

主讲老师: Fox老师

1. RestTemplate整合Sentinel

Spring Cloud Alibaba Sentinel 支持对 RestTemplate 的服务调用使用 Sentinel 进行保护, 在构造 RestTemplate bean的时候需要加上 @SentinelRestTemplate 注解。

@SentinelRestTemplate 注解的属性支持限流(blockHandler, blockHandlerClass) 和降级(fallback, fallbackClass)的处理。

引入依赖

```
1 <!--加入nacos-client-->
2 <dependency>
3   <groupId>com.alibaba.cloud</groupId>
4   <artifactId>spring-cloud-alibaba-nacos-discovery</artifactId>
5 </dependency>
6
7 <!--加入ribbon-->
8 <dependency>
9   <groupId>org.springframework.cloud</groupId>
10  <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
11 </dependency>
12
13 <!--加入sentinel-->
14 <dependency>
15   <groupId>com.alibaba.cloud</groupId>
16   <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
17 </dependency>
18
19 <!--加入actuator-->
20 <dependency>
21   <groupId>org.springframework.boot</groupId>
22   <artifactId>spring-boot-starter-actuator</artifactId>
23 </dependency>
```

RestTemplate添加@SentinelRestTemplate注解

```
1 @Bean
2 @LoadBalanced
3 @SentinelRestTemplate(
4   blockHandler = "handleException", blockHandlerClass = GlobalExceptionHandler.class,
5   fallback = "fallback", fallbackClass = GlobalExceptionHandler.class
6 )
```

```

7 public RestTemplate restTemplate() {
8     return new RestTemplate();
9 }

```

异常处理类定义需要注意的是该方法的参数和返回值跟

org.springframework.http.client.ClientHttpRequestInterceptor#interceptor 方法一致，其中参数多出了一个 BlockException 参数用于获取 Sentinel 捕获的异常。

源码跟踪:

com.alibaba.cloud.sentinel.custom.SentinelBeanPostProcessor

com.alibaba.cloud.sentinel.custom.SentinelProtectInterceptor#intercept

```

1 // UserController.java
2 @RequestMapping(value = "/findOrderByUserId/{id}")
3 // @SentinelResource(value = "findOrderByUserId",
4 // fallback = "fallback", fallbackClass = ExceptionUtil.class,
5 // blockHandler = "handleException", blockHandlerClass = ExceptionUtil.class
6 // )
7 public R findOrderByUserId(@PathVariable("id") Integer id) {
8     //ribbon实现
9     String url = "http://mall-order/order/findOrderByUserId/"+id;
10    R result = restTemplate.getForObject(url,R.class);
11
12    return result;
13 }
14
15
16 public class GlobalExceptionUtil {
17     /**
18      * 注意: static修饰, 参数类型不能出错
19      * @param request org.springframework.http.HttpServletRequest
20      * @param body
21      * @param execution
22      * @param ex
23      * @return
24      */
25     public static SentinelClientHttpResponse handleException(HttpServletRequest request,
26     byte[] body, ClientHttpRequestExecution execution, BlockException ex) {
27         R r = R.error(-1, "===被限流啦===");
28         try {
29             return new SentinelClientHttpResponse(new
30             ObjectMapper().writeValueAsString(r));
31         } catch (JsonProcessingException e) {
32             e.printStackTrace();
33         }
34         return null;
35     }
36 }

```

```

35
36 public static SentinelClientHttpResponse fallback(HttpRequest request,
37 byte[] body, ClientHttpRequestExecution execution, BlockException ex) {
38 R r = R.error(-2, "===被异常降级啦===");
39 try {
40 return new SentinelClientHttpResponse(new
41 ObjectMapper().writeValueAsString(r));
42 } catch (JsonProcessingException e) {
43 e.printStackTrace();
44 }
45 return null;
46 }

```

添加yml配置

```

1 server:
2   port: 8801
3
4 spring:
5   application:
6     name: mall-user-sentinel-ribbon-demo #微服务名称
7
8   #配置nacos注册中心地址
9   cloud:
10     nacos:
11       discovery:
12         server-addr: 127.0.0.1:8848
13
14     sentinel:
15       transport:
16         # 添加sentinel的控制台地址
17         dashboard: 127.0.0.1:8080
18         # 指定应用与Sentinel控制台交互的端口，应用本地会起一个该端口占用的HttpServer
19         port: 8719
20
21     #暴露actuator端点 http://localhost:8800/actuator/sentinel
22     management:
23       endpoints:
24         web:
25           exposure:
26             include: '*'
27
28     #true开启sentinel对resttemplate的支持，false则关闭 默认true
29     resttemplate:
30       sentinel:

```

31 enabled: true

Sentinel RestTemplate 限流的资源规则提供两种粒度：

- httpmethod:schema://host:port/path：协议、主机、端口和路径
- httpmethod:schema://host:port：协议、主机和端口

GET:http://mall-order/order/findOrderByUserId/1	0	0	0	0	1	0	+ 流控	+ 降级	+ 热点
							+ 授权		
GET:http://mall-order	0	0	0	0	8	0	+ 流控	+ 降级	+ 热点
							+ 授权		
GET:http://mall-order/order/findOrderByUserId/4	0	0	0	0	7	0	+ 流控	+ 降级	+ 热点
							+ 授权		

测试限流

资源名	GET:http://mall-order/order/findOrderByUserId/1		
针对来源	default		
阈值类型	<input checked="" type="radio"/> QPS <input type="radio"/> 线程数	单机阈值	1
是否集群	<input type="checkbox"/>		

测试降级

修改服务提供者mall-order

```
1 @RequestMapping("/findOrderByUserId/{userId}")
2 public R findOrderByUserId(@PathVariable("userId") Integer userId) {
3
4     //模拟异常
5     if(userId==5){
6         throw new IllegalArgumentException("非法参数异常");
7     }
8
9     log.info("根据userId:"+userId+"查询订单信息");
10    List<OrderEntity> orderEntities = orderService.listByUserId(userId);
11    return R.ok().put("orders", orderEntities);
12 }
```

编辑降级规则

资源名	GET:http://mall-order/order/findOrderByUserId/5			
熔断策略	<input type="radio"/> 慢调用比例 <input type="radio"/> 异常比例 <input checked="" type="radio"/> 异常数			
异常数	1			
熔断时长	2	s	最小请求数	2

← → ↺ ⓘ localhost:8801/user/findOrderByUserId/5

```
{
  msg: "===被异常降级啦===",
  code: -2
}
```

2. OpenFeign整合Sentinel

Sentinel 适配了 Feign 组件。如果想使用，除了引入 spring-cloud-starter-alibaba-sentinel 的依赖外还需要 2 个步骤：

配置文件打开 Sentinel 对 Feign 的支持：feign.sentinel.enabled=true

```
feign:
  sentinel:
    enabled: true    #开启sentinel对feign的支持
```

加入 spring-cloud-starter-openfeign 依赖使 Sentinel starter 中的自动化配置类生效：

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-openfeign</artifactId>
4 </dependency>
5
```

在Feign的声明式接口上添加fallback属性

```
1 @FeignClient(value = "mall-order", path = "/order", fallback =
  FeignClientFallback.class)
2 public interface OrderFeignService {
3
4   @RequestMapping("/findOrderByUserId/{userId}")
5   public R findOrderByUserId(@PathVariable("userId") Integer userId);
6
7 }
```

```

8
9 @Component //必须交给spring 管理
10 public class FallbackOrderFeignService implements OrderFeignService {
11     @Override
12     public R findOrderByUserId(Integer userId) {
13         return R.error(-1,"=====服务降级了=====");
14     }
15 }

```

添加fallbackFactory属性

```

1 @Component
2 public class FallbackOrderFeignServiceFactory implements FallbackFactory<OrderFeignService> {
3     @Override
4     public OrderFeignService create(Throwable throwable) {
5
6         return new OrderFeignService() {
7             @Override
8             public R findOrderByUserId(Integer userId) {
9                 return R.error(-1,"=====服务降级了=====");
10             }
11         };
12     }
13 }

```

UserController

```

1 @Autowired
2 OrderFeignService orderFeignService;
3
4 @RequestMapping(value = "/findOrderByUserId/{id}")
5 public R findOrderByUserId(@PathVariable("id") Integer id) {
6     //feign调用
7     R result = orderFeignService.findOrderByUserId(id);
8     return result;
9 }

```

注意：主启动类上加上@EnableFeignClients注解，开启Feign支持

测试

关闭mall-order服务，访问<http://localhost:8801/user/findOrderByUserId/4>自动降级了

```
{
  msg: "====服务降级了====",
  code: -1
}
```

3. Sentinel整合Dubbo实战

Sentinel 提供 Dubbo 的相关适配 Sentinel Dubbo Adapter，主要包括针对 Service Provider 和 Service Consumer 实现的 Filter。相关模块：

- `sentinel-apache-dubbo-adapter`（兼容 Apache Dubbo 2.7.x 及以上版本，自 Sentinel 1.5.1 开始支持）
- `sentinel-dubbo-adapter`（兼容 Dubbo 2.6.x 版本）

引入此依赖后，Dubbo 的服务接口和方法（包括调用端和服务端）就会成为 Sentinel 中的资源，在配置了规则后就可以自动享受到 Sentinel 的防护能力。

Sentinel Dubbo Adapter 还支持配置全局的 fallback 函数，可以在 Dubbo 服务被限流/降级/负载保护的时候进行相应的 fallback 处理。用户只需要实现自定义的 DubboFallback 接口，并通过 DubboAdapterGlobalConfig 注册即可。默认情况会直接将 BlockException 包装后抛出。同时，我们还可以配合 Dubbo 的 fallback 机制 来为降级的服务提供替代的实现。

Provider端

对服务提供方的流量控制可分为**服务提供方的自我保护能力**和**服务提供方对服务消费方的请求分配能力**两个维度。

Provider 用于向外界提供服务，处理各个消费者的调用请求。为了保护 Provider 不被激增的流量拖垮影响稳定性，可以给 Provider 配置 **QPS 模式** 的限流，这样当每秒的请求量超过设定的阈值时会自动拒绝多的请求。限流粒度可以是 *服务接口* 和 *服务方法* 两种粒度。若希望整个服务接口的 QPS 不超过一定数值，则可以为对应服务接口资源（resourceName 为**接口全限定名**）配置 QPS 阈值；若希望服务的某个方法的 QPS 不超过一定数值，则可以为对应服务方法资源（resourceName 为**接口全限定名:方法签名**）配置 QPS 阈值。

限流粒度可以是服务接口和服务方法两种粒度：

- 服务接口：resourceName 为 接口全限定名，如 `com.tuling.mall.service.UserService`

- 服务方法：resourceName 为 接口全限定名:方法签名，如
com.tuling.mall.service.UserService:getById(java.lang.Integer)

Consumer端

对服务提供方的流量控制可分为**控制并发线程数**和**服务降级**两个维度。

控制并发线程数

Service Consumer 作为客户端去调用远程服务。每一个服务都可能会依赖几个下游服务，若某个服务 A 依赖的下游服务 B 出现了不稳定的情况，服务 A 请求 服务 B 的响应时间变长，从而服务 A 调用服务 B 的线程就会产生堆积，最终可能耗尽服务 A 的线程数。我们通过用并发线程数来控制对下游服务 B 的访问，来保证下游服务不可靠的时候，不会拖垮服务自身。基于这种场景，推荐给 Consumer 配置**线程数模式**的限流，来保证自身不被不稳定服务所影响。采用基于线程数的限流模式后，我们不需要再显式地去进行线程池隔离，Sentinel 会控制资源的线程数，超出的请求直接拒绝，直到堆积的线程处理完成，可以达到**信号量隔离**的效果。

服务降级

当服务依赖于多个下游服务，而某个下游服务调用非常慢时，会严重影响当前服务的调用。这里我们可以利用 Sentinel 熔断降级的功能，为调用端配置基于平均 RT 的降级规则。这样当调用链路中某个服务调用的平均 RT 升高，在一定的次数内超过配置的 RT 阈值，Sentinel 就会对此调用资源进行降级操作，接下来的调用都会立刻拒绝，直到过了一段设定的时间后才恢复，从而保护服务不被调用端短板所影响。同时可以配合 fallback 功能使用，在被降级的时候提供相应的处理逻辑。

1.引入依赖

```
1 <dependency>
2   <groupId>com.alibaba.cloud</groupId>
3   <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
4 </dependency>
5 <!--Sentinel 对 Dubbo的适配 Apache Dubbo 2.7.x 及以上版本-->
6 <dependency>
7   <groupId>com.alibaba.csp</groupId>
8   <artifactId>sentinel-apache-dubbo-adapter</artifactId>
9 </dependency>
```

2.接入sentinel dashboard，yml中增加配置

```
1 spring:
```



```

2  cloud:
3  sentinel:
4  transport:
5  # 添加sentinel的控制台地址
6  dashboard: 127.0.0.1:8080
7
8  #暴露actuator端点
9  management:
10  endpoints:
11  web:
12  exposure:
13  include: '*'

```

3.consumer端配置流控规则测试

```

1  @RequestMapping("/info/{id}")
2  public User info(@PathVariable("id") Integer id) {
3  User user = null;
4  try {
5  user = userService.getById(id);
6  } catch (Exception e) {
7  e.printStackTrace();
8  }
9  return user;
10 }
11
12 @PostConstruct
13 public void init() {
14 DubboAdapterGlobalConfig.setConsumerFallback(
15 (invoker, invocation, ex) -> AsyncRpcResult.newDefaultAsyncResult(
16 new User(0, "===fallback=="), invocation));
17 }
18

```

spring-cloud-dubbo-
consumer-user-feign-sentinel
(1/1)

实时监控
簇点链路
流控规则
降级规则

spring-cloud-dubbo-
consumer-user-feign-sentinel

流控规则
192.168.3.1:8719
关键字
刷新

资源名	来源应用	流控模式	阈值类型	阈值	阈值模式	流控效果	操作
com.tuling.mall.service.UserService.getById(java.lang.Integer)	default	直接	QPS	3	单机	快速失败	编辑 删除

共 1 条记录, 每页 10 条记录

测试: <http://localhost:8082/user/info/1>

← → ↻ ⓘ localhost:8082//user/info/1

```
{
  id: 0,
  name: "===fallback=="
}
```

4.provider端配置流控规则测试

spring-cloud-dubbo-provider-user-feign-sentinel (1/1)
实时监控
簇点链路

流控规则

192.168.3.1:8721

关键字

刷新

资源名	来源应用	流控模式	阈值类型	阈值	阈值模式	流控效果	操作
com.tuling.mall.service.UserService:getId(java.lang.Integer)	default	直接	QPS	2	单机	快速失败	编辑 删除

共 1 条记录, 每页 10 条记录

```
1 @RequestMapping("/getById/{id}")
2 @SentinelResource("getId")
3 public User getById(@PathVariable("id") Integer id) {
4     User user = null;
5     try {
6         user = userMapper.getById(id);
7     } catch (Exception e) {
8         e.printStackTrace();
9     }
10    return user;
11 }
12
13 @PostConstruct
14 public void init() {
15     DubboAdapterGlobalConfig.setProviderFallback(
16         (invoker, invocation, ex) -> AsyncRpcResult.newDefaultAsyncResult(new User(0, "===provider fallback=="), invocation));
17 }
```

← → ↻ ⓘ localhost:8082//user/info/1

```
{
  id: 0,
  name: "===provider fallback=="
}
```

5. consumer中配置mock实现, 关闭provider服务, 测试mock降级

```
1 @DubboReference(mock = "com.tuling.mall.user.mock.UserServiceDubboMock")
2 private UserService userService;
3
4 public class UserServiceDubboMock implements UserService {
```

```
5  @Override
6  public List<User> list() {
7      return null;
8  }
9
10 @Override
11 public User getById(Integer id) {
12     return new User(0, "====mock====");
13 }
14 }
```

← → ↻ ⓘ localhost:8082//user/info/1

```
{
    id: 0,
    name: "====mock===="
}
```

文档: 09 Sentinel整合RestTemplate&openFegi...

链接: [http://note.youdao.com/noteshare?](http://note.youdao.com/noteshare?id=4453f26fca70f6640f9dd8be2960179b&sub=55CFA9A4B4614ABD9AAA04B08A58EC95)

[id=4453f26fca70f6640f9dd8be2960179b&sub=55CFA9A4B4614ABD9AAA04B08A58EC95](http://note.youdao.com/noteshare?id=4453f26fca70f6640f9dd8be2960179b&sub=55CFA9A4B4614ABD9AAA04B08A58EC95)