# SOFAStack

## 服务网格
## 使用指南

产品版本：AntStack Plus 1.11.0

文档版本：20221021

蚂蚁集团
ANT GROUP

# 法律声明


## 蚂蚁集团版权所有©2022，并保留一切权利。

未经蚂蚁集团事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。


## 商标声明

蚂蚁集团 ANT GROUP及其他蚂蚁集团相关的商标均为蚂蚁集团所有。本文档涉及的第三方的注册商标，依法由权利人所有。

## 免责声明

由于产品版本升级、调整或其他原因，本文档内容有可能变更。蚂蚁集团保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在蚂蚁集团授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过蚂蚁集团授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

# 通用约定

| 格式 | 说明 | 样例 |
| --- | --- | --- |
| 危险 | 该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 危险<br>重置操作将丢失用户配置数据。 |
| 警告 | 该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 警告<br>重启操作将导致业务中断，恢复业务时间约十分钟。 |
| 注意 | 用于警示信息、补充说明等，是用户必须了解的内容。 | 注意<br>权重设置为0，该服务器不会再接受新请求。 |
| 说明 | 用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。 | 说明<br>您也可以通过按Ctrl+A选中全部文件。 |
| > | 多级菜单递进。 | 单击**设置**> **网络**> **设置网络类型**。 |
| **粗体** | 表示按键、菜单、页面名称等UI元素。 | 在**结果确认**页面，单击**确定**。 |
| `Courier字体` | 命令或代码。 | 执行 `cd /d C:/window` 命令，进入Windows系统文件夹。 |
| *斜体* | 表示参数、变量。 | `bae log list --instanceid` *Instance_ID* |
| [] 或者 [a\|b] | 表示可选项，至多选择一个。 | `ipconfig [-all\|-t]` |
| {} 或者 {a\|b} | 表示必选项，至多选择一个。 | `switch {active\|stand}` |

# 目录

1.产品简介 07
1.1. 什么是服务网格 07
1.2. 产品优势 08
1.3. 产品架构 09
1.4. 应用场景 10
2.应用快速接入 12
2.1. 服务网格 SDK 12
2.1.1. SDK 版本说明 12
2.1.2. SOFABoot 服务接入服务网格 SDK 13
2.1.3. Dubbo 服务接入服务网格 SDK 15
2.1.4. Spring Cloud 服务接入服务网格 SDK 16
2.2. 应用注入 Sidecar 17
2.2.1. 通过 CAFE 注入 Sidecar 17
2.2.2. 通过 Mesh 控制台注入 Sidecar 21
3.集群管理 27
3.1. 创建集群 27
3.2. 查看集群 30
3.3. 开通入口网关 30
3.4. 开通 Mesh 35
3.5. 编辑或删除集群 37
4.服务目录 40
4.1. 概述 40
4.2. 查看服务目录 40
4.3. 管理服务目录 45
4.4. 手动添加应用 46
4.5. 查看应用依赖 48

4.6. 查看实时拓扑 49
5.Sidecar 管理 54
5.1. Sidecar 版本管理 54
5.2. Sidecar 注入规则 55
5.3. Sidecar 动态配置 64
6.注册中心管理 67
6.1. 添加注册中心 67
6.2. 使用注册中心 68
6.3. 编辑或删除注册中心 72
7.插件管理 75
7.1. 协议插件 75
7.2. 功能插件 79
7.3. 协议转换 82
7.4. 最佳实践 85
8.服务治理 102
8.1. 服务限流 102
8.1.1. 管理服务限流规则 102
8.1.2. 验证服务限流规则 108
8.2. 服务路由 113
8.2.1. 管理服务路由规则 113
8.2.2. 验证服务路由规则 120
8.3. 服务熔断 123
8.3.1. 管理服务熔断规则 123
8.3.2. 验证服务熔断规则 127
8.4. 服务降级 131
8.4.1. 管理服务降级规则 131
8.4.2. 验证服务降级规则 133
8.5. 故障注入 134

8.5.1. 管理故障注入规则 134
8.5.2. 验证故障注入规则 137
8.6. 故障隔离 140
8.6.1. 管理故障隔离规则 140
8.6.2. 验证故障隔离规则 143
9.安全管理 148
9.1. 通信加密 148
9.2. 请求防篡改 151
9.3. 访问授权 154
10.审计目录 158
11.动态配置 160
11.1. 概述 160
11.2. 动态配置快速入门 160
11.3. 新增动态配置 163
11.4. 推送动态配置 165
11.5. 使用注解标识配置类 166
11.6. 导出和导入动态配置 167
11.7. 教程示例：使用动态配置 170
12.账号管理 173
13.角色管理 175

# 1.产品简介

## 1.1. 什么是服务网格

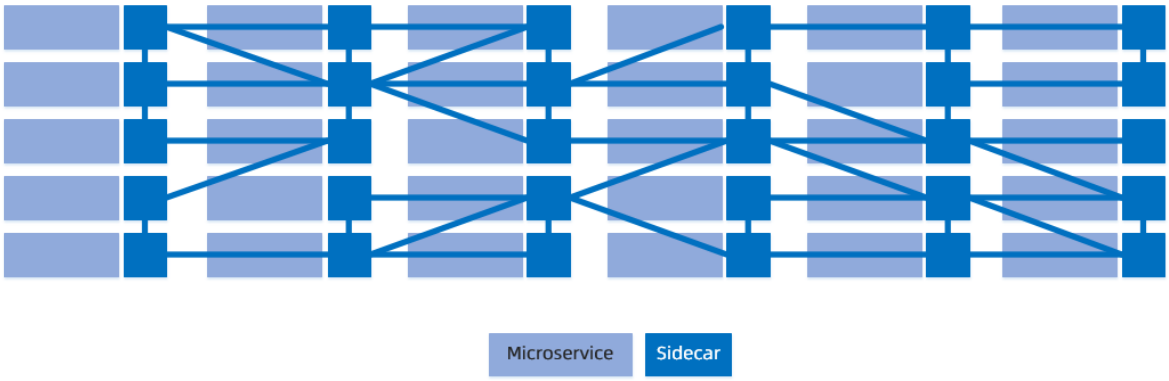
服务网格（SOFAStack Mesh）是蚂蚁集团自主研发的基于金融级生产实践的增强版服务网格平台，将传统微服务和 Service Mesh 技术进行了深度融合，其核心技术经过了蚂蚁集团的大规模生产实践验证。它深度、无缝对接了 SOFAStack 经典应用服务和容器应用服务，为客户提供了简单易用的 Service Mesh 架构的支撑平台。

### SOFAStack 服务网格概述

SOFAStack 服务网格（简称服务网格）= 传统微服务 + Service Mesh。

服务网格旨在提供与平台无关、语言无关、轻量无侵入的云原生分布式架构解决方案，在兼容 Kubernetes 和 Istio 生态的基础上还能支持虚拟机场景，同时支持 SOFA、Dubbo、SpringCloud 微服务框架的服务治理能力。在整个接入过程中，无需修改业务代码，帮助企业以最小的成本完成云原生落地，加速数字化转型。

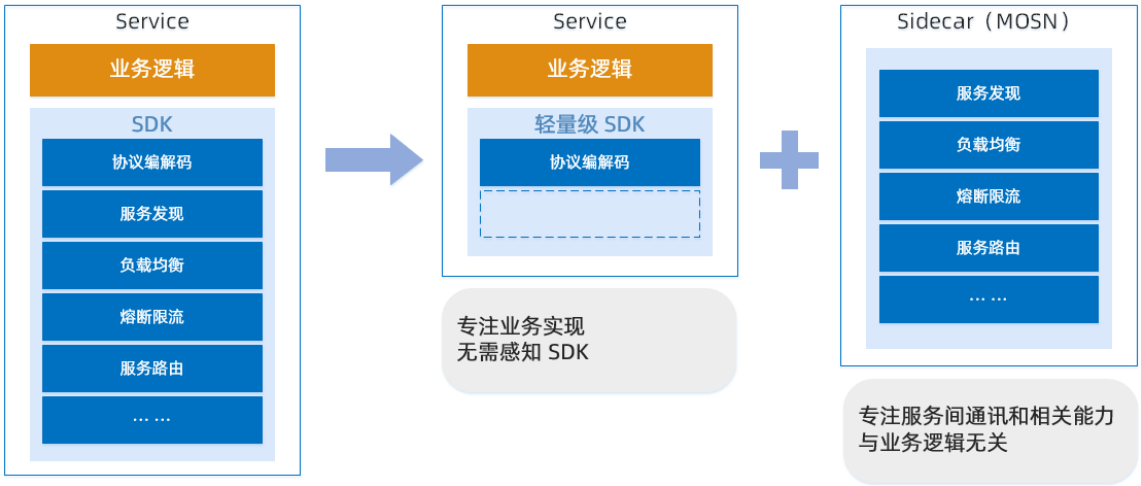
### Service Mesh 图解




从一个全局视角来看，Sidecar 连接成网，组成 Service Mesh。

TCP 协议催生了分布式系统，分布式系统催生了微服务，Service Mesh 就是下一代微服务技术的代名词，是微服务时代的 TCP 协议。Service Mesh 以 Sidecar 形式，将服务治理从业务逻辑中剥离，并拆解为独立进程，实现异构系统的统一治理和网络安全。

### Sidecar 模式图解




Service Mesh 将 SDK 拆解为单独进程，并以 Sidecar 模式进行部署，使得业务进程专注于业务逻辑。

Sidecar 是一个轻量级的网络代理，它们与应用程序部署在一起，对所有流入与流出的网络请求进行拦截，实现各种网络策略，例如：服务发现与负载均衡、流量拆分、故障注入、熔断器以及分阶段发布等功能。

## 功能一览

服务网格包含以下子产品和功能，可以进行各种服务管理和监控操作。

- 服务管控

  支持异构服务的统一注册与服务详情查看，包括其基本信息、服务提供者以及消费者信息等。

- 服务治理

  支持服务限流、服务路由、服务熔断降级、服务鉴权，提供了精细化调配的服务路由与服务限流功能，保证应用高可用，保障业务持续运行。

- Sidecar 管理

  支持查看及管理当前工作空间中的 Sidecar 实例，提供了 Sidecar 状态、Pod 状态、Sidecar 注入时间等信息。

# 1.2. 产品优势

在传统单体式架构向微服务架构迁移的过程中，随着应用微服务数量的增加，微服务间的通信、监控以及安全性的管理成为新的挑战。服务网格作为应用与基础设施的桥梁，突破传统的 SDK 接入方式，以对应用透明的方式处理服务之间、服务与基础设施间的通信，实现应用研发和基础设施最大程度地解耦，让应用研发只需专注于业务开发。

## 技术业务分离

在之前，微服务体系都是由中间件团队提供一个 SDK 给业务应用使用，在 SDK 中会集成各种服务治理的能力，如：服务发现、负载均衡、熔断限流、服务路由等。有了服务网格之后，我们就可以把 SDK 中的大部分能力从应用中剥离出来，拆解为独立进程，以 Sidecar 的模式部署。通过将服务治理能力下沉到基础设施，可以让业务更加专注于业务逻辑，中间件团队则更加专注于各种通用能力，真正实现独立演进，透明升级，提升整体效率。

## 异构服务治理

既能借助蚂蚁集团久经考验的微服务框架 SOFA 在云上构建微服务应用，也可以支持原生 Dubbo 和 Spring Cloud 上云，无需构建 ZooKeeper、Eureka、Consul 等微服务依赖的自建服务，极大降低运维成本。

### 轻量无侵入接入

业务应用系统通过 Service Mesh 技术架构轻量级接入，实现对应用无侵入的服务注册与服务治理方案，减少改造成本。

### 跨平台支持

该方案支持容器平台、虚拟机平台，能够满足企业用户未容器化的场景对 Service Mesh 架构转型的需求。

### 网络安全

通过网格安全，可以更方便地实现应用的身份标识和访问控制，辅之以数据加密，就能实现全链路可信，从而使得服务可以运行于零信任网络中，提升整体安全水位。

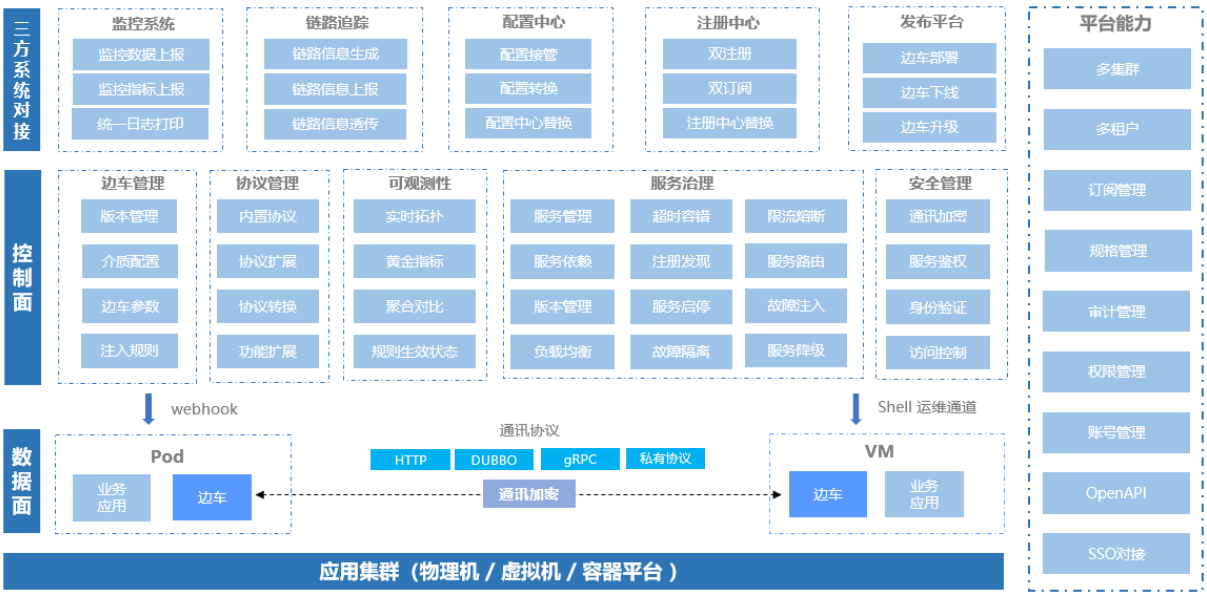
### 简单易用易维护

SOFAStack 服务网格提供集中式图形化易操作的管理平台，支持多租户管理能力，满足企业级高级特性需求，简化分布式应用的服务管理、服务治理、可观察性等能力，让用户便捷的对应用服务统一管理和治理。

## 1.3. 产品架构

服务网格的产品架构逻辑上分为控制层面和数据层面。

- 控制层面：管理代理（蚂蚁自研 MOSN），用于管理流量路由、运行时策略执行等。
- 数据层面：由一系列代理（蚂蚁自研 MOSN）组成，用于管理和控制服务之间的网络通信。

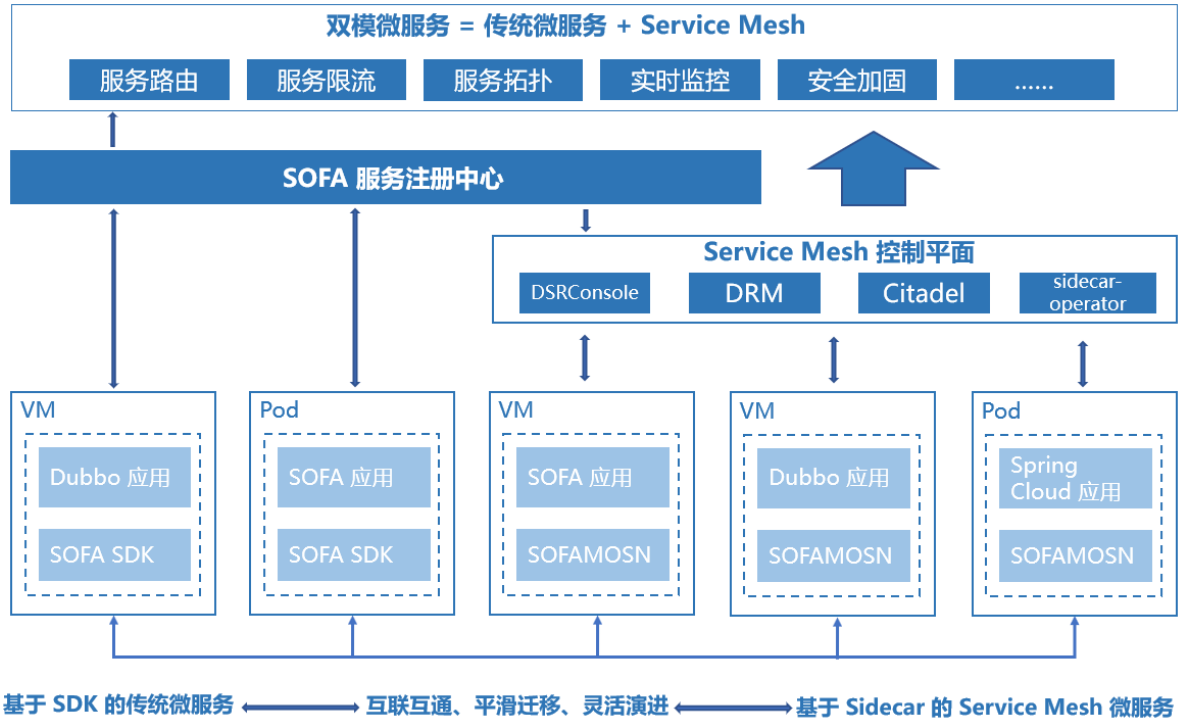



### 系统架构

SOFAStack 服务网格结合了 SOFAStack 微服务的优势，将 SOFAStack 微服务和 Service Mesh 双剑合璧，即基于 SDK 的传统微服务可以和基于 Sidecar 的 Service Mesh 微服务实现下列目标：

- 互联互通：两个体系中的应用可以相互访问。
- 平滑迁移：应用可以在两个体系中迁移，对于调用该应用的其他应用，做到透明无感知。

- 灵活演进：在互联互通和平滑迁移实现之后，我们就可以根据实际情况进行灵活的应用改造和架构演进。




在控制面上，引入了 DRM 实现配置的下发（如服务路由规则），在服务发现上保留了独立的 SOFA 服务注册中心。

在数据面上，使用了自研的 SOFA MOSN，不仅支持 SOFA 应用，同时也支持 Dubbo 和 Spring Cloud 应用。

在部署模式上，我们不仅支持容器 K8s，同时也支持虚拟机场景。

# 1.4. 应用场景

依据服务网格特点和优势，服务网格适用于以下应用场景。

## 服务治理与业务逻辑解耦

在运行时，SDK 和业务应用的代码是混合在一个进程中运行的，耦合度非常高，会产生如下问题：

- 升级成本高：每次升级都需要业务应用修改 SDK 版本号，重新发布。在业务飞速发展的时候，更倾向于专注业务，不希望分散精力做升级等事情。
- 版本碎片化严重：由于升级成本高，但中间件还是会向前发展，久而久之，就会导致线上 SDK 版本各不统一、能力参差不齐，造成很难统一治理。
- 中间件演进困难：由于版本碎片化严重，导致中间件向前演进过程中需要兼容各种老版本的逻辑，无法实现快速迭代。

使用 Service Mesh 后，您可以将 SDK 中的大部分能力从应用中剥离拆解为独立进程，以 Sidecar 的模式部署。通过将服务治理能力下沉到基础设施，可以让业务更加专注于业务逻辑，中间件团队则更加专注于各种通用能力，真正实现独立演进、透明升级，提升整体效率。

## 多语言、多协议支持

随着新技术的发展和人员更替，在同一家公司中往往会出现使用各种不同语言、不同框架的应用和服务，为了能够统一管控这些服务，以往的做法是为每种语言、每种框架都重新开发一套完整的 SDK，维护成本非常高，而且对中间件团队的人员结构也带来了很大的挑战。

有了服务网格之后，通过将主体的服务治理能力下沉到基础设施，多语言的支持就轻松很多了，只需要提供一个非常轻量的 SDK，甚至很多情况都不需要一个单独的 SDK，就可以方便地实现多语言、多协议的统一流量管控、监控等治理需求。

## 云原生架构转型助力

从单体应用到微服务架构改造以及全面容器化的云原生架构基础往往带来很高的改造成本。SOFAStack 服务网格可以满足未容器化的虚拟机部署方案，也可以兼容过渡阶段的部分容器化和虚拟化混合部署的场景，加速企业云原生架构转型。

## 金融场景网络安全

当前很多公司的微服务体系建设都建立在内网可信的假设之上，然而这个原则在当前大规模上云的背景下可能显得有点不合时宜，尤其是涉及到一些金融场景的时候。

通过服务网格可以更方便地实现应用的身份标识和访问控制，辅之以数据加密，就能实现全链路可信，从而使得服务可以运行于零信任网络中，提升整体安全水位。

# 2.应用快速接入

## 2.1. 服务网格 SDK

### 2.1.1. SDK 版本说明

本文提供了服务网格 sofa-registry-cloud-all SDK 的版本信息，包含其发布时间、更新点等，以便您按需获取适用的 SDK。

应用服务接入 SOFA 注册中心，只需在 endpoint 下的 `pom.xml` 文件中引入以下依赖：

```
<!-- 需将 x.x.x 替换为 SDK 具体的版本号 -->
<dependency>
   <groupId>com.alipay.sofa</groupId>
   <artifactId>sofa-registry-cloud-all</artifactId>
   <version>x.x.x</version>
</dependency>
```

#### 1.2.8 (2020-03-25)

**新增**

- 支持取消代理接口不走 Service Mesh。
- 支持 Dubbo 框架非 Dubbo 协议注册与发现。
- 支持服务鉴权，透传调用方应用名。

**修复**

服务多次暴露场景下，错误使用 path 构建 dataId 导致寻址错误和 RPC 调用失败问题。

#### 1.2.7 (2020-03-16)

> **说明：** 不推荐使用此版本。

**新增**

支持 Dubbo 泛化调用。

**修复**

- 同一个接口多次暴露或者消费构建 dataId 错误，导致获取不到服务提供者问题。
- 通过 API 创建注册中心时，触发多次向 MOSN 注册 appInfo，导致部分服务或者订阅数据丢失问题。

#### 1.2.6 (2020-03-05)

> **说明：** 不推荐使用此版本。

**修复**

- 1.2.5 版本中 Gateway 调用依赖时的 `global filter class not found` 错误。

- 商业版 MOSN 版本小于等于 1.2.4 中，服务推送 path 可能不正确的问题。支持 Dubbo 分组版本调用。

### 1.2.5 (2020-03-03)

> **说明：** 不推荐使用此版本。

**新增**

支持 Spring Cloud Gateway 服务调用接入 Mesh。

**更新**

- 升级 fastjson 版本至 1.2.66（修复安全漏洞版本）。
- 提升测试覆盖率至 75.94%。
- 要求的 JDK 版本为 JDK6 及以上。

**修复**

SDK 输出 provider 信息错误，修正 log 输出级别。

### 1.2.4 (2020-02-19)

**新增**

支持 Dubbo 框架无缝接入 Servce Mesh。

**更新**

兼容 Dubbo 2.6.x 和 Dubbo 2.7.x。

**修复**

Mesh Registry 重复 Bean 定义导致 Spring Cloud 应用无法启动的问题。

### 1.2.3 (2020-01-02)

**更新**

- 优化 Spring Cloud 服务注册与订阅，减少与 MOSN 进行 HTTP 请求交互。
- 优化第一次与 MOSN 订阅无数据时，触发重试订阅数据。
- 优化 SDK 与 Eureka 兼容性，引入 SDK 无需排除 Eureka。
- 更改 LOG JAR 包依赖为 TEST，防止和业务日志框架包冲突。

# 2.1.2. SOFABoot 服务接入服务网格 SDK

在使用服务网格之前，您需要先接入服务网格 SDK，本文介绍如何将 SOFABoot 服务接入。

## 操作步骤

1. 升级 SOFABoot 依赖版本至最新版本。

   版本信息，请参见 SOFABoot 版本说明。

```
<parent>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>sofaboot-enterprise-dependencies</artifactId>
    <version>3.x.x</version>
</parent>
```

2. 通过以下任一方式，添加应用启动参数：
   - 在配置文件中增加配置：

     在 `app/web/src/main/resources/config/application.properties` 文件中新增如下配置：

     ```
     com.alipay.env=shared
     com.alipay.instanceid=                    //填写实例 ID。
     com.alipay.sofa.rpc.bolt-port=12201 //SOFA 应用的 Bolt 端口是 12200，与 Sidecar 中的端口冲突，需要修改应用的端口。
     ```

     您可以在中间件中台（OSP）获取实例 ID，如下图所示：

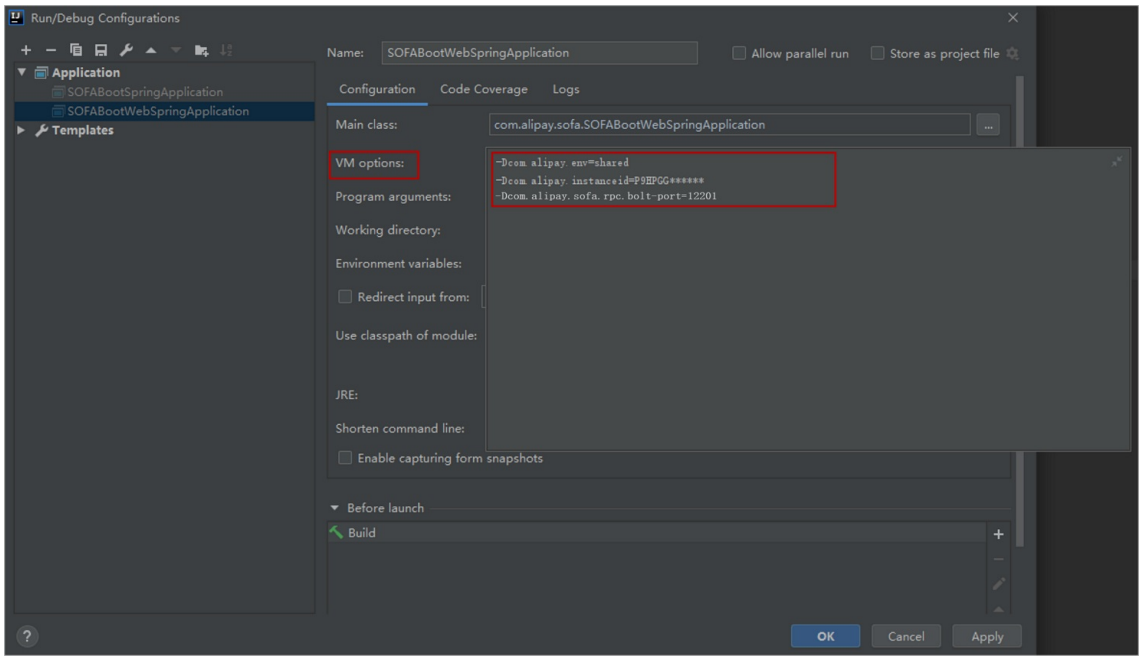
     

   - 指定 JVM 启动参数：

     > **说明**
     >
     > 这种方式优先级高于代码。

     ```
     -Dcom.alipay.env=shared
     -Dcom.alipay.instanceid=                    //填写实例 ID。
     -Dcom.alipay.sofa.rpc.bolt-port=12201 //SOFA 应用的 Bolt 端口是 12200，与 Sidecar 中的端口冲突，需要修改应用的端口。
     ```

在 IntellIJ IDEA 中的配置示例如下截图：




## 2.1.3. Dubbo 服务接入服务网格 SDK

在使用服务网格之前，您需要先接入服务网格 SDK，本文介绍如何将 Dubbo 服务接入。

### 操作步骤

1. 引入 `sofa-registry-cloud-all` SDK 依赖。

   版本信息，请参见 SDK 版本说明。

```
<dependency>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>sofa-registry-cloud-all</artifactId>
    <!-- 替换 x.x.x 为该 SDK 最新版本号 -->
    <version>x.x.x</version>
</dependency>
```

2. 配置 Dubbo 的注册中心，使用 dsr： `<dubbo:registry address="dsr://dsr"/>` 。

3. 通过以下任一方式，添加应用启动参数：

   - 在 `dubbo.properties` 中配置如下：

```
com.alipay.instanceid=                    //填写实例 ID。
```

您可以在中间件中台（OSP）获取实例 ID，如下图所示：

- 指定 JVM 启动参数：

```
-Dcom.alipay.env=shared
-Dcom.alipay.instanceid=                    //填写实例 ID。
```

- 指定系统环境变量：

```
SOFA_INSTANCE_ID=                    // 填写实例 ID。
```

# 2.1.4. Spring Cloud 服务接入服务网格 SDK

在使用服务网格之前，您需要先接入服务网格 SDK，本文介绍如何将 Spring Cloud 服务接入。

## Spring Cloud 服务

1. 引入以下 `sofa-registry-cloud-all` SDK 依赖。

   版本信息，请参见 SDK 版本说明。

```
<dependency>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>sofa-registry-cloud-all</artifactId>
    <!-- 替换 x.x.x 为该 SDK 最新版本号 -->
    <version>x.x.x</version>
</dependency>
```

2. 通过以下任一方式，添加应用启动参数。
   - 在应用 YAML 文件中指定以下参数：

```
sofa:
registry:
  discovery:
        instanceId:        //填写实例 ID。
```

您可以在中间件中台（OSP）获取实例 ID，如下图所示：

- 指定 JVM 启动参数：

```
-Dcom.alipay.instanceid=                    //填写实例 ID。
```

- 指定系统环境变量：

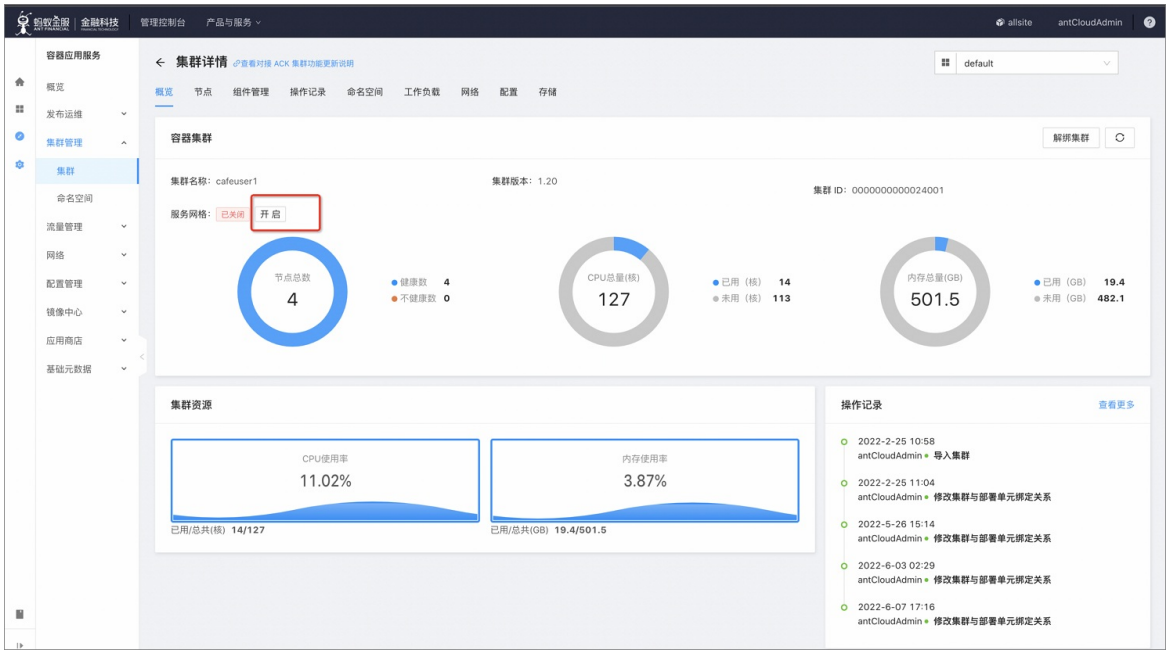
```
SOFA_INSTANCE_ID=                 //填写实例 ID。
```

# 2.2. 应用注入 Sidecar

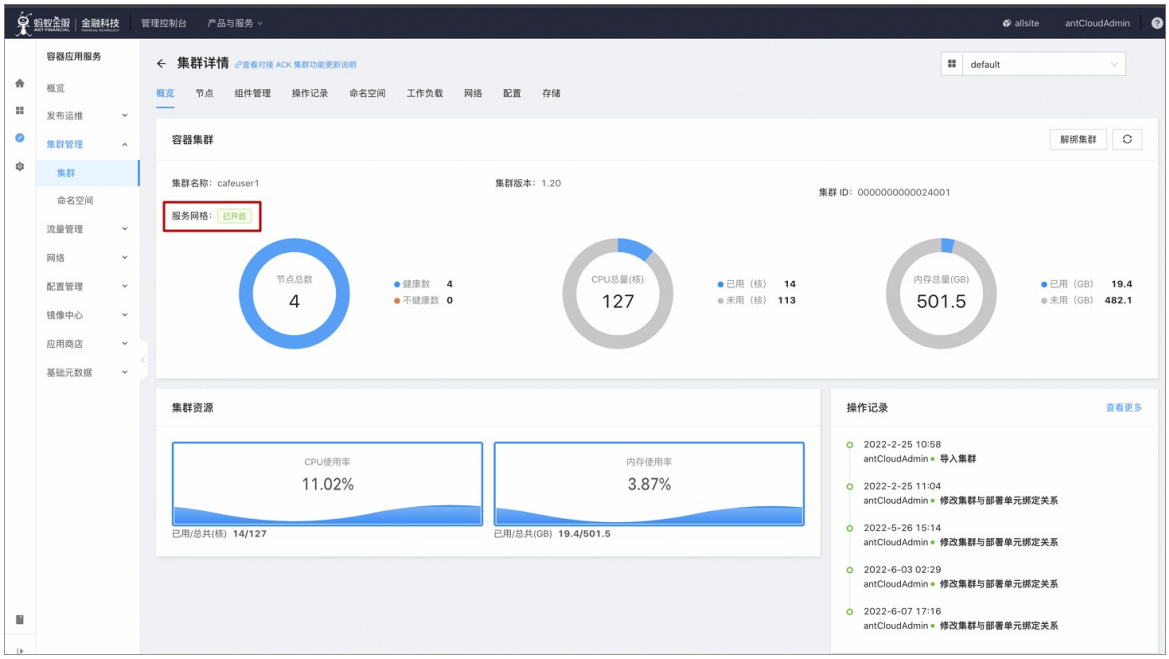
## 2.2.1. 通过 CAFE 注入 Sidecar

### 步骤一：开通服务网格。

1. 登录 CAFE 控制台。
2. 选择 **集群管理** > **集群**。
3. 在 **概览** 页签的 **容器集群** 区域，开启服务网格。



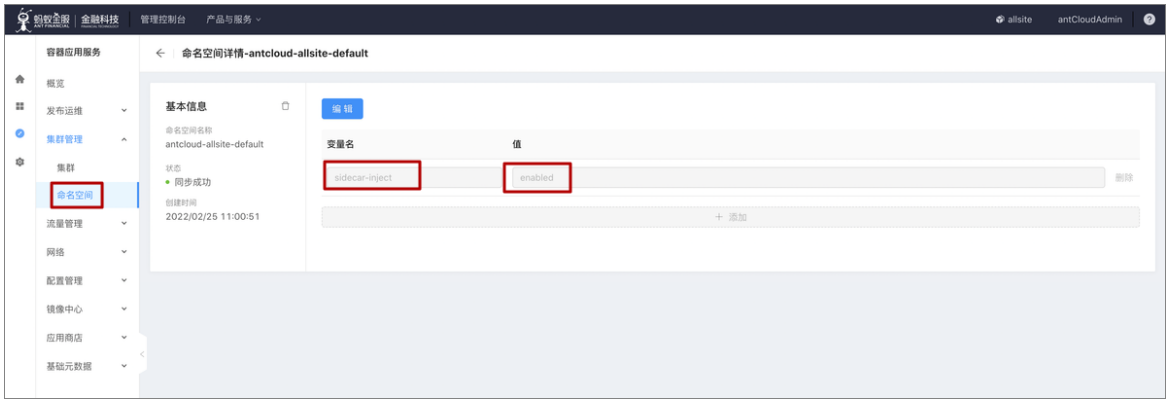

开通成后，显示如下：




服务网格控制台的 **集群管理** 页面也会新增一条集群记录，状态显示为 **已开通**。

## 步骤二：为命名空间增加 label

1. 进入目标命名空间，然后单击 **编辑**。
2. 新增一个 label，变量名为：sidecar-inject；变量值为：enabled。



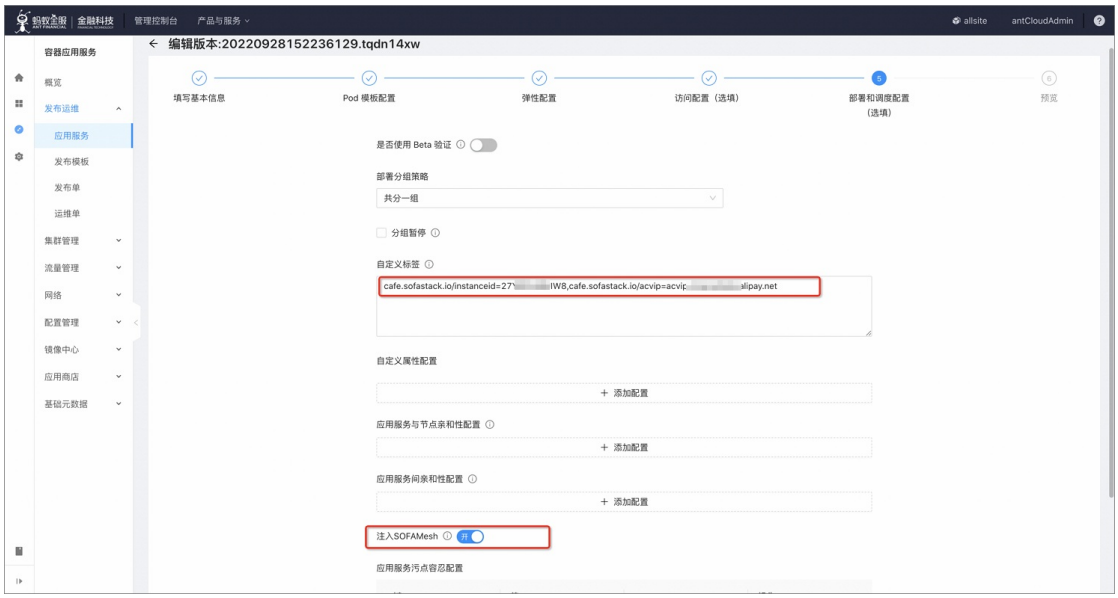

## 步骤三：编辑并发布应用

1. 编辑应用服务。
   i. 在左侧导航栏选择 **发布运维** > **应用服务**。

ii. 编辑目标应用服务，在 **部署和调度配置** 步骤添加自定义标签，并开启 **注入 SOFAMesh**。

标签格式如下：

```
cafe.sofastack.io/instanceid=27Y***IW8,cafe.sofastack.io/acvip=acvip-****alipay.net
```

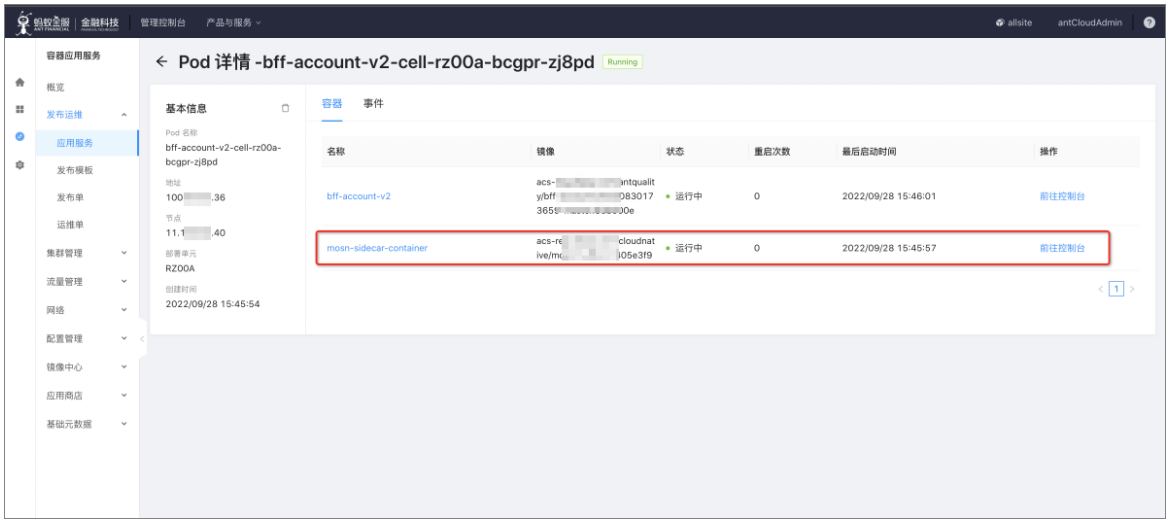



**说明**

instanceid（实例标识）和 acvip 地址可在您的 OSP 控制台获取，示例如下：

2. 发布应用服务。

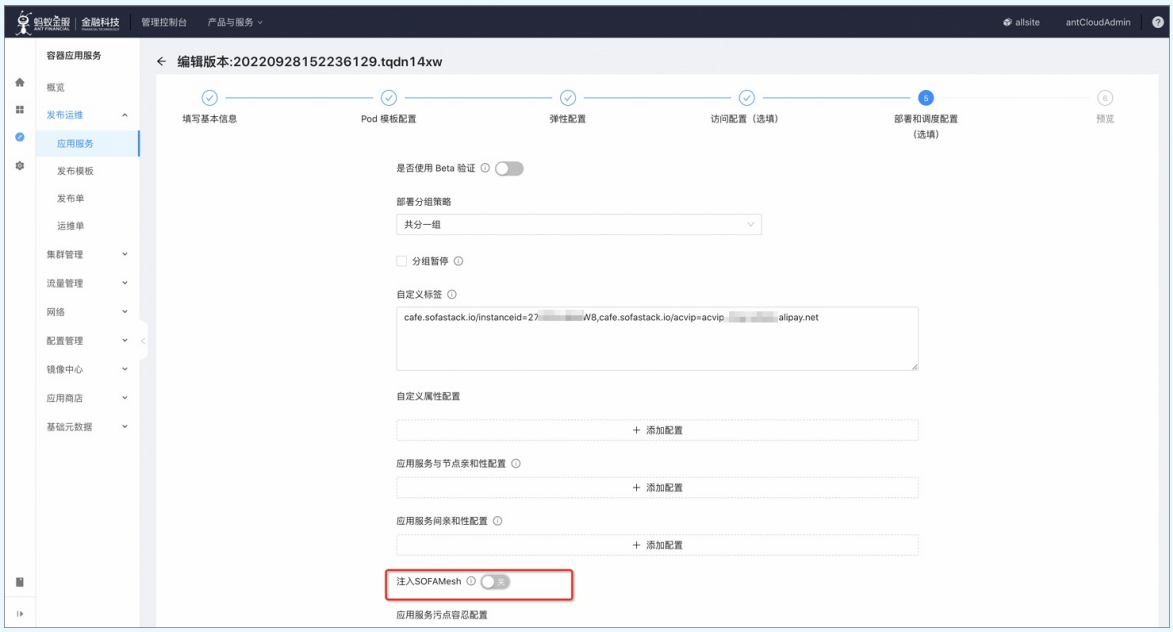
   应用服务发布完成后，可以在应用服务的详情页面查看应用服务是否成功注入 Sidecar。



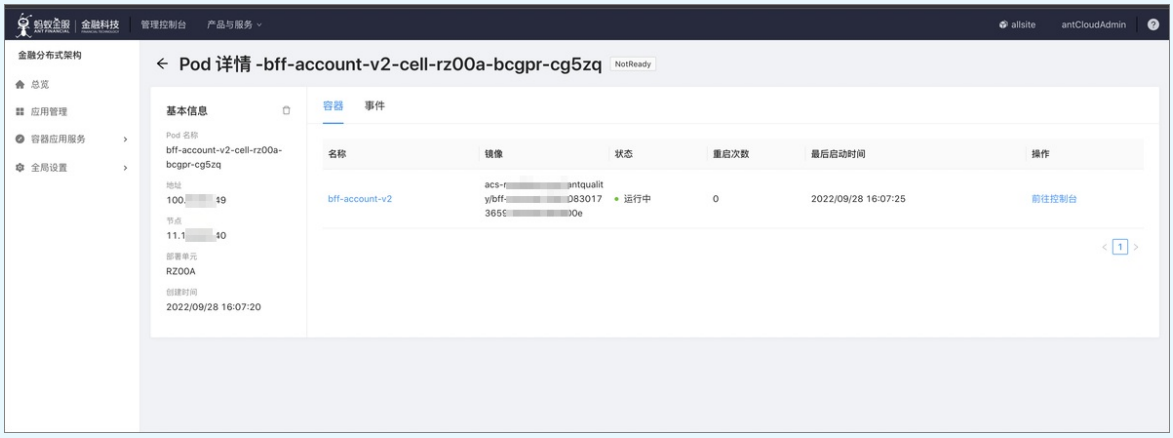

例如上图中，在容器的详情页出现了 mosn-sidecar-container 容器，表示 Sidecar 注入成功。

**说明**

如果您后续不再需要注入 Sidecar，您可以编辑应用服务，在 **部署和调度配置** 步骤关闭 **注入 SOFAMesh**，并重新发布应用即可。




容器详情页中只剩一个容器，则表示关闭成功。



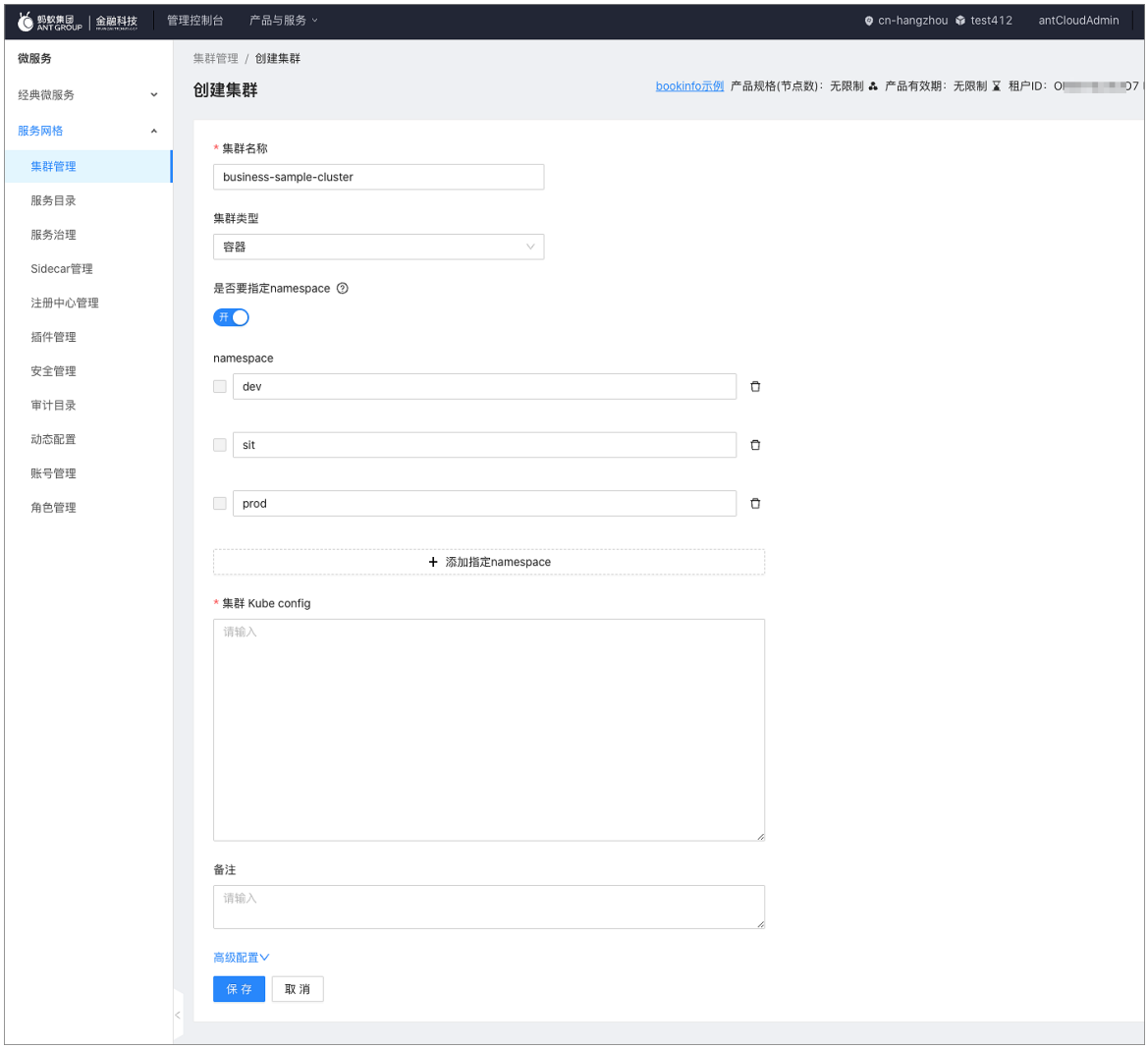

# 2.2.2. 通过 Mesh 控制台注入 Sidecar

## 步骤一：开通数据面集群

开通集群前，您需要准备好能够操作 K8s 集群资源的 kubeconfig ，一般取 master 节点的 `/root/.kube/config` 。

1. 登录微服务控制台。
2. 在左侧导航栏单击 **服务网格 > 集群管理**。
3. 单击 **创建集群**，并录入您的集群信息。

配置说明，请参见 创建集群。您可以打开 **是否要指定 namespace** 开关，以指定特定的 namespace，在命名空间维度控制 sidecar 注入的范围。




4. 为集群开启 Mesh 能力。

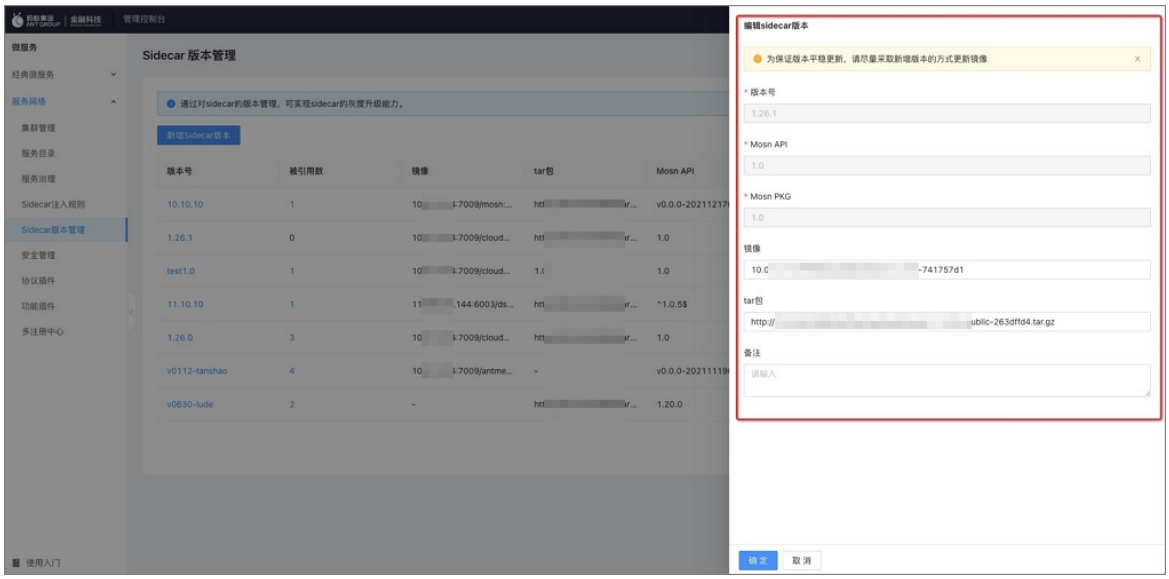
   操作方式，请参见 开通 Mesh。当 **开通 Mesh** 列的 **未开通** 按钮变为 **已开通**，表示集群已完成开通。

> **说明**
>
> 虚拟机集群创建后， Mesh 能力默认开启。如果您创建的是虚拟机集群，请跳过该步骤。

## 步骤二：创建 Sidecar 版本

1. 在左侧导航栏单击 **Sidecar 管理**，然后单击 **Sidecar 版本管理**。
2. 单击 **新增 Sidecar 版本**，并配置版本信息。

   配置说明，请参见 Sidecar 版本管理。

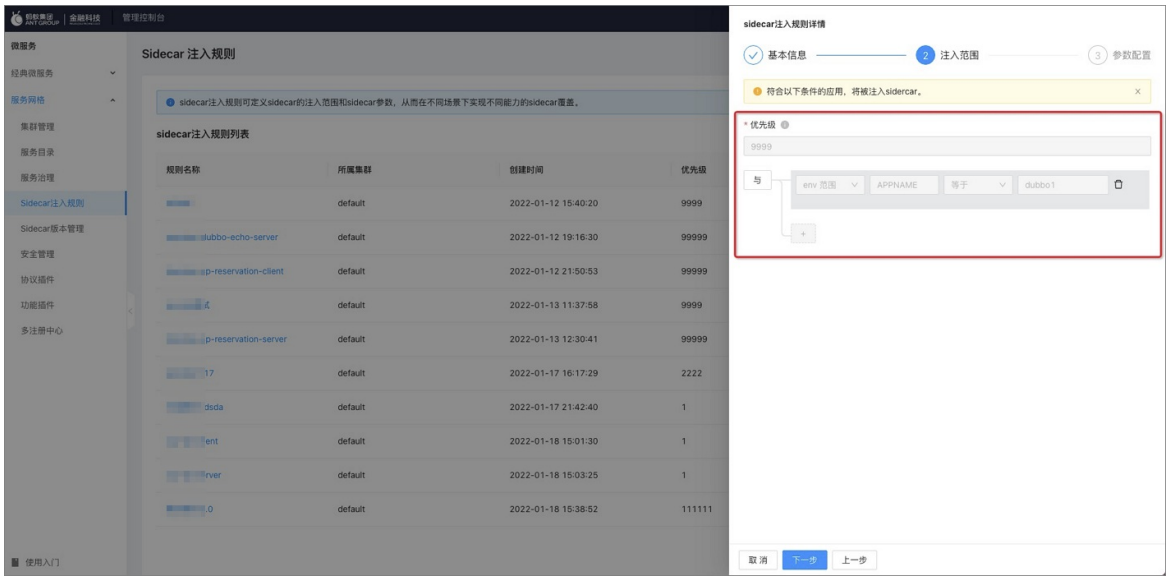



## 步骤三： 创建 Sidecar 注入规则

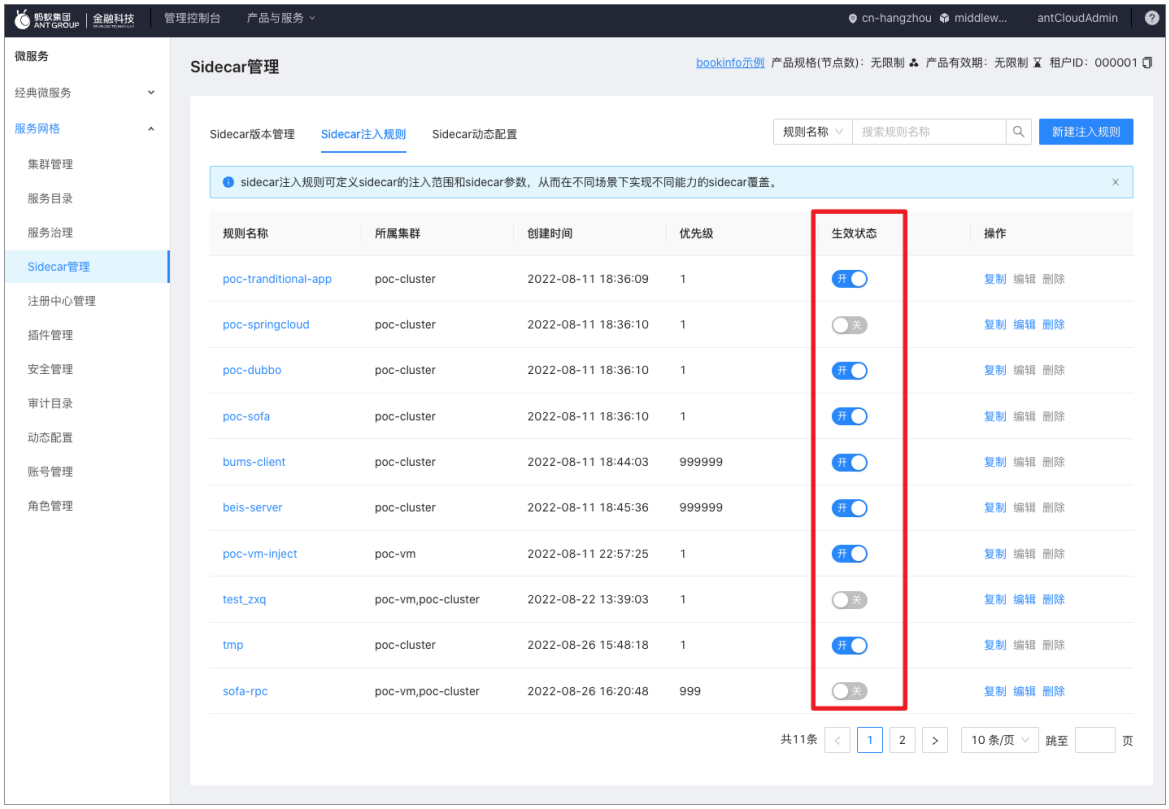
应用旁边的 Sidecar 需要通过注入规则的执行来完成安装。Sidecar 注入规则可定义 Sidecar 的注入范围和 Sidecar 参数（即哪些应用注入什么样的 Sidecar ），从而在不同场景下实现不同能力的 Sidecar 覆盖。

1. 单击 **Sidecar 注入规则**。
2. 单击 **新建注入规则**，并按照流程创建 Sidecar 注入规则。

配置说明，请参见 Sidecar 注入规则。




3. sidecar 注入模版创建完成后，将对应注入规则的 **生效状态** 置为 **开**。

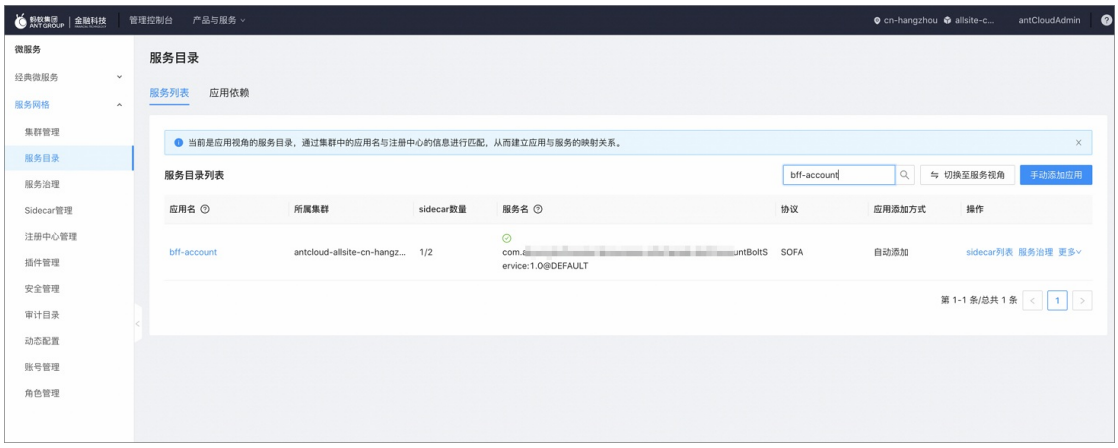



4. 注入 sidecar。

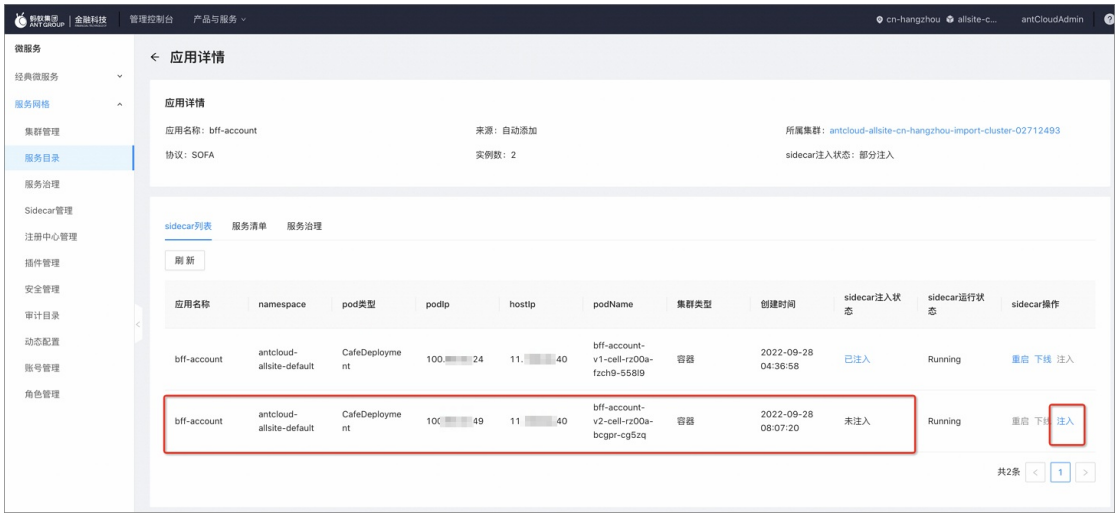
   i. 在左侧导航栏单击 **服务目录**。

ii. 找到需要注入 Sidecar 的应用，然后单击应用名称进入应用详情页。

您可以通过搜索查找目标应用。

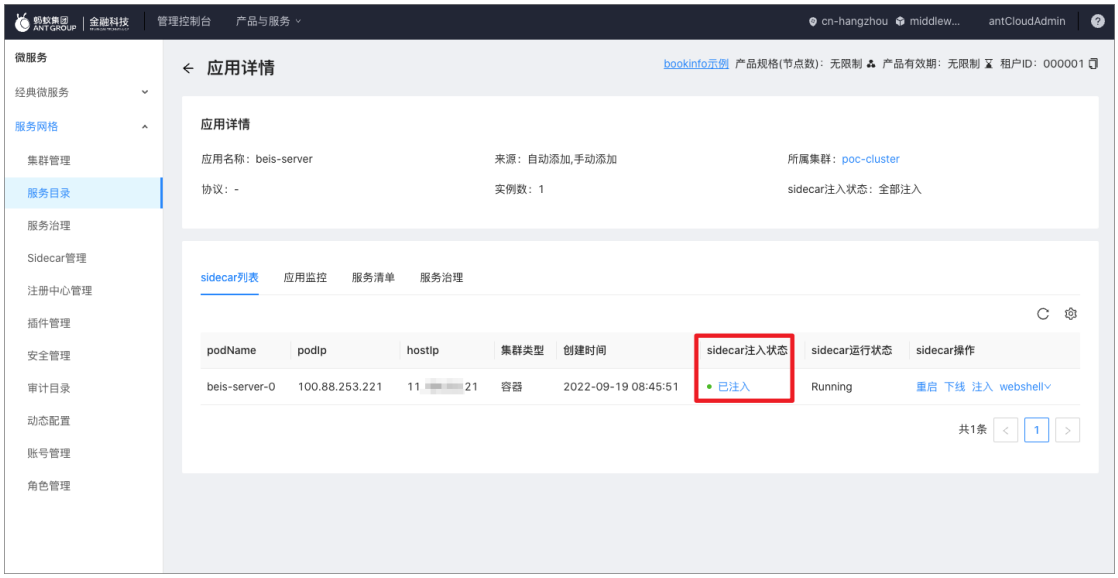



iii. 在应用详情页的 **Sidecar 列表** 页签，单击目标应用右侧的 **注入**。



iv. 选择目标规则后，单击 **确定**。

注入后，**Sidecar 注入状态** 显示为 **已注入**，则表示 Sidecar 注入成功。



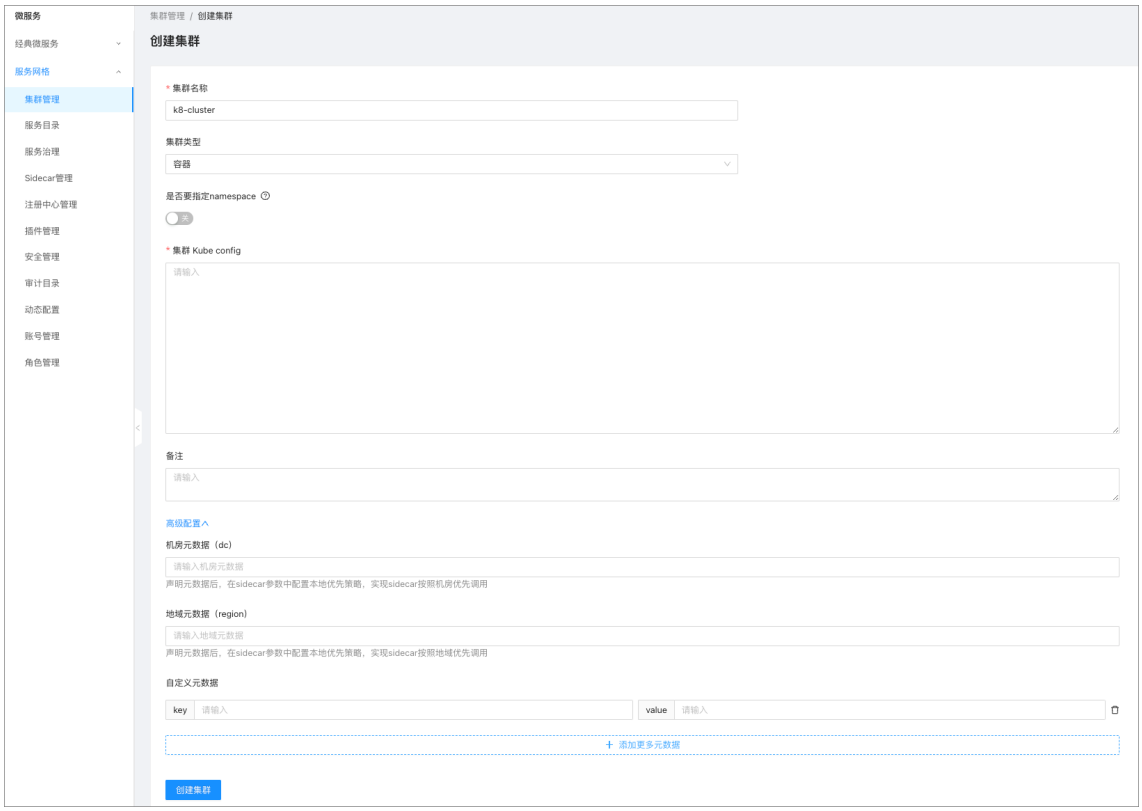
# 3.集群管理

## 3.1. 创建集群

集群是系统在运行过程中云资源（包括云主机等资源）的集合，您可以通过在集群管理页面创建应用集群，并添加集群的相关授权信息，实现服务网格对应用集群的纳管，使应用集群具备 Service Mesh 能力。

### 操作步骤

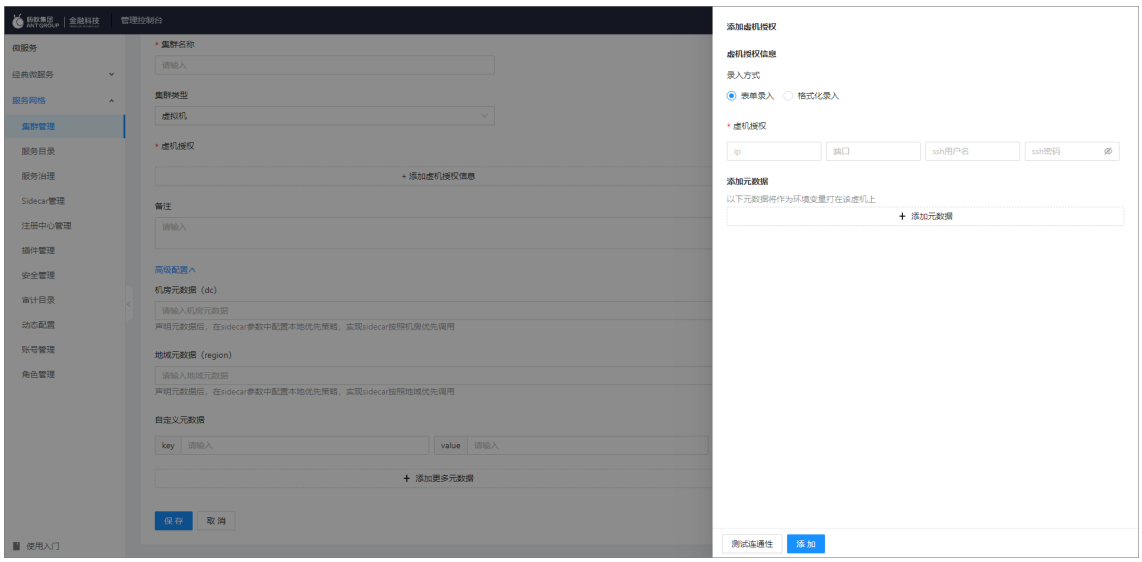
1. 登录微服务控制台。
2. 在左侧导航栏单击 **服务网格 > 集群管理**。
3. 在 **集群管理** 页面单击 **创建集群**，然后配置以下参数：
   - 创建容器集群




| 参数 | 说明 |
| --- | --- |
| **集群名称** | 填写集群名称。<br>可包含数字、字母、短划线（-）、英文句点（.），且必须以数字或字母开头和结尾。 |
| **集群类型** | 选择 **容器**。 |

| | |
|---|---|
| **是否要指定 namespace** | 选择是否指定 namespace。若指定，则只有该 namespace 开通 Mesh 功能；否则，集群下所有 namespace 均开通 Mesh 功能 。<br>您可以单击 **添加指定 namespace**，同时指定多个 namespace。若指定的 namespace 不存在，则自动创建对应的 namespace。 |
| **集群 Kube config** | 填写从 Kubernetes 集群配置文件中获取的 `kube_config.yaml` 文件内容。<br>系统会将 `kube_config.yaml` 配置文件的内容转成 JSON 格式存储。 |
| **备注**（可选） | 填写集群的说明信息。 |
| **高级配置** | 填写集群的高级选项，包括：<br>■ **机房元数据（dc）**：填写集群的数据中心 DC（datacenter）名称。<br>机房元数据是为了在多机房部署时，区分当前集群所在的机房，从而实现本机房优先等高级策略。<br>■ **地域元数据（region）**：填写集群的 region。<br>地域元数据是为了在多地部署，比如多两地三中心部署时区分当前集群所在的城市，从而实现本地域优先等高级策略。<br>■ **自定义元数据**：单击 **添加更多元数据**，并填写 key 和 value。<br>集群元数据将作为全局环境变量添加在该集群的所有 sidecar 上。您可以添加多条集群元数据。<br>自定义元数据会渲染到sidecar的环境变量中。 |

- 创建虚拟机集群



| 参数 | 说明 |
| --- | --- |
| **集群名称** | 填写集群名称。<br>可包含数字、字母、短划线（-）、英文句点（.），且必须以数字或字母开头和结尾。 |
| **集群类型** | 选择 **虚拟机**。 |
| **虚机授权** | 点击 **添加虚机授权信息**，然后填写虚拟机的信息，包括虚拟机 IP、端口、SSH 用户名和密码。<br>支持 **表单录入** 和 **格式化录入**：<br>■ **表单录入**：在表格中分别填写 IP、端口、用户名和密码信息。<br>■ **格式化录入**：通过 `IP:port username password` 格式录入虚拟机信息。例如：`10.0.0.1:80 admin admin`。<br>单击 **添加元数据**，可为虚拟机添加环境变量。 |
| **备注**（可选） | 填写集群的说明信息。 |
| **高级配置** | 填写集群的高级选项，包括：<br>■ **机房元数据（dc）**：填写集群的数据中心 DC（datacenter）名称。<br>机房元数据是为了在多机房部署时，区分当前集群所在的机房，从而实现本机房优先等高级策略。<br>■ **地域元数据（region）**：填写集群的 region。<br>地域元数据是为了在多地部署，比如多两地三中心部署时区分当前集群所在的城市，从而实现本地域优先等高级策略。<br>■ **自定义元数据**：单击 **添加更多元数据**，并填写 key 和 value。<br>集群元数据将作为全局环境变量添加在该集群的所有 sidecar 上。您可以添加多条集群元数据。<br>自定义元数据会渲染到sidecar的环境变量中。 |

4. 单击 **确定**。

5. 返回集群列表后，您可以看到所有已创建的集群。

**入口网关** 和 **开通 Mesh** 显示入口网关和 Mesh 的开通情况，单击状态可进行配置。如果该集群已经开启了 Mesh，则不允许直接删除，需要先关闭再删除。

# 3.2. 查看集群

集群创建后，您可以在集群列表中查看集群的相关信息。

## 操作步骤

1. 登录微服务控制台。
2. 在左侧导航栏单击 **服务网格 > 集群管理**。
3. 在 **集群管理** 页面查看集群信息。

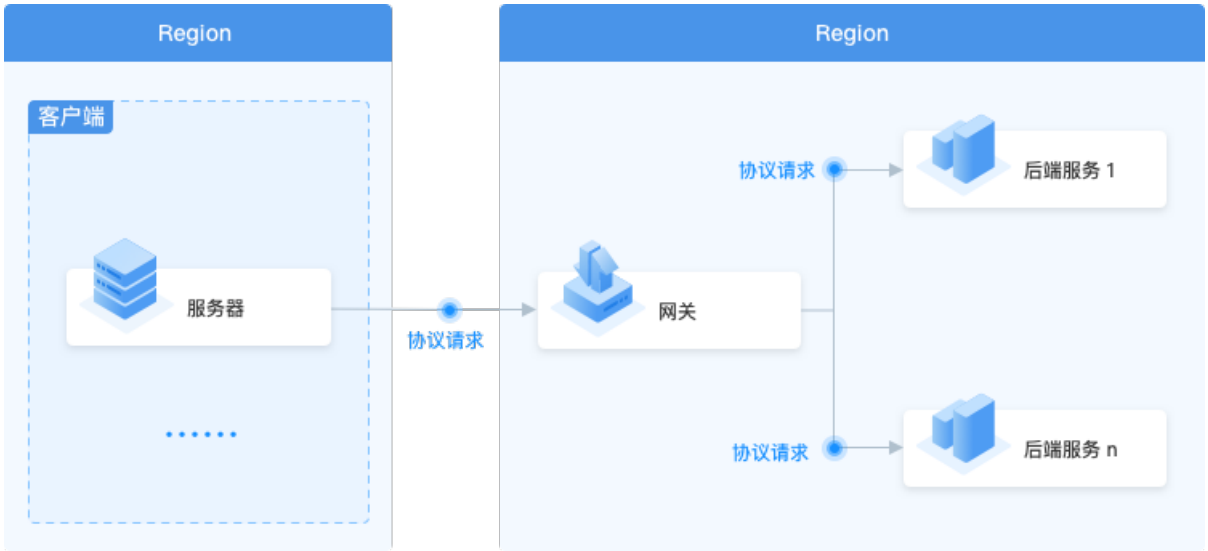
您可以查看集群名称、类型、地址、应用数量、Sidecar 数量等信息，其中：**入口网关** 和 **开通 Mesh** 显示入口网关和 Mesh 的开通情况，单击状态可进行配置。如果该集群已经开启了 Mesh，则不允许直接删除，需要先关闭再删除。

# 3.3. 开通入口网关

入口网关作为集群访问入口，实现跨网络打通或内部服务对外暴露等功能。

## 使用场景

目前主要用入口网关实现集群间网络打通，没有此需求请不要开启入口网关，以免产生额外的网络开销。 开通入口网关后，需将应用注册到网关。当服务方应用已注册到网关，且调用方和服务方不在同一个 region 时，调用方会通过网关调用服务。

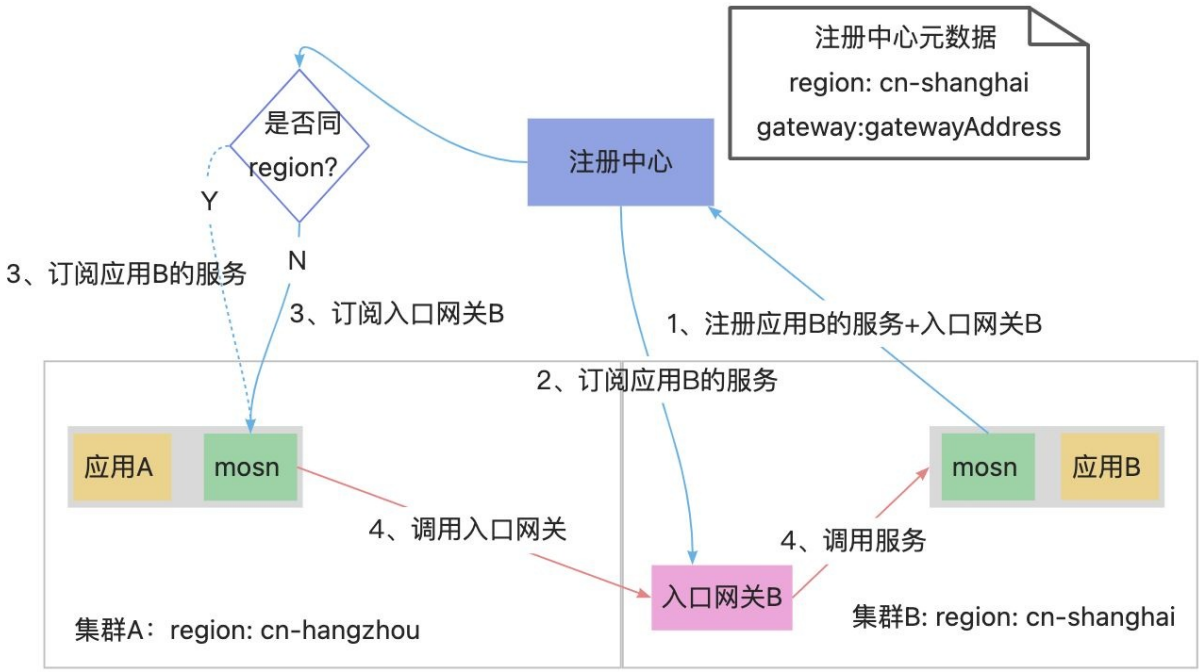



## 注意事项

- 当前只支持容器集群开通入口网关。
- 只有服务端、调用端不在同的 region 时，才能走网关调用。

  region 信息在开通集群，或编辑集群时指定。详情请参见 创建集群。

## 原理说明




## 前置条件

服务端所在集群开通了入口网关且应用 B 已注册到入口网关。

## 流程说明

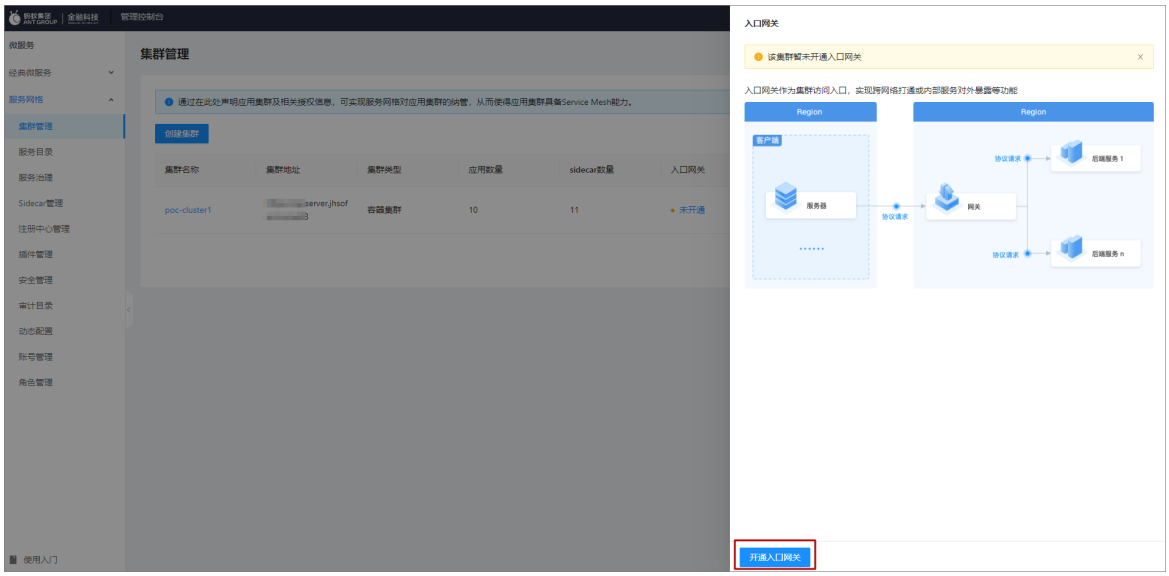
1. 服务端应用 B 选择通过网关对外暴露服务，注册服务时会携带入口网关的地址。
2. 入口网关 B 会订阅应用 B 的服务。

3. 调用方应用 A 订阅服务时，判断当前服务的 region 是否和本集群的 region 相同。如果不同，则订阅入口网关 B。
4. 应用 A 订阅入口网关 B 后，流量会通过入口网关 B 打到应用 B。

## 操作步骤

1. 登录微服务控制台。
2. 在左侧导航栏单击 **服务网格 > 集群管理**。
3. 在集群列表单击目标集群 **入口网关** 列的 **未开通**。

4. 在 **入口网关** 面板单击 **开通入口网关**。

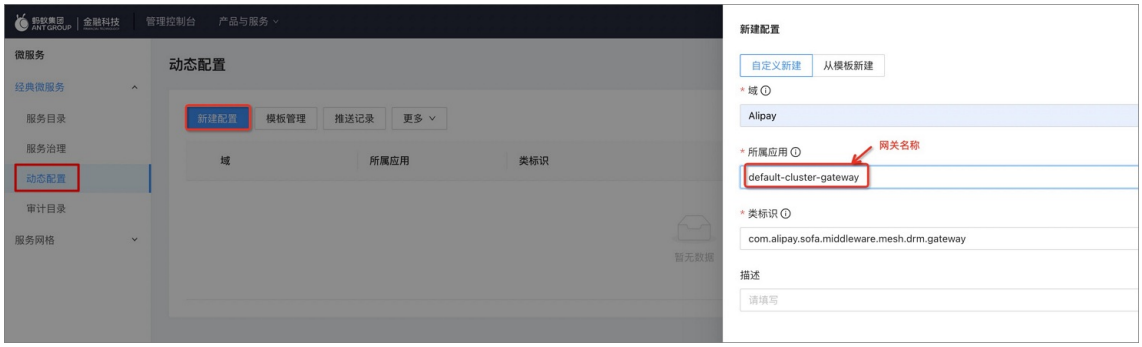



入口网关开通成功后，页面会显示网关名称。

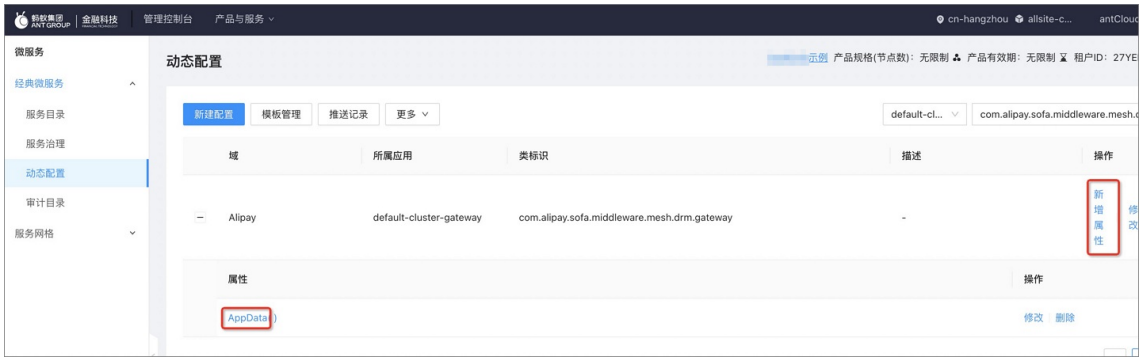
5. 注册应用到入口网关。

   在动态配置中，配置要注册到入口网关的应用列表。

   - 所属应用为：网关名称。
   - 类标识为：com.alipay.sofa.middleware.mesh.drm.gateway。




- 属性为：AppData。

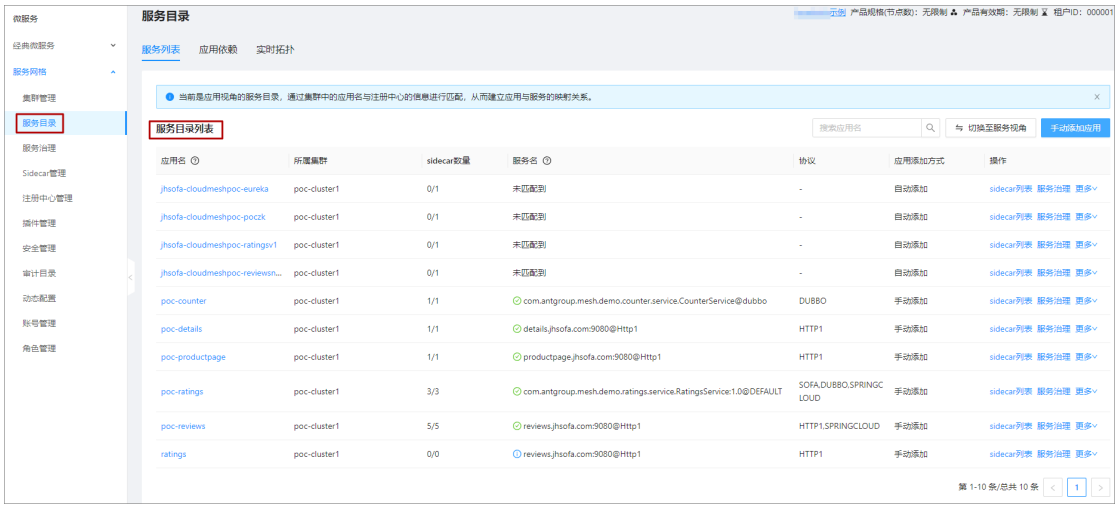



- 推送值为：应用列表，例如：["springcloud-server-train"]。

  当有多个应用要注册时，用英文逗号（,）号分隔，例如["appName1","appName2",...]。如果要删掉应用，则将对应的应用名从列表中删除，重新推送即可。

6. 在服务端（集群 B）注入或重新安装 Sidecar，然后发布服务。

i. 在服务目录中查找到服务端（集群 B）的应用。




ii. 单击目标应用右侧的 **sidecar 列表**，然后单击 **sidecar 操作** 列的 **注入**。

iii. 重新发布服务。

7. 以同样的方式在客户端（集群 A）注入或重新安装 Sidecar，然后订阅服务。

# 3.4. 开通 Mesh

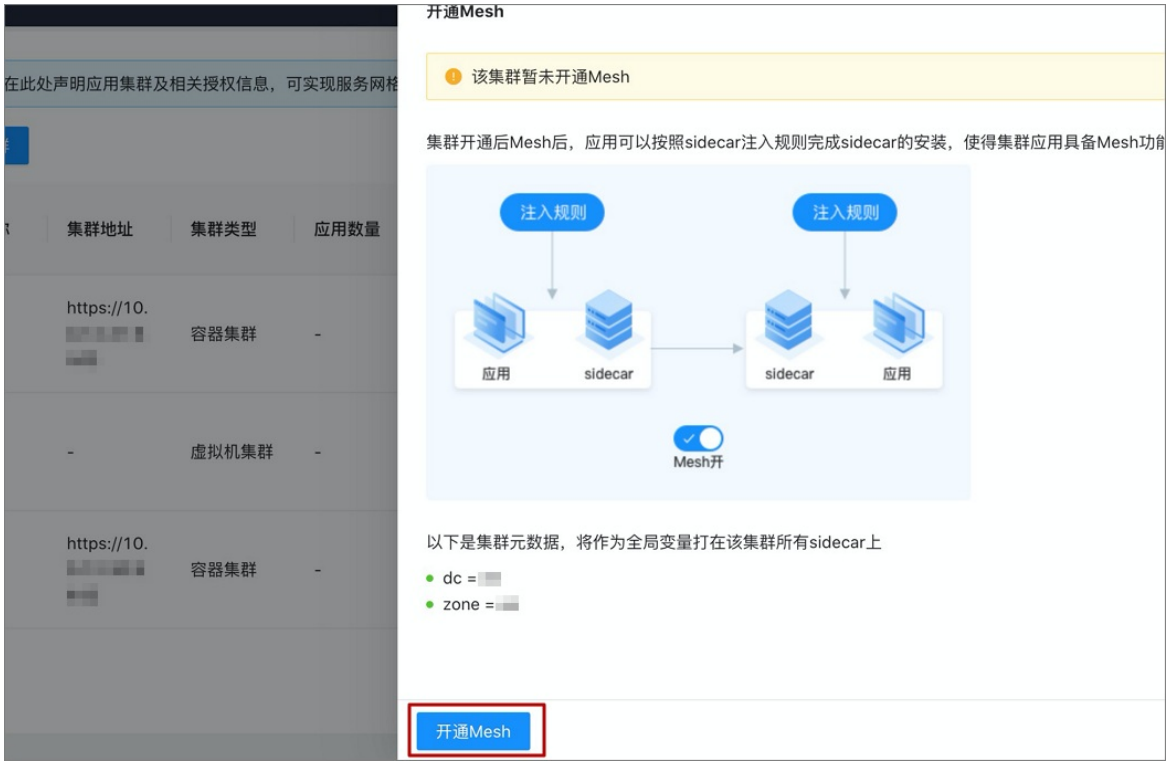
容器集群创建后，若应用需使用 Mesh 能力，您还需要为集群开通 Mesh 功能。

## 操作步骤

> **说明**
>
> 虚拟机集群创建后，Mesh 能力默认为开通状态。

1. 登录微服务控制台。
2. 在左侧导航栏单击 **服务网格 > 集群管理**。
3. 在 **集群管理** 页面，单击目标集群 **开通 Mesh** 列的 **未开通**。

4. 在 **开通 Mesh** 面板，单击 **开通 Mesh**。




5. 开通成功后，会显示集群的元数据信息。

# 3.5. 编辑或删除集群

您可以对集群信息进行修改，对于不需要的集群，您可以选择删除。

## 编辑集群

您可以根据需要，随时修改已创建的集群信息。

1. 在 **集群管理** 页面，单击目标集群右侧的 **编辑**。

2. 在 **编辑集群** 面板修改配置信息。

   您可以修改高级配置、虚拟机集群的账号和密码、添加虚拟机授权、容器集群的指定 namespace 等信息。

3. 修改完成后，单击 **确定**。

## 删除集群

删除集群的操作仅仅是对集群的元数据进行删除，不会删除您所创建的集群。

1. 在服务集群列表，单击目标集群右侧的 **删除**。

> **重要**
>
> 若目标集群已开通 Mesh，您需要先关闭 Mesh 功能。
>
> a. 单击目标集群 **开通 Mesh** 列的 **已开通**。

b. 在 **开通 Mesh** 面板 单击 **关闭 Mesh**。

2. 单击 **确定**。

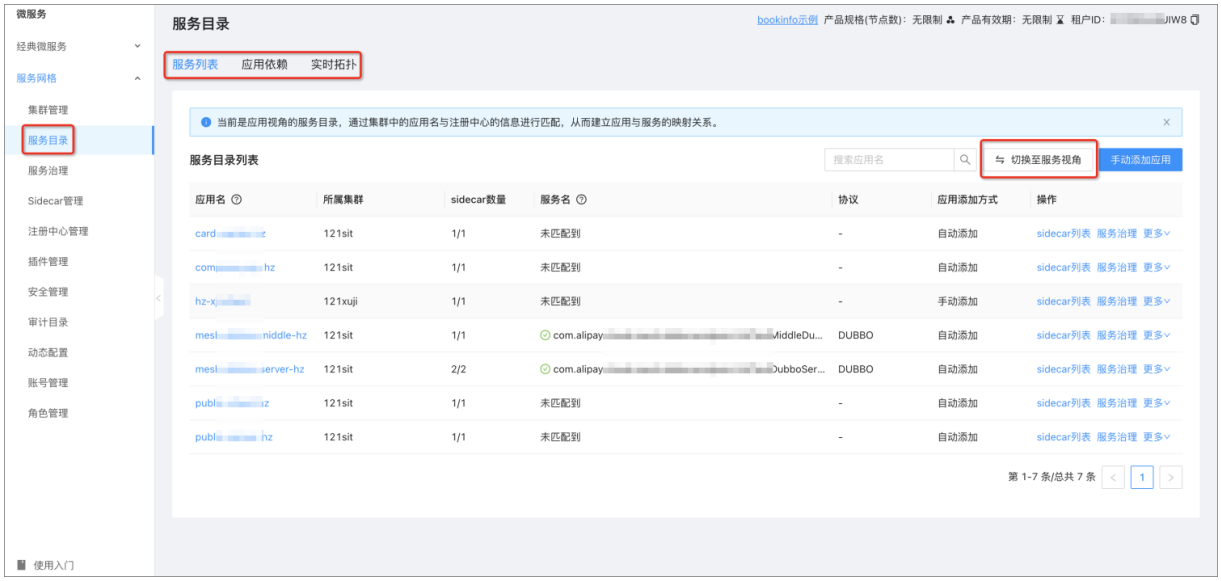
# 4.服务目录

## 4.1. 概述

服务目录是为了让用户可以直观的感受到当前系统中有哪些服务、应用，以及每个服务、应用的具体情况，支持服务视角或应用视角两种不同的角度进行查看。

通过服务目录，您可以：

- 查看当前系统中的服务列表和应用列表。
- 查看具体每个服务的注册和被订阅情况。
- 对应用发起 Sidecar 的注入操作。
- 查看具体每个应用的 Sidecar 注入情况、调用情况（如 QPS、成功率、是否触发了某个服务治理规则等），或对改应用配置服务治理规则。
- 查看各应用之间的依赖关系（订阅关系）。



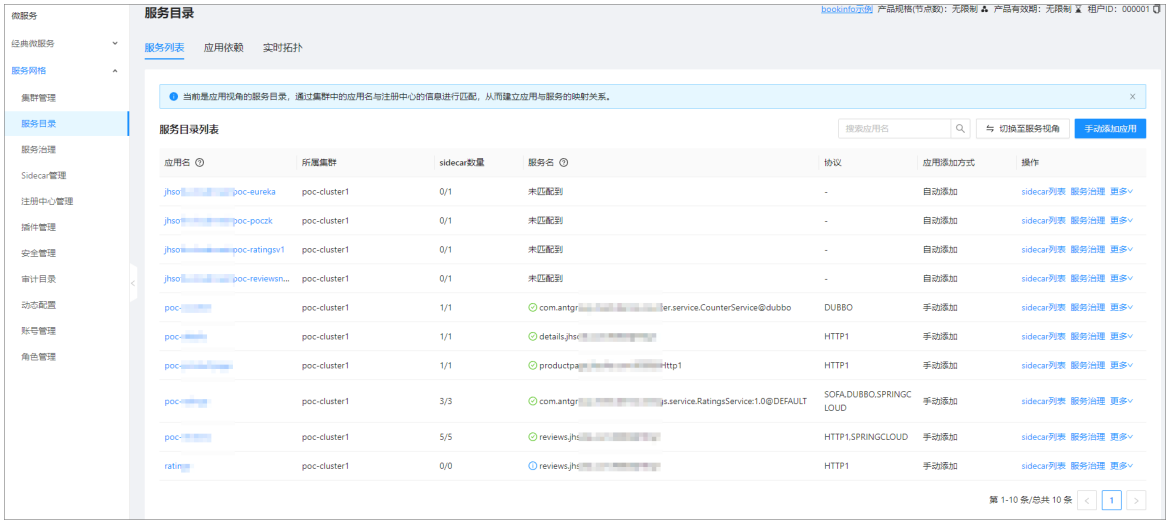

## 4.2. 查看服务目录

您可以通过服务目录功能查看所有可用的应用和服务。

### 应用视角

应用视角下，服务目录的每一行代表一个应用。

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 服务目录**。
3. 单击 **服务列表** 页签，然后单击 **切换至应用视角**。




> **说明**
>
> 如果当前已是应用视角，则无需单击 **切换至应用视角**。

您可以在服务目录列表中查看所有已发布的应用，包括应用名、所属集群、sidecar 数量、服务名等信息：

- 所属集群：该应用在 **集群管理** 菜单中所属的集群。
- sidecar 数量：展示 已注入 Sidecar 的 Pod（或实例）数量/该应用总的 Pod（或实例）数量。
- 服务名：该应用所发布服务的列表。前面的绿色对钩表明该服务已经注册到注册中心。
- 协议：当前应用的微服务协议类型，如 SPRINGCLOUD、DUBBO。
- 应用添加方式：展示该应用是通过从 K8s 集群中自动扫描出来的，还是通过右上角 **手动添加应用** 方式手动创建的。

> **说明**
>
> - 容器场景下，系统会去各个容器集群的 APIServer 请求所有的 deployments 和 statefulsets，并通过 IP 和 name 字段去跟注册中心匹配，如果匹配成功，则显示服务名。
> - 虚拟机场景没有容器类似的机制，您需要手动录入应用的元信息，包括应用名、注册的服务、订阅的服务、虚机清单等，录入成功的数据也会显示在服务目录中。手动录入应用操作请参见 手动添加应用。

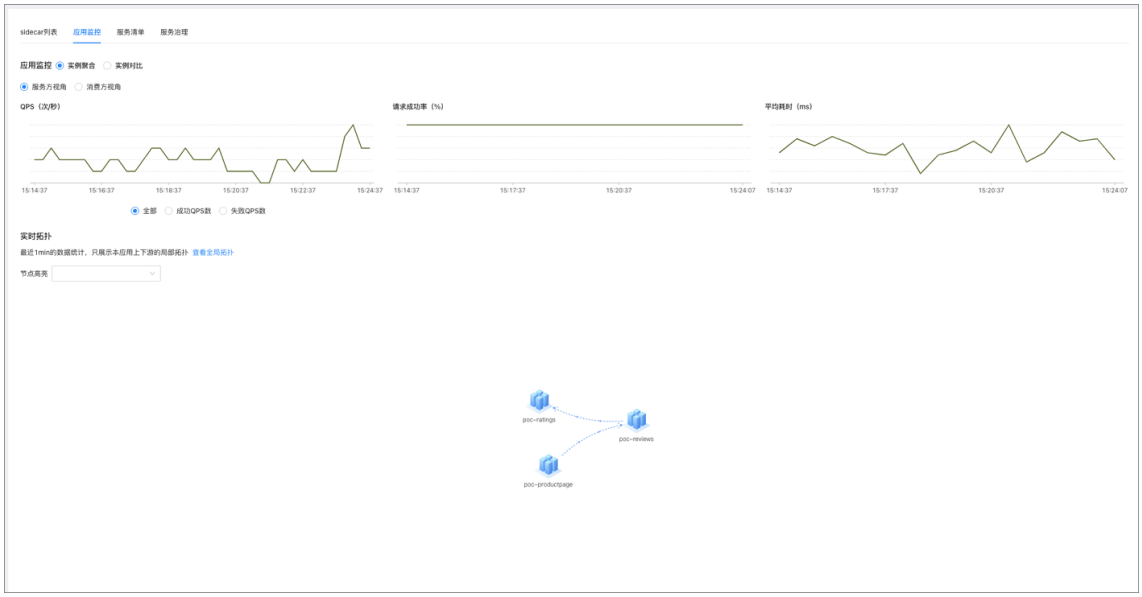
4. 单击目标应用名称查看应用详情。

应用详情页分为 2 个部分，上面会展示该应用的基本信息，下面的区域分为 **sidecar 列表**、**应用监控**、**服务清单** 和 **服务治理** 四个页签。

- **应用详情**：包括应用名称、来源、所属集群、协议、实例数、Sidecar 注入状态等信息。
- **sidecar 列表**：sidecar 列表会以列表的形式展示当前应用的实例列表的情况。
  - **podName**：容器场景展示 podName；虚拟机场景展示 IP。
  - **集群类型**：容器或虚拟机。
  - **sidecar 注入状态**：当前实例的 sidecar 注入状态（未注入、注入中、已注入或注入失败）。
    - 如果是已注入状态，单击会进入 Sidecar 详情页；悬停会展示当前应用命中的是哪个 Sidecar 注入规则，单击规则名称则会进入规则详情页。
    - 如果是注入中状态，单击时会滚动刷新 Sidecar 的注入进程。
  - **sidecar 运行状态**：当前实例的 sidecar 运行状态（Runing或Error）。
  - **sidecar 操作**：可发起 Sidecar 的重启、下线、注入操作。单击 **webshell** 可进入 webshell 页，进入时可选择进入应用容器还是 MOSN 容器。

> **重要**
>
> 当前只支持对整个应用进行 Sidecar 的注入、下线或重启，不允许指定具体哪个实例。

- **应用监控**：应用监控会展示该应用当前时间的监控数据，包含流量情况和实时拓扑。

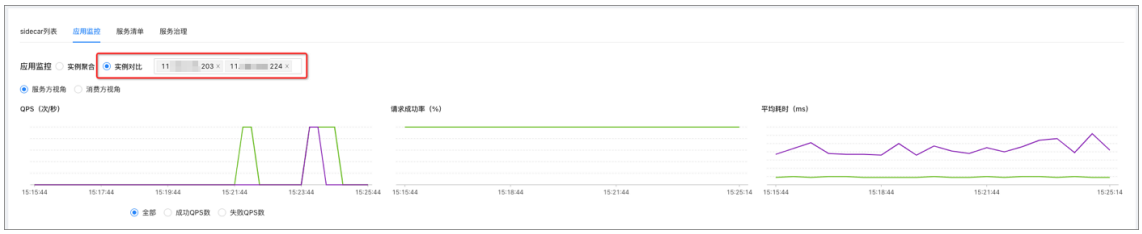



流量情况包含 QPS、请求成功率和平均耗时三个指标，支持服务方视角和消费方视角：

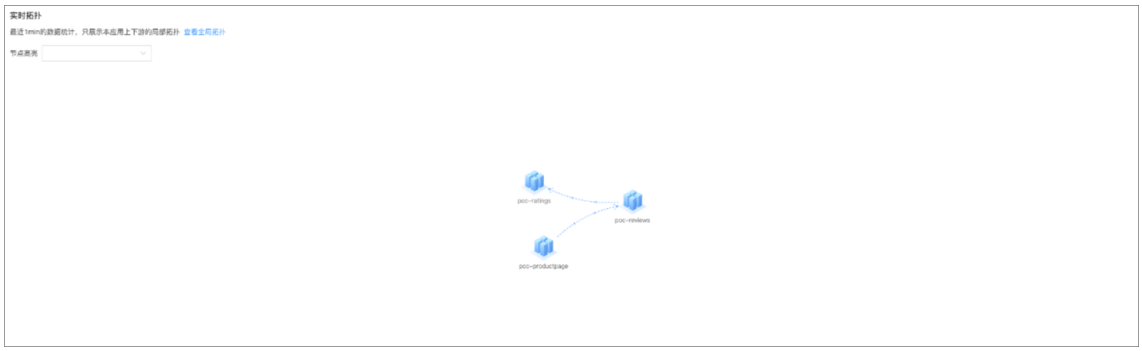
- 服务方视角：应用作为服务方场景的监控情况。
- 消费方视角：应用作为消费方发起请求调用的监控情况。

流量情况支持实例聚合模式和实例对比模式：

- 聚合模式是指以应用作为整体统计。
- 实例对比模式是指具体到某个应用实例观察流量，可以多个应用实例进行对比。

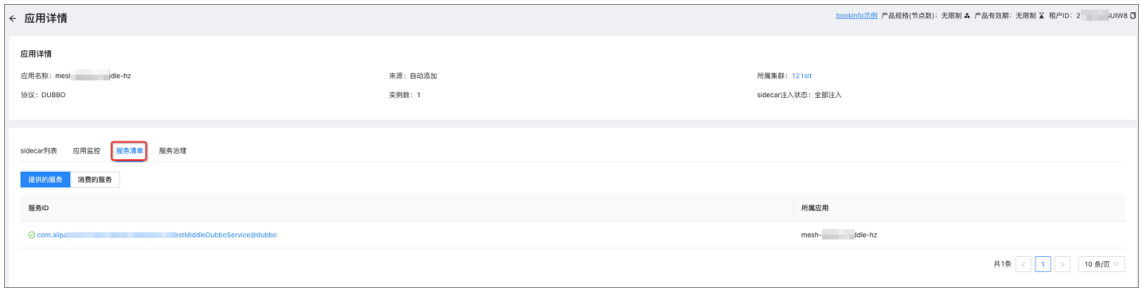



实时拓扑可以展示最近 1 分钟产生过调用关系，以本应用为中心上下游一跳的局部拓扑。

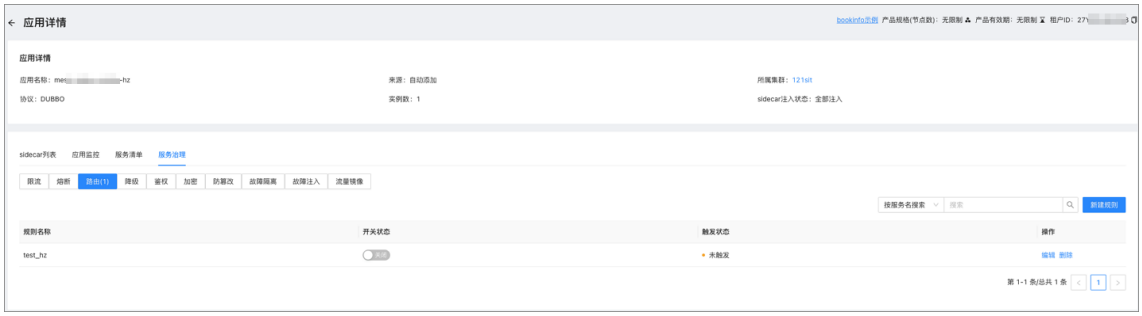



点击具体某个应用 > 查看详情，可以看到该应用服务治理规则的配置和触发情况。详情请参见 查看实时拓扑。

- **服务清单**：展示当前应用所注册的服务和订阅服务的列表。



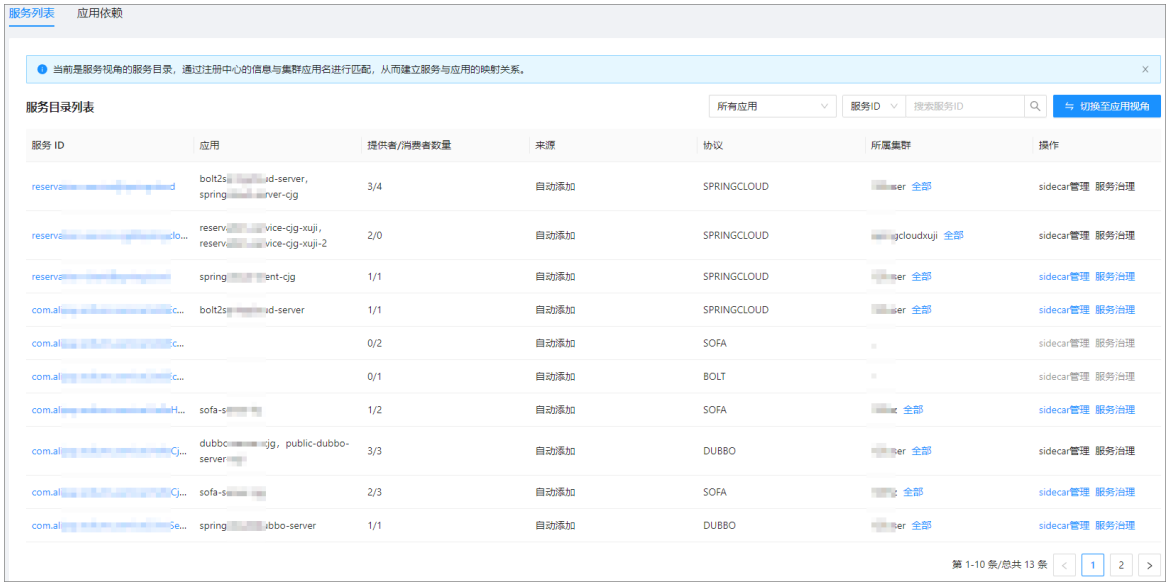

- **服务治理**：服务治理汇总展示本应由所有已配置的服务治理规则，可以在这里直接对服务治理规则进行查看或编辑，便于从应用的视角统筹应用的治理情况。详情请参见手册中服务治理的内容。




## 服务视角

服务视角下，服务目录的每一行代表一个服务，一个服务可能对应多个应用。

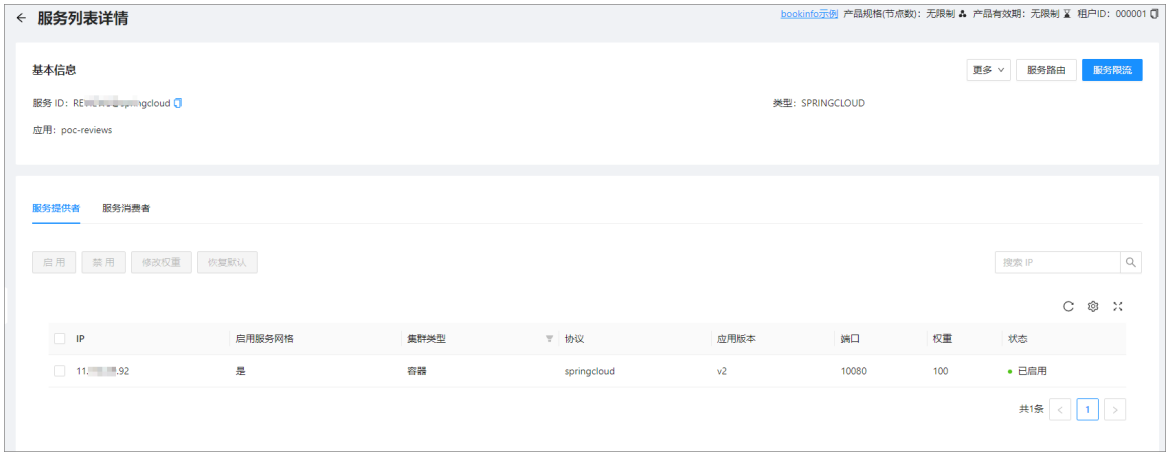
1. 在 **服务列表** 页签单击 **切换至服务视角**。




您可以在服务目录列表中查看所有已发布的服务，包括服务 ID、提供服务的应用、服务提供者数量、服务消费者数量等。

2. 单击目标服务 ID 可查看服务详情。

   您可以查看服务的基本信息、服务提供者和服务消费者列表：




- **基本信息**：包含服务 ID、类型、应用等信息。
- **服务提供者**：查看服务提供者的 IP、端口、应用名、权重、禁用和启用状态信息。您还可以对目标服务进行启用、禁用、修改权重、恢复默认等操作。
- **服务消费者**：查看服务消费者的 IP 与应用名。

# 4.3. 管理服务目录

您可以通过服务目录功能管理所有可用的应用和服务。

## 管理应用

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 服务目录**。
3. 单击 **服务列表** 页签，然后单击 **切换至应用视角**。

> **说明**
>
> 如果当前已是应用视角，则无需单击 **切换至应用视角**。

您可以管理应用的 Sidecar 列表、对应用进行服务治理：

- Sidecar 列表

  单击目标应用右侧的 **Sidecar 列表**，您可以查看该应用的所有实例和 Sidecar 注入状态。对于已注入 Sidecar 的实例，您可以进行重启、下线操作。对于未注入 Sidecar 的实例，您可以单击 **注入**，手动注入 Sidrcar。

  - 重启

    a. 单击目标实例右侧的 **重启**。

    b. 在 **请选择重启方式** 对话框选择重启的方式：

    - **重新安装 sidecar**：以重新安装 Sidecar 的方式重启该实例。
    - **重启 sidecar**：直接重启 Sidecar。

    c. 单击 **确定**。

  - 下线

    单击目标实例右侧的 **下线**，系统将下线该实例，您可以单击 **下线中**，查看下线的过程。

- 注入

  进行注入操作前，您需要先创建 Sidecar 注入规则。详情请参见 Sidecar 注入规则。

  a. 单击目标实例右侧的 **注入**。

  b. 在 **sidecar 注入规则** 对话框选择注入规则后，单击 **确定**。

     您可以单击 **注入中**，查看 Sidecar 注入过程。

- 服务治理

  单击目标应用右侧的 **服务治理**，您可以查看应用涉及的服务路由、服务熔断、服务限流等规则。您还可以开启、关闭、编辑、删除指定规则。关于服务治理的更多信息，请参见 服务治理。

## 管理服务

在 **服务列表** 页签单击 **切换至服务视角**，您可以管理应用的 Sidecar 列表、对应用进行服务治理，操作与管理应用一致。

# 4.4. 手动添加应用

对于一些天生不具备注册发现机制的应用，您可以通过手动添加应用，将他们纳入服务目录，实现精准管控。

## 背景信息

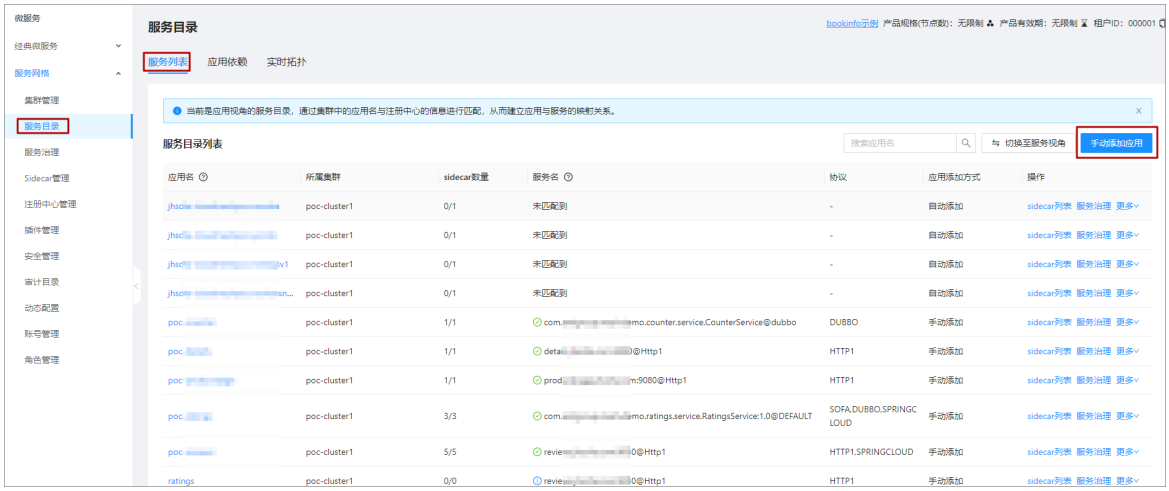
容器场景下，系统会去各个容器集群的 APIServer 请求所有的 deployments 和 statefulsets，并通过 IP 和 name 字段去跟注册中心匹配。如果匹配成功，则会显示在服务目录中。对于某些天生不具备注册发现机制的传统单体应用，您需要手动添加应用，将他们纳入服务目录。

虚拟机场景没有容器类似的机制，您需要手动录入应用的元信息，包括应用名、注册的服务、订阅的服务、虚机清单等。录入成功的数据也会显示在服务目录中。

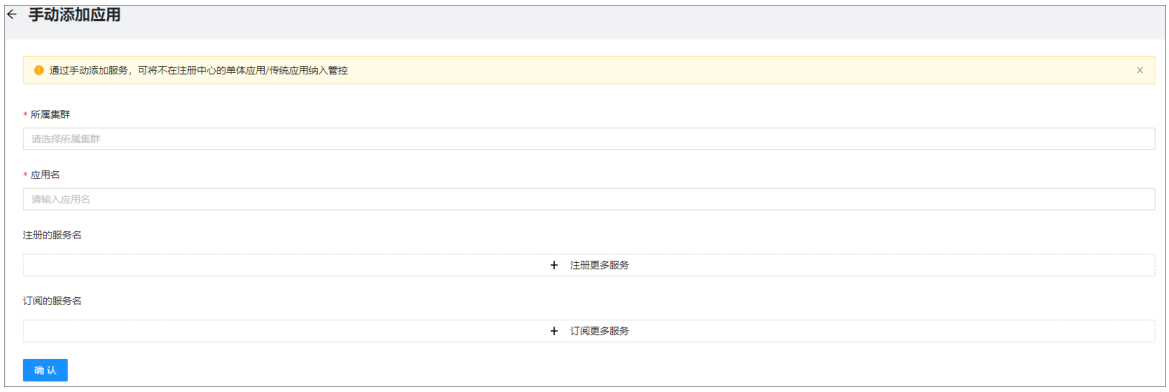
## 操作步骤

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 服务目录**。
3. 单击 **服务列表** 页签，然后单击 **切换至应用视角**。

> **说明**
>
> 如果当前已是应用视角，则无需单击 **切换至应用视角**。




4. 单击 **手动添加应用**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **所属集群** | 选择应用所属的集群。 |
| **应用名** | 输入应用名称。 |
| **注册的服务名** | 单击 **注册更多服务**，然后添加服务的 dataId、协议类型、端口号信息。<br>说明<br>目前支持 SOFA、Dubbo、SpringCloud 三种协议类型。<br>单击 **添加元数据** 还可以为服务添加元数据信息。<br>您可以同时添加多个服务和多个元数据。 |

| | |
|---|---|
| **订阅的服务名** | 单击 **订阅更多服务**，然后添加服务的 dataId、协议类型、端口号信息。<br>说明<br>目前支持 SOFA、Dubbo、SpringCloud 三种协议类型。<br>单击 **添加元数据** 还可以为服务添加元数据信息。您可以同时添加多个服务和多个元数据。 |
| **应用所在虚机** | 选择应用所属的虚拟机。<br>仅当所属集群为虚拟机集群时配置。 |
| **工作负载名称** | 选择应用的工作负载。<br>您可以输入工作负载名称以搜索工作负载（Deployment 和 StatefulSet），支持模糊匹配。<br>仅当所属集群为容器集群时配置。 |

5. 单击 **确认**。

# 4.5. 查看应用依赖

通过应用依赖可以从全局的角度看到各个应用之间的订阅依赖关系。

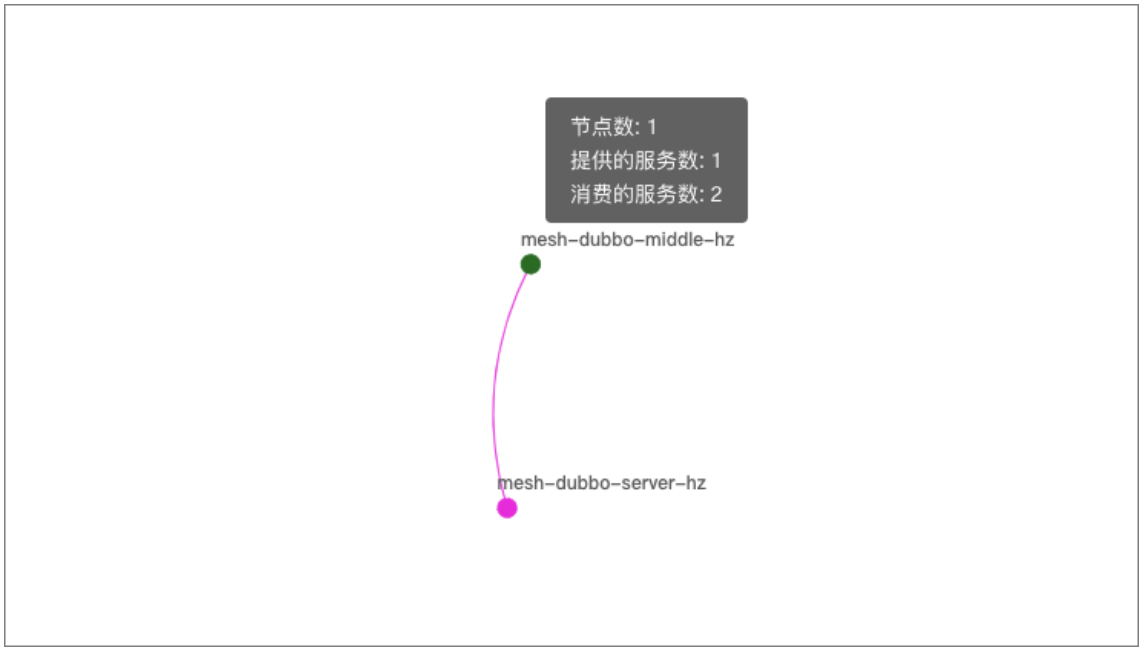
## 操作步骤

1. 登录微服务控制台。
2. 在左侧导航栏选择 **服务网格** > **服务目录**，然后单击 **应用依赖** 页签。

   在 **应用依赖** 页面，您可以执行以下操作：

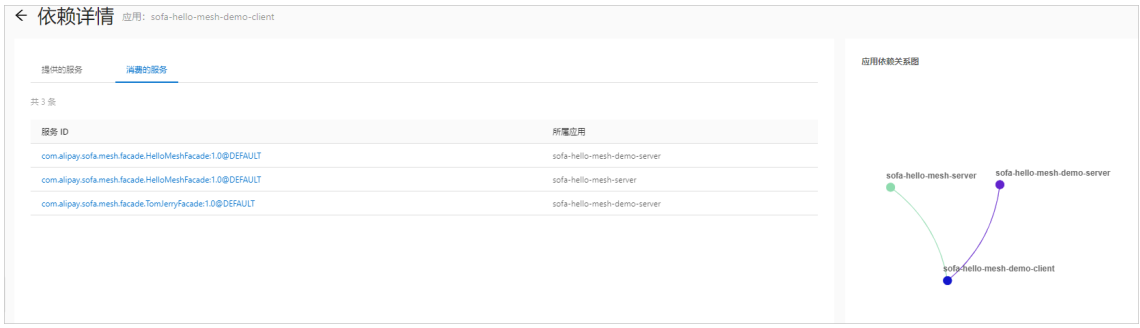
   - 查看有依赖关系的所有应用，包括服务提供者和消费者。

     如果连线颜色和应用颜色一致，表明该应用是作为服务提供者，否则是服务消费者。
   - 将鼠标悬浮在应用图标上，可以查看所选应用的依赖关系、节点数、提供的服务数、消费的服务数。




- 单击应用图标可查看提供的服务和消费的服务详情。

  您可以在服务列表单击目标服务 ID，查看服务的详细信息。




  例如上图展示了消费的服务信息：

  - 应用名：`sofa-hello-mesh-demo-client`
  - 依赖关系图：消费的服务共 3 个，来自于 2 个应用。
    - 2 个服务来自于 `sofa-hello-mesh-demo-server` 。
    - 1 个服务来自于 `sofa-hello-mesh-server` 。

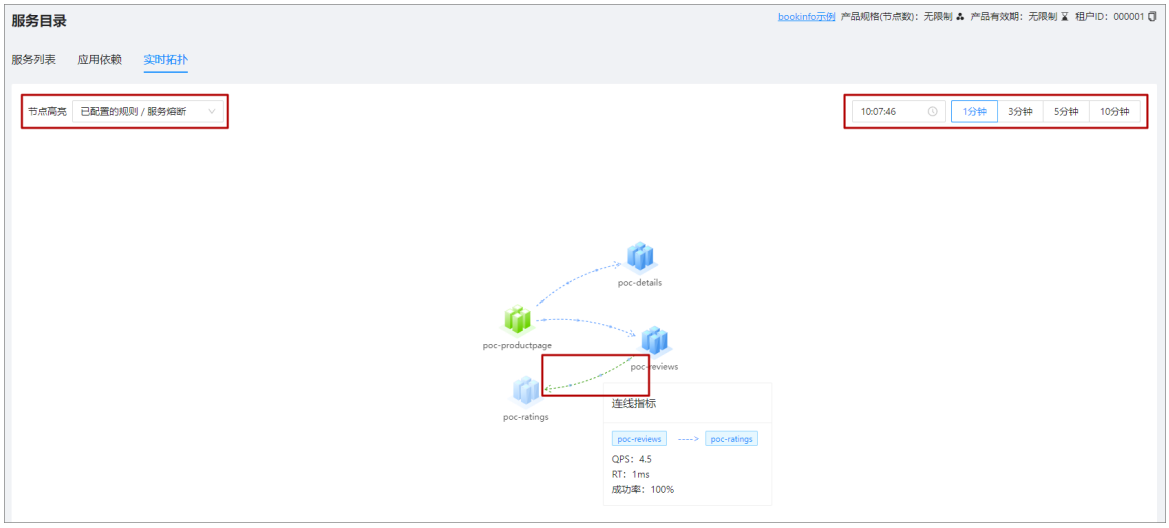
# 4.6. 查看实时拓扑

实时拓扑可以展示某个时间点（7 天以内）之前 1、3、5、10 分钟内，产生过调用关系的应用调用拓扑。

## 操作步骤

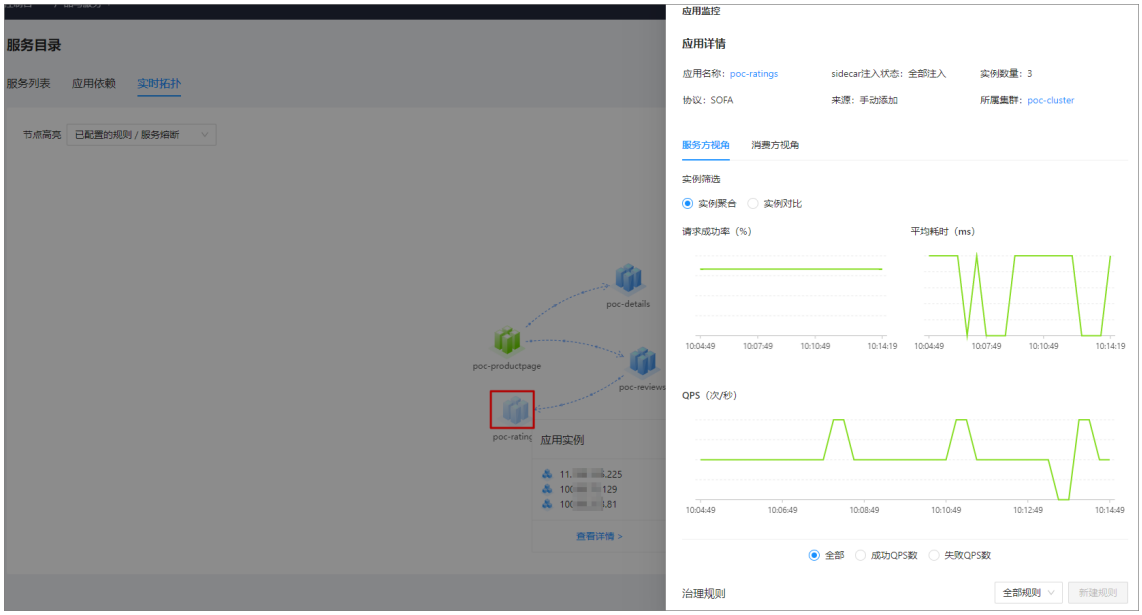
1. 登录微服务控制台。
2. 在左侧导航栏选择 **服务网格** > **服务目录**，然后单击 **实时拓扑**。
3. 在右侧设置想查看的时间。

您可以设置 7 天内的某个时间点，然后选择 1、3、5、10，即可查看某个时间点对应分钟内的调用关系。




您也可以设置节点高亮、查看连线指标或查看节点详情：

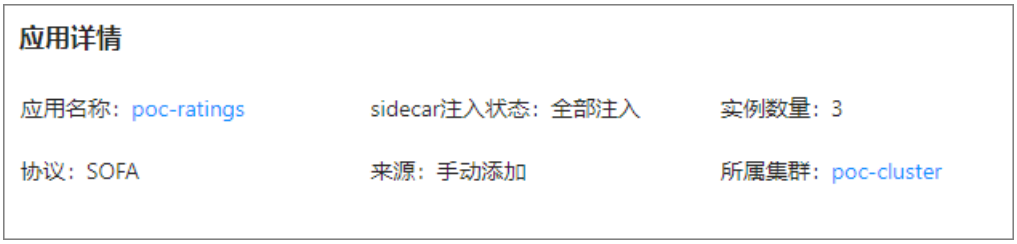
- 节点高亮：支持按照配置或触发了某种规则作为条件进行节点高亮，比如选中 已配置的规则-服务限流，则当前配置了服务限流的应用会被高亮。
- 连线指标：连线方向是从服务调用方指向服务提供方，单击连线时会展示该调用链路的基本流量指标（QPS、平均耗时和成功率）。
- 节点详情：单击应用会显示该应用的具体实例 IP，点击查看详情会展开该应用实时的监控情况。更多信息请参见下方应用监控详情说明。



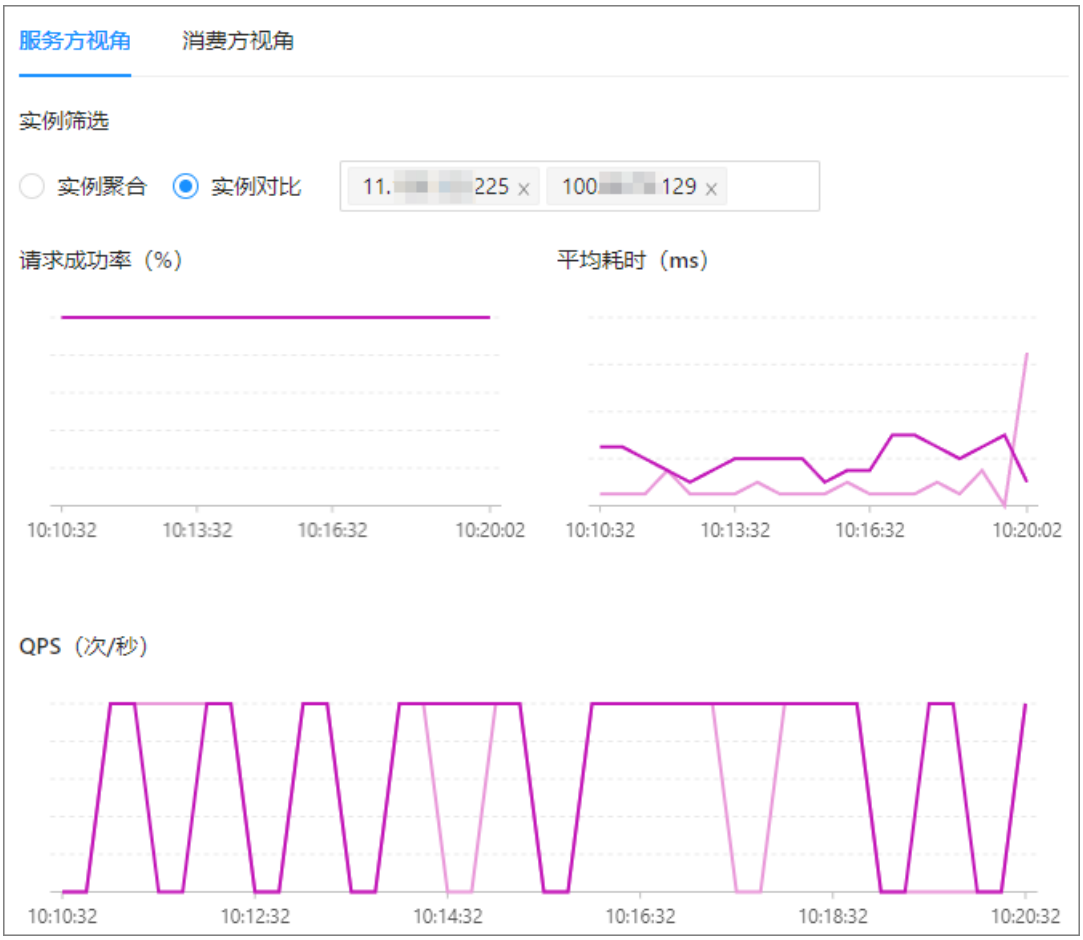

## 应用监控详情

应用监控详情从上到下包含 3 个展示区域：

- 最上面应用详情部分展示该应用的基本信息。



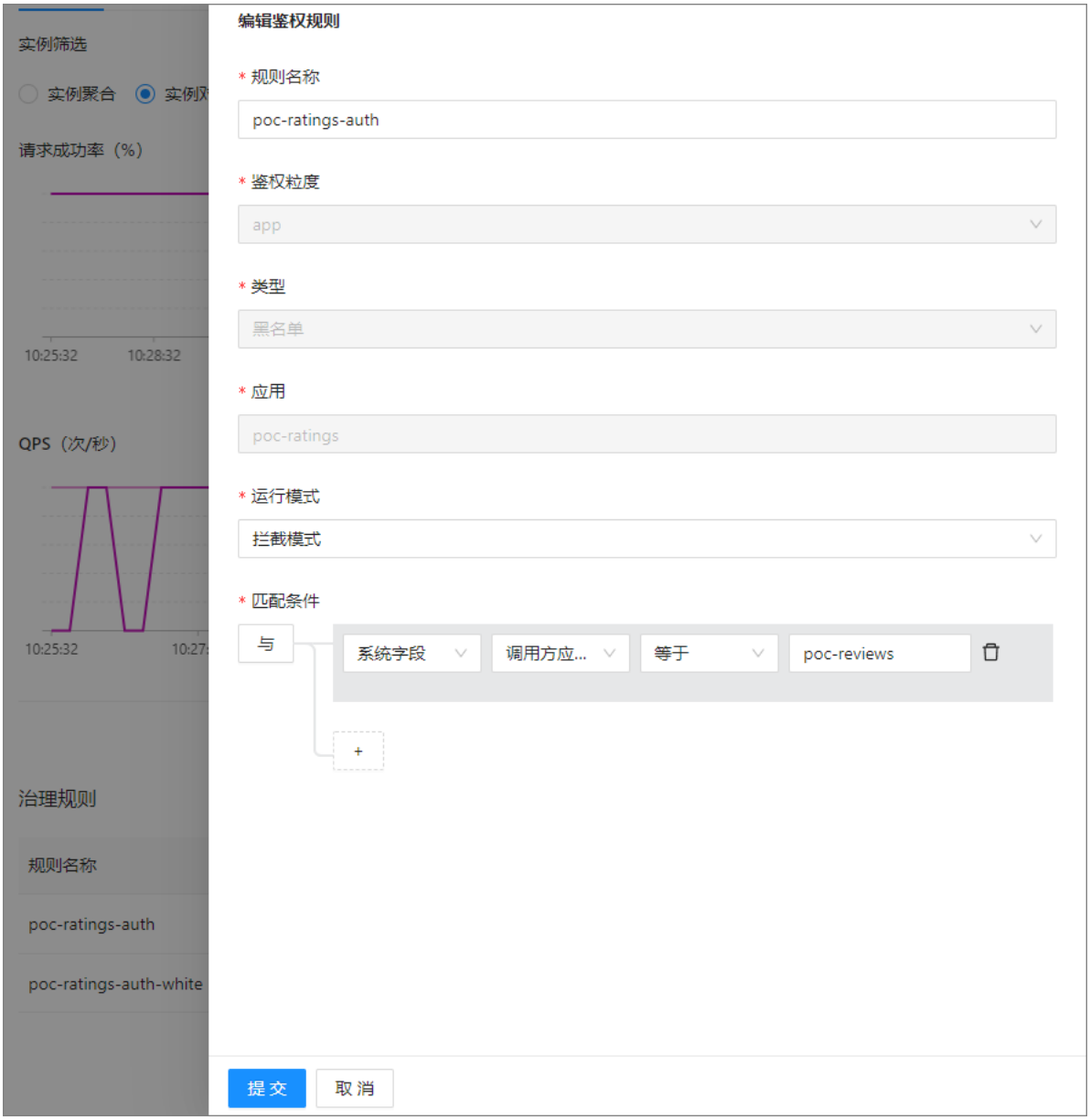

- 中间部分展示该应用的基本流量指标，每 10 秒钟自动刷新一次。支持服务方和消费方两个视角，支持实例聚合模式和实例对比模式两种。




  - 聚合模式：是指以应用作为整体统计。
  - 实例对比模式：是指具体到某个应用实例观察流量，可以选择多个应用实例进行对比。

- 最下面治理规则部分汇总展示本应所有已配置的服务治理规则，您可以在这里直接对服务治理规则进行查看或编辑，便于从应用的视角统筹应用的治理情况。单击 **编辑** 可以直接发起对该治理规则的修改。

治理规则
全部规则
新建规则
规则名称
开关状态
触发状态
操作
poc-ratings-auth
关闭
已触发（2022-09-01 15:10:44）
编辑 删除
poc-ratings-auth-white
关闭
已触发（2022-09-03 09:11:01）
编辑 删除
poc-ratings-falt-inject
关闭
已触发（2022-09-01 14:32:21）
编辑 删除
poc-ratings-mtls
关闭
未触发
编辑 删除
第 1-4 条/总共 4 条
1


实例筛选
实例聚合
请求成功率（%）
10:25:32
10:28:32
QPS（次/秒）
10:25:32
治理规则
规则名称
poc-ratings-auth
poc-ratings-auth-white
编辑鉴权规则
* 规则名称
poc-ratings-auth
* 鉴权粒度
app
* 类型
黑名单
* 应用
poc-ratings
* 运行模式
拦截模式
* 匹配条件
与
系统字段
调用方应...
等于
poc-reviews
+
提 交
取 消

**重要**

规则触发时间是本应用最后一次触发该规则的时间，跟当前规则是否处于开启状态无关。

# 5.Sidecar 管理

Sidecar 管理提供 Sidecar 的版本管理、注入规则管理和动态配置管理能力，目的是让您可以灵活的控制应用所注入的 Sidecar 的版本。

Sidecar 管理的一般使用方式为：

1. 创建 Sidecar 版本，一个版本代表一个特定的 Sidecar 镜像或可执行文件。
2. 定义 Sidecar 注入规则，即应用满足什么条件时会注入对应版本的 Sidecar，以及注入时要对 Sidecar 做怎样的初始化配置。

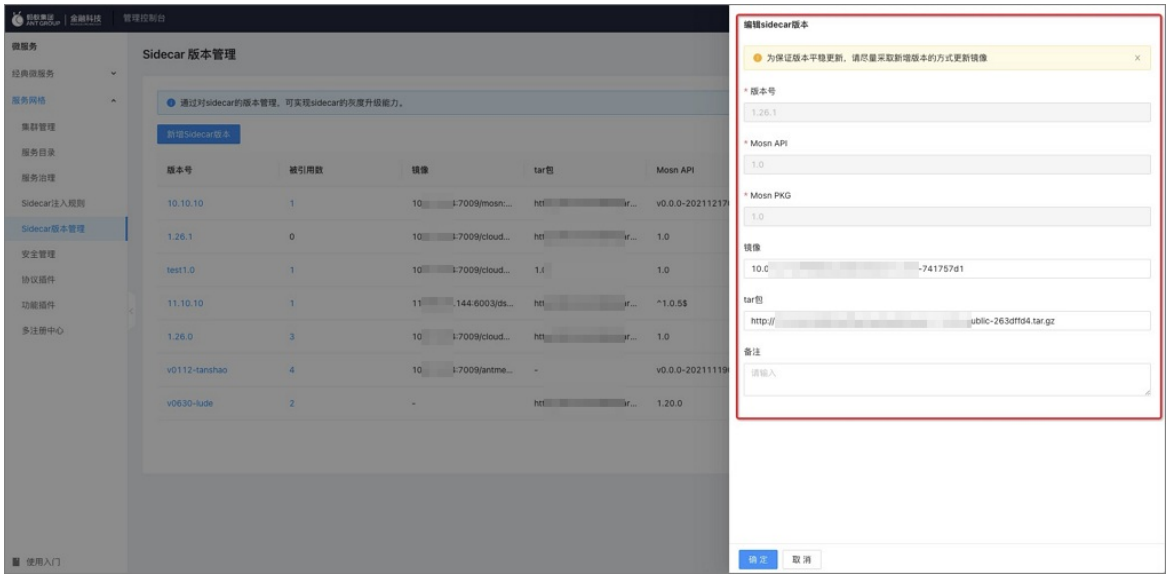
   针对注入了 Sidecar 的应用，如果需要在运行时对应用添加环境变量，可以在 Sidecar 动态配置中实现。

# 5.1. Sidecar 版本管理

由于 Sidecar 本身存在版本迭代，为了可以实现 Sidecar 的灰度发布，需要对 Sidecar 自身的版本进行管理。每新增一个 Sidecar 版本，都需要声明该版本的镜像地址（对应到容器场景）和二进制 TAR 包地址（对应到虚机场景）。Sidecar 版本管理便于从用户的视角管理一个特定的 MOSN 镜像或可执行文件。

## 新增 Sidecar 版本

1. 登录微服务控制台。
2. 在左侧导航栏单击 **服务网格** >**Sidecar 管理**，然后单击 **Sidecar 版本管理** 页签。
3. 单击 **新增 Sidecar 版本**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| 版本号 | 设置 Sidecar 的版本号。 |

| | |
|---|---|
| Mosn API | 填写 MOSN 和 Sidecar 插件依赖的 mosn.api 版本号，默认值为：`v0.0.0-20211217011300-b851d129be01` 。<br>说明<br>Sidecar 插件需要与 MOSN 主体具备相同的 API 和 PKG 版本，以确保插件与 MOSN 的兼容性。 |
| Mosn PKG | 填写 MOSN 和 Sidecar 插件插件依赖的 mosn.pkg 版本号，默认值为：`v0.0.0-20211217101631-d914102d1baf` 。 |
| 镜像 | 填写版本的镜像地址。镜像地址为 Sidecar 在镜像仓库中的地址。<br>若不清楚镜像地址，可联系现场交付同学。 |
| tar 包 | 填写版本的 TAR 包地址。 |
| 备注 | 填写版本的备注信息。 |

4. 单击 **确定**。

## 编辑 Sidecar 版本

1. 在版本列表单击目标版本右侧的 **编辑**。

   说明

   如果版本正在使用，则无法编辑。

2. 按照需求修改版本信息。

   您可以修改版本的镜像地址、TAR 包地址、备注信息。

3. 配置完成后，单击 **确定**。

## 删除 Sidecar 版本

1. 在版本列表单击目标版本右侧的 **删除**。

   说明

   如果版本正在使用，您需要先关闭使用该版本的 Sidecar 注入规则。

2. 单击 **确定**。

# 5.2. Sidecar 注入规则

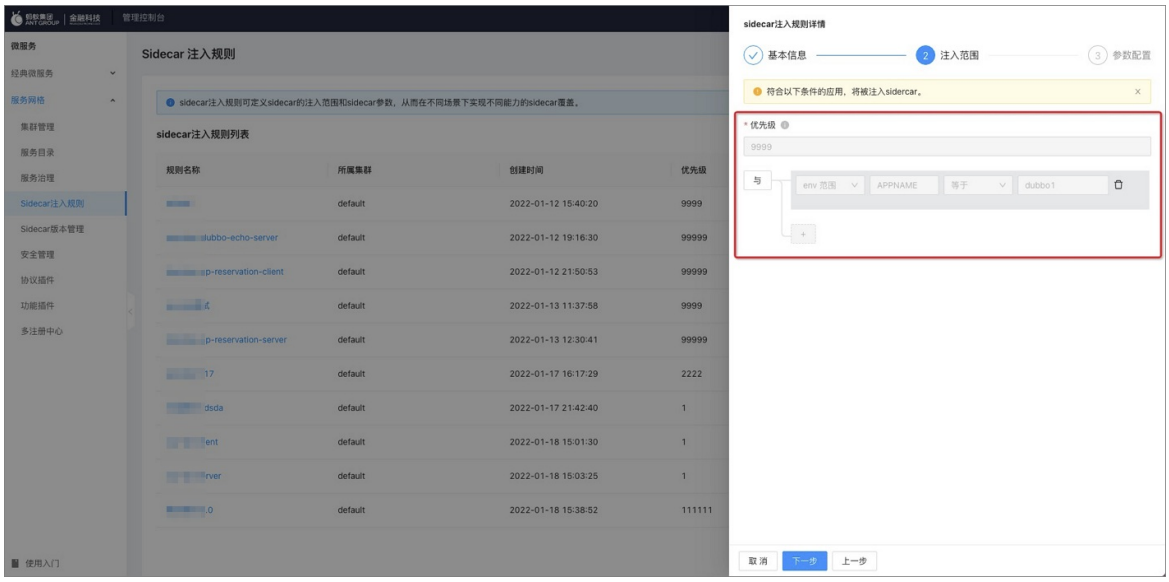
应用旁边的 Sidecar 需要通过注入规则的执行来完成安装。Sidecar 注入规则可定义 Sidecar 的注入范围和 Sidecar 参数（即哪些应用注入什么样的 Sidecar），从而在不同场景下实现不同能力的 Sidecar 覆盖。

## 新建 Sidecar 注入规则

1. 登录微服务控制台。
2. 在左侧导航栏单击 **服务网格** >**Sidecar 管理**，然后单击 **Sidecar 注入规则** 页签。
3. 单击 **新建注入规则**。
4. 在 **基本信息** 步骤填写以下参数后，单击 **下一步**。

| 参数 | 说明 |
| --- | --- |
| **规则名称** | 填写规则名称，最多不超过 128 个字符。 |
| **所属集群** | 选择注入规则生效的集群。<br>您也可以选中 **全局生效**，让该规则在所有集群生效。 |

5. 在 **注入范围** 步骤配置 **优先级** 和匹配条件后，单击 **下一步**。




- **优先级**：配置注入规则的优先级。数字越大，优先级越高。
- **匹配条件**：匹配条件根据选择的集群不同，配置信息也不同。支持多个规则的逻辑运算，当前只支持 **与** 运算，即当您配置多个条件时，所有条件同时匹配才能生效。

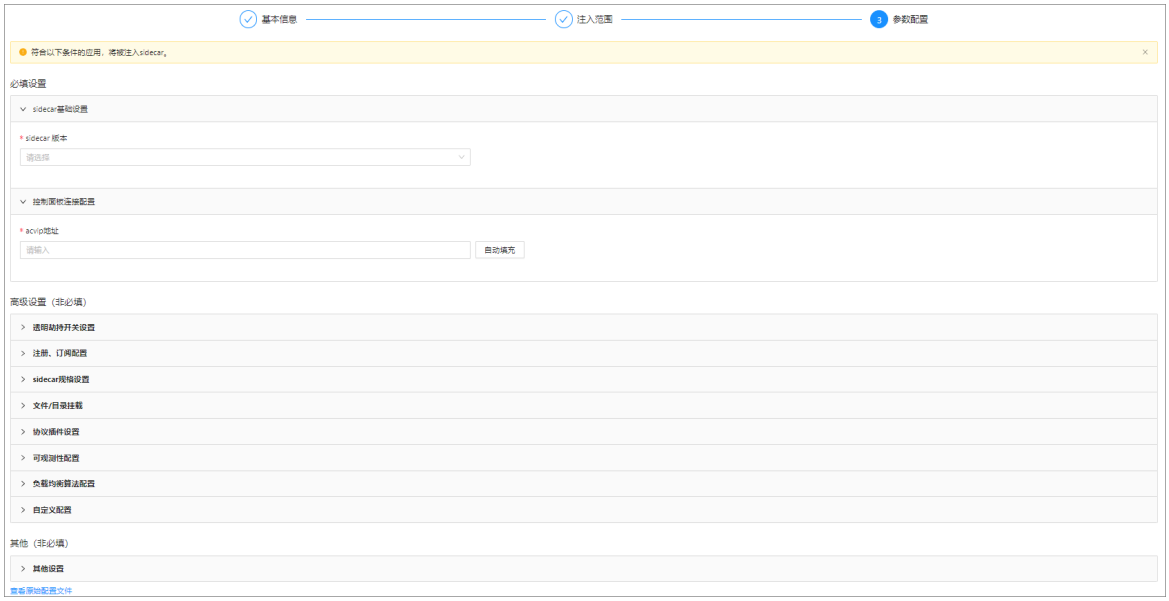
- 虚拟机集群

| 配置范围 | 字段名 | 逻辑 | 字段值 |
| --- | --- | --- | --- |
| **env 范围** | 输入字段名 | 支持等于、不等于、属于、不属于、正则五种逻辑。 | 配置字段名对应的字段值。 |
| **系统范围** | 应用名称 | | 配置应用名称。例如：crpc-client。 |
| | nodeip | | 配置节点的 IP 地址。例如：192.168.1.112。 |

- 容器

| 配置范围 | 字段名 | 逻辑 | 字段值 |
| --- | --- | --- | --- |
| **env 范围** | 输入字段名 | 支持等于、不等于、属于、不属于、正则五种逻辑。 | 配置字段名对应的字段值。 |
| **系统范围** | 应用名称 | | 配置应用名称。例如：crpc-client。 |
| | 集群名称 | | 配置集群名称。例如：hadoop_001。 |
| | deployment 名称 | | 配置 deployment 名称。例如：crpc-client-deploy。 |
| | sts 名称 | | 配置 sts 名称。例如：crpc-client-sts。 |
| **label 范围** | 输入字段名 | | 配置 label 字段值。例如：<br>`app: dubbo-client-zxy`<br>`app.kubernetes.io/instance: dubbo-client-zxy`<br>`app.kubernetes.io/name: dubbo-client-zxy`<br>`app.kubernetes.io/version: 1.0.0`<br>`cafe.sofastack.io/tenant: ALIPAYCN`<br>`cafe.sofastack.io/workspace: middleware` |
| **annotation 范围** | 输入字段名 | | 配置 annotation 字段值。例如：<br>`prometheus.io/path: /metrics`<br>`prometheus.io/port: "34903"`<br>`prometheus.io/scrape: "true"` |

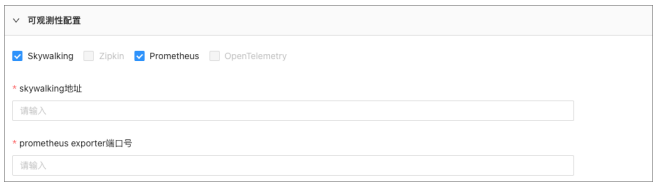
6. 在 **参数配置** 步骤配置以下参数后，单击 **提交**。

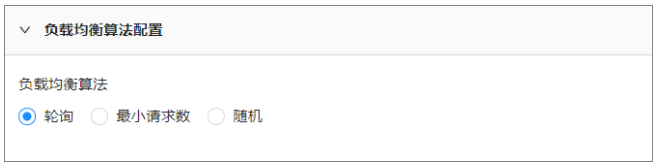
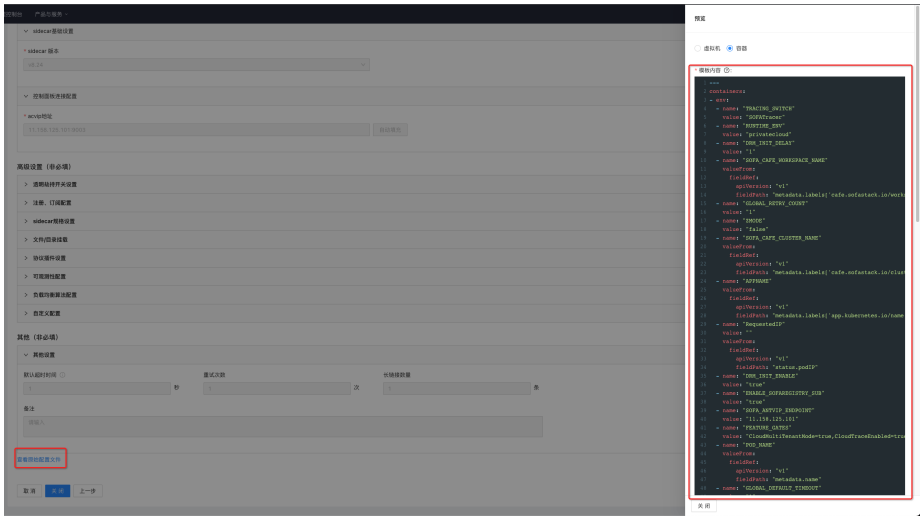



| 区域 | 参数 | 说明 |
| --- | --- | --- |
| **必填设置** | **sidecar 版本** | 选择 Sidecar 的版本。<br>如何创建版本请参见 Sidecar 版本管理。 |
| | **acvip 地址** | 填写 ACVIP 服务器地址，您可以单击 **自动填充**，系统会将 DRM 推送的 ACVIP 地址填入。 |

| | | |
|---|---|---|
| | **透明劫持开关设置** | 透明劫持可实现无侵入式的流量到 Sidecar 的劫持能力。开启透明劫持后，Sidecar 会直接从操作系统层面把流量劫持过来进行处理。透明劫持支持微服务体系和非微服务体系。<br> <br><br>○ **微服务体系**<br>■ **微服务类型**：当前支持 Dubbo、SOFA 和SpringCloud 三种微服务协议（单选）。<br>■ **注册中心**：是指劫持到服务发布/订阅的请求后，转发到哪个注册中心。可以选择在注册中心管理中填写的注册中心，支持多选。<br>○ **非微服务体系**：会将端口起点至端口终点范围内的流量进行劫持，并指定当前协议所对应的协议插件（协议插件用于对劫持到的流量按照约定的协议进行解析，转换为 MOSN 可以处理的格式），支持多个端口范围。协议插件的配置方式请参见 协议插件。 |

| | | |
|---|---|---|
| **高级设置**（可选） | **注册、订阅配置** | 设置注册和订阅配置：<br> <br><br>○ **注册配置**<br>填写应用的注册参数：<br>■ **注册中心**：如果不需要改变应用原来的注册行为，则选中 **高级配置** 中的 **使用应用自身的注册配置**；如果需要改变原有的注册行为，则需要在配置一个或多个注册中心，从而实现多注册行为。<br>置空时，会使用内置的 SOFARegistry。<br>■ **注册参数**：注册时，将填写的注册参数作为 metadata 注册到注册中心。您填写的值为 Key，Value 将会从环境变量或 Pod 标签中获取。<br>您可以单击 **添加更多注册参数**，同时添加多个注册参数。<br>○ **订阅配置**<br>填写应用的订阅配置：<br>■ **注册中心**：如果不需要改变应用原来的订阅行为，则选中 **高级配置** 中的 **使用应用自身的订阅配置**；如果需要改变原有的订阅行为，则需要填写一个或者多个注册中心，从而实现多订阅行为。<br>置空时，会使用内置的 SOFARegistry。<br>■ **服务发现优先策略**：若填写此项，则路由时会根据配置的 Key 优先进行路由的筛选。<br>您可以单击 **添加更多优先策略**，同时添加多个优先策略。<br>更多信息请参见 使用注册中心。 |

| | |
|---|---|
| **sidecar 规格设置** | 设置 Sidecar 的使用规格，如 1C2G100G 代表 CPU 为 1 核、内存 2 GB、磁盘 100 GB。<br>您可选择预定义的规格，也可以自定义，如果是自定义规格，必须保留 1 位小数，如 1.0。<br>仅针对容器场景生效。<br> |
| **文件/目录挂载** | 填写 Sidecar 目录和宿主机目录，将 Sidecar 容器内的目录挂在到宿主机上，以满足日志采集等场景。<br>仅针对容器场景生效。<br> |
| **协议插件设置** | 输入或选择指定协议插件。<br>通过加载协议插件，可让 Sidecar 支持第三方协议。 |
| **可观测性配置** | 可观测性设置指的是 Sidecar 可以对接 SOFATracer（系统内置，不需要配置）、Skywalking、Zipkin、Prometheus 或 OpenTelemetry 做链路和监控信息的上报，从而支持链路追踪，监控等可观测性功能。<br>  <br>Skywalking、Zipkin、OpenTelemetry 都是链路平台，所以是互斥的，只能选择一个，而且要填写对应的上报地址。<br>Prometheus 为系统自带，因此只需要声明 exporter 的端口号即可（一般不填使用默认值即可），内置的 Prometheus 会自动到 Sidecar 的这个端口上收集数据。 |

| | | |
|---|---|---|
| | | 设置负载均衡算法，指定调用方找到可用的服务列表后，按照什么策略选择一个目标地址调用。支持轮训、最小请求数和随机三种：<br>  <br><br>○ **轮训**：按照顺序依次选择目标地址调用。<br>○ **最小请求数**：选择请求此处最少的目标地址调用。<br>○ **随机**：随机选择目标地址调用。 |
| | **自定义配置** | 配置指定 Key 和 Value，该配置将作为环境变量添加在 Sidecar 上。<br> |
| **其他**（可选） | **默认超时时间** | 设置注入规则的超时时间。<br>Sidecar 超时时间会优先从请求中获取，如果获取不到，则采用默认值。 |
| | **重试次数** | 设置注入失败的重试次数。 |
| | **长链接数量** | 设置注入规则的长链接数量。 |
| | **备注** | 添加规则的备注信息。 |
| **查看原始配置文件** | 配置完成后，单击 **查看原始配置文件**，可以预览 Sidecar 最后的 YAML 文件。<br>  | |

> **重要**
>
> 模板配置完成后，默认是不开启的，需要单击 **开启** 按钮生效。

## 复制注入规则

您可以通过复制已创建的规则，快速生成一个新的规则。

1. 在注入规则列表单击目标注入规则右侧的 **复制**。
2. 按照流程配置注入规则的相关参数。

   具体配置与创建规则一致，您可以对复制的规则参数进行修改。
3. 配置完成后，单击 **提交**。

## 编辑注入规则

1. 在注入规则列表单击目标注入规则右侧的 **编辑**。

   > **说明**
   >
   > 如果规则处于开启状态，您需要先将生效状态改为关闭。

2. 按照流程修改注入规则的相关参数。

   您可以修改规则的注入范围和参数配置。
3. 配置完成后，单击 **提交**。

## 删除注入规则

1. 在注入规则列表单击目标注入规则右侧的 **删除**。

   > **说明**
   >
   > 如果规则处于开启状态，您需要先将生效状态改为关闭。

2. 单击 **确定**。

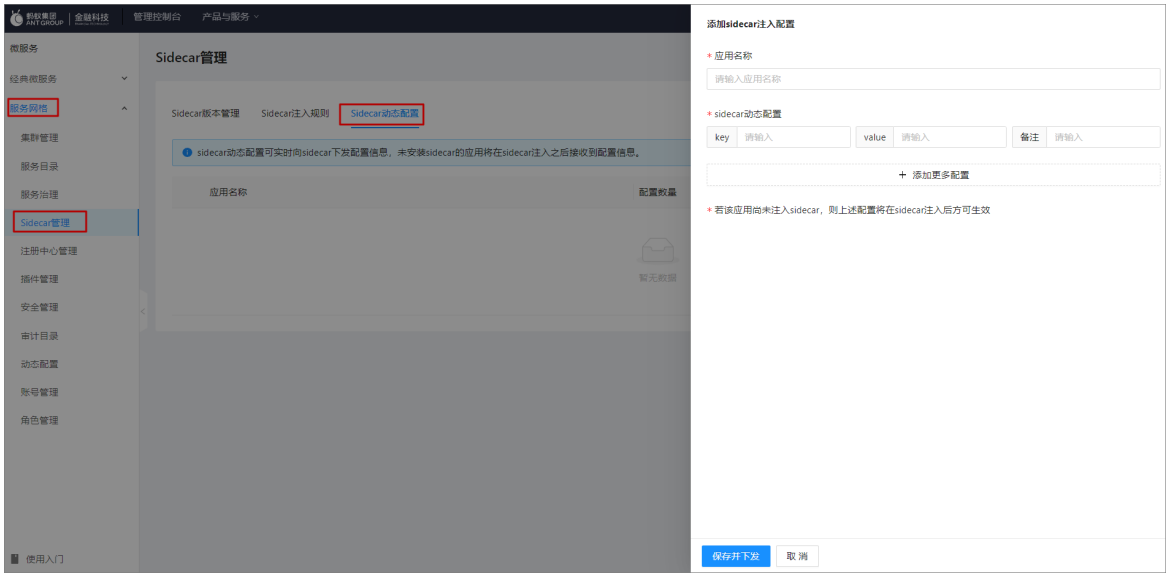
# 5.3. Sidecar 动态配置

动态配置为开发者提供一种数据动态变更的操作，您可以借助 API 实现相关数据的读取。

> **说明**
>
> 动态配置开发可以参考 用户开发手册 和 开发视频。

## 新建 Sidecar 动态配置

1. 登录微服务控制台。
2. 在左侧导航栏单击 **服务网格** >**Sidecar 管理**，然后单击 **Sidecar 动态配置** 页签。
3. 单击 **新建配置**，然后配置以下参数：



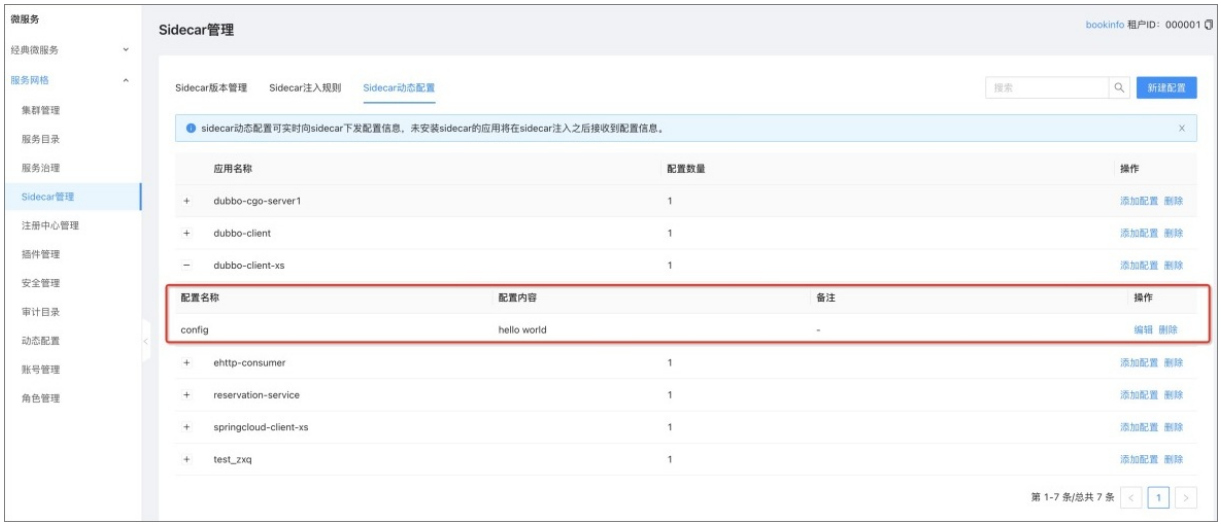

| 参数 | 说明 |
| --- | --- |
| 应用名称 | 服务目录中应用对应的名字。例如设置为 reservation-service，保存配置后，所有 reservation-service 应用都会收到这个配置。<br>最多不超过 255 个字符。 |
| sidecar 动态配置 | 设置动态配置的 Key、Value 和备注信息。<br>○ key：开发者在数据共享中心通过 SDK 获取的数据的标识。<br>○ value：数据的实际值。<br>○ 备注：建议备注当前配置使用场景。<br>单击 **添加更多配置** 可一次添加多个动态配置。 |

4. 单击 **保存并下发**。

> **说明**
>
> 应用中可以过滤日志 `[xprotocol] [config] receive dataId` 查看是否下发成功。

## 查看 Sidecar 动态配置

配置保存后，您可以在动态配置列表查看到配置的信息：




- 配置名称：数据的标识，即配置时填写的 Key。
- 配置内容：数据的值，即配置时填写的 Value。
- 备注：即配置时填写的备注信息，一般建议备注当前配置使用场景。

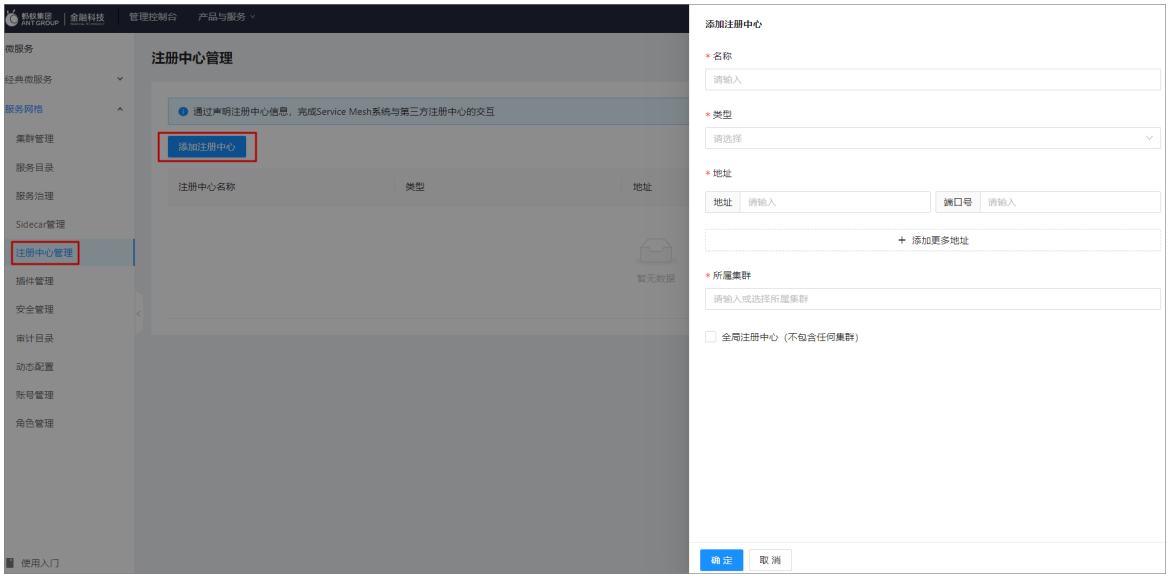
# 6.注册中心管理

## 6.1. 添加注册中心

Service Mesh 基于服务注册和发现的基本原理，经常涉及对接不同的注册中心，从而带来不同的注册中心访问协议、数据格式等差异。多注册中心页面可统一管理注册中心的元信息，通过在 Sidecar 注入时进行注册和订阅的配置，使 Sidecar 沿袭、增强或改变原有的注册订阅行为。

### 配置步骤

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 注册中心管理**。
3. 单击 **添加注入中心**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **名称** | 配置的注册中心名称。 |
| **类型** | 选择注册中心类型，支持 ZooKeeper、Eureka、SOFARegistry、Consul、Nacos。 |
| **地址** | 填写注册中心的 IP 地址（也可以填写域名）和端口号。<br>单击 **添加更多地址**，可添加多个注册中心地址。<br>**重要**<br>同一个租户的同一个集群下，IP 地址不能重复添加。 |

| | |
|---|---|
| **所属集群** | 选择注册中心所在集群。<br>配置后，您可以在 **服务目录** 页面看到集群应用和其注册信息的对应关系。如果某个注册中心与所有集群都有关联，或者无法确定关联关系，可选中 **全局注册中心**。 |

4. 单击 **确定**。

# 6.2. 使用注册中心

您可以通过 Sidecar 注入规则为 Sidecar 注册或订阅注册中心。

Sidecar 注入规则配置步骤，请参见 Sidecar 注入规则。其中，注册和订阅配置如下：

## 注册配置

| 参数 | 说明 |
|---|---|
| **注册中心**（可选） | 选择在添加注册中心时录入的注册中心，可以选择多个，Sidecar 会同时向选中的注册中心进行注册。置空时，会使用内置的 SOFARegistry。<br>如果不需要改变应用原来的注册行为，则直接选中 **高级配置** 中的 **使用应用自身的注册配置** 即可。 |
| **注册参数**（可选） | Sidecar 会从环境变量或 Pod 标签获取该值，并以该 Key-Value作为 metadata，与服务一并注册到注册中心（例如标记当前数据中心的 datacenter 参数，或者 K8s 的 label。可用于服务发现优先策略的场景）。 |

| | |
|---|---|
| **高级配置** | • **注册到共享注册中心**：共享注册中心是 Mesh 中的注册中心（当前默认为 SOFARegistry，注册时一定会向该注册中心注册，作为 Mesh 管控台的数据来源）。<br>• **应用自身的注册配置**：选择是否使用应用自身的订阅配置，一般不建议选中，除非接入了 SOFA SDK。<br>如果取消选中 **使用应用自身的注册配置**，则无论应用中如何配置注册中心的地址，都会被上面 Sidecar 中配置的注册中心地址所覆盖。 |

## 配置示例

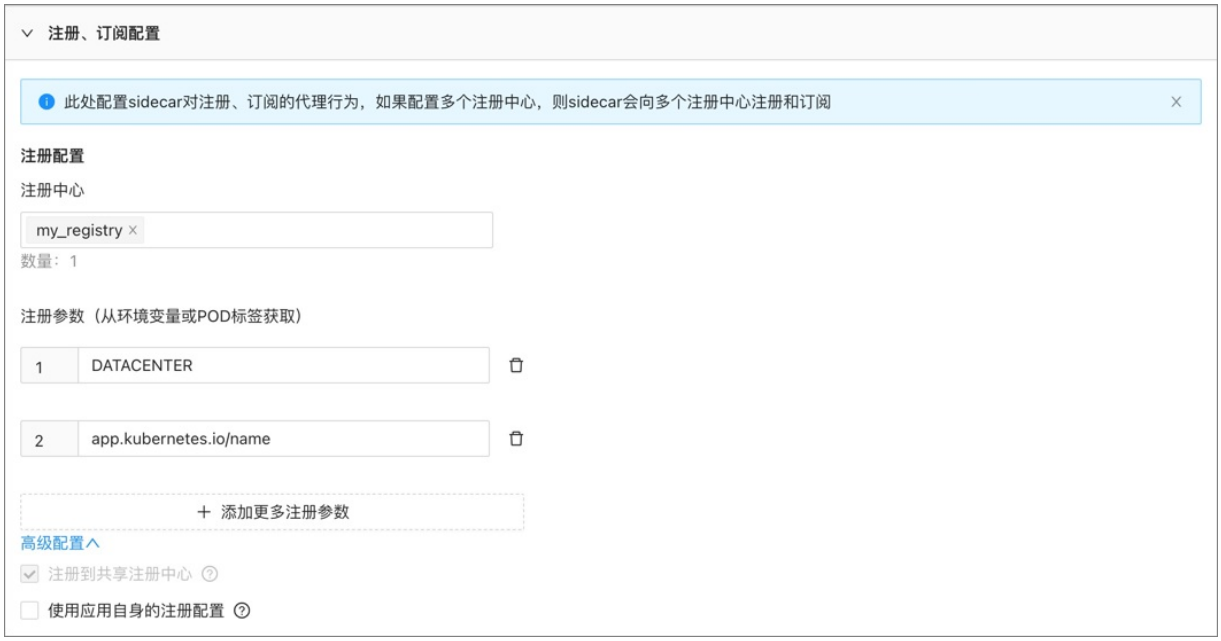
如当前环境中某个应用的 K8s YAML 为：

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app: mesh-dubbo-provider
  name: mesh-dubbo-provider
  namespace: hz
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mesh-dubbo-provider
  template:
    metadata:
      labels:
        app: mesh-dubbo-provider
        app.kubernetes.io/name: mesh-dubbo-provider
    spec:
      containers:
      - env:
        - name: DATACENTER
          value: hangzhou-a
        - name: dubbo.registry.address
          value: zookeeper://127.0.0.1:2181
        - name: dubbo.protocol.port
          value: "20880"
        - name: com.alipay.instanceid
          value: 000
        image: xxx/xxx/xxx/mesh-dubbo-provider:test.image.tag
        imagePullPolicy: IfNotPresent
        name: server
      terminationGracePeriodSeconds: 30
  updateStrategy:
    rollingUpdate:
      partition: 0
    type: RollingUpdate
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app: mesh-dubbo-provider
  name: mesh-dubbo-provider
  namespace: hz
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mesh-dubbo-provider
  template:
    metadata:
      labels:
        app: mesh-dubbo-provider
        app.kubernetes.io/name: mesh-dubbo-provider
    spec:
      containers:
      - env:
        - name: DATACENTER
          value: hangzhou-b
        - name: dubbo.registry.address
          value: zookeeper://127.0.0.1:2181
        - name: dubbo.protocol.port
          value: "20880"
        - name: com.alipay.instanceid
          value: 000
        image: xxx/xxx/xxx/mesh-dubbo-provider:test.image.tag
        imagePullPolicy: IfNotPresent
        name: server
      terminationGracePeriodSeconds: 30
  updateStrategy:
    rollingUpdate:
      partition: 0
    type: RollingUpdate
```

该应用作为 provider 存在，同时在 hangzhou-a 和 hangzhou-b 这两个机房都有该应用。该应用的 Sidecar 注入规则如上文注册配置，在向注册中心注册时会携带各自的 datacenter 元数据信息。

Sidecar 注入规则中的注册配置如下：




则注册结果如下：

- hangzhou-a 机房的应用在向 ZooKeeper 注册服务时，会把 `DATACENTER=hangzhou-a` 以及 `app.kubernetes.io/name=mesh-dubbo-provider` 作为 metadata 一并写入注册中心中，作为该服务附带的 metadata 属性。
- hangzhou-b 机房的应用在向 ZooKeeper 注册服务时，会把 `DATACENTER=hangzhou-b` 以及 `app.kubernetes.io/name=mesh-dubbo-provider` 作为 metadata 一并写入注册中心中，作为该服务附带的 metadata 属性。

## 订阅配置

| 参数 | 说明 |
| --- | --- |
| **注册中心**（可选） | 选择在添加注册中心时录入的注册中心，可以选择多个，Sidecar 会同时向选中的注册中心进行注册。置空时，会使用内置的 SOFARegistry。<br>如果不需要改变应用原来的注册行为，则直接选中 **高级配置** 中的 **使用应用自身的注册配置** 即可。 |
| **服务发现优先策略**（可选） | 与注册配置中的 **注册参数** 搭配使用，则路由时会根据配置的 Key 优先进行路由的筛选。 |
| **高级配置** | • **从共享注册中心订阅**：共享注册中心是 Mesh 中的注册中心SOFARegistry，订阅时可以不选择该项，SOFARegistry 会持有所有通过 Sidecar 注册的应用或服务信息。默认选中。<br>• **使用应用自身的订阅配置**：选择是否使用应用自身的订阅配置，一般不建议选中，除非接入了 SOFA SDK。<br>如果不选中 **使用应用自身的订阅配置**，则无论应用中如何配置注册中心的地址，都会被上面 Sidecar 中配置的注册中心地址所覆盖。 |

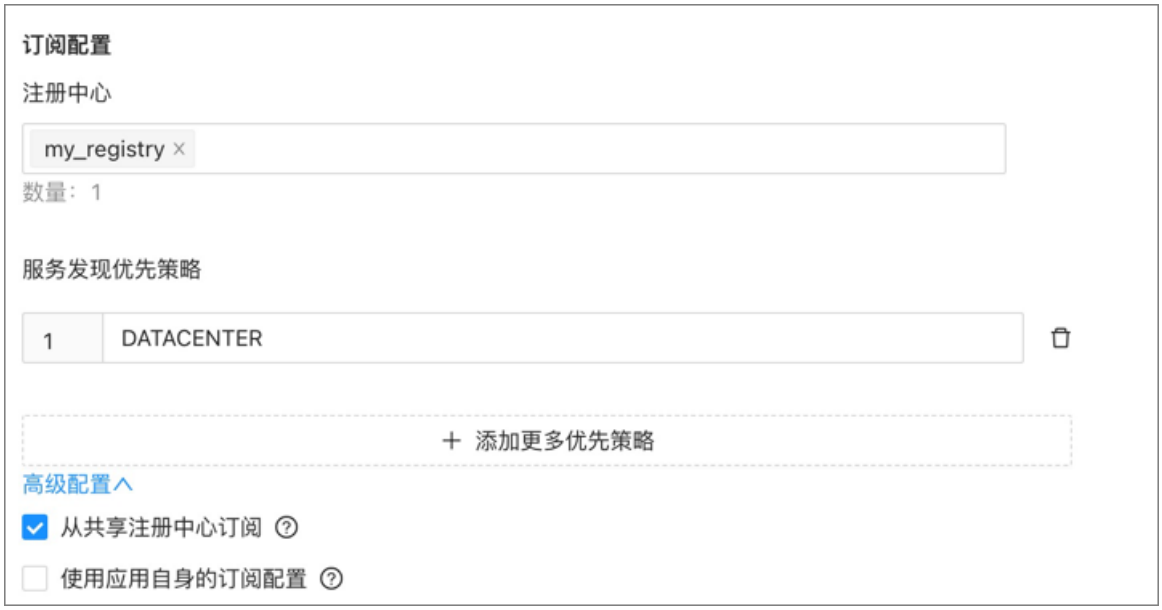
## 配置示例

如当前环境中某个应用的 K8s YAML 为：

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app: mesh-dubbo-client
  name: mesh-dubbo-client
  namespace: hz
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mesh-dubbo-client
  template:
    metadata:
      labels:
        app: mesh-dubbo-client
        app.kubernetes.io/name: mesh-dubbo-client
    spec:
      containers:
      - env:
        - name: DATACENTER
          value: hangzhou-a
        - name: dubbo.registry.address
          value: zookeeper://127.0.0.1:2181
        - name: dubbo.protocol.port
          value: "20880"
        - name: com.alipay.instanceid
          value: 000
        image: xxx/xxx/xxx/mesh-dubbo-client:test.image.tag
        imagePullPolicy: IfNotPresent
        name: server
      terminationGracePeriodSeconds: 30
  updateStrategy:
    rollingUpdate:
      partition: 0
    type: RollingUpdate
```

该应用作为 consumer 存在，且部署在 hangzhou-a 机房。

订阅配置如下：




当该部署在 hangzhou-a 机房的 consumer 应用访问上述分别部署在 hangzhou-a 机房和 hangzhou-b 机房的 provider 应用时，请求会优先去往部署在 hangzhou-a 机房的 provider 应用。

# 6.3. 编辑或删除注册中心

您可以随时修改已创建的注册中心信息，对于不再需要的注册中心，您可以选择删除。

## 编辑注册中心

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 注册中心管理**。
3. 单击目标注册中心右侧的 **编辑**。

4. 修改完成后，单击 **确定**。

注册中心的地址修改后，需要把对应 Sidecar 注入规则关闭后再开启，才能真正生效新修改的地址。

## 删除注册中心

1. 在注册中心列表单击目标注册中心右侧的 **删除**。
2. 然后单击 **确定**。

# 7.插件管理

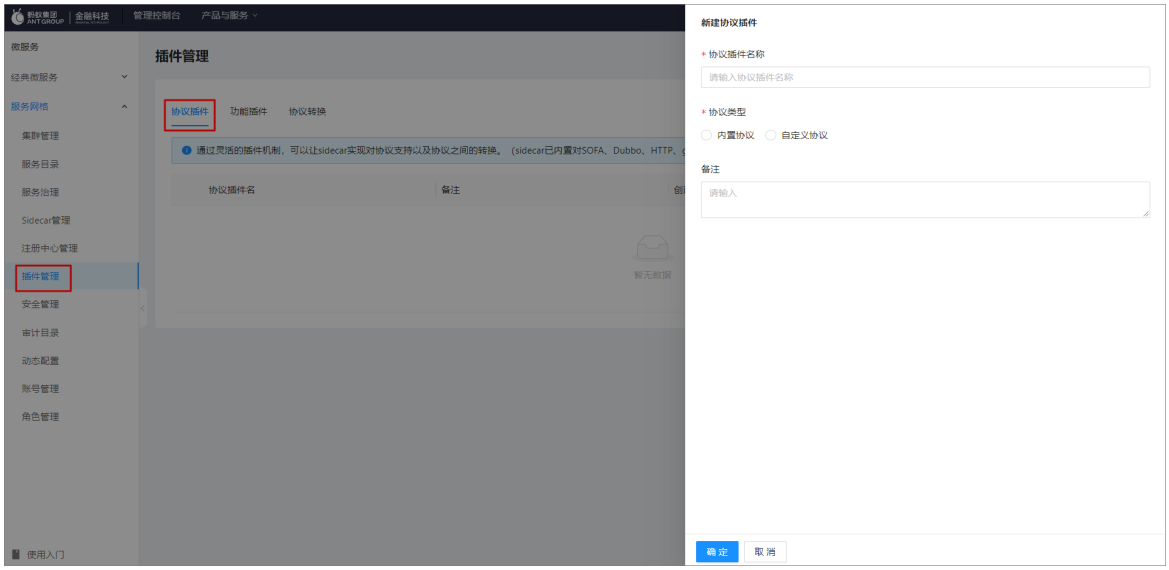
## 7.1. 协议插件

Mesh 作为一个应用层的数据治理的组件，提供灵活的协议扩展能力，实现对第三方协议支持、协议之间的转换简单灵活的接入。目前 Mesh 支持 SOFA、Dubbo、HTTP、SpringCloud 协议，如果无法满足用户需求可以通过实现协议插件完成接入。

> **说明**
>
> 协议插件开发请参见 用户开发手册 和 开发视频。

### 新建协议插件

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 协议插件**。
3. 单击 **新建协议插件**，配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **协议插件名称** | 设置协议插件的名称。 |
| **协议类型** | 选择协议插件的类型：<br>○ **内置协议**：选择内置的协议类型，支持 CRPC、Bolt、Dubbo。<br>○ **自定义协议**：输入自定义的协议插件类型。 |
| **备注** | 填写插件的备注信息。 |

4. 单击 **确定**。

## 新增版本

每个协议插件都可以配置多个版本，您可以通过版本控制实现灰度发布能力。进行版本控制前，您需要编译并打包协议插件。

> **说明**
>
> 例如编译一个 SpringCloud 协议扩展插件，可进入 插件项目，编译 SpringCloud 协议扩展插件。开发步骤请参见 用户开发手册。
>
> ```
> make codec plugin=springcloud && make pkg-codec plugin=springcloud
> ```

1. 在协议插件列表，单击目标插件右侧的 **新增版本**，然后配置以下参数：
   - 内置协议

     若您创建的协议插件使用的是内置协议，则配置以下参数：

| 参数 | 说明 |
| --- | --- |
| **版本号** | 设置版本号。 |
| **内置协议版本** | 选择内置协议的版本。 |
| **备注** | 设置版本的备注信息。 |

| | |
|---|---|
| **关联功能插件** | 选中此项可以关联一个功能插件，实现基于该协议的功能扩展。常见的功能插件有：协议转换、日志格式化、自定义路由等。<br>功能插件的配置方式请参见 功能插件。 |

- 自定义协议

若您创建的协议插件使用的是自定义协议，则配置以下参数：

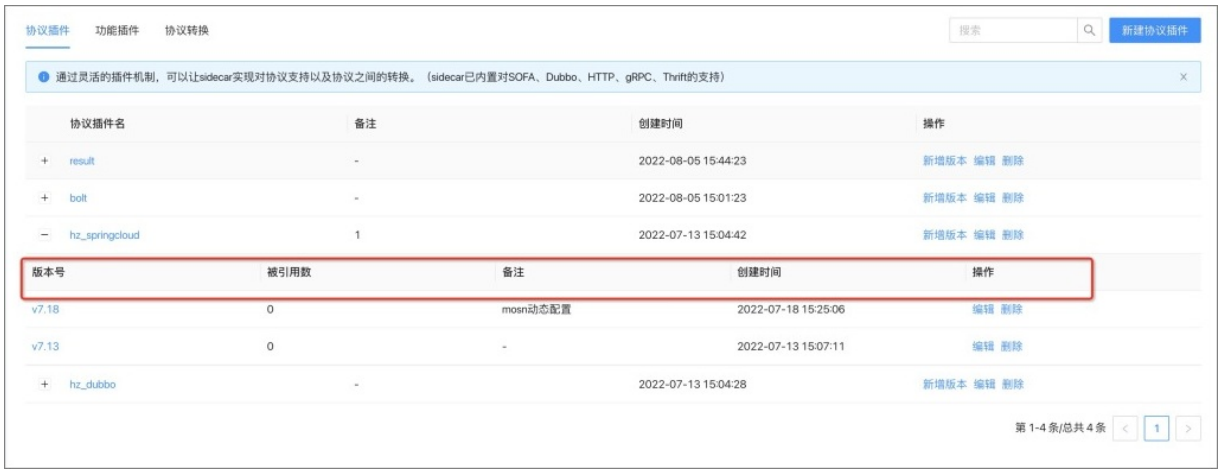
| 参数 | 说明 |
|---|---|
| **版本号** | 设置版本号。 |

| | |
|---|---|
| **插件主体** | 配置插件主体，支持以下两种方式：<br>■ 上传文件：切换到上传模式后，您可以直接上传插件主体文件。<br>■ 制品库地址：切换到制品库模式后，您可以直接配置插件主体的地址，系统会通过该地址获取插件主体。<br>目前仅支持 ZIP 格式的文件。 |
| **备注** | 设置版本的备注信息。 |
| **关联功能插件** | 选中此项可以关联一个功能插件，实现基于该协议的功能扩展。常见的功能插件有：协议转换、日志格式化、自定义路由等。<br>功能插件的配置方式请参见 功能插件。 |

2. 单击 **确定**。

## 查看协议插件

协议插件配置完成后，可在插件列表中查看协议插件的信息：




- 版本号：协议插件的版本号。
- 被引用数：被打开 sidecar 的规则引入数量。
- 备注：版本的备注信息。
- 创建时间：插件创建时间。
- 操作：支持编辑和删除插件。删除插件时，需要将引用的插件 Sidecar 注入规则全部释放掉。

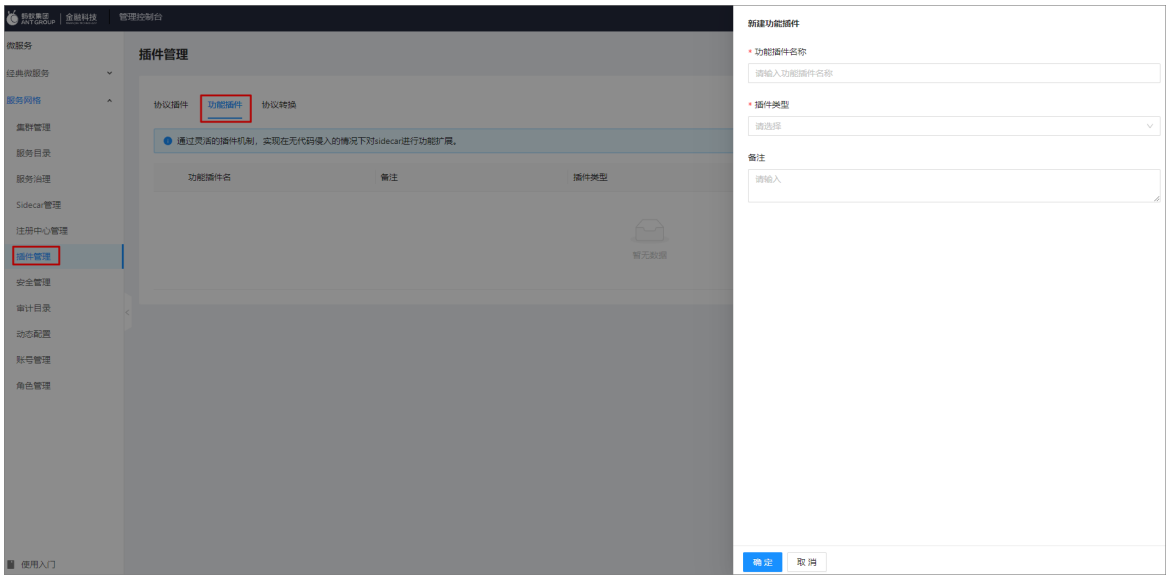
# 7.2. 功能插件

与协议插件类似，功能插件也是基于 Go plugin 机制实现。不同的是，功能插件必须关联到一个协议插件上才可以生效。功能插件也可以进行多版本管理，实现灰度发布的能力。

> **说明**
>
> 功能插件开发请参见 用户开发手册 和 开发视频。

## 新建功能插件

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 功能插件**。
3. 单击 **新建功能插件**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **功能插件名称** | 设置功能插件的名称。 |
| **插件类型** | 选择功能插件的类型：<br>○ **协议转换插件**：协议转换可以做到不同协议之间的互转，实现异构互通。<br>○ **拦截器插件**：拦截器支持普遍意义上的 filter 机制，比如日志格式化、自定义路由、第三方系统对接等。 |
| **备注** | 填写插件的备注信息。 |

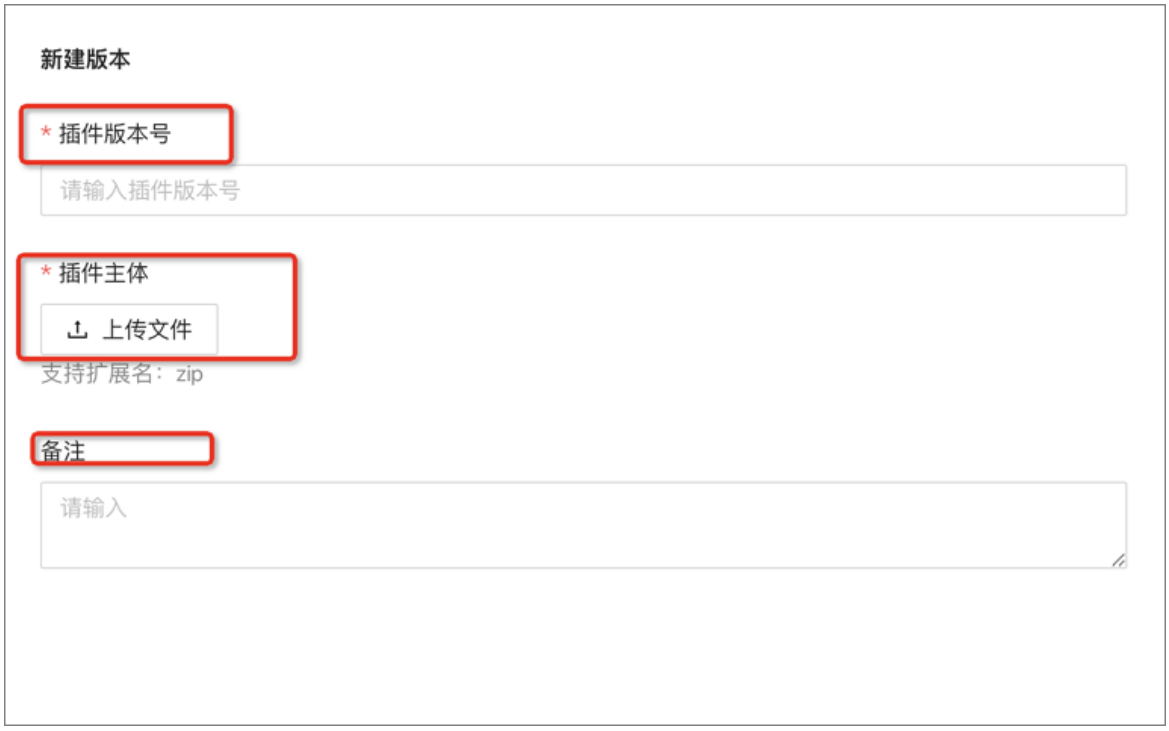
4. 单击 **确定**。

## 新增版本

每个功能插件都可以配置多个版本，您可以通过版本控制实现灰度发布能力。进行版本控制前，您需要编译并打包功能插件。

> **说明**
>
> 您可进入 插件项目 编译插件，编译方式请参见 拦截器编译。

1. 在功能插件列表，单击目标插件右侧的 **新增版本**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **插件版本号** | 设置功能插件的版本号。 |
| **插件主体** | 上传功能插件的主体文件，仅支持 ZIP 格式。 |
| **备注** | 设置版本的备注信息。 |

2. 单击 **确定**。

## 查看功能插件

功能插件配置完成后，可在插件列表中查看协议插件的信息：

- 版本号：功能插件的版本号。
- 被引用数：被打开 sidecar 的规则引入数量。
- 备注：版本的备注信息。
- 创建时间：插件创建时间。
- 操作：支持编辑和删除插件。删除插件时，需要将引用的插件 Sidecar 注入规则全部释放掉。

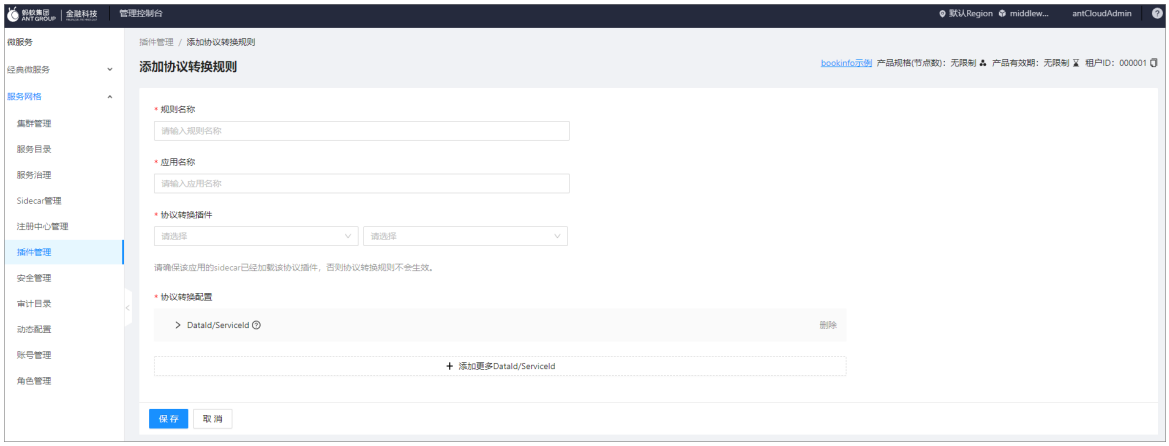
# 7.3. 协议转换

协议转换是为协议转换提供配置下发能力，您可以通过关联协议插件动态下发协议转换配置。

> **说明**
>
> 协议转换配置详细说明请参见 协议转换插件。

## 配置步骤

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 插件管理**，然后单击 **协议转换** 页签。
3. 单击 **添加协议转换规则**，然后配置以下参数：



| 参数 | 说明 |
| --- | --- |
| 规则名称 | 设置协议转换规则的名称，最多不超过 255 个字符。 |
| 应用名称 | 填写服务目录中应用对应的名称。 |
| 协议转换插件 | 选择在功能插件中配置的协议转换插件和版本。 |
| 协议转换配置 | 当前协议转换插件的动态配置：<br>○ DataId/ServiceId：从客户端请求中解析出来的服务唯一标识。<br>○ **字段映射配置**：两种协议之间请求头，请求体的字段映射配置。<br>您可以单击 **添加更多DataId/ServiceId** 一次添加多个配置。 |

4. 单击 **保存**。
5. 将协议转换规则的 **状态** 改为 **开启**。

说明

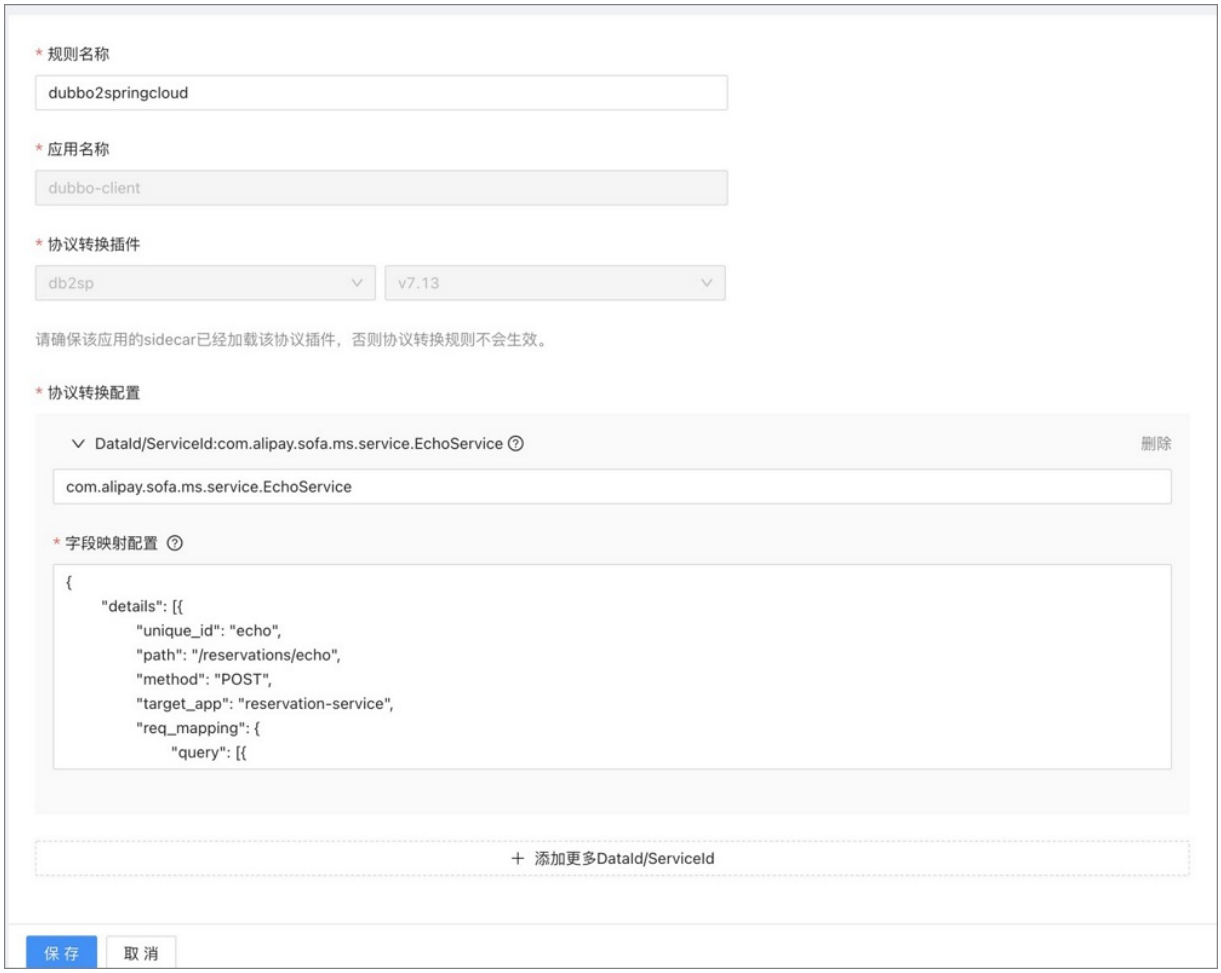
## 查看协议转换规则

协议转换规则配置完成后，您可以在规则列表中查看协议转换规则：

- 规则名称：协议转换规则的规则名称。
- 协议转换插件：功能插件中引用协议转换插件。
- 插件版本：功能插件中引用协议转换插件版本号。
- 状态：协议转换配置生效状态。状态为 **开启** 才能生效。
- 操作：支持插件的更新和删除操作。

## 配置示例

协议转换开发示例请参见 标准 Dubbo 和 SpringCloud 协议互转实践 - 线上实战篇，配置如下：




| 参数 | 内容 |
| --- | --- |
| 规则名称 | dubbo2springcloud |
| 应用名称 | dubbo-client |
| 协议转换插件 | db2sp & v7.13 提前上传好协议转换插件，关联即可 |
| DataId | com.alipay.sofa.ms.service.EchoService |
| 字段映射配置 | `{"details": [{"unique_id":"echo","path":"/reservations/echo","method":"POST","target_app":"reservation-service","req_mapping":{"query": [{"type":"java.lang.String","key":"message"}]}}]}` |

配置完成后，将规则状态修改为 **开启**，配置将下发到应用的服务节点。

# 7.4. 最佳实践

## 背景信息

公司 A 了解 Mesh 可以解决异构系统集成问题，具备数据流统一治理，业务无感知接入能力，想深入体验一下。具体要求如下：

- 实现扩展协议 dubbo 和 springcloud 的接入。
- 实现 dubbo 应用对 springcloud 服务的互通集成。
- 记录每次访问的耗时、协议名称、应用名称、访问时间。

## 前期准备

### dubbo 和 springcloud 协议插件

当前 插件项目 实现了dubbo、springcloud 协议插件，打包命令如下：

```
make codec plugin=springcloud && make pkg-codec plugin=springcloud
make codec plugin=dubbo && make pkg-codec plugin=dubbo
```

### 日志插件

当前 插件项目 实现了 logger 插件，打包命令如下：

```
make filter plugin=logger && make pkg-filter plugin=logger
```

### dubbo 和 springcloud 协议转换插件

当前 插件项目 实现了 dubbo 转 springcloud 插件，打包命令如下：

```
make trans plugin=dubbo2springcloud && make pkg-trans plugin=dubbo2springcloud
```

### dubbo 应用

dubbo 应用作为客户端，YAML 如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app: dubbo-client
  name: dubbo-client
  namespace: hz
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dubbo-client
  serviceName: dubbo-client
  template:
    metadata:
      labels:
        app: dubbo-client
        app.kubernetes.io/name: dubbo-client
        app.kubernetes.io/version: 1.0.0
        bjapp: dubbo
    spec:
      containers:
      - env:
        - name: APPNAME
          value: dubbo-client
        image: yidwang/dubbo-springcloud:v1.0
        imagePullPolicy: IfNotPresent
        name: dubbo-client
```

## springcloud 应用

springcloud 应用作为服务端，YAML 如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app: springcloud-server
  name: springcloud-server
  namespace: hz
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: springcloud-server
  serviceName: springcloud-server
  template:
    metadata:
      labels:
        app: springcloud-server
        app.kubernetes.io/name: springcloud-server
        app.kubernetes.io/version: 1.0.0
        bjapp: springcloud
    spec:
      containers:
      - env:
        - name: APPNAME
          value: springcloud
        - name: PORT
          value: "18080"
        image: yidwang/demo_http_server
        imagePullPolicy: IfNotPresent
        name: springcloud-server
```

## 系统集成

## 服务部署

1. 部署 dubbo 服务。

```
kubectl apply -f dubbo.yaml
```

2. 部署 springcloud 服务。

```
kubectl apply -f springcloud.yaml
```

3. 查看服务状态。

```
whj at bogon in ~/workflow/1.21-test
$ k9s get pod dubbo-client-0  -n hz
NAME              READY   STATUS    RESTARTS   AGE
dubbo-client-0    1/1     Running   0          5m13s

whj at bogon in ~/workflow/1.21-test
$ k9s get pod springcloud-server-0 -n hz
NAME                    READY   STATUS    RESTARTS   AGE
springcloud-server-0    1/1     Running   0          65s
```

## 功能插件

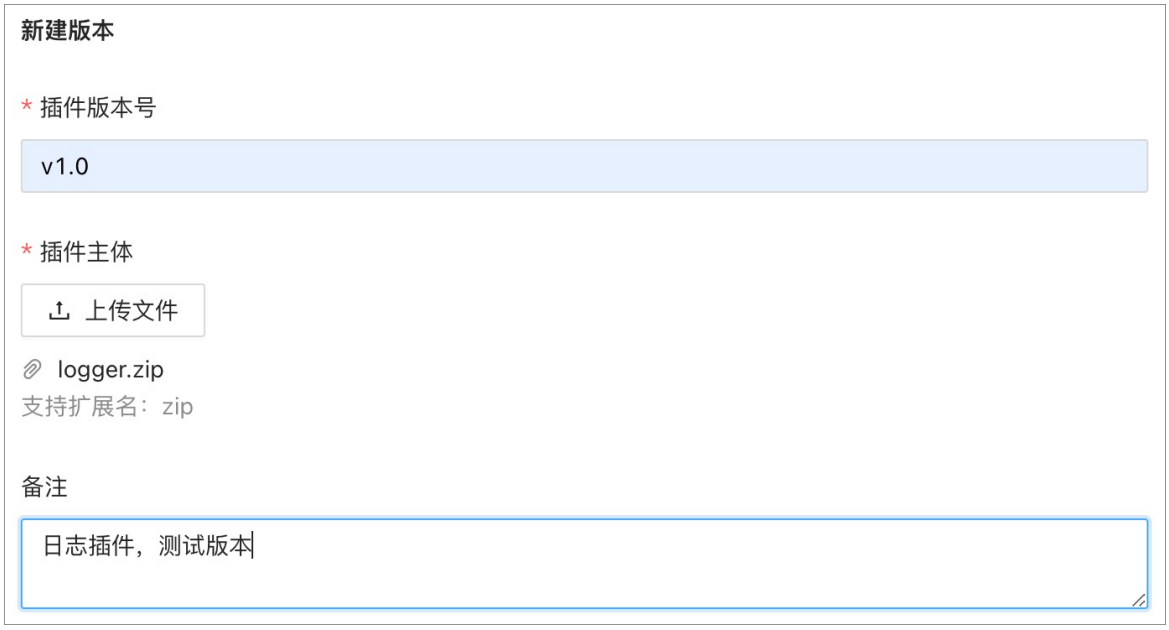
### 日志插件

1. 新建功能插件。
   i. 登录微服务控制台。
   ii. 选择 **服务网格** > **插件管理**，然后单击 **协议插件** 页签。
   iii. 单击**新建功能插件**，然后配置以下参数：

| 参数 | 内容 |
| --- | --- |
| 功能插件名称 | logger |
| 插件类型 | 拦截器 |
| 备注 | 日志插件，打印 Access 日志使用。 |

2. 新增功能插件版本。

   单击 **新建版本**，然后配置以下参数：




| 参数 | 内容 |
| --- | --- |
| 插件版本号 | v1.0。 |
| 插件主体 | 上传编译好的日志压缩包。 |
| 备注 | 日志插件，测试版本。 |

## 协议转换插件

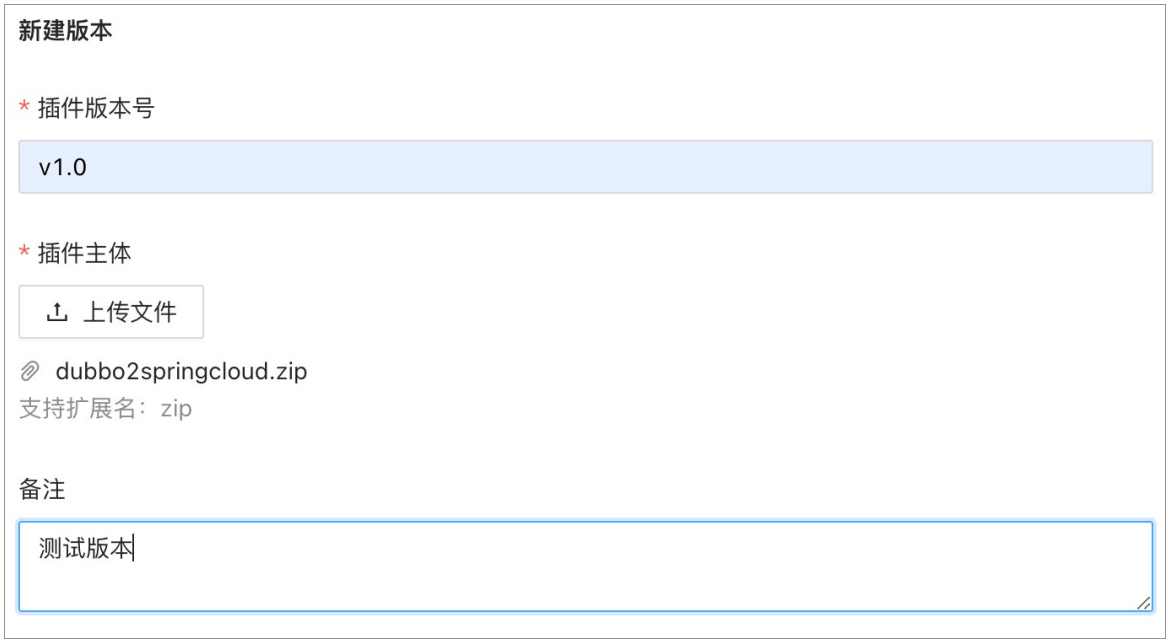
1. 新建功能插件。
    i. 单击 **协议插件** 页签。

ii. 单击新建功能插件，然后配置以下参数：

| 参数 | 内容 |
| --- | --- |
| 功能插件名称 | dubbo2springcloud。 |
| 插件类型 | 协议转换。 |
| 备注 | dubbo 转 springcloud 协议转换插件。 |

2. 新增功能插件版本。

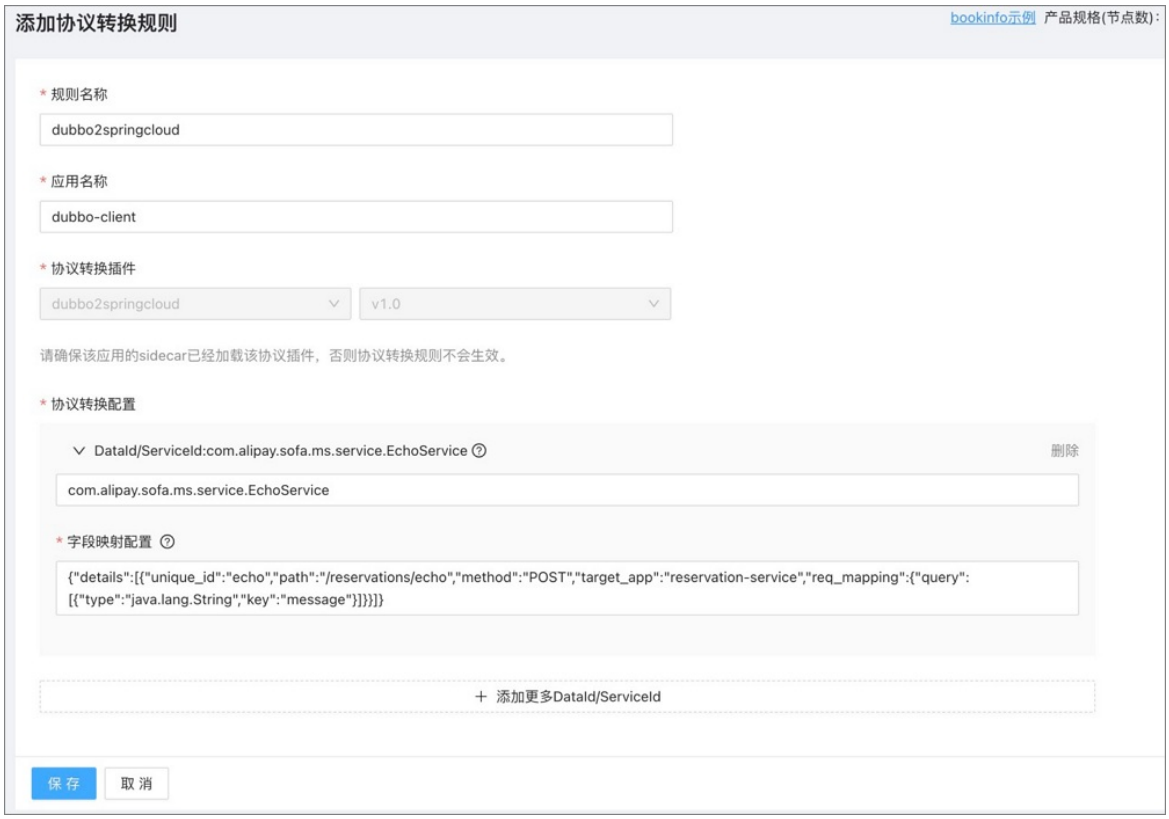
单击 **新建版本**，然后配置以下参数：



| 参数 | 内容 |
| --- | --- |
| 插件版本号 | v1.0 |
| 插件主体 | 上传编译好的插件。 |
| 备注 | 测试版本。 |

3. 新建协议转换配置。

   单击 **配置协议转换规则**，然后配置以下参数：




| 参数 | 内容 |
| --- | --- |
| 规则名称 | dubbo2springcloud。 |
| 应用名称 | dubbo-client。 |
| 协议转换插件 | dubbo2springcloud & v1.0<br>提前上传好协议转换插件，关联即可。 |

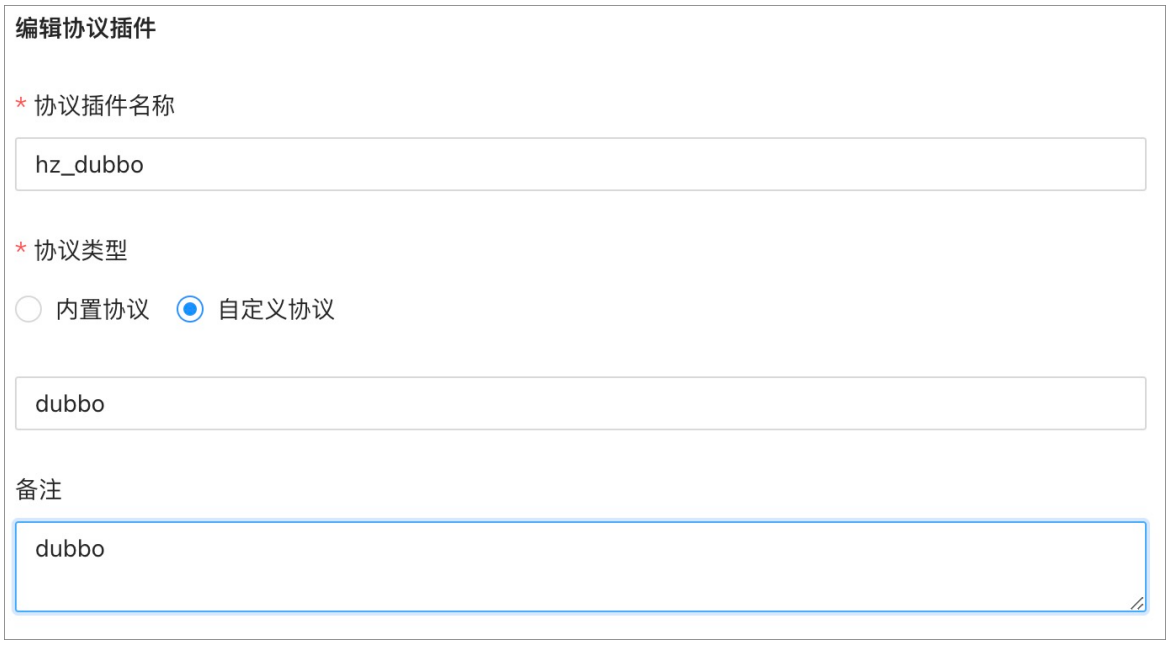
| | |
|---|---|
| DataId | com.alipay.sofa.ms.service.EchoService |
| 字段映射配置 | {"details": [{"unique_id":"echo","path":"/reservations/echo","method":"POST","target_app":"reservation-service","req_mapping":{"query": [{"type":"java.lang.String","key":"message"}]}}]} |

# 协议插件

## dubbo 协议插件

1. 新建协议插件。

   单击 **新建协议插件**，然后配置以下参数：




| 参数 | 内容 |
|---|---|
| 协议插件名称 | hz_dubbo |
| 协议类型 | 自定义协议。 |
| 备注 | dubbo |

2. 新增版本。

单击目标插件右侧的 **新增版本**，然后配置以下参数：

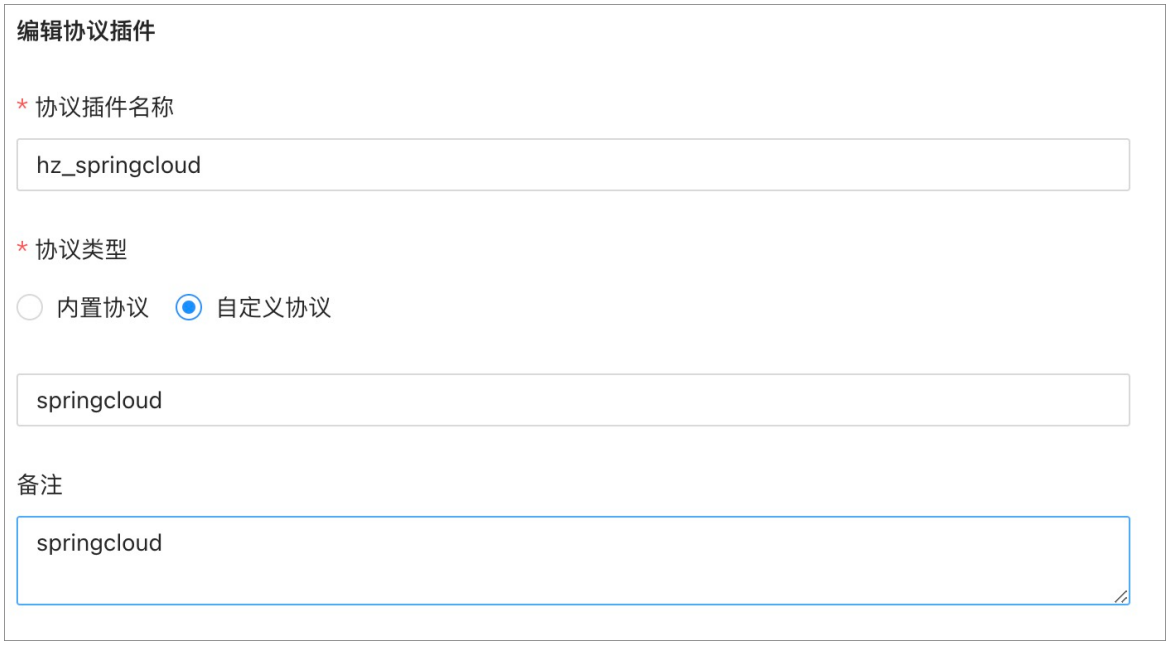
| 参数 | 内容 |
| --- | --- |

| | |
|---|---|
| 版本号 | v1.0。 |
| 插件主体 | 上传打包好的协议插件。 |
| 备注 | dubbo 测试。 |
| 关联功能插件 | 关联日志插件：logger-v1.0<br>关联协议转换插件：dubbo2spirngcloud-v1.0 |

## springcloud 协议插件

1. 新建协议插件。

   单击 **新建协议插件**，然后配置以下参数：




| 参数 | 内容 |
|---|---|
| 协议插件名称 | hz_springcloud。 |
| 协议类型 | 自定义协议。 |
| 备注 | springcloud。 |

2. 新增版本。

   在协议插件列表，单击目标插件右侧的 **新增版本**，然后配置以下参数：

| 参数 | 内容 |
| --- | --- |

| | |
|---|---|
| 版本号 | v1.0。 |
| 插件主体 | 上传打包好的协议插件。 |
| 备注 | springcloud 测试。 |
| 关联功能插件 | 关联日志插件：logger-v1.0。 |

## sidecar 注入

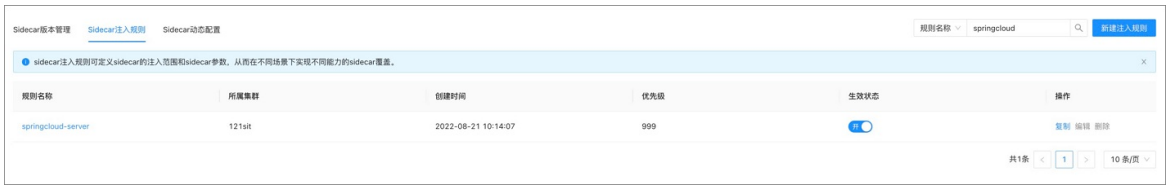
1. 为 springcloud 新建 sidecar 注入规则。
   i. 在左侧导航栏选择 服务网格> Sidecar 管理，然后单击 **Sidecar 注入规则**。
   ii. 单击 **新建注入规则**，然后配置以下参数：

| 参数 | 内容 |
|---|---|
| 规则名称 | springcloud-server。 |
| 所属集群 | 全局生效。 |
| 优先级 | 999。 |
| Sidecar 版本 | 对应版本即可。 |
| acvip地址 | 单击 **自动填充**。 |
| 协议插件设置 | hz_springcloud*v1.0。 |

其他参数说明，请参见 Sidecar 注入规则。

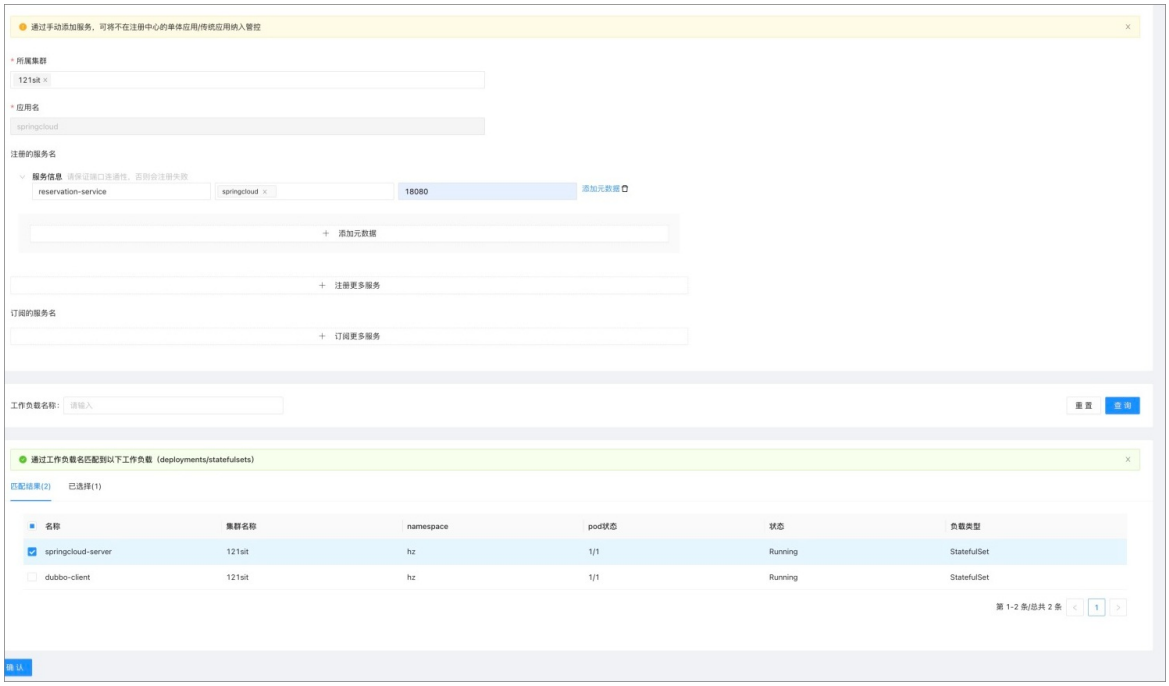
2. springcloud sidecar 注入规则开启。

在 Sidecar 注入规则列表，将刚刚创建的规则的生效状态改为 **开启**。




3. springcloud 服务声明。

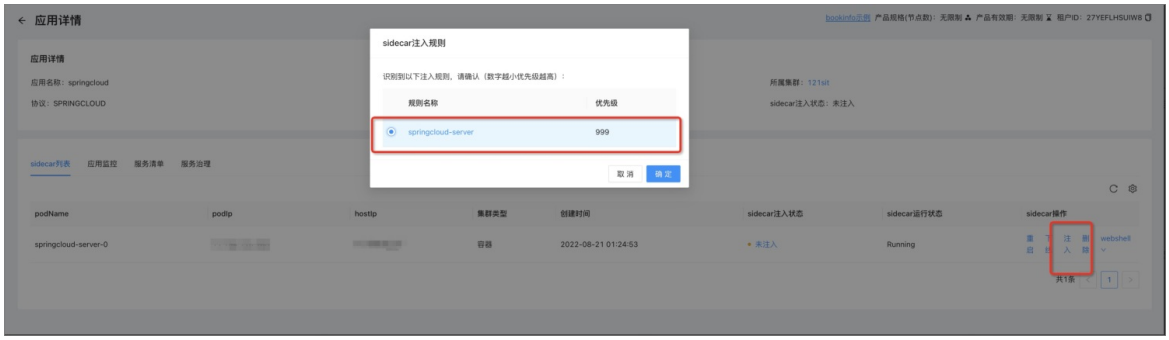
选择 **服务网格>服务目录>服务列表**，单击**手动添加应用**，然后配置以下参数：




| 参数 | 内容 |
| --- | --- |
| 所属集群 | K8s 的集群。 |
| 应用名 | springcloud。 |
| 注册的服务名 | reservation-service & springcloud & 18080。 |
| 工作负载名称 | springcloud-server。 |

4. springcloud 服务 sidecar 注入。

选择 **服务网格>服务目录>服务列表**，单击 **dubbo**，然后单击 **注入**。

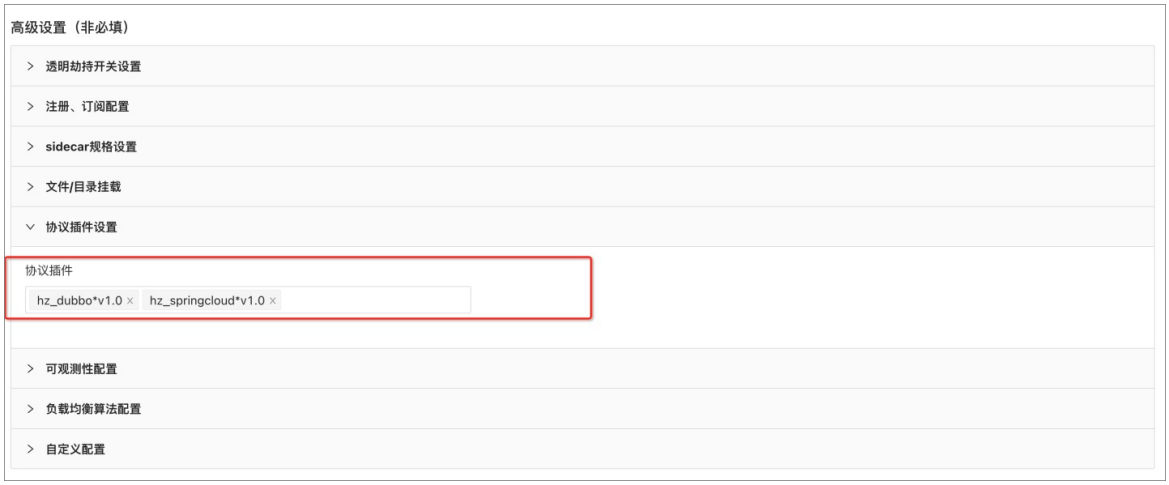



5. 查看 springcloud 服务 sidecar 注入状态。

   单击 **刷新** 查看注入状态。

6. dubbo sidecar 新建 sidecar 注入规则。

   新建注入规则，然后配置以下参数，其中：




| 参数 | 内容 |
| --- | --- |
| 规则名称 | dubbo-client。 |
| 所属集群 | 全局生效。 |
| 优先级 | 999。 |
| sidecar 版本 | 对应版本即可。 |

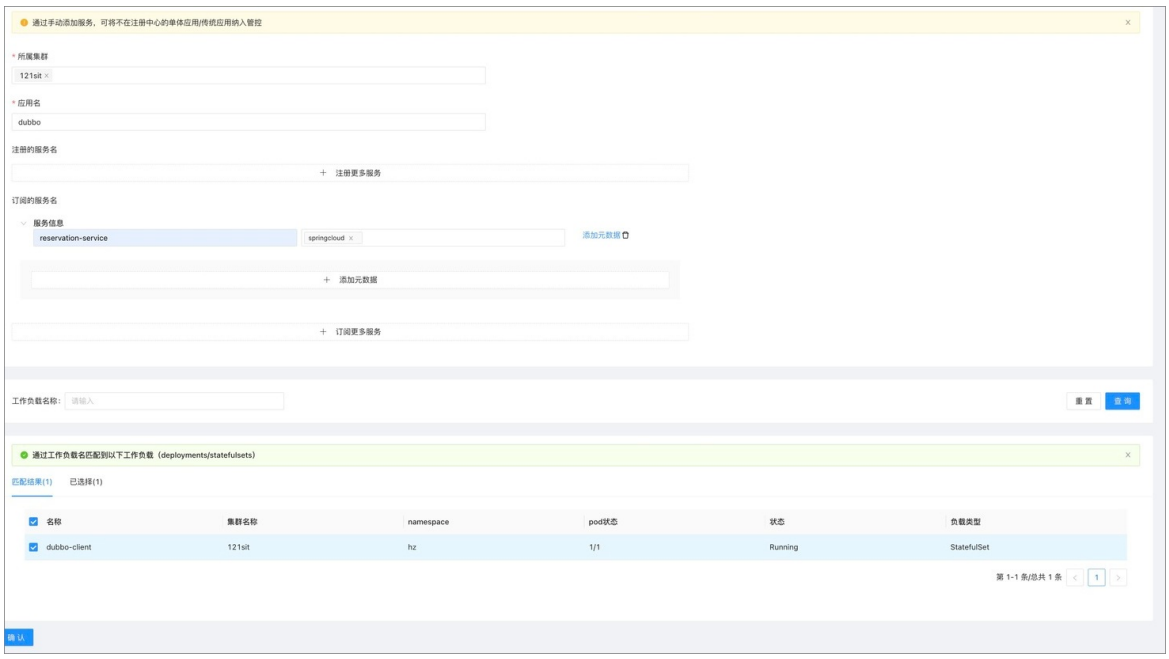
| | |
|---|---|
| acvip地址 | 单击 **自动填充**。 |
| 协议插件设置 | hz-dubbo v1.0、hz_springcloud v1.0。 |

其他参数说明，请参见 Sidecar 注入规则。

7. 将 dubbo sidecar 注入规则开启。

8. dubbo 服务订阅声明。

选择 **服务网格>服务目录>服务列表**，单击**手动添加应用**，然后配置以下参数：

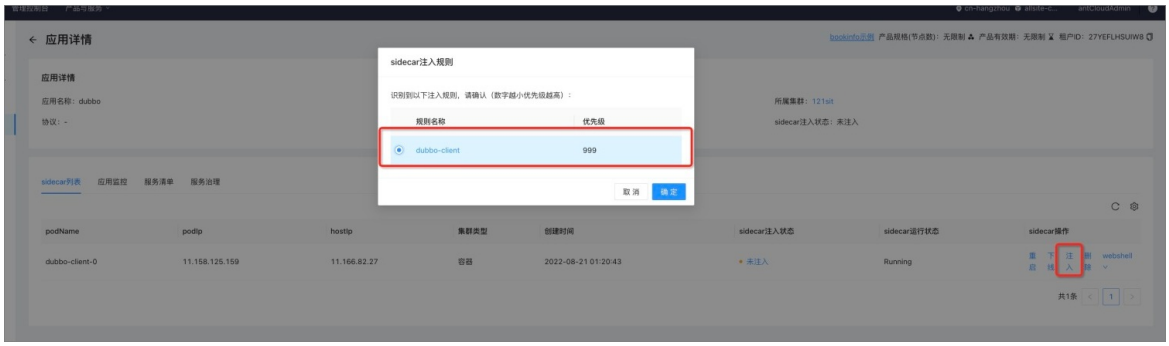



| 参数 | 内容 |
|---|---|
| 所属集群 | K8s 的集群。 |
| 应用名 | dubbo。 |

| | |
|---|---|
| 服务信息 | reservation-service & springcloud。 |
| 工作负载名称 | dubbo-client。 |

9. dubbo 服务 sidecar 注入。

   选择**服务网格>服务目录>服务列表**，单击 **dubbo**，单击 **注入**。




10. 查看 dubbo 服务 sidecar 注入状态。

    单击刷新图标。

## 服务验证

1. 选择 **服务网格>插件管理>协议转换**，单击 **开启**，配置将下发到应用的服务节点。

2. dubbo 发送请求。

```
curl 'localhost:10999/meshtest/dubbo/hello?message=hello'
```

3. dubbo 发送请求完成：

```
[root@dubbo-client-0 /home/admin]
#curl  'localhost:10999/meshtest/dubbo/hello?message=helloo'
hello
```

4. 查看 dubbo 服务访问日志。

```
[root@dubbo-client-0 /home/admin/logs/mosn]
#tail -f mosn_client.log
2022-08-22 00:47:09.919||{,"kind":"client","status":"200","callerProtocol":"dubbo","listenerAddress":"[redacted]","targetAddress":"[redacted]","callerAddress"
:"[redacted]","duration":"6.063158ms","startTime":"2022-08-22 00:47:09.913","mosnDuration":"1.595461ms","[redacted]":"5","app_name":"
dubbo-client"}
2022-08-22 00:47:14.566||{,"kind":"client","status":"200","callerProtocol":"dubbo","listenerAddress":"[redacted]","targetAddress":"[redacted]","callerAddress"
:"[redacted]","duration":"1.834749ms","startTime":"2022-08-22 00:47:14.564","mosnDuration":"772.736μs","mosnRequestDuration":"0","mosnReponseDuration":"1","app_name":"d
ubbo-client"}
```

5. springcloud 收到请求，查看访问日志。

```
#tail -f mosn_server.log
2022-08-22 00:47:09.919||{,"kind":"server","status":"200","callerProtocol":"springcloud","listenerAddress":"[redacted]","targetAddress":"[redacted]","callerAddress
":"[redacted]","duration":"1.337114ms","startTime":"2022-08-22 00:47:09.918","mosnDuration":"861.594μs","mosnRequestDuration":"0","mosnReponseDuration":"0","app_name":"
springcloud"}
2022-08-22 00:47:14.566||{,"kind":"server","status":"200","callerProtocol":"springcloud","listenerAddress":"[redacted]","targetAddress":"[redacted]","callerAddress
":"[redacted]","duration":"886.52μs","startTime":"2022-08-22 00:47:14.565","mosnDuration":"492.597μs","mosnRequestDuration":"0","mosnReponseDuration":"0","app_name":"sp
ringcloud"}
```

# 8.服务治理

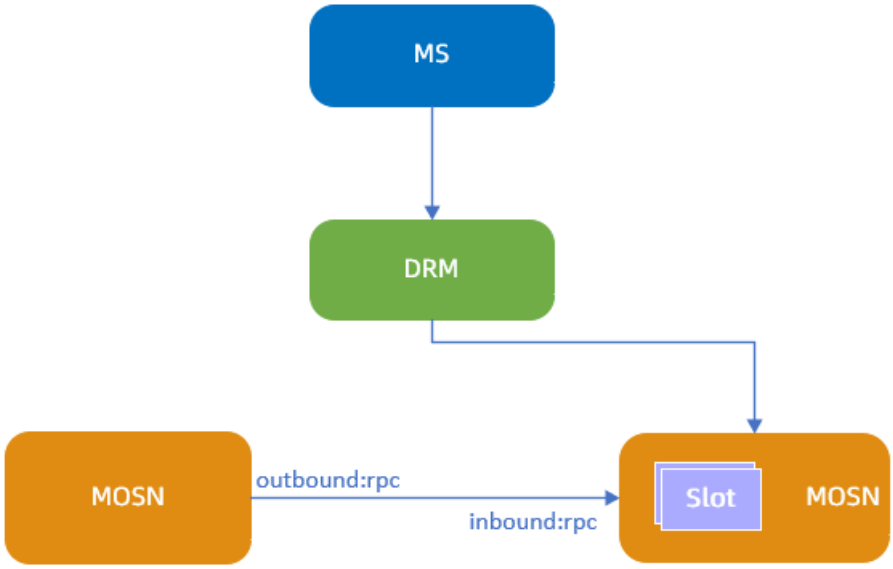
## 8.1. 服务限流

### 8.1.1. 管理服务限流规则

服务限流是对服务调用的速率或并发进行限制。这是在资源有限的情况下，分布式系统应对突发流量的一种常用的自我保护和保护上下游系统的保障可用性手段。SOFAMesh 当前提供了单机限流的功能。

#### 限流原理

服务限流生效流程如下：




1. 服务限流通过微服务管控台 MS 下发限流配置到分布式配置中心 DRM。
2. DRM 将限流的动态配置下发到 MOSN。

   限流规则在客户端生效。
3. 客户端应用进行访问时，MOSN 会监控应用流量，如果触发限流规则，应用流量即被限制。

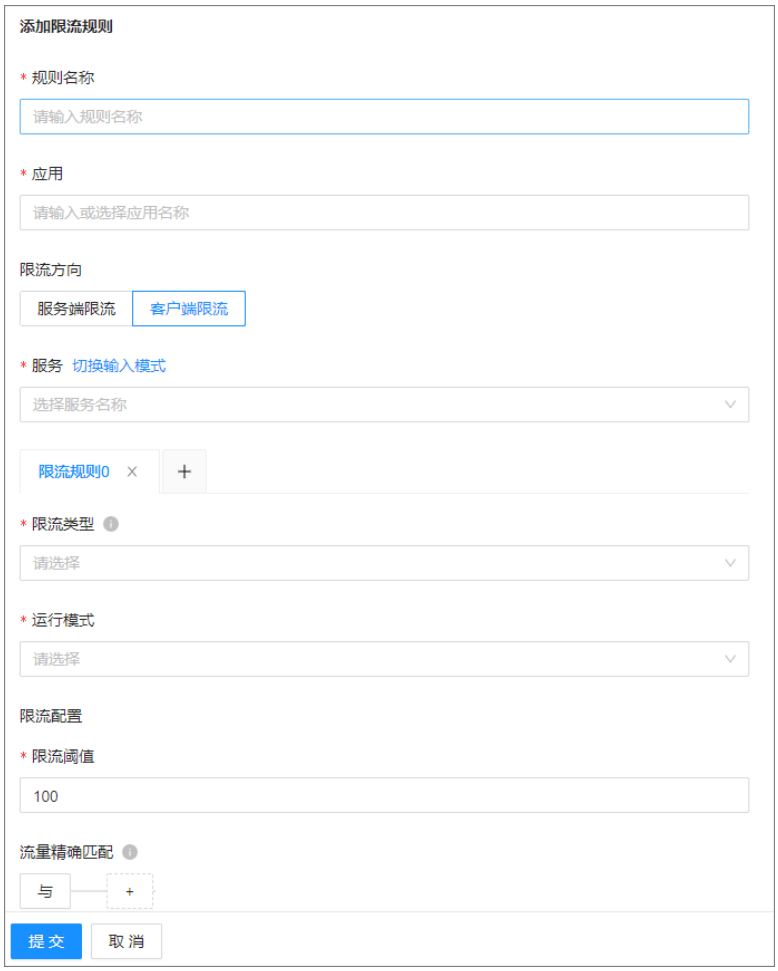
#### 注意事项

- 您可以添加多条限流规则，若存在服务名和匹配条件相同，但限速不同的限流规则时，生效限速更小的限流规则。
- 请根据您的实际环境合理配置限速规则，不合理的限速规则可能导致应用访问异常。

#### 添加限流规则

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格** > **服务治理**，然后单击 **服务限流** 页签。

3. 单击 **添加限流规则**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **规则名称** | 配置服务限流的规则名称。 |
| **应用** | 配置应用名称。星号（*）表示所有应用。<br>需填写客户端的应用 |
| **限流方向** | 目前仅支持客户端限流，规则在客户端生效。**服务** 填写客户端需要调用的服务；**方法** 填写客户端需要调用服务端服务的方法。 |
| **服务** | 配置为客户端的服务。单击 **切换输入模式** 可在手动填写与下拉选择之间切换。 |
| **方法** | 配置限流的方法。星号（*）表示所有方法。<br>需填写客户端的方法。 |

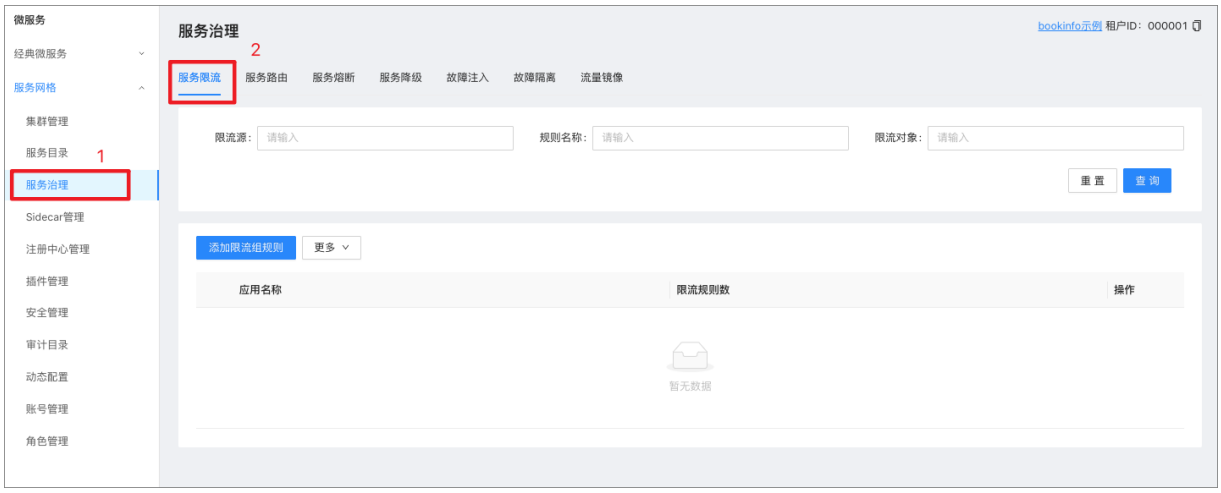
| | |
|---|---|
| **限流类型** | 选择限流的类型，可选值为：<br>○ **QPS**：单位时间内允许的最大请求数。<br>○ **最大并发数**：同一时间内允许的最大访问线程数。 |
| **运行模式** | 设置限流规则的运行方式，可选值为：<br>○ **拦截模式**：限流规则生效。<br>○ **观察者模式**：限流规则不生效，仅在 MOSN 中打印限流日志。 |
| **限流算法** | 使用令牌桶（Token Bucket）算法进行限流。仅在 **限流类型** 为 **QPS** 时设置。<br>系统会以一个恒定的速度往桶里放入令牌，而如果请求需要被处理，则需要先从桶里获取一个令牌，当桶里没有令牌可取时，则拒绝服务。详情请参见 令牌桶算法。 |
| **令牌桶系数** | 设置令牌桶的系数，以决定令牌桶的最大值。默认为 1。<br>令牌数最大值 =（限流阈值 ÷ 单位时间）× 令牌桶系数 |
| **单位时间 ms** | 设置限速规则的单位时间，单位为 ms。仅在 **限流类型** 为 **QPS** 时设置。 |
| **限流阈值** | 根据选择的限流类型设置限流阈值：<br>○ **限流类型** 为 **QPS** 时：表示单位时间内允许通过最大请求数的值，超过该数值的请求会被限流。<br>○ **限流类型** 为 **最大并发数** 时：表示同一时间内允许连接的最大访问线程数，超过该阈值的线程会被限流。 |
| **流量精确匹配** | 设置流量的匹配条件，满足匹配条件的流量才会使用限流规则。置空此项时表示匹配所有流量。<br>您可以配置多条匹配条件，多个条件是与的关系，按顺序进行匹配。参数配置如下：<br>○ **字段**：可选择系统字段和请求头。<br>○ **字段名**：根据字段类型有不同的值。<br>■ 系统字段：包括流量类型、调用方应用名、调用方 IP、服务方应用名。<br>■ 请求头：请求头是指协议的请求头，比如 Dubbo 协议取的是 attachment，HTTP 协议取的是 Request Header。用户可以在应用系统中自定义请求头参数和值。<br>○ **选择逻辑**：包括等于、不等于、属于、不属于、正则。<br>○ **字段值**：字段名对应的值。 |

4. 单击 **提交**，然后单击 **确定**。

5. 在限流规则列表中，将刚刚创建的限流规则的状态改为 **开**。

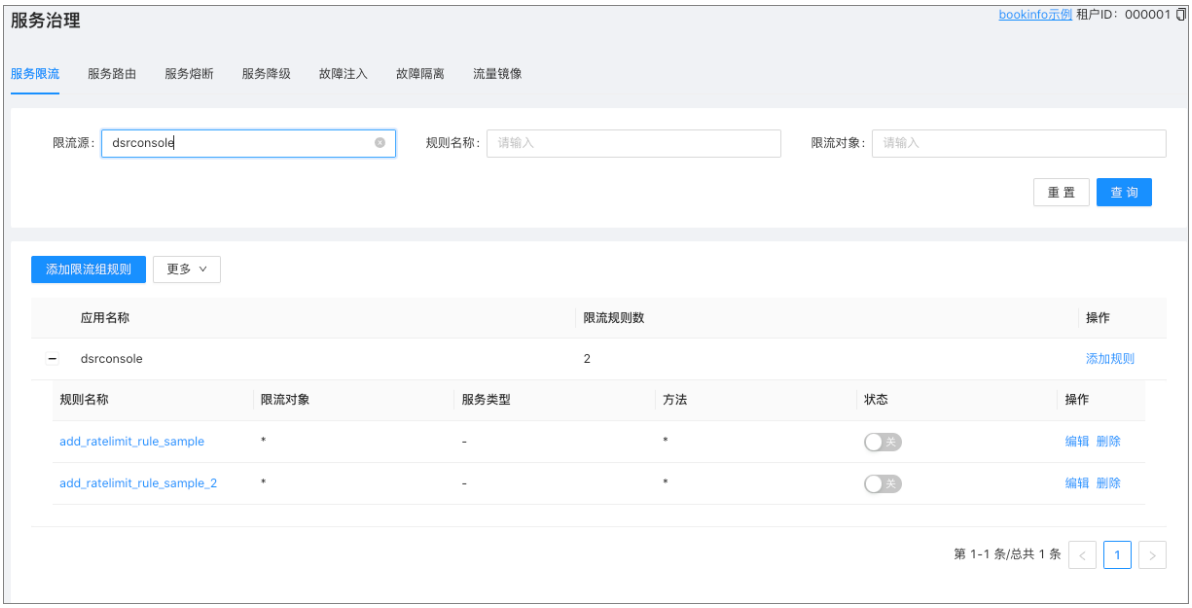
## 搜索限流规则

服务限流规则支持从三个维度进行模糊搜索：



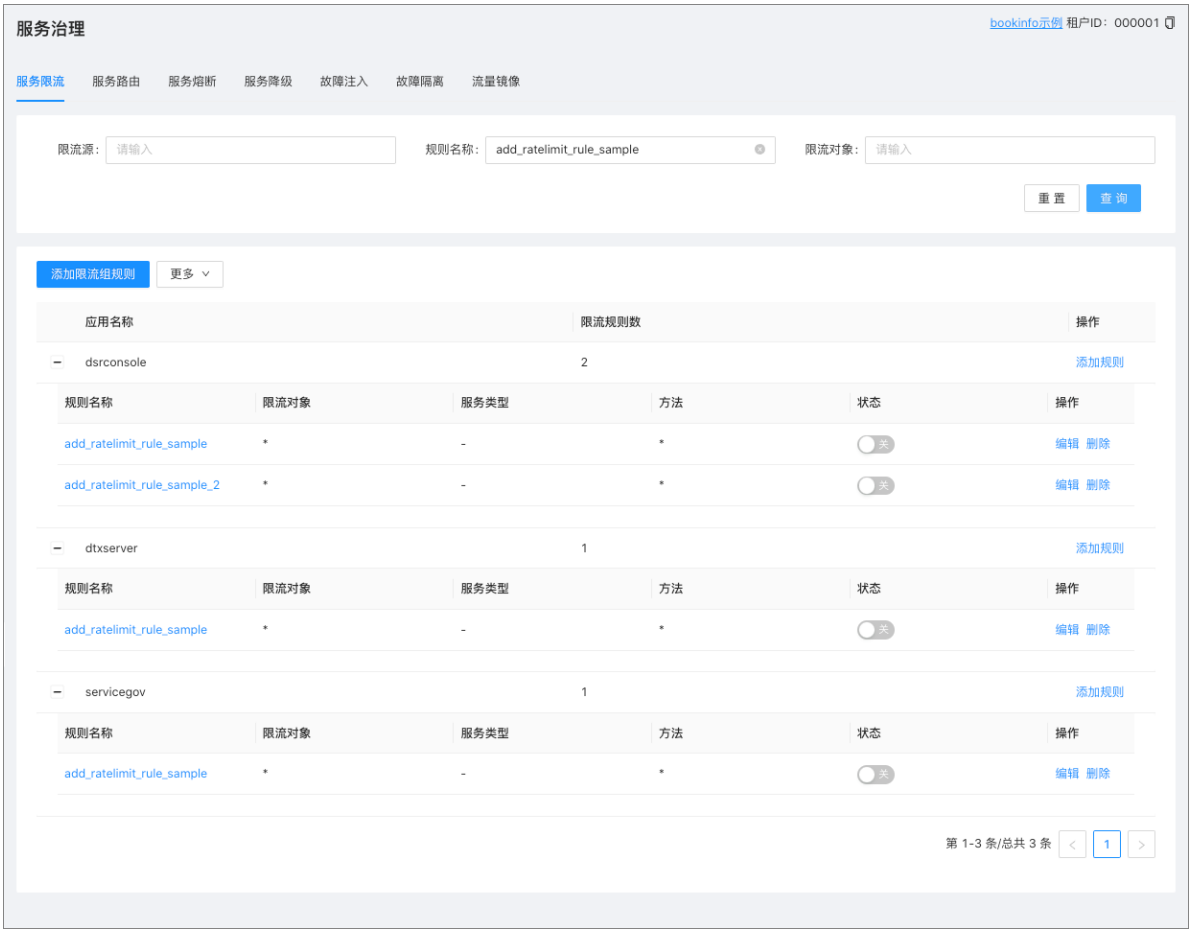

- 限流源，即应用名称

  例如：查询 dsrconsole 应用下的所有规则。



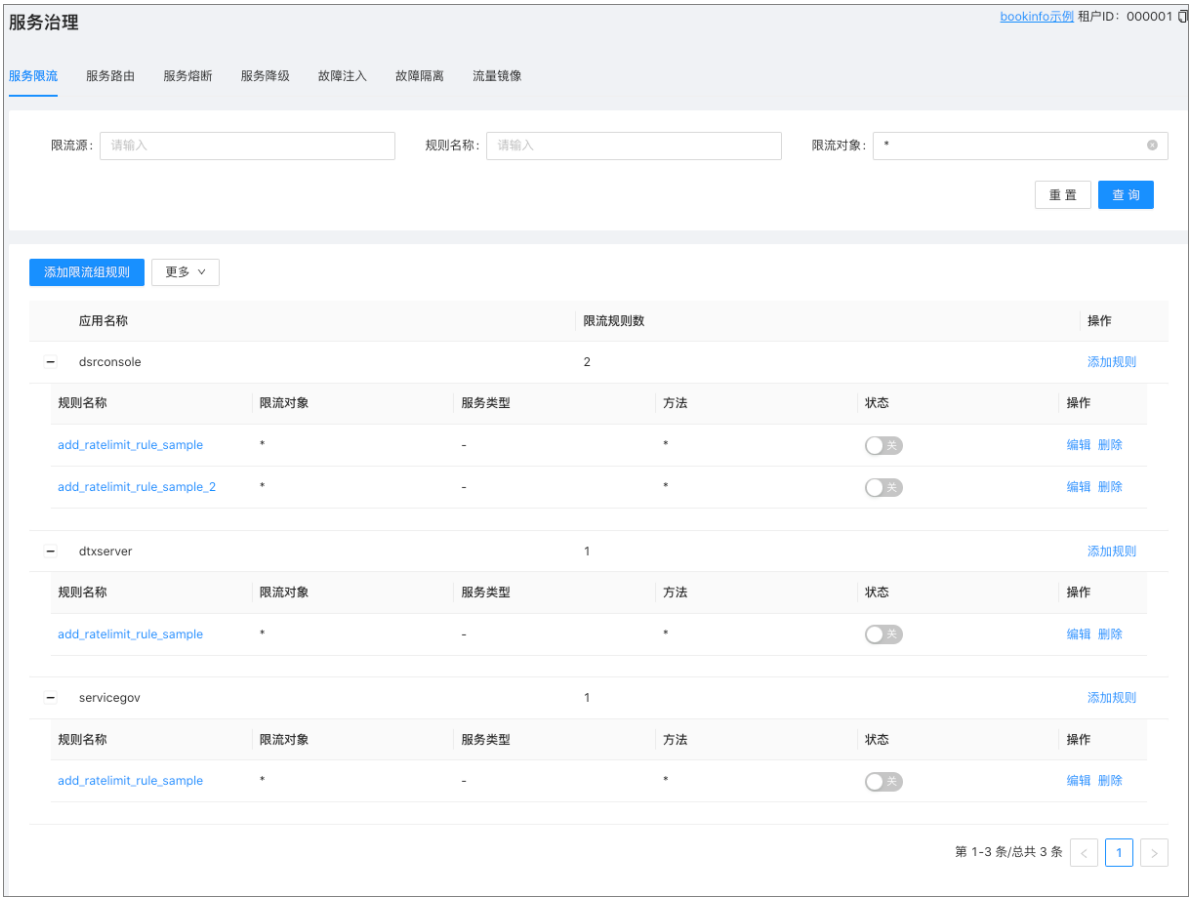

- 规则名称

  例如：查询规则名称为 `add_ratelimit_rule_sample` 的规则。




- 限流对象，即服务名称

  例如：查询限流对象为 * 的规则。




限流规则从应用的维度分组展示（或叫 限流组规则）

## 编辑限流规则

您可以随时编辑已创建的限流规则，编辑规则会在提交后实时生效。

1. 在 **服务限流** 页签，单击目标限流规则右侧的 **编辑**。
2. 按需求编辑限流规则后，单击 **提交**。

## 导出限流规则

您可以将已配置的限流规则导出，以备份您的限流规则。

1. 在 **服务限流** 页签，选择 **更多 > 导出**。
2. 选中目标规则，然后单击 **导出**。

## 导入鉴权规则

您可以导入备份的鉴权规则，以快速创建鉴权规则。

1. 在 **服务鉴权** 页签，选择 **更多 > 导入**。
2. 单击 **选择文件**，然后选中目标文件后，单击 **确定**。

## 删除限流规则

您可以删除已创建的限流规则，删除规则的操作会实时生效，请谨慎操作。

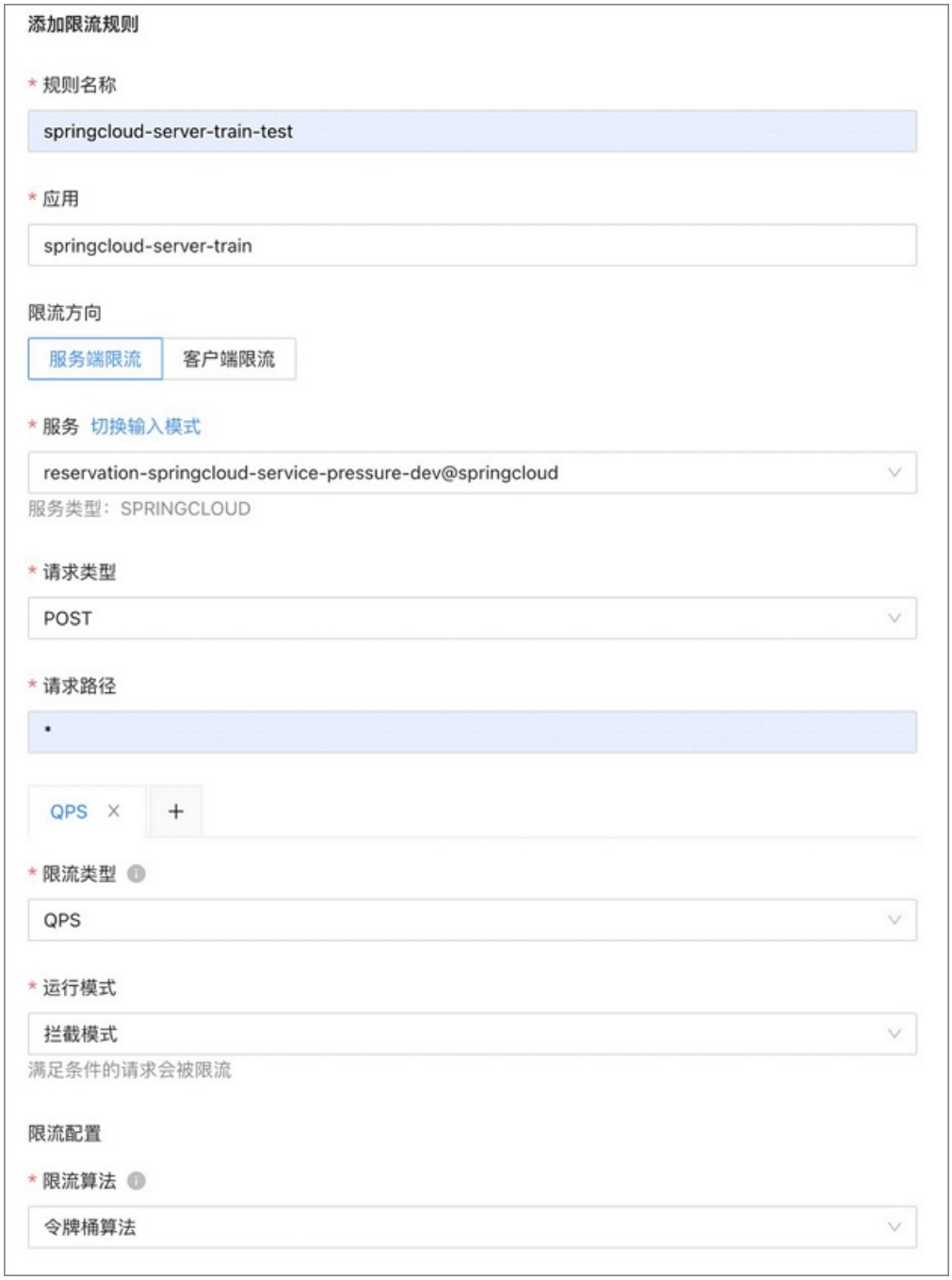
1. 在 **服务限流** 页签，单击目标限流规则右侧的 **编辑**。
2. 单击 **确定**。

# 8.1.2. 验证服务限流规则

本文以标准 SpringCloud 为例，演示限流后的服务请求状态。

## 服务端限流

限流配置如下：




**测试限流效果**

进入客户端的 MOSN 容器，访问客户端暴露的接口：

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问 8084 端口的 hi 接口
curl 0:8084/reservations/hi?count=10
```

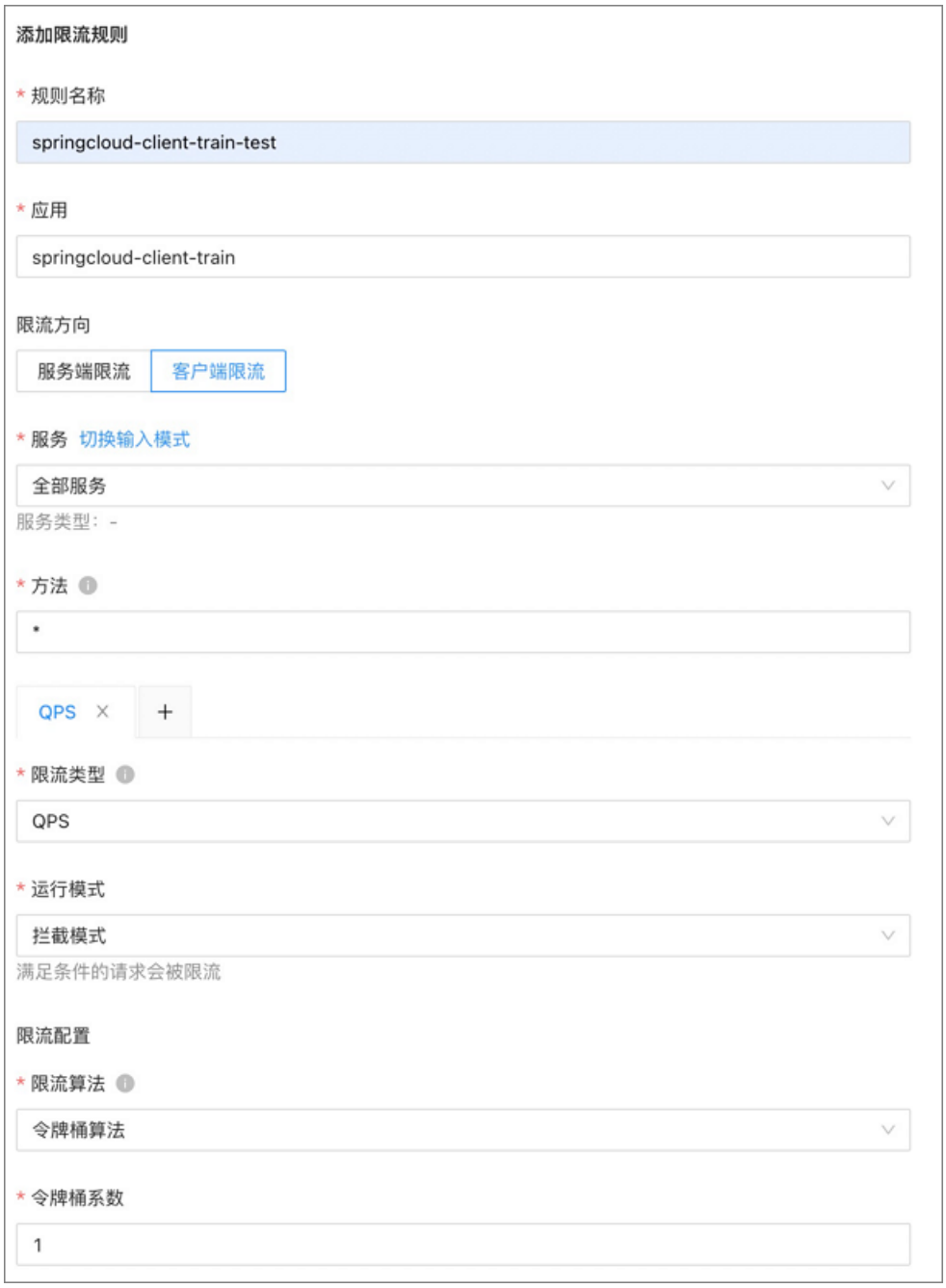
可以看到返回结果中，有请求被限流了：

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 301 毫秒</br>总平均耗时 : 30.1 毫秒</br>单个请求平均耗时 : 28.1 毫秒</br>QPS : 33 </br><h5>Thread-26</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 128 毫秒, 结果: Hi %f\K!! 我是[10.0.200.97]<br>
第 1次调用, sleep 0 毫秒, 耗时 133 毫秒, 结果: Hi %f\K!! 我是[10.0.200.38]<br>
第 2次调用, sleep 0 毫秒, 耗时 9 毫秒, 结果: Hi %f\K!! 我是[10.0.200.97]<br>
第 3次调用, sleep 0 毫秒, 耗时 11 毫秒, 结果: Hi %f\K!! 我是[10.0.200.38]<br>
第 4次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
mosn limit exception<br>
第 5次调用, sleep 0 毫秒, 耗时 4 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
mosn limit exception<br>
第 6次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 7次调用, sleep 0 毫秒, 耗时 4 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 8次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 9次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
</div>
```

这里能调用通 4 次，是因为服务端有两台服务器，限流方向为服务端限流时，即在服务端侧进行统计，每台服务端限流 2 笔交易，总计 4 笔。

## 客户端限流

限流配置如下：




**测试限流效果**

进入客户端的 MOSN 容器，访问客户端暴露的接口：

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问8084端口的 hi 接口
curl 0:8084/reservations/hi?count=10
```

可以看到返回结果中，只有两笔请求能够正常调用：

```
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 39 毫秒</br>总平均耗时 : 3.9 毫秒</br>单个请求平均耗时 : 1.9 毫秒</br>QPS : 256 </br><h5>Thread-40</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 9 毫秒, 结果: Hi ***'4! 我是[10.0.200.97]<br>
第 1次调用, sleep 0 毫秒, 耗时 10 毫秒, 结果: Hi ***'4! 我是[10.0.200.38]<br>
第 2次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 3次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 4次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 5次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 6次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 7次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 8次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 9次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
</div>
```

## 最大并发数限流

限流配置如下：

**测试限流效果**

发起访问请求：

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问8084端口的 hi 接口
curl 0:8084/reservations/hi?count=10\&concurrent=3
```

**说明**

`concurrent` 参数表示线程数量。

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10\&concurrent=3
总请求数 : 30</br>总响应耗时 : 68 毫秒</br>总平均耗时 : 2.2666666666666666 毫秒</br>单个请求平均耗时 : 6.233333333333333 毫秒</br>QPS : 441 </br><h5>Thread-
<div>
第 0次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 1次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 2次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 3次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 4次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 5次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 6次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 7次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 8次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 9次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
</div>
<h5>Thread-43</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 9 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 1次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 2次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 3次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 4次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 5次调用, sleep 0 毫秒, 耗时 9 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 6次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 7次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 8次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 9次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
</div>
<h5>Thread-45</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 1次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 2次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 3次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 4次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 5次调用, sleep 0 毫秒, 耗时 4 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 6次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 7次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
第 8次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.38]<br>
第 9次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi Y y0=V00G! 我是[10.0.200.97]<br>
</div>
```

MOSN 分辨不出来源请求具体是属于哪一个线程，所以无法做到两个线程里的请求全部通过，第三个线程里的请求全部拒绝，只能是一个估算的值。要想精确限流多少个请求，请使用 QPS 限流。

# 8.2. 服务路由

## 8.2.1. 管理服务路由规则

当服务消费者面临多个服务提供者时，需要通过路由规则来确定具体的服务提供者。服务路由功能提供了灵活的路由功能，允许您定义多条服务路由规则，可以帮助您解决多个场景下的难题。

### 功能简介

服务路由多用于线下测试连调、蓝绿发布、灰度引流场景，将因版本升级造成的问题影响降到最低。

- 线下测试联调

  线下测试时，可能会缺少相应环境。您可以将测试应用注册到线上，然后开启路由规则，在本地进行测试。

- 蓝绿发布

  蓝绿发布时，您可以通过路由规则将流量切换到绿组，以测试新版本应用是否正常。

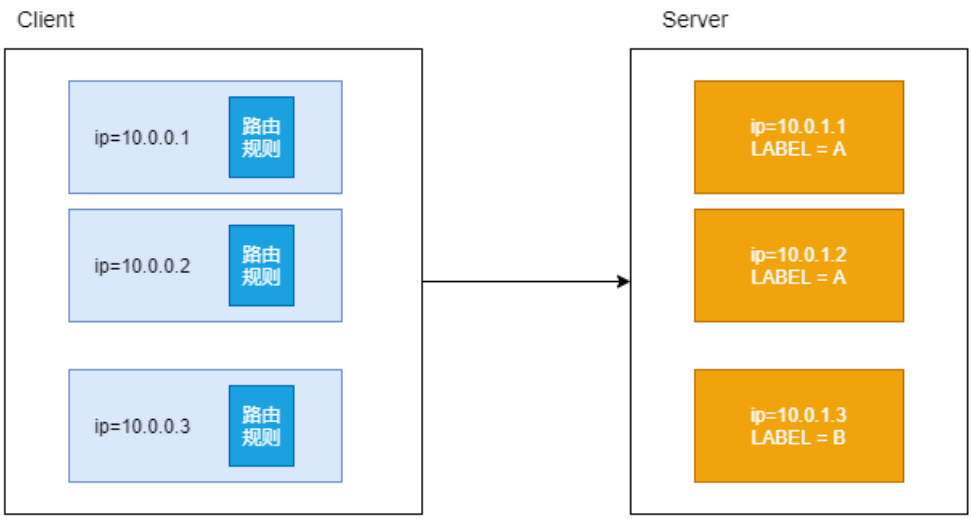
- 灰度引流

  灰度发布时，您可以通过路由规则将流量引导到灰度发布的应用版本上，测试灰度发布的版本是否正常。

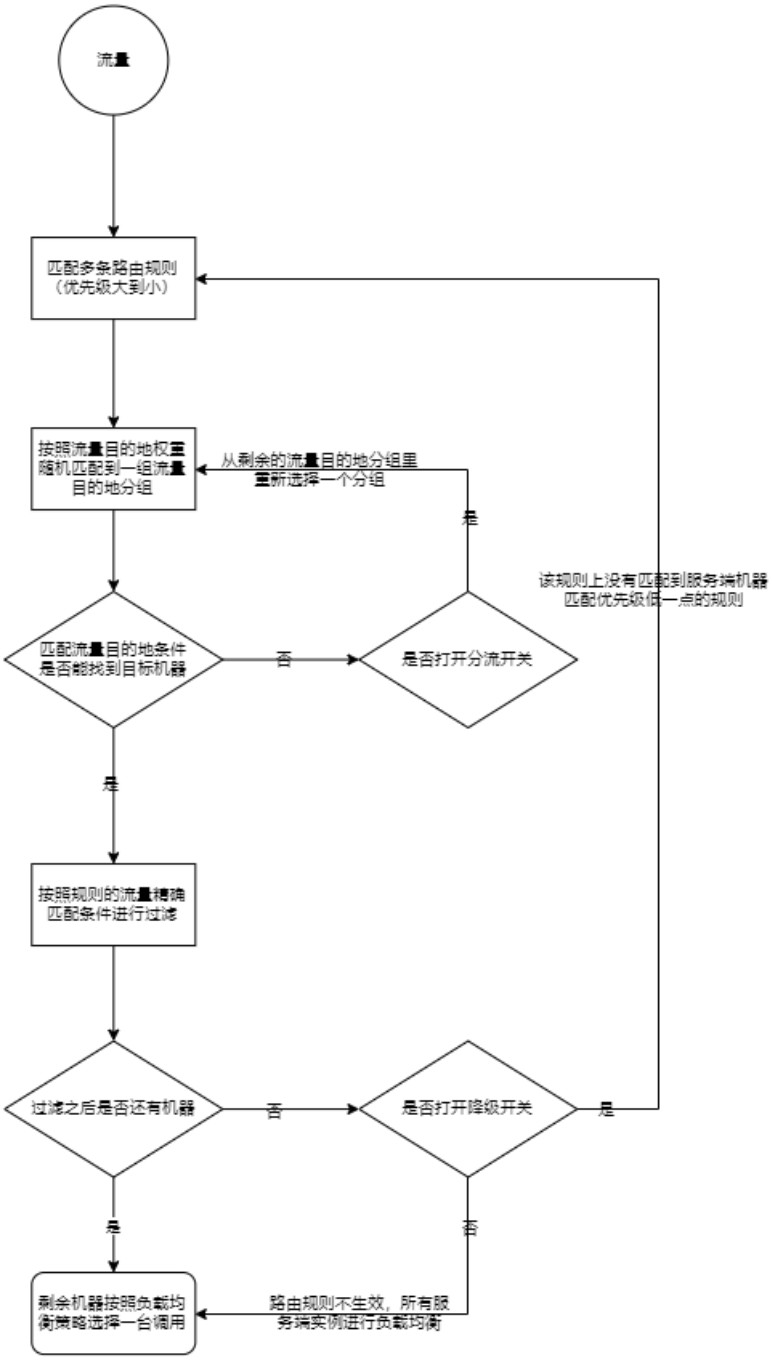
> **说明**
>
> 当前版本支持 SOFA、Spring Cloud 和 Dubbo 服务的路由。

配置的路由规则实际在服务消费方（即客户端）生效，您可以配置服务路由过滤服务提供方。目前主要有 RPC-Client（经典微服务生效）和 MOSN（服务网格生效）两种执行实体。

在部署服务提供方时，您可以在机器上加固定格式的标签，比如 LABEL=A。



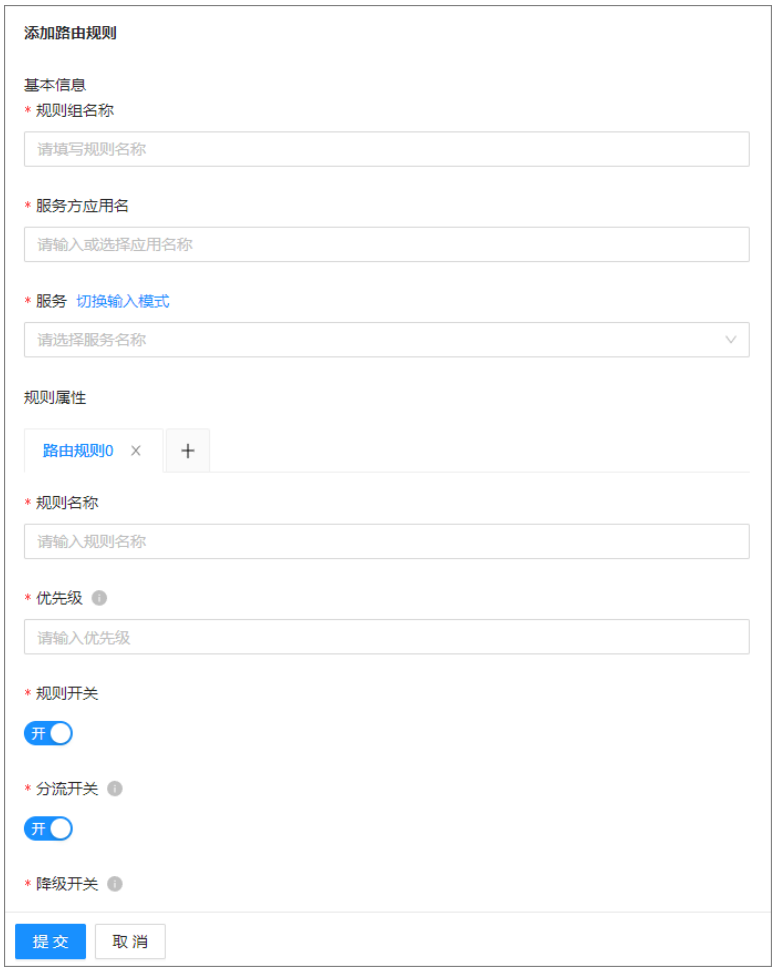

路由规则生效流程如下图所示：




## 添加路由规则

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格 > 服务治理**，然后单击 **服务路由** 页签。

3. 单击 **添加服务路由**，然后配置以下参数：




| 区域 | 参数 | 说明 |
| --- | --- | --- |
| **基本信息** | **规则组名称** | 配置路由规则组名称。 |
| | 服务方应用名 | 配置服务端的应用名称。 |
| | **服务** | 配置应用包含的服务。<br>单击 **切换输入模式** 可在手动填写与下拉选择之间切换。 |
| | **规则名称** | 配置路由规则名称。 |
| | **优先级** | 配置路由的优先级，数字越大优先级越高。<br>同时存在多条路由时，按照路由优先级大小进行匹配。 |

| | | |
|---|---|---|
| 规则属性 | 规则开关 | 配置是否开启该条路由规则。默认开启。 |
| | 分流开关 | 当流量目的地分组没有服务实例时，是否转发到其他分组。<br>○ 开启后，流量在高权重分组中未匹配到服务实例，会继续去匹配其他分组。<br>○ 关闭后，流量不会匹配其他分组。 |
| | 降级开关 | 当整条规则都匹配不到服务实例时，是否继续匹配其他规则。<br>○ 开启后，流量未在当前规则中匹配到服务实例时，会继续去匹配优先级更低的路由规则。<br>○ 关闭后，流量不会匹配其他路由规则。 |
| | 流量精确匹配（可选） | 通过流量匹配条件来过滤需要执行路由规则的请求，满足这些条件的流量才会使用这条规则，不满足这条规则的就走降级开关逻辑，往低优先级的路由规则去匹配。不填表示匹配所有流量。<br>多个条件按照顺序执行，直到被某一个规则被拦截或全部通过。规则主要包括下述内容：<br>○ **字段**：包括系统字段、请求头。<br>○ **字段名**：根据字段类型有不同的值。<br>■ 系统字段：包括流量类型、调用方应用名、调用方 IP、服务方应用名。<br>■ 请求头：请求头是指协议的请求头，比如 Dubbo 协议取的是 attachment，HTTP 协议取的是 Request Header。用户可以在应用系统中自定义请求头参数和值。<br>○ **选择逻辑**：包括等于、不等于、属于、不属于、正则。<br>○ **字段值**：字段名对应的值。 |
| | 权重 | 设置当前流量目的地分组的权重，多个流量目的地的权重之和为 100%。<br>您可以单击最下方的 **添加** 按钮，配置多个流量目的地分组。 |

| | | |
|---|---|---|
| **流量目的地** | **服务实例分组条件** | 配置服务实例分组条件，多个条件之间是与的关系。<br>○ 字段名：配置服务实例属性。可从下拉列表选择，或者自定义输入。<br>您还可以配置自定义 label，配置示例见下表的 label 配置示例。<br>○ 逻辑：等于。<br>○ 字段值：配置字段名对应的值。<br>字段名是您自定义的 label 时，此处填写 label 对应的 value。<br>您可以单击下方的 **添加** 按钮，配置多个分组条件。 |

label 配置示例：

| label 分类 | 容器发布 | | 虚拟机发布 | |
|---|---|---|---|---|
| | 示例 | 说明 | 示例 | 说明 |
| 服务路由 label | app.custom.label.test:"aaa"<br>app.custom.label.app:"bbb" | 需要在 yaml 上进行配置，key 的前缀是 `app.custom.label` 。 | APP_CUSTOM_LABELS:'{"app.custom.label.test":"aaa","app.custom.label.app":"bbb"}' | env 环境变量的 `APP_CUSTOM_LABELS` 字段。 |
| | APP_CUSTOM_LABELS:'{"app.custom.label.test":"aaa","app.custom.label.app":"bbb"}' | 如果 yaml 上没有 label 配置，会获取 env 环境变量的 `APP_CUSTOM_LABELS` 字段。 | | |
| 分组路由 label | app.group.labels: "aaa" | 需要在 yaml 上进行配置，key 是 `app.group.labels` 。 | APP.GROUP.LABELS="aaa" | env 环境变量的 `APP_GROUP_LABELS` 字段。 |

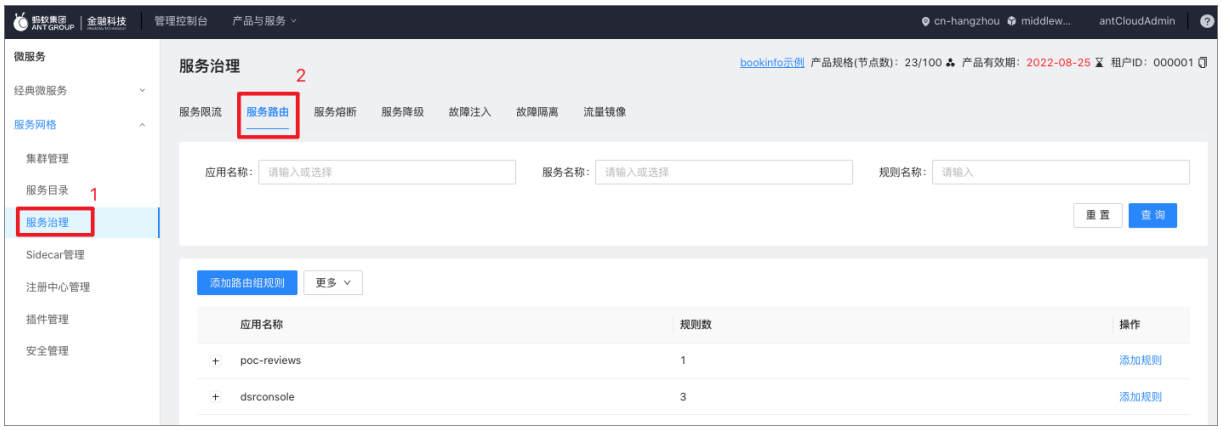
| | | | | |
|---|---|---|---|---|
| | APP.GROUP.LABELS="aaa" | 如果 yaml 上没有 label 配置，会获取 env 环境变量的 `APP_GROUP_LABELS` 字段。 | | |
| 集群路由 label | cafe.sofastack.io/cluster=mesh1 | 集群路由容器发布方式。 | - | - |

4. 单击 **提交**。
5. 在服务路由列表中，将刚创建的路由规则的状态改为 **开启**。

> **重要**
>
> 您可以添加多条路由规则，相同应用名称的路由规则会被合并。

## 搜索服务路由

服务路由规则支持从三个维度进行模糊搜索：




- 应用名称
- 服务名称
- 规则名称

## 编辑路由规则

您可以随时编辑已创建的路由规则，规则提交后实时生效。

1. 在 **服务路由** 页签，单击目标应用左侧的加号（+）。
2. 单击目标路由规则右侧的 **编辑**。
3. 按需求编辑路由规则后，单击 **提交**。

## 删除路由规则

您可以删除已创建的路由规则，删除规则会实时生效，请谨慎操作。

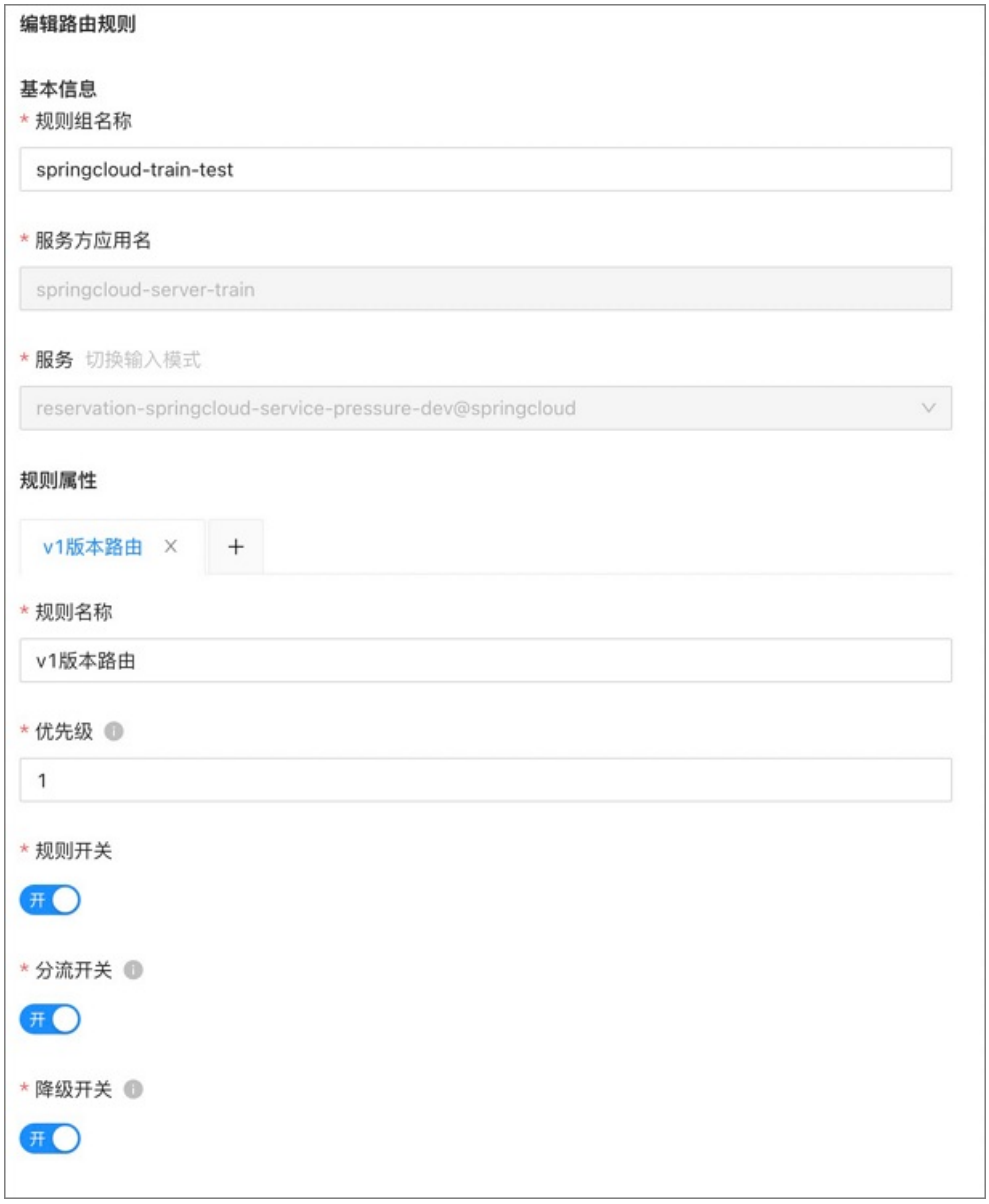
1. 在 **服务路由** 页签，单击目标应用左侧的加号（+）。
2. 单击目标路由规则右侧的 **删除**。
3. 单击 **确定**。

# 8.2.2. 验证服务路由规则

本文以标准 SpringCloud 为例演示配置路由规则后的服务请求状态。

## 版本路由

路由规则配置如下：



**规则验证**

进入容器，访问暴露的 hi 接口：

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问 8084 端口的 hi 接口
curl 0:8084/reservations/hi?count=10
```

未开启路由规则时，流量均匀调拨到 v1.container、v2.container 版本的服务上：

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 76 毫秒</br>总平均耗时 : 7.6 毫秒</br>单个请求平均耗时 : 7.2 毫秒</br>QPS : 131 </br><h5>Thread-83</h5>
<div>
第 0次调用，sleep 0 毫秒，耗时 7 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.38]<br>
第 1次调用，sleep 0 毫秒，耗时 8 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.97]<br>
第 2次调用，sleep 0 毫秒，耗时 7 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.38]<br>
第 3次调用，sleep 0 毫秒，耗时 7 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.97]<br>
第 4次调用，sleep 0 毫秒，耗时 7 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.38]<br>
第 5次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.97]<br>
第 6次调用，sleep 0 毫秒，耗时 7 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.38]<br>
第 7次调用，sleep 0 毫秒，耗时 11 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.97]<br>
第 8次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.38]<br>
第 9次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi ××5×Ù)××O! 我是[10.0.200.97]<br>
</div>
```

开启路由规则后，出所有流量均被路由到 v1.container 版本的服务器上：

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 57 毫秒</br>总平均耗时 : 5.7 毫秒</br>单个请求平均耗时 : 5.6 毫秒</br>QPS : 175 </br><h5>Thread-85</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 1次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 2次调用, sleep 0 毫秒, 耗时 5 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 3次调用, sleep 0 毫秒, 耗时 5 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 4次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 5次调用, sleep 0 毫秒, 耗时 5 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 6次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 7次调用, sleep 0 毫秒, 耗时 5 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 8次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
第 9次调用, sleep 0 毫秒, 耗时 5 毫秒, 结果: Hi ©©©!©©BY5! 我是[10.0.200.38]<br>
</div>
```

## 多流量目的地匹配

规则配置如下：

配置 20% 流量往 v2.container 版本路由，80% 流量往 v1.container 版本路由，调用后可以看到符合预期结果。

```
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 57 毫秒</br>总平均耗时 : 5.7 毫秒</br>单个请求平均耗时 : 5.7 毫秒</br>QPS : 175 </br><h5>Thread-87</h5>
<div>
第 0次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi  Brm@@! 我是[10.0.200.38]<br>
第 1次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi  Brm@@! 我是[10.0.200.38]<br>
第 2次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi  Brm@@! 我是[10.0.200.97]<br>
第 3次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi  Brm@@! 我是[10.0.200.38]<br>
第 4次调用，sleep 0 毫秒，耗时 5 毫秒，结果：Hi  Brm@@! 我是[10.0.200.38]<br>
第 5次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi  Brm@@! 我是[10.0.200.38]<br>
第 6次调用，sleep 0 毫秒，耗时 5 毫秒，结果：Hi  Brm@@! 我是[10.0.200.38]<br>
第 7次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi  Brm@@! 我是[10.0.200.38]<br>
第 8次调用，sleep 0 毫秒，耗时 5 毫秒，结果：Hi  Brm@@! 我是[10.0.200.97]<br>
第 9次调用，sleep 0 毫秒，耗时 6 毫秒，结果：Hi  Brm@@! 我是[10.0.200.38]<br>
</div>
```

# 8.3. 服务熔断

## 8.3.1. 管理服务熔断规则

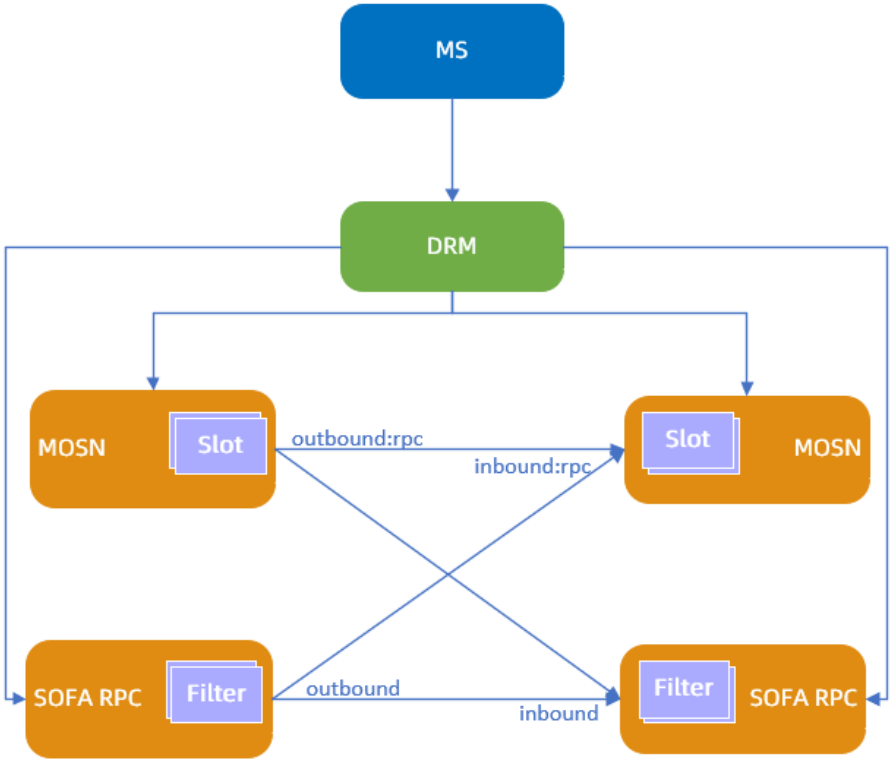
当服务中的服务端接口不稳定，出现频繁超时或错误时，可能会引起服务调用雪崩。您可以对应用开启服务熔断功能，使有故障的服务端及时返回错误，并释放系统资源，提高用户体验和系统性能。

### 功能简介

您可以通过下述操作让故障处于可控范围：

- 通过监控或者服务拓扑查看到某个服务延时较大、错误率较多后，进行服务治理。
- 选择异常服务后，进行自动熔断设置，可自定义熔断条件，例如：在某个时间段内（例如 10s）请求数达到某个值，且错误率或者延时达到某个值。

满足条件，则可以触发熔断。服务熔断后还可以持续测试该服务，进行自动熔断恢复。服务熔断的规则发布流程如下：

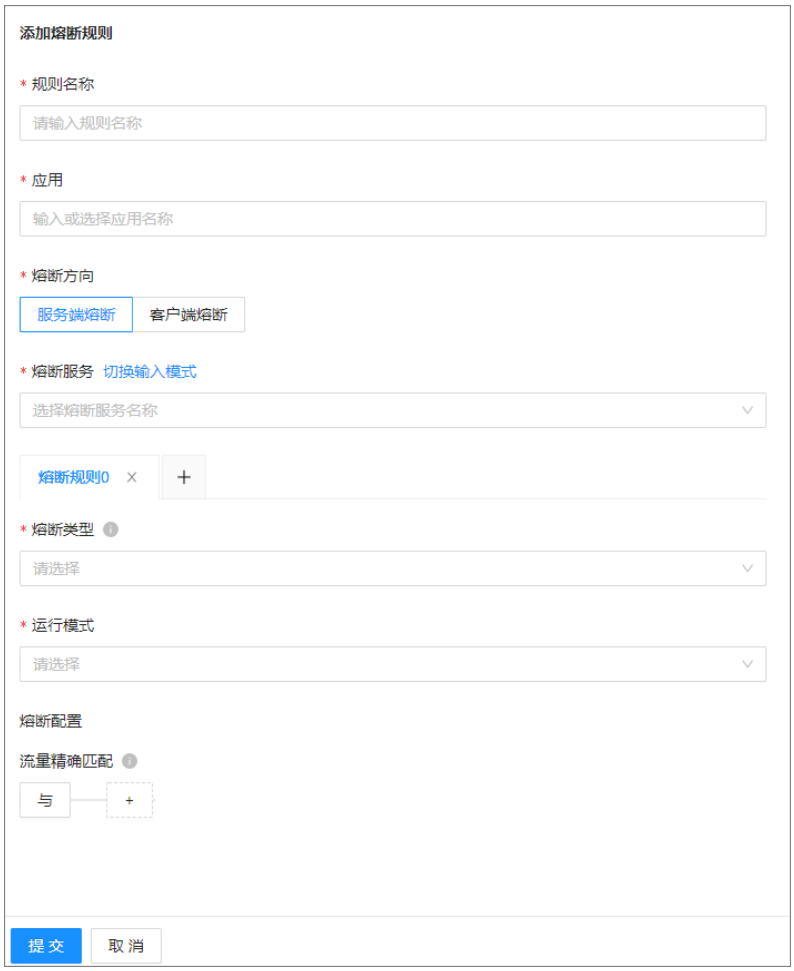


1. 通过微服务管控台 MS 下发熔断的动态配置到分布式配置中心 DRM。
2. DRM 将熔断的动态配置下发到 MOSN。
3. MOSN 根据平均 RT 和错误率触发熔断。

   熔断可在客户端或服务端生效。

## 添加熔断规则

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格** > **服务治理**，然后单击 **服务熔断** 页签。
3. 单击 **添加熔断组规则**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **规则名称** | 配置服务熔断规则的名称。<br>支持中文、英文、数字、下划线。 |
| **应用** | 配置检测的目标应用。<br>您可以手动填写或在下拉列表中选择。 |

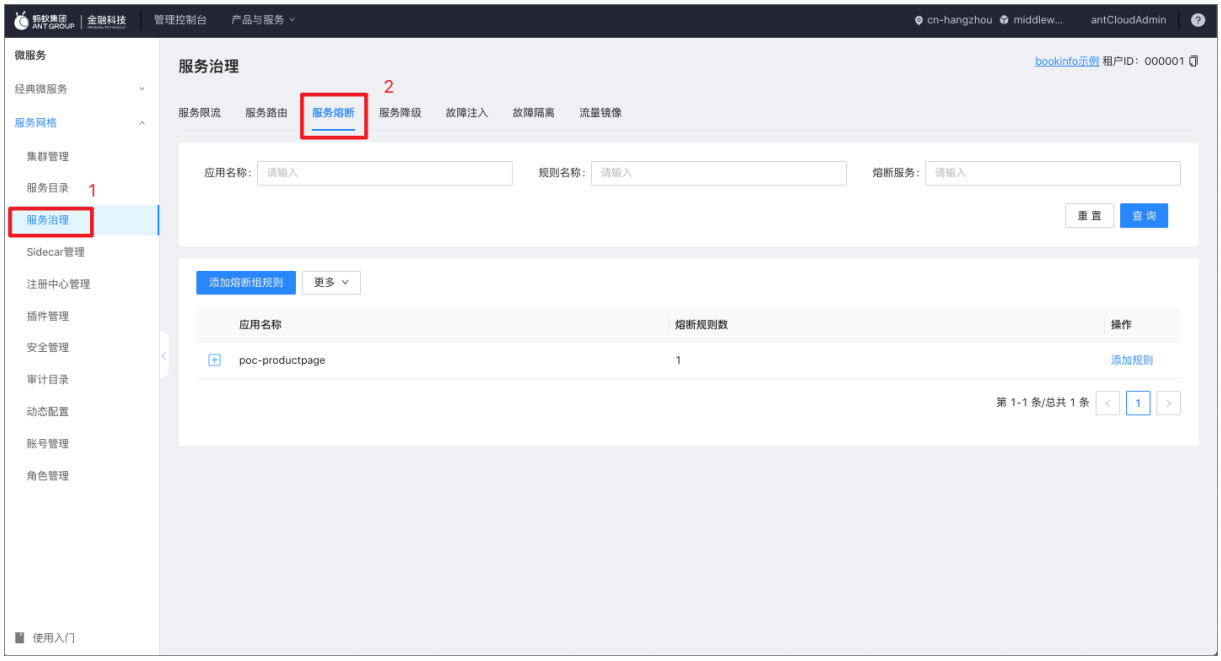
| 参数 | 说明 |
| --- | --- |
| **熔断方向** | 配置服务熔断的方向，取值如下：<br>○ **客户端熔断**：表示熔断规则在客户端生效。在客户端侧统计调用服务端的请求指标，应用填写客户端的应用，服务和方法填写引用的服务端的服务和方法。<br>○ **服务端熔断**：表示熔断规则在服务端生效，每个服务端单独统计请求指标，应用填写服务端的应用，服务和方法填写服务端的服务和方法。 |
| **熔断服务** | 配置熔断的服务。单击 **切换输入模式** 可在手动填写与下拉选择之间切换。 |
| **方法** | 配置需要熔断的服务的方法名。 |
| **熔断类型** | 配置熔断的类型，取值如下：<br>○ **错误比率**：熔断时间窗口内的请求错误率大于错误率阈值时，触发熔断。<br>○ **平均 RT**：熔断时间窗口内的请求数阈值大于平均 RT 阈值时，触发熔断。 |
| **运行模式** | 配置熔断的运行模式，取值如下：<br>○ **拦截模式**：满足条件的请求会触发熔断规则。<br>○ **观察者模式**：熔断规则不会被触发，只会在 MOSN 里打印日志。 |
| **熔断配置** | 配置触发熔断的各项参数：<br>○ **熔断指标窗口**：和请求数阈值搭配使用，表示熔断指标窗口内至少有请求数阈值的请求就会纳入熔断指标统计。单位为秒。<br>○ **请求数阈值**：和熔断指标窗口搭配使用。单位为个。<br>○ **错误率阈值**：请求数的平均错误率达到此阈值时触发熔断。熔断类型配置为 **错误比率** 时配置。<br>○ **平均 RT 阈值**：请求响应时间超过此阈值时触发熔断。单位为毫秒。熔断类型配置为 **平均 RT** 时配置。<br>○ **熔断时间段**：触发熔断后的熔断时长，熔断时长之内的请求会直接返回错误。当熔断超过该时间后会再次尝试调用服务端，如果还是失败则继续触发熔断。<br>○ **RPC 超时时间**：经典微服务场景会使用。单位为毫秒。熔断类型配置为 **错误比率** 时配置。 |

| 参数 | 说明 |
| --- | --- |
| **流量精确匹配**（可选） | 配置之后，只有满足匹配条件的流量才会使用这份熔断规则。置空表示匹配所有流量。<br>您可以配置多条匹配条件，多个条件是与的关系，按顺序进行匹配。参数配置如下：<br>○ **字段**：可选择系统字段和请求头。<br>○ **字段名**：根据字段类型有不同的值。<br>■ 系统字段：包括流量类型、调用方应用名、调用方 IP、服务方应用名。<br>■ 请求头：请求头是指协议的请求头，比如 Dubbo 协议取的是 attachment，HTTP 协议取的是 Request Header。用户可以在应用系统中自定义请求头参数和值。<br>○ **选择逻辑**：包括等于、不等于、属于、不属于、正则。<br>○ **字段值**：字段名对应的值。 |

4. 单击 **提交**，然后单击 **确定**。
5. 在熔断规则列表中，将刚刚创建的熔断规则的状态改为 **开**。

## 搜索熔断规则

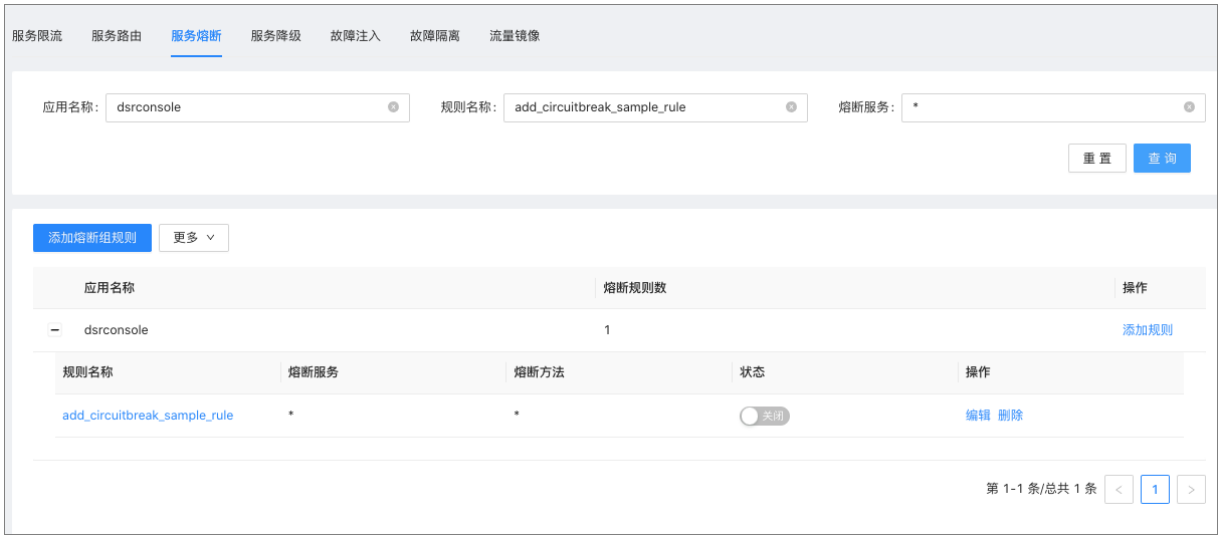
支持从三个维度进行模糊搜索：




- 应用名称
- 规则名称
- 熔断服务

熔断规则从应用的维度分组展示（或叫 熔断组规则）。

搜索示例：查询 dsrconsole 应用下规则名称为 `add_circuitbreak_sample_rule` 服务为 `*` 的规则列表。




## 编辑熔断规则

您可以随时编辑已创建的熔断规则，规则提交后会实时生效。

1. 在 **服务熔断** 页签，单击目标熔断规则右侧的 **编辑**。
2. 按需求编辑熔断规则后，单击 **提交**。

## 删除熔断规则

您可以删除已创建的熔断规则，删除操作会实时生效，请谨慎操作。

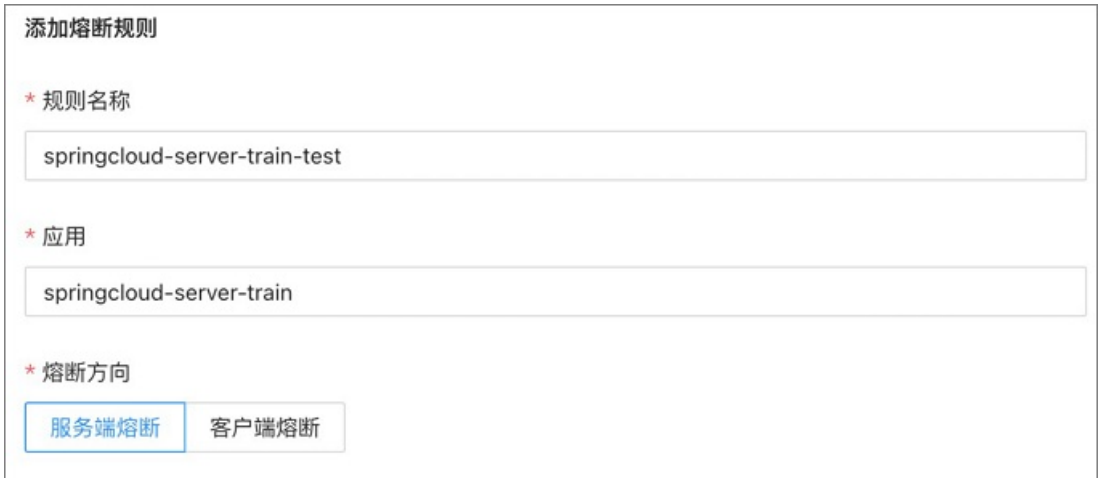
1. 在 **服务熔断** 页签，单击目标熔断规则右侧的 **删除**。
2. 单击 **确定**。

# 8.3.2. 验证服务熔断规则

本文以标准 SpringCloud 演示配置熔断规则后的请求状态。

## 错误比率熔断

熔断规则配置如下：



**验证熔断规则**

1. 在服务端侧设置接口的抛错或者超时。

```
kubectl exec -it springcloud-server-train-v1-0 -c mosn-sidecar-container bash

curl 0:8080/status/set?throwException=true\&sleep=0
```

2. 进入客户端容器，多次调用客户端接口。

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问8084端口的 hi 接口
curl 0:8084/reservations/hi?count=10
```

调用 springcloud-server-train-v1 的请求都异常了，返回 500 错误：

```
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 76 毫秒</br>总平均耗时 : 7.6 毫秒</br>单个请求平均耗时 : 3.4 毫秒</br>QPS : 131 </br><h5>Thread-15</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 9 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T09:54:29.256+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = G0\u00070\u0001k\u00070z0","path":"/reservations/hi2"}<br>
第 1次调用, sleep 0 毫秒, 耗时 9 毫秒, 结果: Hi G00k0z0! 我是[10.0.200.97]<br>
第 2次调用, sleep 0 毫秒, 耗时 9 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T09:54:29.274+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = G0\u00070\u0001k\u00070z0","path":"/reservations/hi2"}<br>
第 3次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi G00k0z0! 我是[10.0.200.97]<br>
第 4次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T09:54:29.288+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = G0\u00070\u0001k\u00070z0","path":"/reservations/hi2"}<br>
第 5次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi G00k0z0! 我是[10.0.200.97]<br>
第 6次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T09:54:29.302+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = G0\u00070\u0001k\u00070z0","path":"/reservations/hi2"}<br>
第 7次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi G00k0z0! 我是[10.0.200.97]<br>
第 8次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T09:54:29.317+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = G0\u00070\u0001k\u00070z0","path":"/reservations/hi2"}<br>
第 9次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi G00k0z0! 我是[10.0.200.97]<br>
</div>
```

3. 开启熔断规则。

   开启熔断规则后，发现异常服务返回两次异常（500 错误）后便发生了熔断，并返回 509 错误（这个 509 是 MOSN 返回的，请求并没有到异常的服务）。

```
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 65 毫秒</br>总平均耗时 : 6.5 毫秒</br>单个请求平均耗时 : 3.6 毫秒</br>QPS : 153 </br><h5>Thread-19</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 10 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T10:39:12.645+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = 00500h0\t\u0015\u001A","path":"/reservations/hi2"}<br>
第 1次调用, sleep 0 毫秒, 耗时 9 毫秒, 结果: Hi 00500h0 ! 我是[10.0.200.97]<br>
第 2次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T10:39:12.663+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = 00500h0\t\u0015\u001A","path":"/reservations/hi2"}<br>
第 3次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi 00500h0 ! 我是[10.0.200.97]<br>
第 4次调用, sleep 0 毫秒, 耗时 4 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 5次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi 00500h0 ! 我是[10.0.200.97]<br>
第 6次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 7次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi 00500h0 ! 我是[10.0.200.97]<br>
第 8次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 9次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi 00500h0 ! 我是[10.0.200.97]<br>
</div>
```

## 平均 RT 熔断

熔断规则配置如下：

编辑熔断规则
* 规则名称
springcloud-server-train-test
* 应用
springcloud-server-train
* 熔断方向
服务端熔断
客户端熔断
* 熔断服务 切换输入模式
reservation-springcloud-service-pressure-dev@springcloud
服务类型：SPRINGCLOUD
* 请求类型
POST
* 请求路径
*
平均 RT
+
* 熔断类型
平均 RT
* 运行模式
拦截模式
满足条件的请求会被熔断
熔断配置
* 熔断指标窗口 (s)
10
* 请求数阈值
2
* 平均 RT 阈值（ms）
200
* 熔断时间段（ms）
10000
流量精确匹配
与
+

**验证熔断规则**

1. 在服务端侧设置接口的抛错或者超时。

```
kubectl exec -it springcloud-server-train-v1-0 -c mosn-sidecar-container bash


curl localhost:8080/status/set?throwException=false\&sleep=500
```

2. 进入客户端容器，多次调用客户端接口。

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问8084端口的 hi 接口
curl 0:8084/reservations/hi?count=10
```

调用 springcloud-server-train-v1 的请求都有 500 毫秒的超时：

```
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 2587 毫秒</br>总平均耗时 : 258.7 毫秒</br>单个请求平均耗时 : 258.7 毫秒</br>QPS : 3 </br><h5>Thread-22</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 510 毫秒 结果: Hi 0?????gl! 我是[10.0.200.38]<br>
第 1次调用, sleep 0 毫秒, 耗时 11 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.97]<br>
第 2次调用, sleep 0 毫秒, 耗时 510 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.38]<br>
第 3次调用, sleep 0 毫秒, 耗时 10 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.97]<br>
第 4次调用, sleep 0 毫秒, 耗时 508 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.38]<br>
第 5次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.97]<br>
第 6次调用, sleep 0 毫秒, 耗时 507 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.38]<br>
第 7次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.97]<br>
第 8次调用, sleep 0 毫秒, 耗时 507 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.38]<br>
第 9次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi 0?????gl! 我是[10.0.200.97]<br>
</div>
```

3. 开启熔断规则。

开启熔断规则后，发现耗时请求调用两次后便发生了熔断，并返回 509 错误（这个 509 是 MOSN 返回的，请求并没有到耗时的服务）。

```
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 1060 毫秒</br>总平均耗时 : 106.0 毫秒</br>单个请求平均耗时 : 105.2 毫秒</br>QPS : 9 </br><h5>Thread-26</h5>
<div>
第 0次调用, sleep 0 毫秒, 耗时 508 毫秒, 结果: Hi ??/=??v?D! 我是[10.0.200.38]<br>
第 1次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi ??/=??v?D! 我是[10.0.200.97]<br>
第 2次调用, sleep 0 毫秒, 耗时 508 毫秒, 结果: Hi ??/=??v?D! 我是[10.0.200.38]<br>
第 3次调用, sleep 0 毫秒, 耗时 8 毫秒, 结果: Hi ??/=??v?D! 我是[10.0.200.97]<br>
第 4次调用, sleep 0 毫秒, 耗时 3 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 5次调用, sleep 0 毫秒, 耗时 6 毫秒, 结果: Hi ??/=??v?D! 我是[10.0.200.97]<br>
第 6次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 7次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi ??/=??v?D! 我是[10.0.200.97]<br>
第 8次调用, sleep 0 毫秒, 耗时 2 毫秒, 结果: status 509 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 9次调用, sleep 0 毫秒, 耗时 7 毫秒, 结果: Hi ??/=??v?D! 我是[10.0.200.97]<br>
</div>
```

# 8.4. 服务降级

## 8.4.1. 管理服务降级规则

服务降级是在请求量和服务器压力急剧增大，服务端节点压力随之增大，出现响应延迟或者错误的情况下，根据实际业务情况，将某些不重要的服务进行降级处理，释放服务器资源以保证核心业务的正常运行。

### 添加服务降级规则

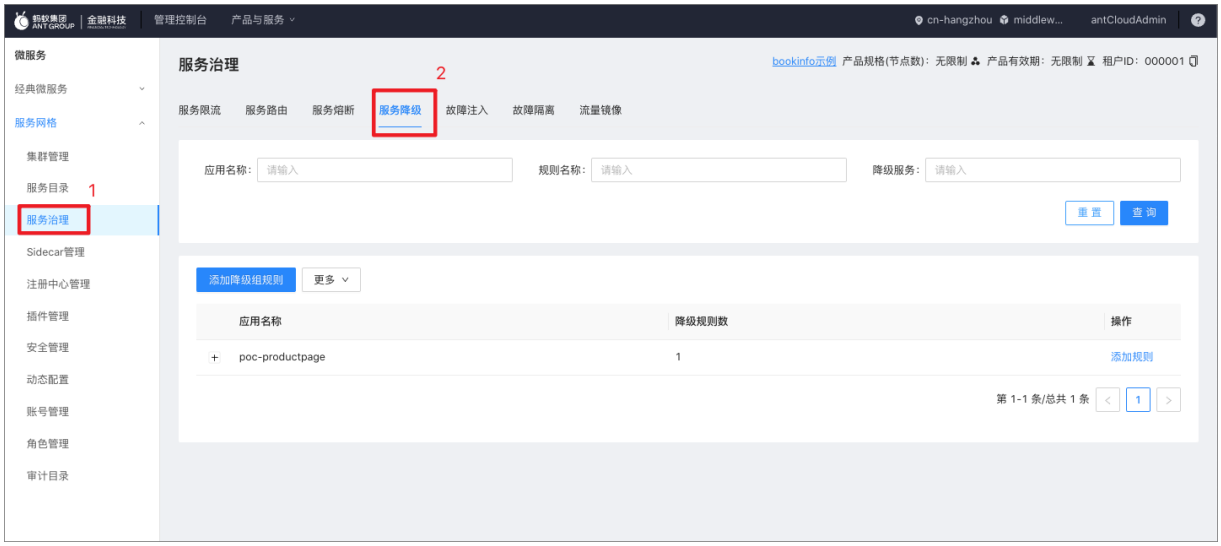
1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格** > **服务治理**，然后单击 **服务降级** 页签。
3. 单击 **添加降级组规则**，然后配置以下参数：

| 配置项 | 说明 |
| --- | --- |
| 规则名称 | 配置降级规则的名称。 |
| 调用方应用 | 降级在客户端生效，这里填写或选择客户端应用。 |
| 运行模式 | 设置降级规则的运行模式，取值如下：<br>○ 拦截模式：降级规则生效。<br>○ 观察者模式：降级规则不会生效，只会在 MOSN 中打印日志。 |
| 降级服务 | 选择或填写客户端引用的服务端服务名称。星号（*）表示所有服务。 |
| 降级方法 | 填写待降级的方法名。星号（*）表示所有方法。 |

4. 单击 **提交**，然后单击 **确定**。
5. 在降级规则列表中，将刚刚创建的规则状态修改为 **开**。

## 搜索服务降级规则

支持从三个维度进行模糊搜索：




- 应用名称
- 规则名称
- 降级服务

## 编辑服务降级规则

您可以随时编辑已创建的服务降级规则，规则提交后实时生效。

1. 在 **服务降级** 页签，单击目标服务降级规则右侧的 **编辑**。
2. 按需求编辑服务降级规则后，单击 **提交**。

## 删除服务降级规则

您可以删除已创建的服务降级规则，删除操作实时生效，请谨慎操作。

1. 在 **服务降级** 页签，单击目标服务降级规则右侧的 **删除**。
2. 单击 **确定**。

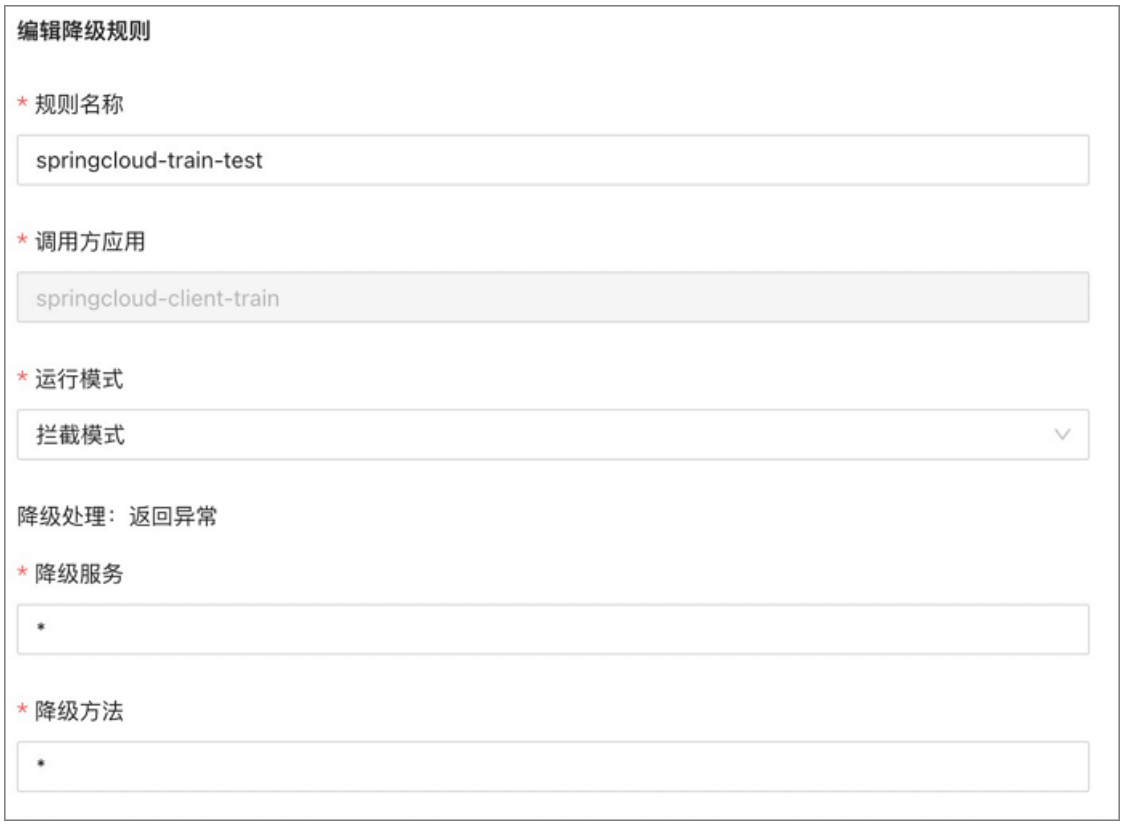
# 8.4.2. 验证服务降级规则

本文以标准 SpringCloud 为例，演示配置降级规则后的请求状态。

降级粒度有三种：

- 应用级降级：对整个客户端应用进行降级处理，由这个应用发出的任何请求都会被降级，服务和方法名需要填写为星号（*）。
- 服务级降级：对客户端应用的引用的某一个服务进行降级，方法名需要填写为星号（*）。
- 方法级降级：对引用服务的某一个方法进行降级，填写具体方法名。

降级规则配置如下：




降级规则开启后，进入客户端容器，多次调用客户端接口：

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问 8084 端口的 hi 接口
curl 0:8084/reservations/hi?count=10
```

可以看到所有的请求都被降级处理掉了：

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 22 毫秒</br>总平均耗时 : 2.2 毫秒</br>单个请求平均耗时 : 0.0 毫秒</br>QPS : 454 </br><h5>Thread-31</h5>
<div>
第 0次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
mosn downgrade exception<br>
第 1次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 2次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 3次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 4次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 5次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 6次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 7次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 8次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 9次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 503 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
</div>
```

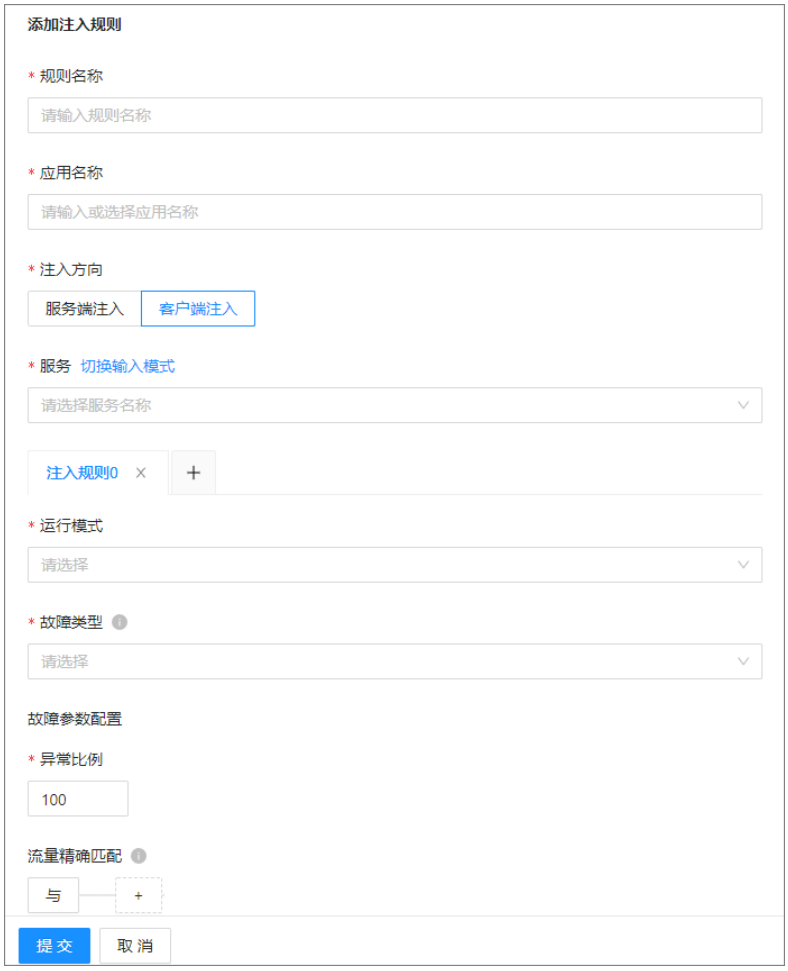
# 8.5. 故障注入

## 8.5.1. 管理故障注入规则

您可以通过故障注入功能向测试应用注入故障，检测应用面对异常时的处理情况。您可以根据检测的情况调整您的应用，以减少应用在正式使用时出现的异常问题。多用于测试环境。

## 添加故障注入规则

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格 > 服务治理**，然后单击 **故障注入** 页签。
3. 单击 **添加注入组规则**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **规则名称** | 设置故障注入规则的名称。<br>最多不超过 255 个字符。 |
| **应用** | 选择或填写目标应用的名称。星号（*）表示所有应用。 |
| **注入方向** | 设置故障注入的方向，可选值为：<br>○ **服务端注入**：在应用的服务端注入故障。<br>○ **客户端注入**：在应用的客户端注入故障。 |

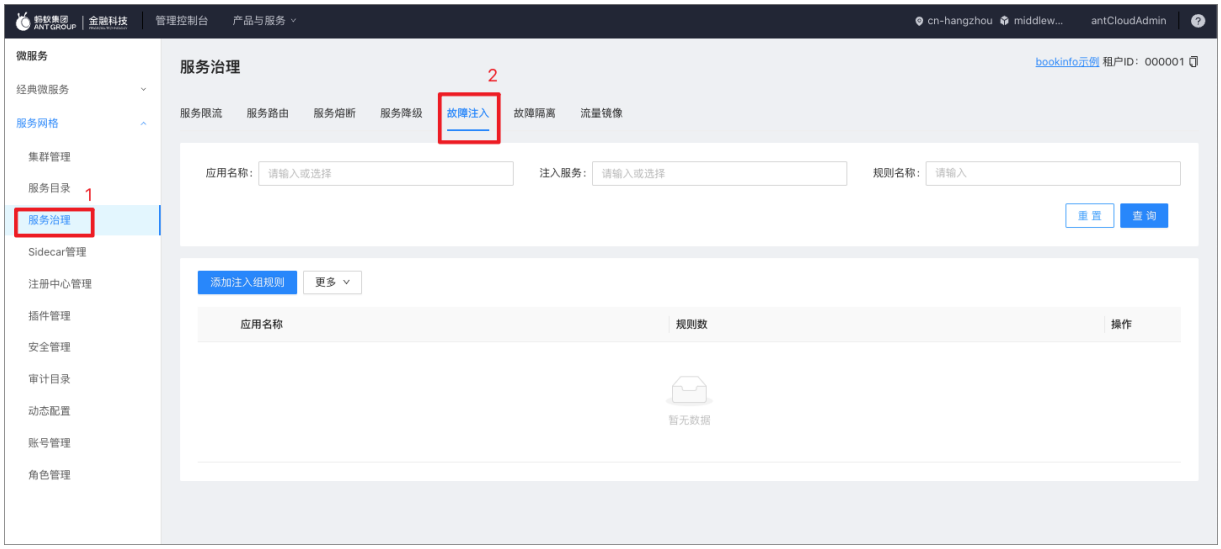
| 参数 | 说明 |
| --- | --- |
| 服务 | 配置注入故障的服务。单击 **切换输入模式** 可在手动填写与下拉选择之间切换。 |
| 方法 | 配置故障注入的方法。星号（*）表示所有方法。 |
| 运行模式 | 配置故障注入规则的运行模式，取值如下：<br>○ **拦截模式**：满足条件的故障注入请求会被注入。<br>○ **观察者模式**：满足条件的故障注入请求不会被注入，只会在 MOSN 里打印日志。 |
| 故障类型 | 故障注入支持注入错误或者超时等事件，方便服务的异常测试，用于模拟服务异常的情况。取值如下：<br>○ **中断异常**：注入运行时异常，中断请求并返回既定的错误状态码。<br>○ **超时异常**：满足条件的请求，会增加响应时间，请求正常调用。 |
| 异常比例 | 设置注入异常流量的比例。例如设置为 80，则只注入 80% 的异常流量。 |
| 流量精确匹配（可选） | 设置流量的匹配条件，满足匹配条件的流量才会使用故障注入规则。置空此项时表示匹配所有流量。<br>您可以配置多条匹配条件，多个条件是与的关系，按顺序进行匹配。参数配置如下：<br>○ **字段**：可选择系统字段和请求头。<br>○ **字段名**：根据字段类型有不同的值。<br>■ **系统字段**：包括流量类型、调用方应用名、调用方 IP、服务方应用名。<br>■ **请求头**：请求头是指协议的请求头。例如 Dubbo 协议取的是 attachment，HTTP 协议取的是 Request Header。用户可以在应用系统中自定义请求头参数和值。<br>○ **选择逻辑**：包括等于、不等于、属于、不属于、正则。<br>○ **字段值**：字段名对应的值。 |

4. 单击 **提交**，然后单击 **确定**。
5. 在故障注入规则列表中，将刚刚创建的故障注入规则的状态改为 **开启**。

> **说明**
>
> 当多条故障注入规则针对同一个服务时，只会生效第一条。

## 搜索故障注入规则

支持从三个维度进行模糊搜索：




- 应用名称
- 注入服务
- 规则名称

## 编辑故障注入规则

您可以随时编辑已创建的故障注入规则，规则提交后实时生效。

1. 在 **故障注入** 页签，单击目标故障注入规则右侧的 **编辑**。
2. 按需求编辑故障注入规则后，单击 **提交**。

## 删除故障注入规则

您可以删除已创建的故障注入规则，删除操作实时生效，请谨慎操作。

1. 在 **故障注入** 页签，单击目标故障注入规则右侧的 **删除**。
2. 单击 **确定**。

# 8.5.2. 验证故障注入规则

本文以标准 SpringCloud 为例，演示配置故障注入规则后的请求状态。

## 服务端故障注入

故障注入规则配置如下：

故障注入规则开启后，进入客户端容器，多次调用客户端接口：

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问 8084 端口的 hi 接口
curl 0:8084/reservations/hi?count=10
```

可以看到所有的请求都被注入了一个异常：

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 28 毫秒</br>总平均耗时 : 2.8 毫秒</br>单个请求平均耗时 : 0.0 毫秒</br>QPS : 357 </br><h5>Thread-33</h5>
<div>
第 0次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 1次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 2次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 3次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 4次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 5次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 6次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 7次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 8次调用，sleep 0 毫秒，耗时 3 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
第 9次调用，sleep 0 毫秒，耗时 2 毫秒，结果：status 500 reading ReservationServiceSpringcloudPressure#sayHi(String)<br>
</div>
```

## 客户端故障注入

故障注入规则配置如下：

在客户端的应用里配置了超时规则，规则开启后，进入客户端容器，多次调用客户端接口：

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问 8084 端口的 hi 接口
curl 0:8084/reservations/hi?count=10
```

可以看到所有的请求都被注入了睡眠时间 1000 ms：

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10
总请求数 : 10</br>总响应耗时 : 10034 毫秒</br>总平均耗时 : 1003.4 毫秒</br>单个请求平均耗时 : 0.0 毫秒</br>QPS : 0 </br><h5>Thread-36</h5>
<div>
第 0次调用，sleep 0 毫秒，耗时 1005 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 1次调用，sleep 0 毫秒，耗时 1003 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 2次调用，sleep 0 毫秒，耗时 1003 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 3次调用，sleep 0 毫秒，耗时 1004 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 4次调用，sleep 0 毫秒，耗时 1003 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 5次调用，sleep 0 毫秒，耗时 1003 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 6次调用，sleep 0 毫秒，耗时 1003 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 7次调用，sleep 0 毫秒，耗时 1003 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 8次调用，sleep 0 毫秒，耗时 1003 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
第 9次调用，sleep 0 毫秒，耗时 1003 毫秒，结果：Read timed out executing POST http://reservation-springcloud-service-pressure-dev/reservations/hi2<br>
</div>
```

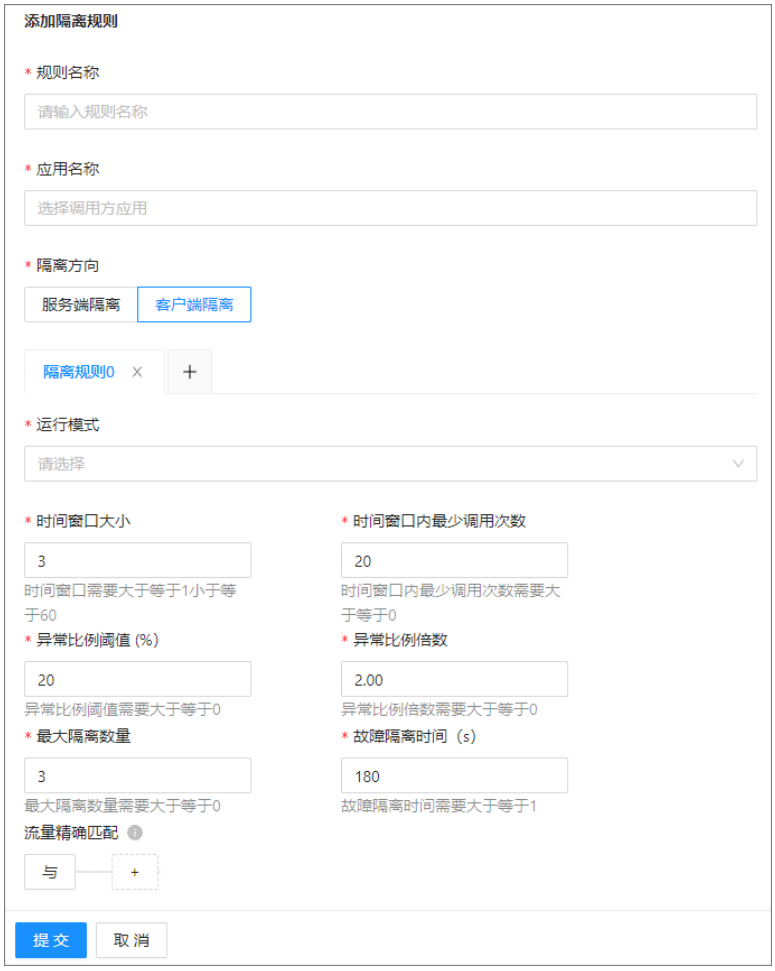
# 8.6. 故障隔离

## 8.6.1. 管理故障隔离规则

故障隔离是在服务实例出现故障时，实现实例级别的精细化摘流隔离，使故障影响范围更小，提高服务的可用性。

### 添加故障隔离规则

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格** > **服务治理**，然后单击 **故障隔离** 页签。
3. 单击 **添加隔离规则**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **规则名称** | 配置故障隔离规则的名称。<br>最多支持 255 个字符。 |
| **应用** | 填写或选择客户端应用。 |
| **隔离方向** | 设置故障隔离的方向，可选值为：<br>○ **服务端隔离**：从服务端侧发现有故障的服务器，并将之暂时隔离。<br>○ **客户端隔离**：从客户端侧发现有故障的服务器，并将之暂时隔离。 |
| **运行模式** | 配置故障隔离规则的运行方式，取值如下：<br>○ **拦截模式**：故障隔离规则生效，有故障的服务器会被暂时隔离。<br>○ **监控模式**：有故障的服务器不会被隔离，只打印日志。 |

| 参数 | 说明 |
| --- | --- |
| **时间窗口大小** | 设置故障检测的时间，与 **时间窗口内最少调用次数** 配合，只有在指定时间段内进行指定次数的请求才会被采集。<br>取值范围：(1,60]<br>单位：秒 |
| **时间窗口内最少调用次数** | 设置指定时间内的最少调用请求次数。<br>取值 ≥ 0。 |
| **异常比例阈值（%）** | 设置异常请求的比例，配合 **异常比例倍数** 参数来确定是否隔离目标服务器。<br>当异常请求的比例 > **异常比例阈值** × **异常比例倍数** 时，系统认为目标服务器异常，将隔离目标服务器。<br>取值范围：(0,100] |
| **异常比例倍数** | 设置异常比例的倍数，配合 **异常比例阈值** 参数来确定是否隔离目标服务器。<br>取值 ≥ 0。 |
| **最大隔离数量** | 设置服务器的隔离数量，即多台服务器故障时最多隔离几台。<br>取值 ≥ 0。 |

4. 单击 **提交**，然后单击 **确定**。
5. 在故障隔离规则列表中，将刚刚创建的故障隔离规则的状态改为 **开**。

## 配置示例

某应用有 A、B、C 三台服务器，三台服务的状态如下：

- A：正常
- B：异常，异常率 20%
- C：异常，异常率 40%

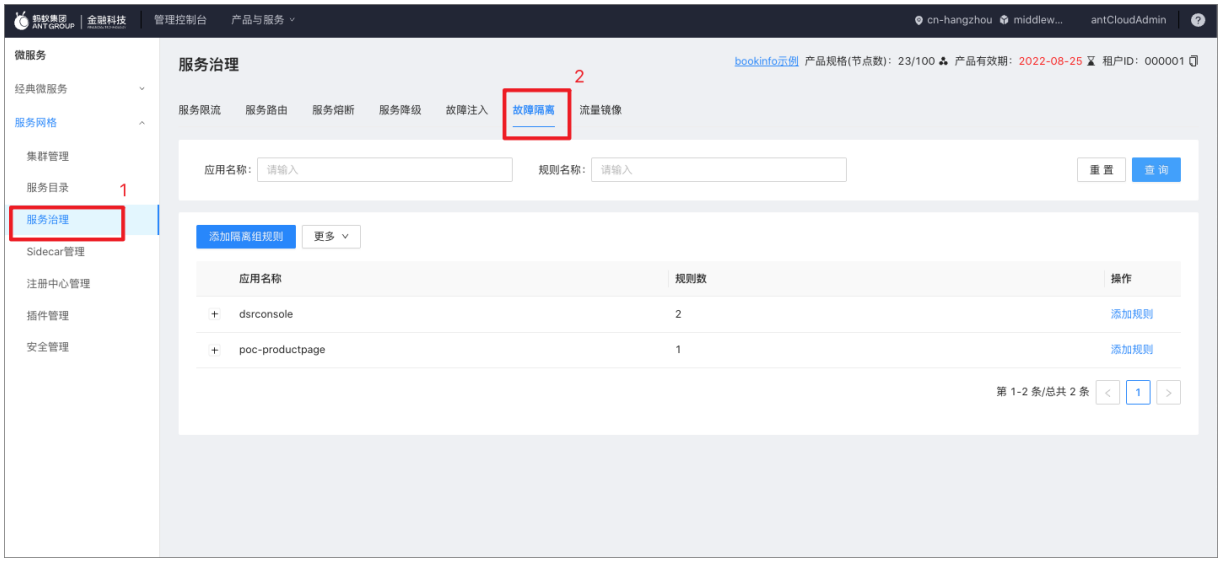
已配置的故障隔离规则如下：

- 时间窗口大小：10
- 时间窗口内最少调用次数：20
- 异常比例阈值：20
- 异常比例倍数：1
- 最大隔离数量：1

如果 10 秒内有 20 个请求，且异常率 ≥ 20%（异常比例倍数 × 异常比例阈值），会触发故障隔离规则。因为 B 的故障率更高，系统会优先将 B 隔离。此时，若将最大隔离数量调整为 2，则 A、B 均会被剔除。

## 搜索故障隔离规则

支持从两个维度模糊搜索：




- 应用名称
- 规则名称

## 编辑故障隔离规则

您可以随时编辑已创建的故障隔离规则，规则提交后实时生效。

1. 在 **故障隔离** 页签，单击目标故障隔离规则右侧的 **编辑**。
2. 按需求编辑故障隔离规则后，单击 **提交**。

## 删除故障隔离规则

您可以删除已创建的故障隔离规则，删除操作会实时生效，请谨慎操作。

1. 在 **故障隔离** 页签，单击目标故障隔离规则右侧的 **删除**。
2. 单击 **确定**。

# 8.6.2. 验证故障隔离规则

本文以标准 SpringCloud 为例，演示配置故障隔离规则后的请求状态。

## 服务端故障隔离

故障隔离规则配置配置如下：

**验证故障隔离规则**

1. 设置服务端接口出错。

   故障隔离会隔离掉符合条件的异常的服务端实例，因此我们需要设置服务端接口出错：

```
kubectl exec -it springcloud-server-train-v1-0 -c mosn-sidecar-container bash

curl 0:8080/status/set?throwException=true
```

2. 进入客户端容器，多次调用客户端接口。

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问8084端口的 hi 接口
curl 0:8084/reservations/hi?count=10\&sleep=2000
```

调用 springcloud-server-train-v1 的请求都异常，返回了 500 错误：

```
#curl 0:8084/reservations/hi?count=10\&sleep=2000
总请求数 : 10</br>总响应耗时 : 20077 毫秒</br>总平均耗时 : 2007.7 毫秒</br>单个请求平均耗时 : 1003.2 毫秒</br>QPS : 0 </br><h5>Thread-54</h5>
<div>
第 0次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:10.654+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = QÕ65{6\\D6","path":"/reservations/hi2"}<br>
第 1次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi QÕ65{6\D6! 我是[10.0.200.97]<br>
第 2次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:14.669+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = QÕ65{6\\D6","path":"/reservations/hi2"}<br>
第 3次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi QÕ65{6\D6! 我是[10.0.200.97]<br>
第 4次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:18.685+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = QÕ65{6\\D6","path":"/reservations/hi2"}<br>
第 5次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi QÕ65{6\D6! 我是[10.0.200.97]<br>
第 6次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:22.700+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = QÕ65{6\\D6","path":"/reservations/hi2"}<br>
第 7次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi QÕ65{6\D6! 我是[10.0.200.97]<br>
第 8次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:26.716+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = QÕ65{6\\D6","path":"/reservations/hi2"}<br>
第 9次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi QÕ65{6\D6! 我是[10.0.200.97]<br>
</div>
```

3. 开启故障隔离规则。

故障隔离规则开启并且其中一台服务端异常设置后，我们可以调用客户端接口进行验证。

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10\&sleep=2000
总请求数 : 10</br>总响应耗时 : 20074 毫秒</br>总平均耗时 : 2007.4 毫秒</br>单个请求平均耗时 : 1404.8 毫秒</br>QPS : 0 </br><h5>Thread-78</h5>
<div>
第 0次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:41:15.422+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = 6F6y66\u0016I0\u0001","path":"/reservations/hi2"}<br>
第 1次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi 6F6y66I0! 我是[10.0.200.97]<br>
第 2次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:41:19.437+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = 6F6y66\u0016I0\u0001","path":"/reservations/hi2"}<br>
第 3次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi 6F6y66I0! 我是[10.0.200.97]<br>
第 4次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:41:23.453+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = 6F6y66\u0016I0\u0001","path":"/reservations/hi2"}<br>
第 5次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi 6F6y66I0! 我是[10.0.200.97]<br>
第 6次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi 6F6y66I0! 我是[10.0.200.97]<br>
第 7次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi 6F6y66I0! 我是[10.0.200.97]<br>
第 8次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi 6F6y66I0! 我是[10.0.200.97]<br>
第 9次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi 6F6y66I0! 我是[10.0.200.97]<br>
</div>
```

验证结果说明，前 10s 内调用了异常服务，10s 之后异常的服务被隔离掉了。请求发送到正常的服务上了。

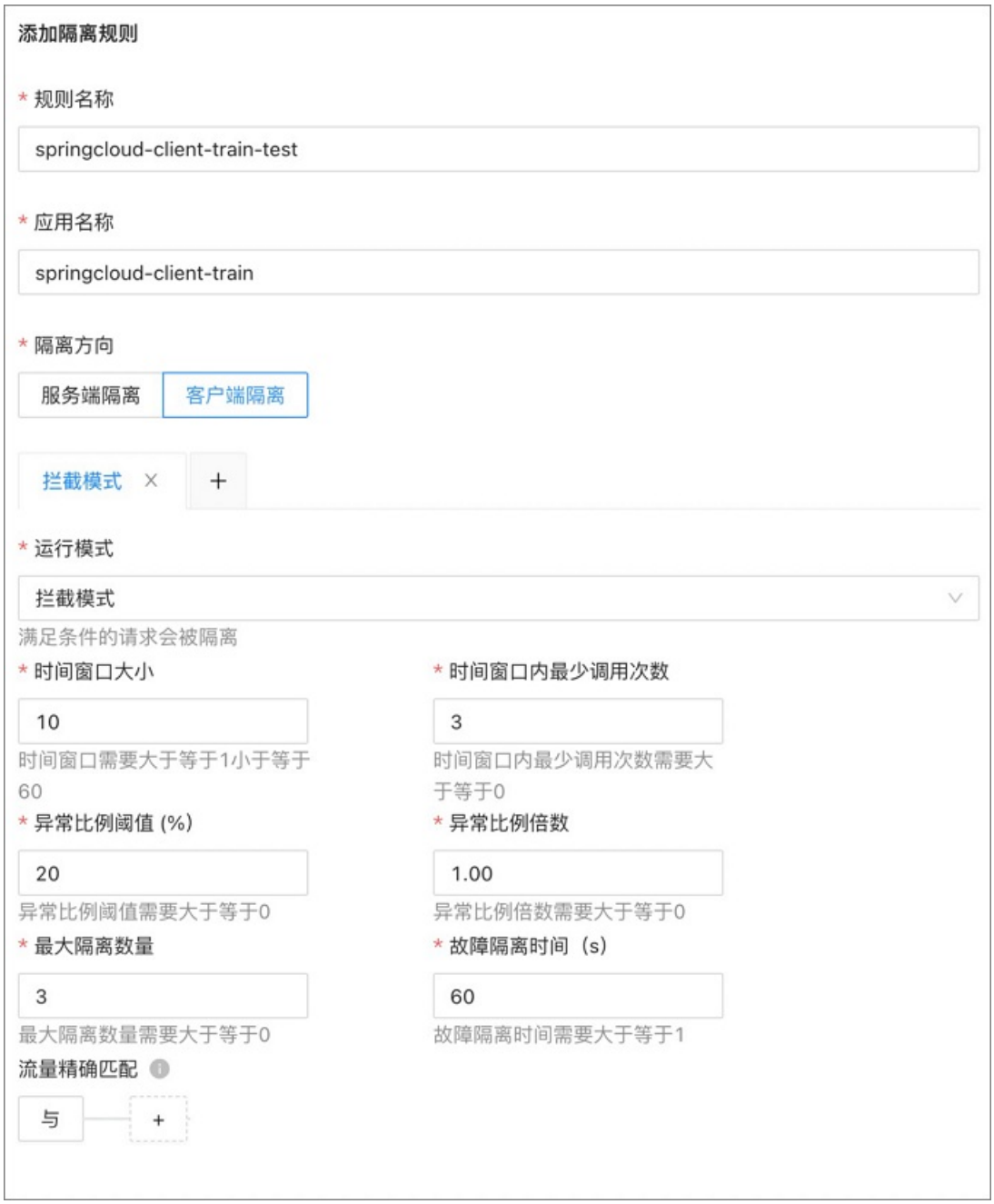
隔离期间关闭服务端异常，再次进行调用仍然只会调用到一台机器：

```
curl 0:8080/status/set?throwException=false
```

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10\&sleep=2000
总请求数 : 10</br>总响应耗时 : 20079 毫秒</br>总平均耗时 : 2007.9 毫秒</br>单个请求平均耗时 : 2007.9 毫秒</br>QPS : 0 </br><h5>Thread-59</h5>
<div>
第 0次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 1次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 2次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 3次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 4次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 5次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 6次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 7次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 8次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
第 9次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi Y]66$J696! 我是[10.0.200.97]<br>
</div>
```

## 客户端故障隔离

故障隔离规则配置配置如下：




**验证故障隔离规则**

1. 设置服务端接口出错。

   故障隔离会隔离掉符合条件的异常的服务端实例，因此我们需要设置服务端接口出错。

```
kubectl exec -it springcloud-server-train-v1-0 -c mosn-sidecar-container bash

curl 0:8080/status/set?throwException=true
```

2. 进入客户端容器，多次调用客户端接口。

```
kubectl exec -it springcloud-client-train-0 -c mosn-sidecar-container bash

## 访问8084端口的 hi 接口
curl 0:8084/reservations/hi?count=10\&sleep=2000
```

调用 springcloud-server-train-v1 的请求都异常，返回了 500 错误：

```
#curl 0:8084/reservations/hi?count=10\&sleep=2000
总请求数 : 10</br>总响应耗时 : 20077 毫秒</br>总平均耗时 : 2007.7 毫秒</br>单个请求平均耗时 : 1003.2 毫秒</br>QPS : 0 </br><h5>Thread-54</h5>
<div>
第 0次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:10.654+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = Q0005{0\\D0","path":"/reservations/hi2"}<br>
第 1次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi Q0005{0\D0! 我是[10.0.200.97]<br>
第 2次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:14.669+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = Q0005{0\\D0","path":"/reservations/hi2"}<br>
第 3次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi Q0005{0\D0! 我是[10.0.200.97]<br>
第 4次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:18.685+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = Q0005{0\\D0","path":"/reservations/hi2"}<br>
第 5次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi Q0005{0\D0! 我是[10.0.200.97]<br>
第 6次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:22.700+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = Q0005{0\\D0","path":"/reservations/hi2"}<br>
第 7次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi Q0005{0\D0! 我是[10.0.200.97]<br>
第 8次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:19:26.716+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = Q0005{0\\D0","path":"/reservations/hi2"}<br>
第 9次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi Q0005{0\D0! 我是[10.0.200.97]<br>
</div>
```

3. 开启故障隔离规则。

故障隔离规则开启并且其中一台服务端异常设置后，我们可以调用客户端接口进行验证。

```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10\&sleep=2000
总请求数 : 10</br>总响应耗时 : 20074 毫秒</br>总平均耗时 : 2007.4 毫秒</br>单个请求平均耗时 : 1404.8 毫秒</br>QPS : 0 </br><h5>Thread-78</h5>
<div>
第 0次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:41:15.422+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = 0F6y00\u0016I0\u0001","path":"/reservations/hi2"}<br>
第 1次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi 0F6y00I0! 我是[10.0.200.97]<br>
第 2次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:41:19.437+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = 0F6y00\u0016I0\u0001","path":"/reservations/hi2"}<br>
第 3次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi 0F6y00I0! 我是[10.0.200.97]<br>
第 4次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: status 500 reading ReservationServiceSpringcloudPressure#sayHi(String); content:
{"timestamp":"2022-05-05T12:41:23.453+0000","status":500,"error":"Internal Server Error","message":"some exception happend. name = 0F6y00\u0016I0\u0001","path":"/reservations/hi2"}<br>
第 5次调用, sleep 2000 毫秒, 耗时 2006 毫秒, 结果: Hi 0F6y00I0! 我是[10.0.200.97]<br>
第 6次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi 0F6y00I0! 我是[10.0.200.97]<br>
第 7次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi 0F6y00I0! 我是[10.0.200.97]<br>
第 8次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi 0F6y00I0! 我是[10.0.200.97]<br>
第 9次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi 0F6y00I0! 我是[10.0.200.97]<br>
</div>
```

验证结果说明，前 10s 内调用了异常服务，10s 之后异常的服务被隔离掉了。请求发送到正常的服务上了。

隔离期间关闭服务端异常，再次进行调用仍然只会调用到一台机器：

```
curl 0:8080/status/set?throwException=false
```

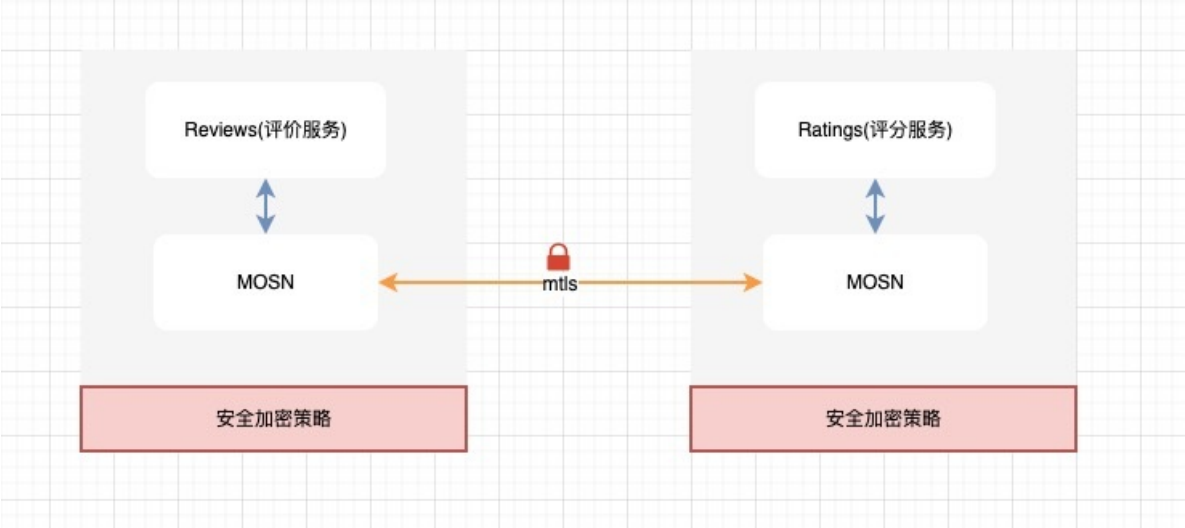
```
[root@springcloud-client-train-0 /home/admin]
#curl 0:8084/reservations/hi?count=10\&sleep=2000
总请求数 : 10</br>总响应耗时 : 20082 毫秒</br>总平均耗时 : 2008.2 毫秒</br>单个请求平均耗时 : 2008.0 毫秒</br>QPS : 0 </br><h5>Thread-82</h5>
<div>
第 0次调用, sleep 2000 毫秒, 耗时 2007 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 1次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 2次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 3次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 4次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 5次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 6次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 7次调用, sleep 2000 毫秒, 耗时 2009 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 8次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
第 9次调用, sleep 2000 毫秒, 耗时 2008 毫秒, 结果: Hi k00L000F! 我是[10.0.200.97]<br>
</div>
```

# 9.安全管理

## 9.1. 通信加密

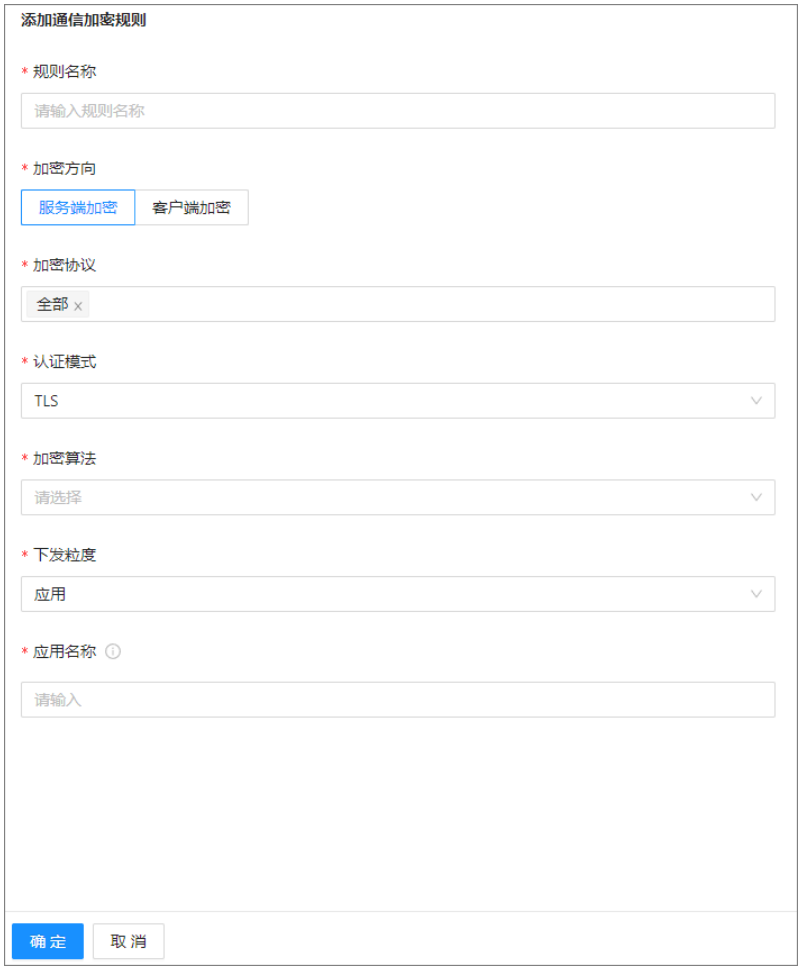
通信加密功能是对调用链路上的请求信息和响应信息进行加密，使通信链路具备认证、报文加密能力。

通信加密流程如下：




### 创建通信加密规则

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格** > **安全管理**，然后单击 **通信加密**。
3. 单击 **添加通信加密规则**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| 规则名称 | 填写加密规则的名称，最多不超过 255 个字符。 |
| 加密方向 | 配置通信加密的方向，可选值：<br>○ 服务端加密：在服务端侧的链路进行加密。<br>○ 客户端加密：在客户端侧的链路进行加密。 |
| 加密协议 | 支持全部协议加密，也支持单协议加密。支持 DEFAULT(bolt)、springcloud、dubbo、crpc。<br>可下拉框选择，也可手动输入自定义协议。 |

| | |
|---|---|
| 认证模式 | 配置加密规则的认证模式，可选值：<br>○ TLS：单向加密，可分阶段上线全站加密，支持可控降级。选择 TLS 加密只需要配置服务端的应用即可。<br>○ mTLS：双向加密，服务端会校验客户端生成的证书。需要在服务端和客户端都配置一份规则，规则的加密算法一致，即均为普密或均为国密。 |
| 加密算法 | 用于在 SSL / TLS 握手期间协商安全设置的算法的组合，即密码套件。<br>○ 普密套件：如：ECDHE-ECDSA-AES256-GCM-SHA384<br>○ 国密套件：如：ECDHE-RSA-SM4-SM3。<br>**重要**<br>国密需要特殊边车镜像。 |
| 下发粒度 | 配置规则下发的粒度，目前支持 **应用**。安全加密规则作用于目标应用上，与该应用的通信都会被加密。 |
| 应用名称 | 配置要加密的应用名称，同一个应用不能同时生效多种规则。 |

4. 单击 **确定** 按钮。
5. 在通信加密规则列表，将刚刚配置的加密规则的状态修改为 **开**。

以 SpringCloud 应用为例， 通过网络抓包即可观察到加密效果：

- 正常流量抓包

```
20:58[redacted]515, offset 0, flags [DF], proto TCP (6), length 52)
    [redacted].amanda: Flags [.], cksum 0xa2a4 (correct), seq 3659, ack 7131, win 488, options [nop,
nop,TS val 1684854202 ecr 2081975005], length 0
E..4.g@.;.RJ..}.dX.Q..'`."...\P{...........
dl..|.j.GET /ratings/0 H
20:58:20.004713 IP (tos 0x0, ttl 59, id 616, offset 0, flags [DF], proto TCP (6), length 170)
    [redacted]253.81.amanda: Flags [P.], cksum 0xc00d (correct), seq 3659:3777, ack 7131, win 488, options
 [nop,nop,TS val 1684854699 ecr 2081975005], length 118
E....h@.;.Q...}.dX.Q..'`."...\P{...........
dl..|.j.GET /ratings/0 HTTP/1.1
User-Agent: Java/1.8.0_302
Host: 127.127.3.75:10088
Accept: */*
Connection: keep-alive

timeout=60
Conn
20:58[redacted]6641, offset 0, flags [DF], proto TCP (6), length 282)
    [redacted]3.33014: Flags [P.], cksum 0xec1f (incorrect -> 0x3c83), seq 7131:7361, ack 3777, win
227, options [nop,nop,TS val 2081975507 ecr 1684854699], length 230
E.....@.@.4.dX.Q..}.'`...\P{.".%...........
|.l.dl..HTTP/1.1 200 OK
Date: Thu, 18 Aug 2022 12:58:19 GMT
Content-Type: application/json
Content-Length: 74
Keep-Alive: timeout=60
Connection: keep-alive

{"id":0,"count":2,"countErr":null,"ratings":{"reviewer1":5,"reviewer2":4}}38e40bed8e5b3d92
```

请求报文

响应报文

- 加密流量抓包

```
22:41:29.016277 IP (tos 0x0, ttl 64, id 53514, offset 0, flags [DF], proto TCP (6), length 52)
    [redacted].81.amanda: Flags [.], cksum 0xeb39 (incorrect -> 0x3d3d), seq 9836, ack 16411, win 1009,
options [nop,nop,TS val 1691043711 ecr 2088164514], length 0
E..4.
@.@.~...}.dX.Q..'`.b@..n.......9.....
d.G.|v.............)...
22:41:29.519376 IP (tos 0x0, ttl 64, id 53515, offset 0, flags [DF], proto TCP (6), length 200)
    [redacted]253.81.amanda: Flags [P.], cksum 0xebcd (incorrect -> 0x15ba), seq 9836:9984, ack 16411, win
1009, options [nop,nop,TS val 1691044214 ecr 2088164514], length 148
E.....@.@.~...}.dX.Q..'`.b@..n.............
d.Iv|v.............=
..V..<f..q....x..'3.<(.pv....l.^.
s.....H.$....67T....+$..FH..4k.s|"0ix.0...
.*N/...+_.lZ.'......-..=.....i=...#..z..)...A?a..... ..e&..V..1.L....UA..
22:41:29.525127 IP (tos 0x0, ttl 59, id 57675, offset 0, flags [DF], proto TCP (6), length 311)
    [redacted].203.38668: Flags [P.], cksum 0x35c4 (correct), seq 16411:16670, ack 9984, win 979, option
s [nop,nop,TS val 2088165023 ecr 1691044214], length 259
E..7.K@.;.rbdX.Q..}.'`...n...bAR....5......
|v..d.Iv............=7........E.1....l....0.).K.........}......L....9A.l...G.uj..2..;.mK.b..J...."2ZUD.c...s7.......1.y....+...
.......
..._.....n...?[........<...X.........C.u.....@....#.%K1...y#<.C`.dL.r.CKB....M..#.p...T.p.....{.G$D...X.`,X\.....q......W..p+C..
...+................
```

## 编辑通信加密规则

您可以随时修改已创建的通讯加密规则，规则提交后实时生效。

1. 在通信加密规则列表，单击目标规则右侧的 **编辑**。
2. 根据您的需求修改加密规则后，单击 **确定**。

   您可修改 **规则名称**、**加密协议**、**认证模式**、**加密算法**。

## 删除通信加密规则

您可以随时删除不再需要的通讯加密规则，删除操作实时生效，请谨慎操作。

1. 在通信加密规则列表，单击目标规则右侧的 **删除**。

   已开启的通讯加密规则无法删除，您需要将通讯加密规则的 **状态** 开关关闭。

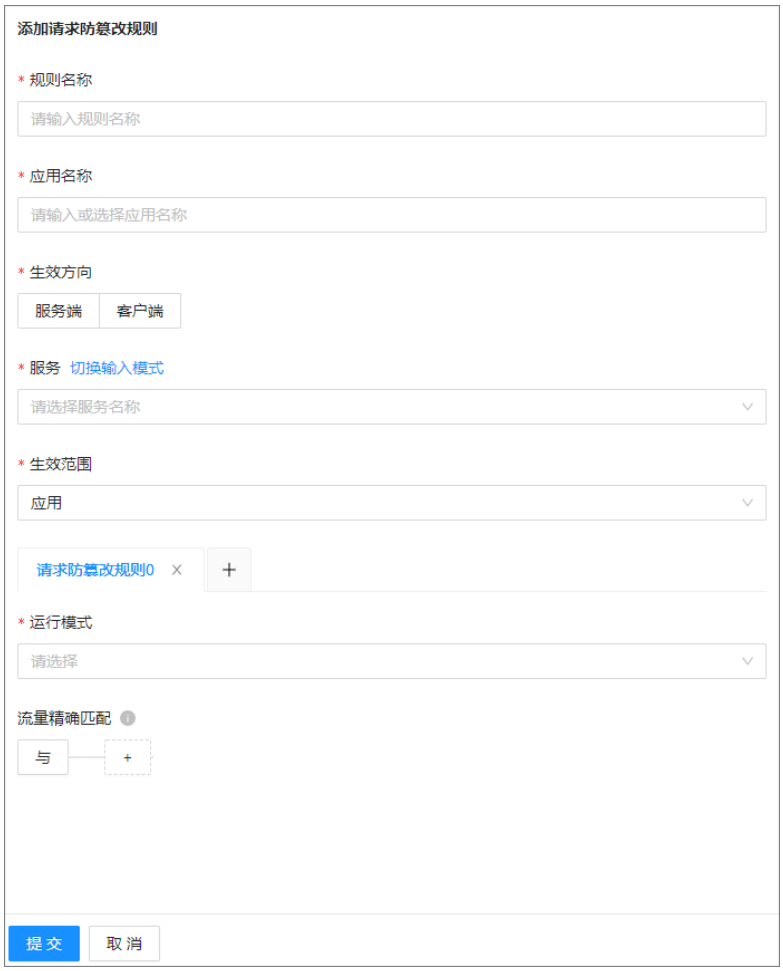
2. 单击 **确认**。

# 9.2. 请求防篡改

请求防篡改功能可以为服务消费者和服务提供者的通信链路增加数据完整性校验， 防止报文被篡改。开启请求防篡改后，边车将客户端请求报文体增加数字签名，服务端边车将验证数字签名、校验报文完整性，进而达到保护服务提提供者效果。

## 操作步骤

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格 > 安全管理**，然后单击 **请求防篡改**。
3. 单击 **添加请求防篡改规则**，然后配置以下参数：




| 参数 | 说明 |
| --- | --- |
| **规则名称** | 配置规则名称，最多不超过 255 个字符。 |
| **应用名称** | 选择或填写目标应用名称。星号（*）表示所有应用。 |
| **生效方向** | 配置规则生效方向，可选值：<br>○ **服务端**：请求防篡改规则作用于服务端（ingress）。<br>○ **客户端**：请求防篡改规则作用于客户端（egress）。 |

| | |
|---|---|
| **服务** | 细粒度治理控制，可选择全部服务，或选择当前应用下指定服务。若选择指定服务， 则只有该属于该服务的流量生效。 单击 **切换输入模式** 可在手动填写与下拉选择之间切换。 |
| **生效范围** | 配置规则生效的范围，可选值：<br>○ **应用**：规则作用于目标应用上，该应用的请求都会被保护。<br>○ **label**：仅作用于当前应用下， 带有当前 label 的实例（Pod）上生效。 |
| **label** | 填写目标机器 label 的 key 和 value。您可以配置多个 label，请求防篡改规则将在拥有这些 label 的机器上生效。<br>仅在 **下发粒度** 选择 **label** 时配置此项。 |
| **运行模式** | 配置规则的运行模式，取值如下：<br>○ **拦截模式**：满足条件的请求会触发请求防篡改规则。<br>○ **观察者模式**：请求防篡改规则不会实际生效，满足触发条件时，只会在 MOSN 里打印日志。 |
| **流量精确匹配** | 配置之后，只有满足匹配条件的流量才会使用这条规则。置空表示匹配所有流量。<br>您可以配置多条匹配条件，多个条件是与的关系，按顺序进行匹配。参数配置如下：<br>○ **字段**：可选择系统字段和请求头。<br>○ **字段名**：根据字段类型有不同的值。<br>■ 系统字段：包括流量类型、调用方应用名、调用方 IP、服务方应用名。<br>■ 请求头：请求头是指协议的请求头，比如 Dubbo 协议取的是 attachment，HTTP 协议取的是 Request Header。用户可以在应用系统中自定义请求头参数和值。<br>○ **选择逻辑**：包括等于、不等于、属于、不属于、正则。<br>○ **字段值**：字段名对应的值。 |

4. 单击 **提交**，然后单击 **确定**。

   针对同一应用，您可以配置多条请求防篡改规则，多条规则是或的关系。

5. 在规则列表，将刚创建的规则状态改为 **开**。

以 HTTP 请求为例，服务端网络抓包可发现在客户端的请求报文头中将携带报文内容的数字签名及数字证书 ID， 服务端将会对数字签名进行验证。

```
20:50:39.508442 IP (tos 0x0, ttl 59, id 60613, offset 0, flags [DF], proto TCP (6), length 52)
    11.158.125.203.57156 > 100.88.253.81.amanda: Flags [.], cksum 0xeedc (correct), seq 3192, ack 1611, win 765, options [nop,
nop,TS val 1684394203 ecr 2081515006], length 0
E..4..@.;.g...}.dX.Q.D'`....U~4............
de..|.e.................
20:50:40.083589 IP (tos 0x0, ttl 59, id 60614, offset 0, flags [DF], proto TCP (6), length 508)
    11.158.125.2[illegible]56 > 100.88.253.81.amanda: Flags [P.], cksum 0x6a4b (correct), seq 3192:3648, ack 1611, win 765, options
 [nop,nop,TS val 1684394778 ecr 2081515006], length 456
E.....@.;.f"..}.dX.Q.D'`....U~4.....jK.....
de..|.e.GET /ratings/0 HTTP/1.1
User-Agent: Java/1.8.0_302
Host: 1[illegible]:10088
Accept: */*
Connection: keep-alive
Clientchecksum: 73c87ae58dc9be001d0221c0f1117c31d7d62a9e1eec014fdebed93b29a818293cdeeec0f58bde9d0fa0de28b90243f77e38e40bed8e5b
3d92b2129cc90dc68b639d9f74f5219cfbb598ed91d0c51f2f6085ac6738511505f525d3414b06f44c82395ac71de5c4f7d69082fc47a321e2c10d8307481a
1b59fef7f39c047a373a
X-Mosn-Cert-Id: certId-dsr-ca-0C1B3B34778240000C1B3E[illegible]
```

## 编辑请求防篡改规则

您可以随时编辑已创建的请求防篡改规则，规则提交后实时生效。

1. 在规则列表中，单击目标应用左边的加号（+）。
2. 单击目标规则操作列的 **编辑**。
3. 根据您的需求修改规则后单击 **提交**。

## 导出请求防篡改规则

您可以将已配置的请求防篡改规则导出，以备份您的请求防篡改规则。

1. 在 **服务鉴权** 页签，选择 **更多** > **导出**。
2. 选中目标规则后，单击 **导出**。

## 导入请求防篡改规则

您可以导入备份的请求防篡改规则，以快速创建请求防篡改规则。

1. 在 **服务鉴权** 页签，选择 **更多** > **导入**。
2. 单击 **选择文件**，然后选中目标文件后，单击 **确定**。

## 删除请求防篡改规则

您可以随时删除已创建的请求防篡改规则，删除操作实时生效，请谨慎操作。

1. 在规则列表中，单击目标应用左边的加号（+）。
2. 单击目标规则操作列的 **删除**，然后单击 **确定**。

# 9.3. 访问授权

访问授权功能可以对服务消费方的请求进行鉴权，鉴权通过才能访问。

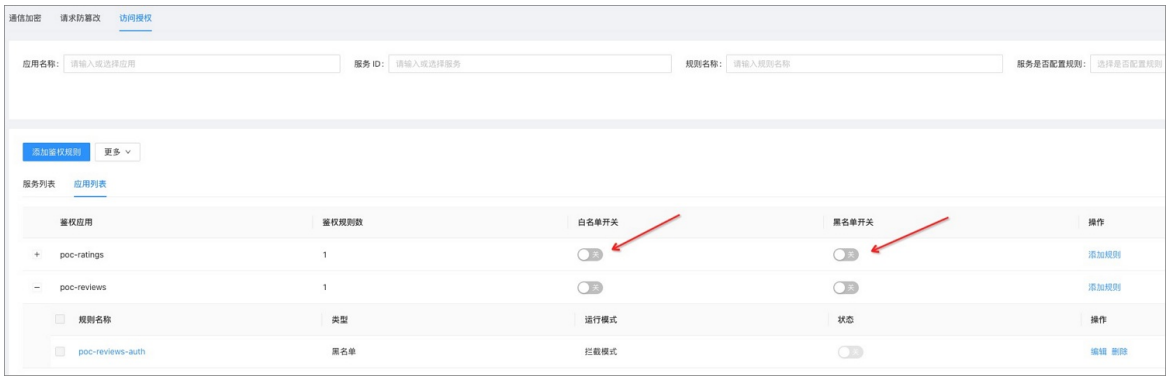
## 创建鉴权规则

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格** > **安全管理**，然后单击 **访问授权**。
3. 单击 **添加通信加密规则**，然后配置以下参数：

| 参数 | 说明 |
| --- | --- |
| 规则名称 | 填写访问鉴权规则的名称。<br>仅支持中文、英文、数字、下划线（_），最多 255 个字符。 |
| 鉴权粒度 | 配置鉴权的粒度，可选值为：<br>○ **应用级**：对某个应用添加鉴权规则。<br>○ **服务级**：对应用下的一个或多个服务添加鉴权规则。 |
| 类型 | 配置鉴权类型，可选值为：<br>○ 白名单：选择此项时，匹配条件的请求会被放通。<br>○ 黑名单：选择此项时，匹配条件的请求会被拒绝。 |
| 应用 | 选择或填写待鉴权的应用。星号（*）表示所有应用。 |

| | |
|---|---|
| 服务 | 选择或填写应用下的一个或多个服务。<br>仅在 **鉴权粒度** 选择 **服务级** 时配置。 |
| 运行模式 | 配置访问鉴权规则的运行方式，取值如下：<br>○ **拦截模式**：鉴权规则生效，则拒绝请求。<br>○ **观察者模式**：鉴权规则生效时，不拒绝请求，只打印日志。 |
| 匹配条件 | 配置鉴权规则的匹配条件，符合条件的流量会被鉴权。<br>可配置多条匹配规则，各匹配规则之间是与的关系。参数配置如下：<br>○ **字段**：可选择系统字段和自定义字段。<br>○ **字段名**：根据字段类型有不同的值。<br>■ 系统字段：可选择调用方应用名、调用方 IP、服务方应用名、服务方方法名。<br>■ 自定义字段：根据实际需求自行设置字段名。<br>○ **选择逻辑**：包括等于、不等于、属于、不属于、正则。<br>○ **字段值**：填入所选字段的值。 |

4. 单击 **提交**，然后单击 **确定**。
5. 在鉴权规则列表中，将刚刚创建的鉴权规则的状态改为 **开**。




6. 根据设置的类型打开白名单或黑名单的开关。

   规则类型为白名单时，打开白名单开关；反之，打开黑名单开关。否则，鉴权规则不生效。

## 编辑鉴权规则

您可以随时编辑已创建的鉴权规则，规则提交后实时生效。

1. 在 **访问鉴权** 页签，单击目标鉴权规则右侧的 **编辑**。
2. 按需求编辑鉴权规则后，单击 **提交**。

## 删除鉴权规则

您可以删除已创建的鉴权规则，删除操作会实时生效，请谨慎操作。

1. 在 **访问鉴权** 页签，将目标鉴权规则的状态改为 **关**。

2. 单击目标鉴权规则右侧的 **删除**。然后单击 **确定**。

## 导出鉴权规则

您可以将已配置的鉴权规则导出，以备份您的鉴权规则。

1. 在 **访问鉴权** 页签，选中需要导出的鉴权规则。
2. 单击 **批量导出**，然后单击 **确定**。

您也可以选择 **更多** > **全部导出**，一次性导出所有的鉴权规则。导出的鉴权规则会存放在浏览器默认的下载文件夹中。

## 导入鉴权规则

您可以导入备份的鉴权规则，以快速创建鉴权规则。

1. 在 **访问鉴权** 页签，选择 **更多** > **全部导入**。
2. 单击 **选择文件**，然后选中目标文件后，单击 **确定**。

# 10.审计目录

审计目录功能用于查询用户的操作，并对操作进行溯源。操作审计的维度是实体 ID + 实体类型。

> **说明**
>
> 实体 ID 和实体类型相同的数据会保留最近 20 条，与按时间来保留数据的设计相比，该数据保留方式的优点在于：即使实体 ID 和实体类型已经创建了很长时间，您仍可以通过审计记录查询出数据。而如果以时间为保留标准，超过预定时间后，数据将会丢失。

## 应用场景

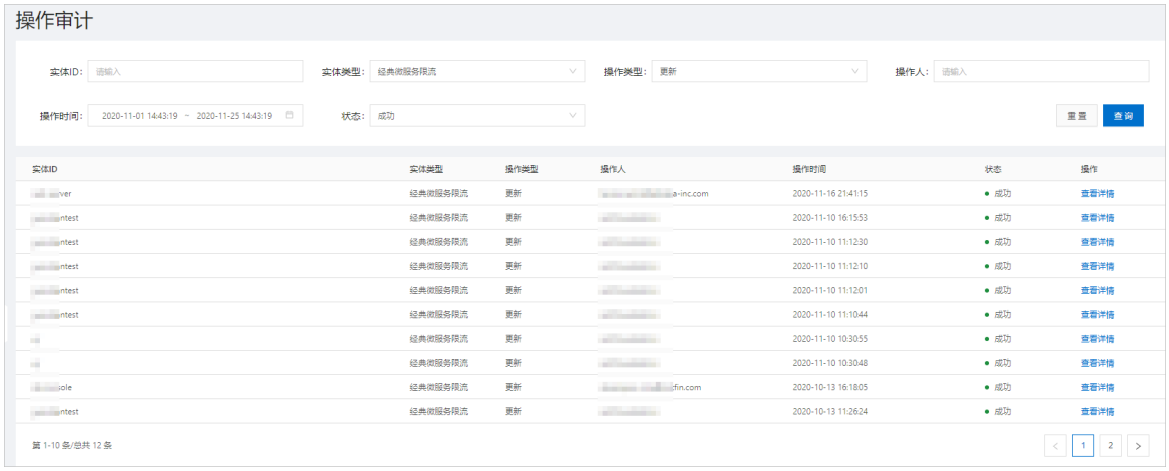
假设 `org.example.services.echoService.XyEchoService:1.0.0:default@crpc` 服务多次更新了路由规则，且部分路由规则失效了。您想查一下每次路由规则更新的内容，以及谁修改了失效的路由规则，可以通过下述步骤实现：

1. 根据实体 ID（服务级的实体 ID 是服务 ID；应用级的实体 ID 是应用名）和实体类型（服务路由）查询出最近的 20 条数据。
2. 单击审计记录的 **查看详情**，分析请求参数和响应参数。

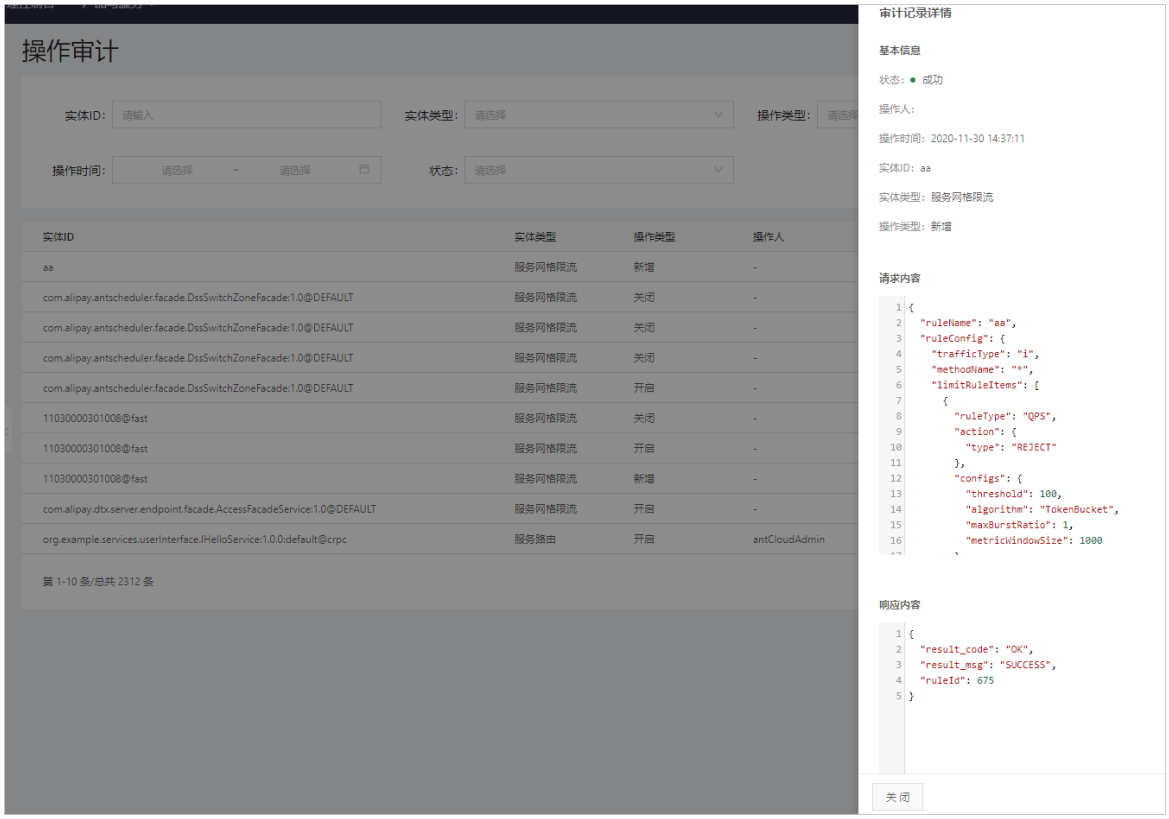
## 查询审计记录

1. 登录微服务控制台。
2. 在左侧菜单栏选择 **服务网格** > **审计目录**。
3. 在操作记录列表中查看操作审计的基本信息。

   基本信息包括实体 ID、实体类型、操作类型、操作人等。您也可以在搜索栏查询指定操作。



4. 单击目标操作右侧的 **查看详情**，查看操作的详细信息。



# 11.动态配置

## 11.1. 概述

动态配置（Distributed Resource Management，简称 DRM）是一个分布式环境下，实时动态的配置管理框架。可以在应用没有重启的情况下，完成配置的动态更新。广泛用于业务参数配置、应急开关切换等场景。

动态配置是服务网格下的模块之一，您只需要在每个环境开通服务网格，即可使用动态配置。用户实例之间的数据通过实例标识进行逻辑隔离，保证数据安全。

动态配置具有以下优点：

- 编程 API 简单

  面向注解和普通 JavaBean 编程，编程方式统一且简单。

- 实时性高

  秒级推送能力，集群实时配置变更。

- 一致性高

  除变更推送能力外，客户端还定时检查数据版本，一旦有数据不一致就触发主动拉数据。

## 11.2. 动态配置快速入门

动态配置（DRM）功能是基于 SOFABoot 工程实现的。本文介绍如何快速实现动态配置。

### 前提条件

- 已完成环境配置，配置详情请参见 搭建环境。
- 已下载 示例工程，并将示例工程主 `pom.xml` 文件的 SOFABoot 版本号改为最新版。SOFABoot 版本信息，请参见 版本说明。

### 配置步骤

1. 在本地开发应用。

   i. 在 SOFABoot 工程的待创建动态配置类的模块中，添加下述依赖：

   ```
   <dependency>
       <groupId>com.alipay.sofa</groupId>
       <artifactId>ddcs-enterprise-sofa-boot-starter</artifactId>
   </dependency>
   ```

   您无需关注该依赖版本。

   ii. 进行安全配置。

   为保障中间件的安全性，所有的调用均需要验证访问者的身份。配置方式，请参考 引入 SOFA 中间件。

   iii. 创建动态配置类。

   配置类代码示例：

```
@DObject(region ="AntCloud", appName ="dynamic-configuration-tutorial", id ="com.antcloud.tutorial.configuration.DynamicConfig")
    public class DynamicConfig{

        @DAttribute
        private String name;

        @DAttribute
        private int age;

        @DAttribute
        private boolean man;

        public void init(){
            DRMClient.getInstance().register(this);
        }

        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;
        }

        public boolean isMan(){
            return man;
        }

        public void setMan(boolean man){
            this.man = man;
        }
    }
```

代码说明：

- 该配置类是一个普通的 Java 类，要符合 Java Bean 的规范，有若干私有属性，属性有对应的 `get` 和 `set` 方法。例如上面的 `name` 、 `age` 、 `man` 称为资源属性，资源属性只允许 `String` 和基本类型。

- 在配置类上加上 `@DObject` 注解，它的包名是 `com.alipay.drm.client.api.annotation` 。`@DObject` 需要提供 `id` 、 `region` 、 `appName` 属性。
  - `id` ：是全站唯一的字符串，一般用全类名来保证唯一，如不设值，则默认为全类名。
  - `region` ：用于区分不同组织的域，如可为每个子公司设定独立域。
  - `appName` ：是应用名。
- 在资源属性上加上 `@DAttribute` 注解，包名同样是 `com.alipay.drm.client.api.annotation` 。
- 动态配置框架将通过反射的方式调用 `get` 、 `set` 方法，从而读写资源属性。在特殊应用场景下，允许改变 `get` 、 `set` 方法的内容，但是不可以修改方法的形式（方法名、参数、返回值）。因为系统启动时动态配置框架会检查该属性是否符合 Java Bean 的规范，如果不符合，会跳过注册这个属性。
- 提供两个可选的注解 `@BeforeUpdate` ， `@AfterUpdate` 。如果需要在每个属性更新前或更新后执行统一的操作，例如打印日志，可以提供参数 `(String,Object)` 和无返回值的方法，并打上相应注解。

  > **说明**
  > 这两个方法被调用时，传入的参数都是属性名和本次 set 方法的入参，并不是对应的私有属性更新前和更新后的值。这两个方法只适合用来执行打日志等次要任务，真正的业务逻辑要放在 set 方法中。

- 调用 `register` 方法注册到动态配置客户端后，即可享受服务端动态修改数据后的秒级推送能力。

iv. 进行 xml 配置。

示例如下：

```
<bean id="dynamicConfig" class="com.antcloud.tutorial.configuration.tutorial.config
.DynamicConfig" init-method="init"/>
```

2. 启动应用。

   启动应用，分为 2 种情形：

   - 本地启动：启动本地开发所完成的 SOFABoot 工程。
   - 云端启动：将本地工程打包后发布到云端启动，详情请参见 云端运行。

3. 配置云端动态配置。

   前往微服务控制台进行动态配置项的创建、管理与推送操作，实现在不重启应用的情况下，完成配置的动态更新。详情参见 新增动态配置 和 推送动态配置。

# 11.3. 新增动态配置

动态配置属性以键值对的形式定义，隶属于某一动态配置类。配置类与属性的关系可类比 Java 中的类与属性的关系。

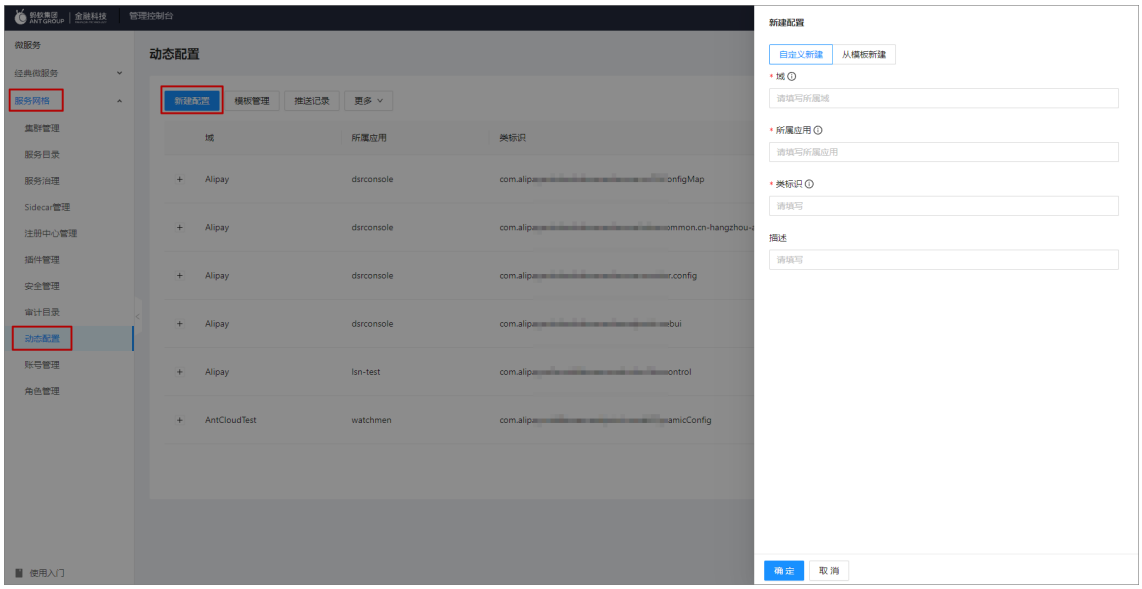
## 配置步骤

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 动态配置**。
3. 单击 **新建配置**，然后配置以下参数：

   您可以通过以下两种方式创建配置：

   - 自定义新建

     单击 **自定义新建**，然后配置以下参数：

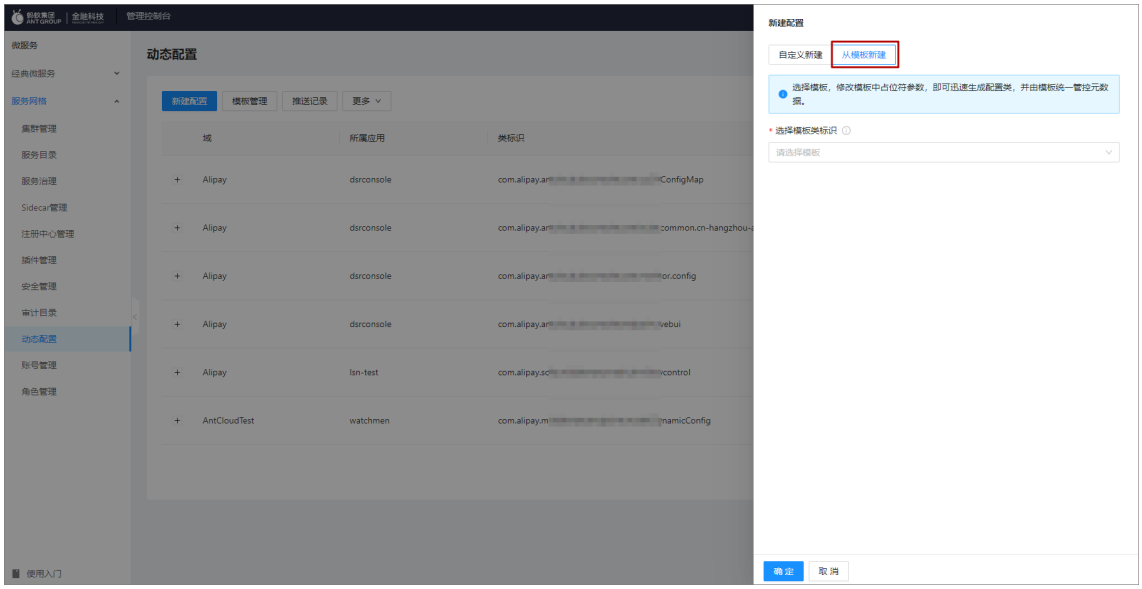



| 参数 | 说明 |
| --- | --- |
| **域** | 填写配置类的命名空间。<br>只能包含大小写字母、数字或者下划线（_）。 |
| **所属应用** | 填写配置类所属的应用名。<br>■ 只能以英文字母开头。<br>■ 可以包含大小写字母、数字、下划线（_）、短划线（-）。 |

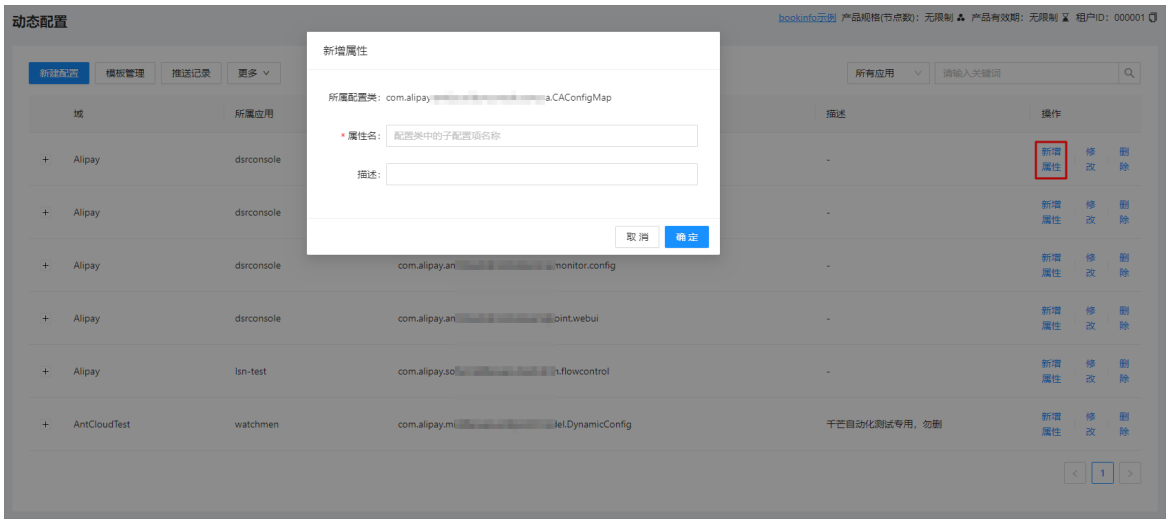
| | |
|---|---|
| **类标识** | 代表配置类的一个字符串，跟应用代码中 `@DObject` 注解的 ID 字段一致，通常使用全类名。<br>■ 只能以英文字母开头。<br>■ 可以包含大小写字母、数字、下划线（_）、短划线（-）、英文句号（.）。 |
| **描述**（可选） | 填写配置类的描述信息。 |

- 从模板新建

  单击 **从模板新建**，然后根据模板要求填写相关参数。




4. 单击 **确定**。
5. 单击刚创建的配置类后侧的 **新增属性**，然后填写 **属性名** 与 **描述**。




属性为 key-value 的形式。具体属性值在推送至服务器时定义。

6. 单击 **确定**。

   属性配置完成后，您可以选择将属性值直接发布到目标服务器上或通过灰度推送进行测试。更多信息，请参见 推送动态配置。

# 11.4. 推送动态配置

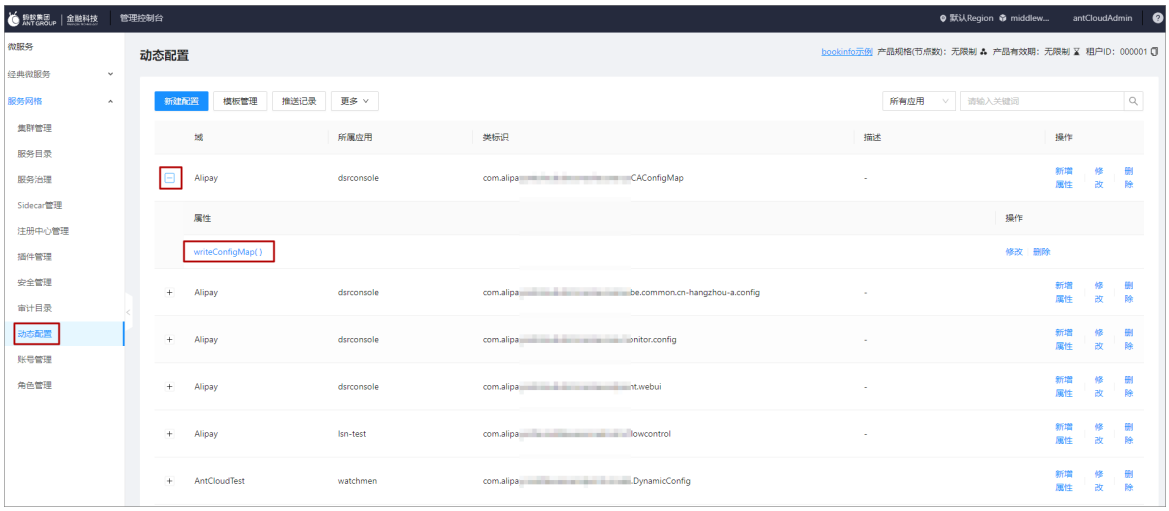
属性配置完成后，您可以根据实际需求推送动态配置。

目前支持以下两种方式推送动态配置：

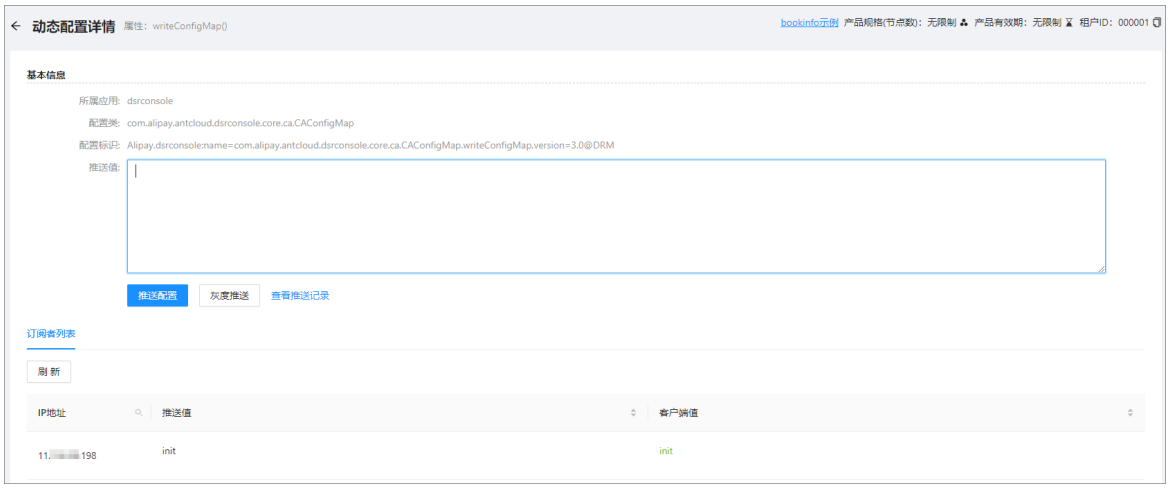
- 直接推送：立即将配置发布至所有订阅服务器。建议在验证配置无误后再进行此操作。
- 灰度推送：仅将配置推送到几台服务器进行测试验证，并不保存数据到数据库。

## 操作步骤

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 动态配置**。
3. 单击目标动态配置左侧的加号（+），然后单击目标属性名称。




4. 配置 **推送值** 后，单击需要的推送方式。




   - 直接推送

     单击 **推送配置**，然后单击 **确定**。动态配置将立即将发布至所有订阅服务器。

- 灰度推送
  a. 单击 **灰度推送**。
  b. 选中需要推送的 IP 后，单击 **推送**。

     只有已经订阅该配置的服务器 IP 地址会显示在弹窗列表中。您可以使用右上角的搜索栏来快速筛选要查找的服务器 IP 地址。灰度推送时，您的配置仅推送到目标服务器进行测试验证，数据不保存到数据库。

## 查看推送记录

单击属性基本信息页面的 **查看推送记录** 链接以查看推送记录列表。

每条推送记录包括以下信息：

推送记录

| 推送时间 | 操作者 | 推送值 | 推送结果 | 操作 |
|---|---|---|---|---|
| 2022-08-31 18:59:45 | admin | init | 成功 | 复用 |
| 2022-08-11 22:32:11 | admin | init | 成功 | 复用 |
| 2022-04-21 23:21:32 | admin | init | 成功 | 复用 |
| 2022-04-21 19:33:40 | admin | init | 成功 | 复用 |
| 2022-04-21 19:01:50 | admin | init | 成功 | 复用 |
| 2022-04-21 18:18:19 | admin | init | 成功 | 复用 |

< 1 >

- 推送时间
- 操作者
- 推送值：本次操作推送的属性值。
- 推送结果：成功或失败。

单击 **复用** 将关闭推送记录窗口并将当前推送记录的推送值填充到推送值文本框中，方便您重新进行推送。

# 11.5. 使用注解标识配置类

动态配置的主要编程方式为使用注解标识配置类信息。本文介绍如何覆盖注解配置，实现更灵活的动态配置类初始化。

> **说明**
>
> 如何完全使用注解方式配置一个动态配置类，请参见 动态配置快速入门。

### 覆盖注解配置方式

动态配置客户端提供两种方式注册配置类：

- 直接注册含有所有注解配置的配置类

```
DistributedResourceManager#register(Object resourceObject);
```

- 注册配置类实例的同时，传入覆盖注解的配置项

```
DistributedResourceManager#register(Object resourceObject, Config config);
```

Config 包含 `@DObject` 中的所有属性，在 Config 中配置的值会覆盖注解配置。

### 属性注解高阶用法

动态配置默认用法是当服务端推送配置后，客户端启动时会默认同步加载服务端配置值。如果您希望服务端配置值仅在运行期生效，或者不希望客户端在启动期同步拉取配置值，可通过 `@DAttribute` 中的 `DependencyLevel` 来定义此属性的依赖等级。

属性依赖等级有以下几种：

| 依赖等级 | 依赖描述 |
| --- | --- |
| NONE | 无依赖，启动期不加载服务端值。启动此级别后，客户端仅会接收在运行期间服务端产生的配置推送。 |
| ASYNC | 异步更新，启动期异步加载服务端值，不关注加载结果。 |
| WEAK | 弱依赖，启动期同步加载服务端推送值。当服务端不可用时不影响应用正常启动；服务端可用后，客户端会依靠心跳检测重新拉取到服务端值。 |
| STRONG | 强依赖，启动期同步加载服务端值。如服务端未设置值，则使用代码初始化值。如从服务端获取数据请求异常或客户端设值异常时，均会抛出异常，应用启动失败。 |
| EAGER | 最强依赖，启动期必须拉取到服务端值。如服务端未推送过值则抛异常，应用启动失败。 |

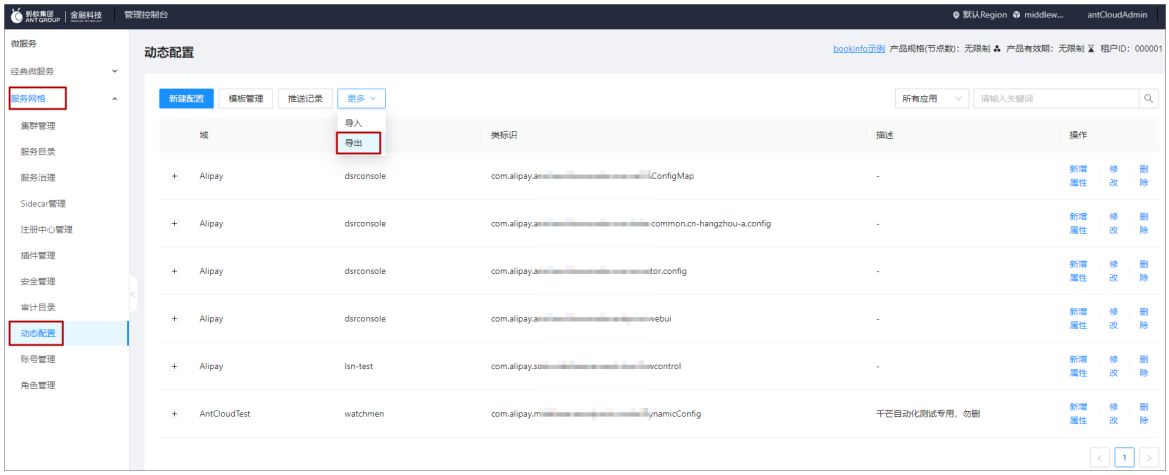
## 11.6. 导出和导入动态配置

动态配置提供配置快速导出和导入功能，您可以通过导出功能将动态配置备份到本地，也可通过导入功能快速恢复动态配置。支持在不同环境中导出和导入动态配置。

### 导出动态配置

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 动态配置**。
3. 在 **动态配置** 页面，选择 **更多 > 导出**。

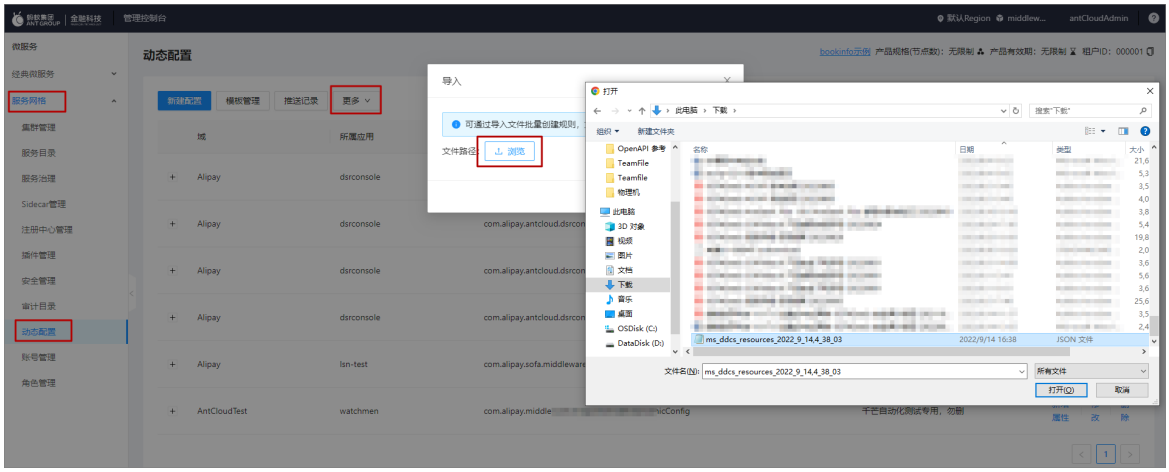
   动态配置文件为 JSON 格式，存放在浏览器默认下载文件夹中。




## 导入动态配置

1. 在 **动态配置** 页面，选择 **更多 > 导入**。
2. 单击 **浏览**，选中目标文件后单击 **打开**。

   支持导入 JSON 和 TXT 格式的动态配置文件。




## 动态配置文件格式

动态配置文件每一行对应一个完整的配置类 JSON 结构，配置类中可包含多个属性，多个配置类的 JSON 数据以换行符分隔。内容格式如下：

```
{
    "region":"Alipay",
    "appName":"testModel",
    "name":"测试配置",
    "resourceId":"com.alipay.test",
    "attributes":[
        {
            "attributeName":"age",
            "name":"年龄"
        },
        {
            "attributeName":"name",
            "name":"名称"
        }
    ]
}
```

参数说明如下：

| 参数 | 说明 |
| --- | --- |
| region | 配置类的命名空间。 |
| appName | 配置类所属的应用名。 |
| resourceId | 代表配置类的一个字符串，跟应用代码中 `@DObject` 注解的 ID 字段一致，通常使用全类名。 |
| name | 配置类的描述。 |
| attributes | 配置类的属性：<br>• attributeName：属性名。<br>• name：属性描述。 |

配置文件示例：

```
[
    {
        "region":"Alipay",
        "appName":"testModel1",
        "name":"配置类描述",
        "resourceId":"com.alipay.test",
        "attributes":[
            {
                "attributeName":"age",
                "name":"属性描述"
            },
            {
                "attributeName":"name",
                "name":"属性描述"
            }
        ]
    },
    {
        "region":"Alipay",
        "appName":"testModel2",
        "name":"配置类描述",
        "resourceId":"com.alipay.test",
        "attributes":[
            {
                "attributeName":"age",
                "name":"属性描述"
            },
            {
                "attributeName":"name",
                "name":"属性描述"
            }
        ]
    }
]
```

# 11.7. 教程示例：使用动态配置

本文通过一个示例介绍动态配置的业务编码和日志排查。

## 准备工作

动态配置的示例是基于 SOFABoot 开发的。学习本课程前，确保您对 SOFABoot 有一定程度的了解。

- 已搭建 SOFABoot 基础环境。操作步骤，请参见 SOFABoot 环境搭建。
- 如果您在线下有联调环境，想在本地编译调试，需要了解 SOFABoot 项目如何编译运行。详情请参见 SOFABoot 编译运行。
- 如果您需要在服务器上部署示例代码，请参见 SOFABoot 应用发布。
- 下载 动态配置的示例工程。

## 步骤 1：云端管控

录入示例代码，配置云端管控动态配置类。详情请参见 动态配置快速入门。

## 步骤 2：发布应用

操作步骤，请参见 发布应用。

应用成功发布之后，您可以在动态配置控制台的资源查询页面查看运行代码的 ECS 和这些 ECS 内存中的属性值。

## 步骤 3：推送资源

在动态配置属性详情页输入需修改的值，然后推送配置。此配置会被及时推送至关心此配置的客户端，并修改客户端对应的动态配置属性值。操作步骤，请参见 推送动态配置。

推送配置后，可实时查看对此配置感兴趣的客户端与客户端中此属性的内存值。

## 查看运行结果

您可以通过 SSH 工具登录到应用的 ECS，查看客户端日志和业务日志：

## 动态配置客户端启动日志

日志目录： `/home/admin/drm/drm-boot.log` 。

- 如果没有异常日志，表明动态配置客户端启动正常。
- 根据资源的 ID 进行查找，如示例中的资源 ID 为 `com.antcloud.tutorial.configuration.DynamicConfig` ，就可以通过 `grep com.antcloud.tutorial.configuration.DynamicConfig drm-boot.log` 看到这个资源注册的相关日志。

## 动态配置推送日志

如果客户端启动正常，在推送资源后，您可以在动态配置的推送日志 `/home/admin/drm/drm-monitor.log` 中查找如下信息：

- “ `Receive update command from zdrmdata server` ”表示从服务端收到了更新资源的命令。
- “ `Query data from zdrmdata` ”表示客户端到服务端查询并更新了最新的推送值。

## 业务日志

在代码中，我们还通过 Logger 在 `set` 和 `before/update` 中打印了自己的业务日志，根据 log4j 中的路径，业务日志会打印在 `/home/admin/logs/service/default.log` 中。

## 代码解析

`DynamicConfig` 是一个动态配置类，具有以下特点：

- 是一个普通的 Java 类，符合 Java Bean 的规范。
- 它的属性均具有 `get/set` 方法（没有会导致资源注册失败），例如代码中的 `str` ，称为资源属性。资源属性只允许 `String` 和基本类型。

- 资源类上加上注解 `@DObject`（包名为 `com.alipay.drm.client.api.annotation`），属性需要加上 `@DAttribute` 注解。
- `@BeforeUpdate` 和 `@AfterUpdate` 在每个属性更新前或更新后执行统一的操作，例如打印日志。这两个方法是可选的，只适合做一些非业务主流程的逻辑。
- 业务逻辑需要放在属性的 `set` 方法中执行。
- 注意如果业务逻辑执行耗时很长，最好能够异步处理，避免超时后动态客户端报错 “ `interrupt` ”。

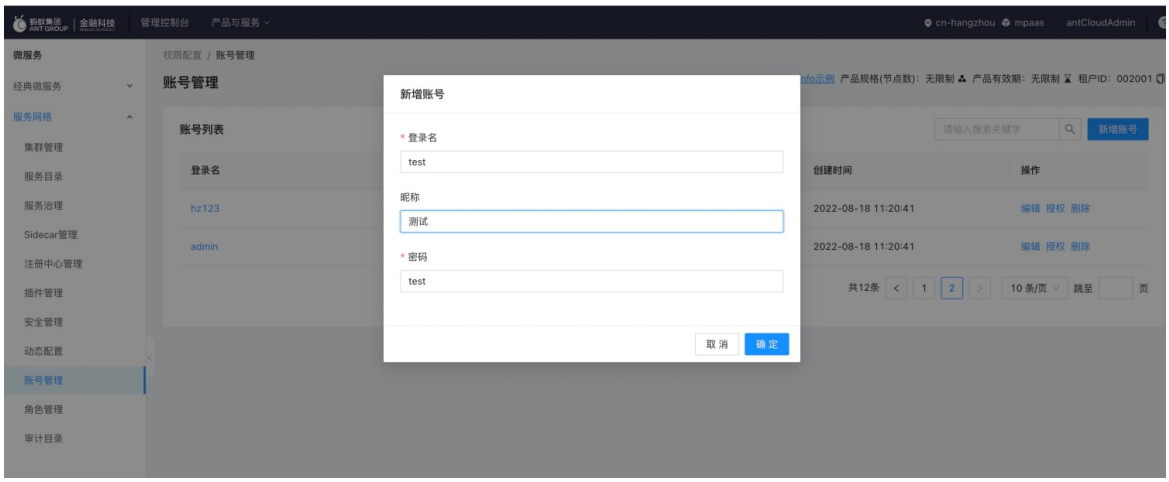
# 12.账号管理

账号管理是服务网格控制台自包含的一套用户管理机制，管理员可以在某个租户下创建账号密码，也可以给账号分配权限，用户使用账号密码即可完成登陆。目前版本只有一个租户和 3 个工作空间（middware、mpass、core）。

## 新增账号

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 账号管理**。
3. 单击 **新增账号**，然后配置账号的 **登录名**、**昵称**、**密码**。

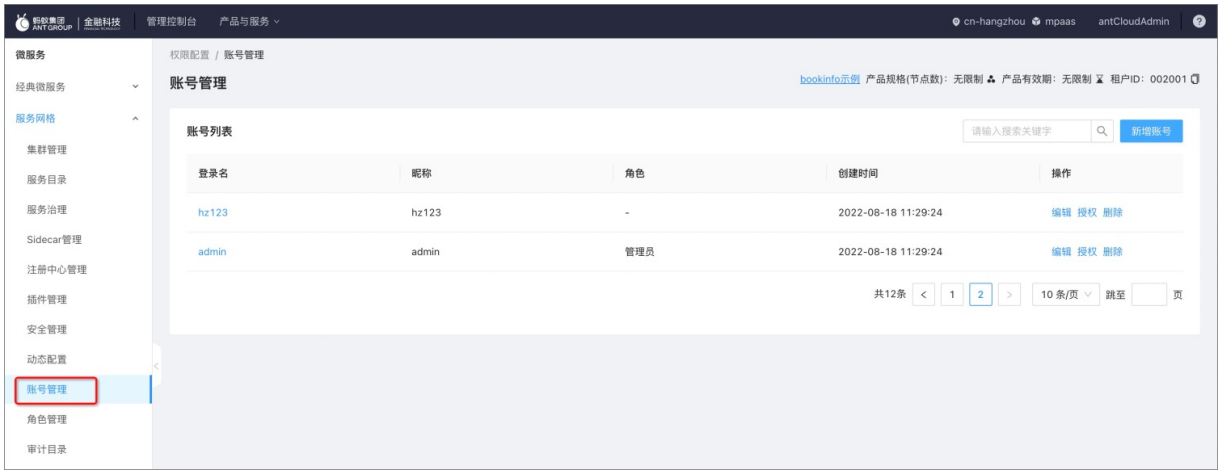
   其中，昵称为可选参数，登陆名不能重复。




4. 单击 **确定**。

## 查看账号列表

通过账号列表可以查看基本信息，如登录名、昵称、给用户授予的角色等信息。




## 为账号授权

单击目标账号右侧的 **授权**，可以为账号授予不同的角色。角色分系统角色和自定义角色：

- 系统角色：包含开发者、管理员、观察者。

- 观察者：只有查询数据的权限，但不能查询角色和账号。
- 开发者：有查询数据、操作功能的权限，但不能查询和操作角色和账号。
- 管理员：有查询、操作所有数据和功能的权限，包括查询和操作角色和账号。

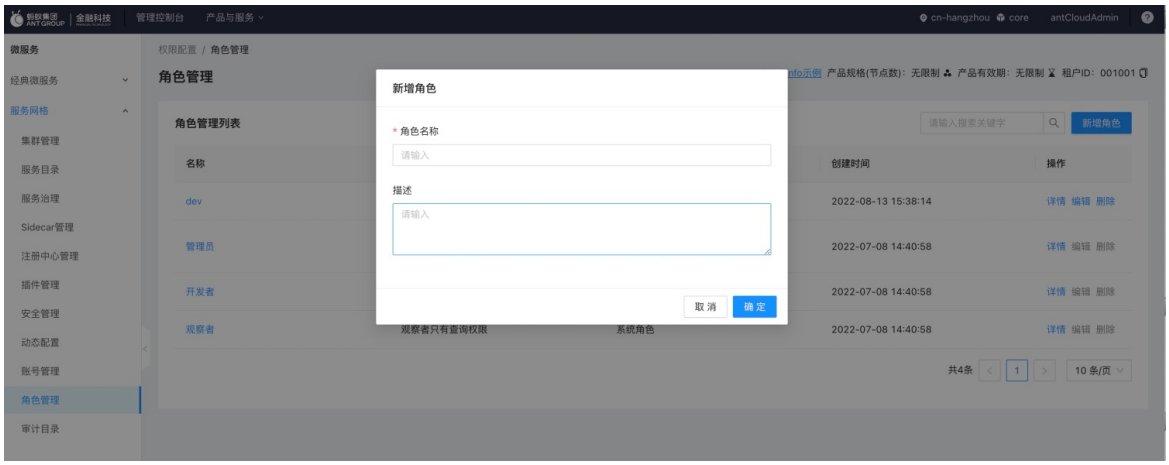
- 自定义角色：请到角色管理页面进行操作。详情请参见 角色管理。

# 13.角色管理

角色管理是服务网格控制台自包含的一套权限管理机制，既可以管理接口粒度的功能权限，也可以管理应用粒度的数据权限。角色是多个权限的集合，管理员首先创建角色，然后可以给角色绑定功能权限和数据权限，还可以将角色授予给某个账号。

## 新增角色

1. 登录微服务控制台。
2. 在左侧导航栏，选择 **服务网格 > 角色管理**。
3. 单击 新增角色，然后输出 **角色名称** 和 **描述**。



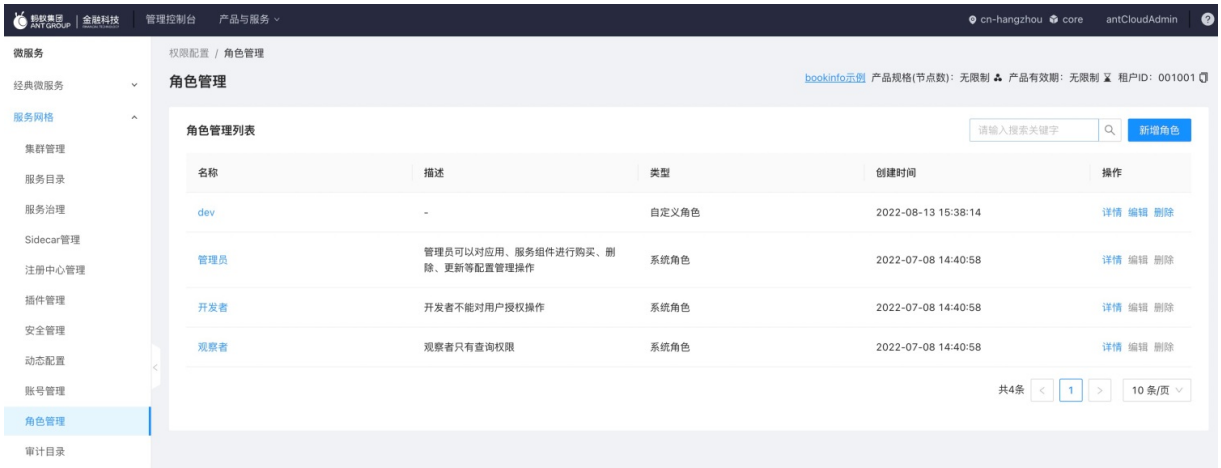

4. 单击 **确定**。

新增的角色类型都是自定义角色。

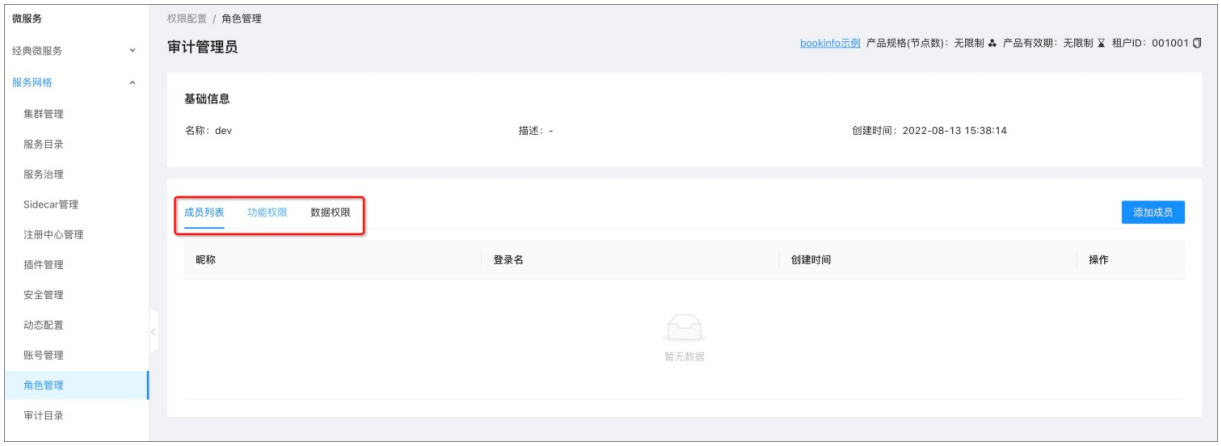
## 查看角色列表

通过角色列表可以查看所有角色基本信息，包含角色名称、描述、角色类型。角色类型分为系统角色和自定义角色。

- 系统角色是系统内置的角色，不可更改和删除。系统角色包含开发者、管理员、观察者：
  - 观察者：只有查询数据的权限，但不能查询角色和账号。
  - 开发者：有查询数据、操作功能的权限，但不能查询和操作角色和账号。
  - 管理员：有查询、操作所有数据和功能的权限，包括查询和操作角色和账号。
- 自定义角色通过新增角色来定义。




单击目标角色名称可查看角色的详细信息。

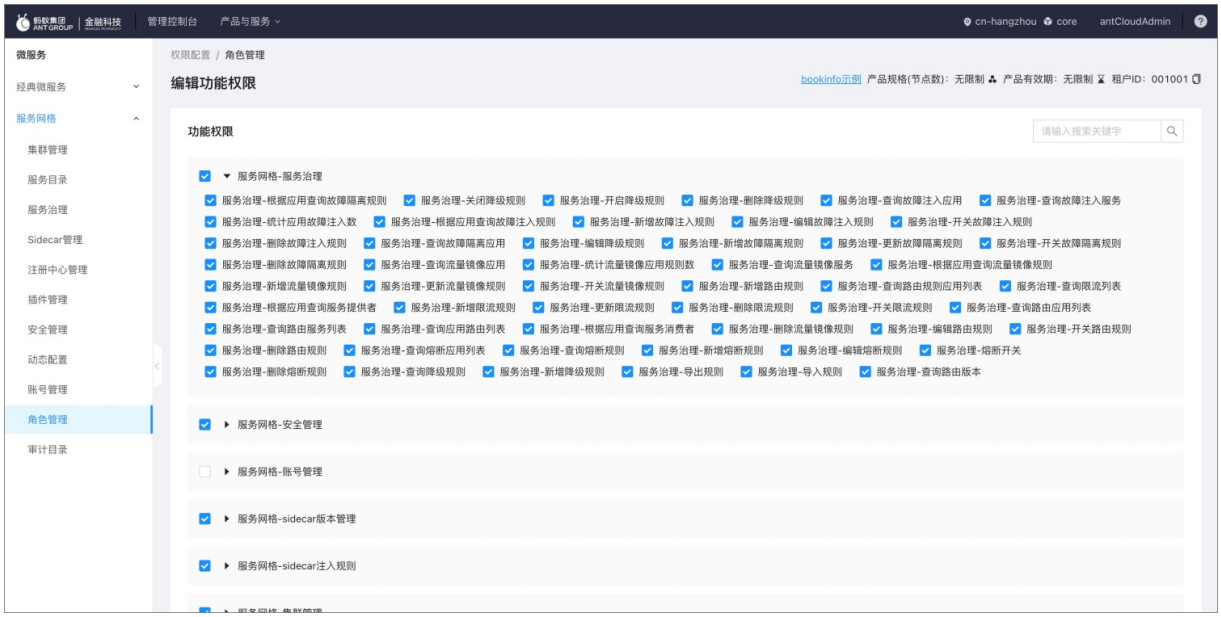



进入角色详情页面可以查看授予了该角色的成员（账号）列表，角色绑定的功能权限和和数据权限，还可以给角色添加成员（账号），绑定和解绑功能权限和数据权限。

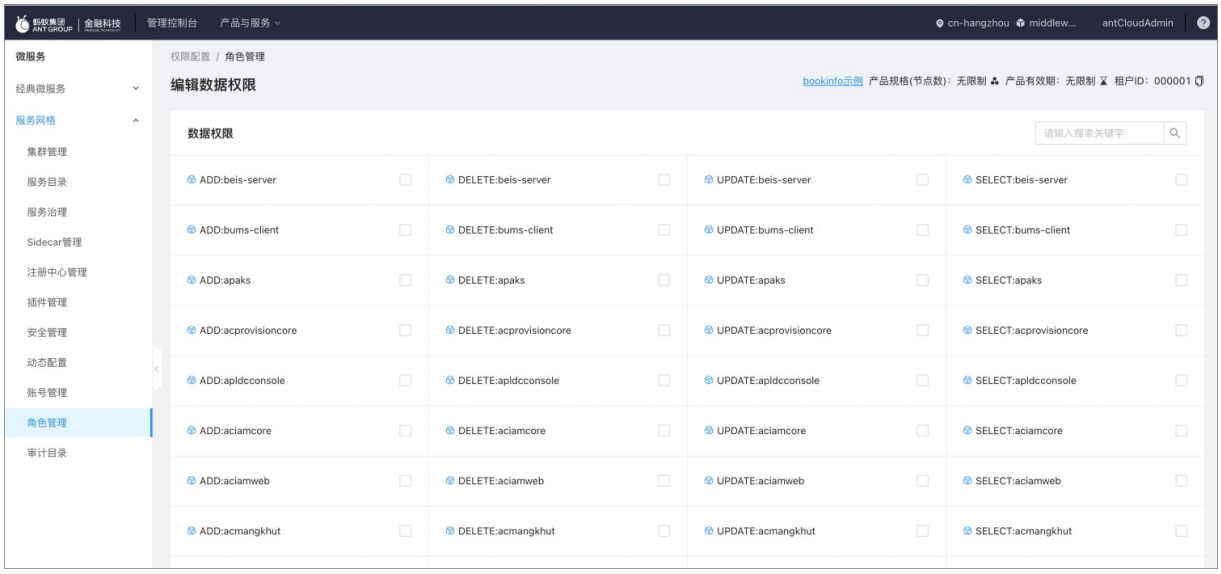
## 编辑功能权限

功能权限是接口粒度，按照接口类别和接口名称进行组织，管理员可选择接口类别或者接口名称。

选择目标权限即可为角色添加对应权限，当选择接口类别时该类别下的所有接口都会被选中。




## 编辑数据权限

数据权限是应用粒度，有应用名称和对应用的操作类型组成。其中应用名称和服务目录应用视角下的应用名称对应。

操作类型有 ADD、DELETE、UPDATE、SELECT 四种类型，分别表示包含应用的创建、删除、更新、查询操作。选择目标权限即可为角色添加对应权限。