DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年

面向微服务软件开发方法研究进展

吴化尧[1] 邓文俊[1]

[1]（南京大学计算机科学与技术系 南京 210023）

Research Progress on the Development of Microservices

Wu Huayao[1], Deng Wenjun[1]

[1]（*Department of Computer Science and Technology*, *Nanjing University*, *Nanjing* 210023）

Abstract Microservices are the latest, and probably the most popular, technology to realize the well-known Service-Oriented Architecture (SOA). They have been widely applied in many important industrial applications, and have also attracted increasing attentions in academia. In order to aid the effective development of high quality microservices, in this study, we present a systematic review of the microservices literature, focusing on the various software engineering activities in the development of microservices. Specifically, we collected and analyzed the currently available methods, tools, and practices for the requirements analysis, design and implementation, testing, and refactoring for microservices. We also discussed issues and opportunities in future researches of this field.

Key words microservice; software development; research progress

摘要 微服务是面向服务体系结构的最新发展趋势和研究热点，其不仅在工业实践中形成了广泛且重要的应用，在学术界也受到日益增长的关注。本文以软件工程生命周期中的各项活动为主线，系统全面地对当前的微服务软件开发方法进行梳理和总结，尤其分析并讨论了面向微服务软件开发在需求分析、设计和实现、测试、以及重构上的已有方法、工具和实践，并给出一些该领域的未来研究方向，从而为更加科学有效地开发高质量微服务提供参考和借鉴。

关键词 微服务；软件开发；研究进展
中图法分类号 TP311

为了满足企业动态多变的业务需求、提高开发效率和方便业务扩展，软件工程研究人员在单体应用程序开发的基础上提出了面向服务软件开发方法。面向服务软件开发使用相互独立的服务作为构建应用程序的基本单元，这可以在不影响系统运行的情况下对服务进行增删和修改，从而实现软件应用程序的快速构建和动态调整。此外，每个服务都可以独立开发并选用最合适的技术体系，有助于提升软件开发的效率和系统性能。

微服务是面向服务软件开发的最新发展趋势，其在面向服务的基础上进一步降低了系统的耦合度[1]。微服务通常采用去中心化的服务管理方式，每个服务都具有很强的可伸缩性，且能够快速部署和更改。微服务还充分借鉴了云计算、容器技术以及 DevOps 等新的实践方式，已成为当前软件工程研究的一大前沿和热点。此外，面向微服务软件开发方法在工业界也得到了很多成功的应用，例如腾讯的微信[1]和优步的产品架构[2]等。

目前，已有一些研究工作对面向服务软件开发方法进行了总结[2]，但在遵循这些已有的理论和方法之

[1] https://segmentfault.com/a/1190000014171624
[2] https://blog.didispace.com/uber-micro-service-action/

收稿日期：2019-06-30
基金项目：国家重点研发计划（2018YFB1003800）
通信作者：吴化尧（hywu@nju.edu.cn）

外，微服务本身具有的一些新特性也要求人们设计相应的软件开发方法。本文因此尝试对面向微服务软件开发方法进行全面系统的总结，以帮助研究者和从业者了解该领域的最新研究进展。具体地，我们首先使用系统文献调研（Systematic Literature Review）方法收集当前与面向微服务软件开发相关的研究论文；随后，分别在需求分析、设计与实现、测试、以及重构这四个软件工程生命周期的关键活动上总结已有的方法、工具和实践；在此基础上，讨论面向微服务软件开发未来的研究方向。

本文第 1 节描述微服务相关的背景知识，第 2 节介绍使用的文献收集方法以及展示文献汇总结果，第 3、4、5 和 6 节分别从需求分析、设计与实现、测试、以及重构的角度来总结微服务软件开发的当前研究进展，第 7 节根据研究现状给出一些未来研究方向，第 8 节对全文进行总结。

1 背景

软件架构定义了应用程序的组件结构及其之间的相互关系[3]。随着客户需求的不断演化，应用程序通常会变得庞大且复杂，这就需要在软件开发时定义并选择最合适的架构方式来尽可能地以最小成本满足客户的各种功能和非功能性需求[4]。

单体架构将整个应用程序部署为一个统一的解决方案，其中组成它们的模块无法独立运行[5]，采用这一架构开发的软件被称为单体应用程序。单体架构的优势在于其易于开发、测试和部署，但单体架构只适用于小规模应用的开发，随着应用程序的增长，单体应用程序会变得难以理解和修改，也很难对开发团队成员进行调整。此外，单体应用程序中对任何小组件的更新都必须重新测试和部署整个应用程序，使得持续开发变得非常困难；单体应用程序还会产生技术锁定，其中所有组件都必须坚持一开始就选择的框架和技术[6]，使得最终实现的系统难于达到最优的状态。

为了克服单体架构的潜在缺陷，满足应用程序对于可扩展性、持续开发和技术选择自由等需求，面向服务架构（Service Oriented Architecture，SOA）应运而生。这一概念由 Gartner 公司在 1996 年首次提出[7]，它将整个应用程序分解为若干个相互独立、自包含、可重用的服务，使得整个应用程序具有动态、松耦合和分布式的特性。其中，每个服务是实现特定业务能力的与平台无关实体，通过良好定义的接口和按照统一的标准进行通信，并可以对其进行描述、发布、发现和动态组合[8]。

SOA 使得系统组件（即服务）之间的结构和相互关系便于理解，同时通过对服务的独立增删和修改来动态适应不断变化的市场环境和需求。与单体架构相比，SOA 主要具有下述三项重要优势：

1）松耦合。在一个面向服务应用程序中，不同服务间的关系是松耦合的，即服务不严格依赖于其他服务，因此可以在不影响其他服务的情况下对服务进行增加、删除和修改以适应不断变化的需求[9]。例如，对于单体架构而言，对部分代码进行修改都需要对整个系统进行测试，这在很大程度上增加了维护的成本；而在 SOA 的测试阶段只需要测试修改过的服务，避免了重复测试。

2) 模块化和可重用。SOA 使用一批设计为可复用的服务作为构建应用程序的基本模块[5]，其中单个服务实现特定的功能，不同服务之间可以进行组合来实现更强大的功能。不同的系统可以重用已有的服务来满足自身需求，从而减少软件的重复开发和管理。

3）技术独立。服务之间相互独立，不同的服务可以使用不同的技术栈（如编程语言、数据库等）来分别实现。服务也不需要特定的框架或硬件来支撑，具有很好的可移植性。

微服务架构可以视为面向服务软件架构（SOA）的一个特定子类型，其不仅可用于构建单个软件应用程序，也可用于将单体应用程序迁移为微服务应用程序。与传统 SOA 通常被视为集成解决方案、依赖于企业服务总线或其他同类重量级中间件等产品[10]不同，微服务仅依赖于轻量级技术；同时，微服务的服务粒度更小，采用更加通用且轻量的接口，更强调分布式特性（如负载均衡、服务发现等），也更适应 DevOps 和容器等技术。

1.1 面向服务架构

面向服务架构（SOA）涉及三种不同的参与者，分别是服务提供者、服务消费者和服务注册中心[7]。服务提供者提供服务并将服务注册到服务注册中心，服务消费者通过服务注册中心获取和使用服务，而服务注册中心作为中间平台联通服务提供者和消费者。

SOA 通常使用基于 XML 的简单对象访问协议 SOAP（Simple Object Access Protocol）、Web 服务描述语言 WSDL（Web Services Description Language）、统一描述、发现和集成协议 UDDI（Universal Description, Discovery, and Integration）和 Web 服务工业标准（WS-）等相关技术构建动态、松散耦合的分布式系统[11]。其中，SOAP 是一种数据交换协议，主要描述了三种参与者之间传输消息的格式以及这种消息通过何种方式进行传输；WSDL 则用于描述 Web 服务以及服务消费者如何访问该服务的具体接口；

UDDI 则用于对 Web 服务进行注册和检索。

1.2 微服务架构

有研究者认为微服务一词由 Lewis 和 Fowler 在 2014 年首次提出[12,13]，不过据其所述，这一概念至少在 2012 年之前就已存在[14]。它同样指的是一种架构风格，即将单个应用程序开发为一套小型服务（即微服务）的集合，其中每个服务都在自己的流程中运行，并以轻量级机制（例如 HTTP）进行通信。这些服务围绕业务功能构建，可通过全自动部署机制独立部署而几乎无需集中管理，也可以使用不同的编程语言进行编写，并使用不同的数据存储技术管理自身数据[14]。

微服务架构在软件开发上主要有两种应用，其中一种是从需求分析出发，从无到有的开发一个新的微服务应用程序，本文第 3、4 和 5 节将分别从需求分析、设计与实现、测试这三个角度来梳理和总结相关研究的当前研究进展。另一种应用是将已有系统（通常是单体应用程序）重构到微服务架构，使得系统在摆脱部分原有缺陷的同时享受到微服务带来的优势，本文第 6 节将讨论这种应用的当前研究进展。

2 文献收集和汇总

本文采用系统文献调研方法来全面系统地收集所有与微服务软件开发相关的研究论文。具体的，我们利用 ACM Digital Library、IEEE Xplore、ScienceDirect、Springer Link 和 DBLP 这五个英文数据库，以及中国知网和万方知识服务平台这两个中文数据库对 2019 年 6 月之前发表的微服务开发相关文章进行检索。其中，我们使用表 1 中的中英文关键词分别在上述数据库中进行检索。

Table 1 Keywords of Literature Search

表 1 论文检索关键词

英文文献	("microservice" OR "micro service") AND "software engineering"
	("microservice" OR "micro service") AND " requirement"
	("microservice" OR "micro service") AND " design"
	("microservice" OR "micro service") AND " development"
	("microservice" OR "micro service") AND " test"
	("microservice" OR "micro service") AND ("refactor" OR "migrate" OR "migrating" OR "migration")
中文文献	“微服务” AND “软件工程”
	“微服务” AND “需求”
	“微服务” AND “测试”
	“微服务” AND “开发”
	“微服务” AND（“重构” OR “迁移”）

然后，对于各个数据库中检索到的文献，应用下述标准来进行筛选。我们将符合全部以下条件的文献考虑在内（即包含标准）：

- 关注软件工程生命周期（需求分析、设计与实现、测试）和重构中一个或多个阶段
- 发表形式为会议、期刊、书籍等

将符合以下任一条件的文献排除在外（即排除标准）：

- 仅将微服务开发作为例子来进行讨论
- 以中文和英文之外的语言发表

最后，应用滚雪球方法对筛选后的文献进行进一步扩充，即对包含文献的所有参考文献都应用上述标准进行筛选，补充发现新的相关文献，并重复这一步骤直到不再加入新的文献为止。

我们最终收集了 89 篇与面向微服务软件开发方法相关的研究论文。图 1 给出了这些论文在软件工程生命周期的各项关键活动上的分布情况，其中设计与实现占比 53%，是当前研究最多的子领域。图 2 进一步给出了这些不同领域的逐年分布图。

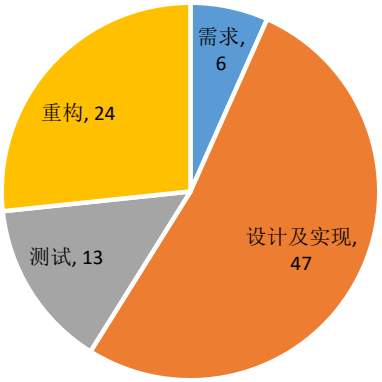

Fig. 1 Distribution of software engineering activities addressed by research papers

图 1 研究论文在软件工程问题上的分布

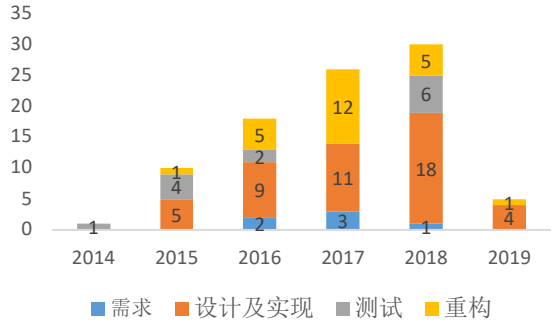

Fig. 2 Annual number of publications of research papers

图 2 研究论文的逐年发表数

3 面向微服务的软件需求分析

软件需求通常分为功能需求和非功能需求。功能需求是软件产品要实现的功能，用户利用这些功能来完成特定的任务；非功能需求决定软件产品的技术规格，它们刻画了软件在应用时的相关约束和特性[15]，是软件系统的重要质量属性[16]。

表 2 给出了与需求分析相关的 6 篇论文中所涉及的需求分析对象、功能需求来源、以及考虑的非功能性需求。我们发现在功能需求的获取上，半数研究直接引用已有需求[17,18]或者分析类似应用的需求[19]，而未涉及需求分析相关方法和流程的使用。我们同时也发现，由于微服务架构的引入对应用程序的非功能属性带来新的挑战和要求，使得现有研究愈发重视应用程序的非功能需求。在已有的 6 篇涉及需求分析的研究中，半数未考虑非功能需求[19,20]或者考虑了但没有给出具体说明[18]。其余的研究均考虑且直接给出对应非功能需求分析结果。

具体地，Richter 等人[17]在将电子座位预订系统迁移到微服务过程中将微服务架构的特点转化为一致性、容错、可扩展性和效率、负载均衡、可移植性、部署和可维护性等非功能需求。在一致性上考虑避免为不同顾客提供相同的座位以及禁止多次预订同一座位，在容错上考虑容忍多个服务实例、虚拟机或基础设施组件的故障并自动从故障中恢复等，在可扩展性上考虑服务、虚拟机和服务器的水平可伸缩性以及应用程序的较高负荷上限。Wizenty 等人[21]在远程护理应用程序的开发中考虑了可扩展性、安全性、可用性需求，在可扩展性方面要求能够灵活地集成和提供新的功能以及能够同时处理上千个应用实例，在安全性方面重点考虑了对用户个人数据的保护，同时在高可用性的基础上增加弹性、健壮性和功能独立性。Schneider 等人[22]在互联汽车解决方案实现中考虑了可扩展性、可交换性、重用、持续部署、代码质量等需求，在可扩展性上要求能够以有效的方式复制使用的更加频繁或者资源更紧张的组件，在重用上要求划分的服务将能在多个项目中使用，在持续部署上要求添加到服务中的特性必须立即可见以便能够自动测试与其他服务的兼容性。

由上述分析可知，面向微服务的软件需求分析相关研究在数量上较少，且这些研究表现为开发或者迁移特定应用程序研究的一部分内容，其中绝大多数研究对于功能需求和非功能需求的分析局限于直接给出结果而缺乏对于具体方法或者流程的讨论。

Table 2 Current researches on requirement analysis of microservices

表 2 微服务需求分析的已有研究

文献	需求分析对象	功能需求获取	考虑的非功能需求
[17]	将电子座位预订系统迁移到微服务	已有系统的需求	一致性、容错、可扩展性和效率、负载均衡、可移植性等
[18]	在线电影流式订阅系统	需求数据库	来自需求数据库
[19]	公共投诉系统	类似应用的设计文档和相应文献	未考虑
[20]	监督控制和数据采集（SCADA）系统	未说明来源	未考虑
[21]	远程护理应用程序	参与式设计方法	可扩展性、安全性、可用性
[22]	互联汽车解决方案	未考虑功能需求	可扩展性、可交换性、重用、持续部署、代码质量等

4 面向微服务的软件设计与实现

面向微服务的软件设计与实现是微服务开发研究中的重点，相关研究论文占据了目前所有论文中超过一半的比例（如图 1 所示）。尤其在工业界中，在面向微服务的设计上已形成了一系列公认的架构模式，如使用 API 网关来管理外部请求和进行服务组合、在客户端或者服务端实现服务发现、以及每个服务配备一个数据库来进行数据存储等[23]。

在 SOA 中，关于架构设计模式的研究已经较为成熟，为了验证 SOA 中已有的设计模式能否适用于微服务当中，Bogner 等人[24]总结了五条微服务特定原则（有界上下文、去中心化、轻量级通信、单一系统、技术异质性），然后定性分析了 Erl 等人以及 Rotem-Gal-Oz 的三本著作[25,26,27]中提及的 118 种 SOA 模式是否违反这些标准以判断是否适用于微服务。他们发现分别有 74 种（63%）、30 种（25%）和 14 种（12%）模式分别完全适用、部分适用和不适用于微服务，可见 SOA 与微服务在设计上具有许多共性。

在本文中，我们将微服务应用的整个设计空间看成是诸多相关设计领域的集合，这些设计领域不仅涵盖了上述公认的架构模式，还有与微服务密切相关的

技术如容器、DevOps 等。我们参考 Haselböck 等人在设计领域上的一次微服务领域专家访谈研究[28]，再加上对其他文献的调研，将微服务系统的整个设计空间划分为两类主要的设计领域：系统设计、组织结构和流程。表 3 给出了设计领域的分类，以及每个子设计领域需要考虑的具体因素以及对应文献。下面我们将对每一个子设计领域的已有研究进行介绍。

Table 3 Design domain of microservice software

表 3 微服务软件设计领域

设计领域	子设计领域	具体因素	文献
系统设计	接口	如何设计微服务内部以及微服务之间通信的接口	[21][29][31-36]
	数据管理	如何设计每个微服务的数据库、限制微服务对数据库的访问能力、保证数据在全局上的一致性等	[38][39]
	模块化	如何划分微服务	[29][42-46]
	服务发现	如何将已有服务注册到服务注册中心并供其他服务调用	[21][31][34-35][47-50][52-53]
	服务组合	如何将已有的多个独立的、功能明确的服务进行组合	[54-57]
	容错	如何在服务运行时检测故障并从中恢复	[47]
	安全	如何保证外部对于系统的访问安全以及系统内部服务之间相互调用的安全	[21][32-35][52-53][60-61]
	负载均衡	如何保证同一服务的不同实例之间负载均衡	[47][53][62-63]
	监控和日志	如何完成监控指标的确定，监控指标对应数据以及日志信息的收集、存储和分析	[50][64-67]
组织结构和流程	开发团队组织	如何组织微服务开发团队	[69]
	部署	如何对微服务进行部署	[21][29][33-34][53][71-72][74-80]
	DevOps	如何在微服务开发和运维中运用 DevOps 流程	[81][83]

4.1 系统设计

系统设计关注于微服务架构的设计，包括接口、数据管理、模块化（服务设计，包含服务划分、粒度等）、服务发现、服务组合、容错、安全、负载均衡、监控和日志等的设计。

（1）接口

微服务接口设计需要考虑如何设计微服务内部通信以及微服务之间进行通信的接口，包括采用何种通信机制、提供什么样的接口形式、每个微服务提供多少接口以及接口提供什么功能或者数据等[28]，并且将这些记录在接口规格说明书中。

就通信机制而言，微服务内部和微服务之间通信具有不同的特点，因此在通信机制的选择上也有所不同。对于微服务内部通信，其具有交互频繁、传输数据量不确定、接口数量多等特点，在实现上较为复杂。针对这一问题，Yan 等人[29]认为需要使用拥有高吞吐量、低时延、可扩展、易用性的 RPC 框架，他们还在跨平台性、服务治理、成熟度和性能上比较了 Dubbo、Thrift 和 gRPC 三种主要的 RPC 框架接口上的不同特点。对于微服务之间通信而言，主要有同步请求/响应模式和异步消息通信模式两种方式，且这两种方式各有优劣：

- 同步请求/响应模式无需中间件，具有较好的通用性，但是通信机制较为单一，只支持请求/响应模式，在一定程度上降低了系统可用性[30]，主要实现有基于 HTTP 协议的 Restful API，基于 RPC 和序列化支持多种消息格式的 Thrift，以及二进制格式的 Protocol Buffer 和 Avro。目前研究主要使用基于 HTTP 协议的 Restful API[21,31,32,33,34,35,36]来进行服务之间的通信。
- 异步消息通信模式无需等待请求响应，支持发布/订阅、发布/异步响应和请求/异步响应等多种通信机制，提高了系统的可用性，主要实现有 AMQP 的 RabbitMQ 和 Apache 的 Kafka[33]，此外还需要接口管理规范来表明接口如何被调用和管理，以及对关系复杂的接口进行监控和维护，目前主要有 3 种比较主流的 Restful API 管理规范：Swagger、API Blueprint 和 RAML[29]。Yan 等人[29]还从成熟度、活跃度、使用范围和支持语言数等角度比较了这三种接口管理规范，认为 Swagger 是目前最适合的微服务接口描述规范和业务平台微服务化的接口管理规范。

在接口这一微服务设计领域，Haselböck 等人[37]进行了一项关于微服务 API 管理的设计空间分析和决策文档的案例研究，识别了用于组织 API 管理任务

的设计决策，并提出了一个设计决策模型，该模型包括该领域的关注点、可选的设计选项、设计选项对应优势和缺陷、设计选项所需的系统组件和实现技术。

（2）数据管理

数据管理中的数据指的是服务完成特定功能所需的数据。SOA 在数据管理与单体架构类似，一个系统使用一个数据库，而微服务中每个服务独立拥有一个存储自身所需数据的私有数据库（或私有数据表），这个数据库的类型可随微服务的不同而有不同的选择。这一领域关注如何设计每个微服务的数据库、限制微服务对其他服务数据库的访问能力、保证共享数据在全局上的一致性等。

由于微服务之间采用的数据库类型可能不同，要保证共享数据在全局上的一致性必须实现跨异质数据库复制数据。针对这一问题，Viennot 等人[38]提出了 Synapse，一个易于使用的用于集成大规模、数据驱动服务的框架，它可以支持多种 SQL 和 NoSQL 数据库之间的数据复制（实现跨异质数据库复制数据），使得依赖于它的服务以相互隔离的、可伸缩的方式共享数据从而确保不同服务中共享数据的一致性。

此外，微服务架构引入了许多远程过程调用、Restful API 或消息传递，以便跨不同的流程和服务器将服务组合在一起，但是这在一定程度上也放大了数据丢失的风险[39]。针对这一问题，Messina 等人[39]提出了数据库即服务的架构模式，即将数据库功能作为一个微服务（数据库集成业务逻辑），使得服务与数据之间严格耦合，每当数据库需要扩展其功能时，可以直接在数据库中嵌入实现该扩展功能的业务逻辑，由此服务可以直接访问数据(无需第三方库也不存在网络问题)，在降低架构的复杂性和相关风险的同时也在速度和可伸缩性方面获得更多的改进。

（3）模块化

在 SOA 中，作为模块的服务的大小原则上可以处于小型应用程序服务到非常大的企业服务之间，而事实上 SOA 中的服务通常是大型产品甚至子系统[40]。与之相比，微服务是小型，细粒度的服务。微服务架构将单个应用程序开发为一套小型服务，但如何将单个程序划分为若干个满足以上特性的微服务存在许多问题。

为此，Yan 等人[29]提出了水平拆分加垂直拆分的服务划分方式，水平拆分按照不同的业务域进行拆分，拆分的边界充分考虑业务的独立性和专业性，垂直拆分把一个业务功能里的不同模块或者组件进行拆分，从而拆分成具有不同扩展需求和迭代频率的服务。

此外，领域驱动的设计（Domain-Driven Design，DDD）[41]是目前研究中进行服务拆分的常用方法。DDD 是一个模型驱动的软件开发方法，是一系列软件设计实践、技术和原则的集合，在诸多领域中具有广泛的实践和研究。DDD 提供了将领域模型解耦成有界上下文并表达它们之间的关系的功能[41]，每个有界上下文自然地成为一个微服务的候选，因此 DDD 与微服务设计非常契合。但是，由于一些 DDD 本身的特点，将 DDD 直接用于微服务中会带来许多问题：

- 由于 DDD 中的有界上下文和领域模型缺乏对语义知识表达的支持，并且领域模型在系统架构上是分散的，因此当 DDD 用于微服务设计时会产生语法、结构和语义上的理解或转换问题。基于此，Diepenbrock 等人[42]研究了基于本体的微服务架构领域驱动方法，提出了基于 DDD 原理建模微服务的元模型，该模型包括领域模型的概念、它们对有界上下文的分配以及有界上下文之间共享模型的定义，然后使用语义功能扩展了这个元模型，使得能够表达领域模型和共享模型的语义以及它们在有界上下文之间的关系，从而使得领域驱动的微服务设计中能够表达语义关系。
- 由于 DDD 并非一个形式化的建模语言，它使用了非正式的 UML 类图来表达领域模型，这在一定程度上阻碍了对于模型的验证和转换。为了解决这一问题，Rademacher 等人[43]为领域驱动的微服务建模提供初始 UML 配置文件，其不仅为 DDD 提供原型和约束，还支持将结构化领域模型表示为 UML 类图，解决了模型的验证和转换问题。
- 由于 DDD 缺乏现有软件开发过程方法中的软件开发过程描述和分类，在微服务设计中难以应用 DDD 的概念和方法。为此，Steinegger 等人[44]提出了一个 DDD 开发过程来根据需求构建微服务并将重点放在需求分析、设计、实现和测试上，随后将其应用于一个案例研究当中，并指出了其中存在的问题和缺陷。随后他们进一步提出了领域驱动的基于设计的微服务软件开发活动[45]，然后根据 DDD 的要求介绍了构建基于微服务的应用程序所需的软件开发活动，最后将其应用到案例研究中并探讨其中的局限性。

此外，Rademacher 等人[46]还讨论了将 DDD 应用于微服务时存在的三个挑战，分别是从领域模型中推导服务、基础设施组件建模以及在自治团队中进行领域模型建模，他们从领域驱动的角度讨论了这些挑战并根据模型驱动的开发提出了应对措施。

（4）服务发现

服务发现分为客户端发现和服务端发现，其对应逻辑分别位于客户端和专门的路由服务（即负载均衡

器）中[30]。这一领域主要关注如何将已有服务注册到服务注册中心、并且提供供其他服务调用的具体方式。

服务发现在 SOA 中已存在很多方法（如基于服务质量、语法、语义的服务发现方法），这些方法通常也可用于微服务架构。然而，由于微服务中服务个数大幅度增加且 SOA 与微服务之间所用技术栈存在很大差异，因此适用于 SOA 的服务发现方法未必全都适用于微服务。

在服务发现这一微服务设计领域，Haselböck 等人[47]在现有文献的基础上提出了一个设计决策模型，该模型包括该领域的关注点、可选的设计选项、设计选项对应优势和缺陷、设计选项所需的系统组件和实现技术。

此外，为了解决微服务发现中的问题以及提升微服务发现的性能，研究者在对已有问题进行讨论的同时也给出了相应的解决方案。

例如，针对容器中运行的微服务的服务发现和通信问题，Stubbs 等人[48]提出了一个可扩展的解决方案 Serfnode，Serfnode 是一个 Docker 镜像并可以包含多个 Docker 容器，新加入的容器镜像形成一个分散的 Serfnodes 集群，并在其中广播自己以供其他服务发现。

针对云平台时代下微服务数目成倍增长、且在给定需求和优先级的条件下难以在海量服务中找到最合适的服务的问题，Franca 等人[49]提出了一个从技术，社会（如同行的使用信息和评论）和语义三个角度来挑选云端微服务的框架 DIRECTOR，结果表明其可以通过发现和比较微服务来支持软件获取，并且能够在数百个候选对象中推荐出最适合相关需求的微服务。

针对传统的单节点部署在容错能力上表现较差的问题，Tang 等人[50]提出了一个分布式的服务发现机制，该机制通过服务发现数据的特点来改进 Raft 算法（解决分布式系统环境下的一致性问题的算法）从而确保集群中节点服务发现数据的高度一致性，从而提高服务发现的可用性。结果表明该机制不仅能够满足服务发现数据的强一致性要求，而且能在出现错误后更快地恢复数据的一致性。

目前，在服务发现领域已有很多开源软件产品，包括 Eureka、Etcd、Consul、Zookeeper[51]和 Redis 等[33]，目前研究主要使用 Eureka 来实现服务发现功能[21,31,34,35,52,53]。

（5）服务组合

服务组合关注如何将已有的多个独立的、功能明确的服务组合成满足应用需求且功能更为完善的整体应用以提供更加复杂和强大的功能。

服务的组合有编排和协同两种方式。服务编排是指通过中央调解器（如服务使用者或集成中心）协调多个服务，而服务编排是指在没有中央调解器的情况下协调多个服务调用[40]。SOA 同时支持这两种方式并且长期将 WS-BPEL 和 WS-CDL 语言分别作为编排和协同的主要方式。与之相比，微服务架构的去中心化文化和高度独立性使其更倾向于选择服务协同这种方式[5]。

目前，研究者和企业已经提出了一些服务组合方法，有的还开发了相应的服务组合系统、工具或者平台。在服务组合方法上，Spring Cloud 通过轻量级的事件驱动机制来实现服务组合，并利用一个事件处理中介如 RabbitMQ 或 Apache Kafka 来协调组合服务调用；Netflix 的服务组合开源框架 Netflix Conductor[54]将系统所需的微服务和系统服务结合并定义在同一个工作流中，形成一个完整的服务组合蓝图来实现某项功能。Camilli 等人[55]还提出了一种基于工作流的微服务描述和验证的形式化方法，用于对微服务组合的形式化建模，并指定了一种基于 Petri 网的微服务的形式化语义，以一种工作流形式的组合方法来对微服务组合方案的可行性进行验证。

在服务组合平台上，Yahia 等人[56]开发了一个用于服务组合的事件驱动轻量级平台 Medley，其基于用于描述编排的特定领域语言和生成高效代码的编译器，评估表明其可以在主流服务器和嵌入式设备上进行扩展，同时消耗较少数量的资源。

此外，在目前大规模视频应用中，由于使用的服务选择方法往往在未考虑细粒度的在线服务能力的同时也未考虑视频任务的特点，使得服务组合的整体效率下降。针对这一问题，Zhang 等人[57]为基于微服务的视频云计算平台提供了一种性能感知服务路径选择方法以发现合适的服务来进行组合，他们首先通过一个性能评估模型来考虑微服务实例的处理能力、视频处理任务的特点以及微服务实例之间的数据传输条件，然后基于性能模型使用最短路径算法来搜索和更新最佳微服务组合路径。

（6）容错

容错使得软件系统在运行时能够检测故障并从故障中恢复以防止故障对系统运行造成过多影响，这要求系统能够快速检测故障和从故障中恢复。微服务涉及的故障通常是无法访问到服务（服务崩溃或者网络延迟）、服务过载、以及微服务发生错误等。

在容错这一微服务设计领域，Haselböck 等人[47]在现有文献的基础上提出了一个设计决策模型，该模型包括该领域关注点、可选设计选项、设计选项对应优势和缺陷、设计选项所需的系统组件和实现技术。

目前，Hystrix[34,52,53]是常用的容错开源框架，其综合考虑了服务超时、负载、错误及服务隔离等因素，

是 Spring Cloud 中解决交互时超时和服务容错的组件。 Hystrix 使用命令模式包装服务调用逻辑，每个命令在单独线程中执行，提供了断路器、调用超时设置和资源隔离三方面功能保障服务可靠性[53]。

（7）安全

目前研究在安全相关的设计与实现上主要关注两方面的问题，一方面是外部对于系统访问的安全性，另一方面是系统内部服务之间相互调用的安全性。

在外部对于系统访问的安全性上，目前研究主要通过 API 网关[21,32,33,34,35,53,58]来进行身份验证和授权。API 网关实现的功能较多，其不仅能够封装微服务系统内部架构，为各个客户端提供统一接口，负责处理来自客户端的请求，还能根据请求调用单个微服务或者通过服务编排调用多个微服务并返回结果，即起到服务发现和组合的作用。此外，API 网关还能够对请求和服务的响应进行负载均衡，实现缓存、路由、访问控制、服务代理、监控和日志等。

目前微服务开发常用的开源服务网关包括 Netflix 的 Zuul，Mashape 的 Kong、Tyk 和 gRPC 等[33]。OAuth2[59]也常用于保障外部对于系统的访问安全，这是一种常用的专门针对跨平台的应用之间授权而设计的框架协议。

Fu 等人[60]分析了传统访问控制技术在微服务环境中的局限性和不足，提出了一种基于 RBAC(一个基于角色的访问控制模型)的访问控制优化模型，与现有方案相比，该方案提高了访问策略的表达能力，在一定程度上降低了计算成本，提高了微服务的安全性和运行效率。

此外，在系统内部服务之间相互调用的安全性上，网络中错综复杂的情况使得调用并不安全（微服务之间不能完全信任彼此）。针对这一问题，Yarygina 等人[61]提供了一个开源原型安全框架，该框架使用 MTLS（Mutual Transport Layer Security）、自托管 PKI（Public Key Infrastructure）和安全令牌建立信任和保护微服务通信。

（8）负载均衡

一个微服务通常具有多个实例，负载均衡将来自服务请求分配到同一服务的特定实例进行处理，保证每个实例处理的请求数目大致上保持一致。

在负载均衡领域的设计指导上，Haselböck 等人[47]在现有文献的基础上提出了关于负载均衡的设计决策模型，该模型包括该领域的关注点、可选的设计选项、设计选项的优势和缺陷、设计选项所需的系统组件和实现技术。

负载均衡可进一步分为客户端负载均衡和服务端负载均衡，其中客户端负载均衡由客户端提供支持，而服务端负载均衡则需要使用一个独立的负载均衡服务，其需要维护服务实例地址列表[30]，在收到请求后将请求平均分发给对应服务实例。目前 Ribbon 是常用的客户端负载均衡工具[53]，API 网关的请求转发和微服务间的调用等实际上都是通过 Ribbon 来实现的。与之相比，服务端负载均衡可以通过硬件或者软件来实现。硬件指的是在服务器节点间安装专门用于负载均衡的设备如 Array 、F5 等实现负载均衡; 而软件指的是在服务器上安装一些具有均衡负载功能的软件如 LVS、Nginx 等来完成请求分发工作。

除了现有的实现负载均衡的工具外，还有研究对负载均衡算法和工具进行了研究。例如，Niu 等人[62]提出了基于消息队列（只支持使用这种微服务通信方式的系统）的面向微服务调用链的负载均衡算法（COLBA），通过减少网络开销和减少操作复杂度来降低响应时间; Rusek 等人[63]为运行在 OpenVZ 容器中的微服务提供了一个非集中式的负载均衡系统，该系统相比于集中式的容器编排系统能提升一定的性能。

（9）监控和日志

微服务监控的主要目的是在系统运行时对基础设施、服务等的性能进行观察，而日志则用于在发生故障后快速排除错误。目前这一领域主要关注监控指标的确定，监控指标对应数据以及日志信息的收集、存储和分析。

在对监控相关的设计指导上，Haselböck 等人[64]提出了监控设计领域上的决策指导模型，用于生成监控数据、对监控数据进行管理和处理，以及向涉众传播和呈现监控信息。

不同的监控指标的选择需要不同的方法来对相应数据进行收集和分析。例如 Chen 等人[51]选择平均响应时间、单位时间失败次数、CPU 和网络负载等作为监控指标，并通过在程序里设置探针或者以日志收集的方式来实现对这些监控数据的收集，然后通过 logstash 或 kafka 方式将监控数据推送到监控中心进行实时计算，统计每秒查询率、平均/最大响应时间、平均错误数等数据，最后通过阈值决定是否发送报警。Ding 等人[65]将网络、应用响应时间、每秒钟系统能够处理的交易或事务的数量、系统同时处理的请求数或者事务数以及微服务的调用路径作为监控指标，并基于这些指标的历史数据建立逻辑回归模型，确定微服务的调用路径在正常情况下的安全置信区域，在出现异常的情况下及时进行报警。

此外，还有研究对微服务的状态、负载或网络流量进行监控。例如 Liu 等人[66]设计实现了一个微服务监控框架，该框架可对微服务的状态和负载进行监控，

然后基于聚类分析算法对实时和历史数据进行统计分析。Sun 等人[67]提出了安全即服务模式，通过为网络虚拟机管理程序添加新的 API 原语 FlowTap（用于云基础设施的原语，能够支持细粒度的虚拟网络监控），为网络流量构建灵活的监控和策略实施基础架构，从而监控微服务应用程序的安全。

4.2 组织架构和流程

微服务组织架构和流程关注开发和操作微服务团队的组织、以及构建微服务系统所必须的流程（如部署流程、DevOps 流程等）。

（1）开发团队组织

“康威定律”指出，设计系统的团队组织产生的设计等同于组织之内、组织之间的沟通结构[68]。微服务系统的设计结构与传统软件的设计结构并不相同，因此必须以不同于传统软件开发团队组织的方式来组织微服务开发团队。

为此，Carrasco 等人[69]认为应通过微服务分离跨职能团队，开发团队专注于单个或多个微服务的所有开发过程，他们完全对自己的微服务负责，而不是每个团队完成其中的一部分，也即每个团队具有完全开发微服务所需的所有技能以确保团队和微服务的独立性，同时又不能忽视开发团队之间的协作。

（2）部署

为了降低微服务部署和操作（运营）的成本、优化资源配置以及方便服务的重部署和重发布，对部署流程的设计不容忽视。相关研究主要关注采用何种方式进行部署（例如将服务部署在容器或者虚拟机上）、部署相关问题的解决以及提供相应平台和工具。

微服务和 SOA 都将系统划分为服务，但划分的方式并不相同，使得从部署的角度来看 SOA 仍然是一个单体应用，而无法对单个服务进行独立部署[70]。目前，微服务的部署主要有两种方式。第一种是将多个微服务实例部署在单个物理机或虚拟机上，优点是部署速度很快，资源的有效利用率高，缺点是服务实例之间隔离性较差，可能会出现某个实例占用过多内存或 CPU 资源的情况[71]。第二种是将单个微服务实例部署在单个主机上，这个主机既可以是虚拟机也可以是容器（轻量级的虚拟机），使用虚拟机的缺点是资源的有效利用率不高，服务实例的维护效率降低等；而使用容器则具有轻量灵活、服务实例可被自动化部署、降低部署难度并提高效率等优点[30]。

将微服务装入容器进行独立的部署和扩展已成为面向微服务软件开发的一种必然趋势[21,29,33,34,53,72]，目前研究主要使用的容器技术为 Docker[73]。Docker 和其他容器技术可与容器编排技术（如 Kubemetes、Mesos、Docker Swarm）相结合实现容器分配和管理的自动化。

现有研究致力于解决部署相关问题的同时降低成本，例如 Zhou 等人[74]提出了对于容器中微服务的离线和在线调度框架以解决调度问题，该框架模型能够在适应复杂的云计算工作的同时实现计算上和经济上的效率，能够最大限度地降低部署微服务的成本；Guerrero 等人[75]提出了一个非优势排序遗传算法（最常用的多目标优化算法之一，该研究中用于优化多种资源分配的相互组合问题以获得最优解）来解决对微服务所在容器资源的分配问题，实现了对于容器配置的优化和弹性管理，实验结果证明，客观上该方法比 Kubernetes 中实施的容器管理策略具有更好的价值；Wan 等人[76]提出了一个算法来确定容器的放置位置和任务的分配问题，使得在保证应用程序的服务延迟要求的同时，最小化应用程序部署成本和运营成本，提高容器中应用程序部署和维护的可扩展性和弹性。Zheng 等人[77]提出了 SmartVM，一个支持业务 SLA（Service-Level-Agreement，通常用于指定预期的服务质量标准）且以微服务为中心的部署框架，该框架能够简化构建和部署可动态伸缩的微服务的流程以解决大规模微服务的管理问题，评估结果表明其在部署成本、资源利用和 SLA 遵从性上均优于传统的单体和最先进的微服务部署方法。

此外，也有研究者为微服务部署提供了平台和工具的支持。例如 Gabbrielli 等人[78]提出了一个以 Jolie 语言书写的工具 JRO，用于实现微服务的自动和优化部署；Guo 等人[79]提出了一个基于微服务和容器技术的云件部署 PAAS 平台，可直接用于部署微服务；Li 等人[71]提出了应用部署引擎 OpsFlow，通过组合多种自动化部署技术以应对异构应用与部署环境，从而实现高效自动化的微服务应用部署

就使用容器来部署微服务而言，也存在一个容器部署一个微服务和一个容器部署多个微服务两种方式。Shadija 等人[80]对这两种方式进行了性能上的比较，发现在服务延迟上两者并没有明显区别。

（3）DevOps

如今的企业需要以前所未有的速度响应客户需求的变化，这使得许多公司都开始开展 DevOps 运动并实施持续交付[81]。DevOps 是一组简化和集成软件开发过程、部署和维护的技术，旨在缩短更改系统和将更改运用到生产环境之间的时间，同时在代码和交付机制方面保证软件产品的质量[82]。

持续交付是 DevOps 实践的重要组成部分。Chen[81]通过四年的实践发现，DevOps 对于可部署性和可修改性水平的要求很高，单体架构难以满足这一

要求，而微服务架构在这点上十分契合。同时，对于微服务软件而言，每个微服务都是一个可部署单元，需求的快速变化要求微服务能够被快速自动的部署。因此，持续交付和微服务的结合势在必行。

此外，DevOps 工具链在相关实践的自动化上起到很大的促进作用，能够支持在短时间内完成高质量的交付。在微服务软件开发 DevOps 工具链的研究上，Ebert 等人[83]研究了一系列 DevOps 工具，包括有助于实现快速迭代，减少手动耗时任务的构建工具如 Apache Ant、Maven、Rake 以及 Gradle；尽早将开发者工作集成起来的持续集成工具如 Jenkins、TeamCity 以及 Bamboo；实现基础设施管理自动化的配置管理工具如 Puppet、Chef 以及 Ansible；以及保持基础设施稳定性和性能的日志工具（如 Graylog2）和监控工具（如 Nagios 和 New Relic）。

除了上述单个设计领域的具体实现之外，还有微服务框架和服务网格为多个设计领域的实现提供了解决方案，研究者能在其基础上快速构建微服务。具体地，微服务框架是微服务体系结构的具体实现及解决方案，是企业、研究者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[30]，目前研究主要使用 Spring Cloud[34,35,52,53]这一微服务框架。另一方面，服务网格将微服务中实现服务发现、熔断、限流、降级、分布式跟踪等功能的逻辑提取出来，通过统一的控制平面来定义这些策略，并为每一个微服务实例部署一个称为 sidecar 的代理。微服务之间通过 sidecar 代理进行通信，sidecar 负责在服务通信时应用和执行预先定义的治理策略，从而为所有微服务提供完全集成的服务治理环境。

5 面向微服务的软件测试

与单体架构相比，微服务架构拥有众多服务，这些服务一起运行并且更容易发生变化，因此在测试阶段需要考虑服务所使用的语言、框架以及基础设施等；同时，对某个服务的测试可能会涉及其他服务，这些均为微服务软件的测试带来了巨大的挑战。

5.1 测试策略

根据微服务自身特点，Clemson[84]，Savchenko[85]以及 Cohn[86]对如何测试微服务应用程序给出了自己的策略。Clemson 提出了用于微服务测试的五层金字塔模型[84]，包含单元测试、集成测试、组件测试、合同测试和端到端测试这五个层次，其在测试的覆盖范围上依次增大。其中，单元测试关注微服务内部的模块；集成测试验证微服务内部模块之间的交互、以及内部模块与外部组件（包括其他微服务，数据存储和高速缓存）之间的交互；组件测试将单个微服务独立出来进行测试，以验证每个微服务本身能否满足相应需求；合同测试用于验证微服务是否满足服务消费方所期望的合同；而端到端测试则是测试应用程序的整体行为，以最终软件产品的视角来验证应用程序是否满足业务目标。

Savchenko 等人[85]提出的测试策略包括单元测试、集成测试以及系统测试三个阶段，其中单元测试和系统测试分别对应于 Clemson 五层测试中的组件测试和端到端测试，集成测试则是对微服务之间的通信进行测试，包括对于通信协议和格式、死锁解决方案、共享资源使用和消息传递顺序的测试。

Cohn[86]的测试策略包含单元测试、合同测试、集成测试和端到端测试四个层次，其中单元测试、合同测试和端到端测试与 Clemson 五层测试中同名层次对应，集成测试则对应于 Savchenko 等人的测试策略中的集成测试。

5.2 研究进展

我们基于不同的测试领域对收集的 13 篇测试相关文献进行分类，表 4 给出了分类的结果。我们发现收集的论文中没有针对 Clemson 测试策略中单元测试和集成测试的相关研究。

Table 4 Current researches on testing of microservices

表 4 微服务测试的已有研究

研究内容	文献
测试策略	[84-86]
组件测试	[87-91]
合同测试	[92]
端到端测试	[93]
验收测试	[95]
回归测试	[94]
变异测试	[96]

在组件测试上，Wu 等人[87]提出了一个基于非侵入式故障注入的微服务应用容错性能测试框架，使用该框架可定制和执行测试用例，以验证目标服务的容错方式和性能；Heorhiadi 等人[88]提出了一个系统测试微服务故障处理能力的框架 Gremlin，允许操作员通过操纵网络层的服务间消息来执行测试。此外，由于微服务系统中服务个数过多，对微服务进行逐一测试会耗费大量人力物力，因此需要对测试进行自动化。Camargo 等人[89]提出了一个自动化执行性能测试的

方法，方法将包含测试参数的规范附加到每个服务，从而对这些参数进行自动化测试；Rahman 等人[90]研究了如何并行执行内存中的对于微服务的测试，他们使用 Docker 提供的操作系统级虚拟化，在开发人员的容器化沙箱中提供并行测试执行的服务框架，从而实现了微服务的并行测试。此外，还有研究者利用组件测试来帮助确定部署方案，例如 Avritzer 等人[91]对部署完成后的微服务在负载上进行自动性能分析以定量评估微服务部署方案（不同内存分配、cpu 分配、容器副本），从而帮助确定最优部署配置。

在合同测试上，Ma 等人[92]提出了 GMAT (基于图的微服务分析和测试)方法，该方法能够自动生成系统对应的服务依赖图，在开发的早期阶段通过分析风险服务调用链来检测异常，并在开发新版本的目标系统时跟踪服务之间的联系。

在端到端测试上，Meinke 等人[93]使用基于学习的测试（LBT）来进行端到端测试，以这种测试方法来评估微服务系统的功能正确性及其对注入故障的鲁棒性。

此外，还有研究者研究了微服务开发相关回归测试、验收测试和变异测试技术。在回归测试上，Kargar 等人[94]提供了一个自动运行的方法，他们将回归测试放入微服务应用开发的持续交付步骤当中，在应用新版本交付时根据可测量的指标（如资源使用率，系统故障率，系统性能等）将应用与先前版本进行比较，并在需要时阻止发布最新版本。该方法将最后开发版本作为黑盒来测试其可操作性并使用输入流作为测试的输入（通过 Twitter 提供的 Diffy 工具实现），降低生产成本的同时大大降低了运行测试的成本。

在验收测试上，Rahman 等人[95]提出了一个可重用的自动测试架构来检验微服务仓库的可重用性，可审计性和可维护性是否满足预期行为及其验收标准，该架构还将最小化可审计性和系统集成维护问题，以及减少开发人员和测试人员之间的冲突，允许他们在实现相同的应用程序目标的同时独立地在单独的微服务仓库上进行迭代检验。

在变异测试上，Winzinger[96]提出了创建可能的变异算子的初步设想，该算子的应用可以通过变异检测来检验出错误的测试用例从而保证测试用例的质量，进一步提高微服务系统的质量。

目前，与微服务测试相关的研究仍旧较少，现有研究在关注测试策略中的组件测试、合同测试以及端到端测试的同时还对回归测试、验收测试和变异测试相关技术进行了一定程度的探索，且研究者主要关注单个微服务的自动化测试框架，而缺乏对于微服务内部逻辑测试的研究。

6 面向微服务的重构

面向微服务的软件重构旨在将传统上难于维护和扩展的单体应用程序（通常是企业中的遗留系统）重构为微服务应用程序（即面向微服务的重构或者说迁移到微服务），目前与这一活动相关的研究主要集中在重构的流程、方法、决策、挑战、实证和经验研究以及如何提取微服务上。表 5 出了本文收集到的 24 篇相关文献的分类结果。

Table 5 Current researches on refactor of microservices

表 5 微服务重构的已有研究

研究内容	文献
重构流程	[97-99]
重构方法	[100-110]
重构决策	[111]
提取微服务	[101-102] [104-106] [112-115]
实证和经验研究	[69][82] [97] [99] [116-118]

在重构流程上，研究者致力于将重构过程进行分解以更好地进行管理。Francesco 等人[97]认为重构过程与将已有系统迁移到面向服务架构的过程一致，包括逆向工程（分析已有系统并且识别出能够作为服务的候选元素）、架构转换（将已有系统体系结构重构为面向微服务的体系结构）和前向工程（最终确定、实施和部署新系统的设计）这三个步骤；Fan 等人[98]则从软件生命周期的角度考虑迁移流程，并总结了每个阶段的方法和工具。Taibi 等人[99]则给出了一个由三个流程组成的迁移过程框架，其中两个用于从头开始重新开发整个系统，另一个用于在现有系统之上创建具有微服务架构的新功能。

在重构方法上，Fritzsch 等人[100]将已有的重构方法分为四类，包括静态代码分析辅助方法、元数据辅助方法、工作负载-数据辅助方法、以及动态微服务组合方法。静态代码分析辅助方法[101,102]需要应用程序的源代码并通过可能的中间阶段从中派生出分解方案；元数据辅助方法[103,104,105,106]使用更抽象的输入数据，例如 UML 图形式的架构描述用例和接口；工作负载-数据辅助[107]方法通过测量模块或功能级别上应用程序的操作数据（例如通信、性能等）来找到合适的服务削减，并使用此数据来确定合适的分解和粒度；动态微服务组合方法[108,109,110]通过描述微服务运行时环境来更全面地解决分解问题。

在重构决策上，Christoforou 等人[111]给出了与迁移到微服务的决策相关的关键概念和驱动因素，并提出一个决策支持系统以支持微服务重构过程中的决策制定。

在微服务的提取上，研究者致力于提出可行的从单体应用中提取出候选微服务的方法，其中多数基于有向图或者无向图方法，少数基于机器学习方法和其他方法。具体地，

- 在使用无向图方法提取微服务上，研究者往往结合使用图形聚类算法。例如，Gysel 等人[104]提出了 Service Cutter 方法，即从软件工程构件(如域模型和用例)中提取出耦合信息，并表示为无向加权图，以查找和评估联系紧密的聚类，通过图形聚类算法识别出微服务；Mazlami 等人[105]同样使用了聚类算法来聚类无向加权图，他们提出了三种形式化的耦合策略，并将它们嵌入到基于图的聚类算法当中，随后在基于 Web 的原型中实现了一个微服务提取模型，该模型能在重构场景中通过上述算法对无向加权图（依据整体代码库的信息来构造）进行处理从而推荐微服务候选项。
- 在使用有向图方法提取微服务上，研究者使用有向图来构建依赖关系图、数据流图和功能调用图。例如，Levcovitz 等人[101]构建了业务功能和数据库表的依赖关系图，他们首先根据应用中 facade（调用业务功能的入口）、业务功能或数据库表创建一个依赖关系图（其中顶点表示 facade、业务功能或数据库表，边表示来自 facade 对业务功能、业务功能之间的调用以及业务功能对数据库的访问），最后根据获取的业务功能和数据库表组成的依赖关系识别出候选微服务；Chen 等人[112]构建了数据流图，他们从业务逻辑中构建详细的数据流图，然后将具有相同输出数据类型的相同操作组合在一起从而将数据流图进行压缩，最后根据“操作+输出数据”的描述风格提取出候选微服务；Ren 等人[113]构建了功能调用图，他们结合静态和动态分析来获得单体应用程序的功能调用图，利用功能之间的耦合程度以及功能聚类识别出微服务；
- 在基于机器学习方法提微服务上，Abdullah 等人[114]为了提高应用程序的可扩展性和性能而设计了一种基于黑箱的方法，该方法使用应用程序访问日志和非监督机器学习方法自动将应用程序分解为微服务，文章中还提出了一种动态选择合适资源类型的方法，以提高基于微服务的应用程序的整体性能。
- 在基于其他方法提取微服务上，Kecskemeti 等人[115]基于 ENTICE 项目的结果（即其图像合成和优化工具）提出将单体服务解耦成微服务的方法，并提供了有关这些结果如何帮助振兴过去单体服务的见解，以及应用哪些技术来帮助未来的微服务开发人员；Escobar 等人[102]根据企业 Java Bean（EJB）操作的数据进行分离和分组，根据算法评估各个 EJB 之间的耦合性，然后将一个或若干个 EJB 映射为一个微服务；Baresi 等人[106]提出了一种基于 OpenAPI 规范描述的预见/可用功能的语义相似性的解决方案，该方法通过利用参考词汇表识别潜在的候选微服务，作为细粒度的内聚操作组（和相关联的资源）。

在重构相关的实证和经验研究上，Francesco 等人[97]进行了一项关于工业中采用微服务的迁移实践的实证研究，他们调查了来自于 16 家不同的 IT 公司的 18 名从业者，并设计调查问卷询问被研究者在迁移过程中进行了哪些活动以及面临哪些挑战，最终报告了被研究者在逆向工程、架构转换和前向工程阶段进行的各项活动（如对已有系统信息的研究、设计新架构执行的任务和如何开始迁移等）和遇到的各种挑战（如原系统的高度耦合、服务界限的识别和微服务工作所需的基础设施的设置）；Taibi 等人[99]则调查了 21 名采用微服务架构的从业者并分析了他们迁移到微服务的动机和过程中遇到的困难，他们发现可扩展性和可维护性是推动面向微服务重构发展的主要因素，而遇到的主要困难则在于难以将单体系统解耦成微服务、将遗留数据库中的数据进行迁移以及分割给各个服务以及实现微服务之间的通信；而 Carrasco 等人[69]则通过对 34 篇学术论文以及 24 篇灰色文献进行调研，并总结了在微服务重构过程中常见的 9 个陷阱。

此外，Balalaie 等人[116]报告了他们将单体软件架构迁移到微服务架构的经验，其发现尽管微服务架构为可伸缩性和可用性提供了高度灵活性，但也引入了新的复杂性和困难。他们同时还研究了一些工业规模的软件迁移项目，从中识别和收集了一组迁移和重构的设计模式[117]，并探讨了在一个现实案例中如何实现 DevOps 的平稳迁移[82]；Furda 等人[118]则讨论了将单体应用程序迁移到微服务中的三个挑战（多租户、有状态和一致性）；

目前，面向微服务的重构相关的研究数量在微服务软件开发中仅次于设计与实现。研究者不仅关注对于重构整体流程和方法的探索，提出了很多将单体应用迁移为微服务的方法，同时也还在实证和经验研究中学习和分享实践经验。

7 未来研究方向

面向微服务软件开发作为一个新兴的研究领域，早已在注重实践的工业界得到了广泛的运用，并派生出一系列的开源工具和平台。但通过本文调研，我们

发现目前学术界对于微服务开发的研究尚处于早期阶段，与工业实践之间还存在很大缺口，尤其在微服务需求分析、设计与实现、测试和重构等方向上都需要进一步的深入研究。

在微服务软件需求分析方面，目前微服务需求分析仍主要沿用传统的软件工程需求分析方法[17,18,19,21]，很少有专门针对微服务的需求分析研究。一方面我们需要针对微服务的特点来研究新的功能需求分析方法并开发相应的需求分析工具，同时也要综合考虑微服务开发和测试等若干后续阶段，从而形成一整套与微服务软件开发相契合的方法体系和工具系统。另一方面，不同微服务软件在非功能需求存在很多共同之处[17,21,22]，而现有研究在这一方面仍较为割裂，尤其缺乏相应的实践标准（如考虑哪些以及如何考虑非功能需求）来为研究者提供规范的参考。

在微服务软件设计与实现方面，我们一方面需要全面了解已有的工业界实践方式，并在此基础上分析当前仍存在的问题和挑战。其中工业界的相关研究和实践留下了众多灰色文献[13]，虽然其中内容可能参差不齐，但仍旧存在着巨大的参考和学习价值。另一方面，我们还需要研究微服务软件设计的整体质量评估方法。微服务软件设计过程中需要考虑的方面非常多，同时对应的实现也存在众多选择[29,37,47]，我们因此需要有效的质量评估方法来对设计质量进行分析，从而帮助设计人员对于系统整体有一个清晰的认识，同时也能更好地协助解决设计与实现上的决策问题。

在微服务测试方面，庞大的服务数量使得微服务软件的测试工作量十分巨大，在这种情况下，纯粹依靠手工来进行测试并不现实。因此，一方面我们需要研究更加准确、快速、全面的微服务测试方法，并提出新的测试覆盖标准；另一方面我们也需要在面向微服务的测试建模、测试用例生成、测试预期输出判定、以及故障定位等多个阶段开发相应的自动化测试工具，形成全面系统的微服务测试方法体系。

在微服务重构方面，如何从单体应用中分解出微服务这一问题已受到广泛关注且产生了很多研究成果[101,104,106,114]，然而这些研究成果是否有效仍需要进进一步的验证，同时也需要更多的经验研究去对不同的重构方法进行比较和评估。

8 总结

面向微服务软件开发是近年来软件工程领域研究的热点话题。为了了解该领域当前的研究进展，本文首先收集了目前与微服务软件开发相关的 89 篇研究论文，随后从需求分析、设计与实现、测试、以及重构的角度对当前已有的方法、工具和实践进行了分析和总结，并对微服务软件开发可能的一些研究方向进行了讨论。

经过不足十年的发展，微服务在工业实践中已经取得了广泛且丰富的应用，但学术界对于微服务开发的研究尚处于早期阶段，与实践应用之间仍存在着很大鸿沟。作为面向服务体系结构的最新研究和发展趋势，微服务在未来具有良好的研究和应用前景，本文因而能为相关研究者和开发团队提供参考和借鉴，以进一步提高微服务软件开发的效率和质量。

参 考 文 献

[1] Xiao Z, Wijegunaratne I, Qiang X. Reflections on SOA and Microservices[C] //2016 4th International Conference on Enterprise Systems (ES). IEEE, 2016: 60-67

[2] Ramollari E, Dranidis D, Simons A J H. A survey of service oriented development methodologies[C] //The 2nd European Young Researchers Workshop on Service Oriented Computing. 2007, 75

[3] Tvedt R T, Costa P, Lindvall M. Evaluating Software Architectures[J]. Advances in Computers, 2004, 61(5): 1-43

[4] Sharma A, Kumar M, Agarwal S. A complete survey on software architectural styles and patterns[J]. Procedia Computer Science, 2015, 70: 16-28

[5] Dragoni N, Giallorenzo S, Lafuente A L, et al. Microservices: yesterday, today, and tomorrow[M] //Present and ulterior software engineering. Springer, Cham, 2017: 195-216

[6] Namiot D, Sneps-Sneppe M. On micro-services architecture[J]. International Journal of Open Information Technologies, 2014, 2(9): 24-27

[7] Ling Xiaodong. A review of SOA[J]. Computer Applications and software, 2007, 24(10): 122-124 (in Chinese)
(凌晓东. SOA 综述[J]. 计算机应用与软件, 2007, 24(10): 122-124)

[8] Papazoglou M P , Traverso P, Dustdar S , et al. Service-Oriented Computing: State of the Art and Research Challenges[J]. Computer, 2007, 40(11): 38-45

[9] Aziz M W, Rashid M. Extended meta-model for service-oriented development of embedded real-time systems[C] //2017 First International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT). IEEE, 2017: 1-7

[10] Jamshidi P , Pahl C , Nabor C. Mendonça, et al. Microservices: The Journey So Far and Challenges Ahead[J]. IEEE Software, 2018, 35(3): 24-35

[11] Rothenhaus K J , Michael J B , Shing M T. Architectural Patterns and Auto-Fusion Process for Automated Multisensor Fusion in SOA System-of-Systems[J]. IEEE Systems Journal, 2009, 3(3): 304-316

[12] Butzin B, Golatowski F, Timmermann D. Microservices approach for the internet of things[C] //2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2016: 1-6.

[13] Soldani J, Tamburri D A, Van Den Heuvel W J. The pains and gains of

microservices: A Systematic grey literature review[J]. Journal of Systems and Software, 2018, 146: 215-232

[14] James Lewis, Martin Fowler. Microservices: A Definition of this New ArchitecturalTerm[OL]. [2019-06-27]. https://www.martinfowler.com/articles/microservices.html

[15] Sotelo K I G, Baron C, Esteban P, et al. How to find non-functional requirements in system developments[J]. IFAC-PapersOnLine, 2018, 51(11): 1573-1578

[16] SEBoK contributors. Guide to the Systems Engineering Body of Knowledge (SEBoK) [OL]. [2019-06-27]. http://sebokwiki.org Sommerville

[17] Richter D, Konrad M, Utecht K, et al. Highly-available applications on unreliable infrastructure: Microservice architectures in practice[C] //2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2017: 130-137

[18] Hassan S, Bahsoon R. Microservices and their design trade-offs: A self-adaptive roadmap[C] //2016 IEEE International Conference on Services Computing (SCC). IEEE, 2016: 813-818

[19] Suryotrisongko H , Jayanto D P , Tjahyanto A . Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot[J]. Procedia Computer Science, 2017, 124: 736-743

[20] Porrmann T, Essmann R, Colombo A W. Development of an event-oriented, cloud-based SCADA system using a microservice architecture under the RAMI4. 0 specification: Lessons learned[C] //IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society. IEEE, 2017: 3441-3448

[21] Wizenty P N, Rademacher F, Sorgalla J, et al. Design and Implementation of a Remote Care Application Based on Microservice Architecture[C] //Federation of International Conferences on Software Technologies: Applications and Foundations. Springer, Cham, 2018: 549-557

[22] Schneider T, Wolfsmantel A. Achieving Cloud Scalability with Microservices and DevOps in the Connected Car Domain[C] //Software Engineering (Workshops). 2016: 138-141

[23] Taibi D, Lenarduzzi V, Pahl C. Architectural Patterns for Microservices: A Systematic Mapping Study[C] //CLOSER. 2018: 221-232

[24] Bogner J, Zimmermann A, Wagner S. Analyzing the Relevance of SOA Patterns for Microservice-Based Systems[J]. ZEUS, 2018, 9: 9-16

[25] Erl T. SOA Design Patterns (paperback)[M]. Pearson Education, 2008

[26] Erl T, Carlyle B, Pautasso C, et al. Soa with rest: Principles, patterns &constraints for building enterprise solutions with rest[M]. Prentice Hall Press, 2012

[27] Rotem-Gal-Oz A, Bruno E, Dahan U. SOA patterns[M]. Shelter Island: Manning, 2012

[28] Haselböck S, Weinreich R, Buchgeher G. An Expert Interview Study on Areas of Microservice Design[C] //2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA). IEEE, 2018: 137-144

[29] Yan Liyun, Yang Xinzhang,He Zhenwei, et al. Micro-service solution of operator's business platform[J]. Telecommunications Science, 2018, 34(11): 172-180 (in Chinese)
(严丽云, 杨新章, 何震苇, 等. 运营商业务平台微服务化方案[J]. 电信科学, 2018, 34(11): 172-180)

[30] Xin Yuanyuan ,Niu Jun , Xie Zhijun , et al. Survey of implementation framework for microservices architecture[J]. Computer Engineering and Applications, 2018, 54(19): 16-23 (in Chinese)
(辛园园, 钮俊, 谢志军, 等. 微服务体系结构实现框架综述[J]. 计算机工程与应用, 2018, 54(19): 16-23)

[31] Le V D, Neff M M, Stewart R V, et al. Microservice-based architecture for the NRDC[C] //2015 IEEE 13th International Conference on Industrial Informatics (INDIN). IEEE, 2015: 1659-1664

[32] Idoughi D, Abdelouhab K A, Kolski C. Towards a microservices development approach for the crisis management field in developing countries[C] //2017 4th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM). IEEE, 2017: 1-6

[33] Jiang Yong. Design of infrastructure based on microservices architecture[J]. Computer Engineering and Software, 2016, 37(5): 93-97 (in Chinese)
(蒋勇. 基于微服务架构的基础设施设计[J]. 软件, 2016, 37(5): 93-97)

[34] Lv Yacong, Liu Zichen, Zhang Yucheng. Design of Agricultural Machinery Monitoring System based on Micro-Service Architecture[J]. Communications Technology , 2019, 52(02): 504-511 (in Chinese)
(吕亚聪, 刘子辰, 张玉成. 基于微服务架构的农机车辆监控系统设计[J]. 通信技术, 2019, 52(02): 504-511)

[35] Liu Gang. Dealer management system based on micro service architecture[J]. Journal of Computer Applications, 2018, 38(S2): 243-249 (in Chinese)
(刘罡. 基于微服务架构的汽车经销商管理系统[J]. 计算机应用, 2018, 38(S2): 243-249)

[36] Mao Lin. Research on Jiangsu Agricultural Administrative Law Enforcement System Based on Microservice[J]. Journal of Anhui Agricultural Sciences. v.46; No.606(29): 183-186 (in Chinese)
(毛林. 面向微服务的江苏农业行政执法系统研究[J]. 安徽农业科学, v.46; No.606(29): 183-186)

[37] Haselböck S, Weinreich R, Buchgeher G, et al. Microservice Design Space Analysis and Decision Documentation: A Case Study on API Management[C] //2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA). IEEE, 2018: 1-8

[38] Viennot N, Lécuyer M, Bell J, et al. Synapse: a microservices architecture for heterogeneous-database web applications[C] //Proceedings of the Tenth European Conference on Computer Systems. ACM, 2015: 21

[39] Messina A, Rizzo R, Storniolo P, et al. A simplified database pattern for the microservice architecture[C] //The Eighth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA). 2016

[40] Richards M. Microservices vs. service-oriented architecture[M]. O'Reilly Media, 2015

[41] Evans E. Domain-driven design: tackling complexity in the heart of software[M]. Addison-Wesley Professional, 2004

[42] Diepenbrock A, Rademacher F, Sachweh S. An ontology-based approach for domain-driven design of Microservice architectures[J]. INFORMATIK 2017, 2017

[43] Rademacher F, Sachweh S, Zündorf A. Towards a UML profile for domain-driven design of microservice architectures[C] //International Conference on Software Engineering and Formal Methods. Springer, Cham, 2017: 230-245.

[44] Steinegger R H, Giessler P, Hippchen B, et al. Overview of a Domain-Driven Design Approach to Build Microservice-Based Applications[C] //The Thrid Int. Conf. on Advances and Trends in Software Engineering. 2017

[45] Hippchen B, Giessler P, Steinegger R, et al. Designing microservice-based applications by using a domain-driven design approach[J]. International Journal on Advances in Software, 2017: 432-445

[46] Rademacher F, Sorgalla J, Sachweh S. Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective[J]. IEEE Software, 2018, 35(3): 36-43

[47] Haselböck S, Weinreich R, Buchgeher G. Decision guidance models for microservices: service discovery and fault tolerance[C] //Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems. ACM, 2017: 4

[48] Stubbs J, Moreira W, Dooley R. Distributed systems of microservices using docker and serfnode[C] //2015 7th International Workshop on Science Gateways. IEEE, 2015: 34-39

[49] França M, Werner C. Perspectives for Selecting Cloud Microservices[C] //2018 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, 2018: 56-59

[50] Tang W, Wang L, Xue G. Design of High Availability Service Discovery for Microservices Architecture[C] //Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences. ACM, 2019: 253-257

[51] Cheng Lin, Wang Haining, Gao Cuncheng. Application Research of Microservice in Power Trading System [J]. Power System Technology, 2018 (in chinese)
(承林, 王海宁, 高春成. 微服务在电力交易系统中的应用研究[J]. 电网技术, 2018)

[52] Zhou Yongsheng, Hou Fengyu , Sun Wen , et al. Invoicing Management System Based on SpringCloud Microservice Architecture[J]. Industrial Control Computer, 2018, 31(11): 133-134+137 (in Chinese)
(周永圣, 侯峰裕, 孙雯, 等. 基于 SpringCloud 微服务架构的进销存管理系统的设计与实现[J]. 工业控制计算机, 2018, 31(11): 133-134+137)

[53] Long Xinzheng, Peng Yiming, Li Ruomiao. Information service platform based on microservice framework[J]. Journal of Southeast University(Natural Science Edition) , 2017(47): 52 (in Chinese)
(龙新征, 彭一明, 李若淼. 基于微服务框架的信息服务平台[J]. 东南大学学报：自然科学版, 2017(47):52)

[54] Baraiya V, Singh V. Netflix Conductor: A microservices orchestrator [OL]. [2019-06-27]. https://medium.com/netflix-techblog/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40

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

[56] Yahia E B H, Réveillère L, Bromberg Y D, et al. Medley: An event-driven lightweight platform for service composition[C] //International Conference on Web Engineering. Springer, Cham, 2016: 3-20

[57] Zhang H, Yang N, Xu Z, et al. Microservice Based Video Cloud Platform with Performance-Aware Service Path Selection[C] //2018 IEEE International Conference on Web Services (ICWS). IEEE, 2018: 306-309

[58] Krylovskiy A, Jahn M, Patti E. Designing a smart city internet of things platform with microservice architecture[C] //2015 3rd International Conference on Future Internet of Things and Cloud. IEEE, 2015: 25-30

[59] Oauth2[OL]. [2019-06-27]. https://oauth.net/2/

[60] Fu G, Sun J, Zhao J. An optimized control access mechanism based on micro-service architecture[C] //2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2). IEEE, 2018: 1-5

[61] Yarygina T, Bagge A H. Overcoming security challenges in microservice architectures[C] //2018 IEEE Symposium on Service-Oriented System Engineering (SOSE). IEEE, 2018: 11-20

[62] Niu Y, Liu F, Li Z. Load balancing across microservices[C] //IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018: 198-206

[63] Rusek M, Dwornicki G, Orłowski A. A decentralized system for load balancing of containerized microservices in the cloud[C] //International Conference on Systems Science. Springer, Cham, 2016: 142-152.

[64] Haselböck S, Weinreich R. Decision guidance models for microservice monitoring[C] //2017 IEEE International Conference on Software Architecture Workshops (ICSAW). IEEE, 2017: 54-61

[65] Ding Xueying, Liu Di, Qiu zhen. Design and Implementation of Application Monitoring System Based on Micro Service Architecture[J]. Electric Power Information and Communication Technology, 2018, 16(07): 75-79 (in Chinese)
(丁学英, 刘迪, 邱镇. 基于微服务架构的应用监控系统设计与实现[J]. 电力信息与通信技术, 2018, 16(07): 75-79)

[66] Liu Yitian, Liu Shijin, Guo Wei, et al. Flexible microservice monitoring framework. Computer Systems & Applications, 2017, 26(10): 139-143 (in Chinese)
(刘一田, 刘士进, 郭伟, 等. 柔性微服务监控框架[J]. 计算机系统应用, 2017, 26(10): 139-143)

[67] Sun Y, Nanda S, Jaeger T. Security-as-a-service for microservices-based cloud applications[C] //2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2015: 50-57

[68] Conway M E. How do committees invent[J]. Datamation, 1968, 14(4): 28-31

[69] Carrasco A, Bladel B, Demeyer S. Migrating towards microservices: migration and architecture smells[C] //Proceedings of the 2nd International Workshop on Refactoring. ACM, 2018: 1-6

[70] Wolff E. Microservices: flexible software architecture[M]. Addison-Wesley Professional, 2016

[71] Li chao, Hua Lei, Song Yunkui. OpsFlow: Automatic Application Deployment Engine for DevOps[J]. Computer & Digital Engineering, 2019, 47(01): 190-194+247
(李超, 花磊, 宋云奎. OpsFlow: 一种面向 DevOps 的应用自动化部署引擎[J]. 计算机与数字工程, 2019, 47(01): 190-194+247)

[72] Cui Wei, Li Chunyang, Liu Di, et al. Microservices Oriented Unified Application Development Platform[J]. Electric Power Information and Communication Technology, 2016(9): 12-17 (in Chinese)
(崔蔚, 李春阳, 刘迪, 等. 面向微服务的统一应用开发平台[J]. 电力信息与通信技术, 2016(9): 12-17)

[73] Docker[OL]. [2019-06-27]. https://www.docker.com/

[74] Zhou R , Li Z , Wu C . Scheduling Frameworks for Cloud Container Services[J]. IEEE/ACM Transactions on Networking, 2018, 26(1): 436-450

[75] Guerrero C, Lera I, Juiz C. Genetic algorithm for multi-objective optimization of container allocation in cloud architecture[J]. Journal of Grid Computing, 2018, 16(1): 113-135

[76] Wan X, Guan X, Wang T, et al. Application deployment using Microservice and Docker containers: Framework and optimization[J]. Journal of Network and Computer Applications, 2018, 119: 97-109

[77] Zheng T, Zheng X, Zhang Y, et al. SmartVM: a SLA-aware microservice deployment framework[J]. World Wide Web, 2019, 22(1): 275-293

[78] Gabbrielli M, Giallorenzo S, Guidi C, et al. Self-reconfiguring microservices[M] //Theory and Practice of Formal Methods. Springer, Cham, 2016: 194-210

[79] Guo D, Wang W, Zeng G, et al. Microservices architecture based cloudware deployment platform for service computing[C] //2016 IEEE Symposium on Service-Oriented System Engineering (SOSE). IEEE, 2016: 358-363

[80] Shadija D, Rezai M, Hill R. Microservices: granularity vs. performance[C] //Companion Proceedings of the10th International Conference on Utility and Cloud Computing. ACM, 2017: 215-220

[81] Chen L. Microservices: architecting for continuous delivery and DevOps[C] //2018 IEEE International Conference on Software Architecture (ICSA). IEEE, 2018: 39-397

[82] Balalaie A , Heydarnoori A , Jamshidi P . Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture[J]. IEEE Software, 2016, 33(3): 42-52

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

[84] Toby Clemson. Testing Strategies in a Microservice Architecture[OL]. [2019-06-27]. https://martinfowler.com/articles/microservice-testing/#anatomy-modules

[85] Savchenko D I, Radchenko G I, Taipale O. Microservices validation: Mjolnirr platform case study[C] //2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 2015: 235-240

[86] Arvind Sundar,Technical Test Lead. An Insight into Microservices Testing Strategies[R]. infosys.com, 2018

[87] Wu N, Zuo D, Zhang Z. An extensible fault tolerance testing framework for microservice-based cloud applications[C] //Proceedings of the 4th International Conference on Communication and Information Processing. ACM, 2018: 38-42

[88] Heorhiadi V, Rajagopalan S, Jamjoom H, et al. Gremlin: Systematic resilience testing of microservices[C] //2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2016: 57-66

[89] Camargo A, Salvadori I, Mello R S, et al. An architecture to automate performance tests on microservices[C] //Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services. ACM, 2016: 422-429

[90] Rahman M, Chen Z, Gao J. A service framework for parallel test execution on a developer's local development workstation[C] //2015 IEEE Symposium on Service-Oriented System Engineering. IEEE, 2015: 153-160

[91] Avritzer A, Ferme V, Janes A, et al. A Quantitative Approach for the Assessment of Microservice Architecture Deployment Alternatives by Automated Performance Testing[C] //European Conference on Software Architecture. Springer, Cham, 2018: 159-174

[92] Ma S P, Fan C Y, Chuang Y, et al. Using Service Dependency Graph to Analyze and Test Microservices[C] //2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2018, 2: 81-86

[93] Meinke K , Nycander P . Learning-Based Testing of Distributed Microservice Architectures: Correctness and Fault Injection[M] // Software Engineering and Formal Methods. Springer Berlin Heidelberg, 2015

[94] Kargar M J, Hanifizade A. Automation of regression test in microservice architecture[C] //2018 4th International Conference on Web Research (ICWR). IEEE, 2018: 133-137

[95] Rahman M , Gao J . [IEEE 2015 IEEE Symposium on Service-Oriented System Engineering (SOSE) - San Francisco Bay, CA, USA (2015.3.30-2015.4.3)] 2015 IEEE Symposium on Service-Oriented System Engineering - A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development[J]. 2015: 321-325

[96] Winzinger S. Mutation Testing for Microservices[C] //ZEUS. 2018: 17-19

[97] Di Francesco P, Lago P, Malavolta I. Migrating towards microservice architectures: an industrial survey[C] //2018 IEEE International Conference on Software Architecture (ICSA). IEEE, 2018: 29-2909

[98] Fan C Y, Ma S P. Migrating monolithic mobile application to microservice

architecture: An experiment report[C] //2017 IEEE International Conference on AI & Mobile Services (AIMS). IEEE, 2017: 109-112

[99] Taibi D , Lenarduzzi V , Pahl C . Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation[J]. IEEE Cloud Computing, 2017, 4(5): 22-32

[100] Fritzsch J, Bogner J, Zimmermann A, et al. From Monolith to Microservices: A Classification of Refactoring Approaches[C] //International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. Springer, Cham, 2018: 128-141

[101] Levcovitz A, Terra R, Valente M T. Towards a technique for extracting microservices from monolithic enterprise systems[J]. arXiv preprint arXiv:1605.03175, 2016

[102] Escobar D, Cárdenas D, Amarillo R, et al. Towards the understanding and evolution of monolithic applications as microservices[C] //2016 XLII Latin American Computing Conference (CLEI). IEEE, 2016: 1-11

[103] Ahmadvand M, Ibrahim A. Requirements reconciliation for scalable and secure microservice (de) composition[C] //2016 IEEE 24th International Requirements Engineering Conference Workshops (REW). IEEE, 2016: 68-73

[104] Gysel M, Kölbener L, Giersche W, et al. Service cutter: A systematic approach to service decomposition[C] //European Conference on Service-Oriented and Cloud Computing. Springer, Cham, 2016: 185-200

[105] Mazlami G, Cito J, Leitner P. Extraction of microservices from monolithic software architectures[C] //2017 IEEE International Conference on Web Services (ICWS). IEEE, 2017: 524-531

[106] Baresi L, Garriga M, De Renzis A. Microservices identification through interface analysis[C] //European Conference on Service-Oriented and Cloud Computing. Springer, Cham, 2017: 19-33

[107] Mustafa, O., Gómez, J.M. Optimizing economics of microservices by planning for granularity level Experience Report[R], 2017

[108] Hassan S, Ali N, Bahsoon R. Microservice ambients: An architectural meta-modelling approach for microservice granularity[C] //2017 IEEE International Conference on Software Architecture (ICSA). IEEE, 2017: 1-10

[109] Klock S, Van Der Werf J M E M, Guelen J P, et al. Workload-based clustering of coherent feature sets in microservice architectures[C] //2017 IEEE International Conference on Software Architecture (ICSA). IEEE, 2017: 11-20

[110] Procaccianti, G. et al.: Towards a MicroServices Architecture for Clouds[R]. VU University Amsterdam , 2016

[111] Christoforou A, Garriga M, Andreou A S, et al. Supporting the decision of migrating to microservices through multi-layer fuzzy cognitive maps[C] //International Conference on Service-Oriented Computing. Springer, Cham, 2017: 471-480

[112] Chen R, Li S, Li Z. From monolith to microservices: a dataflow-driven approach[C] //2017 24th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2017: 466-475.

[113] Ren Z, Wang W, Wu G, et al. Migrating Web Applications from Monolithic Structure to Microservices Architecture[C] //Proceedings of the Tenth Asia-Pacific Symposium on Internetware. ACM, 2018: 7

[114] Abdullah M, Iqbal W, Erradi A. Unsupervised learning approach for web application auto-decomposition into microservices[J]. Journal of Systems and Software, 2019, 151: 243-257

[115] Kecskemeti G, Marosi A C, Kertesz A. The ENTICE approach to decompose monolithic services into microservices[C] //2016 International Conference on High Performance Computing & Simulation (HPCS). IEEE, 2016: 591-596

[116] Balalaie A, Heydarnoori A, Jamshidi P. Migrating to cloud-native architectures using microservices: an experience report[C] //European Conference on Service-Oriented and Cloud Computing. Springer, Cham, 2015: 201-215

[117] Balalaie A, Heydarnoori A, Jamshidi P, et al. Microservices migration patterns[J]. Software: Practice and Experience, 2018, 48(11): 2019-2042

[118] Furda A , Fidge C , Zimmermann O , et al. Migrating Enterprise Legacy Source Code to Microservices: On Multi-Tenancy, Statefulness and Data Consistency[J]. IEEE Software, 2017:1-1