Spring Cloud Sentinel 服务哨兵

历史修订

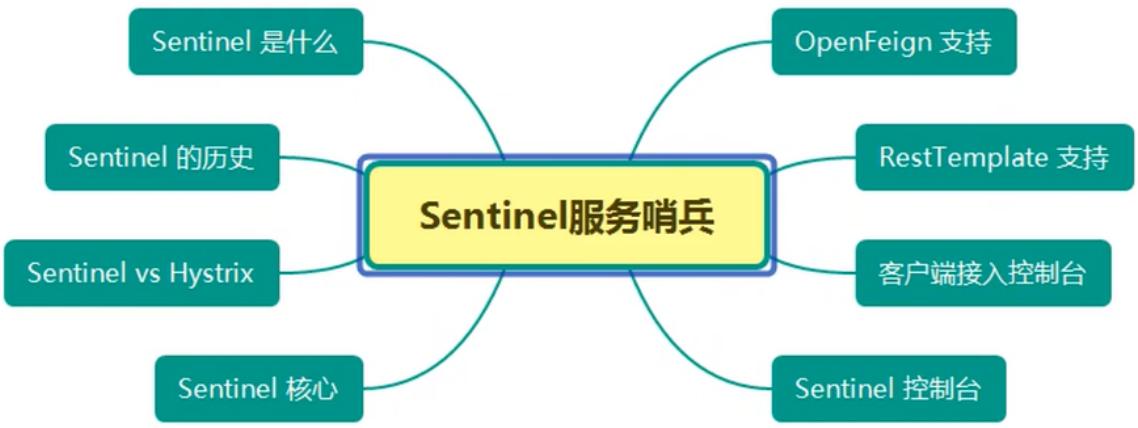
本次修订日期: 2023-03-02	**下次修订日期:**

修订编号	修订日期	变更描述	说明
V0.1	2023-03-02	起草	李宗在
V0.2	2023-03-03	测试验证	李宗在
V0.3			

1. 技术介绍

- Spring Boot
- Spring Cloud
- Sentinel
- Feign
- Mybatis/Mybatis-Plus
- MySQL
- Docker
- Ubuntu
- Redis
- Postman
- swagger

2. 学习目标



3. 什么是Sentinel

由于Netflix 中多项开源产品已进入维护阶段，不再开发新的版本，就目前来看是没有什么问题的。但是从长远角度出发，我们还是需要考虑是否有可替代产品使用。比如本文中要介绍的 Alibaba Sentinel 就是一款高性能且轻量级的流量控制、熔断降级可替换方案。

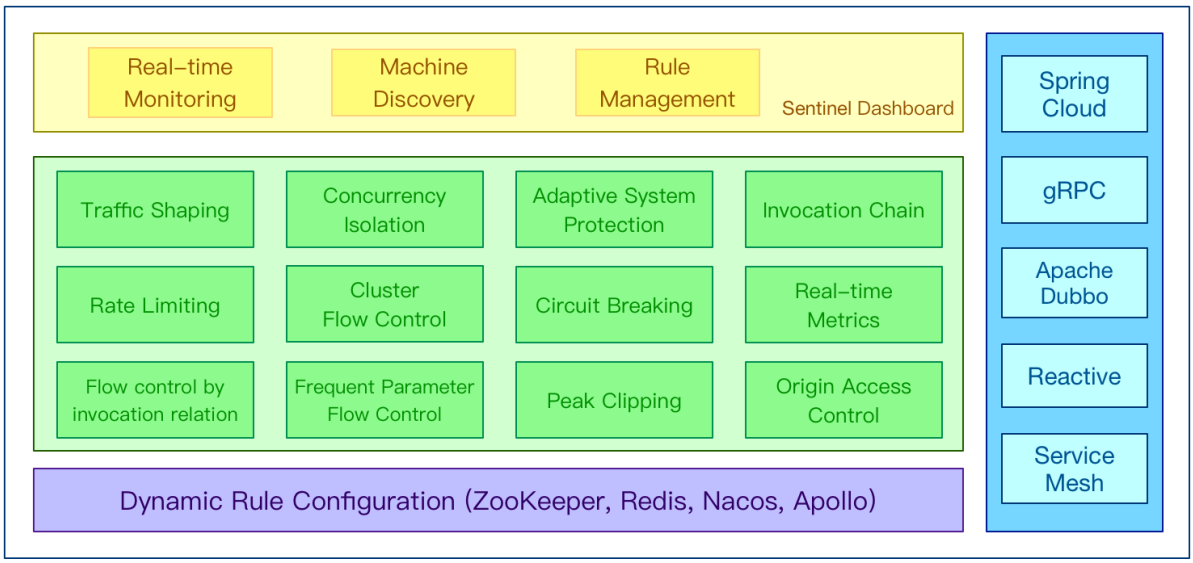
随着微服务的流行，服务和服务之间的稳定性变得越来越重要。Sentinel 以流量为切入点，从流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性。

Sentinel 具有以下特征:

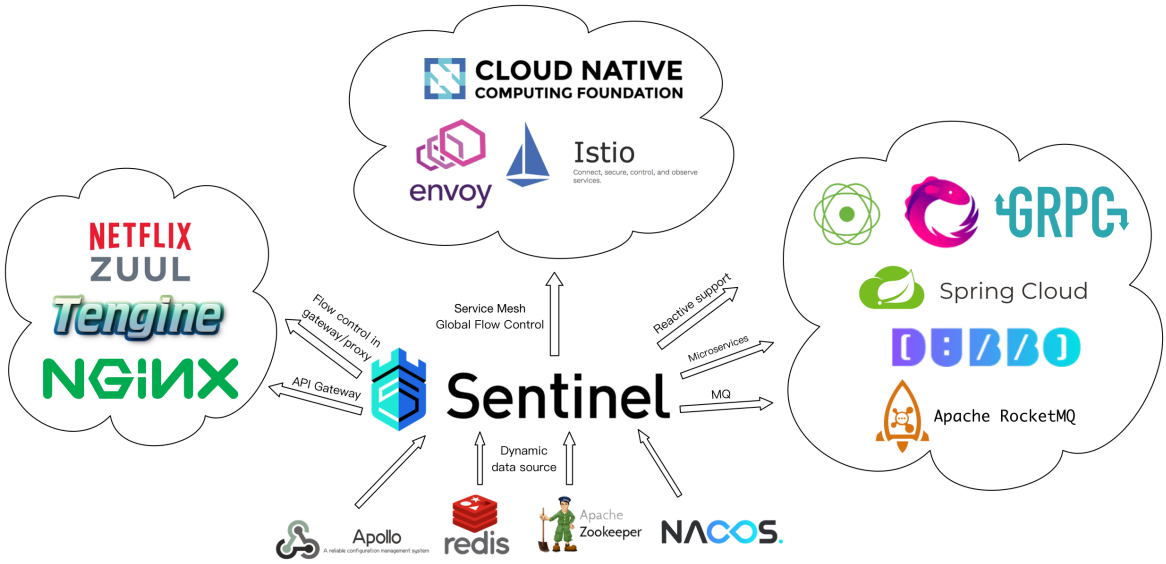
- **丰富的应用场景**：Sentinel 承接了阿里巴巴近 10 年的双十一大促流量的核心场景，例如秒杀（即突发流量控制在系统容量可以承受的范围）、消息削峰填谷、集群流量控制、实时熔断下游不可用应用等。

- **完备的实时监控**：Sentinel 同时提供实时的监控功能。您可以在控制台中看到接入应用的单台机器秒级数据，甚至 500 台以下规模的集群的汇总运行情况。
- **广泛的开源生态**：Sentinel 提供开箱即用的与其它开源框架/库的整合模块，例如与 Spring Cloud、Dubbo、gRPC 的整合。您只需要引入相应的依赖并进行简单的配置即可快速地接入 Sentinel。
- **完善的 SPI 扩展点**：Sentinel 提供简单易用、完善的 SPI 扩展接口。您可以通过实现扩展接口来快速地定制逻辑。例如定制规则管理、适配动态数据源等。

Sentinel 主要特征



Sentinel 开源生态



Sentinel 目前已经针对 Servlet、Dubbo、Spring Boot/Spring Cloud、gRPC 等进行了适配，用户只需引入相应依赖并进行简单配置即可非常方便地享受 Sentinel 的高可用流量防护能力。Sentinel 还为 Service Mesh 提供了集群流量防护的能力。未来 Sentinel 还会对更多常用框架进行适配。

Sentinel 分为两个部分:

- 核心库（Java 客户端）不依赖任何框架/库，能够运行于所有 Java 运行时环境，同时对 Dubbo / Spring Cloud 等框架也有较好的支持。
- 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。

4. Sentinel发展历史

- 2012 年，Sentinel 诞生，主要功能为入口流量控制。
- 2013-2017 年，Sentinel 在阿里巴巴集团内部迅速发展，成为基础技术模块，覆盖了所有的核心场景。Sentinel 也因此积累了大量的流量归整场景以及生产实践。
- 2018 年，Sentinel 开源，并持续演进。

5. Sentinel vs Hystrix

5.1 Hystrix

官网：**https://github.com/Netflix/Hystrix**

> Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.

Hystrix 的关注点在于隔离和熔断为主的容错机制，超时或被熔断的调用将会快速失败，并可以提供 fallback 机制。

5.2 Sentinel

官网：**https://github.com/alibaba/Sentinel**

Sentinel 的关注点在于:

- 多样化的流量控制
- 熔断降级
- 系统负载保护
- 实时监控和控制台

Hystrix 迁移 Sentinel 方案

Sentinel 提供了从 Hystrix 迁移到 Sentinel 的方案，官网： https://github.com/alibaba/Sentinel/wiki/Guideline:-从-Hystrix-迁移到-Sentinel

5.3 总结

	Sentinel	Hystrix
隔离策略	信号量隔离（并发线程数限流）	线程池隔离/信号量隔离
熔断降级策略	基于响应时间、异常比率、异常数	基于异常比率
实时指标实现	滑动窗口（LeapArray）	滑动窗口（基于 RxJava）
规则配置	支持多种数据源	支持多种数据源
扩展性	多个扩展点	插件的形式
基于注解的支持	支持	支持
调用链路信息	支持同步调用	不支持
限流	基于 QPS / 并发数，支持基于调用关系的限流	有限支持
流量整形	支持慢启动、匀速器模式	不支持
系统负载保护	支持	不支持
控制台	开箱即用，可配置规则、查看秒级监控、机器发现等	较为简单
常见框架的适配	Servlet、Spring Cloud、Dubbo、gRPC 等	Servlet、Spring Cloud Netflix

6. Sentinel核心

Sentinel 的使用可以分为两个部分：

- 核心库（Java 客户端）：不依赖任何框架/库，能够运行于 Java 7 及以上的版本的运行时环境，同时对 Dubbo / Spring Cloud 等框架也有较好的支持（见 **主流框架适配** ）。
- 控制台（Dashboard）：控制台主要负责管理推送规则、监控、集群限流分配管理、机器发现等。

7. Sentinel控制台

Sentinel 提供一个轻量级的开源控制台，它提供机器发现以及健康情况管理、监控（单机和集群），规则管理和推送的功能。

官网文档： https://github.com/alibaba/Sentinel/wiki/控制台

7.1 获取安装包

您可以从 **release 页面** 下载最新版本的控制台 jar 包。

您也可以从最新版本的源码自行构建 Sentinel 控制台：

- 下载 **控制台** 工程
- 使用以下命令将代码打包成一个 fat jar: `mvn clean package`

7.2 启动控制台

启动命令如下，本文使用的是目前最新 1.7.1 版本：

```
java -Dserver.port=8080 -Dcsp.sentinel.dashboard.server=localhost:8080 -
Dproject.name=sentinel-dashboard -jar sentinel-dashboard-1.7.1.jar
```

> **注意**：启动 Sentinel 控制台需要 JDK 版本为 1.8 及以上版本。

其中 `-Dserver.port=8080` 用于指定 Sentinel 控制台端口为 `8080` 。

从 Sentinel 1.6.0 起，Sentinel 控制台引入基本的**登录**功能，默认用户名和密码都是 `sentinel` 。可以参考 **鉴权模块文档** 配置用户名和密码。

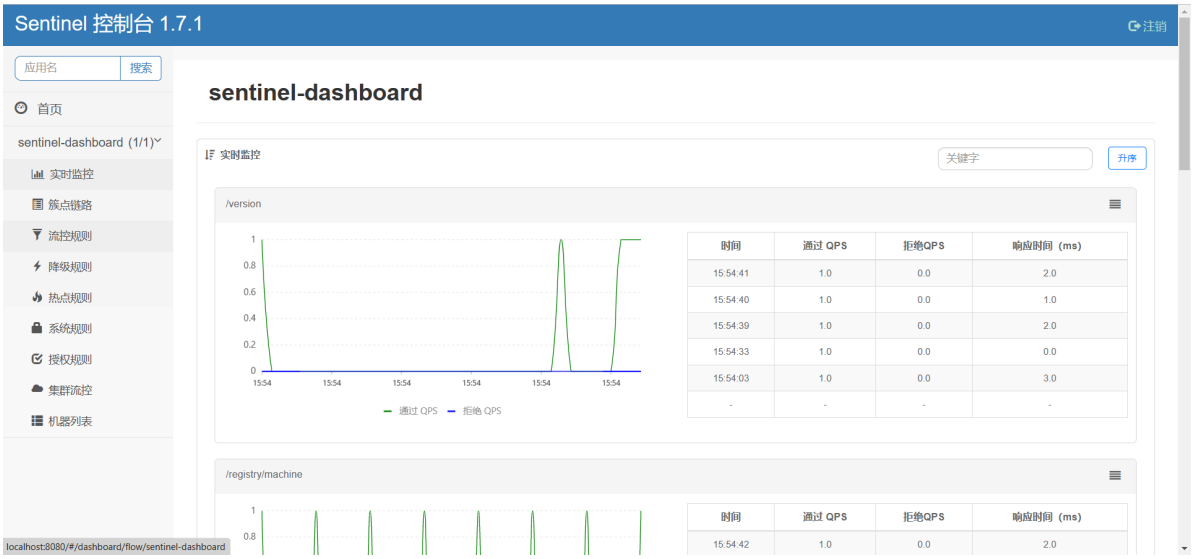
> 注：若您的应用为 Spring Boot 或 Spring Cloud 应用，您可以通过 Spring 配置文件来指定配置，详情请参考 **Spring Cloud Alibaba Sentinel 文档** 。

为了方便启动，可以编写一个启动脚本 `run.bat` ：

```
java -Dserver.port=8080 -Dcsp.sentinel.dashboard.server=localhost:8080 -
Dproject.name=sentinel-dashboard -jar sentinel-dashboard-1.7.1.jar
pause
```

7.3 访问控制台

访问： http://localhost:8080/



8. 客户端接入控制台

控制台启动后，客户端需要按照以下步骤接入到控制台：

- 添加依赖
- 定义资源
- 定义规则

先把可能需要保护的资源定义好，之后再配置规则。也可以理解为，只要有了资源，我们就可以在任何时候灵活地定义各种流量控制规则。在编码的时候，只需要考虑这个代码是否需要保护，如果需要保护，就将之定义为一个资源。

由于我们的项目是 Spring Cloud 项目所以借助官方文档来进行学习。

Spring 官网文档： https://spring-cloud-alibaba-group.github.io/github-pages/greenwich/spring-cloud-alibaba.html

Github 文档： https://github.com/alibaba/spring-cloud-alibaba/wiki/Sentinel

8.1 添加依赖

父工程需要添加如下依赖：

```
<properties>
  <!--spring-cloud-alibaba.version 版本号-->
  <spring-cloud-alibaba.version>2.1.0.RELEASE</spring-cloud-alibaba.version>
</properties>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-dependencies</artifactId>
            <version>2.1.0.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

子工程需要添加如下依赖：

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

8.1.2 配置文件

客户端需要启动 Transport 模块来与 Sentinel 控制台进行通信。

`order-service-rest` 的 application.yml

```
spring:
  cloud:
    sentinel:
      transport:
        port: 8719
        dashboard: localhost:8080
```

这里的 `spring.cloud.sentinel.transport.port` 端口配置会在应用对应的机器上启动一个 Http Server，该 Server 会与 Sentinel 控制台做交互。比如 Sentinel 控制台添加了一个限流规则，会把规则数据 push 给这个 Http Server 接收，Http Server 再将规则注册到 Sentinel 中。

8.1.2 初始化客户端

确保客户端有访问量，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。

简单的理解就是：访问一次客户端，Sentinel 即可完成客户端初始化操作，并持续向控制台发送心跳包

首先确保 Sentinel 是启动状态，然后依次启动 eureka-server，eureka-server02，product-service，order-service-rest。

8.1.3 访问控制台

多次访问： http://localhost:9090/order/1 然后查看控制台 `实时监控` 结果如下：

8.2 定义资源

资源 是 Sentinel 中的核心概念之一。我们说的资源，可以是任何东西，服务，服务里的方法，甚至是一段代码。最常用的资源是我们代码中的 Java 方法。Sentinel 提供了 `@SentinelResource` 注解用于定义资源，并提供了 AspectJ 的扩展用于自动定义资源、处理 `BlockException` 等。

官网文档： https://github.com/alibaba/Sentinel/wiki/ 如何使用#定义资源

8.2.1 注解资源

```
package com.example.service.impl;

import com.alibaba.csp.sentinel.annotation.SentinelResource;
import com.alibaba.csp.sentinel.slots.block.BlockException;
import com.example.pojo.Product;
import com.example.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

/**
 * 商品管理
 */
@Service
public class ProductServiceImpl implements ProductService {

    @Autowired
    private RestTemplate restTemplate;

    /**
     * 根据主键查询商品
     *
     * @param id
     * @return
     */
    @SentinelResource(value = "selectProductById",
            blockHandler = "selectProductByIdBlockHandler", fallback =
"selectProductByIdFallback")
    @Override
    public Product selectProductById(Integer id) {
        return restTemplate.getForObject("http://product-service/product/" + id,
Product.class);
    }

    // 服务流量控制处理，参数最后多一个 BlockException，其余与原函数一致。
    public Product selectProductByIdBlockHandler(Integer id, BlockException ex) {
        // Do some log here.
        ex.printStackTrace();
        return new Product(id, "服务流量控制处理-托底数据", 1, 2666D);
    }

    // 服务熔断降级处理，函数签名与原函数一致或加一个 Throwable 类型的参数
    public Product selectProductByIdFallback(Integer id, Throwable throwable) {
        System.out.println("product-service 服务的 selectProductById 方法出现异常，异常信息
如下："
                + throwable);
        return new Product(id, "服务熔断降级处理-托底数据", 1, 2666D);
    }

}
```

> 注意：注解方式埋点不支持 private 方法。

`@SentinelResource` 用于定义资源，并提供可选的异常处理和 fallback 配置项。
`@SentinelResource` 注解包含以下属性：

- `value`：资源名称，必需项（不能为空）
- `entryType`：entry 类型，可选项（默认为 `EntryType.OUT`）

- `blockHandler` / `blockHandlerClass`：`blockHandler` 对应处理 `BlockException` 的函数名称，可选项。blockHandler 函数访问范围需要是 `public`，返回类型需要与原方法相匹配，参数类型需要和原方法相匹配并且最后加一个额外的参数，类型为 `BlockException`。blockHandler 函数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，则可以指定 `blockHandlerClass` 为对应的类的 `Class` 对象，注意对应的函数必需为 static 函数，否则无法解析。
- `fallback`

  ：fallback 函数名称，可选项，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。fallback 函数签名和位置要求：
  - 返回值类型必须与原函数返回值类型一致；
  - 方法参数列表需要和原函数一致，或者可以额外多一个 `Throwable` 类型的参数用于接收对应的异常。
  - fallback 函数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，则可以指定 `fallbackClass` 为对应的类的 `Class` 对象，注意对应的函数必需为 static 函数，否则无法解析。
- `defaultFallback`

  （since 1.6.0）：默认的 fallback 函数名称，可选项，通常用于通用的 fallback 逻辑（即可以用于很多服务或方法）。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，则只有 fallback 会生效。defaultFallback 函数签名要求：
  - 返回值类型必须与原函数返回值类型一致；
  - 方法参数列表需要为空，或者可以额外多一个 `Throwable` 类型的参数用于接收对应的异常。
  - defaultFallback 函数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，则可以指定 `fallbackClass` 为对应的类的 `Class` 对象，注意对应的函数必需为 static 函数，否则无法解析。
- `exceptionsToIgnore`（since 1.6.0）：用于指定哪些异常被排除掉，不会计入异常统计中，也不会进入 fallback 逻辑中，而是会原样抛出。

> 注：1.6.0 之前的版本 fallback 函数只针对降级异常（`DegradeException`）进行处理，**不能针对业务异常进行处理**。

特别地，若 blockHandler 和 fallback 都进行了配置，则被限流降级而抛出 `BlockException` 时只会进入 `blockHandler` 处理逻辑。若未配置 `blockHandler`、`fallback` 和 `defaultFallback`，则被限流降级时会将 `BlockException` **直接抛出**（若方法本身未定义 throws BlockException 则会被 JVM 包装一层 `UndeclaredThrowableException`）。

从 1.4.0 版本开始，注解方式定义资源支持自动统计业务异常，无需手动调用 `Tracer.trace(ex)` 来记录业务异常。Sentinel 1.4.0 以前的版本需要自行调用 `Tracer.trace(ex)` 来记录业务异常。

8.2.2 访问控制台

8.2.3 定义规则

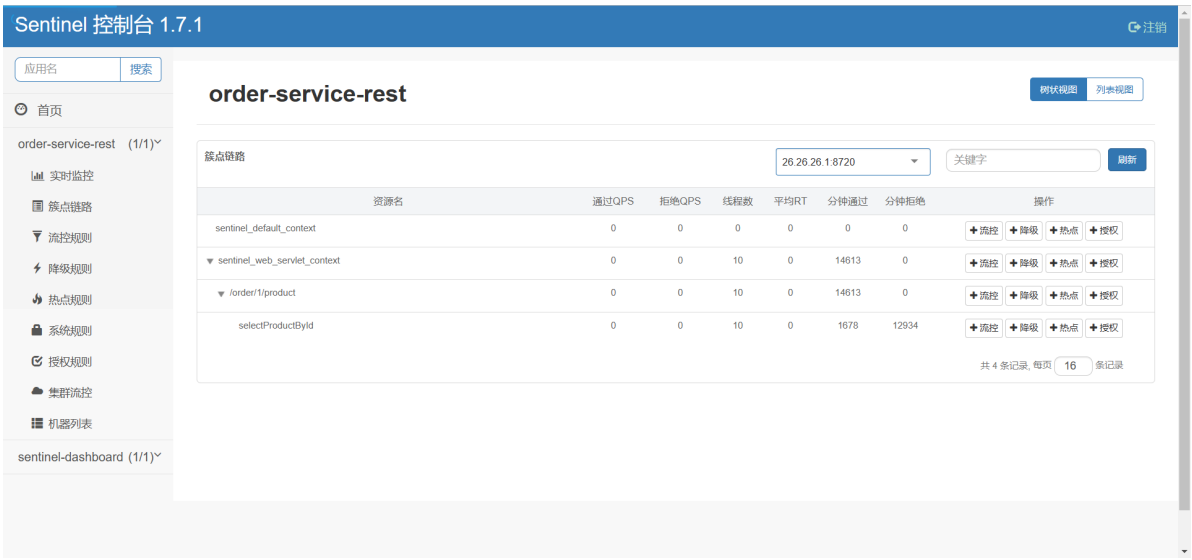
Sentinel 的所有规则都可以在**内存态中动态地查询及修改，修改之后立即生效。**同时 Sentinel 也提供相关 API，供您来定制自己的规则策略。

Sentinel 支持以下几种规则：**流量控制规则**、**熔断降级规则**、**系统保护规则**、**来源访问控制规则** 和 **热点参数规则。**

8.2.4 流量控制规则

选择 簇点链路 找到定义好的资源 `selectProductById` 并点击对应的规则按钮进行设置。

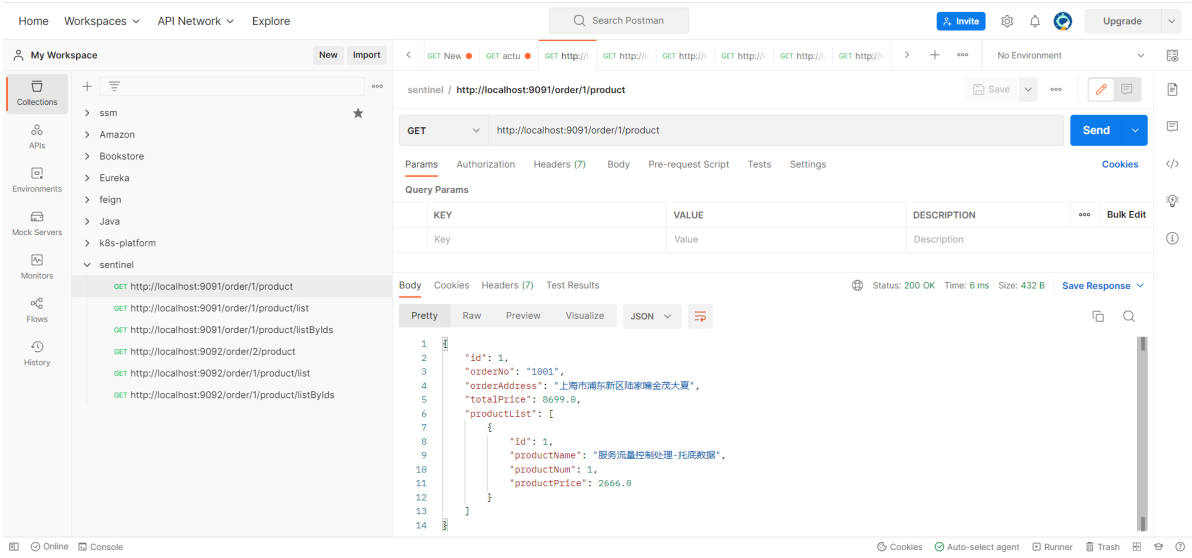
比如我们设置一个流量控制规则，定义资源访问的 QPS 为 1（每秒能处理查询数目）。



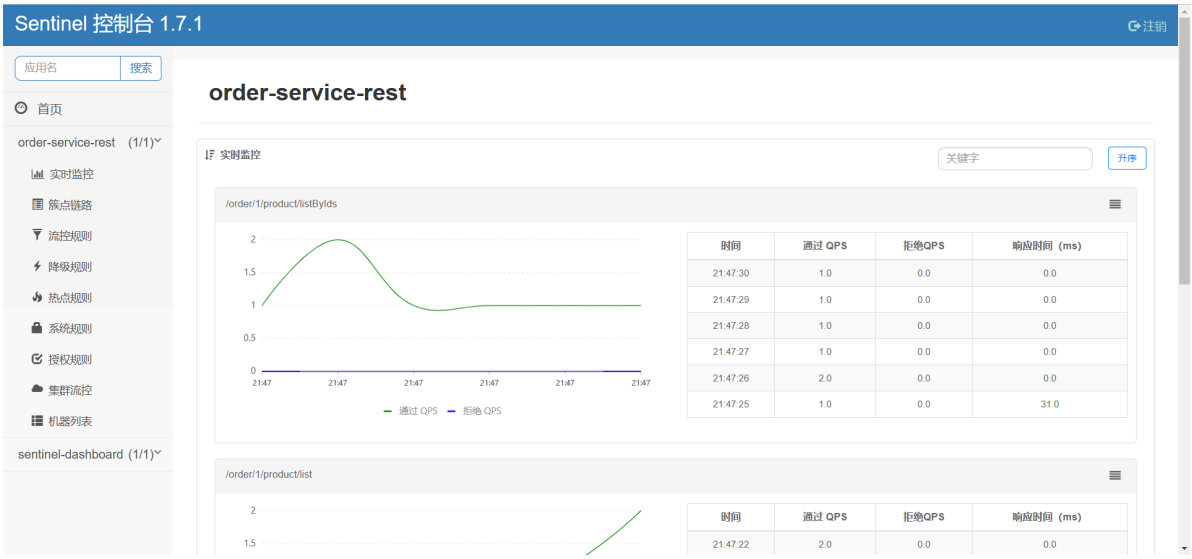
8.2.5 调用微服务接口

> ☐ 建议使用Jmeter 压力测试工具进行测试，可以观察到QPS请求成功数等

http://localhost:9091/order/1/product



8.2.6 获取流量监控结果



下图为 `Hystrix turbine` 微服务多集群度量指标和健康监控, 可以与上图阿里巴巴Sentinel 进行比较。

8.2.7 熔断降级规则

修改 `order-service-rest` 项目中的核心代码，模拟服务出错。

```
package com.example.service.impl;

import com.alibaba.csp.sentinel.annotation.SentinelResource;
import com.alibaba.csp.sentinel.slots.block.BlockException;
import com.example.pojo.Product;
import com.example.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

/**
 * 商品管理
 */
@Service
public class ProductServiceImpl implements ProductService {

    @Autowired
```

```
    private RestTemplate restTemplate;

    /**
     * 根据主键查询商品
     *
     * @param id
     * @return
     */
    @SentinelResource(value = "selectProductById",
            blockHandler = "selectProductByIdBlockHandler", fallback =
"selectProductByIdFallback")
    @Override
    public Product selectProductById(Integer id) {
        // 模拟查询主键为 1 的商品信息会导致异常
        if (1 == id)
            throw new RuntimeException("查询主键为 1 的商品信息导致异常");
        return restTemplate.getForObject("http://product-service/product/" + id,
Product.class);
    }

    // 服务流量控制处理，参数最后多一个 BlockException，其余与原函数一致。
    public Product selectProductByIdBlockHandler(Integer id, BlockException ex) {
        // Do some log here.
        ex.printStackTrace();
        return new Product(id, "服务流量控制处理-托底数据", 1, 2666D);
    }

    // 服务熔断降级处理，函数签名与原函数一致或加一个 Throwable 类型的参数
    public Product selectProductByIdFallback(Integer id, Throwable throwable) {
        System.out.println("product-service 服务的 selectProductById 方法出现异常，异常信息
如下："
                + throwable);
        return new Product(id, "服务熔断降级处理-托底数据", 1, 2666D);
    }

}
```

熔断降级规则支持相应时间、异常比例、异常数三种方式。

8.2.8 动态规则扩展

`SentinelProperties` 内部提供了 `TreeMap` 类型的 `datasource` 属性用于配置数据源信息。支持：

- 文件配置规则
- Nacos 配置规则
- ZooKeeper 配置规则
- Apollo 配置规则
- Redis 配置规则

8.2.8.1 文件配置规则

Sentinel 支持通过本地文件加载规则配置，使用方式如下（限流规则作为演示）：

```yaml
spring:
  cloud:
    sentinel:
      datasource:
        ds1:
          file:
            file: classpath:flowRule.json
            data-type: json
            rule-type: flow
```

`flowRule.json` 对应 `com.alibaba.csp.sentinel.slots.block.RuleConstant` 各属性。

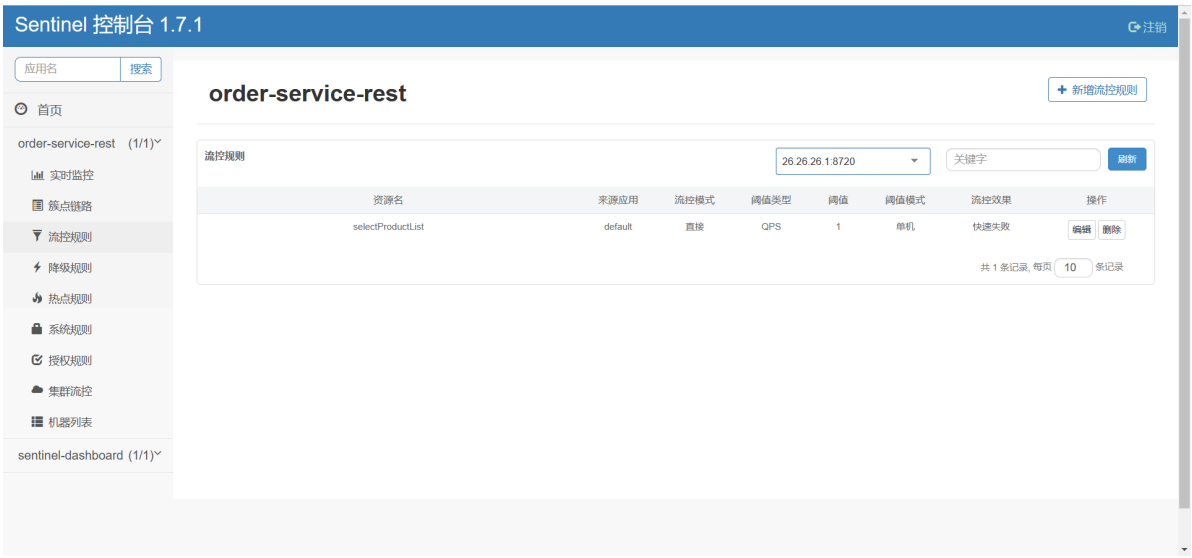
```json
[
  {
    "resource": "selectProductList",
    "count": 1,
    "grade": 1,
    "limitApp": "default",
    "strategy": 0,
    "controlBehavior": 0
  }
]
```

重要属性：

Field	说明	默认值
resource	资源名，资源名是限流规则的作用对象	
count	限流阈值	
grade	限流阈值类型，QPS 模式（1）或并发线程数模式（0）	QPS 模式
limitApp	流控针对的调用来源	`default`，代表不区分调用来源
strategy	调用关系限流策略：直接、链路、关联	根据资源本身（直接）
controlBehavior	流控效果（直接拒绝 / 排队等待 / 慢启动模式），不支持按调用关系限流	直接拒绝
clusterMode	是否集群限流	否

8.2.8.2 访问Sentinel控制台

访问客户端以后，刷新控制台，查看流控规则如下：



8.3 RestTemplate请求工具

> RestTemplate是一个同步的web http客户端请求模板工具, 是基于spring框架的底层的一个知识点

Spring Cloud Alibaba Sentinel 支持对 RestTemplate 调用的服务进行服务保护。需要在构造 RestTemplate Bean 时添加 `@SentinelRestTemplate` 注解。

8.3.1 启动类

> ☐ 添加@SentinelRestTemplate注解,
>
> ```
> @SentinelRestTemplate(blockHandler = "handleException", blockHandlerClass =
> ExceptionUtil.class, fallback = "fallback", fallbackClass =
> ExceptionUtil.class)
> ```

```
package com.example;

import com.alibaba.cloud.sentinel.annotation.SentinelRestTemplate;
import com.example.exception.ExceptionUtil;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class OrderServiceRestApplication {

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

    public static void main(String[] args) {
        SpringApplication.run(OrderServiceRestApplication.class, args);
    }

}
```

8.3.2 服务熔断处理类

ExceptionUtil.java 必须使用静态方法。

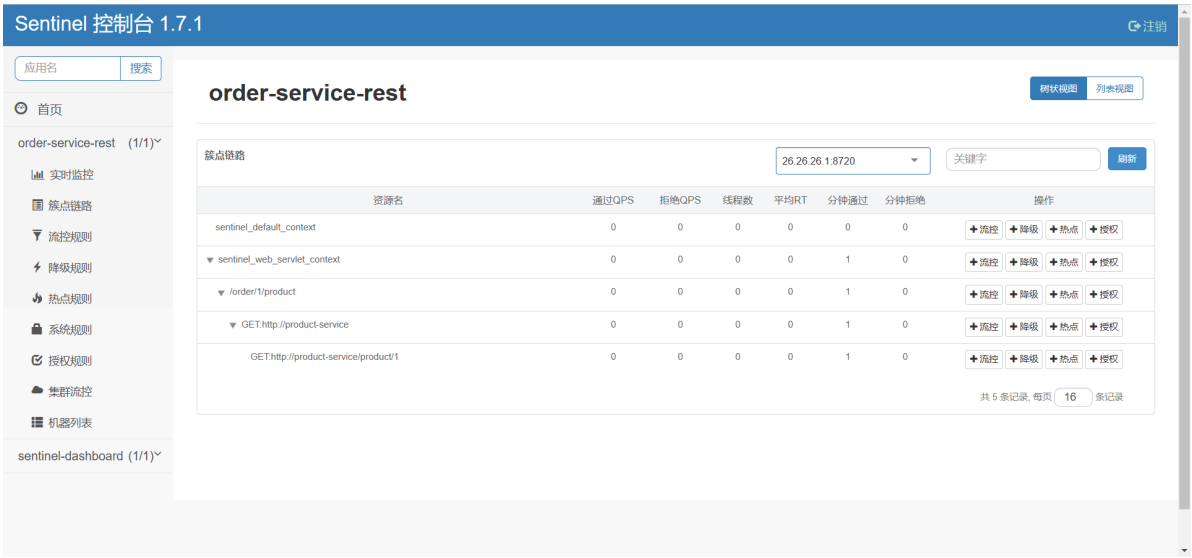
```
package com.example.exception;

import com.alibaba.cloud.sentinel.rest.SentinelClientHttpResponse;
import com.alibaba.csp.sentinel.slots.block.BlockException;
import com.alibaba.fastjson.JSON;
import com.example.pojo.Product;
import org.springframework.http.HttpRequest;
import org.springframework.http.client.ClientHttpRequestExecution;
import org.springframework.http.client.ClientHttpResponse;

public class ExceptionUtil {

    // 服务流量控制处理
    public static ClientHttpResponse handleException(HttpRequest request,
```

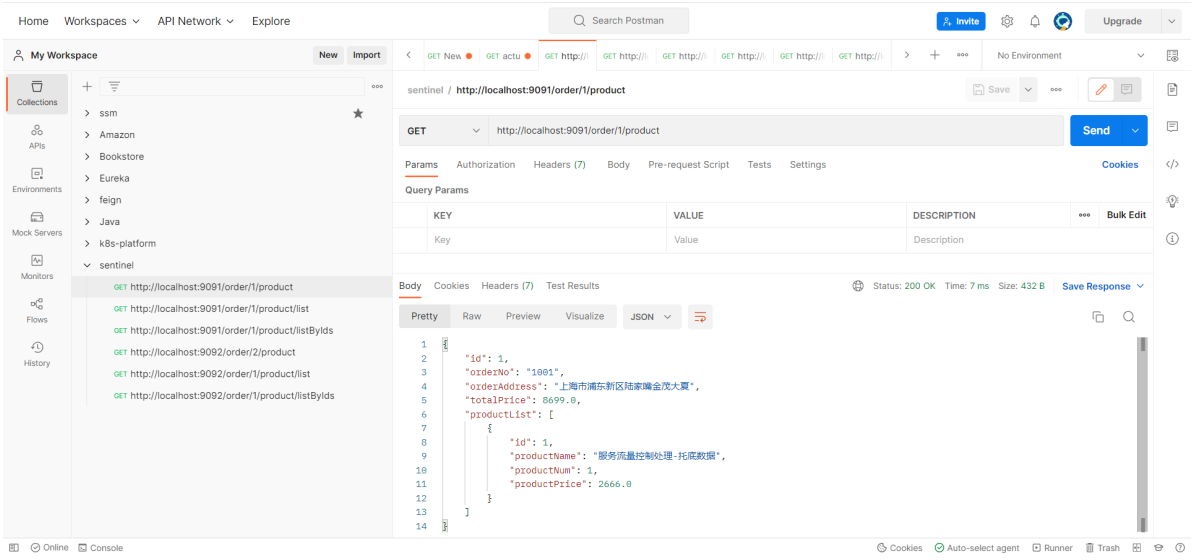
```
                                                    byte[] body,
                                                    ClientHttpRequestExecution
execution,
                                                    BlockException exception) {
        exception.printStackTrace();
        return new SentinelClientHttpResponse(
                JSON.toJSONString(new Product(1, "服务流量控制处理-托底数据", 1, 2666D)));
    }

    // 服务熔断降级处理
    public static ClientHttpResponse fallback(HttpRequest request,
                                              byte[] body,
                                              ClientHttpRequestExecution
execution,
                                              BlockException exception) {
        exception.printStackTrace();
        return new SentinelClientHttpResponse(
                JSON.toJSONString(new Product(1, "服务熔断降级处理-托底数据", 1, 2666D)));
    }

}
```

8.3.3 访问控制台

控制台设置流量控制规则，定义资源访问的 QPS 为 1（每秒能处理查询数目）。

快速刷新页面多次访问： http://localhost:9090/order/1/product 结果如下：






```
{
    "id": 1,
    "orderNo": "1001",
    "orderAddress": "上海市浦东新区陆家嘴金茂大夏",
    "totalPrice": 8699.0,
    "productList": [
        {
            "id": 1,
            "productName": "服务流量控制处理-托底数据",
            "productNum": 1,
            "productPrice": 2666.0
        }
    ]
}
```


8.4 OpenFeign支持

其实不管是 Hystrix 还是 Sentinel 对于 Feign 的支持，核心代码基本上是一致的，只需要修改依赖和配置文件即可.

8.4.1 添加依赖

```
<!-- spring cloud openfeign 依赖 -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<!-- spring cloud alibaba sentinel 依赖 -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
</dependency>
```

8.4.2 配置文件

```
server:
  port: 9091 # 端口

spring:
  application:
    name: order-service-feign # 应用名称
  cloud:
    sentinel:
      transport:
        port: 8719
        dashboard: localhost:8080

# 配置 Eureka Server 注册中心
eureka:
  instance:
    prefer-ip-address: true       # 是否使用 ip 地址注册
    instance-id: ${spring.cloud.client.ip-address}:${server.port} # ip:port
  client:
    service-url:                  # 设置服务注册中心地址
      defaultZone: http://localhost:8761/eureka/,http://localhost:8762/eureka/

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

8.4.3 熔断降级

```
package com.example.fallback;

import com.example.pojo.Product;
import com.example.service.ProductService;
import feign.hystrix.FallbackFactory;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

/**
 * 服务熔断降级处理可以捕获异常
```

```
 */
@Component
public class ProductServiceFallbackFactory implements FallbackFactory<ProductService>
{

    // 获取日志，在需要捕获异常的方法中进行处理
    Logger logger = LoggerFactory.getLogger(ProductServiceFallbackFactory.class);

    @Override
    public ProductService create(Throwable throwable) {
        return new ProductService() {
            @Override
            public Product selectProductById(Integer id) {
                logger.error("product-service 服务的 selectProductById 方法出现异常，异常信
息如下："
                        + throwable);
                return new Product(id, "托底数据", 1, 2666D);
            }
        };
    }

}
```

8.4.4 消费服务

8.4.4.1 ProductService

```
package com.example.service;

import com.example.fallback.ProductServiceFallbackFactory;
import com.example.pojo.Product;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

// 声明需要调用的服务
@FeignClient(value = "product-service", fallbackFactory =
ProductServiceFallbackFactory.class)
public interface ProductService {

    /**
     * 根据主键查询商品
     *
     * @param id
     * @return
     */
    @GetMapping("/product/{id}")
    Product selectProductById(@PathVariable("id") Integer id);

}
```

8.4.4.2 OrderServiceImpl

```
package com.example.service.impl;

import com.example.pojo.Order;
import com.example.service.OrderService;
import com.example.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Arrays;

@Service
public class OrderServiceImpl implements OrderService {

    @Autowired
    private ProductService productService;

    /**
     * 根据主键查询订单
     *
     * @param id
     * @return
     */
    @Override
    public Order selectOrderById(Integer id) {
        return new Order(id, "order-001", "中国", 2666D,
                Arrays.asList(productService.selectProductById(1)));
    }

}
```

8.4.4.3 OrderController

```
package com.example.controller;

import com.example.pojo.Order;
import com.example.service.OrderService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/order")
public class OrderController {

    @Autowired
    private OrderService orderService;

    /**
     * 根据主键查询订单
     *
     * @param id
     * @return
```

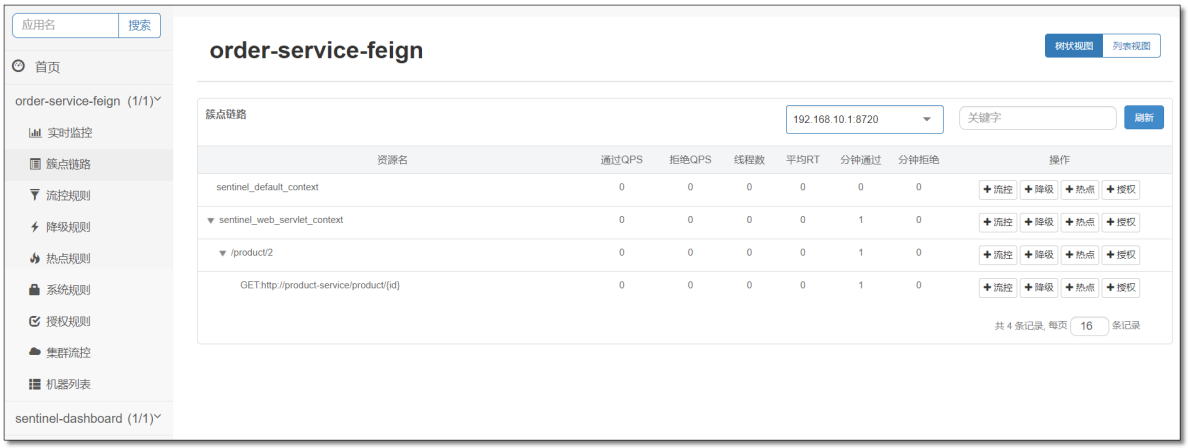
```
     */
    @GetMapping("/{id}")
    public Order selectOrderById(@PathVariable("id") Integer id) {
        return orderService.selectOrderById(id);
    }

}
```

8.4.4.4 启动类

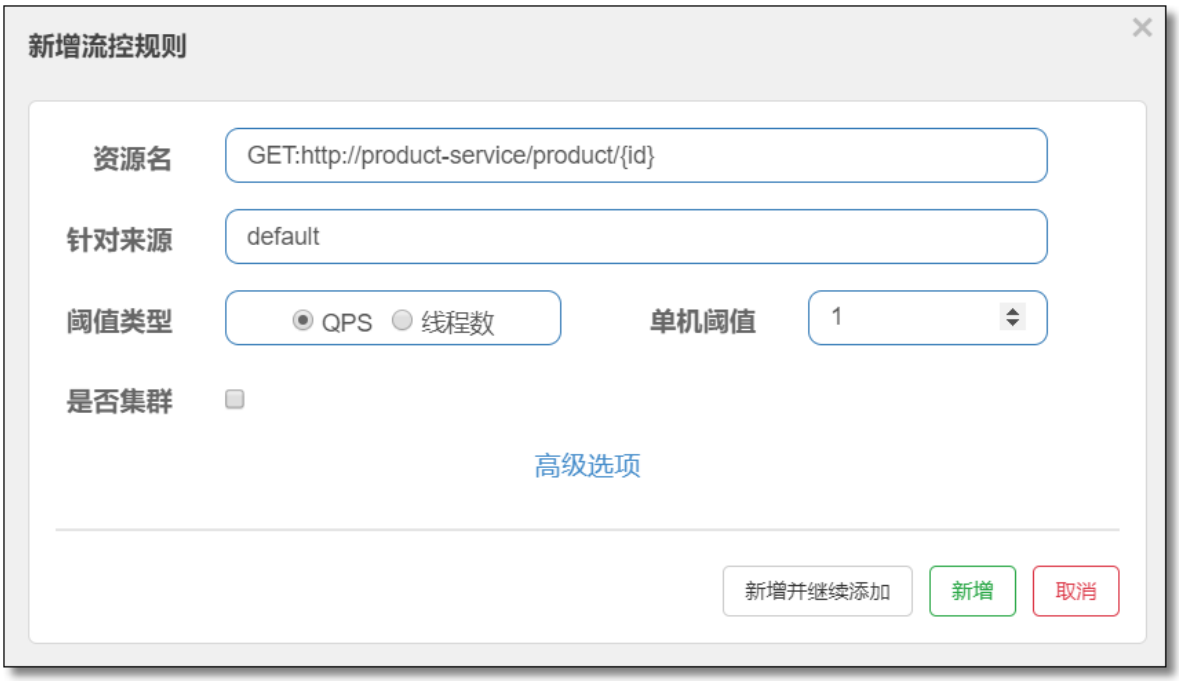
```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

// 开启 FeignClients 注解
@EnableFeignClients
@SpringBootApplication
public class OrderServiceFeignApplication {

    public static void main(String[] args) {
        SpringApplication.run(OrderServiceFeignApplication.class, args);
    }

}
```

8.4.4.5 测试

控制台信息如下:



添加流量控制规则，定义资源访问的 QPS 为 1（每秒能处理查询数目）。



快速刷新页面多次访问： http://localhost:9091/order/1/product 结果如下：


```
{
    "id": 1,
    "orderNo": "1001",
    "orderAddress": "上海市浦东新区陆家嘴金茂大夏",
    "totalPrice": 8699.0,
    "productList": [
        {
            "id": 1,
            "productName": "服务流量控制处理-托底数据",
            "productNum": 1,
            "productPrice": 2666.0
        }
    ]
}
```