微服务体系结构实现框架综述

辛园园,钮 俊,谢志军,张开乐,毛昕怡
XIN Yuanyuan, NIU Jun, XIE Zhijun, ZHANG Kaile, MAO Xinyi

宁波大学 信息科学与工程学院,浙江 宁波 315211
Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, Zhejiang 315211, China

XIN Yuanyuan, NIU Jun, XIE Zhijun, et al. Survey of implementation framework for microservices architecture. Computer Engineering and Applications, 2018, 54(19):10-17.

Abstract: In order to improve the scalability of enterprise-level applications, the microservice-based software architecture can be used to divide a monolithic application into a set of small services which can cooperate with each other. This will allow us to develop these services freely, deploy them independently and maintain them easily, so it can satisfy the demand of business development better. At present, as specific solutions of the microservices architecture, some microservices frameworks have been successfully implemented by many large enterprise and open source. This paper mainly discusses and compares the concepts of service-oriented architecture, Web services and microservices, and presents the key technologies and core functional modules in the practice of microservices architecture. Then, it analyzes and compares the characteristics and core components of mainstream implementation frameworks of microservices architecture. Finally, it analyzes the challenges of microservices composition and service composition schemes in microservice framework, and summarizes this paper.

Key words: monolithic application; Service-Oriented Architecture(SOA); Web service; microservices; microservice framework

摘 要: 为提高企业级应用的可伸缩性,基于微服务的软件体系结构将单体应用细化为可相互协作、配合的一组小服务,使得服务间开发自由、独立部署、易于维护,更好地满足企业发展需求。目前,微服务框架作为微服务架构的具体实现方案,已被很多大型企业成功实施并开源。论述面向服务体系结构、Web服务及微服务相关概念并作比较;给出微服务体系结构实践中的关键技术以及核心功能模块;分析对比主流微服务体系结构实施框架及其核心部件的特征和差异;探讨微服务组合面临的挑战及微服务框架中的服务组合方案,并总结全文。

关键词: 单体应用;面向服务体系结构(SOA);Web服务;微服务;微服务框架
文献标志码: A **中图分类号:** TP391 **doi:** 10.3778/j.issn.1002-8331.1808-0014

1 引言

随着计算机技术、网络技术和通讯技术的疾速发展,人们需要分析、设计越来越多高效、可靠的大型软件系统,如可供全球用户使用的大型实时社交软件Twitter、Facebook、WeChat等,在线流媒体服务平台Netflix[1]以及在线电子商务平台如Amazon、Alibaba(淘宝)等。在设计阶段,良好的体系结构设计是大型软件系统能够正常运行的前提和基础[2]。

一般来说,软件系统的体系结构规定了系统各个构件单元的集合及构件之间交互、协作或通信应该遵循的规范或协议。一直以来,企业或开发人员都在努力寻找好的软件体系结构设计来构建应用系统,以期望满足企业业务需求,提高开发效率,方便业务扩展并能适应时代、技术发展的潮流。大量传统应用系统由于具备规模较小,结构简单,用户群体小及实时通信要求低等特点,通常采用传统的单体架构模式。所谓单体架构即指将

基金项目: 浙江省自然科学基金(No.LY15F020010,No.Y16F020002);浙江公益技术应用研究项目(No.2014C31059);软件工程国家重点实验室开放课题(No.SKLSE2014-10-05);教育部嵌入式与服务计算重点实验室开放课题(No.ESSCKF201302)。

作者简介: 辛园园(1993—),女,硕士,研究领域为服务计算、微服务,E-mail:xin_yuany@163.com;钮俊(1976—),男,博士,副教授,研究领域为服务计算、物联网、微服务等;谢志军(1974—),男,博士,副教授,研究领域为传感器网络、数据流、服务计算等;张开乐(1994—),男,硕士,研究领域为物联网、微服务;毛昕怡(1993—),女,硕士,研究领域为微服务。

收稿日期: 2018-08-06 **修回日期:** 2018-09-15 **文章编号:** 1002-8331(2018)19-0010-08

整个软件系统的功能模块及运行数据等作为整体看待，统一地设计、开发、打包及部署运行[3]。然而，对于需要应对业务逻辑复杂，数据量庞大和具有高实时性，高可靠性，高伸缩性运行要求的在线软件系统来说，单体架构存在较多不足。首先，单体应用的代码庞大，代码耦合度高，系统灵活性较差；其次，单体架构受技术限制，所有模块都采用相同技术，难以实现技术协同；第三，单体应用的创建或部署时间较长、成本较高，难以实现持续发布及部署。更为重要的是，单体应用难以被扩展，可伸缩性较差[4]。

近年来，随着云计算、容器虚拟化以及集成了开发、测试、部署和运营为一体的DevOps[5]等技术的兴起和发展，微服务受到工程界和学术界的极大关注，成为信息科学领域的重点研究对象之一[6]。其基本思想是将传统的单体应用按业务功能拆分为一系列可被独立设计、开发、部署、运维的软件服务单元，服务间彼此配合、相互协作以实现最终价值[4]。实际上，微服务可看作面向服务体系结构(Service-Oriented Architecture，SOA)[7]的一种具体实现。另一方面，与Web服务相比，微服务更具优势，例如采用去中心化管理及轻量级容器部署，提供监控预警及多种高可用容错策略，服务数据独立且开发自由。

微服务体系结构的思想将为企业级系统应用的实现提供指导，而微服务框架(Microservice Framework)则是微服务体系结构的具体实现及解决方案。它是企业在构建微服务时考虑将众多关键技术如服务部署、服务通信和服务发现等集成为一个整体，从而形成的一种框架或方案。实质上，微服务框架是众多优秀开发者已有实践经验的总结，企业在实践或运用微服务框架时，不仅可以提高开发效率，降低开发成本，还可以对框架进一步优化及拓展以满足多样化业务需求。目前，微服务框架已被越来越多的中大型企业成功实践并并取得较好的发展，较典型的有Amazon、Netflix、Uber等公司。本文首先回顾并对比了服务计算、Web服务及微服务的基本概念；第3章介绍了当前较为主流的微服务体系结构实现框架，重点就这些框架中的关键技术及核心部件展开详细分析及对比；第4章介绍微服务框架中服务组合模式存在的问题、服务组合应考虑的关键因素及组合方法研究现状，最后总结了全文。

2　Web服务与微服务基本概念

2.1　Web服务及基于微服务的体系结构

随着互联网技术的发展，大型IT系统一般采用分布式计算模式，以优化资源配置并提高系统可靠性、可用性和灵活性等。为了便于分布式信息系统的设计、开发与集成，以及提高系统架构的灵活性、复用性和可增长性，面向服务的体系结构SOA因此产生。SOA体系结构将定义良好的，具有开放接口并独立于软硬件平台以及实现技术的单个服务组件关联起来，以构造整体应用并采用松耦合的方式保护既有IT基础设施[7]。实际上，SOA只是一种架构思想，而Web服务及其相关标准和SOAP、WSDL、UDDI等协议的出现，则为SOA的具体实践提供了技术支撑和处理方案。Web服务基于SOA架构理念，采用一套标准技术实现了对企业间服务资源的共享和复用。SOA体系结构及Web服务等相关标准和技术的产生，为构造松耦合的大型分布式应用指明了较好的方向，并做了开拓性工作。

尽管Web服务为跨平台的企业开发提供了方便，但是在开发模式上，仍然采用的是单体架构。单体架构由于自身特点较适合小型应用的开发，并不适用于业务复杂度较高、业务需求量较大的中、大型企业。微服务体系结构思想的出现，则较好地解决了上述难题。其核心要义在于基于面向服务的思想，对传统大型应用系统进行功能分解，推动细粒度服务的使用。微服务架构(MicroServices Architecture，MSA)则指根据应用系统的业务需求，通过对预定义的微服务进行重组而形成企业级应用的分布式体系结构[8]。它主要将传统概念上的单体应用在功能、数据等方面进行分解，划分为多个具有明确边界并可被自由重组的小规模子服务。这些子服务间采用轻量级通信方式实现交互、协作，每个服务都有自己的数据库并可在独立进程中被部署、运行等，服务之间保持技术异构性，可由不同的团队选择合适的工具、语言进行开发。

与单体架构相比，微服务架构的优势在于：(1)微服务按业务功能划分，每个服务都具备特定的功能，易于开发、维护等；(2)每个独立的微服务可以由不同的语言基于不同的平台开发，灵活性较好；(3)子服务可独立部署，能够实现可持续集成及交付；(4)容错能力强大，单个微服务出现问题不会影响系统其他服务的运行；(5)可实现动态按需实时扩展等。目前，微服务体系结构的思想已被应用于很多大型公司的分布式应用系统中。

2.2　Web服务与微服务架构的比较

Web服务作为SOA的具体实现，遵循相应技术规范，为面向网络的分布式应用提供统一的服务注册、发现、调用等机制。尽管Web服务体系结构与微服务架构在实现服务治理功能上存在相似之处，但在架构本质设计方面仍存在一定差别(具体如表1)。实际上，Web服务体系重在解决异构资源的整合，通过企业服务总线ESB(Enterprise Service Bus)将不同服务集成到企业应用中。尽管ESB在企业级应用构建中获得成功，但其重

量级特性使其灵活性降低,不能很好地支持大型企业级应用的持续变更、发布及部署等。而微服务架构提倡轻量级的服务架构理念,在实践时采用去中心化的思想,并无Web服务中的重量级ESB,服务间的交互也采用如RESTful HTTP[3]这类轻量级协议。另一方面,在微服务架构中,服务是按业务功能划分,服务间彼此独立并具有独立数据库,因此,可单独部署运行在轻量级Docker容器[9]中实现敏捷性开发及部署,更适合现代企业的发展。

3　微服务体系结构实现框架

微服务体系结构为构建具有较高伸缩性需求的大规模应用系统提供了“蓝图”,在实施微服务架构时,还应具体给出服务发现、服务通信、服务容错、负载均衡和服务部署等解决方案[10]。到目前为止,工程界已出现若干基于微服务体系结构的实践框架(Microservice Framework,MF),如阿里巴巴集团的Dubbo[11]、微博团队的Motan[12-13]、Lightbend公司的Lagom[14],以及Google和IBM等共同开发的Istio等[15],这些微服务框架通常由服务注册与发现、负载均衡、服务网关和服务容错等核心部件聚合而成。本章基于以上构成微服务框架的核心部件及技术,对目前使用较主流的四种微服务框架即Dubbo、Motan、gRPC[16]和Spring Cloud[17-18]展开详细分析及比较。

3.1　微服务框架基本介绍

随着微服务体系结构的不断发展及成熟,微服务框架已逐渐成为一些中大型企业从传统架构转型到微服务架构的最佳化实践。在本文重点介绍的四种主流微服务框架中,根据服务调用方式以及功能特色可将它们分为RPC(Remote Procedure Call)型微服务框架和RESTful微服务框架两种[19-21],这些框架的基本特征如表2所示。

其中,Dubbo、Motan和gRPC属于RPC型微服务框架,这类框架可以像调用本地服务一样调用远程服务,从而实现高效可靠的网络透明传输。Dubbo和Motan属服务治理型RPC框架,主要提供丰富的服务治理功能。与Dubbo相比,Motan去除了Dubbo中一些不常用的组件和配置,功能虽不如Dubbo强大,但比较简单、易用。Dubbo自2017年重新维护后又增加了对Node.js、Python等语言的支持并与当当网维护的Dubbox进行了合并[11],增加了对RESTful的远程调用,并且在序列化方式等方面也进行了扩展。gRPC则可提供对Java、Objective-C、C++、PHP等多种语言接口的支持,但未实现较为全面的服务治理功能。Spring Cloud来源于Spring Boot框架[17],本质上是一种RESTful的微服务框架,设计时从资源的角度对系统进行拆分并为每个资源设置特定的URI。与另外三种RPC框架相比,Spring Cloud提供了搭建微服务体系结构几乎所需的全套功能,而在Dubbo中实际上只实现了Spring Cloud中服务治理的部分。Spring Cloud因其功能全面、部署方便且操作简单,是业界目前使用最广泛的微服务框架。

3.2　微服务框架关键技术及核心部件

微服务作为一种分布式系统的解决方案,为构建复杂的分布式微服务应用需提供服务注册与发现、服务间通信和服务部署等关键技术。同时,与传统面向服务体系结构的实施方案相比,微服务体系结构的伸缩性较好,更能满足大型应用的需求。因此,除需具备传统实施方案中基本的如服务注册与发现,服务组合等功能外,微服务体系结构还需某些扩充模块,并对已有模块进行扩展或优化。这些扩充模块主要包括负载均衡、API网关和容错机制等。

3.2.1　服务发现机制与注册中心

微服务提倡轻量级的服务架构理念,单个微服务一

表1　Web服务体系与微服务架构的比较

Web服务体系	微服务架构
企业级,自顶向下展开实施	团队级,自底向上展开实施
服务由多个子系统组成,粒度大	系统被拆分为多个服务,粒度细
企业服务总线,集中式开发	去中心化管理,松散的服务架构
集成方式复杂(ESB/WSDL/SOAP)	集成方式简单(HTTP/REST/JSON)
服务依赖性强,部署复杂	服务彼此独立,能单独部署

表2　微服务框架基本特征比较

微服务框架	Dubbo	Motan	gRPC	Spring Cloud
服务治理	√	√	×	√
多语言支持	√	×(支持php client和C server)	√	√
多注册中心	√	√	√	√
多容错方案	√	√	×(只Failover)	√
多序列化方式	√	√	×(只protobuf)	√
服务调用方式	RPC/RESTful	RPC	RPC	RESTful

般部署在轻量级容器如Docker中。在运行时，服务实例可能随时被克隆、销毁或重定位，故需要一种动态的服务发现机制。通常，在实现服务发现时，服务提供者需先将服务实例的地址信息注册到注册中心，注册中心则负责管理服务实例地址并提供心跳检查等机制，而服务发现组件作为服务调用方，通过注册中心实现服务查询并获得可用的服务实例地址列表。实际上，服务发现组件部署位置并不是固定的，一般可分为客户端发现和服务端发现两种[10]。

客户端发现中服务发现逻辑由客户端实现，客户端作为服务发现组件基于上述方法确定服务实例的网络地址。服务端发现则是把服务发现逻辑委托给了专门的路由服务。当客户端有服务请求时需将请求发送给路由服务，再由路由服务与注册中心交互，以发现满足请求的服务。与客户端发现相反，此方式并不需客户端设计服务发现逻辑，但需要额外的路由服务作为中间件，存在一定的性能损耗。

在微服务体系结构中，要实现服务发现，服务注册中心尤为重要。注册中心主要负责管理服务提供者和消费者的URL地址及路由信息等，并实现服务注册、发布、健康检查和故障排除等功能。如表3，对目前较为通用的服务注册中心进行了归纳和比较。

其中，Zookeeper是分布式系统使用较典型的注册中心，它提供统一命名、集群管理、状态同步、分布式应用配置管理等功能，并能很好地解决分布式系统数据一致性问题，较适合于大型分布式系统应用[22]。Etcd一般仅存储一些键值对数据，故适用于少量数据的情形。Consul是Google公司开发及维护的注册中心，与Zookeeper、Etcd相比，它的功能更为全面，如内置了服务发现功能，在实现时也不需依赖其他插件，更容易部署。Redis属于key-value存储服务器，主要以事件的发布、订阅实现服务发现，而Multicast主要采用广播的形式发布、订阅消息，但这种组播的方式受网络结构因素影响较大，比较适合小型应用使用。Netflix Eureka属于RESTful的微服务注册与发现组件[23]。与Zookeeper、Redis等注册中心相比，Eureka在实现服务发现时优先保证服务可用性并且属于轻量级组件，易于部署、维护。

3.2.2　负载均衡

在微服务体系结构中，为了提高伸缩性，单元微服务通常具有多个进程级服务实例，服务实例的个数随着服务请求的数量而动态变化。当服务请求数量增多时，相应的服务实例个数有可能随之增加。相反地，服务实例也可因服务请求数量的减少而被销毁。因此，需要通过某种负载均衡算法，将服务请求分发至特定的服务实例进行处理。

最简单的负载均衡算法来自著名的Round Robin算法，即轮询法[24]。其思想是将多个可用服务实例组织成一个循环队列，然后根据实例顺序轮流分派服务请求。该方法一般仅适用于服务实例处理能力相差不大的情形，然而要应对多个在性能、负载能力方面差异较大的服务实例时，可采用加权轮询法[25]。该算法在简单轮询的基础上，根据服务实例的性能及负载能力为其设置不同的权重，随后将请求按顺序及权重分配到合适的服务实例上。实际上，上述轮询算法并未真正考虑服务实例的实际请求连接数（会话数）及当前负载，较好的方法是采用最小连接数算法（Least Connections scheduling，LC）[26]。该算法可根据服务实例当前的连接数，动态的选取连接数最小的服务实例处理服务请求，以提升服务实例的利用率及请求负载能力。此外，负载均衡算法还包括按请求随机分配的简单随机算法、按key值分配的哈希算法等[27]。

在MSA中，负载均衡通常与服务发现组件部署在一起。在实现时，负载均衡作为服务消费方，维护服务发现获得的服务地址列表并基于负载均衡算法实现服务请求的具体调用。通常，根据负载均衡分布的位置，可将其分为服务端负载、客户端负载及独立进程负载三种。

服务端负载将服务负载交由一个独立的负载均衡器处理（如图1(a)）。目前，较典型的负载均衡器有LVS、Nginx、HAProxy和F5等，这些负载均衡器位于客户端与服务端之间，并作为一个独立的服务，主要维护服务端服务实例地址列表，此时的地址列表是运维人员预先为服务配置指向负载均衡器的DNS实现。该方案实现较简单，但负载均衡器作为中间件很容易造成单点瓶颈问题并存在一定性能损耗。

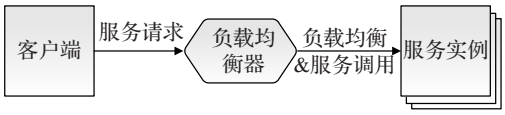
客户端负载（如图1(b)）主要针对服务端负载的不足改进，将负载均衡功能集成到了服务消费方进程内，负载能力转由客户端提供。与服务端负载不同的是，此时所有客户端维护的服务实例列表均来源于注册中心。该方案解决了服务端负载单点瓶颈的问题，但也带

表3　服务注册中心比较

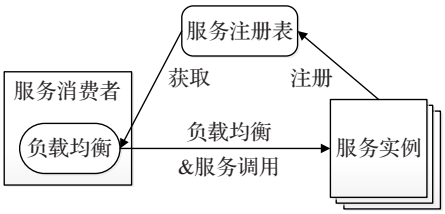
特征	Zookeeper	Etcd	Consul	Redis	Multicast	Netflix Eureka
主要算法	Paxos	Raft	Raft	Raft	组播路由算法	—
健康检查	支持	—	支持	支持	支持	支持
CAP理论	CP	CP	CA	CP	CP	AP
支持框架	Dubbo/Motan/gRPC..	gRPC	Motan/gRPC..	Dubbo	Dubbo	Spring Cloud

来了基于不同技术栈的客户端实现负载均衡开发难度较大等问题。

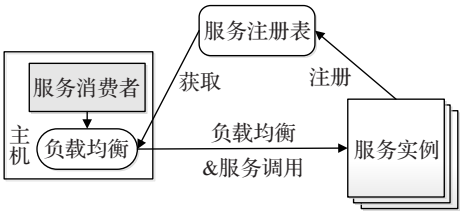
独立进程负载是将客户端与负载均衡作为两个独立进程运行在同一个主机中(如图1(c))。当有客户端请求时直接发送至负载均衡独立进程,再由负载均衡独立进程采用类似客户端负载的方法实现服务请求调用。此方案综合了服务端及客户端负载的优缺点,但存在部署较复杂、不易维护等问题。



(a)服务端负载均衡



(b)客户端负载均衡



(c)独立进程负载均衡

图1　微服务框架负载均衡模式

3.2.3　服务网关

在MSA中,服务网关主要位于网络的边缘并作为外部服务请求的统一入口,主要提供身份认证与安全检查、请求路由管理、服务组合、请求分派与维护、压力检测、负载均衡等功能,其基本结构如图2所示。通常,当有用户发来服务请求时,网关需先对服务请求进行权限验证、API监控、流量控制等过滤操作,随后由智能路由实现服务请求的具体转发及调用等。实际上,服务网关可作为服务发现和负载均衡组件,对于服务发现模式中客户端和服务端发现也都可由它实现[10]。其中,对于客户端发现,服务网关主要为客户端提供对服务注册中心的访问,此时网关只是充当一个访问代理,对于服务端发现,服务网关则简单地充当一个路由器。

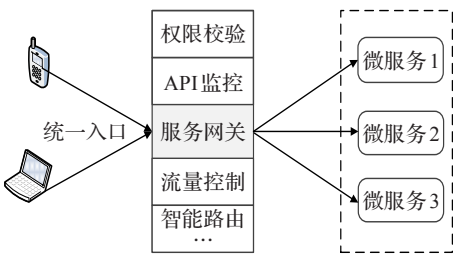


图2　服务网关基本结构

随着微服务相关技术的日趋成熟,服务网关作为微服务体系结构中比较重要的组件,目前已存在多种技术选型。例如,Netflix Zuul[28]是目前使用较典型的服务网关组件,它由Netflix开发,主要作为协调客户端与微服务的中间层并提供权限验证、压力检测、负载分配、动态路由等较为全面的服务网关功能。实际上,Zuul主要负责处理RESTful的服务请求及调用,然而在不同微服务业务场景下,仍存在外部客户端是RESTful的接口请求,而内部服务之间却是RPC通讯的情况。显然,为同一系统实现两套不同类型的API接口太过繁琐,gRPC Gateway则解决了这一问题[29]。gRPC Gateway为gRPC框架的服务网关插件,它可读取gRPC服务请求并为其生成反向代理服务器,最终实现将RESTful的HTTP/JSON API转化为内部gRPC的形式,从而解决了服务内外接口不兼容的问题。

3.2.4　服务容错

在微服务体系结构中,通常存在多种服务间调用及依赖关系。微服务实例之间的动态调用链可能会因某个服务实例响应超时、发生错误或负载过高等原因导致多个相关联的微服务不可用,因此需要容错机制。

目前,常见的微服务框架均提供了服务容错方案。其中,最基本的是处理服务超时,如超时重试机制[8]。它主要对服务请求设置超时响应时间,并根据设置的服务请求时间和记录的服务请求次数,决定是否进行重试操作。若服务因负载过高发生故障时,可采用限流和熔断器两种策略。其中,限流主要对服务进行限流处理,可分为控制服务请求并行数量和限制访问速率两种。熔断器则会记录、监测服务执行情况,当某个实例不可用时,可立即更换至新的服务实例,也可通过计算失败率,当其达到某个阈值时拒绝接收服务请求并将其直接返回[10]。若服务发生错误不可用时,可采用回退机制,主要有快速失败、故障沉默和自定义处理三种策略[30]。同时,为保证服务发生错误不会影响到其他服务资源可采用故障隔离机制如壁仓隔离模式,它可为不同的服务分配不同的进程池,实现服务间的隔离[8]。企业在实践微服务框架时,也可根据自身需求自定义容错方案。

3.2.5　服务部署与通信

微服务框架除具备上述核心部件,还应重点考虑服务部署及服务间通信的问题。与Web服务相比,单个微服务实例的部署更为灵活方便,通常在微服务框架中服务部署方式一般有三种:其一,多服务实例部署于单个虚拟机或物理机中。这种方式下实例间资源共享,资源利用率高,但彼此之间隔离性较差。其二,单服务实例运行于单独虚拟机中,服务实例间实现彻底隔离,系统

伸缩性较好，但也容易导致资源浪费及实例维护效率降低等问题。其三，单个微服务实例运行于单个轻量级容器如Docker中，服务实例可被灵活部署并保持彼此间较好隔离，且可降低部署难度并提高效率。

鉴于微服务采用分布式系统结构，各个微服务基于上述服务部署方式被部署在不同的节点上，而服务间的通信则是通过网络传输，因此微服务框架在实现时应保证良好的通信机制，从而实现准确、高效的信息交换。通常，在微服务框架中服务间交互有同步请求响应以及异步通信两种模式。在同步请求响应模式下，客户端发出通信请求并等待回应，一般采用REST和Thrift两种协议。这种通信模式没有中间件作代理，通用性较好，实现较简单，但其通信机制较为单一，只支持请求/响应模式，在一定程度上降低了系统可用性。异步通信模式则不需等待请求响应，其通信模式一般基于消息队列，通过消息中间件缓存消息，可实现客户端与服务端的松耦合。该模式可采用如AMQP、KAFKA等多种协议[8]，并支持请求/异步响应、发布/订阅和发布/异步响应等多种通信机制，有效地提高了系统的可用性。异步通信模式更适合较复杂的微服务应用场景并可实现较为可靠的消息传递，但其引入了消息队列中间件，在一定程度上加重了消息系统开发、部署和运维的复杂性。通常，在主流微服务框架中，会混用这两种通信模式以提高系统可伸缩性。

3.3　微服务框架关键技术及核心部件比较

在主流微服务框架中，Dubbo、Motan和gRPC都支持多种注册中心，同时基于不同的负载均衡软件可实现如随机、轮询、一致性哈希等多种负载均衡算法[31]（具体如表4）。Spring Cloud则集成了一系列用于构建分布式系统所需的开发工具包，其中负载均衡软件采用的是Netflix Ribbon[32]，与Nginx这类负载均衡软件相比，Ribbon更关注于服务实例的请求并发处理能力，而不只是简单的请求转发。例如Ribbon支持的BestAvailableRule算法，可动态的感知服务实例的负载情况，并选择当前最小并发请求数的服务实例分发请求。对于容错方面，Spring Cloud集成了Netflix Hystrix作为容错组件[33]。它综合考虑了服务超时、负载、错误及服务隔离等多种情形，主要提供线程隔离、服务降级和熔断器等多种容错技术。Hystrix也可被其他组件集成使用，例如Netflix Zuul实际上集成了Hystrix用于网关层的内部容错处理。Dubbo、Motan和gRPC则更关注于服务超时、错误的情形，提供内置集群容错方案。另外，在服务网关方面，Spring Cloud和gRPC自身就可提供网关组件，而服务治理型框架中Dubbo和Motan主要侧重于服务内部接口之间的RPC调用，本身并不提供服务网关功能，实际使用时可结合开源的服务关组件如Netflix Zuul、Kong[34]作为补充。

4　微服务框架中的服务组合方案

在微服务体系结构中，基于单一职责原则，将传统的单体应用进行有效的拆分，以最终实现敏捷开发及部署。目前，主流微服务框架中所提供的服务注册与发现、服务容错和服务通信等核心技术为实现敏捷性开发和部署提供了相应的技术支撑，然而在应对多样性的外部服务请求时还需重点考虑微服务组合技术，即如何将多个功能明确、单一的微服务通过某种机制，聚合成满足应用需求、功能更为完整的整体应用以提供服务。

在微服务组合方面，一种直观的做法就是借鉴Web服务组合的思路和策略。在Web服务组合中，通常采用Choreography（编排）或Orchestration（编制）两种服务聚合策略，并分别采用CDL（Choreography Description Language）语言、BPEL（Business Process Execution Lan-

表4　主流微服务框架中关键技术及核心部件比较

微服务框架	Dubbo	Motan	gRPC	Spring Cloud
注册中心	Zookeeper/Redis/Multicast	Zookeeper/Consul	Zookeeper/Consul/Etcd	Netflix Eureka
负载均衡位置	客户端/服务端	客户端	客户端/服务端	客户端
负载均衡软件	LVS/Nginx/HAProxy	Cluster	Apache/Nginx/HAProxy	Netflix Ribbon
容错部件	集群容错	集群容错	集群容错	Netflix Hystrix
服务网关	—	—	gRPC Gateway	Netflix Zuul
服务通信	同步/异步调用	异步调用	同步/异步调用	同步/异步调用
容错策略	Failover/Failfast/Failsafe /Failback /Forking/Broadcast/ Available/Mergeable	Failover/Failfast	Failover	请求封装/跳闸机制/资源隔离/服务监控/回退机制/自我修复
负载均衡算法	Random/LeastActive/ RoundRobin/ConsistentHash	Random/LocalFirst/ ActiveWeight/Round Robin/ConsistentHash/ ConfigurableWeight	Random/RoundRobin/ ConsistentHash	Random/Retry/ZoneAvoidance/ AvailabilityFiltering/ RoundRobin/BestAvailable/ WeightedResponse/ WeightedResponseTime

guage)语言来描述服务之间的交互、协作或通信过程。尽管服务编排、编制语言在Web服务组合中获得成功,但难以直接被应用至微服务组合中。事实上,上述两种语言属于底层基于句法的描述语言,随着交互服务数量的增加将导致组合服务代码复杂性的增大。另一方面,它们要求单元服务有定义良好的接口以及交互信息的强类型约束。但是,快速变化的微服务使得难以快速定义其接口并且难以被快速部署。同时,前述两种语言需要有中心执行引擎如ESB等,难以被用于微服务体系结构中[35]。

在实现微服务组合时有几个方面需重点考虑:首先,微服务提倡轻量级的服务架构理念,在实现服务组合时应避免引入企业服务总线这类重量型执行引擎,同时在设计服务组合方法时应使其能适用于轻量级的微服务部署及执行。其次,微服务促进了服务的动态性,在运行时服务实例的地址等信息都是动态变化的,因此服务组合方法应能动态绑定服务地址等信息,并能应对服务因响应超时、发生错误等意外情况导致不可用的情况。再者,微服务是无状态的,服务本身并不存储特定信息,而服务间的通讯也仅通过语言无关的API进行,因此如何根据服务请求匹配到一组满足需求的服务组合,是需重点考虑的一个方面。当前,学术界、工程界在微服务组合方面进行了探索研究,在主流微服务框架中也提供了实现服务组合的具体解决方案[35-39]。例如,Spring Cloud通过轻量级的事件驱动机制来实现服务组合,在实施时利用一个事件处理中介如RabbitMQ或Apache Kafka来协调组合服务调用[17]。当有外部服务请求时将直接发送服务调用事件至事件处理中介,事件中介将自动完成服务的具体调用并返回一个包含执行结果的服务执行事件到事件处理引擎,再由事件处理引擎根据服务业务规则触发满足服务请求的后续服务的调用。另外,文献[35]也提出了一种基于事件驱动的轻量级服务组合平台Medley,但这个轻量级平台在实现服务组合时重点侧重于对组合服务的描述。该平台为微服务组合设计了一种专门的服务组合描述语言,并通过编译器为组合描述生成可执行代码,最终部署运行在Medley平台上,从而实现对大量的现有服务进行组合。除此之外,Netflix公司为实现服务组合专门设计了一个可用于生产环境的开源框架即Netflix Conductor[36]。它通过将系统所需的微服务和系统服务结合起来,一起定义在一个工作流中,从而形成一个完整的实现某种特定功能的服务组合蓝图。通过事件触发工作流,实现预定义服务的具体调用。类似的,文献[37]也提出了一种基于工作流的微服务描述和验证的形式化框架,并指定了一种基于Petri网的微服务的形式化语义。本质上讲,这种方法实际上也是一种工作流形式的微服务组合方法,但其主要完成微服务组合方案的可行性验证,更偏重于对微服务组合的形式化建模。

5 结束语

目前,国内外对微服务体系结构及实现机制的研究异常活跃。本文首先论述了微服务体系结构的产生背景、相关概念,并指出传统单体架构的不足及微服务体系结构的发展背景及优势。其次,对微服务架构实践关注的服务注册与发现、服务通信、服务部署等关键策略进行了分析,重点就主流微服务实施框架及其核心部件展开详细分析和对比,可为基于微服务体系结构的企业级应用开发提供实施框架的选择、帮助与指导。最后,探讨了微服务组合相关问题。随着持续发布及部署需求的增加及轻量级容器如Docker的推广,微服务体系结构将更为广泛地被应用于分布式系统的设计及实施。微服务组合作为微服务体系结构中较为重要的一个方面,目前的研究却基本集中于通过预定义服务组合工作流模型以实现服务组合的方式。因此,下一步将重点考虑高效的在线动态微服务组合机制的研究。

参考文献:

[1] Leung A,Spyker A,Bozarth T.Titus:Introducing containers to the Netflix cloud[J].Communications of the ACM,2018,61(2):38-45.

[2] van Vliet H,Tang A.Decision making in software architecture[J].Journal of Systems and Software,2016,117:638-644.

[3] Dragoni N,Giallorrnzo S,Lafuente A,et al.Microservices:Yesterday,today,and tomorrow[J].Present and Ulterior Software Engineering,2017,4:195-216.

[4] Thönes J.Microservices[J].IEEE Software,2015,32(1):116.

[5] Ebert C,Gallardo G,Hernantes J,et al.DevOps[J].IEEE Software,2016,33(3):94-100.

[6] Balalaie A,Heydarnoori A,Jamshidi P.Microservices architecture enables devops:Migration to a cloud-native architecture[J].IEEE Software,2016,33(3):42-52.

[7] Papazoglou M P,Heuvel W J.Service oriented architectures:Approaches,technologies and research issues[J].VLDB Journal,2007,16(3):389-415.

[8] Newman S.Building microservices:designing fine-grained systems[M].[S.l.]:O'Reilly Media,Inc,2015.

[9] Boettiger C.An introduction to Docker for reproducible research[J].ACM Sigops Operating Systems Review,2015,49(1):71-79.

[10] Montesi F,Weber J.Circuit breakers,discovery,and API

gateways in microservices[J].arXiv preprint arXiv:1609.05830,2016.

[11] Dubbo TEAM.Apache Dubbo[EB/OL].[2018-07-24].http://dubbo.apache.org/.

[12] 李庆丰.Motan:支撑微博千亿调用的轻量级RPC框架[EB/OL].[2018-07-24].https://blog.csdn.net/heyc861221/article /details/80125762.

[13] Weibocom.motan[EB/OL].[2018-07-24].https://github.com/weibocom/motan.

[14] Lagom Team.Lagom[EB/OL].[2018-07-24].https://www.lagomframework.com/.

[15] Istio Team.Istio Preliminary 1.0[EB/OL].[2018-07-24].https://istio.io/.

[16] Grpc Team.GRPC[EB/OL].[2018-07-25].https://grpc.io/.

[17] Spring Team.Spring Cloud[EB/OL].[2018-07-24].https://springcloud.cc/.

[18] Cosmina I.Spring microservices with spring cloud[M]//Pivotal Certified Professional Spring Developer Exam.CA:Apress,Berkeley,2017:435-459.

[19] Wang X,Wang S,Hao Z,et al.Research on the construction of regional credit bank platform based on microservices[C]//Proceedings of the 10th International Conference on Measuring Technology and Mechatronics Automation,2018:452-456.

[20] Nguyen T.Benchmarking performance of data serialization and RPC frameworks in microservices architecture:gRPC vs.Apache Thrift vs.Apache Avro[Z].2016.

[21] Sill A.The design and architecture of microservices[J].IEEE Cloud Computing,2016,3(5):76-80.

[22] Hunt P,Konar M,Junqueira F P,et al.ZooKeeper:wait-free coordination for Internet-scale systems[C]//Proceedings of USENIX Annual Technical Conference,2010.

[23] Netflix.Eureka[EB/OL].[2018-07-24].https://github.com/Netflix/eureka.

[24] Anurag U,Hitesh H.Optimization in round robin process scheduling algorithm[M].New Delhi:Springer India,2016:457-467.

[25] Li T,Baumberger D,Hahn S.Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin[J].ACM Sigplan Notices,2009,44(4):65-74.

[26] Choi D J,Chung K S,Shon J G.An improvement on the weighted least-connection scheduling algorithm for load balancing in Web cluster systems[M]//Grid and Distributed Computing,Control and Automation.Berlin,Heidelberg:Springer,2010:127-134.

[27] Pattanaik P A,Roy S,Pattnaik P K.Performance study of some dynamic load balancing algorithms in cloud computing environment[C]//Proceedings of the 2nd International Conference on Signal Processing and Integrated Networks,2015:619-624.

[28] Netflix.Zuul[EB/OL].[2018-07-24].https://github.com/Netflix/zuul.

[29] Gajek F.API diversity for microservices in the domain of connected vehicles[Z],2018.

[30] 阿里云.微服务架构实践之服务容错[EB/OL].[2018-07-24].https://www.aliyun.com/jiaocheng/793252.html.

[31] 马原.基于RPC的高并发网络通信中负载均衡的研究[D].杭州:浙江理工大学,2017.

[32] Netflix.Ribbon[EB/OL].[2018-07-24].https://github.com/Netflix/ribbon.

[33] Netflix.Hystrix[EB/OL].[2018-07-24].https://github.com/Netflix/hystrix.

[34] Kong.Kong Community Edition(CE)[EB/OL].[2018-07-24].https://konghq.com/kong-community-editio.

[35] Yahiae B H,Réveillère L,Bromberg Y D,et al.Medley:An event-driven lightweight platform for service composition[C]//Proceedings of International Conference on Web Engineering,2016:3-20.

[36] Gómez B.CONDUCTOR,the newest tool from Netflix for orchestration of microservices[EB/OL].[2018-07-24].https://en.paradigmadigital.com/dev/conductor- newest- thing-ne tflix-orchest ration-microservices/.

[37] Camilli M,Bellettini C,Capra L,et al.A formal framework for specifying and verifying microservices based process flows[C]//Proceedings of the International Conference on Software Engineering and Formal Methods,2017:187-202.

[38] Guidi C,Lanese I,Mazzara M,et al.Microservices:A language-based approach[J].Present and Ulterior Software Engineering,2017:217-225.

[39] Alshuqayran N,Ali N,Evans R.A systematic mapping study in microservice architecture[C]//Proceedings of the 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications,2016:44-51.