微服务引擎

目录

产品简介

产品定义.....3
产品特性.....3
产品优势.....5
应用场景.....7
产品规格.....10
名词解释.....14
使用限制.....17

购买指南

计费说明.....18
订购与退订.....23
欠费与续费说明.....25

快速入门

注册配置中心.....28
云原生网关.....38
微服务治理中心.....56

用户指南

注册配置中心.....87
云原生网关.....127
微服务治理中心.....268
权限管理.....351

最佳实践

注册配置中心.....364
使用云原生网关实现蓝绿、金丝雀发布及AB实验.....379
通过自建网关实现全链路灰度.....382
基于消息队列RocketMQ实现全链路灰度.....395
基于微服务治理中心实现无损上线与无损下线.....407
基于Ingress-APISIX网关实现全链路灰度.....410

常见问题

目录

注册配置中心..... 423
云原生网关..... 436
微服务治理中心..... 438

产品简介

产品定义

微服务引擎 MSE 面向业界主流开源微服务项目，提供注册配置中心、云原生网关、微服务治理能力，助力企业搭建微服务平台，实现微服务应用的快速开发和高可用运维。

MSE产品包含以下子产品：注册配置中心、微服务治理中心、云原生网关。您在构建自己的微服务体系时，既可单独使用某个子产品，也可搭配使用以便获得微服务生态的最佳实践。

子产品	描述
注册配置中心	面向业界主流开源微服务项目，致力于帮助用户发现、配置和管理微服务，实现动态服务发现、服务配置、服务元数据及流量管理等功能。
云原生网关	将传统的流量网关和业务网关合二为一，提供全局性和独立业务域级别的流量管理策略，与后端业务紧耦合策略配置。
微服务治理中心	兼容SpringCloud、Dubbo等开源微服务框架，支持无侵入的接入应用，提供标签路由、灰度发布、无损上下线、服务降级、服务鉴权等服务治理的能力。

产品优势

开源增强

100% 兼容Spring Cloud、Dubbo微服务开源项目，无缝对接K8s，无厂商绑定，并在稳定性、性能、可运维性上提供更强的能力。

低成本

节省自建微服务引擎的人力成本，集成流量网关和微服务网关功能，降低50%资源开销，缩短请求时间，降低运维复杂度。

高集成

与天翼云云原生产品体系无缝对接，实现服务的可见、可管、可控，提供一站式的微服务解决方案。

无侵入

即买即用，无需修改任何业务代码、配置、镜像即可接入微服务引擎，提供丰富的微服务治理能力。

产品特性

注册配置中心

注册配置中心面向业界主流开源微服务项目，致力于帮助用户发现、配置和管理微服务，实现动态服务发现、服务配置、服务元数据及流量管理等功能。

功能模块	功能详情	相关文档
配置管理	支持应用环境配置的集中管理能力，提供配置的“增删改查”、监听查询、历史查询、配置回滚、克隆同步等功能，便于用户把控配置历史轨迹。	配置管理
实例管理	支持可视化的实例管理，包括接入实例的基础信息查看，实例重启、单节点重启等功能，便于用户统一管理已接入的微服务实例。	管理实例
服务管理	支持可视化的服务管理，包括服务创建与详细信息查看，服务实例的添加、删除、上下线等，便于用户统一管理已接入的服务。	管理服务
完全标准的引擎使用	完全符合开源软件的标准使用，客户更改引擎接入点地址后，无需修改任务代码，即可使用。	版本特性
权限管理	支持安全相关控制管理能力，包括主子账号权限分级、JWT认证、黑白名单访问控制等，帮助用户更好的把握操作安全。	Nacos引擎访问权限
实例监控	支持可视化的引擎监控能力，包括监控大盘，注册中心、配置中心、推送监控等细分项目监控功能，提供包括连接数、TPS和QPS等34项指标的监控。	引擎监控

云原生网关

云原生网关同时具备传统的流量网关和业务网关功能（Zuul、Spring Cloud Gateway等），提供全局性和独立业务域级别的流量管理策略，支持Nacos和K8s等多种服务发现方式，支持TLS加密通信和多种身份认证方式，构筑安全的流量入口。

功能模块	功能详情	相关文档
服务管理提升应用稳定性	打通Nacos、K8s等多种服务发现数据源，支持多种服务负载均衡策略，通过限流、熔断、降级等策略提升整个访问链路稳定性。	服务管理
丰富灵活的路由策略	支持多种路由策略进行多服务路由转发，从请求路径、请求方法、请求头和请求参数等维度进行规则设置，实现请求限流、重写、Header 设置、跨域等。	路由策略配置
防御鉴权保障应用安全	支持JWT，OIDC等多种鉴权方式，对用户进行身份验证，支持自定义访问控制策略，支持全局或者路由级别的黑白名单配置，最大限度控制安全风险。	安全认证配置

功能模块	功能详情	相关文档
可观测能力快速定位问题	集成天翼云LTS、APM等可观测组件，提供网关实例监控、业务监控、日志分析、全链路追踪等功能，高效定位问题，提高应对核心链路问题的响应速度。	监控分析

微服务治理中心

微服务治理支持SpringCloud、Dubbo等开源微服务框架，无侵入接入应用，提供标签路由、灰度发布、无损上下线、服务降级、服务鉴权等服务治理的能力。

功能模块	功能详情	相关文档
服务管理	支持可视化的服务管理，包括服务创建与详细信息查看，服务实例的添加、删除、上下线等，便于用户统一管理已接入的服务。	查询服务
标签路由	通过标签将一个或多个服务的提供者划分到同一个分组，从而约束流量只在指定分组中流转，实现流量隔离的目的。	标签路由
全链路灰度	通过创建泳道规则将一个或多个应用的相同版本划分到同一个泳道中，从而约束流量只在指定泳道中流转，实现全链路的流量隔离的目的。	全链路灰度
离群摘除	检测Spring Cloud、Dubbo应用实例的可用性并进行动态调整，以保证服务成功调用，从而提升业务的稳定性和服务质量。	离群摘除
无损上下线	通过提供服务预热、延迟注册和微服务生命周期与K8s生命周期对齐等一系列功能保障应用安全发布，无损下线能够在应用下线过程中实现服务消费者无感知。	无损上线
应用防护	从流量控制、隔离控制、热点参数防护、Web 防护等多个维度来保障业务的稳定性，提供更专业稳定的流量防护手段、秒级的流量水位分布分析功能。	流量防护

产品优势

注册配置中心

核心能力	传统平台	使用配置注册中心RCC
高资源利用率	资源获取效率低（>1天） 资源利用率低（<30%）。	自助高效获取资源（分钟级）。动态调整资源（伸缩）。

核心能力	传统平台	使用配置注册中心RCC
高效开发	架构耦合，牵一发而动全身。只能按大颗粒系统发布版本，响应周期长。	基于契约(Open API)的开发模式，架构解耦，让微服务的开发、测试、文档、协作和管控活动标准化、自动化。高性能REST/RPC微服务开发框架，提供开箱即用的工具，降低开发门槛。提供SpringCloud等组件，敏捷高效；支持微服务级滚动升级。
简化配置	配置项复杂，每个环境都需进行单独的配置，易出错。	提供集中配置管理能力。实现配置文件与环境解耦，一次维护，多个环境共用。配置文件支持多版本，可查看历史并快速回滚。
一键发布	手动打补丁方式的方式进行发布，中断业务。	支持一键滚动升级，支持业务升级时新老实例均衡分布，业务不中断。
集中监控	无监控手段，系统出故障无法及时感知，故障难定位。	实时图形化展示应用监控指标，CPU占用、告警、节点异常、运行日志、关键事件等实时掌握。

云原生网关

核心能力	传统平台	使用云原生网关
云原生特性	1.流量网关+业务网关模式，部署运维工作量大，资源消耗多。2.云原生组件支持不够。	1.同时具备流量网关和微服务网关能力，资源消耗减半。2.无缝对接云原生组件，更贴合云原生架构。
开箱即用	需要投入人力处理部署、运维和定制化开发工作。	即买即用型实例，用户只需专注于业务开发，无需关注资源购买及部署运维，更专业、更弹性、更可靠。
高集成	集成云原生组件的能力较弱。	可与自研产品体系无缝对接，例如容器服务、注册中心、日志服务、应用监控等，提供一站式的微服务解决方案。
高扩展	扩展能力较弱。	支持多语言插件化扩展，场景丰富、应用灵活。
监控分析	监控指标不够丰富，实现链路追踪等能力需要的成本较高。	提供丰富的指标监控、日志分析、链路追踪等功能，实现服务的可见、可管、可控。
安全可靠	不支持复杂的安全认证机制。	支持HTTPS加密通信，提供jwt、oidc认证，ip黑白名单等安全策略。
低成本	需要自己维护网关组件。	网关能力托管到云端，节省用户维护成本。

微服务治理中心

对比项	自建治理中心	微服务治理中心
成本	自建主机资源成本。	治理资源全托管，无需自建CPU和内存资源。

对比项	自建治理中心	微服务治理中心
人力成本	需自行搭建治理框架，开发难度大，维护成本高。	无侵入，无需改造应用，直接接入即可使用治理功能。功能持续集成，使用简单便捷。
效率	支持开发环境隔离，多套环境互不干扰。支持自动化回归测试，大大节省测试人力成本。	开发测试提效。
线上发布提效	常规变更为避免出现问题，研发人员不得不在深夜变更，发布效率低下。	支持金丝雀发布、全链路灰度等多种灰度规则控制，轻松实现流量切换，无需深夜变更。
稳定	提供流量控制、熔断降级等流量防护能力，保证线上业务稳定运行。	流量防护。
无损上下线	需要自研。	保证在进行应用发布、重启、下线等操作时，应用可以无损变更，无流量损失。
应用安全	需要自研。	提供微服务间调用鉴权的能力。

应用场景

注册配置中心

分布式协同

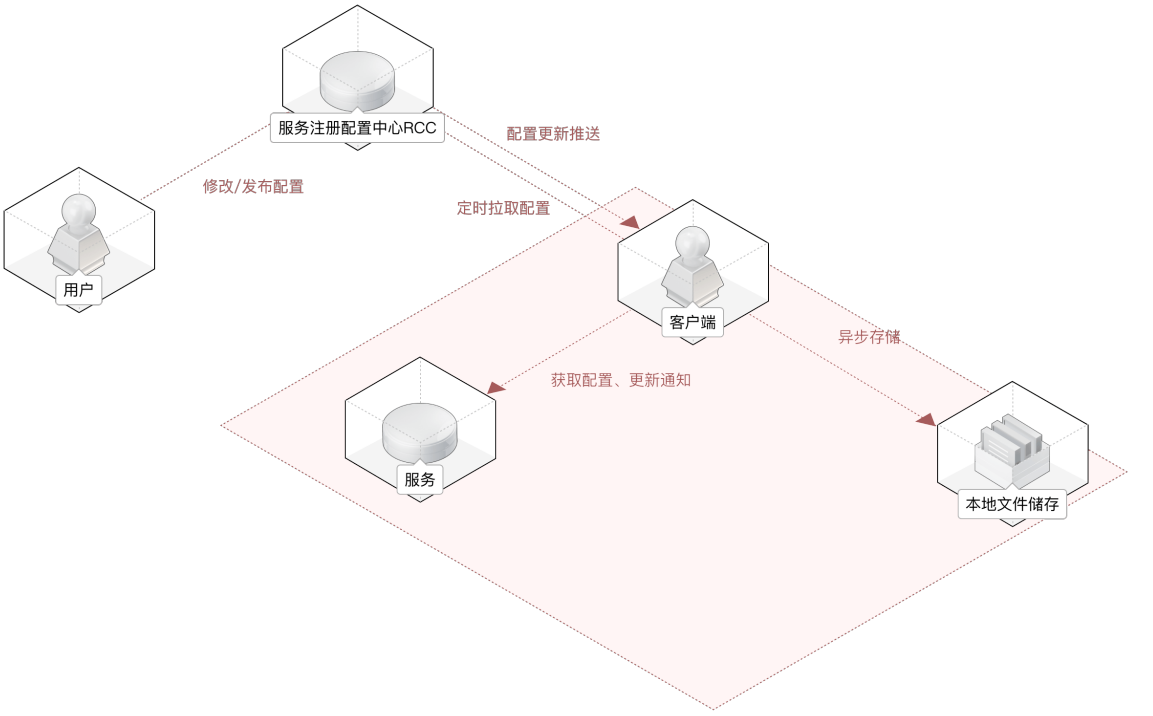
提供分布式系统中服务注册发现及协同调度能力，确保系统状态一致性并完成预期的功能请求，包括状态管理，分布式锁，服务发现，监控检查等。助力用户轻松构建稳定、高可用的分布式服务。

分布式系统中一个绕不过去的问题是如何协同系统中各个分布的模块，确保系统状态的一致性并完成预期的功能，包括状态管理，分布式锁，服务发现等；将分布式系统这一核心必备能力抽象成为组件对外提供服务，使得开发者不用过多了解分布式细节，即可基于这列组件轻松构建稳定、高可用的分布式服务。

分布式配置管理

提供分布式应用配置集中管理能力，包括配置热生效、配置历史查看、配置数据回滚等，支持主流的Nacos 配置中心，支持原生数据无缝迁移，帮助您高效解决分布式环境下多应用的配置管理问题。

在分布式服务架构中，当系统从一个单体应用，被拆分成分布式系统上一个个服务节点后，配置文件也必须跟着迁移分割，这样配置就分散了，不仅如此，分散中还包含着冗余，而配置中心将配置从各应用中剥离出来，对配置进行统一管理，应用自身不需要自己去管理配置。



云原生网关

流量接入&服务治理

传统的服务架构中采用流量网关+业务网关实现流量接入与分发；云原生网关融合了流量网关和业务网关能力，作为微服务架构中东西向和南北向流量的入口，提供了tls卸载，服务发现，请求路由，负载均衡，熔断限流等流量治理能力。

认证鉴权

在云上发布的服务一般需要对前端的请求进行认证鉴权，云原生网关提供了jwt、oidc、ip黑白名单及一系列扩展插件，实现对应用请求的身份认证和访问控制。

可视化运维

微服务网关提供丰富的可观测能力，包括指标监控，日志监控及全链路追踪能力；无需特殊配置即可获取丰富的可观测能力。

微服务治理中心

消除变更风险

依托无损上下线和全链路灰度能力，全面消除变更过程中的风险。

消除系统雪崩风险

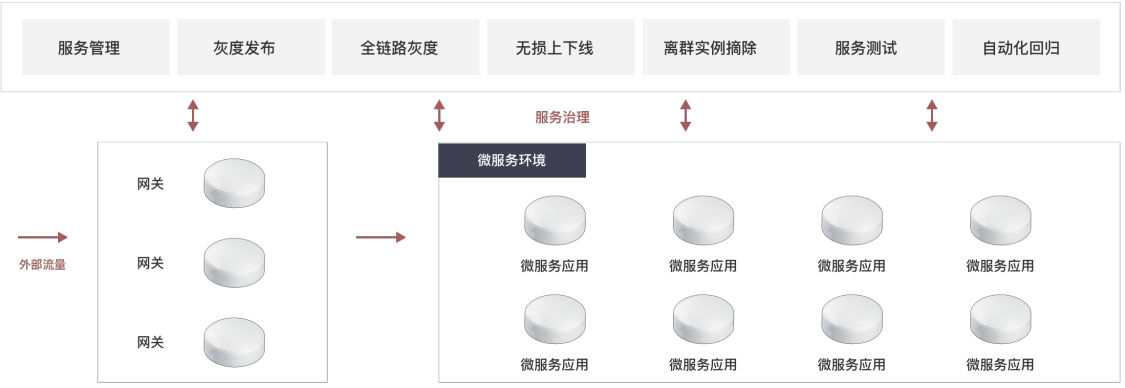
依托于限流、降级、熔断、隔离等能力，可以在出现偶发的流量洪峰和依赖服务出现异常时，有效地限流保护、削峰填谷、隔离故障、降级保护。

实现敏捷开发

依托于开发环境隔离能力，可以在不增加物理机器成本的前提下，低成本扩展出多套逻辑隔离的开发环境，有效地解决环境抢占和冲突问题，实现敏捷开发。

轻松实现灰度上线

依托标签路由功能，轻松实现灰度发布、蓝绿发布等功能。



产品规格

注册配置中心

MSE注册配置中心提供了Nacos、Zookeeper、Eureka、Consul四种引擎的实例供您选用。不同引擎可选择的具体规格参数有差别，典型版本的规格详情和预估的处理能力详见下述列表，您可以参考下述表格来选择开通适合您业务的注册配置中心。

说明

实例的规格系列分为单机版和集群版，其中单机版只提供单节点，不具备高可用性，不适用于生产环境；同时，单机版本也不支持规格升降配操作，无法切换到集群版本，请提前评估；

Nacos

微服务注册中心Nacos引擎包括配置中心和注册中心，提供动态配置、服务发现和服务健康监测等简单易用的特性，经过实践检验具备高性能和高可用性，帮助用户管理配置、注册和发现微服务，更加快捷方便地构建、交付和管理微服务平台。

主机类型	版本	支持实例数
X86-通用型	单机版2C4G	600
X86-通用型	集群版2C4G3节点	6000
X86-通用型	集群版4C8G3节点	12000
X86-通用型	集群版8C16G3节点	24000
X86-通用型	集群版16C32G3节点	48000

说明

如果有服务发现和配置管理两种场景，建议两种场景分别创建MSE Nacos集群进行使用；如果需要同一个集群用于服务发现和配置管理，请将上表的相应数据除以2再进行评估；

如果您使用的Nacos客户端版本是1.x版本，建议您将客户端版本升级到2.x版本，或者将上表的相应数据除以2再进行评估；

上表中的数据为3个节点集群的整体能力评估数据，并非集群中单个节点的能力评估数据，因此在进行容量评估时，不需要将数据乘以3；

Zookeeper

微服务注册中心ZooKeeper是一个分布式应用程序协调服务，是 Google Chubby 的开源实现。利用ZooKeeper的存储配置，实现配置信息的集中式管理和数据的动态更新，保证数据一致性。MSE提供的ZooKeeper企业级服务，分为单机版和集群版两种，前者适用于开发测试，后者致力于在性能、可观测和高可用方面做了诸多提升，用于生产环境。

主机类型	版本	支持实例数
X86-通用型	单机版2C4G	500
X86-通用型	集群版2C4G3节点	4000
X86-通用型	集群版4C8G3节点	7000

主机类型	版本	支持实例数
X86-通用型	集群版8C16G3节点	14000
X86-通用型	集群版16C32G3节点	25000

Eureka

微服务注册中心Eureka是一个基于RestfulAPI#格开发的服务注册与发现组件，用于在微服务框架中实现服务发现、负载均衡和故障转移，与Springcloud生态有较好的兼容。微服务注册中心Eureka引擎对Netflix开源版本进行了优化，致力于构建全托管式、更安全、更稳定、更高性能的分布式服务注册中心。

主机类型	版本	每秒查询请求数	每秒注册请求数
X86-通用型	单机版2C4G	3000	1500
X86-通用型	集群版2C4G3节点	8000	4500
X86-通用型	集群版4C8G3节点	16000	9000
X86-通用型	集群版8C16G3节点	32000	18000
X86-通用型	集群版16C32G3节点	64000	36000

Consul

微服务注册中心Consul引擎是一款分布式高可用的服务发现和配置共享服务组件，由go语言编写，提供高性能、高可用的服务特性。

主机类型	版本	每秒查询请求数	每秒写入请求数
X86-通用型	单机版2C4G	2000	400
X86-通用型	集群版2C4G 3节点	4800	960
X86-通用型	集群版4C8G 3节点	7600	1500
X86-通用型	集群版8C16G 3节点	12000	2400
X86-通用型	集群版16C32G 3节点	19000	3800

云原生网关

云原生网关版本&规格

版本	节点规格	节点数量
基础版	2C4G/4C8G/8C16G/16C32G	1
集群版	2C4G/4C8G/8C16G/16C32G	2～9

x86架构云原生网关性能数据

主机类型	版本	安全水位线QPS（30% CPU，应答包1K，HTTP请求）
X86-通用型	基础版2C4G	2500
X86-通用型	基础版4C8G	5000

主机类型	版本	安全水位线QPS（30% CPU，应答包1K，HTTP请求）
X86-通用型	基础版8C16G	9000
X86-通用型	集群版2C4G3节点	7500
X86-通用型	集群版4C8G3节点	15000
X86-通用型	集群版8C16G3节点	27000
X86-通用型	集群版16C32G3节点	60000

arm架构云原生网关性能数据

主机类型	版本	安全水位线QPS（30% CPU，应答包1K，HTTP请求）
ARM-鲲鹏通用型	基础版2C4G	1500
ARM-鲲鹏通用型	基础版4C8G	3000
ARM-鲲鹏通用型	基础版8C16G	6000
ARM-鲲鹏通用型	集群版2C4G3节点	4500
ARM-鲲鹏通用型	集群版4C8G3节点	9000
ARM-鲲鹏通用型	集群版8C16G3节点	18000
ARM-鲲鹏通用型	集群版16C32G3节点	36000

微服务治理中心

微服务治理中心支持专业版及企业版两种规格，您可以按需选择合适的规格，具体能力差异如下：

核心功能	功能说明	专业版	企业版
服务查询	支持查看应用下服务的提供者、消费者和接口元数据等信息。	√	√
微服务可观测	支持查看最近5分钟的监控数据。	√	√
金丝雀发布	支持在应用发布时，可以为新版本的应用打上gray的标签，通过按流量比例路由或按内容路由的方式，将灰度流量引入带有gray标签的应用中，从而达到小规模验证的目的。	√	√
标签路由	支持将每个服务打上一个标签，通过标签将标签相同的服务分为同一个分组，然后约束流量在同一个分组内流转。	√	√

核心功能	功能说明	专业版	企业版
无损上下线	无损上线：支持在服务上线时，提供服务预热、延迟注册服务的能力解决流量损失问题。无损下线：保证应用在下线、重启时流量零损耗。	√	√
错误注入	支持模拟微服务间异常调用。	√	√
离群实例摘除	支持监测下游实例的可用性，并摘除异常实例。	√	√
推空保护	支持当注册中心返回了空列表，此时客户端忽略该空返回的变更，从缓存中获取上一次正常的服务端地址进行服务访问。	√	√
服务鉴权	支持为提供者的服务设置鉴权规则，允许或拒绝某个消费者访问服务。	√	√
服务测试	支持在控制台填写调用参数、发起服务调用，并得到服务调用的结果。	√	√
自动化回归	支持通过用例管理和用例集管理能力实现功能快速回归。	√	√
服务Mock	支持模拟真实后端服务。	√	√
事件中心	支持通过事件类型和事件来源维度查询事件记录，感知微服务治理事件。	√	√
操作日志	支持记录关键治理中心操作日志。	√	√
流量防护	支持以流量为切入口，对请求流量进行流量控制、熔断降级和系统保护等操作。	×	√
网关防护	支持针对SpringCloud Gateway和Zuul应用实现流量控制。	×	√
全链路灰度	支持将多个相同版本的应用划分为同一个泳道，通过全链路流量控制的功能将相同版本的应用隔离成一个独立的运行环境（泳道）。	×	√

核心功能	功能说明	专业版	企业版
功能开关	提供了一个轻量级的动态配置框架，可以在项目中快速接入配置，并在控制台实时管理配置项。	×	√
数据库治理	支持SQL监控统计、SQL流量防护、连接池治理、数据库灰度、数据库读写路由等功能。	×	√
全局鉴权	支持通过创建鉴权规则，实现多个微服务之间通信的身份验证。	×	√

名词解释

注册配置中心

Nacos

集群

集群指提供一定分布式协同能力的一组服务所需要的云资源组合，关联了若干云服务器节点、负载均衡等云资源。您可以理解为集群是“同一个子网中一个或多个弹性云服务器（又称：节点）”，通过相关技术组合而成的计算机群体，为容器运行提供了计算资源池。

节点

每一个节点对应一台服务器（可以是虚拟机实例或者物理服务器），实例运行在节点上。集群中的节点数量可以伸缩，可以调整机器规格。

命名空间

用于进行租户粒度的配置隔离。不同的命名空间下，可以存在相同的 Group 或 Data ID 的配置。Namespace 的常用场景之一是不同环境的配置的区分隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离等。

配置

在系统开发过程中，开发者通常会将一些需要变更的参数、变量等从代码中分离出来独立管理，以独立的配置文件的形式存在。目的是让静态的系统工件或者交付物（如 WAR，JAR 包等）更好地和实际的物理运行环境进行适配。配置管理一般包含在系统部署的过程中，由系统管理员或者运维人员完成。配置变更是调整系统运行时的行为的有效手段。

服务

通过预定义接口网络访问的提供给客户端的软件功能。

服务发现

在计算机网络上，（通常使用服务名）对服务下的实例的地址和元数据进行探测，并以预先定义的接口提供给客户端进行查询。

实例

提供一个或多个服务的具有可访问网络地址（IP:Port）的进程。

Zookeeper

Znode

Zookeeper树状数据结构上的结点，可以存储少量数据。Znode分为持久节点、临时节点（Ephemeral）、顺序持久型和顺序短暂型。

会话（Session）

会话是指客户端与zookeeper服务器的连接，Zookeeper中的会话叫session，客户端靠与服务器建立一个长连接来维持一个session，通过这个连接，客户端能够与Zookeeper服务器保持有效的通话，也能向Zookeeper集群发起请求并得到响应。

Watch

watcher（事件监听器）：Zookeeper允许用户在指定的节点上去注册事件监听器watcher，在服务端数据节点发生变化时，Zookeeper会把该变化通知给感兴趣的客户端（即订阅了该服务端节点的客户端）

Eureka

集群

集群指提供一定分布式协同能力的一组服务所需要的云资源组合，关联了若干云服务器节点、负载均衡等云资源。您可以理解为集群是“同一个子网中一个或多个弹性云服务器（又称：节点）”，通过相关技术组合而成的计算机群体，为容器运行提供了计算资源池。

节点

每一个节点对应一台服务器（可以是虚拟机实例或者物理服务器），实例运行在节点上。集群中的节点数量可以伸缩，可以调整机器规格。

服务

通过预定义接口网络访问的提供给客户端的软件功能。

服务注册

客户端（服务提供者）将自身服务信息上传到注册中心的过程。服务注册后，其他服务消费者将可以通过注册中心获取到其服务地址，并发起调用。

服务发现

在计算机网络上，（通常使用服务名）对服务下的实例的地址和元数据进行探测，并以预先定义的接口提供给客户端进行查询。

实例

提供一个或多个服务的具有可访问网络地址（IP:Port）的进程。

云原生网关

域名

一串用点分隔的字符，作为互联网中某个实体的名称；在云原生网关中可以根据请求中的域名信息进行路由匹配，域名还可以关联证书，实现TLS加密访问。

服务来源

网关转发请求到的服务的来源，当前支持天翼云容器引擎和天翼云注册配置中心（Nacos引擎）作为服务来源。

服务

网关路由转发到后端的服务，可以是部署在K8s或者注册到Nacos的服务，也可以是固定地址。

路由规则

一个路由规则可以基于配置匹配路径、header等参数，讲请求转发到指定的后端服务；同时可以配置header、限流等策略。

认证鉴权

网关需要对请求的身份进行校验，当前云原生网关支持jwt、oidc及通过扩展插件的方式实现对请求的身份认证和鉴权。

微服务治理中心

服务契约

微服务接口描述，应用成功接入微服务治理中心后，便可以通过微服务治理中心查看服务接口、路径等API信息。

标签路由

标签路由指的是通过标签将一个或多个服务的提供者划分到同一个分组，并约束流量在指定的分组中流转，实现流量隔离的目的。

离群实例摘除

实时检测下游实例的可用性，并动态隔离异常实例，保证服务成功调用，提升业务的稳定性和服务质量。

无损上线

在应用上线时，对应用做服务预热和延迟注册的操作，避免出现服务抖动或异常调用的情况。

无损下线

无损下线指的是在应用进行重启、下线等变更操作时，能实现无感知下线，保证业务连续无损。

流量防护

流量防护提供专业的流量控制、熔断降级手段，保证应用高可用运行。

流量控制

实时监控流量的QPS或并发线程数等指标，控制达到阈值的流量，避免应用被瞬时流量高峰冲垮，从而保证应用的高可用性。

熔断降级

限制不稳定资源的调用，让请求快速失败，避免影响其他资源而导致级联错误。

使用限制

注册配置中心

限制项	限制值
Nacos单实例命名空间数量上限。	50个
Nacos单实例配置文件数量上限。	10000个
Nacos单个配置文件大小上限。	100 KB
ZooKeeper单个Session所创建的临时节点（ephemeral）类型上限。	2000个
ZooKeeper数据管理-数据导入文件大小限制。	200M
Eureka单节点服务实例注册数量上限。	1000个

云原生网关

限制功能	说明
IP黑白名单	当前基于请求云原生网关的网络层IP进行黑白名单管控，对于使用ELB的场景，网关看到的网络层IP是ELB的出口IP。后续会优化该功能，支持基于X-Forwarded-For中的IP进行管控的能力。

微服务治理中心

概述

在使用微服务治理中心之前时，您需要通过本文提前了解微服务治理中心的使用限制。

版本限制

模块	限制	说明
微服务框架-Spring Cloud	Spring Cloud Edgware及以上版本。	客户端：Feign、RestTemplate。
微服务框架-Dubbo	Dubbo 2.5.3及以上版本。	无
接入方式	支持ECS、云容器引擎应用接入。	如果使用ECS接入，需要在ECS的/etc/resolv.conf文件首行添加nameserver 100.95.0.1。
注册中心	支持应用使用Nacos、Eureka和Zookeeper三种引擎。	无
JDK版本	支持JDK1.8+版本应用接入。	无
VPC	应用所在的VPC需手动接入微服务治理中心。	接入方法，见文档ECS微服务应用接入MSE治理中心。

当您的应用满足上述限制后，通过ECS或云容器引擎接入微服务治理中心，可以使用微服务治理中心相关功能。

计费说明

注册配置中心计费说明

计费项

微服务引擎-注册配置中心的计费项包含注册配置中心托管的普通实例费用、数据盘费用，其中数据盘费用单独计算。

计费方式

目前注册配置中心提供包年包月和按需付费两种付费模式。

- 按需付费：按需付费是后付费模式，您只需按实际情况进行使用，然后按照使用账单付费即可。为避免造成您的损失，如果您不想再使用此产品，则需要您在微服务引擎-注册配置中心-实例管理页面内将指定实例退订，系统才会正式停止计费。
- 包年包月：包年包月是预付费模式，按照包年包月的方式进行付费购买。只要您实际接入的实例数不超过您购买的实例数，不会造成额外的费用。包年包月的模式下，您需要在到期前进行续费或选择自动续费的方式，确保产品持续可用。

计费规则

实例计费规则

注册配置中心实例计费规则如下：

版本类型	主机类型	CPU(核）	内存（GB）	按需（元/节点/小时）	包月（元/节点/月）
注册配置中心单机版	X86-通用型	2	4	0.461	221
注册配置中心集群版	X86-通用型	2	4	0.77	369
	X86-通用型	4	8	1.413	677
	X86-通用型	8	16	2.7	1294
	X86-通用型	16	32	5.274	2528
注册配置中心单机版（鲲鹏）	ARM-鲲鹏-通用型	2	4	0.6	255
	ARM-鲲鹏-计算增强型	2	4	0.8	364
注册配置中心集群版（鲲鹏）	ARM-鲲鹏-通用型	2	4	0.9	425
	ARM-鲲鹏-通用型	4	8	1.7	779
	ARM-鲲鹏-通用型	8	16	3.2	1489
	ARM-鲲鹏-通用型	16	32	6.1	2908
	ARM-鲲鹏-计算增强型	2	4	1.3	606
	ARM-鲲鹏-计算增强型	4	8	2.4	1111

	ARM-鲲鹏-计算增强型	8	16	4.5	2122
	ARM-鲲鹏-计算增强型	16	32	8.7	4144
	ARM-鲲鹏-计算增强型	32	64	17.3	8288
注册配置中心单机版（飞腾）	ARM-飞腾-通用型	2	4	0.6	255
	ARM-飞腾-计算增强型	2	4	0.8	338
注册配置中心集群版（飞腾）	ARM-飞腾-通用型	2	4	0.9	425
	ARM-飞腾-通用型	4	8	1.7	779
	ARM-飞腾-通用型	8	16	3.2	1489
	ARM-飞腾-通用型	16	32	6.1	2908
	ARM-飞腾-计算增强型	2	4	1.2	564
	ARM-飞腾-计算增强型	4	8	2.2	1033
	ARM-飞腾-计算增强型	8	16	4.2	1973
	ARM-飞腾-计算增强型	16	32	8.1	3854
注册配置中心单机版（海光）	X86-海光-通用型	2	4	0.6	255
	X86-海光-计算增强型	2	4	0.8	345
注册配置中心集群版（海光）	X86-海光-通用型	2	4	0.9	425
	X86-海光-通用型	4	8	1.7	779
	X86-海光-通用型	8	16	3.2	1489
	X86-海光-通用型	16	32	6.1	2908
	X86-海光-计算增强型	2	4	1.2	574
	X86-海光-计算增强型	4	8	2.2	1052
	X86-海光-计算增强型	8	16	4.2	2011
	X86-海光-计算增强型	16	32	8.2	3926
	X86-海光-计算增强型	32	64	16.4	7852

以订购注册配置中心集群版4C8G为例，若订购3节点，则价格为：677 * 3 = 2031元/月。

数据盘计费规则

注册配置中心数据盘计费规则如下：

产品规格	按需标准价格（元/G/小时）	包月标准价格（元/G/月）
高IO（SAS）	0.0009	0.4
超高IO（SSD）	0.0017	1.2

说明

当前注册配置中心每节点将默认开通 100G 高IO（SAS）用于存储数据，您无需变更。若有特殊要求，请联系客服进行操作，感谢。

以订购注册配置中心集群版为例，若订购3节点，则价格为：0.4 * 100 *3 = 120元/月。

云原生网关计费说明

计费项

MSE云原生网关的计费项包含MSE托管的网关实例费用和数据盘费用，其中数据盘费用单独计算。

计费方式

目前云原生网关提供包年包月和按需付费两种付费模式。

- 按需付费：按需付费是后付费模式，您只需按实际情况进行使用，然后按照使用账单付费即可。 为避免造成您的损失，如果您不想再使用此产品，则需要您在应用管理页面内删除应用，系统才会正式停止计费。
- 包年包月：包年包月是预付费模式，按照包年包月的方式进行付费购买。 只要您实际接入的实例数不超过您购买的实例数，不会造成额外的费用。 包年包月的模式下，您需要在到期前进行续费或选择自动续费的方式，确保产品持续可用。

计费规则

实例计费规则

云原生网关实例计费规则如下：

版本类型	主机类型	CPU(核)	内存（GB）	按需（元/节点/小时）	包月（元/节点/月）
云原生网关基础版	X86-通用型	2	4	0.84	422
	X86-通用型	4	8	1.68	844
	X86-通用型	8	16	3.35	1688
	X86-通用型	16	32	6.7	3376
云原生网关集群版	X86-通用型	2	4	0.84	422
	X86-通用型	4	8	1.68	844
	X86-通用型	8	16	3.35	1688
	X86-通用型	16	32	6.7	3376

云原生网关集群版(鲲鹏系列主机)	ARM-鲲鹏	2	4	1.764	886.2
	ARM-鲲鹏	4	8	3.528	1772.4
	ARM-鲲鹏	8	16	7.035	3544.8
	ARM-鲲鹏	16	32	14.07	7089.6
云原生网关集群版(飞腾系列主机)	ARM-飞腾	2	4	1.176	590.8
	ARM-飞腾	4	8	2.352	1181.6
	ARM-飞腾	8	16	4.69	2363.2
	ARM-飞腾	16	32	9.38	4726.4
云原生网关集群版(海光系列主机)	X86-海光	2	4	1.26	633
	X86-海光	4	8	2.52	1266
	X86-海光	8	16	5.025	2532
	X86-海光	16	32	10.05	5064

以订购云原生网关集群版2C4G为例，若订购3节点，则价格为：422*3=1266元/月。

数据盘计费规则

云原生网关数据盘计费规则如下：

产品规格	按需标准价格（元/G/小时）	包月标准价格（元/G/月）
超高IO（SSD）	0.0017	1.2

以订购云原生网关集群版为例，若订购3节点，每节点50G，则价格为：1.2 * 50 * 3 = 180元/月。

微服务治理中心计费说明

计费项

MSE微服务治理中心的计费项为实例使用费用。

计费方式

目前微服务治理中心提供按需和资源包抵扣两种付费模式。

按需付费

按需付费是后付费模式，您只需按实际情况进行使用，然后按照使用账单付费即可。为避免造成您的损失，如果您不想再使用此产品，则需要您在应用管理页面内删除应用，系统才会正式停止计费。

微服务治理中心按需计费具体计费规则如下：

版本系列	按需（元/实例/小时）
专业版	0.04元
企业版	0.125元

资源包抵扣

您在开通按需资源后，可购买对应的资源包用量，购买资源包后，您使用产品的用量将会使用资源包进行抵扣，超出资源包量的部分将按需计费，资源包到期后资源包的用量将失效，不再用于抵扣。

专业版资源包：

资源包类型	资源包规格	资源包有效期	资源包价格	举例说明
专业版资源包	7200实例*小时	1个月	270元	10个Agent，使用时间为30天，需要资源包的规格为10×24×30=7200 Agent*小时。
专业版资源包	36,000实例*小时	3个月	1250元	50个Agent，使用时间为30天，需要资源包的规格为50×24×30=36000 Agent*小时。
专业版资源包	144,000实例*小时	6个月	4600元	200个Agent，使用时间为30天，需要资源包的规格为200×24×30=144000 Agent*小时。
专业版资源包	360,000实例*小时	6个月	10500元	500个Agent，使用时间为30天，需要资源包的规格为500×24×30=360000 Agent*小时。
专业版资源包	876,000实例*小时	12个月	22900元	100个Agent，使用时间为365天，需要资源包的规格为100×24×365=876000 Agent*小时。
专业版资源包	1,752,000实例*小时	18个月	39900元	200个Agent，使用时间为365天，需要资源包的规格为200×24×365=1752000 Agent*小时。
专业版资源包	4,380,000实例*小时	18个月	88000元	500个Agent，使用时间为365天，需要资源包的规格为500×24×365=4380000 Agent*小时。

企业版资源包：

资源包类型	资源包规格	资源包有效期	资源包价格	举例说明
企业版资源包	7200实例*小时	1个月	850元	10个Agent，使用时间为30天，需要资源包的规格为10×24×30=7200 Agent*小时。
企业版资源包	36,000实例*小时	3个月	3900元	50个Agent，使用时间为30天，需要资源包的规格为50×24×30=36000 Agent*小时。
企业版资源包	144,000实例*小时	6个月	14000元	200个Agent，使用时间为30天，需要资源包的规格为200×24×30=144000 Agent*小时。
企业版资源包	360,000实例*小时	6个月	32000元	500个Agent，使用时间为30天，需要资源包的规格为500×24×30=360000 Agent*小时。
企业版资源包	876,000实例*小时	12个月	68000元	100个Agent，使用时间为365天，需要资源包的规格为100×24×365=876000 Agent*小时。
企业版资源包	1,752,000实例*小时	18个月	120000元	200个Agent，使用时间为365天，需要资源包的规格为200×24×365=1752000 Agent*小时。
企业版资源包	4,380,000实例*小时	18个月	260000元	500个Agent，使用时间为365天，需要资源包的规格为500×24×365=4380000 Agent*小时。

订购与退订

订购

1. 进入天翼云控制中心，搜索微服务引擎MSE，点击购买图标，即可进入到订购配置页面。

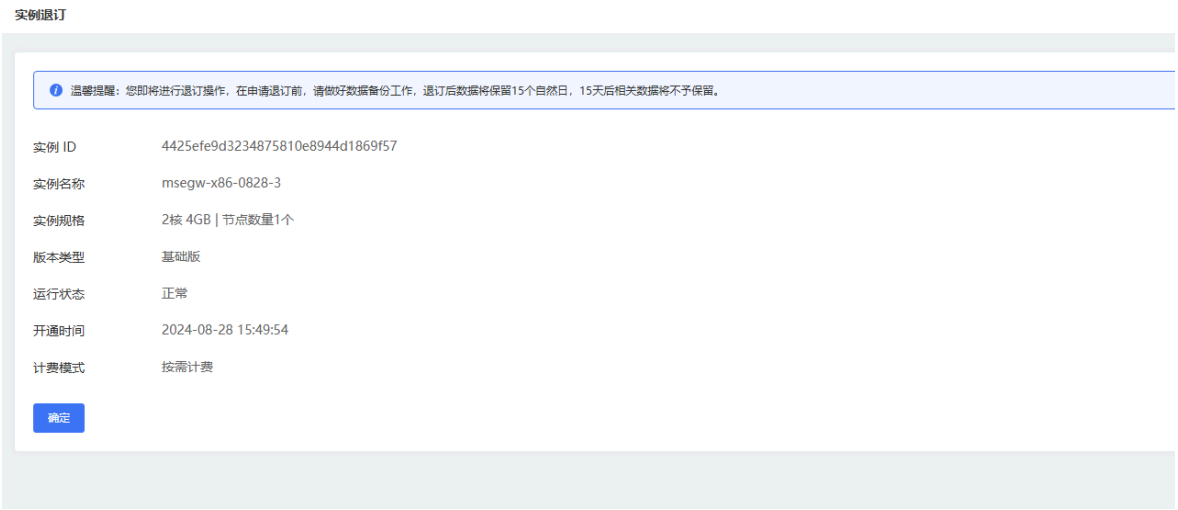
退订

退订规则

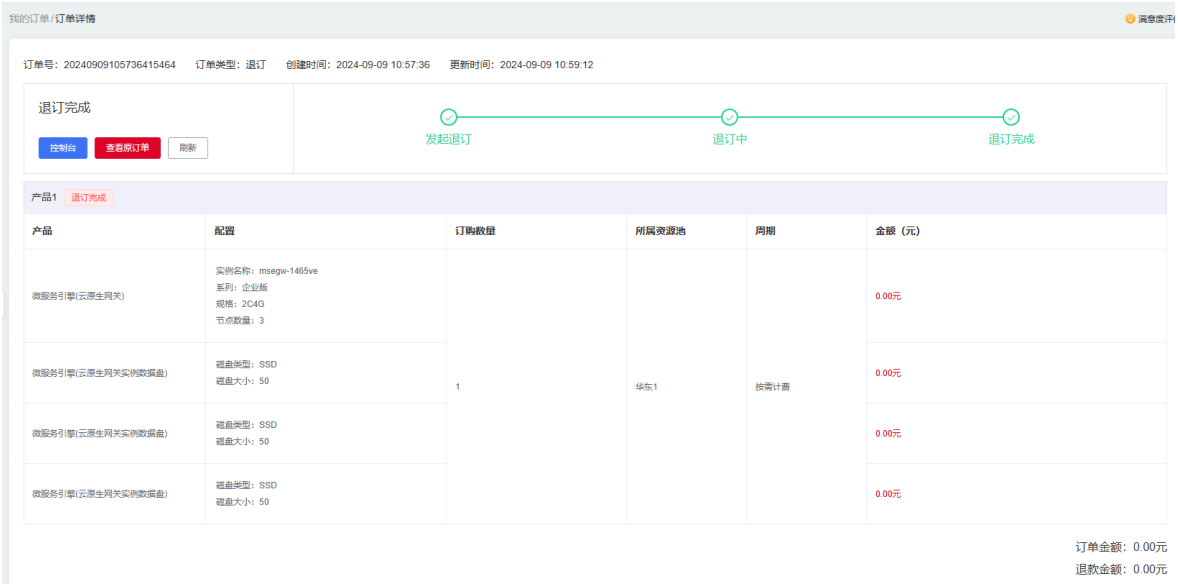
产品退订相关规则详细参见：退订规则说明。

退订操作

以云原生网关子产品为例，进入微服务引擎控制台，在左侧导航栏中单击云原生网关-网关列表，在列表中针对某个实例点击退订按钮，在您确认退订后，单击确定按钮。



退订后，可在个人中心查看退订订单状态及金额情况。



注意

退订成功后，不可登录到控制台或者通过OpenAPI进行应用变更等操作。

欠费与续费说明

续订规则

产品续订规则详细参见：续订规则说明。

按需计费

当您的账户余额不足以支付账单金额，且微服务引擎MSE服务处于欠费状态时，系统将限制您登录控制台且无法通过API进行应用的变更等操作。

在欠费期间，微服务引擎MSE不会对您的应用实例及数据做任何变动，但会持续进行计费。

您可以通过账号充值的方式，恢复正常使用。

注意

云公司将保留该实例资源、继续存储客户的数据十五（15）日（即自操作权限被暂停之日的暂停开始时刻至第十五（15）日相同时刻为期限届满）；如前述十五（15）日期间届满仍未充值、缴纳足额服务费用，云公司有权在前述期间届满时立即释放客户的实例资源，并删除实例数据。

包年包月计费

当您微服务引擎MSE的包年包月订单即将到期，且未选择自动续订时，您可以进行手工续订操作。

注意

服务期限届满后，云公司将保留该实例资源、继续存储客户的数据十五（15）日（即自操作权限被暂停之日的暂停开始时刻至第十五（15）日相同时刻为期限届满）；如前述十五（15）日期间届满仍未续订和续费，云公司有权在前述期间届满时立即释放客户的实例资源，并删除实例数据。

以云原生网关为例

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 -> 网关列表 ；
4. 在实例列表点击续订；

5. 进入实例续订页，选择续订时长 ，提交订单，完成续费；


天翼云
控制中心
实例续订
温馨提醒：您即将进行续订操作，提交后将产生支付订单，请在48小时内完成支付，否则操作失败。
实例 ID
fa6432042084494db2539d778095cecc
实例名称
mse-rg5te3
实例规格
2核 4GB | 节点数量1个
版本类型
基础版
运行状态
正常
开通时间
2023-12-22 23:27:48
过期时间
2024-01-22 23:35:22
续订时长
1个月
2个月
3个月
4个月
5个月
6个月
1年
新过期时间
2024-02-22 23:35:22
费用

注册配置中心

注册配置中心快速接入示例

概述

注册配置中心提供多种类型的引擎，您可以按需创建满足您需求的服务实例，不需要部署和维护这些基础微服务组件，开发者可以完全专注微服务应用本身的开发。

以下步骤将演示创建一个注册配置中心实例，并改造一个微服务快速接入您创建的注册配置中心。演示中开通的引擎类型为Nacos。

创建一个注册配置中心实例

1. 首先，从天翼云官网控制中心> 微服务工具与平台 > 微服务引擎MSE，点击进入产品页面。
2. 左上角选择目标资源，这里以华东1为例，进入微服务引擎控制台。
3. 左侧菜单栏点击注册配置中心> 实例列表，进入注册配置中心控制台实例列表页面。
4. 点击实例列表左上角的创建实例跳转至产品订购页面。
5. 选择系列/引擎类型，实例规格，选择节点数量，选择子网、VPC和安全组后，点击确认订单。不同资源池节点订购方式有所不同，以具体功能页面为准。
6. 点击下一步、跳转至订单确认页面，点击确认提交订单，支付完成后等待5~10分钟，可以在控制台页面看到开通成功的实例。

进入实例的管理页面

- 实例开通完成后，进入实例列表页面，点击实例ID或者实例名称，即可进入实例的管理页面。
- 获取实例的内网连接地址，该地址就是您在VPC内部连接该Nacos引擎的访问地址。
- 另外基础信息还包括规格、磁盘、网络以及节点状态等一系列信息。

SpringCloud应用接入Nacos配置中心

将Spring Cloud应用快速接入Nacos配置中心，集成Nacos配置管理功能，可遵循如下步骤。

前提条件

- 下载Maven并设置Maven环境变量。
- 已有Spring Cloud应用。
- 已创建MSE Nacos引擎。

在控制台创建配置

1. 进入MSE注册配置中心控制台，点击实例ID进入您创建的Nacos实例，点击配置管理->配置列表，选定命名空间如prod，点击创建配置。填写配置的Data ID、Group和配置内容，示例如下：

Data ID：nacos-provider
Group：test-group
配置内容：config1=abc

2. 选定配置格式为Properties，点击发布。
3. 接下来为prod命名空间建立读写权限用户。

4. 点击权限控制->用户管理，点击创建用户，填写您的用户名密码，点击确认。
5. 点击权限控制->角色管理，点击添加角色，填写角色名为PROD_ADMIN，用户名下拉框选中您刚才所创建的用户，点击确认。
6. 点击权限控制->权限管理，点击添加权限，角色名选择PROD_ADMIN，命名空间选择prod，动作选择“读写”，点击确认。

客户端接入配置中心

在您的Spring Cloud应用中引入如下依赖，替换其中¥{spring-cloud-starter.version}为与当前应用SpringBoot、SpringCloud版本适配的spring-cloud-starter-alibaba-nacos-config版本，替换其中{nacos-client.version}为当前开源nacos-client的最新稳定版，推荐替换为版本2.3.2。

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
  <version>${spring-cloud-starter.version}</version>
  <exclusions>
    <exclusion>
      <groupId>com.alibaba.nacos</groupId>
      <artifactId>nacos-client</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>com.alibaba.nacos</groupId>
  <artifactId>nacos-client</artifactId>
  <version>${nacos-client.version}</version>
</dependency>
```

在应用配置bootstrap.properties（或yml）中添加Nacos相关参数，其中{your_nacos_addr}填写ip:port格式的Nacos服务端节点地址，多个节点用英文逗号分隔。

```
spring.application.name=nacos-provider

spring.cloud.nacos.config.server-addr={your_nacos_addr}
spring.cloud.nacos.config.namespace=prod
spring.cloud.nacos.config.username={your_nacos_username}
spring.cloud.nacos.config.password={your_nacos_password}
```

注意

若您的SpringBoot版本大于等于2.4，还需要额外添加依赖spring-cloud-starter-bootstrap以使能bootstrap.properties：

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-bootstrap</artifactId>
</dependency>
```

在应用代码中添加如下Controller，通过访问/getConfig1路径来获取config1动态配置：

```
@RestController
@RefreshScope
public class ProviderController {

   @Value("${config1:default}")
   private String config1;

   @GetMapping("/getConfig1")
   public String getConfig1() {
      log.info("/getConfig1");
      return config1;
   }

}
```

启动应用。应用启动成功后，会自动连接到Nacos配置中心。

感知配置变更

在应用所部署机器使用如下curl指令进行访问：

```
curl 'localhost:{your_demo_port}/getConfig1'
```

可获取含有config1配置内容的返回信息。在未连接Nacos配置中心的情况下，此时将返回default；成功连接Nacos配置中心的情况下，此时将返回abc。

再次进入MSE注册配置中心控制台，点击实例ID进入您创建的Nacos实例，点击配置管理->配置列表，选定prod命名空间，点击nacos-provider配置右侧的“编辑配置”，修改配置内容为如下值：

```
config1=def
```

再次调用上述curl指令，此时将返回def，证明Nacos配置已成功推送到客户端。

SpringCloud应用接入Nacos注册中心

将Spring Cloud应用快速接入Nacos注册中心，集成Nacos服务注册发现功能，可遵循如下步骤。

前提条件

- 下载Maven并设置Maven环境变量。
- 已有Spring Cloud应用。
- 已创建MSE Nacos引擎。

在控制台创建用户

- 进入MSE注册配置中心控制台，点击实例ID进入您创建的Nacos实例，为prod命名空间建立读写权限用户。
- 点击权限控制->用户管理，点击创建用户，填写您的用户名密码，点击确认。
- 点击权限控制->角色管理，点击添加角色，填写角色名为PROD_ADMIN，用户名下拉框选中您刚才所创建的用户，点击确认。

- 点击权限控制->权限管理，点击添加权限，角色名选择PROD_ADMIN，命名空间选择prod，动作选择“读写”，点击确认。

服务提供者接入注册中心

在您的Spring Cloud应用中引入如下依赖，替换其中{spring-cloud-starter.version}为与当前应用SpringBoot、SpringCloud版本适配的spring-cloud-starter-alibaba-nacos-discovery版本，替换其中{nacos-client.version}为当前开源nacos-client的最新稳定版，推荐替换为版本2.3.2。

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
  <version>${spring-cloud-starter.version}</version>
  <exclusions>
    <exclusion>
      <groupId>com.alibaba.nacos</groupId>
      <artifactId>nacos-client</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>com.alibaba.nacos</groupId>
  <artifactId>nacos-client</artifactId>
  <version>${nacos-client.version}</version>
</dependency>
```

在应用配置application.properties（或yml）中添加Nacos相关参数，其中{your_nacos_addr}填写ip:port格式的Nacos服务端节点地址，多个节点用英文逗号分隔。

```
spring.application.name=nacos-provider

spring.cloud.nacos.discovery.server-addr={your_nacos_addr}
spring.cloud.nacos.discovery.namespace=prod
spring.cloud.nacos.discovery.username={your_nacos_username}
spring.cloud.nacos.discovery.password={your_nacos_password}
```

注意

此处配置参数namespace需要填写命名空间ID

在应用代码中添加如下Controller，暴露服务提供者的服务路径/getCarInfo：

```
@RestController
public class ProviderController {

  @GetMapping("/getCarInfo")
  public String getCarInfo() {
    return "Benz";
  }
```

}

启动应用。应用启动成功后，会自动连接到Nacos注册中心。

进入MSE注册配置中心控制台，点击实例ID进入您创建的Nacos实例，点击服务管理->服务列表，选定prod命名空间，可以看到您所注册的nacos-provider服务，其下存在一个健康状态实例。

服务消费者接入注册中心

在您的Spring Cloud应用中引入如下依赖，替换其中{spring-cloud-starter.version}为与当前应用SpringBoot、SpringCloud版本适配的spring-cloud-starter-alibaba-nacos-discovery版本，替换其中{nacos-client.version}为当前开源nacos-client的最新稳定版，推荐替换为版本2.3.2。

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
  <version>${spring-cloud-starter.version}</version>
  <exclusions>
    <exclusion>
      <groupId>com.alibaba.nacos</groupId>
      <artifactId>nacos-client</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>com.alibaba.nacos</groupId>
  <artifactId>nacos-client</artifactId>
  <version>${nacos-client.version}</version>
</dependency>
```

在应用配置application.properties（或yml）中添加Nacos相关参数，其中{your_nacos_addr}填写ip:port格式的Nacos服务端节点地址，多个节点用英文逗号分隔。

```
spring.application.name=nacos-consumer

spring.cloud.nacos.discovery.server-addr={your_nacos_addr}
spring.cloud.nacos.discovery.namespace=prod
spring.cloud.nacos.discovery.username={your_nacos_username}
spring.cloud.nacos.discovery.password={your_nacos_password}
```

注意

此处配置参数namespace需要填写命名空间ID

在应用中添加如下代码，定义指向nacos-provider服务的FeignClient，并暴露/getCarInfo路径，内部逻辑调用nacos-provider服务的/getCarInfo路径：

```
@FeignClient(value = "nacos-provider")
public interface NacosProviderFeignClient {
```

```
    @GetMapping("/getCarInfo")
    String getCarInfo();

}

@RestController
public class ConsumerController {

    @Autowired
    private NacosProviderFeignClient nacosProviderFeignClient;

    @GetMapping("/getCarInfo")
    public String getPersonInfo() {
        return nacosProviderFeignClient.getCarInfo();
    }

}
```

启动应用。应用启动成功后，会自动连接到Nacos注册中心。

进入MSE注册配置中心控制台，点击实例ID进入您创建的Nacos实例，点击服务管理->服务列表，选定prod命名空间，可以看到您所注册的nacos-consumer服务，其下存在一个健康状态实例。

确保上述部署的服务提供者和服务消费者的网络连通，在服务消费者应用所部署机器使用如下curl指令进行访问：

```
curl 'localhost:{your_demo_port}/getCarInfo'
```

此时nacos-consumer会自动到Nacos服务端获取nacos-provider的地址进行访问，完成服务发现。返回“Benz”则证明服务发现和调用成功。

微服务注册配置中心使用说明

概述

为了您更好的使用注册配置中心，本文将介绍实例创建时的注意事项、Spring Cloud应用和Dubbo应用开发时需要添加的服务注册中心配置等内容。

创建实例

创建实例的过程，详见本文档的创建Nacos实例、创建Zookeeper引擎和创建Eureka引擎相关章节。

注册中心使用说明

Nacos

SpringCloud应用使用MSE的Nacos注册中心

增加依赖项，以maven为例：

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
  <version>${spring-cloud-starter-alibaba.version}</version>
</dependency>
```

增加配置项：

```
spring:
 cloud:
  nacos:
   discovery:
    server-addr: ${_NACOS_SERVER_ADDRESS}
    namespace: ${_NACOS_NAMING_NAMESPACE_ID}
    username: ${_NACOS_SERVER_USERNAME}
    password: ${_NACOS_SERVER_PASSWORD}
```

注意

此处客户端配置中namespace需要填写命名空间ID

Dubbo应用使用MSE的Nacos注册中心

增加依赖项，以maven依赖为例：

```
<!--dubbo相关-->
<dependency>
  <groupId>org.apache.dubbo</groupId>
  <artifactId>dubbo-spring-boot-starter</artifactId>
  <version>${dubbo.version}</version>
</dependency>
<dependency>
  <groupId>org.apache.dubbo</groupId>
  <artifactId>dubbo-registry-nacos</artifactId>
  <version>${dubbo.version}</version>
</dependency>
```

通过application.yml 增加配置：

```
dubbo:
 registry:
  address: nacos://Nacos访问地址
  application:
```

```
  name: 服务名称
 protocol:
  name: dubbo
  port: 22035
 scan:
  base-packages: 扫描的包路径
 provider:
  timeout: 30000
```

Zookeeper

Dubbo应用接入 ZooKeeper注册中心

在您需要注册到Zookeeper的Dubbo服务中，引入以下Maven坐标：

```
<!--dubbo zk-->
<dependency>
 <groupId>org.apache.dubbo</groupId>
 <artifactId>dubbo-dependencies-zookeeper</artifactId>
 <version>2.7.13</version>
 <type>pom</type>
</dependency>
```

然后，在启动配置application.properties中加入以下配置：

```
dubbo.registry.address=zookeeper://{ip}:$${port}?timeout=60000
```

Spring Cloud应用接入 ZooKeeper注册中心

SpringCloud应用配置中填写如下配置

```
spring:
 application:
  name: springboot-demo-vl
 cloud:
  zookeeper:
   connect-string: 192.168.160.62:47588
   discovery:
   enabled: true
```

Connect-string 是ZooKeeper集群的地址和端口，spring.application.name是服务注册的名称，也是节点的名称。

Eureka

Spring Cloud应用接入 Eureka注册中心

相关依赖导入完毕后，在您的Spring Cloud应用配置中加入如下配置即可：

```
spring.application.name=eureka-provider
server.port=xxxx
```

```
#指定向eureka注册的本机ip地址（多网卡的情况下，最好手工指定）
#eureka.instance.ip-address=10.x.x.x
eureka.instance.prefer-ip-address=true
#需要注册到的eureka服务端的地址，多个节点的地址使用英文分割
eureka.client.service-url.defaultZone=http://${ip}:${port}/eureka
#eureka客户端相关配置
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
eureka.client.should-unregister-on-shutdown=true
```

创建Nacos实例

概述

Nacos引擎包括配置中心和注册中心，提供动态配置、服务发现和服务健康监测等简单易用的特性，经过实践检验具备高性能和高可用性，帮助用户管理配置、注册和发现微服务，更加快捷方便地构建、交付和管理微服务平台。

为了方便您开通实例，从天翼云官网控制中心> 微服务工具与平台 > 微服务引擎MSE，点击进入产品页面，点击左侧菜单栏的注册配置中心>实例列表，进入注册配置中心控制台。本文将介绍如何创建一个Nacos引擎实例。

操作步骤

1. 进入微服务引擎控制台，并在左上方选中您需要开通产品的资源池。点击“注册配置中心”页签，进入“实例列表”页签；
2. 点击页面左上方的“创建实例”按钮，进入实例开通页面；
3. 选择开通配置，配置项详细说明如下：

- 计费模式：目前可选择“包年包月”或“按需付费”，具体计费模式和规则可点击右侧的“查看产品定价”查看；
- 系列：目前可选择“单机版”或“集群版”，推荐您开通集群版，以保障实例的高可用性；
- 引擎类型：若您需要开通Nacos引擎，则选中“Nacos”；
- 实例名称：输入该实例的名称，实例名称是帮助您区分不同实例的重要标识；
- 实例规格：可选择将实例搭载在指定CPU、内存规格的机器上；
- 节点数量：可选择开通实例的节点数，节点数越大，实例越高可用；
- 部署模式和可用区：可选择将多节点实例部署在不同可用区，进一步提高可用性；
- 虚拟私有云：选择将实例开通在指定的虚拟私有云（VPC）下；
- 所在子网：选择将实例开通在该虚拟私有云指定的子网下；
- 安全组：为该实例绑定指定的安全组规则；
- 购买时长和自动续期：若您选择“包年包月”，则需要选择您购买的周期，并选中到期时是否自动续期；

4. 配置确认后，点击右下方的“下一步”按钮，进入开通配置确认页面，确认无误后，点击“提交订单”，进入订单结算页面。完成支付后，等待5至10分钟，即可完成开通；
5. 开通完成后，您可以在微服务引擎控制台的“注册配置中心”页签下找到您开通的实例。

创建ZooKeeper引擎

概述

微服务注册中心ZooKeeper是一个分布式应用程序协调服务，是 Google Chubby 的开源实现。利用ZooKeeper的存储配置，实现配置信息的集中式管理和数据的动态更新，保证数据一致性。MSE提供的ZooKeeper企业级服务，分为单机版和集群版两种，前者适用于开发测试，后者致力于在性能、可观测和高可用方面做了诸多提升，用于生产环境。

本文将介绍如何创建一个ZooKeeper引擎实例。

操作步骤

1. 进入微服务引擎控制台，并在左上方选中您需要开通产品的资源池。点击“注册配置中心”页签，进入“实例列表”页签；
2. 点击页面左上方的“创建实例”按钮，进入实例开通页面；
3. 选择开通配置，配置项详细说明如下：

- 计费模式：目前可选择“包年包月”或“按需付费”，具体计费模式和规则可点击右侧的“查看产品定价”查看；
- 系列：目前可选择“单机版”或“集群版”，推荐您开通集群版，以保障实例的高可用性；
- 引擎类型：若您需要开通ZooKeeper引擎，则选中“ZooKeeper”；
- 实例名称：输入该实例的名称，实例名称是帮助您区分不同实例的重要标识；
- 实例规格：可选择将实例搭载在指定CPU、内存规格的机器上；
- 节点数量：可选择开通实例的节点数，节点数越大，实例越高可用；
- 部署模式和可用区：可选择将多节点实例部署在不同可用区，进一步提高可用性；
- 虚拟私有云：选择将实例开通在指定的虚拟私有云（VPC）下；
- 所在子网：选择将实例开通在该虚拟私有云指定的子网下；
- 安全组：为该实例绑定指定的安全组规则；
- 购买时长和自动续期：若您选择“包年包月”，则需要选择您购买的周期，并选中到期时是否自动续期；

4. 配置确认后，点击右下方的“下一步”按钮，进入开通配置确认页面，确认无误后，点击“提交订单”，进入订单结算页面。完成支付后，等待5至10分钟，即可完成开通；
5. 开通完成后，您可以在微服务引擎控制台的“注册配置中心”页签下找到您开通的实例。

创建Eureka引擎

概述

Eureka是一个Java体系的服务发现组件，用于实现服务的自动注册与发现即客户端完成微服务项Eureka注册，采用心跳续约策略，而服务端作为注册中心，通过接受心跳保持服务健康状态。本文将介绍如何创建一个Eureka引擎实例。

操作步骤

1. 进入微服务引擎控制台，并在左上方选中您需要开通产品的资源池。点击“注册配置中心”页签，进入“实例列表”页签；
2. 点击页面左上方的“创建实例”按钮，进入实例开通页面；
3. 选择开通配置，配置项详细说明如下：

- 计费模式：目前可选择“包年包月”或“按需付费”，具体计费模式和规则可点击右侧的“查看产品定价”查看；
- 系列：目前可选择“单机版”或“集群版”，推荐您开通集群版，以保障实例的高可用性；
- 引擎类型：若您需要开通Eureka引擎，则选中“Eureka”；
- 实例名称：输入该实例的名称，实例名称是帮助您区分不同实例的重要标识；
- 实例规格：可选择将实例搭载在指定CPU、内存规格的机器上；
- 节点数量：可选择开通实例的节点数，节点数越大，实例越高可用；
- 部署模式和可用区：可选择将多节点实例部署在不同可用区，进一步提高可用性；
- 虚拟私有云：选择将实例开通在指定的虚拟私有云（VPC）下；
- 所在子网：选择将实例开通在该虚拟私有云指定的子网下；
- 安全组：为该实例绑定指定的安全组规则；
- 购买时长和自动续期：若您选择“包年包月”，则需要选择您购买的周期，并选中到期时是否自动续期；

4. 配置确认后，点击右下方的“下一步”按钮，进入开通配置确认页面，确认无误后，点击“提交订单”，进入订单结算页面。完成支付后，等待5至10分钟，即可完成开通；
5. 开通完成后，您可以在微服务引擎控制台的“注册配置中心”页签下找到您开通的实例。

云原生网关

云原生网关最简应用

概述

云原生网关作为微服务架构流量的入口支持对接天翼云注册配置中心、云容器引擎，并通过ELB实现外部访问暴露。同时云原生网关也可以不搭配以上组件独立使用，本文介绍云原生网关最简使用案例，架构如下：

体验流程：创建网关实例 **>** 添加固定地址服务 **>** 添加路由 **>** 验证访问

创建网关实例

有两个入口可以进入云原生网关开通页面：

1. 从天翼云官网控制中心**>** 微服务工具与平台 **>** 微服务引擎MSE，点击订购图标，进入产品订购页面。
2. 从云原生网关控制台实例列表页面的新建实例入口进入网关实例开通页面，选择实例规格、节点数及网络相关配置，确认提交；云原生网关订购配置说明如下

参数	描述
计费模式	支持包年包月和按需计费方式，费用说明请参照计费说明。
购买时长	可以根据实际需求进行选择，支持1个月、2个月、3个月、4个月、5个月、6个月、1年。
自动续期	您可以选择开启自动续期，避免云原生网关到期后无法使用。
自动续期购买时长	当开启自动续期，可以选择续期时长，支持1个月、2个月、3个月、4个月、5个月、6个月、1年。

参数	描述
系列	支持基础版和集群版。基础版：只支持单可用区部署，且节点数量为1。集群版：自动将控制节点以及工作节点平均分配部署至各可用区，且节点数量不少于2。
实例规格	架构支持X86（通用、海光）、ARM（通用、鲲鹏、飞腾），具体以当前资源池提供的选项为准。实例规格支持2C4G 、4C8G 、8C16G 、16C32G规格，规格支撑能力说明请参照产品规格。
数据盘	支持超高IO ，最低大小50G，可根据实际需求自定义填写，填写值必须为50的倍数。
节点数量	基础版固定1个节点，无需填写；集群版您可以根据实际需求填写节点数量，节点数量不少于2。
部署方式	用户无需修改，基础版固定选中单可用区部署；集群版时，如果当前资源池支持多可用区且节点数量大于2时，则默认为多可用区部署，单可用区部署置灰，否则为单可用区部署。
可用区	基础版需要选择任意一个可用区进行部署；集群版会自动将控制节点以及工作节点平均分配部署至各可用区。
虚拟私有云	选择虚拟私有云，若您还没有虚拟私有云，请参照创建虚拟私有云。
所在子网	选择所在子网，若您还没有所在子网，请参照创建所在子网。
安全组	选择安全组，若您还没有可用安全组，请参照创建安全组。
实例名称	自定义实例名称，最长40字符，只能包含小写字母、数字及分隔符(-)，且必须以小写字母开头，数字或小写字母结尾。
企业项目	网关实例关联的企业项目，可以到IAM控制台创建企业项目。
启用监控指标	启用后，可在控制台观测分析中查看系统和API的流量、成功率、延迟等监控指标，若您还没有开通应用性能监控产品，可先点击提示链接前往开通。
启用链路追踪	可选择采集百分比启用，启用后，可在控制台观测分析中查看API请求的链路追踪信息，若您还没有开通应用性能监控产品，可先点击提示链接前往开通。
启用访问日志采集	启用后，可在控制台观测分析中查看访问日志，若您还没有开通云日志服务，可先点击提示链接前往开通。

网关实例创建可能需要5分钟左右时间，请耐心等待。

添加服务

首先部署好后端服务，进入实例详情页，选择服务列表菜单，点击创建服务，选择固定地址服务，填入后端服务的ip和端口，如下图所示：

添加路由

进入实例详情页路由配置菜单，选择创建路由，根据路由的规则填写表单，比如这里创建路由名称为test，匹配路径为/test，请求的后端服务为刚才创建的固定地址服务testsvc，如图所示：

验证访问

由于没有绑定公网ELB，我们可以在网关同VPC内进行测试，我们使用curl命令验证可以看到返回预期内的结果。

```
[root@msegw-vmpojohwww apisix]# curl http://[redacted]:27151/test -sv
*   Trying [redacted]:27151...
* Connected to [redacted] port 27151 (#0)
> GET /test HTTP/1.1
> Host: [redacted]:27151
> User-Agent: curl/7.71.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 404 Not Found
< Content-Type: text/html; charset=utf-8
< Content-Length: 225
< Connection: keep-alive
< Date: Mon, 18 Mar 2024 06:32:17 GMT
```

使用云容器引擎接入云原生网关

概述

云原生网关体验流程如下：

创建网关实例 > 绑定ELB > 添加服务来源 > 添加服务 > 配置路由规则 > 测试验证 > 查看监控

前置条件

1. 已创建云原生网关实例；
2. 已创建ELB实例；
3. 已开通天翼云日志服务（LTS）和应用性能监控服务（APM），网关实例开启了指标监控、链路追踪和日志服务；

绑定ELB

实例开通成功后，需要绑定到ELB提供外部访问；当前支持私网ELB和公网ELB；可以到ELB开通页面在网关同VPC下创建ELB实例；

从网关列表页选择指定网关进入网关详情页，在基础信息页可以看到网关入口选项，选择绑定ELB，在下拉列表中选择已经开通的公网或者私网ELB实例及端口，绑定即可。

绑定完成后可以看到当前绑定的ELB列表及访问地址。

添加服务来源

通过网关列表页面选择网关实例进入对应实例的详情页，在服务来源子菜单下可以添加云原生网关的服务来源，当前支持将与网关同vpc下的nacos实例和云容器引擎集群作为服务来源

添加服务

在网关实例服务列表菜单下可以创建服务，当前支持从容器、Nacos服务来源创建服务，或者创建固定地址的服务；

选择从容器创建服务时需要指定：

（1）服务所在的命名空间；

（2）在服务列表栏可以看到选择的命名空间下的服务信息（K8s Service），每个服务可能有多个端口，在云原生网关中，每个端口都可以创建为一个服务；

（3）根据需求可以配置后端服务的请求协议（HTTP、HTTPS、GRPC、GRPCS）、websocket选项、MTLS选项等；

配置路由规则

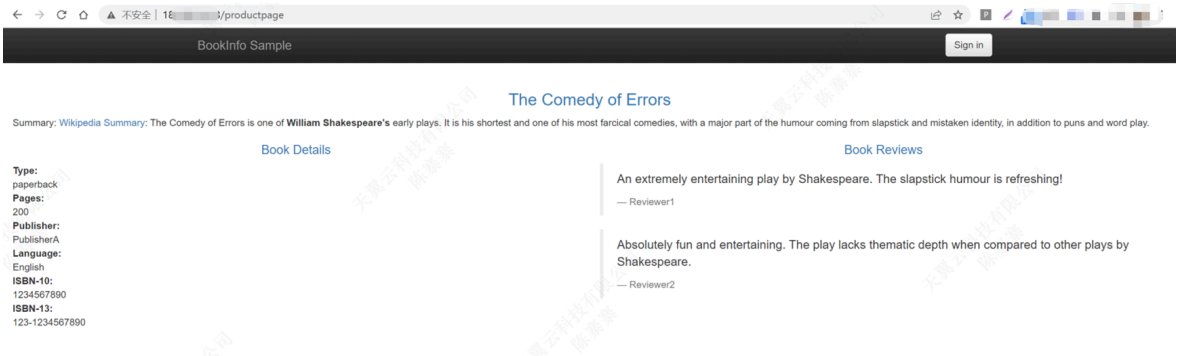
在路由配置菜单左上角进入路由配置页面，路由匹配规则之间是“与”的关系，必须全部满足才算匹配；核心的路由配置项说明如下：

配置项	说明
名称	路由名称，用于标识一条路由规则。
域名	用于和请求中的域名进行匹配，不填则任何请求都可以匹配。
路径	匹配请求的path（不含query参数），当前支持前缀匹配和精确匹配。
方法	匹配请求中的HTTP方法。
优先级	当多个路由同时匹配一个请求时，路径匹配深度较大的路由优先；路径匹配相同的情况下，路由优先级高（数字大）的优先匹配。
请求header	匹配请求中的HTTP header。
请求query	匹配请求中的HTTP query参数。
目标服务	当前支持单服务、多服务、标签路由、mock路由和重定向。

在路由的目标服务选项中选择刚才创建好的后端服务即可。

测试验证

通过绑定的ELB公网地址访问，结果符合预期。



查看监控

调用链

在观测分析菜单下链路追踪子菜单下可以根据接口路径查询到链路追踪信息（需要确保您的网关实例已开启链路追踪，并且采样率大于零才可以采集到链路追踪数据）。

指标监控

在监控分析子菜单可以看到业务监控信息，当前支持的指标如下：

指标	说明
入流量	请求进入网关的带宽。
出流量	网关应答的带宽。
配置中心连接状态	网关和控制面连接状态，1为正常，0位异常。
请求成功率	网关返回HTTP 2XX的比例。
404比例	网关返回HTTP 404的比例。
5XX比例	网关返回HTTP 5XX的比例。
失败率	网关返回HTTP 4XX和5XX的比例。
平均延迟	网关收到请求到返回应答的平均时延（ms）。
P50延迟	网关处理请求50分位耗时。
P95延迟	网关处理请求95分位耗时。
P99延迟	网关处理请求99分位耗时。
QPS	网关每秒处理请求数。
连接数	网关连接数统计。

访问日志

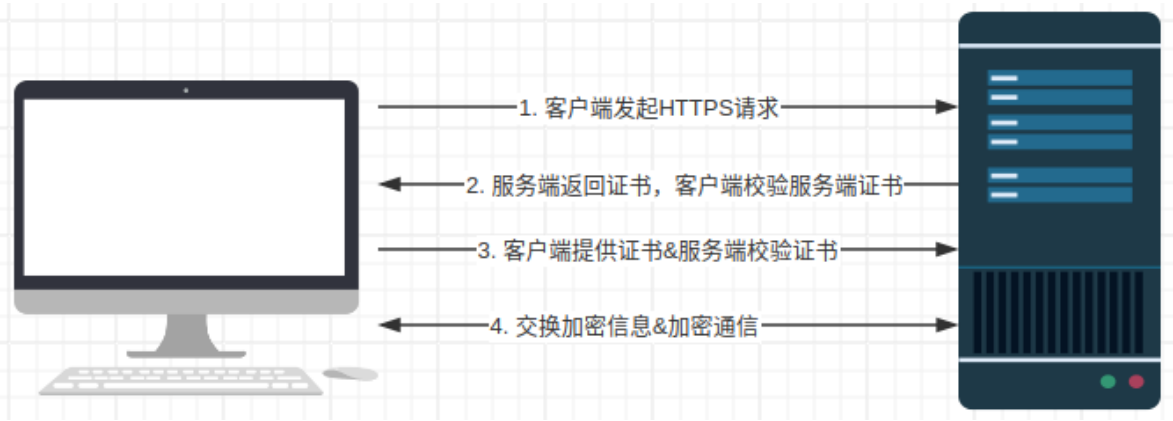
日志中心记录了网关的访问日志，当前网关访问日志记录的字段有：

字段	说明
server	请求访问到的网关实例信息。
request	请求信息，主要包括请求method、uri、header等。
response	应答信息，包括状态码、header等。
start_time	请求开始时间戳。
client_ip	请求客户端ip。
latency	网关处理请求总时延。
upstream_latency	上游应答耗时。
route_id	请求匹配到的路由id。
apisix_latency	网关自身处理耗时。
service_id	后端服务id。
upstream	上游服务地址

使用云原生网关实现后端双向TLS认证

概述

云原生网关作为代理服务接收下游服务（DownStream）请求，并转发到上游服务（Upstream）；对于上游服务来说，云原生网关收敛了外部的请求，统一经过网关转发到上游。从安全角度考虑，上游服务可以添加认证鉴权规则实现对网关的身份认证和访问鉴权，确保上游服务安全性；其中一种实现方式就是在上游服务配置TLS策略，要求网关转发请求时提供客户端证书，实现对网关的身份认证，具体流程如下：



云原生网关支持配置证书，用于实现请求后端服务时服务端对网关的证书认证，本文说明具体配置流程。

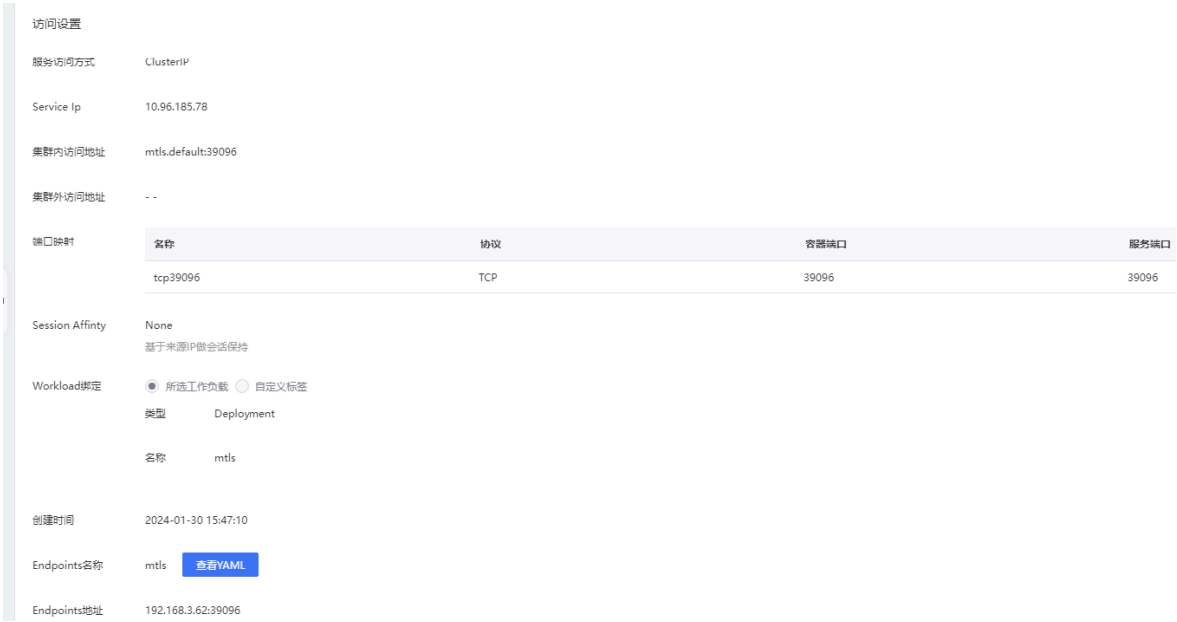
前提条件

1. 部署后端HTTPS服务，并配置要求请求客户端提供证书；
2. 已开通云原生网关实例；

部署后端服务

我们采用云容器引擎部署后端服务，镜像采用我们的demo应用（已配置证书，并要求客户端请求时提供证书），部署完成后在云容器引擎控制台可以看到容器启动：

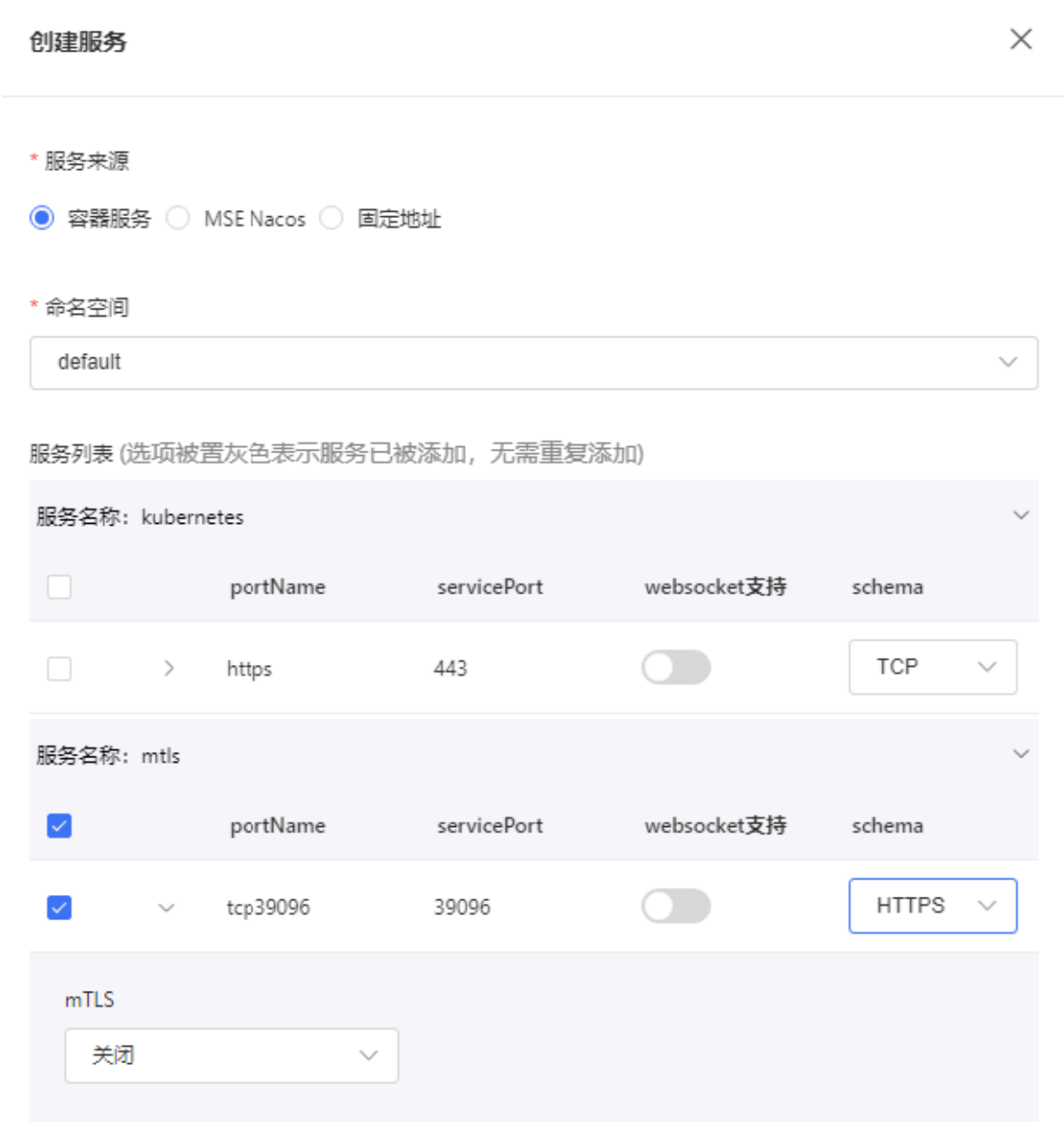
还需要部署关联到工作负载的Service，用于暴露工作负载，如下：



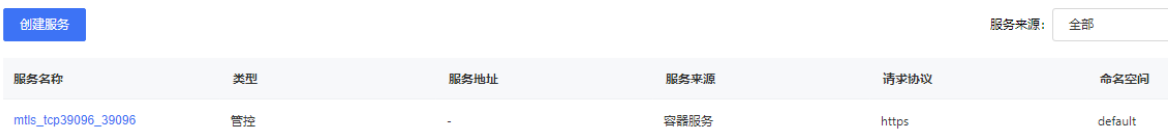
使用云原生网关配置路由转发

首先进入云原生网关控制台服务来源菜单下，添加云容器引擎服务来源，选择我们部署了后端服务的云容器引擎集群，添加完成后如下所示：

进入 服务列表-> 创建服务 功能，选择从容器创建后端服务，选择我们部署的命名空间和服务，服务协议选择HTTPS（暂时先关闭mTLS选项），如下：



创建完成后，服务如下：



进入 路由配置-> 创建路由 页面，配置路由转发规则（匹配路径/api/1/reviews，请求转发到上面创建的服务）如下：


创建路由
* 匹配路径（Path）
/api/1/reviews
方法（Method）
Method匹配值，可多选，不填则匹配所有的HTTP方法
优先级
如果不同路由包含相同uri,值越大优先级越高,路由将被优先匹配,默认值为0
请求头（Header）
+ 添加请求头
请求参数（Query）
+ 添加请求参数
Cookie
+ 添加Cookie参数
是否启用参数规整化匹配
目标服务
单服务
多服务
标签路由
Mock
重定向
mtls_tcp39096_39096

结果验证

通过云原生网关访问接口http://192.168.4.96:27151/api/1/reviews

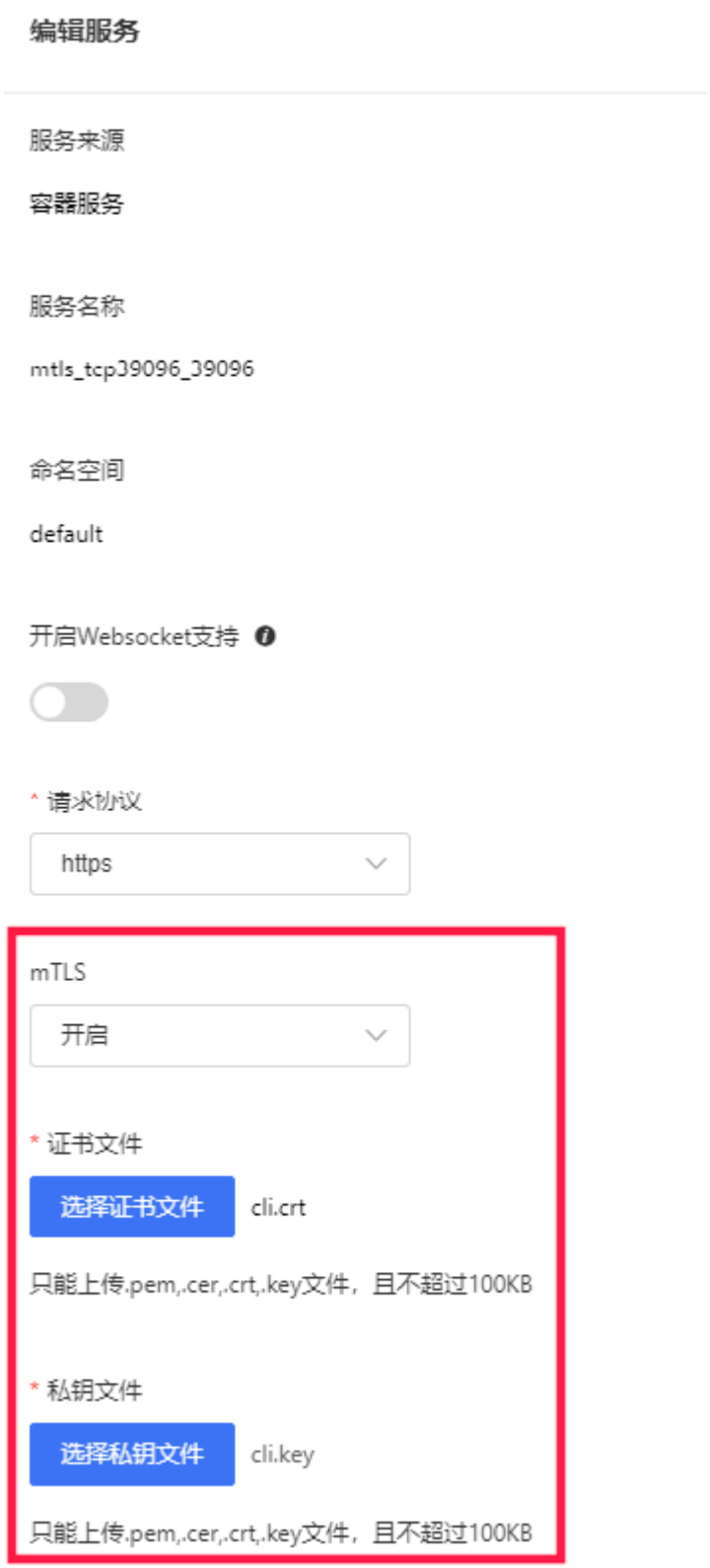
```
curl http://192.168.4.96:27151/api/1/reviews -sv
*   Trying 192.168.4.96:27151...
* Connected to 192.168.4.96 (192.168.4.96) port 27151 (#0)
> GET /api/1/reviews HTTP/1.1
> Host: 192.168.4.96:27151
> User-Agent: curl/7.71.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 502 Bad Gateway
< Date: Tue, 30 Jan 2024 08:07:10 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 154
< Connection: keep-alive
<
<html>
<head><title>502 Bad Gateway</title></head>
<body>
<center><h1>502 Bad Gateway</h1></center>
<hr><center>openresty</center>
</body>
</html>
* Connection #0 to host 192.168.4.96 left intact
```

可以看到请求报错了，后台查看debug日志可以看到错误信息如下（SSL握手报错）：

```
sslv3 alert bad certificate:SSL alert number 42) while SSL handshaking to upstream
```

mTLS证书配置&验证

进入服务列表菜单，编辑刚才创建的服务，开启服务mTLS认证并上传相关证书，如下：


编辑服务
服务来源
容器服务
服务名称
mtls_tcp39096_39096
命名空间
default
开启Websocket支持
* 请求协议
https
mTLS
开启
* 证书文件
选择证书文件
cli.crt
只能上传.pem,.cer,.crt,.key文件，且不超过100KB
* 私钥文件
选择私钥文件
cli.key
只能上传.pem,.cer,.crt,.key文件，且不超过100KB

保存后再次请求接口，可以看到返回了正确的结果：

```
curl http://192.168.4.96:27151/api/1/reviews -sv
*   Trying 192.168.4.96:27151...
* Connected to 192.168.4.96 (192.168.4.96) port 27151 (#0)
> GET /api/1/reviews HTTP/1.1
> Host: 192.168.4.96:27151
> User-Agent: curl/7.71.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Date: Tue, 30 Jan 2024 08:13:57 GMT
<
* Connection #0 to host 192.168.4.96 left intact
[{"id":1,"productId":1,"reviewer":"Reviewer1","text":"This is the 1st reviewer"}]
```

使用云原生网关实现后端访问签名

概述

后端访问签名功能支持您在后端服务中开启身份认证后，填写设定的一对Access Key和Secret Key，并根据选择的认证算法和加密算法，以及加密Headers列表，计算出相应的签名信息。当通过网关请求该服务时，会将该服务的签名信息设置在Header请求头中，从而后端服务对比网关的签名和服务器端计算的签名是否一致进行身份验证与鉴权。

前提

1. 已开通云原生网关实例
2. 已部署能够进行验证Header签名信息的后端服务

网关中实现后端访问签名

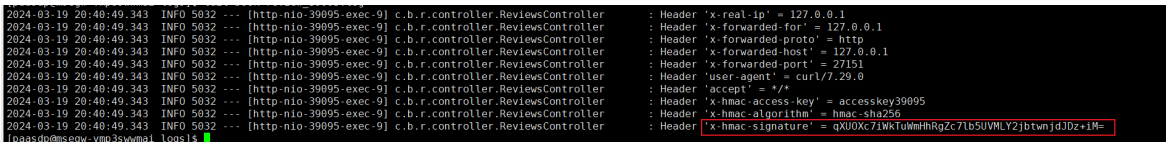
1. 部署服务，可选择容器部署、Nacos注册或固定地址。
2. 进入服务详情。
3. 后端访问签名栏点击编辑。
4. 打开 开启身份认证开关。
5. 选择认证算法（目前云原生网关只支持Hmac算法，后续将支持更多算法），填写加密关键信息。
6. 点击保存。
7. 为该服务创建路由。
8. 请求路由。

其中，Hmac认证算法的关键属性如下：

名称	描述
Access Key	唯一标识符。
Secret Key	与Access Key配对使用。
加密算法	支持hmac-sha1，hmac-sha256和hmac-sha512三种加密算法。
加密Headers列表	要在加密计算中使用的headers 列表。指定后客户端请求只能在此范围内指定 headers，如果未指定，就会在所有客户端请求指定的 headers 加入加密计算。
URL参数编码	当设置为是时，将对签名中的URI参数进行编码，默认为是。

结果验证

在服务中开启并配置后端访问签名后，请求路由，网关会在转发给后端服务的Header头X-HMAC-SIGNATURE中携带签名信息。

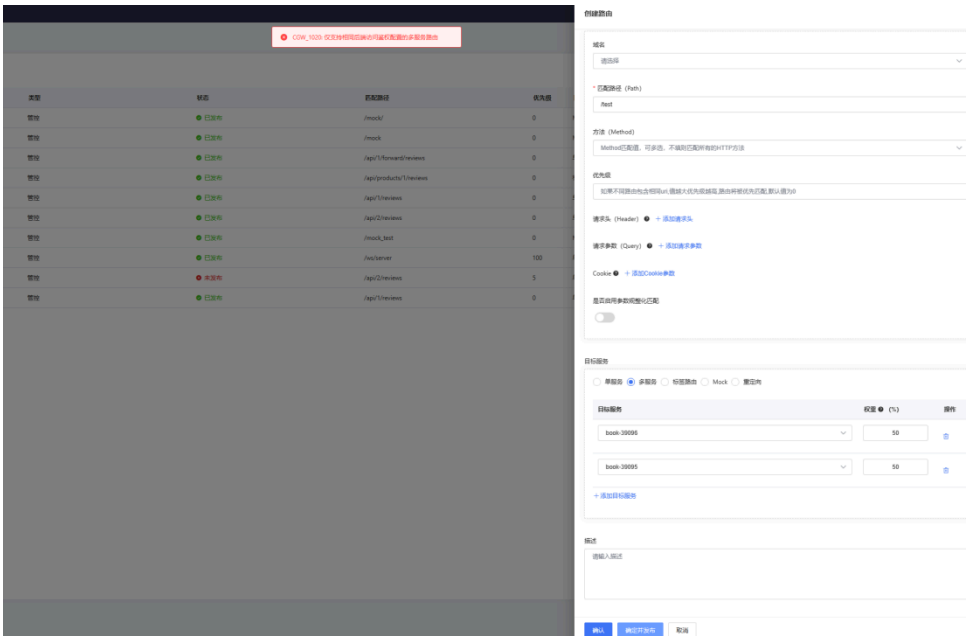


注意

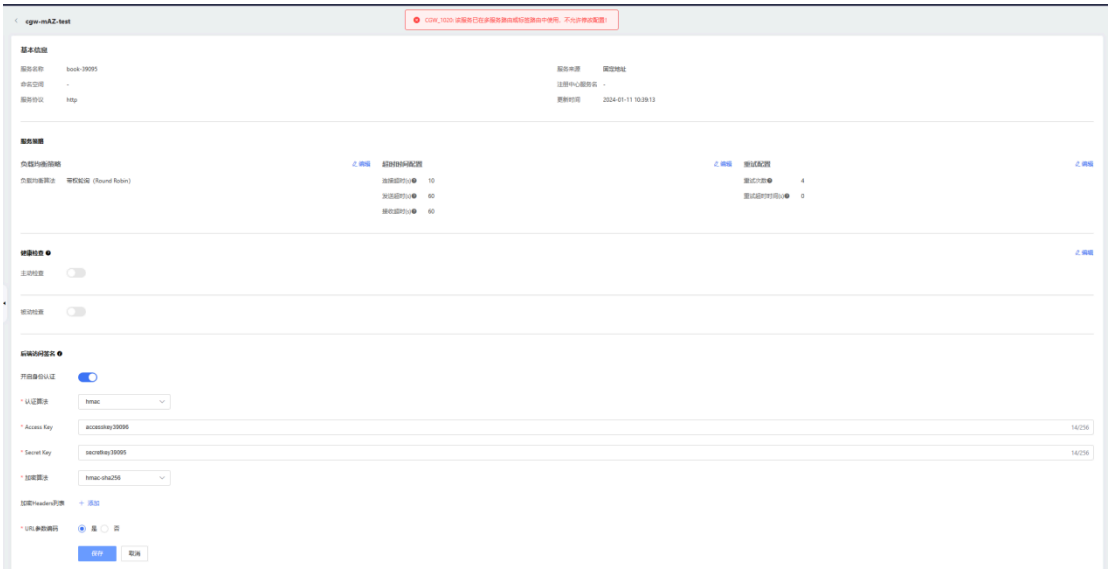
1. 当创建多服务路由或标签路由时，仅支持以下情况：

- 全部服务配置了相同的后端访问签名配置。
- 只有其中一个后端服务配置了后端访问配置。
- 全部服务都没有配置后端访问签名配置。

如果多服务路由或标签路由中选择的服务中配置了不同的后端访问签名配置，则无法创建。



2. 当配置了后端访问签名配置的服务已经在多服务路由或标签路由中引用时，则不允许修改配置信息。若要修改，请先删除该路由。



使用云原生网关实现websocket流量代理

概述

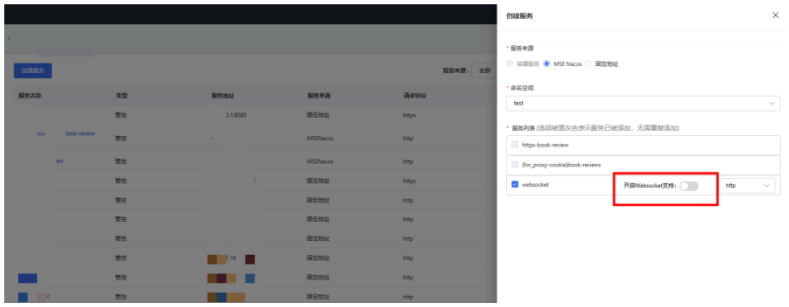
WebSocket 协议允许客户端和服务器之间进行实时的双向数据传输，从而确保了连接的持久性和低延迟。可以在云原生网关中开启websocket支持，实现websocket流量代理。

前提

1. 已开通云原生网关实例；
2. 已部署后端websocket server服务

云原生网关中开启WebSocket支持

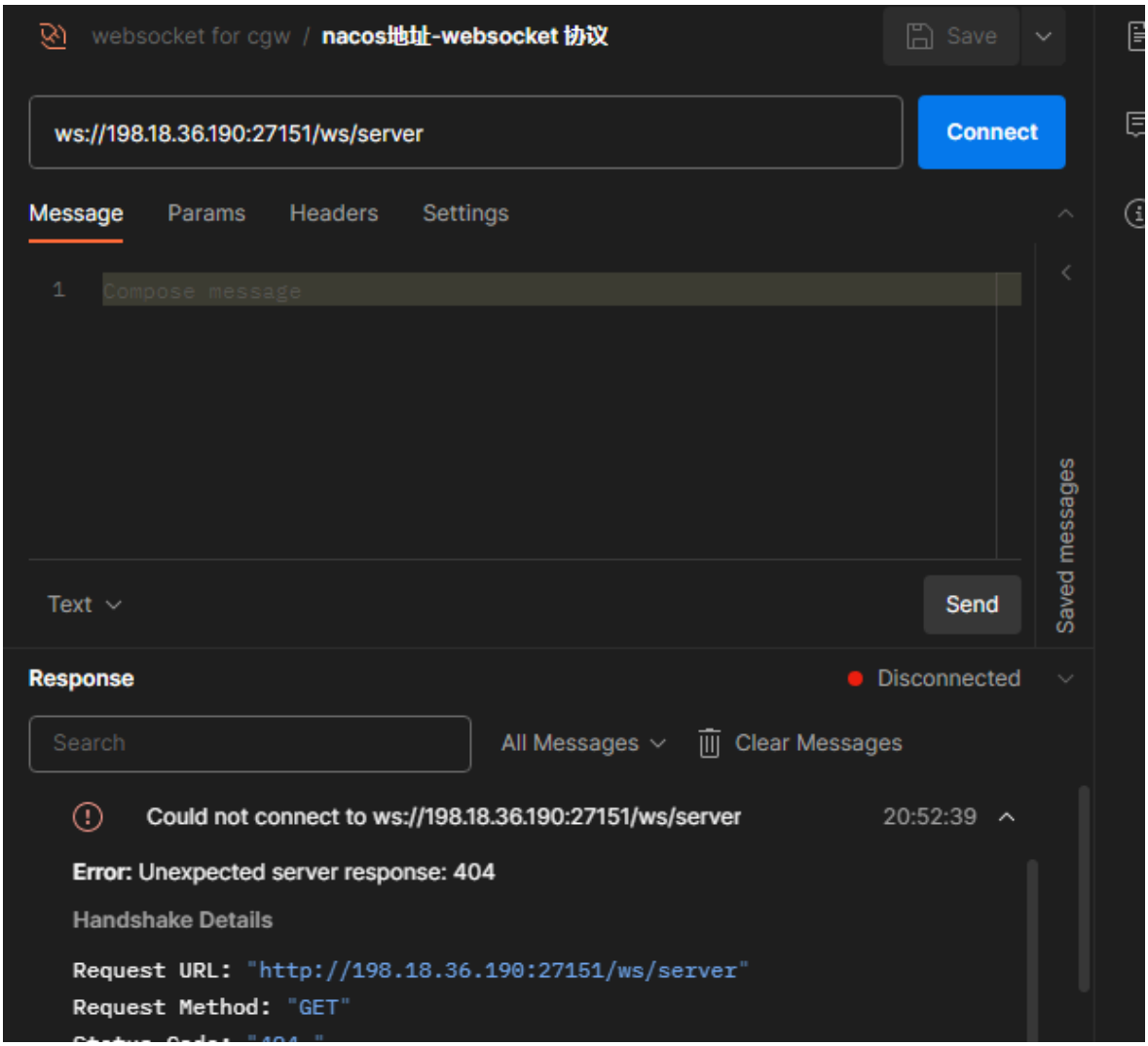
我们采用在MSE Nacos中注册后端websocket服务的方式进行服务部署，后端服务示例可部署demo应用（暴漏websocket应用路径为/ws/server）。在创建服务时打开“开启websocket支持”开关，则可对该服务进行websocket协议请求。



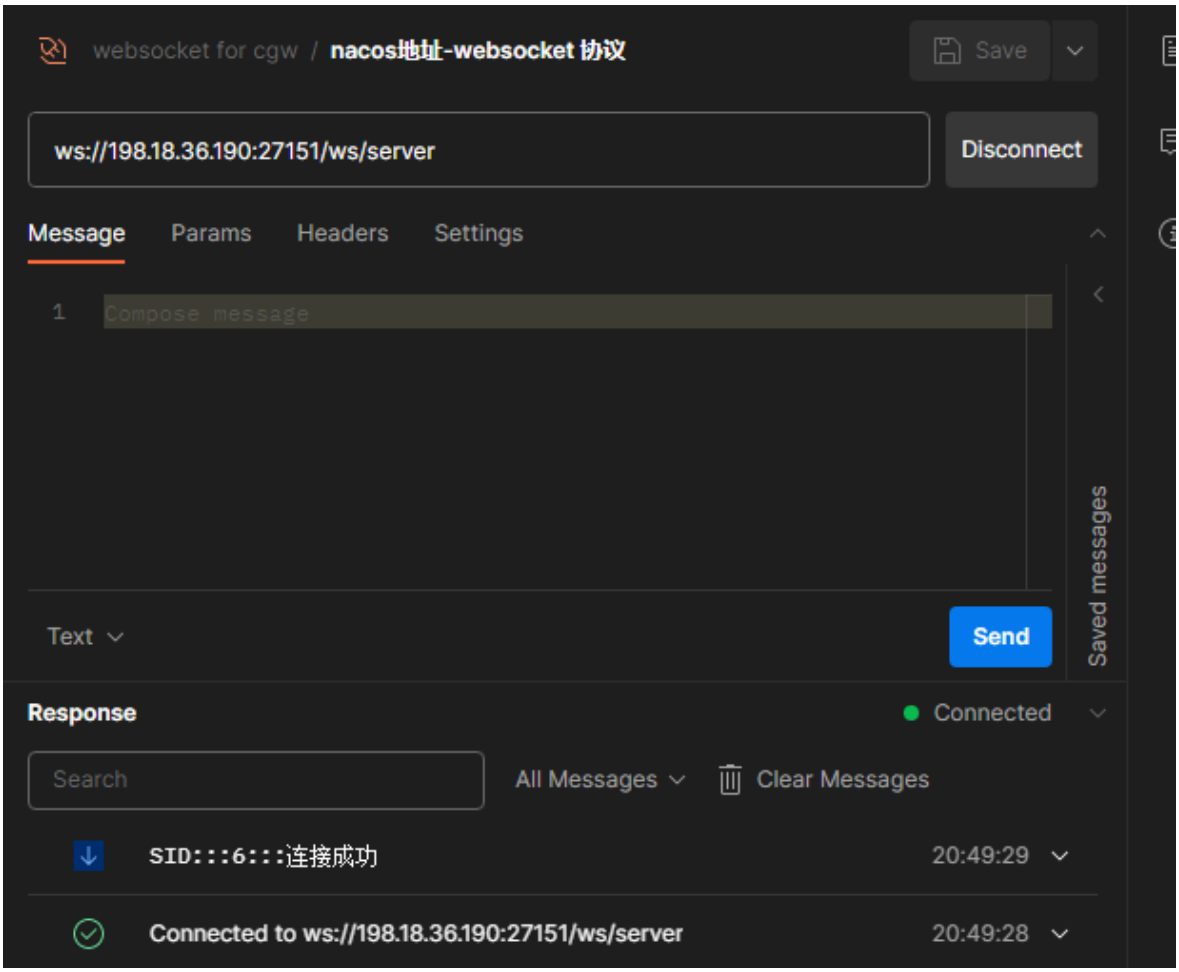
为该服务创建一条路由，匹配路径填写/*即可。

结果验证

可通过postman软件发起到网关的websocket请求，请求协议前缀为ws://或wss://。当服务关闭websocket支持时，请求失败。



当服务开启websocket支持时，请求成功。



注意

后端服务开启/关闭websocket支持无法在已经被引用的多服务路由或标签路由上生效。

微服务治理中心

微服务治理入门概述

要使用微服务治理中心，用户可以参考“ECS微服务应用接入MSE治理中心”或“云容器引擎应用接入微服务治理中心”，在云主机或云容器引擎中安装微服务治理组件，即可将部署的SpringCloud应用或Dubbo应用接入微服务治理中心，然后使用微服务治理中心的治理功能。

微服务治理中心支持Springcloud、Dubbo框架的微服务应用。功能列表如下所示：

模块	SpringCloud&Dubbo
服务查询	支持查看服务列表和服务契约。
接口详情	支持查看接口列表和接口监控数据。

模块	SpringCloud&Dubbo
节点详情	支持查看节点列表和节点监控数据。
流量治理	支持无损上下线、标签路由、离群摘除、服务鉴权等治理规则。
流量防护	支持流量控制、熔断降级、自适应控制、Web防护等防护能力。
开发测试治理	支持服务测试、自动化回归、服务Mock等测试能力。
数据库治理	支持Sql监控、数据库灰度、数据库读写路由、连接池治理等数据库治理能力。
全链路灰度	支持按泳道对应用分组，并按泳道对流量进行路由。

微服务应用接入微服务治理中心后，即可支持服务查询、接口详情、节点详情，其他治理能力需要在控制台创建规则，启用规则后才会生效。

ECS微服务应用接入MSE治理中心

概述

部署在ECS的应用，可以通过安装MSE Agent的方式接入微服务治理中心，使用微服务治理中心提供的一系列服务治理能力。

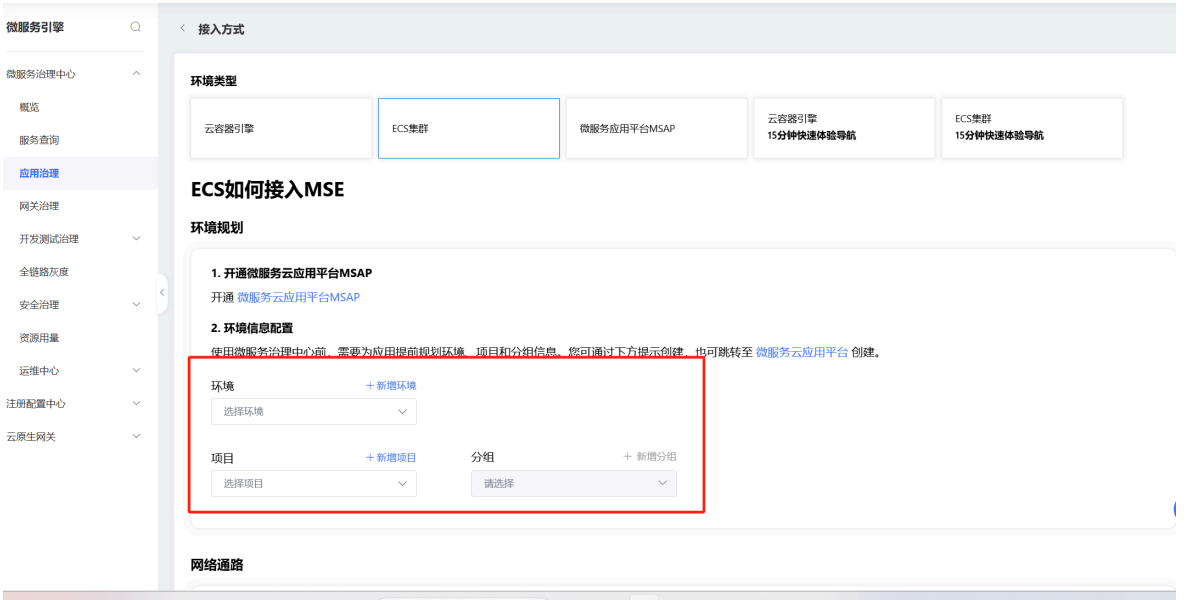
环境规划

1.开通微服务云应用平台MSAP

开通 微服务云应用平台。

2.环境信息配置

使用微服务治理中心前，需要为应用提前规划环境、项目和分组信息。您可通过进入微服务治理控制台->应用治理->应用接入->ECS集群进行配置，也可跳转至微服务云应用平台创建。



网络通路

1.DNS配置

上报域名为内网域名，需要在DNS配置文件（/etc/resolv.conf）首行添加nameserver 100.95.0.1 才生效。

2.创建VPC终端节点

使用微服务治理中心前，需要在目标VPC中创建终端节点。创建路径：微服务治理中心控制台->应用治理->应用接入->ECS集群->网络通路。

进入应用接入页面。

点击ECS集群，选择VPC，点击创建终端节点。

点击确定授权并创建VPC终端节点。

点击确定后，刷新页面，显示VPC的终端节点状态为已创建。

安装 MSE Agent

1.下载 MSE Agent

下载路径：微服务治理中心控制台->应用治理->应用接入->ECS集群页面->下载Agent按钮。

2.进入Agent压缩包所在目录并将其解压至任意工作目录下

unzip MseAgent.zip -d /user.workspace/

3.添加JVM参数到应用启动脚本中

在应用服务的启动脚本中添加以下JVM参数

-javaagent:{user.workspace}/MseAgent/oneagent/core/oneagent@0.0.2/one-java-agent.jar

```
-Dmsgc.enable=true
-Dmsgc.licenseKey={your licenseKey}
-Dmsgc.appName={your app name}
-Dmsgc.namespace={your namespace}
-Dmsgc.project={your project code}
-Dmsgc.group={your group code}
-Dmsgc.msc.endpoint=msgc-{region-code}.cnsp-internal.ctyun.cn:27144
-noverify
```

参数解释：

标签名	标签值
msgc.licenseKey	是我们为您自动生成的接入凭证，请保管好此凭证，不要泄露给第三方。可在微服务治理中心控制台->应用治理->应用接入->ECS集群页面获取。
msgc.appName	是您的应用名，请将其替换成您自己的应用名。
msgc.namespace	是您接入的环境，默认可不填。如填写可在微服务治理中心控制台->应用治理->应用接入->ECS集群->环境规划页面获取。
msgc.project	是您接入的项目，默认可不填。如填写可在微服务治理中心控制台->应用治理->应用接入->ECS集群->环境规划页面获取。
msgc.group	是您接入的项目分组，默认可不填。如填写可在微服务治理中心控制台->应用治理->应用接入->ECS集群->环境规划页面获取。

4.验证应用已接入MSE

查看应用列表，确认你的应用已接入到微服务治理中心。

云容器引擎应用接入微服务治理中心

概述

部署在云容器引擎的应用可以直接通过安装cubems插件实现应用接入。

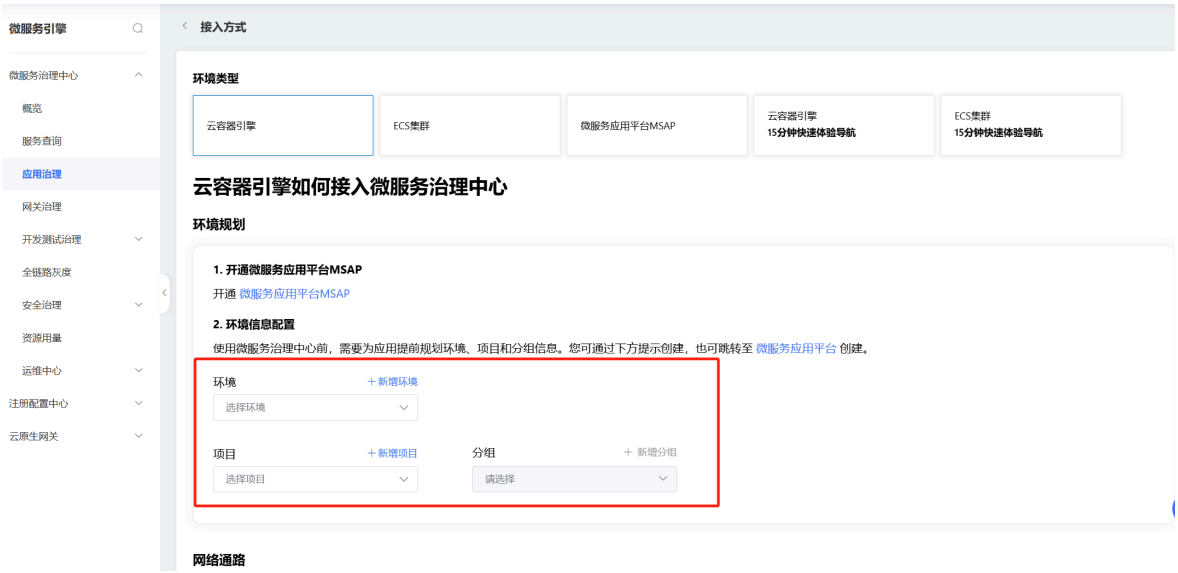
环境规划

1.开通微服务云应用平台MSAP

开通 微服务云应用平台。

2.环境信息配置

使用微服务治理中心前，需要为应用提前规划环境、项目和分组信息。您可通过进入微服务治理控制台->应用治理->应用接入->云容器引擎进行配置，也可跳转至微服务云应用平台创建。



网络通路

1.DNS配置

上报域名为内网域名，需要在DNS配置文件（/etc/resolv.conf）首行添加nameserver 100.95.0.1 才生效。

2.创建VPC终端节点

使用微服务治理中心前，需要在目标VPC中创建终端节点。创建路径：微服务治理中心控制台->应用治理->应用接入->ECS集群->网络通路。

进入应用接入页面。

点击云容器引擎，选择VPC，点击创建终端节点。

点击确定授权并创建VPC终端节点。

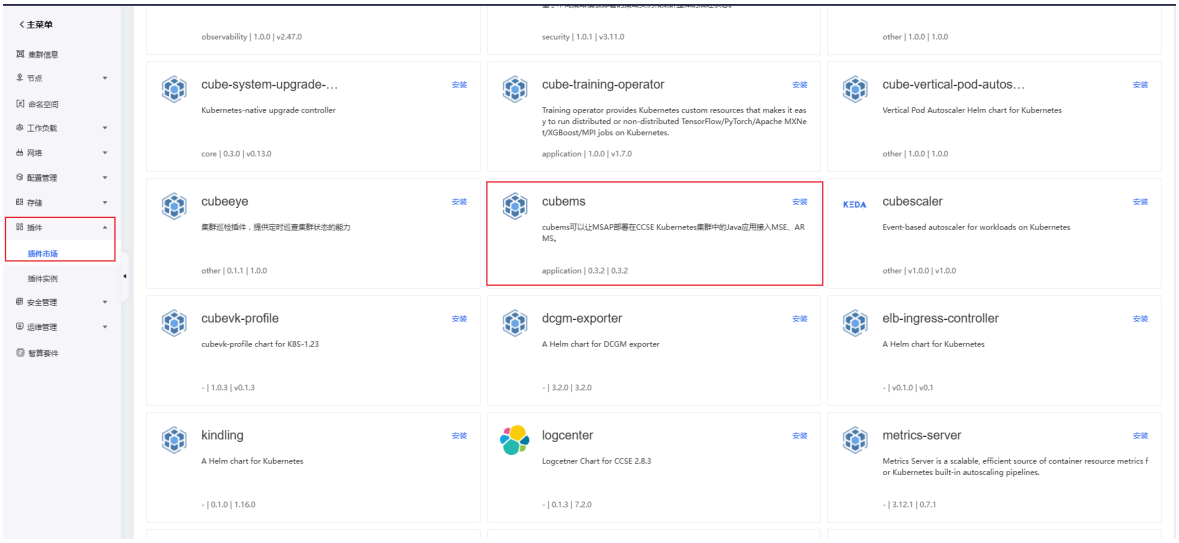
点击确定后，刷新页面，显示VPC的终端节点状态为已创建。

安装治理插件

1.登录“云容器引擎”控制台。

2.在左侧菜单栏选择“集群”，点击目标集群。

3.在集群管理页面点击“插件”-“插件市场”，选择“cubems”插件安装。



接入流程

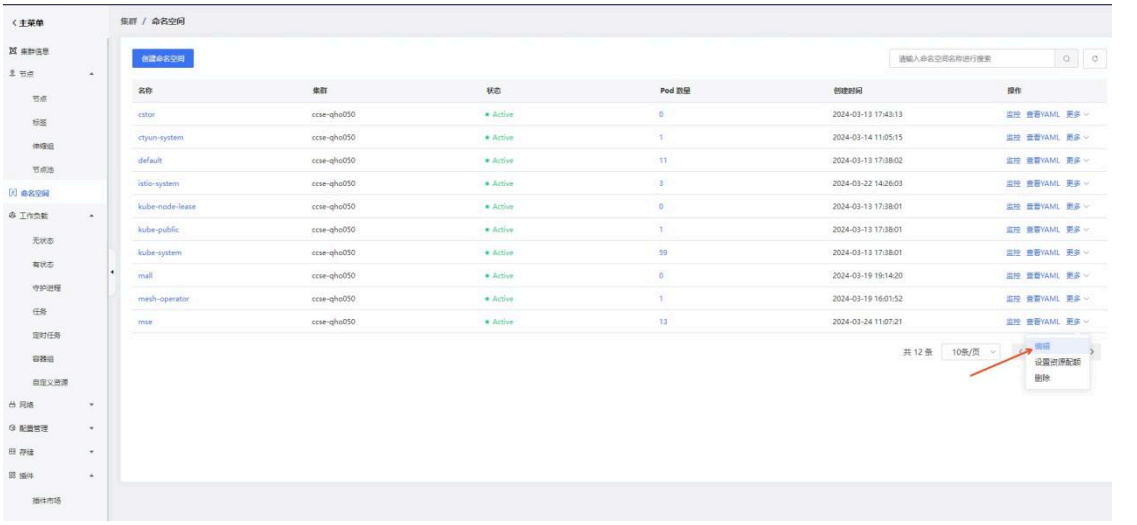
场景一：为命名空间中的应用开启微服务治理

步骤1：注入微服务治理中心

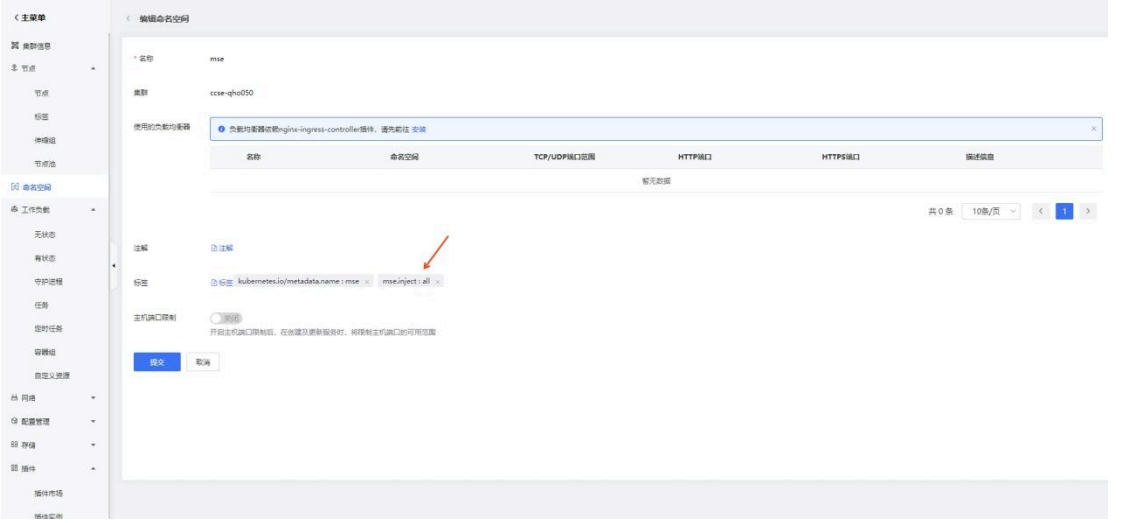
1.登录“云容器引擎”控制台。

2.在左侧菜单栏选择“集群”，点击目标集群。

3.在集群管理页面点击“命名空间”，选择目标命名空间，点击“更多”，随后点击“编辑”。



4.在编辑命名空间页面，新增“标签”，标签名为：mse.inject，标签值为：all或optional。



标签名	标签值
mse.inject	1、all：在当前命名空间下的所有应用都会自动接入微服务治理中心。2、optional：在当前命名空间下的应用存在环境变量 MSE_CUBE_AUTO_ENABLE=on 时会自动接入微服务治理中心。

步骤2：配置微服务治理启动参数

在发布应用时，配置指定环境变量，可指定注入微服务治理中心的应用名、命名空间和标签等信息。可通过编辑应用的YAML文件进行修改。

```
spec:
 template:
  spec:
  containers:
   env:
   - name:"MSE_APP_NAME"
     value:"app-a"
   - name:"MSE_NAMESPACE"
     value:"default"
   - name:"MSGC_PROJECT"
     value:"default"
   - name:"MSGC_GROUP"
     value:"default"
```

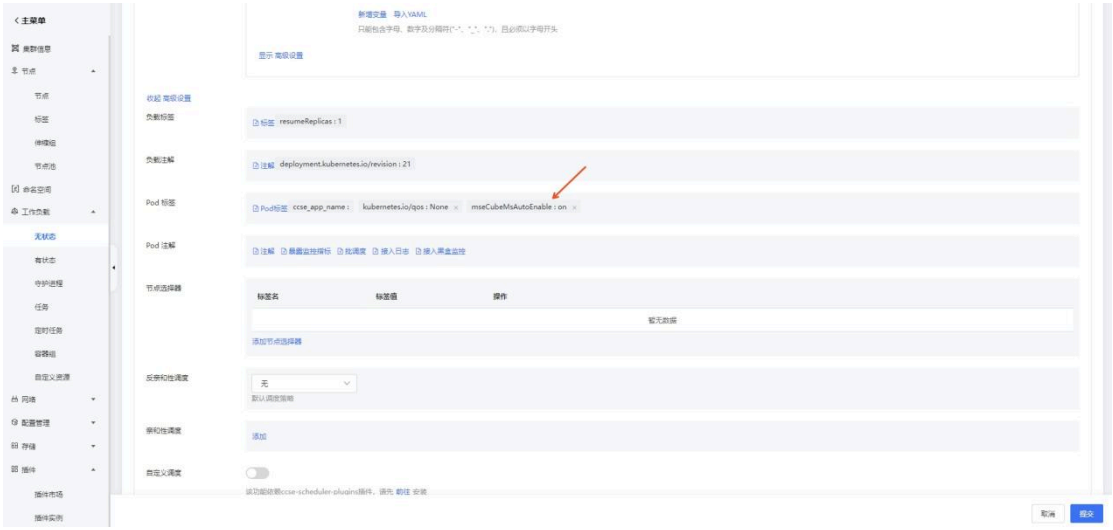
环境变量配置如下：

标签名	标签值
MSE_APP_NAME	接入到微服务治理中心的应用名。
MSE_NAMESPACE（选填）	接入到微服务治理中心的命名空间，默认为：default。如需环境规划可在微服务治理中心控制台->应用治理->应用接入->云容器引擎->环境规划页面获取。
MSGC_PROJECT（选填）	接入到微服务治理中心的项目，默认为：default。如需环境规划可在微服务治理中心控制台->应用治理->应用接入->云容器引擎->环境规划页面获取。
MSGC_GROUP（选填）	接入到微服务治理中心的分组，默认为：default。如需环境规划可在微服务治理中心控制台->应用治理->应用接入->云容器引擎->环境规划页面获取。

场景二：为单个应用开启微服务治理

步骤1：注入微服务治理中心

1. 登录“云容器引擎”控制台。
2. 在左侧菜单栏选择“集群”，点击目标集群。
3. 在集群管理页面点击“工作负载”-“无状态”，选择目标命名空间。
4. 在Deployment列表页选择指定Deployment，并点击“全量替换”，进入Deployment编辑页。
5. 在Deployment编辑页点击“显示高级设置”，新增“Pod标签”：mseCubeMsAutoEnable:on。



标签名	标签值
mseCubeMsAutoEnable	on

步骤2：配置微服务治理启动参数

在发布应用时，配置指定环境变量，可指定注入微服务治理中心的应用名、命名空间和标签等信息。可通过编辑应用的YAML文件进行修改。

```
spec:
 template:
  spec:
  containers:
   env:
   - name: "MSE_APP_NAME"
     value: "app-a"
   - name: "MSE_NAMESPACE"
     value: "default"
   - name: "MSGC_PROJECT"
      value: "default"
   - name: "MSGC_GROUP"
      value: "default"
```

环境变量配置如下：

标签名	标签值
MSE_APP_NAME	接入到微服务治理中心的应用名。
MSE_NAMESPACE（选填）	接入到微服务治理中心的命名空间，默认为：default。如需环境规划可在微服务治理中心控制台->应用治理->应用接入->云容器引擎->环境规划页面获取。

标签名	标签值
MSGC_PROJECT（选填）	接入到微服务治理中心的项目，默认为：default。如需环境规划可在微服务治理中心控制台->应用治理->应用接入->云容器引擎->环境规划页面获取。
MSGC_GROUP（选填）	接入到微服务治理中心的分组，默认为：default。如需环境规划可在微服务治理中心控制台->应用治理->应用接入->云容器引擎->环境规划页面获取。

验证应用已接入MSE微服务治理

查看应用列表，确认您的应用已经接入到MSE微服务治理。

15分钟完成微服务治理中心能力快速体验

ECS集群15分钟快速体验

前置条件

创建VPCE

1.上报域名为内网域名，需要在DNS配置文件（/etc/resolv.conf）首行添加nameserver 100.95.0.1 才生效。

2.使用微服务治理中心前，需要在目标VPC中创建终端节点。创建路径：微服务治理中心控制台->应用治理->应用接入->ECS集群->网络通路。

进入应用接入页面。

点击ECS集群，选择VPC，点击创建终端节点。

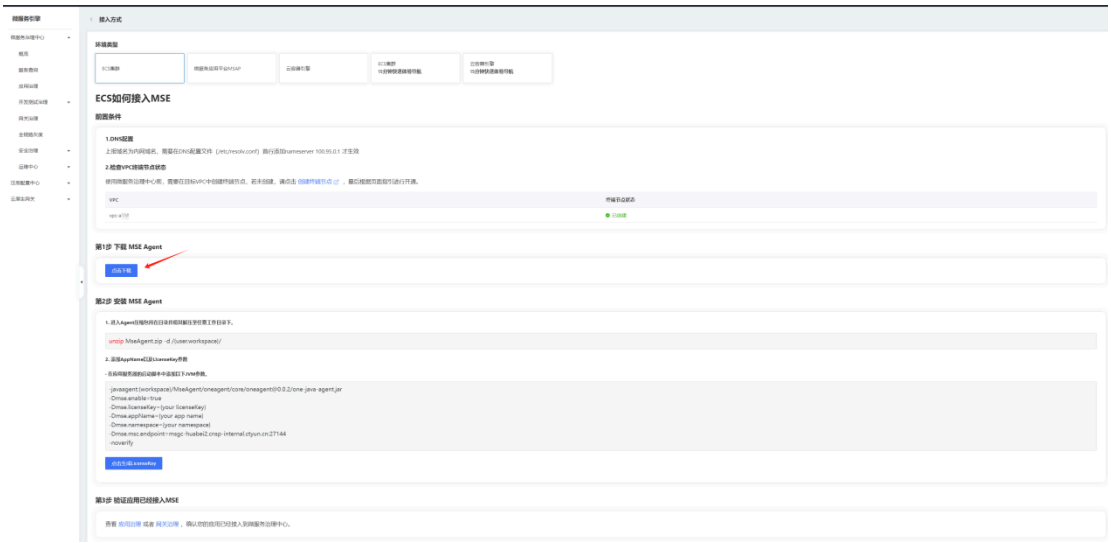
点击确定授权并创建VPC终端节点。

点击确定后，刷新页面，显示VPC的终端节点状态为已创建。

下载MSE Agent

1. 进入微服务治理中心->应用治理->应用接入页面。

2. 点击下载MSE Agent并上传至云主机。



开通微服务引擎-注册配置中心Nacos

进入微服务引擎控制台，选择注册配置中心->实例列表->创建实例，开通注册配置中心Nacos。

部署Demo

Demo介绍

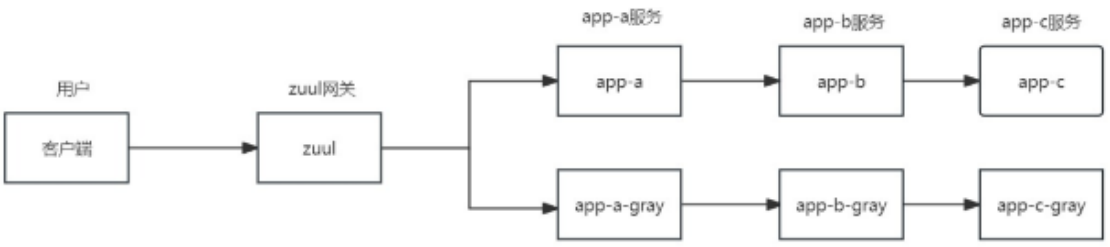
该Demo快速体验服务治理中心，可体验应用详情、接口详情、流量治理、流量防护等主要功能，提供以spring boot为框架的simple-demo单体web应用，和以spring cloud、feign和nacos为框架的微服务架构应用app-a、app-b和app-c，方便用户快速体验微服务治理能力。（spring boot应用：推荐CPU1C以上，内存1024以上，spring cloud应用：推荐CPU2C以上，内存2048以上）。

1. simple-demo使用为快速上手demo，使用spring boot启动接入服务治理。

应用名	服务框架版本	涉及组件
simple-demo	spring boot 2.7.17	spring boot 2.7.17

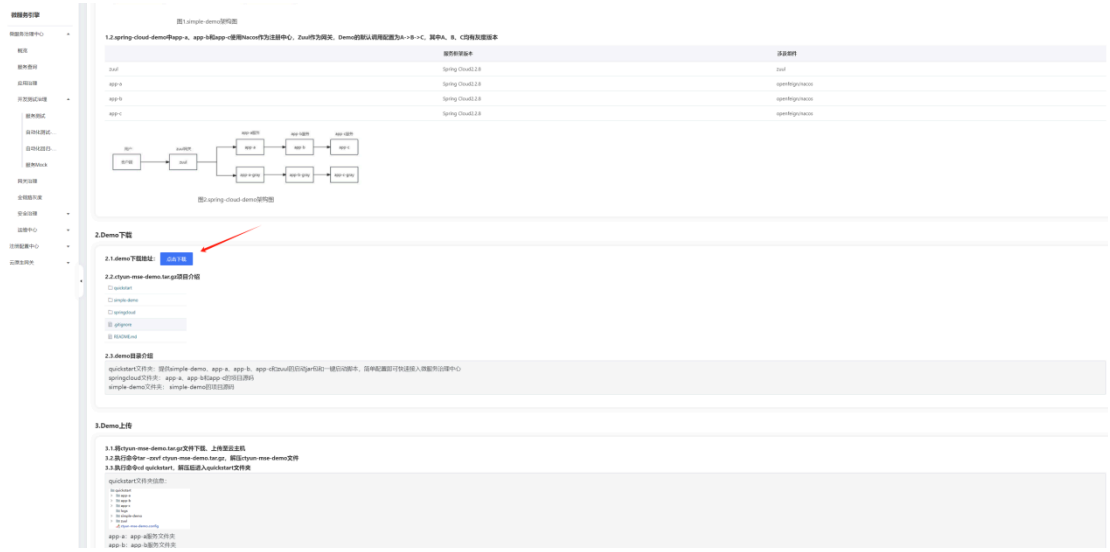
2. spring-cloud-demo中app-a、app-b和app-c使用Nacos作为注册中心，Zuul作为网关，Demo的默认调用配置为A->B->C，其中A、B、C均有灰度版本。

应用名	服务框架版本	涉及组件
zuul	spring boot 2.7.17	spring boot 2.7.17
app-a	Spring Cloud2.2.8	openfeign/nacos
app-b	Spring Cloud2.2.8	openfeign/nacos
app-c	Spring Cloud2.2.8	openfeign/nacos



Demo下载

1. Demo下载地址：微服务治理中心控制台->应用治理->应用接入->ECS 15分钟快速体验导航



2. ctyun-mse-demo.tar.gz项目介绍

quickstart文件夹：提供simple-demo、app-a、app-b、app-c和zuul的启动jar包和一键启动脚本，简单配置即可快速接入微服务治理中心。

springcloud文件夹：app-a、app-b和app-c的项目源码。

simple-demo文件夹：simple-demo的项目源码。

Demo上传

1. 将ctyun-mse-demo.tar.gz文件下载、上传至云主机。
2. 执行命令tar -zxvf ctyun-mse-demo.tar.gz，解压ctyun-mse-demo文件。
3. 执行命令cd quickstart，解压后进入quickstart文件夹。quickstart文件夹信息：

app-a：app-a服务文件夹

app-b：app-b服务文件夹

app-c：app-c服务文件夹

logs：项目启动后日志存放路径

simple-demo：快速上手demo

zuul ： 网关服务

ctyun-mse-demo.config ： 启动配置文件

Demo启动

1. 启动simple-demo，快速体验接入流程。

- 编辑配置文件ctyun-mse-demo.config，修改simple-demo端口（可选），mse-agent-path、mse-licenseKey和mse-msc-endpoint参数。

simple-demo-server-port：simple-demo默认启动端口为26150

mse-agent-path：前置条件、agent上传云主机存放路径

mse-licenseKey：前置条件、控制台生成licenseKey

mse-msc-endpoint：前置条件、控制台获取mse-msc-endpoint

- 执行命令cd simple-demo，进入simple-demo文件夹。
- 执行命令sh start_simple.sh，启动脚本start_simple.sh。
- 执行命令more ../logs/simple-demo-info.log,查看logs文件中的info.log文件是否启动成功。

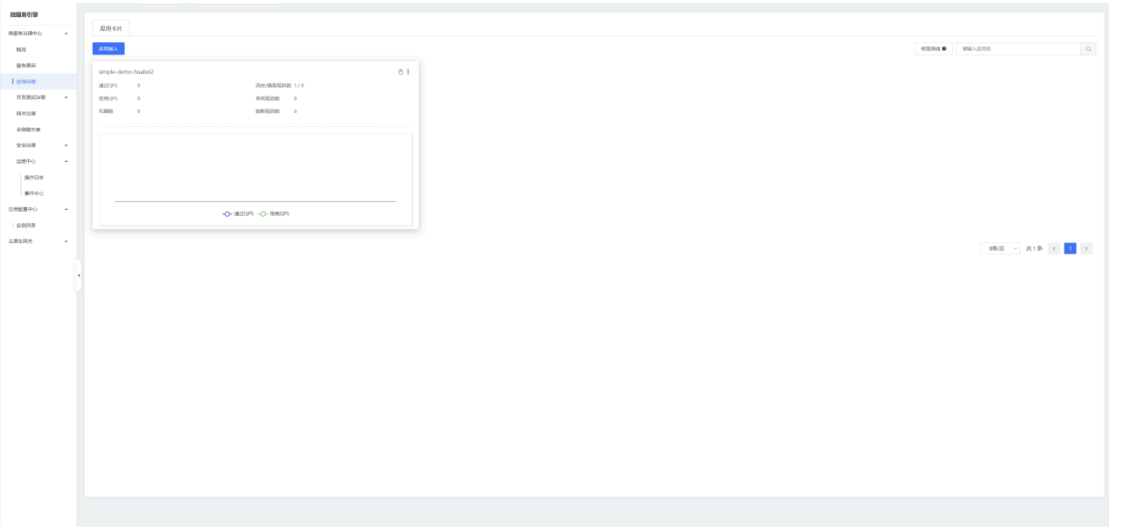
```
============================================== start simple-demo server ==============================================
Deployment Path: /app/msgc
 Start Time: 2023-11-22 09:48:3
agent_path /app/msgc licenseKey  XXXXXXXXXX
PING 33.0.0.13 (33.0.0.13) 56(84) bytes of data.
64 bytes from 33.0.0.13: icmp_seq=1 ttl=64 time=0.291 ms

--- 33.0.0.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.291/0.291/0.291/0.000 ms
============================================== simple-demo server Successfully started==============================================
2023-11-22 09:48:35.645 INFO [oneagent plugin mse-agent start] : Start plugin, name: mse-agent
2023-11-22 09:48:35.645 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin, name: ctyunArmsAgent-plugin
GlobalIgnoredTypesConfigurer configureIgnoredClassLoaders
2023-11-22 09:48:36.532 INFO [oneagent plugin mse-agent start] com.ctg.mse.agent.starter.initializer.LogbackInitializer : mse agent log path /root/logs/mse/agent.1333264.node-vm-3
2023-11-22 09:48:45.644 INFO [oneagent plugin ahas-agent start] : Start plugin, name: ahas-agent
2023-11-22 09:48:45.662 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.AgentLauncher : ahas agent starting ...
2023-11-22 09:48:45.754 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.initializer.LogbackInitializer : ahas agent log path /root/logs/ctg/ahas/logs.1333264.node-
vm-3
2023-11-22 09:48:46.937 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin success, name: ctyunArmsAgent-plugin
2023-11-22 09:48:50.149 INFO [oneagent plugin mse-agent start] : Start plugin success, name: mse-agent
2023-11-22 09:49:22.777 INFO [oneagent plugin ahas-agent start] : Start plugin success, name: ahas-agent

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.12.RELEASE)

2023-11-22 09:49:33.189  INFO 1333264 --- [           main] com.ctyun.msgc.b.BApplication            : No active profile set, falling back to default profiles: default
```

- 查看应用治理或者网关治理，确认您的应用已经接入到微服务治理中心。



启动app-a、app-b、app-c和zuul，快速体验微服务治理能力。

编辑配置文件ctyun-mse-demo.config，修改应用A、B、C端口信息（可选）、注册中心Nacos相关配置、mse-agent-path、mse-licenseKey、mse-mscmendpoint等信息。配置详情如下：

app-a-server-port：设置A服务端口：默认26160

app-b-server-port：设置B服务端口：默认26165

app-c-server-port：设置C服务端口：默认26170

zuul-server-port：设置zuul服务端口：默认26180

nacos-server-addr：前置条件、nacos服务器地址

nacos-server-username：前置条件、 nacos服务器用户名

nacos-server-password：前置条件、nacos服务器密码

nacos-namespace：前置条件、nacos服务器命名空间

mse-agent-path：前置条件、agent上传云主机存放路径

mse-licenseKey：前置条件、控制台生成licenseKey

mse-msc-endpoint：前置条件、控制台获取mse-msc-endpoint

启动app-a服务

1. 执行命令cd app-a，进入文件夹。
2. 执行命令sh start_app_a.sh，启动脚本。
3. 执行命令more ../logs/app-a-info.log，查询日志。

```
================================================ start app-a server ================================================
Deployment Path: /app/msgc
 Start Time: 2023-11-22 09:48:3
agent_path /app/msgc licenseKey  XXXXXXXXXXX
PING 33.0.0.13 (33.0.0.13) 56(84) bytes of data.
64 bytes from 33.0.0.13: icmp_seq=1 ttl=64 time=0.291 ms

--- 33.0.0.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.291/0.291/0.291/0.000 ms
================================================  app-a server Successfully started================================================
2023-11-22 09:48:35.645 INFO [oneagent plugin mse-agent start] : Start plugin, name: mse-agent
2023-11-22 09:48:35.645 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin, name: ctyunArmsAgent-plugin
GlobalIgnoredTypesConfigurer configureIgnoredClassLoaders
2023-11-22 09:48:36.532 INFO [oneagent plugin mse-agent start] com.ctg.mse.agent.starter.initializer.LogbackInitializer : mse agent log path /root/logs/mse/agent.1333264.node-vm-3
2023-11-22 09:48:45.644 INFO [oneagent plugin ahas-agent start] : Start plugin, name: ahas-agent
2023-11-22 09:48:45.662 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.AgentLauncher : ahas agent starting ...
2023-11-22 09:48:45.754 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.initializer.LogbackInitializer : ahas agent log path /root/logs/ctg/ahas/logs.1333264.node-
vm-3
2023-11-22 09:48:46.937 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin success, name: ctyunArmsAgent-plugin
2023-11-22 09:48:50.149 INFO [oneagent plugin mse-agent start] : Start plugin success, name: mse-agent
2023-11-22 09:49:22.777 INFO [oneagent plugin ahas-agent start] : Start plugin success, name: ahas-agent

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.12.RELEASE)

2023-11-22 09:49:33.189  INFO 1333264 --- [           main] com.ctyun.msgc.b.BApplication            : No active profile set, falling back to default profiles: default
```

4. 查看应用治理或者网关治理，确认您的应用已经接入到微服务治理中心。

启动app-b服务

1. 执行命令cd app-b，进入文件夹。
2. 执行命令sh start_app_b.sh，启动脚本。
3. 执行命令more ../logs/app-b-info.log，查询日志。

```
================================================ start app-b server ================================================
Deployment Path: /app/msgc
 Start Time: 2023-11-22 09:48:3
agent_path /app/msgc licenseKey  XXXXXXXXXXX
PING 33.0.0.13 (33.0.0.13) 56(84) bytes of data.
64 bytes from 33.0.0.13: icmp_seq=1 ttl=64 time=0.291 ms

--- 33.0.0.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.291/0.291/0.291/0.000 ms
================================================  app-b server Successfully started================================================
2023-11-22 09:48:35.645 INFO [oneagent plugin mse-agent start] : Start plugin, name: mse-agent
2023-11-22 09:48:35.645 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin, name: ctyunArmsAgent-plugin
GlobalIgnoredTypesConfigurer configureIgnoredClassLoaders
2023-11-22 09:48:36.532 INFO [oneagent plugin mse-agent start] com.ctg.mse.agent.starter.initializer.LogbackInitializer : mse agent log path /root/logs/mse/agent.1333264.node-vm-3
2023-11-22 09:48:45.644 INFO [oneagent plugin ahas-agent start] : Start plugin, name: ahas-agent
2023-11-22 09:48:45.662 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.AgentLauncher : ahas agent starting ...
2023-11-22 09:48:45.754 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.initializer.LogbackInitializer : ahas agent log path /root/logs/ctg/ahas/logs.1333264.node-
vm-3
2023-11-22 09:48:46.937 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin success, name: ctyunArmsAgent-plugin
2023-11-22 09:48:50.149 INFO [oneagent plugin mse-agent start] : Start plugin success, name: mse-agent
2023-11-22 09:49:22.777 INFO [oneagent plugin ahas-agent start] : Start plugin success, name: ahas-agent

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.12.RELEASE)

2023-11-22 09:49:33.189  INFO 1333264 --- [           main] com.ctyun.msgc.b.BApplication            : No active profile set, falling back to default profiles: default
```

4. 查看应用治理或者网关治理，确认您的应用已经接入到微服务治理中心。

启动app-c服务

1. 执行命令cd app-c，进入文件夹。
2. 执行命令sh start_app_c.sh，启动脚本。

3. 执行命令more ../logs/app-c-info.log，查询日志。

```
============================================== start app-c server ==============================================
Deployment Path: /app/msgc
 Start Time: 2023-11-22 09:48:3
agent_path /app/msgc licenseKey  XXXXXXXXXX
PING 33.0.0.13 (33.0.0.13) 56(84) bytes of data.
64 bytes from 33.0.0.13: icmp_seq=1 ttl=64 time=0.291 ms

--- 33.0.0.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.291/0.291/0.291/0.000 ms
============================================== app-c server Successfully started==============================================
2023-11-22 09:48:35.645 INFO [oneagent plugin mse-agent start] : Start plugin, name: mse-agent
2023-11-22 09:48:35.645 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin, name: ctyunArmsAgent-plugin
GlobalIgnoredTypesConfigurer configureIgnoredClassLoaders
2023-11-22 09:48:36.532 INFO [oneagent plugin mse-agent start] com.ctg.mse.agent.starter.initializer.LogbackInitializer : mse agent log path /root/logs/mse/agent.1333264.node-vm-3
2023-11-22 09:48:45.644 INFO [oneagent plugin ahas-agent start] : Start plugin, name: ahas-agent
2023-11-22 09:48:45.662 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.AgentLauncher : ahas agent starting ...
2023-11-22 09:48:45.754 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.initializer.LogbackInitializer : ahas agent log path /root/logs/ctg/ahas/logs.1333264.node-
vm-3
2023-11-22 09:48:46.937 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin success, name: ctyunArmsAgent-plugin
2023-11-22 09:48:50.149 INFO [oneagent plugin mse-agent start] : Start plugin success, name: mse-agent
2023-11-22 09:49:22.777 INFO [oneagent plugin ahas-agent start] : Start plugin success, name: ahas-agent

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.12.RELEASE)

2023-11-22 09:49:33.189  INFO 1333264 --- [           main] com.ctyun.msgc.b.BApplication            : No active profile set, falling back to default profiles: default
```

4. 查看应用治理或者网关治理，确认您的应用已经接入到微服务治理中心。

启动zuul

1. 执行命令cd zuul，进入文件夹。
2. 执行命令sh start_zuul.sh，启动脚本。
3. 执行命令more ../logs/zuul-info.log，查询日志。

```
============================================== start zuul server ==============================================
Deployment Path: /app/msgc
 Start Time: 2023-11-22 09:48:3
agent_path /app/msgc licenseKey  XXXXXXXXXX
PING 33.0.0.13 (33.0.0.13) 56(84) bytes of data.
64 bytes from 33.0.0.13: icmp_seq=1 ttl=64 time=0.291 ms

--- 33.0.0.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.291/0.291/0.291/0.000 ms
============================================== zuul server Successfully started==============================================
2023-11-22 09:48:35.645 INFO [oneagent plugin mse-agent start] : Start plugin, name: mse-agent
2023-11-22 09:48:35.645 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin, name: ctyunArmsAgent-plugin
GlobalIgnoredTypesConfigurer configureIgnoredClassLoaders
2023-11-22 09:48:36.532 INFO [oneagent plugin mse-agent start] com.ctg.mse.agent.starter.initializer.LogbackInitializer : mse agent log path /root/logs/mse/agent.1333264.node-vm-3
2023-11-22 09:48:45.644 INFO [oneagent plugin ahas-agent start] : Start plugin, name: ahas-agent
2023-11-22 09:48:45.662 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.AgentLauncher : ahas agent starting ...
2023-11-22 09:48:45.754 INFO [oneagent plugin ahas-agent start] com.ctg.ahas.agent.starter.initializer.LogbackInitializer : ahas agent log path /root/logs/ctg/ahas/logs.1333264.node-
vm-3
2023-11-22 09:48:46.937 INFO [oneagent plugin ctyunArmsAgent-plugin start] : Start plugin success, name: ctyunArmsAgent-plugin
2023-11-22 09:48:50.149 INFO [oneagent plugin mse-agent start] : Start plugin success, name: mse-agent
2023-11-22 09:49:22.777 INFO [oneagent plugin ahas-agent start] : Start plugin success, name: ahas-agent

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.12.RELEASE)

2023-11-22 09:49:33.189  INFO 1333264 --- [           main] com.ctyun.msgc.b.BApplication            : No active profile set, falling back to default profiles: default
```

4. 查看应用治理或者网关治理，确认您的应用已经接入到微服务治理中心。

常见错误

1. 请先修改云主机/etc/resolv.conf文件，在首行添加nameserver 100.95.0.1。

答：前置条件，vi /etc/resolv.conf，在首行添加nameserver 100.95.0.1。

2. 端口是否被占用。

答：配置文件重新修改服务启动端口。

3. mse-msc-endpoint地址错误，请重新接入。

答：根据前置文件获取mse-msc-endpoint。

4. Java Not Installed。

答：需要安装java环境。

5. agent path error。

答：上传至云服务器的Agent路径错误。

6. nacos addr or username or password error。

答：nacos的账号密码地址出现错误。

云容器引擎集群15分钟快速体验

前置条件

创建VPCE

使用微服务治理中心前，需要在目标VPC中创建终端节点。创建路径：微服务治理中心控制台->应用治理->应用接入->ECS集群->网络通路。

进入应用接入页面。

点击云容器引擎，选择VPC，点击创建终端节点。

点击确定授权并创建VPC终端节点。

点击确定后，刷新页面，显示VPC的终端节点状态为已创建。

部署Demo

Demo介绍

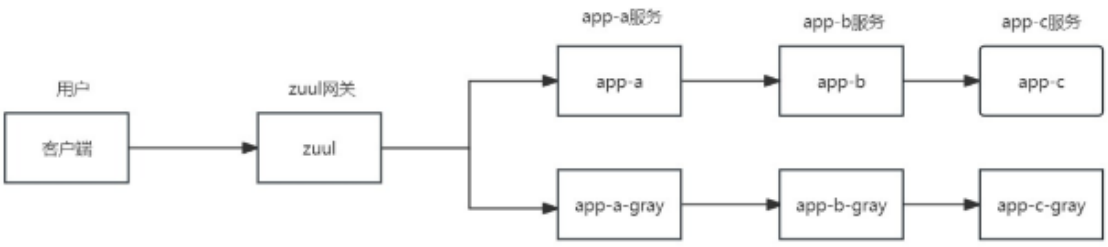
该Demo快速体验服务治理中心，可体验应用详情、接口详情、流量治理、流量防护等主要功能，提供以spring boot为框架的simple-demo单体web应用，和以spring cloud、feign和nacos为框架的微服务架构应用app-a、app-b和app-c，方便用户快速体验微服务治理能力。（spring boot应用：推荐CPU1C以上，内存1024以上，spring cloud应用：推荐CPU2C以上，内存2048以上）。

1. simple-demo使用为快速上手demo，使用spring boot启动接入服务治理。

应用名	服务框架版本	涉及组件
simple-demo	spring boot 2.7.17	spring boot 2.7.17

2. spring-cloud-demo中app-a、app-b和app-c使用Nacos作为注册中心，Zuul作为网关，Demo的默认调用配置为A->B->C，其中A,B,C均有灰度版本。

应用名	服务框架版本	涉及组件
zuul	spring boot 2.7.17	spring boot 2.7.17
app-a	Spring Cloud2.2.8	openfeign/nacos
app-b	Spring Cloud2.2.8	openfeign/nacos
app-c	Spring Cloud2.2.8	openfeign/nacos



Demo下载

1. Demo下载地址：微服务治理中心控制台->应用治理->应用接入->云容器引擎 15分钟快速体验导航。

2. ctyun-mse-demo.tar.gz项目介绍。

quickstart文件夹：提供simple-demo、app-a、app-b、app-c和zuul的启动jar包和一键启动脚本，简单配置即可快速接入微服务治理中心。

springcloud文件夹：app-a、app-b和app-c的项目源码。

simple-demo文件夹：simple-demo的项目源码。

Demo镜像打包

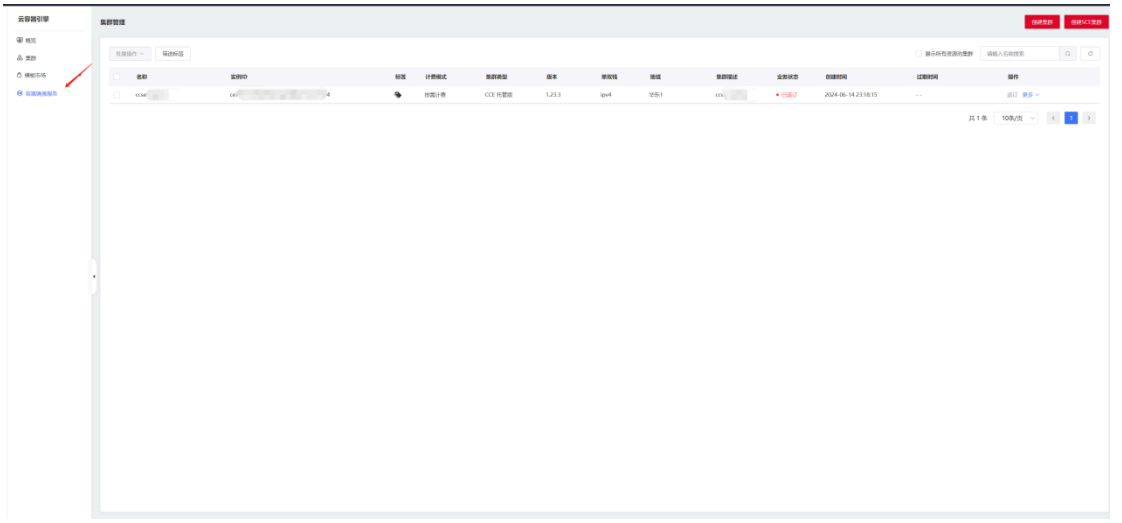
demo镜像制作并上传云容器引擎镜像仓库。（需提前制作jdk镜像包）

1. 通过jar文件和dockerfile文件制作镜像。

```
FROM jdk8
ADD ./app-a.jar /usr/local
WORKDIR /usr/local
RUN cd /usr/local && ls
CMD [“java”, “-jar”, “/usr/local/app-a.jar”]
```

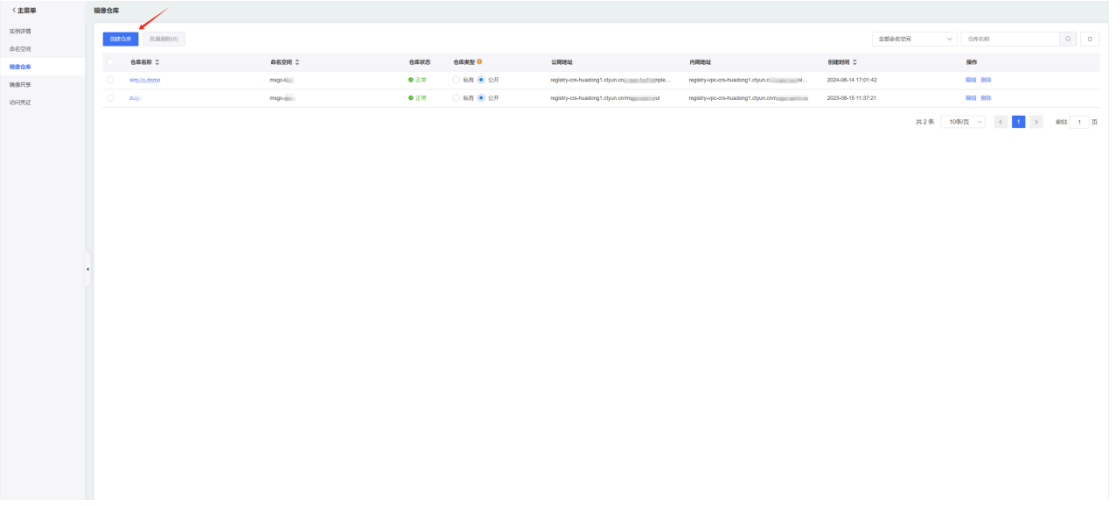
2. 通过容器镜像服务创建镜像仓库，镜像服务->实例列表->镜像仓库->创建仓库

进入云容器引擎控制台，点击容器镜像服务。



在实例列表页面，选择容器镜像服务实例。

进入镜像仓库菜单，点击创建仓库。



选择命名空间，设置仓库名称，点击创建。

3. demo上传至云容器引擎镜像仓库。

- 执行docker build -f ./Dockerfile-a -t msgc-app-a:1.1 .命令，构建demo镜像。

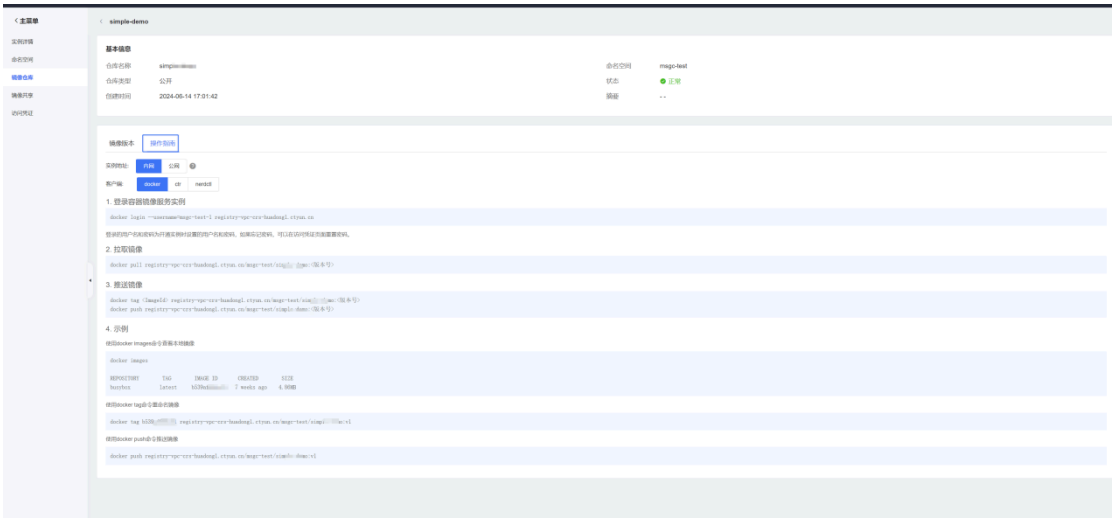
```
docker build -f ./Dockerfile-a -t msgc-app-a:1.1 .
```

- 登录容器镜像服务实例。

```
docker login --username={user} registry-crs-{regionCode}.ctyun.cn
```

- 推送镜像。

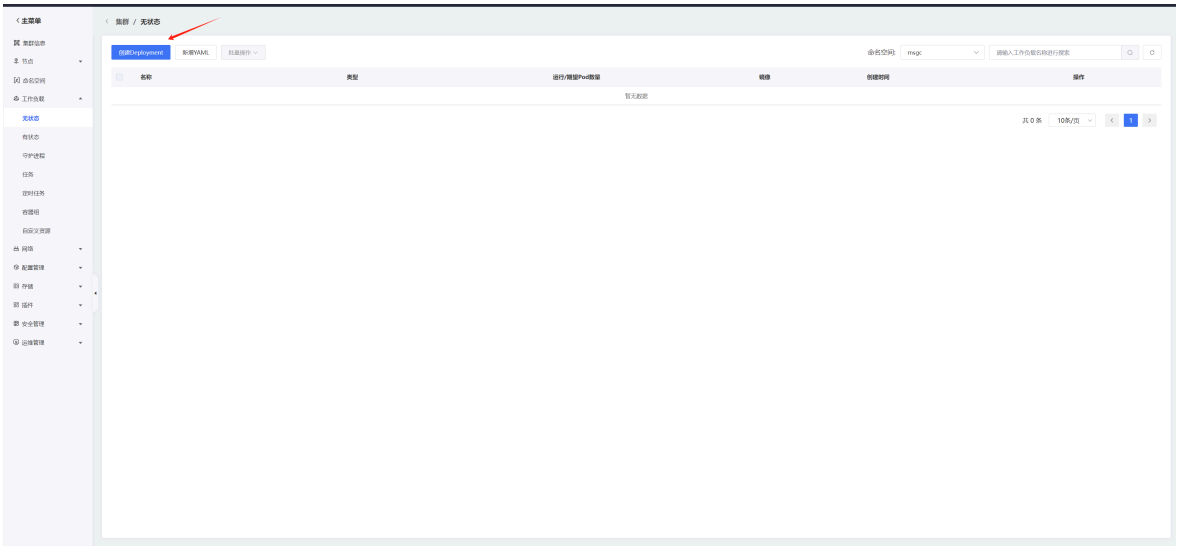
docker tag <ImageId> registry-vpc-crs-{regionCode}.ctyun.cn/msgc-test/<镜像名>:<版本号>
docker push registry-vpc-crs-{regionCode}.ctyun.cn/msgc-test/<镜像名>:<版本号>



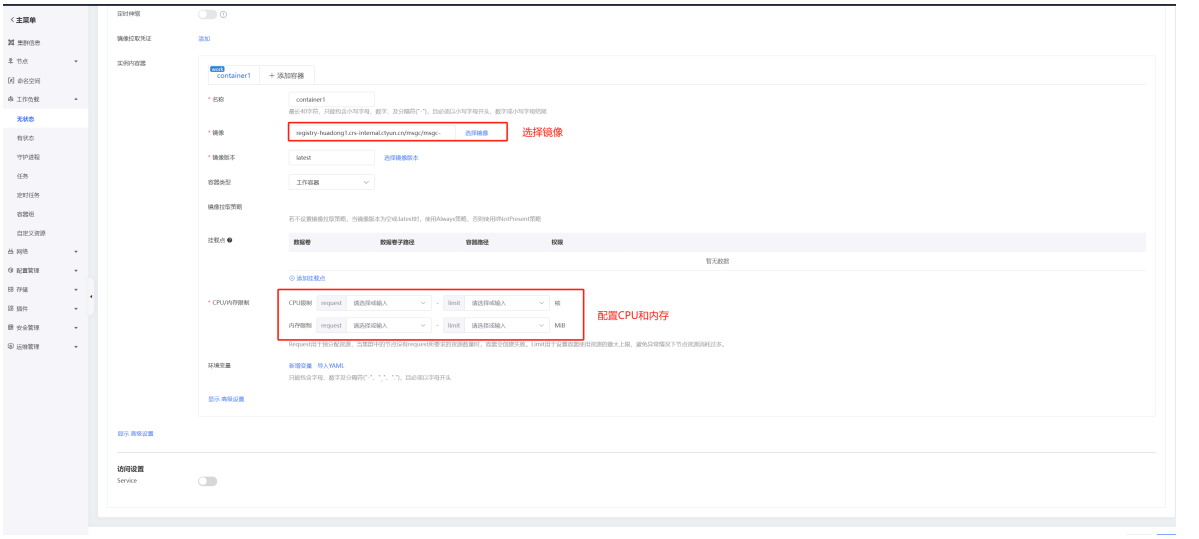
Demo部署

云容器引擎部署demo

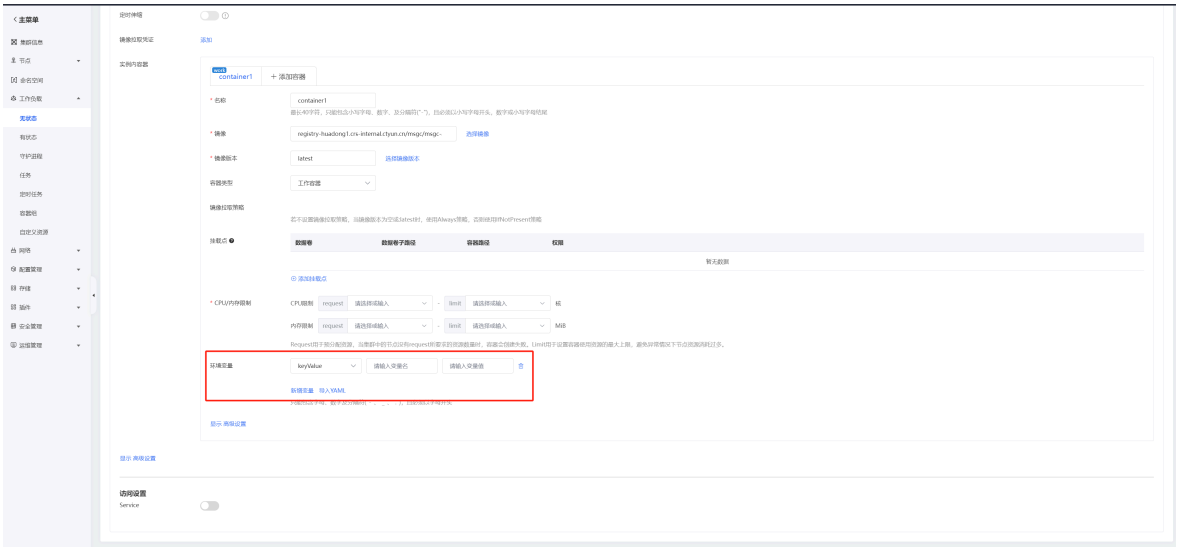
1. 新增部署应用，进入云容器引擎控制台，选择目标集群，点击工作负载->无状态->创建Deployment。



2. 镜像添加，在“镜像”中“选择镜像”然后选择上传的demo镜像image-test/{镜像名}。
3. 配置CPU/内存限制。



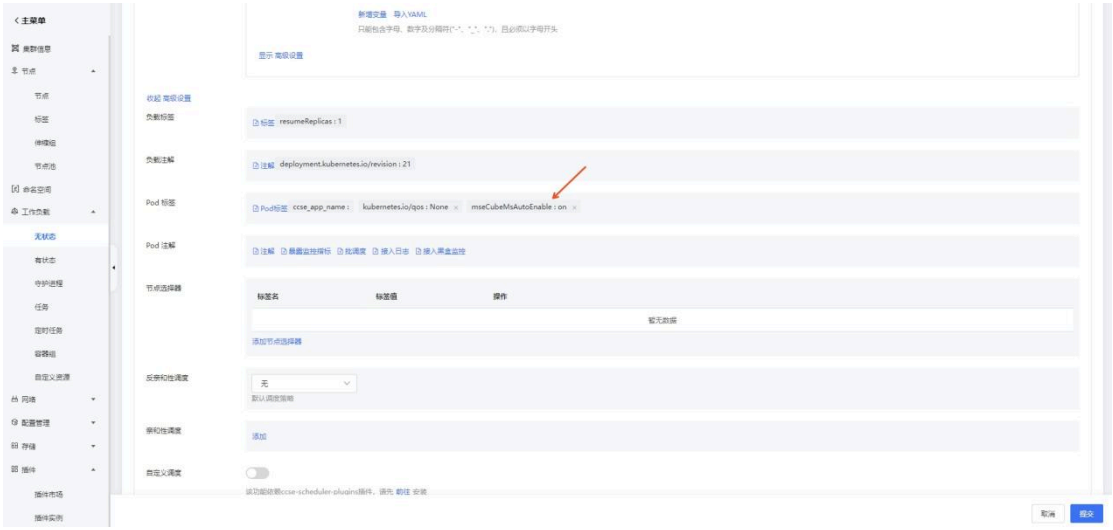
4. 配置环境。在“环境变量下”，新增变量添加环境配置 。



spring.cloud.nacos.discovery.server-addr：nacos服务地址
spring.cloud.nacos.discovery.username：nacos用户名
spring.cloud.nacos.discovery.password：nacos密码
spring.cloud.nacos.discovery.namespace：nacos命名空间

为应用开启微服务治理能力

1. 在Deployment编辑页，点击“显示高级设置”，新增“Pod标签”：mseCubeMsAutoEnable:on。



标签名	标签值
mseCubeMsAutoEnable	on

2. 在发布应用时，配置指定环境变量，可指定注入微服务治理中心的应用名、命名空间和标签等信息。

环境变量配置如下：

标签名	标签值
MSE_APP_NAME	接入到微服务治理中心的应用名。
MSE_NAMESPACE（选填）	接入到微服务治理中心的命名空间，默认为：default。

3. 完成编辑后点击“提交”，发布到容器即可。

验证应用已经接入MSE

查看应用治理或者网关治理，确认您的应用已经接入到微服务治理中心。

如何在微服务治理中添加自定义接口实现流量防护

概述

应用通过Java探针接入微服务治理中心以后，您可结合msgc-flow-sdk对任意代码块进行埋点。埋点后可在微服务治理中心控制台查看自定义埋点接口的监控数据，也可针对自定义埋点进行流量防护。

限制条件

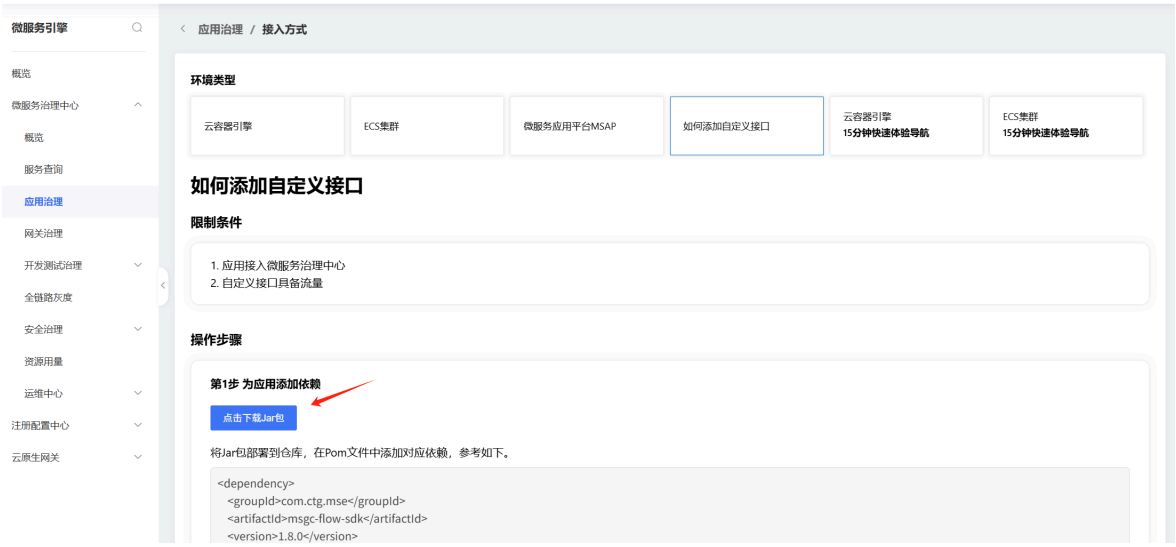
1.应用已接入微服务治理中心。

2.自定义埋点接口具备流量。

操作步骤

第1步 为应用添加依赖

依赖下载路径：微服务治理中心控制台->应用治理->应用接入->如何添加自定义接口->点击下载jar包。



将jar包部署到仓库，在Pom文件中添加对应依赖，参考如下：

```
<dependency>
   <groupId>com.ctg.mse</groupId>
   <artifactId>msgc-flow-sdk</artifactId>
   <version>1.8.0</version>
</dependency>
```

第2步 在应用工程中添加埋点

自定义方式埋点

```
SphU.entry(String name, EntryType trafficType, int batchCount, Object... args)
```

使用示例

```
Entry entry = null;
try {
   entry = SphU.entry("resourceName", EntryType.IN, 1, paramA);
   //业务逻辑
}catch (BlockException e){
   // 触发防护规则，在此处进行处理
}catch (Exception e){
   // 保证异常被SDK感知
   Tracer.traceEntry(e, entry);
   throw e;
}finally {
   // 务必保证enter成功后最终会exit，并且传入了对应的参数
   if (entry != null) {
      entry.exit();
   }
```

}

自定义埋点参数说明

参数名	说明
name	接口名称
trafficType	IN - 入口流量 、OUT - 出口流量、INTERNAL - 内部调用
batchCount	表示每次调用计数为多少，通常传 1
args	接口入参

结果验证

进入 应用治理 ，选择对应的应用卡片，在接口详情 > 自定义接口查看对应接口监控信息。

注册配置中心

实例管理

概述

注册配置中心支持可视化的实例管理，便于用户统一管理。在实例开通完毕后，您可以在控制台查看实例状态，网络、节点等信息，也可以在必要时进行扩容和退订操作。本节主要介绍如何在控制台查看、管理注册配置中心实例。

前提条件

1. 已开通微服务引擎MSE注册配置中心实例，参考章节：创建Nacos实例；
2. 注册配置中心实例状态正常；

查看实例

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行管理按钮均可跳转至实例基础信息页面，查看实例的基础信息和节点信息；

重启实例

MSE控制台页面支持重启实例和重启节点，可以根据业务需求进行选择。

实例滚动重启

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 基础信息右上角点击重启实例，即可重启整个实例，多节点实例会滚动重启各个节点；
5. 进行实例重启操作后，页面会跳转到实例列表页面，在实例列表页面该实例的状态为“变更中”，点击状态旁边的小图标可以查看重启流程细节；

单节点重启

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在下方实例节点部分，可以看到节点列表，点击目标节点对应行右侧的操作按钮，即可重启目标节点；

实例扩缩容

扩/缩容是指将实例资源进行升/降配以实现实例对业务容量支持能力的提升/降低。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例所在行的“扩容”按钮可跳转到实例扩容页面，根据该页面的扩容选项可以进行扩容操作；

4. 进行扩缩容操作后，页面会跳转到实例列表页面，在实例列表页面该实例的状态为“变更中”，点击状态旁边的小图标可以查看扩容流程细节；

续订

对于包年包月订购类型的实例，您可以点击续订按钮对实例进行续期。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例所在行的“续订”按钮可跳转到实例扩容页面，根据该页面的续订时长选项进行续订操作；

实例名称修改

注册配置中心支持修改实例的名称，便于用户管理。在实例开通完毕后，您可以在控制台修改实例名称，对应监控指标告警信息都会同步调整。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行"管理"按钮均可跳转至实例基础信息页面；
4. 在实例基础信息页面，点击实例名称后的编辑图标；
5. 在修改实例名称的弹框中，输入新实例名称并点击修改按钮；

退订实例

退订实例是指主动停止整个集群实例。当实例退订后，实例处于不可用的状态。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 实例列表找到目标实例对应行，最右侧操作栏点击退订，跳转至退订页面；
4. 退订页面点击确定后实例将被退订；

网络管理

安全组切换

概述

MSE注册配置中心支持切换实例的安全组配置，便于用户进行网络配置。在实例开通完毕后，您可以在控制台修改实例安全组。本节主要介绍如何切换注册配置中心实例的安全组。

前提条件

已开通微服务引擎MSE注册配置中心实例，参考章节：创建Nacos实例 、创建ZooKeeper引擎

开通MSE注册配置中心实例并且状态正常；

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行"管理"按钮均可跳转至实例基础信息页面；
4. 在实例基础信息页面，点击安全组属性后的编辑图标；
5. 在修改安全组的弹框中，选中要修改的安全组并点击修改按钮；

主子账号

概述

MSE注册配置中心已接入主子账号体系，可区分两种账号权限，实现主账号对子账号的数据权限管理与功能权限管理，支持系统策略和自定义策略的授权方式。本章介绍如何使用主子账号体系管理子账号权限。

背景

1. 主账号 ：用户在天翼云注册后自动创建，该账号对其所拥有的资源具有完全的访问权限，可以重置用户密码、分配用户权限等。如果需要多人共同使用天翼云资源，由于账号是付费主体为了确保账号安全，建议创建子用户来进行日常管理工作。
2. 子账号 ：主账号认证为企业账号后，在天翼云用户中心页面创建出来的账号。子账号的用户名、密码统一由主账号创建管理。子账号同样可以登录访问天翼云控制台，登录入口与主账号相同，受主账号赋予的权限限制。
3. 企业项目： 将云资源、企业成员按项目进行管理，通过企业项目将云资源、带有权限的用户组绑定到一起，用户使用项目内云资源的权限受用户组的授权限制。

注意

一个实例只能归属一个企业项目（可变更），一个子账号可以同时在多个企业项目中。

4. 策略： 是描述一组权限集的语言，它可以精确地描述被授权的资源集和操作集，通过策略，用户可以自由搭配需要授予的权限集。通过给用户组授予策略，用户组中的用户就能获得策略中定义的权限。
5. 系统策略： 系统预置的常用权限集，主要针对不同云服务的只读权限或管理员权限，比如对组件的只读权限、普通用户权限和管理员权限等等；系统策略只能用于授权，不能编辑和修改。
6. 数据权限： 看到的数据不一样。主账号看到所有实例，子账号只能看到所属项目中的实例。
7. 功能权限： 主账号可以进行所有控制台操作，子账号对单个组件实例拥有的操作权限由主账号授权。
8. 功能权限授权： 给子账号在企业项目A下增加一个策略，即代表该子账号对企业项目A下的实例拥有了策略中定义的权限，策略以外的操作会被禁止。

数据权限控制

数据权限的权限码为统一值instance-list，策略中包含instance-list的权限码才具备查看实例的权限。如果在用户组直接授权包含instance-list的策略，则该用户组的子用户能看到所有实例。如果在企业项目中的用户组授权包含instance-list的策略，则该用户组的子用户只能看到该企业项目下的实例。

操作步骤：

1. 进入IAM管理页面：

- 登录天翼云官网，鼠标悬浮至右上角个人信息，点击个人信息进入账号中心页面。
- 在账号中心页面中，点击统一认证服务进入IAM管理页面。

2. 创建用户组：

- 在IAM管理页面中，点击左侧菜单栏中用户组进入用户组管理页面。
- 在用户组管理页面中，点击创建用户组按钮，在弹出框中填写用户组名称与描述，点击确定即可。

3. 创建子用户：

- 在IAM管理页面中，点击左侧菜单栏中用户进入用户管理页面。
- 在用户管理页面中，点击创建用户按钮，进入创建页面。

- 在创建页面中，首先需要配置用户基本信息。填写用户名称、手机号、邮箱和密码等信息，点击下一步加入用户组。
- 在加入用户组页面，选择对应的用户组，点击添加按钮加入已选用户组，点击下一步即可。

4. 创建企业项目：

- 在IAM管理页面中，点击左侧菜单栏中企业项目进入企业项目管理页面。
- 在企业项目管理页面中，点击创建企业项目按钮，在弹出框中填写企业项目名称与描述，点击确定即可。

5. 在企业项目中添加用户组：

- 在企业项目管理页面中，点击企业项目的查看用户组按钮进入企业项目的用户组管理页面。
- 在企业项目的用户组管理页面中，点击设置用户组按钮，弹出设置用户组弹框。
- 在弹出框中，选择用户组再点击>按钮加入已选用户组，最后点击确定按钮即可。

6. 对企业项目中的用户组设置策略：

- 在企业项目管理页面中，点击企业项目的查看用户组按钮进入企业项目的用户组管理页面。
- 在企业项目的用户组管理页面中，点击用户组的设置策略按钮弹出设置策略弹框。
- 在弹出框中选择对应策略，点击>按钮加入已选策略，最后点击确定按钮即可。

功能权限控制

主账号对子账号的功能权限控制是通过给用户组授权策略实现的，策略包括系统策略和自定义策略。策略中可定义“允许”的操作和“拒绝”的操作，“拒绝”的优先级大于“允许”。

操作步骤：

1. 进入IAM管理页面：

- 登录天翼云官网，鼠标悬浮至右上角个人信息，点击“个人信息”进入账号中心页面。
- 在账号中心页面中，点击“统一认证服务”进入IAM管理页面。

2. 创建用户组：

- 在IAM管理页面中，点击左侧菜单栏中“用户组”进入用户组管理页面。
- 在用户组管理页面中，点击“创建用户组”按钮，在弹出框中填写用户组名称与描述，点击确定即可。

3. 创建子用户：

- 在IAM管理页面中，点击左侧菜单栏中“用户”进入用户管理页面。
- 在用户管理页面中，点击“创建用户”按钮，进入创建页面。
- 在创建页面中，首先需要配置用户基本信息。填写用户名称、手机号、邮箱和密码等信息，点击下一步加入用户组。
- 在加入用户组页面，选择对应的用户组，点击“添加”按钮加入已选用户组，点击下一步即可。

4. 授权：

- 在IAM管理页面中，点击左侧菜单栏中“用户组”进入用户组管理页面。
- 在用户组管理页面中，点击对应用户组的“授权”按钮，进入授权页面。
- 在授权页面中，勾选对应策略，点击下一步进入设置授权范围页面。
- 在设置授权范围页面中，选择授权范围，点击“确定”按钮就可完成授权。

迁移上云

概述

MSE注册配置中心具备将自建ZooKeeper、Nacos和Eureka等多种注册配置中心平滑迁移至MSE注册配置中心实例，本章介绍如何使用迁移上云功能实现平滑迁移。

前提条件

1. 已开通微服务引擎MSE，参考创建Nacos实例
2. 开通Nacos实例并且状态正常；
3. 自建注册中心与MSE Nacos实例在同一VPC子网内；

操作步骤

自建服务迁移至MSE Nacos

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，选择一个引擎类型为Nacos的实例，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 选择迁移上云> 概览页面，点击安装按钮安装迁移工具；
5. 安装完成后，打开迁移集群管理页面，点击新增集群按钮新增自建集群；
6. 在弹出框中输入集群名称、选择集群类型、填写用户名密码以及集群IP列表，点击新建按钮即可添加自建集群；
7. 打开同步任务管理页面，点击新增任务，输入任务名称、选择自建集群与目标命名空间、勾选需要同步的服务，最后点击新增按钮完成同步任务的新增；
8. 打开服务管理> 服务列表，选择目标命名空间，确认自建集群的服务是否迁移成功；

自建服务迁移至MSE ZooKeeper

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在实例列表页面，选择一个引擎类型为ZooKeeper的实例，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
3. 选择迁移上云> 概览页面，点击安装按钮安装迁移工具；
4. 安装完成后，打开迁移集群管理页面，点击新增集群按钮新增自建集群；
5. 在弹出框中输入集群名称、选择集群类型、填写集群IP列表，点击新建按钮即可添加自建集群；
6. 打开同步任务管理页面，点击新增任务，输入任务名称、选择需要迁移的自建集群，最后点击新增按钮完成同步任务的新增；
7. 打开数据管理> Znode管理菜单，查看对应数据节点是否迁移成功；

服务端负载均衡

前提条件

1. 已开通微服务引擎MSE
2. 开通MSE实例并且状态正常
3. 开通弹性负载均衡ELB

4. 设置绑定的ELB端口必须处于放行状态(可查看安全组策略确认)

操作步骤

绑定ELB

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 如果引擎类型为Nacos，点击绑定ELB端口，在弹出框中选择ELB类型后，选择与实例同一个VPC内网下的ELB实例，需要选择绑定两个端口，分别是服务主端口和Grpc端口，选择主端口后会自动偏移Grpc端口，无需手动输入GRPC端口。
5. 如果是非Nacos引擎，则不需要绑定两个端口，只需选择ELB后，选择需要绑定的主端口即可。
6. 建议端口使用范围在1024-49151之间。
7. 绑定提交后，会后台异步处理绑定过程，可在页面绑定列表刷新查看绑定状态，如果绑定失败，请确认是否存在已占用端口冲突。

解绑ELB

选择需要解绑的ELB绑定信息，点击解绑ELB按钮，如果是Nacos引擎类型，则会同时解绑主端口和GRPC端口。

使用方式

成功绑定ELB后，可以使用ELB绑定信息列表中的ELB入口地址+端口进行访问；访问流量会平摊到所有实例节点；当实例扩/缩容时，会自动添加/删除ELB入口地址映射的实例节点。

资源标签

概述

MSE注册配置中心已经支持资源标签功能，提供为资源添加标签以及根据标签筛选资源的功能，本章介绍如何使用资源标签功能。

添加标签

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 光标悬浮至对应实例的标签列图标，点击“添加”按钮，会显示弹出框。
4. 在弹出框中点击“添加标签”按钮，选择已有标签键值对或者创建自定义标签键值对，点击“确定”按钮即可添加对应标签。

批量绑定标签

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 选择对应实例，点击批量标签>绑定标签按钮。
4. 弹出框显示已选择的实例资源，选择对应标签或填写自定义标签，点击便签右侧“确定”按钮，将对应标签加入已选择标签列表。
5. 点击弹出框右下角的确定按钮完成批量绑定标签。

批量解绑标签

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 选择对应实例，点击批量标签>解绑标签按钮。
4. 弹出框显示已选择的实例资源，选择对应标签，点击解绑按钮即可批量解绑标签。

修改标签

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 光标悬浮至对应实例的标签列图标，点击编辑按钮，会显示弹出框。
4. 在弹出框中，修改对应的标签键值对，点击修改按钮即可完成修改。

解绑标签

解绑标签操作会解绑对应标签与实例的关联关系，但不会删除对应标签，在选择标签时依然可见该标签。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 光标悬浮至对应实例的标签列图标，点击编辑按钮，会显示弹出框。
4. 在弹出框中，点击解绑按钮即可解绑对应的标签键值对。

标签筛选

标签筛选功能可以根据选择的标签筛选实例资源，每次最多同时筛选5个标签。只要实例资源拥有已选标签中的一个即会被筛选出来。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 点击标签筛选按钮，点击对应的标签键值对可将标签加入已选标签，即可根据已选标签筛选出对应实例资源。

Nacos

版本特性

概述

MSE注册配置中心在开源Nacos的基础上研发了推送轨迹等新功能，提供了丰富的运维手段，保障高可用性。本文介绍 Nacos 引擎版本的功能特性。

版本特性

Nacos引擎类型分为集群版、单机版，所支持的功能特性如下：以下表格中，✓表示支持此功能，✕ 表示不支持此功能。

功能	支持情况
支持Nacos 1.X版本客户端	✓
HTTP OpenAPI	✓
注册中心	✓

功能	支持情况
配置中心	✓
监控分析	✓
告警管理	✓
配置加密	✓
支持Nacos 2.X版本客户端	✓
推空保护	✓
推送轨迹	✓
支持多AZ高可用部署	✓

引擎版本

Nacos引擎基于开源进行了优化和定制开发。开源核心版本目前为2.1.0，后续会随着开源版本一起更新。

开源核心版本	MSE优化版本	描述
2.1.0	2.1.0.1	修复反序列化漏洞。增加推送轨迹功能。增加黑白名单支持。增加推空保护功能，请参见推空保护。优化推送轨迹功能，大量推送时更加稳定。修复Spring Security身份认证绕过漏洞。

Nacos风险管理

概述

Nacos引擎系统支持风险管理功能，可在通过控制台风险管理页面点击一键风险检查按钮来触发，评估当前Nacos引擎的风险。检查结果信息主要包括Nacos引擎的规格、配置、服务、白名单等相关的风险情况，以及针对风险情况提出建议和处理方案，并将结果、评分、建议记录下来供查询。

前提条件

已创建Nacos引擎实例。具体操作请参见章节：创建Nacos实例。

健康度检查

1. 登录微服务引擎-MSE注册配置中心管理控制台，并在顶部菜单栏选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表 。
3. 在实例列表页面，单击目标实例名称或者实例ID，进入基础信息页面。
4. 在左侧导航栏，单击风险管理进入风险管理页面。在集群健康度区域，单击一键健康度检查进入异步扫描。点击检查后会提示提交检查操作成功。
5. 检查结果预计需要5~10分钟返回。检查结果显示在几圈健康度页面下方区域。左侧为检查到的风险项和对应的建议。右侧为检查历史记录。

Nacos SDK应用和限制

概述

MSE Nacos 适用于各种微服务业务系统和应用场景，既可以单独使用也可以与微服务云应用平台、云容器引擎、应用服务网格等组合使用。为了您服务的稳定性，需要注意一些限制。本文介绍MSE Nacos SDK的应用以及相关的限制因素。

SDK的应用

如果您使用的技术栈是Java，既可以通过开源的Nacos客户端 SDK，也可以通过集成Spring Cloud、Dubbo等框架集成Nacos客户端访问Nacos实例，实现服务注册发现和配置管理；如果您业务使用的技术栈是Go、C++、Python、Nodejs，也可以通过开源客户端或者相关框架，访问 Nacos实例。另外， Nacos作为微服务系统的核心组件之一，也可以与服务网格和微服务网关、服务治理等组件整合使用，为云原生应用开发者提供更强大的能力。

技术栈	原生SDK	框架（Spring Boot）	框架（Spring Cloud）
Java	Naocs提供Java SDK 连接实例。	Spring Boot框架的接入方案请参考章节：Nacos Spring Boot快速接入	Spring Cloud框架的接入方案请参考章节：如何在MSE上为Spring Cloud应用构建服务注册中心？

SDK的使用限制

Java

不推荐的版本	不推荐原因	解决方案
0.X ~ 1.3.X	版本陈旧，影响性能。	升级至1.4.3及以上版本。
小于1.2.0	该部分版本不支持username和password注入，即不支持鉴权。	升级至1.4.3及以上版本。
1.3.3	该版本客户端不支持在密码中使用特殊字符。	升级至1.4.3及以上版本。
1.4.0 ~ 1.4.2	1.4.2版本使用配置加解密功能时，getConfigAndSignListener接口查询加密配置时返回内容为明文。	升级至1.4.3及以上版本。

Go

不推荐的版本	不推荐原因	解决方案
1.0.X ~ 1.1.X	客户端版本较低，存在较严重bug，可能导致服务不稳定等问题。	升级至2.3.2及以上版本。
2.0.0 ~ 2.1.0	当连接断开重连时，该客户端注册的实例不会自动回复，导致服务掉线。	升级至2.3.2及以上版本。
2.2.9	存在无法自动续期token问题。	升级至2.3.2及以上版本。

Spring Cloud Alibaba

不推荐的版本	不推荐原因	解决方案
小于2.2.1.RELEASE	不支持Nacos客户端鉴权参数注入。	升级至2.2.9.RELEASE及以上版本。

不推荐的版本	不推荐原因	解决方案
小于2.2.4.RELEASE	Nacos Client日志配置存在bug，业务日志与Nacos日志混合输出。	升级至2.2.9.RELEASE及以上版本。
2.2.4.RELEASE至2.2.6.RELEASE	默认依赖Nacos-Java-Client 1.4.2及以下版本。	升级至2.2.9.RELEASE及以上版本。

版本最佳实践

若您使用Spring Cloud框架集成Nacos客户端访问Nacos实例，推荐您使用如下的版本搭配。

Spring Cloud Alibaba	Spring Cloud	Spring Boot
2.2.9.RELEASE	Hoxton.SR12	2.3.12.RELEASE

管理命名空间

概述

Nacos实例中不同命名空间的服务是隔离的。通常命名空间可以用于不同环境的服务的隔离，建议为不同环境创建不用的命名空间。下面介绍注册配置中心的命名空间管理。

前提条件

已创建Nacos引擎实例，具体操作可参考创建Nacos实例。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心> 实例列表。
3. 在实例列表页面，单击目标实例ID 、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，单击命名空间，然后在页面左上角单击创建命名空间，弹出创建命名空间对话框，填写命名空间名称和描述信息，需满足下表参数规范，其中命名空间ID选填，如果不填则提交后会自动生成；
5. 在命名空间列表，查看命名空间已创建成功；
6. 命名空间创建成功后，也可以在命名空间列表的操作列，按需执行如下操作：
   a. 修改命名空间：点击编辑，根据提示修改命名空间名称和描述信息；
   b. 删除命名空间：单击删除，弹出框提示确认需要删除的命名空间名称，点击确定即可删除命名空间。

命名空间参数

参数	是否必填	说明
命名空间名称	必填	可自定义填写命名空间名称，支持非@、#、$、%、^、&和*，最短2个字符，最长128个字符。
命名空间ID	非必填	仅支持大小写字母、数字、短划线（-）和下划线（_），且不超过128个字符。命名空间ID必须唯一。

注意

1，MSE注册配置中心实例会默认创建一个public命名空间，这个命名空间不可修改和删除，一般仅做测试使用。当客户端未指定命名空间时，会默认使用public命名空间。我们建议生产环境业务不要使用public命名空间或者不指定命名空间，尽可能使用独立的命名空间，以达到权限管理和资源隔离的目的；

2，当业务客户端创建配置或者注册服务到一个不存在的命名空间，服务端不会响应错误，也不会自动创建命名空间，但是无法在控制台上查看到创建的服务实例或者配置，造成疑惑，因此业务侧客户端应谨慎配置已经存在的命名空间。

设置黑白名单

概述

在实例开通完毕后，可以通过设置Nacos实例的黑白名单，以此来限制一定范围内的IP地址（可以单个或者多个，可以是IP或者是IP段）访问实例。在设置白名单时，需要获取访问设备的IP地址，确保配置白名单后，设备能够正常访问Nacos。在设置黑名单时，可以禁止部分IP段的设备访问Nacos。当白名单和黑名单配置存在冲突时，以黑名单优先，即在冲突时，设备是无法访问Nacos实例集群的。

前提条件

1. 已开通微服务引擎MSE，参考章节：创建Nacos实例
2. 已开通Nacos实例并且状态正常

注意

- MSE注册配置中心默认不支持公网访问。
- 黑白名单认证时通过访问的请求头部中获取客户端IP，可能与公网地址不同，请确保配置的IP与请求携带的IP相同。
- 即使配置了白名单,仍然需要认证才能访问,目前实例默认没有放开免密访问.如果需要开启,可以在参数管理中通过修改authEnabled参数并重启集群来开启，这将会产生被他人免密使用的风险，请谨慎开启。
- IP地址格式：X.X.X.X/X，斜杠后为掩码。若IP地址设置为127.0.0.1/32，表示禁止所有地址的访问。多个公网IP地址或地址段，每个地址或地址段之间用英文半角逗号（,）分隔。子网掩码范围为1~32。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在菜单中，选择权限控制-黑白名单管理，点击黑白名单配置右侧的编辑图标 。
5. 在白名单配置中，输入允许访问的IP地址段，并点击确定。

- 如果白名单配置内容为空，表示所有地址均可访问Nacos实例。
- 如果填写的IP地址加掩码，表示允许所设置的IP地址段访问实例。

6. 在黑名单配置中，输入禁止访问的IP地址段，并点击确定。

- 如果黑名单配置内容为空，表示所有地址均可访问Nacos实例。
- 如果填写的IP地址加掩码，表示禁止所设置的IP地址段访问实例。

管理服务

概述

在实例开通成功后，可以将业务服务注册到Nacos实例，通过MSE注册配置中心控制台可以对Nacos实例上的服务及服务实例进行管理。本节介绍如何管理服务和服务实例。

前提条件

1. 已开通微服务引擎MSE注册配中心Nacos实例，参考章节：创建Nacos实例
2. Nacos实例并且状态正常；
3. 客户端已成功注册服务；

注意

推荐使用的Nacos客户端版本号为2.1.0及以上版本。

查询服务

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，选择您要操作的Nacos实例，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在基础信息页面，点击服务管理>服务列表，下拉框选择命名空间，查看当前命名空间下的服务列表。若需要查看指定服务下的服务实例列表，可单击右侧操作中的“查看”；
5. 调整左上角筛选框筛选类型为“服务名称”或“分组名称”，可按条件筛选服务，支持模糊查询；
6. 点击上方“显示空服务”按钮，可开启或关闭显示空服务开关，关闭后，空服务将无法被查出，空服务指的是不含有任何实例的服务。

创建服务

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，选择您要操作的Nacos实例，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在基础信息页面，点击服务管理>服务列表，选择命名空间，查看当前命名空间下的服务列表；
5. 点击左上角的“创建服务”按钮，在弹窗中分别填写服务名称、分组名称、实例来源、实例列表和保护阈值为自定义值，点击“确定”，即可创建服务。

说明

实例来源说明：

1，控制台注册：选择该类型，将会创建永久服务。服务端永不主动驱逐永久服务，仅当外部调用服务注销时驱逐。服务端默认对永久服务进行TCP主动探测；

2，SDK注册：选择该类型，将会创建临时服务。当心跳超时时，或外部调用服务注销时，服务端将驱逐临时服务。临时服务通常通过客户端心跳进行维续；

修改服务保护阈值

健康保护阈值为Nacos服务元数据中的一个属性，当该注册服务下的健康注册实例数 / 该注册服务下的总注册实例数的比值 <= 健康保护阈值时，表明此时健康实例比例过低，为避免全部流量将剩余健康实例打垮，Nacos服

务端将进入推送保护状态，此时Nacos引擎会推送该服务下的所有服务实例到服务订阅者（而不是默认行为：只推送健康服务实例）；您可以在控制台服务详情页面修改服务保护阈值。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在基础信息页面，点击服务管理> 服务列表，选择命名空间，查看当前命名空间下注册的服务列表。
5. 找到目标服务所在行，点击“查看”按钮，进入服务详情页面，进一步点击“保护阈值”旁边的编辑按钮，设置具体的值，有效范围0~1之间的小数；

删除服务

您可以在控制台服务管理服务列表页面删除服务，只有当服务中的实例数量为0时才允许删除。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在基础信息页面，点击服务管理> 服务列表，选择命名空间，查看当前命名空间下注册的服务列表；
5. 找到目标服务所在行，确认提供者数量为0，点击右侧删除按钮，弹出框点击确定，即可删除服务；

查看服务详情

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在基础信息页面，点击服务管理> 服务列表，选择命名空间，查看当前命名空间下注册的服务列表；
5. 找到目标服务所在行，点击“查看”按钮查看服务详情。服务详情主要包括应用实例集群，健康检查类型，以及实例信息。单击集群列表左侧的“>” 图标，可以展开该集群下的服务的实例。服务实例以集群的维度展示；

实例列表主要包括如下信息：

字段	说明
IP	实例IP
PORT	实例端口
健康状态	实例是否处于健康状态(正常/异常)
临时实例	临时实例，需要客户端主动保持心跳，默认5秒一次，连续三次心跳异常则标记为不健康，30秒后会被自动清理；持久化实例，由服务端探测服务是否正常，不会自动清理。
权重	取值范围为[0,10000]，默认为1。
心跳间隔	心跳间隔时间，默认5秒。
元数据	包括实例描述信息，如容灾策略、负载均衡策略、各种自定义标签 (label)等。

6. 服务如果为空，即没有注册的实例或者实例以全部被清理掉，Nacos会自动清理服务；
7. 在服务详情页面，点击“订阅者”栏目，可查看当前服的订阅者列表。

注意

此处仅对2.x版本客户端订阅者的信息进行展示

操作服务实例

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在基础信息页面，点击服务管理> 服务列表，选择命名空间，查看当前命名空间下注册的服务列表；
5. 找到目标服务所在行，点击查看按钮查看服务详情。单击集群列表左侧的“>” 图标，可以展开该集群下的服务实例；
6. 点击“创建实例”按钮，在弹出的窗口中依次自定义填写集群名称、IP、端口、上下线状态和权重后，点击确定，即可在指定服务、指定集群下创建实例；
7. 点击右侧操作列权重按钮，可以修改服务实例的权重，权重取值范围为[0-10000]，含义为万分之N，数值与权重成正比；
8. 点击上线（下线）按钮，可以将实例状态改为上线或者下线状态（点击上线或下线按钮后请点击实例集群列表行的刷新按钮刷新状态）；
9. 点击删除按钮，可以将实例元数据删除；

推送轨迹

概述

推送轨迹是指客户端注册了Listener监听Nacos的配置或者服务，当服务端的配置或服务发生变化时，主动推送给客户端，轨迹就是记录推送过程中的相关情况，包括触发时间、客户端、服务或者配置、推送耗时等信息。通过查询推送轨迹信息，可以清晰的确认服务或配置的推送情况，有利于提高问题排查效率。本节介绍推送轨迹的具体功能以及如何在控制台查看推送轨迹数据。

前提条件

1. 已开通微服务引擎MSE，参考章节：创建Nacos实例
2. 已开通Nacos实例并且状态正常
3. 已注册服务

注册中心推送轨迹

注册中心推送轨迹数据记录了服务推送的信息。当您在使用Nacos作为注册中心监听注册的服务后，当服务发生变化，客户端却没有收到变更推送时，就可以通过推送轨迹数据来确认时推送的动作没有触发、推送失败抑或是客户端返回了错误信息。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在基础信息页面，点击服务管理> 服务列表，选择命名空间，查看当前Nacos注册的服务列表。
5. 找到目标服务所在行，点击推送轨迹按钮，可以快速查看该服务的推送轨迹。也可以直接点击服务管理> 推送轨迹，然后选择命名空间，选择服务和分组，查看对应服务的推送轨迹数据。

6. 在推送轨迹页面，还可以根据需要选择查询维度：服务或者IP。服务维度查询需要输入或选择分组和服务名称，选择时间或者自定义时间，点击查询。IP维度查询客户端收到的全部推送信息,需要输入客户端IP作为查询参数.

配置中心推送轨迹

配置中心推送轨迹数据记录了配置推送的信息。当您在使用Nacos配置后，当配置发生变化，某台客户端配置却没有生效等场景，就可以通过配置推送轨迹数据来辅助定位问题。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在基础信息页面，点击配置管理>配置列表，选择命名空间，查看当前配置。
5. 找到目标配置所在行，点击推送轨迹按钮，可以快速查看该配置服务的推送轨迹。也可以直接点击配置管理> 推送轨迹，然后选择命名空间，选择Data ID和分组，查看对应配置的推送轨迹数据。
6. 在推送轨迹页面，还可以根据需要选择查询维度：配置或则IP。配置维度查询需要输入或选择分组和Data Id，选择时间或者自定义时间，点击查询。IP维度查询客户端收到的全部推送信息,需要输入客户端IP作为查询参数。

鼠标悬停在变更时间右侧的信息按钮上面，还可以看到变更事件的详细信息。例如：5.X.X.42在2024-01-04 11:37:52发起部署发布[dataId=-provider.properties,group =DEFAULT_GROUP],类型为普通配置。另外，点击配置维度查询按钮可以切换到配置维度查询当前配置的推送情况。

配置管理

概述

本章节主要介绍配置相关的管理操作，主要包括创建配置、同步配置、管理配置、查看历史版本、监听查询、查询推送轨迹、配置加密、配置灰度发布等。

前提条件

1. 已开通微服务引擎MSE，参考章节：创建Nacos实例；
2. 已开通Nacos实例并且状态正常。

管理配置

MSE 注册配置中心Nacos 提供配置的增删改改查操作。您可以通过控制台页面管理配置，发布之后，可以动态生效。

查询配置

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面；
3. 在基础信息页面，左侧菜单点击配置管理>配置列表，选择命名空间，查看当前配置；
4. 在配置列表页面可以根据Data ID、Group 、归属应用和标签维度筛选，然后点击查询；
5. 点击配置项的“查看”按钮，可以查看当前配置的属性和内容信息；

创建配置

配置就是将应用中的参数、变量等从具体的代码逻辑中提取出来，集中保存到一个文件中。这样在需要变更时只需要修改配置文件即可。Nacos可以集中托管配置文件，客户端监听配置，当配置变更时，自动推送至应用客户端。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在基础信息页面，点击配置管理>配置列表，选择命名空间，查看当前配置；
5. 配置列表页面左上角点击创建配置，在弹出框中填写配置的DataID、分组、配置格式、配置内容等信息，点击发布即可创建配置，成功后显示在配置列表；

字段	说明
命名空间	不可编辑，在创建配置前，在配置列表页面下拉框选择。
Data ID	配置ID。配置的标识。建议按照业务规范命名。
Group	分组。可以通过分组对配置进行细粒度分类。
数据加密	配置数据是否加密。加密配置的DataId会自动加上cipher-前缀。详情请参见配置加密。
配置格式	配置内容的格式，默认为text，支持text、json、properties、yaml、xml、html格式。
配置内容	配置文件的内容。配置的内容常见建议不要大于100KB，若配置内容过大，可拆成多个小配置。
描述	配置的描述信息。
应用	配置归属应用的名称。
标签	配置的标签。

编辑配置

1. 在配置列表页面，点击编辑按钮，进入配置详情页面；
2. 在配置详情页面，可以查看配置的具体信息；
3. 相较于创建配置的页面，编辑配置页面额外提供了Beta发布和配置内容对比的高级特性；
4. 在创建Beta配置时，需要同时配置beta发布的IP，多个配置可使用英文逗号分隔；
5. 当创建一个Beta配置后，点击编辑配置，会出现两个tab， 一个tab为发布beta之前的正式配置，另一个tab为beta配置，在Beta这个tab可以选择将beta发布为正式配置，或者将beta配置停止，停止后配置内容恢复到创建beta之前的正式内容；
6. 当修改完配置，点击发布按钮时，会弹出一个对比框，对比配置内容变化的地方，供用户参考，降低误操作的可能性。

删除配置

1. 在配置列表页面，点击目标配置行操作列的删除按钮。
2. 在删除配置的对话框中确认需要删除的配置后点击确定。

同步配置

当业务部署在多个环境或者业务环境迁移时，可能会需要将配置同步到相同Nacos实例的其他命名空间或者其他Nacos实例。MSE Nacos支持支持将配置批量同步至相同MSE Nacos实例的指定命名空间或MSE Nacos实例。

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在基础信息页面，点击配置管理>配置列表，选择命名空间，查看当前配置；
5. 点击需要同步的目标配置的单选按钮，选中配置，或者在表格表头的单选框选中批量选中；
6. 然后点击配置同步按钮，打开同步配置弹出框，选择目标实例（不选则默认本实例），选择目标命名空间，再选择相同配置的处理策略，可以修改同步之后的配置名称和分组名称；
7. 点击同步按钮，提交同步操作。返回的结果包括同步成功的数量、跳过以及覆盖的数量；

提示：

1. 相同配置策略分为三种。

策略	说明
终止导入	遇到相同Data ID和Group的配置，直接终止导入，后续的导入操作不再执行。
跳过	遇到相同Data ID和Group的配置直接跳过，不覆盖配置继续执行。
覆盖	遇到相同Data ID和Group的配置直接覆盖配置，继续执行。

2. 在同步配置的弹出框中，可以修改待同步的配置的Data ID和分组，修改后的值仅在目标实例命名空间下生效，原配置不会被改变。

查看历史版本

MSE 注册配置中心提供了配置历史查询功能。目前默认仅保存30天以内的变更记录。本文介绍如何查看配置历史版本。

查看历史版本

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击配置管理>配置列表，选择命名空间，查看当前配置。
4. 在左侧导航栏，选择历史版本。
5. 在历史版本页面选择命名空间、分组、和Data ID，点击搜索，即可查看配置的历史。

字段	说明
Data ID	配置的Data ID
Group	配置的Group
更新时间	配置更新发布的时间
所属应用	配置所属应用
操作类型	配置操作类型：插入、更新

6. 最右侧操作列提供查看和会滚的操作。点击查看可以查看所选的历史版本的配置的详细信息。
7. 点击回滚会弹出框对比所选历史版本和当前最新版本的对比信息。点击确认回滚，即可将配置回滚至历史版本。

监听查询

客户端注册监听配置，服务端在内存中维护一个监听客户端列表，当配置信息变更后，自动推送至客户端。

监听查询

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击配置管理> 监听查询，选择命名空间。
4. 可以根据需要选择查询维度：配置或则IP。配置维度查询需要输入或选择分组和Data ID，点击查询该配置推送到机器的状态。
5. IP维度查询该机器监听的所有配置，需要填入IP作为参数查询。

配置透视

MSE 注册配置中心提供了配置透视功能。支持查看配置监听客户端的信息以及客户端与服务端配置内容的对比。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击配置管理> 监听查询，选择命名空间。
4. 可以根据需要选择查询维度：配置或则IP。配置维度查询需要输入或选择分组和Data ID，点击查询监听该配置的客户端信息，包括客户端IP、客户端当前配置的MD5和客户端当前Nacos版本等信息。
5. IP维度查询该机器监听的所有配置，需要填入IP作为参数查询。

查询推送轨迹

客户端监听配置，当配置变更时，服务端将变更推送至客户端。如果修改某个客户端配置未生效，则可以借助推送轨迹定位问题。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击配置管理> 推送轨迹，选择命名空间。
4. 找到目标配置所在行，点击推送轨迹按钮，可以快速查看该配置服务的推送轨迹。也可以直接点击配置管理> 推送轨迹，然后选择命名空间，选择Data ID和分组，查看对应配置的推送轨迹数据。
5. 在推送轨迹页面，还可以根据需要选择查询维度：配置或则IP。配置维度查询需要输入或选择分组和Data ID，选择时间或者自定义时间，点击查询。IP维度查询客户端收到的全部推送信息,需要输入客户端IP作为查询参数.

配置加密

MSE注册配置中心的配置文件一般都是以明文的格式存储，但是对于部分敏感数据可能需要加密存储，通过SPI插件机制实现配置加密和解密，从而减少数据泄露风险。

创建加密配置

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。

2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击配置管理> 配置列表，选择命名空间，点击创建配置。
4. 在创建配置面板，开启数据加密开关，选择加密算法，默认为AES-256加密算法。
5. 选择配置格式，填写配置内容，然后点击发布。
6. 加密配置默认dataId增加cipher-aes前缀，在MSE管理控制台查看加密配置时看到的解密后的明文。

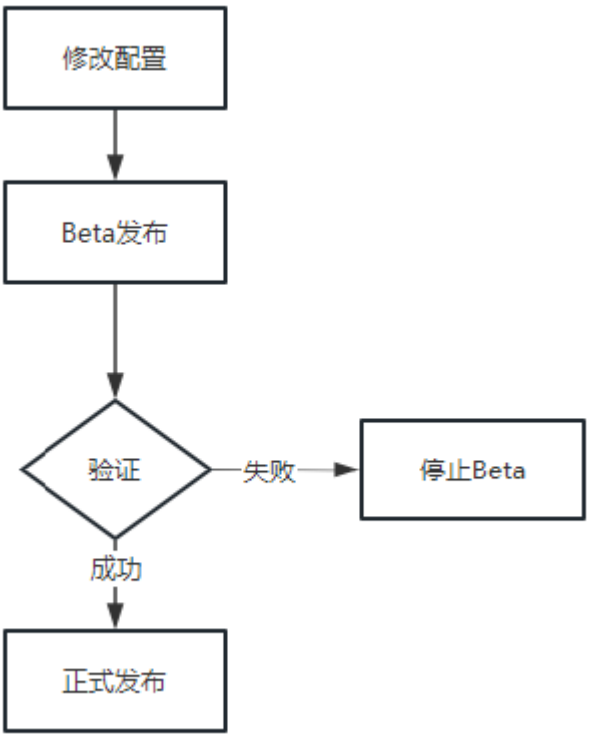
配置灰度发布

MSE 注册配置中心支持配置灰度发布功能，即创建一个beta配置，指定使用范围，经过验证后可以发布为正式配置或者回滚。缩小配置变更的影响范围。

背景

使用Nacos作为配置中心集中托管配置，当配置变更时自动推送给所有监听的应用客户端，非常方便。但同时也使得误操作影响的范围更大，一个误操作，可能导致众多的应用出现问题。

因此为了避免大范围出现问题，可以在进行配置变更时，使用Beta配置功能，指定一小部分范围的应用作为验证，仅指定范围的应用能收到更新推送。如果验证无误，则可以将Beta配置发布为正式配置，推送到全量客户端。如果验证存在问题，则可以停止Beta配置，配置将恢复到上一个正式配置。整个过程如下图所示：



创建Beta配置

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击配置管理> 配置列表，选择命名空间，查看配置列表。
4. 在目标配置的操作列，点击编辑，打开编辑面板，打开Beta发布开关。

5. 点击IP输入框，在输入框中填写灰度推送的IP地址。既可以填写IP，也可以配置IP段。
6. 修改完配置后，点击发布。

查看Beta配置

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击配置管理> 配置列表，选择命名空间，查看配置列表。
4. 在发布了Beta配置的目标配置的操作列点击编辑，打开编辑面板。
5. 在编辑配置面板的Beta 页中，查看Beta发布信息。
6. 在验证完毕后可以点击正式发布，将beta发布为正式配置，或者点击停止beta，取消Beta发布，配置内容恢复到创建beta之前的正式内容。

配置导入导出

MSE Nacos引擎提供配置数据的导出能力和其他Nacos和Apollo配置中心的数据导入能力。本文介绍如何使用Nacos导入和导出数据。

Nacos配置导出

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面；
3. 在基础信息页面，左侧菜单点击配置管理> 配置列表，选择命名空间，查看配置列表；
4. 选择需要导出的配置项，点击导出配置按钮，弹出框会显示本次待导出的配置项；
5. 点击开始导出按钮，即可下载导出的配置文件。

Nacos配置导入

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击配置管理> 配置列表，选择命名空间，查看配置列表。
4. 点击导入配置按钮，弹出窗口中“导入配置类型”选择“Nacos配置”，如需要导入到其他命名空间，请先在配置列表页面下拉框切换命名空间。
5. 选择对应的相同配置策略，点击上传文件按钮，上传需要导入的配置文件。当前支持上传的文件遵循开源Nacos规范，仅支持zip压缩包格式文件，可将其他Nacos配置导出的结果文件直接上传；
6. 点击确定导入按钮开始导入配置，稍等片刻即可完成导入。
7. 导入完成后，会在弹出框显示导入结果。

相同配置策略说明：

相同配置策略，即在配置导入的过程中，目标命名空间已存在相同配置的处理策略。三种策略的说明如下：

相同配置策略	说明
终止导入	终止本次导入
跳过导入	跳过已存在的配置
覆盖导入	覆盖已存在的配置

Apollo配置导入

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；

2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面；
3. 在基础信息页面，左侧菜单点击配置管理> 配置列表，选择命名空间，查看配置列表；
4. 点击导入配置按钮，弹出窗口中“导入配置类型”选择“Apollo配置”，如需要导入到其他命名空间，请先在配置列表页面下拉框切换命名空间；
5. 选择对应的相同配置策略（策略说明见前文），点击上传文件按钮，上传需要导入的配置文件，只能上传properties格式文件，可以同时选择多个文件且总大小不超过5MB；
6. 点击确定导入按钮开始导入配置，稍等片刻即可完成导入。
7. 导入完成后，会在弹出框显示导入结果。

Apollo的数据模型和MSE Nacos的数据模型存在不同之处，但可以进行一一对应，业务侧可根据技术栈对应调整；比如Apollo中环境（env）的概念可以对应Nacos中的命名空间（namespace），可以用来隔离不同环境。Apollo中集群（cluster）对应Nacos的分组，表示这些配置存在相关性。Apollo的命名空间则对应Naocs的DataID，对应具体的配置。

Apollo数据模型	Nacos数据模型
环境（env）	命名空间（namespace）
集群（cluster）	分组（group）
命名空间（namespace）	dataId

补充说明

以下是针对Apollo配置的导出方式说明，主要介绍Apollo原生控制台导出配置文件的两种操作路径

1. 以环境维度批量导出：
   a. 在Apollo原生控制台登录后进入“我的应用”页面，页面的右上方，选择管理员工具 > 配置导出导入；
   b. 在配置导出导入页面，勾选选择导出的环境，然后单击导出，导出的文件为zip压缩包；
   c. 解压压缩包进入导出的环境名称的目录就能看到导出的配置，文件名称格式为appId+集群+配置名称.properties。
2. 以单个配置维度导出：
   a. 在Apollo原生控制台登录后进入“我的应用”页面,单击目标应用名称，进入配置页面；
   b. 点击右上角齿轮图标>”导出Namespace” ，可以导出一个 {应用名称}.properties文件。

导入的Apollo配置文件命名方式与在Nacos中最终导入结果名称存在对应关系，因此导入前可以人工调整待导入文件，以达到预期目标

1. 导入文件名为“appId+group+config.properties”的文件表示将properties配置内容导入为Nacos实例的“appId”命名空间下的“group”分组下名称为“config”的配置项。如果“appId”这个命名空间不存在将会导入错误，需要在命名空间管理中创建命名空间ID为“appId”的命名空间然后再行导入；
2. 导入文件名为“group+config.properties”的文件表示将properties配置内容导入为Nacos默认public命名空间下的“group”分组下名称为“config”的配置项；
3. 导入文件名为“config.properties”的文件表示将properties配置内容导入为Nacos默认public命名空间下的默认的DEFAULT_GREOUP分组下名称为“config”的配置项。

配置加密

MSE注册配置中心的配置数据一般都是以明文的格式存储，但是对于部分敏感数据可能需要加密存储和传输，通过SPI插件机制实现配置加密和解密，从而减少数据泄露风险。

创建加密配置

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表，单击实例ID 或者"管理"按钮跳转至基础信息页面；
3. 在基础信息页面，左侧菜单点击配置管理> 配置列表，选择命名空间，点击创建配置；
4. 在创建配置面板，开启数据加密开关，选择加密算法，默认为AES-256加密算法；
5. 选择配置格式，填写配置内容，然后点击发布；
6. 加密配置默认dataId增加cipher-aes前缀，在MSE管理控制台查看加密配置时看到的是解密后的明文；

存储加密配置

加密配置一旦被创建，将在Nacos数据库中被加密存储。加密算法与在控制台创建该配置时所选用的加密算法相吻合。

传输加密配置

若要实现加密配置的加密传输，客户端需要同时满足如下几个条件：

1，选用Java客户端，且客户端Java-sdk版本大于等于2.1.0；

2，下载天翼云MSE客户端配置加密插件（点击下载);

3，业务侧配置maven依赖，用户可将插件依赖包先安装到本地maven仓库，在业务工程maven依赖中引入nacos-aes-encryption-plugin和nacos-encryption-plugin依赖；

```
<dependency>
  <groupId>com.ctg.mse</groupId>
  <artifactId>nacos-encryption-plugin</artifactId>
  <version>1.0.0</version>
</dependency>

<dependency>
  <groupId>com.ctg.mse</groupId>
  <artifactId>nacos-aes-encryption-plugin</artifactId>
  <version>1.0.0</version>
</dependency>
```

4，在nacos-client下排除依赖开源nacos-encryption-plugin包

```
<dependency>
  <groupId>com.alibaba.nacos</groupId>
  <artifactId>nacos-client</artifactId>
  <version>2.1.0</version>
  <exclusions>
    <exclusion>
      <groupId>com.alibaba.nacos</groupId>
      <artifactId>nacos-encryption-plugin</artifactId>
    </exclusion>
```

```
    </exclusions>
</dependency>
```

注意

当您使用Java-sdk1.x版本客户端时，加密配置将被以明文形式传输，无法实现传输加密；当您使用Java-sdk2.x版本客户端，但未同时满足上述两个条件时，加密配置将被以密文形式传输，但客户端在得到密文时，将无法对加密内容进行解密。

引擎监控

概述

MSE 注册配置中心实例创建成功后，可以通过监控查看实例运行情况。本节主要介绍查看Nacos监控分析的数据。

查看监控数据

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心> 实例列表，单击实例ID 或者管理按钮跳转至基础信息页面。
3. 在基础信息页面，左侧菜单点击监控分析，共有概览、注册中心、配置中心、JVM和系统资源五个监控维度，每个维度有数量不等的各项指标。
4. 关于各个监控页面的说明如下：

   默认监控周期为最近的30分钟，也可以选择1小时、6小时、或者自定义时间段。

   监控图标中部分图标时节点维度显示监控指标，故集群拥有多个节点时，部分图标可能会有多条颜色不同的数据曲线。

监控页面	说明
概览	包括节点数、配置数、服务数、长连接数以及读写的QPS。
注册中心监控	包括服务数、服务实例数、订阅者数、读写QPS和RT。
配置中心监控	包括配置数、监听者数，24小时读写次数，配置中心读写QPS和读写RT。
JVM监控	主要包括YoungGC 和FullGC次数和时间，年轻代、老年代和堆内存使用情况。
系统资源监控	包括系统CPU、内存、磁盘读写流量、网卡流量、5分钟系统负载情况。

告警管理

概述

通过实例的告警管理能力，您可以对实例状态进行实时监控，并在特定指标异常时，将消息以不同形式（短信、邮件、翼连）推送到相关负责人，以便及时感知问题、处理线上故障。

管理通知对象

在配置告警规则之前，您需要先添加通知对象。通知对象包括联系人、联系人组、翼连、WebHook集成四种形式。

1. 联系人：一个告警规则所通知的“个人”，通知渠道包括该“个人”的手机短信和个人邮箱。
2. 联系人组：多个联系人组成的逻辑团体。若告警通知到联系人组，将会通知联系人组下的每个联系人。
3. 翼连：通知到翼连群。
4. WebHook集成：通过调用预先指定的地址进行告警通知。

下面以最简单的联系人为例，演示如何添加通知对象。

1. 进入实例引擎控制台->告警管理->通知组页签，点击“新建联系人”按钮。
2. 在弹出的窗口中，填写联系人的对应信息，即可完成创建。

管理告警规则

告警规则决定了一次告警发生的阈值、通知的对象和渠道，以及通知的内容。完成联系人创建后，进入实例引擎控制台->告警管理->告警规则页签，可以管理您的告警规则。

下面将演示如何创建一个“Nacos引擎配置数量过多”告警规则并使其生效，触发告警。

1. 创建通知策略。进入实例引擎控制台->告警管理->通知策略页签，点击“创建通知策略”按钮，填写相关信息，并指定通知对象为上一步骤中创建的联系人，完成通知策略创建；
2. 点击“创建告警规则”按钮，在弹出的窗口中填写告警相关信息。其中，通知策略指定为步骤1中创建的通知策略，告警分组选择“Nacos引擎”，告警指标选择“配置数”，根据您的需求选择判断条件，告警等级等配置。
3. 完成告警规则创建后，可在告警规则列表中查询到该条目。告警规则创建完毕后默认启用，可通过右侧“操作”中的“停止”来使该规则失效。

告警规则生效后，可在实例引擎控制台->告警管理->告警事件历史页签中，对告警事件的当前状态进行追踪。

Nacos引擎访问权限

概述

MSE中Nacos引擎默认开启鉴权功能，限制未认证或者未授权的用户访问实例。本节介绍Nacos访问鉴权。

Nacos鉴权默认支持JWT认证，通过用户、密码、角色、权限共同作用来赋予用户权限。

前提条件

1. 已开通微服务引擎MSE
2. 已开通Nacos实例并且状态正常

用户管理

在控制台查看、创建、修改和删除用户。用户可以用来访问Nacos实例。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 选择权限控制> 用户管理页面，查看当前实例的用户列表。
5. 点击左上角创建用户，填入用户名、密码和确认密码，点击确定，即可创建。
6. 点击用户列表目标行操作列的编辑按钮，可以修改密码。
7. 点击用户列表目标行操作列的删除按钮，可以删除用户。

角色管理

在控制台查看、创建、删除角色。角色可以和用户绑定，这样用户就拥有了该角色的权限。

操作步骤

1. 进入权限控制> 角色管理页面，查看当前实例的用户列表。
2. 点击左上角添加角色，填入角色名，选择用户，即可创建。
3. 点击角色列表目标行操作列的删除按钮，可以删除角色。
4. 右上角输入框可以根据用户名搜索角色信息。

权限管理

在控制台可以添加删除角色权限。角色可以和用户绑定，这样用户就实现了用户权限的控制。

操作步骤

1. 进入权限控制> 权限管理页面，查看当前实例的授权情况。
2. 点击左上角添加权限，选择角色名，选择命名空间，选择只读、只写、和读写权限，点击确定，即可为角色添加权限。
3. 点击权限列表目标行操作列的删除按钮，可以删除权限。

ZooKeeper

ZooKeeper的使用场景和MSE ZooKeeper的优势

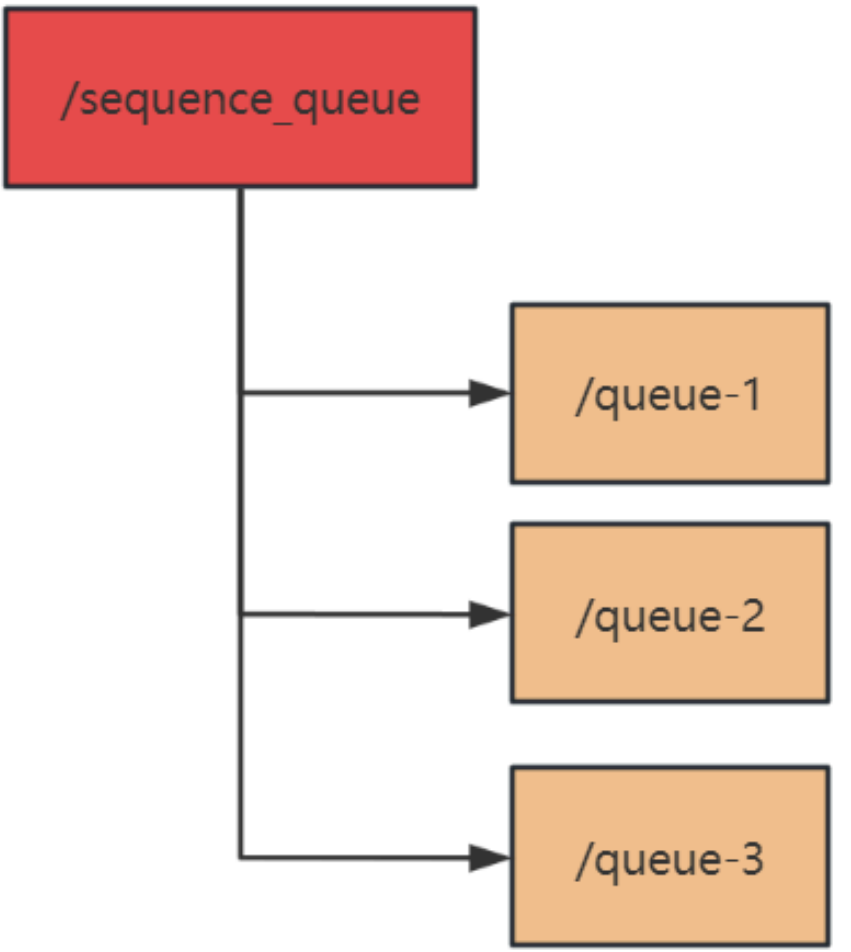
ZooKeeper 常用的技术应用场景如下所述。

场景一：分布式协调

分布式锁 ：在分布式环境中，程序都在独立的节点上，分布式锁是控制分布式系统之间同步访问共享资源的一种方式，分布式锁主要有如下2种类型：

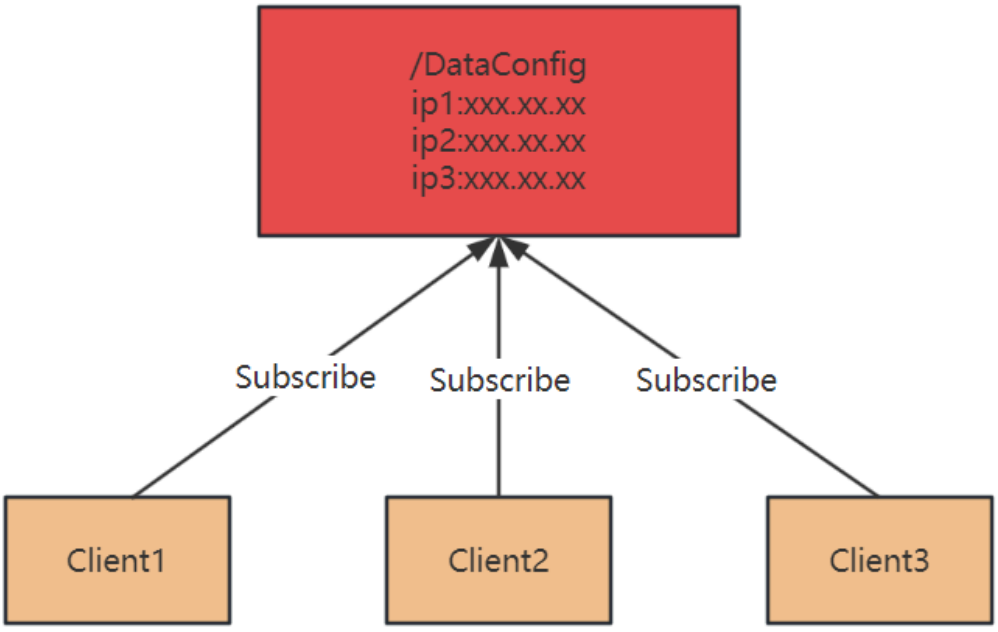
- 独占锁：主要实现原理是利用ZooKeeper在一个具体路径下每个进程创建一个有序的临时节点，每个进程会判断自己的节点是否序号最小的节点，如果是则获得锁，如果不是则创建一个监听等待前一个序号小的临时节点释放锁。
- 共享锁：共享锁可以支持多个进程同时获取这把锁进行读操作，但是如果某个进程要获取写操作的权限，那么在写操作之前是没有读操作的数据，并且该进程是第一个获取到写操作类型锁的。

分布式队列 ：队列功能可以利用ZooKeeper的有序节点，实现先进先出（First Input First Output，简称FIFO）的分布式队列，即先进入队列的先被消费，后加入队列的后被消费。在创建znode时开启sequence 和ephemeral模式，则被创建的节点结尾是一个递增的值，且不会重复。



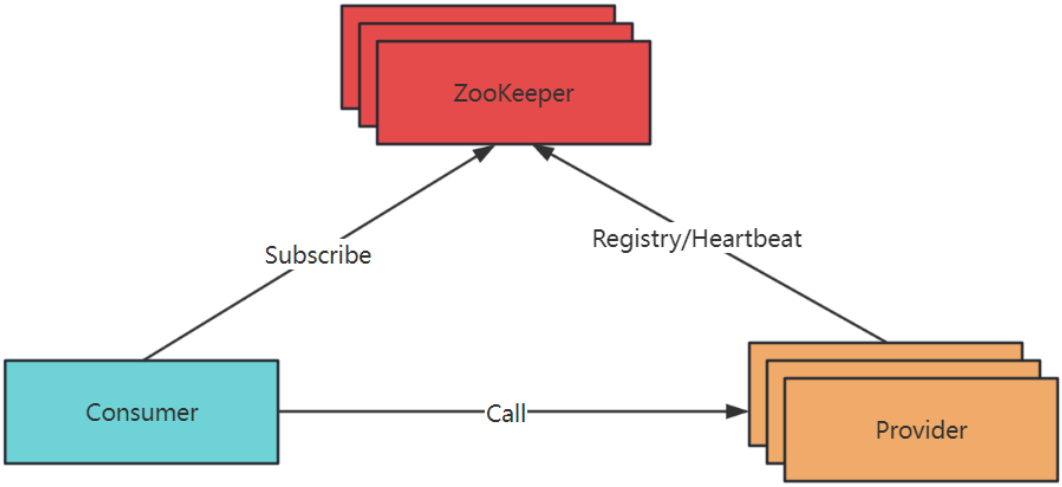
场景二：配置中心

运用ZooKeeper的存储模式，实现配置信息的集中管理和数据的动态更新，保证了配置数据的一致性和实时性。



场景三：微服务领域的注册中心场景

在微服务场景里，利用ZooKeeper的注册和监听功能。ZooKeeper可以用作Dubbo和Spring Cloud的注册中心。



MSE 提供的ZooKeeper企业级服务

MSE提供的ZooKeeper企业级服务，分为单机版和集群版两种，更多关于单机版和集群版的功能特性，请参见版本特性。

优势一：稳定高可用

多AZ部署：平均部署可用区，提高集群容灾能力。例如，当一个5节点的ZooKeeper集群，部署在3个可用区的时候，它应该是2/2/1的分布，任意一个可用区出现故障，不影响ZooKeeper的整体可用性。

高可用负载均衡：MSE ZooKeeper自动对用户请求做负载均衡，会把请求压力均衡到后端的节点上去，并且能保障请求会到监控正常的节点上面去。

数据安全：MSE ZooKeeper提供了快照备份能力，在集群出现意外状况时候，能快速重建恢复集群的数据，保障数据的安全。

优势二：可观测性增强

提供监控中心：MSE ZooKeeper提供了多达20余项常用的监控指标，包括业务指标和系统资源指标等，供您免费开启使用。

支持核心告警规则：支持配置告警规则，一旦发生指标异常，及时进行告警，可以满足日常运维使用。

优势三：性能提升

写入性能提升：ZooKeeper的写入性能和磁盘性能是强相关。MSE ZooKeeper提供了自研的超高IO磁盘供您选择，整个写入TPS性能可提高约25%。

参数性能调优：MSE ZooKeeper对JVM参数进行了调优，堆的参数设置会根据不同的规格进行动态调整。

优势四：丰富的控制台功能

对比项	MSE ZooKeeper	自建ZooKeeper
系统搭建成本	资源全托管免运维，故障节点自动摘除。	自行购买各种资源搭建系统，运维升级需投入精力，人工成本高。
易用性	提供可视化的数据管理，支持页面修改全局参数，重启生效。	不支持可视化。
高可用	多AZ部署，故障自动检测以及数据及时备份成快照。	需自行开发高可用保障体系。
性能	集成超高IOS磁盘，提高ZooKeeper读写性能。	需要自行调试。
监控告警	对集群状态，znode数，连接数，请求平均延时等指标监控。	没有监控体系，需要自己搭建。

版本特性

MSE ZooKeeper引擎分为单机版和集群版，下面介绍ZooKeeper引擎版本的功能特性。

1. MSE ZooKeeper单机版：单节点部署，不具备高可用能力，用于开发测试环境。
2. MSE ZooKeeper集群版：支持多节点部署，具备高可用能力，用于生产环境。
3. MSE ZooKeeper 3.8.1.x （单机和集群版）是基于社区开源ZooKeeper3.8.1推出的一个版本。

版本类型	MSE ZooKeeper版本	描述
集群版	3.8.1.1	支持通过导入快照（Snapshot）导入节点数据。增加子节点个数和数据大小限制，避免子节点过大影响性能。新增数据轨迹能力，支持通过节点路径和SessionID查询增删改查操作记录。支持扩容节点、引擎规格。新增监控中心，提供20余项常用监控指标项。合并社区分支更新优化项，BugFix。

节点扩容

概述

为了满足业务需求，可以选择扩容ZooKeeper引擎的节点数量，本文将介绍如何进行扩容。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

开通ZooKeeper实例并且状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、或者目标行“扩容”按钮可跳转到实例扩容页面。
4. 进入扩容页面后，选需要扩容的节点数，进行节点扩容时，要求新增后节点数量必须大于原节点数量，如原节点数量已是最大值，则不支持再进行节点扩容操作。
5. 选择对应要扩容的节点数量，提交订单，等待扩容完毕就能在实例详情看到对应的节点数。

数据管理

概述

ZooKeeper引擎的数据管理功能，提供了ZooKeeper节点树上数据节点的可视化管理能力，支持增删查改和ACL控制能力，同时对注册到ZooKeeper实例上的服务信息进行了提取和展示，使您更方便地查看注册服务相关的元数据信息。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

开通ZooKeeper实例并且状态正常。

操作步骤

ZNode管理

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，选择数据管理>ZNode管理。
5. 引擎中默认包含ZooKeeper节点，ZooKeeper节点中默认包含两个子节点，分别是config和quota。ZooKeeper节点及其子节点纳入保留节点，不允许用户进行写操作。

6. 点击创建节点按钮，可以创建节点，输入节点路径和节点信息，单个ZNode最大支持64K数据量。
7. 创建完成后，可以返回节点管理页面查看节点和数据信息。在节点管理页面，通过右键鼠标可以删除、增加子节点和同级节点。

ACL权限

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池 ；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在左侧导航栏，选择数据管理>ZNode管理；
5. 选择一个ZNode节点，在右下方会出现该节点的详细ACL权限信息；
6. 目前提供的权限修改有写权限（w），删除权限(d)，创建权限（c）,点击下拉框可以编辑权限；
7. 取消或新增对应权限后，页面对应ACL权限的示会相应变化；

注意

ACL适用于使用ZooKeeper客户端或者API访问实例集群的场景，MSE注册配置中心控制台自带超管权限，因此您可以在控制台上操作所有数据，请您慎重操作，同时控制台也提供了审计日志查询功能以帮助您进行溯源；

重置ZooKeeper集群数据

概述

在ZooKeeper使用过程中如果出现误用，会有大量冗余数据，导致集群内存溢出，这时可以使用集群节点重置功能，需注意重置节点属于不可逆操作，会将集群除了系统节点/ZooKeeper以外的所有节点重置。重置前需要进行确认。请谨慎操作。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

开通ZooKeeper实例并且状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，选择数据管理> Znode管理。
5. 在Znode管理页面，单击重置数据。
6. 在重置数据对话框，输入实例ID的后四位（实例ID可从基础信息页面获取），单击确定。

数据导入导出

概述

ZooKeeper提供通过快照（Snapshot）导入数据，同时支持导出快照（Snapshot）和事务日志（Trransaction log）文件。本文介绍如何使用ZooKeeper导入和导出数据。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

开通ZooKeeper实例并且状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，选择数据管理> Znode管理 > 数据导入导出。

数据导出

在Znode管理中，点击数据导出按钮，选择导出的数据类型，目前仅支持导出最新的快照文件和最新的3个事务日志文件。导出过程预计需要2到5分钟，导出完毕后可以在下载列表获取文件。

点击下载按钮，即可下载相关文件的压缩包。如果导出失败，请按照报错信息及提示重新操作导出。

数据导入

单击导入后，选中要导入的快照文件进行导入，目前仅支持Snapshot格式的快照文件导入，请注意导入文件格式。考虑容量安全问题，数据导入功能目前限制导入的文件大小为200M。

导入文件会导致集群短暂不可用，原本集群的数据将被重置，集群将重新选主。

数据轨迹

概述

在使用ZooKeeper的过程中可能会遇到Znode变更的情况，而在Znode变更之后，若相关的客户端没有收到变更事件的推送，会导致排查问题的难度加大，此时可以通过ZooKeeper数据轨迹功能提高问题的排查效率。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

开通ZooKeeper实例并且状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，选择数据管理> 数据轨迹。

变更记录

可以通过两个维度去查询数据轨迹：路径和SessionId，同时也支持查看近7天的数据轨迹数据。

变更时间：表示当前Znode变更事件发生的时间。

Path:表示该Znode的Path。

变更及推送事件：表示当前时间段内的Znode变更及推送事件。变更事件中显示本次变更的事件类型；推送事件中展现推送发生时间，及被推送的具体客户端的SessionId。

监控引擎

概述

MSE引擎开通成功后，我们提供可视化的引擎监控能力，包括业务监控、JVM监控、系统资源监控、TopN监控等细分项目监控功能。本文将介绍如何查看ZooKeeper的监控数据。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

开通ZooKeeper实例并且状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，选择监控分析> 业务监控/资源监控/JVM监控/TopN监控。

监控分析

根据需要观察的监控指标选择对应的页签，业务监控/资源监控/JVM监控/TopN监控等细分项目监控功能。

业务监控：支持监测ZooKeeper实例的Znode、Watch数、平均延迟、服务数和QPS等业务指标。

资源监控：支持实例CPU、内存、磁盘、网络等资源使用情况。

JVM监控：支持查看GC情况，堆内存使用率等。

TopN监控：支持查看tps，qps，变更次数等关键特征值排名靠前的ZNode。

默认的监控周期为30分钟，单击自定义时间，可以选择想要查看的时间周期。

单击右上角的刷新按钮，可以刷新当前选定时间的监控数据。

服务管理

概述

Zookeeper可支持用作服务的注册中心，我们提供了Dubbo服务和SpringCloud服务的查询管理能力，本文将介绍如何查看注册到ZooKeeper的服务。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

注册服务到相应ZooKeeper引擎。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行管理按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，选择数据管理> 服务管理。在服务管理的列表页面，可以看到注册到ZooKeeper上的服务。
5. 单击查看，可以查看相关服务的Provider的IP、端口、元数据、分组、版本等详细信息。
6. 点击订阅者，查看服务订阅者的IP和端口信息。

健康风险检测

概述

ZooKeeper引擎系统会定期扫描Zookeeper引擎的可用情况。如果您想随时查看ZooKeeper引擎的巡检指标，可通过控制台手动触发一键健康检查，来评估当前引擎系统的风险事项，健康检查通过Zxid溢出、集群平均响应时间、节点健康状态等维度进行评分，最终会针对风险情况提出建议和处理方案，并且会记录风险检测的历史。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

开通ZooKeeper实例并且状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行管理按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，选择风险管理> 点击一键健康检查。

设置引擎参数

概述

如果在使用ZooKeeper引擎时，您有特殊要求可以参考本节内容修改对应的参数，优化引擎性能。本文介绍如何修改查看Zookeeper的引擎参数。

前提条件

已开通微服务引擎MSE，参考章节：创建ZooKeeper引擎

开通ZooKeeper实例并且状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行管理按钮均可跳转至实例基础信息页面。
4. 在左侧导航栏，选择参数管理

修改参数

ZooKeeper引擎系统会使用一些默认参数进行启动，如果您在使用过程中，需要对引擎参数进行特殊定制，可以在参数管理页面进行修改，点击左上角编辑按钮->修改相应属性->点击保存并重启按钮，等待2-5分钟即可重启生效。

注意

- 修改参数重启会引起集群的短暂不可用，请在业务空闲时进行操作。
- ZooKeeper可以通过MinSessionTimeout和MaxSessionTimeout来限制客户端设置的超时时间。服务端会将小于MinSessionTimeout的客户端超时时间强制设置为MinSessionTimeout；将大于MaxSessionTimeout的客户端超时时间强制设置为MaxSessionTimeout， 例如， 当MinSessionTimeout为3000 ms ， MaxSessionTimeout为5000 ms时，如果客户端中设置sessiontimeout为分别为1000 ms和6000 ms，最终服务端和客户端协商的超时时间就是3000 ms和5000 ms。
- 我们不建议将MinSessionTimeout和MaxSessionTimeout 设置比默认值还小，这样有可能会导致客户端和服务端频繁发生连接超时的情况导致会话断开，从而导致不可预知的告警故障。

升级引擎版本

概述

本文将介绍如何对Zookeeper引擎实例进行版本升级。推荐您总是将引擎版本升级到当前可用的最新版本。具体的版本特性相见文档中的版本说明相关章节。对开通完成且实例状态正常的Zookeeper引擎，可以进行“实例升级”操作。

操作流程

1. 进入微服务引擎控制台-注册配置中心-实例列表页签，选择对应Zookeeper实例右侧操作中的更多下拉框，选中引擎版本升级。
2. 在弹出的引擎版本升级信息框中，在版本下拉框中选中您需要升级到的目标版本。选中目标版本后，下方将显示当前版本的新特性。确认好升级版本后，点击确认升级按钮可提交实例升级请求。对于拥有多个节点的实例，理论上升级对业务运行无损（特别需要考虑实例高位运行的情况，即已接入实例数和该规格支持的实例数的比率较高时，参考值70%），建议在业务低峰期进行操作。
3. 提交升级请求后，您可以在实例列表中筛选运行状态为“变更中”的实例，可以看到您实例的当前状态。
4. 升级完毕后，实例状态会恢复为“正常”，您可以在实例管理页面确认引擎的当前版本。
5. 若您需要回滚到升级之前的版本，可以在引擎版本升级点击后，弹窗中的升级历史记录中对应条目后方点击回滚，当前仅支持回滚最近一次升级记录。

告警管理

概述

通过实例的告警管理能力，您可以对实例状态进行实时监控，并在特定指标异常时，将消息以不同形式（短信、邮件、翼连）推送到相关负责人，以便及时感知问题、处理线上故障。

管理通知对象

在配置告警规则之前，您需要先添加通知对象。通知对象包括联系人、联系人组、翼连、WebHook集成四种形式。

- 联系人：一个告警规则所通知的“个人”，通知渠道包括该“个人”的手机短信和个人邮箱。
- 联系人组：多个联系人组成的逻辑团体。若告警通知到联系人组，将会通知联系人组下的每个联系人。
- 翼连：通知到翼连群。
- WebHook集成：通过调用预先指定的地址进行告警通知。

下面以最简单的联系人为例，演示如何添加通知对象。

1.进入实例引擎控制台->告警管理->通知组页签，点击新建联系人按钮。

2.在弹出的窗口中，填写联系人的对应信息，即可完成创建。

管理告警规则

告警规则决定了一次告警发生的阈值、通知的对象和渠道，以及通知的内容。完成联系人创建后，进入实例引擎控制台->告警管理->告警规则页签，可以管理您的告警规则。

下面将演示如何创建一个“ZooKeeper引擎延迟过大”告警并使其生效，触发告警。

1. 创建通知策略。进入实例引擎控制台->告警管理->通知策略页签，点击创建通知策略按钮，填写相关信息，并指定通知对象为上一步骤中创建的联系人，完成通知策略创建；

2. 点击创建告警规则按钮，在弹出的窗口中填写告警相关信息。其中，通知策略指定为步骤1中创建的通知策略，告警分组选择“ZooKeeper引擎”，告警指标选择“平均延迟”，根据您的需求选择合适的判断条件，告警等级。
3. 完成告警规则创建后，可在告警规则列表中查询到该条目。告警规则创建完毕后默认启用，可通过右侧操作中的停止来使该规则失效。
4. 告警规则生效后，可在实例引擎控制台->告警管理->告警事件历史页签中，对告警事件的当前状态进行追踪。

SASL认证

概述

默认情况下，ZooKeeper集群不会对客户端进行强制身份认证，任何客户端都可以访问ZooKeeper数据，存在安全隐患。目前，MSE ZooKeeper支持SASL身份认证，通过用户名和密码对客户端进行身份认证，提升Zookeeper数据的安全性。

前提条件

创建微服务引擎MSE，参考章节：创建ZooKeeper引擎；

开通引擎版本为3.8.1.9及以上的ZooKeeper实例，并且实例状态正常；

操作步骤

1.登录微服务引擎MSE注册配置中心管理控制台，选择资源池；

2.在左侧导航栏，选择注册配置中心 > 实例列表；

3.在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；

4.在左侧导航栏，选择权限管理> SASL认证；

5.在用户管理页面，单击创建用户。在创建用户面板中输入用户名、密码和确认密码，然后单击确定即可增加新的用户身份信息配置；

6.单击对应用户操作列中的复制配置，以下内容会被复制到粘贴板，密码需要手动补充，然后将其保存在客户端的任意文件中（如jaas.conf）；

```
Client {
      org.apache.zookeeper.server.auth.DigestLoginModule required
      username=test
      password="";
};
```

7.对于Java使用Zookeeper原生SDK或者Apache Curator框架的应用时，需要进行以下操作：

- 假设之前已经保存的配置文件路径为/path/to/jaas.conf，在Java应用启动的时候，指定以下系统属性。

-Djava.security.auth.login.config=/path/to/jaas.conf // 填写配置保存的文件的路径

- 重启客户端应用后，客户端会自动读取认证信息配置并向Zookeeper服务端进行身份认证。

8.客户端配置完成之后，在左侧导航栏，选择参数管理。在参数管理页面，将requireClientSASLAuth的值设置为true，即可强制要求客户端连接服务端时进行SASL身份认证；

9.用户创建成功后，可以在用户列表的操作列，按需执行如下操作；

- 重置密码：点击重置密码，在密码重置面板中输入用户名、新密码和确认密码。当用户密码发生变更后，以该用户身份连接服务端的客户端将会断开连接；
- 删除用户：单击删除，弹出框提示确认需要删除的用户名称，点击确认即可删除用户。当用户被删除后，以该用户身份连接服务端的客户端将会断开连接；

说明

- 将requireClientSASLAuth设置为true会导致未通过SASL身份认证的客户端无法请求Zookeeper服务端；
- Java客户端请确保依赖的ZooKeeper依赖版本为3.4及以上；
- Zookeeper服务端的引擎版本为3.8.19.及以上；

设置黑白名单

概述

本章节介绍MSE ZooKeeper引擎的黑白名单功能，在实例开通完毕后，可以通过设置ZooKeeper实例黑白名单来限制一定范围内的IP地址（可以单个或者多个，可以是IP或者是IP段）访问实例。当白名单和黑名单配置存在冲突时，以黑名单规则优先。

前提条件

1. 已开通微服务引擎MSE ZooKeeper引擎实例，参考章节：创建ZooKeeper实例；
2. 已开通ZooKeeper实例并且状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 点击右侧权限控制 > 黑白名单管理，点击页面“黑白名单配置”标题后面的编辑按钮，进入编辑模式；
5. 打开“白名单配置 ”后面的开关按钮，开启白名单，在白名单配置输入框中，输入允许访问的IP地址段，并点击确定；
   a. 如果白名单配置内容为空，表示所有地址均不可访问ZooKeeper实例；
   b. 如果填写了IP地址加掩码，表示允许所设置的IP地址段访问实例；
   c. 如果开关状态为关闭，则配置内容不生效
6. 打开“黑名单配置 ”后面的开关按钮，在黑名单配置输入框中，输入禁止访问的IP地址段，并点击确定；
   a. 如果黑名单配置内容为空，表示所有地址均可访问ZooKeeper实例；
   b. 如果填写的IP地址加掩码，表示禁止所设置的IP地址段访问实例；
   c. 如果开关状态为关闭，则配置内容不生效；

注意

- 如果您绑定了ELB，并使用ELB访问，则这部分流量不受黑白名单控制；可以去ELB控制台设置相关规则；
- 黑白名单认证时通过访问的请求头部中获取客户端IP，可能与公网地址不同，请确保配置的IP与请求携带的IP相同；
- 即使配置了白名单,仍然需要通过其他类型的认证才能访问，如ACL、SASL认证；
- IP地址格式支持IPv4及IPv6：X.X.X.X/X或X:X:X:X:X:X:X:X/X，斜杠后为掩码。若白名单设置为空，表示禁止所有地址的访问，请谨慎设置。多个公网IP地址或地址段，每个地址或地址段之间用英文半角逗号（,）分隔。

Eureka

版本特性

Eureka引擎目前发布了1.10.14.1版本，各版本支持的功能特性如下：

版本	功能特性
1.10.14.1	基于开源Eureka版本1.10.14扩展，除了支持基本的Eureka注册中心功能之外，还提供引擎监控、服务上下线、服务数据轨迹功能。说明：MSE注册配置中心Eureka引擎相比开源版本在一定程度上增强了可用性，使用注册配置中心Eureka可以更好地支撑生产业务。

建议您升级到当前可用的最新引擎版本，升级流程相见文档：引擎版本升级

服务管理

当您开通MSE Eureka实例后，MSE会对注册在其上的服务进行管理，您可以在服务管理界面对其进行管理端操作。

前提条件

1. 已开通MSE产品。
2. 已创建注册配置中心Eureka实例。

操作流程

进入实例管理页面后，在左侧选择页签服务管理可进入服务管理页面。服务管理页签包含两个子页签：服务列表和数据轨迹。

服务列表

展示当前注册到Eureka的所有服务，以及服务所包含实例的详细信息。您可以点击服务右侧操作中的详情常看对应服务的详细实例信息。详细信息中包含服务实例的主机名、访问地址、健康状态、注册时间等与客户端相关的基础信息，您可以在这里对注册到Eureka的服务的状态进行全局监控。

使用实例条目右侧操作中的下线或上线功能（当服务实例状态为“上线”时，展示“下线”按钮，反之，则展示“上线”按钮），可以在管理端设置该服务实例的对外可见状态。若服务实例状态为下线，则其他服务消费者将无法获取到该实例的相关信息，从而不会对该服务实例发起调用请求。

数据轨迹

展示针对某个服务的详细操作记录，包含上线、下线、请求、心跳、注册、注销事件。

点击服务右侧的数据轨迹界面，即可跳转到数据轨迹界面，通过在页面上方输入操作类型和所属服务来过滤操作记录，页面将会按照时间展示针对所属服务下某一实例这个操作对象在某一时间点发生了何种操作。

通过服务数据轨迹功能，您可以观察服务这个基本元素在Eureka中的完整生命周期流转过程，辅助排查注册中心故障。

引擎监控

MSE引擎创建成功后，您可以对引擎进行监控，包括客户端（使用该引擎的应用实例）连接数、引擎的TPS和QPS。

前提条件

已开通MSE产品。

已创建注册配置中心Eureka实例。

操作流程

进入实例管理页面后，在左侧选择页签监控分”可进入Eureka引擎监控页面。

Eureka监控指标一共分为四类：

- 概览：当前Eureka实例的核心业务指标
- 注册中心监控：当前Eureka实例的业务指标详情
- JVM监控：当前Eureka实例各节点机器Eureka进程的核心JVM指标
- 系统资源监控：当前Eureka实例各节点机器的系统资源指标

在实例监控页面，您可以通过各类指标观测实例的当前的运行状态以及历史运行状态，从而对注册中心问题进行及时追踪、诊断。

此外，Eureka引擎监控页面还包含了多项注册中心业务指标，如当前服务总数、健康服务数、当前实例数、注册注销等各类业务请求TPS、响应时间等。

您可以将鼠标悬停在图表上方任意位置，来查看当前时间点的指标精确值；也可以通过点击图表下方的曲线名称，对该曲线进行显示/隐藏。

升级引擎版本

前提条件

已开通MSE产品。

已创建注册配置中心Eureka实例。

操作流程

推荐您总是将引擎版本升级到当前可用的最新版本。具体的版本特性相见文档中的版本说明相关章节。对开通完成且实例状态正常的Eureka引擎，可以进行实例升级操作。具体操作流程如下：

1. 进入微服务引擎控制台-注册配置中心-实例列表页签，选择对应Eureka实例右侧操作中的更多下拉框，选中引擎版本升级。
2. 在弹出的引擎版本升级信息框中，在版本下拉框中选中您需要升级到的目标版本。选中目标版本后，下方将显示当前版本的新特性。确认好升级版本后，点击确认升级按钮可提交实例升级请求。对于拥有多个节点的

实例，理论上升级对业务运行无损（特别需要考虑实例高位运行的情况，即实例最大QPS/该规格支持的最大QPS的比率较高时，参考值70%），建议在业务低峰期进行操作。

3. 提交升级请求后，您可以在实例列表中筛选运行状态为“变更中”的实例，可以看到您实例的当前状态。
4. 升级完毕后，实例状态会恢复为正常，您可以在实例管理页面确认引擎的当前版本。
5. 若您需要回滚到升级之前的版本，可以在引擎版本升级点击后，弹窗中的升级历史记录中对应条目后方点击回滚，当前仅支持回滚最近一次升级记录。

告警管理

概述

通过实例的告警管理能力，您可以对实例状态进行实时监控，并在特定指标异常时，将消息以不同形式（短信、邮件、翼连）推送到相关负责人，以便及时感知问题、处理线上故障。

管理通知对象

在配置告警规则之前，您需要先添加通知对象。通知对象包括联系人、联系人组、翼连、WebHook集成四种形式。

- 联系人：一个告警规则所通知的“个人”，通知渠道包括该“个人”的手机短信和个人邮箱。
- 联系人组：多个联系人组成的逻辑团体。若告警通知到联系人组，将会通知联系人组下的每个联系人。
- 翼连：通知到翼连群。
- WebHook集成：通过调用预先指定的地址进行告警通知。

下面以最简单的联系人为例，演示如何添加通知对象。

1.进入实例引擎控制台->告警管理->通知组页签，点击“新建联系人”按钮。

2.在弹出的窗口中，填写联系人的对应信息，即可完成创建。

管理告警规则

告警规则决定了一次告警发生的阈值、通知的对象和渠道，以及通知的内容。完成联系人创建后，进入实例引擎控制台->告警管理->告警规则页签，可以管理您的告警规则。

下面将演示如何创建一个“Eureka引擎注册服务健康实例数量占比过少”多告警并使其生效，触发告警。

1. 创建通知策略。进入实例引擎控制台->告警管理->通知策略页签，点击“创建通知策略”按钮，填写相关信息，并指定通知对象为上一步骤中创建的联系人，完成通知策略创建；
2. 点击“创建告警规则”按钮，在弹出的窗口中填写告警相关信息。其中，通知策略指定为步骤1中创建的通知策略，告警分组选择“Eureka引擎”，告警指标选择“健康实例数量占比”，根据您的需求选择合适的判断条件，告警等级。
3. 完成告警规则创建后，可在告警规则列表中查询到该条目。告警规则创建完毕后默认启用，可通过右侧“操作”中的“停止”来使该规则失效。
4. 告警规则生效后，可在实例引擎控制台->告警管理->告警事件历史页签中，对告警事件的当前状态进行追踪。

设置翼连告警

除通过联系人渠道（短信、邮件）进行告警外，还可以将告警信息推送到翼连群。通过翼连群进行告警，成功率高、延迟低，推荐您使用这种方式进行告警。下面将演示如何创建一个“Eureka引擎注册服务健康实例数量占比过少”的翼连告警。

1. 进入实例引擎控制台->告警管理->通知组页签，在页面上方选择“翼连”类别，点击“新建翼连群”按钮，在弹出的窗口中填写相关信息，主要是翼连群号。
2. 在翼连创建群聊后，翼连群号可在下方位置查看并复制）。
3. 按照文档“管理告警规则”中的操作方式，创建告警规则，方法同前文“管理告警规则”。其中，在创建通知策略时，选择“翼连”类别，并选中刚才所创建的通知对象，完成绑定。其他步骤同“管理告警规则”中的规则创建方式。
4. 告警规则创建完成并生效后，当告警条件触发时，告警消息将会被推送到指定的翼连群。

数据备份与恢复

概述

MSE注册配置中心支持对实例集群内数据的备份和恢复能力。数据备份提供自动备份和手动备份两种模式；数据恢复将实例数据恢复到备份被创建时的数据快照，该操作恢复整个实例集群数据而非某个节点数据。

前提条件

已开通微服务引擎MSE注册配置中心，参考章节： 创建Nacos实例 、 创建ZooKeeper引擎

开通MSE注册配置中心实例并且状态正常。

操作步骤

数据备份

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行管理按钮均可跳转至实例基础信息页面；
4. 在左侧导航栏，点击备份管理；
5. 打开左上方自动备份按钮，设置备份间隔时间，即可按照时间间隔自动将数据进行备份；
6. 也支持点击立即备份按钮进行手动备份，手动备份后，可以点击右侧刷新按钮进行任务刷新，查看本次备份的状态；
7. 备份成功后，在下面备份列表页面可以看见已经成功备份的引擎数据；

数据恢复

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行管理按钮均可跳转至实例基础信息页面；
4. 在左侧导航栏，点击备份管理；
5. 查看已经成功备份的数据列表，选择一个需要恢复的数据备份，在操作列表里点击恢复操作；
6. 点击确认后开始恢复引擎，实例状态会进入“变更中”状态，实例会短暂的不可用；
7. 恢复操作成功后集群数据会被重置为备份被创建时的状态；

实例参数管理

概述

在使用MSE注册配置中心各类型引擎实例时，实例的参数采用最佳实践值，可以实现良好的运行性能和安全控制效果，通常不需要您变更。如果您有特殊要求可以参考本节内容修改对应的实例参数。本文介绍如何在MSE注册配置中心控制台配置实例参数。

前提条件

已开通微服务注册配置中心引擎实例，参考章节：创建Nacos实例 、创建ZooKeeper引擎

MSE注册配置中心实例状态正常。

操作步骤

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池；
2. 在左侧导航栏，选择注册配置中心 > 实例列表；
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面；
4. 在左侧导航栏，选择参数管理；
5. 在参数管理页面中，点击重启生效参数/运行时生效参数的编辑，修改相应的参数，点击保存即可；

注意：

- 修改参数时每次只能修改同一类型参数（重启生效参数或者运行时生效参数）；
- 修改重启生效参数需要重启注册中心实例，实例滚动重启，不影响业务可用性；

云原生网关

网关管理

新建云原生网关

概述

MSE云原生网关同时具备传统的流量网关和业务网关功能，提供全局性和独立业务域级别的流量管理策略，支持Nacos和K8s等多种服务发现方式，本文介绍如何新建云原生网关。

操作步骤

1. 进入微服务引擎MSE控制台
2. 在顶部菜单栏选择资源池
3. 单击左侧导航栏云原生网关 > 网关列表
4. 单击网关列表页面左上角创建网关按钮
5. 在云原生网关订购页选择您要订购的配置，并单击右下角下一步按钮

云原生网关订购配置说明

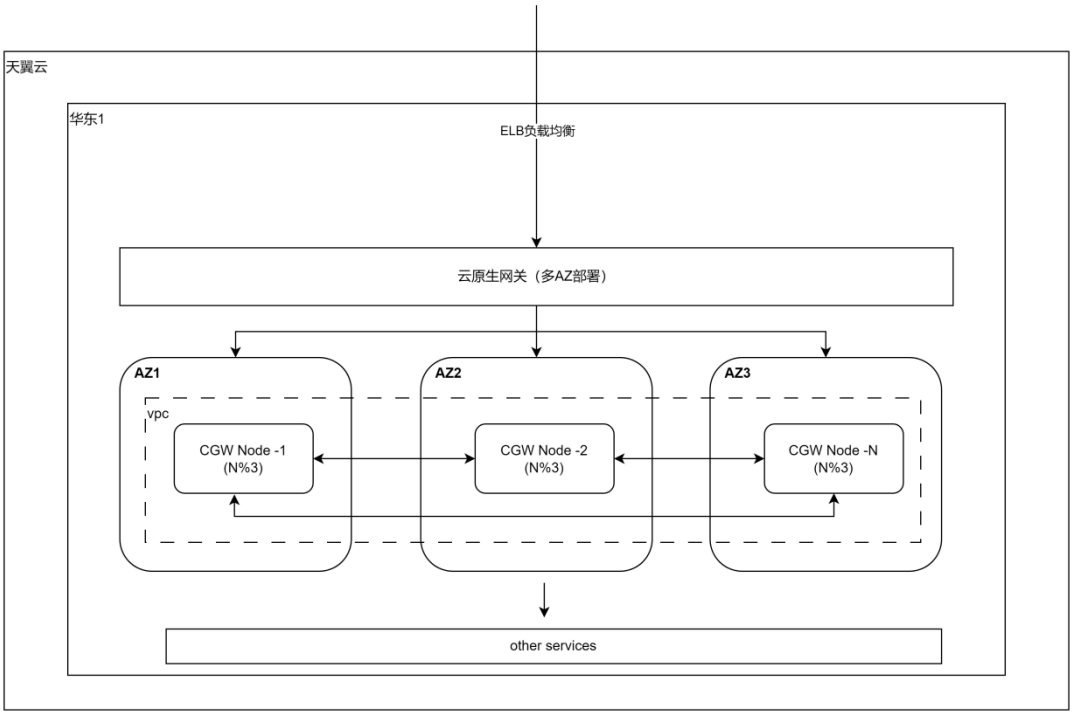
参数	描述
计费模式	支持包年包月和按需计费方式，费用说明请参照购买指南->计费说明。

参数	描述
购买时长	可以根据实际需求进行选择，支持1个月、2个月、3个月、4个月、5个月、6个月、1年。
自动续期	您可以选择开启自动续期，避免云原生网关到期后无法使用。
自动续期购买时长	当开启自动续期，可以选择续期时长，支持1个月、2个月、3个月、4个月、5个月、6个月、1年。
系列	支持基础版和集群版。基础版：只支持单可用区部署，且节点数量为1。集群版：自动将控制节点以及工作节点平均分配部署至各可用区，且节点数量不少于2。
实例规格	架构支持X86（通用、海光）、ARM（通用、鲲鹏、飞腾），具体以当前资源池提供的选项为准。实例规格支持2C4G 、4C8G 、8C16G 、16C32G规格，规格支撑能力说明请参照产品规格。
数据盘	支持超高IO ，最低大小50G，可根据实际需求自定义填写，填写值必须为50的倍数。
节点数量	基础版固定1个节点，无需填写；集群版您可以根据实际需求填写节点数量，节点数量不少于2。
部署方式	用户无需修改，基础版固定选中单可用区部署。集群版时，如果当前资源池支持多可用区且节点数量大于2时，则默认为多可用区部署，单可用区部署置灰，否则为单可用区部署。
可用区	基础版需要选择任意一个可用区进行部署。集群版会自动将控制节点以及工作节点平均分配部署至各可用区。
虚拟私有云	选择虚拟私有云，若您还没有虚拟私有云，请参照创建虚拟私有云。
所在子网	选择所在子网，若您还没有所在子网，请参照创建所在子网。
安全组	选择安全组，若您还没有可用安全组，请参照创建安全组。
实例名称	自定义实例名称，最长40字符，只能包含小写字母、数字及分隔符(-)，且必须以小写字母开头，数字或小写字母结尾。
企业项目	网关实例关联的企业项目，可以到IAM控制台创建企业项目。
启用监控指标	启用后，可在控制台观测分析中查看系统和API的流量、成功率、延迟等监控指标，若您还没有开通应用性能监控产品，可先点击提示链接前往开通。
启用链路追踪	可选择采集百分比启用，启用后，可在控制台观测分析中查看API请求的链路追踪信息，若您还没有开通应用性能监控产品，可先点击提示链接前往开通。
启用访问日志采集	启用后，可在控制台观测分析中查看访问日志，若您还没有开通云日志服务，可先点击提示链接前往开通。

基于多可用区部署的容灾能力

- 可用区是同一服务区内，电力和网络互相独立的地理区域，一般是一个独立的物理机房，这样可以保证可用区的独立性。
- 区域（Region）指物理的数据中心。每个区域完全独立，这样可以实现最大程度的容错能力和稳定性。资源创建成功后不能更换区域。一个区域内有多个可用区，一个可用区发生故障后不会影响同一区域内下的其它可用区。

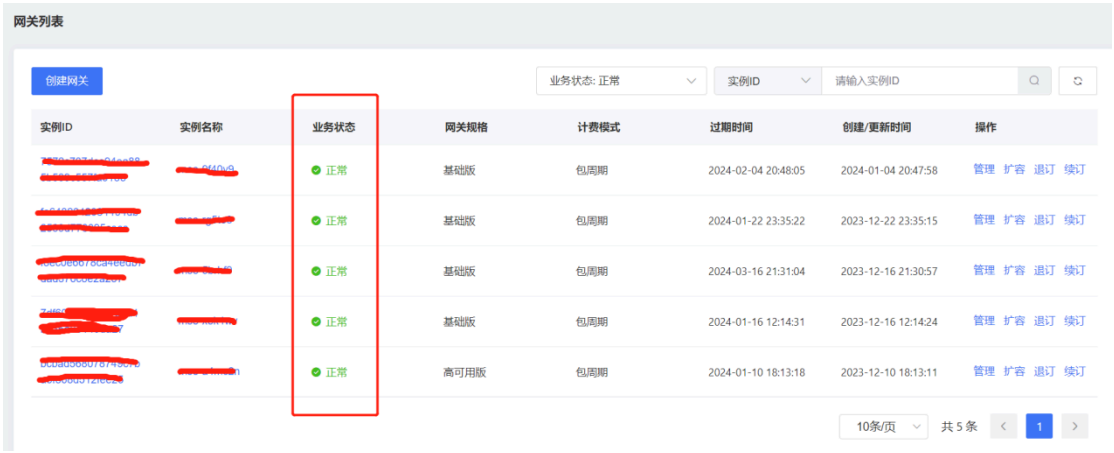
在多可用区部署模式下，云原生网关提供的同城跨AZ部署架构如下；同城中心之间物理距离较近，网络延迟低，可防范AZ级别性质的灾难损害，例如火灾、断网或断电等。



当云原生网关集群具有3个节点及3个节点以上时，具备多AZ容灾能力。一旦某个AZ出现故障时，流量会自动流量另外两个AZ，控制面仍然可以正常对外提供服务；一旦出现两个AZ故障时，将会导致集群控制面异常。

结果验证

在网关实例列表页可以查看新建的网关实例ID、名称、业务状态等信息，且业务状态显示正常则表示新建网关实例成功。



查看网关详情

您可以在网关详情页查看网关基本信息和网关入口。

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 在云原生网关详情页您可以查看当前网关实例的基本信息和网关入口信息；

基本信息

网关入口

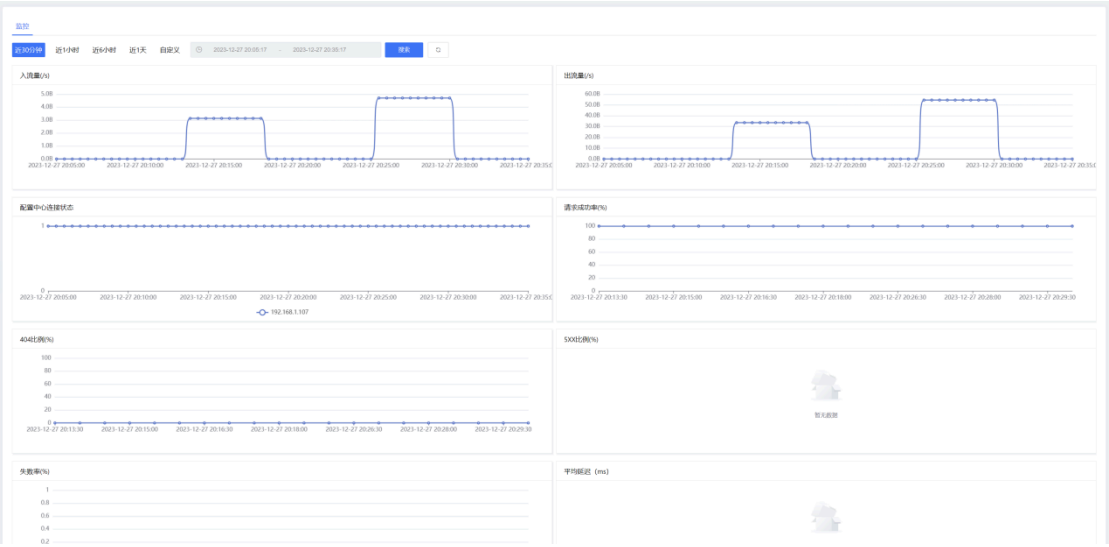
查看已绑定的ELB列表，您可以进行 绑定ELB 、解绑ELB操作。操作说明请参照管理网关入口ELB。

查看网关监控数据

云原生网关支持监控分析、资源监控、日志中心三种监控手段。

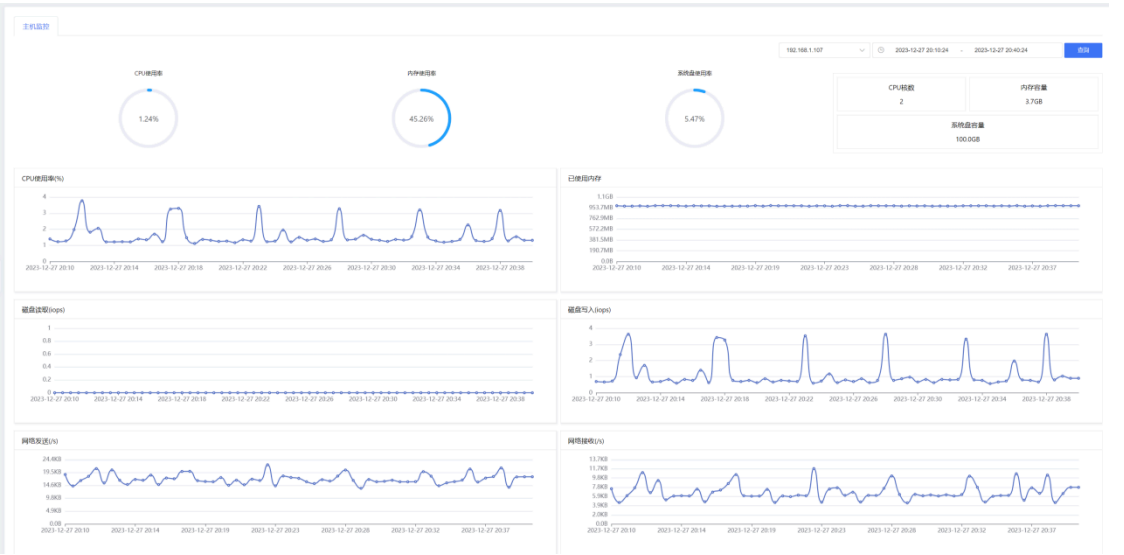
查看监控分析

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航页观测分析 > 监控分析；
6. 在监控分析页可以查看入流量、出流量、配置中心连接状态、请求成功率、404比例、5XX比例、失败率、平均延迟、P50延迟、P95延迟、P99延迟、QPS、连接数指标监控；



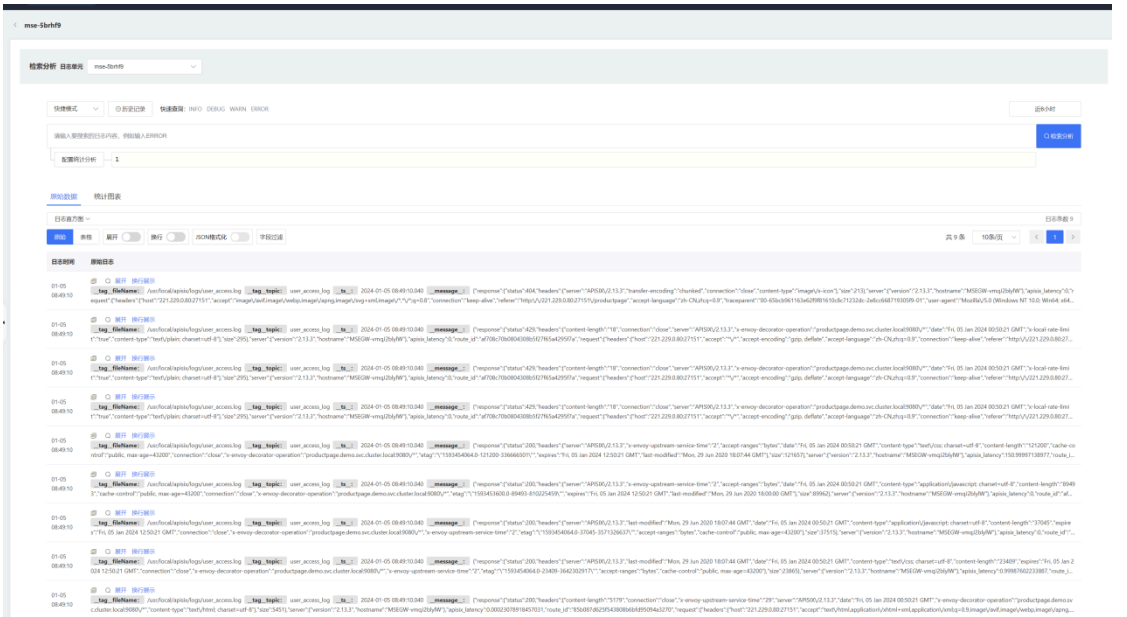
查看资源监控

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航页 观测分析>资源监控 ；
6. 在资源监控页上方，您可以查看CPU使用率、内存使用率、系统盘使用率、CPU核数、内存容量、系统盘容量数据概览；
7. 在资源监控页上方，您可以查看CPU使用率、已使用内存、磁盘读取、磁盘写入、网络发送、网络接收图表监控；



查看日志中心

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航页 观测分析>日志中心 ；



管理网关入口ELB

概述

云原生网关支持绑定一个或多个ELB作为网关入口，从而将访问流量自动分发到多台网关节点，实现更高水平的应用程序容错性能。当应用程序的流量和负载超过单台网关节点的承受能力时，您可以通过绑定ELB来实现负载均衡，保障应用程序的可用性。

绑定ELB

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 在网关详情页下方网关入口区域，您可以单击右侧绑定ELB按钮；
6. 在绑定ELB面板中配置ELB参数，单击左下角确认按钮；

ELB配置参数说明

参数	描述
类型	支持公网和私网
ELB	选择同vpc下的ELB实例，若您还没有ELB实例，请先创建ELB实例。
HTTP端口	配置HTTP端口
HTTPS端口	配置HTTPS端口

注意

80、443、8080、8443常见敏感端口需进行备案后才可配置

7. 在网关入口区域可以查看新绑定的ELB，刚绑定时关联状态为开始绑定，系统会自动刷新，直至关联状态更新为 绑定成功 。

列表参数说明如下：

参数	描述
ELB ID	ELB的ID，单击ELB ID可以查看ELB详情。
入口地址（ip）	ELB的ip地址，即网关入口的访问地址，您可以根据实际需求添加域名。
HTTP端口	HTTP端口
HTTPS端口	HTTPS端口
类型	公网或私网
关联状态	网关实例与ELB的关联状态，当关联状态为绑定成功时可以正常访问。
关联时间	网关实例与ELB关联的时间。
操作	您可以单击解绑ELB进行解绑操作。

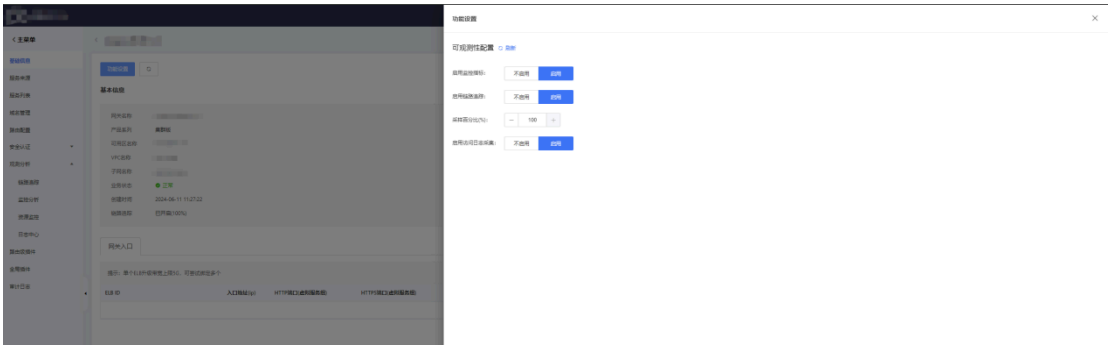
解绑ELB

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 在网关详情页下方网关入口ELB列表区域，您可以单击操作列 解绑ELB ；

网关配置

概述

云原生网关集成了天翼云链路追踪，提供链路数据采集上报、查询及分析能力。帮助您分析微服务调用拓扑，快速发现系统问题点和瓶颈。您可以根据需要配置观测能力，如下图所示：



监控分析

云原生网关集成了天翼云应用性能监控 ，提供系统、路由两个维度的QPS、成功率、延迟等监控能力。帮助您实时监控系统和路由的关键指标，快速发现系统问题点和瓶颈。

启用流程

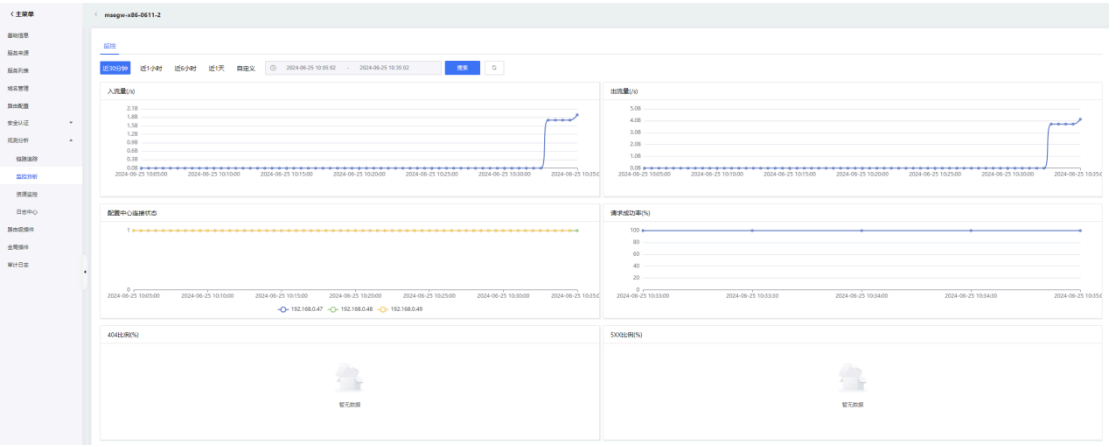
1. 开通并进入网关实例的控制台；
2. 左侧导航栏点击基础信息 ；
3. 单击左上角功能设置按钮；
4. 若您还没有开通应用性能监控产品，需先点击功能设置面板上的提示链接前往开通；
5. 已开通应用性能监控产品，可在功能设置面板上，启用监控分析切换到启用，然后单击左下角确认按钮；
6. 页面会自动刷新，也可点击刷新按钮，查看启用情况；

停用流程

1. 左侧导航栏点击基础信息 ；
2. 单击左上角功能设置按钮；
3. 启用监控分析切换到不启用，然后单击左下角确认按钮；
4. 页面会自动刷新，也可点击刷新按钮，查看停用情况；

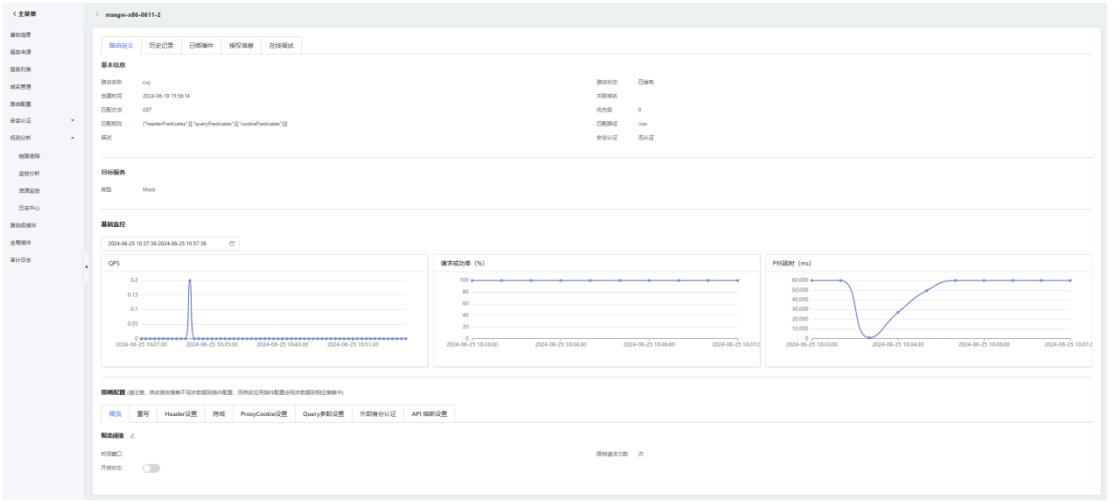
系统监控结果验证

1. 单击左侧导航栏 观测分析-> 监控分析 ；
2. 页面能正常进入并显示数据，则说明开启成功；



路由监控结果验证

1. 单击左侧导航栏 路由配置 ；
2. 选择需要查看的路由，进入路由信息页；
3. 基础监控显示数据，则说明开启成功；



链路追踪

云原生网关集成了天翼云应用性能监控，提供链路数据采集上报、查询及分析能力。帮助您分析微服务调用拓扑，快速发现系统问题点和瓶颈。

启用流程

1. 开通并进入网关实例的控制台；
2. 左侧导航栏点击基础信息 ；
3. 单击左上角功能设置按钮；
4. 若您还没有开通应用性能监控产品，需先点击功能设置面板上的提示链接前往开通；
5. 已开通应用性能监控产品，可在功能设置面板上，启用链路追踪切换到启用，并根据实际需求填写采样百分比 ，然后单击左下角确认按钮；

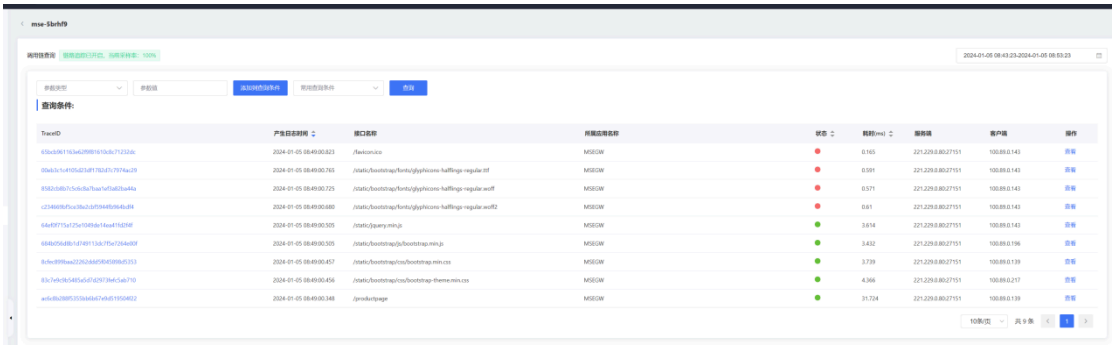
6. 页面会自动刷新，也可点击刷新按钮，查看启用情况；

停用流程

1. 左侧导航栏点击基础信息 ；
2. 单击左上角功能设置按钮；
3. 启用链路追踪切换到不启用，然后单击左下角确认按钮；
4. 页面会自动刷新，也可点击刷新按钮，查看停用情况；

结果验证

1. 单击左侧导航栏观测分析->链路追踪 ；
2. 页面能进入，则说明开启成功；
3. 模拟发起路由请求，若命中采集百分比，可在页面中查看链路追踪信息；



参数说明详见链路追踪->用户指南->应用概览

参数	说明
TraceID	调用链的唯一标识。
产生日志时间	该调用链出现日志记录的时间点，支持升序或者降序排序。
接口名称	显示发起API调用时的接口名称。
所属应用	显示当前应用实例所属应用的名称。
状态	与HTTP状态码对应。
耗时（ms）	一个完整的链路调用所消耗的时间。
服务端	调用链中第一段的被调用的应用的IP端口。
客户端	调用链中第一段的发起调用的应用的IP地址。

访问日志

云原生网关集成了天翼云云日志服务，提供访问日志的上报、查询功能。帮助您快速定位问题。

启用流程

1. 开通并进入网关实例的控制台；
2. 左侧导航栏点击基础信息 ；
3. 单击左上角功能设置按钮；

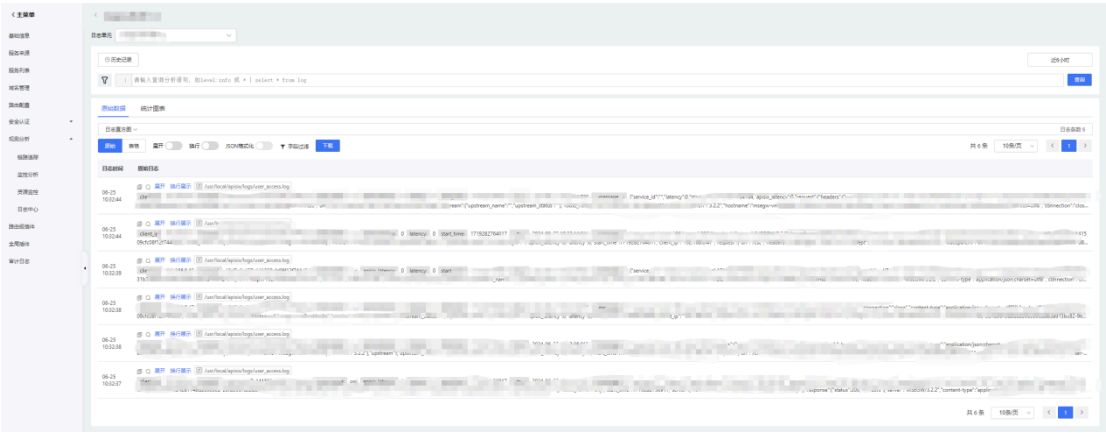
4. 若您还没有开通云日志服务产品，需先点击功能设置面板上的提示链接前往开通；
5. 已开通云日志服务产品，可在功能设置面板上，启用访问日志采集切换到启用，然后单击左下角确认按钮；
6. 页面会自动刷新，也可点击刷新按钮，查看启用情况；

停用流程

1. 左侧导航栏点击基础信息 ；
2. 单击左上角功能设置按钮；
3. 启用访问日志采集切换到不启用，然后单击左下角确认按钮；
4. 页面会自动刷新，也可点击刷新按钮，查看停用情况；

结果验证

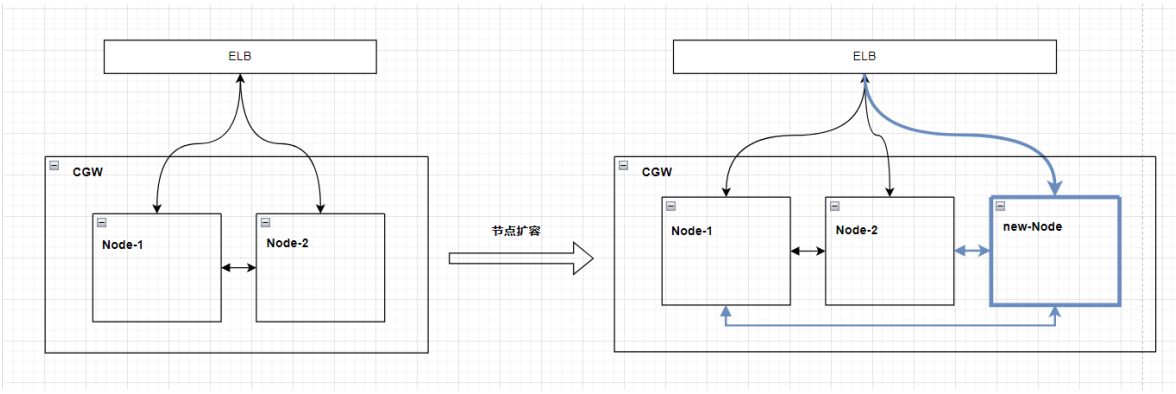
1. 单击左侧导航栏 观测分析-> 日志中心 ；
2. 页面能进入，则说明开启成功；
3. 模拟发起路由请求，日志中心可查看到相关访问日志；



网关扩容

概述

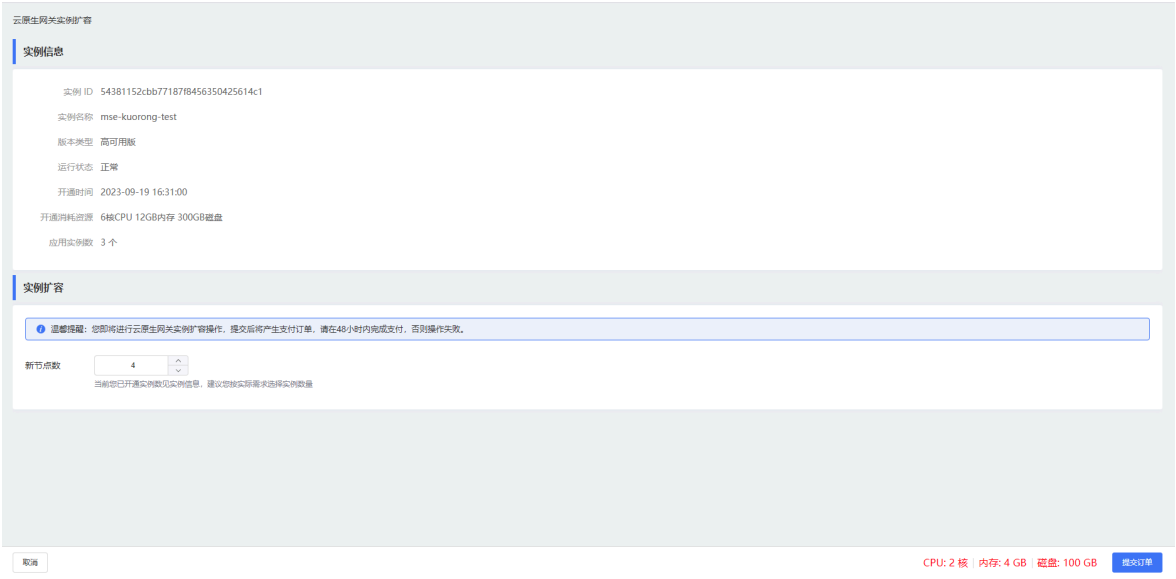
节点扩容可以满足业务规模增长的需求；云原生网关目前支持高可用版的节点扩容，基础版不支持扩容能力；扩容的大概过程如图所示，集群扩容后，将更新ELB与网关实例的绑定关系，在此过程中，通过ELB作负载均衡来实现无损节点扩容。



操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 **>** 网关列表 ；
4. 选择需要扩容 的实例 ， 点击扩容，跳转扩容订单页；

- 实例信息部分展示变更前实例原有的基本信息
- 实例扩容部分为变更后的总节点数



5. 在实例扩容栏目， 选择新节点数 ， 提交订单，完成付款；

- 新节点表示实例扩容完成后集群具有的总节点数

6. 返回微服务引擎MSE控制台，单击左侧导航栏云原生网关 **->** 网关列表 ；
7. 查看实例当前状态（扩容中） ，此时实例不可进行路由等资源的配置，但不影响已有路由转发；

8. 扩容完成后， 实例状态恢复为正常 ， 实例可以正常访问；
9. 查看实例基本信息，实例节点数、连接信息都已经更新完成；

网关续订

如何为已开通的云原生网关进行续费？

实例计费模式为包年包月，当实例生命周期达到截至使用时间时，云原生网关实例会处于到期停机的状态，然后保留15天，保留期内，用户可以通过续费恢复实例；若超出保留期实例将被真正地清理。

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 在实例列表点击“续订”；

5. 进入实例续订页，选择 续订时长 ，提交订单，完成续费；

网关退订

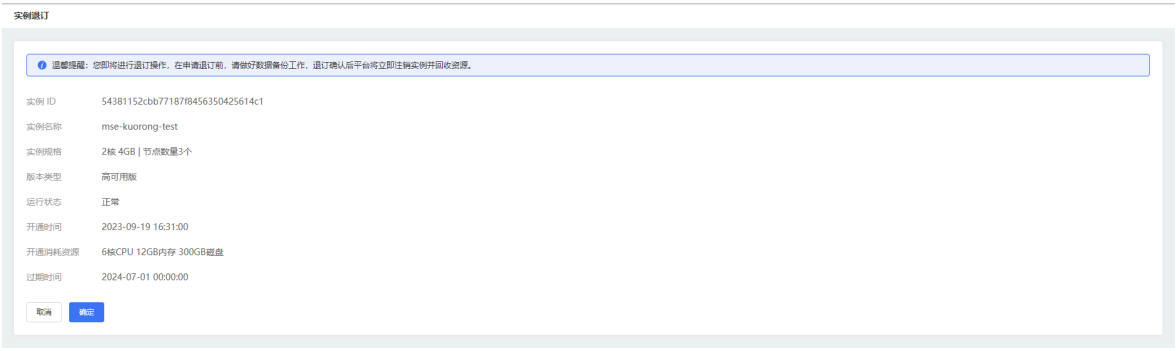
概述

在云原生网关控制台实例列表页面，选择需要退订的实例，点击退订，在弹出的对话框中点击确认，实例资源会在后台自动释放。

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 **>** 网关列表 ；
4. 选择要退订的实例，点击退订；

5. 跳转退订页，提交订单，进行退订；



6. 返回微服务引擎MSE控制台，单击左侧导航栏云原生网关 -> 网关列表 ；
7. 查看实例列表，退订实例不可访问；

HTTP2.0支持

云原生网关默认支持了HTTP/2协议，能有效提高Web应用的性能和用户体验。在使用过程中，需要注意几点：

1. HTTP/2通常是在HTTPS的基础上运行，因此需要确保路由配置时使用HTTPS请求，也即是需要绑定域名到路由上，使用SSL端口（在云原生网关中我们将SSL端口固定为27154）请求路由，示例：

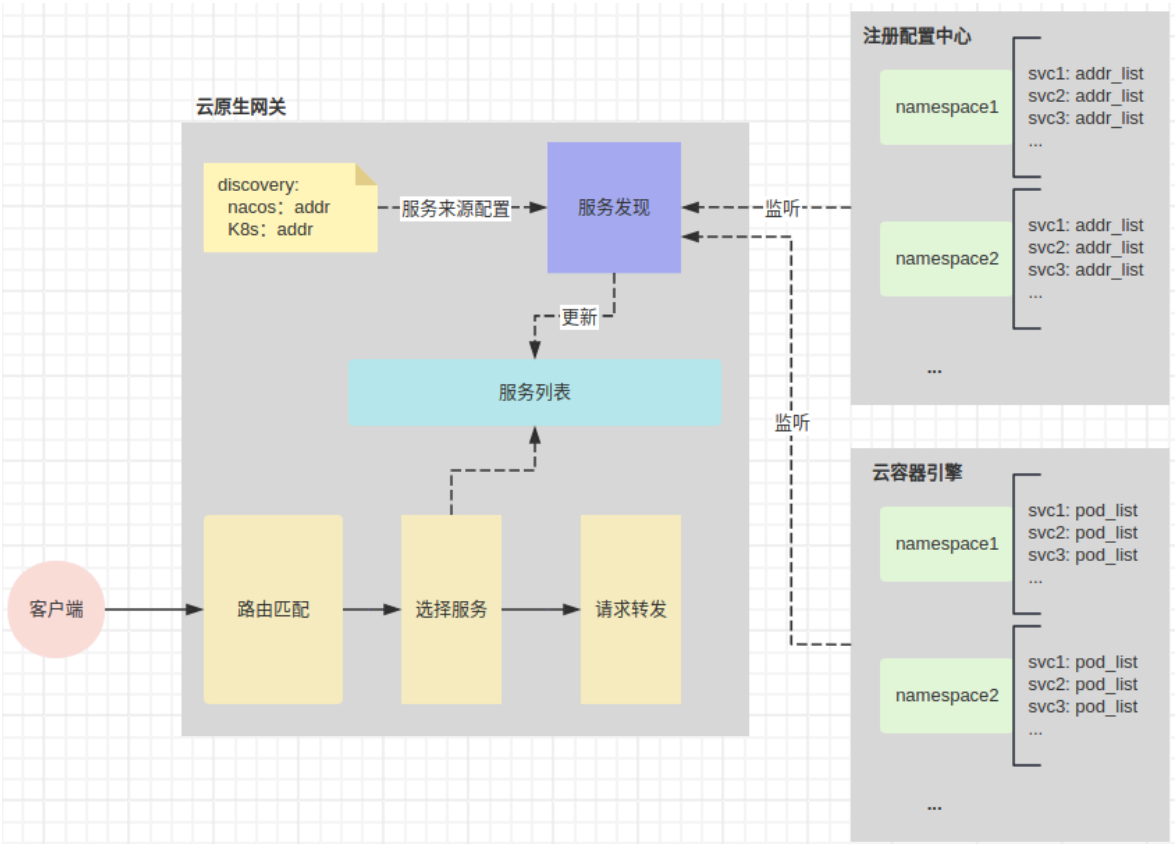
```
curl -i --http2 https://your_domain:ssl_port/your_uri
```

2. 服务端也需要支持并使用HTTP/2协议

服务来源管理

概述

云原生网关无缝对接天翼云注册配置中心和云容器引擎，通过服务发现模块监听服务来源，动态感知后端服务。整体架构如下：



创建 云容器引擎服务来源

云原生网关支持云容器引擎作为服务来源，本文介绍添加云容器引擎作为服务来源。

1. 进入微服务引擎MSE控制台
2. 在顶部菜单栏选择资源池
3. 单击左侧导航栏云原生网关 -> 网关列表
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮
5. 单击左侧导航栏 服务来源
6. 单击左上角按钮 创建来源
7. 在弹出的页面中，来源类型选择容器服务；在容器集群下拉列表中选择要添加的集群（当前仅支持添加与网关实例同VPC内的容器集群）
8. 根据需要勾选是否监听K8s Ingress选项并配置监听命名空间的标签，详细参考Ingress管理章节

创建 注册配置中心 服务来源

云原生网关支持MSE注册配置中心作为服务来源，本文介绍添加注册配置中心作为服务来源。

1. 进入微服务引擎MSE控制台
2. 在顶部菜单栏选择资源池
3. 单击左侧导航栏云原生网关 -> 网关列表
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮
5. 单击左侧导航栏 服务来源
6. 单击左上角按钮 创建来源
7. 在弹出的页面中，来源类型选择MSE Nacos（当前仅支持Nacos引擎）；在集群下拉列表中选择要添加的集群（当前仅支持添加与网关实例同VPC内的集群）

删除服务来源

在网关实例服务来源列表页，右侧操作栏可以对已经添加的服务来源进行删除操作

说明

如果有服务定义依赖当前服务来源，则不能删除该服务来源；需要删除该服务来源下定义的所有服务才可以删除该服务来源。

服务列表

概述

云原生网关支持将请求转发的预定义的目标服务，本章节介绍云原生网关中的服务生命周期管理。

云原生网关中服务相关概念如下：

概念	说明
服务	网关转发请求的目标，可以通过配置路由引用相应的服务，将请求转发到指定的目标。
容器服务	从云容器引擎中发现的服务。
注册中心服务	从注册配置中心发现的服务。
固定地址服务	固定配置的IP+端口服务。
Dubbo服务	添加服务时，服务scheme选择DUBBO。

概念	说明
服务scheme	请求后端服务的协议，当前支持 HTTP、HTTPS、GRPC、GRPCS、DUBBO，默认为HTTP。
服务版本	云原生网关支持基于容器和注册中心服务中的元数据（容器服务中的POD标签或者注册中心服务的元数据）对服务进行分组，并定义为不同版本，即为服务版本概念。

创建 服务

云原生网关支持创建容器服务、注册中心服务和固定地址服务，本章节介绍每种服务创建操作流程。

创建容器服务

前置条件

1. 已开通云原生网关实例和同VPC下的云容器引擎实例，已添加云容器引擎作为云原生网关的服务来源
2. 服务已部署到云容器引擎并配置相关Service暴露服务

操作流程

1. 进入微服务引擎MSE控制台
2. 在顶部菜单栏选择资源池
3. 单击左侧导航栏云原生网关 -> 网关列表
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ，也可以单击操作列中的管理按钮
5. 单击左侧导航栏 服务列表
6. 单击左上角按钮 创建服务
7. 在弹出的页面中选择容器服务来源，选择要添加的服务所在的命名空间
8. 在服务列表中选择要添加的服务，根据需求做不同的配置

容器服务配置说明

配置	说明
命名空间	后端服务部署在云容器引擎中的命名空间。
servicePort	对应云容器引擎Service中定义的服务端口，每个端口可以定义为一个独立的云原生网关服务。
websocket支持	websocket开关，打开之后支持websocket流量代理到后端。
协议	后端服务的协议，支持 HTTP、HTTPS、GRPC、GRPCS、DUBBO，默认为HTTP。
mTLS	是否打开后端服务双向TLS认证（后端服务对网关证书认证）。
证书文件	后端服务开启双向TLS认证后，云原生网关提供的证书文件。
私钥文件	后端服务开启双向TLS认证后，云原生网关提供的私钥文件。

创建注册中心服务

前置条件

1. 已开通云原生网关实例和同VPC下的注册配置中心实例（Nacos引擎），已添加注册配置中心实例作为云原生网关的服务来源
2. 后端服务已部署并注册到注册配置中心

操作流程

1. 进入微服务引擎MSE控制台
2. 在顶部菜单栏选择资源池
3. 单击左侧导航栏云原生网关 -> 网关列表
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮
5. 单击左侧导航栏 服务列表
6. 单击左上角按钮 创建服务
7. 在弹出的页面中选择MSE Nacos服务来源，选择要添加的服务所在的命名空间
8. 在服务列表中选择要添加的服务，根据需求做不同的配置

注册中心服 务配置 说明

配置	说明
命名空间	后端服务注册到注册配置中心的命名空间。
websocket支持	websocket开关，打开之后支持websocket流量代理到后端。
协议	后端服务的协议，支持 HTTP、HTTPS、GRPC、GRPCS、DUBBO，默认为HTTP。
mTLS	是否打开后端服务双向TLS认证（后端服务对网关证书认证）。

配置	说明
证书文件	后端服务开启双向TLS认证后，云原生网关提供的证书文件。
私钥文件	后端服务开启双向TLS认证后，云原生网关提供的私钥文件。

创建固定地址服务

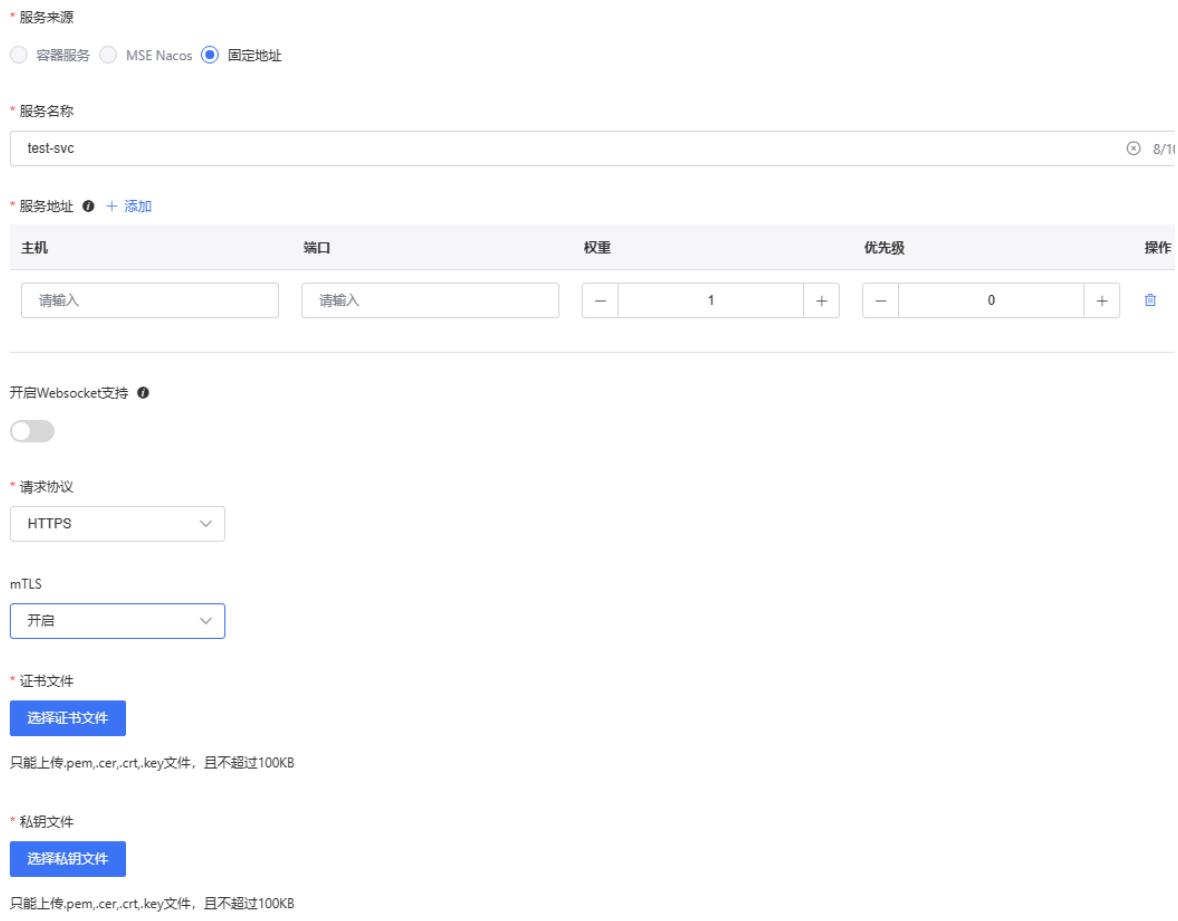
前置条件

1. 已开通云原生网关实例

操作流程

1. 进入微服务引擎MSE控制台
2. 在顶部菜单栏选择资源池
3. 单击左侧导航栏云原生网关 -> 网关列表
4. 您可以在网关列表页单击需要查看的网关实例ID或者 例名称 ， 也可以单击操作列中的管理按钮
5. 单击左侧导航栏服务列表
6. 单击左上角按钮创建服务

在弹出的页面中选择添加固定地址类型的服务，如下图所示：



添加固定地址服务的配置参数包括：

参数	说明
服务名称	用于唯一标识一个后端服务。
服务地址	1. 支持配置服务节点的地址、端口、权重和优先级。2. 优先访问高优先级节点，只有在高优先级的节点不可用或者尝试过，才会访问一个低优先级的节点；3. 只有在优先级相同时，权重才会生效；4. 优先级默认值为0，可以取负数表示备份节点，表示只有其他节点均不可用时，才会启用备份节点；5. 此处节点权重采用比例形式，表示不同节点之间的流量比例。
请求协议	后端服务的协议，支持 HTTP、HTTPS、GRPC、GRPCS、DUBBO，默认为HTTP。
mTLS	是否打开后端服务双向TLS认证（后端服务对网关证书认证）。
证书文件	后端服务开启双向TLS认证后，云原生网关提供的证书文件。

参数	说明
私钥文件	后端服务开启双向TLS认证后，云原生网关提供的私钥文件。

删除服务

前提条件

没有路由在引用要删除的服务（先删除引用当前服务的路由）

操作流程

1. 进入微服务引擎MSE控制台
2. 在顶部菜单栏选择资源池
3. 单击左侧导航栏云原生网关 -> 网关列表
4. 您可以在网关列表页单击需要查看的网关实例ID或者实例名称 ，也可以单击操作列中的管理按钮
5. 单击左侧导航栏 服务列表
6. 在右侧操作栏点删除按钮

编辑服务

云原生网关支持对已添加的服务进行编辑，本章节介绍服务编辑功能。

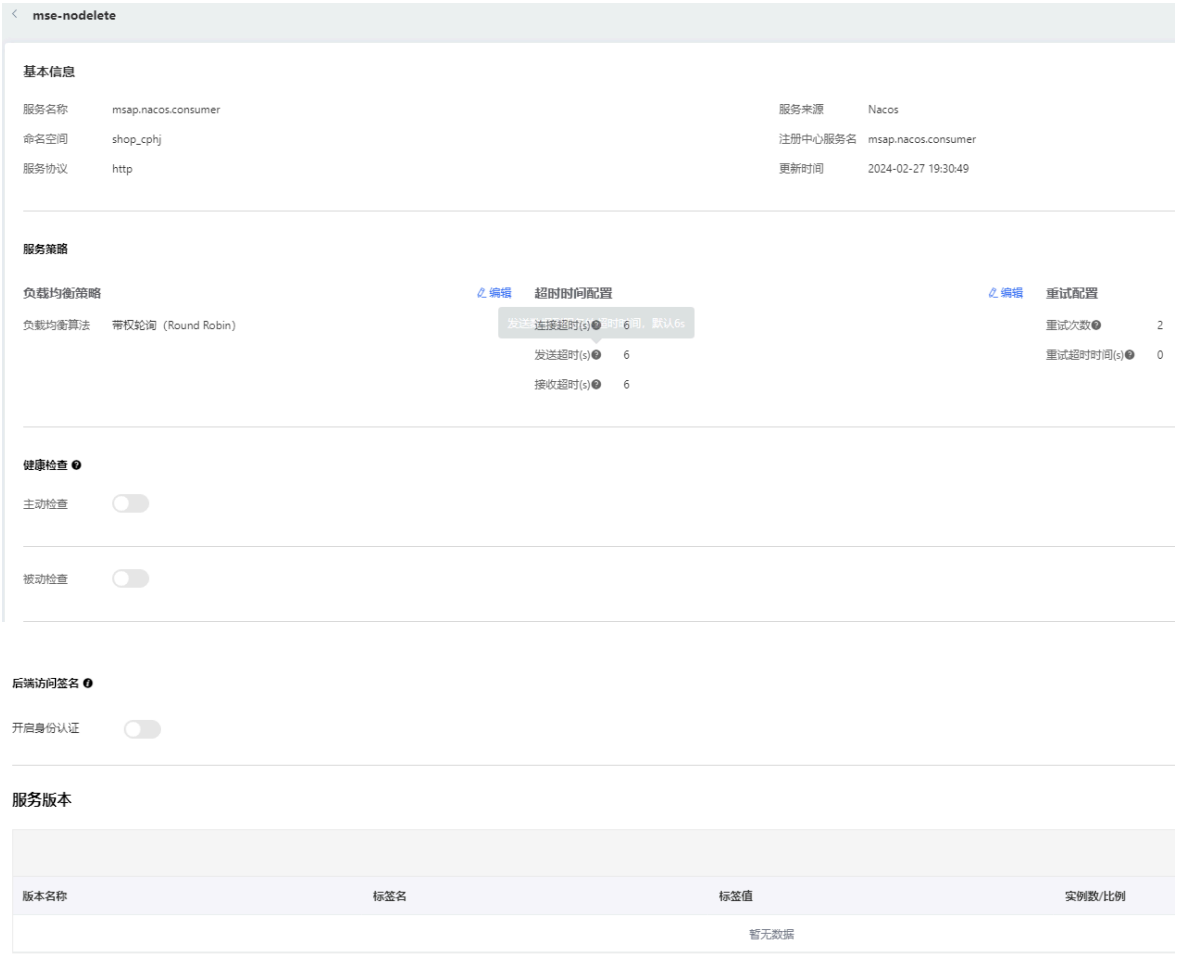
操作流程

1. 进入微服务引擎MSE控制台
2. 在顶部菜单栏选择资源池
3. 单击左侧导航栏云原生网关 -> 网关列表
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ，也可以单击操作列中的管理按钮
5. 单击左侧导航栏 服务列表
6. 在右侧操作栏点编辑按钮

当前允许编辑的服务选项包括：websocket选项、请求协议、mTLS配置

服务策略配置

云原生网关支持丰富的服务策略配置，点击服务列表项操作栏的策略配置按钮或者服务名称进入服务详情页面，可以实现后端服务级别的策略配置，包括负载均衡、超时重试、健康检查、后端访问签名、服务版本等，如下图：



策略配置项和支持的服务类型如下表：

配置项	说明	支持的服务类型
负载均衡策略	后端多实例的负载均衡策略，当前支持：1，轮询（Round Robin）2，一致性哈希（CHash）3，指数加权移动平均法（EWMA）4，最小连接数（Least Conn）	容器/注册中心/固定地址
超时	当前支持配置连接超时、发送超时、接收超时，默认都是6s。	容器/注册中心/固定地址
重试	重试次数，默认为0，表示不重试。 重试超时时间：原请求和重试的总耗时超过该时间则不再继续重试。	容器/注册中心/固定地址
健康检查	后端服务监控检查配置，包括主动健康检查和被动健康检查。	容器/注册中心/固定地址

配置项	说明	支持的服务类型
后端访问签名	请求后端时基于HMAC算法计算签名，用于后端对网关的回源请求进行鉴权。	容器/注册中心/固定地址
服务版本	针对容器和Nacos服务，基于标签定义服务的不同版本，可以在路由配置中实现多版本的蓝绿和金丝雀发布能力。	容器/注册中心

健康检查配置

云原生网关对后端服务的健康检查分为主动健康检查和被动健康检查。

主动健康检查通过预设的探针，主动探测上游节点的存活性，目前支持 HTTP、HTTPS、TCP 三种探针类型。当发往某个健康节点的若干个连续探测请求都失败时，则该节点将被标记为不健康，不健康的节点将会被网关的负载均衡器忽略，无法收到请求；若发往某个不健康的节点的连续若干个探测请求都成功，则该节点将被重新标记为健康，进而可以被代理。

主动健康检查配置界面如下图：


主动检查
类型
HTTP
超时时间(s)
1
并行数量
10
端口
范围1~65535
请求路径
请求头
+ 请求头
健康状态
* 间隔时间(s)
1
* 成功次数
2
* 状态码
200, 302
不健康状态
超时次数
3
间隔时间(s)
1
* 状态码
429, 404, 500, 501, 502, 503, 504, 505
* HTTP失败次数
5
* TCP失败次数
2

主动健康检查支持的配置项有：

配置项	说明
探测类型	当前支持TCP、HTTP、HTTPS探测。
超时时间	探测请求超时时间。
并行数量	并发主动探测的最大数量。
端口	探测的服务端口。
请求路径	探测请求的路径，仅对HTTP和HTTPS探测有效。
请求头部	探测时指定请求头部，仅对HTTP和HTTPS探测有效。
健康状态定义	对于不健康节点的探测配置，用于判断节点是否恢复健康。1，探测时间间隔（秒）2，成功次数：主动探测成功达到次次数时认为节点是健康的 3，状态码：对于HTTP和HTTPS探测，定义了哪些状态码是健康的，如2XX，3XX
不健康状态定义	对于健康节点的探测配置，用于判断节点是否监控。1，探测时间间隔（秒）2，超时次数：超时次数大于或者等于该配置时认为节点不健康 3，状态码：定义了哪些状态码是异常的，如4XX，5XX 4，HTTP失败次数：HTTP失败次数大于等于该值时认为节点不健康5，TCP失败次数：TCP失败大于等于该值时认为节点不健康

被动健康检查是指，通过网关接收到的上游节点的响应状态来判断对应的上游节点是否健康。相对于主动健康检查，被动健康检查的方式无需发起额外的探测请求，缺点是无法提前感知节点状态，需要有一定量的失败请求才能触发故障节点的剔除。若发向某个健康节点的若干个连续请求都被判定为失败，则该节点将被标记为不健康。

被动健康检查的配置如下图：

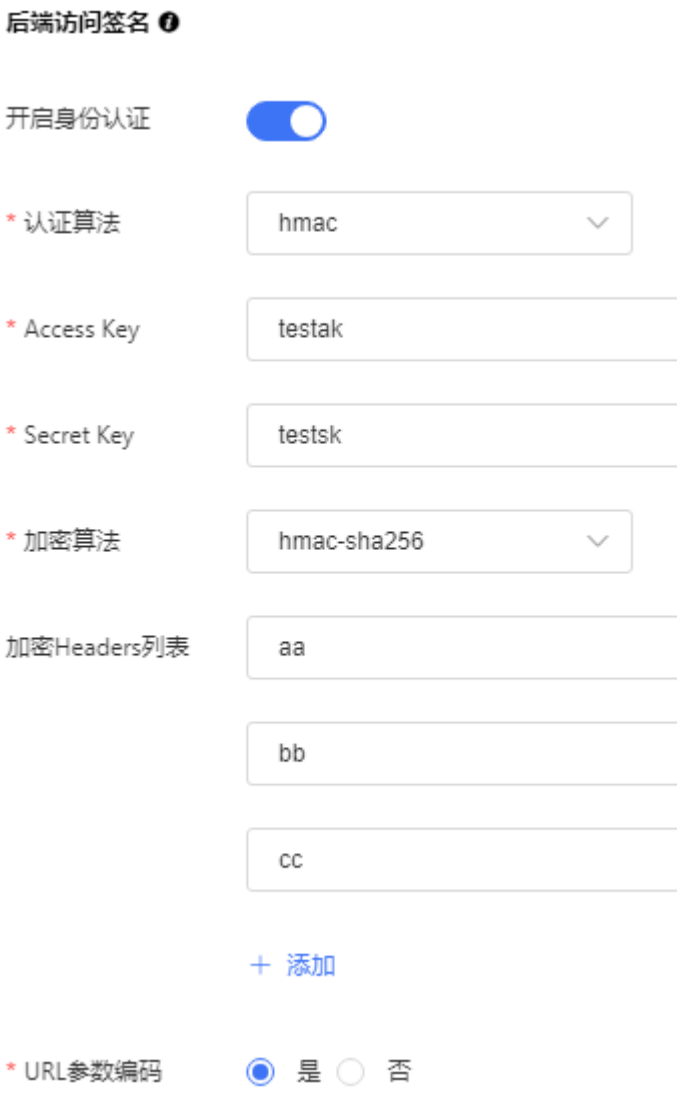
被动健康检查的配置说明：

配置	说明
类型	当前支持TCP、HTTP、HTTPS三种类型。
健康状态定义	状态码：成功对应的状态码，主要是2XX、3XX等。成功次数：成功的次数超过该值则认为节点健康。
不健康状态定义	超时次数：超时次数超过该值则认为节点不健康。TCP失败次数：TCP失败超过该值则认为节点不健康。HTTP失败次数：HTTP失败超过该值则认为节点不健康。状态码：失败对应的状态码，包括4XX、5XX等。

后端访问签名

使用云原生网关的场景下，后端服务不直接对外暴露服务，而是通过云原生网关将请求转发到后端。从安全角度，配置认证鉴权策略限制只允许云原生网关等作为请求方访问自己。针对这种场景，云原生网关可以配置特

定的签名策略，用于后端服务的认证鉴权。当前云原生网关添加服务时可以配置HMAC算法签名策略，如下图所示：



配置选项说明如下：

选项	说明
认证算法	当前仅支持HMAC。
Access Key	后端服务校验签名使用的HMAC access key。
Secret Key	后端服务校验签名使用的HMAC secret key。
加密算法	HMAC加密算法，支持hmac-sha1，hmac-sha256，hmac-sha512。
加密Headers列表	用于计算签名的header列表。

选项	说明
URL参数编码	URL参数是否经过编码之后再计算签名，默认是。

服务版本管理

服务的版本一般是服务迭代过程中的概念，在云原生网关的场景下，服务版本引申为服务的标签概念，一个服务可能有多个版本（标签）；基于服务的版本（标签）可以实现如标签路由、灰度发布、高可用部署等能力；如对于一个服务存在多个版本的场景，基于不同版本的实例上所携带的标签不同，将服务定义为多个不同的版本；根据请求的特征匹配，路由到不同版本的服务中可以实现标签路由；将服务的流量在多个版本之间分配，可以实现金丝雀、蓝绿等灰度功能；根据服务的部署标签定义的版本则可以应用在服务的高可用部署上。

对于容器或者Nacos来源的服务，云原生网关支持根据服务的元数据定义不同的版本，如下图对于容器服务，可以根据pod的标签定义不同的服务版本：

配置说明如下：

配置	说明
版本名称	自定义的版本名称。
标签名	版本对应的标签名，可选值来自当前服务的元数据。
标签值	版本标签名对应的标签值，可选值来自当前服务的元数据。
实例数/比例	所选的标签选择到的实例数/相对于整体实例数的比例。

添加完服务版本则可以看到如下：

域名管理

关联域名

云原生网关提供域名配置能力，支持配置协议、证书、路由配置。本文介绍如何关联域名。

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 域名管理 ；
6. 单击左上角按钮 关联域名 ；
7. 在关联域名面板填写域名相关配置，单击左下角确定按钮；

域名配置说明如下：

参数	描述
域名	自定义域名填写，如mse.ctyun.com，同时支持泛域名如*.ctyun.com。
协议	支持HTTP和HTTPS协议，HTTPS协议需上传证书。
证书名称	选择HTTPS时填入，自定义填写证书名称。
证书文件	选择HTTPS时需上传证书文件，支持pem、cer，crt，key格式。
私钥文件	选择HTTPS时需上传私钥文件，支持pem、cer，crt，key格式。

结果验证

在域名管理列表页可以查看新关联的域名信息，且状态显示已发布 ，则说明域名关联成功。

变更域名

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 -> 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 域名管理 ；
6. 单击域名管理页操作列编辑按钮；
7. 在编辑域名面板编辑协议、证书等信息，单击左下角确定按钮；

结果验证

在域名管理列表页查看域名的协议、证书等信息是否已更新。

删除域名

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 域名管理 ；
6. 单击域名管理页操作列删除按钮；
7. 单击弹出窗口确定按钮；

结果验证

在域名管理列表页查看域名是否已被删除。

路由配置

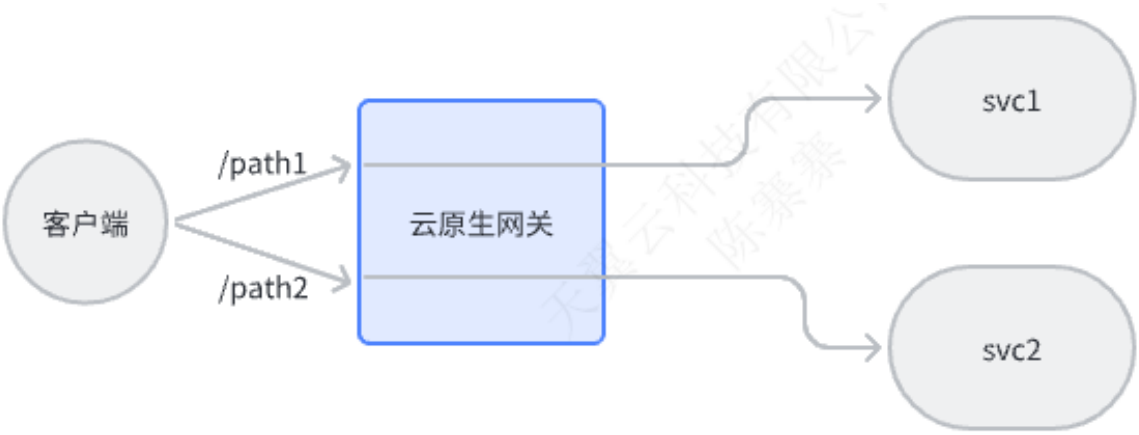
路由概述

概述

云原生网关基于用户配置的host，path，query，header，cookie等信息配置特定的转发规则，当前云原生网关支持多种路由方式，包括单服务路由、多服务路由、标签路由、mock和重定向路由等。

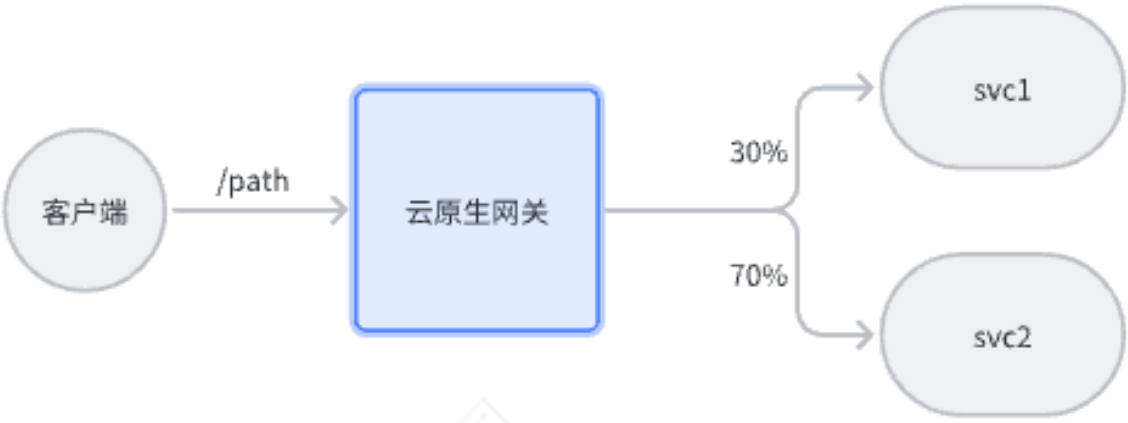
单服务路由

单服务路由根据用户配置的路由规则将请求转发到单个后端服务，例如请求匹配/path1转发到svc1，匹配/path2转发到svc2



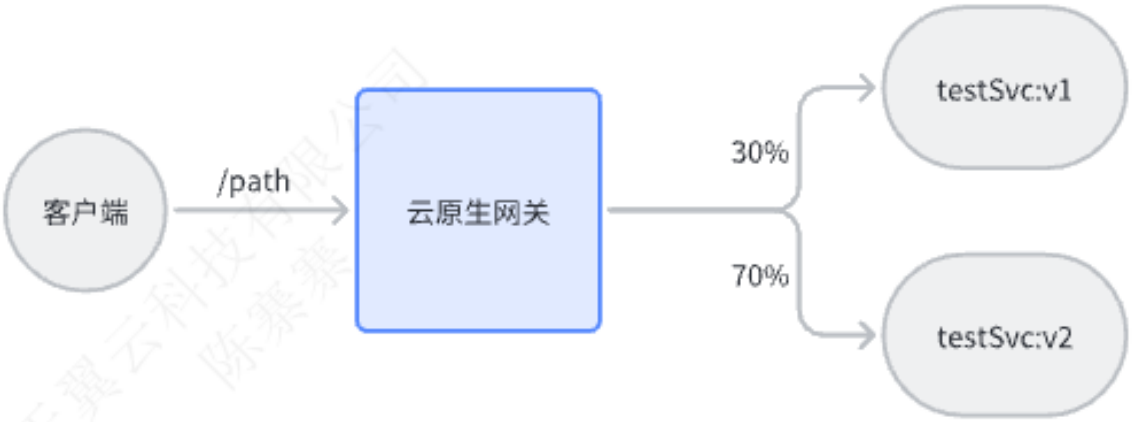
多服务路由

多服务路由模式下支持将请求转发到多个不同的后端服务，每个后端服务配置不同的访问权重（总权重之和为100%）；该模式可用于灰度发布等场景；如对于到/path的请求，30%转发到svc1,70%转发到svc2

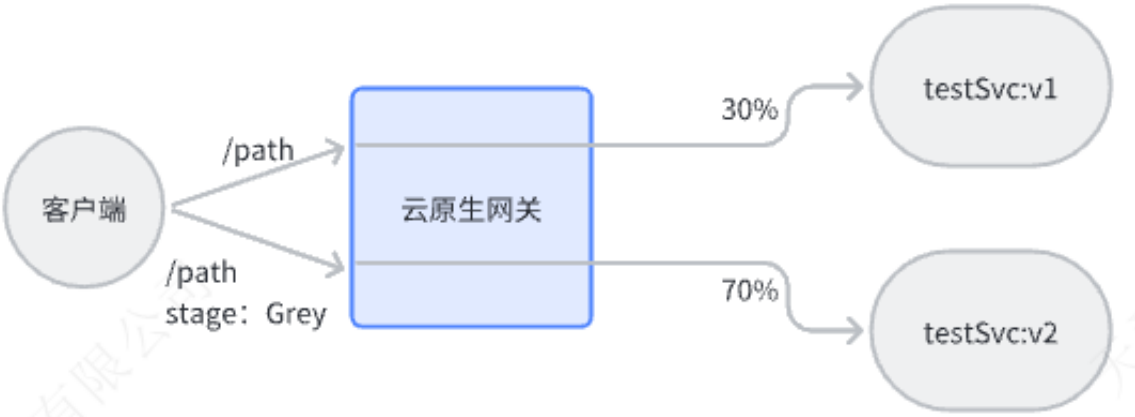


标签路由

针对同一个服务，可以根据nacos或者k8s中记录的服务标签信息将服务分组；标签路由模式下，支持将请求在同一个服务的不同标签分组之间分流；例如，根据服务标签信息将testSvc打上v1和v2两个标签，可以在匹配路由时30%的流量转发到v1版本，70%的流量转发到v2版本：

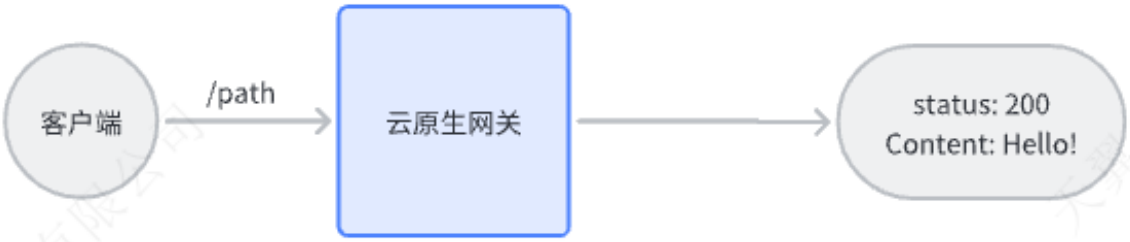


或者根据请求的内容将请求转发到不同的服务版本：



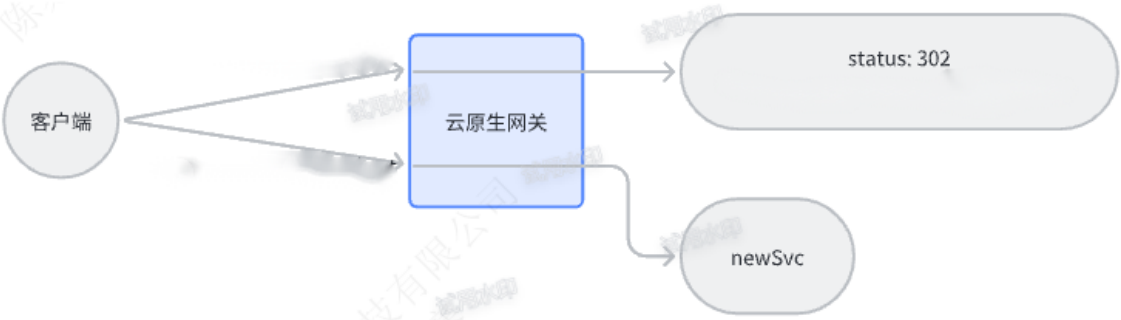
mock路由

该模式主要用于测试场景，可以配置路由直接返回mock内容，包括HTTP状态码，返回内容等信息，例如针对/path的请求返回HTTP 200，内容为"Hello!"：



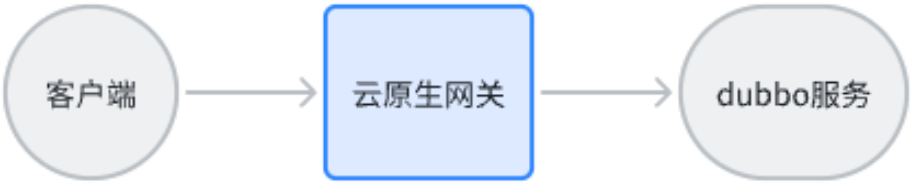
重定向路由

返回3xx状态码以及重定向地址，引导调用方访问重定向的地址，如下：



dubbo代理路由

通过内部dubbo服务代理插件请求dubbo服务，并返回结果。



本功能目前对后端dubbo服务有如下限制：

1. 本功能使用hessian2作为反序列化协议，请确保后端dubbo服务使用该协议作为默认的数据序列化协议。
2. 本功能对dubbo服务的数据返回有相关要求，来确保能将数据结果正确映射成http响应，示例代码如下所示：
   a. 使用Map<String, Object>作为结果返回体；
   b. 其中固定key值body映射为http响应结果；
   c. 固定key值status映射为http响应状态码；
   d. 自定义key值和value值映射为http响应体header。

```
public Map<String, Object> tengineDubbo(Map<String, Object> context) {
  for (Map.Entry<String, Object> entry : context.entrySet()) {
    System.out.println("Key = " + entry.getKey() + ", Value = " + entry.getValue());
  }

  Map<String, Object> ret = new HashMap<String, Object>();
```

```
    ret.put("body", "dubbo success\n");
    ret.put("status", "200");
    ret.put("test", "123");

    return ret;
}
```

路由匹配规则

概述

云原生网关支持多种路由匹配规则。彼此之间可以相互组合，路由将会根据所有条件进行“与”的关系匹配。

基于域名的匹配

支持根据请求的域名进行匹配，支持绑定多个域名。也支持类似*.ctyun.com的泛域名。

基于路径的匹配

路径匹配支持精确匹配和前缀匹配模式，云原生网关会优先尝试精确匹配，若无法命中精确匹配，再尝试前缀匹配。

精确匹配

完整匹配给定的路径，如/foo/bar匹配请求路径为/foo/bar的请求。

前缀匹配

末尾使用'*'代表使用前缀匹配，如/foo/bar/*匹配/foo/bar、/foo/bar/baz、/foo/bar/a/b/c等请求。

基于请求方法的匹配

支持根据HTTP请求方法的匹配，例如GET、POST、PUT、DELETE等。可绑定多种请求方法。

基于请求头的匹配

支持根据HTTP请求头进行匹配。多个请求头之间是“与”的匹配关系。

基于请求参数的匹配

支持URL参数进行匹配。多个参数之间是“与”的匹配关系。

基于Cookie的匹配

支持Cookie进行匹配。多个Cookie之间是“与”的匹配关系。

基于参数规整化的匹配

支持从Header、Query或者Cookie中获取参数，可以进一步对参数求哈希值（Java String hashCode方法）、取模等；对于运算的结果支持匹配一组枚举值或者范围；该种匹配方式可用于实现多活容灾路由等场景。

新建路由规则

概述

输入路由名称，匹配域名、路径、方法、header、query等参数，目标转发地址为服务列表里面配置的地址；路径匹配支持精确匹配和前缀匹配模式，精确匹配如/foo/bar匹配请求路径为/foo/bar的请求，前缀匹配时，/foo/bar/*匹配/foo/bar、/foo/bar/baz、/foo/bar/a/b/c等请求。

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；

3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 路由配置 ；
6. 单击路由配置列表页左上角创建路由按钮；
7. 在创建路由面板填写路由相关配置，并单击确认或确认并发布按钮，如您单击的是确认按钮，则需要在路由配置列表页，单击操作列发布按钮发布路由；

路由配置的规则之间是“与”的关系，必须全部满足才算匹配，路由配置项说明如下：

参数	说明
路由名称	路由名称，用于标识一条路由规则。
认证方式	支持无认证和应用授权无认证。无需认证：建议调试阶段使用。应用授权：表示完成应用的授权操作后才有权访问。
域名	用于和请求中的域名进行匹配，不填则任何请求都可以匹配；可选项从域名管理中添加的域名选择。
匹配路径	匹配请求的path（不含query参数），当前支持前缀匹配和精确匹配。
方法	匹配请求中的HTTP方法。
优先级	当多个路由同时匹配一个请求时，路径匹配深度较大的路由优先；路径匹配相同的情况下，路由优先级高（数字大）的优先匹配。
请求头（header）	匹配请求中的HTTP header。
请求参数（Query）	匹配请求中的HTTP query参数。
Cookie	通过Cookie进行路由匹配，多个参数之间是“与”的关系。
是否启用参数规整化匹配	启用后支持对参数进行取模，并根据取模结果进行精确或者范围匹配。
参数类型	启用参数规整化匹配选择，支持 Header、Query、Cookie。
是否Hash	启用参数规整化匹配选择，是否对参数进行哈希处理后再取模，哈希函数为Java String hashCode。
取模数值	启用参数规整化匹配选择，自定义填写取模数值。
标记类型	支持精确和匹配精确。精确匹配，可用英文逗号分隔多个精确值。范围：最大值和最小值均是闭区间，[min,max]。
目标服务	当前支持单服务、多服务、标签路由、mock路由、重定向和dubbo代理。
描述	路由描述。

结果验证

在路由配置列表页可以查看新建的路由信息，且状态显示已发布 ，则说明新建路由成功。

变更路由规则

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 路由配置 ；
6. 单击路由配置列表页操作列编辑按钮；
7. 在修改路由面板修改路由相关配置，并单击确认或确认并发布按钮，如您单击的是确认按钮，则需要在路由配置列表页，单击操作列发布按钮发布路由；

结果验证

在路由配置页面查看配置是否已更新。

发布路由规则

前提条件

路由规则处于未发布状态。

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 路由配置 ；
6. 单击路由配置列表页操作列发布按钮；

结果验证

在路由配置页面，查看状态是否显示已发布 。

下线路由规则

前提条件

路由规则处于已发布状态。

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 路由配置 ；
6. 单击路由配置列表页操作列下线按钮；

结果验证

在路由配置页面，查看状态是否显示未发布。

删除路由规则

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 路由配置 ；
6. 单击路由配置列表页操作列删除按钮；

结果验证

在路由配置页面，查看路由规则是否已被删除。

查看路由详情

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 -> 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 路由配置 ；
6. 单击路由配置列表页操作列查看按钮可以看到路由名称、匹配规则、后端服务、监控、路由策略等信息

路由定义

路由详情页包含信息说明如下：

详情信息	说明
基本信息	路由名称、创建时间、匹配规则等。
历史记录	每次发布路由会产生一条历史记录，可以查看历史版本或者基于历史版本拷贝路由配置。
目标服务	路由目标服务信息。
基础监控	路由级基础监控，包括QPS、请求成功率和p95耗时。
已绑插件	当前路由已经绑定的插件列表。
授权信息	当前路由已授权信息列表。

历史记录

记录了当前路由的发布历史，可以用于审计或者基于历史发布的版本创建新路由等。

已绑插件

可以在此页查看当前路由绑定的插件，进行绑定插件和解除绑定操作，详见路由级插件

授权信息

对于开启了消费者访问鉴权的路由，可以在此页查看已经授权的消费者列表，添加授权和解除授权操作，详见章节如何授权消费者调用API。

路由策略配置

概述

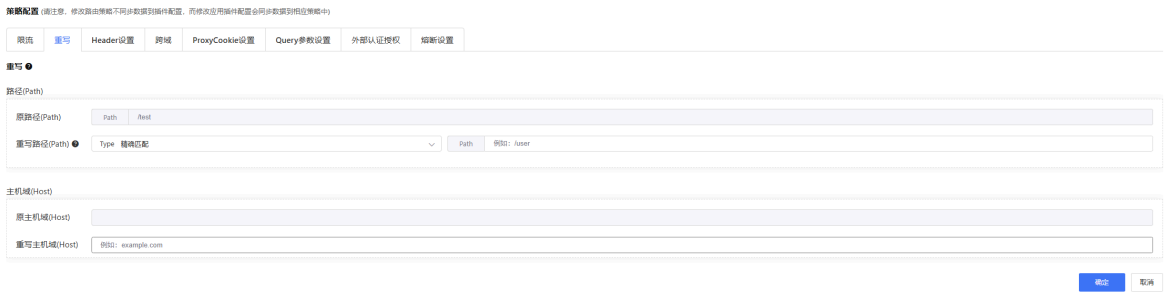
在路由详情页除了路由基础信息之外，还可以配置路由策略，当前支持路由级限流、重写、Header配置、跨域、Cookie重写、外部认证授权和熔断等策略配置

限流

当前实现为单机限流，基于时间窗口实现，可以配置时间窗口大小（秒）以及在一个时间窗口内限制的请求数。

重写

重写策略可以实现请求向上游转发请求时重写path和host，配置如下：



配置参数说明如下：

配置	说明
重写路径匹配类型	支持精确匹配、前缀匹配和正则匹配，只有路径匹配的请求才会对路径进行重写。
重写路径	重写的目标路径。
重写主机域名	重写的目标主机域名。

Header配置

Header配置支持对请求和响应的头部做修改，配置如下：

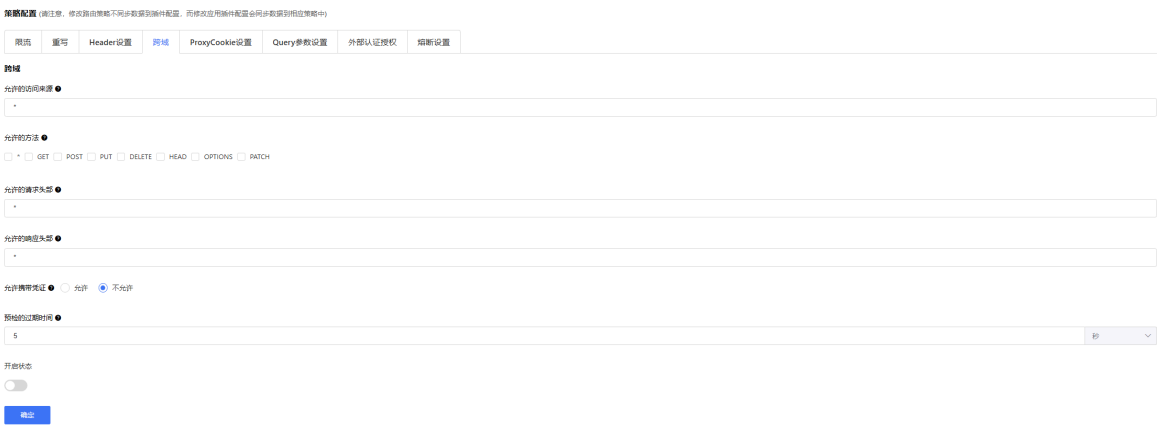
配置参数说明：

配置	说明
开启状态	开启时策略才生效。
Header类型	网关与后端交互时支持对请求和应答的头部做修改。

配置	说明
操作类型	支持新增、修改、删除操作。新增：若header key已存在，则在末尾追加header value；否则新增。修改：若header key不存在，则新增header kv；否则覆盖已有header value值。删除：若header key存在，则删除；否则忽略该header key。
Header Key	头部Key。
Header Value	头部Value。

跨域设置

云原生网关支持路由级别的跨域资源共享（CORS），配置如下：



CORS配置说明如下：

配置项	说明
允许访问的来源	作用于Access-Control-Allow-Origin头部，格式如：scheme://host:port，比如: https://foo.ctyun.com:8080，多个值使用','分割，'*'表示所有Origin均允许通过。
允许的方法	作用于Access-Control-Allow-Methods头部，表示允许的访问方法。
允许的请求头部	作用于Access-Control-Allow-Headers头部，允许跨域访问时请求方携带哪些CORS规范以外的Header，多个值使用','分割，'*'来表示所有Header均允许通过。
允许的响应头部	作用于Access-Control-Expose-Headers头部，允许浏览器和js脚本访问的响应头部。
允许携带凭证	作用于Access-Control-Allow-Credentials头部。
预检的过期时间	作用于Access-Control-Max-Age头部。
开启状态	开启时才生效。

ProxyCookie配置

该配置支持对上游响应Set-Cookie头部重写，当前支持对Set-Cookie头部里的Domain和Path进行重写，具体配置如下：

配置项说明：

配置项	说明
proxy_cookie_domain匹配规则	匹配上游应答Set-Cookie头部的Domain字段，支持正则匹配。
proxy_cookie_domain替换值	如果匹配，Set-Cookie头部Domain字段将被替换成该配置值。
proxy_cookie_path匹配规则	匹配上游应答Set-Cookie头部的Path字段，支持正则匹配。
proxy_cookie_path替换值	如果匹配，Set-Cookie头部Path字段将被替换成该配置值。

Query参数设置

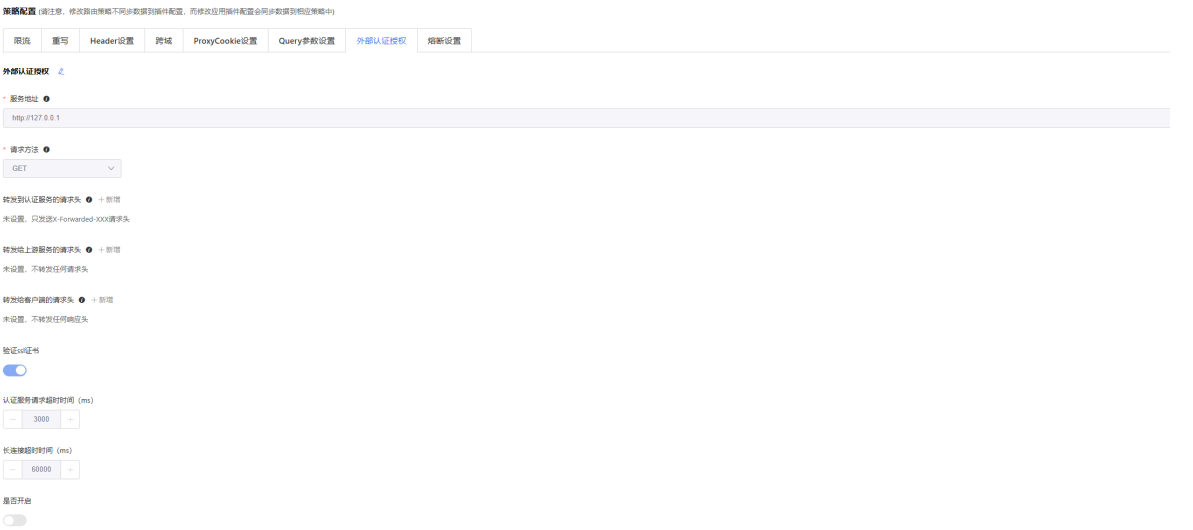
该配置支持对请求参数进行修改，具体配置如下：

配置项说明：

配置	说明
开启状态	开启时策略才生效。
操作类型	支持新增、修改、删除操作。新增：若请求参数已存在，则在末尾追加；否则新增。修改：若请求参数不存在，则新增该参数；否则覆盖已有参数值。删除：若请求参数存在，则删除；否则忽略该参数。
参数名	请求参数key。
参数值	请求参数Value。

外部认证授权

该配置支持通过第三方外部服务进行身份认证与授权。当身份认证失败时，可以实现自定义错误或者重定向到认证页面的场景。具体配置如下：

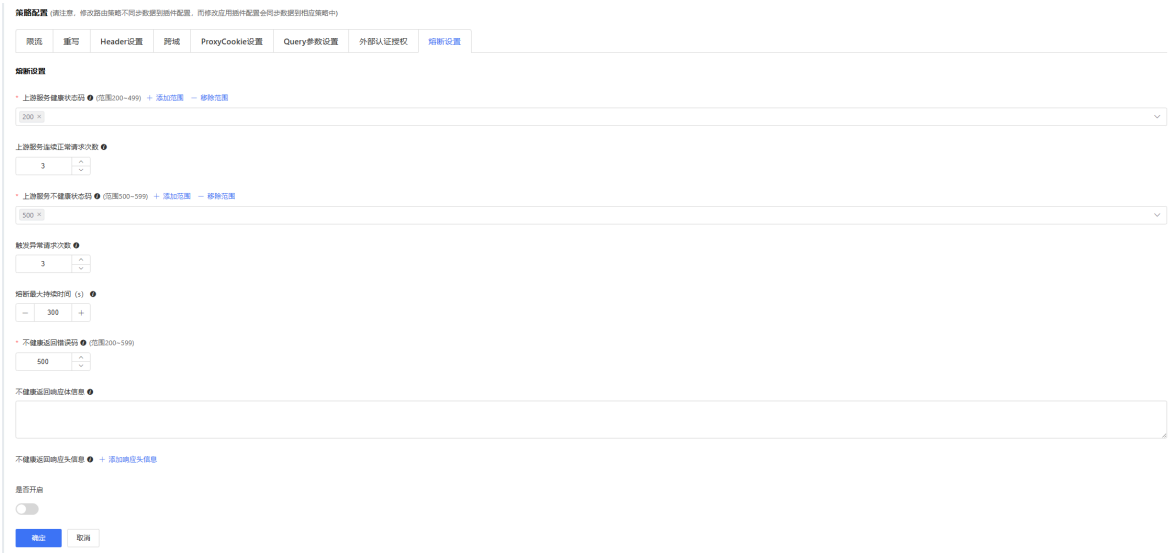


配置项说明

配置	说明
开启状态	开启时配置才生效。
服务地址	设置外部认证服务的地址（例如：https://localhost:9188）。
请求方法	客户端向认证服务发送请求的方法。当设置为POST时，会将请求体转发给认证服务。
转发到认证服务的请求头	设置需要由客户端转发给认证服务的请求头。如果没有设置，则只发送如X-Forwarded-XXX的请求头。
转发给上游服务的请求头	认证通过时，由认证服务转发给上游服务的响应头。如果不设置则不转发任何响应头。
转发给客户端的请求头	认证失败时，由认证服务向客户端发送的响应头。如果不设置则不转发任何响应头。
验证ssl证书	当开启时，验证SSL证书，默认开启。
认证服务请求超时时间	认证服务请求超时时间。
长连接超时时间	长连接超时时间。

熔断设置

该配置支持在触发上游服务不健康状态时进行熔断，从而保护上游业务服务。具体配置如下：



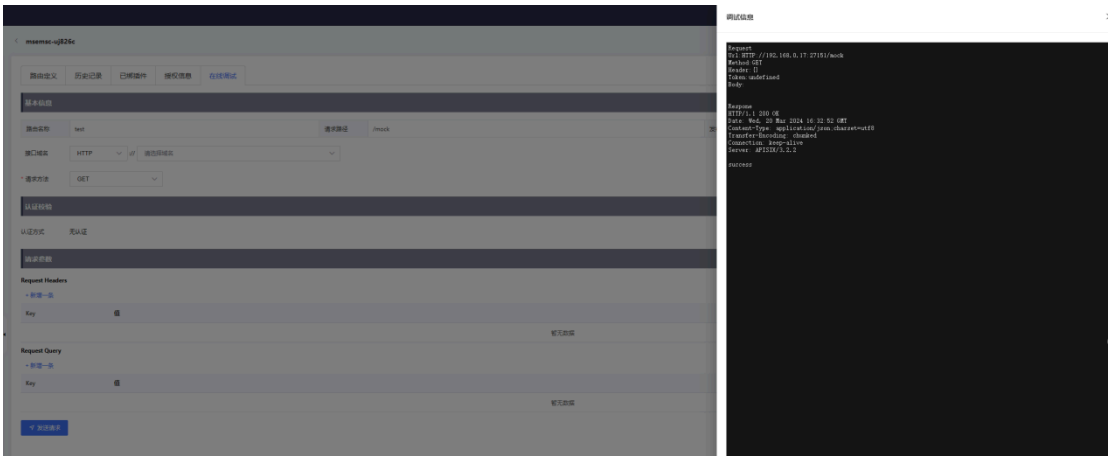
配置项说明

配置	说明
开启状态	开启时配置才生效。
上游服务健康状态码	上游服务处于健康状态时的HTTP状态码。
上游服务连续正常请求次数	上游服务触发健康状态的连续正常请求次数。
上游服务不健康状态码	上游服务处于不健康状态时的HTTP状态码。
触发异常请求次数	上游服务在一定时间内触发不健康状态的异常请求次数。
熔断最大持续时间	上游服务熔断的最大持续时间，以秒为单位。
不健康返回错误码	当上游服务处于不健康状态时返回的HTTP错误码。
不健康返回响应体信息	当上游服务处于不健康状态时返回的HTTP响应体信息。
不健康返回响应头信息	当上游服务处于不健康状态时返回的HTTP响应头信息。该字段仅在配置了不健康返回响应体信息时才生效。

路由在线调试

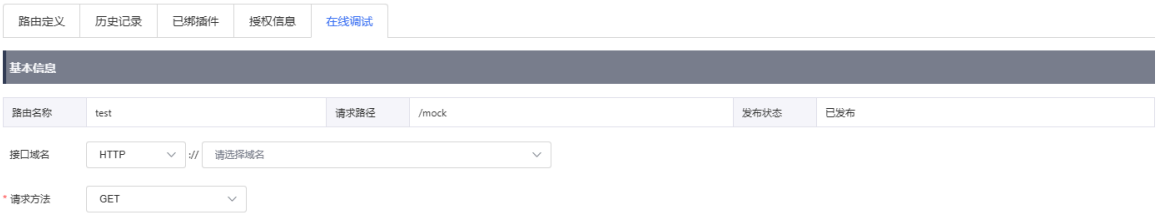
概述

在线调试为用户提供了快速便捷的路由调试能力，让用户能够简单高效的进行测试验证。路由调试配置页整体分为两个部分：左侧为路由请求配置页，包含请求方法、请求路径、请求参数等信息；右侧为路由调试结果展示窗口，包含请求信息、响应信息等



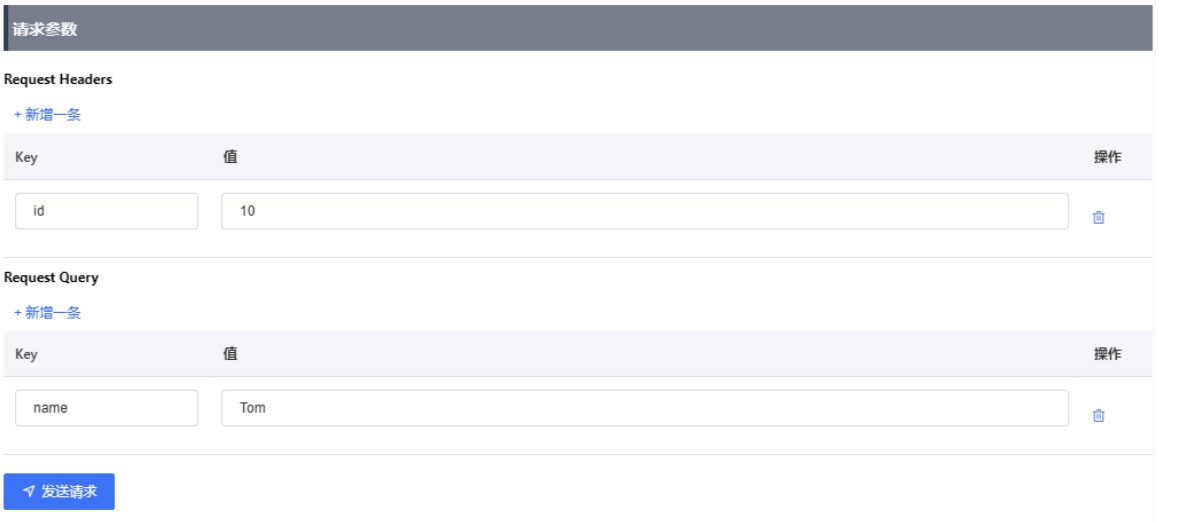
具体方法

- 选择不同的域名、请求方法灵活配置



- 若配置有授权关系，则可以选择不同的授权应用进行验证

- 添加请求参数、请求头部等信息进行调试



安全管理配置

设置IP黑白名单

概述

云原生网关支持通过配置ip黑名单和白名单的方式限制客户端访问网关；黑白名单不能同时开启，同时只有一种能生效。

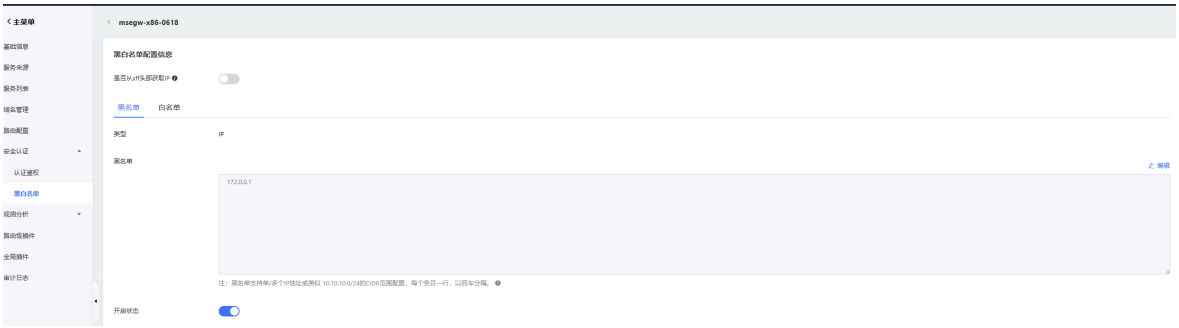
云原生网关默认读取请求中的Remote_addr字段值作为客户端IP（即网络层IP）；如果您的客户端访问出口存在七层代理，此时Remote_addr字段值为出口代理地址，可通过开启从xff头部获取ip配置选项，从X-Forwarded-For字段中获取客户端真实IP。

注意

通过ELB访问云原生网关时，网关看到的网络层地址是ELB出口IP；当前ELB采用四层代理模式，不会添加xff头部；我们将在后续的版本中优化ELB访问时的IP限制能力。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 在左侧导航栏，选择安全认证> 黑白名单。
4. 切换tab栏，填写黑名单或白名单。

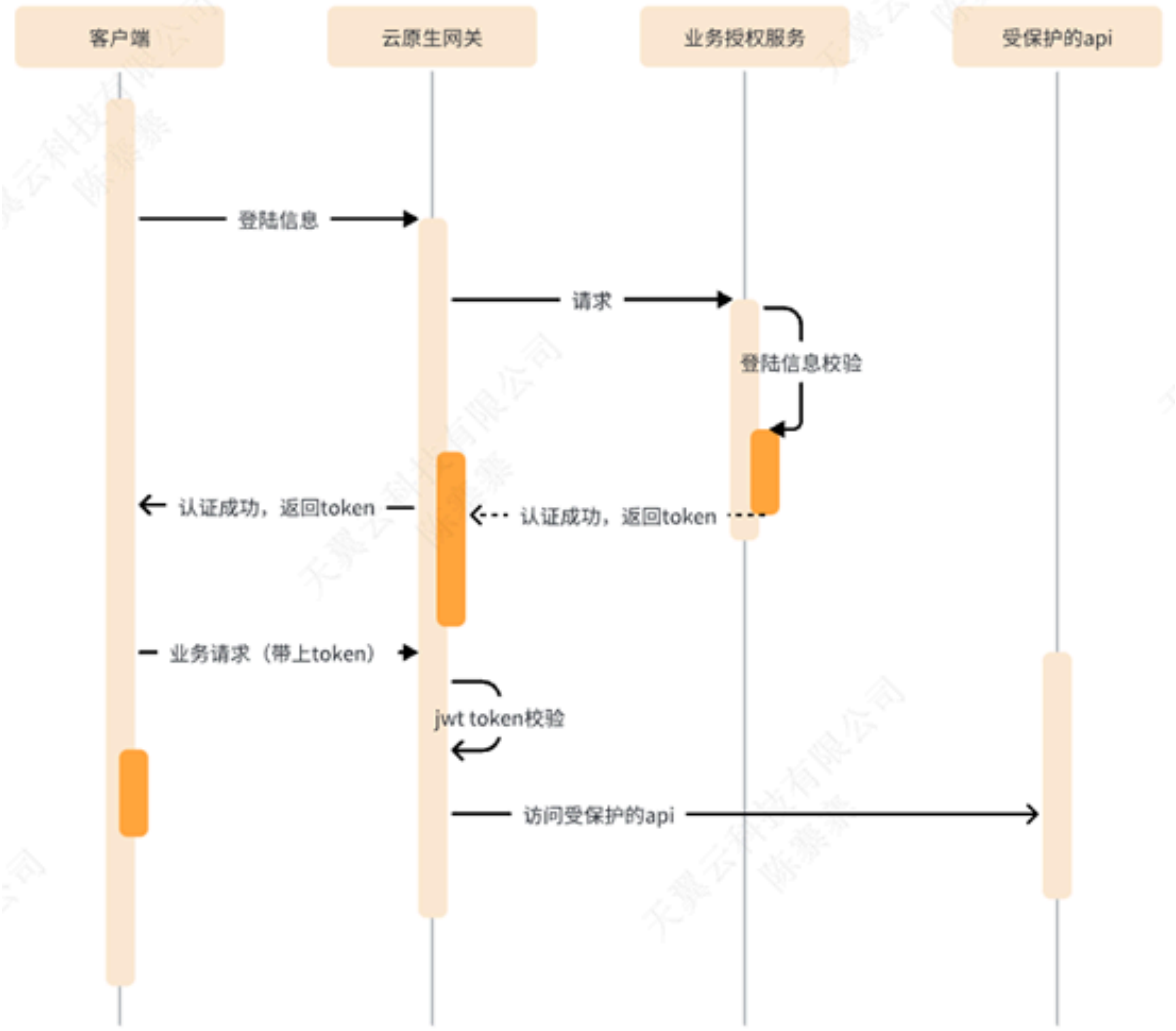


配置全局JWT认证策略

概述

JWT全称为 JSON Web Token，是一种开放标准（RFC 7519），它定义了一种在通信各方之间以JSON对象形式安全传输信息的方式。JWT可以使用HMAC、RSA、ECDSA等算法签名；在应用中，JWT可以用于用户身份认证和访问鉴权。云原生网关作为微服务的访问入口，是实现认证鉴权的理想节点，当前云原生网关支持JWT认证方式，交互流程如下：

1. JWT签发阶段，云原生网关对这类接口访问不做认证，将请求透传到业务授权服务。
2. 业务授权服务认证成功后签发JWT并返回给客户端，在后续的请求中客户端会带上JWT。
3. 云原生网关收到正常的业务请求时提取JWT并校验签名信息，校验通过则放过请求，否则返回401状态码。



JWT策略配置

云原生网关支持全局的JWT策略配置，在 安全认证 > 认证鉴权 菜单下，选择创建鉴权，鉴权类型选择JWT，填写相关参数即可创建JWT认证策略，JWT配置参数说明如下：

配置	说明
鉴权名称	唯一标识一个JWT鉴权配置，只能包含大小写字母、数字和‘-’，且不能以‘-’开头或者结尾。
加密算法	支持HS256、HS512、RS256。
密钥	选择HS256/ HS512算法时填写，需要指定是否base64编码，并提供密钥。
RSA密钥	选择RS256算法时填写，需要提供RSA算法公钥和私钥。
JWT Token配置	网关从请求获取JWT Token的位置，支持指定header、query和Cookie的key。

配置	说明
过期时间	JWT Token过期时间，默认86400秒。

配置全局OIDC认证策略

概述

OIDC全称为OpenID Connect，是一种基于OAuth 2.0协议的扩展。它允许客户端应用程序在获取访问令牌的同时也能获取关于最终用户身份的信息。OIDC的主要目的是为了进行用户身份验证，同时也可以传递用户属性。

OIDC协议中主要有四种角色

（1）资源所有者：这是指实际的用户，即系统中的真实个体，他们拥有访问受保护资源的权限。

（2）客户端：也被称为依赖方 (Relying Party)，是指希望访问资源的应用程序。客户端需要向认证服务器申请访问令牌和ID Token。

（3）认证服务器 (ID Provider)：负责处理用户的认证和授权请求。它会验证用户的身份，并在认证成功后发放访问令牌和ID Token给客户端。

（4）业务后端服务：存储受保护资源的服务器，这些资源只能通过有效的访问令牌访问。资源服务器会检查访问令牌的有效性，以决定是否允许客户端访问资源。

典型的OIDC工作流程通常如下：

（1）客户端应用向认证服务器请求授权，提供其客户端ID和其他参数。

（2）请求被重定向到认证服务器并由用户进行登录。

（3）登录成功后，用户会被重定向回客户端应用，并附带一个授权码（Authorization Code）。

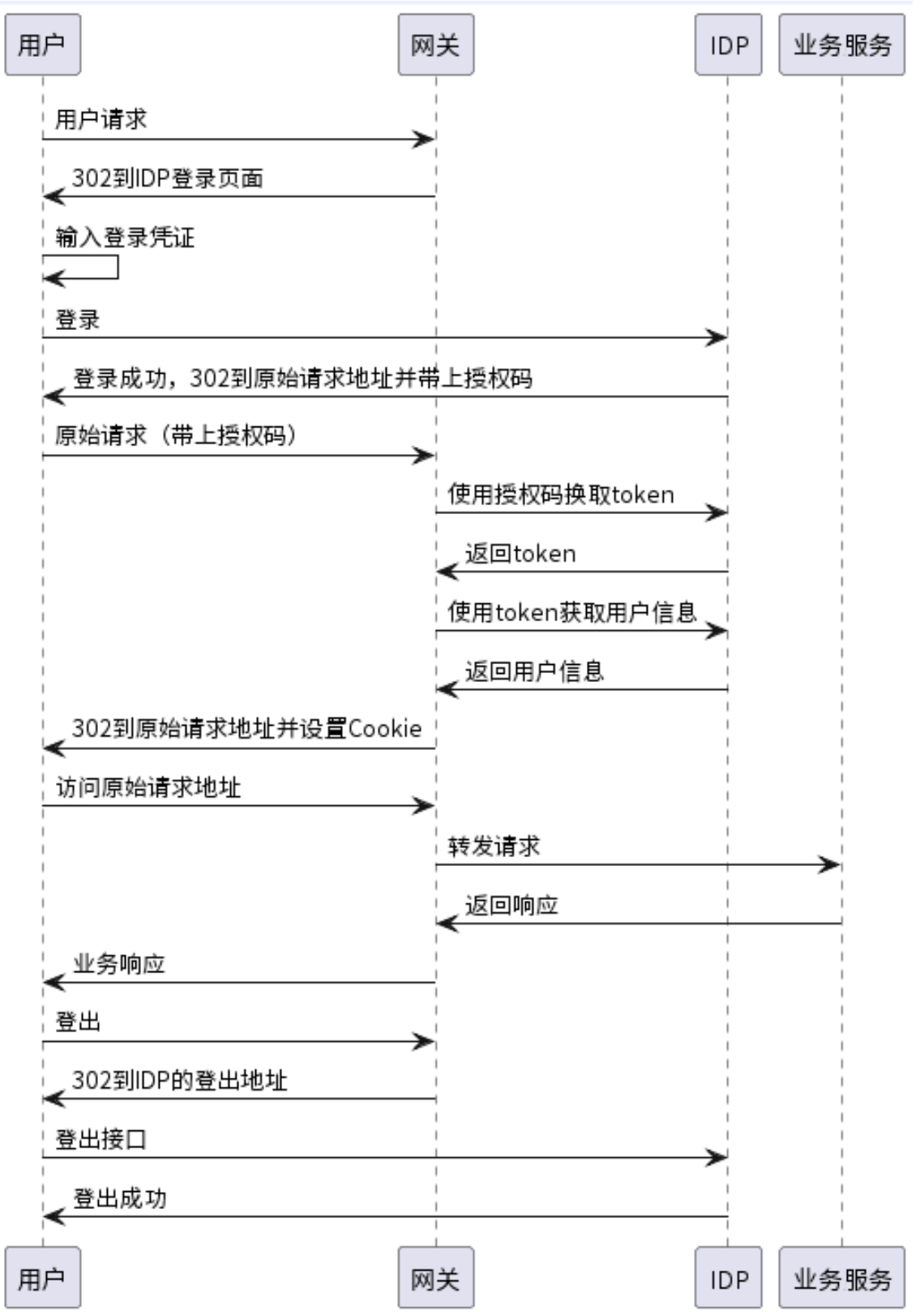
（4）客户端应用使用授权码向认证服务器请求访问令牌和ID Token。

（5）认证服务器返回访问令牌和ID Token给客户端应用。

（6）客户端应用可以使用访问令牌来访问受保护的资源，同时使用ID Token来进行用户身份验证。

OIDC广泛应用于单点登录（SSO）场景中，用户只需要在一个地方登录就可以访问多个不同的应用程序和服务。

在云原生网关中支持配置OIDC策略，云原生网关作为客户端实现OIDC认证，流程如下


用户
网关
IDP
业务服务
用户请求
302到IDP登录页面
输入登录凭证
登录
登录成功，302到原始请求地址并带上授权码
原始请求（带上授权码）
使用授权码换取token
返回token
使用token获取用户信息
返回用户信息
302到原始请求地址并设置Cookie
访问原始请求地址
转发请求
返回响应
业务响应
登出
302到IDP的登出地址
登出接口
登出成功
用户
网关
IDP
业务服务

OIDC策略配置

前置条件

1. 部署好身份认证服务（IDP）
2. 在IDP中为云原生网关申请客户端（clientid，client secret等）

OIDC策略配置

在 安全认证 > 认证鉴权菜单下，选择创建鉴权，鉴权类型选择OIDC，填写相关参数即可；参数说明如下

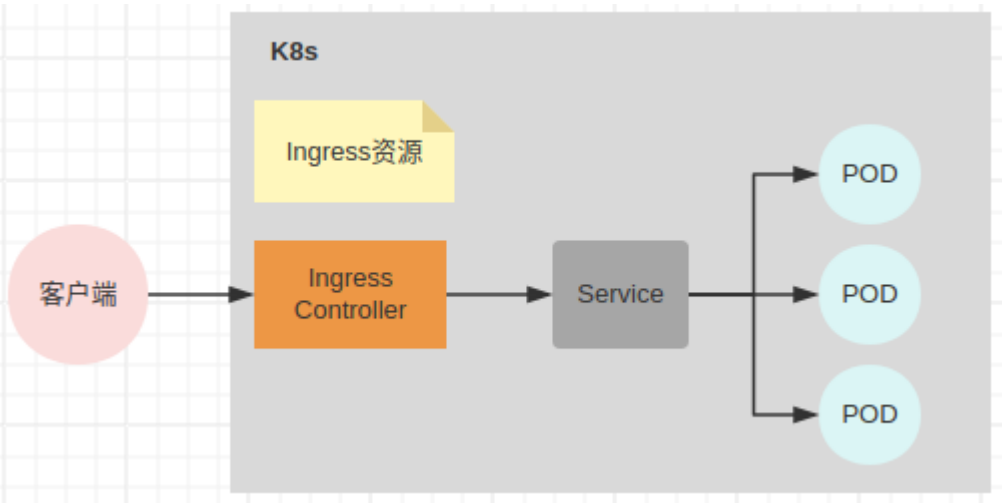
参数	说明
clientId	云原生网关作为OIDC客户端的clientid。
clientSecret	云原生网关作为OIDC客户端的client secret。
discovery	OIDC配置发现端点，通常以 .well-known/openid-configuration 的形式附加在 OpenID Provider 的基础 URL 上。
realm	对于一些IDP实现（如Keycloak），realm代表一个虚拟的边界或环境，每个realm内有自己独立的配置，用于实现逻辑上的隔离。
redirectUri	IDP认证成功后重定向的uri。
scope	IDP中定义的当前客户端（云原生网关）可以访问的范围，比如openid scope 表示请求用户的基本身份信息，而 profile 和 email 则分别表示请求更详细的用户资料和电子邮件地址。
bearerOnly	打开该选项时，网关只会对请求中的鉴权信息做校验。
sslVerify	网关是否校验IDP的证书信息。
setAccessTokenHeader	是否在转发到后端的请求头部中设置Access Token。
accessTokenInAuthorizationHeader	setAccessTokenHeader为true时生效，如果打开则在Authorization头部设置Access Token，否则在X-Access-Token头部设置Access Token。
setIdTokenHeader	打开时将在转发给后端的X-ID-Token头部设置id token。
setUserInfoHeader	打开时将在转发给后端的X-Userinfo头部设置user info。
logoutPath	登出路径。
timeout	网关和IDP通信的超时时间。
introspectionEndpoint	token校验端点。
introspectionEndpointAuthMethod	请求token校验端点的方法。
publicKey	token校验的公钥。
tokenSigningAlgValuesExpected	token签名校验算法。

Ingress管理

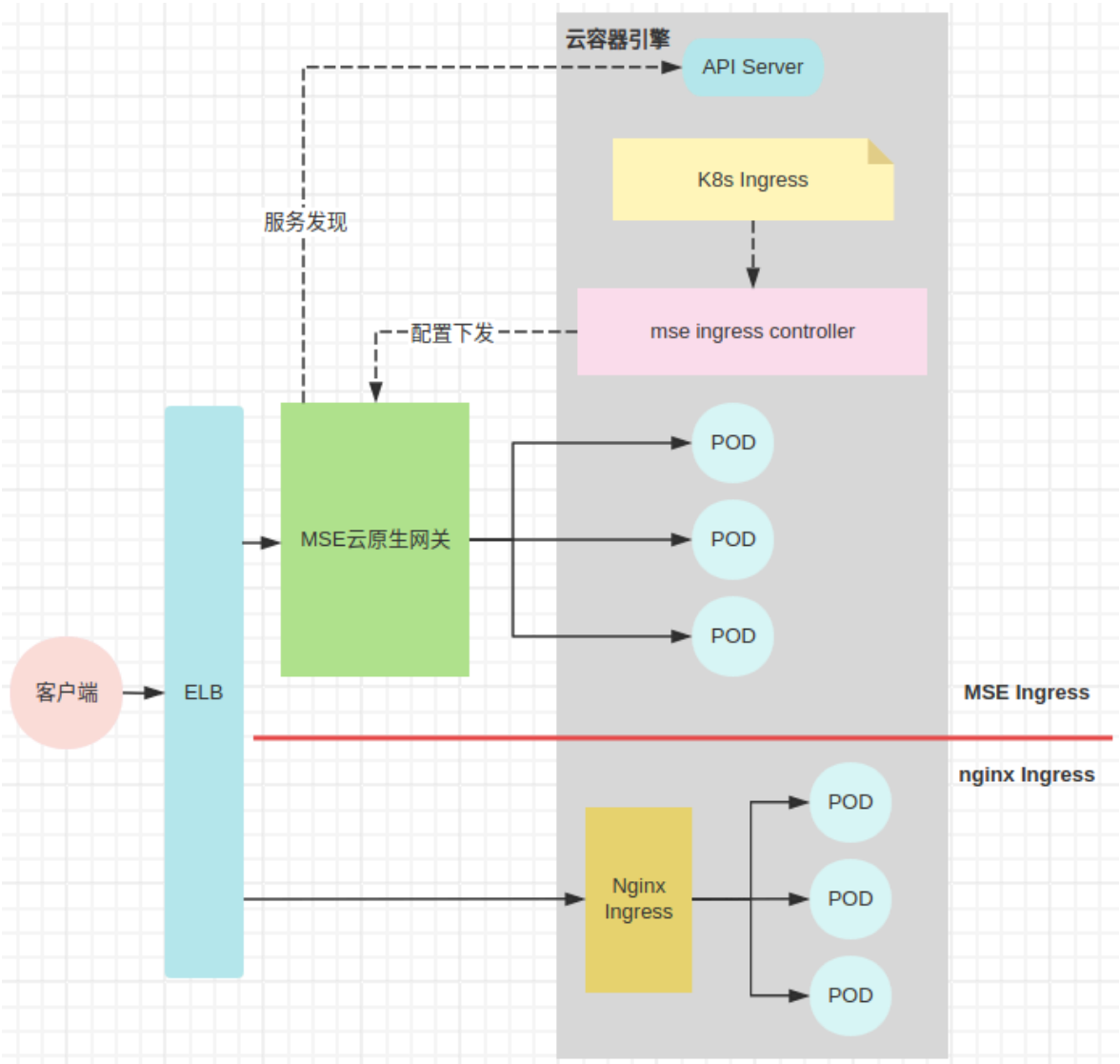
Ingress概述

概述

K8s Ingress用于暴露集群内部的服务给给外部访问，作为流量入口承载所有外部进入集群内部的流量；通过K8s Ingress资源可以配置外部流量访问集群内部服务的各种规则，实现流量路由、负载均衡等丰富的流量治理策略。除了路由策略配置，还需要数据面实际承载流量，执行Ingress中指定的路由策略，这里的数据面对应K8s Ingress controller，整体架构如下：



MSE云原生网关支持标准K8s Ingress，当前MSE云原生网关为虚机部署模式，通过在云容器引擎集群中部署mse ingress controller把Ingress资源转成网关配置资源并下发，整体架构如下：



组件功能说明

1. mse ingress controller：K8s controller，监听K8s Ingress及apisix Ingress资源，转成数据面配置并下发到数据面。
2. mse云原生网关：承载业务流量的数据面，接收控制台及mse ingress controller下发的配置指令。

注意

通过Ingress生成的配置不可以通过控制台修改

Ingress配置接入指南

前置条件

1. 开通云容器引擎集群
2. 开通MSE云原生网关实例，绑定公网ELB

3. 需要部署的镜像已经上传到镜像仓库

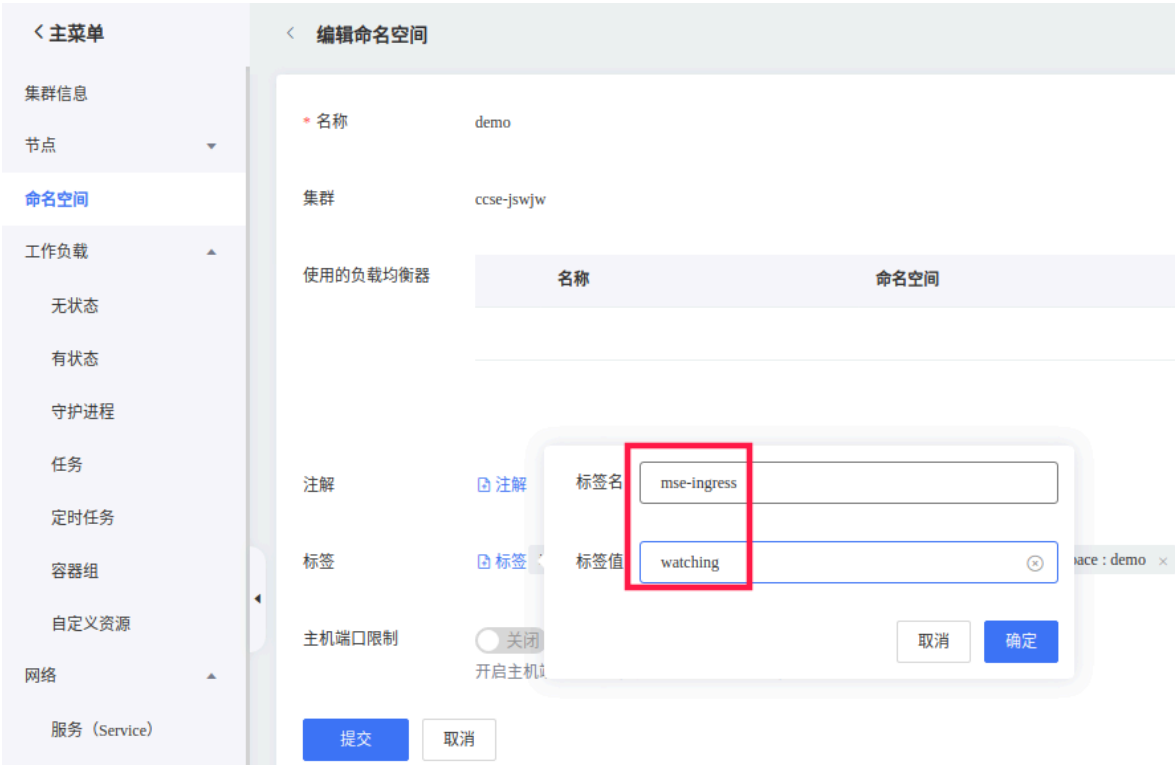
云原生网关配置

1. 配置云容器引擎服务来源，安装Ingress controller（注意配置我们要监听的命名空间的标签，这里我们只监听带有mse-ingress=watching的命名空间）

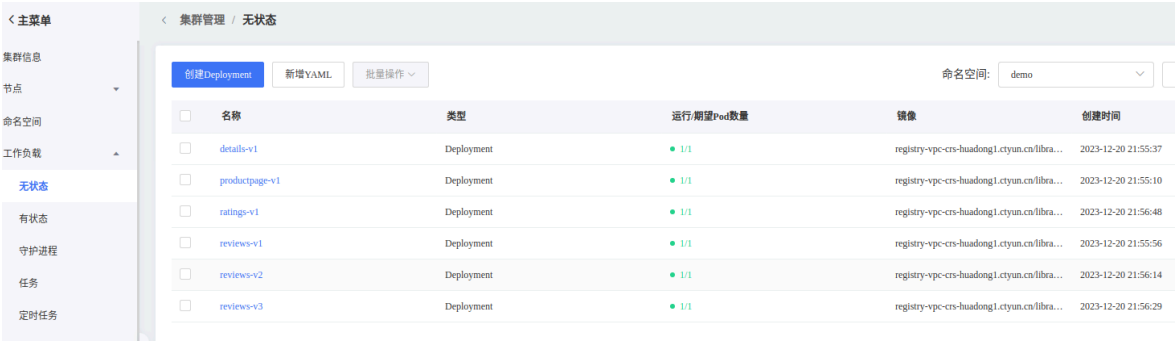
创建完成后可以看到云容器引擎上Ingress controller安装成功

2. 部署应用

给demo应用添加标签mse-ingress=watching



在demo命名空间下部署我们的测试应用（请确保镜像已经推送到镜像仓库），这里我们使用istio的bookinfo说明，部署完成后demo命名空间下的应用列表如下：



如果需要正常访问bookinfo应用需要配置以下路由规则：

路径精确匹配：/productpage，/login，/logout

路径前缀匹配：/static/，/api/v1/products/

访问后端productpage服务

其中/productpage 路径是我们应用的访问入口

此时我们访问云原生网关的ELB地址：http://121.229.70.242:27156/productpage

由于此时我们还没有创建服务和路由规则，访问会返回404错误

```
[redacted]:~$ curl http://121.229.70.242:27156/productpage
{"error_msg":"404 Route Not Found"}
```

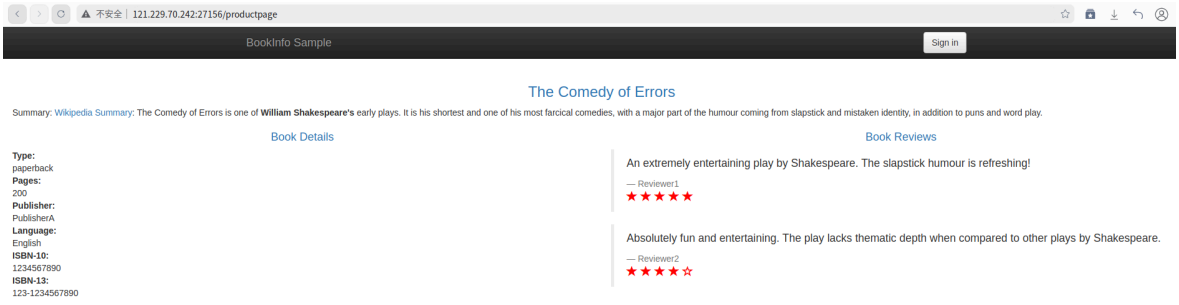
通过K8s Ingress实现路由管控

首先清除上一步创建的apisix ingress资源，在demo命名空间内创建K8s Ingress资源，定义如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: httpserver-ingress
 namespace: demo
spec:
 ingressClassName: mse
 rules:
 - http:
    paths:
    - backend:
       service:
         name: productpage
         port:
           number: 9080
      path: /productpage
      pathType: Exact
    - backend:
       service:
         name: productpage
```

```
      port:
        number: 9080
    path: /static/
    pathType: Prefix
  - backend:
      service:
        name: productpage
        port:
          number: 9080
    path: /login
    pathType: Exact
  - backend:
      service:
        name: productpage
        port:
          number: 9080
    path: /logout
    pathType: Exact
  - backend:
      service:
        name: productpage
        port:
          number: 9080
    path: /api/v1/products/
    pathType: Prefix
```

刷新浏览器，可以看到bookinfo应用正常运行：



监控分析

链路追踪

概述

在开启了链路追踪并配置采样率大于0，网关会根据采样率配置上报链路追踪数据，链路追踪基于traceid将调用链上下游串联起来，方便快速定位问题。

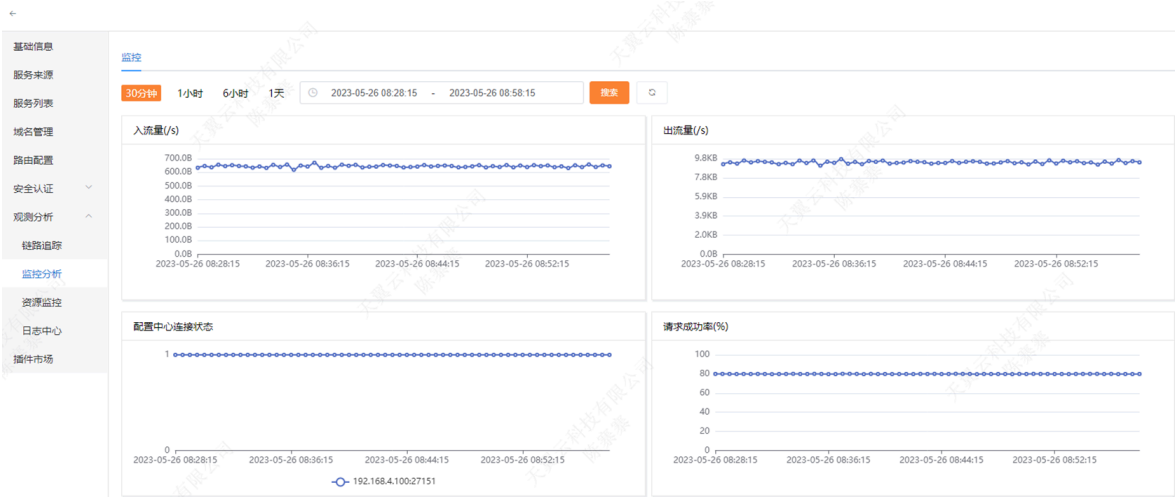
查看链路追踪数据

进入云原生网关控制台，选择对应的实例，进入观测分析 > 链路追踪页面查看调用链情况，当前支持根据请求信息（如接口，traceid等）查询链路追踪数据，点击一条链路数据可以看到调用链路的瀑布图，方便快速了解到本次调用涉及到哪些服务，调用关系如何，以及哪些环节出现异常（错误或者耗时过长等情况）。

监控分析

进入监控分析页面，可以看到监控指标统计，主要包括出入流量、请求成功率、4xx&5xx比例、分位耗时等，如下图所示：

出入流量、网关配置中心连接状态，请求成功率：

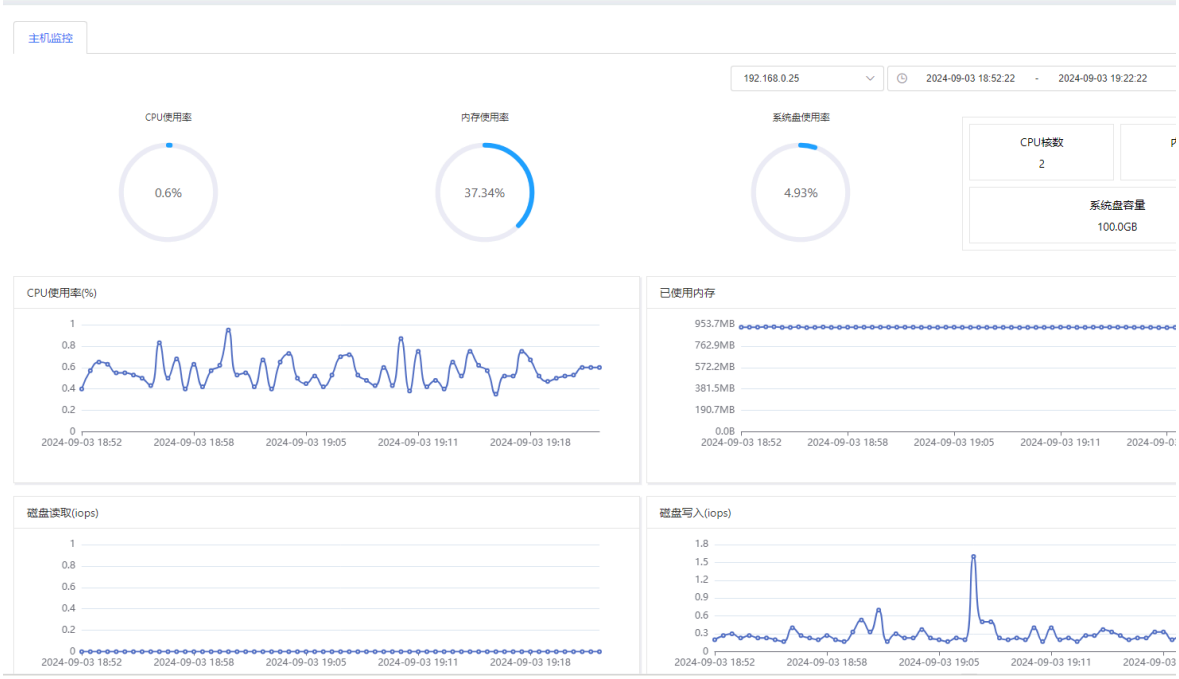


4xx、5xx比例，请求失败率，平均耗时

分位耗时和网关连接数统计

资源监控

资源监控可以看到网关实例的CPU、内存、网络、磁盘相关的指标监控，如下图如所示：



日志中心

概述

云原生网关对接天翼云日志服务（LTS）实现了访问日志采集、上报和查询能力；使用此功能前需要先开通云日志服务，同时在云原生网关控制台开启日志功能。

每个云原生网关实例在云日志服务中会创建一个日志项目，项目名称为MSEGW-${网关实例名称}，项目内有一个日志单元对应访问日志，单元名称为网关实例名称，您可以在云原生网关控制台 观测分析 > 日志中心 菜单 或者在云日志服务控制台查看网关访问日志。

访问日志格式说明

网关访问日志示例

{"start_time":1725411921254,"request":{"size":335,"method":"POST","uri":"/foo/bar","headers":{"accept":"text/plain, application/json, application/*+json, */*","content-length":"0","uber-trace-id":"b32bce4ff1f00f7b:899a1fd65c39be02:1c3372663d9eae03:0","connection":"keep-alive","user-agent":"Java/1.8.0_212","host":"foo.ctyun.com","content-type":"application/json"},"url":"http://foo.ctyun.com:27151/foo/bar","querystring":{}},"service_id":"a4db25fc03294d5dbeb9e7752381c972","server":{"version":"3.2.2","hostname":"msegw-vmxxxxxx"},"apisix_latency":0,"latency":24.00016784668,"client_ip":"100.89.x.x","response":{"size":625,"headers":{"via":"1.1 alb/v3.4.5","content-length":"427","date":"Wed, 04 Sep 2024 01:05:21 GMT","connection":"close","server":"MSEGW/3.2.2","content-type":"application/json;charset=UTF-8"},"status":400},"upstream":{"upstream_addr":"10.121.x.x:80","upstream_status":"400","upstream_latency":25,"upstream_name":"foo-service"},"route_id":"ddd342a2a3f34405bb8650ad4e","route_name":"test-route"}

访问日志字段说明如下

字段	说明
start_time	请求开始时间
request	请求信息
service_id	服务id
server	网关节点信息
apisix_latency	网关自身处理耗时（不包括上游服务耗时）
latency	总请求耗时（网关处理耗时和上游服务耗时之和）
client_ip	客户端IP
response	应答信息
upstream	上游信息，包括上游地址，上游返回的HTTP状态码，上游耗时；当服务访问异常时可以重点关注此字段，确认是否时上游服务出了问题。
route_id	路由id
route_name	路由名称

插件市场

插件概述

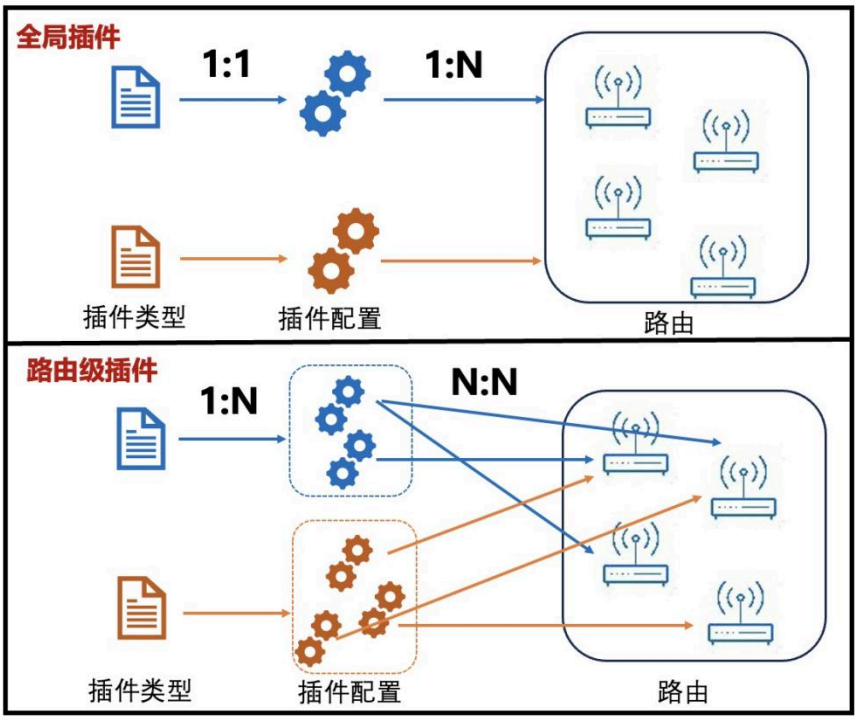
概述

云原生网关中提供了丰富的插件功能，可以满足用户特定的流量管理、可观测性、安全性能、请求/响应转换等需求，从而提高了网关的可扩展性，使得用户能够根据具体的应用场景进行灵活配置，实现高度个性化的云原生网关服务。

插件配置

目前云原生网关支持全局和路由级别两种粒度的插件配置：

1. 全局插件 ：应用于整个网关实例，对所有的路由都生效。一种插件类型只能有一个全局插件配置，应用于全体路由上。
2. 路由级插件 ：可灵活绑定在某一条路由上，只对特定的路由生效，允许根据具体的路由定制化功能。一种插件类型可以有多个不同的路由级插件配置，每个路由级插件配置可灵活应用于多条路由上。
3. 路由策略 ：也是由插件配置实现的，支持限流、重写等策略，应用在具体路由上。



当在路由上同时配置了某一种插件类型的全局插件、路由级插件和路由策略时，插件生效的优先级为：全局插件> 路由策略 = 路由级插件，即以全局插件中的配置为准。若在路由上只同时配置同一插件类型的路由级插件和路由策略，则以最新的插件配置为准。

全局插件

概述

全局插件可应用于整个网关实例，对所有API路由都生效，目前云原生网关提供了6种全局插件。

查看全局插件配置列表

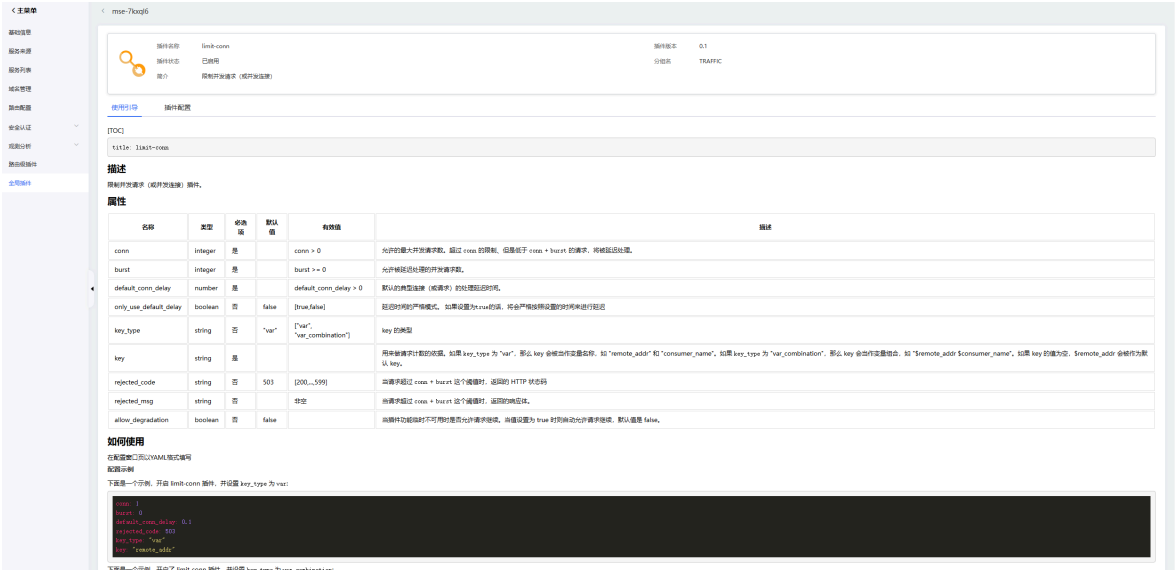
操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 您可以在网关列表页单击需要查看的网关实例ID或者 实例名称 ， 也可以单击操作列中的管理按钮；
5. 单击左侧导航栏 全局插件 ，可查看全部插件和已启用插件配置列表；

查看全局插件详情

操作步骤

1. 全部插件中展示了目前云原生网关所提供的全局插件列表；
2. 点击插件图标进入插件详情页，展示插件的基本信息，使用引导页面上展示详细的配置属性；

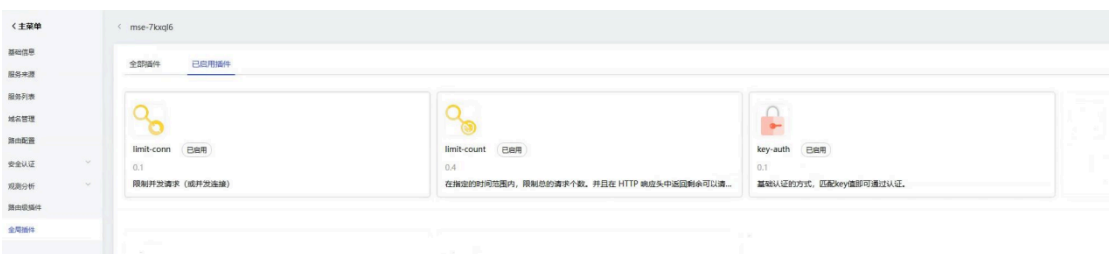


编辑全局插件配置

操作步骤

1. 从全部插件页面点击插件图标进入插件详情页；
2. 插件配置页面展示了当前插件的配置模板；
3. 点击编辑按钮，可对该插件的配置进行编辑修改；
4. 编辑完成后，点击保存按钮，即可保存该插件配置；
5. 若要放弃配置更改，可点击取消按钮；
6. 也可进入已开启插件页面，查看目前正在生效的插件配置，对已开启插件配置进行编辑；

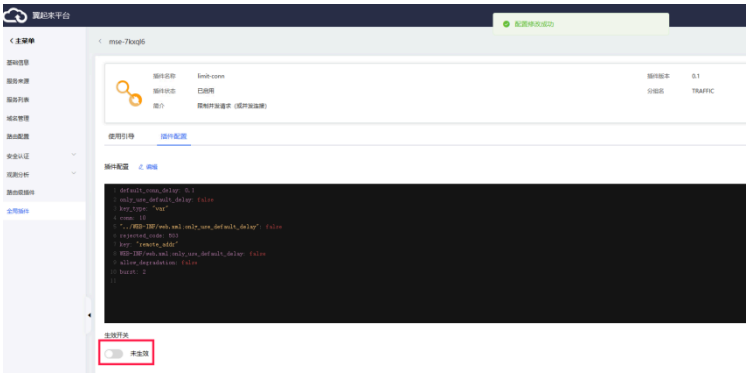

```
default_conn_delay: 0.1
only_use_default_delay: false
key_type: "var"
conn: 10
"../WEB-INF/web.xml;only_use_default_delay": false
rejected_code: 503
key: "remote_addr"
WEB-INF/web.xml;only_use_default_delay: false
allow_degradation: false
burst: 2
```

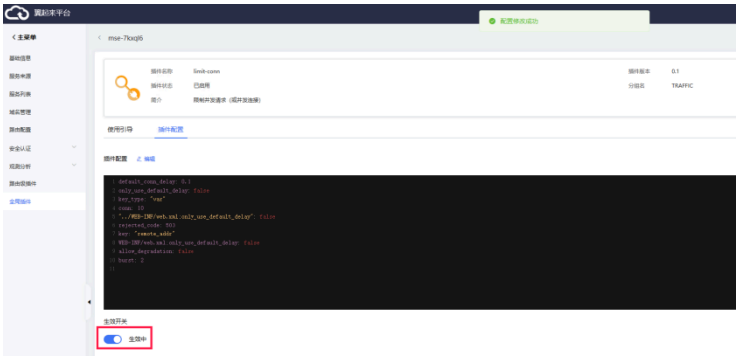


启停全局插件配置

操作步骤

1. 从全部插件页面点击插件图标进入插件详情页；
2. 插件配置页面展示了当前插件的配置模板；
3. 全局插件配置初始状态默认为关闭，显示 未生效 ；
4. 插件配置保存后，开启生效开关，则将下发该配置到所有路由上，显示生效中；
5. 若想关闭该插件配置，关闭生效开关，则取消该插件配置在所有路由上的作用，显示 未生效 ；
6. 也可进入已开启插件页面，查看目前正在生效的插件配置，然后启停生效开关；





路由级插件

插件类型及代码对照表

当要求subPluginType传入String时，填写pluginName值；当要求subPluginType传入Integer时，填写pluginCode值。

路由级插件	插件名称pluginName	插件代码pluginCode
cors	cors	14
basic-auth	basic-auth	28
key-auth	key-auth	30
proxy-cache	proxy-cache	38

路由级插件	插件名称pluginName	插件代码pluginCode
proxy-rewrite	proxy-rewrite	39
limit-conn	limit-conn	42
limit-count	limit-count	43
header-rewrite	header-rewrite	1003
proxy-cookie	proxy-cookie	1006
limit-count-by-client	limit-count-by-client	1010

插件说明与使用介绍

概述

目前云原生网关共支持8种插件配置，可以满足用户的身份认证、流量控制、安全、信息重写等需求。每个插件的配置详情与使用方式可参考以下介绍。

身份认证类

key-auth 插件

描述

key-auth 是一个认证插件，在header或者query中匹配key值即可通过认证

作用范围

该插件即可用于全局插件，也可用于路由级插件。全局插件配置的优先级高于路由级插件配置，当同时在某一路由上配置了key-auth的全局插件和路由级插件时，需带上全局插件配置中的key值才能通过。

属性

名称	类型	必选项	默认值	有效值	描述
key	string	必需	--	--	在header或者query中通过apikey携带key值进行认证。

如何启用

在配置窗口页以YAML格式填写

配置示例

下面是一个示例，填写key-auth配置信息:

“key”:”test”

启用/停用

在配置页面设置生效开关

验证插件

未启用插件时API请求结果

```
$ curl -i http://198.20.4.150:27151/apitest
HTTP/1.1 200
```

```
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Date: Fri, 23 Dec 2022 03:53:53 GMT
Server: APISIX/2.13.3
rspTest: hello,world

[{"id":1,"productID":1,"reviewer":"Reviewer1","text":"THIS IS A GRAY VERSION "}]
```

启用插件不带apikey时API的请求结果

```
$ curl -i http://198.20.4.150:27151/apitest
HTTP/1.1 401 Unauthorized
Date: Fri, 23 Dec 2022 03:54:31 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Server: APISIX/2.13.3

{"message":"Missing API key found in request"}
```

启用插件带apikey时API的请求结果

apikey的位置可以在header中也可以在query中

- 在header中使用

```
$ curl -i http://198.20.4.150:27151/apitest -H "apikey: test"
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Date: Fri, 23 Dec 2022 03:54:55 GMT
Server: APISIX/2.13.3
rspTest: hello,world

[{"id":1,"productID":1,"reviewer":"Reviewer1","text":"THIS IS A GRAY VERSION"}]
```

- 在query中使用

```
$ curl -i http://198.20.4.150:27151/apitest?apikey=test
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Date: Fri, 23 Dec 2022 03:55:28 GMT
Server: APISIX/2.13.3
rspTest: hello,world
```

[{"id":1,"productID":1,"reviewer":"Reviewer1","text":"An extremely entertaining play by Shakespeare"}]

basic-auth 插件

描述

Basic-auth 是一个认证插件，匹配指定的用户名和密码即可通过认证。

作用范围

该插件即可用于全局插件，也可用于路由级插件。全局插件配置的优先级高于路由级插件配置，当同时在某一路由上配置了basic-auth的全局插件和路由级插件时，需正确带上全局插件配置中的用户名和密码值才能通过。

属性

consumer 端配置

名称	类型	必选项	描述
username	string	必须	用户名
password	string	必须	用户的密码

如何启用

在配置窗口页以YAML格式填写

配置示例

填写basic-auth的配置信息

username:"xxx"

password:"********"

启用/停用

在配置页面设置生效开关

验证插件

未启用插件时API请求结果

```
$ curl -i http://198.20.4.150:27151/apitest
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Date: Fri, 23 Dec 2022 03:53:53 GMT
Server: APISIX/2.13.3
rspTest: hello,world
```

[{"id":1,"productID":1,"reviewer":"Reviewer1","text":"THIS IS A GRAY VERSION "}]

启用插件时API请求结果

- 缺少 Authorization header

```
$ curl -i http://198.20.4.150:27151/apitest
```

```
HTTP/1.1 401 Unauthorized
Date: Fri, 23 Dec 2022 04:23:56 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
WWW-Authenticate: Basic realm='.'
Server: APISIX/2.13.3

{"message":"Missing authorization in request"}
```

- 用户名不存在

```
$ curl -i -umse:msegW@9527 http://198.20.4.150:27151/apitest
HTTP/1.1 401 Unauthorized
Date: Fri, 23 Dec 2022 04:25:08 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Server: APISIX/2.13.3

{"message":"Invalid user key in authorization"}
```

- 密码错误

```
$ curl -i -umsegw:msegW http://198.20.4.150:27151/apitest
HTTP/1.1 401 Unauthorized
Date: Fri, 23 Dec 2022 04:25:49 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Server: APISIX/2.13.3

{"message":"Password is error"}
```

- 成功请求

```
$ curl -i -umsegw:msegW@9527 http://198.20.4.150:27151/apitest
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Date: Fri, 23 Dec 2022 04:26:16 GMT
Server: APISIX/2.13.3
rspTest: hello,world

[{"id":1,"productID":1,"reviewer":"Reviewer1","text":"An extremely entertaining play by Shakespeare.
"}]
```

流量控制类

并发限制limit-conn 插件

描述

限制并发请求（或并发连接）插件。

作用范围

该插件即可用于全局插件，也可用于路由级插件。全局插件配置的优先级高于路由级插件配置，当同时在某一路由上配置了limit-conn的全局插件和路由级插件时，以全局插件配置中设置的属性值为准。

属性

名称	类型	必选项	默认值	有效值	描述
conn	integer	是	--	conn > 0	允许的最大并发请求数。超过 conn 的限制、但是低于conn +burst 的请求，将被延迟处理。
burst	integer	是	--	burst >= 0	允许被延迟处理的并发请求数。
default_conn_delay	number	是	--	default_conn_delay > 0	默认的典型连接（或请求）的处理延迟时间。
only_use_ default_delay	boolean	否	false	[true,false]	延迟时间的严格模式。如果设置为true的话，将会严格按照设置的时间来进行延迟。
key_type	string	否	"var"	["var", "var_combination"]	key 的类型。
key	string	是	--	--	用来做请求计数的依据。如果key_type 为"var"，那么 key 会被当作变量名称，如 "remote_addr" 和 "consumer_name"。如果key_type 为"var_combination"，那么 key 会当作变量组合，如 "remote_addr consumer_name"。如果 key 的值为空，$remote_addr 会被作为默认 key。

名称	类型	必选项	默认值	有效值	描述
rejected_code	string	否	503	[200,...,599]	当请求超过conn+burst 这个阈值时，返回的HTTP状态码。
rejected_msg	string	否	--	非空	当请求超过conn+burst 这个阈值时，返回的响应体。
allow_degradation	boolean	否	false	--	当插件功能临时不可用时是否允许请求继续。当值设置为true 时则自动允许请求继续，默认值是false。

如何使用

在配置窗口页以YAML格式填写

配置示例

下面是一个示例，开启 limit-conn 插件，并设置 key_type 为var:

```
conn: 1
 burst: 0
 default_conn_delay: 0.1
 rejected_code: 503
 key_type: "var"
 key: "remote_addr"
```

下面是一个示例，开启了 limit-conn 插件，并设置 key_type 为var_combination:

```
conn: 1
 burst: 0
 default_conn_delay: 0.1
 rejected_code: 503
 key_type: "var_combination"
 key: "$consumer_name $remote_addr"
```

停用/启用

在配置页面设置生效开关

验证插件

上面启用的插件的参数表示只允许一个并发请求。当收到多个并发请求时，将直接返回 503 拒绝请求。

```
 curl -i http://127.0.0.1:9080/index.html?sleep=20

 503 Service Temporarily Unavailable
```

```
# 503 Service Temporarily Unavailable

---

openresty
```

这就表示 limit-conn 插件已经生效了。

限流limit-count 插件

描述

在指定的时间范围内，限制总的请求个数。并且在 HTTP 响应头中返回剩余可以请求的个数，支持本地限流和全局限流两种限流方式。

当前配置模版中配置了三种模版可供选择：

- 基础配置为本地限流策略，云原生网关每个节点单独计算限流次数。整个集群的限流计数为配置的count 节点数。
- Redis单节点配置和Redis集群配置两种配置都是云原生网关的全局限流配置，此时count配置为集群全局限流计数。

作用范围

该插件即可用于全局插件，也可用于路由级插件。全局插件配置的优先级高于路由级插件配置，当同时在某一路由上配置了limit-count的全局插件和路由级插件时，以全局插件配置中设置的属性值为准。

属性

名称	类型	必选项	默认值	有效值	描述
count	integer	必须	--	count > 0	指定时间窗口内的请求数量阈值。
time_window	integer	必须	--	time_window > 0	时间窗口的大小（以秒为单位），超过这个时间就会重置。
key_type	string	可选	"var"	["var", "var_combination", "constant"]	key 的类型。
key	string	可选	"remote_addr"	--	用来做请求计数的依据，详情参见key的使用小节。如果 key 的值为空，$remote_addr 会被作为默认 key。

名称	类型	必选项	默认值	有效值	描述
rejected_code	integer	可选	503	[200,...,599]	当请求超过阈值被拒绝时，返回的HTTP 状态码。
rejected_msg	string	可选	--	非空	当请求超过阈值被拒绝时，返回的响应体。
allow_degradation	boolean	可选	false	--	当限流插件功能临时不可用时（例如，Redis 超时）是否允许请求继续。当值设置为 true 时则自动允许请求继续，默认值是 false。
show_limit_quota_header	boolean	可选	true	--	是否在响应头中显示X-RateLimit-Limit和X-RateLimit-Remaining （限制的总请求数和剩余还可以发送的请求数），默认值是true。
group	string	可选	--	非空	配置同样的group的 Route 将共享同样的限流计数器。
redis_host	string	当policy为redis时必填	--	--	当使用redis 限速策略时，该属性是 Redis 服务节点的地址。
redis_port	integer	可选	6379	[1,...]	当使用redis 限速策略时，该属性是 Redis 服务节点的端口。
redis_password	string	可选	--	--	当使用redis 或者redis-cluster 限速策略时，该属性是 Redis 服务节点的密码。
redis_timeout	integer	可选	1000	[1,...]	当使用redis 或者redis-cluster 限速策略时，该属性是 Redis 服务节点以毫秒为单位的超时时间。

名称	类型	必选项	默认值	有效值	描述
redis_cluster_nodes	array	当policy 为 redis-cluster 时必填	--	--	当使用redis-cluster 限速策略时，该属性是 Redis 集群服务节点的地址列表（至少需要两个地址）。
redis_cluster_name	string	当policy 为 redis-cluster 时必填	--	--	当使用redis-cluster 限速策略时，该属性是 Redis 集群服务节点的名称。

count说明

- 当使用本地计数策略，即policy设置为local时，count数值为每个网关节点的限流计数。整个集群的限流计数=count 节点数；
- 当使用redis时，即policy设置为redis或redis-cluster时，count记录在redis中，表示整个集群的全局限流计数。

key的使用说明

用来做请求计数的有效值。

key的类型

key的取值类型使用key_type标识，key_type支持三种值："var", "var_combination", "constant"

- key_type为"constant"时，那么 key 会被当作常量。
- key_type为"var"时， key 会被当作变量名称。
- key_type 为 "var_combination"，那么 key 会当作变量组。比如如果设置 "remote_addr consumer_name" 作为 key，那么插件会同时受 remote_addr 和 consumer_name 两个变量的约束。

例如，可以使用主机名（或服务器区域）作为关键字，以便限制每个主机名规定时间内的请求次数。我们也可以使用客户端地址作为关键字，这样我们就可以避免单个客户端规定时间内多次的连接我们的服务。

当前接受的 key如下：

key	key_type为var时填写	key_type为var_combination时填写
"remote_addr"（客户端 IP 地址）	remote_addr	$remote_addr
"server_addr"（服务端 IP 地址）	server_addr	$server_addr
请求头中的值"X-Forwarded-For"	http_x_forwarded_For	$http_x_forwarded_For
请求头中的值"X-Real-IP"	http_x_real_ip	$http_x_real_ip
"consumer_name"（consumer 的 username）	consumer_name	$consumer_name
"service_id"	service_id	$service_id

如何使用

在配置窗口页以YAML格式填写

配置示例

基于请求上下文和自定义表达式的限流配置示例。

通过将key_type设置为var或var_combination来支持基于请求上下文和自定义表达式的限流。支持各种请求参数和内置的nginx变量,以及变量组合。

常用参数示例：

remote_addr ： 客户端ip

consumer_name：消费者名称；其他nginx变量也支持

arg_name:url参数，name表示参数名称

http_name:请求header参数

cookie_name:请求的cookie参数

下面是一个示例，开启了limit count 插件，并设置key_type 为var，key设置为remote_addr表示基于客户端ip进行请求限流：

```
count:2
 time_window:60
 rejected_code:503
 key_type:"var"
 key:"remote_addr"
```

下面是一个示例，开启了limit count 插件，并设置key_type 为var，key设置为arg_userId，可对不同的url参数userId分别进行限流计数：

```
"allow_degradation":false,
 "count":20,
 "key":"arg_userId",
 "key_type":"var",
 "policy":"local",
 "rejected_code":503,
 "show_limit_quota_header":true,
 "time_window":30
```

下面是一个示例，开启了limit count 插件，并设置key_type 为var_combination,key值设置为consumer_name remote_addr，表示同时基于消费者名称和客户端ip进行限流计数。

```
count:2

time_window:60

rejected_code:503

key_type:"var_combination"
```

key: "$consumer_name $remote_addr"

下面是一个示例，开启了limit count 插件，并设置key_type 为constant：

基于常量的限流配置示例

通过设置key_type 为constant，key 的值将会直接作为常量来处理

count: 2

time_window: 60

rejected_code: 503

key_type: "constant"

key: "remote_addr"

全局限流配置示例

在页面上选择Redis单节点配置或者Redis集群配置，并设置redis相关参数即可开启全局限流配置。

Redis可选择通过平台开通redis实例，或者自行部署，并在配置中填入连接信息即可。

配置示例：

```
# 时间窗口内的请求数量阈值
# [必填]特别地，当使用redis时，表示整个集群的请求计数。
count: 30
# [必填]时间窗口的大小（以秒为单位）
time_window: 60
# [可选]请求超过阈值被拒绝时，返回的 HTTP 状态码
rejected_code: 503
# [可选]key 的类型
key_type: "var"
# [可选]用来做请求计数的依据
key: "remote_addr"
# [可选]当设置rejected_msg时，非空。默认可不填
# rejected_msg: "Requests are too frequent, please try again later."
# [可选]当限流插件功能临时不可用时（例如，Redis 超时）是否允许请求继续。当值设置为 true 时则自动允许请求继续，默认值是 false
allow_degradation: false
# [可选]是否在响应头中显示 X-RateLimit-Limit 和 X-RateLimit-Remaining （限制的总请求数和剩余还可以发送的请求数），默认值是 true
show_limit_quota_header: true
# [必填]速率限制策略，使用单节点redis
policy: "redis"
# [必填]redis服务地址
redis_host: "127.0.0.1"
# [可选]redis服务端口
redis_port: 6379
```

```
# [可选]redis服务密码
redis_password: "password"
# [可选]redis服务的datebase
redis_database: 1
# [可选]redis服务的超时时间，单位毫秒
redis_timeout: 1000
```

停用/启用

在配置页面设置生效开关

验证插件

上述配置限制了 60 秒内只能访问 2 次，前两次访问都会正常访问：

```
curl -i http://127.0.0.1:9080/index.html
```

响应头里面包含了X-RateLimit-Limit 和X-RateLimit-Remaining，他们的含义分别是限制的总请求数和剩余还可以发送的请求数：

```
HTTP/1.1 200 OK
 Content-Type: text/html
 Content-Length: 13175
 Connection: keep-alive
 X-RateLimit-Limit: 2
 X-RateLimit-Remaining: 0
 Server: APISIX web server
```

当你第三次访问的时候，就会收到包含 503 返回码的响应头：

```
HTTP/1.1 503 Service Temporarily Unavailable
 Content-Type: text/html
 Content-Length: 194
 Connection: keep-alive
 Server: APISIX web server


 503 Service Temporarily Unavailable

 # 503 Service Temporarily Unavailable


 ---

openresty
```

同时，如果你设置了属性rejected_msg 的值为"Requests are too frequent, please try again later."，当你第三次访问的时候，就会收到如下的响应体：

```
HTTP/1.1 503 Service Temporarily Unavailable
Content-Type: text/html
Content-Length: 194
Connection: keep-alive
Server: APISIX web server

{"error_msg":"Requests are too frequent, please try again later."}
```

这就表示limit count 插件生效了。

结果缓存proxy-cache插件

描述

proxy-cache 插件提供了缓存后端响应数据的能力，可以根据响应码和请求模式等属性来指定需要缓存的数据。

作用范围

该插件目前只支持用于路由级插件。

属性

名称	类型	必选项	默认值	有效值	描述
cache_key	array[string]	可选		["$host", "$request_uri"]	缓存key，可以使用变量。例如：["$host", "$request_uri"]。
cache_bypass	array[string]	可选			当该属性的值不为空或者非0时则会跳过缓存检查，即不在缓存中查找数据，可以使用变量，例如：["$arg_bypass"]。
cache_method	array[string]	可选	["GET", "HEAD"]	["GET", "POST", "HEAD"]	根据请求method决定是否需要缓存。
cache_http_status	array[integer]	可选	[200, 301, 404]	[200, 599]	根据HTTP响应码决定是否需要缓存。
hide_cache_headers	boolean	可选	false		当设置为true时将Expires和Cache-Control响应头返回给客户端。
cache_control	boolean	可选	false		当设置为true时遵守HTTP协议规范中的Cache-Control的行为。

名称	类型	必选项	默认值	有效值	描述
no_cache	array[string]	可选			当此参数的值不为空或非0时将不会缓存数据，可以使用变量。
cache_ttl	integer	可选	300秒		当选项cache_control未开启或开启以后服务端没有返回缓存控制头时，提供的默认缓存时间。

如何启用

在配置窗口页以YAML 格式填写

配置示例

下面是一个示例，开启proxy-cache 插件，并配置了一些属性。表示对于路径为“/hello”的路由，在60秒内，对于返回结果为200的GET请求将返回缓存结果，其中缓存的key为请求uri+"-cache-id"的拼接字符串。

```
curl http://127.0.0.1:9180/apisix/admin/routes/1 \
-H 'X-API-KEY: edd1c9f034335f136f87ad84b625c8f1' -X PUT -d '
{
   "plugins": {
      "proxy-cache": {
         "cache_key":  ["$uri", "-cache-id"],
         "cache_bypass": ["$arg_bypass"],
         "cache_method": ["GET"],
         "cache_http_status": [200],
         "hide_cache_headers": true,
         "cache_control": false,
      "cache_ttl": 60
      }
   },
   "upstream": {
      "nodes": {
         "127.0.0.1:1999": 1
      },
      "type": "roundrobin"
   },
   "uri": "/hello"
}'
```

启用/停用

在路由上绑定/解绑插件

验证插件

在路由上绑定结果缓存插件后，第一次请求Apisix-Cache-Status状态为MISS

```
$curl -i 127.0.0.1:27151/api/1/reviews
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Apisix-Cache-Status: MISS
Date: Mon, 18 Mar 2024 12:02:07 GMT
Server: APISIX/2.13.3

[{"id":1,"productId":1,"reviewer":"Reviewer1","text":"This is the 1st reviewer"}]
```

再次请求，命中缓存，Apisix-Cache-Status状态为HIT

```
$ curl -i 127.0.0.1:27151/api/1/reviews
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Date: Mon, 18 Mar 2024 12:02:07 GMT
Server: APISIX/2.13.3
Age: 3
Apisix-Cache-Status: HIT

[{"id":1,"productId":1,"reviewer":"Reviewer1","text":"This is the 1st reviewer"}]
```

安全限制类

跨域CORS 插件

描述

cors 是一个跨域访问插件。

作用范围

该插件即可用于全局插件，也可用于路由级插件。全局插件配置的优先级高于路由级插件配置，当同时在某一路由上配置了cores的全局插件和路由级插件时，以全局插件配置中设置的属性值为准。

属性

名称	类型	必选项	默认值	描述
allow_origins	string	可选	"*"	允许跨域访问的来源，作用于Access-Control-Allow-Origin 头部，格式如：scheme://host:port，比如:https://ctyun.com:8081。多个值时使用","分割，不允许携带凭证时可以使用"*"来表示所有Origin均允许通过。
allow_methods	string	可选	"*"	允许跨域访问的方法，作用于Access-Control-Allow-Methods 头部。多个值时使用","分割，支持“GET,POST,PUT,DELETE,HEAD,O方法。不允许携带凭证时可以使用"*"来表示所有 Method 均允许通过。
allow_headers	string	可选	"*"	允许跨域访问时请求方携带哪些非CROS 规范以外的 Header，多个值时使用","分割，不允许携带凭证时可以使用"*"来表示所有 Header 均允许通过。
expose_headers	string	可选	"*"	允许跨域访问时响应方携带哪些非CROS 规范以外的 Header，多个值时使用","分割，不允许携带凭证时可以使用"*"来表示所有 Header 均允许通过。
max_age	integer	可选	5	浏览器缓存CORS 结果的最大时间，单位为秒，在这个时间范围内浏览器会复用上一次的检查结果，-1 表示不缓存。作用于Access-Control-Max-Age 头部。

名称	类型	必选项	默认值	描述
allow_credential	boolean	可选	false	是否允许携带凭证，作用于Access-Control-Allow-Credentials 头部。

如何启用

在配置窗口页以 YAML 格式填写

配置示例

在服务对象中配置cors插件，此处设置缓存结果的最大时间max_age=600。

```
curl http://192.168.0.95:27152/apisix/admin/routes/1 -H 'X-API-KEY: 2571e288
e8f4cd273cab342440068469' -X PUT -d '> {
>    "name": "test12",
>    "uri": "/hello",
>    "plugins": {
>       "cors": {
>          "max_age": 600
>       }
>    },
>    "upstream": {
>       "type": "roundrobin",
>       "nodes": {
>          "127.0.0.1:39087": 1
>       }
>    }
> }'
```

启用/停用

在配置页面设置生效开关

验证插件

请求接口，发现已返回cors相关的 header，代表插件已经生效。

```
curl -v http://192.168.0.95:27151/hello -X GET
...
< Server: APISIX/2.13.3
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: *
< Access-Control-Max-Age: 600
< Access-Control-Expose-Headers: *
< Access-Control-Allow-Headers: *
...
```

信息重写类

重写Proxy-rewrite插件

描述

proxy-rewrite 是一个上游服务信息重写插件。

作用范围

该插件即可用于全局插件，也可用于路由级插件。全局插件配置的优先级高于路由级插件配置，当同时在某一路由上配置了proxy-rewrite的全局插件和路由级插件时，以全局插件配置中设置的属性值为准。

属性

名称	类型	必选项	有效值	描述
uri	string	否	--	转发到上游的新uri地址。
regex_uri	array[string]	否	["GET", "POST", "PUT", "HEAD", "DELETE", "OPTIONS","MKCOL", "COPY", "MOVE", "PROPFIND", "PROPFIND","LOCK", "UNLOCK", "PATCH", "TRACE"]	转发到上游的新uri地址，使用正则表达式匹配来自客户端的uri，当匹配成功后使用模板替换转发到上游的 uri，未匹配成功时将客户端请求的 uri 转发至上游。当 uri 和 regex_uri 同时存在时，uri 优先被使用。
host	string	否	--	转发到上游的新host地址。

如何使用

在配置窗口页以 YAML 格式填写

配置示例

下面是一个示例，在指定的 API 上开启了proxy-rewrite 插件。

```
curl http://192.168.0.95:27152/apisix/admin/routes/1 -H 'X-API-KEY: 2571e288
e8f4cd273cab342440068469' -X PUT -d '
{
   "methods": ["GET"],
   "uri": "/test/index.html",
   "plugins": {
      "proxy-rewrite": {
         "uri": "/test/home.html",
         "scheme": "http",
         "host": "ctyun.com",
         "headers": {
            "X-Api-Version": "v1",
            "X-Api-Engine": "apisix",
            "X-Api-useless": ""
         }
```

```
        }
    },
    "upstream": {
        "type": "roundrobin",
        "nodes": {
            "127.0.0.1:80": 1
        }
    }
}'
```

启用/停用

在配置页面设置生效开关

验证插件

发送请求，查看上游服务的 access.log，如果输出信息与配置一致：

```
curl -X GET http://192.168.0.95:27152/test/index.html

127.0.0.1 - [2/Aug/2023:10:52:20 +0800] ctyun.com GET
/test/home.html HTTP/1.1 200 38 - curl/7.29.0 - 0.000 199 107
```

这就表示proxy-rewrite 插件已经生效了。

重写 proxy-cookie 插件

描述

proxy-cookie插件是对响应头部set-cookie字段中的path属性或者domain属性进行重写，校验的对象：

- 请求目标服务的返回头部Set-Cookie中Path属性。
- 请求目标服务的返回头部Set-Cookie中Domain属性。

注意

set-cookie头部不区分大小写

作用范围

该插件只能用于路由级插件，因为每条路由响应头部set-cookie字段中的path属性或者domain属性不同，要视具体情况而定。

属性

名称	类型	必选项	默认值	有效值	描述
domain	Object	可选	--	--	domain域的重写配置
domain	use	boolen	可选	false	[false, true]
domain	regex	string	--	--	--
domain	replacement	string	--	--	--
path	Object	可选	--	--	path域的重写配置

名称	类型	必选项	默认值	有效值	描述
path	use	boolen	可选	false	[false, true]
path	regex	string	--	--	--
path	replacement	string	--	--	--

如何启用

在配置窗口页以 YAML 格式填写。

配置示例

1. 样例

配置proxy-cookie插件内容，关闭domain，开启path。

```
curl http://192.168.0.95:27152/apisix/admin/routes/1 -H 'X-API-KEY: 2571e288
e8f4cd273cab342440068469' -X PUT -d '
{
   "name": "test",
   "uri": "/hello",
    "plugins": {
       "proxy-cookie": {
 "domain": {
    "use": false
 },
 "path": {
    "regex": "/c/app",
  "use": true,
  "replacement": "/"
 },
"disable": true
 }
   },
    "upstream": {
      "type": "roundrobin",
      "nodes": {
          "127.0.0.1:39087": 1
      }
   }
}'
```

2. 示例场景

以Path属性为例，Domain属性同理。

- 测试场景1：当set-cookie中不存在Path属性时

用例1-1：属性关闭

即 proxy_cookie_path = {use=false}， 表示不对Path属性进行重写

用例1-2：属性开启

proxy_cookie_path = {use=true， regex="/app/a/b", replacement="/a/b"}， 表示不对Path属性进行重写

proxy_cookie_path = {use=true， regex="^/app/a/b", replacement="/a/b"}， 表示不对Path属性进行重写

- 测试场景2：当set-cookie中存在Path属性时

如set-cookie: Path=/c/app/a/b

用例2-1 属性关闭

即 proxy_cookie_path = {use=false}， 表示不对Path属性进行重写

用例2-2 属性开启

proxy_cookie_path = {use=true， regex="/app/a/b", replacement="/a/b"}，表示对Path属性进行重写，重写后，set-cookie: Path=/c/a/b

proxy_cookie_path = {use=true， regex="*/a/b", replacement="/a/b"}， 表示对Path属性进行重写，重写后，set-cookie: Path=/a/b

proxy_cookie_path = {use=true， regex="c/a/b", replacement="/a/b"}， 表示对Path属性进行重写，重写后（未匹配时，原样输出），set-cookie: Path=/c/app/a/b

启用/启停

在配置页面设置生效开关。

验证插件

假定请求响应中的Set-cookie字段信息如下

配置示例如样例中设置，对应的yaml如下

```
# 对应proxy_cookie_domain
domain:
# 是否更改domain属性，默认为false，当设置为true时匹配规则和替换值不可为空
 use: false
 # 匹配规则
 #regex: "ctyun.com"
 # 替换值
 #replacement: ""
# 对应proxy_cookie_path
path:
 # 是否更改path属性，默认为false，当设置为true时匹配规则和替换值不可为空
 use: true
 # 匹配规则
 regex: "/c/app"
 # 替换值
 replacement: "/"
```

请求接口，发现在header的Set-cookie字段中带上了Path与Domain的值，代表插件已经生效。

头部重写header-rewrite 插件

描述

header-rewrite插件可以对请求/响应头部的header参数进行新增、修改和删除操作。

作用范围

该插件只能用于路由级插件，因为每条路由请求/响应头部的header参数不同，要视具体情况而定。

属性

名称	类型	必选项	默认值	有效值	描述
request	Object	可选	--	--	修改请求header,支持新增、修改、删除操作。
request	add	Array	--	--	--
request	update	Array	--	--	--
request	delete	Array	--	--	--
response	Object	可选	--	--	修改响应header,支持新增、修改、删除操作。
response	add	Array	--	--	--
response	update	Array	--	--	--
response	delete	Array	--	--	--

支持的类型：

- request：请求头部类型
- response：响应头部类型

允许的操作：

- 新增：增加一个头部
- 修改：存在则更新；不存在则增加
- 删除：去掉一个指定key，不需要指定value

如何使用

在配置窗口页以 YAML 格式填写

配置示例

模板参考：

```
# 修改请求header,支持新增、修改、删除操作，操作类型不为空时，其目标header key也不能为空
request:
 # 新增操作,当存在目标header key时，末尾追加值;否则，新增一个header
 add:
   num: 123
```

```
  name:"cgw"
 # 更新操作,当存在目标header key时，更新header值;否则，新增一个header
 update:
  num:234
  name:"CGW"
 # 删除操作,当存在目标header key时，删除目标header;否则忽略
 delete:
  - id
  - name
# 修改响应header,支持新增、修改、删除操作，操作类型不为空时，其目标header key也不能为空
response:
 # 新增操作,当存在目标header key时，末尾追加值;否则，新增一个header
 add:
  num:123
  name:"cgw"
 # 更新操作,当存在目标header key时，更新header值;否则，新增一个header
 update:
  num:234
  name:"CGW"
 # 删除操作,当存在目标header key时，删除目标header;否则忽略
 delete:
  - id
  - name
```

启用/停用

在配置页面设置生效开关。

验证插件

请求接口，若发现已返回header发生变更，代表插件已经生效。

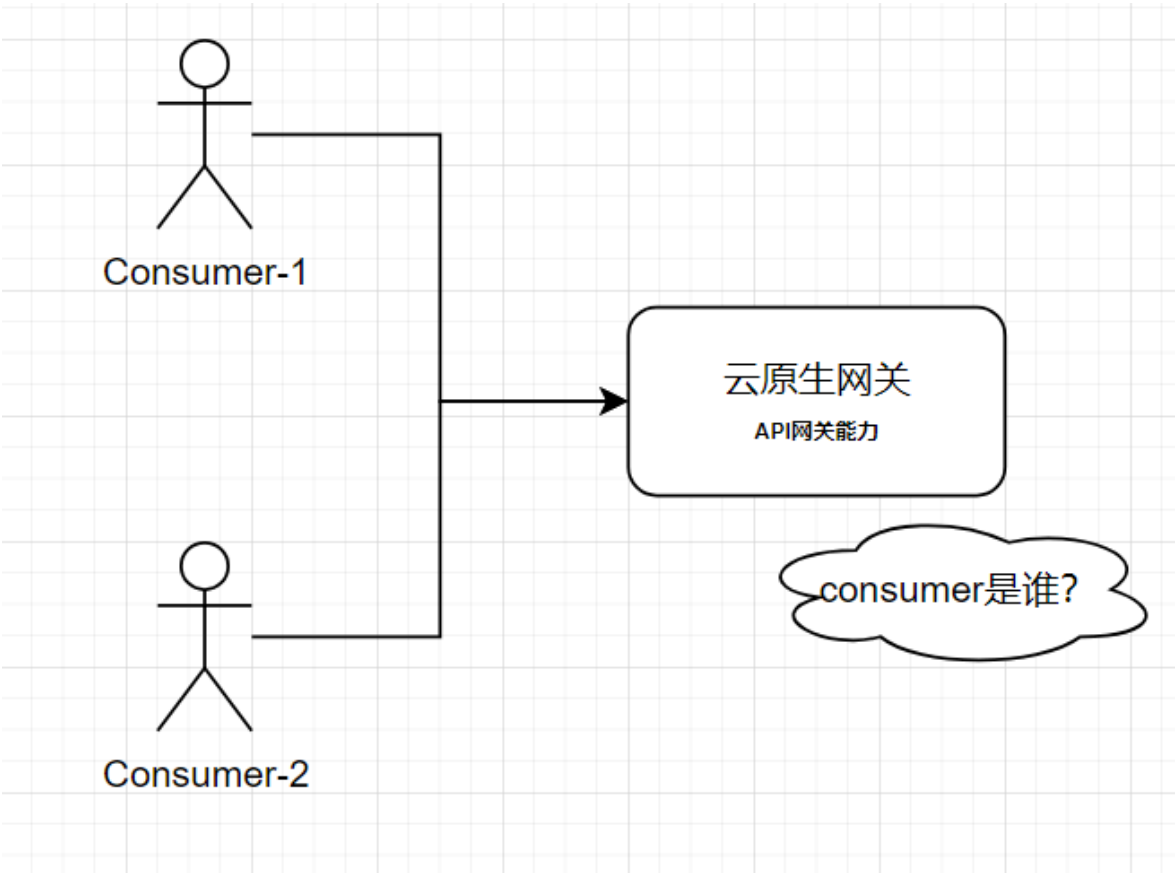
API托管

概述

API网关是API 托管服务，提供 API 的完整生命周期管理，包括创建、维护、发布、运行、下线、运维监测、安全管控等阶段。通过API网关服务，可以将您的数据、业务逻辑或功能安全可靠的开放出来，以快速建设以API为核心的系统架构，为业务系统、合作伙伴提供连接。

消费者

消费者通常是网关的调用方，比如可以是客户端程序等，所以通常也可以称为应用；当消费者请求路由到网关时，网关会对消费者进行识别。网关要判断这个调用者有没有访问当前路由的权限，如果没有，网关就会拒绝当前请求，否则就会通过路由请求并对请求进行进一步的处理。识别过程我们叫做鉴权，那么鉴权之前，为确保具有路由请求的权限，以允许对应消费者访问路由，会给目标路由进行授权，也即是消费者授权或者叫做应用授权。



摘要签名认证

当前网关支持的认证鉴权方式为摘要签名认证方式。API网关提供前端签名及验签能力，前端签名及验签主要两点用途：

1. 验证客户端请求的合法性，确认请求中携带授权后的AK生成的签名。
2. 防止请求数据在网络传输过程中被篡改。

API的拥有者可以在网关控制台的应用管理页面生成APP，每个APP会携带一对签名密钥（APP Key和APP Secret），API拥有者将API授权给指定的APP（APP可以是API拥有者颁发或者API调用者所有）后，API调用者就可以用APP的签名密钥来调用相关的API了。

AppKey和AppSecret

客户端调用API 时，需要使用已授权签名密钥对请求内容的关键数据进行加密签名计算，并且将APP Key和加密后生成的字符串放在请求的 Header 传输给API网关，API网关会读取请求中的APP Key的头信息，并且根据APP Key的值查询到对应的APP Secret的值，使用APP Secret对收到的请求中的关键数据进行签名计算，并且使用自己的生成的签名和客户端传上来的签名进行比对，来验证签名的正确性。只有签名验证通过的请求才会发送给后端服务，否则API网关会认为该请求为非法请求，直接返回错误应答。

AppCode简单认证

扩展功能，暂未上线，作为保留字段。

AppCode简单认证就如同字面含义，旨在为用户提供一种快捷简单的认证方式去进行授权签名，无需在客户端实现复杂的签名算法，其作用等同于AppKey和AppSecret配对使用的效果。

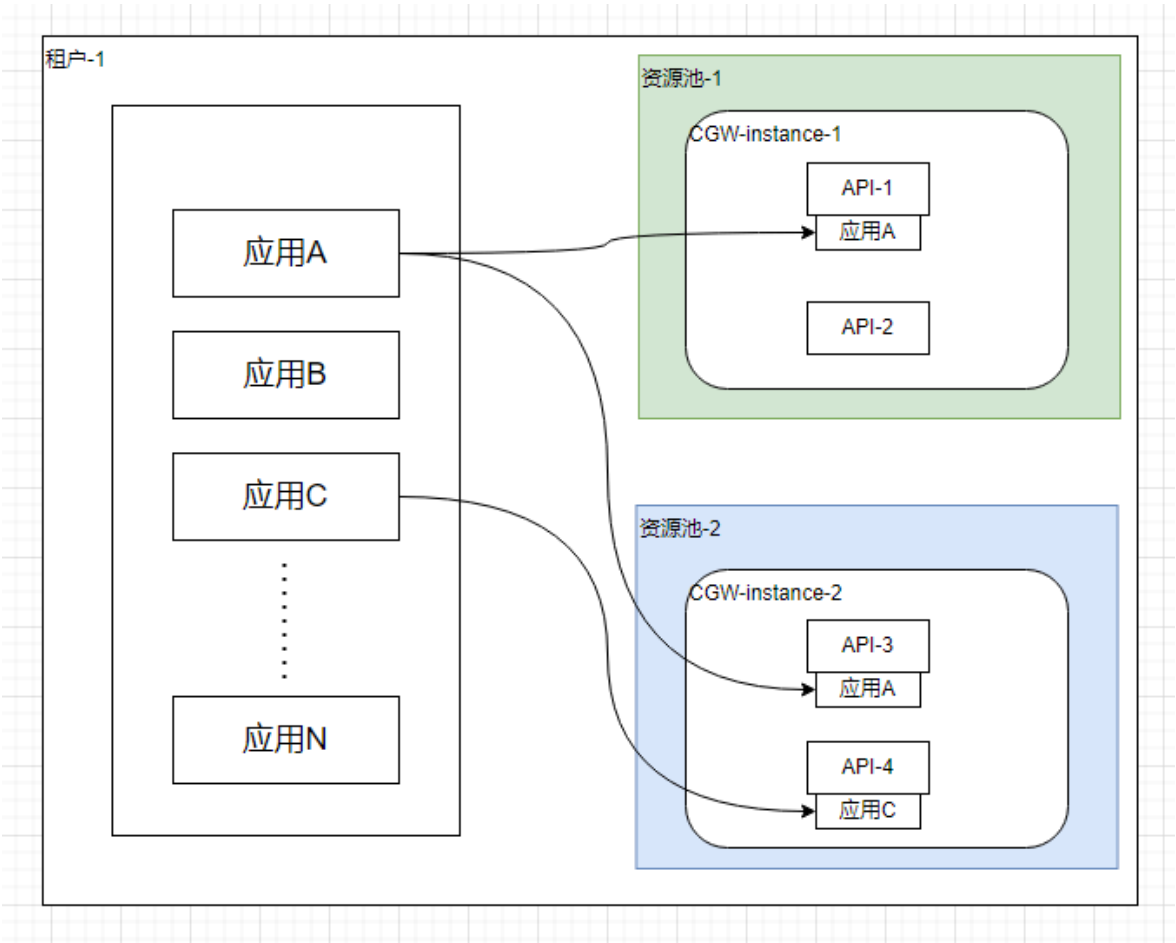
后续将会允许用户通过header或者query参数调用AppCode，进行简单认证。

注意

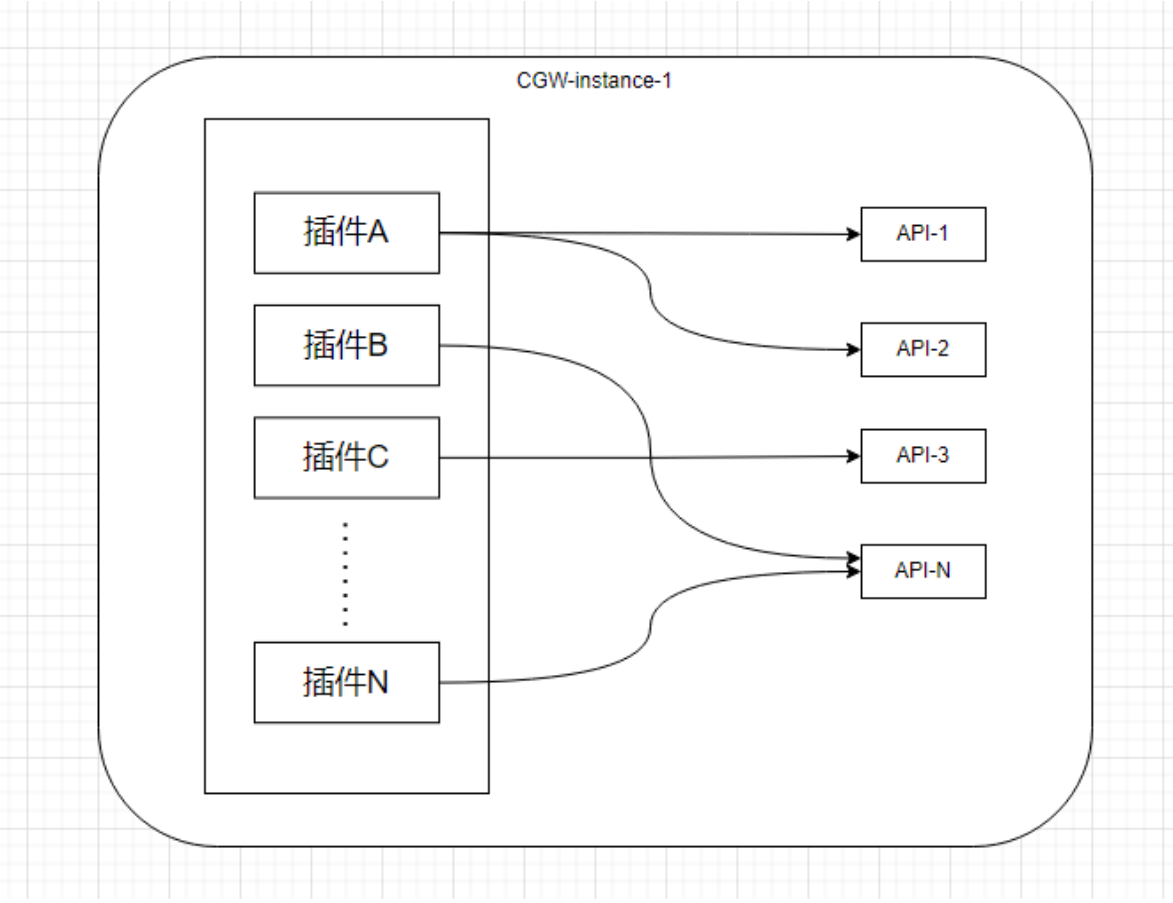
简单认证方式调用非常省事，免去了复杂的签名过程，但是把AppCode作为明文暴露网络中传输，会带来一些安全隐患，存在丢失AppCode的风险。

API对消费者鉴权和自定义鉴权插件之间的关系?

消费者/应用配置可以在不同资源池中可以共享，在租户/用户级别上是隔离的，也即是说用户User-1在区域Region-1创建了的应用App-A，当用户User-1切换区域到Region-B后，在应用列表可以看到App-A，并且可以对App-A进行修改、解除API的授权等操作，该操作在用户User-1下所有区域及区域下已授权的所有API中进行同步。



自定义鉴权插件，对网关来讲属于实例级别的应用范围。用户在当前实例Instance-1下添加的鉴权配置（插件配置管理），只适用于实例Instance-1中所有的API。



总体来讲，消费者适用于租户级别下的API授权管理，不受区域、资源池、实例的限制；自定义鉴权插件，通常是作用与实例级别下的API授权管理，范围更小。

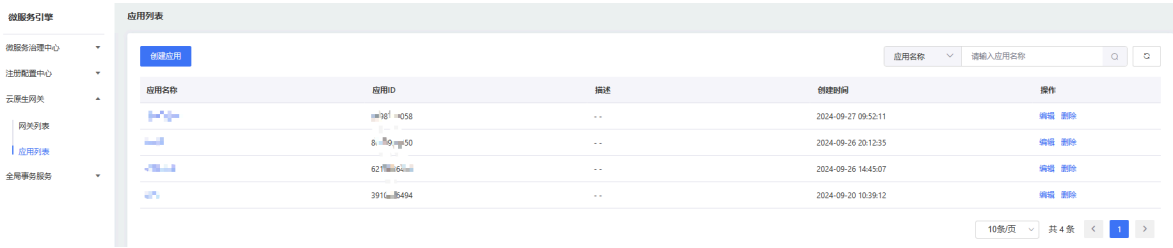
消费者管理

消费者（应用）管理

应用列表

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 应用列表 ；



创建应用

操作步骤

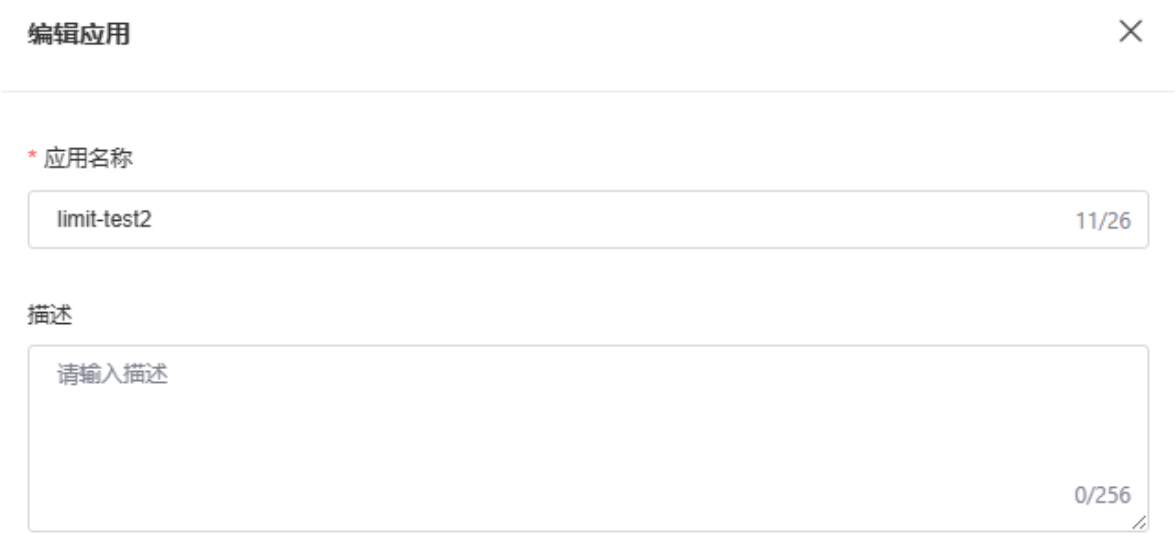
1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 应用列表 ；
4. 点击创建应用按钮，填写应用名称、AppKey、AppSecret、AppCode、描述等信息；
   AppKey、AppSecret、AppCode为空时，将自动生成；

编辑应用

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 应用列表 ；

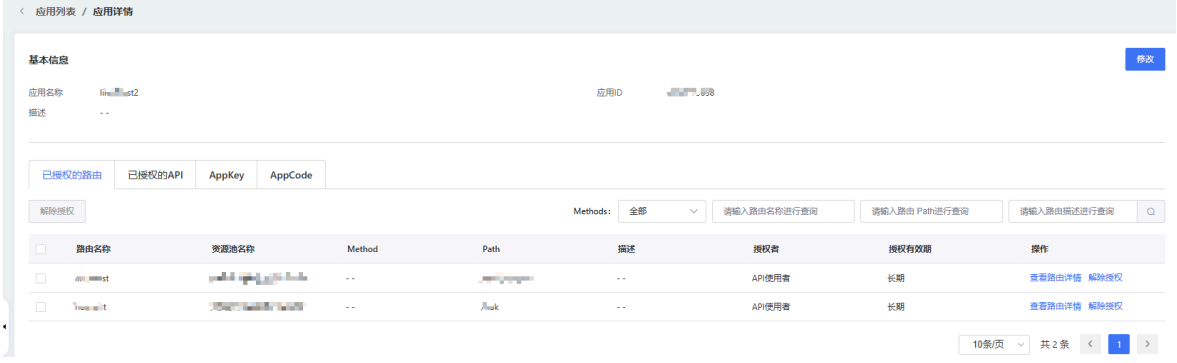
4. 选择某个应用，点击应用的编辑按钮 ，可修改应用的名称和描述信息；确定后完成编辑；



查询应用信息

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 应用列表 ；
4. 点击应用名称，进入应用详情页，可以查看当前应用有哪些授权的路由、API，查看应用的AppKey、AppSecret，支持重置AppSecret、AppCode；



添加应用授权

应用授权需要在路由或者API的视角下完成，且只有路由、API的认证方式为APP时才允许进行授权操作。

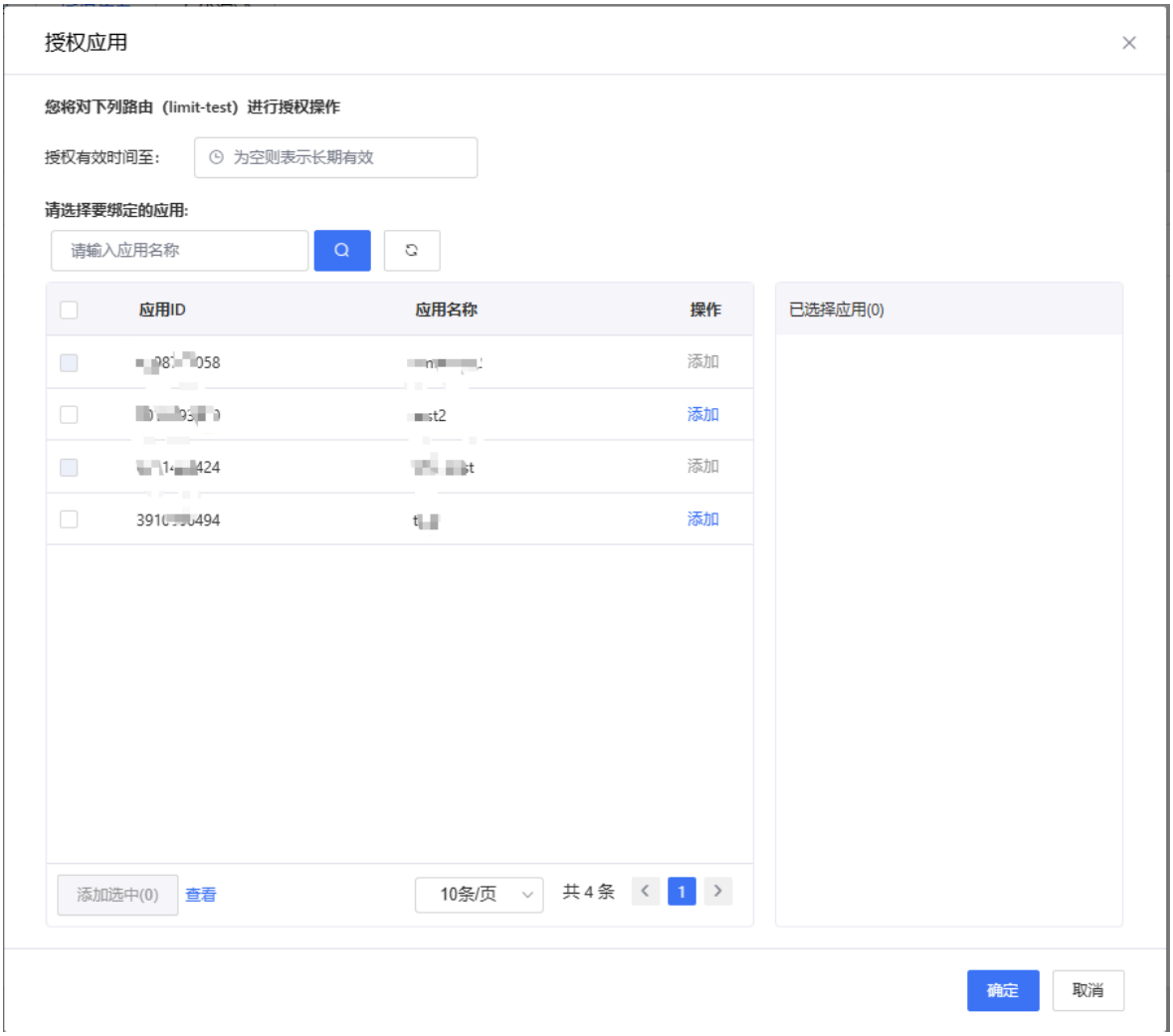
注意：Mock路由中对应用设置过期时间的使用限制参见云原生网关常见问题说明

路由视角下的授权应用

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；

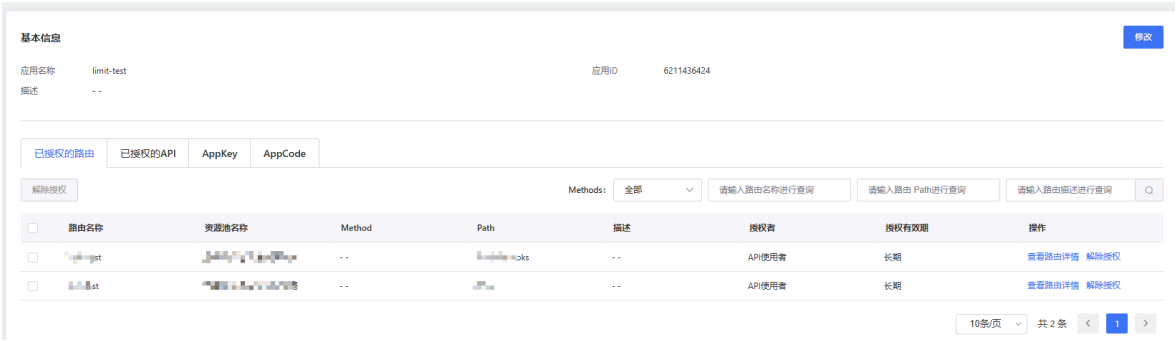
3. 单击左侧导航栏云原生网关 > 实例列表 ；
4. 选择实例进入 实例详情 >路由配置;
5. 选择 需要授权的路由 >点击路由名称>路由详情>授权信息页;
6. 点击“添加授权”，在可选的应用列表中添加一个或多个应用配置，点击确定开始对路由的授权;



7. 授权完成后，在路由的视角下可查看此路由已绑定应用授权信息，也可以在应用的视角下，可查看此应用已绑定路由信息。

   路由视角下的授权信息

   应用视角下的授权的路由信息



API视角下的授权应用

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 实例列表 ；
4. 选择实例进入 实例详情 >API托管->API列表;
5. 选择 需要授权的API >点击API名称>API详情页>授权信息页;
6. 点击“添加授权”，在可选的应用列表中添加一个或多个应用配置，点击确定开始对API的授权； 
7. 授权完成后，在API的视角下可查看此API已绑定应用授权信息，也可以在应用的视角下，可查看此应用已绑定API信息

   API视角下的授权信息

   应用视角下的API授权信息 

解除应用授权

用户可以根据业务需求，对路由进行授权解除，实现路由级别的安全管控。目前网关提供了两种视角下的解除操作，在路由/API视角下和在App视角下。同时也支持单个解除授权和批量解除授权。

在路由的视角下

操作步骤

1. 进入微服务引擎MSE控制台；

2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 选择实例->实例详情>路由配置,， 选择目标路由；
5. 路由详情>授权信息，选中需要解除授权的应用 > 批量接触授权 ; 

在API的视角下

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 网关列表 ；
4. 选择实例->实例详情 > API托管 > API列表，选择目标API；
5. API详情>授权信息，选中需要解除授权的应用 >批量接触授权 ; 

在应用的视角下

解除对路由的授权

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 应用列表 ；
4. 选择目标应用>应用详情>已授权路由；
5. 勾选一个或多个进行批量移除授权，或者通过列表中移除授权按钮进行单个关系的移除；

   批量解除

   单个解除

解除对API的授权

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 应用列表 ；
4. 选择目标应用>应用详情>已授API列表；
5. 勾选一个或多个进行批量移除授权，或者通过列表中移除授权按钮进行单个关系的移除；

   批量解除 

   单个解除 

删除应用

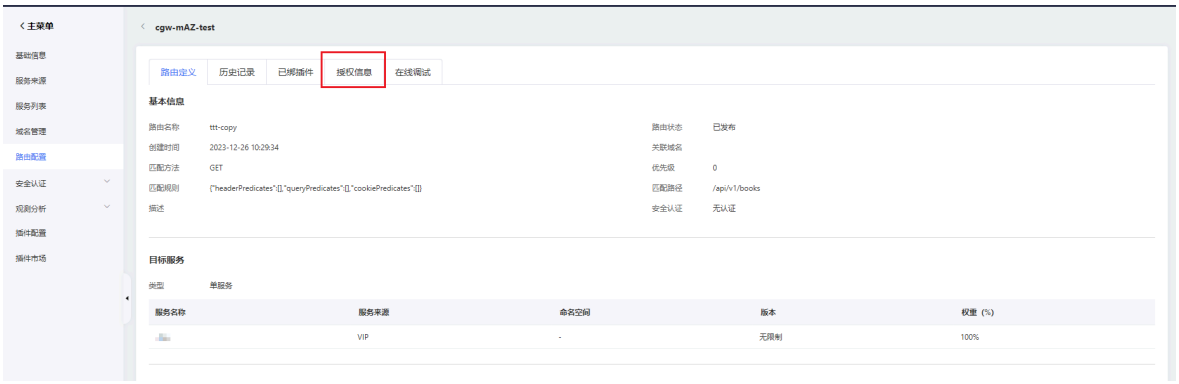
操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 > 应用列表 ；
4. 选择目标应用，点击删除 ；

该操作会移除此应用下已授权的路由和API关系，请谨慎操作。 

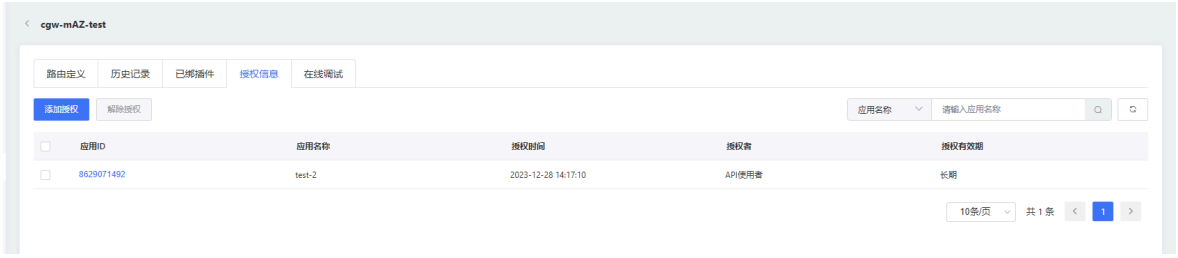
如何授权消费者调用API

在API信息页面，可以看到授权信息栏目



进入授权信息页，点击添加授权，选择应用添加，开始对API的授权。

授权完成后，通过携带签名即可通过网关的认证鉴权使用API。



签名的计算参见下节内容，假定得到签名signature后，请求示例如下：

```
$ curl -i "http://127.0.0.1:9080/index.html?name=james&age=36" \
-H "X-HMAC-SIGNATURE: ${signature}" \
-H "X-HMAC-ALGORITHM: hmac-sha256" \
-H "X-HMAC-ACCESS-KEY: ${appKey}"
```

消费者调用API签名计算

签名过程

客户端

1. 从原始请求中提取关键请求信息，构造签名字符串；
2. 利用加密算法和AppSecret对签名字符串进行加密处理，得到签名；
3. 将签名相关的头部信息加入到原始请求中，发送请求；

服务端

1. 从接收到的请求中提取关键请求信息，得到一个用来签名的签名串；
2. 从接收到的请求中读取APPKey，通过APPKey查询到对应的APPSecret；
3. 使用加密算法和APPSecret对关键请求信息签名串进行加密处理，得到签名；

4. 从接收到的请求中读取客户端签名，对比服务器端签名和客户端签名的一致性。

签名生成公式

签名的计算公式为signature = HMAC-SHAx-HEX(secret_key, signing_string)，从公式可以看出，想要获得签名需要得到 secret_key 和 signing_string 两个参数。其中 secret_key 为对应应用所配置的，signing_string 的计算公式为 signing_string = HTTP Method + \n + HTTP URI + \n + canonical_query_string + \n + access_key + \n + Date + \n + signed_headers_string。如果 signing_string 中的某一项不存在，也需要使用一个空字符串代替。

字段解释

1. HTTP Method：指 HTTP 协议中定义的 GET、PUT、POST 等请求方法，必须使用全大写的形式。
2. HTTP URI：要求必须以“/”开头，不以“/”开头的需要补充上，空路径为“/”。
3. Date：请求头中的 Date （ GMT 格式 ）。
4. canonical_query_string：是对于 URL 中的 query（ query 即 URL 中 ? 后面的 key1=valve1&key2=valve2 字符串）进行编码后的结果。
5. signed_headers_string：是从请求头中获取客户端指定的字段，并按顺序拼接字符串的结果。

其中canonical_query_string 编码步骤如下：

提取URL 中的 query 项，即 URL 中 ? 后面的 key1=valve1&key2=valve2 字符串。

将query 根据&分隔符拆开成若干项，每一项是 key=value 或者只有 key 的形式。

当该项只有key 时，转换公式为 url_encode(key) + "=" 的形式。

当该项是key=value 的形式时，转换公式为 url_encode(key) + "=" + url_encode(value) 的形式。这里 value 可以是空字符串。

将每一项转换后，以key 按照字典顺序（ ASCII 码由小到大）排序，并使用 & 符号连接起来，生成相应的 canonical_query_string 。

签名字符串拼接示例

以下面请求为例：

$ curl -i http://127.0.0.1:9080/index.html?name=james&age=36

根据签名生成公式生成的signing_string 为：

"GET

/index.html

age=36&name=james

"

注意

最后一个请求头也需要+ \n。

生成签名

使用Python 来生成签名 SIGNATURE：

```
import base64
import hashlib
import hmac

secret = bytes('my-secret-key', 'utf-8')
message = bytes("""GET
/index.html
age=36&name=james

""", 'utf-8')

hash = hmac.new(secret, message, hashlib.sha256)
# to lowercase base64
print(base64.b64encode(hash.digest()))
```

不同语言的签名生成样例

以下提供一个更为简单的脚本帮助用户快速得到请求时使用的签名header

Python3

执行样例：python3 app_token_gen.py uri method ak sk query

```
# app_token_gen.py
import sys
import base64
import hashlib
import hmac

inputs_num = len(sys.argv)
if inputs_num < 5:
 print("请依次传入uri、method、ak、sk、query(可选)的值，如
python3 app_token_gen.py uri method ak sk query")
 sys.exit(1)
uri = sys.argv[1]
method = sys.argv[2]
ak = sys.argv[3]
sk = sys.argv[4]
query_param = ''
if inputs_num >=6:
 query_param = sys.argv[5]
if ak is None or sk is None:
 print("请依次传入uri、method、ak、sk、query(可选)的值，如
python3 app_token_gen.py uri method ak sk query")
 sys.exit(1)
```

```
secret = bytes(sk, 'utf-8')
signature_message_template = "'%s
%s
%s
%s

'"

signature_message = signature_message_template % (method, uri, query_param, ak)
message = bytes(signature_message, 'utf-8')
hash = hmac.new(secret, message, hashlib.sha256)

# to lowercase base64
signature_code = base64.b64encode(hash.digest())
signature_code_str = str(signature_code)

headers_template ="'-H "X-HMAC-ALGORITHM: hmac-sha256" -H "X-HMAC-ACCESS-KEY: %s" -H "X-HMAC-
SIGNATURE: %s"'"
signature_code_len = len(signature_code_str)
start = 2
end = signature_code_len-1
signature = signature_code_str[start:end]

hmac_headers = headers_template % (ak, signature)

# print some info
print("待签名的字符串信息：" + signature_message)
print("原始encode后的编码值：" + signature_code_str)
print("使用的hmac header：" + hmac_headers)
```

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.xml.bind.DatatypeConverter;

class Main {
 public static void main(String[] args) {
  try {
    String secret = "the shared secret key here";
    String message = "this is signature string";

    Mac hasher = Mac.getInstance("HmacSHA256");
    hasher.init(new SecretKeySpec(secret.getBytes(), "HmacSHA256"));
```

```
    byte[] hash = hasher.doFinal(message.getBytes());

    // to lowercase hexits
    DatatypeConverter.printHexBinary(hash);

    // to base64
    DatatypeConverter.printBase64Binary(hash);
  }
  catch (NoSuchAlgorithmException e) {}
  catch (InvalidKeyException e) {}
 }
}
```

GO

```
package main

import (
  "crypto/hmac"
  "crypto/sha256"
  "encoding/base64"
  "encoding/hex"
)

func main() {
  secret := []byte("the shared secret key here")
  message := []byte("this is signature string")

  hash := hmac.New(sha256.New, secret)
  hash.Write(message)

  // to lowercase hexits
  hex.EncodeToString(hash.Sum(nil))

  // to base64
  base64.StdEncoding.EncodeToString(hash.Sum(nil))
}
```

Ruby

```
require 'base64'
require 'openssl'

secret = 'the shared secret key here'
message = 'this is signature string'

# to lowercase hexits
OpenSSL::HMAC.hexdigest('sha256', secret, message)
```

```
# to base64
Base64.encode64(OpenSSL::HMAC.digest('sha256', secret, message))
```

NodeJS

```
var crypto = require('crypto');

var secret = 'the shared secret key here';
var message = 'this is signature string';

var hash = crypto.createHmac('sha256', secret).update(message);

// to lowercase hexits
hash.digest('hex');

// to base64
hash.digest('base64');
```

PHP

```
<?php

$secret = 'the shared secret key here';
$message = 'this is signature string';

// to lowercase hexits
hash_hmac('sha256', $message, $secret);

// to base64
base64_encode(hash_hmac('sha256', $message, $secret, true));
```

Lua

```
local hmac = require("resty.hmac")
local secret = 'the shared secret key here'
local message = 'this is signature string'
local digest = hmac:new(secret, hmac.ALGOS.SHA256):final(message)

--to lowercase hexits
ngx.say(digest)

--to base64
ngx.say(ngx.encode_base64(digest))
```

Shell

```
SECRET="the shared secret key here"
MESSAGE="this is signature string"
```

```
# to lowercase hexits
echo -e $MESSAGE | openssl dgst -sha256 -hmac $SECRET

# to base64
echo -e $MESSAGE | openssl dgst -sha256 -hmac $SECRET -binary | base64
```

后端服务

后端服务管理

概述

后端服务是云原生网关处理外部请求的实际业务处理模块。支持多环境管理，不同环境可以绑定不同的服务，甚至是来自不同来源的服务，确保在多环境下灵活应对各种业务需求。

操作步骤

1. 登录**微服务引擎MSE**云原生网关管理控制台，选择**资源池**。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > 后端服务**，可创建新的后端服务，也可以点击要管理的后端服务。

创建后端服务

后端服务的流程为：创建服务来源->创建服务->创建后端服务->后端服务绑定服务

分组管理

分组管理

概述

API分组管理用于将相关API组织在一起，方便统一管理和配置，实现更高效的运维。

操作步骤

1. 登录微**服务引擎MSE**云原生网关管理控制台，选择**资源池**。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > 分组管理**。

创建分组

在分组列表中，点击**创建API分组**即可打开创建API分组的弹窗。在弹窗中，需要填写**基础路径**（BasePath）。BasePath表示该分组下所有API的访问路径前缀，访问分组下的API时，需要添加该前缀。如果不需要设置特定前缀，可以填写“/”。基础路径在创建后，可点击**编辑**进行修改。分组创建后，即可在分组下创建API。

删除分组

点击**删除**，即可删除API分组，如果分组下存在API，会禁止删除分组。

绑定域名

概述

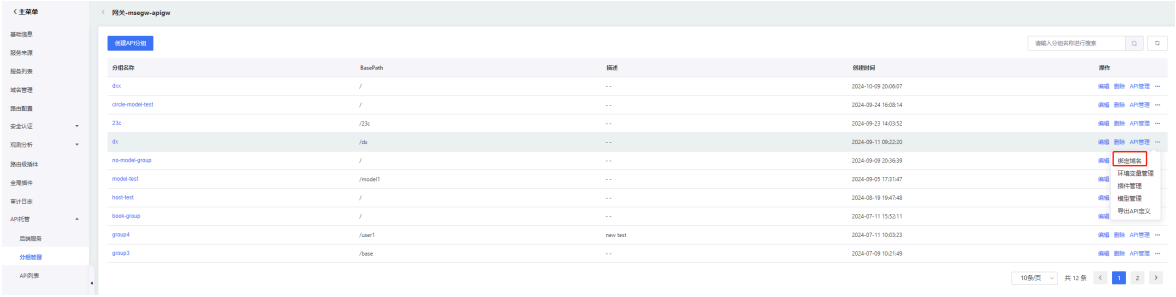
API分组支持绑定域名。绑定域名后，可通过该域名访问分组下的API，但前提是该域名需指向该云原生网关实例。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > 分组管理**。

绑定域名

点击对应分组右侧省略号中的**绑定域名**，即可进入绑定域名弹窗。也可以在分组详情页中，点击域名模块的修改按钮进入。



云原生网关的API支持多环境隔离，提供两种路由方式：请求头路由环境和域名路由环境。两种方式可独立配置，也可同时使用。

- **请求头路由环境**：启用后，通过在请求头中传入 X-Ca-Stage=PROD/TEST/DEV，即可将请求路由到相应的环境。
- **域名路由环境**：绑定域名到特定环境后，可通过该域名访问API，网关会自动将请求路由至对应的环境。

  绑定域名也存在一定的约束：

  - 基于请求头部区分环境时，每个环境可以不关联域名或者关联相同的域名；
  - 不基于请求头部区分环境时，每个环境必须关联至少一个域名且域名不能有重复；
  - 同一个分组最多支持5个域名
  - 基于请求头部区分环境时，使用X-Ca-Stage请求头确定环境。其中环境名称分别为：生产-PROD；测试-TEST；开发-DEV

绑定插件

概述

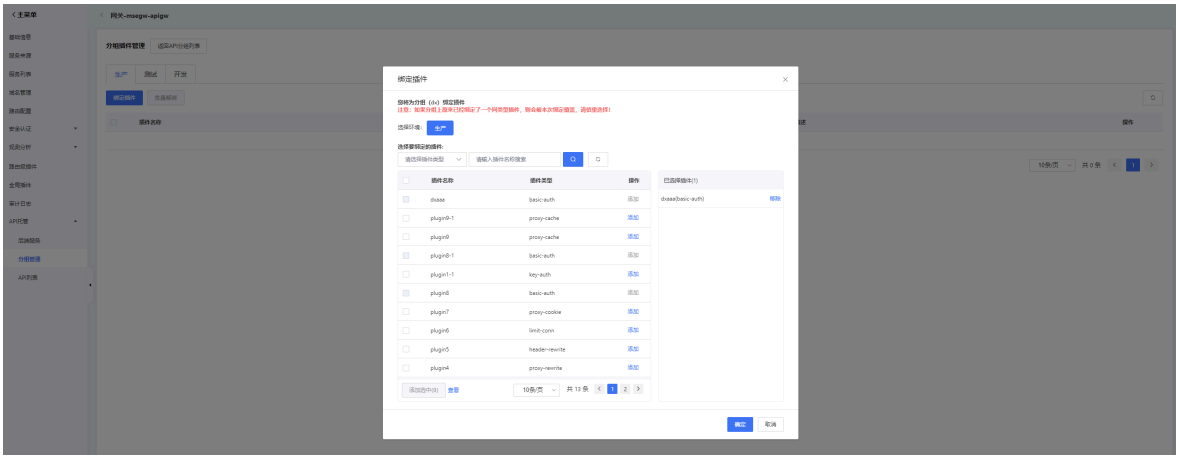
插件是云原生网关的扩展机制，用于增强网关功能，支持动态加载，能够在请求的不同阶段（如请求处理、转发、响应）执行自定义逻辑。API分组可绑定插件，绑定后插件将应用于该分组下的所有API。此外，插件支持环境隔离，仅在所绑定的特定环境中生效。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > 分组管理**。

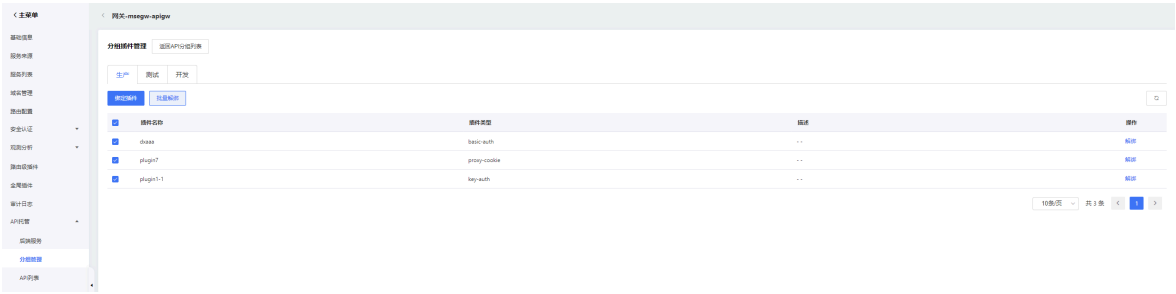
绑定插件

分组只能绑定路由级插件，需先在路由级插件页面创建插件。然后在分组管理页面，点击分组右侧省略号中的**插件管理**进入插件绑定页面。选择对应环境后，点击**绑定插件**按钮，进入绑定插件弹窗，将所需插件添加到右侧区块，点击确定后即可完成批量绑定。每个分组只能绑定一个同类型插件，若已有同类型插件，本次绑定会覆盖。同时，同类型插件中，API绑定的优先级高于分组绑定的插件。



解绑插件

插件支持解绑，在分组插件管理页面，可单个、也可批量进行插件解绑。



环境变量

概述

API分组环境变量用于在不同的环境（生产，测试，开发）中引用同一个环境变量中不同的值，从而可以灵活引用变量。通常用于动态调整配置项，比如数据库连接、外部服务的 URL、认证密钥等。这减少了硬编码，并使得应用程序能够根据部署环境自动适应配置变化。

创建环境变量

1. 进入API托管->分组管理菜单页
2. 点击环境变量管理，进入环境变量管理页，在对应的环境tab页下点击新增环境变量按钮，填写变量名和变量值进行创建。

注意

每个环境下允许最多创建50个变量；创建同名环境变量时会覆盖已有环境变量值。

删除环境变量

用户可以对环境下的环境变量进行删除操作。注意：删除前请先确保该环境变量没有在API定义中使用。

使用环境变量

创建了环境变量后，可以在API定义中在请求Path、入参默认值、后端服务地址部分加入变量，以**#**变量名**#**表示，如请求path可以填写“**/#**version**#**”。

当该API定义发布到该环境时，API定义中的变量就会取值对应的变量值，即发布过程中变量标识会被相应环境的变量值替换。例如在生产环境中定义了变量version的值为v1，则上述API的请求path会被解析为/v1。

注意

1. 变量名严格区分大小写。
2. 如果在API定义中设置了变量，那么一定要在要发布的环境上配置变量名**&**变量值，否则变量无赋值，API将无法正常调用。

3. API调试暂不支持进行环境变量解析。

网关响应

概述

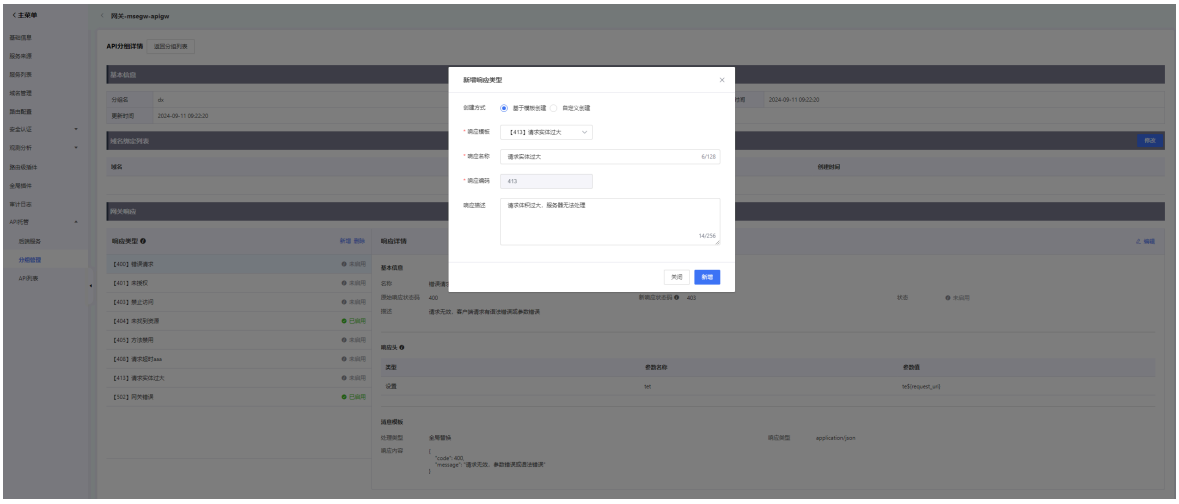
网关响应是指云原生网关在接收到后端服务的响应后，依据用户配置的响应规则对响应内容进行处理，并将处理后的结果返回给客户端。用户可以在API分组中配置网关响应，启用后，该响应规则会对分组内的所有API生效。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > 分组管理**，进入要查看的API分组详情，即可看到网关响应模块。

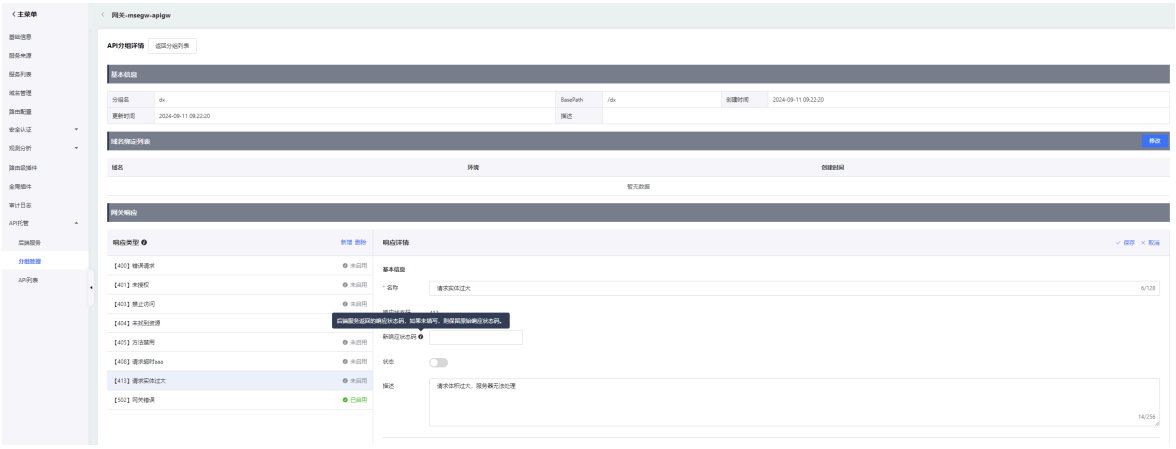
新增网关响应

点击网关响应模块内的**新增**按钮，即可进入网关响应新增弹窗。支持基于模版创建和自定义创建两种方式。响应编码支持200-598，每个分组下，同一个响应编码只能添加一个响应规则。



响应状态码

默认情况下，新建的响应规则会保留后端服务返回的原始响应状态码。如果需要修改响应状态码，可以通过编辑响应规则并填入新的状态码。例如，在413的响应规则中填入500，则客户端接收到的所有413响应都会被替换为500状态码。



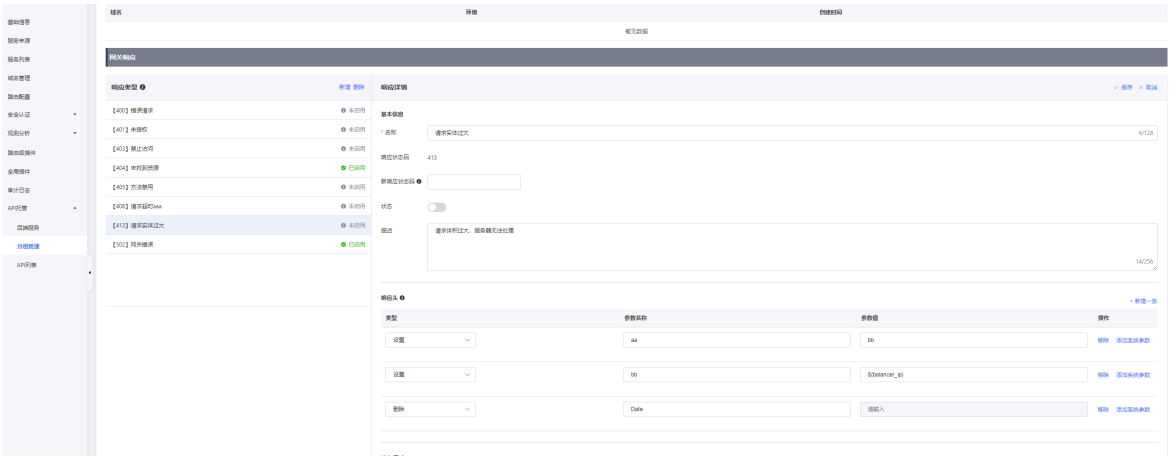
响应头

默认情况下，新建的响应规则不会处理响应头。如果需要修改，可以编辑响应规则并添加对响应头的处理逻辑。响应头的处理支持两种操作方式：设置和删除。

· **设置** ：若响应头不存在则新增，若已存在则覆盖。

· **删除** ：用于移除指定的响应头。

参数值可以是常量、系统参数，或两者的组合。系统参数可以从预定义列表中选择，也可以手动输入，使用**${}** 进行包裹。当匹配到该系统参数时，会自动替换为对应的系统变量值。

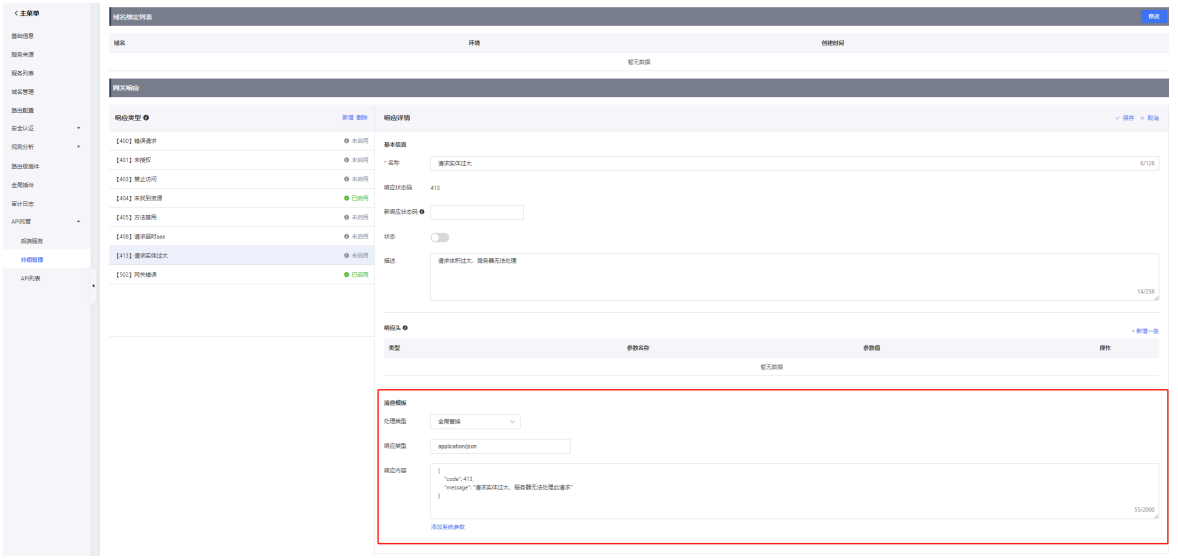


响应体

在基于模板创建的响应规则中，默认会将响应内容设置为 JSON 格式，并覆盖后端服务返回的响应内容。响应体处理规则可进行修改，支持两种替换方式：全局替换和规则替换。

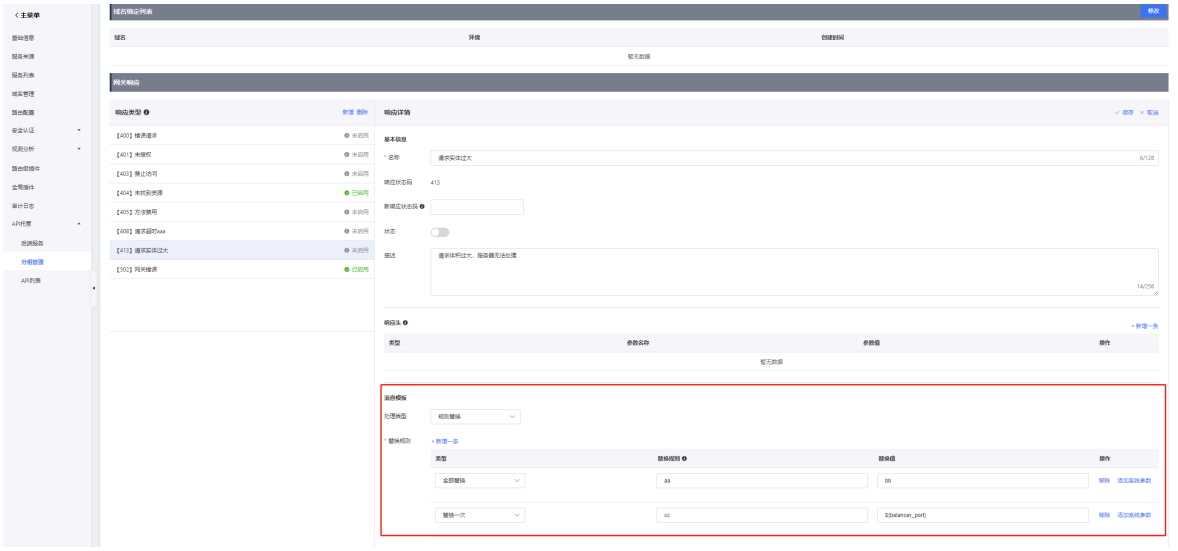
全局替换

根据配置的响应类型和具体的响应内容，直接覆盖后端服务返回的完整响应。



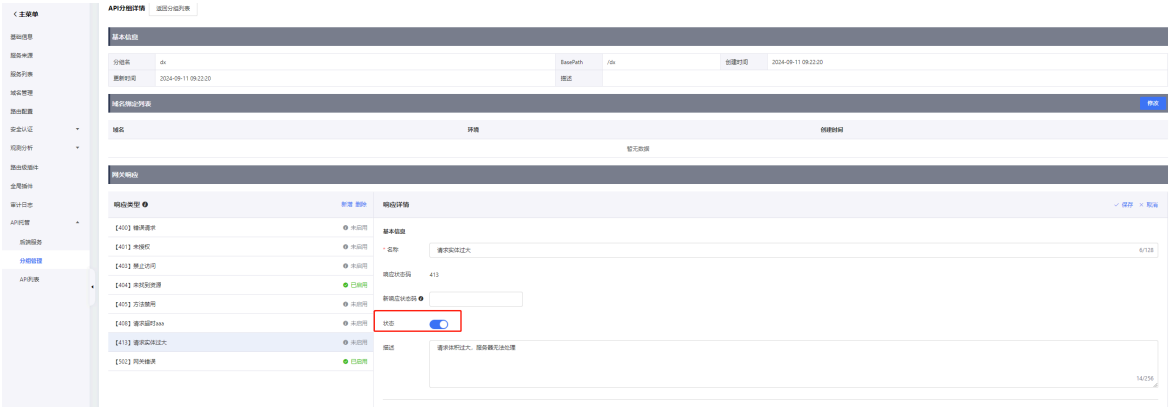
规则替换

根据配置的替换规则，对后端服务的响应体进行部分替换。支持使用常量或正则表达式进行匹配，匹配成功后可选择全量替换或仅替换首次匹配项。替换值可以是常量、系统参数，或两者的组合。



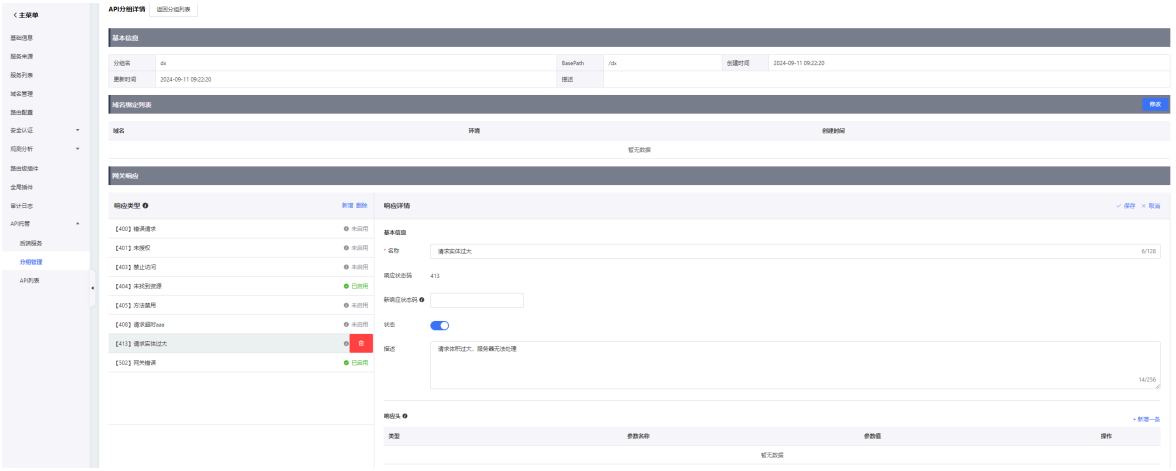
启用网关响应

新创建的网关响应规则默认未启用。要使其生效，请点击启用按钮并保存配置。必须配置响应头、响应体或两者，才能启用该规则。



删除网关响应

响应规则支持单个删除和全量删除。当鼠标悬停在某个响应规则上时，会出现红色的删除图标，点击即可单个删除该规则。新增按钮旁边的删除按钮用于删除该分组下的所有响应规则。删除操作不可撤销，请谨慎操作。



模型管理

概述

API分组模型主要用于对HTTP协议的请求数据和响应结果进行描述。网关通过在API分组中定义JSON Schema模型，来规范用户API中数据的组织方式，例如请求参数或返回值的字段等。目前分组模型主要用于在API定义的请求body或返回结果中引用，在API和SDK导出时，关联的模型会生成对应的文档内容，方便用户查看。

分组模型定义需要遵循规则：

1. 定义需符合Json Schema格式；
2. 仅支持创建元素属性为object类型的JSON Schema；
3. 目前暂不支持模型间的引用，即不支持模型定义中通过$ref引用其他模型。

可参考以下模型定义：

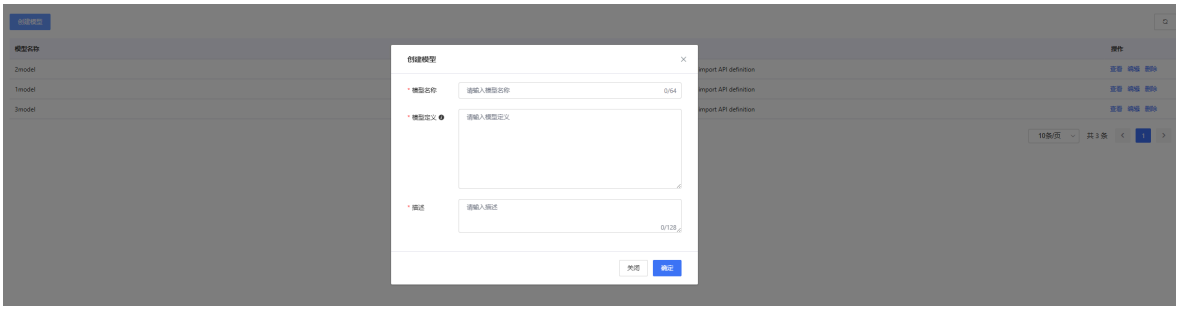
```
{
 "required": [
  "name"
```

```
],
"type": "object",
"properties": {
 "id": {
  "format": "int64",
  "type": "integer"
 },
 "name": {
  "pattern": "^\\d{3}-\\d{2}-\\d{4}$",
  "type": "string"
 },
 "dog": {
  "type": "object",
  "properties": {
   "id": {
    "format": "int64",
    "maximum": 100,
    "exclusiveMaximum": true,
    "type": "integer"
   },
   "name": {
    "maxLength": 10,
    "type": "string"
   }
  }
 }
}
}
```


创建模型定义

控制台入口：

1. 进入API托管->API分组菜单页。
2. 点击模型管理，进入模型管理页面，点击创建按钮，填写模型名称，定义和描述进行创建。



Swagger导入创建模型：

网关还支持通过导入Swagger定义的方式创建模型，Swagger文件中的Model内容会在该分组下自动生成模型。

注意

通过Swagger导入模型时，同名模型将直接被覆盖。

编辑模型定义

控制台入口：

1. 进入**API托管->API分组**菜单页。
2. 点击**模型管理**，进入模型管理页面，点击**编辑**按钮，可对已有模型进行更新

删除模型定义

用户可以对分组下的模型进行删除操作。

注意

网关不维护模型和API的关联关系，删除模型请谨慎操作。

导出API

概述

网关支持将API分组下的API按照OAS2.0和OAS3.0的格式导出，方便用户实现跨账号，跨地域间甚至跨平台的数据迁移。

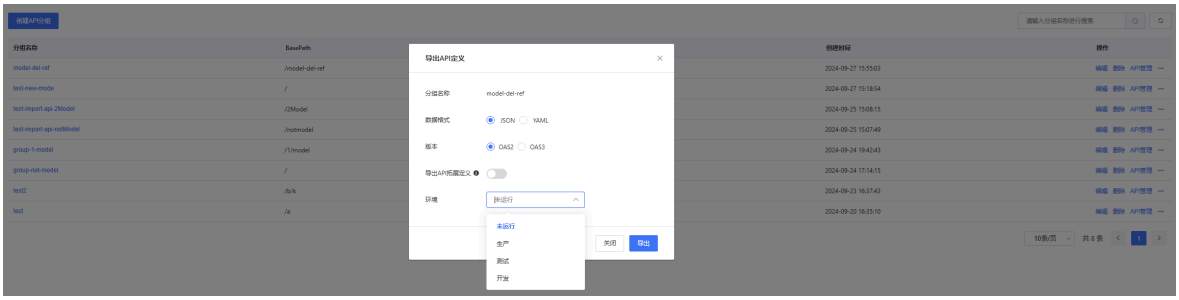
OpenAPI 规范（OAS），是定义一个标准的、与具体编程语言无关的RESTful API的规范。

导出标准OAS格式的API定义

网关目前支持两种API导出方式

通过分组导出API定义

1. 进入 **API 托管->分组管理** 菜单页。
2. 点击 **导出API定义** 按钮，在弹出框中指定导出的**数据格式，环境**以及是否需要 **导出API网关拓展定义** 。
3. 点击**导出**按钮后浏览器会下载API定义压缩包.



直接批量导出API定义

1. 进入 **API 托管->API管理** 菜单页。
2. 批量勾选API后在**批量操作**下拉框中点击**导出API定义** 按钮，在弹出框中指定导出的**数据格式，环境**以及是否需要 **导出API网关拓展定义** 。
3. 点击**导出**按钮后浏览器会下载API定义压缩包.

注意

批量导出API定义时仅支持导出同一个分组下的API

API扩展定义字段

如果在导出OAS规范的定义中选择了导出API网关扩展字段，在导出的API定义文件中会添加网关的扩展字段。扩展属性被设计为总是以 "x-" 为前缀的模式字段，此字段的值可以是 null、原始类型、数组或对象。可以包含任意有效的 JSON 格式的值。

扩展字段	OAS中位置	说明	类型
x-ctyun-apigateway-api-name	Operation	API名称	String
x-ctyun-apigateway-is-anti-replay	Operation	是否设置防重放攻击	Boolean
x-ctyun-apigateway-backend	Operation	后端服务定义	Object
x-ctyun-apigateway-common-parameters	Operation	常量参数定义	Object
x-ctyun-apigateway-system-parameters	Operation	系统参数定义	Object
x-ctyun-apigateway-request-argument-mode	Operation	入参请求模式	String
x-ctyun-apigateway-response-messages	Responses	错误码错误信息	String
x-ctyun-apigateway-parameter-demo	Parameter	参数定义示例值	String
x-ctyun-apigateway-backend-location	Parameter	后端服务参数映射位置	String
x-ctyun-apigateway-backend-name	Parameter	后端服务参数映射名称	String
x-ctyun-apigateway-success-demo	Operation	返回结果示例	String
x-ctyun-apigateway-failed-demo	Operation	失败结果示例	String
x-ctyun-apigateway-request-body-schema	Operation	请求body引用的模型名	String
x-ctyun-apigateway-request-body-description	Operation	请求body内容描述	String

说明

1. 导出OAS2格式的定义时,会导出分组的base path;
2. 每次可以导出的API数量限制为50个API;
3. 分组模型管理里中定义的模型定义如果无法解析，导出的内容中将不包含模型定义;
4. 以API为单位导出时，所有勾选的API会导出到一个文件中;
5. 未勾选导出API网关扩展字段时，导出OAS文件中的OperationId为API定义的请求path和请求method的拼接后的字符串，如“exportlByGET”;
6. 勾选导出API网关扩展字段后，在API网关定义API以外的内容，如绑定的插件、授权的APP、分组的域名、后端服务在各环境上的定义等，均不会被导出。

API导出样例

以下是一个OAS2版本的导出定义示例:


```
{
 "swagger":"2.0",
 "info":{
  "description":"Export from api group group4",
  "version":"1.0.0",
  "title":"Ctyun Gateway API"
 },
 "basePath":"/user1",
 "paths":{
  "/uhe/:c":{
   "post":{
    "summary":"",
    "operationId":"uhecByPOST",
    "schemes":["http", "https"],
    "parameters":[{
     "name":"a",
     "in":"query",
     "description":"",
     "required":false,
     "type":"string",
     "x-ctyun-apigateway-parameter-demo":"1",
     "x-ctyun-apigateway-backend-location":"query",
     "x-ctyun-apigateway-backend-name":"a"
    },{
     "name":"b",
     "in":"header",
     "description":"",
     "required":false,
     "type":"string",
     "x-ctyun-apigateway-parameter-demo":"2",
     "x-ctyun-apigateway-backend-location":"header",
     "x-ctyun-apigateway-backend-name":"b"
    },{
```

```
      "name" : "c",
      "in" : "path",
      "description" : "",
      "required" : false,
      "type" : "string",
      "x-ctyun-apigateway-parameter-demo" : "test",
      "x-ctyun-apigateway-backend-location" : "path",
      "x-ctyun-apigateway-backend-name" : "change"
     } ],
     "x-ctyun-apigateway-request-argument-mode" : "PASS",
     "x-ctyun-apigateway-common-parameters" : [ {
      "paramName" : "num",
      "paramValue" : "2",
      "paramLocation" : "query",
      "description" : ""
     } ],
     "x-ctyun-apigateway-api-name" : "testaa-clone-clone",
     "x-ctyun-apigateway-system-parameters" : [ ],
     "x-ctyun-apigateway-backend" : {
      "gwServiceCode" : "ea180dd257714a56af39d3408fdf072b",
      "postPath" : "/ko/[change]/",
      "postMethod" : "POST",
      "timeout" : 10000.0
     },
     "x-ctyun-apigateway-is-anti-replay" : false
    }
  },
 "definitions" : {
  "schema3" : {
   "type" : "object",
   "required" : [ "name" ],
   "properties" : {
    "id" : {
     "type" : "integer",
     "format" : "int32"
    },
    "name" : {
     "type" : "string"
    },
    "status" : {
     "type" : "string"
    },
    "dogProject" : {
     "type" : "object",
     "properties" : {
      "name" : {
```

```
        "type":"string"
       },
       "id":{
        "type":"integer",
        "format":"int32"
       }
      }
     }
    }
   }
  }
}
```

API管理

API管理

概述

云原生网关的API管理集中控制API的路由、流量和安全。用户可以配置API路由规则、请求处理、认证授权等功能，确保高效、安全的访问。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > API列表**。

创建API

API的创建过程分为四个步骤：

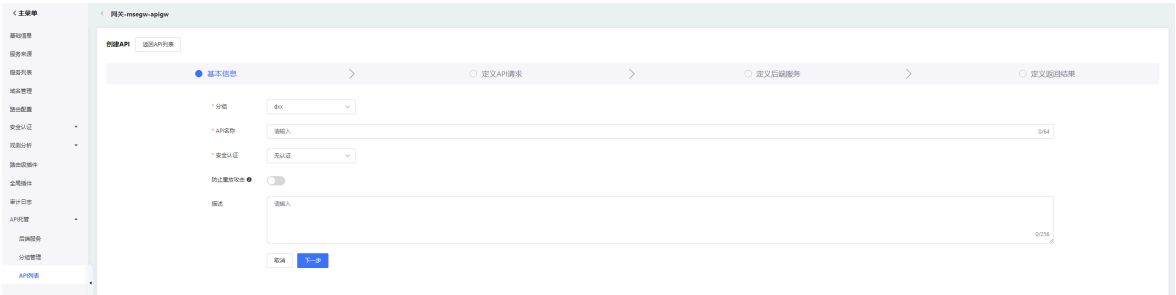
1、基本信息

分组：API所属的分组，创建后不可修改。

API名称：可填写API的业务含义。

安全认证：选择API的安全认证方式，开启应用授权后，只有授权的应用才能访问该API。

防重放攻击：开启后，请求头必须包含x-ca-timestamp和x-ca-nonce参数。



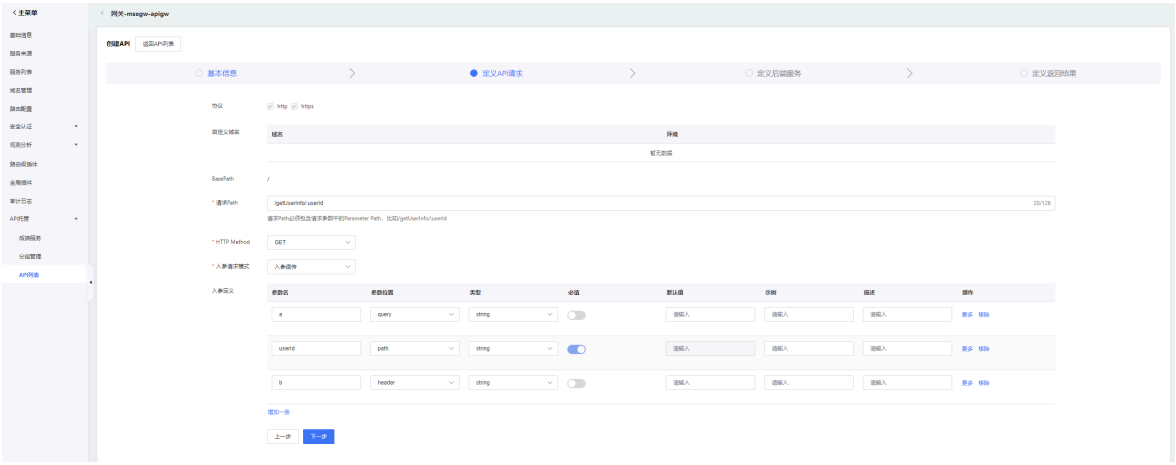
2、定义API请求

请求路径（Path）：API的访问路径。如果分组设置了BasePath，API的最终访问路径将为BasePath+Path。创建后不可修改。

请求方法（HTTP Method）：指定API的请求方式。创建后不可修改。

入参请求模式：支持三种模式，入参映射（过滤未知参数）、入参映射（透传未知参数）、入参透传。

入参定义：支持在query、header、path中定义参数。如果在path中定义了参数，必须确保该参数包含在Path中。



3、定义后端服务

后端服务：负责处理请求的后端服务。

后端请求路径（Path）：网关转发请求到后端服务的路径。如果在Path中定义了参数，后端请求路径中必须包含相应参数。

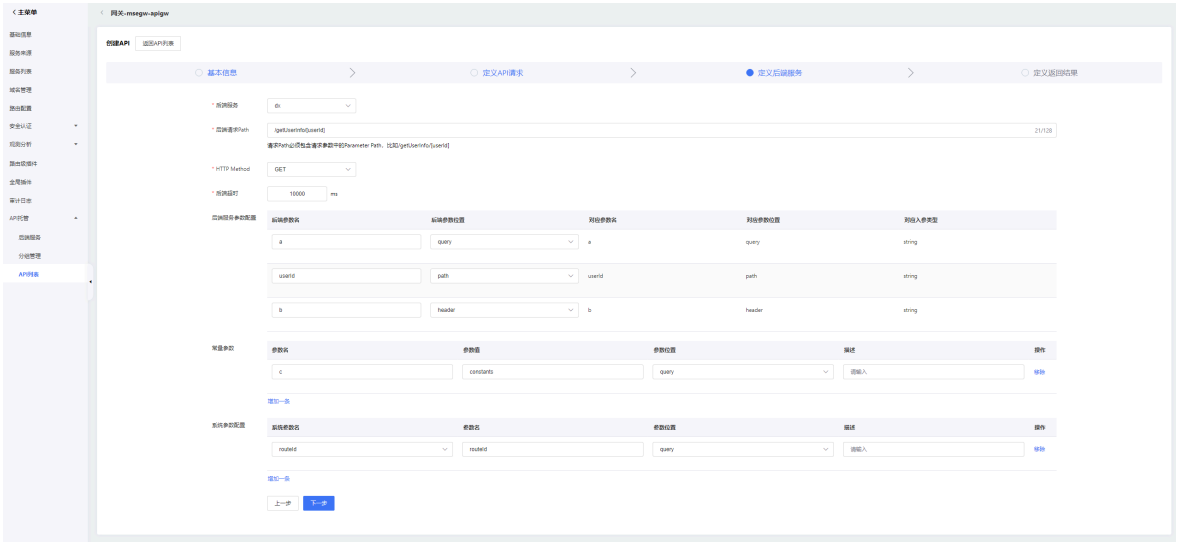
请求方法（HTTP Method）：指定请求后端服务的方式。

后端超时：设置请求后端服务的超时时间。

后端服务参数配置：将请求的入参与后端服务的入参一一对应，支持修改请求参数名。

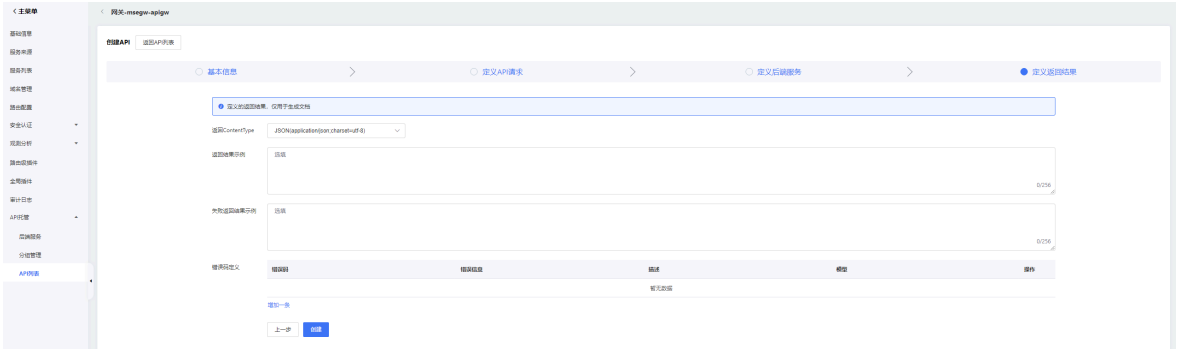
常量参数：可以添加常量参数，并透传到后端服务。

系统参数配置：支持添加系统参数，并透传到后端服务。



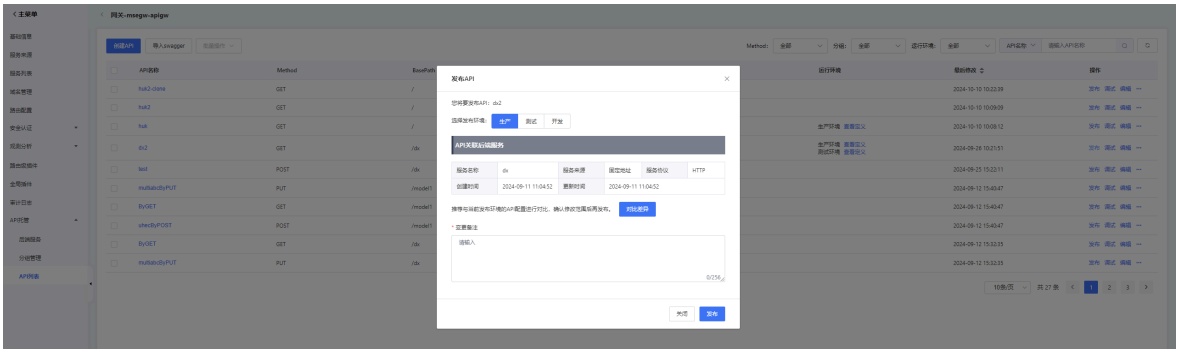
4、定义返回结果

这里设置的API返回结果，仅用于生产文档，不会对API的请求、响应产生影响。



API发布

创建或修改API后，不会立即生效，需进行发布操作。API支持多环境管理，发布时需选择要发布的目标环境。



API下线

已发布的API，支持下线操作。点击对应API右侧省略号的**下线**按钮可进入。

API克隆

API支持复制，会基于当前API，创建一个新的API。

API删除

API支持单个、批量删除操作。已发布的API，需要先下线后，才能进行删除。

绑定插件

概述

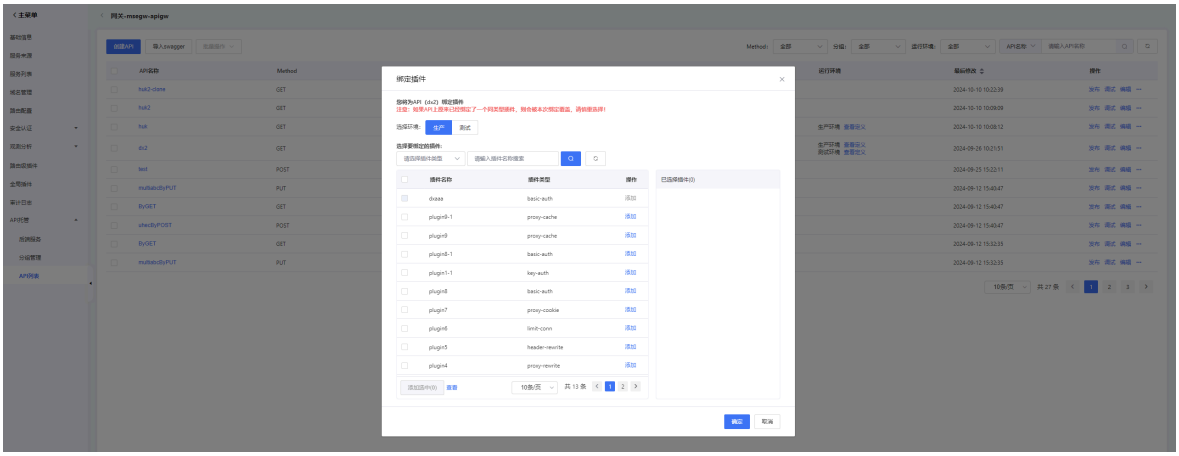
插件是云原生网关的扩展机制，用于增强网关功能，支持动态加载，能够在请求的不同阶段（如请求处理、转发、响应）执行自定义逻辑。插件支持环境隔离，仅在所绑定的特定环境中生效。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > API列表**。

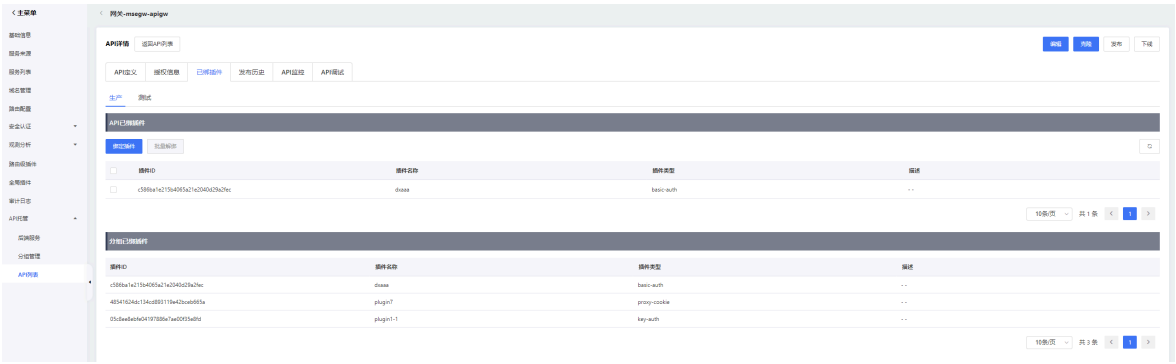
绑定插件

API只能绑定路由级插件，需先在路由级插件页面创建插件。然后在API列表页中，点击API右侧省略号中的**绑定插件**进入插件绑定页面。只有发布后的API，才能绑定插件。将所需插件添加到右侧区块，点击确定后即可完成批量绑定。每个API只能绑定一个同类型插件，若已有同类型插件，本次绑定会覆盖。同时，同类型插件中，API绑定的优先级高于分组绑定的插件。



解绑插件

API的插件支持解绑，进入API详情页，在已绑插件区块，可单个、也可批量进行插件解绑。



API授权

概述

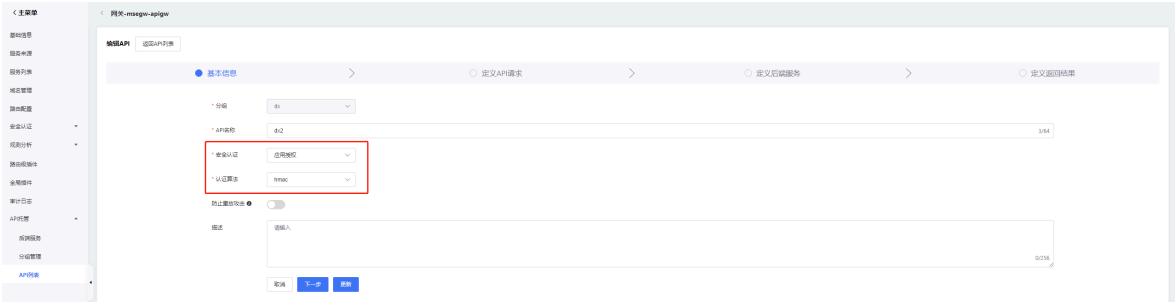
API授权用于保护API安全访问，确保只有授权用户能调用API。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > API列表**。
4. 点击进入对应的API详情页，上方导航栏点击**授权信息**。

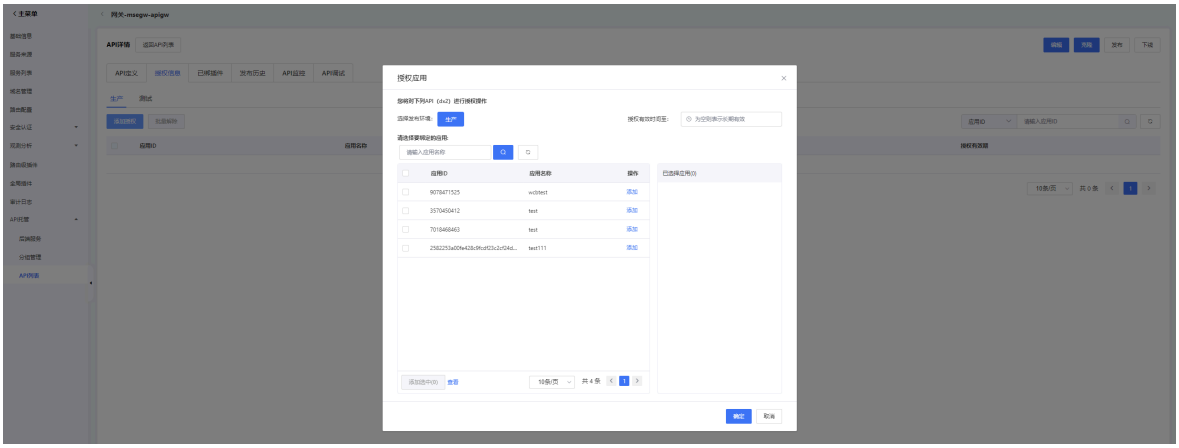
开启API授权

只有开启了应用授权的API，才能进行授权访问。可在创建时开启，也可编辑API，安全认证选择应用授权开启。



应用授权

应用授权支持环境隔离，只有API发布到指定环境后才能进行授权。将需要授权的应用添加到右侧区块，点击确定即可完成批量授权。授权可设置有效期或选择长期有效。



解除应用授权

在API应用授权信息列表中，可选择授权应用进行移除。

关闭应用授权

编辑API，安全认证方式，选择无认证，即可关闭应用授权。

发布历史

概述

云原生网关会记录API每一次的发布快照。您可在发布历史中查看每一次发布的API详细信息。并且可以选择任何一次快照进行重新发布，实现版本切换。

操作步骤

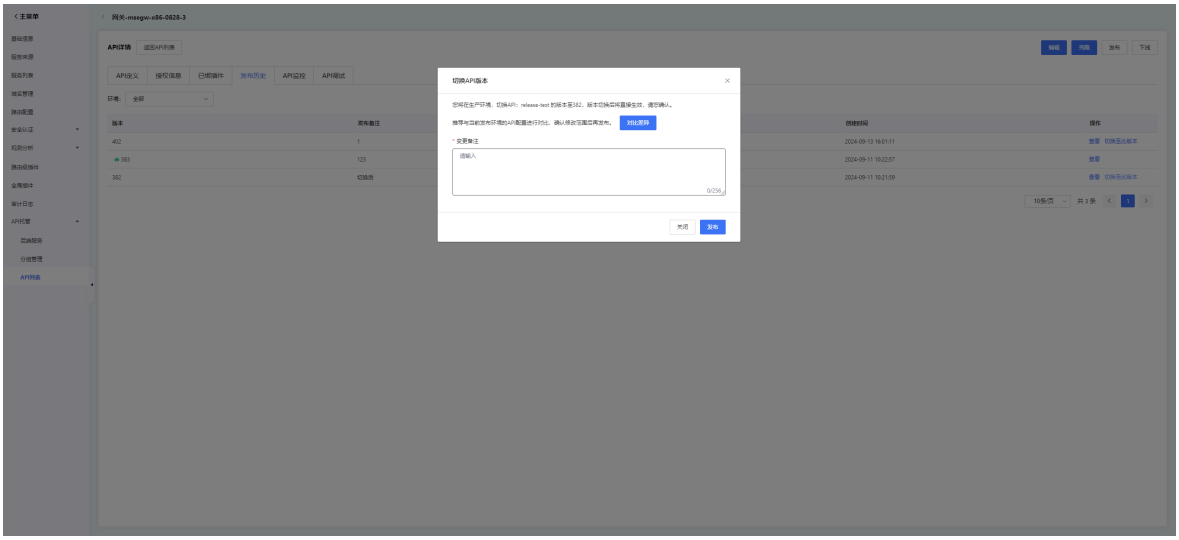
1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > API列表**，进入要查看的API详情。
4. 在上面导航栏，选择**发布历史**，即可查看API的发布历史，可选择不同环境进行过滤。

查看API快照

在发布历史列表中，选择对应版本，可以查看API快照。

版本切换

点击**切换至此版本**，可以指定历史版本重新发布到对应的环境中。



API监控

概述

云原生网关支持API维度的监控。支持QPS、请求成功率、平均延迟等指标。如果要查看API监控，需要先开通**应用性能监控**产品。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > API列表**，进入要查看的API详情。
4. 在上面导航栏，选择**API监控**，即可查看。

API调试

概述

API调试功能允许开发者在云原生网关中测试和验证API的行为。通过调试界面，用户可以发送请求，查看响应，检查请求参数和返回数据。该功能帮助快速识别和解决问题，确保API的正确性和稳定性。调试支持不同环境的测试，增强了开发和维护的效率。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择**云原生网关 > 网关列表**，进入对应网关实例的控制台。
3. 在左侧导航栏，选择**API托管 > API列表**，进入要查看的API详情。
4. 在上面导航栏，选择**API调试**，即可使用。

导入API

概述

网关支持导入OAS2.0和OAS3.0格式的API定义，帮助用户快速将现有系统的API接入网关。API网关既支持标准OAS格式的API定义的导入、也支持OAS定义扩展后的API定义导入。

导入标准OAS格式的API定义

导入API的流程主要包括:

- 选择基本信息(必填)与全局配置(选填).
- 文本填写或文件导入OAS 2.0或OAS 3.0格式的API定义.
- 进行预检并根据检查结果进行调整.
- 执行导入,成功导入后会生成对应的API和分组模型.

对应控制台操作为:

1, 进入**API托管->API管理**菜单页.

2, 在API列表页点击**导入Swagger**按钮,进入API导入配置页面. a. 基本信息,必填

参数	描述
导入分组	提供分组下拉列表，当API定义版本为OAS2时，用户填写的basePath需与所选分组的basePath一致，否则会预检失败
是否覆盖	当选择覆盖时,如果遇到请求路径和http请求冲突时，自动覆盖原有API；当不选择覆盖时,如果遇到请求路径和http请求冲突时，返回错误提示API已经存在
API定义版本	可选OAS2.0和OAS3.0。会校验所填写的API定义与所选版本是否适配
后端服务	如果在OAS定义中未填写"x-ctyun-apigateway-backend"扩展字段值，则以该服务作为后端服务；否则优先使用OAS中定义的服务信息

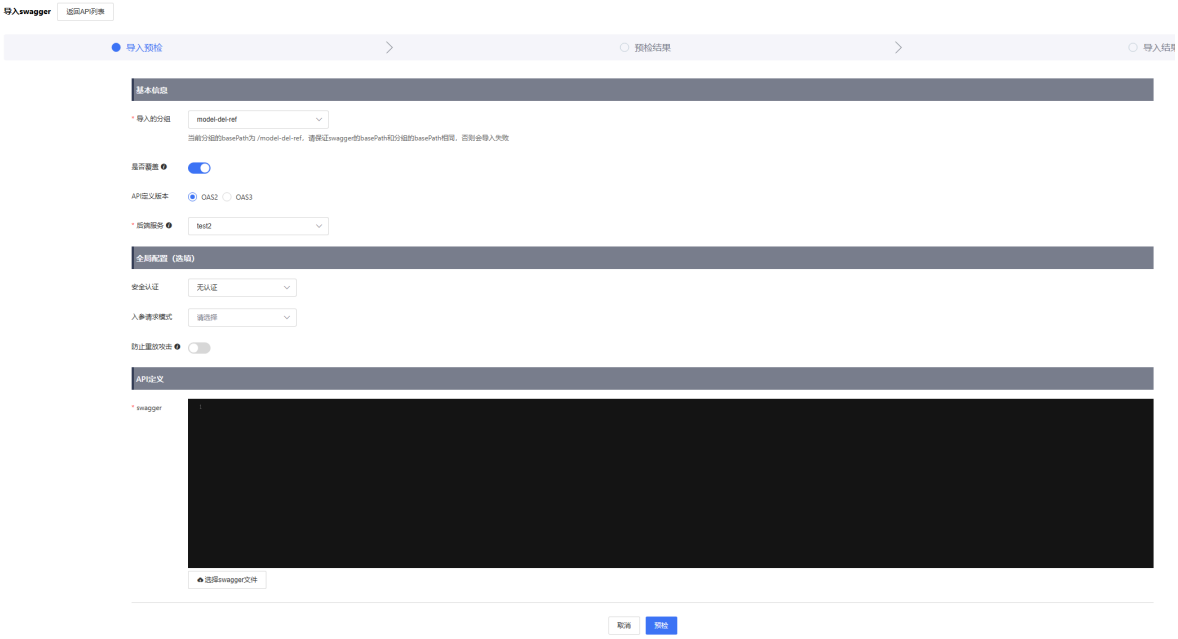
b. 全局配置,选填.主要是对入参请求模式,防止重放攻击进行API全局设置。如果用户OAS定义中通过扩展字段填写了该配置信息,则优先使用用户配置.

c. API定义,确定计划导入的OAS定义内容，支持文本填入yaml或json格式的API定义内容,也支持上传本地API定义文件。

3，单击**预检**按钮，系统将会对计划导入的内容进行检查，检测出OAS定义中的API定义和模型定义，以及告警和错误信息。

4，预检过程中没有错误信息和警告信息（或者忽略警告信息之后），可执行导入。单击**导入Swagger**按钮，系统真正开始导入API，导入API需要一定时间，导入过程请勿关闭浏览器。

5，待系统执行完毕后，可查看到API导入结果。



标准OAS2.0格式与API网关的对应

本章节重点介绍OAS 2.0的格式定义与网关的API定义有对应的映射字段，对于没有对应映射的OAS中的定义，不会影响API的导入。

1. Swagger Object：
   a. BasePath：对应分组的BasePath，导入时需要认证；
2. Path Item Object：
   a. Path：对应API定义中请求部分的请求path；
   b. Method：对应API定义中请求部分的Method，支持GET，POST，PUT，HEAD，DELETE，PATCH，OPTIONS；
3. Operation Object：
   a. Summary：对应API定义的描述；
   b. OperationId：对应API定义的API名称，如果存在扩展字段“x-ctyun-apigateway-api-name”则使用扩展字段中的内容作为API的名称；
   c. Schemes：对应API定义中请求部分支持的协议，优先级高于Swagger Object中的定义；

4. Parameter Object：

   a. Name：对应API定义中请求部分的参数名称；

   b. In：对应API定义中请求部分的参数位置，支持path，query，head，formdata；

   c. Description：对应API定义中请求部分的参数描述；

   d. Required：对应API定义中请求部分的参数是否必填；

5. Response Object：

   a. HTTP Status Code：对应API定义中定义返回结果中错误码定义中的错误码；

   b. Description：对应API定义中定义返回结果中错误码定义中的描述；

   c. Schema：对应API定义中定义返回结果中错误码定义中的模型；

6. Definitions Object：

   该项定义中的对象，在导入的过程中，API网关会将模型创建在分组下，查看模型定义可以通过分组管理列表下，对应分组的操作栏中的模型管理查看。

标准OAS3.0格式与API网关的对应

本章节重点介绍OAS 3.0的格式定义与API网关的API定义有对应的映射字段，对于没有对应映射的OAS中的定义，不会影响API的导入。

1. Path Item Object

   a. Path：对应API定义中请求部分的请求path；

   b. Method：对应API定义中请求部分的Method，支持GET，POST，PUT，HEAD，DELETE，PATCH，OPTIONS；

2. Operation Object

   a. Summary：对应API定义的描述；

   b. OperationId：对应API定义的API名称，如果存在扩展字段“x-ctyun-apigateway-api-name”则使用扩展字段中的内容作为API的名称；

3. Parameter Object

   a. Name：对应API定义中请求部分的参数名称；

   b. In：对应API定义中请求部分的参数位置，支持path，query，head，formdata；

   c. Required：对应API定义中请求部分的参数是否必填；

4. Schema Object：

   a. Description：对应API定义中请求部分的参数描述；

   b. Type：对应API定义中请求部分的参数类型；

5. Responses Object

   a. HTTP Status Code：对应API定义中定义返回结果中错误码定义中的错误码；

   b. Description：对应API定义中定义返回结果中错误码定义中的描述；

   c. Content：对应API定义中定义返回结果中错误码定义中的模型；

导入具有API网关扩展字段的OAS格式定义

swagger支持导入具有API网关扩展字段的OAS格式定义

扩展字段	OAS中位置	含义	类型	示例	说明
x-ctyun-apigateway-api-name	Operation	API名称	String	“api-name-test”	优先于operationId的值作为api名称
x-ctyun-apigateway-is-anti-replay	Operation	是否设置防重放攻击	Boolean	true	只支持true和false，默认false
x-ctyun-apigateway-backend	Operation	后端服务定义	Object	{ "gwServiceCode":"ea9180dd257714a56af39d3408fcf07d","postPath":"/", "postMethod":"GET","timeout":10000.0}	gwServiceCode为已创建的API服务Code，必填;·postPath为后端请求path，不填写则默认与api中的path一致；·postMethod为后端转发method，不填写则与api中的method一致；·timeout为后端服务超时时间，单位是ms，不填写默认为10000.
x-ctyun-apigateway-common-parameters	Operation	常量参数定义	Object	[{"paramName":"spuinstId","paramValue":"swdeefd","paramLocation":"query","description":""},{"paramName":"version","paramValue":"v1","paramLocation":"header","description":""}]	该字段的值是一个对象列表，可填写多个常量参数。·paramName为常量参数名称·paramValue为常量参数值·paramLocation为常量参数位置，只支持query和header两种，不填写则默认query·description为描述信息

扩展字段	OAS中位置	含义	类型	示例	说明
x-ctyun-apigateway-system-parameters	Operation	系统参数定义	Object	[{"systemParamName":"routeName","paramName":"aaa","paramLocation":"query","description":""},{"systemParamName":"upstreamPort","paramName":"port","paramLocation":"header","description":""}]	该字段的值是一个对象列表，可填写多个系统参数。·systemParamName为系统参数名称，目前支持的有upstreamId,upstreamPort,routeId,routeName,serviceId,serviceName·paramName为对应的赋值参数名称·paramLocation为系统参数位置，只支持query和header两种，不填写则默认query·description为描述信息
x-ctyun-apigateway-request-argument-mode	Operation	入参请求模式	String	“MAPPING_FILTER”	支持三种入参请求模式：·MAPPING_FILTER：入参映射（支持过滤参数）·MAPPING_PASS:入参映射（透传未知参数）·PASS：入参透传不填写默认为PASS
x-ctyun-apigateway-response-messages	Responses	错误码错误信息	String	“This is errorCodemessage”	
x-ctyun-apigateway-parameter-demo	Parameter	参数定义示例值	String		
x-ctyun-apigateway-backend-location	Parameter	后端服务参数映射位置	String	“path”	只支持path,query和header，不填写时默认与api入参定义参数位置一致
x-ctyun-apigateway-backend-name	Parameter	后端服务参数映射名称	String	“abc”	不填写时默认与api入参定义参数名称一致

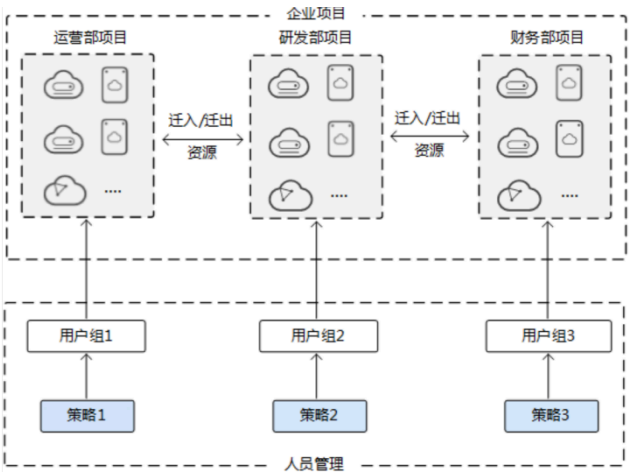
扩展字段	OAS中位置	含义	类型	示例	说明
x-ctyun-apigateway-success-demo	Operation	返回结果示例	String	“This is a failed demo”	
x-ctyun-apigateway-failed-demo	Operation	失败结果示例	String	“This is a success demo”	
x-ctyun-apigateway-request-body-schema	Operation	请求body引用模型名称	String	“people”	只有当请求方法为post，patch，put，delete，且body数据类型为Json时，该字段才生效，引用的模型名称应存在模型定义中
x-ctyun-apigateway-request-body-description	Operation	请求body内容描述	String		生效条件同上

主子账号

术语解释

概述

云原生网关目前已接入主子账号体系，通过主账号对子账号并对其分配访问权限，用户可以实现对企业中云服务和资源的访问及操作权限，避免账号滥用。主子帐号支持数据权限管理与功能权限管理，支持系统策略和自定义策略的授权方式。本章介绍如何使用主子账号体系进行权限管理。



术语解释

主账号：用户在天翼云注册后自动创建，该账号对其所拥有的资源具有完全的访问权限，可以重置用户密码、分配用户权限等。如果需要多人共同使用天翼云资源，由于账号是付费主体为了确保账号安全，建议创建子用户来进行日常管理工作。

子账号：主账号认证为企业账号后，在天翼云用户中心页面创建出来的账号。子账号的用户名、密码统一由主账号创建管理。子账号同样可以登录访问天翼云控制台，登录入口与主账号相同，受主账号赋予的权限限制。

企业项目：将云资源、企业成员按项目进行管理，通过企业项目将云资源、带有权限的用户组绑定到一起，用户使用项目内云资源的权限受用户组的授权限制。

说明

一个实例只能归属一个企业项目（可变更），一个子账号可以同时在多个企业项目中。

策略：是描述一组权限集的语言，它可以精确地描述被授权的资源集和操作集，通过策略，用户可以自由搭配需要授予的权限集。通过给用户组授予策略，用户组中的用户就能获得策略中定义的权限。

系统策略：系统预置的常用权限集，主要针对不同云服务的只读权限或管理员权限，比如对组件的只读权限、普通用户权限和管理员权限等等；系统策略只能用于授权，不能编辑和修改。

云原生网关中提供了两种系统策略：cgw viewer只读权限，只能对实例资源进行读操作；cgw admin管理权限，可以对实例资源进行管理维护，包括读与写操作，等同于主账号访问权限。

名称	类型	作用范围	描述	操作
cgw viewer	系统策略	全局	微服务引擎-云原生网关cgw-CTIAM默认浏览者策略, 对实例资源仅拥有只读权限	查看 | 编辑 | 删除
cgw admin	系统策略	全局	微服务引擎-云原生网关cgw-CTIAM默认管理者策略, 对实例资源拥有管理维护权限	查看 | 编辑 | 删除

自定义策略：创建自定义策略可以支持更细粒度的授权，对于云原生网关，可以自定义功能访问权限以及资源访问权限。

数据权限：看到的数据不一样。主账号看到所有实例，子账号只能看到所属项目中的实例。

功能权限：主账号可以进行所有控制台操作，子账号对单个组件实例拥有的操作权限由主账号授权。

功能权限授权：给子账号在企业项目A下增加一个策略，即代表该子账号对企业项目A下的实例拥有了策略中定义的权限，策略以外的操作会被禁止。

权限控制

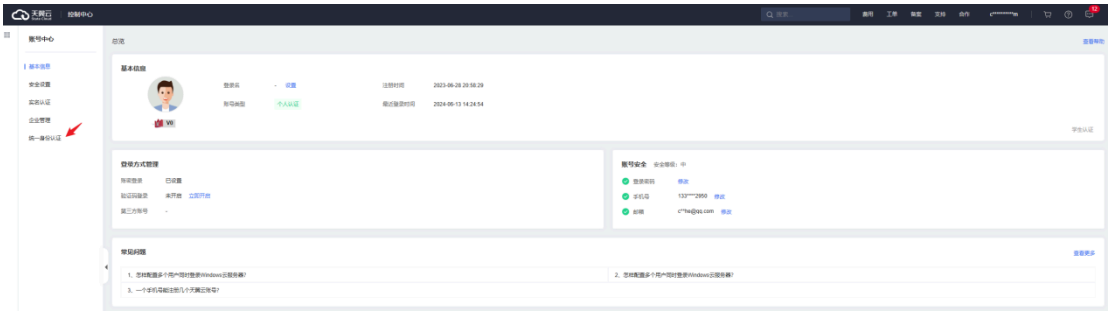
数据权限控制

数据权限用于控制主子帐号能访问的数据不一样。主账号看到所有实例，子账号只能看到所属项目中的实例。其权限码为统一值instance-list，只有在策略中授予instance-list的权限码才具备查看实例的权限。如果在用户组授权包含instance-list的策略，则该用户组的子用户能看到所有实例。如果在企业项目中的用户组授权包含instance-list的策略，则该用户组的子用户只能看到该企业项目下的实例。

操作步骤：

进入IAM管理页面：

- 登录天翼云官网，鼠标悬浮至右上角个人信息，点击个人信息进入账号中心页面
- 在帐号中心页面中，点击统一认证服务进入IAM管理页面

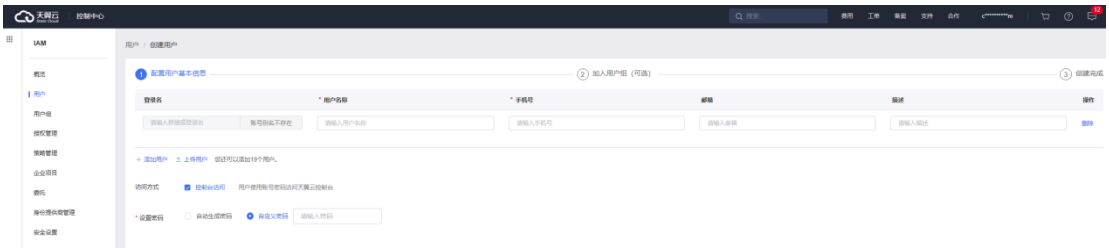


创建用户组：

- 在IAM管理页面中，点击左侧菜单栏中用户组进入用户组管理页面
- 在用户组管理页面中，点击创建用户组按钮，在弹出框中填写用户组名称与描述，点击确定即可

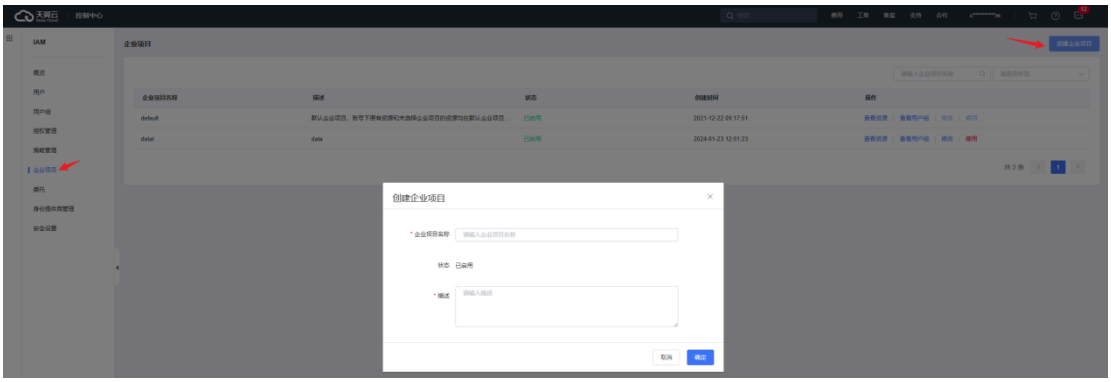
创建子用户：

- 在IAM管理页面中，点击左侧菜单栏中用户进入用户管理页面
- 在用户管理页面中，点击创建用户按钮，进入创建页面
- 在创建页面中，首先需要配置用户基本信息。填写用户名称、手机号、邮箱和密码等信息，点击下一步加入用户组
- 在加入用户组页面，选择对应的用户组，点击添加按钮加入已选用户组，点击下一步即可



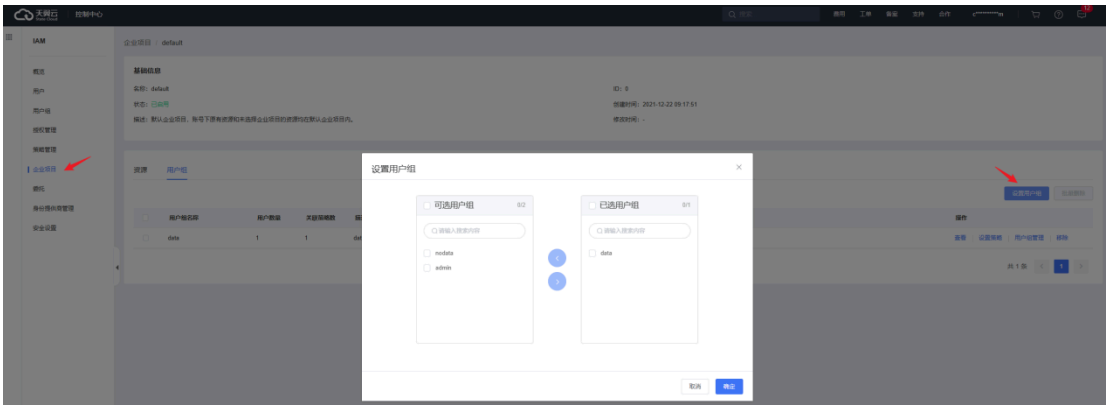
创建企业项目：

- 在IAM管理页面中，点击左侧菜单栏中企业项目进入企业项目管理页面
- 在企业项目管理页面中，点击创建企业项目按钮，在弹出框中填写企业项目名称与描述，点击确定即可



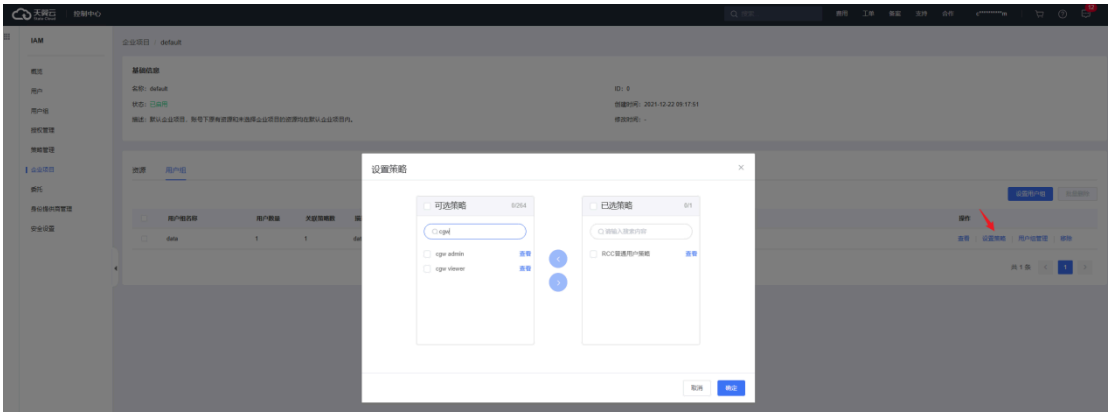
在企业项目中添加用户组：

- 在企业项目管理页面中，点击企业项目的查看用户组按钮进入企业项目的用户组管理页面
- 在企业项目的用户组管理页面中，点击设置用户组按钮，弹出设置用户组弹框
- 在弹出框中，选择用户组再点击>按钮加入已选用户组，最后点击确定按钮即可



对企业项目中的用户组设置策略：

- 在企业项目管理页面中，点击企业项目的查看用户组按钮进入企业项目的用户组管理页面
- 在企业项目的用户组管理页面中，点击用户组的设置策略按钮弹出设置策略弹框
- 在弹出框中选择对应策略，点击>按钮加入已选策略，最后点击确定按钮即可



功能权限控制

主账号可以通过给用户组授权策略实现对子账号的功能权限，策略包括系统策略和自定义策略。策略中可定义“允许”的操作和“拒绝”的操作，多个策略以”or”的关系进行评估，总体遵循Deny优先原则。

操作步骤：

进入IAM管理页面：

- 登录天翼云官网，鼠标悬浮至右上角个人信息，点击“个人信息”进入账号中心页面
- 在帐号中心页面中，点击“统一认证服务”进入IAM管理页面

创建用户组：

- 在IAM管理页面中，点击左侧菜单栏中“用户组”进入用户组管理页面
- 在用户组管理页面中，点击“创建用户组”按钮，在弹出框中填写用户组名称与描述，点击确定即可

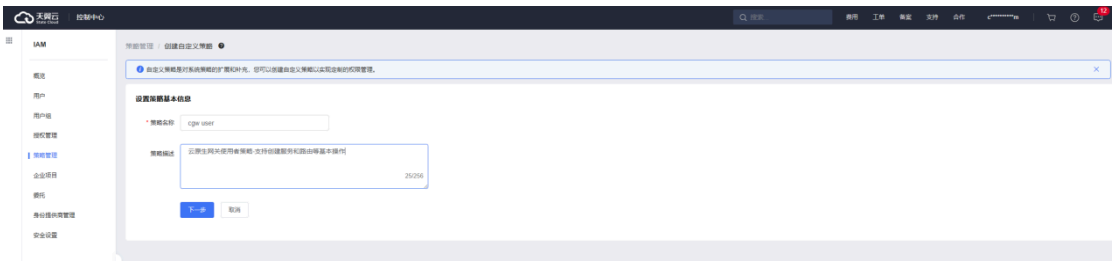
创建子用户：

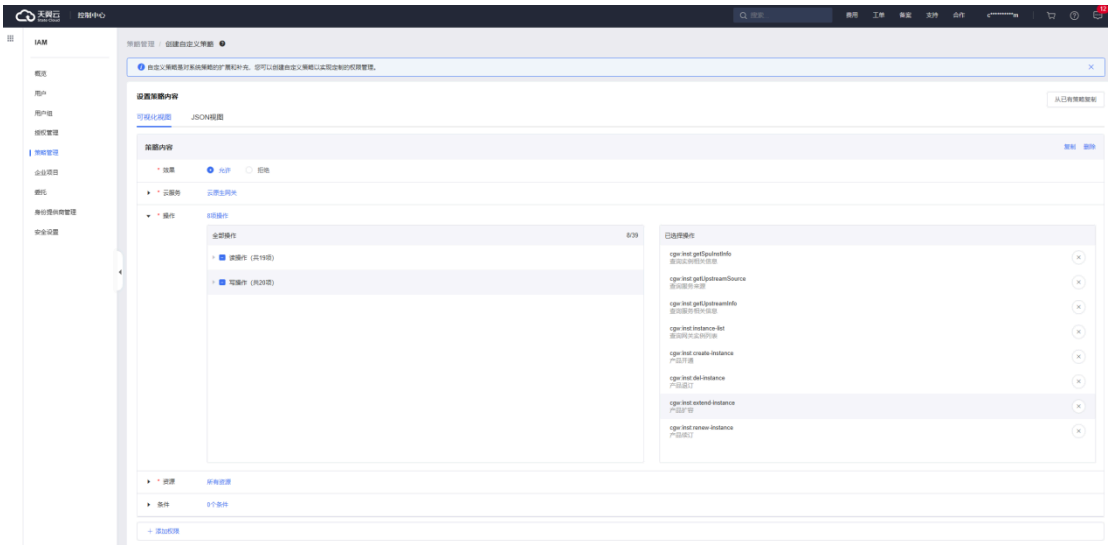
- 在IAM管理页面中，点击左侧菜单栏中“用户”进入用户管理页面
- 在用户管理页面中，点击“创建用户”按钮，进入创建页面

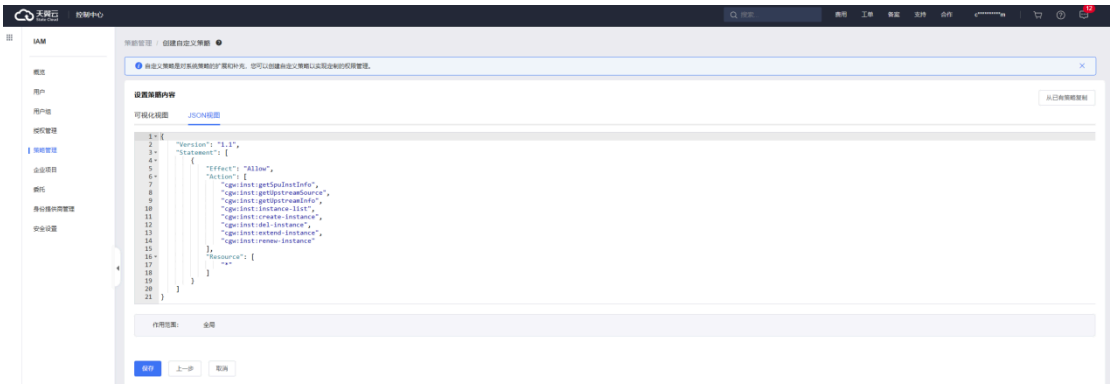
- 在创建页面中，首先需要配置用户基本信息。填写用户名称、手机号、邮箱和密码等信息，点击下一步加入用户组
- 在加入用户组页面，选择对应的用户组，点击“添加”按钮加入已选用户组，点击下一步即可

创建策略：

- 在IAM管理页面中，点击左侧菜单栏中“策略管理”进入用户组管理页面
- 在策略管理页面中，点击“创建自定义策略”，进入创建页面
- 在创建页面中，填写策略名称与策略描述，点击下一步
- 进入设置策略内容页面，可选择可视化视图和JSON视图。可视化视图中支持选择效果，云服务，操作，资源等信息，操作更灵活。JSON视图支持手动配置权限组，权限展示更加清晰。点击保存即成功创建自定义策略

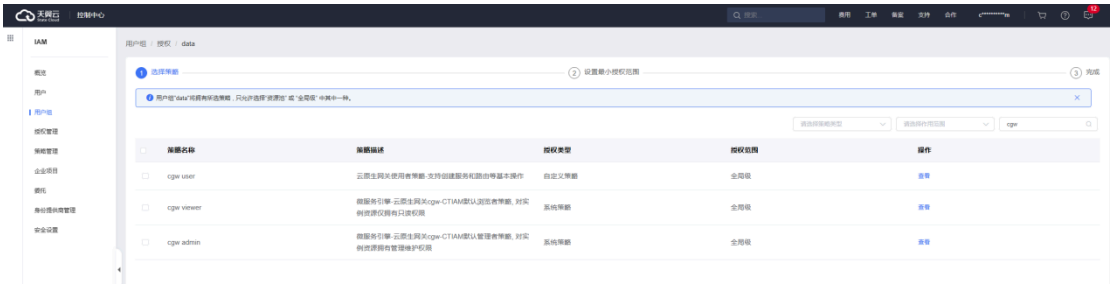






授权：

- 在IAM管理页面中，点击左侧菜单栏中“用户组”进入用户组管理页面
- 在用户组管理页面中，点击对应用户组的“授权”按钮，进入授权页面
- 在授权页面中，勾选对应策略，点击下一步进入设置授权范围页面
- 在设置授权范围页面中，选择授权范围，点击“确定”按钮就可完成授权



资源标签

概述

云原生网关目前支持在实例列表中的资源标签功能，提供为资源添加标签以及根据标签筛选资源的功能，本章介绍如何使用资源标签功能。

添加标签

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 光标悬浮至对应实例的标签列图标，点击“添加”按钮，会显示弹出框。

4. 在弹出框中点击“添加标签”按钮，选择已有标签键值对或者创建自定义标签键值对，点击“确定”按钮即可添加对应标签。

批量绑定标签

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 选择对应实例，点击批量标签>绑定标签按钮。

4. 弹出框显示已选择的实例资源，选择对应标签或填写自定义标签，点击便签右侧“确定”按钮，将对应标签加入已选择标签列表。

5. 点击弹出框右下角的确定按钮完成批量绑定标签。

批量解绑标签

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 选择对应实例，点击批量标签>解绑标签按钮。
4. 弹出框显示已选择的实例资源，选择对应标签，点击解绑按钮即可批量解绑标签。

修改标签

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 光标悬浮至对应实例的标签列图标，点击编辑按钮，会显示弹出框。
4. 在弹出框中，修改对应的标签键值对，点击修改按钮即可完成修改。

解绑标签

解绑标签操作会解绑对应标签与实例的关联关系，但不会删除对应标签，在选择标签时依然可见该标签。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 光标悬浮至对应实例的标签列图标，点击编辑按钮，会显示弹出框。
4. 在弹出框中，点击解绑按钮即可解绑对应的标签键值。

一键解绑标签

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 光标悬浮至对应实例的标签列图标，点击编辑按钮，会显示弹出框。
4. 在弹出框中，点击一键解绑按钮即可解绑该实例与所有标签的关联关系。

删除标签

删除标签操作会解绑对应标签与实例的关联并且删除对应标签，在选择标签时该标签已不存在。

操作步骤

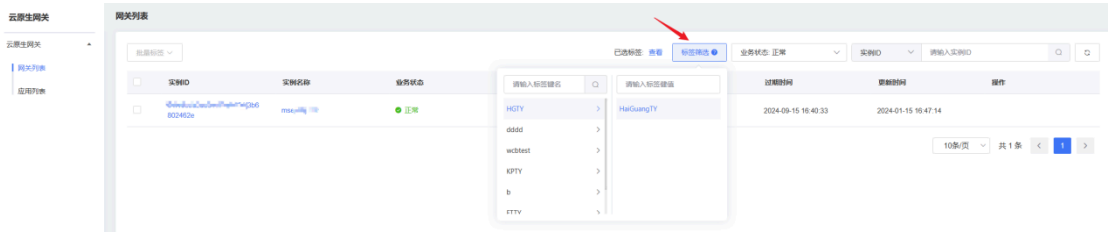
1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 光标悬浮至对应实例的标签列图标，点击编辑按钮，会显示弹出框。
4. 在弹出框中，在对应的标签键值对行的操作项中删除按钮即可完成删除。

标签筛选

标签筛选功能可以根据选择的标签筛选实例资源，每次最多同时筛选5个标签。只要实例资源拥有已选标签中的一个即会被筛选出来。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 点击标签筛选按钮，点击对应的标签键值对可将标签加入已选标签，即可根据已选标签筛选出对应实例资源。



审计日志

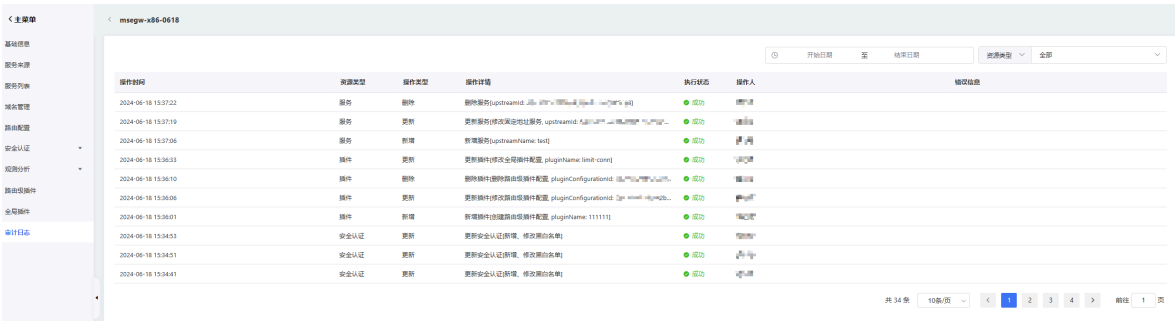
概述

您可以在云原生网关控制台上查看审计日志。如果当前帐号为主帐号，则可以查询当前帐号及其所有子账号在控制台上的操作日志；如果当前帐号为某个主账号下的子账号，则可以查询该子账号在控制台的操作日志。审

计日志用于收集云原生网关实例的相关操作日志，记录资源的变动情况。其中资源主要包括实例、安全认证、路由、服务、服务来源、插件、应用管理、域名、ELB和观测分析。变动情况包括增加、删除和修改。

操作步骤

1. 登录微服务引擎MSE云原生网关管理控制台，选择资源池。
2. 在左侧导航栏，选择云原生网关 > 网关列表。
3. 在左侧导航栏，选择审计日志。
4. 可选择开始-结束时间，资源类型，操作类型，操作人，操作详情进行过滤。



微服务治理中心

微服务环境管理

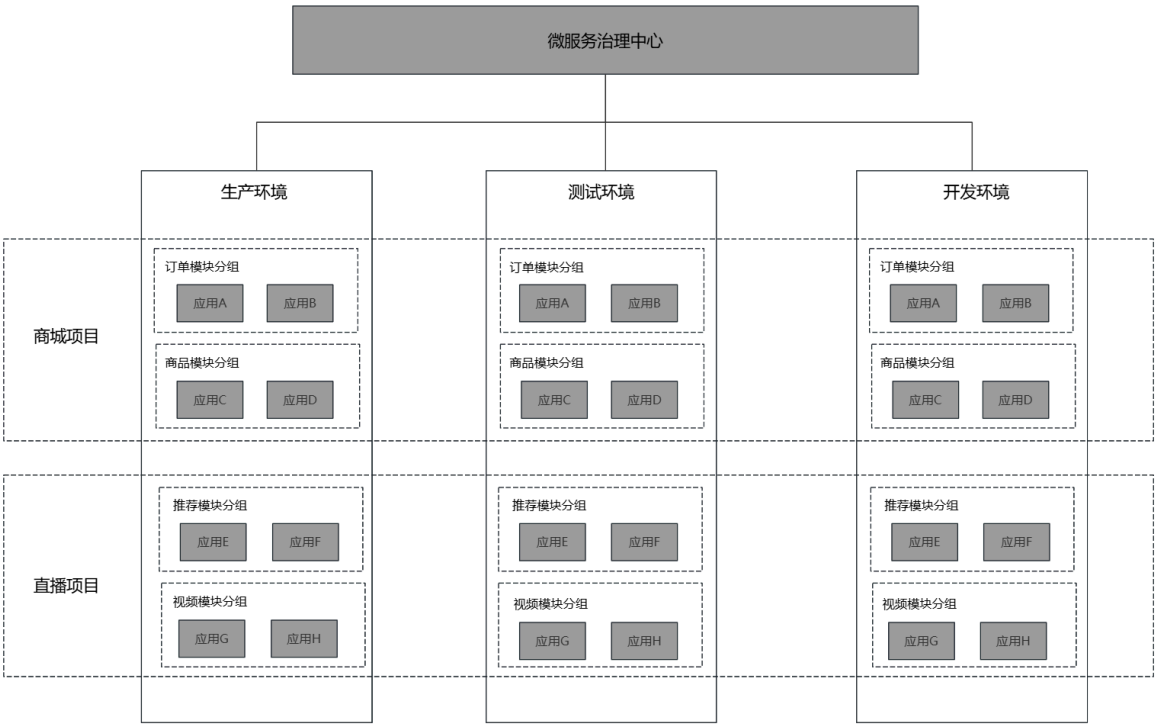
概述

微服务治理中心环境是基于微服务治理中心应用的上层概念，用于组织微服务应用。通过环境，用户可以实现同资源池下不同环境下微服务应用的区分，实现对每个环境下治理规则配置的严格隔离，应对多环境治理需求，保证微服务治理的准确性。

应用场景

用户通常具有生产、测试、开发环境，环境之间相互隔离。用户可以将生产应用全部接入生产环境，如prod，测试应用全部接入测试环境，如test。从而能有效隔离生产应用和测试应用的治理规则，降低对生产环境的影响，保障稳定性。

除环境以外，还提供应用项目和应用分组的能力。用户可以按业务将应用划分为不同的项目，如商城项目，按应用的职能划分为不同的分组，如商城项目中可以包含订单、商品、用户等分组。



若您无需使用环境、项目和分组的能力，也可以不创建环境、项目和分组，应用将接入默认环境。

前提条件

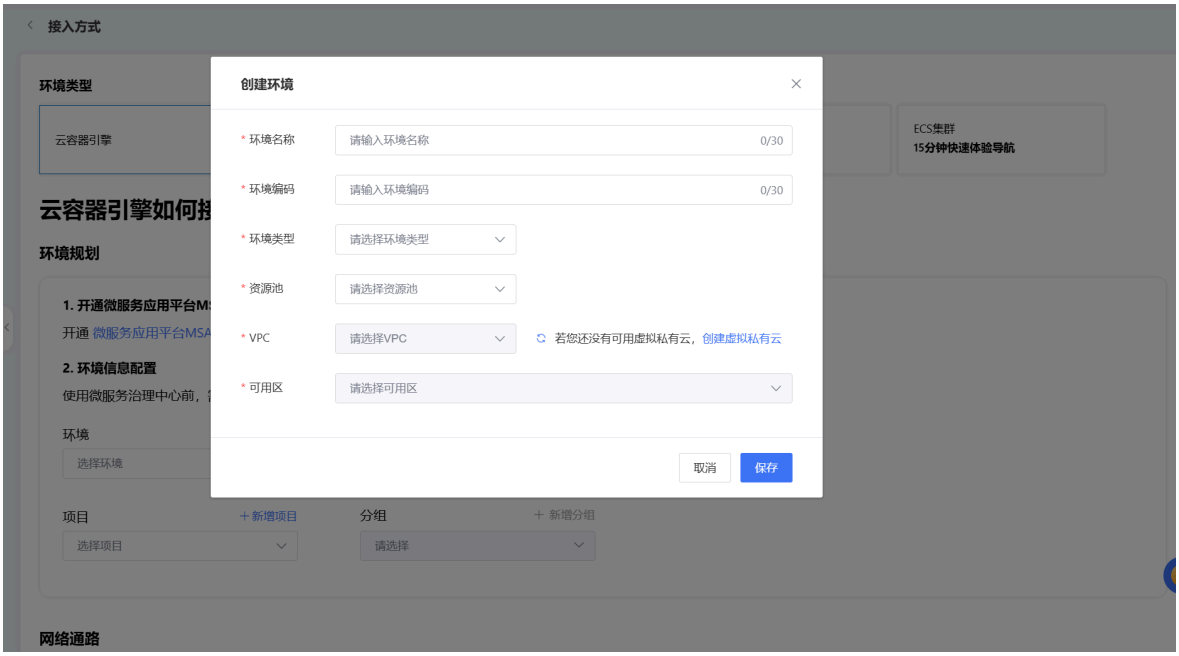
- 开通微服务云应用平台。

创建环境

- 登录微服务治理中心控制台。
- 在控制台左侧栏选择应用治理。
- 在应用治理页面点击左上角“应用接入”标签，进入应用接入导航页。

- 在应用接入导航的云容器引擎或ECS集群页面点击“新增环境”。

- 在“新增环境”弹窗填写环境名称、环境编码、环境类型、资源池、VPC、可用区等信息，点击保存完成创建。



环境参数说明：

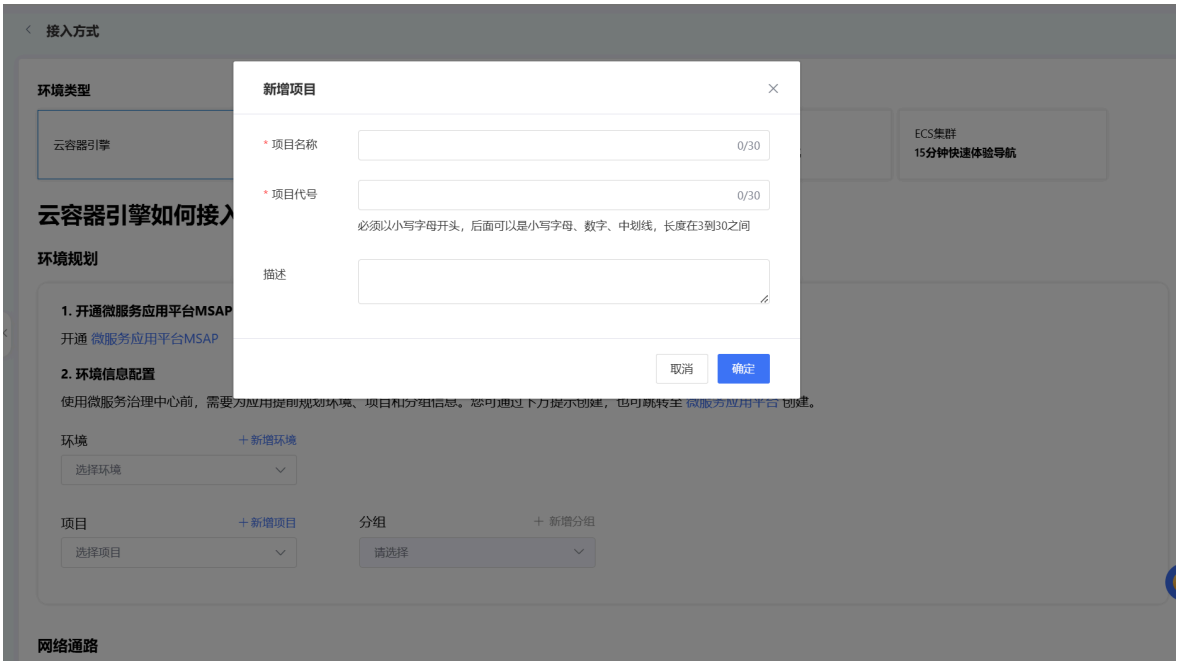
参数	说明
环境名称	当前环境的名称。
环境编码	环境对应的编码，输入环境名称后会自动生产编码，也可手动修改。
环境类型	可选开发环境、测试环境、预发环境或生产环境。
资源池	默认选中当前资源池。
VPC	当前资源池的VPC。
可用区	选中VPC包含的可用区。

创建项目

- 登录微服务治理中心控制台。
- 在控制台左侧栏选择应用治理。
- 在应用治理页面点击左上角“应用接入”标签，进入应用接入导航页。

- 在应用接入导航的云容器引擎或ECS集群页面点击“新增项目”。

- 在“新增项目”弹窗填写项目名称、项目代号等信息，点击确定完成创建。



项目参数说明：

参数	说明
项目名称	应用所属项目的名称。

参数	说明
项目代号	应用所属项目的代号。
描述	项目的描述。

创建分组

- 登录微服务治理中心控制台。
- 在控制台左侧栏选择应用治理。
- 在应用治理页面点击左上角“应用接入”标签，进入应用接入导航页。
- 在应用接入导航的云容器引擎或ECS集群页面，选择指定分组后，再点击“新增分组”。

- 在“新增分组”弹窗填写分组名称、分组代号和描述，点击确定完成创建。

分组参数描述：

参数	说明
分组名称	应用所属分组的名称。
分组代号	应用所属分组的代号。
描述	分组的描述。

删除环境、项目和分组

微服务治理中心控制台不提供环境、项目和分组的删除能力，请跳转至微服务云应用平台操作。

服务查询

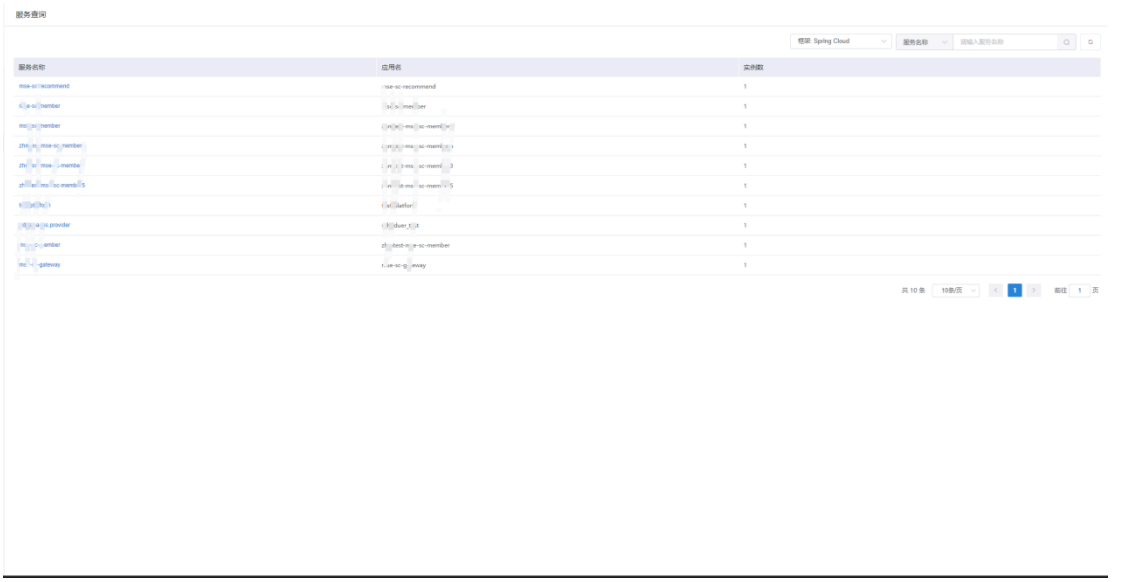
查询服务

概述

接入微服务治理中心以后，可以在控制台查看SpringCloud和Dubbo应用的服务列表和服务详情。其中服务是指注册到注册中心的SpringCloud和Dubbo服务，应用是指启动的实体。

查看服务列表

登录微服务治理中心控制台，在左侧导航栏选择微服务治理中心->服务查询，在服务查询页面，可通过筛选SpringCloud或Dubbo选项，来选择查看目标服务列表，列表展示服务名称，应用名，实例数。

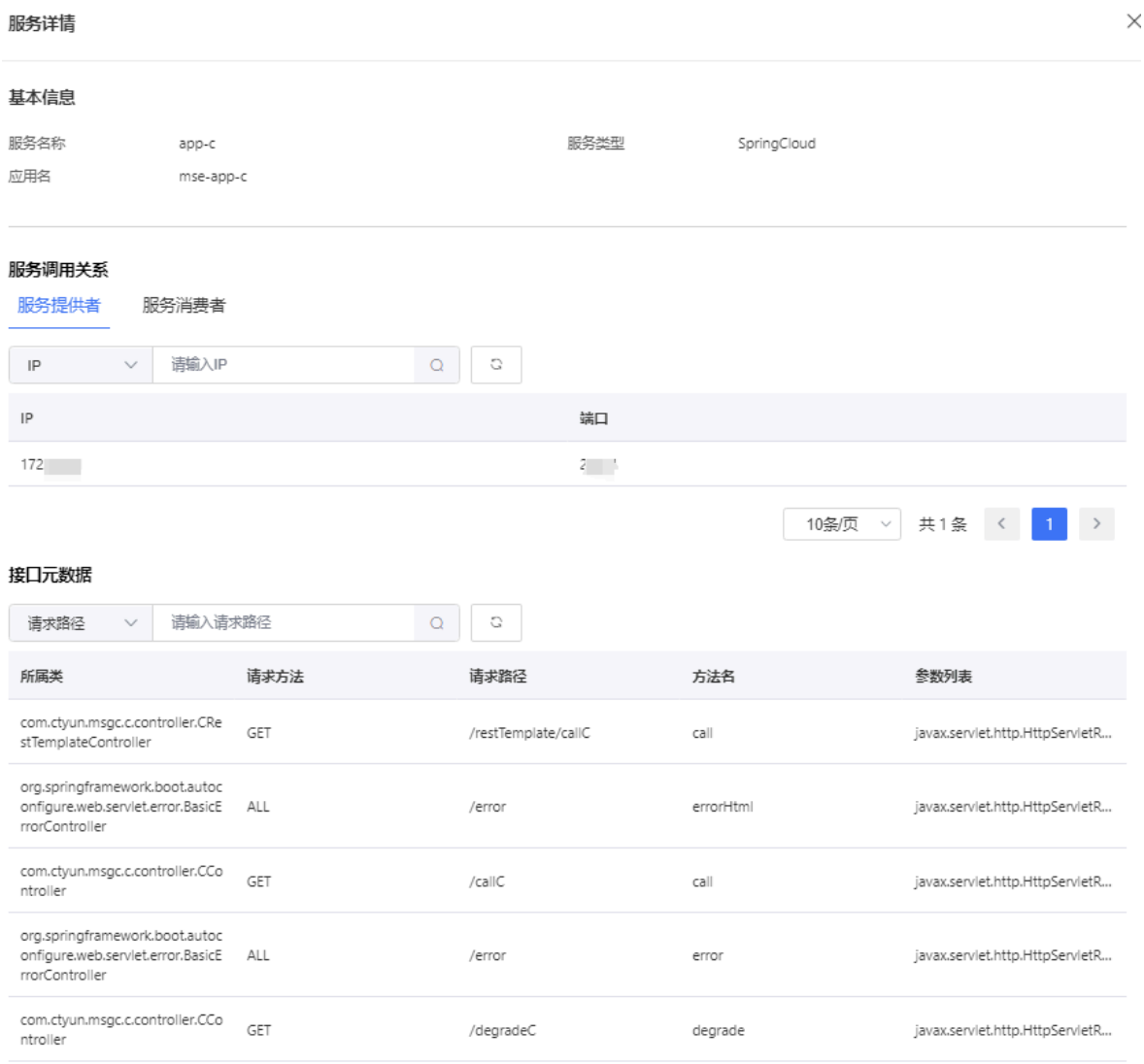


服务名称：指注册到注册中心的SpringCloud和Dubbo服务的服务名。

应用名：指通过启动参数mse.appName设置的应用名称。

查看服务详情

在服务列表页面单击服务名称，可以查看服务信息、服务调用关系、接口元数据。



如果服务是SpringCloud服务，包含如下信息：

- 基本信息：包含服务名称、服务类型、应用名称。
- 服务调用关系：包含提供者自身的地址列表和消费者的地址列表。服务提供者展示IP和端口信息，服务消费者展示IP和应用名称信息。
- 接口元数据：包含所属类、请求方法、请求路径、方法名和参数列表。

如果服务是Dubbo服务，包含如下信息：

- 基本信息：包含服务名称、服务类型、应用名称。
- 服务调用关系：包含提供者自身的地址列表和消费者的地址列表。服务提供者展示IP和端口信息，服务消费者展示IP和应用名称信息。
- 接口元数据：包含方法名、参数列表和返回数据。

查询服务契约

概述

在您应用接入微服务治理中心后，通过服务契约功能，您无需在应用中引入依赖，直接部署后，便可以通过服务契约在线查看微服务接口、路径等API信息，除了可以查看接口信息以外，还可以通过服务契约功能方便使用服务测试能力。

应用场景

开发测试联调

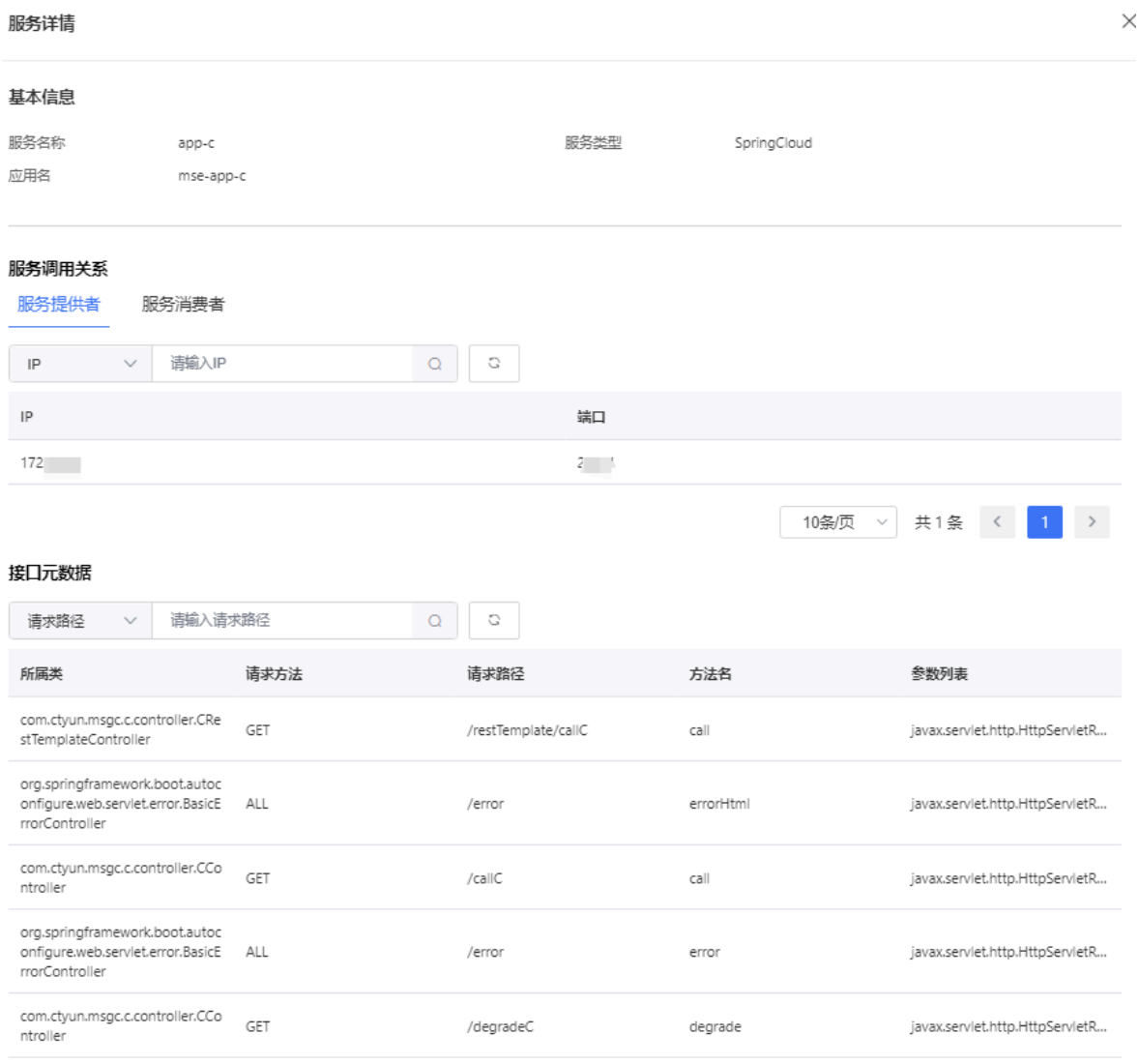
在前后端进行联调时，研发人员可以通过服务契约查看当前应用的接口信息，提升联调效率。

版本变更

在进行版本变更操作时，研发人员可以根据服务契约功能查看应用的接口是否有更新，从而判断版本是否发布成功。同时也可以结合服务测试功能，对新接口进行简单的测试验证。

操作步骤

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择服务查询。
3. 在服务查询列表页面点击服务名。
4. 在服务详情页面，可以看到服务接口的基本信息，接口元数据等信息。



应用治理

应用概览

概述

应用概览主要是方便用户直观的看到当前应用请求的整体状态，通过应用概览页面可以查看应用错误请求数、总请求数、平均响应时间、应用基础信息和服务列表等数据。

功能入口

1. 登录微服务治理中心控制台。
2. 在控制台左侧栏选择应用治理。
3. 在应用治理页面的应用卡片页面单击目标应用卡片。
4. 在左侧导航栏选择应用详情。

功能介绍

应用概览页面会展示错误请求数、总请求数、平均响应时间、应用基础信息和服务列表等数据。其中错误请求数、总请求数、平均响应时间都是应用入口接口的统计，不包括应用内部方法调用的统计。

错误请求数：展示近5分钟请求异常的总数。

总请求数：展示近5分钟应用入口接口的请求总数。

响应时间：展示近5分钟应用入口接口的平均响应时间。

应用信息：当前应用的基础信息，包括应用ID、应用名称、接入方式和应用框架类型。

服务列表：当前应用的服务列表，内容与服务查询页面一致。

流量治理

金丝雀发布

概述

金丝雀发布是指在应用发布时，可以为新版本的应用打上gray的标签，通过按流量比例路由或按内容路由的方式，将灰度流量引入带有gray标签的应用中，从而达到小规模验证的目的。

版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Edgware及以上版本	客户端：Feign、RestTemplate；负载均衡：Ribbon、LoadBalancer。
Dubbo	2.5.3+	支持Alibaba Dubbo、Apache Dubbo。
注册中心	Nacos、Eureka、Zookeeper	无。
jdk版本	1.8+	无。

操作步骤

假设有两个服务app-a和app-b，调用关系为app-a->app-b，这两个服务都已接入微服务治理中心。此时app-b有灰度应用需要发布，需要实现在app-a调用app-b的过程中，将灰度流量引入到app-b的灰度应用中进行小规模验证。

部署app-b灰度应用

部署app-b灰度应用，需为灰度应用打上灰度标签，主要提供如下两种方式： 为云容器引擎集群应用设置标签，可以为容器添加环境变量MSE_SERVICE_TAG=gray。 为ECS应用设置标签，在应用启动时，添加JVM启动参数-Dctgcloud.service.tag=gray。

查看金丝雀页面

完成灰度应用部署后，即可查看应用当前存在的标签。 1.登录微服务治理中心控制台。 2.在左侧导航栏选择 微服务治理中心 ->应用治理。 3.在应用治理页面单击目标应用卡片。 4.在左侧导航栏选择流量治理 - 金丝雀，即可查看灰度应用标签。

设置灰度规则并引入流量

金丝雀提供两种灰度规则，分别是按比例路由和按内容路由，按比例路由指的是将指定比例的流量路由到灰度应用，按内容路由是指针对请求头、请求参数或请求体中的内容做匹配，将满足匹配规则的流量路由到灰度应用。

按比例路由

1.登录微服务治理中心控制台。 2.在左侧导航栏选择 微服务治理中心 ->应用治理。 3.在应用治理页面单击目标应用卡片。 4.在左侧导航栏选择流量治理 - 金丝雀，点击引入流量。 5.设置灰度应用的流量比例。

按内容路由

1.登录微服务治理中心控制台。 2.在左侧导航栏选择 微服务治理中心 ->应用治理。 3.在应用治理页面单击目标应用卡片。 4.在左侧导航栏选择流量治理 - 金丝雀，点击引入流量。 5.设置灰度应用的内容规则。


引入流量
按比例灰度
按内容灰度
流量规则
* 框架类型
SpringCloud
* Path
/**
* 条件模式
同时满足下列条件
满足下列任一条件
条件列表
参数类型
请输入Header字段名
值
操作
Header
tag
=
gray
+ 添加新的规则条件
+ 添加新的入口流量规则
是否链路传递
取消
确定

按内容灰度参数说明：

参数	说明
框架类型	Spring Cloud/Dubbo。
Path	Spring Cloud为接口路径，Dubbo为接口。
条件模式	满足一个条件，或满足所有条件。
条件列表	可以设置Header、Cookie、Parameter和Body Content四种参数类型。
是否链路传递	代表开启全链路流控。

完成发布

完成小规模流量验证后，点击发布完成，未打标版本的流量比例会被调整为100%，此时所有的流量都会转发到未打标应用中。 若发布失败，点击回滚，配置的规则会被清楚，此时所有的流量都会转发到未打标应用中。

标签路由

概述

标签路由是将每个服务打上一个标签，通过标签将标签相同的服务分为同一个分组，然后约束流量在同一个分组内流转，以此实现灰度发布、金丝雀发布、蓝绿发布等功能。

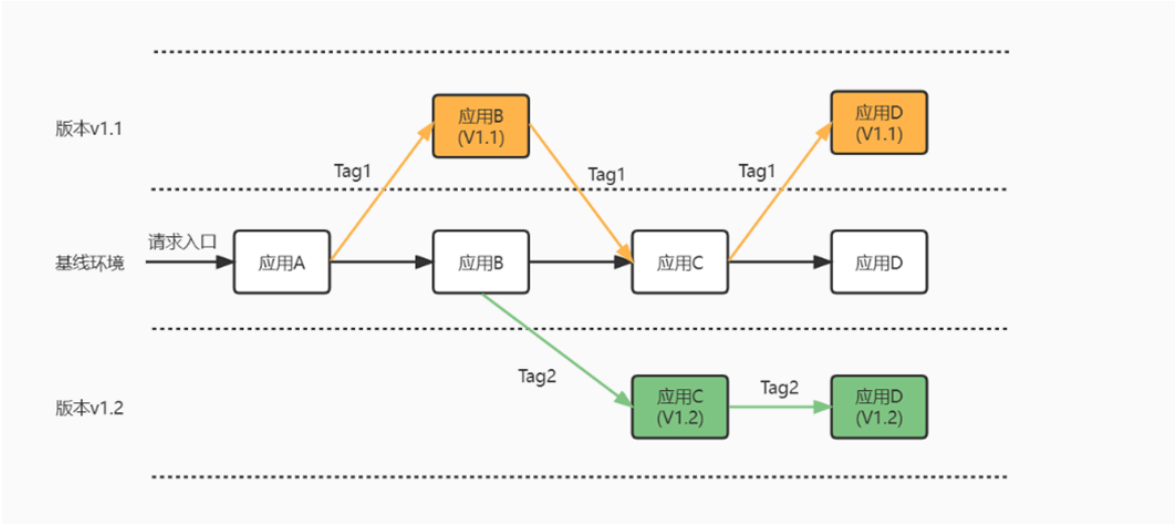
版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Edgware及以上版本	客户端：Feign、RestTemplate负载均衡：Ribbon、LoadBalancer。
Dubbo	2.5.3~2.7.8	支持Alibaba Dubbo、Apache Dubbo。
注册中心	Nacos、Eureka、Zookeeper	--
jdk版本	1.8+	--

应用场景

A/B测试

在营销活动中，经常会有A/B测试，A/B测试是为了测试不同的规则对业务的影响，所以一个应用会有多个版本同时运行，可以通过标签的方式区分不同的版本，对不同版本的应用进行流量隔离，将特殊用户的流量路由到特殊版本，从而实现A/B测试。



开通标签路由

步骤1：为应用设置标签

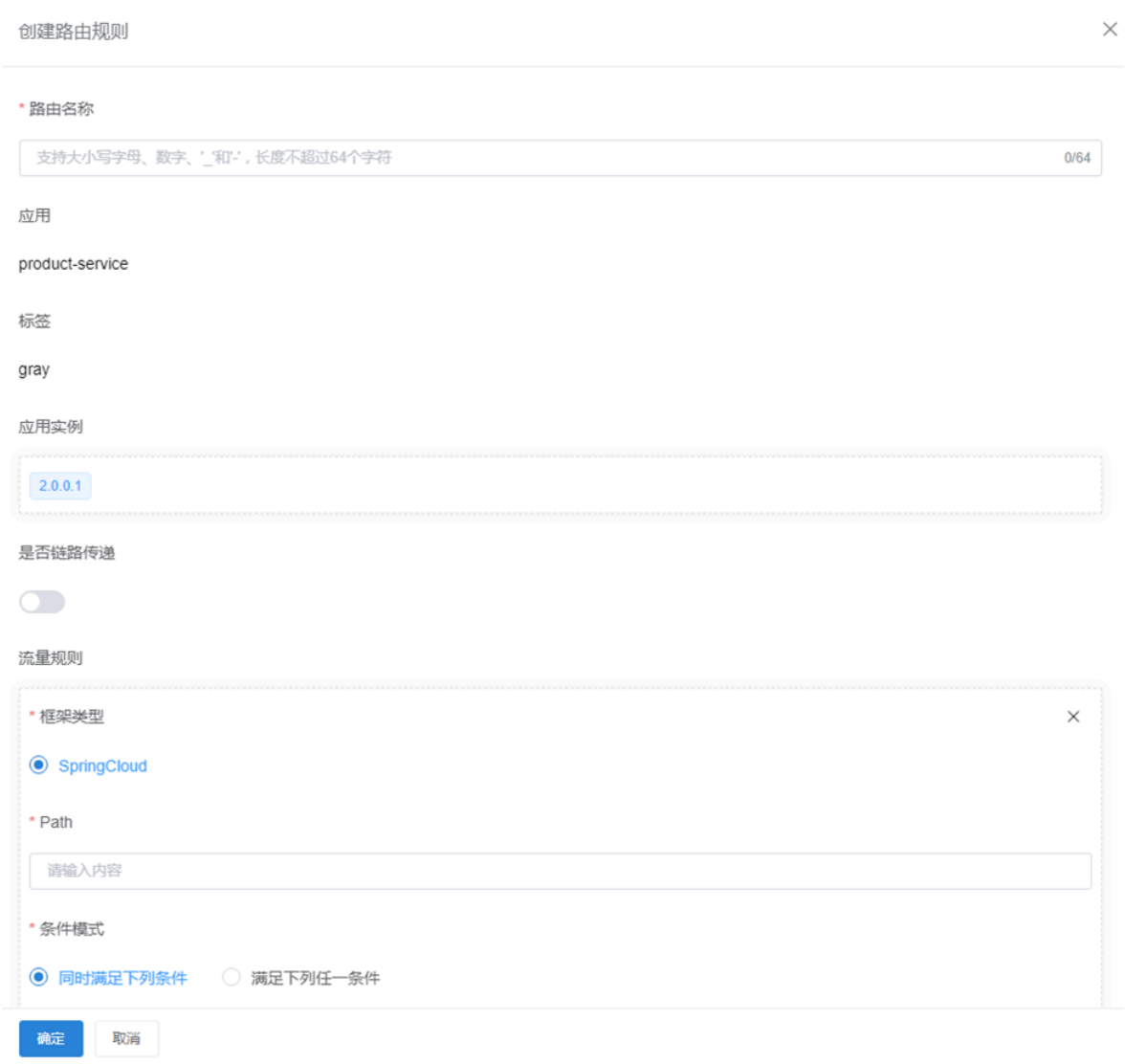
为云容器引擎集群应用设置标签，应用容器添加环境变量MSE_SERVICE_TAG=gray。

为ECS应用设置标签，在启动应用时，添加JVM启动参数-Dctgcloud.service.tag=gray。

步骤2：在微服务治理中心控制台创建标签路由

1. 登录微服务治理中心控制台。
2. 在左侧导航栏选择 微服务治理中心 ->应用治理。
3. 在应用治理页面单击目标应用卡片。
4. 在应用页面左侧导航栏选择流量治理 - 标签路由，查看服务标签。

5. 在标签路由页点击流量分配，为标签按比例分配流量。
6. 在标签路由页的流量规则栏新增流量规则。



流量规则参数说明：

参数	说明
路由名称	路由规则的名称。
应用	所选的应用。
标签	设置的标签名。
应用实例	设置了该标签的实例ip。
是否链路传递	代表开启全链路流控。
框架类型	Spring Cloud Dubbo。
Path	SpringCloud为path路径，Dubbo为接口。
条件模式	满足一个条件，或者满足所有条件。

参数	说明
条件列表	可以设置Header、Cookie、Parameter和Body Content四种参数类型。
是否开启流量规则	流量规则开关。

无损上线

概述

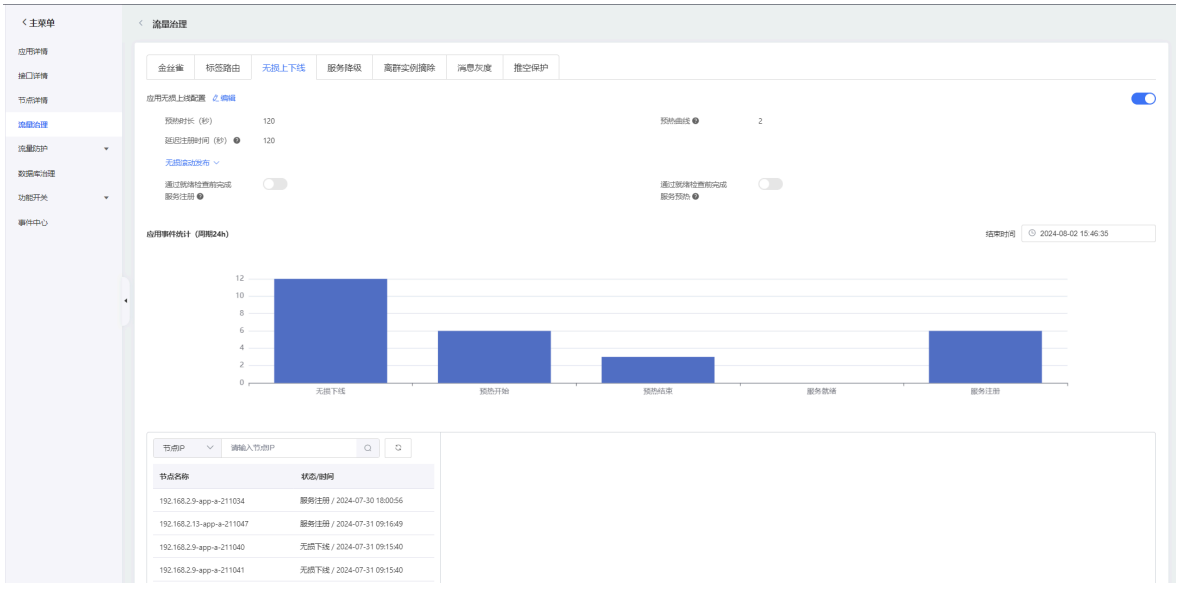
在应用的发布、重启过程中，不可避免的会造成流量的损失，可能会因为服务没有预热而出现抖动，或因为服务没有完全初始化之前就注册在注册中心导致访问失败。微服务治理提供了服务上线的保护能力，提供服务预热、延迟注册服务的能力解决流量损失问题。

版本限制

框架	限制	详情
Spring Cloud	注册中心： Nacos、Eureka、Zookeeper	若用户修改了ZoneAwareLoadBalancer则不支持预热。
Dubbo	2.7相关版本	Dubbo 2.6和3.0不支持预热。
Jdk版本	1.8+--	--

开通无损上线

1. 登录微服务治理中心控制台。
2. 在左侧导航栏选择 微服务治理中心 ->应用治理。
3. 在应用治理页面单击目标应用卡片。
4. 在应用页面左侧导航栏选择流量治理 - 无损上下线，配置无损上线规则。



服务预热

应用在启动后，由于内部资源还未彻底初始化，直接处理大流量可能会导致出现请求阻塞或报错的情况。在应用刚启动的一段时间内，通过小流量预热的方式使应用完成内部资源的初始化，这样可以有效的避免应用上线后出现的抖动问题。

预热参数说明：

参数	说明
预热时长（秒）	应用实例下一次启动的预热时间，默认预热时长为120秒。服务预热时长设置范围为0~86400秒（即24小时）。
预热曲线	默认为2（适合于一般预热场景），表示在预热周期内服务提供者的流量接收曲线形状呈2次曲线形状。预热曲线设置范围为0~20。

延迟注册

应用在启动时，会存在内部资源还未初始化完成，服务就被暴露到注册中心，被外部的消费者调用，这种情况可能会导致调用报错。通过设置延迟注册，可以在应用的内部资源初始化完成之后再将服务注册到注册中心，供外部消费者调用。

参数	说明
延迟注册时间（秒）	延迟注册时间， 时长设置范围为0~86400秒（即24小时）。

无损下线

概述

在微服务场景中，当服务提供者下线时，若消费者无法及时感知到提供者的下线状态，则可能会出现异常请求。通过配置无损下线来保证应用在下线、重启时流量零损耗。

版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Dalston及以上版本	支持负载均衡组件为ribbon的应用 不支持负载均衡组件为Spring Cloud LoadBalancer的应用。
Dubbo	2.5.3+--	--
Jdk版本	1.8+	--

离群摘除

概述

在微服务场景中，当服务提供者的实例出现异常时，服务消费者无法感知到提供者出现异常，此时就可能出现异常调用。通过配置离群摘除功能可以实时监测下游实例的可用性，摘除异常实例，提升业务的可用性。

版本限制

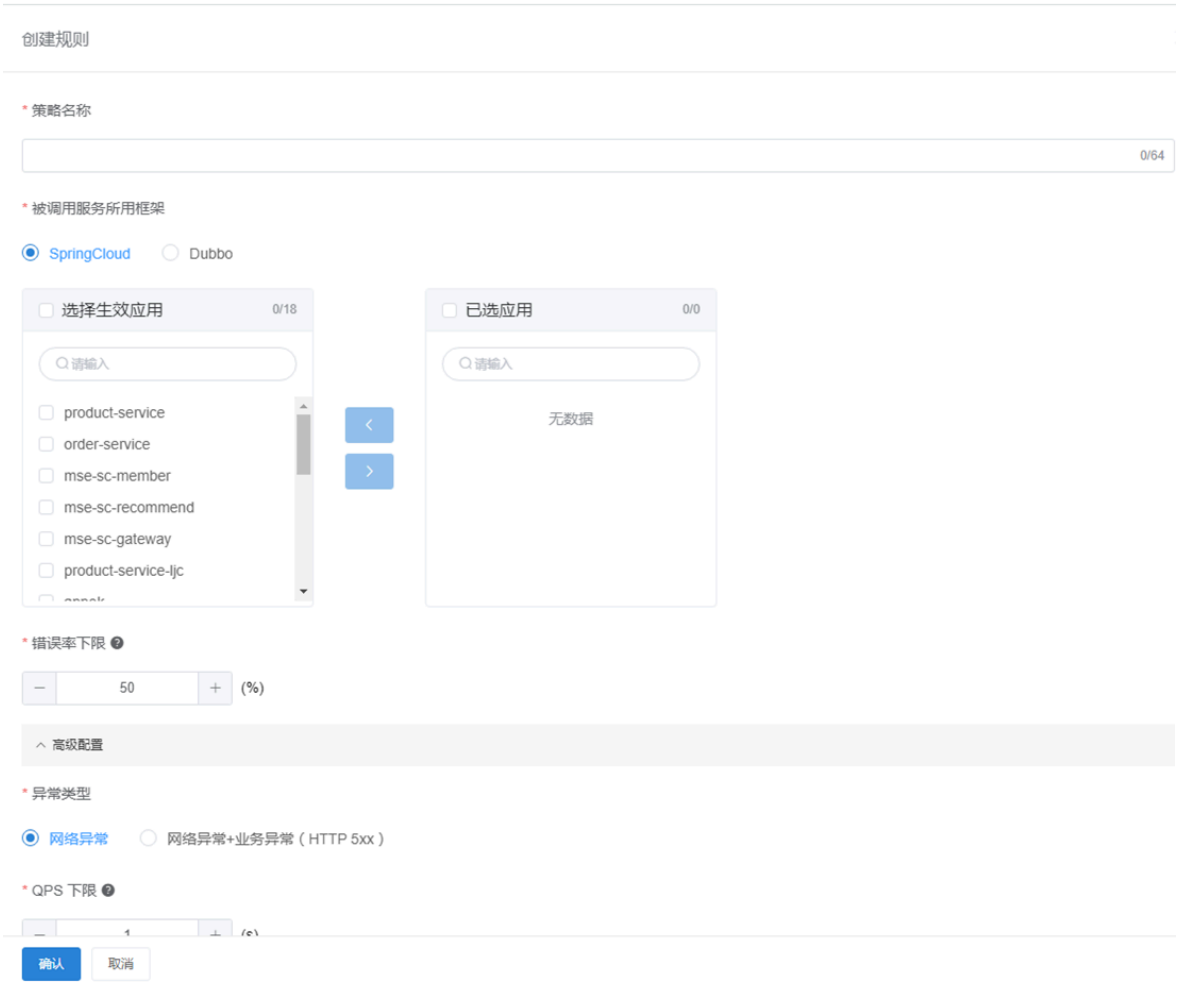
框架	限制	详情
Spring Cloud	Spring Cloud Dalston及以上版本	--

框架	限制	详情
Dubbo	2.5.3+	支持Apache Dubbo 不支持Alibaba Dubbo
Jdk版本	1.8+	--

开通离群摘除

1. 登录微服务治理中心控制台。
2. 在左侧导航栏选择 微服务治理中心 ->应用治理。
3. 在应用治理页面单击目标应用卡片。
4. 在应用页面左侧导航栏选择流量治理 - 离群实例摘除，可查看离群实例摘除规则列表。

4. 在创建离群实例摘除页面配置相关参数，并单击确认。



离群摘除规则参数说明：

参数	说明
策略名称	离群摘除规则的名称。
被调用服务所用框架	Spring Cloud或Dubbo。
选择生效应用	选择生效应用后，该应用调用的异常应用实例会被摘除。
错误率下限	被调用的应用中某个应用实例的错误率高于设置的域值后，将摘除该实例。默认值为50%。例如该实例在统计时间窗口内被调用10次，有6次调用失败，错误率为60%，超过了配置的错误率域值（50%），则从应用中移除该实例。
异常类型	目前只支持网络异常+业务异常（HTTP 5xx）。
QPS下限	QPS按照统计时间窗口进行计算，默认为10秒。
摘除实例比例上限	摘除的异常实例比例上限，即达到阈值后，不再摘除异常实例。

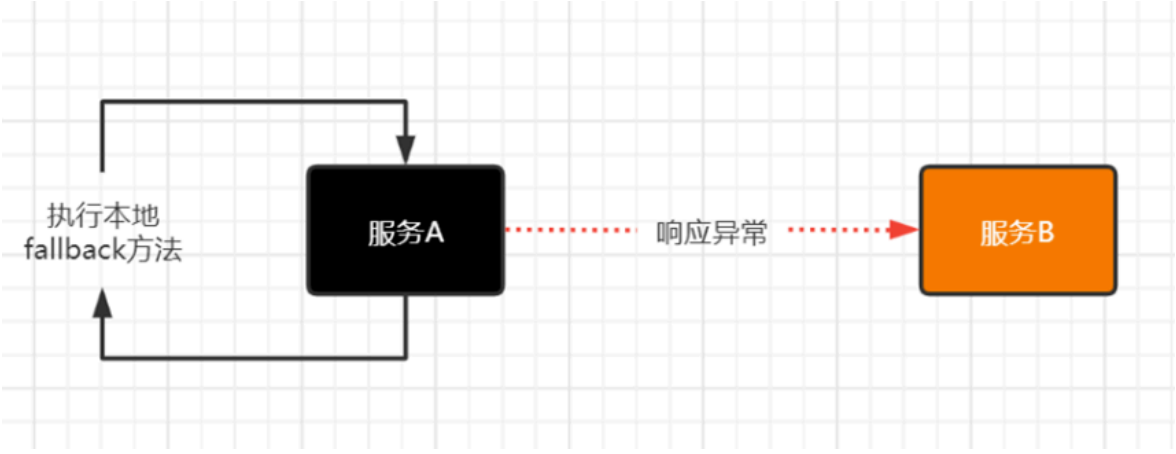
参数	说明
恢复检测单位时间	摘除的异常实例比例上限，即达到阈值后，不再摘除异常实例。
未恢复累计次数上限	持续对异常实例进行检测，检测间隔随检测次数按恢复检测单位时间线性增加，当达到设置的检测次数上限后，会按最长时间间隔持续检测异常实例是否恢复。
默认状态	默认是否开启离群摘除规则。

服务降级

概述

在微服务场景中，当下游服务出现异常，或下游服务返回的不是预期内的结果时，这时会对上游业务造成影响。通过配置服务降级功能，可以对下游服务进行降级处理，返回预期内的结果。

服务降级是指在下游服务出现不可用或响应过慢时。上游服务主动调用本地的降级逻辑，迅速返回给用户。降级逻辑中可以返回异常码，也可以返回一个固定的数据。熔断可以理解为降级中的一部分。



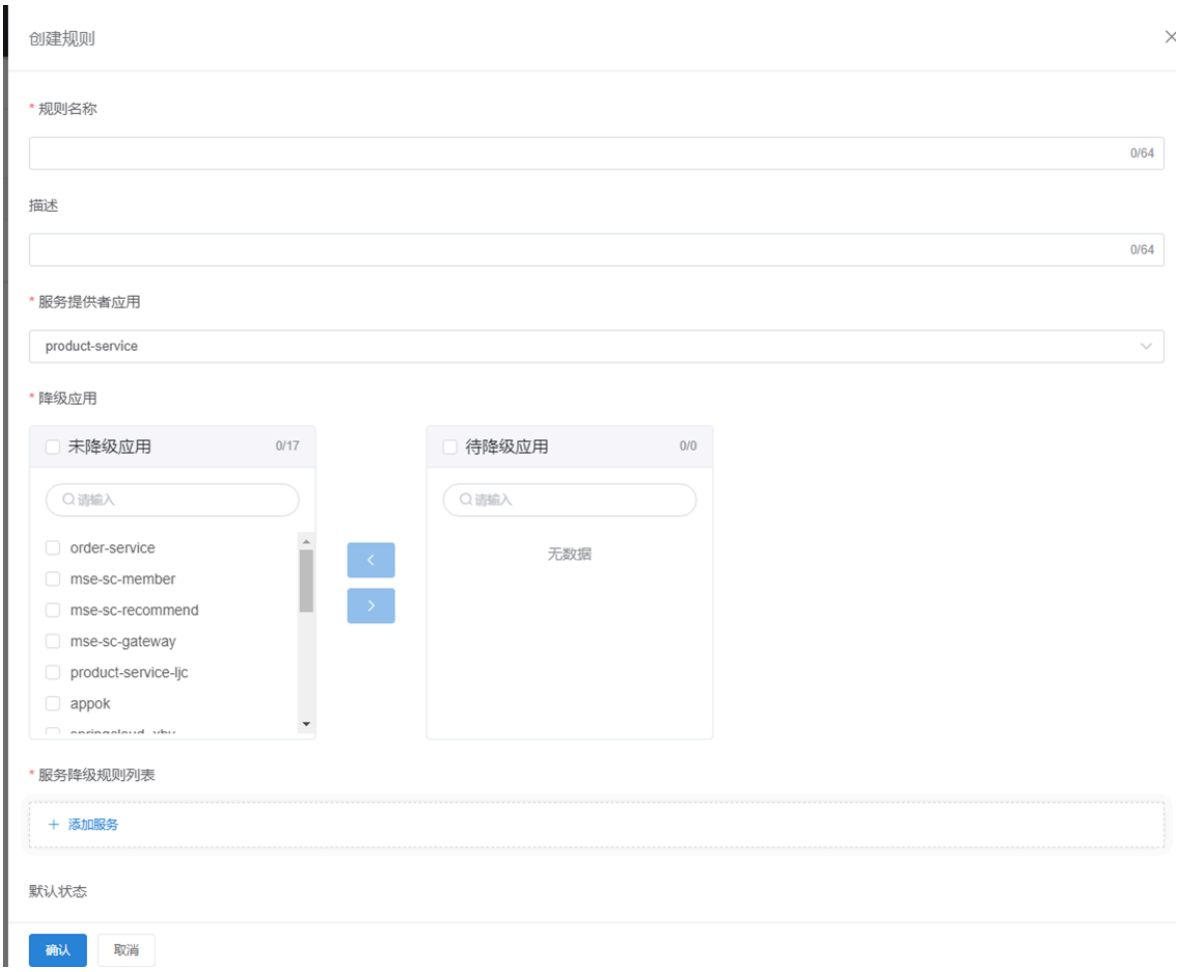
版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Dalston及以上版本	客户端：Feign、RestTemplate
Dubbo	2.5.3+	--
jdk版本	1.8+	--

开通服务降级

1. 登录微服务治理中心控制台。
2. 在左侧导航栏选择 微服务治理中心 ->应用治理。
3. 在应用治理页面单击目标应用卡片。
4. 在应用页面左侧导航栏选择流量治理 - 服务降级，可查看降级规则列表。

5. 在创建降级规则页面配置相关参数，并单击保存。



服务降级规则参数说明：

参数	说明
规则名称	服务降级规则的名称。
描述	规则的详情描述。
服务提供者应用	服务提供者，被降级的应用。
降级应用	选择应用为待降级应用。

参数	说明
服务降级规则列表	--
框架类型	SpringCloud和Dubbo。
服务路径	服务的接口。
请求方法	GET/POST。
执行策略	所有请求生效/异常请求生效。
降级策略	降级策略分为四种，分别是返回Null值、返回Exception异常、返回自定义Json数据、自定义回调。
默认状态	默认打开或关闭规则。

错误注入

概述

错误注入功能提供模拟微服务间异常调用的能力，可应用于故障演练、能力验证等场景。该能力支持调用延时和调用异常响应的模拟，通过该能力可进行应用容错能力研发、测试和演示，提高开发测试人员工作效率。

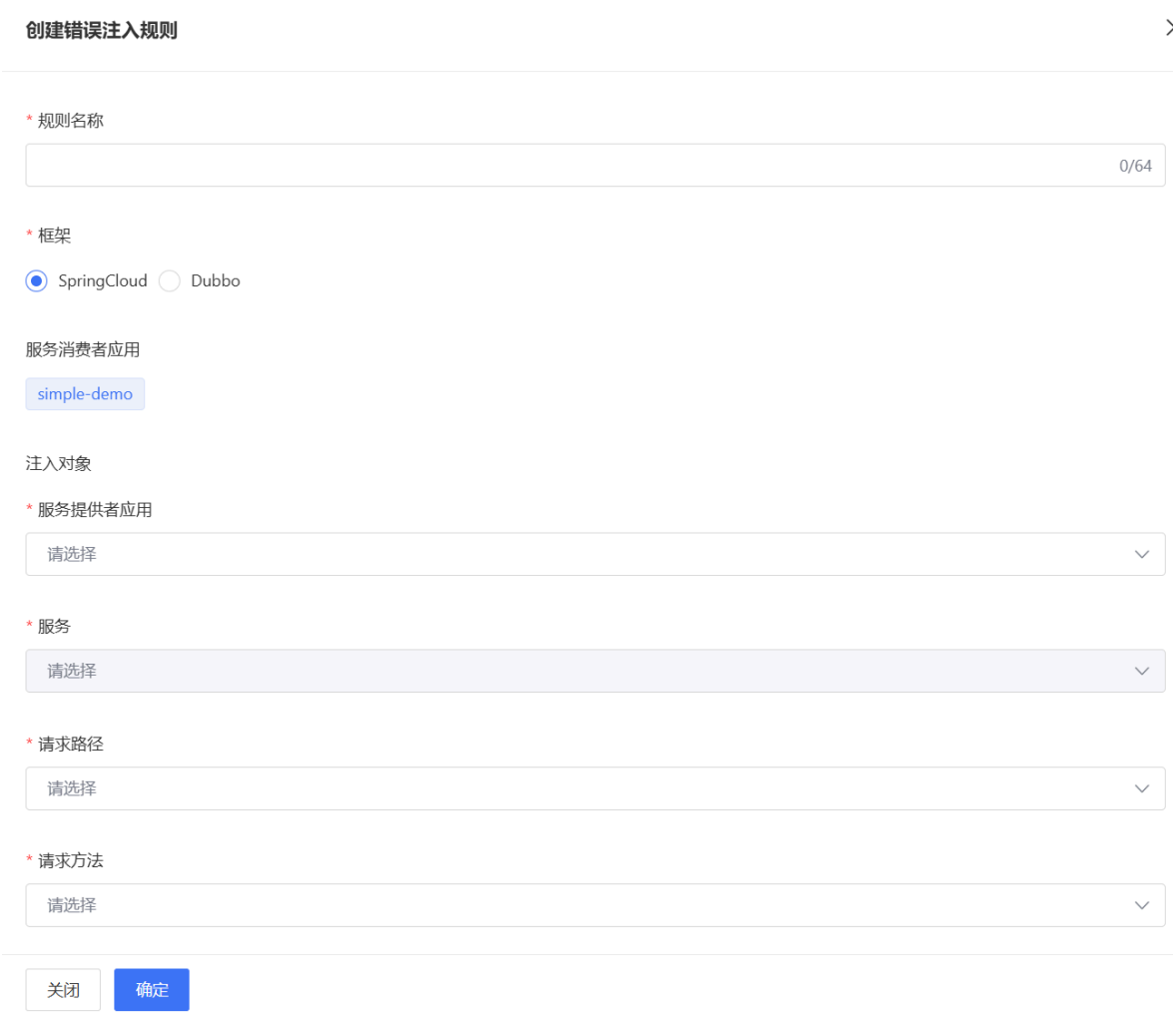
版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Dalston及以上版本	客户端：Feign、RestTemplate
Dubbo	2.5.3+	无
jdk版本	1.8+	无

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用详情页后，点击流量治理，选择错误注入页面

5. 点击创建错误注入规则按钮，完成错误注入规则创建。



错误注入规则参数说明：

参数	说明
规则名称	错误注入规则的名称
框架	支持SpringCloud和Dubbo框架
服务消费者应用	服务消费者对应的应用名称
服务提供者应用	服务提供者对应的应用名称
服务	被调用服务标识SpringCloud框架：服务注册名称Dubbo框架：服务名称/分组/版本信息，格式为{DubboService}:{group}:{version}， 如 com.example.DemoService:dev:v1
请求路径	SpringCloud框架被调用接口的路径

参数	说明
请求方法	SpringCloud框架Http请求方法支持GET/POST/PUT/DELETE选择ALL匹配所有方法
服务方法	Dubbo框架被调用的方法，格式为{MethodName}:{ParamType...}如 sayHello:java.lang.String,java.lang.String
类型	延迟：模拟调用延迟，可自定义延迟时间错误：SpringCloud框架下模拟调用返回的Http状态码和响应体；Dubbo框架下模拟调用触发的异常，可输入异常类全路径和异常信息
触发概率	触发此规则相关模拟调用的概率

消息灰度

概述

在使用全链路灰度能力时，若涉及消息的灰度，可以开启消息灰度能力。

版本限制

框架	限制	详情
RocketMQ	4.5.0+	当前只支持RocketMQ的灰度。
jdk版本	1.8+	无。

开启消息灰度

1.登录微服务治理中心控制台。 2.在左侧导航栏选择 微服务治理中心 ->应用治理。 3.在应用治理页面单击目标应用卡片。 4.在左侧导航栏选择流量治理 - 消息灰度，即可配置消息灰度规则。

消息灰度参数说明：

参数	说明
未打标环境忽略标签	未打标环境默认会消费所有环境的消息。若配置“未打标环境忽略标签”，则未打标环境会忽略配置的标签。
开启消息灰度开关	消息灰度生效开关。

参数	说明
消息灰度过滤侧	客户端过滤：在消费端MSE Agent过滤，会拉取所有的消息，建议提前评估消息量。服务端过滤：在服务端通过SQL92方式过滤，需要RocketMQ服务端支持。

推空保护

概述

推空保护功能用于处理客户端在请求注册中心订阅服务端地址列表时，在服务端注册异常的场景下，注册中心返回了空列表，此时客户端忽略该空返回的变更，从缓存中获取上一次正常的服务端地址进行服务访问。能够在注册中心在进行变更（变配、升降级）或遇到突发情况（例如，可用区断网断电）或其他不可预知情况下的列表订阅异常收到空的地址列表推送时，可以有效保护业务调用，增加业务可靠性。

版本限制

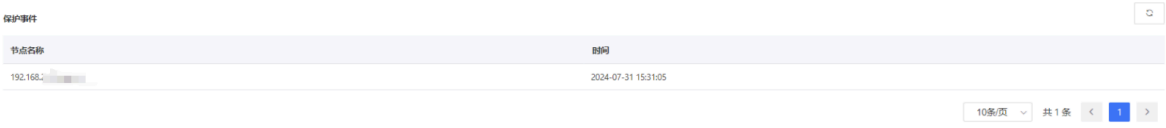
框架	限制	详情
Spring Cloud	Spring Cloud Edgware及以上版本	客户端：Feign、RestTemplate；负载均衡：Ribbon、LoadBalancer。
Dubbo	2.5.3+	支持Alibaba Dubbo、Apache Dubbo。
注册中心	Nacos、Eureka、Zookeeper	无。
jdk版本	1.8+	无。

开启推空保护

1.登录微服务治理中心控制台。 2.在左侧导航栏选择 微服务治理中心 ->应用治理。 3.在应用治理页面单击目标应用卡片。 4.在左侧导航栏选择流量治理 - 推空保护，即可开启推空保护。

查看推空保护事件

若发生推空保护，在推空保护页面即可查看推空保护事件。



流量防护

概述

流量防护以流量为切入点，从流量控制、熔断降级、系统负载保护等多个维度来保障业务的稳定性，提供更专业稳定的流量防护手段、秒级的流量水位分布分析功能。广泛用于秒杀场景、消息削峰填谷、集群流量控制、实时熔断等场景中。

应用治理是一种多维度的流量保障工具，能够有效避免业务中的流量问题，提供秒级的流量监控和分析功能，有效保障系统的稳定性和可靠性。其中，流量控制、熔断降级、系统负载保护等多个功能模块相互配合，可以有效应对各种流量挑战，为业务的稳定发展提供有力支撑。该系统广泛应用于秒杀场景、消息削峰填谷、集群流量控制、实时熔断等多个场景中，可以有效保障业务的安全性和稳定性。

配置流控规则

配置流控规则的原理是监控应用或服务流量的QPS指标，当指标达到设定的阈值时立即拦截流量，避免应用被瞬时的流量高峰冲垮，从而保障应用高可用性。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择流量防护 - 规则管理，在流控规则页签单击新增流控规则，然后在弹出的对话框中配置规则信息，参数说明如下。
5. 设置完成单击新增。

页面参数说明如下：

参数	描述
接口名称	待流控的资源名称。
来源应用	该规则针对的来源应用，默认来源应用设为default，代表不区分来源应用。
统计维度	选择资源调用关系进行流控。• 当前接口 ：直接控制来自来源应用中调用来源的访问流量，如果来源应用为default则不区分调用来源。通常应用于流量匀速通过的场景。• 关联接口 ：控制当前资源的关联资源的流量。通常应用于资源争抢时，留足资源给优先级高接口的场景。• 链路入口 ：控制该资源所在的调用链路的入口流量。选择链路入口后需要继续配置 入口资源 ，即该调用链路入口的上下文名称。通常应用于预热启动避免大流量冲击的场景。
单机QPS阈值	触发对流控接口的统计维度对象的QPS阈值。

参数	描述
流控效果	选择流控方式来处理被拦截的流量。• 快速失败 ：达到阈值时，立即拦截请求。按照应用系统设置中的适配模块配置信息，进行内容返回。• 预热启动 ：需设置具体的预热时间。如果系统在此之前长期处于空闲的状态，当流量突然增大的时候，该方式会让处理请求的速率缓慢增加，经过设置的预热时间以后，到达系统处理请求速率的设定值。默认会从设置的QPS阈值的1/3开始慢慢往上增加至设置的QPS值，多余请求会按照快速失败处理。• 排队等待 ：请求匀速通过，允许排队等待，通常用于请求调用削峰填谷等场景。需设置具体的超时时间，达到超时时间后请求会快速失败。
是否开启	打开开关表示启用该规则，关闭开关表示禁用该规则。开关修改之后会立即生效。

使用场景

常用场景 1 ：当资源争抢时，需留足资源给优先级高的接口

为了确保提交的数据不丢失，我们将数据库读写分离，并给写库分配了更高的优先级。对于读取数据的请求，如果过于频繁，会对写入操作进行限流。

在新建流控防护规则对话框中配置以下规则信息：

1. 统计维度选择关联接口。
2. 流控效果选择快速失败。
3. 关联接口阈值为10。

当写库操作的QPS超过10之后，读库操作会被限流以保证留足资源给写库操作，避免写库操作数据丢失。

常用场景 2 ：预热启动避免大流量冲击

采用流量控制的方法，在流量入口处进行控制，以缓慢增加的方式让流量通过，在一定时间内达到阈值上限，以便系统能够预热。这种方法最适合应对突发流量的场景。

在新建流控防护规则对话框中配置以下规则信息：

1. 统计维度选择链路入口。
2. 流控效果选择预热启动。
3. 单机QPS阈值为90。
4. 预热时间为10s。

预热流控方式下，默认会从设置的QPS阈值的1/3开始慢慢往上增加至QPS设置值。本示例中，当入口的QPS超过30（即90÷3）时，会在预热的10s内缓慢增长至90。

常用场景 3 ：削峰填谷，使流量匀速通过

请求流量具有波峰波谷的特点，流量管理的原理是将前期的高峰流量延迟到后期再处理，以最大化满足所有请求，并保证用户体验。

在新建流控防护规则对话框中配置以下规则信息：

1. 统计维度选择当前接口。
2. 流控效果选择排队等待。
3. 配置匀速模式下请求单机QPS阈值为5。
4. 等待时长为5s。

配置隔离规则

隔离规则，通过控制接口或调整依赖的并发线程数，以保证系统的稳定性。这种方法适用于应用程序内部或下游依赖出现不稳定的情况，例如慢SQL或下游应用响应时间变慢等。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，新建隔离规则：在左侧导航栏，单击流量防护，在流量防护 - 规则管理 - 隔离规则页，单击新增流控规则按钮。
5. 在新建隔离规则对话框中配置规则信息：
   a. 在选择防护场景页面，修改接口名称，然后单击下一步。
   b. 在配置防护规则页面，配置防护规则，然后单击下一步（说明若需对隔离防护规则进行编辑，则直接进入配置防护规则）。
   c. 在配置防护行为页面，然后单击下一步并单击新增。
   d. 在隔离规则页面，选择对应的规则并在状态栏下单击开启。
   e. 在温馨提示页面，单击确定，开启已配置的防护规则。

使用场景

常用场景 1：保障自身资源充足

当运行该请求的响应时间变长，会导致线程的并发数变大。当并发数超过阈值以后，微服务治理将拒绝多余的请求，直到堆积的任务完成，并发线程数变少。达到将异常隔离，减小不稳定性的效果。例如某个SQL执行时间为20毫秒，预期该请求每秒有50个。

在新增隔离防护规则对话框中配置以下规则信息：

1. 填写接口名称和来源应用。
2. 统计维度选择当前接口。
3. 并发数阈值为50。

常用场景 2： 有一定相关联性的接口

当关联接口被来源应用调用QPS超过阈值时，会对当前接口来源应用的请求进行限流，有一定的相关性的方法来配置规则。例如数据库读操作和数据库写操作这两个资源分别代表数据库读写，数据库写操作接口优先级更高。

为保证读写资源争抢时，数据库写操作的接口可以留足资源，可在新增隔离防护规则对话框中配置以下规则信息：

1. 接口名称为数据库读操作。
2. 统计维度选择关联接口。
3. 关联接口名为数据库写操作。
4. 并发数阈值为10。

常用场景 3： 针对入口链路来配置隔离规则

将入口处的资源进行分离，以确保更高优先级的入口。当调用callstack入口的请求数量超过阈值时，会对来自于来源应用的请求进行隔离流控。这样可以确保重要的请求能够得到及时处理，并且不会影响其他请求的处理。

在新增隔离防护规则对话框中配置以下规则信息：

1. 填写接口名称和来源应用。
2. 统计维度选择链路入口。
3. 并发数阈值设置为10。

配置熔断规则

熔断规则可以监控应用程序内部或下游依赖的响应时间或异常情况，并在达到指定的阈值时立即降低下游依赖的优先级。在指定的时间内，系统将不会调用该不稳定的资源，以确保应用程序的高可用性。当指定时间过后，再重新恢复对该资源的调用。这样可以保证应用程序的稳定性和可靠性。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，新建熔断规则：在左侧导航栏，单击流量防护，在流量防护 - 规则管理 - 熔断规则页，单击新增熔断规则按钮。
5. 在新增熔断规则或新增规则的对话框中配置规则信息。

使用场景

常用场景 1 ：慢调用熔断示例

为了保证当前接口的性能，如果出现第三方服务的响应时间过慢，则会对其进行熔断操作。

常用场景 2 ：异常熔断示例

为了保证更好的用户体验，如果第三方内容展示出现异常，且异常比例较高，则会对其进行熔断操作。

常用场景 3 ：熔断个数

如第三方异常个数达到指定数量，则会对其进行熔断操作。

WEB场景防护

Web场景防护功能是一种用于保护提供Web服务的应用程序的防护机制，可以对访问请求中的参数项进行精细化的流量控制。该功能可以用于处理使用主流Web框架（如Servlet容器、Spring Web、SpringBoot）的应用程序，并配置了API粒度的请求参数解析，以便对请求中的IP地址、主机名、头部信息和URL参数等参数维度进行流量控制。通过这种控制，可以保护应用程序和系统的稳定性。

背景信息

在提供Web服务的场景中，除了API维度的限流降级防护，对于来源IP和访问参数等资源调用的限流防护可以更好地保证业务应用的正常运行。在一些大流量的Web业务场景下，可能需要对多个接口进行限制，以保证系统的稳定性。通过对当前访问频次最高的来源IP或访问频次最高的商品序号进行有针对性的限制，可以有效地保护系统。比如：

- 对于一段时间内最频繁购买的商品序号，可以进行限制以防止击穿缓存导致大量请求到数据库。
- 对于一段时间内频繁大量访问的来源IP，可以进行限制以防止利用虚假信息恶意刷单。这样可以保证系统的稳定性和可靠性。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择流量防护 - WEB场景 - WEB流控，单击新增WEB防护规则。
5. 在新增WEB防护规则对话框，配置规则信息：

- 在选择防护场景页面，修改接口名称，然后单击下一步。
- 在配置防护规则页面，配置防护规则，然后单击下一步。
- 在配置防护行为页面，然后单击下一步并单击新增。
- 在下面规则列表的页面，选择对应的规则并在状态栏下单击开启。

配置主动降级规则

主动降级规则可以指定哪些接口需要进行降级，以及在何种情况下需要进行降级。被降级的接口会自动触发自定义的降级行为，以确保系统的稳定性和可靠性。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，新建隔离规则：在左侧导航栏，单击流量防护，在流量防护 - 规则管理 - 主动降级规则页，单击新增主动降级规则按钮。
5. 在设置主动降级规则的对话框，完成以下配置，然后单击新建。

使用场景

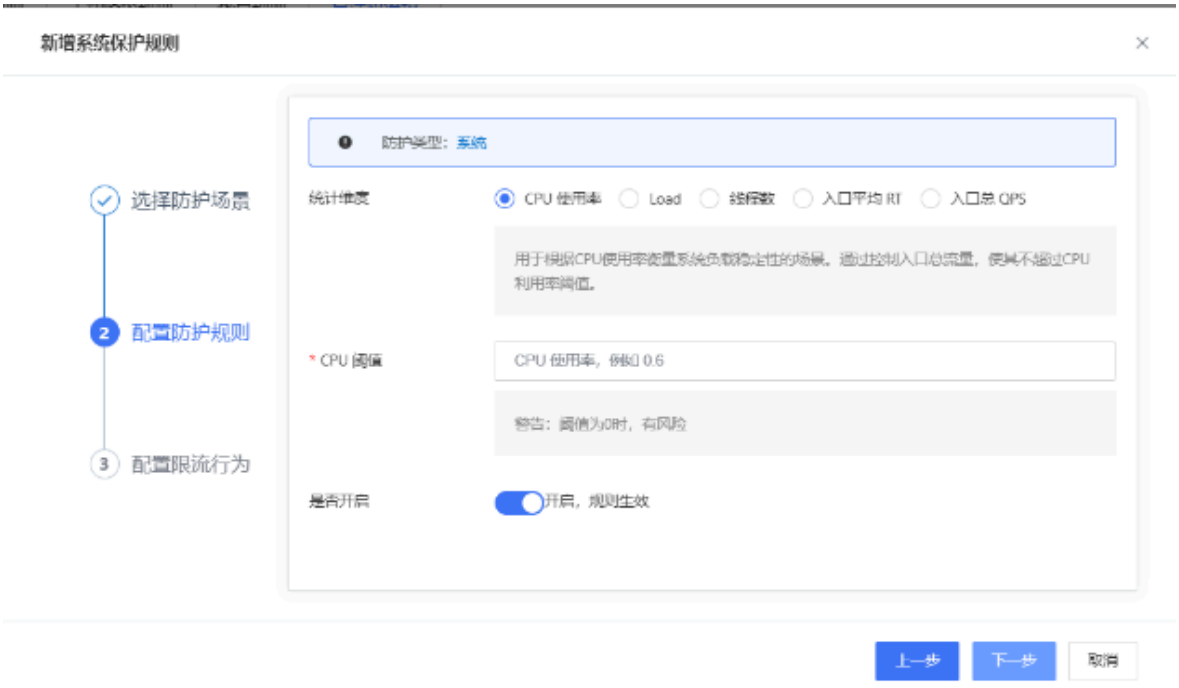
当某个资源发生异常，可以指定对接口进行降级，而不会执行原来的逻辑。

配置自适应流控

系统支持自动化流控或手动设置流控规则，以确保系统的资源分配和负载平衡。自动化流控是根据系统的CPU使用率动态调整应用程序的入口流量，以达到最大吞吐量状态下的系统稳定运行；手动设置流控规则则是从整体角度出发，人工设置规则来控制应用程序的入口流量，以保证系统的负载平衡。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，新建隔离规则：在左侧导航栏，单击流量防护，在流量防护 - 规则管理 - 自适应规则页，单击新增自适应规则按钮。
5. 在页面右上角，关闭自适应流控，在对话框中单击确定关闭。
6. 在自适应流控页签左上角单击新建系统保护规则。
7. 在新建系统保护规则对话框中配置规则信息。



参数说明

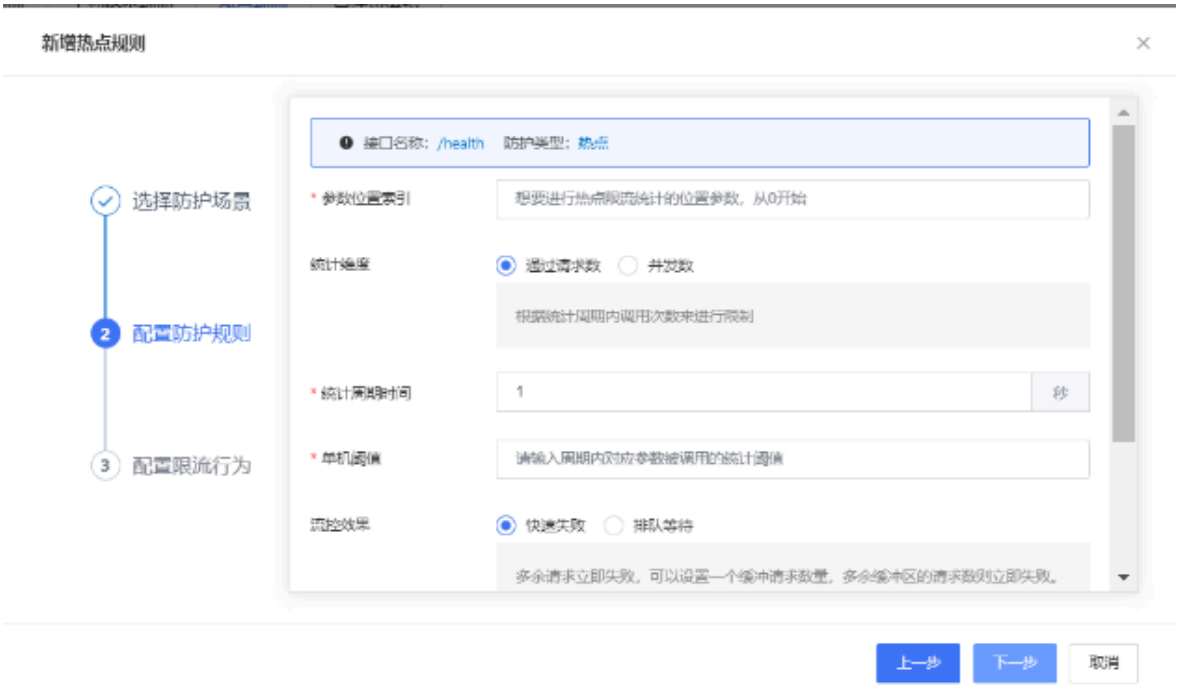
参数	描述	使用场景说明
CPU使用率	当系统CPU使用率超过阈值即触发系统保护，阈值设置范围为0.0~1.0（代表0%~100%）。	适用于设置基础资源水位的场景，比如需要保证一定的冗余水位。但系统水位不宜过高，需要留部分水位。
Load	当系统的Load1超过阈值，且系统当前的并发线程数超过系统容量时才会触发系统保护。系统容量由系统的maxQps * minRt计算得出。	适用于设置基础资源水位的场景，比如需要保证一定的冗余水位。但系统水位不宜过高，需要留部分水位。
线程数	当单台机器上所有入口流量的并发线程数达到阈值即触发系统保护。	适用于设置基础资源水位的场景，比如需要保证一定的冗余水位。但系统水位不宜过高，需要留部分水位。
入口平均RT	当单台机器上所有入口流量的平均RT达到阈值即触发系统保护，单位是毫秒。	适用于衡量入口请求的场景。
入口总QPS	当单台机器上所有入口流量的QPS达到阈值即触发系统保护。	适用于衡量入口请求的场景。

配置热点规则

为应用程序配置热点规则后，微服务治理中心会对系统中的资源调用次数进行分析，并根据配置的热点规则来限制包含热点参数的资源调用，以保证系统的稳定性。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，新建隔离规则：在左侧导航栏，单击流量防护，在流量防护 - 规则管理 - 热点规则页，单击新增热点规则按钮。
5. 在新增热点限流规则对话框中，配置规则信息。
6. 单击新增。



使用场景

常用场景 1：秒杀场景

为了保证系统稳定性，可以配置热点规则，当超过一定量的阈值后，系统会让满足热点规则的请求流量排队等待。例如，对于购买同一商品的请求，如果在1秒内调用次数超过100次，则其他请求将被等待处理。在新建热点规则对话框中，可以配置以下规则信息：

- 阈值：设置超过多少次请求会被限流。
- 等待时间：设置等待时间，单位为秒。
- 调用次数：设置调用次数的阈值。

例如购买同一商品，1s内调用超过100次请求后，则其余请求进行等待。在新建热点规则对话框中配置以下规则信息：

- 填写接口名称。
- 统计维度选择通过请求数。
- 统计周期时间设置为1s，单机阈值设置为100。

- 流控效果选择排队等待。
- 超时时间设置为30 ms。

常用场景 2：调用请求频繁，占用较多系统资源

例如当调用修改请求较多的时候，会占用了写数据库较多资源，则可以对其进行热点快速失败的处理，稍后再修改。在新建热点规则对话框中配置以下规则信息：

- 填写接口名称。
- 统计维度选择并发数。
- 统计周期时间设置为1s，单机阈值设置为100。
- 流控效果选择快速失败。

配置 WEB行为

在Web行为分析中，可以对Web类型的埋点资源进行监控，以便在满足某种规则时触发自定义处理行为。例如，如果某个Web接口触发了流控规则，则可以返回"Blocked by Sentinel"的提示文本。

新增行为

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，在左侧导航栏，单击流量防护，在流量防护 - 防护配置 - 行为管理，单击新增行为，在新增行为对话框中完成以下配置，然后单击新建。

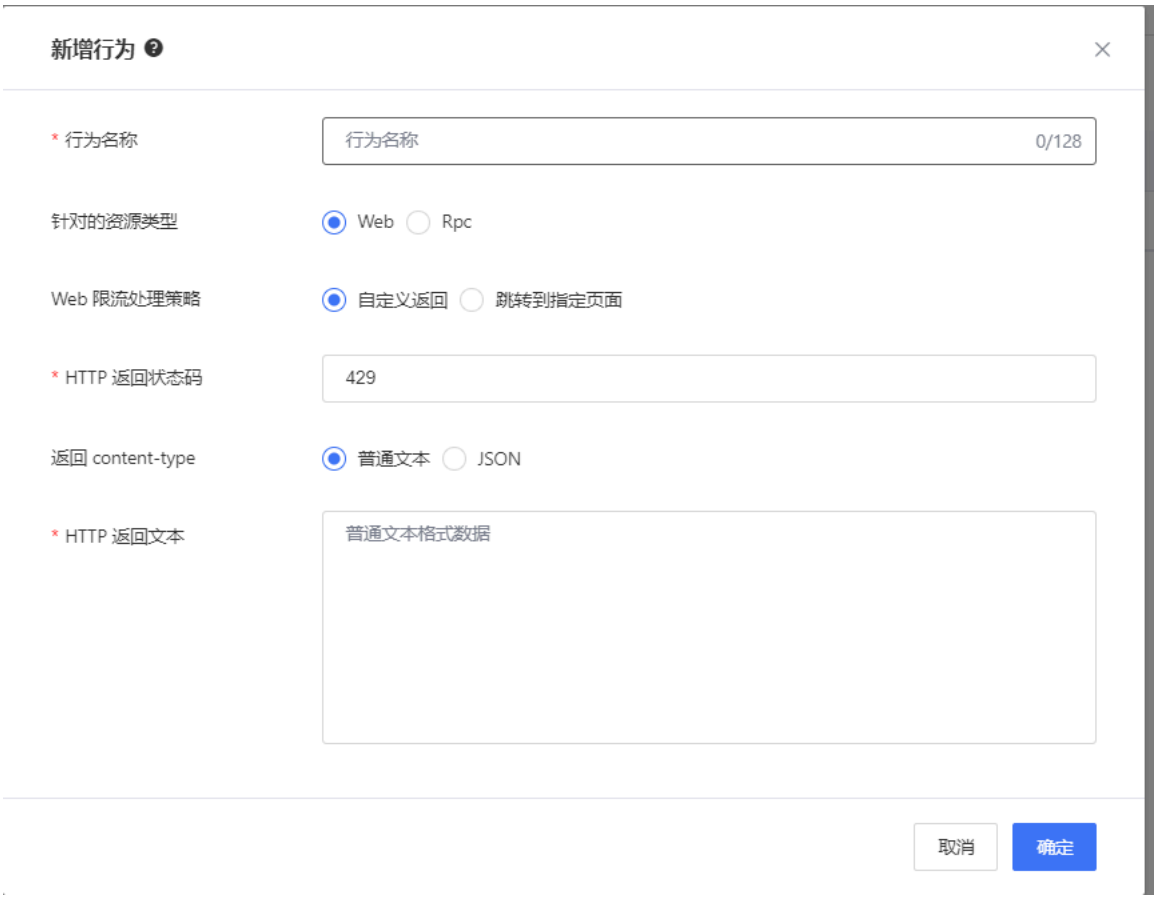
修改或删除行为

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，在左侧导航栏，单击流量防护，在流量防护 - 防护配置 - 行为管理。
5. 行为列表页，您可以查看各个行为的具体描述，修改或删除行为。

关联行为

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择规则管理，在流控规则页签单击新增流控规则，然后在弹出的对话框中配置规则信息。
5. 完成选择防护场景和配置防护规则后，在配置限流行为区域，完成下列设置：

   选择接口类型为Web；在关联行为的下拉列表中选择目标行为进行关联，或单击新增行为来创建新的行为进行关联。

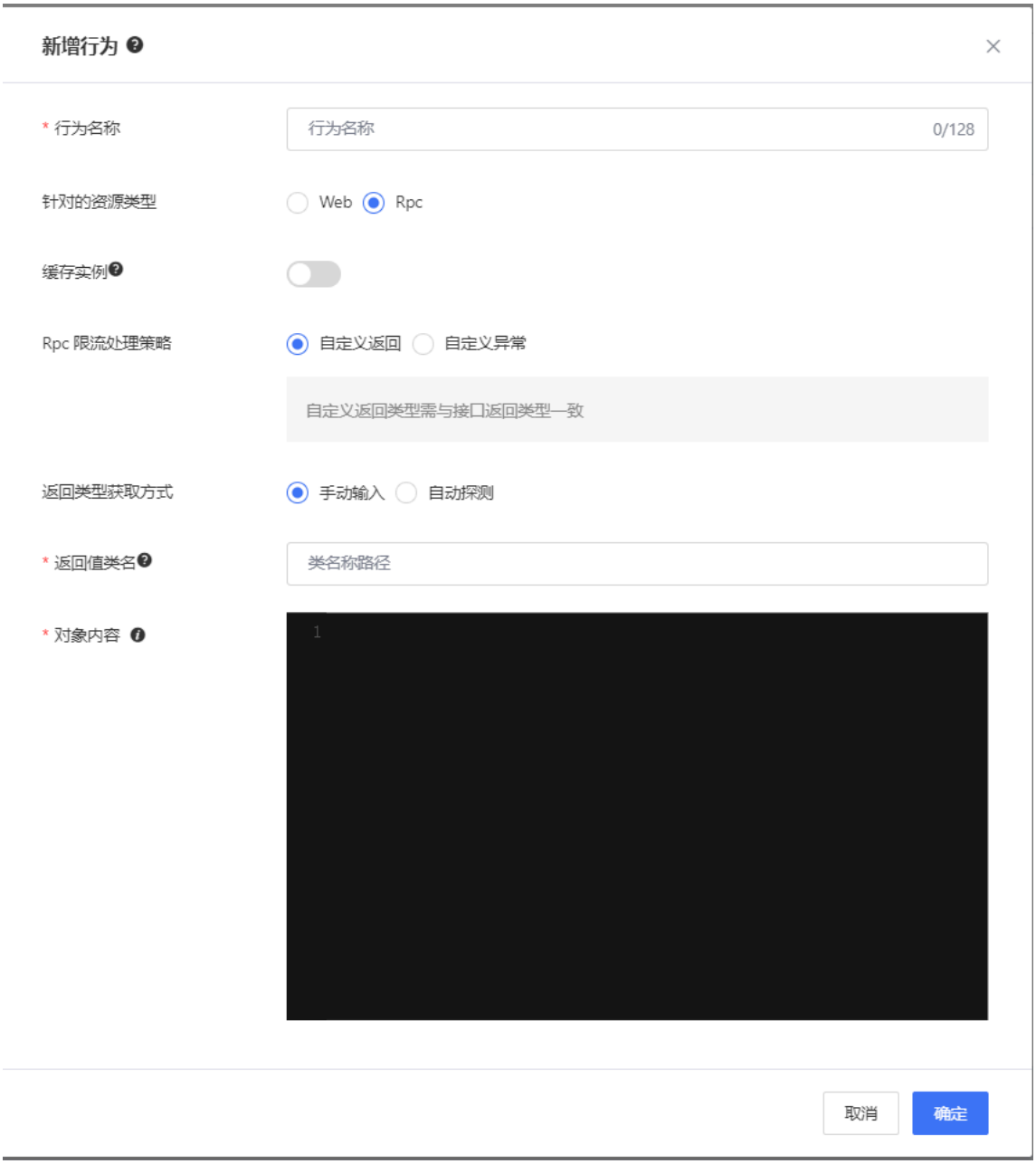


配置 RPC行为

在RPC行为分析中，可以对RPC类型的监控对象进行监控，以便在满足某种规则时返回自定义的处理行为。例如，如果某个RPC接口触发了流控规则，则可以返回一个自定义的接口返回值。这样，可以提高系统的可靠性和稳定性，并且能够更好地满足用户的需求。

新增行为

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，在左侧导航栏，单击流量防护，在流量防护 - 防护配置 - 行为管理，单击新增行为，在新增行为对话框中完成以下配置，然后单击新建。
5. 在新增行为对话框中完成以下配置。
6. 单击新建。新增的行为会显示在应用管理页面的行为管理页签中。



修改或删除行为

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 进入应用之后，在左侧导航栏，单击进入流量防护，在流量防护 - 防护配置 - 行为管理。
5. 在行为列表页，您可以查看各个行为的具体描述，修改或删除行为。

关联行为

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择规则管理，在流控规则页签单击新增流控规则，然后在弹出的对话框中配置规则信息。
5. 完成选择防护场景和配置防护规则后，在配置限流行为区域，完成下列设置：

   选择接口类型为Rpc；在关联行为的下拉列表中选择目标行为进行关联，或单击新增行为来创建新的行为进行关联。
6. 单击下一步后，单击新增。

数据库治理

基本详情

概述

数据库治理针对开发运维人员访问、使用数据库时的常见场景，提供SQL访问审计、数据库稳定性治理、数据流量治理三个方面的能力。MSE提供了SQL监控统计、SQL流量防护、连接池治理、数据库灰度、数据库读写路由等功能，可以从数据库服务维度进行开发提效和性能优化。

基本详情

基本详情页提供应用SQL访问流量和应用资源的实时统计数据。根据数据库相关的应用实时指标，可以及时发现由数据库访问引起的应用性能下降问题，并定位高并发或高RT的SQL以及异常的数据库连接，从而支撑后续SQL语句优化、SQL接口流量防护和连接池配置等系统调优操作。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择数据库治理，在基本详情页签下可以查看应用的SQL流量概览、应用事件、Top SQL洞察和Druid连接池统计数据。

SQL流量概览

可以查看应用在一定时间内SQL流量的通过QPS、限流QPS、异常QPS指标、RT、并发数据。

应用事件

可以查看SQL请求触发的应用防护事件。

Top SQL洞察

可以查看相对高RT的SQL语句及其QPS、RT和并发数据，快速定位SQL对应用业务的具体影响。并且可以通过单击操作下方的SQL防护，管理和配置SQL接口的流量控制和并发隔离规则。

Druid连接池统计数据

可以查看连接池不同数据源下的连接数量、SQL请求执行情况等信息。参数说明请参见连接池治理章节。

SQL详情

概述

SQL详情页提供SQL接口维度的监控能力以及防护配置能力。通过SQL流量监控能力，可以及时定位异常SQL，并针对异常的SQL采用流量防护提供的接口流控和并发隔离能力，避免异常SQL挤占应用的资源。

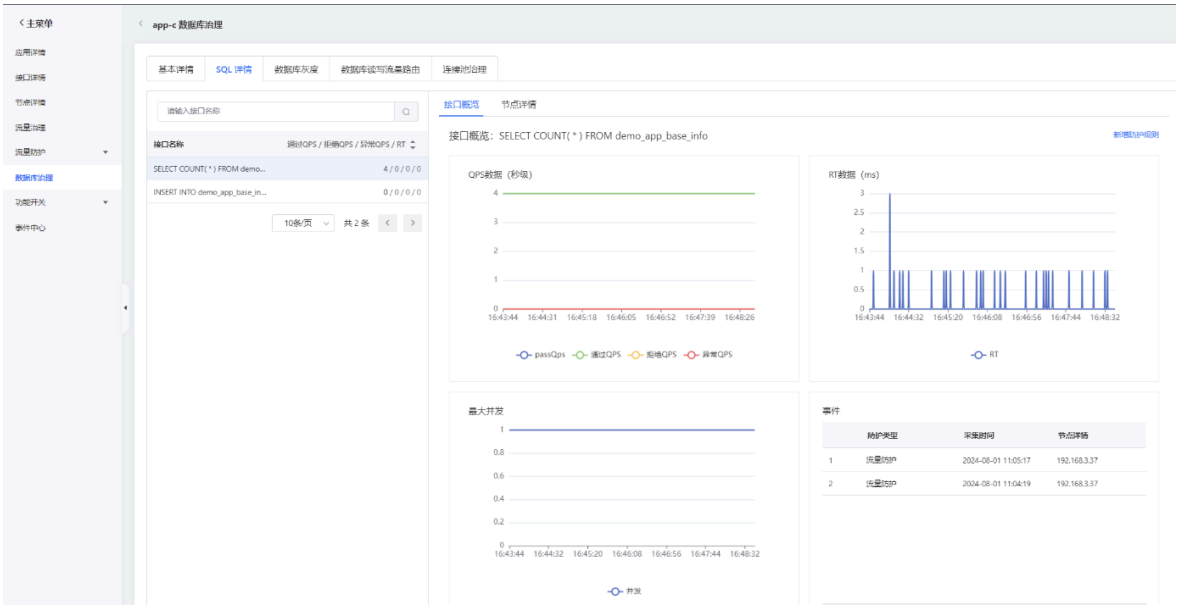
使用限制

JDBC驱动支持的版本如下：

组件	支持驱动版本
MySQL JDBC	5.x/8.x

功能入口

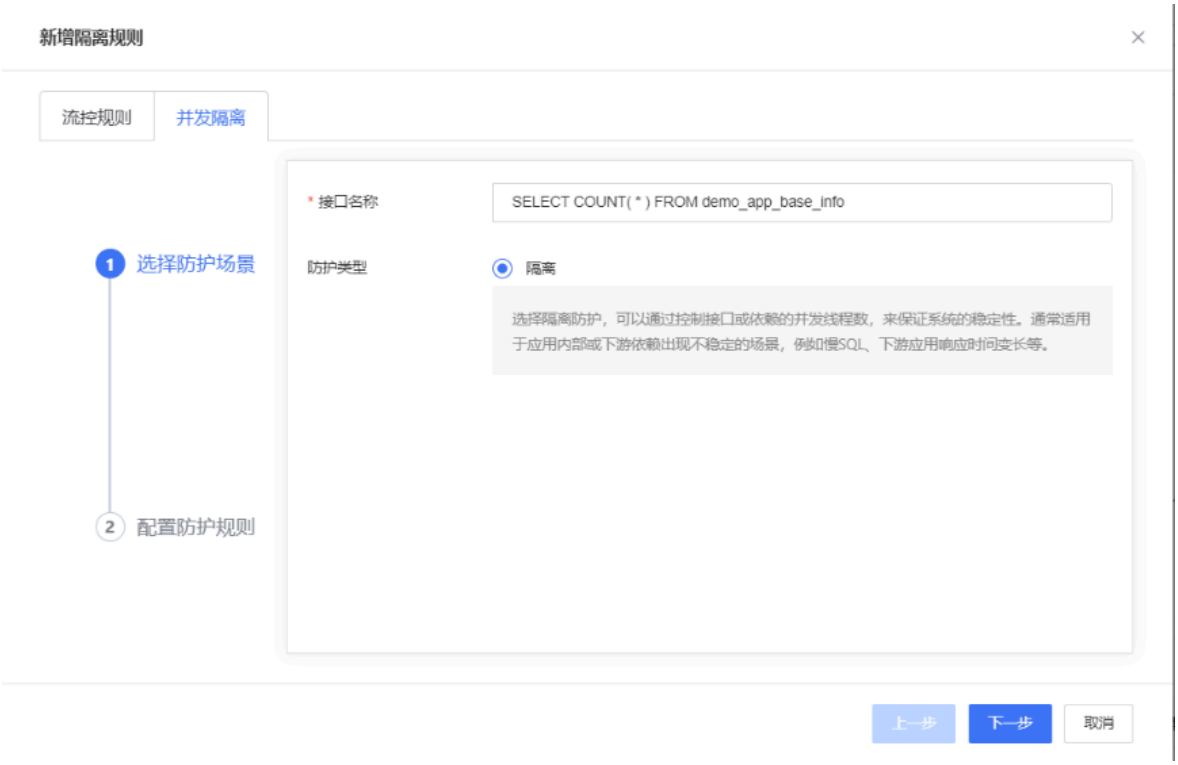
1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择数据库治理，在SQL详情页签下可以查看应用的不同SQL的QPS、RT和并发等详细数据。



5. 单击节点详情可以查看分节点流量。
6. 在接口概览页签，单击新增防护规则，在新增规则对话框内选择接口流控或并发隔离页签进行防护规则配置。

- 接口流控：流控规则会控制流量在阈值之内，拒绝阈值外的请求，可以避免异常SQL请求占用过多服务资源，保证高优先级的请求正常处理。详情请参考流量防护-配置流控规则。

- 并发隔离：隔离规则会控制流量保持在稳态，在请求堆积的情况下保证已有请求的可用资源。详情请参考流量防护-配置隔离规则。



数据库灰度

概述

数据库灰度在应用层提供灰度环境数据隔离能力。其原理是将数据层的流量路由到数据库影子表中，从而与原有数据进行隔离。结合全链路灰度，可以实现网关到后端服务再到数据库的全链路环境隔离，方便进行灰度环境的服务验证。

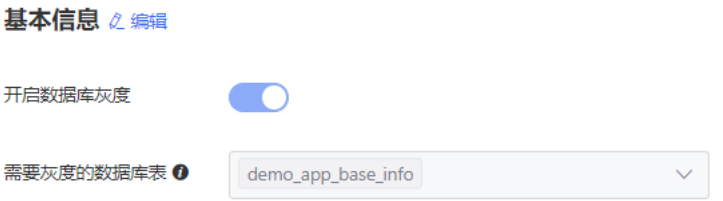
使用限制

JDBC驱动支持的版本如下：

框架	限制	详情
Druid	Spring Cloud Dalston及以上版本	--
HikariCP	2.3.13或以上版本	--
jdk版本	1.8+	--

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择数据库治理，在数据库灰度页签下可以进行数据库灰度的配置。



- **步骤一**

开启前检查灰度影子表，需要您先开启全链路灰度，针对需要灰度的数据库表进行对应灰度表的创建（灰度表创建规则为默认环境的表名后面加上“_标签名”，例如gray环境，默认环境表名为mse_table，那么灰度表名为mse_table_gray）。

如果您只需要针对某些表进行灰度操作，那么您需要配置“需要灰度的数据库表”列表，填入您需要进行灰度访问的数据库表名集合，同时确保您需要灰度的表都有提前创建灰度表。

- **步骤二**

开启数据库灰度。

- **步骤三**

开始验证灰度版本，灰度环境的SQL流量会自动操作对应环境的灰度表。

数据库读写路由

概述

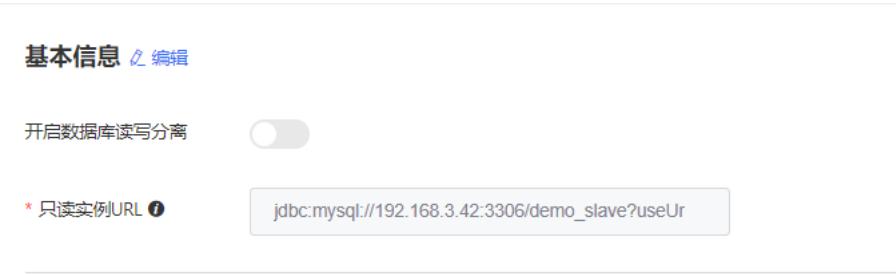
读写路由可以减轻单个节点的压力，实现数据负载均衡，提高系统的可伸缩性和容错能力。数据库读写路由功能无需改变代码，同时提供接口级别的读写访问控制以及主从一致性确认。

应用场景

- 读多写少的应用场景下，引入读写路由方案，使从库分担主库的数据读取压力，避免锁的竞争，在数据库访问的维度提升微服务的性能。
- 某些业务对数据库的查询访问过多，侵占主要业务的数据库资源，影响数据库实例的稳定性。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择数据库治理，在数据库读写路由页签下单击开启数据库读写分离右侧的编辑图标，配置正确的只读示例URL，开启读写路由分离开关，单击确认。



5. 单击新增规则，配置相关规则参数，单击新建。

规则参数说明

参数	说明
接口类型	需要进行读写流量路由接口的类型。
接口名称	需要进行读写流量路由接口的名称。
是否需要强一致性保证	开启强一致性保证，MSE会异步校验数据库主从实例的同步状态，保证当前接口的读请求在主从状态一致的条件下，才会路由至只读实例。
是否开启	接口级别的读写分离开关，当存在读写流量路由规则时，只有开启状态下的规则内所涉及的接口会被路由。

连接池治理

概述

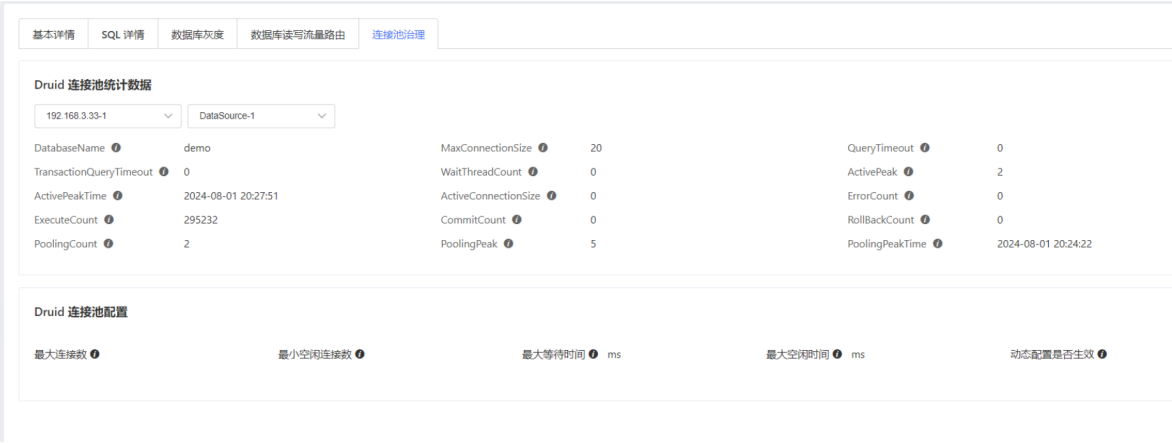
连接池治理提供连接池关键实时统计数据的监控能力以及动态连接池配置能力。通常应用会引入连接池以优化访问数据库的性能，在此场景下，可以基于连接池统计数据，动态调整连接池配置以适应线上环境变化。

使用限制

目前连接池治理仅支持Druid数据源，且要求Agent为最新版本。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择数据库治理，在连池统计数据区接池治理页签下，选择实例IP和进程ID对应的实例，选择数据源，查看连接池统计数据。
5. 在连接池动态配置区域，点击编辑，输入连接池配置，打开动态配置开关，单击确定。



统计数据参数说明

参数	说明
DatabaseName	数据库的名称。
MaxConnectionSize	最大连接数。
QueryTimeout	SQL请求的超时时长。
TransactionQueryTimeout	SQL事务的超时时长。
WaitThreadCount	当前等待获取连接的线程数。
ActivePeak	活跃连接峰值。
ActivePeakTime	活跃连接数量处于峰值的时间。
ActiveConnectionSize	活跃的连接数。
ErrorCount	SQL请求的错误数。
ExecuteCount	SQL请求的执行数。
CommitCount	SQL请求的提交数。
RollBackCount	SQL请求的回滚数。
PoolingCount	连接池中数据库连接的数量。
PoolingPeak	连接池中数据库连接数量的峰值。
PoolingPeakTime	连接池数量处于峰值的时间。

功能开关

概述

通常业务代码中包含许多的配置项用于控制服务启停、黑名单、提示文案等业务逻辑。开发者通常希望可以动态、实时地去查看和修改配置项，并且期望编写尽可能少的额外代码。

功能开关提供了一个轻量级的动态配置框架，可以在项目中快速接入配置，并在控制台实时管理配置项。与传统的配置中心不同，您无需关注配置项的解析逻辑和配置中心运维，声明配置对应的变量即可在应用配置控制台查看配置聚合信息并进行统一的管理。

使用场景

常用场景1： 业务降级

在高并发的场景下，可以声明开关变量，对非核心业务逻辑加入开关控制，然后在性能负载大的情况下在控制台控制开关，屏蔽非核心业务逻辑，从而提升系统性能。

常用场景2： 黑白名单

在常见的访问控制场景下，可以声明黑白名单变量，在控制台对黑白名单变量进行修改推送，动态控制黑白名单。

接入应用

在Pom文件中加入以下依赖

```
1.<dependency>
2.    <groupId>cn.ctyun</groupId>
3.    <artifactId>mse-switch-client</artifactId>
4.    <version>1.0-SNAPSHOT</version>
5.</dependency>
```

声明功能开关

在字段上加上cn.ctyun.mse.center.annotation.AppSwitch 注解，字段修饰符必须为 public static。

（可选）在字段上加上cn.ctyun.mse.center.annotation.SwitchGroup注解，定义开关的分类，注解缺失时默认按类名进行分类。

```
1.public class ConfigTest {
2.    @AppSwitch(des = "String 类型开关")
3.    public static String stringSwitch = "string";
4.
5.    @AppSwitch( des = "Float 类型开关" )
6.    public static Float floatSwitch = 4.87F;
7.}
```

初始化开关

调用以下方法完成初始化，方法参数为可变参数，支持初始化多个开关常量类。SwitchManager.init(ConfigTest.class)。

配置启动参数并重新部署

```
-Dmse.licenseKey={your licenseKey}

-Dmse.namespace=default
```

-Dswitch.endpoint=xxxxxxxxxx

验证接入结果

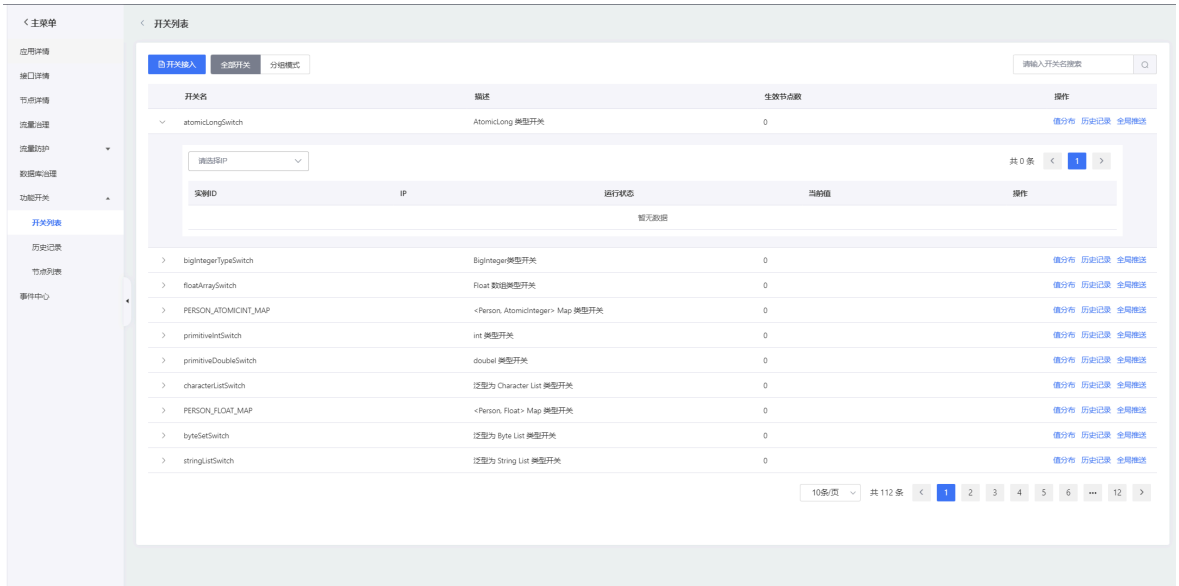
进入控制台-应用-功能开关页面，若开关列表出现声明的开关且节点列表出现应用所在节点，则说明接入成功。

查看功能开关

功能开关可以查看开关类型、生效节点数、开关分布信息、使用实例 ID 和 IP 等信息。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择功能开关，进入目标应用的开关列表，列表展现方式有分组模式和全部开关模式，可以展开查看开关的描述、生效节点数、使用的实例ID、IP 等信息。



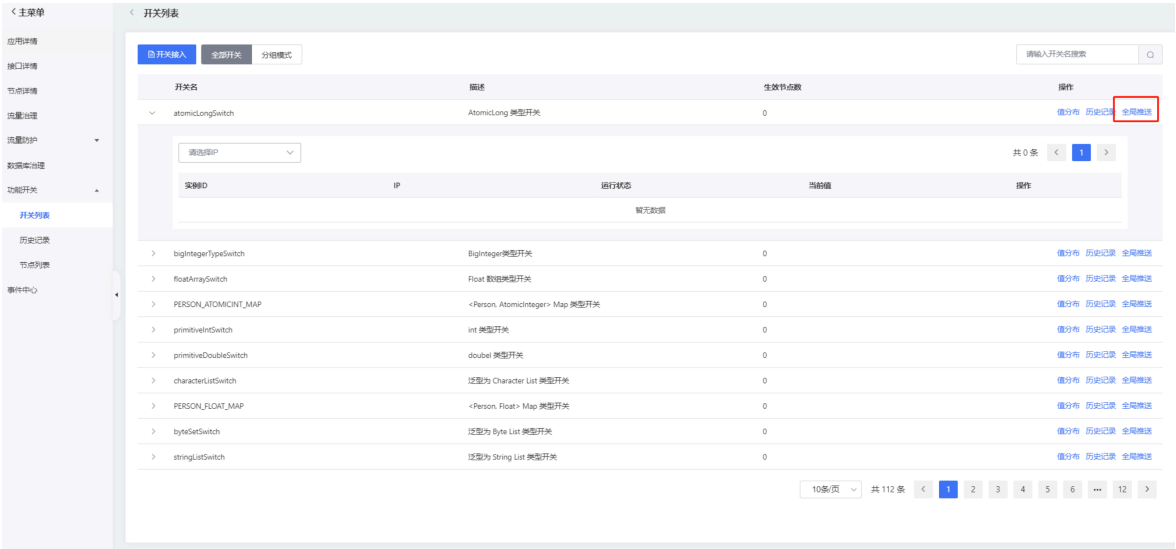
5. 单击操作列的值分布，即可查看对应开关信息和分布信息，包括值编号、开关值等。

推送开关值

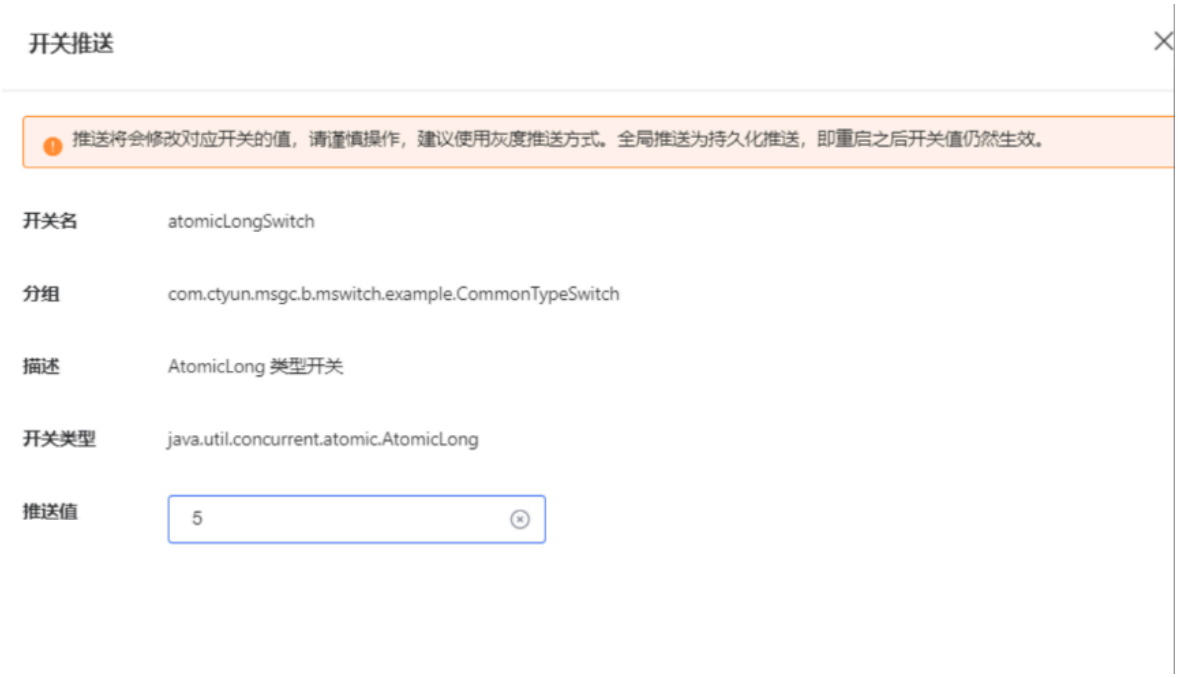
功能开关提供开关值的推送能力，动态覆盖应用的配置项。功能开关支持全局推送和单机推送两种推送方式。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。
3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择功能开关，进入目标应用的开关列表。
5. 单击开关列表页面操作列的全局推送或单机推送，在右侧弹出开关推送页面，在此页面中可查看开关名、分组、开关类型等信息，并可以编辑推送值。



6. 完成编辑推送值后，单击下一步：值对比，会显示出修改点。若还需修改，则单击上一步：返回修改，若修改完成，则单击单机推送或全局推送（全局推送为持久化推送，即重启之后开关值仍然生效）。



查看历史记录

功能开关提供开关推送的历史记录查询，包括推送的值、类型、操作时间、生效节点等信息。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择应用治理。

3. 在应用治理页面的应用卡片页签单击目标应用卡片。
4. 在左侧导航栏选择功能开关，单击历史记录，进入目标应用的开关推送历史记录页面。
5. 按开关名、推送类型、操作时间等条件过滤。

6. 单击操作列下的查看，可以查看此条推送记录的开关名、分组、推送类型、推送值和生效IP。

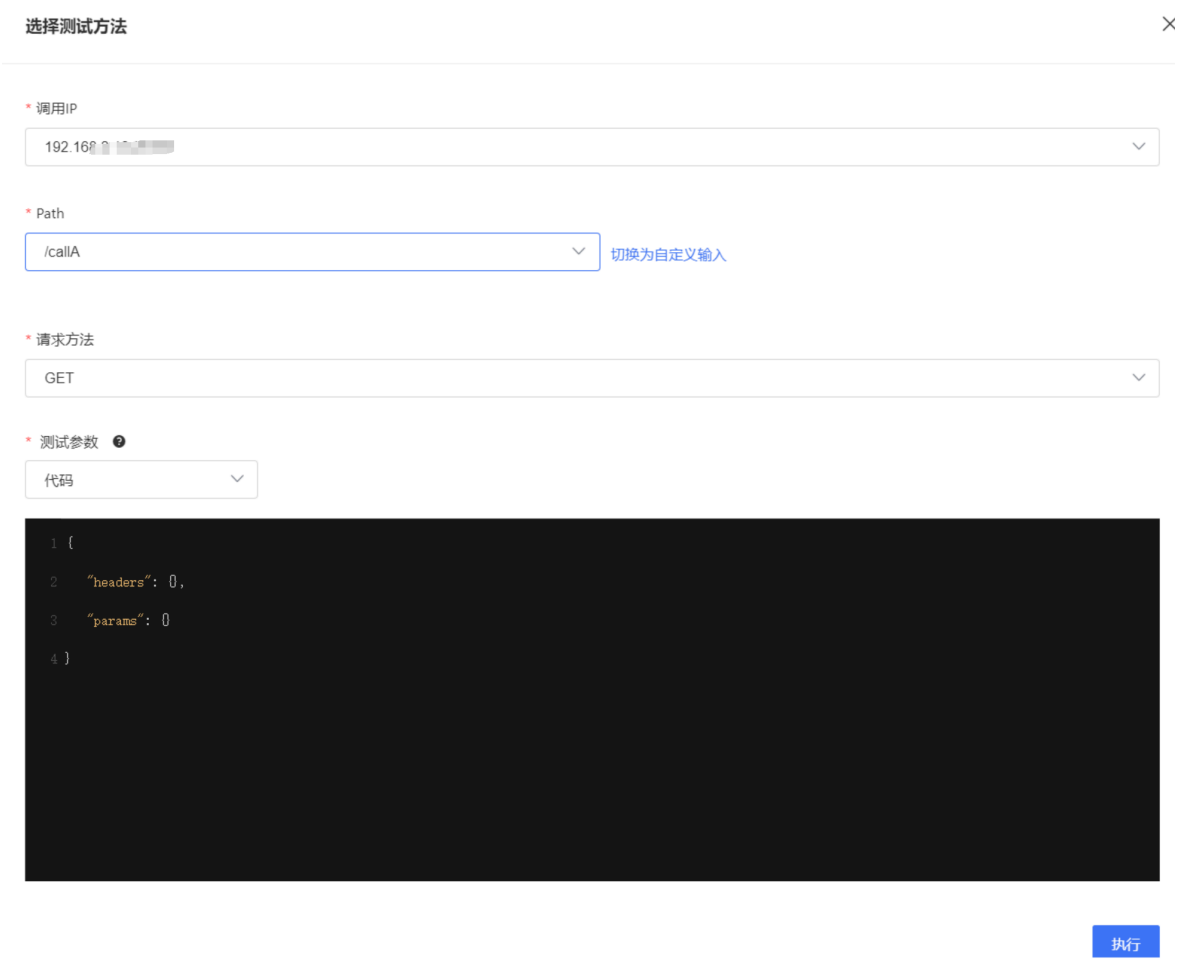
开发测试治理

服务测试

在日常开发中，开发人员或测试人员需要临时调用线上服务来调试已经部署的服务或查询线上数据。服务测试功能可以让您在控制台填写调用参数、发起服务调用，并得到服务调用的结果。

登录微服务治理中心控制台，在左侧导航栏选择微服务治理中心->开发测试治理->服务测试，在测试列表可以查看已开启的微服务应用的相关信息，包括服务名称、应用名称、实例数量等。

选择需要测试的服务，点击测试，在选择测试方法面板中设置测试相关参数，包括调用IP、Path、请求方法、测试参数，然后单击执行，在结果区域查看测试是否成功。



参数说明：

参数	说明
调用IP	选中服务的提供者ip。
Path	方法请求路径。
请求方法	支持GET、POST、PUT和DELETE。
测试参数	方法请求携带的请求数据，包含headers和params。 示例：{"headers":{"source":"inner"},"params":{"username":"test"}}GET：headers代表请求头、params代表请求参数。POST：headers代表请求头、params代表请求体。PUT：headers代表请求头、params代表请求体。DELETE：headers代表请求头、params代表请求参数。

自动化回归-用例管理

概述

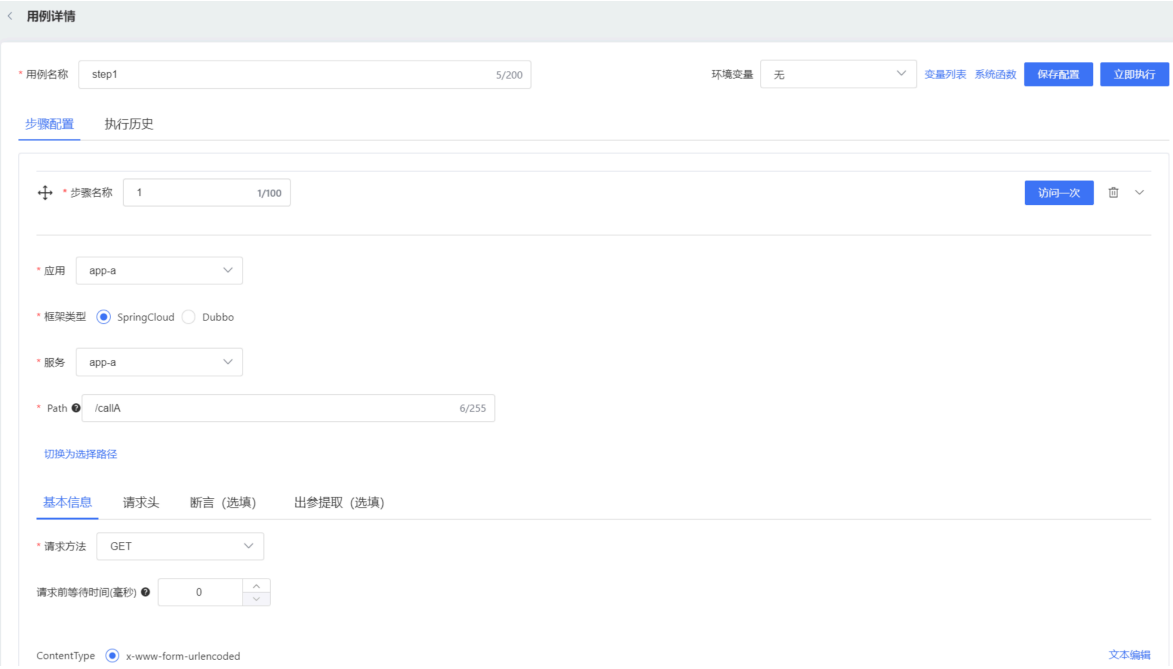
用例管理功能可以让您在控制台快速编排被测服务，帮助您高效管理、回归业务测试场景，完成业务快速验证和交付。

用例列表

登录微服务治理中心控制台，在左侧导航栏选择微服务治理中心->开发测试治理->用例管理，在用例列表可以查看已创建用例的相关信息。

用例创建

在用例列表点击创建用例，进入创建用例页面。一个测试用例可以包含多个测试步骤，多个步骤通过串联方式执行，后序步骤可以使用前序步骤中的出参提取变量。

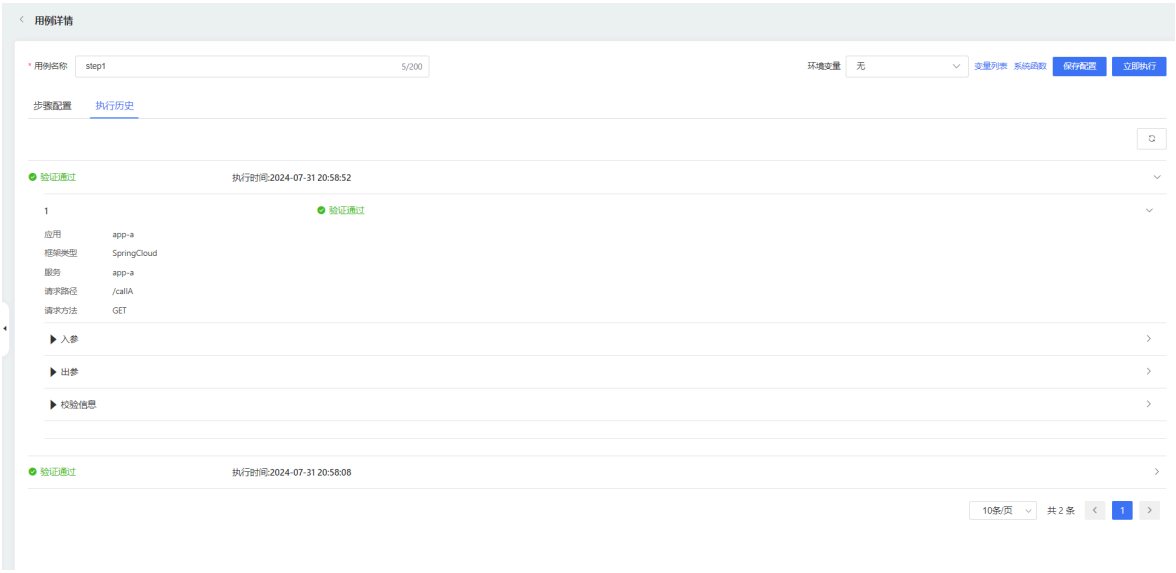


访问一次

在用例创建页面中，点击访问一次，弹出单步骤调试结果，查看此次请求入参和出参。单击出参提取助手，弹出出参提取助手窗口，选择需要提取的出参参数进行复制。

用例执行

保存用例后，点击执行，可以在执行历史页面查看本次执行入参、出参和校验信息。



其他操作

复制测试用例：在用例管理列表，单击操作列的复制，可生成一条新的测试用例。

删除测试用例：在用例管理列表，单击操作列的删除，可删除该测试用例。

自动化回归-用例集

概述

用例集功能可以让您在控制台关联和执行多个测试用例，帮助您高效管理测试用例，完成业务快速验证和交付。

用例集列表

登录微服务治理中心控制台，在左侧导航栏选择微服务治理中心->开发测试治理->用例集，在用例集列表可以查看已创建用例集的相关信息。

用例集创建

在用例集列表点击创建用例集，进入创建用例集页面。

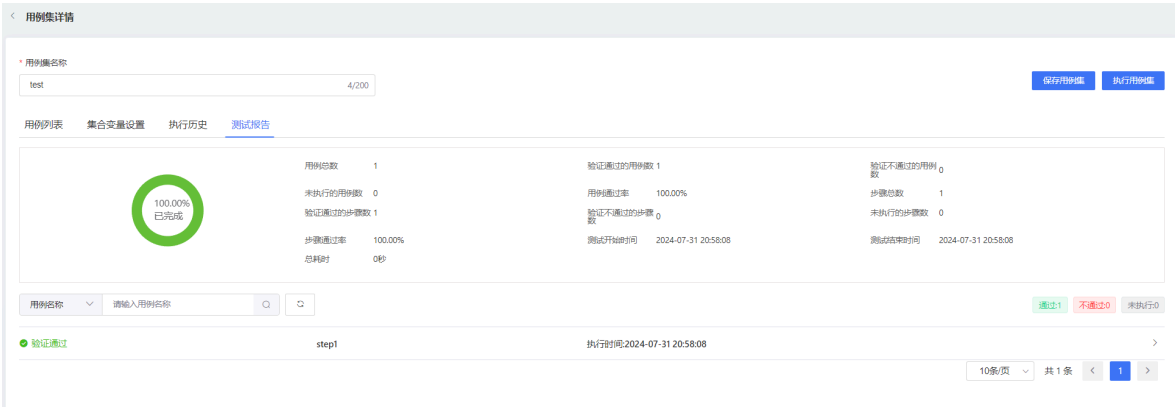
用例关联

创建用例集后，点击详情，查看关联的用例列表，支持导出脚本线下编辑后再导入。

点击关联用例，可以对用例进行关联或者取消关联。

用例集执行

1. 保存用例集后，点击执行，可以在执行历史页面查看相关信息。
2. 点击测试报告，可以查看报表统计、关联用例及相关步骤的详细信息。



其他操作

删除用例集：在用例集列表，单击操作列的删除，可删除该用例集。

服务 Mock

在系统研发阶段，经常会有模拟数据的需求。通过配置服务Mock，可以根据请求参数返回不同的结果，并且随机生成返回的结果，模拟真实的后端服务，支持前后端联调。

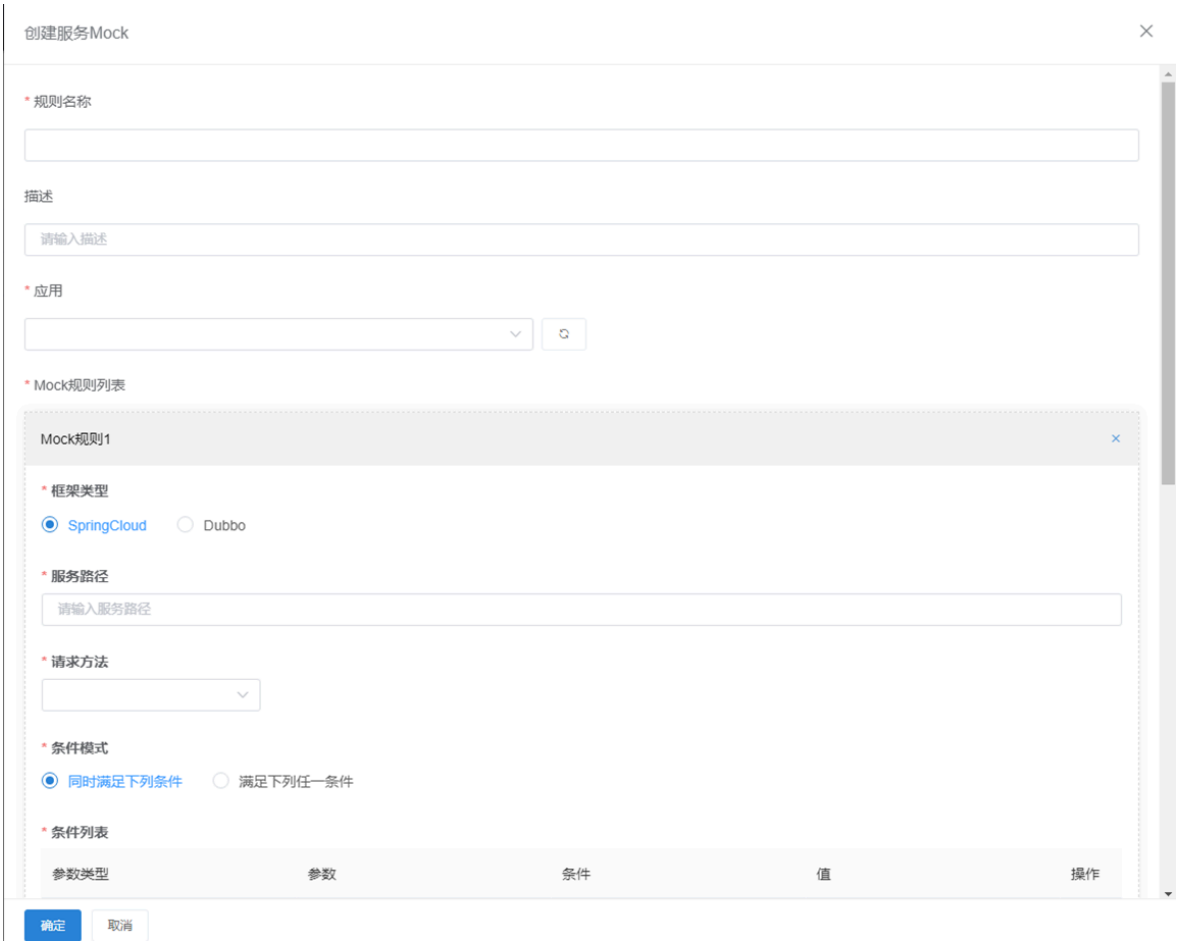
版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Dalston及以上版本	客户端：Feign、RestTemplate
Dubbo	2.5.3+	--
jdk版本	1.8+	--

开通服务Mock

1. 登录微服务治理中心控制台。
2. 在左侧导航栏选择开发测试治理->服务Mock。
3. 在服务Mock页面单击创建服务Mock。

4. 在创建服务Mock页面配置相关参数，并点击保存。



服务Mock规则参数说明：

参数	说明
规则名称	服务Mock规则的名称。
描述	规则的详情描述。
应用	需要Mock的应用。
Mock规则列表	以下为Mock相关规则。
框架类型	分为SpringCloud和Dubbo SpringCloud需要设置服务路径和请求方法 Dubbo需要设置请求方法。
条件模式	同时满足下列条件和满足下列任一条件。
条件列表	单击添加新的规则条件：1，对SpringCloud应用，支持的参数是：Parameter、Header、Cookie、Body。2，对Dubbo应用，支持的入参是：RpcContext，Parameter。
Mock策略	默认支持返回自定义Json数据。
返回数据	自定义Mock返回数据。

参数	说明
返回延迟	自定义请求响应时间。单位ms。
默认状态	默认打开或关闭规则。

网关治理

概述

可以对网关进行流量控制，支持流量治理和网关防护，网关防护可以从流量入口处拦截突发的流量，防止下游服务被压垮。网关防护的主要功能包括：

1. 对路由配置中的特定路由进行流量控制，或者自定义一组API进行流量控制。
2. 对请求的客户端IP、Header或者URL参数进行流控。
3. 限制某个API的调用频率，支持秒、分钟、小时、天等多个时间维度。

接口详情

在接口详情页面，主要展示该应用所有接口的通过QPS、限流QPS、异常QPS指标、RT、并发数据等。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择网关治理。
3. 在网关治理页面的应用卡片页签单击目标网关卡片。
4. 在左侧导航栏选择接口详情。

节点详情

概述

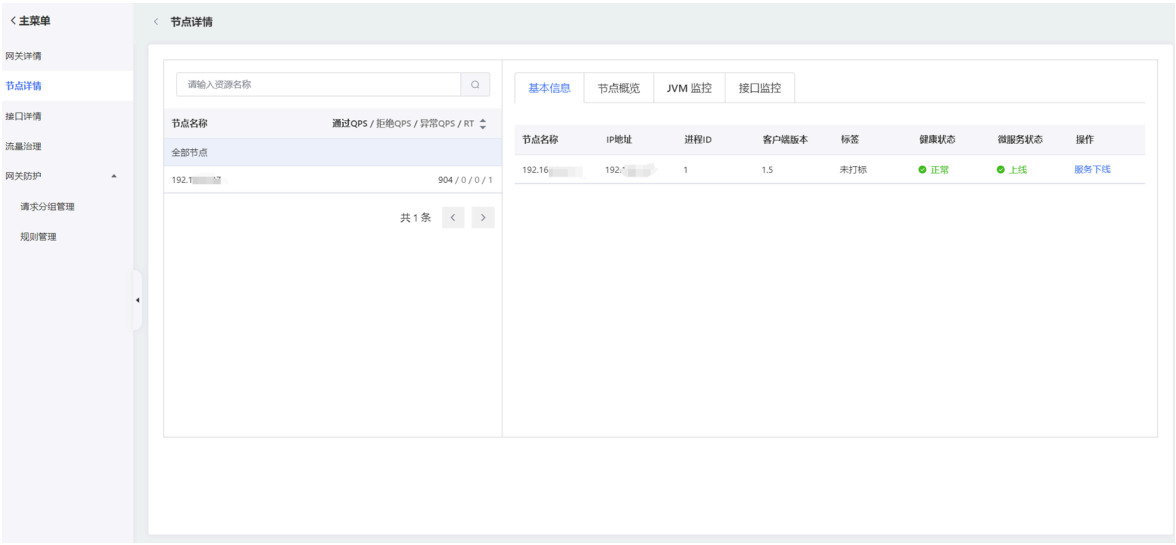
在节点详情页面，可以查看所有节点的通过QPS、限流QPS、异常QPS、RT、并发等指标，并且可以在此页面对接口管理流控规则进行操作。一个节点对应一个JVM进程，当多个JVM接入单机后，即展示为多个节点。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择网关治理。
3. 在网关治理页面的应用卡片页签单击目标网关卡片。
4. 在左侧导航栏中选择节点详情。

功能介绍

节点页面展示了应用的所有节点详细信息以及这些节点当前的通过QPS、限流QPS、异常QPS和RT等数据。同时，该页面还展示5分钟内的流量监控指标，节点所在机器的CPU信息、LOAD信息、物理内存、DISK和网络流量。



此外，该页面还提供了所在节点的5min内的JVM信息，包括堆内存详情，非堆内存详情，元空间详情，JVM线程数，缓冲区详情和GC等数据。

流量治理

标签路由

标签路由是将每个服务打上一个标签，通过标签将标签相同的服务分为同一个分组，然后约束流量在同一个分组内流转，以此实现灰度发布、金丝雀发布、蓝绿发布等功能。

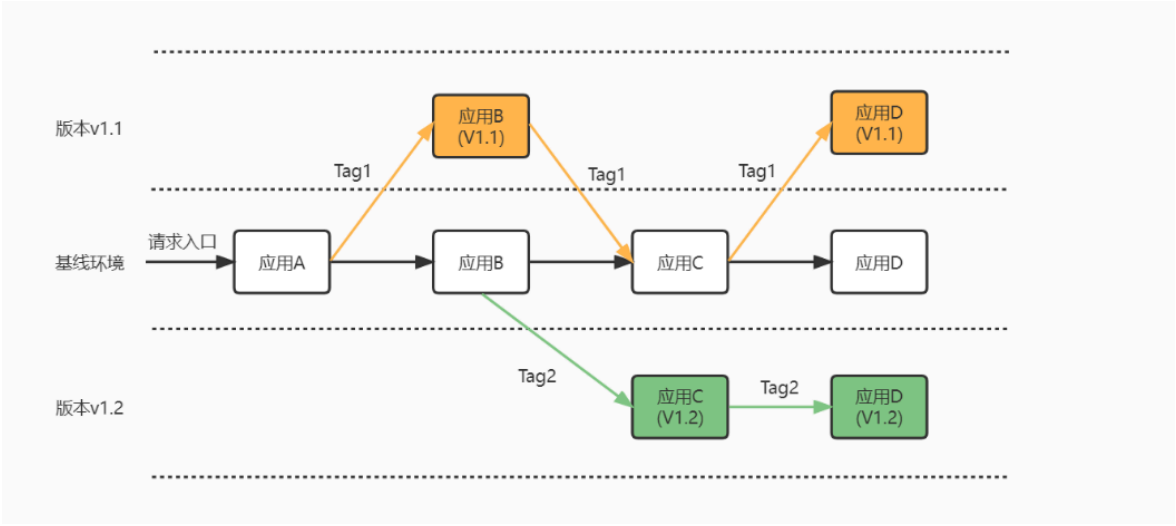
版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Edgware及以上版本	客户端：Feign、RestTemplate 负载均衡：Ribbon、LoadBalancer
Dubbo	2.5.3~2.7.8	支持Alibaba Dubbo、Apache Dubbo
注册中心	Nacos、Eureka、Zookeeper	--
jdk版本	1.8+	--

应用场景

A/B测试

在营销活动中，经常会有A/B测试，A/B测试是为了测试不同的规则对业务的影响，所以一个应用会有多个版本同时运行，可以通过标签的方式区分不同的版本，对不同版本的应用进行流量隔离，将特殊用户的流量路由到特殊版本，从而实现A/B测试。



开通标签路由

步骤1：应用设置标签。

为云容器引擎集群应用设置标签，为应用容器添加环境变量MSE_SERVICE_TAG=gray。

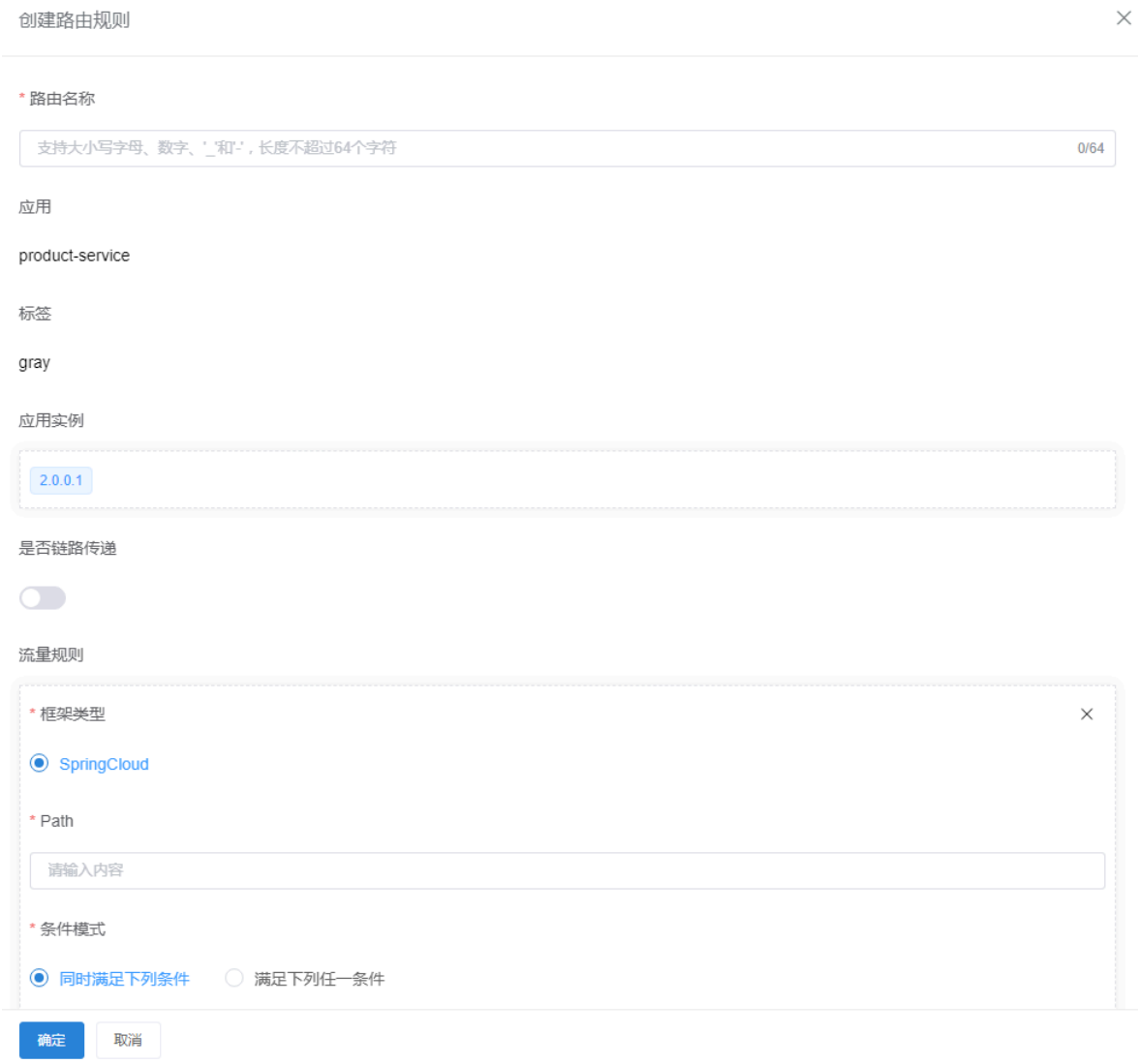
为ECS应用设置标签，在启动应用时，添加JVM启动参数-Dctgcloud.service.tag=gray。

步骤2：在微服务治理中心控制台创建标签路由。

- 登录微服务治理中心控制台。
- 在左侧导航栏选择 微服务治理中心 ->网关治理。
- 在网关治理页面单击目标应用卡片。
- 在网关页面左侧导航栏选择流量治理 - 标签路由，查看服务标签。

- 在标签路由页点击流量分配，为标签按比例分配流量。

- 在标签路由页的流量规则栏新增流量规则。



流量规则参数说明：

参数	说明
路由名称	路由规则的名称。
应用	所选的应用。
标签	设置的标签名。
应用实例	设置了该标签的实例ip。
是否链路传递	代表开启全链路流控。
框架类型	Spring Cloud Dubbo。
Path	SpringCloud为path路径，Dubbo为接口。

参数	说明
条件模式	满足一个条件，或者满足所有条件。
条件列表	可以设置Header、Cookie、Parameter和Body Content四种参数类型。
是否开启流量规则	流量规则开关。

离群摘除

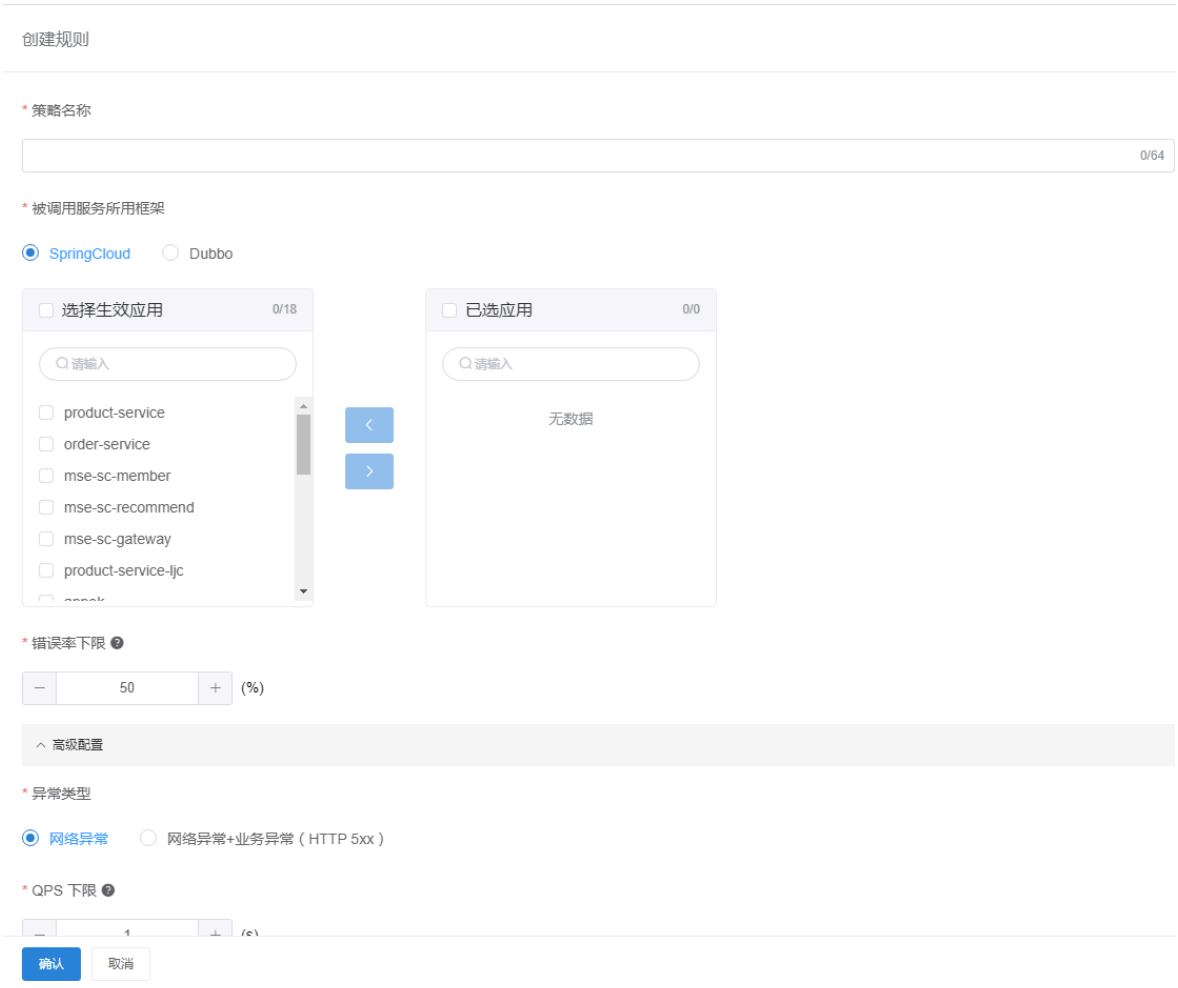
在微服务场景中，当服务提供者的实例出现异常时，服务消费者无法感知到提供者出现异常，此时就可能出现异常调用。通过配置离群摘除功能可以实时监测下游实例的可用性，摘除异常实例，提升业务的可用性。

版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Dalston及以上版本	--
Dubbo	2.5.3+	支持Apache Dubbo 不支持Alibaba Dubbo
Jdk版本	1.8+	--

开通离群摘除

1. 登录微服务治理中心控制台。
2. 在左侧导航栏选择 微服务治理中心 ->应用治理。
3. 在应用治理页面单击目标应用卡片。
4. 在应用页面左侧导航栏选择流量治理 - 离群实例摘除，可查看离群实例摘除规则列表。
5. 在创建离群实例摘除页面配置相关参数，并单击确认。



离群摘除规则参数说明：

参数	说明
策略名称	离群摘除规则的名称。
被调用服务所用框架	Spring Cloud或Dubbo。
选择生效应用	选择生效应用后，该应用调用的异常应用实例会被摘除。
错误率下限	被调用的应用中某个应用实例的错误率高于设置的域值后，将摘除该实例。默认值为50%。例如该实例在统计时间窗口内被调用10次，有6次调用失败，错误率为60%，超过了配置的错误率域值（50%），则从应用中移除该实例。
异常类型	目前只支持网络异常+业务异常（HTTP 5xx）。
QPS下限	QPS按照统计时间窗口进行计算，默认为10秒。
摘除实例比例上限	摘除的异常实例比例上限，即达到阈值后，不再摘除异常实例。

参数	说明
恢复检测单位时间	发现实例异常后，检测异常实例的单位时间（修改）。
未恢复累计次数上限	持续对异常实例进行检测，检测间隔随检测次数按恢复检测单位时间线性增加，当达到设置的检测次数上限后，会按最长时间间隔持续检测异常实例是否恢复。
默认状态	默认是否开启离群摘除规则。

流量防护

请求分组管理

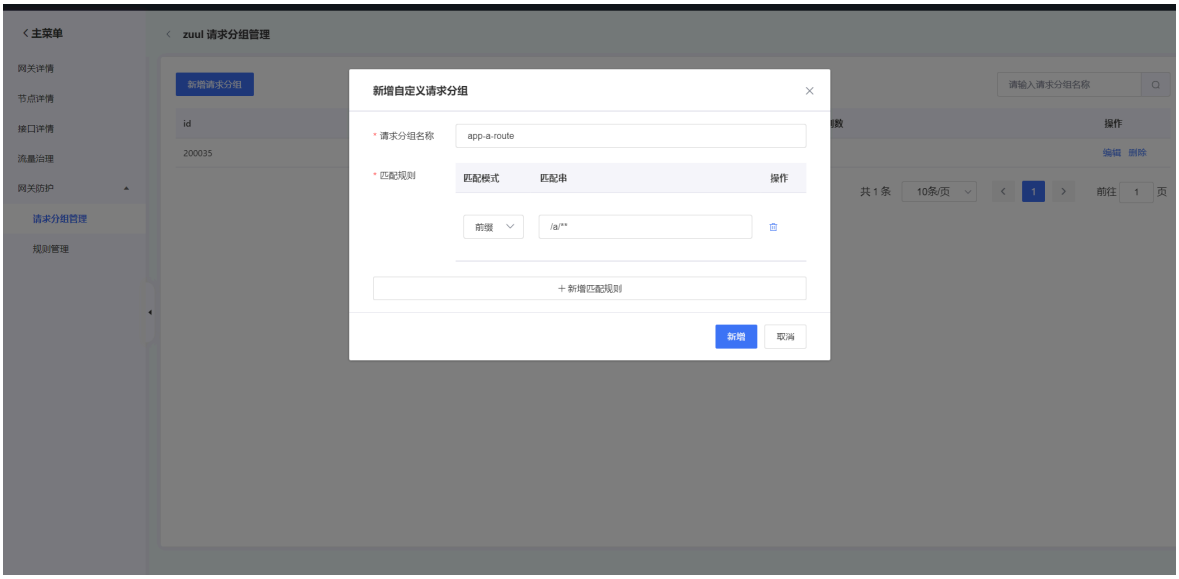
可以创建请求分组，并自定义每个请求分组下的URL路径匹配规则，以增强网关防护功能。通过这种方式，您可以对特定的请求分组进行流量控制，以确保网络安全。网关防护可以针对自定义的请求分组进行流量控制。

新建自定义请求

1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择网关治理。
3. 在网关治理页面的应用卡片页签单击目标网关卡片。
4. 在左侧导航栏选择网关防护 - 请求分组管理，单击新增请求分组。
5. 在新建自定义请求分组对话框中，填写请求分组名称（说明该名称需要全局唯一，并且不能与路由配置文件中的路由ID重复）。
6. 填写URL路径匹配规则，先选择匹配模式，再根据匹配模式的要求填写匹配串。匹配模式分为以下三类：

- 精确模式：严格按照给定的匹配串来匹配URL路径。示例：/path代表严格按照/path这个路径来匹配。
- 前缀模式：按照给定的匹配串来进行前缀匹配，匹配串需符合Spring Web风格。示例：/path/** 代表匹配以/path/开头的所有URL，像/path/123这种URL都可以匹配。
- 正则模式：按照给定的正则表达式匹配串来进行匹配。
- 匹配串：根据匹配模式的要求填写匹配串。

7. 单击+新增匹配规则，可添加多个URL路径匹配规则。
8. 单击新增，完成自定义请求分组的创建。新增的请求分组将出现在请求分组管理页面。

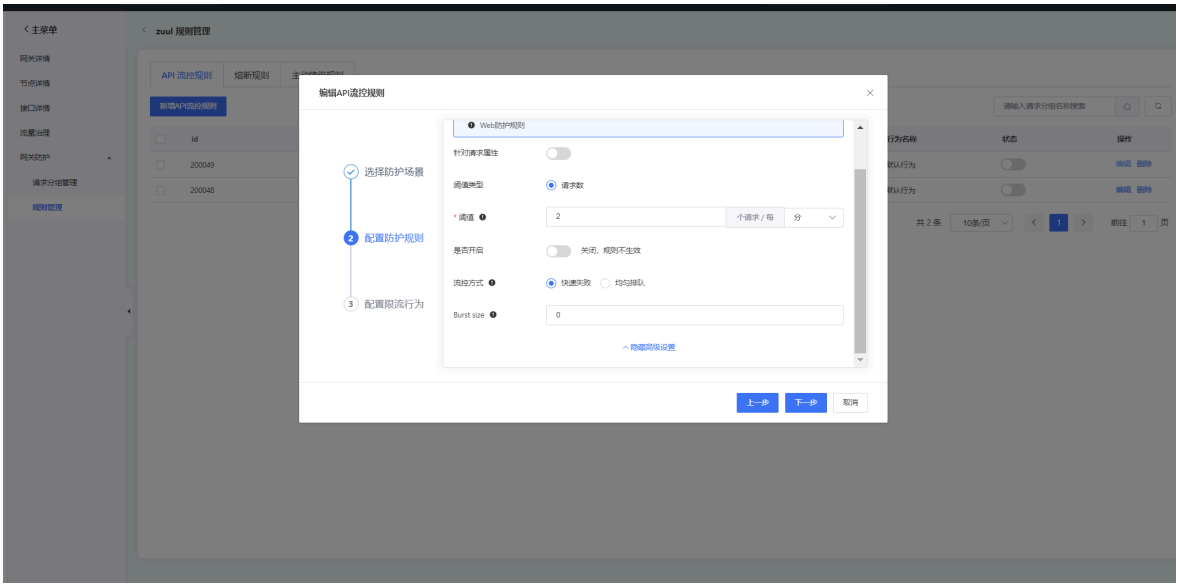


API流控规则

为网关应用配置网关流控规则后，微服务治理将从流量入口处拦截激增的流量，防止下游服务被压垮。这样做可以保证网络的稳定性和可靠性。

新建网关流控规则

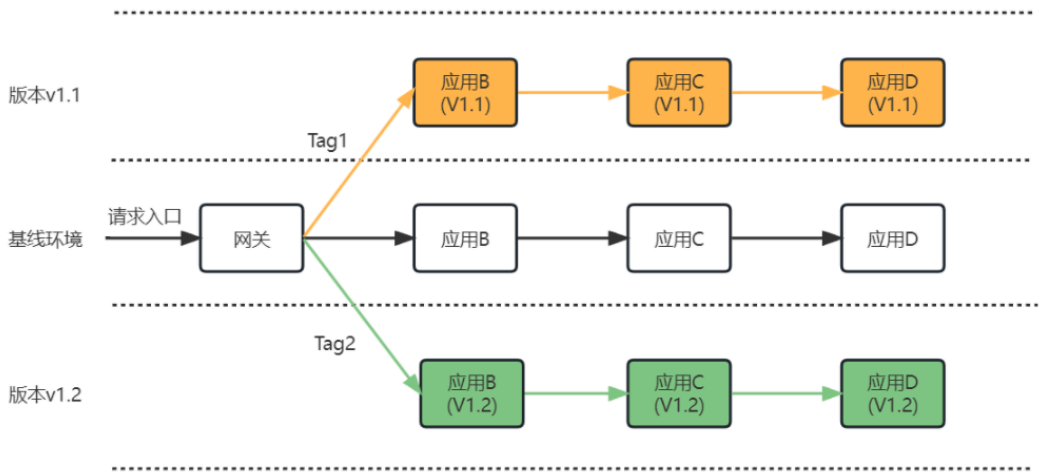
1. 登录微服务治理控制台。
2. 在控制台左侧导航栏中选择网关治理。
3. 在网关治理页面的应用卡片页签单击目标网关卡片。
4. 在左侧导航栏选择网关防护 - 规则管理 - API流控规则，在新增流控规则对话框中，配置流控规则。
5. 单击新增。新增的规则将出现在API流控规则页面。



全链路灰度

概述

在微服务场景中，一次版本的发布可能会涉及多个应用的灰度发布。全链路灰度功能可以将多个相同版本的应用划分为同一个泳道，通过全链路流量控制的功能将相同版本的应用隔离成一个独立的运行环境（泳道），通过设置泳道规则将请求流量路由到目标版本的应用。



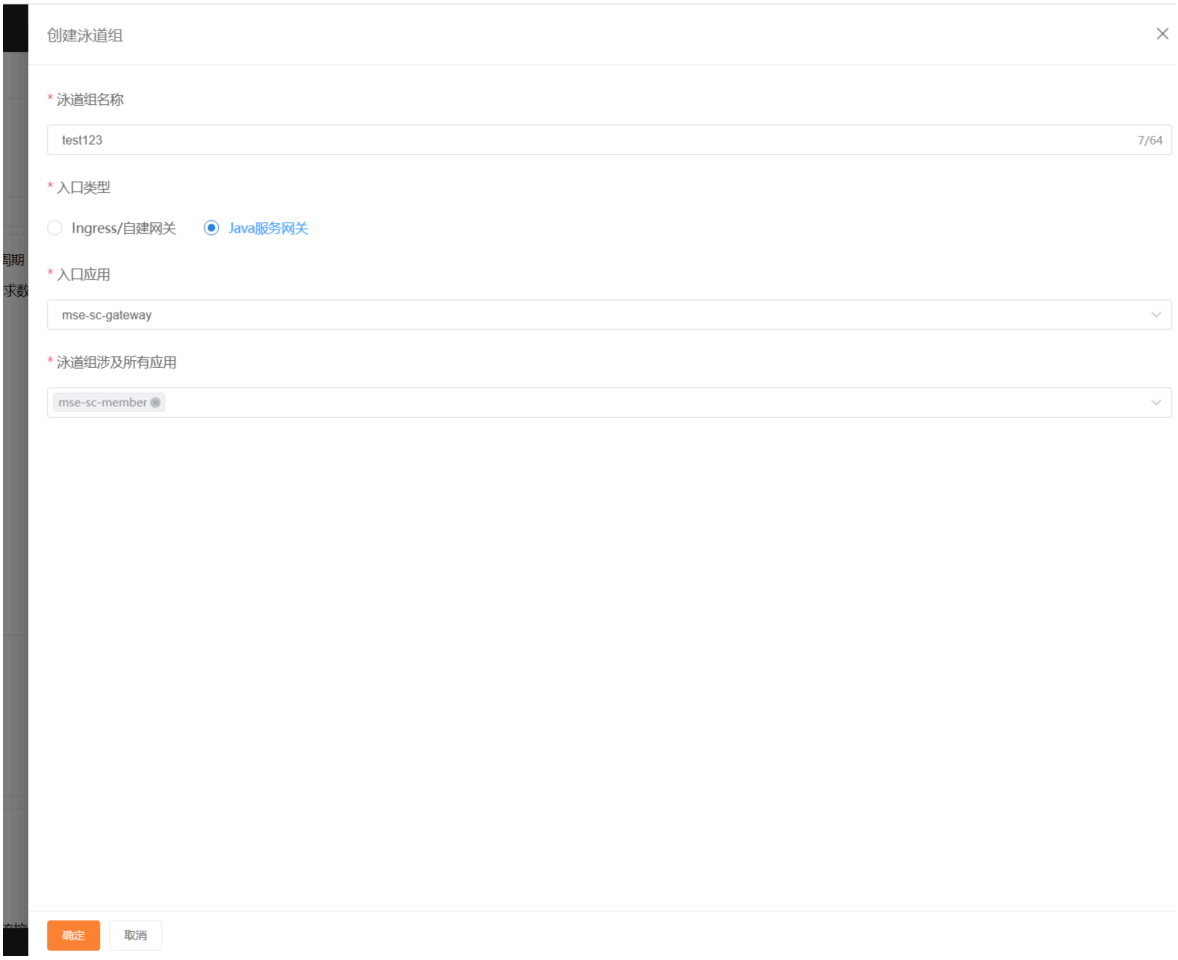
版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Edgware及以上版本	客户端：Feign、RestTemplate
Dubbo	2.5.3-2.7.8	--
jdk版本	1.8+	--

开通全链路灰度

步骤1：创建泳道组

1. 登录微服务治理控制台。
2. 在左侧导航栏选择全链路灰度，点击创建泳道组。
3. 在创建泳道组页面，设置泳道组相关参数，然后单击确定。



参数详情：

参数	说明
泳道组名称	自定义设置的泳道组名称。
入口类型	选择Ingress/自建网关需要自定义路由规则，选择Java服务网关可以在控制台设置入口应用流量规则。
入口应用	选择Java服务网关时的入口应用。
泳道组涉及的应用	当前泳道组涉及的应用。

步骤2：创建泳道

1. 找到目标泳道组页面，点击创建第一个分流泳道。
2. 设置泳道名称，选择目标应用所属标签，创建泳道。

参数详情：

参数	说明
泳道名称	泳道的名称。
添加应用	选择应用所属的标签。

安全治理

全局鉴权

微服务治理中心提供全局鉴权能力，通过创建鉴权规则，无需变更所有服务即可便捷地实现多个微服务之间通信的身份验证，搭建微服务架构安全访问体系。

功能入口

1. 登录微服务治理控制台。
2. 在控制台左侧栏选择安全治理，单击全局鉴权。

相关操作

- 创建鉴权规则：单击 创建鉴权 ，配置鉴权信息，单击确认完成创建。
- 查看鉴权规则：单击目标鉴权规则操作列的详情 ，查看鉴权规则信息。
- 开启鉴权规则：单击目标鉴权规则操作列的开启 ，使鉴权规则生效。
- 关闭鉴权规则：单击目标鉴权规则操作列的关闭 ，关闭鉴权规则。
- 编辑鉴权规则：单击目标鉴权规则操作列的编辑 ，编辑鉴权规则。
- 删除鉴权规则：单击目标鉴权规则操作列的删除 ，删除鉴权规则。

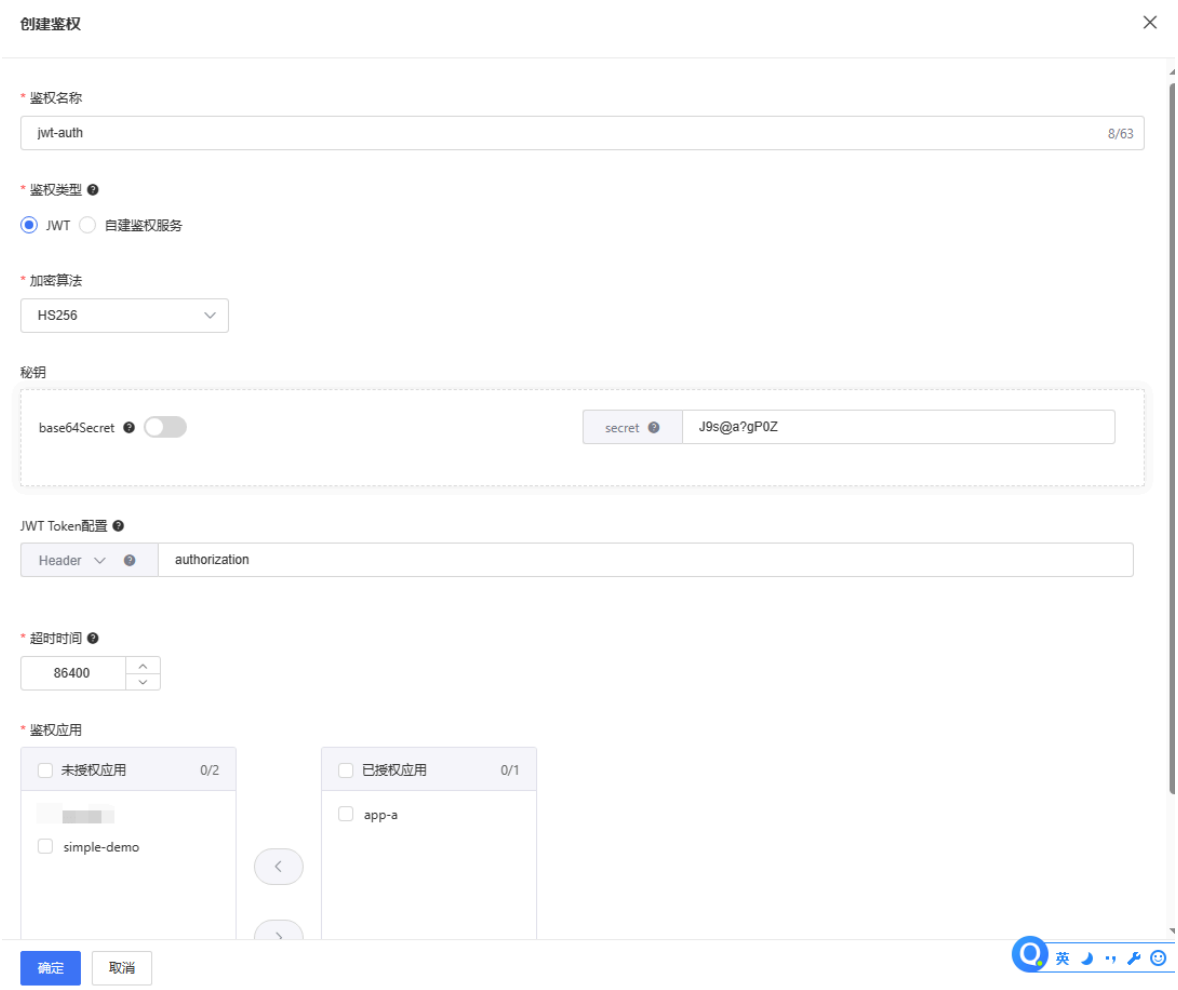
JWT鉴权

JWT（JSON Web Token）用于网络应用间以JSON安全传输信息，该信息可以由使用密钥（使用 HMAC 算法）或使用RSA密钥对进行签名，因此JWT是可以被信任和验证的。

微服务治理中心的JWT鉴权能力基于JWT标准实现了一套服务安全调用的身份验证机制，无需中心化的鉴权服务，只需接入微服务治理中心并配置JWT鉴权，应用即可通过Token验证请求者的身份。

创建鉴权

1. 进入全局鉴权页，单击创建鉴权。
2. 在鉴权类型中选择JWT鉴权，配置JWT鉴权相关参数，最后单击确定。



配置项	描述
鉴权名称	唯一标识鉴权规则的名称。
鉴权类型	选择JWT。
加密算法	支持HS256、HS512、RS256。
base64编码	选择HS256/ HS512算法时填写，指定密钥是否base64编码。
密钥	选择HS256/ HS512算法时填写，提供密钥。
RSA密钥	选择RS256算法时填写，需要提供RSA算法公钥和私钥。
JWT Token配置	从请求获取JWT Token的位置，支持选择header、query或Cookie，并指定对应的key。

配置项	描述
过期时间	JWT Token过期时间（单位：秒），默认值86400。
鉴权应用	选择需要鉴权的应用范围。注意 ：应用将验证令牌，并携带令牌信息进行下游调用，因此对于每个鉴权应用，其上游调用链路中的应用需纳入鉴权范围。

验证鉴权

1. 开启鉴权。
2. 请求鉴权应用，预期请求结果为拒绝访问。
3. 签发Token。

   签发方式一：调用微服务治理中心OpenAPI获取签发的Token。

   签发方式二：用户可以基于加密算法、base64编码、密钥配置生成Token，满足开放标准（RFC 7519）。
4. 根据配置携带Token请求鉴权应用，预期请求结果为正常访问。

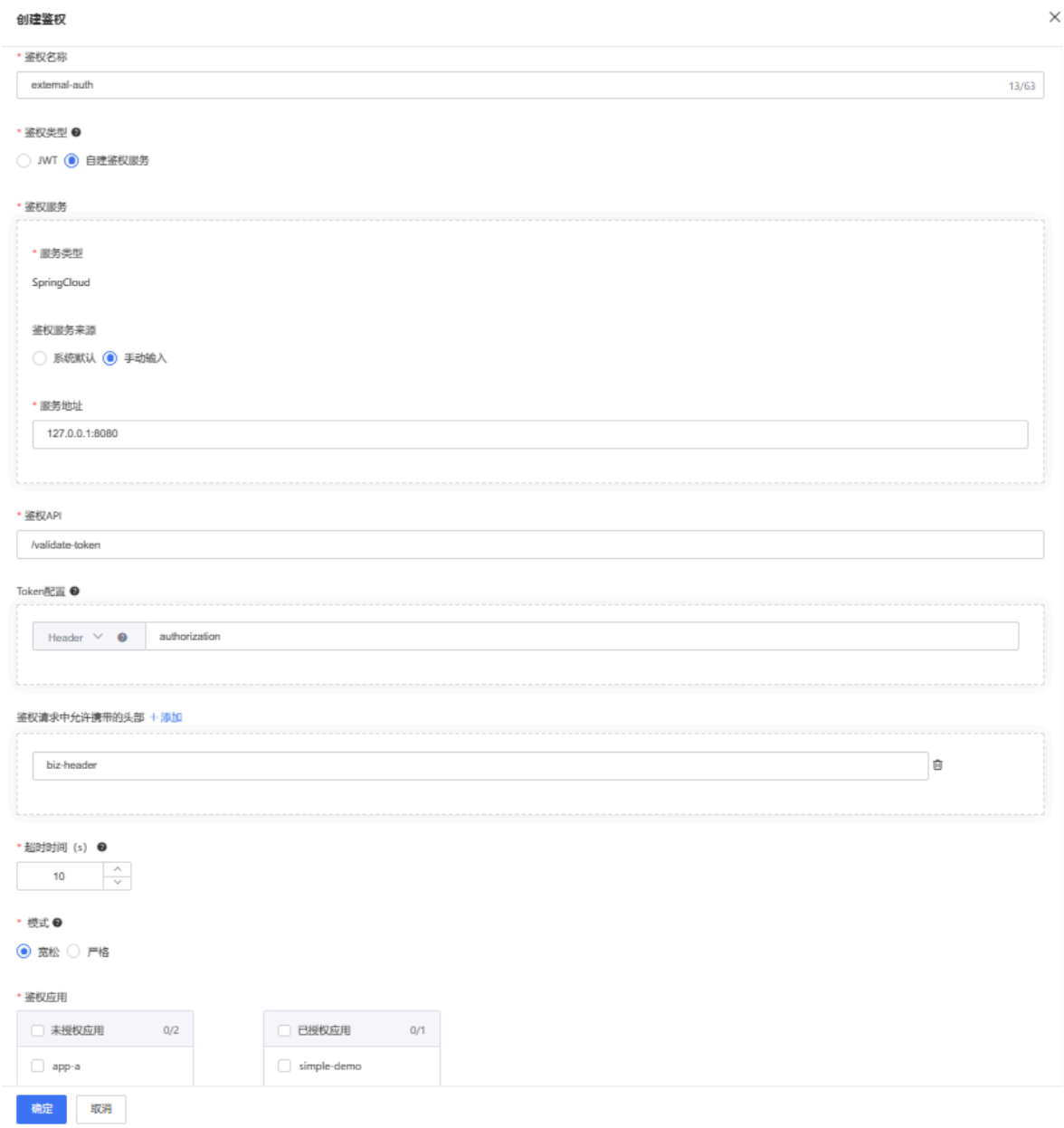
自建服务鉴权

当Token为用户自定义的格式，服务受到请求后，需要访问中心化鉴权服务来验证该Token，以实现对Token的安全验证。

微服务治理中心的自建服务鉴权能力支持服务在接收请求时，携带Token访问额外的鉴权服务验证身份，实现接口的安全调用。

创建鉴权

1. 进入全局鉴权页，单击创建鉴权。
2. 在鉴权类型中选择自建服务鉴权，配置自建服务鉴权相关参数，最后单击确定。



配置项	描述
鉴权名称	唯一标识鉴权规则的名称。
鉴权类型	选择自建服务鉴权。
鉴权服务	选择自建鉴权服务，可以通过下拉选择已接入服务和地址或手动输入服务地址。
鉴权API	鉴权服务提供的鉴权API的路径。

配置项	描述
Token配置	从请求获取Token的位置，支持选择header、Cookie，并指定对应的key。
鉴权请求中允许携带的头部	如果需要额外携带客户端请求中的头部，那么需要在字段中按需配置头部。说明： Host、Path头部会被默认添加。
超时时间	请求鉴权服务最大等待时间（单位：秒），默认值：10。
模式	支持宽松模式和严格模式，默认使用宽松模式：宽松模式：当鉴权服务不可用时（鉴权服务建立连接失败或者返回5xx请求），接受客户端请求。严格模式：当鉴权服务不可用时（鉴权服务建立连接失败或者返回5xx请求），拒绝客户端请求。
鉴权应用	选择需要鉴权的应用范围。注意 ：应用将验证令牌，并携带令牌信息进行下游调用，因此对于每个鉴权应用，其上游调用链路中的应用需纳入鉴权范围。

校验规则：

自建鉴权服务接收请求后，可根据Token和请求头完成身份验证，并返回验证结果。

1. 仅通过HTTP状态码判定。

- 鉴权服务返回HTTP状态码为200，表明Token有权限访问该后端资源。
- 鉴权服务返回HTTP状态码为401或403，表明Token无权访问该后端资源。

2. 通过指定HTTP响应头部判定，适用于响应HTTP状态码要求为200。

- 鉴权服务的响应头部x-msgc-auth-result=true，表明Token有权限访问该后端资源。
- 鉴权服务的响应头部x-msgc-auth-result=false，表明Token无权访问该后端资源。

验证鉴权

1. 开启鉴权。
2. 请求鉴权应用，预期请求结果为拒绝访问。
3. 根据配置携带自主颁发的Token请求应用，鉴权应用将携带Token向自建鉴权服务发出请求，若请求结果符合校验规则，则预期请求结果为正常访问，否则拒绝访问。

服务鉴权

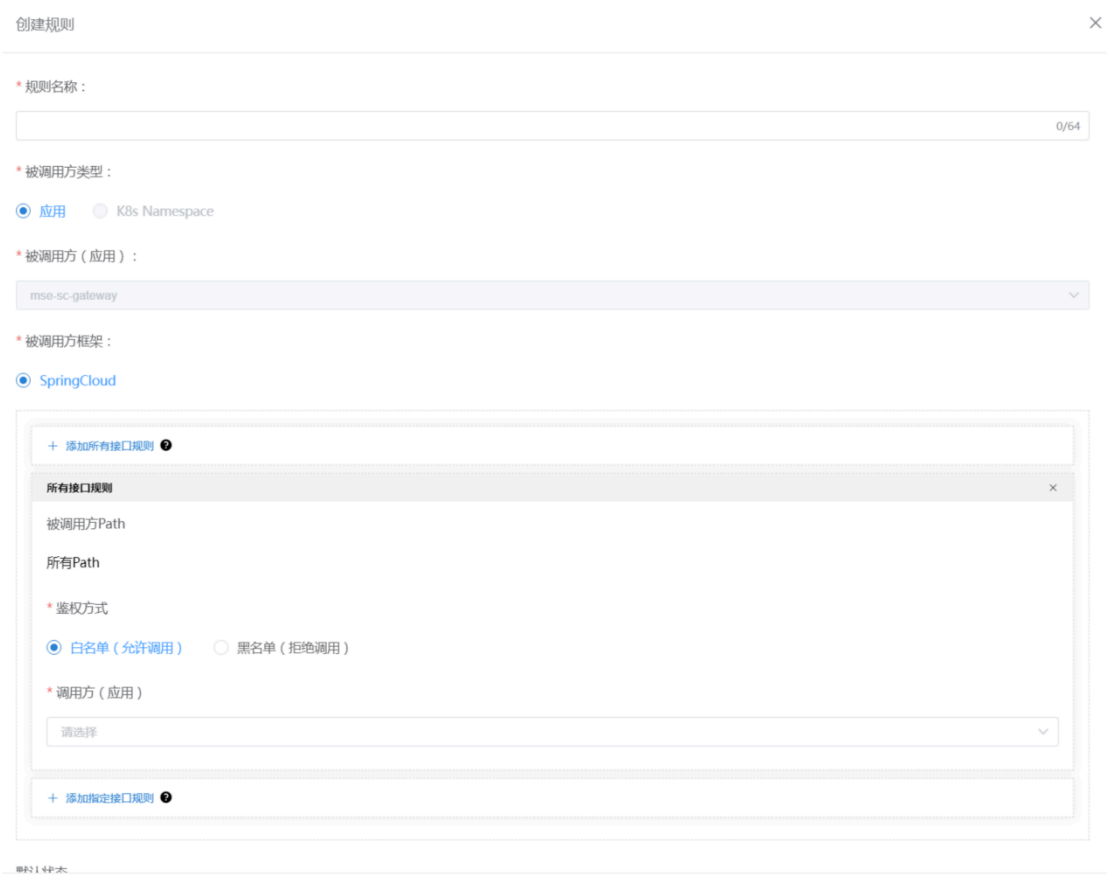
在微服务场景中，若提供者和消费者使用的是同一个注册中心，则消费者默认可以调用同一个注册中心下的提供者服务。服务鉴权可以为提供者的服务设置鉴权规则，允许或拒绝某个消费者访问服务。

版本限制

框架	限制	详情
Spring Cloud	Spring Cloud Dalston及以上版本	客户端：Feign、RestTemplate
Dubbo	2.5.3+	无
jdk版本	1.8+	无

开通服务鉴权

1. 登录微服务治理控制台。
2. 在左侧导航栏选择安全治理，点击服务鉴权。
3. 在服务鉴权页面单击创建服务鉴权。



服务鉴权规则参数说明：

参数	说明
规则名称	服务鉴权规则的名称。
被调用方应用	鉴权规则生效的应用。
被调用方框架	SpringCloud或者Dubo。

接口规则：

鉴权方式	白名单/黑名单
调用方	涉及的调用方。
默认状态	默认开启关闭状态。

运维中心

概述

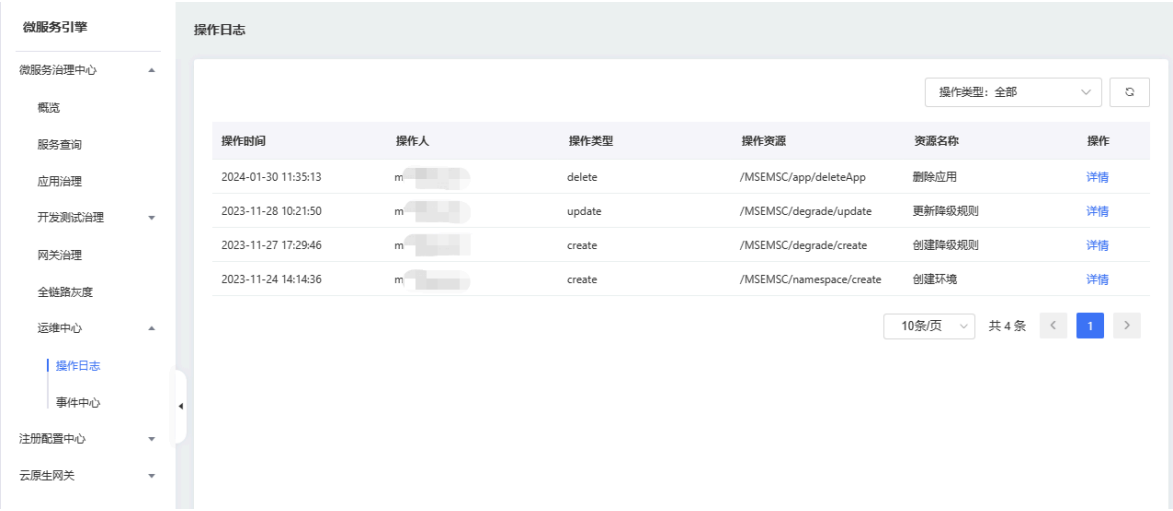
微服务治理中心运维中心提供操作日志及事件中心功能。

操作日志

操作日志提供对治理过程中关键治理操作的记录，通过操作日志功能，用户可以进行行为追踪和问题排查，及时发现治理过程中的不当操作，定位治理操作对应用的影响。

功能入口

- 登录微服务治理控制台。
- 在控制台左侧导航栏中选择运维中心>操作日志。



- 在操作类型下拉框可以选择对应的操作类型，筛选操作。

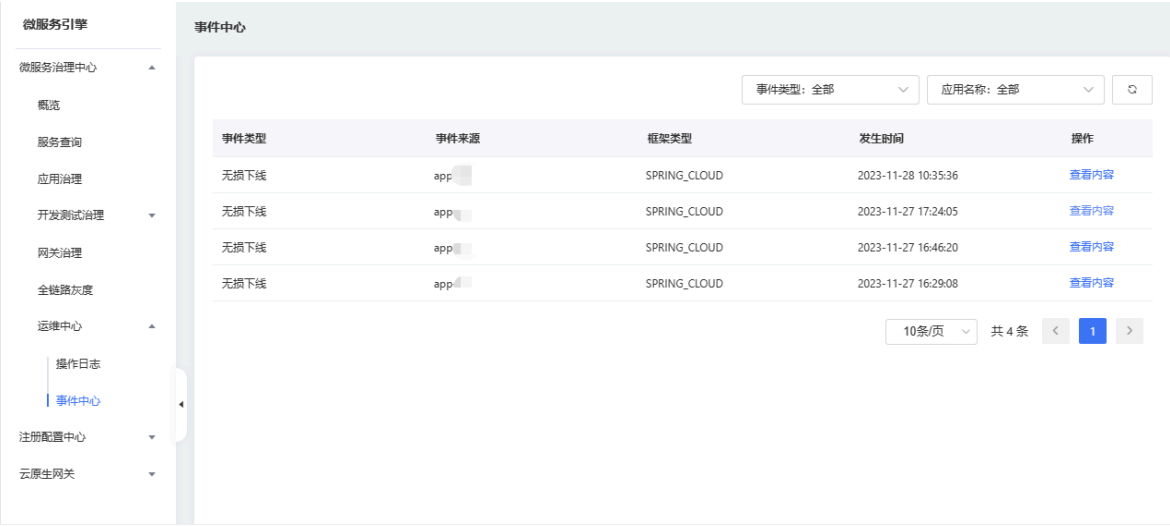
- 在对应日志项的操作列单击详情，获取对应操作的额外信息。

事件中心

事件中心提供对微服务治理事件数据的统一展示能力，通过事件中心，用户可以从事件类型和事件来源维度查询事件记录，感知微服务治理中心下应用的状态，从而验证微服务治理配置的正确性，排查系统的异常问题。

功能入口

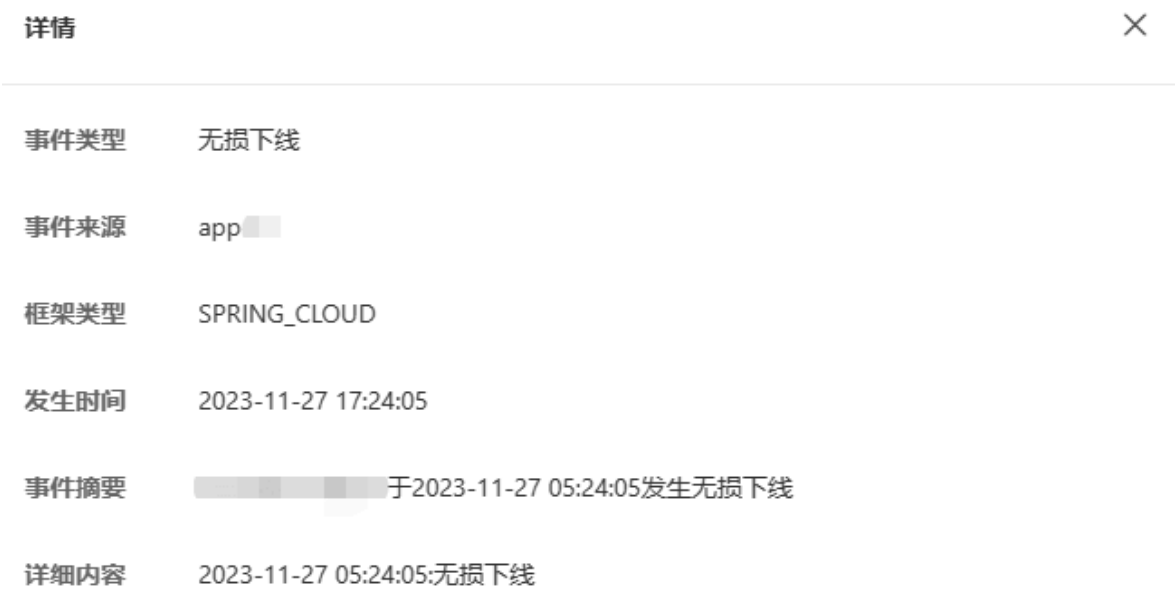
- 登录微服务治理控制台。
- 在控制台左侧导航栏中选择运维中心>事件中心。



- 在事件类型下拉框可以选择对应的事件类型，筛选对应类型的事件记录。

- 在应用名称下拉框可以选择对应的应用资源，筛选对应应用下的事件记录。

- 在对应事件记录项的操作列单击查看详情，获取对应事件记录的详细信息。



事件参数

参数	描述
事件类型	事件的所属类型：无损下线、推空保护、离群实例摘除、离群实例恢复。
事件来源	事件的来源应用名称。
框架类型	应用框架：SpringCloud、Dubbo。
发生时间	事件的触发时间。
事件摘要	事件的摘要描述，包含所在节点、所属应用、事件类型、发生时间等信息。
详细内容	事件的详细描述。

系统管理

权限管理

概述

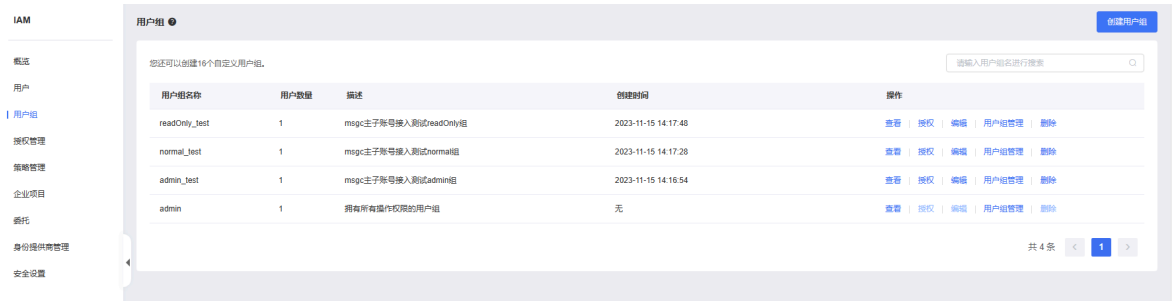
微服务治理中心支持IAM权限管控，通过创建IAM用户并对其分配访问权限，用户可以实现对云服务和资源的访问及操作权限，避免分享账号。

前提条件

已创建IAM用户以及相应用户组，操作详情请参见创建IAM用户。

添加权限

- 使用主账号登录IAM控制台。
- 在控制台左侧导航栏中选择用户组。
- 选择相应的用户组项，在右侧操作列单击授权。

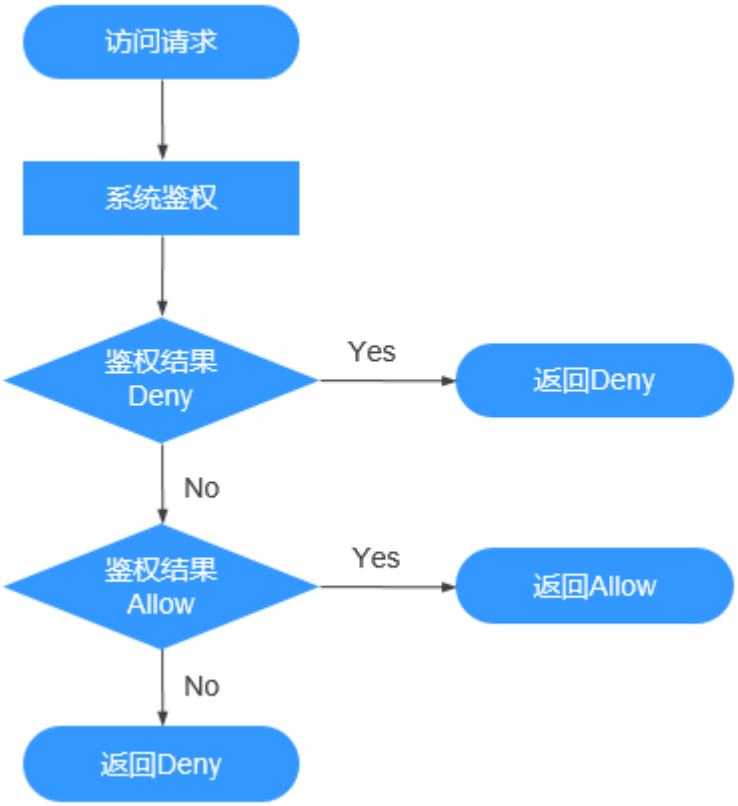


- 在用户组权限管理页面，勾选对应的权限策略，根据提示完成授权。

权限说明

权限策略是灵活的授权项，可以精确到功能和资源两个维度，对于微服务治理中心，控制台功能和应用资源将分别进行权限管控，实现细粒度的访问控制。

多个策略以”or”的关系进行评估，总体遵循Deny优先原则。



权限策略包含系统策略和自定义策略两种。

- 系统策略：微服务治理中心提供三种预置策略。
  - 微服务治理中心查看者：只读权限，拥有控制台信息查询权限。
  - 微服务治理中心使用者：读写权限，拥有除管理微服务治理中心产品和管理应用外的所有权限。
  - 微服务治理中心管理者：管理权限，等同于主账号访问权限，拥有所有功能操作权限。
- 自定义策略：创建自定义策略可以支持更细粒度的授权，对于微服务治理中心，可以自定义功能访问权限以及资源访问权限。具体操作步骤请参考创建自定义策略。
  - 配置示例：

```
{
  "Version": "1.1",
"Statement": [
  // 具有服务鉴权功能权限和所有资源池特定命名空间下所有应用的访问权限
    {
      "Effect": "Allow",
      "Action": [
        "msgc:inst:getAppInfo",
        "msgc:inst:getServiceInfo",
        "msgc:inst:readAuthConfig",
        "msgc:inst:writeAuthConfig"
```

```
        ],
        "Resource": [
          "ctrn:msgc:*:xxxxxxxx:namespace/${namespaceVal}/app/*"
        ]
      }
    ]
}
```

- 资源路径说明：

路径模板 ctrn:msgc:${regionCode}:${accountId}:namespace/${ns}/app/${appId}

参数说明：

参数	描述
regionCode	资源池标识，支持通配符‘*’。
accountId	账户标识，系统自动填充。
ns	命名空间名称，支持通配符‘*’。
appId	应用标识，支持通配符‘*’。

资源标签管理

概述

资源标签是对资源的标识。资源标签在组件之间互通，用户可以实现对资源的统一管理，快速查找到同一特征标签下的资源。微服务治理中心支持用户对应用资源进行标签添加、标签绑定和解绑、标签删除以及标签应用查找。

功能入口

- 登录微服务治理控制台。
- 在控制台左侧栏选择应用治理。
- 在右侧应用卡片页面即可进行资源标签相关操作。

查看应用标签绑定信息

在应用治理页面的应用卡片页面，指针悬浮于应用右上角，查看该应用已绑定的标签。

编辑标签

在应用治理页面的应用卡片页面，指针悬浮于应用右上角，点击添加或编辑按钮，进入标签编辑页面。

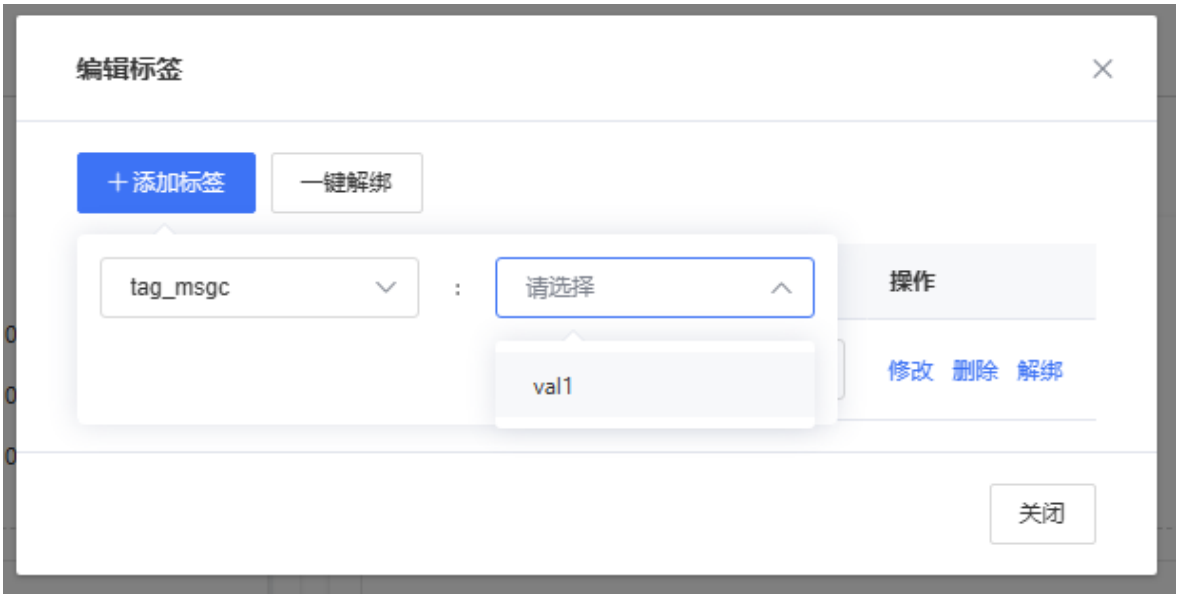
绑定标签

点击添加标签，可以为应用绑定新增标签或已有标签。

- 绑定新增标签：在左侧输入框输入标签键，并选中，然后在右侧输入框输入标签值，点击确定，完成新增标签并为应用绑定此标签。
- 绑定已有标签：在左侧标签键输入框点击，选中已有标签键，在右侧标签值输入框选中标签值，点击确定，为应用绑定选中的标签。

注意

标签一旦添加即绑定应用。同一个应用不可重复添加同一标签，以标签键作区分。



修改标签

在已有标签上，在左侧标签键输入框和右侧标签值输入框完成标签键值的编辑，点击修改，点击确定，完成标签修改。

解绑标签

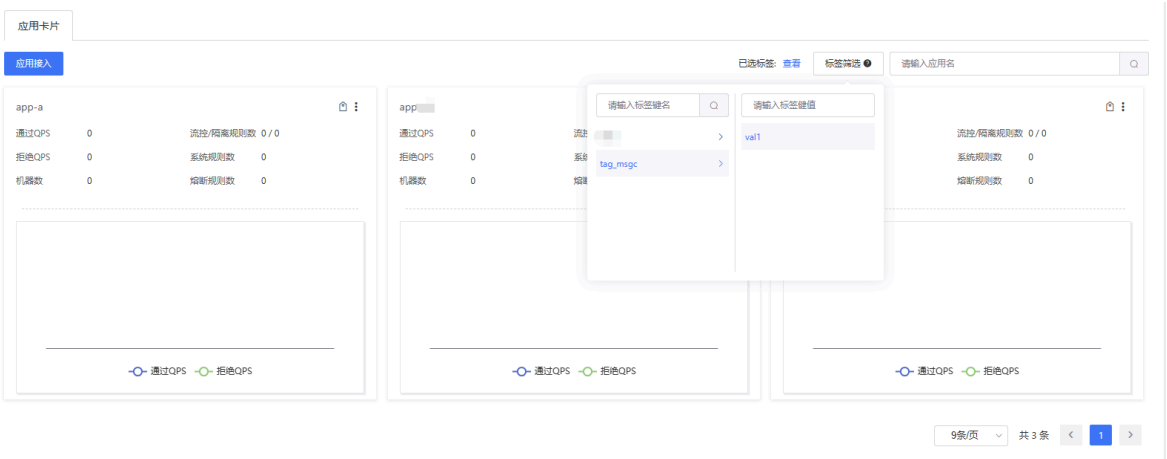
点击解绑并确定，完成对应应用的标签解绑。此操作不会删除被解绑的标签，该标签仍能被其他资源绑定。

删除标签

点击删除并确定，完成对对应标签的删除。此操作将删除该标签，一经删除，该标签对其它应用和其它组件均不可见。

根据标签查找应用

在应用卡片页面右上角，点击标签筛选按钮，可筛选微服务治理中心选中标签下的所有应用。首先选择标签键，再选择标签值，即完成筛选。应用卡片页面将展示已绑定所选标签的应用。



权限管理

概述

如果您需要针对不同资源，对用户设置不同的访问权限，以达到用户之间的权限隔离，您可以使用统一身份认证服务（Identity and Access Management，简称IAM）进行精细的权限管理。该服务提供用户身份认证、权限分配、访问控制等功能，可以帮助您安全地控制云资源的访问。

通过IAM，您可以为其他账号创建IAM用户，并使用策略来控制他们对云资源的访问范围。IAM是云服务提供权限管理的基础服务，无需付费即可使用，您只需要为您账号中的资源进行付费，关于IAM的详细介绍，参见：统一身份认证

如果云账号已经能满足您的要求，不需要创建独立的IAM用户进行权限管理，您可以跳过本章节，不影响您使用微服务引擎MSE的其他功能。

概念

1. 主账号 ：用户在天翼云注册后自动创建，该账号对其所拥有的资源具有完全的访问权限，可以重置用户密码、分配用户权限等。如果需要多人共同使用天翼云资源，由于账号是付费主体为了确保账号安全，建议创建子用户来进行日常管理工作。
2. 子账号 ：主账号认证为企业账号后，在天翼云用户中心页面创建出来的账号。子账号的用户名、密码统一由主账号创建管理。子账号同样可以登录访问天翼云控制台，登录入口与主账号相同，受主账号赋予的权限限制。
3. 企业项目： 将云资源、企业成员按项目进行管理，通过企业项目将云资源、带有权限的用户组绑定到一起，用户使用项目内云资源的权限受用户组的授权限制。

注意

一个实例只能归属一个企业项目（可变更），一个子账号可以同时在多个企业项目中。

4. 策略： 是描述一组权限集的语言，它可以精确地描述被授权的资源集和操作集，通过策略，用户可以自由搭配需要授予的权限集。通过给用户组授予策略，用户组中的用户就能获得策略中定义的权限。策略中可定义“允许”的操作和“拒绝”的操作，“拒绝”的优先级大于“允许”。
5. 系统策略： 系统预置的常用权限集，主要针对不同云服务的只读权限或管理员权限，比如对组件的只读权限、普通用户权限和管理员权限等等；系统策略只能用于授权，不能编辑和修改。
6. 数据权限： 看到的数据不一样。主账号看到所有实例，子账号只能看到所属项目中的实例。
7. 功能权限： 主账号可以进行所有控制台操作，子账号对单个组件实例拥有的操作权限由主账号授权。
8. 功能权限授权： 给子账号在企业项目A下增加一个策略，即代表该子账号对企业项目A下的实例拥有了策略中定义的权限，策略以外的操作会被禁止。默认情况下，新建的IAM用户没有任何权限，您需要将其加入用户组，并给用户组授予策略，才能使得用户组中的用户获得策略定义的权限，这一过程称为授权。具体授权请参考：用户组授权-统一身份认证

微服务引擎MSE系统权限策略

当前微服务引擎MSE各子产品都已提供预置权限策略，包括产品管理员、使用者等角色，不同的角色权限已授予不同策略。具体策略及描述如下：

子产品	权限策略	当前权限策略描述
注册配置中心	微服务引擎-注册配置中心RCC-user	微服务引擎-注册配置中心RCC默认使用者策略，拥有实例的使用权限，无订购、扩缩容、退订等管理权限（适用于一类节点资源池）
注册配置中心	微服务引擎-注册配置中心RCC-admin	微服务引擎-注册配置中心RCC默认管理者策略，拥有所有权限（适用于一类节点资源池）
云原生网关	微服务引擎-云原生网关CGW-admin	微服务引擎-云原生网关CGW默认管理者策略，拥有所有权限（适用于一类节点资源池）
云原生网关	微服务引擎-云原生网关CGW-viewer	微服务引擎-云原生网关CGW默认查看者策略，拥有实例的只读权限（适用于一类节点资源池）
微服务治理中心	微服务引擎-微服务治理中心MSGC-admin	微服务引擎-微服务治理中心MSGC默认管理者策略，对应用拥有管理权限（适用于一类节点资源池）
微服务治理中心	微服务引擎-微服务治理中心MSGC-user	微服务引擎-微服务治理中心MSGC默认使用者策略，对应用拥有治理权限（适用于一类节点资源池）
微服务治理中心	微服务引擎-微服务治理中心MSGC-viewer	微服务引擎-微服务治理中心MSGC默认查看者策略，对应用仅拥有只读权限（适用于一类节点资源池）

微服务引擎MSE全量功能权限

微服务引擎MSE子产品各功能模块均支持单独配置授权，权限策略是灵活的授权项，可以精确到功能和资源维度，实现细粒度的访问控制。具体详情如下：

注册配置中心RCC

归属模块	授权项	权限说明	权限依赖	系统策略		IAM项目	企业项目
				注册配置中心RCC-admin	注册配置中心RCC-user		
产品订购	rcc:inst:create-instance	开通实例	无	✓	✗	✓	✗
	rcc:inst:renew-instance	续订实例	无	✓	✗	✓	✗
	rcc:inst:del-instance	退订实例	无	✓	✗	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略		IAM项目	企业项目
				注册配置中心RCC-admin	注册配置中心RCC-user		
	rcc:inst:extend-instance	实例扩容	无	✓	✗	✓	✗
	rcc:inst:disk-expand	磁盘扩容	无	✓	✗	✓	✗
	rcc:inst:node-expand	节点扩容	无	✓	✗	✓	✗
	rcc:inst:spec-expand	规格扩容	无	✓	✗	✓	✗
	rcc:inst:instance-billmode	包周期按需互转	无	✓	✗	✓	✗
集群管理	rcc:inst:instance-list	实例列表	无	✓	✓	✓	✗
	rcc:inst:cluster_monitor_read	查询集群监控信息	无	✓	✓	✓	✗
	rcc:inst:cluster_param_manage	集群参数管理	无	✓	✓	✓	✗
	rcc:inst:cluster_risk_manage	集群风险管理	无	✓	✓	✓	✗
	rcc:inst:cluster_version_manage	集群版本管理	无	✓	✓	✓	✗
ELB绑定信息管理	rcc:inst:elb_bind_manager	ELB绑定管理	无	✓	✓	✓	✗
	rcc:inst:elb_info_read	ELB绑定信息查询	无	✓	✓	✓	✗
迁移上云	rcc:inst:migration_cluster_manage	迁移集群管理	无	✓	✓	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略		IAM项目	企业项目
				注册配置中心RCC-admin	注册配置中心RCC-user		
	rcc:inst:migration_task_manage	迁移任务管理	无	✓	✓	✓	✗
	rcc:inst:migration_tool_manage	迁移上云工具管理	无	✓	✓	✓	✗
Nacos	rcc:inst:nacos_service_manage	Nacos服务管理	无	✓	✓	✓	✗
	rcc:inst:nacos_namespace_manage	Nacos命名空间管理	无	✓	✓	✓	✗
	rcc:inst:nacos_aksk_manage	Nacos认证管理	无	✓	✓	✓	✗
	rcc:inst:nacos_user_manage	Nacos用户管理	无	✓	✓	✓	✗
	rcc:inst:nacos_role_manage	Nacos角色管理	无	✓	✓	✓	✗
	rcc:inst:nacos_perm_manage	Nacos用户权限管理	无	✓	✓	✓	✗
	rcc:inst:nacos_property_manage	Nacos黑白名单管理	无	✓	✓	✓	✗
	rcc:inst:nacos_config_manage	Nacos配置管理	无	✓	✓	✓	✗
Zookeeper	rcc:inst:zk_service_manage	Zookeeper服务管理	无	✓	✓	✓	✗
	rcc:inst:zk_znode_manage	Zookeeper数据节点管理	无	✓	✓	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略		IAM项目	企业项目
				注册配置中心RCC-admin	注册配置中心RCC-user		
	rcc:inst:zk_blackandwhite_list	Zookeeper黑白名单管理	无	✓	✓	✓	✗
	rcc:inst:zk_track_manage	Zookeeper数据轨迹	无	✓	✓	✓	✗
Eureka	rcc:inst:eureka_service_manage	Eureka服务管理	无	✓	✓	✓	✗
	rcc:inst:eureka_track_manage	Eureka数据轨迹	无	✓	✓	✓	✗

云原生网关CGW

归属模块	授权项	权限说明	权限依赖	系统策略		IAM项目	企业项目
				云原生网关CGW-admin	云原生网关CGW-viewer		
产品订购	cgw:inst:create-instance	产品开通	无	✓	✗	✓	✗
	cgw:inst:del-instance	产品退订	无	✓	✗	✓	✗
	cgw:inst:extend-instance	产品扩容	无	✓	✗	✓	✗
	cgw:inst:instance-billmode	包周期按需互转	无	✓	✗	✓	✗
	cgw:inst:renew-instance	产品续订	无	✓	✗	✓	✗
网关实例管理	cgw:inst:getSpuInstInfo	查询实例相关信息	无	✓	✓	✓	✗
	cgw:inst:instance-list	查询网关实例列表	无	✓	✓	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略		IAM项目	企业项目
				云原生网关 CGW-admin	云原生网关 CGW-viewer		
	cgw:inst:updateSpuInst	更新实例信息	无	✓	✗	✓	✗
网关配置管理	cgw:inst:updateTraceAnalysis	更新链路分析配置	无	✓	✗	✓	✗
	cgw:inst:getTraceAnalysis	获取链路分析配置	无	✓	✓	✓	✗
ELB管理	cgw:inst:getElb	查询ELB相关信息	无	✓	✓	✓	✗
	cgw:inst:updateElb	修改ELB相关信息	无	✓	✗	✓	✗
服务来源管理	cgw:inst:getUpstreamSource	查询服务来源	无	✓	✓	✓	✗
	cgw:inst:updateUpstreamSource	更新服务来源	无	✓	✗	✓	✗
	cgw:inst:getNacosInfo	查询Nacos相关信息	无	✓	✓	✓	✗
	cgw:inst:getK8s	查询K8s相关信息	无	✓	✓	✓	✗
服务管理	cgw:inst:getUpstreamInfo	查询服务相关信息	无	✓	✓	✓	✗
	cgw:inst:updateUpstream	更新服务	无	✓	✗	✓	✗
	cgw:inst:getUpstreamVersions	获取服务版本信息	无	✓	✓	✓	✗
	cgw:inst:updateUpstreamVersions	更新服务版本	无	✓	✗	✓	✗
域名管理	cgw:inst:getDomainsInfo	查询域名相关信息	无	✓	✓	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略		IAM项目	企业项目
				云原生网关 CGW-admin	云原生网关 CGW-viewer		
	cgw:inst:updateDomains	修改域名	无	✓	✗	✓	✗
路由配置	cgw:inst:getRoutesInfo	查询路由相关信息	无	✓	✓	✓	✗
	cgw:inst:updateRoutes	修改路由	无	✓	✗	✓	✗
	cgw:inst:getRouteSnapshotInfo	查询路由快照相关信息	无	✓	✓	✓	✗
	cgw:inst:updateRouteSnapshot	修改路由快照	无	✓	✗	✓	✗
安全认证	cgw:inst:getAuthInfo	查询认证相关信息	无	✓	✓	✓	✗
	cgw:inst:updateAuth	修改认证	无	✓	✗	✓	✗
	cgw:inst:getBlackWhiteListInfo	查询黑白名单相关信息	无	✓	✓	✓	✗
	cgw:inst:updateBlackWhiteList	修改黑白名单	无	✓	✗	✓	✗
观测分析	cgw:inst:getLogCenterInfo	查询日志相关信息	无	✓	✓	✓	✗
	cgw:inst:updateLogCenter	修改日志	无	✓	✗	✓	✗
	cgw:inst:getMonitorInfo	查询监控相关信息	无	✓	✓	✓	✗
路由级插件	cgw:inst:getPluginConfigurationInfo	获取路由级插件相关信息	无	✓	✓	✓	✗
	cgw:inst:updatePluginConfiguration	修改路由级插件	无	✓	✗	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略		IAM项目	企业项目
				云原生网关 CGW-admin	云原生网关 CGW-viewer		
全局插件	cgw:inst:getPluginsInfo	获取全局插件相关信息	无	✓	✓	✓	✗
	cgw:inst:updatePlugins	修改全局插件	无	✓	✗	✓	✗
应用管理	cgw:inst:getAppManageInfo	查询应用管理相关信息	无	✓	✓	✓	✗
	cgw:inst:updateAppManage	修改应用管理	无	✓	✗	✓	✗
API托管	cgw:inst:getApiGwServiceInfo	查询API托管服务管理相关信息	无	✓	✓	✓	✗
	cgw:inst:updateApiGwService	修改API托管服务管理	无	✓	✗	✓	✗
	cgw:inst:getApiGwGroupInfo	查询API托管分组管理相关信息	无	✓	✓	✓	✗
	cgw:inst:updateApiGwGroup	修改API托管分组管理	无	✓	✗	✓	✗
	cgw:inst:getApiGwApiInfo	查询API托管API管理相关信息	无	✓	✓	✓	✗
	cgw:inst:updateApiGwApi	修改API托管API管理	无	✓	✗	✓	✗

微服务治理中心MSGC

归属模块	授权项	权限说明	权限依赖	系统策略			IAM项目	企业项目
				微服务治理中心MSGC-admin	微服务治理中心MSGC-user	微服务治理中心MSGC-viewer		
产品订购	msgc:inst:create-instance	产品开通	无	✓	✗	✗	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略			IAM项目	企业项目
				微服务治理中心MSGC-admin	微服务治理中心MSGC-user	微服务治理中心MSGC-viewer		
	msgc:inst:del-instance	产品退订	无	✓	✗	✗	✓	✗
	msgc:inst:extend-instance	产品扩容	无	✓	✗	✗	✓	✗
	msgc:inst:renew-instance	产品续订	无	✓	✗	✗	✓	✗
应用管理	msgc:inst:getAppInfo	查询应用相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:getOverview	查询应用概览信息	无	✓	✓	✓	✓	✗
	msgc:inst:manageApp	管理应用	无	✓	✗	✗	✓	✗
服务监控	msgc:inst:getResourceInfo	查询接口相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:getServiceInfo	查询服务相关信息	无	✓	✓	✓	✓	✗
服务配置	msgc:inst:writeDubboConfig	修改Dubbo相关配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeNodeConfig	修改节点相关配置	无	✓	✓	✗	✓	✗
流量治理	msgc:inst:readAdaptiveFlowConfig	查询自适应流控相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readDegradeConfig	查询服务降级相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readHotParamRule	读取热点规则相关信息	无	✓	✓	✓	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略			IAM项目	企业项目
				微服务治理中心MSGC-admin	微服务治理中心MSGC-user	微服务治理中心MSGC-viewer		
	msgc:inst:readLosslessInfo	查询无损上下线相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readMqGrayRouteConfig	查询消息灰度相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readOutlierConfig	查询离群实例摘除相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readPushProtection	查询推空保护相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readTagInfo	查询流量标签相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:writeAdaptiveFlowConfig	修改自适应流控配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeDegradeConfig	修改服务降级配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeHotParamRule	修改热点规则	无	✓	✓	✗	✓	✗
	msgc:inst:writeLosslessConfig	修改无损上下限配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeMqGrayRouteConfig	修改消息灰度配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeOutlierConfig	修改离群摘除配置	无	✓	✓	✗	✓	✗
	msgc:inst:writePushProtection	修改推空保护配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeSwitchConfig	修改功能开关配置	无	✓	✓	✗	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略			IAM项目	企业项目
				微服务治理中心MSGC-admin	微服务治理中心MSGC-user	微服务治理中心MSGC-viewer		
	msgc:inst:writeTagCanaryConfig	修改金丝雀配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeTagRouteConfig	修改标签路由配置	无	✓	✓	✗	✓	✗
流量防护	msgc:inst:readCircuitBreakConfig	查看服务熔断相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readClientConfig	查询流量防护基础设置	无	✓	✓	✓	✓	✗
	msgc:inst:readClusterInfo	查询集群防护配置	无	✓	✓	✓	✓	✗
	msgc:inst:readFallbackBehavior	查询防护行为相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readFlowConfig	查询流控隔离相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readManualDegradeConfig	查询主动降级相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readWebFlowConfig	查询WEB防护相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:writeCircuitBreakConfig	修改服务熔断配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeClientConfig	修改流量防护基础配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeClusterInfo	修改集群防护配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeFallbackBehavior	修改防护行为	无	✓	✓	✗	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略			IAM项目	企业项目
				微服务治理中心MSGC-admin	微服务治理中心MSGC-user	微服务治理中心MSGC-viewer		
	msgc:inst:writeFlowConfig	修改流控隔离配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeManualDegradeConfig	修改主动降级配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeWebFlowConfig	修改WEB防护配置	无	✓	✓	✗	✓	✗
数据库治理	msgc:inst:readDatabaseGrayConfig	查询数据库灰度相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readDatabaseReadWriteConfig	查询数据库读写路由相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readDataSourceInfo	查询连接池相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:writeDatabaseGrayConfig	修改数据库灰度配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeDatabaseReadWriteConfig	修改数据库读写路由配置	无	✓	✓	✗	✓	✗
	msgc:inst:writeDataSourceConfig	修改连接池配置	无	✓	✓	✗	✓	✗
功能开关	msgc:inst:getSwitchInfo	查询功能开关信息	无	✓	✓	✓	✓	✗
	msgc:inst:getSwitchLog	查询功能开关日志	无	✓	✓	✓	✓	✗
开发测试治理	msgc:inst:readMockConfig	查询服务Mock相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:writeMockConfig	修改服务Mock配置	无	✓	✓	✗	✓	✗

归属模块	授权项	权限说明	权限依赖	系统策略			IAM项目	企业项目
				微服务治理中心MSGC-admin	微服务治理中心MSGC-user	微服务治理中心MSGC-viewer		
全链路灰度	msgc:inst:readSwimmingGroup	查询泳道组相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:readSwimmingLane	查询泳道相关信息	无	✓	✓	✓	✓	✗
	msgc:inst:writeSwimmingGroup	修改泳道组	无	✓	✓	✗	✓	✗
	msgc:inst:writeSwimmingLane	修改泳道	无	✓	✓	✗	✓	✗
网关治理	msgc:inst:readGatewayApiDef	查询网关API信息	无	✓	✓	✓	✓	✗
	msgc:inst:writeGatewayApiDef	修改网关API配置	无	✓	✓	✗	✓	✗
运维中心	msgc:inst:getEvent	查询应用事件信息	无	✓	✓	✓	✓	✗
	msgc:inst:getOperLog	查询操作日志	无	✓	✓	✓	✓	✗

注册配置中心

应用迁移

注册中心迁移

MSE Nacos Sync迁移方案

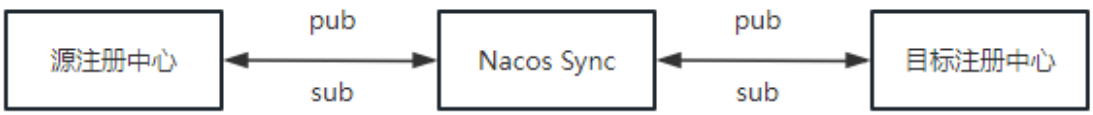
本文介绍基于Nacos Sync 注册中心的迁移方案。

迁移方案简介

Nacos sync 是一个支持多种注册中心的同步组件，目前支持Nacos 和Zookeeper 注册中心 的双向同步，另外还支持Eureka 和Consul数据同步到Nacos。

迁移工具介绍

在迁移过程中，Nacos sync能够将源集群上的服务信息同步到目标集群实例上，从而实现配置中心之间的平滑迁移。



Nacos Sync 本身是无状态的，将任务等状态数据保存在数据库中，所以水平扩展非常方便。Nacos Sync 通过定时任务补偿机制处理宕机节点未处理完毕的任务数据。注册配置中心实例已经内置同步工具，无需用户额外安装，迁移方法可以参考章节：迁移上云。

Nacos Sync适配Zookeeper、Nacos和Eureka 的服务注册逻辑，能够实现从Zookeeper、Nacos和Eureka 自动获取服务，一键同步，操作非常简单。

支持的注册中心类型

源注册中心	目标注册中心	说明
Nacos	Nacos	Nacos原生服务类型。
Zookeeper	Nacos	基于Curator实现的服务发现功能和Dubbo服务。
Zookeeper	Zookeeper	服务和配置。
Eureka	Nacos	Eureka原生服务类型。Eureka原生服务名为大写，但Nacos Sync会将服务名名转换成小写。

从Dubbo Nacos迁移到MSE

本文介绍如何从Dubbo+Nacos 迁移服务到MSE。

前提条件

1. 已经创建Nacos 实例。

使用限制

1. 源注册中心、目标注册中心二者之间必须保障网络互通。

迁移步骤

迁移步骤请参考章节：迁移上云。

验证同步结果

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心 > 实例列表。
3. 在实例列表页面，单击目标实例ID、实例名称或者目标行管理按钮均可跳转至实例基础信息页面。
4. 在基础信息页面，点击服务管理> 服务列表，选择命名空间，查看当前Nacos注册的服务列表。确认选择的服务是否同步成功。

迁移Dubbo客户端

1. 修改Dubbo客户端的XML配置文件，将dubbo:registry address中的源Nacos的访问地址，替换为目标Nacos的访问地址，替换Nacos用户名和密码。
2. 重启客户端，此时Dubbo客户端连接的就是目标Nacos实例。
3. 此时源Nacos集群就可以关闭了。

到此，即实现了从源注册中心到目标注册中心的平滑迁移。

从Dubbo ZooKeeper迁移到 MSE ZooKeeper

本文迁移指导适用于使用ZooKeeper作为Dubbo的注册中心需要迁移到MSE ZooKeeper的场景。迁移功能是利用ZooKeeper的快照（Snapshot）进行数据的迁移。

前提条件：

已创建MSE ZooKeeper集群。具体操作，请参考创建ZooKeeper引擎。

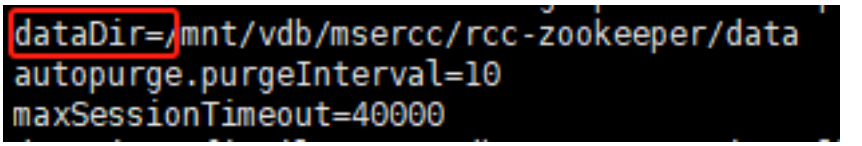
使用限制：

1. 迁移前确保源ZooKeeper停止同步服务，并且有生成最新的快照文件。
2. 该方式不支持同步分布式锁、临时节点数据。

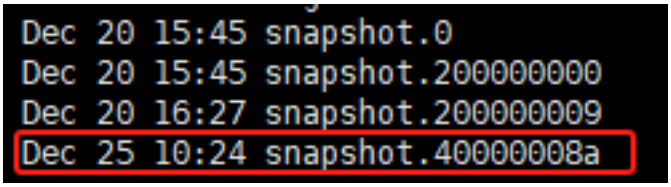
迁移步骤：

步骤一：导出源ZooKeeper快照。

1. 在源ZooKeeper部署机器上，查看zoo.cfg配置文件，获取dataDir配置目录。



2. 进入dataDir目录后，进入version-2/目录找到最新的快照文件进行下载保存。



3. 在MSE ZooKeeper实例详情页面中，进入数据管理->Znode管理->点击数据导入。
4. 点击上传文件，选中刚刚下载保存的快照文件，点击确定，等待集群重启完毕即可，预计需要等待2-5分钟。
5. 验证数据迁移结果，可以在数据管理-Znode管理中查看相应的节点验证数据。
6. 迁移Dubbo客户端，找到Dubbo客户端的配置文件，将Dubbo客户端的Endpoint替换为MSE ZooKeeper的Endpoint。修改示例如下：

```
dubbo.registry.address=zookeeper://xxx.xx.xx.x:port
```

配置中心迁移

原生Nacos迁移至MSE

MSE Nacos相对于开源Nacos额外提供了配置加密、推送轨迹等功能，保障高可用，并提供了丰富的运维工具，操作十分方便。本节介绍从开源Nacos迁移到天翼云Nacos实例。

前提条件

1. 已经创建Nacos实例，参考章节：创建Nacos实例。
2. 已经在Nacos实例上创建需要迁移的命名空间。

迁移配置

1. 在开源控制台导出需要迁移的配置。

- 登陆开源Nacos自带的原生控制台。
- 在配置列表页面选择命名空间，点击单选框选中需要迁移的配置。
- 在配置列表下方点击导出，然后选择导出选中的配置。
- 然后在下载中可以看到导出的压缩包文件nacos_config_exportxxxxx.zip。

2. 在MSE控制台页面导入配置。

- 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
- 在左侧导航栏，选择注册配置中心 > 实例列表。
- 在实例列表页面，单击目标实例ID、实例名称或者目标行“管理”按钮均可跳转至实例基础信息页面。
- 在基础信息页面，点击配置管理>配置列表，选择命名空间。
- 点击导入配置，在弹出的导入配置框中确认目标命名空间，选择相同配置的处理策略（策略详细介绍见同步配置），点击上传配置文件，最后点击确定，即可开始导入文件。
- 导入后查看配置列表，可以看到配置已经导入成功。

3. 在应用完成迁移以前，如果需要变更配置文件，则需要在两边同步更新，以避免业务出现不一致的情况。

修改业务配置参数

根据不同的应用和框架，需要修改的配置参数不完全相同。但总体来说最重要的四个字段是Nacos服务端访问地址，命名空间ID，用户名以及密码。

迁移Java应用

Java应用一般情况下是直接依赖nacos-client。以pom.xml依赖文件为例：

```
<dependency>
 <groupId>com.alibaba.nacos</groupId>
 <artifactId>nacos-client</artifactId>
  <version>${nacos.version]</version>
</dependency>
```

如果是在代码中直接使用properties方式配置Nacos访问信息，则需要修改。

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.SERVER_ADDR, "Nacos访问地址");
properties.put(PropertyKeyConst.NAMESPACE, “命名空间");
properties.put(PropertyKeyConst.USERNAME, "username);
properties.put(PropertyKeyConst.PASSWORD, “password");
```

迁移Spring Boot 应用

Spring boot应用一般情况下Nacos的相关配置在application.properties文件中。

同样是修改Nacos服务端访问地址，命名空间ID，用户名以及密码等字段。

```
properties.put(PropertyKeyConst.SERVER_ADDR, "Nacos访问地址");
nacos.config.server-addr=${Nacos的访问地址}
nacos.config.namespace=${命名空间ID}
nacos.config.username=${用户名}
nacos.config.password=${密码}
```

迁移spring-cloud应用

Spring cloud应用一般情况下存在如下依赖：

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
  <version>${spring-cloud-starter-alibaba.version}</version>
</dependency>
```

一般情况下Nacos配置文件包含在bootstrap.properties 或者bootstrap.yml文件中。需要修改的配置项如下所示：

```
spring:
 cloud:
  nacos:
   config:
    server-addr: ${_NACOS_SERVER_ADDRESS}
    username: ${_NAOCS_USERNAME}
    password: ${_NAOCS_PASSWORD}
    namespace: ${_NACOS_CONFIG_NAMESPACE}
```

源配置中心下线

当配置迁移完毕且应用已经全部切换至目标配置中心后，可以停止更新源配置中心，确认没有连接或监听的情况下，可以将源注册中心停止，完全使用目标配置中心。

从Apollo迁移至MSE Nacos

本节内容介绍如何从Apollo迁移配置至MSE Nacos实例。

前提条件

1. 已经创建Nacos实例，参考章节创建Nacos实例。
2. 已经在Nacos实例上创建需要迁移的命名空间。

同步配置信息

1. 登录原生的Apollo控制台，本文以Apollo官方demo地址为例。
2. 在我的应用页面点击需要迁移配置的应用名称。
3. 在右上角点击管理员工具> 配置导入导出 > 点击单选框选中选择需要导出的环境，然后点击导出按钮，这种方式导出的是Apollo该环境的所有配置，所以是一个压缩包，解压后可以查看其中包含的properties文件。
4. 按照Apollo和Nacos数据结构的对应关系,进入到对应的环境目录，可以看到导出的配置形如： appId+集群名+配置名称.properties。Apollo和Nacos数据结构对应关系如下：

Apollo数据结构	Nacos数据结构
环境	Namespace
集群	group
Namespace	dataId

5. 进入注册配置中心Nacos引擎控制台>配置管理>配置列表页面，点击配置导入按钮，在弹出的对话框中选择Apollo配置导入，然后选择导入策略，选择待导入的properties文件并提交，其中导入策略有两种：

   （1）跳过导入：导入过程中发现存在与待导入配置dataId和group相同的配置存则跳过；

   （2）覆盖导入：导入过程中发现存在与待导入配置dataId和group相同的配置存则覆盖其内容。

6. 执行完毕后，查看Nacos对应命名空间配置是否同步成功。

修改应用配置

1. 修改依赖信息

将apollo-client客户端依赖改为nacos-client依赖，如下所示：

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
  <version>2.2.9.RELEASE</version>
</dependency>
```

2. 修改代码

(1) 如果迁移的应用代码中使用了@EnableApolloConfig注解，需要将所有@EnableApolloConfig注解的类替换为@ConfigurationProperties注解。

(2) 如果迁移的应用代码中中使用了@ApolloConfig注解，将其修改为@ConfigurationProperties(prefix = "")注解，并且在bootstrap.properties中增加相关配置项。

(3) 使用的@ApolloConfigChangeListener可以使用nacos-sdk 中的中注册监听器代替。

(4) 代码中使用的@ApolloJsonValue 可以使用@Value替代，对应的 JsonBean类型使用String代替。

3. **修改配置信息**

将Apollo相关配置改为MSE Nacos相关的配置，实例如下:

application.yml文件中的Apollo配置:

```
apollo:
 meta: http://81.68.181.139:8080
 bootstrap:
  enabled: true
  eagerLoad:
   enabled: true
  namespaces: application,TEST1.C_DEMO
listeners: "application,TEST1.C_DEMO"
```

bootstrap.yml文件中的Nacos配置：

```
spring:
 application:
  name: TEST1.C_DEMO
 cloud:
  nacos:
   config:
    server-addr: Nacos访问地址
    namespace: prod
```

4. **重启服务。**

5. **修改配置验证监听情况**。修改配置后，可以在应用的INFO日志中看到INFO o.s.c.e.event.RefreshEventListener - Refresh keys changed:[] 等字样，说明应用监听配置正常。

Zookeeper持久化数据的迁移

数据迁移功能是利用ZooKeeper的快照（Snapshot）进行持久化数据的迁移。

前提条件

已创建MSE ZooKeeper集群。具体操作，请参考创建ZooKeeper引擎。

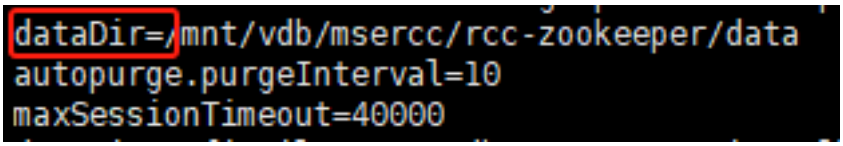
使用限制

1. 迁移前确保源ZooKeeper停止同步服务，并且有生成最新的快照文件。
2. 该方式不支持同步分布式锁、临时节点数据。

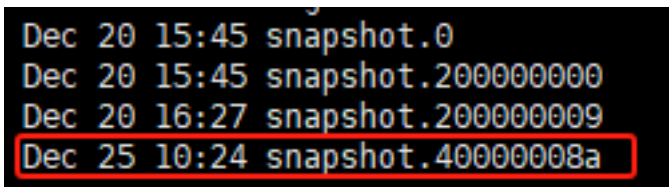
迁移步骤

步骤一：导出源ZooKeeper快照

1. 在源ZooKeeper部署机器上，查看zoo.cfg配置文件，获取dataDir配置目录。



2. 进入dataDir目录后，进入version-2/目录找到最新的快照文件进行下载保存。



3. 在MSE ZooKeeper实例详情页面中，进入数据管理->Znode管理->点击数据导入。
4. 点击上传文件，选中刚刚下载保存的快照文件，点击确定，等待集群重启完毕即可，预计需要等待2-5分钟。
5. 验证数据迁移结果，可以在Znode管理中查看相应的节点数据验证。

应用开发

注册配置中心高可用最佳实践

概述

对于注册配置中心而言，高可用是非常关键的。注册配置中心是微服务体系中的核心依赖，当注册配置中心不可用时，整个微服务体系都将无法对外提供服务。

本文将介绍注册配置中心Nacos引擎的高可用最佳实践。遵照下述设置，将发挥注册配置中心最大的高可用能力。

版本推荐

spring-cloud：推荐使用2.2.6.RELEASE及以上版本。

dubbo：推荐使用2.7.12及以上版本。

spring-boot：推荐使用2.3.x及以下版本，2.4.x版本存在兼容问题，不推荐使用。

集群高可用

实例在开通时将提供以下选项，遵照这些选项进行设置，将提升集群的高可用能力。

1. 多节点：提供多节点集群的选项。对于Nacos引擎而言，节点数为奇数（3，5，7，9）。集群节点数越多，可用性越高。
2. 跨可用区：对于多节点的Nacos集群，可提供跨可用区部署。使用跨可用区部署时，节点的故障发生事件将相互独立，集群可用性将进一步提高。

注意

您需要在开通实例时指定可用区模式为“多可用区部署”，才能触发跨可用区高可用能力。

此外，注册配置中心实例将提供额外的手段以提高集群的可用性，这些手段是实例自动附带的，默认为开启状态。这些手段包括：

1. 服务进程保活。-Nacos引擎提供了进程保活机制对机器中的Nacos进程进行探测和自动拉起。
2. 典型异常故障自愈。
3. 服务可用性实时探活，极端情况下，不可用时自动告警，人工将介入恢复，保障可用性。

注册中心高可用

Nacos引擎提供了以下服务端机制来提高注册中心的高可用能力，这些能力默认为开启状态，您无需开启即可获得高可用能力提升：

1. 服务端推空保护。当Nacos服务端在一定时间窗口检测到的不健康服务增量大于阈值时，服务端将进入推空保护模式，向客户端推送服务的旧数据，而不是空服务，以保障服务消费者不会获取到空服务信息，从而提高可用性。
2. 客户端心跳上报，服务端定期驱逐不健康服务，以保证服务消费者获取到的服务数据的实时性。

配置中心高可用

开源Nacos提供了以下客户端机制来提高配置中心的高可用能力，这些能力默认为开启状态，您无需开启即可获得高可用能力提升：

配置本地缓存。当配置中心服务不可用时，客户端将从本地缓存中获取上一次成功获取到的配置数据。

如何在MSE上为Spring Cloud应用构建服务注册中心？

版本和依赖

SpringBoot、SpringCloud Alibaba、OpenFeign等存在版本对应关系，版本不匹配可能会出现问题。以如下以来的版本为例说明如何接入Nacos。

```
<dependencies>
    <dependency>
       <!--配置中心依赖-->
      <groupId>com.alibaba.cloud</groupId>
      <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
      <exclusions>
        <exclusion>
          <groupId>com.alibaba.nacos</groupId>
          <artifactId>nacos-client</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
   <!--客户端版本依赖-->
    <dependency>
      <groupId>com.alibaba.nacos</groupId>
      <artifactId>nacos-client</artifactId>
      <version>2.1.0</version>
    </dependency>
     <!--注册中心版本依赖-->
    <dependency>
      <groupId>com.alibaba.cloud</groupId>
      <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    <dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
      </dependency>
  <dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-openfeign</artifactId>
   <version>3.0.6</version>
</dependency>
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-loadbalancer</artifactId>
   <version>3.0.6</version>
</dependency>
      <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-bootstrap</artifactId>
        <version>3.0.6</version>
      </dependency>
</dependencies>
```

接入注册中心
服务注册

接入注册中需要的增加的依赖是spring-cloud-starter-alibaba-nacos-discovery，在第二节中已经给出。

注意

实例中SpringBoot 2.6.1 SpringCloud版本为2021.0.4.0，这两个版本需要匹配。如果使用其他版本，可以从Spring Cloud 官方查询匹配版本。

在本地创建服务提供者应用工程，然后添加依赖，并开启服务注册与发现功能，并将注册中心指定为Nacos Server。 接入注册中心需要在配置文件中增加相关配置，以yml文件为例，可以加在application-prod.yml中增加如下配置文件：

```
spring:
 cloud:
  nacos:
    discovery:
      username: ${_NAOCS_USERNAME}
      password: ${_NACOS_PASSWORD}
      server-addr: ${_NACOS_SERVER_ADDRESS}
      namespace: ${_NACOS_NAMING_NAMESPACE}
      group: ${_NACOS_NAMING_GROUP}
```

以上配置中变量的实际值，可以通过环境变量的方式提供。环境变量可以在云容器引擎中配置。

除修改配置文件以外，需要在Spring Boot 项目的启动类上增加@EnableDiscoveryClient注解。如下列代码所示：

```
@EnableDiscoveryClient
@SpringBootApplication
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class,args);
   }
}
```

修改配置文件后启动应用。启动成功后可以在控制台页面查看当前注册的服务。

服务发现

除了服务注册以外，注册中心也提供服务发现的功能。Nacos服务发现可以使用RestTemplate和FeignClient两个客户端来调用注册的服务。如下以使用FeignClient调用为例：

1. 调用方增加依赖：

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-openfeign</artifactId>
   <version>3.0.6</version>
</dependency>
```

2. 在 Spring Boot 项目的启动文件上添加 @EnableFeignClients 注解，开启 OpenFeign，具体实现代码如下：

```
@EnableFeignClients
@EnableDiscoveryClient
@SpringBootApplication
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class,args);
   }
}
```

3. 最重要的一步，创建OpenFeign与服务提供者的调用接口，具体实现代码如下：(示例这里调用的是自己的接口)

```
@FeignClient(name = "demo-test")
public interface OrderFeignService {

   @GetMapping("/call/{name}")
   String call(@PathVariable(value = "name") String name);
}
```

4. 然后在controller中访问接口，就可以获得通过feign请求的的结果。

```
@RestController
@RequestMapping("feign")
public class DiscoverController {
```

```
    @Resource
    private IFeignClientService iFeignClientService;

    @GetMapping("/echo/{message}")
    public String getMessage(@PathVariable String message) {
      return iFeignClientService.call(message);
    }

}
```

5. 最后请求feign/echo/{message}接口即可通过feign调call接口了。

至此通过FeignClient调用服务接口的流程就实现了。

接入配置中心

接入配置中需要的增加的依赖是spring-cloud-starter-alibaba-nacos-config，在第二节中已经给出。

接入配置中心需要在配置文件中增加相关配置，以yml文件为例，需要在bootstrap.yml中增加如下配置信息：

```
spring:
 cloud:
  nacos:
   # config center
   config:
    username: ${_NAOCS_USERNAME}
    password: ${_NAOCS_PASSWORD}
    server-addr: ${_NACOS_SERVER_ADDRESS}
    namespace: ${_NACOS_CONFIG_NAMESPACE}
    group: ${_NACOS_CONFIG_GROUP}
    prefix:
```

其中 namespace 默认值为空，也就是public命名空间，group 默认值为DEFAULT_GROUP， prefix属性为非必要属性，可以按照需要决定是否配置。配置文件中的值既可以配置真实值，也可以配置为从环境变量中获取。

为了方便获取和使用配置，可以自定义配置类。在自定义属性类上需要增加注解，否则远程配置更新是客户端的配置不会自动更新。如下所示，定义了一个前缀为user的属性类，自动绑定user前缀的属性。

```
@RefreshScope
@ConfigurationProperties(prefix = "user")
public class User implements InitializingBean, DisposableBean {
  private String name;
  private int age;
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getAge() {
```

```
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

配置完成后，启动应用。在启动应用后查看启动日志，会发现在一般情况下，应用监听三个配置文件而不论这三个远程配置文件是否存在。

```
[fixed-prod-IP_47588] [subscribe] +paas-default+prod
[fixed-prod-IP_47588] [add-listener] ok, tenant=prod, dataId=, group=paas-default, cnt=1
[Nacos Config] Listening config: dataId=, group=paas-default
[fixed-prod-IP_47588] [subscribe] -prod.properties+paas-default+prod
[fixed-prod-IP_47588] [add-listener] ok, tenant=prod, dataId=-prod.properties, group=paas-default, cnt=1
[Nacos Config] Listening config: dataId=-prod.properties, group=paas-default
[fixed-prod-IP_47588] [subscribe] .properties+paas-default+prod
[fixed-prod-IP_47588] [add-listener] ok, tenant=prod, dataId=.properties, group=paas-default, cnt=1
[Nacos Config] Listening config: dataId=.properties, group=paas-default
```

默认情况下，监听的配置文件的dataId=${prefix}-prefix-{spring.profile.active}.${file-extension}。

其中profix默认为${spring.application.name}。

file-extension表示配置的类型，默认为properties。

如果没有指定spring.profile.active，那么dataId就变成了profix.{profix}.profix.{file-extension}。

启动时会发现启动日志中会打出多个：

${prefix}

${prefix}-{spring.profile.active}

${prefix}-{spring.profile.active}.${file-extension}

说明其实是可以匹配配置中心中配置的多条配置名称。匹配优先级是：3>2>1，精确匹配。

通过 控制台更新配置，在应用端可以接收到更新的推送。如果服务端同时存在多个监听的配置，则当更新高优先级的配置时，客户端才会接收到更新。

客户端接收到更新推送，变更为最新的值。至此，说明接入配置中心成功，配置可以动态更新了。

如何在MSE上为Dubbo应用构建服务注册中心？

事前准备

本文的目的是关于Dubbo 应用接入MSE注册配置中心。首先是工程准备，一般情况下是两个Spring Boot 应用和一个公共的接口模块。

模块说明：

1. common-api，公共接口模块（接口），供服务消费者和服务提供者调用。
2. dubbo-provider服务提供者模块（接口实现类），依赖common-api模块。
3. dubbo-consumer服务消费者模块（controller），依赖common-apii模块。

消费者和提供者通过公共接口模块进行rpc远程调用。

Dubbo主要相关的依赖项和版本如下所示：

```
<!--dubbo相关-->
<dependency>
   <groupId>org.apache.dubbo</groupId>
   <artifactId>dubbo-spring-boot-starter</artifactId>
   <version>2.7.15</version>
</dependency>

<dependency>
   <groupId>org.apache.dubbo</groupId>
   <artifactId>dubbo-registry-nacos</artifactId>
   <version>2.7.15</version>
</dependency>
```

在接入之前需要有一个MSE Nacos实例，并获得期访问信息。

公共模块

公共接口模块里面只有一个接口，没有配置文件，打jar包。

```
public interface InfoService{
   String getInfo();
}
```

Provider模块

application.yml配置文件：

```
server:
 port: 22020
spring:
 application:
   name: dubbo-provider

dubbo:
 registry:
   address: nacos://访问地址?username=nacos&password=密码
```

```
 application:
  name: dubbo-provider
 protocol:
  name: dubbo
  port: PORT
 scan:
  base-packages: com.ctg.mse.dubbo.pro.service
 provider:
  timeout: 30000
```

接口实现类，该类实现了公共接口模块创建的接口.注意包路径必须包含在配置文件的sacn.base-package路径中。

```
package com.ctg.mse.dubbo.pro.service;

import com.ctg.mse.common.InfoService;
import org.apache.dubbo.config.annotation.DubboService;
import org.springframework.stereotype.Component;

@Component
@DubboService
public class InfoServiceImpl implements InfoService {
   @Override
   public String getInfo() {
      return "Hello , dubbo provider method!";
   }
}
```

Provider启动类：

```
@EnableDubbo
@SpringBootApplication
public class DubboProviderApplication {
   public static void main(String[] args) {
      SpringApplication.run(DubboProviderApplication.class, args);
   }
}
```

Consumer模块

application.yml配置文件：

```
server:
 port: 21060
spring:
 application:
  name: dubbo-consumer

dubbo:
```

```
 registry:
   address: nacos://Nacos访问地址?username=用户名&password=密码
 application:
   name: dubbo-consumer
 consumer:
   timeout: 30000
```

controller，调用公共接口模块创建的接口：

```
@RestController
public class InfoController {

   @DubboReference(check = false)
   private InfoService infoService;

   @GetMapping("/getInfo")
   public String getInfo() {
      return infoService.getInfo();
   }
}
```

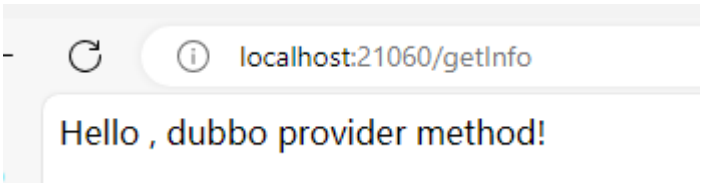
Consumer启动类：

```
@EnableDubbo
@SpringBootApplication
public class DubboConsumerApplication {
   public static void main(String[] args) {
      SpringApplication.run(DubboConsumerApplication.class, args);
   }
}
```

服务调用测试

启动服务提供者和服务消费者，打MSE 开Nacos控制台>服务管理-服务列表页面查看注册中心中的服务，查看服务注册情况。

调用服务消费者的getInfo接口，服务提供者会返回结果。



至此，dubbo融合MSE Nacos作为注册中心成功。

使用云原生网关实现蓝绿、金丝雀发布及AB实验

概述

蓝绿部署（Blue-Green Deployment）和金丝雀部署（Canary Deployment）是部署中常用的两种策略，用于在生产环境中引入新版本的应用程序或服务。这两种部署策略旨在降低风险并确保新版本的稳定性，同时允许逐步发布或回滚变更。

蓝绿部署中存在两个完全独立的生产环境（通常称为蓝环境和绿环境）被用于部署不同版本的应用程序。最初，蓝环境是当前正在运行的稳定版本，而绿环境是新版本的部署目标。一旦绿环境成功部署并通过测试，可以将流量切换到绿环境，并将蓝环境作为备份或回滚选项保留。这种方式可以确保在生产环境中保持稳定，并在需要时快速回滚到之前的版本。

金丝雀部署是一种逐步发布新版本的策略。在金丝雀部署中，新版本的应用程序或服务仅在一小部分用户或服务器上进行部署，这些用户或服务器被称为金丝雀群体。通过监控金丝雀群体的性能和稳定性，可以评估新版本的表现，并在没有负面影响的情况下逐步扩大金丝雀群体的规模，直到最终将新版本部署到整个生产环境。如果金丝雀部署中发现了问题或负面影响，可以快速回滚到之前的版本，以避免对所有用户造成影响。

步骤

部署服务

首先在容器内部署两个版本的服务，两个版本的Deployment分别挂到reviews-v1和reviews-v2服务下：

```
apiVersion: v1
kind: Service
metadata:
 name: reviews-v1
 labels:
  workloadKind: Deployment
  workloadName: reviews-v1
spec:
 ports:
  - port: 9080
    targetPort: 9080
    name: http
 selector:
  app: reviews
  version: v1
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: reviews-v1
spec:
 replicas: 1
 selector:
```

```
  matchLabels:
    name: reviews-v1
 template:
  metadata:
    labels:
     app: reviews
     version: v1
     name: reviews-v1
     source: CCSE
  spec:
    containers:
     - name: reviews
       image: 'registry-vpc-crs-huadong1.cnsp-internal.ctyun.cn/library/istio-examples-
bookinfo-reviews-v1:1.16.2'
       imagePullPolicy: IfNotPresent
       env:
        - name: LOG_DIR
          value: "/tmp/logs"
       volumeMounts:
        - name: tmp
          mountPath: /tmp
        - name: wlp-output
          mountPath: /opt/ibm/wlp/output
    volumes:
     - name: wlp-output
       emptyDir: {}
     - name: tmp
       emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
 name: reviews-v2
 labels:
  workloadKind: Deployment
  workloadName: reviews-v2
spec:
 ports:
  - port: 9080
    targetPort: 9080
    name: http
 selector:
  app: reviews
  version: v2
---
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
 name: reviews-v2
spec:
 replicas: 1
 selector:
  matchLabels:
   name: reviews-v2
 template:
  metadata:
   labels:
    app: reviews
    version: v2
    name: reviews-v2
    source: CCSE
  spec:
   containers:
    - name: reviews
      image: 'registry-vpc-crs-huadong1.cnsp-internal.ctyun.cn/library/istio-examples-
bookinfo-reviews-v2:1.16.2'
      imagePullPolicy: IfNotPresent
      env:
       - name: LOG_DIR
         value: "/tmp/logs"
      volumeMounts:
       - name: tmp
         mountPath: /tmp
       - name: wlp-output
         mountPath: /opt/ibm/wlp/output
   volumes:
    - name: wlp-output
      emptyDir: {}
    - name: tmp
      emptyDir: {}
```

添加服务到网关

进入云原生网关控制台，选择目标实例，进入服务来源菜单，将容器集群添加为服务来源；

再进入服务管理菜单，选择 创建服务，选择服务所在的命名空间，选择刚才部署的reviews-v1和reviews-v2服务，添加为网关服务。

创建路由

进入路由配置菜单，选择创建路由。

基于请求特征访问灰度版本

在请求头部匹配规则处可以添加规则End-User头部为jason时访问reviews-v2版本，这样在请求带上End-User:jason头部时将访问到v2版本。

基于比例做灰度流量控制

创建路由时选择多服务路由，选择路由目标为reviews-v1和reviews-v2，并配置流量比例，这样在请求时会按照指定的比例访问到对应版本的服务。

通过自建网关实现全链路灰度

概述

您可以基于微服务治理在不修改任何业务代码的情况下，实现全链路灰度的流量控制。本文介绍用户如何通过自建网关实现全链路灰度功能。

前提条件

1、用户已开通微服务治理中心企业版。

2、用户已开通云容器引擎。

背景信息

在微服务架构下，一次需求可能会同时修改多个微服务应用。在发布应用时，通常将这些应用划分为同一个分组，使灰度流量始终在灰度应用中流转。当上游有灰度流量时，会通过引流的方式将灰度流量引导至灰度分组，在此次链路调用过程中，如果存在一些微服务没有灰度环境，那这些请求在下游时依然能回到灰度环境中，以此实现全链路灰度。

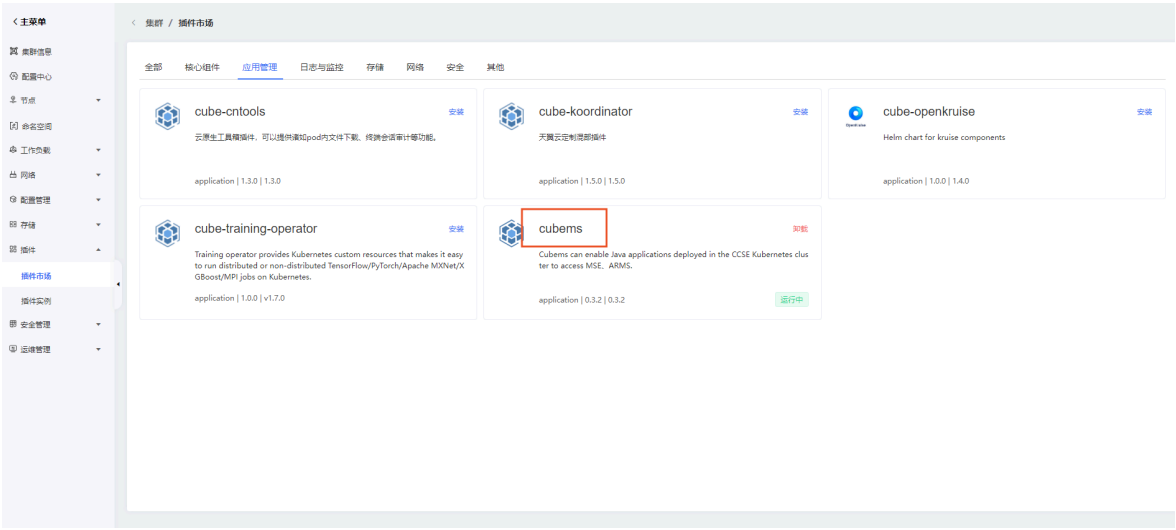
通过使用微服务治理中心，可以在不修改业务代码的情况下，轻松实现全链路灰度。本文介绍如何通过自建网关实现全链路灰度。

部署Demo应用

准备自建入口网关msgc-zuul，准备应用msgc-app-a，msgc-app-b和msgc-app-c。调用过程是msgc-app-a -> msgc-app-b -> msgc-app-c。

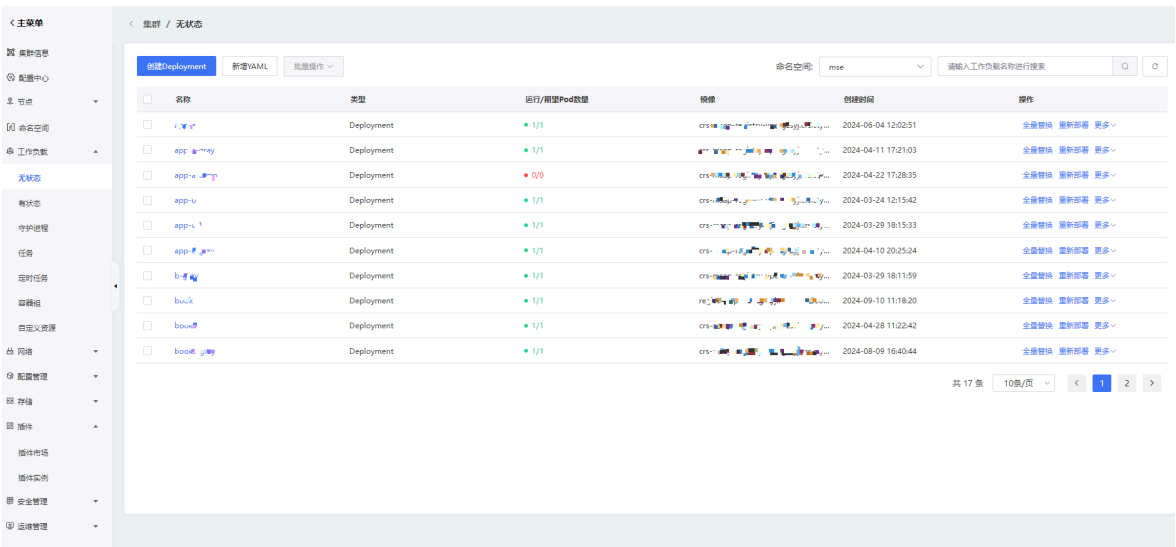
步骤1：在云容器引擎集群中安装微服务治理插件：

1. 登录云容器引擎控制台。
2. 在左侧菜单栏选择集群，点击目标集群。
3. 在集群管理页面点击插件-插件市场，选择cubems插件安装。



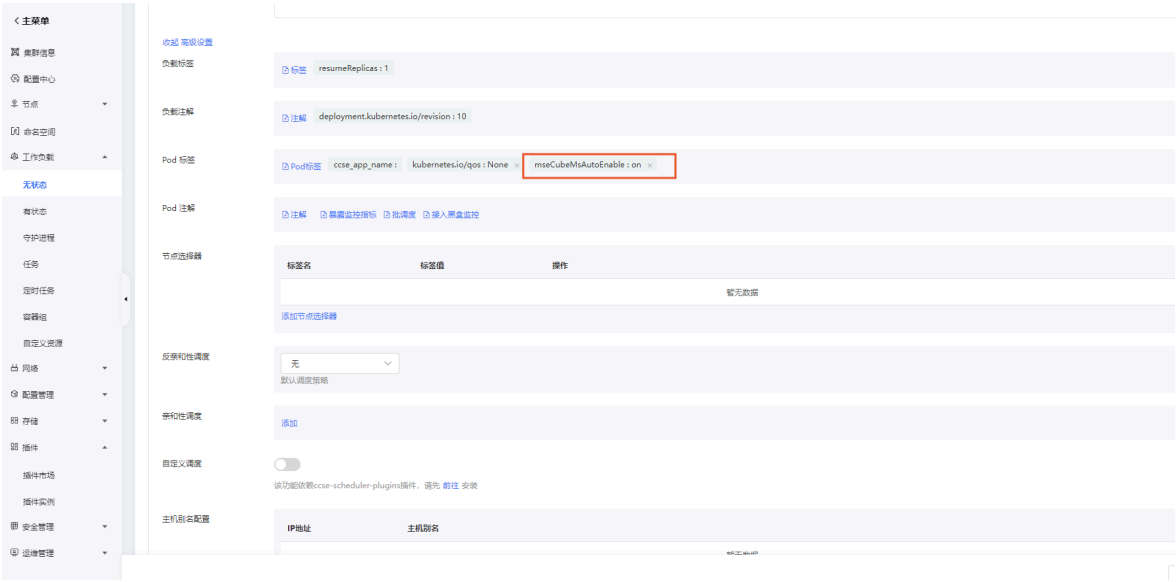
步骤2：为应用开启微服务治理能力：

1. 登录云容器引擎控制台。
2. 左侧菜单栏选择集群，点击目标集群。
3. 在集群管理页面点击工作负载-无状态，选择目标命名空间。



4. 在Deployment列表页选择指定Deployment，并点击全量替换，进入Deployment编辑页。

5. 在Deployment编辑页点击显示高级设置，新增Pod标签: mseCubeMsAutoEnable:on。



6. 在发布应用时，配置指定环境变量，可指定注入微服务治理中心的应用名、命名空间和标签等信息。

环境变量配置如下：

环境变量名	环境变量值
MSE_APP_NAME	接入到微服务治理中心的应用名。
MSE_SERVICE_TAG	应用标签信息，如灰度应用可配置gray。
MSE_NAMESPACE（选填）	接入到微服务治理中心的命名空间，默认为：default。

7. 完成编辑后点击提交，重新发布容器即可接入。

app-a应用的配置：

基线：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-a"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-a"
 template:
  metadata:
```

```
    labels:
     mseCubeMsAutoEnable: "on"
     name: "app-a"
   spec:
    containers:
    - env:
     - name: "MSE_APP_NAME"
       value: "app-a"
image: "镜像仓库域名/xxx/app-a:latest"
     imagePullPolicy: "Always"
     name: "app-a"
     ports:
     - containerPort: 26160
     livenessProbe:
      tcpSocket:
       port: 26160
      initialDelaySeconds: 10
      periodSeconds: 30
     resources:
      limits:
       cpu: "1"
       memory: "1Gi"
      requests:
       cpu: "1"
       memory: "1Gi"
```

灰度：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-a"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-a"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-a"
  spec:
```

```
    containers:
    - env:
      - name: "MSE_APP_NAME"
        value: "app-a"
- name: "MSE_SERVICE_TAG"
        value: "gray"
image: "镜像仓库域名/xxx/app-a:latest"
      imagePullPolicy: "Always"
      name: "app-a"
      ports:
      - containerPort: 26160
      livenessProbe:
        tcpSocket:
          port: 26160
        initialDelaySeconds: 10
        periodSeconds: 30
      resources:
        limits:
          cpu: "1"
          memory: "1Gi"
        requests:
          cpu: "1"
          memory: "1Gi"
```

app-b应用的配置：

基线：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-b"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
    name: "app-b"
 template:
  metadata:
    labels:
      mseCubeMsAutoEnable: "on"
      name: "app-b"
  spec:
    containers:
```

```
    - env:
      - name: "MSE_APP_NAME"
        value: "app-b"
 image: "镜像仓库域名/xxx/app-b:latest"
      imagePullPolicy: "Always"
      name: "app-b"
      ports:
      - containerPort: 26160
      livenessProbe:
        tcpSocket:
          port: 26160
        initialDelaySeconds: 10
        periodSeconds: 30
      resources:
        limits:
          cpu: "1"
          memory: "1Gi"
        requests:
          cpu: "1"
          memory: "1Gi"
```

灰度：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-b"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
    name: "app-b"
 template:
  metadata:
    labels:
      mseCubeMsAutoEnable: "on"
      name: "app-b"
  spec:
    containers:
    - env:
      - name: "MSE_APP_NAME"
        value: "app-b"
 - name: "MSE_SERVICE_TAG"
```

```
      value: "gray"
 image: "镜像仓库域名/xxx/app-b:latest"
     imagePullPolicy: "Always"
     name: "app-b"
     ports:
     - containerPort: 26160
     livenessProbe:
      tcpSocket:
       port: 26160
      initialDelaySeconds: 10
      periodSeconds: 30
     resources:
      limits:
       cpu: "1"
       memory: "1Gi"
      requests:
       cpu: "1"
       memory: "1Gi"
```

app-c应用的配置：

基线：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-c"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-c"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-c"
  spec:
   containers:
   - env:
    - name: "MSE_APP_NAME"
     value: "app-c"
 image: "镜像仓库域名/xxx/app-c:latest"
     imagePullPolicy: "Always"
```

```
      name: "app-c"
      ports:
      - containerPort: 26160
      livenessProbe:
       tcpSocket:
        port: 26160
       initialDelaySeconds: 10
       periodSeconds: 30
      resources:
       limits:
        cpu: "1"
        memory: "1Gi"
       requests:
        cpu: "1"
        memory: "1Gi"
```

灰度：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-c"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-c"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-c"
  spec:
   containers:
   - env:
     - name: "MSE_APP_NAME"
      value: "app-c"
- name: "MSE_SERVICE_TAG"
      value: "gray"
image: "镜像仓库域名/xxx/app-c:latest"
     imagePullPolicy: "Always"
     name: "app-c"
     ports:
```

```
      - containerPort: 26160
      livenessProbe:
       tcpSocket:
        port: 26160
       initialDelaySeconds: 10
       periodSeconds: 30
      resources:
       limits:
        cpu: "1"
        memory: "1Gi"
       requests:
        cpu: "1"
        memory: "1Gi"
```

zuul应用的配置：

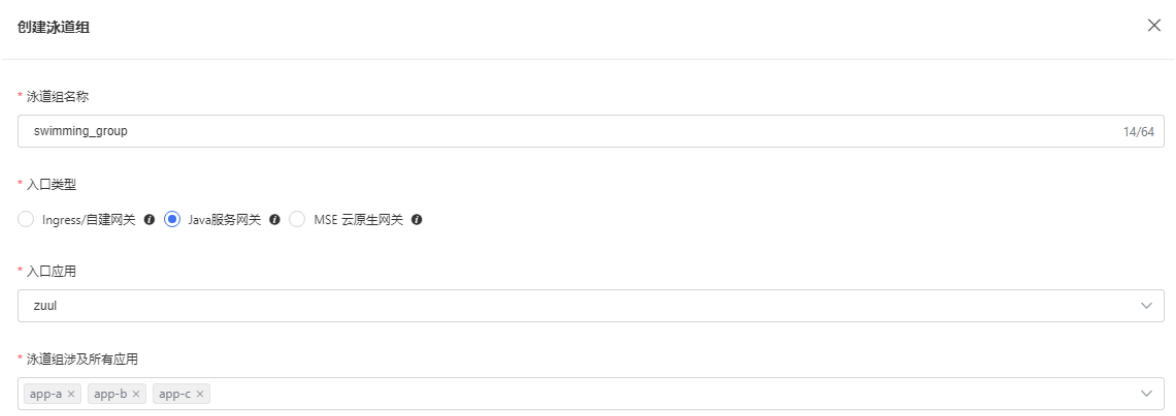
```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "zuul"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "zuul"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "zuul"
  spec:
   containers:
   - env:
    - name: "MSE_APP_NAME"
     value: "zuul"
image: "镜像仓库域名/xxx/zuul:latest"
    imagePullPolicy: "Always"
    name: "zuul"
    ports:
    - containerPort: 26160
    livenessProbe:
     tcpSocket:
      port: 26160
```

```
    initialDelaySeconds: 10
    periodSeconds: 30
  resources:
    limits:
      cpu: "1"
      memory: "1Gi"
    requests:
      cpu: "1"
      memory: "1Gi"
```

创建泳道组及灰度泳道

step1：创建泳道组

1. 登录微服务治理控制台。
2. 在左侧导航栏选择全链路灰度，点击创建泳道组。
3. 在创建泳道组页面，设置泳道组相关参数，然后单击确定。



step2：创建泳道

1. 找到目标泳道组页面，点击创建第一个分流泳道。
2. 设置泳道名称，选择目标应用所属标签，创建泳道。

场景应用及验证：通过设置泳道组路由规则，实现全链路灰度

step1：设置Header路由规则

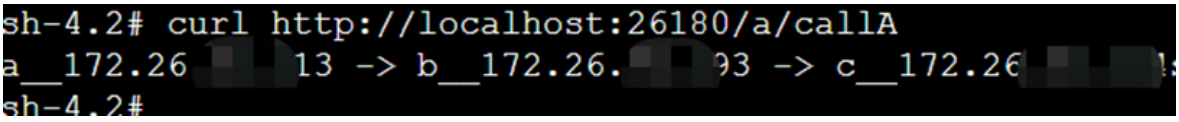
设置路由规则，通过前端访问传过来不同的Header，在自建网关通过header判断将流量路由到指定的泳道。

step2：通过Header验证灰度流量

通过自建网关访问app-a->app-b->app-c。

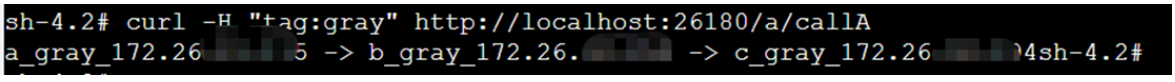
1. 访问基线环境，不带上参数名为tag的header，流量路由到基线环境。

curl http://localhost/a/callA



2. 访问灰度环境，带上tag=gray的header，流量会自动路由到灰度泳道。

curl -H “tag:gray” http://localhost/a/callA



基于上述实践，可以看到，当前端请求带上tag=gray的header时，会路由到灰度环境。

step3：通过设置Parameter实现标签路由

路由规则除了设置Header以外，还可以通过设置Cookie、Parameter和Body Content实现流量路由。

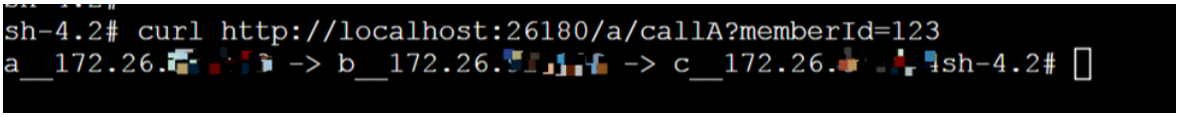
设置路由规则，通过前端访问传过来不同的Parameter，在自建网关通过Parameter判断将流量路由到指定的泳道。

step4：通过Parameter验证灰度流量

通过自建网关访问app-a->app-b->app-c。

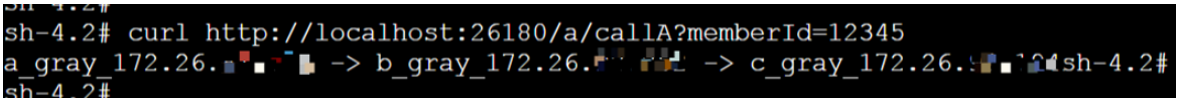
1. 访问基线环境，当member不等于12345时，流量路由到基线环境。

curl http://localhost/a/callA?memberId=123



2. 访问灰度环境，带上memberId=12345的参数，流量会自动路由到灰度泳道。

curl http://localhost/a/callA?memberId=12345



基于上述实践，可以看到，当前端请求带上memberId=12345的参数，会路由到灰度环境。

基于消息队列RocketMQ实现全链路灰度

概述

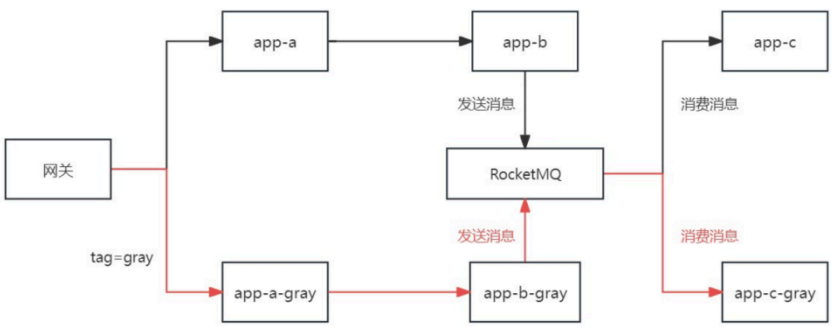
本文介绍在使用消息队列（RocketMQ）这种异步场景下，可以在不修改业务代码的情况下，实现异步场景的灰度，从而实现全链路灰度。本文介绍基于消息队列RocketMQ实现全链路灰度。

背景介绍

在大多数业务场景中对于消息的灰度并没有RPC调用那么严格，但是当全链路灰度调用中涉及到消息消费时，如果消息消费没有按照全链路流量规则路由，则会导致通过消息产生的流量逃逸，从而破坏全链路规则，导致出现一些不符合预期的情况。

如下图所示，本文分别部署网关、app-a、app-a-gray、app-b、app-b-gray、app-c、app-c-gray以及RocketMQ，模拟一个真实的全链路灰度场景。

通过网关调用app-a应用的接口，当满足路由规则后，灰度流量会被路由到app-a-gray，app-a-gray又会调用app-b-gray，随后由app-b-gray发送灰度消息，app-c-gray将会收到灰度消息，而app-c不会收到灰度消息。



前提条件

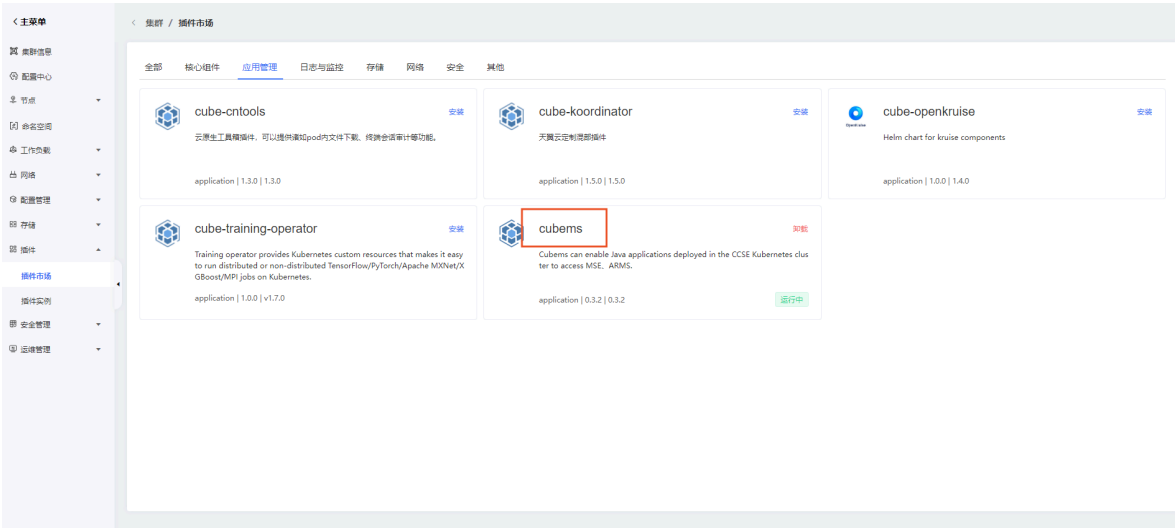
1. 用户已开通微服务治理中心企业版。
2. 用户已开通云容器引擎。
3. 用户已部署RocketMQ，且RocketMQ版本在4.5.0以上，broker.conf中已配置enablePropertyFilter=true。

部署Demo应用

准备自建入口网关msgc-zuul，准备应用msgc-app-a，msgc-app-b和msgc-app-c。调用过程是msgc-app-a -> msgc-app-b -> msgc-app-c。

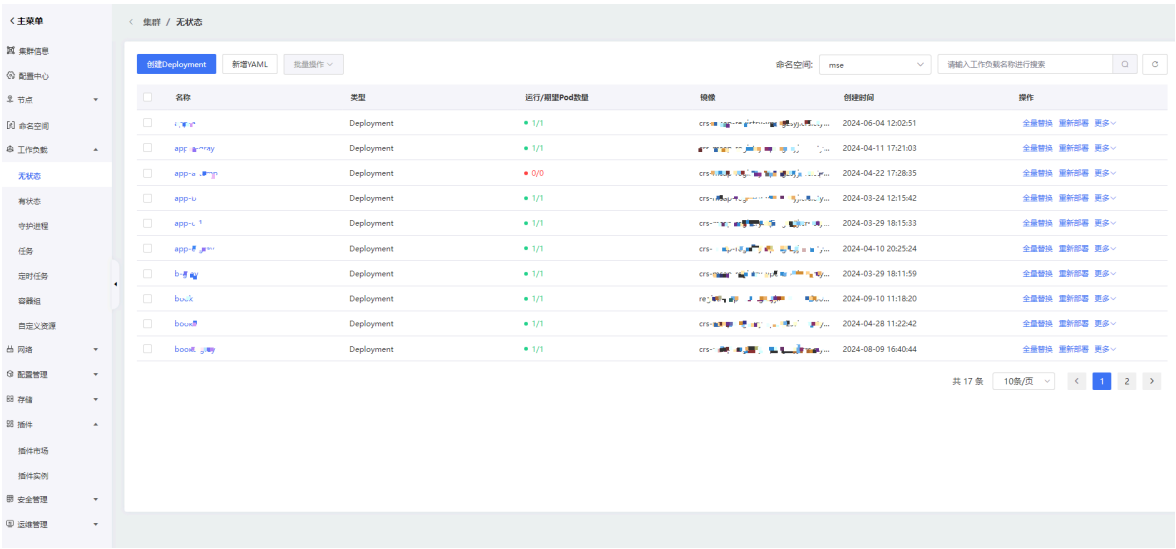
步骤1：在云容器引擎中安装微服务治理插件：

1. 登录“云容器引擎”控制台。
2. 在左侧菜单栏选择“集群”，点击目标集群。
3. 在集群管理页面点击“插件”-“插件市场”，选择“cubems”插件安装。

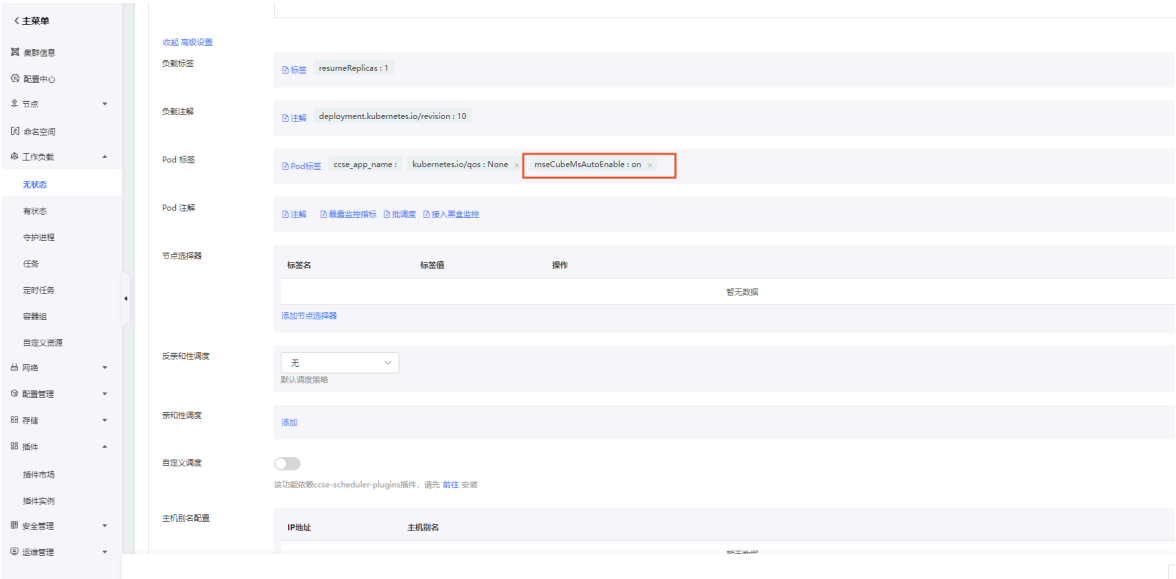


步骤2：为应用开启微服务治理能力：

1. 登录“云容器引擎”控制台。
2. 左侧菜单栏选择“集群”，点击目标集群。
3. 在集群管理页面点击“工作负载”-“无状态”，选择目标命名空间。



4. 在Deployment列表页选择指定Deployment，并点击“全量替换”，进入Deployment编辑页。
5. 在Deployment编辑页点击“显示高级设置”，新增“Pod标签”：mseCubeMsAutoEnable:on。



6. 在发布应用时，配置指定环境变量，可指定注入微服务治理中心的应用名、命名空间和标签等信息。

环境变量配置如下：

环境变量名	环境变量值
MSE_APP_NAME	接入到微服务治理中心的应用名。
MSE_SERVICE_TAG	应用标签信息，如灰度应用可配置gray。
MSE_NAMESPACE（选填）	接入到微服务治理中心的命名空间，默认为：default。

7. 完成编辑后点击“提交”，重新发布容器即可接入。

app-a应用的配置

基线：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-a"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-a"
 template:
  metadata:
```

```
    labels:
     mseCubeMsAutoEnable: "on"
     name: "app-a"
   spec:
    containers:
    - env:
     - name: "MSE_APP_NAME"
       value: "app-a"
image: "镜像仓库域名/xxx/app-a:latest"
     imagePullPolicy: "Always"
     name: "app-a"
     ports:
     - containerPort: 26160
     livenessProbe:
      tcpSocket:
       port: 26160
      initialDelaySeconds: 10
      periodSeconds: 30
     resources:
      limits:
       cpu: "1"
       memory: "1Gi"
      requests:
       cpu: "1"
       memory: "1Gi"
```

灰度：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-a"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-a"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-a"
  spec:
```

```
    containers:
    - env:
      - name: "MSE_APP_NAME"
        value: "app-a"
- name: "MSE_SERVICE_TAG"
        value: "gray"
image: "镜像仓库域名/xxx/app-a:latest"
     imagePullPolicy: "Always"
     name: "app-a"
     ports:
     - containerPort: 26160
     livenessProbe:
       tcpSocket:
         port: 26160
       initialDelaySeconds: 10
       periodSeconds: 30
     resources:
       limits:
         cpu: "1"
         memory: "1Gi"
       requests:
         cpu: "1"
         memory: "1Gi"
```

app-b应用的配置

基线：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-b"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
    name: "app-b"
 template:
  metadata:
    labels:
      mseCubeMsAutoEnable: "on"
      name: "app-b"
  spec:
    containers:
```

```
    - env:
      - name: "MSE_APP_NAME"
        value: "app-b"
 image: "镜像仓库域名/xxx/app-b:latest"
      imagePullPolicy: "Always"
      name: "app-b"
      ports:
      - containerPort: 26160
      livenessProbe:
        tcpSocket:
          port: 26160
        initialDelaySeconds: 10
        periodSeconds: 30
      resources:
        limits:
          cpu: "1"
          memory: "1Gi"
        requests:
          cpu: "1"
          memory: "1Gi"
```

灰度：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-b"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
    name: "app-b"
 template:
  metadata:
    labels:
      mseCubeMsAutoEnable: "on"
      name: "app-b"
  spec:
    containers:
    - env:
      - name: "MSE_APP_NAME"
        value: "app-b"
 - name: "MSE_SERVICE_TAG"
```

```
      value: "gray"
image: "镜像仓库域名/xxx/app-b:latest"
     imagePullPolicy: "Always"
     name: "app-b"
     ports:
     - containerPort: 26160
     livenessProbe:
      tcpSocket:
       port: 26160
      initialDelaySeconds: 10
      periodSeconds: 30
     resources:
      limits:
       cpu: "1"
       memory: "1Gi"
      requests:
       cpu: "1"
       memory: "1Gi"
```

app-c应用的配置

基线：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-c"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-c"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-c"
  spec:
   containers:
   - env:
    - name: "MSE_APP_NAME"
     value: "app-c"
image: "镜像仓库域名/xxx/app-c:latest"
     imagePullPolicy: "Always"
```

```
      name: "app-c"
      ports:
      - containerPort: 26160
      livenessProbe:
       tcpSocket:
        port: 26160
       initialDelaySeconds: 10
       periodSeconds: 30
      resources:
       limits:
        cpu: "1"
        memory: "1Gi"
       requests:
        cpu: "1"
        memory: "1Gi"
```

灰度：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-c"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-c"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-c"
  spec:
   containers:
   - env:
     - name: "MSE_APP_NAME"
      value: "app-c"
- name: "MSE_SERVICE_TAG"
      value: "gray"
image: "镜像仓库域名/xxx/app-c:latest"
     imagePullPolicy: "Always"
     name: "app-c"
     ports:
```

```
      - containerPort: 26160
      livenessProbe:
       tcpSocket:
        port: 26160
       initialDelaySeconds: 10
       periodSeconds: 30
      resources:
       limits:
        cpu: "1"
        memory: "1Gi"
       requests:
        cpu: "1"
        memory: "1Gi"
```

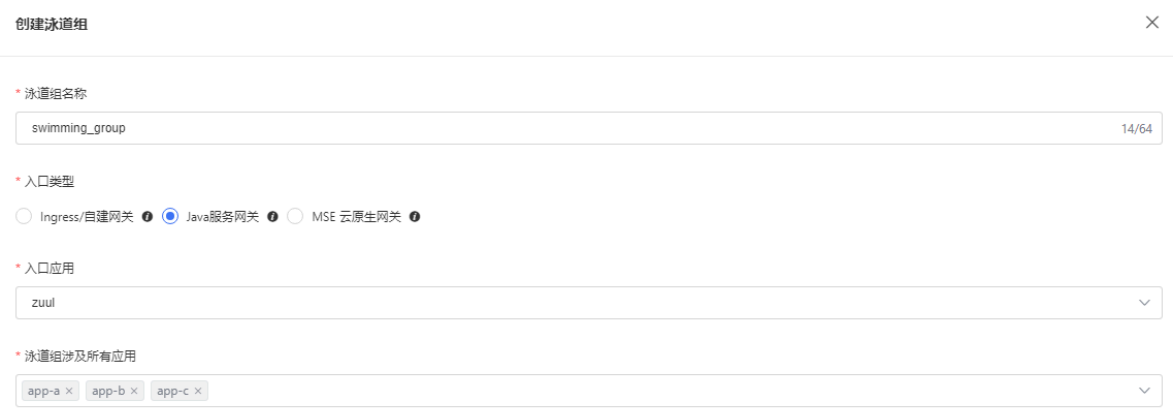
zuul应用的配置：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "zuul"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "zuul"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "zuul"
  spec:
   containers:
   - env:
    - name: "MSE_APP_NAME"
     value: "zuul"
image: "镜像仓库域名/xxx/zuul:latest"
    imagePullPolicy: "Always"
    name: "zuul"
    ports:
    - containerPort: 26160
    livenessProbe:
     tcpSocket:
      port: 26160
```

```
    initialDelaySeconds: 10
    periodSeconds: 30
  resources:
    limits:
      cpu: "1"
      memory: "1Gi"
    requests:
      cpu: "1"
      memory: "1Gi"
```

创建泳道组

1. 登录微服务治理控制台。
2. 在左侧导航栏选择全链路灰度，点击创建泳道组。
3. 在创建泳道组也页面，设置泳道组相关参数，然后单击确定。



创建泳道

1. 找到目标泳道组页面，点击创建第一个分流泳道。
2. 设置泳道名称，选择目标应用所属标签，创建泳道。

设置泳道组路由规则

设置路由规则，通过前端访问传过来不同的Header，在自建网关通过header判断将流量路由到指定的泳道。

开启消息灰度

1. 进入微服务治理中心控制台，点击应用治理。
2. 依次点击应用卡片app-a、app-b和app-c。
3. 选择流量治理->消息灰度。
4. 点击编辑，分别打开app-a、app-b和app-c的消息灰度开关。

结果验证

通过自建网关访问app-a->app-b->app-c。

不携带Header参数tag=gray请求app-a接口/a/mqGray，发现app-c获得消息消费。

携带Header参数tag=gray请求app-a接口/a/mqGray，发现app-c-gray获得消息消费。

基于微服务治理中心实现无损上线与无损下线

概述

无损上线主要通过服务预热和延迟注册实现，无损下线主要通过应用优雅停机实现。假设有应用app-a和app-b两个应用，app-a一直都有请求调用app-b，本文演示在app-b上线与下线时，如何实现无损上线与下线。

前提条件

用户已开通微服务治理中心企业版。

背景信息

在微服务架构中，在应用变更时，会涉及到应用的上线与下线，若处理不当，此时会造成一些请求异常。因此，在请求量大的应用系统中，为了避免应用变更带来的错误请求，一般会选择在半夜这种流量较小的时候发布。在半夜发布虽然避免了对业务造成较大的影响，但却不能从根本上解决问题，同时也会造成较大的运维成本。基于以上背景，微服务治理中心在应用发布的过程中，提供了服务延迟注册、服务预热、主动通知下线、主动等待等手段，实现了应用变更的无损上线与下线，有效避免了流量损失，降低研发运维成本。

准备工作

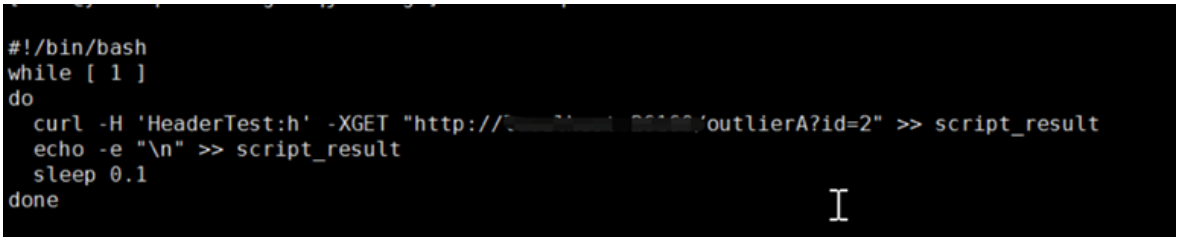
部署Demo应用

本文准备两个应用，分别是app-a和app-b，其中调用关系为app-a->app-b。

部署一个app-a：

部署一个app-b：

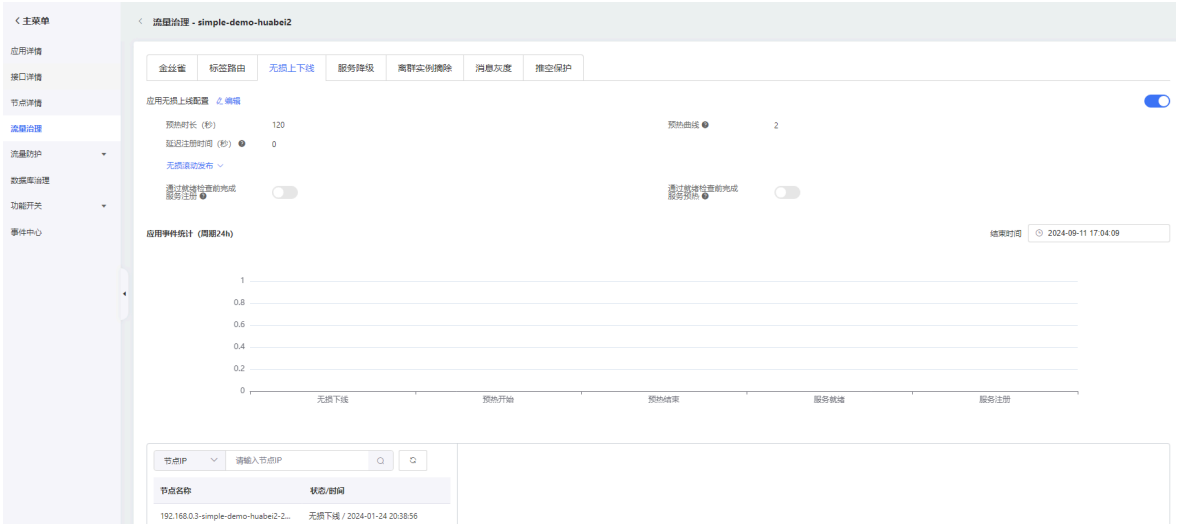
启动脚本访问app-a->app-b：



此时查看微服务治理中心控制台，app-a和app-b都有流量。

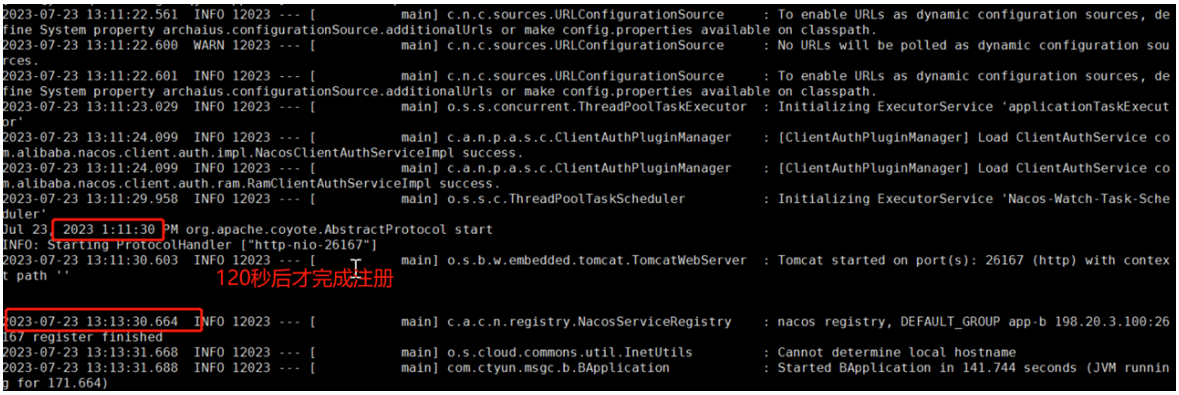
验证无损上线

打开app-a和app-b的无损上下线开关。



在当前启动的应用基础上，再启动一个app-b，此时app-b总共有两个节点，新启动的app-b会发生延迟注册和服务预热两个事件。

延迟注册：



由上图所示，服务在启动后120秒后，才完成服务注册，达到延迟注册的效果。

服务预热：

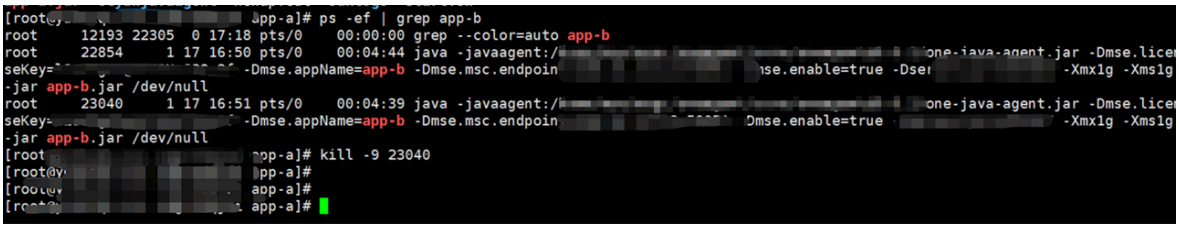
由上图所示，服务在启动后，先是小流量预热，随后逐渐达到最大流量。

验证无损下线

微服务治理中心的无损下线主要是通过主动通知注册中心下线和主动通知消费者来实现。在应用下线时，需要使用kill -15 pid命令，通过下线钩子函数通知应用即将下线，达到无损下线的效果。如果采用kill -9 pid，将会是有损下线。

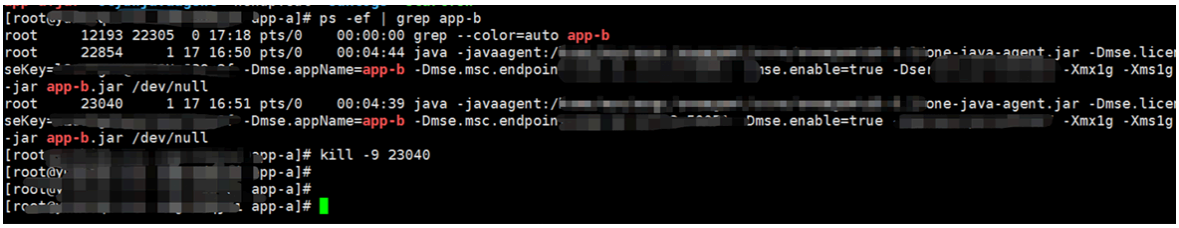
当前部署一个app-a，两个app-b。在有流量访问的情况下，下线app-b其中一个节点。

使用kill -9 pid下线。

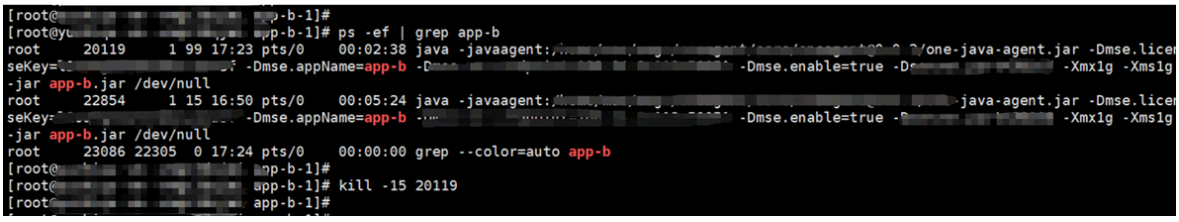


执行kill命令后，查看微服务治理中心控制台，此时出现了异常调用。

使用kill -15 pid下线。



执行kill命令后，查看微服务治理中心控制台，此时没有出现异常调用。



通过以上实践，证明了微服务治理中心可以实现无损下线。

基于Ingress-APISIX网关实现全链路灰度

概述

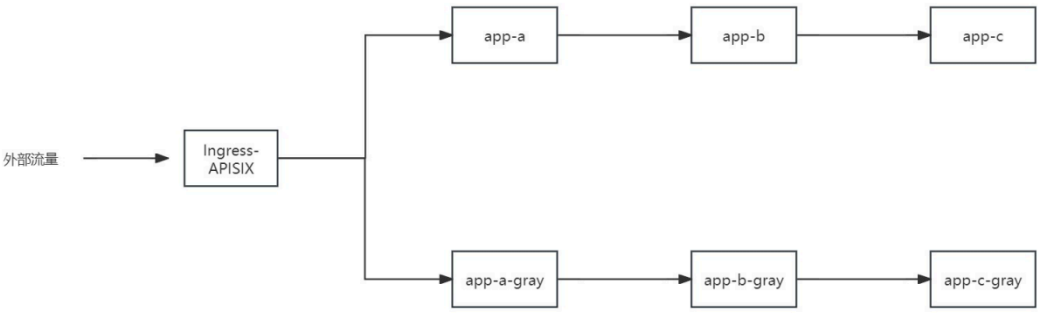
APISIX是动态、实时、高性能的API网关，可以提供丰富的流量管理能力，可以作为K8s Ingress控制器。本文介绍安装Ingress-APISIX后，通过灵活的路由能力，在无需修改业务代码的情况下，实现全链路灰度能力。

背景信息

在微服务架构下，一次需求可能会同时修改多个微服务应用。在发布应用时，通常将这些应用划分为同一个分组，使灰度流量始终在灰度应用中流转。当上游有灰度流量时，会通过引流的方式将灰度流量引导至灰度分组，在此次链路调用过程中，如果存在一些微服务没有灰度环境，那这些请求在下游时依然能回到灰度环境中，以此实现全链路灰度。

本文将APISIX提供的灵活的路由能力和微服务治理中心的全链路灰度能力相结合，在无需修改业务代码的情况下，轻松实现全链路灰度能力。

本文的演示应用由Ingress-APISIX和Spring Cloud应用组成，Spring Cloud应用有3个，分别是app-a、app-b和app-c，服务之间通过Nacos注册中心实现服务注册与发现，部署完成后，通过Ingress-APISIX配置路由规则，来访问后端服务。



前提条件

1. 用户已开通微服务治理中心企业版。
2. 用户已开通云容器引擎。
3. 用户已开通注册配置中心Nacos。

准备工作

部署Spring Cloud应用

1. 登录云容器引擎，选择左侧菜单“集群”，再点进目标集群。
2. 在集群管理页面，点击工作负载->无状态。
3. 选择命名空间，点击“新增YAML资源”。
4. 本文部署app-a、app-b和app-c三个应用，每个应用分别部署一个基线版本和一个灰度版本，同时开通一个注册配置中心Nacos，用于实现服务注册与发现。

部署app-a应用基线版本YAML：

```
apiVersion:"apps/v1"
kind:"Deployment"
metadata:
 name:"app-a"
namespace:"default"
spec:
 progressDeadlineSeconds:600
 replicas:1
 revisionHistoryLimit:10
 selector:
  matchLabels:
   name:"app-a"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable:"on"
    name:"app-a"
  spec:
```

```
    containers:
    - env:
      - name: "MSE_APP_NAME"
        value: "app-a"
image: "镜像仓库域名/xxx/app-a:latest"
      imagePullPolicy: "Always"
      name: "app-a"
      ports:
      - containerPort: 26160
      livenessProbe:
        tcpSocket:
          port: 26160
        initialDelaySeconds: 10
        periodSeconds: 30
      resources:
        limits:
          cpu: "1"
          memory: "1Gi"
        requests:
          cpu: "1"
          memory: "1Gi"
```

部署app-a应用灰度版本YAML：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  name: "app-a"
namespace: "default"
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      name: "app-a"
  template:
    metadata:
      labels:
        mseCubeMsAutoEnable: "on"
        name: "app-a"
    spec:
      containers:
      - env:
        - name: "MSE_APP_NAME"
          value: "app-a"
```

```
- name: "MSE_SERVICE_TAG"
      value: "gray"
image: "镜像仓库域名/xxx/app-a:latest"
     imagePullPolicy: "Always"
     name: "app-a"
     ports:
     - containerPort: 26160
     livenessProbe:
      tcpSocket:
       port: 26160
      initialDelaySeconds: 10
      periodSeconds: 30
     resources:
      limits:
       cpu: "1"
       memory: "1Gi"
      requests:
       cpu: "1"
       memory: "1Gi"
```

部署app-b应用基线版本YAML：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-b"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-b"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-b"
  spec:
   containers:
   - env:
    - name: "MSE_APP_NAME"
      value: "app-b"
image: "镜像仓库域名/xxx/app-b:latest"
     imagePullPolicy: "Always"
```

```
      name: "app-b"
      ports:
      - containerPort: 26160
      livenessProbe:
       tcpSocket:
        port: 26160
       initialDelaySeconds: 10
       periodSeconds: 30
      resources:
       limits:
        cpu: "1"
        memory: "1Gi"
       requests:
        cpu: "1"
        memory: "1Gi"
```

部署app-b应用灰度版本YAML：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-b"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-b"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-b"
  spec:
   containers:
   - env:
     - name: "MSE_APP_NAME"
       value: "app-b"
- name: "MSE_SERVICE_TAG"
      value: "gray"
image: "镜像仓库域名/xxx/app-b:latest"
    imagePullPolicy: "Always"
    name: "app-b"
    ports:
```

```
      - containerPort: 26160
      livenessProbe:
       tcpSocket:
        port: 26160
       initialDelaySeconds: 10
       periodSeconds: 30
      resources:
       limits:
        cpu: "1"
        memory: "1Gi"
       requests:
        cpu: "1"
        memory: "1Gi"
```

部署app-c应用基线版本YAML：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-c"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-c"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-c"
  spec:
   containers:
   - env:
    - name: "MSE_APP_NAME"
     value: "app-c"
image: "镜像仓库域名/xxx/app-c:latest"
    imagePullPolicy: "Always"
    name: "app-c"
    ports:
    - containerPort: 26160
    livenessProbe:
     tcpSocket:
      port: 26160
```

```
      initialDelaySeconds: 10
      periodSeconds: 30
     resources:
      limits:
       cpu: "1"
       memory: "1Gi"
      requests:
       cpu: "1"
       memory: "1Gi"
```

部署app-c应用灰度版本YAML：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
 name: "app-c"
namespace: "default"
spec:
 progressDeadlineSeconds: 600
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   name: "app-c"
 template:
  metadata:
   labels:
    mseCubeMsAutoEnable: "on"
    name: "app-c"
  spec:
   containers:
   - env:
     - name: "MSE_APP_NAME"
       value: "app-c"
- name: "MSE_SERVICE_TAG"
      value: "gray"
image: "镜像仓库域名/xxx/app-c:latest"
     imagePullPolicy: "Always"
     name: "app-c"
     ports:
     - containerPort: 26160
     livenessProbe:
      tcpSocket:
       port: 26160
      initialDelaySeconds: 10
      periodSeconds: 30
```

```
    resources:
     limits:
      cpu: "1"
      memory: "1Gi"
     requests:
      cpu: "1"
      memory: "1Gi"
```

在云容器引擎中安装Ingress APISIX

自主安装apisix、apisix-ingress-controller和Dashboard组件。具体安装步骤可参考：Apache APISIX Helm Chart 参考安装。

对app-a进行网络配置

针对入口应用app-a，分别为基线版本和灰度版本配置两个K8s Service，用于Ingress APISIX转发流量。

1. 登录云容器引擎，选择左侧菜单“集群”，再点进目标集群。
2. 在集群管理页面，点击网络->服务。
3. 选择命名空间，点击“新增YAML”，依次创建下面两个YAML资源。

app-a-base-service对应app-a应用的基线版本：

```
apiVersion: "v1"
kind: "Service"
metadata:
 name: "app-a-base-service"
 namespace: "default"
spec:
 ports:
  - name: "http"
    port: 26160
    protocol: "TCP"
    targetPort: 26160
 selector:
  name: "app-a"
```

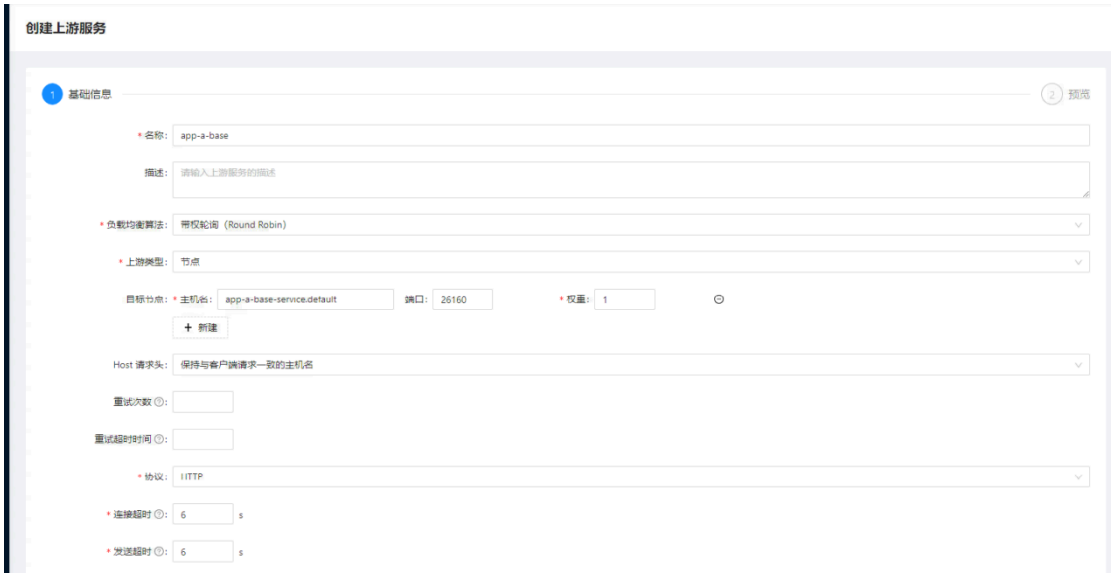
app-a-base-service对应app-a应用的基线版本：

```
apiVersion: "v1"
kind: "Service"
metadata:
 name: "app-a-gray-service"
 namespace: "default"
spec:
 ports:
  - name: "http"
    port: 26160
```

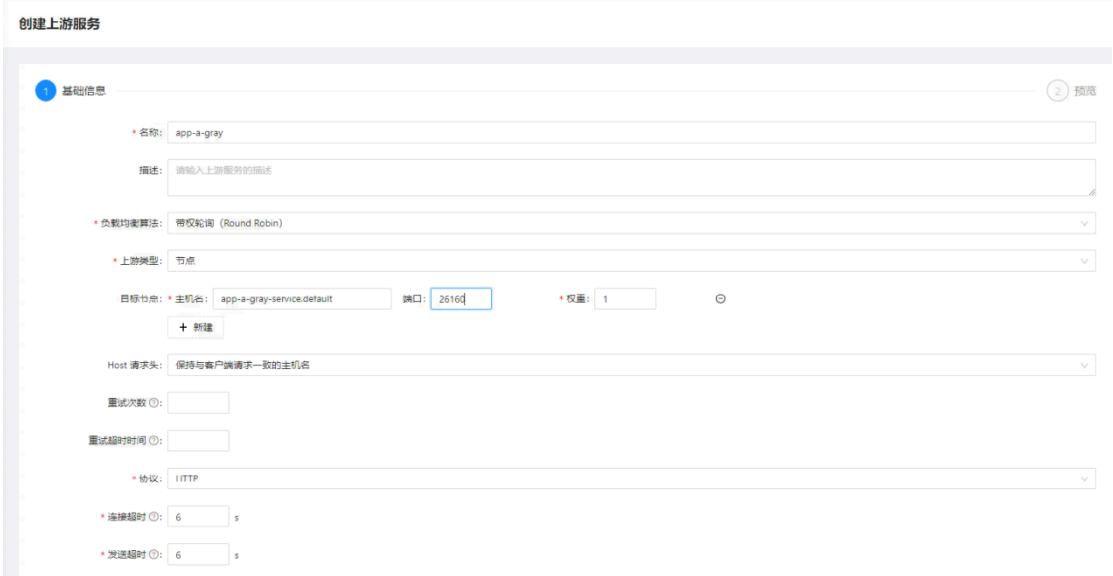
```
    protocol: "TCP"
    targetPort: 26160
  selector:
    name: "app-a-gray"
```

4. 登录APISIX控制台，点击“上游”，在上游管理页面中点击创建，分别为app-a和app-a-gray配置上游服务，配置详情如下：

app-a：



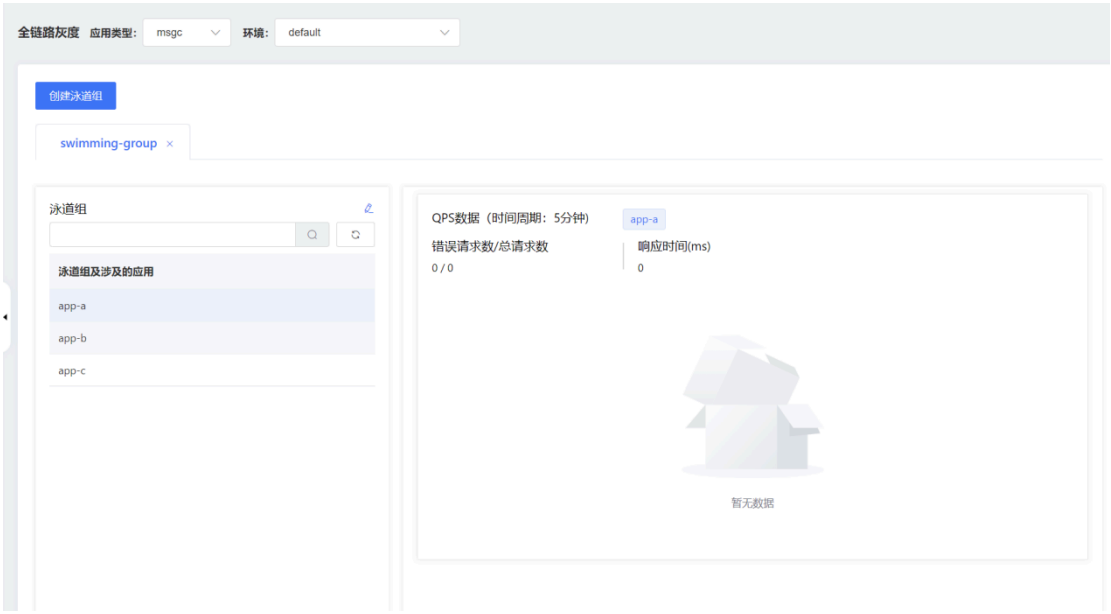
app-a-gray：



创建全链路灰度泳道和泳道组

完成准备工作后，即可在微服务治理中心控制台创建全链路灰度泳道和泳道组。

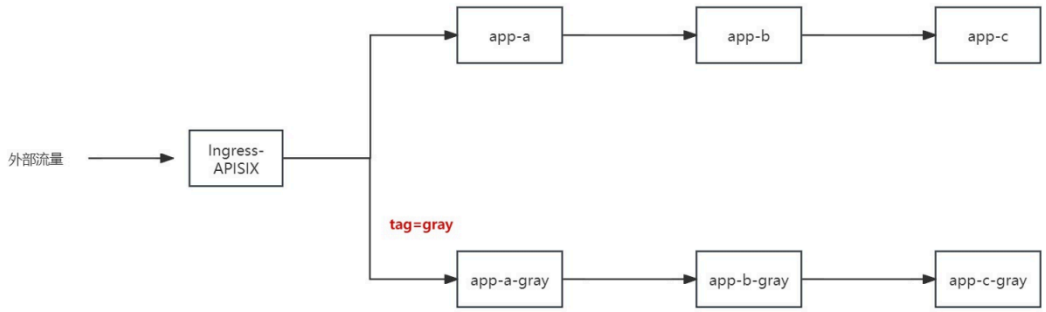
1. 登录微服务治理中心控制台，点击全链路灰度。
2. 在全链路灰度页面点击“创建泳道组及泳道”。
3. 设置泳道组名称，选择入口类型为“Ingress/自建网关”，选择“泳道组涉及所有应用”：app-a、app-b和app-c。



4. 点击创建第一个分流泳道，设置“泳道名称”，选择标签“gray”，点击创建泳道。

通过 APISIX配置路由规则， 按照指定请求参数进行路由，实现全链路灰度

通过在Ingress-APISIX设置路由规则，将请求携带参数tag=gray的流量路由到app-a-gray。根据全链路灰度规则，携带参数tag=gray的请求路径是Ingress-APISIX > app-a-gray > app-b-gray > app-c-gray。



配置APISIX路由规则

1. 登录APISIX控制台，点击左侧菜单“路由”，然后点击“创建”。
2. 在创建路由页面，首先设置路由信息，分别设置app-a-route和app-a-gray-route的路由名称、路由版本和高级匹配条件。

app-a-route：设置路由名称为app-a-route，路由版本1.0.0，设置优先级为0,点击下一步，选择上游服务app-a-base，点击下一步，创建app-a-route路由。

```
{
 "uri": "/*",
 "name": "app-a-base",
```

```
 "methods":[
  "GET",
  "POST",
  "PUT",
  "DELETE",
  "PATCH",
  "HEAD",
  "OPTIONS",
  "CONNECT",
  "TRACE",
  "PURGE"
 ],
 "upstream_id":"49xxxxxxxxx937",
 "labels":{
  "API_VERSION":"1.0.0"
 },
 "status":1
}
```


app-a-gray-route：设置路由名称为app-a-gray-route，路由版本1.0.0，设置优先级为1,创建高级匹配条件，设置规则请求参数tag==gray，点击下一步，选择上游服务app-a-base，点击下一步，创建app-a-route路由。


```
{
 "uri":"/*",
 "name":"app-a-gray-route",
 "priority":1,
 "methods":[
  "GET",
  "POST",
  "PUT",
  "DELETE",
  "PATCH",
  "HEAD",
  "OPTIONS",
  "CONNECT",
  "TRACE",
  "PURGE"
 ],
 "vars":[
  [
   "arg_tag",
   "==",
   "gray"
  ]
 ],
 "upstream_id":"498xxxxxxxxx009",
 "labels":{
```

```
  "API_VERSION": "1.0.0"
 },
 "status": 1
}
```

结果验证

说明：192.168.xx.xx:9080为Ingress-APISIX的ip和端口。

不携带参数访问接口：curl http://192.168.xx.xx:9080/callA，路由到基线环境。

```
curl http://192.168.xx.xx:9080/callA
a__192.168.xx.xx -> b__192.168.xx.xx -> c__192.168.xx.xx
```

携带参数访问接口：curl http://192.168.xx.xx:9080/callA?tag=gray，路由到灰度环境。

```
curl http://192.168.xx.xx:9080/callA?tag=gray
a_gray_192.168.xx.xx -> b_gray_192.168.xx.xx -> c_gray_192.168.xx.xx
```

注册配置中心

注册配置中心基本问题

MSE注册配置中心如何使用？

微服务引擎注册配置中心是面向业界主流开源微服务项目，致力于帮助用户发现、配置和管理微服务，实现动态服务发现、服务配置、服务元数据及流量管理等功能。

MSE注册配置中心是否支持公网访问？

默认不可以公网访问，但可以通过绑定ELB实现公网访问。

MSE注册配置中心是否具备自动续费功能？

MSE支持自动续费功能。

注册配置中心如何开启IPv6访问？

1、确保当前注册配置中心实例所处的vpc已具备IPv6子网；如不具备，跳转至vpc网络控制台，创建或者开启IPv6子网； 2、开启实例IPv6，在实例订购页，网络配置中选择某个vpc，勾选IPv6（第一步具备时才可勾选），选择好其他配置后提交订购即可；

IPv4如何升级IPv6?

当前只有新开实例支持IPv6，存量IPv4实例暂不支持升级。

如何通过IPv6访问实例？

实例的IPv6访问有两种方式：1、通过内网IPv6地址访问，访问地址在实例详情中的基础信息页中已展示。2、通过ELB访问，此方式需要确保ELB实例本身已开启IPv6访问，目前注册配置中心实例暂只支持ELB的内网IPv6访问。

注册配置中心资源权限问题

MSE集群提供的ZooKeeper、Nacos、Eureka是独立实例还是共享实例？

各引擎都是物理隔离的网络实例，为每一个客户提供了物理隔离的独立实例，客户之间不共享。

MSE提供的ZooKeeper实例和开源版本完全一致吗？

在版本兼容性上和Apache开源的ZooKeeper完全一致，API使用方式相同，无需修改代码即可开始使用，不影响现有有实例。在功能上MSE提供了额外的商业化能力，增强了可视化、安全管控、便捷运维等方面的能力。

Nacos集群支持配置管理吗 ？

MSE控制台左侧菜单栏提供配置管理功能，支持包括配置创建、修改、订阅查询、删除等操作。

MSE能进行扩缩容吗？升降配吗？

支持实例集群节点扩容，实例规格升配，实例节点磁盘扩容；支持实例集权节点缩容。

MSE中Nacos命名空间怎么使用？

命名空间命的使用场景之一是对不同环境的配置和服务进行粗粒度的隔离。在不同的命名空间下，可以存在相同的配置或者服务名。例如开发环境和测试环境的隔离。命名空间创建完成后，将命名空间ID配置在应用中。服务注册时，命名空间ID会注册到命名空间中。

配置代码示例：

Spring Cloud：

spring.cloud.nacos.discovery.namespace=5cbb70a5-88b8-4fd9-84c1-d43479a

Dubbo：

Properties方式：

dubbo.registry.parameters.namespace=5cbb70a5-88b8-4fd9-84c1-d43479a

增加了配置中心的配置后，服务启动时会自动监听命名空间ID下匹配的配置项。

配置代码：

Spring Cloud：

spring.cloud.nacos.config.namespace=5cbb70a5-88b8-4fd9-84c1-d43479ae

Dubbo：

dubbo.dubbo.config-center.namespace=5cbb70a5-88b8-4fd9-84c1-d43479ae

黑 白名单填写的 IP地址是公网地址还是私网地址？

既可以是公网地址也可以是私网地址，取决于请求头中自动携带的客户端IP。

注册配置中心配置问题

注册配置中心监控支持Prometheus动态读取Nacos的服务列表而获取metrics吗？

支持，但在MSE里面不支持展示。

部署了Spring Cloud应用，但是使用的是Consul注册中心，如何实现注册中心的托管呢？

MSE目前已上线了Consul引擎，诚邀您开通试用。

如何退订服务？退订后资源会立刻释放吗？

1. 在实例列表页面选择目标实例退订。
   a. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
   b. 在左侧导航栏，选择注册配置中心> 实例列表。
   c. 在实例列表页面，选择目标实例所在行，点击操作列“更多”下面的“退订”按钮，跳转至确认页面。
   d. 在退订确认页面请确认需要退订的实例信息，无误后点击确认即可退订。
   e. 退订后实例无法再提供服务。
2. 退订后实例会被停止，无法访问，但资源不会立即释放，一般情况下数据会保留15天，准确保留时间以系统为准。

可以续订服务吗？服务能自动续期吗？

包周期的实例可随时进行续订，续订后，到期时间将后延您所续订的时间。服务在购买时可以设置自动续期，您只需要选中“自动续费”按钮即可。

购买的实例性能无法满足实际需求怎么办？

1. 购买实例前建议您先评估实际需求，然后参考产品规格章节 ，预留30%左右的容量，然后订购对应能力规格的产品。

2. 如果购买的实例无法随着业务发展无法满足实际需求可以使用扩容操作，对订购的实例进行扩容，扩容操作包括三种类型节点扩容（最高支持扩至9节点）、规格扩容（增加实例节点内存容量和CPU数量）、磁盘扩容（增加实例节点数据盘容量）。具体的操作可以参考如下步骤：
   a. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
   b. 在左侧导航栏，选择注册配置中心> 实例列表。
   c. 在实例列表页面，选择目标实例所在行，点击操作列扩容按钮，跳转至扩容页面。
   d. 在扩容页面选择扩容的类型以及目标规格，点击确认后提交订单，支付完毕后可在控制台实例列表页面查看扩容状态。

说明

1. 节点扩容限制只能3->5->7->9逐步扩容，不能一次增加超过半数的节点。
2. 规格扩容只能选择相同架构和类型下的2C4G、4C8G、8C16G等既定规格扩容，不能自定义规格。
3. 磁盘扩容必须是步长(50G)的整数倍，不能随意指定数字。

如何管理已注册服务

注册配置中心各引擎均提供服务管理能力，详情请参考注册配置中心菜单下各引擎子菜单中的管理服务功能。目前Nacos和Eureka引擎支持对服务进行上下线操作，Zookeeper引擎仅支持对服务进行查询。

操作步骤如下：

1. 登录微服务引擎MSE注册配置中心管理控制台，选择资源池。
2. 在左侧导航栏，选择注册配置中心> 实例列表。
3. 在实例列表页面，选择目标实例所在行，点击实例名称或者ID进入基础信息页面。
4. 基础信息页面左侧菜单栏选择点击服务管理，即可查看以注册的服务。

如何管理配置

点击查看按注册配置中心Nacos、Zookeeper提供配置管理能力，详情请参考注册配置中心菜单下Nacos、Zookeeper引擎子菜单中的配置管理功能。

Nacos引擎支持在配置管理页面根据DataID和group等条件进行查询，查询到的配置可以对应进行删除，同步，修改操作，修改操作可以改变其值，值类型，以及数据加密等，详细操作请参考Nacos引擎配置管理 。

Zookeeper引擎，配置可以通过Znode的方式进行存储和获取，详细操作请参考ZooKeeper引擎数据管理。

注册配置中心实例问题

订购的ZooKeeper/Nacos/Eureka实例为什么没有外网地址？

如果购买的实例没有绑定ELB，那么您的实例没有外网地址。您绑定ELB就可以用ELB的地址实现外网访问实例。

Nacos实例升级会不会影响用户正常服务？

对于多节点集群，重启不影响用户服务，实例升级过程中，各节点服务器依次滚动重启，保证用户使用Nacos服务不间断；对于单机实例，重启可能会受到服务中断影响，所以建议生产（线上）环境使用三节点以上的集群进行服务。

生产环境下Nacos设置多少个节点比较好呢？

Nacos建议的规格书2n+1个节点，最佳值需根据实际需求和规格能力计算获得。

Nacos支持开源的Nacos控制台吗？

默认不支持访问Nacos开源控制台。

Nacos Client 支持哪些 版本，推荐使用哪个版本？

Nacos Client 1.2.1版本以上均支持，推荐使用Nacos 2.x系列版本的Nacos Client。

MSE里的Nacos服务怎么下线？

服务管理页面，选择指定的命名空间和服务名称，查看服务详情，按服务实例的维度进行下线操作。

如果注册配置中心的某个实例被删除， 天翼云会提供恢复的解决方案吗？如果有的话需要多久能恢复？

实例过期时间提前7天会有短信通知，实例退订后数据会保留15天，在此期间可以恢复实例。超过时间则无法恢复。

如果在 注册配置中心 中创建多个实例，服务之间可以相互调用吗？

购买MSE时需要选择VPC ID，说明只能在该VPC内访问相应的MSE实例。如果多个VPC之间可以相互访问，则MSE也可以跨VPC调用。

注册配置中心实例类型买错了，怎么退掉重新购买？

如果您的实例是按需付费模式，则直接退订即可。如果您的实例是包周期模式，则需要退费。

如何在不同账号之间迁移实例？

目前暂不支持在不同账号之间迁移实例，若您需要将实例进行迁移，请遵循如下步骤进行手动迁移：

Nacos

1. 开通新账号，新建Nacos实例；
2. 迁移Nacos配置。使用Nacos配置导入导出能力，将旧实例的完整配置下载为本地文件，再导入到新实例；
3. 迁移Nacos服务。提供了Nacos Sync迁移工具，可提供服务自动平滑迁移能力，但目前仅支持相同vpc下实例的自动迁移，您需要将服务手动注册到新实例上，以完成服务迁移；
4. 迁移命名空间、访问权限等信息。目前没有针对该类元数据信息的迁移方案，需要用户在新实例进行手动创建。

Zookeeper

1. 开通新账号，新建Zookeeper实例；
2. 迁移Zookeeper持久化数据。迁移Zookeeper的Znode数据，方法详见Zookeeper持久化数据的迁移的迁移章节；
3. 迁移Zookeeper服务。由于Zookeeper服务数据默认为非持久化数据，对服务数据的迁移目前没有对应的平滑迁移方案，您需要将服务手动注册到新实例上，以完成服务迁移。

Eureka

1. 开通新账号，新建Eureka实例；
2. 迁移Eureka服务。对服务数据的迁移目前没有对应的平滑迁移方案，您需要将服务手动注册到新实例上，以完成服务迁移。

常见问题

Nacos FAQ

Nacos引擎的命名空间怎么使用？

命名空间是Nacos引擎内部的逻辑数据隔离分区概念。命名空间的常用场景之一是不同环境的配置和服务的区分隔离，例如开发环境、测试环境和生产环境的资源隔离等。不同的命名空间下，可以存在相同的Group、DataId或服务名称。命名空间创建完成后，将命名空间ID配置在应用中。服务注册时会根据配置注册到指定的命名空间中，如果没有指定命名命名空间，会默认注册到public。如果注册到一个不存在的命名空间ID，也能够提示注册成功，但是在控制台无法可视化操作该服务，创建对应的命名空间后就可以正常操作了。

配置代码：Spring Cloud yml方式（properties方式同理）。

```
spring:
 cloud:
  nacos:
   config:
    server-addr: ${NACOS_SERVER_ADDRESS}
    namespace: ${NACOS_CONFIG_NAMESPACE}
discovery:
    server-addr: ${NACOS_SERVER_ADDRESS}
    namespace: ${NACOS_NAMING_NAMESPACE}
```

Dubbo yml方式（properties方式同理）。

```
dubbo:
 registry:
  address: nacos://Nacos地址
  parameters[namespace]: 命名空间ID
```

生产环境下-Nacos引擎设置多少个节点比较好呢？

1. 购买实例前建议您先评估实际需求，然后参考章节：产品规格 ，预留30%左右的容量，然后订购对应能力规格的产品。
2. Nacos提供单机版和集群版，单机版只有一个节点，不建议生产使用。集群版提供3、5、7、9节点集群，集群节点数量越多，对故障节点数量的容错能力就越强，只要查过半数的节点正常就能正常提供服务例如9节点集群，即使同时有4个节点宕机或故障，依然能正常提供服务。建议您根据实际的需要以及服务的可用性要求订购对应的实例。
3. 如果购买的实例无法随着业务发展无法满足实际需求可以使用扩容操作。具体的操作可以参考章节：管理实例。

如何创建专属账户给客户端提供服务？

MSE中Nacos引擎默认开启鉴权功能，访问未认证或者授权的用户访问实例。

Nacos鉴权默认支持JWT认证，通过用户、角色、权限三者共同作用来赋予用户权限，详情请参见章节Nacos引擎访问权限 。

是否支持回滚配置的历史版本？

MSE 注册配置中心提供了配置历史查询功能。目前默认仅保存30天以内的变更记录。查看和回滚历史配置。

在基础信息页面，左侧菜单点击配置管理>配置列表，选择命名空间，查看当前配置。在左侧导航栏，选择历史版本。在历史版本页面选择命名空间、分组、和DataId，点击搜索，即可查看配置的历史。详细操作请参见章节配置管理。

在配置历史页面选择指定版本的历史配置击回滚，确认后即可回滚至历史版本。

在配置列表页面点击编辑按钮，可以查看当前配置的内容，可以确认是回滚至历史版本。回滚操作也会产生一条新的配置变更记录。

Nacos上修改服务实例的权重不生效问题

问题现象

控制台服务详情页面修改服务实例时，设置了实例的权重，但实际调用流量时，流量分配未按照预期权重进行分配。

问题原因

1. 应用框架不支持按权重分配流量负载均衡。
2. 应用框架有自身的负载均衡配置方式，不使用Nacos的权重属性进行负载均衡。
3. 应用未使用权重值进行地址选择。

解决方案

Nacos中服务实例的权重仅为实例的一个属性。当修改权重值后，权重值会伴随实例信息被推送到Nacos客户端。实际使用时由于使用方式的不同，可能会导致权重属性被忽略。

1. 如果所使用的应用框架不支持按权重分配流量。例如Spring Cloud等框架，此类框架仅识别流量值为0（不引入流量）和非0（引入流量），不支持按照Nacos实例中的流量值进行流量负载均衡。您可以按照以下方法解决。

- 寻找扩展插件以增加流量分配支持。
- 更换其他应用框架。

2. 应用框架有自身的负载均衡配置方式，不支持使用Nacos的权重属性。如Dubbo、Spring Cloud Alibaba + Loadbalance等。可以在社区中询问或根据对应框架文档查看如何配置负载均衡。
3. 未使用应用框架进行开发，但是应用在处理时未使用权重值进行地址选择。

应用在获取到实例列表后，根据其中的weight字段实现自定义的选择逻辑。

使用Nacos客户端默认的selectOneHealthyInstance方法进行选择。

Nacos 连接超时问题

问题现象

当程序连接Nacos出现超时问题时，可能出现如下几种报错：

Connection timed out

Read Timeout

TimeoutException: Waited 3000 milliseconds

问题原因

可能是以下几种原因，导致程序连接Nacos出现超时问题。

1. 客户端与服务端的网络传输异常，导致客户端发出的请求无法抵达服务端或服务端的回复无法抵达客户端。或者服务端处理请求速度慢，导致客户端误认为超时。
2. 使用VPN导致的网络问题。
3. 客户端的处理线程阻塞或异常，亦或客户端处于Full GC、OOM或CPU争抢等状态， 无法及时处理服务端返回的数据包，导致客户端误认为超时。
4. Nacos集群处于异常状态，无法响应请求。

解决方案

1. 如果仅有某一个客户端节点出现超时报错，可能是这些客户端节点与Nacos之间的网络出现问题，或是这些客户端节点存在异常或阻塞。

此时，您可在错误所在的客户端节点上，使用ping、telnet和curl等命令，访问Nacos集群。查看客户端监控是否存在高CPU使用率、频繁FullGC和OOM等信息，以此排查是否存在网络问题。

ping ${mse.nacos.host}

telnet ${mse.nacos.host}:47588

curl ${mse.nacos.host}:47588/nacos/v1/ns/service/list

2. 如果使用了VPN，请关闭VPN或查看VPN设置后重试。
3. 如错误信息存在于所有的客户端节点，可以在监控分析页面查看Nacos实例的监控信息：

- 在概览页签，查看引擎的每秒查询数和每秒操作数是否超过了每秒处理请求数（TPS）。
- 关于每秒处理数的取值，请参见实例能力评估。
- 在连接数监控页签，查看客户端版本数量和长链路数量是否超过了连接数。
- 关于连接数的取值，请参见实例能力评估。
- 在jvm监控页签，查看引擎Full GC是否频繁出现。
- 当通过ELB访问Nacos时，查看资源的入口流量和出口流量是否超出购买时指定的带宽大小。
- 在系统资源监控tab页查看资源的内存使用率和CPU使用率是否接近或超过100%，导致实例出现异常。

4. 如果仅是偶尔发生超时错误，考虑设置更长的超时时间避免此类问题。

- 若Java Client版本为1.0.0~1.4.X，则建议升级客户端版本到2.x系列版本。
- 若Java Client版本为2.0.0~2.1.1， 请将Java Client版本先升级至2.1.2及以上，再设置超时时间。
- 若Java Client版本为2.1.2及以上，请在应用进程的JVM参数中添加如下参数：

-Dnacos.remote.client.grpc.timeout=${请求超时，单位毫秒，默认3000};

检测所连接的服务端是否健康，不健康则触发重连。

-Dnacos.remote.client.grpc.server.check.timeout=${服务端健康检测，单位毫秒，默认3000};

检测当前连接状态是否健康，不健康则触发重连。

-Dnacos.remote.client.grpc.health.timeout=${连接健康检测，单位毫秒，默认3000};

Nacos 连接失败问题

问题现象

当程序连接Nacos出现连接失败问题时，可能会出现如下几种报错。

Client not connected,currentstatus:STARTING

Client not connected,currentstatus:UNHEALTHY

no available server, currentServerAddr: xxxxx

Connection refused

问题原因

可能是如下几种原因，导致程序连接Nacos出现连接失败。

Nacos的访问地址或端口配置错误。

访问Nacos集群时，网络异常。

使用VPN导致的网络问题。

客户端存在高CPU使用率、频繁FullGC等问题。

解决方案

1. 在错误所在的客户端节点上，使用ping、telnet和curl等命令，访问Nacos集群，排查是否存在网络问题。

ping ${mse.nacos.host}

telnet ${mse.nacos.host}:47588

telnet ${mse.nacos.host}:48588

curl ${mse.nacos.host}:47588/nacos/v1/ns/service/list

2. 检查应用的相关配置，是否配置了正确的MSE实例访问地址、端口等信息。
3. 如果报错信息为Connection refused ，请从紧随其后的报错信息中查看实际连接的地址与MSE实例的连接地址是否不相同。例如Connection refused: /127.0.0.1:48588说明某些配置错误地指向了本机地址。
4. 如果使用了VPN，请检查VPN设置是否正确。若不正确，请关闭VPN或修改VPN设置后重试。
5. 若通过上述步骤还无法定位问题，注册配置中心控制台的监控分析页面，查看Nacos的如下信息：

- 在概览页签，查看引擎的每秒查询数和每秒操作数是否超过了每秒处理请求数（TPS）。
- 关于每秒处理数的取值，请参见实例能力评估。
- 在连接数监控页签，查看长链路数量是否超过了连接数。
- 关于连接数的取值，请参见实例能力评估。
- 在jvm监控页签，查看引擎是否频繁出现Full GC。
- 在资源监控页签，查看资源的入口流量和出口流量是否超出购买时指定的带宽大小。
- 在资源监控页签，查看资源的内存使用率和CPU使用率是否接近或超过100%，导致节点可能出现异常。
- 资源的内存使用率和CPU使用率接近或超过100%，请尝试变更实例规格进行升配，请参见变更实例规格。

Nacos 持久化实例健康检查异常问题

问题现象

当在Nacos中注册的持久化实例选择健康检查方式为HTTP/TCP时，服务实例的健康状态始终显示为不健康，但实例配置或状态正常。

可能原因

MSE的Nacos为托管类产品，部署在内网资源池的vpc中，不与应用程序部署在一起。出于安全规范的考量，仅开放单向请求，在网络层面禁止从服务端向外部发起的TCP连接/HTTP请求。上述原因可能导致健康检查始终会以超时等网络原因显示失败。

解决方案

将注册的服务类型修改为非持久化。即注册服务提供者时，指定ephemeral字段为true或移除对ephemeral字段的设置（ephemeral字段缺省值为true）。

如何 查找Nacos-Client日志

Nacos-Client的日志根据相关的编程语言不同而有所差异，不同的编程语言版本Client的日志获取方式如下：

Java Nacos Client

Java语言的Nacos-Client的日志一般在应用服务所在节点的{user.home}/logs/nacos/目录下，{user.home}为启动应用服务进程的系统用户的根目录。

若使用的是Spring Cloud，部分低版本Spring Cloud会覆盖Nacos-Client的日志配置，导致日志输出在应用服务的日志中。

其中，naming.log是注册中心模块相关日志，config.log是配置中心模块相关日志。2.0.0之后版本中，Nacos-Client新增了remote.log，remote.log是gRPC连接相关的日志。

Go Nacos-Client

Go语言的Nacos-Client的日志默认在/tmp/nacos/log/目录下，可以通过LogDir:参数修改日志路径。

Go语言的Nacos-Client日志不区分具体模块内容，应该所有的日志都会在同样的日志文件中。

Python Nacos-Client

Python语言的Nacos-Client使用Python的Logging模块，会和应用的Logging模块保持一致并输出到应用的日志中。

C++ Nacos-Client

C++语言的Nacos-Client的日志默认在应用所在目录下，文件名为nacos-sdk-cpp.log，可通过Logger.cpp中的setBaseDir设置日志目录。

引擎升级或重启时，出现访问 异常或中断服务

集群升级或重启会轮转进行。不同节点数量的集群可能出现的情况不同。

1. 3个节点以上的集群：不会出现中断服务，无法访问的情况，但仍可能出现少量的短时间的请求失败，客户端会立刻重试恢复。
2. 单节点集群：会导致集群不可用并中断服务。仅开发版集群为单节点，不保证高可用。

控制台还能查到不存在的服务 实例IP如何处理的

问题现象

应用服务实例停止，注册配置中心控制台仍能看到该服务实例。

应用服务重启或发布后，注册配置中心控制台仍能看到该服务实例。

可能原因

服务实例并未彻底关闭，进程仍然存在并发送心跳维持连接，导致Nacos未摘除服务提供。

有额外的应用进程在发送心跳维持连接，导致Nacos未摘除服务提供者。

解决方案

1. 确认该服务实例已经不应该在线的情况下，先在MSE控制台上对该服务提供者执行下线操作，防止有更多流量进入到该故障节点。
2. 根据部署环境的不同排查服务提供者是否未彻底关闭：

- 直接部署到云主机：登录到对应IP的云主机上，使用ps -ef | grep ${应用名}、netstat -anp | grep 48588或netstat -anp | grep 47588等命令，查看服务提供者进程是否还存在，是否与Nacos还保持着连接。如果是，则确认后关闭该进程。
- 通过自建Kubernetes、Docker或容器服务部署：检查是否存在幽灵Pod或Container（即Pod或Container已经不可见，但对应的程序进程未终止销毁），可通过在Node或宿主机上执行**ps -ef | grep ${应用名}**等命令，查看是否该应用提供者的个数等同于期望个数。如果不相同，则确认后找到该幽灵Pod并彻底关闭。

Zookeeper FAQ

为什么Zookeeper会出现zxid溢出？

本文介绍在使用ZooKeeper时，客户端出现zxid（Zookeeper事务ID）溢出的问题现象、问题原因和解决方案。

问题现象

ZooKeeper集群强制选主，并重置zxid低32位的计数值。

问题原因

zxid是一个长64位的数字。高32位用来表示当前Leader的周期，低32位用来表示当前请求产生的事物在当前Leader周期内的位置。每产生一个新的事务，zxid的低32位就会自动加1。当zxid达到最大值，即zxid的低32位达到0xffffffff，就会触发集群强制选主，并重置zxid低32位的计数值（zxid高32位变为新Leader的周期，低32位变为0）。

解决方案

目前，Server没有规避zxid溢出的方法，请在业务侧提前规避。

对于使用ZooKeeper作为注册配置中心的使用场景：集群选主不会影响到正常使用，客户端在集群选主之后会自动重连恢复。

为什么Zookeeper会出现APIError 溢出？

本文将介绍使用MSE ZooKeeper时，客户端出现APIError报错的问题现象，问题原因和解决方案。

问题现象

使用ZooKeeper客户端出现APIError报错，如下所示：

org.apache.zookeeper.KeeperException$APIErrorException: KeeperErrorCode = APIError for /xxx。

问题原因

在使用MSE ZooKeeper时客户端触发了MSE ZooKeeper的限流策略。

解决方案

保障ZooKeeper单个会话创建的临时节点（ephemeral）数量少于2000个。

为什么Zookeeper客户端会出现Conection Loss？

本文将介绍使用MSE ZooKeeper时，客户端出现Conection Loss报错的问题现象，问题原因和解决方案。

问题现象

使用ZooKeeper客户端出现APIError报错，如下所示：

org.apache.zookeeper.KeeperException$ConnectionLossException: ZooKeeper connection lost. Trying to reconnect。

问题原因

出现Zookeeper客户端出现conection loss有可能两个原因，一是网络发生波动不稳定情况，二是连接超时，超过了设置的连接时间。

解决方案

可以尝试在参数管理页面对超时时间（minSessionTimeout和syncLimit）适当放宽，详情参考章节：ZooKeeper引擎参数设置。

Eureka FAQ

如何解决Eureka连接失败问题？

问题现象

使用Eureka客户端连接-Eureka服务端时，报错Cannot execute request on any known server，如下图所示。

或是报错There is no known eureka server; cluster server list is empty，如下图所示。

```
com.netflix.discovery.shared.transport.TransportException: There is no known eureka server; cluster server list is empty
  at com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClient.execute(RetryableEurekaHttpClient.java:108) ~[eureka-client-1.10.14.jar:1.10.14]
  at com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorator.register(EurekaHttpClientDecorator.java:56) ~[eureka-client-1.10.14.jar:1.10.14]
  at com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorator$1.execute(EurekaHttpClientDecorator.java:59) ~[eureka-client-1.10.14.jar:1.10.14]
  at com.netflix.discovery.shared.transport.decorator.SessionedEurekaHttpClient.execute(SessionedEurekaHttpClient.java:77) ~[eureka-client-1.10.14.jar:1.10.14]
  at com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorator.register(EurekaHttpClientDecorator.java:56) ~[eureka-client-1.10.14.jar:1.10.14]
  at com.netflix.discovery.DiscoveryClient.register(DiscoveryClient.java:876) ~[eureka-client-1.10.14.jar:1.10.14]
  at com.netflix.discovery.InstanceInfoReplicator.run(InstanceInfoReplicator.java:121) [eureka-client-1.10.14.jar:1.10.14]
  at com.netflix.discovery.InstanceInfoReplicator$1.run(InstanceInfoReplicator.java:101) [eureka-client-1.10.14.jar:1.10.14] <7 internal calls>
  at java.lang.Thread.run(Thread.java:748) [na:1.8.0_301]
```

问题原因

Eureka客户端无法连通任意您指定的Eureka服务端地址，或是您指定的Eureka服务端地址配置不符合标准格式。

解决方案

请检查您在客户端配置中指定的eureka.client.service-url.defaultZone配置项，确保所填写的服务端地址正确且在客户端侧能够telnet通。格式需为http://{server-ip}:{server-port}/eureka，如果为多节点服务端，则不同节点之间使用英文逗号分隔，如下图所示。


```
eureka.client.service-url.defaultZone=http://localhost:28761/eureka,http://localhost:28762/eureka,http://localhost:28763/eureka
```

如何诊断注册到Eureka服务的健康状态？

注册到Eureka的服务有时可能会因为宕机或者网络原因而无法继续提供服务，Eureka提供了健康检测机制来保证消费者不会调用到不健康的服务。

您也可以在Eureka实例控制台-服务管理-服务列表，选择服务，点击详情按钮，查看当前服务实例的实时健康状态。当服务实例出现不健康的情况时，Eureka服务端会定期对其进行驱逐。若此类情况发生，请检查您服务的真实状态。

如何查看Eureka开源控制台UI？

默认不支持查看您实例的Eureka开源控制台UI，您可以通过我们提供的Eureka实例控制台对Eureka的基本信息进行查询，并进行相关管理端操作，功能更加全面。

常见问题

如何解决Eureka服务消费者获取不到服务提供者地址的问题？

问题现象

您已经试图将服务消费者和服务提供者注册到Eureka，但服务消费者获取不到提供者地址，报错Load balancer does not have available server for client:{your-service-name}，如下图所示。

```
2024-01-03 14:10:10.628 ERROR 33040 --- [nio-9087-exec-1] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request
 processing failed; nested exception is java.lang.RuntimeException: com.netflix.client.ClientException: Load balancer does not have available server for client: eureka-provider-other] with root cause

com.netflix.client.ClientException: Load balancer does not have available server for client: eureka-provider-other
  at com.netflix.loadbalancer.LoadBalancerContext.getServerFromLoadBalancer(LoadBalancerContext.java:483) ~[ribbon-loadbalancer-2.3.0.jar:2.3.0]
  at com.netflix.loadbalancer.reactive.LoadBalancerCommand$1.call(LoadBalancerCommand.java:184) ~[ribbon-loadbalancer-2.3.0.jar:2.3.0]
  at com.netflix.loadbalancer.reactive.LoadBalancerCommand$1.call(LoadBalancerCommand.java:180) ~[ribbon-loadbalancer-2.3.0.jar:2.3.0]
  at rx.Observable.unsafeSubscribe(Observable.java:10327) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeConcatMap.call(OnSubscribeConcatMap.java:94) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeConcatMap.call(OnSubscribeConcatMap.java:42) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.Observable.unsafeSubscribe(Observable.java:10327) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OperatorRetryWithPredicate$SourceSubscriber$1.call(OperatorRetryWithPredicate.java:127) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.schedulers.TrampolineScheduler$InnerCurrentThreadScheduler.enqueue(TrampolineScheduler.java:73) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.schedulers.TrampolineScheduler$InnerCurrentThreadScheduler.schedule(TrampolineScheduler.java:52) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OperatorRetryWithPredicate$SourceSubscriber.onNext(OperatorRetryWithPredicate.java:79) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OperatorRetryWithPredicate$SourceSubscriber.onNext(OperatorRetryWithPredicate.java:45) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.util.ScalarSynchronousObservable$WeakSingleProducer.request(ScalarSynchronousObservable.java:276) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.Subscriber.setProducer(Subscriber.java:209) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.util.ScalarSynchronousObservable$JustOnSubscribe.call(ScalarSynchronousObservable.java:138) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.util.ScalarSynchronousObservable$JustOnSubscribe.call(ScalarSynchronousObservable.java:129) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:48) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:30) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:48) ~[rxjava-1.3.8.jar:1.3.8]
```

问题原因

服务名为{your-service-name}的服务在Eureka中不存在或健康状态不为健康。

解决方案

检查您服务名为{your-service-name}的服务注册情况。确保客户端成功注册，注册成功后，客户端将打印如下日志。

注册完成后，到Eureka控制台再次确认服务的注册情况，在服务管理-服务列表可以看到注册的服务，点击服务详情，可以查看对应的ip端口信息。

当服务消费者和服务提供者均在Eureka注册成功，在控制台服务列表中显示时，服务调用才有可能成功。

如何解决消费者无法获取最新提供者信息的问题？

问题现象

您已经试图将服务消费者和服务提供者注册到Eureka，且在Eureka控制台已确保服务被成功注册，但仍报错Load balancer does not have available server for client:{your-service-name}，如下图所示。

```
2024-01-03 14:10:10.628 ERROR 33040 --- [nio-9087-exec-1] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request
 processing failed; nested exception is java.lang.RuntimeException: com.netflix.client.ClientException: Load balancer does not have available server for client: eureka-provider-other] with root cause

com.netflix.client.ClientException: Load balancer does not have available server for client: eureka-provider-other
  at com.netflix.loadbalancer.LoadBalancerContext.getServerFromLoadBalancer(LoadBalancerContext.java:483) ~[ribbon-loadbalancer-2.3.0.jar:2.3.0]
  at com.netflix.loadbalancer.reactive.LoadBalancerCommand$1.call(LoadBalancerCommand.java:184) ~[ribbon-loadbalancer-2.3.0.jar:2.3.0]
  at com.netflix.loadbalancer.reactive.LoadBalancerCommand$1.call(LoadBalancerCommand.java:180) ~[ribbon-loadbalancer-2.3.0.jar:2.3.0]
  at rx.Observable.unsafeSubscribe(Observable.java:10327) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeConcatMap.call(OnSubscribeConcatMap.java:94) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeConcatMap.call(OnSubscribeConcatMap.java:42) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.Observable.unsafeSubscribe(Observable.java:10327) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OperatorRetryWithPredicate$SourceSubscriber$1.call(OperatorRetryWithPredicate.java:127) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.schedulers.TrampolineScheduler$InnerCurrentThreadScheduler.enqueue(TrampolineScheduler.java:73) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.schedulers.TrampolineScheduler$InnerCurrentThreadScheduler.schedule(TrampolineScheduler.java:52) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OperatorRetryWithPredicate$SourceSubscriber.onNext(OperatorRetryWithPredicate.java:79) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OperatorRetryWithPredicate$SourceSubscriber.onNext(OperatorRetryWithPredicate.java:45) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.util.ScalarSynchronousObservable$WeakSingleProducer.request(ScalarSynchronousObservable.java:276) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.Subscriber.setProducer(Subscriber.java:209) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.util.ScalarSynchronousObservable$JustOnSubscribe.call(ScalarSynchronousObservable.java:138) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.util.ScalarSynchronousObservable$JustOnSubscribe.call(ScalarSynchronousObservable.java:129) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:48) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:30) ~[rxjava-1.3.8.jar:1.3.8]
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:48) ~[rxjava-1.3.8.jar:1.3.8]
```

或是您的服务提供者信息（如：地址）发生了变化，但消费者从服务端获取到的仍为旧信息，导致服务调用失败。

问题原因

正常配置下，Eureka客户端会以一定的时间间隔定期向服务端获取服务的增量更新信息。当服务消费者端的数据获取间隔（eureka.client.registry-fetch-interval-seconds）设置过长时，消费者将无法及时感知到服务端最新数据的更新。

解决方案

将您服务消费者客户端配置中的数据获取间隔（eureka.client.registry-fetch-interval-seconds）时间改小，如改为5秒，如下图所示。但过小的时间间隔会增大客户端的工作负担。

```
eureka.client.registry-fetch-interval-seconds=5
```

云原生网关

计费类

如何为开通的云原生网关续费?

实例计费模式为包年包月，当实例生命周期达到截至使用时间时，云原生网关实例会处于到期停机的状态，然后保留15天，保留期内，用户可以通过续费恢复实例；若超出保留期实例将被真正地清理。

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏云原生网关 -> 网关列表 ；
4. 在实例列表点击“续订”；

5. 进入实例续订页，选择 续订时长 ，提交订单，完成续费；

管理类

云原生网关支持哪些服务来源？

云原生网关当前支持从天翼云Nacos注册中心或者云容器引擎集群发现服务；添加完服务来源之后，可以从Nacos或者云容器引擎指定命名空间中发现服务，配置为网关代理转发的目标服务；网关通过监听指定namespace下的服务，实时动态感知服务节点列表变化，实现网关的动态路由转发能力。

为什么添加服务来源的时候只能看到部分N acos 或者云容器引擎实例？

云原生网关属于某一个指定的vpc网络，选择Nacos或者云容器引擎作为服务来源时会过滤集群列表，只能选择和当前网关同一vpc下的集群作为服务发现来源。

云原生网关如何支持https访问？

通过域名管理功能可以配置访问域名，对于有https访问需求的场景，可以配置域名的https证书；通过网关节点的27154端口或者外部ELB的https端口（需要配置转发规则转发到网关的27154端口）访问，实现https加密通信。

路由配置支持哪些匹配规则？

当前路由支持根据请求的域名、路径、HTTP方法、header、query匹配；路径匹配支持精确匹配和前缀匹配，对于前缀匹配/abc*可以匹配请求/abc，/abcd/ef，/abc/def/cc。

一个请求同时匹配到多个路由时，最终会使用哪条路由规则？

uri精确匹配优先级最高；对于多个uri前缀匹配同时命中的情况，匹配深度较高的优先；相同uri的路由，如果存在其他匹配条件，则优先按其他条件匹配；其他条件也都同时匹配的情况下，优先级数字较大的路由优先匹配。

为什么Mock路由添加应用授权时指定的过期时间不生效？

目前应用授权中过期时间的校验阶段与路由执行mock策略时的阶段是同一阶段，且mock的优先级高于应用授权过期时间的校验，因此当对mock路由指定应用的过期时间时，会忽略过期时间的校验。

ip访问控制功能同时支持黑名单和白名单方式吗？

ip访问控制不支持同时开启黑名单和白名单。

链路追踪为什么没有数据？

在基础信息->功能设置页面确认当前是否开启了链路追踪功能以及当前采样率设置；没有开启链路追踪或者采样率过低可能导致看不到链路追踪数据。

云原生网关如何开启IPv6访问？

1、确保当前云原生网关实例所处的vpc已具备IPv6子网；如不具备，跳转至vpc网络控制台，创建或者开启IPv6子网； 2、开启实例IPv6，在实例订购页，网络配置中选择某个vpc，勾选IPv6（第一步具备时才可勾选），选择好其他配置后提交订购即可；

IPv4如何升级IPv6?

从云原生网关控制台，实例列表中更多->点击开启IPv6，确定提交。如此实例的vpc不具备IPv6子网，会报错提示用户，提交完成后按提示信息，刷新实例详情，查看基础信息中的IPv6访问信息。

如何通过IPv6访问实例？

网关实例的IPv6访问有两种方式：1、通过内网IPv6地址访问，访问地址在实例详情中的基础信息页中已展示。2、通过ELB访问，此方式需要确保ELB实例本身已开启IPv6访问，目前云原生网关实例暂只支持ELB的内网IPv6访问。

微服务治理中心

计费类

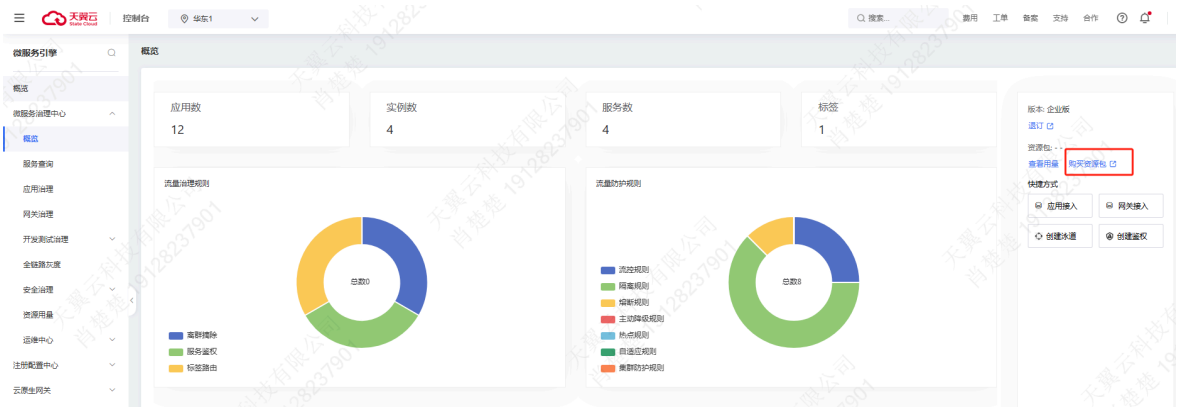
如何为开通的微服务治理中心购买资源包抵扣?

您在开通按需资源后，可购买对应的资源包用量，购买资源包后，您使用产品的用量将会使用资源包进行抵扣，超出资源包量的部分将按需计费，资源包到期后资源包的用量将失效，不再用于抵扣。

操作步骤

1. 进入微服务引擎MSE控制台；
2. 在顶部菜单栏选择资源池；
3. 单击左侧导航栏微服务治理中心 ；

4. 在概览页面点击“购买资源包”；



5. 进入资源包订购页面，选择所需规格，点击下一步，即可完成资源包订购。

6. 资源包订购后，可在资源用量菜单中查看使用情况。

操作类

服务启动成功却没有在微服务治理中心展示？

服务启动成功后，没有在微服务治理中心控制台展示。主要有以下两个排查方法：

1、服务框架自查。确定自身服务是否使用了SpringCloud或Dubbo框架，若使用其他微服务框架可能不会在微服务治理中心控制台展示应用。

2、检查网络状态。检查当前服务所在的vpc与启动时配置的mse.msc.endpoint网络互通，若网络不通，跳转至“概览”页，检查当前vpc是否已接入微服务治理中心。

购买微服务治理以后，为何还需要为VPC开通微服务治理通道？

用户所使用的vpc默认与微服务治理中心的控制面网络不互通，在网络不通的情况下无法进行服务治理，所以需要为使用微服务治理中心的vpc开通微服务治理通道。

在云容器引擎发布应用，为何没有加入微服务治理中心？

检查是否为集群安装cubems插件，其次检查是否配置对应的pod标签。

在应用变更阶段，服务查询展示的实例数与实际数不一致？

在实例下线时，微服务治理中心对服务下线的感知会有一定延迟，可能会出现展示的实例数与实际数不一致的情况。

管理类

什么是实例数？

实例数指的是接入微服务治理中心的进程个数，通过MseAgent启动一个Java进程，则可以认为是一个实例数。无论是资源包还是按量计费，都是以接入微服务治理中心的实例数为统计单位。

应用数、实例数和服务数三者的区别是什么？

应用与实例是一对多的关系，一个应用可以部署多个实例。服务数指的是应用作为提供者，在注册中心注册的服务，对于SpringCloud框架的应用，由于提供的是http服务，所以一个应用的SpringCloud服务数一般为1，对于Dubbo框架的应用，由于提供的是dubbo服务，一个接口即可理解为一个服务，所以一个应用的Dubbo服务数可以为n。

金丝雀发布、标签路由和全链路灰度有什么区别？

金丝雀发布、标签路由和全链路灰度都可以将流量通过请求参数路由到指定的打标应用上。三者区别主要在使用场景上面不同，金丝雀发布主要应用在应用发布上，当应用需要更新时，可以通过金丝雀发布的功能，将小流量引入到新版本的服务，等功能验证成功后，再全量发布；标签路由可以作为一个基础能力，用户可以在此功能上拓展，更加灵活的控制流量路由；全链路灰度主要应用在多版本研发、多版本迭代的场景上，与金丝雀发布和标签路由不同的是，全链路灰度作用于多个应用，而金丝雀发布和标签路由主要作用于单个应用。

无损上下线没有生效？

无损上线：首先检查使用的微服务框架是否是SpringCloud或Dubbo框架，其次检查提供者和消费者是否都已经正常接入了微服务治理中心。再检查延迟注册时间是否合适，是否已经超过延迟注册的时间。

无损下线：首先检查使用的微服务框架是否是SpringCloud或Dubbo框架，其次检查提供者和消费者是否都已经正常接入了微服务治理中心。再检查服务下线时是否使用的非kill -9命令。

离群摘除没有生效？

检查设置的离群摘除规则，判断当前异常数是否满足设置的QPS下限（QPS下限需要以时间窗口维度统计，如设置为1，SpringCloud时间窗口维度是10s，则需要满足10s内至少有10个请求），判断是否满足错误率下线，判断是否满足实例的最大摘除比例。

在全链路灰度中，在灰度泳道中没有下游应用，流量会怎么路由？

若在灰度泳道中没有下游应用，流量会优先路由到基线泳道（即未打标泳道）。

如何感知应用发生了服务治理事件？

当应用触发了服务治理事件时，会记录一条事件记录。如发生无损上下线、离群摘除、推空保护时，会在事件中心记录事件。