HZ BOOKS
华章IT

PEARSON

软 件 工 程 技 术 丛 书

基于模型的软件开发

Model-Based Development Applications

[美] H. S. 莱曼（H.S. Lahman）著　廖彬山 王慧 马苏宏 译

机械工业出版社
China Machine Press

软件工程技术丛书

基于模型的软件开发

Model-Based Development *Applications*

[美] H. S. 莱曼 (H.S. Lahman) 著 廖彬山 王慧 马苏宏 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

基于模型的软件开发 /（美）莱曼（Lahman，H.S.）著；廖彬山，王慧，马苏宏译 . —北京：机械工业出版社，2015.7
（软件工程技术丛书）
书名原文：Model-Based Development：Applications

ISBN 978-7-111-50737-6

I. 基… II. ①莱… ②廖… ③王… ④马… III. 软件开发 IV. TP311.52

中国版本图书馆 CIP 数据核字（2015）第 150378 号

本书版权登记号：图字：01-2011-4809

基于模型的软件开发

出版发行：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：关　敏　　责任校对：董纪丽
印　　刷：三河市宏图印务有限公司　　版　　次：2015 年 8 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：23.75
书　　号：ISBN 978-7-111-50737-6　　定　　价：99.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

译 者 序

这是一本关于一种特定的软件设计方法实践的书。MBD（Model-Based Software Development，基于模型的软件开发）方法基本上是一种 OO（Object-Oriented，面向对象）方法。其基本观点是：通过静态结构和动态结构的开发，生成面向对象的分析模型，然后通过工具的转换，进而转换为应用程序框架。

本书第一部分着眼于面向对象方法诞生的历史背景，使我们能够了解传统方法存在的问题，也就是面向对象方法寻求解决的问题。因为 MBD 方法基本上是一种面向对象的方法，所以为了充分了解这种方法，第一部分大致介绍了面向对象开发的基础。由于面向对象的方法不如传统软件开发方法那样直观，因此需要理解面向对象的方法为什么通过这样的方式在实践中应用。

第二部分讨论面向对象开发的基本原则如何应用于 MBD 方法中。从应用的基本结构（即应用的骨架）开始，依次讨论应用的其他内容。在应用程序的整个生命周期中，该结构倾向于相对稳定，称作静态模型。面向对象范式的一个显著特征是对问题域的抽象，在静态模型中，与之相关的大部分内容都将实现。在静态模型中会识别面向对象应用的基本元素——对象，及其属性和相互之间的关系。通过在客户的问题域中抽象出这些内容，能够保证软件拥有稳定的结构。

第三部分讨论如何以一种系统的、一致的方法来描述动态计算需要的内容。这种系统的方法具有一个概念性的框架，称为动态模型。动态模型的基础元素（对象状态机）与静态模型中对象的行为扮演着本质上相同的角色。

本书的主要读者对象为具有较少 OO 经验的人。本书假定读者具有一些粗略的 UML（Unified Modeling Language，统一建模语言）知识。本书还假定读者具有一些软件开发经验。除此之外，本书第二类读者是从传统的程序开发环境向 OO 范式转变的开发人员。

希望大家在阅读本书时有一个愉快的经历。

本书的翻译工作主要由王慧和马苏宏完成，审校工作由廖彬山完成。

由于译者水平有限，书中错漏在所难免，恳请广大读者指正。

前　言

软件开发是一项极其复杂的智力活动，它是一门朝气蓬勃并且仍在迅速发展的学科。软件开发还不够完善，因此迄今人们仍然在试图找出开发软件的好方法。

尽管如此，多年来软件开发方法仍然获得了大幅提升。许多设计方法学不断发展以促进软件设计的各个方面。其中之一是结构化设计方法，该方法提供了一种非常直观的方式，用以很好地匹配图灵和冯·诺依曼的硬件计算模型。

问题

尽管结构化设计明显优于它之前的特定方法，但它存在着一个致命的弱点：当用户需求随着时间的推移改变时，软件往往很难随之修改，大型的应用尤其如此。与此同时，应用的规模和复杂性迅速膨胀。另外，新的语言、技术、操作系统、数据存储范式、用户界面范式、硬件等以惊人的速度出现在计算领域中。然而，商业条件一直在要求软件产品更快、成本更低地投入市场。

希望

因此，一些新的设计方法出现了，这些方法从实践中吸取了来之不易的经验和教训。同时，计算领域提出了一些革命性的新观点。其中之一就是面向对象的范式，其主要目标为：在软件产品的生命周期中，随着需求出现不可避免的变更，保证大型应用的可维护性。

本书介绍一种特定软件设计方法的实践，该方法称为基于模型的开发方法，其主要基础是 Shlaer-Mèllor 方法[⊖]。通常情况下应用 OO 范式，特定情况下应用 MBD 方法能够使大型应用获得更强的健壮性和可维护性。

本书主要内容

尽管本书使用 UML（Unified Modeling Language，统一建模语言）作为表示法，但其应用是很浅显的。有很多非常优秀的书籍描述了如何使用 UML 进行软件设计，

⊖ 方法的名称来自 Sally Shlaer 和 Steve Mellor 两位作者。该方法最早于 20 世纪 80 年代初期提出，之后经历了多次修改，产生了一些变种，MBD 方法是变种之一。

因此本书没有过多地描述UML的语法。同样，本书遵循了MBD的设计方法，但是该设计方法主要为下列真正目的提供背景支持：

本书的主要目标在于描述，为什么在一般情况下使用OO方法和在特殊情况下使用MBD方法是在宣传一种做事情的特殊方法。

不存在一种唯一正确的方法可以设计和开发所有软件。设计更多地依赖于特定的开发环境，包括从业务目标到工具再到团队文化等所有内容。最终，企业将决定在它的环境中哪一组工具是最高效的。为了做到这一点，决策者应当了解为什么MBD的方法工具集得以应用在许多常见的环境中。更重要的是，相关人员需要充分理解基本原理，从而能够根据特定的环境加以调整进而应用。

实现面向对象的设计需要一种独特的思维，该思维在硬件计算模型中很不直观。本书真正关心的是设计软件时如何思考，特定的表示法和方法学不是本书的侧重点。因此，本书以大量的篇幅探索好的软件设计的思考过程，甚至故意提供了一些不好的初步设计，用以证明该方法的自我纠正能力。

为了获得这样的理解，有必要描述软件开发的传统方法（面向对象方法之前）在某些方面如何失败，以及面向对象范式如何改正这些缺点。尽管结构化方法为1970年之前软件开发的混乱带来了切实的秩序，但它不是万能的。到20世纪80年代，软件明确显示，严重的可维护性问题仍然存在，这些问题正是面向对象范式要解决的。

同样，如果不讨论一种方法学的某些基础理论，那么就没有办法描述该方法学是否能够很好地工作。然而，这是一本软件开发人员写给软件开发人员的书，因此本书有意识地使用了不具有数学严谨性的实践术语来描述理论问题。

因为本书的主要内容是抽象OOA（Object Oriented Analysis，面向对象分析）模型的建立，因此书中没有很多OOPL（Object Oriented Programming Language，面向对象编程语言）代码。从方法学的名称顾名思义，其重点在于抽象建模而不是编写传统的源语言代码。实际上，当一个具有“转化质量”的OOA模型开发完成后，模型就是代码。换句话说，OOA建模使用的表示法是一种扩展的UML符号，其中增加了符合MDA的抽象动作语言（Abstract Action Language，AAL）的内容。表示法是4GL（Fourth Generation Language，第四代语言）[⊖]而不是3GL（Third Generation Language，第三代语言），但是模型像任何一个3GL程序一样可执行。模型是独立实现的，它是一个针对功能需求解决方案的完整、准确而清晰的规格说明。

我们需要指出的最后一点是，作者的实践开发经验是以几十年而不是几年来衡量

⊖ 计算机语言根据其历史革新和抽象程度被大致划分为几代。第一代语言是机器码，直接在硬件寄存器中设置二进制数字。第二代语言为类似于指令指示的这类事物引入了符号名称。第三代语言是在重用过程、块结构以及基于堆栈的范围等概念方面的重大进步。3GL代表了能够书写的程序在规模上的巨大进步，控制了软件开发半个世纪。第四代语言通过提高抽象层次突破了前几代的计算机语言，程序可以通过一种基本上独立于特定计算环境的方式进行描述。

的。尽管本书的重点在于解释为什么这样做事情，但它绝对不是一本理论书。本书的基础是在现实世界中行之有效的方法。

路线图

本书的主题限定于OOA层的应用开发，主要分为三部分。第一部分回顾了历史并介绍了基本的面向对象的原则。引言部分包括对结构化开发问题的讨论，这些问题正是OO范式想要解决的。第二部分关乎为问题的解决方案构造一个静态的结构。这一部分描述了OO范式与其他方法的最大不同，主要表现了OO范式在问题域抽象方面的独特视角。第三部分描述了解决方案的动态方面，尤其是积极地使用有限状态机进行行为描述。

读者对象

本书的主要受众人群为具有较少OO经验的人。本书假定读者具有一些粗略的UML[⊖]知识。本书还假定读者具有一些软件开发经验，接触过类似C语言的课程项目等。这里假设的经验层次包括：基本具备有关计算机和编程的一些知识，基本熟悉一些常见的缩略语，例如KISS[⊜]等。

第二类读者是从传统的程序开发环境向OO范式转变的大量人群。他们中的许多开发者直接跃进到使用面向对象的编程语言写程序，因为他们已经具有了丰富的编程经验（即，相信如果一个人已经了解了一种编程语言，就能了解全部的编程语言）。可悲的是，这样的开发者往往会开发出大量不好的OO代码，因为没有人告诉他们为什么面向对象的分析和设计与结构化的分析和设计之间存在着巨大的差异。如果你是其中之一，那么你需要忘掉之前所学过的所有关于设计软件的内容，从零开始。

转化的作用

MBD的一个主要特点是，它是基于转化（translation）的一系列方法之一。也就是说，MBD是一种这样的方法，在该方法中，一个解决方案使用某种表示法（例如UML）抽象为模型，然后使用一个完整的代码发生器将模型自动转化为一种实现。转化具有一些明显的优势，因为它代表了计算领域的一种自动化的逻辑扩展，能够提高

⊖ 这并不是非常重要，因为书中会解释语法的含义。除此之外，像Kendall Scott的《UML Explained》（Addison-Wesley，2001）等书物美价廉且易于阅读，这些书可以提供远多于在MBD设计中所需要的UML信息。

⊜ 保持简洁、直接。

生产力、实现规模经济并增加可靠性。缺点在于这并不容易做到。模型编译器面临的优化问题比3GL编译器面临的优化问题更加复杂。无论如何，当前已经存在一些商业代码生成器能够为基于转化的方法提供100%的代码生成。

尽管大部分转化方法出现于对象管理组织（Object Management Group，OMG）成立之前，由OMG所规范的模型驱动架构（Model-Driven Architecture，MDA）还是极大地推动了这些方法。MDA为即插即用工具提供急需的标准，为生成完整的代码提供概念框架。从一个抽象的、独立于实现的模型获得第三代语言代码或者程序集是一项重要的工作，特别是对当前复杂的IDE（Integrated Development Environment，集成开发环境）来说。这项工作大大受益于MDA的概念及其形式化。

然而，MBD并不依赖于转化。MBD中产生的模型与传统开发中OOA产生的模型本质是相同的，可以通过手工的方式生产OOD模型和3GL代码。MBD模型的构建只是比典型的OOA模型更加严格，因为代码生成器是没有想象力的，它们处理命令中的内容，而不是暗示的内容。开发者必须手工实现转化。

致谢

非常感谢Steve Mellor和Sally Shlaer的工作，他们的方法是MBD的基础。他们是软件开发转化方法的开创者，并为OOA模型提供了急需的设计严谨性。除此之外，Steve Mellor是我见过的最好的OO建模者，他提供的例子让人受益匪浅。

同样感谢Rebecca Wirfs-Brock对手稿一丝不苟和悉心的审校。

Pathfinder Solutions为本书的方法提供了优秀的试验场，Greg Eakman、Carolyn Duby和Peter Fontana提供了特别的支持。

还要感谢Addison-Wesley出版社的Chris Guzikowski和Raina Chrobak的耐心与信心。谢谢Elizabeth Ryan、Diane Freed和Chris Zahn的编辑工作。

尽管这本书于我退休之后开始撰写，我还是要感谢Teradyne/ATB公司，他们提供了世界一流的软件开发工厂，其中充满了明星开发人员，他们为本书中的许多想法提供了原型。

最后，我要感谢30多年以来因特网上的大量用户为本书中概念的解释提供的共鸣版。读者读到的清晰描述大部分精炼自公共论坛上的描述。

引　言

历史是一场噩梦，我试图从中清醒。

——乔伊斯

40年前，百万行的程序被认为是巨型程序，仅供美国国防部（Department of Defense，DoD）[⊖]内部巨大的主机系统使用。按常规估计，开发这样的程序需要1000名工程师工作10年。今天，大多数PC上的应用都超过百万行，并且有许多在千万行范围之内。而且，人们期望它们能在几年甚至更短的时间之内开发完成。因此，在程序规模不断增长的同时，客户期望开发费用能越来越少。

在高科技日新月异的当今社会，软件快速老化已经成为常态。在公司的经济发展模式中，增长的要求比以往任何时代都更受重视。诸多因素促使公司在短时间内创造出更多、更好、更与众不同的产品。由于这些新产品的研发制造都需要依赖日渐庞大的各式软件，因此软件开发人员完成工作、缩短上市时间的压力不断增大。

软件开发人员同时奋战在软件开发的另外一条战线上。40年以前，对于发布的代码，全美的平均缺陷率为150个/KLOC[⊜]。但是该数据没有人关注，因为大好形势之时，没有人去研究如何写“好的”软件。GOTO语句和全局数据占主流地位，设计在黑板上或者鸡尾酒餐巾上完成，DEL键永远是程序员们的最爱。软件开发如同一些电气专业一样晦涩难懂，因此，坦率地说，那时没有人在意这些。

但是，这样一个乌托邦式的情况不可能永久持续。在20世纪70、80年代发生了两次变革。第一，一场由日本方面开始的质量革新出现，亚洲其他国家纷纷效仿，随后缓慢波及到西方工业国家。此次变革使得用户进入了一个新时代，他们不用再把每周六分之一的时间花在修理车子或者电视上，这才是用户真正喜欢的。第二，在此次质量革新之后，软件的影响逐渐渗透到生活的方方面面，突然间软件就成为经常出错的典型。现在用户逐渐意识到了还有更好的质量革新方式，于是对于这些经常出错的软件，他们不再喜欢了。

因此，要求软件开发人员在更短的时间之内开发规模更大的程序，同时还要求将

⊖ 这是一本有关软件开发的书，所以我们必须使用缩略语，这是一个普遍的规则。

⊜ 如果不知道这是什么意思，那你需要先去学习比这本书更基础的书籍。这个统计数据是指以千行计的原始COBOL代码的统计，COBOL是相当啰嗦的语言。

缺陷率控制在5西格玛的范围之内[⊖]。更大的挑战是，20世纪80年代市场流行的口号是“质量是免费的”。虽然每一位软件开发人员都知道这是一派胡言，因为软件质量与开发时间是此消彼长的一种平衡。但是，市场压力的影响远远大于软件开发人员的影响，没有人听到软件开发人员的呐喊。因此20世纪80年代对于“软件危机”一词更新更深层的含义是以软件开发人员的健康为代价的。

第二个重大事件是关于计算的技术大爆炸。以前技术进步的主要特征是新的语言或者操作系统的出现。但是PC软件数量的不断增长，程序互操作性的需求和万维网的出现，促使计算技术创新不断[⊜]。从以架构策略为基础的计划工作到测试过程，现在的软件开发人员在方方面面都有多种选择。更夸张的是，当开发人员还在熟悉旧技术的时候，新的技术已经出现了，速度之快令人咂舌。

最终，软件开发人员面临着不断产生的需求变更。计算域不是唯一面对创新和变革的领域。美国联邦会计标准局（Federal Accounting Standards Bureau，FASB）的核心管理信息系统IRS以及各式各样的其他美国政府机关不定期在颁布变更指令。产品的激烈竞争迫使市场支持者不断推陈出新，以保持与竞争对手步伐一致。同样，在学术界，标准、技能和工程学科的技术方面也日新月异，逐渐摆脱混乱形成统一。“需求万年不变”和项目瀑布模型的日子一去不复返了。

总之，现代的软件开发人员都面临着一个古老的问题，即：如何把5磅的东西尽快放到一个容量仅3磅的袋子里，并保证没有任何遗漏。今天，和半个世纪前相比，我们有着更好的工具、方法、过程和技术，但问题也是成比例增长的。基本汇编语言（Basic Assembly Language，BAL）要解决的软件危机问题仍然如影随形，这也是本书的第一条忠告。

软件开发是一个不可能完成的任务，开发人员唯一要做的就是不断应对并保持微笑。

1. 研究现状

1995年的某次会议上，主讲人要求当时的与会人员中使用微软Windows操作系统的人员举手示意，数百人举起了手。然后，他要求过去一周内系统崩溃过的人再次举手，大半以上的人仍然举手。接下来，他让那些认为微软在将来会被市场淘汰的人继续举手，结果所剩寥寥无几。该主讲人用此次实例来说明和验证自己的说法，那就

⊖ 假设每一行代码有出现一个或者多个缺陷的可能性，5西格玛代表缺陷出现的频率是五个标准差，中值为每行代码一个缺陷。这个质量级别对应的缺陷率是每100万行230个缺陷，比20世纪60年代的标准高很多。

⊜ 有趣的是，许多与万维网相关的核心技术其实是老技术的再生。我最近读到一篇文章，发表在一份相当著名的软件开发杂志上，作者声称第一种标记语言发明于1986年。实际上，标记语言的应用比这早了20年。可悲的是，这说明业内并不了解为什么标记语言在20世纪70年代末被遗弃。但是，这是另外一本书的主题。

是，软件质量不是一个事关公司生存的必要条件。

这一结论在今天很难保证是对的，尤其是在微软与 Linux[㊀]的市场大战中可见一斑。决定一个软件公司在市场中生存的关键因素，其实是他们是否能够提供用户想要的东西。从广义上讲，这一点可大致从产品的可靠性、功能性、易用性和可用性（即市场时效性）四个方面评估，当然开发人员花费在这四个方面的成本是不同的，如图 I-1 所示。用户总是希望各方面都得到满足，但是毕竟预算有限，因此供应商决定综合的总成本，然后由市场来检验用户的满意度。

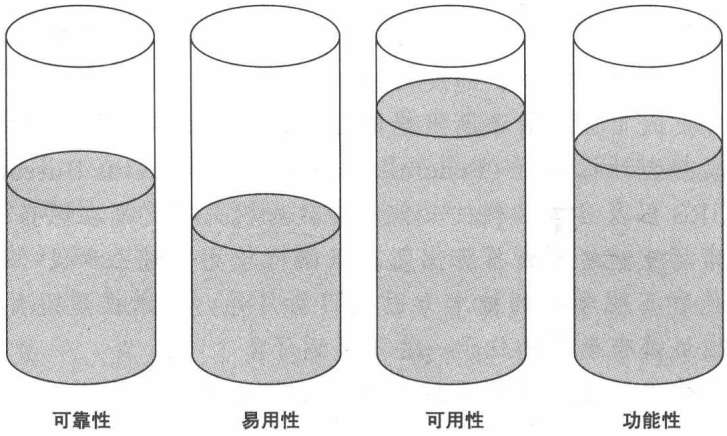

图 I-1　权衡冲突的开发优先级

如何满足这四个方面基本上是一项市场决策。在一个既定市场给定价格下，市场支持者总是期望通过一个最佳营销战略来赚取他们的第一桶金。而成本最小化的关键在于软件开发人员。这种开发成本的降低会直接影响产品市场竞争优势的确立。这种优势的价值将取决于特定的市场地位。因此，衡量软件开发现状的问题是：运用什么样的工具才能有效降低成本且满足这四个方面的要求。

我们有各种各样的软件工具来帮助我们，如版本控制系统。我们也有大量的过程，从 XP 到正式方法。我们还有很多种软件设计方式方法。我们有像 RUP[㊁]和 CMM[㊂]这样的过程框架，还有汇编语言、集成开发环境（IDE）、度量系统、估计模型和其他各种各样的工具。

我们有各种珍贵的数据，可以整理出可用的方法、技术和实践经验，形成最佳的组合。

一帮黑客宅在他们的卧室里，经过数月消耗掉无数可乐和披萨后黑掉某一应用的

㊀ 实际上，微软吸取了教训。现在微软产品的可靠性已经成为一种竞争力。

㊁ RUP（Rational Unified Process）不是真的过程，它主要是一个过程框架，能够根据自身的开发环境进行高度裁剪。

㊂ CMM（Capabiliity Maturity Model）是一个真正的过程框架，它描述良好的软件必须具备什么样的功能，而不是如何实现。因此，符合 CMM 是良好软件开发的一个必要非充分条件。

日子一去不复返了。如果工作面试时，你告诉招聘经理去年你总共写了100KLOC，那你可以跟这份工作“拜拜”了。因为招聘经理都明白效率如此低下的原因是编写的代码最后都不可靠，并且是一堆无法维护的烂摊子。有效降低满足这四个方面的成本，需要的是有组织、有系统地进行软件开发。

目前业界有太多口惠而不实的软件工程。作为一门工程学科我们可能还需要走很长的路。软件开发是一门艺术，尽管非常晦涩难懂。其精髓在于这是一个在给定环境下运用各种不同方法和工具的大杂烩，即一个协调统一的开发系统。除了提供这种集成系统的规则外，我们的“软件工程”只不过是“这里有一个框架和一些可选择的组合。寻找正确的候选者并在特定开发环境中把它们恰当地组合在一起，则是留给学生的练习。”

2. 什么在起作用

然而现实情况是，软件开发的艺术在过去的半个世纪里有了长足的发展。同样数目的开发人员能够在更短的时间内更高质量地完成更大规模的项目，因此我们更应该去做一些更有帮助的事情。其中棘手的问题是，如何弄清哪些事情是正确的。

早在1970年之前，人们就有基于几十年实践经验总结的宝贵教训。例如，曾几何时FORTRAN的GOTO语句和COBOL的ALTER语句被视为处理复杂编程问题的强大工具，然而10年后，编程经理不再使用这些语句，因为使用这些功能的程序在需求变更时很难维护[⊖]。

所有这些来之不易的教训形成了**开发者经验**的概念。当一个程序员犯过足够多的错误并在错误中吸取了各种教训后，他就会逐渐成长为软件开发的行家。遗憾的是，软件开发人员总是会重新经历这些错误，因为没有一个关于这些软件开发经验教训的系统总结或者统一标准。

20世纪60年代末，第三代编程语言中出现了各种各样的编程方法，软件设计方法开始兴起。早期，大家只是把一些经常犯的错误和实践经验公布于众。不过，很快软件专家们开始将这些实践经验总结成系统的设计方法学。这些方法学在细节上各有不同但是很多特征是有共同点的，所以它们都属于结构化开发（Structured Development，SD）。因为它们是相当抽象的视图——有关软件设计的一张大图，而且读者会发现它能够很方便地使用专门的表示法。很多倡导者也很学术化，表示法往往倾向于图形化，这使集合论和图论中的理论约束也得以应用。

结构化开发是20世纪中期最简单也是最重要的软件开发成果。毫不夸张地说，

⊖ 问题在于这些语句以“动态”的方式改变了程序的控制流，但是在代码中无法直接可见。当顺序和分支都明确时，去了解其他人的代码是很困难的。因此这种程序导致一个绕口的简写，即WORN，一旦写出，永不可读（Write Once；Read Never）。

它让软件开发摆脱了杂乱无章的状态。除此之外，它是数据处理的增长动力，数据处理在20世纪80年代演变为重要的信息技术。在企业界，很多入门级的COBOL程序员艰难地做出了大量的报表，从而为日常的企业运行提供了前所未有的可视性。更好的是，这些报告通常相当准确。

当然，结构化开发在软件开发的漫漫长路上仅是第一步。正如所预期的，它是新的，但不是万能的，从20世纪70年代起它的一些缺陷就开始显现了。只有理解这些缺陷是什么，才能理解20世纪80年代面向对象开发的发展原因和发展方式。

目　录

第二部分　静态模型