ALIBABA CLOUD

阿里云

应用高可用服务

应用高可用服务公共云合集

文档版本：20220608

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

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

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.动态与公告

1.1. 功能发布记录

本文介绍应用高可用服务AHAS（Application High Availability Service）产品的发布历史。

AHAS版本说明

变更时间	变更分类	产品模块	变更内容	产品文档
2021年02月	新功能	流量防护	支持Ingress接入Nginx防护，对其设置流控规则，实现应用高可用。	使用Ingress-sentinel实现流控
	新功能	流量防护	新增配置主动降级规则功能，主动降级规则可以指定对某些接口进行降级，无需修改代码，即可调整接口返回逻辑。	配置主动降级规则
2021年01月	新功能	故障演练	新增消息演练功能。AHAS可以自动感知消息服务架构拓扑，为您推荐演练方案，帮助您快捷进行演练，提前识别消息组件的不合理使用，规避风险。	什么是消息演练
2020年12月	新版本、新规格	故障演练	故障演练新增包年包月资源包，新规格区分入门包和企业包，包含月包、半年包和年包，可以满足不同客户需求的同时控制成本。	资源包
	新功能	流量防护	网关上可实现集群流控，从集群维度进行流量控制。	配置集群流控规则
	新版本、新规格	流量防护	AHAS发布流量防护多款规格的包月资源包，时长包含1个月到半年，节点数从20到1000，满足您的多种规模。且可与按量抵扣、后付费配合使用。	资源包

变更时间	变更分类	产品模块	变更内容	产品文档
2020年11月	新功能	故障演练	新增演练空间。演练空间可以对演练组织协同管理。可以根据业务、团队、活动等组织形态，灵活协同演练和成员，有效组织演练。	• 演练空间概述 • 管理演练空间
	体验优化	流量防护	优化集群流控功能，集群与应用相关联，去掉不必要的设置内容，简化集群流控的操作。	配置集群流控规则
2020年10月	新功能	故障演练	新增强弱依赖治理，以应用维度自动识别上下游依赖，引导用户治理业务系统非预期依赖。提前发现依赖问题可能导致的故障，避免其影响用户体验。	• 强弱依赖治理概述 • 应用强弱依赖治理
	体验优化	概览	子账号可在控制台上快速获取主账号ID。	如何查看主账号UID?
2020年09月	新功能	流量防护	可导出部分接口的某个时间段内的报告，便于做汇报、统计等。	接口详情
2020年08月	新功能	流量防护	集群流控可以精确地控制整个集群的调用总量，结合单机限流兜底，更好地发挥流量防护的效果。	配置集群流控规则
	新功能	流量防护	通过Nginx Sentinel模块可以快速接入至AHAS，当有请求流量时，可以在网关防护中查看Nginx网关请求的实时QPS与RT数据。	将Nginx接入流量防护
	新功能	流量防护	• 流量防护全新UI，应用概览数据统计更直观，体验更佳。 • 防护监控数据增加分位值、TOP统计及日粒度统计，便于您更好的查看接口。	接入应用方式

变更时间	变更分类	产品模块	变更内容	产品文档
2020年04月	体验优化	概览	AHAS概览页全新升级，高可用体系直观呈现。	无
2020年03月	新功能	流量防护	• 新增功能预案功能。 • AHAS流量防护接入方式新增GO语言。	接入Go应用
2019年12月	新功能	流量防护	AHAS计费模型优化，增加入门级防护模式。	应用基础设置
2019年06月	新功能	故障演练	故障演练支持Java故障注入。	演练场景说明
	新功能	架构感知	• 架构感知支持云服务Redis的实例识别。 • 展示资源使用情况。 • 架构感知主机层支持按照可用区划分。	云资源视图
2019年05月	新功能	架构感知	架构感知： • 主机层增加集群过滤条件。 • 集成云监控的指标报警信息（ECS、RDS、SLB）。 • 增加对CPU使用率和Load监控。	云资源视图
	新功能	流量防护	流控降级： 增加网关流控功能，支持API分组。	什么是网关防护
	新功能	架构感知	架构感知： • 主机层增加集群过滤条件。 • 集成云监控的指标报警信息（ECS、RDS、SLB）。 • RDS、SLB实例节点展示具体实例描述。	云资源视图

变更时间	变更分类	产品模块	变更内容	产品文档
2019年04月	新地域	概览	应用高可用服务AHAS上海Region开通上线。	无
	新功能	架构感知	架构感知： • 支持RDS云服务的实例识别。 • 容器组中Service Ingress支持流量配比展示。 • 应用名称识别与流控名称同步。 • 支持展示SLB实例监控信息，可识别SLB实例信息。	什么是架构感知
	新功能	架构感知	• 支持展示Ingress入口。 • 支持展示非Running状态的Pod。 • 架构感知记录回溯时间修改为7天。	无
	新功能	故障演练	故障演练： • 支持系统指标图的检测。 • 支持磁盘、网卡的下拉选择。	创建演练
2019年03月	新功能	流量防护	流量防护： • 新增子账号授权功能。 • 增加规则变更的操作日志。	应用概览
	体验优化	流量防护	流量防护： • 规则弹窗中默认折叠**来源应用**。 • 水位详情增加文案说明。 • 支持TPS跌0的展示。 • 支持监控图的缩放。 • 增加RT指标坐标轴。	无

变更时间	变更分类	产品模块	变更内容	产品文档
2019年01月	新功能	故障演练	故障演练服务全新升级，提供更流畅的演练体验。	什么是故障演练
	新功能	架构感知	架构感知支持采集并展示容器服务Kubernetes事件。	事件列表
2018年12月	新功能	流量防护	支持开源Sentinel SDK接入，允许用户独立使用AHAS的流控降级功能。	接入Dubbo应用
	新功能	流量防护	AHAS Agent最新版本支持探针的自动拉起与升级。	无
	体验优化	流量防护	AHAS Agent性能优化，降低资源消耗。	无
	新功能	架构感知	架构感知支持表格化展示所有节点，并支持导出表格数据。	列表信息
	新地域	流量防护	应用高可用服务AHAS公网Region开通上线，支持不限地域、不限厂商的用户接入。	应用高可用服务接入
	新功能	流量防护	优化流控降级： • 流控降级前端监控改版。 • 展现请求调用链路细节和调用关系。 • 新增机器列表，提供机器IP地址、进程ID等具体信息。	什么是应用防护
	新功能	流量防护	新增系统规则，从应用级别进行入口流量管控。	自适应流控
	新功能	架构感知	架构感知中的进程拓扑图支持结构化展示和多维度筛选。 进程拓扑图展示为四层：第一层是云服务或入网、出网进程，第二层是应用服务进程，第三层是存储服务进程，第四层是其它进程。	什么是架构感知

变更时间	变更分类	产品模块	变更内容	产品文档
2018年11月				
	新功能	架构感知	架构感知支持查看节点详情的监控数据。	什么是架构感知
	新地域	流量防护	应用高可用服务AHAS深圳Region开通上线。	无
	新功能	架构感知	架构感知支持通过简单易上手的操作体验Demo实例。	无
	新功能	架构感知	架构感知支持TCP6协议。	无
2018年09月	产品发布上线	AHAS	• 架构感知：自动感知应用的拓扑结构，以可视化的方式直观呈现应用对基础架构的依赖关系和组件间的依赖关系，并持续记录上述依赖关系，让您对系统架构一目了然。 • 应用高可用能力测评（现已更名为“故障演练”）：根据架构感知模块捕捉到的架构数据主动制造故障，检验应用系统及其各组件在故障下的可用性表现，从而验证应用系统的高可用能力，提前暴露故障隐患，帮助您针对性地应对风险。 • 应用高可用防护（现已更名为“流控降级”）：提供零代码改动、可一键启用的流控和降级两大可用性防护服务，为您的应用保障高可用性。 • 规则管理：提供统一的流控和降级规则管理模块，方便您集中管理规则。	什么是应用高可用服务AHAS

1.2. 探针版本说明

1.2.1. 应用高可用探针版本说明

应用高可用探针（即AHAS探针），使用架构感知、故障演练功能前需安装此探针。本文介绍AHAS探针版本发布历史。

版本号	变更时间	变更内容
1.14.0	2021年6月10日	• 支持托管其他故障注入工具—LitmusChaos。 • 支持自动安装和卸载Litmus。 • 支持下发Litmus故障演练。
1.13.2	2021年4月26日	• 支持异步故障下发。 • 优化探针性能。 • 支持金融云地域。 • 新增部分演练场景。 • 支持Windows操作系统。
1.13.1	2021年04月09日	新增探针卸载接口。
1.13.0	2021年01月22日	• 错误码统一处理，更新错误提示，并给出处理提示。 • 故障注入之前增加host机器环境检测。
1.12.0	2020年12月15日	• 支持短连接拓扑。 • 支持Kubernetes下Java演练场景。 • 支持Docker演练场景。 • 优化演练稳定性。 • 优化数据采集和上报。
1.11.0	2020年10月16日	• 新增DNS数据采集。 • 新增部分演练场景。
1.9.0	2020年07月10日	• 支持应用维度演练。 • 新增部分演练场景。
1.8.0	2020年04月28日	• 优化Kubernetes数据采集。 • 新增部分演练场景。
1.7.0	2020年01月10日	• 适配Kubernetes v1.16数据采集和演练。 • 新增部分演练场景。
1.6.0	2019年07月29日	新增Kubernetes演练场景。

版本号	变更时间	变更内容
1.5.0	2019年07月10日	• 新增对Kubernetes Virtual Node的支持。 • 添加了TLS认证通信方式。 • 添加数据压缩传输的能力。 • 优化Kubernetes数据上报方式：一个集群meta数据只上报一份。 • 精简数据上报内容，降低对带宽占用。数据总量降低1/5。
1.4.7	2019年06月25日	• 新增Java应用演练场景，和开源功能对齐。 • Kubernetes部署添加affinity亲和性配置，防止Agent部署在VK上。
1.4.6	2019年05月13日	• 新增磁盘IO数据采集。 • 修改ChaosBlade安装逻辑。 • 新增ChaosBlade升级功能。 • 修复/proc访问限制。
1.4.5	2019年05月05日	• 添加非running pod上报。 • 添加ingresses资源上报。 • 修改eventer版本至1.0.1，添加推送开关。 • ChaosBlade支持网卡和磁盘查询。
1.4.4	2019年04月12日	• 新增events组件，抓取Kubernetes事件流。 • 修复数据推送。 • 修改数据推送策略。 • 更新ChaosBlade版本。
1.4.3	2019年03月29日	• 采集数据增量上报。 • 建立连接时上报CPU、内存信息。 • 优化AHAS Agent安装失败错误提示。 • 优化Kubernetes下AHAS Agent Volumes。 • 新增Service EXTERNAL-IP数据上报。 • 修改容器sar信息采集周期。 • AHAS Agent接入容器服务应用目录。
1.4.2	2019年02月26日	修复部分容器进程、网络数据未抓取问题。
1.4.1	2019年02月19日	• 日志打印修改。 • 修复启动关闭超时。 • 修复helm chart镜像地址。 • 新增卸载退出码。

版本号	变更时间	变更内容
1.4.0	2019年01月30日	• 新增对Kubernetes集群的支持。 • 新增对Kubernetes集群自动升级的支持。 • 添加kubectl安装Agent。 • 添加helm chart安装Agent。 • 修复stime为0时，进程数据抓取问题。 • 优化网络抓取逻辑，当服务器存在5万连接时，CPU使用率在10%以下。 • AHAS Java Agent支持集群限流。
1.3.3	2019年01月25日	• 修复兜底方案。 • 设备connect新增版本字段。
1.3.2	2019年01月21日	Sentinel SDK上线。
1.3.1	2018年12月27日	支持探针自动升级。

1.2.2. 应用高可用Java探针版本说明

应用高可用Java探针，这是针对JVM的Java探针，如果需要使用流量防护功能，可安装此探针。本文记录了Java Agent的版本发布说明。

Java Agent

版本号	版本说明
1.9.0	• 支持JDK 6-11。 • 支持触发规则统计、异常统计。 • 修复Spring Web适配模块在异Servlet场景下可能出现的统计问题。
1.8.6	• Dubbo插件支持Dubbo 2.7.6+版本。 • 自适应流控新增采样日志。 • 支持上海金融云环境。 **说明** 建议1.8.x版本Java Agent至少升级至该版本。
1.8.5	• 新增主动降级规则支持。 • 支持控制台配置Web fallback行为。
1.8.4	支持MST压测功能。

版本号	版本说明
1.8.3	• 新增Dubbo 2.7.x adapter支持。 • 支持通过启动参数指定全局滑动窗口配置。 • 修复HTTP状态码统计模块线程池的问题。 • 完善对非HotSpot JVM的支持。
1.8.2	修复HTTP状态码统计模块可能导致CPU被占满的问题。
1.8.1	• 新增HTTP接口状态码统计，支持Spring MVC/Spring Boot/Spring Cloud/Spring Cloud Gateway/Zuul 1.x。 • 新增指标统计，操作系统指标新增内存、网络、磁盘等指标种类；JVM指标新增GC、JVM内存、线程数等指标。
1.8.0	• 新增Spring Web插件支持（REST API会自动提取pattern），原有的Servlet插件默认不开启。 • 新增MyBatis插件支持，原有的JDBC插件默认不开启。 • 同步SDK 1.7.x最新相关特性（自适应流控）。
1.7.10	• 修复熔断降级模块在同资源配置多个规则时可能出现无法从half-open状态中恢复的问题。 • 修复fastjson版本导致JDK 1.7下Agent无法正常工作的问题。
1.7.9	• 集群流控支持单机大流量场景（批量请求）。 • 修复熔断降级规则阈值比例为100%时不生效的问题。
1.7.8	• 修复Spring Cloud Gateway、Zuul等网关插件不生效的问题。 • 应用appType支持通过环境变量配置。
1.7.7	修复JDK 11下agent初始化失败的问题。
1.7.6	熔断降级支持渐进式恢复策略。
1.7.5	• 熔断降级特性改进：引入半开启探测模式，支持任意统计时长，RT模式改进为慢调用比例模式。 • 集群流控通信与异常处理优化。 • 支持张家口region。
1.7.4	• 支持集群流控特性。 • 支持SAE方式接入。
1.7.3	支持MySQL CJ Driver的SQL埋点。

版本号	版本说明
1.7.2	• 支持启动参数和环境变量控制所有插件模块的开启和关闭。 • 改进SQL含换行符情况下的处理，添加SQL埋点长度限制。 • 去除RT统计限制。 • 修复部分非daemon线程导致main函数执行结束后无法退出的问题。
1.7.1	• 同步基线至开源1.7.0版本，优化内存占用。 • 增加通用配置功能，支持簇点数目限制、context数目限制、来源数目限制、最大统计RT等基础配置项，支持动态配置。 • 增加热点规则支持。 • 默认关闭MQ相关插件以提升启动速度。
1.7.0	新增Memcached (spymemcached)、gRPC、Jedis、RocketMQ（callback模式）、RabbitMQ插件支持。 **说明** MQ插件可能会导致应用启动速度变慢。
1.6.0	新增MySQL（5.x driver）或Oracle SQL埋点支持。

1.3. 产品公告

1.3.1. AHAS功能开关商业化公告

尊敬的阿里云用户，AHAS功能开关功能将于2022年03月30日00:00:00起正式商用。如果您需要继续使用在公测期间开通的功能开关服务，将会产生相应的费用，请您关注。

商用收费说明

对于是否继续使用公测期间开通的功能开关功能，请根据实际需求及时处理。

- 继续使用功能开关：
  您无需进行任何操作，AHAS功能开关功能将于2022年03月30日00:00:00开始按照约定的计费规则进行计费。AHAS功能开关的计费规则，请参见价格说明。
- 不再使用功能开关：
  请您于2022年03月30日00:00:00前及时卸载接入功能开关的SDK来关闭该功能。具体操作，请参见如何卸载SDK埋点。

2.Ingress/Nginx防护

2.1. 什么是Ingress/Nginx防护

Nginx为目前比较流行的高性能开源服务器，Ingress则为实际的K8s流量入口。Ingress/Nginx防护作为应用侧的上游，可以提前对业务流量做控制，从而有效地保证下游服务不会因流量激增而导致系统瘫痪。

功能特性

Ingress/Nginx防护提供的主要功能如下：

功能	描述
请求分组管理	支持请求自定义分组。用户可以以该分组为粒度对其进行流量控制，并在AHAS控制台对其进行全方位的监控。请求分组通过域名加路径组合的方式定义，支持精确、前后缀、正则三种匹配模式，同时路径支持rewrite开关。
实时监控	Ingress/Nginx防护提供实时秒级监控能力，您接入Ingress/Nginx防护后，可以通过快速访问AHAS控制台，查看实时的接口以及自定义请求分组的QPS、RT等信息。
集群流控	通过设定集群流量总阈值，实现以服务或API粒度的控制流量的阈值，因此您无需关注集群机器的物理差异、负载均衡等因素。
自定义流控行为	当流量被Ingress/Nginx防护阻挡时，用户可以在前端配置自定义返回码和返回文本内容。

应用场景

秒杀购物

接入Ingress/Nginx防护后，通过提前对自定义请求分组的流量限制，可以有效地保证系统在峰值时不会因流量激增而导致系统瘫痪。

慢调用阻塞

下游的服务一旦出现慢调用频繁，比如SQL查询、异常的第三方调用等，都会占用网关侧的系统资源，从而连累下游其它正常服务。Ingress/Nginx防护可以有效地限制慢调用的频次来保护整个系统不被过多的慢调用拖垮。

精细化流控

网关侧做为总流量的入口，业务请求杂乱并且难以归类。Ingress/Nginx防护提供了自定义请求分组的功能，通过域名加路径组合的方式，同时支持精确、前后缀、正则三种匹配模型，以便用户能够更好地定义请求分组，并进一步对其做流量控制。

版本支持

使用Ingress/Nginx防护所需版本要求如下：

- 操作系统：支持主流Linux发行版本，包括CentOS、Ubuntu、Alpine等。
- Nginx版本：要求Nginx版本在1.10.3以上（不包括1.10.3）。若您当前使用的是低版本的Nginx，建议您升级后再使用。
- Nginx生态：支持主流的Nginx衍生产品，包括OpenRestry、Tengine、Kong、Apisix等。

接入方式

Ingress和Nginx如何接入流量防护的具体操作，请参见将Nginx接入流量防护和使用Ingress-sentinel实现流控。

2.2. 接入Ingress/Nginx防护

2.2.1. 将Nginx接入流量防护

Nginx是一款高性能开源的HTTP服务器，可通过Nginx Sentinel模块快速接入到AHAS中。当有请求流量时，您可以在AHAS控制台中查看Nginx网关请求的实时QPS和RT等数据。本文介绍如何将Nginx接入流量防护。

接入流量防护

1. 登录AHAS控制台，然后在页面左上角选择地域。
2. 在控制台左侧导航栏中选择**流量防护 > Ingress/Nginx防护**，然后单击**Ingress/Nginx接入**。
3. 单击**Nginx**页签，并根据**接入**页签下的接入指引完成Nginx接入。

体验Demo

AHAS提供体验Demo，可以让您通过Demo快速体验Nginx防护功能。

1. 登录AHAS控制台，然后在页面左上角选择地域。
2. 在控制台左侧导航栏中选择**流量防护 > Ingress/Nginx防护**，然后单击**Ingress/Nginx接入**。
3. 单击**Nginx**页签，并根据**体验Demo**页签下的接入指引完成Demo启动。

相关文档

Nginx Sentinel详细配置文档，请参见Nginx Sentinel模块配置（旧版）。

2.2.2. 使用Ingress-sentinel实现流控

ACK提供了Ingress-sentinel流控功能，通过Ingress-sentinel流控功能可以快速接入到AHAS中。当有请求流量时，您可以在AHAS控制台中查看Nginx网关请求的实时QPS和RT等数据。本文介绍如何使用Ingress-sentinel功能实现流控。

前提条件

- 开通AHAS
- 创建Kubernetes托管版集群
- 请确保ACK集群中的Ingress Controller运行正常，且不低于0.44.0版本。

背景信息

应用高可用服务AHAS（Application High Availability Service）提供了应用架构探测感知，故障注入式高可用能

力评测和流控降级高可用防护三大核心能力，其中针对Nginx的流控降级高可用防护功能已集成到ACK提供的Ingress-Nginx-Controller中，您可用根据实际需求开启该功能，查看流控数据。

使用限制

开启Ingress-sentinel流控功能，会消耗一定的系统资源。您使用的机器需要符合一定的配置，以下为配置说明：

- 最低配置：CPU为1000m Core，内存为2048 MiB。
- 建议配置：CPU为4000m Core及以上，内存为8192 MiB及以上。

开启Ingress-sentinel流控功能

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在配置项页面名称搜索框中搜索nginx-configuration，找到nginx-configuration，然后单击nginx-configuration**操作**列的**编辑**。
6. 自定义应用名称。

   在**编辑**面板单击**添加**，设置**名称**为sentinel-params，**值**为--app=demo-k8s-2020。

   > **说明** --app=demo-k8s-2020字段中demo-k8s-2020为应用名称，您可以根据实际需求替换应用名称。

7. （可选）如果您的集群所在地域不属于深圳、北京、上海、张家口、杭州，则还需要配置证书。

   单击**添加**，设置**名称**为sentinel-params，**值**为--app=demo-k8s --license=xxx。

   > **说明** 关于获取License的更多信息，请参见查看License。

8. （可选）如果您希望接收到超过流控配置上限的请求时，系统能够返回HTTP 429错误码（表示请求数过多）。则您可以配置流控HTTP返回码。

   单击**添加**，设置**名称**为sentinel-block-action，**值**为=429。

   > **说明** 如果您想要自定义HTTP错误码，可以根据实际需求修改429。

9. 开启Ingress-sentinel流控功能。

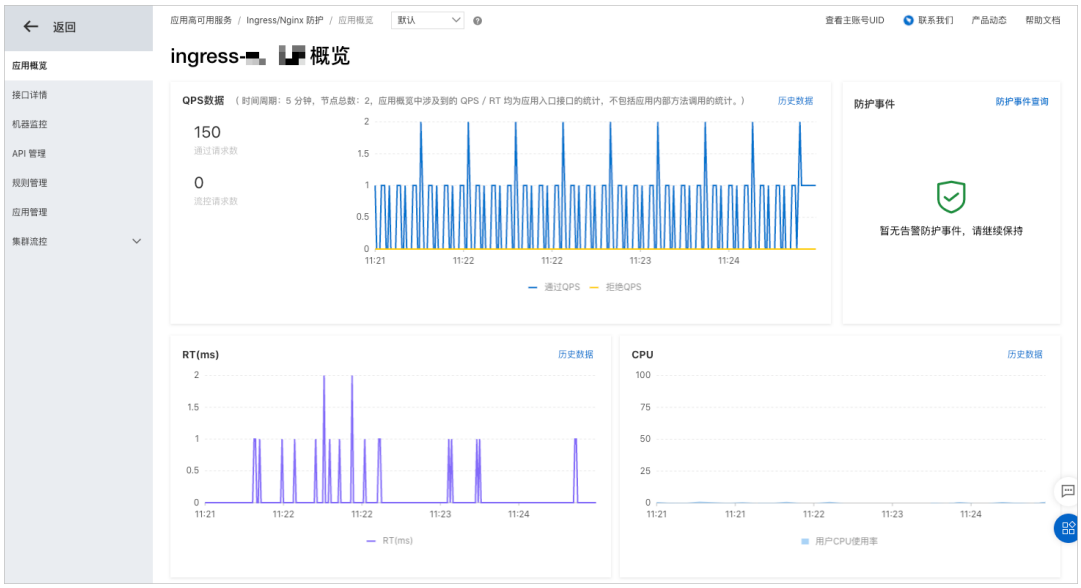
   > **说明** 接入Ingress-sentinel流控功能后默认不会触发流控，只有配置开启流控规则，并且请求流量超过流控规则限制后才会触发流控。

   单击**添加**，设置**名称**为use-sentinel，**值**为true。然后单击**确定**。

配置流控规则

登录AHAS控制台，在控制台左侧导航栏选择**流量防护 > Ingress/Nginx防护**，单击目标应用卡片，可以看到应用名称对应的流控监控窗口。具体操作请参见配置流控规则、配置隔离规则或配置熔断规则。

如果您需要配置更详细的流控规则，请参见请求分组管理 和配置流控规则。规则配置完成后，您可以在**应用概览**页面实时查看流控指标。



2.3. 配置防护规则

2.3.1. 配置流控规则

配置流控规则的原理是监控应用或服务流量的QPS指标，当指标达到设定的阈值时立即拦截流量，避免应用被瞬时的流量高峰冲垮，从而保障应用高可用性。本文介绍如何设置Nginx应用的流控规则。

前提条件

- 将Nginx接入流量防护
- 请求分组管理

操作步骤

1. 登录AHAS控制台，然后在页面左上角选择地域。
2. 在控制台左侧导航栏选择**流量防护 > Ingress/Nginx防护**，然后在**Ingress/Nginx防护**页面，单击Nginx资源卡片。
3. 选择以下任意一种方法进入**新增流控规则**页面：
   - 在左侧导航栏选择**应用概览**，然后单击页面下方目标接口**操作**列中的**流控**。
   - 在左侧导航栏选择**接口详情**，在**接口详情**页面单击资源卡片右上角 + 或 ⚙ 图标。
   - 在左侧导航栏选择**规则管理**，单击**新增流控规则**。
4. 在**新增流控规则**对话框中设置参数，然后单击**新建**。
   - **接口名称**：流控的目标接口，即在API管理中自定义或系统定义的API名称。
   - **单机QPS阈值**：设置的QPS阈值，等同于单机能够承受的限额。

   当流控规则生效，您可以在**应用概览**或**接口详情**页面通过QPS的统计图表查看。具体操作，请参见应用概览。

2.3.2. 配置集群流控规则

集群流控可以精确控制集群内某个服务的实时调用总量，适用于网关流量控制的场景。本文主要介绍设置集群流控的操作步骤。

计费说明

自2021年03月22日起，集群流控功能公测期结束，正式开始计费。集群流控功能按应用申请的QPS量级计费，具体计费方式，请参见价格页面。

> **说明** 在2021年03月22日之前创建，且在03月22日之后未进行变更的集群暂不开启计费。

集群流控试用档位的Token Server可供您继续测试使用，不会产生额外费用。试用档位单个应用QPS阈值之和不超过2000，接口总流量不超过3000。

> **说明** 试用档位仅供测试效果使用，不保证稳定性，请勿在生产环境使用。

前提条件

- 已开通AHAS专业版，若没有开通，请进入开通页面。
- Nginx Sidecar版本需在1.3.0及以上；Ingress需使用最新版本。

步骤一：选择档位创建集群

1. 登录AHAS控制台，然后在页面左上角选择地域。
2. 在控制台左侧导航栏中选择**流量防护 > Ingress/Nginx防护**。
3. 在**Ingress/Nginx防护**页面单击目标应用卡片。
4. 在左侧导航栏单击**集群流控 > 集群配置**。
5. 在**集群流控资源配置**区域内，选择**集群类型**为**试用**，单击**创建**，然后在对话框中单击**确认**。

   总配置量级即最大QPS，表示需要流控的接口所能承载的预估的最大QPS，代表可能到来的最大流量。

   > **说明** 实际流量（无论是否被流控）超出配置的最大QPS后，流控策略会退化到单机模式。

   创建集群后，系统会自动为该应用分配集群的Token Server。
6. （可选）单击**Token Client设置**区域**操作**列的**编辑**，设置**Token请求超时时间**，然后单击**确定**。

在某些场景下，集群流控Client与Token Server之间的网络通信时延较高，需要调整超时时间。

> **说明** Token请求超时时间单位为ms，取值范围为(0,10000]，一般不建议超过20 ms。公网环境网络延时较高，建议设置超时时长约为50 ms，但不建议超过80 ms。

步骤二：设置集群流控规则

1. 在目标应用管理页的左侧导航栏单击**规则管理**，选择目标方案页签，然后单击**新增流控规则**。
2. 在**新增流控防护规则**对话框，开启**是否集群流控**开关，并设置相关参数。

参数	描述	示例
接口名称	设置接口名称，可选择对应Nginx API分组名称。	default:localhost:8080
是否集群流控	开启此开关，即对集群内资源的调用总量进行限制。	开启
是否开启	开启此开关，规则生效；关闭此开关，规则不生效。	开启
集群阈值	表示该接口的限流阈值。	100
统计窗口时长	集群流控统计的时间窗口长度，取值范围为1秒~24小时。	1秒
失败退化策略	当出现连接失败、通信失败或Token Server不可用等情况时，流控规则可选择退化到单机限流模式或直接通过而忽略失败情况： ○ **退化到单机限流**：当出现通信失败的情况时，退化到设置的单机阈值来进行流控。需要在规则中配置单机退化阈值，代表单机的兜底阈值。 ○ **直接通过**：当出现通信失败的情况时，请求直接通过。	退化到单机限流
退化阈值自动调整	开启后会自动调整退化阈值，默认关闭。 **说明** 此功能要求SDK版本≥1.8.6。	关闭
退化单机阈值	代表单机的兜底阈值。当失败退化策略选择退化到单机限流时，需要设置此选项。	10

> **注意** Nginx集群流控默认采用批量模式请求Server以提升性能，其准确性根据流量分布的影响可能存在2%～6% 的偏差。

3. 单击**新增**，完成规则创建。

创建规则完成后，可以在**规则设置**页面查看到创建的集群流控规则，**阈值模式**为**集群总体**。

2.4. 请求分组管理

请求分组管理功能可以自定义分组名称和规则，将流量根据规则归类为不同的分组。此分组名称用于定义Nginx防护的流控对象，便于您设置流控规则。本文介绍请求分组的管理流程、自定义配置等内容。

前提条件

- 将Nginx接入流量防护
- 使用Ingress-sentinel实现流控

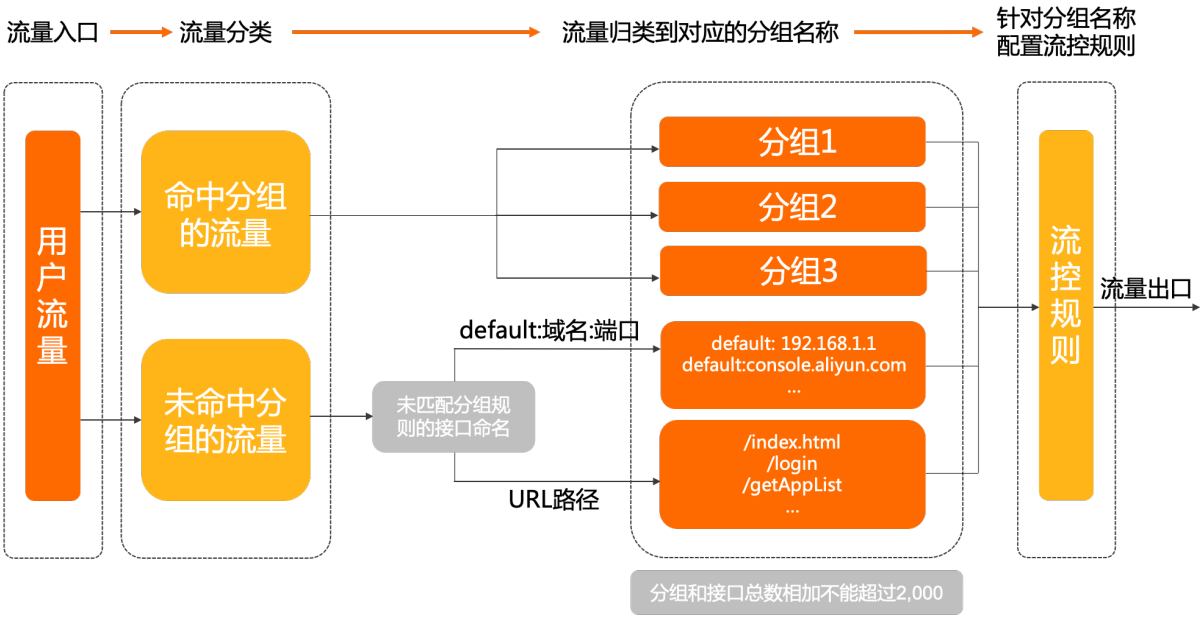
请求分组管理流程

Ingress/Nginx防护中的请求分组管理流程如下：

1. 注入流量：用户流量进入到系统中。
2. 流量分类：系统对流量进行分类，分为匹配分组规则的分组流量和不匹配分组规则的接口流量。
3. 流量归类到对应的分组名称：
   - 符合自定义分组规则的流量将归类为对应的分组名称。
   - 不符合自定义分组规则的流量，系统将自动按照域名与端口或URL路径的方式，将流量收敛并生成相应的接口名称。

   > **说明** 分组和接口总数相加不能超过2,000。

4. 配置流控规则：对不同的分组设置流控规则。



自定义请求分组

1. 登录AHAS控制台，然后在页面左上角选择地域。

2. 在控制台左侧导航栏选择**流量防护 > Ingress/Nginx防护**，然后在**Ingress/Nginx防护**页面，单击目标资源卡片。
3. 在左侧导航栏，单击**请求分组管理**。
4. 在**请求分组管理**页面，单击**新增请求分组**。
5. 在**新增自定义请求分组**对话框，设置相关参数，然后单击**新增**。

参数	描述	
请求分组名称	只能包含字母、数字以及特殊字符下划线（_）、短划线（-）、半角句号（.）、半角冒号（:），不超过1024个字符。	
域名和端口	**匹配模式**	○ **无限制**：对请求路径没有限制条件，系统对请求路径不做任何检查，直接通过。 ○ **精确匹配**：域名要与匹配串完全一致。例如设置精确匹配的匹配串为console.aliyun.com，则这条分组规则只对该域名下的请求有效。 ○ **后缀匹配**：只适用于域名和端口的匹配规则。例如设置后缀匹配的匹配串为aliyun.com，则这条分组规则对所有aliyun.com二级域名下的请求都有效，例如console.aliyun.com、img.aliyun.com、oss.aliyun.com等。 ○ **正则表达式**：指标准正则表达式，表示该分组规则对符合该正则表达式的请求有效。 例如以下正则表达式： ■ *^www**：表示所有以www开头的字符串。 ■ *[a-zA-Z0-9][-a-zA-Z0-9]{0,62}(\.[a-zA-Z0-9][-a-zA-Z0-9]{0,62})+\.?*：符合域名格式得到匹配，例如192.168.1.1 aliyun.com。
	匹配串	输入对应的匹配内容。

参数	描述	
URL路径	匹配模式	○ **无限制**：对请求路径没有限制条件，系统对请求路径不做任何检查，直接通过。 ○ **精确匹配**：请求路径要和匹配串完全一致，例如/img/test.jpg，只有完全匹配这个路径的流量才会命中规则。 ○ **前缀匹配**：只适用于URL路径的匹配规则。例如设置前缀匹配的匹配串为/api/，则这条分组规则对以/api/为开头的请求有效。例如/api/、/api/test1、/api/test2。 ○ **正则表达式**：指标准正则表达式，表示该分组规则对符合该正则表达式的请求有效。 例如以下正则表达式： ■ *.*(.jpg|.png)*：表示以所有.jpg和.png结尾的字符串。 ■ *Chapter[^1-5]*：表示除去Chapter1~Chapter5。例如该规则对Chapter6有效，但是对Chapter3无效。
	匹配串	输入对应的匹配内容，表示该分组规则对符合该匹配模式和匹配串的请求有效。
匹配客户端URL路径	对于大部分网关服务器如Nginx都提供了rewrite功能： ○ 关闭此开关，将对比rewrite之后的客户端URL路径是否符合上述规则。默认是关闭。 ○ 开启此开关，将会忽略网关服务器配置的rewrite规则，还是对比原始的客户端URL路径是否符合该条自定义分组规则。	

> **说明**
> 自定义请求分组规则优先级说明：
> 如果一个请求流量符合多条分组规则，系统默认选择最先符合的那一条规则，后续不再继续匹配。**请求分组管理**页面的分组名称已按照规则的优先级从高到低顺序排列。
> 分组规则的优先级如下：
>
> i. 总体优先级：规则方式>匹配模式>匹配串长度。
> ii. 规则方式：域名与端口>URL路径。
> iii. 匹配模式：精确匹配>后缀匹配或前缀匹配>正则表达式>无限制。
> iv. 匹配串长度：长的匹配串>短的匹配串。
> v. 如果是正则表达式，按照添加的时间，越后添加的权重越低。

设置未匹配分组规则的请求接口命名方式

若有流量未匹配到自定义的任一分组规则，系统会自动对其接口进行命名，设置命名生成方式的操作步骤如下：

1. 登录AHAS控制台，然后在页面左上角选择地域。
2. 在控制台左侧导航栏选择**流量防护 > Ingress/Nginx防护**，然后在**Ingress/Nginx防护**页面，单击目标资源卡片。
3. 在左侧导航栏，单击**请求分组管理**。
4. 在**请求分组管理**页面，选择**未匹配分组请求的接口名称生成方式**。
   - **default:域名和端口**（推荐）：用流量的域名和端口作为接口命名的方式。例如流量URL为http://demo.aliyun.com/api/login，则接口名称为default:demo.aliyun.com。这种方式的收敛度较高，推荐您使用这种收敛方式。
   - **URL路径**：用流量的URL路径作为接口命名的方式。例如流量URL为http://demo.aliyun.com/api/login，则接口名称为/api/login。这种命名方式发散性较高，不建议选择。特别是在流量遭到恶意攻击的时候，极易出现接口详情里接口过多难以管理的情况。
5. 在提示对话框中单击**确认修改**。

后续步骤

定义成功的分组名称会在**接口详情**页面展示，您可以对目标接口设置流控规则。具体操作，请参见配置流控规则。

2.5. Nginx Sentinel模块配置（新版）

本文介绍使用Nginx接入Sentinel流量防护的配置说明。

加载动态模块指令

- 语法：`load_module "/path/to/module.so";`
- 默认值： 无
- 配置上下文：`main`
- 是否必须配置：是

> **说明** 使用Nginx Sentinel模块前，您必须先使用`load_module`指令加载。此指令必须出现在主配置文件最开始的位置（即主配置`events`配置之前）。

示例配置如下：

```
load_module "/opt/nginx-sentinel-cpp-linux/ngx_sentinel_cpp_module.so";
```

应用配置指令

应用名称设置指令：ahas_app_name

- 语法：`ahas_app_name ahas_demo_app;`
- 默认值：无
- 配置上下文：`main`
- 是否必须配置：是

日志路径设置指令：sentinel_log_path

- 语法：`sentinel_log_path /opt/nginx-sentinel-cpp-linux;`
- 默认值：`~/logs/csp`
- 配置上下文：`main`
- 是否必须配置：否

> **说明** 该指令可以帮助您及时获取日志信息，建议您在接入Sentinel流量防护时选择配置该项。

命名空间设置指令：ahas_namespace

- 语法：`ahas_namespace default;`
- 默认值：`default`
- 配置上下文：`main`
- 是否必须配置：否

License设置指令：ahas_license

- 语法：`ahas_license <license>;`
- 默认值：无
- 配置上下文：`main`
- 是否必须配置：否

> **说明** 若您以公网方式接入Sentinel流量防护时需要配置该项。

区域设置指令：ahas_region_id

- 语法：`ahas_region_id cn-public;`
- 默认值：无
- 配置上下文：`main`
- 是否必须配置：否

> **说明** 正常情况下系统会自动生成`region-id`您无需自行设置，若需要手动指定您可以自行设置该项。

2.6. Nginx Sentinel模块配置（旧版）

本文介绍使用Nginx接入Sentinel流量防护的配置说明。

安装Sentinel Sidecar

下载AHAS Sentinel Sidecar，并解压到本地。

建议解压到 `/opt` 目录下，操作命令如下：

```
curl -L -O https://ahasoss-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com/sidecar/latest/ahas-sentinel-sidecar-linux.tar.gz
sudo tar -xzf ahas-sentinel-sidecar-linux.tar.gz -C /opt/
```

解压之后的安装目录为 `/opt/ahas-sentinel-sidecar-linux` ，Nginx Sentinel动态模块文件位于安装目录的 `lib/<os>-nginx-<version>/` 子目录下。

全局配置

- **加载动态模块load_module**
  - 语法： `load_module "/path/to/module.so";`
  - 默认值： 无
  - 配置上下文： `main`

  使用Nginx Sentinel模块前，必须先使用 `load_module` 指令加载之。此指令必须出现在主配置文件最开始的位置（主配置 `events` 配置之前）。
  以CentOS 7、Nginx-1.16.1、Sentinel Sidecar安装目录 `/opt/ahas-sentinel-sidecar-linux` 为例，示例配置如下。

  ```
  load_module "/opt/ahas-sentinel-sidecar-linux/lib/centos7-nginx-1.16.1/ngx_sentinel_module.so";
  ```

  > **说明** Nginx自身限制预编译动态模块与Nginx版本及编译配置绑定，升级Nginx版本或修改编译参数时必须同时更新预编译动态模块。

  如果Sentinel Sidecar安装包未包含您使用的Nginx版本，请提供您的操作系统版本和 `Nginx -V` （大写的V）的完整输出文本，以便为您提供预编译Nginx模块。
- **Nginx Sentinel模块初始化sentinel_init**
  - 语法： `sentinel_init sidecar unix:<path> | <address>:<port>;`
  - 默认值：无
  - 配置上下文： `main`

  使用 `sentinel_init` 指令配置Nginx Sentinel模块初始化参数。当前即配置Sentinel Sidecar的监听地址，示例配置如下。

  ```
  sentinel_init sidecar unix:/tmp/sentinel-sidecar.sock;
  ```
- **Sentinel Sidecar运行命令sentinel_sidecar_run**
  - 语法： `sentinel_sidecar_run "/path/to/sentinel-sidecar.sh" --app=<name> [--license=<license>] [--ns=<namespace>];`
  - 默认值：无
  - 配置上下文： `main`

  使用 `sentinel_sidecar_run` 指令配置Sentinel Sidecar运行命令。Sentinel Sidecar使Nginx工作进程在独立的子进程中运行。Nginx Sentinel模块与Sidecar异步通信实现流控功能。
  在阿里云VPC环境接入时，配置如下：
  - Sentinel Sidecar启动脚本路径，如 `/opt/ahas-sentinel-sidecar-linux/bin/sentinel-sidecar.sh` 。
  - AHAS应用名 `--app=<name>` 。这里配置的应用名即在AHAS控制台上查看和管理的应用名。

- 可选配置，AHAS环境（namespace） `--ns=<namespace>` 。

在公网环境接入时，还需要添加License配置：
公网接入License `--license=<license>` ，可登录AHAS控制台查询您的License，具体操作，请参见查看License。

> **注意** License是机密信息，请注意保密。

以阿里云VPC环境，Sentinel Sidecar安装目录 `/opt/ahas-sentinel-sidecar-linux` 为例，示例配置如下。

```
sentinel_sidecar_run "/opt/ahas-sentinel-sidecar-linux/bin/sentinel-sidecar.sh" --app=demo;
```

HTTP限流配置

限流行为sentinel_block_action

- 语法： `sentinel_block_action off | =code;`
- 默认值： `sentinel_block_action =429;`
- 配置上下文： `http` ， `server` ， `location`

Nginx添加Sentinel模块后，可使用 `sentinel_block_action` 指令配置限流行为。默认配置为 `sentinel_block_action =429;` ，即触发限流时Nginx立即返回 `HTTP 429` 错误码。 `sentinel_block_action` 指令可分别在Nginx配置文件的*http/server/location*配置上下文进行配置。

- 当客户端访问指定*location*下的URL请求时，如果该*location*未配置限流行为，则使用其所在server的配置。
- 如果*server*也未配置限流行为，则使用*HTTP*下的配置。
- 如果*HTTP*下也未配置，则使用内置默认配置，即返回 `HTTP 429` 错误码。

如Nginx同时作为静态资源服务器和HTTP反向代理服务器时，静态资源不需要限流，可在HTTP主配置关闭sentinel流控检查，在需要限流的location下再按需开启流控功能。

`sentinel_block_action` 指令支持如下配置项：

- `off` ：关闭Sentinel流控检查，即不使用Sentinel模块，直接放行所有请求。
- `=code` ：执行流控检查并执行限流拦截，触发限流时Nginx立即返回指定的HTTP错误码，拒绝处理此请求。

> **说明**
> - 若需要将关注的请求URL归类为业务API，请参见请求分组管理。
> - 若需要为业务API配置流控规则，请参见配置流控规则。

2.7. Ingress Sentinel ConfigMap配置

本文介绍使用Ingress接入Sentinel流量防护的ConfigMap配置说明。

指令名称	示例值	是否必须	说明
ahas-app-name	app_name	是	AHAS应用名称。
ahas-license	<license>	否	AHAS License，以公网方式接入Sentinel流量防护时需要配置该项。
ahas-region-id	cn-public	否	指定VPC地域，一般情况下无需配置。

指令名称	示例值	是否必须	说明
sentinel-block-action	429	否	流量被阻塞（Block）时指定的返回码，默认为429。

2.8. Ingress/Nginx插件版本说明

本文介绍Ingress/Nginx插件版本发布说明。

版本说明

版本号	发布时间	版本说明
1.0.1	2022年01月24日	• 支持Daemon On模式 • 支持控制台配置自定义返回码（全局维度）
1.0.0	2022年01月04日	使用原生SDK，大幅度提升性能，降低CPU损耗

版本号	发布时间	版本说明
1.3.0	2021年07月01日	• 支持集群流控 • 优化插件性能
1.2.0	2021年01月24日	• 优化日志大小 • 优化内存占用
1.0.1	2021年01月03日	• 支持流量请求分组 • 优化插件性能

接入方式（旧版Sidecar）

3.功能开关

3.1. 什么是功能开关

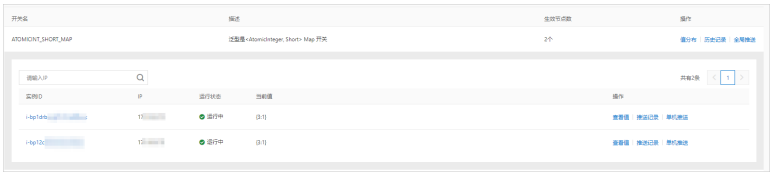
功能开关是一个轻量级的动态配置框架，通过功能开关可以动态管理代码中的配置项，根据需求为某个应用开启或关闭部分功能，或设置某个性能指标的阈值。功能开关通常用于设置黑白名单、运行时动态调整日志级别、降级业务功能等场景。

背景信息

通常业务代码中包含许多的配置项，这些配置项用于控制各种各样的业务逻辑，例如一个bool类型的变量控制某个功能是否开启，一个list控制访问白名单或黑名单，一个String控制提示信息。开发者通常希望可以动态、实时地去查看和修改配置项，并且期望不需要编写额外的代码来管理，此时就可以利用AHAS功能开关来实时修改和查看对应的配置项。与传统的配置中心不同，开发者使用AHAS功能开关时，无需关注配置项的解析逻辑，只需声明对应的变量，加上AHAS功能开关的注解即可在功能开关控制台对配置进行动态管理。

主要功能

- **查看功能开关**：在AHAS控制台功能开关中，可以直观查看应用中包括哪些开关，具体操作步骤，请参见查看功能开关。



- **查看开关值分布**：在AHAS控制台功能开关中，可以直观地查看对应开关值信息和分布信息，具体操作步骤，请参见查看功能开关。

- **设置开关推送**：在AHAS控制台功能开关中，设置开关的推送值，推送成功后，业务代码里会实时生效。具体操作步骤，请参见设置开关推送。

> **说明** AHAS功能开关还支持灰度分批推送，您可以先在一批机器验证后再全局发布，防止预期外的变更导致线上故障。

例如在大促到来的时候，可以通过开关将非核心的业务逻辑降级，减少一些非必要的资源消耗。操作流程可参考以下示例：

1. 在代码中增加核心业务开关、植入埋点和业务逻辑。
2. 在AHAS控制台**功能开关**中查看业务开关的信息和值分布。
3. 在AHAS控制台**功能开关**中将此开关的**推送值**设为true。

4. 在控制台上修改配置项，推送成功后，业务代码里会实时生效。代码中的此开关变量即变为 `true` 。即动态实时的通过功能开关控制业务逻辑。

常用场景

- **运行时动态调整日志级别**：在不同的应用场景下，您可能需要调整日志的级别，得到更有效的日志信息。功能开关提供了在应用运行时动态修改日志级别的功能。只要在应用中增加日志级别开关，然后在控制台中设置开关的推送值，即可快速的调整日志运行的级别，从而得到更有效的日志信息。具体操作步骤，请参见运行时动态调整日志级别。
- **主动降级业务功能**：通常一个业务功能包含许多的业务逻辑，其中可以区分出一些核心业务和非核心业务。在高并发的情况下，例如618、双十一等场景，为了提升系统性能，系统需要减少非必要业务的资源消耗，对非必要的业务功能进行主动降级。只要在应用中定义降级业务开关，然后在控制台设置开关推送，即可快速实现业务的降级。具体操作步骤，请参见主动降级业务功能。
- **黑白名单功能**：黑白名单是常用的访问控制规则，通过功能开关可以快速实现黑白名单功能。只要在应用中增加黑名单开关或白名单开关，然后在控制台设置开关推送即可。具体操作步骤，请参见快速实现黑白名单功能。

注意事项

在有些IDE中，尤其是使用Spring Boot技术栈的时候，SwitchManager和用户自己的代码使用的是不同的ClassLoader加载的，会导致功能开关在云端修改后，在用户的工程中由于不同ClassLoader的问题取不到最新修改的值。

接入指引

使用SDK接入

使用Spring Boot Starter接入

使用SDK接入

适用于Java语言的任何框架，只需简单定义、初始化后即可接入功能开关。

使用Spring Boot Starter接入

Spring Boot类应用专用接入方式，简单的接入流程。

3.2. 接入应用

3.2.1. 使用SDK接入

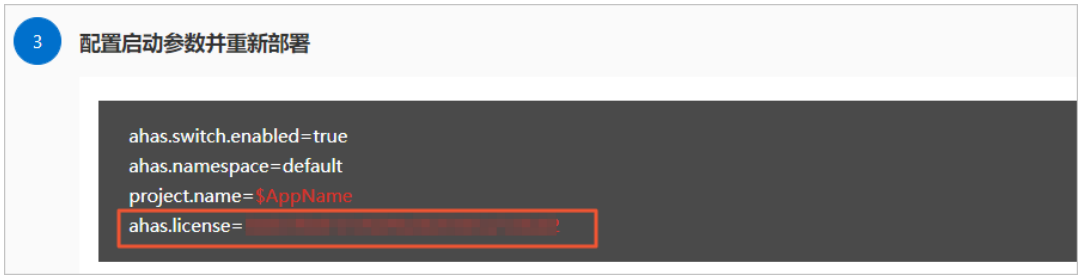
将应用接入功能开关后，即可使用功能开关的全部功能。本文将帮助您了解如何使用SDK方式接入应用。

操作步骤

1. 登录 AHAS控制台，然后在页面左上角选择**地域**。

2. 在左侧导航栏选择**功能开关**，然后在页面右上角单击**新应用接入**。
3. （可选）在**新应用接入**页面中查看并保存 License。

> **说明** License在页面的第三步描述中有显示，仅公网环境接入需要License。



4. 在Pom文件中加入以下依赖。

```
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>ahas-switch-client</artifactId>
    <version>x.y.z</version>
</dependency>
```

> **说明** 在**新应用接入**页面查看Pom依赖最新版本，将 `x.y.z` 替换为新版本的版本号。

5. 定义功能开关。
   i. 在字段上加上 `com.taobao.csp.switchcenter.annotation.AppSwitch` 注解，字段修饰符必须为 `public static` 。例如以下代码：

```
public class CommonTypeSwitch {
    @AppSwitch(des = "String 类型开关", level = Level.p2)
    public static String stringSwitch = "string";
    @AppSwitch(des = "Integer 类型开关", level = Level.p1)
    public static Integer integerSwitch = 2;
}
```

   ii. 初始化。

```
/*
    调用此方法完成初始化，请尽量确保在应用启动时调用。应用名参数可不填，默认取 project.name 启动参数作为appName参数。
    常量类参数是可变参数，支持同时注册多个常量类。如有需要，可按把不同功能模块的开关定义到不同常量类来做管控。
 */
SwitchManager.init("appName", CommonTypeSwitch.class);
```

6. 配置启动参数。
   - 非公网：

```
//将 AppName 替换为自定义的应用名称。
ahas.namespace=default
project.name=AppName
```

   - 公网：

```
//将 AppName 替换为自定义的应用名称，将 <license> 替换为真实值。
ahas.namespace=default
project.name=AppName
ahas.license=<license>
```

7. 重新部署您的应用。

执行结果

启动应用并调用配置埋点的方法。若该应用出现在AHAS控制台**功能开关**页面，则说明接入成功。

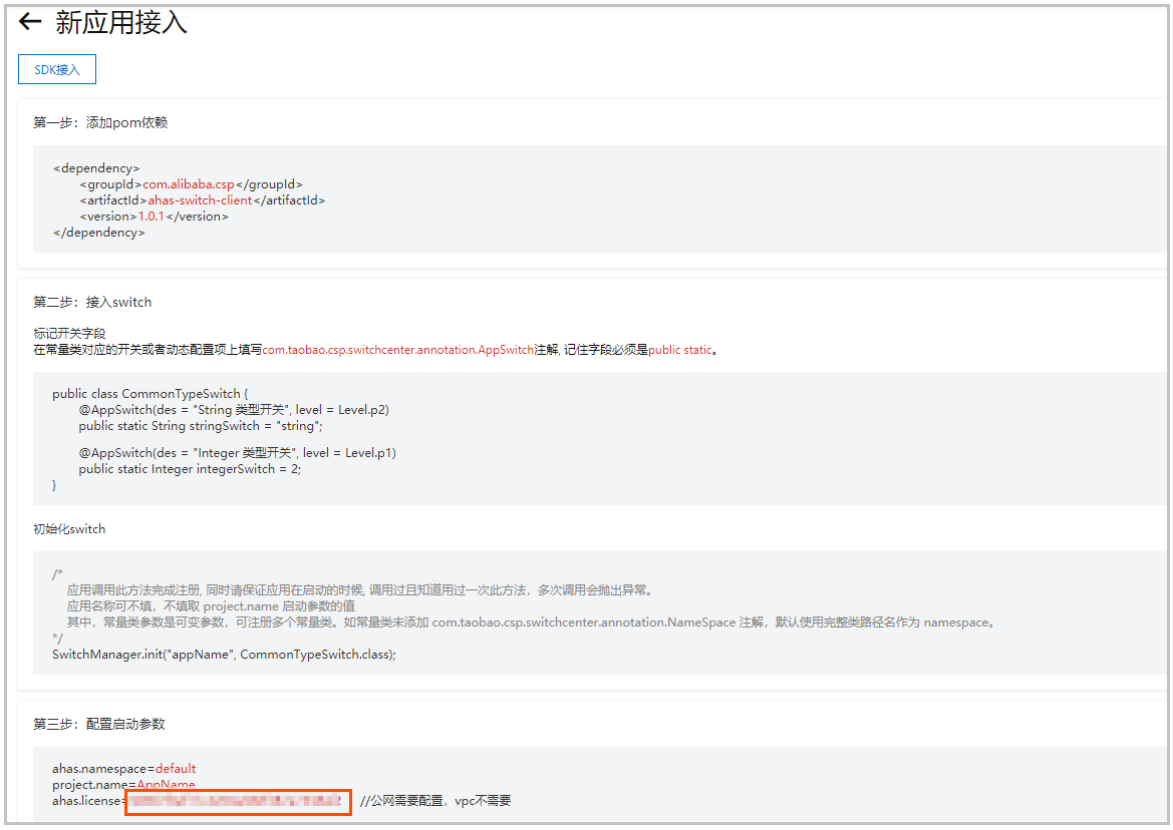
3.2.2. 使用Spring Boot Starter接入

将应用接入功能开关后，即可使用功能开关的全部功能。本文将帮助您了解如何使用Spring Boot Starter接入。

操作步骤

1. 登录 AHAS控制台，然后在页面左上角选择**地域**。
2. 在左侧导航栏选择**功能开关 > 开关管理**，然后在页面右上角单击**应用接入**。
3. （可选）在**新应用接入**页面查看并保存License。

> **说明** 仅公网环境接入需要License。



4. 在Pom文件中加入以下依赖。

```
<dependency>
  <groupId>com.alibaba.csp</groupId>
  <artifactId>spring-boot-starter-ahas-switch-client</artifactId>
  <version>x.y.z</version>
</dependency>
```

> **说明** 在 **新应用接入**页面查看Pom依赖最新版本，将 `x.y.z` 替换为新版本的版本号。

5. 在相关常量类上添加 `com.alibaba.csp.ahas.switchcenter.anotation.Switch` 注解，同时在对应字段上加 `com.taobao.csp.switchcenter.annotation.AppSwitch` 注解，字段修饰符必须为 `public static` 。
例如：

```
@Switch
public class SwitchConfig {
    @AppSwitch(des = "String 类型开关", level = Level.p2, callback = TestCallback.class)
    public static boolean test_switch = false;
}
```

6. 在 *application.properties* 中添加以下配置项。

非公网 公网

7. 重新部署您的应用。

执行结果

启动应用并调用配置埋点的方法。若该应用出现在AHAS控制台**功能开关 > 开关管理**页面，则说明接入成功。

3.3. 新增功能开关

一个业务通常由多个系统、多个功能模块组成，为保证某些业务的动态性，后端程序通常会用开关来控制程序的逻辑，以达到在系统运行时切换运行逻辑的目的。本文介绍如何新增功能开关。

前提条件

您已接入新应用，详情请参见使用SDK接入和使用Spring Boot Starter接入。

通过 Java SDK 接入

通过 Java SDK 接入的应用请参见以下步骤新增功能开关。

1. 定义功能开关。

在字段上加上 `com.taobao.csp.switchcenter.annotation.AppSwitch` 注解，字段修饰符必须为 `public static` 。
例如以下代码：

```
public class CommonTypeSwitch {
    @AppSwitch(des = "String 类型开关", level = Level.p2)
    public static String stringSwitch = "string";
    @AppSwitch(des = "Integer 类型开关", level = Level.p1)
    public static Integer integerSwitch = 2;
    @AppSwitch(des = "Boolean 类型开关", level = Level.p4)
    public static Boolean booleanSwitch = true;
    @AppSwitch(des = "AtomicInteger 类型开关", level = Level.p1)
    public static AtomicInteger atomicIntegerSwitch = new AtomicInteger(21);
    @AppSwitch(des = "AtomicBoolean 类型开关", level = Level.p1)
    public static AtomicBoolean atomicBooleanSwitch = new AtomicBoolean(true);
    @AppSwitch(des = "AtomicLong 类型开关", level = Level.p1)
    public static AtomicLong atomicLongSwitch = new AtomicLong(4L);
    @AppSwitch(des = "泛型为 String List 类型开关", level = Level.p1)
    public static List<String> stringListSwitch = new ArrayList<String>();
    @AppSwitch(des = "泛型是<Integer, String> Map 开关", level = Level.p4)
    public static Map<Integer, String> INT_STRING_MAP = new HashMap<Integer, String>();
    @SuppressWarnings("deprecation")
    @AppSwitch(des = "Date类型开关", level = Level.p1)
    public static Date dateTypeSwitch = new Date(114, 6, 3);
    @AppSwitch(des = "BigInteger类型开关", level = Level.p1)
    public static BigInteger bigIntegerTypeSwitch = BigInteger.valueOf(38888);
    @AppSwitch(des = "BigDecimal类型开关", level = Level.p1)
    public static BigDecimal bigDecimalTypeSwitch = BigDecimal.valueOf(3.00000000001);
    @AppSwitch(des = "枚举类型开关", level = Level.p1)
    public static EnumType enumTypeSwitch = EnumType.ITEM1;
    @AppSwitch(des = "泛型为List<Integer>的LinkedList", level = Level.p1)
    public static List<List<Integer>> LIST_INT_LINKEDLIST = new LinkedList<List<Integer>>();
    @AppSwitch(des = "泛型为<Map<String, Integer>, Map<String, Integer>>的HashMap", level = L
evel.p1)
    public static Map<Integer, Map<String, Map<String, Integer>>> MAP_MAP_HASHMAP = new Hash
Map<Integer, Map<String, Map<String, Integer>>>();
}
```

2. 调用注册方法进行注册。

```
/*
 应用调用此方法完成注册，同时请保证应用在启动的时候，调用过且知道用过一次此方法，多次调用会抛出异常。
 应用名称可不填，不填取 project.name 启动参数的值其中，常量类参数是可变参数，可注册多个常量类。
 如常量类未添加 com.taobao.csp.switchcenter.annotation.NameSpace 注解，默认使用完整类路径名作为 n
amespace。
*/
SwitchManager.init("appName", CommonTypeSwitch.class);
```

3. 配置启动参数。
   - 非公网

     ```
     //将 AppName 替换为自定义的应用名称。
     ahas.namespace=default
     project.name=AppName
     ```

   - 公网

```
//将 AppName 替换为自定义的应用名称，将 <license> 替换为真实值。
ahas.namespace=default
project.name=AppName
ahas.license=<license>
```

> **说明** 仅公网环境接入需要 License，您可在**新应用接入**页面查看并保存 License，详情请参见查看并保存 license。

4. 重新部署您的应用。

通过 Spring Boot 接入

通过 Spring Boot 接入的应用请参见以下步骤新增功能开关。

1. 定义功能开关。
   - 在相关开关类上加上 `@Switch` 注解。
   - 在相关常量类上加 `com.alibaba.csp.ahas.switchcenter.anotation.Switch` 注解，同时在对应字段上加 `com.taobao.csp.switchcenter.annotation.AppSwitch` 注解，字段修饰符必须为 `public static` 。

```
@Switch
public class SwitchConfig {
    @AppSwitch(des = "Boolean 类型开关", level = Level.p2, callback = TestCallback.class)
    public static boolean test_switch = false;
}
```

2. 配置启动参数。

   在 application.properties 中添加以下配置项。

   - 非公网

```
#指定您要接入的特定的 AHAS 环境。
ahas.namespace=default
#自定义您的应用名称。
project.name=AppName
```

   - 公网

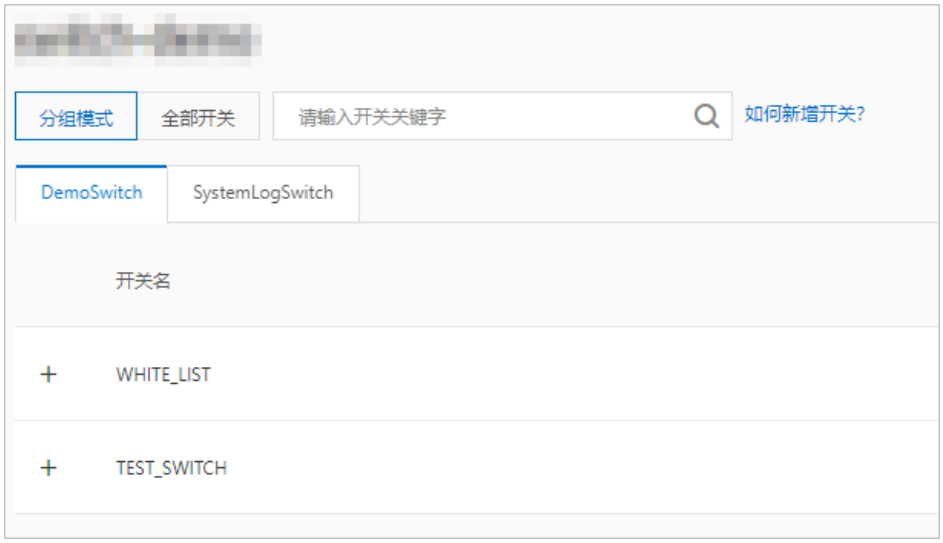
```
#指定您要接入的特定的 AHAS 环境。
ahas.namespace=default
#自定义您的应用名称。
project.name=AppName
#配置 License 信息。
ahas.license=<license>
```

> **说明** 仅公网环境接入需要 License，您可在**新应用接入**页面查看并保存 License，详情请参见查看并保存 license。

3. 重新启动您的应用。

执行结果

添加完成后，在**功能开关**页面单击目标应用的资源卡片，进入目标应用的**开关列表**页面，可查看到新增开关的相关信息。



更多信息

如果您需要自定义功能开关的分组，可在代码中添加 `com.taobao.csp.switchcenter.annotation.NameSpace` 注解；如果没有自定义，分组类别默认取 class 后面的类名，请参见以下示例。

```
package com.taobao.csp.switchcenter.example;
import com.taobao.csp.switchcenter.annotation.AppSwitch;
import com.taobao.csp.switchcenter.annotation.NameSpace;
import com.taobao.csp.switchcenter.bean.Switch.Level;
  @NameSpace(nameSpace = "customNamespace") //customNamespace为自定义分组名。
  public class PrimitiveTypeSwitch {       //PrimitiveTypeSwitch为默认分组名。
  @AppSwitch(des = "int 类型开关", level = Level.p1)
  public static int primitiveIntSwitch = 1;
  @AppSwitch(des = "doubel 类型开关", level = Level.p1)
  public static double primitiveDoubleSwitch = 1.121;
}
```

> **说明** 其中 `com.taobao.csp.switchcenter.bean.Switch.Level` 指开关的重要程度，分为p1、p2、p3、p4四个档位，p1的重要程度最高，p4的重要程度最低。

3.4. 管理功能开关

3.4.1. 查看功能开关

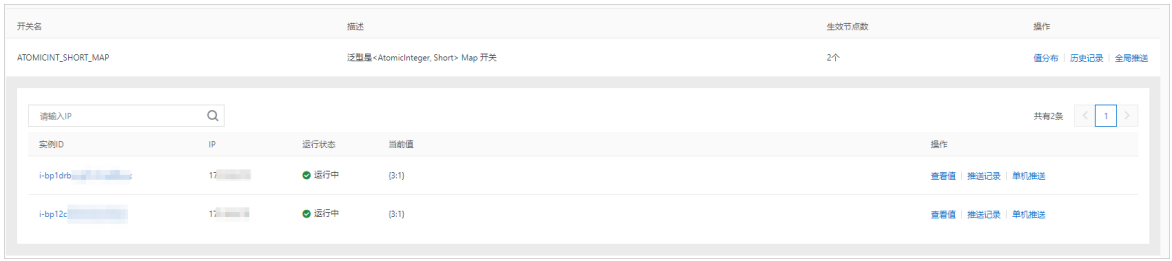
本文介绍如何查看功能开关的相关信息，包括开关类型、生效节点数、使用实例 ID 和 IP 等信息。

前提条件

您已成功新增功能开关，具体详情请参见新增功能开关。

操作步骤

1. 登录 AHAS 控制台，在页面左上角选择**地域**。
2. 在控制台左侧导航栏选择**功能开关**，在**应用列表**页面单击目标应用的资源卡片。进入目标应用的**开关列表**页面，查看开关的描述、生效节点数、使用的实例 ID、IP 等信息。



在**应用列表**页面，列表展现方式有**分组模式**和**全部开关**模式。

- **分组模式**：所有的开关按照 `NameSpace` 进行分组。
- **全部开关**模式：直接展示所有的开关。

3. 单击**操作**列的**值分布**，即可查看对应开关信息和分布信息，包括值编号、开关值等。

参数名	说明
开关名	自定义的开关名，详情可参见新增功能开关。
描述	开关的备注信息，记录开关的用途，即注解中引号中的内容。例如 `@AppSwitch(des = "String 类型开关")` 。
生效节点数	表示此开关生效的节点个数。
实例 ID	表示此开关被引用的实例 ID 节点。
运行状态	运行状态指的是节点的运行状态，包括**运行中**和**已停机**。
当前值	推送时输入的推送值。

4. 单击实例后**操作**列下的**查看值**，也可查看当前节点上此开关的值信息。

3.4.2. 设置开关推送

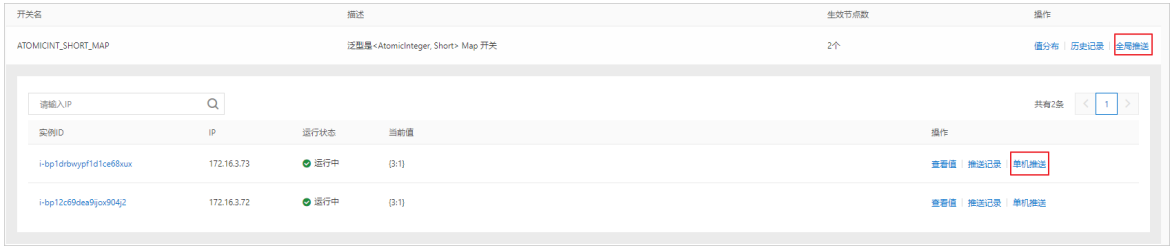
无需写死的URL、接口名、阈值和读取文件用的编码、黑白名单等，您可以直接使用功能开关设置推送值，快速创建运行时能覆盖的动态配置。功能开关支持全局推送、单机推送和灰度推送。本文介绍如何使用开关推送功能。

前提条件

您已成功新增功能开关，具体详情请参见新增功能开关。

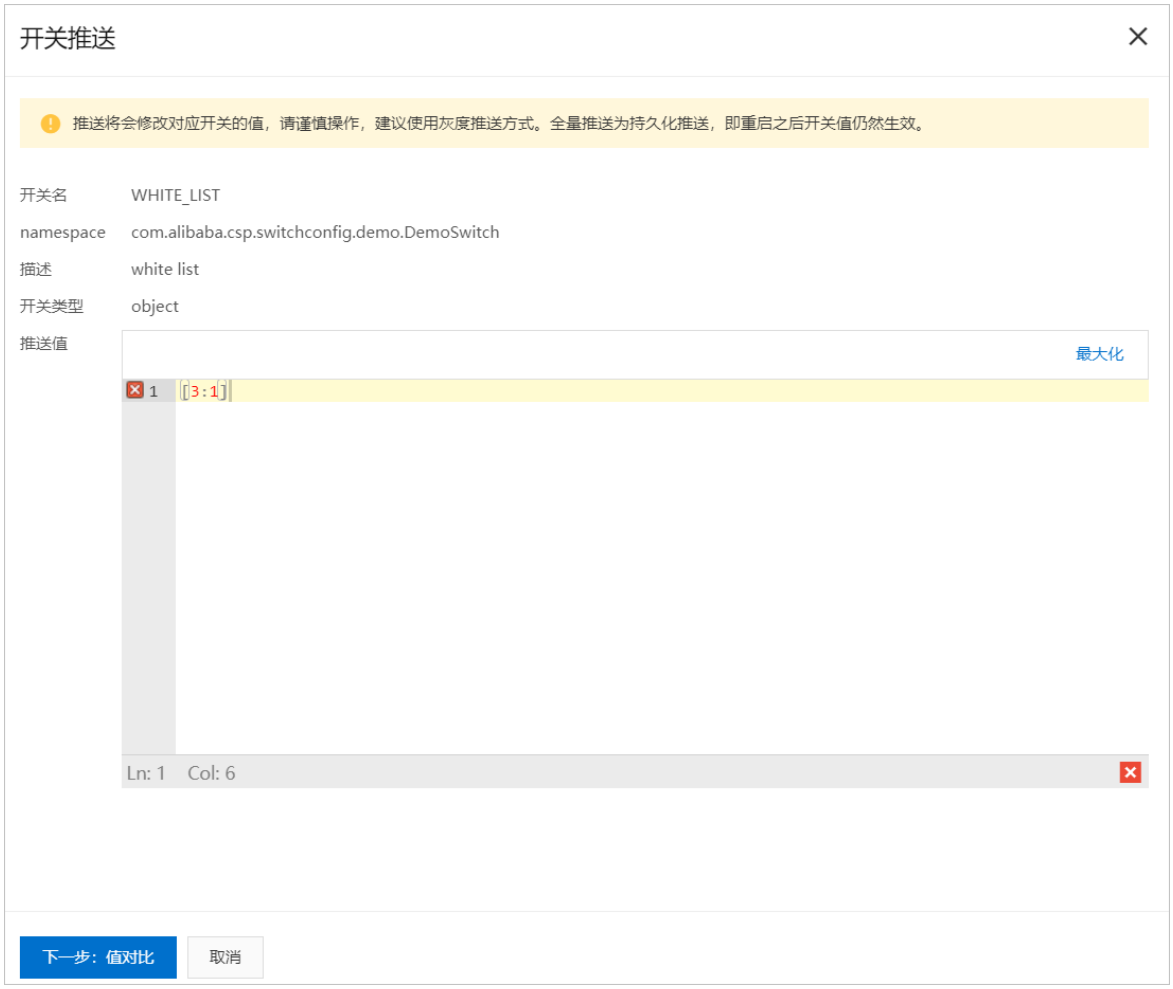
操作步骤

1. 登录AHAS控制台，然后在页面左上角选择**地域**。
2. 在控制台左侧导航栏选择**功能开关**，在**应用列表**页面单击目标应用的资源卡片。进入目标应用的**开关列表**页面。



3. 单击**开关列表**页面**操作**列的**全局推送**或**单机推送**，在右侧弹出**开关推送**页面，在此页面中可查看开关名、namespace、开关类型等信息，也可以编辑**推送值**。

> **说明** 开关的推送类型需在代码中定义，详情请参见变更回调。

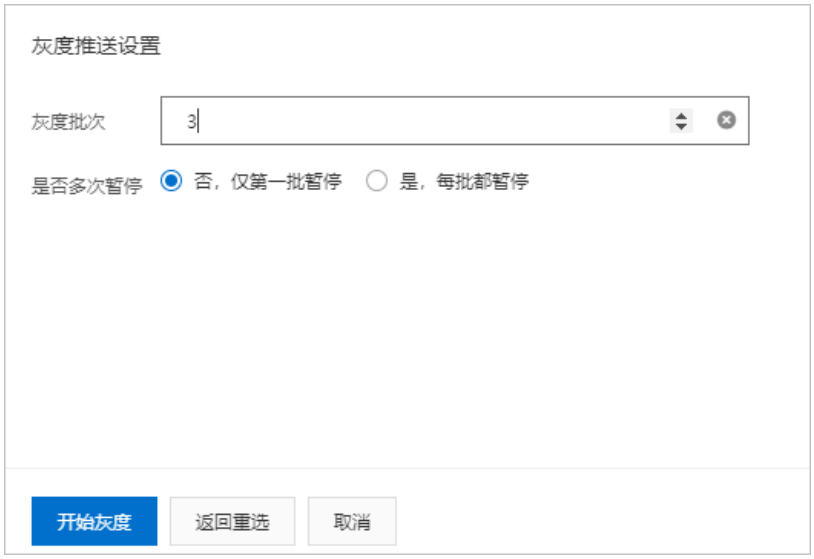


4. 编辑完成推送值后，单击**下一步：值对比**，会显示出修改点。若还需修改，则单击**上一步：返回修改**，若修改完成，则单击**单机推送**或**全局推送**
5. （可选）设置开关的灰度推送。
   i. 在目标开关的**操作**列单击**全局推送**，进入**开关推送**页面，在此页面中编辑推送值。
   ii. 单击左下角的**灰度推送**，弹出**灰度推送设置**页面。

iii. 设置**灰度批次**，选择**是否多次暂停**。然后单击**开始灰度**。

灰度推送即分批推送，可先推送一批机器试看推送效果，防止因全量推送而引起应用故障。

- **灰度批次**：指推送的批次数，范围为2至机器总数。每批的机器数为总机器数/批次数。按机器顺序推送，同一批次内推送机器并行，多批次间按顺序推送。例如有10台机器，**灰度批次**设为3，则先推送前3台机器，再推送3台机器，最后再推送4台机器。
- **是否多次暂停**：仅第一批暂停，表示推送完第一批机器数后暂停推送，待单击**继续推送**后，再继续推送。也可以设置为每批都暂停。



3.4.3. 历史记录

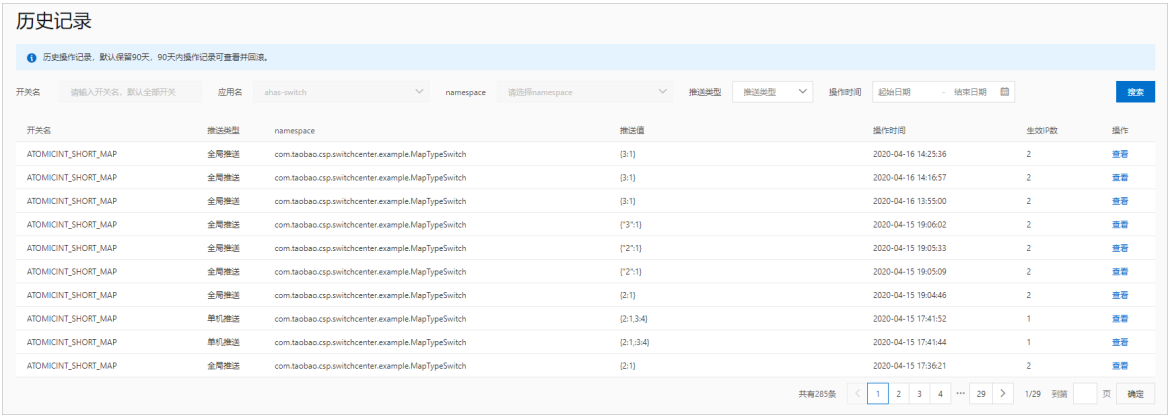
本文介绍如何查询功能开关推送的历史记录，包括推送的类型、操作时间等信息。

前提条件

您已成功新增功能开关，具体详情请参见新增功能开关。

操作步骤

1. 登录 AHAS 控制台，然后在页面左上角选择**地域**。
2. 在控制台左侧导航栏选择**功能开关**，在**应用列表**页面单击目标应用的资源卡片。进入目标应用的**开关列表**页面。
3. 在左侧导航栏选择**历史记录**。在**历史记录**页面展示了所有开关 90 天内的推送记录，包括**开关名**、**推送类型**等信息。
4. 按**开关名**、**推送类型**、**操作时间**等条件过滤。



参数名	说明
推送类型	包括全局推送和单机推送。 ○ 全局推送为持久化推送，即重启服务之后，开关仍然生效。 ○ 单机推送为内存化推送，重启之后将失效。
推送值	当前开关的推送值。
操作时间	默认保留90天的操作记录，90天内的操作记录可以查看。

5. 单击**操作**列下的**查看**，可以查看此条推送记录的开关名、namespace、推送类型、推送值和生效 IP。

3.5. Spring Config配置项

本文介绍如何对Spring Config的配置项执行修改或者持久化等操作。

前提条件

接入应用：

- Agent方式接入，与流量防护共用Agent，配置-Dahas.switch.agent.plugin.group.enabled=true。
- 应用配置SDK方式接入。具体操作，请参见使用SDK接入。
- 应用配置Spring Boot Starter方式接入。具体操作，请参见使用Spring Boot Starter接入。

背景信息

Spring Demo文件：

- UserController类对象：为@Value配置，用于测试@PostConstruct注解，此阶段可识别持久化值，部分内容如下。

```
@RestController
public class UserController {
    @Value("${project.name}")
    public String name;
    @Value("${user.ahas}")
    public boolean ahas;
    @Value("${user.number}")
    public int num;
    @Value("${destination}")
    public String destinationStr;
    @Autowired
    private Destination destination;
    @PostConstruct
    private void init(){
        System.out.println("[UserController] init()  value: "+ destinationStr +" , " + num +
" , "+ ahas + " , " + name);
        System.out.println("[UserController] init()  configuration: "+destination.getAvg()+"
, " + destination.getMax() + " , "+ destination.getMin());
    }
```

- DemoConfig类对象：为@Value配置，用于测试InitializingBean，afterPropertiesSet函数，初始化阶段可读取到持久化值，内容如下。

```
@Configuration
public class DemoConfig implements InitializingBean {
    @Autowired
    private RequestProperties requestProperties;
    @Override
    public void afterPropertiesSet() {
        System.out.println("[DemoConfig] init()  port: " + requestProperties.getPort() + " ,i
nterface: " + requestProperties.getInter());
    }
}
```

- RequestProperties类对象：为@ConfigurationProperties配置，Value模式。

```
@Component
@ConfigurationProperties(value = "request")
public class RequestProperties {
    private int port;
    private String inter;
    public int getPort() {
        return port;
    }
    public void setPort(int port) {
        this.port = port;
    }
    public String getInter() {
        return inter;
    }
    public void setInter(String inter) {
        this.inter = inter;
    }
}
```

- Destination类对象：为@ConfigurationProperties配置，Prefix模式。

```
@Component
@ConfigurationProperties(prefix = "property.destination")
public class Destination {
    private int max;
    private int min;
    private int avg;
    public int getMax() {
        return max;
    }
    public void setMax(int max) {
        this.max = max;
    }
    public int getMin() {
        return min;
    }
    public void setMin(int min) {
        this.min = min;
    }
    public int getAvg() {
        return avg;
    }
    public void setAvg(int avg) {
        this.avg = avg;
    }
}
```

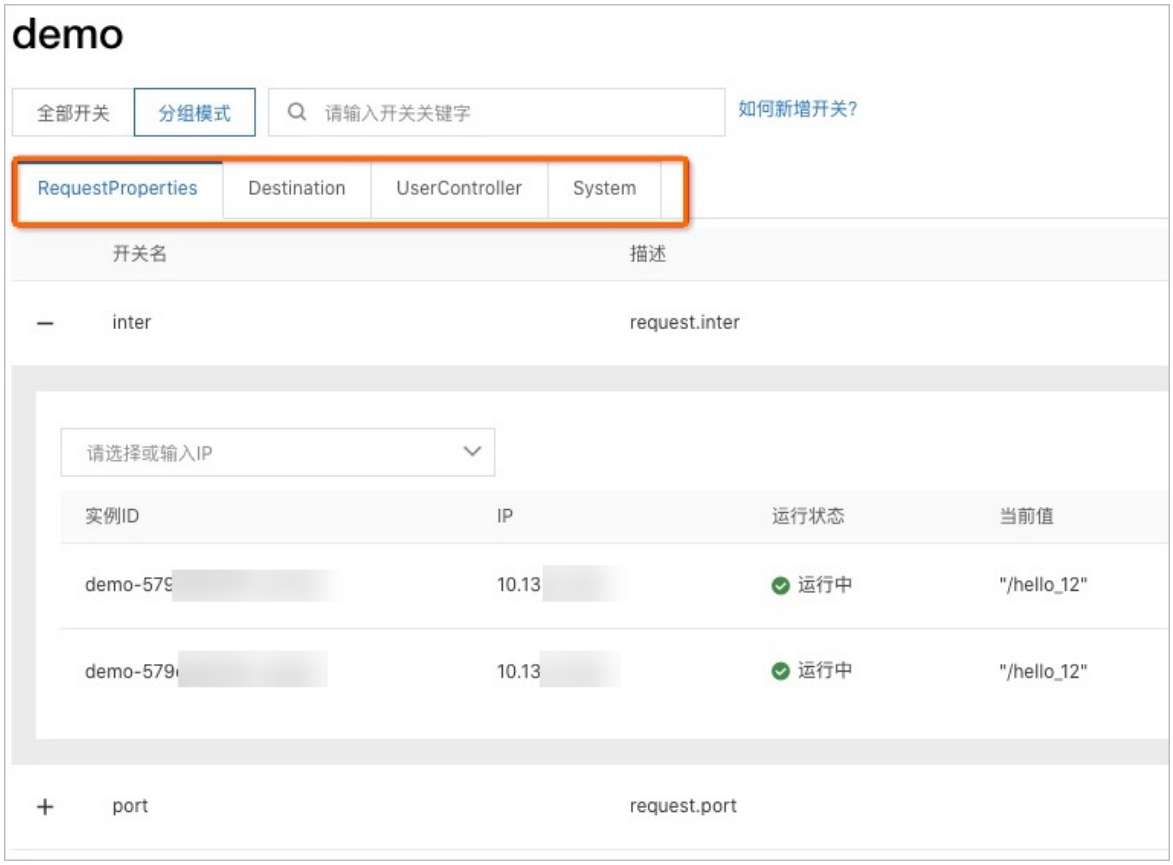
- application.properties配置内容如下。

```
user.ahas=false
user.number=123
request.port=8081
request.inter=/hello
destination=sun
property.destination.max=300
property.destination.min=10
property.destination.avg=100
```

查看应用的开关配置

1. 登录 AHAS控制台，然后在页面左上角选择**地域**。
2. 在左侧导航栏选择**功能开关**，在**应用列表**页面可看到已接入的应用。
3. 单击目标应用卡片，然后单击**分组模式**。

   如下图所示，可看到以分组形式展示的开关配置项。其中各分组名为识别出的类名，**开关名**为类中字段名，**描述**内容为@Value方式（注解中的内容）或者@ConfigurationProperties方式（Spring配置文件）的配置项，具体请参见背景信息中的介绍。



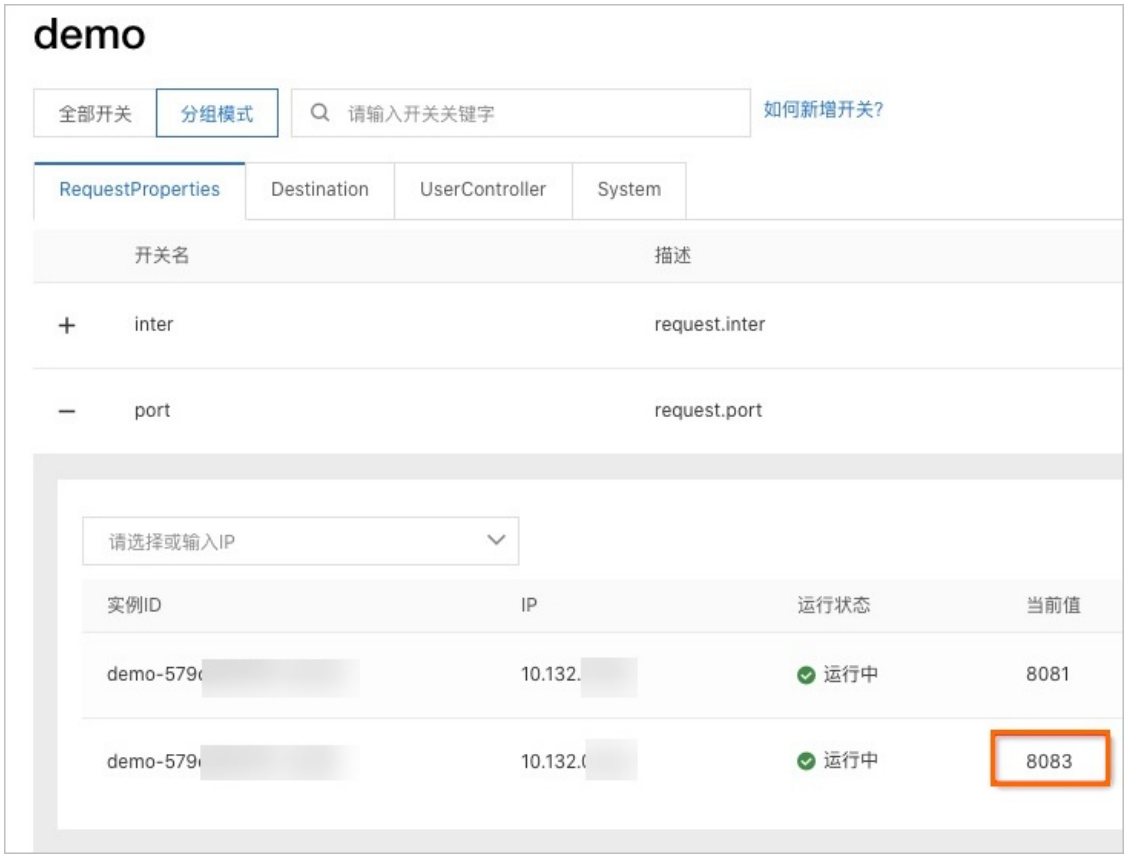
开关配置值的修改

对开关配置值的修改分为单机推送和全局推送两种方式。单机推送的值不会持久化，仅当前微服务实例生命周期有效；全局推送方式会进行值持久化，微服务实例再次启动后可看到持久化值。

单机推送

例如，对RequestProperties类对象中的port字段执行单机推送。

推送后应用中的deport字段的值已被修改，可在控制台中查看推送结果。

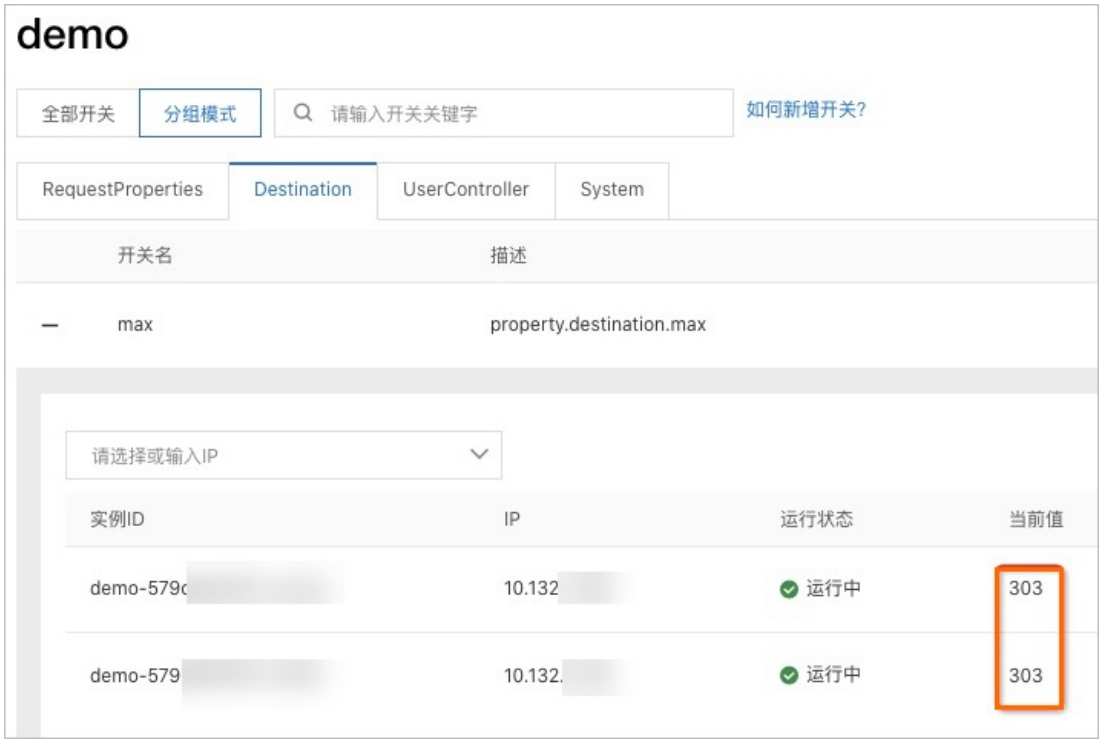


同样的方式，您还可以对UserController类对象中num的字段执行单机推送。您可在历史记录中查看历史推送操作。具体操作，请参见查看历史推送记录。

全局推送

例如，对Destination类对象中的max字段执行全局推送。

推送后应用中的max字段已被修改，同时数据已经持久化，可在控制台中查看推送结果。



同样的方式，您还可以对UserController类对象中的destinationStr字段执行全局推送。

开关配置值的持久化验证

初始化阶段

重启应用，在InitializingBean，afterPropertiesSet函数初始化阶段与@PostConstruct初始化阶段均可被读取到已持久化的值。

可通过测试Demo中的启动日志查看，内容如下：

```
[DemoConfig] init() ----------------->>>>>>>>>>>>>>>>>>> port: 8081 ,interface: /hello_switch
[UserController] init()  ----------------->>>>>>>>>>>>>>>>>>> value: stars , 123 , false , demo_mse
[UserController] init()  ----------------->>>>>>>>>>>>>>>>>>> configuration: 150 , 303 , 101
```

@ConfigurationProperties配置

在控制台可看到通过全局推送方式推送的RequestProperties类对象的inter字段的配置项为持久化值，而通过单机推送的port字段的配置项仍为Spring原始配置内容。

@Value配置

在控制台可看到通过全局推送方式推送的UserController类对象的destinationStr字段的配置项为持久化值，而通过单机推送的num字段的配置项仍为Spring原始配置内容。

查看历史推送记录

1. 登录 AHAS控制台，然后在页面左上角选择**地域**。
2. 在控制台左侧导航栏选择**功能开关**，在**应用列表**页面单击目标应用的资源卡片。进入目标应用的**开关列表**页面。
3. 在**开关列表**页面选择System页签，然后单击右上角的历史记录，可查看以往所有的推送记录。

日志动态调整

测试样例如下：

```
private static final Random RANDOM = new Random();
private static final Logger logger = LoggerFactory.getLogger("xx");
   @GetMapping("/hello")
   public ResponseEntity<String> hello(){
       int random = RANDOM.nextInt(100);
       logger.info("[SystemLog] ------------------- info : {} ",random);
       logger.debug("[SystemLog] ------------------- debug : {} ",random);
       logger.error("[SystemLog] ------------------- error : {} ",random);
       if(random < 30) {
           return new ResponseEntity<String>("BAD", HttpStatus.BAD_REQUEST);
       } else if(random > 50) {
           return new ResponseEntity<String>("BAD", HttpStatus.SERVICE_UNAVAILABLE);
       } else {
           return new ResponseEntity<String>("OK", HttpStatus.OK);
       }
   }
```

1. 设置日志级别。
   i. 在**开关列表**页面选择System页签，然后在搜索框中搜索 SYSTEM_LOG_CONFIG 开关，即日志级别开关。

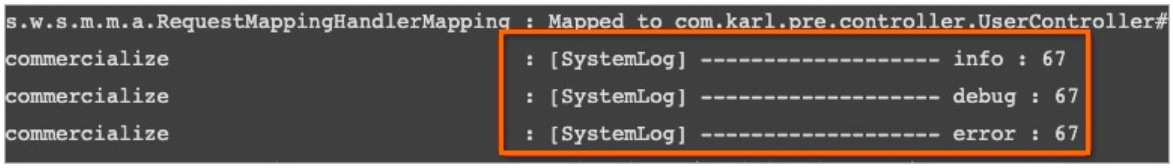
   ii. 单击**操作**列的**单击推送**或右上角的**全局推送**，按照 `<loggerName,loggerLevel>` 格式填写日志运行的配置，然后单击**全局推送**或**单机推送**。即可修改全部机器或是单台机器的日志运行级别。这里以全局方式为例。
   推送值格式：`Key` 为 `LoggerName`，`Value` 为日志级别。如需修改全局日志级别，`LoggerName` 为 `root`，如下所示。

```
{
 "root": "ERROR"
}
```

2. 查看日志打印情况。

```
[nio-8382-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherSe
[nio-8382-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
[nio-8382-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 23 ms
[nio-8382-exec-1] commercialize                            : [SystemLog] ------------------- info : 71
[nio-8382-exec-1] commercialize                            : [SystemLog] ------------------- error : 71
: key log4j.appender.logfile
ppender named "logfile".
[nio-8382-exec-2] commercialize                            : [SystemLog] ------------------- error : 44
[nio-8382-exec-4] commercialize                            : [SystemLog] ------------------- error : 44
[nio-8382-exec-3] commercialize                            : [SystemLog] ------------------- error : 32
[gPullingdefault] s.n.www.protocol.http.HttpURLConnection  : sun.net.www.MessageHeader@56e6b0f810 pairs: {null:
```

若将日志级别设为“DEBUG”，则日志打印结果如下：



灰度推送

您还可以在控制台设置开关的灰度推送。

1. 在目标开关的右上角单击**全局推送**，进入**开关推送**页面，在此页面中编辑推送值。
2. 单击左下角的**灰度推送**，在弹出的**灰度推送设置**页面，设置**灰度批次**，选择**是否多次暂停**。然后单击**开始灰度**。
   灰度推送成功后，在**开关推送**的**进度**列会显示已完成。

3.6. 变更回调

如果在某些场景中，您的程序中需要监听开关值的变更来做变更回调，功能开关客户端中提供了全局变更回调和单个开关变更回调两种方法。

全局变更回调

使用 Listener（完整包名：com.taobao.csp.switchcenter.core.Listener）接口即可监听任意开关的变化。

SDK 方式 | Spring Boot starter 方式

```
public class TestListener implements com.taobao.csp.switchcenter.core.Listener {
        @Override
        public void valueChange(String appName, String nameSpace, String name, String value) {
           //当 Field 值变更成功时，会调此方法。不要依赖 value 字段转型， 可直接依赖对应字段值。
        }
    }

    //注册 Listener
    SwitchManager.addListenner (new TestListener());
```

单个开关回调

实现 com.taobao.csp.switchcenter.core.SwitchCallback 接口。

```
public class TestCallback implements SwitchCallback {
    @Override
    public void excute(String nameSpace, String name, String value) {
        // TODO Auto-generated method stub
    }
}
```

在 `AppSwitch` 注解上填写 callback 字段。

```
@AppSwitch(des = "测试开关", level = Level.p1, callback = TestCallback.class)
public static Map<String, String> test_switch = new HashMap<String, String>();
class TestCallback implements SwitchCallback {
    public void excute(String nameSpace, String name, String value){
    }
}
```

4.权限管理

4.1. 访问控制概述

借助访问控制RAM的RAM用户，您可以实现权限分割的目的，按需为RAM用户赋予不同权限，并避免因暴露阿里云账号密钥造成的安全风险。

应用场景

以下是需用到访问控制RAM的典型场景。

- 借助RAM用户实现分权
  企业A的某个项目（Project-X）上云，购买了多种阿里云产品，例如：ECS实例、RDS实例、SLB实例、OSS存储空间等。项目里有多个员工需要操作这些云资源，由于每个员工的工作职责不同，需要的权限也不一样。企业A希望能够达到以下要求：
  - 出于安全或信任的考虑，A不希望将云账号密钥直接透露给员工，而希望能给员工创建独立账号。
  - 用户账号只能在授权的前提下操作资源。A随时可以撤销用户账号身上的权限，也可以随时删除其创建的用户账号。
  - 不需要对用户账号进行独立的计量计费，所有开销都由A来承担。

  针对以上需求，可以借助RAM的授权管理功能实现用户分权及资源统一管理。
- 借助RAM角色实现跨账号访问资源
  云账号A和云账号B分别代表不同的企业。A购买了多种云资源来开展业务，例如：ECS实例、RDS实例、SLB实例、OSS存储空间等。
  - 企业A希望能专注于业务系统，而将云资源运维、监控、管理等任务授权给企业B。
  - 企业B还可以进一步将A的资源访问权限分配给B的某一个或多个员工，B可以精细控制其员工对资源的操作权限。
  - 如果A和B的这种运维合同关系终止，A随时可以撤销对B的授权。

  针对以上需求，可以借助RAM角色实现跨账号授权及资源访问的控制。

权限策略

AHAS支持的系统权限策略为：

- AliyunAHASFullAccess：AHAS的完整权限。
- AliyunAHASReadOnlyAccess：AHAS的只读权限。

4.2. 为RAM用户设置AHAS服务权限

借助访问控制RAM的RAM用户（子账号），您可以实现权限分割的目的，按需为RAM用户赋予不同权限，并避免因暴露阿里云账号（主账号）密钥造成的安全风险。

背景信息

出于安全考虑，您可以为阿里云账号（主账号）创建RAM用户（子账号），并根据需要为这些RAM用户赋予不同的权限，这样就能在不暴露阿里云账号密钥的情况下，实现让RAM用户各司其职的目的。例如，假设企业A希望让部分员工处理日常运维工作，则企业A可以创建RAM用户，并为RAM用户赋予相应权限，此后员工即可使用这些RAM用户登录控制台。

AHAS权限说明

AHAS支持借助RAM用户实现分权，即为该RAM用户开启控制台登录权限，并按需授予以下权限。

- AliyunAHASFullAccess：AHAS的完整权限。该权限提供AHAS控制台所有操作的权限。具有该权限的RAM用户可以查看应用的监控数据，同时可以执行规则的创建、编辑、删除等操作。

- AliyunAHASReadOnlyAccess：AHAS的只读权限。该权限提供AHAS控制台界面的浏览、访问权限。具有该权限的RAM用户可以查看应用的监控信息、配置的防护规则以及限流触发产生的告警等。

> **说明** 具有AliyunAHASReadOnlyAccess权限的RAM用户无法在控制台进行防护规则的创建、编辑、删除等操作，如果需要对防护规则、行为管理等进行操作，请为RAM用户添加AliyunAHASFullAccess权限。

前提条件

- 开通RAM
- 开通AHAS
- 创建RAM用户

所有用户权限操作，请在阿里云账号环境下进行操作，RAM用户环境下无法为当前RAM用户添加任何权限，RAM用户请联系阿里云账号使用者帮助完成权限添加操作。

为RAM用户添加只读权限（AliyunAHASReadOnlyAccess）

如果希望RAM用户能够在AHAS控制台浏览监控等页面，在使用RAM用户之前，需要为其添加相应的只读权限，添加步骤如下：

1. 登录RAM控制台。
2. 在左侧导航栏中选择**身份管理 > 用户**。
3. 在**用户**页面中通过搜索框查找需要授权的用户，然后单击**操作**列的**添加权限**。
4. 在**添加权限**面板的**选择权限**区域，通过关键字搜索AliyunAHASReadOnlyAccess权限策略，并单击权限策略将其添加至右侧的**已选择**列表中，然后单击**确定**。

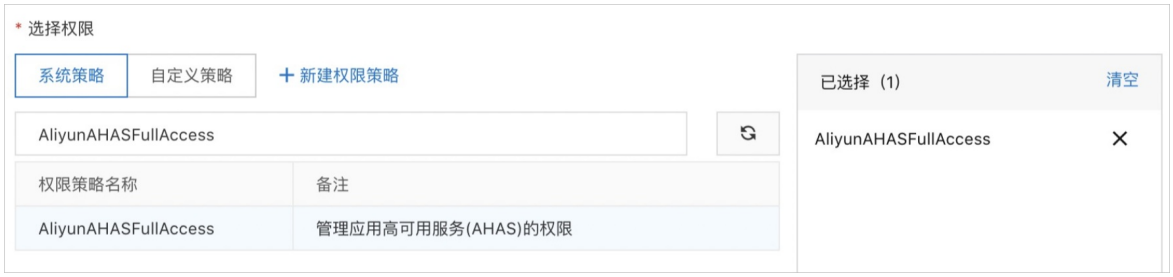
> **说明** 可添加的权限请参见AHAS权限说明。

5. 在**添加权限**的授权结果页面，查看授权信息摘要，并单击**完成**。

为RAM用户添加完整权限（AliyunAHASFullAccess）

如果希望RAM用户能够在AHAS控制台进行规则的创建、编辑等操作，在使用RAM用户之前，需要为其添加完整权限，添加步骤如下：

1. 登录RAM控制台。
2. 在左侧导航栏中选择**身份管理 > 用户**。
3. 在**用户**页面中通过搜索框查找需要授权的用户，然后单击**操作**列的**添加权限**。
4. 在**添加权限**面板的**选择权限**区域，通过关键字搜索AliyunAHASFullAccess权限策略，并单击权限策略将其添加至右侧的**已选择**列表中，然后单击**确定**。



> **说明** 可添加的权限请参见AHAS权限说明。

5. 在**添加权限**的授权结果页面，查看授权信息摘要，并单击**完成**。

> **说明** 如果您需要进行权限控制，但还未创建RAM用户，请先在您的阿里云账号下创建RAM用户。如何创建RAM用户的具体操作，请参见创建RAM用户。

后续步骤

使用阿里云账号创建好RAM用户后，即可将RAM用户的登录名称及密码或者AccessKey信息分发给其他用户。其他用户可以按照以下步骤使用RAM用户登录AHAS控制台。

1. 在浏览器中打开RAM用户登录入口。
2. 在**RAM用户登录**页面上，输入RAM用户登录名称，单击**下一步**，并输入RAM用户密码，然后单击**登录**。

> **说明** RAM用户登录名称的格式为*<$username>@<$AccountAlias>*或*<$username>@<$AccountAlias>.onaliyun.com*。*<$AccountAlias>*为账号别名，如果没有设置账号别名，则默认值为阿里云账号的ID。

3. 在RAM用户的**用户中心**页面上搜索并单击**应用高可用服务**或**AHAS**，即可访问应用高可用服务控制台。

> **说明** 若您目前使用的是RAM用户，请联系阿里云账号使用者为您添加相关权限。

相关文档

- 为RAM用户设置流量防护应用权限

4.3. 跨云账号授权

使用企业A的阿里云账号（主账号）创建RAM角色并为该角色授权，并将该角色赋予企业B，即可实现使用企业B的主账号或其RAM用户（子账号）访问企业A的阿里云资源的目的。

背景信息

假设企业A购买了多种云资源来开展业务，并需要授权企业B代为开展部分业务，则可以利用RAM角色来实现此目的。RAM角色是一种虚拟用户，没有确定的身份认证密钥，需要被一个受信的实体用户扮演才能正常使用。为了满足企业A的需求，可以按照以下流程操作：

1. 企业A创建RAM角色
2. 企业A为该RAM角色添加权限
3. 企业B创建RAM用户
4. 企业B为RAM用户添加AliyunSTSAssumeRoleAccess权限
5. 企业B的RAM用户通过控制台或API访问企业A的资源

可以为RAM角色添加的AHAS权限策略为：

- AliyunAHASFullAccess：AHAS的完整权限。
- AliyunAHASReadOnlyAccess：AHAS的只读权限。

步骤一：企业A创建RAM角色

首先需要使用企业A的阿里云账号登录RAM控制台并创建RAM角色。

1. 登录RAM控制台，在左侧导航栏中选择**身份管理 > 角色**，并在**角色**页面单击**创建角色**。
2. 在**创建角色**面板中执行以下操作并单击**关闭**。
   i. 在**当前可信实体类型**区域选择**阿里云账号**，并单击**下一步**。
   ii. 在**角色名称**文本框内输入RAM角色名称，在**选择信任的云账号**区域选择**其他云账号**，并在文本框内输入企业B的云账号，单击**完成**。

> **说明** RAM角色名称中允许使用英文字母、数字和短划线（-），长度不超过64个字符。

步骤二：企业A为该RAM角色添加权限

新创建的角色没有任何权限，因此企业A必须为该角色添加权限。

1. 在RAM控制台左侧导航栏中选择**身份管理 > 角色**。
2. 在**角色**页面单击目标角色**操作**列中的**添加权限**。
3. 在**添加权限**面板的**选择权限**区域，通过关键字搜索需要添加的权限策略，并单击权限策略将其添加至右侧的**已选择**列表中，然后单击**确定**。

> **说明** 可添加的权限参见背景信息部分。

4. 在**添加权限**的授权结果页面，查看授权信息摘要，并单击**完成**。

步骤三：企业B创建RAM用户

接下来要使用企业B的阿里云账号登录RAM控制台并创建RAM用户。

1. 登录RAM控制台，在左侧导航栏中选择**身份管理 > 用户**，并在**用户**页面单击**创建用户**。
2. 在**创建用户**页面的**用户账号信息**区域，输入**登录名称**和**显示名称**。

> **说明** 登录名称中允许使用小写英文字母、数字、英文句号（.）、下划线（_）和短划线（-），长度不超过128个字符。显示名称不可超过24个字符或汉字。

3. （可选）如需一次创建多个用户，则单击**添加用户**，并重复上一步。
4. 在**访问方式**区域，选中**控制台访问**或**OpenAPI调用访问**，并单击**确定**。创建的RAM用户显示在**用户**页面。

> **说明** 为提高安全性，请仅选中一种访问方式。

   - 如果选中**控制台访问**，则完成进一步设置，包括自动生成默认密码或自定义登录密码、登录时是否要求重置密码，以及是否开启MFA多因素认证。
   - 如果选中**OpenAPI调用访问**，则RAM会自动为RAM用户创建AccessKey（API访问密钥）。

> **注意** 出于安全考虑，RAM控制台只提供一次查看或下载AccessKeySecret的机会，即创建AccessKey时，因此请务必将AccessKeySecret记录到安全的地方。

步骤四：企业B为RAM用户添加权限

企业B必须为其阿里云账号下的RAM用户添加AliyunSTSAssumeRoleAccess权限，RAM用户才能扮演企业A创建的RAM角色。

1. 在RAM控制台左侧导航栏中选择**身份管理 > 用户**。
2. 在**用户**页面找到需要授权的用户，单击**操作**列中的**添加权限**。
3. 在**添加权限**面板的**选择权限**区域，通过关键字搜索AliyunSTSAssumeRoleAccess权限策略，并单击该权限策略将其添加至右侧的**已选择**列表中，然后单击**确定**。
4. 在**添加权限**的授权结果页面，查看授权信息摘要，并单击**完成**。

后续步骤

完成上述操作后，企业B的RAM用户即可按照以下步骤登录控制台访问企业A的云资源。操作步骤如下：

1. 在浏览器中打开RAM用户登录入口。
2. 在**RAM用户登录**页面上，输入RAM用户登录名称，单击**下一步**，并输入RAM用户密码，然后单击**登录**。

   > **说明** RAM用户登录名称的格式为 *<$username>@<$AccountAlias>* 或 *<$username>@<$AccountAlias>.onaliyun.com*。*<$AccountAlias>* 为账号别名，如果没有设置账号别名，则默认值为阿里云账号（主账号）的ID。

3. 在**子用户用户中心**页面上，将鼠标指针移到右上角头像，并在浮层中单击**切换身份**。
4. 在**阿里云 - 角色切换**页面，输入企业A的**企业别名**或**默认域名**，以及**角色名**，然后单击**切换**。
5. 对企业A的阿里云资源执行操作。

4.4. AHAS服务关联角色

本文介绍AHAS服务关联角色AliyunServiceRoleForAHAS以及如何删除该角色。

背景信息

AHAS服务关联角色AliyunServiceRoleForAHAS是AHAS在某些情况下，为了完成自身的某个功能，需要获取其他云服务的访问权限而提供的RAM角色。更多关于服务关联角色的信息请参见服务关联角色。

应用场景

AHAS架构感知的资源拓扑、流量防护等功能需要访问负载均衡SLB（Server Load Balancer）、专有网络VPC（Virtual Private Cloud）、云服务器ECS（Elastic Compute Service）等云服务的资源时，可通过自动创建的AHAS服务关联角色AliyunServiceRoleForAHAS获取访问权限。

权限说明

AHAS服务关联角色AliyunServiceRoleForAHAS具备的云服务的访问权限如下所示，更多权限说明请参见权限策略管理。

ECS的访问权限 ❯

SLB的访问权限 ❯
VPC的访问权限 ❯

删除AHAS服务关联角色

如果您需要删除AHAS服务关联角色AliyunServiceRoleForAHAS，请注意删除AliyunServiceRoleForAHAS后，会影响您AHAS数据的获取。删除AliyunServiceRoleForAHAS的操作步骤如下。

> **说明** 如果当前账号下还存在ACK集群，则需先删除这些集群，才能删除AliyunServiceRoleForAHAS，否则提示删除失败。

1. 登录RAM控制台，在左侧导航栏中单击**身份管理 > 角色**。
2. 在**角色**页面的搜索框中，输入**AliyunServiceRoleForAHAS**，自动搜索到名称为AliyunServiceRoleForAHAS的RAM角色。
3. 在右侧**操作**列，单击**删除**。
4. 在**删除角色**对话框，单击**确定**。

> **注意** 删除服务关联角色后会影响您AHAS数据的获取，请谨慎删除！

常见问题

问：为什么我的RAM用户无法自动创建AHAS服务关联角色AliyunServiceRoleForAHAS?

答：您需要拥有指定的权限才能自动创建或删除AliyunServiceRoleForAHAS。因此，在RAM用户无法自动创建AliyunServiceRoleForAHAS时，您需为其添加以下权限策略。

```
{
    "Statement": [
        {
            "Action": [
                "ram:CreateServiceLinkedRole"
            ],
            "Resource": "acs:ram:*:主账号ID:role/*",
            "Effect": "Allow",
            "Condition": {
                "StringEquals": {
                    "ram:ServiceName": [
                        "ahas.aliyuncs.com"
                    ]
                }
            }
        }
    ],
    "Version": "1"
}
```

> **说明** 请将 `主账号ID` 替换为您实际的阿里云账号（主账号）ID。

5.最佳实践

5.1. 流量防护最佳实践

5.1.1. PTS和AHAS共同保障应用稳定性

为提高应用高可用性，可以结合使用PTS与AHAS。首先使用PTS压测评估系统瓶颈，然后使用AHAS以系统瓶颈指标为阈值设置流控、降级、系统或隔离规则，保障系统稳定性。

背景信息

随着应用系统的频繁迭代，保障应用系统的稳定性越来越重要。主要原因如下：

- 随着应用上云的普及，单机架构向分布式架构演进，系统之间的依赖关系、调用链路变的十分复杂。
- 业务发展带来的服务端迭代越来越快，在性能管理上很难有足够的投入，经常会产生未知的隐患导致性能的大幅下降。

行业痛点：

- 对系统提供服务的能力不清楚，不知道如何进行压测，写脚本门槛太高。
- 压测工具维护很麻烦，压测流量不稳定，施压能力有限。
- 业务接口没有流量保护，瞬间流量超过上限就会压垮系统。
- 下游依赖服务不稳定，经常调用超时影响核心接口，影响系统稳定性。
- 非关键业务调用占用太多资源，核心业务的稳定性。

解决方案

借助于阿里巴巴内部多年高可用体系沉淀下来的经验，结合使用性能测试PTS和应用高可用服务AHAS，即可从压测、流量防护两个维度协助保障应用的稳定性。PTS是具备强大分布式压测能力的SaaS压测平台，可模拟海量用户的真实业务场景，全方位验证业务站点的性能、容量和稳定性。AHAS则以流量为切入点，从流量控制、熔断降级、热点防护和系统保护等多个维度来帮助保障服务的稳定性，同时提供秒级的流量监控分析功能。

产品优势

AHAS优势 ▼

经典案例

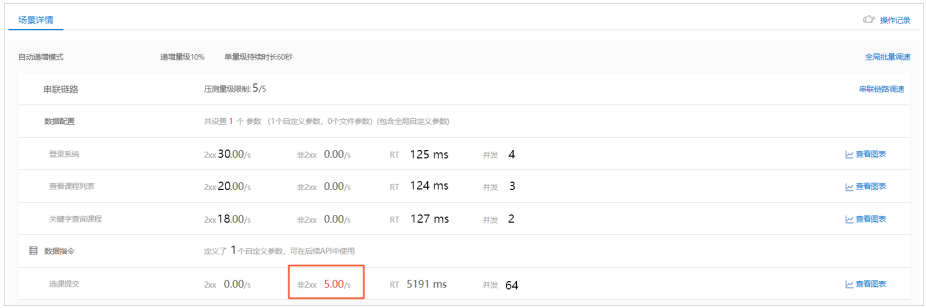
PTS和AHAS组成的压测流控方案，不仅在阿里内部淘宝、天猫等电商领域有着广泛的应用，在互联网金融、在线教育、游戏、直播行业和其他大型政央企行业也有着大量的实践。

在新冠疫情防护中，PTS和AHAS一体化的压测和防护为全国各地区的健康码提供了强有力的支撑。通过PTS平台对支付宝健康码服务进行压测，测出系统对外核心接口的服务能力以及相应的系统水位。同时使用AHAS根据压测结果中不同接口和系统指标配置限流、隔离以及系统保护等规则，在最短的时间内接入应用并持续提供防护，保障系统稳定性。

使用方法

1. 开通服务并购买资源包。
   - 开通PTS服务。
   - 开通AHAS。
2. 容量评估。

i. 使用PTS快速构建高仿真业务压测并发起压测，详情请参见如何在一分钟内发起压测？。
压测过程中可以在**场景详情**页签中查看各API的压测信息。例如本示例中**选课提交** API出现非2xx错误5/s。



ii. 在PTS控制台观察压测发起侧（客户侧）及服务侧（云监控）的端到端全监控，详情请参见查看监控详情，了解压测下的业务表现和各核心系统的性能水位情况。

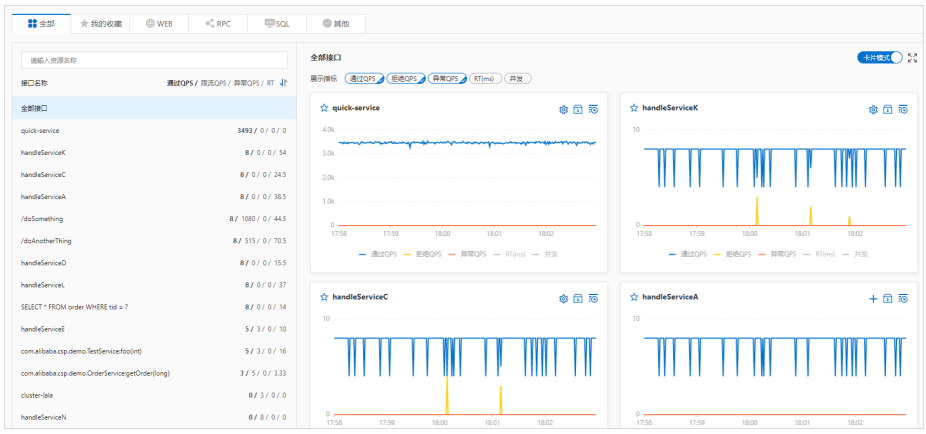
3. 设置流控、降级、系统和隔离规则。

i. 将应用接入AHAS，详情请参见接入应用方式。

ii. 根据所压测接口的RT响应情况、系统负载以及当前压测的请求量为接口配置流控、降级、系统和隔离规则。

- 配置流控规则：流控规则一般用于入口流量的防护，避免突发流量超过系统所能承受的最大值而压垮系统。
- 配置熔断规则：对于系统中存在的弱依赖服务，为了避免依赖的下游服务不可用拖累整个系统，可以在调用方设置降级规则。
- 自适应流控：为系统设置整体防护规则，对于CPU、Load等指标超过设定的阈值时触发系统限流，保障系统整体的可用性。
- 配置隔离规则：对于线程资源消耗比较多的接口，可以创建隔离规则，避免该接口在并发量大的情况下占用过多系统资源导致线程池满等问题。

4. 配置流量大盘，详情请参见创建流量大盘。
通过监控详情提供的多方位监控指标，动态调整接口的规则阈值并实时推送。



相关文档

- 关于AHAS应用防护的接入和使用，请参见开通AHAS、接入和规则配置。
- 关于PTS的开通和使用，请参见开通服务、创建压测场景和启动压测。

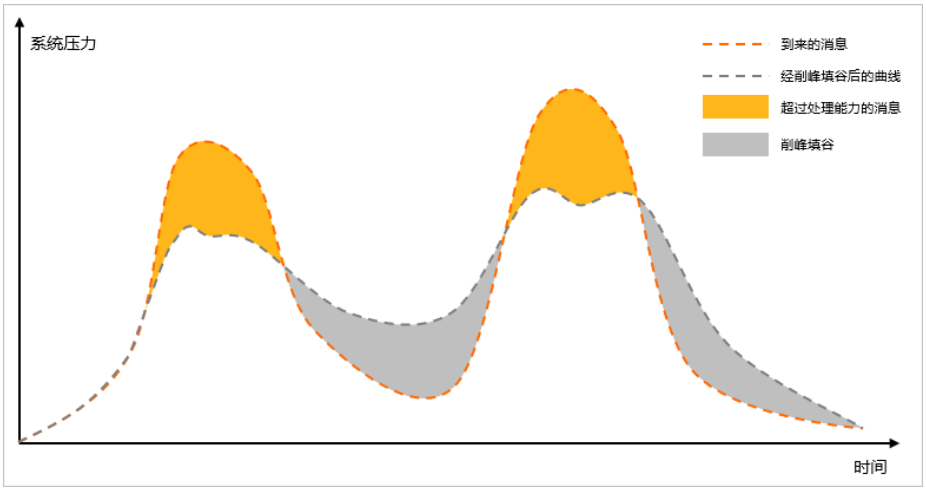
5.1.2. AHAS为消息队列RocketMQ版消费端削峰填谷

AHAS应用防护功能与消息队列RocketMQ版组合，可以让MQ消费端负载保持在消息处理水位之下，同时尽可能处理更多消息，达到削峰填谷的效果。本文以AHAS应用防护的匀速处理请求的能力为例，说明如何对RocketMQ消费端进行限流。

背景信息

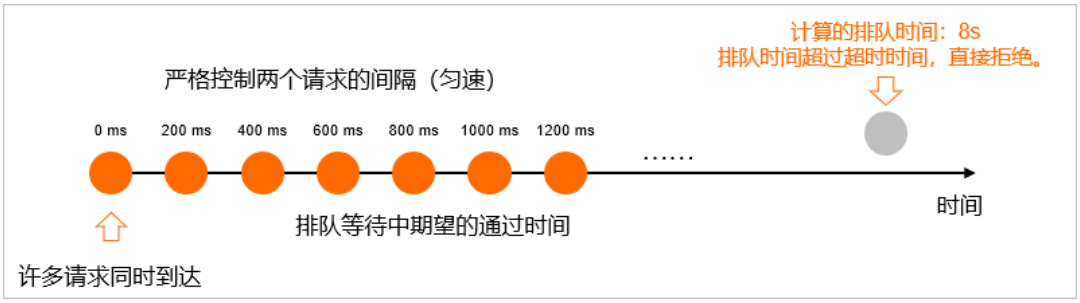
在消息队列RocketMQ版中，消费者消费消息时，很可能出现因消息发送量突增而消费者来不及处理的情况，导致消费方负载过高，进而导致影响系统稳定性。

在实际场景中，消息的到来具有瞬时性、不规律性，导致系统可能出现空闲资源。利用AHAS应用防护的匀速处理请求的能力，可以把超过消费端处理能力的消息（图中黄色部分）均摊到后面系统空闲时去处理，让系统负载处在一个稳定的水位，同时尽可能地处理更多消息，起到削峰填谷的作用。



AHAS应用防护在削峰填谷的场景时，以固定的间隔时间让请求通过，以稳定的速度逐步处理这些请求，避免流量突刺造成系统负载过高。同时堆积的请求将会排队，逐步进行处理；当请求排队预计超过最大超时时长的时候则直接拒绝，而不是拒绝全部请求。

例如，在RocketMQ的场景下，配置匀速模式下请求QPS为8，则每200 ms处理一条消息，多余的处理任务将排队；同时配置超时时间为8秒，预计的排队时长超过8秒的处理任务将会被直接拒绝。



前提条件

- 已在消息队列RocketMQ中发送和订阅消息，请参见消息队列RocketMQ版快速入门。
- 已开通AHAS，请参见开通AHAS。

步骤一：接入AHAS应用防护

下面将介绍如何快速在消息队列RocketMQ Consumer（消费端）接入和使用AHAS应用防护服务 。您可以下载Demo工程来完成以下步骤。

1. 在Consumer的pom文件中引入AHAS应用防护（即Sentinel）依赖。

```
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>ahas-sentinel-client</artifactId>
    <version>x.y.z</version>
</dependency>
```

> **说明** 请在AHAS依赖仓库查看依赖最新版本。

2. 定义资源。
   由于消息队列RocketMQ Consumer未提供相应拦截机制，而且每次收到都可能是批量的消息，因此用户需要在处理消息时手动定义资源。
   定义消息处理逻辑为消息被拒绝后会记录错误并触发重新投递，代码示例如下。

```
private static Action handleMessage(Message message, String groupId, String topic) {
        Entry entry = null;
        try {
            // 定义资源。为了便于标识，资源名称定义为Group ID和Topic的组合。Group ID和Topic可以通过消息队列RocketMQ控制台获得。
            entry = SphU.entry("handleMqMessage:" + groupId + ":" + topic);
            // 业务真实的消息处理逻辑。
            System.out.println(System.currentTimeMillis() + " | handling message: " + message);
            return Action.CommitMessage;
        } catch (BlockException ex) {
            // 编写被流控的消息的处理逻辑。示例：记录错误或进行重试。
            System.err.println("Blocked, will retry later: " + message);
            // 会触发消息重新投递
            return Action.ReconsumeLater;
        } finally {
            if (entry != null) {
                entry.exit();
            }
        }
    }
```

   消费者订阅消息的逻辑示例如下。

```
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe(topic, "*", (message, context) -> {
    return handleMessage(message);
});
consumer.start();
```

   关于消息队列RocketMQ版如何订阅消息，请参见消息队列RocketMQ版文档。

3. 登录AHAS控制台，获取AHAS启动参数。
   i. 在控制台最上方地域列表中，选择地域为**公网**。
   ii. 在左侧导航栏选择**流量防护 > 应用防护**，单击左上角**新应用接入**。
   iii. 选择**SDK接入 > 自定义埋点**页签，查看启动参数。
   启动参数示例如下。

```
-Dproject.name=MqConsumerDemo -Dahas.license=<License>
```

   其中 `MqConsumerDemo` 表示应用名，可自定义； `<License>` 表示您的授权证书，请修改为真实值。

4. 在Consumer中添加启动参数。

5. 启动Publisher开始发送消息，再启动Consumer开始接收消息。
   启动Publisher、Consumer后，本地IDE的**console**区域开始打印消息发送日志、消息接收日志，通过查看日志判断消息发送情况。

步骤二：配置削峰填谷规则

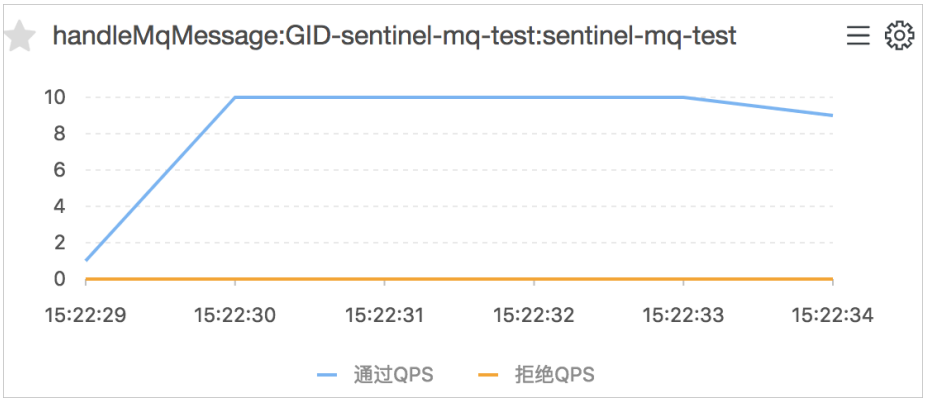
将应用接入AHAS应用防护后，需要为其配置规则来实现削峰填谷。

1. 登录AHAS控制台，在左侧导航栏选择**流量防护 > 应用防护**。
2. 在**应用防护**页面单击目标应用资源卡片，进入该应用管理界面。
3. 在左侧导航栏选择**应用概览**，在目标接口的**操作**列中，单击**流控**，填写流控规则，并单击**新建**完成创建。
   详情请参见配置流控规则。
   - **流控效果**：排队等待。
   - **单机QPS阈值**：QPS，设置QPS阈值为10，代表每100 ms匀速通过一个请求。
   - **超时时间**：2000 ms，超出此超时时间的请求将立即被拒绝，不会进入队列。
4. 通过消息队列RocketMQ Producer端向Consumer批量发送消息，查看流控效果。
   - 在Consumer控制台，通过观察消息头部的时间戳（如下所示），可以发现消息消费的速率是匀速的，大约每100毫秒消费一条消息。同时，不断有排队的处理任务完成，超出等待时长的处理请求直接被拒绝。

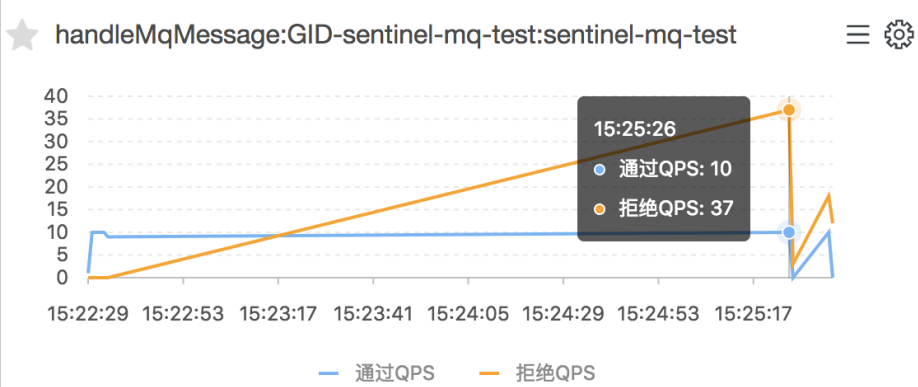
```
1550732955137 | handling message: Hello MQ 2453
1550732955236 | handling message: Hello MQ 9162
1550732955338 | handling message: Hello MQ 4944
1550732955438 | handling message: Hello MQ 5582
1550732955538 | handling message: Hello MQ 4493
1550732955637 | handling message: Hello MQ 3036
1550732955738 | handling message: Hello MQ 1381
1550732955834 | handling message: Hello MQ 1450
1550732955937 | handling message: Hello MQ 5871
```

> **说明** 在处理被拒绝请求的时候，需要根据业务需求，决定是否重新消费消息。

   - 在AHAS控制台的应用详情页面，单击**监控详情**，查看消息处理的监控曲线。



   如果没有使用匀速限流模式，该消息处理的监控曲线会如下图。
   如果不开启匀速模式，只会同时处理10条消息，其余的全部被拒绝。即使后面的时间系统资源充足，多余的请求也无法被处理，因而浪费了许多空闲资源。两种模式对比说明匀速模式下消息处理能力得到了更好的利用。



消息队列Kafka Consumer接入示例

与消息队列RocketMQ版类似，消息队列Kafka Consumer也可通过类似方法接入和使用AHAS应用防护服务。

示例：在处理消息的逻辑处定义资源。

```
private static void handleMessage(ConsumerRecord<String, String> record, String groupId, String
topic) {
    pool.submit(() -> {
        Entry entry = null;
        try {
            // 定义资源。为了便于标识，资源名称定义为Group ID和Topic的组合。Group ID和Topic可以通过MQ控
制台获得。
            entry = SphU.entry("handleKafkaMessage:" + groupId + ":" + topic);
            // 业务的消息处理逻辑。
            System.out.printf("[%d] Receive new messages: %s%n", System.currentTimeMillis(), rec
ord.toString());
        } catch (BlockException ex) {
            // Blocked
            // 在处理请求被拒绝的情况时候，需要根据业务需求，决定是否重新消费消息。
             System.err.println("Blocked: " + record.toString());
        } finally {
            if (entry != null) {
                entry.exit();
            }
        }
    });
}
```

示例：消费者订阅消息的逻辑如下。

```
while (true) {
    try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        // 必须在下次poll之前消费完这些数据，且总耗时不得超过SESSION_TIMEOUT_MS_CONFIG。
        // 建议开一个单独的线程池来消费消息，然后异步返回结果。
        for (ConsumerRecord<String, String> record : records) {
            handleMessage(record, groupId, topic);
        }
    } catch (Exception e) {
        try {
            Thread.sleep(1000);
        } catch (Throwable ignore) {
        }
        e.printStackTrace();
    }
}
```

相关文档

本文介绍的是AHAS应用防护服务的一个场景：请求匀速。AHAS应用防护服务还可以处理更复杂的各种情况。

- 流量控制：针对不同的调用关系，以不同的运行指标（如QPS、线程数、系统负载等）为基准，对资源调用进行流量控制，将随机的请求调整成合适的形状（请求匀速、Warm Up等）。
- 熔断降级：当调用链路中某个资源出现不稳定的情况，如平均响应时间增高、异常比例升高的时候，使对此资源的调用请求快速失败，避免影响其它的资源导致级联失败。
- 系统负载保护：从系统的维度提供保护。当系统负载较高的时候，提供保护机制，让系统的入口流量和系统的负载达到一个平衡，保证系统在能力范围之内处理最多的请求。

详情请参见AHAS应用防护。

5.1.3. 针对慢SQL的自动防护

慢SQL是比较致命的影响系统稳定性的因素之一。针对此类场景，AHAS应用防护提供了SQL级别的识别与防护，您可以根据监控详情为慢SQL配置流控降级规则保障系统的稳定性。本文介绍针对慢SQL如何设置应用防护。

背景信息

系统中出现慢SQL可能会导致CPU、负载异常和系统资源耗尽等情况。严重的慢SQL发生后可能会拖垮整个数据库，对线上业务产生阻断性的风险。线上生产环境出现慢SQL可能原因如下：

- 网络速度慢、内存不足、I/O吞吐量小、磁盘空间被占满等硬件原因。
- 没有索引或者索引失效。
- 系统数据过多。
- 在项目初期没有对SQL的性能做好考量。

使用AHAS应用防护识别慢SQL并为其配置流控降级规则的具体操作流程如下：

接入AHAS应用防护	>	查看监控	>	配置慢SQL防护规则

步骤一：接入AHAS应用防护

AHAS应用防护通过自动检测常见的DAO类、JDBC驱动类等自动识别应用中的SQL语句，您可以通过Java Agent或者Java SDK两种接入方式来实现对SQL的监控和拦截。

> **说明** 其中Java Agent接入方式目前支持MySQL JDBC和Oracle JDBC驱动，SDK接入方式目前支持MyBatis框架下的SQL识别。第三方组件和框架的版本支持情况详见支持组件列表。

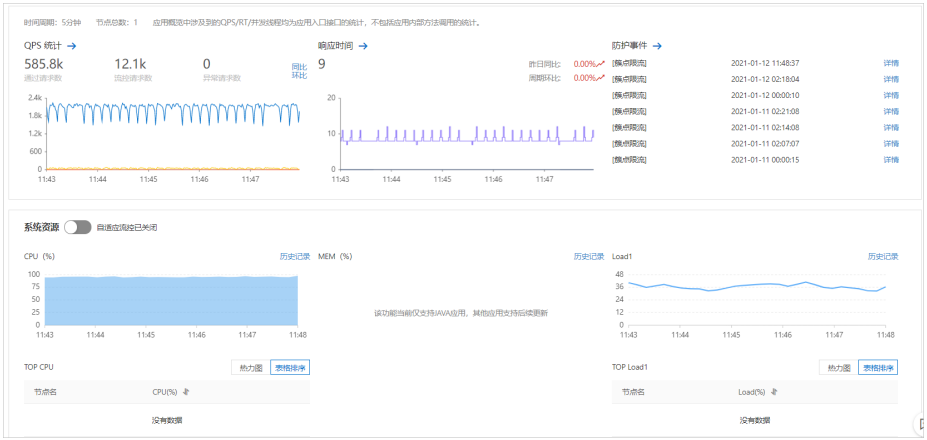
将应用接入AHAS请参见接入应用方式。

步骤二：查看监控

将应用接入AHAS应用防护服务后，您可以监控应用和资源API维度的实时数据（细化至秒级），从而评估系统的整体表现，并为流控降级规则的配置提供重要依据。具体监控指标包括QPS、响应时间、流控降级接口数等。

1. 登录AHAS控制台，在控制台左上角选择应用接入的地域。
2. 在控制台左侧导航栏中选择**流量防护 > 应用防护**。
3. 在**应用列表**页面单击目标应用的资源卡片。
4. 在**应用概览**页面查看应用的限流指标详情、QPS热力图等情况。

   应用概览



5. 在左侧导航栏单击**接口详情**，在**接口详情**页面查看每条SQL语句的调用及执行情况。

步骤三：配置慢SQL防护规则

根据AHAS自动识别的SQL语句，可以对出现慢SQL的应用配置线程数维度的流控或降级规则，当出现慢SQL调用时限制同一时刻执行的SQL数量，防止过多的慢SQL语句执行把资源耗尽。

流控规则

针对慢SQL防护的流控规则的统计维度有当前接口和关联接口，具体配置，请参见配置流控规则。

- **当前接口**：在该模式下，阈值配置为当前SQL资源，超过设置的值后多余的请求将被拒绝。

  当前接口

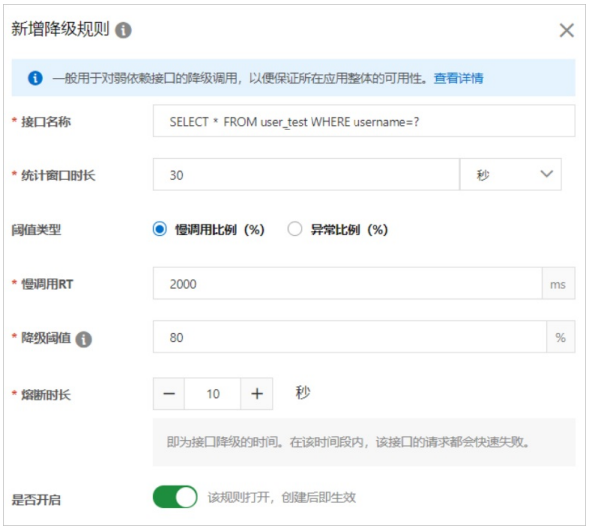
- **关联接口**：阈值配置为关联SQL资源，关联的资源请求超过设置的值之后，该资源的调用将被拦截。

关联接口

降级规则

慢SQL防护的还可以使用降级规则，根据SQL执行的RT设置对应的阈值以及时间窗口，超过指定的RT值后在时间窗口内SQL执行将被降级，抛出包装好的异常。具体操作，请参见配置熔断规则。

降级规则



资源被流控降级后会报`BlockException`类异常（限流会抛流控异常FlowException，降级会抛出降级异常DegradeException），您可以根据异常信息进行后续业务处理。

5.1.4. 为应用配置水平弹性伸缩

只要容器服务Kubernetes版中的Java应用接入了AHAS应用防护组件后，您的应用实例就可以自动根据AHAS应用防护收集的指标（如QPS、平均响应时间等）进行弹性伸缩，系统可以自动根据实时的流量情况进行扩缩容，保证系统的可用性。本文介绍如何为接入了AHAS应用防护，且在容器服务Kubernetes版中部署的应用配置水平弹性伸缩。

前提条件

- 创建Kubernetes托管版集群
- 开通AHAS

背景信息

弹性伸缩可以从调度层弹性，主要是负责修改负载的调度容量变化。例如，容器水平伸缩HPA（Horizontal Pod Autoscaling）是典型的调度层弹性组件，通过HPA可以调整应用的副本数，调整的副本数会改变当前负载占用的调度容量，从而实现调度层的伸缩。弹性伸缩是阿里云容器服务Kubernetes版ACK（Alibaba Cloud Container Service for Kubernetes）上被广泛采用的功能，更多信息，请参见弹性伸缩概述。

步骤一：安装alibaba-cloud-metrics-adapter

您可以通过alibaba-cloud-metrics-adapter使用云指标进行Pod的水平伸缩（HPA）。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用市场**。
3.
4. 在**创建**面板中，选择集群和命名空间，然后单击**下一步**。

   命名空间和**发布名称**为默认值，无需设置。
5. 在**参数配置**页面，设置相应参数，然后单击**确定**。

参数	描述
AccessKeyId	您的阿里云AccessKeyId。
AccessKeySecret	您的阿里云AccessKeySecret。
Region	您集群所在的地域，例如cn-qingdao、ap-southeast-1。

> **说明** 如果您的集群和专有网络VPC之间有专线，专线会被自动使用。

步骤二：安装ack-ahas-sentinel-pilot

ack-ahas-sentinel-pilot是AHAS应用防护组件，只要为部署在ACK中的Java应用安装AHAS应用防护组件后，您无需修改任何代码，就能借助AHAS对Java应用进行全方位系统防护，针对性的对系统进行流量管控、服务降级等操作。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用市场**。
3. 在**应用市场**页面单击**应用目录**页签，然后搜索并选中**ack-ahas-sentinel-pilot**。
4. 在右侧**创建**区域，选择目标集群，然后单击**创建**。

步骤三：为Java应用开启AHAS

1. 在容器服务管理控制台左侧导航栏单击**集群**。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 为Java应用开启AHAS。

   以下步骤分别对应创建新应用和已有应用这两种情况：
   - 如需在创建新应用的同时开启AHAS应用防护，请按以下步骤操作。
     a. 在**无状态**页面，单击右上角的**使用模板创建**。
     b. 选择**示例模板**，并在模板中将以下 `annotations` 添加到 `spec>template>metadata` 层级下。

```
annotations:
  #是否开启AHAS应用防护插件，on、true表示开启，off、false表示关闭。
  ahasPilotAutoEnable: "on"
  #AHAS应用名。
  ahasAppName: "<your-deployment-name>"
  #AHAS license。公网环境需配置AHAS license，VPC环境无需配置，AHAS license可以在AHAS控制台>应用防护>新应用接入中获取。
  #ahasLicenseKey: "<your-license>"
```

完整YAML示例模板如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: agent-foo-on-pilot
  labels:
    name: agent-foo-on-pilot
spec:
  replicas: 1
  selector:
    matchLabels:
      name: agent-foo-on-pilot
  template:
    metadata:
      labels:
        name: agent-foo-on-pilot
      annotations:
        ahasAppName: "foo-service-on-pilot"
        ahasNamespace: "default"
    spec:
      containers:
        - name: master
          image: registry.cn-hangzhou.aliyuncs.com/sentinel-docker-repo/foo-service:l
atest
          imagePullPolicy: Always
          ports:
            - containerPort: 8700
          volumeMounts:
            - name: foo-service-logs
              mountPath: /foo-service/logs
            - name: foo-service-config
              mountPath: /foo-service/config
          resources:
            limits:
              cpu: "0.5"
              memory: 500Mi
            requests:
              cpu: "0.5"
              memory: 500Mi
      volumes:
        - name: foo-service-logs
          emptyDir: {}
        - name: foo-service-config
          configMap:
            name: foo-service-cm-pilot
            items:
              - key: application.yaml
                path: application.yml
```

- 如需为现有应用开启AHAS应用防护，请按以下步骤操作。
  a. 单击**无状态**或**有状态**页签，然后在目标应用右侧**操作**列中选择**更多 > 查看YAML**。

b. 在**编辑YAML**对话框中将以下 `annotations` 添加到 `spec>template>metadata` 层级下，并单击**更新**。

```
annotations:
  ahasPilotAutoEnable: "on"
  ahasAppName: "<your-deployment-name>"
  #ahas namespace，命名空间，默认default。
  ahasNamespace: "default"
  #ahas license，公网需要。
  #ahasLicenseKey: "<your-license>"
```

> **说明** 请将 `<your-deployment-name>` 替换为您的应用名称。

部署成功后，登录AHAS控制台，在左侧导航栏选择**流量防护 > 应用防护**，在顶部菜单栏选择对应的地域和环境，可以看到该应用。

5. （可选）在集群管理页左侧导航栏中，选择**服务与路由 > 服务**，获取目标服务的外部端点IP。

   您可以通过SLB对外暴露的外部端点IP来访问该服务。

步骤四：创建HPA配置

本文针对示例Java应用来手动创建HPA，通过kubectl命令实现容器自动伸缩配置。

> **说明**
> AHAS应用防护目前支持四种判断指标：
> - ahas_sentinel_total_qps：应用的总QPS（pass+block，平均到每台机器）。
> - ahas_sentinel_pass_qps：应用的通过QPS（平均到每台机器）。
> - ahas_sentinel_block_qps：应用的拒绝QPS（平均到每台机器）。
> - ahas_sentinel_avg_rt：应用的平均响应时间。

1. 使用kubectl连接集群。

   关于kubectl的具体操作，请参见通过kubectl工具连接集群。

2. 创建配置文件*ahas-sentinel-hpa.yml*。

   示例：

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: ahas-sentinel-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta2
    kind: Deployment
    name: agent-foo-on-pilot
  minReplicas: 1
  maxReplicas: 3
  metrics:
    - type: External
      external:
        metric:
          name: ahas_sentinel_total_qps
          selector:
            matchLabels:
              # If you're using AHAS Sentinel pilot, then the appName and namespace
              # can be retrieved from the annotation of target Deployment automatically.
              # ahas.sentinel.app: "foo-service-on-pilot"
              # ahas.sentinel.namespace: "default"
        target:
          type: Value
          # ahas_sentinel_total_qps > 30
          value: 30
```

上述的配置针对agent-foo-on-pilot Deployment对应的应用进行自动弹性伸缩，判断的指标是AHAS应用防护中统计的应用QPS（平均到每台机器），超过30就进行扩容，最大扩容副本数为3。

3. 执行 `kubectl apply -f ahas-sentinel-hpa.yml` 来创建HPA配置。

示例压测流量runner的Deployment ❯

验证弹性结果

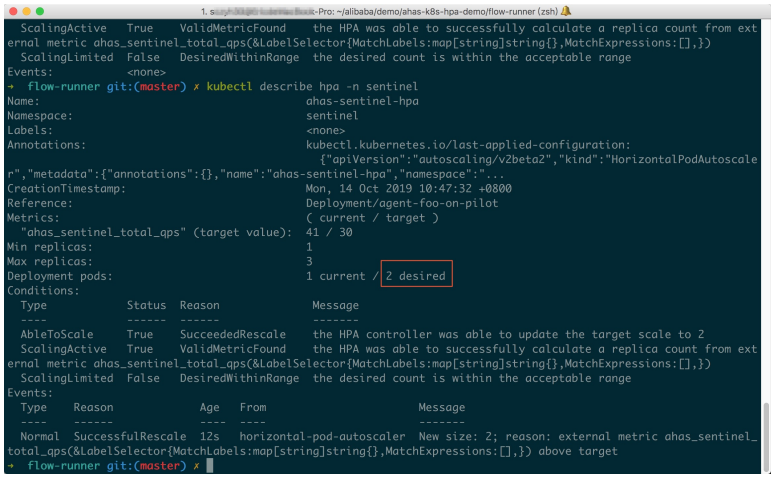
本文提供了发送压测流量的镜像，方便测试弹性效果。

通过可以控制该runner的副本数来控制压测的QPS，例如上述示例中调整replica=3即对应30QPS。

1. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
2. 单击目标服务**操作**列的**伸缩**，在**伸缩**对话框，设置**所需容器组数量**为4。

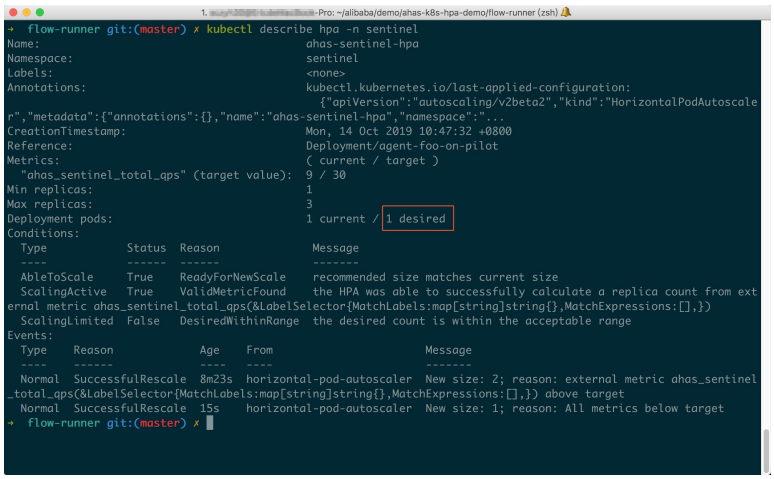
   本文将replica数目调成4，使得QPS超过临界值，触发扩容。

   约5分钟后，可以看到对应的应用容器数目自动扩容成了2。

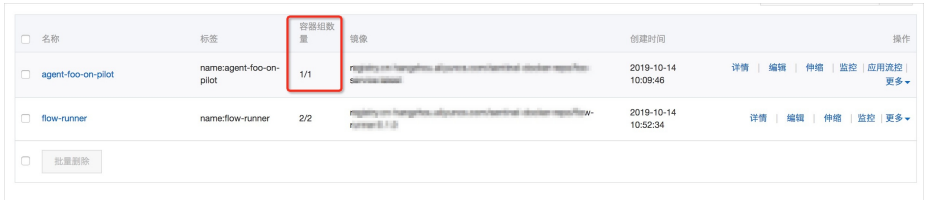


登录AHAS控制台，在左侧导航栏选择**流量防护 > 应用防护**，单击应用卡片，然后在应用详情页的左侧导航栏选择**机器监控**，可以看到增加的实例。

3. 将示例中的 `replica` 设置为3。
   约5分钟后，可以看到对应的应用容器数目自动缩容成了1。



在容器服务管理控制台的左侧导航栏单击**集群**。在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。在集群管理页左侧导航栏中，选择**工作负载 > 无状态**，可以看到容器组数量缩容为1。



5.2. 故障演练最佳实践

5.2.1. 强弱依赖治理最佳实践

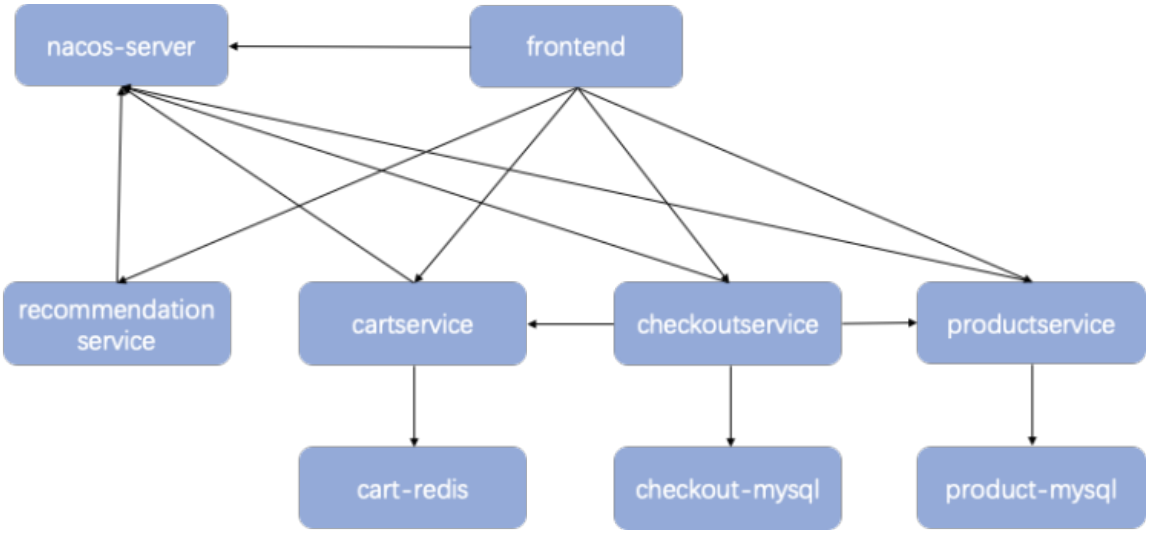
本文以对一个部署在Kubernetes上的微服务应用进行强弱依赖治理为例，介绍通过场景化演练来发现依赖问题、暴露风险的整个过程。

背景信息

关于强弱依赖治理的更多信息，请参见强弱依赖治理概述和应用强弱依赖治理。

示例说明

本文示例的应用是一个开源电商Demo，提供了商品选购、购物车、下单等功能。由于应用程序不存在对外部云服务的依赖，并且提供了预构建镜像，所以该应用可以直接部署在任意Kubernetes集群上。Demo应用架构如下图所示。



该微服务Demo由9个服务构成：

- nacos-server：注册中心。
- frontend：前端。
- recommendationservice：推荐系统，负责对首页的产品列表进行排序。
- cartservice：购物车。
- cart-redis：购物车信息储存的数据库。
- checkoutservice：下单。
- checkout-mysql：下单信息储存的数据库。
- productservice：商品。
- product-mysql：商品数据库。

部署应用

部署应用支持以下2种方法：

- 部署方法1：使用预构建镜像快速部署。
  运行prebuild中的所有YAML文件，使用预构建的镜像快速部署。

  ```
  cd prebuild/
  for i in *.yaml; do kubectl apply -f $i; done
  ```

- 部署方法2：手动编译并部署。

i. 在根目录下运行 `mvn clean install` 。

ii. 进入每个子模块目录*src/{submodule}*，运行 `./build.sh` ，编译并构建镜像文件。

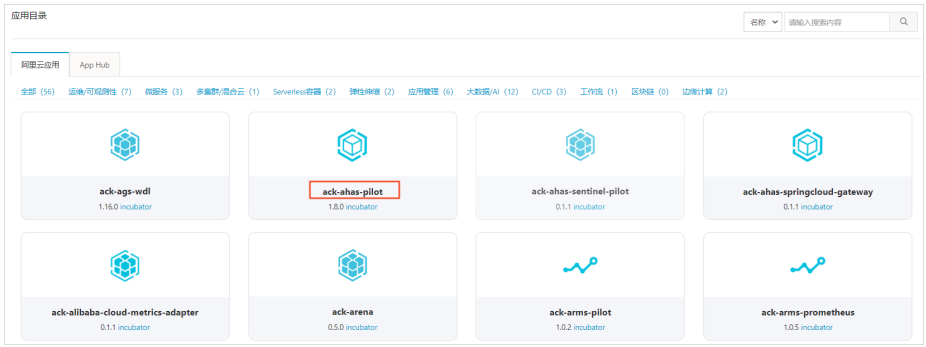
iii. 修改YAML文件中镜像。

iv. 部署应用。

创建治理方案

一个治理方案针对一个应用，例如需要对电商Demo的前端页面进行治理，则选择frontend应用。在主机模式下，接入探针时需填写应用名；在K8s环境下，您需要通过Label来识别应用名。

1. 登录AHAS控制台。
2. 在左侧导航栏选择**故障演练 > 演练方案**，然后在**微服务演练**页面单击**强弱依赖治理**。
3. 在**强弱依赖治理**页面，单击**创建治理方案**。
4. 输入**方案名称**，单击**新应用接入**，选择Kubernetes，接入新应用。
   i. 登录容器服务管理控制台。
   ii. 在左侧导航栏选择**市场 > 应用目录**，单击**ack-ahas-pilot**，根据具体情况修改参数，单击**创建**，详情请参见架构感知监控。



接入探针后，Kubernetes中打标签为 `app=<name>` 的Pod将显示在AHAS控制台故障演练的治理应用中。

```
Labels:         app=frontend
```

5. 单击**下一步**，进入**依赖分析**，注入流量。

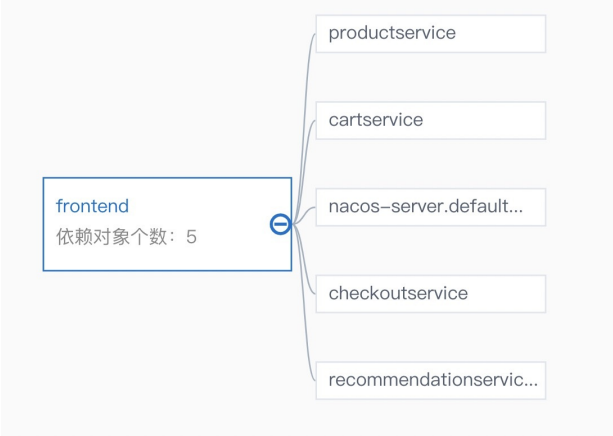
   由于依赖关系的准确识别是需要流量的，如果在流量不足的测试环境中接入，则需要您手动提供流量。推荐使用PTS等工具创建压测提供流量，本文示例将压测API中的链接地址更换为frontend的公网访问地址，可自定义流量注入链路。

   i. 登录PTS控制台。

ii. 在左侧导航栏选择**压测中心 > 创建场景**，在**创建场景**页面单击**PTS压测**，然后在**场景配置**中创建压测场景。详情请参见创建压测场景。

6. 在**依赖分析**页面，系统自动识别依赖关系。

   进入**依赖分析**页面，在应用存在正常流量的情况下，系统会自动分析建立依赖关系。如果超过一定时长依赖关系无变化，则表示在当前流量关系下，依赖已完全展示。本文示例识别出frontend有5个依赖。



7. 单击**下一步**，进入**依赖预判**，进行业务依赖判断。

   业务依赖是指对识别到的依赖进行强弱关系的预判，依赖预判不能脱离业务的特性，系统将依赖分为强依赖与弱依赖。如本示例下图所示。

   通过以上业务依赖的预判可以得出以下结论：

   - 前端对商品推荐服务预判为弱依赖，表示当推荐服务发生故障时前端正常访问不应该受阻。
   - 在购物链路中，商品服务product对商品数据库product-mysql预判为强依赖。表示如果扣减库存失败，则应该阻断下单流程，否则可能导致超卖的错误。

8. 单击**下一步**，进入**依赖验证**。选择要验证的依赖，单击**去验证**。

   在对业务分析进行依赖预判后，应通过故障注入的方式验证真实依赖关系是否与预判相符，例如注入依赖的服务间的网络延迟故障。强弱依赖的验证可以有多种指标，例如监控与日志的报警，请求的返回状态码等等。
   本文示例预期frontend与cartservice为弱依赖，单击**去验证**，开始故障演练。

验证结果是前端页面无法显示（或HTTP返回结果非200），如下所示。

9. 演练完成后，单击**恢复**或**终止**。在**结果反馈**页面，记录**结论**和**验证结果**。返回到**依赖验证**页面继续验证其他依赖。
10. 单击**方案归档**，结束1个应用的强弱依赖治理。归档后将无法进行再次分析、验证，但治理报告可正常下载。未手工归档的方案，系统将在30天内自动归档。

总结

强弱依赖治理提供了一系列自动化功能，包括应用接入、依赖智能识别、对应演练自动创建、依赖验证等。为了实现全面准确的依赖分析，您需要提供全面覆盖依赖的流量。

利用强弱依赖治理梳理出应用的依赖关系的强弱，能够提前发现风险、发现不合理的依赖关系，进而提升应用的高可用能力。

演示操作

以下通过3个视频，演示下强弱依赖治理的整个过程。

1. 准备工作
2. 操作步骤
3. 总结

5.2.2. 混沌工程缓存实战系列-Redis

Redis是一个开源高性能的Key-Value存储系统，虽然Redis本身具备了非常高的可用性，但是在实际应用中也会随着系统业务的复杂性以及不合理的使用，而导致很多的问题。本文将讲述如何通过混沌工程来暴露可能存在的使用风险，提升缓存问题的应急能力。

缓存重要性

Redis是一个开源高性能的Key-Value存储系统，因为其极高的读写性能，丰富的数据类型，原子性的操作以及其他特性而被广发运用。

Redis的应用场景包括且不限于以下场景：

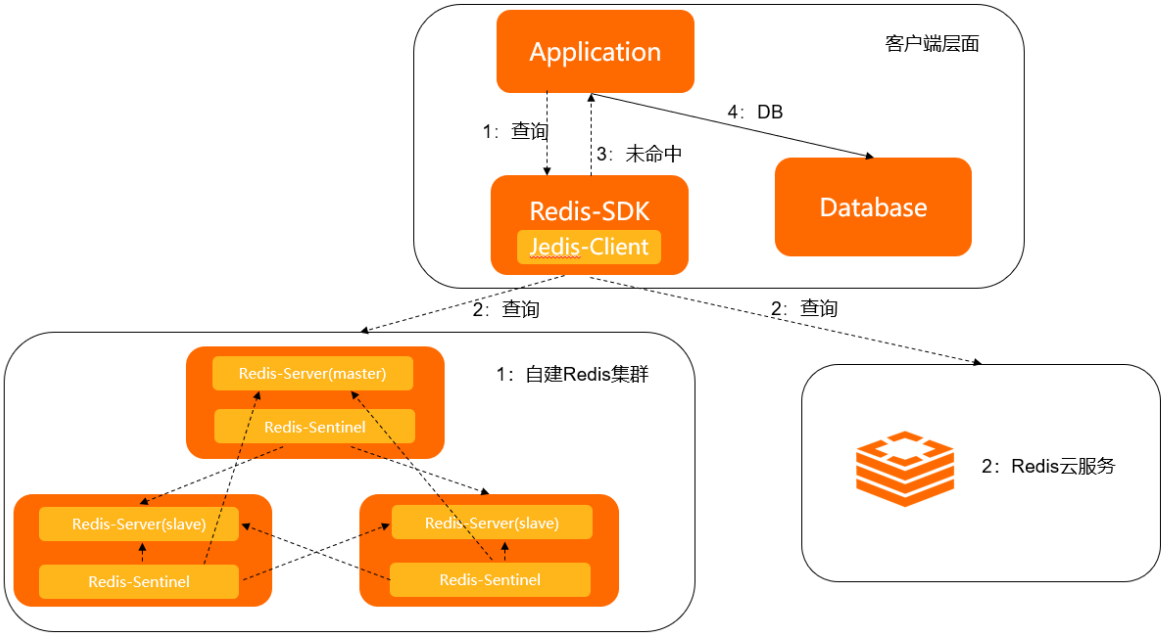
- 用来做分布式缓存。
- 用来做分布式锁。
- 用来处理某些特定高并发业务，例如秒杀等。

示例架构

在实施混沌工程之前，先了解业务是如何使用Redis的。由于Redis最常用来做分布式缓存，本文以简单的商品查询场景为例，涉及的基本信息如下：

- 业务场景是查询商品信息，首先查询缓存；如果没有查询到，则查询数据库。
- 使用Jedis连接Redis，并且使用了Jedis-pool的技术。
- Redis是自建的集群（当然也可以使用云服务），并且使用Sentinel技术来提升集群的高可用性。更多信息，请参见Redis Sentinel文档。

示例架构图如下：



从架构图可以看出，在Jedis配置、缓存查询、网络传输、服务端处理这条链路上，每个环节都有可能出现问题。借助混沌工程可以了解到问题发生时对系统、业务的影响面是否符合预期。

梳理演练场景

对于示例应用，可以按照以下思路来梳理演练场景：

1. 明确缓存监控的指标。
2. 分析影响这些指标可能的因素、故障场景、参数等。
3. 因为客户端层面的影响面可控，所以可以尝试从客户端层面去制造故障。
4. 因为服务端出现故障更加真实，所以可以从服务端层面去制造故障，但对于问题定位和排查的要求会更高。
5. 注入故障，观察指标的变化。

缓存监控指标

目前支持的可监控的缓存指标如下：

指标	说明
缓存QPS	QPS是最通用也是最易观察的指标。

指标	说明
缓存命中率	缓存未命中可能会在大流量下引发穿透、击穿、雪崩等问题，如果业务没有做好应急处理，很容易压垮数据库。 • 穿透：Key对应的数据在数据源并不存在，每次针对此Key的请求从缓存获取不到，请求都会到数据源，从而可能压垮数据源。例如用一个不存在的用户ID获取用户信息，不论缓存还是数据库都没有，若黑客利用此漏洞进行攻击可能压垮数据库。 • 击穿：Key对应的数据存在，但在Redis中过期。此时若有大量并发请求，这些请求发现缓存过期一般都会从后端数据库加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把后端数据库压垮。 • 雪崩：当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，也会给后端系统（例如数据库）带来很大压力。
缓存RT	缓存响应时间。缓存RT对业务的影响分成多个方面。如果RT变化较少，对于业务访问缓存很少次数的情况下影响可控。但是如果一条请求需要多次访问缓存，那么哪怕RT只是几毫秒的增长，也会因为访问次数过多引起总的RT增长过多，引发蝴蝶效应，造成业务异常。
缓存成功率	缓存的请求成功率过低也会造成和命中率一样的后果。
业务成功率	在对业务成功率要求较高的业务当中，缓存必定是作为一个弱依赖存在。当缓存出现问题，系统应该有其他兜底策略。但是随着系统越来越复杂，改造的越来越频繁，原本预计的缓存弱依赖也会不经意间被改造成强依赖，一旦出现这种情况，就会导致业务受损。

影响因素

由于影响系统的因素有很多，例如机房、电源、集群服务、操作系统、应用配置等。

本文主要梳理操作系统层面和应用层面的影响因素：

- 系统层面的影响因素有网络、磁盘、IO、内存、CPU等因素。
- 应用层面的影响有超时配置、连接池配置、查询不合理等因素。

结合缓存监控指标、操作系统层面和应用层面的影响因素，本文从客户端和服务端两个角度来分析最终影响系统的因素和后果（假设业务请求QPS保持稳定）。

客户端

因素	模拟手段	可能后果	可能影响指标
网络延迟	6379端口网络延迟	• 读写请求RT变长 • 连接池满	• QPS • RT • 成功率
网络中断	6379端口网络丢包	• 读写失败 • 无法连接	• QPS • RT • 成功率

因素	模拟手段	可能后果	可能影响指标
单次查询耗时过长	如果Key过多，可以模拟Keys*查询	• 单次请求RT变长 • 连接池满	• QPS • RT • 成功率
连接池设置不合理（连接池过小或者过高）	Jedis连接池占满	无法建立连接	• QPS • RT • 成功率
缓存未命中	Jedis返回值拦截	• 穿透 • 击穿 • 雪崩	命中率
缓存异常	Jedis抛异常	缓存强依赖，业务失败。	成功率

服务端

因素	模拟手段	可能后果	可能影响指标	如何改进
磁盘空间不足	磁盘填充	• 开启了AOF会导致日志无法写入。 • 无法备份缓存数据。	• QPS • RT • 成功率	• 监控磁盘利用率。 • 禁AOF。
网络异常	• 端口延迟 • 端口丢包	指定客户端请求超时。	• QPS • RT • 成功率	• 网络监控。 • 集群。
连接池满	建立网络连接	无法分配新连接，客户端建连失败。	无	• 设置timeout和tcp-keeplive参数。 • 网络监控。
单次查询耗时过长	如果Key过多，可以模拟Keys*查询。	• 单次请求RT变长。 • 连接池占满。	• QPS • RT • 成功率	• 避免Keys*类查询。 • RT监控。
IO读写过高	磁盘读写IO过高	读写变慢，响应超时。	• QPS • RT • 成功率	• 避免Key类查询。 • 主从持久化策略修改。
内存不足	Jedis返回值拦截	• 内存不足导致AOF无法备份。 • 内存不足导致无法写入。	• 命中率 • 成功率	• 内存监控。 • 设定内存阈值，合理内存策略。 • 缓存设定失效，避免冷数据库过多。

因素	模拟手段	可能后果	可能影响指标	如何改进
CPU过高	CPU满载	读写RT变长。	• RT • 成功率	系统监控。

实战演练

下面通过Chaos故障演练平台从客户端层面来评测业务对Redis的合理使用。

1. 分析演练系统。

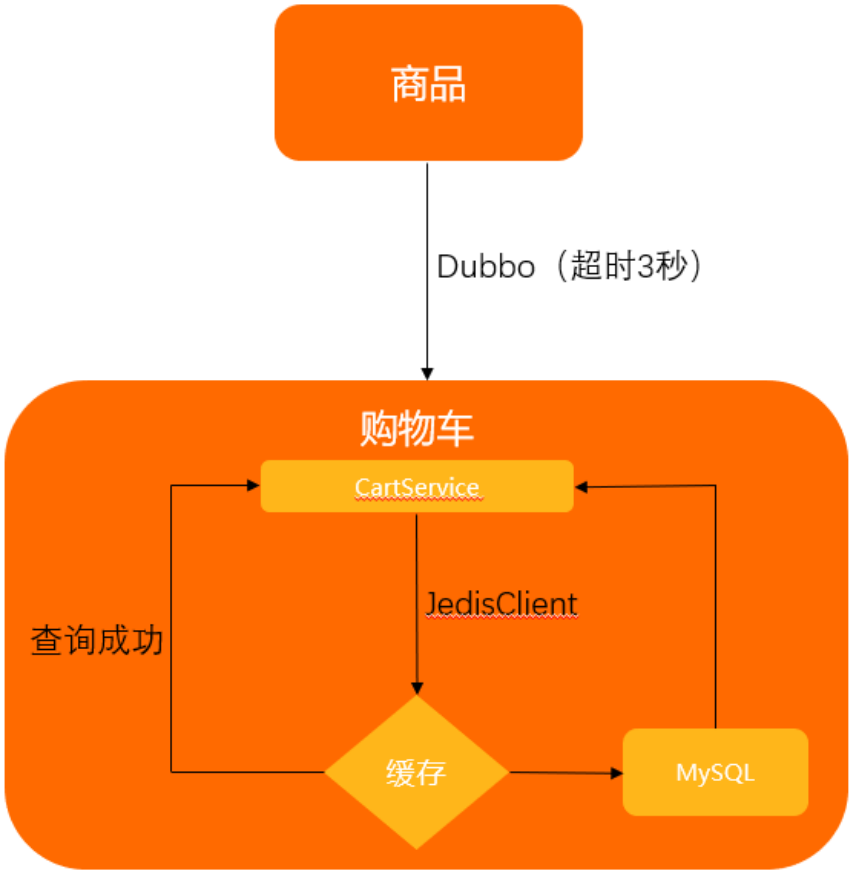
   本文示例的业务场景是简单的商品购物车查询，为了便于理解，对该系统做了逻辑简化，同时为了尽可能的模拟真实情况，您可以制定相关的业务指标如下：

   - 整个系统分为首页和购物车页面。
   - 首页通过Dubbo来调用购物车接口，购物车服务端的接口超时设置为3000 ms。
   - 每一次购物车的内部查询，都需要查询50次的缓存（为了更好观看演练效果，次数稍微放大），每次缓存的操作约10 ms。
   - 购物车的内部查询优先经过缓存，失败了以后再使用数据库。
   - 连接缓存的SDK使用Java的JedisClient，设置的超时时间为100 ms。

   核心查询代码如下：

```
    //弱依赖缓存。
    @Override
    public List<CartItem> viewCart(String userId) {
        try {
            for (int i = 0; i < 50; i++) {
                logger.info("query redis,count:" + i);
                redisRepository.getUserCartItems(userId);
            }
            return redisRepository.getUserCartItems(userId);
        } catch (Exception exception) {
            logger.error("get data from redis failed ,use local data", exception);
        }
        logger.info("get data from redis failed ,query from db");
        return cartDBRepository.findByUserId(userId).stream().map(this::fromCart).collect(Co
llectors.toList());
    }
```

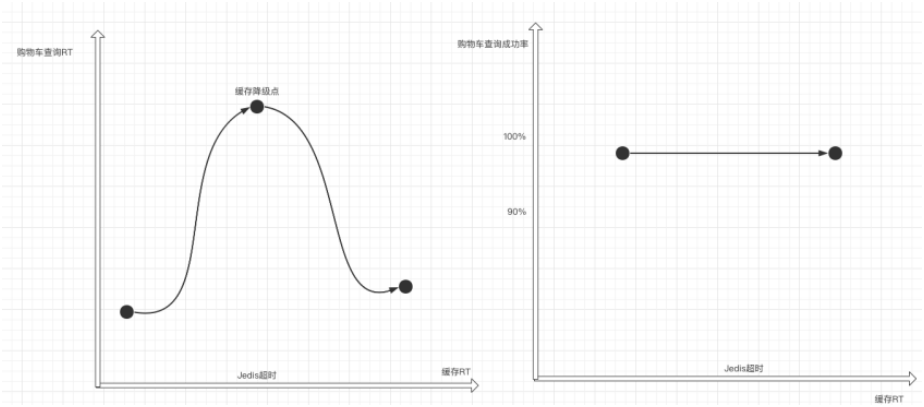
   相关的架构图如下：



2. 假设演练场景。

   从影响因素里可以看到影响Redis使用稳定性有很多原因，这里挑选一个场景：评测网络延迟对Redis使用的影响，来观察RT变化之后业务能否继续保持正常服务。
   基于网络延迟这个场景，可以提出这样的假设：

   - 缓存的RT变化不应该影响到购物车查询的成功率。
   - 由于缓存RT的增加导致购物车查询RT会先增加，接着缓存RT增加到一定的值使得缓存彻底无法访问，此时会触发缓存降级，购物车会查询数据库，这样又会使得查询RT回落。
   - 虽然RT变化了，但是因为操作了强弱依赖治理，购物车查询成功率不会有很明显的变化。



3. 设计演练场景。

   针对上面的假设结合系统特征，可以设计出以下的演练场景：

   - Redis延迟增加20 ms，缓存累计耗时50*20=1000 ms，此时CartServiceRT小于配置的接口超时3000 ms，

业务正常。

- Redis延迟增加80 ms，缓存累计耗时50*80=4000 ms，购物车内部查询继续，但此时CartServiceRT大于配置的接口超时3000 ms，购物车查询失败。
- Redis延迟增加10000 ms，此时缓存超出了Jedis的超时配置时间100 ms，使得查询缓存故障，导致查询路径切换至数据库，此时业务正常。

4. 实施演练。

通过阿里云Chaos演练平台可以快速的配置以上的演练场景，并且结合平台提供的业务探活功能，可以快速实现整个故障演练的自动化评测。

i. 通过探针管理向Cart服务所在的机器安装演练探针。

ii. 创建演练场景。

本示例创建网络延迟的故障场景。

a. 登录AHAS控制台，在左侧栏选择**故障演练 > 我的空间**。

b. 在**我的空间**页面，单击**新建演练 > 新建空白演练**。

c. 在**演练配置**页面，填写相关参数，选择**演练内容**为**主机内网络延迟**。更多参数信息，请参见创建演练。

d. 单击**主机内网络延迟**，在**本地监听端口**文本框输入*6379*，在**延迟时间**文本框输入延迟时间。更多信息，请参见网络类场景。

iii. 增加业务探活的节点。

由于要观测演练前和故障注入后系统的业务情况，因此除了故障注入节点之外，还需要增加业务探活的节点。故障演练提供了类似K8s的探活功能，可以通过访问指定接口来判断业务是否可用。参数配置说明如下：

参数	描述	示例值
`failureThreshold`	重试次数，重试几次失败后判断为校验失败。	5
`periodSeconds`	探测时间间隔。	2秒
`successThreshold`	连续成功几次算成功。	2
`url`	需要探测的URL。	http://www.example.com（购物车的查询地址）
`method`	GET或POST方法。	GET

最终配置成如下完整演练流程：

注意 在演练前需要确保业务系统处于正常状态，所以在故障注入前需要判断下应用是否可用。

iv. 执行演练。具体操作，请参见执行演练。

配置完毕之后，可以发起自动演练、自动探测，最终得出结论（故障演练支持演练节点自动推进，也支持手动一步步推进）。

v. 验证结果。

从演练执行结果可以看出，最终的运行结果和假设一致，当延迟注入80 ms之后，购物车不可用。但当延迟注入20 ms和10000 ms时候，虽然购物车可用，但还需要进一步验证是否如预期：一个是RT延长但是接口未超时，一个是缓存降级导致的业务成功。
可以通过单击校验购物车是否可用的节点来查看业务成功的原因：

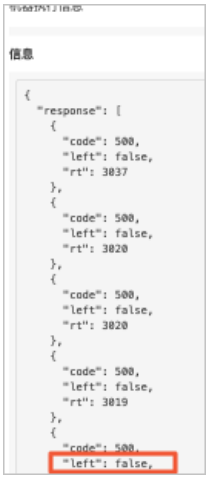
a. 查看演练开始的探活节点，单击购物车校验是否可用，查看探活记录。发现查询RT处于正常范围内。


```
{
  "response": [
    {
      "code": 200,
      "left": true,
      "rt": 26
    },
    {
      "code": 200,
      "left": true,
      "rt": 21
    },
    {
      "code": 200,
      "left": true,
      "rt": 18
    },
    {
      "code": 200,
      "left": true,
      "rt": 19
    },
    {
      "code": 200,
      "left": true,
      "rt": 19
    }
  ]
```


b. 查看注入20 ms之后的探活节点。发现业务RT明显增长，但是还是在超时的3秒内，因此业务正常。


```
{
  "response": [
    {
      "code": 200,
      "left": true,
      "rt": 2028
    },
    {
      "code": 200,
      "left": true,
      "rt": 1949
    },
    {
      "code": 200,
      "left": true,
      "rt": 2071
    },
    {
      "code": 200,
      "left": true,
      "rt": 2020
    },
    {
      "code": 200,
      "left": true,
```

c. 查看注入80 ms之后的探活节点，发现业务异常。



d. 查看注入10000 ms之后的探活节点，发现RT回落，此时业务正常。

> **说明** 但RT相比正常值还是有所延长。这是由于缓存出现故障，导致购物车查询缓存失败，此时购物车则需再去查询数据库。这个查询路径切换的过程导致RT相较于正常值有所延长。

通过以上的演练证明了以下几点：

- 缓存RT轻微增长，对业务影响可控。但是如果业务内部存在多次的缓存查询，会导致整体RT增加明显，就像本示例RT延长处于客户端连接超时范围内，无法触发弱依赖降低，但是整个接口RT超时，最终导致业务受损。
- 在缓存RT增长很明显的情况下，缓存降级策略能够正常生效，使得业务正常访问。当然在实际情况中，这种兜底策略可能导致数据库直接崩溃。

演练价值

通过对不同网络延迟的演练，可以了解到缓存RT变化对系统造成的影响，以及防护策略有效性。随着业务规模的不断增长，这个简单的业务系统也会面临新的问题：

- 在某次重构中，又新增加了缓存查询，结果导致20 ms的延迟使得接口整体超时。
- 业务逻辑简单的时候，能够很好的分析强弱依赖。但是随着微服务的膨胀，以及代码多次重构，可能原有的弱依赖在某次变更中变成了强依赖，这种通过功能测试是无法发现的。
- 本示例Jedis设置的超时时间是100 ms，不同业务对RT的要求不同，您可以根据实际情况设置合理的超时时间。

上述的一些问题都要通过故障演练来发现。在日常的发布、架构升级中除了功能测试、性能测试的回归，还需要进行常态化的故障演练，同时演练的形态和场景复杂性也要不断扩充。对于故障演练来说，难的不是注入手段，而是对业务架构、业务场景的理解。故障注入不是目的，演练的目的是加深对系统的理解，这样当真实的问题来临时候，才能更加有信心的去处理。

5.3. 功能开关最佳实践

5.3.1. 运行时动态调整日志级别

在特定的场景下，您需要针对性地动态调整日志级别，以便输出更多的日志信息排查线上问题，或是减少日志打印带来的性能消耗。功能开关提供了在应用运行时动态修改日志级别的功能，在不同的应用场景下，您可以随时调整日志的级别，得到更有效的日志信息。

背景信息

在开发Java程序时，我们经常会用到各种各样的日志框架。为了避免在程序正常运行时输出不必要的信息，我们在使用日志框架时会设置默认的日志级别。而程序在线上运行时，我们需要在特定的场景下针对性地动态调整日志级别。

> 说明
>
> - 支持的日志框架：Log4j、Log4j2、Logback。
> - Spring Boot、SDK 版本要求：版本号≥1.0.3。

操作步骤

1. 使用 Spring Boot 或 Java SDK 接入应用，详情请参见使用Spring Boot Starter接入和使用SDK接入。
2. 登录 AHAS 控制台，在页面左上角选择**地域**。
3. 在控制台左侧导航栏选择**功能开关**，在**应用列表**页面单击目标应用的资源卡片。进入目标应用的**开关列表**页面。
4. 在**开关列表**页面搜索到 SYSTEM_LOG_CONFIG 开关，即日志级别开关。

5. 单击**操作**列的**全局推送**或**单机推送**，按照 `<loggerName,loggerLevel>` 格式填写日志运行的配置，然后单击**全局推送**或**单机推送**。即可修改全部机器或是单台机器的日志运行级别。

   推送值格式：`Key` 为 `LoggerName`，`Value` 为日志级别。如需修改全局日志级别，`LoggerName` 为 `root`，如下所示。

```
{
 "root": "ERROR"
}
```

5.3.2. 主动降级业务功能

通常一个业务功能包含许多的业务逻辑，其中可以区分出一些核心业务和非核心业务。在高并发的情况下，例如618、双十一等场景，为了提升系统性能，系统需要减少非必要业务的资源消耗，对非必要的业务功能进行主动降级。本文介绍如何通过功能开关快速实现业务功能主动降级。

前提条件

您已在功能开关中接入了应用，详情请参见使用Spring Boot Starter接入和使用SDK接入。

操作步骤

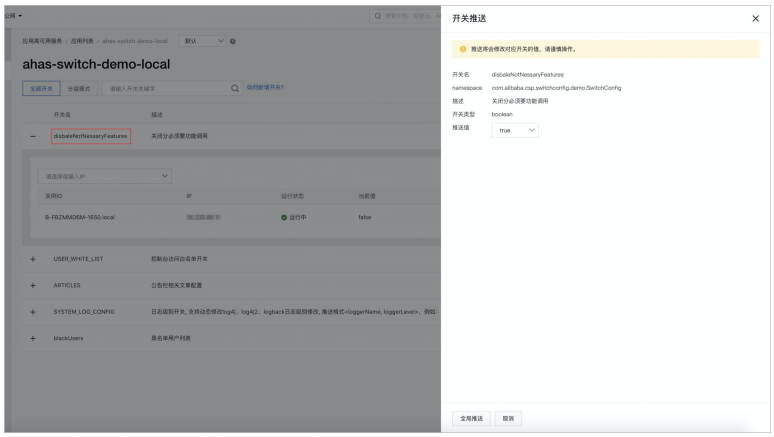
1. 在代码中定义功能开关。

```
@Switch
public class SwitchConfig {
    @AppSwitch(des = "关闭非必要功能调用")
    public static boolean disableNotNessaryFeatures = false;
 }
```

2. 在代码中植入埋点，然后重新发布代码。

```
if (SwitchConfig.disableNotNessaryFeatures) {
  // 关闭非必要功能后的处理逻辑。
}
// 正常业务逻辑。
```

3. 在AHAS控制台左侧导航栏选择**功能开关**，在**应用列表**页面单击目标应用的资源卡片。进入目标应用的**开关列表**页面。
4. 在**开关列表**页面搜索到disableNotNessaryFeatures开关，即降级业务开关。
5. 设置开关推送的配置，详情请参见设置开关推送。



5.3.3. 快速实现黑白名单功能

黑白名单是常用的访问控制规则，可以实现对不同用户身份的识别和过滤，达到控制用户权限的目的。本文介绍如何通过功能开关快速实现黑白名单功能。

前提条件

您已在功能开关中接入了应用，详情请参见使用Spring Boot Starter接入和使用SDK接入。

操作步骤

以下操作步骤以增加黑名单用户列表为例。

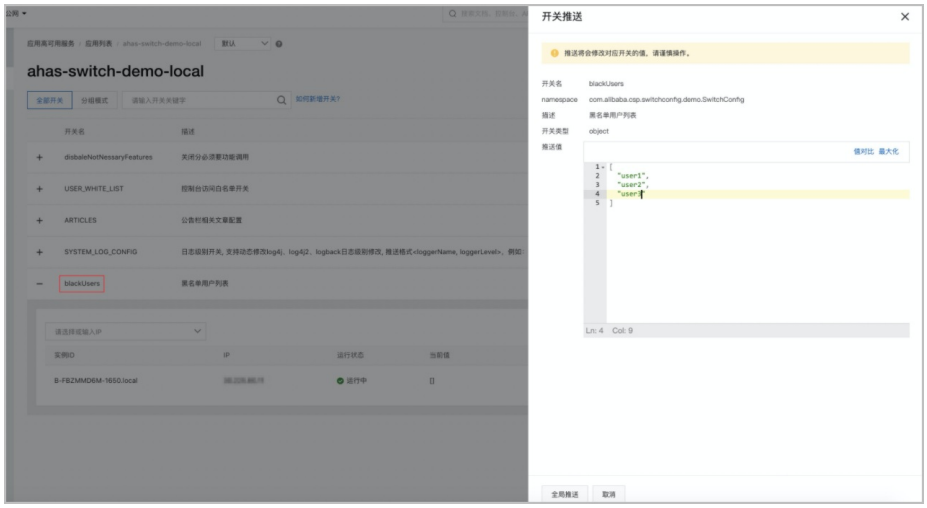
1. 定义功能开关。

```
@Switch
public class SwitchConfig {
    @AppSwitch(des = "黑名单用户列表")
    public static List<String> blackUsers = new ArrayList<String>();
 }
```

2. 在代码中植入埋点，然后重新发布代码。

```
if (SwitchConfig.blackUsers.contains(userId)) {
  // 黑名单用户处理逻辑。
}
// 正常用户处理逻辑。
```

3. 在AHAS控制台左侧导航栏选择**功能开关**，在**应用列表**页面单击目标应用的资源卡片。进入目标应用的**开关列表**页面。
4. 在**开关列表**页面搜索到 blackUsers开关，即黑名单开关。
5. 设置开关推送的配置，详情请参见设置开关推送。



5.4. 多活容灾最佳实践

5.4.1. 电商业务多活实践

本文介绍一个由多个微服务实现的完整业务应用利用MSHA产品改造为异地多活架构的过程。

前提条件

- 创建2个集群，请参见创建Kubernetes托管版集群。
- 开通阿里云应用配置管理ACM，请参见开通ACM服务。
- 开通阿里云数据库MySQL版的RDS或DRDS，请参见RDS MySQL数据库。
- （可选）开通阿里云云解析DNS，请参见云解析DNS。

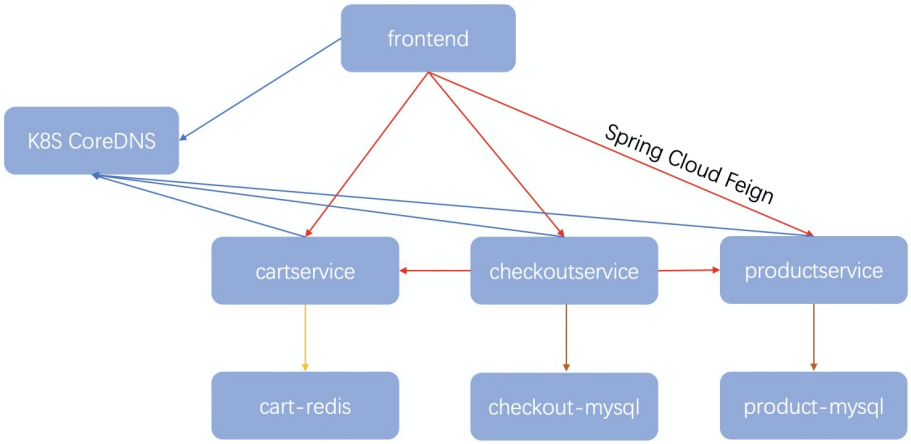
开通产品

1. 公测阶段，您需要申请开通并配置MSHA。
2. 确定您需要的多活架构类型（仅异地、仅同城、异地+同城），本文示例以仅异地为例。
3. 确定多活业务类型，例如电商业务可以申请两种业务类型：导购和交易。若您无特殊需求，则不需要处理，MSHA会为您生成默认业务类型。

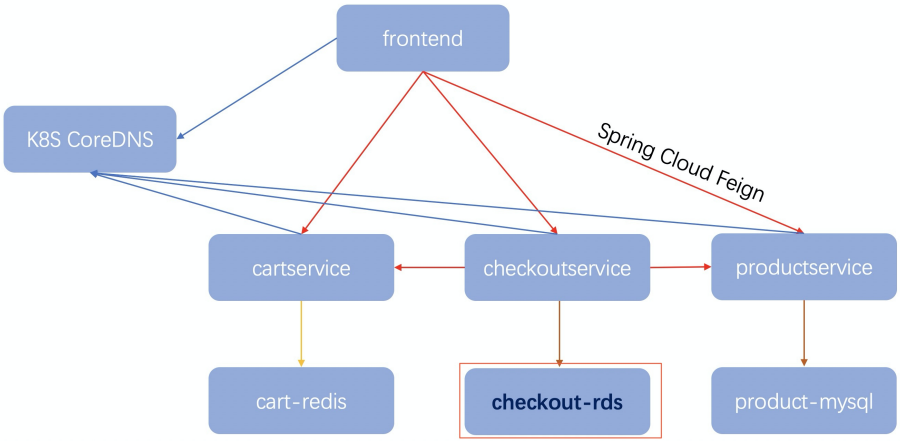
改造架构

假设这个业务由以下微服务共同实现，服务发现依赖于K8s实现，服务间调用利用Feign实现。

- frontend，一个传统的MVC服务。负责和用户交互。
- cartservice，购物车服务。记录用户的购物车数据，自建Redis存储。
- productservice，产品服务。存储商品信息，包含商品详情、库存等，自建MySQL存储。
- checkoutservice，下单服务。从购物车拉取商品信息，并从产品服务检验库存，完成下单，自建MySQL存储。



为了演示方便，本文示例仅将下单服务进行单元化改造，改造完成后，下单服务会有单元保护、跨单元订单操作等功能，为此需要将自建MySQL换成阿里云RDS，如下图所示。



主要原因是自建MySQL非标准数据库，MSHA无法管控，后续阿里云会提供一系列的标准，并支持符合这些标准的自建MySQL。

控制台接入

1. 配置命名空间。
   i. 登录AHAS控制台，在控制台左侧导航栏中选择**多活容灾**。
   ii. 在左侧导航栏选择**基础配置 > 命名空间**，单击**新增命名空间**。
   iii. 填写**命名空间名称**，例如交易单元化的正式环境，导购单元化的测试环境。
   iv. 从**业务类型**下拉列表中选择业务类型，选择**容灾架构类型**为**异地双活**，然后选择启用的多活组件。
   本示例中选择启用接入层和数据层。若需要新增业务类型，具体操作，请参见新建命名空间。

v. 设置**接入层路由标提取方式**。

路由标提取方式支持Header和Cookie的排列组合提出方式，本示例先从HTTP Header中找routerId的值，若没找到再从HTTP Cookie中找routerId的值。

vi. 单击**添加ACM**，为每个单元配置相关信息，包含ACM以及接入层集群。

vii. 从**接入层集群**下拉列表中，选择接入层集群。

若需要新增接入层集群，具体操作，请参见管理MSFE接入层集群。

viii. 单击**下一步**，添加路由标解析规则。

路由标解析规则是指路由标到路由ID的解析过程，本示例中routeId=111123，按照10000取模，那么路由ID就是1123。

ix. 单击**确定**，添加单元分流。即哪些ID到哪些单元，例如0~4999到杭州，5000~9999到北京。然后单击**确定**。

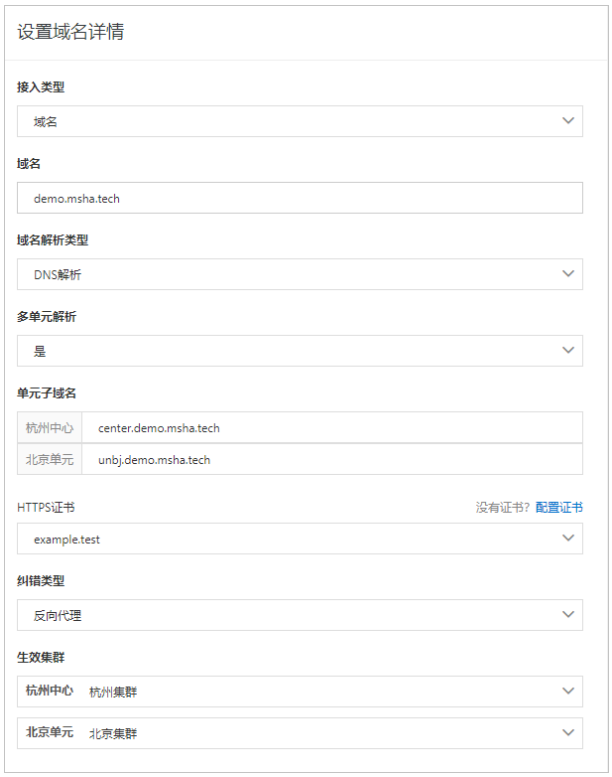
2. 配置接入层。

i. 在左侧导航栏选择**异地双活 > MSFE配置**。

ii. 单击的**新增域名**，录入域名。具体操作，请参见配置MSFE。

> 说明
> - 如果是阿里云DNS**域名解析类型**请选择DNS解析，如果是其它供应商请选择不解析。
> - 纠错类型支持两种：反向代理和重定向，用户可按需选择。

本文示例如下。



iii. 单击右上角的**新增域名**，**接入类型**选择IP，录入URI，然后单击**确定**。

本文示例中的这两个IP分别是杭州和北京单元的Frontend Service对外提供的IP，可以是真实IP，也可以是VIP。

iv. 录入完成后，单击**操作**列的**生效**。

3. 配置异地数据层。

i. 在左侧导航栏选择**异地双活 > 数据层配置**。

具体操作，请参见配置数据层。
本文示例中是杭州和北京各自的checkout-RDS。

ii. 配置完成后，在**数据层配置**页面预览，可见数据已经在同步中。

改造数据面

- **服务层**
  由于服务层无多活逻辑，所以您仅需做多活参数透传，也就是 `routerId` 和 `unitType` （如果仅default的话，就不需要传）。
  对于 `unitType` ，接入层会在用户请求Header加上，所以服务层可以直接从Header中获取；对于 `routerId` ，则需要服务层按照自己的配置进行解析（自然的前端在发起请求的时候必须在Header或Cookie中带上 `routerId` 参数），按照在全局配置引导中的配置， `frontend` 解析多活参数的代码如下。

```
@Component
public class UnitLogicInterceptor extends HandlerInterceptorAdapter {
    private Logger logger = LoggerFactory.getLogger(UnitLogicInterceptor.class);
    private static final String DEFAULT_UNIT_TYPE = "unit_type";
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler)
            throws Exception {
        if (cookies != null){
            for (Cookie cookie : cookies) {
                if ("routerid".equalsIgnoreCase(cookie.getName())){
                    // 放进thread local便于透传。
                    threadLocalRouterId.set(Long.valueOf(cookie.getValue()));
                }
                if ("Unit-Type".equalsIgnoreCase(cookie.getName())){
                    threadLocalUnitType.set(cookie.getValue());
                }
            }
        }
           Enumeration<String> headerNames = request.getHeaderNames();
        if (headerNames != null) {
            while (headerNames.hasMoreElements()) {
                String name = headerNames.nextElement();
                if ("routerid".equalsIgnoreCase(name)){
                    threadLocalRouterId.set(Long.valueOf(request.getHeader(name)));
                }
                if ("Unit-Type".equalsIgnoreCase(name)){
                    threadLocalUnitType.set(request.getHeader(name));
                }
            }
        }
        if (threadLocalUnitType.get() == null){
            // 为后面跳过接入层直接访问服务层作准备。
            threadLocalUnitType.set(DEFAULT_UNIT_TYPE);
            logger.info("default setUnitType {}", DEFAULT_UNIT_TYPE);
        }
        return true;
    }
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object h
andler, ModelAndView modelAndView)
            throws Exception {
        threadLocalRouterId.remove();
        threadLocalUnitType.remove();
    }
}
```

frontend需要将多活参数透传给checkoutservice，有以下两种做法：

- 隐式透传，利用请求附加属性透传，让checkoutservice自行解析。比如Feign的requestInterceptor，dubbo的context，请参见context。
- 显示透传，直接改造service签名，加入多活参数，本Demo用第二种。

```
@Controller
public class AppController {
    public static ThreadLocal<Long> threadLocalRouterId = new ThreadLocal<>();
    public static ThreadLocal<String> threadLocalUnitType = new ThreadLocal<>();
```

```
    @PostMapping("/checkout")
    @ResponseBody
    public Map<String, String> checkout(@RequestParam(name = "email") String email,
                                        @RequestParam(name = "street_address") String streetA
ddress,
                                        @RequestParam(name = "zip_code") String zipCode,
                                        @RequestParam(name = "city") String city,
                                        @RequestParam(name = "state") String state,
                                        @RequestParam(name = "country") String country,
                                        @RequestParam(name = "credit_card_number") String cre
ditCardNumber,
                                        @RequestParam(name = "credit_card_expiration_month")
int creditCardExpirationMonth,
                                        @RequestParam(name = "credit_card_cvv") String credit
CardCvv) {
        String orderId = orderDAO.checkout(email, streetAddress, zipCode,
                                           city, state, country, creditCardNumber,
                creditCardExpirationMonth, creditCardCvv,
                threadLocalRouterId.get()+"",threadLocalUnitType.get());
        return new HashMap<String,String>(2){{
                    put("status","302");
                    put("location","/checkout/" + orderId+"/"+threadLocalRouterId.get());
        }};
    }
}
@Service
public class OrderDAO {
    @Autowired
    private CheckoutServiceInner checkoutService;
    public String checkout(String email, String streetAddress, String zipCode, String city, S
tring state, String country,
                           String creditCardNumber, int creditCardExpirationMonth, String cre
ditCardCvv, String userId, String unitType) {
        return checkoutService.checkout(email, streetAddress, zipCode,
                city, state, country,  creditCardNumber,
                creditCardExpirationMonth, creditCardCvv, userId,unitType);
    }
    @FeignClient(name = "checkoutservice")
    public interface CheckoutServiceInner {
        @PostMapping("/checkout0")
        String checkout(@RequestParam("email") String email,
                        @RequestParam("streetAddress") String streetAddress,
                        @RequestParam("zipCode") String zipCode,
                        @RequestParam("city") String city,
                        @RequestParam("state") String state,
                        @RequestParam("country") String country,
                        @RequestParam("creditCardNumber") String creditCardNumber,
                        @RequestParam("creditCardExpirationMonth") int creditCardExpirationMo
nth,
                        @RequestParam("creditCardCvv") String creditCardCvv,
                        @RequestParam("userId") String userId,
                        @RequestParam("unitType") String unitType
        );
    }
}
```

- **数据层**

  i. 执行以下命令，数据层引入msha-client。

```
<dependency>
   <groupId>com.aliyun.unit.router</groupId>
   <artifactId>msha-sdk-client</artifactId>
   <version>1.0.4-SNAPSHOT</version>
</dependency>
```

  ii. 启动参数指定 `ramrole` （也可选AK/SK的方式），以及ACM的Server地址ACMDOMAIN和命名空间ACMTENANT，这样msha-client才能从ACM获取单元规则。

> **说明** 如果应用不是在阿里云，则还需要手动指定Region-ID和Zone-ID（要与全局配置引导保持一致），这样msha-client才知道本地属于哪个单元。

```
java -Dspring.profiles.active=$UNITFLAG -Dram.role.name=acm-role
-Daddress.server.domain=$ACMDOMAIN
-Dtenant.id=$ACMTENANT
-jar /app/checkoutservice-provider-0.0.1-SNAPSHOT.jar
```

  iii. 数据层操作数据库时，调用msha-client传递多活参数。

```
@Override
    public String checkout(String email, String streetAddress, String zipCode, String cit
y, String state, String country,
                           String creditCardNumber, int creditCardExpirationMonth, String
creditCardCvv, String userId, String unitType) {
        Order order =new Order();
        UUID uuid = UUID.randomUUID();
        order.setOrderId(uuid.toString());
        order.setUserId(userId);
        //获取购物车商品。
        List<CartItem> items = cartDao.cleanCartItems(userId);
        List<ProductItem> productItems = new ArrayList<>();
        for (CartItem item : items) {
            productItems.add(new ProductItem(item.getProductID(), item.getQuantity(), ord
er.getOrderId(),item.getProductName()));
        }
        //校验库存。
        List<ProductItem> lockedProductItems = productDao.confirmInventory(productItems);
        //保存商品列表。
        order.setProductItemList(productItems);
        int lockedProductNum = 0;
        for (ProductItem item : lockedProductItems) {
            if (item.isLock()) {
                lockedProductNum++;
            }
        }
        if (lockedProductNum > 0) {
            //状态为1表示至少有一件商品购买成功。
            order.setStatus(1);
            //计算价格。
            //校验、保存地址。
            //生成订单，支付。
            //运输商品。
        } else {
```

```
            //表示所有商品都购买失败。
            order.setStatus(-1);
        }
        OrderForm orderForm = new OrderForm(order);
        logger.info("orderForm {} order {}",orderForm ,order);
        try {
            // 传递参数。
            RouterContextClient.setUnitContext(userId,unitType);
            orderFormRepository.save(orderForm);
        }catch (Exception e){
            logger.error("save order error",e);
            for (CartItem item : items) {
                // 购物车回滚。
                cartDao.addToCart(item, Long.valueOf(userId));
            }
            // 返回错误信息。
            return "error:"+e.getCause().getCause().getMessage();
        }
        // 清理。
        RouterContextClient.clearUnitContext();
        return order.getOrderId();
    }
```

功能演示

- **接入层分流**
  对不同ID用户，接入层会将用户分流到规则确定的单元。
- **单元保护**
  假设接入层请求打错了，数据层能够将该请求拦截，避免数据脏写。本文示例以**跳过接入层，直接访问服务**模拟这个过程。
- **切流**
  假设某个单元出了故障，您需要将用户流量切走，让另一个单元为其提供服务。本文示例以**挂掉杭州单元的productservice**来模拟故障，此时这些0~4990 ID的用户都无法使用服务，在MSHA控制台进行切流操作，这样这部分ID就能使用北京单元提供的服务了。
  可以看到购物车cartservice没做单元化，所以切流后的购物车就没数据了，但是订单checkoutservice做了单元化，所以切流后用户仍可以正常查看之前的订单，并进行新的下单操作。

5.4.2. 读多写少型业务场景多活实践

多活容灾MSHA（Multi-Site High Availability）是在阿里巴巴电商业务环境演进出的多活容灾架构解决方案。本文通过一个电商业务导购链路案例，介绍典型的读多写少型业务场景，如何基于多活容灾解决方案（AHAS-MSHA）帮助业务实现多活容灾架构。

背景信息

本文示例应用包含以下模块：

- frontend：入口Web应用，负责和用户交互。
- cartservice：购物车应用。记录用户的购物车数据，使用自建的Redis。
- productservice：商品应用。提供商品、库存服务，使用RDS MySQL。
- checkoutservice：下单应用。将购物车中的商品生成购买订单，使用RDS MySQL。

技术栈：

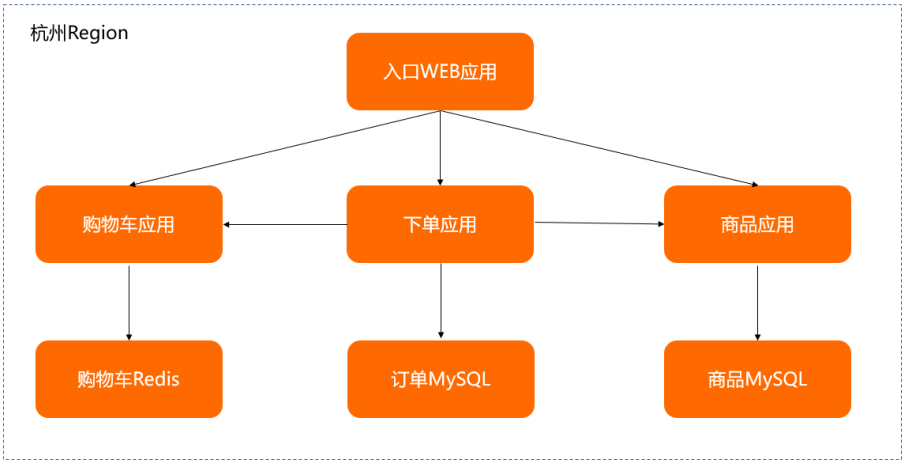
- SpringBoot。
- RPC框架：SpringCloud，注册中心使用自建的Eureka。

体验地址：

- 多活容灾控制台：
  i. 登录AHAS控制台。
  ii. 在控制台左侧导航栏单击**多活容灾**。
  iii. 在顶部菜单栏，**命名空间**选择**官方示例命名空间**。
- 电商业务页面：Demo。

案例背景：一次故障的发生

例如本示例的电商业务，包括导购、购物车、交易等业务场景。但在电商业务初期，很多互联网企业都没有考虑容灾问题，只在单地域进行了部署，部署的电商应用架构1.0如下图所示，只在杭州单元部署了相关业务。



与许多企业一样，该电商业务首次开始考虑容灾建设，是源于一次商品应用的故障，导致导购页面长时间无法访问，电商业务瘫痪。虽然故障最终得以解决，但故障导致的客户流失和企业口碑影响，对快速发展的业务造成不小的打击，迫使企业开始考虑容灾能力的建设。

这次故障中受损的导购业务，是典型的读多写少型业务场景，包括以下链路：

- 导购页面的展示，是读链路。
- 电商后台发布、上架商品（从而在导购页面能够展示出来），是写链路。

导购业务，用户关注的是导购页中的商品信息，通常不关注商品的上架过程。因此读链路是核心，而写链路是可以被接受短暂的不可用。这个读多写少的业务特点，就非常适合采用**异地多读**架构。读链路异地多活而写链路保持单点（单地域写），这样建设成本低、改造内容少、投入产出比高。所以接下来，我们将导购业务读链路相关的应用、中间件、数据库进行异地部署和多活改造。

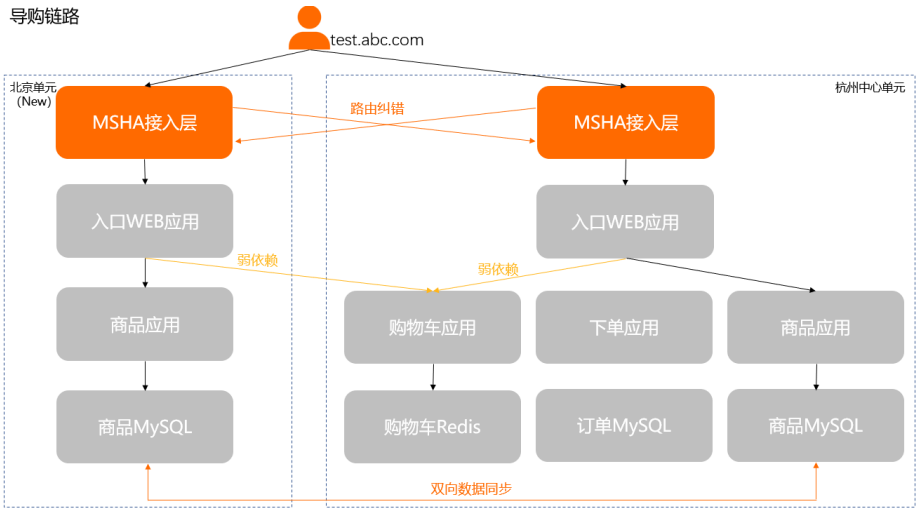
异地多读架构改造

基于MSHA多活容灾解决方案，可以快速的帮助业务进行异地多读容灾建设。

多活改造和MSHA接入包括以下方面：

- 分区维度：电商业务适合使用UserID来作分流标识。只需将域名流量Http Header/Cookie带上UserID标识，MSHA接入层就可以根据标识进行流量的路由。
- 改造范围：将导购链路相关的入口Web应用、商品应用以及商品MySQL数据库，在新地域进行冗余和对等的部署。
- 管控配置：进入MSHA控制台进行各层多活资源的配置（接入层域名/URI/SLB等，数据层数据同步等）。

改造后的应用架构如下图所示。



复现故障

改造完成容灾架构后，还需验证容灾能力是否符合预期。接下来将历史故障进行复现，通过制造真实的故障来验证容灾恢复能力。

1. **演练准备。**
   i. 登录AHAS控制台。
   ii. 在控制台左侧导航栏选择**多活容灾**。
   iii. 在左侧导航栏选择**监控大盘**，在顶部菜单栏，选择**命名空间**为**官方示例命名空间**。
   iv. 在**异地双活**区域，查看业务稳态监控指标。

   > **说明** 基于MSHA流量监控或其他监控能力，确定业务稳态的监控指标，以便在故障发生时判断故障影响面以及在故障恢复后判断业务的实际恢复情况。

   演练预期如下：
   - 导购链路对购物车应用是弱依赖（导购页会展示用户放入购物车的商品数量），弱依赖故障不影响业务。
   - 导购链路对商品应用是强依赖，强依赖故障将导致业务不可用，因此故障的爆炸半径应该控制在单元内。

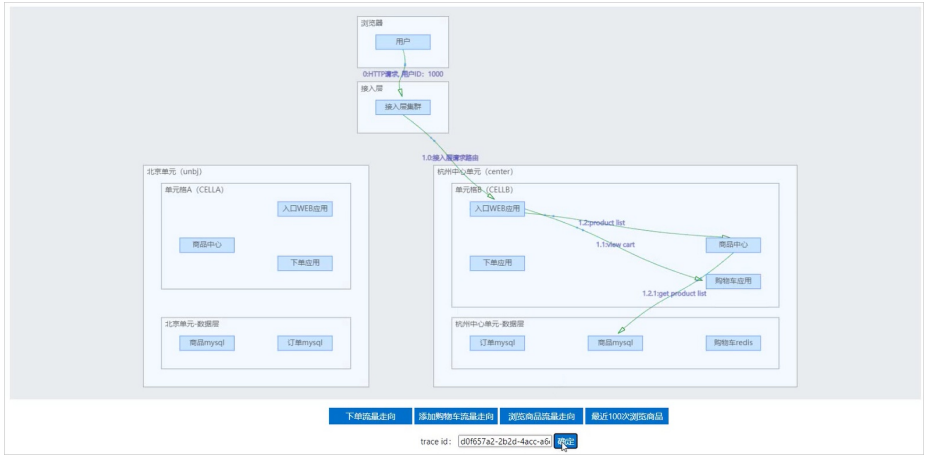
   v. 创建故障演练。

   创建杭州单元商品中心故障的演练，具体操作，请参见创建演练。
2. **故障注入。**

i. 在多活容灾的**监控大盘**页面**异地双活**区域，查看故障演练前配置的路由规则。

本示例中UserID为0~1575之间的用户会路由到杭州单元，UserID为1576~9999之间的用户会路由到北京单元。

示例的MSHA商城商品应用链路如下图所示，UserID为1000的用户路由到杭州单元。

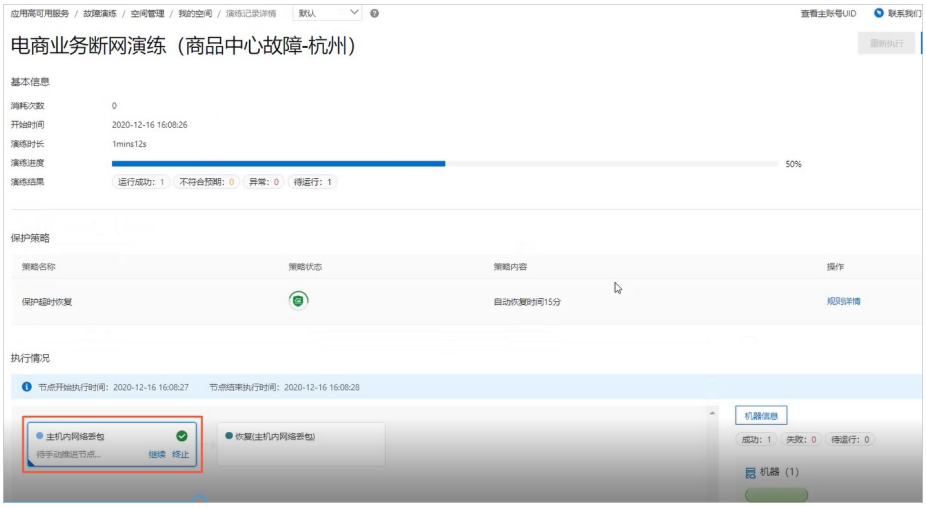


ii. 对杭州单元的商品应用注入故障。

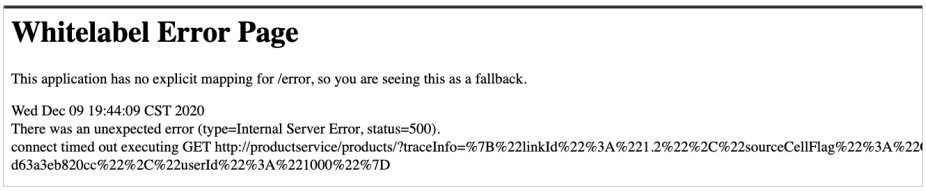
a. 在AHAS控制台的左侧导航栏选择**故障演练 > 我的空间**。

b. 在**我的空间**单击演练准备中创建的演练，然后单击**演练**。

c. 在**开始执行演练**对话框中单击**确认**。



若故障注入成功，UserID为1000的用户路由到的杭州单元会受到影响，导购页访问异常，符合预期。



d. （可选）验证爆炸半径。

验证爆炸半径是否控制在故障单元内：

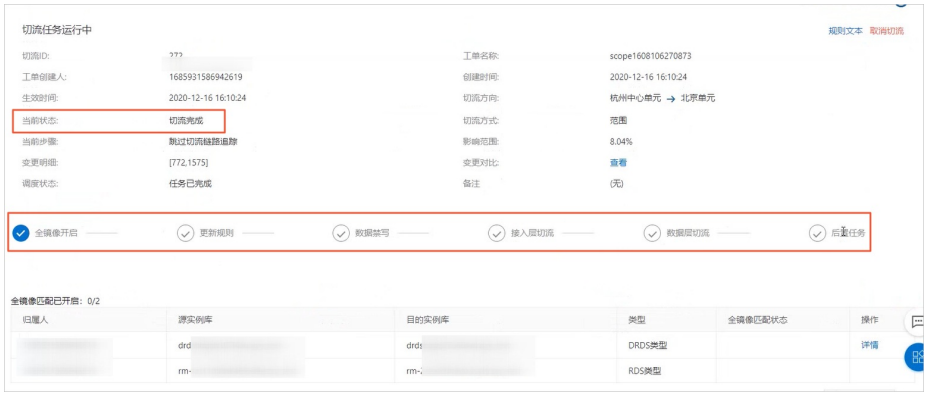
- 预期：UserID为2000的用户路由到北京单元，不受杭州单元故障的影响。
- 结果：导购页访问正常，符合预期。

切流恢复

接下来将验证故障场景下的容灾恢复能力。在杭州单元发生故障的情况下，可以使用MSHA切流功能将受影响的用户流量切换到另外的单元，进行快速业务恢复（这里区别于传统的思路，不是去排查、处理和修复故障，而是立即使用切流进行恢复，将业务恢复和故障恢复解耦）。

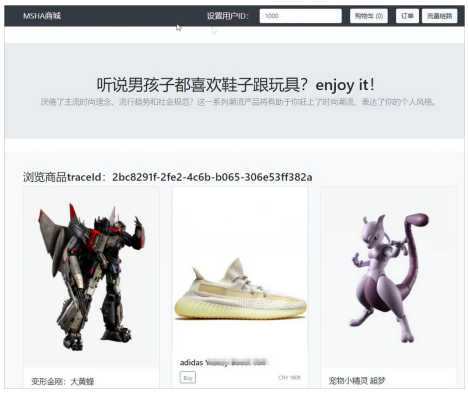
容灾切换预期：将UserID为1000的用户切流到北京单元，切流后该用户将路由到北京单元，不受杭州单元故障的影响。

1. 登录AHAS控制台。
2. 在控制台左侧导航栏中单击**多活容灾**。
3. 在左侧导航栏选择**基础配置 > 命名空间**，在顶部菜单栏选择**官方示例命名空间**。
4. 在多活容灾的左侧导航栏选择**切流 > 异地双活切流**。
5. 在**异地双活切流**页面，单击**切流**。
6. 在**切流详情**页面的**规则调整**区域，滑动**北京单元**的滑块，使得UserID 1000在**北京单元**的规则内。
7. 单击**生成预览**，然后在**生成**区域单击**执行预检查**，在**切流检查**区域，单击**确认**。
8. 在**切流确认**对话框中，单击**确定**。
   在**切流任务**页面的**当前状态**显示**切流完成**，表示切流已成功。

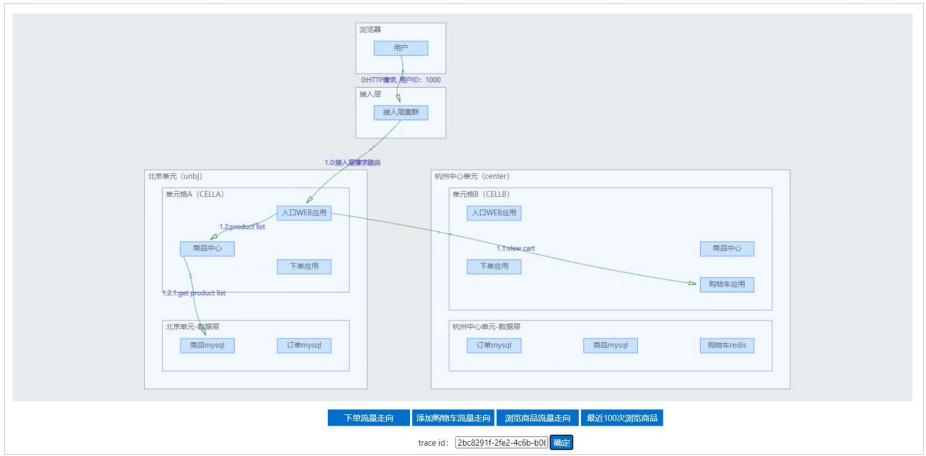


9. 刷新MSHA商城导购页。

   MSHA商城导购页可再次正常访问，符合预期。



   通过查看导购请求的实际调用链路，UserID为1000的用户已路由到北京单元，不受杭州单元故障的影响，符合预期。



功能演示

故障注入和切流恢复的功能演示如下。

-

后续步骤

您还需要进行以下操作：

- 终止注入故障，具体操作，请参见停止演练。
- 反馈演练结果，记录演练识别到的风险问题，具体操作，请参见反馈演练结果。
- 回切流量，将MSHA的路由规则恢复到演练前的状态，具体操作，请参见异地双活切流。

- 查看业务的稳态指标已恢复，具体操作，请参见本文复现故障。

相关文档

- 什么是故障演练
- 为什么需要多活容灾？

5.4.3. 流水单据型业务场景多活实践

多活容灾MSHA（Multi-Site High Availability）是在阿里巴巴电商业务环境演进出的多活容灾架构解决方案。本文通过一个电商业务下单链路案例，介绍典型的流水单据型业务场景，如何基于多活容灾解决方案（AHAS-MSHA）帮助业务实现多活容灾架构。

背景信息

本文示例应用包含以下模块：

- frontend：入口Web应用。负责和用户交互。
- cartservice：购物车应用。记录用户的购物车数据，使用自建的Redis。
- productservice：商品应用。提供商品、库存服务，使用RDS MySQL。
- checkoutservice：下单应用。将购物车中的商品生成购买订单，使用RDS MySQL。

技术栈：

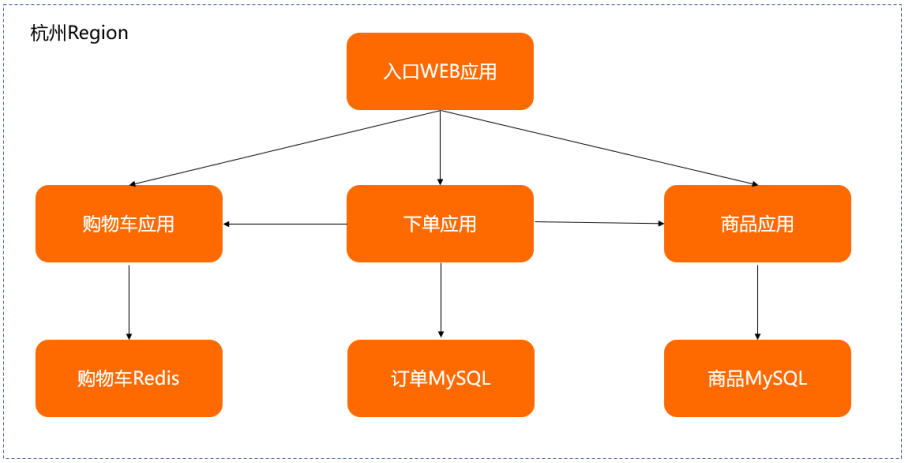
- SpringBoot。
- RPC框架：SpringCloud，注册中心使用自建的Eureka。

Demo体验地址：

- 多活容灾控制台：
  i. 登录AHAS控制台。
  ii. 在控制台左侧导航栏选择**多活容灾**。
  iii. 在顶部菜单栏，**命名空间**选择**官方示例命名空间**。
- 电商业务页面：Demo。

案例背景：一次故障的发生

例如本示例的电商业务，包括导购、购物车、交易等业务场景。但在电商业务初期，很多互联网企业都没有考虑容灾问题，只在单地域进行了部署，部署的电商应用架构1.0如下图所示，只在杭州单元部署了相关业务。



在读多写少型业务场景多活实践中，已经将导购链路进行了异地多读改造，而该业务后续在一次大促期间，遭遇了一次订单应用大面积故障，导致大促期间下单业务长时间无法使用，于是下单业务的容灾建设也提上了议程。下单业务是典型的流水单据型业务场景，相比导购，是更为复杂的读写业务，结合业务场景和业务容灾诉求，**异地多活**是适合此业务的容灾建设方案。

异地多活容灾架构改造

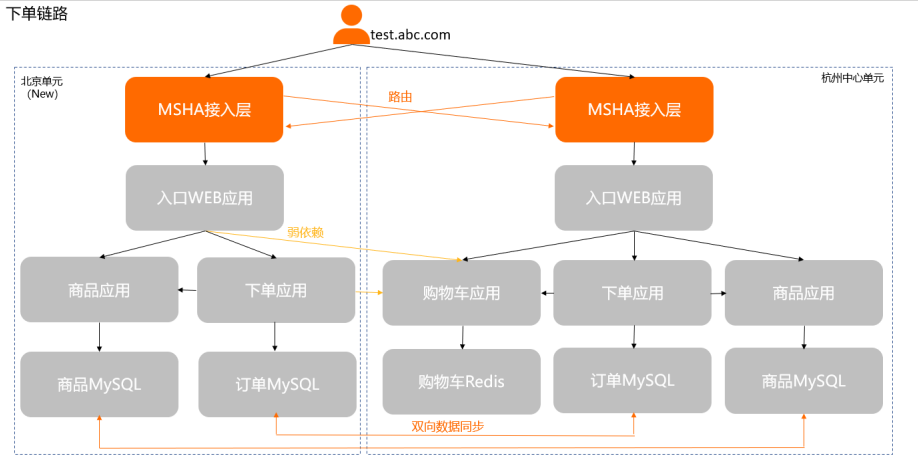
基于MSHA多活容灾解决方案，可以快速的帮助业务进行异地多活容灾建设。

> **说明** 下单链路强依赖购物车应用，完整的多活容灾建设，后续还应将购物车应用也改造为异地多活。

多活改造和MSHA接入包括以下方面：

- 改造范围：下单应用和订单数据库进行两地域部署。
- MSHA接入：将下单链路的应用安装上Agent，从而无侵入的实现SpringCloud RPC跨单元路由功能和数据防脏写功能。
- 管控配置：进入MSHA控制台进行各层多活资源的配置（接入层域名、URI、SLB、数据层数据同步等）。

改造后的应用架构如下图所示。



复现故障

改造完成容灾架构后，还需验证容灾能力是否符合预期，接下来将历史故障进行复现，通过制造真实的故障来验证容灾恢复能力。

1. **演练准备。**
   i. 登录AHAS控制台。
   ii. 在控制台左侧导航栏选择**多活容灾**。
   iii. 在左侧导航栏选择**监控大盘**，在顶部菜单栏，选择**命名空间**为**官方示例命名空间**。
   iv. 在**异地双活**区域，查看业务稳态监控指标。

   > **说明** 基于MSHA流量监控或其他监控能力，确定业务稳态的监控指标，以便在故障发生时判断故障影响面以及在故障恢复后判断业务的实际恢复情况。

   演练预期如下：
   - 下单链路对订单应用是强依赖，强依赖故障会影响业务不可用。
   - 故障爆炸半径控制在单元内。

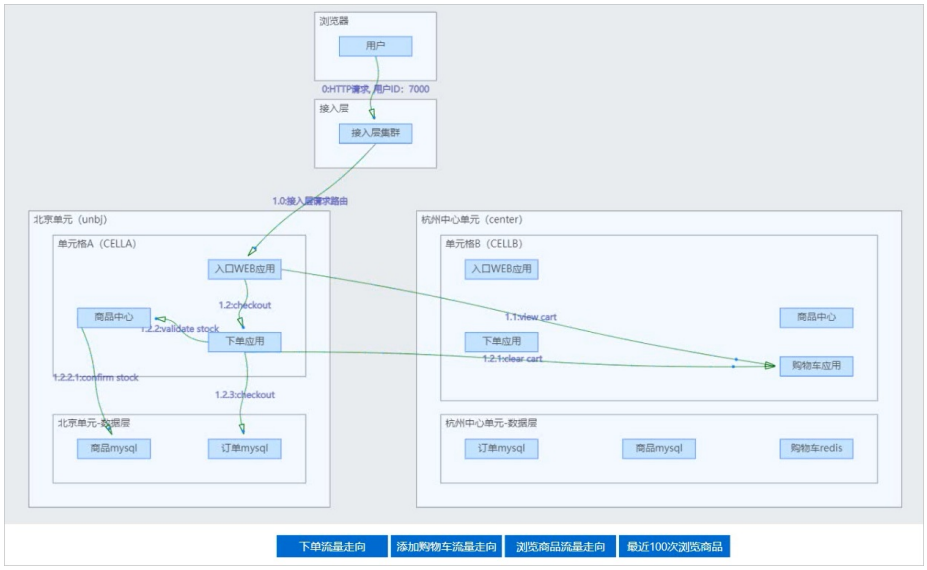
   v. 创建故障演练。
   创建北京单元下单应用故障的演练，具体操作，请参见创建演练。

2. **故障注入。**

   i. 在多活容灾的**监控大盘**页面**异地双活**区域，查看故障演练前配置的路由规则。

   本示例中UserID为0~6652之间的用户会路由到杭州中心单元，UserID为6653~9999之间的用户会路由到北京单元。

   示例的MSHA商城下单应用链路如下图所示，当UserID为7000的用户路由到北京单元。



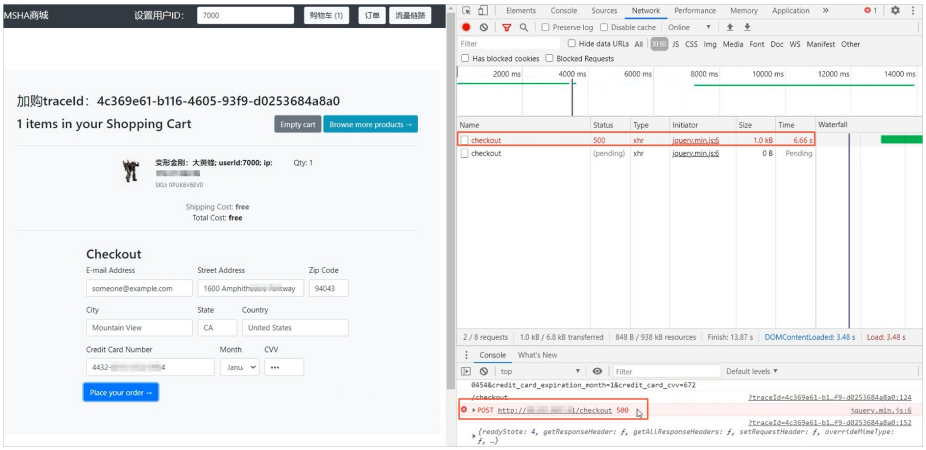
ii. 对北京单元的下单应用注入故障。

a. 在AHAS控制台的左侧导航栏选择**故障演练 > 我的空间**。

b. 在**我的空间**单击演练准备中创建的演练，然后单击**演练**。

c. 在**开始执行演练**对话框中单击**确认**。

若故障注入成功，UserID为7000的用户路由到的北京单元会受到影响，下单页访问异常，符合预期。



d. （可选）验证爆炸半径。

验证爆炸半径是否控制在故障单元内：

- 预期：UserID为2000的用户路由到杭州单元，不受北京单元故障的影响。
- 结果：下单正常，符合预期。

切流恢复

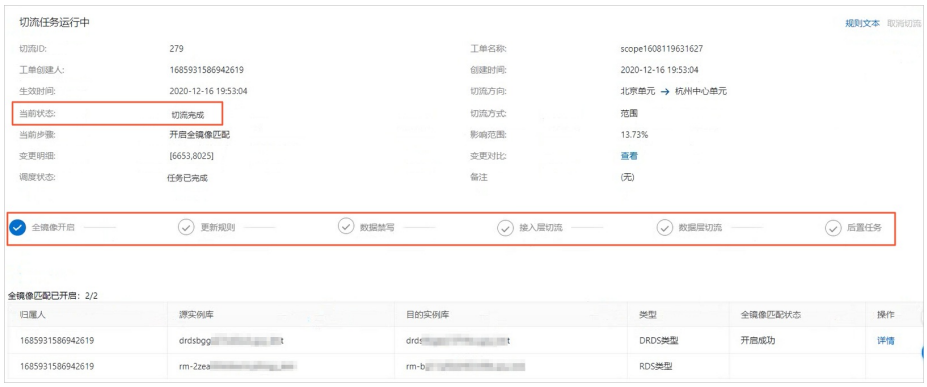
验证故障场景下的容灾恢复能力。在北京单元发生故障的情况下，可以使用MSHA切流功能将受影响的用户流量切换到另外的单元，进行快速业务恢复。

> **说明** 这里区别于传统的解决思路，不是去排查、处理和修复故障，而是立即使用切流进行恢复，将业务恢复和故障恢复解耦。

容灾切换预期：将UserID为7000的用户切流到杭州单元，切流后该用户将路由到杭州单元，不受北京单元故障的影响。

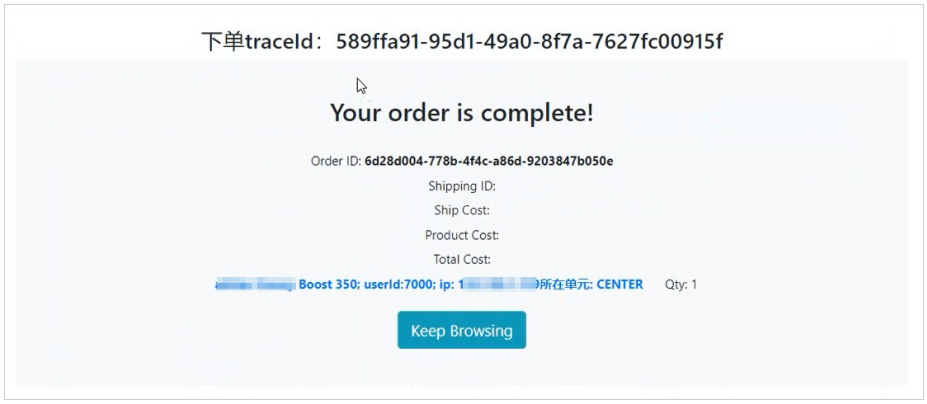
1. 登录AHAS控制台。
2. 在控制台左侧导航栏中单击**多活容灾**。
3. 在左侧导航栏选择**基础配置 > 命名空间**，在顶部菜单栏选择**官方示例命名空间**。

4. 在多活容灾的左侧导航栏选择**切流 > 异地双活切流**。
5. 在**异地双活切流**页面，单击**切流**。
6. 在**切流详情**页面的**规则调整**区域，滑动**杭州中心单元**的滑块，使得UserID 7000在**杭州中心单元**的规则内。
7. 单击**生成预览**，然后在**生成**区域单击**执行预检查**，在**切流检查**区域，单击**确认**。
8. 在**切流确认**对话框中，单击**确定**。
   在**切流任务**页面的**当前状态**显示**切流完成**，表示切流已成功。

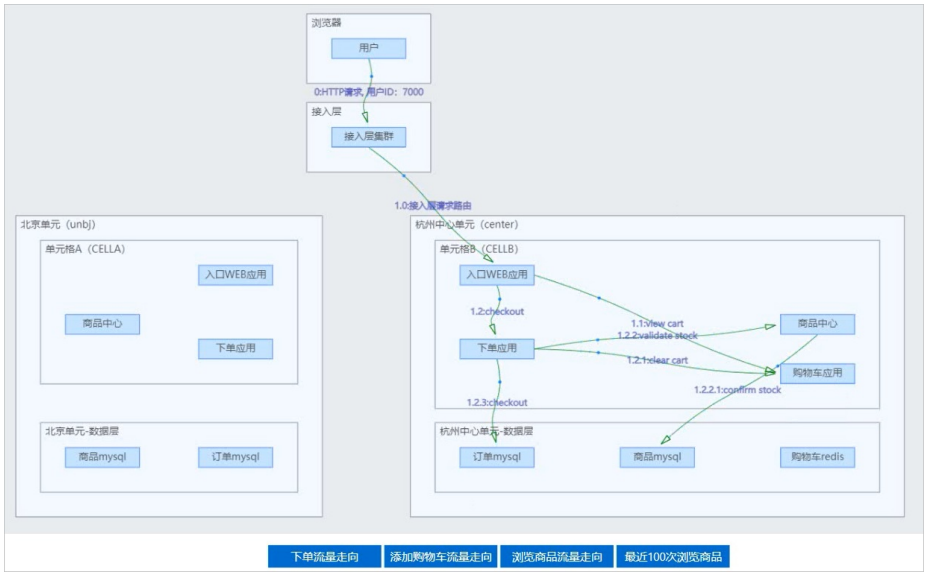


9. 重新在MSHA商城下订单。

   MSHA商城下单正常，符合预期。



通过查看下单请求的实际调用链路，UserID为7000的用户已路由到杭州单元，不受北京单元故障的影响，符合预期。



功能演示

故障注入和切流恢复的功能演示如下。

-

后续步骤

您还需要进行以下操作：

- 终止注入故障，具体操作，请参见停止演练。
- 反馈演练结果，记录演练识别到的风险问题，具体操作，请参见反馈演练结果。
- 回切流量，将MSHA的路由规则恢复到演练前的状态，具体操作，请参见异地双活切流。
- 查看稳态业务指标已恢复，具体操作，请参见本文复现故障。

相关文档

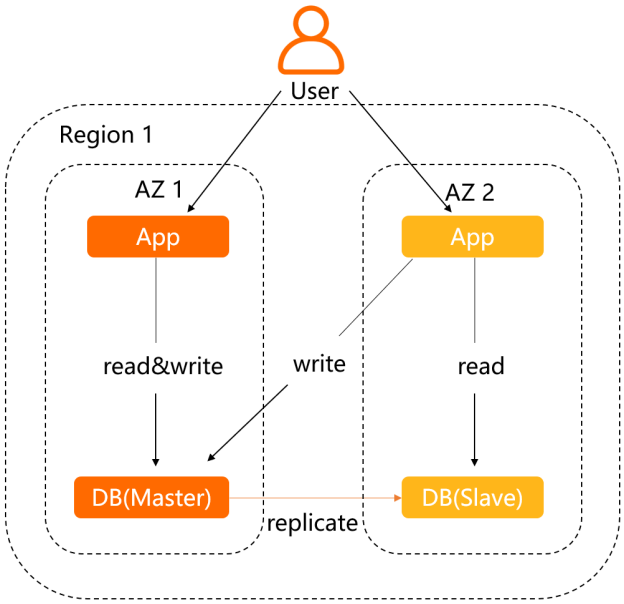
- 什么是故障演练
- 为什么需要多活容灾？

5.4.4. 同城多活架构实践

多活容灾MSHA（Multi-Site High Availability）是一个云原生的多活容灾架构解决方案。本文介绍同城多活容灾架构的建设原则和难点，并通过一个电商业务案例，介绍如何基于MSHA来快速、无侵入的帮助业务实现同城多活容灾架构。

同城多活架构介绍

同城多活（DB主备）的架构图如下：



同城多活架构包含以下主要特征：

- 应用可用区级多活。
- 数据库跨可用区主备。
- RPO：分钟级（AZ级故障）。
- RTO：分钟级（AZ级故障）。

应用场景：

- 针对可用区级的故障、灾难，期望业务具备分钟级恢复能力的场景。

- 应用多可用区部署的情况下，期望RPC调用可用区内封闭，以避免跨可用区网络请求带来的RT增长。

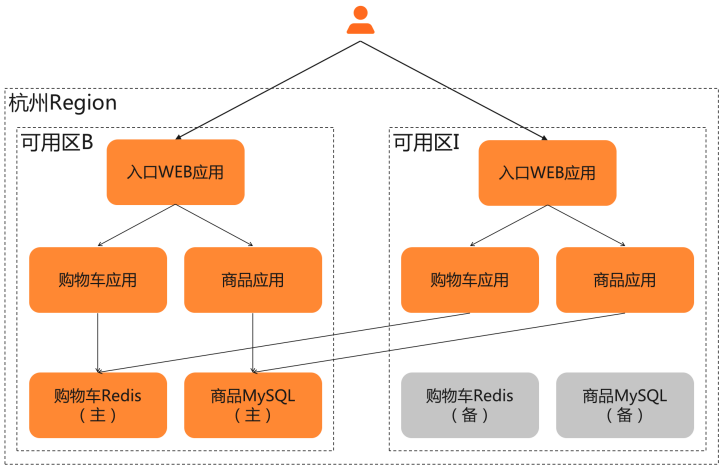
建设原则：

- 保证冗余。
- 保持对等。
- 保持封闭。

建设难点：

- 流量管理难度高。
  - 当发生可用区级故障、灾难时：
    - 需具备HTTP、HTTPS流量的可用区级动态调配分流能力。
    - 需具备对故障AZ的RPC、MQ、任务调度流量切零能力。
  - 如果业务RT敏感，需具备可用区内流量封闭的能力以避免跨可用区的网络传输带来的RT增长。
- 统一管控难度大。
  - 需对接支持众多的云产品和开源框架。
  - 切零规则、流量可用区内封闭规则、环境隔离规则、甚至异地多活路由规则共存的情况下，需保证规则的优先级、兼容性和冲突解决策略。
- 业务无侵入难度大：要实现HTTP、RPC、MQ、任务调度等流量管控能力，通常需要业务应用配合改造，对业务代码侵入大。

业务背景信息



本示例的电商业务包含以下应用：

- frontend：入口Web应用，负责和用户交互。
- cartservice：购物车应用。记录用户的购物车数据，使用自建的Redis。
- productservice：商品应用。提供商品、库存服务，使用RDS MySQL。

技术栈：

- SpringBoot。
- RPC框架：SpringCloud，注册中心使用自建的Eureka。

案例背景：一次故障的发生

本示例的电商业务已自行进行了同城容灾能力建设，在杭州的多个可用区进行应用的对等部署，并自行实现了可用区级粒度的入口流量控制。但因为一次线上可用区级故障，才发现将故障可用区的HTTP流量切换到其他可用区后，下游的RPC、MQ调用仍然有概率访问到故障可用区内的机器，业务仍然无法使用，导致电商页面长时间无法访问，甚至电商业务瘫痪。

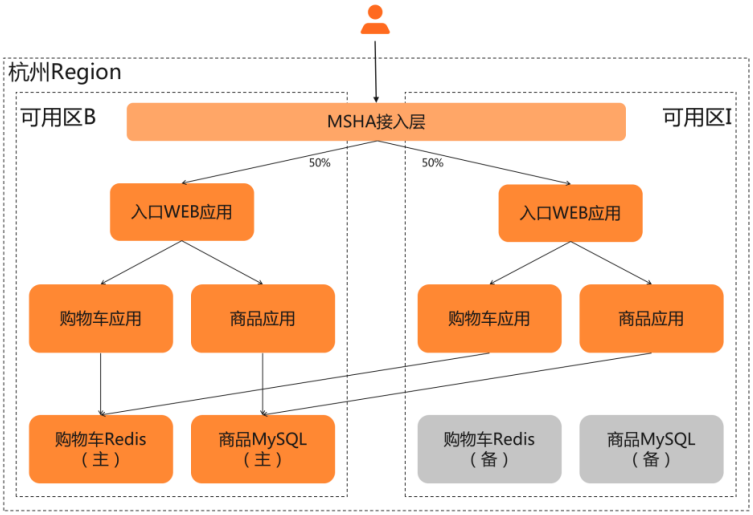
虽然故障最终得以解决，但故障导致的客户流失和企业口碑影响，对快速发展的业务造成不小的打击，迫使企业开始重视同城多活容灾能力的建设，以及定期做故障演练确保故障恢复能力的有效性。

同城多活架构改造

基于MSHA多活容灾解决方案，您可以快速地对业务进行同城多活容灾架构建设。

多活改造和MSHA接入包括以下方面：

1. 开通并配置MSHA需要启用的多活模块。具体操作，请参见开通并配置MSHA。
2. 创建同城多活的命名空间，选择容灾架构类型为同城多活，将应用部署所在的2个可用区定义为2个单元格。具体操作，请参见新建命名空间。
3. 安装MSHA探针。具体操作，请参见为Java应用手动安装探针。
4. 配置接入层MSFE。具体操作，请参见配置MSFE。
   改造后的应用架构如下图所示：



复现故障

基于MSHA完成同城多活架构建设后，还需验证容灾能力是否符合预期。接下来将历史故障进行复现，通过制造真实的故障来验证容灾恢复能力。

1. **演练准备。**
   i. 登录AHAS控制台。
   ii. 在控制台左侧导航栏单击**多活容灾**。
   iii. 在左侧导航栏单击**监控大盘**，在顶部菜单栏，选择**命名空间**为**官方示例命名空间**。

iv. 在**同城多活**区域，查看业务监控指标。

> **说明** 基于MSHA流量监控或其他监控能力，确定业务稳态的监控指标，以便在故障发生时判断故障影响面以及在故障恢复后判断业务的实际恢复情况。

演练预期：电商首页展示的查询链路对商品应用是强依赖，强依赖故障将导致业务不可用，且故障的爆炸半径应该控制在单元格内。

v. 创建故障演练。

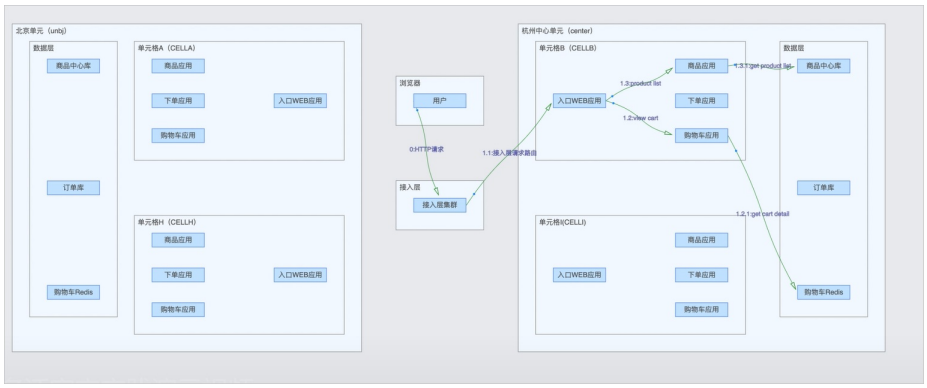
创建杭州单元格B下的商品应用故障演练（例如网络丢包）。具体操作，请参见创建演练。

2. **故障注入。**

i. 在多活容灾的**监控大盘**页面**同城多活**区域，查看故障演练前配置的路由规则。

本示例中杭州地域的各个单元格流量比例为50%。

示例的MSHA商城商品应用链路如下图所示，UserID为1000的用户路由到单元格B。



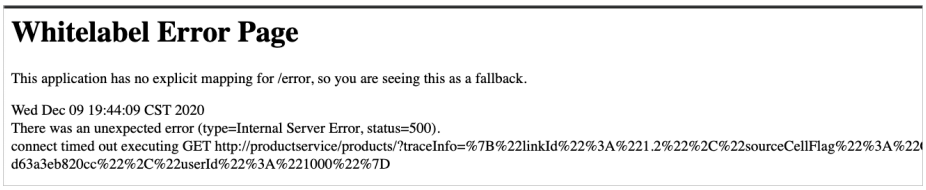
ii. 对杭州域单元格B下的商品应用注入故障。

a. 在AHAS控制台的左侧导航栏选择**故障演练 > 我的空间**。

b. 在**我的空间**单击演练准备中创建的演练，然后单击**演练**。

c. 在**开始执行演练**对话框中，单击**确认**。

若故障注入成功，打开电商首页，有概率出现访问异常，符合预期。

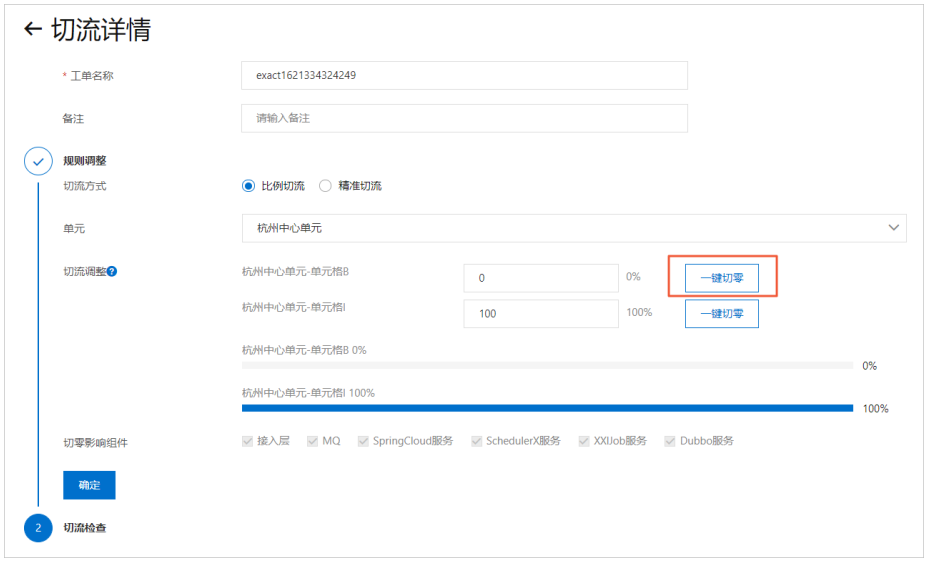


切流恢复

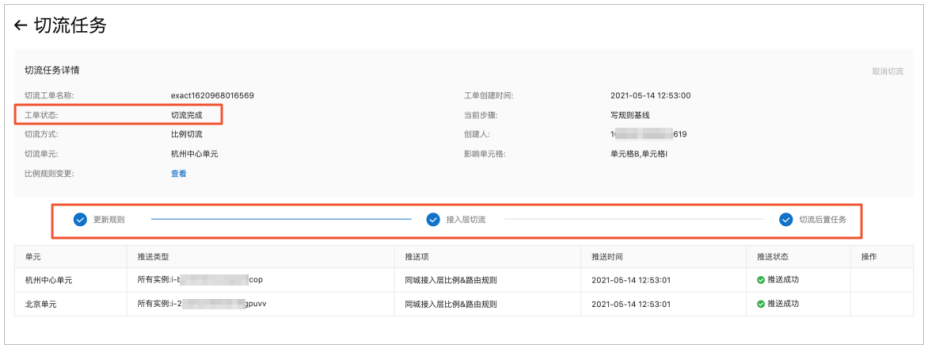
接下来将验证故障场景下的容灾恢复能力。在杭州单元格B的商品应用发生故障的情况下，可使用MSHA切流功能将流量全部切换到另外的单元格，进行快速业务恢复（这里区别于传统的思路，不是去排查、处理和修复故障，而是立即使用切流进行恢复，将业务恢复和故障恢复解耦）。

容灾切换预期：将100%的流量比例都切换到单元格I，切流后业务完全恢复，不受单元格B的故障影响。

1. 登录AHAS控制台。
2. 在控制台左侧导航栏中单击**多活容灾**。
3. 在左侧导航栏选择**基础配置 > 命名空间**，在顶部菜单栏选择**官方示例命名空间**。
4. 在多活容灾的左侧导航栏选择**切流 > 同城多活切流**。
5. 在**同城多活切流**页面，单击**切流**。
6. 在**切流详情**页面的**规则调整**区域，选择**切流方式**为**比例切流**，从**单元**下拉列表中选择**杭州中心单元**，单击**杭州中心单元-单元格B**的**一键切零**。



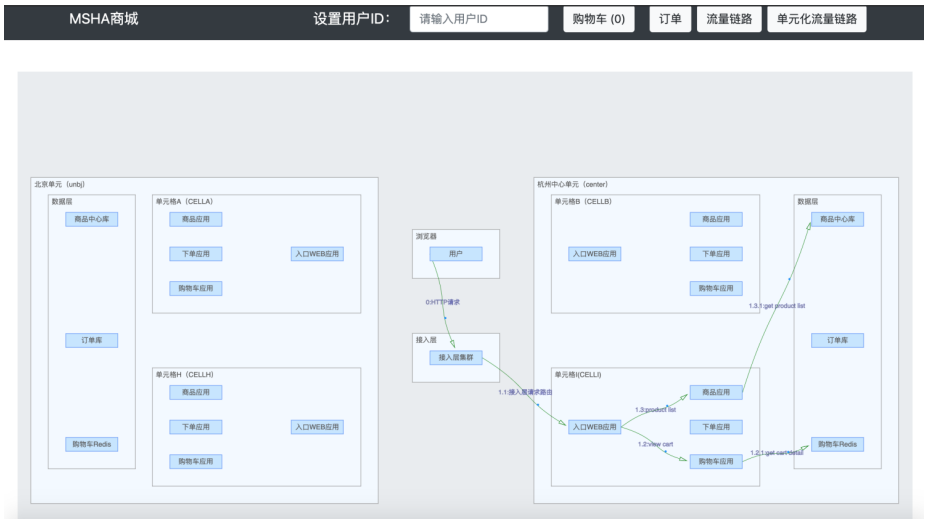
7. 单击**确定**。在**切流检查**区域检查完成后，单击**切流**。
8. 在**切流任务**页面，查看**工单状态**为**切流完成**。表示切流已成功。



验证结果：

i. 刷新MSHA商城首页，多次访问均能正常展示，符合预期。

ii. 通过查看实际调用链路，首页的流量始终访问到杭州单元格I中，不受单元格B故障的影响，符合预期。



功能演示

故障注入和切流恢复的功能演示如下。

后续步骤

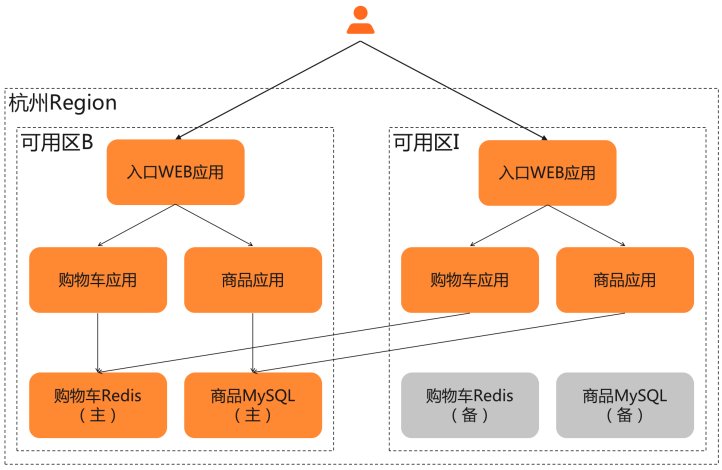
您还需要进行以下操作：

- 终止注入故障。具体操作，请参见停止演练。
- 反馈演练结果，记录演练识别到的风险问题。具体操作，请参见反馈演练结果。
- 回切流量，将单元格B和单元格I的流量比例，通过再次切流恢复到各50%（演练前的状态）。具体操作，请参见同城多活切流。
- 在控制台的监控大盘，查看业务的稳态指标已恢复。

5.4.5. 同城流量封闭实践

多活容灾MSHA（Multi-Site High Availability）是一个云原生的多活容灾架构解决方案，同城流量封闭是指调用都在同一可用区内进行，减少和避免跨可用区调用带来的响应时间增长。本文通过一个电商业务案例，介绍如何进行同城流量封闭的操作。

业务背景信息



本示例的电商业务包含以下应用：

- frontend：入口Web应用，负责和用户交互。
- cartservice：购物车应用。记录用户的购物车数据，使用自建的Redis。
- productservice：商品应用。提供商品、库存服务，使用RDS MySQL。

技术栈：

- SpringBoot。
- RPC框架：SpringCloud，注册中心使用自建的Eureka。

案例背景

本示例电商业务在进行了多可用区部署后，避免了在一个可用区部署的单点风险。也因此导致应用间的RPC调用会出现跨可用区的情况。跨可用区的网络传输带来了RT的延迟，导致电商页面加载和操作出现明显的响应慢和卡顿现象。为了保障用户体验，业务开始考虑如何减少、避免跨可用区调用来避免RT的猛增。

假设同地域的网络双向延迟约为5 ms，若一个调用链包含100次RPC调用，则最多可能出现100次跨可用区调用，最大RT延迟为500 ms（5 ms×100）。

同城多活架构改造

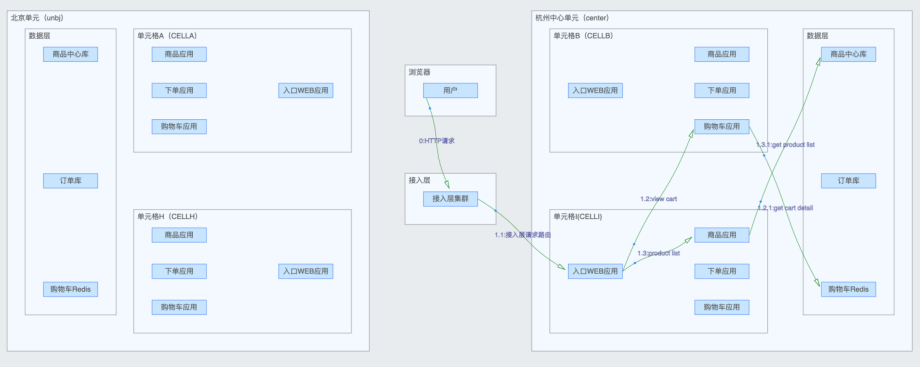
同城多活架构改造，请参见同城多活架构实践。

开启单元格流量封闭策略

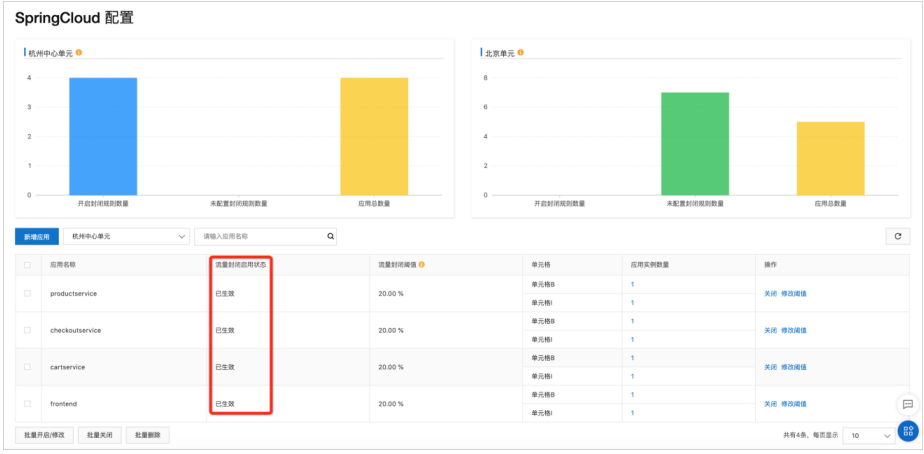
MSHA提供的同城多活架构能力，除了容灾恢复（同城切流）能力外，还针对常见RPC框架（HSF、SpringCloud、Dubbo）提供了单元格流量封闭的能力。下面将电商业务的各个应用，均配置和启用阈值为20%的单元格流量封闭策略。

> **说明** 阈值可选范围为[0,100]。阈值表示：当一个单元格内应用健康的机器和总机器占比≥${阈值} 时，则流量封闭策略生效。

1. 封闭策略开启前，验证是否存在跨单元格（可用区）的RPC调用。
   i. 登录AHAS控制台。
   ii. 在控制台左侧导航栏中单击**多活容灾**。
   iii. 在左侧导航栏选择**同城多活 > SpringCloud配置**，在顶部菜单栏选择**官方示例命名空间**。
   iv. 在**SpringCloud配置**页面，确认各个应用**流量封闭启用状态**为**未配置**。
   v. 在MSHA商城，查看导购请求的实际调用链路，存在跨单元格（可用区）的RPC调用。



2. 封闭策略开启后，验证是否存在跨单元格（可用区）的RPC调用。
   i. 登录AHAS控制台。
   ii. 在控制台左侧导航栏中单击**多活容灾**。
   iii. 在左侧导航栏选择**同城多活 > SpringCloud配置**，在顶部菜单栏选择**官方示例命名空间**。
   iv. 在**SpringCloud配置**页面，选中所有应用，单击**批量开启/修改**。
   v. 在**修改阈值**的文本框输入**20**，然后单击**确定**。
   所有应用开启流量封闭状态，如下图所示。

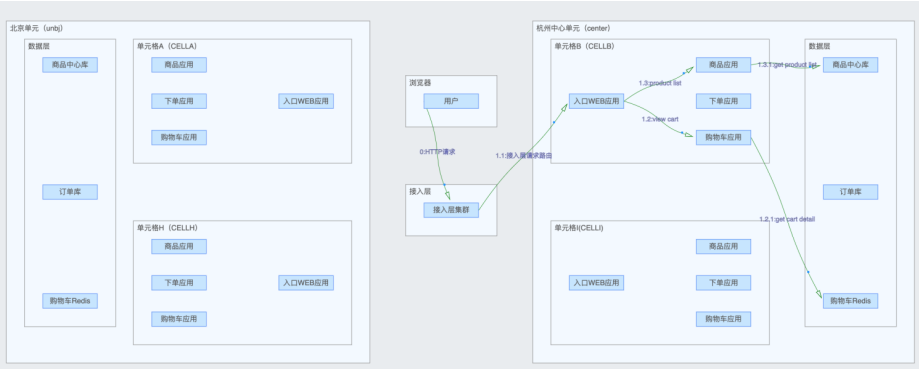


vi. 在MSHA商城，查看导购请求的实际调用链路，不存在跨单元格（可用区）的RPC调用。

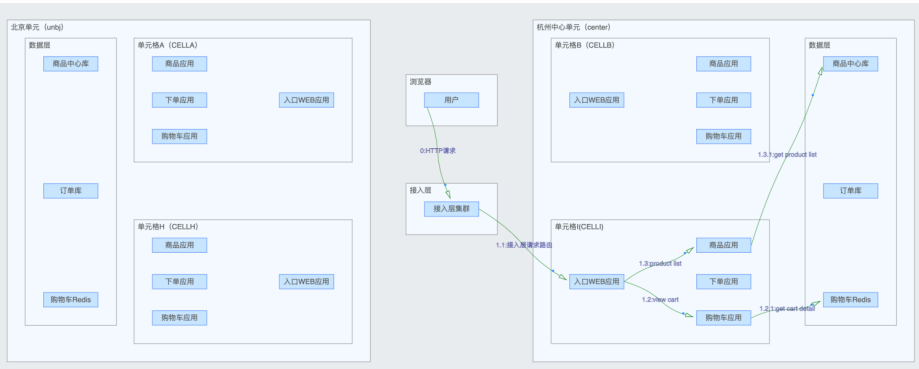
路由规则如下：

- 流量按比例（50%）分流到2个单元格的入口Web应用。
- RPC调用在单元格（可用区）内封闭。

RPC调用在单元格B内封闭如下图所示：



RPC调用在单元格I内封闭如下图所示：



5.4.6. 业务流量隔离功能实践

多活容灾MSHA（Multi-Site High Availability）是一个云原生的多活容灾架构解决方案，提供包括HTTP和RPC在内的业务流量隔离功能。本文通过一个电商业务案例来介绍MSHA的业务流量隔离功能。

背景介绍

业务流量隔离功能包含以下四个适用场景：

- 灰度发布
  传统的金丝雀发布、滚动发布和分批发布采用的思路均是先部署少量机器进行观察，再逐步部署剩余机器，以此来控制变更发布上线的风险。这种方式通常不具备精确引流和中间件隔离的能力。而灰度发布是通过搭建业务流量隔离的灰度环境，按需将流量引流进灰度环境机器，便于进行精准小流量的验证和观测。由于灰度发布只是发布流量中的一个短暂验证阶段，又因为每次灰度发布所需的业务流量可能不一致，因此建议开发人员按需执行灰度发布，发布完毕后释放业务流量。
- 安全生产环境
  区别于一般的灰度发布，安全生产环境是搭建一套与生产环境中间件隔离的环境，包含独立配套的监控告警系统，便于灰度发布、故障演练、链路压测、算法调优等，能有效优化研发流程和线上产品服务稳定性。由于中间件隔离，上游应用需接入安全生产环境后，下游应用才会接收到流量，因此安全生产环境搭建好之后，需要常态化的保留和运行。
- 线上专属环境

- 预案、故障演练
  通过在流量隔离的环境进行风险预案演练以及故障演练，可保证安全地进行不宜在线上进行的高风险操作和演练验证，常态化地进行反脆弱建设验证，保证系统容灾容错的能力。
- VIP业务流量重保
  可以通过搭建流量隔离环境，将VIP业务流量引流到专属的隔离环境，来做重保护航，避免系统上不同业务的资源抢占和故障影响，例如在大促期间对重点商家、VIP客户流量做重保护航。

- 日常多项目开发、测试
  当一个系统或应用存在多个项目并行进行迭代开发时，如果共用一个开发、测试环境，往往会由于不同分支改动互相干扰、代码合并部署协同成本高、交付时间周期不同等因素，导致日常开发、联调、测试效率低的问题。通过搭建多版本流量隔离环境，在各自独立的环境开发、联调和测试，可以解决多项目并行开发导致的互相冲突干扰的问题，大大提高开发效率。

上述业务流量隔离场景所涉及的建设难点主要包括：

- 灵活流量控制难
  - 若期望业务流量隔离环境与生产环境共用一套中间件，则需在中间件层面植入流量隔离、引流、路由逻辑。
  - 若期望业务流量隔离环境与生产环境不共用一套中间件，为避免在业务隔离环境完整部署全链路应用（否则会因为下游应用无可用机器而造成调用失败的问题），需要在中间件层面植入兜底调用回生产环境的逻辑。
- 统一管控难
  - 要实现HTTP、RPC、MQ、任务调度等流量的隔离，需对接支持众多云产品和开源框架。
- 无业务代码侵入难
  - 要实现HTTP、RPC、MQ、任务调度等流量管控能力，通常需要业务应用配合改造，对业务代码侵入大。
- 和容灾流量规则共存时，兼容不冲突难
  - 多套流量规则（业务流量隔离规则、同城多活比例规则、可用区内流量封闭规则、异地多活路由规则）共存的情况下，需考虑规则的优先级、互斥性和兼容性。

根据上述业务流量隔离适用场景和建设难点，现有两种实现方案：

- 方案1：隔离环境和线上普通环境共用一套中间件
  该方案需要具备入口流量与中间件（RPC、MQ、调度任务等）的流量隔离能力。灰度发布、线上专属环境和日常多项目开发、测试场景通常采用该方案来实现。
- 方案2：隔离环境和线上普通环境使用独立中间件
  该方案需要具备无下游应用时兜底调用（RPC、MQ、调度任务等）到线上普通环境的能力，否则需要上下游应用全量部署，且需要运维保障第二套线上环境的稳定性。而要实现兜底调用，则需要具备中间件数据同步和流量隔离的能力。安全生产环境场景通常采用该方案来实现。

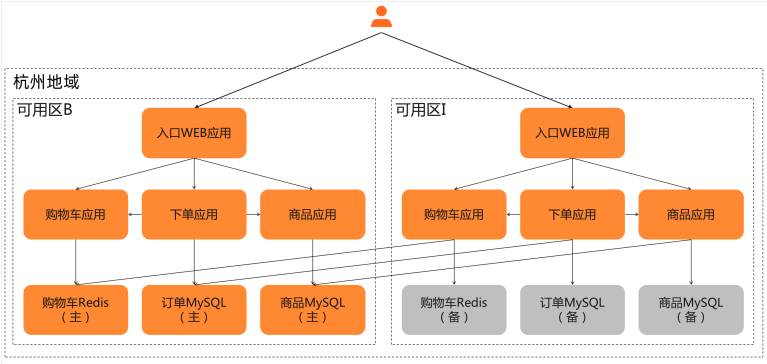
案例实践二：安全生产环境

案例实践

本文采用一个电商业务案例来介绍业务流量隔离功能的最佳实践。

该电商业务案例包含以下应用：

- frontend：入口Web应用，负责和用户交互。
- cartservice：购物车应用。记录用户的购物车数据，使用自建的Redis。
- productservice：商品应用。提供商品、库存服务，使用RDS MySQL。
- checkoutservice：下单应用。将购物车中的商品生成购买订单，使用RDS MySQL。



所涉及技术栈：

- SpringBoot
- RPC框架：SpringCloud（注册中心使用自建的Eureka）

案例体验地址：多活容灾MSHA控制台

1. 登录AHAS控制台。
2. 在控制台左侧导航栏，选择**多活容灾**。
   在控制台进行操作时，**命名空间**需选择**官方示例命名空间**。

业务需求

从历史故障来看，超过一半的故障是由发布变更导致的。如果发布流程增加灰度流程以及灰度时长，那么这些故障是有机会提前被发现和避免的。因此为了降低代码或配置变更后一次性全量发布上线可能造成的影响，业务期望建设灰度发布的能力，能做到随时灰度、灵活灰度（精准引流）。其具体需求内容包括：

- 针对变更涉及的一个或多个应用，在灰度发布阶段能够在生产环境扩容一批机器，并划分成为灰度环境。灰度环境默认不会有任何流量进入（包括HTTP、RPC、MQ、定时任务调度流量）。
- 支持自定义灰度环境引流规则，仅满足引流条件的流量才可进入灰度环境机器。
- RPC调用下游应用时，进入灰度环境的流量需要在该环境内封闭。但如果下游应用在灰度环境无可用机器，则兜底调用到线上环境。
- 发布流程结束后，释放灰度环境机器，实现按需使用、按需付费。

基于MSHA实现精准流量灰度的方案

步骤一：搭建灰度环境

1. 将需要灰度的应用扩容出1～2个实例，作为灰度环境。
2. 基于MSHA单元格模型，创建一个新的单元格作为灰度环境逻辑区域。
3. 将上述应用实例划分归属到灰度环境单元格。
4. 单元格配置为开启流量隔离，以保证默认情况下不会有任何流量进入。

> 说明
> - 单元格：是一个逻辑区域的概念。MSHA基于单元格的粒度进行流量控制，支持按可用区或指定JVM启动参数的方式将应用实例划分归属到对应的单元格。
> - 开启流量隔离：配置为开启流量隔离的单元格，流量比例固定为0%。在不配置引流规则的情况下不会有任何HTTP、RPC、MQ、定时任务调度流量进入。

步骤二：配置灰度引流规则

1. 创建新的SLB实例，用于对灰度环境的入口应用实例进行负载均衡。
2. 基于MSHA的域名统一接入能力以及HTTP流量按单元格分流的能力，在MSHA控制台将灰度环境单元格对应的回源地址配置为上述创建的SLB VIP。

3. 基于MSHA的HTTP和RPC引流能力，按需配置引流规则，从而让精准流量进入灰度环境。

步骤三：发布完成，释放资源

1. 在灰度发布结束后，一键关闭所有灰度环境引流规则。
2. 删除灰度环境单元格。
3. 释放灰度环境应用实例。

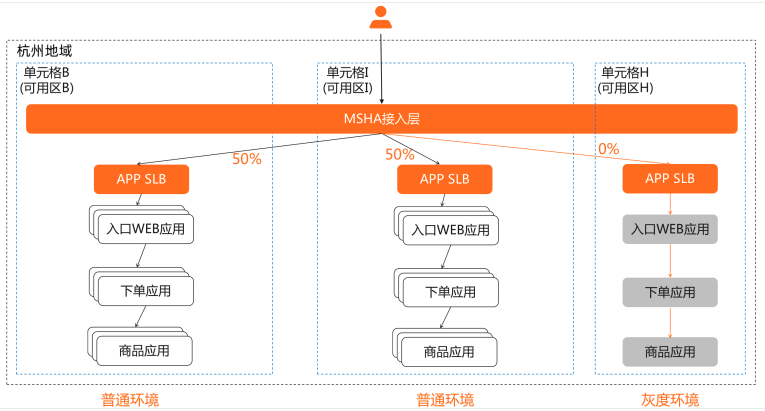
接入MSHA实践

场景一：HTTP流量灰度引流

为了满足业务需求，电商系统增添了一个新功能。此次变更涉及入口Web应用、下单应用和商品应用。完成变更后，需将这3个应用发布到灰度环境进行灰度观察和验证。

接入MSHA的操作流程如下：

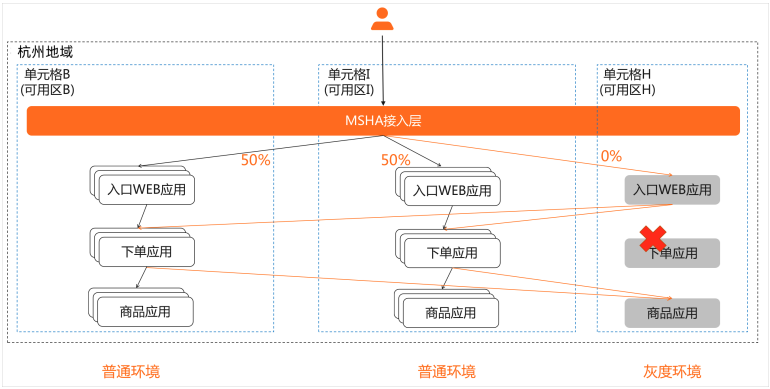
1. 扩容机器并创建一个新的单元格，作为灰度环境（如下图中单元格H）。
2. 将变更涉及的2个应用扩容出机器并划分归属到单元格H。
3. 创建VPC类型的SLB实例，配置监听HTTP 80端口，后端服务器组选择单元格H内的入口Web应用。
4. 配置域名、URI，并配置每个单元格对应的回源IP、Port信息。其中单元格H的回源IP、Port需要配置为上述创建的SLB VIP，Port为80。
5. 配置HTTP引流规则并开启流量染色，以便RPC调用下游应用时，流量能够根据染色标在灰度环境内封闭。



完成操作后可从以下四个方面验证灰度能力：

1. HTTP引流验证：配置HTTP引流规则，多次刷新电商页面并观察监控和日志，查看是否所有用户ID为10的流量都进入了单元格H。
2. 流量全链路染色和路由能力验证：配置HTTP引流规则和链路染色，多次刷新电商页面并观察监控和日志，查看是否所有用户ID为10的流量都进入了单元格H中的下单应用实例。
3. 下游无可用机器时兜底能力验证：关闭单元格H内的所有下单应用实例，多次刷新电商页面并观察监控和日志。
    i. 查看是否所有用户ID为10的流量都正常返回。
    ii. 查看是否流量都进入了单元格B或单元格I内的下单应用实例。

iii. 查看是否流量都回到了单元格H内的商品应用实例。



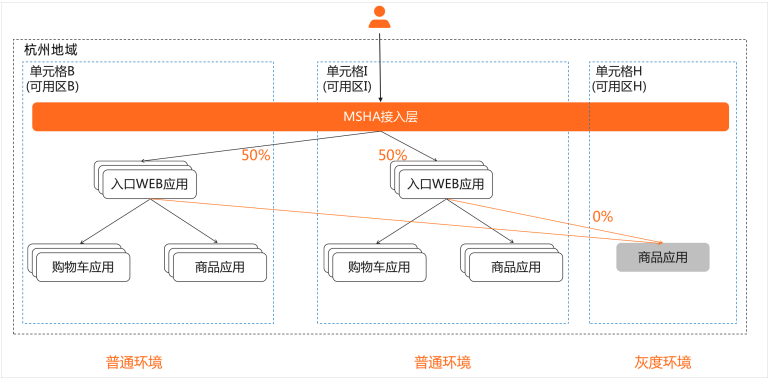
4. 流量封闭验证：关闭所有引流规则，多次刷新电商页面，观察监控是否有流量进入单元格H。

场景二：RPC流量灰度引流

为了满足业务需求，电商系统进行了一次功能优化。此次变更涉及商品应用，完成变更后，此应用需先在灰度环境进行灰度观察和验证。

接入MSHA的操作流程如下：

1. 扩容机器并创建一个新的单元格，作为灰度环境（如下图中单元格H）。
2. 将变更涉及的1个应用扩容出机器并划分归属到灰度环境的单元格H。
3. 配置SpringCloud引流规则。



完成操作后可从以下三个方面验证灰度能力：

1. RPC引流验证：配置SpringCloud引流规则，多次刷新电商页面并观察监控和日志，查看是否所有用户ID为20的流量都进入了单元格H。
2. 流量封闭验证：关闭所有引流规则，多次刷新电商页面，观察监控是否有流量进入单元格H。
3. 下游无可用机器时兜底能力验证：关闭单元格H内的所有下单应用实例，多次刷新电商页面并观察监控和日志。
   i. 查看是否所有用户ID为10的流量都正常返回。
   ii. 查看是否流量都进入了单元格B或单元格I内的下单应用实例，并且遵循了同城切0的逻辑。
   iii. 查看是否流量都回到了单元格H内的商品应用实例。

业务需求

从历史故障分析，大部分故障是因为没有经过灰度直接全量发布到生产环境所造成的。因此当前的需求是建设安全生产环境，通过安全生产环境的小流量灰度验证和观测，提前暴露问题和风险。具体需求内容包括：

- 采用跟生产环境不同的中间件（微服务注册中心、MQ实例等），搭建一套常态化运行的安全生产环境，并建立统一的灰度发布流程，经过安全生产环境的灰度发布和观测验证后，才允许发布到生产环境。

- 固定从生产环境引流1%的HTTP流量进入安全生产环境。
- RPC调用下游应用时，进入安全生产环境的流量需要在该环境内封闭。但如果下游应用在安全生产环境无可用机器时，则兜底调用到线上环境。
- 发布流程结束后，安全生产机器仍然保留，以便下游应用随时利用安全生产环境进行灰度验证。

基于MSHA搭建安全生产环境的方案

MSHA控制台提供一站式的流量管控能力。通过以下步骤的操作，能够让业务具备安全生产环境的灰度能力。

步骤一：搭建环境

1. 创建新的容器服务ACK集群，将需要灰度的应用创建出1～2个实例，作为安全生产环境。
2. 安全生产环境的应用使用独立的中间件，包括微服务注册中心、MQ实例等。
3. 基于MSHA单元格模型，创建一个新的单元格作为安全生产环境的逻辑区域。
4. 将上述应用实例划分归属到安全生产环境单元格。
5. 单元格配置为开启流量隔离，以保证默认情况下不会有任何流量进入。

> 说明
>
> - 单元格：是一个逻辑区域的概念。MSHA基于单元格的粒度进行流量控制，支持按可用区或指定JVM启动参数的方式将应用实例划分归属到对应的单元格。
> - 开启流量隔离：配置为开启流量隔离的单元格，流量比例固定为0%，在不配置引流规则的情况下不会有任何HTTP、RPC、MQ、定时任务调度流量进入。

步骤二：配置灰度引流规则

1. 创建新的SLB实例，用于对安全生产环境的入口应用实例进行负载均衡。
2. 基于MSHA的域名统一接入能力以及HTTP流量按单元格分流的能力，在MSHA控制台将安全生产环境单元格对应的回源地址配置为上述创建的SLB VIP。
3. 基于MSHA的HTTP引流能力，配置HTTP引流规则，固定引流1%到安全生产环境。

步骤三：配置服务同步

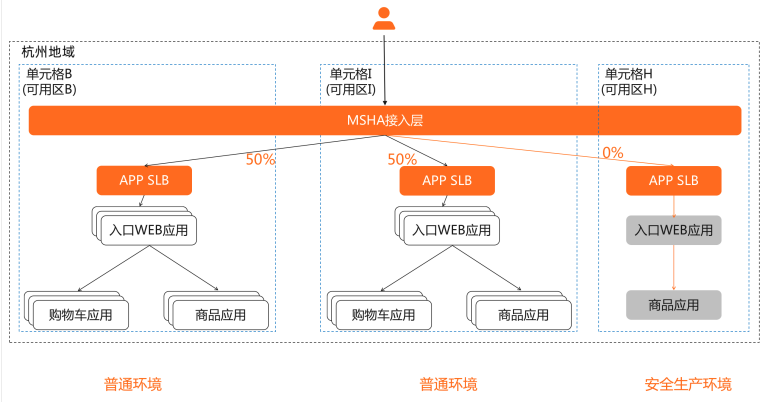
利用MSHA服务同步的能力，将生产环境的RPC服务同步到安全生产环境的Eureka注册中心，以便安全生产环境的流量在RPC调用下游发现无可用机器时，能够兜底调用回生产环境。

接入MSHA实践

为了满足业务需求，电商系统新增了一个功能，此次变更涉及入口Web应用和商品应用。为了满足安全生产“无灰度，不发布“的要求，需将这2个应用先发布到安全生产环境进行一段时间的灰度观察和验证。

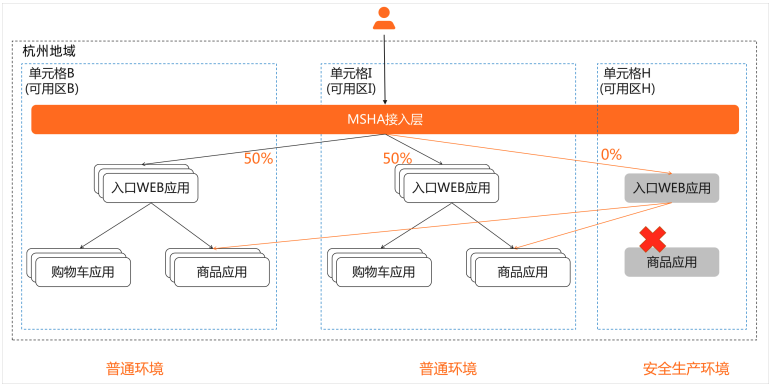
接入MSHA的操作流程如下：

1. 创建一个新的单元格，作为安全生产环境（如下图中单元格H）。
2. 创建新的容器服务ACK集群，将变更涉及的2个应用创建出1～2个实例，并划分归属到单元格H。
3. 创建VPC类型的SLB实例，配置监听HTTP 80端口，后端服务器组选择单元格H内的入口Web应用。
4. 配置域名、URI，并配置每个单元格对应的回源IP和Port信息。其中单元格H的回源IP和Port需要配置为上述创建的SLB VIP，Port为80。
5. 配置HTTP引流规则并开启流量染色，以便RPC调用下游应用时，流量能够根据染色标在安全生产环境内封闭。
6. 配置注册中心服务同步，将生产环境的RPC服务同步到安全生产环境的Eureka注册中心。



完成操作后可从以下四个方面验证灰度能力：

1. HTTP引流验证：配置HTTP引流规则，多次刷新电商页面并观察监控和日志，查看是否所有用户ID为10的流量都进入了单元格H。
2. 流量全链路染色和路由能力验证：配置HTTP引流规则和链路染色，多次刷新电商页面并观察监控和日志，查看是否所有用户ID为10的流量都进入了单元格H中的商品应用实例。
3. 下游无可用机器时兜底能力验证：关闭单元格H内的所有商品应用实例，多次刷新电商页面并观察监控和日志。
   i. 查看是否所有用户ID为10的流量都正常返回。
   ii. 查看是否流量都进入了单元格B或单元格I内的商品应用实例。



4. 流量封闭验证：关闭所有引流规则，多次刷新电商页面，观察监控是否有流量进入单元格H。

5.4.7. 混合云应用双活容灾最佳实践

越来越多的企业在数字化转型和上云进程中选择混合云的形态（云+自建IDC或云+其他厂商云）来进行容灾建设，一方面不会过度依赖单一云厂商，另一方面还能充分利用已有的线下IDC资源。MSHA云原生多活容灾解决方案，支持混合云多活容灾产品能力。本文会通过一个业务Demo案例，介绍混合云容灾建设的难点，以及如何基于MSHA来快速搭建应用双活架构并具备分钟级业务恢复能力。

背景信息

A企业是一个零售行业电商交易平台，业务系统部署在自建IDC机房，存在以下痛点：

- 业务仅在IDC单机房部署，缺少容灾能力。
- IDC容量不足，物理机器升级替换周期长，不足以支撑业务的快速发展。

业务在快速发展过程中，多次遇到容量不足以及故障问题引起了公司高层的重视，因此决定进行容灾能力建设。由于自建IDC是公司已有资产且稳定使用多年，同时不希望过度依赖于云，因此期望建立IDC+云的混合云形态容灾架构。

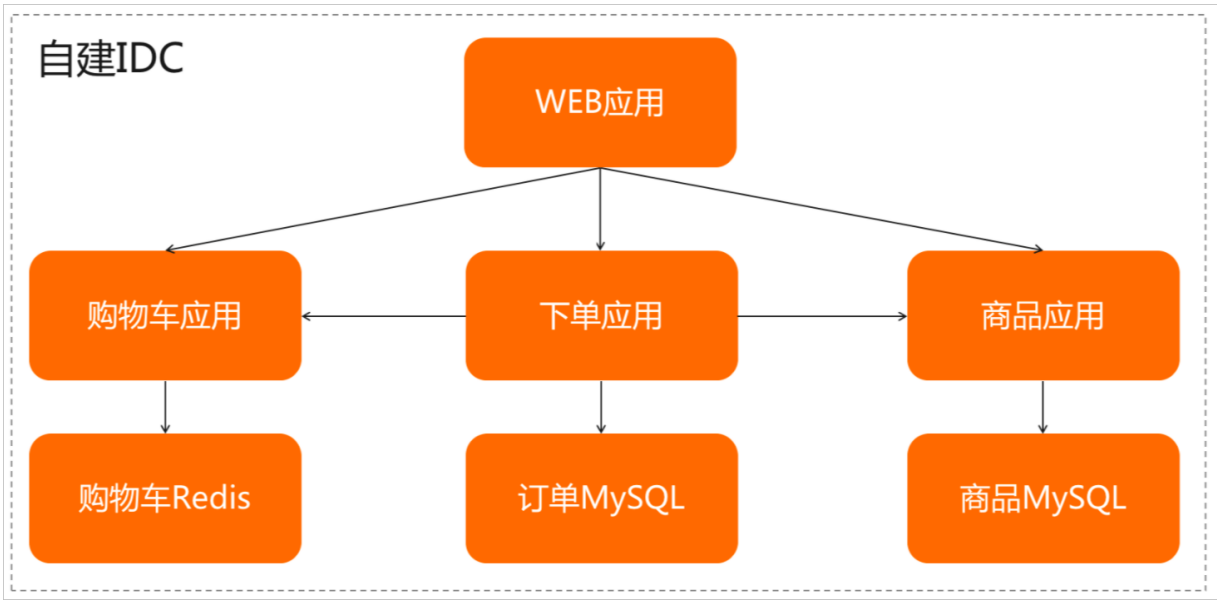
A企业当前的应用部署架构

电商交易平台包含的应用：

- frontend：Web应用，负责和用户交互。
- cartservice：购物车应用，提供购物车添加、存储和查询服务。
- productservice：商品应用，提供商品、库存服务。

技术栈：

- SpringBoot。
- RPC框架：SpringCloud、Dubbo，注册中心使用自建的Nacos、ZooKeeper。
- 数据库：Redis和MySQL。



混合云容灾目标

A企业业务容灾的需求如下：

需求	说明
云上云下互容灾，切换RTO为分钟级。	期望云上云下相互容灾，继续发挥IDC的价值，且不100%依赖于云。面对IDC或云故障场景，关键时刻要敢切换、能切换，且切换RTO要求小于10分钟。
无数据一致性风险。	云上云下的两个数据中心数据强一致，日常态和容灾切换过程中都要避免存在脏写等数据一致性风险。
一站式管控。	业务容灾涉及的技术栈框架和云产品，需要统一管控、统一运维、统一切换，操作收敛在一站式管控平台，方便故障场景快速白屏化操作，自动化执行。
实施周期短，改造成本低。	业务存在多个产品线，依赖关系复杂、调用链路长，且处于高速发展频繁迭代时期，期望容灾建设不会给业务研发团队带来改造负担。

建设难点

建立IDC+云的混合云形态容灾架构难点如下：

难点	说明

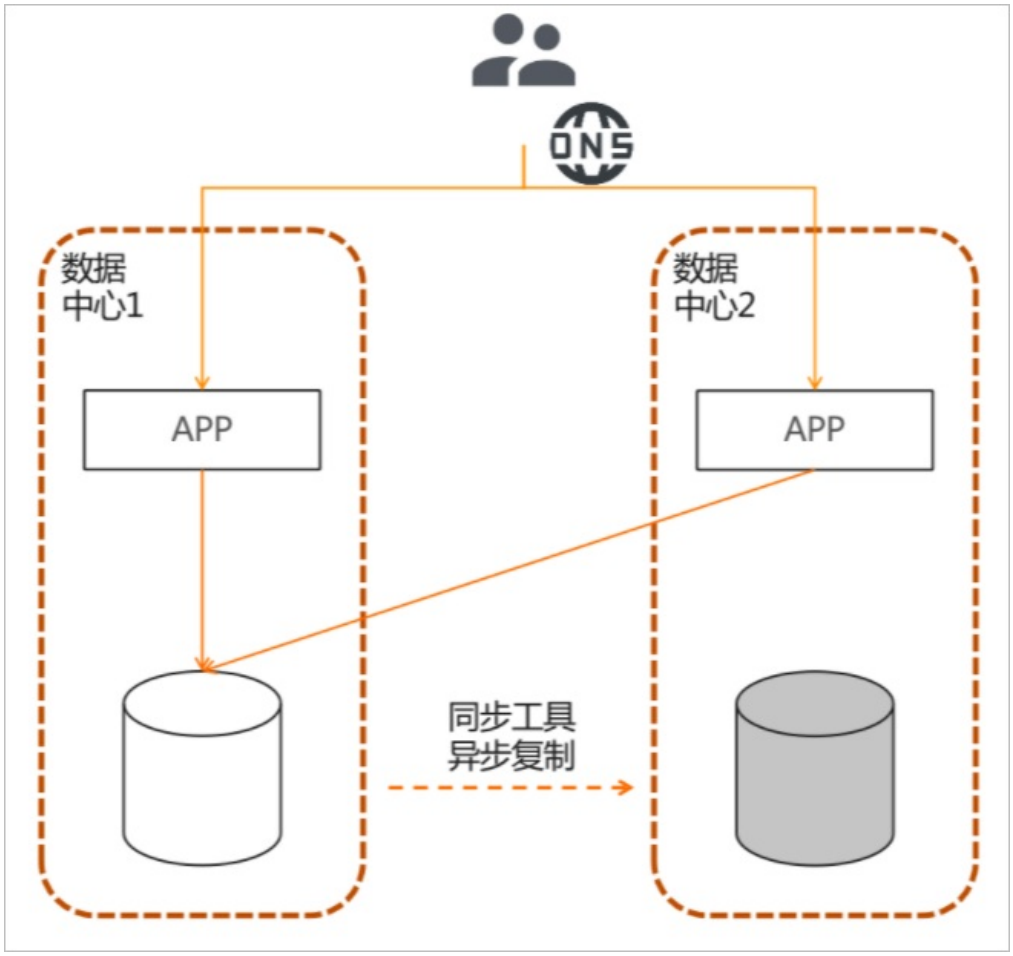
难点	说明
流量管理难度高	• 若采用DNS将流量按权重解析到云上和云下，存在修改DNS解析生效时间长的问题（通常为十分钟或小时级，具体操作，请参见解析生效时间FAQ。 • 业务应用所依赖的Redis和MySQL，IDC内采用开源自建而云上直接使用云产品，要实现开源自建+云产品的容灾切换能力难。
容灾切换数据质量保障难	容灾切换过程中，可能因数据同步延迟导致读到旧数据，以及切换规则推送到分布式应用节点时间不一致等原因可能造成云上云下数据库同时读写而出现脏写的问题，整个切换过程数据质量保障是关键点及难点。
无业务代码侵入难	要实现Redis、MySQL容灾切换能力，通常需要业务应用配合改造，对业务代码侵入大。

解决方案

结合业务容灾需求和混合云IDC+云形态的特点，采用应用双活架构能够较好的满足业务容灾诉求。

应用双活架构

架构简图：



限制：业务需要能接受两地距离带来的网络延迟。

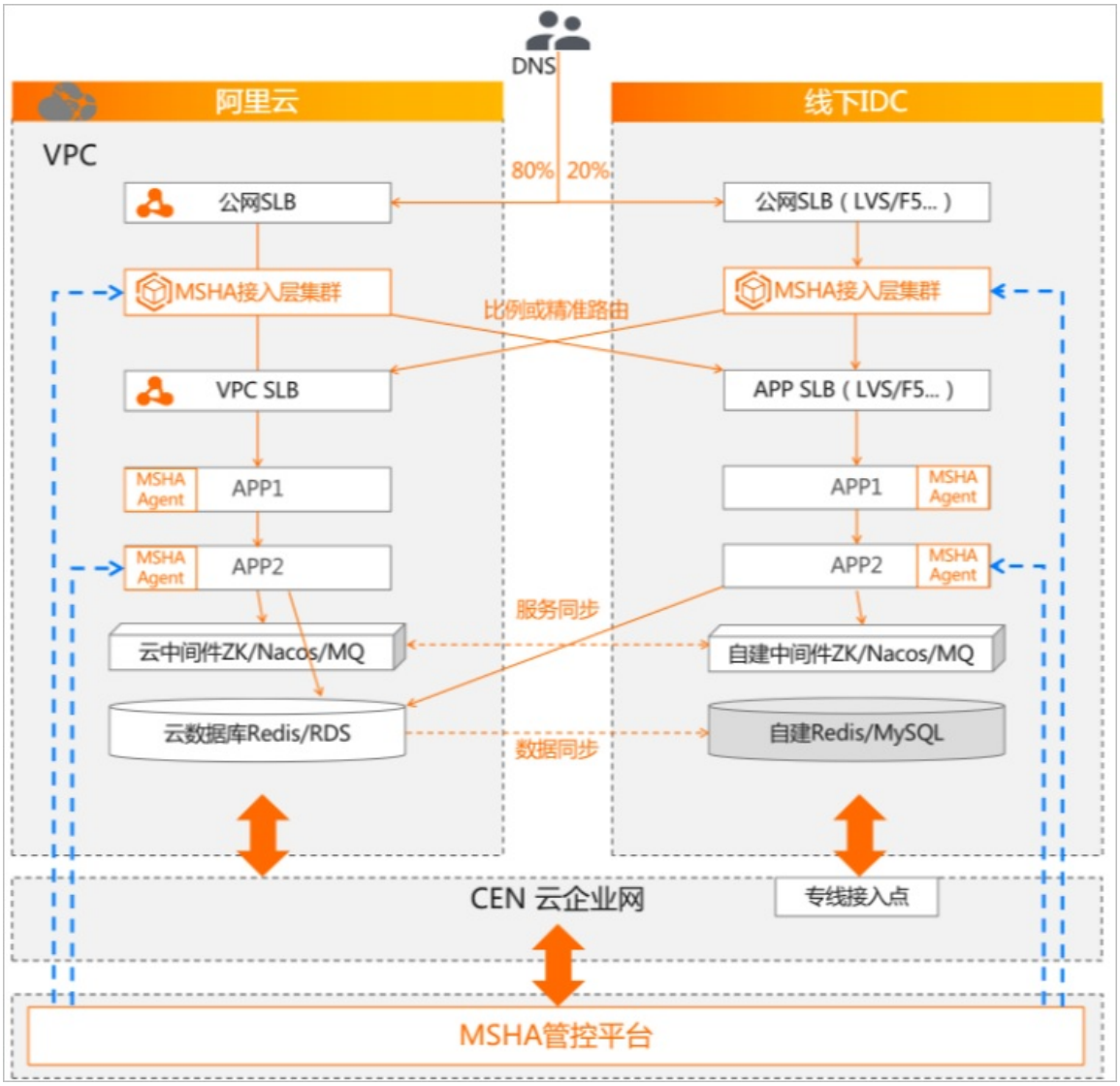
架构规范：

- 选择离IDC物理距离≤200 km的云上地域（Region），网络延迟较低（约5~7 ms）。

- 应用、中间件云上云下冗余对称部署，同时对外提供服务（应用双活）。
- 数据库异地主备，异步复制备份。应用读写同一数据中心的数据库，避免考虑一致性问题。

解决方案

架构简图：



解决方案：

- 应用流量双活：
  - 业务应用云上云下对称部署，并基于MSHA接入层集群，来承接入口HTTP/HTTPS流量，按照比例或精准路由规则云上云下分流。
  - 多活控制台提供MSFE集群界面白屏化的部署、扩缩容、监控等常规运维能力，以及应对故障场景的分钟级切流能力。
- 服务互通和同单元优先调用：
  - 业务应用需要按业务产品线分批上云，过程中存在下游应用仅IDC部署的情况。利用MSHA注册中心同步功能，可实现云上云下服务互通，助力业务上云。
  - 同时基于MSHA-Agent的切面能力，在Dubbo/SpringCloud服务调用时，Consumer优先调用同单元内的Provider，从而避免跨机房调用带来的网络延迟，减小业务请求RT。
- 数据同步及数据库连接切换：

- 数据库异地主备部署，云上云下应用日常态均读写云上Redis和RDS数据库，无需考虑数据一致性问题。MSHA控制台通过集成DTS同步组件，支持云上云下的数据同步（异步复制）。
- 同时基于MSHA-Agent切面能力，具备应用数据库访问连接的切换能力，云上Redis或RDS故障则可将读写访问连接切换到IDC内的Redis或MySQL，反之亦然。切换过程中还具备禁写保护能力，避免产生读到旧数据以及脏写等数据质量问题。

- 一站式管控及无业务代码侵入
  - MSHA控制台，支持HTTP、数据库访问流量的统一管控、统一切换，操作收敛在一站式管控平台，方便故障场景快速白屏化操作，自动化执行。
  - 同时针对业务应用MSHA提供了Agent接入方式，无需业务代码改造即可获得相关容灾切换能力。

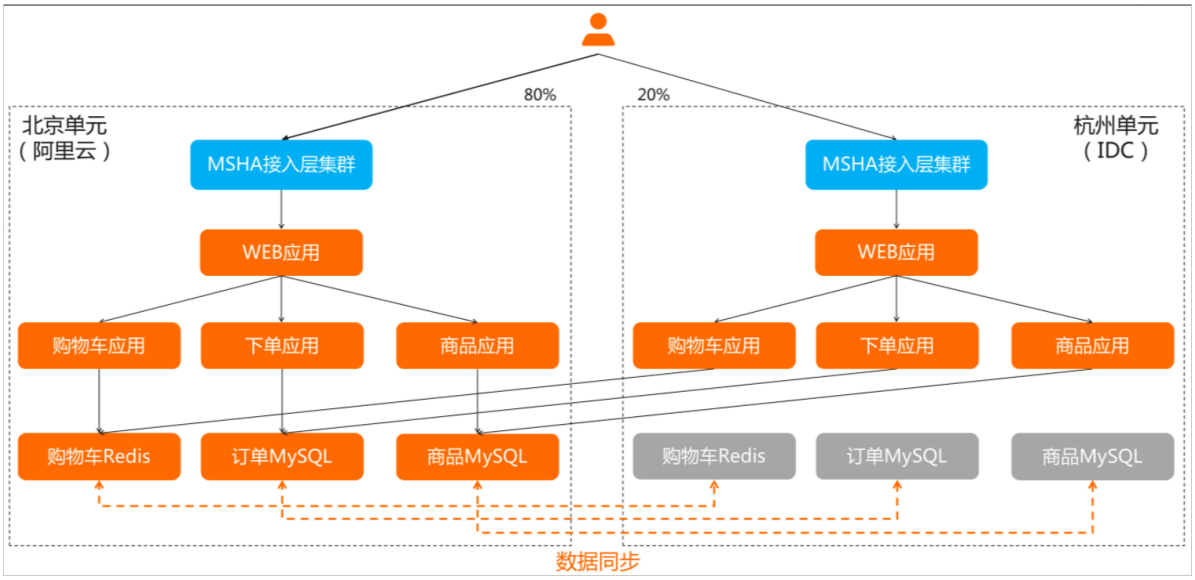
改造内容

A企业建立IDC+云的混合云形态容灾架构，所需改造的内容如下：

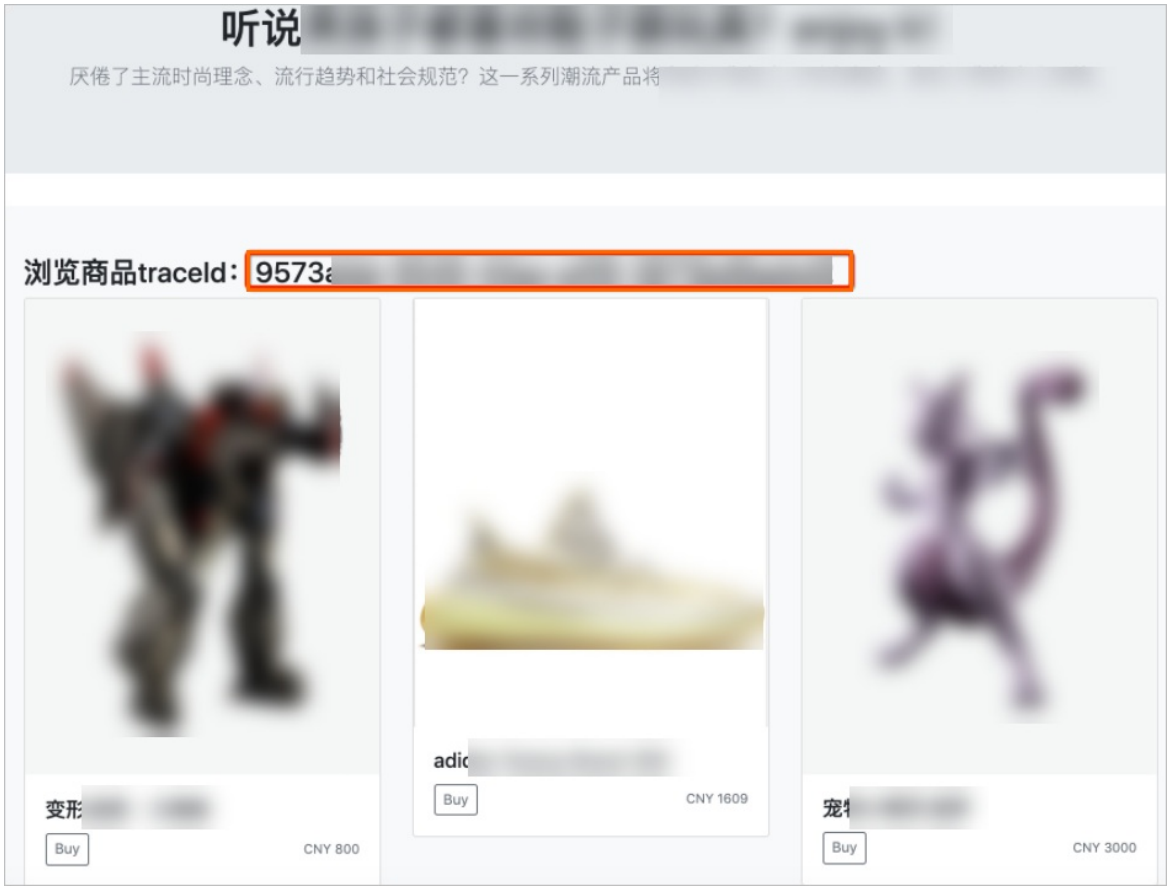
改造点	说明
应用上云	选择跟自建IDC较近的阿里云地域，云上完全冗余的部署一套应用、中间件和数据库，以便搭建云上云下双活容灾架构。在这个Demo案例中，选择杭州地域（Region）作为容灾单元。
网络打通	接入CEN云企业网，实现云上云下网络互通。具体操作，请参见多接入方式构建企业级混合云。
接入集群部署和配置	• 云上云下部署MSHA接入层集群（MSFE），上挂SLB用于公网接入以及MSFE集群的负载均衡。具体操作，请参见管理MSFE接入层集群。 • 录入域名、URI和后端应用地址，从而具备云上云下分流和分钟级切流能力。具体操作，请参见配置MSFE。
应用	• 云上分批部署业务应用。 • JAVA应用安装MSHA-Agent，从而具备微服务同单元优先调用以及数据库访问连接切换能力。具体操作，请参见为Java应用手动安装探针。
中间件和数据库	• 云上部署MSE托管ZooKeeper/Nacos注册中心、云数据库Redis和RDS，建议使用跨可用区部署高可用版本，具备同城双活容灾能力。 • 若存在某应用仅IDC部署的情况，需要配置注册中心的服务同步。具体操作，请参见注册中心同步集群。 • 配置云数据库Redis或RDS和自建Redis或MySQL的数据同步。具体操作，请参见配置数据层。

改造后的应用部署架构：

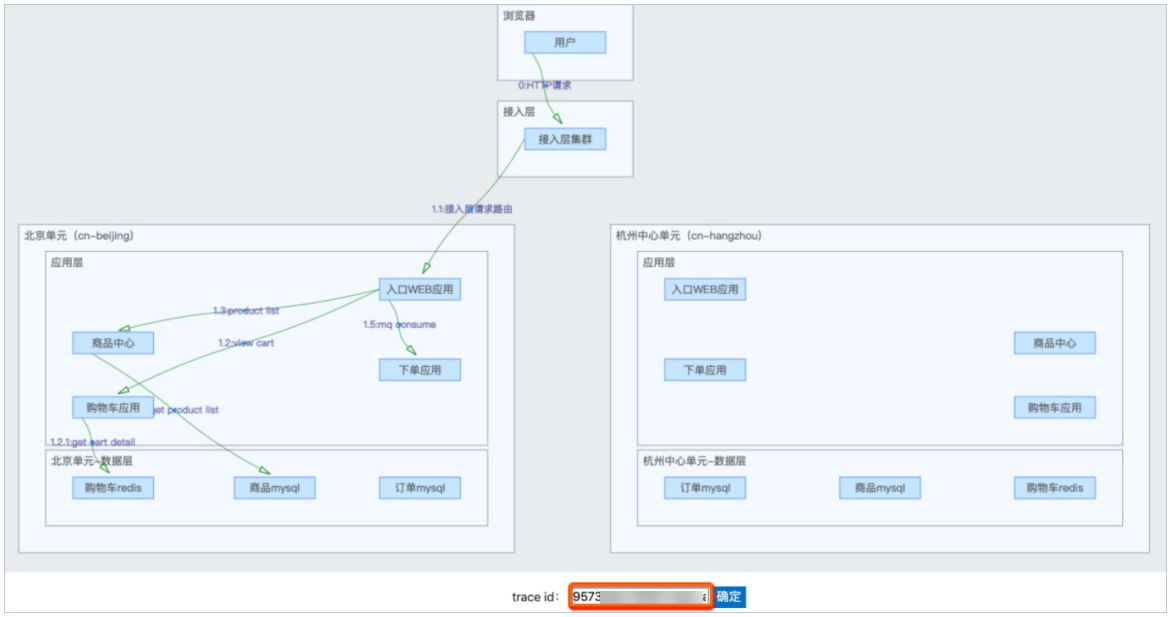
- 日常场景：IDC+云上同时承担业务流量（即应用双活）。



- 访问电商Demo首页，查看实际流量调用链：概率性的访问到北京或杭州单元，均读写北京单元内的数据库。
  - 电商Demo首页图：



- 均读写北京单元内的数据库架构简图：



容灾能力

- RPO：≤1分钟（依赖于DTS同步性能）。
- RTO：≤1分钟（依赖于DTS同步延迟，MSHA组件实现秒级切换。整体RTO≤1分钟）。

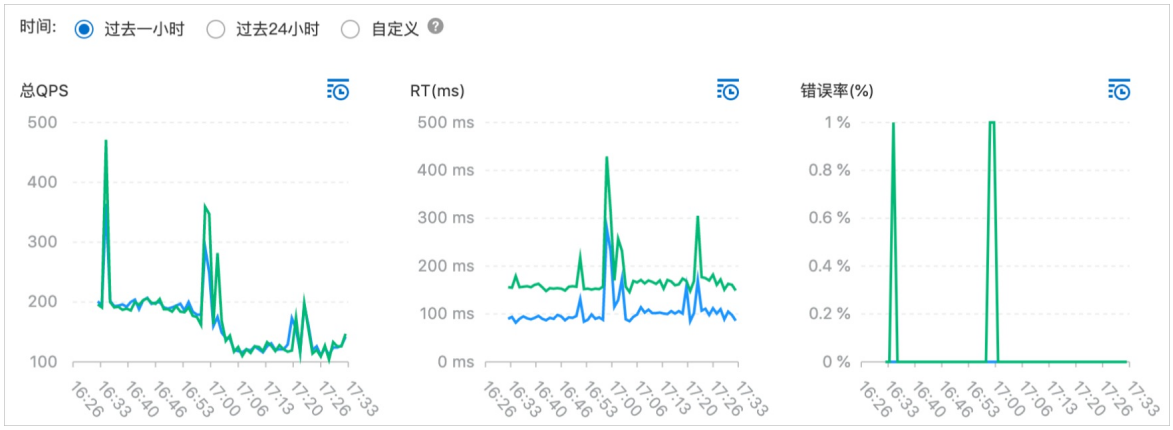
容灾能力验证

基于MSHA完成应用双活架构建设后，还需验证业务容灾能力是否符合预期。接下来将制造真实的故障，来验证容灾恢复能力。

步骤一：演练准备

1. 登录AHAS控制台。
2. 在控制台左侧导航栏中单击**多活容灾**。
3. 在左侧导航栏单击**监控大盘**，并在顶部选择目标命名空间，查看各项监控指标。

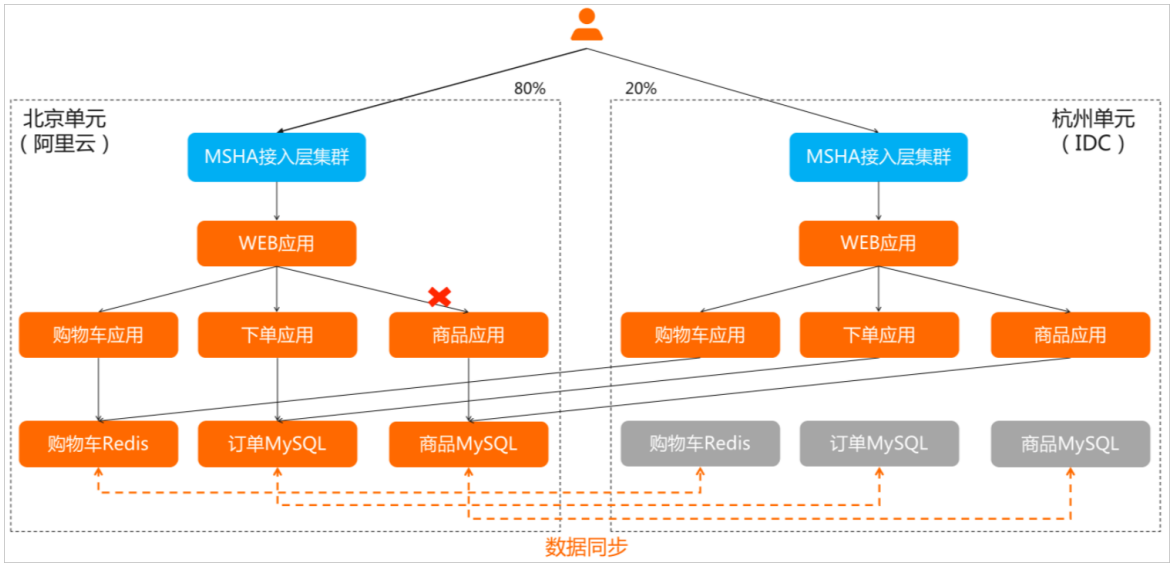
> **说明** 演练前，基于MSHA流量监控或其他监控产品，确定业务稳态的监控指标（如日常情况RT≤200ms，错误率<1%），以便在故障发生时判断故障影响面以及在故障恢复后判断业务的实际恢复情况。

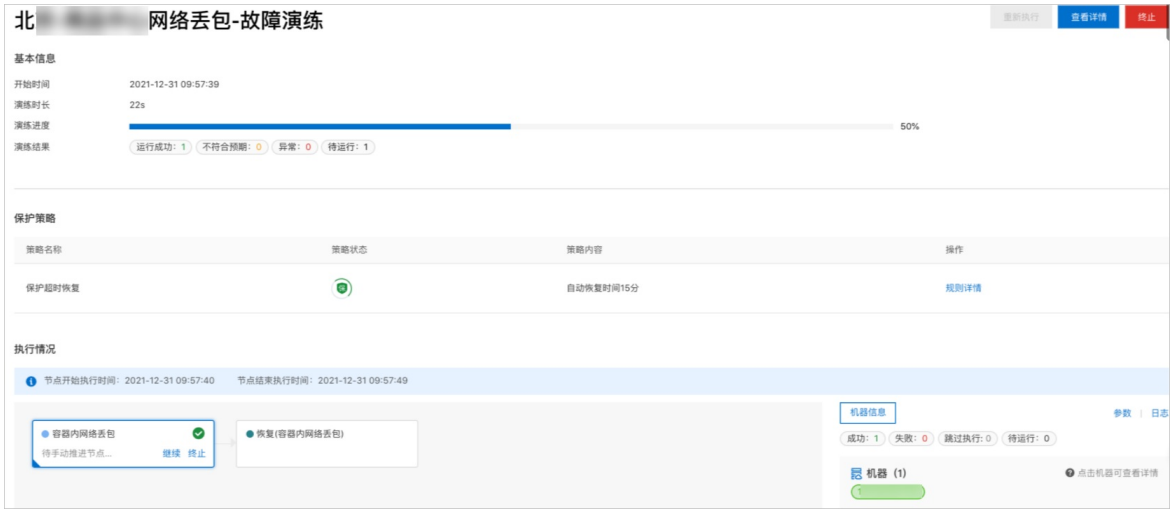


步骤二：应用故障注入

这里使用阿里云故障演练产品，对阿里云-北京地域的商品应用注入故障。

1. 登录AHAS控制台。
2. 在左侧导航栏选择**故障演练 > 我的空间**，并在顶部选择地域。
3. 在**我的空间**页面搜索配置好的演练（50%概率网络丢包），然后在该演练名称对应的**操作**列单击**演练**。在弹出的对话框中单击**确认**。



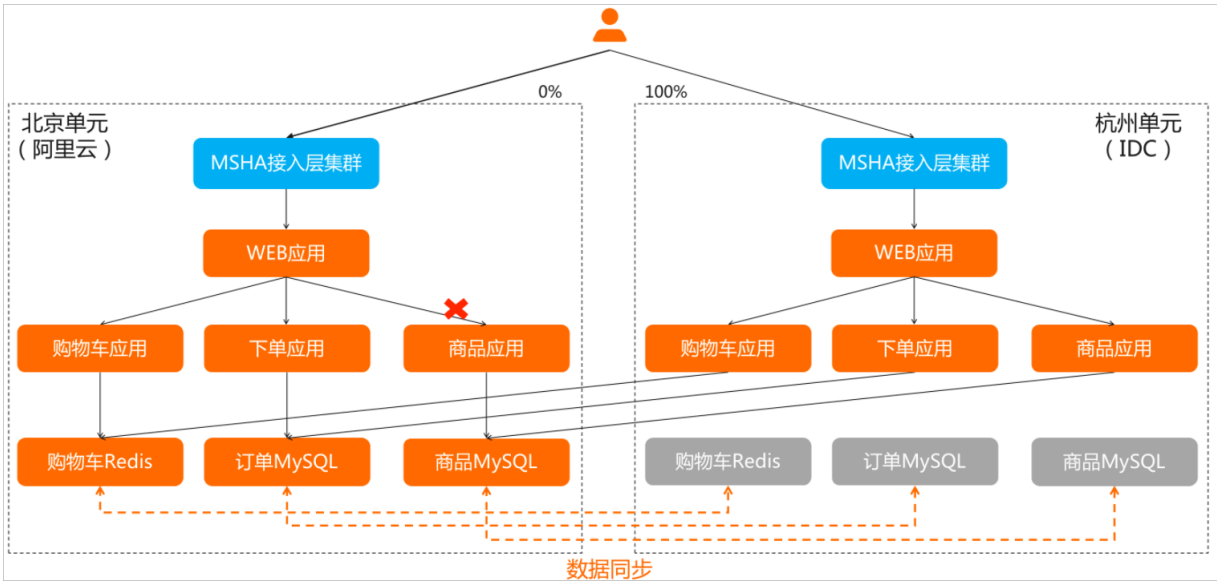


故障注入成功后，打开电商首页或进行下单，有概率出现访问异常，则符合预期。

步骤三：切流恢复

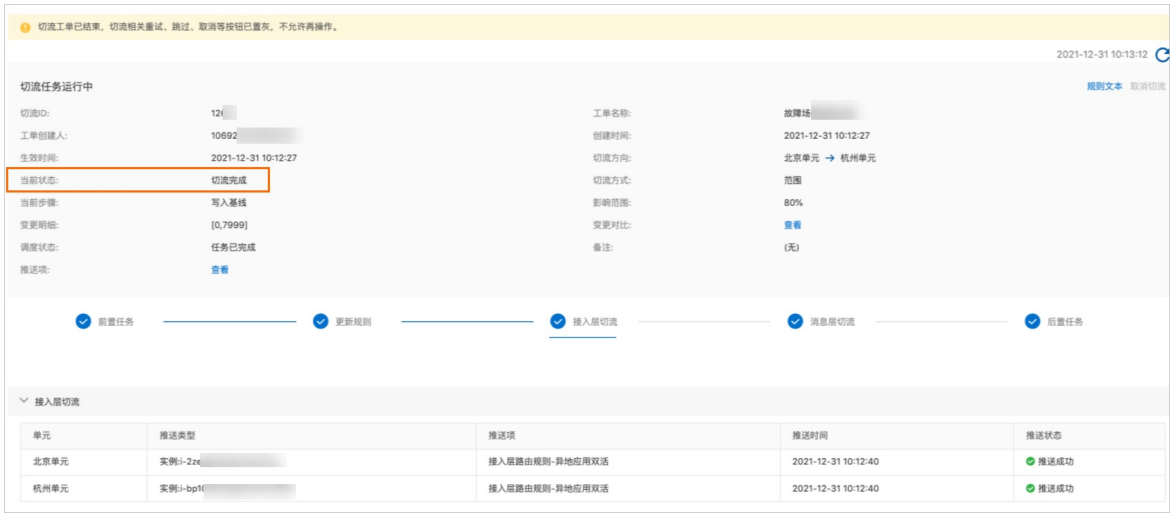
在北京单元的商品应用故障的情况下，可以通过MSHA切流功能，将云上入口流量切0，快速恢复业务。

预期效果：100%流量切换到杭州单元后，业务完全恢复，不受北京单元的故障影响。



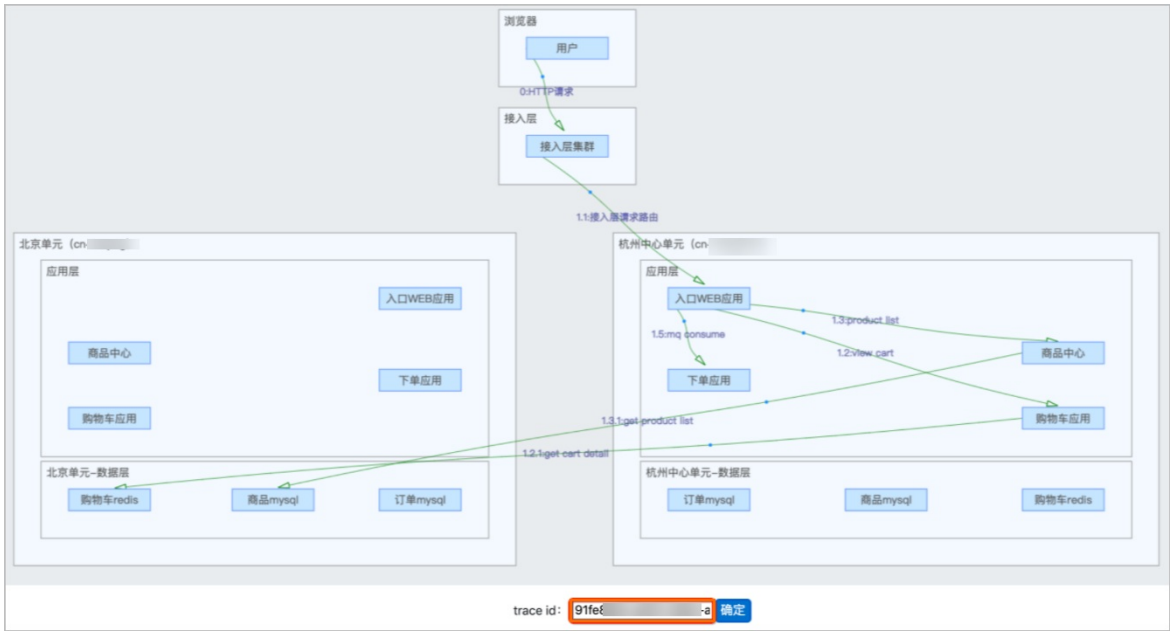
1. 登录AHAS控制台。
2. 在控制台左侧导航栏中单击**多活容灾**。
3. 在左侧导航栏选择**切流 > 异地应用双活切流**，并在顶部选择地域。
4. 在**异地应用双活切流**页面单击**切流**，然后在第①步**规则调整**区域的北京单元右侧单击**一键切流**，并单击**生成预览**，将北京单元的应用流量一键切零。

5. 单击第②步的**执行预检查**，然后单击第③步**切流检查**区域的**确认**，开始切流。
若**切流任务运行中**页面的**当前状态**显示**切流完成**，表示切流已成功。



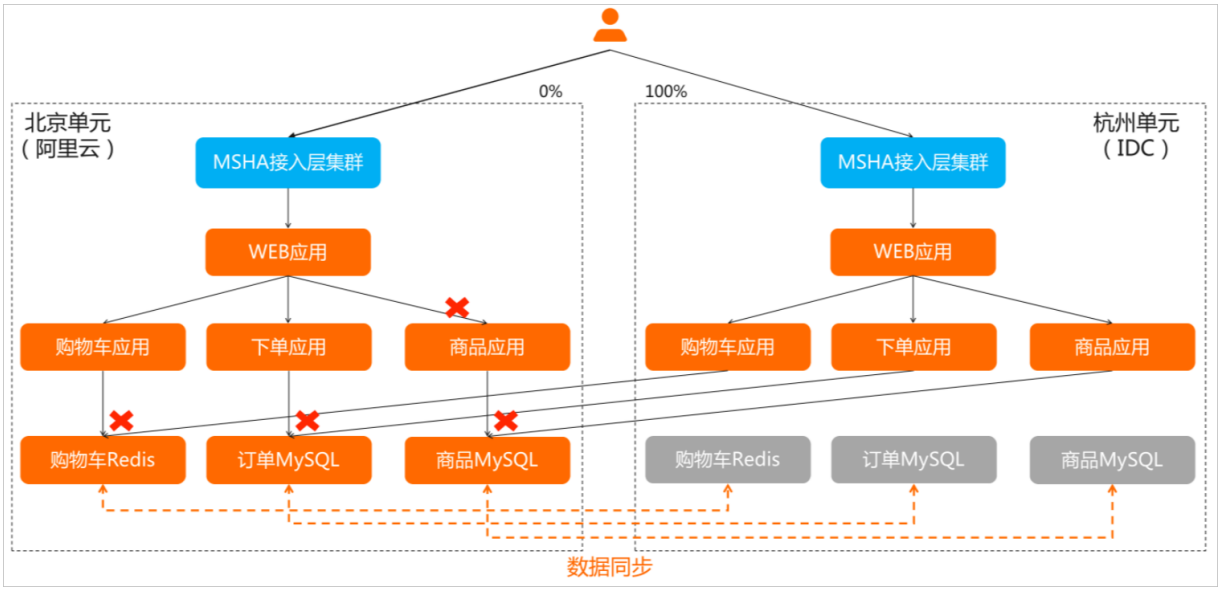
6. 刷新电商Demo首页，多次访问均能正常展示，则符合预期。

查看实际流量调用链：流量始终访问到杭州单元，读写北京单元内的数据库。



步骤四：数据库故障注入

从上面调用链可以看出，杭州单元内的应用仍然访问的是北京单元的Redis、MySQL数据库。以同样的方式执行步骤二对北京单元的Redis、MySQL数据库注入故障，制造数据库故障场景。

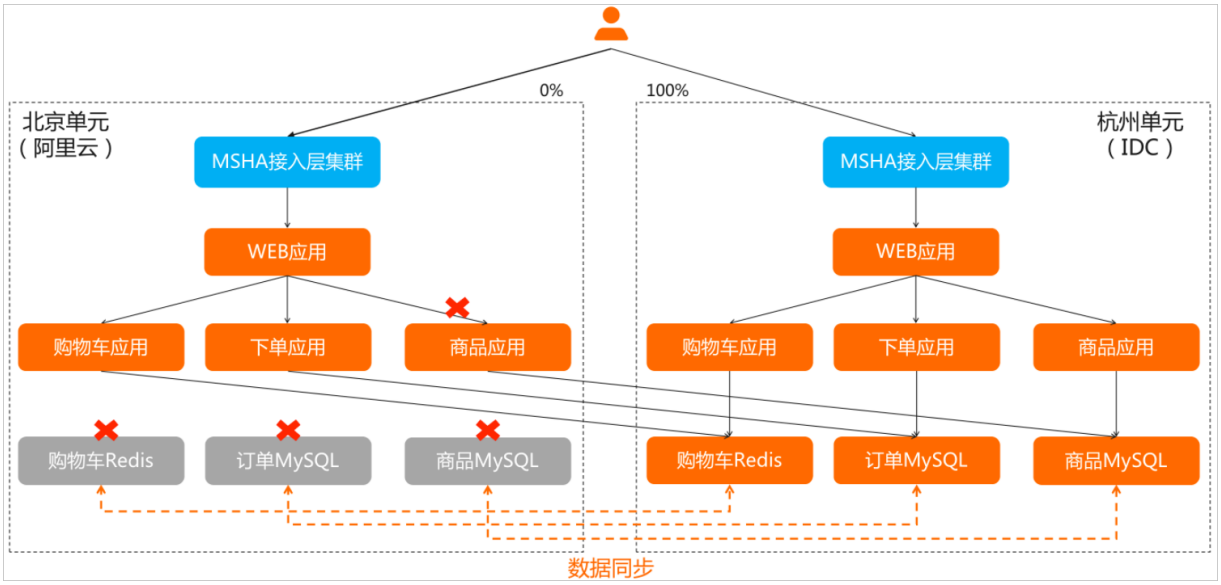


故障注入成功后，打开电商首页或进行下单始终访问异常，则符合预期。

步骤五：切换数据库进行恢复

在北京单元的数据库故障的情况下，可以通过MSHA数据库切换功能，将应用访问的Redis或MySQL的连接切换至杭州单元的数据库（切换过程中会等待数据同步追平，期间会短暂禁写）。

预期效果：应用连接的数据库切换到杭州后，业务完全恢复，不受北京单元的故障影响。



1. 登录AHAS控制台。
2. 在控制台左侧导航栏中单击**多活容灾**。
3. 在左侧导航栏选择**异地应用双活 > 数据层配置**。
4. 在数据保护规则列表中，通过搜索查询商品、订单、购物车数据库，依次单击**主备切换**。

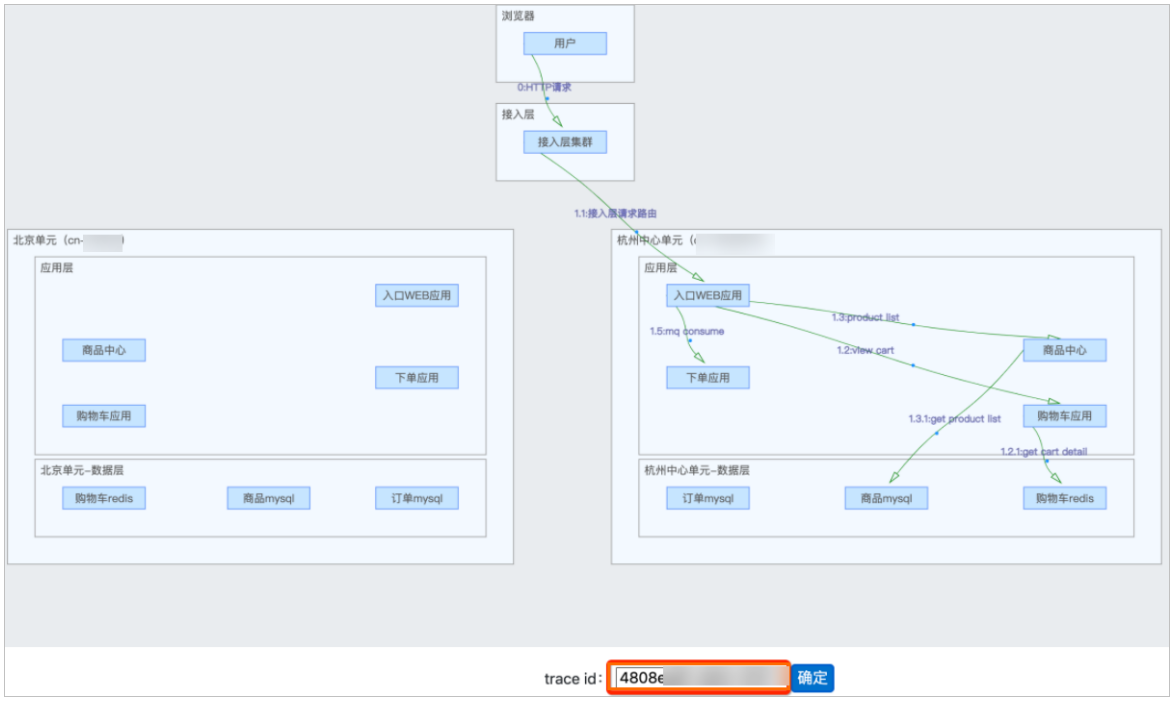
   商品数据库：

   购物车数据库：

5. 单击**主备切换**后，进入**预检查结果**页面，确认各检查项状态正常后，然后单击**确认执行**，在主备切换详情页面自动执行切换流程。

在主备切换详情页面，可以查看切换进度和切换结果，当**任务进度**为100%，表示切换完成。主备切换完成后的结果如下图所示。

商品、订单、购物车数据库都完成主备切换后，多次访问电商Demo首页或进行下单，发现均已正常，主备切换后业务功能完全恢复，则符合预期。



通过MSHA多活容灾助力企业进行混合云应用双活容灾建设的实践案例，给出了容灾架构建设实践方法，同时利用Chaos故障演练产品注入真实故障，来验证故障场景业务容灾能力是否符合预期。若您在使用过程中有任何疑问，欢迎您搜索钉钉群号“31623894”加入“多活容灾（MSHA）交流群”获取帮助。

5.4.8. 使用云监控对MSFE进行监控和报警实践

MSFE是多活流量统一入口，采用多台ECS集群化部署。您可以使用云监控对每台ECS开启主机监控和报警，以便出现主机性能瓶颈或者系统故障问题时能够快速发现和处理，进而规避风险。本文介绍如何使用云监控对MSFE进行监控和报警。

背景信息

云监控针对主机监控和报警，以及应用分组、可用性监控、Dashboard等基础功能均不收费。想要了解更多云监控信息，请参见什么是云监控。

开启主机监控

1. 登录云监控控制台。
2. 在左侧导航栏，单击**主机监控**。
3. 在**实例列表**页面，通过在搜索框中输入主机名称或者实例ID等搜索需要开启监控的ECS，然后在**插件状态（全部）**列单击**点击安装**来安装云监控插件，对每台ECS开启主机监控和报警。

4. 单击**操作**列的**监控图表**，可查看ECS实例的CPU、内存、负载、网络、连接、磁盘、IO、进程等监控指标详情。具体监控项详情请参见监控项说明。

配置报警规则

ECS类型的指标是从物理机层面采集的，数据准确性低于Agent从VM内部采集的数据，因此推荐您查看Agent采集的指标配置报警规则。推荐配置报警规则的监控项名称如下，具体详情请参见操作系统监控项。

监控内容	推荐配置报警规则的监控项名称	说明
CPU使用率	（Agent）cpu.total	一台ECS只运行一个Tengine进程服务，Tengine对资源的消耗主要体现在CPU使用率上，通常情况下，若CPU使用率达到60%，就需要考虑扩容。
Load	（Agent）load.5m.percore	CPU平均每核过去5分钟的系统平均负载。
内存使用率	（Agent）memory.used.utilization	-
磁盘使用率	（Agent）disk.usage.utilization_device	Tengine进程会打印tengine-access_log和tengine-error_log日志。 日志采用滚动覆盖的方式，滚动覆盖保留的文件数为7，全部日志文件最大会占用21 G磁盘空间。建议添加磁盘使用率报警规则，避免出现磁盘满的问题。

5.4.9. 使用SLS对MSFE日志进行收集、监控和报警实践

本文介绍使用SLS对MSFE日志进行收集、监控和报警。

背景信息

MSFE是多活流量统一入口，采用多台ECS集群化部署，一台ECS部署一个Tengine进程。您可以使用日志服务（SLS），对每台ECS上的Tengine日志进行统一收集和存放，能够有效追踪因Tengine日志滚动覆盖导致的问题。同时便于对日志搜索查询，以及基于日志进行问题排查和数据分析。同时，您还可以通过SLS配置Error日志报警规则，以便在Tengine自身或后端服务器出现问题能够及时发现并处理，从而规避风险。

接入层日志

Tengine进程会打印tengine-access_log和tengine-error_log日志。日志采用滚动覆盖的方式，滚动覆盖保留的文件数为7，全部日志文件最大会占用21 G磁盘空间。

- 日志路径：/home/admin/tengine/logs
- 日志文件名：tengine-access_log和tengine-error_log

```
-rw-r--r-- 1 root admin    1M Oct 12 21:17 error.log
-rw-rw-r-- 1 root admin  505M Nov 24 15:25 tengine-access_log
-rw-rw-r-- 1 root admin 2049M Nov 24 15:13 tengine-access_log.1
-rw-rw-r-- 1 root admin 2049M Nov 24 14:29 tengine-access_log.2
-rw-rw-r-- 1 root admin 2049M Nov 24 13:47 tengine-access_log.3
-rw-rw-r-- 1 root admin 2049M Nov 24 13:01 tengine-access_log.4
-rw-rw-r-- 1 root admin 2049M Nov 24 12:08 tengine-access_log.5
-rw-rw-r-- 1 root admin 2049M Nov 24 11:21 tengine-access_log.6
-rw-rw-r-- 1 root admin 2049M Nov 24 10:34 tengine-access_log.7
-rw-rw-r-- 1 root admin    1M Nov 24 15:13 tengine-error_log
-rw-rw-r-- 1 root admin    1M Nov 23 23:04 tengine-error_log.1
-rw-rw-r-- 1 root admin    1M Nov 22 23:04 tengine-error_log.2
-rw-rw-r-- 1 root admin    1M Nov 21 23:04 tengine-error_log.3
-rw-rw-r-- 1 root admin    1M Nov 20 23:07 tengine-error_log.4
-rw-rw-r-- 1 root admin    1M Nov 19 23:15 tengine-error_log.5
-rw-rw-r-- 1 root admin    1M Nov 18 23:05 tengine-error_log.6
-rw-rw-r-- 1 root admin    1M Nov 17 23:56 tengine-error_log.7
-rw-r--r-- 1 root admin    1M Aug 19 21:15 tengine-yun.pid
```

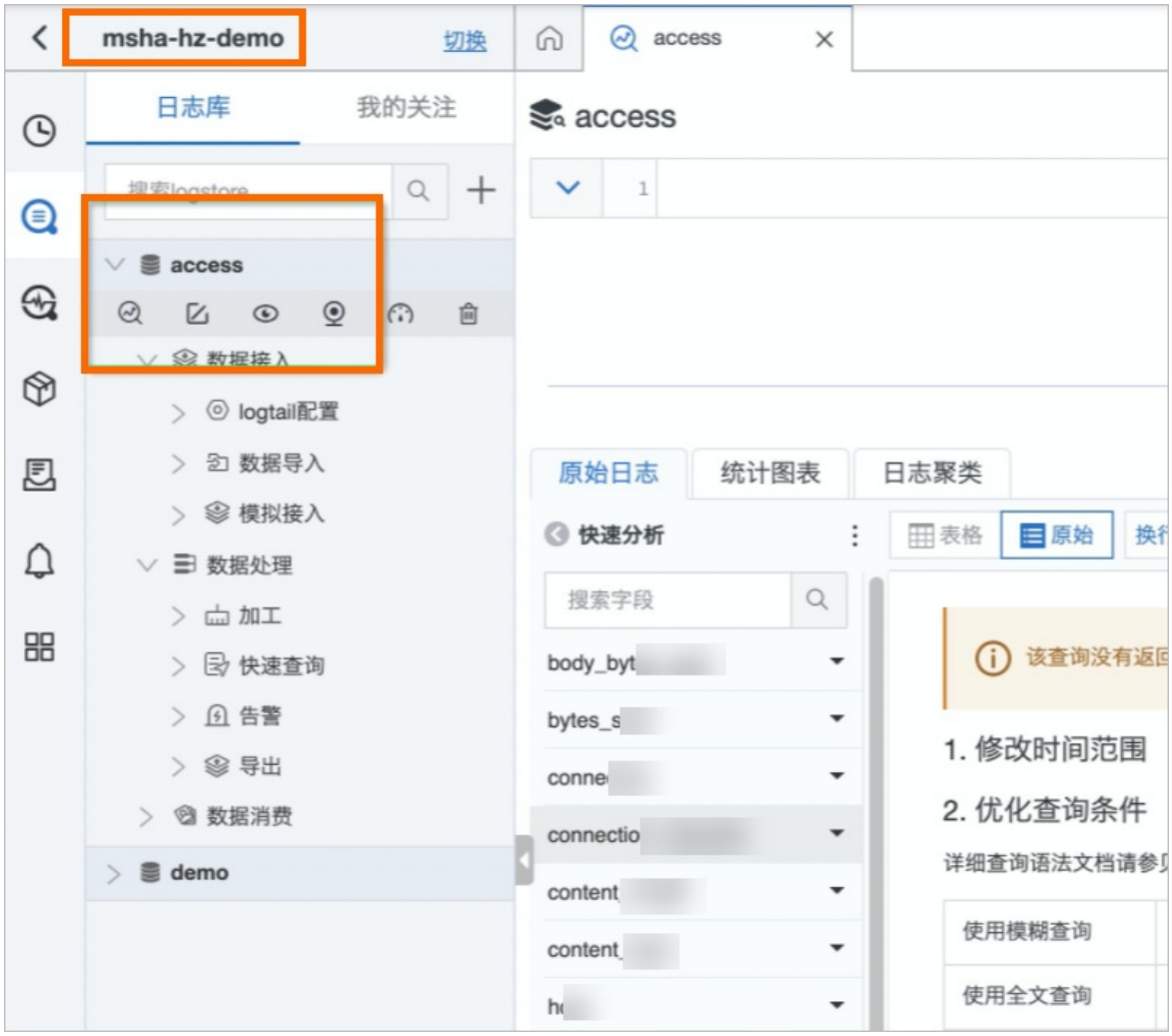
步骤一：接入前准备

1. 登录日志服务控制台。
2. 创建项目（Project）。
   i. 在Project列表区域，单击**创建Project**。
   ii. 在**创建Project**面板中，按照如下说明配置参数，其他参数均可保持默认配置。

参数	描述
Project名称	Project的名称，全局唯一。创建Project成功后，无法更改其名称。
所属地域	Project的数据中心。建议选择与ECS相同的地域，即可使用阿里云内网采集日志，加快采集速度。 创建Project后，无法修改其所属地域，且日志服务不支持跨地域迁移Project。

   iii. 单击**确定**。
3. 创建日志库（Logstore）。

   创建Project完成后，系统会提示您创建一个Logstore。
   在**创建Logstore**面板中，按照如下说明配置参数，其他参数均可保持默认配置。这里创建的Project名称为msha-hz-demo，创建的Logstore名称为access。



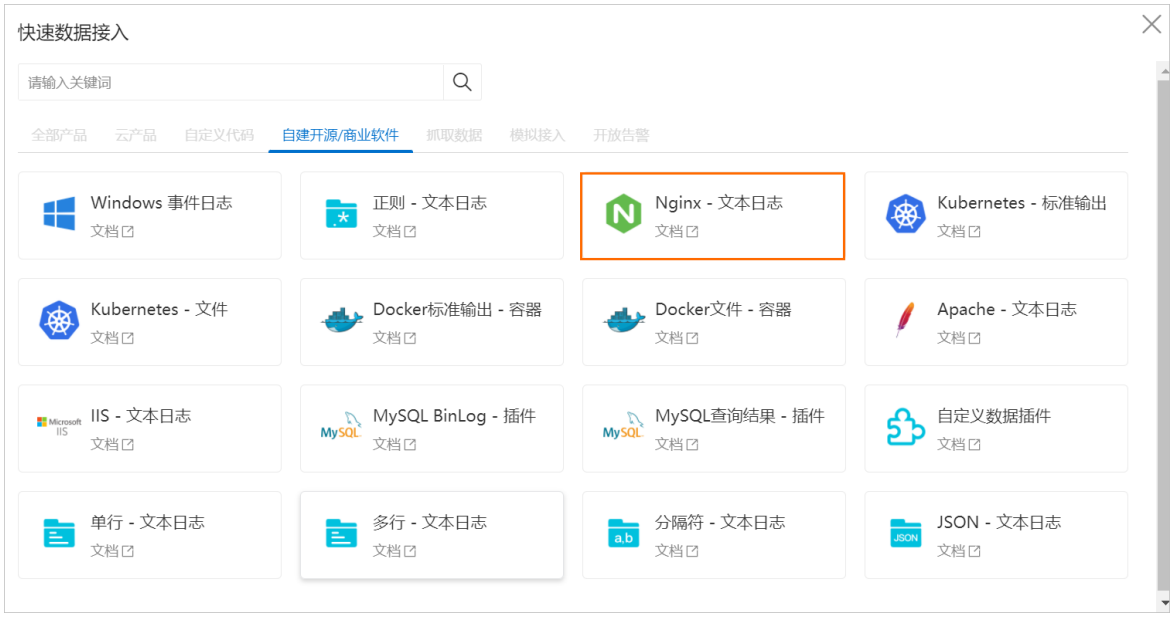
参数	描述
Logstore名称	Logstore的名称，在其所属Project内必须唯一。创建Logstore成功后，无法更改其名称。
Shard数目	日志服务使用Shard读写数据。 一个Shard提供的写入能力为5 MB/s、500次/s，读取能力为10 MB/s、100次/s。如果一个Shard就能满足您的业务需求，您可配置**Shard数目**为1。
自动分裂Shard	开启自动分裂功能后，如果您写入的数据量超过已有Shard服务能力，日志服务会自动根据数据量增加Shard数量。 如果您确保配置的Shard数量已满足业务需求，可关闭**自动分裂Shard**开关。

步骤二：配置access_log日志收集

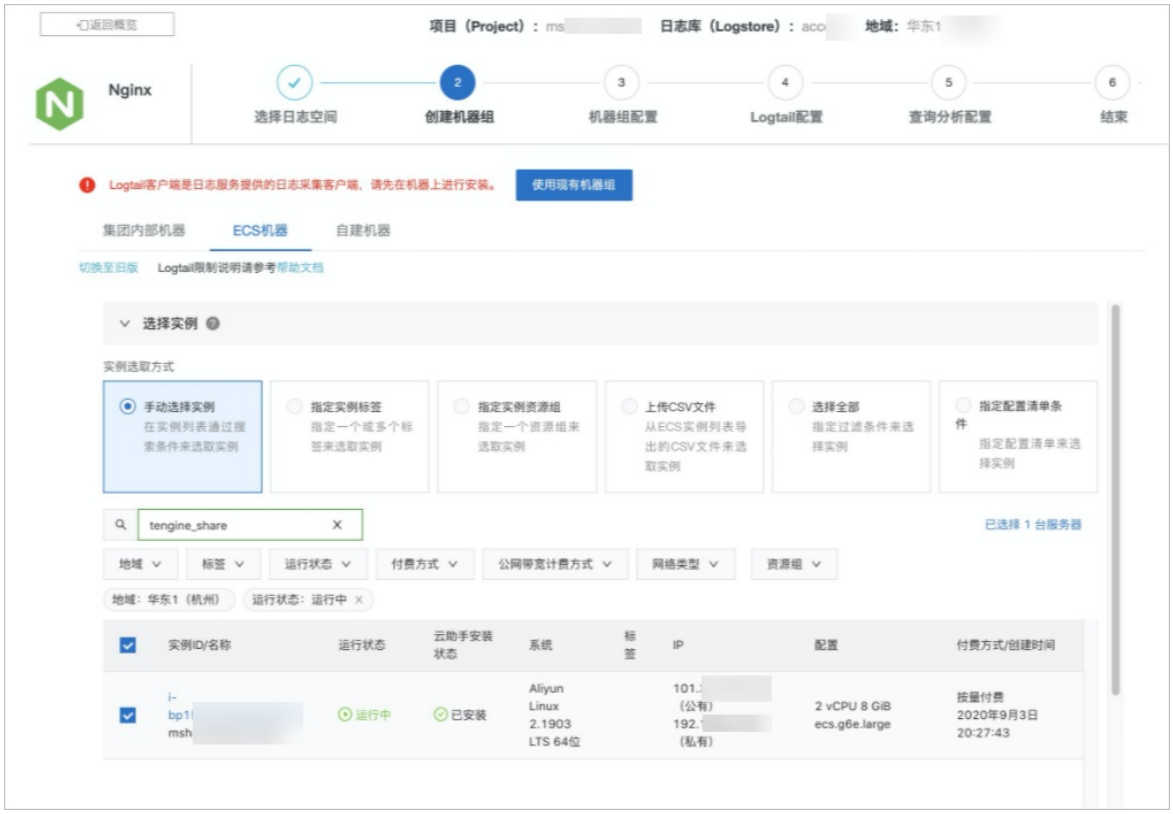
1. 单击日志库access左侧的 > 图标，展开该日志库。然后单击**数据接入**左侧的 > 图标并展开。
2. 将鼠标放置**logtail配置**区域，然后单击**logtail配置**右侧的 + 图标，添加数据接入。

> **说明** 通过Logtail配置页面安装Logtail日志收集插件以及配置要收集的日志路径等，Logtail日志插件信息请参见Logtail日志插件数据采集。

3. 在弹出的**快速数据接入**面板，单击**Nginx-文本日志**。



4. 在**Nginx**页面单击**ECS机器**页签，在**实例选取方式**区域选择**手动选择实例**或者**指定实例资源组**，然后选择需要进行日志收集的ECS实例，单击左下角的**立即执行**，然后单击右下角的**确定安装完毕**。



5. 在第②步中输入机器组名称，然后单击**下一步**，进入第③步**机器组配置**页面，在**我的机器组**区域，选择源机器组，然后单击**下一步**。
6. 在第④步**Logtail配置**页签，设置相关参数，完成配置。

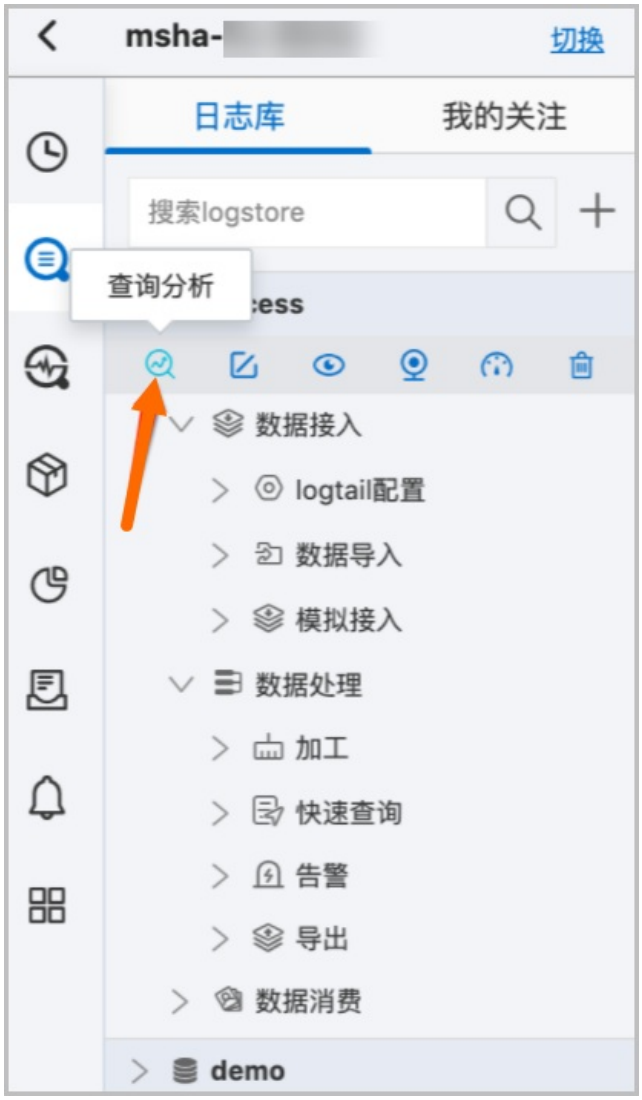
配置日志access_log，其中日志格式配置如下：

```
log_format proxyformat
 "$status|$upstream_status|$remote_addr|$upstream_addr|$request_time_usec|$upstream_response
_time|$time_local|$request_method|$scheme://$log_host:$server_port$request_uri|$body_bytes_s
ent|$http_referer|$http_user_agent|$http_x_forwarded_for|$http_accept_language|$connection_r
equests|$eagleeye_traceid|$cell|$ups|$ufe_code|$local_cell|$cell_key|$cell_router_id|$router
_rule|$local|$https_host_mode|$https_redirect_mode";
```

日志样例为：

```
200|200|127.0.xx.xx|115.29.xx.xx:80|16905|0.017|09/Jul/2021:19:49:29
+0800|GET|http://47.99.xx.xx:80/get?code=200|32|-|curl/7.29.0|-|-
|1|c0a80964162583136942xxxxxe6637|center|47_99_xx_xx@get_center_default|A1|1|||5bec72a6-8e0c
-4ad1-9846-87xxxxxbd034_unitcell_type|default|0|0_http
```

7. 返回**日志库**页面，单击 图标，查看access_log日志数据。

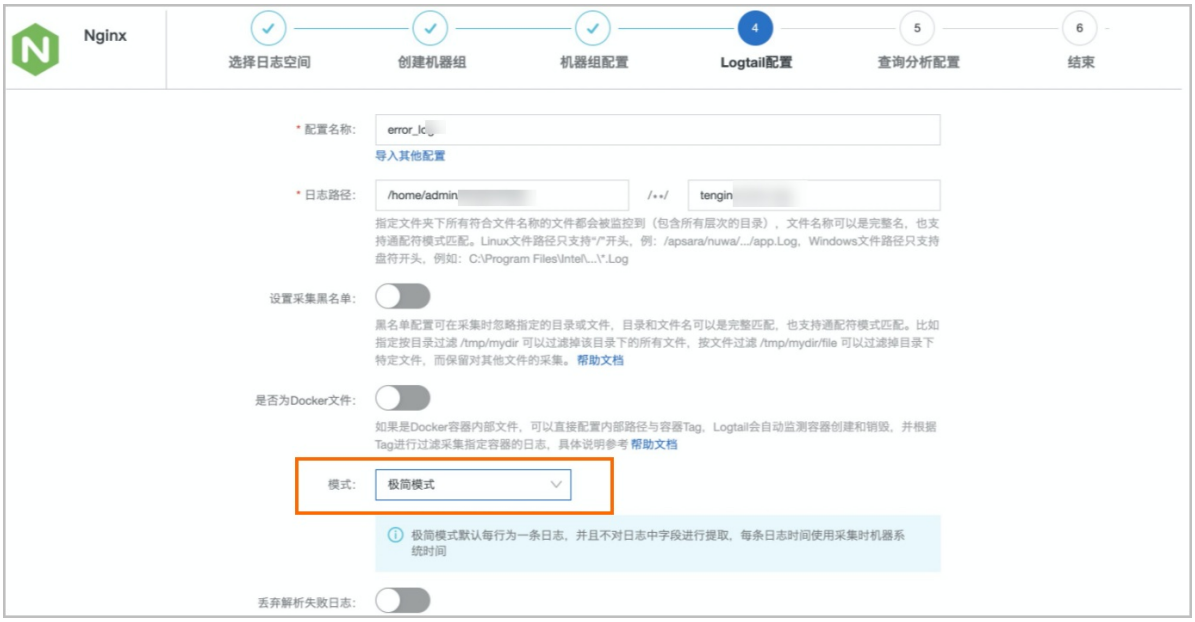


access_log日志数据详情，请参见查询和分析日志数据。

步骤三：配置error_log日志收集

配置error_log日志收集的步骤与配置access_log日志收集的步骤类似，因此请参见步骤二：配置access_log日志收集完成error_log日志的配置与收集。配置error_log日志过程中需注意以下几点：

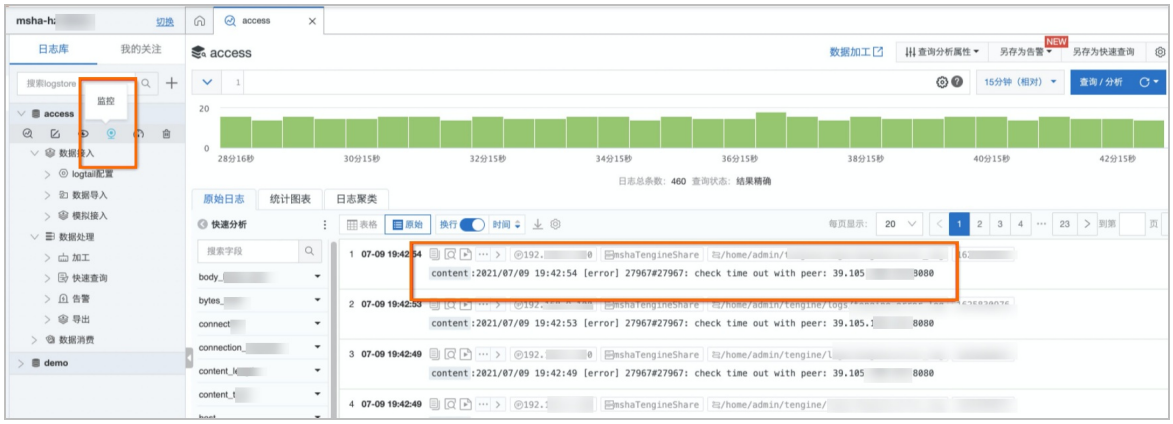
- 建议选择与access_log日志不同的Logstore对error_log日志进行logtail配置，以便进行独立的error_log报警配置。
- 由于error_log日志规则不是结构化的，因此在第④步**Logtail配置**页签设置相关参数时，其中**模式**参数选择**极简模式**即可，配置完成后，返回**日志库**页面，单击 图标，查看error_log日志数据。error_log日志数据详情，请参见查询和分析日志数据。



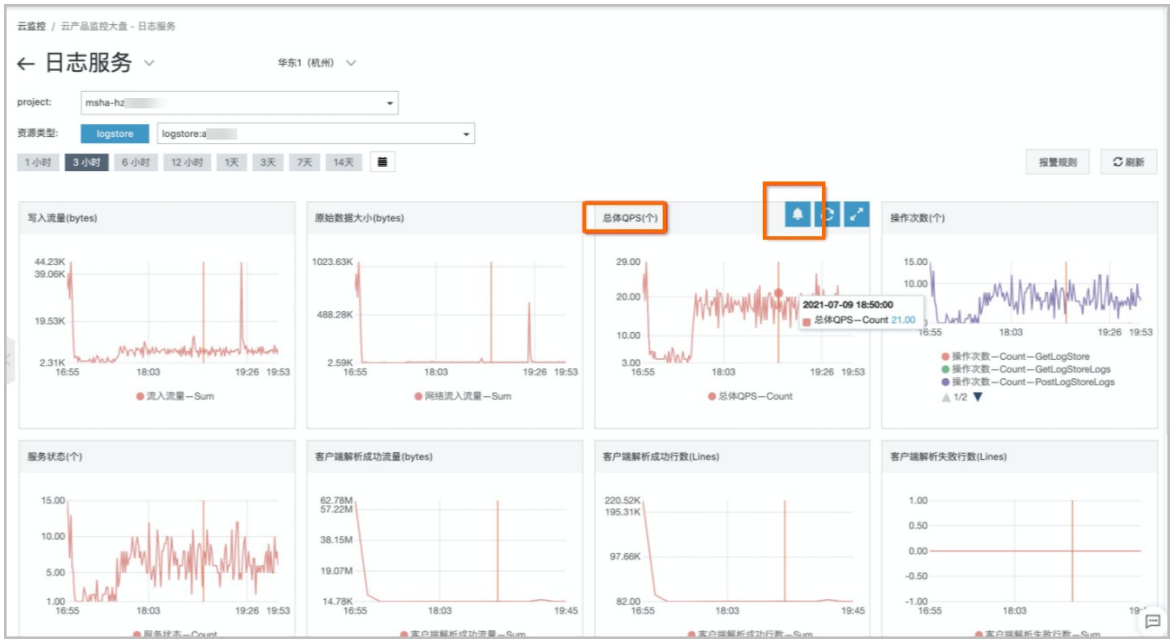
步骤四：配置error_log日志报警

若出现error_log日志，说明Tengine自身或后端服务器存在报错。建议您配置error_log报警规则，以便快速发现和处理报错问题。

1. 在**日志库**页面，单击 图标，查看error_log日志的监控信息。



2. 登录云监控控制台。
3. 在左侧导航栏选择**云产品监控**。
4. 在搜索框中输入**日志服务**，然后单击**日志服务**。
5. 选择对应Project名称右侧**操作**列的**监控图表**。
6. 在**日志服务**页面的**总体QPS（个）**区域右侧单击图标。然后在弹出的**创建报警规则**页面，根据相关参数，完成报警配置。具体操作，请参见创建报警规则。



相关文档

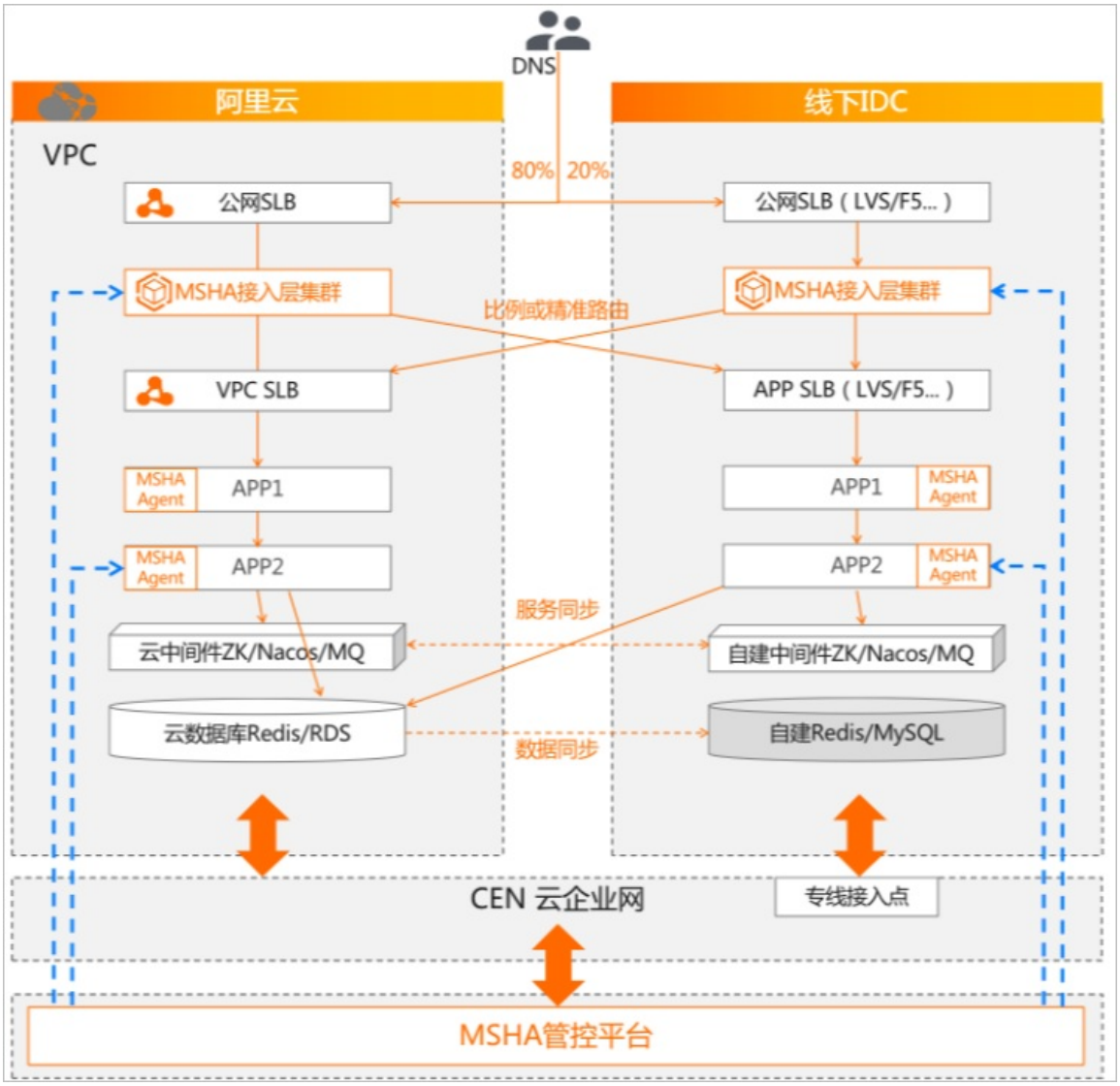
- 什么是日志服务
- 数据采集概述
- 使用Nginx模式采集日志
- 安装Logtail（ECS实例）
- 分析Nginx访问日志
- API概览
- 云监控
- 日志服务定价详情

5.4.10. MSHA应用双活架构接入Helloworld

本文通过一个Helloworld示例为您介绍如何接入MSHA应用双活架构并进行功能测试。

背景信息

应用双活容灾架构如下。更多信息，请参见混合云应用双活容灾最佳实践。



步骤一：前置准备

1. Demo准备。

   这里用电商交易平台Demo示例介绍。例如，该平台包含如下应用：

   - Frontend：Web应用，负责和用户交互。
   - Cartservice：购物车应用，提供购物车添加、存储和查询服务。
   - Productservice：商品应用，提供商品、库存服务。

   技术栈：

   - SpringBoot。
   - 微服务框架：SpringCloud、Dubbo，注册中心使用Nacos。
   - 数据库：Redis和MySQL。

2. 网络互通。

   您可以基于阿里云产品云企业网CEN实现网络互通。

   - 混合云网络打通：您可以通过高速通道物理专线、VPN网关、智能接入网关这3种方式打通网络。具体操作，请参见多接入方式构建企业级混合云。
   - 阿里云多Region网络互通：您可以通过云企业网实现多Region的网络互通。具体操作，请参见使用云企业

网实现跨地域跨账号VPC互通（企业版）。

步骤二：开通MSHA并录入多活资源

1. 开通并配置MSHA。具体操作，请参见开通并配置MSHA。
2. 配置业务类型、单元和命名空间。
   i. 登录AHAS控制台。
   ii. 在控制台左侧导航栏中单击**多活容灾**。
   iii. 在左侧导航栏选择**基础配置 > 业务类型和单元**，然后单击**新增业务类型**。
   iv. 在**新增业务类型和单元配置**页面，填写业务类型名称和业务类型标识，然后单击**+ 添加单元**，在弹出的**添加单元**面板，填写相关参数，完成新建业务类型和单元配置。

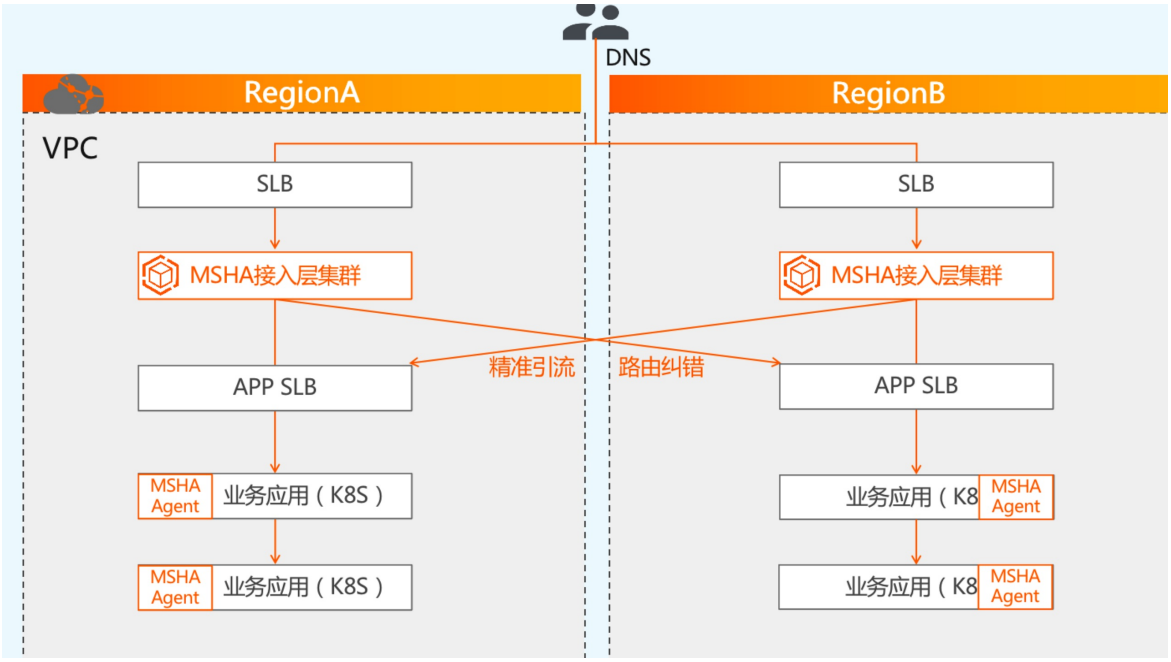
   > **说明** 业务类型通常用于业务隔离。例如，交易业务和导购业务可能使用了不同的容灾架构。

   v. 在左侧导航栏选择**基础配置 > 命名空间**，然后单击**新增命名空间**，配置架构类型、多活组件、单元下所使用的接入层集群和Nacos命令通道等。配置参数更多信息，请参见新建命名空间。

   > **说明**
   > - 命名空间通常用于环境隔离。例如，测试环境、预发环境、生产环境。
   > - MSHA使用Nacos作为管控命令通道，来实现管控命令的异步并发推送、容灾规则的多级缓存以及命令通道自身的高可用分布式部署能力。这里推荐直接购买阿里云MSE产品的Nacos实例，全托管且有稳定性SLA保障。

3. 部署MSFE接入层集群。在MSHA控制台，您可以将MSFE Tengine实例部署到多台ECS上，组成MSFE接入层集群。

   接入层集群部署架构图：



   i. 将ECS导入MSHA控制台用于部署MSFE集群实例。具体操作，请参见新增服务器。
   ii. 将SLB导入MSHA控制台用于MSFE集群的前置负载均衡。具体操作，请参见新增SLB。

iii. 创建MSFE集群，控制台会自动执行部署任务流程。具体操作，请参见创建集群或共享集群。

> 说明
> - 所需的ECS容量建议：生产环境需要跨可用区部署多台，同时需要根据业务峰值QPS来评估需要的ECS，通常建议1000 qps/c。例如，业务峰值QPS 1w，使用4 C、16 G规格的ECS，那么则需要的ECS台数为：1w/1000/4c=2.5（台）。
> - 您需要优先购买阿里云ECS和SLB服务。

4. 配置MSFE，包括录入域名/IP、URI、后端应用地址。

   创建好MSHA命名空间和MSFE接入层集群后，您就可以录入接入层域名、URI、后端应用回源地址等相关资源。录入并生效后，MSFE集群就会针对这些来源域名/IP的流量，来进行流量规则的处理和转发。具体操作，请参见配置MSFE。

   > 注意
   > - 新增域名时：
   >   若使用的是阿里云DNS，则域名解析类型选择**解析**，其他情况则选择**不解析**。
   > - 新增域名后，您还需要配置URI：
   >   - 架构类型：默认选择**应用双活**。
   >   - 回源应用IP:Port：需要填写后端应用负载均衡设备的IP和端口号。例如，上海单元（cn-shanghai）的域名和URI，对应的后端应用的负载均衡设备IP和端口为1.1.xx.xx:80。

5. 部署注册中心同步集群。

   在MSHA控制台，您可以将注册中心同步实例部署到多台ECS上，组成同步集群。具体操作，请参见步骤二：配置注册中心同步集群。

6. 配置注册中心同步任务。

   部署好注册中心同步集群后，即可配置注册中心服务同步任务，目前支持Nacos-Nacos、Zk-Zk。具体操作，请参见步骤四：配置注册中心同步任务。

7. 改造应用，即修改Redis/MySQL连接地址信息。

   改造前
   业务应用仅在杭州单元部署，应用的Redis/MySQL连接地址配置如下。

```
#Redis连接地址
spring.redis.host=r-***.redis.rds.aliyuncs.com
#MySQL连接地址
spring.datasource.url=jdbc:mysql://rm-***.mysql.rds.aliyuncs.com:3306/dbname?characterEncoding=utf8&useSSL=false&serverTimezone=GMT
```

   > 说明
   > - 以上为杭州单元的Redis/MySQL连接地址。
   > - 若您是SpringBoot应用，则连接地址配置在application.properties文件中。

   进行应用双活改造
   您需要在第二个单元（即北京单元）对称部署业务应用和Redis/MySQL数据库实例。北京单元部署的Redis/MySQL实例地址如下。

```
#Redis连接地址
r-***.redis.rds.aliyuncs.com
#MySQL连接地址
 rm-***.mysql.rds.aliyuncs.com
```

业务应用的Redis/MySQL连接地址配置中，需要添加上备数据库地址信息（即北京单元的数据库），以便MSHA SDK/Agent识别并实现主备切换功能。配置如下：

```
#Reids连接地址
spring.redis.host=r-***.redis.rds.aliyuncs.com?mshaStandbyHost=r-uuu.redis.rds.aliyuncs.com
#MySQL连接地址
spring.datasource.url=jdbc:mysql://r-***.redis.rds.aliyuncs.com:3306/dbname?characterEncoding=utf8&useSSL=false&serverTimezone=GMT&mshaStandbyHost=rm-***.mysql.rds.aliyuncs.com
```

> **说明** 以上为北京单元的Redis/MySQL连接地址。

改造后

配置改造完成后，应用还需挂载上MSHA Agent，您需要重启应用后才会生效。具体操作，请参见步骤三：为JAVA应用安装Agent。

8. 配置Redis/MySQL数据同步和数据保护规则。
   i. 在MSHA控制台的**管理数据源**页面单击**添加数据源**，录入Redis/MySQL相关数据库连接信息。具体操作，请参见步骤一：创建数据源。
   ii. 在**数据层配置**页面选择目标同步链路，然后单击**添加同步任务**，录入Redis/MySQL相关DTS同步任务信息。支持在MSHA控制台直接创建或录入已经创建好的DTS任务。

   > **说明** 建议您在DTS控制台提前创建好Redis/MySQL（RDS/PolarDB等）同步任务，并确保同步状态为“同步中”后再录入到MSHA控制台。

   iii. 在**数据层配置**页面单击**创建数据保护规则**，配置数据保护禁写、读写分离规则。然后分别进行灰度推送、全量推送，相应保护规则才会通过管控命令通道，推送到应用节点MSHA SDK/Agent上。
9. Redis/MySQL主备切换。
   i. 在**数据层配置**页面，查看当前生效的主数据库。当**类型**列的Read和Write均显示为 图标时则是主数据库，否则为备数据库。
   ii. 单击**操作**列的**主备切换**，进入到切换工单页面，等待预检查完成并提交工单后则会自动执行主备切换流程。
10. 发起切流。具体操作，请参见异地应用双活切流。

步骤三：为JAVA应用安装Agent

为JAVA应用安装Agent的具体操作，请参见为Java应用手动安装探针和配置Nacos作为规则下发通道。安装Agent之前您还需要先下载Agent并配置相关参数。具体操作如下。

1. 下载Agent。

地域	下载地址
杭州单元	http://msha-agent-hangzhou.oss-cn-hangzhou.aliyuncs.com/msha-java-agent.jar
北京单元	http://msha-agent-beijing.oss-cn-beijing.aliyuncs.com/msha-java-agent.jar

2. 配置JVM参数。

```
-javaagent:${探针安装路径}
-Dmsha.license=${当前license}
-Dmsha.namespaces=${命名空间ID}
-Dmsha.app.name=${应用名称}
-Dmsha.nacos.namespace=${Nacos命名空间ID}
-Dmsha.nacos.server.addr=${Nacos服务器访问地址}
```

配置参数说明：

- ${当前license}：您可以在MSHA控制台的**探针管理**页面查看License。
- ${命名空间ID}：您可以在MSHA控制台的**命名空间**页面查看。
- ${应用名称}：为您实际的应用名称，暂不支持中文。
- ${Nacos命名空间ID}：您可以进入Nacos实例的命名空间页面查看。如下图所示：

- ${Nacos服务器访问地址}：格式为${内网地址:内网端口}。例如，`mse-***-nacos-ans.mse.aliyuncs.com:8848`。

阿里云内应用挂载Agent JVM参数配置示例：

```
-javaagent:/home/admin/msha-agent/msha-java-agent.jar
-Dmsha.license=aaabbbccc-d810-4f9c-b8e2-e2836*****e8
-Dmsha.namespaces=aaabbbccc-8903-4a75-b10c-89ad7*****58
-Dmsha.app.name=rpc-demo
-Dmsha.nacos.namespace=aaabbbccc-27a6-4324-9ee7-4d52d*****e0
-Dmsha.nacos.server.addr=mse-aaabbbccc-nacos-ans.mse.aliyuncs.com:8848
```

IDC或其他云内应用挂载Agent JVM参数配置示例：

```
-Dregion-id=idc
-Dzone-id=idc-zone-a
-Dvpc-id=empty
-Downer-account-id=12*****89
-javaagent:/home/admin/msha-agent/msha-java-agent.jar
-Dmsha.license=aaabbbccc-d810-4f9c-b8e2-e2836*****e8
-Dmsha.namespaces=aaabbbccc-8903-4a75-b10c-89ad7*****58
-Dmsha.app.name=app-demo
-Dmsha.nacos.namespace=aaabbbccc-27a6-4324-9ee7-4d52d*****e0
-Dmsha.nacos.server.addr=171.0.xx.xx:8848
```

配置说明：由于IDC或者其他云内，无法获取到ECS机器所属的位置信息，所以相比阿里云内应用挂载Agent的JVM配置，需要增加以下内容：

- -Dregion-id：地域信息。
- -Dzone-id：可用区信息。
- -Dvpc-id：VPC网络ID。仅用于监控数据上报和MSHA控制台探针列表页展示，可配置为empty。
- -Downer-account-id：阿里云账号（主账号）ID。用于监控数据上报。IDC或者其他云与阿里云网络打通后，监控数据会上报至阿里云。

3. 重启应用。

   安装了探针的应用在启动时，会自动上报心跳信息。在MSHA控制台**探针管理**页面，若该应用实例信息显示在列表中且**状态**列为**在线**，则说明探针安装成功。

步骤四：验证双活能力

1. 验证MSFE路由能力。

   i. MSFE集群可视化运维。

验证	说明
测试内容	MSFE集群可视化运维
测试类型	人工UAT测试
前提条件	完成FE接入层集群部署
测试步骤	可视化运维能力包括：扩缩容、Metrics监控、实例健康检查、回源地址健康检查、规则一致性巡检。 
测试结果	符合预期

ii. MSFE引流及验证流量转发能力。

验证	说明
测试内容	MSFE引流及验证流量转发能力
测试类型	人工UAT测试
前提条件	■ 完成FE接入层集群部署完成 ■ 在MSHA控制台完成MSFE配置

测试步骤	a. 本地修改Hosts文件绑定域名，本地制造测试流量引流进入MSHA FE测试。
	Host绑定示例：
	`10.100.xx.xx ***.demo.com`
	`10.5.xx.xx ***.demo2.com`
	b. 变更DNS将生产环境流量引入MSHA FE。流量路径：DNS>SLB>MSFE集群。
	c. 根据流量比例规则，查看FE转发是否正常。例如，初始化时北京单元、杭州单元流量比例为100:0。
	a. 登录FE集群所在的ECS机器，查看Tengine访问日志是否正常、是否有打印访问记录，是否返回200，是否转发了北京单元的后端应用回源地址。文件地址：`/home/admin/tengine/logs/tengine-access_log`
	b. 登录到北京单元的业务应用机器，查看应用日志是否正常，并验证页面功能是否能够正常使用。

测试结果	□符合预期

iii. MSFE比例分流及切流。

验证	说明
测试内容	MSFE比例分流及切流
测试类型	人工UAT测试
前提条件	■ 完成FE接入层集群部署完成 ■ 在MSHA控制台完成MSFE配置
测试步骤	a. 切流。调整北京单元、杭州单元的流量比例为0:100，然后刷新电商Demo首页，查看调用链始终访问到杭州单元的Frontend入口应用。 b. 切流。调整北京单元、杭州单元流量比例为100:0，然后刷新电商Demo首页，查看调用链始终访问到北京单元的Frontend入口应用。 c. 切流。调整北京单元、杭州单元流量比例为50:50，然后刷新电商Demo首页，查看调用链，有一半的概率访问到杭州单元的Frontend入口应用。
测试结果	□符合预期

2. 验证数据库禁写、读写分离能力。

   i. 日常态读写主数据库（读写不分离）。

验证	说明
测试内容	日常态读写主数据库（读写不分离）
测试类型	人工UAT测试
前提条件	a. 应用挂载上MSHA-Agent。 b. MSHA控制台录入数据层配置，包括数据源、数据同步链路和数据保护规则。数据保护规则状态为生效，不允许读写分离，主数据库在北京单元，且北京单元的数据库Read及Write均显示 图标。 北京单元的数据库显示如下图：  数据保护规则如下图： 
测试步骤	a. 北京单元的应用执行数据库读操作（即查询商品详情）、写操作（即下单），应该访问到北京数据库。 b. 杭州单元的应用执行数据库读操作（即查询商品详情）、写操作（即下单），应该访问到杭州数据库。
测试结果	符合预期

ii. 日常态写主读本地（读写分离）。

验证	说明
测试内容	日常态读写主数据库（读写分离）
测试类型	人工UAT测试
前提条件	a. 应用挂载上MSHA-Agent。 b. MSHA控制台录入数据层配置，包括数据源、数据同步链路和数据保护规则。数据保护规则状态为生效，允许读写分离，主数据库在杭州单元，且杭州单元的数据库Read及Write均显示 图标。 
测试步骤	a. 两个单元内的应用执行数据库读操作（即查询商品详情），均访问到本单元数据库。 ■ 北京单元的应用执行数据库读操作，应该访问到北京数据库。 ■ 杭州单元的应用执行数据库读操作，应该访问到杭州数据库。 b. 两个单元内的应用执行数据库写操作（即下单），均访问到主数据库（即杭州单元）。 ■ 北京单元的应用执行数据库写操作，应该访问到杭州数据库。 ■ 杭州单元的应用执行数据库写操作，应该访问到杭州数据库。
测试结果	符合预期

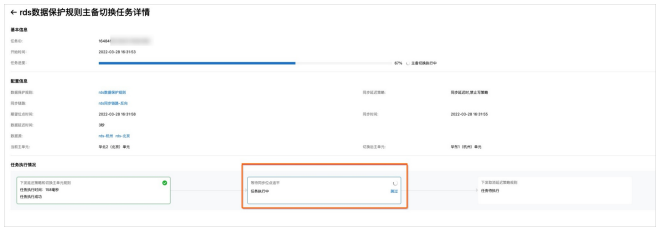
iii. 主备切换（读写不分离）。

验证	说明
测试内容	读写不分离下的主备切换
测试类型	人工UAT测试
前提条件	a. 应用挂载上MSHA-Agent。 b. MSHA控制台录入数据层配置，包括数据源、数据同步链路和数据保护规则。数据保护规则状态为生效，不允许读写分离，主数据库在杭州单元，且杭州单元的数据库Read及Write均显示 图标。
测试步骤	a. 主备切换前，2个单元内的应用执行读操作（即查询商品详情）、写操作（即下单），应该访问到杭州数据库。 b. 主备切换后（即将主数据库从杭州单元切换至北京单元），2个单元内的应用执行读操作（即查询商品详情）、写操作（即下单），应该访问到北京数据库。
测试结果	符合预期

iv. 主备切换（读写分离）。

验证	说明
测试内容	读写分离下的主备切换
测试类型	人工UAT测试
前提条件	a. 应用挂载上MSHA-Agent。 b. MSHA控制台录入数据层配置，包括数据源、数据同步链路和数据保护规则。数据保护规则状态为生效，允许读写分离，主数据库在杭州单元，且杭州单元的数据库Read及Write均显示 图标。
测试步骤	a. 主备切换前，杭州单元内的应用： ■ 执行数据库读操作（即查询商品详情），应该访问到杭州单元数据库。 ■ 执行数据库写操作（即下单），应该访问到杭州单元数据库。 b. 主备切换前，北京单元内的应用： ■ 执行数据库读操作（即查询商品详情），应该访问到北京单元数据库。 ■ 执行数据库写操作（即下单），应该访问到杭州单元数据库。 c. 主备切换后（即将主数据库从杭州单元切换至北京单元），北京单元内的应用： ■ 执行数据库读操作（即查询商品详情），应该访问到北京单元数据库。 ■ 执行数据库写操作（即下单），应该访问到北京单元数据库。
测试结果	符合预期

v. 主备切换数据同步延迟期间禁写。

验证	说明
测试内容	主备切换数据同步延迟期间禁写
测试类型	人工UAT测试
前提条件	a. 应用挂载上MSHA-Agent。 b. MSHA控制台录入数据层配置，包括数据源、数据同步链路和数据保护规则。数据保护规则状态为生效，不允许读写分离，主数据库在北京单元，且北京单元数据库Write显示，杭州单元数据库Write显示图标。
测试步骤	a. 主备切换前，北京单元内的应用执行写操作（即下单），应该访问到北京数据库且SQL执行成功无报错。 b. 制造同步存在延迟的场景（例如，暂停DTS同步任务），然后执行主备切换。那么主备切换流程中，执行到等待同步位点追平阶段后，就会长时间轮训DTS接口判断同步是否追平。  c. 主备切换等待数据同步追平过程中： ■ 杭州单元应用执行写操作（即下单），应该会被禁写（抛出异常）。 ■ 北京单元应用执行写操作（即下单），应该会被禁写（抛出异常）。 d. 主备切换结束后（即等待同步位点追平或跳过同步位点追平） ■ 杭州单元应用执行写操作（即下单），应该访问到杭州数据库且SQL执行成功无报错。 ■ 北京单元应用执行写操作（即下单），应该访问到杭州数据库且SQL执行成功无报错。
测试结果	符合预期