安装并运行 Nacos

下载 Nacos

https://github.com/alibaba/nacos/releases

解压安装包（Mac平台）

以nacos-server-1.3.2.tar.gz为例

tar -zxvf nacos-server-1.3.2.tar.gz

mv nacos /usr/local

cd /usr/local/nacos/bin

启动服务器（独立模式）

sh startup.sh -m standalone

运行成功后访问http://localhost:8848/nacos

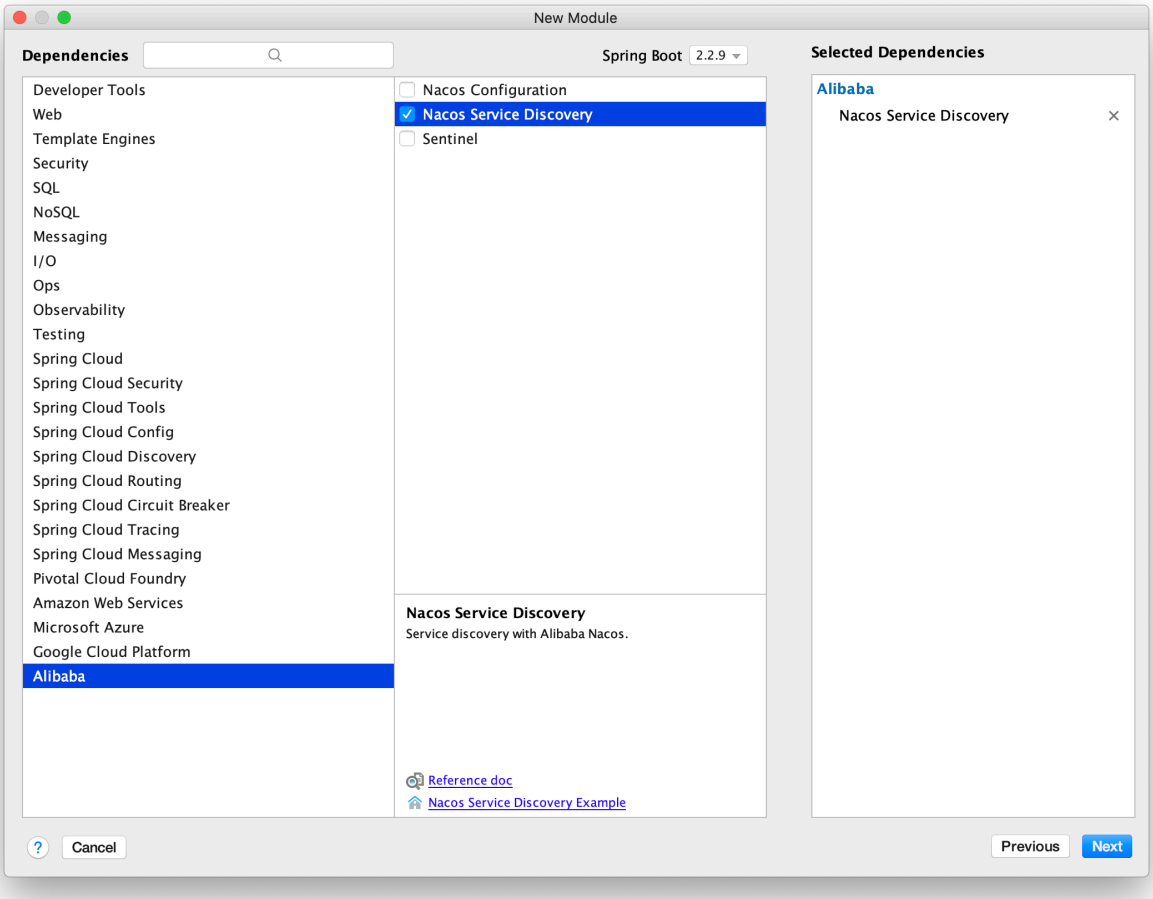
默认账号密码：nacos/nacos

关闭服务器

sh shutdown.sh

Alibaba Nacos Service Discovery 搭建

第一步：创建项目



第二步：添加注解 @EnableEurekaServer
第三步：增加配置application.yml

```
server:
  port: 8761

eureka:
  instance:
    hostname: localhost
  client:
  #声明自己是个服务端
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

第四步：访问注册中心页面

网关 Gateway

熔断降级 Alibaba Sentinel

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

安装 Sentinel Dashboard

下载 dashboard

下载URL：https://github.com/alibaba/Sentinel/releases

运行 dashboard

以 sentinel-dashboard-1.8.0.jar 为例

```
java -jar sentinel-dashboard-1.8.0.jar
```

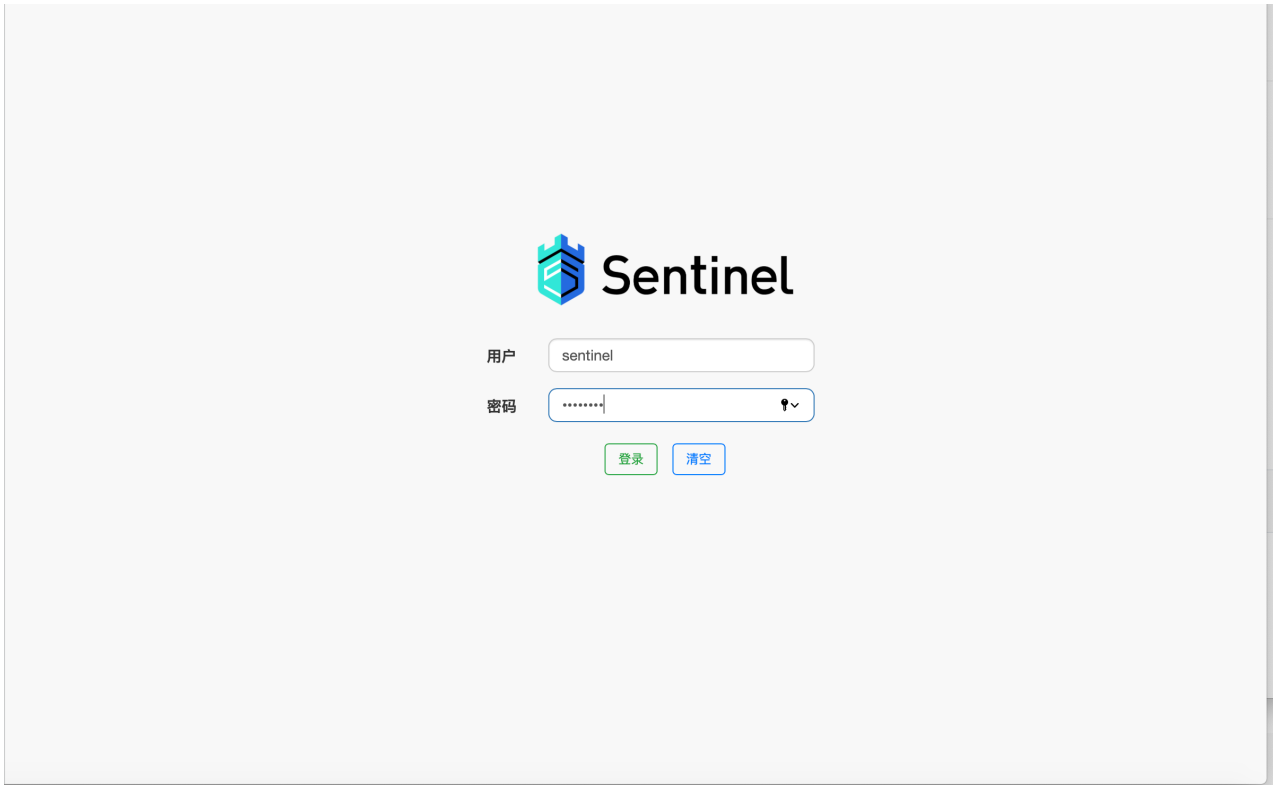
官网文档中运行 dashboard 命令：

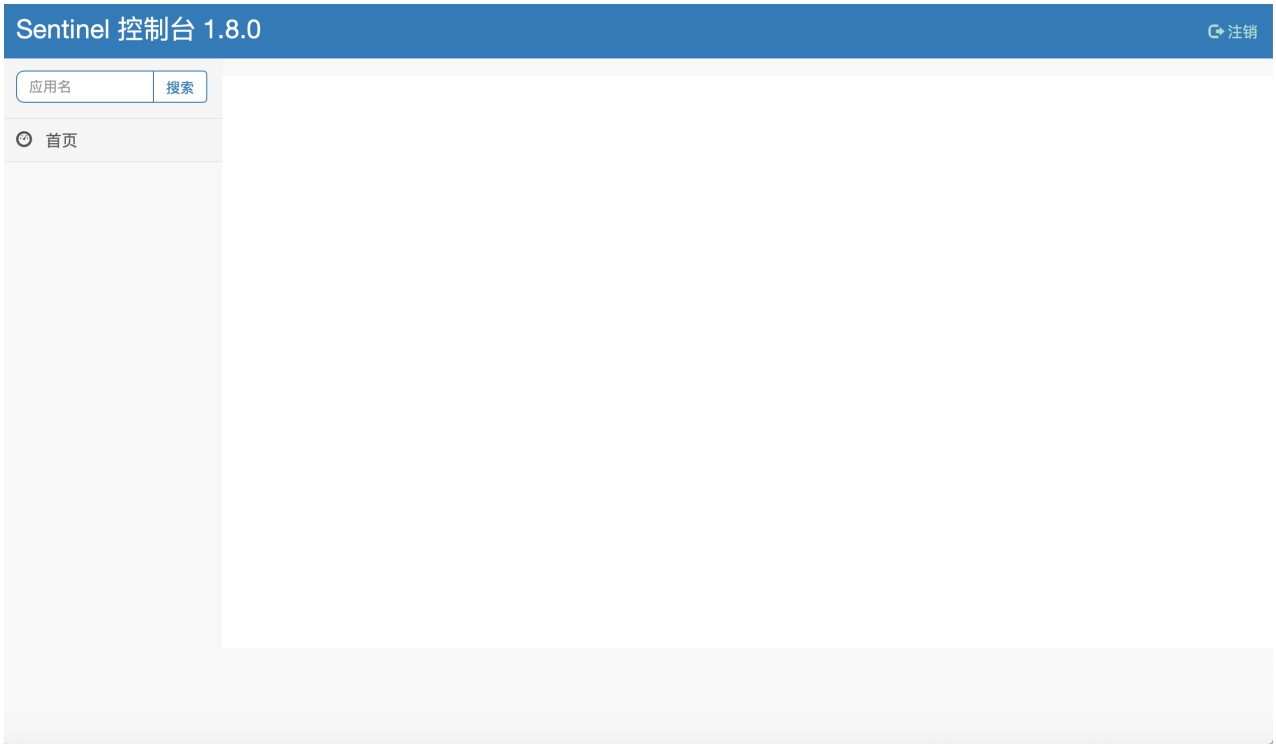
```
java -Dserver.port=8080 -Dcsp.sentinel.dashboard.server=localhost:8080 -
Dproject.name=sentinel-dashboard -jar sentinel-dashboard.jar
```

访问 dashboard

访问URL：http://localhost:8080/#/dashboard

用户/密码：sentinel/sentinel





Sentinel 初始化监控

启动 nacos-server

参考安装并运行 Nacos

Module

以修改 product-service 为例

修改 pom.xml

添加依赖

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

修改 bootstrap.yml

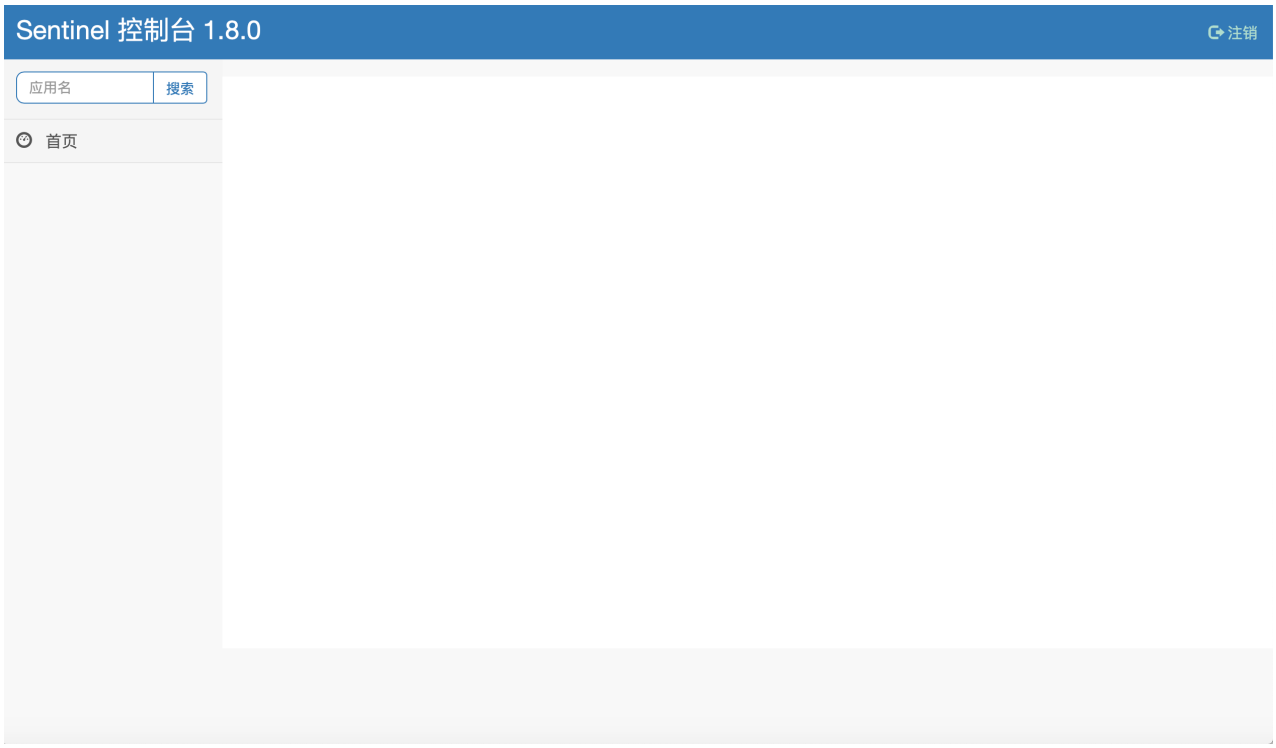
```
server:
  port: 9000
spring:
  application:
    name: product-service
  cloud:
    nacos:
      discovery:
```

```
      server-addr: 127.0.0.1:8848
    config:
      server-addr: 127.0.0.1:8848
      file-extension: yaml
  sentinel:
    transport:
      # 配置 Sentinel Dashboard 地址
      dashboard: 127.0.0.1:8080
      # 默认8719端口，假如被占用会自动从8719开始一次+1扫描，直至找到未被占用的端口
      port: 8719
```

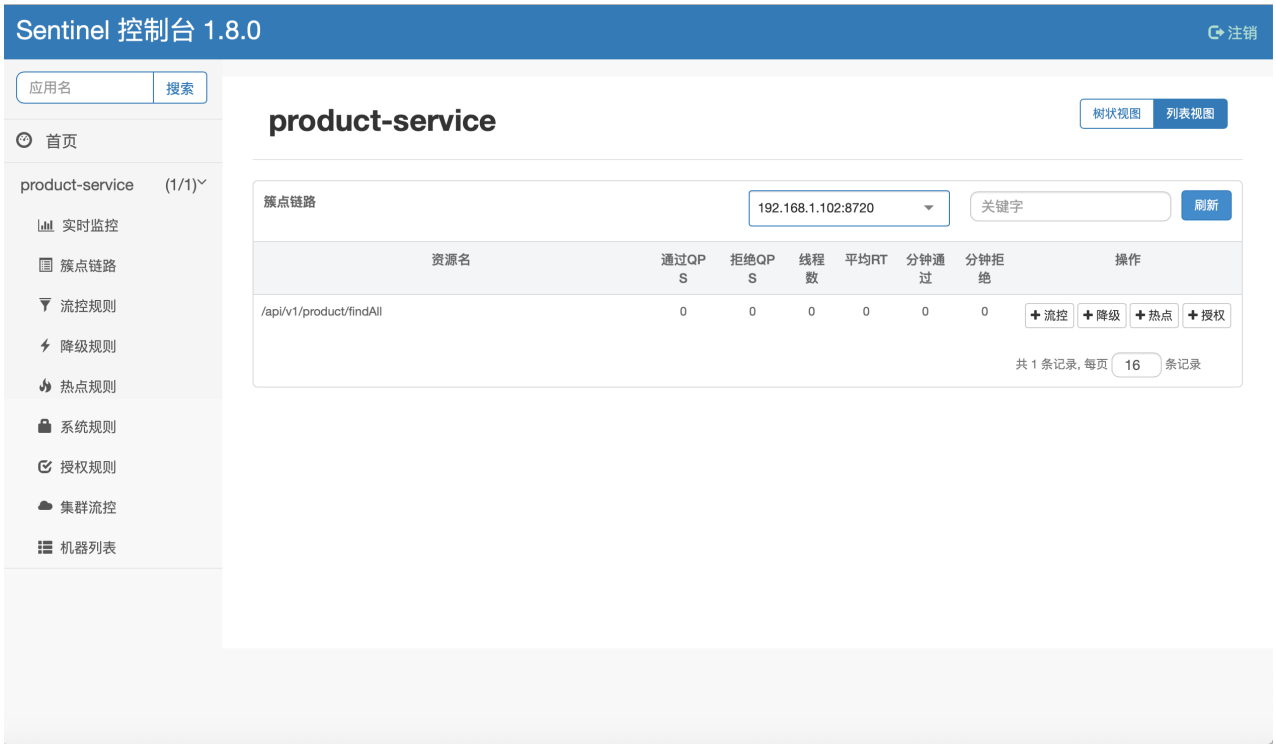
启动 Sentinel Dashboard

启动 product-service

查看 sentinel dashboard



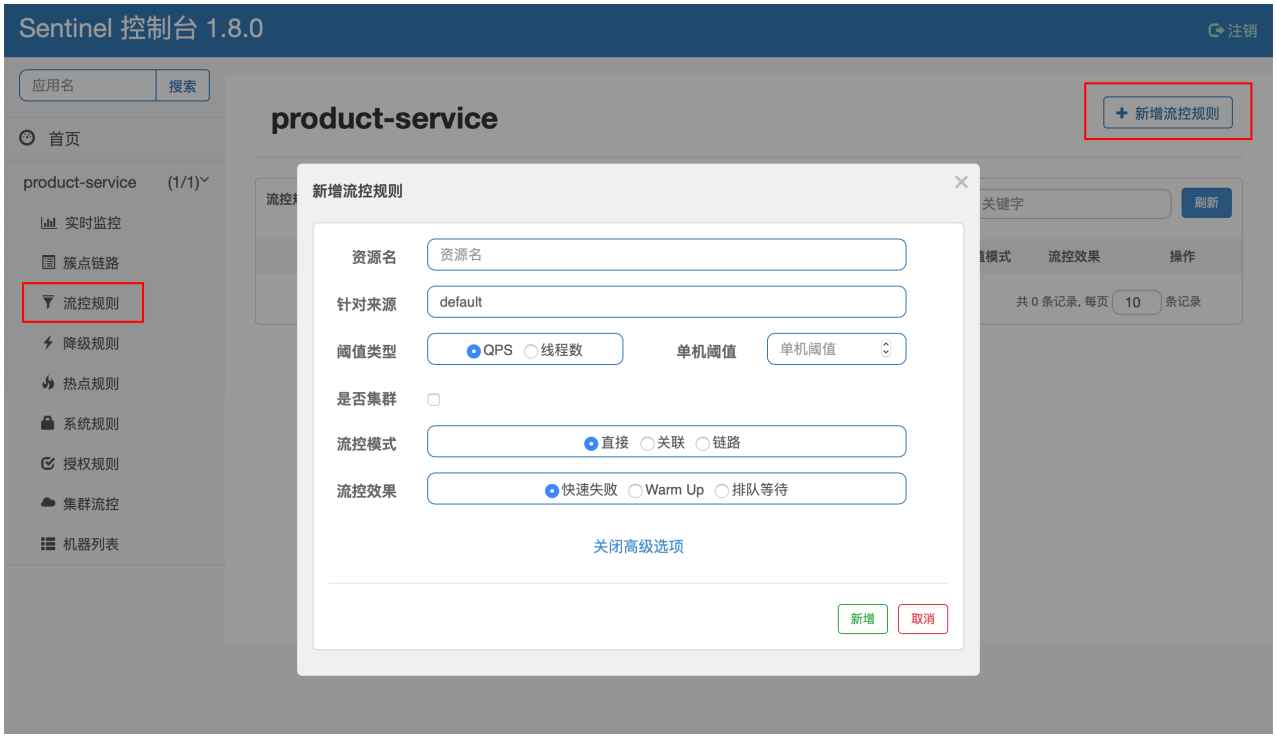
注意：sentinel 用的是懒加载机制，需要访问一次sentinel才能监控到



Sentinel 流控

流控规则

新增流控规则的两种方式：



资源名：唯一，默认请求路径

针对来源：填写微服务名，可以针对调用者进行限流，默认default（不区分来源）

阈值类型+单机阈值：

QPS：当调用该api的QPS（每秒请求数量）达到阈值时，进行限流

线程数：当调用该api的线程数达到阈值时，进行限流

是否集群：

不需要集群

流控模式：

直接：达到限流条件时，直接限流

关联：当关联的资源达到阈值时，限流自己

链路：只记录指定链路上的流量（指定资源从入口资源进来的流量，如果达到阈值，进行限流）（api级别的针对来源）

流控效果：

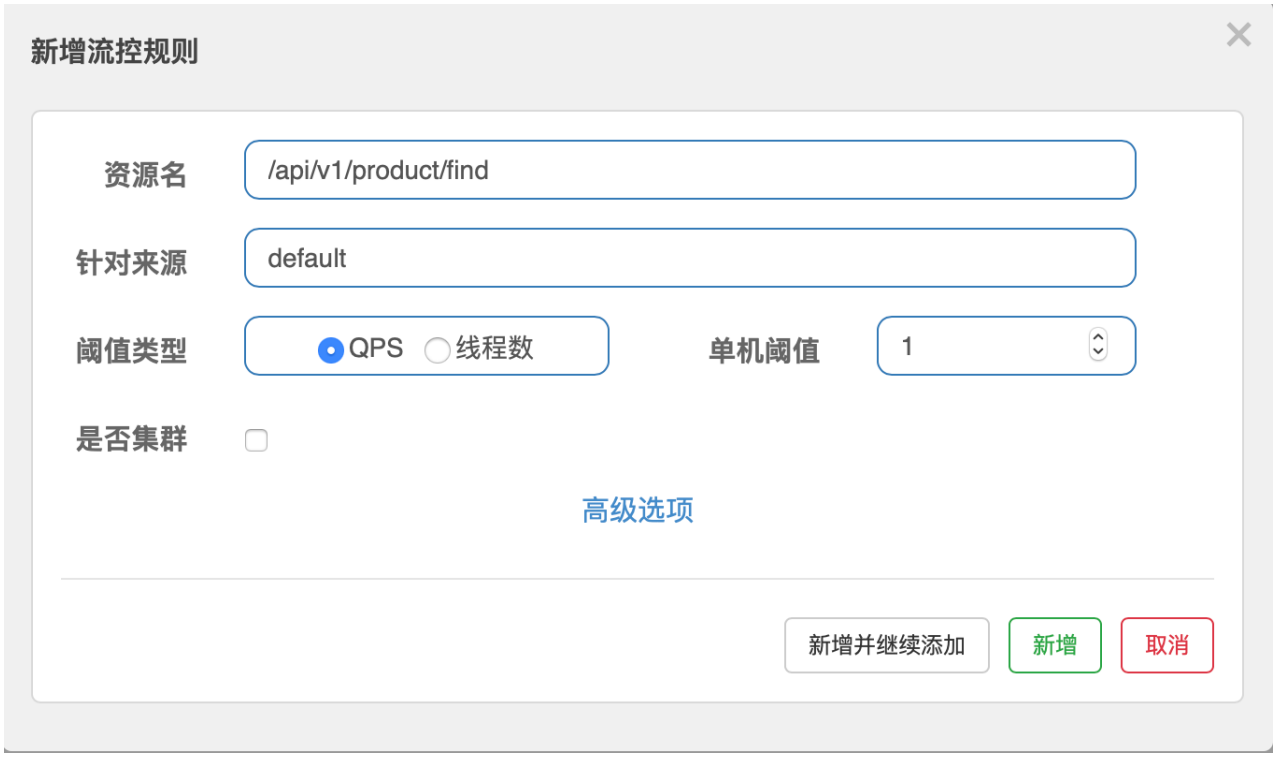
快速失败：直接失败，抛异常

Warm Up：根据coldFactor（冷加载因子，默认3）的值，从阈值/coldFactor经过预热时长，才达到设置的QPS阈值

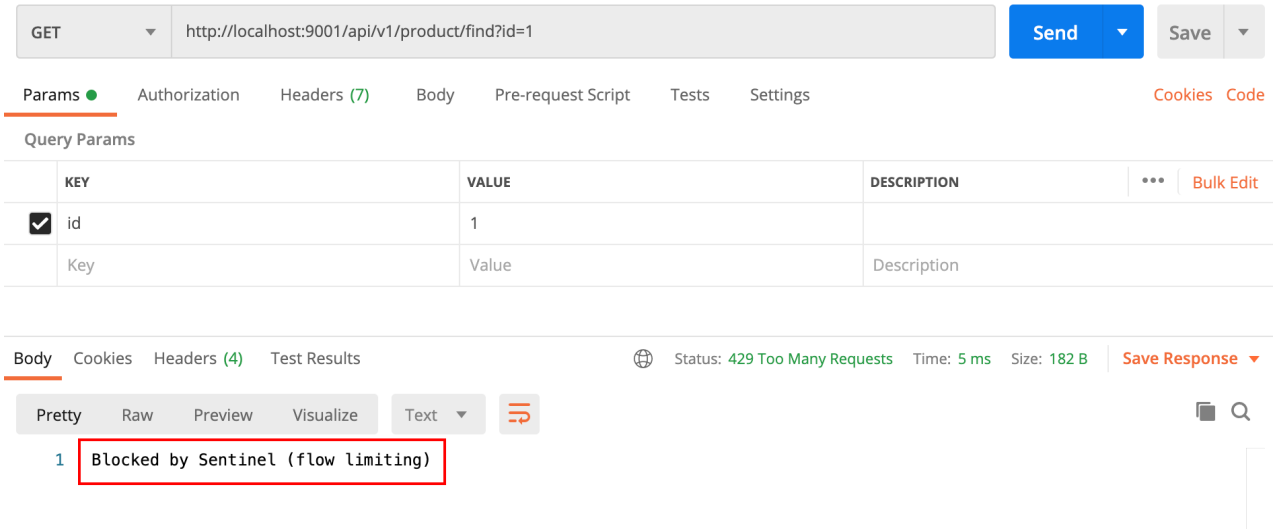
排队等待：匀速排队。让请求匀速通过，阈值类型必须设置为QPS，否则无效

测试

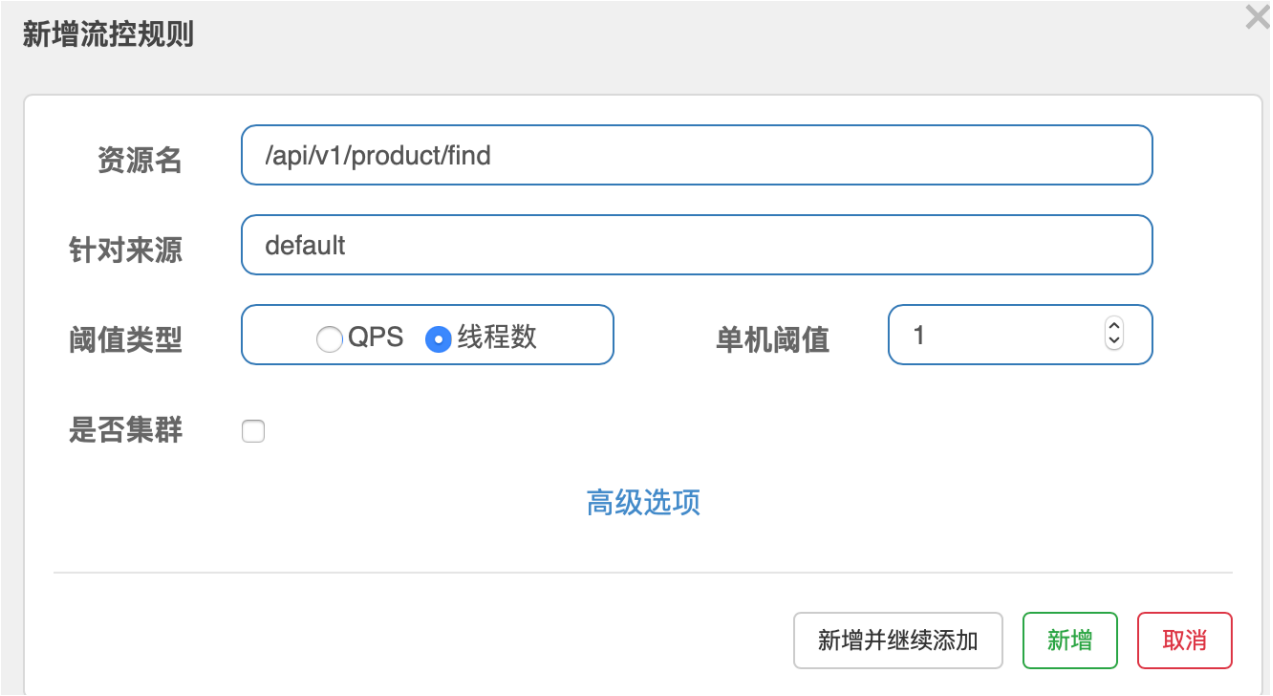
- QPS-直接-快速失败



每秒请求数超出1则sentinel抛出异常



- 线程数-直接-快速失败



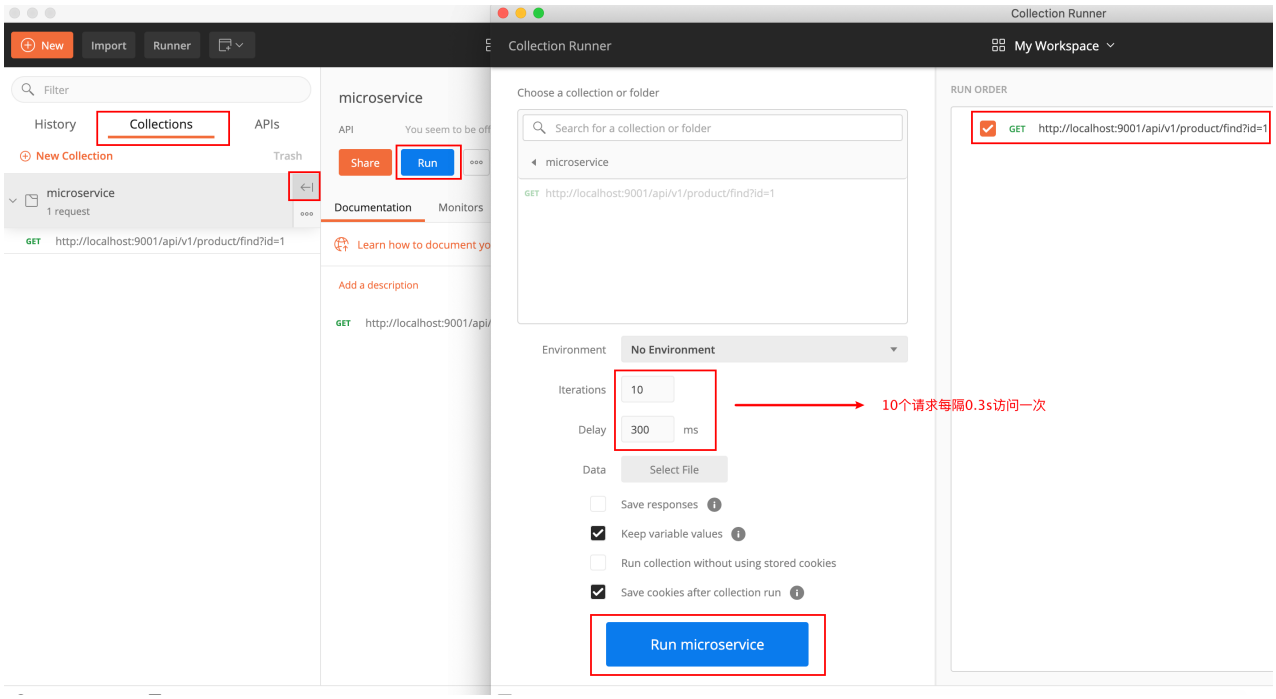
修改/api/v1/product/find接口，添加try/catch块后重启

```
@RequestMapping("/findAll")
public List<Product> findAll(){
    try {
        TimeUnit.MILLISECONDS.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return productService.findAll();
}
```

- QPS-关联-快速失败

/api/v1/product/findAll 依赖 /api/v1/product/find，当/api/v1/product/find 超出QPS时，/api/v1/product/findAll访问失败

使用postman模拟并发密集访问/api/v1/product/find



结果：

postman运行期间访问/api/v1/product/findAll 返回 Blocked by Sentinel (flow limiting)

- QPS-直接-Warm-up

阈值除以coldFactor（默认为3），经过预热时长后才会达到阈值

阈值为10，预热时长5秒。即初始阈值为10/3约等于3，过了5秒后阈值才慢慢恢复到10

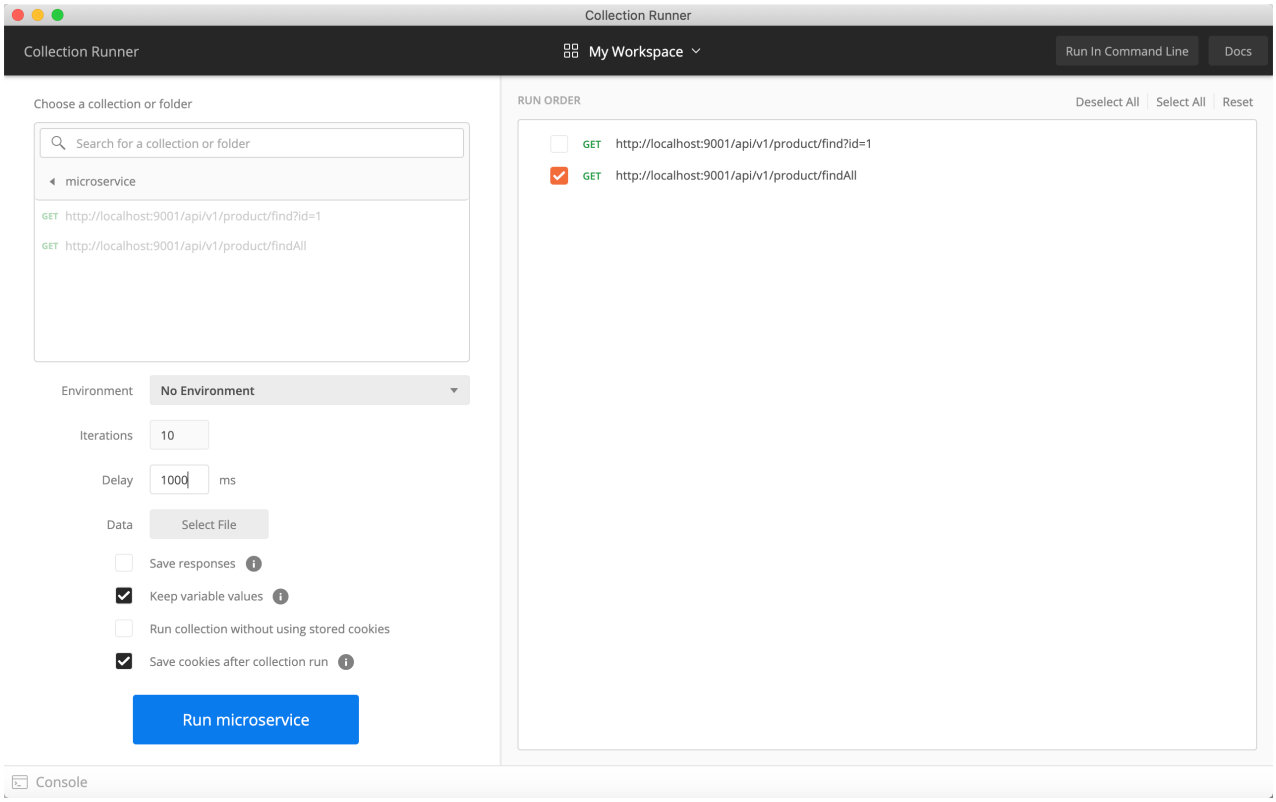
- QPS-直接-排队等待

QPS为1，超出QPS就排队等待，等待的超时时间为2000毫秒

修改 ProductController

ProductController 添加 @Slf4j

```
@RequestMapping("/findAll")
public List<Product> findAll(){
    log.info(Thread.currentThread().getName());
    return productService.findAll();
}
```



Sentinel 降级

降级规则

https://github.com/alibaba/Sentinel/wiki/熔断降级

- 慢调用比例 (`SLOW_REQUEST_RATIO`)：选择以慢调用比例作为阈值，需要设置允许的慢调用 RT（即最大的响应时间），请求的响应时间大于该值则统计为慢调用。当单位统计时长（`statIntervalMs`）内请求数目大于设置的最小请求数目，并且慢调用的比例大于阈值，则接下来的熔断时长内请求会自动被熔断。经过熔断时长后熔断器会进入探测恢复状态（HALF-OPEN 状态），若接下来的一个请求响应时间小于设置的慢调用 RT 则结束熔断，若大于设置的慢调用 RT 则

会再次被熔断。

- 异常比例（`ERROR_RATIO`）：当单位统计时长（`statIntervalMs`）内请求数目大于设置的最小请求数目，并且异常的比例大于阈值，则接下来的熔断时长内请求会自动被熔断。经过熔断时长后熔断器会进入探测恢复状态（HALF-OPEN 状态），若接下来的一个请求成功完成（没有错误）则结束熔断，否则会再次被熔断。异常比率的阈值范围是 `[0.0, 1.0]`，代表 0% - 100%。

- 异常数（`ERROR_COUNT`）：当单位统计时长内的异常数目超过阈值之后会自动进行熔断。经过熔断时长后熔断器会进入探测恢复状态（HALF-OPEN 状态），若接下来的一个请求成功完成（没有错误）则结束熔断，否则会再次被熔断。

测试

Sentinel 热点

热点规则

https://github.com/alibaba/Sentinel/wiki/热点参数限流

测试

使用@SentinelResource注解：ProductController 中添加测试方法，@SentinelResource值唯一，并配置blockHandler方法

```
@RequestMapping(value = "/test/paramFlowRule")
@SentinelResource(value = "test/paramFlowRule", blockHandler =
"testParamFlowRule_handler")
public String testParamFlowRule(@RequestParam(value = "param1", required =
false) String param1, @RequestParam(value = "param2", required = false) String
param2) {
    return "Test ParamFlowRule";
}

public String testParamFlowRule_handler(String param1, String param2,
BlockException e) {
    return "Test ParamFlowRule Handler";
}
```

配置规则：资源名为@SentinelResource中的value值

结果：超出单机阈值，通过blockHandler中的方法处理 Test ParamFlowRule Handler

测试参数例外项

配置规则

参数类型**仅支持基本类型和字符串类型**

@SentinelResource处理的是dashboard配置的违规情况，使用blockHandler中配置的兜底方法处理

Sentinel 授权

授权规则

https://github.com/alibaba/Sentinel/wiki/系统自适应限流

系统保护规则是应用整体维度的，而不是资源维度的，并且**仅对入口流量生效**

测试

@SentinelResource

blockHandlerClass

blockHandler

配置违规

fallback

运行异常

若都配置，则只会进入blockHandler处理（即配置处理优先）

规则持久化

将配置规则持久化到Nacos保存

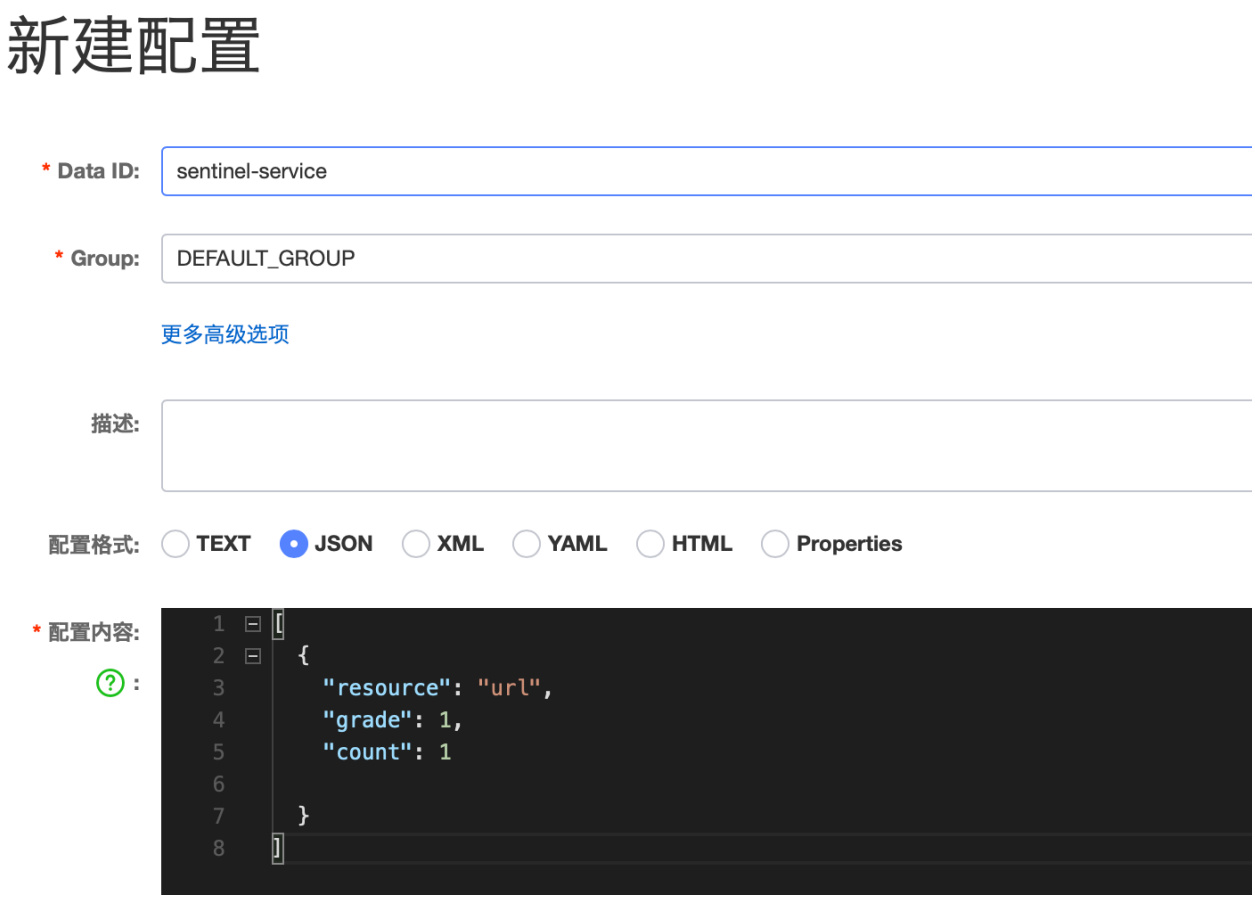
```
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-datasource-nacos</artifactId>
</dependency>
```

修改 bootstrap.yml

```
server:
  port: 9000
```

```yaml
spring:
  application:
    name: product-service
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
      config:
        server-addr: 127.0.0.1:8848
        file-extension: yaml
    sentinel:
      transport:
        # 配置 Sentinel Dashboard 地址
        dashboard: 127.0.0.1:8080
        # 默认8719端口，假如被占用会自动从8719开始一次+1扫描，直至找到未被占用的端口
        port: 8719
      datasource:
        dsl:
          nacos:
            server-addr: 127.0.0.1:8848
            dataId: sentinel-service
            groupId: DEFAULT_GROUP
            data-type: json
            rule-type: flow
```

Nacos 添加配置



resource: 资源名称

resource：资源名称； limitApp：来源应用； grade：阈值类型，0表示线程数，1表示QPS； count：单机阈值； strategy：流控模式，0表示直接，1表示关联，2表示链路； controlBehavior：流控效果，0表示快速失败，1表示Warm Up，2表示排队等待； clusterMode：是否集群。