基于运营商视角的服务网格技术评测与集成方案

严丽云[1]，杨新章[1]，何震苇[1]，林园致[1]，侯韶新[2]
（1. 中国电信股份有限公司研究院，广东 广州 510630;
2. 中国移动通信集团南方基地，广东 广州 510640）

摘　要：微服务架构随着企业软件架构的变革而变革，从嵌入式、一体化的微服务架构逐步向非侵入、解耦式的服务网格发展，微服务的各种技术架构也不断涌现，运营商如何选型和集成成为微服务布局的重要环节。根据运营商业务特性，侧重于对服务网格（非侵入式架构）技术的评测和集成范畴。评测中从多个维度对比分析开源、主流的服务网格技术以及云服务提供商的微服务解决方案，得出微服务技术、架构的发展趋势。集成范畴重点介绍了两大领域主流技术-Istio 技术与 Kubernetes 集群的集成实践心得，并创新性地提出了支持复杂场景的异构微服务框架及发展思路。最后结合运营商 5G SBA 业务差异化需求，进行了运营商服务网格技术的重点研究及优化方向的探讨。

关键词：微服务架构；服务网格；容器
中文分类号：TP399
文献标识码：A
doi: 10.11959/j.issn.1000−0801.2020154

Service mesh technology evaluation and integration scheme based on telecom operator perspective

YAN Liyun[1], YANG Xinzhang[1], HE Zhenwei[1], LIN Yuanzhi[1], HOU Shaoxin[2]
1. Research Institute of China Telecom Co., Ltd., Guangzhou 510630, China
2.China Mobile Communication Group Southern Base, Guangzhou 510640, China

Abstract: Microservice architecture has evolved along with the changes in enterprise software architecture, from the embedded and integrated micro-service architecture to the non-intrusive and decoupled service mesh.The various of micro-service architectures have sprung up. Choose which microservice architecture to integrate has become an important part of the layout of microservices for the telecom operators. The evaluation and integration of service mesh (non-intrusive architecture) technologies based on the operator's business characteristics was focused on. In the evaluation, choose the open source, mainstream service mesh technologies and cloud service provider's microservice solutions were compared and analyzed from multiple dimensions, and the development trend of microservice technology and architecture was obtained. The integration category focused on the integration of Istio technology and Kubernetes which were mainstream technologies in their respective fields, and a heterogeneous microservice framework and development ideas that supported complex scenarios was innovatively proposed. Finally, combined with the differentiated needs of operators' 5G SBA services, the key research and optimization directions of the operator's service mesh technology were probed.

Key words: micro-service architecture, service mesh, container

收稿日期：2019−09−30；修回日期：2020−05−10

1　引言

随着在互联网行业中广泛地落地实施，微服务体系架构正逐渐普及兴盛。国内的运营商以及金融行业，按照工业和信息化部以及相关监管部门的要求，开始推进软件体系结构的转型，向微服务、容器化转型。

从业务现状需求来看，运营商亟待改造现有单体应用架构。运营商单体应用之殇在于：版本升级周期长，无法及时响应市场需求；应用升级割接难度大，应用无法永久在线，不得不停机中断服务并进行完备的割接保障；技术选型单一，单体服务采用技术单一，无法灵活选择新技术；接口不标准，使用 ESB（enterprise service bus，企业服务总线）这种重量级的解决方案，难以管理；系统资源利用率低，大量业务虚拟机利用率不到 10%；自动化运维程度低，存在大量的人工运维、低效运维现象。

从业务发展趋势来看，微服务、容器将作为未来运营商基础设施发展方向。3GPP R15 的 5G 网络规范中，在 5G 控制面引入了基于服务化架构（service-based architecture，SBA），每个网络功能（network function）对外提供服务化的接口[1]。运营商需要一个开放、可靠、高效、简化运维和智能的云平台，这契合了云原生应用基于微服务、容器和 DevOps 之上构建应用的核心理念。

2　微服务技术框架

微服务的定义众多，且没有标准实现方式。敏捷开发方法的创始人之一 Martin Fowler 将微服务定义为一种架构模式，提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值[2]。

微服务的分布式服务的复杂性决定了它需要一种架构完成服务发现、服务通信、负载均衡、服务监控、灰度发布等一系列服务治理工作，需要涉及服务动态配置管理、系统监控和路由管理等复杂运维内容。微服务技术框架孕育而生，解决分布式服务治理和复杂运维管理问题。

随着微服务技术的发展，业界已出现较多种微服务架构，主流的技术架构有 Spring Cloud、Dubbo、Istio、Linkerd、ServiceComb，本文从架构、成熟度和应用场景 3 个维度对主流技术架构进行分析。

根据架构与服务的耦合度，大致将其分为侵入式架构和非侵入式架构两类，如图 1（a）、图 1（b）所示。其中 Spring Cloud、Dubbo 属于侵入式架构，开发人员通过集成相应的类库 SDK，通过配置以及调用 SDK 接口的方式进行微服务开发。Istio、Linkerd 类属非侵入式架构，将微服务治理功能剥离微服务外，通过数据面网络代理提供，微服务之间通过网络代理实现透明通信。而 ServiceComb 比较特殊，它是同时支持类库 SDK（支持 Java 和 Go 两种语言）和网络代理的混合式微服务方案，网络代理 Mesher 实现了 SDK 基本对等的服务治理能力，通过控制面 Service Center、Apollo 实现服务发现和服务配置。

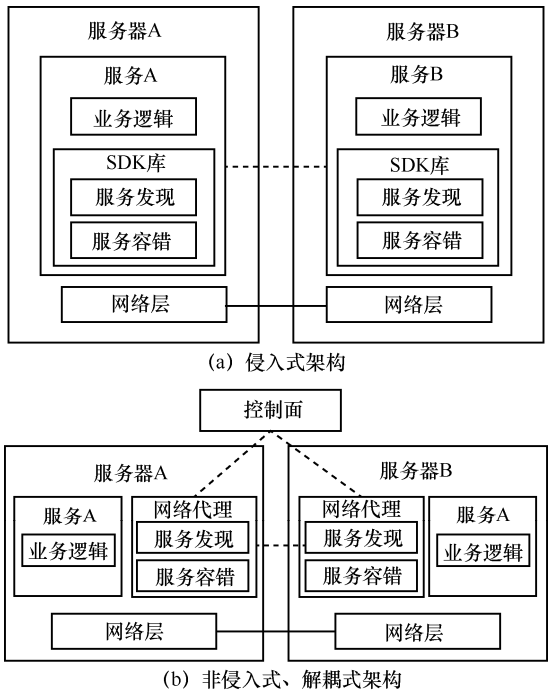


图 1　根据架构与服务的耦合度分类

在应用场景和成熟度方面，侵入式架构的微服务框架起步时间较早，从 SOA 架构起就已存在的技术，成熟度较高，应用场景也较多。Spring Cloud 是业界覆盖量比较高的一个优秀的开源微服务框架，其特点是基于 Spring Boot 微容器能力，融合了 Netflix、HashiCorp 等组织所开源的组件，形成一个配置驱动、能力相对完备、高度可扩展的全家桶式服务框架[3]，适合 Java 语言开发的微服务应用。Dubbo 是阿里开源的微服务框架，广泛应用于阿里巴巴系统，适用于应用通过 RPC 协议实现服务通信，可以对接 Spring Boot，也是与 Java 语言强相关的一种微服务架构。非侵入式架构应用场景在于多语言多技术栈、应用改造流程复杂的服务架构，这一类应用不适合采用 SDK 框架进行改造，目前在公有云服务中应用较多，成熟度较侵入式架构低。

结合运营商自身的特点，即在多厂商联合开发运营的模式下，非侵入式的开发架构更适合运营商介入运维，在微服务治理架构方面倾向于选择非侵入式架构的方式，因此也是本文研究的重点内容[4]。

3 服务网格评测

非侵入式微服务架构又被称为服务网格（service mesh）技术，顾名思义基于数据面服务代理构建的服务拓扑类似一张网，所有的服务流量将被服务代理截获，根据策略进行重定向，从而实现服务负载均衡、服务路由、熔断、灰度发布、错误注入等高级服务治理的功能。服务网格的定位是微服务通信基础设施层，它负责为构建复杂的云原生应用传递可靠的网络请求，实现服务治理能力，并且屏蔽底层基础设施和上层业务实现[5]。服务网格目前业界主流的架构有 Linkerd1.0、Linkerd2.0、Istio、ServiceComb 这 4 类，以下将对主流架构以及解决方案进行综合评测分析。

3.1 服务网格架构评测

Linkerd 是由 Buoyant 公司开发并由 CNCF 管理的开源项目，经历了两个大版本变迁，架构上做了很多大的调整。Linkerd1.0 基于 Scala 编写，旨在基于每个主机进行部署，但由于其相对较大的内存占用，随后促成了 Linkerd2.0 的开发。Linkerd2.0 将控制面和数据面拆分，整合了 Conduit 作为其数据面，相较于 Linkerd1.0 内存占用有很大的改善。应用方面，经过 Twitter、Pinterest、Tumblr 和 PagerDuty 等高流量公司生产验证。

Istio 是 Google、IBM 和 Lyft 联合开源的微服务服务网格框架，可以说 Istio 是如今最炙手可热的服务网格，主要特点是与 Kubernetes 集成度高，但性能却成为 Istio 短板。架构方面，控制面由 3 个组件组成，分别是 Pilot、Mixer、Citadel。数据面对接 Envoy，也可以对接其他网络代理，如 Linkerd、NginMesh。应用方面，公有云服务商中广泛支持“Kubernetes+Istio”的组合。

ServiceComb 是由华为贡献给 Apache 基金会的开源项目。架构方面，ServiceComb 是一种 RPC 的框架，同时支持侵入式的 Java/Go 语言微服务的治理和非侵入式、多语言微服务治理，分别通过 Java-chassis、Go-chassis 和 Mesher 进行支持。应用方面，ServiceComb 是华为云微服务引擎（cloud service engine，CSE）的核心，是华为微服务的事实标准。ServiceComb 也已在数十家国内企业中使用，采用高性能的异步调用的内核，运营商将其用于 5GC 网元开发微服务治理基座。

本文就运营商微服务实践中，微服务框架实际技术选型和集成中比较关注的方面进行综合评测。架构与设计方面，主要考虑架构的合理性、扩展性，开发语言，部署模式；扩展性和易用性，主要考虑与现有基于 Kubernetes 技术的容器管理平台集成度，可视化和自动化程度；功能和性能方面，主要考虑可视化运维能力、服务治理功能全面性、安全性和性能。服务网格架构评测见表 1。

表 1　服务网格架构评测

对比项		Linkerd1.0	Linkerd2.x（Conduit）	Istio	ServiceComb
架构及设计	架构	一体化	控制面和数据面分离	控制面和数据面分离	控制面和数据面分离
	数据面组件	Linkerd	Linkerd-proxy\Envoy	Envoy \Linkerd \Nginmesh	Mesher
	控制面组件	—	Web 组件、控制组件、度量组件	流量治理组件、安全认证组件、度量组件	服务发现、分布式事务处理、配置中心
	开发语言	Scala	GO	GO	—
	数据面部署模式	主机、DaemonSet	Sidecar、DaemonSet	Sidecar	Sidecar、Edgeservice
扩展性和易用性	平台支持	Kubernetes、AWS ECS、DC/OS 和 Docker	Kubernetes	Kubernetes、Consul、Eureka（旧版本）、Cloud Foundry	Kubernetes、DockerDocker、虚拟机、裸机
	协议支持	HTTP1.1 、 HTTP2 、gRPC，不支持 TCP	HTTP1.1 、 HTTP2 、gRPC、TCP	HTTP1.1、HTTP2、gRPC、TCP	gRPC、HTTP
	外挂注入	手动	手动、自动	手动、自动	手动
	控制台	支持，Web 控制台	支持，Web 控制台	不支持	支持，Web 控制台
功能和性能	服务治理	较不全面	较不全面	全面	较不全面
	监控	集成 Prometheus、Zipkin	集成 Prometheus 、Grafana，不支持分布式跟踪	集成 Prometheus、Jaeger、Kiali	Zipkin、Prometheus、Skywalking
	安全性	TSL 支持，不支持主机间的身份验证	TSL 支持，不支持主机间的身份验证	支持 TSL、Citadel 密钥和证书管理、Pilot 分发身份验证	支持 TLS，服务间 RSA 认证
	高可用	试验阶段	试验阶段	有条件下支持(k8 多副本模式、PodAntiAffinity 开启）	支持
	性能	开销多，性能好	开销少、性能好	开销少、性能差	开销多、性能好

（1）架构及设计

架构分为一体化架构和分离架构：分离式架构控制面和数据面可解耦，可独立升级和维护，可适配不同技术实现，更加灵活优越，架构方面 Istio 最为突出，采用分离式架构，并且已支持不同的数据面。部署方式根据数据面所在的位置不同分为 EdgeService、主机、DaemonSet、Sidecar 部署，不同方式数据面组件部署密度有所差别，其中 Sidecar 模式服务代理与服务按 1:1 的比例部署，部署密度最高，EdgeService 模式部署密度最低，一个集群可以选择主备或多实例部署，是 ServiceComb Mesher 独特的部署模式，数据面的部署密度决定了流量控制的粒度和性能，Istio 只支持 Sidecar 的方式部署，可支持最为灵活的流量控制，但却损失了性能，其他几种方案是支持多种部署模式的，可以根据业务需求来选择。

（2）功能和性能

从功能方面，Istio 支持的服务治理功能最全面。安全方面，Istio 表现最为优越，全面支持服务的安全认证和通信的加密。但是在性能方面，由于存在数据面高密度部署和 Mixer 组件的远程调用性能损失，Istio 性能最低，Linkerd1.0 开销最多，Linkerd2.0 性能表现较为优越。

（3）易用性和扩展性

易用性方面，Istio 原生不带图形化界面控制台，管理不够直观，但目前云服务提供商普遍集

成 Istio 并进行了图形化的优化。扩张性方面，现支持主流的 Kubernetes 平台成为服务网格的标配，但 Istio、ServiceComb 还支持多平台。

从评测中可以发现服务网格控制面技术和数据面技术存在许多混合搭配的趋势，分别以各领域领先技术为生态向外围扩展支持，目前控制面和数据面业界最炙手可热的技术分别是 Istio、Evnoy。Istio 可以支持多款数据面技术，包括 Envoy、Go-chassis、NginMesh；Envoy 则支持 Istio、Linkerd 等多款控制面，可以预见服务网格技术将会向通用化和插件化方面发展。数据面方面，CNCF 正在筹建通用数据面 API 工作组，以制定数据面的标准 API，API 以 Proto3 的规范方式定义，并通过定义良好的稳定 API 版本控制策略从现有的 Envoy xDS（xDiscovery Service，一系列服务发现）API 逐步演变，将涵盖服务发现、负载均衡、路由发现、监听器配置、安全发现、负载报告、健康检查等服务治理功能[6]。控制面方面，微软、Linkerd 等联合开展的开源项目 SMI（service mesh API），为实现控制面标准化接口，它定义了一组通用的可移植 API，为开发人员提供跨不同服务网格技术的互操作性，其中包括 Istio、Linkerd 和 Consul Connect。除此之外控制面设计扩展性强，支持许多适配器或插件，兼容更多的平台实现，例如 Istio 的 Pilot xDS API、Mixer Adpter 等。

3.2 解决方案评测

华为云关于微服务方面的产品主要包含应用服务网格（application service mesh，ASM）、微服务引擎（cloud service engine，CSE）、微服务云应用平台（ServiceStage）、CSE Java SDK（ServiceComb 商用版本），其中应用服务网格是基于云容器引擎（cloud container engine，CCE）上集成并定制的 Istio 服务网格。华为云的服务网格产品涵盖较为全面，有专注于运行态的微服务托管也有提供微服务应用从开发到运维全生命周期的管理，有集成方案也有自研方案，并且通过 ServiceStage 支持异构微服务平台。

腾讯云微服务平台 TSF 服务框架层嵌入了 spring cloud 以及 service mesh、服务生命周期管理包括自研的 PaaS 以及治理运维平台。service mesh 部分基于 Istio、Envoy 进行改造，对 Istio 能力进行扩展和增强，对 Consul 进行完整适配，从而可以与 Kubernetes 解耦，除了支持 Kubernetes 外还支持虚拟机和裸金属的服务。

阿里云 Istio on ACK 支持 Kubernetes 上一键部署 Istio 以及 Sidecar 注入，支持图形化界面的服务治理，支持的治理功能和华为云相差无几，另外增加了服务网关的管理，并提供了 Istio Mesh Expansion 整合能力，在 Kubernetes 之中的 Istio 服务网格提供将虚拟机或物理裸机集成进入到服务网格的方法。

Rancher2.3 版本以上已经支持 Istio，Rancher 的自身定位决定了它支持在多云部署的架构下使用服务网格，可以支持 GKE、AKS、EKS 以及本地 Kubernetes 集群的 Istio 使用，服务治理方面目前只支持基于命令行方式操作，另外因为其是开源的架构，定制化支持方面存在天然优势，便于用户构建私有的微服务管理平台。解决方案评测见表 2。

4 集成方案

4.1 Istio 与 Kubernetes 集成

微服务与容器在轻量、快速部署、运维等特征的匹配，使微服务运行在容器中正成为一种标准实践，从业界实践的热度和评测结论，本文选择 Istio 作为微服务治理工具，而 Istio 在集群管理、应用容器部署和容器编排方面非它能力所及，所以在业界微服务平台实践中，Istio 微服务架构通常需要与容器编排系统集成共同支撑微服务应用全生命周期管理。此外，Istio 还支持虚拟机、裸机运行，借助集成可实现多云、混合云下微服务治理能力。

表 2　解决方案评测

对比项	华为云 ASM	阿里云 Istio on　ACK	腾讯云 TSF	Rancher+Istio
部署 Istio	一键部署	一键部署	一键部署	自行准备 helm 环境后一键部署
服务治理 GUI 支持	较全面	较全面	较不全面	不支持，基于 Istioctl 命令行管理
服务治理管理	灰度发布（基于流量、基于请求内容）、负载均衡、连接池管理、熔断、故障注入	灰度发布（基于流量）、负载均衡、连接池管理、熔断、故障注入、服务网关	服务限流、灰度发布	—
安全 GUI 支持	支持 mutualTLS	不支持	不支持	不支持
承载方式	容器集群（Kubernetes）	容器集群（Kubernetes）	容器、虚拟机、裸金属	容器集群（Kubernetes）
监控	精简（集成外部的 APM 框架对应用监控、资源监控、流量监控）	全面（分布式追踪 OpenTracing/Jaeger、遥测数据 Prometheus、故障诊断与检测工具 Weave Scope、性分析服务 Kiali）	服务拓扑、调用链跟踪、日志查询	全面（分布式追踪 OpenTracing/Jaeger、遥测数据 Prometheus、分析服务 Kiali）
收费方式	按照实例数计费（Pod），根据实例规模不同分：基础版/专业版/企业版，其他资源另外收费	免费，集群资源等其他服务另外收费	按照实例数计费（Pod 或云服务器），根据实例规模不同分：基础版/专业版/铂金，其他资源另外收费	免费
应用场景	公有云微服务开发集成	公有云微服务开发集成	混合编排，异构微服务架构	支持多云部署的场景（GKE、EKS、AKS、自定义 k8 集群）

Istio 1.0 版本后其基本架构和 API 逐渐稳定，基于此版本后进行开发集成已经具备条件并且业界正在实践。Istio 的设计上特别强调其架构上的可扩展性，即通过框架定义与实现解耦的方式，以多平台支持的 Adpater 对接不同后端，支持 Consul、Kubernetes、Eureka（旧版本）、Cloud Foundry 等平台。但就现阶段实现，从代码或者文档的细节去细看其功能，Istio 和 Kubernetes 结合最紧密，也是业界公有云运营商普遍支持的一种方式。Istio 和 Kubernetes 在集成中各有分工共同成就了基于容器的微服务治理及运行平台的最佳实践。

- 场景上，Kubernetes 为应用负载的部署、运维、扩/缩容等提供了强大的支持，并提供了一定的服务发现和负载均衡能力[7]，但是较深入细致的流量治理能力，由 Istio 补齐。
- 架构上，Istio 和 Kubernetes 更是深度的结合，得益于 Kuberntes Pod 的设计，数据面的 Sidecar 作为一种高性能轻量的代理自动注入 Pod 中和业务容器部署在一起，接管业务容器的 inbound 和 outbound 的流量，从而实现对业务容器中服务访问的治理。
- 功能上，Istio 沿用了 Kubernetes 服务发现、服务配置和健康检查的功能。Istio 基于 Pilot 的 Adapter 机制集成 Kubernetes ETCD 实现服务发现，从而避免了两套名字服务；服务配置 Istio 所有路由规则、控制策略配置都是通过 Kubernetes CRD 表达，configmap 进行存取；Istio 节点健康检查能力基于 Kubernetes 的 ReadinessProbe 机制实现。

Istio 与 Kubernetes 支持多集群部署模式，目前有 3 种拓扑类型：多控制面连通拓扑，每个集群有独立的服务网格，各网格之间服务实例无法共享，互相访问不透明；单网络单控制面拓扑，多集群共享一个服务网格控制面，要求集群处于同一网络中，实例之间可互通且 CIRD 不冲突，常使用 VPN 连通多集群；多网络单控制面拓扑，多集群共享一个服务网格控制面，集群有独立的网络，通常以 Gateway 形式连通各个实例。

4.2 Istio 的混合部署模式

Istio 除了与 Kubernetes 完美兼容外，得益于 Envoy Lyft 构建，Istio 支持非 Kubernetes 环境，支持 Nomad/Consul、Cloud Foundry 等环境[8]，并且支持将运行在虚拟机和物理机的服务集成到运行在 Kubernetes 集群上的 Istio 服务网格中，以满足混合云的业务需求。业界也有相应实现，例如腾讯云的微服务框架 TSF，支持混合基础设施架构下服务网格。这样做的好处很明显，即与 Kubernetes 解耦，适用于从遗留系统往服务网格上迁移的场景以及应用服务混合编排的场景。本文对比了 Kubernetes 环境和非 Kubernetes 容器环境下 Istio 的部署与使用上的区别，具体见表 3。

- 部署方面，当 Istio 与 Kubernetes 集成时，由于 Istio 资源可以通过 CRD 方式定义，部署比较方便，通过 yaml 文件进行安装，或通过 heml 包完成安装，可以实现一键部署；在非 Kubernetes 环境下，通过 Docker-compose 部署 Istio 控制面组件运行在容器里，Consul 只配置了 Pilot 组件，其他组件需另行安装。
- 服务发现方面，当 Istio 与 Kubernetes 集成时，延用 Kubernetes 服务发现功能；当与非 Kubernetes 集成时，需要引入服务发现组件和服务发现的机制，业界比较常用的是 Consul 做服务发现组件。
- 认证方面，当 Istio 与 Kubernetes 集成时，可以延用 Kubernetes 的认证体系，即 Service Account。当与非 Kubernetes 集成时，Istio 采用 on-premise 的方式建立用户账户，不需要依赖平台的认证体系。
- Sidecar 注入方面，当 Istio 与 Kubernetes 集成时，Sidecar 注入是通过 Kubernetes 的 Init Containers 进程形式将 Istio-init 和 proxy 注入 Pod 中，通过 Kubernetes 的 Webhook 实现 Sidecar 自动注入。当 Istio 与非 Kubernetes 系统集成，Sidecar 需要手动注入，由于没有 Pod 存在，如果是容器，Sidecar 需要手动地与服务一同注入容器中，如果是虚拟机，则需要在成为服务网格节点的每个虚拟机上运行 Istio Sidecar 进程。
- 流量路由方面，由于 Istio 针对 Kubernetes 实现了网络插件（Istio CNI Plugin），支持自动写入 iptables，让 Pod 所在的 network namespace 的网络流量转发到 proxy 进程，而非 Kubernetes 环境下需要手动设置 iptables 规则，实现服务的流量拦截，较多人工干预。
- 应用部署方面，当 Istio 与 Kubernetes 集成的时候，应用部署可以采用 kubectl 方式、

表 3 集成模式对比

对比项	基于 Kubernetes	基于 nomad&Consul
控制面部署	Heml/CRD 资源安装	Job 方式/容器
服务发现	ETCD	Consul
认证方式	基于 Kubernetes 认证体系	Istio 支持本地认证
注入 Sidecar	自动、手动	手动、打包到任务组
Sidecar 注入位置	Pod	任务组
Sidecar 流量路由	系统自动修改 iptables 规则	需设置 iptables 规则
应用部署	kubectl apply/create、helm	Docker-compose 方式

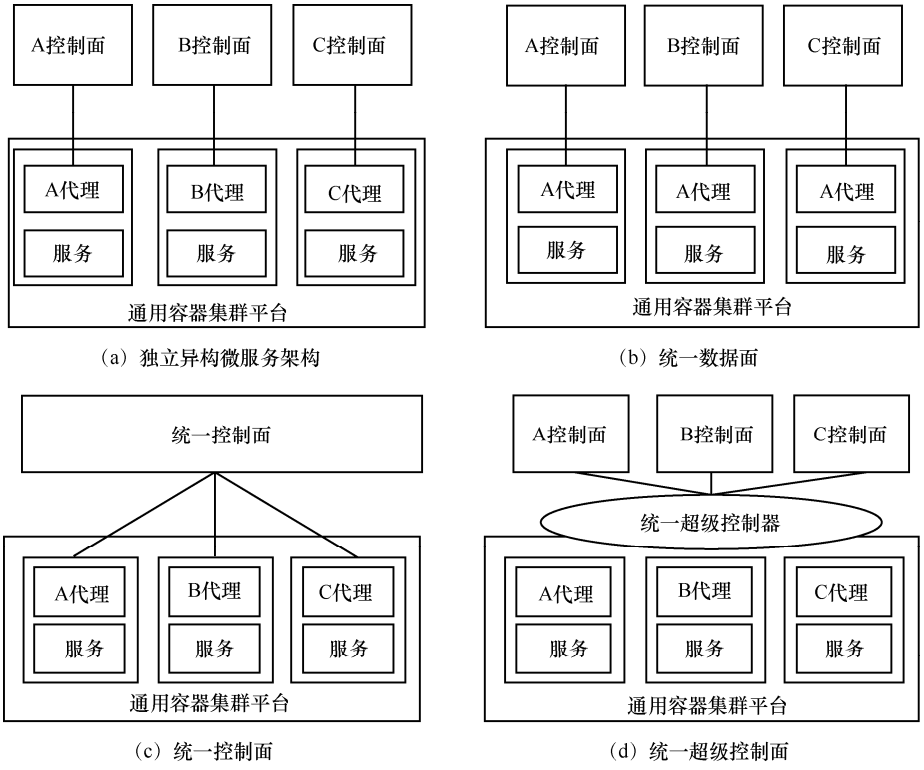
helm 方式部署，采用 helm 的方式可以大大地简化微服务部署的流程。

4.3　异构微服务框架集成

在运营商 CT 域中，存在多专业 VNF 承载需求，不同专业可能选择不同的微服务架构实现。同时，伴随着 IT 域多地域部署、安全等级隔离、多云和混合云、异地容灾等复杂的场景需求，多集群下异构微服务框架也成为下一步需要研究的重点工作。异构微服务架构即是在统一的容器集群管理平台之上（目前最主流的是 Kubernetes[7]，以下简称平台）对接不同技术架构的微服务框架的解决方案。

异构微服务框架支持有 4 种模式，如图 2 所示。

- 第一种是独立异构微服务框架，从技术上来讲，共享根 CA 证书的多个不同集群（具有不同的控制面）上的服务之间可以有效地通信。例如 spring cloud、ServiceComb、Istio 并存，每种框架都有独立的控制面和数据面，平台仅进行资源隔离和 UI 路由，目前现有的实现有 ServiceStage，支持 Spring Cloud、Istio 和 ServiceComb。
- 第二种是统一控制面、异构数据面，这种有可分成两个子方案，一种是南向接口适配方案，如基于 Istio 接入 Go-chassis、Envoy；基于 ServiceComb 接入 Java-chassis、Mesher；另一种是超级控制器方案，通过超级控制器南向对接异构微服务控制器，即在 Istio、Linkerd 等之上封装一层独立的控制器，通过超级控制器将统一控制接口翻译成异构控制器可识别的控制指令。与第一种方案的区别是平台可以有统一的控制器接口，目前由微软、Linkerd 等合作开展的开源项目 SMI（service mesh interface）即采用这样的思路，定义了控制面的接口标准，允许网格提供者提供自己的最佳实现。



（a）独立异构微服务架构　（b）统一数据面

（c）统一控制面　（d）统一超级控制面

图 2　异构微服务框架支持的 4 种模式

表 4　5G 云原生应用和互联网业务云原生应用的主要差异

对比项	互联网云原生应用	5G 云原生应用
服务拆分	几乎适用于所有模块	只适合于控制面，不适合数据面
接口开放	基本采用 REST API	需考虑接口性能，GRPC 可能更常用
应用状态	应用无状态，状态保存在数据库中	原则上状态与业务分离，网络状态实时性对状态缓存性能要求更高
硬件解耦	软硬件完全解耦	可要求硬件提供特定功能，如高速存储、网络硬件加速等
弹性扩展	完全水平扩展	水平扩展与垂直扩展相结合
抗脆弱性	在 3 条 9 的设施上提供 3 条 9 服务	在 3 条 9 的设施上提供 5 条 9 服务
网络架构	容器组网方式	多网络平面支持，混合组网（虚拟机、容器、物理机）要求
应用属性	互联网公司内应用	跨多厂商、多专业的服务网元

注：“3 条 9”代表 99.9%，“5 条 9”代表 99.999%，“9”越多代表全年服务可用时间越长、服务越可靠

- 第三种是统一数据面代理方案，一个数据面代理对接不同的控制器，例如 Mesher 可以对接 Istio 和 ServiceComb。

5　结束语

随着 5G SBA 的进程推进，基于容器承载 5GC 的微服务网元以及基于微服务进行网元治理将成为运营商下一步研究重点，由于 CT 域网元的切分粒度和 IT 域的粒度不同，对应用的安全性和性能要求也不同，IT 域互联网云原生服务在进行服务化设计时，主要是面向时延不敏感的互联网业务提供“尽力而为”的服务，相比之下，5G 无线通信具有明确的超低时延、高可靠业务场景需求，而服务化架构增加 5G 通信系统的控制信令交互，因而使得 5G 云原生应用和互联网业务云原生应用存在许多方面的差异，主要差异总结见表 4。

5G 云原生应用与互联网云原生的差异，主要集中在性能、扩展性、组网、异构网元支持等方面，针对不同的难点，微服务容器平台需进行研究和优化的方面如下。

（1）针对现有主流服务网格架构的性能瓶颈问题，引入支持 GRPC 接口的 Istio 服务网格方案，需要对 Istio/Envoy 的架构优化，例如减少 Envoy 与 Mixer 的交互次数，集成 VPP 优化 Envoy 性能等。

（2）针对应用容器跨集群部署调度的难题，以满足应用弹性扩展需求，需要引入基于 Istio/Envoy 的跨 Kubernetes 集群服务网格组网方案，支持跨集群微服务应用部署与网络控制。

（3）针对应用的混合组网需求，研究混合环境下（容器、虚拟机、裸机）的 Istio 组网方案，需要解耦 Kubernetes 对 Istio 的功能支持，主要包括：采用 Consul 作为配置中心，脱离 Kubernetes 的 CRD 能力；使用 Envoy 原生的健康管理功能，脱离 Kubernetes 的健康管理功能；增强 Pilot-agent 能力，支持服务自服务注册，摆脱 Kubernetes 的服务注册功能。

（4）针对多厂商、多专业异构微服务部署的难题，提出了异构服务网格网络统一管理方案，引入服务网格超级控制器 SMI 和 Service Mesh Registry，统一服务网格北向接口，支持多微服务框架统一管理。

参考文献：

[1] 李保星. 5G 的“新”—全新的核心网[J]. 通信世界, 2017(13): 12-14.
LI B X. 5G's “new” - a new core network. Communication World, 2017(13): 12-14.

[2] 纽曼. 微服务设计[M]. 北京: 人民邮电出版社, 2016.
Newman. Micro service design[M]. Beijing: Posts & Telecom Press, 2016.

[3] DUA R, RAJA A R, KAKADIA D. Virtualization vs containerization to support PaaS[C]//Proceedings of 2014 IEEE International Conference on Cloud Engineering. Piscataway: IEEE Press, 2014: 41-42.

[4] 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.
YAN L Y, YANG X Z, HE Z W, et al. Research on microservices

scheme of operator service platform[J]. Telecommunications Science, 2018, 34(11): 172-180.
[5] 龚正, 吴治辉, 叶伙荣, 等. Kubernetes 权威指南[M]. 北京: 电子工业出版社, 2016.
GONG Z, WU Z H, YE H R, et al. Kubernetes authoritative guide[M]. Beijing: Publishing House of Electronics Industry, 2016.
[6] YOUSIF M. Microservices[J]. IEEE Cloud Computing, 2016, 3(5): 4-5.

[作者简介]

严丽云（1985– ），女，现就职于中国电信股份有限公司研究院，主要研究方向为云计算、容器技术。

杨新章（1972– ），男，博士，中国电信股份有限公司研究院高级工程师，主要从事业务平台、云计算技术、移动互联网方面的工作。

何震苇（1976– ），男，中国电信股份有限公司研究院工程师，主要从事容器、PaaS、分布式架构、NFV 等方面的研发工作。

林园致（1996– ），女，中国电信股份有限公司研究院研究员，主要从事容器技术、CICD 方面的工作。

侯韶新（1987– ），男，现就职于中国移动通信集团南方基地，主要从事云数据中心的维护工作。