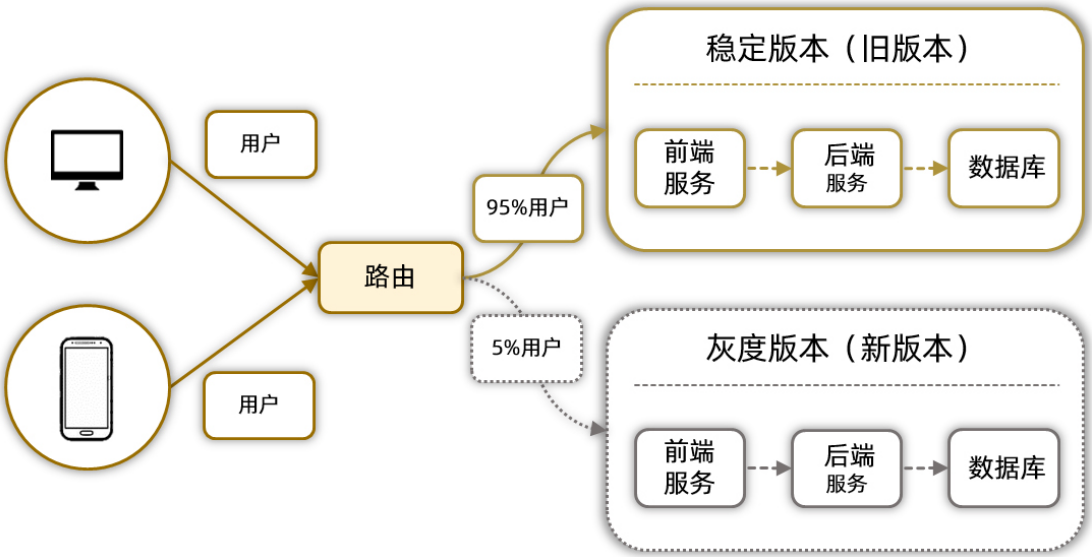
主讲老师：Fox

> **有道笔记地址：** https://note.youdao.com/s/7yNVS8iG

1. 灰度发布

1.1 什么是灰度发布

灰度发布 Gray Release（又名金丝雀发布 Canary Release）。比如不停机旧版本，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，通过监控观察无问题，逐步扩大范围，最终把所有流量都迁移到新版本上。



常见使用场景

- 切分一定比例的流量到新版本
- 灰度新版本到部分用户，通常基于 Header 或 Cookie 进行流量的策略来实现

1.2 基于云原生网关Higress 实现灰度发布

Higress提供复杂的路由处理能力，支持基于Header、Cookie以及权重的灰度发布功能。灰度发布功能可以通过设置注解来实现，为了启用灰度发布功能，需要设置注解higress.io/canary: "true"。通过不同注解可以实现不同的灰度发布功能。

> 说明：当多种方式同时配置时，灰度方式选择优先级为：基于Header > 基于Cookie > 基于权重（从高到低）。

部署两个版本的服务

1）在集群中部署第一个版本的 Deployment，本文以 nginx-v1 为例。YAML 示例如下：

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: nginx-v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
      version: v1
  template:
    metadata:
      labels:
        app: nginx
        version: v1
    spec:
      containers:
      - name: nginx
        image: "openresty/openresty:centos"
        ports:
        - name: http
          protocol: TCP
          containerPort: 80
        volumeMounts:
        - mountPath: /usr/local/openresty/nginx/conf/nginx.conf
          name: config
          subPath: nginx.conf
      volumes:
      - name: config
        configMap:
          name: nginx-v1

---

apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: nginx
    version: v1
  name: nginx-v1
```

```
data:
  nginx.conf: |-
    worker_processes  1;

    events {
        accept_mutex on;
        multi_accept on;
        use epoll;
        worker_connections  1024;
    }

    http {
        ignore_invalid_headers off;
        server {
            listen 80;
            location / {
                access_by_lua '
                    local header_str = ngx.say("nginx-v1")
                ';
            }
            location /hello {
                access_by_lua '
                    local header_str = ngx.say("hello nginx-v1")
                ';
            }
        }
    }

---

apiVersion: v1
kind: Service
metadata:
  name: nginx-v1
spec:
  type: ClusterIP
  ports:
  - port: 80
    protocol: TCP
```

```
    name: http
  selector:
    app: nginx
    version: v1
```

2）再部署第二个版本的 Deployment，本文以 nginx-v2 为例。YAML 示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
      version: v2
  template:
    metadata:
      labels:
        app: nginx
        version: v2
    spec:
      containers:
      - name: nginx
        image: "openresty/openresty:centos"
        ports:
        - name: http
          protocol: TCP
          containerPort: 80
        volumeMounts:
        - mountPath: /usr/local/openresty/nginx/conf/nginx.conf
          name: config
          subPath: nginx.conf
      volumes:
      - name: config
        configMap:
          name: nginx-v2
```

```
---

apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: nginx
    version: v2
  name: nginx-v2
data:
  nginx.conf: |-
    worker_processes  1;

    events {
        accept_mutex on;
        multi_accept on;
        use epoll;
        worker_connections  1024;
    }

    http {
        ignore_invalid_headers off;
        server {
            listen 80;
            location / {
                access_by_lua '
                    local header_str = ngx.say("nginx-v2")
                ';
            }
            location /hello {
                access_by_lua '
                    local header_str = ngx.say("hello nginx-v2")
                ';
            }
        }
    }

---

apiVersion: v1
```

```
kind: Service
metadata:
  name: nginx-v2
spec:
  type: ClusterIP
  ports:
  - port: 80
    protocol: TCP
    name: http
  selector:
    app: nginx
    version: v2
```

```
[root@k8s-master01 higress-demo]# kubectl get svc|grep nginx
nginx-v1                      ClusterIP   10.96.1.196    <none>        80/TCP           10m
nginx-v2                      ClusterIP   10.96.1.99     <none>        80/TCP           9m42s
```

基于Header灰度发布

- 只配置higress.io/canary-by-header：基于Request Header的名称进行流量切分。当请求包含该Header并其值为always时，请求流量会被分配到灰度服务入口；其他情况时，请求流量不会分配到灰度服务。
- 同时配置higress.io/canary-by-header和higress.io/canary-by-header-value：基于Request Header的名称和值进行流量切分。当请求中的header的名称和header的值与该配置匹配时，请求流量会被分配到灰度服务；其他情况时，请求流量不会分配到灰度服务。

1. 请求Header为higress: always时将访问灰度服务nginx-v2；其他情况将访问正式服务nginx-v1，配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    higress.io/canary: "true"
    higress.io/canary-by-header: "higress"
  name: higress-demo-canary
spec:
  ingressClassName: higress
  rules:
    - http:
        paths:
```

```
          - backend:
              service:
                name: nginx-v2
                port:
                  number: 80
            path: /hello
            pathType: Exact
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: higress-demo
spec:
  ingressClassName: higress
  rules:
    - http:
        paths:
          - backend:
              service:
                name: nginx-v1
                port:
                  number: 80
            path: /hello
            pathType: Exact
```

部署后执行以下命令进行测试

```
curl  -H "higress: always" http://10.96.1.19/hello
```

```
[root@k8s-master01 higress-demo]# kubectl get svc -n higress-system
NAME                 TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)
higress-console      NodePort       10.96.0.191   <none>        8080:30363/TCP
higress-controller   ClusterIP      10.96.3.186   <none>        8888/TCP,15051/TCP,15010/TCP,15012/TCP,443/TCP,15014/TCP
higress-gateway      LoadBalancer   10.96.1.19    <pending>     80:30332/TCP,443:31448/TCP
[root@k8s-master01 higress-demo]# curl  -H "higress: always" http://10.96.1.19/hello
hello nginx-v2
[root@k8s-master01 higress-demo]# curl  -H "higress: always" http://10.96.1.19/hello
hello nginx-v2
[root@k8s-master01 higress-demo]# curl   http://10.96.1.19/hello
hello nginx-v1
[root@k8s-master01 higress-demo]# curl   http://10.96.1.19/hello
hello nginx-v1
```

2. 请求Header为higress: v2时将访问灰度服务nginx-v2；其他情况将访问正式服务nginx-v1，配置如下：

```
apiVersion: networking.k8s.io/v1
```

```yaml
kind: Ingress
metadata:
  annotations:
    higress.io/canary: "true"
    higress.io/canary-by-header: "higress"
    higress.io/canary-by-header-value: "v1"
  name: higress-demo-canary
spec:
  ingressClassName: higress
  rules:
    - http:
        paths:
          - backend:
              service:
                name: nginx-v2
                port:
                  number: 80
            path: /hello
            pathType: Exact

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: higress-demo
spec:
  ingressClassName: higress
  rules:
    - http:
        paths:
          - backend:
              service:
                name: nginx-v1
                port:
                  number: 80
            path: /hello
            pathType: Exact
```

部署后执行以下命令进行测试

```
curl  -H "higress: v2" http://10.96.1.19/hello
```

```
[root@k8s-master01 higress-demo]# curl  -H "higress: v1" http://10.96.1.19/hello
hello nginx-v1
[root@k8s-master01 higress-demo]# curl  -H "higress: v2" http://10.96.1.19/hello
hello nginx-v2
[root@k8s-master01 higress-demo]# curl  -H "higress: v2" http://10.96.1.19/hello
hello nginx-v2
[root@k8s-master01 higress-demo]# curl  -H "higress: always" http://10.96.1.19/hello
hello nginx-v2
[root@k8s-master01 higress-demo]# curl  -H "higress: always" http://10.96.1.19/hello
hello nginx-v2
[root@k8s-master01 higress-demo]# curl  -H "higress: always" http://10.96.1.19/hello
hello nginx-v2
[root@k8s-master01 higress-demo]# curl  -H "higress: always1" http://10.96.1.19/hello
hello nginx-v1
```

基于Cookie灰度发布

- higress.io/canary-by-cookie：基于Cookie的Key进行流量切分。当请求的Cookie中含有该Key且其值为always时，请求流量将被分配到灰度服务；其他情况时，请求流量将不会分配到灰度服务。

> 说明：基于Cookie的灰度发布不支持自定义设置Key对应的值，只能是always。

请求的Cookie为demo=always时将访问灰度服务nginx-v2；其他情况将访问正式服务nginx-v1。配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    higress.io/canary: "true"
    higress.io/canary-by-cookie: "demo"
  name: higress-demo-canary
spec:
  ingressClassName: higress
  rules:
    - http:
        paths:
          - backend:
              service:
                name: nginx-v2
                port:
                  number: 80
            path: /hello
```

```
            pathType: Exact
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: higress-demo
spec:
  ingressClassName: higress
  rules:
    - http:
        paths:
          - backend:
              service:
                name: nginx-v1
                port:
                  number: 80
            path: /hello
            pathType: Exact
```

部署后执行以下命令进行测试

```
curl  --cookie "demo=always" http://10.96.1.19/hello
```

```
[root@k8s-master01 higress-demo]# kubectl delete -f header-canary2.yaml
ingress.networking.k8s.io "higress-demo-canary" deleted
ingress.networking.k8s.io "higress-demo" deleted
[root@k8s-master01 higress-demo]# kubectl apply -f cookie-canary.yaml
ingress.networking.k8s.io/higress-demo-canary created
ingress.networking.k8s.io/higress-demo created
[root@k8s-master01 higress-demo]# curl  --cookie "demo=always" http://10.96.1.19/hello
hello nginx-v2
[root@k8s-master01 higress-demo]# curl   http://10.96.1.19/hello
hello nginx-v1
[root@k8s-master01 higress-demo]# curl  --cookie "demo=always" http://10.96.1.19/hello
hello nginx-v2
```

基于权重灰度发布

- higress.io/canary-weight：设置请求到指定服务的百分比（值为0~100的整数）
- higress.io/canary-weight-totatl：设置权重总和，默认为100

配置灰度服务nginx-v2的权重为30%，配置正式服务nginx-v1的权重为70%。

```
apiVersion: networking.k8s.io/v1
```

```yaml
kind: Ingress
metadata:
  annotations:
    higress.io/canary: "true"
    higress.io/canary-weight: "30"
  name: higress-demo-canary
spec:
  ingressClassName: higress
  rules:
    - http:
        paths:
          - backend:
              service:
                name: nginx-v2
                port:
                  number: 80
            path: /hello
            pathType: Exact

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: higress-demo
spec:
  ingressClassName: higress
  rules:
    - http:
        paths:
          - backend:
              service:
                name: nginx-v1
                port:
                  number: 80
            path: /hello
            pathType: Exact
```

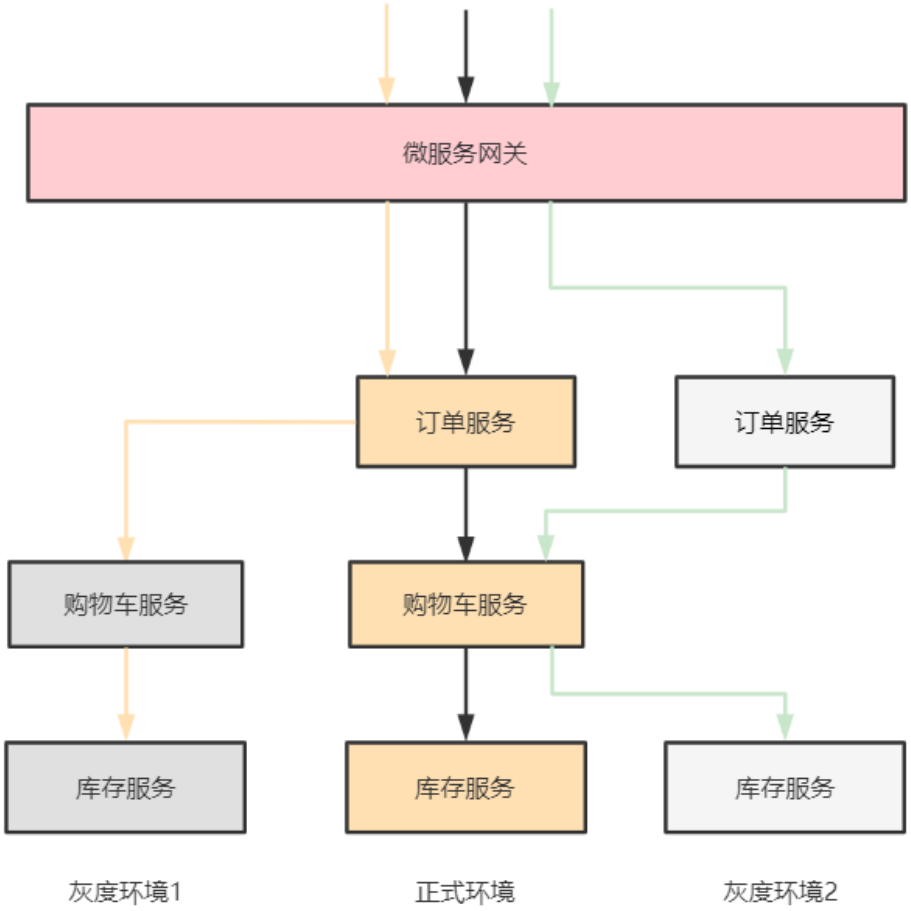
部署后执行以下命令进行测试

```
for i in {1..10}; do curl -H "Host: canary.example.com" http://10.96.1.19/hello; done;
```

```
[root@k8s-master01 higress-demo]# kubectl apply -f weight-canary.yaml
ingress.networking.k8s.io/higress-demo-canary created
ingress.networking.k8s.io/higress-demo created
[root@k8s-master01 higress-demo]# for i in {1..10}; do curl  http://10.96.1.19/hello; done;
hello nginx-v2
hello nginx-v1
hello nginx-v1
hello nginx-v2
hello nginx-v2
hello nginx-v1
hello nginx-v1
hello nginx-v1
hello nginx-v1
hello nginx-v2
[root@k8s-master01 higress-demo]# for i in {1..10}; do curl  http://10.96.1.19/hello; done;
hello nginx-v1
hello nginx-v1
hello nginx-v1
hello nginx-v2
hello nginx-v1
hello nginx-v2
```

1.3 什么是微服务全链路灰度

微服务体系架构中，服务之间的依赖关系错综复杂，有时某个功能发版依赖多个服务同时升级上线。我们希望可以对这些服务的新版本同时进行小流量灰度验证，这就是微服务架构中特有的全链路灰度场景，通过构建从网关到整个后端服务的环境隔离来对多个不同版本的服务进行灰度验证。在发布过程中，我们只需部署服务的灰度版本，流量在调用链路上流转时，由流经的网关、各个中间件以及各个微服务来识别灰度流量，并动态转发至对应服务的灰度版本。如下图：



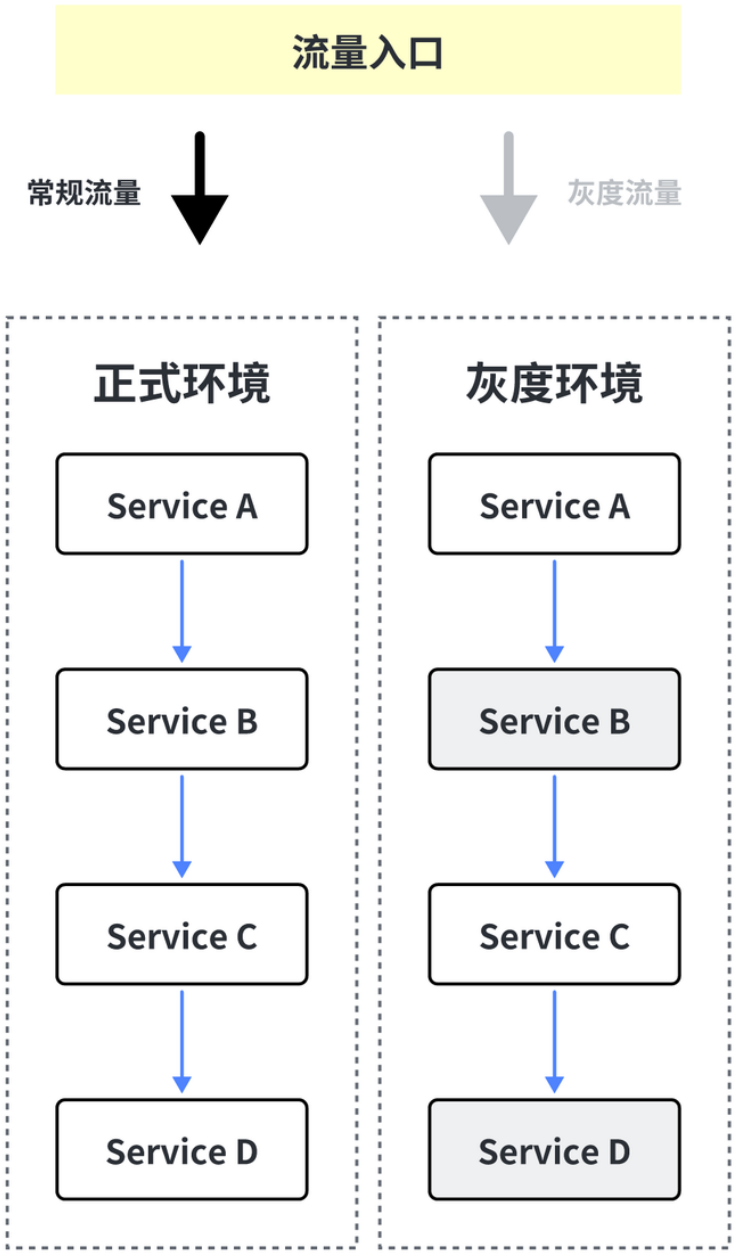
上图可以很好展示这种方案的效果，我们用不同的颜色来表示不同版本的灰度流量，可以看出无论是微服务网关还是微服务本身都需要识别流量，根据治理规则做出动态决策。当服务版本发生变化时，这个调用链路的转发也会实时改变。相比于利用机器搭建的灰度环境，这种方案不仅可以节省大量的机器成本和运维人力，而且可以帮助开发者实时快速的对线上流量进行精细化的全链路控制。

2. 全链路灰度设计思路

如何在实际业务场景中去快速落地全链路灰度呢？目前，主要有两种解决思路，基于**完整环境隔离**和基于**服务流量路由**。

2.1 基于完整环境隔离

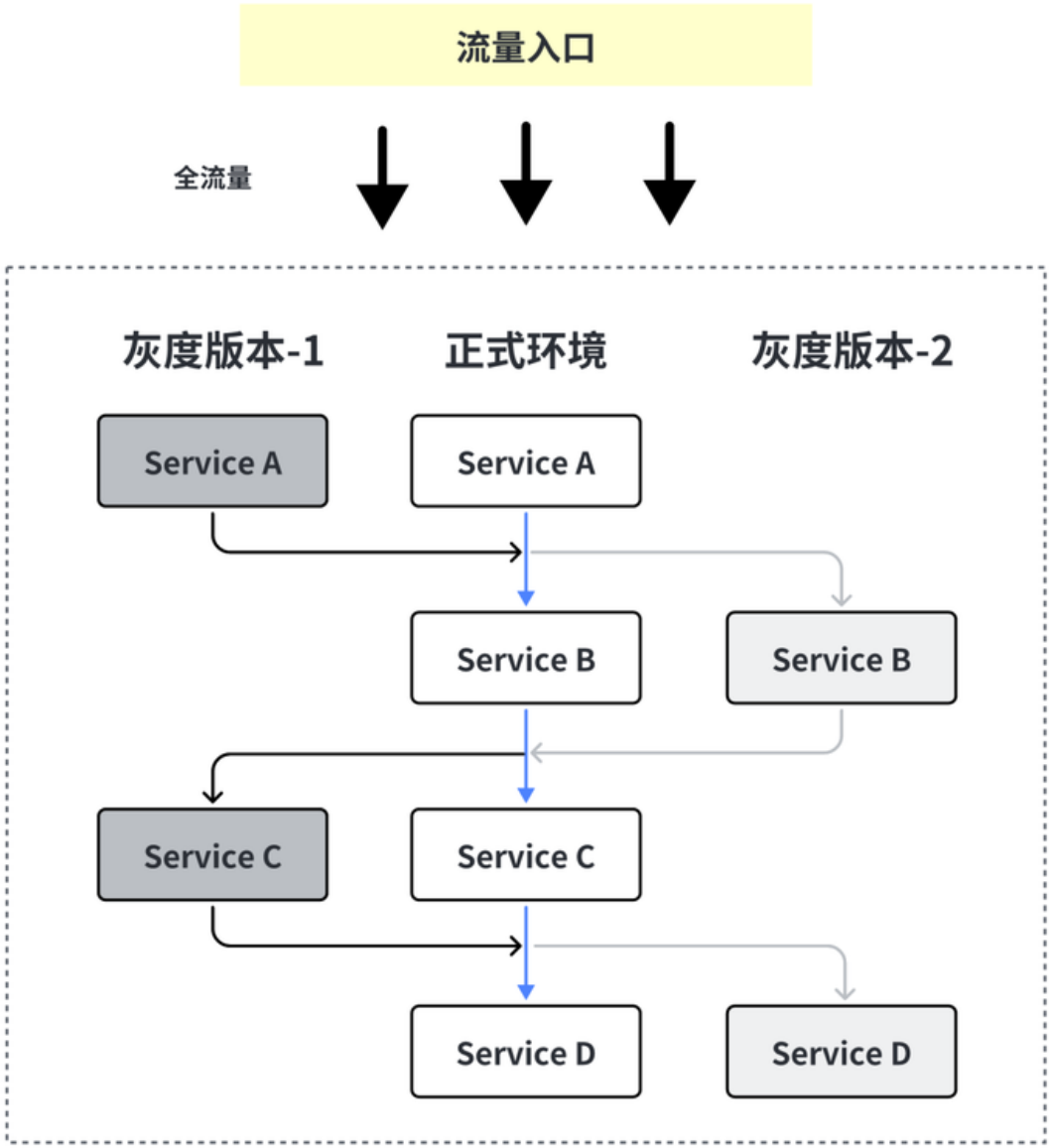
这种方案需要为要灰度的服务搭建一套网络隔离、资源独立的环境，在其中部署服务的灰度版本。由于与正式环境隔离，正式环境中的其他服务无法访问到需要灰度的服务，所以需要在灰度环境中冗余部署这些线上服务，以便整个调用链路正常进行流量转发。



这个方案一般用于企业的测试、预发开发环境的搭建，对于线上灰度发布引流的场景来说其灵活性不够。况且，微服务多版本的存在在微服务架构中是家常便饭，需要为这些业务场景采用堆机器的方式来 维护多套灰度环境。如果您的应用数目过多的情况下，会造成运维、机器成本过大，成本和代价远超收益；如果应用数目很小，就两三个应用，这个方式还是很方便的，可以接受的。

2.2 基于服务流量路由

我们只需部署服务的灰度版本，流量在调用链路上流转时，由流经的网关、各个中间件以及各个微服务来识别灰度流量，并动态转发到对应服务的灰
度版本。



全链路流量路由目前有两种主流实现：

1. **基于服务发现组件**：通过支持为服务设置元数据的服务注册中心，如 Nacos，可以标记服务实例的特征，例如灰度版本。每个服务可以通过注册中心获取其他服务实例的版本信息，并通过修改代码逻辑或 Java Agent 实现流量路由。
2. **基于 Istio**：采用 Istio 这个开源 Service Mesh 组件，通过在每个服务的容器中部署 Envoy 透明代理，拦截服务之间的网络通信并按指定规则转发，从而实现了全链路流量路由，无需对现有代码进行修改。

要想实现全链路灰度，我们需要解决以下问题：

1.链路上各个组件和服务能够根据请求流量特征进行动态路由。

2.需要对服务下的所有节点进行分组，能够区分版本。

3.需要对流量进行灰度标识、版本标识。

4.需要识别出不同版本的灰度流量。

接下来，会介绍解决上述问题需要用到的技术

标签路由

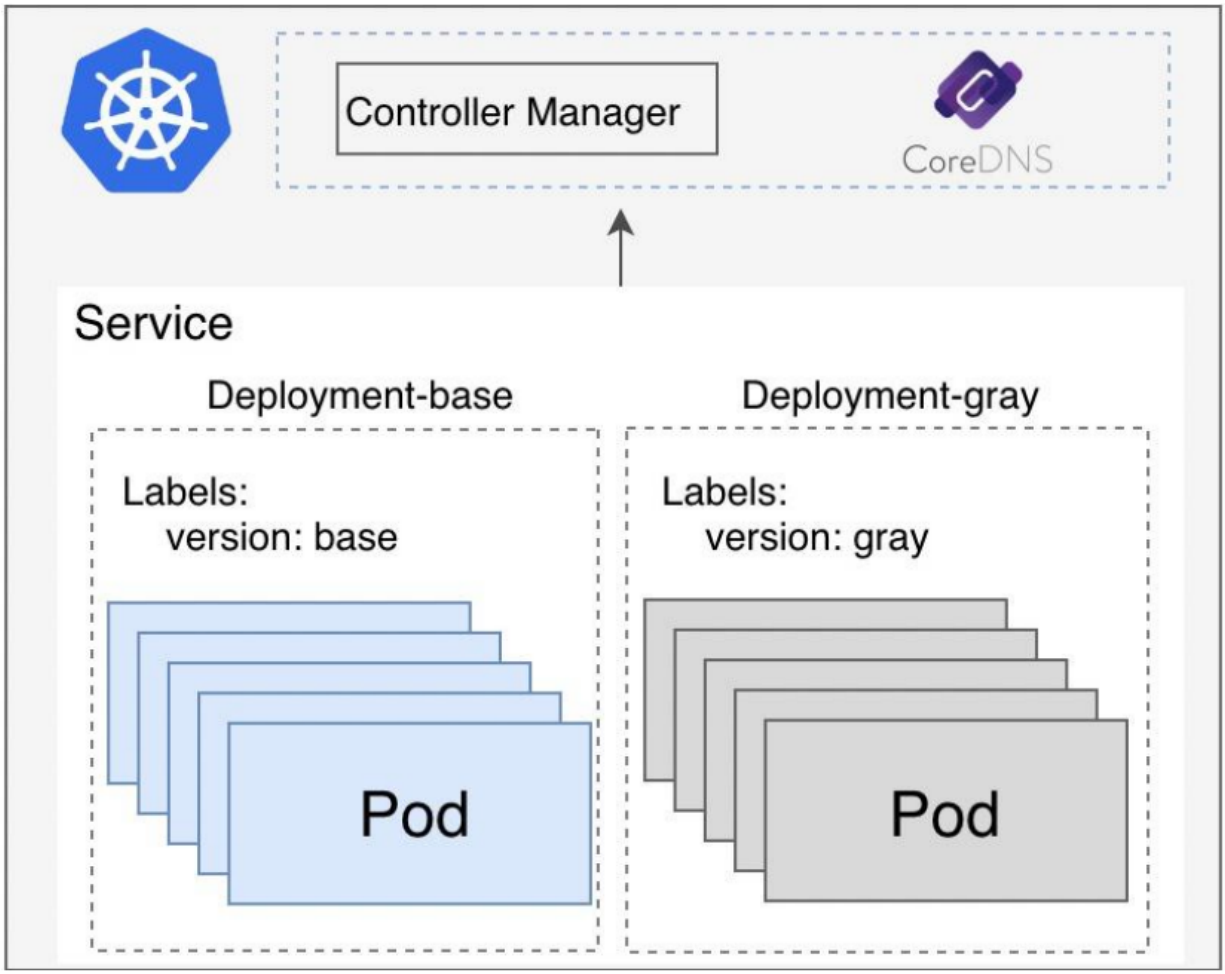
标签路由通过对服务下所有节点按照标签名和标签值不同进行分组，使得订阅该服务节点信息的服务消费端可以按需访问该服务的某个分组，即所有节点的一个子集。服务消费端可以使用服务提供者节点上的任何标签信息，根据所选标签的实际含义，消费端可以将标签路由应用到更多的业务场景中。

节点打标（服务实例染色）

那么如何给服务节点添加不同的标签呢?

基于Kubernetes Service服务发现

在使用Kubernetes Service作为服务发现的业务系统中，服务提供者通过向ApiServer提交Service资源完成服务暴露，服务消费端监听与该Service资源下关联的Endpoint资源，从Endpoint资源中获取关联的业务Pod 资源，读取上面的Labels数据并作为该节点的元数据信息。所以，我们只要在业务应用描述资源Deployment中的Pod模板中为节点添加标签即可。

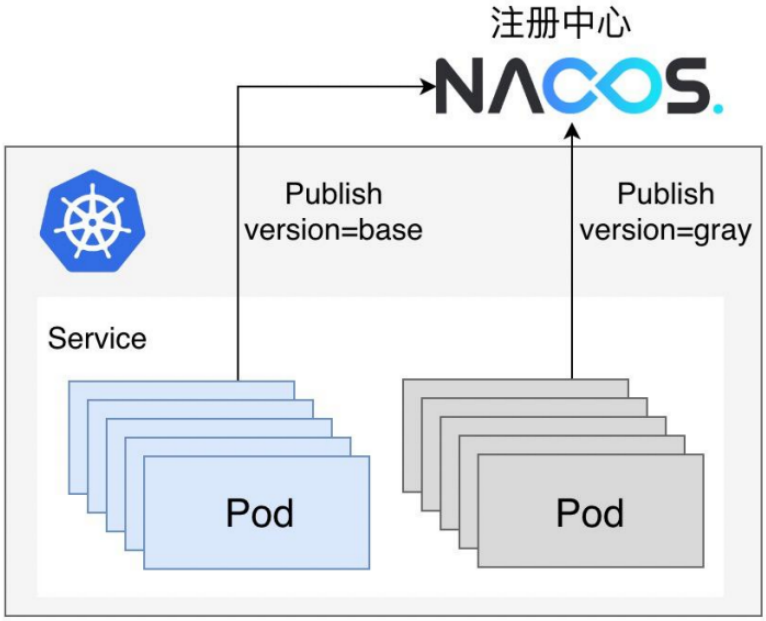


可以在K8S的部署配置中，在template配置中，添加labels的方式完成版本打标。

```
spec:
  template:
    metadata:
      labels:
        app: user
        version: 2.0.0
```

基于Nacos注册中心服务发现

在使用Nacos作为服务发现的业务系统中，一般是需要业务根据其使用的微服务框架来决定打标方式。如果Java应用使用的Spring Cloud微服务开发框架，我们可以为业务容器添加对应的环境变量来完成标签的添加操作。比如我们希望为节点添加版本灰度标，那么为业务容器添加`spring.cloud.nacos.discovery.metadata.version=gray`，这样框架向Nacos注册该节点时会为其添加一个标签`verison=gray`。



流量染色

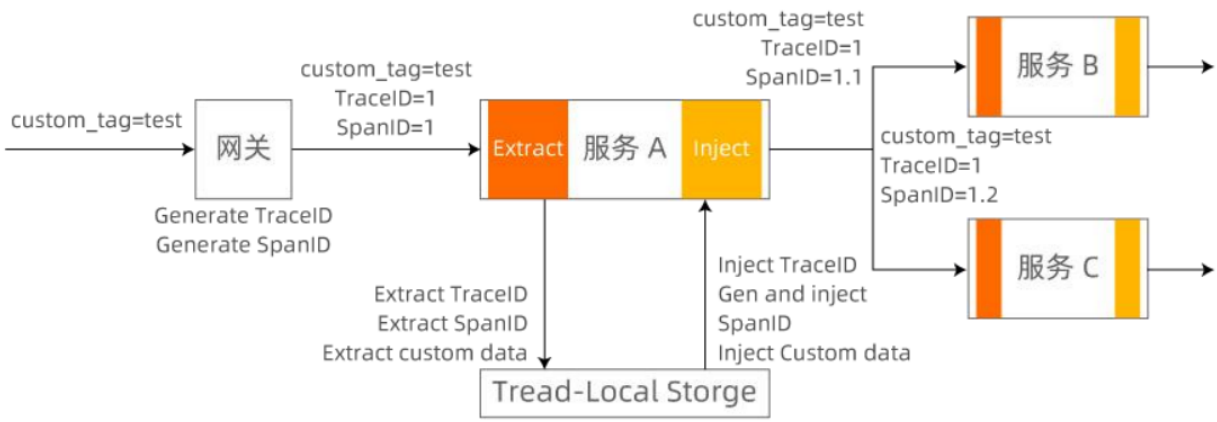
请求链路上各个组件如何识别出不同的灰度流量?

答案就是流量染色，为请求流量添加不同灰度标识来方便区分。我们可以在请求的源头上对流量进行染色，前端在发起请求时根据用户信息或者平台信息的不同对流量进行打标。如果前端无法做到，我们也可以在微服务网关上对匹配特定路由规则的请求动态添加流量标识。此外，流量在链路中流经灰度节点时，如果请求信息中不含有灰度标识，需要自动为其染色，接下来流量就可以在后续的流转过程中优先访问服务的灰度版本。

打标灰度标签透传

如何保证灰度标识能够在链路中一直传递下去呢?

借助于分布式链路追踪思想，我们也可以传递一些自定义信息，比如灰度标识。业界常见的分布式链路追踪产品都支持链路传递用户自定义的数据，其数据处理流程如下图所示:



总结

首先，需要支持动态路由功能，对于Spring Cloud、Dubbo开发框架，可以对出口流量实现自定义Filter，在该Filter中完成流量识别以及标签路由。同时需要借助分布式链路追踪技术完成流量标识链路传递以及流量自动染色。此外，需要引入一个中心化的流量治理平台，方便各个业务线的开发者定义自己的全链路灰度规则。

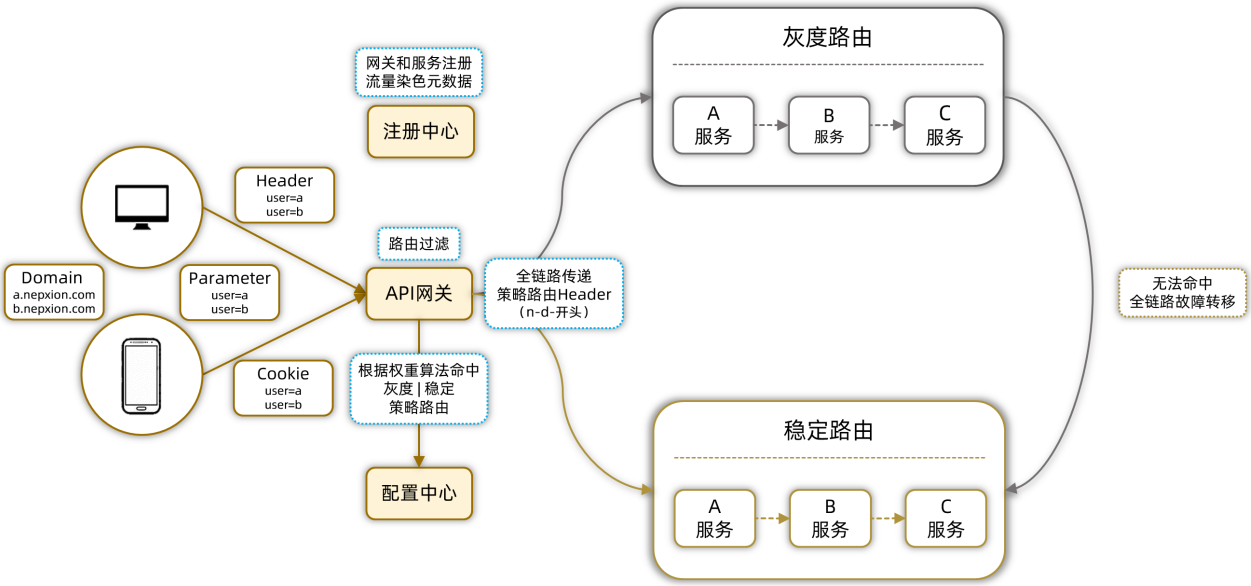
实现全链路灰度的能力，无论是成本还是技术复杂度都是比较高的，以及后期的维护、扩展都是非常大的成本。

3. 大厂微服务治理全链路灰度解决方案

3.1 基于Discovery框架实现全链路灰度

Discovery【探索】微服务框架，基于Spring Cloud & Spring Cloud Alibaba，Discovery服务注册发现、Ribbon & Spring Cloud LoadBalancer负载均衡、Feign & RestTemplate & WebClient调用、Spring Cloud Gateway & Zuul过滤等组件全方位增强的企业级微服务开源解决方案，更贴近企业级需求，更具有企业级的插件引入、开箱即用特征。

> 官方文档：全链路灰度发布



全链路版本条件权重灰度发布

指定百分比流量分配

- 灰度路由，即服务a和b 1.1版本被调用到的概率为5%
- 稳定路由，即服务a和b 1.0版本被调用到的概率为95%

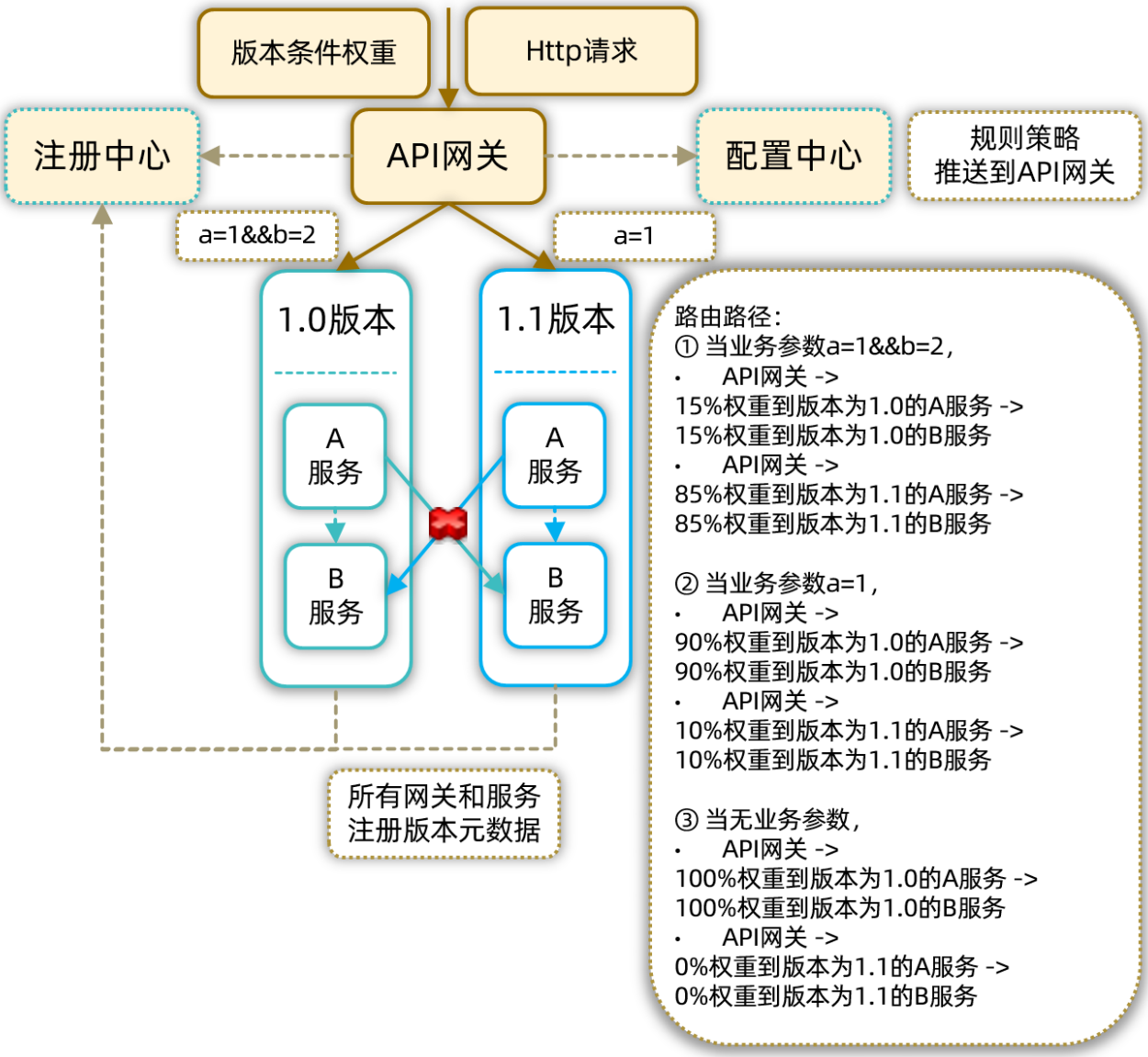
在nacos配置中心修改网关控制的灰度发布策略

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<rule>
    <strategy-release>
        <conditions type="gray">
            <condition id="gray-condition" version-id="gray-route=5;stable-route=95"/>
        </conditions>

        <routes>
            <route id="gray-route" type="version">{"discovery-guide-service-a":"1.1", "discovery-guide-service-b":"1.1"}</route>
            <route id="stable-route" type="version">{"discovery-guide-service-a":"1.0", "discovery-guide-service-b":"1.0"}</route>
        </routes>
    </strategy-release>
</rule>
```

根据前端传递的Header参数动态选择百分比流量分配

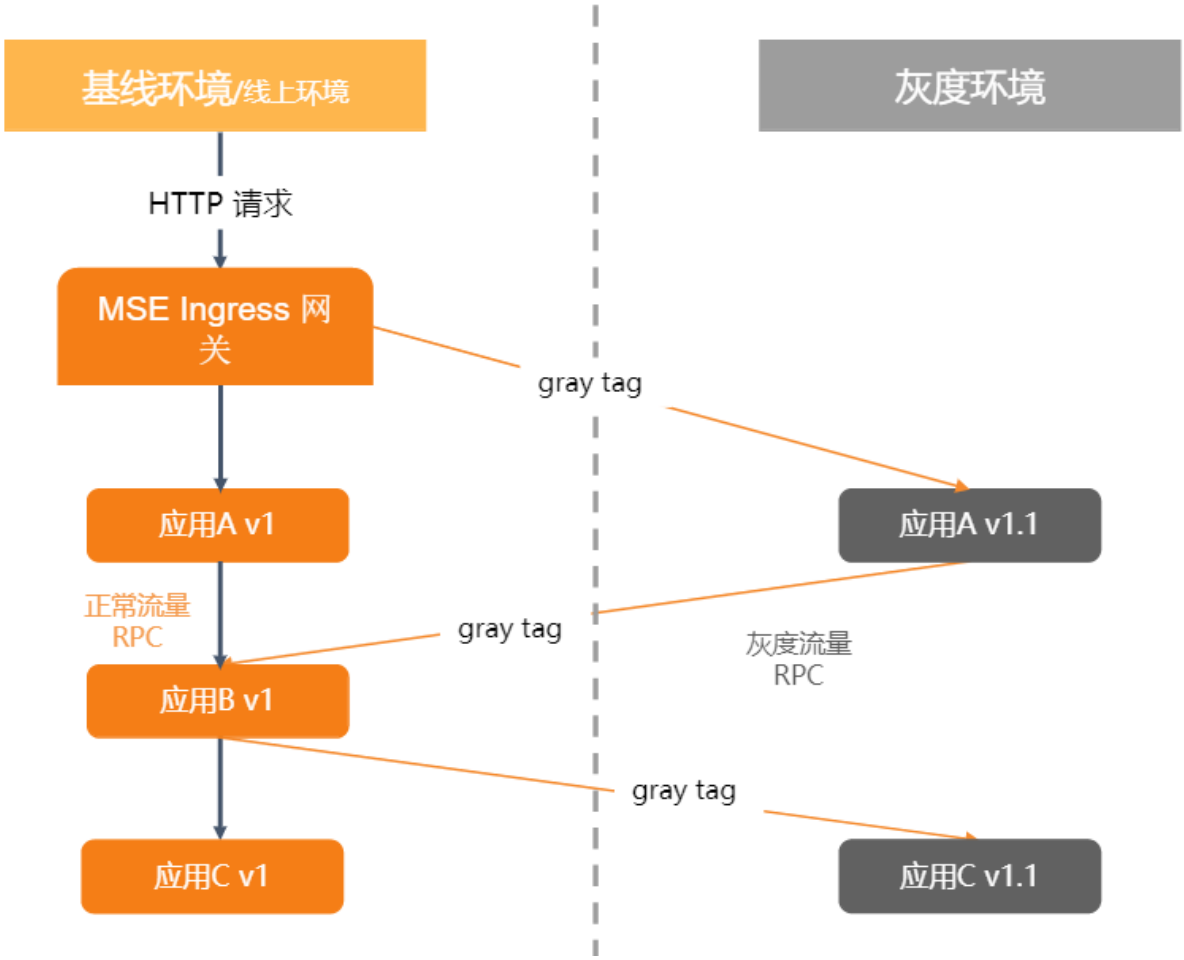
```
<?xml version="1.0" encoding="UTF-8"?>
<rule>
    <strategy-release>
        <conditions type="gray">
            <!-- 灰度路由1，条件expression驱动 -->
            <condition id="gray-condition-1" expression="#H['a'] == '1'" version-id="gray-route=10;stable-route=90"/>
            <!-- 灰度路由2，条件expression驱动 -->
            <condition id="gray-condition-2" expression="#H['a'] == '1' and #H['b'] == '2'" version-id="gray-route=85;stable-route=15"/>
            <!-- 兜底路由，无条件expression驱动 -->
            <condition id="basic-condition" version-id="gray-route=0;stable-route=100"/>
        </conditions>

        <routes>
            <route id="gray-route" type="version">{"discovery-guide-service-a":"1.1", "discovery-guide-service-b":"1.1"}</route>
            <route id="stable-route" type="version">{"discovery-guide-service-a":"1.0", "discovery-guide-service-b":"1.0"}</route>
        </routes>
    </strategy-release>
</rule>
```



3.2 基于阿里云MSE实现全链路灰度

阿里云MSE服务治理产品就是一款基于Java Agent实现的无侵入式企业生产级服务治理产品，您不需要修改任何一行业务代码，即可拥有不限于全链路灰度的治理能力，并且支持近5年内所有的 Spring Boot、Spring Cloud和Dubbo。

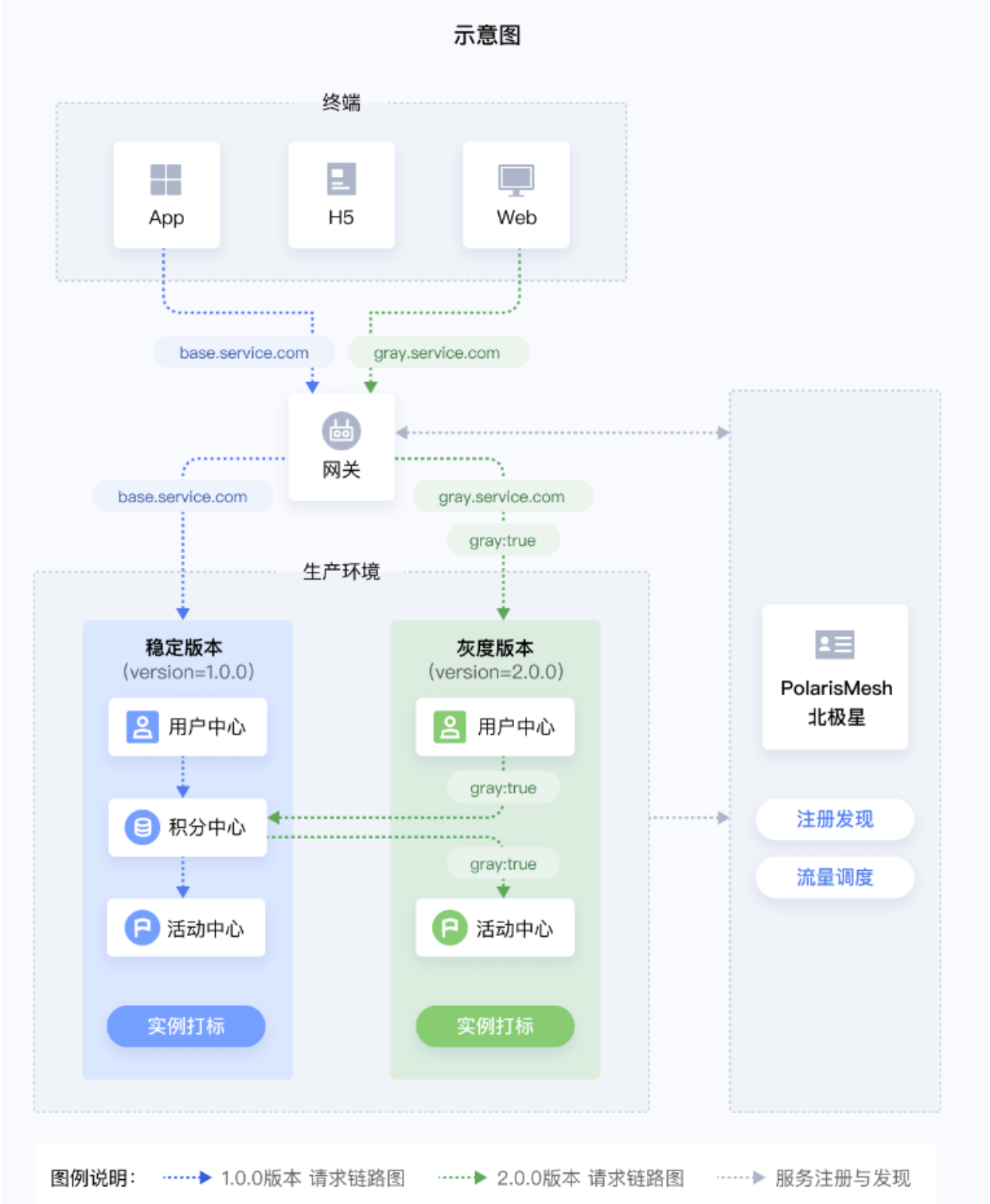
> 官方文档：全链路灰度



3.3 基于腾讯云polarismesh实现全链路灰度

北极星是腾讯开源的服务治理平台，致力于解决分布式和微服务架构中的服务管理、流量管理、配置管理、故障容错和可观测性问题，针对不同的技术栈和环境提供服务治理的标准方案和最佳实践。

可以基于域名分离的方式实现全链路灰度，通过不同的域名区分灰度环境和稳定环境。用户的请求通过灰度域名访问到灰度版本的服务，通过稳定域名访问到稳定版本的服务。



3.4 基于华为云Sermant实现全链路灰度

Sermant（也称之为Java-mesh）是华为云开源的基于Java字节码增强技术的无代理服务网格，其利用Java字节码增强技术为宿主应用程序提供服务治理功能，以解决大规模微服务体系结构中的服务治理问题。

> 官方文档：https://sermant.io/zh/document/plugin/router.html#%E5%8A%9F%E8%83%BD%E4%BB%8B%E7%BB%8D