SpringCloudAlibaba

Nacos

1 Nacos服务发现

依赖

父工程引入SpringCloud和SpringCloudAlibaba依赖

```
<dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Hoxton.SR3</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
<dependency>
                <groupId>com.alibaba.cloud</groupId>
                <artifactId>spring-cloud-alibaba-dependencies</artifactId>
                <version>2.2.1.RELEASE</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
```

子工程引入nacos依赖

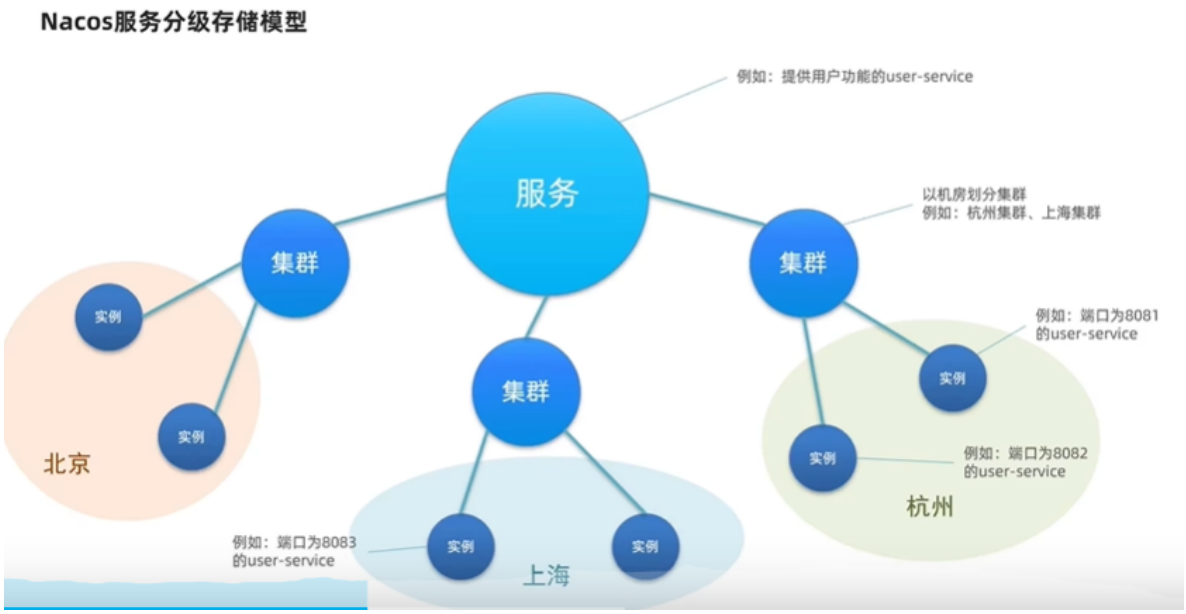
```
<dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
        </dependency>
```

nacos配置文件（默认端口就是8848，不配也可以找到）

```
spring:
  cloud:
    nacos:
      discovery:
        # 指定nacos server地址
        server-addr: localhost:8848
```

2 Nacos服务分级存储模型

服务--集群--实例



服务跨集群调用问题

服务调用尽可能选择本地集群的服务，跨集群调用延迟较高。本地集群不可访问时，再去访问其他集群

nacos集群配置

```
spring:
  cloud:
    nacos:
      discovery:
        # 指定nacos server地址
        server-addr: localhost:8848
        cluster-name: XM #集群名称，自定义，XM代指厦门
```

总结

3 NacosRule负载均衡

在yml文件中设置负载均衡

注意：Ribbon的负载均衡默认采用轮询算法

//此配置表示采用随机算法，my-nacos为你要调用的服务名

```
my-nacos:
  ribbon:
    NFLoadBalancerRuleClassName: com.cyxy.config.NacosWeightedRule
```

//基于权重，需要在服务中手动添加权重配置类

```
my-nacos:
  ribbon:
    NFLoadBalancerRuleClassName: com.cyxy.config.NacosWeightedRule
```

权重配置类

```
import com.alibaba.cloud.nacos.NacosDiscoveryProperties;
import com.alibaba.cloud.nacos.ribbon.NacosServer;
import com.alibaba.nacos.api.exception.NacosException;
import com.alibaba.nacos.api.naming.NamingService;
import com.alibaba.nacos.api.naming.pojo.Instance;
import com.netflix.client.config.IClientConfig;
import com.netflix.loadbalancer.AbstractLoadBalancerRule;
import com.netflix.loadbalancer.BaseLoadBalancer;
import com.netflix.loadbalancer.ILoadBalancer;
import com.netflix.loadbalancer.Server;
import lombok.extern.slf4j.Slf4j;

import javax.annotation.Resource;
```

```
@Slf4j
public class NacosWeightedRule extends AbstractLoadBalancerRule {

    //Ribbon基于权重的算法

    @Resource
    private NacosDiscoveryProperties nacosDiscoveryProperties;

    @Override
    public void initWithNiwsConfig(IClientConfig iClientConfig) {
        //读取配置文件
    }

    @Override
    public Server choose(Object o) {
        ILoadBalancer loadBalancer = this.getLoadBalancer();
        BaseLoadBalancer baseLoadBalancer = (BaseLoadBalancer) loadBalancer;
        //获取要请求的微服务名称
        String name = baseLoadBalancer.getName();
        //获取服务发现的相关API
        NamingService namingService =
nacosDiscoveryProperties.namingServiceInstance();
        try {
            Instance instance = namingService.selectOneHealthyInstance(name);
            log.info("选择的实例是port={},instance=
{}",instance.getPort(),instance);
            return new NacosServer(instance);
        } catch (NacosException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

//NacosRule-----优先选择本地集群，在本地集群的多个服务中采用随机的负载均衡算法

```
my-nacos:
  ribbon:
    NFLoadBalancerRuleClassName: com.alibaba.cloud.nacos.ribbon.NacosRule
```

注意：本地集群如果没有服务则会跨集群进行访问，同时会有警告信息出现

总结

1. NacosRule负载均衡策略

① 优先选择同集群服务实例列表
② 本地集群找不到提供者，才去其它集群寻找，并且会报警告
③ 确定了可用实例列表后，再采用随机负载均衡挑选实例

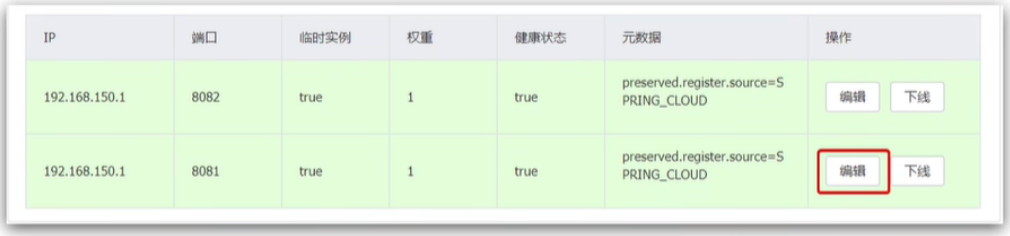
4 Nacos服务实例的权重设置

nacos提供了权重配置来控制访问频率，权重越大则访问频率越高,权重设置为0则不会被访问

设置权重方式

根据权重负载均衡

1. 在Nacos控制台可以设置实例的权重值，首先选中实例后面的编辑按钮



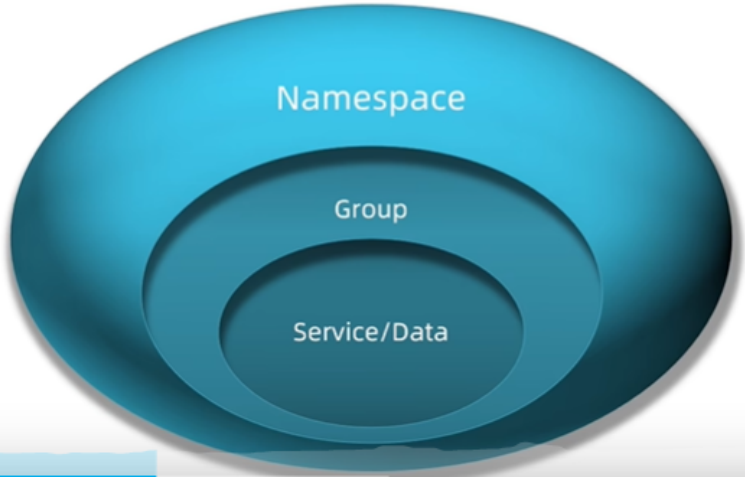
IP	端口	临时实例	权重	健康状态	元数据	操作
192.168.150.1	8082	true	1	true	preserved.register.source=SPRING_CLOUD	编辑 下线
192.168.150.1	8081	true	1	true	preserved.register.source=SPRING_CLOUD	编辑 下线

2. 将权重设置为0.1，测试可以发现8081被访问到的频率大大降低

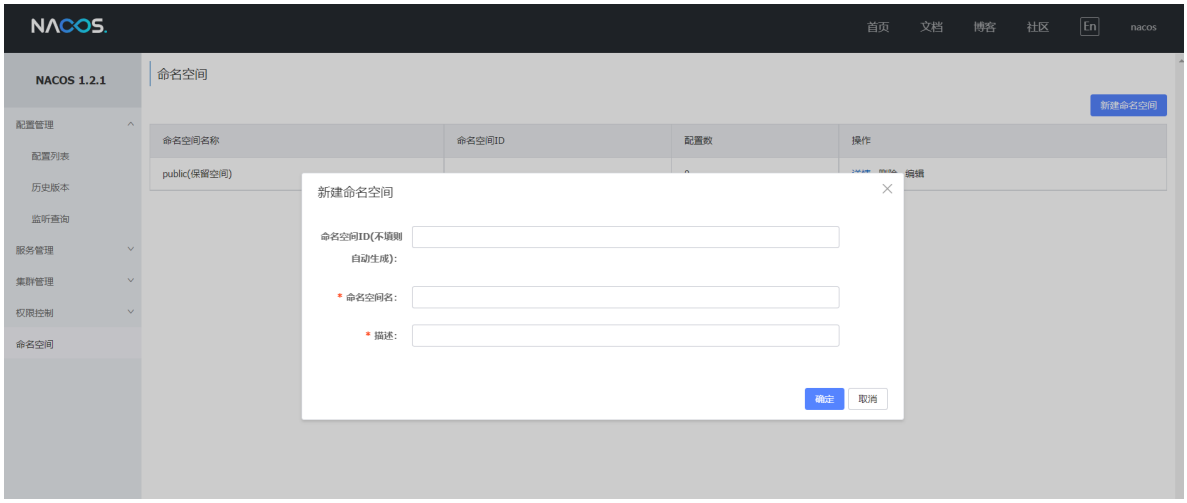
5 Nacos环境隔离

环境隔离 - namespace

Nacos中服务存储和数据存储的最外层都是一个名为namespace的东西，用来做最外层隔离



在nacos控制台进行配置



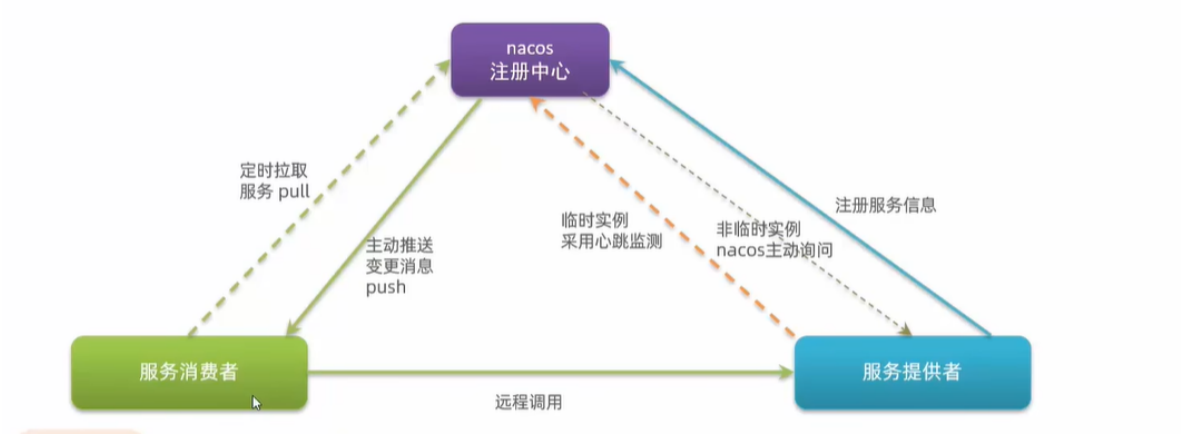
在yml中的namespace配置命名空间的id

```
spring:
  cloud:
    nacos:
      discovery:
        # 指定nacos server地址
        server-addr: localhost:8848
        cluster-name: XM #集群名称，自定义，XM代指厦门
        namespace: 492a71383-dhjds7864e-adw2 #配置命名空间的id，此id为新建命名空间时nacos控制台自动生成的
```

1. Nacos环境隔离
   ① namespace用来做环境隔离
   ② 每个namespace都有唯一id
   ③ 不同namespace下的服务不可见

注意：不同命名空间下的服务是不可以调用的

6 Nacos的临时实例与非临时实例



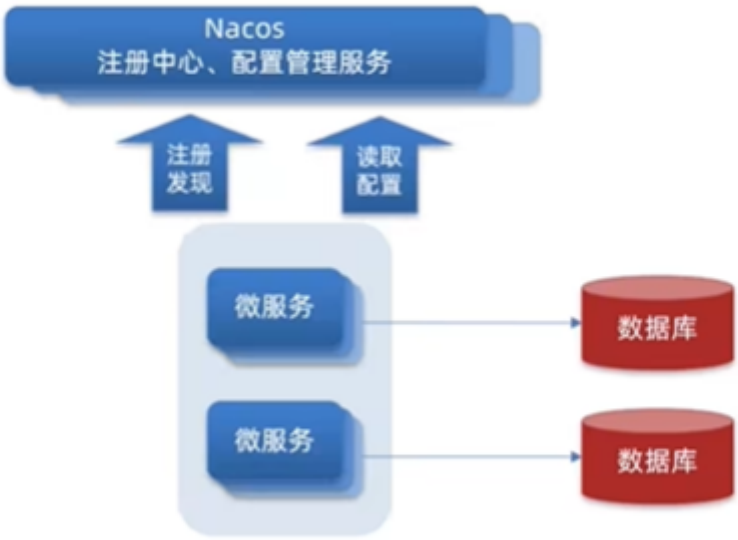

```
spring:
  cloud:
    nacos:
      discovery:
        ephemeral: false #是否是临时实例
```

临时实例down掉后，会被nacos删除，非临时实例则会等待它恢复

7 Nacos与Eureka的区别

1. Nacos与eureka的共同点
   - ① 都支持服务注册和服务拉取
   - ② 都支持服务提供者心跳方式做健康检测
2. Nacos与Eureka的区别
   - ① Nacos支持服务端主动检测提供者状态：临时实例采用心跳模式，非临时实例采用主动检测模式
   - ② 临时实例心跳不正常会被剔除，非临时实例则不会被剔除
   - ③ Nacos支持服务列表变更的消息推送模式，服务列表更新更及时
   - ④ Nacos集群默认采用AP方式，当集群中存在非临时实例时，采用CP模式；Eureka采用AP方式

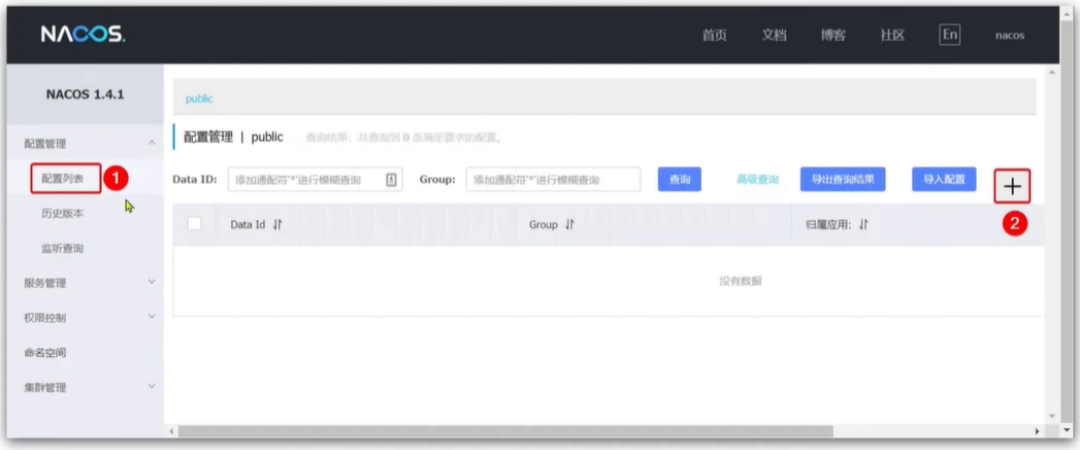
8 Nacos实现配置管理



在nacos终端中添加配置文件

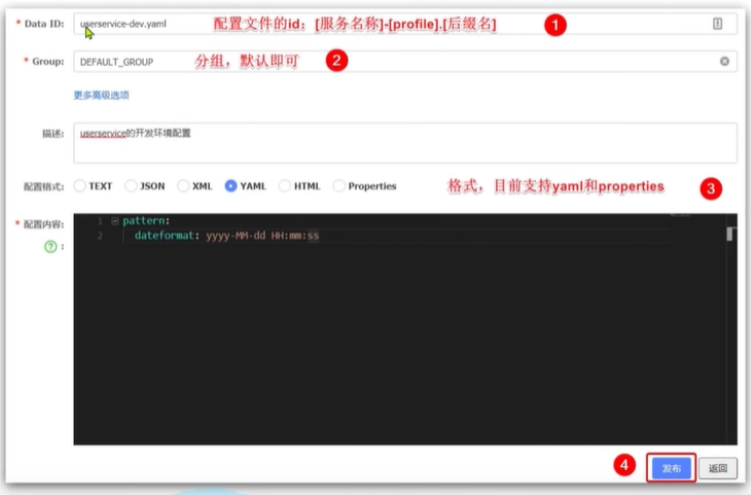
统一配置管理

在Nacos中添加配置信息：



统一配置管理

在弹出表单中填写配置信息：

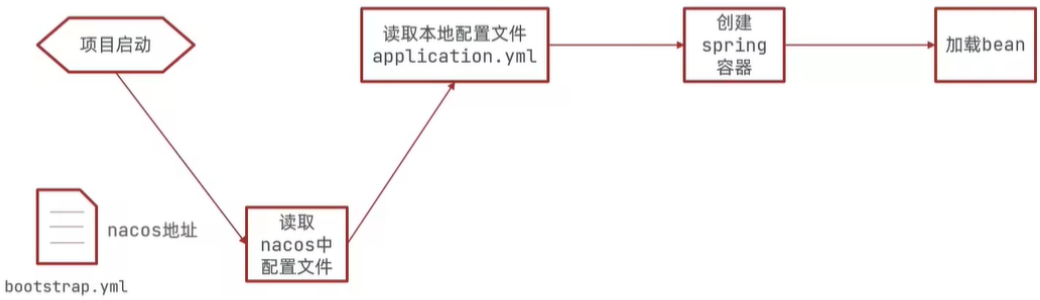


注意配置文件中的内容应该是经常有需求变化的内容

SpringBoot中读取统一配置文件

统一配置管理

配置获取的步骤如下：



引入客户端依赖

1. 引入Nacos的配置管理客户端依赖：

```
<!--nacos配置管理依赖-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
```

配置bootstrap.yaml

2. 在userservice中的resource目录添加一个bootstrap.yml文件，这个文件是引导文件，优先级高于application.yml：

```
spring:
  application:
    name: userservice # 服务名称
  profiles:
    active: dev #开发环境，这里是dev
  cloud:
    nacos:
      server-addr: localhost:8848 # Nacos地址
      config:
        file-extension: yaml # 文件后缀名
```

```
spring:
  application:
    name: my-nacos #服务名称
  profiles:
    active: dev #开发环境，这里是dev
  cloud:
    nacos:
      server-addr: localhost:8848 #nacos地址
      config:
        file-extension: yaml #文件后缀名
```

此时的application.yml文件中就可以将nacos地址和服务名称删除

```
spring:
  cloud:
    nacos:
      discovery:
        # 指定nacos server地址
#         server-addr: localhost:8848
        cluster-name: XM #集群名称，自定义，XM代指厦门
        namespace: 492a71383-dhjds7864e-adw2 #配置命名空间的id，此id为新建命名空间时
nacos控制台自动生成的
        ephemeral: false #是否是临时实例
#  application:
#    name: my-nacos
server:
  port: 8081
```

拉取配置文件测试

统一配置管理

我们在user-service中将pattern.dateformat这个属性注入到UserController中做测试：

```
@RestController
@RequestMapping("/user")
public class UserController {

    // 注入nacos中的配置属性
    @Value("${pattern.dateformat}")
    private String dateformat;

    // 编写controller，通过日期格式化器来格式化现在时间并返回
    @GetMapping("now")
    public String now(){
        return LocalDate.now().format(
                DateTimeFormatter.ofPattern(dateformat, Locale.CHINA)
        );
    }
    // ... 略
}
```

9 Nacos配置热更新(配置自动更新)

两种方式实现

方式1

Nacos中的配置文件变更后，微服务无需重启就可以感知。不过需要通过下面两种配置实现：

- 方式一：在@Value注入的变量所在类上添加注解@RefreshScope

```java
@Slf4j
@RestController
@RequestMapping("/user")
@RefreshScope
public class UserController {

    @Value("${pattern.dateformat}")
    private String dateformat;
```

方式2

Nacos中的配置文件变更后，微服务无需重启就可以感知。不过需要通过下面两种配置实现：

- 方式二：使用@ConfigurationProperties注解

```java
@Component
@Data
@ConfigurationProperties(prefix = "pattern")
public class PatternProperties {
    private String dateformat;
}
```

Nacos配置更改后，微服务可以实现热更新，方式：

① 通过@Value注解注入，结合@RefreshScope来刷新

② 通过@ConfigurationProperties注入，自动刷新

注意事项：

- 不是所有的配置都适合放到配置中心，维护起来比较麻烦
- 建议将一些关键参数，需要运行时调整的参数放到nacos配置中心，一般都是自定义配置

10 Nacos 多环境配置共享

多环境配置共享

微服务启动时会从nacos读取多个配置文件：

- [spring.application.name]-[spring.profiles.active].yaml，例如：userservice-dev.yaml
- [spring.application.name].yaml，例如：userservice.yaml

无论profile如何变化，[spring.application.name].yaml这个文件一定会加载，因此多环境共享配置可以写入这个文件

如果本地配置文件与nacos客户端的两个配置文件有相同配置时，优先级为

多种配置的优先级：

- 服务名-profile.yaml >服务名称.yaml > 本地配置

微服务会从nacos读取的配置文件：

① [服务名]-[spring.profile.active].yaml，环境配置

② [服务名].yaml，默认配置，多环境共享

优先级：

① [服务名]-[环境].yaml >[服务名].yaml > 本地配置

11 Nacos集群配置

看Nacos集群搭建配置文档配合黑马视频使用。

黑马视频地址

```
https://www.bilibili.com/video/BV1LQ4y127n4?
p=29&vd_source=21e0104c53b895468585eecd2498df15
```

Gateway

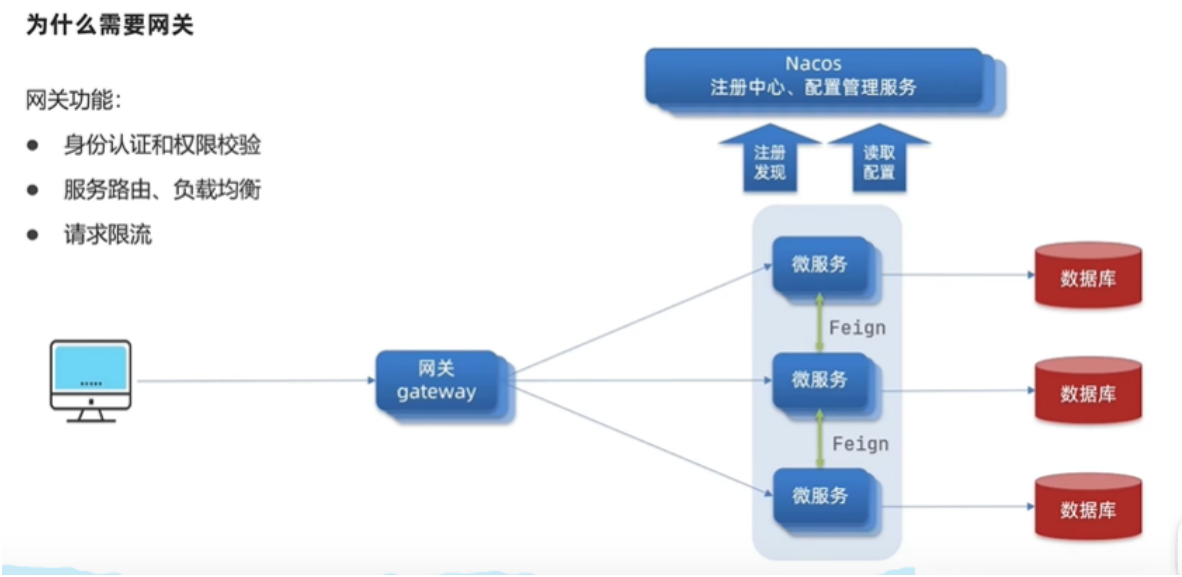
1 为什么需要网关

网关功能：

身份认证和权限校验

服务路由和负载均衡

请求限流



SpringCloud实现网关的两种方式:

Gateway

Zuul

为什么选择gateway

Zuul是基于Servlet的实现，属于阻塞式编程。而SpringCloudGateway则是基于Spring5中提供的WebFlux，属于响应式编程的实现，具备更好的性能。

网关的作用:

- 对用户请求做身份认证、权限校验
- 将用户请求路由到微服务，并实现负载均衡
- 对用户请求做限流

2 搭建网关服务

gateway依赖和nacos依赖

```
  <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-gateway</artifactId>
        </dependency>

        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
        </dependency>
```

路由的yml

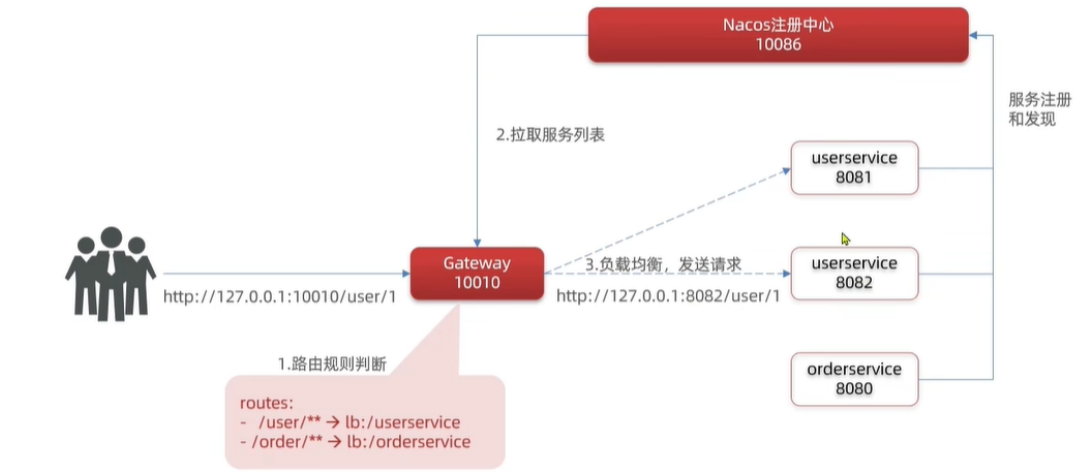
```
server:
  port: 9999
spring:
  application:
    name: gateway-service
  cloud:
    nacos:
      server-addr: localhost:8848
    gateway:
      routes:
        - id: user-service # 路由标识，必须唯一，自定义
          uri: lb://userservice # lb表示负载均衡，userservice为服务名称，路由到目标地址
          predicates: #路由断言，判断请求是否符合要求
            -Path=/user/** #路径断言，判断路径是否以user开头
        - id: order-service
          uri: lb://orderservice
          predicates:
            -Path=/order/**
```

2. 编写路由配置及nacos地址

```
server:
  port: 10010 # 网关端口
spring:
  application:
    name: gateway # 服务名称
  cloud:
    nacos:
      server-addr: localhost:8848 # nacos地址
    gateway:
      routes: # 网关路由配置
        - id: user-service # 路由id，自定义，只要唯一即可
          # uri: http://127.0.0.1:8081 # 路由的目标地址 http就是固定地址
          uri: lb://userservice # 路由的目标地址 lb就是负载均衡，后面跟服务名称
          predicates: # 路由断言，也就是判断请求是否符合路由规则的条件
            - Path=/user/** # 这个是按照路径匹配，只要以/user/开头就符合要求
```

网关映射过程

搭建网关服务



3 路由断言工厂

```
predicates: #路由断言，判断请求是否符合要求
        -Path=/user/** #路径断言，判断路径是否以user开头
```

配置文件中断言只是简单的字符串，需要到断言工厂中进一步转换

Spring提供了11种基本的Predicate工厂：

名称	说明	示例
After	是某个时间点后的请求	- After=2037-01-20T17:42:47.789-07:00[America/Denver]
Before	是某个时间点之前的请求	- Before=2031-04-13T15:14:47.433+08:00[Asia/Shanghai]
Between	是某两个时间点之前的请求	- Between=2037-01-20T17:42:47.789-07:00[America/Denver], 2037-01-21T17:42:47.789-07:00[America/Denver]
Cookie	请求必须包含某些cookie	- Cookie=chocolate, ch.p
Header	请求必须包含某些header	- Header=X-Request-Id, \d+
Host	请求必须是访问某个host（域名）	- Host=**.somehost.org,**.anotherhost.org
Method	请求方式必须是指定方式	- Method=GET,POST
Path	请求路径必须符合指定规则	- Path=/red/{segment},/blue/**
Query	请求参数必须包含指定参数	- Query=name, Jack或者- Query=name
RemoteAddr	请求者的ip必须是指定范围	- RemoteAddr=192.168.1.1/24
Weight	权重处理	

注意：断言条件可以有多个，并且条件间是逻辑且的关系

PredicateFactory的作用是什么？

读取用户定义的断言条件，对请求做出判断。

4 路由的过滤器配置（网关过滤器）

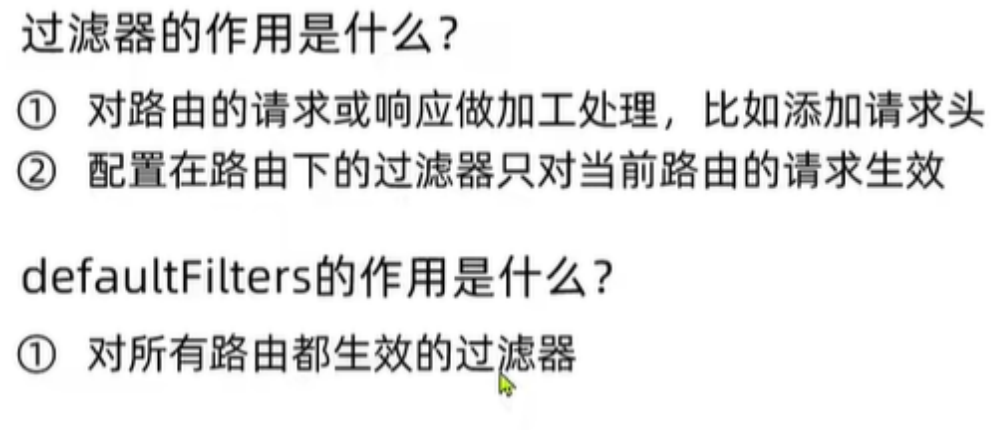
GatewayFilter是网关提供的一种过滤器，可以对进入网关的请求和微服务返回的响应做处理

在yml中给userservice的路由添加过滤器

```
server:
  port: 9999
spring:
  application:
    name: gateway-service
  cloud:
    nacos:
      server-addr: localhost:8848
    gateway:
      routes:
        - id: user-service # 路由标识，必须唯一，自定义
          uri: lb://userservice # lb表示负载均衡，userservice为服务名称，路由到目标地址
          predicates: #路由断言，判断请求是否符合要求
            -Path=/user/** #路径断言，判断路径是否以user开头
          filters:
            - AddRequestHeader=Truth,SpringNB #过滤器，给userservice添加一个请求头
Truth为key
```

配置默认过滤器

```
server:
  port: 9999
spring:
  application:
    name: gateway-service
  cloud:
    nacos:
      server-addr: localhost:8848
    gateway:
      routes:
        - id: user-service # 路由标识，必须唯一，自定义
          uri: lb://userservice # lb表示负载均衡，userservice为服务名称，路由到目标地址
          predicates: #路由断言，判断请求是否符合要求
            -Path=/user/** #路径断言，判断路径是否以user开头
#          filters:
#            - AddRequestHeader=Truth,SpringNB #过滤器，给userservice添加一个请求头
        - id: order-service
          uri: lb://orderservice
          predicates:
            -Path=/order/**
      default-filters: #默认过滤器，所有路由都会执行
        - AddRequestHeader=Truth,SpringNB
```



5 全局过滤器（GlobalFilter）

全局过滤器的作用也是处理一切进入网关的请求和微服务响应，与GatewayFilter的作用一样

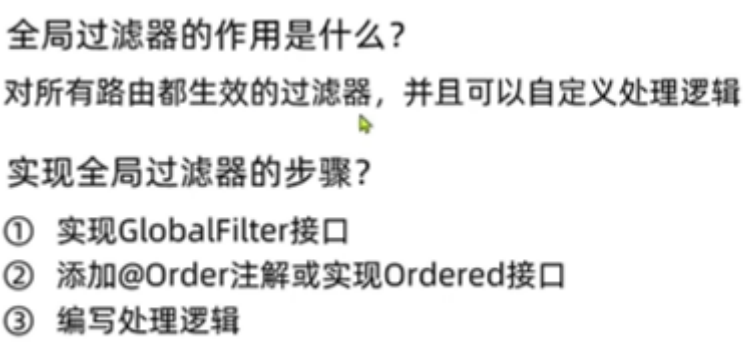
区别在于Gateway是通过配置定义，处理逻辑是固定的，而GlobalFilter的逻辑需要自己手写代码去实现。

定义方式实现GlobalFilter接口

```
public interface GlobalFilter {
    /**
     *  处理当前请求，有必要的话通过{@link GatewayFilterChain}将请求交给下一个过滤器处理
     *
     * @param exchange 请求上下文，里面可以获取Request、Response等信息
     * @param chain 用来把请求委托给下一个过滤器
     * @return {@code Mono<Void>} 返回标示当前过滤器业务结束
     */
    Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain);
}
```

```
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.core.annotation.Order;
import org.springframework.http.HttpStatus;
import org.springframework.http.server.reactive.ServerHttpRequest;
import org.springframework.stereotype.Component;
import org.springframework.util.MultiValueMap;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Mono;

@Order(-1)//设置过滤器执行顺序
@Component
public class AuthorizedFilter implements GlobalFilter {

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain
chain) {
        //1 获取请求参数
```

```
        ServerHttpRequest request = exchange.getRequest();
        MultiValueMap<String, String> params = request.getQueryParams();


        //2 获取参数中的authorization参数
        String auth = params.getFirst("authorization");

        //3 判断参数是否等于admin
        if (auth.equals("admin")){
            //4 是，放行
            return chain.filter(exchange);
        }




        //5 否，拦截

        //5.1 设置拦截码
        exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);

        //5.2 拦截请求
        return exchange.getResponse().setComplete();
    }
}
```

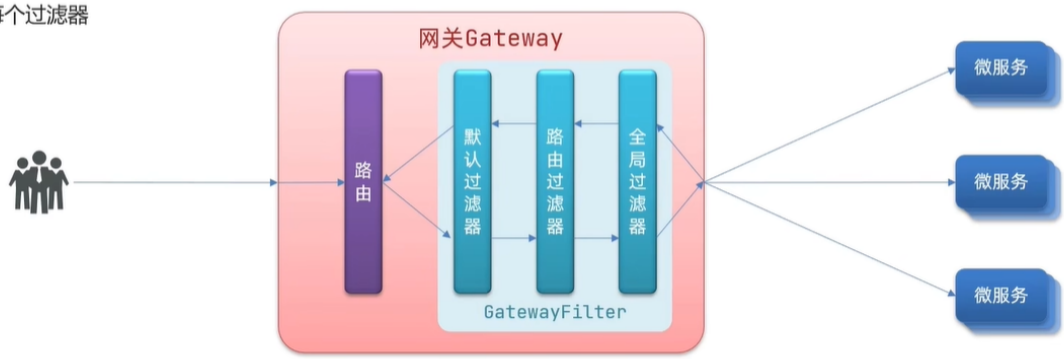


6 过滤器执行顺序

过滤器执行顺序

请求进入网关会碰到三类过滤器：当前路由的过滤器、DefaultFilter、GlobalFilter

请求路由后，会将当前路由过滤器和DefaultFilter、GlobalFilter，合并到一个过滤器链（集合）中，排序后依次执行每个过滤器



过滤器执行顺序

- 每一个过滤器都必须指定一个int类型的order值，**order值越小，优先级越高，执行顺序越靠前。**
- GlobalFilter通过实现Ordered接口，或者添加@Order注解来指定order值，由我们自己指定
- 路由过滤器和defaultFilter的order由Spring指定，默认是按照声明顺序从1递增。
- 当过滤器的order值一样时，会按照 defaultFilter > 路由过滤器 > GlobalFilter的顺序执行。

7 网关的跨域配置

跨域

跨域：域名不一致就是跨域，主要包括：

- 域名不同： www.taobao.com 和 www.taobao.org 和 www.jd.com 和 miaosha.jd.com
- 域名相同，端口不同：localhost:8080和localhost8081

跨域问题：浏览器禁止请求的发起者与服务端发送跨域ajax请求，请求被浏览器拦截的问题

cors跨域

```
server:
  port: 8686
spring:
  application:
    name: gateway
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]':
            allowed-origins:
              - "http://localhost:8383"
```

```
            - "http://localhost:8282"
          allowed-headers: "*"
          allowed-methods: "*"
          max-age: 3600
      discovery:
        locator:
          enabled: true
```

```
spring:
  cloud:
    gateway:
      globalcors:
        add-to-simple-url-handler-mapping: true
        cors-configurations:
          '[/**]':
            allowedOrigins: #允许哪些网站的跨域请求
              - "http://localhost:8383"
              - "http://localhost:8282"
            allowedMethods: #允许的跨域ajax的请求方式
              - "GET"
              - "POST"
              - "PUT"
              - "DELETE"
              - "OPTIONS"
            allowedHeaders: "*" #允许在请求中携带的头信息
            allowCredentials: true #是否允许携带cookie
            maxAge: 360000 #这次跨域检测的有效期
```