彩虹桥架构演进之路–负载均衡篇 | 得物技术

原创 新一 得物技术 2024年11月13日 18:30 上海

目录

一、前言
二、背景
三、核心名称解释
四、现有架构回顾
1. 现有架构
2. 主要痛点
五、自建元数据中心&SDK 增强
1. 架构详解
2. 容灾能力
3. 一些思考
六、总结

一 前言

一年一更的彩虹桥系列又来了，在前面两期我们分享了在稳定性和性能2个层面的一些演进&优化思路。近期我们针对彩虹桥 Proxy 负载均衡层面的架构做了一次升级，目前新架构已经部署完成，生产环境正在逐步升级中，借此机会更新一下彩虹桥架构演进之路系列的第三篇。

阅读本文预计需要20～30分钟，建议不熟悉彩虹桥的同学在阅读本文前，可以先看一下前两篇彩虹桥架构演进的文章：
得物数据库中间件平台“彩虹桥”演进之路
彩虹桥架构演进之路–性能篇 | 得物技术

二 背景

彩虹桥目前依赖 SLB 做负载均衡和节点发现，随着业务发展流量越来越高，SLB 带宽瓶颈逐渐暴露，虽然在半年前做过一次双 SLB 改造临时解决了带宽瓶颈，但运维成本也随之变高。除了带宽瓶颈外，SLB 无法支持同区优先访问，导致难以适配双活架构。所以准备去除彩虹桥对 SLB 的强依赖，自建彩虹桥元数据中心，提供负载均衡和节点发现等能力，同时支持同区访问等能力来更好的适配双活架构。下面会详细介绍一下彩虹桥元数据中心以及 SDK 相关能力的相关细节。

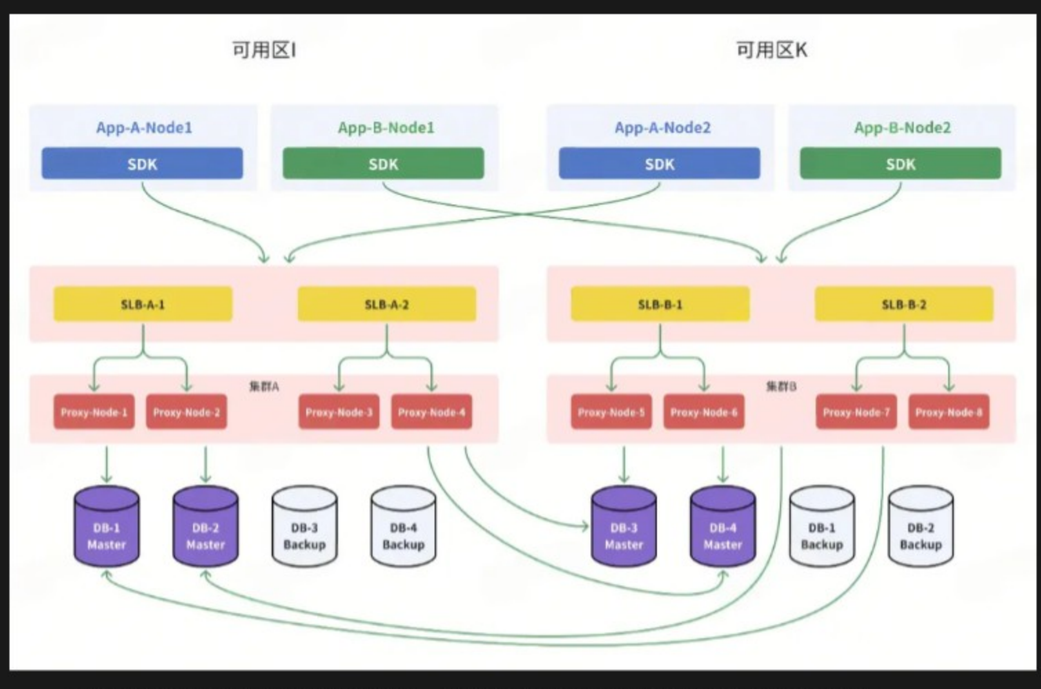
三 核心名称解释

名词	解释
Proxy	彩虹桥代理节点，独立进程部署
Rainbow	自研连接池 SDK，集成在业务服务
SLB	阿里云负载均衡产品，彩虹桥所用 SLB 为 4 层负载均衡
可用区	不同可用区分别代表不同的物理机房
bifrost-admin	彩虹桥管理后台（控制流）
MetaCenter（新增）	元数据中心，负责计算 Proxy 可用节点列表以及提供 Proxy 可用节点列表查询 OpenAPI

四 现有架构回顾

在开始介绍彩虹桥元数据中心之前，我们先来回顾一下彩虹桥目前架构，以及存在的一些痛点。

现有架构



- 业务服务集成 SDK 通过域名访问，请求经过 SLB 转发到具体的 Proxy 节点。
- 每个集群挂载双 SLB，SDK 通过 DNS 解析轮训路由到2个 SLB，2个 SLB 挂载不同的后端节点。
- 每个集群部署的 Proxy 节点均为一个可用区，双活架构为集群维度多可用区部署。
- 业务侧大多数为多可用区混布，单同一个逻辑库只会连接一个彩虹桥集群，由于彩虹桥一个集群内的节点均为同一可用区，所以业务服务–彩虹桥这条链路必然会出现一半节点跨区访

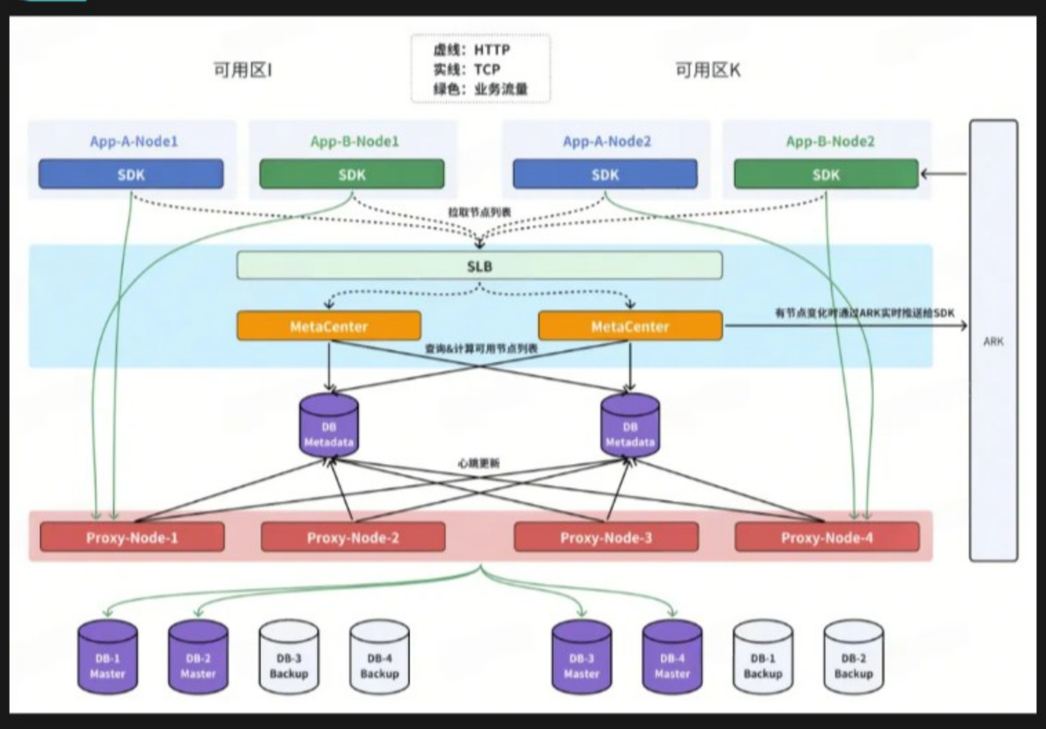
问。

- 彩虹桥集群按照业务域划分，彩虹桥集群所属业务域的 RDS 大多数都会跟彩虹桥集群同区。比如彩虹桥交易集群为i区，归属交易集群的逻辑库挂载的 RDS 大多数也都是i区。

主要痛点

- SLB 带宽已达瓶颈（5Gb/s，历史上出现过多次 SLB 带宽达到 100%的情况），目前彩虹桥单集群挂载了双 SLB 暂时解决带宽瓶颈但仍存在痛点：
  1. SLB 扩容流程较复杂（配置监听、配置虚拟服务器组、监听绑定虚拟服务组，配置调度算法、更新域名解析的等），基于目前发布系统能力无法实现全自动化。根据之前混沌工程演练结果，SLB 扩容流程需要30分钟左右。
  2. SLB 扩容后，需要改域名解析，DNS 解析生效需要一段时间（域名 TTL 1 分钟，本地缓存10分钟），新 SLB 需要10分钟左右才开始逐渐承载流量，无法实现 SLB 快速扩容。
- 单可用区故障时，需要人工操作切流到其他可用区集群，SLA 难以保证（目前无法自动化判定单可用区故障，且集群级别流量调度需要人工预估集群负载，难以实现自动化切流）。
- SLB 目前支持最低权重为1/100，粒度较粗，无法支撑发布过程中的更小流量灰度需求。
- Proxy 单个集群所有节点均为同一个 AZ，需要与下游 RDS 保证同 AZ，跨集群流量调度灵活性差，很难实现多可用区流量均衡（目前由于大部分 RDS 为 I 区，Proxy 多可用区流量非常不均衡：i区 90%/k 区流量 10%）。

五 自建元数据中心&SDK 增强



元数据中心独立部署

- 新增 Metadata 数据库，多可用区部署（需要跟集群中的 Proxy 同区）。
- 新增 MetaCenter 服务，多可用区部署。
- Proxy 连接所有 Metadata 数据库，注册&心跳都会写入到所有数据库。
- MetaCenter 服务定时查询所有 metadata 数据库，基于心跳版本号和多个数据库的并集筛选出健康的节点列表存储到内存中。
- MetaCenter 服务提供 API，查询 MetaCenter 内存中的可用节点列表数据。
- SDK 启动时会去通过7层 SLB 访问 MetaCenter 提供的 API 拉取节点列表并存储到内存，运行中每隔 5s 更新一次。
- MetaCenter 每次计算时如果有节点下线，通过 ARK 实时下发拉取事件给 SDK，SDK 会立刻重新拉取一次节点列表。
- SDK 通过下发的节点列表做负载均衡，优先路由到同可用区的 Proxy 节点，其次按照节点权重轮训。
- SDK 轮训间隔时间和节点变更事件下发开关均为可配置，实时生效。

架构详解

Metadata 数据库

node_info		
PK	id	bigint
	cluster_name	varchar
	address	varchar
	beat_version	bigint
	config_version	bigint
	weight	int
	metadata	varchar
	enabled	tinyint(1)

节点表结构设计

- beat_version：心跳版本号，只有上报心跳时会更新。
- config_version：配置版本号，更新权重&状态时会更新。
- enabled：是否启用

Proxy

节点启动时

- 注册：启动时会去所有 metadata 数据库注册当前节点，如果 node_info 不存在对应节点记录，则新增，如果存在则修改权重为初始权重。
- 启动完成后需要调用 bifrost-admin 提供的调用节点启用 API（发布脚本）

```
update node_info
set weight = 1, config_version = #{config_version}
where cluster_name = ? and address = ?
```

节点运行时

- 心跳：定时更新所有 metadata 数据库节点记录的 beat_version 字段

```
update node_info set beat_version = beat_version + 1
where cluster_name = ? and address = ?
```

节点下线

- 调用 bifrost-admin 提供的下线 OPEN API（发布脚本）

MetaCenter(Heimdall)

- 启动时

初始化心跳版本号：记录所有 metadata 数据库每个节点最新 beat_version 和初始化心跳丢失次数到内存

i 区，	beat_version	心跳丢失次数
1.2.3.10	149	0
1.2.3.20	182	0

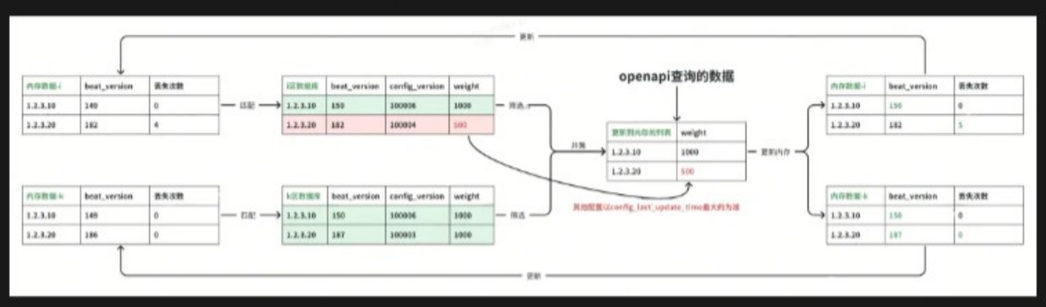
k 区	beat_version	心跳丢失次数
1.2.3.10	149	0
1.2.3.20	182	0

- 运行时

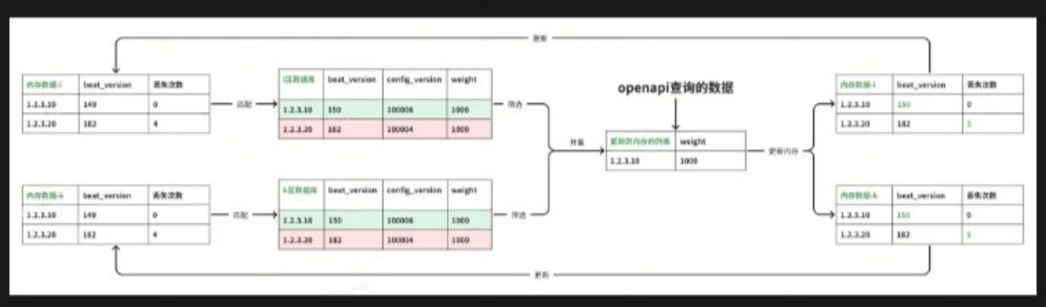
定时查询节点信息（3s 一次），筛选可用节点并写入到内存中，提供 OpenAPI 给 SDK 调用，每个库均执行以下操作，最终会得到每个库的可用节点列表，最后把多个 list 求并集，得到最终的可用列表，写入到内存中。

查出所有列表数据后，对比内存中的 beat_version 与数据库中的 beat_version，如不相同则更新内存，如果相同说明对应节点心跳有丢失，如果丢失次数超过阈值，则剔除此节点。

节点列表中除了 ip、端口信息外，还有权重，启用状态属性， 这些属性都属于控制流变更，如果出现2边数据库不一致场景，以 config_version 最大的为准。



1.2.3.20节点与K区网络断开

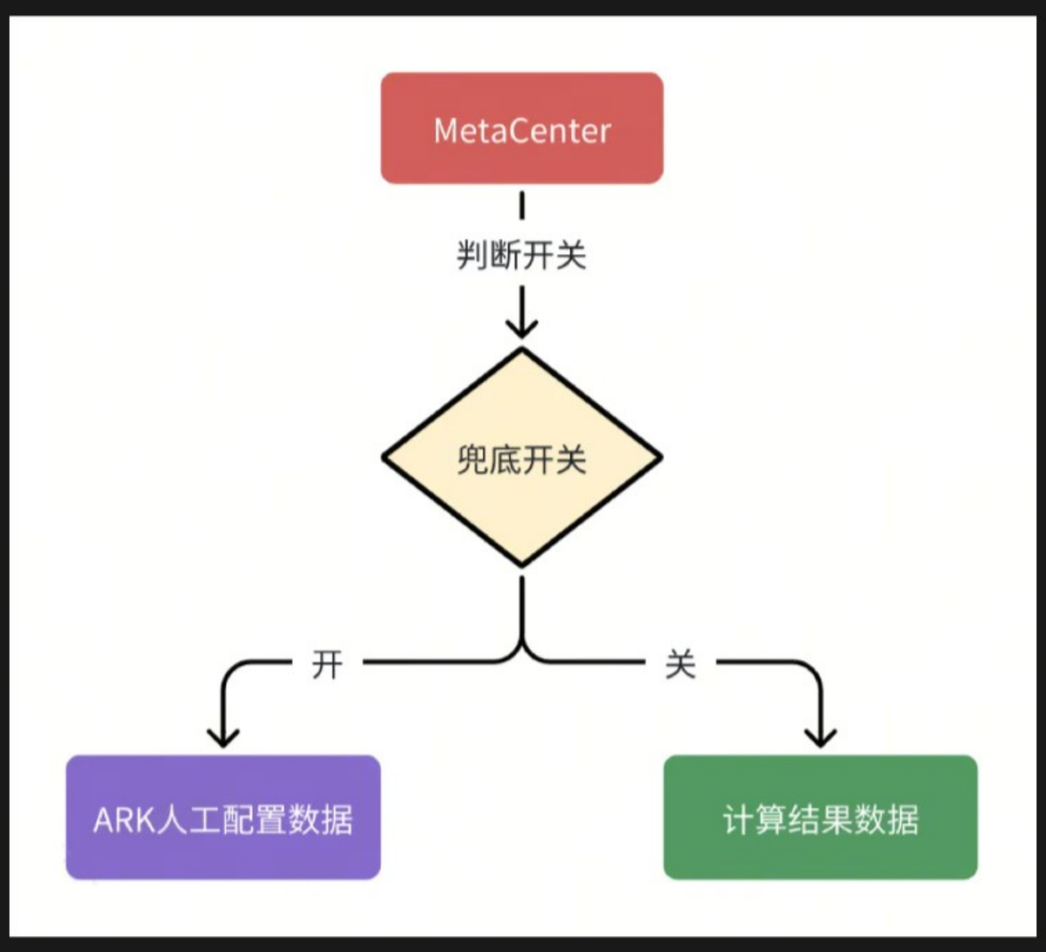


1.2.3.20节点宕机

如果本次计算时有节点列表变化，会下发一个变更事件到 ARK（value 为时间戳-秒），SDK 在收到次配置变更后会立刻到 MetaCenter 拉取一次节点列表，以弥补定时轮训的延时。

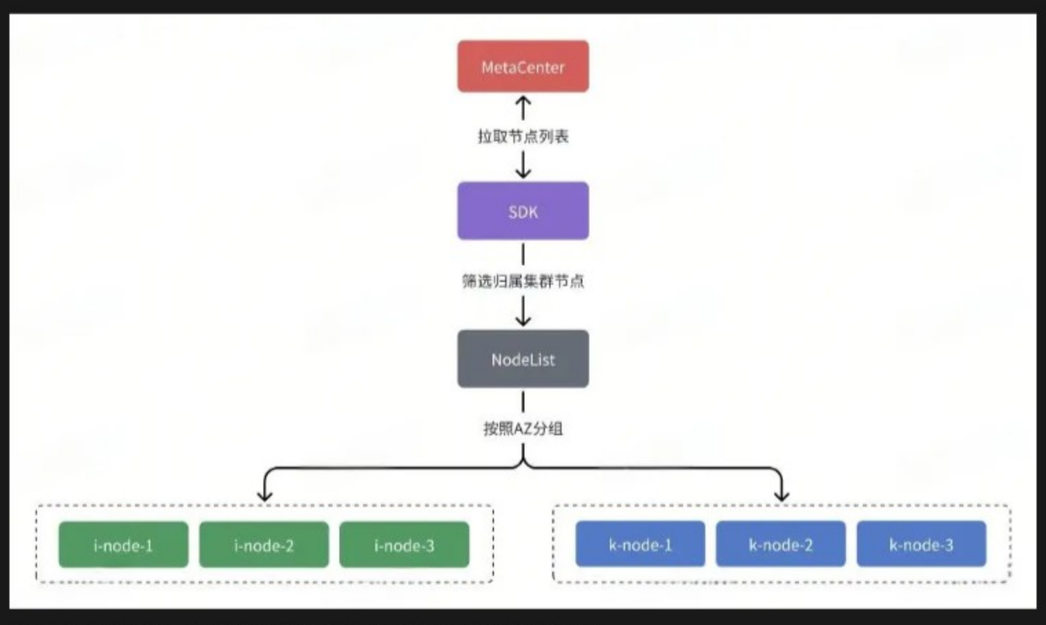
- 兜底配置

MetaCenter 提供的 OpenAPI 是通过计算后存入内存的数据，为了可以人工干预节点列表，需要支持开关一键切换至人工配置的节点列表数据。



SDK(Rainbow)

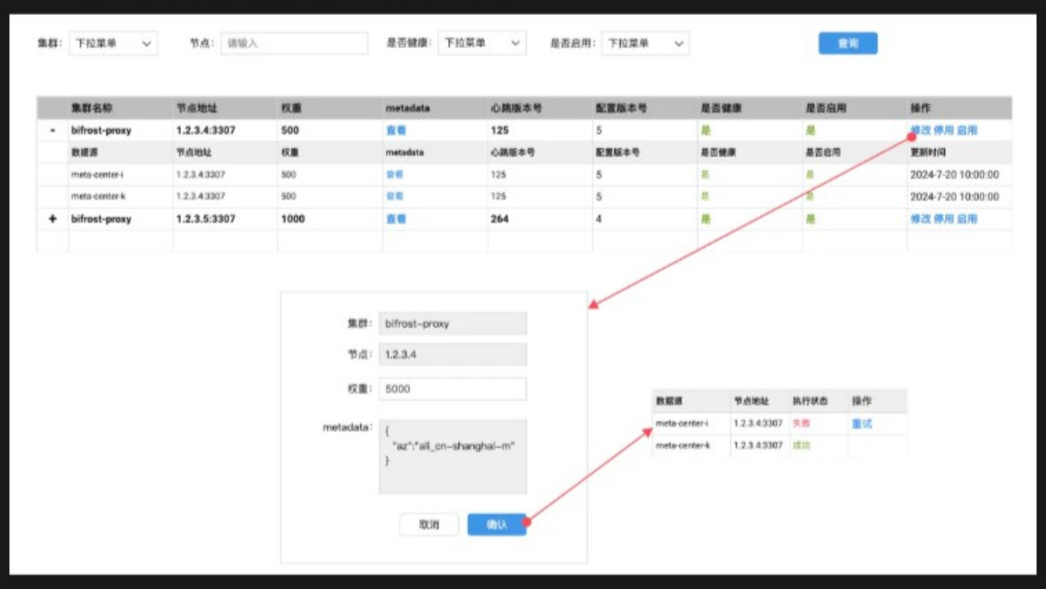
- SDK 启动时会去通过7层 SLB 拉取节点列表并存储到内存，运行中每隔5s更新一次。
  如果拉取失败，启动时报错，运行中不做任何处理，等待下次拉取。
  如果拉取的可用节点列表为空，启动时报错，运行时兜底不做任何处理，等待下次拉取。
- 拉取的可用节点列表与内存中做对比，如果有节点被移除，需要优雅关闭对应的存量连接（如果被移除节点超过1个，则不做驱逐）。
  当可用节点数量/所有节点数量 < X%时，忽略本次变更，不更新内存中的可用节点列表。
- 拉取的节点数据会按照可用区进行分组，分为同可用区&跨可用区2个队列
  负载均衡时优先从同 AZ 节点队列中进行加权轮训。
  当同AZ节点权重总和/所有节点权重总和 < Y%时，同 AZ 节点优先策略失效，退化为所有节点加权轮训。
  当同AZ可用节点 < Z时，同 AZ 节点优先策略失效，退化为所有节点加权轮训 。
- 需要新增查询节点列表的监控埋点&以上三种计算结果的埋点



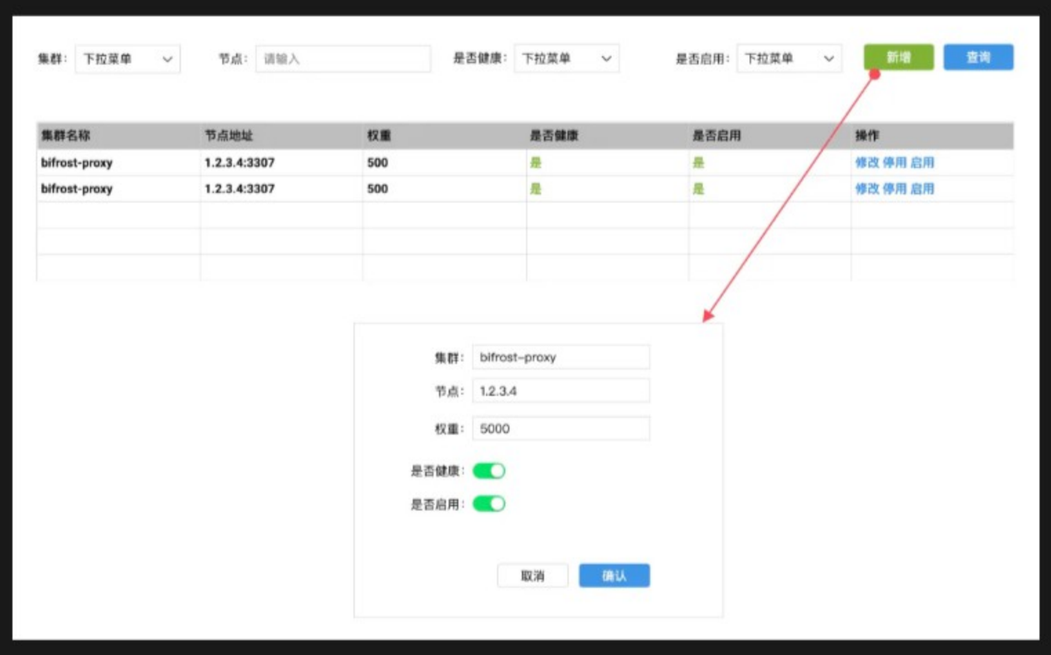
另外 SDK 支持一键动态切换至走老架构方式（4层 SLB）

管理后台

- 新增页面【节点管理】，用于查询&管理节点



- 新增页面【兜底节点管理】，用于管理兜底节点列表。



- 提供节点上下线 API，给发布系统调用。

修改状态会去所有 metadata 数据库执行，只有一个库成功就返回成功，如所有库都修改失败，则返回失败。

```
update node_info
set enabled = 0, config_version = #{config_version}
where ip = ? and port = ?
```

容灾能力

表格中的是否有影响和故障恢复时间均指 SDK-Proxy 的访问链路，Proxy-DB 链路不在范围内。

case/影响	是否有影响	故障恢复时间	备注
单个 MetaCenter 宕机	无	0	多可用区部署不影响
单个 Metadata 数据库宕机	无	0	双可用区部署，单个 metadata 数据库宕机无影响
跨区网络中断	无	0	SDK-Proxy 不受影响，但如果 Proxy-DB 跨区会受影响
单个 Proxy 宕机	有	20s	20s 内客户端感知并主动剔除存量连接
可用区全局故障	有	30s	彩虹桥 20s 内恢复完成，依赖 RDS 切换时间 30s

- 可用区i全部宕机举例

参考以下时间线，可在30s左右完成恢复。

- i区 Metadata 数据库故障，无影响。

一些思考

Q：为什么不用 sylas（得物注册中心产品）做注册中心，而是要自建元数据中心做服务发现？
彩虹桥和 sylas 均为 P0 级别服务，对稳定性要求极高，在架构设计之初需要充分考虑到互相依赖可能带来的级联故障，在与注册中心相关同学沟通后，决定自建彩虹桥元数据中心，实现自闭环。

Q：为什么不是传统的基于 Raft 协议的三节点来实现服务发现，而是用多套数据源做 merge？
Raft 是工程上使用较为广泛的强一致性、去中心化、高可用的共识算法，在分布式系统中，适用于高一致性、容错性要求高的场景。但 Raft 协议需要维护领导者选举和日志复制等机制，性能开销较大，其次 Raft 协议相对复杂，在开发、维护、排障等方面会非常困难，反之采用多数据源求并集的方式更简单，同时也具备单节点故障、整个可用区故障以及跨区网络中断等多种复杂故障下的容灾能力。

Q：如何在 SLB 切换到新架构的过程中保障稳定性？
可灰度：支持单个上游节点粒度的灰度
可回滚：支持一键动态切换至 SLB 架构
可观测：大量埋点数据可实时进行观测，有问题可快速回滚。

六 总结

自建元数据中心后，将给彩虹桥带来一系列收益：

- 应用服务通过 SDK 直接连接 Proxy 节点，摆脱了对 SLB 的依赖，解决了带宽瓶颈和额外网络开销问题，并提高了流量灰度控制的精细度。
- 简化了扩容流程，扩容时只需增加 Proxy 节点大大缩短整个扩容时间。
- 多可用区容灾实现自动故障转移，无需人工干预。
- SDK 具备了同 AZ 路由能力，更好适配双活架构。

往 期 回 顾

1.解析Go切片：为何按值传递时会发生改变？| 得物技术
2.得物精准测试平台设计与实现
3.基于IM场景下的Wasm初探：提升Web应用性能 | 得物技术
4.实时特征框架的生产实践 | 得物技术
5.增长在流量规则巡检的探索实践 | 得物技术

文 / 新一

“

扫码添加小助手微信

如有任何疑问，或想要了解更多技术资讯，请添加小助手微信：

彩虹桥 2 # 负载均衡 1 # 架构演进 1 # 元数据中心 2

彩虹桥 · 目录 ≡

‹ 上一篇 · 得物数据库中间件平台“彩虹桥”演进之路