## 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

## Spring Cloud Alibaba Sentinel降级规则



降级策略：

（1）RT: 平均响应时间 (DEGRADE\_GRADE\_RT)，当 1s 内持续进入 N 个请求，对应时刻的平均响应时间（秒级）均超过阈值（count，以 ms 为单位），那么在接下的时间窗口（DegradeRule 中的 timeWindow，以 s 为单位）之内，对这个方法的调用都会自动地熔断（抛出 DegradeException），注意 Sentinel 默认统计的 RT 上限是 4900 ms，超出此阈值的都会算作 4900 ms，若需要变更此上限可以通过启动配置项 -Dcsp.sentinel.statistic.max.rt=xxx 来配置；

（2）异常比例：异常比例 (DEGRADE\_GRADE\_EXCEPTION\_RATIO)是指当资源的每秒异常总数占通过量的比值超过阈值（DegradeRule 中的 count）之后，资源进入降级状态，即在接下的时间窗口（DegradeRule 中的 timeWindow，以 s 为单位）之内，对这个方法的调用都会自动地返回，异常比率的阈值范围是 [0.0, 1.0]，代表 0% - 100%；

（3）异常数：异常数 (DEGRADE\_GRADE\_EXCEPTION\_COUNT)是指当资源近1分钟的异常数目超过阈值之后会进行熔断，注意由于统计时间窗口是分钟级别的，若timeWindow小于 60s，则结束熔断状态后仍可能再进入熔断状态；

具体请参考视频演示；

## Spring Cloud Alibaba Sentinel热点规则

何为热点？热点即经常访问的数据，很多时候我们希望统计某个热点数据中访问频次最高的 Top K 数据，并对其访问进行限制，比如：

商品 ID 为参数，统计一段时间内最常购买的商品 ID 并进行限制；

用户 ID 为参数，针对一段时间内频繁访问的用户 ID 进行限制；

热点参数限流会统计传入参数中的热点参数，并根据配置的限流阈值与模式，对包含热点参数的资源调用进行限流，热点参数限流可以看做是一种特殊的流量控制，仅对包含热点参数的资源调用生效；



热点规则需要使用@SentinelResource("app")注解，否则不生效；

参数必须是7种基本数据类型才会生效；

热点参数可以对某个具体的热点参数进行限流；

具体请参考视频演示；

## Spring Cloud Alibaba Sentinel系统规则



**Load**，Load自适应（仅对Linux/Unix-like机器生效）：系统的load1作为启发指标，进行自适应系统保护，当系统load1超过设定的启发值，且系统当前的并发线程数超过估算的系统容量时才会触发系统保护，系统容量由系统的maxQps \* minRt估算得出，设定参考值一般是 CPU cores \* 2.5；

**平均 RT**：当单台机器上所有入口流量的平均RT达到阈值即触发系统保护，单位是毫秒；

**并发线程数：**当单台机器上所有入口流量的并发线程数达到阈值即触发系统保护；  
**入口 QPS：**当单台机器上所有入口流量的 QPS 达到阈值即触发系统保护；

**CPU usage**（1.5.0+ 版本）：当系统 CPU 使用率超过阈值即触发系统保护（取值范围 0.0-1.0），比较灵敏；

具体请参考视频演示；

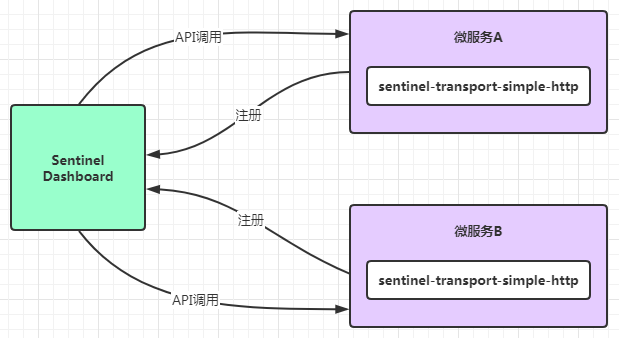
## Spring Cloud Alibaba Sentinel授权规则

实现RequestOriginParser接口，在接口方法中实现区分来源；

@Component  
public class MyRequestOriginParser implements RequestOriginParser {  
 @Override  
 public String parseOrigin(HttpServletRequest request) {  
 String origin = request.getParameter("origin");  
 return origin;  
 }  
}

具体请参考视频演示；

## Spring Cloud Alibaba Sentinel Dashboard通信原理



微服务暴露给Sentinel Dashboard的API接口列表：<http://localhost:8719/api>

懒加载、饥饿加载：  
*#true表示饥饿加载*spring.cloud.sentinel.eager=true

Sentinel Dashboard控制台配置项，可以修改；



比如修改Sentinel Dashboard控制台登录密码；

## Spring Cloud Alibaba Sentinel三种保护应用方式

1、直接拦截我们所有controller的请求url路径；

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；

底层通过一个拦截器对请求url进行拦截：

com.alibaba.csp.sentinel.adapter.spring.webmvc.SentinelWebInterceptor

可以通过如下配置关闭对微服务的保护：

*#关闭sentinel对controller的url的保护*spring.cloud.sentinel.filter.enabled=false

具体请参考视频演示；

2、通过代码方式保护应用；

@GetMapping("/test3/{app}")  
public String test3(@PathVariable("app") String app) {  
 System.*out*.println("/test3/{app} --> " + app);  
 Entry entry = null;  
 try {  
 entry = SphU.*entry*("test3");  
 return restTemplate.getForObject("http://29-nacos-discovery-provider/test", String.class);  
 } catch (BlockException e) {  
 e.printStackTrace();  
 return "熔断降级了...";  
 } finally {  
 if (entry != null) {  
 entry.exit();  
 }  
 }  
}

这种方式就是手动写代码，稍微繁琐一些；具体请参考视频演示；

3、通过@SentinelResource(value = "app")注解保护应用；

| 属性 | 作用 | 是否必须 |
| --- | --- | --- |
| value | 资源名称 | 是 |
| entryType | entry类型，标记流量的方向，取值IN/OUT，默认是OUT | 否 |
| blockHandler | 处理BlockException的函数名称。函数要求： 1. 必须是 public 2.返回类型与原方法一致 3. 参数类型需要和原方法相匹配，并在最后加 BlockException 类型的参数。 4. 默认需和原方法在同一个类中。若希望使用其他类的函数，可配置 blockHandlerClass ，并指定blockHandlerClass里面的方法。 | 否 |
| blockHandlerClass | 存放blockHandler的类。对应的处理函数必须static修饰，否则无法解析，其他要求：同blockHandler。 | 否 |
| fallback | 用于在抛出异常的时候提供fallback处理逻辑。fallback函数可以针对所有类型的异常（除了 exceptionsToIgnore 里面排除掉的异常类型）进行处理。函数要求： 1. 返回类型与原方法一致 2. 参数类型需要和原方法相匹配，Sentinel 1.6开始，也可在方法最后加 Throwable 类型的参数。 3.默认需和原方法在同一个类中。若希望使用其他类的函数，可配置 fallbackClass ，并指定fallbackClass里面的方法。 | 否 |
| fallbackClass | 存放fallback的类。对应的处理函数必须static修饰，否则无法解析，其他要求：同fallback。 | 否 |
| defaultFallback | 用于通用的 fallback 逻辑。默认fallback函数可以针对所有类型的异常（除了 exceptionsToIgnore 里面排除掉的异常类型）进行处理。若同时配置了 fallback 和 defaultFallback，以fallback为准。函数要求： 1. 返回类型与原方法一致 2. 方法参数列表为空，或者有一个 Throwable 类型的参数。 3. 默认需要和原方法在同一个类中。若希望使用其他类的函数，可配置 fallbackClass ，并指定 fallbackClass 里面的方法。 | 否 |
| exceptionsToIgnore | 指定排除掉哪些异常。排除的异常不会计入异常统计，也不会进入fallback逻辑，而是原样抛出。 | 否 |
| exceptionsToTrace | 需要trace的异常 | Throwable |

具体请参考视频演示；

## Spring Cloud Alibaba Sentinel对RestTemplate流控熔断

*#true开启sentinel对resttemplate的支持，false则关闭*resttemplate.sentinel.enabled=true

@SentinelRestTemplate(*/\*blockHandler="blockA", blockHandlerClass=*

*MyBlockHandlerClass.class\*/* fallback="fallbackA", fallbackClass = MyBlockHandlerClass.class)  
@LoadBalanced *//负载均衡*@Bean  
public RestTemplate restTemplate() {  
 return new RestTemplate();  
}

@GetMapping("/test2")  
public String test2() {  
 return restTemplate.getForObject("http://29-nacos-discovery-provider/test", String.class);  
}

响应结果对象：SentinelClientHttpResponse;

具体请参考视频演示；

## Spring Cloud Alibaba Sentinel对Feign流控熔断

*#true开启sentinel对feign的支持，false则关闭*feign.sentinel.enabled=true

@FeignClient(name = "29-nacos-discovery-provider",  
 */\*fallback = EchoServiceFallback.class,\*/* fallbackFactory = EchoServiceFallbackFactory.class,  
 configuration = FeignConfiguration.class)  
public interface EchoService {

@GetMapping("/notFound")

String notFound();

}

具体请参考视频演示；