云原生中间件白皮书（2020年）

云原生产业联盟

Cloud Native Industry Alliance，CNIA

2020年7月

版权声明

编写说明

牵头编写单位：中国信息通信研究院

参与编写单位：阿里云计算有限公司、华为技术有限公司、腾讯云计算（北京）有限公司、北京百度网讯科技有限公司、浙江蚂蚁小微金融服务集团、腾讯科技（深圳）有限公司、中国移动研究院、中通服软件科技有限公司、普元信息技术股份有限公司、中移（苏州）软件技术有限公司、北京凌云雀科技有限公司

编写组成员：

中国信息通信研究院：栗蔚、陈屹力、郑立、刘如明、闫丹、焦辉

阿里云计算有限公司：李小平、易立、冯嘉、朱松、石兵

华为技术有限公司：姜宁、俞岳

腾讯云计算（北京）有限公司：罗茂政、韩欣、陈宁国、任秀森、王炜

腾讯科技（深圳）有限公司：许振文

浙江蚂蚁小微金融服务集团：李克鹏、朴昕阳、岳小均、张森

北京百度网讯科技有限公司：杨帆、周岳骞、王璞

中国移动研究院：陈鹏祥

中通服软件科技有限公司：韩凤伦

普元信息技术股份有限公司：孟庆余

中移（苏州）软件技术有限公司：孙方彬

北京凌云雀科技有限公司：陈皓

前 言

随着云原生技术在各个行业快速落地，基于云计算基础设施的应用开发架构已经发生了巨大的变革。中间件作为云平台业务上云的关键组件，起到重要支撑作用。然而，目前云原生时代中间件范畴尚未明确定义，云原生中间件的生态现状并不清晰，普及推广云原生中间件对加速云原生技术落地有着积极作用。

本白皮书首先阐述了云原生和中间件的基本概念，回顾了中间件从电子信息时代到云原生时代的发展历程，总结了当前云原生时代应用的开发特点，同时开创性地提出了设计云原生中间件的十个关键要素，最后梳理了目前典型的云原生中间件服务，并预测了未来云原生中间件的发展趋势。

目 录

一、 概述

（一） 云原生概述

云原生是一系列云计算技术体系和企业管理方法的集合，既包含了实现应用云原生化的方法论，也包含了落地实践的关键技术。云原生应用利用容器、服务网格、微服务、不可变基础设施和声明式 API 等代表性技术，来构建容错性好、易于管理和便于观察的松耦合系统。云原生通过利用可靠的自动化手段对系统做出频繁、可预测的重大变更，让应用随时处于待发布状态。云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用，借助平台的全面自动化能力，跨多云构建微服务，持续交付部署业务生产系统。[1]

经过几年的发展，云原生的理念不断丰富和落地。中国信息通信研究院经过多方面研究，总结云原生的概念为“适合云的应用”和“好用的云架构”[2]。基于云原生的技术和管理方法，用户能够更好地把业务生于云或迁移到云平台，从而享受云的高效和持续的服务能力。

（二） 中间件定义

中间件是指网络环境下处于操作系统、数据库等系统软件和应用软件之间的一种起连接作用的分布式软件。中间件主要解决异构网络环境下分布式应用软件的互连与互操作问题。提供标准接口、协议，

[1] 引自 CNCF 对云原生的定义
[2] 引自《云原生技术实践白皮书（2019 年）》

屏蔽实现细节，提高应用系统易移植性为其主要技术优势。它使用户能使用一种脚本语言来选择和连接已有的服务，从而生成简单程序的软件开发工具。中间件涉及软件的各个领域，是供公用应用程序编程接口的软件。中间件在操作系统、网络和数据库之上，应用软件的下层，总的作用是为处于自己上层的应用软件提供运行与开发的环境，帮助用户灵活、高效地开发和集成复杂的应用软件。中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源。

二、 中间件发展历程

（一） 中间件的起源

1946 年，世界上第一台电子计算机 ENIAC 的诞生，标志着人类进入了电子信息时代。上世纪六十年代，最早的程序语言 Fortran 出现，标志着现代意义上的软件诞生。为了更好地支撑应用软件系统，中间件随之登上历史舞台。作为连接应用和底层资源的桥梁，中间件能够屏蔽基础硬件、操作系统和通讯协议的异构性，为应用开发者提供统一标准的交互界面。

随着软件的不断发展及其规模的不断增大，分布式系统应运而生。随之出现的还有异构分布网络环境下软件系统的通信、互操作、协同、事务、安全等共性难题。不断出现的新业务需求驱动应用架构和信息系统持续演进，进而要求中间件不断地凝练共性问题，提供标准化的

支撑机制。BELL 实验室于 1990 年开发的 Tuxedo 系统，标准化了分布式交易事务的控制方式。IBM 在 1994 年发布了消息队列服务 MQ，标准化了不同应用程序间异步通信的方式，解决了分布式系统中通讯服务问题。

中间件作为解决共性问题的标准化工具，大大降低了系统的复杂度和协作成本，逐渐成为网络应用的基础设施。

（二） 互联网时代的中间件

到了互联网时代，用户数量爆发增长导致了互联网业务的快速增长，越来越多的应用程序开始部署在分布式的网络环境里运行。传统中间件以类库和框架的形式来加强应用能力，标准化程度和交互性能亟待提升。由于中间件构件模型类库和架构没有统一的标准，不同节点下的中间件自身在构件描述、发布、调用、互操作协议及数据传输等方面呈现出巨大的差异性。另外，以类库和框架的形式提供能力必然使得中间件与业务应用有极强的耦合度，存在可移植性差、适应性低等问题，进而使得应用在不同分布式节点上的交互变得困难重重。

另一方面，爆发增长的用户带的巨大流量和数据也冲击互联网应用，耦合度高的传统中间件难以适配动态多变的流量的互联网环境。在设计之初仅考虑支撑当前应用的能力也使得中间件在技术上具有较大的局限性，在复杂的分布式互联场景下无法很好的支撑上层应用系统。与此同时，多变的互联网流量对基础资源的灵活配置的需求也史无前例的增大，快速增长的业务与僵化的 IT 基础设施之间的矛盾

日益严重。中间件需要在存在多种硬件系统的分布式异构环境中，去支撑各种各样的系统软件，以及风格各异的网络协议和网络体系结构。这些痛点驱动着软件与中间件的技术革新，如何使用中间件技术更好的复用业务，提升 IT 基础设施的业务敏捷性，是互联网时代中间件服务应该考虑的关键问题。

（三） 云计算为中间件提供了平台

云计算时代的到来为企业灵活多变的业务和僵化的 IT 基础设施之间的尖锐矛盾提供了完美的解决方案。弹性的云计算资源根据业务流量进行扩缩，提高资源利用率；平台化的资源托管，解决了传统集群的运维繁琐问题；云计算重构了企业 IT 基础设施，在业务系统逐渐迁移到云上的过程中，中间件起到了至关重要的作用。

业务上云的过程往往不是一蹴而就，企业在云计算构建及落地过程中，出于对新兴技术未知的顾虑以及对核心数据资产的保护，往往是共有云、私有云的混合部署，这样对企业业务不会造成太大的影响，同时提高了资源调度的灵活性。在原有 IT 资源、共有云、私有云等不同的平台之间，以及部署其上的不同应用之间进行数据传递和消息互通至关重要，云计算时代的中间件就扮演了实现不同平台上的应用互联互通标准化的重要角色。同时云计算也为中间件技术的发展提供了更广阔的空间和舞台，中间件逐渐成为云基础构建中的一部分，企业用户能够使用云中间件将业务流程逐步迁移到云集成服务上，最终实现灵活扩展和降本增效。

（四） 开源推动中间件技术发展

作为软件代码标准化的方式，开源在云计算兴起过程中发挥了至关重要的作用，开源打破了技术垄断，为企业提供了一个共同制订事实标准的平等机会。蓬勃发展的云计算与快速前行的开源发挥了相互促进的作用。目前，在与云计算相关的虚拟化、容器、分布式存储、自动化运维等领域，开源已经成为技术主流，深刻影响着云计算的发展方向。

在开源技术开放协作的理念下，传统中间件之间相互孤立的鸿沟将不复存在。随着开源的价值逐渐获得国内外公司的认同，越来越多中间件领域的优质项目涌现出来，如阿里开源的分布式服务框架 Dubbo、分布式消息队列 RocketMQ，腾讯开源的微服务框架 TARS 等，已经被国内外众多企业用于生产。

（五） 云原生赋予中间件新的内涵

在容器及编排技术、开源、微服务等云原生理念的带动下，将应用部署到云端已经是不可逆转的趋势。在现有业务代码不变的情况下，想要让分布式系统无缝入云，如何设计云原生中间件以支撑应用的云上变迁成为关键问题。云原生时代的应用更加轻量化，在对外提供功能保持一致的前提下，将与核心业务无关的能力剥离出去。这些能力将以中间件的形式下沉到基础设施中，成为云的一部分，从而加强和改善应用的运行环境，实现应用轻量化。

由此可以窥见云原生赋于中间件新的内涵，即云原生中间件下沉到云基础设施，保持功能不变的情况下与应用解耦，在运行时为应用动态赋能，支撑上层应用系统。云原生中间件是指在公有云、私有云和混合云等动态环境中，用于构建和运行可弹性扩展的应用，持续交付部署业务生产系统的分布式中间件。云原生中间件能提供应用管理、发布部署、运维编排、监控分析、容灾应急等全生命周期管理的 PaaS 能力，支撑云原生应用的开发与管理，满足经典和云原生架构的运维保障需求。云原生中间件在应用开发方面，能提供开发者体验工具支撑、API 开放能力、产品定制能力、微服务中间件平台、服务市场应用商店等，来支持云原生应用的开发与管理。

三、 云原生应用开发特点

传统企业应用随着业务越来越庞大会遇到各类问题。如在运维方面，传统集群运维繁琐，对于人员技能要求高导致运维效率低；业务架构从单体架构向分布式架构转变时，架构改造以及服务拆分带来的技术学习难度高；自建 IT 基础资源使整个应用运行成本极高，且大量计算资源无法被充分利用，资源利用率低。云原生的出现极大改善了企业业务上云的难度，其主要有以下优势：

云生云长，充分利用云平台服务优势。云原生应用可以快速构建并部署到平台上，平台提供了简单快捷的扩展能力并与硬件解耦，提供了更大的灵活性、弹性和跨云环境的可移植性。云原生将云计算作

为竞争优势，原生云意味着将云目标从IT成本节约转向业务增长引擎。

敏捷弹性，致力于高效高可用设计。在传统的旧基础设施故障时，服务是可能会受到影响的。在一个云原生的世界中，团队特别关注于为弹性和高可用进行架构设计。云原生焦点帮助开发人员和架构师设计不受环境中故障影响的在线系统，快速弹性的重建和保持系统可用。

灵活扩展，云原生应用具备多云间扩展的灵活性。公有云的厂商绑定现象一直为人诟病，但是使用支持云原生技术的云平台，企业可以将构建在任何(公有或私有)云上的应用快速迁移，兼具不同云服务商的优势服务能力无需担心锁定[3]。

云原生应用的开发，有如下特点：

（一） 容器与编排

容器化是指将应用整合到容器中并且运行起来的这个过程。容器是为应用而生的，能够简化应用的构建、部署和运行过程。用户将应用及其所需的所有配置、依赖关系和环境变量打包成容器镜像，使应用不再受环境限制，可以在不同的环境下轻松移植。

容器编排提供高性能可伸缩的容器应用管理能力，为容器化应用提供了全生命周期的管理，如应用的发布与回滚、应用配置管理等；容器编排可以根据应用请求的资源量在容器集群中选择合适的节点来调度资源，对于负载过高的业务进行弹性扩容；通过检测容器集群

[3] 引自《云原生技术实践白皮书（2019年）》

中所有节点，自动隔离故障节点并将应用迁移，使应用系统具备较强的自愈能力，极大简化了运维管理的复杂程度。

（二） 微服务化

微服务化是一种这种开发的风格和方法，以开发一组小型服务的方式来开发一个独立的应用系统。其中每个小型服务都运行在自己的进行中，采用轻量的机制来相互通信。这些服务围绕业务功能进行构建，并能通过全自动的部署机制来进行独立部署。这些微服务可以使用不同的语言来编写，并且可以使用不同的数据存储技术。服务之间是松耦合的，可以独立地对每个服务进行升级、部署、扩展和重启等操作，从而可以实现频繁的迭代更新。微服务技术天生支持快速迭代、弹性扩展的特点，使企业能够在不确定性下提升发展速度及抗风险能力。

微服务化的应用易于开发和维护，一个微服务只关注一个特定的业务功能，业务清晰、代码量较少，开发和维护单个微服务相对简单。而整个应用是由若干个微服务构建而成的，所以整个应用很容易维持在一个可控状态。**微服务技术栈不受限制**。在微服务架构中，可以结合项目业务及团队的特点，合理地选择技术栈。例如某些服务可使用关系型数据库 MySQL；某些微服务有图形计算的需求，可以使用 Neo4j。

微服务虽然有很多吸引人的地方，但它并不是银弹。使用微服务架构面临很多挑战，如其运维要求较高，因为更多的服务意味着更多

的运维投入。在单体架构中，只需要保证一个应用的正常运行。而在微服务中，需要保证几十甚至几百个服务服务的正常运行与协作，这给运维带来了很大的挑战。对使用微服务构建的分布式系统而言，系统容错、网络延迟、分布式事务等都会带来巨大的挑战。

目前微服务架构也在不断演进，从初代侵入式架构，到非侵入式服务网格，再到当前火热的利用 Serverless 来架构微服务的多运行时微服务架构（Muti-Runtime Microservices），微服务架构能力更加全面，进一步降低了云原生应用的开发难度。

（三） 计算存储分离

云原生业务应用部分往往设计成无状态的，相比起逻辑较简单的存储部分，运行时出现错误概率较高，**计算存储分离的架构使得计算部分的故障恢复变得简单，从而使系统更加健壮。**分离后的架构职责清晰，**从业务负载来看，**将计算与存储负载解耦，消除了存储负载和计算负载对资源诉求的不平衡；**从资源调度来看，**计算资源、存储资源独立灵活的扩展，通过共享存储池，减少副本数实现降低成本，提高整个系统的资源利用率；在云环境中，集中云上存储能更容易的提供数据保护、精简配置、快照克隆、重删压缩、自动分层等企业级存储服务。

（四） 服务网关

服务网关是一个服务器，是系统的唯一入口，其对系统内部架构进行封装，为每个客户端提供一个定制的 API。服务网关核心要点是，所有的客户端都通过统一的网关接入微服务，并在网关层处理所有的非业务功能。服务网关负责请求转发、合成和协议转换。在云原生应用中，服务网关有以下几个优点：与微服务注册中心连接，实现服务无感知扩缩；服务网关对于无法访问的服务，可以做到自动熔断；方便的实现应用的蓝绿部署、金丝雀发布或 A/B 发布。

（五） 分布式事务模式

传统的企业开发系统往往是以单体应用形式存在的，不会横跨多个数据库。云原生时代应用的分布化使得企业无可避免的要面对分布式架构带来的数据一致性问题。由于大型云原生应用往往是由一系列分布式应用构成的，开发语言和技术栈也相对比较繁杂。在微服务架构盛行的今天，一个看起来简单的功能，内部可能需要调用多个服务并操作多个数据库或分片来实现，这样必然带来分布式事务问题。所以在进行云原生应用设计时需要考虑分布式事务模式。

云原生的解决方案是将分布式事务问题从业务中剥离出来，作为一个独立的技术面来单独管理，以服务的形式给构建在云上的应用提供简单、易用、高效的解决方案，使开发者专注于业务逻辑的设计和开发，不需要考虑分布式事务的问题，从而加快项目交付、迭代的速度。

（六） 业务服务无状态化

状态化的判断是指两个来自相同发起者的请求在服务器端是否具备上下文关系。**无状态服务是指该服务运行的实例不会在本地存储需要持久化的数据，并且多个实例对于同一个请求响应的结果是完全一致的。**当访问该服务的请求到达服务一端后，负载均衡会随机找到一个实例来完成该请求的响应。这类服务的实例可能会因为一些原因停止或者重新创建（如扩容时）。这时，这些停止的实例里的所有信息（除日志和监控数据外）都将丢失（重启容器即会丢失）。

云原生应用为了承载高并发，往往拆分成多组进程，每组进程完成特定的工作，根据并发量用多个副本公共承担流量压力。如果状态全部保存在本地，会给架构的横向扩展带来瓶颈。所以要将云原生应用的业务逻辑的部分设计成无状态的，这样无状态的部分可以很容易的横向扩展，在用户分发的时候，可以很容易分发到新的进程进行处理，而状态保存到后端。

四、 云原生中间件十要素

中间件的使命是服务于上层应用系统，第三章节讨论了云原生应用开发的特点，为了设计出能更好支撑云原生应用的中间件，我们从**基础资源、设计原则、运行时状态、呈现形态**四个维度抽象出以下十个关键要素。

（一） 容器原生

云原生（Cloud Native）架构，部署容器化（Containerization）

云原生中间件应进行云原生架构设计，以容器化形式进行服务部署。容器化部署可支持中间件服务快速启动，可以灵活完成服务及资源的扩缩容。**容器隔离了用户底层 IaaS 资源的差异，利用容器编排可轻松实现多实例（Multiple Service Instances）部署；容器原生的中间件应用程序和容器镜像占用更少的资源，**对于多容器部署的场景有更好的优化策略，提升基础资源利用率。

（二） 服务状态

有状态（Stateful）和无状态（Stateless）服务分离

在云原生时代，中间件架构设计时需要定义服务的状态：有状态部分和无状态部分。**无状态部分主要伴随业务逻辑的产生，**如服务间通信、链路跟踪等。无状态服务是指该服务运行的实例不会在本地存储数据，并且多个实例对于同一个请求响应的结果是完全一致的。**有状态部分是指该中间件可以产生并存储数据，**并且在创建一个新的有状态服务时，可以通过备份恢复这些数据，以达到数据持久化的目的。所以将状态保存在有状态的中间件中，如分布式缓存、消息队列等。

无状态的业务逻辑部分可以很容易的横向扩展，在用户分发的时候，可以很容易分发到新的进程进行处理。在有状态中间件设计时，应考虑扩容时状态的迁移、复制、同步等机制，以更好的支撑无状态的业务层。

（三） 组件模块化

组件模块（Component Modules）

云原生中间件设计时应考虑可插拔、松耦合、可动态编排的组件化特征。每个组件都是高度抽象的、自包含的、封闭的并和其它的组件相有一定的逻辑隔离，使得不同的角色专注于其擅长领域的工作。开发人员可以通过组合模块的形式调度涉及到中间件，快速支撑业务，适用系统的运行时特征。松耦合的中间件让开发人员可以在处理每个中间件时都能够独立于其他中间件来工作。云原生中间件通过功能上的分离，对外提供统一的应用程序编程接口（API）供开发人员调用，使开发者可以专注于每项服务的核心功能，以提供细粒度的功能。

（四） 事件驱动

采用事件驱动架构 (Event-Driven Architecture)

事件驱动架构作为一种应用间集成模式，天然适合云原生中间件的调度和集成。在事件驱动的体系结构中，当服务执行其他中间件可能感兴趣的工作时，该服务将生成一个事件。其他服务使用这些事件，进而执行由该事件触发的任务。事件成为了可以被消费的对象，而不仅仅是在函数间传递的临时参数，从而可以同时被多个中间件消费。中间件不需要直接和生成事件的服务进行交互，仅通过监听事件，触发对应的操作，从而降低了服务内部的复杂度。事件驱动架构具备以下优点：

1. 异步：基于事件的架构是异步的，没有阻塞。这使得中间件可以在工作单元完成后自由地转移到下一个任务，而不用担心之前发生了什么或者接下来会发生什么。
2. 松耦合：中间件之间不需要(也不应该)知道或依赖于其他服务。事件模型下的中间件可以独立的、更容易的更新、测试和部署。
3. 易于扩展：由于服务在事件驱动的体系结构下解耦，而且服务通常只执行一项任务，因此跟踪特定服务的瓶颈，并对该服务(且仅对该服务)进行扩展变得很容易。
4. 恢复支持：带有队列的事件驱动架构可以通过“重播”过去的事件来恢复丢失的工作。当用户需要恢复时，这对于防止数据丢失非常有用。

（五） 可观测

可观测（Observable）

在由微服务和容器等技术形成的高度复杂的应用系统运行态中，可观测性成为云原生中间件必须具备的能力。可观测性包含了监控、告警、日志聚合、分布式跟踪和依赖分析等部分，通过收集处理数据来定位问题，并简化信息的访问，实时深入的观察整个应用系统的健康状态，从业务资源计量等多个维度进行度量。日志、指标和请求跟踪是可观测性的基础。

从 CNCF 早期毕业的项目 Prometheus 就是一个用于观测服务的云原生监控工具。由此可见可观测性在云原生技术中的重要性。

Prometheus 几乎可以监控所有内容，它还整合了像 Grafana 这样主流的可视化工具。

（六） 韧性设计

云原生中间件具备韧性（Resiliency）

具备韧性设计的中间件具有高延时宽容、容错和故障恢复逻辑。可以防止连锁故障，允许快速失败和快速恢复，且具备较强的系统自愈能力和抵抗外部冲击，为其上层运行的应用系统提供高性能、高容错、高安全地支撑。设计具有韧性设计的中间件，必须从两个方面考虑：

在最初设计和规划时，考虑中间件自身的在不同场景下的健壮性和鲁棒性。针对系统在运行过程中，可能出现导致程序崩溃的各种情况的应对能力，如计算机软件在输入错误、磁盘故障、网络过载或有意攻击情况下，如何保障系统安全、稳定运行的措施、设计等。

中间件的功能和特性在设计时，应兼顾外围应用系统的适应能力。因此，基于云原生架构设计应用的同时，选择具有熔断机制的微服务治理框架，将在整体上为云原生应用提供系统鲁棒性，包括超时、重试、限流、服务降级等方式，从而降低整个系统崩溃的风险。

（七） 弹性伸缩

资源可扩展（scalability），服务可弹性（Elasticity）

可扩展指中间件具备资源按需动态伸缩能力，在保证业务连续性前提下，可以独立于其他服务对于底层资源进行扩展或者收缩。随着流量，需求和使用量的增加，中间件应用应具有弹性。弹性意味着当资源根据需求按比例地减少或者增加时，系统的吞吐量将自动地向下或者向上缩放，从而满足不同的需求。弹性是建立在可扩展的基础之上的，并通过添加自动的资源管理概念对其进行了扩充。

（八） 动态部署

服务全生命周期的动态部署（Dynamic Deployment）

云原生中间件具备服务全生命周期的动态部署与发布能力，在完成开发构建之后，能够以多种策略进行发布，如滚动发布、灰度发布、蓝绿发布等；具备多种部署策略，如批量并发部署、任务定时部署、分阶段部署等；具有版本控制功能，如版本追溯与回滚。

由于云原生中间件具备独立的运行进程，每个中间件也可以独立动态的部署。当某个中间件发生变更时无需编译、部署整个应用。如果应用需要多个中间件来进行支撑，那么多个中间件具备一系列可并行的发布流程，使得发布更加高效，同时降低对生产环境所造成的风险，最终缩短应用交付周期。

（九） 统一响应式与声明式的 API

将响应式 API 与声明式 API 统一（Unifies Reactive and Declarative API）

云原生中间件承担了运行时为应用动态赋能的重任，应用与中间件以 API 调用的形式进行通信与控制。响应式 API 描述为了达到某一个效果或者目标所需要完成的指令；声明式 API 描述的是应用期望的目标状态发出的指令。根据云原生的理念，应用无需知道下层中间件及资源的具体实现方式。声明式 API 即可使中间件在调用时无需关注实现细节，在应用运行时动态赋予。

大多数开发人员都熟悉命令式编程模，并希望在云原生应用开发时利用这种经验。同时，开发人员正在采用容器、事件驱动和微服务模型来进行应用的开发，以构建高度并发的应用程序。云原生时代的中间件开发，应将两种开发模型无缝地集成在同一个平台中，从而在一个组织中产生强大的杠杆作用。

（十） 平台化

以云平台（Cloud Planform）的形式提供服务

将中间件功能下沉到基础设施，以云平台的形式对外输出能力，提供中间件的接口供用户按需调用，用户无需关注中间件服务的下层资源调度与运维，更加聚焦轻量级的业务应用。

五、 云原生中间件典型服务

（一） 分布式消息队列

分布式消息系统是指的利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行分布式系统的集成的服务。消

息队列是在消息的传输过程中保存消息的容器。消息队列管理器在将消息从它的源中继到它的目标时充当中间人。队列的主要目的是提供路由并保证消息的传递；如果发送消息时接收者不可用，消息队列会保留消息，直到可以成功地传递它。通过提供消息传递和消息排队模型，它可在分布环境下扩展进程间的通信，并支持多通讯协议、语言、应用程序、硬件和软件平台，实现应用系统之间的可靠异步消息通信，能够保障数据在复杂的网络中高效、稳定、安全、可靠的传输。当前主流的消息队列有 RocketMQ、Kafka、Pulsar、RabitMQ 等。

RocketMQ 是阿里巴巴研发的一款低延迟、高可靠、可伸缩、易于使用的消息中间件。RocketMQ 由阿里巴巴贡献给了 Apache 基金会并于 2017 年成为顶级开源项目。RocketMQ 由 Java 语言开发，采用发布—订阅模式传递消息，具有高效灵活的水平扩展能力和海量消息堆积能力，近年来逐渐获得国内外企业的认可。

Kafka 是由 LinkedIn 公司开发的一个高吞吐量的分布式消息系统，开发它的目标是为处理实时数据提供一个统一、高通量、低等待的平台。Kafka 最大的特性就是可以实时地处理大量数据以满足各种需求场景。Kafka 由 Scala 语言开发，其客户端在主流的编程语言里面都有对应的支持，如 Scala、C++、Python、Java、GO、PHP 等。Kafka 由 LinkedIn 于 2010 年贡献给了 Apache 基金会并成为顶级开源项目。

Pulsar 是由 Yahoo 开发的 Pub-Sub 模式的分布式消息平台，在 2016 年开源，并于 2018 年 9 月毕业成为 Apache 基金会的顶级项目。

其拥有灵活的消息模型和直观的客户端 API，是一种用于服务器到服务器消息传递的多租户高性能解决方案。

RabbitMQ 是一个基于高级消息队列协议实现的消息系统，其服务器端用 Erlang 语言编写，支持多种客户端，如 Python、.NET、PHP、Java、JMS 等。RabbitMQ 在易用性、扩展性、高可用性等方面表现优良[4]。

（二） 分布式事务系统

云原生应用对大规模的事务处理有着较为苛刻的需求。事务处理监控界于 Client 和 Server 之间，进行事务管理与协调、负载平衡、失败恢复等，以提高系统的整体性能。分布式事务系统中间件适用于联机交易处理系统，主要功能是管理分布于不同计算机上的数据的一致性，保障系统处理能力的效率与均衡负载。

业界著名的 CAP 理论告诉我们，在设计和实现一个分布式系统时，需要将数据一致性、系统可用性和分区容忍性放在一起考虑。而在分布式系统中，一致性（Consistency）、可用性（Availability）和分区容忍性（Partition Tolerance）这 3 个要素最多只能同时满足两个。一致性是指分布式环境下多个节点的数据是否强一致。可用性是指分布式服务能一直保证可用状态。当用户发出一个请求后，服务能在有限时间内返回结果。分区容忍性特指对网络分区的容忍性。分布式事务

[4] 郑立，陈屹力，刘如明，等. 基于开源的消息队列云服务研究[J]. 信息通信技术与政策 2020(5):52-56.

中间件的目的是保障分布式存储中数据一致性，而跨库事务会遇到各种不可控制的问题，如个别节点宕机。

目前主流的分布式事务解决方案有：

a) TCC 补偿型事务解决方案

TCC 指的是 Try（尝试）、Confirm（确认）、Cancel（取消），其核心思想是：针对每个操作，都要注册一个与其对应的确认和补偿（撤销）操作。它分为三个阶段：

1. Try 阶段主要是对业务系统做检测及资源预留
2. Confirm 阶段主要是对业务系统做确认提交，Try 阶段执行成功并开始执行 Confirm 阶段时，默认 Confirm 阶段是不会出错的。即：只要 Try 成功，Confirm 一定成功。
3. Cancel 阶段主要是在业务执行错误，需要回滚的状态下执行的业务取消，预留资源释放。

b) 两阶段提交

两阶段提交（Two-phase Commit，2PC），通过引入协调者来协调参与者的行为，并最终决定这些参与者是否要真正执行事务。

c) 本地消息表

1. 本地消息表与业务数据表处于同一个数据库中，这样就能利用本地事务来保证在对这两个表的操作满足事务特性，并且使用了消息队列来保证最终一致性。

2. 在分布式事务操作的一方完成写业务数据的操作之后向本地消息表发送一个消息，本地事务能保证这个消息一定会被写入本地消息表中。
3. 之后将本地消息表中的消息转发到 Kafka 等消息队列中，如果转发成功则将消息从本地消息表中删除，否则继续重新转发。
4. 在分布式事务操作的另一方从消息队列中读取一个消息，并执行消息中的操作。

（三） 分布式配置服务

随着业务的发展，微服务的数量、程序的配置日益增多（各种微服务、服务器地址、参数等），传统的配置文件方式和数据库的方式已无法满足开发人员对配置管理的要求，如配置修改后实时生效、灰度发布、分环境管理配置、完善的权限审核机制等。因此，需要配置中心来统一管理配置，把业务开发者从复杂以及繁琐的配置中解脱出来，只需专注于业务代码本身，从而能够显著提升开发以及运维效率。同时将配置和发布包解藕也进一步提升发布的成功率，并为运维的细力度管控、应急处理等提供强有力的支持。

在云原生环境中，同类型的服务往往会部署到很多个实例上。分布式配置管理服务可以轻松地管理成千上百个服务实例的配置问题。分布式配置服务有以下特点：

1. 权限控制。为开发人员赋予不同的配置权限。
2. 审计日志。所有的修改需要记录操作日志，方便问题溯源。

3. 环境管理。对开发、测试、生成环境下的配置进行隔离。
4. 配置回滚。当发现配置错误，或者在该配置下程序发生异常可以立即回滚到之前的版本。
5. 灰度发布。通过随机修改部分应用的配置进行灰度发布，当测试正常后在推送到所有的应用。
6. 高可用。在配置服务出现问题时，应用可以使用本地缓存配置。

（四） API 网关

API 网关将各系统对外提供服务的微服务聚合起来，所有要调用这些服务的请求都需要通过 API 网关进行，基于这种方式网关可以对 API 进行统一管控，例如：认证、鉴权、流量控制、协议转换、监控等等。

API 网关的流行得益于近几年微服务架构的兴起，原本一个庞大的业务系统被拆分成许多粒度更小的系统进行独立部署和维护，这种模式势必会带来更多的跨系统交互，企业 API 的规模也会成倍增加，API 网关就逐渐成为了微服务架构的标配组件。其主要功能如下：

1. 提供防攻击、防重放、请求加密、身份认证、权限管理、流量控制等多重手段保证 API 安全，降低 API 开放风险。
2. 提供 API 定义、测试、发布、下线等全生命周期管理，并生成 SDK、API 说明文档，提升 API 管理、迭代的效率。
3. 提供便捷的监控、报警、分析、API 市场等运维、运营工具，降低 API 运营、维护成本。

使用 API 网关的最大优点是，它封装了应用程序的内部结构。客户端只需要同网关交互，而不必调用特定的服务。API 网关为每一类客户端提供了特定的 API。这减少了客户端与应用程序间的交互次数，还简化了客户端代码。

（五） 分布式缓存

随着互联网的飞速发展，各行各业对互联网服务的要求也越来越高，互联网系统很多常见的存储类场景都面临着容量和稳定性风险。此时，本地缓存已无法满足需要，分布式缓存由于其高性能、高可用性等优点迅速许多公司接受并使用。

分布式缓存是传统缓存概念的扩展，其中数据被放置在本地的临时存储中以便快速检索。这意味着不同的机器或服务器将其缓存内存的一部分贡献到可由多个节点和虚拟机访问的大型池中。

分布式缓存由一个服务端实现管理和控制，有多个客户端节点存储数据，可以进一步提高数据的读取速率。分布式缓存能够高性能地读取数据、能够动态地扩展缓存节点、能够自动发现和切换故障节点、能够自动均衡数据分区，而且能够为使用者提供图形化的管理界面，部署和维护都十分方便。分布式缓存广泛应用于云原生应用环境中，因为它提供了出色的可扩展性和容错能力。分布式缓存可以跨越多个节点或服务器，这允许它生成只需添加更多服务器即可实现容量扩展。

（六） 链路跟踪服务

随着业务越来越复杂，系统也随之进行各种拆分，特别是随着微服务架构和容器技术的兴起，看似简单的一个应用，后台可能有几十个甚至几百个服务在支撑，一个前端的请求可能需要多次的服务调用最后才能完成，当请求变慢或者不可用时，我们无法得知是哪个后台服务引起的，这时就需要解决如何快速定位服务故障点，这时就需要链路跟踪服务。

链路追踪为分布式应用的开发者提供了完整的调用链路还原、调用请求量统计、链路拓扑、应用依赖分析等工具，可以帮助开发者快速分析和诊断分布式应用架构下的性能瓶颈，提高微服务时代下的开发诊断效率。

链路追踪的主要功能有：

1. 分布式调用链查询和诊断：追踪分布式架构中的所有微服务用户请求，并将它们汇总成分布式调用链。
2. 应用性能实时汇总：通过追踪整个应用程序的用户请求，来实时汇总组成应用程序的单个服务和资源。
3. 分布式拓扑动态发现：用户的所有分布式微服务应用和相关 PaaS 产品可以通过链路追踪收集到的分布式调用信息。
4. 多语言开发程序接入：基于 OpenTracing 标准，全面兼容开源社区，例如 Jaeger 和 Zipkin。
5. 丰富的下游对接场景：收集的链路可直接用于日志分析，且可对接到 MaxCompute 等下游分析平台。

六、 云原生中间件发展趋势

（一） 技术架构上趋于统一

随着云原生技术体系的日益完善，以及开源力度的不断深入，各类云原生中间件技术架构将逐步走向统一，有望彻底解决传统中间件厂商之间产品互不兼容导致的用户绑定问题。云原生中间件设计十要素的理念提供了统一的构建思想和框架模型，使得云原生中间件产品可以跨云跨平台管理，以实现产品的易于开发、易于移植、易于维护、易于演化。统一的技术架构也代表了云原生中间件从单一功能性产品到统一云平台化的重要趋势。

（二） 深度融合云原生热点技术

热点云原生技术如服务网格（Service Mesh）、无服务器架构（Serverless）正在重塑云原生应用开发模式，也为云原生中间件的发展提供了思路。Mesh 化的架构提供了将中间件从业务进程中分离的方法，让中间件 SDK 与业务逻辑代码进一步解耦，利用容器的共享资源实现业务无感知的独立部署、升级，为中间件下沉到基础设施提供了实践路径。Serverless 架构天然适用于事件驱动触发的中间件执行相关任务，如数据计算、请求响应、没有复杂调用的周期任务等。Serverless 化的中间件提供了更好的用户体验，能够做到按需使用、按量收费、IaaS 资源的免运维，用户可以快速部署运行应用，聚焦核心业务。

（三） 云原生中间件领域标准化进程持续深入

当前我国云原生中间件领域技术标准建设仍处于空白阶段，中国信息通信研究院一直紧跟云计算领域风向，密切关注云原生中间件技术的发展变化，相继制定了《分布式应用架构技术能力要求第一部分:微服务平台》、《分布式中间件服务技术能力要求 第1部分：分布式事务服务》、《分布式中间件服务技术能力要求 第2部分：分布式消息队列》系列标准，对云原生中间件产品的服务能力进行要求，引导服务贴近产业实际需求。未来将继续发挥标准的引领指引作用，做好云原生中间件领域的设计规划，持续深化以用户需求为中心的技术规范和能力要求。