## 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、ribbon、feign（spring cloud）

## What is Feign？

Feign 是 Netflix 公司开发的一个声明式的REST调用客户端；（调用远程的restful风格的http接口的一个组件）

Github：<https://github.com/OpenFeign/feign>

Feign使编写Java HTTP客户端更加容易；

目前最新版本：[OpenFeign 10.10.1](https://github.com/OpenFeign/feign/releases/tag/10.10.1)

<https://github.com/OpenFeign/feign/releases>

Feign 10.x及更高版本基于Java 8构建，并且可在Java 9、10和11上运行，对于需要JDK 6兼容性的应用程序，需要使用Feign 9.x；

我们平时大部分时间都说Feign，实际上这个项目的全称是叫OpenFeign；

Http远程调用组件其实很多，比如：

HttpURLConnection（JDK） java.net.\*

HttpClient（apache）

RestTemplate（Spring）

OkHttp（android）

Feign（Netflix）

Spring Cloud Feign对Ribbon负载均衡进行了简化，在其基础上进行了进一步的封装，在配置上大大简化了开发工作，它是一种声明式的调用方式，它的使用方法是定义一个接口，然后在接口上添加注解，使其支持了Spring MVC标准注解和HttpMessageConverters，Feign可以与Ribbon组合使用以支持负载均衡；

restTemplate + ribbon 和 feign + ribbon（都是负载均衡远程http调用）

## Feign能干什么？

Feign旨在简化微服务消费方（调用者，客户端）代码的开发，前面在使用Ribbon+RestTemplate进行服务调用时，利用RestTemplate对http请求的封装处理，形成了一套模版化的调用方式，但是在实际开发中，由于服务提供者提供的接口非常多，一个接口也可能会被多处调用，Feign在Ribbon+RestTemplate的基础上做了进一步封装，在Feign封装之后，我们只需创建一个接口并使用注解的方式来配置，即可完成对服务提供方的接口绑定，简化了使用Ribbon + RestTemplate的调用，自动封装了服务调用客户端，减少代码开发量；

## 使用Feign实现消费者

Feign本身是一个组件，也就是一个jar包，spring cloud对这个jar包制作了一个starter，方便我们开发使用；

1、添加依赖

要添加的依赖主要是spring-cloud-starter-openfeign，如下：

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

格式：spring-cloud-starter-[机构]-[模块]

比如：

spring-cloud-starter-openfeign

spring-cloud-starter-alibaba-nacos-discovery

2、声明服务

定义一个EchoService接口，通过@FeignClient 注解来指定服务名称，进而绑定服务，然后再通过SpringMVC中提供的注解来绑定服务提供者提供的接口， 如下：

@FeignClient(name = "29-nacos-discovery-provider")  
public interface EchoService {  
 @GetMapping("/echo/{str}")  
 String echo(@PathVariable("str") String str);

}

这相当于绑定了一个名叫29-nacos-discovery-provider的服务提供者提供的/echo/{str}接口；

3、开启feign注解

在项目入口类上添加@EnableFeignClients注解表示开启Spring Cloud Feign的支持功能；

4、使用Controller调用服务

接着来创建一个Controller来调用远程的服务，如下：

**public class** EchoController {  
 @Autowired  
 **private** EchoService **echoService**;  
 */\*\*  
 \* 使用feign进行调用  
 \** ***@return*** *\*/* @RequestMapping(**"/feign/{str}"**)  
 **public** ResultObject feign(@PathVariable String str) {  
 *//调用远程的一个controller, restful的调用* **return echoService**.echo(str);  
 }  
}

5、测试

依次启动注册中心、服务提供者和feign实现服务消费者，然后访问如下地址：

<http://localhost:8090/feign/123456>

## 使用Feign实现服务消费调用

**负载均衡：**

我们知道，Spring Cloud 提供了Ribbon来实现负载均衡，使用Ribbon直接注入一个RestTemplate对象即可，Ribbon为RestTemplate已经做好了负载均衡的配置；

在Spring Cloud下，使用Feign也是直接可以实现负载均衡调用，定义一个有@FeignClient注解的接口，然后使用@RequestMapping注解到方法上映射远程的REST服务，此方法也是做好负载均衡配置的，底层也是采用Ribbon进行负载均衡；

通过feign只需要定义服务绑定接口且以声明式的方法，非常优雅而简单的实现了服务调用；

Feign配置超时时间：

feign.client.config.default.read-timeout=1  
feign.client.config.default.connect-timeout=1  
feign.client.config.29-nacos-discovery-provider.read-timeout=1

## Feign脱离Ribbon的使用

@FeignClient(name = "baidu",  
 url = "http://www.baidu.com")  
public interface FeignService {  
 @GetMapping  
 String baidu();  
}

以上是直接调用某一个远程http接口，没有使用ribbon进行负载均衡；

## 流量防卫兵Sentinel

在分布式系统里，许多服务之间通过远程调用实现信息交互，调用时不可避免会出现调用失败，比如超时、异常等原因导致调用失败，Sentinel能够保证在一个服务出问题的情况下，不会导致整体服务失败，避免级联故障（服务雪崩），以提高分布式系统的弹性；

比如电商中的用户下订单，我们有两个服务，一个下订单服务，一个减库存服务，

当用户下订单时调用下订单服务，然后下订单服务又调用减库存服务，如果减库

存服务响应延迟或者没有响应，则会造成下订单服务的线程挂起等待，如果大量

的用户请求下订单，或导致大量的请求堆积，引起下订单服务也不可用，如果还

有另外一个服务依赖于订单服务，比如用户服务，它需要查询用户订单，那么用

户服务查询订单也会引起大量的延迟和请求堆积，导致用户服务也不可用。

所以在微服务架构中，很容易造成服务故障的蔓延，引发整个微服务系统瘫痪不

可用。

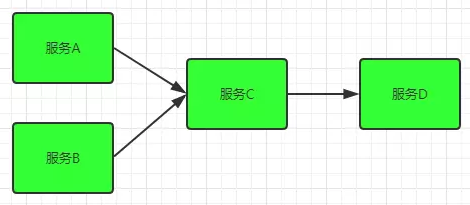


图1

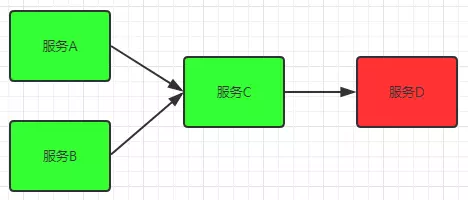


图2

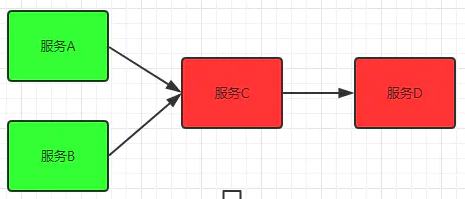


图3

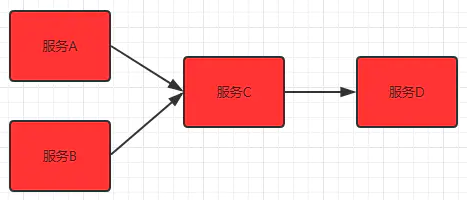
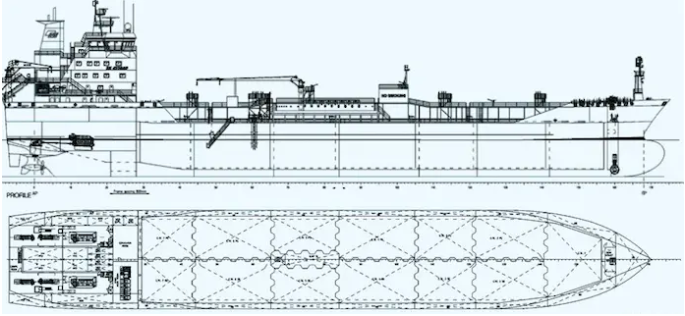


图4

常用的容错方案或思想

1. 超时，设置比较短的超时时间，调用不成功，很短时间就释放线程，避免大量线程堵塞等待，导致服务cpu、内存等资源飙高；(快速失败)
2. 限流，超过设置的阈值就拒绝，比如评估系统的QPS是3000，那么就可以设置限流阈值是2800；
3. 仓壁保护，就是一艘船不是一个船舱，而是把一个船舱划分为多个船舱，某个船舱进水了，其他船舱都不受到影响；



4、断路器，熔断器也有叫断路器，他们表示同一个意思，最早来源于微服务之父 Martin Fowler 的论文 CircuitBreaker 一文，“熔断器”本身是一种开关装置，用于在电路上保护线路过载，当线路中有电器发生短路时，能够及时切断故障电路，防止发生过载、发热甚至起火等严重后果。

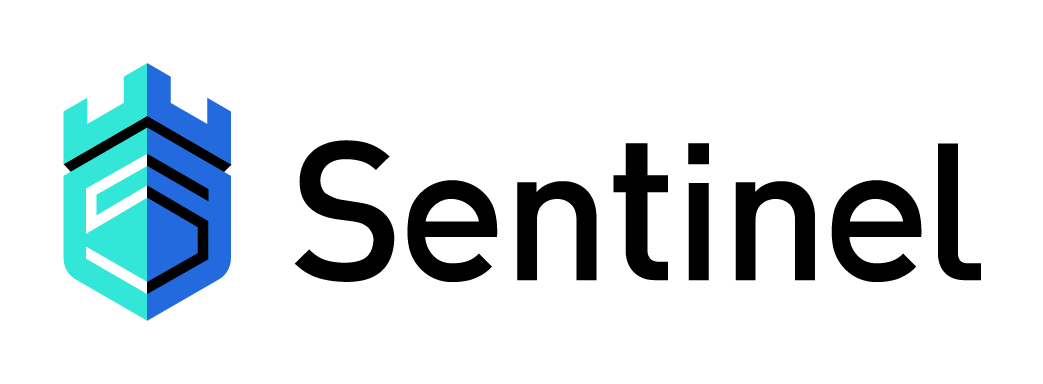
## What is Sentinel？

随着微服务的流行，服务与服务之间的调用稳定性变得越来越重要；

1、当服务访问量达到一定程度，流量扛不住的时候，该如何处理？

2、服务之间相互依赖，当服务A出现响应时间过长，影响到服务B的响应，进而产生连锁反应，直至影响整个依赖链上的所有服务，该如何处理？

这是分布式、微服务开发不可避免的问题，Sentinel以流量为切入点，从流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性；



2012 年，Sentinel 诞生，主要功能是提供请求流量控制；

2013-2017 年，Sentinel在阿里巴巴集团内部大量用于生产实践，成为基础技术模块，覆盖了所有的核心场景；

2018 年，Sentinel 对外开源，并持续演进和版本迭代。

Sentinel在经过阿里巴巴内部一系列秒杀大促，特别是双11这样电商大促中的锤炼，目前已有不少企业在使用开源版本和云版本的Sentinel，包括顺丰、vivo、每日优鲜、拼多多、易企秀、爱奇艺、融金所、VIPKID、喜马拉雅FM、百融金服等；

更多使用者：<https://github.com/alibaba/Sentinel/issues/18>

按照官方的定义：

A lightweight powerful flow control component enabling reliability and monitoring for microservices.

轻量级的流量控制、熔断降级Java 组件，是分布式系统的流量防卫兵；

Github：<https://github.com/alibaba/Sentinel>



Sentinel主要由两部分组成：

1. 核心库（Java 客户端）：Sentinel的核心库不依赖任何第三方框架/库，能够运行于所有 Java环境，同时对 Dubbo / SpringBoot / Spring Cloud 等框架也有很好的支持；
2. 控制台（Dashboard）基于 Spring Boot 开发，打包后可以直接运行，不需要额外的 Tomcat 等应用容器；

## Spring Cloud Alibaba Sentinel进行容错限流

1、添加依赖：

*<!--spring-cloud-starter-alibaba-sentinel-->*<dependency>  
 <groupId>com.alibaba.cloud</groupId>  
 <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>  
</dependency>

*<!--spring-boot-starter-actuator-->*<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>

2、配置文件：

*#指定sentinel-dashboard控制台的连接地址*spring.cloud.sentinel.transport.dashboard=192.168.172.128:8080

3、我们可以通过访问http://{微服务注册的ip地址}:8719/api接口查看微服务暴露给Sentinel控制台调用的API列表；比如访问：<http://localhost:8719/api>

## Sentinel Dashboard管控台

Sentinel Dashboard （Sentinel控制台）：基于 Spring Boot 开发，是一个单独的应用，打包后可以直接通过spring-boot方式进行启动，不需要额外的 Tomcat 等应用容器，主要提供一个轻量级的控制台，它提供机器发现、单机资源实时监控、集群资源汇总，以及规则管理的功能。

只需要对应用进行简单的配置，就可以使用这些功能。

下载官方提供的Sentinel Dashboard的jar包

<https://github.com/alibaba/Sentinel/releases>

启动

使用如下命令启动编译后的控制台：

java -jar sentinel-dashboard-1.7.2.jar

访问

登录账号，默认用户名密码都是 sentinel

Sentinel 会在客户端首次调用的时候进行初始化，开始向控制台发送心跳包，所以要确保客户端有访问量；

Sentinel Dashboard是一个独立的web应用，可以接受客户端的连接，然后与客户端之间进行通讯，他们之间使用http协议进行通讯，客户端代码需要引入：

<dependency>

<groupId>com.alibaba.csp</groupId>

<artifactId>sentinel-transport-simple-http</artifactId>

<version>1.6.3</version>

</dependency>

Spring cloud alibaba如下的依赖已经包含了上面的依赖，所以上面的依赖不需要再单独添加；

*<!--spring-cloud-starter-alibaba-sentinel-->*<dependency>  
 <groupId>com.alibaba.cloud</groupId>  
 <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>  
</dependency>

## Sentinel流控规则



资源名：一般是我们的请求路径；

针对来源：来自于哪个应用；

阈值类型：分为QPS或线程数；

单机阈值：单个节点的QPS或线程数阈值；

是否集群：被请求的服务是否集群部署；

流控模式：

1. 直接，就是直接对该资源进行控制；
2. 关联，关联某一个资源（/app2），被关联的资源/app2达到阈值2，则限制当前资源/test的访问；
3. 链路，记录指定链路上的流量；

流控效果：

1. 快速失败 ，直接限制；
2. Warm Up，根据coldFactor（默认为3）的值，从 阈值/coldFactor，经过预热的时长，才达到设置的QPS阈值，比如设置QPS阈值为100，那么100/3 =33，用33作为最初的阈值，然后在10秒到达100后再开始限流；
3. 排队等待，在QPS阈值到达后，新的请求就等待，直到超时，可以适用于突发流量的请求；

## Spring Cloud Alibaba Sentinel

Sentinel为springboot程序提供了一个starter依赖，由于sentinel starter依赖默认情况下就会为所有的HTTP服务提供限流埋点，所以在springboot 中的Controller都可以受到Sentinel的保护；

只需为应用添加 spring-cloud-starter-alibaba-sentinel依赖，所有的HTTP接口都能获得Sentinel保护，当然，我们还需要为Sentinel配置保护的规则；