# 注册配置治理

# Polaris（北极星）

腾讯云

# 文档目录

Polaris（北极星）
北极星概述
快速入门
引擎管理
引擎管理
标签管理
查看日志
服务管理（注册中心）
命名空间管理
服务管理
服务实例管理
服务实例隔离
服务实例元数据管理
服务别名
关联 K8s 集群
跨地域使用指南
服务治理（治理中心）
服务路由
就近路由
动态路由
熔断降级
服务熔断与主动探测
实例熔断
服务限流
无损上下线
无损上线
无损下线
全链路灰度
联动云原生网关实现全链路灰度
配置管理（配置中心）
配置分组
配置文件
配置轨迹
配置 Agent 安装
可观测性
引擎运行监控
注册配置监控
服务调用监控
监控排障
配置告警
事件中心
操作记录
权限控制（权限中心）
Java 应用开发
Spring Cloud 应用开发
Spring Cloud Tencent 概述
Spring Cloud 版本支持
Spring Cloud 接入
注册发现

配置管理
服务限流
服务熔断
动态路由
全链路灰度
无损上下线
使用 Java Agent
Java Agent 概述
使用 Java Agent 接入
Java Agent 支持无损上下线
Java Agent 支持配置管理
使用 Java SDK
Polaris - Java 接入
注册发现
负载均衡
动态路由
就近访问
服务限流
分布式限流
熔断降级
配置管理
可观测性
二次寻址
Go 应用开发
Polaris-Go 接入
gRPC-Go 接入
二次寻址使用指南
就近访问使用指南
迁移指南
TSF Consul 迁移方案
Nacos 迁移方案
协议兼容接入
Eureka 迁移方案
Eureka协议使用命名空间
从 Eureka 迁移到 Polaris

# Polaris（北极星）
# 北极星概述

最近更新时间：2025-04-17 16:00:52

**腾讯微服务平台（TSF）- 注册配置治理 - Polaris（北极星）用于解决分布式或者微服务架构中的服务注册与发现、故障容错、流量控制和安全问题，提供快速部署、高可用容灾、免运维、一键搭建北极星网格能力。**

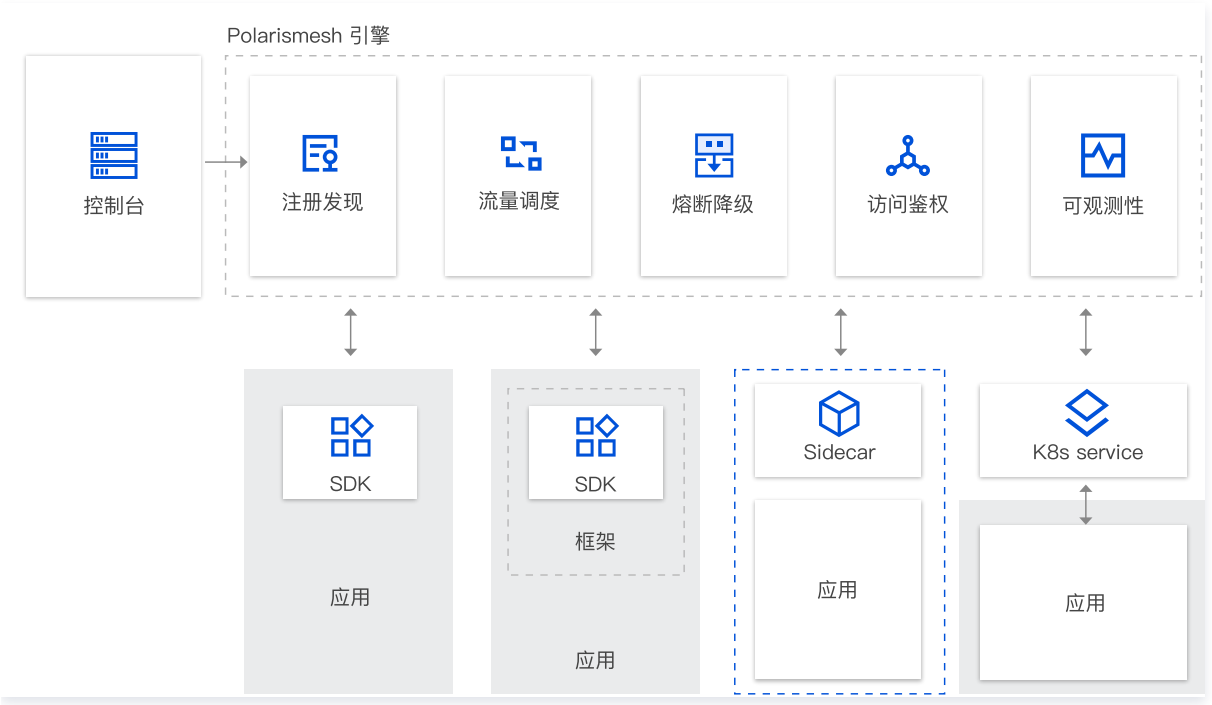
Polaris（中文名北极星，以下简称**北极星**）是腾讯开源的服务发现和治理中心。北极星在腾讯内部的服务注册数量超过百万，日接口调用量超过十万亿，通用性和稳定性都得到了大规模的验证。

## 核心特性

- **拥抱开源**：支持开源主流客户端接入，例如：Spring Cloud、Mesh、gRPC 等。同时，北极星已正式对外开源。
- **服务注册与发现**：高可用、大容量的服务注册中心，整合企业存量服务数据。
- **路由和负载均衡**：动态调度现网流量，实现灰度发布等各种流量调度策略。
- **故障熔断和限流**：自动进行容灾切换和服务降级，保证服务调用的可靠性。
- **可观测性**：数据面上报服务调用统计和运行记录，实现运行状态可观测性。

## 接入方式

**北极星网格为解决异构系统、多语言场景下的服务治理痛点，提供多种接入方式，可根据系统现状自行选择。如下图所示：**




| 接入方式 | 支持的选型 |
| --- | --- |
| SDK | Java / Go 。 |
| 框架 | 提供 Spring Cloud / gRPC 官方集成，其他框架也可以自行集成。 |
| 服务网格 | 通过 polaris-sidecar 接入，兼容 xDS 协议和 envoy。 |
| K8s服务治理 | 支持 K8s Service 自动注册到 Polarismesh，扩展其治理能力。 |

# 快速入门

最近更新时间：2025-06-04 11:46:02

## 操作场景

Polaris（北极星）用于解决分布式或者微服务架构中的服务注册与发现、故障容错、流量控制和安全问题，支持多语言客户端、集成多种主流服务框架，帮助用户实现高效、稳定、高可用、免运维的服务治理能力。
本文介绍在 TSF 控制台创建一个 Polaris（北极星）实例的操作步骤和应用接入的流程，帮助您快速了解如何使用北极星。

## 前提条件

已获取访问授权

## 创建引擎实例

1. 登录 TSF 控制台 。
2. 左侧导航栏单击**注册配置治理中心**下的**Polaris（北极星）**进入北极星实例详情页面，单击**新建**。
3. 在购买页根据自身业务需求选择相关配置。

> **说明：**
> Polaris（北极星）默认支持同城多活高可用架构。

| 参数 | 说明 |
| --- | --- |
| 计费模式 | 支持包年包月和按量付费两种计费方式，如果您的北极星网格使用时间在一个月以上，建议采用预付费（包年包月）模式，具体价格请参见 Polarismesh 价格说明 。 |
| 开源版本 | 支持 1.9.0.1 |
| 产品版本 | • 企业版：企业版适用于生产环境或测试环境。支持多可用区部署。具有服务治理中心全部能力：注册中心+配置中心+治理中心。<br>• 基础版：基础版提供注册中心和配置中心的能力，适用于生产环境或测试环境，如需治理能力，推荐购买企业版。<br>• 开发版：仅支持单节点，1核1G的规格，建议仅用于体验。开发版没有容灾能力。为了保证您业务的稳定性，正式环境强烈建议标准版。<br>• 标准版：可用于测试、生成环境。支持多节点部署。（当前已售罄） |
| 规格 | 根据业务需求选择服务实例配额数量。 |
| 地域 | 选择与您部署业务最靠近的地域。 |
| 集群网络 | 所选择的私有网络必须和业务所在的私有网络一致。所选择的子网不用和业务所在的私有网络一致。 |
| 名称 | 最长60个字符，支持中英文大小写、-、_，名称一旦创建后不支持修改。 |
| 资源标签 | 用于分类管理资源，选填，具体使用方法可参见 标签管理 。 |

4. 单击**立即购买**，在 Polaris 列表页面的状态栏，您可以查看到引擎创建的进度。

## 后续步骤

引擎实例创建完成后，您可以根据您的业务场景选择对应的参考文档体验应用接入的操作流程。

| 应用场景 | 参考文档 |
| --- | --- |
| Java 应用开发 | Spring Cloud 接入 |
| | Java Agent 接入 |
| | polaris-Java 接入 |

| | |
|---|---|
| Go 应用开发 | polaris-Go 接入 |
| | gRPC-Go 接入 |
| 迁移指南 | Eureka 迁移指南 |

# 引擎管理
# 引擎管理

最近更新时间：2025-04-17 16:00:52

## 操作场景

本文介绍通过腾讯微服务平台（TSF）控制台创建、查看、销毁引擎的操作步骤，快速了解 TSF 控制台操作流程。

## 操作步骤

### 创建引擎

1. 登录 TSF 控制台，在左侧导航栏选择 Polaris（北极星）。
2. 在引擎实例列表页，单击新建。
3. 在新建北极星网格页，根据自身业务需求选择相关配置。

| 参数 | 说明 |
| --- | --- |
| 计费模式 | 支持包年包月和按量付费两种计费方式，具体价格请参见 北极星价格说明 。 |
| 开源版本 | 1.17.0.0 |
| 产品版本 | • 企业版：企业版适用于生产环境或测试环境。支持多可用区部署。具有服务治理中心全部能力：注册中心 + 配置中心 + 治理中心。<br>• 基础版：基础版提供注册中心和配置中心的能力，适用于生产环境或测试环境，如需治理能力，推荐购买企业版。<br>• 开发版：开发版仅支持单节点，1核1G的规格，建议仅用于体验。开发版没有容灾能力。为了保证您业务的稳定性，生产环境强烈建议企业版或基础版。<br>• 标准版（已售罄）：适用于生产环境，具有注册中心、配置中心和基础服务治理能力。 |
| 产品规格 | 根据业务需求选择接入节点配额数量。节点数为线上接入北极星的客户端数，即为访问北极星服务端的应用进程个数。接入北极星客户端类型包括 polaris sdk、polaris agent、sidecar、polaris controller 等。 |
| 部署架构 | 支持高可用架构，按您业务实际需要选择同城多可用区和跨城多可用区。 |
| 地域配置 | • 同城多可用区：根据您业务实际的架构规划，选择与您部署业务最靠近的地域。<br>  ○ 注册配置治理-北极星将引擎实例自动部署在多可用区。<br>  ○ 实际部署架构可在引擎创建完成后，单击引擎 ID 进入实例信息页，最下方部署架构版块中，可以看到具体部署架构。<br>• 跨城多可用区：根据您业务实际的架构规划，选择多个地域进行引擎实例组网。<br>  ○ 默认首个地域为集群主地域，主地域不可删除。<br>  ○ 主地域与其他地域在使用上没有区别，均支持数据读写。<br>  ○ 多个地域支持分别配置接入节点规格。<br>  ○ 多个地域支持分别配置接入集群网络。<br>  ○ 最多支持 5 个地域。 |
| 集群网络 | 根据业务网络规划怎么集群网络，所选择的私有网络必须和业务所在的私有网络一致。所选择的子网没有一致性的限制。 |
| 监控与日志 | 选择是否开启事件中心、操作记录、告警能力。<br>⚠ 注意：<br>• PolarisMesh 事件中心、操作记录、告警数据存储在 CLS 日志服务。开启后自动创建日志集 tse_logset，及主题：polarismesh_event、polarismesh_operation、polarismesh_alarm，您在polarismesh控制台上可以查询到这些事件、操作记录和告警配置。<br>• 请注意 CLS 日志服务为独立计费产品，例如：100万条事件和操作记录产生费用约10.29元/月，购买页中不包括这部分费用。具体计费信息查看 CLS 费用详情 。 |

| | |
|---|---|
| 名称 | 最长60个字符，支持中英文大小写、-、_，名称一旦创建后不支持修改。 |
| 资源标签 | 选填，具体使用方法可参见 标签管理 。 |

4. 单击**立即购买**，在 Polaris 列表页面的状态栏，您可以查看到引擎创建的进度。

## 访问方式

单击实例“ID”，在引擎详情页的**实例信息**页面，可以查看到引擎的访问地址。

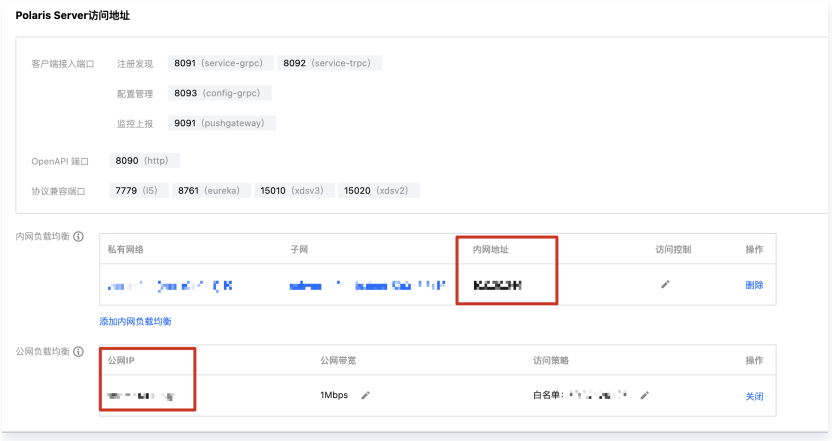
### 控制台访问地址

默认开启公网访问，您可以单击公网地址跳转链接访问 Polarismesh 控制台。公网访问建议设置访问安全策略，您可以通过设置白名单的方式控制公网访问。

> **注意**
> 为了您的业务安全，请尽快登录 Polarismesh 控制台修改初始密码。Polarismesh 控制台的初始登录账号和密码如下：
> - 初始用户名：polaris
> - 初始密码：polaris@实例ID


### 客户端访问地址

客户端访问地址是 Polarismesh 服务端暴露的地址，供用户服务连接到 Polarismesh 进行服务通信、服务配置时使用。客户端访问方式默认提供 VPC 内网访问。公网访问方式可用于开发调试或辅助管理，生产使用建议内网访问方式，避免外网访问的潜在安全风险。若因特殊需要开启公网访问，建议配置不包含0.0.0.0的白名单。




例如 Java 应用，使用 polaris-java sdk 的方式，可在应用的 classpath 当前目录下，添加 polaris.yml 文件，配置服务端地址信息。

```
global:
  serverConnector:
    addresses:
```

```
    - 10.10.0.73:8090
```

### 监控数据上报地址

在客户端配置监控上报地址 上传北极星监控数据。

## 销毁引擎

> **注意：**
> 北极星网格引擎销毁后，服务不可用且数据销毁，该操作不可逆，请谨慎操作。

1. 登录 TSF 控制台 。
2. 在Polaris（北极星）下的 Polaris 页面，找到目标引擎，单击操作列的**删除**。
3. 在二次确认的弹框中单击**删除**，完成北极星引擎销毁。

# 标签管理

最近更新时间：2025-04-17 16:00:52

## 操作场景

该任务指导您为微服务引擎实例绑定标签及修改标签。

## 操作前提

- 拥有腾讯云主账号，且已经开通腾讯云访问管理服务。
- 主账号下至少有一个子账号，且已根据 子账号获取访问授权 完成授权。
- 至少拥有一个**微服务引擎实例**。
- 至少拥有一个**标签**，若您没有，可以前往 标签控制台 > **标签列表**进行新建。

## 操作步骤

### 资源绑定标签

1. 使用**主账号**登录到 TSF 控制台，选择已有的引擎实例并单击进入详情页。


2. 单击进入某个引擎实例的实例信息页，单击标签的编辑图标。


3. 单击添加，选择需要绑定的标签键与标签值，完成为TSE实例绑定标签。


### 编辑标签

1. 登录 TSF 控制台，选择已有的引擎实例并单击进入详情页。
2. 单击进入某个引擎实例的基本信息页，单击标签的编辑图标。


3. 您可以对实例绑定的标签进行添加、删除、修改等操作。


## 附录

更多标签使用方法请参见 **标签** 。
若您需要微服务引擎实例的按标签鉴权能力，请参见 **按标签授权** 。

# 查看日志

最近更新时间：2025-06-11 15:55:02

## 操作场景

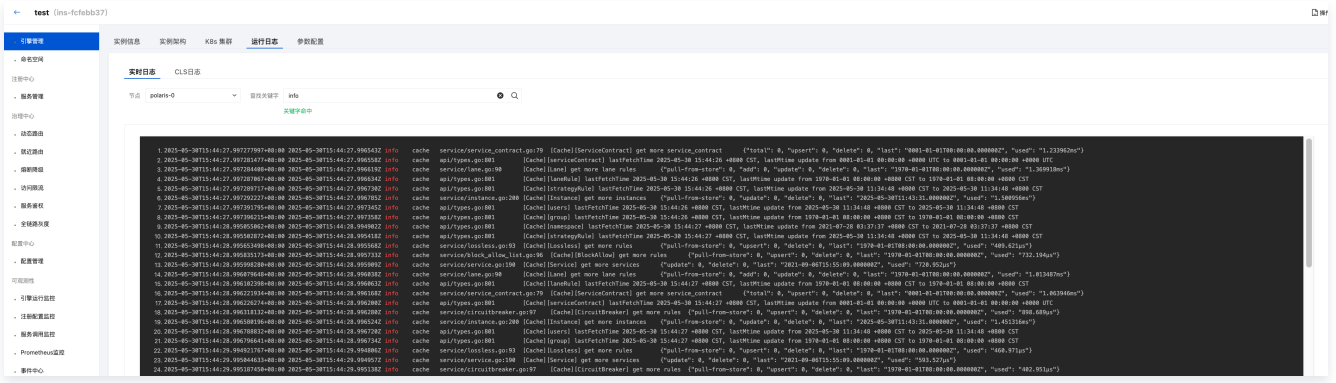
本文为您介绍服务治理中心运行日志的使用说明。用户可以通过节点的运行日志，了解节点的运行状况，定位问题，辅助集群的应用开发和运维。

## 操作步骤

1. 登录 TSF 控制台，左侧导航栏选择 **Polaris（北极星）**进入引擎实例列表页。
2. 在引擎实例列表页面，单击目标实例的“ID”，进入基本信息页面。
3. 在顶部页签单击**运行日志**。
4. **实时日志**

   4.1 选择好**节点**后，可查看对应节点的运行日志。

   4.2 在日志页面的搜索框，可以通过关键字查询相关日志。输入关键词查询，例如：“info”，注意日志检索区分大小写。



5. **CLS日志**

   PolarisMesh 持久化日志、事件中心、操作记录、告警数据存储在 CLS 服务中，开启 CLS 日志服务后可查看。

# 服务管理（注册中心）
# 命名空间管理

最近更新时间：2025-04-17 16:00:53

## 操作场景

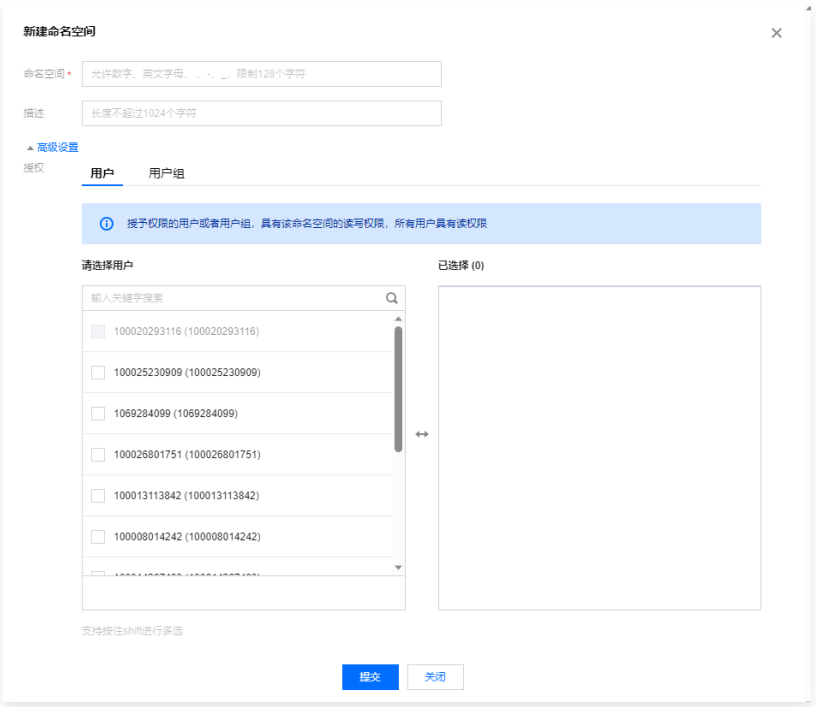
命名空间（Namespace）是对一组资源和对象的抽象集合，用于服务和实例的逻辑隔离。同一命名空间下的资源命名必须唯一，但是跨命名空间允许存在同名的资源。命名空间常用于多个团队或者项目之间的资源区分隔离，如区分环境：开发、测试、预发布、生产环境等；区分业务。
Polaris 默认存在2个命名空间：

- Polaris：Polaris 命名空间存放的是北极星的系统服务，Polaris 自身的集群发现及管理需要依赖 Polaris 命名空间下的服务。
- default：默认命名空间，用户如果没有多命名空间的需求，可以直接使用 default 命名空间。

## 创建命名空间

1. 登录 TSF 控制台，左侧导航栏选择**Polaris（北极星）**，进入引擎实例列表页。
2. 在引擎实例列表页，单击**实例列表**，进入实例信息页面。
3. 单击目标实例引擎操作栏的**控制台**，输入用户名和密码，进入 POLARIS MESH 控制台页面。


4. 在左侧导航栏选择**命名空间**，单击**新建**，填写命名空间名称和描述信息。




5. 单击**提交**，完成命名空间创建。

## 编辑命名空间

1. 在**命名空间**页面，找到目标命名空间，单击操作列的**编辑**。
2. 在编辑命名空间页，修改编辑命名空间的描述信息。
3. 单击**提交**，完成命名空间编辑。

> **说明：**
> 仅支持修改命名空间描述信息，命名空间名称创建后不可修改。

## 删除命名空间

1. 在**命名空间**页面，找到目标命名空间，单击操作列的**删除**。
2. 在删除的确认弹框中，单击**删除**即可删除命名空间。

> **注意：**
> - 命名空间删除后无法恢复，请确保不需要后再删除。
> - 如果命名空间下存在服务，则不可删除。需要先将命名空间下的服务删除后，命名空间才可被删除。

# 服务管理

最近更新时间：2025-04-09 11:24:22

## 操作场景

在 Polaris（北极星）中，每个服务具有唯一的服务名。业务通过服务名来做服务发现。对于一次 RPC 调用来说，是主调服务将请求发送给被调服务。服务实例是服务下注册的具体的业务节点，通常业务会在服务下注册多个节点，有时也被称为节点或 RS。
Polaris（北极星）提供可视化的服务管理界面，服务注册至注册中心后，您可以在腾讯微服务平台控制台查看服务上下线和运行情况，并对服务进行编辑修改等。
该任务指导您通过腾讯微服务平台控制台，如何在某一北极星引擎下进行服务管理。

## 前提条件

- 已 创建北极星网格引擎 。
- 已 创建命名空间 。

## 操作步骤

### 新建服务

#### 场景一

如果您通过如下四种方式接入 Polaris（北极星），我们将提供服务自动注册能力，您无需在控制台手动新建服务，在 TSF 控制台上可直接查看服务详情。

| 接入方式 | 支持类型 |
| --- | --- |
| SDK | 支持 Java、Go。 |
| 框架 | 提供 Spring Cloud 和 gRPC 官方集成，其他框架也可以自行集成。 |
| 服务网格 | 通过 polaris-sidecar 接入，兼容 xDS 协议和 envoy。 |
| K8s 服务治理 | 支持 K8s Service 自动注册到 Polarismesh，扩展其治理能力。 |


如果您需要**服务标签**等能力，则通过**编辑服务**的能力即可实现。

#### 场景二

如果您使用其他方式接入 Polari（北极星），则需要在 TSF 控制台手动**新建服务**与**服务实例**，步骤如下：

1. 登录 腾讯微服务平台控制台，在左侧导航栏下选择 **Polaris（北极星）**，进入北极星实例列表页面。
2. 单击目标实例引擎操作栏的**控制台**，输入用户名和密码，进入 POLARIS MESH 控制台页面。


3. 在左侧导航栏选择**服务管理**，单击**新建**。
4. 在**新建服务**页，根据自身业务需求选择相关信息。


- 命名空间：选择创建好的命名空间。
- 服务名：填写服务名称，允许数字、英文字母、.、-、_，限制128个字符。
- 部门：可选，根据您的实际情况填写。
- 业务：可选，根据您的实际情况填写。
- 服务标签：可选，服务标签可用于标识服务的用处、特征，格式为 key:value。
- 描述：可选，填写服务描述信息。

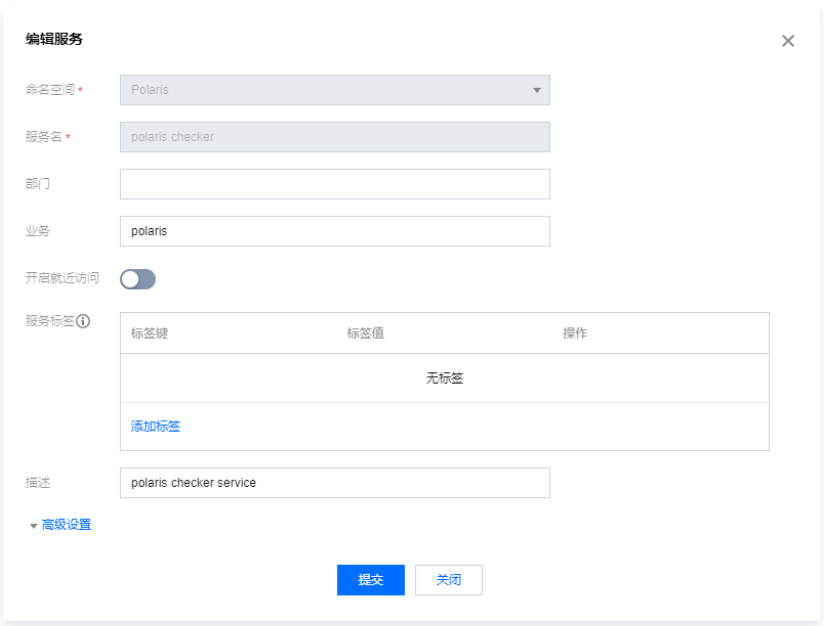
5. 单击**提交**，完成新建服务。
6. 单击下拉按钮，可以查看服务相关属性。


后续操作：服务创建完成后，您需要继续该服务下创建服务实例，参见 服务实例管理 。

## 编辑服务

1. 登录 腾讯微服务平台控制台 ，进入北极星实例列表页。
2. 选择目标引擎实例后，在**北极星**下的**服务列表**页面，选择目标服务，单击操作列的**编辑**。


3. 在**编辑服务**页，根据自身业务需求编辑相关信息。




4. 单击**提交**，完成编辑服务。

## 删除服务

1. 登录 腾讯微服务平台控制台，进入北极星实例列表页。
2. 选择目标引擎实例后，在**北极星**下的**服务列表**页面，选择目标服务，单击操作列的**删除**。


3. 在二次弹窗页面确认删除。

# 服务实例管理

最近更新时间：2025-04-17 16:00:53

## 操作场景

该任务指导您如何通过 TSF 控制台在某一北极星实例下进行服务实例管理。

## 操作前提

- 已 创建北极星引擎实例 。
- 已 创建命名空间 。
- 已 创建服务 。

## 操作步骤

### 新建与查看服务实例

#### 场景一

如果您通过如下四种方式接入 Polaris（北极星），我们将提供服务自动注册能力。您无需**新建服务**与**服务实例**，即查看服务实例详情。

| 接入方式 | 支持类型 |
| --- | --- |
| SDK | 支持 Java、Go。 |
| 框架 | 提供 Spring Cloud 和 gRPC 官方集成，其他框架也可以自行集成。 |
| 服务网格 | 通过 polaris-sidecar 接入，兼容 xDS 协议和 envoy。 |
| K8s 服务治理 | 支持 K8s Service 自动注册到 Polarismesh，扩展其治理能力。 |

步骤如下：

1. 登录 TSF 控制台，进入北极星实例列表页。
2. 选择目标引擎实例后，在**北极星**下的**服务管理**页面，选择目标服务，单击目标服务操作列的控制台按钮。

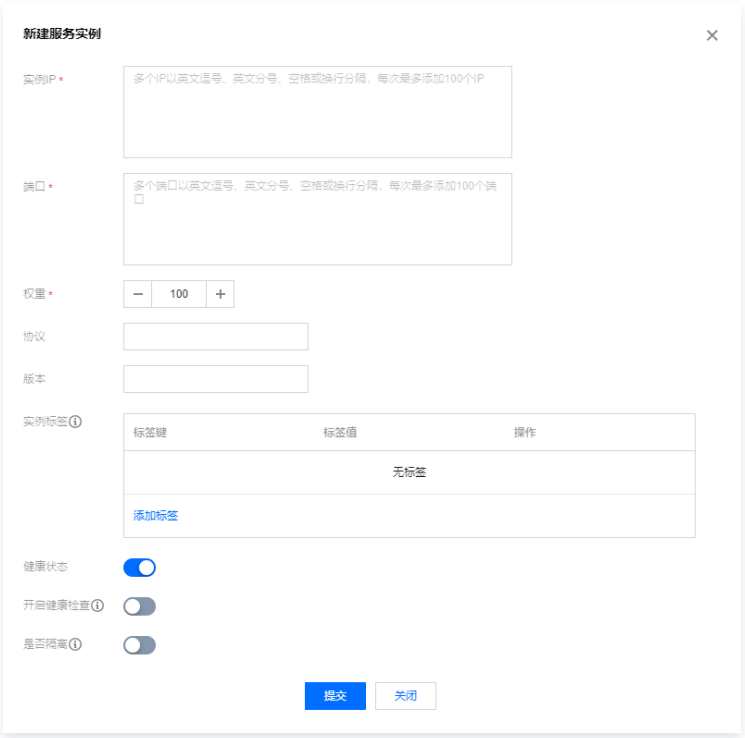

3. 在左侧导航栏选择**服务列表**，选择某一具体服务，单击服务名进入服务实例页。您可以查看服务实例 IP、端口、协议、健康状态、隔离状态等信息。


#### 场景二

如果您使用其他方式接入 Polaris（北极星），则需要在 TSF 控制台手动**新建服务**与**服务实例**，步骤如下：

1. 登录 TSF 控制台，在左侧导航栏选择**Polaris（北极星）**。
2. 选择目标引擎实例后，单击目标实例引擎操作栏的**控制台**，输入用户名和密码，进入 POLARIS MESH 控制台页面。


3. 在左侧导航栏选择**服务列表**，选择某一具体服务，单击服务名进入服务实例页。
4. 在服务实例页面单击**新建**。
5. 在**新建服务实例**页，根据自身业务需求选择相关信息。




   - 实例 IP：填写服务实例 IP。
   - 端口：填写服务实例端口。

     > **注意：**
     > 同一服务下 IP 和端口组合需唯一。

   - 权重：设置实例占据的流量权重。
   - 协议：可选，填写服务实例协议。
   - 版本：可选，填写服务实例版本。
   - 实例标签：可选，实例标签可用于标识实例的用处、特征，格式为 `key:value` 。
   - 健康状态：关闭后，实例状态将显示异常，无法被调用。
   - 开启健康检查：开启后，服务端负责检查服务实例的健康状态。
   - 是否隔离：隔离状态下，主调方无法发现隔离的服务实例，无论实例 IP 是否健康。
6. 单击**提交**，完成新建服务实例。

## 编辑服务实例

1. 在**服务列表**页面，选择某一具体服务，单击服务名进入服务实例页。
2. 在服务实例页面，选择目标服务实例，单击操作列的**编辑**按钮。


3. 在**编辑服务实例**页，根据自身业务需求编辑相关信息。
4. 单击**提交**，完成编辑服务实例。

## 删除服务实例

1. 在**服务列表**页面，选择某一具体服务，单击服务名进入服务实例页。
2. 在服务实例页面，选择目标服务实例，单击操作列的**删除**按钮。

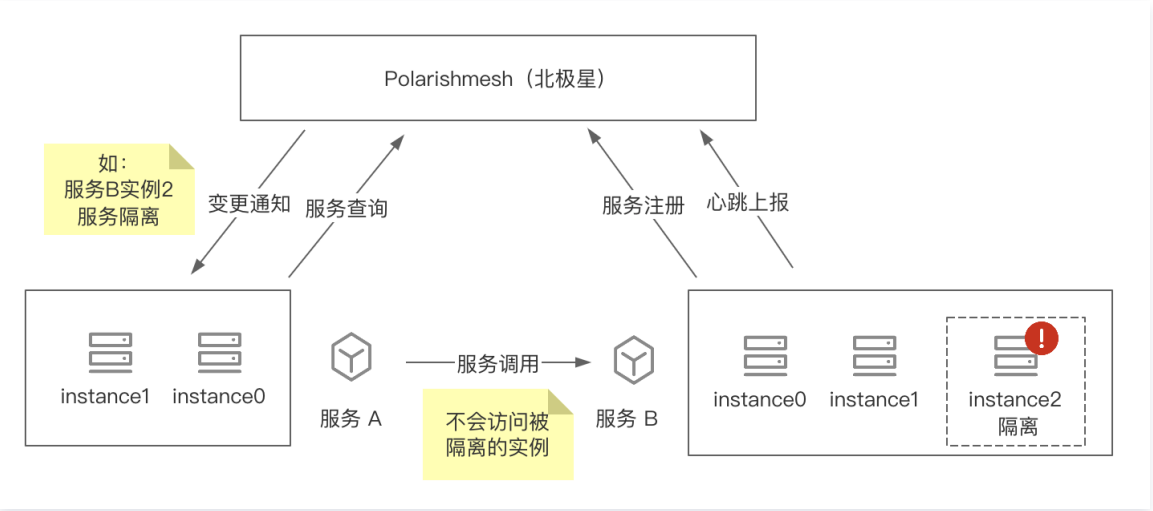

3. 在二次弹窗页面确认删除。

# 服务实例隔离

最近更新时间：2025-04-17 16:00:53

## 操作场景

在微服务场景中，当服务提供者的某些实例出现异常时，一方面，需要避免服务消费者访问到异常实例，另一方面，需要保留异常现场，便于后续的问题排查。北极星服务实例隔离的功能会将异常实例隔离，您可以登录实例排查问题，同时确保服务消费者不会访问到异常实例。在异常恢复后，您可以取消实例隔离，恢复至正常使用。提升您业务的稳定性与质量。
例如在下图的示例场景中，存在服务 A 与服务 B 两个服务，服务 A 为服务消费者，服务 A 调用服务 B。服务B部署了3个实例，并注册至 Polaris（北极星），服务 A 向 Polaris（北极星）查询服务 B 信息。
当服务 B 的实例2发生异常时，会导致服务 A 的部分调用失败，您可以将实例2设置为隔离状态。此时，Polaris（北极星）会将变更通知服务 A，服务 A 更新访问服务 B 的 IP列表，从而实现服务A访问服务 B 时，不会访问到被隔离的实例，以保证服务成功调用。




本文介绍通过腾讯微服务平台控制台使用服务实例隔离的能力。

## 操作步骤

### 隔离服务实例

1. 登录 TSF 控制台，在左侧导航栏选择**Polaris（北极星）**进入北极星引擎实例列表页。
2. 单击目标实例引擎操作栏的**控制台**，输入用户名和密码，进入 POLARIS MESH 控制台页面。


3. 在左侧导航栏选择**服务列表**，选择某一具体服务，单击服务名进入服务实例页。
4. 选择异常实例，单击**其他操作**，单击**修改隔离状态**。


5. **打开**是否隔离的开关，单击**提交**。


返回服务实例页面，可以查看到服务实例的隔离状态已变更为**已隔离**。此时，服务被调用时，不会调度至该被隔离的实例上。


## 取消隔离

当实例正常后，即可取消服务实例隔离。

1. 在**服务实例**列表页面，选择需要恢复的实例，单击**其他操作**，单击**修改隔离状态**。


2. **关闭**是否隔离的开关，单击**提交**。


3. 返回服务实例页面，可以查看到服务实例的隔离状态已变更为**不隔离**。此时，服务被调用时，可以调度至该实例。

# 服务实例元数据管理

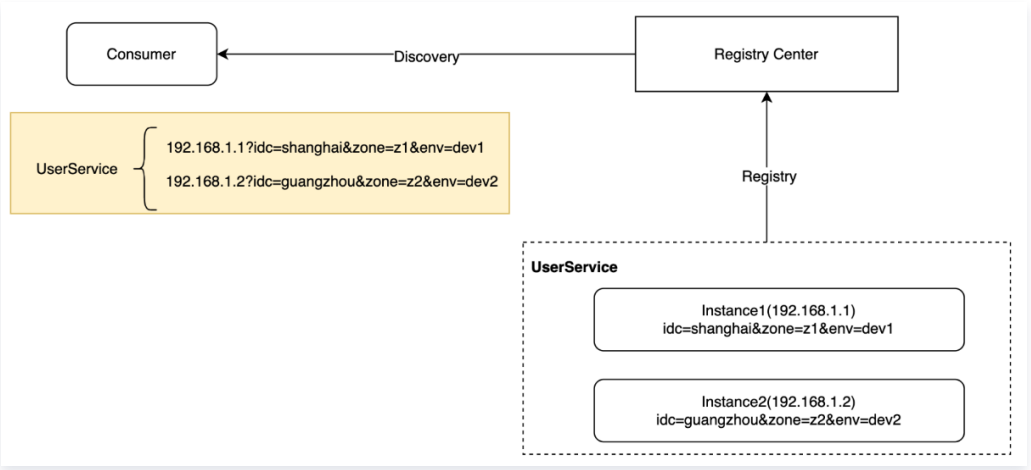
最近更新时间：2025-04-17 16:00:53

## 使用场景

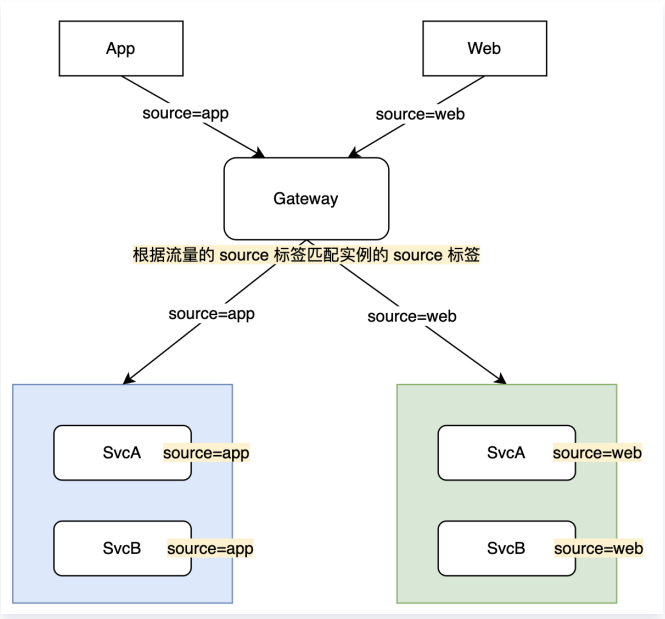
**元数据概念：**服务实例通常带有一系列的标签信息，例如实例所属的机房信息、地域信息、环境信息等，这些信息统称为服务实例的元数据。实例元数据通常也叫实例标签。

**使用场景：**

- 实例注册到注册中心时，会带上自身的元数据信息。当消费方从注册中心获取到实例时，既可以同时获取到每个实例的元信息。




- 通过元数据路由的能力，将指定特征的流量转发至指定特征的服务实例上，可用于灰度发布等场景。




## 操作步骤

### 设置元数据信息

为服务实例设置元数据的方法主要有以下四种，您可根据实际情况选择最适合的方式即可。如您同时通过几种方式设置，则生效优先级为：**自定义SPI > 环境变量 > 启动参数 > 应用配置**。云控制台设置的标签和客户端标签不冲突的情况下同时存在，若 key 相同的情况下，取最新的数据值。

#### 方案1：通过腾讯云控制台设置

1. 登录 TSF 控制台，在左侧菜单栏选择 **Polaris（北极星）**进入引擎实例列表页。
2. 单击选择目标实例后，进入实例详情页，在左侧菜单栏选择**服务管理**，进入服务管理页面。


3. 单击目标服务名，进入服务详情页。单击目标实例的**编辑**按钮，即可添加或修改目标实例的元数据信息。



### 方案2：通过项目配置文件设置

在 Spring Cloud 项目里的 application.yml 中配置元数据信息，以下示例中设置 idc=shanghai，env=dev1的两个元数据。应用在启动注册时，会自动读取配置文件并带上 idc=shanghai 和 env=dev1 两个元数据信息。

```
spring:
  cloud:
    tencent:
      metadata:
        content:
          idc: shanghai
          env: dev1
```

### 方案3：通过应用启动参数设置

Spring Boot/Spring Cloud 应用配置文件定义的配置项都可以通过 -D 启动参数设置，例如通过以下方式覆盖 env 值为 dev2：

```
Java -Dspring.cloud.tencent.metadata.content.env=dev2 -jar demo.jar
```

### 方案4：通过环境变量设置

环境变量的方式完全跟运行的应用解耦。Spring Cloud Tencent 约定了前缀 SCT_METADATA_CONTENT_ 的环境变量为应用的元数据信息。例如：

```
SCT_METADATA_CONTENT_IDC=shanghai
SCT_METADATA_CONTENT_ENV=dev1
```

### 方案5：自定义实现SPI

前面三种方式为 SCT（Spring Cloud Tencent 的缩写）内置的方式，但是并不一定符合每个公司自己的规范或者实现。例如：

- 把元数据放到机器上的某一个配置文件里，例如 /etc/metadata。
- 启动时，调用 CMDB 的接口获取元信息。

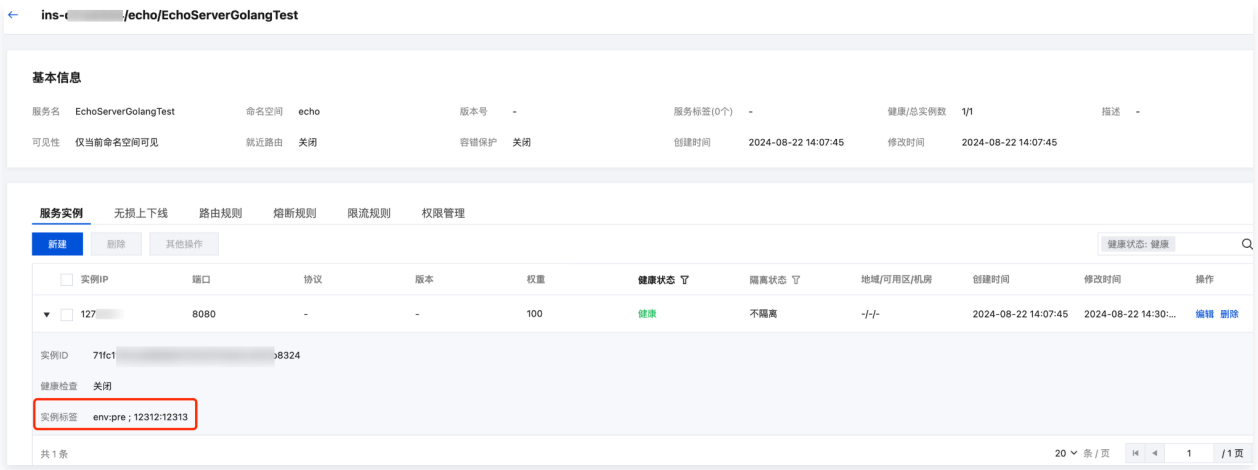
所以 SCT 定义了 InstanceMetadataProvider SPI，支持通过实现内部提供的接口来进行读取，开发者可以实现相关的方法，提供对应的元数据，方便用户自己实现元数据来源。SCT 在注册前，回调 SPI 获取元数据信息，并注册到注册中心。示例代码：

```
@Component
public class CustomMetadata implements InstanceMetadataProvider {

    @Override
    public Map<String, String> getMetadata() {
        Map<String, String> metadata = new HashMap<>();
        metadata.put("k1", "v1");
        return metadata;
    }

    @Override
    public String getZone() {
        return "shenzhen-zone-1";
    }
}
```

## 查看元数据信息

1. 登录 TSF 控制台，在左侧导航栏选择Polaris（北极星）进入北极星引擎实例列表页。
2. 选择目标引擎实例，在引擎实例详情页，进入服务管理页面。


3. 单击目标服务名，进入服务详情页。展开目标实例左侧三角图标，即可查看实例标签（元数据）信息。


ins-[redacted]/echo/EchoServerGolangTest
基本信息
服务名 EchoServerGolangTest
命名空间 echo
版本号 -
服务标签(0个) -
健康/总实例数 1/1
描述 -
可见性 仅当前命名空间可见
就近路由 关闭
容错保护 关闭
创建时间 2024-08-22 14:07:45
修改时间 2024-08-22 14:07:45
服务实例
无损上下线
路由规则
熔断规则
限流规则
权限管理
新建
删除
其他操作
健康状态: 健康
实例IP
端口
协议
版本
权重
健康状态
隔离状态
地域/可用区/机房
创建时间
修改时间
操作
127[redacted]
8080
-
-
100
健康
不隔离
-/-/-
2024-08-22 14:07:45
2024-08-22 14:30:...
编辑 删除
实例ID 71fc1[redacted]8324
健康检查 关闭
实例标签 env:pre ; 12312:12313
共 1 条
20 条 / 页
1
/ 1 页

# 服务别名

最近更新时间：2025-04-09 11:24:22

## 操作场景

服务别名可以看作是服务的映射，访问服务别名等同于访问服务，允许多个服务别名指向一个服务。服务别名主要适用于以下场景：

- 您不想用 service 名作为 polarismesh 服务的名字。例如您希望 polarismesh 的服务名是大写，但是 K8s 的 service 名限制只能小写。这时可以使用别名注解指定一个大写的 polarismesh 服务名。
- 您希望将某个 namespace 下的某个 service 暴露到另外命名空间中。这时可以使用服务别名指定另一个命名空间。
- 对不同的外部客户需要用不同的应用名提供服务，但是实际上后端真实的服务是同一个。

## 操作步骤

### 新建别名

1. 登录 腾讯微服务平台控制台，在左侧菜单栏选择**Polaris（北极星）**进入引擎实例列表页。
2. 单击目标实例引擎操作栏的**控制台**，输入用户名和密码，进入 POLARIS MESH 控制台页面。


3. 在左侧导航栏选择**服务别名**，单击**新建**按钮，根据自身业务需求填写以下配置信息：

| 参数 | 说明 |
| --- | --- |
| 服务别名 | 服务别名为数字、英文字母等字符组成的字符串。 |
| 命名空间 | 选择服务别名所属的命名空间，和映射的服务可以不在同一个命名空间。可根据您的业务需要自行选择。 |
| 指向服务 | 服务别名指向的服务。 |
| 描述 | 您对服务别名的描述。 |

4. 单击**提交**，创建服务别名。


### 查看服务别名

1. 服务别名创建完成后，可以查看服务别名。


2. 单击 图标，可根据命名空间进行过滤。

3. 支持基于命名空间与指向服务的检索。

# 关联 K8s 集群

最近更新时间：2025-05-16 14:47:32

## 操作场景

北极星提供将 K8s 集群关联到 Polaris 的能力，Polaris Controller 可以同步您 Kubernetes 集群上的 Namespace，Service，Endpoints 等资源到 Polaris 中，从而实现 K8s Service 自动注册到 Polaris ，使用 Polaris API 和多语言 SDK 可以访问，使用 gRPC 和 Spring Cloud 等开源框架也可以访问。主要适用于以下场景：

- 异构系统与多技术栈场景下，SpringCloud 等框架服务调用 K8s 集群服务。
- 跨集群场景下的服务调用。

本文介绍通过 TSF 控制台使用 K8s 集群的能力。

## 操作步骤

### 创建引擎

1. 登录 TSF 控制台，在左侧菜单栏中选择 **Polaris（北极星）**。
2. 在引擎实例列表页面，单击目标引擎的“ID”，进入基本信息页面。
3. 在顶部页签单击 **K8s 集群**，单击**关联集群**。

> **说明：**
> 1. 添加容器集群会安装 Polaris-Controller 组件，请确保您的集群至少有 1C2G 的空闲资源，否则会添加失败。
> 2. 容器集群中安装 Polaris-Controller 组件后默认支持 K8s Service 服务同步（按需同步）和 Java Agent 自动注入能力。您也可以在控制台选择**全量同步**使用 k8s 全量同步方式。

- 集群选择：可以根据自身业务需求选择目标关联的 K8s 集群，支持 TKE（容器集群）/ EKS（弹性容器集群）。
- K8s Service 同步：也可以根据业务的特点选择服务的同步方式，支持全量同步（集群内全部 service 自动同步至北极星）/按需同步（仅指定的 namespace 或者 service 自动同步至北极星）。
- Java Agent 接入：容器集群中安装 Polaris-Controller 组件后默认支持 Java Agent 自动注入能力。


## 原理说明

- 在 TSF 控制台关联 tke/eks 集群后，会自动在您对应的 tke/eks 集群中部署 polaris-controller 。具体请参见 资源清单 。
- polaris-controller 默认会同步 K8s 集群所有的 namespace，service 和 endpoints 。具体请参见 polaris-controller 同步行为 。
- 您可以在 K8s 的 service 指定注解，操作 polaris-controller 同步的行为。具体指引和示例请参见 polaris-controller 支持的注解 。

### 资源清单

在您的 tke/eks 集群创建的 K8s 资源清单如下：

| 资源类型 | 资源名 | 资源用途 |
| --- | --- | --- |
| namespace | polaris-system | polaris-controller 和其相关配置都在这个命名空间下 |
| deployment | polaris-controller | 提供同步服务的工作负载 |
| configmap | injector-mesh, polaris-sidecar-injector | polaris-controller 使用的配置文件 |
| serviceAccount | polaris-controller | 提供访问 K8s 资源需要的权限 |
| clusterRole | polaris-controller | 提供访问 K8s 资源需要的权限 |
| clusterRoleBinding | polaris-controller | 提供访问 K8s 资源需要的权限 |
| mutatingWebhookConfiguration | polaris-sidecar-injector | 提供自动注入能力 |

可以使用下面图片中的命令查看到上述的资源。

```
[root@VM-1-7-centos ~]# kubectl get ns | grep polaris-system
polaris-system    Active   16d
[root@VM-1-7-centos ~]# kubectl get deploy -n polaris-system
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
polaris-controller   1/1     1            1           16d
[root@VM-1-7-centos ~]# kubectl get configmap -n polaris-system
NAME                      DATA   AGE
injector-mesh             1      16d
polaris-sidecar-injector  2      16d
[root@VM-1-7-centos ~]# kubectl get serviceaccount -n polaris-system | grep polaris-controller
polaris-controller   1         16d
[root@VM-1-7-centos ~]# kubectl get clusterrole | grep polaris-controller
polaris-controller                                                     2021-11-02T07:57:51Z
[root@VM-1-7-centos ~]# kubectl get clusterrolebinding | grep polaris-controller
polaris-controller                                  ClusterRole/polaris-controller
[root@VM-1-7-centos ~]# kubectl get mutatingWebhookConfiguration  | grep polaris-sidecar-injector
polaris-sidecar-injector            1          16d
```

## polaris-controller 同步行为

polaris-controller 默认会同步 K8s 集群所有的 namespace，service 和 endpoints 的行为。

| 资源类型 | 资源行为 |
| --- | --- |
| namespace | • 创建 namespace 时，会在 polarismesh 创建一个同名的命名空间。<br>• 删除 namespace 时，不会移除 polarismesh 的同名命名空间。 |
| service | • 创建 service 时，不论创建的 service 的类型，都会在 polarismesh 对应的命名空间下创建一个同名的服务。<br>• 删除 service 时，不会移除 polarismesh 对应命名空间下的服务，会移除 polarismesh 中对应服务的实例（只移除本 K8s 集群同步到 polarismesh 对应服务下的实例）。 |
| endpoints | • 某个 service 的 endpoints 创建时，会把 endpoints 列表中 IP 和 port 每个组合，作为 polarismesh 对应服务下的一个服务实例，同步到 polarismesh 。如果 IP 在 endpoints 的就绪地址列表中，则 polarismesh 服务实例状态为健康，如在未就绪地址列表中，则 polarismesh 状态为不健康。<br>• 某个 service 的 endpoints 地址列表变化时，会动态的同步到 polarismesh 。地址列表增加，则在 polarismesh 服务下注册新实例；地址列表减少，则反注册 polarismesh 服务下对应的实例。<br>• 某个 service 的 endpoints 被移除时，本 K8s 集群同步到 polarismesh 对应服务下的实例都会从 polarismesh 的服务中移除。 |

## polaris-controller 支持的注解

您可以在 k8s 的 service 指定注解，操作 polaris-controller 同步的行为，当前支持以下注解。

| 注解名称 | 注解说明 |
| --- | --- |
| polarismesh.cn/sync | 是否同步这个服务到 polarismesh。true 同步，false 不同步，默认不同步。 |
| polarismesh.cn/aliasService | 把 k8s service 同步到 polarismesh 时，同时创建的服务别名的名字。 |
| polarismesh.cn/aliasNamespace | 创建的别名所在的命名空间，配合 polarismesh.cn/aliasService 使用。 |

### 如何按需同步

polaris-controller 默认会同步 K8s 集群所有的 service。某些场景下，您可能只想同步部分 service 到 polarismesh，这时可以将同步方式选择为（按需同步），同时使用 `polarismesh.cn/sync` 注解来指定需要同步的服务。

- 按需同步单个服务：下面的 service 创建时，polaris-controller 会将该服务同步到 polarismesh。

```
apiVersion: v1
kind: Service
metadata:
  name: details
  annotations:
    polarismesh.cn/sync: true
... ...
```

- 按需同步命名空间下所有的服务：为命名空间打上标签，polaris-controller 会将该命名空间下所创建的所有服务同步到 polarismesh。

```
apiVersion: v1
kind: Namespace
metadata:
  name: default
  annotations:
    polarismesh.cn/sync: "true"... ...
```

### 创建别名示例

polaris-controller 默认会以 service 名字，创建一个对应的 polarismesh 服务。可能有以下情况，需要创建服务别名：

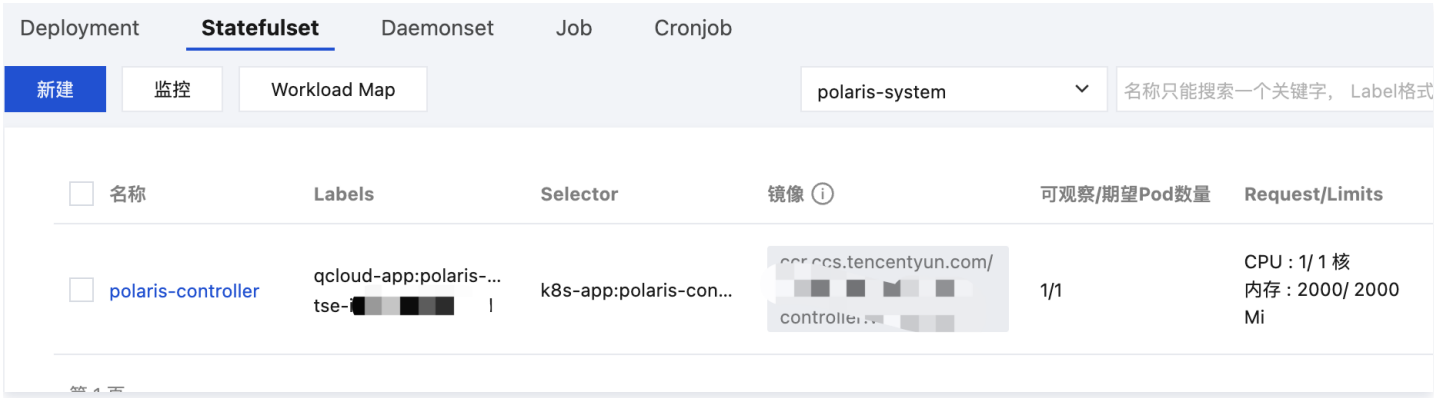
- 您不想用 service 名作为 polarismesh 服务的名字。例如您希望 polarismesh 的服务名是大写，但是 K8s 的 service 名限制只能小写。这时可以使用别名注解指定一个大写的 polarismesh 服务名。
- 您希望将某个 namespace 下的某个 service 暴露到另外命名空间中。这时可以使用别名注解指定另一个命名空间。

下面示例的 service 创建时，polaris-controller 会在 polarismesh 的 development 命名空间下创建一个名为 productpage 的服务。同时也会在 Development 命名空间下创建一个名为 Productpage 的服务别名。

```
apiVersion: v1
kind: Service
metadata:
  namespace: development
  name: productpage
  annotations:
    polarismesh.cn/aliasService: Productpage
    polarismesh.cn/aliasNamespace: Development
... ...
```

## 如何确认polaris-controller版本

点击集群ID，跳转到容器控制台页面，查看polaris-system命名空间下polaris-controller这个statefulset的镜像版本。


Deployment
Statefulset
Daemonset
Job
Cronjob
新建
监控
Workload Map
polaris-system
名称只能搜索一个关键字， Label格式
名称
Labels
Selector
镜像
可观察/期望Pod数量
Request/Limits
polaris-controller
qcloud-app:polaris-...
k8s-app:polaris-con...
1/1
CPU : 1/ 1 核
内存 : 2000/ 2000 Mi

# 跨地域使用指南

最近更新时间：2025-04-17 16:00:53

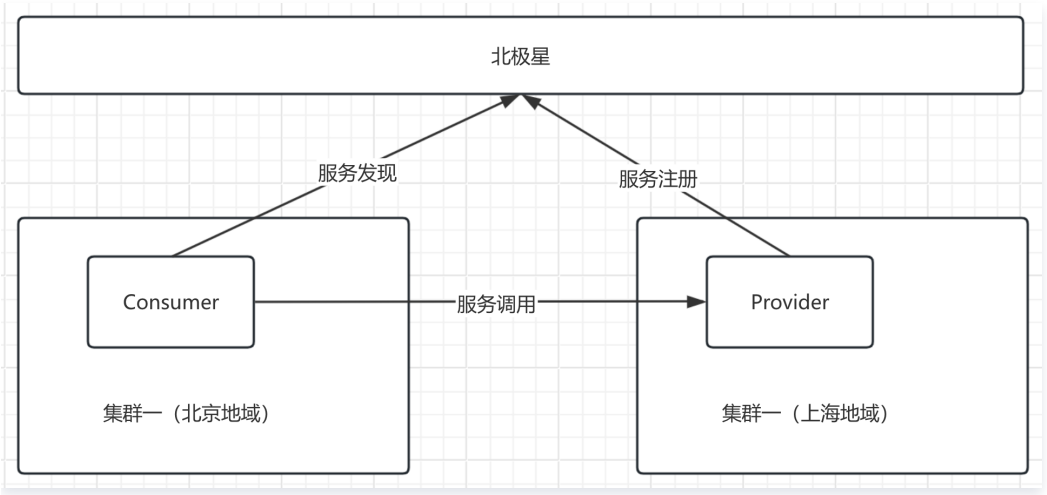
## 功能描述

北极星支持跨地域部署，提供全局注册配置治理中心。支持将多个地域的北极星管控节点组成一套北极星集群，即同大区内跨地域组网。例如：北极星同一个实例中，支持广州节点和北京节点。选择广州作为北极星所在的主地域，多个地域的数据确保一致性。
主地域和从地域对于客户端接入和使用上来说，没有差异，均提供服务注册发现能力。数据最终存储在主地域。

### 使用场景

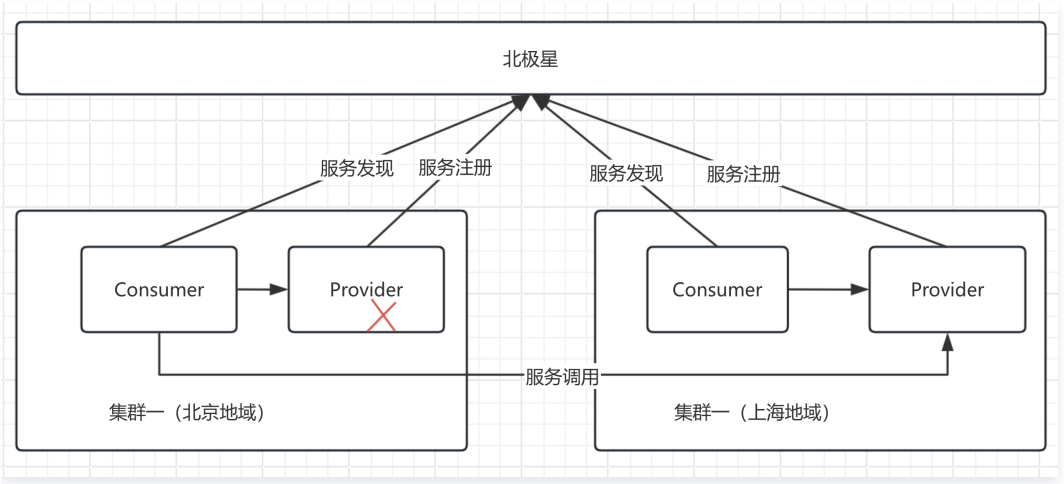
### 场景1：跨地域服务注册发现

北极星提供服务跨地域注册发现，可满足业务服务跨地域调用的场景。在电商、游戏等场景下，业务一般会多个地域部署，以提升系统的稳定性，降低客户访问时间。但是由于某些业务场景的特殊性或者数据同步的复杂性，部分服务无法在多个地域部署，需要跨地域服务调用。例如：某电商应用，北京地域服务访问上海地域的服务。




### 场景2：跨地域容灾场景

北极星支持同城就近路由，跨城容灾。在跨地域容灾场景下，一个服务在两个地域部署节点，正常情况下，就近访问。在某个地域发生故障后，则路由至另一个地域。例如：当广州的 provider 不可用时，consumer 会跨城调用上海的 provider。




## 操作步骤

### 步骤1：跨地域节点组网

1. 登录 TSF 控制台，在左侧菜单栏选择Polaris（北极星）后进入引擎实例列表页。
2. 单击**新建**，创建实例，详情参见 引擎管理 。

部署架构选择**跨城多可用区**，选择跨地域的目标地域，并配置每个地域集群的客户端接入节点数、集群网络信息。


## 步骤2：服务注册发现

| 应用场景 | 参考文档 |
| --- | --- |
| Java 应用开发 | polaris-Java 接入 |
| | gRPC-Java 接入 |
| | Spring Cloud 接入 |
| Go 应用开发 | polaris-Go 接入 |
| | gRPC-Go 接入 |
| | Dubbo-Go 接入 |
| 网关接入 | Nginx 接入 |
| 迁移指南 | Eureka 迁移指南 |

# 服务治理（治理中心）
# 服务路由
# 就近路由

最近更新时间：2025-04-09 11:43:32

## 操作场景

本文将帮助您快速了解如何使用Polaris（北极星）的就近路由能力。

## 前提条件

已创建 PolarisMesh 北极星网格，请参见 创建 Polaris（北极星）治理中心 。

## 操作步骤

使用北极星就近路由的能力，主要包括两部分：

- 在控制台开启就近路由能力。
- 在客户端通过腾讯云 API 或者环境变量获取地域信息。

### 步骤1：在控制台开启就近路由能力

1. 登录 腾讯微服务平台控制台，在左侧导航栏选择**Polaris（北极星）**，进入引擎实例列表页。
2. 单机目标服务实例在左侧导航栏，**治理中心**下，单击**服务管理**，上方下拉框选择目标北极星实例。
3. 单击**目标服务**，单击**路由规则** Tab，选择**就近路由**后开启**就近路由**按钮。

### 步骤2：在客户端通过腾讯云API或者环境变量获取地域信息

详细参见开发文档：Java SDK 开发 > 就近访问 。

# 动态路由

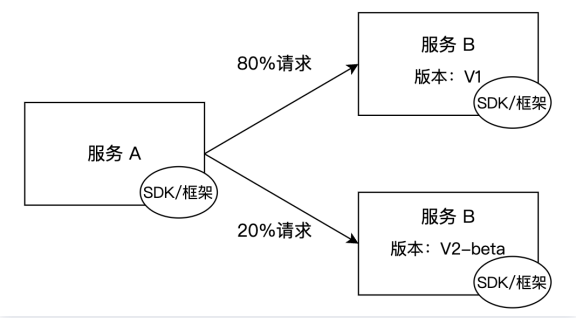
最近更新时间：2025-04-09 11:43:32

## 概述

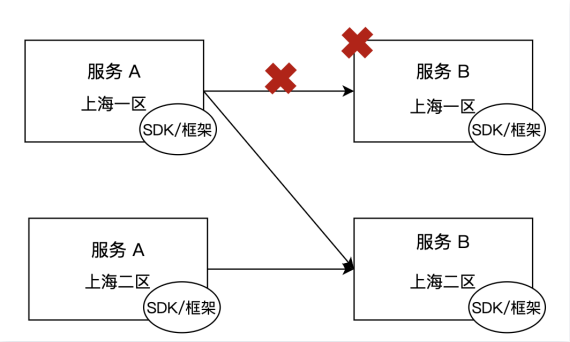
动态路由功能是指用户根据路由规则控制流量分发的机制，将满足规则的流量转发至目标实例分组。该功能通常用于灰度发布、容灾降级等场景。为了满足客户的定制化需求，北极星支持用户为服务实例打上自定义标签，定向分配流量。 总而言之，服务路由功能的主要作用是将调用流量按照自己的需求进行分配。

## 应用场景

- **灰度发布场景**：在微服务的场景下，用户研发新版本上线的迭代周期越来越快，稳定敏捷的上线新版本需要微服务框架能够支持灰度发布、金丝雀发布、滚动发布等发布方式。通过服务路由功能，用户可以配置流量分配权重，设置灰度特性的流量被分配到某个版本号中。通过路由功能，用户也可以为指定灰度用户的请求转发至灰度版本上。路由能力为灰度发布等上线模式提供了无需终止服务的底层能力支持。



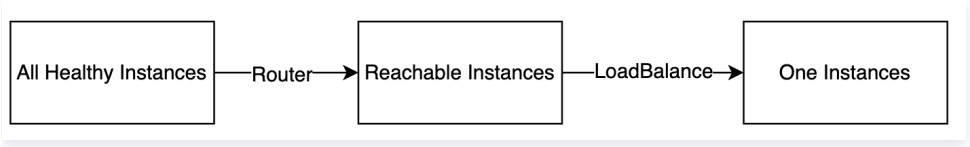

- **容灾场景**：业务为了应用系统整体的健壮性以及高可用，通常会跨可用区、跨地域进行应用部署。服务路由提供配置目标服务优先级的能力，支持在某可用区或者某地域目标服务发生故障后，将请求路由至次优先级的实例分组上。助力用户更加便捷的实现多活方案。




## 实现原理

当 A 服务调用 B 服务时，先从注册中心获取全量 B 服务地址信息。当没有服务路由，直接进行负载均衡，根据负载均衡算法从全量 B 服务地址中挑选一个服务实例发起服务调用。 当加入服务路由阶段后，挑选服务实例分为两个阶段：

- 阶段一：从全量服务地址中根据路由规则选取一批目标服务实例地址。
- 阶段二：从阶段一选取的一批目标服务实例地址中，再根据负载均衡算法挑选一个实例。




## 使用说明

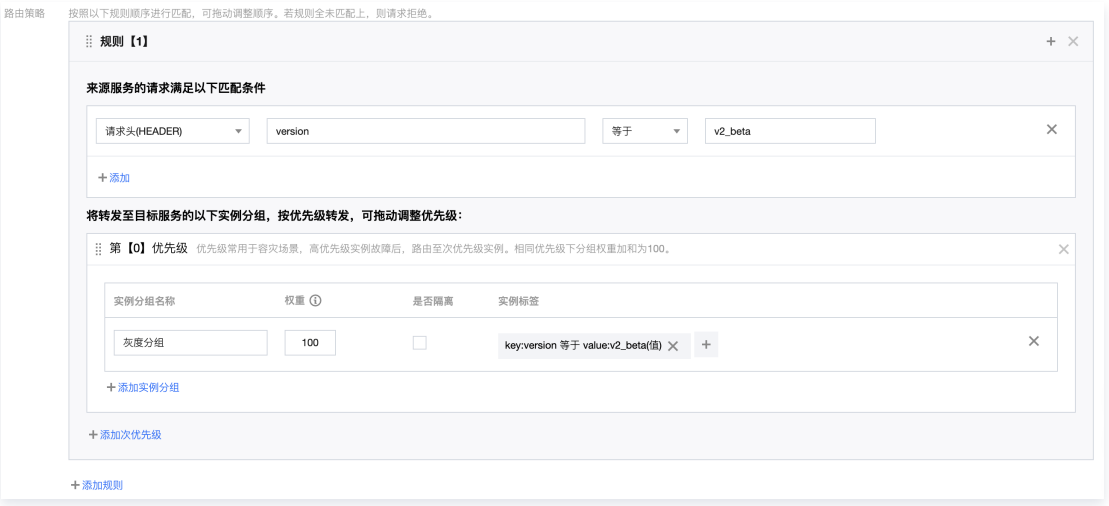
要实现服务路由需要完成两部分操作：

- 在控制台上，设置路由规则。
- 客户端获取路由规则， 根据规则来分发请求。

### 步骤1：设置路由规则

1. 登录 腾讯微服务平台控制台 。
2. 在左侧导航栏，选择 Polaris（北极星），在实例详情页选择目标引擎实例。
3. 单击左侧导航栏**动态路由**，进入动态路由规则展示页面，单击**新建路由规则**。


- 规则名称：填写路由规则名称，不超过64个字符。
- 优先级：优先级数字设置越小，规则匹配顺序越靠前。
- 匹配条件：配置路由规则的主调服务（服务消费者）和被调方（服务提供者），其中主调服务支持选择全部命名空间和全部服务。您可按需选择。




- 路由策略支持配置多条，按顺序进行匹配。可拖动调整顺序。若规则全未匹配上，则请求拒绝。
- 流量匹配策略，非必选。
  - 流量参数位置支持：请求头(Header)、请求Cookie、请求参数(Query)、方法(Method)、主调IP、路径(Path)、自定义。
  - 逻辑关系支持：等于、不等于、包含、不包含、正则表达式、范围表达式。
  - 流量无差别，按百分比进行灰度的场景下，删除流量匹配策略即可。
- 实例分组权重：
  - 相同优先级下，实例分组权重加和为100。
  - 流量百分比场景下，可为不同实例分组配置不同流量百分比。如下图所示，为灰度版本分配10%的流量，90%的流量路由至基线版本。


- 是否隔离：隔离通常用于现网故障或者上线验证场景，先手动摘除，待故障恢复或者验证通过后解除隔离。
- 优先级：优先级常用于容灾场景，高优先级实例故障后，路由至次优先级实例。

4. 单击**提交**，完成新建规则。

## 步骤2：客户端开发添加相关逻辑

根据您对应的接入方式查看开发文档下**动态路由**相关的篇章，以使用 Spring Cloud 接入为例，可参见文档 Spring Cloud 应用开发 > 动态路由 。

# 熔断降级
# 服务熔断与主动探测

最近更新时间：2025-04-17 16:00:53

## 场景说明

Polaris（北极星）支持可视化熔断规则管理，支持设置服务、实例、API 三种隔离级别的熔断规则。**在常见的熔断器的基础之上，北极星还额外支持故障主动探测。**

## 熔断原理

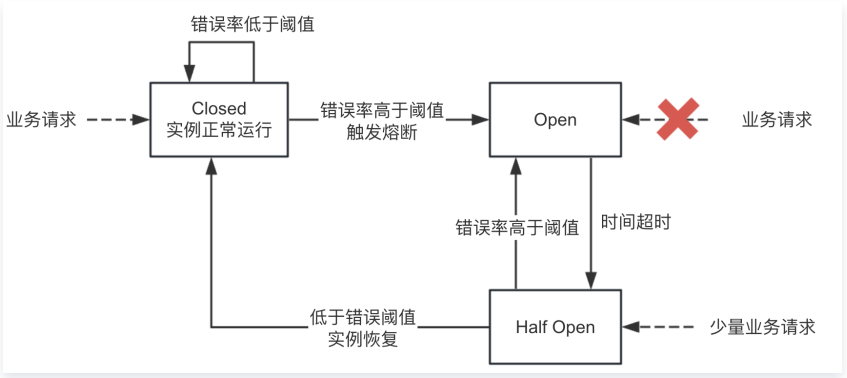
### 定义

**服务熔断定义：当下游的服务因为某种原因导致服务不可用或响应过慢时，上游服务为了保证自己整体服务的可用性，不再继续调用目标服务，直接返回。当下游服务恢复后，上游服务会恢复调用。**

### 熔断器状态

**服务熔断中涉及到关键概念熔断器。**

1. **主动探测能力关闭的情况下，熔断器的状态转化如下：**




1.1 最开始处于 `closed` 状态，一旦检测到错误或慢响应等达到一定阈值，便转为 `open` 状态，此时不再调用下游目标服务。

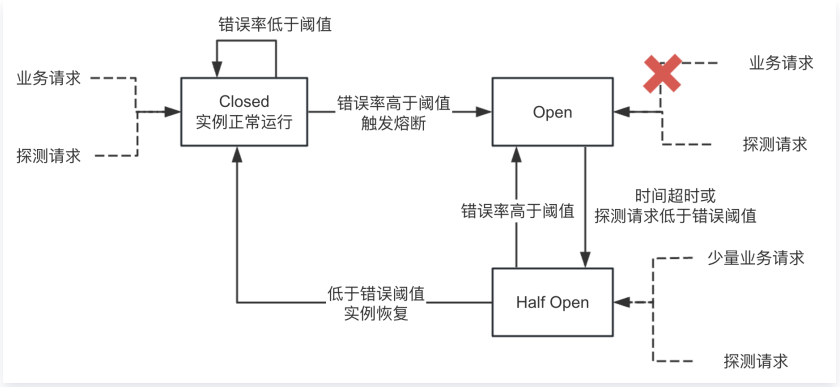
1.2 等待一段时间后，会转化为 `half open` 状态，尝试放行一部分请求到下游服务。

1.3 一旦检测到响应成功，回归到 `closed` 状态，也即恢复服务；否则回到 `open` 状态。

其中熔断器从 `close` 变为 `open` 状态要**同时**满足以下2个条件：

- 前提条件：在滑动时间窗口内至少有一定数量的请求（即**最少请求数**）
- 指标达到阈值：在滑动时间窗口内统计的错误请求率或慢请求率达到一定阈值

2. **打开主动探测能力，断路器的状态流转如下：**




在业务请求的同时，加上主动探测的请求。在没有业务请求时，也能及时了解业务运行状态。

## 隔离级别及场景

北极星支持服务、实例、接口 三种隔离级别的熔断规则：

| 隔离级别 | 请求统计范围 | 熔断对象 | 适用场景 |
| --- | --- | --- | --- |
| 服务 | 下游目标服务的所有实例的所有 API | 服务 | 当下游服务属于不主要的业务，可以熔断所有实例 |
| 实例 | 下游目标服务的单个实例的所有 API | 达到熔断触发条件的实例 | 当下游服务属于比较重要的业务，只对异常的实例进行熔断，避免所有实例被熔断后导致服务雪崩 |
| 接口 | 下游目标服务的所有实例的指定 API | 达到熔断触发条件的单个接口 | 下游服务不同 API 的重要程度不同，需要根据不同 API 设置不同的熔断策略 |

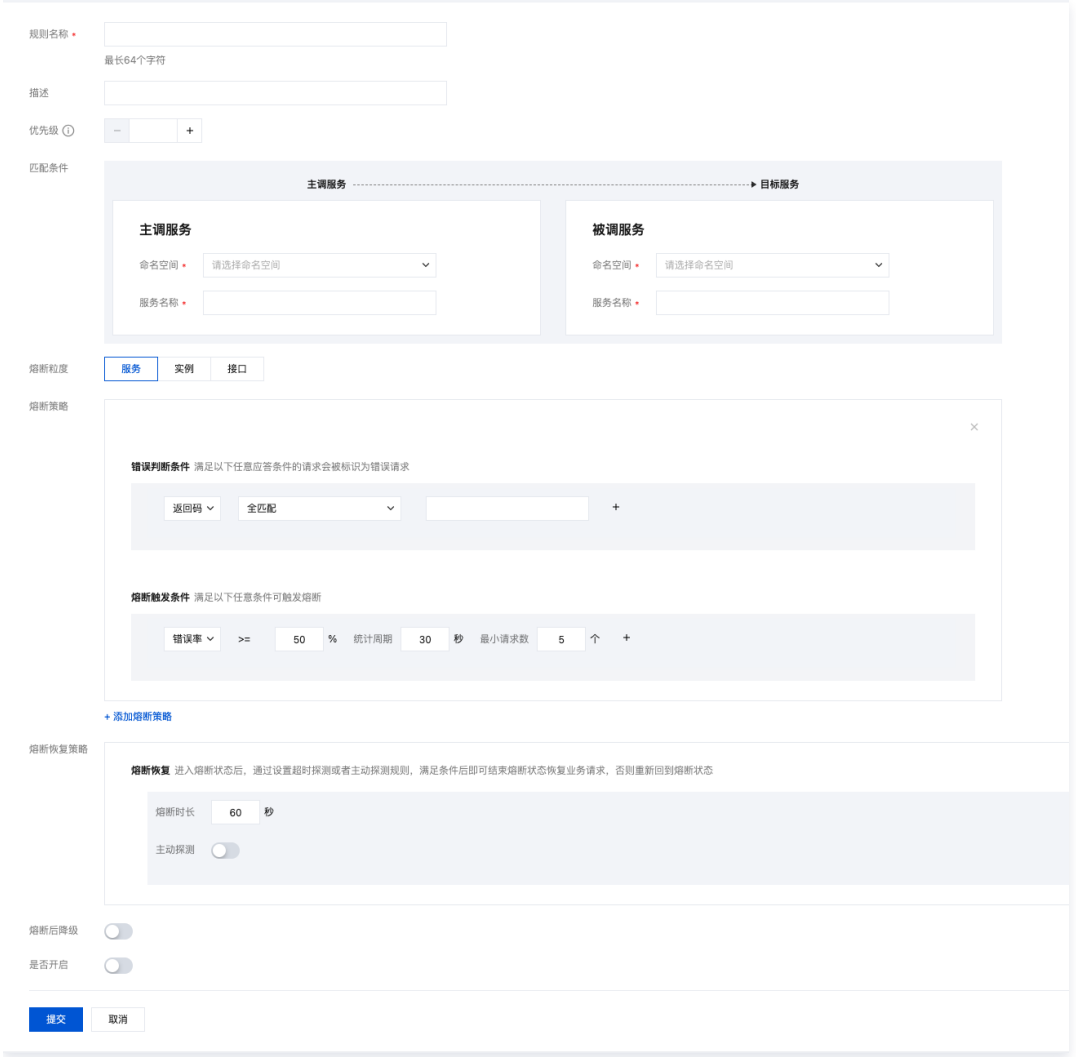
## 使用说明

要实现服务熔断需要完成两部分操作：

- 在控制台上，设置熔断规则、主动探测规则。
- 客户端接入熔断能力。

### 步骤1：设置熔断规则

1. 登录 TSF 控制台 。
2. 在左侧导航栏，选择 Polaris（北极星），在实例详情页选择目标引擎实例。
3. 单击左侧导航栏**熔断降级**，进入熔断规则展示页面，单击**新建熔断规则**。

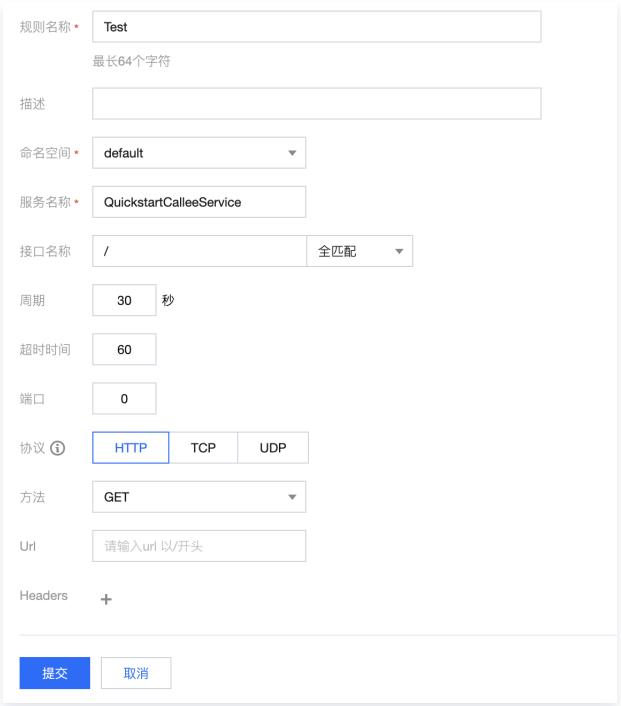



- 规则名称：规则名，需全局唯一，最长64个字符。

- 描述：规则的描述信息，用于补充规则的说明。
- 匹配条件主要用于决定熔断规则的适用范围，客户端根据匹配条件来过滤本地调用所适用的熔断规则。
  - 主调服务：配置作为主调方的服务名和命名空间，可选。不配置则默认对所有的主调生效。
  - 被调服务：配置被调的服务名和命名空间，可选。不配置则默认对所有的被调生效。
  - 被调接口：配置被调的接口名，表明该熔断规则只对调用某个接口的请求生效，可选。接口名支持多种匹配条件。
- 错误判断条件：可配置多个判断条件，满足任意一个条件的请求会被标识为错误请求。支持**返回码**和**时延**2种判断方式。
- 熔断触发条件：可配置多个触发条件，满足任意一个条件即会触发熔断，资源会进入熔断状态。支持**连续错误数**和**错误率**2种触发条件。
  - 连续错误数：统计调用该服务/接口的请求连续错误数，达到阈值即触发熔断。
  - 错误率：统计在周期内调用该服务/接口的请求的错误率，达到阈值即触发熔断。同时为避免少流量下的放大效应，可配置错误率统计的起始请求阈值，请求数超过阈值才进行熔断判断。
- 熔断恢复：资源触发熔断后，会熔断请求调度到该资源一段时间。随后系统会对资源进行恢复尝试，当满足一定次数的连续成功请求数后，资源会恢复正常状态。用户可配置资源熔断时长（单位秒），以及连续成功请求数。
- 可选择开启主动探测。开启后，主调方会根据被调服务/接口所关联的探测规则，向被调方发起探测，探测结果会与业务调用合并，作为熔断触发或恢复的依据之一。开启主动探测后需配置主动探测规则。

## 步骤2：设置主动探测规则（可选）

1. 登录 TSF 控制台 。
2. 在左侧导航栏，单击**服务治理中心 > 流量管理**，进入流量管理页面。
3. 在页面上方选择**熔断降级**页签，选择**主动探测** Tab。
4. 单击**新建主动探测规则**。




- 规则名称：必选。规则名，需全局唯一。
- 描述：可选。规则的描述信息，用于补充规则的说明。
- 命名空间：必选。被调服务的命名空间。
- 服务名称：必选。被调服务的名称。
- 接口名称：可选。被调接口的名称，支持多种匹配条件。
- 周期：可选。探测周期，多久执行一次探测，默认30秒。
- 超时时间：可选，探测最大超时时间，超时未响应则以最大超时时间作为熔断依据。默认60秒。
- 端口：可选，探测端口，默认为服务实例端口。
- 协议：使用什么协议进行探测，会影响接下来的配置，支持 HTTP、TCP、UDP 3种协议。
  - HTTP 协议配置：

- 方法：HTTP 方法，默认 Get。
- Url：探测 Url，默认/。
- Headers：发送探测包所需的消息头，默认空。

- TCP 协议配置：TCP 默认会采用 tcp connect 的方式进行探测，用户也可以配置基于报文的探测手段。
  - Send：配置所需发送的TCP二进制报文，格式为0x开头的十六进制字符串，例如：0x12ab。
  - Receive：配置所接收的TCP二进制报文，可配置多个，不配置则不校验应答。
- UDP 协议配置：UDP 没有连接检测的方式，只能通过基于报文的探测手段。
  - Send：配置所需发送的 UDP 二进制报文，格式为0x开头的十六进制字符串，例如：0x12ab。
  - Receive：配置所接收的 UDP 二进制报文，可配置多个。

### 步骤3：客户端接入熔断能力

根据您对应的接入方式查看开发文档下**熔断降级**相关的篇章，以使用 Spring Cloud 接入为例，可参见文档 Spring Cloud 应用开发 > 服务熔断 。

# 实例熔断

最近更新时间：2025-04-17 16:00:54

## 功能说明

实例级熔断规则用于针对特定分组下多个服务实例或者单个服务实例进行熔断。

## 使用说明

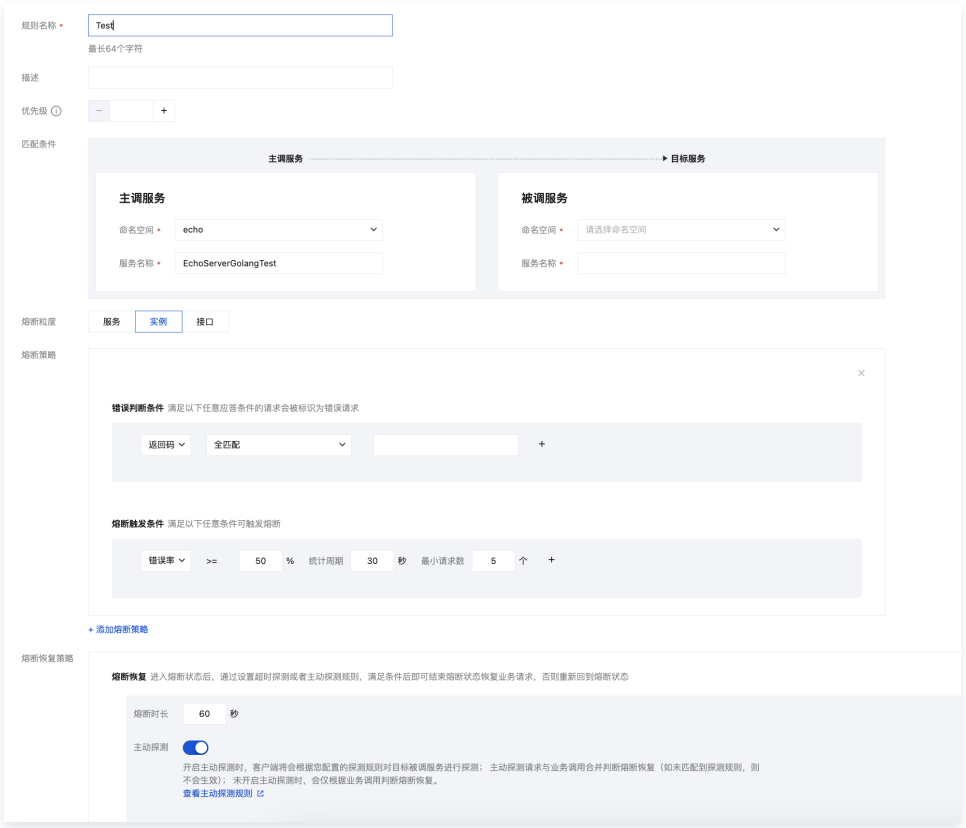
要实现服务熔断需要完成两部分操作：

- 步骤1：在控制台设置熔断规则、主动探测规则。
- 步骤2：客户端接入熔断能力。

详细操作指引可参见下文。

## 步骤1：设置熔断规则

1. 登录 TSF 控制台 。
2. 在左侧导航栏，选择 Polaris（北极星），在实例详情页选择目标引擎实例。
3. 单击左侧导航栏**熔断降级**，进入熔断规则展示页面，单击**新建熔断规则**。




界面各字段含义如下：

基础信息：

- **规则名称**：规则名，需全局唯一，必填，不超过64字符。
- **描述**：规则的描述信息，用于补充规则的说明，非必填。
- **优先级**：数字越小，优先级越大。
- **匹配条件**：匹配条件主要用于决定熔断规则的适用范围，客户端根据匹配条件来过滤本地调用所适用的熔断规则。
  - 主调服务：配置作为主调方的服务名和命名空间，可选。不配置则默认对所有的主调生效。
  - 被调服务：配置被调的服务名和命名空间，可选。不配置则默认对所有的被调生效。
  - 另外，熔断规则中所填的服务名，可以是任意服务名，不一定需要存在于北极星注册中心。
- **熔断粒度**：选择**实例**。

熔断配置：

- **熔断策略**：
  - **错误判断条件**：可配置多个判断条件，满足任意一个条件的请求会被标识为错误请求。支持返回码和时延2种判断方式。
  - **熔断触发条件**：可配置多个触发条件，满足任意一个条件即会触发熔断，资源会进入熔断状态。支持连续错误数和错误率2种触发条件。
  - **连续错误数**：统计调用该实例的请求连续错误数，达到阈值即触发熔断。
  - **错误率**：统计在周期内调用该v的请求的错误率，达到阈值即触发熔断。同时为避免少流量下的放大效应，可配置错误率统计的起始请求阈值，请求数超过阈值才进行熔断判断。
- **熔断恢复策略**：
  - **熔断恢复**：资源触发熔断后，会熔断请求调度到该资源一段时间。随后系统会对资源进行恢复尝试，当满足一定次数的连续成功请求数后，资源会恢复正常状态。用户可配置资源熔断时长（单位秒），以及连续成功请求数。
  - **主动探测**：可选择开启主动探测。开启后，主调方会根据被调服务/接口所关联的探测规则，向被调方发起探测，探测结果会与业务调用合并，作为熔断触发或恢复的依据之一。
  - **是否开启**：选择是否开启该规则。

## 步骤2：客户端接入熔断能力

根据您对应的接入方式查看开发文档下**熔断降级**相关的篇章，以使用 Spring Cloud 接入为例，可参见文档 Spring Cloud 应用开发 > 服务熔断 。

# 服务限流

最近更新时间：2025-06-11 15:55:02

## 场景说明

服务限流主要是保护服务节点或者数据节点，防止瞬时流量过大造成服务和数据崩溃，导致服务不可用。当资源成为瓶颈时，服务需要对请求做限流，启动流控保护机制。

限流的原理是监控服务流量的 QPS 指标，当达到指定的阈值时进行流量控制，避免被瞬时高峰流量冲垮，从而确保服务的高可用。限流模式有两种：

- 单机限流：通过统计单机 QPS 指标，当达到规则指定阈值时对流量进行限制，保障服务实例不被瞬时流量给冲垮。
- 分布式限流：通过统计全局 QPS 指标，当达到规则指定阈值时对流量进行限制，保障服务实例不被瞬时流量给冲垮。

## 操作步骤

1. 登录 TSF 控制台 。
2. 在左侧导航栏，选择 **Polaris（北极星）**，在实例详情页选择目标引擎实例。
3. 单击左侧导航栏 **访问限流**，进入熔断规则展示页面，单击**新建限流规则**，在弹窗中配置好限流规则后，单击**提交**。

**单机限流**

| 参数 | 是否必填 | 说明 |
| --- | --- | --- |
| 接口路径 | 否 | 指定限流规则的接口过滤参数，接口名可对应方法名、HTTP、URL 等信息，不填写代表不过滤。<br>• 接口名称：规则生效所对应的接口名，用于匹配客户端传入的 method 参数，默认为空（全部）。<br>• 匹配方式：接口字段的匹配方式，支持全匹配、不等于、包含、不包含、正则表达式五种匹配模式。 |
| 请求匹配规则 | 否 | 指定限流规则的请求参数匹配条件，不填代表不过滤，支持以下四种参数类型：<br>• 自定义参数：自定义 KEY 和 VALUE，具体的请求参数值可通过 SDK 进行传入。<br>• 请求头（HEADER）：针对协议消息头（http header/grpc header）进行过滤。<br>• 请求参数（QUERY）：针对协议请求参数（http query）进行过滤。<br>• 方法（METHOD）：针对协议的 METHOD（http method/grpc method）进行过滤。<br>• 主调服务：针对微服务调用场景下，主调方的服务名进行过滤。<br>• 主调 IP：针对主调方机器的 IP 地址进行过滤。<br>• 主调实例元数据：通过主调实例标签进行过滤。<br>每种类型参数值支持以下几种值匹配模式：<br>• 全匹配：全值匹配，传入的值与配置的值相同才匹配通过。<br>• 正则表达式：用户配置正则表达式，通过正则表达式对传入的值进行匹配，正则表达式支持 Google RE2 标准 。<br>• 不等于：取反匹配，传入的值与所配置的值不相等才算匹配成功。<br>• 包含：多字符串取 OR 匹配，传入的值只要匹配到其中一个字符串，就算匹配成功。字符串之间使用逗号进行分割。值格式为 `value1,value2,value3` ，匹配到其中一个就算成功。<br>• 不包含：多字符串取反匹配，传入的值必须都没有出现在所配置的字符串列表中，才算匹配通过。值格式为 `value1,value2,value3` ，全部不等于才算成功。 |
| 限流指标 | 是 | 1. 限流指标：请求数<br>指定统计周期内的统计阈值，达到阈值则进行限流。可以配置多个限流阈值，多个限流阈值可同时生效，任意触发了一个即可限流。<br>○ 统计窗口时长：限流阈值的统计时长，单位秒，默认为1秒。<br>○ 请求数阈值：达到限流条件的请求数阈值。默认为1。<br>2. 限流指标：并发数（仅支持单机模式）<br>指定并发数阈值，达到阈值则进行限流。 |
| 合并计算阈值 | 否 | 当限流指标为请求数时，如果目标请求匹配到多个接口及参数，则将匹配到的所有请求汇合，合并计算阈值。 |

| | | |
|---|---|---|
| 限流效果 | 是 | • 快速失败：直接拒绝。<br>• 匀速排队：根据设置的最大排队时长依次通过。 |
| 失败处理策略 | 否 | 仅分布式限流需要选择。<br>• 退化至单击限流；当出现通信失败或者 Token Server 不可用时，限流方案退化到单机限流的模式。<br>• 直接通过：当出现通信失败或者Token Server不可用时，不做额外处理，直接通过。 |
| 失败后返回 Http 脚本 | 否 | 填写普通文本或 JSON 格式的 http 返回值。 |

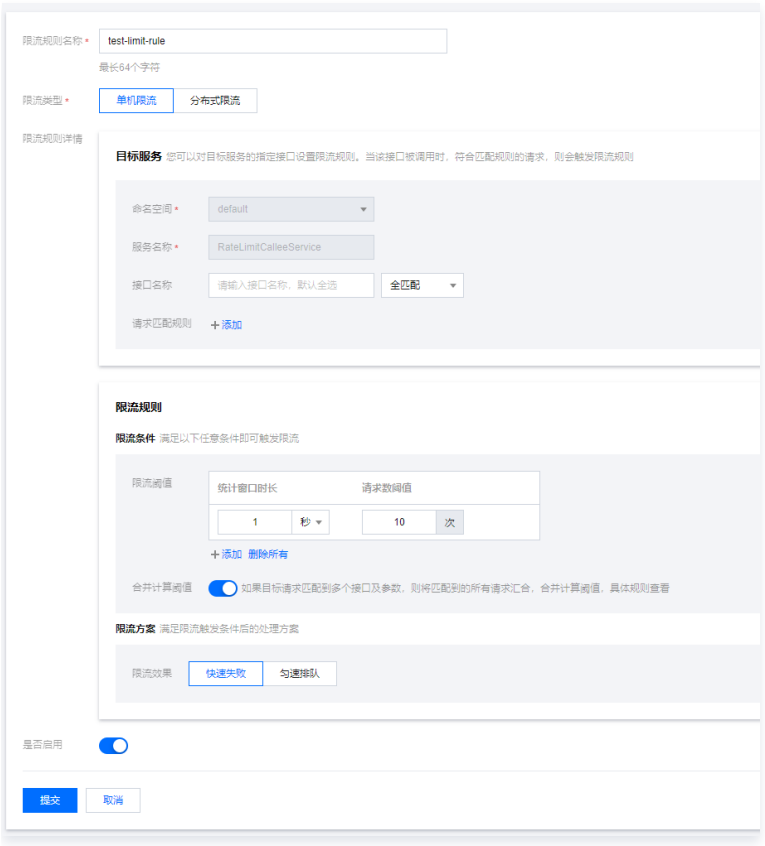
**分布式限流**

| 参数 | 是否必填 | 说明 |
|---|---|---|
| 接口名称 | 否 | 指定限流规则的接口过滤参数，接口名可对应方法名、HTTP U等信息，不填写代表不过滤。<br>• 接口：规则生效所对应的接口名，用于匹配客户端传入的 method 参数，默认为空（全部）。<br>• 匹配方式：接口字段的匹配方式，支持全匹配、不等于、包含、不包含、正则表达式五种匹配模式。 |
| 请求匹配规则 | 否 | 指定限流规则的请求参数匹配条件，不填代表不过滤，支持以下四种参数类型：<br>• 自定义参数：自定义KEY和VALUE，具体的请求参数值可通过SDK进行传入。<br>• 请求头（HEADER）：针对协议消息头（http header/grpc header）进行过滤。<br>• 请求参数（QUERY）：针对协议请求参数（http query）进行过滤。<br>• 方法（METHOD）：针对协议的METHOD（http method/grpc method）进行过滤。<br>• 主调服务：针对微服务调用场景下，主调方的服务名进行过滤。<br>• 主调 IP：针对主调方机器的IP地址进行过滤。<br>每种类型参数值支持以下几种值匹配模式：<br>• 全匹配：全值匹配，传入的值与配置的值相同才匹配通过。<br>• 正则表达式：用户配置正则表达式，通过正则表达式对传入的值进行匹配，正则表达式支持 Google RE2标准 。<br>• 不等于：取反匹配，传入的值与所配置的值不相等才算匹配成功。<br>• 包含：多字符串取OR匹配，传入的值只要匹配到其中一个字符串，就算匹配成功。字符串之间使用逗号进行分割。值格式为’value1,value2,value3‘，匹配到其中一个就算成功。<br>• 不包含：多字符串取反匹配，传入的值必须都没有出现在所配置的字符串列表中，才算匹配通过。值格式为‘value1,value2,value3’，全部不等于才算成功。 |
| 限流阈值 | 是 | 指定统计周期内的统计阈值，达到阈值则进行限流。可以配置多个限流阈值，多个限流阈值可同时生效，任意触发了一个即可限流。<br>• 统计窗口时长：限流阈值的统计时长，单位秒，默认为1秒。<br>• 请求数阈值：达到限流条件的请求数阈值。默认为1。 |
| 合并计算阈值 | 否 | 如果目标请求匹配到多个接口及参数，则将匹配到的所有请求汇合，合并计算阈值，具体规则查看 |
| 失败处理策略 | 是 | 分布式限流需要依赖token server，如果出现token server不可访问，则客户端可以根据配置的规则进行降级，保证用户请求最大限度不受影响。<br>• 退化成单机限流：默认策略，直接退化成单机计算配额的方式进行限流，单机配额=(全局配额/节点数)。<br>• 直接通过：不执行限流，所有请求都直接放通。 |
| 是否启用 | 否 | 启用后，限流规则将立即生效。 |

## 使用示例

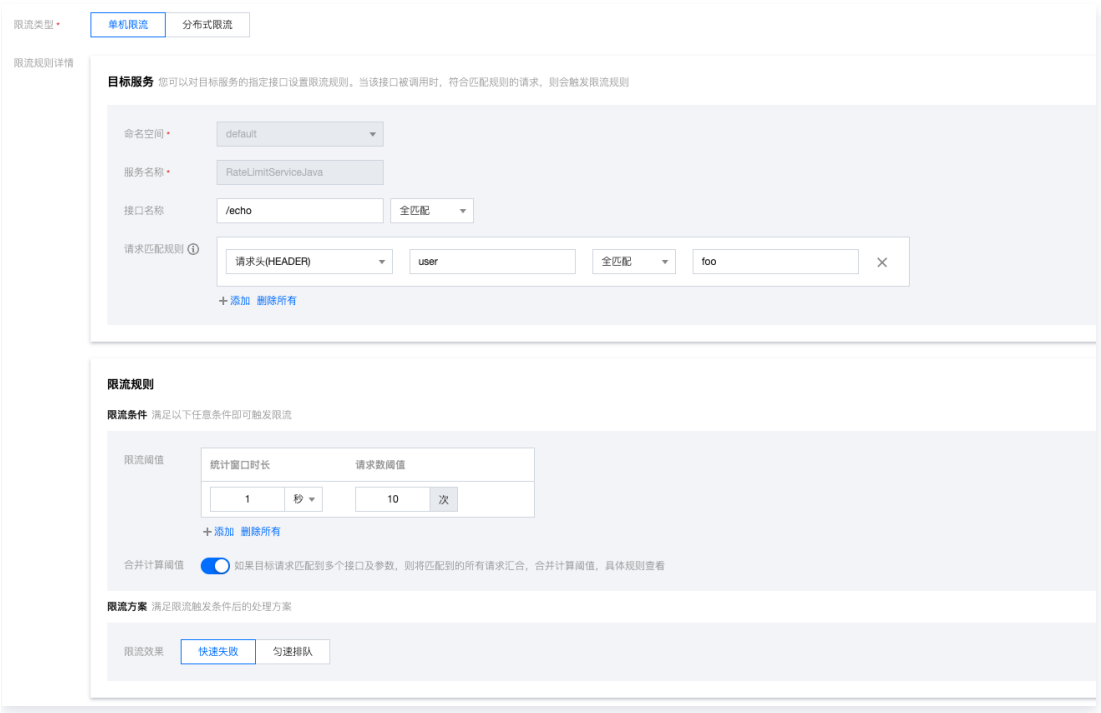
### 针对服务进行限流

在 RateLimitServiceJava 服务下新建限流规则，指定 QPS 为10，限流效果选择直接拒绝。




## 针对接口 + 标签进行细粒度限流

在 RateLimitServiceJava 服务下新建限流规则，指定 QPS 为10，方法名为/echo，针对 HTTP 请求中的 HEADER user=foo 进行限流，限流效果选择直接拒绝。

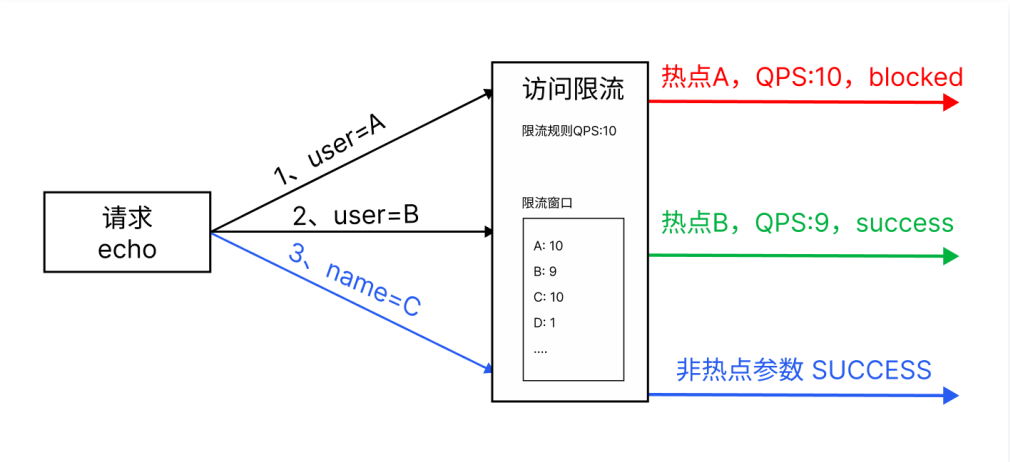



## 针对接口 + 标签进行热点参数限流

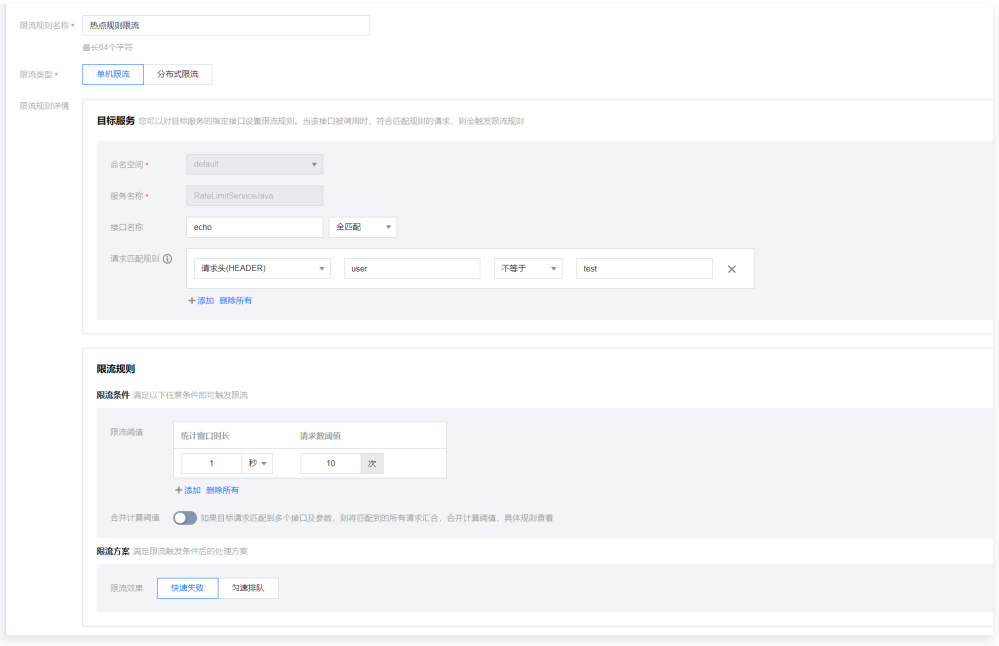
热点参数限流规则配置需要满足2个条件，否则无法生效：

- 存在非全匹配规则：“接口名称”、“请求匹配规则”配置中至少存在一条规则是非全匹配规则，例如“不等于”；如果全是全匹配规则，那该限流只是普通限流。
- 不启用“合并计算阈值”。

热点参数限流是一种针对高并发请求中某些热点参数的限流策略；如上述规则的限流效果图如下：




在 RateLimitServiceJava 服务下新建限流规则，指定 QPS 为10，方法名为/echo，针对 HTTP 请求中的 header user 的每一个参数值进行单独限流，限流效果选择直接拒绝。




## 使用匀速排队

在 RateLimitServiceJava 服务下新建限流规则，指定 QPS 为10，限流效果选择匀速排队。


限流规则名称 *
test-limit-rule
最长64个字符
限流类型 *
单机限流
分布式限流
限流规则详情
目标服务 您可以对目标服务的指定接口设置限流规则，当该接口被调用时，符合匹配规则的请求，则会触发限流规则
命名空间 *
default
服务名称 *
RateLimitCalleeService
接口名称
请输入接口名称，默认全选
全匹配
请求匹配规则
+添加
限流规则
限流条件 满足以下任意条件即可触发限流
限流阈值
统计窗口时长
请求数阈值
1
秒
10
次
+添加 删除所有
合并计算阈值
如果目标请求匹配到多个接口及参数，则将匹配到的所有请求汇合，合并计算阈值，具体规则查看
限流方案 满足限流触发条件后的处理方案
限流效果
快速失败
匀速排队
最大排队时长
1
秒
是否启用
提交
取消

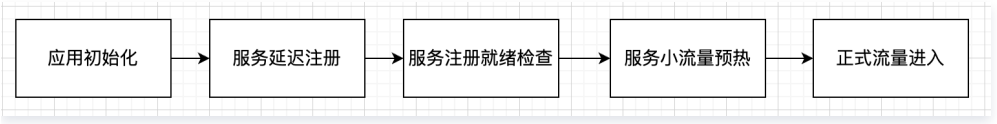
# 无损上下线
# 无损上线

最近更新时间：2025-06-11 15:55:02

## 功能概述

对于任何一个线上应用来说，发布、扩容、缩容、重启等操作不可避免。Java 应用经常会延迟加载，启动后异步加载一些资源，例如：服务需要从文件存储 COS 获取数据或者文件，待数据或者文件拉取完成后才能对外提供服务。如果在应用启动后直接注册服务，会导致服务实际没有就绪而调用失败。为了确保服务无损上线，需要实例成功注册并且就绪后，先小流量预热，再放流量进来。具体功能包含：

- 服务延迟注册
- 服务注册就绪检查（提供服务注册是否成功的接口，以便打通 K8s 就绪检查）
- 服务小流量预热




## 操作步骤

要实现服务路由需要完成两部分操作：

- 在腾讯云控制台上，设置无损上线的规则。
- 客户端接入，并引入无损上线的插件。北极星提供两种接入方式：
  - SpringCloud Tencent
  - Java Agent

下文将详细讲解使用步骤。

## 腾讯云控制台操作步骤

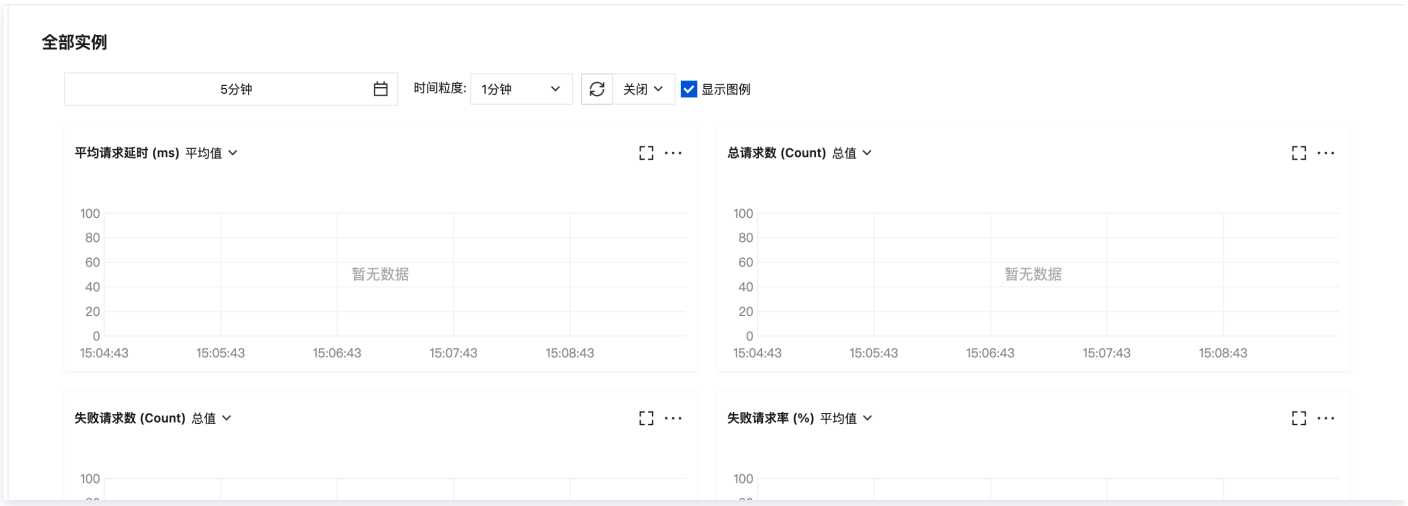
1. 登录 TSF 控制台 。
2. 在左侧导航栏，选择 **Polaris（北极星）**，在实例详情页选择目标引擎实例。
3. 单击左侧导航栏**访问限流**，进入熔断规则展示页面，单击**新建限流规则**，在弹窗中配置好限流规则后，单击**提交**。
4. 在左侧导航栏，**服务治理中心**下，单击**服务管理**，选择目标北极星引擎和目标服务，进入**服务详情页**。
5. 单击**无损上下线**的 TAB。
6. 在无损上线的配置框中，点击**设置**按钮。


7. 按业务诉求，开启配置开关。


| 功能 | 描述 | 字段 |
| --- | --- | --- |
| 延迟注册 | 北极星提供延迟注册的能力，待应用完全启动后，再进行服务注册。北极星支持两种延迟注册方式：<br>• 时长延迟：通过设置延迟注册时间，可让应用在充分初始化后再注册到注册中心对外提供服务。<br>• 探测延迟：探测应用对外暴露的检查接口，接口探测成功，返回200，确认应用可对外服务后再注册到注册中心。 | 时长延迟：<br>• 延迟注册时间：延迟注册的时间，支持0-3600秒，默认延迟注册时间：30秒<br><br>探测延迟：<br>• 探测协议：支持HTTP<br>• 探测方法：支持PUT、GET、POST、DELETE、全部。<br>• 探测接口：填写业务暴露健康检查接口，接口探测成功后，服务注册。例如：/health |
| 服务预热 | 北极星提供服务预热的能力，小流量验证后无误后，再放入正常流量。 | • 预热时长：默认120秒，支持 0~86400 秒（24小时）。<br>• 预热终止保护百分比：当超过一定百分比的节点处于预热阶段，则预热终止，避免流量聚集在少量节点导致过载。默认50%<br>• 预热曲线值：默认为1，支持 1~5 之间的整数值，流量权重会根据配置的预热曲线值呈指数型增长。数值越大，预热曲线梯度越大。<br>• 服务实例权重计算规则：<br>$[(\text{当前时间} - \text{服务实例注册时间}) / \text{预热时长}]^{\text{预热曲线值}}$ |
| 就绪检查 | 就绪检查开关开启后，提供接口判断注册是否完成。和 K8s 就绪检查功能配合使用，当应用注册完成返回200状态码，帮助 K8s 判定应用已就绪；未完成注册返回500，帮助 K8s 判定应用未就绪。<br>**接口：** /readiness<br>**端口：** 28080 | |

8. 可观测性。

   可以实时查看服务无损上线的状态、服务请求等，确保每一步操作符合预期。




## 应用客户端操作步骤

北极星提供两种接入方式：

1. SpringCloud Tencent
2. Java Agent

### SpringCloud Tencent

1. 服务通过 Spring Cloud Tencent 接入北极星服务注册与发现能力。具体步骤请参见：Spring Cloud Tencent 接入北极星 。
2. 引入 spring cloud tencent 无损上下线插件依赖。
   在 pom.xml 中添加依赖：

```
<dependencies>
    <dependency>
        <groupId>com.tencent.cloud</groupId>
        <artifactId>spring-cloud-tencent-lossless-plugin</artifactId>
    </dependency>
</dependencies>
```

### Java Agent

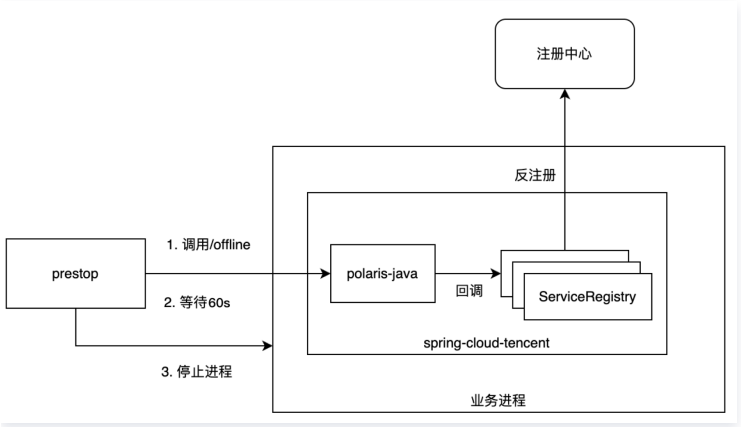
服务通过 Java Agent 接入北极星。具体步骤请参见：Spring Cloud 使用 Java Agent 。

# 无损下线

最近更新时间：2025-06-11 15:55:02

## 功能概述

应用在滚动发布或者下线过程中，被调方服务实例向注册中心发起反注册，主调方从注册中心更新 IP 的过程中，存在时间 gap，导致依然有可能调用到已经下线的实例，从而调用失败。
北极星提供服务无损下线的接口：**/offline**，**端口：28080**，结合 K8s 的生命周期，实现服务无损下线。整体流程如下：




## 操作步骤

要实现服务路由需要完成两部分操作：

- 在腾讯云控制台上，打开无损下线的开关。
- 客户端接入，并引入无损下线的插件。北极星提供两种接入方式：
  - SpringCloud Tencent
  - Java Agent

下文将详细讲解使用步骤。

## 控制台操作步骤

1. 登录 TSF 控制台 。
2. 在左侧导航栏，**服务治理中心**下，单击**服务管理**，选择目标北极星引擎和目标服务，进入**服务详情页**。
3. 单击**无损上下线**的 TAB。
4. 在无损下线的配置框中，点击**设置**按钮。


5. 按业务诉求，开启配置开关。


6. 在容器服务 TKE 等 K8s 应用部署平台中配置 preStop 生命周期检查。
   preStop 配置检查指令：`curl -X PUT http://localhost:28080/offline && sleep 20`

## 应用客户端操作步骤

北极星提供两种接入方式：

- SpringCloud Tencent
- Java Agent

### SpringCloud Tencent

1. 服务通过 Spring Cloud Tencent 接入北极星服务注册与发现能力。具体步骤请参见：Spring Cloud Tencent 接入北极星 。
2. 引入 spring cloud tencent 无损上下线插件依赖。
   在 pom.xml 中添加依赖：

```
<dependencies>
    <dependency>
        <groupId>com.tencent.cloud</groupId>
        <artifactId>spring-cloud-tencent-lossless-plugin</artifactId>
    </dependency>
</dependencies>
```

### Java Agent

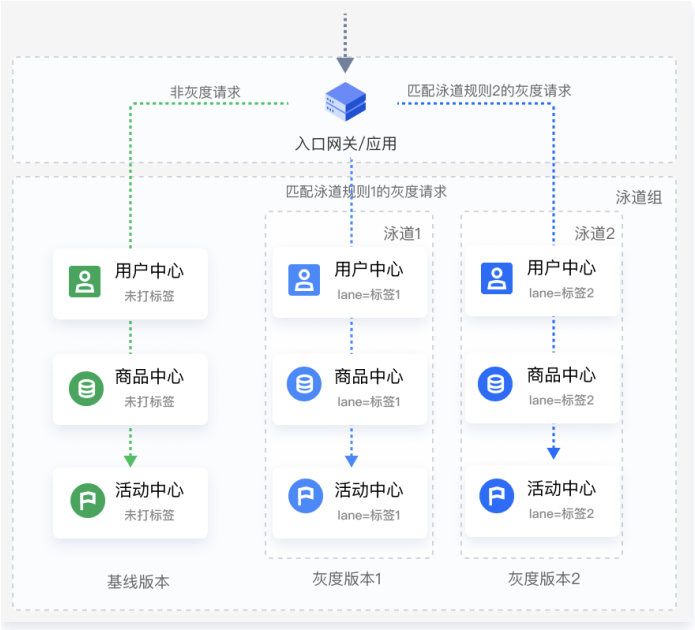
服务通过 Java Agent 接入北极星。具体步骤请参见：Spring Cloud 使用 Java Agent 。

# 全链路灰度
# 联动云原生网关实现全链路灰度

最近更新时间：2025-04-17 16:00:53

## 使用场景

应用发布在软件开发过程中非常常见与频繁，但是变更过程容易引发故障。为了降低每次发布带来的变更风险，可以通过灰度发布策略来解决这一问题。灰度发布是一种软件发布策略，它允许您在生产环境中渐进式部署应用，新版本只对部分用户可见，在问题出现时尽量减少影响。在微服务体系架构中，服务间的依赖关系错综复杂，有时单个服务发版依赖多个服务同时运行联动，对这个新版本服务的上下游进行分组验证，这是微服务架构中特有的全链路灰度发布场景。
TSF-Polaris（北极星）提供的全链路灰度能力，在不修改业务代码的情况下，可视化配置灰度规则，就可以构建出从网关到后端服务的端到端的全链路流量控制。泳道可以将应用灰度发布的版本隔离成一个独立的运行环境。通过设置泳道规则，可以将满足规则的请求流量路由到目标版本的应用。如下图所示，我们划分出一个或多个灰度环境，针对线上灰度版本1的所有服务，我们希望都能配置一定规则。当满足一定规则后，所有请求都会流入灰度环境1中。




## 名词解释

- **泳道组**：一组泳道、服务、入口网关的集合，通常一个业务系统下的应用放在同一个泳道组中。
- **泳道**：一组具有相同特性标签的服务实例节点的集合，是灰度发布规则的目的地。
- **基线**：没有添加至任何泳道的服务节点为基线版本节点。
- **实例标签**：TSF 北极星提供特定的泳道标签，标签 key 为“lane”，value 值为您自定义的「泳道标签值」。打上该标签的实例会自动添加至目标泳道中。

## 使用限制

| 支持的组件 | | 版本限制 |
| --- | --- | --- |
| 北极星 | | • 产品版本：企业版<br>• 引擎版本 >= v 1.18.0.0 |
| 微服务应用 | 北极星 SDK | 版本 >= v 1.15.3 |
| | SpringCloud Tencent | • SpringCloud Tencent 2023版本：>= 1.13.1-2023.0.0<br>• SpringCloud Tencent Hoxton 版本：>= 1.14.0-Hoxton.SR12-RC2 |
| 网关 | 云原生网关 | 版本无限制 |

| | | |
|---|---|---|
| | SpringCloud Gateway | • SpringCloud Tencent 2023版本：>= 1.13.1-2023.0.0<br>• SpringCloud Tencent Hoxton 版本：>= 1.14.0-Hoxton.SR12-RC2 |
| 消息队列 | Ckafka | 建设中 |
| | TDMQ-RocketMQ | 建设中 |
| | TDMQ-Pulsar | 建设中 |

## 前提条件

- 已 创建北极星引擎实例 。
- 我们为您提供了全链路灰度 Demo ，请参见 Demo 。
- 全链路灰度能力需要您在客户端接入北极星服务注册发现、服务路由能力。
  - SpringCloud Tencent 应用接入，请参见 接入全链路灰度 。
  - Java Agent 接入请参见 通过 Java Agent 接入北极星 。

## 配置流程总览

**步骤1：搭建业务系统基线版本**
（1）部署业务应用基线版本。
（2）创建 云原生网关。
（3）关联 北极星注册中心，配置 云原生网关服务路由。
**步骤2：搭建业务系统灰度环境**
（1）部署业务应用灰度版本，并打上灰度标签。
（2）创建泳道组，配置业务系统的入口网关、业务应用。
（3）创建泳道并配置灰度规则，符合泳道规则的请求将被转发至泳道内。

## 步骤1：搭建业务系统基线版

### （1）部署业务应用基线版本。

根据您自身的业务，准备好业务部署的资源，通过 容器化部署 、 虚拟机部署 选择其中一种方式部署业务系统后端应用的基线版本。

- **容器化部署：**
  - 创建 TKE 容器集群，请参见 创建 TKE 集群 。
  - 在所选集群中，创建工作负载，完成业务应用部署，请参见 创建工作负载 。
- **虚拟机部署：**
  - 创建 CVM 虚拟机，请参见 创建 CVM 虚拟机 。
  - 部署业务应用。

### （2）创建云原生网关。

创建云原生网关，参见 创建云原生网关 。

### （3）关联北极星注册中心，并配置服务和路由信息。

服务来源关联北极星注册中心（PolarisMesh），并配置网关服务和路由信息，参见 使用云原生 API 网关访问北极星服务 。

## 步骤2：搭建业务系统灰度环境

如果业务应用要发布新版本，可以通过全链路灰度来同时验证不同应用的灰度版本，以确保发布上线的稳定性。

### （1）部署业务应用灰度版本，并打上灰度标签。

根据您自身的业务，准备好业务部署的资源，通过 容器化部署 、 虚拟机部署 选择其中一种方式部署业务系统后端应用的**灰度版本**。

**容器化部署**

1. **创建 TKE 容器集群**，请参见 创建 TKE 集群 。

2. 在所选集群中，**创建工作负载**，请参见 创建工作负载 。
3. **灰度打标**：北极星提供灰度泳道标签，标签key为“lane”，value 值为您自定义的**版本标签值**。您可以在配置工作负载的过程中，`设置启动参数` 或者 `设置环境变量`（二选一）来配置灰度标签：

   **方案一：设置应用启动参数**

   ```
   Java -Dspring.cloud.tencent.metadata.content.lane=$(lane_value) -jar demo.jar
   ```

   **方案二：设置环境变量**

   ```
   设置环境变量：
   SCT_METADATA_CONTENT_lane=$(lane_value)
   ```

   > **说明：**
   > `$(lane_value)` 替换为您自定义的版本（泳道）标签，例如：grey、v2等。

4. 完成灰度版本部署。

虚拟机部署

1. **创建 CVM 虚拟机**，请参见 创建 CVM 虚拟机 。
2. **灰度打标**：北极星提供灰度泳道标签，标签 key 为“lane”，value 值为您自定义的「版本标签值」。您可以在配置工作负载的过程中，`设置启动参数` 或者 `设置环境变量`（二选一）来配置灰度标签：

   **方案一：设置应用启动参数**

   ```
   Java -jar -Dspring.cloud.tencent.metadata.content.lane=$(lane_value) demo.jar
   ```

   **方案二：设置环境变量**

   ```
   设置环境变量：
   SCT_METADATA_CONTENT_lane=$(lane_value)
   ```

   > **说明：**
   > $(lane_value)替换为您自定义的版本（泳道）标签，例如：v2。

3. 完成灰度版本部署。

## （2）创建泳道组，配置业务系统的入口网关、业务应用。

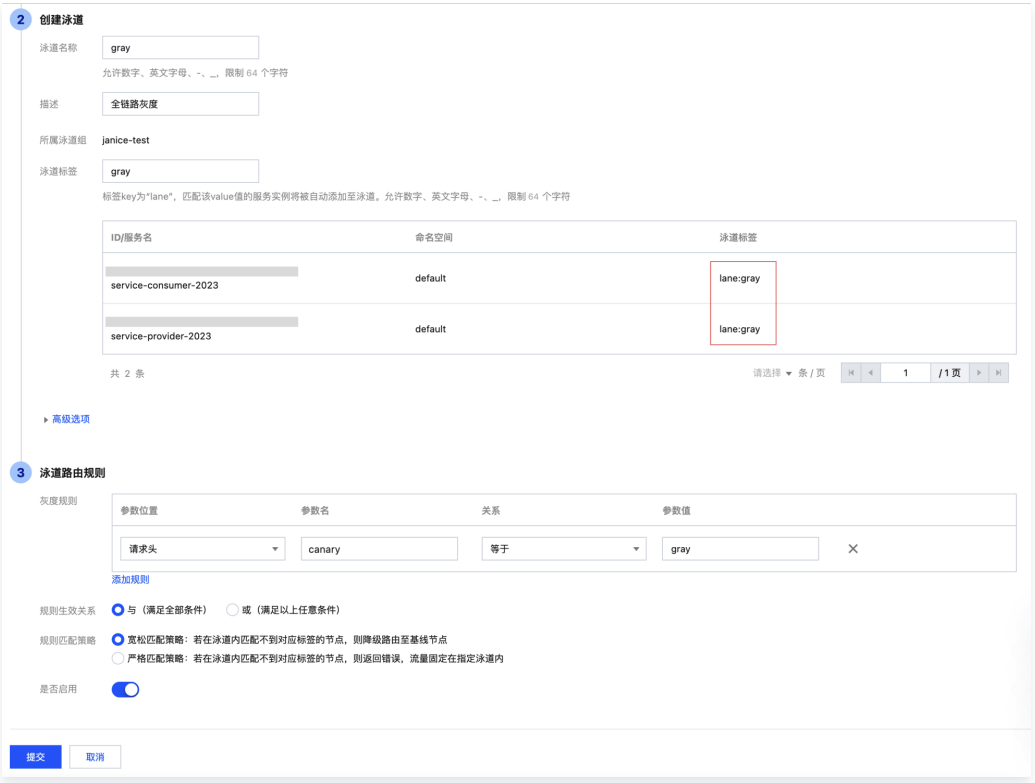
1. 侧边栏单击**全链路灰度** > **新建泳道组**，进入**新建泳道组**的页面。
2. 填写泳道组相关配置，配置入口网关、业务应用。


- **泳道组名称**：允许数字、英文字母、点、短横线、下划线限制 64 个字符。
- **配置泳道入口**：支持对接云原生网关、微服务网关 SpringCloud Gateway、普通微服务应用。自动同步您在北极星所配置的泳道规则。
  - 云原生网关：选择与北极星在相同地域和 vpc 下的网关实例，并选择业务系统入口服务。入口服务支持多选。
  - 微服务网关 SpringCloud Gateway：选择作为系统入口的 SpringCloud Gateway 网关服务。支持多选。
  - 微服务应用：选择作为系统入口的 SpringCloud Tencent 服务。支持多选。
- **配置泳道组服务**：
  - 一般一个泳道组对应一个业务系统，可将业务系统中全部服务全部导入进来。
  - 如果服务暂未注册，也支持手动添加。

3. 单击**提交**按钮，泳道组即被创建，可在泳道组列表中查看。

## （3）创建泳道并配置灰度规则，符合泳道规则的请求将被转发至泳道内。

1. 侧边栏单击**全链路灰度**，在目标泳道组右上角单击**创建泳道**，进**新建泳道**的页面。
2. 填写泳道组相关配置，配置入口网关、业务应用。




- **创建泳道**
  - **泳道名称**：允许数字、英文字母、短横线、下划线，限制 64 个字符。
  - **描述**：填写泳道的描述和用途。
  - **泳道标签**：填写灰度发布服务的泳道标签，北极星自动将符合该标签的服务自动添加至泳道。
    - **使用示例**：如上图，业务系统整体有10个服务，全部添加至泳道组。此次发布其中 2 个服务上线（service-consumer-2023，service-provider-2023），并配置 lane = grey 的标签。则这两个灰度版本实例自动添加至泳道。
- **泳道路由规则**
  - **灰度规则**：
    - **通过配置灰度规则，将符合灰度规则的流量染色并转发至目标泳道内。灰度规则将自动同步至泳道入口（云原生网关、SpringCloud Gateway、入口微服务）。**
    - 灰度规则参数支持请求头、请求方法、请求参数、请求 Cookie、请求路径、主调 IP。
  - **规则生效关系**：支持 与（满足全部条件）、或（满足以上任意条件）。
  - **规则匹配策略**：支持宽松匹配与严格匹配策略，请根据您业务实际情况选择。
    - 宽松匹配策略：若在泳道内匹配不到对应标签的节点，则降级路由至基线节点。
    - 严格匹配策略：若在泳道内匹配不到对应标签的节点，则返回错误，流量固定在指定泳道内。

3. 单击**提交**，即可完成泳道的创建，在泳道组列表中可以查看泳道列表信息。
4. 单击 **云原生网关** > **路由管理** > **服务** > 选择入口服务 > **灰度策略**，查看泳道规则已同步至云原生网关。至此已完成全部的规则配置。

# 配置管理（配置中心）
# 配置分组

最近更新时间：2025-04-17 16:00:53

## 操作场景

北极星提供可视化的配置管理界面，您可以在 TSF 控制台查看配置文件的发布情况，并对配置进行编辑修改等。
该任务指导您如何通过 TSF 控制台在某一北极星引擎下进行配置分组管理。

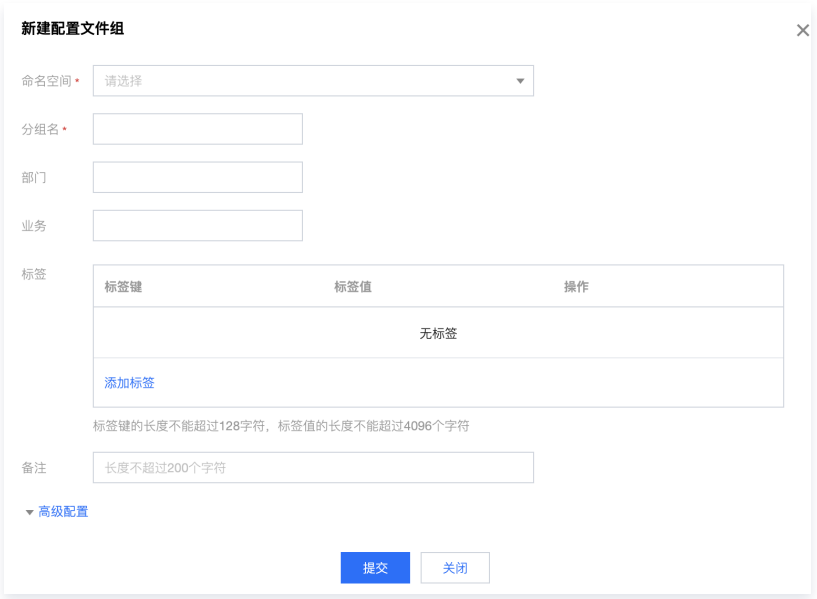
## 前提条件

- 已 创建北极星引擎 。
- 已 创建命名空间 。

## 操作步骤

### 新建配置分组

1. 登录 TSF 控制台，在左侧菜单栏选择 Polaris（北极星），进入引擎实例列表页。
2. 在实例列表页选择目标引擎，点击 配置管理，进入配置分组页面。
3. 在左上角单击新建，根据自身业务需求选择相关信息。
   - 命名空间：选择创建好的命名空间。
   - 分组名：填写服务名称，允许数字、英文字母、.、-、_，限制128个字符。
   - 部门：可选，填写业务部门信息。
   - 业务：填写业务信息。
   - 标签：可选，用于分类管理资源，具体使用方法可参见 标签管理 。
   - 备注：可选，填写配置分组描述信息。




4. 单击**提交**，完成新建配置分组。

后续操作：配置分组创建完成后，您需要继续在该配置分组下创建配置文件，参见 配置文件 。

### 删除配置分组

1. 在左侧导航栏选择**配置管理** > **配置分组**，选择目标配置分组，单击操作列的**删除**。
2. 在二次弹窗页面确认删除。

# 配置文件

最近更新时间：2025-05-16 15:28:02

## 操作场景

该任务指导您通过 TSF 控制台，如何在某一北极星引擎下进行配置文件管理。

## 前提条件

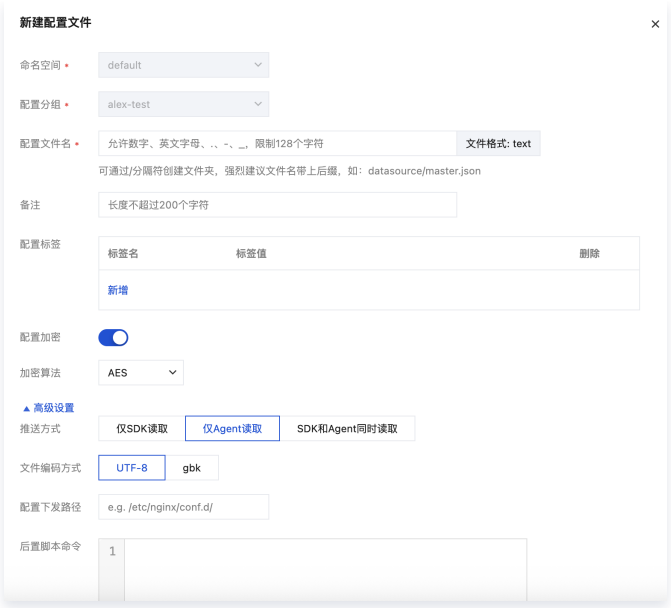
- 已 创建北极星引擎 。
- 已 创建命名空间 。

## 操作步骤

### 创建配置文件

1. 登录 TSF 控制台，在左侧导航栏点击 **Polaris（北极星）** 进入实例列表详情页。
2. 单击目标实例 ID，进入引擎实例信息页，左侧选择配置管理进入配置分组页面。
3. 选择目标配置分组并单击进入以下页面。

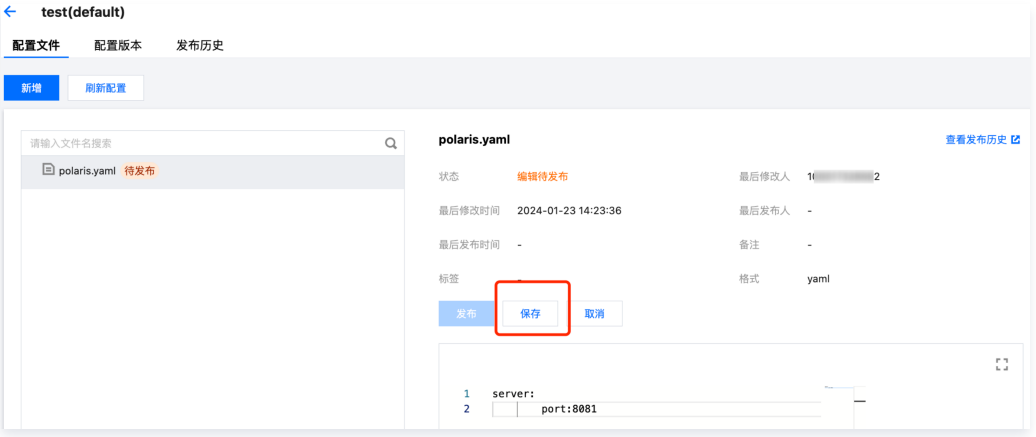

4. 单击新增创建配置文件。
   - 命名空间：目标配置分组所在命名空间。
   - 配置分组：配置分组名。
   - 配置文件名：配置文件的名称；允许数字、英文字母、-、_，限制128个字符。可通过/分隔符创建文件夹，强烈建议文件名带上后缀，如：datasource/master.json。
   - 备注：可选，填写配置文件的描述信息。
   - 配置标签：可选，配置标签可用于标识配置的用处、特征，格式为 `key:value`。
   - 配置加密：可选，支持 AES 配置加密算法。
   - 高级设置：
     - 推送方式：仅 SDK 读取/仅 Agent 读取 / SDK 和 Agent 同时读取。
     - 当选择“仅 Agent 读取”时候：
       - 文件编码方式：UTF-8/gbk。
       - 配置下发路径：配置下发到服务器的路径。
       - 后置脚本命令：配置下发到服务器后执行的命令（**不需要**包含 #! /bin/bash）




5. 单击**提交**，完成新建配置文件。

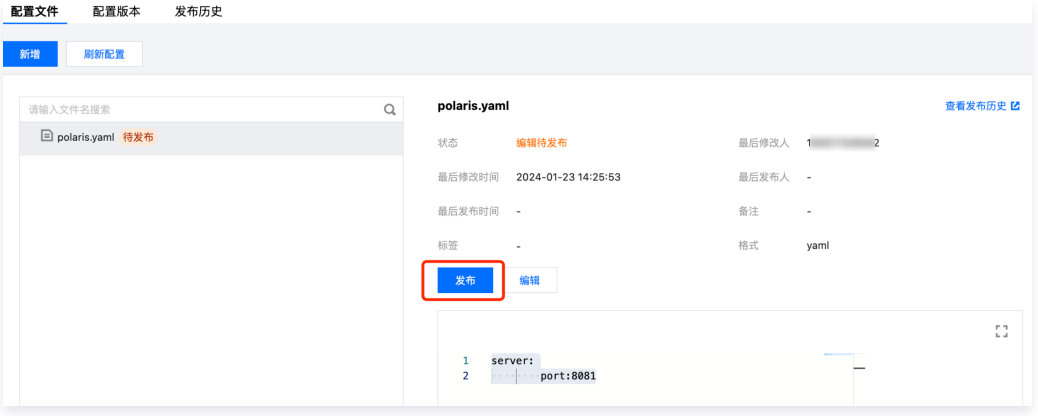
## 编辑配置文件

1. 在配置文件页面，选择目标配置文件。
2. 选择期望编辑的配置文件，并单击**编辑**按钮。
3. 对配置文件内容进行编辑之后，单击**保存**。

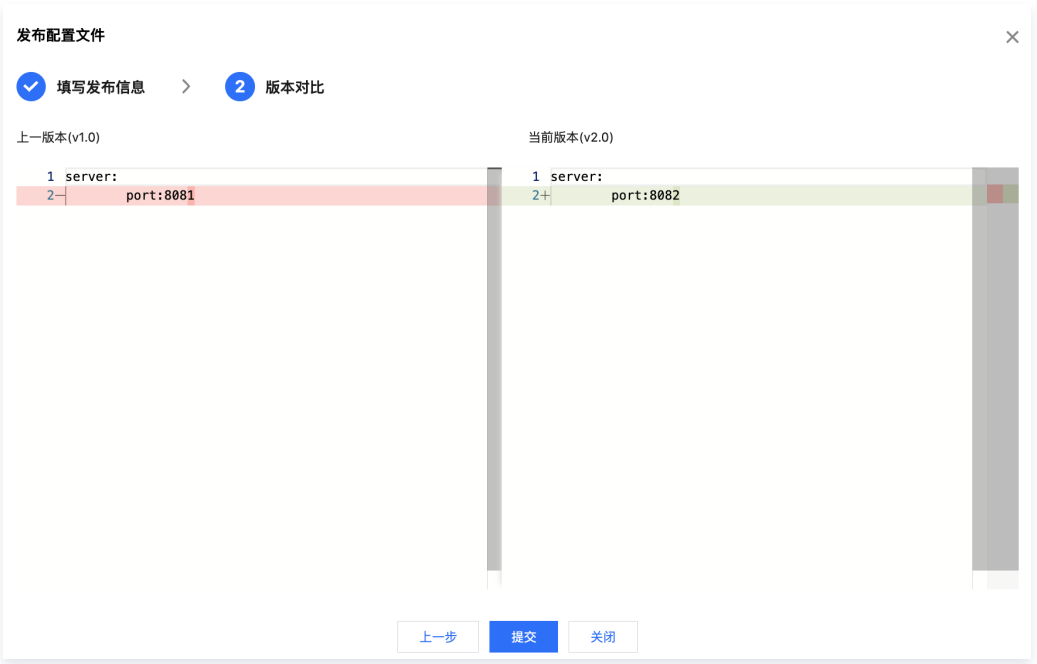



## 发布配置文件

1. 在配置文件页面，选择目标配置文件。
2. 选择期望发布的配置文件，并点击**发布**。

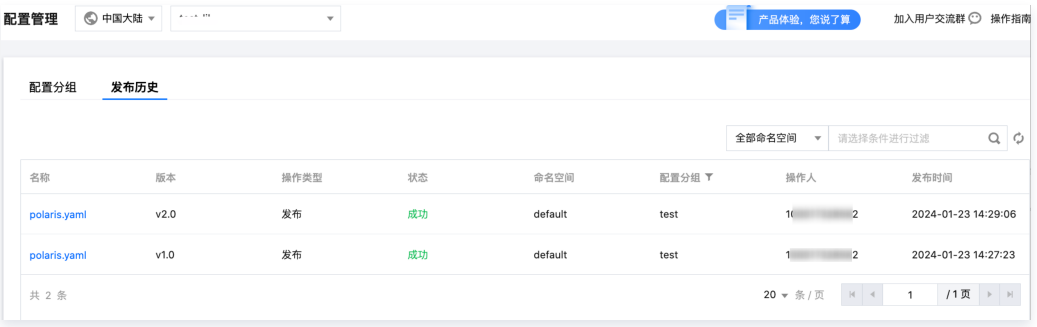



3. 在弹窗中输入配置版本好，并进行二次确认发布动作即可。




## 查看发布历史

1. 在左侧导航栏单击**配置管理**，选择好地域和实例后，单击**发布历史**页签。
2. 在发布历史页面，可以查看配置发布记录，您也可以在右上角通过命名空间和分组名/配置文件名来查询您想要查看的配置发布记录。




## 配置回滚

1. 在左侧导航栏单击**配置管理**，选择好地域和实例后，单击目标配置分组进入配置文件页面。
2. 在页面上方选择**配置版本**，可以查看历史配置版本。


3. 在操作栏单击**版本对比**，可以查看当前版本与历史版本差异对比。


4. 在操作栏单击**回滚**，在弹窗中二次确认后可回滚配置。

# 配置轨迹

最近更新时间：2025-06-11 15:55:02

## 操作场景

该任务指导您通过 TSF 控制台，如何在某一北极星引擎下的配置下发轨迹，查看配置数据从服务端下发至运营端的完整过程与状态流转路径。

## 前提条件

- 已 创建北极星引擎实例 。
- 已 创建命名空间 。

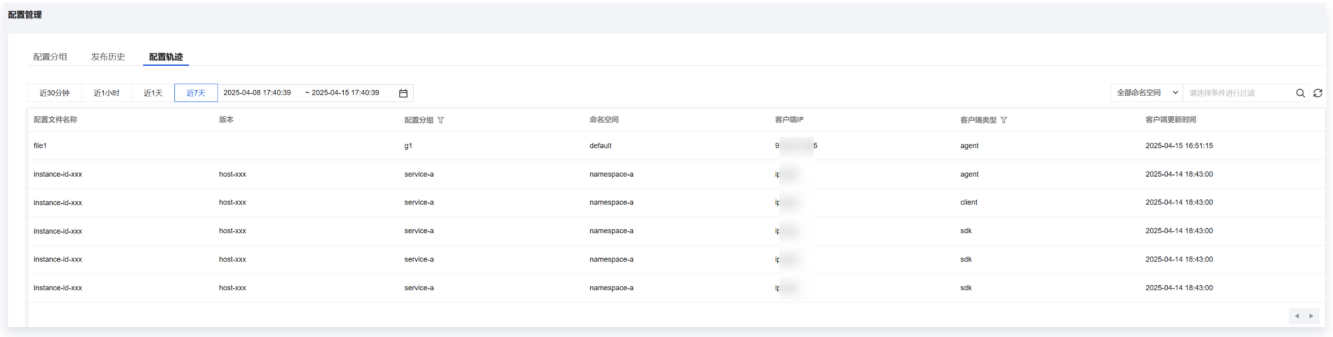
## 操作步骤

### 创建配置文件

1. 登录 TSF 控制台，在左侧导航栏单击 Polaris（北极星）进入实例列表详情页。
2. 单击目标实例 ID，进入引擎实例信息页，左侧选择**配置管理**后，进入**配置管理**详情页。

2.1 选择**发布历史**

选择目标发布记录后，单击操作栏的**配置轨迹**后跳转至**配置轨迹**页面。


2.2 选择**配置轨迹**标签页，可以通过配置分组、客户端IP、客户端类型筛选并查看目标配置轨迹。

# 配置 Agent 安装

最近更新时间：2025-06-11 16:08:51

## 功能概述

北极星创建的配置文件支持推送到本地的对应路径下，创建对应的文件内容。
订阅对应服务端的文件配置后，polaris-config-agent 会自动感知服务端是否有变更，有变更后会同步更新本地文件。

## 操作步骤

## CVM 场景

### 步骤1：下载 agent

- x86版本：polaris-config-agent-x86_64.tar.gz
- arm版本：polaris-config-agent-arm64.tar.gz

### 步骤2：解压

解压下载的 agent.tar.gz 物料包：`tar -zxvf polaris-config-agent.tar.gz` 解压后生成的目录如下：

```
--
|-bin
|-config
```

### 步骤3：启动

到 bin 目录下执行启动脚本 sh start.sh
具体参数有2种方式注入：
1. 通过 start.sh 后添加启动参数。

```
-discover-addresses=${polaris_discover_addresses} -config-addresses=${polaris_config_addresses} -config-groups=${polaris_config_groups} -config-files=${polaris_config_files} -config-token=${polaris_config_token}
```

2. 通过配置文件，agent 启动会自动读取 config 目录下的 polaris-config-agent.yaml 配置文件，可在 config 目录下 `cp polaris-config-agent.example.yaml` 并在 polaris-config-agent.yaml 后在配置文件中填写具体的配置内容。

> ⚠ 注意：
> 启动参数和配置文件都有内容的情况下，优先级为：启动参数 > 配置文件。

## 容器场景

### 步骤1：获取镜像地址

agent 镜像地址：`ccr.ccs.tencentyun.com/tsf_100011913960/polaris-agent`

### 步骤2：配置 sidecar 容器

将 polaris-config-agent 以 sidecar 的方式启动，agent 参数可通过配置环境变量、configmap 的方式注入，具体参数介绍如下：
**参数介绍**

| 参数 | 描述 | 支持环境变量注入 | 参数示例 | 环境变量注入示例 | 是否必须 |
| --- | --- | --- | --- | --- | --- |
| discover- | 北极星服务发现地址 | ${polaris_discover_addresses} | -discover-addresses=127.0.0.1:8091,127.0.0.2:8093 | -discover-addresses=${polaris_discover_addresses} | 可选，如果不填，则 discover-address=config_ip:8091 |

| | | | | | |
|---|---|---|---|---|---|
| addresses | | | | | |
| config-addresses | 配置中心访问地址 | ${polaris_config_addresses} | -config-addresses=127.0.0.1:8093,127.0.0.2:8093 | -config-addresses=${polaris_config_addresses} | 必须 |
| config-token | 配置中心访问 token | ${polaris_config_token} | -config-token=aaaa | -config-token=${polaris_config_token} | 可选 |
| config-groups | 待获取的配置分组 | ${polaris_config_groups} | -config-groups=ns1#group1,ns2#group2 | -config-groups=${polaris_config_groups} | config-groups 和 config-files 任意有一个或两个都有 |
| config-files | 待获取的配置文件 | ${polaris_config_files} | -config-files=ns1#group1#file1,ns2#group2#file2 | -config-files=${polaris_config_files} | config-groups 和 config-files 任意有一个或两个都有 |

# 可观测性
# 引擎运行监控

最近更新时间：2025-06-04 11:46:02

## 操作场景

北极星对运行的实例、接口提供了多项监控指标，用以监测北极星网格节点及服务的运行情况，例如：请求失败总数、请求延迟等**请求指标**，CPU、内存、网络等**系统指标**。您可以根据这些指标实时了解北极星网格的运行状况，针对可能存在的风险及时处理，保障系统的稳定运行。本文为您介绍通过 TSF 控制台查看北极星监控数据的操作方法。

## 监控指标及含义

### 注册配置指标

| 指标分类 | 指标名 | 指标含义 |
| --- | --- | --- |
| 注册中心 | 总服务数 | 注册中心内所有已完成注册的服务总量，包含在线、离线、异常等全状态服务 |
| | 在线服务数 | 当前至少存在1个健康实例的服务数量，可正常接收请求 |
| | 离线服务数 | 在线服务实例为0的服务数量 |
| | 异常服务数 | 存在服务实例但无健康节点的服务数量 |
| | 总服务实例数 | 所有服务实例的全局计数，包含健康、异常、隔离、未隔离状态 |
| | 在线服务实例数 | 通过健康检查且可响应请求的实例数量 |
| | 异常服务实例数 | 连续心跳超时或主动标记为不可用的实例数量 |
| | 隔离服务实例数 | 因熔断策略或人工操作暂停流量转发的实例数量 |
| 配置中心 | 配置文件总数 | 配置文件数量 |
| | 配置分组总数 | 配置分组数量 |
| | 已发布配置文件总数 | 已推送至客户端的配置文件数量 |

### 请求指标

| 指标分类 | 指标名 | 指标含义 |
| --- | --- | --- |
| 请求数 | 请求失败数(次) | 注册中心集群内各节点/接口的请求失败数。 |
| | 请求数(次) | 注册中心集群内各节点/接口的请求次数。 |
| | 请求成功数(次) | 注册中心集群内各节点/接口的请求成功数。 |
| 请求失败细分 | 网络失败数(次) | 注册中心集群内各节点/接口的网络失败数。 |
| | 其他失败数(次) | 注册中心集群内各节点/接口的其他失败数。 |
| | 系统失败数(次) | 注册中心集群内各节点/接口的系统失败数。 |
| 请求延时 | 请求平均延时(ms) | 注册中心集群内各节点/接口的请求平均延时。 |
| | 最大请求延迟(ms) | 注册中心集群内各节点/接口的最大请求延迟。 |
| | 请求最小延时(ms) | 注册中心集群内各节点/接口的请求最小延时。 |

### 系统指标

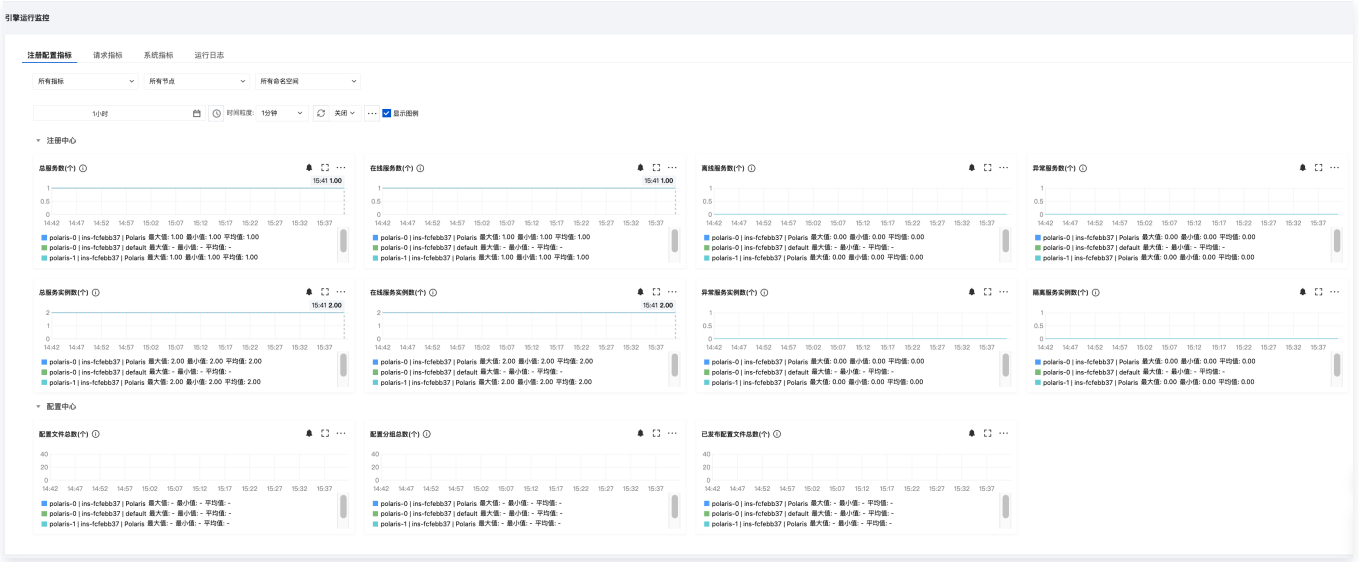
| 指标名 | 指标含义 |
| --- | --- |
| CPU使用率（%） | 注册中心集群内各节点的 CPU 使用率。 |
| 内存使用率（%） | 注册中心集群内各节点的内存使用率。 |
| 网络入包量（个/s） | 注册中心集群内各节点的网络入包量。 |
| 网络入流量（MBytes） | 注册中心集群内各节点的网络入流量。 |
| 网络出包量（个/s） | 注册中心集群内各节点的网络出包量。 |
| 网络出流量（MBytes） | 注册中心集群内各节点的网络出流量。 |
| 磁盘读取次数（次） | 注册中心集群内各节点的磁盘读取次数。 |
| 磁盘读取带宽（MBps） | 注册中心集群内各节点的磁盘读取带宽。 |
| 磁盘写入次数（次） | 注册中心集群内各节点的磁盘写入次数。 |
| 磁盘写入带宽（MBps） | 注册中心集群内各节点的磁盘写入带宽。 |

## 查看监控入口

1. 登录 TSF 控制台，在左侧导航栏选择 **Polaris（北极星）**进入引擎实例列表页。
2. 在引擎实例列表页，单击目标引擎的“ID”，进入基本信息页面。
3. 在左边页签单击**引擎运行监控**，可查看业务指标和系统指标。

### 注册配置指标

**选择好指标、节点和命名空间**，设置好时间范围（支持近1小时、近12小时、近1天、近30天和自定义），可查看对应的监控数据。

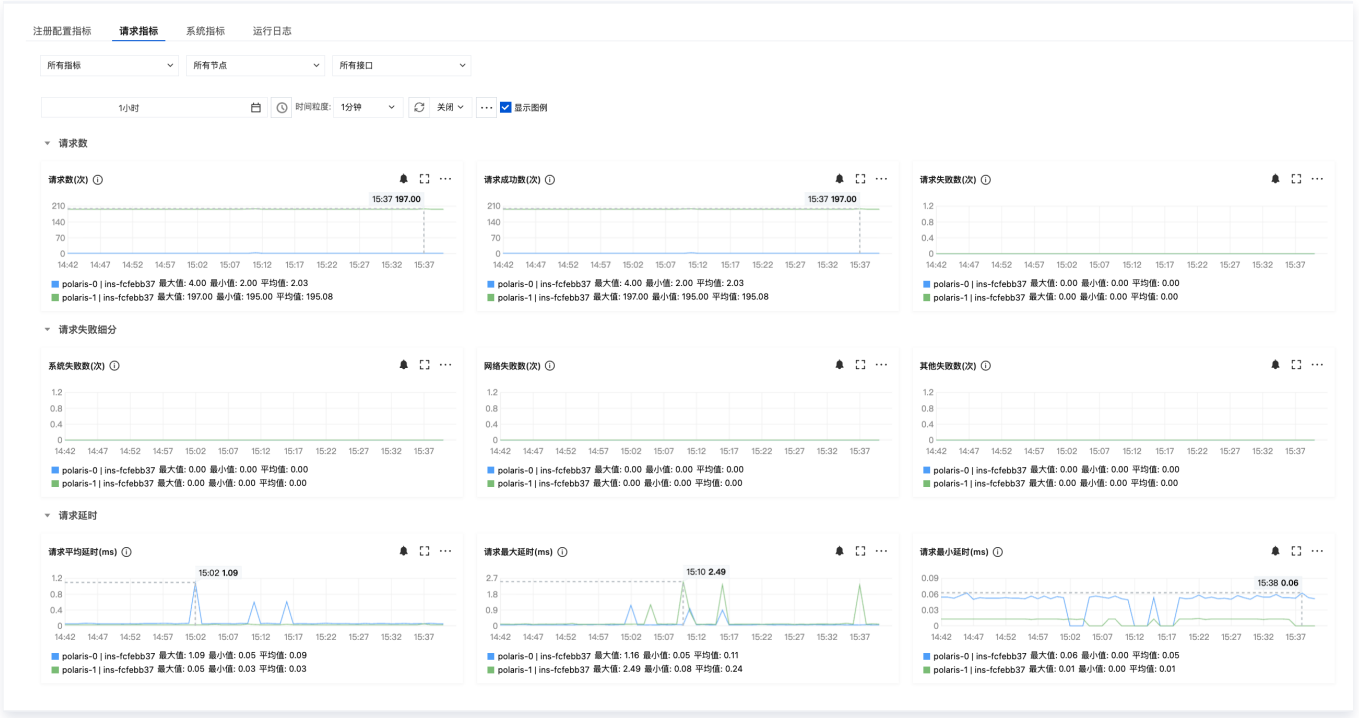



| 图标 | 说明 |
| --- | --- |
|  | 单击可查看监控指标同环比。 |
|  | 单击可刷新获取最新的监控数据。 |
|  | 单击可将图表复制到 Dashboard，关于 Dashboard 请参见 什么是 Dashboard 。 |

| | |
|---|---|
| 显示图例 | 勾选后可在图表上显示图例信息。 |
|  | 跳转至 腾讯云可观测平台控制台 配置告警策略。 |

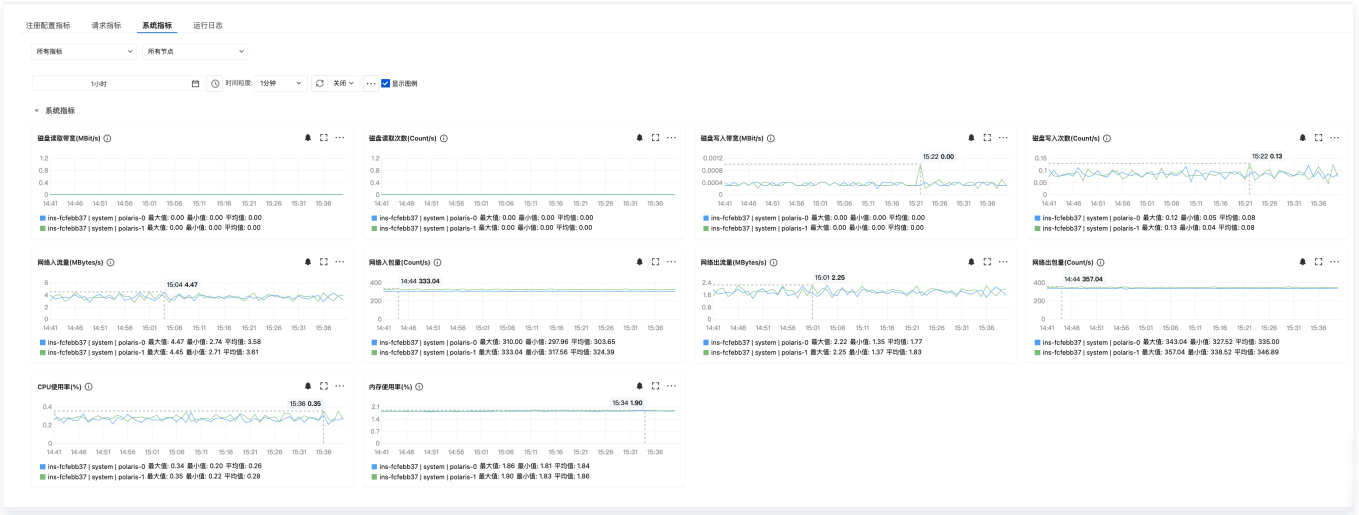
## 系统指标

**选择好指标、节点和接口，设置好时间范围（支持近1小时、近12小时、近1天、近30天和自定义），可查看对应的监控数据。**




## 系统指标

**选择好时间范围（支持近1小时、近12小时、近1天、近3天和自定义）、指标和节点后，查看相应的监控数据。**



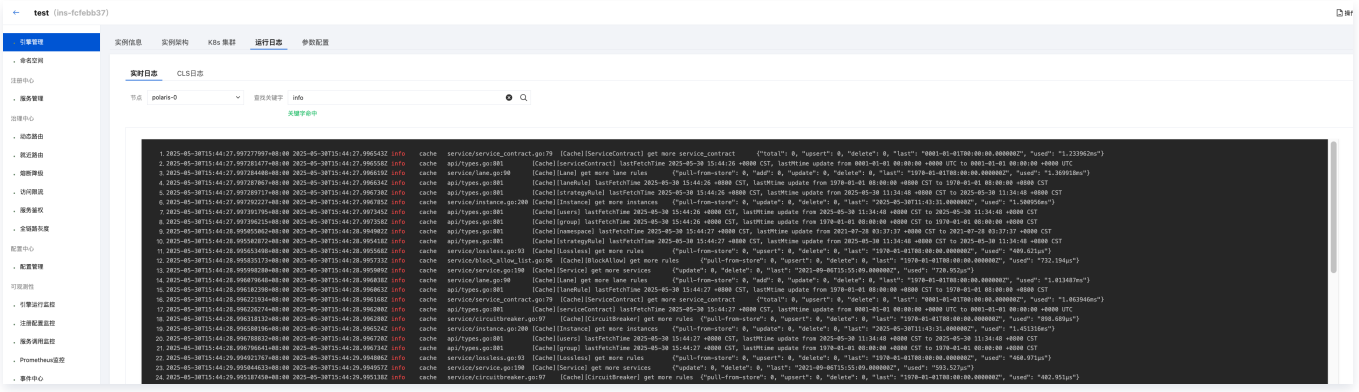

## 运行日志

支持查看实时日志和CLS日志。

### 1. 实时日志

1.1 选择好节点后，可查看对应节点的运行日志。

1.2 在日志页面的搜索框，可以通过关键字查询相关日志。输入关键词查询，例如：“info”，注意日志检索区分大小写。




### 2. CLS日志

PolarisMesh 持久化日志、事件中心、操作记录、告警数据存储在CLS服务中，开启CLS日志服务后可查看。

# 注册配置监控

最近更新时间：2025-04-17 16:00:53

Polaris（北极星）针对运行中实例的注册配置能力提供了多项监控指标，例如：服务概览数据、服务及配置数据、服务端请求数据，用以监测该实例及服务的运行情况。您可以根据这些指标实时了解北极星的运行状况，针对可能存在的风险及时处理，保障系统的稳定运行。本文为您介绍通过 TSF 控制台查看北极星注册配置监控数据的操作方法。

## 监控指标及说明

### 概览

| 指标 | 说明 |
| --- | --- |
| 客户端节点数 | 客户端总节点数。 |
| 客户端连接数 | 包含总连接数、注册中心连接数、配置中心连接数。 |
| 总请求数 | 客户端访问北极星服务端请求数。包含总请求数、成功请求数、失败请求数、请求成功率。 |
| 请求时延 | 客户端访问北极星服务端请求时延。包含均值、最大值、最小值、P99、P95。 |
| 服务数 | 包含总服务数、在线服务数、异常服务数、离线服务数。 |
| 实例数 | 包含总实例数、在线实例数、隔离实例数、异常实例数。 |
| 配置分组数 | 配置分组总数。 |
| 配置数 | 配置文件数、已发布配置文件数。 |

### 服务和配置统计

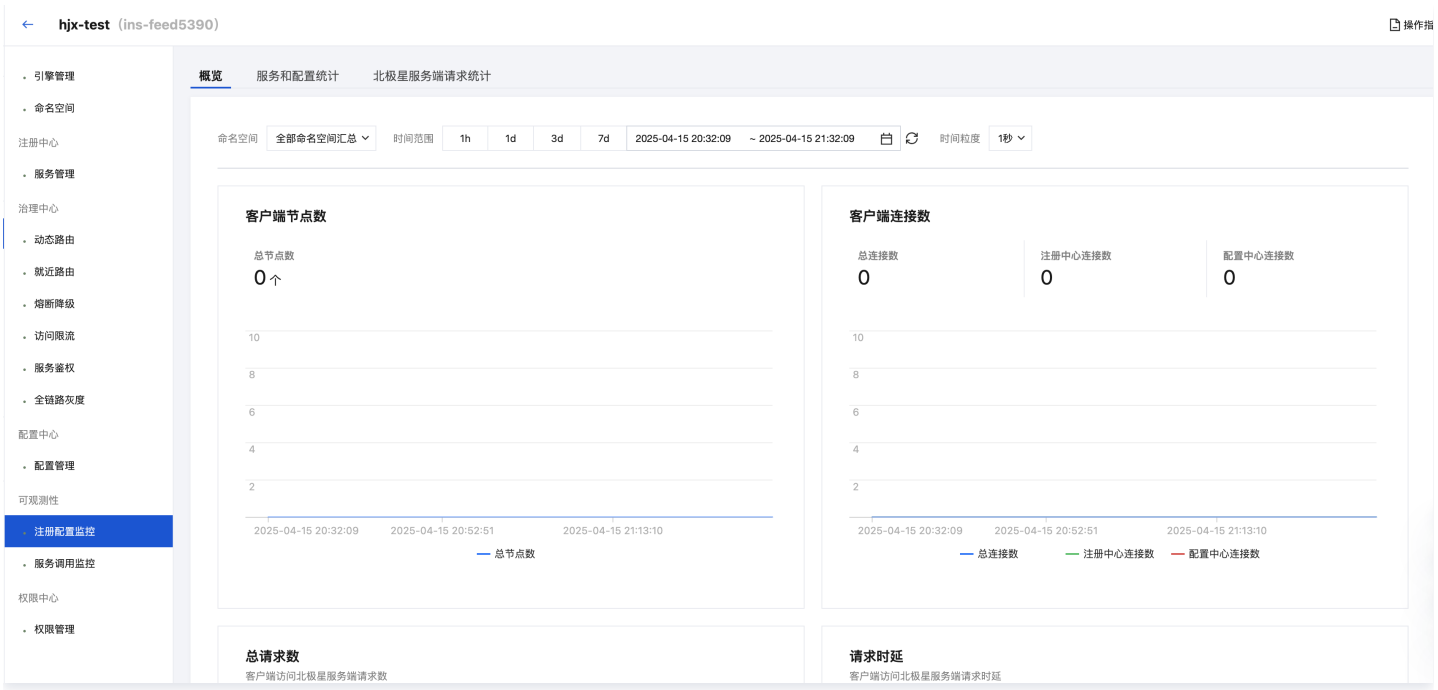
| 指标 | 说明 |
| --- | --- |
| 服务数 | 包含总服务数、在线服务数、异常服务数、离线服务数。 |
| 实例数 | 包含总实例数、在线实例数、隔离实例数、异常实例数。 |
| 配置分组数 | 配置分组总数。 |
| 配置数 | 包含配置文件数、已发布配置文件数。 |

### 北极星服务端请求统计

| 指标 | 说明 |
| --- | --- |
| 请求数 | 客户端访问北极星服务端请求数。包含总请求数、成功请求数、失败请求数、请求成功率。 |
| 请求时延 | 客户端访问北极星服务端请求时延。包含均值、最大值、最小值、P99、P95。 |
| 失败请求数 | 包含总失败请求数、5xx失败请求数、4xx失败请求数、网络失败数。 |
| 返回码分布 | 包含 200、5xx、4xx 返回码。 |

## 查看监控

1. 登录 TSF 控制台 。
2. 在左侧导航栏选择**Polaris（北极星）**后进入引擎实例列表页，单击目标实例 ID。
3. 在左侧页签选择**注册配置监控**，根据您的需求选择要查看的资源、设置好时间范围，即可查看对应的监控数据。


hjx-test（ins-feed5390）
操作指
引擎管理
命名空间
注册中心
服务管理
治理中心
动态路由
就近路由
熔断降级
访问限流
服务鉴权
全链路灰度
配置中心
配置管理
可观测性
注册配置监控
服务调用监控
权限中心
权限管理
概览
服务和配置统计
北极星服务端请求统计
命名空间
全部命名空间汇总
时间范围
1h
1d
3d
7d
2025-04-15 20:32:09 ~ 2025-04-15 21:32:09
时间粒度
1秒
客户端节点数
总节点数
0
10
8
6
4
2
2025-04-15 20:32:09
2025-04-15 20:52:51
2025-04-15 21:13:10
总节点数
客户端连接数
总连接数
0
注册中心连接数
0
配置中心连接数
0
10
8
6
4
2
2025-04-15 20:32:09
2025-04-15 20:52:51
2025-04-15 21:13:10
总连接数
注册中心连接数
配置中心连接数
总请求数
客户端访问北极星服务端请求数
请求时延
客户端访问北极星服务端请求时延

# 服务调用监控

最近更新时间：2025-04-17 16:00:53

Polaris（北极星）针对运行中实例的服务调用情况提供了多项监控指标，例如服务服务请求数、服务请求时延、返回码统计等指标，用以监测该实例及服务的运行情况。您可以根据这些指标实时了解北极星的运行状况，针对可能存在的风险及时处理，保障系统的稳定运行。本文为您介绍通过 TSF 控制台查看北极星服务调用数据的操作方法。

## 监控指标及说明

### 概览

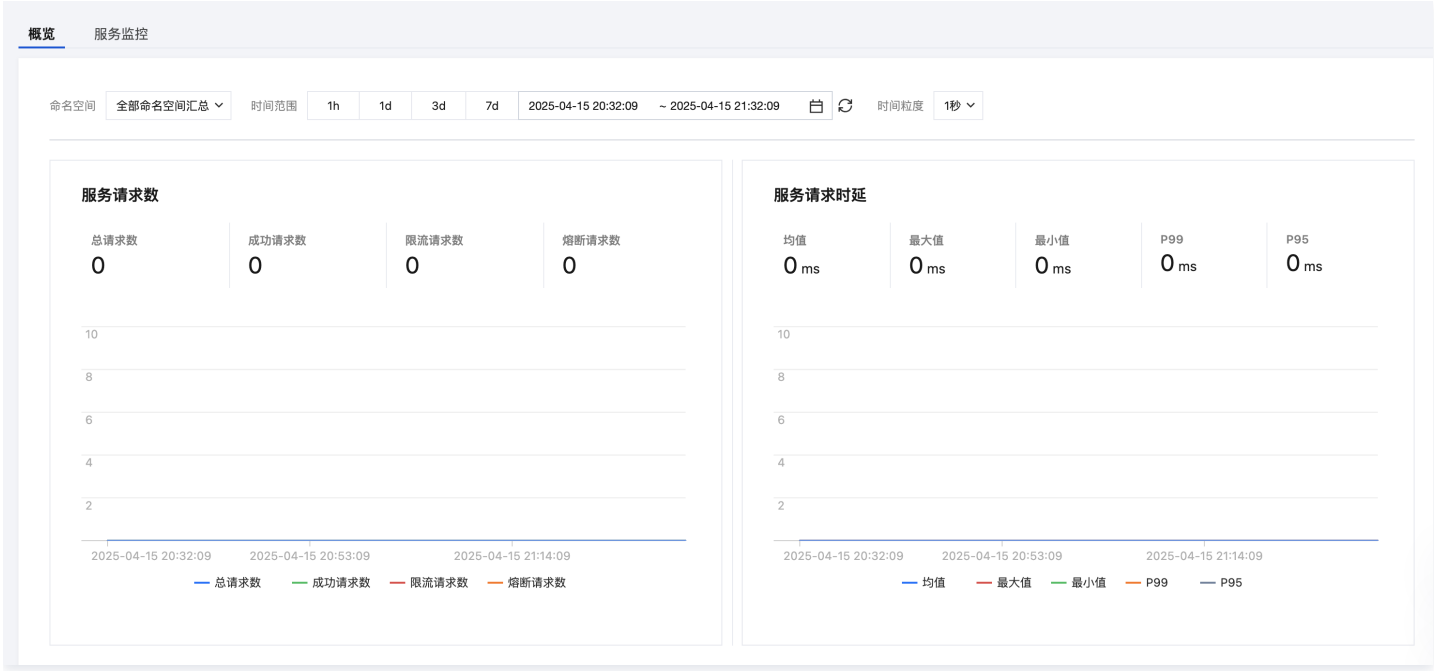
| 指标 | 说明 |
| --- | --- |
| 服务请求数 | 服务请求数量，包含总请求数、成功请求数、限流请求数、熔断请求数。 |
| 服务请求时延 | 服务请求时延，包含均值、最大值、最小值、P99、P95。 |
| 返回码统计 | 服务请求的返回码，包含 200、5xx、4xx 等返回码的次数及占比。 |
| 服务请求成功率分布 | 展示服务请求成功率在 100%、90%~100%、75%~90%、50%~75%、<50% 的分布情况。 |
| 服务请求时延分布 | 展示 P99 时延在 <5ms、5ms~10ms、10ms~50ms、50ms~100ms、>100ms 的分布情况。 |
| 服务列表 | 展示服务列表信息，包含服务名称、命名空间、健康/总实例数、成功率、总请求数、失败请求数等信息。 |

### 服务/接口/实例监控

| 指标 | 说明 |
| --- | --- |
| 服务请求数 | 根据筛选，按命名空间、服务、接口、实例等维度，查看服务/接口的请求数量，包含总请求数、成功请求数、限流请求数、熔断请求数。 |
| 服务请求时延 | 根据筛选，按命名空间、服务、接口、实例等维度，查看服务/接口的请求时延，包含均值、最大值、最小值、P99、P95。 |
| 返回码统计 | 包含 200、5xx、4xx 等返回码的次数及占比。 |
| 服务实例监控 | 展示服务实例监控信息，包含实例 IP、端口、监控状态、成功率、总请求数等信息。 |
| 调用者监控 | 展示服务调用者的监控信息，包含调用者 IP、服务名、命名空间、总请求数、成功率等信息。 |

## 查看监控

1. 登录 TSF 控制台
2. 在左侧导航栏选择**Polaris（北极星）**后进入引擎实例列表页，单击目标实例ID。
3. 在左侧页签选择**注服务调用监控**，根据您的需求选择要查看的资源、设置好时间范围，即可查看对应的监控数据。


概览
服务监控
命名空间
全部命名空间汇总
时间范围
1h
1d
3d
7d
2025-04-15 20:32:09 ~ 2025-04-15 21:32:09
时间粒度
1秒
服务请求数
总请求数
0
成功请求数
0
限流请求数
0
熔断请求数
0
10
8
6
4
2
2025-04-15 20:32:09
2025-04-15 20:53:09
2025-04-15 21:14:09
总请求数
成功请求数
限流请求数
熔断请求数
服务请求时延
均值
0 ms
最大值
0 ms
最小值
0 ms
P99
0 ms
P95
0 ms
10
8
6
4
2
2025-04-15 20:32:09
2025-04-15 20:53:09
2025-04-15 21:14:09
均值
最大值
最小值
P99
P95

# 监控排障

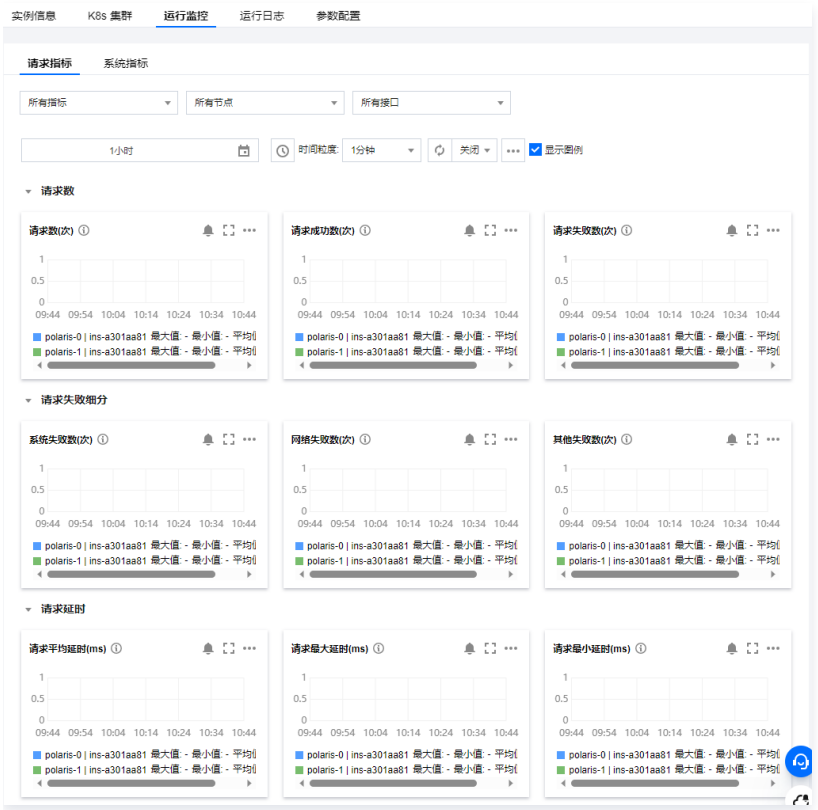
最近更新时间：2025-04-09 11:24:22

## 排障概述

本文主要介绍使用北极星进行排障的基础思路和操作步骤，方便运维人员在使用北极星的过程中进行问题定位和处理。
当您的客户端访问北极星报错时，可以查看北极星网格的监控，初步定位发生了什么错误。

## 1. 找到报错的接口

首先根据客户端的日志，定位到访问的是北极星的哪一个接口。然后进入北极星的监控页面，选择具体的接口，查看监控。如下图，选择**运行监控**，在下拉列表中选择您需要查看的接口。




目前北极星网格监控告警系统支持的接口如下表：

| 接口名 | 接口解释 |
| --- | --- |
| POST:/eureka/apps/{application} | eureka 客户端服务实例注册。 |
| GET:/eureka/apps | eureka 客户端查询全量服务实例。 |
| GET:/eureka/apps/delta | eureka 客户端增量拉取服务实例。 |
| GET:/eureka/apps/{application} | eureka 客户端查询某个服务全部实例。 |
| GET:/eureka/apps/{application}/{instanceid} | eureka 客户端查询某个服务下的某个实例。 |
| PUT:/eureka/apps/{application}/{instanceid} | eureka 客户端服务实例心跳上报。 |
| DELETE:/eureka/apps/{application}/{instanceid} | eureka 客户端服务实例反注册。 |
| PUT:/eureka/apps/{application}/{instanceid}/status | eureka 客户端服务实例状态变更。 |
| DELETE:/eureka/apps/{application}/{instanceid}/status | eureka 客户端删除服务实例变更。 |

| | |
|---|---|
| /v1.PolarisGRPC/RegisterInstance | grpc 客户端服务实例注册。 |
| /v1.PolarisGRPC/DeregisterInstance | grpc 客户端服务实例反注册。 |
| /v1.PolarisGRPC/Heartbeat | grpc 客户端服务实例心跳上报。 |
| /v1.PolarisGRPC/Discover | grpc 客户端服务发现。 |

例如使用下面的命令请求北极星，访问 eureka 接口注册实例。

```
curl -v -X POST -H "Accept: application/json" -H "Content-type: text/plain"
http://{$polaris_ip}:8761/eureka/apps/test -d '{"instance":{"instanceId":"test-1","securePort":
{"$":"0","@enabled":"false"},"ipAddr":"{$client_ip}}","status":"UP","port":{"$":"
{$client_port}}","@enabled":"true"},"hostName":"{$client_ip}"}}'
```

可以在监控页面，选择接口 `POST:/eureka/apps/{application}` ，看到这个请求的监控信息。

## 2. 根据监控信息定位问题

根据监控信息中的 `请求失败细分` 分组，可以初步定位报错的原因，目前包含以下错误类型：

- 其他错误：即北极星网格服务返回的错误码为 `4XX` 的错误，这类错误是入参校验失败，您需要检查您传入的参数和 http header 是否正确。如果您使用非 eureka 的方式接入，且打印了北极星网格服务返回的日志，可以检查日志，北极星网格返回包中带了入参校验失败的原因。
- 系统错误：即北极星网格服务返回的错误码为 `5XX` 的错误，出现这类错误表明北极星网格出现了异常，您需要提工单，联系腾讯云工程师解决。
- 网络错误：即北极星网格服务返回的错误码为 `负数` 的错误。典型的场景是在北极星网格接收客户端数据后，向客户端写数据时，客户端断开了连接，会产生这类错误。您需要检查下您的代码逻辑，是否提前关闭了连接。

# 配置告警

最近更新时间：2025-04-17 16:00:53

## 操作场景

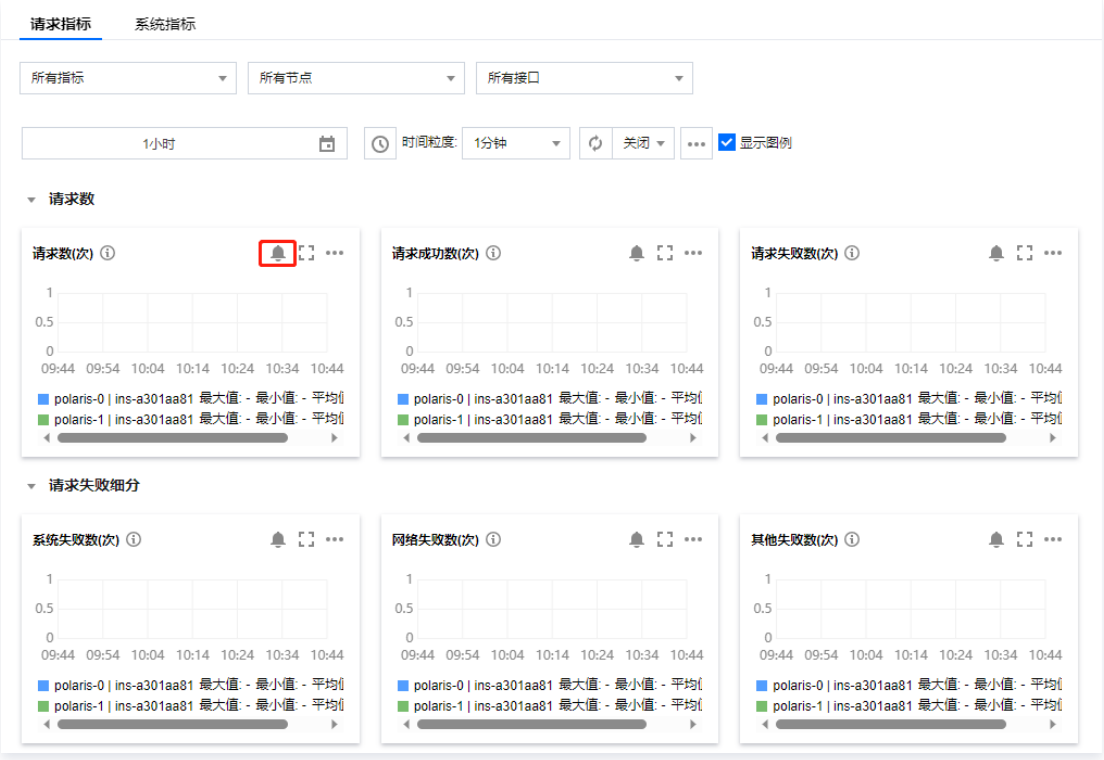
**注册配置治理中心**提供一些关键指标的配置告警功能，配置告警可帮助您及时发现问题并进行处理。本文为您介绍通过控制台配置告警的操作。

## 操作步骤

### 配置告警规则

创建的告警会将一定周期内监控的指标与给定阈值的情况进行比对，从而判断是否需要触发相关通知。当指标状态改变而导致告警触发后，您可以及时进行相应的预防或补救措施，合理地创建告警能帮助您提高应用程序的健壮性和可靠性。

1. 登录 TSF 控制台 。
2. 在**服务治理中心**下的**实例列表**页面，单击目标引擎的“ID”，进入基本信息页面。
3. 在顶部页签单击**运行监控**，在**请求指标**页面单击**下图告警按钮**跳转至 腾讯云可观测平台控制台 配置告警策略。




4. 在告警策略页面，选择好策略类型和要设置告警的实例，设置好告警规则和告警通知模板。
   - **策略类型**：选择**微服务引擎TSE/polarismesh**。
   - **告警对象**：选择需要配置告警策略的 TSE 资源。
   - **触发条件**：支持**选择模板**和**手动配置**，默认选择手动配置，手动配置参见以下说明，新建模板参见 新建触发条件模板 。

     **说明：**
     - 指标：例如“平均请求延迟”，选择统计粒度为1分钟，则在1分钟内，平均请求延迟连续N个数据点超过阈值，就会触发告警。
     - 告警频次：例如“每30分钟警告一次”，指每30分钟内，连续多个统计周期指标都超过了阈值，如果有一次告警，30分钟内就不会再次进行告警，直到下一个30分钟，如果指标依然超过阈值，才会再次告警。

   - **通知模板**：选择通知模板，也可以新建通知模板，设置告警接收对象和接收渠道。
5. 单击**完成**，完成配置。

**说明：**

> 有关告警的更多信息，请参见 腾讯云可观测平台告警服务 。

## 新建触发条件模板

1. 登录 腾讯云可观测平台控制台 。
2. 在左侧导航栏中，单击**告警管理**，在**策略管理**页签，点击新建告警策略。
3. 在**配置告警规则**下的触发条件，点击**选择模板**，点击**新增触发条件模板**。
4. 在新建模板页，配置策略类型。
   - **策略类型**：选择**微服务引擎TSE/polarismesh**，然后根据需要从**请求指标（接口级）/ 请求指标（节点级）/ 系统指标**中选择一个策略。
   - **使用预置触发条件**：勾选此选项，会出现系统建议的告警策略。
5. 确认无误后，单击**保存**。


6. 返回新建告警策略页，单击**刷新**，就会出现刚配置的告警策略模板。

# 事件中心

最近更新时间：2025-06-04 11:46:02

## 场景说明

事件中心将 Polaris（北极星）的服务状态变更事件数据进行统一管理和展示，您可以在事件中心查看服务状态变化详情，同时您也可以为服务事件配置告警通知规则，可帮助您及时发现问题并进行处理。
Polaris（北极星）当前支持的服务事件类型有：

- 实例上线/下线
- 实例恢复健康
- 实例出现异常
- 实例开启、关闭隔离
- 无损上线开始、结束
- 无损预热开始、结束
- 无损下线开始
- 进程结束
- 熔断触发、恢复
- 熔断到半开
- 限流恢复、恢复

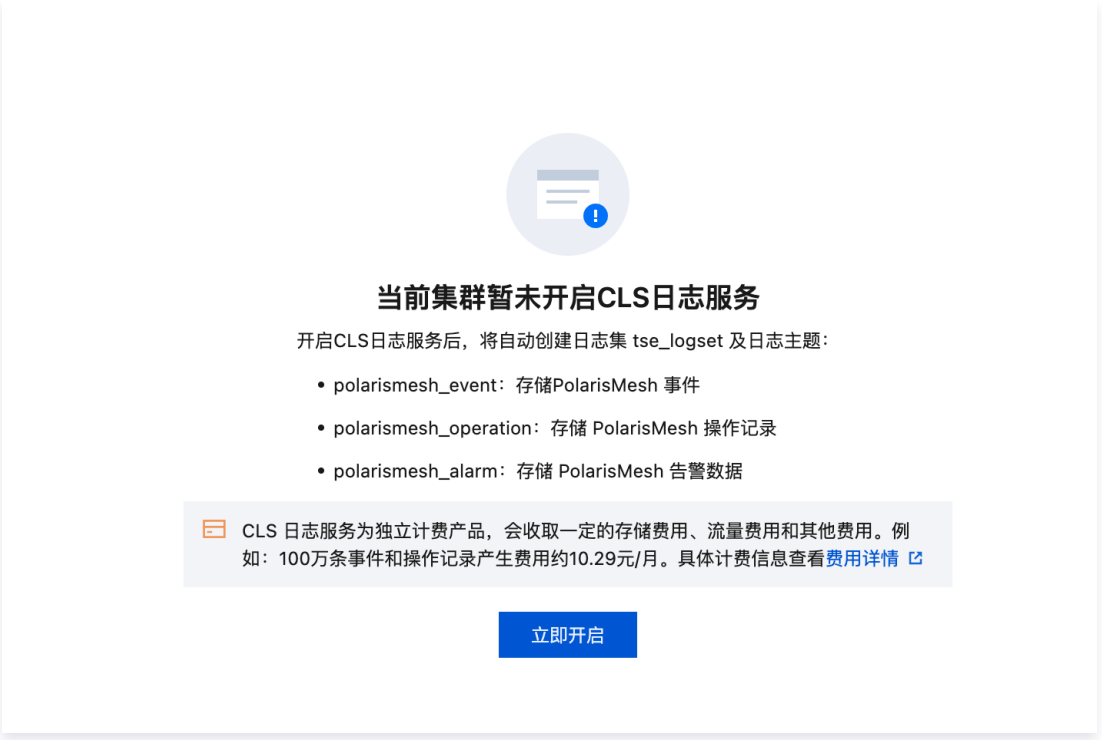
本文介绍如何在 TSF 控制台上查看服务事件详情和配置事件告警规则。

## 前提条件

Polaris（北极星）事件中心、操作记录、告警数据存储在 CLS 日志服务。使用该能力前，您需要在 Polaris 中开启该能力。开启后自动创建日志集 tse_logset，及主题：polarismesh_event、 polarismesh_operation、polarismesh_alarm，您在控制台上可以查询到这些事件、操作记录和告警配置。
开启方案如下，选择其一即可：

- 方案一：您可在 购买 Polaris（北极星）实例 时开启。


- 方案二：直接在**事件中心**页面开启，登录 TSF 控制台，在引擎实例列表页面，单击目标引擎的“ID”，进入基本信息页面，在顶部页签选择**事件中心**，单击**立即开启**。




- **方案三：创建引擎实例后，在左侧导航栏单击的目标引擎实例列表，进入详情页后，在左侧标签栏选择引擎管理 > 运行日志 > CLS日志，单击立刻开启按钮开启 CLS 日志服务。**


## 操作步骤

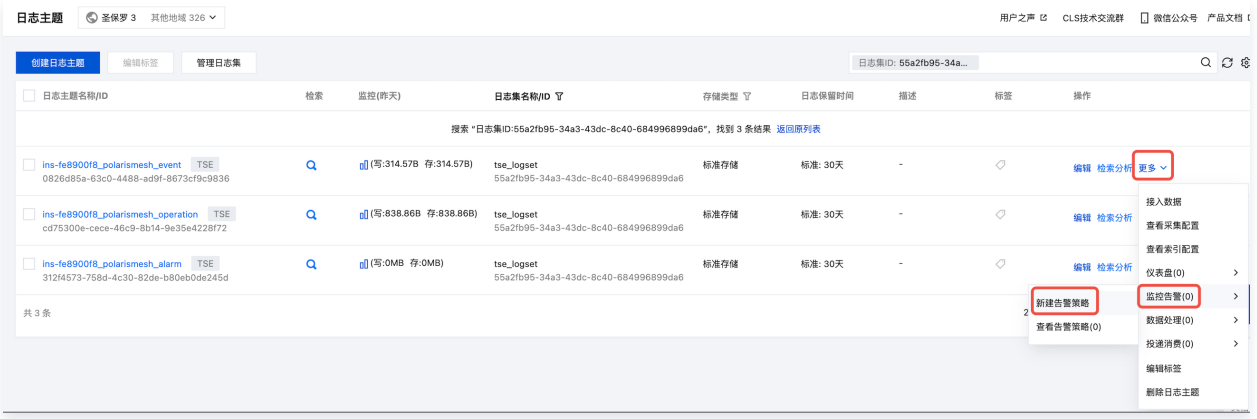
### 查看事件详情

1. 登录 TSF 控制台 。
2. 在引擎实例列表页面，单击目标引擎的“ID”，进入基本信息页面。
3. 在顶部页签选择**事件中心**，选择好时间范围（支持近7天、近30天和自定义时间范围），可查看发生的事件详情，包括时间发生服务对应实例 IP 和事件发生的时间。


## 配置告警规则

1. 登录 TSF 控制台 。
2. 在引擎实例列表页面，单击目标引擎的“ID”，进入基本信息页面。
3. 在顶部页签选择**运行日志**，选择**CLS日志**。
4. 选择日志类型为“事件”对应的**日志集**，跳转至**日志服务**控制台。


5. 跳转至 CLS 控制台进行告警策略配置。
6. 选择日志主题名称后缀为“polarismesh_even”的日志主题，在操作栏单击更多，选择：监控高级-新建告警策略。




7. 在**新建告警策略**页面进行告警策略配置，请参考**日志服务-配置告警策略** 进行告警策略配置。


**各事件对应的 event_type 如下：**

| 事件大类 | 事件子类 | 事件名称 | event_type |
| --- | --- | --- | --- |
| 注册中心 | 服务实例事件 | 实例上线 | InstanceOnline |
| | | 实例下线 | InstanceOffline |
| | | 实例出现异常 | InstanceTurnUnHealth |
| | | 实例恢复健康 | InstanceTurnHealth |
| | 无损上下线及预热 | 无损上线开始 | LosslessOnlineStart |
| | | 无损上线结束 | LosslessOnlineEnd |
| | | 服务预热开始 | LosslessWarmupStart |
| | | 服务预热结束 | LosslessWarmupEnd |
| | | 无损下线开始 | LosslessOfflineStart |
| | | 进程结束 | InstanceThreadEnd |
| 服务治理 | 故障熔断 | 熔断触发 | CiruitBreakerOpen |
| | | 熔断到半开 | CircuitBreakerHalfOpen |
| | | 熔断恢复 | CircuitBreakerClose |
| | 服务限流 | 限流触发 | RateLimitStart |
| | | 限流恢复 | RateLimitEnd |

# 操作记录

最近更新时间：2025-04-17 16:00:53

操作记录将 Polaris（北极星）的控制台操作以及 OpenAPI 调用产生的操作信息进行统一管理和展示，您可以在操作记录查看详情，同时您也可以为操作记录配置告警通知规则，可帮助您及时发现问题并进行处理。

> **注意：**
> 此文档仅针对北极星引擎实例版本 1.14.0.0 以及更高版本。

Polaris（北极星）当前支持的操作记录资源类型有：

- 命名空间
- 服务
- 服务实例
- 路由规则
- 用户
- 用户组
- 鉴权策略
- 配置分组
- 配置文件
- 限流规则
- 熔断规则

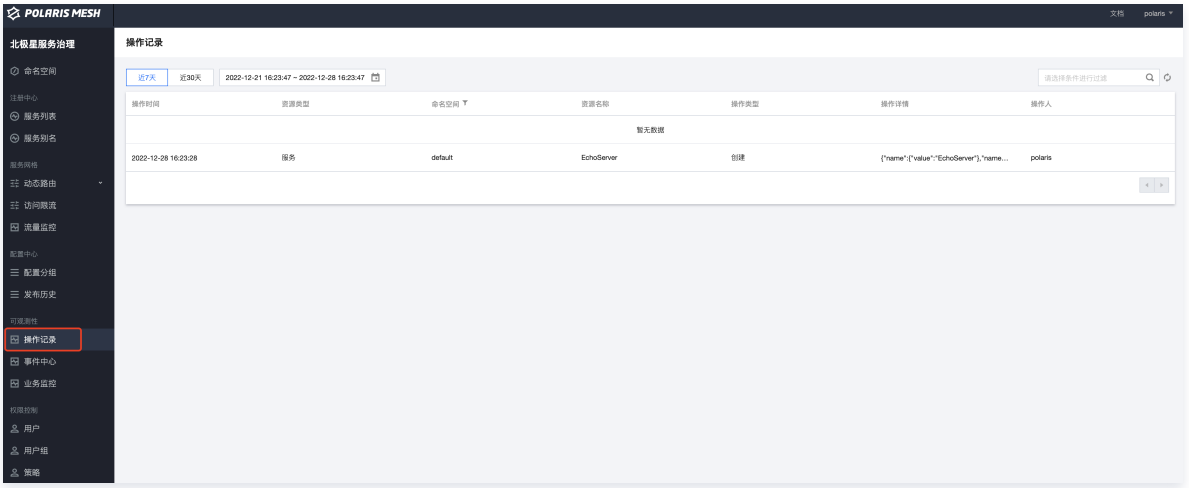
本文介绍如何在 TSF 控制台上查看操作记录详情和配置操作记录告警规则。

## 操作步骤

### 查看操作记录详情

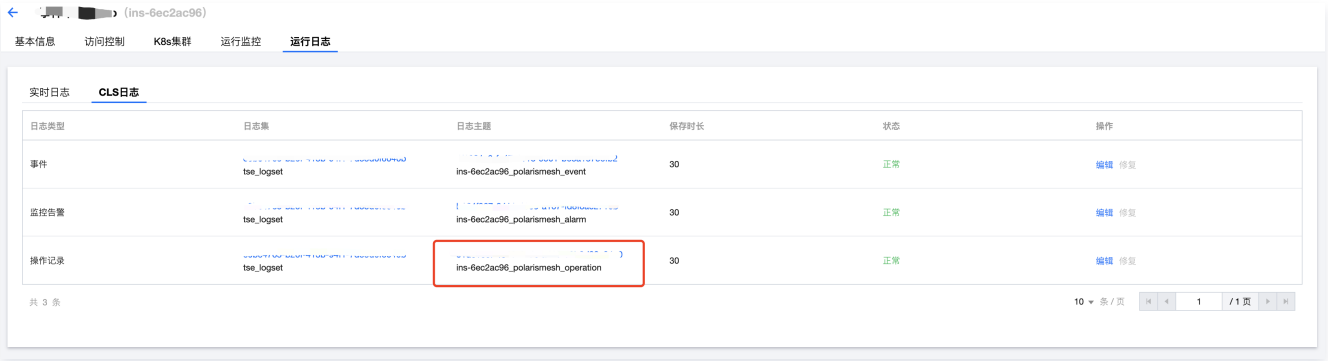
1. 登录 TSF 控制台 。
2. 在左侧导航栏选择 **Polaris（北极星）**，进入引擎实例列表页。
3. 单击目标实例引擎操作栏的**控制台**，输入用户名和密码，进入 POLARIS MESH 控制台页面。


4. 在左侧导航栏选择**操作记录**。您可以查看通过北极星控制台或者 OpenAPI 的操作记录信息。

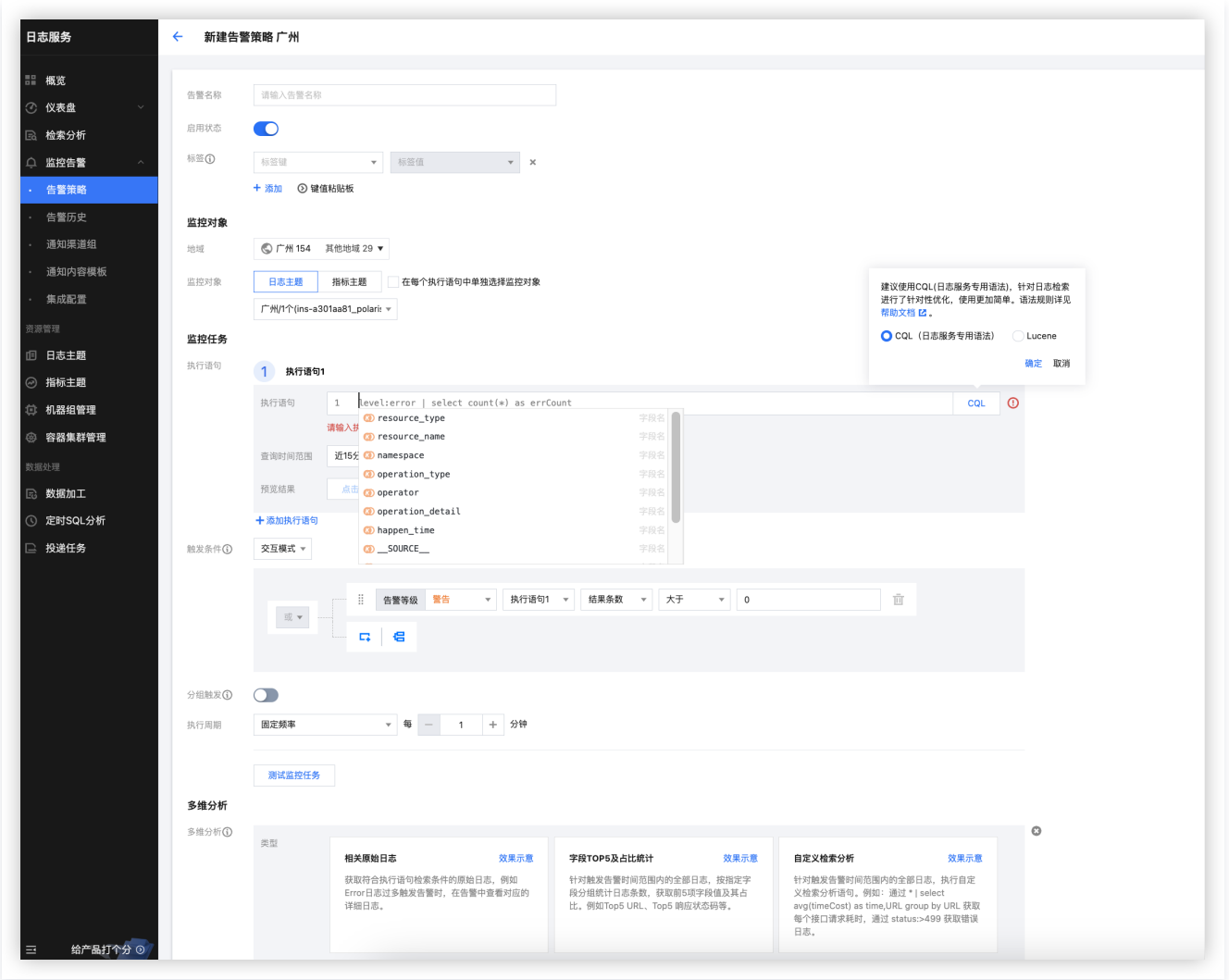


## 配置告警规则

1. 登录 TSF 控制台 。
2. 在左侧导航栏选择 **Polaris（北极星）**，进入引擎实例列表页。
3. 单击目标引擎实例，进入实例信息页后单击引擎管理。
4. 单击**运行日志**，切换到 CLS 日志的页面 tab 后，单击日志类型为操作记录的日志主题。




5. 单击日志服务的告警策略，单击**新建策略**，然后针对操作记录日志主题进行配置告警。



# 权限控制（权限中心）

最近更新时间：2025-04-17 16:00:53

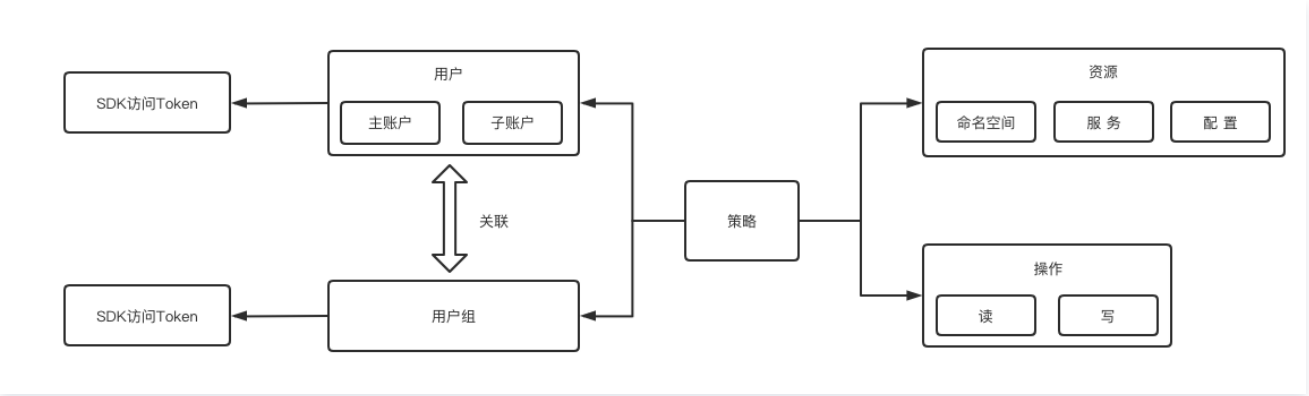
## 基本概念

北极星场景下与账号和用户相关的概念如下：

- **主账号**：也可称为主用户，是所有北极星资源的拥有者，对所有北极星资源有操作权限。在创建 TSF 引擎的时候，TSF 默认会创建名称为 **polaris** 的主账户。
- **子账号**：主账户创建的协作账户。
- **用户组**：北极星的用户组概念，是一组具有相同权限的用户。主账号可以通过创建用户组批量对用户进行授权与管理。**用户组只能由主账号创建和授权。**
- **授权**：主用户可以管理所有子用户以及用户分组的写权限。子用户可将自己拥有写权限的资源权限分配给其他子用户或者用户组。
- **token**：资源权限将通过 token 进行控制和管理，用户与用户组均可生成。

## 操作场景

使用 Polarismesh(北极星) 的鉴权功能，您可以清晰地管理资源与用户的访问权限。北极星基于**命名空间**和**服务**资源维度实现权限管控。
北极星引入策略的概念，将资源的访问权限与不同的用户角色联系到一起。例如下图，




子账号和用户组默认没有写资源的权限，需要主账号为其授权，才能使子账号/用户组具有指定资源的操作权限。
本文指导您在 TSF 控制台 通过开启资源鉴权和新建策略，在主账号、子账号、资源视角下，为用户或用户组授予资源权限。

## 前提条件

主账号至少拥有一个 Polarismesh 引擎实例。若您没有，请前往 创建引擎 。

## 操作步骤

### 步骤1：开启鉴权模式

1. 登录 TSF 控制台 。
2. 在**北极星网格**下的 **polarismesh** 页面，单击已有的引擎实例的“ID”，进入基本信息管理页面。

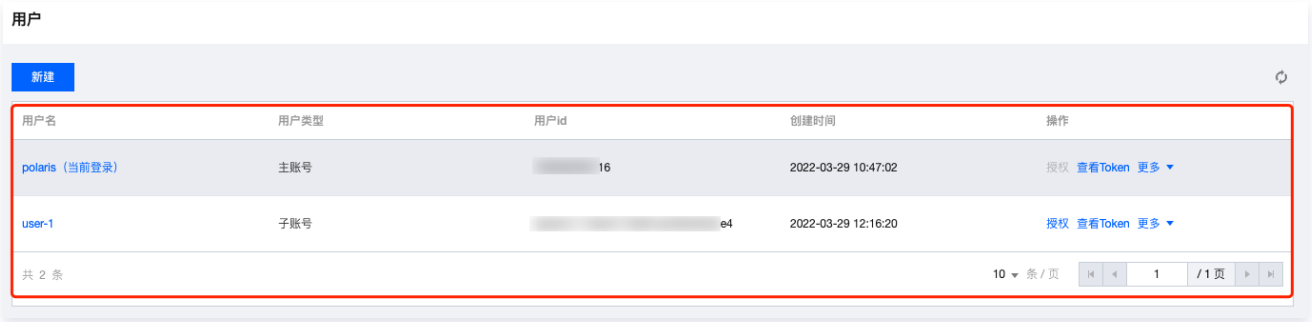
3. 在基本信息页面的**鉴权模块**，单击**服务鉴权**的开关按钮，在弹窗中单击**确定**开启服务鉴权。


4. 开启服务鉴权后，子账号或用户组进行访问控制台、访问北极星资源等操作时，都会需要使用 token 鉴权，没有配置 token 的客户端将无法注册服务。

## 步骤2：创建子账号

1. 在 **polarismesh** 列表页面，单击目标实例引擎操作栏的**控制台**，输入用户名和密码，登录北极星控制台。


2. 进入北极星控制台后，在左侧导航栏选择**用户**，单击**新建**。
3. 输入子账户信息后，单击**确定**，完成子账户创建。



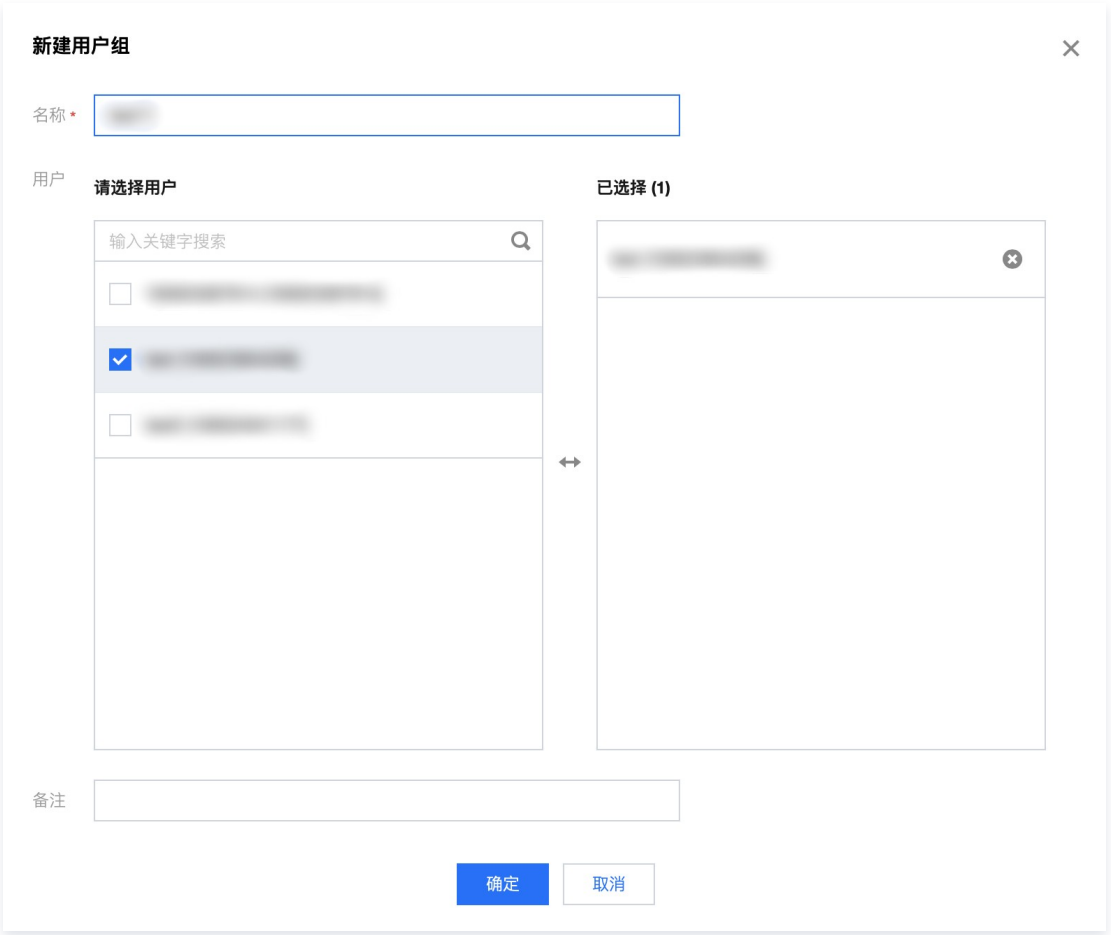

4. 主账号可在**权限控制**页面中查看所有用户或用户组的 token，并且对用户、用户组和权限策略做管理和编辑。

## 步骤3：关联用户组

> **说明**
> **只有主账号才可对用户组进行编辑操作。**

1. 在北极星控制台的左侧导航栏选择**用户组**，进入用户组列表页面，单击**新建用户组**。


2. 根据您的需要，为用户组命名，并且关联子账号。




## 步骤4：授权操作

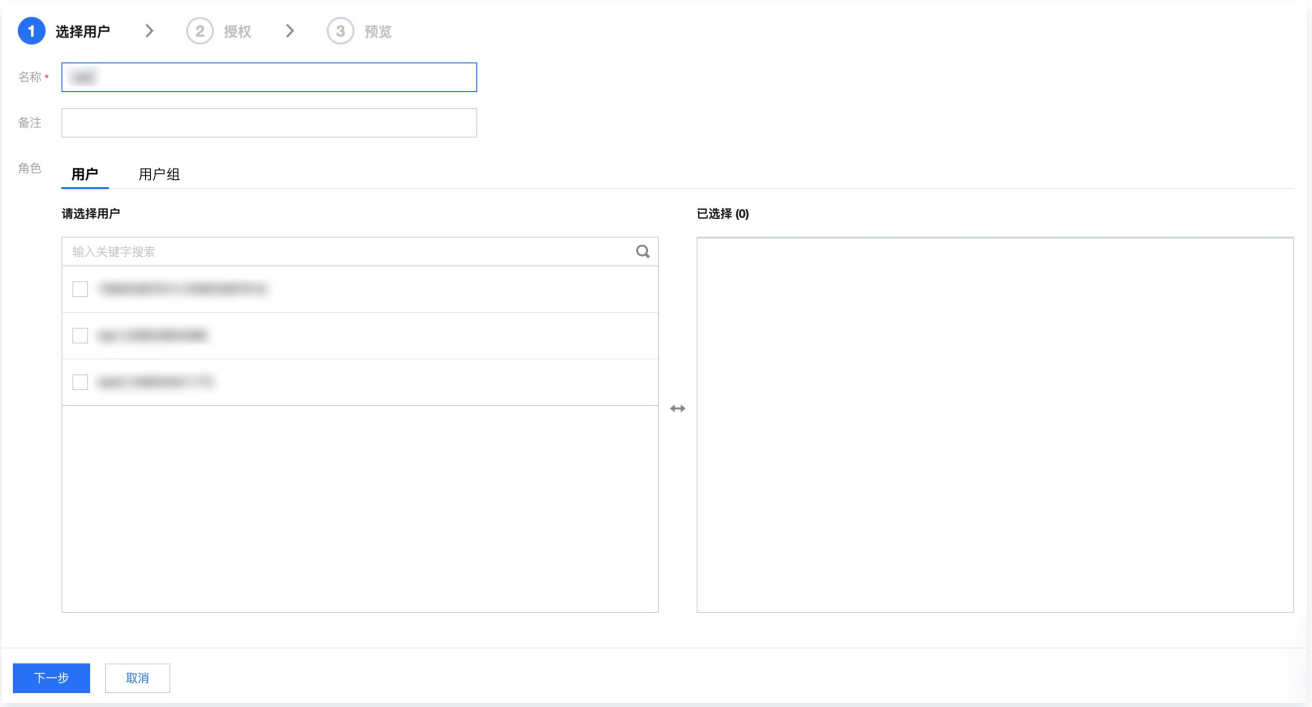
您可以通过以下列表查看不同视角的授权操作。

### 主账号视角

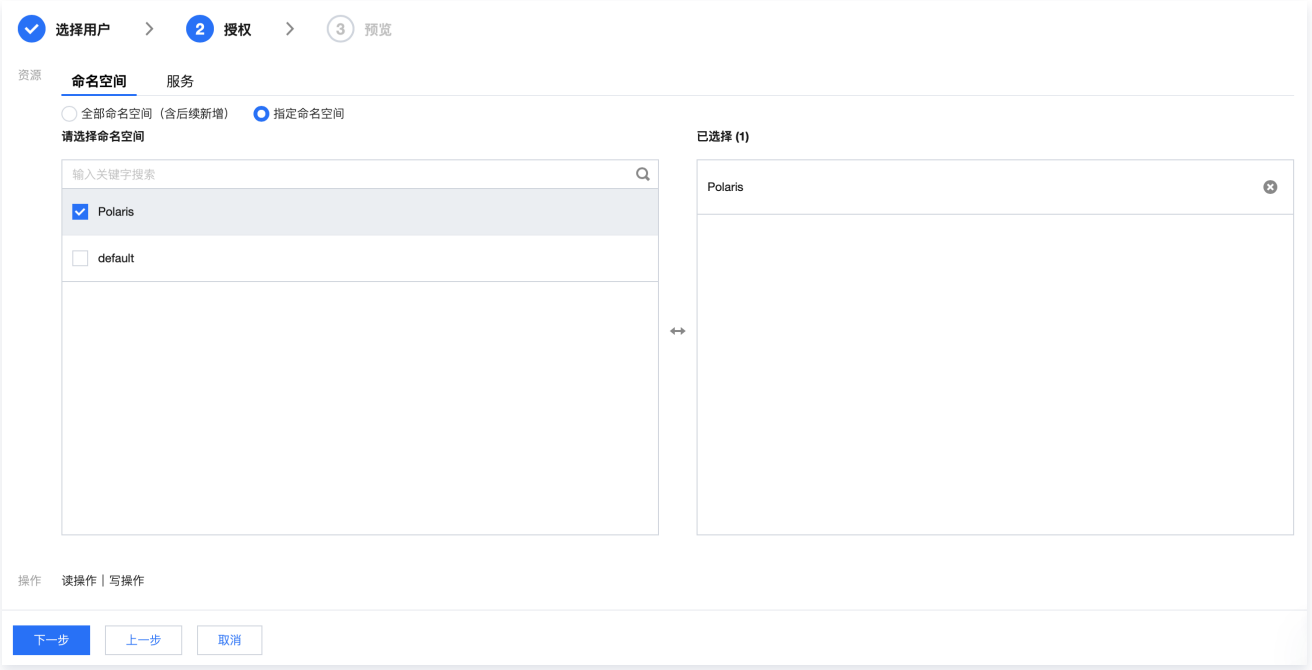
> **说明**
> 请确保已 开启服务鉴权 。

1. 使用主账号登录北极星控制台。
2. 在左侧导航栏选择**权限管理** > **策略**，进入权限管理页面，单击新建策略。

3. 填写策略基本信息，在角色栏可以选择需要授权用户或用户组，单击**下一步**选择授权的资源。




4. 在资源栏可选择北极星的资源类型，包括命名空间、服务等资源，主账号可对所有资源进行操作。




5. 单击**下一步**，进入预览界面，详细展示该策略涉及的用户、用户组以及资源。确认信息无误后，单击**完成**。

6. 主账号可在权限策略列表查阅现有的权限策略，可单击**编辑**进行授权或删除等操作。


### 子账号视角

1. 仅在主账号开启了**鉴权**模式下，使用子账户进入北极星控制台，左侧的导航栏才会出现**策略**选项。
2. 子用户**无法**新建和编辑权限策略，只能查看现有权限策略。


3. 子用户可将自己拥有写权限的资源权限分配给其他子用户或者用户组。具体操作请参见**资源视角**。

### 资源视角

1. 仅在主账号开启了**鉴权**模式下，用户在创建、编辑**命名空间**或**服务**时，才可将该资源授权给其他用户或者用户组。接下来以**命名空间**为例，指导您如何在资源视角给用户授权。
2. 在 TSE 控制台 > **polarismesh**，选择已存在的北极星引擎实例，进入北极星控制台。在左侧导航栏中，选择**命名空间**，进入命名空间列表管理页面。


3. 单击**新建**，进入新建命名空间页面，根据您的需要填写信息，并且可在**高级设置**中选择需要授权的用户或用户组。


4. 单击**提交**，则完成创建命名空间且选择的用户或用户组对该命名空间具有操作权限。
5. 对于已经存在的命名空间，单击**编辑**，可以修改授权的用户或用户组。

## 注意事项

1. 只有主账号开启服务鉴权功能后，才能在控制台查看**策略**列表。
2. 资源隔离性：
   - **所有子用户，对主账号的资源都默认具备读权限。**
   - **所有子用户或者用户分组，都默认获得对所创建的资源的写权限。**
   - 子用户可以获得所在的用户分组的所有资源写权限。
3. 每个用户初始都有一个策略**默认策略**。
   - 子用户没有创建资源的情况下，默认策略中资源为空。
   - 用户创建资源后，自动添加资源至默认策略。
   - 删除资源时，引擎将自动把该资源从默认策略中删除。
4. 如果之前向 Polarismesh 引擎实例导入过腾讯云子账户，建议尽快使用北极星主账户修改已有子账户的登录密码。已有子账户的登录信息如下。
   - 用户名：**{ 腾讯云子账户用户名称 }**
   - 密码：polarismesh@2022

# Java 应用开发
# Spring Cloud 应用开发
# Spring Cloud Tencent 概述

最近更新时间：2024-02-21 14:51:31

目前 Spring Cloud Tencent 主要提供了微服务领域常见的服务注册与发现、配置中心、服务路由、限流熔断等服务治理能力。 Spring Cloud Tencent 已对外开源，更多信息请参见 Spring Cloud Tencent 。


## Spring Cloud Tencent 功能与版本对应列表

| 功能大项 | 功能点 | 功能分组 | 支持版本号 |
| --- | --- | --- | --- |
| 服务注册与发现 | 服务注册 | | 1.0.0+ |
| | 服务发现 | | 1.0.0+ |
| | 心跳上报 | | 1.0.0+ |
| | 自定义健康检查接口 | | 1.1.4+ |
| | 服务列表 | | 1.1.4+ |
| | 多注册多发现 | Consul | 1.2.0+ |
| | | Nacos | 1.9.0+ |
| | 负载均衡 | 权重随机 | 1.0.0+ |
| | | 哈希环 | 1.11.0+ |
| 动态配置 | 动态配置获取与更新 | 远端配置 | 1.2.0+ |
| | | 本地配置 | 1.8.0+ |
| | 默认文件分组读取（bootstrap 和 application） | | 1.5.0+ |
| | 配置变更监听器 | | 1.6.0+ |
| | 配置更新配置方式 | | 1.2.0+ |
| 服务限流 | 限流方式 | 单机限流 | 1.0.0+ |
| | | 分布式限流 | 1.4.0+ |
| | 限流行为 | 快速失败 | 1.0.0+ |

| | | | |
|---|---|---|---|
| | | 匀速排队 | 1.5.0+ |
| | 触发方式 | Servlet | 1.0.0+ |
| | | Reactive | 1.1.0+ |
| | 标签限流 | | 1.3.0+ |
| | 第二代标签解析 | | 1.8.0+ |
| | 动态标签解析 | | 1.5.0+ |
| | 自定义限流响应 | 本地静态配置 | 1.4.0+ |
| | | 代码SPI | 1.11.0+ |
| 服务熔断 | 熔断判断 | 错误码 | 1.0.0+ |
| | | 时延 | 1.11.0+ |
| | 熔断级别 | 实例级熔断 | 1.0.0+ |
| | | 接口级熔断 | 1.10.0+ |
| | | 服务级熔断 | 1.10.0+ |
| | 触发方式 | Feign | 1.0.0+ |
| | | RestTemplate | 1.6.0+ |
| | | WebClient | 1.11.0+ |
| | | SCG | 1.11.0+ |
| | | Zuul | 1.11.3+ |
| 服务路由 | 路由方式 | 自定义路由 | 1.0.0+ |
| | | 元数据路由 | 1.5.0+ |
| | | 就近路由 | 1.5.0+ |
| | 动态标签解析 | | 1.5.0+ |
| | 第二代标签解析 | | 1.8.0+ |
| | SCG 支持服务路由 | | 1.6.0+ |
| | Zuul 支持服务路由 | | 1.7.1+ |
| | spring-retry 支持服务路由 | | 1.8.0+ |
| RPC增强 | Feign 调用增强 | | 1.7.0+ |
| | RestTemplate 调用增强 | | 1.11.0+ |
| | WebClient 调用增强 | | 1.11.0+ |
| | SCG 转发增强 | | 1.11.0+ |
| | Zuul 转发增强 | | 1.11.3+ |
| 元数据传递 | 读取方式 | 本地配置读取 | 1.0.0+ |
| | | 代码 SPI 读取 | 1.0.0+ |
| | 传递方式 | 链路传递 | 1.0.0+ |

| | | | |
|---|---|---|---|
| | | 单跳传递 | 1.7.0+ |
| | | 原始头部传递 | 1.8.0+ |
| | 接收方式 | Servlet | 1.0.0+ |
| | | Reactive | 1.0.0+ |
| | 传递途径 | Feign | 1.0.0+ |
| | | RestTemplate | 1.0.0+ |
| | | Zuul | 1.1.0+ |
| | | SCG | 1.1.0+ |
| | | WebClient | 1.11.0+ |
| 场景化插件 | 测试环境路由插件 | | 1.7.0+ |
| | SCG 流量染色插件 | | 1.7.0+ |
| | 其他注册中心适配器插件 | | 1.11.0+ |
| Actuator Endpoint扩展 | 服务实例列表 | | 1.6.0+ |
| | 动态配置 | | 1.6.0+ |
| | 服务限流规则 | | 1.6.0+ |
| | 服务路由规则 | | 1.7.0+ |
| 监控数据上报 | 采集方式 | Feign | 1.7.0+ |
| | | RestTemplate | 1.7.0+ |
| | | WebClient | 1.11.0+ |
| | | SCG | 1.11.0+ |
| | | Zuul | 1.11.3+ |
| | 上报方式 | 主动上报 | 1.7.1+ |
| | | 被动拉取 | 1.7.1+ |

# Spring Cloud 版本支持

最近更新时间：2025-04-15 16:59:52

## 背景

Spring Cloud TSF/Tencent 为实现了 Spring Cloud 标准微服务 SPI 的服务治理组件，因此其不包含服务调用组件相关的功能。换句话说，用户自身的应用本身可能带有服务调用组件，Spring Cloud TSF/Tencent 能提供基于 Spring Cloud 标准接口的服务调用中的服务治理功能。因为 Spring Cloud TSF/Tencent 实现的是 Spring Cloud 定义的标准接口，而开源的 Spring Cloud 有不同版本的提供，因此 Spring Cloud TSF/Tencent 针对不同版本的 Spring Cloud 也需要对应的支持。
与此同时，Spring Cloud TSF/Tencent 也存在自身的版本规划，以便用户根据自身应用和北极星服务端版本，选择对应的 Spring Cloud TSF/Tencent 版本。所以需要该文档指定 Spring Cloud TSF/Tencent 与 Spring Cloud 的版本对应关系。

## Spring Cloud TSF/Tencent 版本适配规范

Spring Cloud TSF/Tecnent 每1-2年会发布一个大版本（前2位版本号），例如：2022年发布 Spring Cloud TSF 1.46，2024年发布 Spring Cloud Tencent 2.0。Spring Cloud TSF/Tencent 对 Spring Cloud 开源版本的适配策略：

- 默认适配发布年份**前两年**的 Spring Cloud 开源版本，例如：2024年发布的 Spring Cloud Tencent 2.0 默认适配 Spring Cloud 2022。
- 如有必要，最早适配发布年份**前五年**的 Spring Cloud 开源版本，例如：2024年发布的 Spring Cloud Tencent 2.0 最早适配 Spring Cloud Hoxton。
- 在每个版本 EOS 之前（不含 EOS 当年），按需适配上一年到发布年份的 Spring Cloud 开源版本。

## Spring Cloud TSF/Tencent 与服务端版本适配

- 1.x：支持 TSF-Consul。
- 2.x：支持北极星和 TSF-Consul。

## Spring Cloud TSF/Tencent 版本适配列表

### Spring Cloud TSF 1.46

| Spring Cloud 版本 | 2022年 | 2023年 | 2024年 | 2025年（EOS） |
| --- | --- | --- | --- | --- |
| Finchley | 按需适配 | 按需适配 | 按需适配 | 按需适配 |
| Greenwich | 按需适配 | 按需适配 | 按需适配 | 按需适配 |
| Hoxton | 按需适配 | 按需适配 | 按需适配 | 按需适配 |
| **2020** | **默认适配** | **默认适配** | **默认适配** | **默认适配** |
| 2021 | 按需适配 | 按需适配 | 按需适配 | 按需适配 |
| 2022 | - | 按需适配 | 按需适配 | 按需适配 |
| 2023 | - | - | 按需适配 | 按需适配 |
| 2024 | - | - | - | 不适配 |

### Spring Cloud Tencent 2.0

| Spring Cloud 版本 | 2024年 | 2025年 | 2026年 | 2027年（EOS） |
| --- | --- | --- | --- | --- |
| Hoxton | 按需适配 | 按需适配 | 按需适配 | 按需适配 |
| 2020 | 按需适配 | 按需适配 | 按需适配 | 按需适配 |
| 2021 | 按需适配 | 按需适配 | 按需适配 | 按需适配 |
| **2022** | **默认适配** | **默认适配** | **默认适配** | **默认适配** |
| 2023 | 按需适配 | 按需适配 | 按需适配 | 按需适配 |

| | | | | |
|---|---|---|---|---|
| 2024 | - | 按需适配 | 按需适配 | 按需适配 |
| 2025（如有） | - | - | 按需适配 | 按需适配 |
| 2026（如有） | - | - | - | 不适配 |

# Spring Cloud TSF/Tencent 版本号规范

## Spring Cloud TSF

Spring Cloud TSF 采用3位版本号，即 **a.b.c-xyz-RELEASE**，其中每位版本号含义如下：

- **a.b**：和集成的北极星服务端的前两位版本号保持一致。
- **c**：需求开发和缺陷修复版本号。如果有需求开发或缺陷修复，增加此版本号。
- **xyz**：和集成的 Spring Cloud 大版本号保持一致。

示例：

- 版本号 1.46.11-SpringCloud2021-RELEASE 表示对接 1.46.x 版本的 TSF Consul 服务端和 Spring Cloud 2021 版本。
- 如果 Spring Cloud TSF 有需求开发或缺陷修复，TSF Consul 服务端没有变化，版本号变为 1.46.12-SpringCloud2021-RELEASE。

## Spring Cloud Tencent

Spring Cloud Tencent 采用4位版本号，即 **a.b.c.d-xyz**，其中每位版本号含义如下：

- **a.b**：和集成的北极星服务端的前两位版本号保持一致。
- **c**：需求开发版本号。如果有需求开发，增加此版本号。
- **d**：缺陷修复版本号。如果仅有缺陷修复，增加此版本号。
- **xyz**：和集成的 Spring Cloud 版本号保持一致。

示例：

- 版本号 2.0.0.0-2022.0.5 表示对接 2.0.x.x 版本的北极星服务端和 Spring Cloud 2022.0.5 版本。
- 如果 Spring Cloud Tencent 有需求开发，北极星服务端没有变化，版本号变为 2.0.1.0-2022.0.5。
- 如果 Spring Cloud Tencent 有缺陷修复，北极星服务端没有变化，版本号变为 2.0.1.1-2022.0.5。
- 如果 Spring Cloud Tencent 没有变化，北极星服务端需求开发版本号变化：版本号变为 2.0.2.0-2022.0.5。
- 如果 Spring Cloud Tencent 没有变化，北极星服务端缺陷修复版本号变化：版本号变为 2.0.2.1-2022.0.5。

## 相关链接

Spring Cloud Tencent 版本管理
Spring Cloud Tencent 版本发布页

# Spring Cloud 接入

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文通过一个 demo 进行 Spring Cloud 应用接入微服务引擎托管的 PolarisMesh 治理中心的全流程操作演示，帮助您快速了解如何使用北极星网格。

## 前提条件

- 已创建 Polaris 北极星实例，请参见 创建 PolarisMesh 治理中心 。
- 下载 Github 的 demo 源码 到本地并解压。
- 本地编译构建打包机器环境已安装了Java JDK、Maven，并且能够访问 Maven 中央库。
- 根据您自身的业务，已准备好业务部署的资源， 虚拟机部署 、 容器化部署 选择其中一种方式即可。
  - **虚拟机部署**已创建 CVM 虚拟机，请参见 创建 CVM 虚拟机 。
  - **容器化部署**已创建 TKE 容器集群，请参见 创建 TKE 集群 。

## 操作步骤

1. 登录 TSF 控制台 。
2. 在左侧导航栏选择 **Polaris（北极星）**，进入北极星实例引擎列表页后单击目标实例进入实例信息页。
3. 在引擎管理-实例信息页查看访问地址，Spring Cloud 应用访问使用 gRPC 端口（8091）：


4. 修改 demo 中的注册中心地址。

   4.1 在下载到本地的 demo 源码目录 下，分别找到

   ```
   spring-cloud-tencent-examples/polaris-discovery-example/discovery-callee-service/src/main/resources/bootstrap.yml
   ```

和

```
spring-cloud-tencent-examples/polaris-discovery-example/discovery-caller-service/src/main/resources/bootstrap.yml
```

两个文件。

4.2 添加微服务引擎北极星网格地址到项目配置文件中（以

```
spring-cloud-tencent-examples/polaris-discovery-example/discovery-callee-service/src/main/resources/bootstrap.yml
```

为例）。

```
server:
  port: 0
spring:
  application:
    name: DiscoveryCallerService
  cloud:
    polaris:
      enabled: true
      address: grpc://{北极星引擎内网地址 IP}:8091
```

5. 打包 demo 源码成 Jar 包。

5.1 在 `polaris-discovery-example` 源码根目录下，打开 cmd 命令，执行 mvn clean package 命令，对项目进行打包编译。

5.2 编译成功后，生成如下表所示的2个 Jar 包。

| 软件包所在目录 | 软件包名称 | 说明 |
| --- | --- | --- |
| polaris-discovery-example/discovery-callee-service/target | discovery-callee-service-${version}-SNAPSHOT.jar | 服务生产者 |
| polaris-discovery-example/discovery-caller-service/target | discovery-caller-service-${version}-SNAPSHOT.jar | 服务消费者 |

6. 部署 provider 和 consumer 微服务应用，根据您业务实际选择**虚拟机部署**、**容器化部署**以及 **TEM 部署**中的一种部署方式即可。

6.1 **虚拟机部署**

- 上传 Jar 包至 CVM 实例。
- 执行启动命令进行启动：

```
nohup java -Djava.security.egd=file:/dev/./urandom -jar [jar包名称] &
```

6.2 **容器化部署**

- 编写 dockerfile 生成镜像，参考：

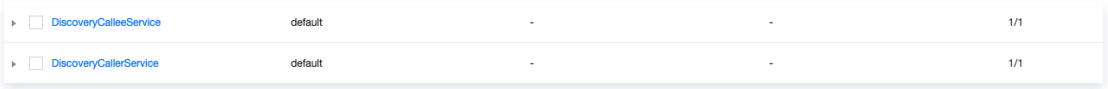
```
FROM java:8
ADD ddiscovery-caller-service-${version}-SNAPSHOT.jar /root/app.jar
ENTRYPOINT  ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/root/app.jar"]
```

- 通过 TKE 部署并运行镜像

7. 确认部署结果。

7.1 进入前面提到的北极星实例页面。

- 选择**服务管理 > 服务列表**，查看微服务 DiscoveryCallerService 和 DiscoveryCalleeService 的实例数量：
- 若实例数量值不为0，则表示已经成功接入微服务引擎。
- 若实例数量为0，或者找不到 DiscoveryCallerService 和 DiscoveryCalleeService 服务名，则表示微服务应用接入微服务引擎失败。



7.2 调用 consumer 的 HTTP 接口。执行 HTTP 调用，其中${app.port}替换为 consumer 的监听端口，${add.address}则替换为 consumer 暴露的地址：

```
curl -L -X GET 'http://${add.address}:${app.port}/echo?value="hello-world'
```

**预期返回值：** hello-world

# 注册发现

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 Spring Cloud 的方式接入 Polaris（北极星）实例，并实现服务注册与发现。

## 前提条件

1. 在开始开发前，请确保您已经参见 下载 Maven下载安装了 Java 和 Maven。
2. 已创建 Polaris（北极星）实例，操作步骤详细参见 引擎管理 。

## 操作步骤

为了方便您快速接入，我们为您准备了 Demo 应用， 点击下载 。

### 步骤1：引入服务注册与发现的依赖

#### 1. 引入 spring cloud tencent 依赖

修改应用根目录下的 pom.xml，添加 `dependencyManagement` ：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.cloud</groupId>
            <artifactId>spring-cloud-tencent-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **注意：**
> 请根据项目 Spring Boot 和 Spring Framework 的版本，选择合适的 Spring Cloud Tencent 版本 。

#### 2. 引入 spring cloud tencent starter

- 方式一：只引入 `spring-cloud-starter-tencent-polaris-discovery` 。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-polaris-discovery</artifactId>
</dependency>
```

- 方式二：通过 `spring-cloud-starter-tencent-all` 引入 SCT 所有 starters。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-all</artifactId>
    <!-- 注意需要指定 type=pom-->
    <type>pom</type>
</dependency>
```

如果使用 Spring Cloud 2020 及以后版本，还需要添加如下依赖：

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

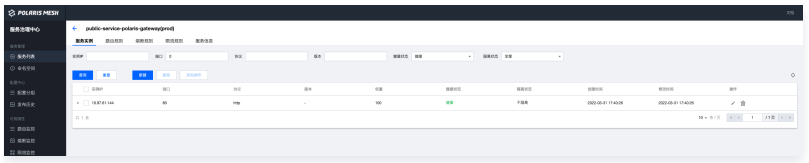
### 步骤2：添加北极星配置文件

1. 在项目的 main/resources 目录下创建 bootstrap.yml 文件。
2. 在 bootstrap.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址查看详细参见：引擎管理 > 客户端访问地址 。

```
spring:
  application:
    name: ${application.name}
  cloud:
    polaris:
      enabled: true
      address: grpc://${修改为Polaris 服务地址}:8091
      namespace: default
```

### 步骤3：启动应用

应用成功启动后，到 北极星控制台 查看服务注册信息。如下图所示：



### 步骤4： 服务调用

在 Spring Cloud 中可通过 `RestTemplate` 或者 `Feign` 发起服务调用。

- **`RestTemplate`**

  只需要在实例化 `RestTemplate` 的地方加上 `@LoadBalanced` 注解即可。

```
@Bean
@LoadBalanced
public RestTemplate restTemplate() {
    return new RestTemplate();
}
```

- **`Feign`**

  通过 `Feign` 框架调用，按照标准的 `Feign` 方式即可。

# 配置管理

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 SpringCloud 的方式接入北极星，使用配置管理功能。

## 前提条件

1. 在开始开发前，请确保您已经参见 下载 Maven下载安装了 Java 和 Maven。
2. 已创建了 Polars（北极星）实例，操作步骤详细参见 引擎管理 。

## 操作步骤

### 步骤1：引入配置管理依赖

#### 1.1 引入 spring cloud tencent 依赖

修改应用根目录下的 pom.xml，添加 `dependencyManagement` ：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.cloud</groupId>
            <artifactId>spring-cloud-tencent-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **注意：**
> 请根据项目 Spring Boot 和 Spring Framework 的版本，选择合适的 Spring Cloud Tencent 版本 。

#### 1.2 引入 spring cloud tencent starter

- 方式一：只引入 `spring-cloud-starter-tencent-polaris-config` 。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-polaris-config</artifactId>
</dependency>
```

- 方式二：通过 `spring-cloud-starter-tencent-all` 引入 sct 所有 starters。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-all</artifactId>
    <!-- 注意需要指定 type=pom-->
    <type>pom</type>
</dependency>
```

如果使用 Spring Cloud 2021 版本，还需要添加如下依赖：

```
<dependency>
```

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

### 步骤2：增加配置文件

> **注意：**
> - 应用启动强依赖配置，所以 `Spring Cloud Config` 是在 `Bootstrap` 阶段加载配置文件，所以需要把北极星相关的配置放在 `bootstrap.yml` 里。
> - `bootstrap.yml` 核心配置内容为配置北极星服务端地址以及注入的配置文件信息。

### 2.1 配置 Polaris（北极星）配置中心地址

1. 在项目的 main/resources 目录下创建 bootstrap.yml 文件。
2. 在 bootstrap.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址详细参见：引擎管理 > 客户端访问地址 。

- 如果北极星配置中心和注册中心是同一套北极星集群，则只需配置 `spring.cloud.polaris.address` 即可。
- 如果部署了两套北极星集群，分别用于注册中心和配置中心，则需配置 `spring.cloud.polaris.address` 用于指定注册中心集群的地址，配置 `spring.cloud.polaris.config.address` 用于指定配置中心的地址。如下所示：

```
spring:
  application:
    name: ${application.name}
  cloud:
    polaris:
      enabled: true
      address: grpc://${修改为 Polaris 服务地址}:8091 # 必填
      namespace: default # 全局 namespace 参数
      config:
        address: grpc://${独立的配置中心}:8093 # 选填，只有在配置中心和注册中心是两个不同的地址时才需要配置
        auto-refresh: true # 选填，当配置发布后，动态刷新 Spring 上下文，默认值为 true
```

### 2.2 注入配置文件

我们推荐的实践教程是在北极星管控端创建一个名为当前应用名 `${spring.application.name}` 的配置分组，Spring Cloud Tencent Config 会自动注入当前应用名分组下的：

- `application-${activeProfile}.properties`
- `application-${activeProfile}.yml`
- `application-${activeProfile}.yaml`
- `application.properties`
- `application.yml`
- `application.yaml`
- `bootstrap-${activeProfile}.properties`
- `bootstrap-${activeProfile}.yml`
- `bootstrap-${activeProfile}.yaml`
- `bootstrap.properties`
- `bootstrap.yml`
- `bootstrap.yaml`

> **说明：**
> 加载顺序依次从上到下，即越先加载的配置，优先级越高。与此同时，远端配置优先级大于本地配置优先级。优先级高的配置覆盖优先级低的配置。

自动注入以上配置文件符合 Spring Boot 的规范，能够满足绝大部分应用场景了。只有当您需要注入额外自定义的配置文件时，才需要在 `yml` 里配置 `spring.cloud.polaris.config.groups` ，如下所示：

```
spring:
  cloud:
    polaris:
      config:
        groups:
          - name: ${spring.application.name} # 选填，注入自定义配置的配置分组
            files: [ "config/application.properties", "config/bootstrap.yml" ] # 注入自定义配置文件列表，当
key 冲突时，排在前面的配置文件优先级高于后面
```

### 步骤3：在代码中使用配置

用户可通过两种方式更新代码中的配置信息：使用配置类 `@ConfigurationProperties` 和 `@Value` 注解。

- `@Value` 比较适用于配置比较少的场景
- `@ConfigurationProperties` 则更适用于有较多配置的情况

1. 通过 `@Value` 注入

```
@Value("${timeout:1000}")
private int timeout;
```

2. 通过 `@ConfigurationProperties` 注入

```
@Component
@ConfigurationProperties(prefix = "teacher")
@RefreshScope //如果使用反射模式，则不需要加这个注解
public class Person {

    private String name;

    private int age;

    String getName() {
        return name;
    }

    void setName(String name) {
        this.name = name;
    }

    int getAge() {
        return age;
    }

    void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" + "name='" + name + '\'' + ", age=" + age + '}';
    }

}.
```

### 步骤4：在北极星控制台添加配置文件

1. 创建命名空间

命名空间为北极星里核心的概念，通过命名空间逻辑隔离资源，例如通常用于标识不同的环境、不同的集群。

> ⚠ **注意：**
> 这里的命名空间需要和应用配置中的命名空间保持一致。

2. 创建配置文件分组

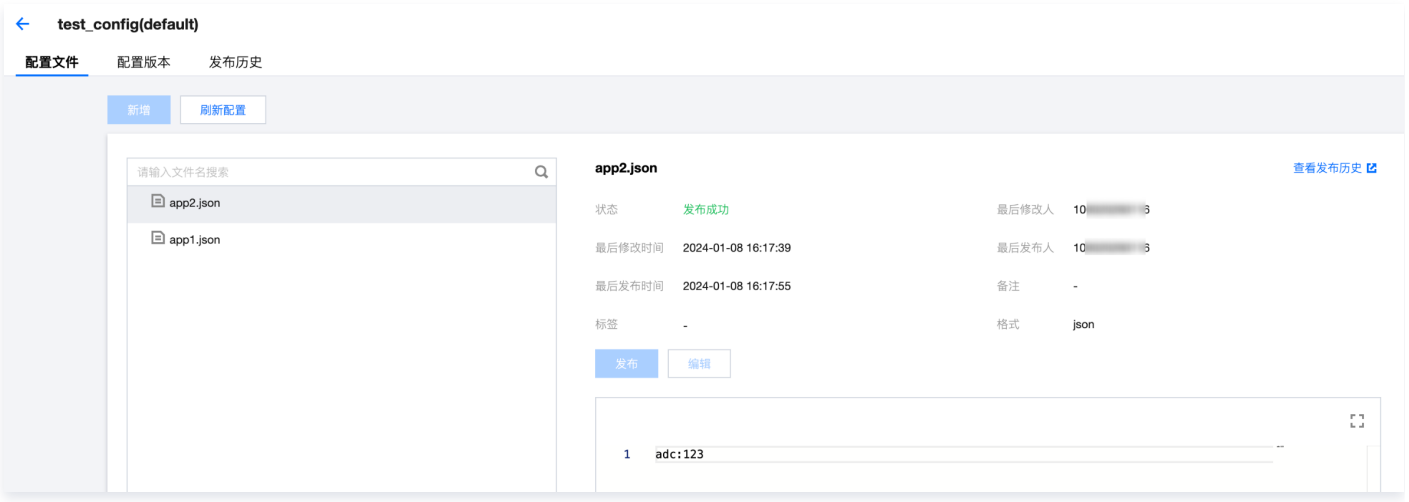
北极星的配置文件分组概念为一组配置文件的集合，推荐把应用名作为一个分组名，例如在我们的示例中，新建一个 `polaris-config-example` 的分组。把 `polaris-config-example` 应用的配置文件都放在 `polaris-config-example` 分组下，这样便于配置管理。

3. 创建并发布配置文件

北极星配置中心的控制台，配置文件名可以通过“/”来按树状目录结构展示，树状结构可以清晰的管理配置文件。例如一个应用下分不同的模块，每个模块都有独立的一组配置文件，则可以创建 `module1/file1.properties`，`module1/file2.yaml`，`module2/file3.yaml`。

> ⚠ **注意：**
> 配置文件名强烈建议带文件后缀，例如 .properties .yaml .yml .json 等。因为客户端会通过文件名后缀来解析文件内容，如果客户端发现不认识的后缀名则默认当做 .properties 文件处理。

配置好的实例如下图所示：




## 步骤5：启动应用

到此接入 `Spring Cloud Tencent Config` 即已完成。

## 步骤6：动态刷新配置

> ⓘ **说明：**
> - `Spring Cloud Tencent 1.7.0` 之前的版本通过 Spring Cloud 标准的 `@RefreshScope` 实现配置动态刷新。`@RefreshScope` 存在以下局限性：
>   - 每次刷新配置，需要重新构建整个 `Spring Context` 重建 Bean。
>   - 代码里层面需要引入 `@RefreshScope` 注解。
> - 1.7.1 及之后版本，Spring Cloud Tencent 优化了动态刷新的实现方式，建议您使用最新版本。

### 6.1 刷新 @Value 属性

应用启动时扫描所有的 Bean，构建 `@Value` 属性对应的 Bean 的映射关系。当属性配置更新时，找到所有的待更新的 Bean 并利用 Java 的反射机制更新属性。

### 6.2 刷新 @ConfigurationProperties 配置类

相比于 @RefreshScope 重建所有的 Bean，优化之后的刷新机制只需要重建更新的配置对应的 `@ConfigurationProperties Bean`，影响面更小。
实现原理请看 `PolarisRefreshAffectedContextRefresher` 和 `AffectedConfigurationPropertiesRebinder`。
如果您想使用传统的 `@RefreshScope` 方式刷新配置，可以配置 `spring.cloud.polaris.config.refresh-type=refresh_context`。

### 6.3 关闭动态刷新能力

在`bootstrap.yml`里配置`spring.cloud.polaris.config.auto-refresh=false`。

# 服务限流

最近更新时间：2025-01-15 12:06:52

## 操作场景

服务限流是最常见的一种服务自我保护措施之一，防止流量洪峰打垮服务。`Spring Cloud Tencent Rate Limit` 模块内置了针对 `Spring Web` 和 `Spring WebFlux` 场景的限流 `Filter`，结合北极星的限流功能帮忙业务快速接入限流能力。

## 前提条件

1. 在开始开发前，请确保您已经参见 下载 Maven下载安装了 Java 和 Maven。
2. 已创建 TSE 治理中心（北极星网格）实例，操作步骤详细参见 引擎管理 。

## 操作步骤

### 步骤1：引入依赖

#### 1.1 引入 spring cloud tencent 依赖

修改应用根目录下的 pom.xml，添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.cloud</groupId>
            <artifactId>spring-cloud-tencent-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **注意：**
> 请根据项目 Spring Boot 和 Spring Framework 的版本，选择合适的 Spring Cloud Tencent 版本 。

#### 1.2 引入 spring cloud tencent starter

- 方式一：只引入 `spring-cloud-starter-tencent-polaris-ratelimit` 。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-polaris-ratelimit</artifactId>
</dependency>
```

- 方式二：通过 `spring-cloud-starter-tencent-all` 引入 sct 所有 starters。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-all</artifactId>
    <!-- 注意需要指定 type=pom-->
    <type>pom</type>
</dependency>
```

如果使用 Spring Cloud 2020 及以后版本，还需要添加如下依赖：

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

```
</dependency>
```

## 步骤2：增加配置文件

1. 在项目的 main/resources 目录下创建 bootstrap.yml 文件。
2. 在 bootstrap.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址详细参见：引擎管理 > 客户端访问地址 。

```
spring:
  application:
    name: ${application.name}
  cloud:
    polaris:
      enabled: true
      address: grpc://${修改为第一步部署的 Polaris 服务地址}:8091
      namespace: default
```

## 步骤3：控制台配置限流规则

请参见 服务治理指南中服务治理 > 服务限流 相关文档，配置限流规则，相关界面如下：



## 步骤4：启动应用

应用启动成功后，访问应用的 HTTP 接口，测试限流是否生效。如果请求被限流将会返回以下内容：

```
The request is deny by rate limit because the throttling threshold is reached
```

# 服务熔断

最近更新时间：2025-04-17 16:00:53

## 熔断场景

Spring Cloud Tencent CircuitBreaker 结合 Polaris 的熔断能力提供了标准的熔断实现，并提供服务级、接口级、节点级三种熔断级别，可以根据需求组合搭配，来实现期望的熔断效果。
本文介绍在本地开发 Java 应用，通过 Spring Cloud 的方式接入 Polaris（北极星），使用服务熔断功能。

## 前提条件

1. 在开始开发前，请确保您已经参见 下载 Maven下载安装了 Java 和 Maven。
2. 已创建 Polaris（北极星）实例，操作步骤详情参见 引擎管理 。

## 操作步骤

### 步骤1：引入依赖

#### 1.1 引入 spring cloud tencent 依赖

修改应用根目录下的 pom.xml，添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.cloud</groupId>
            <artifactId>spring-cloud-tencent-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

**注意：**

- 请根据项目 Spring Boot 和 Spring Framework 的版本，选择合适的 Spring Cloud Tencent 版本 。
- 服务级熔断支持版本号大于1.10.0的 Spring Cloud Tencent 版本 。

#### 1.2 引入 spring cloud tencent starter

**说明：**
因为熔断能力依赖服务发现能力，所以需要主调方和被调方都引入服务注册与发现依赖。

- 方式一：只引入 `spring-cloud-starter-tencent-polaris-circuitbreaker` 。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-polaris-circuitbreaker</artifactId>
</dependency>
```

- 方式二：通过 `spring-cloud-starter-tencent-all` 引入 sct 所有 starters。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-all</artifactId>
```

```
    <!-- 注意需要指定 type=pom-->
    <type>pom</type>
</dependency>
```

如果使用 Spring Cloud 2020 及以后版本，还需要添加如下依赖：

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

### 步骤2：增加相关配置文件并设置熔断方法

Spring Cloud Tencent 支持 Feign、RestTemplate、WebClient、Spring Cloud Gateway 四种组件的熔断，并会自动拉取您在北极星上为被调服务配置的熔断规则进行熔断。选择您适合的方式配置即可。

#### Feign

完整代码可查看 Demo：

1. 添加 Feign 依赖

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

2. 添加 Feign 配置文件

```
spring:
  application:
    name: polaris-circuitbreaker-feign-example
  cloud:
    openfeign:
      circuitbreaker:
        enabled: true
    polaris:
      address: grpc://${修改为 Polaris 服务地址}:8091
      namespace: default
      enabled: true
      loadbalancer:
        enabled: true
      circuitbreaker:
        enabled: true
```

3. 设置 Feign 熔断方法
   - 声明 Feign 调用

     当 Feign fallback 不配置时，会在熔断发生时，从 polaris server 拉取降级配置，进行降级，降级的 response body 会序列化成您在 Feign 中配置的返回对象，请确保能正确的序列化。

```
@Primary
@FeignClient(name = "polaris-circuitbreaker-callee-service", fallback =
ProviderBFallback.class)
public interface ProviderB {

    /**
```

```
     * Get info of service B.
     *
     * @return info of service B
     */
    @GetMapping("/example/service/b/info")
    String info();
}
```

- 声明本地 Feign 熔断方法（可选）

```
  @Component
public class ProviderBFallback implements ProviderB {

    @Override
    public String info() {
        return "fallback: trigger the refuse for service b";
    }
}
```

- 发起 Feign 调用

```
@RestController
@RequestMapping("/example/service/a")
public class ServiceAController {

    @Autowired
    private ProviderB polarisServiceB;

    /**
     * Get info of Service B by Feign.
     * @return info of Service B
     */
    @GetMapping("/getBServiceInfo")
    public String getBServiceInfo() {
        return polarisServiceB.info();
    }
```

### RestTemplate

完整代码可查看 Demo：

1. 添加 RestTemplate 配置文件

```
spring:
  application:
    name: polaris-circuitbreaker-resttemplate-example
  cloud:
    polaris:
      address: grpc://${修改为 Polaris 服务地址}:8091
      namespace: default
      enabled: true
      loadbalancer:
        enabled: true
      circuitbreaker:
```

```
        enabled: true
```

2. **设置 RestTemplate 熔断方法**

   设置 RestTemplate bean，添加 `@PolarisCircuitBreaker` 注解。

```
@Bean
@LoadBalanced
@PolarisCircuitBreaker
public RestTemplate restTemplate() {
    DefaultUriBuilderFactory uriBuilderFactory = new DefaultUriBuilderFactory("http://polaris-
circuitbreaker-callee-service");
    RestTemplate restTemplate = new RestTemplate();
    restTemplate.setUriTemplateHandler(uriBuilderFactory);
    return restTemplate;
}
```

> **说明：**
> @PolarisCircuitBreaker 包含两个参数：
> - String 类型的 fallback 表示返回希望降级的 `response body string`；
> - `PolarisCircuitBreakerFallback` 类型的 `fallbackClass` 表示返回希望降级的具体 `response`，需要自定义实现 `PolarisCircuitBreakerFallback#fallback()` 方法，返回 `PolarisCircuitBreakerHttpResponse`。
>
> 两个参数不能同时配置。当 `@PolarisCircuitBreaker` 不配置任何参数时，会在熔断发生时，从 polaris server 拉取降级配置，进行降级，降级的 response body 会序列化成您在 RestTemplate 中配置的返回对象，请确保能正确的序列化。

3. **发起 RestTemplate 调用**

```
@RestController
@RequestMapping("/example/service/a")
public class ServiceAController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/getBServiceInfo")
    public String getBServiceInfo() {
        return restTemplateFallbackFromPolaris.getForEntity("/example/service/b/info",
String.class);
    }
}
```

### WebClient

完整代码可查看 demo：

1. **添加 WebFlux 依赖**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

2. **设置 WebClient 配置文件**

```
spring:
  application:
    name: polaris-circuitbreaker-webclient-example
  cloud:
    polaris:
      address: grpc://${修改为 Polaris 服务地址}:8091
      namespace: default
      enabled: true
      loadbalancer:
        enabled: true
      circuitbreaker:
        enabled: true
```

3. **设置 WebClient 熔断方法**
   - **设置 WebClient bean**

```
@LoadBalanced
@Bean
WebClient.Builder webClientBuilder() {
    return WebClient.builder()
            .baseUrl("http://polaris-circuitbreaker-callee-service");
}
```

   - **发起 WebClient 调用**

```
@RestController
@RequestMapping("/example/service/a")
public class ServiceAController {

    @Autowired
    private ReactiveCircuitBreakerFactory reactiveCircuitBreakerFactory;

    @Autowired
    private WebClient.Builder webClientBuilder;

    @GetMapping("/getBServiceInfo")
    public Mono<String> getBServiceInfo() {
        return webClientBuilder
                .build()
                .get()
                .uri("/example/service/b/info")
                .retrieve()
                .bodyToMono(String.class)
                .transform(it ->
                        reactiveCircuitBreakerFactory
                                                // create 的参数为被调服务的服务名
                        .create("polaris-circuitbreaker-callee-service")
                                                // 熔断降级方法
                        .run(it, throwable -> Mono.just("fallback: trigger the refuse for
service b"))
                );
    }

}
```

### Spring Cloud Gateway

完整代码可查看 demo：

1. **添加 Spring Cloud Gateway 依赖**

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

2. **设置 Spring Cloud Gateway 配置文件**
   - 以 gateway.discovery.locator 形式配置

```
spring:
  application:
    name: GatewayScgService
  cloud:
    polaris:
      address: grpc://${修改为 Polaris 服务地址}:8091
      namespace: default
      enabled: true
    gateway:
      discovery:
        locator:
          enabled: true
          'predicates[0]':
            name: Path
            args:
              patterns: '''/'' + serviceId + ''/**'''
          'filters[0]':
            name: RewritePath
            args:
              regexp: '''/'' + serviceId + ''/(?<remaining>.*)'''
              replacement: '''/$\{remaining}'''
          'filters[1]':
            # 配置熔断 Filter
            name: CircuitBreaker
            args:
              # statusCodes，可选参数，配置熔断的匹配返回码，缺省时会自动识别 "5**" 为错误，可以
为"4**,5**"的形式，也可以为具体的返回码
              statusCodes: '''404,5**'''
              # fallbackUri，可选参数，配置熔断降级的转发路由方法，缺省时会在熔断触发后拉取 plaris
server 配置的降级作为 response
              fallbackUri: '''forward:/polaris-fallback'''
```

   - 以普通路由形式配置形式配置

```
  spring:
  application:
    name: GatewayScgService
  cloud:
    polaris:
      address: grpc://${修改为 Polaris 服务地址}:8091
      namespace: default
      enabled: true
    gateway:
```

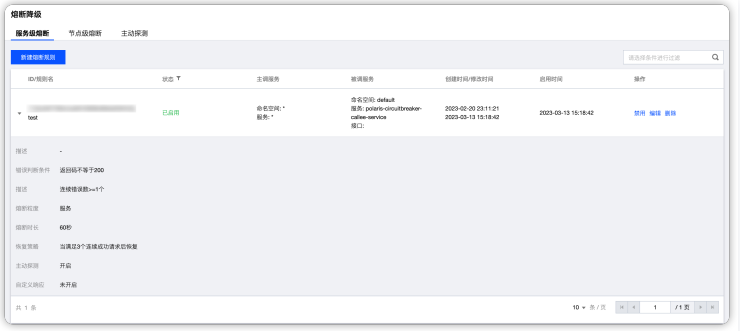
```
      routes:
        - id: polaris-circuitbreaker-callee-service
          uri: lb://polaris-circuitbreaker-callee-service
          predicates:
            - Path=/polaris-circuitbreaker-callee-service/**
```

3. **设置 Spring Cloud Gateway 熔断方法**
   - **设置熔断降级的转发路由方法**

```
@RestController
public class FallbackController {

    @GetMapping("/polaris-fallback")
    Mono<String> getFallback() {
        return Mono.just("fallback: trigger the refuse for service b");
    }
}
```

## 步骤3：验证

**启动应用并在 北极星控制台 设置被调服务的服务级熔断规则；请参见 服务治理指南 > 服务治理 > 熔断降级 > 服务熔断与主动探测 相关文档，相关界面如下：**




- **为了验证熔断能力，被调端至少启动两个实例，一个返回成功，一个返回失败。**
- **测试主调方向被调方发起服务调用，前几个请求会出现往有问题的服务发起调用，当问题数达到您在北极星控制台配置的熔断阈值时，即会触发熔断，主调方不再发起调用，直接返回熔断降级方法。**

# 动态路由

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 Spring Cloud 的方式接入 Polaris（北极星），并实现服务路由。

## 前提条件

1. 在开发前，请确保您已经参见 下载 Maven下载安装了 Java 和 Maven。
2. 已创建 Polaris（北极星）实例，操作步骤详见 引擎管理 。

## 操作步骤

### 步骤1：引入依赖

#### 1.1 引入 spring cloud tencent 依赖

修改应用根目录下的 pom.xml，添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.cloud</groupId>
            <artifactId>spring-cloud-tencent-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **注意：**
> - 请根据项目 Spring Boot 和 Spring Framework 的版本，选择合适的 Spring Cloud Tencent 版本 。
> - 服务路由在 1.5.0版本之后，提供了完善的服务路由能力。所以版本号需大于1.5.0。

#### 1.2 引入 spring cloud tencent starter

- 方式一：同时引入 `spring-cloud-starter-tencent-polaris-discovery` 和 `spring-cloud-starter-tencent-polaris-router` 。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-polaris-discovery</artifactId>
</dependency>

<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-polaris-router</artifactId>
</dependency>
```

- 方式二：通过 `spring-cloud-starter-tencent-all` 引入 sct 所有 starters。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-all</artifactId>
    <!-- 注意需要指定 type=pom-->
```

```
    <type>pom</type>
</dependency>
```

如果使用 Spring Cloud 2020 及以后版本，还需要添加如下依赖：

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

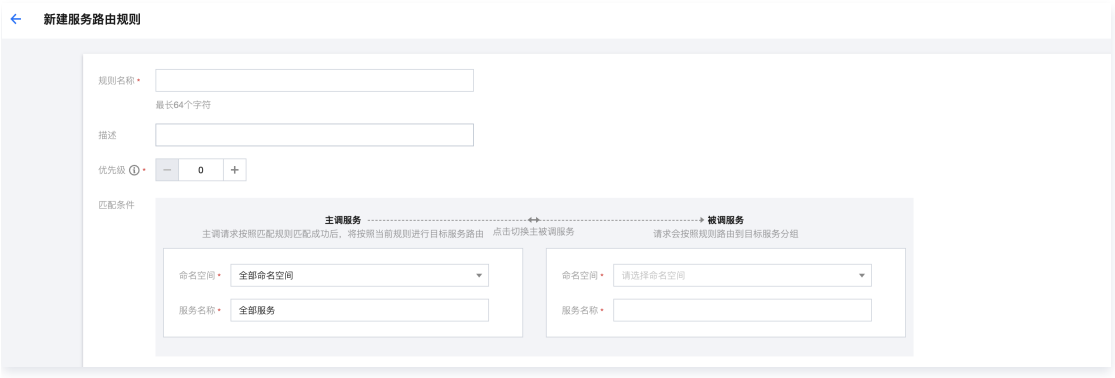
## 步骤2：添加北极星配置文件

1. 在项目的 main/resources 目录下创建 bootstrap.yml 文件。
2. 在 bootstrap.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址查看详见 引擎管理 的客户端访问地址部分。

```
spring:
  application:
    name: ${application.name}
  cloud:
    polaris:
      enabled: true
      address: grpc://${修改为 Polaris 服务地址}:8091
      namespace: default
```

## 步骤3：路由规则下发

启动应用并在 北极星控制台 设置动态路由规则，操作步骤详细参见 服务治理指南 > 服务治理 > 动态路由，相关界面如下：




# 全链路灰度

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 Spring Cloud 的方式接入 Polaris（北极星），并实现全链路灰度能力。

## 前提条件

1. 在开始开发前，请确保您已经参见 下载 Maven下载安装了 Java 和 Maven。
2. 已创建 Polaris（北极星）实例，操作步骤详细参见 引擎管理 。

## 操作步骤

为了方便您快速接入，我们为您准备了 Demo 应用， 点击下载 。

### 步骤1：引入服务注册与发现的依赖

### 1. 引入 spring cloud tencent 依赖

修改应用根目录下的 pom.xml，添加 `dependencyManagement` ：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.cloud</groupId>
            <artifactId>spring-cloud-tencent-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **注意：**
> 请根据项目 Spring Boot 和 Spring Framework 的版本，选择合适的 Spring Cloud Tencent 版本 。

### 2. 引入 spring cloud tencent starter

- 方式一：同时引入 `spring-cloud-starter-tencent-polaris-discovery` 和 `spring-cloud-starter-tencent-polaris-router` 。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-polaris-discovery</artifactId>
</dependency>

<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-polaris-router</artifactId>
</dependency>
```

- 方式二：通过 `spring-cloud-starter-tencent-all` 引入 SCT 所有 starters。

```
<dependency>
    <groupId>com.tencent.cloud</groupId>
    <artifactId>spring-cloud-starter-tencent-all</artifactId>
    <!-- 注意需要指定 type=pom-->
```

```
    <type>pom</type>
</dependency>
```

如果使用 Spring Cloud 2020 及以后版本，还需要添加如下依赖：

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

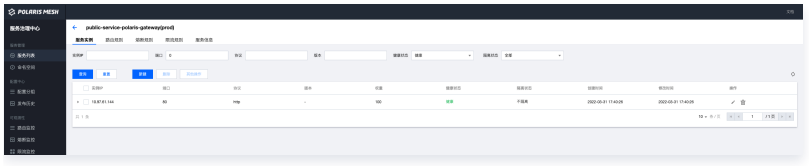
## 步骤2：添加北极星配置文件

1. 在项目的 main/resources 目录下创建 bootstrap.yml 文件。
2. 在 bootstrap.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址详细参见：引擎管理 > 客户端访问地址 。

```
spring:
  application:
    name: ${application.name}
  cloud:
    polaris:
      enabled: true
      address: grpc://${修改为Polaris 服务地址}:8091
      namespace: default
```

## 步骤3：启动应用

应用成功启动后，到 北极星控制台 查看服务注册信息。如下图所示：



## 步骤4： 控制台配置全链路灰度规则

完成客户端接入后，您可在北极星控制台配置您全链路灰度的规则，详情请参见： 全链路灰度配置 。

# 无损上下线

最近更新时间：2025-04-17 16:00:53

## 使用场景

微服务发布过程中，可能会因为服务的变更造成流量的错误或中断。Spring Cloud Tencent 提供了插件来实现无损上下线，其原理为：

- 无损上线：在服务启动后，等待服务完全就绪后再注册到注册中心（无配置健康探测接口时为延迟注册），并对外提供服务。然后再结合 K8s 生命周期，滚动更新下一个节点。
- 无损下线：在服务停止前，先从注册中心注销，拒绝新的请求，等待旧的请求处理完毕后再下线服务。

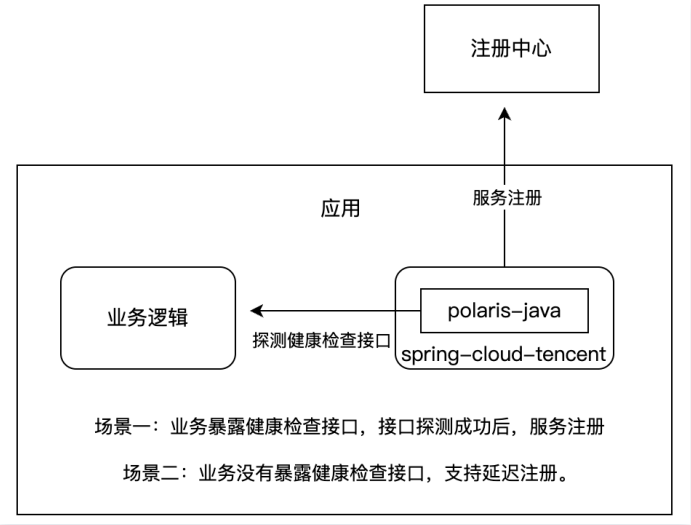
## 无损上线

Polaris（北极星）支持以下 2 种方式支持服务无损上线：服务就绪/延迟注册 和 服务注册就绪检查 。

### 方案一：服务就绪/延迟注册

在一些场景下，服务支持延迟加载，启动后异步加载一些资源，例如：服务需要从文件存储 COS 获取数据或者文件，待数据或者文件拉取完成后才能对外提供服务。如果在应用启动后直接注册服务，会导致服务实际没有就绪而调用失败。因此通过确保服务就绪之后，再进行注册从而对外提供服务，可以确保服务平滑、无损上线。
北极星支持以下两种服务就绪/延迟注册场景：

- 场景一：业务暴露健康检查接口，接口探测成功后，服务注册。
- 场景二：业务没有暴露健康检查接口，则延迟一段时间服务注册，默认延迟注册时长30s，可通过配置自定义延迟注册的时长。




#### 操作步骤

步骤1：服务通过 Spring Cloud Tencent 接入北极星服务注册与发现能力。具体步骤请参见：Spring Cloud Tencent 接入北极星 。
步骤2：引入 spring cloud tencent 无损上下线插件依赖。
在 pom.xml 中添加依赖：

```
<dependencies>
    <dependency>
        <groupId>com.tencent.cloud</groupId>
        <artifactId>spring-cloud-tencent-lossless-plugin</artifactId>
    </dependency>
</dependencies>
```

步骤3：在应用北极星配置文件中添加无损上下线相关配置项。具体参数如下：

| 配置项 Key | 默认值 | 是否必填 | 配置项说明 |
| --- | --- | --- | --- |

| | | | |
|---|---|---|---|
| spring.cloud.polaris.lossless.enabled | false | 否 | 无损上下线开关。 |
| spring.cloud.polaris.lossless.health-check-path | 无 | 否 | 业务程序健康探测接口。 |
| spring.cloud.polaris.lossless.delay-register-interval | 30000 | 否 | 无配置业务程序健康探测接口，延迟注册时间。默认 30000（单位ms）。 |
| spring.cloud.polaris.lossless.health-check-interval | 5000 | 否 | 配置业务程序健康探测接口后，健康探测间隔。默认 5000（单位ms）。 |
| spring.cloud.polaris.admin.host | 0.0.0.0 | 否 | 无损上下线服务绑定的 IP 地址。 |
| spring.cloud.polaris.admin.port | 28080 | 否 | 无损上下线服务绑定的端口号。 |

**配置示例：**
在项目的 main/resources 目录下 bootstrap.yml 文件配置：

```
spring:
  application:
    name: ${application.name}
  cloud:
    polaris:
      enabled: true
      address: grpc://${修改为Polaris 服务地址}:8091
      namespace: default
      # 以下为无损上下线的配置
      lossless:
        enabled: true
        health-check-path: /
        delay-register-interval: 300000
        health-check-interval: 10000
      admin:
        host: 0.0.0.0
        port: 28080
```

## 方案二：服务注册就绪检查

通常 K8s 提供就绪检查机制来对实例在就绪前进行健康检查，它一般会简单地认为端口起来了应用即处于就绪状态。但是实际上，应用端口起来了和服务成功注册之间存在 gap，这样会造成服务没有成功注册，旧的应用实例就被下线，下一个节点开始部署。最终导致消费端调用异常。
TSE 北极星提供服务注册状态的接口和端口，以配合 K8s 就绪检查，当应用注册完成返回200状态码，帮助 K8s 判定应用已就绪；未完成注册返回500状态码，帮助 K8s 判定应用未就绪。尤其在实例滚动更新的过程中，等实例就绪后才滚动更新下一个节点。

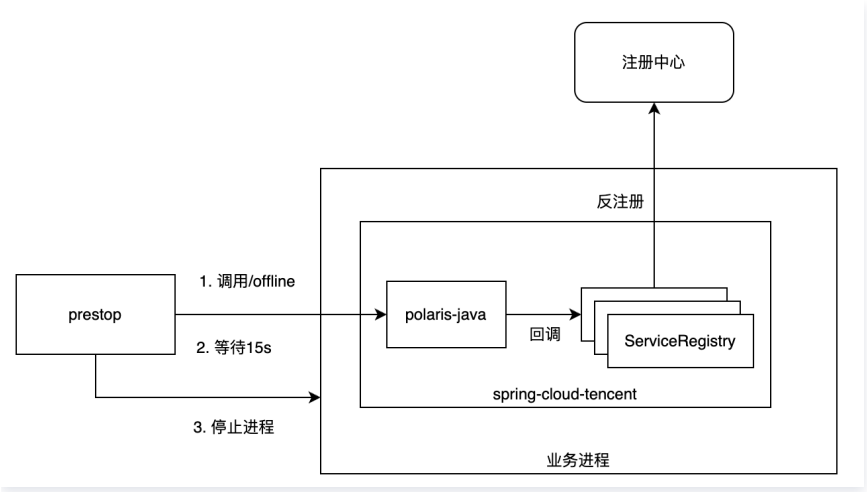
**操作步骤：**

**步骤1**：服务通过 Spring Cloud Tencent 接入北极星服务注册与发现能力。具体步骤参见：Spring Cloud Tencent 接入北极星 。
**步骤2**：引入 spring cloud tencent 无损上下线插件依赖，并添加无损上下线相关配置。具体参见 方案一 。
**步骤3**：在容器服务 TKE 等 K8s 应用部署平台中配置就绪检查，如下图所示。

- 路径：/online。
- 端口：28080。


## 无损下线

应用在滚动发布或者下线过程中，被调方服务实例向注册中心发起反注册，主调方从注册中心更新 IP 的过程中，存在时间 gap，导致依然有可能调用到已经下线的实例，从而调用失败。
TSE 北极星提供服务无损下线的接口：/offline，结合 K8s 的生命周期，实现服务无损下线。整体流程如下：




**操作步骤：**

**步骤1：**服务通过 Spring Cloud Tencent 接入北极星服务注册与发现能力。具体步骤参见：Spring Cloud Tencent 接入北极星 。
**步骤2：**引入 spring cloud tencent 无损上下线插件依赖，并添加无损上下线相关配置。具体参见 方案一 。
**步骤3：**在 容器服务 TKE 等 K8s 应用部署平台中配置 preStop 生命周期检查。
preStop 配置检查指令：`curl -X PUT http://localhost:28080/offline && sleep 20`

# 使用 Java Agent
# Java Agent 概述

最近更新时间：2025-06-11 15:55:02

## 功能概述

Spring Cloud Tencent 支持 Java Agent 的方式进行接入。对于 Spring Cloud 应用（原生 Spring Cloud 应用或者其他 Spring Cloud 扩展的应用），可以不用对应用代码及依赖进行任何变更的情况下，即可使用 Spring Cloud Tencent 的功能。

## Spring Cloud 各版本及功能在 Java Agent 上的支持情况

> **说明：**
> 使用 Java Agent 的方式进行接入时，只支持 Open JDK 和 Kona JDK，不支持 Oracle JDK。

| 功能大类 | 功能项 | Spring Cloud 2023 | Spring Cloud 2022 | Spring Cloud Hoxton |
|---|---|---|---|---|
| | | JDK 17 | JDK 17 | JDK 8 |
| Java Agent 注入 | 虚拟机注入 | 支持 | 支持 | 支持 |
| | 容器化注入 | 支持 | 支持 | 支持 |
| 注册中心 | 服务注册 | 支持 | 支持 | 支持 |
| | 服务发现 | 支持 | 支持 | 支持 |
| | 健康检查（心跳上报） | 支持 | 支持 | 支持 |
| | Nacos&北极星服务双注册 | 不支持 | 不支持 | 不支持 |
| 流量调度 | 动态路由 | 支持 | 支持 | 支持 |
| | 就近路由 | 支持 | 支持 | 支持 |
| | 服务限流 | 不支持 | 支持 | 支持 |
| | 无损上下线 | 支持 | 支持 | 支持 |
| | 全链路灰度 | 支持 | 支持 | 支持 |
| 故障容错 | 故障熔断 | 不支持 | 支持 | 支持 |
| | 主动探测 | 不支持 | 支持 | 支持 |
| | 自定义降级 | 不支持 | 支持 | 支持 |
| 配置管理 | 配置文件订阅 | 支持 | 支持 | 支持 |

# 使用 Java Agent 接入

最近更新时间：2025-06-11 15:55:02

## 功能概述

Spring Cloud Tencent 的 Java Agent 支持针对容器部署的应用以及虚拟机部署的应用进行注入。对于 Spring Cloud 应用（原生 Spring Cloud 应用或者其他 Spring Cloud 扩展的应用），可以不用对应用代码及依赖进行任何变更的情况下，即可使用 Spring Cloud Tencent 的功能。下文将分别对容器场景和虚机场景的使用方法进行说明。

## 前提条件

- 已创建 Polaris（北极星）引擎实例。
- 登录 TSF 控制台。
- 在**服务治理中心**下的**实例列表**页面，单击目标引擎的 ID，进入基本信息页面。

## 使用步骤

1. 登录 TSF 控制台。
2. 在左侧导航栏选择**Polaris（北极星）**后，进入北极星引擎实例列表页。
3. 单击目标引擎的 ID，进入基本信息页面。

容器场景

### 步骤1：安装 polaris-controller

> **说明：**
> 1. 添加容器集群会安装 Polaris-Controller 组件，请确保您的集群至少有 1C2G 的空闲资源，否则会添加失败。
> 2. 容器集群中安装 Polaris-Controller 组件后默认支持 K8s Service 服务同步（按需同步）和 Java Agent 自动注入能力。您也可以在控制台选择**全量同步**使用 k8s 全量同步方式。

1. 在目标引擎**基本信息**页顶部页签单击 **K8s 集群**，单击**关联集群**，单击查看关于 K8s 集群关联到 Polaris 的相关介绍。
2. 根据自身业务需求选择目标关联的 K8s 集群，支持 TKE (容器集群) / EKS (弹性容器集群)。即在目标集群中完成 polaris-controller 的安装。
3. K8s Service 同步接入：
   - 全量同步：集群内全部 service 自动同步至北极星。


   - 按需同步：仅指定的 namespace 或者 service 自动同步至北极星。


## 步骤2：在命名空间开启自动注入功能

- 在目标应用所在的 K8s 命名空间，通过以下命令开启自动注入：

```
kubectl label namespace <命名空间名> polaris-injection=enabled
```

- 通过以下命令查询命名空间是否开启了自动注入：

```
kubectl get namespace -L polaris-injection

输出结果：
NAME              STATUS   AGE    POLARIS-INJECTION
default           Active   3d2h   enabled
```

- 注入范围控制列表

| label 键 | 范围 | label 值 | 含义 | polaris-controller 版本 |
| --- | --- | --- | --- | --- |
| polarismesh.cn/inject | Pod | disabled | 声明当命名空间开启默认注入时，作为黑名单功能使用，指定 pod 不注入 javaagent | v1.7.3及以上版本 |
| polarismesh.cn/inject | Pod | enabled | 声明当命名空间未开启默认注入时，作为白名单功能使用，单独指定 pod 注入 javaagent | v1.7.3及以上版本 |
| polarismesh.cn/inject | Namespace | enabled | 声明该命名空间下的 pod，默认注入 javaagent | v1.7.3及以上版本 |
| polaris-injection | Namespace | enabled | 声明该命名空间下的 pod，默认注入 javaagent | 所有版本支持，待废弃，使用 polarismesh.cn/inject 代替 |

- 具体yaml示例

```
apiVersion: v1
kind: Namespace
metadata:
  name: java-agent-test
  labels:
    polarismesh.cn/inject: enabled
```

- 黑名单功能，不注入java agent。

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: label-block
  namespace: java-agent-test
spec:
  selector:
    matchLabels:
      app: label-block
  template:
    metadata:
      labels:
        app: label-block
        polarismesh.cn/inject: disabled
      annotations:
        polarismesh.cn/javaagent: "true"
    spec:
    ......
```

- 白名单功能，命名空间未开启自动注入的情况下，实现指定 pod 按需注入。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: label-allow
spec:
  selector:
    matchLabels:
      app: label-allow
  template:
    metadata:
      labels:
        app: label-allow
        polarismesh.cn/inject: enabled
      annotations:
        polarismesh.cn/javaagent: "true"
    spec:
    ......
```

### 步骤3：SpringCloud 应用 yaml 文件中声明 Java Agent 标签

- 找到需要注入 Java Agent 的应用（一般是 deployment 或者 statefulset），编辑 yaml 并增加以下标签：

| Annotation键 | Annotation值 | 含义 | polaris-controller版本 |
| --- | --- | --- | --- |
| polarismesh.cn/javaagent | 支持："true"、"false"，必填 | 声明需要往这个 POD 中注入 Java Agent | 所有版本支持 |
| polarismesh.cn/workloadNamespaceAsServiceNamespace | 支持："true"、"false"，缺省 | 声明使用 k8s 命名空间作为服务命名空间注册到北极星，若和 polarismesh.cn/javaagentConfig 中的配置冲突，此配置不生效 | v1.7.3及以上版本 |
| polarismesh.cn/workloadNameAsServiceName | 支持："true"、"false"，缺省 | 声明使用 k8s 工作负载名称作为服务名注册到北极星，若和 polarismesh.cn/javaagentConfig 中的配置冲突，此配置不生效 | v1.7.3及以上版本 |
| polarismesh.cn/javaagentFrameworkName | 缺省，则自动识别应用框架的类型。<br>支持：spring-cloud | 声明应用的框架类型，对于 SpringCloud 应用则填写 spring- | 所有版本支持 |

| | | | |
|---|---|---|---|
| | | cloud，目前仅支持 SpringCloud 类型。 | |
| polarismesh.cn/javaagentFrameworkVersion | 缺省，则自动识别应用框架的版本。<br>也支持手动声明框架版本，例如：2023、2022、hoxton 等。 | 声明应用的框架版本。 | 所有版本支持 |
| polarismesh.cn/javaagentVersion | 如：2.0.1.0。<br>**注意：**<br>生产环境请指定验证后的版本，若不填则自动会在每次实例发生重启时自动注入最新的版本，可能会有新版本的兼容性问题，导致实例启动异常。 | 声明 java-agent 包的镜像版本。 | 所有版本支持 |
| polarismesh.cn/javaagentConfig | 如指定命名空间、服务名并配置无损上线：<br>'{"spring.cloud.polaris.lossless.enabled":"true","spring.cloud.polaris.lossless.delay-register-interval":"30000","spring.cloud.polaris.discovery.namespace":"custom-ns","spring.application.name":"custom-service"}' | 支持用户自定义的 Java Agent 配置，支持指定命名空间、服务名、服务实例元数据、无损上线配置等。格式为 JSON。<br>不填写的配置则使用默认配置。 | 所有版本支持 |

- 配置 yaml 样例：

  默认配置：${spring.application.name}的值作为服务名，default作为北极星命名空间。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: default-ns-name
  namespace: java-agent-test
spec:
  selector:
    matchLabels:
      app: default-ns-name
  template:
    metadata:
      labels:
        app: default-ns-name
      annotations:
        polarismesh.cn/javaagent: "true"
    spec:
        ......
```

- 使用 k8s 工作负载名称作为服务名，k8s 命名空间作为北极星服务命名空间注册。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-info-inject
  namespace: java-agent-test
spec:
  selector:
```

```
    matchLabels:
      app: k8s-info-inject
  template:
    metadata:
      labels:
        app: k8s-info-inject
      annotations:
        polarismesh.cn/javaagent: "true"
        polarismesh.cn/workloadNamespaceAsServiceNamespace: "true"
        polarismesh.cn/workloadNameAsServiceName: "true"
    spec:
        ......
```

- 使用自定义配置，下面的例子为使用 config 里面的配置，使用 custom-service 作为服务名，custom-ns 作为北极星服务命名空间注册。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: custom-ns-name
  namespace: java-agent-test
spec:
  selector:
    matchLabels:
      app: custom-ns-name
  template:
    metadata:
      labels:
        app: custom-ns-name
      annotations:
        polarismesh.cn/javaagent: "true"
        polarismesh.cn/javaagentConfig: '{"spring.cloud.polaris.discovery.namespace": "custom-
ns","spring.application.name":"custom-service"}'
    spec:
        ......
```

修改 yaml 后，需要重新部署应用。重新部署后，会自动完成 java-agent 的注入，可以观察应用日志，如果启动的时候出现以下关键字，则证明 java-agent 注入成功。

```
......
2024-02-01 21:19:16 [INFO ](PolarisAgentBootStrap) [Bootstrap] start bootStrapStarter:default
......
2024-02-01 21:19:16 [INFO ](PluginCreator) [BootStrap] plugin spring-cloud-hoxton-plugin has been
loading
```

虚拟机场景

## 步骤1：下载 java agent

获取最新的 polaris-java-agent 的版本，并下载二进制 zip 包。点击下载 。

## 步骤2：修改配置文件

1. 解压下载后到的 zip 包，进入 polaris-java-agent 的目录，并修改 conf/polaris-agent.conf 文件，填写需要使用的插件名称（需要按照应用的 spring-cloud 版本进行填写），内容示例如下：

```
plugins.enable=spring-cloud-2020-plugin
```

2. 进入 conf/plugin/spring-cloud-2020（不同插件有不同目录，需要按照上一步填写的插件来选择目录）目录中，修改 application.properties 文件：

   2.1 填写正确的应用名（spring.application.name）。

   2.2 填写您购买的北极星服务端的地址信息。服务端地址查看详细参见：引擎管理 > 客户端访问地址 。

```
# 应用名称(必填)
spring.application.name=testSvcName
# 配置北极星服务端地址
spring.cloud.polaris.address=grpc\://9.134.5.52\:8091
```

### 步骤3：添加应用的启动参数

在应用的启动参数中，添加 `-javaagent:<java-agent安装目录>/polaris-agent-core-bootstrap.jar` ，然后重启应用，即可完成 java-agent 的注入。

> **说明：**
> 如果使用 IDEA 进行本地调试，则需要在 VM Option 下添加 `-javaagent` 的参数。

# Java Agent 支持无损上下线

最近更新时间：2024-06-20 16:22:31

## 使用场景

微服务发布过程中，可能会因为服务的变更造成流量的错误或中断。Spring Cloud Tencent 提供了插件来实现无损上下线，其原理为：

- 无损上线：在服务启动后，等待服务完全就绪后再注册到注册中心（无配置健康探测接口时为延迟注册），并对外提供服务。然后再结合K8s生命周期，滚动更新下一个节点。
- 无损下线：在服务停止前，先从注册中心注销，拒绝新的请求，等待旧的请求处理完毕后再下线服务。

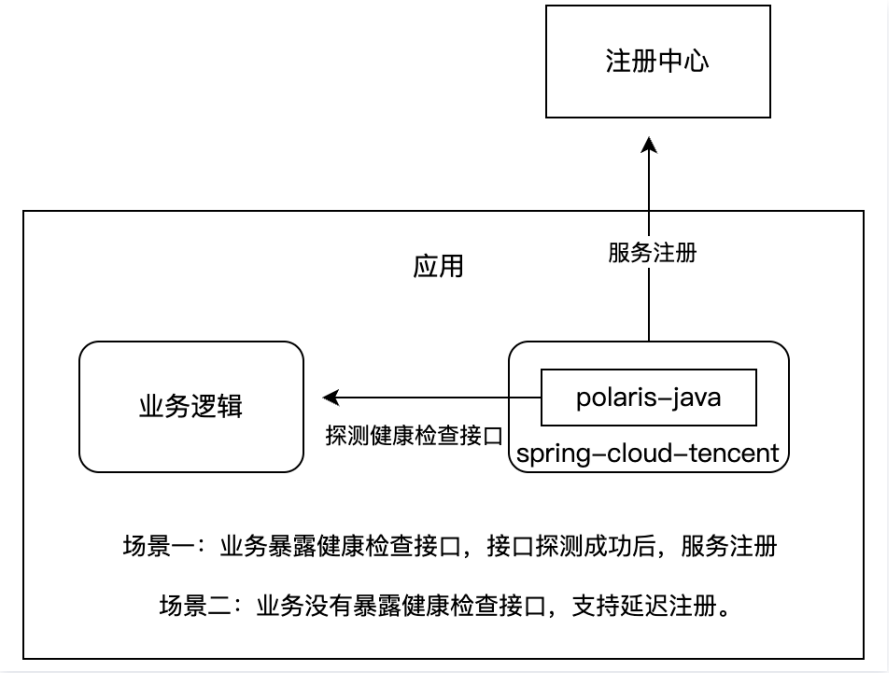
## 无损上线

TSE 北极星支持以下 2 种方式支持服务无损上线：服务就绪/延迟注册 和 服务注册就绪检查 。

### 方案一：服务就绪/延迟注册

在一些场景下，服务支持延迟加载，启动后异步加载一些资源，例如：服务需要从文件存储 COS 获取数据或者文件，待数据或者文件拉取完成后才能对外提供服务。如果在应用启动后直接注册服务，会导致服务实际没有就绪而调用失败。因此通过确保服务就绪之后，再进行注册从而对外提供服务，可以确保服务平滑、无损上线。
TSE 北极星支持以下两种服务就绪/延迟注册场景：

- 场景一：业务暴露健康检查接口，接口探测成功后，服务注册。
- 场景二：业务没有暴露健康检查接口，则延迟一段时间服务注册，默认延迟注册时长30s，可通过配置自定义延迟注册的时长。




#### 操作步骤

**步骤1**：服务通过 Java Agent 接入北极星。具体步骤参见：Spring Cloud 使用 Java Agent 。
**步骤2**：在上一步接入操作中 步骤3 SpringCloud 应用 yaml 文件中声明 Java Agent 标签 中配置无损上下线相关配置项。具体参数如下：

| 配置项Key | 默认值 | 是否必填 | 配置项说明 |
| --- | --- | --- | --- |
| spring.cloud.polaris.lossless.enabled | false | 否 | 无损上下线开关 |
| spring.cloud.polaris.lossless.health-check-path | 无 | 否 | 业务程序健康探测接口 |

| | | | |
|---|---|---|---|
| spring.cloud.polaris.lossless.delay-register-interval | 30000 | 否 | 无配置业务程序健康探测接口，延迟注册时间。默认 30000（单位ms） |
| spring.cloud.polaris.lossless.health-check-interval | 5000 | 否 | 配置业务程序健康探测接口后，健康探测间隔。默认 5000（单位ms） |

**配置示例：**

```
apiVersion: apps/v1
kind: Deployment
......
spec:
......
  template:
    metadata:
      annotations:
        # 声明需要往这个POD中注入javaagent
        polarismesh.cn/javaagent: "true"
        # 声明应用的框架类型，对于SpringCloud应用则填写spring-cloud
        polarismesh.cn/javaagentFrameworkName: spring-cloud
        # 声明应用的框架版本，当前支持hoxton, 2023
        polarismesh.cn/javaagentFrameworkVersion: hoxton
        # 声明java-agent包的镜像版本，可用版本:https://github.com/polarismesh/polaris-java-agent/releases
        polarismesh.cn/javaagentVersion: 1.7.0-RC3
        # 用户自定义的JavaAgent配置，不填写的配置则使用默认配置，格式为JSON。具体配置查看：
https://github.com/polarismesh/polaris-
controller/blob/main/deploy/kubernetes_v1.22/kubernetes/javaagent-configmap.yaml
        # 无损上线配置示例：
        # polarismesh.cn/javaagentConfig: "{\"spring.cloud.polaris.lossless.enabled\": \"true\",
\"spring.cloud.polaris.lossless.delay-register-interval\": \"30000\"}"
......
```

## 方案二：服务注册就绪检查

通常 K8s 提供就绪检查机制来对实例在就绪前进行健康检查，它一般会简单地认为端口起来了应用即处于就绪状态。但是实际上，应用端口起来了和服务成功注册之间存在 gap，这样会造成服务没有成功注册，旧的应用实例就被下线，下一个节点开始部署。最终导致消费端调用异常。
TSE 北极星提供服务注册状态的接口和端口，以配合 K8s 就绪检查，当应用注册完成返回200状态码，帮助 K8s 判定应用已就绪；未完成注册返回500状态码，帮助 K8s 判定应用未就绪。尤其在实例滚动更新的过程中，等实例就绪后才滚动更新下一个节点。

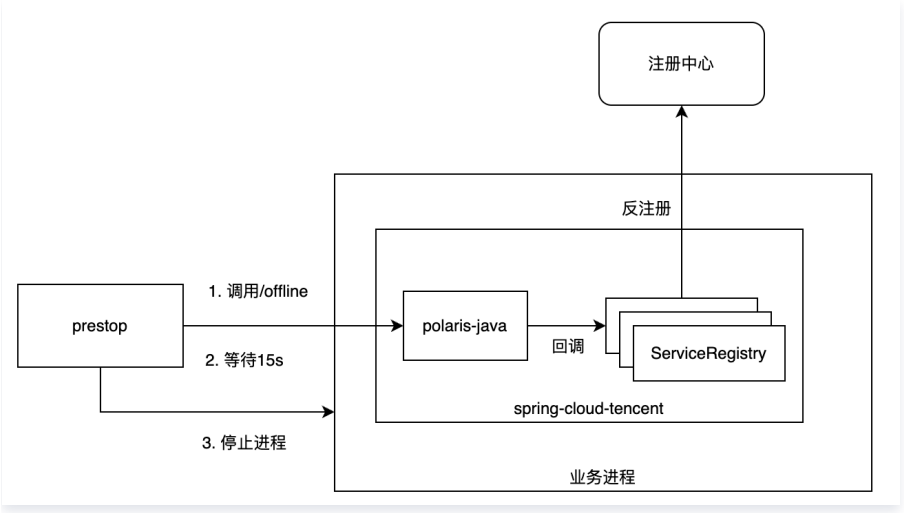
## 操作步骤：

**步骤1**：服务通过 Java Agent 接入北极星。具体步骤参见 Spring Cloud 使用 Java Agent 。
**步骤2**：在上一步接入操作中 步骤3 SpringCloud 应用 yaml 文件中声明 Java Agent 标签 中打开无损上下线的开关。具体参数参见 方案一 。
**步骤3**：在 容器服务 TKE 等 K8s 应用部署平台中配置就绪检查，如下图所示。

- 路径：/online。
- 端口：28080。


## 无损下线

应用在滚动发布或者下线过程中，被调方服务实例向注册中心发起反注册，主调方从注册中心更新 IP 的过程中，存在时间 gap，导致依然有可能调用到已经下线的实例，从而调用失败。
TSE 北极星提供服务无损下线的接口：/offline，结合 K8s 的生命周期，实现服务无损下线。整体流程如下：




### 操作步骤：

**步骤1**：服务通过 Java Agent 接入北极星。具体步骤参见 Spring Cloud 使用 Java Agent 。
**步骤2**：在上一步接入操作中 步骤3 SpringCloud 应用 yaml 文件中声明 Java Agent 标签 中打开无损上下线的开关。具体参数参见 方案一 。
**步骤3**：在 容器服务 TKE 等 K8s 应用部署平台中配置 preStop 生命周期检查。
preStop 配置检查指令：`curl -X PUT http://localhost:28080/offline && sleep 20`

# Java Agent 支持配置管理

最近更新时间：2025-06-11 15:55:02

## 场景说明

该文档指导您通过 Java Agent 接入北极星时，使用配置管理能力的操作步骤。

## 操作步骤

> **说明：**
> Java Agent 的版本要 2.0.1.0及以上。

### 步骤1：通过 Java Agent 接入北极星

服务通过 Java Agent 接入北极星。具体步骤参见： Spring Cloud 使用 Java Agent 。

### 步骤2：配置管理

在上一步接入操作中 步骤3 Spring Cloud 应用 yaml 文件中声明 Java Agent 标签 中配置相关项。具体参数如下：

| 配置项 | 配置值 | 是否必填 | 配置项说明 |
| --- | --- | --- | --- |
| spring.config.import | optional:polaris | 是 | 使用北极星的配置中心 |
| spring.cloud.polaris.config.enabled | true | 是 | 开启北极星配置中心功能 |
| spring.cloud.polaris.config.address | 例如 grpc://127.0.0.1:8093 | 是 | 北极星的配置中心地址 |
| spring.cloud.polaris.config.groups[0].name | 例如 service-provider-2022 | 是 | 北极星配置分组名称 |
| spring.cloud.polaris.config.groups[0].files[0] | config/callee.properties | 是 | 指定北极星配置分组文件名称 |

配置示例：

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: service-provider-2022-a
  name: service-provider-2022-a
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-provider-2022-a
  template:
    metadata:
      labels:
        app: service-provider-2022-a
      annotations:
        # 声明需要往这个POD中注入javaagent
        polarismesh.cn/javaagent: "true"
        # 声明应用的框架类型，对于SpringCloud应用则填写spring-cloud
```

```
        polarismesh.cn/javaagentFrameworkName: spring-cloud
        # 声明应用的框架版本，当前支持hoxton, 2022, 2023
        polarismesh.cn/javaagentFrameworkVersion: 2022
        # 声明javaagent包的镜像版本，配置管理需要使用2.0.1之后的版本，可用版
本:https://github.com/polarismesh/polaris-java-agent/releases
        polarismesh.cn/javaagentVersion: 2.0.1.0
        # 用户自定义的JavaAgent配置，不填写的配置则使用默认配置，格式为JSON。
        polarismesh.cn/javaagentConfig: |
          {
            "spring.config.import": "optional:polaris",
            "spring.cloud.polaris.config.enabled": "true",
            "spring.cloud.polaris.config.address": "grpc://127.0.0.1:8903",
            "spring.cloud.polaris.config.groups[0].name": "service-provider-2022",
            "spring.cloud.polaris.config.groups[0].files[0]": "config/callee.properties",
            "spring.cloud.polaris.lossless.enabled": "true"
          }
    spec:
      containers:
        - image: polarismesh/polaris-javaagent-demo-sc-quickstart-2022-provider-a:2.0.1.0-java17
          imagePullPolicy: Always
          name: provider
          resources:
            limits:
              cpu: "500m"
              memory: 1000Mi
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          command:
            - /bin/bash
            - -c
            - cd /app && java -Dserver.port=65001 -Dspring.cloud.nacos.config.enabled=false -jar
main.jar
          lifecycle:
            preStop:
              exec:
                command: [ "curl","-X","PUT","http://127.0.0.1:28080/offline","&&","sleep","30" ]
          readinessProbe:
            httpGet:
              path: /online
              port: 28080
            initialDelaySeconds: 3
            periodSeconds: 3
      restartPolicy: Always
```

# 使用 Java SDK
# Polaris - Java 接入

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文通过一个 demo 进行应用使用 `polaris-java` 接入微服务引擎托管的 Polaris（北极星）的全流程操作演示，帮助您快速了解如何使用北极星网格。

## 前提条件

- 已创建 Polaris（北极星）实例，请参见 创建 Polaris（实例）。
- 下载 Github 的 demo 源码 到本地并解压。
- 本地编译构建打包机器环境已安装了Java JDK、Maven，并且能够访问 Maven 中央库。
- 根据您自身的业务，已准备好业务部署的资源， `虚拟机部署` 、 `容器化部署` 和 `TEM 部署` 选择其中一种方式即可。
  - **虚拟机部署**已创建 CVM 虚拟机，请参见 创建 CVM 虚拟机。
  - **容器化部署**已创建 TKE 容器集群，请参见 创建 TKE 集群。
  - **TEM部署**已创建 TEM 环境，请参见 创建 TEM 环境。

## 操作步骤

1. 登录 TSF 控制台。
2. 在左侧导航栏选择Polaris（北极星），进入北极星实例引擎列表页后单击目标实例进入实例信息页。
3. 在引擎管理-实例信息页查看访问地址，polaris-java 应用访问使用 gRPC 端口（8091）：


4. 修改 demo 中的注册中心地址。

4.1 在下载到本地的 demo 源码目录 下，分别找到
`quickstart-example\quickstart-example-provider\src\main\resources\polaris.yml` 和
`quickstart-example\quickstart-example-consumer\src\main\resources\polaris.yml` 两个文件。

4.2 添加微服务引擎北极星网格地址到项目配置文件中（以
`quickstart-example\quickstart-example-consumer\src\main\resources\polaris.yml` 为例）。

```
global:
    serverConnector:
    addresses:
    - 10.0.4.6:8091
```

5. 打包 demo 源码成 jar 包。

5.1 在 `quickstart-example` 源码根目录下，打开 cmd 命令，执行 mvn clean package 命令，对项目进行打包编译。

5.2 编译成功后，生成如下表所示的2个 Jar 包。

| 软件包所在目录 | 软件包名称 | 说明 |
| --- | --- | --- |
| \quickstart-example\quickstart-example-provider\target | quickstart-example-provider-${version}-SNAPSHOT.jar | 服务生产者 |
| \quickstart-example\quickstart-example-consumer\target | quickstart-example-consumer-${version}-SNAPSHOT.jar | 服务消费者 |

6. 部署 provider 和 consumer 微服务应用，根据您业务实际选择**虚拟机部署**、**容器化部署**以及 TEM 部署中的一种部署方式即可。

6.1 **虚拟机部署**

- 上传 Jar 包至 CVM 实例。
- 执行启动命令进行启动：

```
nohup java -jar [jar包名称] &
```

6.2 **容器化部署**

- 编写 dockerfile 生成镜像，参考：

```
     FROM java:8


    ADD ./quickstart-example-provider-${VERSION}.jar /root/app.jar

ENTRYPOINT  ["java","-jar","/root/app.jar"]
```

- 通过 TKE 部署并运行镜像。

6.3 **TEM 部署**

- 选择 TEM 环境，注意所选择的环境，其依赖的 VPC，必须和上面已经创建的北极星网格实例所依赖的 VPC 一致：


- 在已选择的环境中，新建 TEM 应用，相关参数填写参考：

- 部署应用，相关参数填写请参考（端口号映射，consumer 默认端口号为15700， provider 默认端口号为15800）：


- 查看访问路径，consumer 应用部署完后，可以在**基本信息 > 访问配置**中查看访问地址，如需公网访问，可以**编辑并更新**开启公网访问：


7. 确认部署结果。

7.1 进入前面提到的微北极星网格实例页面。

- 选择**服务管理 > 服务列表**，查看微服务 EchoServerGRPC 的实例数量：
- 若实例数量值不为0，则表示已经成功接入微服务引擎。
- 若实例数量为0，或者找不到 EchoServerGRPC 服务名，则表示微服务应用接入微服务引擎失败。


7.2 调用 consumer 的 HTTP 接口。执行 http 调用，其中 `${app.port}` 替换为 consumer 的监听端口（默认为16011），`${add.address}` 则替换为 consumer 暴露的地址。

```
curl -L -X GET 'http://${add.address}:${app.port}/echo?value=hello_world'
```

**预期返回值：** `echo: hello_world`

# 注册发现

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 polaris-sdk 的方式接入 Polaris（北极星），并实现服务注册与发现。

## 前提条件

在开发前，请确保您已经下载并安装了 Java 和 Maven。

## 操作步骤

为了方便您快速接入，我们为您准备了 Demo 应用，点击下载 。

### 步骤1：引入北极星依赖

#### 1. 引入 polaris sdk 依赖

修改应用根目录下的 `pom.xml` ，为 `polaris-java` 添加 `dependencyManagement` ：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.polaris</groupId>
            <artifactId>polaris-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **说明：**
> polaris-sdk 版本信息请参见：版本信息 。

#### 2. 引入 polaris starter

```
<dependencies>
    <dependency>
        <groupId>com.tencent.polaris</groupId>
        <artifactId>polaris-all</artifactId>
    </dependency>
</dependencies>
```

### 步骤2：添加北极星配置文件 polaris.yml

1. 在项目的 main/resources 目录下创建 polaris.yml 文件用于初始化 polaris-java SDK。
2. 在 polaris.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址查看详见：引擎管理 > 客户端访问地址 。

```
global:
  serverConnectors:
    - id: polaris
      protocol: grpc
      addresses:
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        - 127.0.0.1:8091
```

```
  #描述：监控及日志数据上报相关配置
  statReporter:
    #描述：是否启用上报
    enable: true
    plugin:
      prometheus:
        type: push
        # 描述：设置 pushgateway 的地址，仅 type == push 时生效
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        address: 127.0.0.1:9091
        #描述:设置metric数据推送到pushgateway的执行周期
        #范围:[1s:...]，默认值:10s
        pushInterval: 10s
```

更多 polaris.yml 配置信息，请 default-config.yml 。

## 步骤3：应用开发

### 1. 服务注册（服务生产者）

#### 1.1 初始化 DiscoveryAPI 实例

```
import com.tencent.polaris.factory.api.DiscoveryAPIFactory;

public static void main(String[] args) throws Exception {
    ProviderAPI providerAPI = DiscoveryAPIFactory.createProviderAPI();
}
```

#### 1.2 注册请求体，设置健康检查、metadata 等信息。

```
// InstanceRegisterRequest 注册服务请求
type InstanceRegisterRequest struct {
    // 必选，服务名
    Service string
    // 必选，命名空间
    Namespace string
    // 必选，服务监听host，支持IPv6地址
    Host string
    // 必选，服务实例监听port
    Port int
    // 可选，资源访问Token，即用户/用户组访问凭据，仅当服务端开启客户端鉴权时才需配置
    ServiceToken string
    // 以下字段可选，默认nil表示客户端不配置，使用服务端配置
    // 服务协议
    Protocol *string
    // 服务权重，默认100，范围0-10000
    Weight *int
    // 实例提供服务版本号
    Version *string
    // 用户自定义metadata信息
    Metadata map[string]string
    // 该服务实例是否健康，默认健康
    Healthy *bool
    // 该服务实例是否隔离，默认不隔离
    Isolate *bool
    // 设置心跳健康检查ttl，单位为s，不填默认为5s，TTL的取值范围为 (0s, 60s]
    // 开启了心跳健康检查，客户端必须以TTL间隔上报心跳
    // 健康检查服务器3个TTL未受到心跳则将实例置为不健康
    TTL *int
```

```
    // Location 当前注册实例的地理位置信息，主要用于就近路由
    Location *Location
    // 可选，单次查询超时时间，默认直接获取全局的超时配置
    // 用户总最大超时时间为(1+RetryCount) * Timeout
    Timeout *time.Duration
    // 可选，重试次数，默认直接获取全局的超时配置
    RetryCount *int
}
```

> **说明：**
> 鉴权文档可以参见：权限控制 。

### 1.3 发起注册请求

在初始化完 `InstanceRegisterRequest` 结构体后，只需要调用 `ProviderAPI.RegisterInstance` 方法即可完成实例注册，并且 `RegisterInstance` 方法内部会自动维护实例的心跳上报。

```
InstanceRegisterResponse registerResp = providerAPI.registerInstance(registerRequest)
```

## 2. 服务发现（服务消费者）

### 2.1 初始化 polaris sdk 实例

```
import com.tencent.polaris.factory.api.DiscoveryAPIFactory;


public static void main(String[] args) throws Exception {
    ConsumerAPI consumerAPI = DiscoveryAPIFactory.createConsumerAPI();
}
```

### 2.2 获取服务提供方的实例信息

北极星提供三种获取目标服务实例的方法：GetAllInstances、GetHealthyInstances、GetOneInstances，分别获取全部实例、健康实例和一个实例。详情如下：

#### （1）GetAllInstances

直接返回目标服务下的所有实例，包括不健康、隔离、权重为0、被熔断的实例，都会在返回的实例列表中。

```
GetAllInstancesRequest request = new GetAllInstancesRequest();
// 设置服务命名空间
request.setNamespace(String namespace);
// 设置服务名称
request.setService(String service);
// 设置超时时间
request.setTimeoutMs(long timeoutMs);

// 调用 ConsumerAPI 执行该请求
consumerAPI.getAllInstance(request);
```

#### （2）GetHealthyInstances

每次获取一批可用服务提供者实例。该方法默认会过滤掉不健康、隔离、权重为0、被熔断的实例。

```
GetInstancesRequest request = new GetInstancesRequest();
// 设置服务命名空间
request.setNamespace(String namespace);
// 设置服务名称
request.setService(String service);
```

```
// 可选，设置主调服务信息，只用于路由规则匹配
SourceService serviceInfo = new SourceService();
// 设置主调服务命名空间
serviceInfo.setNamespace(String namespace);
// 设置主调服务名称
serviceInfo.setService(String service);
// 设置主调方的请求标签信息
serviceInfo.setArguments(Set<RouteArgument> arguments);
request.setServiceInfo(serviceInfo);

// 设置超时时间
request.setTimeoutMs(long timeoutMs);

// 调用 ConsumerAPI 执行该请求
consumerAPI.getInstances(request);
```

### （3）GetOneInstances

每次仅获取一个可用服务提供者实例，该方法会依次执行路由、负载均衡流程。该方法默认会过滤掉不健康、隔离、权重为0、被熔断的实例。

> ⚠ **注意：**
>
> 执行路由流程的条件
>
> - 配置了 `GetOneInstanceRequest.ServiceInfo.Metadata` 属性，会触发自定义路由流程。
> - 设置了 `GetOneInstanceRequest.Metadata` 属性，会触发元数据路由流程。

```
public class Criteria {
    /**
     * 指定负载均衡策略
     */
    private String lbPolicy;
    /**
     * 一致性hash的key
     */
    private String hashKey;
}

GetOneInstanceRequest request = new GetOneInstanceRequest();
// 设置服务命名空间
request.setNamespace(String namespace);
// 设置服务名称
request.setService(String service);
// 可选，元数据信息，仅用于dstMetadata路由插件的过滤
request.setMetadata(Map<String, String> metadata);
// 可选，设置元数据路由兜底措施
// 当前支持的元数据路由兜底措施如下
// - 默认不降级：METADATAFAILOVERNONE("metadataFailoverNone")
// - 降级返回所有节点：METADATAFAILOVERALL("metadataFailoverAll")
// - 返回不包含元数据路由key的节点：METADATAFAILOVERNOTKEY("metadataFailoverNoKey")
request.setMetadataFailoverType();
// 可选，对应自定义路由规则中请求标签中的方法(Method)
request.setMethod(String method);

// 如果需要走 Hash 负载均衡的话，需要设置
Criteria criteria = new Criteria();
request.setCriteria(criteria);

// 可选，设置主调服务信息，只用于路由规则匹配
SourceService serviceInfo = new SourceService();
```

```
// 设置主调服务命名空间
serviceInfo.setNamespace(String namespace);
// 设置主调服务名称
serviceInfo.setService(String service);
// 设置主调方的请求标签信息
serviceInfo.setArguments(Set<RouteArgument> arguments);
request.setServiceInfo(serviceInfo);

// 设置超时时间
request.setTimeoutMs(long timeoutMs);

// 调用 ConsumerAPI 执行该请求
consumerAPI.getOneInstance(request);
```

## 步骤4：应用部署

Demo 应用部署可参见：Demo 应用部署 。

# 负载均衡

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 polaris-sdk 的方式接入 Polaris（北极星），并实现负载均衡功能。

## 前提条件

在开发前，请确保您已经下载并安装了 Java 和 Maven。

## 操作步骤

### 步骤1：引入北极星依赖

#### 1. 引入 polaris sdk 依赖

修改应用根目录下的 `pom.xml`，为 `polaris-java` 添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.polaris</groupId>
            <artifactId>polaris-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **说明：**
> polaris-sdk 版本信息请参见：版本信息 。

#### 2. 引入 polaris starter

```
<dependencies>
    <dependency>
        <groupId>com.tencent.polaris</groupId>
        <artifactId>polaris-all</artifactId>
    </dependency>
</dependencies>
```

### 步骤2：添加北极星配置文件 polaris.yaml

1. 在项目的 main/resources 目录下创建 polaris.yml 文件用于初始化 polaris-java SDK。
2. 在 polaris.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址详细参见：引擎管理 > 客户端访问地址 。

```
global:
  serverConnectors:
    - id: polaris
      protocol: grpc
      addresses:
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        - 127.0.0.1:8091
  #描述：监控及日志数据上报相关配置
  statReporter:
```

```
  #描述：是否启用上报
  enable: true
  plugin:
    prometheus:
      type: push
      # 描述：设置 pushgateway 的地址，仅 type == push 时生效
      # 地址需要替换成您创建的北极星引擎的客户端访问地址。
      address: 127.0.0.1:9091
      #描述：设置metric数据推送到pushgateway的执行周期
      #范围：[1s:...]，默认值：10s
      pushInterval: 10s
```

更多 polaris.yml 配置信息，请参见 default-config.yml 。

## 步骤3：应用开发

### 1. 初始化 RouterAPI 实例

```
import com.tencent.polaris.factory.api.RouterAPIFactory;

public static void main(String[] args) throws Exception {
    RouterAPI routerAPI = RouterAPIFactory.createRouterAPI();
}
```

### 2. 注册请求体，设置执行负载均衡的实例列表、负载均衡策略等信息。

```
public class Criteria {
    // 一致性hash的key
    private String hashKey;
}

ProcessLoadBalanceRequest request = new ProcessLoadBalanceRequest();
// 设置需要参与负载均衡的服务实例
request.setDstInstances(ServiceInstances dstInstances);
// 设置负载均衡策略
// 当前支持的负载均衡策略如下
// - 权重随机负载均衡：weightedRandom
// - 权重一致性负载均衡：ringHash
request.setLbPolicy(String lbPolicy);

// 如果需要走 Hash 负载均衡的话，需要设置
Criteria criteria = new Criteria();
request.setCriteria(criteria);
```

### 3. 负载均衡

您在使用 `ConsumerAPI.getAllInstances` 或者 `ConsumerAPI.getInstances` 获取到服务实例列表后，完成 `ProcessLoadBalanceRequest` 初始化，只需要调用 `RouterAPI.processLoadBalance` 方法即可完成负载均衡。

```
ProcessLoadBalanceResponse resp = routerAPI.processLoadBalance(request)
```

# 动态路由

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 polaris-sdk 的方式接入 Polaris（北极星），并实现服务路由功能。

## 前提条件

在开发前，请确保您已经下载并安装了 Java 和 Maven。

## 操作步骤

### 步骤1：引入北极星依赖

#### 1. 引入 polaris sdk 依赖

修改应用根目录下的 `pom.xml`，为 `polaris-java` 添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.polaris</groupId>
            <artifactId>polaris-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **说明：**
> polaris-sdk 版本信息请参见：版本信息 。

#### 2. 引入 polaris starter

```
<dependencies>
    <dependency>
        <groupId>com.tencent.polaris</groupId>
        <artifactId>polaris-all</artifactId>
    </dependency>
</dependencies>
```

### 步骤2：添加北极星配置文件 polaris.yaml

1. 在项目的 main/resources 目录下创建 polaris.yml 文件用于初始化 polaris-java SDK。
2. 在 polaris.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址查看详见：引擎管理 > 客户端访问地址 。

```
global:
  serverConnectors:
    - id: polaris
      protocol: grpc
      addresses:
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        - 127.0.0.1:8091
  #描述：监控及日志数据上报相关配置
  statReporter:
```

```
    #描述：是否启用上报
    enable: true
    plugin:
      prometheus:
        type: push
        # 描述：设置 pushgateway 的地址，仅 type == push 时生效
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        address: 127.0.0.1:9091
        #描述：设置metric数据推送到pushgateway的执行周期
        #范围：[1s:...]，默认值：10s
        pushInterval: 10s
```

更多 polaris.yml 配置信息，请参见default-config.yml 。

## 步骤3：应用开发

### 1. 初始化 routerAPI 实例

```
import com.tencent.polaris.factory.api.RouterAPIFactory;

public static void main(String[] args) throws Exception {
    RouterAPI routerAPI = RouterAPIFactory.createRouterAPI();
}
```

### 2. 注册请求体：设置路由策略、实例列表、metedata 等信息。

```
// ProcessRoutersRequest 执行路由请求结构体
type ProcessRoutersRequest struct {
    // 可选参数，设置本次路由请求期望执行的路由插件
    // 当前支持的路由插件如下
    // - 自定义路由：ruleBasedRouter
    // - 就近路由：nearbyBasedRouter
    // - 元数据路由：dstMetaRouter
    Routers []string
    // 可选参数，主调服务信息，你可以通过 ServiceInfo.Metadata 设置本次请求的流量标签信息
    SourceService ServiceInfo
    // 必选参数，待执行服务路由的实例列表
    // 1. InstancesResponse, returned from ConsumerAPI.GetAllInstances.
    // 2. DefaultServiceInstances, for user to construct manually.
    DstInstances ServiceInstances
    // 可选参数，对应路由规则中的方法($method)标签
    Method string
    // 可选，单次查询超时时间，默认直接获取全局的超时配置
    // 用户总最大超时时间为(1+RetryCount) * Timeout
    Timeout *time.Duration
    // 可选，重试次数，默认直接获取全局的超时配置
    RetryCount *int
}
```

**说明：**

如果当前 `ProcessRoutersRequest` 还不支持 `AddArgument` 方法，同时服务端版本 >= 1.12.0， `SourceService.Metadata` 对应的 key 名称如下：

- 路径：`$path`
- 方法：`$method`
- 请求头：`$header.{标签键}`
- 请求参数：`$query.{标签键}`

- 请求COOKIE: `$cookie.{标签键}`
- 主调IP: `$caller_ip`
- 自定义: `{标签键}`

### 3. 执行服务路由

您在初始化完 `ProcessRoutersRequest` 结构体后，只需要调用 `RouterAPI.processRouters` 方法即可完成服务路由。

```
ProcessRoutersResponse resp = routerAPI.processRouters(registerRequest)
```

# 就近访问

最近更新时间：2024-01-22 16:19:31

## 操作场景

本文通过一个 demo 进行 Java 应用接入微服务引擎托管的 PolarisMesh 治理中心的全流程操作演示，帮助您快速了解如何使用北极星网格的就近路由能力。

## 前提条件

- 已创建 PolarisMesh 北极星网格，请参见 创建 PolarisMesh 治理中心 。
- 下载 Github 的 demo 源码 到本地并解压。
- 本地编译构建打包机器环境已安装了 Java 环境，并且能够访问 Github。
- 根据您自身的业务，已准备好业务部署的资源，虚拟机部署和容器化部署选择其中一种方式即可。
  - **虚拟机部署**已创建 CVM 虚拟机，请参见 创建 CVM 虚拟机 。
  - **容器化部署**已创建 TKE 容器集群，请参见 创建 TKE 集群 。

## 操作步骤

1. 登录 TSE 控制台 。
2. 在**北极星网格**下的 **polarismesh** 页面，单击页面左上方下拉列表，选择目标地域。
3. 单击目标引擎的“ID”，进入基本信息页面。
4. 查看访问地址，Java 应用访问使用 gRPC 端口（8091）：


5. 修改 demo 中的注册中心地址：
   - 在下载到本地的 demo 源码 目录下，分别找到 `quickstart-example-provider/src/main/resources/polaris.yml` 以及 `quickstart-example-consumer/src/main/resources/polaris.yml` 文件。
   - 添加微服务引擎北极星网格地址以及就近路由相关配置到项目配置文件中（这里以 `quickstart-example-provider/src/main/resources/polaris.yml` 为例）。

**通过环境变量获取 SDK 所在地域信息**

- 通过腾讯云 API 获取当前 CVM /容器 POD 的地域信息

```
export ZONE=$(curl http://metadata.tencentyun.com/latest/meta-data/placement/region)
export CAMPUS=$(curl http://metadata.tencentyun.com/latest/meta-data/placement/zone)
```

- 修改 polaris.yaml 文件

```
global:
  serverConnector:
    addresses:
      # 设置北极星服务端接入地址
      - 10.0.4.6:8091
```

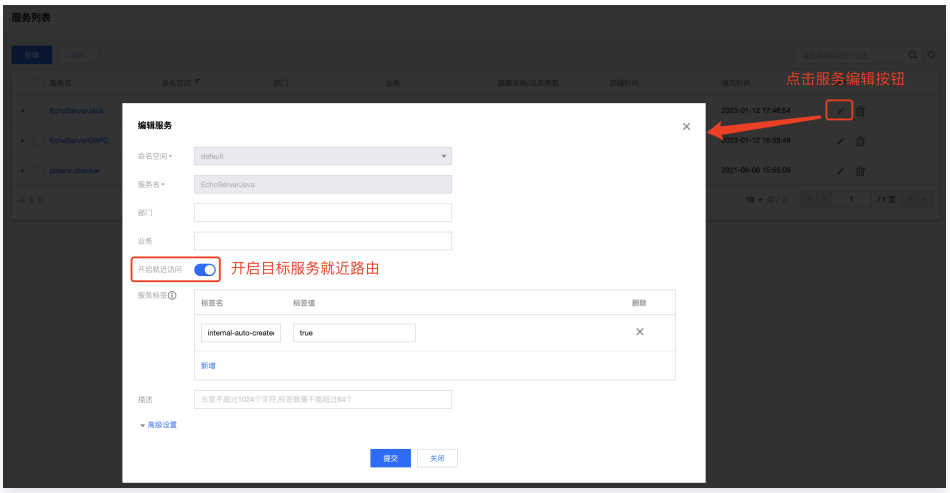
```
# 设置服务调用相关参数
consumer:
  serviceRouter:
    plugin:
      # 设置就近路由插件配置
      nearbyBasedRouter:
        # 描述：就近路由的最小匹配级别，需要显示设置
        # 范围： zone（腾讯云地域信息，eg： ap-guangzhou）、campus（腾讯云可用区， eg： ap-guangzhou-3）
        matchLevel: campus
```

**通过腾讯云 API 获取 SDK 所在地域信息**

```
global:
  serverConnector:
    addresses:
      # 设置北极星服务端接入地址
      - 10.0.4.6:8091
  # 设置 SDK 获取自身地理位置信息的方式
  location:
    providers:
      # 设置 SDK 通过调用 http 接口获取地域信息数据
      - type: remoteHttp
        options:
          zone: http://metadata.tencentyun.com/latest/meta-data/placement/region
          campus: http://metadata.tencentyun.com/latest/meta-data/placement/zone
# 设置服务调用相关参数
consumer:
  serviceRouter:
    plugin:
      # 设置就近路由插件配置
      nearbyBasedRouter:
        # 描述：就近路由的最小匹配级别，需要显示设置
        # 范围： zone（腾讯云地域信息，eg： ap-guangzhou）、campus（腾讯云可用区， eg： ap-guangzhou-3）
        matchLevel: campus
```

6. 部署 provider 和 consumer 微服务应用。详情请参见 Polaris-Java Demo 部署 。
7. 确认部署结果。
   - 进入前面提到的微服务引擎北极星网格实例页面。
   - 选择**注册中心 > 服务列表**，查看微服务 EchoServerJava 的实例数量。
   - 若实例数量值不为0，则表示已经成功接入微服务引擎。同时确认实例所处的腾讯云地址位置信息。
   - 若实例数量为0，或者找不到 EchoServerJava 服务名，则表示微服务应用接入微服务引擎失败。

- 开启 EchoServerJava 的就近路由。




- 调用 consumer 的 HTTP 接口，执行 http 调用，其中 `${app.port}` 替换为 consumer 的监听端口（默认为16011），`${add.address}` 则替换为 consumer 暴露的地址。

```
curl -L -X GET 'http://${add.address}:${app.port}/echo?value=hello_world'

# 预期返回与 consumer 同腾讯云可用区的服务提供者实例信息
预期返回值："echo: hello_world, from: ${目标实例端口}, location: ${实例地理位置信息}";
```

# 服务限流

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 polaris-sdk 的方式接入 Polaris（北极星），并实现服务的服务限流功能。

## 前提条件

在开发前，请确保您已经下载并安装了 Java 和 Maven。

## 操作步骤

### 步骤1：引入北极星依赖

#### 1. 引入 polaris sdk 依赖

修改应用根目录下的 `pom.xml`，为 `polaris-java` 添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.polaris</groupId>
            <artifactId>polaris-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **说明：**
> polaris-sdk 版本信息请详细参见：版本信息 。

#### 2. 引入 polaris starter

```
<dependencies>
    <dependency>
        <groupId>com.tencent.polaris</groupId>
        <artifactId>polaris-all</artifactId>
    </dependency>
</dependencies>
```

### 步骤2：添加北极星配置文件 polaris.yml

1. 在项目的 main/resources 目录下创建 polaris.yml 文件用于初始化 polaris-java SDK。
2. 在 polaris.yml 文件中配置应用名、Polaris（北极星）服务端地址等信息。服务端地址详细参见：引擎管理 > 客户端访问地址 。

```
global:
  serverConnectors:
    - id: polaris
      protocol: grpc
      addresses:
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        - 127.0.0.1:8091
  #描述：监控及日志数据上报相关配置
  statReporter:
```

```
    #描述：是否启用上报
    enable: true
    plugin:
      prometheus:
        type: push
        # 描述：设置 pushgateway 的地址，仅 type == push 时生效
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        address: 127.0.0.1:9091
        #描述：设置metric数据推送到pushgateway的执行周期
        #范围：[1s:...]，默认值：10s
        pushInterval: 10s
```

更多 polaris.yml 配置信息，请参见 default-config.yml 。

## 步骤3：应用开发

### 1.初始化 LimitAPI 实例：

```
import com.tencent.polaris.ratelimit.factory.LimitAPIFactory;


public static void main(String[] args) throws Exception {
    LimitAPI limitAPI = LimitAPIFactory.createLimitAPI();
}
```

### 2.请求配额

```
QuotaRequest quotaRequest = new QuotaRequest();
// 设置需要进行限流的服务信息：设置命名空间信息
quotaRequest.setNamespace(String namespace);
// 设置需要进行限流的服务信息：设置服务名称信息
quotaRequest.setService(String service);
// 设置本次被调用的方法信息
quotaRequest.setMethod(String method);
// 设置本次的请求标签
quotaRequest.setArguments(Set<Argument> arguments)
// 设置需要申请的请求配额数量
quotaRequest.setCount(1);
```

**说明：**

如果当前 QuotaRequest 还不支持 AddArgument 方法，同时服务端版本 >= 1.11.0，SetLabels 对应的 key 名称如下：

- 路径: $path
- 方法: $method
- 请求头: $header.{标签键}
- 请求参数: $query.{标签键}
- 主调服务: $caller_service
- 主调IP: $caller_ip
- 自定义: {标签键}

### 3.发起配额申请请求

在接收到请求之后对 QuotaRequest 结构体完成初始化后，只需要调用 LimitAPI.GetQuota 方法即可完成本次请求配额的申请。

```
QuotaResponse resp = limitAPI.getQuota(registerRequest);
```

# 分布式限流

最近更新时间：2023-08-23 16:21:53

## 操作场景

本文针对用户接入微服务引擎托管的 PolarisMesh 治理中心希望使用北极星分布式限流能力。

## 为何需要用户手动创建

用户接入方式存在多 VPC、公网接入两类场景。不同的接入侧用户需要看到不同的服务端 IP，无法自动创建相关北极星系统服务。

## 前提条件

已创建 PolarisMesh 服务治理中心，请参见 创建 PolarisMesh 治理中心。

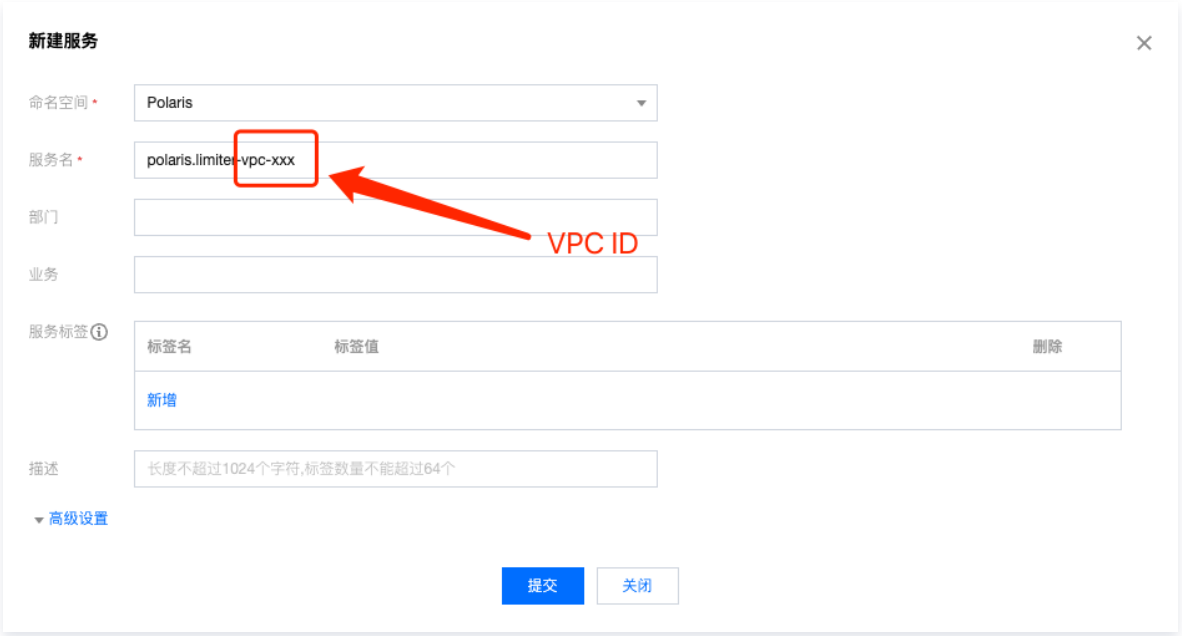
## 操作步骤

1. 登录 TSE 控制台。
2. 在**治理中心**下的 **polarismesh** 页面，单击页面左上方下拉列表，选择目标地域。
3. 单击目标引擎的“ID”，进入基本信息页面。
4. 根据实例信息的控制台。

## 引擎实例未开启客户端公网访问

### 创建 polaris.limiter

1. 在实例的基本信息页面中，记住访问地址中的每个 VPC 内网 IP 地址以及访问端口。
2. 针对每一个内网 VPC 的 IP 地址，在命名空间 **Polaris** 下创建服务 **polaris.limiter-{vpc 名称}**。




3. 单击服务 **polaris.limiter-{vpc 名称}**，进入到服务的实例页面，根据对应上述第一步所记录的 VPC 内网 IP 地址以及端口协议 grpc 创建实例。


4. 按照上述步骤的操作，最终服务 **polaris.limiter-{vpc 名称}** 的实例列表如下：

> **注意：**
> 如果只是单 VPC 接入的场景，则无需参考 **polaris.limiter-{vpc 名称}** 这种命名规则，直接使用 **polaris.limiter** 即可。

**polaris-java 接入相关配置文件说明**

- polaris.yaml 中按照下面示例进行配置二次寻址：

```
global:
  serverConnector:
    addresses:
      - '{对应 vpc 的内网接入地址}:8091'
provider:
  rateLimit:
    enable: true
    limiterNamespace: Polaris
    limiterService: 'polaris.limiter-{vpc 名称}'
```

**spring-cloud-tencent 接入相关配置文件说明**

- 修改 bootstrap.yaml 或者 bootstrap.properties 文件：

```
spring:
  cloud:
    polaris:
```

```
      address: "{对应 vpc 的内网接入地址}:8091"
```

- 在 resources 下创建 polaris.yml 文件。
- 按照下面示例进行配置二次寻址：

```
provider:
  rateLimit:
    enable: true
    limiterNamespace: Polaris
    limiterService: 'polaris.limiter-{vpc 名称}'
```

## 引擎实例开启客户端公网访问

### 创建 polaris.limiter 服务

1. 在实例的基本信息页面中，记住访问地址中的公网 IP 地址以及访问端口。
2. 在命名空间 Polaris下创建服务 polaris.limiter。
3. 单击服务 polaris.limiter，进入到服务的实例页面，根据客户端公网访问 IP 地址以及端口协议 grpc 分别创建实例。


4. 按照上述步骤的操作，最终服务 polaris.limiter 的实例列表如下：

### polaris-java 接入相关配置文件说明

- polaris.yaml 中按照下面示例进行配置二次寻址：

```
global:
```

```
  serverConnector:
    addresses:
      - '{对应公网接入地址}:8091'
provider:
  rateLimit:
    enable: true
    limiterNamespace: Polaris
    limiterService: polaris.limiter
```

**spring-cloud-tencent 接入相关配置文件说明**

- 修改 bootstrap.yaml 或者 bootstrap.properties 文件。

```
spring:
  cloud:
    polaris:
      address: '{对应公网接入地址}:8091'
```

- 在 resources 下创建 polaris.yml 文件。
- 按照下面示例进行配置二次寻址：

```
provider:
  rateLimit:
    enable: true
    limiterNamespace: Polaris
    limiterService: polaris.limiter
```

# 熔断降级

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 polaris-sdk 的方式接入 Polaris（北极星），并实现熔断降级功能。

## 前提条件

在开发前，请确保您已经下载并安装了 Java 和 Maven。

## 操作步骤

### 步骤1：引入北极星依赖

#### 1. 引入 polaris sdk 依赖

修改应用根目录下的 `pom.xml`，为 `polaris-java` 添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.polaris</groupId>
            <artifactId>polaris-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **说明：**
> polaris-sdk 版本信息请参见：版本信息 。

#### 2. 引入 polaris starter

```
<dependencies>
    <dependency>
        <groupId>com.tencent.polaris</groupId>
        <artifactId>polaris-all</artifactId>
    </dependency>
</dependencies>:
```

### 步骤2：添加北极星配置文件 polaris.yaml

1. 在项目的 main/resources 目录下创建 polaris.yml 文件用于初始化 polaris-java SDK。
2. 在 polaris.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址查看详细参见：引擎管理 > 客户端访问地址 。

```
global:
  serverConnectors:
    - id: polaris
      protocol: grpc
      addresses:
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        - 127.0.0.1:8091
  #描述：监控及日志数据上报相关配置
  statReporter:
```

```
    #描述：是否启用上报
    enable: true
    plugin:
      prometheus:
        type: push
        # 描述：设置 pushgateway 的地址，仅 type == push 时生效
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        address: 127.0.0.1:9091
        #描述:设置metric数据推送到pushgateway的执行周期
        #范围:[1s:...]，默认值:10s
        pushInterval: 10s
```

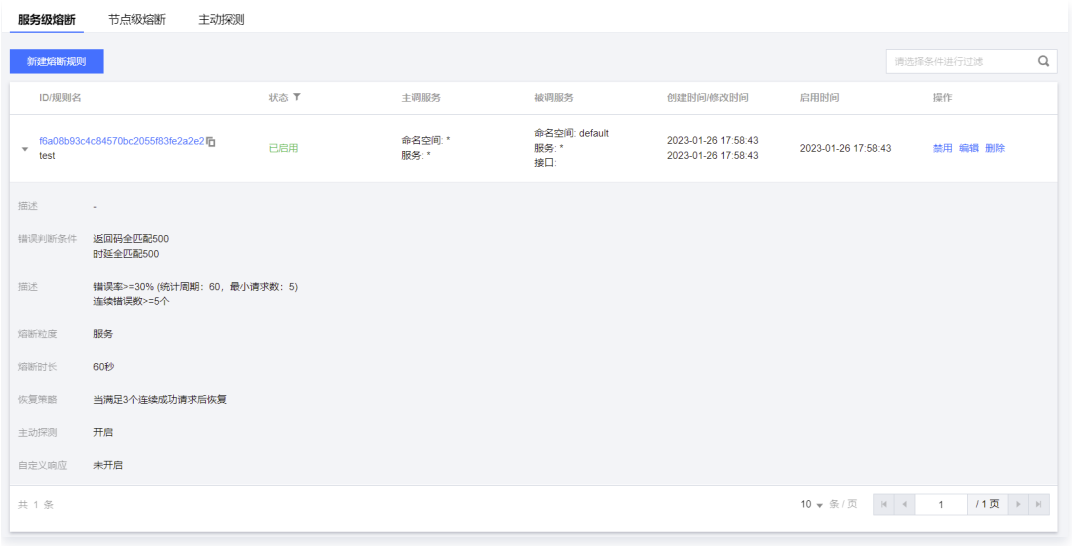
更多 polaris.yml 配置信息，请参见 default-config.yml 。

### 步骤3：使用说明

北极星支持三种维度的熔断降级：服务级、接口级、实例级。

#### 1. 控制台配置熔断规则

请参见 服务治理指南 > 服务治理 > 熔断降级 相关文档，相关界面如下：




熔断规则配置示例：针对 default 命名空间下所有的服务，对于时延大于500毫秒，或者返回码为500的请求，标识为错误请求，一旦一分钟内错误率30%及以上或连续错误数在5个以上，则对服务进行熔断。

### 2. 客户端使用 SDK 进行熔断判断

#### 方法说明：

北极星 Java SDK 提供以下熔断相关的方法，所有的方法都在 `com.tencent.polaris.circuitbreak.api.CircuitBreakAPI` 接口中提供。

- check：检查资源是否可被调用，并对资源获取调用申请。对于半开的资源，如果半开的调用配额申请成功，返回 true，否则返回 false。
- report：该方法供用户在资源调用完成后，上报调用的结果，包括返回码、时延等信息，供熔断逻辑判断。
- makeFunctionalDecorator：创建一个函数调用装饰器 FunctionalDecorator，装饰器可以对 Java 的函数接口进行装饰。装饰后的逻辑，会在函数逻辑调用前，先通过 check 方法检查资源是否可被调用，如果不能被调用，会抛出资源熔断异常（CallAbortedException）。调用完成后，会通过 report 接口上报本次调用结果。

`FunctionalDecorator` 包含以下方法：

  - decorateSupplier：对函数接口 Supplier 进行封装。
  - decorateConsumer：对函数接口 Consumer 进行封装。
  - decorateFunction：对函数 Function 进行封装。
  - decoratePredicate：对函数接口 Predicate 进行封装。

#### 2.2 服务级熔断

```
// 创建CircuitBreakAPI实例
CircuitBreakAPI circuitBreakAPI = CircuitBreakAPIFactory.createCircuitBreakAPI();

// 通过传入服务名(testService1)和命名空间(default)，创建FunctionalDecorator
FunctionalDecoratorRequest makeDecoratorRequest = new FunctionalDecoratorRequest();
        makeDecoratorRequest.setService(new ServiceKey("default", "testService1"));
FunctionalDecorator decorator = circuitBreakAPI.makeFunctionalDecorator(makeDecoratorRequest);

// 封装函数接口
Consumer<Integer> integerConsumer = decorator.decorateConsumer(new Consumer<Integer>() {
    @Override
    public void accept(Integer object) {
      // 执行服务调用...
    }
});

// 通过执行函数接口，进行服务调用
// 在调用过程中，如果出现熔断，会抛出CallAbortedException异常
for (int i = 0; i < 500; i++) {
    try {
       integerConsumer.accept(i);
    } catch(CallAbortedException e) {
       e.printStackTrace();
    }
}
```

### 2.3 接口级熔断

```
// 创建CircuitBreakAPI实例
CircuitBreakAPI circuitBreakAPI = CircuitBreakAPIFactory.createCircuitBreakAPI();

// 通过传入服务名(testService1)、命名空间(default)和方法名(foo)，创建FunctionalDecorator
FunctionalDecoratorRequest makeDecoratorRequest = new FunctionalDecoratorRequest();
makeDecoratorRequest.setService(new ServiceKey("default", "testService1"));
makeDecoratorRequest.setMethod("foo");
FunctionalDecorator decorator = circuitBreakAPI.makeFunctionalDecorator(makeDecoratorRequest);

// 封装函数接口
Consumer<Integer> integerConsumer = decorator.decorateConsumer(new Consumer<Integer>() {
    @Override
    public void accept(Integer object) {
      // 执行服务接口调用...
    }
});

// 通过执行函数接口，进行服务调用
// 在调用过程中，如果出现熔断，会抛出CallAbortedException异常
for (int i = 0; i < 500; i++) {
    try {
       integerConsumer.accept(i);
    } catch(CallAbortedException e) {
       e.printStackTrace();
    }
}
```

### 2.4 实例级熔断

当实例被熔断时，该实例会暂时不接收请求，原本路由到该实例的请求会路由到其他实例。这个过程在服务路由过程中自动完成，用户无需进行额外的熔断状态判断等操作。
执行服务路由：

```
import com.tencent.polaris.factory.api.DiscoveryAPIFactory;


public static void main(String[] args) throws Exception {
    ConsumerAPI consumerAPI = DiscoveryAPIFactory.createConsumerAPI();
    // 执行服务路由，筛选出单个实例，在这个过程中，会自动剔除熔断的实例
    GetOneInstanceRequest getOneInstanceRequest = new GetOneInstanceRequest();
    getOneInstanceRequest.setNamespace("default");
    getOneInstanceRequest.setService("testService1");
    InstancesResponse oneInstance = consumerAPI.getOneInstance(getOneInstanceRequest);
    Instance targetInstance = oneInstance.getInstances()[0];
    // 执行服务调用
    // invoke rpc call with targetInstance
    // 上报调用结果，调用结果用于进行熔断判断
    ServiceCallResult result = new ServiceCallResult();
    result.setNamespace(namespace);
    result.setService(service);
    result.setHost(targetInstance.getHost());
    result.setPort(targetInstance.getPort());
    // 设置返回码
    result.setRetCode(code);
    // 设置调用时延
    result.setDelay(delay);
    // 使用updateServiceCallResult上报调用结果
    consumerAPI.updateServiceCallResult(result);
}
```

# 配置管理

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 polaris-sdk 的方式接入 Polaris（北极星），并实现配置分组管理功能。

## 前提条件

在开发前，请确保您已经下载并安装了 Java 和 Maven。

## 操作步骤

### 步骤1：引入北极星依赖

#### 1. 引入 polaris sdk 依赖

修改应用根目录下的 `pom.xml`，为 `polaris-java` 添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.polaris</groupId>
            <artifactId>polaris-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **说明：**
> polaris-sdk 版本信息请参见：版本信息 。

#### 2. 引入 polaris starter

```
<dependencies>
    <dependency>
        <groupId>com.tencent.polaris</groupId>
        <artifactId>polaris-all</artifactId>
    </dependency>
</dependencies>
```

### 步骤2：添加北极星配置文件 polaris.yml

1. 在项目的 main/resources 目录下创建 polaris.yml 文件用于初始化 polaris-java SDK。
2. 在 polaris.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址详细参见：引擎管理 > 客户端访问地址 。

```
global:
  serverConnectors:
    - id: polaris
      protocol: grpc
      addresses:
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        - 127.0.0.1:8091
  #描述：监控及日志数据上报相关配置
  statReporter:
```

```
    #描述：是否启用上报
    enable: true
    plugin:
      prometheus:
        type: push
        # 描述：设置 pushgateway 的地址，仅 type == push 时生效
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        address: 127.0.0.1:9091
        #描述：设置metric数据推送到pushgateway的执行周期
        #范围：[1s:...]，默认值：10s
        pushInterval: 10s
```

更多 polaris.yml 配置信息，请参见 default-config.yml 。

## 步骤3：应用开发

### 1.获取配置

**（1）初始化 ConfigFileService SDK 实例**

```
import com.tencent.polaris.configuration.factory.ConfigFileServiceFactory;

public static void main(String[] args) throws Exception {
    ConfigFileService configFileService = ConfigFileServiceFactory.createConfigFileService();
}
```

**（2）获取配置文件**

```
// 获取特定远程的配置文件
ConfigFile getConfigFile(String namespace, String fileGroup, String fileName);

// 获取特定远程的配置文件
ConfigFile getConfigFile(ConfigFileMetadata configFileMetadata);
```

**（3）监听配置文件**

```
//获取配置文件
ConfigFile configFile = configFileService.getConfigFile(namespace, fileGroup, fileName);
//添加变更监听器
configFile.addChangeListener(new ConfigFileChangeListener() {
    @Override
    public void onChange(ConfigFileChangeEvent event) {
    }
});
```

**（4）监听配置分组下的已发布文件列表变化**

获取到目标配置分组后，调用配置分组的 addChangeListener 方法监听改配置分组下已发布配置文件列表的变化。

> **说明：**
> 需要更新 polaris-java 的版本至 v1.14.0 及以上版本。

```
ConfigFileGroup configFileGroup = configFileService.getConfigFileGroup(namespace, fileGroup);
if (configFileGroup != null) {
    configFileGroup.addChangeListener(new ConfigFileGroupChangeListener() {
        @Override
        public void onChange(ConfigFileGroupChangedEvent event) {
            Utils.print(event.toString());
```

```
        }
    });
}
```

**ConfigFileGroupChangedEvent 数据结构**

```
public class ConfigFileGroupChangedEvent {
    // 配置分组自身元数据信息
    private final ConfigFileGroupMetadata configFileGroupMetadata;
    // 当前配置分组下的最新已发布的配置文件列表
    private final List<ConfigFileMetadata> configFileMetadataList;

    public ConfigFileGroupChangedEvent(ConfigFileGroupMetadata configFileGroupMetadata,
List<ConfigFileMetadata> configFileMetadataList) {
        this.configFileGroupMetadata = configFileGroupMetadata;
        this.configFileMetadataList = configFileMetadataList;
    }

    public ConfigFileGroupMetadata getConfigFileGroupMetadata() {
        return configFileGroupMetadata;
    }

    public List<ConfigFileMetadata> getConfigFileMetadataList() {
        return configFileMetadataList;
    }

    @Override
    public String toString() {
        return "ConfigFileGroupChangedEvent{" +
                "configFileGroupMetadata=" + configFileGroupMetadata +
                ", configFileMetadataList=" + configFileMetadataList +
                '}';
    }
}
```

## 2.操作配置

**（1）初始化 ConfigFilePublishService SDK 实例。**

```
import com.tencent.polaris.configuration.factory.ConfigFileServicePublishFactory;

public static void main(String[] args) throws Exception {
    ConfigFilePublishService configFilePublishService =
ConfigFileServicePublishFactory.createConfigFilePublishService();
}
```

**（2）操作配置文件：创建、修改、发布配置文件。**

```
// 创建配置文件
void createConfigFile(String namespace, String fileGroup, String fileName, String content);

// 创建配置文件
void createConfigFile(ConfigFileMetadata configFileMetadata, String content);

// 修改配置文件
void updateConfigFile(String namespace, String fileGroup, String fileName, String content);

// 修改配置文件
```

```
void updateConfigFile(ConfigFileMetadata configFileMetadata, String content);

// 发布配置文件
void releaseConfigFile(String namespace, String fileGroup, String fileName);

// 发布配置文件
void releaseConfigFile(ConfigFileMetadata configFileMetadata);
```

# 可观测性

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文介绍在本地开发 Java 应用，通过 polaris-sdk 的方式接入 Polaris（北极星），并实现可观测性功能。

## 前提条件

在开发前，请确保您已经下载并安装了 Java 和 Maven。

## 操作步骤

### 步骤1：引入北极星依赖

#### 1. 引入 polaris sdk 依赖

修改应用根目录下的 `pom.xml`，为 `polaris-java` 添加 `dependencyManagement`：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.tencent.polaris</groupId>
            <artifactId>polaris-dependencies</artifactId>
            <version>${version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

> **说明：**
> polaris-sdk 版本信息请参见：版本信息 。

#### 2. 引入 polaris starter

```
<dependencies>
    <dependency>
        <groupId>com.tencent.polaris</groupId>
        <artifactId>polaris-all</artifactId>
    </dependency>
</dependencies>
```

### 步骤2：添加北极星配置文件 polaris.yml

1. 在项目的 main/resources 目录下创建 polaris.yml 文件用于初始化 polaris-java SDK。
2. 在 polaris.yml 文件中配置应用名、polaris（北极星）服务端地址等信息。服务端地址查看详细参见：引擎管理 > 客户端访问地址 。

```
global:
  serverConnectors:
    - id: polaris
      protocol: grpc
      addresses:
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        - 127.0.0.1:8091
  #描述：监控及日志数据上报相关配置
  statReporter:
```

```
    #描述：是否启用上报
    enable: true
    plugin:
      prometheus:
        type: push
        # 描述：设置 pushgateway 的地址，仅 type == push 时生效
        # 地址需要替换成您创建的北极星引擎的客户端访问地址。
        address: 127.0.0.1:9091
        #描述：设置metric数据推送到pushgateway的执行周期
        #范围：[1s:...]，默认值：10s
        pushInterval: 10s
```

更多 polaris.yml 配置信息，请参见 default-config.yml 。

### 步骤3：应用开发，上报监控指标

**发起调用，并上报请求调用结果。**

1. 初始化 ConsumerAPI SDK 实例：

```
import com.tencent.polaris.factory.api.DiscoveryAPIFactory;

public static void main(String[] args) throws Exception {
    ConsumerAPI consumerAPI = DiscoveryAPIFactory.createConsumerAPI();
}
```

2. 设置服务调用结果信息：

```
public enum RetStatus {
    // 服务调用成功
    RetSuccess,
    // 服务调用失败
    RetFail,
    // 服务调用超时
    RetTimeout,
}

ServiceCallResult result = new ServiceCallResult();
// 设置被调服务所在命名空间
result.setNamespace(String namespace);
// 设置被调服务的服务信息
result.setService(String service);
// 设置被调实例
result.setInstance(Instance instance);
// 设置本次请求的响应码
result.setRetCode(String code);
// 设置本次请求的耗时
result.setDelay(String delay);
// 设置本次请求的结果状态
result.setRetStatus(RetStatus status);
// 设置本次请求的请求标签，格式为 key=value;key=value;
result.setLabels(String labels);
// 设置本次请求调用的方法
result.setMethod(String method);
```

3. 上报请求调用结果：

   您在根据请求调用情况对 ServiceCallResult 结构体完成初始化后，只需要调用 `ConsumerAPI.updateServiceCallResult` 方法即可完成请求调用结果上报。SDK 内部会根据上报的调用结果信息，将其转换为相应的流量调用指标数据，上报至 prometheus。

```
consumerAPI.updateServiceCallResult(ServiceCallResult)
```

# 二次寻址

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文针对用户接入微服务引擎托管的Polaris（北极星）实例希望使用北极星客户端的二次寻址能力。

**警告：**

- 如果您使用的是腾讯自研上云（OA 版）北极星 SDK，由于 OA 版 SDK 默认开启了二次寻址机制（且无法关闭），请参见本文档接入。
- 当前仅针对腾讯自研上云（OA 版）北极星 SDK 的注册发现能力做了兼容性验证，**其他治理能力无兼容性保证**。

## 接入机制说明

### 北极星 SDK 二次寻址

北极星 SDK 的二次寻址机制依赖 **polaris.discover** 以及 **polaris.healthcheck** 这2个服务。因此需要在北极星引擎实例创建出来后，需在控制台手动创建 **polaris.discover** 以及 **polaris.healthcheck** 这两个服务。

### 为何需要用户手动创建

用户接入方式存在多 VPC、公网接入两类场景。不同的接入侧用户需要看到不同的服务端 IP，无法自动创建相关北极星系统服务。

## 前提条件

已创建 Polaris（北极星）实例，请参见 创建 Polaris（北极星）。

## 操作步骤

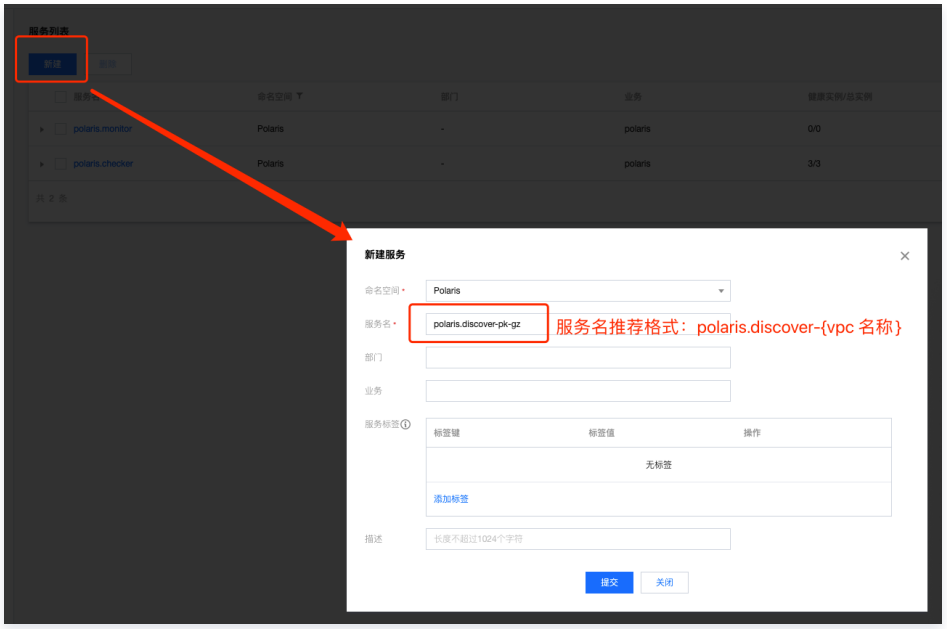
1. 登录 TSF 控制台。
2. 在左侧导航栏选择**Polaris（北极星）**，进入实例引擎列表页。
3. 单击目标引擎的“ID”，进入基本信息页面。
4. 根据实例信息的控制台。

### 引擎实例未开启客户端公网访问

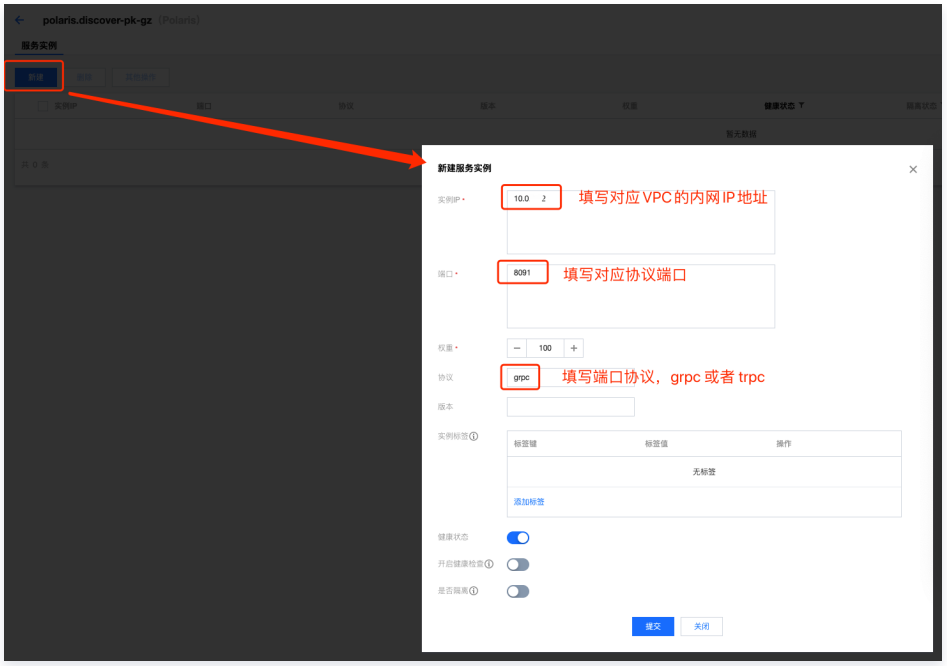
**创建 polaris.discover 以及 polaris.healthcheck 服务**

1. 在实例的基本信息页面中，记住访问地址中的每个 VPC 内网 IP 地址以及访问端口。

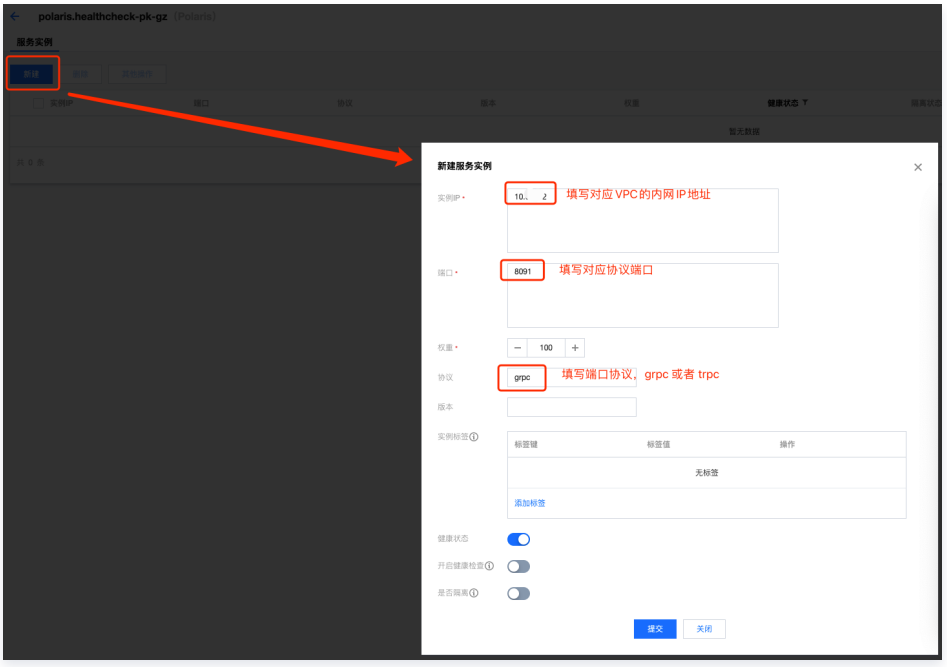

2. 针对每一个内网 VPC 的 IP 地址，在命名空间 Polaris 下创建服务 polaris.discover-{vpc 名称} 以及 polaris.healthcheck-{vpc 名称}。




3. 单击服务 polaris.discover-{vpc 名称}，进入到服务的实例页面，根据对应上述第一步所记录的 VPC 内网 IP 地址以及端口协议 grpc、trpc 分别创建实例。




4. 单击服务 **polaris.healthcheck-{vpc 名称}**，进入到服务的实例页面，根据对应上述第一步所记录的 VPC 内网 IP 地址以及端口协议 grpc、trpc 分别创建实例。




5. 按照上述步骤的操作，最终服务 **polaris.discover-{vpc 名称}** 的实例列表如下：

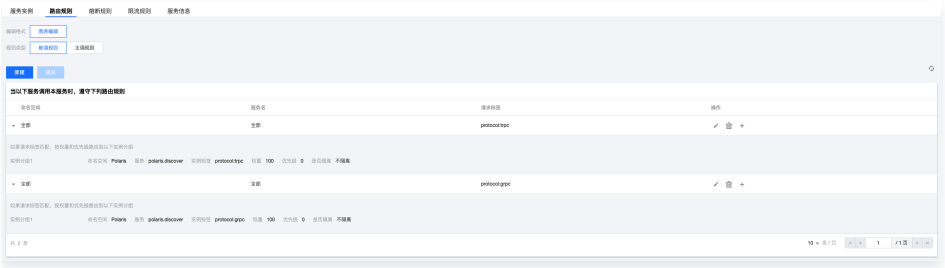
6. 按照上述步骤的操作，最终服务 **polaris.healthcheck-{vpc 名称}** 的实例列表如下：

> ⚠ **注意：**
> 如果只是单 VPC 接入的场景，则无需参考 **polaris.discover-{vpc 名称}** 这种命名规则，直接使用 **polaris.discover** 即可，**polaris.healthcheck-{vpc 名称}** 也直接使用 **polaris.healthcheck** 即可。

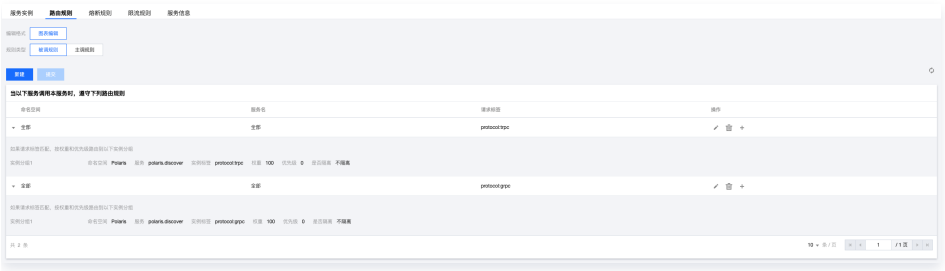
### 创建路由规则

#### 创建路由规则示例

**polaris.discover-{vpc 名称}**



**polaris.healthcheck-{vpc 名称}**



### polaris-java 接入相关配置文件说明

- polaris.yaml 中按照下面示例进行配置二次寻址：

```
global:
  serverConnector:
    addresses:
      - '{对应 vpc 的内网接入地址}:8091'
  system:
    discoverCluster:
      namespace: Polaris
      service: 'polaris.discover-{vpc 名称}'
      sameAsBuiltin: false
    healthCheckCluster:
      namespace: Polaris
      service: 'polaris.healthcheck-{vpc 名称}'
      sameAsBuiltin: false
```

### spring-cloud-tencent 接入相关配置文件说明

- 修改 bootstrap.yaml 或者 bootstrap.properties 文件：

```
spring:
  cloud:
    polaris:
      address: '{对应 vpc 的内网接入地址}:8091'
```

- 在 **resources** 下创建 **polaris.yml** 文件。

- 按照下面示例进行配置二次寻址：

```
global:
  system:
    #服务发现集群
    discoverCluster:
      namespace: Polaris
      service: 'polaris.discover-{vpc 名称}'
      sameAsBuiltin: false
    #健康检查集群
    healthCheckCluster:
      namespace: Polaris
      service: 'polaris.healthcheck-{vpc 名称}'
      sameAsBuiltin: false
```

**trpc-java 接入相关配置文件说明**

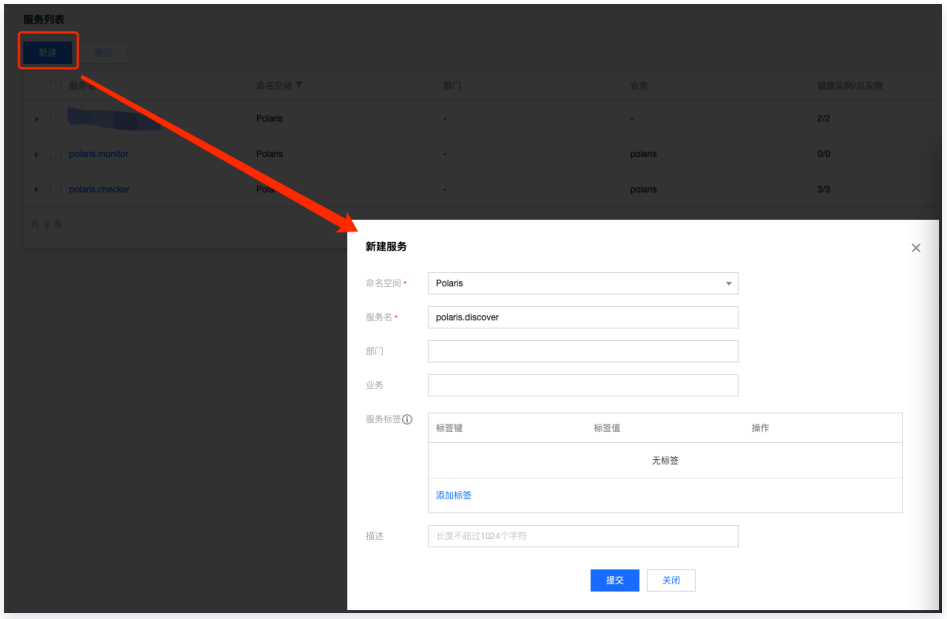
```
plugins:
  registry:
    polaris:
      heartbeat_interval: 3000
      protocol: grpc
      address_list: '{对应 vpc 的内网接入地址}:8091'
      global:
        system:
          discoverCluster:
            namespace: Polaris
            service: 'polaris.discover-{vpc 名称}'
            sameAsBuiltin: false
          healthCheckCluster:
            namespace: Polaris
            service: 'polaris.healthcheck-{vpc 名称}'
            sameAsBuiltin: false
  selector:
    polaris:
      protocol: grpc
      address_list: '{对应 vpc 的内网接入地址}:8091'
      global:
        system:
          discoverCluster:
            namespace: Polaris
            service: 'polaris.discover-{vpc 名称}'
            sameAsBuiltin: false
          healthCheckCluster:
            namespace: Polaris
            service: 'polaris.healthcheck-{vpc 名称}'
            sameAsBuiltin: false
```

## 引擎实例开启客户端公网访问

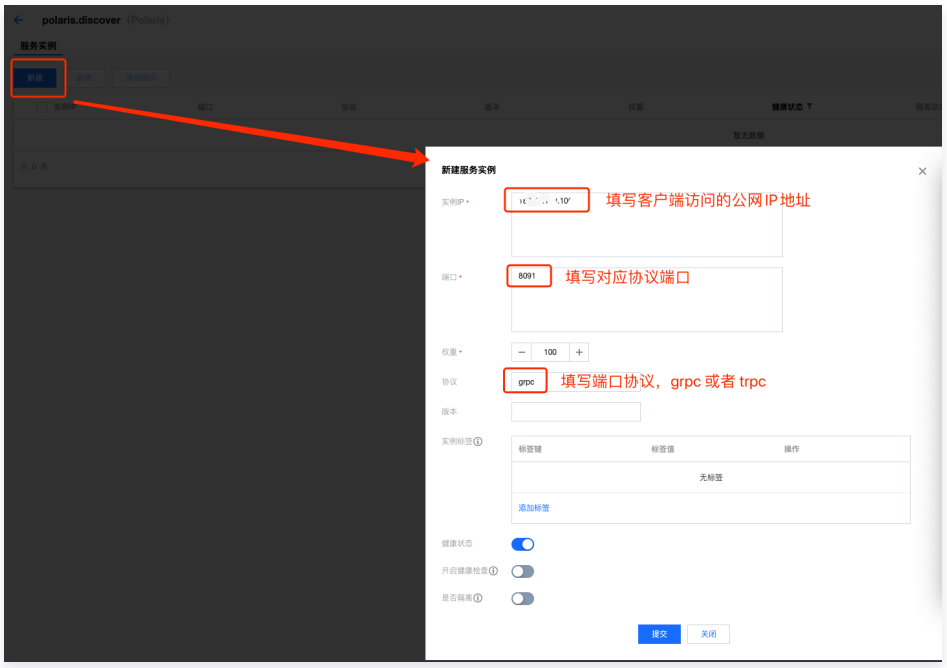
**创建 polaris.discover 以及 polaris.healthcheck 服务**

1. 在实例的基本信息页面中，记住访问地址中的公网 IP 地址以及访问端口。

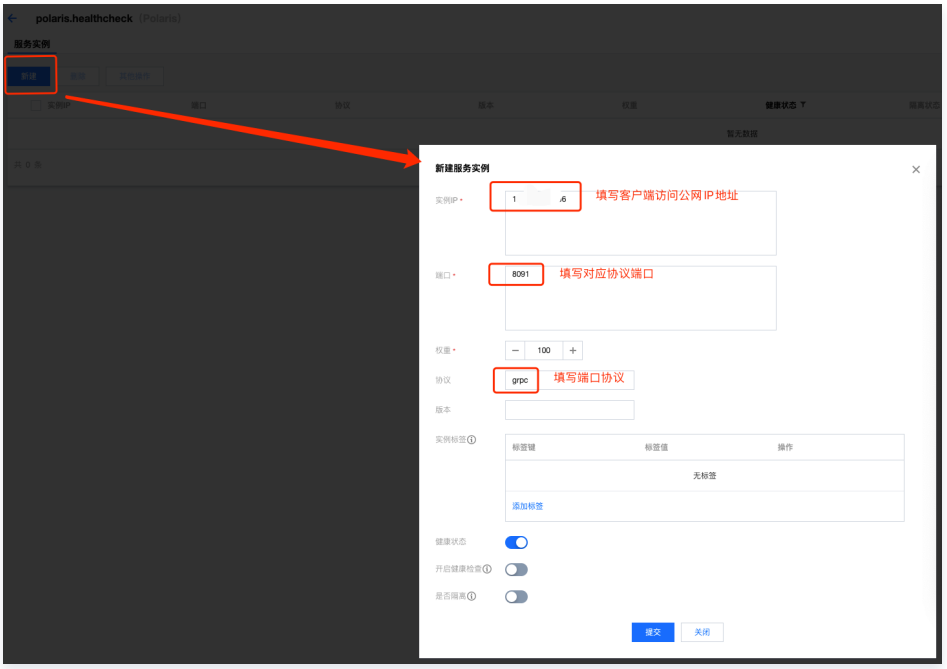

2. 在命名空间 Polaris下创建服务 polaris.discover 以及 polaris.healthcheck。



3. 单击服务 **polaris.discover**，进入到服务的实例页面，根据客户端公网访问 IP 地址以及端口协议 **grpc**、**trpc** 分别创建实例。




4. 单击服务 **polaris.healthcheck**，进入到服务的实例页面，根据客户端公网访问 IP 地址以及端口协议 **grpc**、**trpc** 分别创建实例。

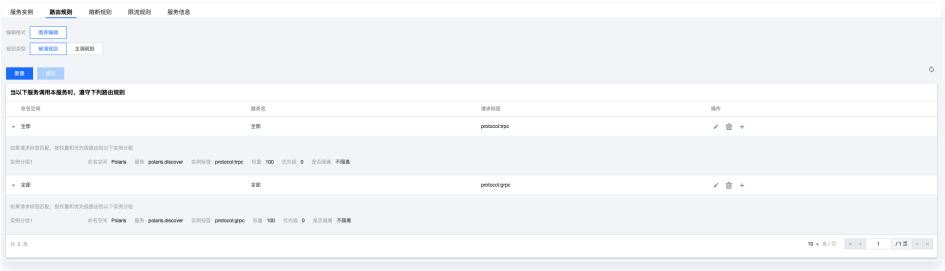



5. 按照上述步骤的操作，最终服务 **polaris.discover** 的实例列表如下：

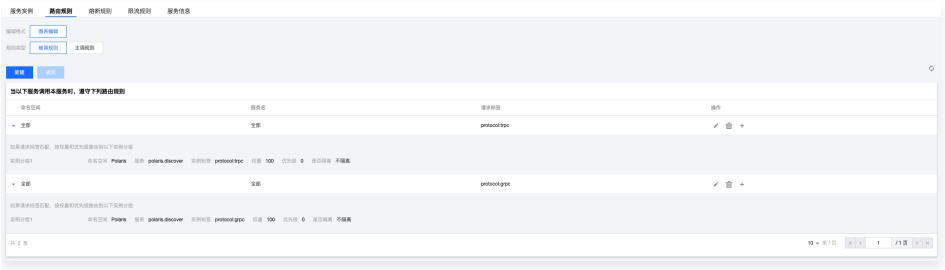
6. 按照上述步骤的操作，最终服务 **polaris.healthcheck** 的实例列表如下：

**创建路由规则**

**polaris.discover**



**polaris.healthcheck**



**polaris-java 接入相关配置文件说明**

- polaris.yaml 中按照下面示例进行配置二次寻址：

```
global:
  serverConnector:
    addresses:
      - '{对应公网接入地址}:8091'
  system:
    discoverCluster:
      namespace: Polaris
      service: polaris.discover
      sameAsBuiltin: false
    healthCheckCluster:
      namespace: Polaris
      service: polaris.healthcheck
      sameAsBuiltin: false
```

**spring-cloud-tencent 接入相关配置文件说明**

- 修改 bootstrap.yaml 或者 bootstrap.properties 文件。

```
spring:
  cloud:
    polaris:
      address: '{对应公网接入地址}:8091'
```

- 在 resources 下创建 polaris.yml 文件。
- 按照下面示例进行配置二次寻址：

```
global:
  system:
    discoverCluster:
      namespace: Polaris
      service: polaris.discover
      sameAsBuiltin: false
```

```
    healthCheckCluster:
      namespace: Polaris
      service: polaris.healthcheck
      sameAsBuiltin: false
```

**trpc-java 接入相关配置文件说明**

```
plugins:
  registry:
    polaris:
      heartbeat_interval: 3000
      protocol: grpc
      address_list: '{对应公网接入地址}:8091'
      global:
        system:
          discoverCluster:
            namespace: Polaris
            service: polaris.discover
            sameAsBuiltin: false
          healthCheckCluster:
            namespace: Polaris
            service: polaris.healthcheck
            sameAsBuiltin: false
  selector:
    polaris:
      protocol: grpc
      address_list: '{对应公网接入地址}:8091'
      global:
        system:
          discoverCluster:
            namespace: Polaris
            service: polaris.discover
            sameAsBuiltin: false
          healthCheckCluster:
            namespace: Polaris
            service: polaris.healthcheck
            sameAsBuiltin: false
```

# Go 应用开发
# Polaris-Go 接入

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文通过一个 demo 进行 Golang 应用接入 Polaris（北极星）的全流程操作演示，帮助您快速了解如何使用北极星网格。

## 前提条件

- 已创建 Polaris（北极星），请参见 创建 Polaris（北极星）。
- 下载 Github 的 demo 源码 到本地并解压。
- 本地编译构建打包机器环境已安装了 golang 环境，并且能够访问 Github。
- 根据您自身的业务，已准备好业务部署的资源，虚拟机部署和容器化部署选择其中一种方式即可。
  - 虚拟机部署已创建 CVM 虚拟机，请参见 创建 CVM 虚拟机。
  - 容器化部署已创建 TKE 容器集群，请参见 创建 TKE 集群。

## 操作步骤

1. 登录 TSF 控制台。
2. 在左侧导航栏选择Polaris（北极星），进入北极星实例列表页。
3. 单击目标引擎的“ID”，进入基本信息页面。
4. 查看访问地址，Golang 应用访问使用 gRPC 端口（8091）：


5. 修改 demo 中的注册中心地址
   5.1 在下载到本地的 demo 源码 目录下，分别找到 `quickstart/consumer/polaris.yaml` 以及 `quickstart/provider/polaris.yaml` 文件。
   5.2 添加微服务引擎北极星网格地址到项目配置文件中（这里以 `quickstart/consumer/polaris.yaml` 为例）。

```
global:
  serverConnector:
    addresses:
      - '10.0.4.6:8091'
```

6. 将源码编译成可执行程序。
   6.1 分别在 consumer 和 provider 这2个目录下，打开 cmd 命令，执行以下命令，对项目进行编译：
   - 编译 consumer：`CGO_ENABLED=0 go build -ldflags "-s -w" -o consumer`
   - 编译 provider：`CGO_ENABLED=0 go build -ldflags "-s -w" -o provider`

   6.2 编译成功后，生成如下表所示的2个二进制包。

| 软件包所在目录 | 软件包名称 | 说明 |
| --- | --- | --- |
| \examples\quickstart\provider | provider | 服务生产者 |
| \examples\quickstart\consumer | consumer | 服务消费者 |

6.3 分别将 consumer 以及 provider 的二进制上传到不同的 CVM 实例中，这里假定上传的路径均为/data/polaris/golang_examples。

7. 部署 provider 和 consumer 微服务应用，根据您业务实际选择**虚拟机部署**、**容器化部署**中的一种部署方式即可。

7.1 **虚拟机部署**

- 上传二进制文件以及配置文件（polaris.yaml）至 CVM 实例。
- 执行启动命令进行启动：

```
./[二进制文件名称]
```

7.2 **容器化部署**

- 编写 dockerfile 生成镜像，参考：

```
FROM golang:alpine
WORKDIR /root
ADD . /root
ENTRYPOINT ./[二进制名称] [启动参数命令]
```

- 通过 TKE 部署并运行镜像。

8. 确认部署结果

8.1 进入前面提到的微北极星网格实例页面。

8.2 选择**服务管理 > 服务列表**，查看微服务 EchoServerGolang 的实例数量。

- 若实例数量值不为0，则表示已经成功接入微服务引擎。
- 若实例数量为0，或者找不到 EchoServerGolang 服务名，则表示微服务应用接入微服务引擎失败。




8.3 调用 consumer 的 HTTP 接口。执行 http 调用，其中 `${app.port}` 替换为 consumer 的监听端口（默认为18080），`${add.address}` 则替换为 consumer 暴露的地址。

```
curl -L -X GET 'http://${add.address}:${app.port}/echo'
预期返回值：Hello, I'm EchoServerGolang Provider
```

# gRPC-Go 接入

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文通过一个 demo 进行 gRPC-Go 应用接入Polaris（北极星）的全流程操作演示，帮助您快速了解如何使用北极星网格。

## 前提条件

- 已创建 Polaris（北极星），请参见 创建 Polaris（北极星）。
- 下载 Github 的 demo 源码 到本地并解压。
- 本地编译构建打包机器环境已安装了 Go，并且能够使用 Go mod 拉取依赖。
- 根据您自身的业务，已准备好业务部署的资源， `虚拟机部署` 、 `容器化部署` 和 `TEM 部署` 选择其中一种方式即可。
  - **虚拟机部署**已创建 CVM 虚拟机，请参见 创建 CVM 虚拟机。
  - **容器化部署**已创建 TKE 容器集群，请参见 创建 TKE 集群。
  - **TEM部署**已创建 TEM 环境，请参见 创建 TEM 环境。

## 操作步骤

1. 登录 TSF 控制台。
2. 在**北极星网格**下的 **polarismesh** 页面，单击页面左上方下拉列表，选择目标地域。
3. 单击目标引擎的“ID”，进入基本信息页面。
4. 查看访问地址，gRPC-Go 应用访问使用 gRPC 端口（8091）：


5. 修改 demo 中的注册中心地址。
   5.1 在下载到本地的 demo 源码目录 下，分别找到 `\examples\quickstart\provider\polaris.yaml` 和 `\examples\quickstart\consumer\polaris.yaml` 两个文件。
   5.2 添加微服务引擎北极星网格地址到项目配置文件中（以 `\examples\quickstart\provider\polaris.yaml` 为例）。

```
global:
  serverConnector:
    addresses:
      - '10.0.4.6:8091'
```

6. 将源码编译成可执行程序。
   6.1 分别在 `consumer` 和 `provider` 这2个目录下，打开 cmd 命令，执行以下命令，对项目进行编译：
   - 编译 consumer： `CGO_ENABLED=0 go build -ldflags "-s -w" -o consumer`
   - 编译 provider： `CGO_ENABLED=0 go build -ldflags "-s -w" -o provider`

   6.2 编译成功后，生成如下表所示的2个二进制包。

| 软件包所在目录 | 软件包名称 | 说明 |
| --- | --- | --- |
| \examples\quickstart\provider | provider | 服务生产者 |
| \examples\quickstart\consumer | consumer | 服务消费者 |

7. 部署 provider 和 consumer 微服务应用，根据您业务实际选择**虚拟机部署**、**容器化部署**以及 **TEM 部署**中的一种部署方式即可。

**7.1 虚拟机部署**

- 上传 Jar 包至 CVM 实例。
- 执行启动命令进行启动：

```
nohup [二进制名称] &
```

**7.2 容器化部署**

- 编写 dockerfile 生成镜像，参考：

```
FROM golang:alpine
WORKDIR /root
ADD . /root
ENTRYPOINT ./[二进制名称]
```

- 通过 TKE 部署并运行镜像。

**7.3 TEM 部署**

- 选择 TEM 环境，注意所选择的环境，其依赖的 VPC，必须和上面已经创建的北极星网格实例所依赖的 VPC 一致：

○ 在已选择的环境中，新建TEM应用，相关参数填写参考：

- 部署应用，相关参数填写请参考（端口号映射，consumer 默认端口号为16011，provider默认端口号为16010）：

- 查看访问路径，consumer 应用部署完后，可以在**基本信息 > 访问配置**中查看访问地址，如需公网访问，可以**编辑并更新**开启公网访问：


8. 确认部署结果。

8.1 进入前面提到的微北极星网格实例页面。

- 选择**服务管理 > 服务列表**，查看微服务 EchoServerGRPC（provider）的实例数量：
- 若实例数量值不为0，则表示已经成功接入微服务引擎。
- 若实例数量为0，或者找不到 EchoServerGRPC 服务名，则表示微服务应用接入微服务引擎失败。


8.2 调用 consumer 的 HTTP 接口。执行 http 调用，其中 `${app.port}` 替换为 consumer 的监听端口（默认为16011），`${add.address}` 则替换为 consumer 暴露的地址。

```
curl -L -X GET 'http://${add.address}:${app.port}/echo?value=hello_world'
```

**预期返回值：** `echo: hello_world`

# 二次寻址使用指南

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文针对用户接入 Polaris（北极星）希望使用北极星客户端的二次寻址能力。

**警告：**

- 如果您使用的是腾讯自研上云（OA 版）北极星 SDK，由于 OA 版 SDK 默认开启了二次寻址机制（且无法关闭），请参见本文档接入。
- 当前仅针对腾讯自研上云（OA 版）北极星 SDK 的注册发现能力做了兼容性验证，**其他治理能力无兼容性保证**。

## 接入机制说明

### 北极星 SDK 二次寻址

北极星 SDK 的二次寻址机制依赖 **polaris.discover** 以及 **polaris.healthcheck** 这2个服务。因此需要在北极星引擎实例创建出来后，需在控制台手动创建 **polaris.discover** 以及 **polaris.healthcheck** 这两个服务。

### 为何需要用户手动创建

用户接入方式存在多 VPC、公网接入两类场景。不同的接入侧用户需要看到不同的服务端IP，无法自动创建相关北极星系统服务。

## 前提条件

已创建 Polaris（北极星），请参见 创建 Polaris（北极星）。

## 操作步骤

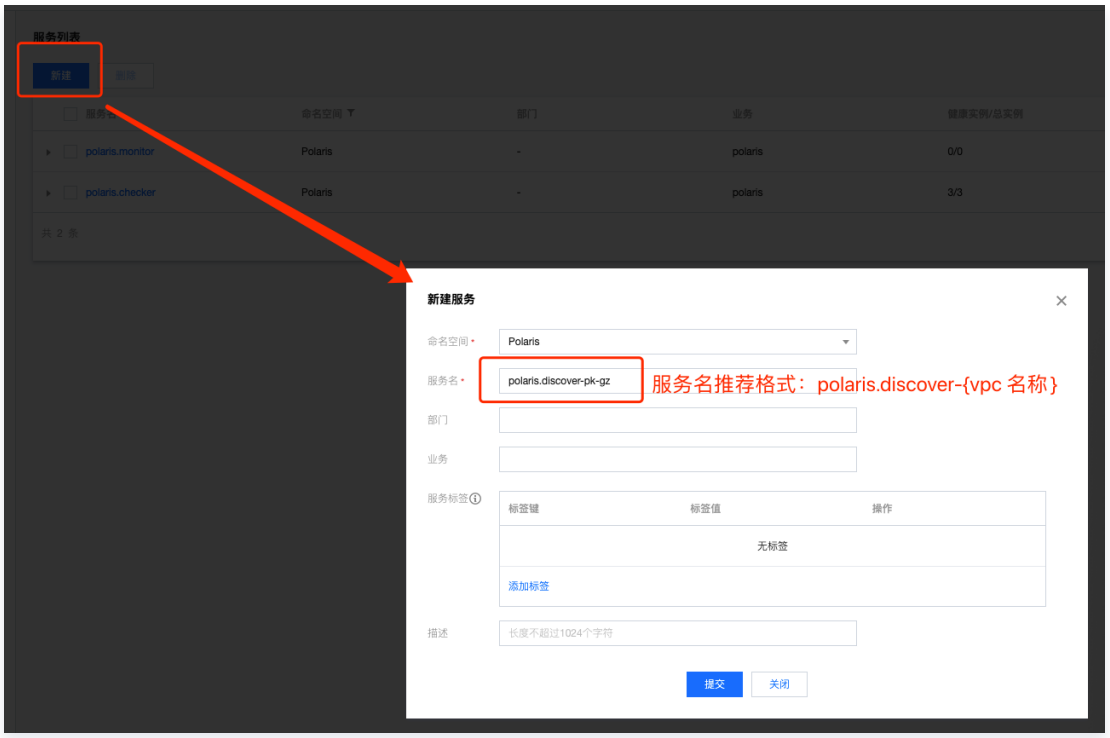
1. 登录 TSF 控制台。
2. 在左侧导航栏选择**Polaris（北极星）**，进入北极星实例列表页。
3. 单击目标引擎的“ID”，进入基本信息页面。
4. 根据实例信息的控制台

### 引擎实例未开启客户端公网访问

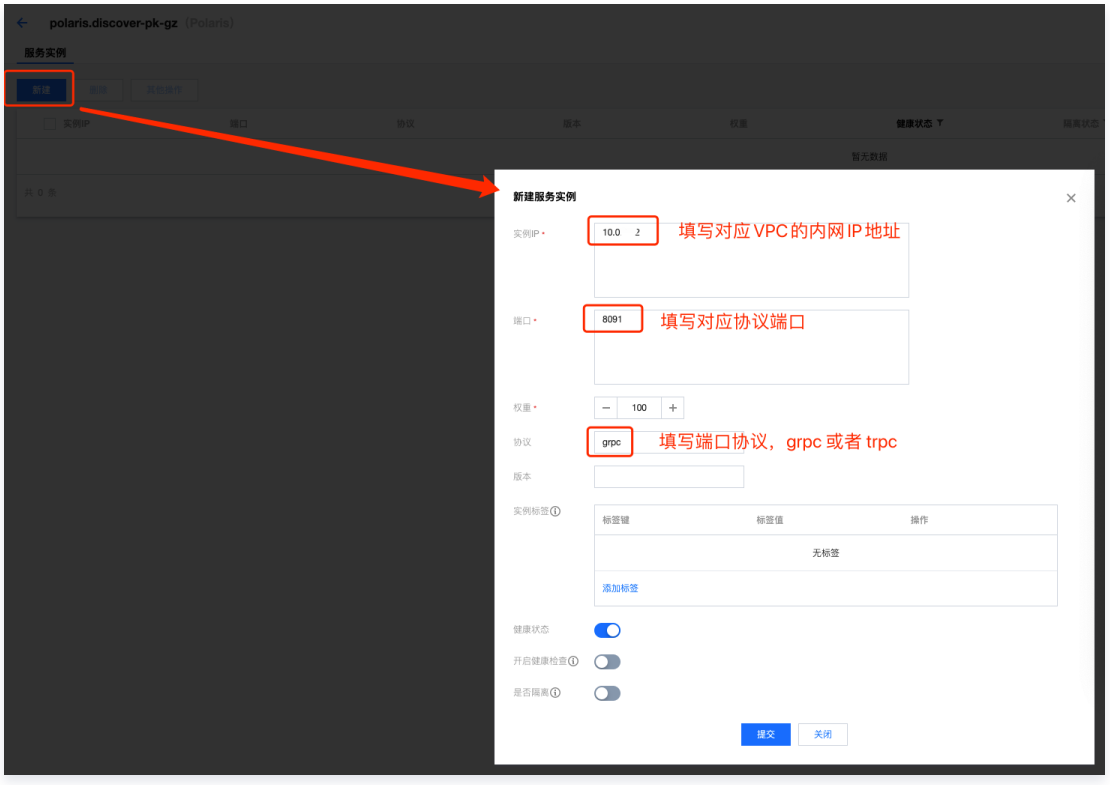
#### 创建 polaris.discover 以及 polaris.healthcheck 服务

1. 在实例的基本信息页面中，记住访问地址中的每个 VPC 内网 IP 地址以及访问端口。

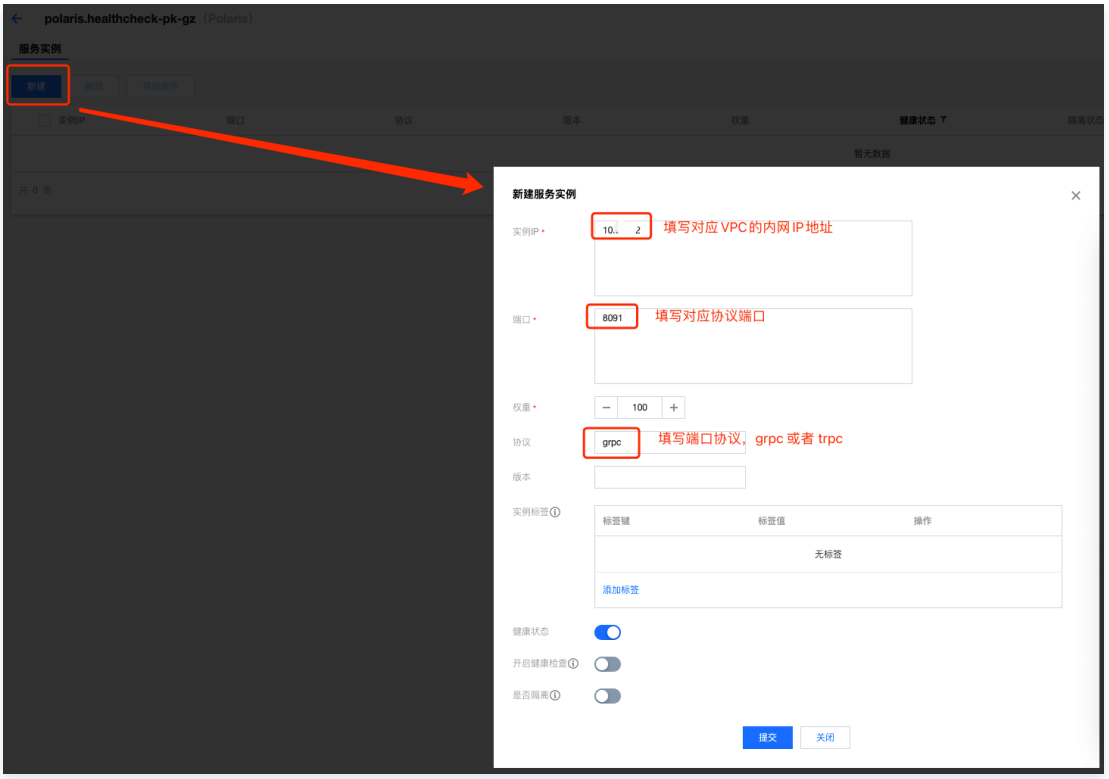
2. 针对每一个内网 VPC 的 IP 地址，在命名空间 Polaris 下创建服务 polaris.discover-{vpc 名称} 以及 polaris.healthcheck-{vpc 名称}。




3. 单击服务 polaris.discover-{vpc 名称}，进入到服务的实例页面，根据对应上述第一步所记录的 VPC 内网 IP 地址以及端口协议 grpc、trpc 分别创建实例。

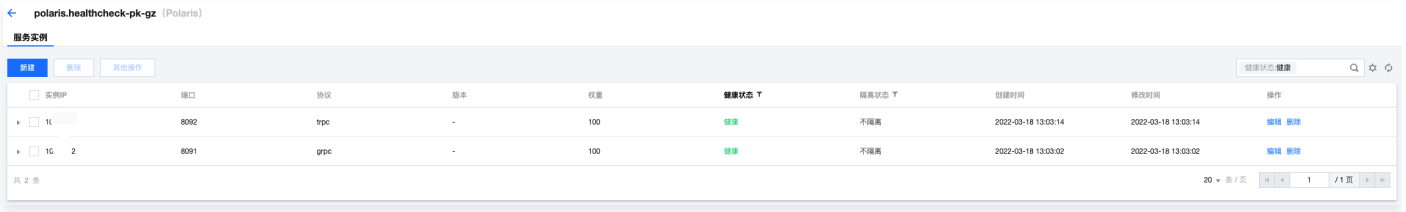



4. 单击服务 polaris.healthcheck-{vpc 名称}，进入到服务的实例页面，根据对应上述第一步所记录的 VPC 内网 IP 地址以及端口协议 grpc、trpc 分别创建实例。




5. 按照上述步骤的操作，最终服务 **polaris.discover-{vpc 名称}** 的实例列表如下：


6. 按照上述步骤的操作，最终服务 **polaris.healthcheck-{vpc 名称}**的实例列表如下：

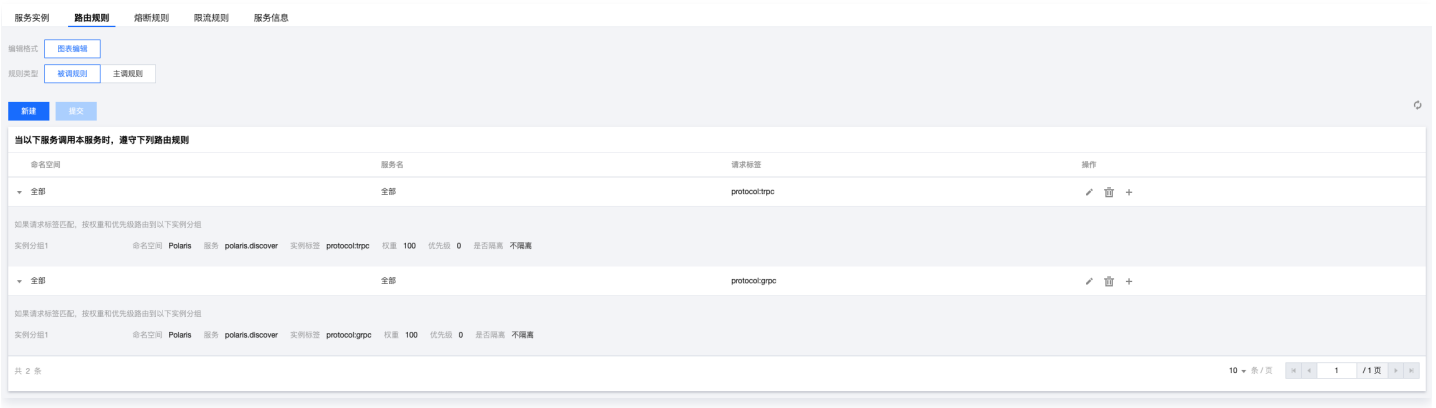



> **注意：**
> 如果只是单 VPC 接入的场景，则无需参考 **polaris.discover-{vpc 名称}** 这种命名规则，直接使用 **polaris.discover** 即可，**polaris.healthcheck-{vpc 名称}** 也直接使用 **polaris.healthcheck** 即可。

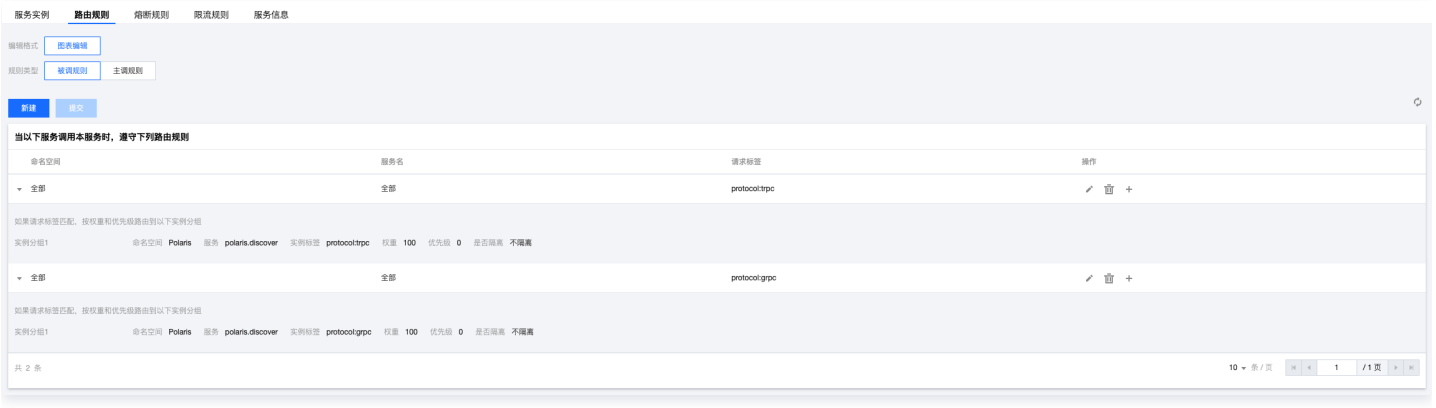
**创建路由规则**

**创建路由规则示例**

**polaris.discover-{vpc 名称}**




**polaris.healthcheck-{vpc 名称}**




**polaris-go 接入相关配置文件说明**

- **polaris.yaml中按照下面示例进行配置二次寻址：**

```
global:
  serverConnector:
    addresses:
      - '{对应 vpc 的内网接入地址}:8091'
  system:
    discoverCluster:
      namespace: Polaris
      service: 'polaris.discover-{vpc 名称}'
    healthCheckCluster:
      namespace: Polaris
      service: 'polaris.healthcheck-{vpc 名称}'
```

**trpc-go 接入相关配置文件说明**

```
plugins:
  registry:
    polaris:
      heartbeat_interval: 3000
      protocol: grpc
      address_list: '{对应 vpc 的内网接入地址}:8091 '
      cluster_service:
        discover: 'polaris.discover-{vpc 名称}'
        health_check: 'polaris.healthcheck-{vpc 名称}'
  selector:
    polaris:
      address_list: '{对应 vpc 的内网接入地址}:8091'
```

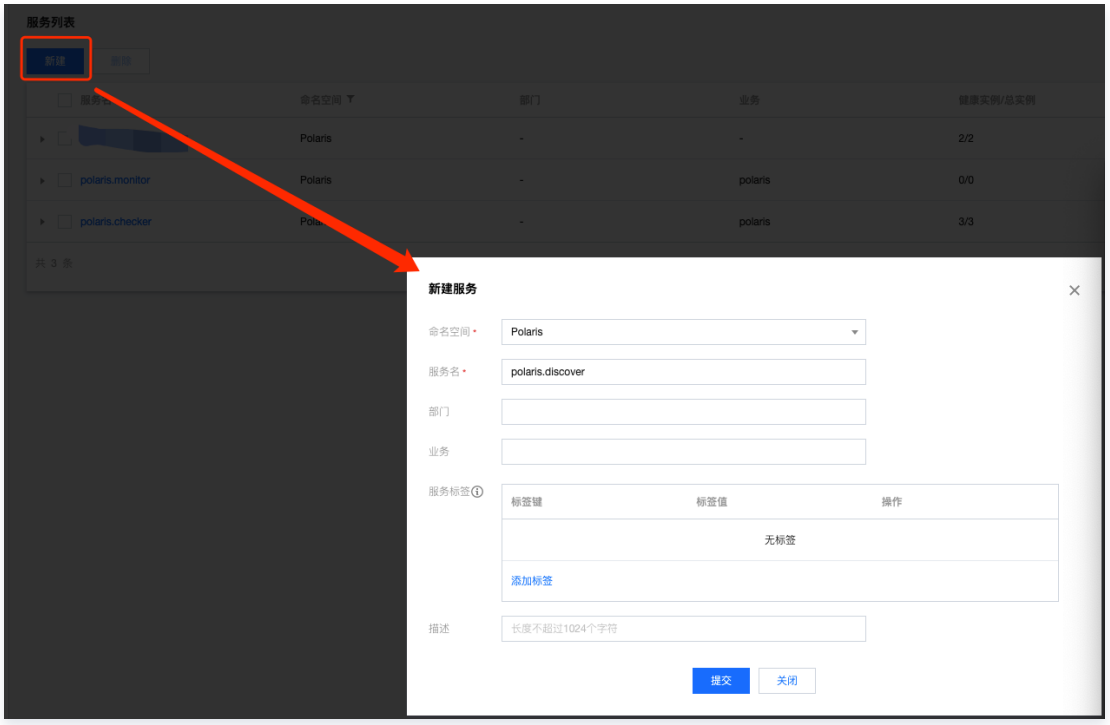
```
    protocol: grpc
    cluster_service:
      discover: 'polaris.discover-{vpc 名称}'
      health_check: 'polaris.healthcheck-{vpc 名称}'
```

## 引擎实例开启客户端公网访问

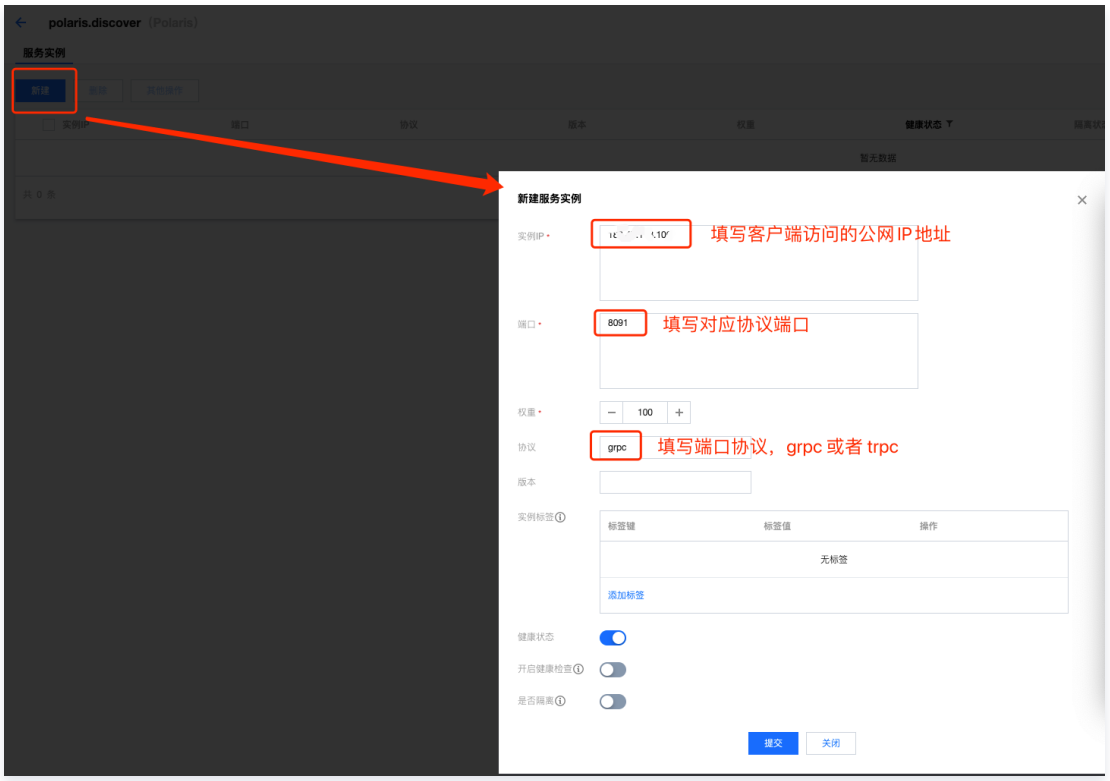
### 创建 polaris.discover 以及 polaris.healthcheck 服务

1. 在实例的基本信息页面中，记住访问地址中的公网 IP 地址以及访问端口。

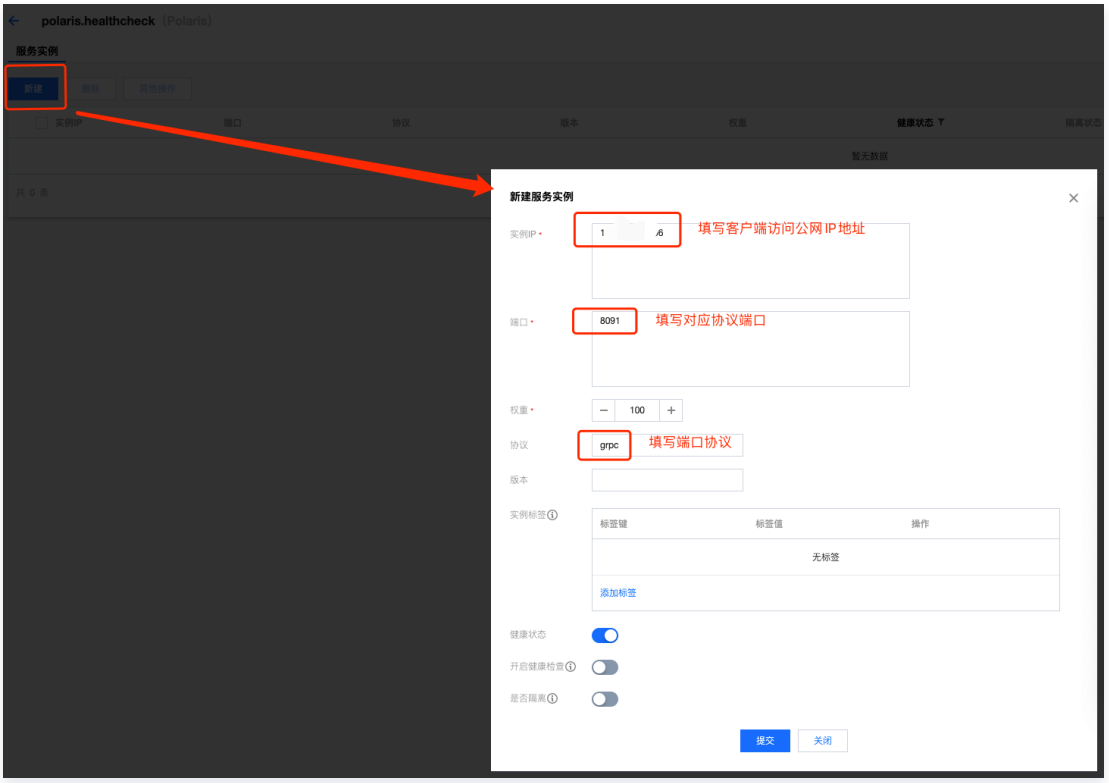

2. 在命名空间 Polaris下创建服务 polaris.discover 以及 polaris.healthcheck。



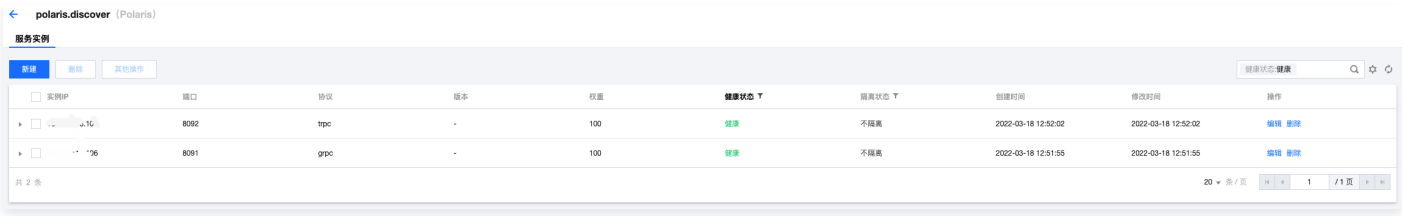
3. 单击服务 **polaris.discover**，进入到服务的实例页面，根据客户端公网访问 IP 地址以及端口协议 **grpc**、**trpc** 分别创建实例。



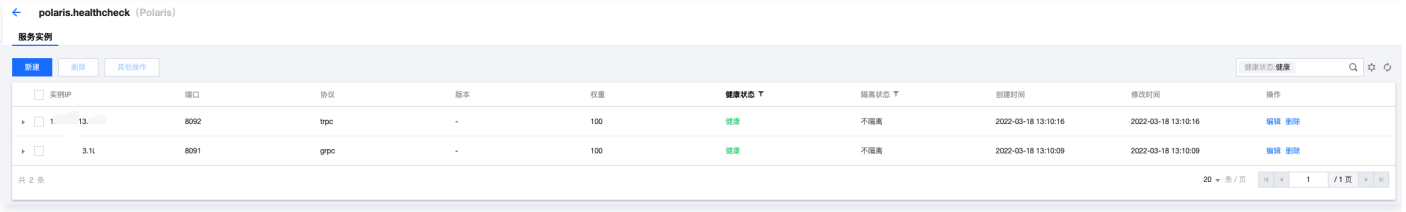

4. 单击服务 **polaris.healthcheck**，进入到服务的实例页面，根据客户端公网访问IP地址以及端口协议 **grpc**、**trpc** 分别创建实例。



5. 按照上述步骤的操作，最终服务 polaris.discover 的实例列表如下：

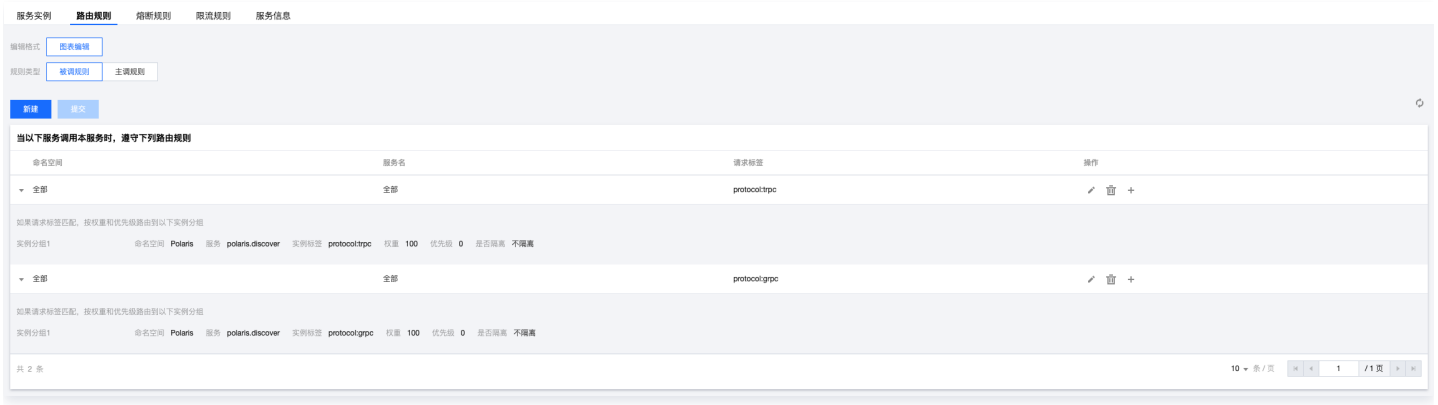



6. 按照上述步骤的操作，最终服务 polaris.healthcheck 的实例列表如下：

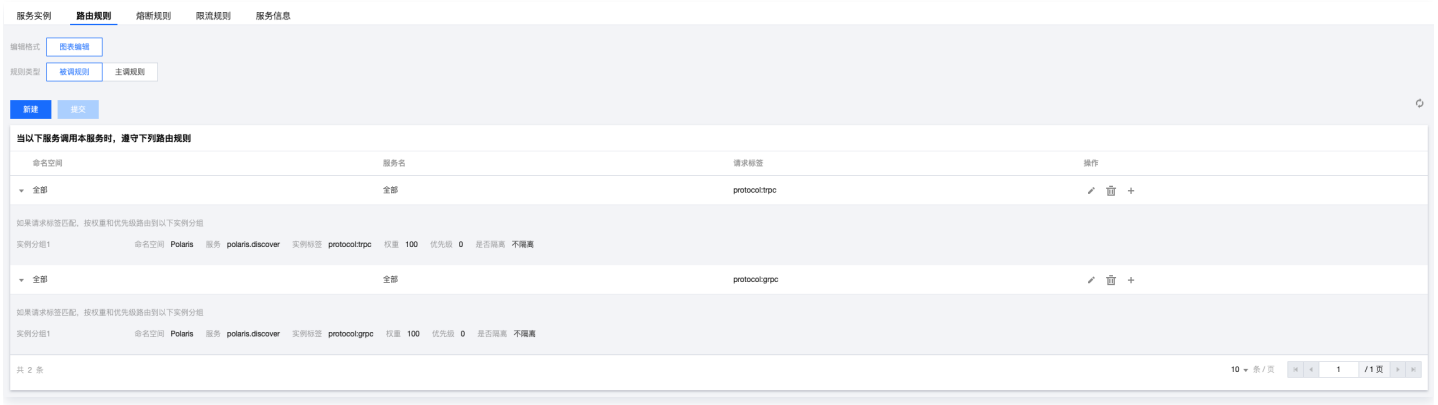



### 创建路由规则

#### polaris.discover




#### polaris.healthcheck




### polaris-go 接入相关配置文件说明

polaris.yaml 中按照下面示例进行配置二次寻址：

```
global:
  serverConnector:
    addresses:
      - '{对应公网接入地址}:8091'
  system:
    discoverCluster:
```

```
      namespace: Polaris
      service: polaris.discover
    healthCheckCluster:
      namespace: Polaris
      service: polaris.healthcheck
```

**trpc-go 接入相关配置文件说明**

```
plugins:
  registry:
    polaris:
      heartbeat_interval: 3000
      protocol: grpc
      address_list: '{对应公网接入地址}:8091'
      cluster_service:
        discover: polaris.discover
        health_check: polaris.healthcheck
  selector:
    polaris:
      address_list: '{对应公网接入地址}:8091'
      protocol: grpc
      cluster_service:
        discover: polaris.discover
        health_check: polaris.healthcheck
```

# 就近访问使用指南

最近更新时间：2025-04-17 16:00:53

## 操作场景

本文通过一个 demo 进行 Golang 应用接入Polaris（北极星）的全流程操作演示，帮助您快速了解如何使用北极星网格的就近路由能力。

## 前提条件

- 已创建 Polaris（北极星）实例，请参见 创建 Polaris（北极星）。
- 下载 Github 的 demo 源码 到本地并解压。
- 本地编译构建打包机器环境已安装了 golang 环境，并且能够访问 Github。
- 根据您自身的业务，已准备好业务部署的资源，虚拟机部署和容器化部署选择其中一种方式即可。
  - 虚拟机部署已创建 CVM 虚拟机，请参见 创建 CVM 虚拟机。
  - 容器化部署已创建 TKE 容器集群，请参见 创建 TKE 集群。

## 操作步骤

1. 登录 TSF 控制台。
2. 在左侧导航栏选择**Polaris（北极星）**，进入北极星实例列表页。
3. 单击目标引擎的“ID”，进入基本信息页面。
4. 查看访问地址，Golang 应用访问使用 gRPC 端口（8091）：


5. 修改 demo 中的注册中心地址：
   5.1 在下载到本地的 demo 源码 目录下，分别找到 `nearby/consumer/polaris.yaml` 以及 `nearby/provider/polaris.yaml` 文件。
   5.2 添加微服务引擎北极星网格地址到项目配置文件中（这里以 `nearby/consumer/polaris.yaml` 为例）。

```
global:
   serverConnector:
     addresses:
       - 10.0.4.6:8091
   location:
     # 设置 polaris-go 进程地理信息的提供插件为 腾讯云 插件
     # 如果当前进程在CVM或者TKE中，则能够自动获取当前进程所在的位置信息
     #
     # 如果自动获取失败，则可以设置插件为 env，然后在 linux中注入以下环境变量
     # POLARIS_INSTANCE_ZONE: 设置 zone 信息，例如 ap-guangzhou
     # POLARIS_INSTANCE_CAMPUS: 设置 IDC 信息，例如 ap-guangzhou-3
     provider: qcloud
consumer:
  serviceRouter:
    # 服务路由链
```

```
    chain:
      # 基于主调和被调服务规则的路由策略(默认的路由策略)
      - ruleBasedRouter
      # 就近路由策略
      - nearbyBasedRouter
    #描述：服务路由插件的配置
    plugin:
      nearbyBasedRouter:
        #描述:就近路由的最小匹配级别
        #范围:region(大区)、zone(区域)、campus(园区)
        matchLevel: campus
```

6. 部署 provider 和 consumer 微服务应用。详情请参见 Polaris-Golang Demo 部署 。
7. 进入前面提到的微服务引擎北极星网格实例页面。

- 选择**服务管理 > 服务列表**，查看微服务 RouteNearbyEchoServer 的实例数量。
  - 若实例数量值不为0，则表示已经成功接入微服务引擎。
  - 若实例数量为0，或者找不到 RouteNearbyEchoServer 服务名，则表示微服务应用接入微服务引擎失败。




- 开启 RouteNearbyEchoServer 的就近路由。




- 确认 consumer 所在 CVM（容器POD）的腾讯云位置信息。
- 调用 consumer 的 HTTP 接口，执行 http 调用，其中 `${app.port}` 替换为 consumer 的监听端口（默认为18080），`${add.address}` 则替换为 consumer 暴露的地址。

```
curl -L -X GET 'http://${add.address}:${app.port}/echo'
预期返回值：Hello, I'm RouteNearbyEchoServer Provider, MyLocInfo's : xxx, host : xxx:xxx
```

# 迁移指南
# TSF Consul 迁移方案

最近更新时间：2025-01-25 16:52:42

## 操作场景

当您在生产环境上已经使用了腾讯云的 TSF Consul 集群作为服务治理中心，并希望将其上已运行的服务迁移至腾讯云的北极星网格时，可以根据您的业务需求，在本指引中选择适合您的迁移方案。
本文主要介绍 TSF Consul 迁移的方案和操作步骤，帮助您的微服务应用的服务治理将 TSF Consul 集群迁移到北极星网格集群。

## 前提条件

- 已创建 PolarisMesh 北极星网格，请参见 创建 PolarisMesh 治理中心 。
- 本地编译构建打包机器环境已安装了Java JDK、Maven，并且能够访问 Maven 中央库。

## 迁移改造

### 应用侧改造

#### 依赖更新

您需要将 TSF Consul 对应的 SDK Spring Cloud TSF 升级为 Spring Cloud Tencent（后称 SCT ）。因为 Spring Cloud TSF 在依赖管理中推荐以其自身为parent，而 SCT 的定位是治理组件，给用户的服务调用组件提供服务治理功能，但不包含服务调用实际的组件，因此需要用户额外引入服务调用等其他组件。
以 Spring Cloud TSF 2021版本为例。业务代码项目**父依赖**更新为如下所示，该父 pom 指定了：

- spring-boot-starter-parent，开源 Spring Boot 的 版本号为2.7.18。
- spring-cloud-tencent-dependencies，开源 SCT 的版本号为2.0.0.0-2021.0.9。
- spring-cloud-dependencies，开源 Spring Cloud 的版本号为2021.0.9。

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <!-- Spring Boot Parent -->
    <parent>
        <artifactId>spring-boot-starter-parent</artifactId>
        <groupId>org.springframework.boot</groupId>
        <version>2.7.18</version>
    </parent>

    <groupId>com.tencent.tsf</groupId>
    <artifactId>tsf-demo</artifactId>
    <version>1.0.0</version>
    <packaging>pom</packaging>

    <modules>
        <module>provider-demo</module>
        <module>consumer-demo</module>
    </modules>

    <properties>
        <!-- Spring Cloud Tencent -->
        <revision>2.0.0.0-2021.0.9</revision>
        <!-- Spring Cloud -->
        <spring.cloud.version>2021.0.9</spring.cloud.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <!-- Spring Cloud Tencent Dependencies -->
            <dependency>
                <groupId>com.tencent.cloud</groupId>
                <artifactId>spring-cloud-tencent-dependencies</artifactId>
                <version>${revision}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>

            <!-- Spring Cloud Dependencies -->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring.cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

业务代码项目**子依赖**更新可以参考下方 pom.xml ，该 pom 引入了如下依赖：

- spring-cloud-starter-tencent-all，SCT 服务治理组件总包。
- spring-boot-starter-web，web应用组件，提供 HTTP 服务给外部调用。
- spring-cloud-starter-openfeign，支持应用使用 Feign 客户端调用其他服务。如果不使用 Feign 客户端，也可以不引入。
- spring-cloud-starter-bootstrap，支持应用使用 bootstrap.xml 或者 bootstrap.yaml 或者 bootstrap.yml 作为配置文件。Spring Cloud 2021 以前的版本默认引入改依赖，Spring Cloud 2021 开始不默认引入，因此需要手动引入。或改为 application.xml 或者 application.yaml 或者 application.yml 。

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>com.tencent.tsf</groupId>
        <artifactId>tsf-demo</artifactId>
        <version>1.0.0</version>
    </parent>

    <artifactId>consumer-demo</artifactId>
    <packaging>jar</packaging>
```

```
    <name>consumer-demo</name>

    <dependencies>
        <!-- Spring Cloud Tencent 服务治理组件 -->
        <dependency>
            <groupId>com.tencent.cloud</groupId>
            <artifactId>spring-cloud-starter-tencent-all</artifactId>
        </dependency>

        <!-- web应用组件，提供 HTTP 服务给外部调用 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- 支持应用使用 Feign 客户端调用其他服务。如果不使用 Feign 客户端，也可以不引入 -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
        </dependency>

        <!-- 开启 Bootstrap 阶段 -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-bootstrap</artifactId>
        </dependency>
    </dependencies>
</project>
```

**配置更新**

SCT 使用了全新的配置，完整配置及其说明如下所示。

```
server:
  port: 48082 # 端口号
spring:
  application:
    name: QuickstartCallerService # 服务名
  config:
    import: optional:polaris # 开启北极星的配置中心
  cloud:
    polaris:
      address: grpc://{北极星的ip}:8091 # 北极星的治理中心地址
      namespace: default # 服务的命名空间，默认default
      enabled: true # 开启北极星服务治理功能
      discovery:
        enabled: true # 开启北极星服务发现，默认true
        register: true  # 开启北极星服务注册，默认true
        heartbeat:
          enabled: true # 开启北极星心跳，默认true
        heartbeat-interval: 5 # 心跳间隔，默认5（秒）
        health-check-url: /quickstart/caller/healthCheck # 心跳前接口检测，默认为空
        zero-protection:
          enabled: true # 开启北极星服务发现零实例保护，默认为false
          is-need-test-connectivity: true # 开启北极星服务发现零实例保护连通性测试，默认为false
        instance-id: QuickstartCallerService-0 # 指定服务注册的实例ID，默认为注册后服务端返回值
        token: NO_TOKEN # 指定连接北极星的TOKEN，默认为空
        weight: 100 # 指定该服务被调用时的权重，默认为100
        version: 2.0.0.0 # 指定该服务注册时的版本号，默认为1.0.0
```

```
    protocol: HTTP # 指定该服务的对外提供服务协议，默认为HTTP
    service-list-refresh-interval: 60000 # 指定服务列表刷新间隔，默认为60000（毫秒）
    eager-load:
      enabled: true # 服务发现预热功能总开关，默认为false
      feign:
        enabled: true # Feign 服务发现预热开关，默认为true
  config:
    address: grpc://{北极星的ip}:8093 # 北极星的配置中心地址
    port: 8093 # 北极星的配置中心端口，默认为8093
    auto-refresh: true # 开启北极星配置自动刷新，默认为true
    refresh-type: reflect # 指定北极星配置刷新机制，默认为refresh_context，可选reflect、refresh_context
    data-source: polaris # 指定配置来源，默认为polaris
    groups:
      - name: ${spring.application.name} # 指定北极星配置分组名
        files: [ "config/caller.properties" ] # 指定北极星配置文件名列表
    enabled: true # 开启北极星配置中心功能
    token: NO_TOKEN # 指定连接北极星的TOKEN，默认为空
    shutdown-if-connect-to-config-server-failed: true # 开启配置中心连接快速失败，默认为true
    local-file-root-path: ./polaris/backup/config # 指定本地配置
    internal-enabled: true # 开启内部配置文件拉取，默认为true
    connect-remote-server: true # 是否连接远端配置中心，默认为true
    crypto:
      enabled: true # 开启配置加密，默认为true
  loadbalancer:
    strategy: roundRobin # 负载均衡策略，默认为polarisWeightedRoundRobin，可选roundRobin、random、
polarisWeightedRandom、polarisRingHash、polarisWeightedRoundRobin
  contract:
    enabled: true # 开启服务契约，默认为true
    exposure: true # 开启服务契约接口，默认为true
    report:
      enabled: true # 开启服务契约上报，默认为true
    base-package: com.tencent.cloud # 指定服务契约扫描的基础路径
    exclude-path: /echo # 排除以下路径的服务契约扫描，逗号分隔
    group: polaris # 指定服务契约分组，默认为polaris
    base-path: /** # 扫描以下路径的服务契约，逗号分隔，默认为/**
    name: Polaris # 服务契约名称，默认为服务名
  circuitbreaker:
    enabled: true # 开启服务熔断功能，默认为true
  ratelimit:
    enabled: true # 开启服务限流功能，默认为true
    reject-http-code: 429 # 服务限流时的返回码，默认为429
    reject-request-tips: ratelimited # 服务限流时的返回字段
    rejectRequestTipsFilePath: reject-tips.html # 服务限流时的返回HTML
    maxQueuingTime: 1000 # 匀速排队限流的最大排队时间，默认为1000（毫秒）
  router:
    enabled: true # 开启服务路由功能，默认为true
    metadata-router:
      enabled: true # 开启元数据路由，默认为true
    namespace-router:
      enabled: true # 开启命名空间路由，默认为false
      fail-over: all # 命名空间路由降级配置，默认为all，可选none
    nearby-router:
      enabled: true # 开启就近访问，默认为false
      match-level: ZONE # 指定最低就近级别，默认为ZONE，可选CAMPUS、ZONE、REGION
    rule-router:
      enabled: true # 开启规则路由，默认为true
      fail-over: all # 规则路由降级配置，默认为all，可选none
  admin:
    port: 28083 # 管理端口，默认为28080
```

```
      stat:
        enabled: true # 开启监控，默认为true
        path: /metrics # Prometheus的path，默认为/metrics
        pushgateway:
          enabled: true # 开启pushgateway模式，默认为false
          address: {北极星的ip}:9091 # 指定pushgateway地址
          push-interval: 30000 # 指定pushgateway间隔，默认为30000（毫秒）
      lossless:
        enabled: false # 开启优雅上下线，默认为false
        health-check-path: /health # 健康检查路径
        delay-register-interval: 30000 # 延迟上线时间，默认为30000（毫秒）
        health-check-interval: 1000 # 健康检查间隔，默认为1000（毫秒）
      local-ip-address: 127.0.0.1 # 指定注册ip
      local-port: 48082 # 指定注册端口号
      logging:
        path: ./polaris/logs # 指定北极星日志路径
    tencent:
      metadata:
        content:
          transk: transv # 指定服务实例元数据
          dispk: dispv
          region: huanan
          zone: shenzhen
          campus: shenzhen-1
        transitive:
          - tranv # 指定传递元数据的key列表
        disposable:
          - dispk # 指定单跳元数据的key列表
        headers:
          - headerk # 指定原始头部传递的key列表
      rpc-enhancement:
        enabled: true # 开启调用增强组件，默认为true
        reporter:
          enabled: true # 开启服务调用上报
          ignore-internal-server-error: true # 是否忽略500，默认为true
          series: server_error # 指定服务错误的HTTP系列码，默认为server_error（5xx）
          statuses: gateway_timeout, bad_gateway, service_unavailable # 指定服务错误的HTTP错误码
management:
  endpoints:
    web:
      exposure:
        include:
          - polarisdiscovery # 开启服务注册发现端点
          - polariscircuitbreaker # 开启服务熔断端点
          - polarisconfig # 开启动态配置端点
          - polarisroute # 开启服务路由端点
          - polarisratelimit # 开启服务限流端点
          - polarismetadata # 开启服务元数据端点
```

### 代码改造

#### 删除 Spring Cloud TSF 注解（可选）

以下注解为 Spring Cloud TSF 定义的功能注解，SCT 做了兼容化处理，无需引入注解也可开启对应功能，或通过配置进行关闭，引入注解不会报错。

```
@EnableTsfUnit // 新单元化单独开启
@EnableTsfAuth // 鉴权
@EnableTsfRoute // 路由
@EnableTsfRateLimit // 限流
```

```
@EnableTsfSleuth // 调用链
@EnableTsfMonitor // 监控
@EnableTsfCircuitBreaker // 熔断
@EnableTsfFaultTolerance // 服务容错
@EnableTsfLane // 泳道
@TsfUnitCall // 单元化相关
@TsfUnitCustomerIdentifier // 单元化相关
@TsfUnitLocalCall // 单元化相关
@EnableTsfZoneFilter
```

**服务容错代码定义改造**

- @TsfFaultTolerance 注解升级为 @FaultTolerance注解。TsfFaultToleranceProperty注解属性去除。
- TsfFaultToleranceStragety 注解属性升级为 FaultToleranceStrategy 注解属性。

**服务监听触发回调代码改造**

Spring Cloud TSF 中服务监听触发回调的接口为 ConsulServiceChangeListener，SCT 中升级为 ServiceInstanceChangeCallback。代码实现可以参考下方示例代码：

```
@Component
@ServiceInstanceChangeListener(serviceName = "provider-demo")
public class ProviderServiceChangeCallback implements ServiceInstanceChangeCallback {

    private static final Logger LOG = LoggerFactory.getLogger(ProviderServiceChangeCallback.class);

    @Override
    public void callback(List<Instance> currentServiceInstances, List<Instance> addServiceInstances,
List<Instance> deleteServiceInstances) {
        String current = generateNodeList(currentServiceInstances);
        String add = generateNodeList(addServiceInstances);
        String delete = generateNodeList(deleteServiceInstances);
        LOG.info("current: {}, add: {}, delete: {}", current, add, delete);
    }

    private String generateNodeList(List<Instance> deleteServiceInstances) {
        StringBuilder nodeListStr = new StringBuilder("[");
        for (Instance instance : deleteServiceInstances) {
            if (nodeListStr.length() > 1) {
                nodeListStr.append(", ");
            }
            nodeListStr.append(instance.getHost()).append(":").append(instance.getPort());
        }
        nodeListStr.append("]");
        return nodeListStr.toString();
    }
}
```

**API Doc 改造**

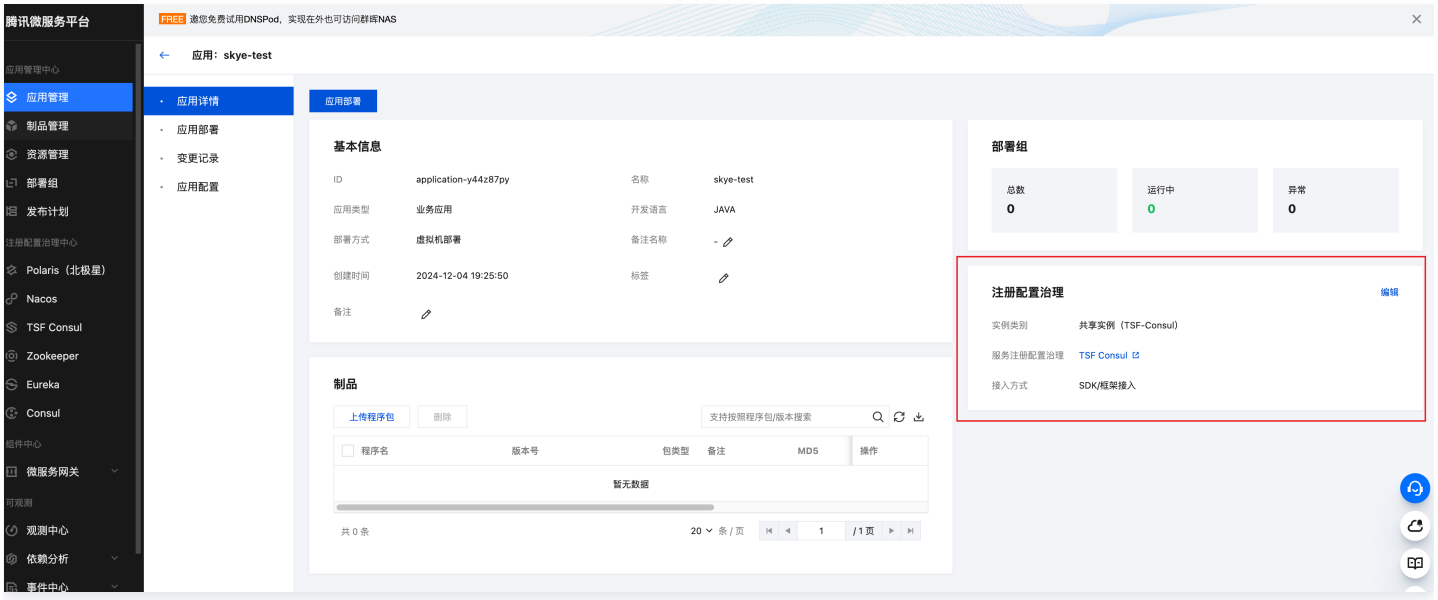
Spring Cloud TSF 使用 swagger 3.0.0 实现 API Doc 的管理，SCT 使用 springdoc 2.2.0 实现。目前开源 swagger 已于2020年停止维护，相关的很多 bugfix 和漏洞不会修复，目前持续维护的是 springdoc，springdoc官方迁移文档 。无法同时使用 swagger 和 springdoc，且为了您的项目的安全考虑，需要按照 springdoc 官方文档修改代码。
springdoc样例：

```
@RestController("/swagger")
@Tag(description = "swagger 测试", name = "swaggerValue1")
public class SwaggerApiController {
```

```
    @RequestMapping(value = "/swagger/findMessages", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    @Operation(method = "POST",
            summary = "根据任务ID查询任务列表",
            description = "根据任务ID查询任务列表Note")
    public List<MessageModel> findMessages(@RequestBody
    @Parameter(name = "msgIds", description = "消息ID列表") List<String> msgIds) {
        List<MessageModel> messageModels = new ArrayList<>();
        MessageModel messageModel = new MessageModel();
        messageModel.setMsgContent("test1");
        messageModel.setMsgId("1");
        messageModel.setSendTime(System.currentTimeMillis());
        MessageUser messageSender = new MessageUser();
        messageSender.setEmail("abc@xxxx.com");
        messageSender.setName("张三");
        messageSender.setOfficeAddress("腾讯大厦");
        messageSender.setPhoneNum("123123123");
        messageModel.setSendUser(messageSender);
        MessageUser messageReceiver = new MessageUser();
        messageReceiver.setEmail("abc@xxxx.com");
        messageReceiver.setName("李四");
        messageReceiver.setOfficeAddress("滨海大厦");
        messageReceiver.setPhoneNum("456456456");
        messageModel.setReceiveUsers(Lists.newArrayList(messageReceiver));
        messageModels.add(messageModel);
        return messageModels;
    }
}
```

## 应用管理改造

引入 Spring Cloud TSF 应用，注册到 TSF Consul 的应用详情如下图所示。




如果需要迁移到北极星网格，除了需要应用更新到 Spring Cloud Tencent，还需要编辑注册配置治理，开发框架选择 Spring Cloud，实例类别选择独享实例（北极星），然后选择可以正常连通的关联实例后，点击确认按钮。


**如果当前应用存在对应的部署组，则不能直接修改注册配置治理，而是需要先删除应用下的全部部署组，或者另外新建一个应用进行迁移。**

# Nacos 迁移方案
# 协议兼容接入

最近更新时间：2024-02-20 09:56:31

## 使用场景

如果您希望使用北极星代替 Nacos 作为新的注册中心，北极星提供了协议兼容的方式，您仅需要更改应用中的 nacos-client 的服务端接入地址即可。无需修改代码或者 pom 依赖。该能力目前开白使用，如需使用，可提工单，联系腾讯云助手协助。

## 前提条件

已创建 PolarisMesh 北极星网格，请参见 创建 PolarisMesh 治理中心 。

## 使用说明

本文通过服务注册和配置管理两个场景介绍北极星接入的使用说明。

### 服务注册

1. Nacos 服务名和北极星服务名映射关系

| Nacos 字段 | Nacos 字段值 | 北极星字段 | 北极星字段值描述 |
| --- | --- | --- | --- |
| namespace | 默认命名空间/非默认命名空间 ID | namespace | default/命名空间名称。 |
| group | DEFAULT_GROUP | service | 北极星服务名称由 nacos group 字段值和 nacos service 字段值拼接而成。${group}__${service} 为最终的北极星服务名，如果 group == DEFAULT_GROUP，则服务名为 ${service}。 |
| service | DEFAULT_GROUP | service | |
| cluster | DEFAULT | instance.metadata | 作为实例标签的一部分，实例标签 key 为 internal-nacos-cluster。 |

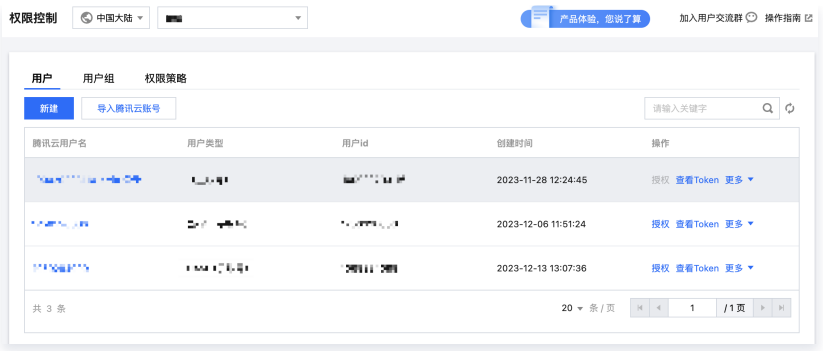
2. 修改服务端接入地址。

Spring Cloud Alibaba

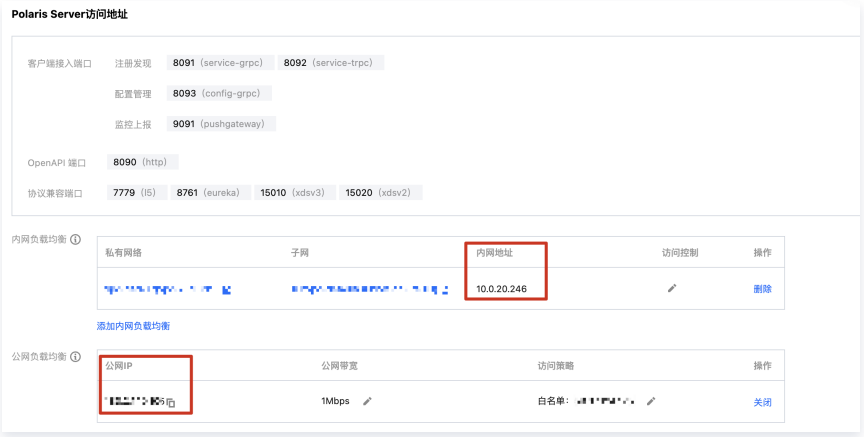
1. **修改应用配置文件：北极星服务端地址和服务端访问账密：**

```
spring.cloud.nacos.username="可任意值，如：username"
spring.cloud.nacos.password="北极星用户/用户组的资源访问凭据 Token"
spring.cloud.nacos.discovery.server-addr="北极星服务端 IP:8848"
spring.cloud.nacos.discovery.namespace="北极星命名空间名称"
```

2. **北极星用户/用户组的资源访问凭据 Token 查看方式：**
   - 登录 TSE 控制台 。
   - 在**服务治理中心**下，单击侧边栏**权限控制** ，选择目标北极星引擎实例。
   - 选择**用户**或者**用户组**，单击**查看Token** 。




3. **北极星服务端 IP 地址查看方式：**
   - 登录 TSE 控制台 。
   - 在**服务治理中心**下，单击侧边栏**实例列表**，单击目标北极星引擎实例。
   - 在引擎详情页的**实例信息**页面，可以查看到引擎的访问地址。

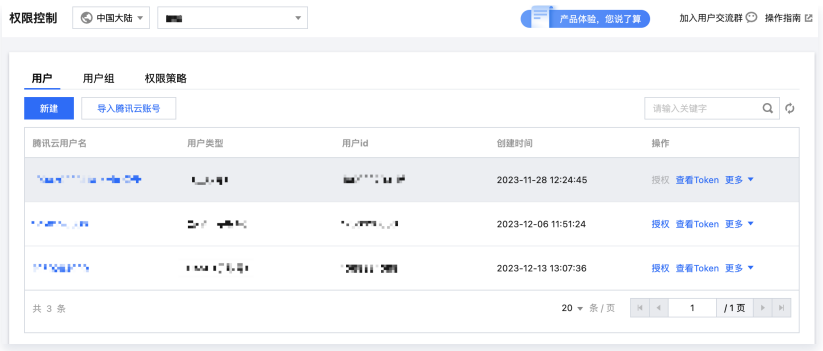



## 原生 Nacos-Client

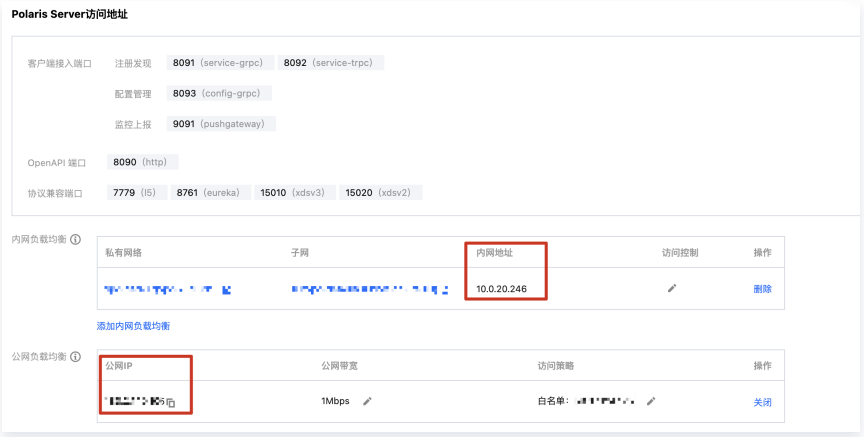
1. **原生Nacos Client修改：北极星服务端地址和服务端访问账密：**

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.SERVER_ADDR, "北极星服务端IP:8848");
properties.put(PropertyKeyConst.NAMESPACE, "北极星命名空间名称");
properties.put(PropertyKeyConst.USERNAME, "可任意值");
properties.put(PropertyKeyConst.PASSWORD, "北极星用户/用户组的资源访问凭据 Token");

// 创建注册发现客户端
NamingService namingService = NacosFactory.createNamingService(properties);
```

2. **北极星用户/用户组的资源访问凭据 Token 查看方式：**
   - 登录 TSE 控制台 。
   - 在 **服务治理中心** 下，点击侧边栏 **权限控制** ，选择目标北极星引擎实例。
   - 选择 **用户** 或者 **用户组**，点击 **查看Token** 。




3. **北极星服务端 IP 地址查看方式：**
   - 登录 TSE 控制台 。
   - 在 **服务治理中心** 下，点击侧边栏 **实例列表** ，点击目标北极星引擎实例。
   - 在引擎详情页的**实例信息**页面，可以查看到引擎的访问地址。

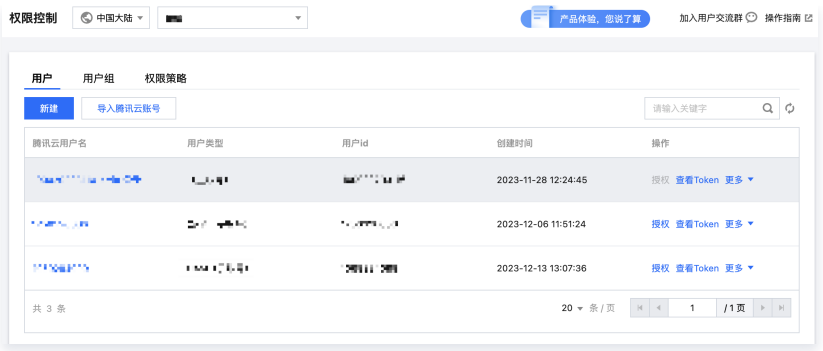



## Dubbo

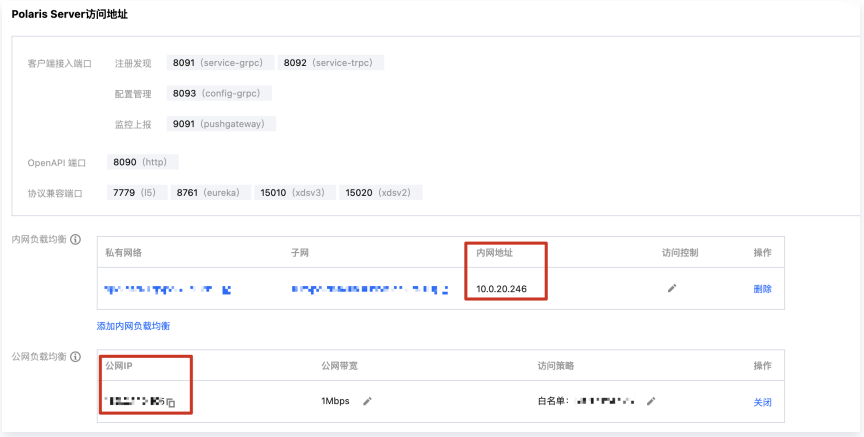
1. **修改应用配置文件：北极星服务端地址和服务端访问账密：**

```
dubbo
 registry
   address: nacos://北极星服务端IP:8848?username=可任意值&password=北极星用户/用户组的资源访问凭据 Token
   parameters.namespace: 北极星命名空间名称
  metadata-report
    address: nacos://北极星服务端IP:8848
```

2. **北极星用户/用户组的资源访问凭据 Token 查看方式：**
   - 登录 TSE 控制台 。
   - 在**服务治理中心**下，单击侧边栏**权限控制** ，选择目标北极星引擎实例。
   - 选择**用户**或者**用户组**，单击**查看Token** 。




3. **北极星服务端 IP 地址查看方式：**
   - 登录 TSE 控制台 。
   - **在服务治理中心下，单击侧边栏实例列表，单击目标北极星引擎实例。**
   - **在引擎详情页的实例信息页面，可以查看到引擎的访问地址。**




## 配置管理

1. **Nacos 配置信息和北极星配置信息映射关系**

| Nacos 字段 | Nacos 字段值 | 北极星字段 | 北极星字段值描述 |
| --- | --- | --- | --- |
| namespace | 默认命名空间/非默认命名空间 ID | namespace | default/命名空间名称 |
| group | DEFAULT_GROUP | group | 北极星配置分组名称 |
| dataId | application.yaml | file_name | 北极星配置文件名称 |

2. **修改服务端接入地址**

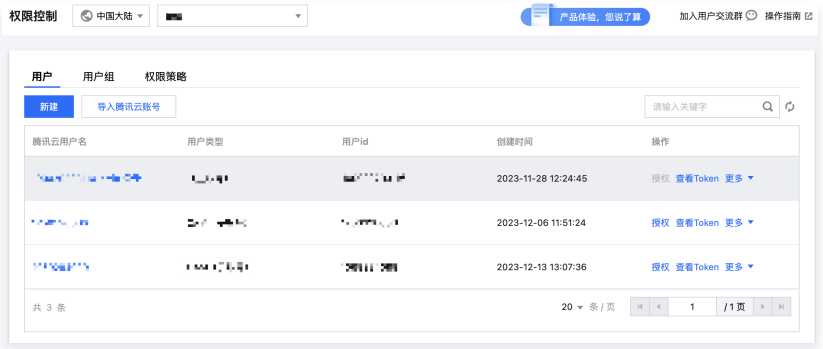
### Spring Cloud Alibaba

1. **修改应用配置文件：北极星服务端地址和服务端访问账密：**

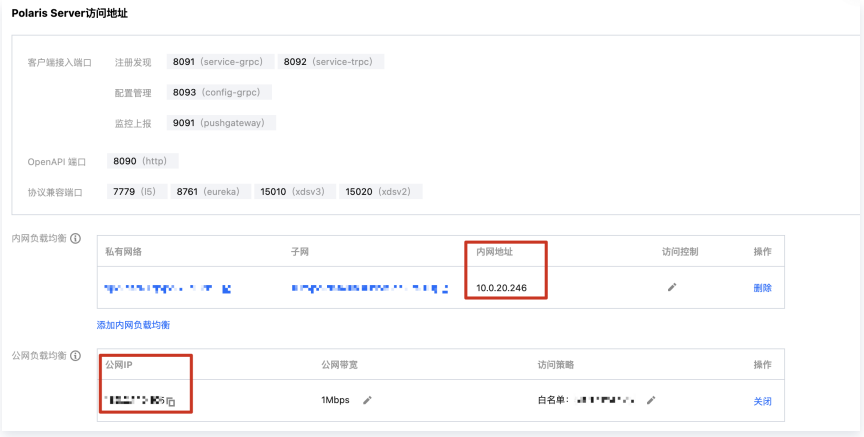
```
spring.cloud.nacos.username="可任意值"
spring.cloud.nacos.password="北极星用户/用户组的资源访问凭据 Token"
spring.cloud.nacos.config.namespace="北极星命名空间名称"
spring.cloud.nacos.config.server-addr="北极星服务端IP:8848"
spring.cloud.nacos.config.group="北极星配置分组名称"
```

2. **北极星用户/用户组的资源访问凭据 Token 查看方式：**
   - 登录 TSE 控制台 。

- 在**服务治理中心**下，单击侧边栏**权限控制**，选择目标北极星引擎实例。
- 选择**用户**或者**用户组**，单击**查看Token**。




3. **北极星服务端 IP 地址查看方式：**
   - 登录 TSE 控制台。
   - 在**服务治理中心**下，单击侧边栏**实例列表**，单击目标北极星引擎实例。
   - 在引擎详情页的**实例信息**页面，可以查看到引擎的访问地址。

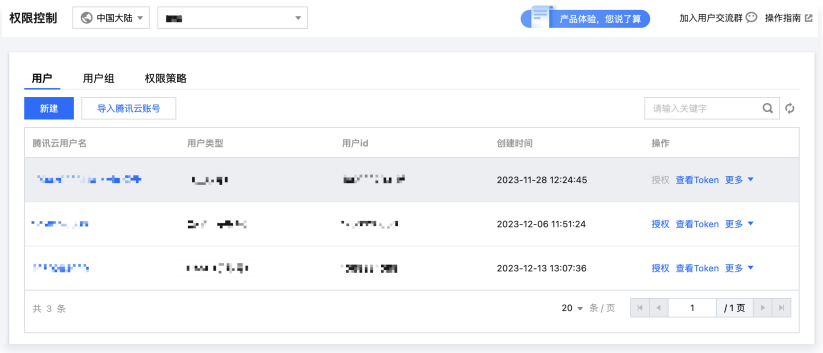



## 原生 Nacos-Client

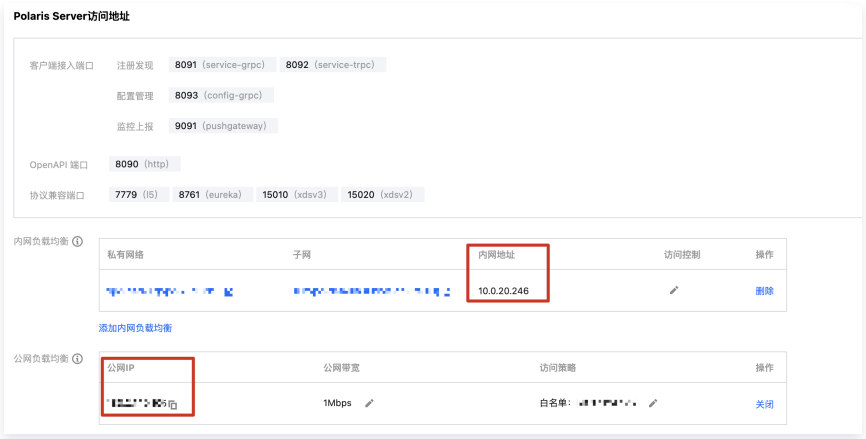
1. **原生Nacos Client修改：北极星服务端地址和服务端访问账密：**

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.SERVER_ADDR, "北极星服务端IP:8848");
properties.put(PropertyKeyConst.NAMESPACE, "北极星命名空间名称");
properties.put(PropertyKeyConst.USERNAME, "可任意值");
properties.put(PropertyKeyConst.PASSWORD, "北极星用户/用户组的资源访问凭据 Token");

// 注册配置客户端
ConfigService configService = new NacosConfigService(properties);
```

2. **北极星用户/用户组的资源访问凭据 Token 查看方式：**
   - 登录 TSE 控制台。
   - 在**服务治理中心**下，单击侧边栏**权限控制**，选择目标北极星引擎实例。
   - 选择**用户**或者**用户组**，单击**查看Token**。




3. **北极星服务端 IP 地址查看方式：**
   - 登录 TSE 控制台 。
   - 在**服务治理中心**下，单击侧边栏**实例列表**，单击目标北极星引擎实例。
   - 在引擎详情页的**实例信息**页面，可以查看到引擎的访问地址。

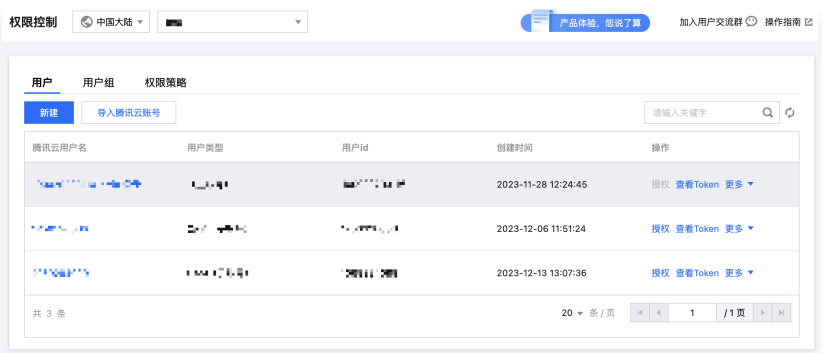



## Dubbo

1. **修改应用配置文件：北极星服务端地址和服务端访问账密：**

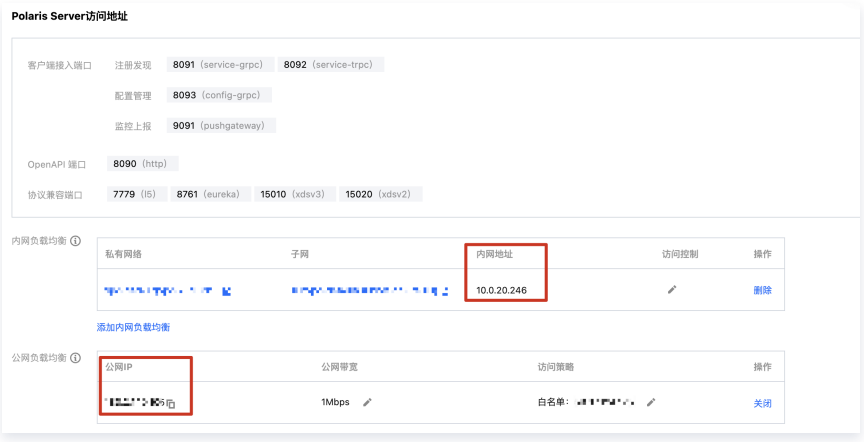
```
dubbo
  config-center
    address: nacos://北极星服务端IP:8848
```

2. **北极星用户/用户组的资源访问凭据 Token 查看方式：**
   - 登录 TSE 控制台 。
   - 在**服务治理中心**下，单击侧边栏**权限控制**，选择目标北极星引擎实例。
   - 选择**用户**或者**用户组**，单击**查看Token** 。




3. **北极星服务端 IP 地址查看方式：**

- 登录 TSE 控制台 。
- 在**服务治理中心**下，单击侧边栏**实例列表** ，单击目标北极星引擎实例。
- 在引擎详情页的**实例信息**页面，可以查看到引擎的访问地址。



# Eureka 迁移方案
# Eureka协议使用命名空间

最近更新时间：2023-04-20 15:59:57

## 介绍

北极星对 Eureka 的 API 进行了全兼容，由于 Eureka 默认不支持命名空间，北极星拓展了一个请求头 `x-namespace` 来支持命名空间的隔离。当有客户端的请求到达服务端，服务端将检查这个 header：

- 如果 header 存在，则使用这个 header 标识的命名空间进行服务注册发现的隔离。
- 如果 header 不存在或者值为空，则使用默认的命名空间 `default` 。

## 示例

下面通过 eureka 协议的api进行说明。向命名空间 `ns1` 下注册服务 `EUREKA-DEMO-PROVIDER` 的一个实例，实例端口为18081。

```
curl -X 'POST' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER' \
-H "Content-type: application/json" \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-d '{
    "instance": {
        "instanceId": "10.91.80.100:eureka-demo-provider:18081",
        "hostName": "10.91.80.100",
        "app": "EUREKA-DEMO-PROVIDER",
        "ipAddr": "10.91.80.100",
        "status": "UP",
        "overriddenStatus": "UNKNOWN",
        "port": {
            "$": 18081,
            "@enabled": "true"
        },
        "securePort": {
            "$": 443,
            "@enabled": "false"
        },
        "countryId": 1,
        "dataCenterInfo": {
            "@class": "com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo",
            "name": "MyOwn"
        },
        "leaseInfo": {
            "renewalIntervalInSecs": 30,
            "durationInSecs": 90,
            "registrationTimestamp": 0,
            "lastRenewalTimestamp": 0,
            "evictionTimestamp": 0,
            "serviceUpTimestamp": 0
        },
        "metadata": {
            "management.port": "18081"
        },
        "homePageUrl": "http://10.91.80.100:18081/",
        "statusPageUrl": "http://10.91.80.100:18081/actuator/info",
        "healthCheckUrl": "http://10.91.80.100:18081/actuator/health",
        "vipAddress": "EUREKA-DEMO-PROVIDER",
        "secureVipAddress": "EUREKA-DEMO-PROVIDER",
        "isCoordinatingDiscoveryServer": "false",
```

```
        "lastUpdatedTimestamp": "1681266990026",
        "lastDirtyTimestamp": "1681266990889"
    }
}' \
-v
```

向命名空间 `ns2` 下注册服务 `EUREKA-DEMO-PROVIDER` 的一个实例，实例端口为18082。

```
curl -X 'POST' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER' \
-H "Content-type: application/json" \
-H "Accept: application/json" \
-H "x-namespace: ns2" \
-d '{
    "instance": {
        "instanceId": "10.91.80.100:eureka-demo-provider:18082",
        "hostName": "10.91.80.100",
        "app": "EUREKA-DEMO-PROVIDER",
        "ipAddr": "10.91.80.100",
        "status": "UP",
        "overriddenStatus": "UNKNOWN",
        "port": {
            "$": 18082,
            "@enabled": "true"
        },
        "securePort": {
            "$": 443,
            "@enabled": "false"
        },
        "countryId": 1,
        "dataCenterInfo": {
            "@class": "com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo",
            "name": "MyOwn"
        },
        "leaseInfo": {
            "renewalIntervalInSecs": 30,
            "durationInSecs": 90,
            "registrationTimestamp": 0,
            "lastRenewalTimestamp": 0,
            "evictionTimestamp": 0,
            "serviceUpTimestamp": 0
        },
        "metadata": {
            "management.port": "18082"
        },
        "homePageUrl": "http://10.91.80.100:18082/",
        "statusPageUrl": "http://10.91.80.100:18082/actuator/info",
        "healthCheckUrl": "http://10.91.80.100:18082/actuator/health",
        "vipAddress": "EUREKA-DEMO-PROVIDER",
        "secureVipAddress": "EUREKA-DEMO-PROVIDER",
        "isCoordinatingDiscoveryServer": "false",
        "lastUpdatedTimestamp": "1681266990026",
        "lastDirtyTimestamp": "1681266990889"
    }
}' \
-v
```

查询 `ns1` 下 `EUREKA-DEMO-PROVIDER` 的实例。

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER' \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-v
## 返回18081端口实例
```

查询 `ns2` 下 `EUREKA-DEMO-PROVIDER` 的实例。

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER' \
-H "Accept: application/json" \
-H "x-namespace: ns2" \
-v
## 返回18082端口实例
```

如上所见，向不同命名空间下相同的服务名进行服务注册发现，已经通过命名空间进行隔离。

# API 支持列表

## 注册服务实例

- 请求方法：PUT
- 路由：/eureka/apps/{appId}
- 示例：

```
curl -X 'POST' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER' \
-H "Content-type: application/json" \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-d '{
    "instance": {
        "instanceId": "10.91.80.100:eureka-demo-provider:18081",
        "hostName": "10.91.80.100",
        "app": "EUREKA-DEMO-PROVIDER",
        "ipAddr": "10.91.80.100",
        "status": "UP",
        "overriddenStatus": "UNKNOWN",
        "port": {
            "$": 18081,
            "@enabled": "true"
        },
        "securePort": {
            "$": 443,
            "@enabled": "false"
        },
        "countryId": 1,
        "dataCenterInfo": {
            "@class": "com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo",
            "name": "MyOwn"
        },
        "leaseInfo": {
            "renewalIntervalInSecs": 30,
            "durationInSecs": 90,
            "registrationTimestamp": 0,
            "lastRenewalTimestamp": 0,
            "evictionTimestamp": 0,
            "serviceUpTimestamp": 0
        },
```

```
        "metadata": {
            "management.port": "18081"
        },
        "homePageUrl": "http://10.91.80.100:18081/",
        "statusPageUrl": "http://10.91.80.100:18081/actuator/info",
        "healthCheckUrl": "http://10.91.80.100:18081/actuator/health",
        "vipAddress": "EUREKA-DEMO-PROVIDER",
        "secureVipAddress": "EUREKA-DEMO-PROVIDER",
        "isCoordinatingDiscoveryServer": "false",
        "lastUpdatedTimestamp": "1681266990026",
        "lastDirtyTimestamp": "1681266990889"
    }
}' \
-v
```

### 健康检查

- 请求方法：PUT
- 路由：/eureka/apps/{appId}/{instanceId}
- 示例：

```
curl -X 'PUT' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER/10.91.80.100:eureka-demo-
provider:18081' \
-H "x-namespace: ns1" \
-v
```

### 取消注册

- 请求方法：DELETE
- 路由：/eureka/apps/{appId}/{instanceId}
- 示例：

```
curl -X 'DELETE' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER/10.91.80.100:eureka-demo-
provider:18081' \
-H "x-namespace: ns1" \
-v
```

### 查询所有实例

- 请求方法：GET
- 路由：/eureka/apps
- 示例：

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/apps' \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-v
```

### 查询 delta 实例

- 请求方法：GET
- 路由：/eureka/apps/delta
- 示例：

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/apps/delta' \
```

```
-H "Accept: application/json" \
-v
```

### 查询某个服务下所有实例

- 请求方法：GET
- 路由：/eureka/apps/{appId}
- 示例：

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER' \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-v
```

### 查询某个服务下某个实例

- 请求方法：GET
- 路由：/eureka/apps/{appId}/{instanceId}
- 示例：

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER/10.91.80.100:eureka-demo-
provider:18081' \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-v
```

### 查询某个实例

- 请求方法：GET
- 路由：/eureka/instance/{instanceId}
- 示例：

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/instance/10.91.80.100:eureka-demo-provider:18081' \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-v
```

### 查询 vip 下的所有实例

- 请求方法：GET
- 路由：/eureka/vips/{vip}
- 示例：

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/vips/EUREKA-DEMO-PROVIDER' \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-v
```

### 查询 secure vip 下的所有实例

- 请求方法：GET
- 路由：/eureka/svips/{svip}
- 示例

```
curl -X 'GET' 'http://127.0.0.1:8761/eureka/svips/EUREKA-DEMO-PROVIDER' \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-v
```

### 强制隔离实例

- 请求方法：PUT
- 路由：/eureka/apps/{appId}/{instanceId}/status
- 示例

```
curl -X 'PUT' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER/10.91.80.100:eureka-demo-
provider:18081/status' \
-H "x-namespace: ns1" \
-v
```

### 关闭强制隔离

- 请求方法：DELETE
- 路由：/eureka/apps/{appId}/{instanceId}/status
- 示例

```
curl -X 'DELETE' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER/10.91.80.100:eureka-demo-
provider:18081/status' \
-H "x-namespace: ns1" \
-v
```

### 更新元数据

- 请求方法：PUT
- 路由：/eureka/apps/{appId}/{instanceId}/metadata?{key}={value}
- 示例

```
curl -X 'PUT' 'http://127.0.0.1:8761/eureka/apps/EUREKA-DEMO-PROVIDER/10.91.80.100:eureka-demo-
provider:18081/metadata?k=v' \
-H "x-namespace: ns1" \
-v
```

### 批量复制

- 请求方法：POST
- 路由：/eureka/peerreplication/batch
- 示例

```
curl -X 'POST' 'http://127.0.0.1:8761/eureka/peerreplication/batch' \
-H "Content-type: application/json" \
-H "Accept: application/json" \
-H "x-namespace: ns1" \
-d '{
    "replicationList":[
        {
            "appName": "EUREKA-DEMO-PROVIDER",
            "id": "10.91.80.100:eureka-demo-provider:18081",
            "lastDirtyTimestamp": 1681266990889,
            "overriddenStatus": "UNKNOWN",
```

```
            "status": "UP",
            "instanceInfo": {
                "instanceId": "10.91.80.100:eureka-demo-provider:18081",
                "hostName": "10.91.80.100",
                "app": "EUREKA-DEMO-PROVIDER",
                "ipAddr": "10.91.80.100",
                "status": "UP",
                "overriddenStatus": "UNKNOWN",
                "port": {
                    "$": 18081,
                    "@enabled": "true"
                },
                "securePort": {
                    "$": 443,
                    "@enabled": "false"
                },
                "countryId": 1,
                "dataCenterInfo": {
                    "@class": "com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo",
                    "name": "MyOwn"
                },
                "leaseInfo": {
                    "renewalIntervalInSecs": 30,
                    "durationInSecs": 90,
                    "registrationTimestamp": 0,
                    "lastRenewalTimestamp": 0,
                    "evictionTimestamp": 0,
                    "serviceUpTimestamp": 0
                },
                "metadata": {
                    "management.port": "18081"
                },
                "homePageUrl": "http://10.91.80.100:18081/",
                "statusPageUrl": "http://10.91.80.100:18081/actuator/info",
                "healthCheckUrl": "http://10.91.80.100:18081/actuator/health",
                "vipAddress": "EUREKA-DEMO-PROVIDER",
                "secureVipAddress": "EUREKA-DEMO-PROVIDER",
                "isCoordinatingDiscoveryServer": "false",
                "lastUpdatedTimestamp": "1681266990026",
                "lastDirtyTimestamp": "1681266990889"
            },
            "action": "Register"
        }

    ]
}' \
-v
```

# 从 Eureka 迁移到 Polaris

最近更新时间：2024-01-11 15:30:01

## 操作场景

本文通过一个 demo 进行应用使用 `spring-cloud-eureka-client` 接入微服务引擎托管的 PolarisMesh 治理中心的全流程操作演示，帮助您快速了解如何使用北极星网格。

## 前提条件

- 已创建 PolarisMesh 北极星网格，请参见 创建 PolarisMesh 治理中心 。
- 下载 Github 的 demo 源码 到本地并解压。
- 本地编译构建打包机器环境已安装了Java JDK、Maven，并且能够访问 Maven 中央库。
- 根据您自身的业务，已准备好业务部署的资源， `虚拟机部署` 、 `容器化部署` 和 `TEM 部署` 选择其中一种方式即可。
  - **虚拟机部署**已创建 CVM 虚拟机，请参见 创建 CVM 虚拟机 。
  - **容器化部署**已创建 TKE 容器集群，请参见 创建 TKE 集群 。
  - **TEM部署**已创建 TEM 环境，请参见 创建 TEM 环境 。

## 操作步骤

1. 登录 TSE 控制台 。
2. 在**北极星网格**下的 **polarismesh** 页面，单击页面左上方下拉列表，选择目标地域。
3. 单击目标引擎的“ID”，进入基本信息页面。
4. 查看访问地址，eureka-client 应用访问使用 eureka 端口（8761）：


5. 修改 demo 中的注册中心地址。
   - 在下载到本地的 demo 源码目录 下，分别找到：
     `eureka/eureka-java/consumer/src/main/resources/application.yml` 和
     `eureka/eureka-java/provider/src/main/resources/application.yml` 两个文件。
   - 添加微服务引擎北极星网格地址到项目配置文件中（以 `eureka/eureka-java/consumer/src/main/resources/application.yml` 为例）。

```
eureka:
    client:
    serviceUrl:
      defaultZone: http://10.0.4.6:8761/eureka/
```

6. 打包 demo 源码成 jar 包。
   6.1 在 `eureka-java` 源码根目录下，打开 cmd 命令，执行 mvn clean package 命令，对项目进行打包编译。
   6.2 编译成功后，生成如下表所示的2个 Jar 包。

| 软件包所在目录 | 软件包名称 | 说明 |
| --- | --- | --- |
| \eureka-java\provider\target | eureka-provider-${version}-SNAPSHOT.jar | 服务生产者 |
| \eureka-java\consumer\target | eureka-consumer-${version}-SNAPSHOT.jar | 服务消费者 |

7. 部署 provider 和 consumer 微服务应用，虚拟机部署方式、容器化部署方式以及TEM部署方式根据您业务实际的部署方式选择一种即可。
   7.1 **虚拟机部署**方式部署 provider 和 consumer 微服务应用。
   - 上传 Jar 包至 CVM 实例。
   - 执行启动命令进行启动：

```
nohup java -jar [jar包名称] &
```

   7.2 **容器化部署**方式部署 provider 和 consumer 微服务应用。
   - 编写 dockerfile 生成镜像，参考：

```
FROM java:8
ADD ./eureka-provider-${VERSION}.jar /root/app.jar
ENTRYPOINT  ["java","-jar","/root/app.jar"]
```

   - 通过 TKE 部署并运行镜像。

   7.3 **TEM 部署**方式部署 provider 和 consumer 微服务应用。
   - 选择 TEM 环境，注意所选择的环境，其依赖的 VPC，必须和上面已经创建的北极星网格实例所依赖的 VPC 一致：

- 在已选择的环境中，新建 TEM 应用，相关参数填写参考：


- 部署应用，相关参数填写请参考（端口号映射，consumer 默认端口号为20002， provider 默认端口号为20001）：

- 查看访问路径，consumer 应用部署完后，可以在**基本信息 > 访问配置**中查看访问地址，如需公网访问，可以**编辑并更新**开启公网访问：


8. 确认部署结果。

8.1 进入前面提到的微北极星网格实例页面。

- 选择**服务管理 > 服务列表**，查看微服务 EUREKA-CONSUMER-SERVICE 和 EUREKA-PROVIDER-SERVICE 的实例数量：
- 若实例数量值不为0，则表示已经成功接入微服务引擎。
- 若实例数量为0，或者找不到具体服务的服务名，则表示微服务应用接入微服务引擎失败。


8.2 调用 consumer 的 HTTP 接口。执行 http 调用，其中 `${app.port}` 替换为 consumer 的监听端口（默认为20002），`${add.address}` 则替换为 consumer 暴露的地址。

```
curl -L -X GET 'http://${add.address}:${app.port}/echo?value=hello_world'
```

**预期返回值：** `echo: hello_world`