IT Trend

Addison Wesley

Design by Contract 原则与实践

Design by Contract by Example

Richard Mitchell
Jim Mckim 著

孟 岩 译

人民邮电出版社
POSTS & TELECOM PRESS

IT Trend 系列丛书

Design by Contract 原则与实践

Richard Mitchell Jim Mckim 著

孟岩 译

人民邮电出版社

IT Trend 系列丛书

Design by Contract 原则与实践

◆ 著　　Richard Mitchell　Jim Mckim
译　　孟　岩
责任编辑　陈冀康

◆ 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　电子函件　315@ptpress.com.cn
网址　http://www. ptpress.com.cn
读者热线　010-67132705
北京汉魂图文设计有限公司制作
北京朝阳展望印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：720×980　1/16
印张：17. 5
字数：324 千字　　2003 年 10 月第 1 版
印数：1-4 000 册　　2003 年 10 月北京第 1 次印刷

著作权合同登记　图字：01 - 2003 - 6649 号

ISBN 7-115-11570-2/TP • 3586

定价：33.00 元

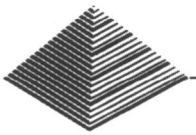
图书在版编目(CIP)数据

Design by Contract 原则与实践/（美）米切尔（Mitchell,R.），（美）麦吉姆（Mckim, J.）著；孟岩译. 北京：人民邮电出版社，2003.10
ISBN 7-115-11570-2

Ⅰ. D… Ⅱ.①米…②麦…③孟… Ⅲ. 软件开发 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2003）第 080309 号

版 权 声 明

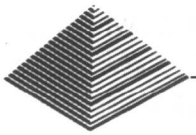


内容提要

本书通过 Eiffel 和 Java 实例，阐述了“契约式设计（Design by Contract）”的思想原则和应用方法，帮助读者掌握如何在面向对象软件设计过程中应用契约。

全书分为 12 章。通过实例探讨了 DbC 的基本概念，引入了 DbC 的六大原则和六大准则，展示了六大原则的应用，分析了 DbC 和继承的关系，介绍了框定规则、DbC 的好处和限制、观察者框架中的契约和先验条件检验；最后给出了一个 Java 范例，并探讨了契约在分析模型中的应用。附录部分是译者专门为 C/C++程序员撰写的，提供用 C/C++进行契约式设计的指导和建议。

本书适合面向对象软件工程师和面向对象语言程序员阅读。



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*）

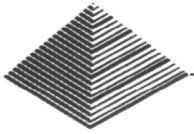
《应用MDA》（*Model Driven Architecture：Applying MDA to Enterprise Computing*）

《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　　序

本书是国内翻译出版第一本以“契约式设计（Design by Contract）”为主题的技术图书。本书的两位作者在契约式设计领域耕耘多年，颇有心得。这本书虽然篇幅不大，内容却十分丰富，技术含量颇高，同时又好读易懂；只要学习过一门面向对象语言，并对面向对象思想略有了解，就不难从本书中汲取丰富的养料，从而大幅提高自己的设计能力和代码质量。

本书的核心内容是契约式设计的六大原则和六大准则，而全部讲解均通过范例展开，实践充分，说理透彻，材料组织可谓出色。看得出来，本书是以多年实践和教学的丰富经验为基础的。作者学风严谨，文风朴实，难怪契约式设计之父 Bertrand. Meyer 在前言中给这本书那么高的评价。

关于契约式设计的意义及技术原则，书中有大量的介绍，我不打算在这里画蛇添足了。我想特别强调的一点就是实践。这本书的大部分例子都是用 Eiffel 写的，虽然我相信读者很容易就可以理解代码的含义，但是如果仅仅停留在理论知识上，没有亲自动手实践，则对于契约式设计的掌握一定会流于形式。读者不妨下载一个 Eiffel 编译器，比如 SmartEiffel，或者出身名门正宗的 ISE Eiffel Studio，通过实践来感受契约式设计。如果你是一个 Java 程序员，则应该先翻翻本书的第 11 章，学习 iContract 的用法，然后用这个工具来配合本书的学习。C++程序员学习契约式设计的困难会更大一些。考虑到这一点，我特别为跟我一样同处于 C++战壕的朋友们撰写了一篇文章，讨论了一些在 C++中进行契约式设计的可能方案，附在本书正文之后。

事实上，译者在实践中感受到，契约式设计中最难的地方是“度”的把握问题。正所谓“过犹不及”，过度地使用契约反而会导致程序更加复杂冗长，带来更多的问题。至于如何把握这个“度”，缺乏足够的实践也就难以有深刻的认识。我想读者如果能够亲自尝试，把契约思想运用到自己的实际工作中，则不单能够提高自己的工作水平，同时也能丰富国内契约式设计社群的经验。

本书原文的语言相当直白浅显，照理说翻译工作应该比较轻松；但实际过程却比预想的艰苦。一方面我痛感自己的技术根底和文字表达能力的不足，无法把

原文的意义简洁准确地表达出来，有些地方翻译得不尽人意。另一方面，讲述契约式设计的图书是第一次引入，翻译上没有前人的经验可以借鉴，加上 Eiffel 学派的很多术语都跟国内开发者所熟悉的 C++、Java 习俗大相径庭，因此不少地方虽然做到了忠实原文，但译文却难免显得有些生涩。除了请读者谅解之外，也希望大家能够批评指正，帮助我不断完善这本译作。

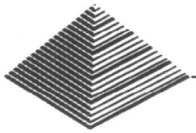
关于这本书中的术语翻译，有一个特别的问题，在译稿还在修改的过程中就已经引起了一些讨论，那就是对于precondition和postcondition的翻译。本书将这两个术语分别翻译成了“先验条件”和“后验条件”，不少师友对此提出了异议。特别是裘宗燕教授，在阅读完译稿之后，曾对译者郑重提出意见，希望把这两个术语的翻译修改为“前条件”和“后条件”。此外，还有一些网友指出，“先验”这个词有其固定的哲学含义，即“先于经验的”，本书这样使用“先验”二字，可能引起歧义。诸位老师与朋友的批评我衷心接受。其实，这里的“验”，有“检验、验证”的意思，指的是执行相关的操作的前后需要验证条件是否成立。译者认为契约式设计之所以有效，不仅在于“有法可依”，而且在于“有法必依”，而之所以能够做到有法必依，正是凭借了这个“验”字。有条件而不检验，则只不过是难看的注释。因此，权衡良久，译者大胆保留了原来的翻译。契约式设计刚刚被引入国内，术语规范还没有形成，因此术语的翻译从一定程度上反映了译者对于相关技术的理解，也算是“一家之言”。欢迎各位读者通过各种途径和我探讨这个问题，也希望通过我们的探讨能够进一步促进术语的规范和统一。

译者将持续关注契约式设计思想的发展，并愿意就此与广大读者朋友密切交流，相互学习。同时，对于此书翻译的问题，我也愿意与读者充分交流。我的电子邮件信箱是 my518@263.net。

藉此机会感谢我的家人、师长和朋友们，特别是本书责任编辑陈冀康先生，他的认真负责使本书的质量获得了很大的提升。裘宗燕教授在百忙之中阅读了本书大部分译稿并提出了宝贵意见，我更要表示深深的谢意。

孟岩

2003 年 7 月于北京



序

提出契约式设计（Design by Contract）[1]这一概念，是为了给从事实际开发工作的一线程序员提供一种方法，以便在尽可能小的代价下开发出可靠性出众的软件系统。契约式设计的基本思想至为简单，然而其应用却可以贯穿于软件创建的全过程，从分析到设计，从文档到调试，甚至也可以渗透到项目管理中。

契约式设计的思想根植于正规的（也就是数学形式的）软件构造分析方法，然而完全理解完整这种正规化形式并不是有效应用契约式设计的必要条件。实际上，它非常容易。正如 James McKim 所说："只要你会写程序，你就会写契约。"尤其是当编程语言和环境能够给予你直接支持时，效果几乎是立竿见影的，Eiffel 就是一个很好的实证。本书的作者正是使用 Eiffel 来介绍新的概念，然后用 Java 和其他编程语言来对这些概念进行补充说明。

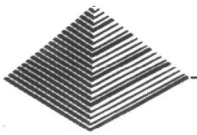
Mitchell 和 McKim 对契约式设计做出了清晰、简练而有力的阐述，并且展示了大量的例子和模式，包括分析和实现步骤。本书中所讲授的经验和思想，来源于两位作者连续多年向数届学生和专业人员教授契约式设计思想的授课经历，也来源于他们面对新问题坚持不懈的应用实践。

这本书能够让人读之欣然，同时，无论您的教育状况、实践经验、编程风格和应用领域如何，都能够从中有所收获，成长为更优秀的程序员。像这样的书，就我所知，实在是凤毛麟角。

Bertrand Meyer

---

[1] 译注：Design by Contract™ 是 ISE 公司的注册商标，目前尚无公认的中文译法。直译为"按契约设计"，近乎口语，术语味道较差，不利于交流和对知识产权的尊重。所以本书使用"契约式设计"这一译法。



前　言

本书的主要内容

契约式设计的本意很简单，就是在设计和编码阶段向面向对象程序中加入断言（assertion）。而所谓断言，实际就是必须为真的假设，只有这些假设为真，程序才可能做到正确无误。契约式设计的主要断言包括**先验条件**(precondition)、**后验条件**(postcondition)以及**不变式**(invariant)：

- 先验条件针对方法(method)[1]，它规定了在调用该方法之前必须为真的条件。
- 后验条件也是针对方法，它规定了方法顺利执行完毕之后必须为真的条件。
- 不变式针对整个类，它规定了该类任何实例调用任何方法时都必须为真的条件。

断言应使用某种编程语言写在代码里（而不是仅仅通过文档加以声明），只有这样：

- 对于程序员而言，它们才有意义，同时能够提供优质的、有用的文档。
- 能够在运行时对它们进行检测，更好地支持测试和调试工作。

本书主要说明如何才能写出好契约。关于这一点，本书列出了 6 条基本原则和一些准则，并通过示例展示了这些原则和准则的应用。

只要学完前 3 章，你就具备了撰写高质量契约的技术能力。其后的章节会帮助你进一步提高。

本书大部分章节讨论如何为单个的类开发契约，而其他章节则讨论与继承相关的契约，以及有关框定规则（frame rule，对于不变条件进行断言的契约）的话

[1] 译注：这里的方法(method)是面向对象语言的术语，来源于 Smalltalk，在 C++中通常叫“成员函数”。不过读者可以从本书后面部分了解到，Eiffel 并没有把 method 作为一个显式概念，而是通过 feature 来抽象 method 和 public data member，这就是 Eiffel 所谓的访问一致性原则（Unified Access Principle）。

题。书末尾处有两个更大的示例，示范了涉及多个类的契约。第 9 章讲的是 GoF 提出的 Observer 模式（Observer pattern，Gamma et al. [1994]）。第 10 章介绍了一个小的应用程序，为了服从一个核心的部分规则，我们特别在用户接口中设计了一个对象。第 12 章讨论了契约在系统分析中的运用。第 8 章总结了使用契约的优点，并将契约式设计与防御式编程(defensive programming)[1]进行比较。第 11 章探讨了如何将契约及接口联系起来，并简要探讨了如何在分布环境中实现契约。

编程语言

书中所列举的例子都优先采用面向对象语言 Eiffel 实现。我们选用这种语言是出于 3 个原因：

1. Eiffel 直接支持契约，所以很方便在示例中说明概念。

2. Eiffel 易读，因而是一种很好的伪码，你可以用任何一种面向对象编程语言来实现它的思想。

3. Eiffet 编译器已经相当成熟和商业化了，因此我们的契约就可以发挥它们预期的规范和测试作用。契约是对类的规范，它精确描述了该类应该提供的服务。契约中的断言则在运行期间测试，以确保正在执行的代码确实合乎规范。

你不必为了理解这些例子而专门去学习 Eiffel 语言。相反，我们相信你一定可以在自己的编程环境中运用本书给出的原则。我们所提出的问题以及给出的建议并不局限于 Eiffel。

我们的确用 Java 重写了两个例子，用到了一个叫做 iContract 的预处理器，这个工具给 Java 提供了契约式设计能力。我们在例子里探讨了一些在 Eiffel 中不会直接碰到的问题，从而用另一种语言向大家展示了契约的运用。

读者对象

本书适合于任何想了解和运用契约式设计的读者。我们希望这本书能对专业人士、学生（特别是前几章）、教师和研究人员有所裨益。

人贵有自知之明，我们知道这本书不是那种能让您在冬天的火炉旁，或夏天的游泳池边悠闲阅读的书。本书的内容必须经过仔细研究才能被理解，最重要的

---

[1] 译注：防御式编程是一项经常容易与契约式设计混淆的技术。简而言之，就是对可能错误的条件实施监测，一旦发现错误，则主动将程序恢复到“正确”状态。此技术争议很大。

是实践。希望你已经有了一个支持契约的编程环境，譬如 Java 编译器和 iContract 工具（更多信息参见参考资料）或者 Eiffel 编译器（同样，更多信息参见参考资料）。

我们并不打算教授面向对象编程，而是假设你已经知道如何使用某一门面向对象的编程语言。为了让使用 Eiffel 和 Java 之外的编程语言的读者也能理解示例，我们尽可能对用 Eiffel 和 Java 写的代码进行了详尽说明。

风格

本书是建立在示例的坚实基础之上的。通常，每一章都以一个简单的例子作为基础。这就意味着经常有大量的代码需要阅读。但是，绝大部分代码都是断言，定义了该程序段要实现的目标。这个级别的代码通常比定义如何实现目标的代码要简单易懂。此外，为了使讨论更顺畅，我们通常会同时分析几行代码。

书中大部分例子都是很简单的。例如，在第 1 章中，我们不是去为一个完整的顾客管理组件撰写契约，而是为一个查找表（或叫字典）撰写契约，而这个查找表正是该顾客管理组件的基础数据结构。这样一来，读者就不至于被太多的细节弄得晕头转向，从而忽视基本原则。一旦你注意到了这些原则，我们相信你就能够将它们运用到更复杂的实际案例中去。

对本书描述的内容，我们进行过精心筛选。有许多其他书籍对契约式设计这一话题进行了有益的、富有见地的阐述，而我们则聚全力于一点（也是本书区别于其他书籍的一点）：提出建议，帮助读者撰写良好的契约。另外，本书当然也不是契约式设计的“终极圣经”。在契约撰写、底层技术和底层理论、契约思想的拓展以及实际价值评估等方面，还有很多工作可做。

网站

本书有一个配套网站：http://www.awl.com/cseng/titles/0-201-63460-0/。这个网站上有书中示例的源码。我们希望你能够下载这些源码，并加以运行。改变实现和契约的代码，掺入一些 bug，然后观察发生的现象。改写例子，做试验，把它们运用到实际项目中，这就是我们学习契约的方法。

致谢

本书曾得到很多人的帮助。首先应该感谢 Bertrand Meyer。Bertrand 把契约式

设计和 Eiffel 推荐给我们使用，我们从中得到了极大的乐趣。他还鼓励我们研究契约式设计，并建议我们把心得编著成书。

Reto Kramer 开发了 iContract 工具，使我们能够用 Java 实现我们的例子。Reto 还帮助我们把用 Eiffel 实现的例子改写为 Java/iContract 的，这些我们在第 11 章中进行了讨论。Chris Thomas 开发了一种能够在 Eiffel 的断言中添加限定词的原型预处理器，我们在第 7 章中就采用了这个预处理器。

同许多契约式设计领域的人士的共事过程使我们受益匪浅，我们也从本书样稿的评审人员那里得到了许多非常有帮助的意见和建议。我们想对以下这些人表达谢意：Jim Armstrong、Vladimir Bacvanski、Richard Bielak、Roger Browne、Christopher Creel、Roger Duke、John Howse、Ali Hamie、Brian Henderson-Sellers、Jean-Marc Jézéquel、Ian Maung、Christine Mingins、Agnes Mitchell、David Mondou、Roger Osmond、Bob Streich、Chris Thomas、Ted Velkoff、Richard Wiener、Jean-Francois Zubillaga，以及不知名的审阅者。如果我们在这个名单中漏写了任何人，请相信这是由于我们糟糕的记性造成的疏忽，原谅我们。相信我们确实对你满怀感激。

我们同样感激下面这些机构：德克萨斯州奥斯汀的 InferData 公司、康涅狄格州哈特福德的 Renssalaer、澳大利亚墨尔本的莫纳什大学以及英格兰的布莱顿大学，他们在本书的撰写过程中提供了帮助。

我们开发示例所用的 Eiffel 编译器是由 ISE 公司（交互式软件工程公司）捐赠的，为此我们也要谢谢他们。

另外，我们还要向家人道歉，我们在写书时占用了大量时间，而没有履行作为丈夫和父亲的职责。谢谢他们的宽容：Agnes、James、Emma、Daniel 和 Julia Mitchell，以及 Mary、Krista、AJ 和 Alex McKim。

最后，我们还要相互致谢，因为我们都是对方的非常令人愉快的、富有成效的工作伙伴。

Richard Mitchell 和 Jim McKim

IT 趋势系列图书

As technology continues to change, we will remain on the cutting edge.
只要科技在发展，我们始终在前沿

什么是 MDA?

Model Driven Architecture（MDA）是 OMG 提出的新的方法学。它是一种基于 UML 以及其他工业标准的框架，支持软件设计和模型的可视化、存储和交换。和 UML 相比，MDA 能够创建出机器可读和高度抽象的模型，这些模型以独立于实现的技术开发，以标准化的方式储存。因此，这些模型可以被重复访问，并被自动转化为纲要（schema）、代码框架（code skeleton）、测试工具（test harnesse）、集成化代码以及各种平台的部署描述。MDA 把建模语言用作一种编程语言而不仅仅是设计语言。

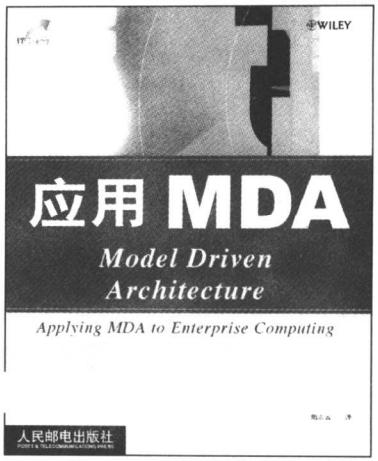
MDA 以一种全新的方式将 IT 技术的一系列新的趋势性技术整合到一起。这些技术包括基于组件的开发（Component-Based Development）、设计模式（Design Pattern）、中间件（middleware）、说明性规约（Declarative Specification）、抽象（abstraction）、多层系统（multi-tiered system）、企业应用整合（Enterprise Application Integration）以及契约式设计（Design by Contract）。MDA 的出现，为提高软件开发效率，增强软件的可移植性、协同工作能力和可维护性，以及文档编制的便利性指明了解决之道。

MDA 被面向对象技术界预言为未来两年里最重要的方法学。

《应用 MDA》

David S. Frankel 著　　鲍志云 译

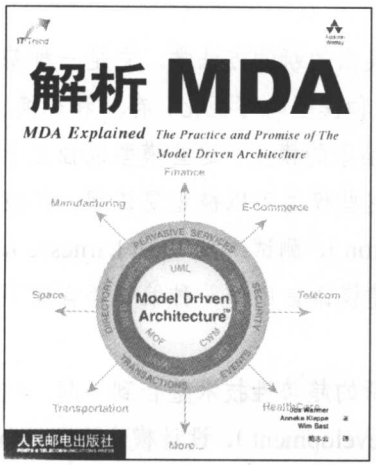
2003 年 10 月出版



David S. Frankel 是 OMG 体系结构委员会成员，负责 MDA 的研发工作。

《应用 MDA》深入描述了 MDA 的概念、关键技术和所有特性，包括 UML 和 MDA 的关系，与 MDA 相关的 MOF、XMI 和 CWM 等等。本书还通过具体实例，帮助你理解如何将 MDA 应用到你的企业系统中。

IT 行业的项目经理和系统架构师应该通过《应用 MDA》一书来学习和了解这一新的技术。在未来的几年里，《应用 MDA》将成为 MDA 学习者的“宝典”。



《解析 MDA》

Jos Warmer 等 著

本书的三位作者是 OMG 所属的 MDA 标准委员会的成员。

《解析 MDA》深入剖析了 MDA 的优越性和实现方法。

开发人员将通过《解析 MDA》掌握定义 MDA 工具的必备知识。

什么是 DbC?

Design by Contract（DbC，契约式设计）是面向对象软件大师 Bertrand Meyer 对软件构造方法的一个重大贡献，无论是在形式化的数学证明中，还是在实践运用中，都被证明是大幅改善软件工程质量的有效手段。该方法在 Eiffel 编程语言中获得直接支持，并且可以通过辅助工具在 Java 语言中运用。

《Design by Contract 原则与实践》

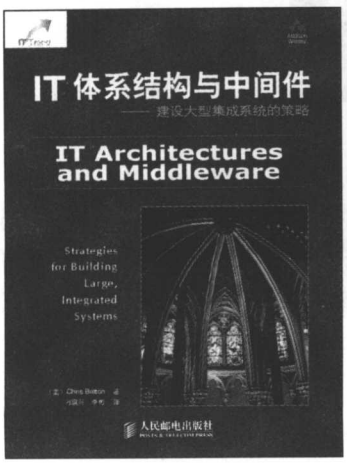
Richard Mitchell　Jim Mckim 著　孟岩 编译

2003 年 10 月出版

本书凝聚作者多年的实践和教学经验，通过 Eiffel 和 Java 的实例阐述了 DbC 的主要原则，并教读者如何实践这些原则，撰写优质高效的契约。译者孟岩还特别撰写了《契约式设计的理解及其在 C/C++中的应用》作为本书的附录，讨论了 DbC 在 C/C++语言中的应用。

什么是 Middleware?

从软件的不同视角出发，对中间件（middleware）会有不同的理解。在众多关于中间件的定义中，被普遍接受的是 IDC 的表述：中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源，中间件位于客户机服务器的操作系统之上，管理计算资源和网络通信。事实上，随着计算机技术的不断发展，中间件的概念和范畴在不断更新，而中间件技术的应用也几乎无处不在。



《IT 体系结构与中间件

——建设大型集成系统的策略》

[美] Chris Britton 著　刁联旺　李彬 译

2003 年 6 月出版

随着电子商务和分布式集成系统所带来的新需求，要求 IT 系统能集成公司的各种应用，以支持更快、更精确的业务流程，并提供信息共享。本书结合在现实中碰到的各种挑战，精辟讲述 IT 体系结构与中间件技术，有助于你在新业务目标、新技术和商家战争中独具慧眼，清楚地思考问题，并有效地面对挑战。