## 一站式微服务解决方案Spring Cloud Alibaba

主讲：Cat 老师

北京动力节点教育科技有限公司  
2020 • 北京

动力节点•版权所有•禁止传播

## 快速回顾

1、Spring家族开源项目梳理

2、Spring Cloud下的开源项目梳理

3、Spring Cloud Alibaba下的开源及商业项目梳理

4、微服务的基础模型：服务消费者-注册中心-服务提供者

5、What is Nacos？

6、Nacos的运行环境部署（Java写的，springboot项目）

7、Nacos的后台web管控台

8、Nacos作为注册中心注册服务

9、Nacos作为注册中心发现/订阅服务

10、服务消费者负载均衡调用服务提供者（ribbon）restTemple、feign

11、Nacos宕机时服务消费者缓存注册中心信息

12、Nacos作为配置中心存储项目各种配置

13、Nacos作为配置中心支持自动配置刷新（不需要重启应用）

14、Nacos配置中心DataId+Group+Properties/yaml+配置内容（比较灵活）

15、Nacos配置中心多环境配置（profile）

即${spring.application.name}-${profile}.${file-extension:properties}

16、Nacos服务配置数据模型（命名空间、Group、Data Id）

17、Nacos数据持久化（mysql）

18、Ncaos集群部署（nginx）

19、主要调用方式：restTemplate、feign、ribbon（spring cloud）

20、流量控制Sentinel（流控、降级、热点、系统、授权 规则）

21、Sentinel Darshboard通信原理

22、Sentinel 对应用保护的三种方式

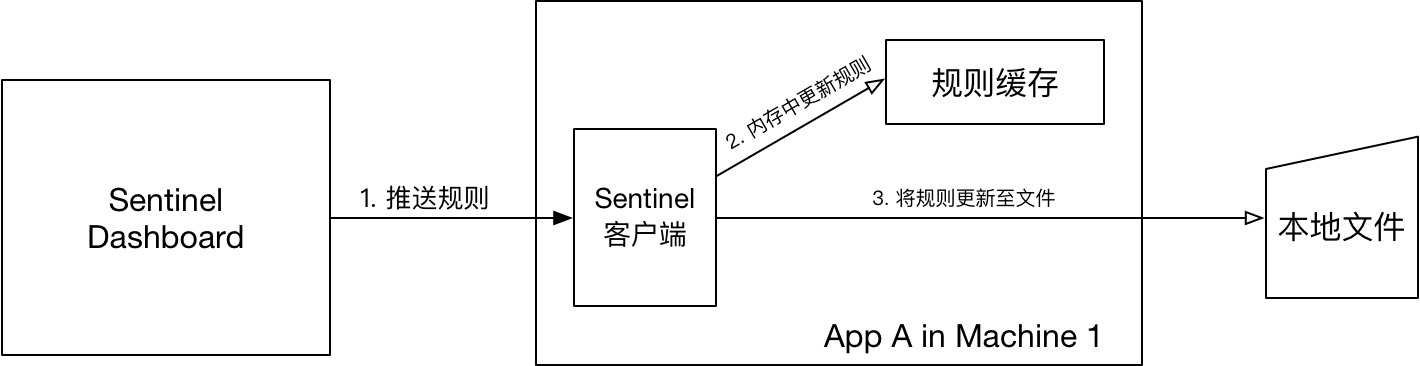
23、Sentinel整合RestTemplate流控熔断

24、Sentinel整合Feign流控熔断

## Sentinel规则持久化

（1）原始模式：这是默认模式，该模式下规则不持久化，重启微服务，配置的限流降级等规则都丢失；

（2）Pull模式：（拉模式）



如上图所示：sentinel dashboard推送规则给微服务，微服务将规则更新到内存，同时将规则更新到本地文件，以实现规则的持久化；

1. Push模式（推模式）：这种方式是将规则存储在nacos配置中心，微服务从nacos配置中心获取规则，这种方式有更好的实时性和一致性保证，**生产环境下一般采用该方式（支持nacos、zookeeper、Apollo等）；**

**但是目前有一个小问题，当我们在**sentinel dashboard控制台更新规则，nacos里面的规则并不能得到更新，后续的版本中可能会解决该问题；

下面我们分别看一下Pull模式：（拉模式）和Push模式（推模式）；

### Pull模式：（拉模式）规则持久化到文件

1、配置依赖；

*<!--sentinel-datasource-extension数据源扩展-->*<dependency>  
 <groupId>com.alibaba.csp</groupId>  
 <artifactId>sentinel-datasource-extension</artifactId>  
</dependency>

2、配置文件；

暂时不需要配置；

3、写代码；

见项目中的 FileDataSourceInit 类；

1. 配置SPI；

resources\META-INF\services\com.alibaba.csp.sentinel.init.InitFunc

### Push模式（推模式）规则持久化到Nacos

1、添加sentinel-datasource-nacos依赖；

*<!--sentinel数据持久化-->*<dependency>  
 <groupId>com.alibaba.csp</groupId>  
 <artifactId>sentinel-datasource-nacos</artifactId>  
</dependency>

2、application.properties配置持久化数据源；

spring.cloud.sentinel.datasource.ds1.nacos.server-addr=192.168.172.128:80  
spring.cloud.sentinel.datasource.ds1.nacos.data-id=${spring.application.name}.json  
spring.cloud.sentinel.datasource.ds1.nacos.group-id=DEFAULT\_GROUP  
spring.cloud.sentinel.datasource.ds1.nacos.data-type=json  
spring.cloud.sentinel.datasource.ds1.nacos.rule-type=*flow*

3、在nacos配置中心配置流控规则；

[

{

"resource": "abc",

"controlBehavior": 0,

"count": 1.0,

"grade": 1,

"limitApp": "default",

"strategy": 0

}  
]

## Spring Cloud Gateway

Spring Cloud Gateway 网关是SpringCloud官方提供的；

原来有一个Zuul网关，是Netflix公司提供的，现在已经不维护了，后面Netflix公司又出来了一个Zuul2.0网关，但由于一直没有发布稳定版本，所以springcloud等不及了就自己推出一个网关，已经不打算整合zuul2.0了；

Spring Cloud Gateway 项目提供了一个用于在Spring MVC之上构建API网关的库，Spring Cloud Gateway旨在提供一种简单而高效的方法来将请求路由到API，并为它们提供跨领域的关注，例如：安全性，监视/度量和弹性等；

当前Spring Cloud Gateway 最新版本：2.2.4；

## Spring Cloud Gateway特征功能

1、建立在Spring Framework 5，Project Reactor和Spring Boot 2.0之上；

2、能够匹配任何请求属性上的路由；

3、谓词和过滤器特定于路由；

4、Hystrix断路器集成；

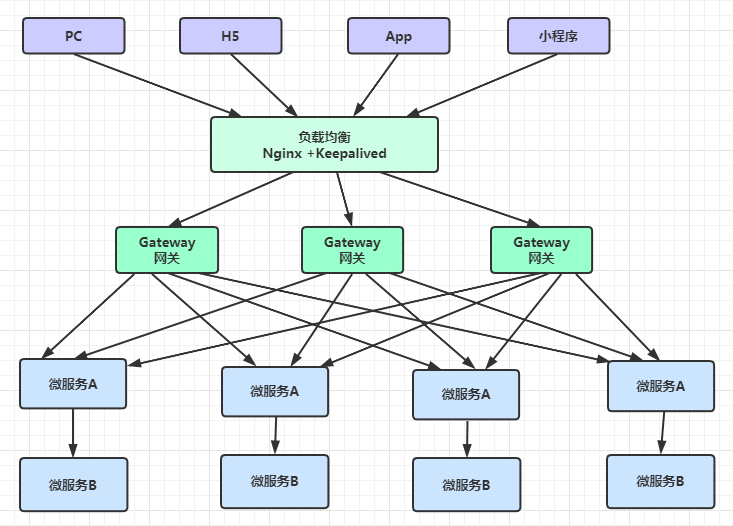
5、Spring Cloud DiscoveryClient的集成；

6、易于编写的谓词和过滤器；

7、请求速率限制；

8、路径改写；

注：spring cloud gateway至少要求JDK8；



## 如何使用Spring Cloud Gateway

要将Spring Cloud Gateway包含在您的项目中，添加

spring-cloud-starter-gateway的依赖即可，如果包括启动器，但不希望启用网关，可以设置：

spring.cloud.gateway.enabled = false

注：不要添加spring-boot-start-web模块的jar依赖，否则会报错；

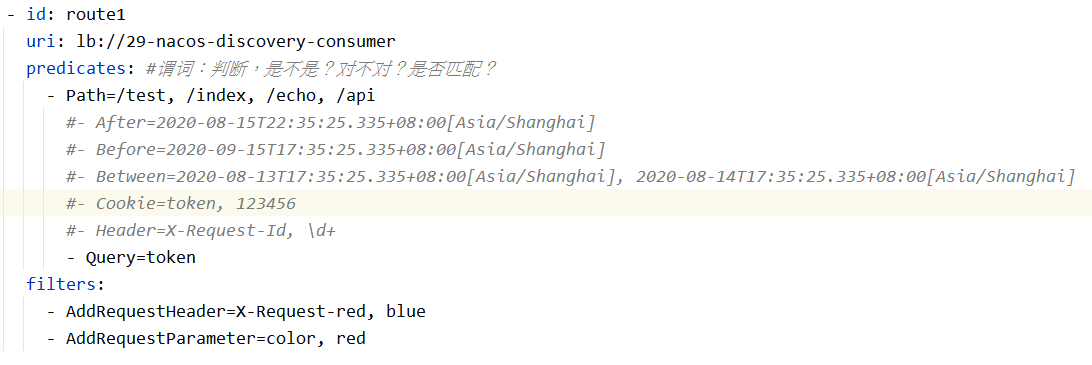
具体步骤：

1、添加gateway网关依赖；

*<!--spring-cloud-starter-gateway 网关-->*<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-gateway</artifactId>  
</dependency>

2、配置application.yml文件；





3、启动运行程序，通过网关转发到具体的服务；

## Spring Cloud gateway核心概念

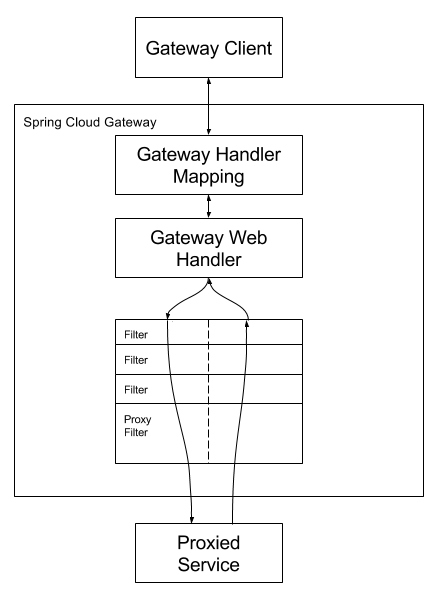
路由：网关的基本构建组成，它由ID，目标URI，谓词集合和过滤器集合定义，如果集合谓词为true，则匹配路由，否则不匹配；

谓词：这是Java 8函数谓词，输入类型是Spring Framework ServerWebExchange，可以匹配HTTP请求中的所有内容，例如请求头或参数；

过滤器：这些是使用特定工厂构造的Spring Framework GatewayFilter实例，可以在发送给下游请求之前或之后修改请求和响应；

## Gateway如何工作？

下图从总体上概述了Spring Cloud Gateway的工作方式：

****

客户端向Spring Cloud Gateway发出请求，如果网关处理程序映射确定请求与路由匹配，则将其发送到网关Web处理程序，该处理程序通过特定于请求的过滤器链运行请求，筛选器由虚线分隔的原因是，筛选器可以在发送代理请求之前和之后运行逻辑，所有“前置”过滤器逻辑均被执行，然后发出代理请求，发出代理请求后，将运行“后”过滤器逻辑；

## 路由谓词工厂

Spring Cloud Gateway将路由匹配作为Spring WebFlux HandlerMapping基础架构的一部分，Spring Cloud Gateway包括许多内置的路由谓词工厂，所有这些谓词都与HTTP请求的不同属性匹配，可以将多个路由谓词工厂结合使用；

总共有11个路由谓词工厂：

1. The After Route Predicate Factory

2. The Before Route Predicate Factory

3. The Between Route Predicate Factory

4. The Cookie Route Predicate Factory

5. The Header Route Predicate Factory

6. The Host Route Predicate Factory

7. The Method Route Predicate Factory

8. The Path Route Predicate Factory

9. The Query Route Predicate Factory

10. The RemoteAddr Route Predicate Factory

11. The Weight Route Predicate Factory

### After路由谓词工厂

After route谓词工厂采用一个参数，即datetime（这是一个Java ZonedDateTime），该谓词匹配在指定日期时间之后发生的请求，以下示例配置了路由后谓词：



这条路由符合2017年1月20日17:42：47时间（[America/Denver]）之后的任何请求；

时间通过获取：System.*out*.println(ZonedDateTime.*now*());

### Before路由谓词工厂

Before路由谓词工厂采用一个参数，即datetime（这是一个Java ZonedDateTime），该谓词匹配在指定日期时间之前发生的请求，下面的示例配置路由之前谓词：



这条路由符合2017年1月20日17:42：47时间（[America/Denver]）之前的任何请求；

### [Between路由谓词工厂](https://docs.spring.io/spring-cloud-gateway/docs/2.2.4.RELEASE/reference/html/" \l "the-between-route-predicate-factory)

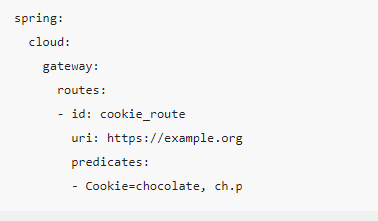
### 路由谓词之间的工厂使用两个参数datetime1和datetime2，它们是java ZonedDateTime对象，该谓词匹配在datetime1之后和datetime2之前发生的请求，datetime2参数必须在datetime1之后，以下示例配置了路由之间的谓词：



该路线与2017年1月20日山区时间（丹佛）之后和2017年1月21日17:42山区时间（丹佛）之前的任何请求相匹配，这对于维护时段可能很有用；

### [Cookie 路由谓词工厂](https://docs.spring.io/spring-cloud-gateway/docs/2.2.4.RELEASE/reference/html/" \l "the-cookie-route-predicate-factory)

### Cookie路由谓词工厂采用两个参数，即cookie名称和一个regexp（这是Java正则表达式），该谓词匹配具有给定名称且其值与正则表达式匹配的cookie，以下示例配置Cookie路由谓词工厂：

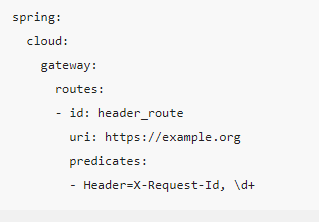


此路由匹配具有名为Chocolate的cookie的请求，该cookie的值与ch.p正则表达式匹配；

举例：curl http://192.168.0.104/index --cookie token=123456

### Header 路由谓词工厂

header 路由谓词工厂使用两个参数，header 名称和一个regexp（这是Java正则表达式），该谓词与具有给定名称的header 匹配，该header 的值与正则表达式匹配，以下示例配置标头路由谓词：

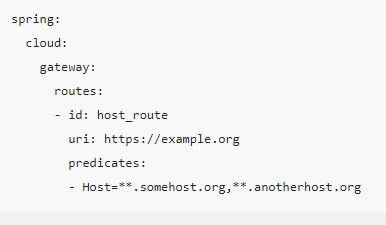


如果请求具有名为X-Request-Id的标头，且其值与\ d +正则表达式匹配（即，其值为一个或多个数字），则此路由匹配；

举例：curl http://192.168.0.104/index --header "X-Request-Id:19228"

### Host 路由谓词工厂

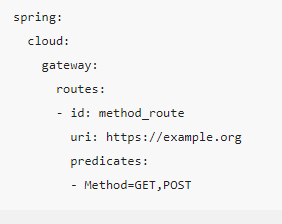
host路由谓词工厂使用一个参数：主机名模式列表，以下示例配置主机路由谓词：



还支持URI模板变量（例如{sub} .myhost.org），如果请求的主机标头的值为www.somehost.org或beta.somehost.org或www.anotherhost.org，则此路由匹配；

### Method 路由谓词工厂

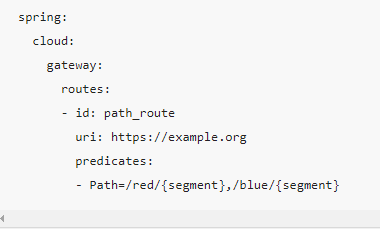
方法路由谓词工厂使用方法参数，该参数是一个或多个参数：要匹配的HTTP方法，以下示例配置方法route谓词：



如果请求方法是GET或POST，则此路由匹配；

### Path路由谓词工厂

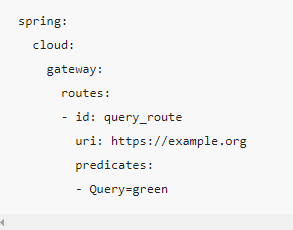
路径路由谓词工厂使用两个参数：Spring PathMatcher模式列表和一个称为matchOptionalTrailingSeparator的可选标志，以下示例配置路径路由谓词：



如果请求路径为例如/red/1或/red/blue或/blue/green，则此路由匹配；

### Query路由谓词工厂

查询路由谓词工厂采用两个参数：必需的参数和可选的regexp（这是Java正则表达式），以下示例配置查询路由谓词：



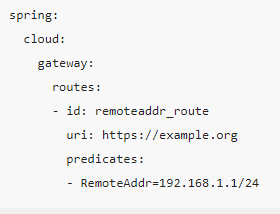
如果请求包含green查询参数，则前面的路由匹配；



如果请求包含值与gree匹配的red查询参数，则上述路由匹配；

### RemoteAddr 路由谓词工厂

RemoteAddr路由谓词工厂使用源列表（最小大小为1），这些源是标记（IPv4或IPv6）字符串，例如192.168.0.1/16（其中192.168.0.1是IP地址，而16是子网掩码）），下面的示例配置RemoteAddr路由谓词：



如果请求的远程地址是例如192.168.1.10，则此路由匹配；

### Weight 路由谓词工厂

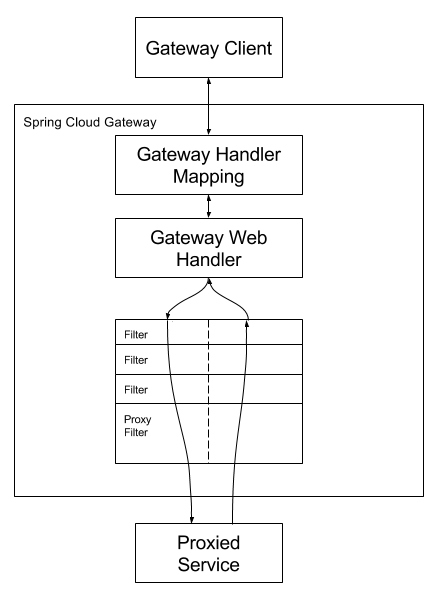
权重路由谓词工厂采用两个参数：group和weight（一个int），权重是按组计算的，以下示例配置权重路由谓词：



这条路由会将约80％的流量转发至weighthigh.org，并将约20％的流量转发至weightlow.org；

## GatewayFilter 工厂

路由过滤器允许以某种方式修改传入的HTTP请求或传出的HTTP响应，Spring Cloud Gateway包括许多内置的GatewayFilter工厂；

****

总共有31个GatewayFilter工厂：

1. The AddRequestHeader GatewayFilter Factory

2. The AddRequestParameter GatewayFilter Factory

3. The AddResponseHeader GatewayFilter Factory

4. The DedupeResponseHeader GatewayFilter Factory

5. The Hystrix GatewayFilter Factory

6. Spring Cloud CircuitBreaker GatewayFilter Factory

7. The FallbackHeaders GatewayFilter Factory

8. The MapRequestHeader GatewayFilter Factory

9. The PrefixPath GatewayFilter Factory

10. The PreserveHostHeader GatewayFilter Factory

11. The RequestRateLimiter GatewayFilter Factory

12. The RedirectTo GatewayFilter Factory

13. The RemoveRequestHeader GatewayFilter Factory

14. RemoveResponseHeader GatewayFilter Factory

15. The RemoveRequestParameter GatewayFilter Factory

16. The RewritePath GatewayFilter Factory

17. RewriteLocationResponseHeader GatewayFilter Factory

18. The RewriteResponseHeader GatewayFilter Factory

19. The SaveSession GatewayFilter Factory

20. The SecureHeaders GatewayFilter Factory

21. The SetPath GatewayFilter Factory

22. The SetRequestHeader GatewayFilter Factory

23. The SetResponseHeader GatewayFilter Factory

24. The SetStatus GatewayFilter Factory

25. The StripPrefix GatewayFilter Factory

26. The Retry GatewayFilter Factory

27. The RequestSize GatewayFilter Factory

28. The SetRequestHost GatewayFilter Factory

29. Modify a Request Body GatewayFilter Factory

30. Modify a Response Body GatewayFilter Factory

31. Default Filters