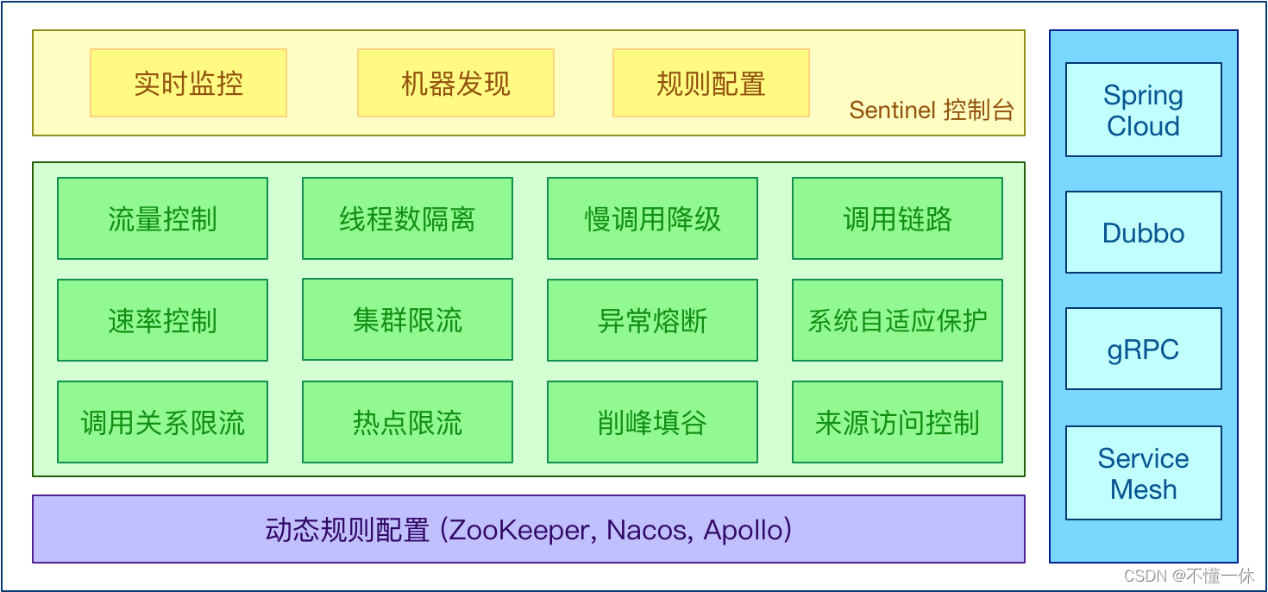
Sentinel熔断降级

1. 概述

Sentinel 是面向云原生微服务的流量监控，熔断降级组件。能够监控并保护你的微服务。

* 1. Sentinel 的主要特性



* 1. Sentinel 分为两个部分

1）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。

2）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

1.3 启动 Sentinel 控制台

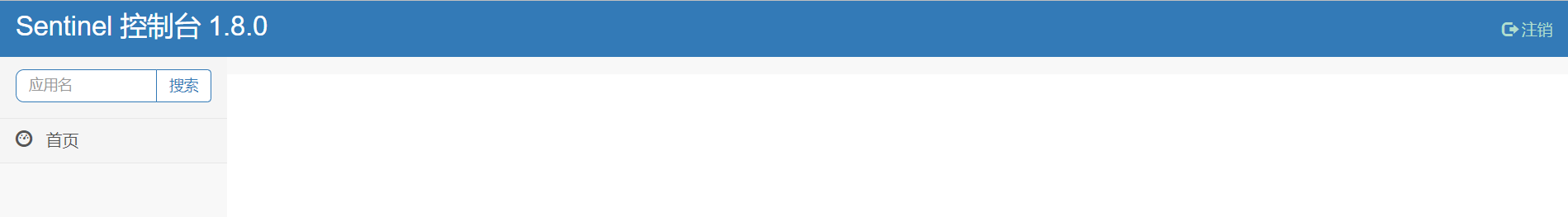
控制台简介以及安装启动：https://github.com/alibaba/Sentinel/wiki/控制台

jar 启动下载地址：https://github.com/alibaba/Sentinel/releases

启动控制台：sh java -Dserver.port=8858 -Dsentinel.dashboard.auth.username=psz -Dsentinel.dashboard.auth.password=psz -jar ./sentinel-dashboard-1.8.0.jar。其中 -Dserver.port=8858 用于指定 Sentinel 控制台端口为 8858。Sentinel 控制台引入基本的登录功能，默认用户名和密码都是 sentinel，此处用户名密码都是psz。



访问<http://127.0.0.1:8858>，用户名密码psz进行Sentinel控制台登录。



1. SpringCloud 生成 sentinel 客户端
   1. 启动 Sentinel 客户端

首先pom.xml配置如下依赖

<dependency>

<groupId>com.alibaba.cloud</groupId>

<artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>

</dependency>

然后application.yml中如下简单配置

spring:

application:

name: stock-service

cloud:

sentinel:

enabled: true

transport:

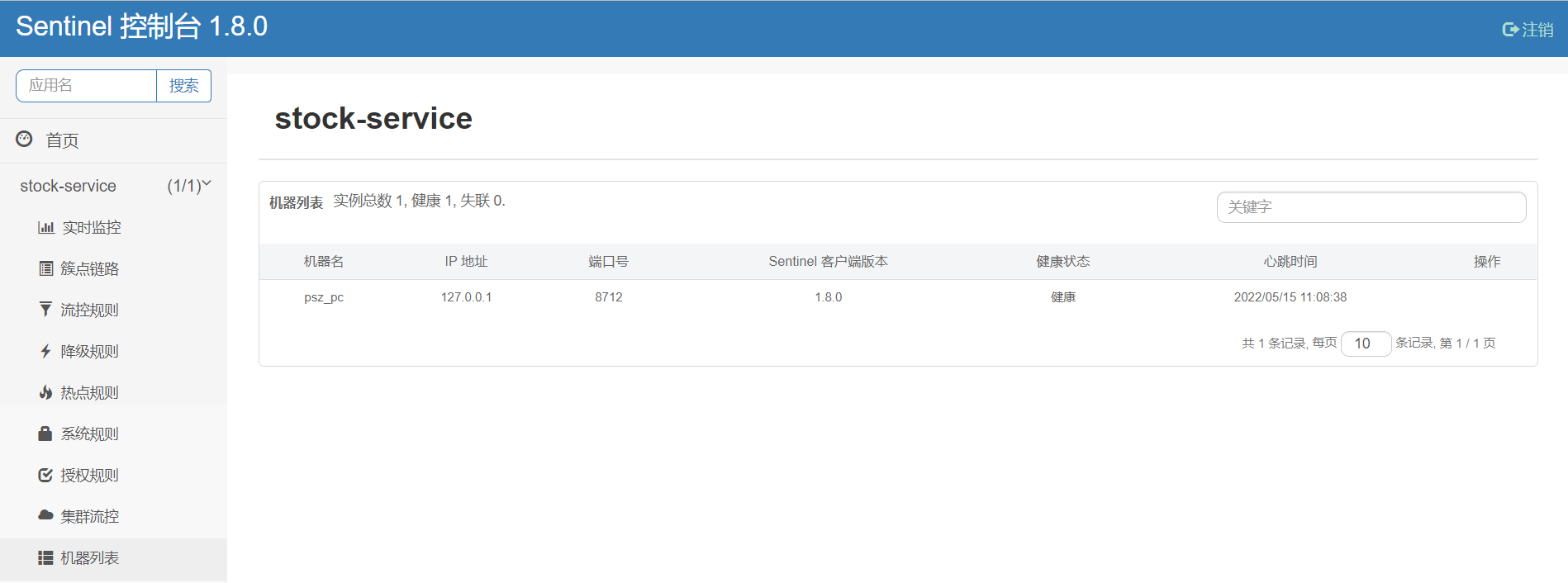
port: 8723

dashboard: 127.0.0.1:8858

clientIp: 127.0.0.1

* 1. Sentinel控制台

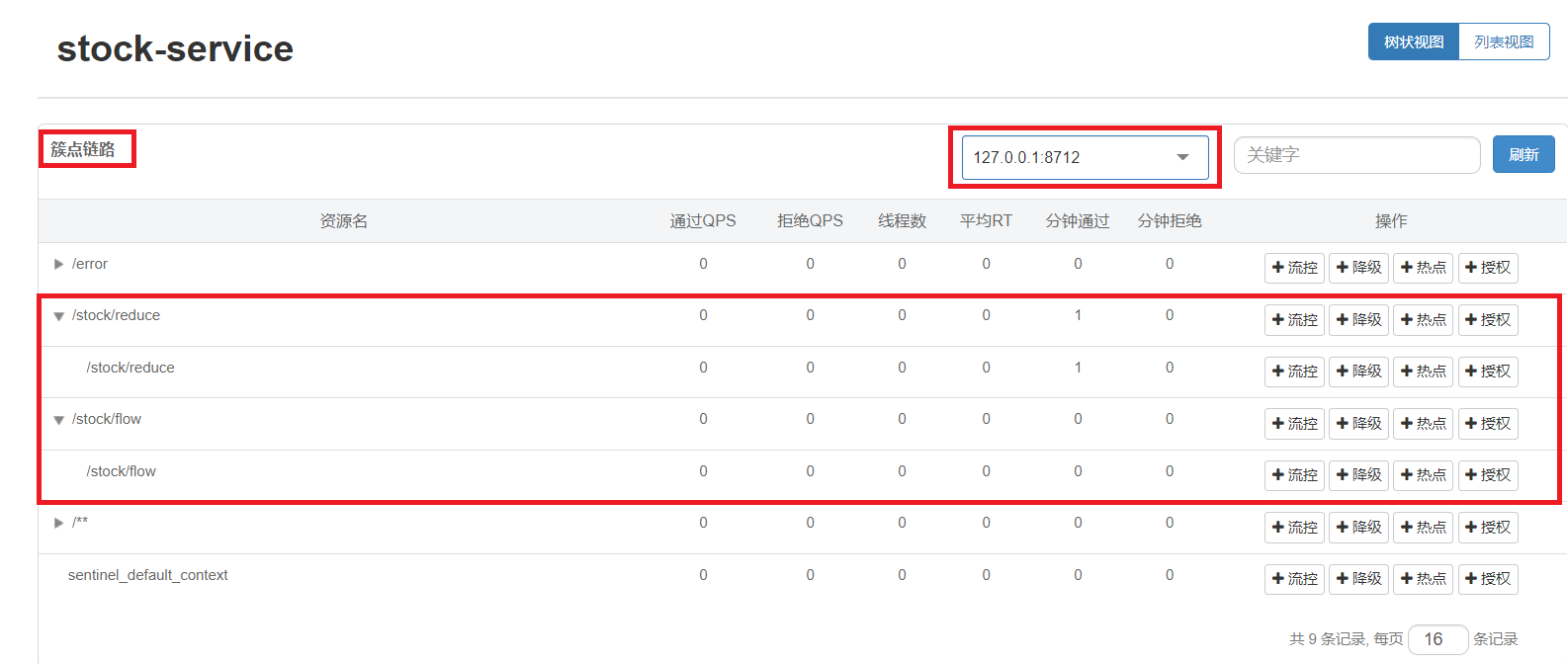
如果在控制台没有找到应用，调用一下进行了 Sentinel 埋点的 URL 或方法，因为 Sentinel 使用了 lazy load 策略。



可以看到，socket-service服务出现在机器列表中。

1. 接入点限流
   1. HTTP 埋点：

Sentinel starter 默认为所有的 HTTP 服务提供了限流埋点，如果只想对 HTTP 服务进行限流，那么只需要引入依赖，无需修改代码。（简单说就是所有的 controller 层接口默认提供限流埋点）



* 1. 自定义埋点：

如果需要对某个特定的方法进行限流或降级，可以通过 @SentinelResource 注解来完成限流的埋点，示例代码如下：

@SentinelResource("resource")

public String hello() {

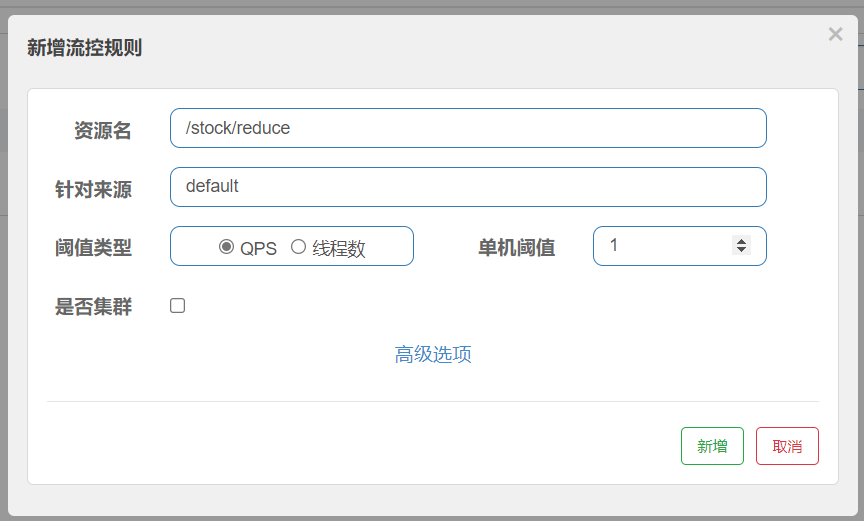
return "Hello";

}

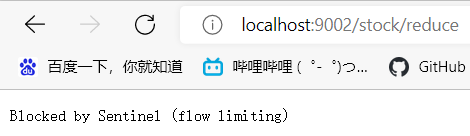
1. URL 流控规则
   1. sentinel 流控配置

点击流控规则，选择 127.0.0.1:8712 客户端，点击新增流控规则，资源名填写需要限流的 URL 相对路径，单机阈值选择需要限流的阈值，点击新增进行确认.

* 1. 如果没使用 @SentinelResource 注解，则资源名为 url 地址
  2. 如果使用 @SentinelResource 注解，则资源名为 @SentinelResource 注解指定的名字



调用流控配置的资源http://localhost:9002/stock/reduce，当 QPS 超过 1 时,可以看到限流效果如下



限流默认处理类 DefaultBlockExceptionHandler

public class DefaultBlockExceptionHandler implements BlockExceptionHandler {

public DefaultBlockExceptionHandler() {

}

public void handle(HttpServletRequest request, HttpServletResponse response, BlockException e) throws Exception {

response.setStatus(429);

PrintWriter out = response.getWriter();

out.print("Blocked by Sentinel (flow limiting)");

out.flush();

out.close();

}

}

* 1. 自定义限流处理逻辑

自定义限流处理逻辑可以有两种实现方式

方式一：新增 BlockExceptionHandler 实现类覆盖默认的 CustomUrlBlockHandler

方式二：使用 @SentinelResource 注解的 blockHandler 和 blockHandlerClass 两个参数指定限流处理逻辑方法

* + 1. 方式一，覆盖默认处理类 DefaultBlockExceptionHandler 实现自定义限流处理逻辑

@Component

public class MyBlockExceptionHandler implements BlockExceptionHandler {

@Override

public void handle(HttpServletRequest request, HttpServletResponse response, BlockException e) throws Exception {

String msg = null;

if (e instanceof FlowException) {

msg = "限流了,请稍后访问";

} else if (e instanceof DegradeException) {

msg = "降级了，返回默认数据";

} else if (e instanceof ParamFlowException) {

msg = "热点参数限流";

} else if (e instanceof SystemBlockException) {

msg = "系统规则（负载/...不满足要求）";

} else if (e instanceof AuthorityException) {

msg = "授权规则不通过";

}

// http状态码

response.setStatus(500);

response.setCharacterEncoding("utf-8");

response.setHeader("Content-Type", "application/json;charset=utf-8");

response.setContentType("application/json;charset=utf-8");

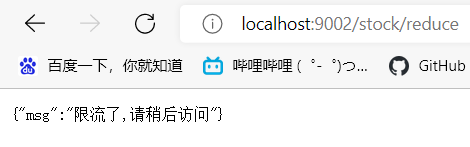
JSONObject jsonObject = new JSONObject();

jsonObject.put("msg", msg);

response.getWriter().write(jsonObject);

}

}



* + 1. 方式二，使用 @SentinelResource 注解实现自定义限流处理逻辑

@SentinelResource 注解参数中有两个比较重要，value和blockHandler。



示例代码：

@RequestMapping(value="/flow")

@SentinelResource(value = "flow", blockHandler = "flowHandleException")

public String flow() {

String rtn = "流控接口" + new Date().getTime();

System.out.println(rtn);

return rtn;

}

public String flowHandleException(BlockException ex) {

System.out.println("Oops: " + ex.getClass().getCanonicalName());

JSONObject jsonObject = new JSONObject();

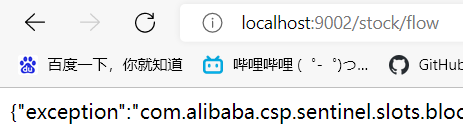
jsonObject.put("param", "null");

jsonObject.put("message", ex.getMessage());

jsonObject.put("exception", ex.getClass().getCanonicalName());

return JSONObject.toJSONString(jsonObject);

}



同时配置方式一与方式二时，方式二生效。

1. 熔断降级配置
   1. 熔断降级说明

Sentinel 提供以下几种熔断策略

1. 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。
2. 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%。
3. 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。

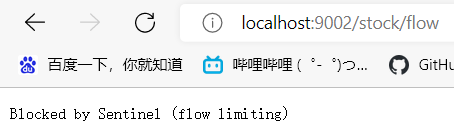


* 1. 慢调用比例



flow 慢调用比例配置说明：统计 1000 ms 内，如果请求 stock/flow 接口（资源名对应）次数大于 1时，如果 90% 响应时长超过 1000 ms，则后续请求熔断 10 s，经过10s 熔断，如果下一个请求依然响应时长超过 1000 ms，则再熔断 10s。

当满足熔断配置规则后，如果没有方式一与方式二配置，则显示如下，否则按照方式一、方式二显示



* 1. 异常比例



flow 异常比例配置说明：统计 1000 ms 内，如果请求 stock/flow 接口（资源名对应）次数大于 1时，如果超过 20% 请求抛出异常，则后续请求熔断 10 s，经过10s 熔断，如果下一个请求依然失败抛出异常，则再熔断 10s

* 1. 异常数



flow 异常比例配置说明：统计 1000 ms 内，如果请求 stock/flow 接口（资源名对应）次数大于 2 时，如果抛出异常数大于 2，则后续请求熔断 10 s，经过10s 熔断，如果下一个请求依然失败抛出异常，则再熔断 10s