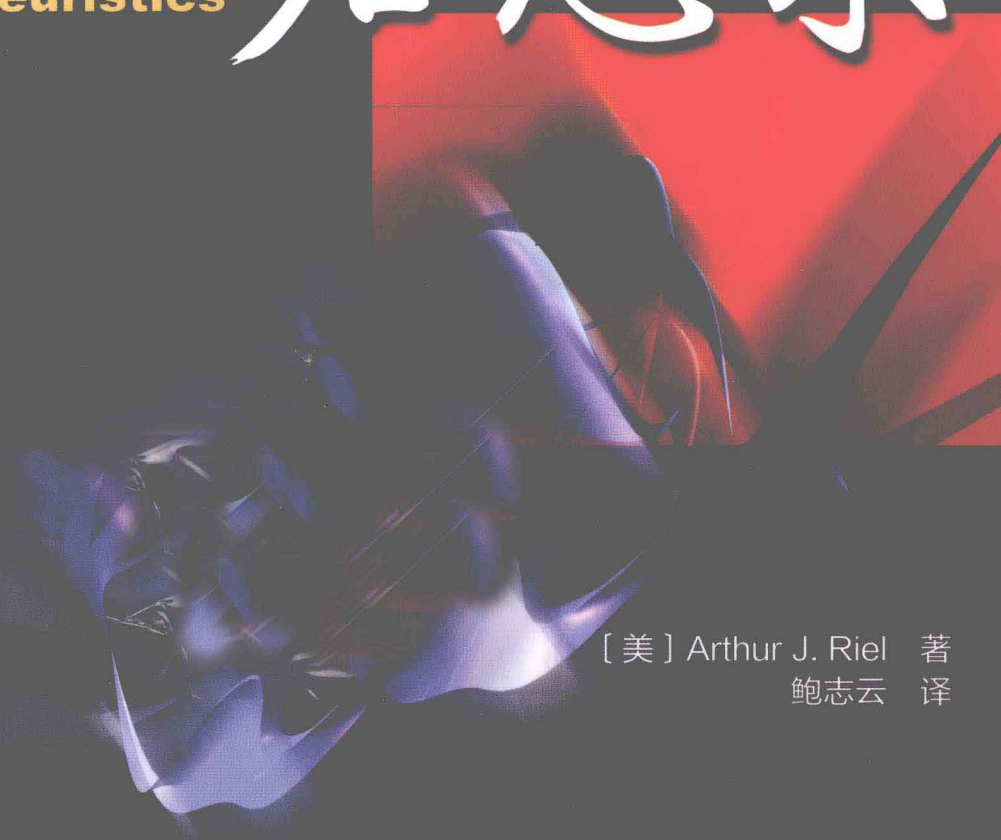
Addison Wesley

OOD

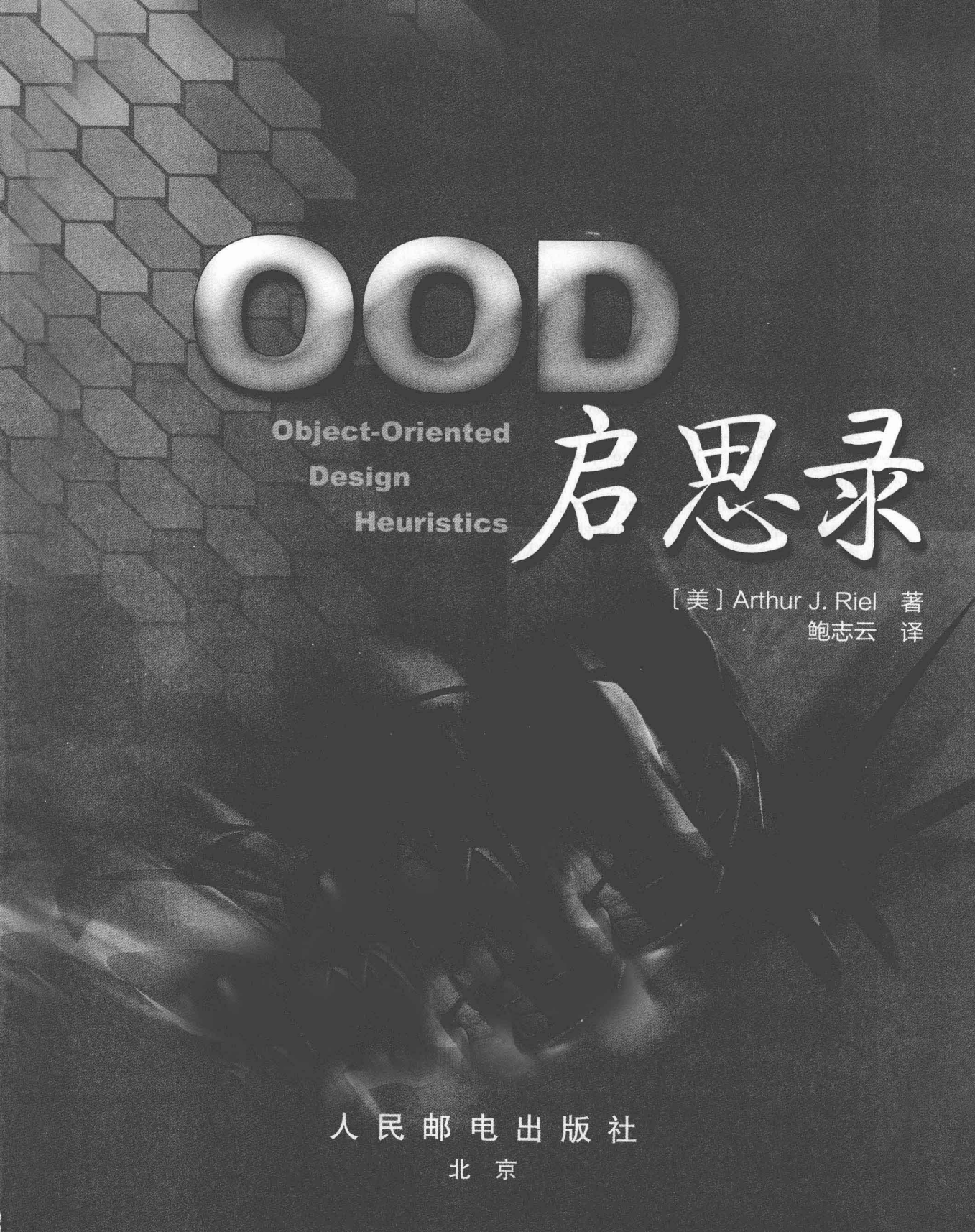
Object-Oriented Design Heuristics

启思录



[美] Arthur J. Riel 著
鲍志云 译

人民邮电出版社
POSTS & TELECOM PRESS



[美] Arthur J. Riel 著
鲍志云 译

人民邮电出版社
北京

图书在版编目（CIP）数据

OOD启思录 / (美) 里尔 (Riel, A. J.) 著 ; 鲍志云译. -- 北京 : 人民邮电出版社, 2011.12
ISBN 978-7-115-26543-2

Ⅰ. ①O… Ⅱ. ①里… ②鲍… Ⅲ. ①面向对象语言—程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2011)第202146号

版 权 声 明

OOD 启思录

◆ 著 [美] Arthur J. Riel
译 鲍志云
责任编辑 傅道坤

◆ 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
三河市潮河印业有限公司印刷

◆ 开本：800×1000 1/16
印张：23.5
字数：514 千字 2011 年 12 月第 1 版
印数：1－3 000 册 2011 年 12 月河北第 1 次印刷

著作权合同登记号 图字：01-2011-5495 号
ISBN 978-7-115-26543-2

定价：59.00 元

读者服务热线：(010)67132705 印装质量热线：(010)67129223
反盗版热线：(010)67171154

内容提要

本书提供了改进面向对象设计的真知灼见。

全书共 11 章，总结出了 60 多条面向对象设计（OOD）的指导原则。这些经验原则涵盖了从类到对象（主要强调它们之间的关系，包括关联、使用、包含、单继承、多继承）到面向对象物理设计的重要主题。本书将帮助你理解经验原则和“设计模式”这一流行概念之间的相互作用。你可以借助经验原则发现设计中所存在的某一方面的问题，而设计模式则提供了解决方案。

本书对各个层次的开发者都有价值，新手能借助本书走上通向面向对象编程的快车道，想提升自己的面向对象开发水准的老手则会受益于本书深具洞察力的分析。本书提供了让你成为更好的软件开发者的途径。

作译者简介

Arthur J. Riel 从事 C 和 C++编程工作已有超过 12 年的经验。目前，他每年在学术界和产业界讲授 40 多次课程。他参与了许多大系统的开发，曾就职于 AT&T 贝尔实验室、Draper 实验室、IBM、东北大学。他还在 *Journal of Object-Oriented Programming*、*The C++ Insider*、*The C/C++ Gazette* 等刊物上发表了众多文章。他还经常在 *OOPSLA*、*Object Expo*、*SCOOP*、*C++ World* 等顶级会议上演讲。

鲍志云　已翻译出版了《对象揭秘：Java、Effiel 和 C++》、《应用 MDA》和《解析 MDA》3 本译著，曾以“紫云英”为笔名在《程序员》、《程序春秋》等刊物发表多篇技术文章，并为《DDJ 软件研发》翻译技术文章。他在学生时代曾参加 ACM/ICPC 亚洲区比赛并获佳绩，曾参加“挑战杯”学术科技作品竞赛并获全国二等奖。他现供职于趋势中国研发中心（Trend Micro CDC）。

译序

拥有丰富面向对象开发经验，并且愿意无保留地告诉你自己的经验，而且还有这个天分能把这些宝贵经验讲述得深入浅出，这样的人不多。而本书作者碰巧就是。Arthur J. Riel 曾工作于贝尔实验室、Draper 实验室、IBM、东北大学，他既有丰富的开发经验，又有丰富的授课经验。而 Authur 整理自己开发与授课心得，并参考了众多面向对象经典著作后写就的《OOD 启思录》一书，也确实无愧乎读者的评价——“面向对象设计领域中的 *Effective C++*”。正如 *Effective C++*能助你迈向 C++专家层面，《OOD 启思录》能助你迈入 OOD 的殿堂。

这本书并不讲述 RUP 等方法学框架，也不讲述 UML 或者 C++语言，而纯粹讲述设计经验（其实原书书名中 heuristics 一词是启发式教学的意思）。这样的书在目前书市中汗牛充栋的 OO 著作中并不多见。而且，和开发方法学著作或者其他 OO 著作中以“best practice”（最佳实践）的方式传授设计经验的做法不同，这本书不仅告诉你好的做法，还告诉你不好的做法，并且告诉你如何识别，如何判断好的（糟糕的）做法，并且教你在面对多种可能的设计方案时如何作出取舍。拥有这些能力正是有经验的优秀开发者同初学者的区别所在。作者能够把这些往往“可意会但难言传”的设计经验提炼成易学易记易参考的“经验原则”的形式，而且伴以分析、讨论与实际应用例子来帮助你理解和消化这些经验，这是非常难能可贵的。

书的语言载体是 C++。这只是因为写作时 C++恰好是业界最流行的语言，并不意味着这些经验原则都只适用于 C++。其实，这些凝聚着智慧的设计经验中大多数是跨越语言（并且跨越时代）的，同语言相关的经验原则只占全书很小一部分。所以，如果您只会 Java 或者 C#，本书对您也依然很有价值并且容易读懂。这是因为书中的 C++代码是比较朴素的风格，没有大量采用模板，没有采用现代的泛型编程技法，看起来和 Java、C#的语法差不多。

对国内的读者，这本书又多了一层含义。任何学过一门编程语言的人，若是想要做一些真正的项目，写出高质量的代码，都会有向优秀开发者学习第一手经验的迫切需求。但偏偏目前国内的软件开发教学尚有与实践脱节之处，目前大学计算机系教学计划中基本都有 C++或 Java 语言课程，但是很少有关于设

计经验的课程，而且高校中也缺乏这样的师资，在这本书之前也缺乏这样的教材。若是纯粹依靠自己开发摸索来获得设计经验，又需要耗费很长时间，而且会走很多弯路，且不够系统。而《OOD 启思录》正好填补了这个空白，很适合刚学完编程语言，希望向面向对象设计层面进阶的读者。

说到设计经验，就不得不提到模式。或许很多读者看过或者听说过《设计模式》这本书，并对书中归纳的 23 个经典模式耳熟能详或者倒背如流。但不知您是否熟知在怎样的场合应当使用哪个模式，又为何要使用那个模式？记住模式容易，学以致用难，因为在实践中场景是千变万化的，难以判定应当采用某一模式。而这正是本书解决的问题之一。《OOD 启思录》第 10 章讲述了书中列出的经验原则同模式的关系：一条经验原则对应了一个或者多个变换模式——从反模式（糟糕的设计——违反了某条经验原则）变换成适合设计需要的模式。

现今，糟糕的设计还有另一个昵称："bad smell"（见 Martin Fowler 的《重构》一书）。那么，变换模式所做的也正是以相应经验原则为指导，把具有 bad smell 的设计重构成合适的模式。这倒是和后来 Joshua Kerievsky 的想法"Refactoring to Patterns"殊途同归了。

依照目前的趋势，可以预言，将会有越来越多的开发工具支持自动识别"bad smells"（Together、Pattern Testing 等），或者支持自动重构（很多 IDE 都支持了），甚至两者皆支持（那就近似可以自动或半自动地改善设计质量了）。事实上，这种做法正是将专家经验植入开发工具。那么，我们是否还需要看这本书中列出的专家经验呢？面对内置专家经验的"智能开发工具"的"自动修改或者建议修改你的代码/设计"的新功能，有 3 种可能的态度：[1] 放弃对代码的控制权，任工具自动去修改 [2] 关闭工具的这项功能 [3]掌握工具这项功能的背后原理，知道每个修改建议的"所以然"，在自己保有对代码控制权的同时借助工具新功能来大幅提高效率。如果您觉得[3]比较适合您，那么最好还是看一下这本书。

本书附有大量代码，可通过如下链接下载：

http://www.awprofessional.com/isapi/product_id~{457BA62C-4660-4632-922B-4CC6E2BDE0BA}/selectDescTypeId~{A5B39B29-323D-4AF1-B0F8-554D9C309CD3}/st~{7CB2A4C6-CB65-41C4-B301-E9516A7E6930}/session_id~{7035CB14-FFF7-4580-A6BE-8DC5EF623BD4}/catalog/product.asp

原书印刷版本并未包含网站上的全部代码，而只是摘选，所以译本也照此处理。此外，因为网站内容更改容易，书本内容印出来后却难以实时更新，所以网站上代码和书中代码细节之处可能会略有"版本差异"，但不会影响阅读和理解。

最后，要感谢人民邮电出版社的陈编辑在众多 OO 书籍中独具慧眼地挑出了这一本。虽然书中内容对我而言多属"温故而知新"，但在翻译过程中我还是获得了不少启迪，受益良多。希望这个译本也能带给您帮助，并且能给您带来愉快的阅读体验。如您发现译文有不妥之处，请不吝指正。我的 E-mail 是：wesley.bao@acm.org 。

鲍志云
于南京
2004 年 7 月

前言

在向几千名学生讲授面向对象分析、设计和实现的过程中，我逐渐意识到，业界很需要可以帮助开发者做出正确决定的指导原则。自 1987 年起我就开始查阅文献，试图找到可以适用于不同层次的开发过程的效率和复杂性参考基准，以便改善面向对象应用程序。除了那些从文献里找出来的指导原则，我还增加了自己的“家酿”原则。这样，我就得出了大约 60 条原则。其中有几条颇有些玩笑意味，但它们同样值得重视。我曾考虑是否把这些原则命名为“面向对象设计与分析的 60 条黄金准则”，不过我想起了 Dykstra 那传奇般的论文 Goto Considered Harmful，该文给用 goto 语句的人戴上了“异教徒”的帽子（听起来就好像他们应当被绑在公司大院的木架上施以火刑一样）。这非常重要，因为该文给业界提出了规则，阻止了使用 goto 语句的人继续有意或无意地破坏他们的系统的可维护性。不幸的是，这样的规则也造成了不良影响：出现了一群心理不太正常的作者，25 年来一直在发表文章声称“在应用程序的某个诡异角落明智地使用 goto 语句，能让代码的可读性高于对应的结构化代码”。当然，这些论文后面还跟着一堆“驳论文”，后面还有“驳—驳论文”，等等。

为了避免类似的宗教战争再次发生，我把本书中的 60 来条指导原则称为经验原则（heuristics）。你不必严格遵守这些原则，违背它们也不会被处以宗教刑罚。但你应当把这些原则看成警铃，若违背了其中的某一条，那么警铃就会响起。你应当严肃对待这些警告并细致检查。如有必要，你应修改设计以消除警告。当然，如果在某个例子中有一些经验原则因这样或那样的原因而不适用，这也是完全正常的。事实上，在面向对象设计的特定场合下两条经验原则常常会互不相容。开发者需要判断在该场合下哪条经验原则更重要一些。

虽然“创建一种‘Riel 的面向对象分析/设计方法学’”想法听起来很诱人，但本书没有发明新的面向对象分析或设计方法学。业界已经有够多的方法学了，它们提供了类似的或是重复的建议，并且对相同的概念使用完全不同的术语。无论采用何种方法学，面向对象开发者遇到的典型问题出现在设计完成之后。但迄今还没有人认真地去解决这个问题。这个问题是：“现在我做完设计了，但这个设计是好？是坏？还是好坏参半？”如果去问面向对象专家，那么好的设计获得的评价是常常是“这个设计‘感觉起来’挺好的”。对开发者来说，这个答案价值不大，但这样的回答却有其内在真理。这位面向对象大师会在下意识里让这个设计走过一个经验原则列表。这个列表是随这位大师的设计经历而积累起来的。如果设计顺利通过了这个经验列表，那么“感觉挺好的”，否则这个设计“感觉不太对”。

本书试图描述出这个原本存于大师们潜意识中的经验原则列表，并且以现实例子为支撑。读者们很快就会意识到，有一些经验原则比其他的原则更重要。经验原则的重要程度取决于违反这条原则造成的后果有多严重。不过我并不把这些原则按照重要程度排序，因为我觉得在很多情况下优先次序取决于应用领域和用户需求，是不能在本书中量化的。例如，在设计中有时会遇到两条经验原则背道而驰的情况，其中一种常见情况是用复杂性和灵活性之间的取舍。问一下你自己，你最想要的是什么，是增加灵活性还是降低复杂性。这样，你就会意识到难以在本书中列出经验原则的优先次序。这些设计经验原则是以现实世界中的例子为背景而定义的，这些例子覆盖了各条经验原则所属的领域。实际例子是向初学者解释面向对象技术的概念的理想平台，因此本书很适合于初学者。借助本书，初学者可以驶上理解面向对象编程的快车道，而不必徘徊于甚至迷失在充斥于 OO 世界的时髦用语之间。与此同时，对经验丰富的面向对象开发者而言，如果他们正在寻求良好的分析和设计经验原则的话，也会发现本书独具魅力，能为他们的开发带来很大帮助。

第 1 章探讨面向对象编程的动因，以 Frederick Brooks 发表于 1987 年的《没有银弹》一文（见参考文献）中论及的一些问题入题。我对面向对象编程的看法是，这是面向动作开发之后的自然进步，或称为演进。软件变得越来越复杂，所以我们必须达到离机器更远的一个新抽象层次，这样才能继续控制软件开发过程。正如结构化方法要比自底向上的编程高一个抽象层次，面向对象技术也比结构化方法提高了一个抽象层次。这并不意味着“自底向上编程或者结构化方法是错误的，面向对象编程才是正确的”。如果能用的内存只有 4KB，那么自底向上的编程完全适用；若能用的内存有 256KB，那么结构化方法非常合适。不过随着硬件越来越强大且越来越便宜，软件的复杂性也扶摇直上。在 20 世纪 80 年代早期，开发者还不需要考虑图形用户界面和多线程应用程序的复杂性；那时候司空见惯的是简单的菜单驱动单线程应用。但是，在不久的将来，没人会购买不带移动视频和语音识别这类多媒体特性的软件产品。越复杂的系统就需要越高层次的抽象，面向对象范型就提供了这一抽象。这不是软件开发的革命，只是正常的演进。

第 2 章讨论了类和对象的概念，它们是面向对象技术的基石。我们把类和对象看作数据及其相关行为的封装，数据和行为的关系是双向的。我们通过现实世界中的例子探讨了发送

消息、定义方法、设计协议的表示法。从这个章节开始陆续列出经验原则。因为到本章为止我们只接触了面向对象范型的一个小小子集，所以这些经验原则是较为简单的，但这并不意味着它们的用处小于稍后章节出现的较复杂的经验原则。

第 3 章探讨了面向动作和面向对象布局之间的不同。方法学的不同布局昭示了面向对象开发的核心真理。面向动作的开发在很大程度上专注于对任务集合的控制，这些任务是通过功能分解的，控制机制是“中央集权”式的；而面向对象的开发则专注于分布式的交互实体集。我确信，这一“范型迁移”意味着思考方式从集中式模型到分布式模型的转变。对于我们这些成长于面向动作开发世界的人来说，面向对象开发的学习曲线也就是面向动作开发的遗忘曲线。我们生活的现实世界更接近于对象模型，而非集中控制机制影响下的模型。很容易看出哪些系统的开发者的思维范型尚未迁移：如果系统中有一个位居中央的全能对象，其他对象都退居次要，那么这样的系统是由依然执着于面向动作布局的开发者创建的。本章提出了很多经验原则，用来指导应用程序布局的优化。

第 4 章到第 7 章通过一系列现实世界中的例子探讨了 5 个主要的面向对象关系：使用（第 4 章）、包含（第 4 章）、单继承（第 5 章）、多继承（第 6 章）、关联（第 7 章）。面向对象设计者感兴趣的大多数经验原则都可以在这些章节中找到。关于继承的章节包含了很多常见的误用继承关系的例子。对减少“泛滥成灾的类”的问题（比如为一个特定的应用设计了太多的类）而言，这些信息至关重要。“泛滥成灾的类”的问题是面向对象开发失败的主要原因之一。

第 8 章探讨了与类相关的数据和行为，这和与对象相关的数据和行为相对。发票类被用作需要与类相关的数据和行为的抽象的例子。我们既描述了 SmallTalk 元类也描述了 C++的关键字机制。此外，我们还比较了 C++元类（也即模板）和 SmallTalk 的元类表示法。

第 9 章讨论了在开发面向对象系统中面向对象物理设计的角色。虽然关于物理设计有很多可以说的，但很多都和面向动作范型中讨论过的东西是重复的。比如，已经有大量文献详细讨论了高效实现的粒度问题（例如，为了提高应用程序的速度，在逐条检测语句之前，请先考虑：替换硬件、替换编译器、替换机制、替换算法）。因此，本书中讨论的物理设计问题要么是只有面向对象范型中才存在的，要么是面向对象范型提供了特殊解决方案的。其中包含：把不友好的（比如，非面向对象的）子系统同面向对象的问题域隔离开的软件包装层、空间持久性和时间持久性、面向对象数据库管理系统和关系数据库管理系统、内存管理和垃圾收集、引用计数、最小公有接口、并行面向对象编程以及用非面向对象语言实现面向对象设计。

1987 年，我参加了 OOPSLA 会议的一个小型研讨会，会上讨论了面向对象范型的过去、今天和未来。在会议上，Kent Beck 谈及了 Christopher Alexander 在建筑学领域发表的论文。Alexander 认为，所有的建筑都有“无名的质”，他试图用称为“模式”的实体来表述这些“无名的质”。Kent 论及了寻找模式（也即已知问题的独立于领域的解决方案，或者面向对象构

架中的有趣结构）的可能性。近来，大量研究正在这个领域展开，模式已经成为对象群体的最活跃前沿。于是，我不得不问自己：“经验原则和模式之间有什么关系？”显然，两者是相关的，因为寻找它们的方式是类似的。我们研究任何一个在许多不同领域出现的结构或问题，然后试图用经验原则或者设计模式的格式来封装该实体。本书第 10 章讨论了设计模式和它们同设计经验原则之间的关系。我相信，模式和经验原则之间最有趣的关系是：经验原则可以告诉设计者，何时该用几个设计模式中的某一个了。对普通的设计者而言，模式的尺度超过了能靠直觉掌握的范围，而经验原则则正好相反，它们很少超过两句话的长度，而且很易于应用。这两者组合起来可以发挥很大的效力。本章也描述了一些设计模式和经验原则共同具备的有趣特性。

本书中前面几个例子可能微不足道，或者不属于计算机科学的领域，但请读者不要一上来就大加鞭挞。在我的讲座中，常常会听到一小部分参与者不久就开始抱怨，他们低声说：“这些信息毫无用处，因为我不会去编写水果篮、有尾巴的狗或者闹钟。”既然本书中很大一部分讨论的是现实世界中的日常事项，那么我可以给出一个合理的解释：如果设计经验原则和模式确实是独立于领域的，那么我为何不选择一个简单的领域来讲授它们呢？在我曾讲授过的设计课程中，我常常会听到某个设计小组大叫：“这是苹果去核问题”或者“这是具有可选尾巴的狗的问题”。一旦理解了一条经验原则，那么如何把它的运用扩展到别的领域就很简单了，不管那个领域有多复杂。

我把第 11 章纳入本书，这是为那些想看到更具“真材实料”的设计例子的人准备的。第 11 章围绕自动柜员机（ATM）展开讲了一些分析和设计问题。很多叙述面向对象范型的不同书籍都使用了 ATM 的例子。我选择这个例子，是因为读者对它很熟悉，而且这个例子在更加“计算机科学化”的领域中描述了设计经验原则和模式的使用。此外，因为这是一个分布式系统（ATM 和银行位于不同的地域），所以我可以在这个例子中描述称为“用代理设计”（design with proxies）的设计技法。这种设计技法使得系统构架师在进行逻辑设计时可以忽略应用程序的分布特性，把这些问题推迟到设计后期。这很重要，因为很多分布式系统的设计缺陷的原因都可以归结为在逻辑设计完成前就考虑分布式处理。

最后，我要说的是，在我的所有课程上，都注意到班级分成了两拨人。一拨人喜欢在抽象领域中生活，他们为了设计而讨论设计，很少讨论实现的问题。另一拨人则难以理解抽象的东西，但是如果你把一段代码拿给他们看，他们很快就会领会讨论的要点。当本书交付审阅时，我请每一位审阅者都指出，这本书是否应当用 C++实现描述设计实例。两位审阅者说，显然这本书需要那些 C++例子，如果没有这些例子的话读者会很难理解那些抽象的概念。另外两位审阅者则说，这是一本关于设计的书，因此同 C++或者别的编程语言没有关系。还有两位审阅者则站在相对中立的位置。这样，我就面临着显而易见的二难困境，无法让所有人都感到满意。我的解决方法是，在正文后面列出了一系列精选的 C++实现作为附录，这些实现所对应的设计例子则在正文中的章节讨论。如果你是个抽象主义者，那么你不读附录就可以了；如果你习惯于通过研究对抽象概念的实现来学习（在多数时间里我也是这样的），那么

你或许会想要研究附录中特定设计问题的实现。这种安排没有用代码扰乱正文中对设计的陈述，我希望这种安排也能让期待看到实现代码的人满意。

请注意：所有 C++例子都在奔腾 100MHz 的 IBM PC 兼容机上用 Borland C++ 4.5 编译测试通过。你常用的 C++编译器也应该可以毫无问题地编译这些例子。

目录