SOFAStack

微服务
运维指南

产品版本：AntStack Plus 1.11.0

文档版本：20220929

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

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.微服务

1.1. 系统组件

微服务（SOFAStack Microservices，简称 SOFAStack MS）主要提供分布式应用常用解决方案。使用微服务框架开发应用，并在应用托管后启动应用，微服务会自动注册到服务注册中心，您可以在微服务控制台进行服务管理和治理的相关操作。微服务主要提供分布式应用常用解决方案，包含 RPC 服务、动态配置、限流熔断等。本文介绍微服务的部署拓扑和资源清单。

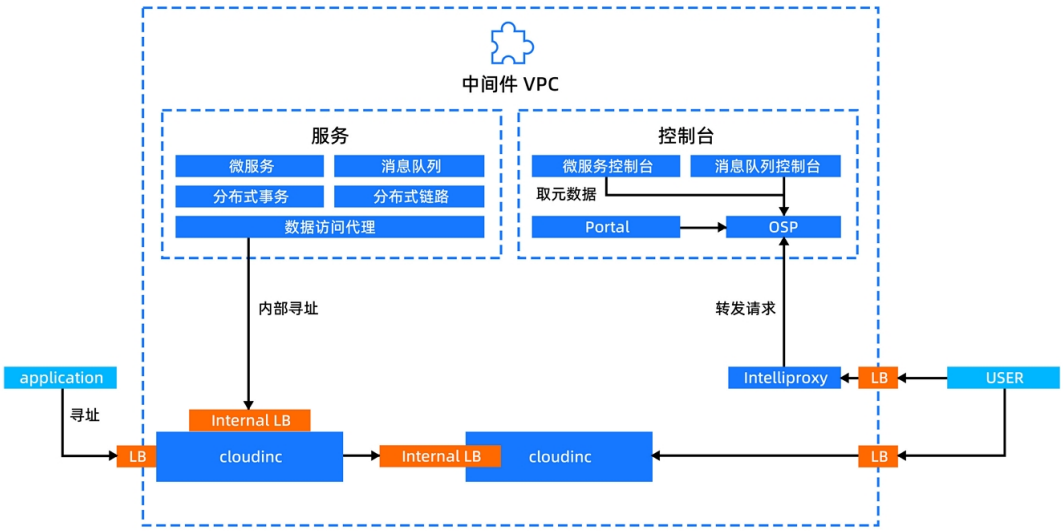
组件角色

微服务产品包括以下组件：

- **微服务控制台**：前端 Web 系统，提供所有图形界面化的操作。
- **注册中心**：主要提供所有服务注册信息的中心存储，同时负责将服务注册信息的更新通知实时推送给服务消费者。
- **定时任务**：提供统一通用的定时任务触发服务，提供定时任务的管理平台。
- **动态配置**：在分布式系统下为各应用的所有环境提供了一个中心化的外部配置，在不停应用集群的情况下，调整整个集群运行时的配置值。

部署拓扑

微服务部署拓扑如下图所示：



微服务产品下主要有以下小集群提供服务：

- sessionserver 和 dataserver 组成的集群提供服务注册发现服务，每台 sessionserver 都单独绑定一个 AnyTunnelSLB。sessionserver、dataserver 和 metaserver 组成的集群提供服务注册发现服务。每台 sessionserver 都单独绑定一个 AnyTunnelSLB。
- drmdata 集群提供动态配置服务，每台 drmdata 都单独绑定一个 AnyTunnelSLB。
- antscheduler 集群提供定时任务服务，目前定时任务触发需要依赖消息队列去通知应用。

- dsrconsole 集群是管理上述集群的控制台，提供友好的 UI 界面给开发者和管理员使用。

ACVIP、Intelliproxy 和 OSP 属于支撑产品，主要提供如下功能：

- ACVIP：提供寻址功能，帮助应用获取微服务产品的访问 IP。
- OSP：提供租户实例信息。
- Intelliproxy：主要提供用户认证和权限校验功能。

资源清单

微服务部署需要如下资源：

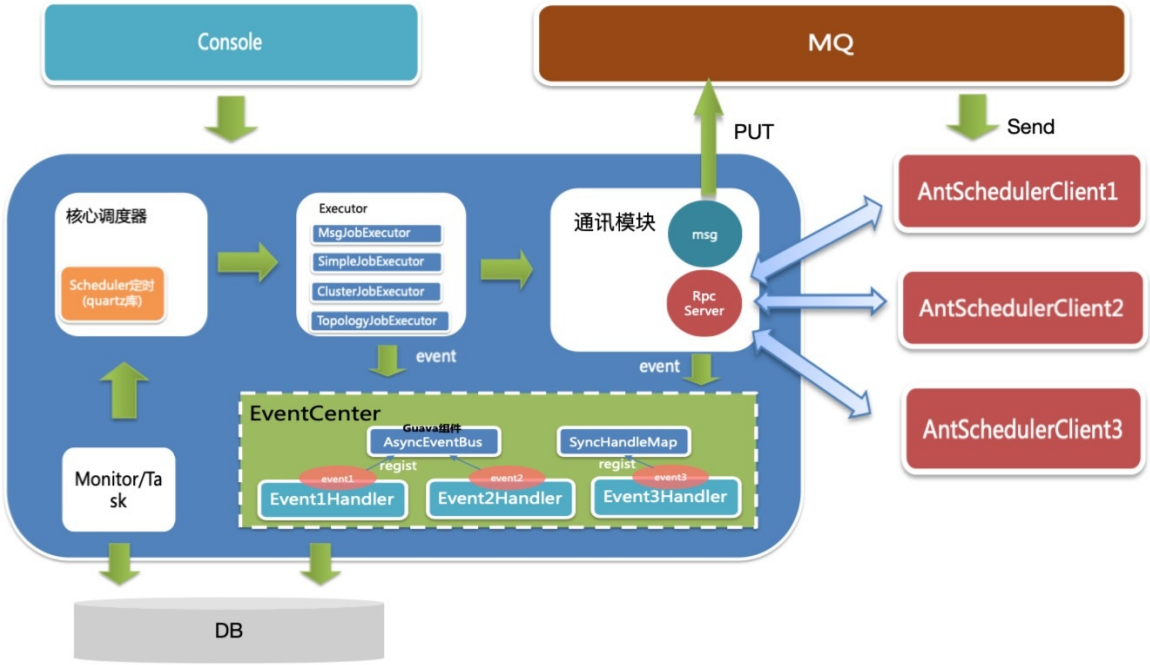
系统	服务器	服务器开放端口	LB 开放端口
SessionServer	2 台（4C8G，40G 磁盘）	9603、9600	9603、9600
MetaServer	3 台（4C8G，240G 磁盘）	9610、9611、9612、9615	9612
DataServer	3 台（8C8G，40G 磁盘）	9602、9604、9606	无
DrmData	2 台（2C4G，40G 磁盘）	80、9870、9880	80、9870、9880
AntScheduler	2 台（2C4G，40G 磁盘）	9000、9001、12200	9001、12200
DsrConsole	2 台（2C4G，40G 磁盘）	80、8341、12200	80
	MySQL 1台（2C4G，40G 磁盘）	无	无

1.2. 同城容灾

1.2.1. antscheduler 同城容灾

1.2.1.1. 产品架构

单机房架构



任务调度由以下几个部分组成：

- **Console**：是任务调度的控制面，一个可视化集中管理平台。方便管理任务、简化运维操作。
- **antscheduler**: 任务调度核心，负责任务的调度和集群间任务分配等功能。
- **Client**：是任务调度的客户端。业务方通过集成 Client 获取 RPC 和集群任务调度能力。
- 外部依赖：
  - **DB**：负责存储任务元数据、执行记录等信息，并作为任务调度集群注册表。
  - **MQ**：消息任务依赖于 MQ。任务调度时通过向 MQ 发送消息、客户端订阅消息的方式完成一次异步调度。

任务调度过程

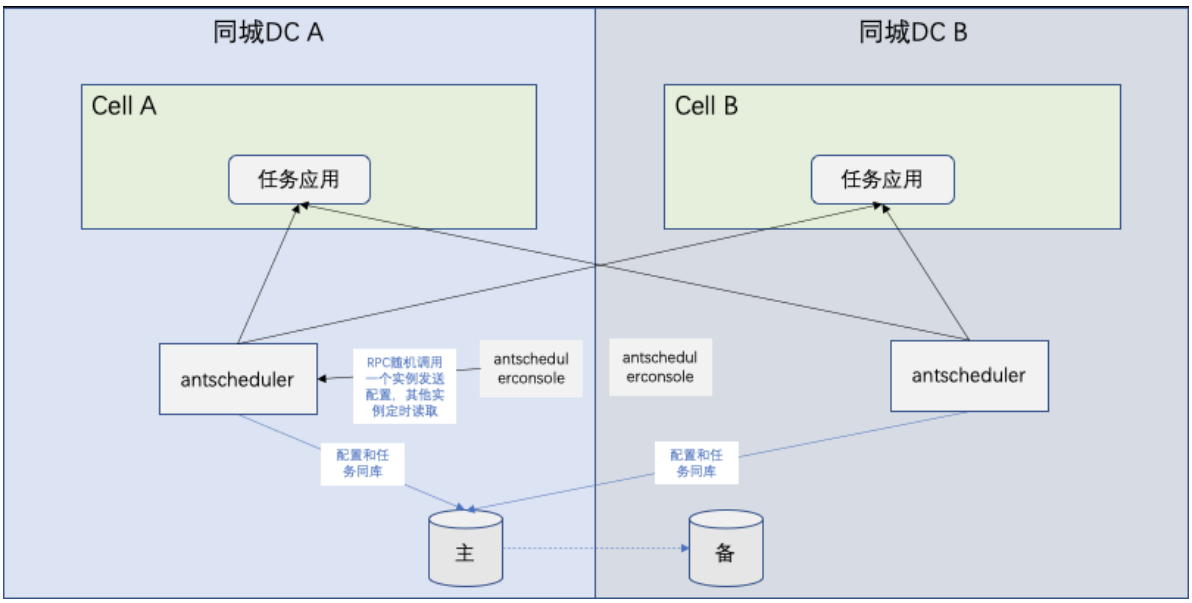
1. 通过 Console 向调度器模块添加任务，并安排调度。
2. 当任务满足执行条件时，任务调度器会触发任务调度，根据任务类型获取对应的调度执行器执行。
3. 调度执行器会通过事件方式创建触发记录、更新触发记录状态。
4. 调度器通过通讯模块通知客户端执行。

   RPC 采用 callback 模式时，客户端会将执行结果告诉服务端。服务端再根据事件模式更新任务执行状态。
5. 监听器监听漏触发和超时的执行记录，分别根据漏触发策略和超时策略进行处理。

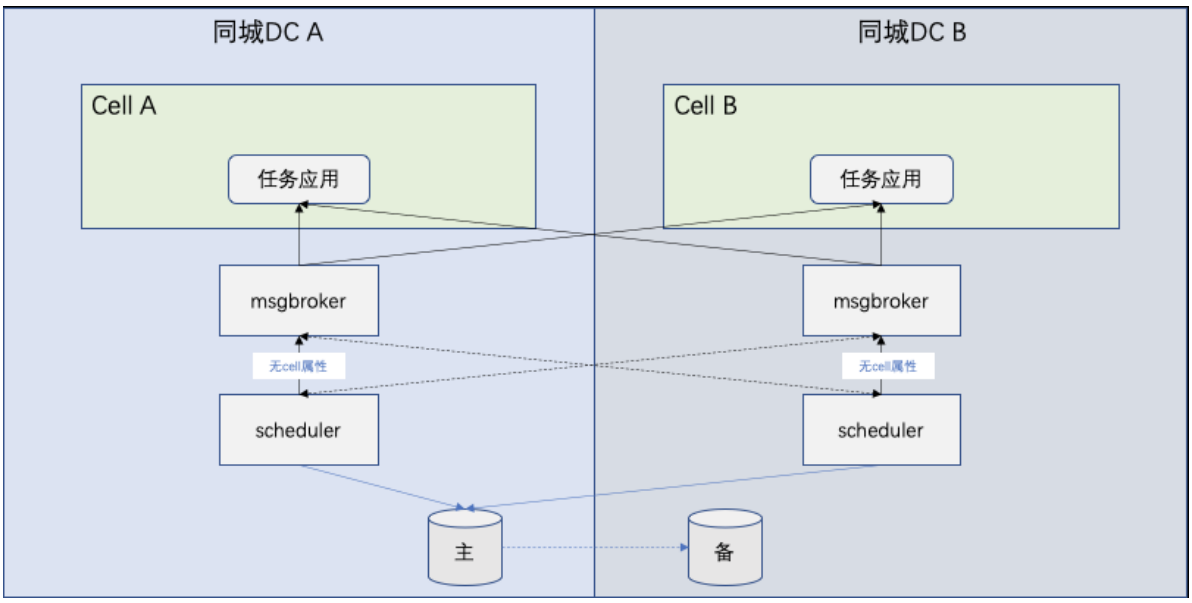
1.2.1.2. 容灾架构

架构图

- 基于 RPC



- 基于消息



部署时，主备机房需对等部署，数据库共库。

> 说明
>
> 由于本地机房延迟最低，TS 客户端优先同本地机房通信，本地机房不可用时，优先使用本地区机房。

产品依赖

外部产品依赖

- **ACVIP**：获取 SOFARegistry 地址。
- **SOFARegistry**：注册自身服务。
- **OSP**：通过调用 OSP facade 进行认证，及获取 instanceid、workspace 等信息。
- **MQ**：基于消息的定时任务需要用到。

底座产品依赖

- **元集群**：元集群主备机房独立部署，无容灾要求。
- **内网 DNS**：
  - Scheduler 通过内网 DNS 提供的域名访问数据库。容灾时，需将数据库域名切换至备机房的数据库。所以依赖内网 DNS 的容灾能力。
  - Scheduler 管控页面通过内网 DNS 提供的域名，双挂主备机房的内网负载均衡，对外提供服务。
- **内网负载均衡**：主备机房各自通过内网负载均衡进行负载。

内网 DNS 和 **内网负载均衡** 在不同底座下，有不同的选择。具体如下表所示：

底座	内网 DNS	内网负载均衡
AntStack	AntDNS	ALB
AntStack Plus 公有云	privatezone	SLB
AntStack Plus 物理机	AntDNS	-
AntStack Plus 物理机 + 飞天	AntDNS（forword opsdns）	-

> 注意
> 物理机底座下，AntStack Plus 目前没有内网负载均衡，实际部署时用客户环境选用的负载均衡设备。

数据库依赖

OB/RDS：主备机房共库，依赖 OB/RDS 的容灾能力。

数据同步

主备机房共库，依赖数据库的同步能力。

网络链路

- 业务应用通过 ACVIP 查询 Scheduler 服务地址，进行点对点通讯和 RPC 调用。
- 用户通过 intelliproxy 访问 MS-Scheduler 控制台。

容灾能力

支持机房级/组件级容灾，容灾切换后无功能降级。

1.2.1.3. 热切换方案

容灾切换说明

- 故障恢复阶段可能出现调度任务到还未完全恢复的机房应用，所以需要在故障后进行显示切换，使得任务只会调度到健康机房，恢复后再取消切换。任务必须幂等，确保任务失败可以无损重试。

- 数据库容灾切换，依赖数据库容灾能力。

切换逻辑

1. 切换元数据库 DB、内网 DNS。
2. 切换管控域名，摘除主机房流量。
3. 切换“已经触发，但还未上报状态”的定时任务。
4. 验证备机房 Scheduler 状态。

切换原则

保证数据不丢失，业务断开后可以立即恢复。

1.2.1.4. 容灾场景处理

常见容灾场景及处理方式如下：

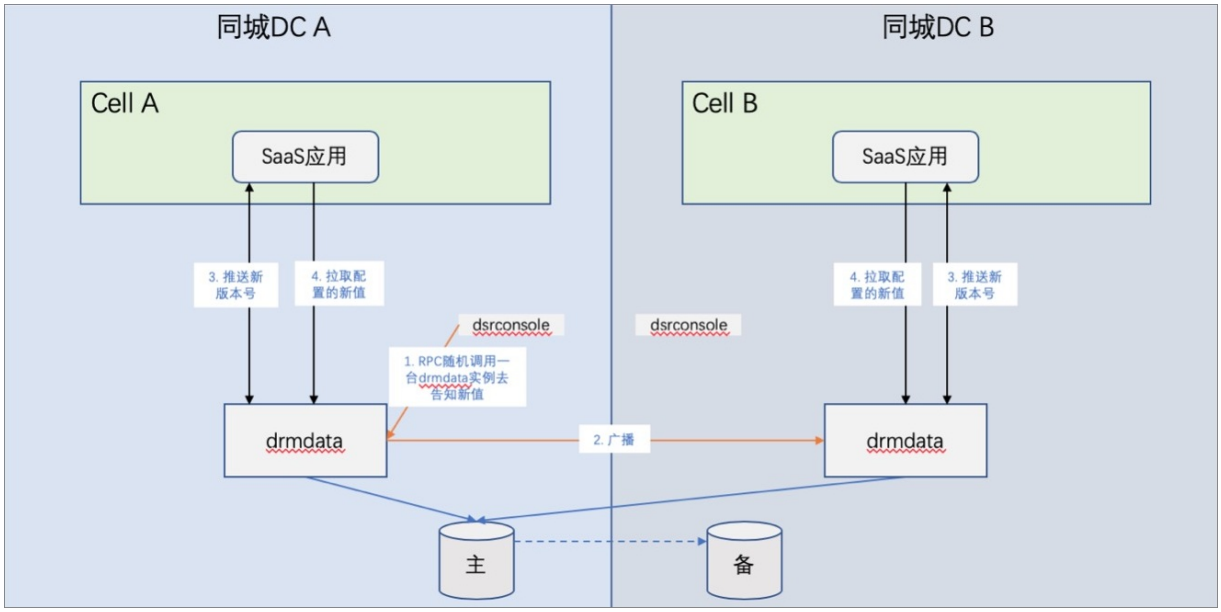
容灾场景	容灾切换	容灾恢复	容灾回切
主机房断电	1. 切换元数据库 DB。 2. 切换管控域名，摘除主机房流量。 3. 切换“已经触发，但还未上报状态”的定时任务。 4. 验证备机房 Scheduler 状态。	确定 ACVIP、SOFARegistry、OSP 恢复后，启动 Scheduler。	1. 切换元数据库 DB。 2. 切换管控域名，挂载主机房流量。 3. 取消切换“已经触发，但还未上报状态”的定时任务。 4. 验证主机房 Scheduler 状态。
备机房断电、网络孤岛、脑裂	1. 切换元数据库 DB。 2. 切换管控域名，挂载主机房流量。 3. 通过守夜人切换“已经触发，但还未上报状态”的定时任务。 4. 验证主机房 Scheduler 状态。	确定 ACVIP、SOFARegistry、OSP 恢复后，启动 Scheduler。	1. 切换元数据库 DB。 2. 切换管控域名，摘除主机房流量。 3. 通过守夜人取消切换“已经触发，但还未上报状态”的定时任务。 4. 验证备机房 Scheduler 状态。

> **说明**
>
> 网络恢复后会自动重连，在网络中断期间，Dsrconsole、ACVIP、OSP 及其他和管控台有关联的组件均无法使用。

1.2.2. DRM 同城容灾

1.2.2.1. 容灾架构

架构图



DRM 的完整推送流程如下：

1. 在 dsrconsole 上点击推送按钮。 最新的数据会被推送到 DB，DB 会返回此 Key 的最新版本号。
2. dsrconsole 通知推送消息到同机房的一台 drmdata 机器，消息内容包括：Key、Value、Version
3. drmdata 收到推送消息后，广播通知消息到所有的机房的 drmdata 机器。
4. 每台 drmdata 收到推送消息后，根据 key 查询所有 Client 建立的长连接信息，通过长连接发送给客户端 Key 最新的 Version。
5. Client 收到推送消息后，通过 drmdata 告知的最新版本号随机调用同机房 drmdata 提供拉取配置的 HTTP 接口，拉取最新的配置。

产品依赖

外部产品依赖

- **ACVIP**：获取 SOFARegistry 地址。
- **SOFARegistry**：注册自身服务。
- **OSP**：通过调用 OSP facade 进行认证，及获取 instanceid、workspace 等信息。

底座产品依赖

- **元集群**：元集群主备机房独立部署，无容灾要求。
- **内网 DNS**：
  - DRM 通过内网 DNS 提供的域名访问数据库。容灾时，需将数据库域名切换至备机房的数据库。所以依赖内网 DNS 的容灾能力。
  - DRM 管控页面通过内网 DNS 提供的域名，双挂主备机房的内网负载均衡，对外提供服务。
- **内网负载均衡**：主备机房各自通过内网负载均衡进行负载。

内网 DNS 和 **内网负载均衡** 在不同底座下，有不同的选择。具体如下表所示：

底座	内网 DNS	内网负载均衡
AntStack	AntDNS	ALB
AntStack Plus 公有云	privatezone	SLB
AntStack Plus 物理机	AntDNS	-
AntStack Plus 物理机 + 飞天	AntDNS（forword opsdns）	-

> 注意
> 物理机底座下，AntStack Plus 目前没有内网负载均衡，实际部署时用客户环境选用的负载均衡设备。

数据库依赖

OB/RDS：主备机房共库，依赖 OB/RDS 的容灾能力。

数据同步

主备机房共库，依赖数据库的同步能力。

容灾能力

支持机房级/组件级容灾，容灾切换后无功能降级。

1.2.2.2. 热切换方案

容灾切换说明

当单边机房不可用时，DRM 会直接从同城的另一个可用机房中拉取配置或者 Client 直接连接同城另一个可用机房的 DRM。

切换逻辑

1. 切换元数据库 DB、内网 DNS。
2. 切换 DRM 管控域名，摘除主机房流量。
3. 验证备机房 DRM 状态。

切换原则

保证数据不丢失，业务断开后可以立即恢复。

1.2.2.3. 容灾场景处理

常见容灾场景及处理方式如下：

容灾场景	容灾切换	容灾恢复	容灾回切
主机房断电或网络孤岛	1. 切换元数据库 DB 内网 DNS。 2. 切换 DRM 管控域名，摘除主机房流量。 3. 验证备机房 DRM 状态。	确定 ACVIP、SOFARegistry、OSP 恢复后，启动 DRM。	1. 切换元数据库 DB 内网 DNS。 2. 切换 DRM 管控域名，挂载主机房流量。 3. 验证主机房 DRM 状态。
备机房断电、网络孤岛、主备机房脑裂	1. 切换元数据库 DB 内网 DNS。 2. 切换 DRM 管控域名，摘除备机房流量。 3. 验证主机房 DRM 状态。	确定 ACVIP、SOFARegistry、OSP 恢复后，启动 DRM。	1. 切换元数据库 DB 内网 DNS。 2. 切换 DRM 管控域名，挂载备机房流量。 3. 验证备机房 DRM 状态。

说明

网络恢复后会自动重连，在网络中断期间，Dsrconsole、ACVIP、OSP 及其他和管控台有关联的组件均无法使用。

1.3. 日常运维

1.3.1. 监控和预警

1.3.1.1. 监控指标和预警项

系统监管指标和预警项清单

监控指标	监控指标说明	预警触发条件
dsrconsole 磁盘空间监控	磁盘空间占用率监控	大于 80%
dsrconsole 内存监控	内存占用率	大于 80%
dsrconsole CPU 利用率	CPU 利用率	大于 80%
sessionserver 磁盘空间监控	磁盘空间占用率监控	大于 80%

监控指标	监控指标说明	预警触发条件
sessionserver 内存监控	内存占用率	大于 80%
sessionserver CPU 利用率	CPU 利用率	大于 80%
metaserver 磁盘空间监控	磁盘空间占用率监控	大于 90%
metaserver CPU 利用率	CPU 利用率	大于 90%
metaserver 内存监控	内存占用率	大于 90%
dataserver 磁盘空间监控	磁盘空间占用率监控	大于 80%
dataserver 内存监控	内存占用率	大于 80%
dataserver CPU 利用率	CPU 利用率	大于 80%
drmdata 磁盘空间监控	磁盘空间占用率监控	大于 80%
drmdata 内存监控	内存占用率	大于 80%
drmdata CPU 利用率	CPU 利用率	大于 80%
antscheduler 磁盘空间监控	磁盘空间占用率监控	大于 80%
antscheduler 内存监控	内存占用率	大于 80%
antscheduler CPU 利用率	CPU 利用率	大于 80%

自定义监控指标和预警项清单

监控指标	监控指标说明	预警触发条件

监控指标	监控指标说明	预警触发条件
服务注册中心水位	水位包含服务注册中心当前注册服务数量，客户端连接数等。 订阅角色：运维管理员	单机客户端连接数超过 2000。
服务注册中心内部长连接断连事件	内部长连接断开一般伴随有网络不稳定等情况，断连会触发一些修复性事件，需关注。 订阅角色：运维管理员	最近 1 分钟内断连超过 1 次。
服务注册中心运行时列表变更及推送	运行时列表变更说明有节点宕机，触发集群自动剔除节点，需考虑替换故障节点。 订阅角色：运维管理员	出现集群列表变更。
服务注册中心端口监控	服务注册中心 Session 角色 9600 可用状态。 订阅角色：运维管理员	9600 端口不可用。
动态配置单机长连接数量	动态配置单机处理客户端长连接数量。	-
动态配置集群长连接数量	动态配置当前集群处理客户端长连接数量。	-
动态配置集群单机推送量	动态配置当前集群单机推送属性值变更总量，分钟级别。	-
动态配置单机 QPS	动态配置服务端以 HTTP 形式返回数据，单机设计 QPS 4000，超过此数据时需考虑扩容。	-
定时任务消息集群发送量	全集群消息的发送总量，分钟级别。 订阅角色：运维管理员	• 当前时间昨日同比下跌 20%。 • 最近5 分钟求和与上 5分钟求和环比下跌超过 20%。 • 最近 5 分钟求和 >10。 多个条件同时满足触发预警。

监控指标	监控指标说明	预警触发条件
定时任务消息发送单机量	定时任务单机消息发送量，分钟级别。	-
微服务服务端存活监控	服务端存活。 订阅角色：运维管理员	8341 端口不可用。
服务注册中心服务端存活监控	服务端存活。 订阅角色：运维管理员	9600、9602 端口不可用。
动态配置服务端存活监控	服务端存活。 订阅角色：运维管理员	9880、12200 端口不可用。
定时任务服务端存活监控	服务端存活。 订阅角色：运维管理员	8080、12200 端口不可用。

1.3.1.2. 预警项运维动作

预警项	预警项说明	运维动作
antscheduler	基于消息的任务通过 MQ 发送消息，需要和 MQ 的服务器建立长连接，当 9529 端口没有正常的连接时，消息无法发送。	确认MQ 是否运行正常： • 若 MQ 服务正常运行，查看 `/home/admin/logs/common-error.log` 日志，检查是否有和 MQ 建立连接失败的日志（例如“no available connection”）。如果没有任何信息，重启服务器再观察。 • 若 MQ 服务运行异常，重启服务器再观察。

预警项	预警项说明	运维动作
	基于 RPC 的任务，客户端和服务端需要建立长连接。您可以通过 9001 端口查看是否有客户端注册。	如果 9001 端口没有任何连接，可能因为没有客户端存在。如果客户端正常运行，可以查看 `/logs/auth-audit.log` 日志是否有客户端的注册信息。如果没有任何信息，说明客户端的注册请求为发送到服务端，您需要查看客户端 `/logs/scheduler/common-error.log` 中注册失败的日志。
dsrconsole	微服务控制台主要提供页面访问服务，主要关注 80、8341、12200 端口的健康状况。	如果出现任意端口关闭，重启应用即可。
drmdata	动态配置主要依靠 9880 推送数据变更给客户端，客户端通过 9880 与服务端建立长连接。	如无客户端注册，可能 ACVIP 寻址配置异常，建议检查 ACVIP 配置。

1.3.1.3. 日常巡检项

antscheduler 系统监控指标巡检项目

巡检项

通过 RMS 控制台查看微服务产品各项系统监控指标。访问方式：

1. 登录 RMS 控制台。
2. 在左侧导航栏单击 **应用监控**，然后单击 **全部应用** 页签。
3. 在 **搜索应用** 文本框输入 `antscheduler`，然后单击 按钮。
4. 单击目标应用名称，即可查看系统监控详情。

运维动作

如果发现预警指标异常，根据对应预警项运维动作说明进行操作。

dsrconsole 系统监控指标巡检项目

巡检项

通过 RMS 控制台查看微服务产品各项系统监控指标。访问方式：

1. 在 RMS 控制台的 **应用监控** 页面，搜索 `dsrconsole`。
2. 单击目标应用名称，即可查看系统监控详情。

运维动作

如果发现预警指标异常，根据对应预警项运维动作说明进行操作。

drmdata 系统监控指标巡检项目

巡检项

通过 RMS 控制台查看微服务产品各项系统监控指标。访问方式：

1. 在 RMS 控制台的 **应用监控** 页面，搜索 `drmdata` 。
2. 单击目标应用名称，即可查看系统监控详情。

运维动作

如果发现预警指标异常，根据对应预警项运维动作说明进行操作。

1.3.2. 系统日志

1.3.2.1. 日志文件清单

系统	日志文件名称	日志文件路径	日志文件描述
session-server	registry-session.log	/home/admin/logs/registry/session/registry-session.log	Session 注册的主要日志。服务发布和服务订阅，以及一些内部任务执行的日志都存在这个文件。
	registry-exchange.log	/home/admin/logs/registry/session/registry-exchange.log	Session 和其他角色交互的日志，比如是否发送pub 到 dataserver 的日志等。
	registry-console.log	/home/admin/logs/registry/session/registry-console.log	Session 和 dsrconsole 的交互日志。此外，session 的 pub 和 sub 数量记录在这个日志。
	registry-connect.log	/home/admin/logs/registry/data/registry-connect.log	Sesssion 连接日志，记录 Session 和 dataserver 之间的连接、Session 和客户端的连接、Session 和meta 的连接变更等。
	gc.log	/home/admin/logs/gc.log	GC 日志

系统	日志文件名称	日志文件路径	日志文件描述
data-server	registry-data.log	/home/admin/logs/registry/data/registry-data.log	Dataserver 的主要日志，data 服务注册以及内存变更和数据备份都集中记录在这个日志。
	registry-connect.log	/home/admin/logs/registry/data/registry-connect.log	Dataserver 连接日志，主要记录 dataserver 之间的连接，以及 dataserver 和 session、meta 的连接变更等。
	gc.log	/home/admin/logs/gc.log	GC 日志
meta-server	registry-meta.log	/home/admin/logs/registry/meta/registry-meta.log	Metaserver 的主要日志，记录 data 和 session 节点 IP 注册过程。
	registry-raft.log	/home/admin/logs/registry/meta/registry-raft.log	Metaserver 接入 jraft 选举过程的主要日志。
	registry-connect.log	/home/admin/logs/registry/meta/registry-connect.log	Metaserver 和其他节点连接的日志。
	gc.log	/home/admin/logs/gc.log	GC 日志
dsrconsole	common-error.log	/home/admin/logs/dsrconsole/common-error.log	系统错误日志
	common-error.log	/home/admin/logs/drmdata/common-error.log	系统错误日志

系统	日志文件名称	日志文件路径	日志文件描述
	app-auth.log	/home/admin/logs/drmdata/app-auth.log	业务系统访问 drmdata 的授权信息，只有通过授权的系统才能访问drmdata 获取配置信息。 **说明** 授权是针对 instanceId 的。
	drmdata-container.log	/home/admin/logs/drmdata/drmdata-container.log	客户端向服务端注册信息。 **说明** 客户端通过向服务端注册长连接来监听配置变更事件。
	drmdata-default.log	/home/admin/logs/drmdata/drmdata-default.log	系统打印的所有的日志信息，包含 INFO、WARNING、ERROR 日志。

系统	日志文件名称	日志文件路径	日志文件描述
drmdata	drmdata-heartbeat.log	/home/admin/logs/drmdata/drmdata-heartbeat.log	客户端心跳信息。 客户端定时每隔 30s 向 drmdata 上报所使用的所有配置项的版本信息，服务端检查客户端上报的配置版本，如果落后则通知客户端更新对应配置。 日志样例： `Y;0ms;GZ00B;UX8YY6AL9CHS;100.81.77.9;240;0;bolt` 。 字段说明： • Y/N：此次心跳检查是否正常。 • 0ms：检测耗时。 • GZ00B：客户端所属单元。 • UX8YY6AL9CHS：instanceId 信息。 • 100.81.77.9：客户端 IP。 • 240：此次心跳客户端携带的配置数。 • 0：此次心跳检测出客户端配置版本落后的数量。 • bolt：心跳检测来源。

系统	日志文件名称	日志文件路径	日志文件描述
	drm-monitor.log	/home/admin/logs/drmdata/drm-monitor.log	drmdata 维护的长连接信息。 日志样例： `[monitorZone]11;DEFAULT_ZONE;4;32;3;` 。 字段说明： • [monitorZone]：zone 维度监控。 • 11：总连接数 • DEFAULT_ZONE：名称 • 4：instanceId 数量 • 32：配置项数 • 3：当前 zone 的连接数
	drmdata-push.log	/home/admin/logs/drmdata/drmdata-push.log	配置变更时，drmdata 向客户端推送变更信息日志。 字段说明： `[push]{instanceId};{配置项};{客户端IP};{0(推送成功)/1(推送失败)}` 。
	drmdata-sync.log	/home/admin/logs/drmdata/drmdata-sync.log	drmdata 集群间同步配置信息。
	drmdata-threadpool.log	/home/admin/logs/drmdata/drmdata-threadpool.log	drmdata 所有线程池监控信息。 字段说明： `{线程池名称},{队列中的任务数},{活跃线程数},{空闲线程数},{总线程数}` 。
	rpc-client.log	/home/admin/logs/antscheduler/rpc-client.log	客户端连接日志

系统	日志文件名称	日志文件路径	日志文件描述
antscheduler	rpc-server.log	/home/admin/logs/antscheduler/rpc-server.log	服务端连接事件日志
	rpc-slave.log	/home/admin/logs/antscheduler/rpc-slave.log	其他服务端建立连接日志
	job-rpc.log	/home/admin/logs/antscheduler/job-rpc.log	RPC 任务执行日志
	job-msg.log	/home/admin/logs/antscheduler/job-msg.log	MSG 任务执行日志
	job-dispatch.log	/home/admin/logs/antscheduler/job-dispatch.log	任务分配日志
	monitor.log	/home/admin/logs/antscheduler/monitor.log	监控日志
	common-error.log	/home/admin/logs/antscheduler/common-error.log	异常日志
	antscheduler-default.log	/home/admin/logs/antscheduler/antscheduler-default.log	启动日志

1.3.2.2. 异常日志运维动作

系统	日志文件名称	报错内容	日志文件描述
-	-	get server list error	客户端未拿到服务端地址列表，可能是服务端寻址配置有误。
-	-	Register task schedule error	客户端注册发布或订阅任务失败，可以确认下是否注册过于频繁。

系统	日志文件名称	报错内容	日志文件描述
-	-	SubscriberNotifyTask execute error, dataId:	订阅数据回调任务执行出错，可以根据异常信息查看下回调方法内是否有处理错误。
-	-	ConfiguratorNotifyTask execute error	配置数据回调任务执行出错，可以根据异常信息查看下回调方法内是否有处理错误。
-	-	Failed trying connect to {}	尝试与服务端建立连接失败，服务端可能有问题。
-	-	add notify task error, dataId: {}, registId: {}	回调任务加入线程池失败，需要确认回调线程池配置是否过小，没有能力消费掉服务端的推送。
drmserver	/home/admin/logs/drm/common-error.log	Query zdrmdata server address empty	未能从 ACVIP 获取到 zdrmdata 服务地址，检查 ACVIP 地址设置是否正确。
data-server	decision.60dt.log	/home/admin/logs/confreg-data/decision.60dt.log	Data 集群踢节点日志，当有故障节点出现时，主节点会剔除故障节点并分发列表给所有 Session、Data 节点。
antscheduler	/home/admin/logs/antscheduler/rpc-server.log	client[xxx:xxx] has no auth	客户端没有权限，查看客户端的 AccessKey 和 SecurityKey 配置是否正确。
	/home/admin/logs/antscheduler/rpc-server.log	client register failed	客户端注册时发生异常，可以查看堆栈信息了解异常原因。
	/home/admin/logs/antscheduler/job-rpc.log	no targets was found	查看客户端注册情况

系统	日志文件名称	报错内容	日志文件描述
	/home/admin/logs/antscheduler/job-msg.log	xxx,xxx,xxx,xx, failed	发送消息失败，检查消息订阅关系是否生效。

1.3.2.3. 日常巡检项

查看微服务相关应用资源和存储资源

巡检项

访问路径：**产品与服务 > 发布部署服务 > 应用 >** `dsrconsole/session-server/data-server/drmdata/scheduler` 。

运维动作

如果发现预警指标异常，根据对应预警项运维动作说明进行操作。

1.3.3. 服务巡检

1.3.3.1. 系统组件监控检查

登录 RMS 监控系统，查看微服务（MS）相关系统组件有无异常告警，需要关注的监控项内容如下表所示。

基础监控检查

应用	分类	指标（关键字）	告警阈值
MS-antscheduler MS-antschedulerconsole MS-drmdata MS-dsrconsole	基础监控-CPU 使用率监控	cpu_usage	>80%
	基础监控-内存使用率监控	mem_usage	>80%
	基础监控-磁盘使用率监控	disk_usage	>80%
	基础监控-load5	load5	>3
MS-antscheduler	基础监控-端口监控	2022、12200	不通
MS-antschedulerconsole	基础监控-端口监控	8080、 12200	不通
MS-drmdata	基础监控-端口监控	2022、9880、12200	不通

应用	分类	指标（关键字）	告警阈值
MS-dsrconsole	基础监控-端口监控	80、2022、8341、12200	不通

自定义业务监控检查

通过 RMS 内核态监控产品查看注册中心产品各项系统监控指标是否有异常告警产生。

如果有异常告警产生，联系技术支持进行排查。

1.3.3.2. 业务功能检查

1. antscheduler 服务验证。 如果需要使用基于消息的功能，需要依赖 msgbroker。此时可执行以下命令，检查是否与 msgbroker 建立连接。

   ```
   netstat -an|grep 9529
   ```

2. 访问 `sofa-ms.{domain}` 控制台页面，检查各页面是否能正常访问无报错。

1.3.4. 管理容器

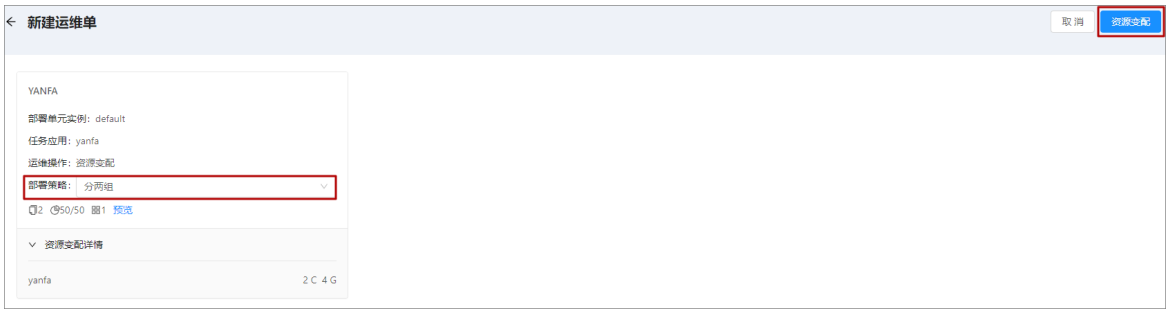
1.3.4.1. 变更容器配置

您可以通过云游 Local 控制台对应用的容器规格配置进行变更。

操作步骤

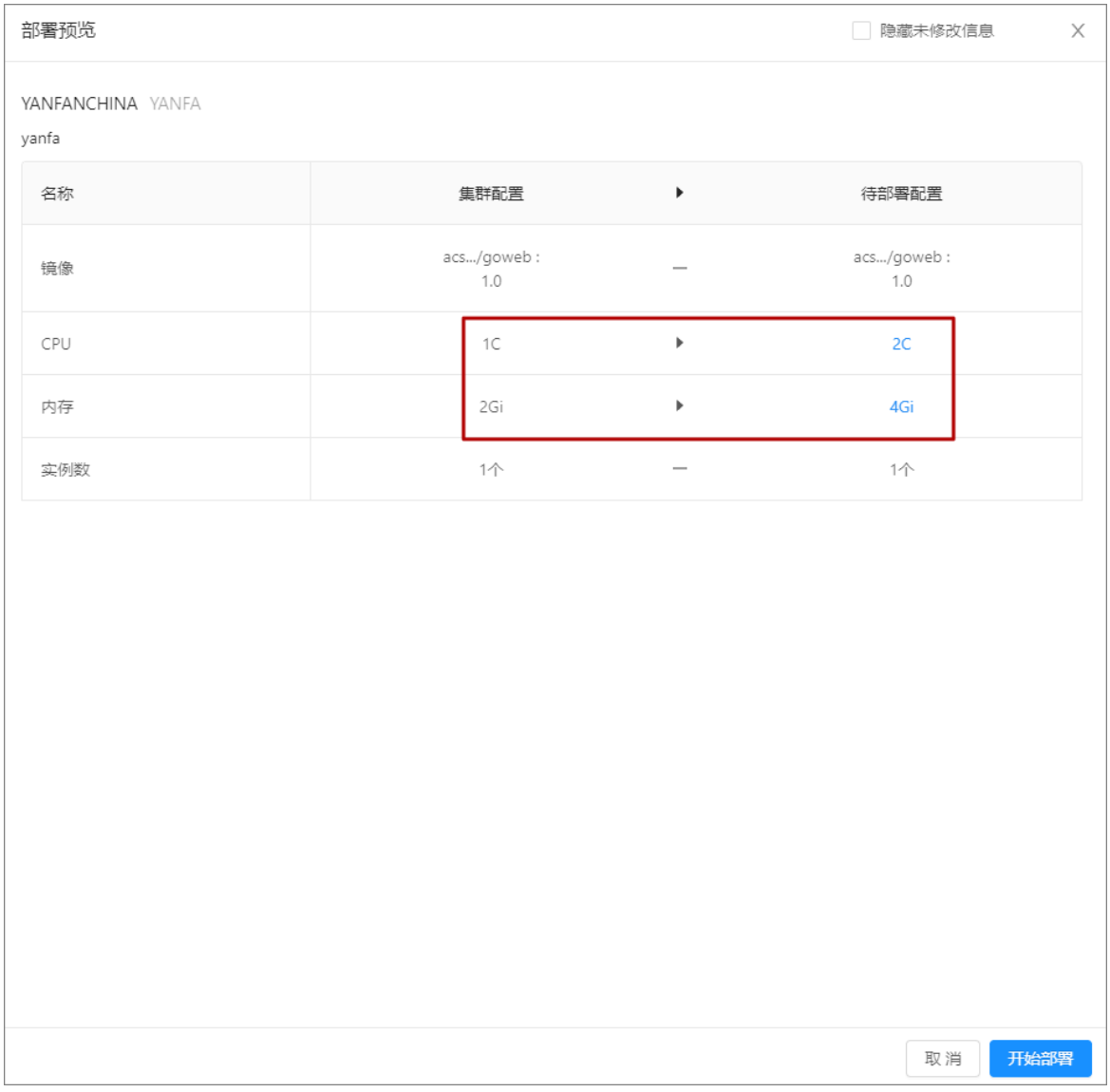
1. 登录云游 Local 控制台。
2. 在左侧导航栏，选择 **产品运维** > **产品基线**。
3. 在产品列表中，单击目标产品。
4. 单击 **应用** 页签，之后单击目标应用。
5. 单击 **容器** 页签，之后单击 **容器变配**。
6. 在 **应用变配规格** 对话框设置 **目标 CPU** 和 **目标内存** 参数，之后单击 **确定**。

7. 在 **新建运维单** 页面，将 **部署策略** 改为 **分两组**。



8. 单击 **资源变配**，在**部署预览**面板确认部署参数无误后单击 开始部署，之后单击 **确定**。

   这里主要确认当前实例的 CPU、内存和期望变更的 CPU、内存。



> **注意**
>
> 变配之后，AntStack Plus 底座环境，容器会自动进行滚动重启；AntStack 1X 和 2X 底座环境，您需要分批次手动重启容器。详情请参见 重启容器。

1.3.4.2. 容器内存与 JVM 堆内存配置比

中间件各产品容器内存与 JVM 堆内存的配置比如下：

产品名称	JVM 堆内存建议配置
动态配置 DRM	根据环境自动适配，适配方式如下： • 系统内存 = 2 GB；JVM = 1000 MB • 系统内存 = 4 GB；JVM = 3000 MB • 系统内存 = 8 GB；JVM = 6000 MB • 系统内存 = 10 GB；JVM = 9000 MB • 系统内存 = 14 GB；JVM = 12000 MB
注册中心 SOFARegistry	3/4 系统内存
DsrConsole	根据环境自动适配，适配方式如下： • 系统内存 = 2 GB；JVM = 1000 MB • 系统内存 = 4 GB；JVM = 3000 MB • 系统内存 = 8 GB；JVM = 6000 MB • 系统内存 = 10 GB；JVM = 9000 MB • 系统内存 = 14 GB；JVM = 12000 MB
任务调度 TS	根据环境自动适配，适配方式如下： • 系统内存 = 2 GB；JVM = 1000 MB • 系统内存 = 4 GB；JVM = 3000 MB • 系统内存 = 8 GB；JVM = 6000 MB • 系统内存 = 10 GB；JVM = 9000 MB • 系统内存 = 14 GB；JVM = 12000 MB
消息队列 MQ	系统自动调整，无需设置。
消息队列 DMS	Console：系统内存 = 4 GB；JVM = 3 GB Broker：系统内存 = 8 GB；JVM = 4 GB
分布式事务 DTX	JVM 堆内存小于系统内存。 例如系统内存为 8 GB，JVM < 8 GB。

产品名称	JVM 堆内存建议配置
数据访问代理 ODP	根据环境自动适配，适配方式如下： • 系统内存 < 2 GB；JVM 使用系统默认值（根据不同版本进行调整） • 2 GB < 系统内存 < 6 GB；JVM = 2048 MB • 6 GB < 系统内存 < 10 GB；JVM = 4800 MB • 10 GB < 系统内存 < 14 GB；JVM = 5600 MB • 系统内存 > 14 GB；JVM = 9600 MB
Mesh	不涉及
API 网关	不涉及
消息队列 SOFAKafka	系统自动调整，无需设置。

1.3.4.3. 重启容器

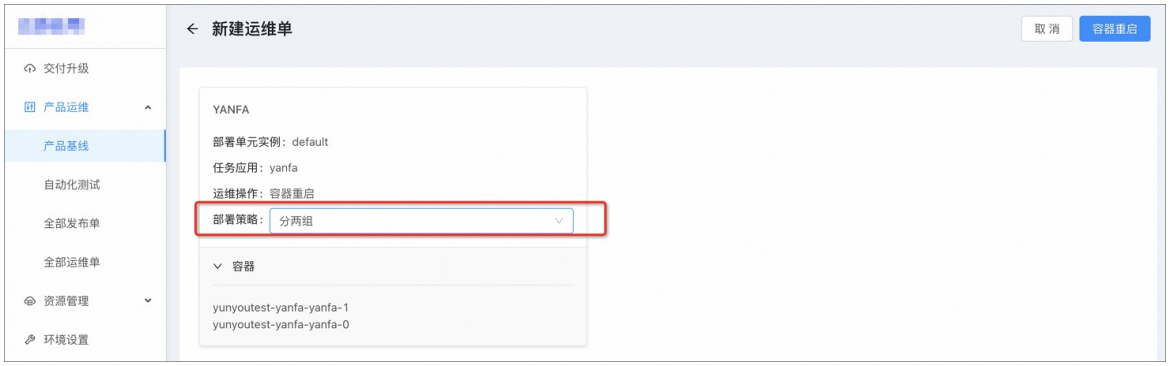
重启容器是指将指定容器（Pod）销毁后重新按期望调度出相同数量的新容器。

操作步骤

1. 登录云游 Local 控制台。
2. 在左侧导航栏，选择 **产品运维** > **产品基线**。
3. 在产品列表中，单击目标产品。
4. 单击 **应用** 页签，然后单击目标应用。
5. 单击 **容器** 页签，然后单击 **批量操作** 的 **重启** 按钮。

6. 选中一个或多个目标容器后单击 **确定**。
7. 在 **新建运维单** 页面单击 **容器重启**，然后单击 **确定**。

如果您选择了多个容器，建议将 **部署策略** 修改为 **分两组**，以免所有运行容器同时销毁，导致业务中断。



1.4. 常见故障处理

1.4.1. RPC 错误码

错误码	错误信息	中文日志	英文日志	可能原因
100004701	ERROR_LDC_ZMODE_FALSE	当前环境的 zmode 开关为 false，LDC 功能将不支持，调用方应在外围做 zmode 判定，请确认。	LDC is off: zmode=false.	• `ystemProperty` 中设置的 zmode 参数不为空，且不为 true。 • `com.alipay.ldc.zone` 设置的 zone 名称没有以 r、dr、g、c、i 开头，且 zmode 为空。
100004702	ERROR_LDC_APPNAME_EMPTY	没有相应的 appName 信息，计算 zone 实例启动失败。	Get ZoneRouteStrategy instance failed, cannot get appName.	`consumerConfig` 中没有 appName 配置，请补充该参数。
100004703	ERROR_LDC_ZMG_URL_EMPTY	获取 zone manager url 失败，计算 zone 实例启动失败。	Get ZoneRouteStrategy instance failed, cannot get zonemng url.	`systemProperty` 中 `zonemng_zone_url` 设置为空，请设置正确值。

错误码	错误信息	中文日志	英文日志	可能原因
100004704	ERROR_LDC_ROUTE_STRATEGY_EMPTY	获取路由计算实例失败，请检查检查配置。	Get ZoneRouteStrategy instance failed, please check configuration.	定义的 `route_stratgy_class` 类在通过反射初始化时遇到错误。
100004707	ERROR_LDC_TARGET_ZONE_EMPTY	按 zone 软负载路由时目标 zone 不能为空。	Target zone cannot be empty while routing by zone.	`zoneAddressContext` 中 `idcSwitch` 开关已打开，但是 `targetZone` 为空。
10000461 6	ERROR_IP_TRANSMIT_REGISTRY_NUL	IpTransmitRegistry 为 null, 请在开始传输之前设置。	IpTransmitRegistry is null, please set it before start transmit.	`TransmitRegistry` 未设置。
10000461 7	ERROR_START_ZK_IP_TRANSMIT_REGISTRY	启动 ZookeeperIpTransmitRegistry zkClient 失败。	Failed to start ZookeeperIpTransmitRegistry zkClient.	启动 zkClient 失败，请查看详细堆栈信息。
10000461 8	ERROR_REG_IP_TO_TRANSMIT_REGISTRY	注册 transmit ip 到 ZookeeperIpTransmitRegistry 失败!	Failed to register transmit ip to ZookeeperIpTransmitRegistry!	注册 transmit IP 到 ZookeeperIpTransmitRegistry 失败，请查看详细堆栈信息。
10000461 9	ERROR_SUB_IP_FROM_TRANSMIT_REGISTRY	从 ZookeeperIpTransmitRegistry 订阅 transmit ip 失败!	Failed to subscribe transmit ip from ZookeeperIpTransmitRegistry!	从 ZookeeperIpTransmitRegistry 订阅 transmit IP 失败，请查看详细堆栈信息。
10000462 0	ERROR_ZK_CLIENT_UNAVAILABLE	Zookeeper client 不可用。	Zookeeper client is not available.	zkClient 为空或 zkClient 状态不为 STARTED。
10000462 1	ERROR_ONLY_ONE_PARAM	基于{0}的类: [{1}], 只支持一个入参!	class based{0}: [{1}], only support one parameter!	某些协议规定 façade 只能包含一个参数。

错误码	错误信息	中文日志	英文日志	可能原因
100004622	ERROR_PROTOBUF_RETURN	基于 protobuf 的类:{0},只支持返回 protobuf message!	class based protobuf: {0}, only support return protobuf message!	protobuf 协议规定返回值必须是 protobuf message。
100004623	ERROR_METHOD_NOT_FOUND	找不到方法: [{0}].[{1}]	Cannot found method: [{0}].[{1}]	没有找到服务提供方的对应方法，请检查调用是否正确。
100004624	ERROR_INVALID_ATTRIBETE	{0}非法 ,原始 URL=[{1}]	{0}is invalid ,originUrl=[{1}]	将 URL 转换成 ProvinderInfo 时，URL 中的属性非法。
100004625	ERROR_AUTH_VERIFY	身份验证被拒绝，token 为空或者验证失败。	Access rejected by the identity verify, token is null or verify fail.	身份验证被拒绝，token 为空或验证失败。
100004626	ERROR_GET_TOKKEN	获取 token 错误,请求中未带有 token。	get token error. request without token.	获取 token 错误，请求中未带有 token，请查看详细堆栈信息。
100004627	ERROR_SCHEDULED_EXECUTOR	向 SofaScheduledExecutorService 提交任务失败。	Submit command into SofaScheduledExecutorService error.	向 SofaScheduledExecutorService 提交任务失败，请查看详细堆栈信息。
100004628	ERROR_CACHE_SIZE_OVER	服务端身份验证缓存到达阈值 = [{0}]。被忽略的 token = [{1}]。	Provider auth identity cache size is beyond threshold = [{0}]. Ignore set token = [{1}].	
100004629	ERROR_ENV_ERROR	获取 {0} 环境变量错误。	get{0}env error.	获取{0}环境变量错误，请查看详细堆栈信息。

错误码	错误信息	中文日志	英文日志	可能原因
100004630	ERROR_DYNAMIC_MANAGER_LOAD	加载 dynamic config manager 失败.	dynamic config manager load error.	加载 `dynamic config manager` 失败，请查看详细堆栈信息。
100004631	ERROR_CONSUMER_IDENTITY_REFRESH	定时刷新Consumer identity 失败.	Consumer identity refresh scheduler error.	定时刷新 `Consumer identity` 失败，请查看详细堆栈信息。
100004632	ERROR_REMOVE_TOKEN_FROM_CACHE	从缓存中删除 token 失败,token:{0}.	remove token:{0}from cache error.	从缓存中删除 token 失败，token:{0}。请查看详细堆栈信息。
100004633	ERROR_REG_DRM_OBJECT	DRM 注册对象 object:{0}失败.	DRM register object{0}error.	DRM 注册对象 object:{0}失败。请查看详细堆栈信息。
100004634	ERROR_COMMAND_BEYOND_QUEUE_SIZE	提交任务数大于队列最大长度:{0}.	Submitted command num is beyond the maxQueueSize{0}.	提交任务数大于队列最大长度:{0}。
100004635	ERROR_SIGNATURE_SIGN_ERROR	签名错误, service:{0}, ak: [{1}], host: [{2}]	signature sign error, service:{0}, ak: [{1}], host: [{2}]	签名时抛出IOException。
100004636	ERROR_BATCH_UNREG	批量反注册发生错误。	Error when batch unregistry.	批量反注册发生错误，请查看详细堆栈信息。
100004637	ERROR_BATCH_UNSUB	批量反订阅发生错误.	Error when batch unSubscribe.	批量反订阅发生错误，请查看详细堆栈信息。

1.4.2. 常见问题

1.4.2.1. 定时任务（antscheduler）日常问题

基于 msg 定时任务配置完成后无法触发

1. 检查环境配置项 `ACVIP.endpoint` 和 instanceId 是否正确按照对应环境进行了配置。
2. 检查代码中的 Eventcode 是否和定时任务控制台、消息中心控制台一致，消息中心控制台的消息类型及订阅关系是否完整配置并且触发生效。
3. 排查机器日志 `logs/tracelog/msg-sub-digest.log` ，确认是否有收到消息。

   如果收到了消息，表明任务已经触发，只是业务日志没有打印或者没有执行业务，需要排查代码对应片段进行分析。

基于 msg 定时任务没有按照频率触发

定时任务依赖消息队列投递，消息队列对于客户端处理超时或者失败的场景会自主发起重试。如果业务机器收到定时任务消息后抛出异常，或由于集群资源有限导致消息处理超时，消息队列会发起重试。

排查机器日志 `logs/tracelog/msg-sub-digest.log` ，日志格式如下：

```
2014-06-19 20:13:33.734,,0ad1348f14031800144081003,0.1.d6502e1f,6fafc09cd670fa39a802626ab1a
ab2b0,1e4cf860680dad989dceec17f1adb837,_NOTIFYTEST,sofa30,P-P-helloworld-pub-group,01,1094B
,5ms,msgbroker.rz00b.alipay.net,0,msgWorkTP-6646123-1-thread-1,mark=T&uid=a2&
```

其中，逗号分隔的倒数第三位表示这条消息的投递次数，如果大于 0，表明当前的消息是重试消息。

如果发现重试消息，可以将对应定时任务频率降低甚至暂停，待重试消息消化完成后手动触发，确认是由于业务异常导致的必现重试，还是由于频率过高导致的处理能力不足。

基于 RPC 定时任务配置后接收不到触发请求

- 如果是 CALLBACK 类型，查看触发记录和执行记录是否成功。
- 如果是 ONEWAY 类型，先改成 CALLBACK 类型再触发试试。一般情况这个问题是由于客户端和服务端没有成功连接导致。

服务节点宕机。

服务节点宕机，直接重启即可。

宕机后，其负责的任务会被分配给其他机器，在未重新分配之前这段时间内需要触发的任务会出现漏触发情况。

1.4.2.2. 动态配置（drmdata）日常问题

应用发布后控制台无法查看到订阅者列表

1. 确认 `ACVIP.endpoint` 配置是否正确，实例标识是否正确。
2. 排查客户端业务应用日志（非 drmdata 机器日志） `logs/drm/drm-boot.log` 中注册的动态配置标识是否与控制台属性对应的配置标识一致。
3. 执行以下命令，排查客户端业务应用日志（非 drmdata 机器日志） `logs/drm/drm-monitor.log` 中配置资源是否注册成功。

   ```
   grep "registersuccess"
   ```

1.4.2.3. DsrConsole 日常问题

DsrConsole 不能访问

1. 确认其他产品控制台是否正常。

   您可以尝试访问其他产品的控制台，例如任务调度，若无法访问，则检查是否是 intelliproxy、OSP、IAM 出现异常；若可以访问，则继续执行下一步。

2. 重启 DsrConsole 容器。

   重启 DsrConsole 不会对您的业务产生任何影响。

1.4.2.4. RPC 常见问题

1.4.2.4.1. RPC 调用出现“maybe write overflow!”报错

问题现象

RPC 调用出现“maybe write overflow!”报错，具体如下：

```
Caused by: com.alipay.remoting.exception.RemotingException: Check connection failed for address: Origin url [openexprod.d1756.alipay.net:12200], Unique key [openexprod.d1756.alipay.net:12200]., maybe write overflow! at com.alipay.remoting.DefaultConnectionManager.check(DefaultConnectionManager.java:250) at com.alipay.remoting.rpc.RpcClientRemoting.invokeWithCallback(RpcClientRemoting.java:71) at com.alipay.remoting.rpc.RpcClient.invokeWithCallback(RpcClient.java:529) at com.alipay.sofa.service.hsf.tbremoting.invoke.component.CallbackInvokeComponent.invoke(CallbackInvokeComponent.java:96) ... 150 more
```

问题原因

RPC 在写出数据时，会检测当前的 `ChannelOutboundBuffer` 大小，若大小超过 `WRITE_BUFFER_HIGH_WATER_MARK`（默认 64 KB），会触发限流，报 write overflow 的异常。 该举措是为了保护客户端，防止无界的 `ChannelOutboundBuffer` 被占满，导致资源耗尽。以下情况会导致异常：

- 网络出现故障，导致写出失败。
- 下游设备压力太大，无法处理超量的网络包。

解决方案

- 压测场景
  i. 排查网络环境。

     您可以通过 `ping` 命令确认网络是否存在丢包、延迟过大等情况。

  ii. 若网络环境没问题的，请确认是否是流量过大，导致下游设备处理不及时。

     如果是流量过大导致的问题，表示已达到压测的瓶颈。您可以通过扩容下游设备、优化性能等方式解决。

- 生产场景

- 建议将 RPC 调用的包拆成多个小包，进行多次调用（推荐）。
- 设备扩容，使用高配网卡机器，以提升网络处理能力以及下游应用处理速度等。

  此方案能一定程度上缓解问题，但不能从根本上解决问题，推荐使用上面的方案。

如果 `WRITE_BUFFER_HIGH_WATER_MARK` 的默认值（64 KB）不能满足您的环境，您也可以在 JVM 参数中修改以下参数：

```
-Dbolt.netty.buffer.low.watermark=32768    // 根据您的实际需求修改，例如设置为 32 KB，则该值为 32 × 1024。
-Dbolt.netty.buffer.high.watermark=65535  // 根据您的实际需求修改。
```

1.4.2.4.2. RPC 序列化和反序列化错误排查

当客户端发生序列化或反序列相关异常时，首先查看 `serviceSide` 字段的值：

```
com.alipay.sofa.rpc.core.exception.SofaRpcException: com.alipay.remoting.exception.SerializationException: Server serialize response exception! the address is 10.253.25.28:12200, id=4527, serverSide=true
```

如果值为 `true`，代表异常发生在服务端，您需要登录服务端，根据客户端收到错误的时间查看日志 `home/admin/logs/bolt/common-error.log`，并根据如下报错信息进行处理：

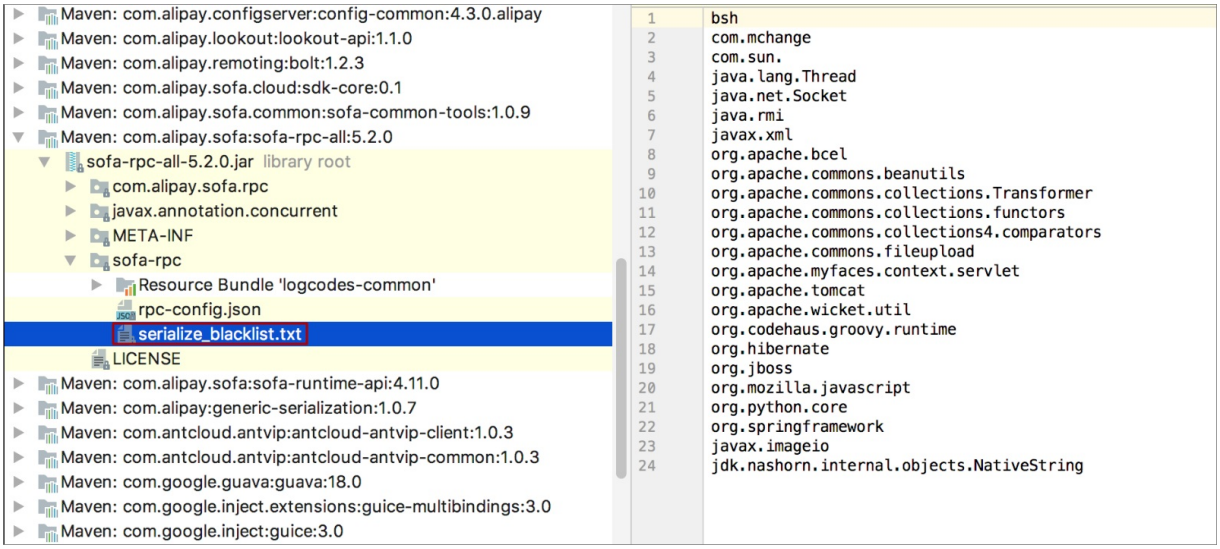
xxx is in blacklist

错误日志样例：

```
2018-03-09 15:58:50,481 ERROR [RpcRequestProcessor#154] [SofaBizProcessor-12200-0-T40] SerializationException occurred when sendResponseIfNecessary in RpcRequestProcessor, id=4527
com.alipay.remoting.exception.SerializationException: Class org.springframework.beans.factory.NoSuchBeanDefinitionException is in blacklist.
        at com.alipay.sofa.rpc.codec.bolt.SofaRpcSerialization.serializeContent(SofaRpcSerialization.java:473)
        at com.alipay.sofa.rpc.codec.AlipaySofaRpcSerialization.serializeContent(AlipaySofaRpcSerialization.java:247)
        at com.alipay.remoting.rpc.protocol.RpcResponseCommand.serializeContent(RpcResponseCommand.java:109)
        at com.alipay.remoting.rpc.RpcCommand.serialize(RpcCommand.java:95)
        at com.alipay.remoting.rpc.protocol.RpcRequestProcessor.sendResponseIfNecessary(RpcRequestProcessor.java:150)
        at com.alipay.remoting.rpc.protocol.RpcAsyncContext.sendResponse(RpcAsyncContext.java:55)
        at com.alipay.sofa.rpc.server.bolt.BoltServerProcessor.handleRequest(BoltServerProcessor.java:193)
        at com.alipay.sofa.rpc.server.bolt.BoltServerProcessor.handleRequest(BoltServerProcessor.java:62)
        at com.alipay.remoting.rpc.protocol.RpcRequestProcessor.dispatchToUserProcessor(RpcRequestProcessor.java:206)
        at com.alipay.remoting.rpc.protocol.RpcRequestProcessor.doProcess(RpcRequestProcessor.java:134)
        at com.alipay.remoting.rpc.protocol.RpcRequestProcessor$ProcessTask.run(RpcRequestProcessor.java:359)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.io.IOException: Class org.springframework.beans.factory.NoSuchBeanDefinitionException is in blacklist.
        at com.alipay.hessian.NameBlackListFilter.resolve(NameBlackListFilter.java:98)
        at com.alipay.hessian.ClassNameResolver.resolve(ClassNameResolver.java:98)
        at com.caucho.hessian.io.Hessian2Output.writeObject(Hessian2Output.java:493)
        at com.caucho.hessian.io.JavaSerializer$FieldSerializer.serialize(JavaSerializer.java:251)
        at com.caucho.hessian.io.JavaSerializer.writeInstance(JavaSerializer.java:208)
        at com.caucho.hessian.io.JavaSerializer.writeObject(JavaSerializer.java:172)
        at com.caucho.hessian.io.Hessian2Output.writeObject(Hessian2Output.java:498)
        at com.alipay.sofa.rpc.codec.bolt.SofaRpcSerialization.serializeContent(SofaRpcSerialization.java:466)
        ... 13 common frames omitted
```

错误原因：

客户端的请求参数或者服务端的响应中包含序列化黑名单中的类型。RPC 黑名单列表存放在 `sofa-rpc-all` JAR 包下的 `sofa-rpc/serialize_blacklist.txt` 。



解决方法：

修复代码，确保在 RPC 调用过程中不要包含黑名单里面的类型。

DeserializationException: java.lang.ClassNotFoundException

错误日志样例：

```
2019-06-27 00:33:05,745 ERROR [RpcRequestProcessor#267] [SOFA-SEV-BOLT-BIZ-12200-3-T11] Des
erializationException occurred when process in RpcRequestProcessor, id=1, deserializeLevel=
DESERIALIZE_ALL com.alipay.remoting.exception.DeserializationException: java.lang.ClassNotF
oundException: com.alipay.share.rpc.facade.Person at com.alipay.sofa.rpc.codec.bolt.SofaRpc
Serialization.deserializeContent(SofaRpcSerialization.java:278) at com.alipay.remoting.rpc.
protocol.RpcRequestCommand.deserializeContent(RpcRequestCommand.java:144) at com.alipay.rem
oting.rpc.RpcCommand.deserialize(RpcCommand.java:117) at com.alipay.remoting.rpc.RpcCommand
.deserialize(RpcCommand.java:138) at com.alipay.remoting.rpc.protocol.RpcRequestProcessor.d
eserializeRequestCommand(RpcRequestProcessor.java:264) at com.alipay.remoting.rpc.protocol.
RpcRequestProcessor.doProcess(RpcRequestProcessor.java:142) at com.alipay.remoting.rpc.prot
ocol.RpcRequestProcessor$ProcessTask.run(RpcRequestProcessor.java:371) at java.util.concurr
ent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) at java.util.concurrent.Thre
adPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) at java.lang.Thread.run(Thread.java:
748) Caused by: com.alipay.sofa.rpc.core.exception.SofaRpcRuntimeException: java.lang.Class
NotFoundException: com.alipay.share.rpc.facade.Person at com.alipay.sofa.rpc.common.utils.C
lassUtils.forName(ClassUtils.java:59) at com.alipay.sofa.rpc.common.utils.ClassUtils.forNam
e(ClassUtils.java:45) at com.alipay.sofa.rpc.common.utils.ClassTypeUtils.getClass(ClassType
Utils.java:87) at com.alipay.sofa.rpc.common.utils.ClassTypeUtils.getClasses(ClassTypeUtils
.java:52) at com.alipay.sofa.rpc.codec.sofahessian.serialize.SofaRequestHessianSerializer.d
ecodeObjectByTemplate(SofaRequestHessianSerializer.java:70) at com.alipay.sofa.rpc.codec.so
fahessian.serialize.SofaRequestHessianSerializer.decodeObjectByTemplate(SofaRequestHessianS
erializer.java:38) at com.alipay.sofa.rpc.codec.sofahessian.SofaHessianSerializer.decode(So
faHessianSerializer.java:164) at com.alipay.sofa.rpc.codec.bolt.SofaRpcSerialization.deseri
alizeContent(SofaRpcSerialization.java:269) ... 9 common frames omitted Caused by: java.lan
g.ClassNotFoundException: com.alipay.share.rpc.facade.Person at java.net.URLClassLoader.fin
dClass(URLClassLoader.java:381) at java.lang.ClassLoader.loadClass(ClassLoader.java:424) at
sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:335) at java.lang.ClassLoader.load
Class(ClassLoader.java:357) at java.lang.Class.forName0(Native Method) at java.lang.Class.f
orName(Class.java:348) at com.alipay.sofa.rpc.common.utils.ClassUtils.forName(ClassUtils.ja
va:57) ... 16 common frames omitted
```

错误原因：

RPC 客户端调用服务时，请求参数中包含某个类型，但是这个类型在服务端不存在。

解决方法：

业务方定位是客户端传参错误还是服务端类型缺失导致，然后再针对错误原因进行修复。

DeserializationException: {xxx类}: expected xxx at xxx

错误日志样例：

```
2019-06-26 21:18:53,193 ERROR [RpcRequestProcessor#267] [SofaBizProcessor-12200-2-T2] Deser
ializationException occurred when process in RpcRequestProcessor, id=1, deserializeLevel=DE
SERIALIZE_ALL com.alipay.remoting.exception.DeserializationException: com.alipay.share.rpc.
facade.Person.status: expected integer at 0x4 java.lang.String (dead) at com.alipay.sofa.rp
c.codec.bolt.SofaRpcSerialization.deserializeContent(SofaRpcSerialization.java:352) at com.
alipay.remoting.rpc.protocol.RpcRequestCommand.deserializeContent(RpcRequestCommand.java:14
4) at com.alipay.remoting.rpc.RpcCommand.deserialize(RpcCommand.java:117) at com.alipay.rem
oting.rpc.RpcCommand.deserialize(RpcCommand.java:138) at com.alipay.remoting.rpc.protocol.R
pcRequestProcessor.deserializeRequestCommand(RpcRequestProcessor.java:264) at com.alipay.re
moting.rpc.protocol.RpcRequestProcessor.doProcess(RpcRequestProcessor.java:142) at com.alip
ay.remoting.rpc.protocol.RpcRequestProcessor$ProcessTask.run(RpcRequestProcessor.java:371)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) at java.
util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) at java.lang.Thr
ead.run(Thread.java:748) Caused by: com.caucho.hessian.io.HessianFieldException: com.alipay
.share.rpc.facade.Person.status: expected integer at 0x4 java.lang.String (dead) at com.cau
cho.hessian.io.JavaDeserializer.logDeserializeError(JavaDeserializer.java:620) at com.cauch
o.hessian.io.JavaDeserializer$IntFieldDeserializer.deserialize(JavaDeserializer.java:513) a
t com.caucho.hessian.io.JavaDeserializer.readObject(JavaDeserializer.java:253) at com.cauch
o.hessian.io.JavaDeserializer.readObject(JavaDeserializer.java:177) at com.caucho.hessian.i
o.Hessian2Input.readObjectInstance(Hessian2Input.java:2746) at com.caucho.hessian.io.Hessia
n2Input.readObject(Hessian2Input.java:2242) at com.caucho.hessian.io.Hessian2Input.readObje
ct(Hessian2Input.java:2230) at com.alipay.sofa.rpc.codec.bolt.SofaRpcSerialization.deserial
izeContent(SofaRpcSerialization.java:339) ... 9 common frames omitted Caused by: com.caucho
.hessian.io.HessianProtocolException: expected integer at 0x4 java.lang.String (dead) at co
m.caucho.hessian.io.Hessian2Input.error(Hessian2Input.java:3435) at com.caucho.hessian.io.H
essian2Input.expect(Hessian2Input.java:3406) at com.caucho.hessian.io.Hessian2Input.readInt
(Hessian2Input.java:1014) at com.caucho.hessian.io.JavaDeserializer$IntFieldDeserializer.de
serialize(JavaDeserializer.java:509) ... 15 common frames omitted
```

错误原因：

客户端调用 RPC 服务时，传入的参数中定义的字段类型和服务端不一致。例如客户端传入一个 `com.alipay.share.rpc.facade.Person` 类型的参数，Person 类型中的字段 status 定义为 String 类型，但是服务端引用的 Person 类中 status 定义为 int 类型。

解决方法：

业务方修复代码，使客户端和服务端引用的参数类型中所定义的字段类型一致。

not found Serializer，type：[1]

错误日志样例：

```
com.alipay.sofa.rpc.core.exception.SofaRpcException: com.alipay.remoting.exception.Serializ
ationException: RPC-020050008: 未找到 Serializer,type:[1]. at com.alipay.sofa.rpc.transport.
bolt.BoltClientTransport.convertToRpcException(BoltClientTransport.java:340) at com.alipay.
sofa.rpc.transport.bolt.BoltClientTransport.syncSend(BoltClientTransport.java:255) at com.a
lipay.sofa.rpc.client.AbstractCluster.doSendMsg(AbstractCluster.java:581) at com.alipay.sof
a.rpc.client.AbstractCluster.sendMsg(AbstractCluster.java:552) at com.alipay.sofa.rpc.filte
r.ConsumerInvoker.invoke(ConsumerInvoker.java:60) at com.alipay.sofa.rpc.filter.ConsumerMes
hFilter.invoke(ConsumerMeshFilter.java:42) at com.alipay.sofa.rpc.filter.FilterInvoker.invo
ke(FilterInvoker.java:100) at com.alipay.sofa.rpc.filter.ConsumerIdentityFilter.invoke(Cons
umerIdentityFilter.java:53) at com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoke
r.java:100) at com.alipay.sofa.rpc.servcegovern.downgrade.DowngradeFilter.invoke(DowngradeF
ilter.java:82) at com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoker.java:100) a
t com.alipay.sofa.rpc.filter.ConsumerProfileFilter.invoke(ConsumerProfileFilter.java:60) at
com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoker.java:100) at com.alipay.sofa.
rpc.filter.EnterpriseConsumerTracerFilter.invoke(EnterpriseConsumerTracerFilter.java:86) at
com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoker.java:100) ..... at org.apach
e.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61) at java.lang.Thre
ad.run(Thread.java:745) Caused by: com.alipay.remoting.exception.SerializationException: RP
C-020050008: 未找到 Serializer,type:[1]. at com.alipay.sofa.rpc.codec.bolt.SofaRpcSerializat
ion.serializeContent(SofaRpcSerialization.java:198) at com.alipay.remoting.rpc.protocol.Rpc
RequestCommand.serializeContent(RpcRequestCommand.java:124) at com.alipay.remoting.rpc.RpcC
ommand.serialize(RpcCommand.java:105) at com.alipay.remoting.rpc.RpcRemoting.toRemotingComm
and(RpcRemoting.java:354) at com.alipay.remoting.rpc.RpcRemoting.invokeSync(RpcRemoting.jav
a:179) at com.alipay.remoting.rpc.RpcClientRemoting.invokeSync(RpcClientRemoting.java:64) a
t com.alipay.remoting.rpc.RpcClient.invokeSync(RpcClient.java:355) at com.alipay.sofa.rpc.t
ransport.bolt.BoltClientTransport.doInvokeSync(BoltClientTransport.java:279) at com.alipay.
sofa.rpc.transport.bolt.EnterpriseBoltClientTransport.doInvokeSync(EnterpriseBoltClientTran
sport.java:97) at com.alipay.sofa.rpc.transport.bolt.BoltClientTransport.syncSend(BoltClien
tTransport.java:252) ... 89 more Caused by: com.alipay.sofa.rpc.core.exception.SofaRpcRunti
meException: RPC-020050008: 未找到 Serializer,type:[1]. at com.alipay.sofa.rpc.codec.Seriali
zerFactory.getSerializer(SerializerFactory.java:87) at com.alipay.sofa.rpc.codec.bolt.SofaR
pcSerialization.serializeContent(SofaRpcSerialization.java:193) ... 98 more
```

错误原因：

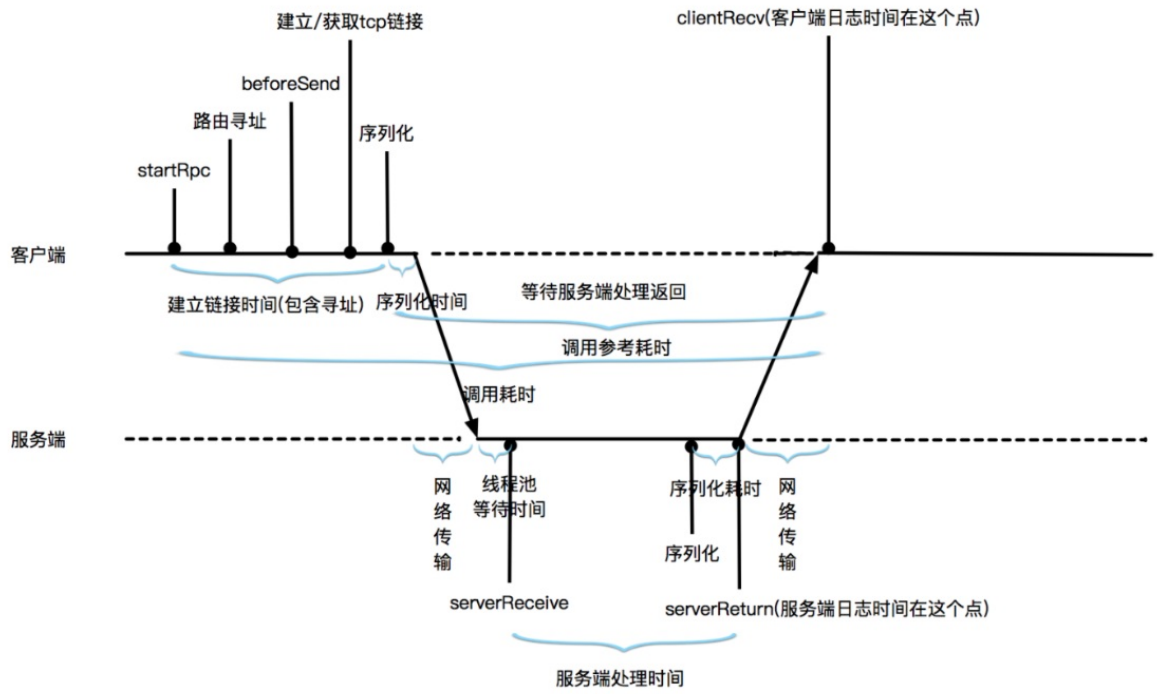
客户项目中依赖的 hessian 和 SOFA 中的 hessian JAR 包冲突。

解决方法：

去掉原有项目中的 hessian JAR 包，以 SOFA 中的包为准。

1.4.2.4.3. 调用 RPC 超时

RPC调用时序图



问题现象

调用 RPC 服务超时，客户端错误日志 `logs/tracelog/middleware_error.log` 中出现如下异常信息：

```
2018-02-28 19:38:03.080,sofa2-rpc-client,79297e491[redacted]9212220438,0,main,timeout_error,rpc,invokeType=sync&uid
&protocol=bolt&targetApp=sofa2-rpc-server&targetIdc=&targetCity=&paramTypes=&methodName=message&serviceName=com.alip
y.share.rpc.facade.SampleService:1.0&targetUrl=10.[redacted]6:12200&targetZone=&,,com.alipay.sofa.rpc.core.exception.Sof
TimeOutException: com.alipay.remoting.rpc.exception.InvokeTimeoutException: Rpc invocation timeout[responseCommand T
MEOUT]! the address is 10.[redacted]:12200
```

排查思路

PRC 调用超时一般可以按照如下几个方面进行排查：

- 服务本身业务超时，如业务代码处理时间过长。

  RPC 在默认情况下超时时间为 3 秒，您可以登录服务端查看 `logs/tracelog/rpc-server-digest.log` 日志，并根据客户端超时日志中的 traceid 查找服务端处理对应请求的日志，确定请求处理的具体时间。

  ```
  [root@iZbp18psc0dmfi6couttpzZ tracelog]# cat rpc-server-digest.log | grep 7929[redacted]2220438
  2018-02-28 19:38:04.085,sofa2-rpc-server,7929[redacted]212220438,0,com.alipay.share.rpc.facade.SampleService:1
  .0,message,bolt,,10.[redacted]44,sofa2-rpc-client,,4000ms,0ms,SofaBizProcessor-12200-0-T4,02,,,1ms,,
  ```

  如果处理时间大于 3 秒或非常接近 3 秒，建议排查服务端本身异常，例如：

  - 服务端业务代码执行慢。
  - 服务端本身有外网服务调用，或者服务端又调用了其他 RPC 服务（client->RPC Server A->RPC Server B），此种情况需要分别排查 A 和 B，定位问题。
  - 服务端有数据库操作，如数据库连接耗时，慢 SQL 等。

- 服务端 RPC 线程池耗尽。

  登录服务端查看 `rpc/bolt-threadpool` 日志，如果发生 RPC 线程池队列阻塞，先确认是否发生超时的时间段有业务请求高峰，或者用 jstack 查看业务线程是否有等待或者死锁情况，导致 RPC 线程耗尽。

RPC 线程池大小、队列长度配置说明请参见 应用维度配置。

- 发生 GC，导致线程停止。

  某些 GC 类型会触发 stop the world，将所有线程挂起。若要排查是否是 GC 导致的问题，可以通过如下方式开启 GC 日志：

  - 在 `config/java_opts` 文件加入启动参数

    ```
    -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:/home/admin/logs/gc.log
    ```

    配置完成后，需要重新打包发布。

  - 用 `kill -15` 命令结束服务端进程，然后手动启动 RPC 服务

    运行 `su admin` 登录 admin 用户，然后用 nohup 形式启动 RPC 服务：

    ```
    nohup java -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:/home/admin/logs/gc.log -Drpc_bind_network_interface=eth0 -Dspring.profiles.active=LuXuan -jar /home/admin/app-run/sofa2-rpcserver-service-1.0-SNAPSHOT-executable.jar &
    ```

  等待下次 RPC 超时发生后，查看 `gc.log` 验证超时的时间段是否有耗时较长的 GC，尤其是 Full GC。

- 发生严重网络问题。

  您可以通过如下方式排查网络问题：

  - 在客户端和服务端运行 `tsar -i 1` 命令查看问题发生的时间点是否有网络重传。
  - 在客户端和服务端同时部署 tcpdump 进行循环抓包，问题发生后分析网络包。
  - 在客户端和服务端运行 `ping` 命令观察是否存在网络延时。
  - 查看客户端和服务端的 TCP Send-Q 和 Recv-Q 是否有积压。

  > **说明**
  >
  > `rpc-client-digest.log` 中的日志打印时间减去调用耗时，可以得到请求发起的时间点；`rpc-server-digest.log` 中的日志打印时间减去执行时间，可以得到服务端开始处理请求的时间点。

- 其他外部因素影响服务器性能，如定时任务、批处理或者与宿主机上其他 VM、Container 发生资源争抢。

1.4.2.4.4. 无法获取服务地址

问题现象

RPC 客户端调用服务时出现 RPC-02306 的报错。

> **说明**
>
> RPC-02306：没有获得服务 [{0}] 的调用地址，请检查服务是否已经推送。

排查思路

您可以从以下几个方面进行排查：

1. 检查服务地址是否推送。

   登录客户端，查看日志 `/home/admin/logs/rpc/rpc-registry.log` 。您可以通过服务接口名过滤日志，找到最后一次推送记录：

   - 如果发现服务端地址没有推送到客户端，建议排查服务是否注册成功。
   - 如果显示服务地址已经推送，但是 RPC-02306 错误发生的时间在服务地址推送之前，这种情况多发生在业务系统自己通过定时任务，或在 bean 初始化完成后就开始调用服务，而此时客户端应用还没启动完成。您可以通过 `address-wait-time` 配置解决，详情请参见 Bolt 引用详细配置。

2. 检查服务是否注册成功。

   登录中间件控制台查看服务注册是否成功，也可以登录服务端查看日志 `/home/admin/logs/registry/registry-client.log` 。如果有服务发布相关的错误，请根据日志信息进一步排查。

3. 检查服务注册中心连接。

   通过以下命令检查客户端和服务端与服务注册中心的连接情况：

   ```
   netstat -anp |grep 9600
   ```

   9600 端口是服务注册中心的监听端口，客户端和服务端与 9600 端口建立长连接，向服务注册中心发布和订阅服务。如果客户端或者服务端与 9600 端口的连接断开，需要重启应用恢复，并进一步排查端口异常断开的原因。

4. 检查服务端和客户端配置信息。

   分别打开服务端和客户端应用的配置文件 `application.properties` ，查看以下参数是否配置相同：

   - com.alipay.env
   - com.alipay.instanceid
   - com.antcloud.antvip.endpoint

5. 检查客户端启动时是否收到 RPC Config 推送。

   确定最近一次 RPC 客户端的启动时间，查看日志 `/home/admin/logs/rpc/rpc-registry.log` ，根据客户端上次启动时间和服务接口名过滤日志，检查对应的接口是否有 "Receive RPC config info" 的记录：

   ```
   Receive RPC config info: service[com.alipay.share.rpc.facade.SampleService:1.0@DEFAULT]
   usable config info[0]   // 0 表示正常。
   ```

> 说明
>
> RPC Config 是 Registry-Session 调用 Registry-Meta 时获取，当 RPC 客户端启动或控制台服务管控发生变更（如权重的调整等）时会推送一次。

如果没有记录，会导致后续无法调用服务，可以考虑重启 RPC 客户端。如果重启客户端也无法生效，可能是因为 SessionServer 无法从 MetaServer 获取 RPC 配置，您可以排查以下几个方面：

i. 登录客户端 9600 端口连接的 SessionServer，根据 RPC 客户端重启时间查看 `common-error.log` 日志中是否有调用 MetaServer 的相关错误。

ii. 登录所有 MetaSever，查看所有节点的进程是否存在。

iii. 查看 MetaServer 的 `common-error.log` 日志是否有 OOM 或者与 MetaServer 主节点 RPC 调用超时等错误。

iv. 执行 MetaServer 的健康检查命令，确认服务状态和选举是否正常。正常情况下会有一台是主节点（leader）角色。

```
curl http://localhost:8080/health/readiness
```

6. 检查RPC服务端地址绑定。

   登录RPC服务端，运行以下命令：

   ```
   ps -ef|grep java
   ```

   查看进程启动参数 `rpc_bind_network_interface` 或 `rpc_enabled_ip_range` 是否绑定了正确的IP地址。配置方式请参见 应用维度配置。

7. 检查 RPC 服务端连接情况 。

   查看 RPC 客户端的 bolt 日志 `/logs/bolt/connection-event.log`，检查问题发生期间 RPC 客户端与 RPC 服务端（根据服务端 IP 过滤）是否有断连和重连事件发生。若一直有断连和重连现象（一般建连成功 3 秒后断开连接，断开后立刻重新连接），则可能是 SOFABoot 旧版本的问题，建议开源版升级至 2.6.2 及以上版本；商业版升级至 3.1.0 及以上版本。

1.4.2.4.5. 引用了 4.1.23.Final 版本 netty-all 导致应用占用 CPU 高

问题现象

应用 CPU 占用长时间达到 100%。

问题原因

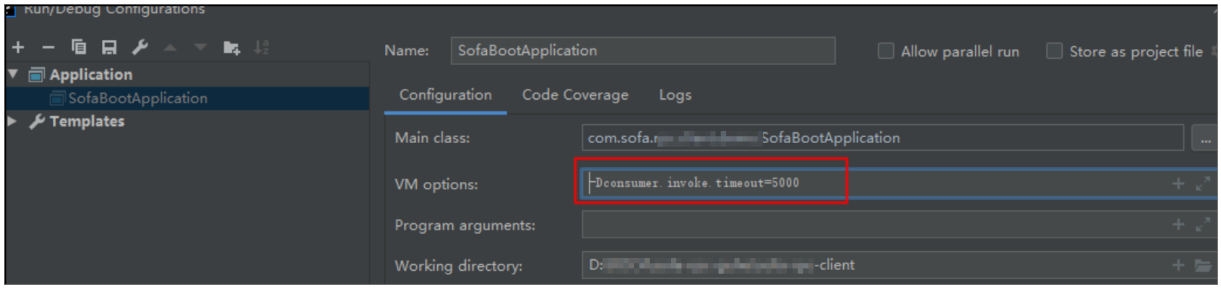
应用单独引用了 4.1.23.Final 版本 netty-all ，此版本存在 CPU 占用高的问题。

解决方案

删除 4.1.23.Final 版本 netty-all 的引用，SOFA RPC 3.4.2 版本引用了 4.1.45.Final 版本的 netty-all，已修复了该问题。

1.4.2.4.6. 是否可以以工程为单位调整 RPC 调用超时时间

您可以在启动参数中增加 `-Dconsumer.invoke.timeout= <超时时间>` 覆盖默认值。



1.4.2.4.7. 非 Spring 环境调用 RPC 时 spanid 丢失

问题现象

通过 Java 编程界面（非 spring 及 sofaboot）直接调用 RPC 服务，SOFATracer 日志打印 spanid 为 0。

问题原因

SOFATracer 中的 SpanId 代表本次调用在整个调用链路树中的位置，用户 demo 调用 RPC 服务时为直接调用，当前位置为 0，所有日志打印 spanid 为 0。

解决方案

无须解决。直接调用 RPC 时在链路第一层，所以显示为 0，若本次调用在整个调用链路树中的位置为第二层，如通过 REST 调用 RPC 服务，则 provider 的日志 `rpc-server-digest.log` 以及 consumer 的 `rpc-client-digest.log` 中会显示 spanid 为 0.1。

1.4.3. 问题排查案例

1.4.3.1. 在微服务管控页面设置权重不生效

问题现象

微服务管控页面配置了权重，并禁用了服务，页面显示已修改，但订阅端未接到推送，实际未生效。

问题原因

客户部署为双机房同库模式，应用机房收敛，应用均部署在主机房，但 DsrConsole 推送配置时推送到了备机房，所以主机房应用未接到推送的配置。

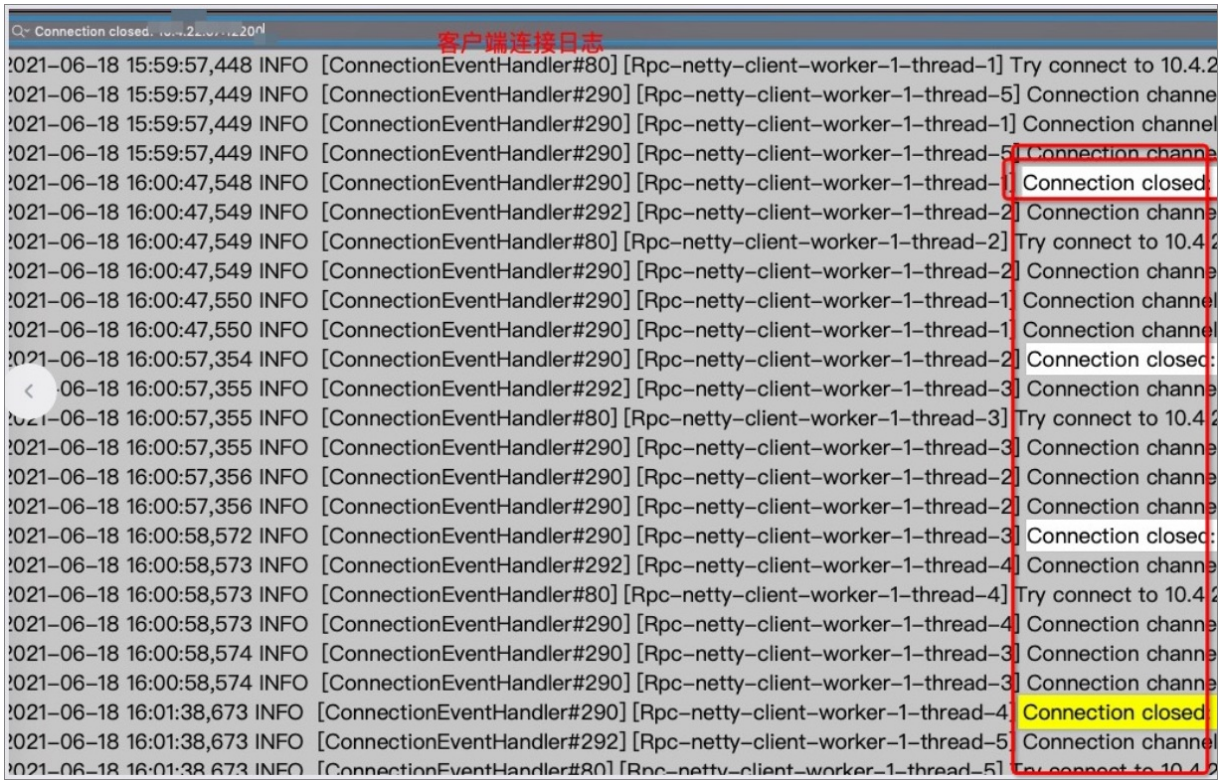
解决方案

在云游 Local 中，将主备机房的微服务应用启动参数的 `registry.meta.server` 配置都改为主备机房所有的 IP，然后重启。

1.4.3.2. SOFABoot 低版本 bolt 存在连接频繁建连、断连

问题现象

consumer 和 provider 的 `home/admin/logs/bolt/connection-event.log` 日志中存在大量建连、断连的日志：



问题原因

SOFA PRC 错误使用了 bolt API，导致无法更新 Connection。

解决方案

将 SOFABoot 2.x 版本升级到 SOFABoot 2.6.5 及以上版本。

1.4.3.3. 更改配置 endpoints.enabled=false 导致健康检查失败

问题现象

发布应用健康检查失败，执行健康检查命令不通过。

问题背景

因业务需要，客户希望关闭以下 endpoints，仅开启 `http://xxx.xxx.xxx.xxx/health/readiness` ：

```
http://xxx.xxx.xxx.xxx/mappings
http://xxx.xxx.xxx.xxx/metrics
http://xxx.xxx.xxx.xxx/configprops
http://xxx.xxx.xxx.xxx/autoconfig
http://xxx.xxx.xxx.xxx/dump
http://xxx.xxx.xxx.xxx/trace
```

将 endpoints 设置为 false（ `endpoints.enable=false` ）后，发现 `http://xxx.xxx.xxx.xxx/health/readiness` 已无法访问。但部署服务后，安全检查失败，服务可以正常启动。

解决方案

按如下方式关闭其他 endpoints：

```
endpoints.metrics.enabled=false
endpoints.heapdump.enabled=false
endpoints.env.enabled=false
endpoints.actuator.enabled=false
endpoints.beans.enabled=false
endpoints.configprops.enabled=false
endpoints.dump.enabled=false
endpoints.info.enabled=false
endpoints.logfile.enabled=false
endpoints.trace.enabled=false
endpoints.mappings.enabled=false
endpoints.autoconfig.enabled=false
```

1.4.3.4. 慢 SQL 引起的 RPC 调用超时

问题描述

在页面上点击查询后，很长时间没返回数据，查看发现后端日志中有 RPC 超时日志。

排查步骤

1. 查看出问题时的服务端节点，未发现负载过高的情况。
2. 排查 RPC 调用链路。

   app-server 为调用方、rpc-sysmanager 为服务方、跟踪 traceid 为 0a827ec516145645762433729165 83。查看日志如下：

   - RPC 发起方错误日志 `middleware_error.log`

     日志显示显示调用超时错误。

   - RPC 客户端摘要日志 `rpc-client-digest.log`

     日志显示发起调用 10s 后超时返回。

   - RPC 服务端摘要日志 `rpc-server-digest.log`

     日志显示 RPC 服务执行时间为 48 多秒，序列化时间为 10 多秒。序列化时间 10 多秒可能是结果太大导致。继续追查，在日志中发现有慢 SQL 日志。

3. 打印慢 SQL 日志。

查看发现对应 traceid 的记录耗时为 9 秒多，SQL 语句中有大量的子查询。可以得出结论：

- 用户的 SQL 复杂，结果集太大导致 RPC 耗时过长。
- 业务代码中有复杂数据库操作大量子查询连接查询等，结果集过大导致序列化时间太长。

解决方案

优化 SQL 性能，减少子查询。结果集尽量小分页查询，提前做好 explain。

1.4.3.5. 系统 OOM 导致微服务节点从注册中心掉线

问题现象

微服务应用其中一个节点从注册中心掉线，重启后恢复。

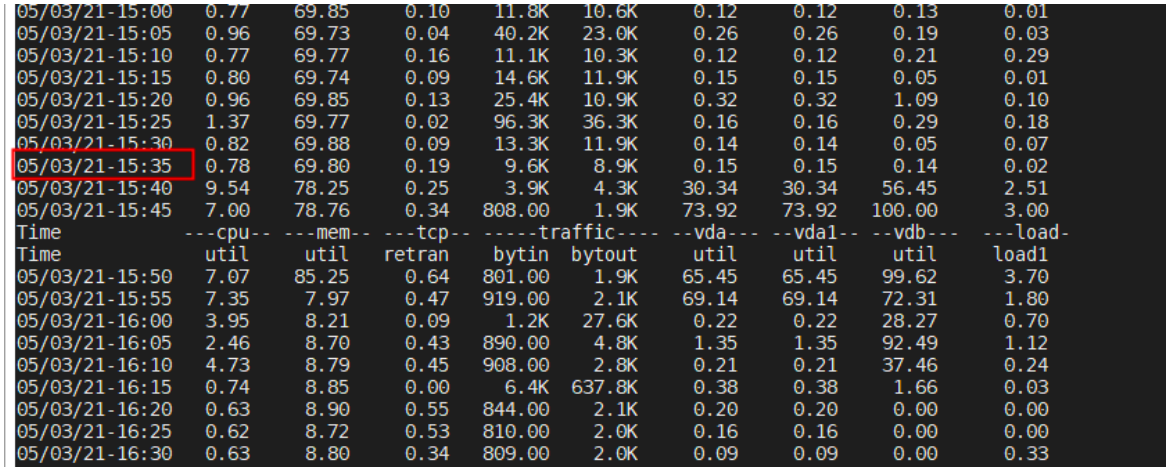
排查步骤

1. 查看掉线节点日志。

   仅发现 `bolt-threadpool.log` 还有掉线前的日志，其他日志都均为重启后的日志。

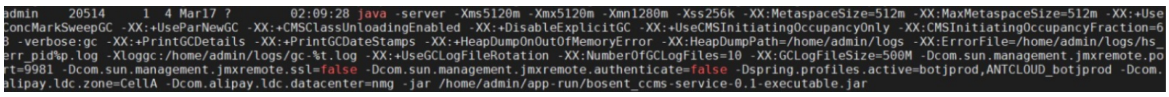
   `bolt-threadpool.log` 线程池日志最后打印时间与应用节点的掉线时间吻合，且无其他掉线前的日志。初步推测应用进程在掉线时就没了，可能的原因是系统 OOM。

2. 查看当前内存监控情况。



Time	cpu	mem	tcp	traffic		vda	vda1	vdb	load
Time	util	util	retran	bytin	bytout	util	util	util	load1
05/03/21-15:00	0.77	69.85	0.10	11.8K	10.6K	0.12	0.12	0.13	0.01
05/03/21-15:05	0.96	69.73	0.04	40.2K	23.0K	0.26	0.26	0.19	0.03
05/03/21-15:10	0.77	69.77	0.16	11.1K	10.3K	0.12	0.12	0.21	0.29
05/03/21-15:15	0.80	69.74	0.09	14.6K	11.9K	0.15	0.15	0.05	0.01
05/03/21-15:20	0.96	69.85	0.13	25.4K	10.9K	0.32	0.32	1.09	0.10
05/03/21-15:25	1.37	69.77	0.02	96.3K	36.3K	0.16	0.16	0.29	0.18
05/03/21-15:30	0.82	69.88	0.09	13.3K	11.9K	0.14	0.14	0.05	0.07
05/03/21-15:35	0.78	69.80	0.19	9.6K	8.9K	0.15	0.15	0.14	0.02
05/03/21-15:40	9.54	78.25	0.25	3.9K	4.3K	30.34	30.34	56.45	2.51
05/03/21-15:45	7.00	78.76	0.34	808.00	1.9K	73.92	73.92	100.00	3.00
05/03/21-15:50	7.07	85.25	0.64	801.00	1.9K	65.45	65.45	99.62	3.70
05/03/21-15:55	7.35	7.97	0.47	919.00	2.1K	69.14	69.14	72.31	1.80
05/03/21-16:00	3.95	8.21	0.09	1.2K	27.6K	0.22	0.22	28.27	0.70
05/03/21-16:05	2.46	8.70	0.43	890.00	4.8K	1.35	1.35	92.49	1.12
05/03/21-16:10	4.73	8.79	0.45	908.00	2.8K	0.21	0.21	37.46	0.24
05/03/21-16:15	0.74	8.85	0.00	6.4K	637.8K	0.38	0.38	1.66	0.03
05/03/21-16:20	0.63	8.90	0.55	844.00	2.1K	0.20	0.20	0.00	0.00
05/03/21-16:25	0.62	8.72	0.53	810.00	2.0K	0.16	0.16	0.00	0.00
05/03/21-16:30	0.63	8.80	0.34	809.00	2.0K	0.09	0.09	0.00	0.33

监控每 5 分钟记录一次，15:40 时内存占用为 78%、容器配置为 4C8G，即内存占用 6 GB 多。查看 Java 进程的 JVM 参数，Xmx 配置为 5 GB，并且有配置 OOM 时自动存储内存 dump，但容器中并未发现 dump 文件。



3. 查看掉线时的系统日志。

```
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [22610]  1102 22610    36923     8281      28        0             0 grep
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [27817]  1102 27817    28074       42      12        0             0 less
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [ 5058]     0  5058    38117      361      76        0             0 sshd
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [ 5060]  1102  5060    38117      359      73        0             0 sshd
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [ 5061]  1102  5061    29391      129      12        0             0 bash
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [10771]  1102 10771    28755       97      12        0             0 grep
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [10935]  1102 10935    28803      127      14        0             0 grep
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [11073]  1102 11073    28232      162      10        0             0 less
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [11130]  1102 11130    28232      162      11        0             0 less
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [11207]  1100 11207    28793       88      12        0             0 diver
-monitor.s
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [11274]  1102 11274   625637   587906    1174        0             0 vim
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: [11424]  1100 11424      595        1       3        0             0 date
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: Out of memory: Kill process 26422 (java) score 618 or sacrifice child
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz kernel: Killed process 26422 (java) total-vm:9466256kB, anon-rss:4945428kB, file-r
ss:0kB, shmem-rss:0kB
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz systemd: Removed slice user-0.slice.
Mar  5 15:37:51 iz6lu059yko96v7ievdxxnz systemd: Stopping user-0.slice.
```

查看发现 anon-rss 占用大约 4.7 GB 内存，未到 Xmx 配置的 5 GB，所以没有自动保存 dump。应该是其他进程需要申请系统内存，触发系统 OOM，导致 Java 进程被 kill。

用户出现问题的原因是应用所在节点容器掉线前内存突增，导致 Java 进程被系统 OOM kill掉了，JVM 未 OOM 无法生成 dump 文件。

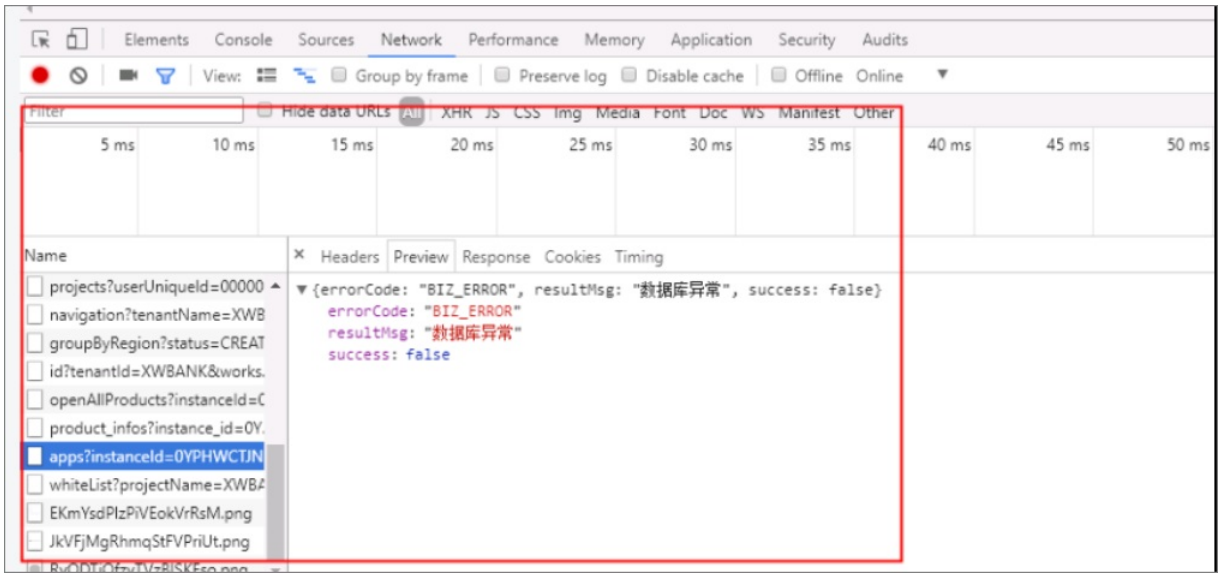
解决方案

重启掉线的节点，并找出其他占用内存高的进程，根据实际需求决定是否扩容。

1.4.3.6. 打开应用依赖报错

问题现象

控制台打开应用依赖出现“数据库异常”报错。



排查步骤

1. 查看打开应用依赖这个操作时 dsrconsole 的 错误日志 `common-error.log` 。

在日志中发现 SQL 执行超时错误：

```
common-error.log
2020-06-04 13:48:46.593 [SOFA-REST-WORK-3-62] ERROR com.alibaba.druid.pool.DruidDataSource - discard connection
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure
The last packet successfully received from the server was 13,284 milliseconds ago.  The last packet sent successfully to the server was 5,004 milliseconds ago.
Caused by: java.net.SocketTimeoutException: Read timed out

The last packet successfully received from the server was 13,284 milliseconds ago.  The last packet sent successfully to the server was 5,004 milliseconds ago.
### The error may exist in URL [jar:file:/home/admin/app-run/dsrconsole-endpoint-1.0.0-executable.jar!/BOOT-INF/lib/dsrconsole-common-dal-1.0.0.jar!/META-INF/dsrconsole/mybatis/RelationMapper.xml]
### The error may involve com.alipay.antcloud.dsrconsole.common.dal.dao.RelationDAO.findRelationsByInstanceIdAndAppName-Inline
### The error occurred while setting parameters
### SQL: select p.app_name as source, s.app_name as target, count(1) as value from service_pub_info as p       left join service_sub_info as s       on p.data_id = s.data_id and p.instance_id = s.instance_id       where p.instance_id
= ?             group by p.app_name, s.app_name
### Cause: com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure

The last packet successfully received from the server was 13,284 milliseconds ago.  The last packet sent successfully to the server was 5,004 milliseconds ago.
; SQL []; Communications link failure

The last packet successfully received from the server was 13,284 milliseconds ago.  The last packet sent successfully to the server was 5,004 milliseconds ago.; nested exception is
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure
```

推测可能是 DB 性能有问题。

2. 查看 DB 性能监控。

   性能监控中发现了上述慢 SQL 日志：

```
      group by p.app_name, s.app_name;
# Time: 2020-06-04T13:49:00.532578+08:00
# User@Host: code_alb[code_alb] @  [1      74]  Id: 42     12
# Query_time: 7.892412  Lock_time: 0.000115 Rows_sent: 413  Rows_examined: 22163
SET timestamp=1591249740;
select p.app_name as source, s.app_name as target, count(1) as value from service_pub_info as p
      left join service_sub_info as s
      on p.data_id = s.data_id and p.instance_id = s.instance_id
      where p.instance_id = 'Z8V      'QHV'

      group by p.app_name, s.app_name;
# Time: 2020-06-04T13:49:30.980446+08:00
# User@Host: code_alb[code_alb] @  [10     4]  Id: 42     63
# Query_time: 11.606935  Lock_time: 0.000156 Rows_sent: 485  Rows_examined: 22308
SET timestamp=1591249770;
select p.app_name as source, s.app_name as target, count(1) as value from service_pub_info as p
      left join service_sub_info as s
      on p.data_id = s.data_id and p.instance_id = s.instance_id
      where p.instance_id = 'OYH      NEYP'
```

3. 查看 DB 的索引以及这条 SQL 的 explain。

```
MySQL [dsrconsole]> show index from service_sub_info;
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table           | Non_unique | Key_name              | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| service_sub_info |          0 | PRIMARY               |            1 | id          | A         |       16869 |     NULL | NULL   |      | BTREE      |         |               |
| service_sub_info |          0 | uniq_check_sum        |            1 | check_sum   | A         |       15664 |     NULL | NULL   | YES  | BTREE      |         |               |
| service_sub_info |          1 | idx_processid_alive   |            1 | process_id  | A         |        1539 |     NULL | NULL   |      | BTREE      |         |               |
| service_sub_info |          1 | idx_processid_alive   |            2 | alive       | A         |        1539 |     NULL | NULL   |      | BTREE      |         |               |
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
4 rows in set (0.00 sec)

MySQL [dsrconsole]> show index from service_pub_info;
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table           | Non_unique | Key_name              | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| service_pub_info |          0 | PRIMARY               |            1 | id          | A         |        5460 |     NULL | NULL   |      | BTREE      |         |               |
| service_pub_info |          0 | uniq_check_sum        |            1 | check_sum   | A         |        5588 |     NULL | NULL   | YES  | BTREE      |         |               |
| service_pub_info |          1 | idx_processid_alive   |            1 | process_id  | A         |         823 |     NULL | NULL   |      | BTREE      |         |               |
| service_pub_info |          1 | idx_processid_alive   |            2 | alive       | A         |         823 |     NULL | NULL   |      | BTREE      |         |               |
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
4 rows in set (0.01 sec)
```

```
MySQL [dsrconsole]> explain select p.app_name as source,s.app_name as target,count(1) as value from service_pub_info as p left join service_sub_info as s on p.data_id = s.data_id and p.instance_id = s.instance_
id where p.instance_id='OYPHWCTJNEYP';
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-----------------------------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra                                               |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-----------------------------------------------------+
|  1 | SIMPLE      | p     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |  5540 |    10.00 | Using where                                         |
|  1 | SIMPLE      | s     | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 15912 |   100.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-----------------------------------------------------+
2 rows in set, 1 warning (0.00 sec)
```

以下为正常索引：

```
MySQL [dsrconsole]> show index from service_sub_info;
+-----------------+------------+-----------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+
| Table           | Non_unique | Key_name                    | Seq_in_index | Column_name     | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
+-----------------+------------+-----------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+
| service_sub_info |          0 | PRIMARY                     |            1 | id              | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_sub_info |          0 | uniq_check_sum              |            1 | check_sum       | A         |        NULL | NULL     | NULL   | YES  | BTREE      |
| service_sub_info |          1 | idx_process_id_alive_old    |            1 | process_id      | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_sub_info |          1 | idx_process_id_alive_old    |            2 | alive           | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_sub_info |          1 | idx_processid_alive         |            1 | process_id      | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_sub_info |          1 | idx_processid_alive         |            2 | alive           | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_sub_info |          1 | key_instance_data_app       |            1 | instance_id     | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_sub_info |          1 | key_instance_data_app       |            2 | data_id         | A         |        NULL | 255      | NULL   |      | BTREE      |
| service_sub_info |          1 | key_instance_data_app       |            3 | app_name        | A         |        NULL | 255      | NULL   | YES  | BTREE      |
+-----------------+------------+-----------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+
9 rows in set (0.01 sec)

MySQL [dsrconsole]> show index from service_pub_info;
+-----------------+------------+-----------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+
| Table           | Non_unique | Key_name                    | Seq_in_index | Column_name     | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
+-----------------+------------+-----------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+
| service_pub_info |          0 | PRIMARY                     |            1 | id              | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_pub_info |          0 | uniq_check_sum              |            1 | check_sum       | A         |        NULL | NULL     | NULL   | YES  | BTREE      |
| service_pub_info |          1 | idx_process_id_alive_old    |            1 | process_id      | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_pub_info |          1 | idx_process_id_alive_old    |            2 | alive           | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_pub_info |          1 | idx_processid_alive         |            1 | process_id      | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_pub_info |          1 | idx_processid_alive         |            2 | alive           | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_pub_info |          1 | key_instance_data_app       |            1 | instance_id     | A         |        NULL | NULL     | NULL   |      | BTREE      |
| service_pub_info |          1 | key_instance_data_app       |            2 | data_id         | A         |        NULL | 255      | NULL   |      | BTREE      |
| service_pub_info |          1 | key_instance_data_app       |            3 | app_name        | A         |        NULL | 255      | NULL   | YES  | BTREE      |
+-----------------+------------+-----------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+
9 rows in set (0.00 sec)
```

可以看出用户的 SQL 中的 where 条件没有加索引，导致出现全表扫描。

解决方案

在 SQL 的 where 条件字段加 instanceId 索引，提高查询效率。

1.4.3.7. 发布部署时检查应用服务报错

问题现象

发布部署控制台在发布部署时，检查服务步骤报错：

排查步骤

用户环境：SOFABoot 版本为 3.1.1；技术栈版本为 2.0。

1. 查看应用启动日志 `stdout.log` 。

```
Failed to instantiate [com.alipay.sofa.rpc.boot.container.ServerConfigContainer]: Factory method 'serverConfigContainer' threw exception; nested exception is java.lang.NumberFormatException: For input string: "eth0"
```

出现类型转换异常， `serverConfigContainer` 未成功注入的报错，应该是配置错误导致。

2. 查看应用配置文件，未发现值为 eth0 的配置项。
3. 再次搜索 stdout 日志。


```
{"level":"DEBUG","time":"2020-09-07 10:44:21.284","logger":"org.springframework.web.context.support.StandardServletEnvironment","content":"Initialized StandardServletEnvironment with PropertySources [StubPropertySource@1879034789 {name='servletConfigInitParams', properties=java.lang.Object@34340fab}, StubPropertySource@716157500 {name='servletContextInitParams', properties=java.lang.Object@2b80d80f}, MapPropertySource@984849465 {name='systemProperties', properties={java.runtime.name=Java(TM) SE Runtime Environment, java.protocol.handler.pkgs=org.springframework.boot.loader, ********** rpc_enabled_ip_range=eth0**********}}]","exception":""}
```


发现应用启动参数里有 `rpc_enabled_ip_range` 的配置，此值为 IP 范围，不应填写 eth0 这样的网卡名称。

解决方案

修改技术栈里的 `rpc_enabled_ip_range` 为正确的 IP 段。

> **说明**
>
> 新建应用时 `rpc_enabled_ip_range` 值为技术栈默认值，若应用为默认配置，则修改技术栈配置即可；若应用进行了修改，还需要修改应用的 `rpc_enabled_ip_range` 值。

1.4.3.8. 发布的 RPC 服务在控制台中无法找到

问题现象

RPC 服务端发布之后，在微服务控制台无法找到该服务。

排查步骤

1. 查看 `/home/admin/logs/registry/registry-client.log` 。

```
2020-03-05 18:21:30,444 INFO RegistryWorkerThread - [Connect] Successfully connected to server: DefaultServerNode{url='100.103.xxx.xxx:9600', host='100.103.xxx.xxx', port=9600, properties=null} 2020-03-05 18:21:30,448 INFO Bolt-conn-event-executor-8-thread-1 - [connection] Client connected, remote address: 100.103.xxx.xxx:9600, localAddress: /172.19.xxx.xxx:32956 2020-03-05 18:21:30,492 INFO RegistryWorkerThread - [register] register to server success, fdba583c-ac4c-4d4b-8902-8de1bc5df04f, PublisherRegister{dataList=[DataBox{data=':12200?rpcVer=50507&serialization=hessian2&weight=100&timeout=3000&appName=xxxAppName&p=1&v=4.0&_SERIALIZETYPE=hessian2&_WEIGHT=100&_TIMEOUT=3000&app_name=xxxAppName&warmupWeight=0&warmupTime=0&startTime=1583403645419'}]}BaseRegister{instanceId='xxxxInstanceID', zone='false', appName='xxxAppName', dataId='com.alipay.xxx.xxx.xxxServiceAPI:1.0@DEFAULT', group='SOFA', processId='null', registId='8ce8a708-****-4de1-974c-f15c467132e1', clientId='null', dataInfoId='null', ip='null', port=null, eventType='REGISTER', version=1, timestamp=1583403690444, attributes={!Signature=xxx=, !AccessKey=xxx, !Algorithm=HmacSHA256, !Timestamp=1583403600000}}, RegisterResponse{success=true, registId='8ce8a708-****-4de1-974c-f15c467132e1', version=1, refused=false, message='Publisher register success!'}
```

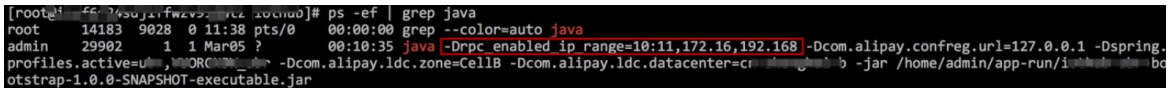
发现该 RPC 服务器已经发布服务成功，但注册信息中，服务器的 IP 为空（ `data=':12200` ）。

2. 查看 12200 端口的监听情况。

```
netstat -anp | grep 12200
```

发现 12200 端口并未监听在应用服务器的 IP（172.19.**.**）上。

3. 通过 `ps -ef | grep java` 命令查看 `rpc_enabled_ip_range` 配置。发布部署参数 `rpc_enabled_ip_range` 的值不在应用服务器对应的 IP 范围。



解决方案

在应用实例的发布部署参数中修改 `rpc_enabled_ip_range` 值为： `10:11,172.19,192.168` ，然后重新发布。

1.4.3.9. 因 DNS 配置错误导致 RPC 调用耗时 30 秒

问题现象

使用 REST 接口触发 RPC 的泛化调用，每次触发都需要 30 秒的时间。

排查步骤

1. 查看 RPC 的 digest 日志 `/home/admin/logs/tracelog/rpc-client-digest.log` 。

```
2020-01-21 10:42:04.186,rmfe_app,0a014046157957449410610155674,0,com.csxbank.rmfe.service.RmfeService:1.0,$genericInvoke,bolt,sync,10.**.**.72:12200,RMFE,,,,,00,2296B,889B,54ms,0ms,0ms,54ms,http-nio-8080-exec-10,REGISTRY,,,,10.1.64.72,12200,,,
```

日志显示，RPC 执行的时间仅 54 毫秒，判断不是 RPC 调用的问题，但从业务发生时间（业务发生调用的时间是 2020-01-21 10:41:34）开始到执行一共耗时 30 秒。

2. 查看 `/home/admin/logs/rpc/rpc-registry.log` 。

```
2020-01-21 10:41:31,330 INFO ObserverNotifyThread-2-thread-5 com.alipay.sofa.rpc.registry.dsr.DsrSubscribeCallback - RPC-00204: Receive RPC service addresses: service[com.csxbank.rmfe.service.RmfeService:1.0@DEFAULT] usable target addresses[3]
```

该接口的地址在业务发生调用前三秒就返回了，判断也不是寻址的问题。RPC 的超时时间是 3 秒，但实际调用耗时 30 秒却没有超时，判断业务在调用 RPC 之前进行了其他工作。

3. 在重现问题时每隔 5 秒收集一次 `jstack $PID` 到 `stack.log` ，至少收集 4 个日志。

   `$PID` 需要使用业务的 Java 进程 ID 替换。

4. 查看 `stack.log` 。

   通过 genericInvoke 快速定位到相应的线程，发现进程卡在了 OS 的方法 `getLocalHostName` 上。

   `getLocalHostName` 的作用是查找本机的 HostName。而 CentOS 查找的顺序是： `/etc/hosts` 文件 > DNS > myhostname。由于业务服务器的 DNS 配置错误，导致 DNS 超时。

解决方案

在业务服务器的 `/etc/hosts` 中添加 IP 与主机名的映射。

1.4.3.10. RPC 调用出现“RejectedExecutionException:Callback handler thread pool has bean exhausted”报错

问题现象

客户在使用 SOFA RPC 做压测时，在 SOFA RPC 的 consumer 端发现大量如下错误：

```
2020-04-21 21:11:05.530,xxx-webapi,c0a8eeeb158747466544135091170 33,0.1,SOFA-SEV-REST-BIZ-8341-7-T34,unknown_error,rpc,invokeType=sync&uid=&protocol=bolt&targetApp=xxx-xxxxxx&targetIdc=&targetCity=&paramTypes=com.xxxxx.xxx.xxxxxx.facade.model.xxxxxxxx&methodName=$genericInvoke&serviceName=com.xxxx.xxx.xxxxxxx.facade.xxxxxxxxxxx:1.0&targetUrl=xxx.xxx.xxx.xxx:12200&targetZone=&,,com.alipay.sofa.rpc.core.exception.SofaRpcException: java.util.concurrent.RejectedExecutionException: Callback handler thread pool has bean exhausted at com.alipay.sofa.rpc.client.AbstractCluster.doSendMsg(AbstractCluster.java:563) at com.alipay.sofa.rpc.client.AbstractCluster.sendMsg(AbstractCluster.java:480) at com.alipay.sofa.rpc.filter.ConsumerInvoker.invoke(ConsumerInvoker.java:60) at com.alipay.sofa.rpc.filter.ConsumerProfileFilter.invoke(ConsumerProfileFilter.java:60) at com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoker.java:96) at com.alipay.sofa.rpc.filter.sofatracer.ConsumerTracerFilter.invoke(ConsumerTracerFilter.java:66) at com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoker.java:96) at com.alipay.sofa.rpc.filter.ConsumerGenericFilter.invoke(ConsumerGenericFilter.java:80) at com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoker.java:96) at com.alipay.sofa.rpc.filter.RpcReferenceContextFilter.invoke(RpcReferenceContextFilter.java:80) at com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoker.java:96) at com.alipay.sofa.rpc.filter.ConsumerExceptionFilter.invoke(ConsumerExceptionFilter.java:37) at com.alipay.sofa.rpc.filter.FilterInvoker.invoke(FilterInvoker.java:96) at com.alipay.sofa.rpc.filte
```

```
r.FilterChain.invoke(FilterChain.java:302) at com.alipay.sofa.rpc.client.AbstractCluster.fi
lterChain(AbstractCluster.java:473) at com.alipay.sofa.rpc.client.FailoverCluster.doInvoke(
FailoverCluster.java:66) at com.alipay.sofa.rpc.client.AbstractCluster.invoke(AbstractClust
er.java:285) at com.alipay.sofa.rpc.client.ClientProxyInvoker.invoke(ClientProxyInvoker.jav
a:83) at com.alipay.sofa.rpc.api.GenericService_proxy_2.$genericInvoke(GenericService_proxy
_2.java) at com.xxxxx.xxx.webapi.endpoint.impl.WebapiRestFacadeRestImpl.dealWithLog(WebapiR
estFacadeRestImpl.java:459) at com.xxxxx.xxx.webapi.endpoint.impl.WebapiRestFacadeRestImpl.
dealWithToken(WebapiRestFacadeRestImpl.java:308) at com.xxxxx.xxx.webapi.endpoint.impl.Weba
piRestFacadeRestImpl.post(WebapiRestFacadeRestImpl.java:158) at sun.reflect.GeneratedMethod
Accessor83.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(Delega
tingMethodAccessorImpl.java:43) at java.lang.reflect.Method.invoke(Method.java:498) at org.
jboss.resteasy.core.MethodInjectorImpl.invoke(MethodInjectorImpl.java:137) at org.jboss.res
teasy.core.ResourceMethodInvoker.invokeOnTarget(ResourceMethodInvoker.java:296) at org.jbos
s.resteasy.core.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:250) at org.jboss.r
esteasy.core.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:237) at org.jboss.rest
easy.core.SynchronousDispatcher.invoke(SynchronousDispatcher.java:377) at com.alipay.sofa.r
pc.server.rest.SofaSynchronousDispatcher.invoke(SofaSynchronousDispatcher.java:49) at org.j
boss.resteasy.core.SynchronousDispatcher.invoke(SynchronousDispatcher.java:200) at org.jbos
s.resteasy.plugins.server.netty.RequestDispatcher.service(RequestDispatcher.java:83) at com
.alipay.sofa.rpc.server.rest.EnterpriseSofaRestRequestHandler.channelRead0(EnterpriseSofaRe
stRequestHandler.java:85) at io.netty.channel.SimpleChannelInboundHandler.channelRead(Simpl
eChannelInboundHandler.java:105) at io.netty.channel.AbstractChannelHandlerContext.invokeCh
annelRead(AbstractChannelHandlerContext.java:373) at io.netty.channel.AbstractChannelHandle
rContext.access$600(AbstractChannelHandlerContext.java:39) at io.netty.channel.AbstractChan
nelHandlerContext$7.run(AbstractChannelHandlerContext.java:364) at io.netty.util.concurrent
.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java:163) at io.netty.util.concurr
ent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:418) at io.netty.c
hannel.nio.NioEventLoop.run(NioEventLoop.java:454) at io.netty.util.concurrent.SingleThread
EventExecutor$5.run(SingleThreadEventExecutor.java:873) at java.lang.Thread.run(Thread.java
:748) Caused by: java.util.concurrent.RejectedExecutionException: Callback handler thread p
ool has bean exhausted at com.alipay.sofa.rpc.context.AsyncRuntime$1.rejectedExecution(Asyn
cRuntime.java:92) at java.util.concurrent.ThreadPoolExecutor.reject(ThreadPoolExecutor.java
:823) at java.util.concurrent.ThreadPoolExecutor.execute(ThreadPoolExecutor.java:1369) at c
om.alipay.sofa.rpc.event.EventBus.post(EventBus.java:122) at com.alipay.sofa.rpc.transport.
bolt.BoltClientTransport.syncSend(BoltClientTransport.java:270) at com.alipay.sofa.rpc.clie
nt.AbstractCluster.doSendMsg(AbstractCluster.java:509) ... 42 more
```

问题原因

报错信息为本地异步线程池已满。查看代码，确认是 RPC 的一个异步任务将收集的一些信息发到 lookout 组件，而这个组件已停止维护。

```
Caused by: java.util.concurrent.RejectedExecutionException: Callback handler thread pool has bean exhausted
    at com.alipay.sofa.rpc.context.AsyncRuntime$1.rejectedExecution(AsyncRuntime.java:92)
    at java.util.concurrent.ThreadPoolExecutor.reject(ThreadPoolExecutor.java:823)
    at java.util.concurrent.ThreadPoolExecutor.execute(ThreadPoolExecutor.java:1369)
    at com.alipay.sofa.rpc.event.EventBus.post(EventBus.java:122)
    at com.alipay.sofa.rpc.transport.bolt.BoltClientTransport.syncSend(BoltClientTransport.java:270)
    at com.alipay.sofa.rpc.client.AbstractCluster.doSendMsg(AbstractCluster.java:509)
    ... 42 more
```

解决方案

可以在 Consumer 的发布部署的环境参数添加或者使用 -D 传入 Java 的启动参数：

```
-Dasync.pool.core=100  // 默认是 10
-Dasync.pool.max=400  //默认是 200
-Dasync.pool.queue=1024  //默认是 256
```

1.4.3.11. RPC tracer 打印不出 rpc-client-digest.log

问题现象

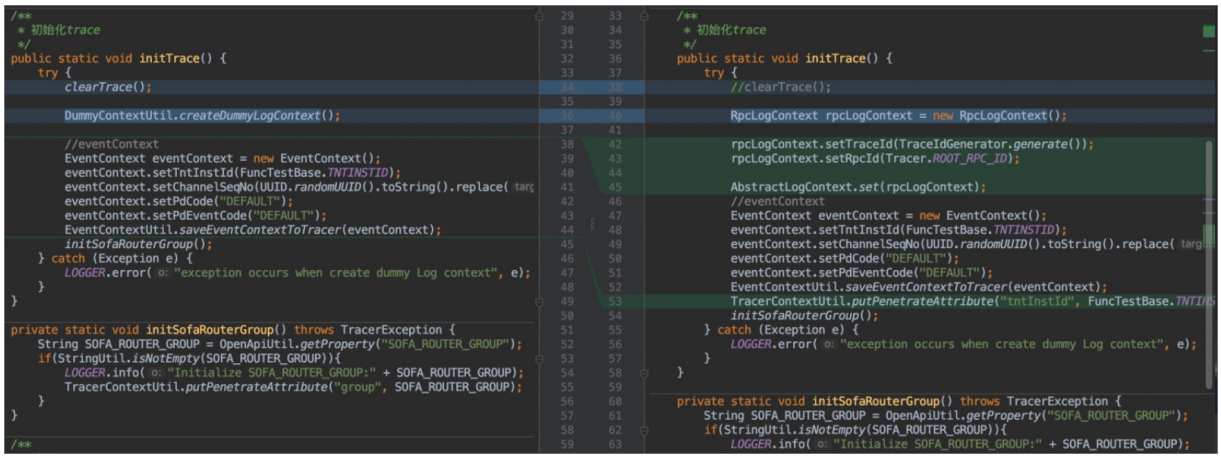
RPC 发生调用后，RPC tracer 打印不出 `rpc-client-digest.log` ，但是可以打印 `rpc-client-stat.log` 。

问题原因

客户的代码中手动的进行了日志初始化，在初始化时将 `isSampled` 设置成了 `false` 。

解决方案

将客户代码修改为下图左侧部分：



```
/**
 * 初始化trace
 */
public static void initTrace() {
    try {
        clearTrace();

        DummyContextUtil.createDummyLogContext();

        //eventContext
        EventContext eventContext = new EventContext();
        eventContext.setTntInstId(FuncTestBase.TNTINSTID);
        eventContext.setChannelSeqNo(UUID.randomUUID().toString().replace( targ
        eventContext.setPdCode("DEFAULT");
        eventContext.setPdEventCode("DEFAULT");
        EventContextUtil.saveEventContextToTracer(eventContext);
        initSofaRouterGroup();
    } catch (Exception e) {
        LOGGER.error( o: "exception occurs when create dummy Log context", e);
    }
}

private static void initSofaRouterGroup() throws TracerException {
    String SOFA_ROUTER_GROUP = OpenApiUtil.getProperty("SOFA_ROUTER_GROUP");
    if(StringUtil.isNotEmpty(SOFA_ROUTER_GROUP)){
        LOGGER.info( o: "Initialize SOFA_ROUTER_GROUP:" + SOFA_ROUTER_GROUP);
        TracerContextUtil.putPenetrateAttribute("group", SOFA_ROUTER_GROUP);
    }
}

/**
```

```
/**
 * 初始化trace
 */
public static void initTrace() {
    try {
        //clearTrace();

        RpcLogContext rpcLogContext = new RpcLogContext();

        rpcLogContext.setTraceId(TraceIdGenerator.generate());
        rpcLogContext.setRpcId(Tracer.ROOT_RPC_ID);

        AbstractLogContext.set(rpcLogContext);
        //eventContext
        EventContext eventContext = new EventContext();
        eventContext.setTntInstId(FuncTestBase.TNTINSTID);
        eventContext.setChannelSeqNo(UUID.randomUUID().toString().replace( targ
        eventContext.setPdCode("DEFAULT");
        eventContext.setPdEventCode("DEFAULT");
        EventContextUtil.saveEventContextToTracer(eventContext);
        TracerContextUtil.putPenetrateAttribute("tntInstId", FuncTestBase.TNTIN
        initSofaRouterGroup();
    } catch (Exception e) {
        LOGGER.error( o: "exception occurs when create dummy Log context", e);
    }
}

private static void initSofaRouterGroup() throws TracerException {
    String SOFA_ROUTER_GROUP = OpenApiUtil.getProperty("SOFA_ROUTER_GROUP");
    if(StringUtil.isNotEmpty(SOFA_ROUTER_GROUP)){
        LOGGER.info( o: "Initialize SOFA_ROUTER_GROUP:" + SOFA_ROUTER_GROUP);
```

> **说明**
>
> SOFABoot 框架会自动打印日志，不需要手动创建。

1.4.3.12. RPC TR 服务调用失败，出现“RPC-02412: Cannot find invoke method”报错

问题现象

RPC TR 服务调用失败， `/home/admin/logs/tracelog/rpc-client-digest.log` 日志中发现以下错误：

```
com.InterfaceName:1.0:captcha,verify,TR,SYNC,10.220.38.79:12200,ifccaptcha,CellA,,,,02,4205
B,341B,5ms,0ms,1ms,5ms,catalina-exec-9,CFS,,F,,10.220.38.39,40558
```

错误代码 02 表示 RPC 逻辑错误。

同时，在客户端应用的 `common-error.log` 中，发现了以下错误：

问题分析

客户端日志出现“RPC-02412: Cannot find invoke method: [verify]”报错，一般有以下原因：

- 服务端发布服务时，Spring XML 中 SOFA service 的连接配置错误。
- 服务端应用和客户端应用发布的版本有差距，导致代码不一致。

根据原因排查如下问题：

- 确认服务端版本

  对比客户端和服务端版本，确认版本一致，且服务端的 `common-error.log` 日志中也存在 RPC-02412 报错：

  ```
  cat /home/admin/logs/rpc/common-error.log | more
  2019-11-26 14:37:12,585 ERROR SofaBizProcessor-4-thread-9      - RPC-02412: Cannot find invoke method: [verify], service: [com.InterfaceName:1.0:captcha]
  java.lang.NoSuchMethodException: RPC-02412: Cannot find invoke method: [verify], service
  ```

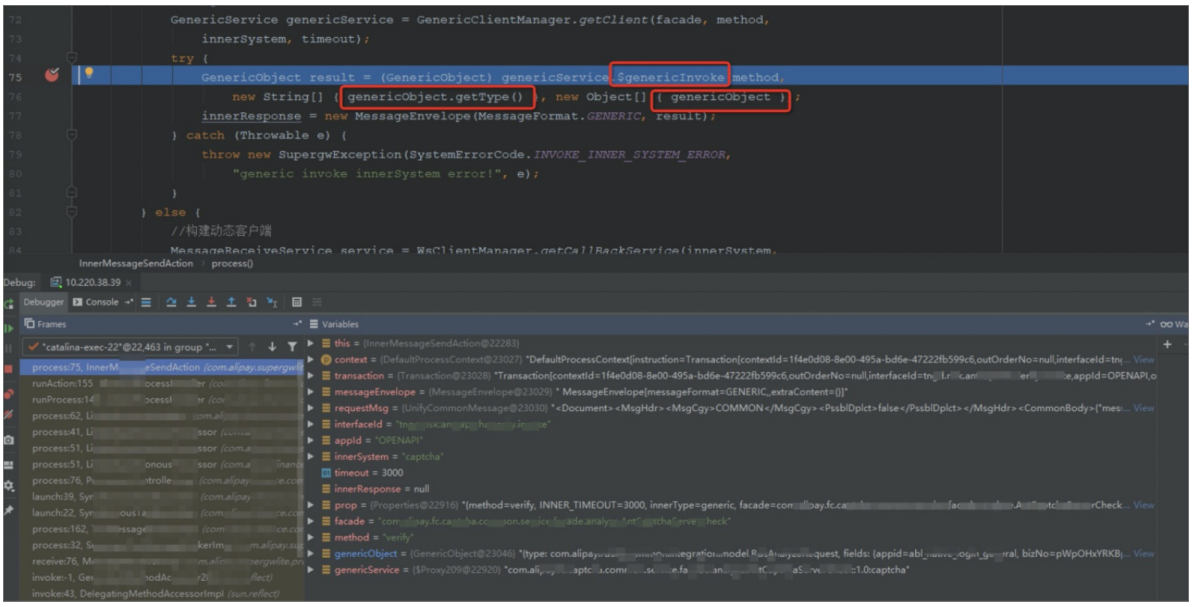
- 使用 TR 工具测试服务

  可以测试通过。

- 确认客户端调用 RPC 接口的方法

  客户使用了泛化调用和编程 API 的方式调用 RPC 接口，该方式不需要将服务端的 Facade 包放在 Maven 仓库或者客户端本地。

查看发现客户端泛化调用 RPC 服务的 verify 方法时，参数和服务端的该方法的参数是否匹配：



解决方案

将客户端的泛化调用参数修改为正确参数，配置方法请参见 泛化调用。

1.4.3.13. DRM 重启失败

问题现象

DRM 重启失败，卡在健康检查失败：

```
curl localhost:9500/checkService | grep false
```

DRM 需要订阅 TaskManageFacade，但是报错“Addresses unavailable”：

```
sofa-healthcheck.log报错:
2019-09-12 13:35:10,521 [Http-default-executor-4-thread-3] - appname:opssla,passed:false
>>[stage]component[id]reference:com.alipay.scheduler.common.service.facade.update.TaskManageFacade:#-704824331[passed]false[msg]tr#Addresses unavailable
>>[stage]component[id]reference:com.alipay.scheduler.common.service.facade.query.SchedulerQueryFacade:#-704824331[passed]false[msg]tr#Addresses unavailable
>>[stage]component[id]reference:com.alipay.zpaas.zappinfo.facade.query.ServerQueryFacade:#1838070941[passed]false[msg]tr#Addresses unavailable
>>[stage]component[id]reference:com.alipay.zpaas.zappinfo.facade.query.AppQueryFacade:#1838070941[passed]false[msg]tr#Addresses unavailable
>>[stage]component[id]reference:com.alipay.schedulerweb.service.facade.operate.TaskOperateFacade:#1348880705[passed]false[msg]tr#Addresses unavailable
2019-09-12 13:35:10,521 [Http-default-executor-4-thread-3] - appname : opssla Health Check is Ended!
```

排查步骤

1. 确认健康检测失败的原因。

   健康检查失败的日志显示 DRM 这台机器需要订阅的 RPC 服务没有订阅到。

2. 确认 DRM 和 DSR 的 session 服务器之间的连接是否正常。

   ```
   sh-4.1# netstat -anp | grep 9600
   tcp        0      0 11.[redacted].142:59482         11.[redacted].110:9600          ESTABLISHED -
   ```

   发现有长连接。

3. 查看 DSR 客户端日志 `/home/admin/logs/confreg/common-error.log` 。

```
2019-09-12 16:38:43,545 ERROR Deploy-Task:file:/home/admin/opssla-run/deploy/opssla.ace/ - Deserialize [{}] error. {}
java.io.EOFException
        at java.io.ObjectInputStream$PeekInputStream.readFully(ObjectInputStream.java:2671)
        at java.io.ObjectInputStream$BlockDataInputStream.readShort(ObjectInputStream.java:3146)
        at java.io.ObjectInputStream.readStreamHeader(ObjectInputStream.java:858)
        at java.io.ObjectInputStream.<init>(ObjectInputStream.java:354)
        at com.alipay.config.common.util.SerializeUtils.javaDeserialize(SerializeUtils.java:148)
        at com.alipay.config.client.work.BaseHttpClient.getResponse(BaseHttpClient.java:121)
        at com.alipay.drm.client.data.ZdrmdataAddressPool.pullInfoFromConfreg(ZdrmdataAddressPool.java:460)
        at com.alipay.drm.client.data.ZdrmdataAddressPool.init(ZdrmdataAddressPool.java:144)
        at com.alipay.drm.client.manager.DistributedResourceManagerImpl.realRegister(DistributedResourceManagerImpl.java:187)
        at com.alipay.drm.client.manager.DistributedResourceManagerImpl.register(DistributedResourceManagerImpl.java:144)
```

发现反序列化错误。

4. 登录 DRM 所连接的 DSR 服务器上，查看 `decision.60dt.log` 。

```
$ tail decision.60dt.log | more
2019-09-12 17:48:43,162 INFO  [decision] masterUrl is blank, got from: 11.[redacted].162
2019-09-12 17:48:54,166 INFO  [decision] masterUrl is blank, got from: 11.[redacted].162
2019-09-12 17:49:05,171 INFO  [decision] masterUrl is blank, got from: 11.[redacted].162
2019-09-12 17:49:16,175 INFO  [decision] masterUrl is blank, got from: 11.[redacted].162
```

发现当时这台 DSR 一直找不到 Data 服务器（11.**.**.162）的 master，所以即使收到客户端的连接，也无法获取数据。

5. 查看 `sdk.60dt.log.2019-09-12` 日志。

发现如下错误：

```
2019-09-12 16:53:04,210 ERROR - [cmdPost]Server handle request is error
java.lang.reflect.InvocationTargetException
        at com.alipay.config.sessionsrv.http.SessionSdkService$$FastClassByCGLIB$$f7df2643.invoke(<generated>)
        at com.google.inject.cglib.reflect.FastMethod.invoke(FastMethod.java:53)
        at com.alipay.config.server.pluginframework.invoker.AbstractInvoker.invokeMethod(AbstractInvoker.java:52)
        at com.alipay.config.server.pluginframework.invoker.DefaultActionInvoker.invoke(DefaultActionInvoker.java:49)
        at com.alipay.config.sessionsrv.http.SessionHttpHandler.excute(SessionHttpHandler.java:85)
        at com.alipay.config.server.httpsrv.AbstractHttpHandler.handleRequest(AbstractHttpHandler.java:191)
        at com.alipay.config.server.httpsrv.AbstractHttpHandler.handle(AbstractHttpHandler.java:69)
        at org.apache.http.nio.protocol.BufferingHttpServiceHandler$RequestHandlerAdaptor.handle(BufferingHttpServiceHandler.java
        at org.apache.http.nio.protocol.SimpleNHttpRequestHandler.handle(SimpleNHttpRequestHandler.java:55)
        at org.apache.http.nio.protocol.AsyncNHttpServiceHandler.processRequest(AsyncNHttpServiceHandler.java:457)
        at org.apache.http.nio.protocol.AsyncNHttpServiceHandler.inputReady(AsyncNHttpServiceHandler.java:320)
        at org.apache.http.nio.protocol.BufferingHttpServiceHandler.inputReady(BufferingHttpServiceHandler.java:135)
        at org.apache.http.impl.nio.DefaultNHttpServerConnection.consumeInput(DefaultNHttpServerConnection.java:179)
        at org.apache.http.impl.nio.DefaultServerIOEventDispatch.inputReady(DefaultServerIOEventDispatch.java:145)
        at org.apache.http.impl.nio.reactor.BaseIOReactor.readable(BaseIOReactor.java:153)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.processEvent(AbstractIOReactor.java:314)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.processEvents(AbstractIOReactor.java:294)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.execute(AbstractIOReactor.java:256)
        at org.apache.http.impl.nio.reactor.BaseIOReactor.execute(BaseIOReactor.java:96)
        at org.apache.http.impl.nio.reactor.AbstractMultiworkerIOReactor$Worker.run(AbstractMultiworkerIOReactor.java:556)
        at java.lang.Thread.run(Thread.java:662)
Caused by: org.apache.http.conn.ConnectTimeoutException: Connect to /11.[redacted].161:9604 timed out
        at org.apache.http.conn.scheme.PlainSocketFactory.connectSocket(PlainSocketFactory.java:125)
        at org.apache.http.impl.conn.DefaultClientConnectionOperator.openConnection(DefaultClientConnectionOperator.java:123)
        at org.apache.http.impl.conn.AbstractPoolEntry.open(AbstractPoolEntry.java:147)
        at org.apache.http.impl.conn.AbstractPooledConnAdapter.open(AbstractPooledConnAdapter.java:108)
        at org.apache.http.impl.client.DefaultRequestDirector.execute(DefaultRequestDirector.java:415)
```

11.**.**.161 可能是之前的 DSR data master 服务器。

6. 尝试直接 SSH 链接 11.**.**.161。

发现无法 SSH，客户的 SRE 之前重新发布了这些 Data 服务器，但是其实这台服务器一直处于异常状态。

解决方案

重启有问题的 DSR Data 服务器 11.**.**.161，并重新发布 confregdata。

1.4.3.14. 发布部署卡在部署服务中，直到 8 分钟后超时

问题现象

发布部署卡在部署服务中，直到 8 分钟后超时，导致发布部署失败。

问题原因

检查业务代码后发现，业务代码在收到 DRM 的 `RefreshCacheDRM.refreshCacheType` 属性值为 `GEOHASH` 的更新后，需要花费 10 多分钟处理业务逻辑。

解决方案

临时方案： 设置 `RefreshCacheDRM.refreshCacheType` 的值为空，暂时不触发业务处理逻辑。

长期方案： 需要优化业务代码，在收到 DRM 的属性更新后，使用异步线程延迟处理该业务，并及时反馈更新成功信号给 DRM。

1.4.3.15. SOFA RPC 泛化调用超过三次后报错

问题描述

普通 Java 应用编程泛化调用 RPC 服务超过三次后出现如下报错信息：

```
Duplicate consumer config with key "" has been referred more than 3 times! Maybe it's wrong config, please check it. Ignore this if you did that on purpose!
```

问题原因

RPC 代理实例仅能为单例，但测试代码未使用单例模式，导致每次均重新构造新的实例。

解决方案

- 使用 SOFABoot 编写测试代码，详情请参见 泛化调用。
- 使用单例模式获取代理实例。

1.4.3.16. DsrConsole common-error.log 日志出现“Query cells from osp error”报错

问题现象

DsrConsole 的 `common-error.log` 日志出现“Query cells from osp error”报错。

问题原因

Accesskey（AccessKey ID、AccessKey Secret） 对应账号的权限不够，未具备以下接口功能权限： • antcloud.deps.singleworkspace.get • antcloud.deps.singleworkspacegroup.get • antcloud.deps.cell.list

解决方案

1. 修改对应账号权限或修改为有足够权限的账号的 AccessKey，并重新发布 OSP。

2. 重启 DsrConsole 容器。

1.4.4. 服务异常应急预案

问题描述

- 定时任务触发失败
- 动态配置推送失败

环境检查

1. 登录云游，检查 antscheduler、drmdata、DsrConsole 容器状态。
2. 登录 RMS 监控页面，检查 antscheduler、drmdata、DsrConsole 端口 9001、9880、80 是否在监听。

实施步骤

- 如果容器为 RUNNING 状态，`kubectl exec` 命令进入容器，执行以下命令，检查 antscheduler、drmdata、DsrConsole 的服务状态。

```
#ps -ef | grep java
```

- 如果容器为 Error 状态，尝试重建 Pod。

  另查看物理节点是否异常，判断是否存在物理机宕机。对故障物理机上运行的容器进行宕机迁移，并在新宿主机上启动容器。

结果验证

1. 手工触发消息、RPC 定时任务成功，并且消息消费端可以成功接收到消息。
2. 在 DRM 控制台，找到一个可以测试的 dataid 进行手工推送，确认消息可以成功推送到消息订阅端。

2.注册中心

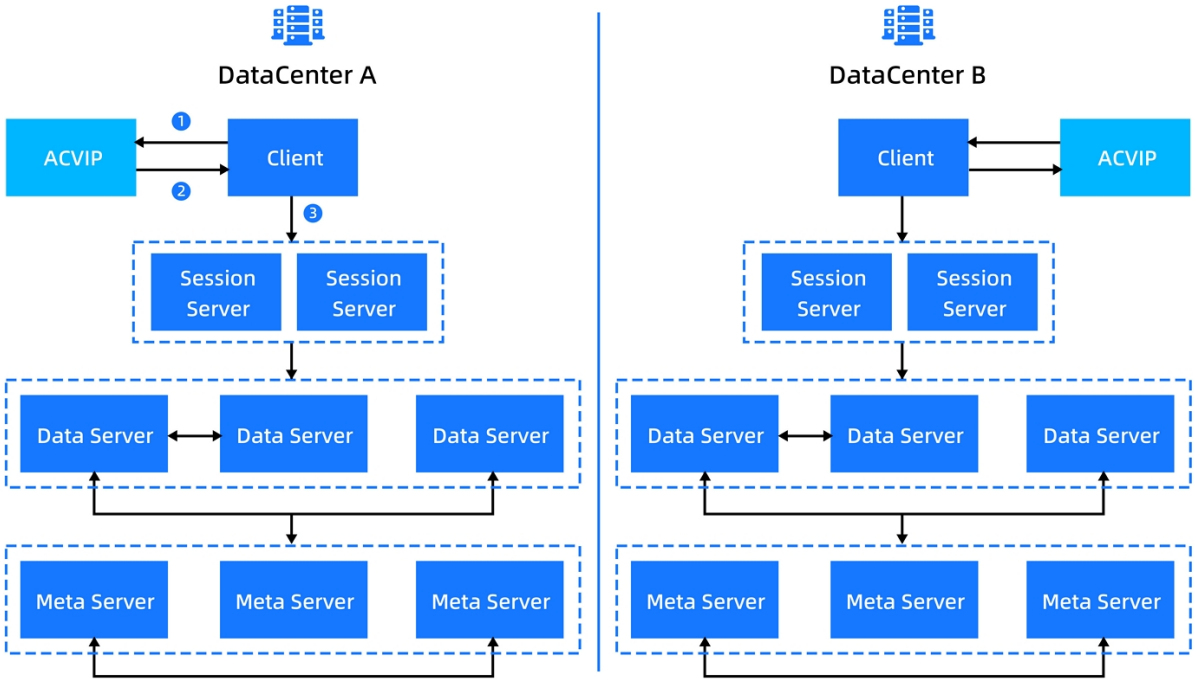
2.1. 产品概述

2.1.1. 产品介绍

注册中心 SOFARegistry 是服务提供者和服务使用者之间的桥梁，它包含 4 个角色：

- **Client**：提供应用接入服务注册中心的基本 API 能力，应用系统通过依赖客户端 JAR 包，通过编程方式调用服务注册中心的服务订阅和服务发布能力。
- **SessionServer**：会话服务器，负责接受 Client 的服务发布和服务订阅请求，并作为一个中间层将写操作转发 DataServer 层。SessionServer 这一层可随业务机器数的规模的增长而扩容。
- **DataServer**：数据服务器，负责存储具体的服务数据。数据按 dataInfoId 进行一致性 hash 分片存储，支持多副本备份，保证数据高可用。这一层可随服务数据量的规模的增长而扩容。
- **MetaServer**：元数据服务器，负责维护集群 SessionServer 和 DataServer 的一致列表，作为 SOFARegistry 集群内部的地址发现服务，在 SessionServer 或 DataServer 节点变更时可以通知到整个集群。

2.1.2. 部署架构



该部署架构图以双机房部署（和 DataCenter B）的场景为例。数据访问流程如下：

1. Client 向服务发现组件（ACVIP）查询注册中心的 SessionServer 的可用域名。
2. ACVIP 返回 SessionServer 的可用域名给 Client。
3. Client 向注册中心的 SessionServer 发起服务发布和订阅的请求。

2.1.3. 附录：基础术语

中文	英文	释义
服务注册中心	SOFARegistry	蚂蚁科技开源的一款服务注册中心产品，基于“发布-订阅”模式实现服务发现功能。同时它并不假定总是用于服务发现，也可用于其他更一般的“发布-订阅”场景。
数据	Data	在服务发现场景下，特指服务提供者的网络地址及其它附加信息。其他场景下，也可以表示任意发布到 SOFARegistry 的信息。
单元	Zone	单元化架构关键概念，在服务发现场景下，单元是一组发布与订阅的集合，发布及订阅服务时需指定单元名，更多内容可参考异地多活单元化架构解决方案。
发布者	Publisher	发布数据到 SOFARegistry 的节点。在服务发现场景下，服务提供者就是“服务提供者的网络地址及其它附加信息”的发布者。
订阅者	Subscriber	从 SOFARegistry 订阅数据的节点。在服务发现场景下，服务消费者就是“服务提供者的网络地址及其它附加信息”的订阅者。
数据标识	DataId	用来标识数据的字符串。在服务发现场景下，通常由服务接口名、协议、版本号等信息组成，作为服务的标识。
分组标识	GroupId	用于为数据归类的字符串，可以作为数据标识的命名空间，即只有 DataId、GroupId、InstanceId 都相同的服务，才属于同一服务。
实例 ID	InstanceId	实例 ID，可以作为数据标识的命名空间，即只有 DataId、GroupId、InstanceId 都相同的服务，才属于同一服务。
会话服务器	SessionServer	SOFARegistry 内部负责跟客户端建立 TCP 长连接、进行数据交互的一种服务器角色。
数据服务器	DataServer	SOFARegistry 内部负责数据存储的一种服务器角色。
元信息服务器	MetaServer	SOFARegistry 内部基于 Raft 协议，负责集群内一致性协调的一种服务器角色。

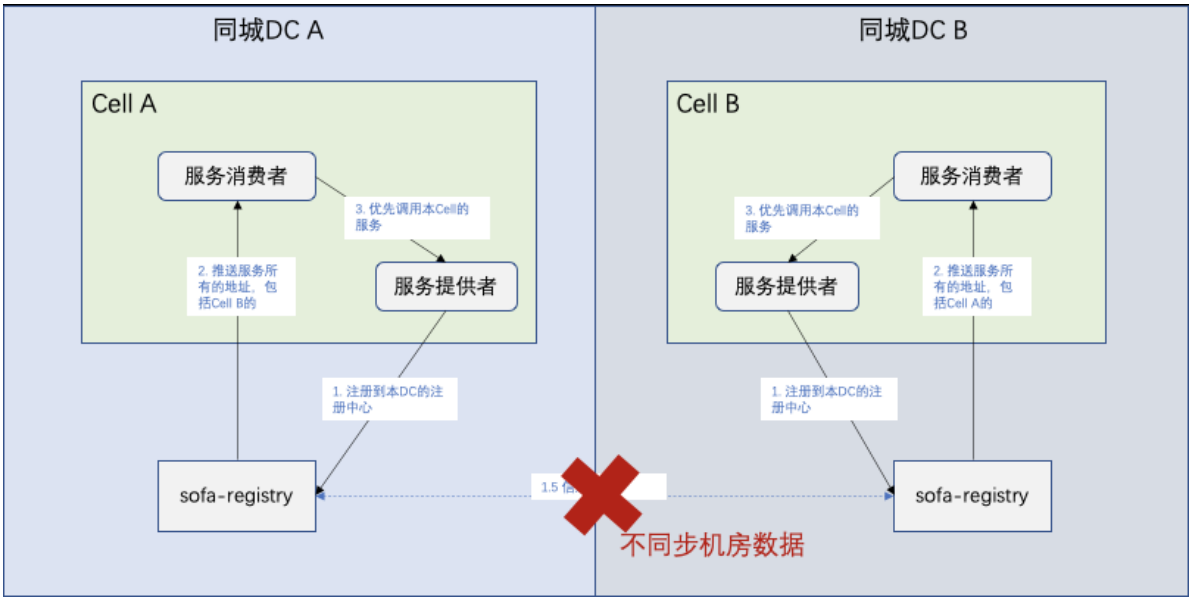
中文	英文	释义
数据中心	Data Center	物理位置、供电、网络具备一定独立性的物理区域，通常作为高可用设计的重要考量粒度。一般可认为：同一数据中心内，网络质量较高、网络传输延时较低、同时遇到灾难的概率较大；不同数据中心间，网络质量较低、网络延时较高、同时遇到灾难的概率较小。

2.2. 同城容灾

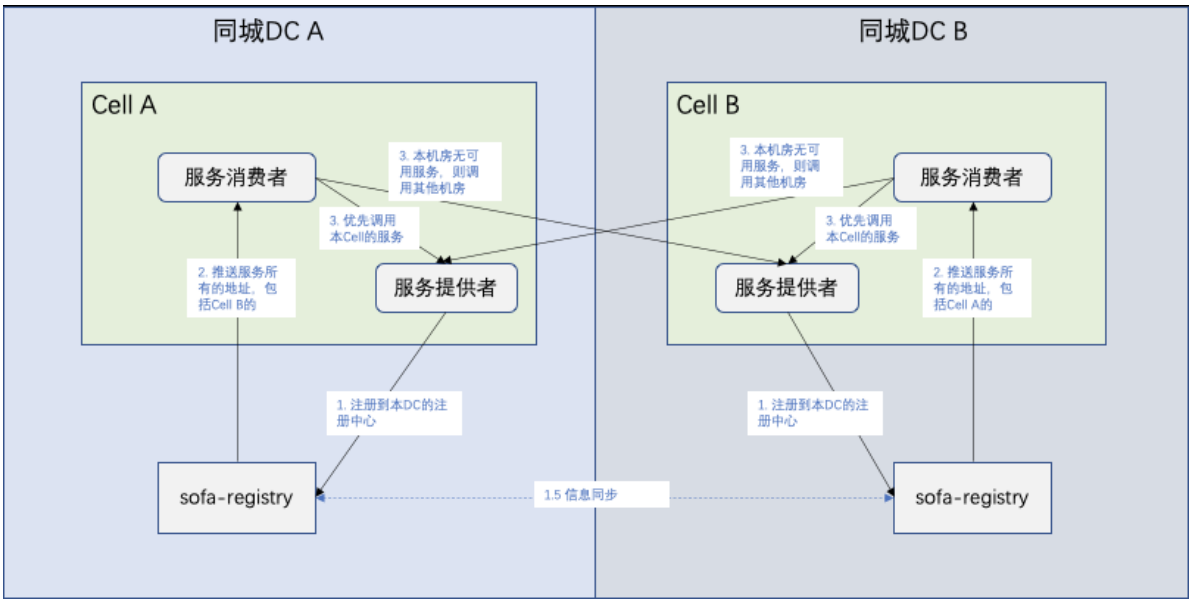
容灾架构

注册中心的容灾架构图如下所示：

- 不开启信息同步的架构



- 开启信息同步的架构



容灾架构说明如下：

- 双机房需对等部署。
- 如果开启信息同步，则数据内部双向同步，点对点通信，可以支持业务应用的机房级高可用。
- 客户端通过自带的 DATACENTER 属性来完成正常情况下的单机房流量收敛。
- 由于存在业务集群跨机房互相访问的情况，所以要求业务集群间网络互通，从而实现“单机房 SOFA Registry 宕机后，不会影响用户应用”。

> 注意
>
> 业务应用通过 ACVIP 查找 SOFA Registry 地址是中间件通用逻辑，架构图中省略了这部分逻辑。

产品依赖

外部产品依赖

ACVIP：客户端依赖 ACVIP 做路由寻址，获得跨机房连接能力。

底座产品依赖

元集群：元集群主备机房独立部署，无容灾要求。

数据库依赖

不依赖数据库，使用内存。

数据同步

SOFA Registry 数据同步采用按需拉取的方式，主备机房会通过“增量同步 + 定时全量拉取”的方式同步 version 信息，但不会实际存储 datum 信息。当请求到对应 version 信息时，如果发现本地无实际 datum 信息，会去对端机房拉取。

网络链路

业务应用通过 ACVIP 查询 SOFA Registry 地址，建立 TCP 长连接。

容灾能力

主机房 SOFA Registry 宕机

主机房 SOFA Registry 宕机后的处理逻辑如下：

- 主机房的业务应用无法访问本机房的 SOFA Registry，但可以通过本地缓存继续访问本机房的服务提供者。
- 主机房的业务应用可通过 ACVIP 自动切换备机房的 SOFA Registry。
- 因为备机房的 SOFA Registry 只保存了主机房的 version 信息，当主机房的业务应用请求数据时，会尝试去主机房拉取 datum 信息，但会获取不到。
- 如果主备机房业务应用对等部署，备机房的 SOFA Registry 会返回本机房注册的服务提供者信息。在主备机房业务集群网络互通的情况下，能保证主机房业务应用的服务连续性。
- 如果主备机房业务应用非对等部署或网络不通，则会影响业务应用的连续性：
  - 非对等部署的情况下，根据用户需求沟通业务应用改造。
  - 网络不通的情况下，可以根据需求，考虑是否进行整体切换。

主机房业务应用（服务提供者）宕机

主机房业务应用（服务提供者）宕机后的处理逻辑如下：

主机房服务消费者访问本机房的 SOFA Registry，获取服务提供者地址。但由于本机房的服务提供者宕机，SOFA Registry 无法获取本机房的服务提供者地址。

- 已开启信息同步且主备机房网络互通

  SOFA Registry 根据本地同步的备机房 version 信息，从备机房拉取服务提供者地址，并返回给服务消费者。

- 未开启信息同步或主备机房网络不通

  无法获取服务提供者地址，业务应用的服务连续性受到影响。

单机房整体宕机

机房需进行整体切换，切换后功能无降级。

网络中断

网络恢复后会自动重连。

> **说明**
>
> 在网络中断期间，Dsrconsole、ACVIP、OSP 及其他和管控台有关联的组件均无法使用。

2.3. 日常运维

2.3.1. 运维目录

目录结构

```
/home/admin 主目录
├── app-run              执行文件目录
├── bin                      日志清理程序目录
├── ccbin                   运行脚本目录
├── conf                     日志清理程序配置文件目录
├── log                      日志目录
├── raftData             持久化数据
└── release                  程序发布目录
```

功能介绍

功能名称	对应程序	路径	说明
registry-meta	registry-server-meta-enterprise-executable.jar	/home/admin/release/run/registry-server-meta-enterprise-executable.jar	MetaServer，用于集群内部互相发现、元数据存储。
registry-data	registry-server-data-enterprise-executable.jar	/home/admin/release/run/registry-server-data-enterprise-executable.jar	DataServer，用于 Pub 数据存储。
registry-session	registry-server-session-enterprise-executable.jar	/home/admin/release/run/registry-server-session-enterprise-executable.jar	SessionServer，用于对接客户端连接、Pub 和 Sub 数据上行和下行。

2.3.2. 启停说明

重启等操作使用云游标准操作。因为注册中心数据变动会引起客户流量的抖动，因此建议每次重启注册中心时进行关闭推送的操作，并进行如下检查：

1. 保存发布前现场。
   - 记录每台 SessionServer 上的连接，命令如下：

     ```
     curl http://<session_host>:9603/connections/query
     或者
     netstat -nl|grep 9600|grep ESTABLISHED
     ```

   - 记录 SessionServer 上的 Pub、Sub 数目，命令如下：

     ```
     curl http://<session_host>:9603/digest/data/count
     ```

   - 记录 DataServer 上的 Pub 数目，命令如下：

     ```
     curl http://<data_host>:9622/digest/datum/count
     ```

2. 关闭推送。

操作方式请参见 关闭推送。

3. 执行发布，发布过程进行验证。

   验证方式请参见下方各服务的验证方式。

> 注意
>
> MetaServer 发布完，为了确保推送开关处于 close 状态，必须再推送一次。

验证方式

- MetaServer 验证方法
  - 检查健康检查接口：`curl http://localhost:9615/health/check`
  - 检查端口：

```
netstat -nl|grep :9610
netstat -nl|grep :9611
netstat -nl|grep :9612
netstat -nl|grep :9615

netstat -nl|grep :9614
```

  - 检查启动日志：`tail -100 /home/admin/logs/registry/meta/registry-meta.log`
  - 检查来自 DataServer 的连接数（应该和 DataServer 的数量一致）：

```
netstat -an|grep :9611|grep ESTABLISHED -c
```

  - 检查来自 SessionServer 的连接数（应该和 SessionServer 的数量一致）：

```
netstat -an|grep :9610|grep ESTABLISHED -c
```

  - 查询 MetaServer 列表：`curl http://localhost:9615/digest/meta/node/query`
  - 查询 DataServer 列表：`curl http://localhost:9615/digest/data/node/query`
  - 查询 SessionServer 列表：`curl http://localhost:9615/digest/session/node/query`
  - 查询 raft 日志：

```
grep StateMachineAdapter /home/admin/logs/registry/meta/registry-raft.log -color
```

- DataServer 验证方法
  - 检查健康检查接口：`curl http://localhost:9622/health/check`
  - 检查端口：

```
netstat -nl|grep :9620
netstat -nl|grep :9621
netstat -nl|grep :9622

netstat -nl|grep :9623
```

- 检查状态是否变为 working：
  ```
  grep "to WORK" /home/admin/logs/registry/data/registry-data.log -color
  ```
- 检查启动日志：`tail -100 /home/admin/logs/registry/data/registry-data.log`
- 检查来自 DataServer 的连接数（应该是全部 IDC 的 DataServer 总数量减 1）：
  ```
  netstat -an | grep ESTABLISHED | awk '{print $4}' | grep :9621 -c
  ```
- 检查向其他data的连接数（应该是全部idc的data总数量减 1）：
  ```
  netstat -an | grep ESTABLISHED | awk '{print $5}' | grep :9621 -c
  ```
- 检查来自 SessionServer 的连接数（SessionServer 的数量 * 10）：
  ```
  netstat -an|grep :9620|grep ESTABLISHED -c
  ```
- 检查向 MetaServer 建立的连接数（应该和 MetaServer 的数量一致）：
  ```
  netstat -an|grep :9611|grep ESTABLISHED -c
  ```

- SessionServer 验证方法
  - 检查健康检查接口：`curl http://localhost:9603/health/check`
  - 检查端口：
    ```
    netstat -nl|grep :9600
    netstat -nl|grep :9601
    netstat -nl|grep :9603

    netstat -nl|grep :9602
    netstat -nl|grep :9604
    ```
  - 检查启动日志：`tail -100 /home/admin/logs/registry/session/registry-session.log`
  - 检查推送状态：`curl http://localhost:9603/digest/pushSwitch`
  - 检查向 DataServer 的连接数量（DataServer 的数量 * 10）：
    ```
    netstat -an|grep :9620|grep ESTABLISHED -c
    ```
  - 检查向 MetaServer 的连接数量（应该和 MetaServer 的数量一致）：
    ```
    netstat -an|grep :9610|grep ESTABLISHED -c
    ```
  - 检查来自 Client 的连接数：`netstat -an|grep :9600|grep ESTABLISHED -c`
  - 检查 9600 端口的 top 连接数：
    ```
    netstat -an|grep 9600|awk '{print $5}'|awk -F: '{print $1}'|sort -rn|uniq -c|sort -rn|head
    ```

- 数据校验

  在每个 DataServer 和 SessionServer 发布前和发布后执行如下命令，然后分别求和后对比，确保 DataServer 和 SessionServer 的数据基本相等（重启前后差值 5% 以内属于正常现象，如果差值过大，则需要确认问题原因）。

- DataServer: `curl http://<data_host>:9622/digest/datum/count`
- SessionServer: `curl http://<session_host>:9603/digest/data/count`

2.3.3. 打开推送

打开推送后，如果服务有更新，SessionServer 会自动将更新的内容推送给已订阅该服务的客户端。日常生产环境中，您需要打开推送。

操作步骤

1. 打开推送。

   调用其中一台 MetaServer 机器即可：`curl http://<meta_host>:9615/stopPushDataSwitch/close`

   。
2. 检查推送状态。
   - 通过以下两种方式检查 MetaServer:
     - grep 日志：

       ```
       grep "stop push data switch to DB result" /home/admin/logs/registry/meta/registry-meta.log -color
       ```

     - 访问 URL: `curl http://<meta_host>:9615/digest/pushSwitch`
   - 通过以下两种方式检查 SessionServer:
     - grep 日志：

       ```
       grep "Fetch session stop push data switch" /home/admin/logs/registry/session/registry-session.log --color
       ```

     - 访问 URL: `curl http://<session_host>:9603/digest/pushSwitch`

2.3.4. 关闭推送

关闭推送指的是在 SessionServer 上设置一个开关，让 SessionServer 不再下发数据，这样可以让客户端使用本地缓存继续运行，从而屏蔽掉注册中心变更中的状态变化。

背景信息

由于注册中心是通过长链接保活，使用 4 元组（客户端 IP+Port、SessionServer 的 IP+Port）标识一个客户端，所以如果 SessionServer 重启时，会有客户端的上下线，如果不关闭推送，客户侧订阅的信息会有抖动。

DataServer 存储聚合的 Pub 数据，单机房 DataServer 重启时，会有数据迁移，可能引起客户侧订阅数据的抖动。MetaServer 负责集群内部服务发现，单机房 DataServer 重启时，理论上没有影响，但是如果引起了 Session 和 Data 的拓扑变化，也有可能有影响。如果需要手动运维，建议多机房同时关闭，避免单机房操作。

操作步骤

1. 关闭推送。

调用其中一台 MetaServer 机器即可，命令如下：

```
curl http://<meta_host>:9615/stopPushDataSwitch/open
```

2. 检查推送状态。
   - 检查 MetaServer，命令如下：

     ```
     curl http://<meta_host>:9615/digest/pushSwitch
     ```

   - 检查 SessionServer，命令如下：

     ```
     curl http://<session_host>:9603/digest/pushSwitch
     ```

2.3.5. 监控和预警

2.3.5.1. 监控指标和预警项

MetaServer

监控项	日志路径	告警配置	说明
System（系统指标）	-	• CPU 使用率 > 90% • 内存使用率 > 90% • 磁盘使用率 > 90%	展示当前的设备负载状态。
checkService（端口指标）	-	9610	MetaServer 的服务端口，如果端口关闭，则 MetaServer 服务不可用。
MetaServer 错误日志	/home/admin/logs/registry/meta/common-error.log	最近 5 分钟持续 > 0	MetaServer 的普通报错。
dataNodeList_is_empty_prod	/home/admin/logs/registry/meta/registry-metrics.log	DataServer 连接数 = 0	当 MetaServer 上显示连接的 DataServer 的数量为 0 时，需要确认本中心的 DataServer 是否服务正常，与 MetaServer 连接正常。
metaNodeList_is_empty_prod	/home/admin/logs/registry/meta/registry-metrics.log	MetaServer 连接数 = 0	当 MetaServer 上显示连接的 MetaServer 的数量为 0 时，需要确认本中心的 MetaServer 节点是否服务正常，MetaServer 之间互相连接是否正常。

监控项	日志路径	告警配置	说明
metaRenew_fail_prod	/home/admin/logs/registry/meta/common-error.log	最近 3 分钟心跳持续 = 0	当 MetaServer 检测 SessionServer 或 DataServer 的心跳持续为 0 时，需要确认本中心的 DataServer 或者 SessionServer 节点是否服务正常，和 MetaServer 连接是否正常。
meta_gc_cms_prod	/home/admin/logs/gc.log	最近 10 分钟求和 > 5	当 MetaServer 的 full GC 监控最近 10 分组的求和 > 5 时，需要确认本中心 MetaServer 的内存是否正常、是否发生 full GC。您可能需要 dump 内存，检查问题原因。
meta_health_check_fail_prod	/home/admin/logs/registry/meta/registry-metrics.log	最近 3 分钟健康检查持续 > 0	当 MetaServer 的健康检查失败时，需要确认本中心 MetaServer 服务是否正常，具体是哪条健康检查有问题。
meta_push_status_is_closed_prod	/home/admin/logs/registry/meta/registry-metrics.log	最近 3 分钟推送数据持续 = 0	当数据推送持续为 0 时，需要确认 MetaServer 上的数据推送开关是否已关闭。
meta_role_change_prod	/home/admin/logs/registry/meta/registry-raft.log	-	当 MetaServer 上发生了 raft 选主时，需要确认引发 jraft 选主的原因，一般是 IO 夯住，或网络抖动导致心跳超时，引发了 jraft 选主。
sessionNodeList_is_empty_prod	/home/admin/logs/registry/meta/registry-metrics.log	SessionServer 连接数 = 0	当 MetaServer 上显示有连接的 SessionServer 的数量为 0 时，需要确认本中心的 MetaServer 节点是否服务正常，SessionServer 和 DataServer 之间互相连接是否正常。

DataServer

监控项	日志路径	告警配置	说明

监控项	日志路径	告警配置	说明
System(系统指标)	-	• CPU 使用率 > 90% • 磁盘使用率 > 90% • load5 > 3 • 内存使用率 > 90%	展示当前的设备负载状态。
checkService（端口指标）	-	9620	DataServer 的服务端口，如果端口关闭，则 DataServer 服务不可用。
dataDatum_total_prod	/home/admin/logs/registry/data/cache-digest.log	总数 > 100000	DataServer 上显示的数据 ID，该数值反应了目前 DataServer 上的数据数量，是一个运行状态的监控数据，用来评估 DataServer 的运行压力。
dataPub_total_prod	/home/admin/logs/registry/data/cache-digest.log	-	DataServer 上显示的数据 Pub，该数值反应了目前 DataServer 上的数据量，是一个运行状态的监控数据，用来评估 DataServer 的运行压力。
dataRefreshLeader_fail_prod	/home/admin/logs/registry/data/common-error.log	最近 3 分钟持续 > 0，短信报警	DataServer 去 MetaServer 上获取最新的 raft leader 失败，您需要确认目前本集群的 MetaServer 状态是否正常，在报错时，是否在进行 raft 选主。
dataRenewEvict_Expired_prod	/home/admin/logs/registry/data/registry-datum-lease.log	最近 60 分钟求和 > 100，短信报警	DataServer 剔除过期数据。当最近 60 分钟剔除数量求和超过 100 时，需要确认目前本集群是否有 SessionServer 出现异常，或者有网络抖动。
dataRenewEvict_Noheartbeat_prod	/home/admin/logs/registry/data/registry-datum-lease.log	最近 60 分钟求和 > 100，短信报警	DataServer 剔除没有心跳的数据。当最近 60 分钟剔除数量求和超过 100 时，需要确认目前本集群是否有 SessionServer 出现异常，或者有网络抖动。
dataRenew_fail_prod	/home/admin/logs/registry/data/common-error.log	最近 3 分钟持续 > 0，短信报警	DataServer 更新数据失败，需要确认 DataServer 为何会出现更新数据失败。

监控项	日志路径	告警配置	说明
dataRenew_Snapshot_prod	/home/admin/logs/registry/data/registry-renew.log	最近 60 分钟求和 > 100，短信报警	DataServer 更新快照。当更新快照持续报错时，需要确认一下原因。
dataSessionOff_trigger(data_missing)_prod	/home/admin/logs/registry/data/registry-data.log	-	表示和 SessionServer 断开，DataServer 上有数据清除 ，需要确认是否有 SessionServer 在进行重启操作。
data_common_error_prod	/home/admin/logs/registry/data/common-error.log	• 最近 5 分钟持续 > 0 • 短信报警最近 10 分钟求和 > 100 满足一个就发短信报警。	DataServer 的普通报错。
data_gc_cms_prod	/home/admin/logs/gc.log	CMS-initial-mark 当出现 Full GC 时会产生告警。	DataServer 上的 full GC报警 。出现时，需要确认进行 full GC 的原因，有可能需要 dump 一下内存。
data_health_check_fail_prod	/home/admin/logs/registry/data/registry-metrics.log	最近 3 分钟持续 > 0，短信报警	DataServer 的健康检查失败，需要确认检查失败的原因。
data_LocalDataServerChangeEvent_prod	/home/admin/logs/registry/data/registry-data.log	最近 10 分钟求和 > 2，短信告警	集群的 DataServer 的节点发生变化 ，需确认是否有 DataServer 重启、宕机或发布中。
data_SessionNotifierInvoke_Error_prod	/home/admin/logs/registry/data/common-error.log	• 最近 10 分钟求和 > 100 • 最近 10 分钟持续 > 0 满足一个就发短信报警。	数据变化后，通知 SessionServer 变化失败。需确认 DataServer 和 SessionServer 之间连接是否正常，是否有网络断开或者 SessionServer 重启的情况。
data_SessionNotifierRetryExceed_Error_prod	/home/admin/logs/registry/data/common-error.log	最近 10 分钟求和 > 0，短信告警	数据变化后，通知 SessionServer 变化超过最大重试次数。需确认 DataServer 和 SessionServer 之间连接是否正常，是否有网络断开或者 SessionServer 重启的情况。

SessionServer

监控项	日志路径	报警配置	说明
System（系统指标）	-	• CPU 使用率 > 90% • 磁盘使用率 > 90% • 内存使用率 > 90%	展示当前的设备负载状态。
checkService（端口指标）	-	9600、9603	SessionServer 的服务端口，如果端口关闭，则 SessionServer 服务不可用。
sessionPub_total	/home/admin/logs/registry/session/registry-console.log	-	SessionServer 上 Pub 的数据统计，统计每台机器上的 Pub 总量，反应了目前 SessionServer 上的数据数量，是一个运行状态的监控数据。用来评估系统压力。
sessionSub_total	/home/admin/logs/registry/session/registry-console.log	-	SessionServer 上 Sub 的数据统计，统计每台机器上的 Sub 总量，反应了目前 SessionServer 上的数据数量，是一个运行状态的监控数据。用来评估系统压力。
session_connect_total	/home/admin/logs/registry/session/registry-console.log	-	SessionServer 上 connect 的数据统计，统计每台机器上的 connect 总量，反应了目前 SessionServer 上的数据数量，是一个运行状态的监控数据。用来评估系统压力。
sessionConDataFailed_Prod	/home/admin/logs/registry/session/common-error.log	最近 5 分钟 > 0	SessionServer 上链接 DataServer 失败的，需要确认失败原因。
sessionConMetaFailed_Prod	/home/admin/logs/registry/session/common-error.log	最近 5 分钟 > 0	SessionServer 上链接 MetaServer 失败，需要确认失败原因。
sessionRefreshLeader_fail_prod	/home/admin/logs/registry/session/common-error.log	最近 3 分钟持续 > 0	SessionServer 上刷新 MetaServer 的 leader 节点失败，需要确认 MetaServer 是在进行 raft 选主，还是有网络问题。

监控项	日志路径	报警配置	说明
sessionRenew_fail_prod	/home/admin/logs/registry/session/common-error.log	最近 2 分钟 > 0	SessionServer 上 renew 数据失败，需要检查和 DataServer 之间的日志，确认是否存在问题。
sessionSyncConsoleError_prod	/home/admin/logs/registry/session/common-error.log	最近 60 分钟 > 200	SessionServer 上同步给 dsrconsole 数据异常，需要看 9601 端口时候是否有监听，监听是否正常，是否有 dsrconsole 的链接进来。 一般短时间报错没有影响。
sessions_deleteButNotExist_prod	/home/admin/logs/registry/session/common-error.log	-	SessionServer 上清理客户端链接数据，但是数据已经被清理了。 短时间没有问题。
session_ClientOff_notifyDataFail_prod	/home/admin/logs/registry/session/common-error.log	最近 30 分钟求和 > 10	客户端断连，SessionServer 通知 DataServer 失败。
session_error_prod	/home/admin/logs/registry/session/common-error.log	最近 10 分钟 > 200	SessionServer 排除了一些单独监控项以外的 error 报警。
session_pub_notifyDataFail_prod	/home/admin/logs/registry/session/common-error.log	最近 30 分钟求和 > 0	SessionServer 上有 Pub 信息同步给 DataServer 失败，需要确认失败的原因。
session_push_error_prod	/home/admin/logs/registry/session/common-push-error.log	最近 5 分钟 > 200	SessionServer 下推 Sub 信息失败，一般是因为客户端有订阅，但是已经断开了连接。当前存在处理上的时间差。
session_receive_sub_check_data_finally_giveUp(missing_client_data)(no_servers)_prod		当前值 > 0	

监控项	日志路径	报警配置	说明
	/home/admin/logs/registry/session/common-default.log		SessionServer 下推 Sub 信息失败，因为数据参数异常解析。
session_receive_sub_check_data_giveUp_prod			
session_setZeroByPushExceed_prod	/home/admin/logs/registry/session/registry-push.log	最近 60 分钟求和 > 50	SessionServer 下推 Sub 信息失败次数超过限制。
session_setZeroBySubFail_prod	/home/admin/logs/registry/session/common-error.log	最近 60 分钟求和 > 0	SessionServer 从 cache 获取信息失败，将数据的 version 更新为 0。
鉴权失败	/home/admin/logs/registry/session/registry-session.log	最近 1 分钟 > 10	SessionServer 执行鉴权失败。 专有云环境，鉴权通常都是关闭的。

2.3.5.2. 报警处理

分类	指标（关键字）	报警阈值	采集间隔	报警间隔	描述	如何处理报警
CPU 使用情况监控	cpu_usage	> 90%	1 分钟	3 分钟	CPU 使用情况监控	扩展容器。
内存使用监控	mem_usage	> 90%	1 分钟	3 分钟	内存使用情况监控	展开或重新启动容器。
磁盘使用情况监控	disk_usage	> 90%	1 分钟	3 分钟	磁盘使用情况监控	清除日志目录中无效的日志文件。
端口监测	8080	It doesn't	1 分钟	3 分钟	端口监控、警告	重新启动或更换容器。

2.3.6. 个性化变更

2.3.6.1. 预置操作

> 注意
>
> 所有变更都需要预先执行。

1. 通过 WebShell 登录到任何元服务器上的 Registry 服务器。
2. 在服务器上关闭推送，执行命令如下：

```
curl http://localhost:9615/stopPushDataSwitch/open
```

3. 验证状态是否真的关闭，验证命令如下：

```
curl http://localhost:9615/digest/pushSwitch
```

2.3.6.2. 新站点发布

1. 导入新 feature，通过 **云游** 部署升级。

> 注意
>
> - 所有服务器都必须在同一时间发布，即要在同一套版本中，因为元服务器（meta-server）是基于 raft 协议的。该协议要求大多数节点启动领头节点选举。
> - 在发布过程中，您需要遵循 meta_server > data_server > session_server 的发布顺序。
> - 如果在发布应用（例如数据服务器）期间计算机运行状况检查失败，则需要重试。当前应用发布成功后，下一个应用方可进入发布流程。
> - Metapushclose 和 metapush 都是关闭推送的一次性任务。在首次部署新站点时，如果元服务器不存在，则 metapushclose 无法发布，可以直接选择 **跳过**。

2. 监控正常后，执行 后置操作 打开推送。
3. 执行后续测试用例。

2.3.6.3. 版本升级

操作步骤

1. 导入新 feature，通过 **云游** 部署升级。
2. 执行 预置操作，然后关闭推送。
3. 发布升级。

> 注意
>
> - 所有服务器都必须在同一时间发布，即要在同一套版本中，因为 meta-server 是基于 raft 协议的。该协议要求大多数节点在同一时间启动领头节点选举。
> - 在发布过程中，您需要遵循“meta_server->data_server-> session_server”的发布顺序：如果在发布应用程序（例如数据服务器）期间计算机运行状况检查失败，则需要重试。当前应用发布成功后，下一个应用方可进入发布流程。

4. 观察监控系统的 pubs/subs 数量，恢复到发布前的水平。

   参考自定义监控中的 pub/sub/watch，并期望数量恢复到发布前的的 90% 到 100%，具体取决于群集大小，约需要 3 到 10 分钟。

5. 监控正常后，执行 后置操作，并打开推送。
6. 打开推送后，执行后续测试用例。

2.3.6.4. 后置操作

> 注意
>
> 完成所有更改后，必需执行此操作。

1. 通过 WebSHELL 登录到任何元服务器上的 Registry 服务器。
2. 在服务器上打开推送，执行命令如下：

```
curl http://localhost:9615/stopPushDataSwitch/close
```

3. 确认 Registry 是否为开启状态：

```
curl http://localhost:9615/digest/pushSwitch
```

2.3.6.5. 容器重启

重启 meta_server

1. 执行 预置操作 关闭推送。
2. 使用 k8s 原生命令重启容器。

   运行状况检查必须成功，不能跳过。

3. 执行 后置操作 打开推送。

重启 data_server

1. 执行 预置操作 关闭推送。
2. 使用 k8s 原生命令重启容器。

   状况检查必须成功，不能跳过。

3. 执行 后置操作 打开推送。

重启 session_server

1. 执行 预置操作 关闭推送。
2. 使用 k8s 原生命令重启容器。

   运行状况检查必须成功，无法跳过。
3. 执行 后置操作 打开推送。

2.3.6.6. 容器上线

一般不需要执行该操作。

2.3.7. 服务巡检

2.3.7.1. 系统组件监控检查

基础监控检查

登录 RMS 监控系统的 **应用监控** > **全部监控** 页面查找 SOFA Registry 对应的系统组件，确认其无异常告警。需要关注的监控项内容如下表所示。

应用	分类	指标（关键字）	告警阈值
registry-session registry-data registry-meta	基础监控-CPU 使用率监控	cpu_usage	>80%
	基础监控-内存使用率监控	mem_usage	>80%
	基础监控-磁盘使用率监控	disk_usage	>80%
	基础监控-load5	load5	>3
registry-session	基础监控-端口监控	9600	不通
registry-data	基础监控-端口监控	9620	不通
registry-meta	基础监控-端口监控	9610	不通

自定义业务监控检查

通过 RMS 内核态监控产品查看注册中心产品各项系统监控指标是否有异常告警产生。

如果有异常告警产生，联系技术支持进行排查。

2.3.7.2. 业务功能检查

SOFA Registry 服务验证

- 验证 meta 是否启动成功：

  登录 meta 节点容器后台，执行以下命令，检查本机健康接口。

  ```
  curl http://localhost:9615/health/check
  ```

- 验证 data 是否启动成功： 登录 data 节点容器后台，执行以下命令，检查本机健康接口。

  ```
  curl http://localhost:9622/health/check
  ```

- 验证 session 是否启动成功：

  登录 session 节点容器后台，执行以下命令，检查本机健康接口。

  ```
  curl http://localhost:9603/health/check
  ```

业务验证（即验证业务方应用）

检查应用所使用的 RPC 是否能正常调用成功即可。

2.4. 常见故障处理

2.4.1. 服务注册中心日常问题

服务注册中心 SessionServer 节点宕机

服务注册中心 SessionServer 部分节点宕机不影响集群稳定性，仅需关注宕机后替换故障节点即可。

服务注册中心 DataServer 节点宕机

- 服务注册中心 DataServer 节点需 1/2 以上存活来保障服务可用性，部分节点宕机集群可自动剔除故障节点并转移数据，但需要确保有 1/2 以上可用节点存活。
- Data 节点扩容：可以逐个加入新机器达到 working 状态。

服务注册中心 MetaServer 节点宕机

- 如果是物理机底座（AntStack Plus AKE）的交付环境，不需要人工干预，MetaServer 宕机后会自动恢复。
- 如果是飞天底座（ACK 容器集群）的交付环境，需要人工恢复。操作步骤如下：
  - 当 Registry 版本低于 2.9.0，且 MetaServer 单机宕机时，需确定 IP 是否变化。若 IP 未变化，则无需处理；若变化则需要 changepeer。步骤如下：

a. 在重启的 MetaServer 上，通过 ifconfig 命令查看本机 IP。

```
[root@zj                                    r-0 /root]
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.106.6.206  netmask 255.255.255.255  broadcast 0.0.0.0
        ether ae:90:20:14:8c:22  txqueuelen 0  (Ethernet)
        RX packets 69061955  bytes 10667725536 (9.9 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 84039570  bytes 23674945295 (22.0 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 9744991  bytes 5123607368 (4.7 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9744991  bytes 5123607368 (4.7 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

b. 执行以下命令，通过 querypeer 查询目前的 IP 信息：

```
curl http://localhost:9615/manage/queryPeer
```

输出示例如下：

```
["10.106.6.214","10.106.9.242","10.106.6.207"]
```

c. 通过 `ifconfig` 命令查询另外两个 MetaServer 的 IP。

```
# ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.106.9.242  netmask 255.255.255.255  broadcast 0.0.0.0
        ether 22:bb:41:6b:51:5f  txqueuelen 0  (Ethernet)
        RX packets 54215465  bytes 10467903916 (9.7 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 28050507  bytes 3749076283 (3.4 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 626362  bytes 40794515 (38.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 626362  bytes 40794515 (38.9 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
# ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.106.6.214  netmask 255.255.255.255  broadcast 0.0.0.0
        ether 4e:7e:3d:43:0a:12  txqueuelen 0  (Ethernet)
        RX packets 55979250  bytes 10655963432 (9.9 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 29098999  bytes 3877014600 (3.6 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 169421  bytes 11030037 (10.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 169421  bytes 11030037 (10.5 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

d. 比较 IP 和 queryPeer 结果，确认重启的 MetaServer 的原 IP。

以上示例中，MetaServer 的 IP 由 10.106.6.207 变为了 10.106.6.206。

e. 进行 changepeer。

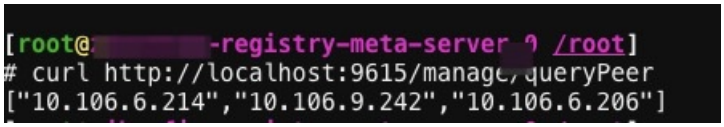
```
curl -XPOST --data ipAddressList="<更新后的 IP 列表>" localhost:9615/manage/changePe
er
```

示例如下：

```
curl -XPOST --data ipAddressList="10.106.6.214,10.106.9.242,10.106.6.206" localhost
:9615/manage/changePeer
```

输出 `{"success":true,"message":null}` 说明变更成功。

f. 再次执行 querypeer，确认变更成功。



g. 执行健康检查。

```
 curl http://localhost:9615/health/check
{"success":true,"message":"MetaServerBoot sessionRegisterServer:true, dataRegisterS
erverStart:true, otherMetaRegisterServerStart:true, httpServerStart:true, raftServe
rStart:true, raftClientStart:true, raftManagerStart:true, raftStatus:Leader"}
```

- 当 Registry 版本高于 2.9.0，且 MetaSever 单机宕机，一般不用处理。
- 当 Registry 版本高于 2.9.0，且 MetaSever 一次宕机了 2 台以上（超过了半数的机器），如果超过 2 个 IP 都变化了，需要删除 raftdata 数据后重新导入数据。

  删除 raftdata 数据详细步骤如下：

  a. （建议）关闭 SessionServer 的推送。

     确认 SessionServer 的推送状态，条件允许的情况下，建议关闭 SessionServer 的推送。

  b. 确认 MetaServer 的进程都已经关闭。

  c. 进入到 Meta 容器中，删除 `/home/admin/raftData/` 目录下所有内容。

  d. 重新启动 MetaServer。

  重新导入数据的步骤如下：

  a. 从 Dsrconsole 上获取还在生效的数据。

```
mysql -h<DB 地址> -u<username> -p<password> -D dsrconsole -e "select config.*,info.
app_name from service_config as config, service_info as info where config.status=0
and info.instance_id=config.instance_id and info.data_id=config.data_id"
```

     DB 地址、username、passwd 需要替换成您实际环境的信息。若获取到数据，则进行以下步骤；若无数据，则无需再导入。

b. 解析获取结果。

```
// 导出 weight 为 null 的数据到 dsrconsole_no 中。
mysql -h<DB 地址> -u<username> -p<password> -D dsrconsole -e "select config.*,info.
app_name from service_config as config, service_info as info where config.status=0
and info.instance_id=config.instance_id and info.data_id=config.data_id and config.
weight is null" > dsrconsole_no

// 生成执行脚本。metaIP 需替换成实际 MetaServer 的 IP。
cat dsrconsole_no | awk '{print "curl -X POST -H \"Content-Type: application/json\"
-d '\''{\"dataId\":\""$3"\", \"group\":\"SOFA.CONFIG\", \"instanceId\":\""$2"\", \"
version\":\"1111111111121\", \"data\":\"override://"$4":"$5"/"$13"\?disabled=true\"
}'\'' -vvv  <metaIp>:9615/persistentData/put"}' > dsrconsole_no_weight_shell

// 导出 weight 不为 null 的数据到 dsrconsole_no 中。
mysql -h<DB 地址> -u<username> -p<password> -D dsrconsole -e "select config.*,info.
app_name from service_config as config, service_info as info where config.status=0
and info.instance_id=config.instance_id and info.data_id=config.data_id and config.
weight is not null" > dsrconsole_no

// 生成执行脚本。metaIP 需替换成实际 MetaServer 的 IP。
cat dsrconsole_no | awk '{print "curl -X POST -H \"Content-Type: application/json\"
-d '\''{\"dataId\":\""$3"\", \"group\":\"SOFA.CONFIG\", \"instanceId\":\""$2"\", \"
version\":\"1111111111121\", \"data\":\"override://"$4":"$5"/"$13"\?disabled=true\&
weight="$7"\"}'\'' -vvv <metaIp>:9615/persistentData/put"}' > > dsrconsole_weight_s
hell
```

c. 执行以下命令导入脚本：

```
bash dsrconsole_no_weight_shell
bash dsrconsole_weight_shell
```

动态配置节点宕机

- 动态配置部分节点宕机无实际影响，客户端会主动重试连接可用节点。
- 重启或扩容后，需确保服务启动成功，数据源连接正确，端口 9880 存活。

2.4.2. 注册中心 MetaServer 选主异常导致大量报错日志

问题现象

注册中心 SessionServer 出现大量报错日志：

排查步骤

1. 检查 MetaServer 日志，发现如下报错：

   ```
   refresh leader failed...

   send notify leader change error...
   ```

   判断 MetaServer 出现选主异常，导致 SessionServer 异常报错。

2. 在所有 MetaServer 容器上执行如下命令：

   ```
   curl http://localhost:9615/health/check
   ```

   发现所有容器都是 follower，没有 leader，确定 MetaServer 选主异常。

解决方案

重启 MetaServer。