模式重写示例

23.3 模式讲解

23.3.1 认识职责链模式

23.3.2 处理多种请求

23.3.3 功能链

23.3.4 职责链模式的优缺点

23.3.5 思考职责链模式

23.3.6 相关模式

第24章 桥接模式 (Bridge)

24.1 场景问

24.1.1 发送提示消息

24.1.2 不用模式的解决方案

24.1.3 有何问题

24.2 解决方案

24.2.1 使用桥接模式来解决问题

24.2.2 桥接模式的结构和说明

24.2.3 桥接模式示例代码

<<研磨设计模式>>

24.2.4 使用桥接模式重写示例

24.3 模式讲解

24.3.1 认识桥接模式

24.3.2 谁来桥接

24.3.3 典型例子——JDBC

24.3.4 广义桥接——Java中无处不桥接

24.3.5 桥接模式的优缺点

24.3.6 思考桥接模式

24.3.7 相关模式

第25章 访问者模式 (Visitor)

25.1 场景问题

25.1.1 扩展客户管理的功能

25.1.2 不用模式的解决方案

25.1.3 有何问题

25.2 解决方案

25.2.1 使用访问者模式来解决问题

25.2.2 访问者模式的结构和说明

25.2.3 访问者模式示例代码

25.2.4 使用访问者模式重写示例

25.3 模式讲解

25.3.1 认识访问者模式

25.3.2 操作组合对象结构

25.3.3 谁负责遍历所有元素对象

25.3.4 访问者模式的优缺点

25.3.5 思考访问者模式

25.3.6 相关模式

附录A 常见面向对象设计原则

A.1 设计模式和设计原则

A.1.1 设计模式和设计原则的关系

A.1.2 为何不重点讲解设计原则

A.2 常见的面向对象设计原则

A.2.1 单一职责原则SRP (Single Responsibility Principle)

A.2.2 开放-关闭原则OCP (Open-Closed Principle)

A.2.3 里氏替换原则LSP (Liskov Substitution Principle)

A.2.4 依赖倒置原则DIP (Dependence Inversion Principle)

A.2.5 接口隔离原则ISP (Interface Segregation Principle)

A.2.6 最少知识原则LKP (Least Knowledge Principle)

A.2.7 其他原则

附录B UML简介

B.1 UML基础

B.1.1 UML是什么

B.1.2 UML历史

B.1.3 UML能干什么

B.1.4 UML有什么

B.2 类图

B.2.1 类图的概念

B.2.2 类图的基本表达

<<研磨设计模式>>

B.2.3 抽象类和接口

B.2.4 关系

B.3 顺序图

B.3.1 顺序图的概念

B.3.2 顺序图的基本表达

临别赠言

不是结束而是新的开始

你该怎么做

参考文献

<<研磨设计模式>>

章节摘录

插图：适配器通过转换调用已有的实现，从而能把已有的实现匹配成需要的接口，使之能满足客户端的需要。

也就是说转换匹配是手段，而复用已有的功能才是目的。

在进行转换匹配的过程中，适配器还可以在转换调用的前后实现一些功能处理，也就是实现智能的适配。

2.何时选用适配器模式建议在以下情况中选用适配器模式。

如果你想要使用一个已经存在的类，但是它的接口不符合你的需求，这种情况可以使用适配器模式，来把已有的实现转换成你需要的接口。

如果你想创建一个可以复用的类，这个类可能和一些不兼容的类一起工作，这种情况可以使用适配器模式，到时候需要什么就适配什么。

如果你想使用一些已经存在的子类，但是不可能对每一个子类都进行适配，这种情况可以选用对象适配器，直接适配这些子类的父类就可以了。

4.3.7 相关模式适配器模式与桥接模式其实这两个模式除了结构略为相似外，功能上完全不同。

适配器模式是把两个或者多个接口的功能进行转换匹配；而桥接模式是让接口和实现部分相分离，以便它们可以相对独立地变化。

<<研磨设计模式>>

媒体关注与评论

高超的大师和蹩脚专家的区别就在于，前者能把复杂的东西讲简单，后者恰恰相反，支持楼主！

——4楼rails201020100-07-19引用 比国内某些所谓的大师写的设计模式书好多了，比国外的大师写的书更容易让人懂，顶博主。

——13楼dayday-up12010-08-04引用 楼主对广义桥接模式的阐述，对我而言，只能用震撼来形容，从来就没有这样想过，但经楼主讲出来，确实又是这样，楼主对模式的理解实在是太深刻了，超赞！

好博文!看到第二篇的时候，基本上就已经说到策略模式的核心点上了。

能从核心扩展到相关应用，扩展到与其他模式的相同与不同，也足以说明了楼主对模式的深刻理解!呵呵，继续期待其他模式。

——4楼po-5342010-09-06引用 对楼主的佩服真是如滔滔江水，能把设计模式写得这么深入浅出、深度广度兼具，在我看过的资料里面，以绝对优势排第一，比市面上很多写模式的书都要好很多。

——8楼Superheizai2010-08-20引用 好强悍的博主，对模式的理解真的是深入，另外那个对流式输出的分析，看java源代码来的吧，真是透彻。

——18楼leveret2010-08-20引用 由criblogs追到这里，楼主才是真正的人才，能把模式说的这么清楚的人的确不多！

确实不像现在很多都是浅尝辄止的设计模式的韦，研磨系列，更深入，更具体!估计楼主这么长时间没更新，肯定受出版社邀约了吧，呵呵。

——6楼superheizai2010-09-06引用 很符合中国人的理解方式，所以觉得很亲近，看起来舒服很多！

——21楼bugrluke2010-09-28引用 膜拜大师的作品，头些天粗略的看了一下《大话设计模式》和《重构》，希望10年后有大师今天的成就。

支持出书！

——12楼EnterLee2010-08-11引用 楼主对模式的理解之深刻，表述之深入浅出，引人入胜，实在是让人佩服。

——7楼dakaiopen2010-09-02引用

<<研磨设计模式>>

编辑推荐

《研磨设计模式》：一本值得反复研读的书.....

<<研磨设计模式>>

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>