主讲老师：Fox老师

```
文档：09-2 Sentinel整合RestTemplate&openFe...
链接：http://note.youdao.com/noteshare?id=040a0e94a3243b992cfdb6f01e1d74e8&sub=3BC3777199324FDB9E3AA1E6AFC357FD
```

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

```
<!--加入nocas-client-->
<dependency>
 <groupId>com.alibaba.cloud</groupId>
 <artifactId>spring-cloud-alibaba-nacos-discovery</artifactId>
</dependency>

<!--加入ribbon-->
<dependency>
 <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
 </dependency>

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

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

RestTemplate添加@SentinelRestTemplate注解

```
@Bean
@LoadBalanced
@SentinelRestTemplate(
 blockHandler = "handleException",blockHandlerClass = GlobalExceptionHandler.class,
 fallback = "fallback",fallbackClass = GlobalExceptionHandler.class
)
public RestTemplate restTemplate() {
 return new RestTemplate();
}
```

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

```
// UserController.java
@RequestMapping(value = "/findOrderByUserId/{id}")
public R findOrderByUserId(@PathVariable("id") Integer id) {
 //ribbon实现
 String url = "http://mall-order/order/findOrderByUserId/"+id;
 R result = restTemplate.getForObject(url,R.class);

 return result;
}


public class GlobalExceptionUtil {
 /**
  * 注意： static修饰，参数类型不能出错
  * @param request org.springframework.http.HttpRequest
  * @param body
  * @param execution
  * @param ex
  * @return
  */
```

```
21  public static SentinelClientHttpResponse handleException(HttpRequest re
quest,
22  byte[] body, ClientHttpRequestExecution execution, BlockException ex) {
23  R r = R.error(-1, "===被限流啦===");
24  try {
25  return new SentinelClientHttpResponse(new ObjectMapper().writeValueAsSt
ring(r));
26  } catch (JsonProcessingException e) {
27  e.printStackTrace();
28  }
29  return null;
30  }
31
32  public static SentinelClientHttpResponse fallback(HttpRequest request,
33  byte[] body, ClientHttpRequestExecution execution, BlockException ex) {
34  R r = R.error(-2, "===被异常降级啦===");
35  try {
36  return new SentinelClientHttpResponse(new ObjectMapper().writeValueAsSt
ring(r));
37  } catch (JsonProcessingException e) {
38  e.printStackTrace();
39  }
40  return null;
41  }
42 }
```

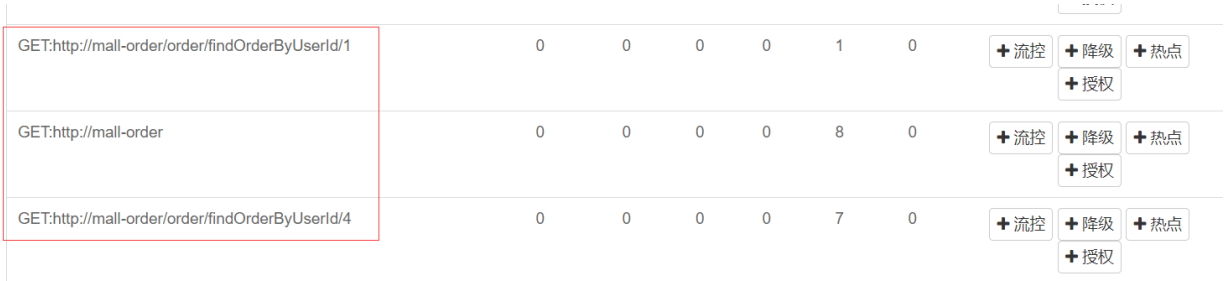
添加yml配置

```
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:
```

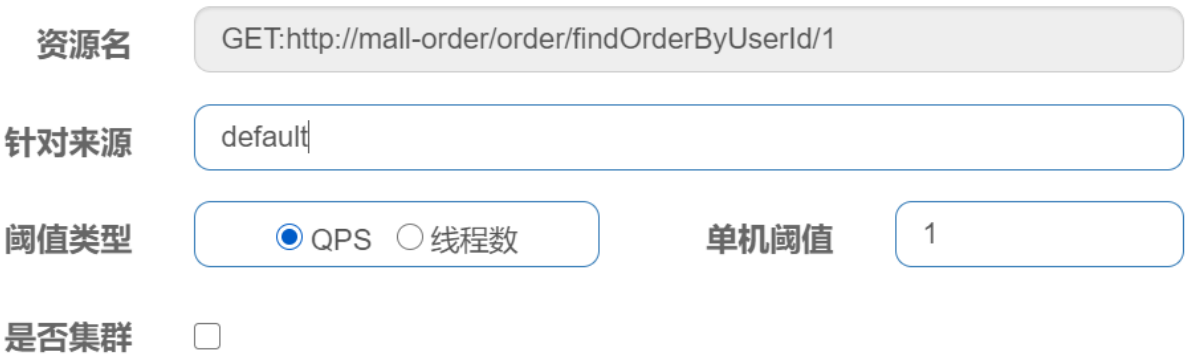
```yaml
  # 添加sentinel的控制台地址
  dashboard: 127.0.0.1:8080
  # 指定应用与Sentinel控制台交互的端口，应用本地会起一个该端口占用的HttpServer
  port: 8719

#暴露actuator端点 http://localhost:8800/actuator/sentinel
management:
  endpoints:
  web:
  exposure:
  include: '*'

#true开启sentinel对resttemplate的支持，false则关闭 默认true
resttemplate:
  sentinel:
  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



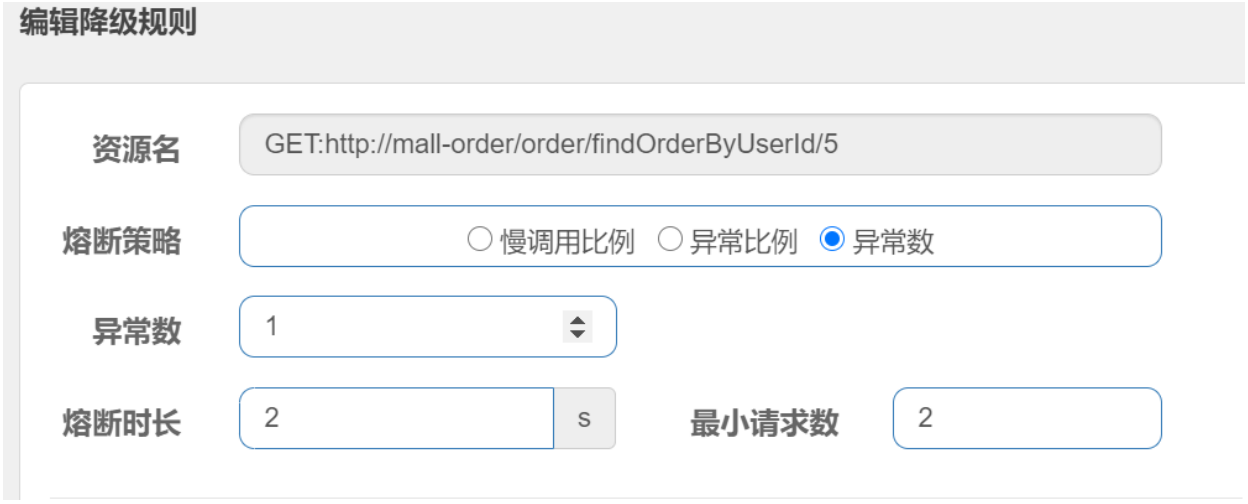
测试限流



测试降级

修改服务提供者mall-order

```
@RequestMapping("/findOrderByUserId/{userId}")
public R findOrderByUserId(@PathVariable("userId") Integer userId) {

 //模拟异常
 if(userId==5){
 throw new IllegalArgumentException("非法参数异常");
 }

 log.info("根据userId:"+userId+"查询订单信息");
  List<OrderEntity> orderEntities = orderService.listByUserId(userId);
  return R.ok().put("orders", orderEntities);
}
```



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

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

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

```

在Feign的声明式接口上添加fallback属性

```
@FeignClient(value = "mall-order",path = "/order",fallback = FallbackOrderFeignService .class)
public interface OrderFeignService {

 @RequestMapping("/findOrderByUserId/{userId}")
 public R findOrderByUserId(@PathVariable("userId") Integer userId);

}

@Component //必须交给spring 管理
public class FallbackOrderFeignService implements OrderFeignService {
 @Override
 public R findOrderByUserId(Integer userId) {
 return R.error(-1,"=======服务降级了========");
 }
}
```

添加fallbackFactory属性

```
@Component
public class FallbackOrderFeignServiceFactory implements FallbackFactory<OrderFeignService> {
 @Override
 public OrderFeignService create(Throwable throwable) {

 return new OrderFeignService() {
 @Override
 public R findOrderByUserId(Integer userId) {
 return R.error(-1,"=======服务降级了========");
```

```
10    }
11    };
12    }
13  }
```

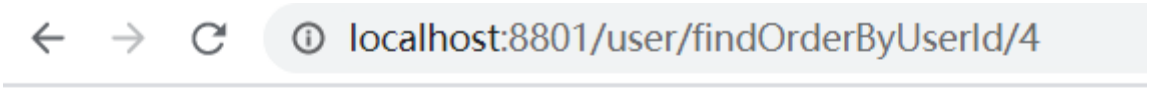
UserController

```
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

```
<dependency>
 <groupId>com.alibaba.cloud</groupId>
 <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
</dependency>
<!--Sentinel 对 Dubbo的适配 Apache Dubbo 2.7.x 及以上版本-->
<dependency>
 <groupId>com.alibaba.csp</groupId>
 <artifactId>sentinel-apache-dubbo-adapter</artifactId>
</dependency>
```

2.接入sentinel dashboard，yml中增加配置

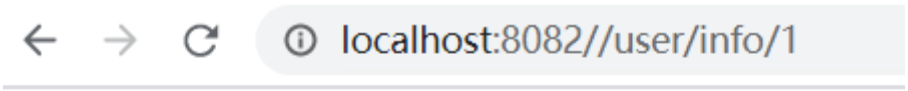
```
spring:
 cloud:
 sentinel:
 transport:
 # 添加sentinel的控制台地址
 dashboard: 127.0.0.1:8080

#暴露actuator端点
management:
  endpoints:
  web:
  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
```

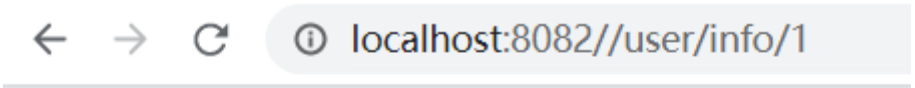
测试： http://localhost:8082/user/info/1

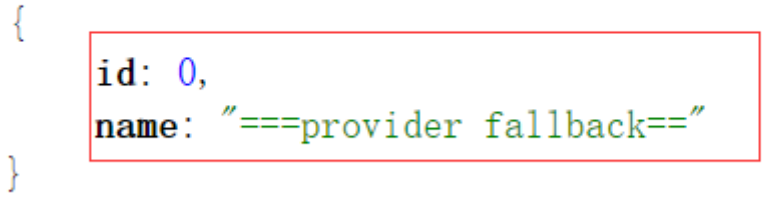



```
{
    id: 0,
    name: "===fallback=="
}
```

4.provider端配置流控规则测试


```java
@RequestMapping("/getById/{id}")
@SentinelResource("getById")
public User getById(@PathVariable("id") Integer id) {
 User user = null;
 try {
 user = userMapper.getById(id);
 } catch (Exception e) {
 e.printStackTrace();
 }
  return user;
 }

@PostConstruct
public void init() {
  DubboAdapterGlobalConfig.setProviderFallback(
  (invoker, invocation, ex) -> AsyncRpcResult.newDefaultAsyncResult(new User(0,"===provider fallback=="), invocation));
 }
```

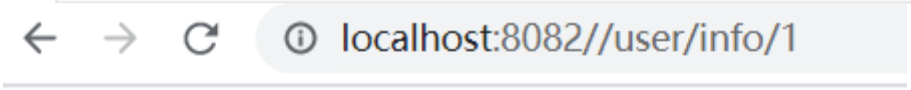




5. consumer中配置mock实现，关闭provider服务，测试mock降级

```java
@DubboReference(mock = "com.tuling.mall.user.mock.UserServiceDubboMock")
private UserService userService;

public class UserServiceDubboMock implements UserService {
 @Override
 public List<User> list() {
 return null;
 }
```

```
9
10  @Override
11  public User getById(Integer id) {
12  return new User(0,"====mock===");
13  }
14 }
```




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