

Sentinel 控制台的功能主要包括：流量控制、降级控制、热点配置、系统规则和授权规则等

## 安装 sentinel 的控制台

### 下载地址

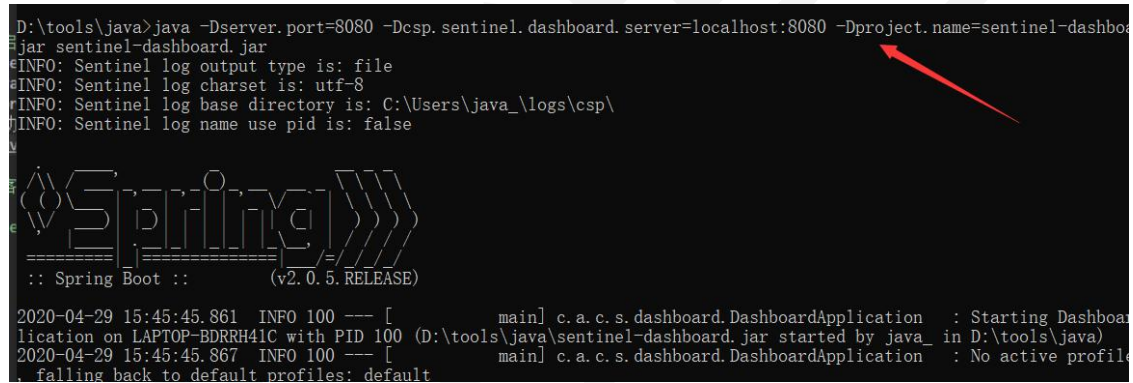
Sentinel 控制台下载地址：<https://github.com/alibaba/Sentinel/releases>  
版本自己选择

### 启动控制台

到 sentinel 所在的目录运行下面的命令

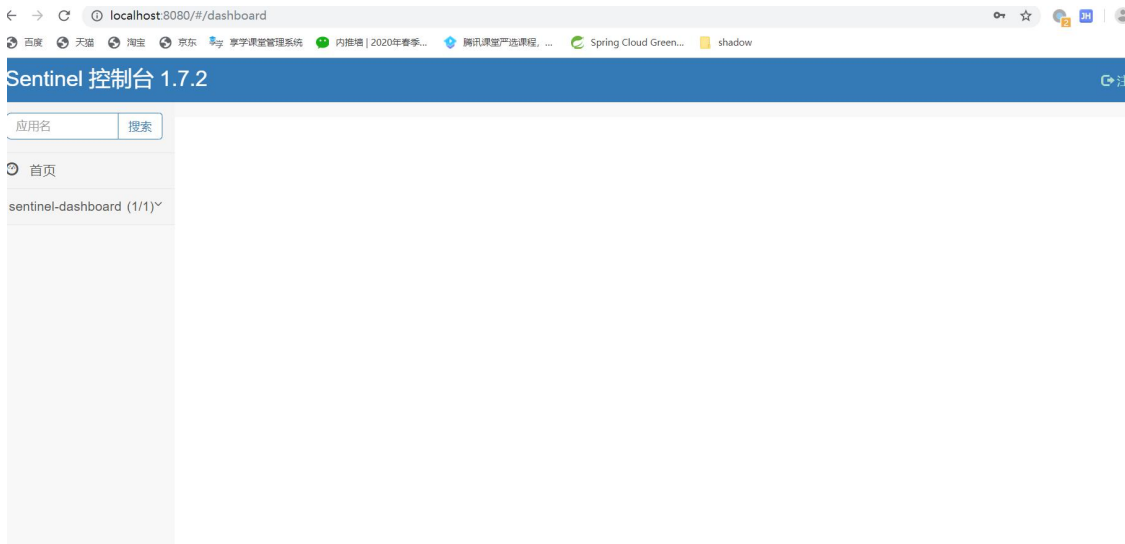
```
java -Dserver.port=8080 -Dcsp.sentinel.dashboard.server=localhost:8080 -Dproject.name=sentinel-dashboard -jar sentinel-dashboard-版本号.jar
```

启动之后访问 localhost:8080;登录即可用户名和密码默认是 sentinel



```
D:\tools\java>java -Dserver.port=8080 -Dcsp.sentinel.dashboard.server=localhost:8080 -Dproject.name=sentinel-dashboard -jar sentinel-dashboard.jar
INFO: Sentinel log output type is: file
INFO: Sentinel log charset is: utf-8
INFO: Sentinel log base directory is: C:\Users\java_\logs\csp\
INFO: Sentinel log name use pid is: false
:: Spring Boot :: (v2.0.5.RELEASE)
2020-04-29 15:45:45.861 INFO 100 --- [main] c.a.c.s.dashboard.DashboardApplication : Starting DashboardApplication on LAPTOP-BDRRH41C with PID 100 (D:\tools\java\sentinel-dashboard.jar started by java_ in D:\tools\java)
2020-04-29 15:45:45.867 INFO 100 --- [main] c.a.c.s.dashboard.DashboardApplication : No active profile found, falling back to default profiles: default
```

登录之后看到左侧的菜单只有默认的一个；因为现在 sentinel 还没有发现其他机器



## 客户端搭建--sc

基于 spring-cloud 的项目来搭建一个 sentinel 客户端

### 1、加入 spring-cloud 的 sentinel 依赖

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

### 2、建立一个 controller

```
package com.shadow.web;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class UserController {  
  
    @GetMapping("login")  
    public String login() throws InterruptedException {  
        System.out.println("login----");  
        return "success";  
    }  
}
```

```

    @GetMapping("log")
    public String log(){
        System.out.println("log----");
        return "success";
    }
}

```

### 3、yml 配置

```

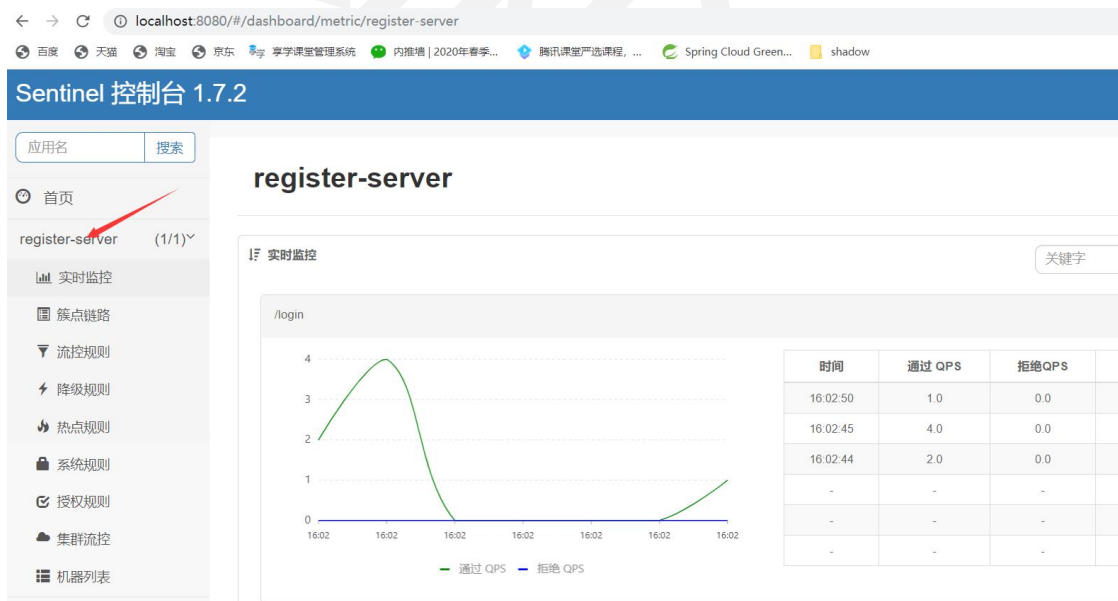
server:
  port: 9000
spring:
  application:
    name: register-server
  sentinel:
    transport:
      port: 8719 #如果加了客户端他就会开启一个 http server 为了 dashboard 能够发过来
      dashboard: localhost:8080

```

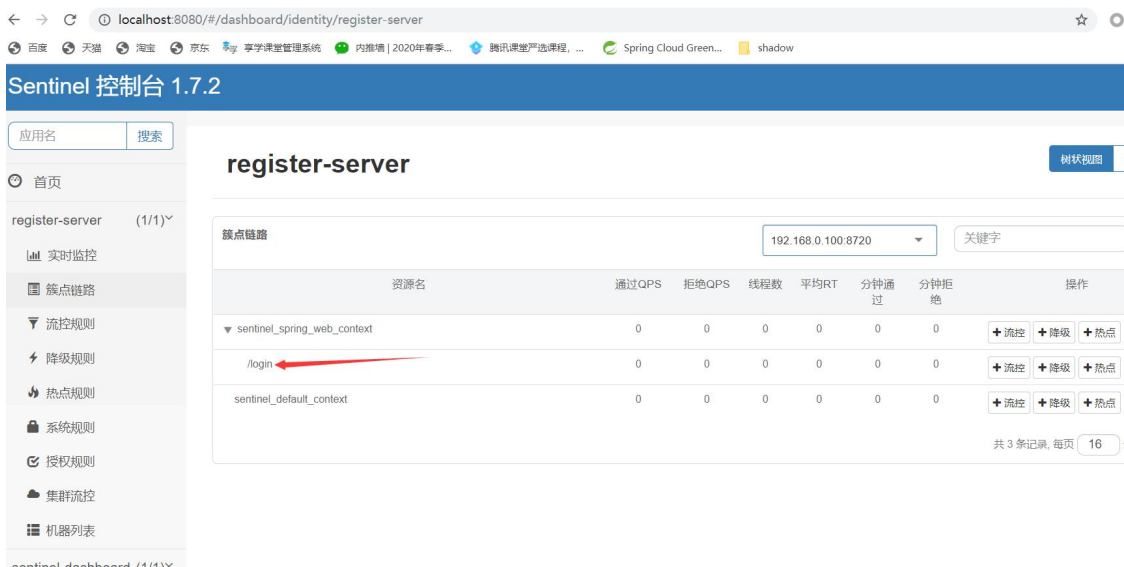
spring.cloud.sentinel.transport.dashboard——sentinel 控制台的 ip 和端口地址

spring.cloud.sentinel.transport.port——sentinel 客户端和控制台通信的端口

然后启动项目；继而访问 localhost:8080/login；跟着刷新 sentinel 的控制台可以看到 sentinel 已经发现了我们的项目 server-register



并且可以在簇点链路中看到刚刚那个请求



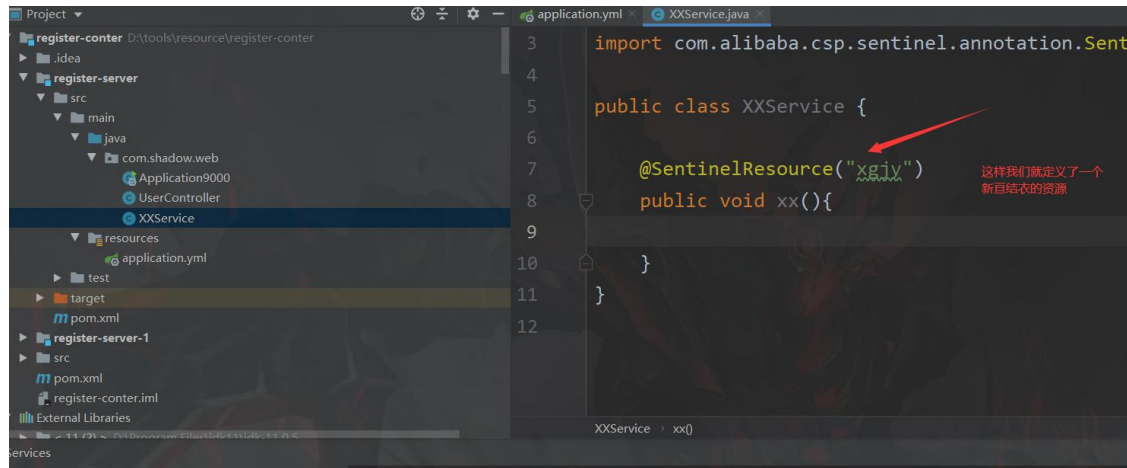
接下来就对这个请求进行流控、降级、授权、热点等配置等等配置了；先来介绍如何添加流控吧；点击+流控的按钮



怎么配置呢？首先了解这概念吧

## sentinel 术语

1、资源：标识资源的唯一名称，springboot 项目默认为 controller 当中的请求路径，也可以在客户端中使用 SentinelResource 配置；比如你有个 service 当中的 xx 方法；你想对这个 xx 方法进行流控那么这个 xx 方法必须是一个资源；



2、针对来源：Sentinel 可以针对服务调用者进行限流，填写微服务名称即 spring.application.name，默认为 default，不区分来源

3、阈值类型、单机阈值

QPS——每秒钟的请求数量，当调用该资源的 QPS 达到阈值的时候，进行限流；  
线程数——当调用该资源的线程数达到阈值的时候，进行限流

4、是否集群：默认不集群；

5、流控模式：

直接：当资源调用达到限流条件的时，直接限流；

关联：当关联的资源请求达到阈值的时候，限流自己；

链路：下节课说

6、流控效果：

快速失败：直接失败；

Warm Up：根据冷加载因子默认值为 3 的值，从阈值/3，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

## QPS 直接失败

新增流控规则

资源名

/login

针对来源

default

阈值类型

☒ QPS ☐ 线程数

单机阈值

1

是否集群

☐

高级选项

新增并继续添加

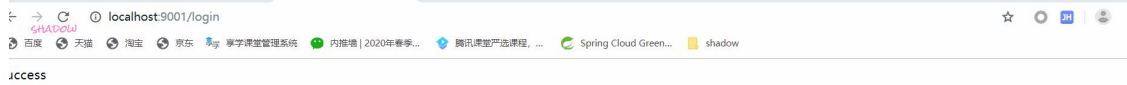
新增

取消

上面设置的效果是，1 秒钟内请求/login 资源的次数达到 1 次以上的时候，进行限流

### 效果演示

首先一秒请求一次 login 没有问题；然后发挥你手速；一秒超过一次之后页面返回 Blocked by Sentinel (flow limiting)；如果手速不够多练练吧——总归有办法的



## 线程数直接失败

把 `login` 方法让其睡眠一秒；因为无论你单身多少年；你的手速不可能超过计算机；所以在刷新的情况下永远只有一个线程；如果让他睡眠 1 秒；这样就能模拟出来服务器这边开多个线程来处理的场景了；

```
@GetMapping("login")
public String login() throws InterruptedException {
    System.out.println("login----");
    Thread.sleep(1000);
    return "success";
}
```

改完之后重启你的项目，然后先访问一次 `login`；因为你重启之后 `sentinel` 控制台这边就检测不到了；需要重新访问一次；跟着添加流控规则

新增流控规则

资源名

/login

针对来源

default

阈值类型

☐ QPS

☒ 线程数

单机阈值

2

是否集群

☐

高级选项

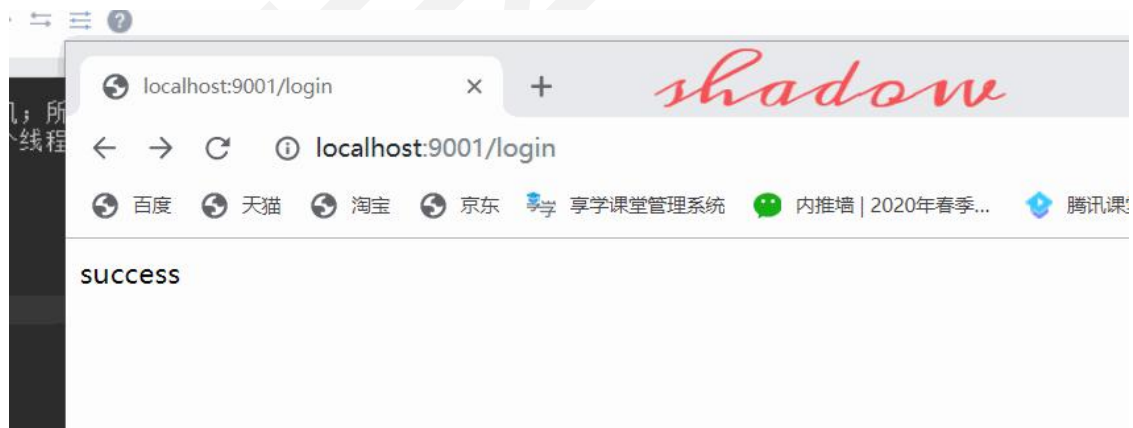
新增并继续添加

新增

取消

## 效果演示

首先慢慢刷新 login 请求；——等睡眠时间过；服务器永远只有一个线程，发觉没问题；但是你疯狂在地址栏里面点击回车（注意不要去刷新，因为刷新是会等到服务器返回之后才能点击；所以效果演示不出来，最好是点击回车在地址栏）就会失败





## 流控模式关联

首先把 login 方法当中的睡眠删除；然后重启项目，访问一下 login；继而去配置流控规则

新增流控规则

资源名

/login

针对来源

default

阈值类型

☒ QPS ☐ 线程数

单机阈值

2

是否集群

☐

流控模式

☐ 直接 ☒ 关联 ☐ 链路

关联资源

/log

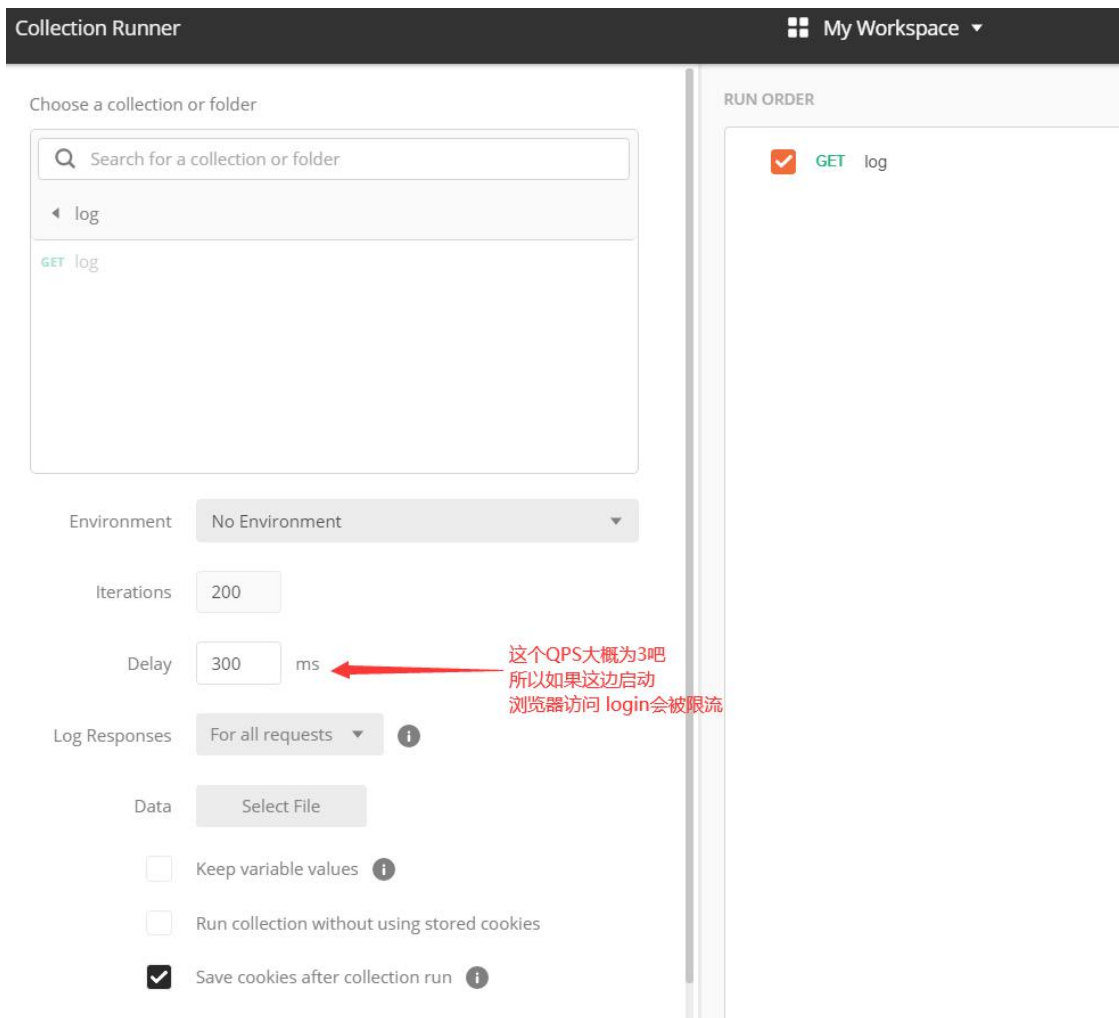
流控效果

☒ 快速失败 ☐ Warm Up ☐ 排队等待

关闭高级选项

上述配置当 1 秒内访问 /log 的次数大于 2 的时候，限流 /login

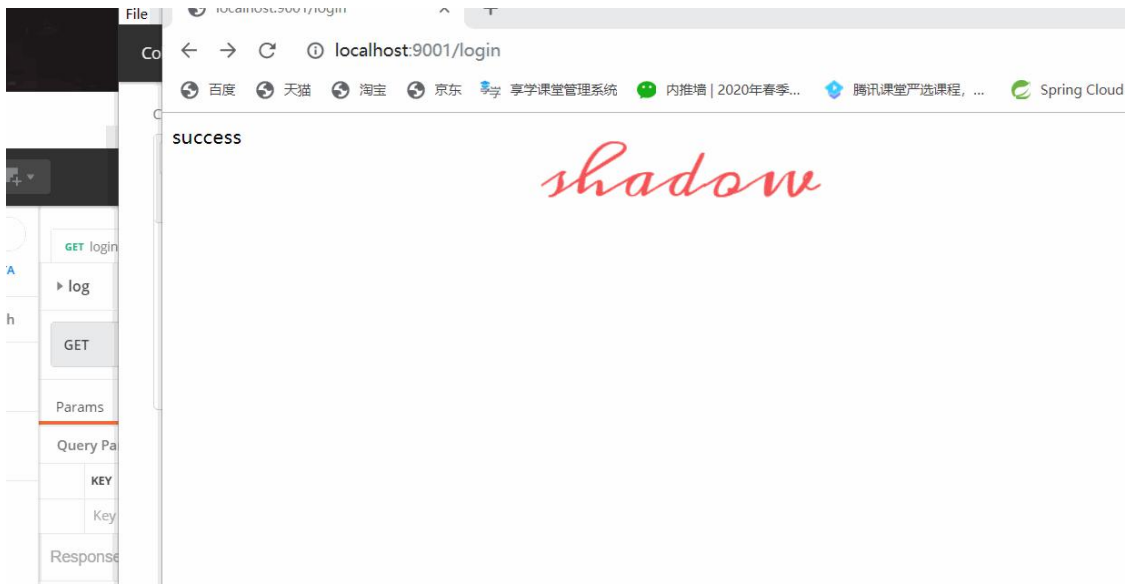
效果演示使用 postman 来密集访问 /log（QPS=3），然后我们手动浏览器请求 /login，看看效果。postman 的如下图



## 效果演示

在外面没有启动 postman 的时候 访问 login 没有问题；当我们启动 postman 之后 log 的 qps 达到了阈值；这个时候再去访问 login 被限流了

效果如下图



## 预热 Warm Up

Warm Up——预热/冷启动方式。当系统长期处于低访问量的情况下，流量突然增加时，可能瞬间把系统压垮。通过 warm up 方式，让通过的流量缓慢增加，在一定时间内逐渐增加到阈值上限，给冷系统一个预热的时间，避免问题

## 新增流控规则

资源名	<input type="text" value="/login"/>		
针对来源	<input type="text" value="default"/>		
阈值类型	<input checked="" type="radio"/> QPS <input type="radio"/> 线程数	单机阈值	<input type="text" value="10"/>
是否集群	<input type="checkbox"/>		
流控模式	<input checked="" type="radio"/> 直接 <input type="radio"/> 关联 <input type="radio"/> 链路		
流控效果	<input type="radio"/> 快速失败 <input checked="" type="radio"/> Warm Up <input type="radio"/> 排队等待		
预热时长	<input type="text" value="10"/>		

[关闭高级选项](#)

新增并继续添加 新增 取消

即请求 QPS 从 阈值 10/3 开始，经预热时长(10s)逐渐升至设定的 QPS 阈值（10）

### 效果演示

假设一开始我们疯狂刷新（让 QPS 大于 3）讲道理一般人的手速可以达到；你会看到被限流了；但是经过 10s 之后再也不会出错了；因为人的极限不可能 1 秒点到 10 次；子路老师这么强也不能达到；如果哪位兄弟 10s 之后还能点击出现限流；请你私聊我

效果演示



localhost:9001/login

shadow

百度 天猫 淘宝 京东 享学课堂管理系统 易推惠 2020年春季 腾讯课堂严选课程, ... Spring Cloud Gi

success

享学课堂