



运营商业务平台微服务化方案

严丽云¹, 杨新章¹, 何震苇¹, 张凌¹, 侯韶新²

(1. 中国电信股份有限公司广州研究院, 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: 运营商在业务驱动和“互联网+”转型背景下, 传统单体应用架构的业务平台存在开发生命周期长、对需求响应慢、效率低、运维难等问题, 对其进行改造势在必行。互联网应用特有的高并发、不间断的特性, 需要架构具备高性能、高效率、高扩展性、易运维等特点, 微服务已成为互联网应用的首选架构, 并快速发展成熟。从运营商业务平台现状及需求出发, 结合微服务关键技术, 从微服务拆分、平台架构、接口设计、DevOps 等方面阐述并实现了业务平台微服务化方案。

关键词: 业务平台; 微服务; 容器

中图分类号: TP399

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2018287

Micro-service solution of operator's business platform

YAN Liyun¹, YANG Xinzhang¹, HE Zhenwei¹, ZHANG Ling¹, HOU Shaoxin²

1. Guangzhou Research Institute of China Telecom Co., Ltd., Guangzhou 510630, China

2. China Mobile Group South Base Co., Ltd., Guangzhou 510640, China

Abstract: The high concurrency and uninterrupted nature of internet applications requires high performance and scalability. Micro-service have become the preferred architecture for internet applications. Under the background of business-driven and “internet +” transformation, it is imperative for operators to implement micro-service transformation of traditional business platforms. The status and requirements of the operator's business platform were analyzed. The micro-service design scheme of the business platform was explained from several dimensions of micro-service splitting, micro-service architecture, interface design and DevOps, combining the key technologies of micro-service and benchmarks multiple technologies.

Key words: business platform, micro-service, container

1 引言

从单体架构, 到服务化, 再到更细粒度的微服务化, 人们对互联网应用软件开发追求孜孜

不倦, 力求在软件开发的复杂度和效率之间达到一个平衡^[1]。在运营商转型背景下以及云、大数据、物联网等新技术催生下, 运营商对 IT 架构的灵活性和高效性需求凸显, 传统架构暴露出巨石

型架构的问题,即开发效率低、可扩展性差、难以运维等,已无法满足业务需求,运营商在 IT 域、业务域和未来网络域(5GC、边缘云等)均有微服务化需求和趋势。

2 业务平台现状分析

运营商基于固定、移动和数据网等核心网络建设了业务平台,为个人和政企用户提供各种信息化应用和行业解决方案等增值服务。运营商业务平台历经多年的发展和演变,现存问题如下:

- 单体架构应用居多,软件架构相对封闭,较少参考和采用业界最佳实践;
- 使用 ESB 或者 Web services 这种重量级的解决方案,接口不标准,难以管理;
- 数据库繁重,运维困难,技术选型受限,不易进行 SLA 分级,易发生数据阻塞;
- 同类功能低水平重复建设,重复造轮子,造成资源浪费、体验不一致、维护困难等诸多问题;
- 系统资源利用率低,弹性伸缩能力差,对故障的应对能力较弱,常因为某个子模块出现性能瓶颈时,为了缓解负载往往只能按照应用整体重复部署,导致其他子模块利用率更低;
- 运维离散化,升级和维护困难,缺少工具,维护难度大。

随着领域驱动设计、持续交付、按需虚拟化、基础设施自动化、小型自治团队和大型集群系统这些技术和理念的流行,微服务也应运而生。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]。一般认为微服务架构应用具备如下特征:小型化,每个服务完成单一职责的业务功能;自治性,每个服务可以独立开发、

构建、部署、运行、升级和伸缩;轻量级通信,服务和组件之间通过轻量级的机制实现彼此间的通信;技术中立,每个服务均可以采用不同的技术,充分发挥不同语言的优势,有利于逐步引入新技术;灵活部署,微服务通常采用容器化的部署方式,进一步提高资源利用率和运维效率;面向故障设计,任意服务节点失效、网络闪断等故障不影响业务正常运行。

微服务以上特性可较好地解决运营商业务平台所存在的问题,为构建面向客户的新型增值服务支撑体系,运营商通过对业务平台进行微服务化改造实施业务网络重构。按业务领域,运营商业务平台大体上可分为管理平台、行业应用平台、公众应用平台和能力平台等几类,各类平台由于其各自特点存在不同的改造难点。管理平台是业务网络的枢纽,主要提供综合业务管理功能、对接产品平台和 IT 系统、在运营商转型过程中发挥核心作用。本文以管理平台微服务化作为示例,分析改造中涉及的技术方案和实现。

3 业务平台微服务化设计

业务平台微服务化主要用于提升平台的高可用性、可扩展性和易维护性,其设计目标及思路主要包括:实现每个服务都围绕着具体业务进行构建,并且能够被独立地部署到生产环境、类生产环境等,可以以微服务原则进行服务拆分实现;每个服务运行在其独立的进程中,以轻量级的容器方式承载,服务与服务间采用轻量级的通信机制互相协作,可以通过轻量级、标准化的接口设计实现;平台可支撑大规模、分布式微服务的运行支撑和运维管控,可以通过引入微服务治理架构实现;实现自动化运维及持续交付持续集成,可以通过引入 CI/CD 流水线设计实现。从微服务拆分、微服务架构设计、接口设计、CI/CD 流水线设计 4 个方面对业务平台微服务化进行研究。



3.1 微服务化拆分

业务平台改造方案需要在架构层面上对应用进行微服务化改造，即将原有一体化的应用按照相关原则拆分成微服务应用，使之更加适合部署在容器云环境，充分利用容器云环境提供的资源和服务，实现高性能、高可用、高容错和可快速交付。

纵观业务平台的业务特点，可分为两类应用：以流程工单为核心的应用，如工单管理、合作伙伴管理等；以数据为核心的应用，如订单、商品、库存等。流程工单之间相互耦合性非常底，共享的基础数据也相当少，容易进行垂直拆分，而基于数据的应用微服务拆分是难点，需要在进行垂直切分的时候考虑到基础数据支撑的底层，然后才是业务流程场景拆分的各个关键业务模块。因此对于以数据为核心的应用，首先考虑水平拆分的问题，在水平拆分后仍然按照领域驱动设计的核心思想来建立共享的领域服务层，再对上层的应用功能模块进行垂直拆分^[2]。

3.2 微服务架构选型

对于被拆分成许多小服务的微服务体系来说，服务之间的依赖和联系比原来更加复杂，对运营和部署有更高的要求。如果没有工具和系统能够提供微服务的治理，微服务就是空中楼阁。目前较为成熟的微服务治理架构主要有 Google 主导的 Kubernetes 和 Istio、Privotal 主导的 SpringCloud 以及阿里主导的 Dubbo。

从架构完整度来看，SpringCloud 架构完整度最高且成熟，但是与之相对应的，其复杂度也很高^[3]。接下来是 Kubernetes 和 Istio，两者单独来看，架构完整度都比 SpringCloud 低，都是主要针对微服务架构的某个方面，Kubernetes 偏向运维，对开发的支持偏弱，Istio 偏向微服务间通信。但是，Kubernetes 和 Istio 可组合起来达到更高的完整度，目前 Istio 还属于项目早期，很多功能还需时间检验；Dubbo 架构完整度较低，功能集中在

服务治理方面，且仅支持 RPC 协议框架，2017 年重新开启维护后架构完整性有所改善，开始针对社区呼声较高的功能进行开发，但是目前还有很多功能处于试用阶段。

从代码侵入性来看，SpringCloud 和 Dubbo 都具有侵入性，这是把双刃剑，一方面可以带来控制的灵活性，另一方面会导致业务代码和非业务代码混合在一起，难以维护，增加了开发人员的负担。而 Kubernetes 和 Istio 都是非侵入性的，从长远来看，非侵入性的框架更易维护，对开发人员更加友好。

从开发语言限制来看，SpringCloud 和 Dubbo 都限制使用 Java 语言，而 Kubernetes 和 Istio 不限制语言。其实这和代码侵入性息息相关，因为框架要侵入代码，所以两者需要使用同构的语言来编写。使用异构语言编写是微服务提倡的开发方式，所以 Kubernetes 和 Istio 更符合微服务的架构设计哲学。

从社区的活跃度来看，SpringCloud 和 Kubernetes 的社区都极其活跃。Istio 虽然发布不久，但是版本更新得非常快，社区活跃度在快速上涨中，被称为下一代微服务架构。Dubbo 国内关注者比较多，现又重启维护，回应社区的热点问题，总体来看社区活跃度中等。

综合评定各方案成熟度和发展趋势，结合运营商自身特点，即在多厂商联合开发运营模式下，非代码入侵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选择 Kubernetes 框架。

3.3 接口设计

3.3.1 内外部通信接口

服务内部通信具有交互频繁、传输数据量不确定性、接口数量多，因此需要拥有高吞吐量、低时延、可扩展、易用性的 RPC 框架。目前主要有 Dubbo、Thrift、gRPC 3 种架构，在诸多方面表现都各具特色，表 1 对这 3 种架构进行了比较，综合评估三者优劣性，Thrift 的跨平台、高性能且

具备较高成熟度等特性，更适合微服务的大型分布式系统场景。

表 1 RPC 框架接口比较

	Dubbo	Thrift	gRPC
跨平台性	不跨平台、开发语言单一	跨平台、多语言	跨平台、多语言
服务治理	支持	不支持	不支持
成熟度	高	高	较低
性能	较低	高	高

与外部系统的通信推荐使用目前流行且成熟度较高、并被广泛使用的技术，同时也能够兼容旧系统接口，便于平滑、顺利地迁移。基于以上原则，选型中采用 REST (representational state transfer) API + SOAP^[4]方式与外部服务进行通信。

3.3.2 接口管理规范选型

业务平台被拆分成微小应用后，以 REST API 标准格式对外提供服务，通过轻量级、无状态、可扩展的 API 网关实现 API 服务流量统一接入、分流和管控。可众多的接口如何调用、管理成为新的问题，服务消费者需要知道目前都有哪些服务，服务提供者需要知道接口都被谁调用，服务管理者需要对关系复杂的接口进行监控和维护。接口管理规范就是解决以上问题的工具。

基于接口管理规范统一微服务 API 的接口规范、文档结构和展现形式，可通过相关工具快速生成前端测试代码和接口文档，通过 API 管理平台实现微服务 API 生命周期管理、API 在线文档、后端服务绑定、流控策略管理、API 监控与统计。

目前主要有 3 种比较主流的 REST 接口描述规范：Swagger、API Blueprint 和 RAML^[5]。表 2 对这 3 种规范进行了比较，三大规范均支持 Top-down 或 APIFirst 的 API 设计模式、在线 API 文档和代码框架生成。只有 Swagger 支持 Bottom-up 或 CodeFirst 的 API 设计模式，便于传统应用服务化改造，同时 Swagger 还支持超过 20 种开发语言，包括当前非常流行的 Go 语言。从成熟

度来看，Swagger 是目前最成熟、使用最广的 REST API 描述规范，同时也被 openAPI 标准组接纳为 openAPI 3.0 的指定规范。因此，Swagger(openAPI) 是目前最适合运营商的多厂商、多技术栈的微服务 API 描述规范，也是业务平台微服务化选型的接口管理规范。

表 2 接口管理规范比较

特性	Swagger	API Blueprint	RAML
成熟度	高	较高	高
活跃度	高	较高	中
标准组织	openAPI	无	无
支持厂商	Reverb	Apiary	Mulesoft
最新版本	3.0	1A9	1.0
描述语言	JSON/YAML	Markdown	YAML
API 设计模式	Top-down/Bottom-up	Top-down	Top-down
API 文档	支持	支持	支持
声明式 API	支持	无	无
客户端 SDK 生成	支持	支持	支持
服务端框架生成	支持	支持	支持
APIMock	支持	部分支持	部分支持
API 编辑器	官方支持	第三方支持	官方支持
JSONSchema	支持	支持	支持
XMLSchema	支持	无	支持
支持语言数	20+	10+	10+

3.4 CI/CD 流水线

微服务的项目数量一般会很多，涉及多个开发团队的合作，通过 CI/CD 工具链，各个开发团队通过持续集成工具向仓库提交构建产物。对于业务平台运营商和软件开发厂商之间存在两种模式的合作界面：模式一是运营商掌握源代码的运维模式；模式二是由厂商掌握源代码由运营商参与运维的模式。需要构建一套可满足以上需求的 CI/CD 流水线，平台所设计的 CI/CD 流水线示意图如图 1 所示。

模式一解决方案：采用常规的 CI/CD 流程，经历开发、自动构建、自动部署 3 个环节。开发团队提交自己代码到仓库后，通过持续集成

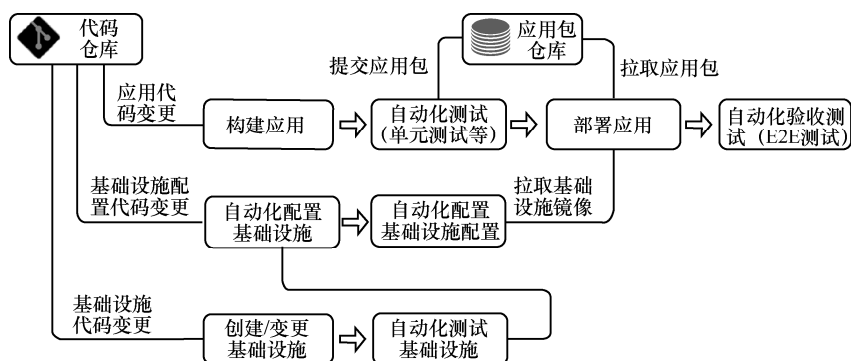


图1 CI/CD 流水线示意图

工具创建构建流水线：代码经过自动构建（一系列自动化测试）之后，最终输出构建产物并将构建产物推送到仓库；通过 Ansible 等自动化部署工具，将构建产物部署到测试环境或生产环境。

模式二解决方案：在代码开发方面仍采用持续构建流水线进行管理，创建基础设施和基础设施配置两条流水线。基础设施流水线对应于基础设施仓库，基础设施仓库对厂商开放，厂商可以将基础设施镜像提交到基础设施仓库，然后触发自动化测试，对镜像的质量进行测试。基础设施配置流水线对应于基础设施配置仓库，运维人员可以对各种基础设置编写配置，拉取所需镜像。运营商通过自动化测试镜像和配置管理介入运维中。

4 业务平台微服务化实现

本文以某运营商管理平台为例进行微服务化

实现。该平台对接产品平台和 IT 系统，提供产品订购、认证鉴权、开通、计费以及用户、SP/SI、业务管理等功能。管理平台的典型逻辑架构如图 2 所示。

4.1 业务平台微服务化关键实现

4.1.1 业务平台拆分实现

业务平台按以下原则进行水平拆分和垂直拆分。

水平拆分：按照不同的业务域进行拆分，拆分的边界要充分考虑业务的独立性和专业性，如订单、商品、库存、用户资源等，按服务的业务功能合理地划出拆分边界，形成独立的业务领域微服务集群，这类系统大都是以数据为核心的应用系统。

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，拆分不同扩展需求和迭代频率的服务，如把公共缓存组件拆分成独立的原子服务，下沉到底层，形成相对独立的原子服务层。

业务平台数据库的原则：数据库独立部署，将大部分数据表设计为不存在直接关联关系，由

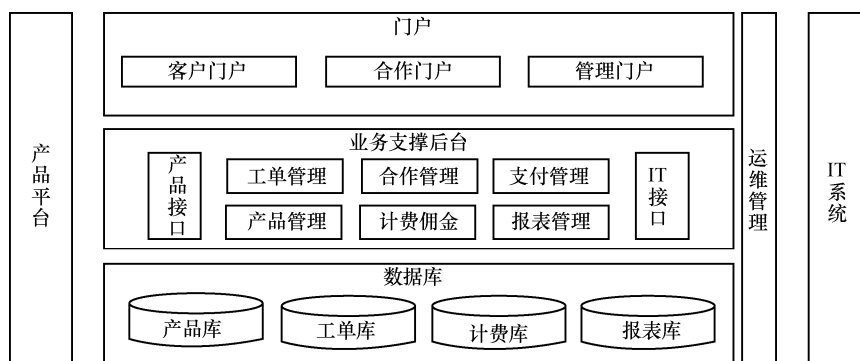


图2 业务平台改造前系统架构

代码将这些表进行关联。

垂直拆分和水平拆分后的分层逻辑图类似于如图 3 所示的系统架构。

- 前端门户：主要用于日常运维管理、客户查询使用和合作伙伴门户工单处理等，前端门户使用 Nginx 进行分发代理。
- 业务服务：是应用运维支撑的集合，包含工单管理、产品管理、客户管理、订单管理、合作伙伴管理、计费管理、安全管理和统计分析等服务。
- 公共服务组件：提供服务监控、日志管理、数据同步等基础服务。
- 接口服务：统一处理系统对接外部的接口，接口模块只进行一些基本的数据校验，涉及业务的流程处理都在工单模块进行，其他模块再通过内部接口统一调用接口模块进行对外的接口调用。
- 数据库集群：独立的服务维护独立的数据库，可以进行独立技术选型、SLA 保证，

使用 MySQL 主从模式，采用 Mycat 进行读写分离。

4.1.2 业务平台的容器化架构实现

将微服务装入容器进行独立的部署和扩展，使其具有可移植性，成为一种必然的趋势^[5-6]。Kubernetes 为 Docker 容器集群提供调度服务，服务注册、服务发现、资源调度、均衡容灾、动态扩缩容等支持^[7]。业务平台容器化架构如图 4 所示，应用平台整体部署在容器资源池中，应用的所有部件均以镜像形式打包，部署在同一个容器集群甚至同一个名字空间中，以实例（pod）或容器形式运行，部件之间通过容器资源池的容器网络通信。

4.1.3 业务平台内外部接口实现

业务平台内外部接口如图 5 所示，内部服务通信采用 Thrift、Redis 消息，前端门户、接口服务与业务服务之间通过 Thrift 协议进行交互；工单服务通过 Redis 存储实现消息列队响应处理。外部通信采用 REST API + SOAP，管理员、合作

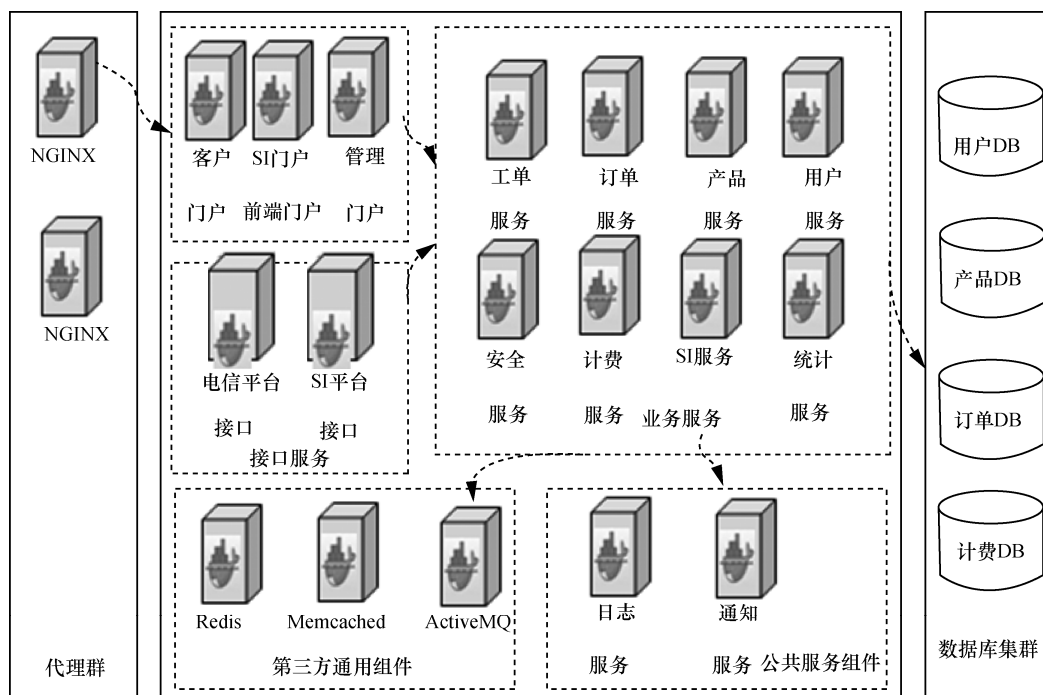


图 3 业务平台微服务化系统架构

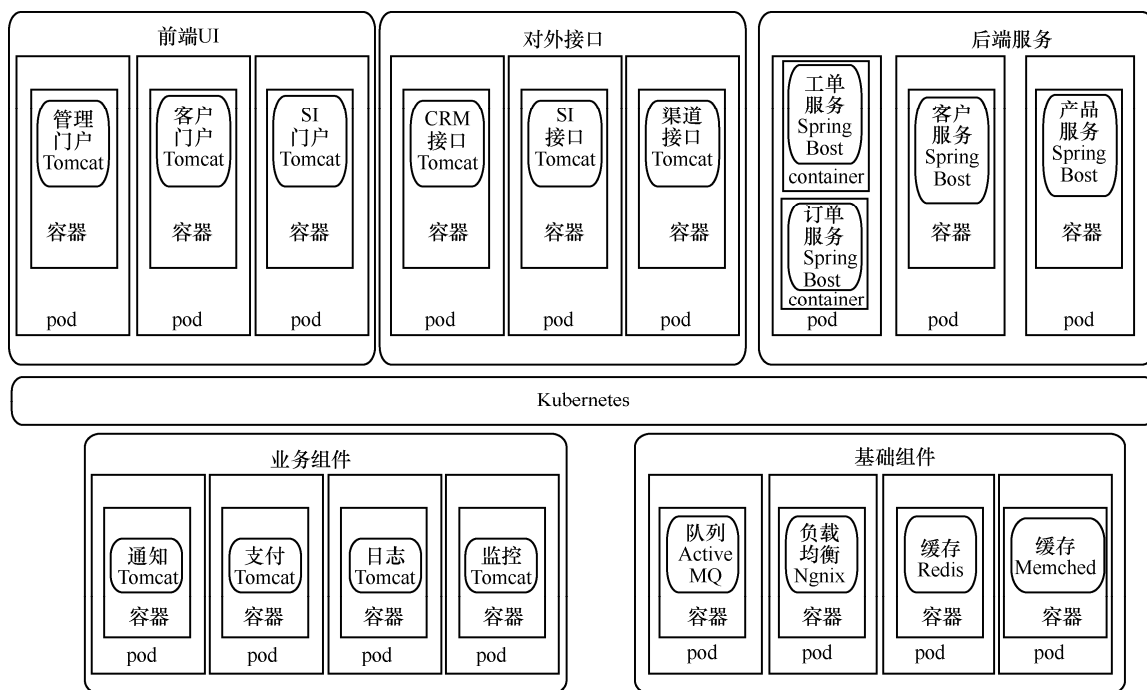


图4 业务平台容器化架构

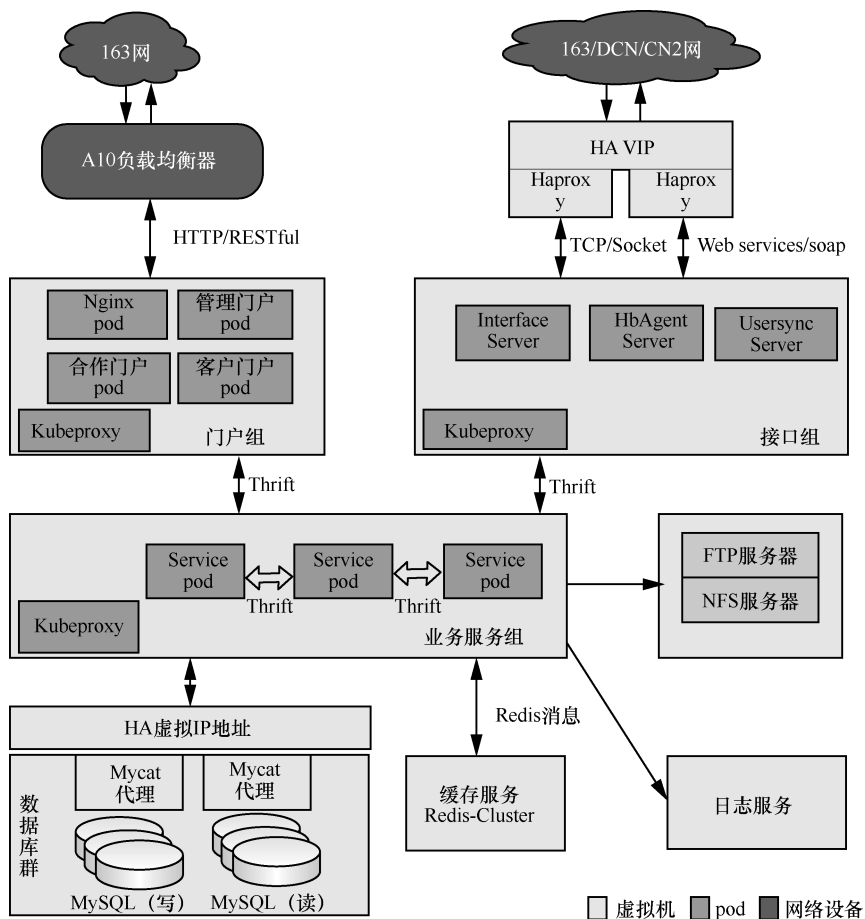


图5 业务平台内外部接口

伙伴及客户门户页面通过 AJAX + RESTful 接口与后台交互数据；小数据量类型业务与 IT 系统及合作伙伴平台通过 Web services 接口进行交互；大数据量类型业务与 IT 系统以及合作伙伴平台通过 Socket 接口进行交互。

4.2 业务平台微服务化步骤

第 4.1 节对业务平台微服务化的关键技术方案进行了分析，针对基于微服务架构的业务平台微服务化改造给出以下主要步骤。

步骤 1 微服务拆分。按业务上下文边界拆分服务，将需要一起变化的相关功能放在同一个服务中，确保拆分的服务是可独立部署升级的业务单元^[8]。

步骤 2 架构设计。根据业务类型、业务规模选择合适的微服务架构模式，确定服务纵向分区、横向分层的原则，确定分区之间与分层之间的交互方式与通信机制。

步骤 3 通信接口设计。确定内部服务通信接口和外部服务通信接口，为了降低服务间耦合，服务对外暴露的信息应尽可能少，并充分利用容器云平台的服务发现机制实现服务之间的透明通信。

步骤 4 服务开发。每个微服务是一个可独立部署的小型应用，可以采用现有语言和技术进行开发，服务的配置文件可通过容器云的环境变量或 ConfigMap 等机制统一管理。

步骤 5 持续集成/交付。构建满足运营商运维模式的 CI/CD 流水线，通过 CI 流水线触发服务的持续集成和交付，生成服务镜像存放到镜像仓库中。

步骤 5 服务编排。通过 service、deployment、RC 等配置文件定义服务所依赖的资源 and 部署方式，包括镜像、计算、存储、网络等资源以及需发布的服务端口和监控端口。

步骤 6 服务部署。服务部署分为新服务部署和服务升级部署两种情况，新服务部署需通过

容器管理平台提交服务的相关配置文件和镜像文件，并通过容管平台触发部署。服务升级部署只需更新镜像文件和配置文件的镜像版本号，不必重新上传配置文件。

步骤 7 服务验证。业务运维人员对部署到测试环境上的服务进行端到端测试，验证服务功能是否满足设计要求。

步骤 8 服务上线。通过测试的服务在生产环境中正式上线。可通过命名空间或 label 机制区分测试服务和生产服务。

5 结束语

业务平台改造促进了传统应用向云原生应用演进，促进了传统架构向标准化微服务架构演进，提升了系统开发效率和平滑扩展性；支持业务快速增长，可实现业务系统不停机状态下的迅速扩容，极大提升扩容速度；支持运营式迭代开发支撑，可实现业务系统快速开发测试部署、模块化升级，提升迭代开发效率，由平均一个月变为平均一周；提升系统资源利用率，功能组件复用、资源调度的精细化控制可极大提升系统资源利用率，整合后系统 CPU 平均利用率提升 200%，节省系统资源，整合后虚拟机数量从 64 台下降到 21 台。

结合平台现状业务平台微服务化演进需要分阶段推进：首先将有状态服务进行无状态化，将原系统去状态化并迁移至容器内；其次，根据业务模块重要性、与其他模块的耦合程度或运维过程中操作频繁程度等维度考量，区分暂时不适合容器化的应用，识别优先进行微服务化的应用进行改造；最终，通过对独立的新业务模块直接以微服务方式进行开发集成，持续完善服务监控、CI/CD，让架构最终演化为微服务架构。改造后的平台仍存在不足，主要集中在大量容器、镜像的管理上，后续演进方向包括容管平台可视化管理、分布式存储集成、扁平化容器网络选型、镜像仓



库分级部署管理以及集群软件和管理平台容器化部署等。

参考文献:

- [1] 陈春霞. 基于容器的微服务架构的浅析[J]. 信息系统工程, 2016(3): 95-96.
CHEN C X. Container-based microservice architecture analysis [J]. Information Systems Engineering, 2016(3): 95-96.
- [2] 纽曼. 微服务设计[J]. 北京: 人民邮电出版社, 2016.
Newman. Micro service design[M]. Beijing: Posts & Telecom Press, 2016.
- [3] 廖建新. 大数据技术的应用现状与展望[J]. 电信科学, 2015, 31(7): 7-18.
LIAO J X. Big data technology: current applications and prospects[J]. Telecommunications Science, 2015, 31(7): 7-18.
- [4] FIELDING R T. Architectural styles and the design of network-based software architecture[D]. Irvine: University of California, 2000.
- [5] MEROÑO-PENUELA A, HOEKSTRA R. grlc makes GitHub taste like linked data APIs[C]// ESWC 2016 Satellite Events, May 29-June 2, 2016, Heraklion, Greece. [S.l.:s.n.], 2016: 342-353.
- [6] YOUSIF M. Microservices[J]. IEEE Cloud Computing, 2016, 3(5): 4-5.
- [7] 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 Electronic Industry, 2016.

- [8] 纪晓峰. 微服务那些事儿[M]. 北京: 电子工业出版社, 2017.
JI X F. Thin services for microservices[M]. Beijing: Publishing House of Electronic Industry, 2017.

[作者简介]



严丽云(1985-), 女, 中国电信股份有限公司广州研究院工程师, 主要从事云计算虚拟化、容器技术方面的研发工作。

杨新章(1972-), 男, 博士, 中国电信股份有限公司广州研究院高级工程师, 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广州研究院工程师, 主要从事容器、PaaS、分布式架构、NFV等方面的研发工作。

张凌(1978-), 女, 中国电信股份有限公司广州研究院高级工程师, 主要研究方向为云计算技术及数据中心技术。

侯韶新(1987-), 男, 中国移动通信集团南方基地工程师, 主要从事 IDC 方面的运维、研究工作。