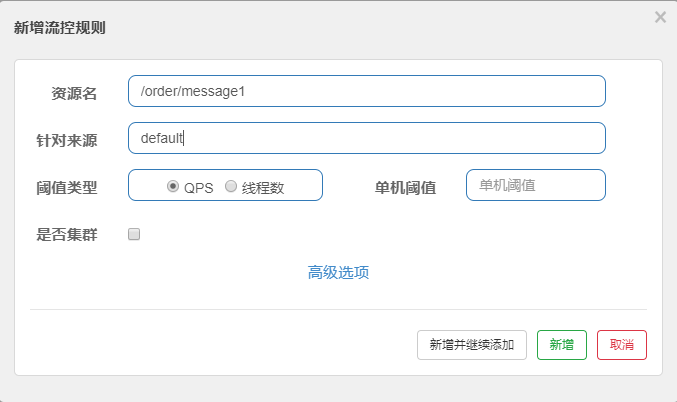
Sentinel规则

# 流控规则

流量控制，其原理是监控应用流量的QPS(每秒查询率) 或并发线程数等指标，当达到指定的阈值时对流量进行控制，以避免被瞬时的流量高峰冲垮，从而保障应用的高可用性。

第1步: 点击簇点链路，我们就可以看到访问过的接口地址，然后点击对应的流控按钮，进入流控规则配置页面。新增流控规则界面如下:



**资源名**：唯一名称，默认是请求路径，可自定义

**针对来源**：指定对哪个微服务进行限流，默认指default，意思是不区分来源，全部限制 **阈值类型/单机阈值**：

QPS（每秒请求数量）: 当调用该接口的QPS达到阈值的时候，进行限流线程数：当调用该接口的线程数达到阈值的时候，进行限流

**是否集群**：暂不需要集群

接下来我们以QPS为例来研究限流规则的配置。

# 简单配置流控规则

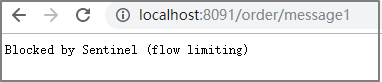
## 简单配置

我们先做一个简单配置，设置阈值类型为QPS，单机阈值为3。即每秒请求量大于3的时候开始限流。

接下来，在流控规则页面就可以看到这个配置。



然后快速访问/order/message1 接口，观察效果。此时发现，当QPS > 3的时候，服务就不能正常响应，而是返回Blocked by Sentinel (ﬂow limiting)结果。



## 配置流控模式

### 概述

点击上面设置流控规则的**编辑**按钮，然后在编辑页面点击**高级选项**，会看到有流控模式一栏。



sentinel共有三种流控模式，分别是：

直接（默认）：接口达到限流条件时，开启限流

关联：当关联的资源达到限流条件时，开启限流 [适合做应用让步]

链路：当从某个接口过来的资源达到限流条件时，开启限流下面呢分别演示三种模式：

### 直接流控模式

直接流控模式是最简单的模式，当指定的接口达到限流条件时开启限流。上面案例使用的就是直接流控 模式。

### 关联流控模式

关联流控模式指的是，当指定接口关联的接口达到限流条件时，开启对指定接口开启限流。

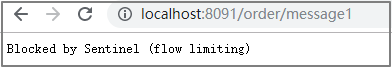
第1步：配置限流规则, 将流控模式设置为关联，关联资源设置为的 /order/message2。



第3步：通过postman软件向/order/message2连续发送请求，注意QPS一定要大于3



第4步：访问/order/message1,会发现已经被限流



### 链路流控模式

链路流控模式指的是，当从某个接口过来的资源达到限流条件时，开启限流。它的功能有点类似于针对 来源配置项，区别在于：**针对来源是针对上级微服务，而链路流控是针对上级接口，也就是说它的粒度 更细。**

第1步： 编写一个service，在里面添加一个方法message

@Service

public class OrderServiceImpl3 { @SentinelResource("message") public void message() {

System.out.println("message");

}

}

第2步： 在Controller中声明两个方法，分别调用service中的方法m

第3步: 禁止收敛URL的入口 context

@RestController @Slf4j

public class OrderController3 {

@Autowired

private OrderServiceImpl3 orderServiceImpl3;

@RequestMapping("/order/message1") public String message1() {

orderServiceImpl3.message(); return "message1";

}

@RequestMapping("/order/message2") public String message2() {

orderServiceImpl3.message(); return "message2";

}

}



从1.6.3 版本开始，Sentinel Web ﬁlter默认收敛所有URL的入口context，因此链路限流不生效。

1.7.0 版本开始（对应SCA的2.1.1.RELEASE)，官方在CommonFilter 引入了

WEB\_CONTEXT\_UNIFY

false

URL 进行链路限流。

参数，用于控制是否收敛context。将其配置为

即可根据不同的

SCA 2.1.1.RELEASE之后的版本,可以通过配置spring.cloud.sentinel.web-context-unify=false即可关闭收敛

我们当前使用的版本是SpringCloud Alibaba 2.1.0.RELEASE，无法实现链路限流。

目前官方还未发布SCA 2.1.2.RELEASE，所以我们只能使用2.1.1.RELEASE，需要写代码的形式实现

**(1)** 暂时将SpringCloud Alibaba的版本调整为2.1.1.RELEASE

<spring-cloud-alibaba.version>2.1.1.RELEASE</spring-cloud-alibaba.version>

1. 配置文件中关闭sentinel的CommonFilter实例化

spring:

cloud:

sentinel:

filter:

enabled: false

1. 添加一个配置类，自己构建CommonFilter实例

第4步: 控制台配置限流规则

package com.itheima.config;

import com.alibaba.csp.sentinel.adapter.servlet.CommonFilter; import org.springframework.boot.web.servlet.FilterRegistrationBean; import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class FilterContextConfig {

@Bean

public FilterRegistrationBean sentinelFilterRegistration() { FilterRegistrationBean registration = new FilterRegistrationBean(); registration.setFilter(new CommonFilter()); registration.addUrlPatterns("/\*");

// 入口资源关闭聚合

registration.addInitParameter(CommonFilter.WEB\_CONTEXT\_UNIFY, "false"); registration.setName("sentinelFilter");

registration.setOrder(1); return registration;

}

}





第5步: 分别通过/order/message1 和/order/message2 访问, 发现2没问题, 1的被限流了

## **配置流控效果**

**快速失败（默认）**: 直接失败，抛出异常，不做任何额外的处理，是最简单的效果

**Warm Up**：它从开始阈值到最大QPS阈值会有一个缓冲阶段，一开始的阈值是最大QPS阈值的

1/3，然后慢慢增长，直到最大阈值，适用于将突然增大的流量转换为缓步增长的场景。

**排队等待**：让请求以均匀的速度通过，单机阈值为每秒通过数量，其余的排队等待； 它还会让设置一个超时时间，当请求超过超时间时间还未处理，则会被丢弃。