微服务

设计原理与架构

MICROSERVICES

Design Principle and Architecture

郑天民 / 著

深入剖析微服务架构设计理念与技术体系

全面阐述实施微服务架构的系统方法与工程实践

涵盖向微服务架构转型的思路、过程和案例

中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

微服务

设计原理与架构

MICROSERVICES

Design Principle and Architecture

郑天民／著

人民邮电出版社

北京

图书在版编目（CIP）数据

微服务设计原理与架构 / 郑天民著. -- 北京 : 人民邮电出版社, 2018.5
ISBN 978-7-115-47882-5

Ⅰ. ①微… Ⅱ. ①郑… Ⅲ. ①互联网络－网络服务器－程序设计 Ⅳ. ①TP368.5

中国版本图书馆CIP数据核字(2018)第025531号

内 容 提 要

本书内容主要包含实施微服务架构的一些方法论和工程实践，首先，通过对微服务架构的基本概念、服务建模、服务拆分和集成的介绍，帮助读者全面理解微服务架构中的设计理念，然后从微服务架构的基础组件、关键要素、实现框架以及管理体系等维度出发，阐述实现微服务架构的工具和实践。最后，本书还给出了从现有系统向微服务架构转型的思路、过程和案例分析。

本书面向立志于成为微服务架构师的后端服务开发人员，读者不需要有很深的技术水平，也不限于特定的开发语言；不过，熟悉 Java EE 常见技术并掌握一定系统设计基本概念，有助于更好地理解书中的内容。同时，本书也可以供具备不同技术体系的架构师同行参考阅读，希望能给日常研发和管理工作带来启发和帮助。

◆ 著　　郑天民
责任编辑　刘　博
责任印制　沈　蓉　彭志环

◆ 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
固安县铭成印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：18.75　　2018 年 5 月第 1 版
字数：398 千字　　2018 年 5 月河北第 1 次印刷

定价：59.80 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前言

随着互联网行业的飞速发展，快速的业务更新和产品迭代给系统开发过程和模式带来了新的挑战。业务需求层出不穷且变化不断、技术发展日新月异、团队规模从无到有快速扩张，因此，系统的复杂性以及对行业变化的快速应变能力等成为软件开发的核心问题。围绕这些问题，如何更为合理地划分系统和团队边界、如何更加有效地组织系统开发过程、如何通过技术手段识别和消除开发过程中的浪费，成为广大软件开发和技术管理人员所需要思考的问题。在这一时代背景下，微服务架构的出现为我们提供了一种具体的解决方案。

本书从微服务的基本概念出发，阐述了微服务架构的方方面面。除了具体实现工具和框架之外，本书还介绍了微服务架构的基本原理和技术体系，并阐述现有系统向微服务架构转型的系统方法，旨在为广大开发人员提供一套系统的、全面的微服务实施指南。

构建微服务架构是一项系统工程，涉及服务建模、实现技术、基础设施和研发过程等各个维度。本书从建模、实现和转型这三个特定角度出发，结合作者自身在互联网行业多年的技术应用与管理工作经历展开论述，介绍微服务架构设计相关的方法论和工程实践，具有较强的针对性和适用性。微服务架构是一个非常广泛的概念，本书整体上是“原理”结合“技术”的行文思路，不仅仅介绍微服务架构实现上的具体工具，更是对这些工具背后的原理和设计思想进行剖析，具备一定广度的同时也提供了对应深度的知识体系。

本书共分为四篇，共计八章内容，分别从不同的领域对微服务架构的各个方面展开讨论。

1. **直面微服务篇**。从微服务的基本概念出发，阐述微服务架构的发展阶段、所具备的优势以及所面临的挑战，并给出实施微服务架构的系统方法。

2. **服务建模篇**。关注于微服务建模，首先介绍服务建模方法，用于明确服务模型的各个维度和表现形式；然后对服务拆分和集成方法进行展开，侧重于从服务的依赖关系、数据、事务边界等维度出发讨论实现策略。

3. **服务实现篇**。作为微服务架构实现过程中的主体知识部分，本篇从微服务架构基础组件、关键要素、实现技术和管理体系等四个角度切入，全面介绍微服务架构实现上的工具框架、技术原理和最佳实践。

4. **服务转型篇**。从实际应用角度出发探讨如何在现有系统的基础上向微服务架构转型，一方面提供技术架构调整的方法和模式，另一方面也阐述了如何从组织过程管理角度出发向微服务架构转型。

通过对本书的系统学习，读者将对微服务架构的基本原理、设计思想和实现方式有全面而深入的了解，为后续的工作和学习铺平道路。

本书的撰写成功要感谢我的家人，特别是我的妻子章兰婷女士，在我占用大量晚上和周末时间的情况下，能够给予极大的支持和理解。感谢以往以及现在公司的同事们，身处业界领先的公司和团队让我得到很多学习和成长的机会，如果没有大家平时的帮助，就不会有本书的诞生。

由于编写时间仓促，水平和经验有限，书中难免有欠妥和不足之处，恳请读者批评指正。

郑天民

2018 年 1 月于杭州钱江世纪城

目 录

第一篇
直面微服务

◆ 本篇内容

本篇从微服务的基本概念出发，阐述微服务架构的方方面面。我们经历了从单块系统到分布式系统的演变，而微服务架构基于分布式系统，也借助了面向服务架构和企业服务总线的设计理念，并做了改进和优化，从而形成一种全新的架构体系。

微服务架构一方面具备技术、业务和组织上的优势，另一方面也在技术架构和研发过程上存在较大挑战。微服务架构的实施需要具备一定前提，构建微服务架构是一项系统工程，涉及服务建模、实现技术、基础设施和研发过程等各个维度；也需要根据现有系统的具体情况采用合适的实施模式。

本篇共有一章，作为开篇总领全书后续章节。

第 1 章

直面微服务架构

随着近年来软件行业（尤其是互联网行业）飞速发展，一方面以电子商务、O2O、移动医疗、在线教育等为代表的互联网和互联网+应用对软件行业的业务结构和体系造成巨大冲击，另一方面，快速的业务更新和产品迭代也给系统开发过程和模式带来新的挑战。业务需求层出不穷且不断变化、技术发展和创新日新月异、团队规模从无到有快速扩张、系统的复杂性以及对行业变化的快速应变能力等成为软件开发的核心问题。围绕这些来自于外部与内部的压力和动力，如何更为合理地划分系统和团队边界、如何更加有效地组织系统开发过程、如何通过技术手段识别和消除开发过程中的浪费成为广大软件开发和技术管理人员需要思考的命题。

针对这些命题，我们的思路首先是根据业务划分系统和领域的边界，对一个大而全的系统进行分而治之，通过一些功能边界明确、业务高度抽象的模块或组件之间的组装去形成更大的业务体系。其次，在系统和领域内部，同样需要对业务体系进行合理建模。通过建立服务体系对服务进行一定拆分和集成，从而降低业务实现的复杂性，并提高服务交互的灵活性。再次，通过服务拆分和集成的手段也可以推动研发团队组织架构的优化，并促进系统持续交付工作的有效开展。围绕这些解决问题的思路，微服务架构（Microservices Architecture）为我们提供了一种具体的解决方案。

所谓微服务（Microservices），就是一些具有足够小的粒度、能够相互协作且自治的服务体系。每个微服务都比较简单，仅关注于完成一个具体业务功能并能很好地完成该功能，而这里的功能代表都是一种业务能力。构建微服务体系需要一套完整的方法论和工程实践，而微服务架构代表的是实现微服务体系的架构模式，即为我们提供了这些方法论和工程实践。通过微服务架构，软件开发过程能够得到改善，开发效率能够得到提高，从而创造更为优秀的产品和用户满意度。微服务架构的提出代表着一种新的架构设计风格和模式，从这个角度讲，微服务架构需要我们理解、学习并应用到日常开发过程中去。但是微服务架构又不是一种完全打破现有技术体系、从无到有所诞生的替代性架构，而是在现有面向服务架构、企业服务总线等思想和技术体系的基础上，伴随着持续交付、虚拟化和容器技术的发展自然而然产生的一种软件设计和架构模式。

微服务架构的基础是分布式系统。本章从分布式系统出发，围绕微服务架构的基本概念展

开，讨论微服务架构与现有架构体系之间的对比，并给出构建微服务架构的系统方法。同时，微服务架构有其优势，但在实施过程中也会面临各种各样的挑战，本章对微服务架构的这些特点也做了分析。

1.1 分布式系统

微服务架构首先表现为一种分布式系统（Distributed System），而分布式系统是对传统单块系统（Monolith System）的一种演进。为了更好地理解和掌握微服务架构的特点，本节对单块系统和分布式系统做简要介绍。

1.1.1 单块系统的问题

在软件技术发展过程的很长一段时间内，软件系统都表现为一种单块系统。时至今日，很多单块系统仍然在一些行业和组织中得到开发和维护。所谓单块系统，简单讲就是把一个系统所涉及的各个组件都打包成一个一体化结构并进行部署和运行。在 Java EE 领域，这种一体化结构很多时候就体现为一个 WAR 包，而部署和运行的环境就是以 Tomcat 为代表的各种应用服务器。图 1-1 所示的就是一个典型的单块系统。我们可以看到在应用服务器上同时运行着面向用户的 Web 组件、封装业务逻辑的 Service 组件和完成数据访问的 DAO（Data Access Object，数据访问对象）组件。这些组件都作为一个整体进行统一的开发、部署和维护。

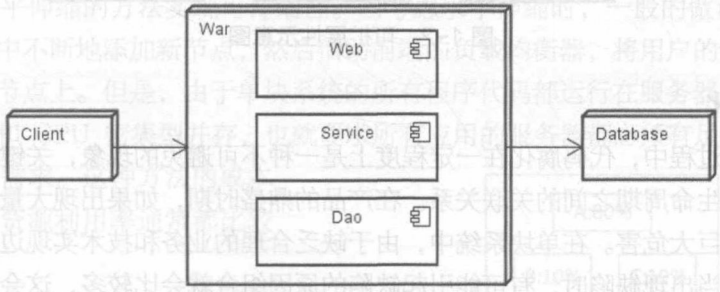

图 1-1 单块系统示意图

显然，图 1-1 所示的单块系统有其存在和发展的固有优势。当团队规模并不是太大的时候，一个单块应用可以由一个开发者团队进行独立维护。该团队的成员能够对单块应用快速学习、理解和修改，因为其结构非常简单。同时，因为单块系统的表现形式就是一个独立的 WAR 包，想要对它进行集成、部署以及实现无状态集群相对也比较简单，通常只要采用负载均衡机制并运行该单块系统的多个实例就能达到系统伸缩性要求。

但在另一方面，随着公司或者组织业务的不断扩张、业务结构的不断变化以及用户量的不断增加，单块架构的优势已逐渐无法适应互联网时代的快速发展，面临着越来越多的挑战。在使用单块系统时，我们不得不面对以下问题。

1. 业务复杂度

对于大多数系统而言，架构设计是为了满足业务需求的。衡量架构好坏与否的一个重要方面是看其面对复杂业务变更时所应该具有的灵活性，也就是我们通常所说的可扩展性（Extensibility）。可扩展性是指系统在经历不可避免的变更时所具有的灵活性，与针对提供这样的灵活性所要付出的成本进行平衡的能力。所谓可扩展，可扩展的是业务。即当往SystemA 中添加新业务 NewSubSystem 时，如果不需要改变原有的各个子系统而只需把新业务封闭在一个新的子系统中就能完成整体业务的升级，我们就可以认为系统具有较好的可扩展性，可扩展性示意图如图 1-2 所示。显然，单块系统不具备良好的可扩展性，因为对系统业务的任何一处进行修改，都需要重新构建整个系统并进行发布。单块系统内部没有根据业务结构进行合理的业务拆分是导致其可扩展性低下的主要原因。

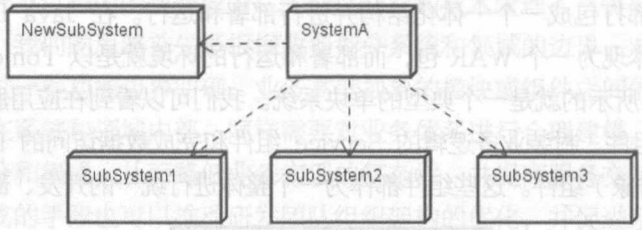

图 1-2　可扩展性示意图

2. 代码腐化

在软件开发过程中，代码腐化在一定程度上是一种不可避免的现象，关键是腐化的时间和程度与整个产品生命周期之间的关联关系。在产品的鼎盛时期，如果出现大量的代码腐化会对产品的发展带来巨大危害。在单块系统中，由于缺乏合理的业务和技术实现边界，随着产品业务功能的增多，当出现缺陷时，有可能引起缺陷的原因组合就会比较多，这会导致分析、定位和修复缺陷的成本相应增高，也就意味着缺陷的平均修复周期可能会花费更长时间，从而影响到产品的正常迭代和演进。同时，随着功能不断叠加，单块系统的代码结构也日益复杂，在开发人员对全局功能缺乏深度理解的情况下，修复一个缺陷的同时还有可能引入其他的缺陷，在很多技术团队并不具备完善的自动化测试机制的客观条件下，很可能导致问题越修越多的不良循环。

3. 团队问题

互联网行业由于产品价值与市场时机密切相关，普遍崇尚快速试错和迭代发布，很多公司或组织会在比较短的时间内扩充产品功能和开发团队，也就意味着对于过程资产建设和人才培养等方面并不会投入太多的成本，这就要求新加入团队的成员能够快速融入团队并进行代码开发和维护。然而，由于在单块系统中所有的业务和代码在很大程度上无序地混合在一起，存在大量错综复杂的业务和代码结构、由于历史原因所造成的迥然不同的开发风格以及看似复杂但已经不被使用的遗留代码，使得新员工了解行业背景、熟悉应用程序业务、配置本地开发环境等看似简单的任务都变得并不简单。

另外，单块系统的集中式管理方式，使得系统内部的技术体系和开发方式很难得到扩展。随着应用程序的复杂性逐渐增加以及功能越来越多，如果团队希望尝试引入新的框架和技术，或者对现有技术栈升级，通常都会面临不小的风险。也即意味着初始的技术选型严重限制了单块系统将来采用不同开发语言或框架的能力。但互联网行业中技术体系变化和业务体系变化一样快速，技术体系的无法扩展和升级也会导致现有团队中不同出身和开发背景的成员对系统代码产生一种开发惰性，也无法吸引到优秀的开发人员。

4. 伸缩性问题

前面讲到单块系统的可扩展性很差，实际上它的可伸缩性同样很有问题。所谓可伸缩（Scalability），伸缩的是性能，即当系统性能出现问题时，如果我们只需要简单添加应用服务器等硬件设备就能避免系统出现性能瓶颈，那么该系统无疑具备较高的可伸缩性。通常，我们会考虑采用水平伸缩的方法实现可伸缩性。当考虑水平伸缩时，一般的做法是建立一个集群，通过在集群中不断地添加新节点，然后借助前端的负载均衡器，将用户的请求按照某种算法分配到不同的节点上。但是，由于单块系统的所有程序代码都运行在服务器上的同一个进程中，内存密集型和 CPU 密集型并存，也就要求所有应用的服务器都必须有足够的内存和强劲的 CPU 来满足需求。这种方法的成本会比较高，而且资源利用率通常都比较低下。

以图 1-3 为例，单块系统中的组件 A 的负载已经达到了 80%，也就是到了不得不对系统运行能力进行扩容的时候。但同一系统的其他两个组件 B 和 C 的负载还没有达到其处理能力的 20%。由于单块系统中的各个组件是打包在同一个 WAR 包中的，因此通过添加一个

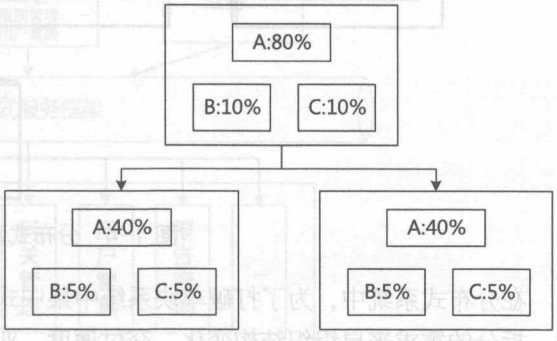

图 1-3 单块系统的伸缩性问题

额外的系统实例虽然可以将需要扩容组件的负载降低一半，但是显然其他组件的利用率变得更为低下，造成资源浪费。另外，对于那些需要保持类似会话（Session）数据的需求而言，扩容之后的运行机制在如何保持各个服务器之间数据的一致性上，也存在较大的实现难度。

针对以上集中式单块系统所普遍存在的问题，基本的解决方案就要依赖于分布式系统的合理构建。

1.1.2 分布式系统的基本特征

分布式系统，是指硬件或软件组件分布在不同的网络计算机上，彼此之间仅仅通过消息传递进行通信和协调的系统。我们从这个定义中可以看出分布式系统包含两个区别于单块系统的本质性特征：一个是网络，分布式系统的所有组件都位于网络之中，对于互联网应用而言，则位于更为复杂的互联网环境中；另一个是通信和协调，与单块系统不同，位于分布式系统中的各个组件只有通过约定、高效且可靠的通信机制进行相关协作才能完成某一项业务功能。这是我们在设计和实现分布式系统时首先需要考虑的两个方面。图 1-4 所示的就是从软件开发视图出发得到的一个典型的分布式系统，包含了分布式服务、消息中间件和分布式缓存等常见的用于构建分布式系统的技术实现方式。显然，这些工具位于一个封闭或开放的网络环境中，相互之间通过服务的注册和发现、消息传递、数据的缓存共享等机制完成协作。

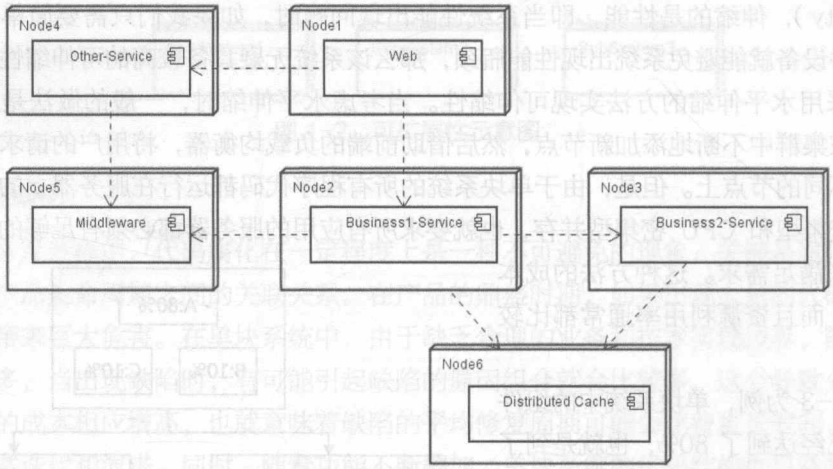

图 1-4 分布式系统示意图

在分布式系统中，为了打破单块系统中集中式的系统架构，我们引入系统拆分的思想和实践。拆分的需求来自组织结构变化、交付速度、业务需求以及技术需求所引起的变化。一般认为系统拆分的基本思路有两种，即纵向（Vertical）拆分和横向（Horizontal）拆分。

所谓纵向拆分，就是将一个大应用拆分为多个小应用，如果新业务较为独立，那么就直接将其设计部署为一个独立的应用系统即可。如图 1-5 所示，我们可以将移动医疗系统中的预约挂号业务拆分成订单、医院和用户等独立业务子系统。纵向拆分关注于业务，通过梳理产品线，将内聚度较高的相关业务进行剥离从而形成不同的子系统。

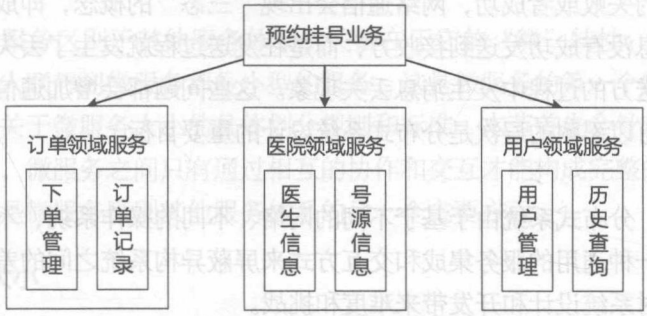

图 1-5　分布式纵向拆分示意图

相对于纵向拆分的面向业务特性，横向拆分更多地关注于技术。所谓横向拆分，就是通过将可以复用的业务拆分出来，独立部署为分布式服务，调用这些分布式服务，构建复杂的新业务。所以，横向拆分的关键在于识别可复用的业务，设计服务接口并规范服务依赖关系。横向拆分的基本实现方式是构建分布式服务体系，图 1-6 是对图 1-5 所示的预约挂号业务进行横向拆分的结果。可以看到，当我们把订单、医生、号源和用户等业务抽象成独立的垂直化服务，并在各个服务上层实现分布式环境下的调用和管理框架，系统的业务就可以转变为一种排列组合的构建方式。如基于订单和支付服务，我们可以构建出业务 1，而业务 2 可能只依赖于医院和用户管理服务。分布式服务框架提供了一种按需构建的机制，在保证各个分布式服务的技术、团队、交付独立发展的前提下，确保业务整合的灵活性和高效性。

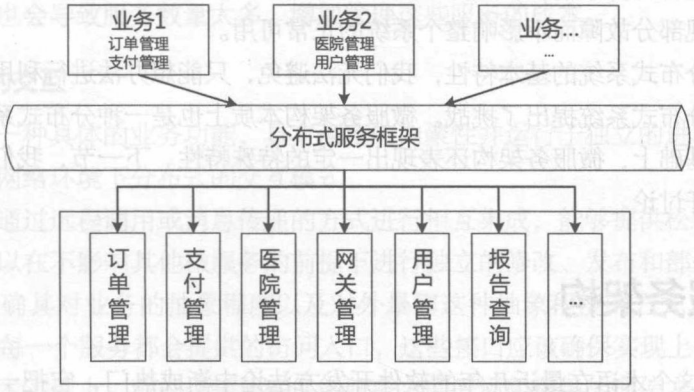

图 1-6　分布式横向拆分示意图

然而，分布式系统相较于单块系统而言具备优势的同时，也存在一些不得不考虑的特性，包括但不限于以下几点。

- 网络传输的三态性

构建分布式系统依赖网络通信，而网络通信表现为一个复杂且不可控的过程。相比于单机系统中函数式调用的失败或者成功，网络通信会出现“三态”的概念，即成功、失败与超时。由于网络原因，消息没有成功发送到接收方，而是在发送过程就发生了丢失现象；或者接收方处理后，响应给发送方的过程中发生消息丢失现象。这些问题都会增加通信的代价。如何使通信的代价降到用户可以忍耐的层次是分布式系统设计的重要目标。

- 异构性

相较单块系统，分布式系统由于基于不同的网络、不同的操作系统、不同的软件实现技术体系，必须要考虑一种通用的服务集成和交互方式来屏蔽异构系统之间的差异。异构系统之间的不同处理方式会对系统设计和开发带来难度和挑战。

- 负载均衡

在集中式系统中，各部件的任务明确。由于分布式系统是多机协同工作的系统，为了提高系统的整体效率和吞吐量，必须考虑最大程度发挥每个节点的作用。负载均衡是保证系统运行效率的关键技术。

- 数据一致性

在分布式系统中，数据被分散或者复制到不同的机器上，如何保证各台主机之间的数据一致性将成为一个难点。因为网络的异常会导致分布式系统中只有部分节点能够正常通信，从而形成了网络分区（Network Partition）。

- 服务的可用性

分布式系统中的任何服务器都有可能出现故障，且各种故障不尽相同。而运行在服务器上的服务也可能出现各种异常情况，服务之间出现故障的时机也会相互独立。通常，分布式系统要设计成允许出现部分故障而不影响整个系统的正常可用。

以上问题是分布式系统的基本特性，我们无法避免，只能想办法进行利用和管理，这就给我们设计和实现分布式系统提出了挑战。微服务架构本质上也是一种分布式系统，但在遵循通用分布式特性的基础上，微服务架构还表现出一定的特殊特性。下一节，我们将围绕微服务架构的特殊特性展开讨论。

1.2 微服务架构

微服务架构这个术语在最近几年的软件开发方法论中渐成热门，它把一种特定的软件应用设计方法描述为能够独立部署的服务套件。目前对于微服务架构还缺乏统一的标准化定义，