IT Trend 系列丛书

解 析 MDA

Anneke Kleppe
Jos Warmer 著
Wim Bast

鲍志云 译

人 民 邮 电 出 版 社

图书在版编目(CIP)数据

解析 MDA / （美）克莱普（Kleppe, A.）等著；鲍志云译.
—北京：人民邮电出版社，2004.2
ISBN 7-115-11812-4

Ⅰ.解… Ⅱ.①克…②鲍… Ⅲ.软件开发 Ⅳ.TP311.52

中国版本图书馆 CIP 数据核字（2003）第 121852 号

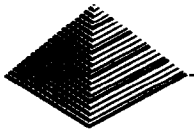
版权声明

内容提要

MDA（模型驱动架构）是OMG提出的新的软件开发方法学。本书通过精练的文字、独到的案例介绍MDA的概念、应用和相关技术。

全书可分为3个部分，共12章。前3章介绍MDA的基本概念、框架和发展现状。第4～6章，以“Rosa早餐服务系统”为例，介绍MDA的应用和实施。第7～12章探讨变换、元建模、变换的定义、OMG标准和其他技术等深入话题，最后展望MDA的未来。全书的最后给出了术语表和Rosa系统的代码。

本书能够帮助IT项目经理、开发者和研究人员认识和了解MDA，也适合对MDA感兴趣的读者阅读参考。



出版者的话

在知识经济时代，IT 技术的每一次创新和发展总是带动着生产力巨大的进步和提高，也吸引着大量的优秀人才不断投身技术革命。作为 IT 行业从业者，不管你是经理人还是普通的工程技术人员，都需要敏锐的技术嗅觉、灵活的创新精神和旺盛的学习能力，以适应不断变化发展的技术趋势。

软件产业在 IT 产业中历来占有重要的地位，进入 21 世纪，这种地位继续得到巩固和凸现。纵观软件产业的发展历史，计算技术经历了“以机器为中心—以应用为中心—以企业为中心”的变化，同时，我们不难得出这样的结论：软件开发方法的进步有助于维持软件产品质量、全寿命期、生产成本的平衡。

以企业为中心的计算主要包括以下方面：

- 基于组件开发（component-based development）——将制造业中发展成熟的概念应用到软件开发中去。
- 设计模式（design patterns）——这也和制造业过程有共同之处。
- 中间件（middleware）——它在操作系统之上又迈出了提升计算平台抽象层次的一步。
- 说明性规约（declarative specification）——相对简单的说明取代了相对复杂的程序代码。
- 企业构架（enterprise architecture）——通过分离关注点来组织企业软件。
- 企业应用集成（enterprise application integration）——将孤立的遗产系统集成进企业系统的整体。
- 契约式设计（design by contract）——它有助于推动高质量的软件工程。

作为将这一系列新的趋势性技术整合到一起的开发方法，Model Driven Architecture（MDA）的出现，为提高软件开发效率，增强软件的可移植性、协同工作能力和可维护性，以及文档编制的便利性指明了解决之道。也正因为如此，MDA 被面向对象技术专家预言为未来两年里最重要的方法学。

面对这一系列新的技术名词，你是否感到有些无所适从？或者你只是听说过，也许你对它们还略知一二，但相信大多数读者还没有系统地学习过相关的理论知

识。没关系，只要具备前面提到的 IT 从业人员的优秀素质，你肯定会对这些技术话题产生浓厚的兴趣，那么，你最需要的就是比较系统的技术书籍。

“IT Trend”系列图书是人民邮电出版社从 Addison Wesley 和 John Wiley 两大知名的国外专业出版公司引进、精心策划和出版的一套高技术定位、高制作品质的系列图书。图书内容定位覆盖了我们前边提到的那些 IT 趋势性技术，而且原书几乎都是全世界范围内第一本论述相关技术的专著。

这套图书的首批出版计划包括：

《IT 体系结构与中间件——建设大型集成系统的策略》（*IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*）

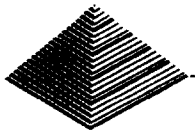
《应用 MDA》（*Model Driven Architecture：Applying MDA to Enterprise Computing*）

《解析 MDA》（*MDA Explained:The Practice and Promise of The Model Driven Architecture*）

《Design by Contract 原则与实践》（*Design by Contract by Example*）

《基于组件开发》（*Component-based Development：Principles and Planning for Business Systems*）

我们希望这套新技术图书能够为广大读者打开一扇窗口、提供一个机会，以了解那些正在和将要对 IT 产业发展产生重大影响的技术趋势，帮助你准确展望和把握行业的技术前景。我们期待“IT Trend”系列图书“一步领先”的精彩，能够带给你“步步领先”的收获。



译者序

软件开发是什么

Alistair Cockburn 在他的 *Agile Software Development*[1]一书中归纳了业界对软件开发的看法即将以 C.A.R Hoare 为代表的数学观、以 Bertrand Meyer 为代表的工程观、以很多程序员为代表的工艺观，还有一些程序员则认为软件开发是神秘的创造行为。当然，近 20 年来，也有越来越多的人对软件开发持建模观，比如 Ivar Jacobson 就曾声称：软件开发就是建模。本书作者也指出：代码就是模型。Cockburn 则在他的书中独树一帜地提出：软件开发是一种协作游戏。

自然，持不同软件开发观的项目主导者会关注软件开发过程的不同方面。为了节省资源，我们希望软件开发领域的研究者和项目主导者（实践者）的关注焦点是真正决定项目成败的那个方面。否则，学术界投入大量时间精力去研究对项目成败无足轻重的因素，项目主导者把大量人力物力用于控制项目中无关紧要的方面（如 Cockburn 调侃地指出的：开发场所的环境湿度），那岂不白费力气？

那么，这个“至关重要”的方面究竟是什么呢？是工具？是过程？是整个方法学？还是人？或者是别的我们尚未注意到的因素？目前，没有人知道确切答案。或许，每个方面都对项目成败有些影响吧。无论如何，因为 MDA 将会对软件开发的各个方面都产生深远影响，所以不管您对软件开发持何种观点，您都无法回避 MDA。下面我将简述 MDA 对软件开发各方面带来的影响。

MDA 改变了协作游戏的角色和规则

好吧，我们就按照 Cockburn 的说法，把软件开发看作协作游戏好了。不过，任何游戏总要有参与者和游戏规则吧？目前，编码员(coder)是重要的游戏参与者，

---

[1] 编者：本书英文影印版（书号：7-115-11786）由人民邮电出版社于 2003 年 8 月出版，中文版《敏捷软件开发》（书号：7-115-11364）于 2003 年 11 月出版。

但在 MDA 版本的协作游戏中，没有这个角色了，取而代之的是建模者。但是，MDA 也引入了另一个新游戏——这个游戏不是编写软件产品，而是编写变换规则。变换规则市场会逐渐成长，就像基于组件的开发启动了组件市场那样。在新游戏中，原来的编码员中的精英人物将找到他们新的位置，而他们也将自豪地发现，他们编写的代码将获得空前程度的复用。至于游戏规则的改变，我在这里说不好也说不全，请您在玩新版本的游戏时慢慢体会吧。

MDA 改变了开发过程

目前，许多项目经理都很注重开发过程。或许因为过程对项目成败真的很重要，或许仅仅因为过程是软件开发中项目经理惟一可以施加较大影响的方面。无论如何，MDA 对开发过程的改变不容忽视。

比如，开发过程的需求分析阶段依然存在，不过需求分析员要编写的不再是需求分析文档，而是 PIM——平台独立模型。需求分析文档和 PIM 有什么区别？阅读需求分析文档的是人，是设计师或者程序员，但阅读 PIM 的则主要是类似于编译器的自动工具。

既然需求分析阶段产生的工件改变了，那么依赖需求分析阶段结果的设计阶段自然也要改变，而“编码”这项工作则需要完全重新定义了。测试、部署等阶段也会有相应改变。此处不再详叙，请阅读本书正文。

MDA 改变了开发工具

随着技术的进步，开发工具的改变一直都没有停止。当主流开发语言是汇编的时候，您可曾想象到会有包含自动完成、重构功能、集成调试器的 IDE？你可曾想到会有一天汇编代码不再由人手写而是由编译器自动生成并且可以高度优化？那么，当主流开发语言的抽象层次即将再次跃升，开发工具的革命也将到来。在 MDA 的世界中，“变换工具”扮演了传统编译器的角色，传统编译器则退居目前汇编器（就是把汇编语言翻译成机器语言的程序）的地位，其余各层工具依次后退。调试器也将逐渐进化，就如同从机器码级调试（汇编语言级调试）向源码级调试的过渡那样，慢慢从源码级调试过渡到模型级调试。在 IDE 中最重要的也不再是基于文本的代码编辑窗口，而是基于图形的建模窗口。人们将像现在谈论一个 API 函数那样谈论一个设计模式（design pattern），而代码模式（idiom）将完全由变换工具自动生成，不再是人们关心的内容。

MDA 让你重新认识文档、代码、模型

以前，我们倾向于认为，给人看的文档或者模型不需要写得太精确，因为人总会有很强的理解力，人的大脑能够“全自动”地更正一些无关紧要的错误并补全一些省略之处。另外，文档或者模型写得太精确是浪费时间，因为文档和模型又不能变成可以运行的产品，你总是需要用代码把模型重新翻译一遍。Cockburn 和 XP 的一些推崇者的观点更极端：文档和模型不重要，人们拿着文档或者围在画着模型的白板前的讨论才重要，因为真正的沟通不是发生于阅读文档之时，而是发生于人与人的讨论中。

好吧，或许以前确实如此。但 MDA 将完全颠覆这一现实。模型不再主要是给人看的了，而主要是给机器看的。写得精确一点也不再是浪费时间，因为只写一遍（您不需要再把文档和模型手工翻译成代码）而且早晚要认真地写一遍。至于围在白板前的讨论——如果是在讨论如何编码实现某个模型，那么很抱歉，这样的讨论不再需要了。当然，其他方面的沟通还是需要的，但必须承认，游戏规则已经改变，游戏中的关卡已经改变，您有了不少新的“通关任务”，而很多老任务则自然取消了。

MDA 带来了数学般的精确性

是的，凡是能让机器理解和自动处理的东西都必须是数学般地精确。您在编译程序时有没有遇到过这样的编译器信息：“警告：第 nnn 行代码具有二义性”？那意思就是，请您把代码写得更精确些。那么，MDA 要说的就是，请您把模型建得更精确性。MDA 工具会严格检查您的模型以确保这一点。

MDA 为新方法学提供了土壤

如果软件开发是游戏，那么方法学就是攻略。或许高手不需要攻略也能把游戏玩通关，但大多数人在攻略的指导下能少走很多弯路。MDA 制定了新的游戏规则，那么我们自然可以期待新版本的攻略如雨后春笋般出现。即便是同一个游戏，您手中有了不同的战斗工具、不同的装备，玩法也会不同。那么，既然 MDA 带来了很多新一代的工具，新的方法学的诞生也不足为奇了。

既然提到方法学，我就再多说几句。把软件工程中“methodology”这一术语译为“方法学”其实颇具误导性，因为这个词的内涵往往不是哲学老师常挂于嘴边的“世界观和方法学”的那个方法学，而是指一系列你需要照着做的方法，或

者说一系列约束开发人员的规则。它不是“研究方法的学科”，而就是一系列方法和规则的集合。

按照规则的多少和约束的强弱，我们可以大致地把方法学分为重型和轻型两种。重型方法学比较正规和严谨。在采用重型方法学的项目中，开发人员具有较强的可替换性，因为方法学本身强制要求开发者把他所创造的所有东西都记录在案(按照该方法学规定的格式)，所以参与项目的新人能借助这些文档很快上手(前提是新人也熟悉这种方法学规定的格式)，这样，开发人员跳槽对项目的冲击也相对较小。项目经理们可能会比较偏爱这样的方法学，因为这样一来他们掌控的因素比较多，风险就比较小。开发人员则不会喜欢这样的方法学，因为在采用重型方法学的项目中，他们只是可替换的螺丝钉，难以感觉到自己的重要性；而且做连篇累牍的文案工作纯属利他（对经理、对可能加入的新人有利），毫不利己（很无聊很费时间，而且占用的是自己本可用于开发的时间）。

轻型方法学则具有相反的特质。记录在案的东西不多，交付的就是代码以及可用的产品，当然还有测试用例。大多数交流是口头的、非正式的，很高效，但也只存在项目成员的脑海中。如果成员从项目中离去，那么他脑海中的这些东西也随之带走。因为开发人员往往都希望自己具有不可替代的重要性，而且一般都觉得写程序比写文档好玩，再者轻装向前可以走得比较快（因为不必把时间浪费于编写正规文档），所以开发人员一般都比较偏爱轻型方法学。

一般而言，大型项目采用重型方法学好一点，因为项目人手多，周期长，即便所有员工都很爱戴老板、很忠于公司、很喜欢这个项目，但这么多人在这么长时间内一个都不跳槽、一个都不生病、一个都不结婚生孩子也是挺难办到的。而小型项目则往往采用轻型方法学好一点。Cockburn 提出的水晶方法族就充分考虑了项目规模的因素，当然，还考虑了项目紧要性等别的因素。

那么，MDA 有没有对某种类型的方法学特别垂青呢？没有，MDA 是“轻重咸宜”的。既然 XP（极限编程）俨然已是轻型方法的招牌，那么自有好事者用模型替换代码，提出了 XM（极限建模）。轻型方法的另一特征是迭代和重构，MDA 极佳的同步特性也为之提供了有力支持。而重型方法也能从 MDA 获益匪浅。重型方法有一大特征就是书写详尽的文档，创建大量的模型，那么 MDA 说：让文档更详尽些吧，让模型更精确些吧……详尽精确到机器都能理解并自动编译了，这一量变到质变的转换也就完成了。

从业界，从本书中，我们已经看到，一些传统的方法学正从 MDA 这一变革中汲取新的养分，而新的方法学也能从 MDA 的土壤中茁壮成长。或许未来若干年中，又会有一批涉及 MDA 的新方法学著作出现。

创造性的脑力劳动是无可替代的

所有的改革都会在一定程度上重新分配社会资源， MDA 也不例外。不过 MDA 所威胁到的是只会老老实实地把详尽的设计文档翻译成 C++或者 Java 代码的人。

社会发展的历史在一定程度上是一部机器逐渐替代人的劳动的历史。在某种意义上，部分人失业是社会进步的必然代价。不要试图阻止技术进步的脚步，因为技术进步的同时也会创造新的工作机会。比如 MDA 很可能就会创造出新的变换定义集市场。但是，只要您从事的工作具有创造性，就无法被机器取代。

软件设计是需要创造性的，这一创造性或者体现在代码中，或者体现在文档中。在 MDA 出现之前，如果我们认真地编写文档，然后认真地编写代码，那么我们进行了两遍创造性劳动，这浪费了劳动力。而有些软件成熟度（CMM）级别高的企业（特别是印度和日本企业）是这样做的：认真地编写文档，代码则是文档的精确翻译。更多的中国企业则是这样做的：文档敷衍了事（敷衍 CMM 检查组或者敷衍上级领导和客户），创造性劳动则在编码阶段做。这些做法的优劣我在这里不去评述，但只要您做的是创造性工作，那么在 MDA 的世界中您会如鱼得水，因为工具只是为您节约了做无聊琐事的时间，让您可以把精力集中到创造性过程中去。

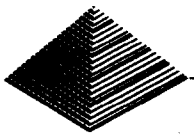
前段时间曾有业界“大量需要软件蓝领”的声音，我不知道当时是否真的有此需要。但我在此大胆预言：MDA 一旦普及，软件蓝领会大量失业。因此，我敬请读者不要把“软件蓝领”作为自己的职业生涯目标。如果要在未来立足软件开发业，请您永远不要放弃自己创造性思维的能力。此外，花些时间阅读一下您手中这本薄薄的小书对您也会有些帮助。

希望本书会带给您愉快的阅读体验。本书翻译中若有不当之处，欢迎您来信指正。我的 E-mail 地址是：zmelody@sohu.com。

鲍志云

2003 年 11 月

于南京



原书序

“开始越早，耗时越长。”1975 年，Fred Brooks 在他的开创性的软件工程著作 *The Mythical Man Month*[1]中提到了这句看似荒谬的话，目的是强调在软件工程中“磨刀不误砍柴工”。如果你在需求分析上偷工减料，那么你就会浪费时间去设计客户不想要的东西。如果你在设计上压缩时间开支，那么你就会编写很多无助于解决客户问题的代码。还可以接着类推下去。

Brooks 博士的格言在警示我们：良好的设计不容忽视。20 年前如此，今天依然如此。但是，很多其他东西已经改变了。他的书基于他在 OS/360 项目的经验，OS/360 是当时最庞大的软件项目。但是自那时起软件大小和复杂性的增长大大出乎人们的意料，可供选择的实现技术也是“乱花渐欲迷人眼”。我们应当使用 C++，Java，Visual Basic，还是 C#？选择 CORBA，还是.NET？或者选择 Web Service，ebXML，EJB，JavaScript，ASP，JSP，SQL，ODBC，JDBC？今天，一个软件项目可能会使用前面列出的好几种技术，让它们各尽所能。我们也越来越意识到软件维护的重要性。一些执行关键任务的软件可能会连续用上好几十年，这期间还要饱受需求变动和技术革新带来的升级之苦。随着时间流逝，在维护上的开销逐渐超过了其他所有花销的总和。

本书描述了模型驱动构架（Model Driven Architecture, MDA）方法，它可以用来创建良好的设计，应付多样化的实现技术，延长软件的生命期。MDA 基于广为使用的可视化、存储、交换软件设计和模型的业界标准。在这些标准中，最著名的就是统一建模语言（Unified Modeling Language，UML）。OMG 创建了 UML，为工程师交换和表达思想提供了通用的、广泛理解的可视化语言，从而为良好设计加油鼓劲。可视化建模的运用也因此而快速普及。但是，人们常常只把可视化建模看作是一种给软件画画的方法，接下来还必须花大力气才能把这样的画翻译

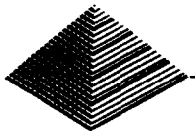
[1] 译注：中译本名为《人月神话》。

成可执行代码。缩减设计的一个传统理由是，详尽的模型只不过是一堆纸，与其浪费时间纸上谈兵，还不如把这些时间用在真刀真枪写代码上面。部分地出于这个原因，今天仍有一种趋势“强调创建可执行代码，而不是‘无关紧要’的设计”。

与此相反，MDA 非常强调建立设计。不是纸上谈兵的设计，而是存储在标准仓库（repositories）中、机器可读的模型。投入到这些模型的智力成本不是无所事事地坐在纸上等着被重新塑造成代码。相反，MDA 模型可以被多家厂商提供的自动化工具识别并用来自动生成模式（schema）、代码框架（code skeleton）、测试框架（test harnesse）、整合代码（integration code）以及部署脚本（deployment script），能够支持典型项目中用到的多种平台。因此，投入到 MDA 模型的设计努力会被多次复用来生成各种组件。这些模型可以在应用的整个生命期中不断更新，以精确反映经过多次维护的软件现在是怎样工作的，而不是停留在设计阶段结束时的一幅静止画面。简而言之，MDA 是一个在今天多平台 IT 环境中创建良好设计的构架。

本书的作者们描述和书写 MDA 前景实属游刃有余，因为他们在 OMG 参与了 MDA 的创建，还参与了实现 MDA 的商业产品的开发，并且在实践中使用了 MDA。本书扎根于坚实的理论，但它依然是一本实践者手册，围绕着现实世界中的例子而展开，并为那些永远面临着“按时按预算地完成明年的项目”的难题的 IT 专业人士提供了指导。阅读本书的时间绝不会是浪费，因为，请记住：“开始越早，耗时越长”。

Andrew Watson
OMG 副总裁兼技术总监
2002 年 11 月 17 日



作者序

多年来，我们 3 人一直在用面向对象技术开发软件。一开始，我们使用面向对象的编程语言，比如 C++、Smalltalk 和 Eiffel。不久，我们就感觉需要在更高的抽象层次描述我们的软件。在最早的 OOA/OOD 方法（比如 Coad/Yourdon 和 OMT）出现之前，我们使用自己发明的泡泡箭头图。这就自然地带来了这样的问题："那个箭头是什么意思啊？"，"圆圈和方块有什么差别啊？"因此，我们很快决定使用新出现的方法来设计和描述我们的软件。

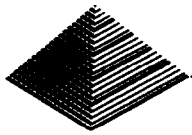
在这些年中，我们发现，我们花在设计模型上的时间比花在编码上的时间还多。模型能帮助我们搞定越来越庞大复杂的系统。能有一个好的软件模型，写起代码来也省事不少，而且在许多情况下模型使得代码几乎不言自明。

在 1997 年，我们中有人参与了定义第一个面向对象建模标准，也就是 UML（统一建模语言）。这是推动软件业界开展建模运动的重要里程碑。当 OMG 提出了 MDA（模型驱动构架）时，我们感觉到这是顺其自然迈出的下一步。人们试图从他们的高层次模型中榨出更多一点价值来，MDA 方法能让他们得到更多。

那时，我们意识到，这些年来我们一直都很自然地向模型驱动开发之路走来。我们同自己创建的系统奋战多年所积累的每一点智慧，都与 MDA 这一"软件创建新概念"吻合得那么完美。我有这种感觉："啊哈，就是它了！"当我们在软件业多年摸索后第一次接触到面向对象思想；当我们第一次读到 GoF 的设计模式著作，我们也有这个感觉。那么，MDA 很可能就是软件开发方式的下一个重大进步。MDA 把软件开发的注意焦点提到了更高的抽象层次，因此也提升了 IT 业的成熟度。

我们知道，OMG 总裁 Richard Soley 向我们描述的美好而动人的 MDA 世界现在还未到来。MDA 的一些部分依然在开发之中，但是，也有些部分今天已经能投入使用了。我们想通过本书让你领会：在今天和明天，MDA 意味着什么，你又能从中获得什么。

Anneke Kleppe
Jos Warmer
Wim Bast
2003 年 3 月于荷兰 Soest



前　言

在计算机科学领域，模型驱动构架（MDA）还相当新。本书将以广阔的视角带你领略 MDA 的基本内容。我们回答了“什么是模型？它们同代码有什么关系？”这类问题。我们还讨论了 MDA 的优势以及怎样实现这些优势。

MDA 如何兑现许诺的好处？“模型的变换”这一概念起到了至关重要的作用。我们将解释什么是变换，有哪些类型的变换，怎样来定义变换。对于哪些类型和哪些形式的变换会给业界带来重大好处，哪些不会，本书也毫不含糊。最后，本书包含了许多有趣的实际变换例子。

谁应当阅读本书

本书面向的读者群很广。如果你想知道 MDA 是什么，想知道 OMG 的不同标准以及声称支持这些标准的工具都扮演了什么样的角色，那么本书就很适合你。本书能够帮助技术经理了解 MDA，并帮助他们判断在什么时候以什么方式将 MDA 用于他们的项目才是最佳的选择。本书也非常适合更具经验的软件开发者，只要他们对建模和在更高抽象层次上的编程感兴趣。我们假定读者已经了解了 UML（统一建模语言）和 OCL（对象约束语言）。如果你具有 Java、EJB、SQL、JSP 的知识当然更好，特别是这些知识能够帮助你理解书中的例子。不过你不懂这些东西也不要紧。

怎样使用本书

当然，书是用来读的，本书从头到尾都很有趣，所以我们鼓励你把它一次读完。但是，我们也知道不是所有人都有足够的时间和技术背景来完整读一遍，所以我们建议 3 种阅读方案。

经理：不想知道 MDA 背后全部技术细节的经理们应当阅读第 1 至 3 章、第 11 章和第 12 章。第 4 章是可选项。

开发者：对应用 MDA 开发软件更感兴趣的人应该阅读第 1 至 7 章、第 11 章

和第 12 章。

专家：为了彻底理解 MDA 的技术背景，你需要阅读整本书。

第 5 章、第 6 章和第 10 章的例子细节可以略读。这不会妨碍你对全书其余部分的理解。但是，理解这一点很重要：MDA 运用于现实世界中时，这些例子章节中的所有细节都确实存在，虽然它们有时候被黑盒封装了。

字体约定

所有代码实例都用等宽字体。

相关主题的信息

参考书目中列出的书籍能为你提供更多关于 MDA 的信息。下面的 OMG 站点也是一个信息源泉：

- http://www.omg.org/mda

本书的支持和样例的实现

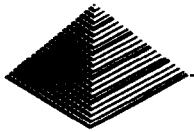
第 4、5、6、10 章中的 Rosa's Breakfast 例子可以用 OptimalJ 工具执行。你可以从下面的页面下载该工具的免费测试版以及完整的例子：

- http://www.klasse.nl/mdaexplained

致谢

我们要感谢审阅者们花时间阅读本书的不同程度的书稿。没有他们的建设性建议，这本书就不会是现在这个样子。他们与我们分享见解，我们从中获益匪浅。我们想要感谢 Frank Baerveld、Jeroen Bruijning、Aldo Eisma、Ghica van Emde-Boas、Peter van Emde-Boas、Martin Gogolla、Martin Matula、Hans van Oosten 和 Andrew Watson。特别的致谢献给 Heinrich Hußmann，他那时正专心“抗洪”以避免 2002 年夏天 Dresden 的洪灾弄湿他的脚和家具，但他还是认真阅读了本书的第一稿。另一位需要特别感谢的人是 John Daniels，他积极而彻底的批评促使我们对第一稿的很多部分进行了重写。

我们还要感谢 Addison-Wesley 的团队，特别是 Mary O'Brien ——我们的编辑，虽然姗姗来迟，却把所有的协作细节处理得那么妥帖。



目 录