HZ BOOKS 华章IT

ISTE WILEY

架构师书库

软件架构

[法] 穆拉德·沙巴纳·奥萨拉赫（Mourad Chabane Oussalah）编著
姚军 译

SOFTWARE ARCHITECTURE 1

从软件架构的概念、发展和最常见的范式入手，详细介绍20年来软件架构领域取得的研究成果

全面讲解软件架构的知识、工具和应用，涵盖复杂分布式系统开发、服务复合和自适应软件系统等当今最炙手可热的主题

机械工业出版社
China Machine Press

架构师书库

SOFTWARE ARCHITECTURE 1

软件架构

[法] 穆拉德·沙巴纳·奥萨拉赫（Mourad Chabane Oussalah）编著

姚军 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

软件架构 /（法）穆拉德·沙巴纳·奥萨拉赫（Mourad Chabane Oussalah）编著；姚军译．
—北京：机械工业出版社，2016.8
（架构师书库）
书名原文：Software Architecture 1

ISBN 978-7-111-54264-3

I. 软… II. ①穆… ②姚… III. 软件设计 IV. TP311.5

中国版本图书馆 CIP 数据核字（2016）第 159521 号

本书版权登记号：图字：01-2014-5464

软件架构

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：曲　熠　　责任校对：董纪丽
印　　刷：北京文昌阁彩色印刷有限责任公司　　版　　次：2016 年 8 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：12.5
书　　号：ISBN 978-7-111-54264-3　　定　　价：59.00 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 ••

从 20 世纪 70 年代起，没有哪一个领域取得的成就能与计算机行业比肩，而在行业的发展中，计算机软件开发方法的变化有着最深远的意义。

毫不夸张地说，我们这一代人无不深受软件行业发展的影响。在学生时代，计算机还只是能够出出算术题的小孩玩具，而如今，软件的影响已经超越了计算机本身，从我们出行时的各种交通工具到日常使用的所有电子设备，无不受到软件的控制。软件无所不在，渗透到各行各业和生活的不同侧面，这也给软件行业带来了新的挑战。如何提高生产率？如何充分利用整个行业数十年来积累的知识？业界的研究人员不断地提出新的工具和新的思想，努力满足日益增长的需求。

软件架构概念的出现，显著地改善了软件工程界的面貌，这一概念脱胎于软件工程学诞生以来提出的各种编程范式与思想，重点在于捕捉复杂软件系统的架构元素，重用过去项目中的既有经验，并解决系列产品中的可变性问题。软件架构真正实现了软件制造的工业化，使软件从最初的实验室产品和人力密集的“手工业”产品，发展成为可以在生产线上制造的高效率产业，从而缓解了互联网、物联网蓬勃发展对软件生产造成的压力。

本书从软件架构的概念、发展以及最常见的架构范式入手，详细介绍了 20 年来软件架构领域取得的研究成果，其中引用了大量经典的研究文献，为我们提供了现代软件架构的完整图景。细读此书，可以帮助你真正了解软件架构的各方面知识，以及其在当今最火热的复杂分布式系统开发、服务复合和自适应软件系统中的应用，从而在设计、开发和理解软件架构上迈出第一步。本书也是浩如烟海的软件架构文献的出色导读，通过它可以对各种各样的架构、工具有初步了解，从而进一步地学习软件架构，提高技术水平。

软件架构的发展日新月异，回顾过去也是为了放眼未来。在翻译本书的过程中，译者深深感受到了科技的力量，常为繁多的新颖想法与概念所吸引，同时担心自己的水平有限，未能完全展现原书的精彩之处。希望广大读者多多批评指教。

本书的翻译主要由姚军完成，徐锋、陈志勇、刘建林、白龙、方翊、陈霞、林耀成、宁懿等也为翻译工作做出了贡献，在此感谢机械工业出版社的缪杰编辑和其他工作人员对翻译及出版工作的大力支持和帮助。

译者

2016 年 7 月

前 言

在过去20年间，出现了多种描述软件架构的语言，促进了以架构为焦点的应用程序开发。一般来说，这些语言提供了描述和分析软件系统所用的正式或半正式标记法。它们通常搭配一些工具，这些工具用于分析和模拟，有时也用于生成已建模系统的代码。软件架构对复杂分布式系统的发展做出了贡献。它们的主要特征一方面在于管理系统的抽象及表达水平的能力，另一方面则是考虑系统结构与行为建模的能力。现在，任何复杂软件系统设计与开发中的关键问题之一都是架构（即组成该系统架构元素的组织），这已经是广为接受的看法。好的架构有助于系统关键属性（可靠性、可移植性、互操作性）的形成。相反，糟糕的架构可能给系统造成灾难性的后果。而且，在开发期间，好的架构可以作为系统的“良心”。实际上，好的架构指导系统的演化过程，例如，它指明了系统的哪些方面可以在不破坏完整性的情况下进行修改。

近年来，新设计的应用程序，尤其是那些专用于面向对象、基于组件、面向服务、面向代理、基于模型的分布式信息系统设计及开发的应用程序，凸显出受控于架构元素及相关结构的演化特性。此类系统的可持续性、适应性和可伸缩性已经成为非常重要的经济学问题。实际上，由于这些系统往往需要历经数年（人力资源/月）的开发和更新，因此它们的生命期必须得以延续，尤其要应对软件架构设计者、应用程序构建者和最终用户不断变化的需求。在这一背景下，学术和行业团体提出，新开放软件架构（即能够与其他架构通信及接口的架构）的设计和开发应该具有适应性（可以根据运营条件和不同应用领域设计）和可伸缩性（可以进行改良，以处理初始设计时没有预计到的服务）。

实际上，软件架构为复杂系统的如下固有需求提供了很好的应对之策：

- 在不同环境和背景中使用和重用系统的需求。因此，系统必须具备重新建模（改造、演化和再工程）的能力，以满足特殊使用条件下的需求（如分布式基础设施、有限资源和不同功能构成的不同视角）以及新的技术需求（组件、服务、架构视图等）。
- 采取可重用设计方法、构造可重用架构元素（对象、组件、服务及代理），降低系统开发及维护成本的需求。这些架构元素可以标识并从现有系统中抽取，在未来的开发中重用。
- 在最低成本下快速更新这些系统的需求。在这种情况下，自动化转换过程（结构和行为）、促进这些活动的重用（重用演化过程）是必要的。
- 控制复杂度（系统的理解和开发）的同时以高度抽象进行处理的需求。

目前，有鉴于此，在基于组件、基于服务架构（SOA）、基于代理架构和基于模型架构方面已经出现了一些提案。在任何情况下，我们所面临的挑战都包括质量和效率的改善，以及软件制造的工业化。

而且，由于大量提案和解决方案的出现，因此审视软件工程中与架构相关的研究和应用也是很有必要的。

为此，我们编写了本书，它用不同的技术和架构方法分析架构特征、模式与风格的代表模型、规范、验证以及管理这些连贯自治元素的工程方法，介绍面向对象、基于组件、面向服务、面向代理和基于模型的软件架构范式。

本书的主要目标读者是项目经理、项目负责人、架构师、设计师、开发人员和软件架构用户、理工院校教师、工程师，以及理工院校的大学生及研究生。

Mourad Chabane Oussalah

目 录

译者序
前 言

第 1 章 软件架构中面向对象、基于组件、面向代理和面向服务的范式 …… 1

1.1 引言 …… 1
1.2 历史 …… 2
1.2.1 面向对象范式 …… 2
1.2.2 基于组件范式 …… 3
1.2.3 面向代理范式 …… 3
1.2.4 面向服务范式 …… 4
1.3 软件架构 …… 6
1.3.1 面向对象软件架构 …… 6
1.3.2 基于组件软件架构 …… 7
1.3.3 面向代理软件架构 …… 8
1.3.4 面向服务架构 …… 10
1.4 概念框架的两个维度：定量和定性 …… 12
1.4.1 概念差异 …… 12
1.4.2 定量维度 …… 19
1.4.3 定性维度 …… 24
1.5 集成开发范式方法 …… 33
1.6 小结与讨论 …… 35
1.7 结语 …… 37
1.8 参考书目 …… 37

第 2 章　参考架构 …… 42

2.1　引言 …… 42
2.2　参考架构的定义 …… 42
2.2.1　参考架构与参考模型的对比 …… 43
2.2.2　参考架构与产品线架构的对比 …… 44
2.3　参考架构模型 …… 45
2.4　参考架构设计 …… 48
2.4.1　信息源调查 …… 49
2.4.2　架构需求确立 …… 50
2.4.3　参考架构设计 …… 51
2.4.4　参考架构评估 …… 53
2.5　参考架构的用途 …… 53
2.6　参考架构的示例 …… 56
2.7　参考架构的前景 …… 57
2.8　结语 …… 59
2.9　参考书目 …… 60

第 3 章　多层次 / 多视图软件架构 …… 64

3.1　引言 …… 64
3.2　现有视点方法 …… 65
3.2.1　引言 …… 65
3.2.2　需求规格中的视图 …… 65
3.2.3　系统建模中的视图 …… 66
3.2.4　编程中的视图 …… 66
3.3　软件架构中的视图 …… 67
3.3.1　视图在软件架构中的贡献 …… 67
3.3.2　“4+1”视图模型 …… 68
3.3.3　ISO/IEC/IEEE 42010 …… 69
3.3.4　视图及超越方法 …… 69
3.3.5　小结 …… 72
3.3.6　当前软件架构方法的局限性 …… 74
3.4　多层次 / 多视图软件架构的定义和基本概念 …… 74
3.4.1　定义 …… 74
3.4.2　概念和基础知识 …… 75

3.5 MoVAL：基于模型、视图和抽象级别的架构 …… 83
3.5.1 引言 …… 83
3.5.2 MoVAL …… 84
3.5.3 MoVAL 元模型 …… 87
3.5.4 案例分析 …… 89
3.6 结语 …… 92
3.7 参考书目 …… 93

第 4 章 软件架构与工具：分布与协调动态重配置管理 …… 95

4.1 引言 …… 95
4.2 背景 …… 96
4.3 分布式应用的动态重配置管理机制 …… 98
4.3.1 集中动态重配置管理 …… 98
4.3.2 分布式系统集中解决方案的局限性 …… 100
4.3.3 分布式重配置管理的优势与风险 …… 101
4.3.4 现有协调机制 …… 102
4.4 重配置基础设施的专门化 …… 104
4.4.1 行为的专门化 …… 105
4.4.2 适配机制分布的专门化 …… 105
4.5 分布式系统动态重配置的局限性和难点总结 …… 106
4.6 重配置管理机制的实施方法 …… 107
4.7 分布动态重配置管理的架构模型 …… 108
4.7.1 用于适配管理的组件类型 …… 109
4.7.2 动态重配置管理的分布 …… 110
4.7.3 适配管理器架构模型 …… 112
4.7.4 重配置机制的专门化 …… 113
4.7.5 重配置过程的协调 …… 115
4.8 结语 …… 128
4.9 参考书目 …… 129

第 5 章 产品线软件架构 …… 133

5.1 软件生产线简介 …… 133

5.1.1 3 种开发风格 …… 135
5.1.2 可变性管理 …… 135
5.1.3 产品线中的架构概念 …… 137
5.2 音乐商店示例 …… 139
5.2.1 领域 …… 139
5.2.2 SongStock 产品线 …… 139
5.2.3 功能需求 …… 140
5.2.4 其他主要需求 …… 140
5.3 领域工程 …… 141
5.3.1 领域分析 …… 141
5.3.2 集成可变性用例 …… 142
5.3.3 特征模型 …… 143
5.3.4 领域设计 …… 144
5.3.5 设计产品线架构 …… 145
5.4 产品工程 …… 148
5.4.1 产品的配置 …… 149
5.4.2 产品衍生 …… 149
5.5 参考架构设计过程 …… 151
5.6 延伸阅读 …… 153
5.6.1 PLA 与参考架构 …… 154
5.6.2 具有影响力的旧文献 …… 155
5.7 结语 …… 158
5.8 参考书目 …… 158

第 6 章 软件架构：Web 服务复合环境下的服务适配技术 …… 165

6.1 引言 …… 165
6.2 Web 服务复合和验证 …… 167
6.3 Web 服务不兼容和适配 …… 171
6.4 适配方法 …… 173
6.5 结语 …… 182
6.6 参考书目 …… 182

第 1 章　软件架构中面向对象、基于组件、面向代理和面向服务的范式

Abdelkrim Amirat, Anthony hock-koon, Mourad Chabane Oussalah

近年来，面向对象、面向代理和面向服务的范式并存，并平行发展。这导致了一些类似概念的出现，专业人士的概念往往与人们对词汇的错误解读共生。从 4 种范式中借鉴不同要素的混合型方法使这一现象更加恶化。而且，结合这些范式的现代应用程序强调它们的相互交织，这使得全面的理解变得更加困难。

本章的目的是根据两种定量和定性的方法提出一个概念对比框架，厘清范式之间的边界。原则是集中于与范式直接相关的概念特性差异，而不是比较实施这些范式的不同技术，目标是使架构师更好地理解采用某种范式或者其他范式的含义和后果。

1.1　引言

根据维基百科的解释，“编程范式是计算机编程的基本风格，决定了在编程语言中如何构建问题解决方案。”本章的重点是软件开发领域的 4 种关键范式：面向对象软件工程（OOSE）、基于组件软件工程（CBSE）、面向代理软件工程（AOSE）和面向服务软件工程（SOSE）。我们将借助分布式应用程序的实际构造方法来研究和分析这些范式。

软件开发范式规定了按照清晰定义的概念与机制来构思问题的信息技术（IT）解决方案的方法。它确定了处理问题的顺序，并从实践的角度提供按照其原则开发这一序列的手段。因此，从分析、设计和开发上，软件开发范式都有自己独特的 IT 解决方案开发风格。

从本质上讲，范式独立于具体的功能问题；但是，为了支持其特性，它可能鼓励某种类型的应用。不过，这些特质通常与具体的反馈相关。如果某个范式很适合于某一实现问题，就会减少对集成过程和使用常见概念框架进行隔离解决方案测试的需求，而这些需求的代价都很高。

在本章中，我们用自顶向下的方法提出一个概念框架，自顶向下方法的原理是集中于与范式直接相关的概念特性差异；与此相反，自底向上方法则研究技术上的差异。我们的基于对比概念框架依赖两种方法：基于产品和过程概念的定量方法，以及基于每个范式特性组织质量标准的定性方法。这些方法将帮助我们澄清关于不同范式概念和技术的错误理解。

1.2 历史

图 1-1 是根据 [SOM 04] 绘制的，展示了软件工程的演变。我们可以看到从结构化编程到当前流行的面向服务和基于模型范式[⊖]的发展。

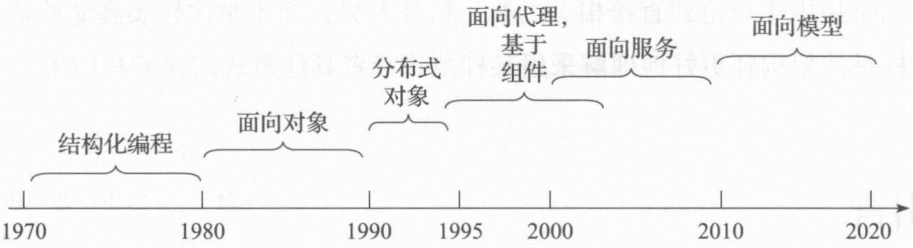

图 1-1 开发范式的演变

1.2.1 面向对象范式

面向对象范式（OOP）是一种面向设计与编程的范式，出现于 20 世纪 60 年代初，Alan Kay 在 20 世纪 70 年代的工作发展了这一范式 [KAY 93]。OOP 由软件模块（对象）的定义和交互组成：对象代表一个概念、一个思路或者现实世界中的任何实体 [OUS 99]，

⊖ 在本章中，我们有意略去了 OMG 提出的模型范式，而重点讨论面向对象、面向代理和面向服务范式，目的是为本章已足够广泛的内容腾出空间。

它有内部结构和行为，能够与其他对象通信。因此，OOP 的目标是表现这些对象及其关系；通过相互关系进行的对象间通信方便了预想功能的实现。

Simula-67 语言为面向对象语言打下了最初的基础：类、多态、继承等 [COX 91]。但是，在 Simula-67 和 Lisp 启发下开发的 Smalltalk 71 和后来的 Small talk 80（Dan Ingalls）[GOL 83] 才开始以 Alan Kay 的工作为基础，并确立了面向对象编程的原则：对象封装、消息、类型、多态（通过子类）；而其他原则（如继承）则从上述原则衍生而来，或者归入实现的范畴。

20 世纪 80 年代，面向对象语言不断涌现：Objective C（20 世纪 80 年代初）、1983 年的 C++（具备类结构的 C 语言）、1984 年的 Eiffel、1987 年的公用 Lisp 对象系统等。20 世纪 90 年代是面向对象编程在软件开发不同领域中得到加强的黄金年代。目前，面向对象方法被当作其他方法的参考模型。

远程方法调用（RMI）机制补充了面向对象方法，其目标是在编程模型中引入分布式概念。RMI 主要基于对象请求代理（ORB）的原理 [GAS 92，VIN 97]。

1.2.2　基于组件范式

基于组件范式是 McIlroy[MCI 68] 提出的，他使用管道组件和过滤器在 UNIX 上实现了一个基础架构。基于组件的开发出现于 20 世纪 90 年代初，以应对面向对象方法在重用和组合上的不足。基于组件的方法强调重用的重要性、问题隔离和促进组合，扩展了面向对象范式 [PAP 07]。

对于开发人员来说，阅读和理解现有代码总是乏味的工作；不过，以组件的形式重用现有代码有很大的好处。实际上，开发人员只需要知道组件包含的内容，而不需要知道实现的方法。此外，在基于组件方法中，组件开发和系统开发之间有明显的区别。对于前者，我们关注的是组件的布局，而对于后者，关注的则是兼容组件的组装和构成。

1.2.3　面向代理范式

面向代理范式出现于 20 世纪 70 年代，是在分布式人工智能（DAI）科学的引导下开发的，Hewitt[HEW 73，HEW 11] 提出了“参与者”（Actor）的概念，即互相竞争的

交互自治实体。20 世纪 90 年代中期，称为多代理系统（MAS）的一组模型出现。在这些模型中，代理被视为具有某些功能的完备实体，能够执行自己的服务或者通过交互使用其他代理的服务。基于组织的多代理系统（OMAS）就是此类新模型的一个例子[FER 03]。

代理按照其相互协作、协调和协商的社会化能力进行区分 [HYA 96]。自治和高层交互是基于代理和面向对象、基于组件和基于服务方法的主要不同点。代理可以分为以下两类。

- 反应式代理：等待一个事件发生，然后对其环境中的变化做出反应。
- 主动式代理：根据自身在环境中的活动做出决策。

软件代理有自己的控制线程，不仅封装自身的代码和状态，还封装其调用。这些代理还可能有规则和各自的目标，随着调用而以活动对象的形式出现。换言之，代理采取行动的时机或者方式由自身决定。

在代理模型中，通信通常是异步进行的。这意味着，从一个代理到另一个代理没有预先定义的控制流。代理可能在任何时候发起内部或者外部自主行为，而不仅在接收到消息时才行动 [HEW 77]。

代理不仅对特定方法调用做出反应，还对环境中的可观测事件做出反应。主动式代理可能会查询环境事件和其他消息，以确定所要采取的措施。

1.2.4 面向服务范式

面向服务范式（SOSE）是相对新型的软件开发范式，起源于 21 世纪初，在该领域中已经扎下了根基。SOSE 直接受到跨国贸易中组织方法的启发，以经典的服务概念为基础。

SOSE 起源于如下环境的要求：需要承受变化越来越快的异构环境的系统（如互联网和 Web 服务）[CAS 03]、智能环境 [WEI 91] 或运行于企业网络上的业务应用程序（如企业资源规划系统（ERP））[PAP 07]。供应商的效率以及对变化需求的响应能力是 SOSE 试图解决的主要软件开发问题。

服务是代表特定功能的软件实体，也是不依赖于任何上下文或者外部服务的自治构件。它可以划分为多个操作，每个操作包含服务所能提供的特定行为。操作和服务正如 OOSE 中的方法和类。SOSE 还有复合服务的概念，通过组合服务描述构成。服务的复合在运行阶段实现。

SOSE 的关键要素之一是服务交互模式，也称面向服务架构（SOA），它使一系列服务能够相互通信。SOA 是设计和理解软件系统的一种手段，它通过发布跟踪接口向应用程序或者其他服务提供服务。

服务是提供商为客户执行的一个操作（功能）；但是，供应商和客户之间的交互通过一个中介（可能是一条总线）建立，中介负责将不同的参与方联系到一起。服务通常作为粗粒度的软件实体实现。它们包含并提供系统实体。这些系统也可以定义为应用层。服务的概念代表遵循如下特性的处理实体。

- 粗粒度：服务提供的操作封装多种功能，在广泛的数据上操作，这一点与基于组件的概念不同。
- 接口：服务可以实现多种接口，多种服务也可以实现共同的接口。
- 架构：每个服务由一个架构描述，架构使我们能够理解服务的行为、条件、代价和非功能属性。
- 可发现：在调用（绑定、引用）服务之前，必须发现（查找）它。
- 单实例：与按需实例化、可以同时具有多个实例的组件不同，服务只能有一个，对应于单例设计模式。
- 松散耦合：服务通过标准与客户和其他服务连接。这些标准确保了解耦——也就是减少依赖。这些标准包括 Web 服务中使用的可扩展标记语言（XML）文档。但是，有多种通信技术管理服务实现的异构性，使其仍能相互通信。在 SOSE 的语境中，耦合包含动态发现服务和自动更改 / 替换这些服务的所有概念。

SOSE 将应用程序视为根据角色交互的一组服务，而不考虑这些服务的位置，以便维持异构的松散耦合软件系统。Web 服务是服务的一个例子，其中使用了 3 个基本要素：作为描述语言的 Web 服务描述语言（WSDL，一种 XML 元语言）；用于实现本地化的通用描述、发现与集成（UDDI）注册中心；以及超文本传输协议（HTTP）或者简单对象访问协议（SOAP）等传输协议。

SOA 本质上是一组相互交互和通信的服务。这种通信只包含返回数据或者活动（多种服务的协调）。SOA 是实施服务的交互模型应用，该术语起源于 2000~2001 年间。

与解决方案技术架构的不同层次相对应的是服务的层次结构。SOA 是公司面对的可重用性、互操作性问题的高效解决方案，并且可以减少实施公司信息系统的各系统之间的耦合。

基于 J2EE 或者 .NET 等平台的电子商务、企业对企业（B2B）或者企业对消费者（B2C）中 Web 服务的标准的出现，使 SOA 成为主流。

1.3 软件架构

多年以来，软件架构以方框和直线的方式描述。直到 20 世纪 90 年代初，软件开发人员才意识到软件架构在软件系统的成功开发、维护和演化中所扮演的重要角色。好的软件架构设计可以得到符合消费者需求且容易更新的产品，而不合适的架构则可能造成灾难性后果，导致项目取消 [TAY 09]。

1.3.1 面向对象软件架构

面向对象建模根据面向对象的概念，创建框图、文本规格和编程源代码以描述软件系统。面向对象建模语言是以图形方式分析和表现软件系统的方法与技术。对象建模有多种方法，如 Wirfs-Brock 的面向对象软件设计（DOSS）、Rumbaugh 的对象建模技术（MOT）、Jacobson 的 OOSE 或者 Booch 的面向对象分析与设计（OOD）。但是，近来这些方法中大部分已经整合到 Booch 等人的统一建模语言（UML）中，因此分析人员已经不再采用它们。面向对象软件架构用于将系统描述为一组类（抽象和封装功能的实体），这些类可以有对象（实例），通过消息发送相互通信 [OUS 99，OUS 05]。

1.3.1.1 面向对象软件架构的优势和劣势

面向对象软件架构提供如下优势：

- 基于精确定义的方法学，在一组需求的基础上开发系统。