Discovery【探索】

云原生微服务
开源解决方案

http://www.nepxion.com
https://github.com/Nepxion

Let us start Discovery

Discovery【探索】

01 总体概述

首席作者简介

- Nepxion开源社区创始人
- 2020年阿里巴巴中国云原生峰会出品人
- 2020年被Nacos和Spring Cloud Alibaba纳入相关开源项目
- 2021年阿里巴巴技术峰会上海站演讲嘉宾
- 2021年荣获陆奇博士主持的奇绩资本，进行风险投资的关注和调研
- 2021年入选Gitee最有价值开源项目
- 2024年入围中国开源创新榜候选项目
- 阿里巴巴官方书籍《Nacos架构与原理》作者之一
- Spring Cloud Alibaba Steering Committee Member、Nacos Group Member
- Spring Cloud Alibaba、Nacos、Sentinel、OpenTracing Committer & Contributor

Discovery Wiki http://nepxion.com/discovery
Polaris Wiki http://nepxion.com/polaris

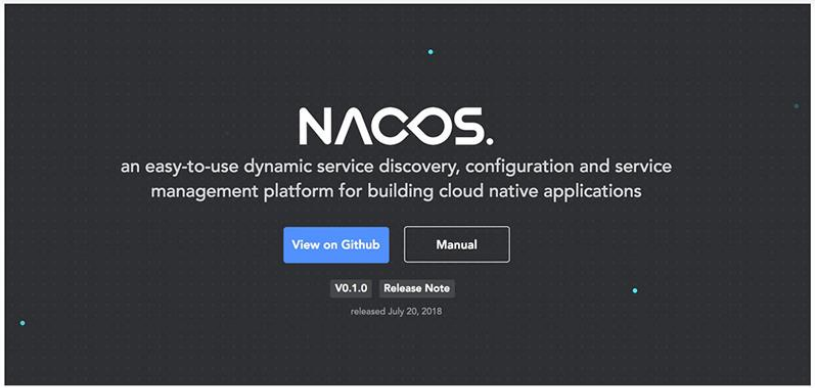
01 总体概述

成就荣誉


GVP — Gitee最有价值开源项目
Gitee Most Valuable Project
祝贺您的开源项目
Discovery
https://gitee.com/nepxion/Discovery/
入选 2021 年 GVP
经审定特颁发本证书，以资鼓励
OSCHINA
Gitee
2021.5.1
DATE
SIGNATURE

01 总体概述

成就荣誉



github项目地址在 这里

更多与 Nacos 相关的开源项目信息

- Nacos
- Dubbo Registry Nacos
- Nacos DNS-F
- Nacos Docker
- Nacos Spring Project
- Nacos Spring Boot
- Spring Cloud Alibaba
- Dubbo
- Sentinel
- Spring Cloud
- Nepxion Discovery
- Spring Cloud Gateway Nacos

01 总体概述

发展历程

诞生故事

时间轴

- 2017年12月开始筹划
- 2018年03月开始编码
- 2018年06月在GitHub开源
- 2018年06月发布v1.0.0，支持Camden版
- 2018年06月发布v2.0.0，支持Dalston版
- 2018年07月发布v3.0.0，支持Edgware版
- 2018年07月发布v4.0.0，支持Finchley版
- 2019年04月发布v5.0.0，支持Greenwich版
- 2020年04月发布v6.0.0，支持Hoxton版
- 2021年04月完成v7.0.0，支持2020版
- 2022年04月完成v8.0.0，支持2021版
- 2023年01月完成v9.0.0，支持2022版
- 2024年03月完成v10.0.0，支持2023版

迭代演进

截至2022年12月

- 主工程超25万行全部代码撰写量
- 2个主工程，6个卫星工程
- 10个大版本迭代
- 超4500个Commits
- 超130个Releases
- 超6000个Stars（GitHub + Gitee）
- 超1300个Forks
- 超80篇Wiki文档，超10万字技术文档

技术特征

基于Spring Cloud技术栈，集成如下

- 1个中间件生态圈：Spring Cloud Alibaba
- 4个注册中心：Nacos、Eureka、Consul、Zookeeper
- 6个配置中心：Apollo、Nacos、Redis、Zookeeper、Consul、Etcd
- 2个网关：Spring Cloud Gateway、Zuul
- 3个限流熔断降级组件：Sentinel、Hystrix、Resilience4J
- 3个调用链监控中间件：SkyWalking 、Jaeger、Zipkin（基于OpenTracing和OpenTelemetry规范）
- 2个日志监控中间件：Elastic Search、Kibana
- 3个指标监控中间件：Prometheus、Grafana、Spring Boot Admin

质量控制

严格的代码质量控制

- 全链路自动化测试，90个双网关全链路组合式自动化测试用例
- 全链路压力性能测试
- 全链路自动化模拟流程测试，面向测试环境
- 全链路自动化流量侦测测试，面向生产环境

01 总体概述

发展历程

感谢支持

- 感谢阿里巴巴中间件Nacos、Sentinel和Spring Cloud Alibaba团队，尤其是Nacos负责人@彦林、@于怀，Sentinel负责人@宿何、@子衿，Spring Cloud Alibaba负责人@铖朴、@良名、@小马哥、@洛夜、@亦盏的技术支持。本框架成为阿里巴巴中间件Nacos和Spring Cloud Alibaba项目的相关开源
- 感谢携程Apollo团队，尤其是@宋顺的技术支持
- 感谢所有Committers和Contributors

04

社区建设

截止2022年12月

- 5个开源技术交流群，近2000人
- 近20个核心Committers提交Pull Requests

02

业界落地

截止2022年12月

- 不完全统计，超160家公司，包括银行、保险、房地产以及互联网等公司使用及调研

03

版本发布

现有和未来的版本计划

- 10.x.x版本（适用于2023.x.x）将继续维护
- 9.x.x版本（适用于2022.x.x）将继续维护
- 8.x.x版本（适用于2021.x.x）将继续维护
- 7.x.x版本（适用于2020.x.x）将继续维护
- 6.x.x版本（同时适用于Finchley、Greenwich和Hoxton）将继续维护
- 5.x.x版本（适用于Greenwich）已废弃
- 4.x.x版本（适用于Finchley）已废弃
- 3.x.x版本（适用于Edgware）不维护，但可用，强烈建议升级
- 2.x.x版本（适用于Dalston）已废弃
- 1.x.x版本（适用于Camden）已废弃

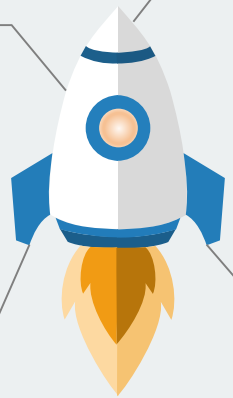
01

01 总体概述

功能全景

基础架构

- Spring Cloud & Spring Cloud Alibaba技术栈
- Discovery服务注册发现增强
- Ribbon & Spring Cloud LoadBalancer负载均衡增强
- Spring Cloud Gateway & Zuul动态路由过滤增强
- Feign & RestTemplate & WebClient调用增强



扩展功能

- 全链路服务无损下线的黑名单屏蔽（基于全局唯一ID、IP地址和端口）
- 网关动态路由（基于Spring Cloud Gateway和Zuul）
- 异步场景支撑（异步跨线程Agent插件、Hystrix线程池隔离插件）
- 注册发现隔离和准入
- Sentinel防护（限流、熔断、降级、授权、自定义和组合式防护）
- 蓝绿灰度埋点和熔断埋点监控（调用链追踪、日志监控、指标监控、告警监控、事件监听）
- 统一配置订阅功能（屏蔽六大配置中心细节）
- 各种模块用户自定义扩展

核心能力

- 全链路蓝绿灰度发布（基于版本、区域、IP地址和端口）
- 全链路路由手工编排、智能编排、无编排蓝绿灰度发布
- 全链路路由隔离（基于环境、可用区亲和性、IP地址和端口、组）
- 全链路路由偏好（基于版本）、路由调试（基于区域）
- 全链路故障转移（基于版本、区域、环境、可用区、IP地址和端口）
- 流量染色（基于配置文件、Git插件、启动参数、系统参数、环境装载）
- 同城 & 异地多活单元化（基于区域）

卫星模块

- DevOps运维平台对接Open API和流程，实现自动化蓝绿灰度发布
- 配置中心、控制台Open API、可视化界面规则策略推送
- 蓝绿灰度可视化服务治理、发布编排建模、流量侦测
- 全链路自动化模拟流程测试、全链路自动化流量侦测测试

01 总体概述

功能全景

部署实施

- 单网关、域网关、非域网关、全局订阅部署
- 全链路网关、服务端到端实施
- 全链路蓝绿灰度发布混合实施，单节点蓝绿灰度发布混合实施
- 条件驱动（Header、Parameter、Cookie、域名、RPC Method）、非条件驱动
- 内置参数（Header）模式下定时Job服务蓝绿灰度

规则策略

- 远程配置与本地配置，远程优先于本地
- 局部配置与全局配置，局部优先于全局
- 规则策略配置与业务配置的分离

触发方式

- 前端触发后端，网关触发服务，服务触发服务（该服务之后的所有服务）蓝绿灰度发布
- 自定义网关、服务的过滤器，负载均衡策略类触发蓝绿灰度发布

条件表达

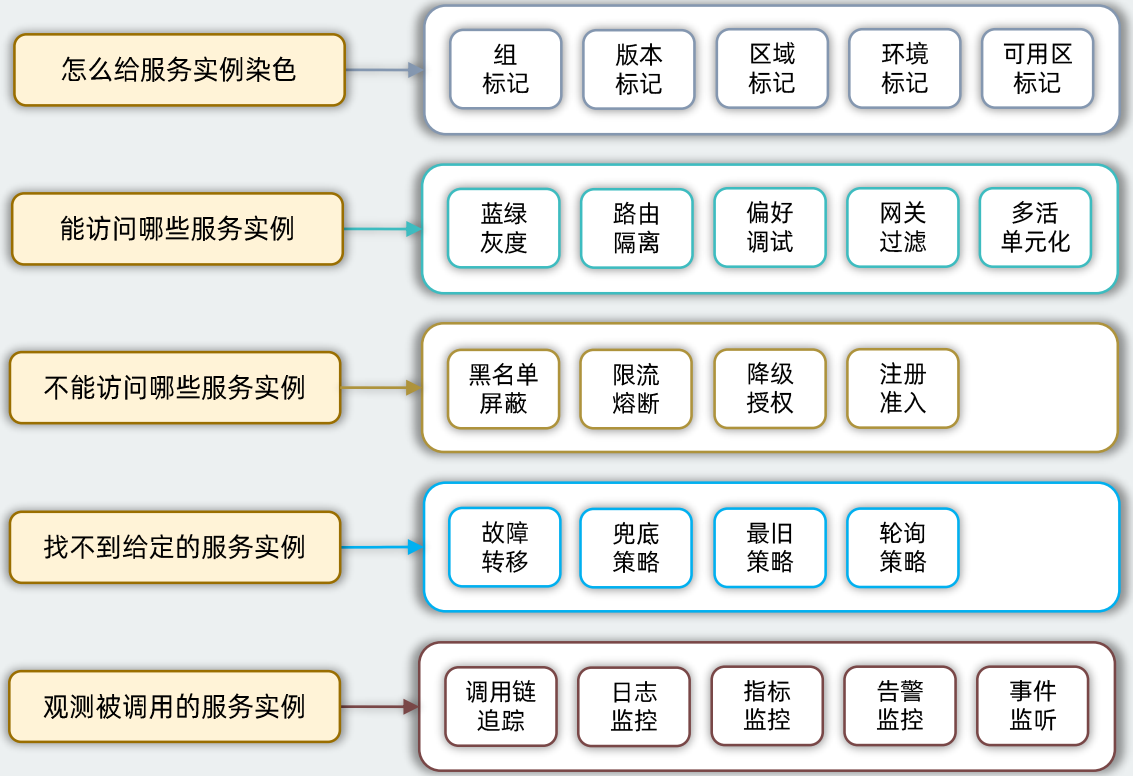
- Spring Spel条件表达式
  支持等于=、不等于!=、大于>、小于<、与&&、或||、匹配matches，以及加减乘除取模等全部标准表达式用法
- Spring Matcher通配表达式
  支持多个通配*、单个通配?等全部标准表达式用法

Discovery Wiki http://nepxion.com/discovery
Polaris Wiki http://nepxion.com/polaris


01

总体概述

功能全景


怎么给服务实例染色
组标记
版本标记
区域标记
环境标记
可用区标记
能访问哪些服务实例
蓝绿灰度
路由隔离
偏好调试
网关过滤
多活单元化
不能访问哪些服务实例
黑名单屏蔽
限流熔断
降级授权
注册准入
找不到给定的服务实例
故障转移
兜底策略
最旧策略
轮询策略
观测被调用的服务实例
调用链追踪
日志监控
指标监控
告警监控
事件监听

01 总体概述

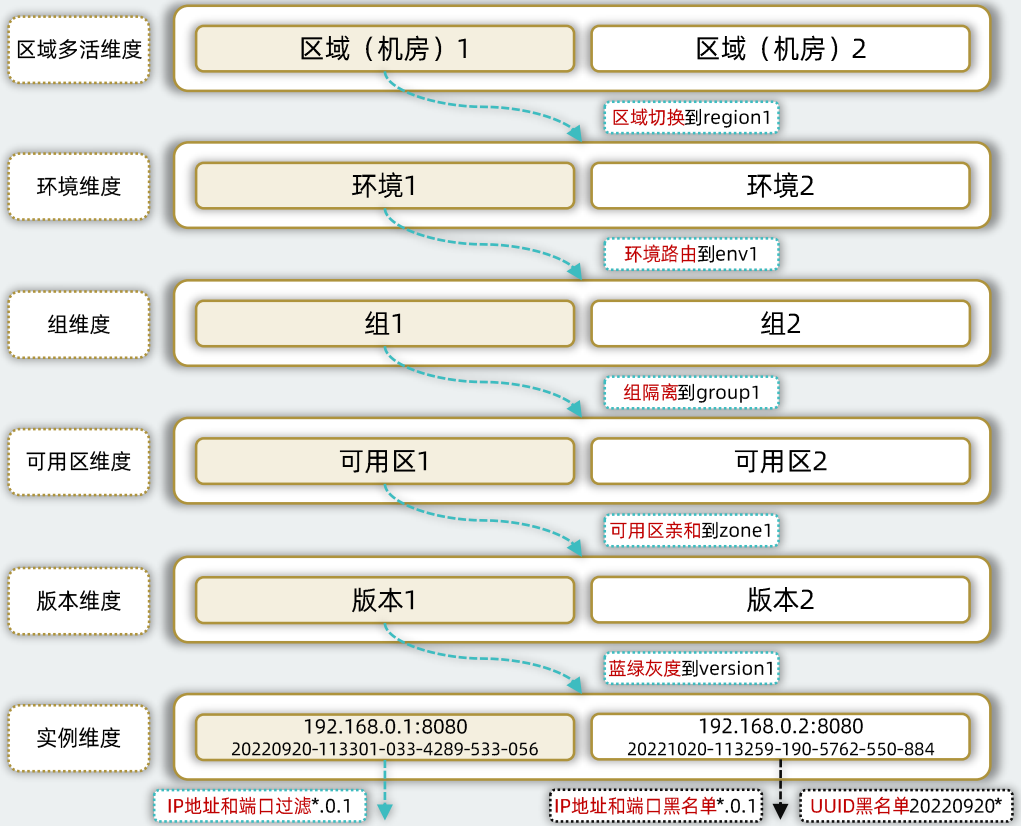
功能全景

七大经典维度实施流量管控

维度	概念	应用场景	功能侧重点	关键头
组（Group）	服务实例的系统ID 系统逻辑分组	路由隔离	① 组负载均衡隔离：调用端和提供端的元数据group是否相同 ② 组Header传值策略隔离：Header（n-d-group）和提供端的元数据group是否相同 ③ 不支持故障转移	n-d-group
版本（Version）	服务实例的版本 适用于生产环境	蓝绿灰度发布 路由转移 故障转移	① 版本条件匹配蓝绿发布 ② 版本权重灰度发布 ③ 版本偏好：非蓝绿灰度发布场景下，路由到相应版本的实例，稳定版本策略、指定版本策略 ④ 版本故障转移：未找到相应版本的服务实例，路由到其它版本，负载均衡策略、稳定版本策略、指定版本策略	n-d-version n-d-version-weight n-d-version-prefer n-d-version-failover
区域（Region）	服务实例的区域 适用于多活单元化 适用于多机房 适用于多环境	蓝绿灰度发布 同城双活/异地多活 路由转移 故障转移	① 区域条件匹配蓝绿发布 ② 区域权重灰度发布 ③ 区域多活单元化 ④ 区域调试路由：多区域路由隔离下跨区服务调用的调试手段 ⑤ 区域故障转移：未找到相应区域的服务实例，路由到其它区域，负载均衡策略、指定区域策略	n-d-region n-d-region-weight n-d-region-transfer n-d-region-failover
环境（Env）	服务实例的环境 适用于测试环境	路由隔离 故障转移	① 环境隔离路由：Header（n-d-env）和提供端的元数据env是否相同 ② 环境故障转移：未找到相应环境的服务实例，路由到其它环境，指定环境（未配置，默认为common）策略	n-d-env n-d-env-failover
可用区（Zone）	服务实例的可用区 适用于多机房	路由隔离 故障转移	① 可用区亲和性隔离路由：调用端和提供端的元数据zone是否相同 ② 可用区故障转移：未找到相应可用区的服务实例，路由到其它可用区，支持负载均衡策略、指定区可用区策略	n-d-zone-failover
IP地址和端口（Address）	服务实例机器地址	蓝绿灰度发布 路由隔离 故障转移 无损下线	① IP地址和端口匹配蓝绿发布 ② IP地址和端口权重灰度发布 ③ IP地址和端口故障转移：未找到相应IP地址和端口的服务实例，路由到其它地址，负载均衡策略、指定区IP地址和端口策略 ④ IP地址和端口无损下线黑名单屏蔽	n-d-address n-d-address-failover n-d-address-blacklist
全局唯一ID	服务实例机器ID	无损下线	① 全局唯一ID无损下线黑名单屏蔽	n-d-id-blacklist

01 总体概述

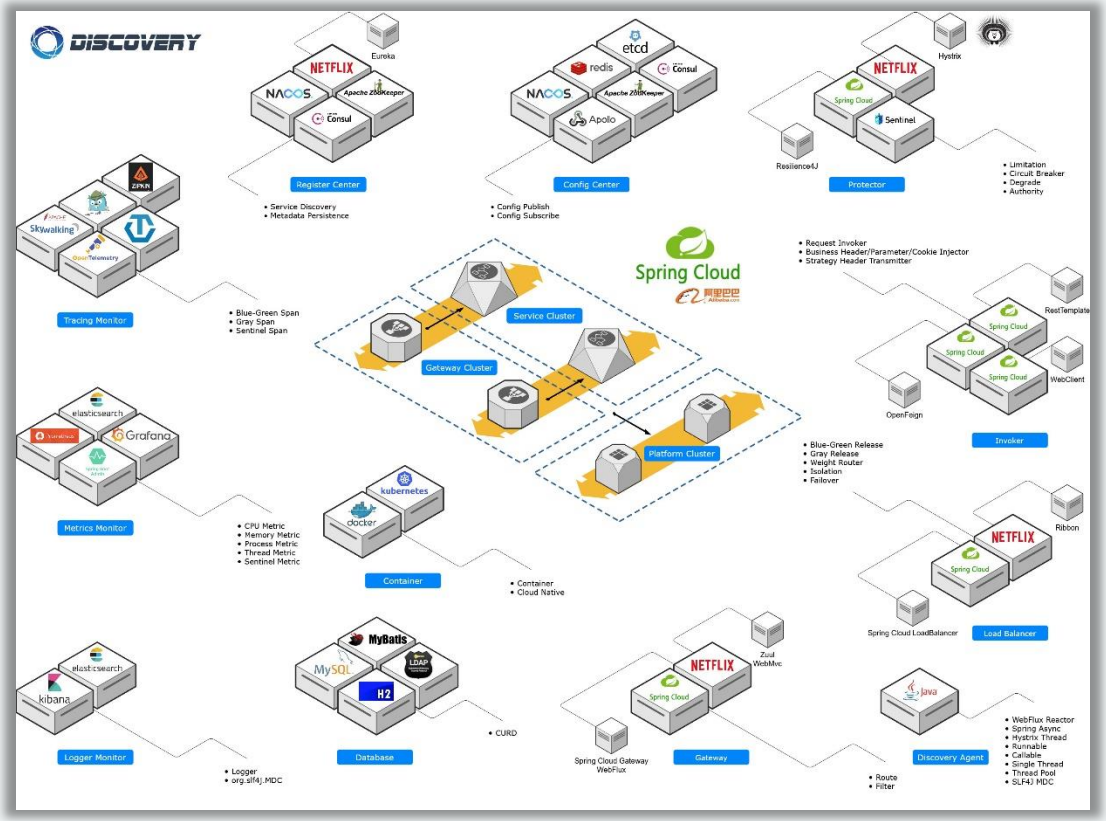
功能全景



七大经典维度宏观到微观并集式过滤

01 总体概述

技术栈全景



01 总体概述

技术栈版本兼容列表

提醒：版本号右边， ↑ 表示>=该版本号， ↓ 表示<=该版本号

框架版本	框架分支	框架状态	Spring Cloud版本	Spring Boot版本	Spring Cloud Alibaba版本
10.x.x 商业版	DiscoveryX/master		2023.x.x	3.2.x	2023.x.x.x
9.x.x 商业版	DiscoveryX/9.x.x		2022.x.x	3.1.x 3.0.x	2022.x.x.x
8.x.x 商业版	DiscoveryX/8.x.x		2021.x.x	2.7.x 2.6.x	2021.x.x.x
7.x.x 商业版	DiscoveryX/7.x.x		2020.x.x	2.5.x 2.4.1 ↑	2021.x
6.x.x	Discovery/6.x.x		Hoxton SR5 ↑ Hoxton Greenwich Finchley	2.3.x.RELEASE 2.2.x.RELEASE 2.1.x.RELEASE 2.0.x.RELEASE	2.2.x.RELEASE 2.1.x.RELEASE 2.0.x.RELEASE
~~5.x.x~~	~~Discovery/5.x.x~~		Greenwich	2.1.x.RELEASE	2.1.x.RELEASE
~~4.x.x~~	~~Discovery/4.x.x~~		Finchley	2.0.x.RELEASE	2.0.x.RELEASE
3.x.x	Discovery/3.x.x		Edgware	1.5.x.RELEASE	1.5.x.RELEASE
~~2.x.x~~	~~Discovery/2.x.x~~		Dalston	1.x.x.RELEASE	N/A
~~1.x.x~~	~~Discovery/1.x.x~~		Camden	1.x.x.RELEASE	N/A

表示维护中 | 表示不维护，但可用，强烈建议升级 | 表示不维护，不可用，已废弃

支持和兼容Java8～Java17以及更高的SDK版本

02 解决方案

蓝绿发布 Blue-Green Deployment

概念

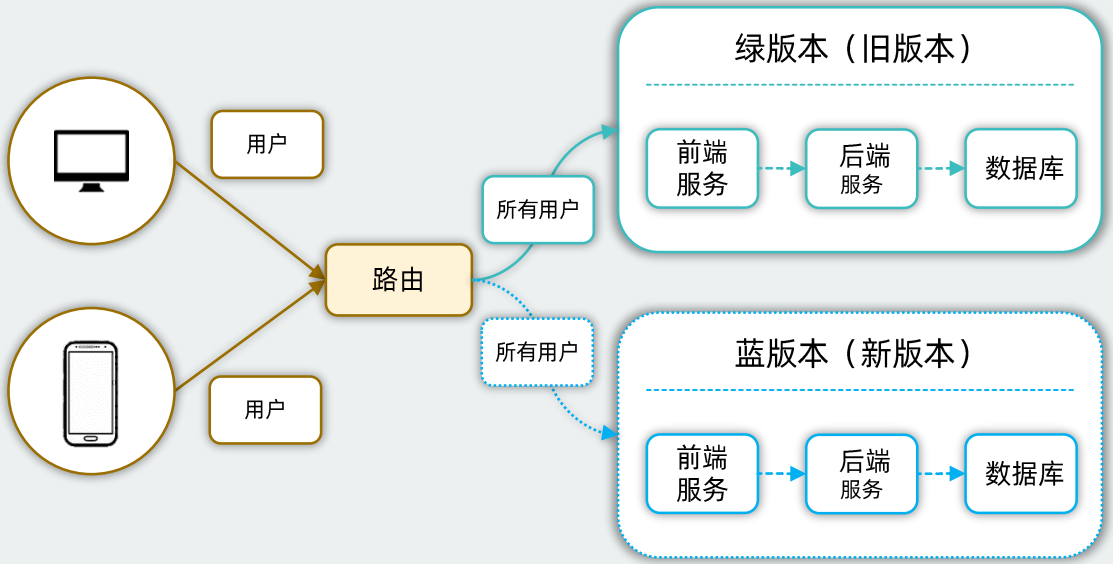
① 不停机旧版本，部署新版本，通过用户标记将流量在新版本和老版本切换
②属无损发布

优点

① 新版本升级和老版本回滚迅速
② 用户可以灵活控制流量走向

缺点

① 成本较高，需要部署蓝/绿两套环境



02 解决方案

灰度发布 Gray Release（又名金丝雀发布 Canary Release）

概念

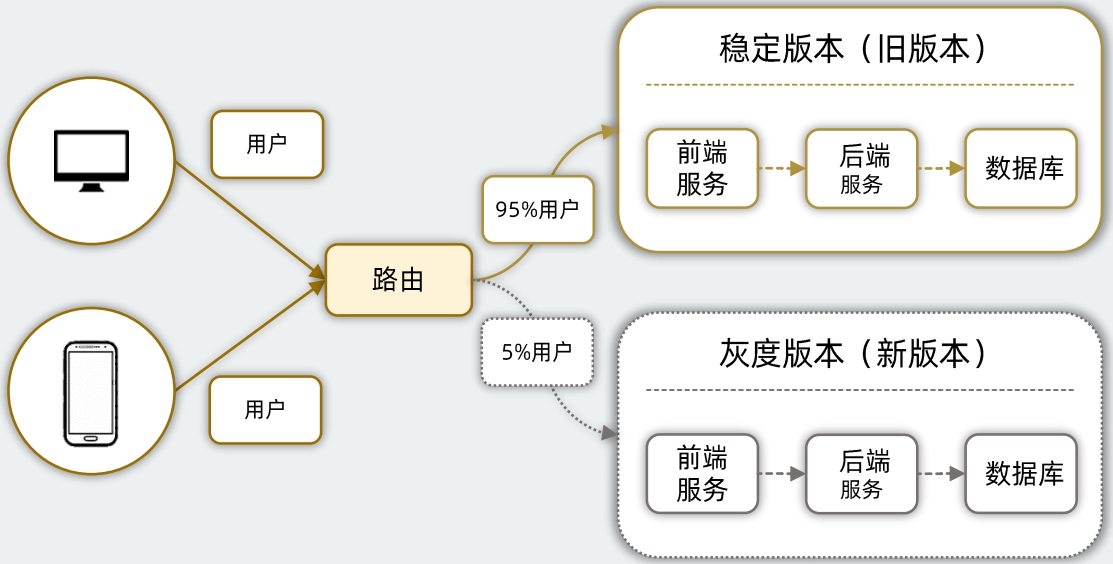
① 不停机旧版本，部署新版本，高比例流量（例如：95%）走旧版本，低比例流量（例如：5%）切换到新版本
② 通过观察无问题，逐步扩大范围，最终把所有流量都迁移到新版本上
③ 属无损发布

优点

① 灵活简单，不需要用户标记驱动
② 安全性高，新版本如果出现问题，只会发生在低比例的流量上

缺点

① 成本较高，需要部署稳定/灰度两套环境



02 解决方案

滚动发布 Rolling Release

概念

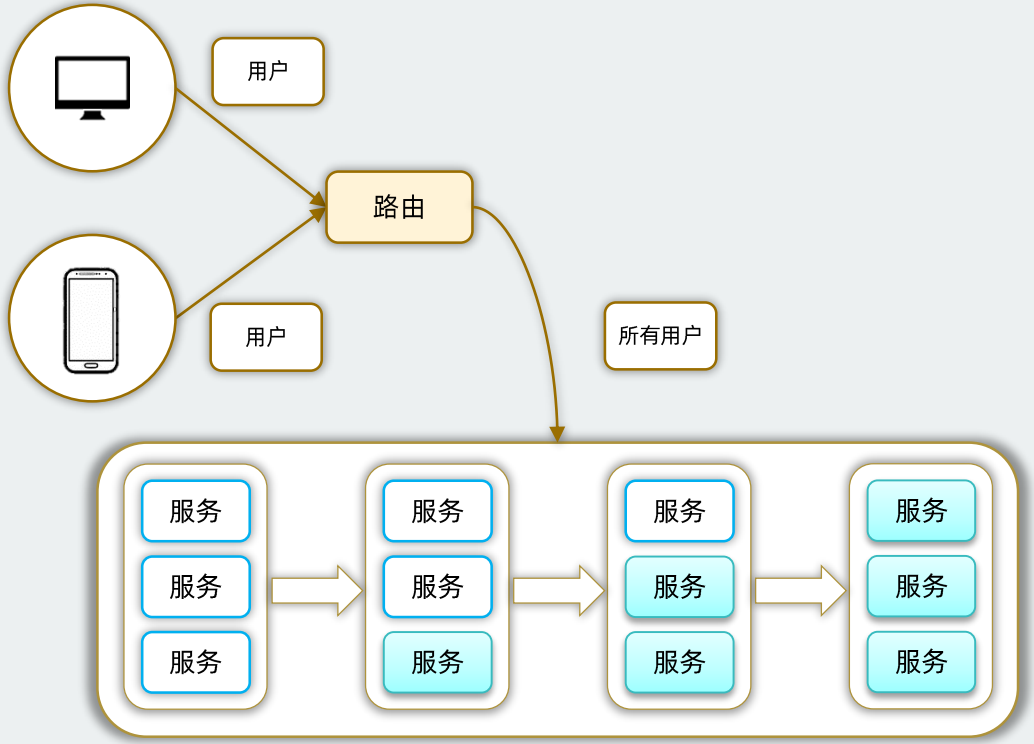
① 每次只升级一个或多个服务
② 通过观察无问题，不断执行这个过程，直到集群中的全部旧版本升级到新版本
③ 属有损发布

优点

① 成本较低，只需要部署一套环境
② 出现问题影响范围，只限于发生在若干台正在滚动发布的服务上

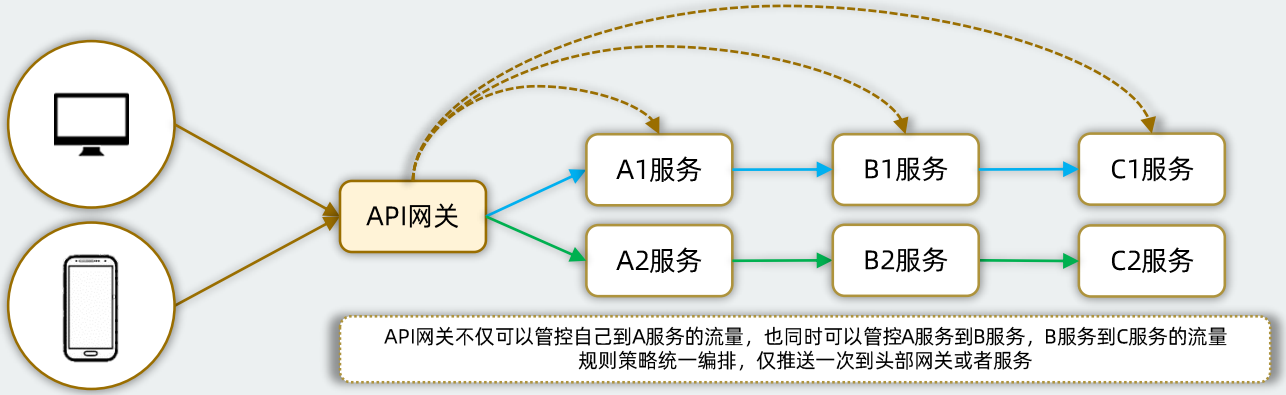
缺点

① 停止旧版本的过程中，无法精确计算旧版本是否已经完成它正在执行的工作，需要靠业务自身去判断
② 旧版本不保留，一旦全部升级完毕后才发现问题，无法快速回滚，必须重新降级部署
③ 发布和回滚需要较长的时间周期



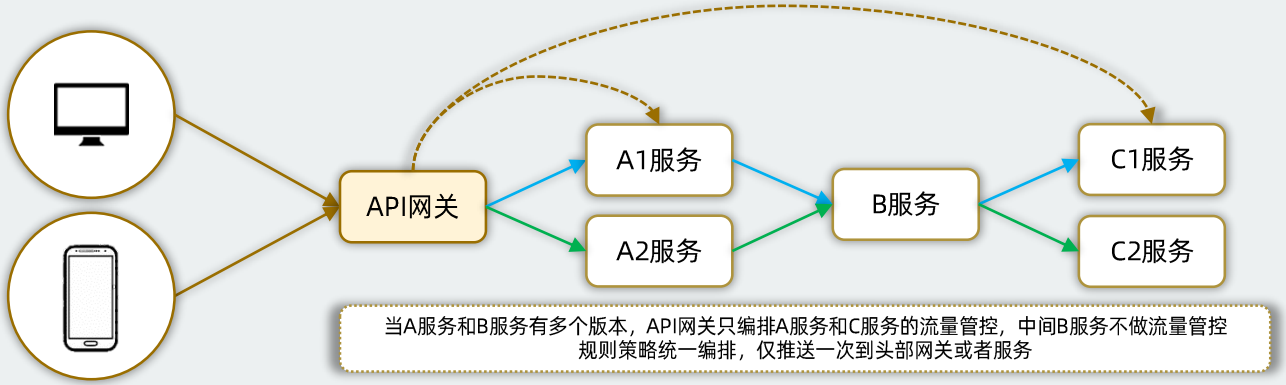
02 解决方案

全链路和单链路



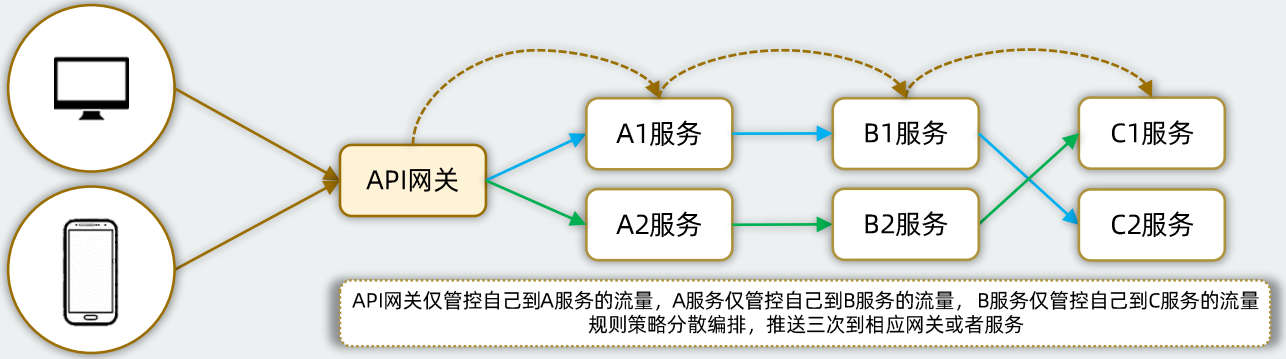
全链路

网关或者服务可以越级管控
它后面所有服务的流量，链路隔离



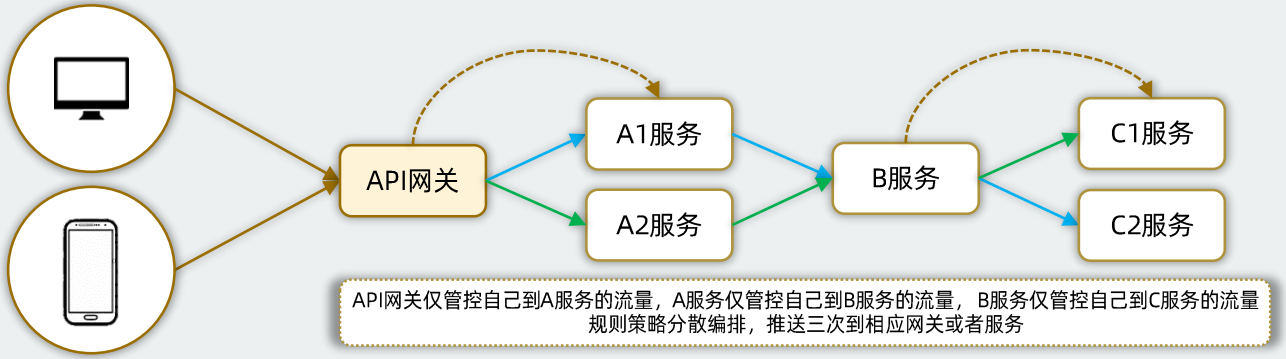
02 解决方案

全链路和单链路



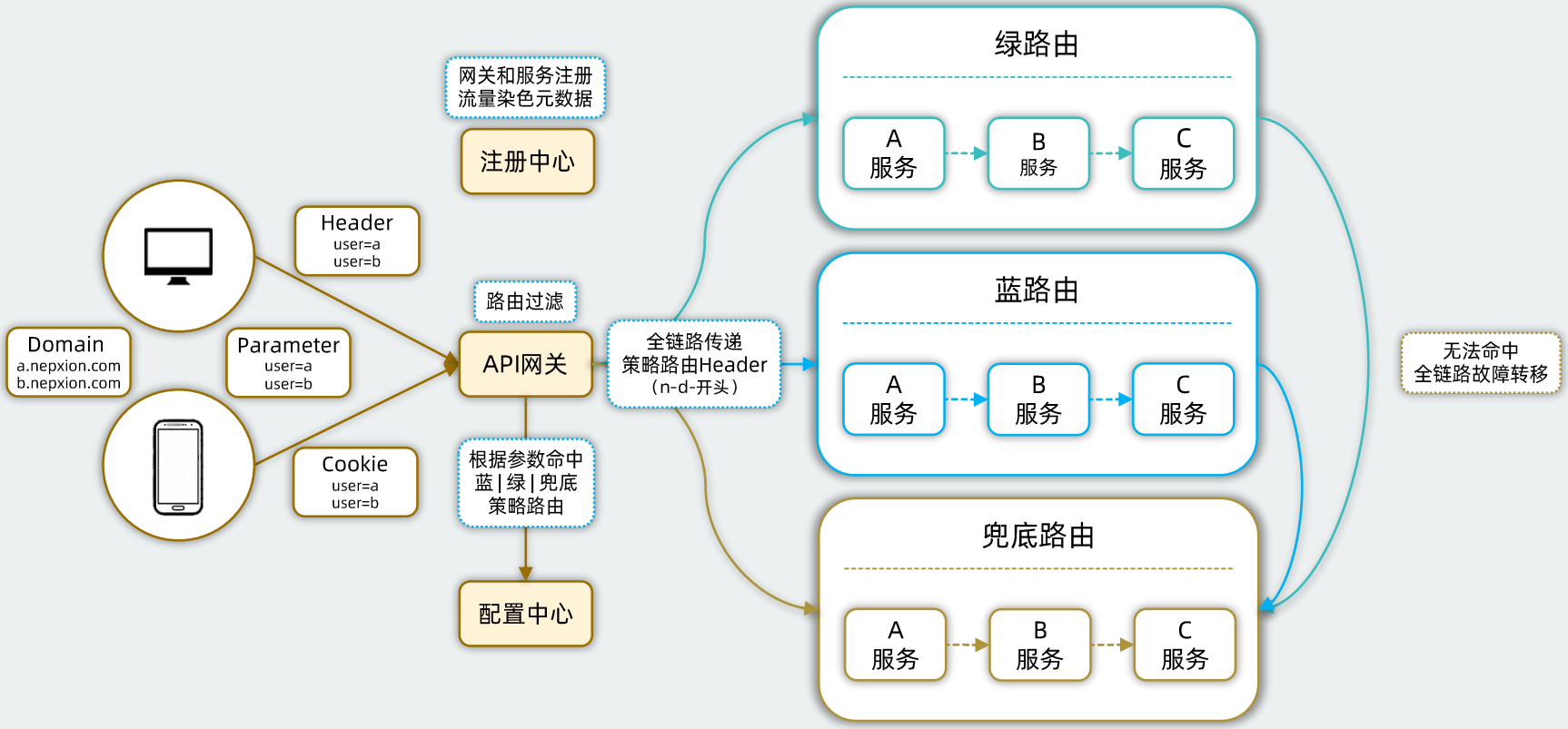
单链路

网关或者服务只可以管控
它下一级服务的流量，链路不隔离



02 解决方案

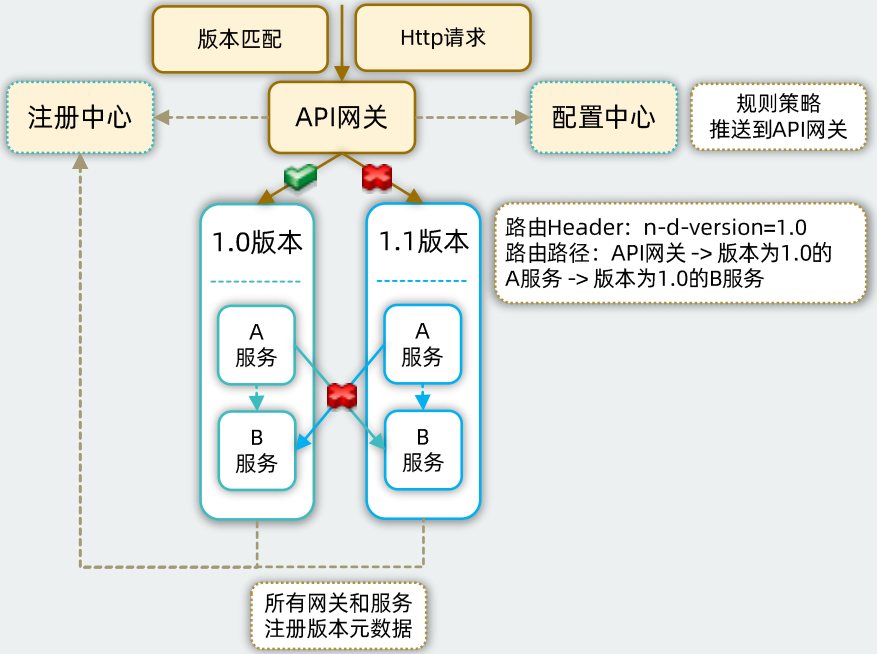
全链路蓝绿发布

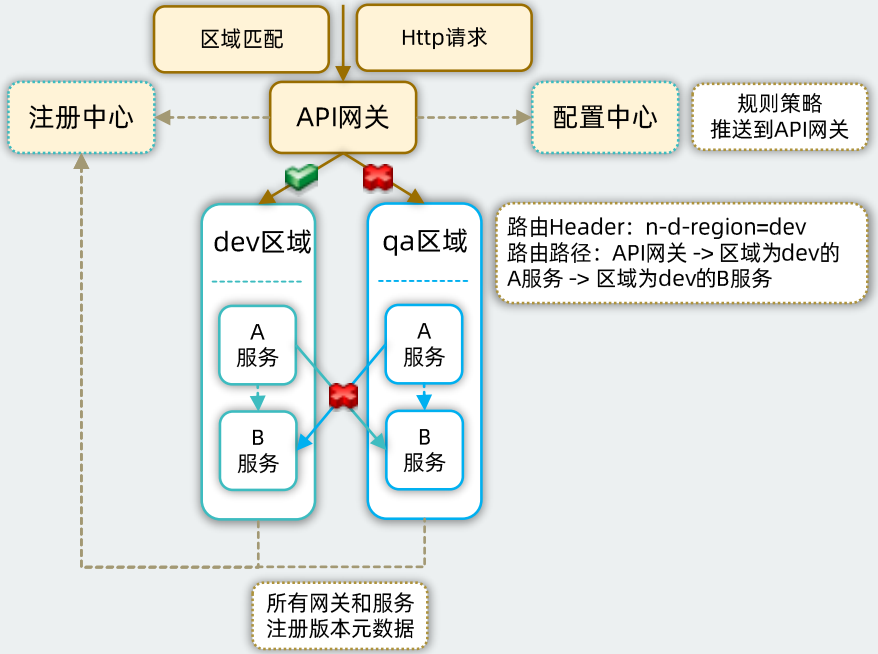


02

解决方案

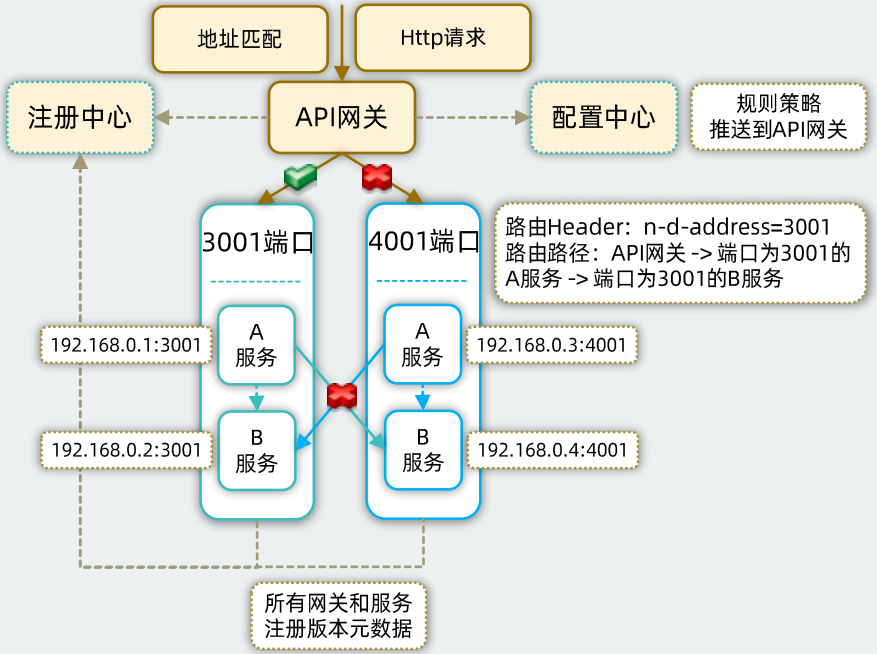
全链路蓝绿发布





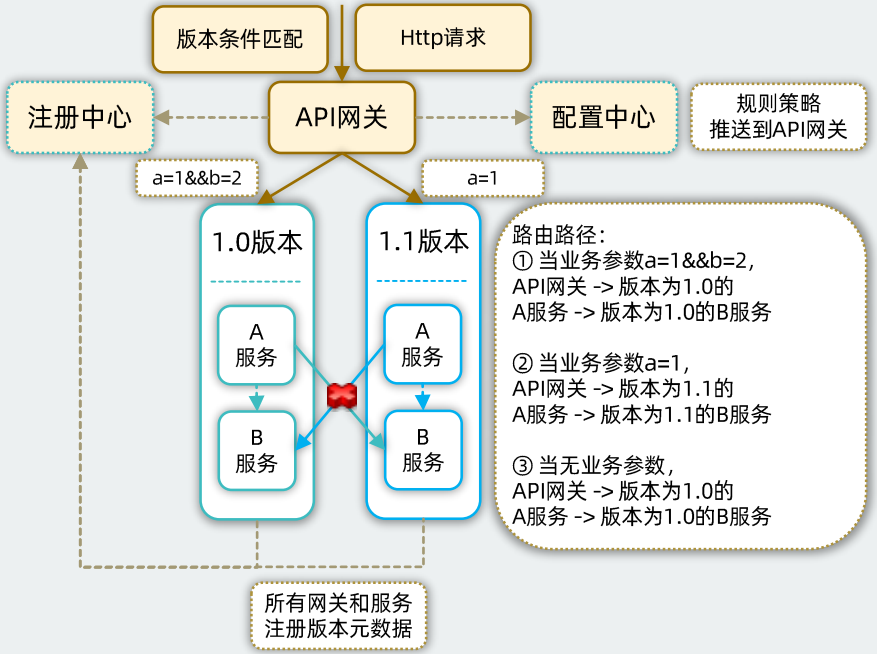
02 解决方案

全链路蓝绿发布



02 解决方案

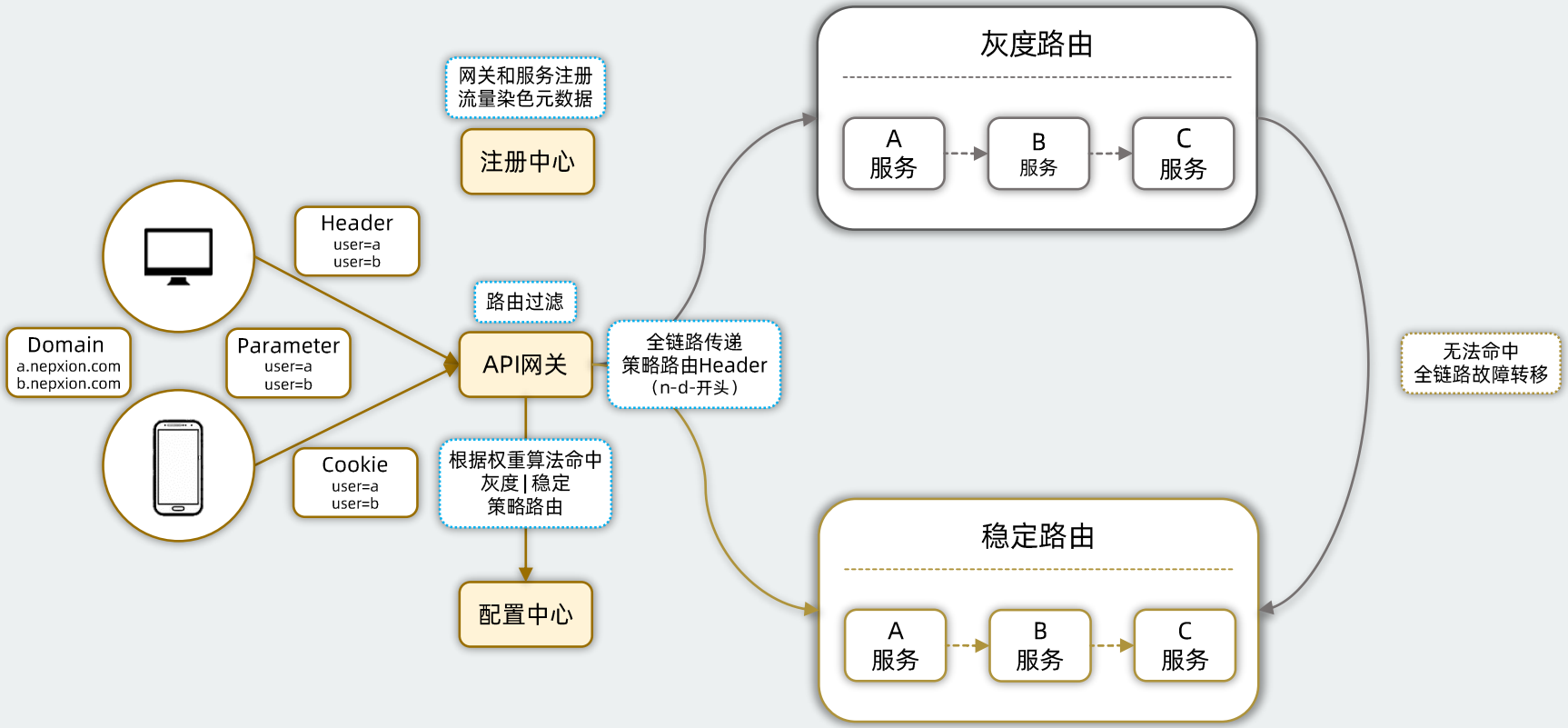
全链路条件蓝绿发布



* 区域条件权重、IP地址和端口条件权重 略

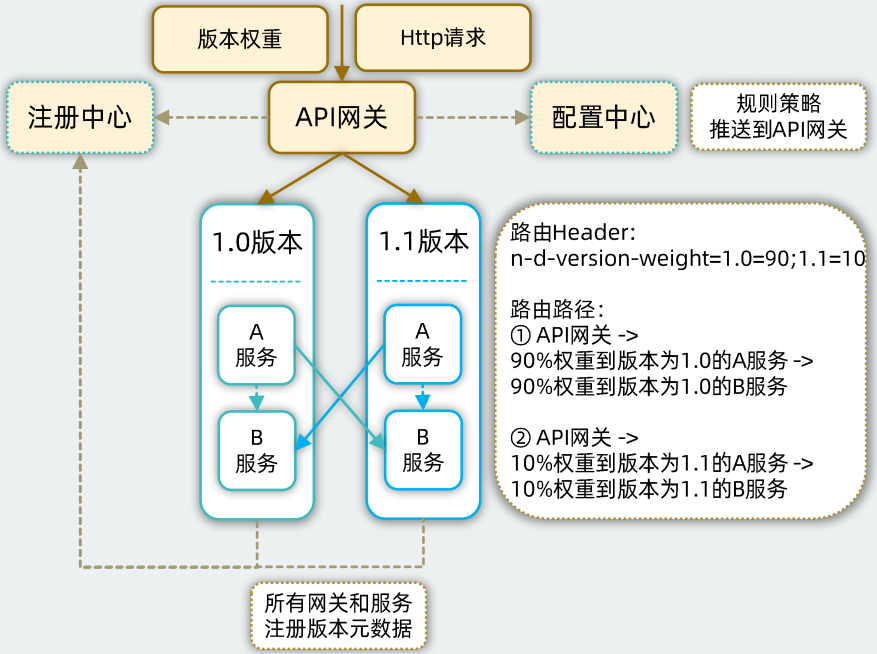
02 解决方案

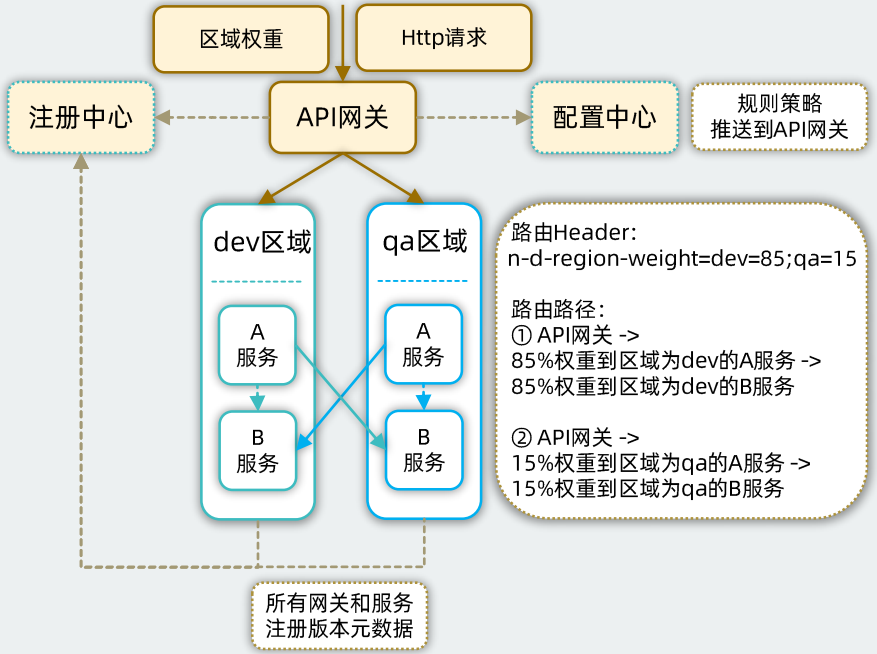
全链路灰度发布



02 解决方案

全链路灰度发布

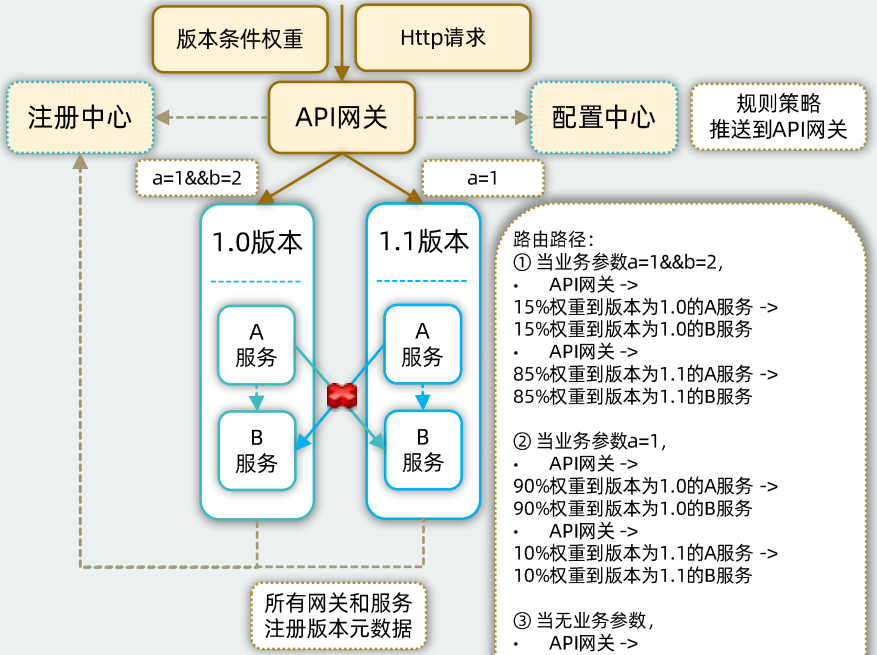




* 服务列表随机权重算法

02 解决方案

全链路条件灰度发布



路由路径：
① 当业务参数a=1&&b=2，
- API网关 ->

15%权重到版本为1.0的A服务 ->
15%权重到版本为1.0的B服务
- API网关 ->

85%权重到版本为1.1的A服务 ->
85%权重到版本为1.1的B服务

② 当业务参数a=1，
- API网关 ->

90%权重到版本为1.0的A服务 ->
90%权重到版本为1.0的B服务
- API网关 ->

10%权重到版本为1.1的A服务 ->
10%权重到版本为1.1的B服务

③ 当无业务参数，
- API网关 ->

100%权重到版本为1.0的A服务 ->
100%权重到版本为1.0的B服务
- API网关 ->

0%权重到版本为1.1的A服务 ->
0%权重到版本为1.1的B服务

* 路由（Header）列表随机权重算法

* 区域条件权重、IP地址和端口条件权重 略

02 解决方案

全链路单网关实施蓝绿灰度

场景

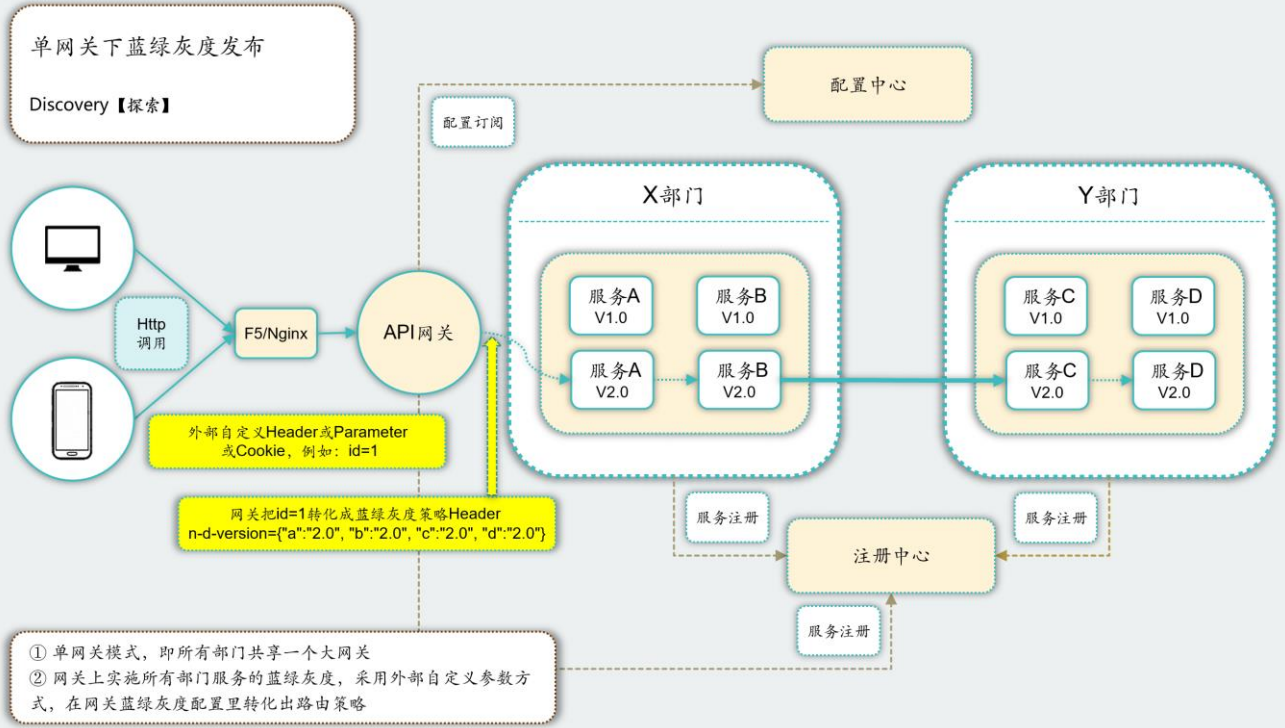
① 所有部门共享一个大网关

思路

① 参数化动态的简单蓝绿灰度发布方式

方案

① 按原生的蓝绿灰度发布来实施



02 解决方案

全链路域网关实施蓝绿灰度

场景

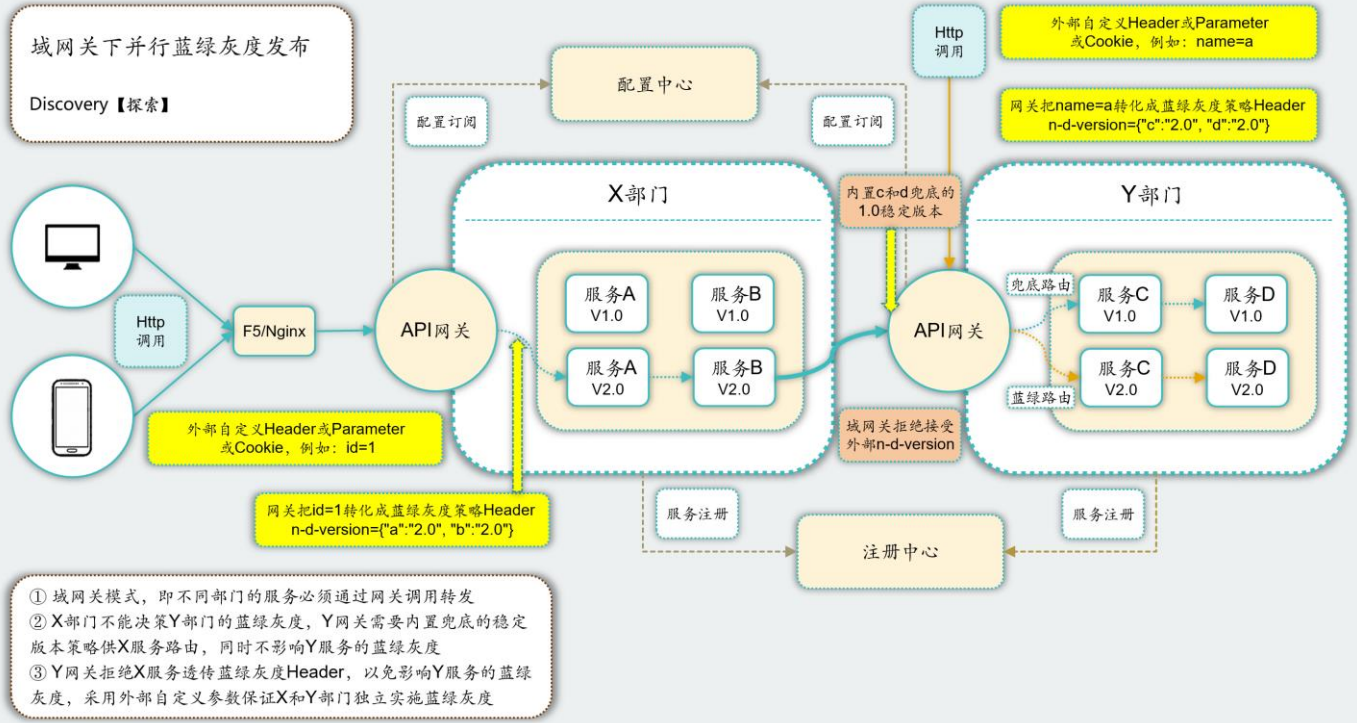
① A部门和B部门都有各自的网关
② A部门服务访问B部门服务，必须通过B部门网关

思路

① 当本部门服务和其它部门服务在同一时刻实施蓝绿灰度发布的时候，会产生混乱
② 本部门服务的蓝绿灰度发布只由本部门的网关来实施，其它部门无权对本部门服务实施

方案

① 域网关拒绝接受外部传入的蓝绿灰度策略
② 域网关配置兜底策略



02 解决方案

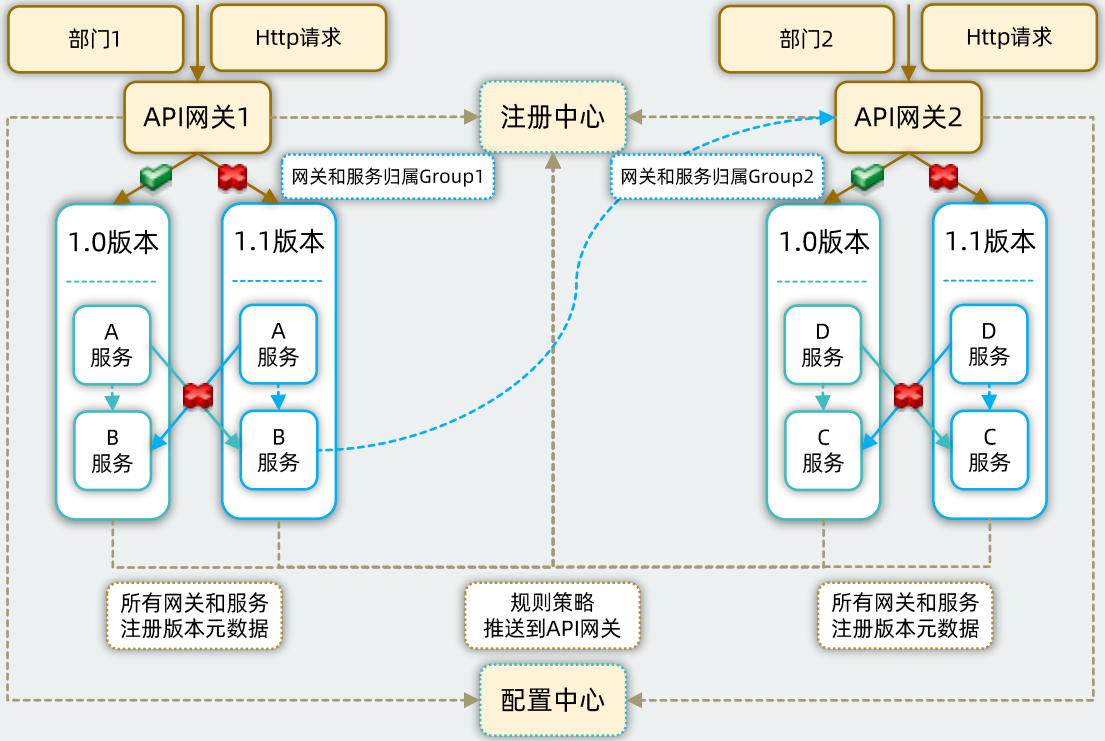
全链路域网关实施蓝绿灰度

示例

① 部门1，包括API网关1，A和B服务各两个版本；部门2，包括API网关2，C和D服务各两个版本
② 本部门服务访问必须链路新旧隔离
③ 本部门无权对其它部门服务实施蓝绿灰度

过程

① 两个部门，所有的服务都在并行执行蓝绿灰度
② API网关2配置兜底路由
③ API网关2拒绝API网关1传递进来的蓝绿灰度规则（默认设置）



详细资料点击此处

02 解决方案

全链路非域网关实施蓝绿灰度

场景

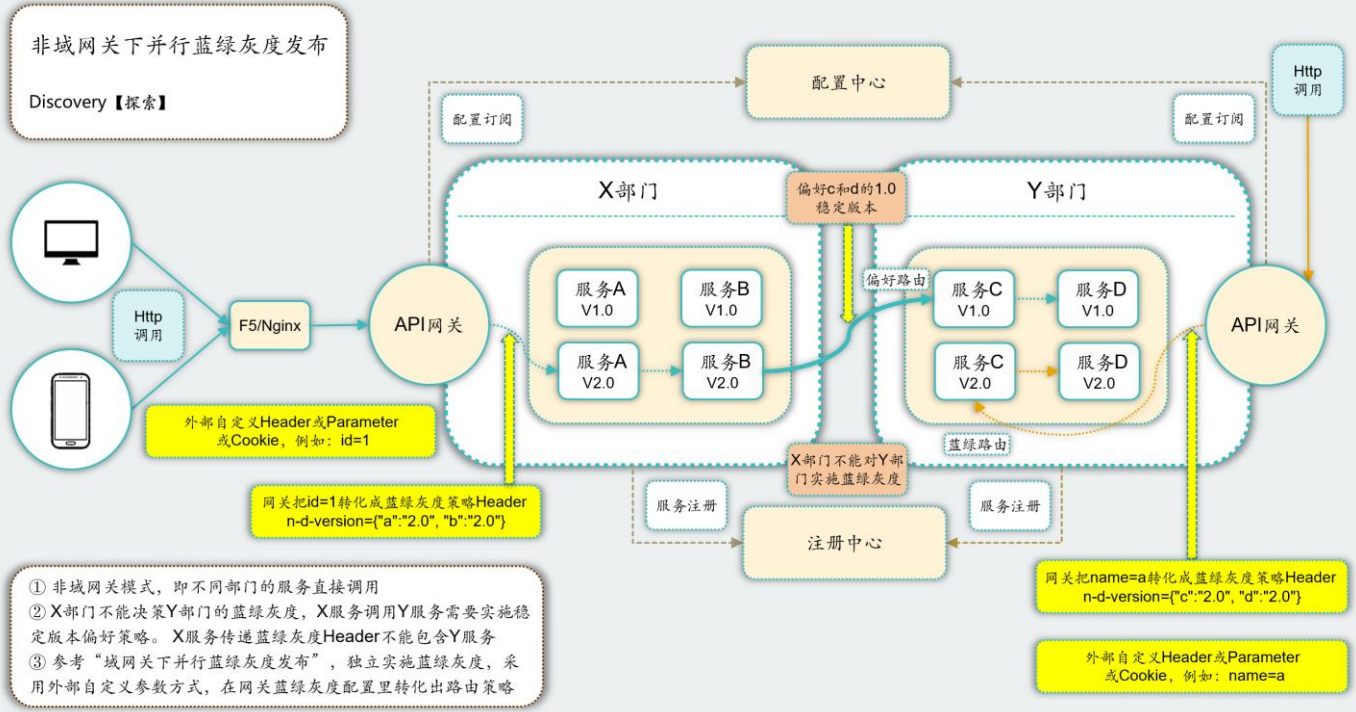
① A部门和B部门都有各自的网关
② A部门服务直接访问B部门服务

思路

① 当本部门服务和其它部门服务在同一时刻实施蓝绿灰度发布的时候，会产生混乱
② 本部门服务的蓝绿灰度发布只由本部门的网关来实施，其它部门无权对本部门服务实施

方案

① 本部门的网关不得配置其它部门服务的蓝绿灰度策略
② 所有服务开启全链路版本偏好路由的稳定版本开关



02 解决方案

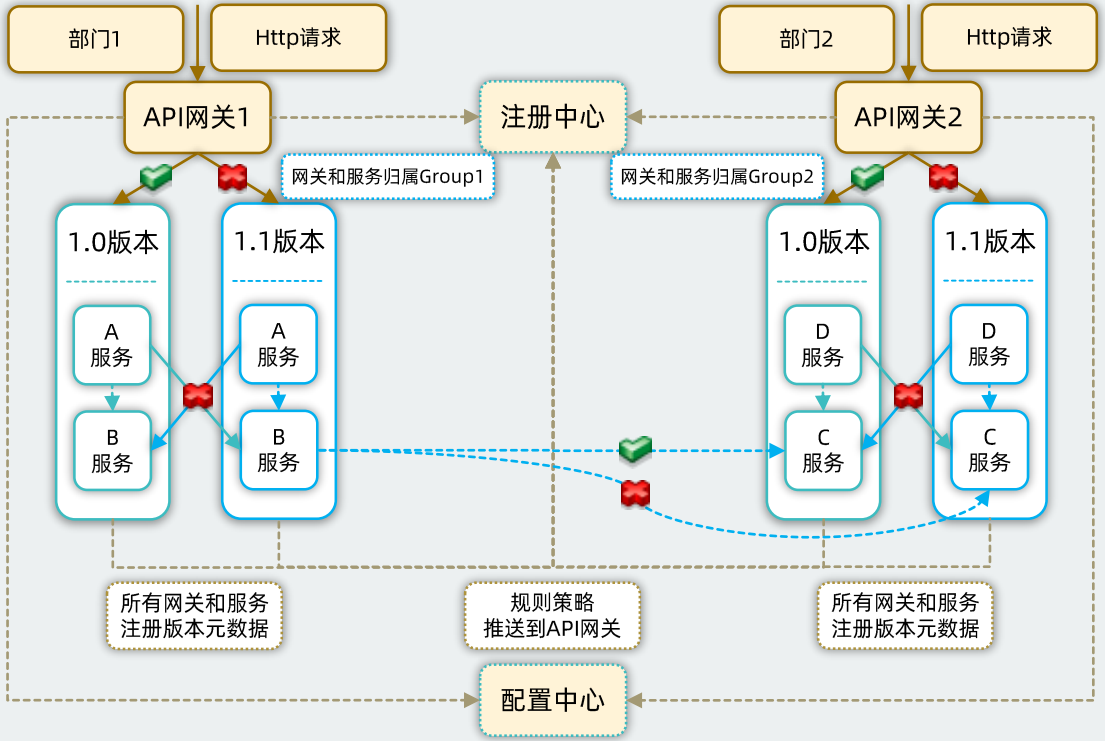
全链路非域网关实施蓝绿灰度

示例

① 部门1，包括API网关1，A和B服务各两个版本；部门2，包括API网关2，C和D服务各两个版本
② 服务的版本流量染色必须可排序
③ 本部门服务访问必须链路新旧隔离
④ 本部门无权对其它部门服务实施蓝绿灰度

过程

① 两个部门，所有的服务都在并行执行蓝绿灰度
② 调用链路 服务A -> 服务B -> 服务C，B服务不能访问未经流量验证的C服务的新版本（1.1版本）
③ 所有服务开启版本偏好。配置如下
spring.application.strategy.version.prefer.enabled=true



详细资料点击此处

02 解决方案

全链路全局订阅实施蓝绿灰度

场景

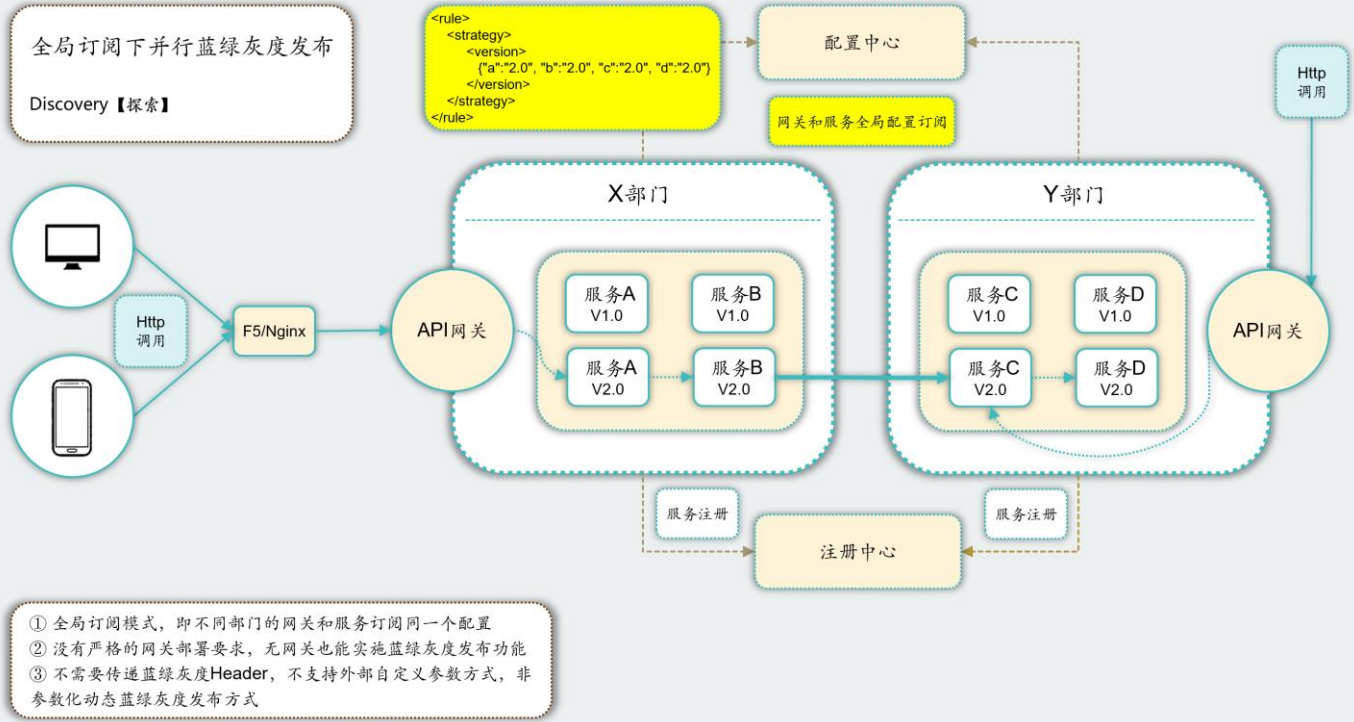
① A部门和B部门是否有各自的网关，服务之间如何调用都不做要求

思路

① 非参数化动态的简单蓝绿灰度发布方式
② 无需Header和外部参数传递

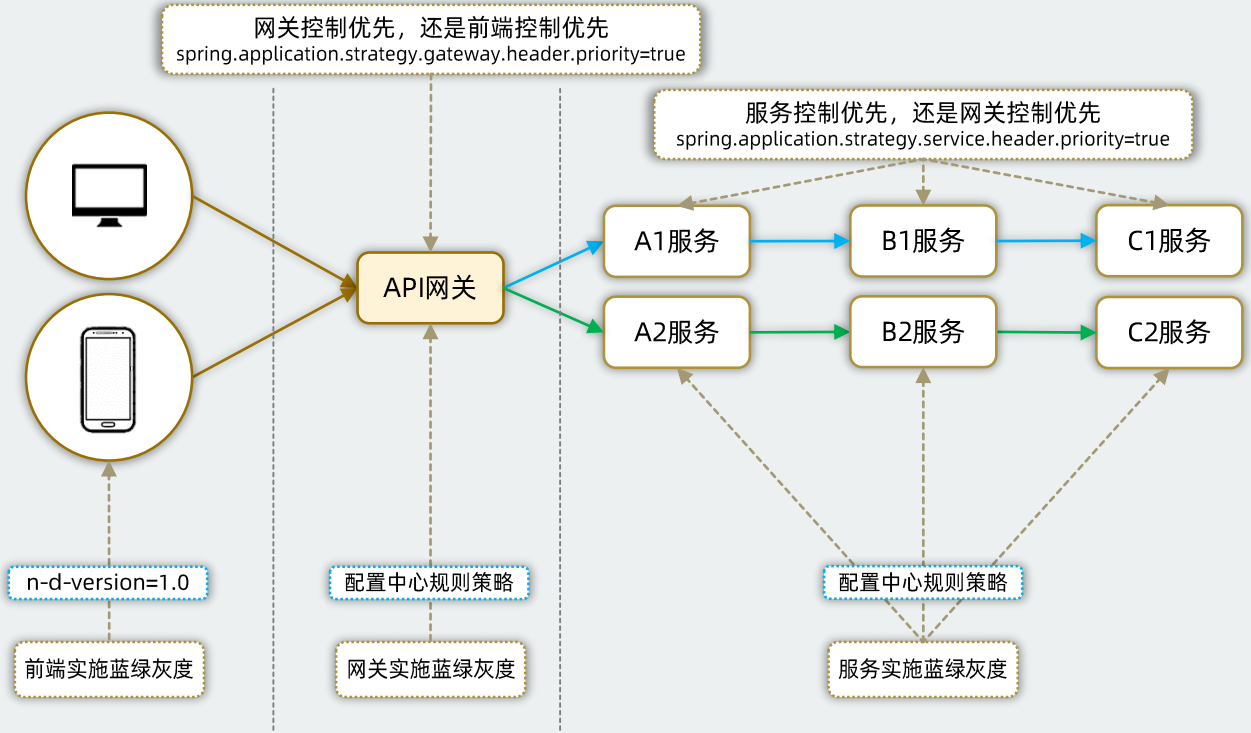
方案

① 不同部门的网关和服务订阅同一个配置
② 订阅的Group和Data Id必须都配置为所有网关和服务的元数据Group值



02 解决方案

端到端混合蓝绿灰度发布



02 解决方案

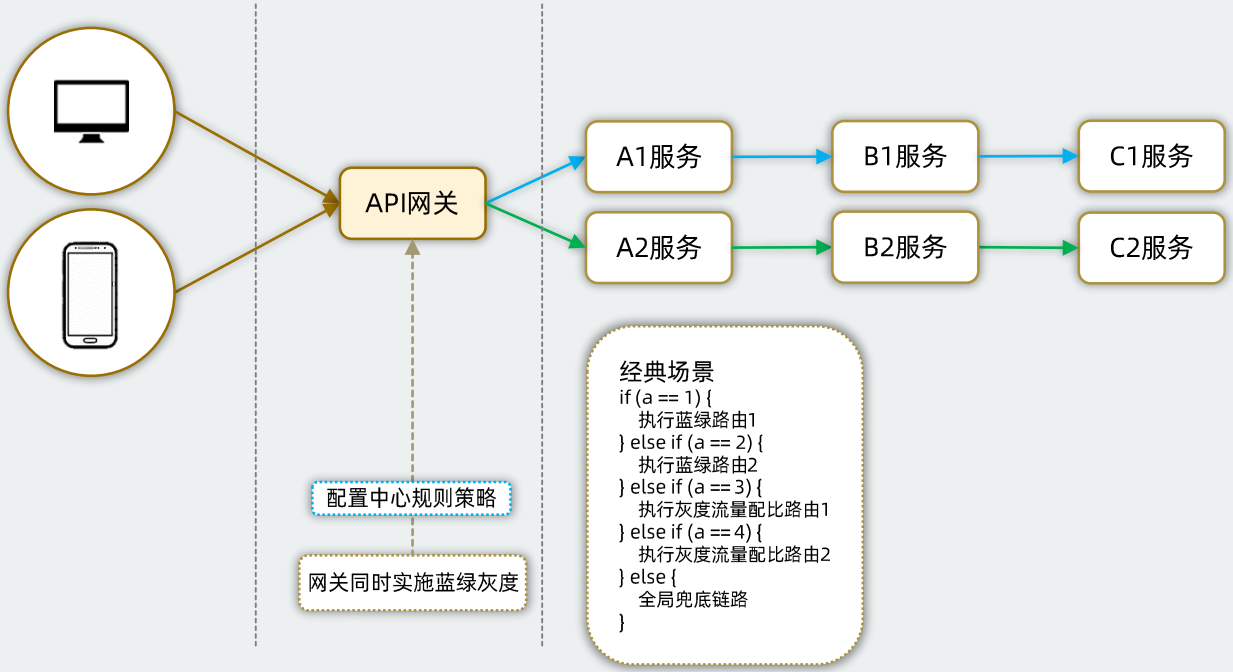
单节点混合蓝绿灰度发布

优先级

① 蓝绿条件策略 > 灰度条件策略
② 蓝绿兜底策略 > 灰度兜底策略 > 全局兜底策略

注意项

① 蓝绿灰度混合发布中，蓝绿兜底存在，灰度发布不会生效
②无条件驱动下的灰度发布，新/旧服务流量配比0:100，其作用等同于蓝绿兜底或者全局兜底



02 解决方案

全链路无编排蓝绿灰度发布

方案

① 版本标签轮番交替使用
② 业务驱动参数轮番交替切换
③ 无兜底路由，走故障转移

优点

① 不需要通过时间戳方式或者数字递增方式去打标签
② 不需要每次发布都要去修改规则策略
③ 不需要指定具体要发布的服务

缺点

① 要牢记每次发布中版本标签切换的情况
② 要牢记业务参数在每次发布驱动链路的情况
③ 要牢记打开故障转移

业务参数驱动的蓝绿规则策略

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy-release>
    <conditions type="blue-green">
      <condition id="condition-0" expression="#H['a'] == '1'" version-id="route-0"/>
      <condition id="condition-1" expression="#H['a'] == '2'" version-id="route-1"/>
    </conditions>
    <routes>
      <route id="route-0" type="version">blue</route>
      <route id="route-1" type="version">green</route>
    </routes>
  </strategy-release>
</rule>
```

简化

无业务参数驱动的简化蓝绿规则策略

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy>
    <version>blue</version>
  </strategy>
</rule>
```

业务参数驱动的灰度规则策略

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy-release>
    <conditions type="gray">
      <condition id="condition-0" expression="#H['a'] == '3'" version-id="route-0=10;route-1=90"/>
      <condition id="condition-1" expression="#H['a'] == '4'" version-id="route-0=90;route-1=10"/>
    </conditions>
    <routes>
      <route id="route-0" type="version">blue</route>
      <route id="route-1" type="version">green</route>
    </routes>
  </strategy-release>
</rule>
```

简化

无业务参数驱动的简化灰度规则策略

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy>
    <version-weight>blue=10;green=90</version-weight>
  </strategy>
</rule>
```

02 解决方案

全链路智能编排实施蓝绿灰度发布

概念

① 路由链路在后台会智能化编排
② 无需关心服务版本的信息
③ 只需配置条件表达式

方案

① 向控制台发送请求
② 控制台根据新旧版本的判断，智能编排出两条新旧路由链路，并给它们赋予不同的条件表达式
③ 解析简单规则策略为最终规则策略，保存至配置中心

重点

① 动态版本：版本号采用时间戳或者数字递增的方式
② 静态版本：版本号采用固定方式。例如，base，gray，green，blue等

兜底规则策略

```
service:
 - a
 - b
sort: version
```

蓝绿规则策略

```
service:
 - a
 - b
blueGreen:
 - expression: "#H['xyz'] == '1'"
   route: green
 - expression: "#H['xyz'] == '2'"
   route: blue
sort: version
```

灰度规则策略

```
service:
 - a
 - b
gray:
 - expression: "#H['xyz'] == '3'"
   weight:
    - 90
    - 10
 - expression: "#H['xyz'] == '4'"
   weight:
    - 70
    - 30
 - weight:
    - 100
    - 0
sort: version
```

二次蓝绿灰度规则策略

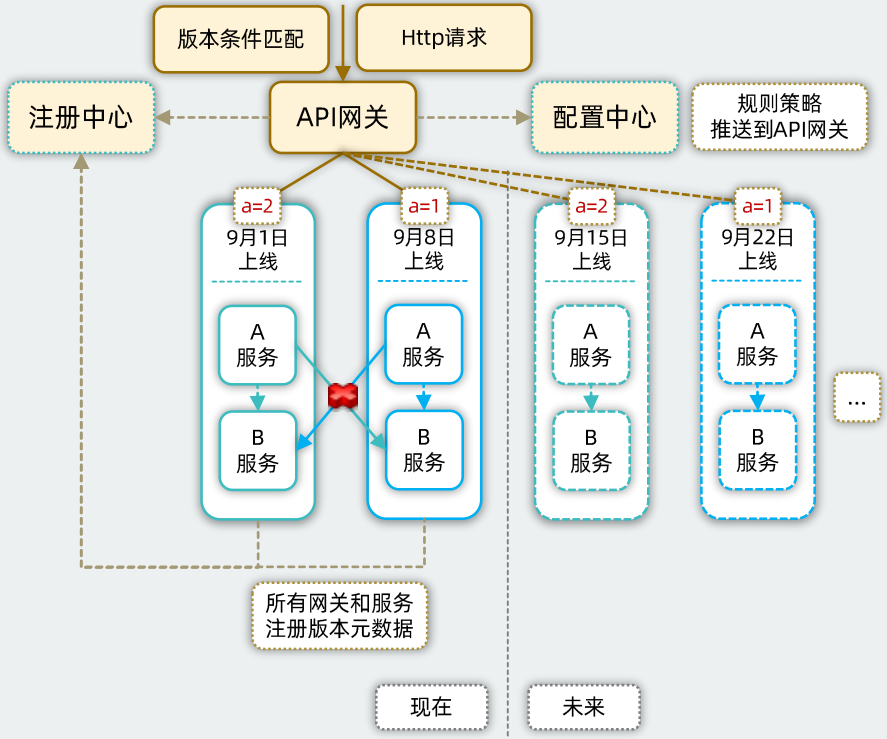
```
service:
 - a
 - b
condition: true
sort: version
```

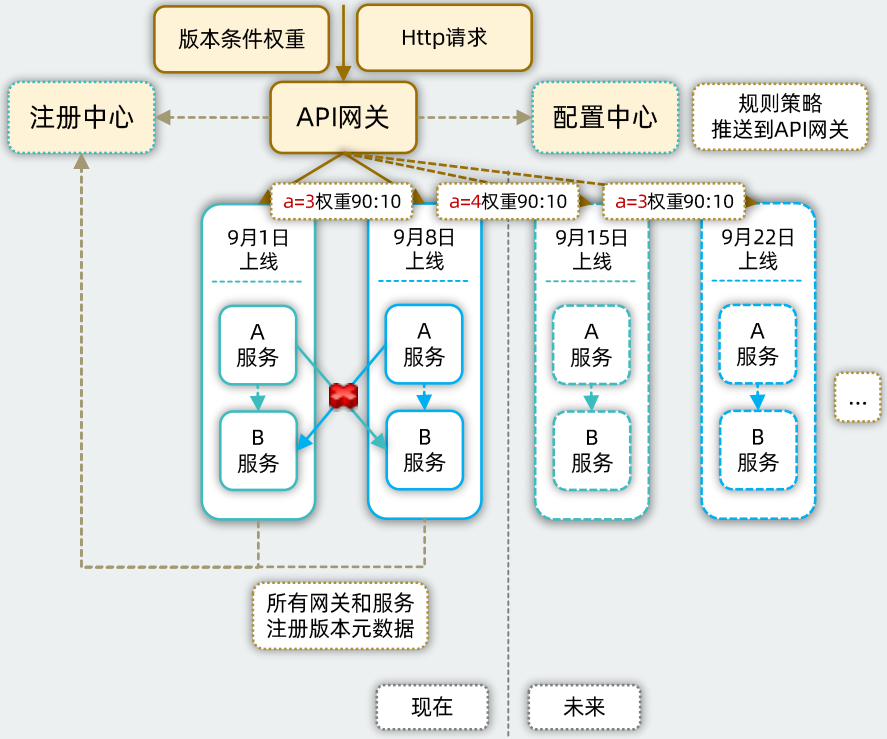
规则策略解释

① 指定要实施的服务列表（例如，a和b）
② 版本号排序类型（sort），可选值为version和time，缺省为version（不需要配置sort: version）
当排序类型为version时，适用于动态版本。排序后版本号列表的第一个值作为旧的稳定版本
当排序类型为time时，适用于动态和静态版本。根据服务实例全局唯一ID的时间戳前缀进行排序，把上线时间最早的服务实例的版本号作为旧的稳定版本
③ 兜底规则策略无需指定条件（expression）
④ 蓝绿规则策略必须指定蓝绿两个条件（expression）和路由（route）项
路由green代表绿（旧版本）路由链路，路由blue代表蓝（新版本）路由链路
⑤ 灰度规则策略条件和权重项可以无数个（不同条件下的不同权重配比），允许有一项条件（expression）为空（无条件驱动下的权重）
灰度权重（weight）必须为两个整数（可以大于100）的数字，按次序为别代表稳定（旧版本）路由链路权重，灰度（新版本）路由链路权重
⑥ 上述三个规则策略组合在一起，为蓝绿灰度混合发布
⑦ 二次蓝绿灰度，表示第一次执行过蓝绿灰度发布后，会保留条件（expression）项，清除路由（route）项，以后蓝绿灰度不必填写条件（expression）项，只需用condition: true代替即可
⑧ 支持Yaml和Json两种格式

02 解决方案

全链路无编排蓝绿灰度发布





02 解决方案

全链路自动化模拟流程测试

方案

① 全链路智能编排 + 流量侦测
② 网关或服务为侦测入口
③ 自动判断结果准确性
④ 服务引入discovery-plugin-admin-center-starter依赖

过程

① 云上测试
- 访问测试平台
- 日志根据测试用例UUID输出和采集，测试并行控制，Web界面展现

② 本地测试
- 获取离线包
- 通过命令行（.bat或者.sh）执行

适用

① 测试环境
② 开发环境

测试结果部分示例

```
【模拟场景3】蓝绿策略，测试全链路侦测，Header：{xyz=1}...
侦测次数：100
侦测结果：discovery-guide-service-a@@1.0 命中次数=100
侦测结果：discovery-guide-service-a@@1.1 命中次数=0
侦测结果：discovery-guide-service-b@@1.0 命中次数=100
侦测结果：discovery-guide-service-b@@1.1 命中次数=0
测试耗时：1 秒

【模拟场景3】灰度策略，测试全链路侦测，Header：{xyz=3}...
侦测次数：500
侦测进度：第100次...
侦测进度：第200次...
侦测进度：第300次...
侦测进度：第400次...
侦测进度：第500次...
侦测结果：discovery-guide-service-a@@1.0 命中次数=448
侦测结果：discovery-guide-service-a@@1.1 命中次数=52
侦测结果：discovery-guide-service-b@@1.0 命中次数=448
侦测结果：discovery-guide-service-b@@1.1 命中次数=52
期望结果：旧版本路由权重=90%, 新版本路由权重=10%
最终结果：旧版本路由权重=89.6%, 新版本路由权重=10.4%
测试耗时：7 秒
```

详细资料点击此处

02 解决方案

全链路自动化流量侦测测试

方案

① 全链路流量侦测
② 网关或服务为侦测入口
③ 人工判断结果准确性
④ 服务引入discovery-plugin-admin-center-starter依赖

过程

① 云上测试
- 访问测试平台
- 日志根据测试用例UUID输出和采集，Web界面展现

② 本地测试
- 获取离线包
- 通过命令行（.bat或者.sh）执行

适用

① 生产环境

测试结果部分示例

```
【侦测场景1】测试全链路侦测...
侦测次数：10
侦测结果：[ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]
侦测结果：[ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]
侦测结果：[ID=discovery-guide-gateway][V=1.1] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]
侦测结果：[ID=discovery-guide-gateway][V=1.1] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]
侦测结果：[ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]
侦测结果：[ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]
侦测结果：[ID=discovery-guide-gateway][V=1.1] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]
侦测结果：[ID=discovery-guide-gateway][V=1.1] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]
侦测结果：[ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.0] -> [ID=discovery-guide-service-b][V=1.0]
侦测结果：[ID=discovery-guide-gateway][V=1.0] -> [ID=discovery-guide-service-a][V=1.1] -> [ID=discovery-guide-service-b][V=1.1]
测试耗时：0 秒
【侦测场景1】结束
```

详细资料点击此处

02 解决方案

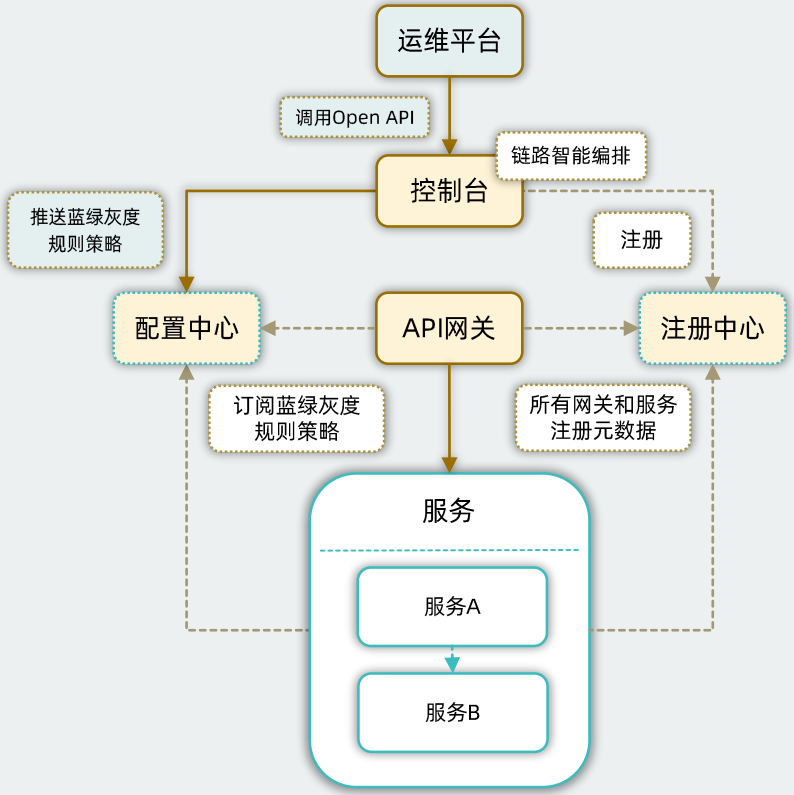
对接DevOps运维平台实施蓝绿灰度发布

架构

① 控制台需要连接注册中心和配置中心
② 控制台建议实现高可用架构，控制台前面部署API网关和运维平台对接

方案

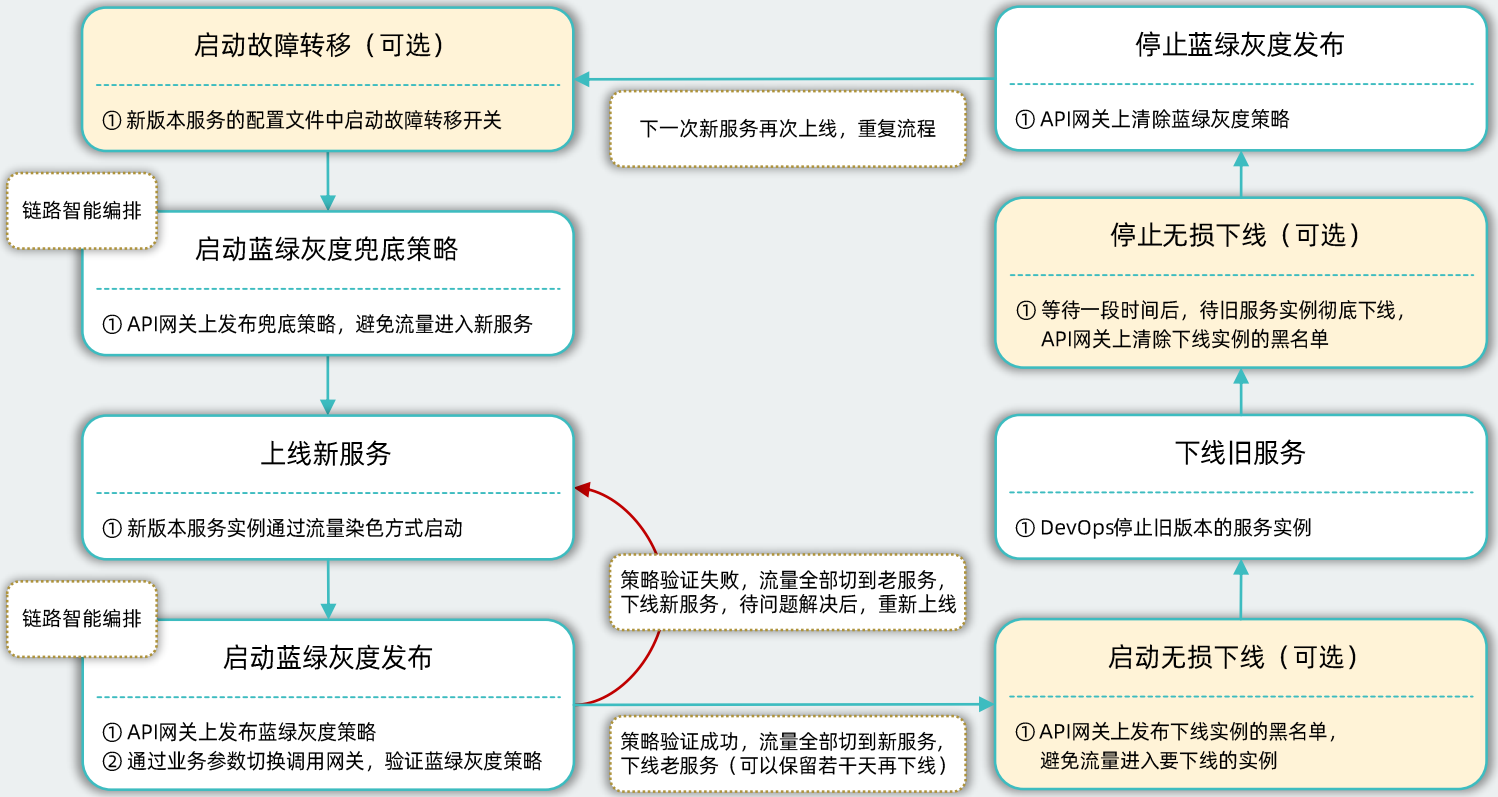
① 运维平台调用控制台的Open API，控制台进行链路智能编排
② 控制台把最终蓝绿灰度规则策略推送到配置中心



控制台部署点击此处

02 解决方案

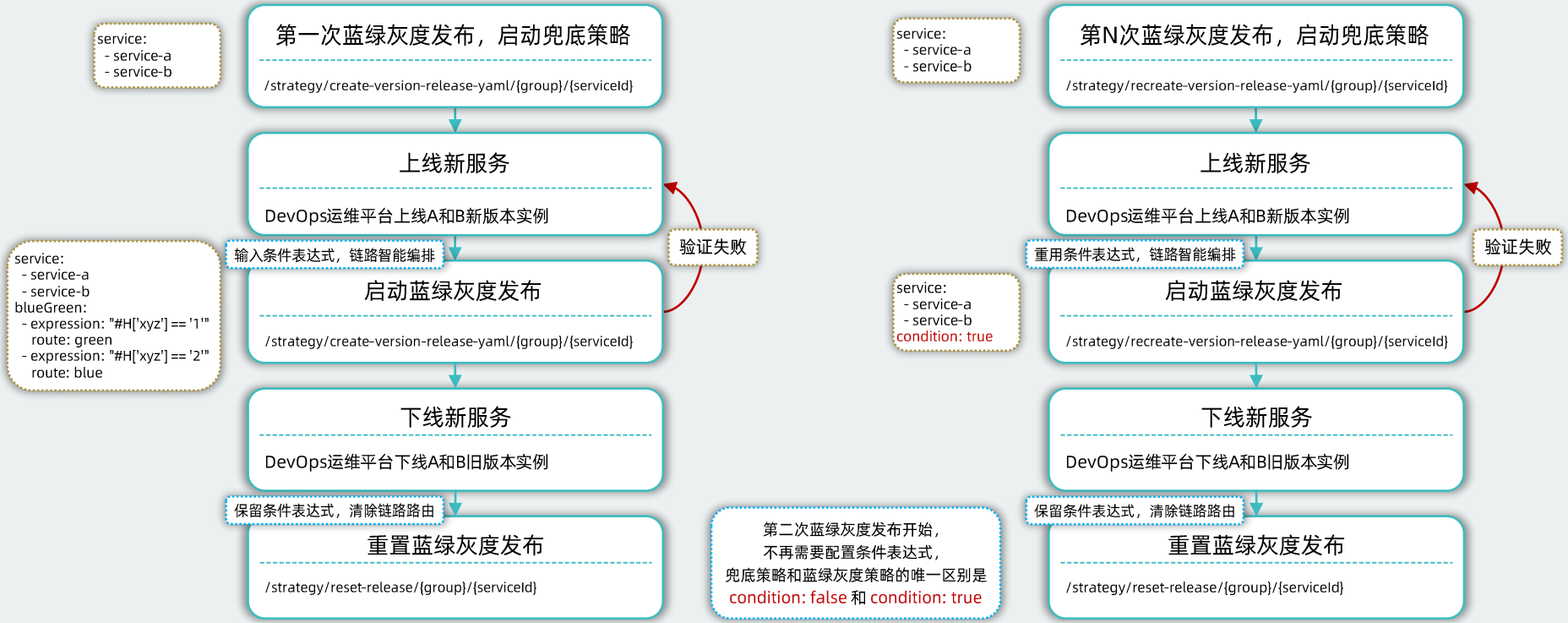
对接DevOps运维平台实施蓝绿灰度发布全流程



详细资料点击此处

02 解决方案

对接DevOps运维平台实施蓝绿灰度发布链路智能编排流程



02 解决方案

多活和单元化

单元

① 中心单元
- 部署在核心机房，机器性能，承载能力高
- 中心单元部署全局服务、核心服务和共享服务
- 中心单元是普通单元的特殊形式，限制一个

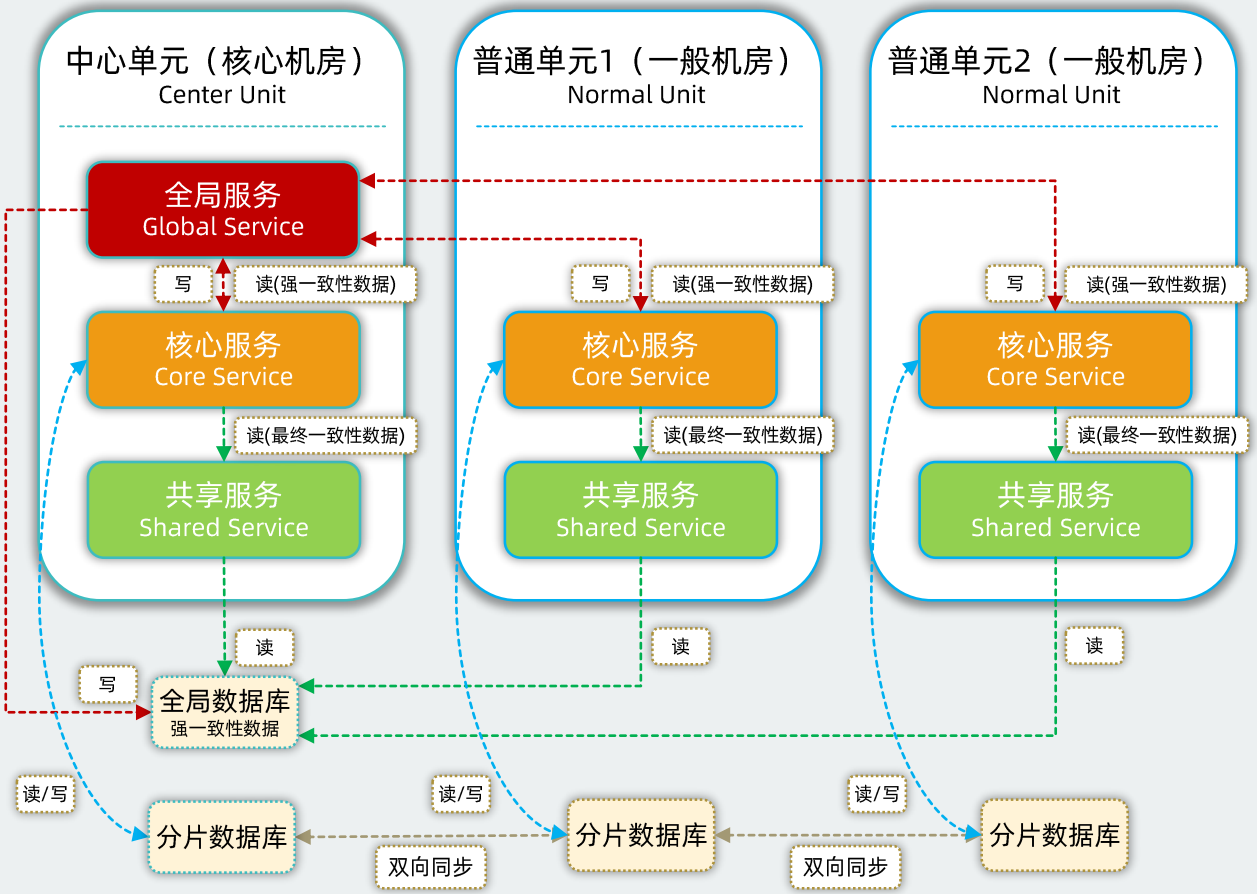
② 普通单元
- 部署在一般机房，机器性能，承载能力一般
- 中心单元部署核心服务和共享服务
- 普通单元可以水平扩容为N个

服务

① 全局服务
- 具有数据强一致性和实时性高要求
- 多活单元化拆分存在很大的难度
- 数据在中心单元写，中心单元读

② 共享服务
- 全局服务的代理服务，读服务
- 共享服务和全局服务实现读写分离。全局服务的最终一致性数据由共享服务暴露，在各自单元被核心服务读

③ 核心服务
- 多活单元化分片，按地域划分，就近原则访问
- 数据在各自单元写，各自单元读
- 全局服务的的强一致性数据，由全局服务直接暴露，被核心服务读



* 方案参考阿里巴巴《应用多活技术白皮书》的规范标准

02 解决方案

多活和单元化

服务配置

① 多活服务（主要是核心服务和共享服务），执行如下操作

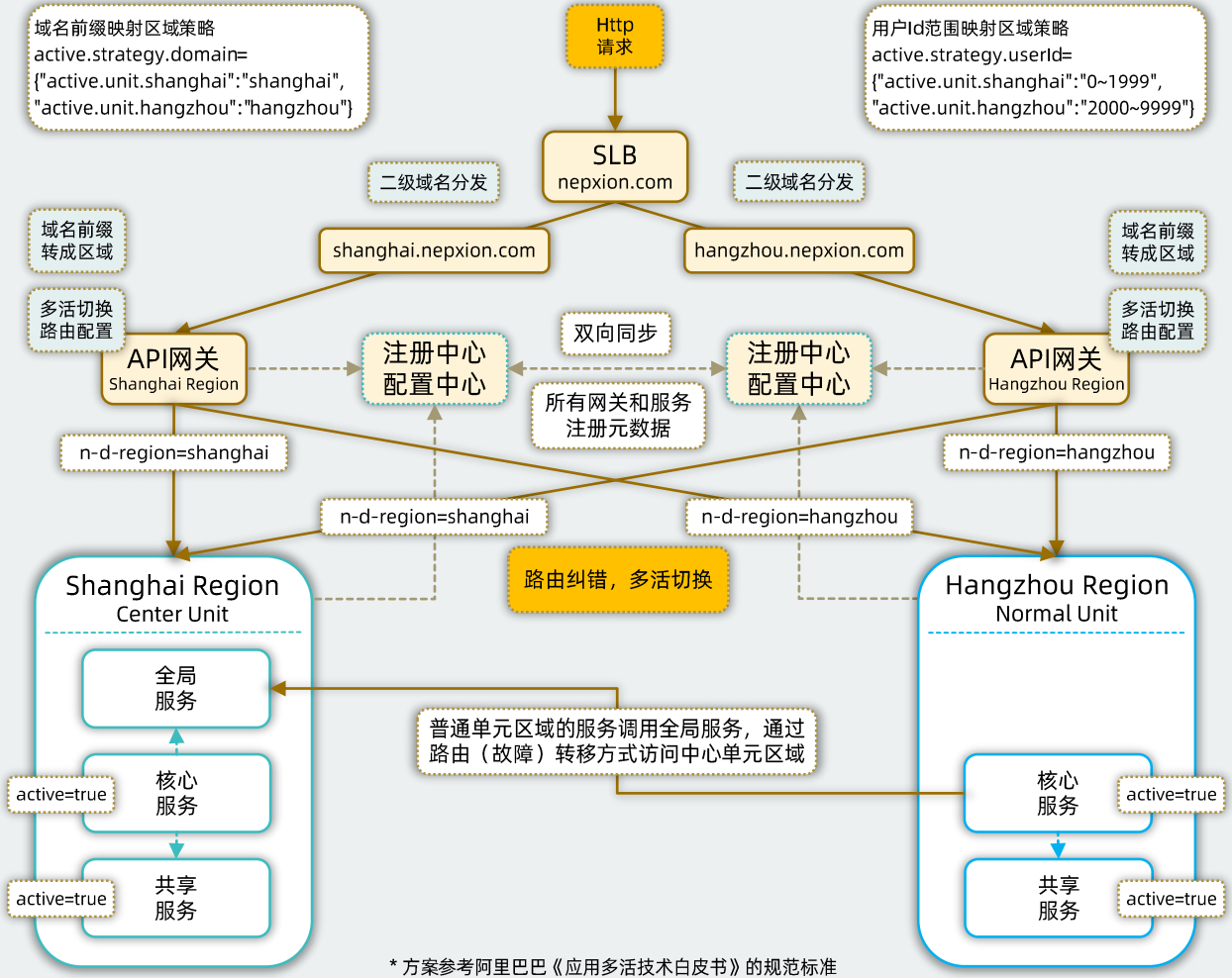
- 开启路由（故障）转移开关
- 标记元数据为多活属性（active=true）

流量分拨

① 前置的SLB或者下级Nginx根据请求IP进行二级域名分发
② API网关根据请求域名前缀转成区域，并赋值给Header n-d-region全链路传递，实现区域隔离路由

多活切换

配置多活切换的路由配置
① 域名前缀映射区域策略
② 用户Id范围映射区域策略
③ 自定义映射区域策略



* 方案参考阿里巴巴《应用多活技术白皮书》的规范标准

02 解决方案

全链路隔离路由

版本偏好路由

① 非蓝绿灰度发布场景下，路由到老的稳定版本的实例，或者指定版本的实例

区域调试路由

① 开发环境（个人电脑环境）与测试环境（线上环境）进行联调，支持开发环境和测试环境相互回访

② 开发环境通过区域Header值与服务实例区域值进行对比实现路由，测试环境通过开关控制实现隔离

环境隔离路由

① 服务实例的服务实例环境值和全链路传递的环境Header值进行对比实现隔离

可用区亲和性隔离路由

① 调用端和提供端的服务实例环境值进行对比实现隔离

名词解释

- 服务实例环境值：服务实例的元数据Metadata的env配置值，-Dmetadata.env
- 服务实例区域值：服务实例的元数据Metadata的region配置值，-Dmetadata.region
- 服务实例组值：服务实例的元数据Metadata的group配置值，-Dmetadata.group
- 环境Header值：n-d-env的Header值
- 区域Header值：n-d-region的Header值
- 组Header值： n-d-group的Header值

IP地址和端口隔离路由

① 调用端和提供端的IP地址和端口进行对比实现隔离

组隔离路由

① 基于组Header传值策略的提供端隔离，提供端服务实例组值和组Header值进行对比实现隔离

② 基于组负载均衡策略的消费端隔离，消费端和提供端服务实例组值进行对比实现隔离

注册发现隔离和准入

① 基于IP地址黑/白名单注册隔离和准入。服务实例的IP地址前缀在黑/白名单中，不允许/允许被注册到注册中心

② 基于IP地址黑/白名单服务发现。服务实例的IP地址前缀在黑/白名单中，不允许/允许被服务发现

③ 基于最大注册数限制隔离和准入。限制某服务实例注册到注册中心的最大数目

02 解决方案

全链路故障转移

版本故障转移

① 无法找到相应版本的服务实例，转移到指定版本的服务实例。三种策略：
老的稳定版本的实例，或者指定版本的实例，或者负载均衡

区域故障转移

① 无法找到相应区域的服务实例，转移到指定区域的服务实例。两种策略：
指定区域的实例，或者负载均衡

环境故障转移

① 无法找到相应环境的服务实例，转移到指定环境的实例。一种策略：
指定环境的实例，如果未指定，默认为common环境

可用区故障转移

① 无法找到相应可用区的服务实例，转移到指定可用区的服务实例。两种策略：
指定可用区的实例，或者负载均衡

IP地址和端口故障转移

①无法找到相应IP地址和端口的服务实例，转移到指定IP地址和端口的服务实例。两种策略：
指定IP地址和端口的实例，或者负载均衡

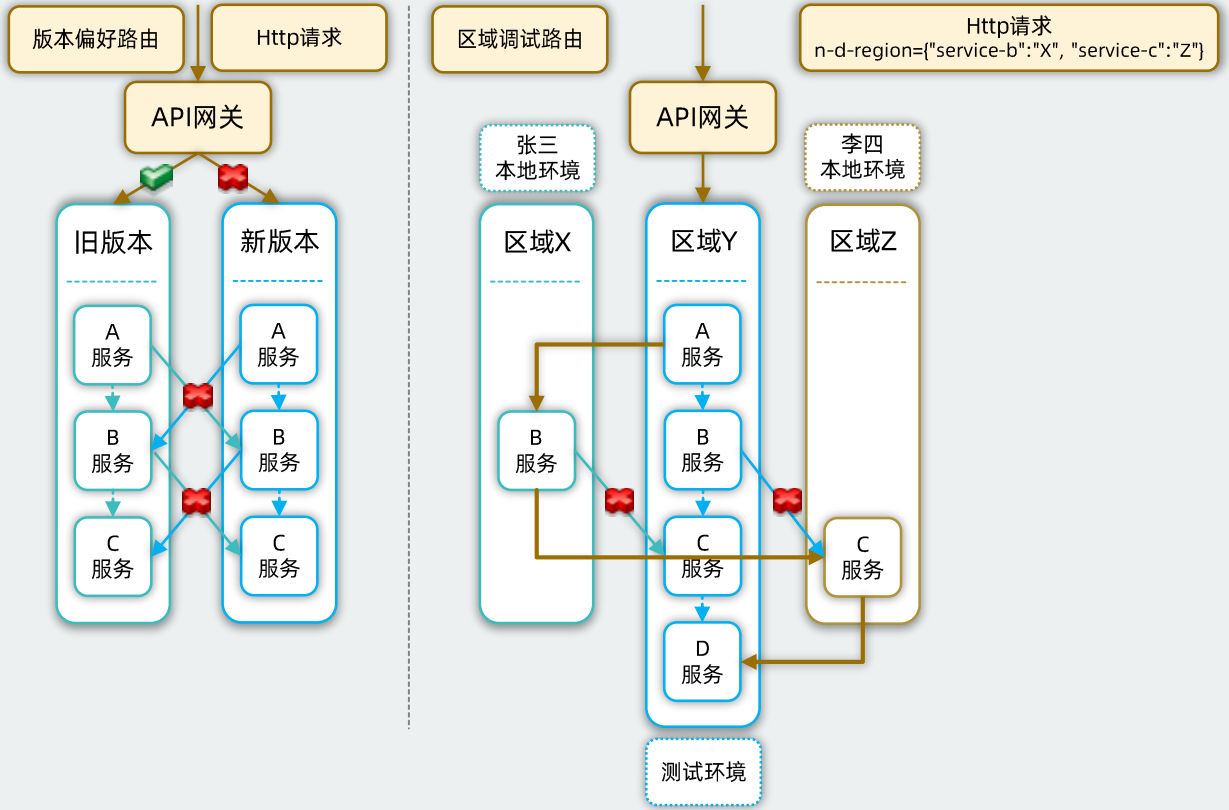
故障转移策略解释

- 版本故障转移：1. 故障转移策略已配置，执行配置的路由
  2. 故障转移策略未配置
     2.1 当spring.application.strategy.version.failover.stable.enabled=true，执行路由到老的稳定版本
     2.2 当spring.application.strategy.version.failover.stable.enabled=false，执行负载均衡
- 环境故障转移：1. 故障转移策略已配置，执行配置的路由
  2. 故障转移策略未配置，执行路由到common环境
- 其它故障转移：1. 故障转移策略已配置，执行配置的路由
  2. 故障转移策略未配置，执行负载均衡

02

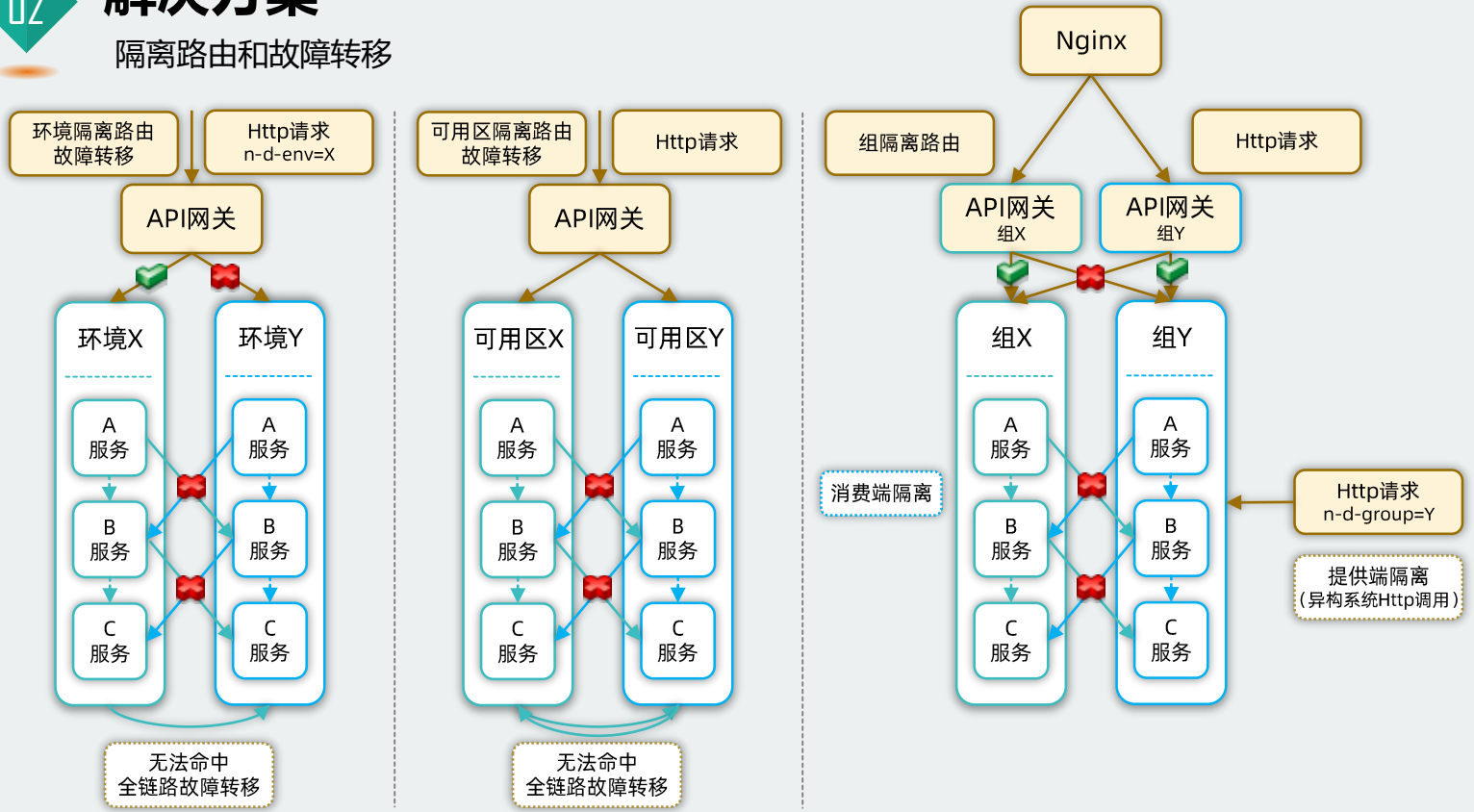
解决方案

偏好路由和调试路由



02 解决方案

隔离路由和故障转移



02 解决方案

接入Discovery【探索】框架的新服务和原生框架的旧服务兼容

场景痛点

- 生产环境中已经上线的旧服务已经使用了原生的Spring Cloud框架，元数据未配置
- 生产环境中即将上线的新服务使用了Discovery框架，元数据已配置
- 如何实现使用Discovery框架的新服务和原生框架的旧服务的兼容和过渡？

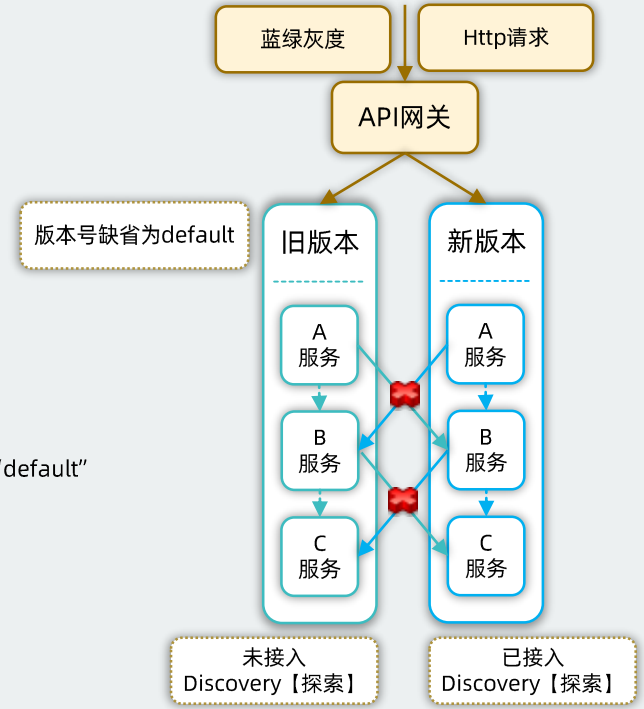
解决方案

归类为如下情景

- 新服务A -> 新服务B，支持蓝绿灰度发布
- 新服务A -> 旧服务B，支持蓝绿灰度发布，规则策略中旧服务版本号设置为“default”
- 旧服务A -> 旧服务B，不支持蓝绿灰度发布
- 旧服务A -> 新服务B，不支持蓝绿灰度发布

引申出其它场景

- 新服务A -> 新服务B（忘记配置元数据），支持蓝绿灰度发布，规则策略中服务B版本号设置为“default”



02

解决方案

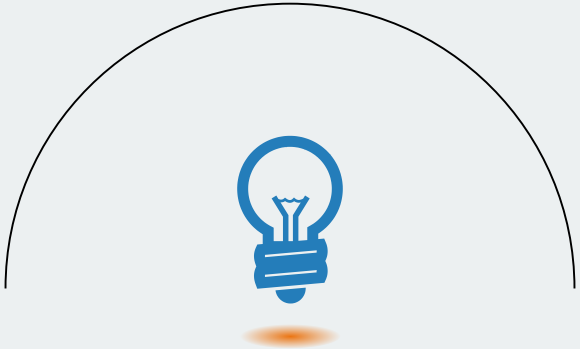
服务实例无损下线

解决方案

采用下线之前，把服务实例添加到屏蔽名单中，负载均衡不会去寻址该服务实例。下线之后，清除该名单。支持如下方式
① 全局唯一ID，根据上线时间屏蔽
② IP地址和端口

应用场景

服务下线场景下，由于Ribbon/Spring Cloud LoadBalancer负载均衡组件存在着缓存机制，当被提供端服务实例已经下线，而消费端服务实例还暂时缓存着它，直到下个心跳周期才会把已下线的服务实例剔除，在此期间，如果发生调用，会造成流量有损



实现途径

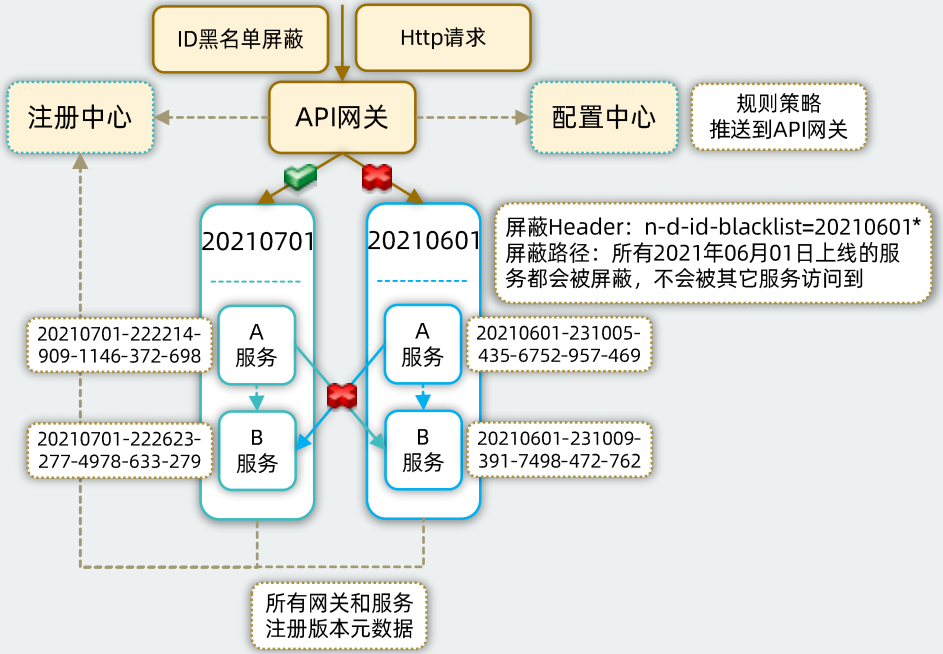
实现途径支持如下方式
① 通过运维平台调用控制台推送到配置中心

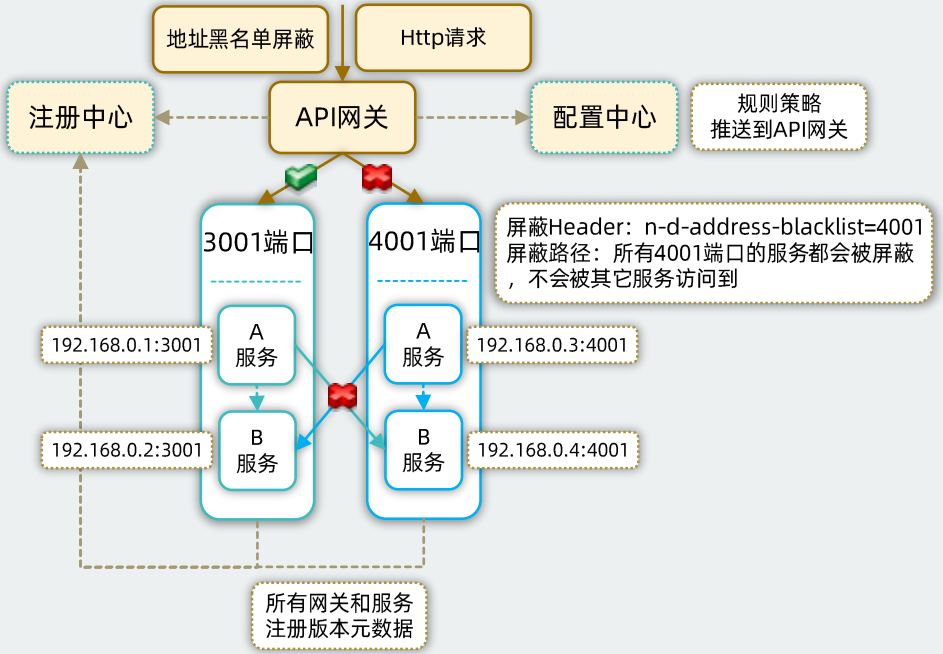
② 在配置中心界面配置

服务下线实时性的流量绝对无损

02 解决方案

服务实例无损下线





02 解决方案

异步场景下全链路蓝绿灰度发布

异步场景描述

寄存在ThreadLocal中的Header等上下文对象，在如下异步场景下，线程切换后，发生丢失情况

- WebFlux Reactor
- @Async
- Hystrix Thread Pool Isolation
- Runnable
- Callable
- Supplier
- Single Thread
- Thread Pool
- SLF4J MDC

通过引入异步跨线程DiscoveryAgent进行解决

引入

启动参数中引入Agent

```
-javaagent:/discovery-agent/discovery-agent-starter-${discovery.agent.version}.jar
-Dthread.scan.packages=com.abc.xyz
```

说明

启动参数说明

/discovery-agent
Agent所在的目录

-Dthread.scan.packages=com.abc.xyz
解决该目录下用户自定义Runnable/Callable/Thread/ThreadPool的异步
当用户未定义上述四个异步类，不需要扫描目录

更多内容参考

适用于一切Java技术栈的异步场景
https://github.com/Nepxion/DiscoveryAgent

* Spring cloud 2020以上（含），必须引入DiscoveryAgent
* Spring cloud Hoxton以下（含），如果含有异步调用场景，必须引入DiscoveryAgent

02 解决方案

用户自定义全场景功能

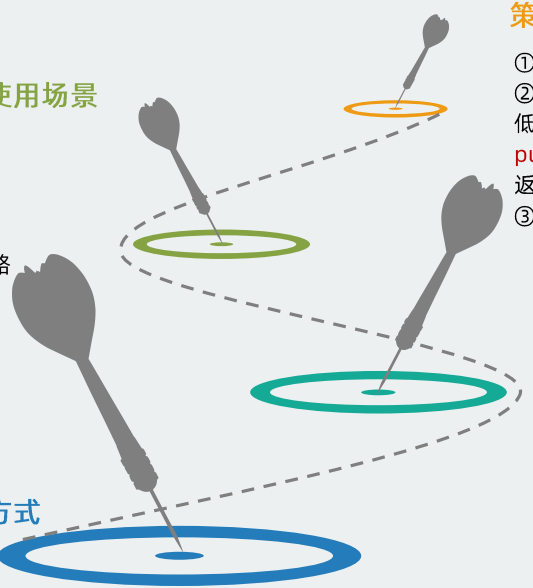
过滤器 RouteFilter 使用场景

① 全链路场景
② 用户自行控制自定义方式和配置方式的优先级
- 自定义根据Header全链路版本匹配路由
- 自定义根据Parameter全链路区域匹配路由
- 自定义根据Cookie全链路IP地址和端口匹配路由
- 自定义根据域名全链路环境隔离
- 自定义全链路版本权重路由
- 自定义外置Header绑定条件驱动参数

③ 只适用于网关

过滤器 RouteFilter 使用方式

- Zuul继承实现
  DefaultZuulStrategyRouteFilter
- Spring Cloud Gateway继承实现
  DefaultGatewayStrategyRouteFilter
- Service继承实现
  DefaultServiceStrategyRouteFilter



策略类 EnabledStrategy 使用场景

① 非全链路场景
② 在过滤器RouteFilter之后执行，优先级最低根据业务参数可以决定如下方法返回值：
public boolean apply(Server server);
返回false，即该服务实例被过滤掉
③ 适用于网关和服务

策略类 EnabledStrategy 使用方式

Zuul、 Spring Cloud Gateway、Service用法相同，继承实现
DefaultDiscoveryEnabledStrategy

02

解决方案

用户自定义全场景功能

No.1

过滤器 RouteFilter 使用示例

```
public class MyGatewayStrategyRouteFilter extends
    DefaultGatewayStrategyRouteFilter {
    @Value("${a.route.version")
    private String aRouteVersion = "{"service-a":"1.0", "service-b":"1.0"}";

    @Value("${b.route.version")
    private String bRouteVersion = "{"service-a":"1.1", "service-b":"1.1"}";

    // 自定义根据Header全链路版本匹配路由
    @Override
    public String getRouteVersion() {
        String user = strategyContextHolder.getHeader("user");
        // 张三用户，路由到a链路
        if (StringUtils.equals(user, "zhangsan")) {
            return aRouteVersion;
        // 李四用户，路由到b链路
        } else if (StringUtils.equals(user, "lisi")) {
            return bRouteVersion;
        }
        // 其它情况，执行内置的蓝绿灰度策略
        return super.getRouteVersion();
    }
}
```

No.2

策略类 EnabledStrategy 使用示例

```
public class MyDiscoveryEnabledStrategy extends
    DefaultDiscoveryEnabledStrategy {
    @Override
    public boolean apply(Server server) {
        String mobile = strategyContextHolder.getHeader("mobile");
        String version = pluginAdapter.getServerVersion(server);

        if (StringUtils.isNotEmpty(mobile)) {
            // 手机号以移动138开头，路由到1.0版本的服务上
            if (mobile.startsWith("138") && StringUtils.equals(version, "1.0")) {
                return true;
            // 手机号以联通133开头，路由到1.1版本的服务上
            } else if (mobile.startsWith("133") && StringUtils.equals(version, "1.1")) {
                return true;
            } else {
                // 其它情况，实例被过滤掉
                return false;
            }
        }
        // 无手机号，实例不被过滤掉
        return true;
    }
}
```

02

解决方案

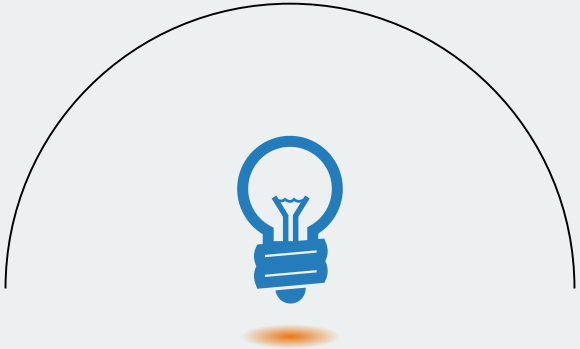
网关动态路由

特色功能

支持网关动态路由启用/禁用模式
持网关多实例动态路由一致性检查

技术特征

支持网关动态路由双写数据库和配置中心，采用类似Apollo版本控制模式，界面标识增/删/改标识，通过发布方式达到数据库和配置中心最终数据一致性



网关动态路由

实现途径

① Spring Cloud Gateway
支持内置断言器和过滤器
支持自定义断言器和过滤器

② Zuul
内置动态路由

02 解决方案

蓝绿灰度发布和Sentinel流量防护一体化

No.1 Sentinel原生支持

- 支持原生Sentinel注解
- 支持原生Sentinel规则
- 统一Sentinel和蓝绿灰度发布订阅的Key

No.2 Sentinel LimitApp内置扩展

基于服务名的防护 | 基于组的防护
基于版本的防护 | 基于区域的防护
基于环境的防护 | 基于可用区的防护
基于IP地址和端口的防护机制

No.3 Sentinel LimitApp自定义扩展

自定义Header、Parameter、Cookie的防护
自定义业务参数的防护
自定义组合式的防护

02 解决方案

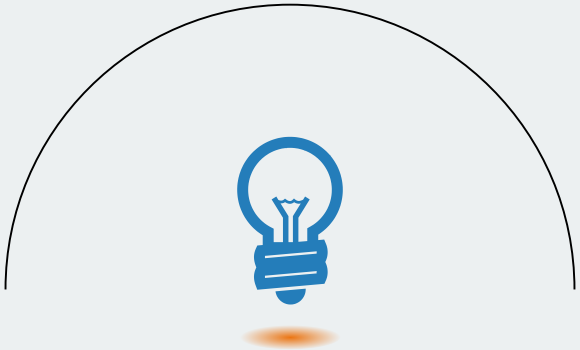
全链路监控

指标（Metrics）监控

① Sentinel熔断指标
1. Pass
2. Block
3. Success
4. Exception

调用链（Tracing）监控

① 基于Opentracing 和 OpenTelemetry 标准协议
② 整合SkyWalking 和 Jaeger
③ 调用链埋点输出
1. 内置蓝绿灰度埋点
2. 用户自定义外置埋点
3. Sentinel熔断埋点



可观测监控

日志（Logging）监控

① 蓝绿灰度参数输出到日志
② 蓝绿灰度Header输出到控制台
③ 蓝绿灰度失败告警，邮件通知
④ 蓝绿灰度埋点调试辅助监控
快速开启参数
-Dstrategy.debug=true

02 解决方案

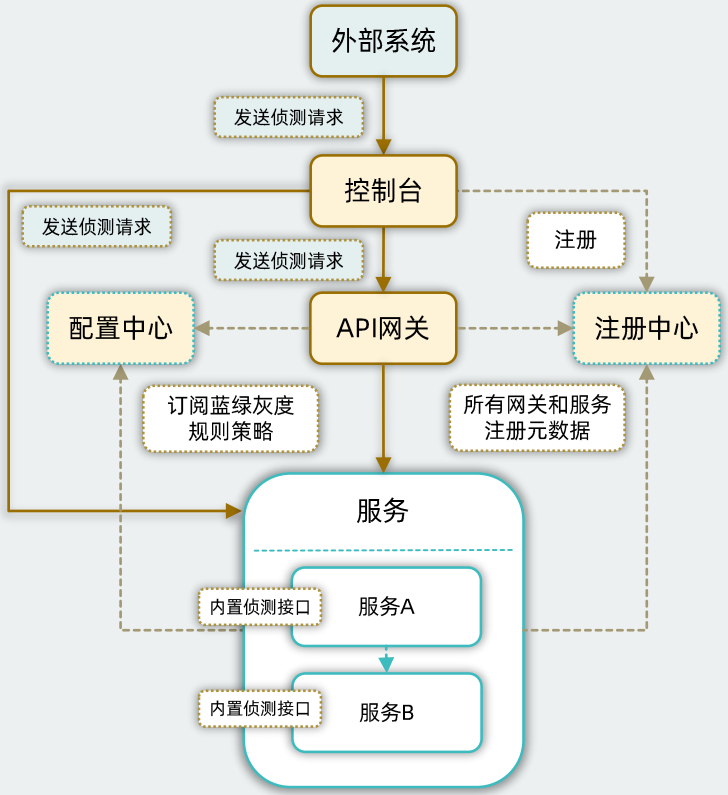
全链路侦测

目的

① 侦测调试蓝绿灰度发布、路由隔离等一系列流量管控手段是否符合预期

方案

① 调用控制台的Open API
② 控制台调用API网关或者服务
③ 业务服务引入discovery-plugin-admin-center-starter依赖



02 解决方案

元数据定义

基于系统参数

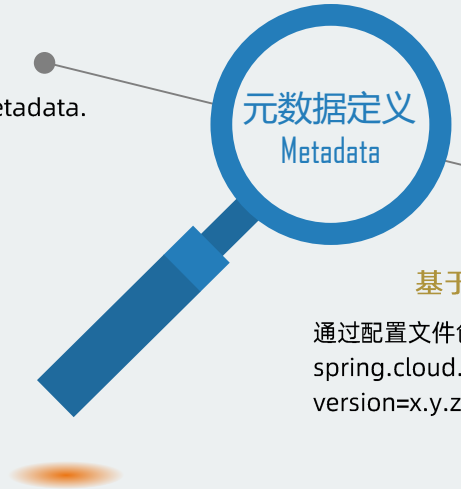
通过系统参数创建
System.setProperty(
"spring.cloud.discovery.metadata.version",
"x.y.z")

基于环境装载

通过EnvironmentPostProcessor环境初始化，装载
PluginMetaDataPreInstallation.
getMetadata().put("version", "x.y.z")

基于启动参数

通过启动参数创建
-Dmetadata.version=x.y.z
--spring.cloud.discovery.metadata.
version=xyz



基于配置文件

通过配置文件创建
spring.cloud.discovery.metadata.
version=x.y.z

基于Git插件

通过集成git-commit-id-plugin
插件自动创建metadata.version

* 以版本Version为例

02 解决方案

核心架构

集成主流服务注册中心中间件SDK
装饰Spring Cloud ServiceRegistry和ServerList
处理Spring Cloud Metadata
增强负载均衡机制

注册中心 Register Center

集成主流配置中心中间件SDK
适配配置订阅和反订阅接口
解析和反解析规则策略

配置中心 Config Center

暴露Rest和Swagger Endpoint接口
提供版本、配置、策略、侦测、路由、黑名单核心API
提供Sentinel、Git扩展API

管理中心 Admin Center

实现网关和服务的蓝绿灰度等编排功能
处理Header、Parameter、Cookie解析、拦截和传递
解决条件表达式和通配表达式逻辑
扩展服务实例列表过滤机制
提供隔离路由和故障转移功能
输出调用链和日志埋点

策略编排 Strategy

02 解决方案

监控架构

基于OpenTracing和OpenTelemetry规范
蓝绿灰度等Header（n-d-*开头）输出
服务关键信息（n-d-service-*开头）输出
业务Header可配置化输出

基于org.slf4j.MDC
蓝绿灰度等Header（n-d-*开头）输出
服务关键信息（n-d-service-*开头）输出
业务Header可配置化输出

基于Prometheus Micrometer
Sentinel Pass、Block、Success、Exception
指标统计输出

基于事件总线（EventBus）
蓝绿灰度等Header（n-d-*开头）输出
服务关键信息（n-d-service-*开头）输出
业务Header可配置化输出
结合DevOps系统发送告警邮件或者通知

调用链监控 Tracing

日志监控 Logging

指标监控 Metrics

上下文告警 Alarm

02

解决方案

平台架构

Nacos、Consul
Eureka、Zookeeper

注册中心 Register Center

Nacos、Apollo、Consul
Etcd、Redis、Zookeeper

配置中心 Config Center

MySQL、H2
MyBatis、Ldap

数据库 Database Plugin

链路编排、蓝绿发布、灰度发布、流量侦测
实例信息、实例摘除
Spring Cloud Gateway网关、Zuul网关动态路由
安全设置、基础应用
系统设置、授权配置
低代码平台、操作日志

核心功能 Core Function

新手入门

Pom引入

Zuul引入

```
<!-- 1.注册中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-register-center-starter-nacos</artifactId>
</dependency>

<!-- 2.配置中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-config-center-starter-nacos</artifactId>
</dependency>

<!-- 3.管理中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-admin-center-starter</artifactId>
</dependency>

<!-- 4.网关策略编排插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-strategy-starter-zuul</artifactId>
</dependency>
```

Spring Cloud Gateway引入

```
<!-- 1.注册中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-register-center-starter-nacos</artifactId>
</dependency>

<!-- 2.配置中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-config-center-starter-nacos</artifactId>
</dependency>

<!-- 3.管理中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-admin-center-starter</artifactId>
</dependency>

<!-- 4.网关策略编排插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-strategy-starter-gateway</artifactId>
</dependency>
```

Service引入

```
<!-- 1.注册中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-register-center-starter-nacos</artifactId>
</dependency>

<!-- 2.配置中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-config-center-starter-nacos</artifactId>
</dependency>

<!-- 3.管理中心插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-admin-center-starter</artifactId>
</dependency>

<!-- 4.服务策略编排插件-->
<dependency>
 <groupId>com.nepxion</groupId>
 <artifactId>discovery-plugin-strategy-starter-service</artifactId>
</dependency>
```

* 以Nacos注册中心和配置中心为例

新手入门

微服务元数据定义

No.1

四大服务注册中心的元数据

```
# 定义微服务的名称
spring.application.name=discovery-gray-service-a
# 定义微服务的端口号
server.port=3001

# 定义服务注册发现中心的元数据
# 定义微服务的所属组名
spring.cloud.discovery.metadata.group=discovery-gray-group
# 定义微服务的版本号
spring.cloud.discovery.metadata.version=1.0
# 定义微服务的所属区域
spring.cloud.discovery.metadata.region=dev
# 定义微服务的所属环境
spring.cloud.discovery.metadata.env=env1
# 定义微服务的所属可用区
spring.cloud.discovery.metadata.zone=zone1
#定义微服务的多活单元化
spring.cloud.discovery.metadata.active=true
```

新手入门

链路手工编排下蓝绿灰度规则策略示例

A

蓝绿发布规则策略简单示例

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy-release>
    <conditions type="blue-green">
      <condition id="condition-0" expression="#H['name'] == 'zhangsan'" version-id="route-0"/>
      <condition id="condition-1" expression="#H['name'] == 'lisi'" version-id="route-1"/>
      <condition id="condition-2" version-id="route-1"/>
    </conditions>

    <routes>
      <route id="route-0" type="version">{"service-a":"2.0", "service-b":"2.0"}</route>
      <route id="route-1" type="version">{"service-a":"1.0", "service-b":"1.0"}</route>
    </routes>
  </strategy-release>
</rule>
```

B

灰度发布规则策略简单示例

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy-release>
    <conditions type="gray">
      <condition id="condition-0" expression="#H['name'] == 'zhangsan'" version-id="route-0=10;route-1=90"/>
      <condition id="condition-1" expression="#H['name'] == 'lisi'" version-id="route-0=90;route-1=10"/>
      <condition id="condition-2" version-id="route-0=0;route-1=100"/>
    </conditions>

    <routes>
      <route id="route-0" type="version">{"service-a":"2.0", "service-b":"2.0"}</route>
      <route id="route-1" type="version">{"service-a":"1.0", "service-b":"1.0"}</route>
    </routes>
  </strategy-release>
</rule>
```

03 新手入门

链路智能编排下蓝绿灰度规则策略示例

A

蓝绿发布规则策略简单示例

```
service:
  - service-a
  - service-b
blueGreen:
  - expression: "#H['name'] == 'zhangsan'"
    route: green
  - expression: "#H['name'] == 'lisi'"
    route: blue
sort: version
```

```
{
  "service": ["service-a", "service-b"],
  "blueGreen": [
  {
     "expression": "#H['name'] == 'zhangsan'",
     "route": "green"
  },
  {
     "expression": "#H['name'] == 'lisi'",
     "route": "blue"
  }
  ],
  "sort": "version"
}
```

B

灰度发布规则策略简单示例

```
service:
  - service-a
  - service-b
gray:
  - expression: "#H['name'] == 'zhangsan'"
    weight:
      - 90
      - 10
  - expression: "#H['name'] == 'lisi'"
    weight:
      - 70
      - 30
  - weight:
      - 100
      - 0
sort: version
```

```
{
  "service": ["service-a", "service-b"],
  "gray": [
  {
     "expression": "#H['name'] == 'zhangsan'",
     "weight": [90, 10]
  },
  {
     "expression": "#H['name'] == 'lisi'",
     "weight": [70, 30]
  },
  {
     "weight": [100, 0]
  }
  ],
  "sort": "version"
}
```

新手入门

链路无编排下蓝绿灰度规则策略示例

A

蓝绿发布规则策略简单示例

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy-release>
    <conditions type="blue-green">
      <condition id="condition-0" expression="#H['name'] == 'zhangsan'" version-id="route-0"/>
      <condition id="condition-1" expression="#H['name'] == 'lisi'" version-id="route-1"/>
    </conditions>

    <routes>
      <route id="route-0" type="version">blue</route>
      <route id="route-1" type="version">green</route>
    </routes>
  </strategy-release>
</rule>
```

B

灰度发布规则策略简单示例

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy-release>
    <conditions type="gray">
      <condition id="condition-0" expression="#H['name'] == 'zhangsan'" version-id="route-0=10;route-1=90"/>
      <condition id="condition-1" expression="#H['name'] == 'lisi'" version-id="route-0=90;route-1=10"/>
    </conditions>

    <routes>
      <route id="route-0" type="version">blue</route>
      <route id="route-1" type="version">green</route>
    </routes>
  </strategy-release>
</rule>
```

03 新手入门

故障转移和无损下线规则策略示例

A

故障转移规则策略简单示例

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy-failover>
    <version-prefer>{"service-a":"1.0", "service-b":"1.0"}</version-prefer>
    <version-failover>{"service-a":"1.1", "service-b":"1.1"}</version-failover>
    <region-transfer>qa</region-transfer>
    <region-failover>dev</region-failover>
    <env-failover>common</env-failover>
    <zone-failover>zone1</zone-failover>
    <address-failover>*1</address-failover>
  </strategy-failover>
</rule>
```

B

无损下线黑名单规则策略简单示例

```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
  <strategy-blacklist>
    <id>{"service-a":"20210601*", "service-b":"20210601*"}</id>
    <address>127.0.0.1:3001</address>
  </strategy-blacklist>
</rule>
```

04 文档指南

文档指南和联系方式

主页地址：http://www.nepxion.com
源码地址：https://github.com/Nepxion/Discovery
镜像地址：https://gitee.com/Nepxion/Discovery
指南地址：https://github.com/Nepxion/DiscoveryGuide
框架文档：https://nepxion.com/discovery
平台文档：https://nepxion.com/discovery-platform
桌面文档：https://nepxion.com/discovery-desktop

微信

钉钉

公众号

文档

Thanks for Watching

Discovery【探索】