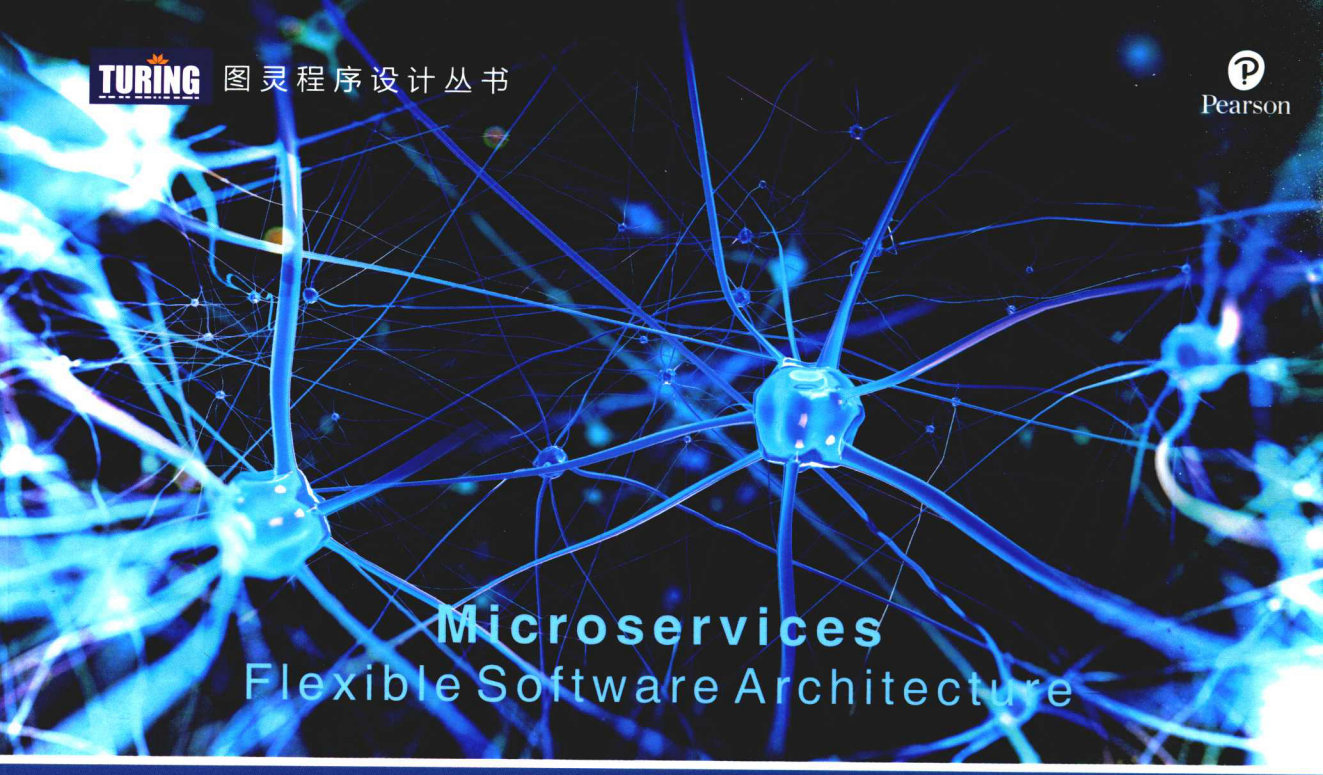
TURING 图灵程序设计丛书

Pearson



Microservices
Flexible Software Architecture

微服务

灵活的软件架构

[德] 埃伯哈德·沃尔夫（Eberhard Wolff）◎著　　莫树聪◎译

- 集结架构师多年实战经验与开发心得，直击微服务架构精髓
- 全面、实用，真实还原各种应用场景，助你做出明智的架构决策

中国工信出版集团　人民邮电出版社 POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Microservices
Flexible Software Architecture

微服务
灵活的软件架构

[德] 埃伯哈德·沃尔夫（Eberhard Wolff）◎著　　莫树聪◎译

人民邮电出版社
北京

图书在版编目（CIP）数据

微服务 : 灵活的软件架构 /（德）埃伯哈德·沃尔夫著 ; 莫树聪译. -- 北京 : 人民邮电出版社, 2019.11
（图灵程序设计丛书）
ISBN 978-7-115-52129-3

Ⅰ. ①微… Ⅱ. ①埃… ②莫… Ⅲ. ①软件设计 Ⅳ. ①TP311.1

中国版本图书馆CIP数据核字(2019)第217862号

内 容 提 要

微服务具有模块性强、可替代性强、可持续开发、可独立伸缩、可持续交付等优点，近年来受到越来越多的开发者以及一些经验老到的架构师的青睐，采用微服务架构的公司也越来越多。本书围绕架构和团队的主题，详细介绍了微服务的各个方面，包括采用微服务的原因、微服务架构的基础知识、微服务的实际应用、如何克服相关的挑战，等等。本书还包含具体的实现示例，在代码层面详细介绍了微服务的技术实现。

本书适合所有希望采用微服务作为架构方案的管理者、架构师、开发者阅读。

◆ 著 [德] 埃伯哈德·沃尔夫
译 莫树聪
责任编辑 温 雪
责任印制 周昇亮

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷

◆ 开本：800×1000 1/16
印张：17
字数：402千字 2019年11月第1版
印数：1-3 500册 2019年11月河北第1次印刷

著作权合同登记号 图字：01-2017-2583号

定价：89.00元

读者服务热线：(010)51095183转600 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

感谢我的家人和朋友的支持。

感谢计算机业界带给我的乐趣。

前　言

“微服务”是一个新名词，但其所代表的理念实际上由来已久。2006 年，Werner Vogels（Amazon 公司 CTO）就曾经在 JAOO[①]会议上做过关于 Amazon Cloud 和 Amazon 合作伙伴模型的演讲。在演讲中，他提到了奠定 NoSQL 基础的 CAP 定理。此外，他还谈到了一些集开发和运维于一体的小团队，这些团队的软件提供多组服务，且每组服务有各自的数据库。现在，这种团队协作方式被称作 DevOps，这种软件架构就是我们说的微服务。

后来，有个客户需要将新技术模块集成到遗留项目中，让我给他们提供解决方案。经过几番将新模块直接集成到遗留代码中的尝试，我们最终开发了一个新应用。这个新应用采用完全不同于遗留项目的技术栈，新旧两个应用之间的耦合仅在于 HTML 链接以及一个共用的数据库。除了共用数据库，这实质上就是一个微服务的方案。那时还是 2008 年。

2009 年，我的另一个客户将其整个基础架构划分成多组 REST 服务，每组服务交由不同团队开发。这在今天也可称为微服务。那时，很多业务建立在 Web 之上的公司都纷纷转向了类似的架构模式。再后来，我意识到持续交付对软件架构的重大影响，而微服务架构对持续交付而言有着诸多优势。

近年来，包括一些经验老到的架构师在内的许多开发人员都开始投身于微服务方案的实践中。这也是我写这本书的初衷。和其他所有框架一样，微服务架构不可能解决一切设计问题。然而，与现有方案相比，微服务是个不错的选择。

内容概览

本书围绕架构和组织的主题，详细地介绍了微服务。当然，具体的程序实现方案也占了相当大的篇幅。书中用了一个完整示例来介绍微服务的具体实现。关于纳米服务（nanoservice）的讨论，则阐明了模块化并未止步于微服务。本书提供了详尽的信息，让读者可以顺利地开始使用微服务。

目标读者

本书的目标读者是所有希望采用微服务作为架构方案的管理者、开发人员和架构师。

---

① JAOO 是由丹麦 Trifork A/S 公司组织的开发者大会，目前已更名为 GOTO 大会。——译者注

管理者

当团队的组织形式适合微服务架构时，微服务才能产生最佳效果。在本书导论部分，管理者可理解微服务的基本理念。在后面的章节中，管理者可重点关注微服务架构对组织机构的影响。

开发人员

开发人员可将本书当作构建微服务的技术大全，通过本书可以掌握微服务的实战技能。本书通过一个详细的微服务示例，以及大量的其他技术介绍（例如纳米服务的示例），来帮助开发人员理解基本概念。

架构师

通过本书，架构师可从架构角度认识微服务，并深入理解相关的技术问题和团队组织问题。

本书特别关注了一些有可能出现的实验情景，以便读者将学到的新知识用于实战。同时，本书还提供了一些获取相关信息的渠道，以帮助感兴趣的读者深入研究相关技术。

结构和内容

本书分为四部分。

第一部分　动机和基础知识

第一部分解释采用微服务的原因，并介绍微服务架构的基础知识。第 1 章介绍了微服务的基本特点，包括优点和缺点。第 2 章介绍了两个采用微服务的案例：一个电商应用和一个信号处理系统。这部分内容能让读者对微服务及其应用有一个大致的了解。

第二部分　微服务是什么，用还是不用

第二部分不仅详细地研究了微服务，还具体分析了微服务的优缺点。

- 第 3 章从 3 个角度研究了“微服务”一词的定义：微服务的大小，康威定律（Conway's Law，该理论认为软件团队的组织结构决定软件架构），以及基于领域驱动设计和限界上下文的技术视角。
- 第 4 章详细介绍了采用微服务的原因。微服务的优势不仅存在于技术层面上，还体现在团队组织上。从业务角度考虑，也有充分的理由采用微服务。
- 第 5 章分析了微服务所带来的独特挑战。其中既包括技术挑战，也包括架构、基础设施、运维方面的挑战。

- 第 6 章主要分析了微服务与面向服务架构之间的区别。这两个概念乍一看似乎关系紧密，但实质上有非常多的不同之处。

第三部分　微服务的实现

第三部分主要研究微服务的实际应用，论述如何在程序中体现第二部分所说的微服务的优点，以及如何应对相关的挑战。

- 第 7 章探讨了基于微服务的系统架构，涉及领域架构及相关的技术挑战。
- 第 8 章介绍了微服务的集成，以及相互通信的不同方案。这部分内容不仅包含了 REST、消息等通信方式，还涉及用户界面（UI 级别的集成）和数据复制（数据库级别的集成）。
- 第 9 章展示了微服务可用的架构，例如命令查询职责分离（CQRS）、事件溯源以及六边形架构。这一章最后指明了应对特定挑战所适合采用的技术。
- 第 10 章主要关注测试。为了让不同微服务能独立部署，测试也应尽可能地独立。但测试不仅要针对各个独立的微服务，还要对系统做整体检查。
- 第 11 章探讨了运维和持续交付。微服务架构产生了大量可部署的构件，因而会增加对基础设施的需求。这也是采用微服务时不得不面对的一个实质性障碍。
- 第 12 章阐明了微服务是如何反过来影响团队结构的。微服务毕竟是一种架构，理应影响并改进团队结构。

第四部分　技术

第四部分在代码层面详细介绍了微服务的技术实现。

- 第 13 章包含一个基于 Java、Spring Boot、Docker 和 Spring Cloud 实现的详尽示例。这一章旨在用一个容易运行的示例应用作为切入点，以实际的技术来阐释微服务的理念，并作为微服务编程的入门指引。
- 第 14 章介绍了粒度比微服务更细的纳米服务。纳米服务需要特定的技术和方案来支持。这一章探讨了不同技术方案及其优缺点。
- 第 15 章展示了如何开始采用微服务。

延伸阅读

本书包含了一些微服务专家从不同角度撰写的文章。每位专家用大概 2 页的篇幅总结了他们对微服务的主要看法。这些文章各具特色：有的与本书观点相辅相成，有的关注点与本书不同，还有的与本书观点相冲突。这说明，软件架构领域通常没有唯一正确的答案，你能看到不同观点百家争鸣。这些文章为读者提供了一个接触不同观点的机会，以便形成自己的见解。

阅读路线

本书内容适合不同类型的读者。当然，读者可以且应该阅读和本职业并非直接相关的章节，但表 P-1 仍然列出了不同类型的读者最适合阅读的章节。

表 P-1　本书阅读路线

章　节	开发人员	架 构 师	管 理 者
第 1 章	○	○	○
第 2 章	○	○	○
第 3 章	○	○	○
第 4 章	○	○	○
第 5 章	○	○	○
第 6 章		○	○
第 7 章		○	
第 8 章	○	○	
第 9 章	○	○	
第 10 章	○	○	
第 11 章	○	○	
第 12 章			○
第 13 章	○		
第 14 章	○	○	
第 15 章	○	○	○

如果仅想从整体上了解微服务，那么推荐阅读每章的总结部分。想学到实际技能的读者可直接从第 13 章和第 14 章开始，这些章节介绍了实际的技术和代码。

各章节的“动手实践”部分通过实际练习，帮助你加深理解。如果某个章节特别吸引你，希望你可以完成相关的练习来更好地掌握这个章节的内容。

补充资料

本书勘误表、示例代码的链接以及其他信息可在 https://microservices-book.com/网站上找到。示例代码可在 https://www.github.com/ewolff/microservice/处获取。[①]

读者在 informit.com 注册本书，可方便地获取下载、更新、修订的内容。访问 informit.com/register，注册账号并登录后，可开始图书注册流程。输入本书英文版 ISBN（9780134602417），然后点击提交按钮。这一流程完成后，就可以在“Registered Products”一栏下面找到本书的更多信息。

① 读者可以访问本书图灵社区页面（https://www.ituring.com.cn/book/1918）提交中文版勘误，并下载示例代码。——编者注

致　　谢

我要感谢所有曾与我讨论过微服务的人们，以及所有曾向我咨询过或与我合作过的人们。但名单太长，无法一一列出。这些交流和讨论都非常有益和有趣！

我尤其要感谢 Jochen Binder、Matthias Bohlen、Merten Driemeyer、Martin Eigenbrodt、Oliver B. Fischer、Lars Gentsch、Oliver Gierke、Boris Gloger、Alexander Heusingfeld、Christine Koppelt、Andreas Krüger、Tammo van Lessen、Sascha Möllering、André Neubauer、Till Schulte-Coerne、Stefan Tilkov、Kai Tödter、Oliver Wolf，以及 Stefan Zörner。

作为英语母语使用者，Matt Duckhouse 大大提升了本书英文版的文字水平和可读性。

我的东家 innoQ 公司在本书写作过程中起到了重要的作用。不少 innoQ 同事的研讨和建议都反映在本书中。

我还想感谢我的家人和朋友，尤其是我的妻子，我埋头写作时经常会忽略她。另外，我要感谢她帮助翻译了本书的英文版。

当然，我还要感谢本书所提及的所有技术的开发者，是他们奠定了微服务发展的基础。另外，还要感谢本书“延展阅读”中所有文章的作者们，感谢他们分享了关于微服务的知识和经验。

Leanpub 为我的翻译工作提供了工具平台。该平台的使用体验非常棒。如果没有 Leanpub，可能就不会有本书的英文版。

Addison-Wesley 出版社的工作让本书英文版的品质更上一层楼。出书过程中，Chris Zahn、Chris Guzikowski、Lori Lyons 以及 Dhayanidhi Karunanidhi 提供了非常有力的支持。

最后，我要感谢 dpunkt.verlag 和 René Schönfeldt，感谢他们在本书最初的德文版出版过程中给予我的支持。

目　录

第一部分
动机和基础知识

第一部分将解释微服务是什么，微服务为何如此有吸引力，以及微服务可以用在哪些地方，并用示例说明微服务在不同案例中的影响。

第 1 章初步定义了何为微服务。

第 2 章通过详细的微服务案例说明了微服务可用在什么地方，并以此凸显了微服务的重要性。

第 1 章
预备知识

本章将概述微服务的概念。1.1 节明确了微服务的定义。1.2 节回答了“为什么采用微服务”这个问题。1.3 节探讨了与微服务相关的挑战。

1.1 微服务概述

本书关注的重点是微服务——一种实现软件模块化的方案。模块化并不是什么新概念。一直以来，我们都将大型系统划分成小模块，以便于软件的实现、理解以及后续开发。

微服务是一种新的模块化方法，但“微服务”一词却没有明确的定义，因此本章就从该词的定义出发，阐述微服务与一般的单体部署（deployment monolith）[①]之间的区别。

微服务定义初探

与单体部署不同的是，构成微服务的模块是以独立进程的形式运行的。这种方式源于 UNIX 思想，该思想可简述为如下三点。

- 一个程序应该仅完成一个任务，且应该完美地完成该任务。
- 程序之间应该能够协同工作。
- 程序之间应该提供一个通用的接口。在 UNIX 中，这个通用接口就是文本流。

微服务一词没有固定的定义。第 3 章将会提供更详细的定义。但以下标准可以作为微服务的基本概念。

- 微服务是一个模块化的概念。采用微服务是为了将大型软件系统分成更小的部分，从而影响团队的组织和软件系统的开发。

---

① 作者强调 deployment monolith（单体部署）是为了区分于所谓的 module monolith（所有代码放在同一个代码库中）以及 runtime monolith（系统以单一应用或进程的形式运行）。——译者注

- 微服务能相互独立地部署。一个微服务内部的变动不会影响其他微服务在生产环境下的部署。
- 微服务能基于不同技术来实现。各个微服务不局限于特定的编程语言或平台。
- 微服务拥有各自独立的数据存储：可以是私有的数据库，也可以是共享数据库中一个完全隔离的模式（schema）。
- 微服务可以自带支持服务，例如搜索引擎或特定的数据库。当然，所有微服务可以共享一个公共平台，例如虚拟机。
- 微服务是自包含（self-contained）的进程或虚拟机（例如支持服务包含在虚拟机内）。
- 微服务间必须通过网络通信。为此，微服务采用支持松耦合的协议，例如 REST 或消息机制。

单体部署

微服务的反面是单体部署。单体部署指的是只能部署成一整块的大型软件系统。单体部署必须作为一个整体通过持续交付流水线的所有流程，如开发、测试和发布。由于单体部署太大，它花费在这些流程上的时间远远超出小型系统。这降低了系统的灵活性，却增加了流程的成本。虽然单体部署的内部结构也可以模块化，但所有模块必须同时部署到生产环境。

1.2 为什么采用微服务

微服务架构允许软件分成模块，使软件的修改更容易。

如图 1-1 所示，微服务有许多显著的优势。

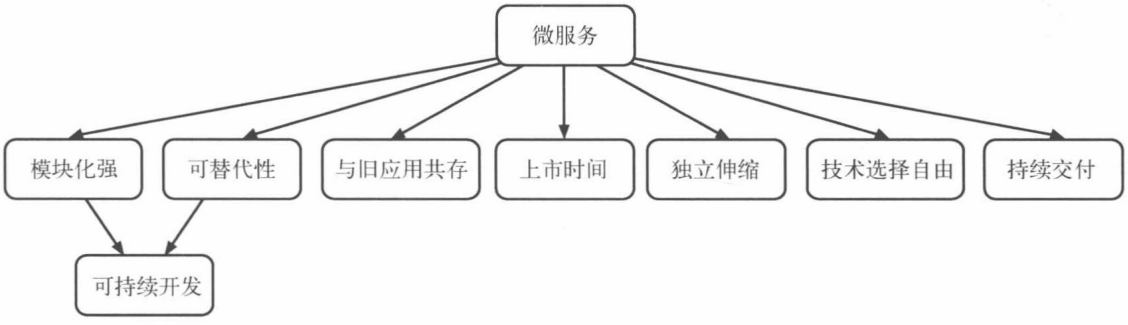


图 1-1 微服务的优势

模块化强

微服务是一种模块化很强的理念。系统如果由多个软件组件构成，例如多个 Ruby GEM 包、Java JAR 包、.NET 程序集、Node.js NPM 包，就容易混进一些不必要的依赖。举例来说，假设有

人在某个预期之外的地方引用了一个类或方法，这就产生了对该类或方法的一个依赖，但该类或方法的开发人员并不知情。开发人员对这个类或方法的任何修改，都可能会意外地导致系统另外一部分出错。这样的依赖会越来越多，问题也会越来越严重，甚至会导致系统无法继续运行或开发。

相反，微服务之间只通过消息、REST 等机制实现的显式接口相互通信，因此调用一个微服务要克服更高的技术障碍，从而降低了引入不必要依赖的可能性。原则上来说，单体部署软件也可以实现高层次的模块化。但实践经验告诉我们，单体部署架构会随时间而腐化。

可替代性强

与单体部署的模块相比，微服务更易于替代。其他组件通过显式接口来使用微服务。如果一个新的微服务所提供的接口和已有的微服务一样，那么它就能替代旧的那个。新的微服务可以在不同的代码库中实现，甚至采用不同的技术，只需提供一致的接口即可。这在传统的系统中通常是不可能实现或很难实现的。

小型的微服务更容易替代。开发软件系统时，开发人员通常会忽视未来更换代码的需求。谁会考虑开发中的系统在未来可能被取代呢？微服务的可替代性降低了错误决策的代价。如果技术或方案的选择仅局限于某个微服务，那么必要的时候可以采用其他技术或方案重新实现这个微服务。

可持续开发

微服务模块化强以及容易替代的特点，使软件开发可以持续进行。开发新项目通常是最简单的，但项目时间越长，生产效率就越低，一个重要的原因就是软件架构的腐化。微服务通过加强模块化来对抗软件的腐化。绑定技术过时以及难以移除旧的系统模块，都是阻碍单体部署持续开发的问题。微服务架构不绑定任何特定技术，可以通过对微服务逐个替代的方式来解决这些问题。

遗留应用的进一步开发

在开发遗留系统时，采用微服务架构是比较简单的：新功能在微服务中实现，无须将新功能加入到难以理解的遗留代码库，因此能够取得立竿见影的效果。新加入的微服务可以只响应特定的请求，其余的请求则转交给遗留系统来处理，转交之前还可以对请求进行处理。这样就不需要彻底更换遗留系统。另外，微服务可以用新技术来开发，不会被遗留系统的技术栈所束缚。

上市时间

微服务能缩短项目的上市时间。如前所述，微服务能够逐个部署到生产环境。假设在某大型系统中，每个团队负责一个或多个微服务，且功能的实现只需修改其负责的微服务，那么各个团

队就能够并行开发，而且功能部署到生产环境的过程中，也不需要与其他团队费时费力地进行沟通协调。因此，与单体部署相比，微服务可让多个团队并行开发更多的功能，并在更短时间内将其部署到生产环境。微服务架构将大团队划分为小团队，小团队分别负责各自的微服务开发，这有助于改进团队的敏捷过程。

独立伸缩性

每个微服务可以独立于其他服务进行伸缩（scale）。如果只有一小部分功能被频繁调用，那么只需对这部分功能进行扩容即可。这能极大地简化基础设施和运维工作。

自由选择技术

采用微服务架构进行开发时，不会限制具体采用的技术。这使得微服务架构能在单个微服务内测试新技术，而不会影响其他服务。由于引入的新技术或原有技术的新版本仅在受限的环境中被采用，测试风险会更小。微服务还能采用特定技术来实现特定功能，例如选择特定的数据库。如果出现问题，那么可以很轻松地替换或撤销微服务，因此风险相对较低。此外，新技术仅局限在单个或少数微服务中，因此降低了采用新技术的风险，并可以在不同的微服务上采用不同的技术。这使得尝试和评估新技术变得更容易，同时有助于提升开发人员的生产效率，避免技术平台落伍，还能吸引高水平的开发人员。

持续交付

微服务有利于持续交付。微服务很小，因而能简化持续交付流水线的实现。微服务能独立部署，部署一个微服务的风险低于一个单体部署系统。保证微服务的安全部署也更容易，例如可并行运行多个不同版本。对很多开发人员来说，便于持续交付是他们采用微服务的主要原因。

以上都是采用微服务的有力依据。哪个原因更重要则由具体情况而定。从业务角度考虑，改进敏捷过程和持续交付通常都很关键。第 4 章将详细分析微服务的优势及其优先级。

1.3 挑战

凡事都有两面性。第 5 章会论述引入微服务所带来的挑战，以及如何应对这些挑战。简而言之，主要的挑战如下。

- **隐藏的关系**。系统架构由服务之间的关系构成，但微服务之间的调用关系并不清晰，这增加了架构工作的挑战性。
- **不易重构**。太强的模块化导致微服务不易重构，因此微服务之间的功能转移会比较困难。采用微服务后就很难改变系统的模块结构。幸好，有一些技巧能够减少这样的问题。

- **领域架构的重要性**。将领域模块化并划分到各个微服务中是很重要的，因为这将决定团队的分工情况。领域架构层面的问题也影响着团队组织，只有稳定的领域架构才能保证微服务开发的独立性。模块化建立起来后就比较难改动，因此后面一旦出现错误就很难修正。
- **微服务运行的复杂性**。微服务系统需要部署、控制和运行许多的组件。这增加了运维的复杂性，以及系统运行所需的基础设施。微服务需要实现运维自动化，以免运维变得费时费力。
- **分布式系统的复杂性**。基于微服务的系统是一个分布式系统，这意味着开发人员将面对更高的复杂性。与进程内的调用相比，网络调用较慢，带宽也更小，因此微服务之间的调用可能会因网络问题而失败。

1.4　总结

本章概述了微服务的概念。本章从微服务的定义开始，回答了“为什么采用微服务”的问题，最后讨论了微服务相关的挑战。