HZ BOOKS 华章IT

MANNING

架构师书库

Microservices Patterns

With examples in Java

微服务架构设计模式

[美] 克里斯·理查森（Chris Richardson） 著 喻勇 译

涵盖44个架构设计模式，系统解决服务拆分、事务管理、查询和跨服务通信等难题

易宝支付CTO陈斌、PolarisTech 联合创始人蔡书、才云科技CEO张鑫等多位专家鼎力推荐

机械工业出版社
China Machine Press

架构师书库

MICROSERVICES PATTERNS

With examples in Java

微服务架构设计模式

[美] 克里斯·理查森（Chris Richardson） 著 喻勇 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

微服务架构设计模式 /（美）克里斯·理查森（Chris Richardson）著；喻勇译 . —北京：机械工业出版社，2019.4（2019.6 重印）
（架构师书库）
书名原文：Microservices Patterns: With Examples in Java

ISBN 978-7-111-62412-7

I. 微… II. ① 克… ② 喻… III. 互联网络 - 网络服务器 IV. TP368.5

中国版本图书馆 CIP 数据核字（2019）第 061419 号

本书版权登记号：图字 01-2018-5458

微服务架构设计模式

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：殷 虹
印　　刷：三河市宏图印务有限公司　　版　　次：2019 年 6 月第 1 版第 3 次印刷
开　　本：186mm × 240mm 1/16　　印　　张：30.25
书　　号：ISBN 978-7-111-62412-7　　定　　价：139.00 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　　读者信箱：hzit@hzbook.com

写给中文版读者的话

7年前，我带着对美食和技术的热情，开始了我的首次中国之旅。在那之前，我对中国的美食和软件社区都知之甚少。7年之后，经过多次中国之行，我对这两者都有了深刻的认识：我爱上了地道的中国菜，也对中国的软件开发者印象深刻。

2012年我首次访问中国，参加我在VMware公司的同事Frank[⊖]举办的几场开发者会议。我一口气在北京和上海做了好几场演讲，包括云计算、Cloud Foundry、Node.js、Spring、NoSQL数据库，当然，还有微服务。我与2000多位参加会议的来宾讨论Cloud Foundry，这次旅行让我意识到中国开发者社区的规模和热情，也让我有机会品尝了地道的中国菜。我甚至还忙里偷闲，在北京参加了一天中餐烹饪课程。

2013年，Frank再次邀请我来到北京，参加中国首场SpringOne大会，发表关于微服务和NoSQL的演讲。这次旅行的亮点是访问豆瓣和百度，这是我与中国科技公司的第一次近距离接触。他们的规模和创新技术都给我留下了非常深刻的印象。在这次旅行中，我参观了北京奥林匹克公园，回忆了曾在这里举行的2008年北京奥运会开幕式。我也抓住机会，继续“进修”中餐烹饪课程。

这次大会结束后不久，我离开VMware公司，再次走上了创业的道路。我搭建了microservices.io网站，撰写了大量的文章和课件，搭乘我钟爱的United Airlines，为世界各地的客户提供微服务架构咨询和培训服务。我还创立了eventuate.io公司，发布了用于微服务架构的数据访问框架。这些工作促成了我和Frank的再度合作，我有幸在2016年4月和8月再次访问中国。从那以后，我在中美之间多次往返，帮助中国的企业客户实施微服务架构。这些公司的业务多种多样，包括保险、汽车制造、电信和企业软件。地域上的跨度，也从北京和上海延伸到了深圳、武汉和杭州。在这些旅行中，我爱上了烤鱼、新疆菜和蒙古菜。

⊖ 即本书译者。——编辑注

中国企业和开发者对微服务架构的热情让我印象深刻。但如同我给所有客户的忠告一样，我想对本书的读者说：

第一，要记住微服务不是解决所有问题的万能“银弹”。

第二，编写整洁的代码和使用自动化测试至关重要，因为这是现代软件开发的基础。

第三，关注微服务的本质，即服务的分解和定义，而不是技术，如容器和其他工具。

第四，确保你的服务松耦合，并且可以独立开发、测试和部署，不要搞成分布式单体（Distributed Monolith），那将会是巨大的灾难。

第五，也是最重要的，不能只是在技术上采用微服务架构。拥抱 DevOps 的原则和实践，在组织结构上实现跨职能的自治团队，这必不可少。

还必须记住：实现微服务架构并不是你的目标。你的目标是加速大型复杂应用程序的开发。

最后，我要感谢中国的所有客户，让我有机会与你们探讨微服务。我还要感谢那些让我能够讨论技术而不用学说中文（这可比微服务难多了）的同传翻译。我希望你会喜欢阅读这本书，它会教你如何成功开发微服务。我期待着再次访问中国，与我的读者见面，帮助更多企业客户实施微服务架构。

Chris Richardson

2019 年 2 月 13 日

译 者 序

2012 年年初，我有幸加入了 VMware 公司的 Cloud Foundry 团队，与 Chris Richardson、Patrick Chanezon、Josh Long 等业界大咖共事，在全球范围内开展 Cloud Foundry 开发者社区和生态建设工作。7 年前，云计算的市场格局与现在大为不同。那时，IaaS 正高歌猛进，PaaS 的价值仍旧备受质疑，“十二原则”还不为人所知，云端分布式系统的架构演化也正“摸着石头过河”。在这个时候，Chris Richardson 率先在业界提出了“Functional Decomposition”（功能性拆分）的概念，提出云计算环境下的分布式软件，应该按照功能性拆分的方式进行架构重构。这个想法与稍后业界公认的“微服务”概念不谋而合。

在 VMware 公司工作期间，以及之后各自的创业经历中，我跟 Chris 保持着良好的个人关系和工作合作关系。Chris 是一个风趣、博学、经验丰富的架构师，他在软件行业有将近 30 年的经验，在 Java 社区更是享有盛名。在离开 VMware 公司后，他建立了 microservices.io 网站，专注微服务架构的咨询和培训工作，我也曾为他牵线搭桥，使他有机会为国内的企业客户提供咨询服务。

经过这些年的发展，微服务已经成为软件领域的新宠，国外 Netflix、Amazon 的成功案例，国内数字化转型的一波波浪潮，推动着 PaaS 厂商和开发者深度关注微服务。大家围绕着微服务展开了大量的讨论。在这个过程中，我们认识到，虽然很多企业客户视微服务如救命稻草，但微服务并不能解决一切问题。很多客户，亦盲从于各种厂商的“忽悠”，着力建设底层基础设施。

面对这些迷茫，Chris 曾对我说，软件的架构设计，就是选择和取舍。面对围绕微服务的众多杂音，开发者和架构师应该具备选择和取舍的能力，应该站在比较高的角度俯瞰全局、权衡利弊，做出正确的架构和技术选择。这也是最初 Chris 写作本书的动机之一：为架构师提供一个微服务的全局视野，并教会架构师如何在纷繁复杂的情况下做出正确的架构选

择和取舍。

本书英文版的写作开始于2017年春天，2018年10月正式出版。在英文版出版后，我集中利用两个多月的时间完成了中文版的翻译工作。这是一本30万字的大部头，Chris曾数次对英文版做出较大的结构性修改。为了确保中文版的一致性和准确性，并且以最快速度翻译出版，中文版初稿完成后，先后经历了7轮修改润色和校对。在后期校对阶段，我邀请了数位好友帮助把关，他们是：薛江波、王天青、季奔牛、刘果、蔡书、张鑫、张扬、黄雨婷、毛艳玲。我特别感谢这些朋友，因为他们细致地校对了所有翻译稿，帮我找到并修正了大量足以让我“晚节不保”的低级错误。蔡书和张鑫还在繁忙的创业工作之余细读整本书，并撰写了推荐序。

本书的中文版出版后，我将与Chris重启针对中国市场的微服务咨询和培训业务。为此，我们发布了中文网站www.chrisrichardson.cn，并有针对性地设计了微服务培训和技术咨询的服务项目。我们期待与读者面对面交流的机会。

喻　勇

2019年2月14日

中文版序一

良马难乘，然可以任重致远；良才难令，然可以致君见尊。

——墨子

曾经有一个客户把他们遇到的微服务问题列出来给我看，当时我觉得头绪万千但又无从说起，于是想到了墨子的这句话。

如果现在有人问我这个问题，那么我会推荐他们一边看 Chris Richardson 的这本书，一边在实践中尝试和体验各种模式的优势与特点，然后大家一起讨论遇到的问题并提出解决思路。

大概从五六年前开始，我在工作中越来越多地谈到了微服务，并参与了一些客户应用的微服务改造，其中不乏成功的例子，当然也有没达到预期的情况。随着网络基础设施的高速发展，以及越来越多的企业和组织需要通过互联网提供服务，在考虑构建可以支持海量请求以及多变业务的软件平台时，微服务架构成为多数人的首选。微服务架构的出现是符合事物发展规律的：当问题足够大、有足够多的不确定性因素时，人们习惯把大的问题拆分成小的问题，通过分割、抽象和重用小而可靠的功能模块来构建整体的方案。但是当这些小的、可重用的部分越来越多时，又会出现新的问题。在相似的阶段，人们遇到的问题通常也是相似的，这个时候我们需要一些共识，需要用一些通用的词汇来描述问题以及解题思路和方案，这也是人们知识的总结。微服务模式就是这样一种总结和概括，是一种可以通用的共识，用于描述微服务领域中的问题及解决方案、方法和思路。这是我向大家推荐这本书的理由之一：讨论微服务的时候，这本书提供了必要的共同语言。

在和 Chris 交流时，我深深地被他高度的思维能力所折服，尤其是对问题的深刻理解和对解决思路的高度抽象。与有敏锐思维且有高度抽象能力的人讨论问题是件快乐的事情，他总是能把自己的经验和概括总结出的信息用清晰的方式表述出来。现在，他把关于微服务的

这些抽象整理成了这本书。可以说，这是广大微服务相关工作人员的福音。在这本书里，不仅有微服务领域已经识别出来的问题、解决思路和解决方案，也有相应的代码例子。这就使得高度抽象的内容有了非常具体的表现，可以帮助我们在遇到问题之前就了解可能的潜在问题；有些代码例子甚至是可以直接使用的。这种知行合一的能力，是我钦佩 Chris 的又一个重要原因，也是我向大家推荐这本书的理由之二：这本书可以帮助微服务相关人员构建知行合一的能力。

在一次关于“架构的关键是什么”的讨论中，我们和 Chris 很快达成了共识：架构就是取舍，进而架构师就是做出取舍的人。大家都认同，做架构的人的特征之一应该是“Independent”（独立），这也是我选择做独立解决方案进而设计产品的重要原因。在我们看来，只有独立才有可能让我们在做架构设计时做出中立和独特的方案。面对问题时，大多数人会希望有人可以给出“正确的”建议，但是多数时候，困扰人们的不是“什么才是正确的”，而是“取舍之间”。这正是我推荐这本书的理由之三：这是一本可以帮你在设计微服务架构时做出取舍的书，它能在你处理微服务相关问题左右为难的时候给你提供参考和建议。

我们生活在一个高速发展的时代，微服务领域的技术、产品、模式日新月异，我们非常有幸参与和见证这个时代的发展。我们从解决昨天的问题里走出来，又走向更多的问题。在这个过程中，我们解决的问题的规模和复杂度都是成倍提升的。相信很多和我一样喜欢体验这种从无到有的过程、喜欢亲手解决问题的成就感、喜欢用独立思维去面对问题的人，都会喜欢这本书。在此，再次对 Chris Richardson 先生表示感谢，他为这个领域贡献了宝贵的知识财富。

蔡　书

独立顾问，PolarisTech 联合创始人

中文版序二

国际数据公司（IDC）研究表明，2018～2021 年间，全球数字化转型方面的直接支出将达到 5.9 万亿美元。埃森哲（Accenture）指出，目前在中国仅有 7% 的企业成功地实现了数字化转型，而这些成功转型的公司，它们的业绩复合增长率是尚未转型的同行企业的 5 倍之多。

数字化转型依赖技术创新。美国风险投资机构 Work-Bench 在《2018 企业软件调研年报》中推论：以微服务为代表的云原生技术是帮助企业实现有效数字化转型的唯一技术途径。数字化转型背景下客户的预期越来越高，需要企业的线上业务能快速迭代满足动态的市场需求，并能弹性扩展应对业务的突发式增长；而微服务由于其敏捷灵活等特性成了满足这些诉求的最佳答案。因此，微服务可成为企业进行数字化转型的强力催化剂。

微服务的概念虽然直观易懂，但“细节是魔鬼”，微服务在实操落地的环节中存在诸多挑战。我们在为企业提供 PaaS、人工智能、云原生平台等数字化转型解决方案时也发现，企业实现云原生，并充分利用 PaaS 能力的第一步，往往是对已有应用架构进行现代化微服务改造，而如何进行微服务拆分、设计微服务逻辑、实现微服务治理等实操问题成为很大的挑战。

本书英文原作由微服务权威架构师 Chris Richardson 先生所著。书中既包含了微服务的原理、原则，又包含了实际落地中的架构设计模式；既包含可举一反三的理念和概念，也包含类似领域驱动设计、Saga 实现事务操作、CQRS 构建事件驱动系统等具体可套用的范式。本书可以帮助读者把传统的单体巨石型应用循序渐进地改造为微服务架构，从微服务的拆分，微服务架构下业务逻辑的设计以及事务、API、通信等的实现，一直到微服务系统的测试与生产上线，帮助读者建立从无到有的完整微服务系统搭建的生命周期。

本书译者在云计算、云原生与微服务领域有多年实践经验和建树，译文既精确地还原了原著的内容，又结合译者自身的理解，让中文版本更加通俗易懂。虽身在云计算行业多年，我在通读译著后依然受益匪浅。相信本书对于企业 CIO 推动公司数字化转型战略、软件开发者提升自身技术架构功力，以及云原生爱好者以微服务切入最新的云原生体系，都有着极其重要的实践指导意义。

张　鑫

才云科技 CEO

前　言

我最喜欢的格言之一是：

未来已经到来，只是还没有平均分布。

——威廉·吉布森，科幻小说作家

这句话的实质是在说，新的想法和技术需要一段时间才能通过社区传播开来并被广泛采用。我发现并深入关注微服务的故事，就是新思想缓慢扩散的一个极好例子。这个故事始于2006年，当时受到AWS布道师一次演讲的启发，我开始走上了一条最终导致我创建早期Cloud Foundry的道路[⊖]，它与今天的Cloud Foundry唯一相同的是名称。Cloud Foundry采用平台即服务（PaaS）模式，用于在EC2上自动部署Java应用程序。与我构建的其他每个企业级Java应用程序一样，我的Cloud Foundry采用了单体架构，它由单个Java Web应用程序（WAR）文件构成。

将初始化、配置、监控和管理等各种复杂的功能捆绑到一个单体架构中，这给开发和运维都带来了挑战。例如，你无法在不测试和重新部署整个应用程序的情况下更改它的用户界面。因为监控和管理组件依赖于维护内存状态的复杂事件处理（CEP）引擎，所以我们无法运行应用程序的多个实例！这是个令人尴尬的事实，但我可以说的是，我是一名软件开发人员，就让我这个无辜的码农来指出这些问题吧[⊜]。

显然，单体架构无法满足应用程序的需求，但替代方案是什么？在eBay和亚马逊等公司，软件界已经开始逐渐尝试一些新东西。例如，亚马逊在2002年左右开始逐步从单体架构迁移（https://plus.google.com/110981030061712822816/posts/AaygmbzVeRq）。新架构用一

⊖ Chris是Cloud Foundry开源PaaS平台的创始人，VMware公司通过SpringSource收购了他的项目，该项目逐步演化为今天的Pivotal Cloud Foundry。——译者注

⊜ 这里原文是“let he who is without sin cast the first stone”。——译者注

系列松散耦合的服务取代了单体。服务由亚马逊称为“两个比萨”的团队所维护：团队规模小到两个比萨饼就能让所有人吃饱。

亚马逊采用这种架构来加快软件开发速度，以便公司能够更快地进行创新并赢得竞争。结果令人印象深刻：据报道，亚马逊平均每 11.6 秒就能够将代码的更改部署到生产环境中！

2010 年年初，当我转向其他项目之后，我终于领悟了软件架构的未来。那时我正在读 Michael T. Fisher 和 Martin L. Abbott 撰写的《 The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise 》(Addison-Wesley Professional，2009) ⊖。该书中的一个关键思想是扩展立方体，如第 2 章所述，它是一个用于扩展应用程序的三维模型。由扩展立方体定义的 *Y* 轴扩展功能将应用程序功能分解为服务。事后来看，这是显而易见的，但对我来说，这是一个让我醍醐灌顶的时刻！如果将 Cloud Foundry 设计为一组服务，我本可以解决两年前面临的挑战！

2012 年 4 月，我首次就这种架构方法发表了题为“Decomposing Applications for Scalability and Deployability”的演讲（www.slideshare.net/chris.e.richardson/decomposing-applications-for-scalability-and-deployability-april-2012)。当时，这种架构并没有一个被普遍接受的名称。我有时称它为模块化多语言架构，因为服务可以用不同的语言编写。

未来还没有平均分布的另一个佐证是，微服务这个词在 2011 年的软件架构研讨会上被用来描述这种架构（https://en.wikipedia.org/wiki/Microservices)。当我听到 Fred George 在 Oredev 2013 上发表演讲时，我第一次遇到这个词，我立刻喜欢上了它！

2014 年 1 月，我创建了 https://microservices.io 网站，以记录我遇到的与微服务有关的架构和设计模式。在 2014 年 3 月，James Lewis 和 Martin Fowler 发表了一篇关于微服务的博客文章（https://martinfowler.com/articles/microservices.html)。随着微服务这个术语被广泛传播，这篇博客文章使整个软件社区开始围绕微服务这个新概念展开更进一步的思考和行动。

小型、松散耦合的团队快速可靠地开发和运维微服务的思想正在通过软件社区慢慢扩散。但是，这种对未来的看法可能与日常现实截然不同。如今，业务关键型企业应用程序通常是由大型团队开发的大型单体应用。虽然软件版本不经常更新，但每次更新都会给所涉及的参与人员带来巨大的痛苦。IT 经常难以跟上业务需求。大家都很想知道如何采用微服务架构来解决所有这些问题。

本书的目标就是回答这个问题。它将使读者对微服务架构、它的好处和弊端，以及应该何时使用微服务架构有一个很好的理解。书中描述了如何解决我们将面临的众多架构设计挑战，包括如何管理分布式数据，还介绍了如何将单体应用程序重构为微服务架构。但本书并不是鼓吹微服务架构的宣言。相反，它的内容围绕着一系列模式进行展开。模式是在特定上

⊖ 中文版已由机械工业出版社引进出版，书名为《架构即未来：现代企业可扩展的 Web 架构、流程和组织》，ISBN: 978-7-111-53264-4。——译者注

下文中发生的问题的可重用解决方案。模式的优点在于，除了描述解决方案的好处之外，还描述了成功实施解决方案时必须克服的弊端和问题。根据我的经验，在选择解决方案时，这种客观性会带来更好的决策。我希望你会喜欢阅读这本书，它会教你如何成功开发基于微服务架构的应用程序。

致谢

写作是一项孤独的活动，但是把粗略的草稿变成一本完整的图书，却需要来自各方的共同努力。

首先，我要感谢 Manning 出版社的 Erin Twohey 和 Michael Stevens，他们一直鼓励我在《 POJOs in Action》之后再写一本书。我还要感谢我的编辑 Cynthia Kane 和 Marina Michaels。Cynthia Kane 帮助我启动了这本书，并在前几章与我合作。Marina Michaels 接替 Cynthia 并与我一起工作到最后。Marina 对书的内容提出了细致和建设性的意见，我将永远感激不尽。我还要感谢 Manning 出版社参与了这本书的出版的其他成员。

我要感谢技术编辑 Christian Mennerich、技术校对员 Andy Miles 以及所有的外部审校员：Andy Kirsch、Antonio Pessolano、Areg Melik-Adamyan、Cage Slagel、Carlos Curotto、Dror Helper、Eros Pedrini、Hugo Cruz、Irina Romanenko、Jesse Rosalia、Joe Justesen、John Guthrie、Keerthi Shetty、Michele Mauro、Paul Grebenc、Pethuru Raj、Potito Coluccelli、Shobha Iyer、Simeon Leyzerzon、Srihari Sridharan、Tim Moore、Tony Sweets、Trent Whiteley、Wes Shaddix、William E. Wheeler 和 Zoltan Hamori。

我还要感谢所有购买 MEAP 预览版[⊖]并在论坛或直接向我提供反馈的人。

我要感谢我曾经参与过的所有会议和聚会的组织者及与会者，他们给了我大量的机会，让我介绍和调整我关于微服务的想法。我要感谢我在世界各地的咨询和培训客户，让我有机会帮助他们将我关于微服务的想法付诸实践。

我还要感谢 Eventuate 公司的同事 Andrew、Valentin、Artem 和 Stanislav 对 Eventuate 产品和开源项目的贡献。

最后，我要感谢妻子 Laura 和孩子 Ellie、Thomas 和 Janet，感谢他们在过去的 18 个月里给予我的支持和理解。这段时间我一直盯着我的笔记本电脑，以至于接连错过了观看 Ellie 的足球比赛、Thomas 学习飞行模拟器，以及尝试与 Janet 一起开设新餐馆这些重要的家庭活动。

谢谢你们！

⊖ MEAP 是 Manning 出版社的一个网上付费试读服务，读者可以在 MEAP 上付费阅读那些尚处于写作过程中的书籍初稿。Chris 的这本书是 2018 年 MEAP 的销量冠军。——译者注

引　言

本书的目标是让架构师和程序员学会使用微服务架构成功开发应用程序。

书中不仅讨论了微服务架构的好处，还描述了它们的弊端。读者将掌握如何在使用单体架构和使用微服务架构之间做出正确的权衡。

谁应该阅读本书

本书的重点是架构和开发，适合负责开发和交付软件的任何人（例如开发人员、架构师、CTO 或工程副总裁）阅读。

本书侧重于解释微服务架构的设计模式和其他概念。无论读者使用何种技术栈，我的目标都是让你们可以轻松读懂这本书。你只需要熟悉企业应用程序架构和设计的基础知识即可。特别是，需要了解三层架构、Web 应用程序设计、关系型数据库、使用消息和基于 REST 的进程间通信，以及应用程序安全性的基础知识等概念。本书的代码示例使用 Java 和 Spring 框架。为了充分利用它们，读者应该对 Spring 框架有所了解。

本书内容安排

本书由 13 章组成。

- 第 1 章描述了所谓“单体地狱”的症状，当单体应用程序超出其架构时会出现这种问题，这可以通过采用微服务架构来规避。这一章还概述了微服务架构模式语言，这也

是本书大部分内容的主题。

- 第 2 章解释了为什么软件架构很重要，描述了可用于将应用程序分解为服务集合的模式，并解释了如何克服在此过程中遇到的各种障碍。
- 第 3 章介绍了微服务架构中强大的进程间通信的几种模式，解释了为什么异步和基于消息的通信通常是最佳选择。
- 第 4 章介绍如何使用 Saga 模式维护服务间的数据一致性。Saga 是通过传递异步消息的方式进行协调的一系列本地事务。
- 第 5 章介绍如何使用领域驱动设计（DDD）的聚合和领域事件等模式为服务设计业务逻辑。
- 第 6 章以第 5 章为基础，解释了如何使用事件溯源模式开发业务逻辑，事件溯源模式是一种以事件为中心的设计思路，用来构建业务逻辑和持久化领域对象。
- 第 7 章介绍如何使用 API 组合模式或命令查询职责隔离（CQRS）模式，这两个模式用来实现查询分散在多个服务中的数据。
- 第 8 章介绍了处理来自各种外部客户端请求的外部 API 模式，例如移动应用程序、基于浏览器的 JavaScript 应用程序和第三方应用程序。
- 第 9 章是关于微服务自动化测试技术的两章中的第一章，介绍了重要的测试概念，例如测试金字塔，描述了测试套件中每种测试类型的相对比例，还展示了如何编写构成测试金字塔基础的单元测试。
- 第 10 章以第 9 章为基础，描述了如何在测试金字塔中编写其他类型的测试，包括集成测试、消费者契约测试和组件测试等。
- 第 11 章介绍了开发生产就绪服务的各个方面，包括安全性、外部化配置模式和服务可观测性模式。服务可观测性模式包括日志聚合、应用指标和分布式追踪。
- 第 12 章介绍了可用于部署服务的各种部署模式，包括虚拟机、容器和 Serverless 模式。还介绍了使用服务网格的好处，服务网格是在微服务架构中处理服务间通信的一个网络软件层。
- 第 13 章介绍了如何通过采用绞杀者（Strangler）模式逐步将单体架构重构为微服务架构，绞杀者模式是指以服务形式实现新功能，从单体中提取模块将其转换为服务。

在学习这些章节的过程中，读者将了解微服务架构的不同方面。

关于本书中的代码

本书包含许多源代码示例，包括带有编号的代码清单和直接体现在正文中的代码。在这两种情况下，源代码都以相同的等宽字体排版，以便与普通文本分开。有些情况下，代码被

设定为粗体字，这用来表示相对之前的章节这些代码的内容已经发生了变化（例如当新功能添加到现有代码行时）。在许多情况下，书中展示的源代码已经重新排版，出版商添加了换行符和缩进，以适应书中可用的页面空间。在极少数情况下，代码中会包括续行标记（➥）。此外，当正文中描述代码时，源代码中的注释通常从代码中删除了。一些代码段落会包含醒目的粗体字提示和解释，这是用来强调重要概念的。

除第 1 章、第 2 章和第 13 章外，每章都包含来自配套示例应用程序的代码。读者可以在 GitHub 代码库（https://github.com/microservices-patterns/ftgo-application）⊖中找到此应用程序的代码。

线上论坛

读者可以免费访问由 Manning 出版社运营的网络论坛，可以在其中对本书发表评论、提出技术问题、分享练习的解决方案，并从作者和其他用户那里获得帮助。要访问论坛并订阅论坛内容，请用 Web 浏览器访问：https://forums.manning.com/forums/microservices-patterns。读者还可以在 https://forums.manning.com/forums/about 上了解有关 Manning 论坛和行为规则的更多信息。

Manning 出版社对读者的承诺是提供一个场所，在这里读者与读者之间以及读者与作者之间可以进行有意义的对话。这并不意味着作者做出了任何具体参与的承诺，作者对论坛的贡献仍然是自愿的（而且是无偿的）。我们建议尝试向作者询问一些具有挑战性的问题，以免他失去兴趣！只要本书出版，论坛和之前讨论的内容就可以从出版商的网站上获取。

其他线上资源

学习微服务架构的另一个重要资源是 Chris Richardson 的个人网站 https://microservices.io。

该网站不仅包含完整的模式语言，还包含指向其他资源（如文章、演示文稿和示例代码）的链接。

关于作者

Chris Richardson 是一名开发者和架构师。他是 Java 社区的著名布道师、JavaOne 等知

⊖ Chris 在此提供了包括 Dockerfile 和测试用例在内的一整套微服务架构实现代码，是非常有价值的一套学习和参考资料。——译者注