一:Ribbon整合Sentinel

第一步:加配置

```
<!--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>

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

第二步：加注解(在我们的RestTemplate组件上添加@SentinelRestTemplate注解)

并且我们可以通过在@SentinelRestTemplate 同样的可以指定我们的blockHandlerClass　fallbackClass　blockHandler　fallback 这四个属性

```
@Bean
@LoadBalanced
@SentinelRestTemplate(
 blockHandler = "handleException",blockHandlerClass = GlobalExceptionHandler.class,
 fallback = "fallback",fallbackClass = GlobalExceptionHandler.class

)
public RestTemplate restTemplate() {
 return new RestTemplate();
 }

```

```
13
14
15 ****************全局异常处理类
16 public class GlobalExceptionHandler {
17
18
19  /**
20  * 限流后处理方法
21  * @param request
22  * @param body
23  * @param execution
24  * @param ex
25  * @return
26  */
27  public static SentinelClientHttpResponse handleException(HttpRequest re
quest,
28  byte[] body, ClientHttpRequestExecution execution, BlockException ex) {
29
30  ProductInfo productInfo = new ProductInfo();
31  productInfo.setProductName("被限制流量拉");
32  productInfo.setProductNo("-1");
33  ObjectMapper objectMapper = new ObjectMapper();
34
35  try {
36  return new SentinelClientHttpResponse(objectMapper.writeValueAsString(p
roductInfo));
37  } catch (JsonProcessingException e) {
38  e.printStackTrace();
39  return null;
40  }
41  }
42
43  /**
44  * 熔断后处理的方法
45  * @param request
46  * @param body
47  * @param execution
48  * @param ex
49  * @return
50  */
51  public static SentinelClientHttpResponse fallback(HttpRequest request,
```

```
52   byte[] body, ClientHttpRequestExecution execution, BlockException ex) {
53   ProductInfo productInfo = new ProductInfo();
54   productInfo.setProductName("被降级拉");
55   productInfo.setProductNo("-1");
56   ObjectMapper objectMapper = new ObjectMapper();
57
58   try {
59   return new SentinelClientHttpResponse(objectMapper.writeValueAsString(p
roductInfo));
60   } catch (JsonProcessingException e) {
61   e.printStackTrace();
62   return null;
63   }
64   }
65 }
```

第三步:添加配置

什么时候关闭:一般在我们的自己测试业务功能是否正常的情况，关闭该配置

```
1  #是否开启@SentinelRestTemplate注解
2  resttemplate:
3   sentinel:
4   enabled: true
```

二:OpenFeign整合我们的Sentinel

第一步加配置: 在tulingvip05-ms-alibaba-feignwithsentinel-order上pom.xml中添加配置

```
1  <!--加入sentinel-->
2  <dependency>
3   <groupId>com.alibaba.cloud</groupId>
4   <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
5  </dependency>
6
7
8  <!--加入actuator-->
9  <dependency>
10   <groupId>org.springframework.boot</groupId>
11   <artifactId>spring-boot-starter-actuator</artifactId>
12  </dependency>
13
14  <dependency>
```

```
<groupId>com.tuling</groupId>
<artifactId>tulingvip03-ms-alibaba-feign-api</artifactId>
<version>0.0.1-SNAPSHOT</version>
</dependency>
```

第二步:在我们的Feign的声明式接口上添加fallback属性或者 fallbackFactory属性

1）为我们添加fallback属性的api

```
@FeignClient(name = "product-center",fallback = ProductCenterFeignApiWithSentinelFallback.class)
public interface ProductCenterFeignApiWithSentinel {

 /**
 * 声明式接口,远程调用http://product-center/selectProductInfoById/{productNo}
 * @param productNo
 * @return
 */
 @RequestMapping("/selectProductInfoById/{productNo}")
  ProductInfo selectProductInfoById(@PathVariable("productNo") String productNo);

}
```

我们feign的限流降级接口(通过fallback没有办法获取到异常的)

```
@Component
public class ProductCenterFeignApiWithSentinelFallback implements ProductCenterFeignApiWithSentinel {
 @Override
 public ProductInfo selectProductInfoById(String productNo) {
 ProductInfo productInfo = new ProductInfo();
 productInfo.setProductName("默认商品");
 return productInfo;
 }
}
```

2)为我们添加fallbackFactory属性的api

```
@FeignClient(name = "product-center",fallbackFactory = ProductCenterFeignApiWithSentielFallbackFactory.class)
public interface ProductCenterFeignApiWithSentinel {

 /**
```

```
5   * 声明式接口,远程调用http://product-center/selectProductInfoById/{productN
o}
6   * @param productNo
7   * @return
8   */
9   @RequestMapping("/selectProductInfoById/{productNo}")
10   ProductInfo selectProductInfoById(@PathVariable("productNo") String pro
ductNo);
11
12  }
```

通过FallbackFactory属性可以处理我们的异常

```
1  @Component
2  @Slf4j
3  public class ProductCenterFeignApiWithSentielFallbackFactory implements F
allbackFactory<ProductCenterFeignApiWithSentinel> {
4   @Override
5   public ProductCenterFeignApiWithSentinel create(Throwable throwable) {
6   return new ProductCenterFeignApiWithSentinel(){
7
8   @Override
9   public ProductInfo selectProductInfoById(String productNo) {
10   log.error("原因:{}",throwable);
11   ProductInfo productInfo = new ProductInfo();
12   productInfo.setProductName("默认商品");
13   return productInfo;
14   }
15   };
16   }
17  }
```

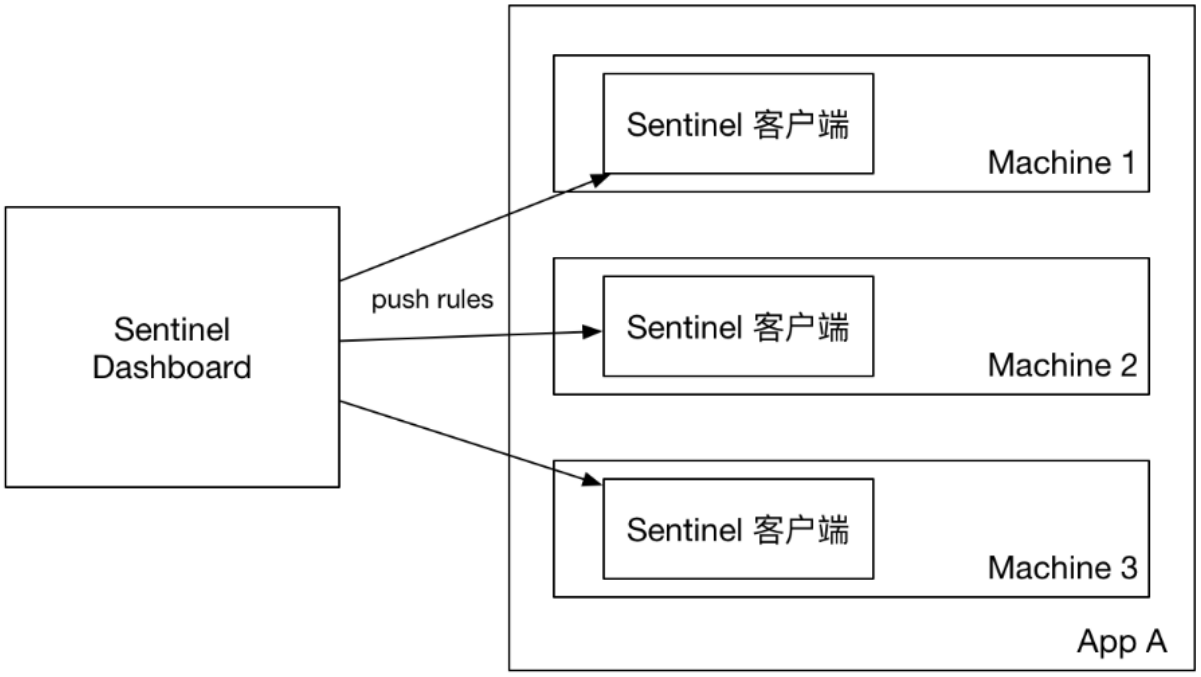
四:Sentinel 规则持久化

我们经过第四节课知道我们的Sentinel-dashboard配置的规则，在我们的微服务以及控制台重启的时候就清空了，因为他是基于内存的.

推送模式	说明	优点	缺点
原始模式	API 将规则推送至客户端并直接更新到内存中，扩展写数据源（`WritableDataSource`）	简单，无任何依赖	不保证一致性；规则保存在内存中，重启即消失。严重不建议用于生产环境
Pull 模式	扩展写数据源（`WritableDataSource`），客户端主动向某个规则管理中心定期轮询拉取规则，这个规则中心可以是 RDBMS、文件 等	简单，无任何依赖；规则持久化	不保证一致性；实时性不保证，拉取过于频繁也可能会有性能问题。
Push 模式	扩展读数据源（`ReadableDataSource`），规则中心统一推送，客户端通过注册监听器的方式时刻监听变化，比如使用 Nacos、Zookeeper 等配置中心。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 push 模式的数据源。**	规则持久化；一致性；快速	引入第三方依赖

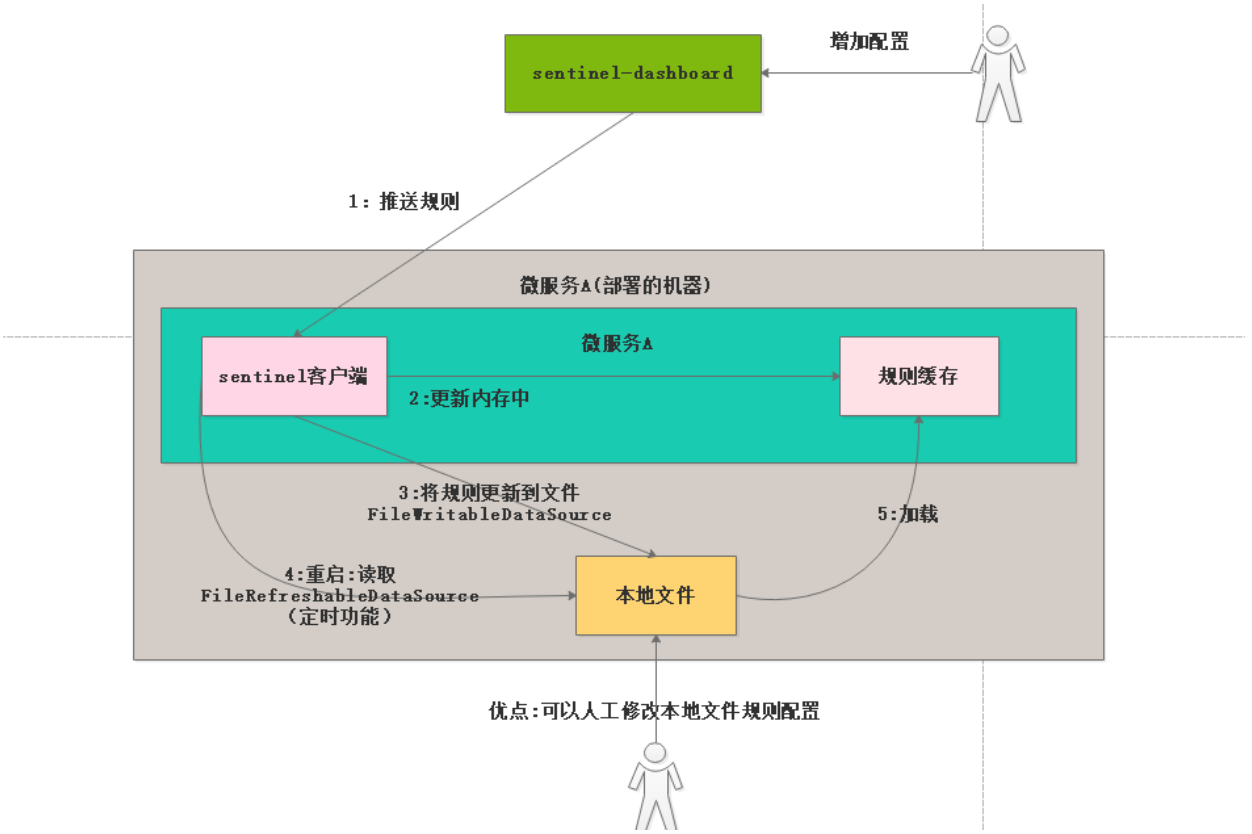
1)原生模式

Dashboard的推送规则方式是通过 API 将规则推送至客户端并直接更新到内存



优缺点:这种做法的好处是简单，无依赖；坏处是应用重启规则就会消失，仅用于简单测试，不能用于生产环境

2)Pull拉模式

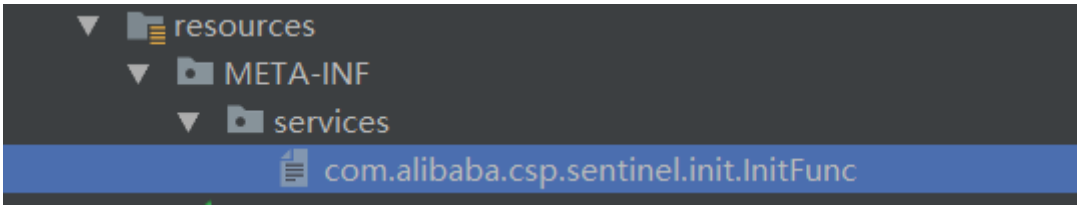


首先 Sentinel 控制台通过 API 将规则推送至客户端并更新到内存中，接着注册的写数据源会将新的规则保存到本地的文件中。使用 pull 模式的数据源时一般不需要对 Sentinel 控制台进行改造。

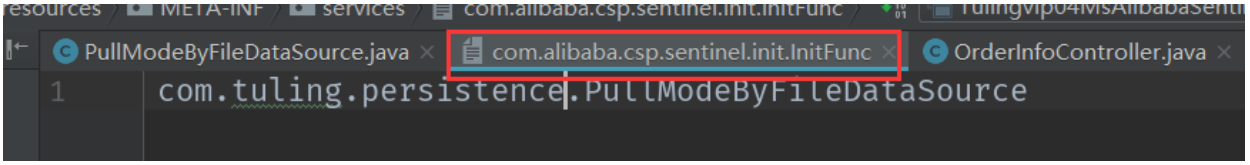
这种实现方法好处是简单，不引入新的依赖，坏处是无法保证监控数据的一致性

客户端Sentinel的改造(拉模式)

通过SPI扩展机制进行扩展,我们写一个拉模式的实现类com.tuling.persistence.PullModeByFileDataSource,然后在工厂目录下创建META-INF/services/com.alibaba.csp.sentinel.init.InitFunc文件

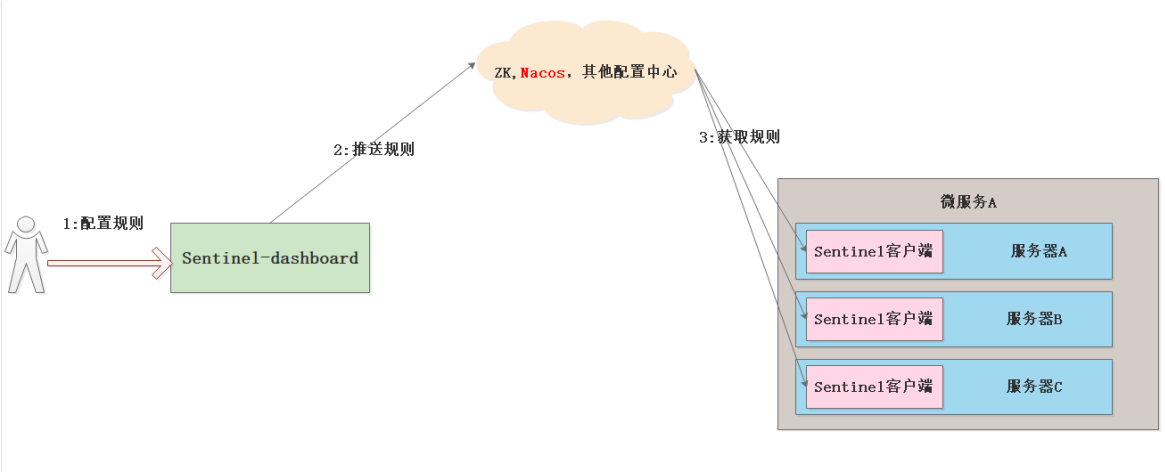


文件的内容就是写我们的拉模式的实现类:



代码在tulingvip05-ms-alibaba-sentinelrulepersistence-order工程的persistence包下

3)推模式:push(已Nacos为例) 生产推荐使用



3.1)、原理简述

- 控制台推送规则：
  - 将规则推送到Nacos或其他远程配置中心
  - Sentinel客户端链接Nacos，获取规则配置；并监听Nacos配置变化，如发生变化，就更新本地缓存（从而让本地缓存总是和Nacos一致）
- 控制台监听Nacos配置变化，如发生变化就更新本地缓存（从而让控制台本地缓存总是和Nacos一致）

3.2)改造方案

微服务改造方案:

①：tulingvip05-ms-alibaba-sentinelrulepersistencepush-order 加入依赖

```
<dependency>
 <groupId>com.alibaba.csp</groupId>
 <artifactId>sentinel-datasource-nacos</artifactId>
</dependency>
```

②: 加入yml的配置

```
spring:
 cloud:
 sentinel:
 transport:
 dashboard: localhost:9999
```

```
datasource:
# 名称随意
flow:
nacos:
 server-addr: localhost:8848
 dataId: ${spring.application.name}-flow-rules
 groupId: SENTINEL_GROUP
 rule-type: flow
 degrade:
 nacos:
 server-addr: localhost:8848
 dataId: ${spring.application.name}-degrade-rules
 groupId: SENTINEL_GROUP
 rule-type: degrade
 system:
 nacos:
 server-addr: localhost:8848
 dataId: ${spring.application.name}-system-rules
 groupId: SENTINEL_GROUP
 rule-type: system
 authority:
 nacos:
 server-addr: localhost:8848
 dataId: ${spring.application.name}-authority-rules
 groupId: SENTINEL_GROUP
 rule-type: authority
 param-flow:
 nacos:
 server-addr: localhost:8848
 dataId: ${spring.application.name}-param-flow-rules
 groupId: SENTINEL_GROUP
 rule-type: param-flow
```

Sentinel-dashboard改造方案:

改造的原理

```
//需要把test注释掉
<dependency>
 <groupId>com.alibaba.csp</groupId>
 <artifactId>sentinel-datasource-nacos</artifactId>
 <!-- <scope>test</scope>-->
```

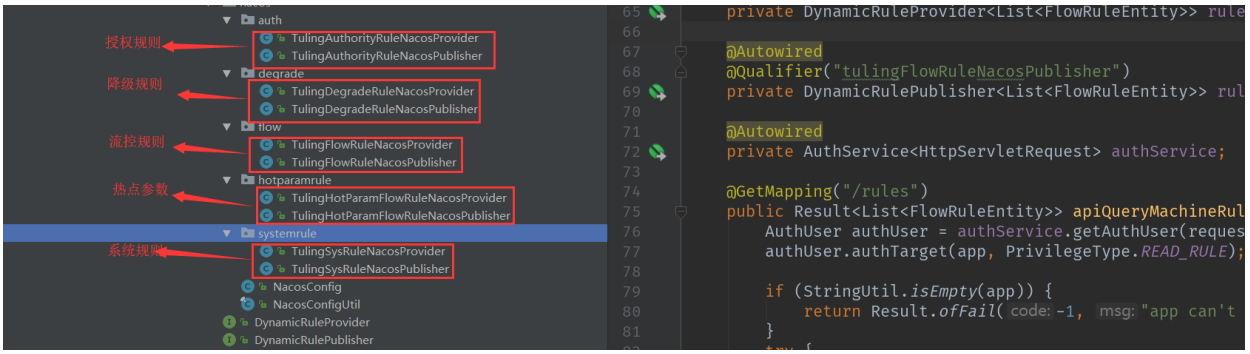
```
</dependency>
```

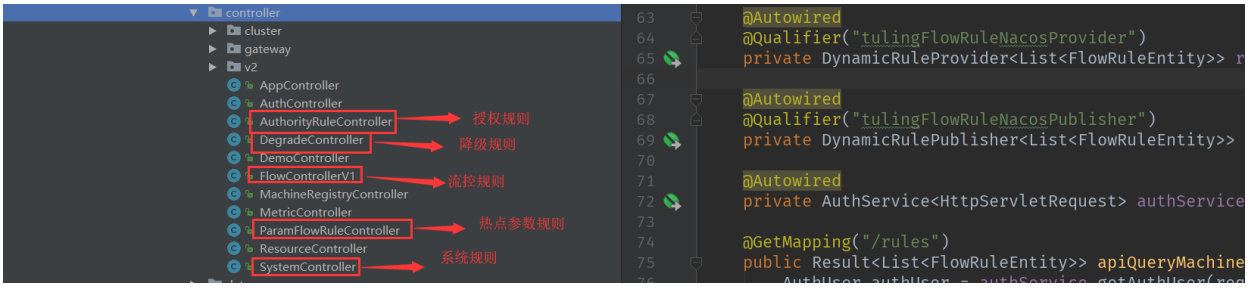
控制台改造主要是为规则实现

- **DynamicRuleProvider：从Nacos上读取配置**
- **DynamicRulePublisher：将规则推送到Nacos上**

在sentinel-dashboard工程目录com.alibaba.csp.sentinel.dashboard.rule 下创建一个Nacos的包

然后把我们的各个场景的配置规则类写道该包下.





我们以ParamFlowRuleController（热点参数流控类作为修改作为演示）

```
/**
 * @author Eric Zhao
 * @since 0.2.1
 */
@RestController
@RequestMapping(value = "/paramFlow")
public class ParamFlowRuleController {

 private final Logger logger = LoggerFactory.getLogger(ParamFlowRuleContr
oller.class);

 @Autowired
 private SentinelApiClient sentinelApiClient;
 @Autowired
 private AppManagement appManagement;
 @Autowired
```

```
16  private RuleRepository<ParamFlowRuleEntity, Long> repository;
17
18  //添加我们自己写的publisher
19  @Autowired
20  @Qualifier("tulingHotParamFlowRuleNacosPublisher")
21  private DynamicRulePublisher<List<ParamFlowRuleEntity>> rulePublisher;
22
23  //添加我们自己写的ruleProvider
24  @Autowired
25  @Qualifier("tulingHotParamFlowRuleNacosProvider")
26  private DynamicRuleProvider<List<ParamFlowRuleEntity>> ruleProvider;
27
28  @Autowired
29  private AuthService<HttpServletRequest> authService;
30
31  。。。。。。。。。。。。。。。
32
33  /**
34  * 查询所有的热点参数规则
35  **/
36  @GetMapping("/rules")
37  public Result<List<ParamFlowRuleEntity>> apiQueryAllRulesForMachine(Htt
pServletRequest request,
38  @RequestParam String app,
39  @RequestParam String ip,
40  @RequestParam Integer port) {
41  AuthUser authUser = authService.getAuthUser(request);
42  authUser.authTarget(app, PrivilegeType.READ_RULE);
43  if (StringUtil.isEmpty(app)) {
44  return Result.ofFail(-1, "app cannot be null or empty");
45  }
46  if (StringUtil.isEmpty(ip)) {
47  return Result.ofFail(-1, "ip cannot be null or empty");
48  }
49  if (port == null || port <= 0) {
50  return Result.ofFail(-1, "Invalid parameter: port");
51  }
52  if (!checkIfSupported(app, ip, port)) {
53  return unsupportedVersion();
54  }
55  try {
```

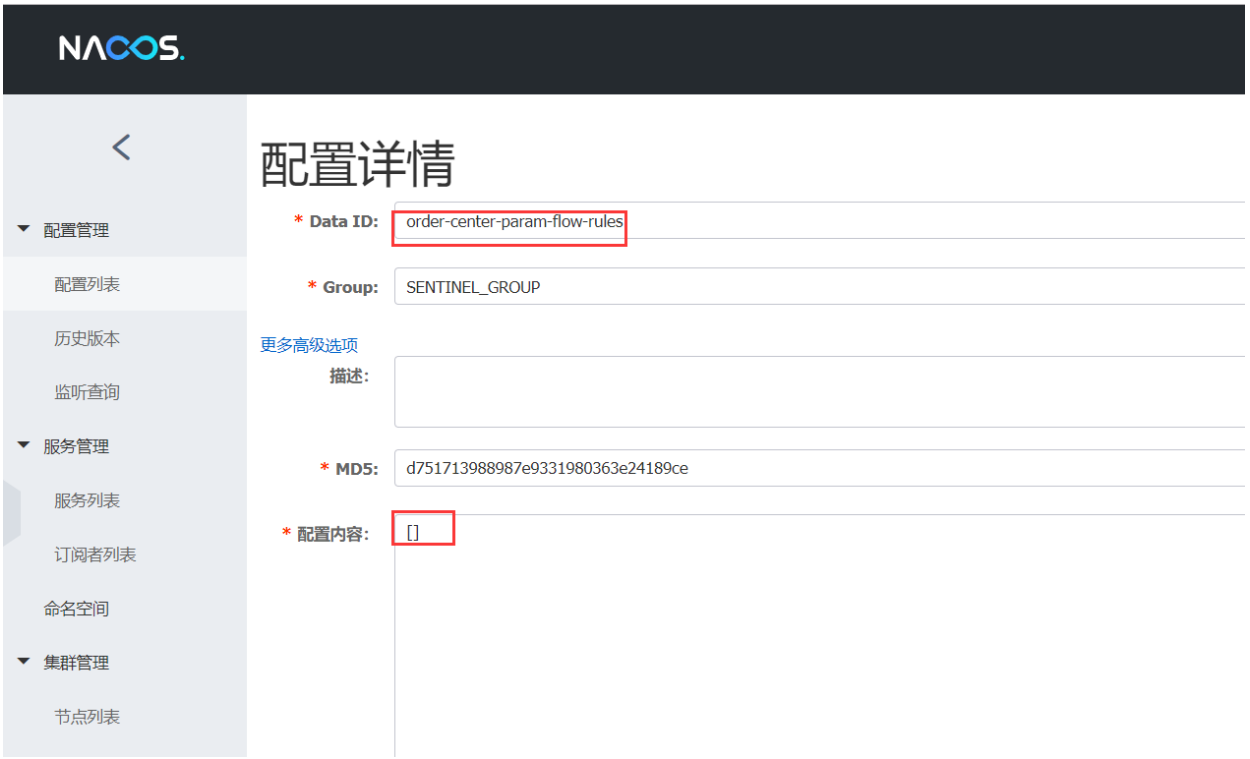
```
56
57  .......
58  //去nacos上获取配置规则
59  List<ParamFlowRuleEntity> rules = ruleProvider.getRules(app);
60  rules = repository.saveAll(rules);
61  return Result.ofSuccess(rules);
62  } catch (ExecutionException ex) {
63  logger.error("Error when querying parameter flow rules",
ex.getCause());
64  if (isNotSupported(ex.getCause())) {
65  return unsupportedVersion();
66  } else {
67  return Result.ofThrowable(-1, ex.getCause());
68  }
69  } catch (Throwable throwable) {
70  logger.error("Error when querying parameter flow rules", throwable);
71  return Result.ofFail(-1, throwable.getMessage());
72  }
73  }
74
75
76  @PostMapping("/rule")
77  public Result<ParamFlowRuleEntity> apiAddParamFlowRule(HttpServletReque
st request,
78  @RequestBody ParamFlowRuleEntity entity) {
79  AuthUser authUser = authService.getAuthUser(request);
80  authUser.authTarget(entity.getApp(), PrivilegeType.WRITE_RULE);
81  Result<ParamFlowRuleEntity> checkResult = checkEntityInternal(entity);
82  if (checkResult != null) {
83  return checkResult;
84  }
85  if (!checkIfSupported(entity.getApp(), entity.getIp(),
entity.getPort())) {
86  return unsupportedVersion();
87  }
88  entity.setId(null);
89  entity.getRule().setResource(entity.getResource().trim());
90  Date date = new Date();
91  entity.setGmtCreate(date);
92  entity.setGmtModified(date);
93  try {
```

```
94   entity = repository.save(entity);
95   //发布规则到nacos配置中心上
96   publishRules(entity.getApp());
97   return Result.ofSuccess(entity);
98   } catch (ExecutionException ex) {
99   logger.error("Error when adding new parameter flow rules",
ex.getCause());
100   if (isNotSupported(ex.getCause())) {
101   return unsupportedVersion();
102   } else {
103   return Result.ofThrowable(-1, ex.getCause());
104   }
105   } catch (Throwable throwable) {
106   logger.error("Error when adding new parameter flow rules", throwable);
107   return Result.ofFail(-1, throwable.getMessage());
108   }
109   }
110
111
112   @PutMapping("/rule/{id}")
113   public Result<ParamFlowRuleEntity> apiUpdateParamFlowRule(HttpServletRe
quest request,
114   @PathVariable("id") Long id,
115   @RequestBody ParamFlowRuleEntity entity) {
116   AuthUser authUser = authService.getAuthUser(request);
117   if (id == null || id <= 0) {
118   return Result.ofFail(-1, "Invalid id");
119   }
120   ParamFlowRuleEntity oldEntity = repository.findById(id);
121   if (oldEntity == null) {
122   return Result.ofFail(-1, "id " + id + " does not exist");
123   }
124   authUser.authTarget(oldEntity.getApp(), PrivilegeType.WRITE_RULE);
125   Result<ParamFlowRuleEntity> checkResult = checkEntityInternal(entity);
126   if (checkResult != null) {
127   return checkResult;
128   }
129   if (!checkIfSupported(entity.getApp(), entity.getIp(),
entity.getPort())) {
130   return unsupportedVersion();
131   }
```

```
132   entity.setId(id);
133   Date date = new Date();
134   entity.setGmtCreate(oldEntity.getGmtCreate());
135   entity.setGmtModified(date);
136   try {
137   entity = repository.save(entity);
138   //更新nacos配置规则
139   publishRules(entity.getApp());
140   return Result.ofSuccess(entity);
141   } catch (ExecutionException ex) {
142   logger.error("Error when updating parameter flow rules, id=" + id, ex.g
etCause());
143   if (isNotSupported(ex.getCause())) {
144   return unsupportedVersion();
145   } else {
146   return Result.ofThrowable(-1, ex.getCause());
147   }
148   } catch (Throwable throwable) {
149   logger.error("Error when updating parameter flow rules, id=" + id, thro
wable);
150   return Result.ofFail(-1, throwable.getMessage());
151   }
152   }
153
154   @DeleteMapping("/rule/{id}")
155   public Result<Long> apiDeleteRule(HttpServletRequest request, @PathVari
able("id") Long id) {
156   AuthUser authUser = authService.getAuthUser(request);
157   if (id == null) {
158   return Result.ofFail(-1, "id cannot be null");
159   }
160   ParamFlowRuleEntity oldEntity = repository.findById(id);
161   if (oldEntity == null) {
162   return Result.ofSuccess(null);
163   }
164   authUser.authTarget(oldEntity.getApp(), PrivilegeType.DELETE_RULE);
165   try {
166   repository.delete(id);
167   //删除nacos上的配置规则
168   publishRules(oldEntity.getApp());
169   return Result.ofSuccess(id);
```

```
170   } catch (ExecutionException ex) {
171   logger.error("Error when deleting parameter flow rules",
ex.getCause());
172   if (isNotSupported(ex.getCause())) {
173   return unsupportedVersion();
174   } else {
175   return Result.ofThrowable(-1, ex.getCause());
176   }
177   } catch (Throwable throwable) {
178   logger.error("Error when deleting parameter flow rules", throwable);
179   return Result.ofFail(-1, throwable.getMessage());
180   }
181   }
182
183
184   private void publishRules(String app) throws Exception {
185   List<ParamFlowRuleEntity> rules = repository.findAllByApp(app);
186   rulePublisher.publish(app, rules);
187   }
188 }
```

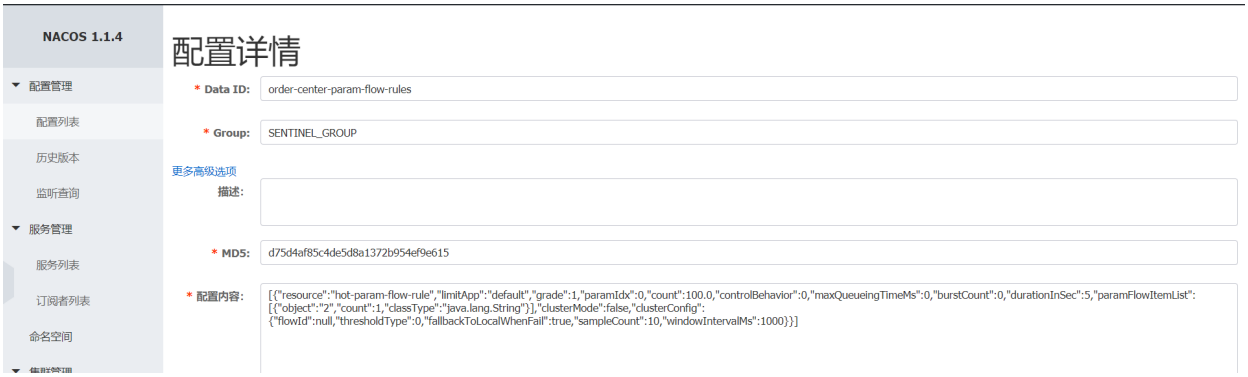
没有添加规则时，nacos的配置中心是如图



添加热点参数规则:(第一个入参的qps为100 统计时间为5s) 而针对的参数值为String类型的

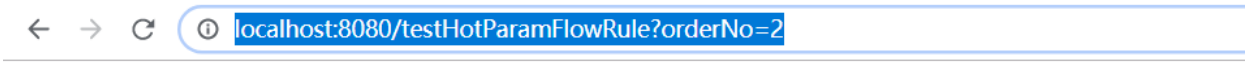
值为2的 流控为1

添加之后:



测试:http://localhost:8080/testHotParamFlowRule?orderNo=2 疯狂的刷新接口

出现了流控了.



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Dec 03 14:22:26 CST 2019
There was an unexpected error (type=Internal Server Error, status=500).
No message available

4)生产实践，必须要实现规则的持久化,你可以使用拉模式或者推模式,但是不管哪种模式都需要你去写代码?但是不想写代码。。。。。。？？怎么办?????

阿里云的 AHAS

开通地址:https://ahas.console.aliyun.com/

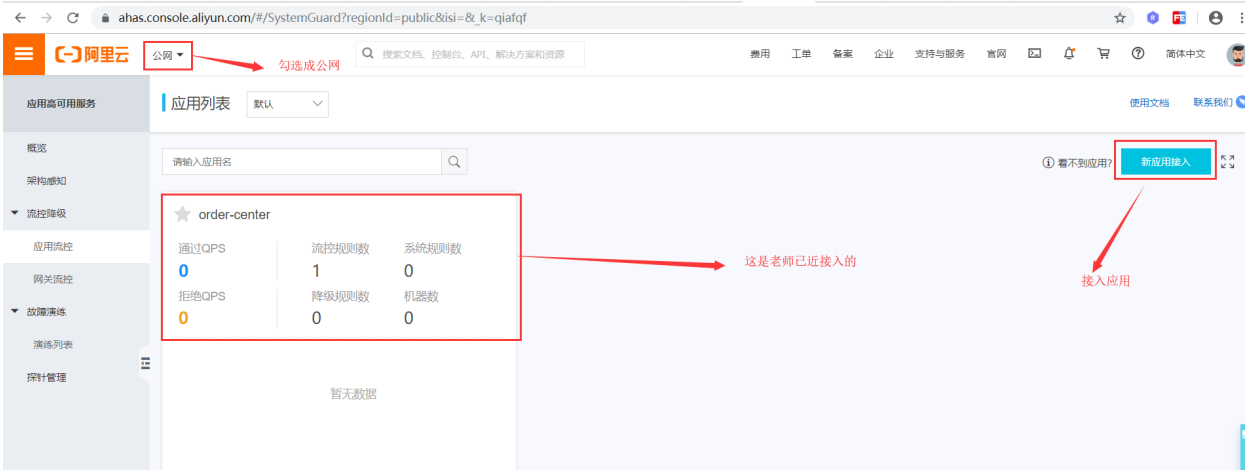
开通规则说明:https://help.aliyun.com/document_detail/90323.html

第一步:访问 https://help.aliyun.com/document_detail/90323.html

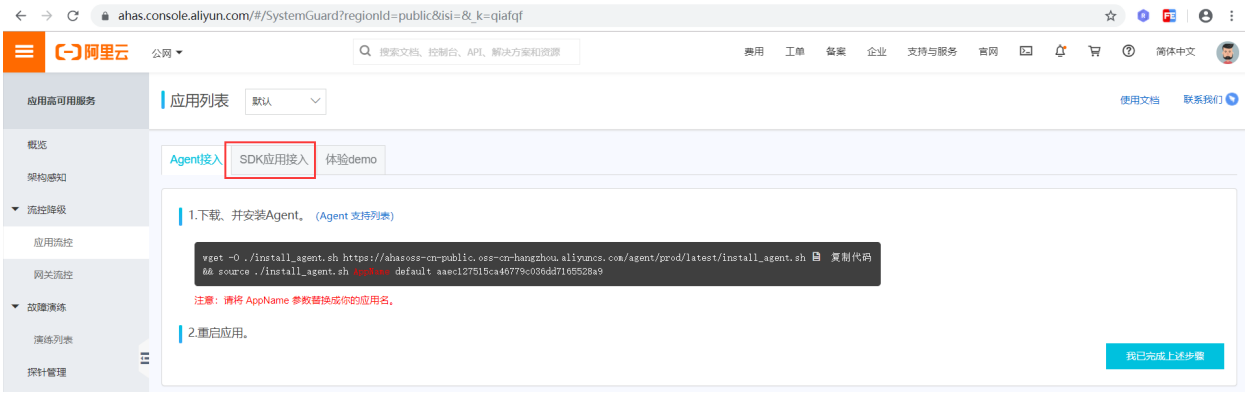
第二步: 免费开通

第三步:开通

第四步:接入应用



第五步:点击接入SDK



第六步:加入我们的应用

以我们的tulingvip05-ms-alibaba-sentinelrulepersistence-ahas-order为例演示

加入ahas的依赖:

```
<dependency>
 <groupId>com.alibaba.csp</groupId>
 <artifactId>spring-boot-starter-ahas-sentinel-client</artifactId>
 <version>1.5.0</version>
</dependency>
```

加入配置: yml的配置

```
ahas.namespace: default
project,name: order-center
ahas.license: aaec127515ca46779c036dd7165528a9
```

测试接口:

```
@SentinelResource("hot-param-flow-rule")
@RequestMapping("/testHotParamFlowRule")
```

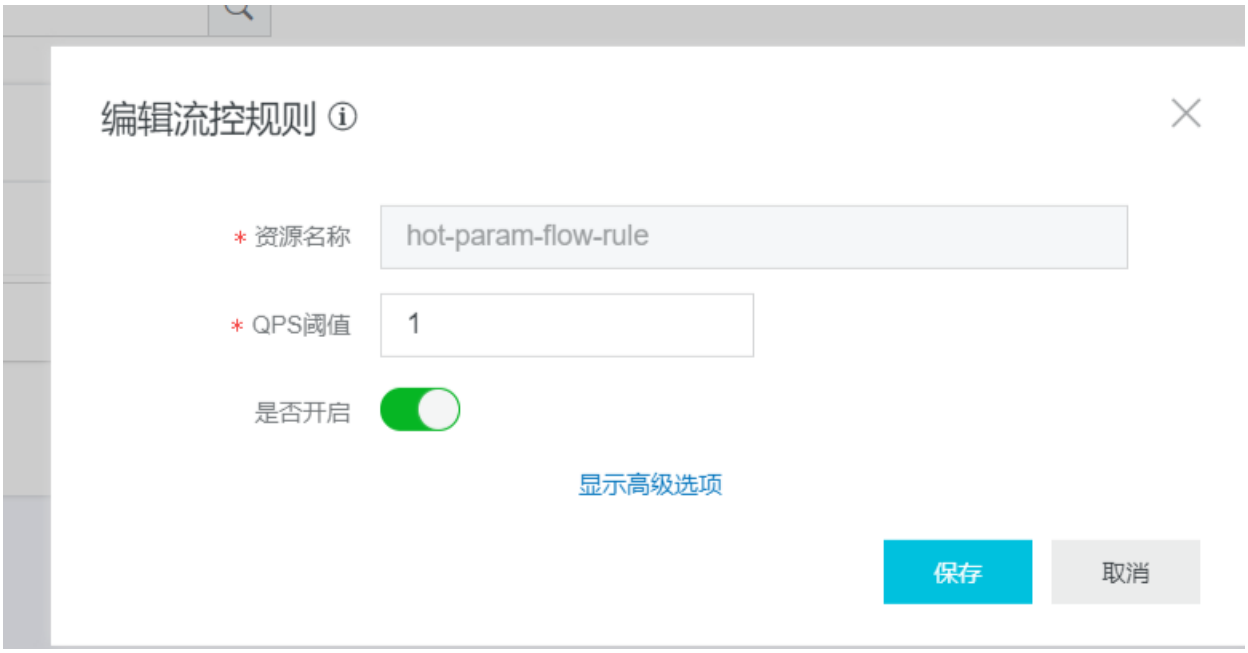
```
3  public OrderInfo testHotParamFlowRule(@RequestParam("orderNo") String ord
erNo) {
4    return orderInfoMapper.selectOrderInfoById(orderNo);
5  }
```

第一次访问接口:

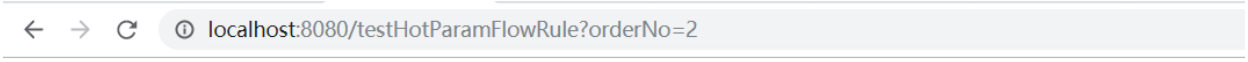

```
{
    "orderNo":  "2",
    "userName":  "smlz",
    "createDt":  "2019-11-26T16:00:00.000+0000",
    "productNo":  "2",
    "productCount":  2
}
```


AHas控制台出现我们的微服务

添加我们自己的流控规则



疯狂刷新我们的测试接口:



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

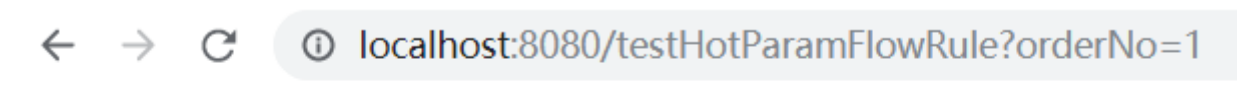
Tue Dec 03 16:01:59 CST 2019
There was an unexpected error (type=Internal Server Error, status=500).
No message available

5)优化错误页面

流控错误页面

Blocked by Sentinel (flow limiting)

降级错误页面



Blocked by Sentinel (flow limiting)

发现这二种错误都是一样的，显然这里我们需要优化**UrlBlockHandler** 提供了一个接口，我们需要

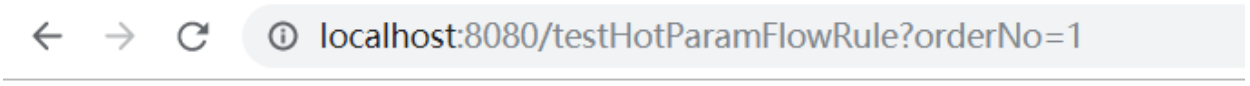
实现这个接口

```
/**
* @vlog: 高于生活，源于生活
* @desc: 类的描述:处理流控,降级规则
* @author: smlz
* @createDate: 2019/12/3 16:40
* @version: 1.0
*/
@Component
public class TulingUrlBlockHandler implements UrlBlockHandler {

  public static final Logger log = LoggerFactory.getLogger(TulingUrlBlockHandler.class);

  @Override
  public void blocked(HttpServletRequest request, HttpServletResponse response, BlockException ex) throws IOException {

  if(ex instanceof FlowException) {
  log.warn("触发了流控");
  warrperResponse(response,ErrorEnum.FLOW_RULE_ERR);
  }else if(ex instanceof ParamFlowException) {
  log.warn("触发了参数流控");
  warrperResponse(response,ErrorEnum.HOT_PARAM_FLOW_RULE_ERR);
  }else if(ex instanceof AuthorityException) {
  log.warn("触发了授权规则");
  warrperResponse(response,ErrorEnum.AUTH_RULE_ERR);
  }else if(ex instanceof SystemBlockException) {
  log.warn("触发了系统规则");
  warrperResponse(response,ErrorEnum.SYS_RULE_ERR);
  }else{
  log.warn("触发了降级规则");
  warrperResponse(response,ErrorEnum.DEGRADE_RULE_ERR);
  }
  }


  private void warrperResponse(HttpServletResponse httpServletResponse, ErrorEnum errorEnum) throws IOException {
```

```
36   httpServletResponse.setStatus(500);
37   httpServletResponse.setCharacterEncoding("UTF-8");
38   httpServletResponse.setHeader("Content-Type","application/json;charset=
utf-8");
39   httpServletResponse.setContentType("application/json;charset=utf-8");
40
41   ObjectMapper objectMapper = new ObjectMapper();
42   String errMsg =objectMapper.writeValueAsString(new ErrorResult(errorEnu
m));
43   httpServletResponse.getWriter().write(errMsg);
44   }
45 }
```

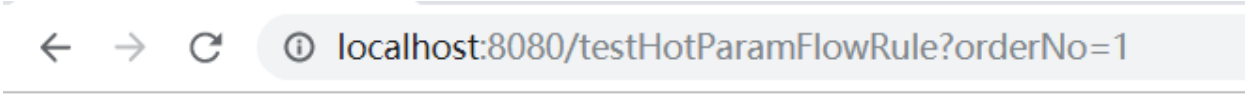
优化后:

流控规则提示



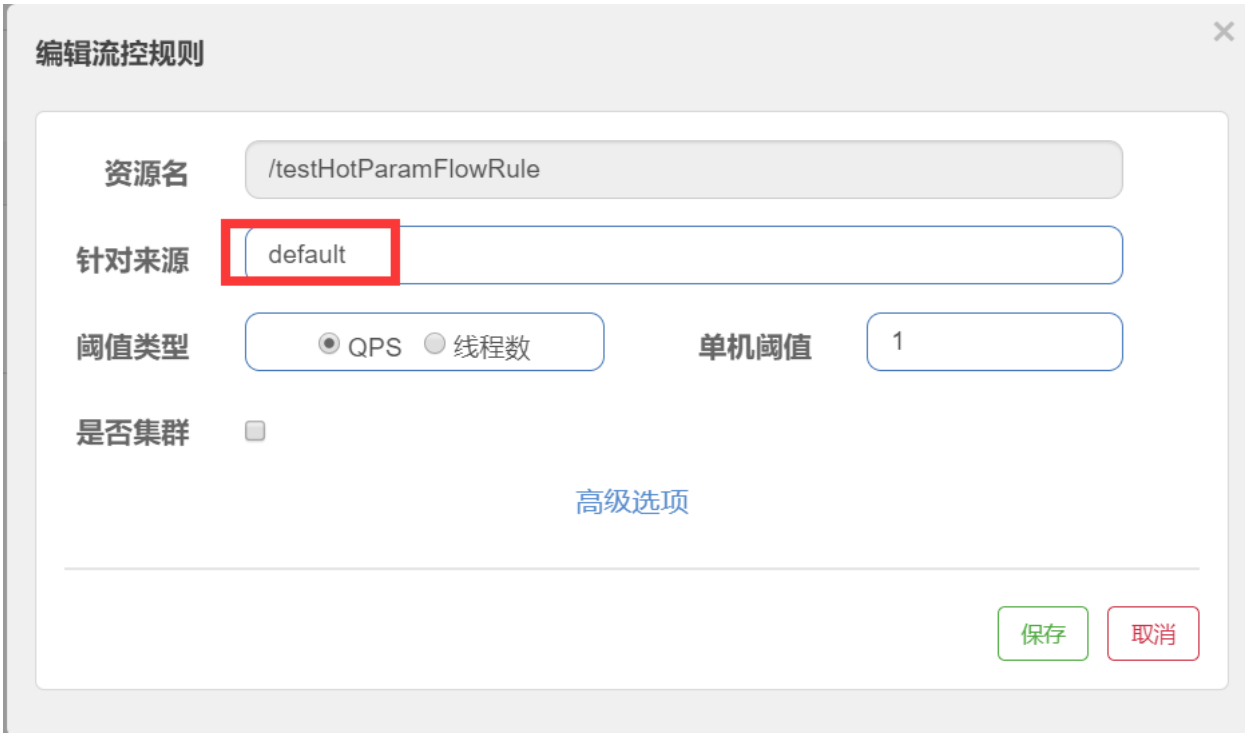

```
{
    "code":  -1,
    "msg":  "触发流控"
}
```


降级规则提示:




```
{
    "code":  -5,
    "msg":  "降级规则触发"
}
```


6)针对来源编码实现

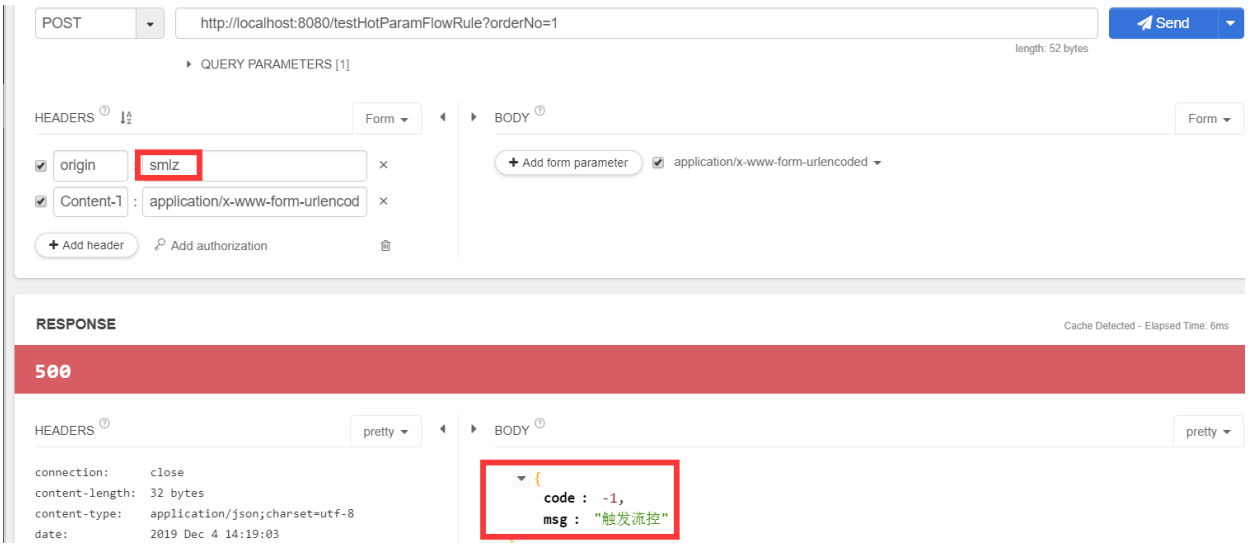


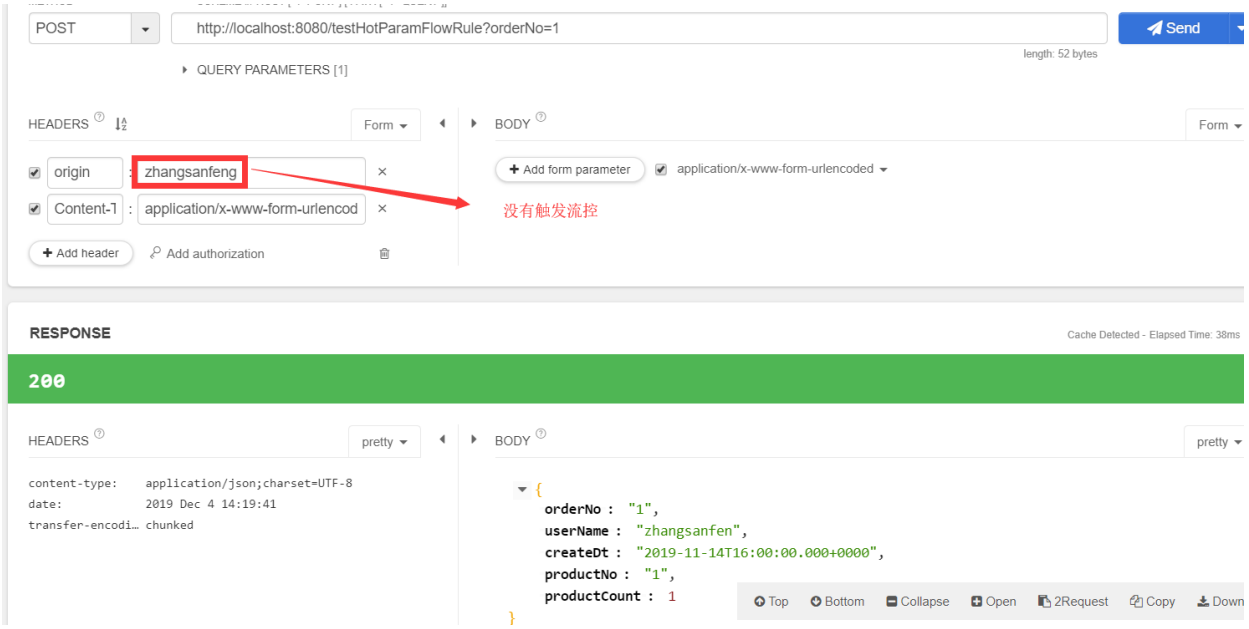
Sentinel提供了一个RequestOriginParser接口，我们可以在这里实现编码从请求头中区分来源

```
/**
* @vlog: 高于生活，源于生活
* @desc: 类的描述:区分来源接口
* @author: smlz
* @createDate: 2019/12/4 13:13
* @version: 1.0
*/
@Component
@Slf4j
public class TulingRequestOriginParse implements RequestOriginParser {

 @Override
 public String parseOrigin(HttpServletRequest request) {
 String origin = request.getHeader("origin");
 if(StringUtils.isEmpty(origin)) {
 log.warn("origin must not null");
 throw new IllegalArgumentException("request origin must not null");
 }
 return origin;
 }
}
```

配置设置区分来源: 为smlz


编辑流控规则
资源名
/testHotParamFlowRule
针对来源
smlz
阈值类型
QPS
线程数
单机阈值
1
是否集群
高级选项
保存
取消


POST
http://localhost:8080/testHotParamFlowRule?orderNo=1
Send
length: 52 bytes
QUERY PARAMETERS [1]
HEADERS
Form
origin
smlz
Content-T
application/x-www-form-urlencod
Add header
Add authorization
BODY
Form
Add form parameter
application/x-www-form-urlencoded
RESPONSE
Cache Detected - Elapsed Time: 6ms
500
HEADERS
pretty
connection: close
content-length: 32 bytes
content-type: application/json;charset=utf-8
date: 2019 Dec 4 14:19:03
BODY
pretty
{
code : -1,
msg : "触发流控"


POST
http://localhost:8080/testHotParamFlowRule?orderNo=1
Send
length: 52 bytes
QUERY PARAMETERS [1]
HEADERS
Form
origin
zhangsanfeng
Content-T
application/x-www-form-urlencod
Add header
Add authorization
BODY
Form
Add form parameter
application/x-www-form-urlencoded
没有触发流控
RESPONSE
Cache Detected - Elapsed Time: 38ms
200
HEADERS
pretty
content-type: application/json;charset=UTF-8
date: 2019 Dec 4 14:19:41
transfer-encodi… chunked
BODY
pretty
{
orderNo : "1",
userName : "zhangsanfen",
createDt : "2019-11-14T16:00:00.000+0000",
productNo : "1",
productCount : 1
}
Top
Bottom
Collapse
Open
2Request
Copy
Down