

## Spring Cloud

- 1 微服务是什么
- 2 Spring Cloud是什么
- 3 Spring Cloud Eureka
- 4 Spring Cloud Ribbon
- 5 Spring Cloud OpenFeign
- 6 Spring Cloud Hystrix
- 7 Spring Cloud Gateway
- 8 Spring Cloud Config
- 9 Spring Cloud Alibaba是什么
- 10 Spring Cloud Alibaba Nacos
- 11 **Spring Cloud Alibaba Sentinel**
- 12 Spring Cloud Alibaba Seata

首页 > Spring Cloud

## Sentinel: Spring Cloud Alibaba高可用流量控制组件 (非常详细)

< 上一节

下一节 >

Sentinel 是由阿里巴巴中间件团队开发的开源项目，是一种面向分布式微服务架构的轻量级高可用流量控制组件。

Sentinel 主要以流量为切入点，从流量控制、熔断降级、系统负载保护等多个维度帮助用户保护服务的稳定性。

Sentinel 具有以下优势：

- **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（将突发流量控制在系统可以承受的范围）、消息削峰填谷、集群流量控制、实时熔断下游不可用服务等。
- **完备的实时监控**: Sentinel 提供了实时监控功能。用户可以在控制台中看到接入应用的单台机器的秒级数据，甚至是 500 台以下规模集群的汇总运行情况。
- **广泛的开源生态**: Sentinel 提供了开箱即用的与其它开源框架或库（例如 Spring Cloud、Apache Dubbo、gRPC、Quarkus）的整合模块。我们只要在项目中引入相应的依赖并进行简单的配置即可快速地接入 Sentinel。此外，Sentinel 还提供 Java、Go 以及 C++ 等多语言的原生实现。
- **完善的 SPI 扩展机制**: Sentinel 提供简单易、完善的 SPI 扩展接口，我们可以通过实现这些扩展接口快速地定制逻辑，例如定制规则管理、适配动态数据源等。

SPI，全称为 Service Provider Interface，是一种服务发现机制。它可以在 ClassPath 路径下的 META-INF/services 文件夹查找文件，并自动加载文件中定义的类。

从功能上来说，Sentinel 与 Spring Cloud Netflix Hystrix 类似，但 Sentinel 要比 Hystrix 更加强大，例如 Sentinel 提供了流量控制功能、比 Hystrix 更加完善的实时监控功能等等。

## Sentinel 的组成

Sentinel 主要由以下两个部分组成：

- **Sentinel 核心库**: Sentinel 的核心库不依赖任何框架或库，能够运行于 Java 8 及以上的版本的运行时环境中，同时对 Spring Cloud、Dubbo 等微服务框架提供了很好的支持。
- **Sentinel 控制台 (Dashboard)** : Sentinel 提供的一个轻量级的开源控制台，它为用户提供了机器自发现、簇点链路自发现、监控、规则配置等功能。

Sentinel 核心库不依赖 Sentinel Dashboard，但两者结合使用可以有效的提高效率，让 Sentinel 发挥它最大的作用。

## Sentinel 的基本概念

Sentinel 的基本概念有两个，它们分别是：资源和规则。

基本概念	描述
资源	资源是 Sentinel 的关键概念。它可以是 Java 应用程序中的任何内容，例如由应用程序提供的服务或者是服务里的方法，甚至可以是一段代码。



我们可以通过 Sentinel 提供的 API 来定义一个资源，使其能够被 Sentinel 保护起来。通常情况下，我们可以使用方法名、URL 甚至是服务名来作为资源名来描述某个资源。

规则	围绕资源而设定的规则。Sentinel 支持流量控制、熔断降级、系统保护、来源访问控制和热点参数等多种规则，所有这些规则都可以动态实时调整。
----	--

## @SentinelResource 注解

@SentinelResource 注解是 Sentinel 提供的最重要的注解之一，它还包含了多个属性，如下表。

属性	说明	必填与否	使用要求
value	用于指定资源的名称	必填	-
entryType	entry 类型	可选项 (默认为 EntryType.OUT)	-
blockHandler	服务限流后会抛出 BlockException 异常，而 blockHandler 则是用来指定一个函数来处理 BlockException 异常的。  简单点说，该属性用于指定服务限流后的后续处理逻辑。	可选项	<ul style="list-style-type: none"><li>blockHandler 函数访问范围需要是 public；</li><li>返回类型需要与原方法相匹配；</li><li>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；</li><li>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 blockHandler 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。</li></ul>
blockHandlerClass	若 blockHandler 函数与原方法不在同一个类中，则需要使用该属性指定 blockHandler 函数所在的类。	可选项	<ul style="list-style-type: none"><li>不能单独使用，必须与 blockHandler 属性配合使用；</li><li>该属性指定的类中的 blockHandler 函数必须为 static 函数，否则无法解析。</li></ul>
fallback	用于在抛出异常（包括 BlockException）时，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。	可选项	<ul style="list-style-type: none"><li>返回值类型必须与原函数返回值类型一致；</li><li>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；</li><li>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。</li></ul>
fallbackClass	若 fallback 函数与原方法不在同一个类中，则需要使用该属性指定 blockHandler 函数所在的类。	可选项	<ul style="list-style-type: none"><li>不能单独使用，必须与 fallback 或 defaultFallback 属性配合使用；</li><li>该属性指定的类中的 fallback 函数必须为 static 函数，否则无法解析。</li></ul>
defaultFallback	默认的 fallback 函数名称，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 函数可以针对所以类型的异	可选项	<ul style="list-style-type: none"><li>返回值类型必须与原函数返回值类型一致；</li><li>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；</li><li>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</li></ul>



	常（除了 exceptionsToIgnore 里面排除掉的异常类型）进行处理。		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	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。	可选项	-

注：在 Sentinel 1.6.0 之前，fallback 函数只针对降级异常（DegradeException）进行处理，不能处理业务异常。

## Sentinel 控制台

Sentinel 提供了一个轻量级的开源控制台 Sentinel Dashboard，它提供了机器发现与健康情况管理、监控（单机和集群）、规则管理与推送等多种功能。

Sentinel 控制台提供的功能如下：

- 查看机器列表以及健康情况：** Sentinel 控制台能够收集 Sentinel 客户端发送的心跳包，判断机器是否在线。
- 监控（单机和集群聚合）：** Sentinel 控制台通过 Sentinel 客户端暴露的监控 API，可以实现秒级的实时监控。
- 规则管理和推送：** 通过 Sentinel 控制台，我们还能够针对资源定义和推送规则。
- 鉴权：** 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。

Sentinel Dashboard 是我们配置和管理规则（例如流控规则、熔断降级规则等）的重要入口之一。通过它，我们不仅可以对规则进行配置和管理，还能实时查看规则的效果。

### 安装 Sentinel 控制台

下面我们就来演示下，如何下载和安装 Sentinel 控制台，具体步骤如下。

1. 使用浏览器访问 [Sentinel Dashboard 下载页面](#) 下载 Sentinel 控制台的 jar 包，如下图。

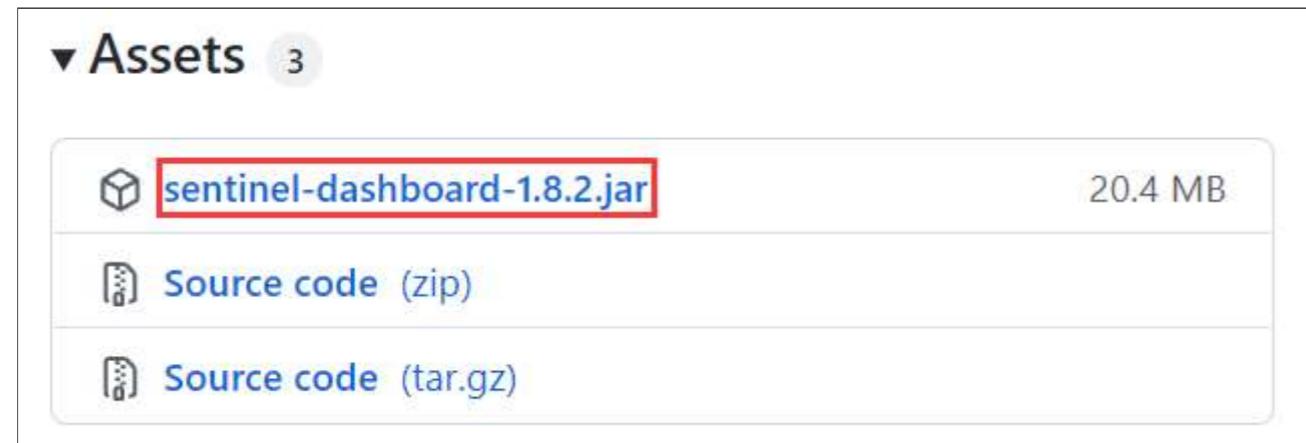


图1：Sentinel 控制台下载

2. 打开命令行窗口，跳转到 Sentinel Dashboard jar 包所在的目录，执行以下命令，启动 Sentinel Dashboard。

```
java -jar sentinel-dashboard-1.8.2.jar
```

3. 启动完成后，使用浏览器访问 “<http://localhost:8080/>” ，跳转到 Sentinel 控制台登陆页面，如下图。





图2: Sentinel 控制台登录页

4. 分别输入用户名和密码（默认都是 sentinel），点击下方的登录按钮，结果如下图。

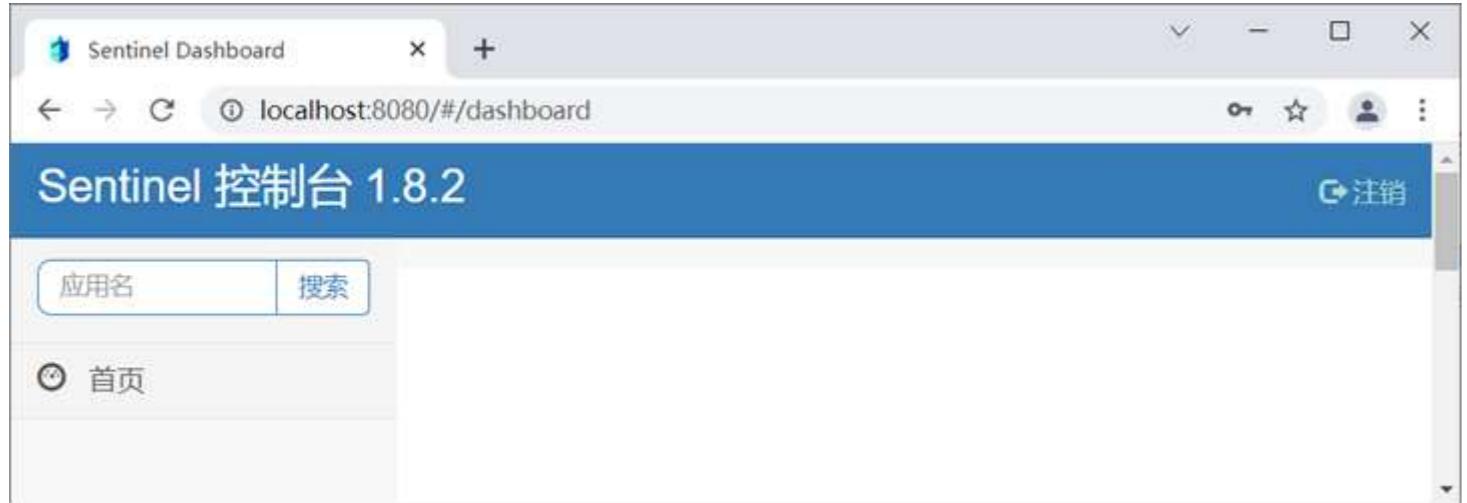


图3: Sentinel 控制台主页

## Sentinel 的开发流程

Sentinel 的开发流程如下：

1. **引入 Sentinel 依赖**: 在项目中引入 Sentinel 的依赖，将 Sentinel 整合到项目中；
2. **定义资源**: 通过对主流框架提供适配或 Sentinel 提供的显式 API 和注解，可以定义需要保护的资源，此外 Sentinel 还提供了资源的实时统计和调用链路分析；
3. **定义规则**: 根据实时统计信息，对资源定义规则，例如流控规则、熔断规则、热点规则、系统规则以及授权规则等。
4. **检验规则是否生效**: 运行程序，检验规则是否生效，查看效果。

## 引入 Sentinel 依赖

为了减少开发的复杂程度，Sentinel 对大部分的主流框架都进行了适配，例如 Web Servlet、Dubbo、Spring Cloud、gRPC、Spring WebFlux 和 Reactor 等。以 Spring Cloud 为例，我们只需要引入 `spring-cloud-starter-alibaba-sentinel` 的依赖，就可以方便地将 Sentinel 整合到项目中。



下面我们就通过一个简单的实例，演示如何将 Sentinel 整合到 Spring Cloud 项目中，步骤如下。

1. 在主工程 spring-cloud-alibaba-demo 下，创建一个名为 spring-cloud-alibaba-sentinel-service-8401 的 Spring Boot 模块，并在其 pom.xml 中添加以下依赖。

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
04.     <modelVersion>4.0.0</modelVersion>
05.     <parent>
06.         <groupId>net.biancheng.c</groupId>
07.         <version>1.0-SNAPSHOT</version>
08.         <artifactId>spring-cloud-alibaba-demo</artifactId>
09.     </parent>
10.
11.     <groupId>net.biancheng.c</groupId>
12.     <artifactId>spring-cloud-alibaba-sentinel-service-8401</artifactId>
13.     <version>0.0.1-SNAPSHOT</version>
14.     <name>spring-cloud-alibaba-sentinel-service-8401</name>
15.     <description>Demo project for Spring Boot</description>
16.     <properties>
17.         <java.version>1.8</java.version>
18.     </properties>
19.
20.     <dependencies>
21.         <!--Nacos 服务发现依赖-->
22.         <dependency>
23.             <groupId>com.alibaba.cloud</groupId>
24.             <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
25.         </dependency>
26.         <!--Sentinel 依赖-->
27.         <dependency>
28.             <groupId>com.alibaba.cloud</groupId>
29.             <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
30.         </dependency>
31.         <!--SpringCloud alibaba sentinel-datasource-nacos 后续做持久化用到-->
32.         <dependency>
33.             <groupId>com.alibaba.csp</groupId>
34.             <artifactId>sentinel-datasource-nacos</artifactId>
35.         </dependency>
36.
37.         <dependency>
38.             <groupId>org.springframework.boot</groupId>
39.             <artifactId>spring-boot-starter-actuator</artifactId>
40.         </dependency>
41.         <dependency>
42.             <groupId>org.springframework.boot</groupId>
```



```

43.          <artifactId>spring-boot-starter-web</artifactId>
44.      </dependency>
45.      <dependency>
46.          <groupId>org.springframework.cloud</groupId>
47.          <artifactId>spring-cloud-starter-openfeign</artifactId>
48.      </dependency>
49.      <dependency>
50.          <groupId>org.springframework.boot</groupId>
51.          <artifactId>spring-boot-devtools</artifactId>
52.          <scope>runtime</scope>
53.          <optional>true</optional>
54.      </dependency>
55.      <dependency>
56.          <groupId>org.projectlombok</groupId>
57.          <artifactId>lombok</artifactId>
58.          <optional>true</optional>
59.      </dependency>
60.      <dependency>
61.          <groupId>org.springframework.boot</groupId>
62.          <artifactId>spring-boot-starter-test</artifactId>
63.          <scope>test</scope>
64.      </dependency>
65.  </dependencies>
66.
67.  <build>
68.      <plugins>
69.          <plugin>
70.              <groupId>org.springframework.boot</groupId>
71.              <artifactId>spring-boot-maven-plugin</artifactId>
72.              <configuration>
73.                  <excludes>
74.                      <exclude>
75.                          <groupId>org.projectlombok</groupId>
76.                          <artifactId>lombok</artifactId>
77.                      </exclude>
78.                  </excludes>
79.              </configuration>
80.          </plugin>
81.      </plugins>
82.  </build>
83. </project>

```

2. 在 spring-cloud-alibaba-sentinel-service-8401 的类路径下，新建一个配置文件 application.yml，配置内容如下。

```

01. server:
02.     port: 8401 #端口

```



```
03.  
04. spring:  
05.   application:  
06.     name: sentinel-service #服务名  
07.   cloud:  
08.     nacos:  
09.       discovery:  
10.         #Nacos服务注册中心(集群)地址  
11.         server-addr: localhost:1111  
12.  
13.   sentinel:  
14.     transport:  
15.       #配置 Sentinel dashboard 地址  
16.       dashboard: localhost:8080  
17.       #默认8719端口, 假如被占用会自动从8719开始依次+1扫描, 直至找到未被占用的端口  
18.       port: 8719  
19.  
20. management:  
21.   endpoints:  
22.     web:  
23.       exposure:  
24.         include: '*'
```

3. 在 net.biancheng.c.controller 下，创建一个名为 SentinelFlowLimitController 的 Controller 类，代码如下。

```
01. package net.biancheng.c.controller;  
02.  
03. import lombok.extern.slf4j.Slf4j;  
04. import org.springframework.beans.factory.annotation.Value;  
05. import org.springframework.web.bind.annotation.GetMapping;  
06. import org.springframework.web.bind.annotation.RestController;  
07.  
08. @RestController  
09. @Slf4j  
10. public class SentinelFlowLimitController {  
11.  
12.     @Value("${server.port}")  
13.     private String serverPort;  
14.  
15.     @GetMapping("/testA")  
16.     public String testA() {  
17.         return "c语言中文网提醒您, 服务访问成功-----testA";  
18.     }  
19.  
20.     @GetMapping("/testB")  
21.     public String testB() {
```



```
22.         return "c语言中文网提醒您，服务访问成功-----testB"
23.     }
24. }
```

4. spring-cloud-alibaba-sentinel-service-8401 主启动类代码如下。

```
01. package net.biancheng.c;
02.
03. import org.springframework.boot.SpringApplication;
04. import org.springframework.boot.autoconfigure.SpringBootApplication;
05. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
06.
07. @SpringBootApplication
08. @EnableDiscoveryClient
09. public class SpringCloudAlibabaSentinelService8401Application {
10.
11.     public static void main(String[] args) {
12.         SpringApplication.run(SpringCloudAlibabaSentinelService8401Application.class, args);
13.     }
14.
15. }
```

5. 依次启动 Nacos Server 集群、spring-cloud-alibaba-sentinel-service-8401，使用浏览器访问“<http://localhost:8401/testA>”，结果如下图。



图4: Sentinel 示例 1

6. 使用浏览器访问 Sentinel 控制台主页，我们发现在“首页”下方新增了一个“sentinel-service”的菜单，而这正是 spring-cloud-alibaba-sentinel-service-8401 的服务名 (spring.application.name)，说明 Sentinel 已经监控到这个服务，如下图。

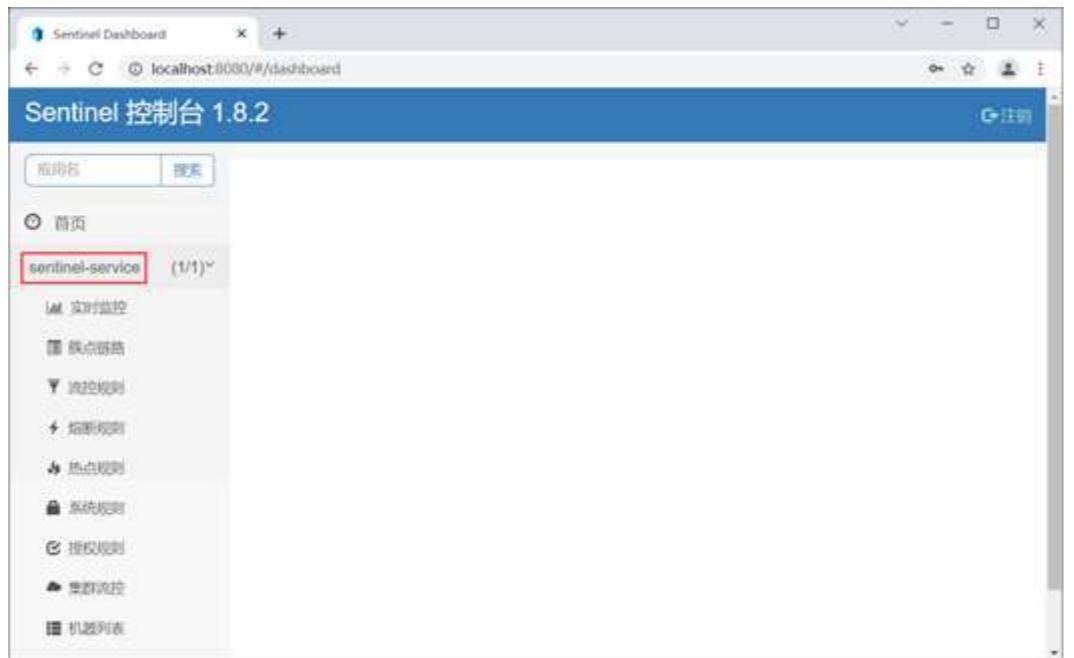


图5: Sentinel 控制台主页

7. 点击“实时监控”，查看 sentinel-service 下各请求的实时监控数据，如下图所示。

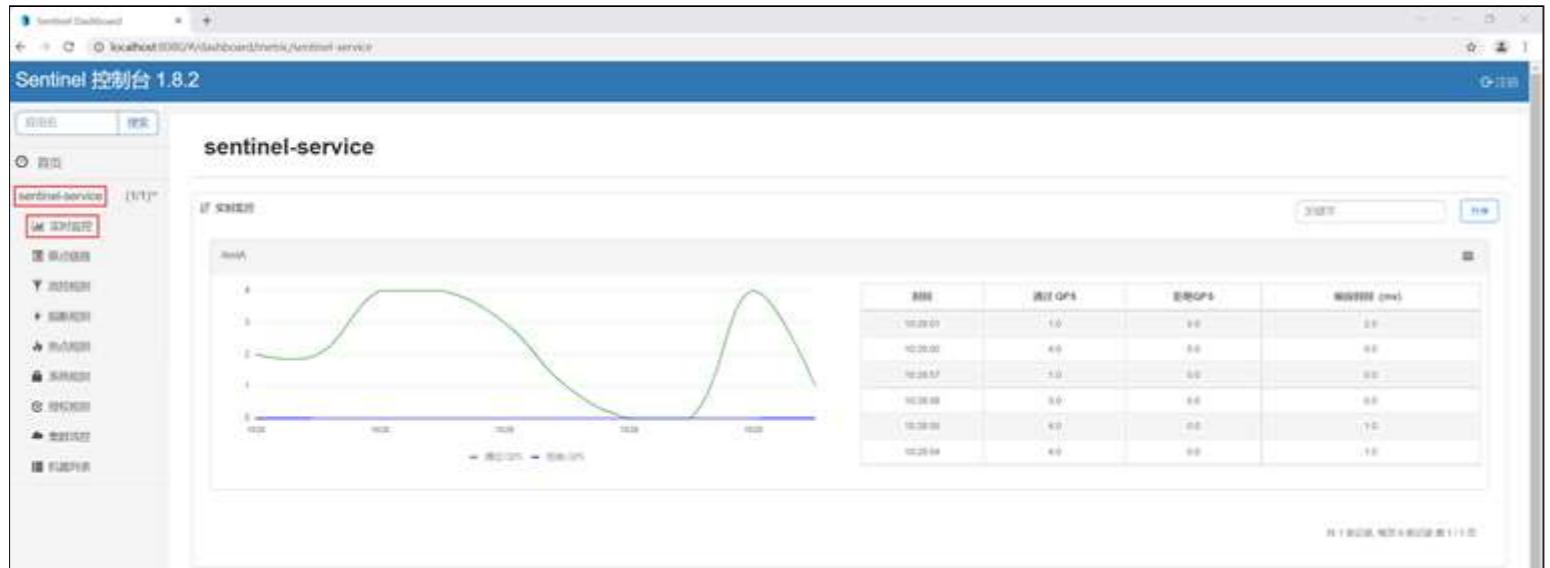


图6: Sentinel 实时监控

## 定义资源

资源是 Sentinel 中的核心概念之一。在项目开发时，我们只需要考虑这个服务、方法或代码是否需要保护，如果需要保护，就可以将它定义为一个资源。

Sentinel 为我们提供了多种定义资源的方式：

- 适配主流框架自动定义资源
- 通过 SpfU 手动定义资源
- 通过 SpfO 手动定义资源
- 注解方式定义资源

适配主流框架自动定义资源



Sentinel 对大部分的主流框架都进行了适配，我们只要引入相关的适配模块（例如 spring-cloud-starter-alibaba-sentinel），Sentinel 就会自动将项目中的服务（包括调用端和服务端）定义为资源，资源名就是服务的请求路径。此时，我们只要再定义一些规则，这些资源就可以享受到 Sentinel 的保护。

我们可以在 Sentinel 控制台的“簇点链路”中，直接查看被 Sentinel 监控的资源，如下图。

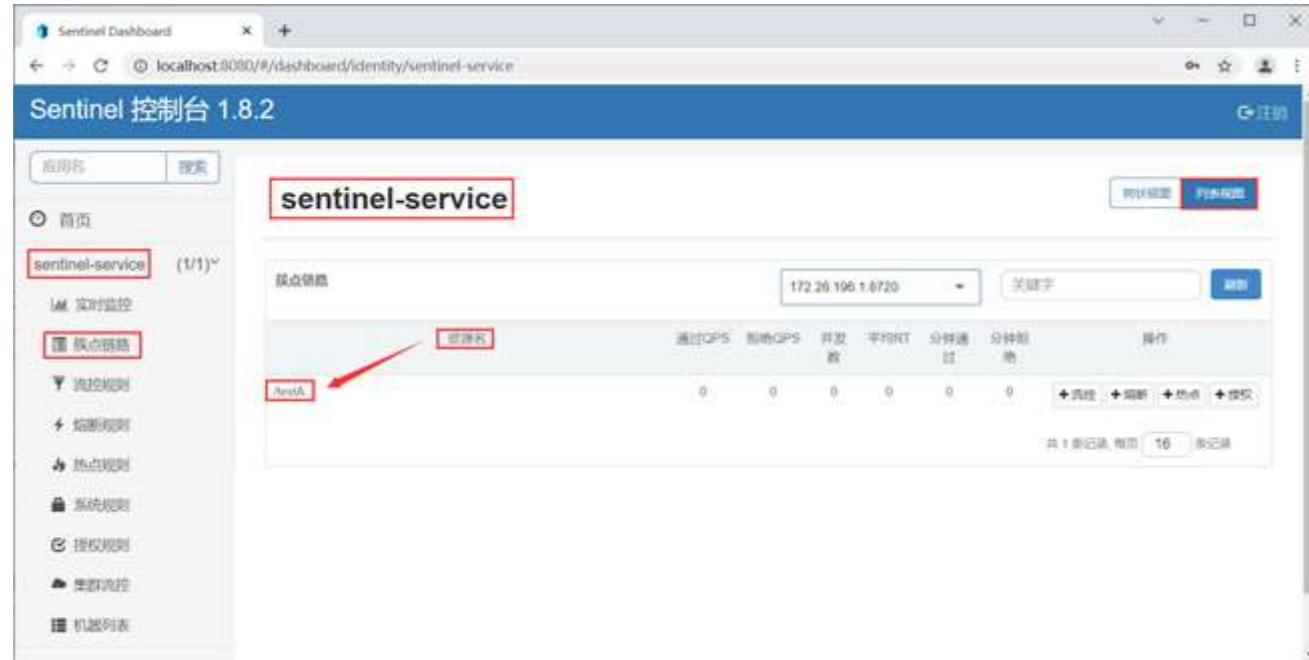


图7：Sentinel 控制台-簇点链路

### 通过 SphU 手动定义资源

Sentinel 提供了一个名为 SphU 的类，它包含的 try-catch 风格的 API，可以帮助我们手动定义资源。

下面我们就通过一个实例，来演示下如何通过 SphU 定义资源。

1. 在 spring-cloud-alibaba-sentinel-service-8401 下的 SentinelFlowLimitController 中，新增一个 testAbySphU() 方法定义一个名为 testAbySphU 的资源，代码如下。

```
01. package net.biancheng.c.controller;
02.
03. import com.alibaba.csp.sentinel.Entry;
04. import com.alibaba.csp.sentinel.Sph0;
05. import com.alibaba.csp.sentinel.SphU;
06. import com.alibaba.csp.sentinel.slots.block.BlockException;
07. import com.alibaba.csp.sentinel.slots.block.RuleConstant;
08. import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
09. import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
10. import lombok.extern.slf4j.Slf4j;
11. import org.springframework.beans.factory.annotation.Value;
12. import org.springframework.web.bind.annotation.GetMapping;
13. import org.springframework.web.bind.annotation.RestController;
14.
15. import java.util.ArrayList;
16. import java.util.List;
17.
```



```
18. @RestController
19. @Slf4j
20. public class SentinelFlowLimitController {
21.
22.     @Value("${server.port}")
23.     private String serverPort;
24.
25.     @GetMapping("/testA")
26.     public String testA() {
27.         return testAbySphU();
28.     }
29.
30.     @GetMapping("/testB")
31.     public String testB() {
32.         return "c语言中文网提醒您，服务访问成功-----testB";
33.     }
34.
35.
36.     /**
37.      * 通过 SphU 手动定义资源
38.      * @return
39.      */
40.     public String testAbySphU() {
41.         Entry entry = null;
42.         try {
43.             entry = SphU.entry("testAbySphU");
44.             //您的业务逻辑 - 开始
45.             log.info("c语言中文网提醒您，服务访问成功-----testA: "+serverPort);
46.             return "c语言中文网提醒您，服务访问成功-----testA: "+serverPort;
47.             //您的业务逻辑 - 结束
48.         } catch (BlockException e1) {
49.             //流控逻辑处理 - 开始
50.             log.info("c语言中文网提醒您，testA 服务被限流");
51.             return "c语言中文网提醒您，testA 服务被限流";
52.             //流控逻辑处理 - 结束
53.         } finally {
54.             if (entry != null) {
55.                 entry.exit();
56.             }
57.         }
58.     }
59. }
```

2. 重启 spring-cloud-alibaba-sentinel-service-8401，使用浏览器访问 “<http://localhost:8401/testA>” , 结果如下。

c语言中文网提醒您，服务访问成功-----testA: 8401



3. 访问 Sentinel 控制台主页，点击 sentinel-service 下的“簇点链路”，结果如下图。

The screenshot shows the Sentinel Dashboard interface for version 1.8.2. The main title is "Sentinel 控制台 1.8.2". On the left sidebar, there are several menu items: 首页 (Home), 实时监控 (Real-time Monitoring) (highlighted with a red box), 簇点链路 (Cluster Point Link) (highlighted with a red box), 流控规则 (Flow Control Rules), 热点规则 (Hotspot Rules), 系统规则 (System Rules), 权限规则 (Permission Rules), 集群监控 (Cluster Monitoring), and 机器列表 (Machine List). The central area is titled "sentinel-service" and shows a table for "簇点链路" (Cluster Point Link) with one entry: "testAblySent1". The table includes columns for "资源名" (Resource Name), "通过QPS" (Throughput QPS), "剩余QPS" (Remaining QPS), "并发数" (Concurrent Requests), "平均RT" (Average RT), "分钟通过量" (Throughput per minute), and "分钟拒绝量" (Rejected requests per minute). At the bottom right of the table, there are four buttons: "添加" (Add), "编辑" (Edit), "删除" (Delete), and "授权" (Authorize). The URL in the browser address bar is "localhost:8080/#/dashboard/identity/sentinel-service".

图8: Sentinel 通过 SphU 定义资源

#### 通过 SphO 手动定义资源

Sentinel 还提供了一个名为 SphO 的类，它包含了 if-else 风格的 API，能帮助我们手动定义资源。通过这种方式定义的资源，发生了限流之后会返回 false，此时我们可以根据返回值，进行限流之后的逻辑处理。

下面我们就通过一个实例，来演示下如何通过 SphO 定义资源。

1. 在 spring-cloud-alibaba-sentinel-service-8401 下的 SentinelFlowLimitController 中，新增一个 testBbySphO() 方法定义一个名为 testBbySphO 的资源，代码如下。

```
01. package net.biancheng.c.controller;
02.
03. import com.alibaba.csp.sentinel.Entry;
04. import com.alibaba.csp.sentinel.SphO;
05. import com.alibaba.csp.sentinel.SphU;
06. import com.alibaba.csp.sentinel.slots.block.BlockException;
07. import com.alibaba.csp.sentinel.slots.block.RuleConstant;
08. import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
09. import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
10. import lombok.extern.slf4j.Slf4j;
11. import org.springframework.beans.factory.annotation.Value;
12. import org.springframework.web.bind.annotation.GetMapping;
13. import org.springframework.web.bind.annotation.RestController;
14.
15. import java.util.ArrayList;
16. import java.util.List;
17.
18. @RestController
```



```
19. @Slf4j
20. public class SentinelFlowLimitController {
21.
22.     @Value("${server.port}")
23.     private String serverPort;
24.
25.
26.     @GetMapping("/testA")
27.     public String testA() {
28.         return testAbySphU();
29.     }
30.
31.     @GetMapping("/testB")
32.     public String testB() {
33.         return testBbySph0();
34.     }
35.
36.     /**
37.      * 通过 SphU 手动定义资源
38.      *
39.      * @return
40.      */
41.     public String testAbySphU() {
42.         Entry entry = null;
43.         try {
44.             entry = SphU.entry("testAbySphU");
45.             //您的业务逻辑 - 开始
46.             log.info("c语言中文网提醒您，服务访问成功-----testA: " + serverPort);
47.             return "c语言中文网提醒您，服务访问成功-----testA: " + serverPort;
48.             //您的业务逻辑 - 结束
49.         } catch (BlockException e1) {
50.             //流控逻辑处理 - 开始
51.             log.info("c语言中文网提醒您，testA 服务被限流");
52.             return "c语言中文网提醒您，testA 服务被限流";
53.             //流控逻辑处理 - 结束
54.         } finally {
55.             if (entry != null) {
56.                 entry.exit();
57.             }
58.         }
59.     }
60.
61.     /**
62.      * 通过 Sph0 手动定义资源
63.      *
64.      * @return
```



```

65. */
66. public String testBbySph0() {
67.     if (Sph0.entry("testBbySph0")) {
68.         // 务必保证finally会被执行
69.         try {
70.             log.info("c语言中文网提醒您，服务访问成功-----testB: " + serverPort);
71.             return "c语言中文网提醒您，服务访问成功-----testB: " + serverPort;
72.         } finally {
73.             Sph0.exit();
74.         }
75.     } else {
76.         // 资源访问阻止，被限流或被降级
77.         // 流控逻辑处理 - 开始
78.         log.info("c语言中文网提醒您，testB 服务被限流");
79.         return "c语言中文网提醒您，testB 服务被限流";
80.         // 流控逻辑处理 - 结束
81.     }
82. }
83. }

```

2. 重启 spring-cloud-alibaba-sentinel-service-8401，使用浏览器访问 “<http://localhost:8401/testB>” ，结果如下。

c语言中文网提醒您，服务访问成功-----testB: 8401

3. 访问 Sentinel 控制台主页，点击 sentinel-service 下的“簇点链路”，结果如下图。

The screenshot shows the Sentinel Dashboard interface. The main title is 'Sentinel 控制台 1.8.2'. On the left, there is a sidebar with several tabs: '应用名' (Application Name), '搜索' (Search), '首页' (Home), 'sentinel-service' (selected), '实例监控' (Instance Monitoring), '簇点链路' (Cluster Path) (highlighted with a red box), '流控规则' (Flow Control Rules), '熔断规则' (Circuit Breaker Rules), '热力规则' (Heatmap Rules), '系统规则' (System Rules), '授权规则' (Authorization Rules), '集群监控' (Cluster Monitoring), and '机器列表' (Machine List). The central area is titled 'sentinel-service' and shows a table for '簇点链路' (Cluster Path). The table has columns: '连接名' (Connection Name), '通过QPS' (Throughput QPS), '拒绝QPS' (Rejected QPS), '并发数' (Concurrent Requests), '平均RPS' (Average RPS), '分钟峰值' (Peak per minute), and '操作' (Operations). There are four entries in the table:

连接名	通过QPS S	拒绝QPS S	并发数 T	平均RPS RPS	分钟峰值 MPS	操作
sentAblySph0	0	0	0	0	0	<span>+ 追加</span> <span>+ 增删</span> <span>+ 热点</span> <span>+ 授权</span>
RestA	0	0	0	0	0	<span>+ 流控</span> <span>+ 增删</span> <span>+ 热点</span> <span>+ 授权</span>
RestB	0	0	0	0	0	<span>+ 流控</span> <span>+ 增删</span> <span>+ 热点</span> <span>+ 授权</span>
sentilitySph0	0	0	0	0	0	<span>+ 流控</span> <span>+ 增删</span> <span>+ 热点</span> <span>+ 授权</span>

图9: Sentinel 通过 Sph0 定义资源

除了以上两种方式外，我们还可以通过 Sentinel 提供的 @SentinelResource 注解定义资源。

下面我们就通过一个实例，来演示下如何通过 @SentinelResource 注解定义资源。

1. 将 spring-cloud-alibaba-sentinel-service-8401 中 SentinelFlowLimitController 类中增加以下代码。

```
01. @GetMapping("/testC")
02. @SentinelResource(value = "testCbyAnnotation") //通过注解定义资源
03. public String testC() {
04.     log.info("c语言中文网提醒您，服务访问成功-----testC: " + serverPort);
05.     return "c语言中文网提醒您，服务访问成功-----testC: " + serverPort;
06. }
```

2. 重启 spring-cloud-alibaba-sentinel-service-8401，使用浏览器访问 “<http://localhost:8401/testC>” ，结果如下。

c语言中文网提醒您，服务访问成功-----testC: 8401

3. 访问 Sentinel 控制台主页，点击 sentinel-service 下的“簇点链路”，结果如下图。

The screenshot shows the Sentinel Dashboard interface. On the left, there's a sidebar with navigation items like '实时监控', '簇点链路' (highlighted with a red box), '流控规则', '熔断规则', '热点规则', '系统规则', '授权规则', '集群监控', and '机架列表'. The main content area has a title 'sentinel-service' and a sub-section 'sentinel-service (1/1)'. Under '簇点链路', there's a table with columns: 簇点名, 通过QPS, 堆栈QPS, 并发数, 平均RPS, 分钟通过量, 分钟拒绝量, and 操作. The table lists several entries, including 'testAbySpEL', 'testC', 'testA', 'testB', 'testCbyAnnotation', and 'testBySpEL'. Each entry has a row of icons at the end: +流控, +熔断, +热点, and +授权.

图10: Sentinel 注解方式定义资源

## Sentinel 流量控制

任何系统处理请求的能力都是有限的，但任意时间内到达系统的请求量往往是随机且不可控的，如果在某一个瞬时时刻请求量急剧增，那么系统就很有可能被瞬时的流量高峰冲垮。为了避免此类情况发生，我们都需要根据系统的处理能力对请求流量进行控制，这就是我们常说的“流量控制”，简称“流控”。

Sentinel 作为一种轻量级高可用流量控制组件，流量控制是它最主要的工作之一。

我们可以针对资源定义流控规则，Sentinel 会根据这些规则对流量相关的各项指标进行监控。当这些指标当达到或超过流控规则规定的阈值时，Sentinel



会对请求的流量进行限制（即“限流”），以避免系统被瞬时的流量高峰冲垮，保障系统的高可用性。

一条流量规则主要由下表中的属性组成，我们可以通过组合这些属性来实现不同的限流效果。

属性	说明	默认值
资源名	流控规则的作用对象。	-
阈值	流控的阈值。	-
阈值类型	流控阈值的类型，包括 QPS 或并发线程数。	QPS
针对来源	流控针对的调用来源。	default，表示不区分调用来源
流控模式	调用关系限流策略，包括直接、链路和关联。	直接
流控效果	流控效果（直接拒绝、Warm Up、匀速排队），不支持按调用关系限流。	直接拒绝

注：QPS 表示并发请求数，换句话说就是，每秒钟最多通过的请求数。

同一个资源可以创建多条流控规则，Sentinel 会遍历这些规则，直到有规则触发限流或者所有规则遍历完毕为止。

Sentinel 触发限流时，资源会抛出 BlockException 异常，此时我们可以捕捉 BlockException 来自定义被限流之后的处理逻辑。

注意：这里我们主要讲解 Sentinel 流控规则的定义与使用，至于详细的流控规则配置，请参考 [Sentinel 官方流控文档](#)。

### 通过 Sentinel 控制台定义流控规则

我们可以通过 Sentinel 控制台，直接对资源定义流控规则，操作步骤如下。

1. 在 spring-cloud-alibaba-sentinel-service-8401 的 SentinelFlowLimitController 中新增一个名为 testD 的服务方法，代码如下。

```
01. /**
02. * 通过 Sentinel 控制台定义流控规则
03. *
04. * @return
05. */
06. @GetMapping("/testD")
07. public String testD() {
08.     log.info("c语言中文网提醒您，服务访问成功-----testD: " + serverPort);
09.     return "c语言中文网提醒您，服务访问成功-----testD: " + serverPort;
10. }
```

2. 重启 spring-cloud-alibaba-sentinel-service-8401，使用浏览器访问 “http://localhost:8401/testD”，结果如下。

c语言中文网提醒您，服务访问成功-----testD: 8401

3. 使用浏览器访问 “http://localhost:8080”，登陆 Sentinel 控制台主页，点击 sentinel-service 下的“簇点链路”，结果如下图。



The screenshot shows the Sentinel Dashboard interface for version 1.8.2. The left sidebar lists various service configurations, with 'sentinel-service' selected. The main content area displays the 'sentinel-service' details, specifically the '流点链路' (Flow Point Path) section. This section includes a table with columns: 资源名 (Resource Name), 通过QPS (Throughput QPS), 拒绝QPS (Rejected QPS), 并发数 (Concurrent Requests), 平均RT (Average RT), 分钟通过 (Throughput per minute), 分钟拒绝 (Rejected per minute), and 操作 (Operations). Three rows are listed: sentinel\_default\_context, sentinel\_spring\_web\_context, and testD. The 'testD' row has its 'testD' entry highlighted with a red box. To the right of the table are several operation buttons: + 流控 (Add Flow Control), + 热断 (Add Circuit Breaker), + 热点 (Add Hotspot), and + 授权 (Add Authorization). At the bottom of the table, there is a pagination bar indicating '共 9 条记录, 每页 16 条记录'.

图11：Sentinel 控制台定义流控规则

4. 点击 “/testD” 右侧的 “+流控” 按钮，在弹出的“新增流控规则”窗口中定义流控规则，如下图。





图12: Sentinel 控制台定义流控规则

5. 点击下方的“新增”按钮，跳转到“流控规则”列表，如下图。

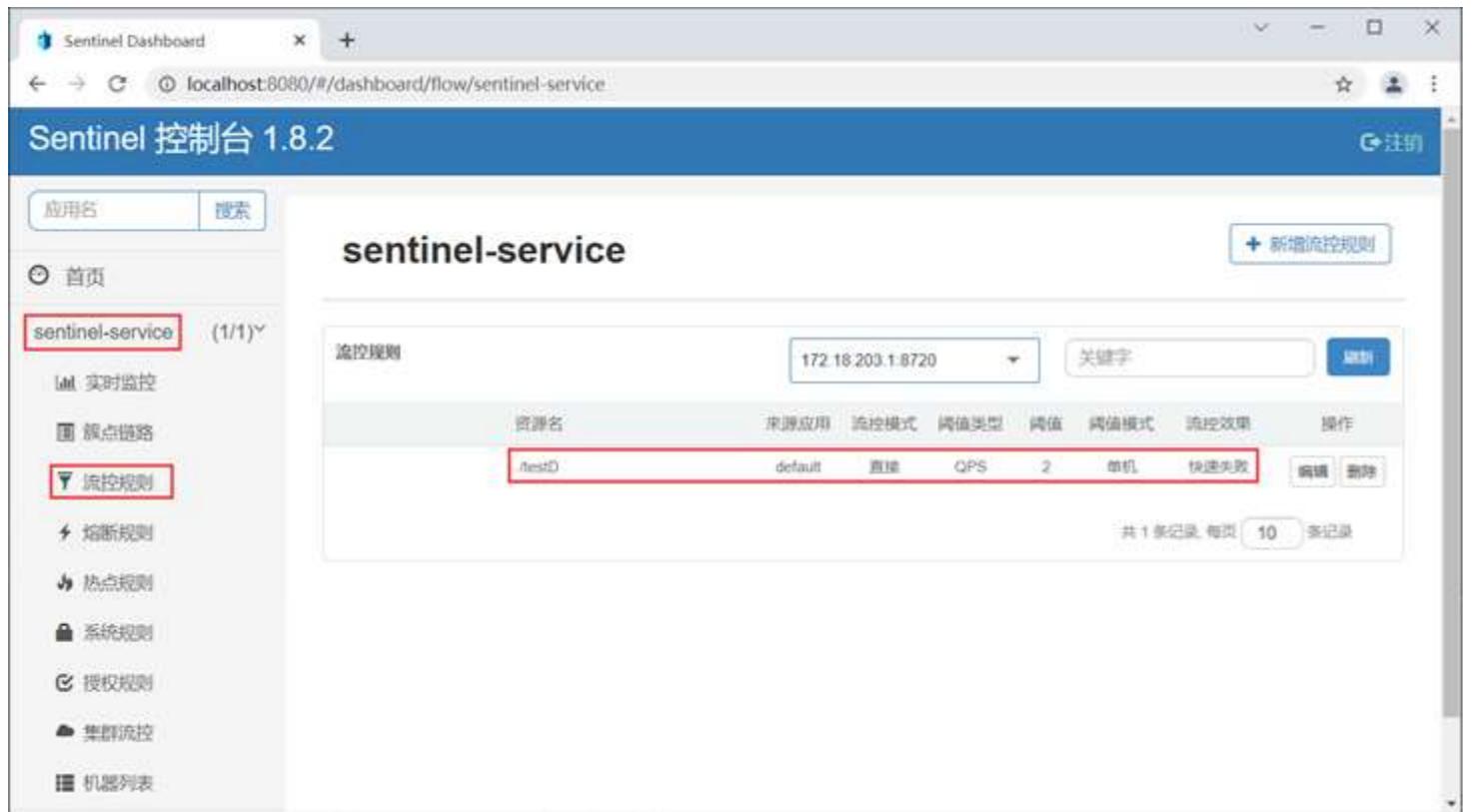


图13: Sentinel 控制台流控规则列表

6. 快速连续 (频率大于每秒钟 2 次) 访问 “<http://localhost:8401/testD>” , 结果如下。

Blocked by Sentinel (flow limiting)

若页面中出现以上信息，则说明该服务已被限流，但这种提示是 Sentinel 系统自动生成的，用户体验不好。

7. 在服务代码中使用 @SentinelResource 注解定义资源名称，并在 blockHandler 属性指定一个限流函数，自定义服务限流信息，代码如下。

```
01. /**
02. * 通过 Sentinel 控制台定义流控规则
03. *
04. */
05. @GetMapping("/testD")
06. @SentinelResource(value = "testD-resource", blockHandler = "blockHandlerTestD") //通过注解定义资源
07. public String testD() {
08.     log.info("c语言中文网提醒您，服务访问成功-----testD: " + serverPort);
09.     return "c语言中文网提醒您，服务访问成功-----testD: " + serverPort;
10. }
11.
12. /**
13. * 限流之后的逻辑
14. * @param exception
15. * @return
16. */
17. public String blockHandlerTestD(BlockException exception) {
18.     log.info(Thread.currentThread().getName() + "c语言中文网提醒您，TestD服务访问失败！您已被限流，请稍后重试");
```

```
19.     return "c语言中文网提醒您, TestD服务访问失败! 您已被限流, 请稍后重试";
20. }
```

在以上代码中，我们通过 @SentinelResource 注解的 blockHandler 属性指定了一个 blockHandler 函数，进行限流之后的后续处理。

使用 @SentinelResource 注解的 blockHandler 属性时，需要注意以下事项：

- 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 blockHandler 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。
- 请务必添加 blockHandler 属性来指定自定义的限流处理方法，若不指定，则会跳转到错误页（用户体验不好）。

7. 重启 spring-cloud-alibaba-sentinel-service-8401，使用浏览器访问 Sentinel 控制台主页，点击 sentinel-service 下的“簇点链路”，结果如下图。

The screenshot shows the Sentinel Dashboard interface. The title bar says 'Sentinel 控制台 1.8.2'. The left sidebar has a tree view with nodes like 'sentinel-service' (selected), '实时监控', '簇点链路' (highlighted with a red box), '流控规则', '熔断规则', '热点规则', '系统规则', '授权规则', '集群流控', and '机器列表'. The main content area is titled 'sentinel-service' and shows a table for '簇点链路'. The table has columns: 资源名 (Resource Name), 通过QPS (Throughput QPS), 拒绝QPS (Rejected QPS), 并发数 (Concurrent Requests), 平均RT (Average RT), 分钟通过 (Throughput per minute), 分钟拒绝 (Rejected per minute), and 操作 (Operations). There are four rows: 'sentinel\_spring\_web\_context', '/testD', 'testD-resource' (highlighted with a red box), and 'sentinel\_default\_context'. Each row has a set of buttons for '流控' (+ Throttle), '熔断' (+ Circuit Breaker), '热点' (+ Hotspot), and '授权' (+ Authorization). The bottom of the table shows '共 4 条记录, 每页 16 条记录'.

图14: Sentinel 控制台-簇点链路

4. 点击资源 “testD-resource” 右侧的 “+流控” 按钮，并在弹出的 “新增流控规则” 窗口中为这个资源定义流控规则，流控规则内容为：QPS 的阈值为 2 (即每秒最多通过 2 个请求)，如下图。



图15: Sentinel 控制台新增流控规则

5. 快速连续（频率大于每秒钟 2 次）访问 “<http://localhost:8401/testD>” , 结果如下。

c语言中文网提醒您，TestD服务访问失败！您已被限流，请稍后重试

#### 通过代码定义流控规则

我们还可以在服务代码中，调用 FlowRuleManager 类的 loadRules() 方法来定义流控规则，该方法需要一个 FlowRule 类型的 List 集合作为其参数，示例代码如下。

```
01. public static void loadRules(List<FlowRule> rules) {
02.     currentProperty.updateValue(rules);
03. }
```

FlowRule 可以通过以下属性定义流控规则，如下表。

属性	说明	默认值
resource	资源名，即流控规则的作用对象	-
count	限流的阈值。	-
grade	流控阈值的类型，包括 QPS 或并发线程数	QPS
limitApp	流控针对的调用来源	default, 表示不区分调用来源

strategy	调用关系限流策略，包括直接、链路和关联	直接
controlBehavior	流控效果（直接拒绝、Warm Up、匀速排队），不支持按调用关系限流	直接拒绝

下面我们就通过一个简单的实例，来演示下如何通过代码定义流控规则，步骤如下。

1. 在 spring-cloud-alibaba-sentinel-service-8401 的 SentinelFlowLimitController 中添加一个 initFlowRules() 方法，为名为 testD-resource 的资源定义流控规则：每秒最多只能通过 2 个请求，即 QPS 的阈值为 2。

```

01. /**
02. * 通过代码定义流量控制规则
03. */
04. private static void initFlowRules() {
05.     List<FlowRule> rules = new ArrayList<>();
06.     // 定义一个限流规则对象
07.     FlowRule rule = new FlowRule();
08.     // 资源名称
09.     rule.setResource("testD-resource");
10.    // 限流阈值的类型
11.    rule.setGrade(RuleConstant.FLOW_GRADE_QPS);
12.    // 设置 QPS 的阈值为 2
13.    rule.setCount(2);
14.    rules.add(rule);
15.    // 定义限流规则
16.    FlowRuleManager.loadRules(rules);
17. }
```

2. 在 testD() 方法中调用 initFlowRules() 方法，初始化流控规则，代码如下。

```

01. @GetMapping("/testD")
02. @SentinelResource(value = "testD-resource", blockHandler = "blockHandlerTestD") // 通过注解定义资源
03. public String testD() {
04.     initFlowRules(); // 调用初始化流控规则的方法
05.     log.info("c语言中文网提醒您，服务访问成功-----testD: " + serverPort);
06.     return "c语言中文网提醒您，服务访问成功-----testD: " + serverPort;
07. }
```

3. 重启 spring-cloud-alibaba-sentinel-service-8401，并使用浏览器访问“<http://localhost:8401/testD>”，结果如下。

c语言中文网提醒您，服务访问成功-----testD: 8401

4. 快速连续（频率大于每秒钟 2 次）访问“<http://localhost:8401/testD>”，结果如下。



c语言中文网提醒您，TestD服务访问失败！您已被限流，请稍后重试

#### 5. 打开命令行窗口，执行以下命令查看资源的实时统计信息。

```
curl http://localhost:8719/cnode?id=testD-resource
```

#### 6. 控制台输出内容如下。

idx	id	thread	pass	blocked	success	total	aRt	1m-pass	1m-block	1m-all	exception
2	testD-resource	0	0.0	0.0	0.0	0.0	0.0	10	16	26	0.0

实时统计信息各列名说明如下：

- thread：代表当前处理该资源的并发数；
- pass：代表一秒内到来的请求；
- blocked：代表一秒内被流量控制的请求数量；
- success：代表一秒内成功处理完的请求；
- total：代表到一秒内到来的请求以及被阻止的请求总和；
- RT：代表一秒内该资源的平均响应时间；
- 1m-pass：则是一分钟内到来的请求；
- 1m-block：则是一分钟内被阻止的请求；
- 1m-all：则是一分钟内到来的请求和被阻止的请求的总和；
- exception：则是一秒内业务本身异常的总和。

## 熔断降级规则

除了流量控制以外，对调用链路中不稳定资源的熔断降级，也是保障服务高可用的重要措施之一。

在分布式微服务架构中，一个系统往往由多个服务组成，不同服务之间相互调用，组成复杂的调用链路。如果链路上的某一个服务出现故障，那么故障就会沿着调用链路在系统中蔓延，最终导致整个系统瘫痪。Sentinel 提供了熔断降级机制就可以解决这个问题。

Sentinel 的熔断降级机制会在调用链路中某个资源出现不稳定状态时（例如调用超时或异常比例升高），暂时切断对这个资源的调用，以避免局部不稳定因素导致整个系统的雪崩。

熔断降级作为服务保护自身的手段，通常在客户端（调用端）进行配置，资源被熔断降级最直接的表现就是抛出 DegradeException 异常。

### Sentinel 熔断策略

Sentinel 提供了 3 种熔断策略，如下表所示。

熔断策略	说明
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（即最大响应时间），若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，且慢调用的比例大于阈值，则接下来的熔断时长



	<p>内请求会自动被熔断。</p> <p>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 则再次被熔断。</p>
异常比例 (ERROR_RATIO)	<p>当单位统计时长 (statIntervalMs) 内请求数目大于设置的最小请求数目且异常的比例大于阈值, 则在接下来的熔断时长内请求会自动被熔断。</p> <p>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%。</p>
异常数 (ERROR_COUNT)	<p>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。</p> <p>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。</p>

注意: Sentinel 1.8.0 版本对熔断降级特性进行了全新的改进升级, 以上熔断策略针对的是 Sentinel 1.8.0 及以上版本。

### Sentinel 熔断状态

Sentinel 熔断降级中共涉及 3 种状态, 熔断状态的之间的转换过程如下图。

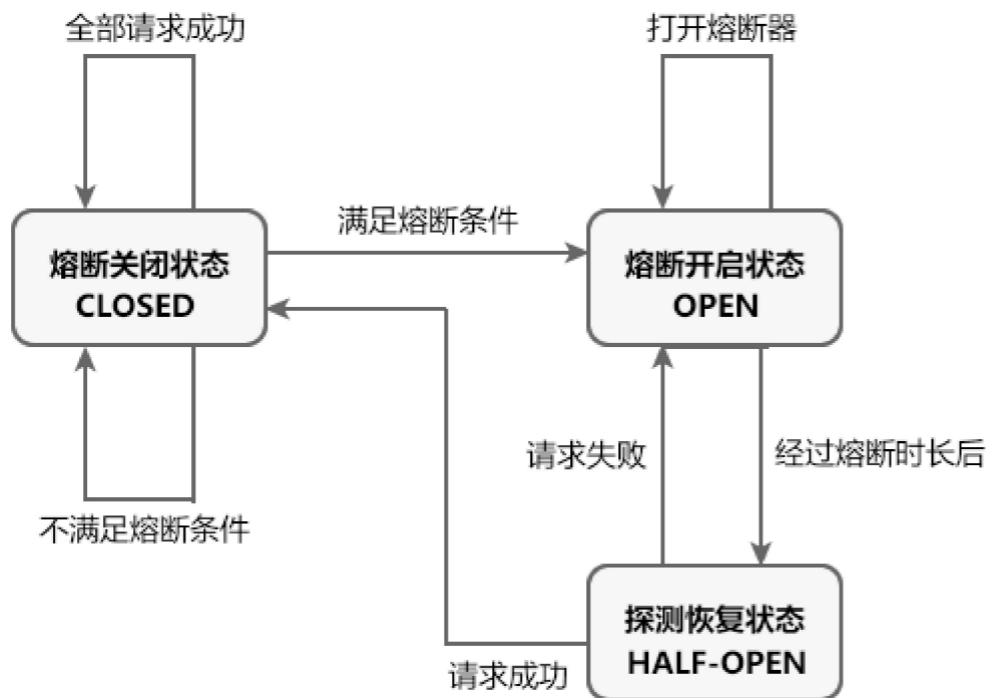


图16: Sentinel 熔断状态转换

Sentinel 熔断降级中共涉及 3 种状态, 如下表。

状态	说明	触发条件
熔断关闭状态 (CLOSED)	处于关闭状态时, 请求可以正常调用资源。	<p>满足以下任意条件, Sentinel 熔断器进入熔断关闭状态:</p> <ul style="list-style-type: none"> <li>• 全部请求访问成功。</li> </ul>

		<ul style="list-style-type: none"> <li>单位统计时长 (statIntervalMs) 内请求数目小于设置的最小请求数目。</li> <li>未达到熔断标准，例如服务超时比例、异常数、异常比例未达到阈值。</li> <li>处于探测恢复状态时，下一个请求访问成功。</li> </ul>
熔断开启状态 (OPEN)	处于熔断开启状态时，熔断器会一定的时间（规定的熔断时长）内，暂时切断所有请求对该资源的调用，并调用相应的降级逻辑使请求快速失败避免系统崩溃。	满足以下任意条件，Sentinel 熔断器进入熔断开启状态： <ul style="list-style-type: none"> <li>单位统计时长内请求数目大于设置的最小请求数目，且已达到熔断标准，例如请求超时比例、异常数、异常比例达到阈值。</li> <li>处于探测恢复状态时，下一个请求访问失败。</li> </ul>
探测恢复状态 (HALF-OPEN)	处于探测恢复状态时，Sentinel 熔断器会允许一个请求调用资源。则若接下来的一个请求成功完成（没有错误）则结束熔断，熔断器进入熔断关闭（CLOSED）状态；否则会再次被熔断，熔断器进入熔断开启（OPEN）状态。	在熔断开启一段时间（降级窗口时间或熔断时长，单位为 s）后，Sentinel 熔断器自动会进入探测恢复状态。

### Sentinel 熔断规则属性

Sentinel 熔断降级规则包含多个重要属性，如下表所示。

属性	说明	默认值	使用范围
资源名	规则的作用对象。	-	所有熔断策略
熔断策略	Sentinel 支持3 中熔断策略：慢调用比例、异常比例、异常数策略。	慢调用比例	所有熔断策略
最大 RT	请求的最大响应时间，请求的响应时间大于该值则统计为慢调用。	-	慢调用比例
熔断时长	熔断开启状态持续的时间，超过该时间熔断器会切换为探测恢复状态（HALF-OPEN），单位为 s。	-	所有熔断策略
最小请求数	熔断触发的最小请求数，请求数小于该值时即使异常比率超出阈值也不会熔断（1.7.0 引入）。	5	所有熔断策略
统计时长	熔断触发需要统计的时长（单位为 ms），如 60*1000 代表分钟级（1.8.0 引入）。	1000 ms	所有熔断策略
比例阈值	分为慢调用比例阈值和异常比例阈值，即慢调用或异常调用占所有请求的百分比，取值范围 [0.0,1.0]。	-	慢调用比例、异常比例
异常数	请求或调用发生的异常的数量。	-	异常数

### Sentinel 实现熔断降级过程

Sentinel 实现熔断降级的步骤如下：

- 在项目中，使用 @SentinelResource 注解的 fallback 属性可以为资源指定熔断降级逻辑（方法）。
- 通过 Sentinel 控制台或代码定义熔断规则，包括熔断策略、最小请求数、阈值、熔断时长以及统计时长等。
- 若单位统计时长 (statIntervalMs) 内，请求数目大于设置的最小请求数目且达到熔断标准（例如请求超时比例、异常数、异常比例达到阈值），Sentinel 熔断器进入熔断开启状态（OPEN）。
- 处于熔断开启状态时，@SentinelResource 注解的 fallback 属性指定的降级逻辑会临时充当主业务逻辑，而原来的主逻辑则暂时不可用。当有请求访问该资源时，会直接调用降级逻辑使请求快速失败，而不会调用原来的主业务逻辑。



5. 在经过一段时间（在熔断规则中设置的熔断时长）后，熔断器会进入探测恢复状态（HALF-OPEN），此时 Sentinel 会允许一个请求对原来的主营业务逻辑进行调用，并监控其调用结果。
6. 若请求调用成功，则熔断器进入熔断关闭状态（CLOSED），服务原来的主营业务逻辑恢复，否则重新进入熔断开启状态（OPEN）。

#### 通过 Sentinel 控制台定义熔断降级规则

我们可以通过 Sentinel 控制台直接对资源定义熔断降级规则。

- 下面我们将通过一个实例，来演示如何通过 Sentinel 控制台，对资源定义降级规则。
1. 在 MySQL 的 bianchengbang\_jdbc 数据库中执行以下 SQL，准备测试数据。

```
DROP TABLE IF EXISTS `dept`;  
CREATE TABLE `dept` (  
    `dept_no` int NOT NULL AUTO_INCREMENT,  
    `dept_name` varchar(255) DEFAULT NULL,  
    `db_source` varchar(255) DEFAULT NULL,  
    PRIMARY KEY (`dept_no`)  
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;  
  
INSERT INTO `dept` VALUES ('1', '开发部', 'bianchengbang_jdbc');  
INSERT INTO `dept` VALUES ('2', '人事部', 'bianchengbang_jdbc');  
INSERT INTO `dept` VALUES ('3', '财务部', 'bianchengbang_jdbc');  
INSERT INTO `dept` VALUES ('4', '市场部', 'bianchengbang_jdbc');  
INSERT INTO `dept` VALUES ('5', '运维部', 'bianchengbang_jdbc');
```

2. 在主工程 spring-cloud-alibaba-demo 下，创建一个名为 spring-cloud-alibaba-provider-mysql-8003 的 Spring Boot 模块，并在其 pom.xml 中添加相关依赖，代码如下。

```
01. <?xml version="1.0" encoding="UTF-8"?>  
02. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
03.     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
04.     <modelVersion>4.0.0</modelVersion>  
05.     <parent>  
06.         <groupId>net.biancheng.c</groupId>  
07.         <version>1.0-SNAPSHOT</version>  
08.         <artifactId>spring-cloud-alibaba-demo</artifactId>  
09.     </parent>  
10.  
11.     <groupId>net.biancheng.c</groupId>  
12.     <artifactId>spring-cloud-alibaba-provider-mysql-8003</artifactId>  
13.     <version>0.0.1-SNAPSHOT</version>  
14.     <name>spring-cloud-alibaba-provider-mysql-8003</name>  
15.     <description>Demo project for Spring Boot</description>  
16.     <properties>  
17.         <java.version>1.8</java.version>
```



```
18. </properties>
19. <dependencies>
20.   <dependency>
21.     <groupId>org.springframework.boot</groupId>
22.     <artifactId>spring-boot-starter-web</artifactId>
23.   </dependency>
24.
25.   <dependency>
26.     <groupId>org.springframework.boot</groupId>
27.     <artifactId>spring-boot-devtools</artifactId>
28.     <scope>runtime</scope>
29.     <optional>true</optional>
30.   </dependency>
31.   <dependency>
32.     <groupId>org.projectlombok</groupId>
33.     <artifactId>lombok</artifactId>
34.     <optional>true</optional>
35.   </dependency>
36.   <dependency>
37.     <groupId>org.springframework.boot</groupId>
38.     <artifactId>spring-boot-starter-test</artifactId>
39.     <scope>test</scope>
40.   </dependency>
41.   <dependency>
42.     <groupId>com.alibaba.cloud</groupId>
43.     <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
44.   </dependency>
45.   <dependency>
46.     <groupId>net.biancheng.c</groupId>
47.     <artifactId>spring-cloud-alibaba-api</artifactId>
48.     <version>${project.version}</version>
49.   </dependency>
50.
51.   <dependency>
52.     <groupId>junit</groupId>
53.     <artifactId>junit</artifactId>
54.     <version>4.12</version>
55.   </dependency>
56.   <dependency>
57.     <groupId>mysql</groupId>
58.     <artifactId>mysql-connector-java</artifactId>
59.     <version>5.1.49</version>
60.   </dependency>
61.
62.   <dependency>
63.     <groupId>ch.qos.logback</groupId>
```



```

64.      <artifactId>logback-core</artifactId>
65.
66.      </dependency>
67.      <dependency>
68.          <groupId>org.mybatis.spring.boot</groupId>
69.          <artifactId>mybatis-spring-boot-starter</artifactId>
70.          <version>2.2.0</version>
71.      </dependency>
72.      <!--添加 Spring Boot 的监控模块--&gt;
73.      &lt;!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-actuator --&gt;
74.      &lt;dependency&gt;
75.          &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
76.          &lt;artifactId&gt;spring-boot-starter-actuator&lt;/artifactId&gt;
77.      &lt;/dependency&gt;
78.  &lt;/dependencies&gt;
79.
80.  &lt;build&gt;
81.      &lt;plugins&gt;
82.          &lt;plugin&gt;
83.              &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
84.              &lt;artifactId&gt;spring-boot-maven-plugin&lt;/artifactId&gt;
85.              &lt;configuration&gt;
86.                  &lt;excludes&gt;
87.                      &lt;exclude&gt;
88.                          &lt;groupId&gt;org.projectlombok&lt;/groupId&gt;
89.                          &lt;artifactId&gt;lombok&lt;/artifactId&gt;
90.                      &lt;/exclude&gt;
91.                  &lt;/excludes&gt;
92.              &lt;/configuration&gt;
93.          &lt;/plugin&gt;
94.      &lt;/plugins&gt;
95.  &lt;/build&gt;
96. &lt;/project&gt;
</pre>

```

3. 在 spring-cloud-alibaba-provider-mysql-8003 的类路径下，创建一个配置文件 application.yml，配置如下。

```

01. server:
02.     port: 8003 #端口
03.
04. spring:
05.     application:
06.         name: spring-cloud-alibaba-provider-mysql
07.
08. cloud:
09.     nacos:
10.         discovery:

```



```
11.     server-addr: localhost:1111
12. ##### 数据库连接 #####
13. datasource:
14.     username: root      #数据库登陆用户名
15.     password: root      #数据库登陆密码
16.     url: jdbc:mysql://127.0.0.1:3306/spring_cloud_db2      #数据库url
17.     driver-class-name: com.mysql.jdbc.Driver
18.
19. management:
20. endpoints:
21.     web:
22.         exposure:
23.             include: "*"  # * 在yaml 文件属于关键字, 所以需要加引号
24. ##### MyBatis 配置 #####
25. mybatis:
26.     # 指定 mapper.xml 的位置
27.     mapper-locations: classpath:mybatis/mapper/*.xml
28.     #扫描实体类的位置,在此处指明扫描实体类的包,在 mapper.xml 中就可以不写实体类的全路径名
29.     type-aliases-package: net.biancheng.c.entity
30.     configuration:
31.         #默认开启驼峰命名法, 可以不用设置该属性
32.         map-underscore-to-camel-case: true
```

4. 在 net.biancheng.c.entity 包下, 创建一个名为 Dept 的实体类, 代码如下。

```
01. package net.biancheng.c.entity;
02.
03. import lombok.AllArgsConstructor;
04. import lombok.Data;
05. import lombok.NoArgsConstructor;
06. import lombok.experimental.Accessors;
07.
08. import java.io.Serializable;
09.
10. @AllArgsConstructor
11. @NoArgsConstructor //无参构造函数
12. @Data // 提供类的get、set、equals、hashCode、canEqual、toString 方法
13. @Accessors(chain = true)
14. public class Dept implements Serializable {
15.     private Integer deptNo;
16.     private String deptName;
17.     private String dbSource;
18. }
```

5. 在 net.biancheng.c.entity 包下, 创建一个名为 CommonResult 的 Java 类, 代码如下。



```
01. package net.biancheng.c.entity;
02.
03. import lombok.AllArgsConstructor;
04. import lombok.Data;
05. import lombok.NoArgsConstructor;
06.
07. @Data
08.@AllArgsConstructor
09.@NoArgsConstructor
10. public class CommonResult<T> {
11.     private Integer code;
12.     private String message;
13.     private T data;
14.
15.     public CommonResult(Integer code, String message) {
16.         this(code, message, null);
17.     }
18. }
```

6. 在 net.biancheng.c.mapper 包下，创建一个名为 DeptMapper 的接口，代码如下。

```
01. package net.biancheng.c.mapper;
02.
03. import net.biancheng.c.entity.Dept;
04. import org.apache.ibatis.annotations.Mapper;
05.
06. import java.util.List;
07.
08. @Mapper
09. public interface DeptMapper {
10.     //根据主键获取数据
11.     Dept selectByPrimaryKey(Integer deptNo);
12.
13.     //获取表中的全部数据
14.     List<Dept> getAll();
15. }
```

7. 在 spring-cloud-alibaba-provider-mysql-8003 的 /resources/mybatis/mapper/ 目录下，创建一个名为 DeptMapper.xml 的 MyBatis 映射文件，配置内容如下。

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
03. <mapper namespace="net.biancheng.c.mapper.DeptMapper">
04.     <resultMap id="BaseResultMap" type="net.biancheng.c.entity.Dept">
05.         <id column="dept_no" jdbcType="INTEGER" property="deptNo"/>
```



```

06.      <result column="dept_name" jdbcType="VARCHAR" property="deptName"/>
07.      <result column="db_source" jdbcType="VARCHAR" property="dbSource"/>
08.  </resultMap>
09.
10.  <sql id="Base_Column_List">
11.      dept_no
12.      , dept_name, db_source
13.  </sql>
14.
15.  <select id="selectByPrimaryKey" parameterType="java.lang.Integer" resultMap="BaseResultMap">
16.      select
17.          <include refid="Base_Column_List"/>
18.      from dept
19.      where dept_no = #{deptNo, jdbcType=INTEGER}
20.  </select>
21.
22.  <select id="GetAll" resultType="net.biancheng.c.entity.Dept">
23.      select *
24.      from dept;
25.  </select>
26. </mapper>

```

8. 在 net.biancheng.c.service 包下，创建一个名为 DeptService 的接口，代码如下。

```

01. package net.biancheng.c.service;
02.
03. import net.biancheng.c.entity.Dept;
04.
05. import java.util.List;
06.
07. public interface DeptService {
08.
09.     Dept get(Integer deptNo);
10.
11.     List<Dept> selectAll();
12. }

```

9. 在 net.biancheng.c.service.impl 包下，创建 DeptService 接口的实现类 DeptServiceImpl，代码如下。

```

01. package net.biancheng.c.service.impl;
02.
03. import net.biancheng.c.entity.Dept;
04. import net.biancheng.c.mapper.DeptMapper;
05. import net.biancheng.c.service.DeptService;
06. import org.springframework.beans.factory.annotation.Autowired;
07. import org.springframework.stereotype.Service;

```



```
08.  
09. import java.util.List;  
10.  
11. @Service("deptService")  
12. public class DeptServiceImpl implements DeptService {  
13.     @Autowired  
14.     private DeptMapper deptMapper;  
15.  
16.     @Override  
17.     public Dept get(Integer deptNo) {  
18.         return deptMapper.selectByPrimaryKey(deptNo);  
19.     }  
20.  
21.     @Override  
22.     public List<Dept> selectAll() {  
23.         return deptMapper.GetAll();  
24.     }  
25. }
```

10. 在 net.biancheng.c.controller 包下，创建一个名为 DeptController 的 Controller 类，代码如下。

```
01. package net.biancheng.c.controller;  
02.  
03. import lombok.extern.slf4j.Slf4j;  
04. import net.biancheng.c.entity.CommonResult;  
05. import net.biancheng.c.entity.Dept;  
06. import net.biancheng.c.service.DeptService;  
07. import org.springframework.beans.factory.annotation.Autowired;  
08. import org.springframework.beans.factory.annotation.Value;  
09. import org.springframework.web.bind.annotation.PathVariable;  
10. import org.springframework.web.bind.annotation.RequestMapping;  
11. import org.springframework.web.bind.annotation.RequestMethod;  
12. import org.springframework.web.bind.annotation.RestController;  
13.  
14. import java.util.List;  
15. import java.util.concurrent.TimeUnit;  
16.  
17. @RestController  
18. @Slf4j  
19. public class DeptController {  
20.     @Autowired  
21.     private DeptService deptService;  
22.  
23.     @Value("${server.port}")  
24.     private String serverPort;  
25. }
```



```

26.     @RequestMapping(value = "/dept/get/{id}", method = RequestMethod.GET)
27.     public CommonResult<Dept> get(@PathVariable("id") int id) {
28.         log.info("端口: " + serverPort + "\t+ dept/get/");
29.         try {
30.             TimeUnit.SECONDS.sleep(1);
31.             log.info("休眠 1秒");
32.         } catch (InterruptedException e) {
33.             e.printStackTrace();
34.         }
35.
36.         Dept dept = deptService.get(id);
37.         CommonResult<Dept> result = new CommonResult(200, "from mysql,serverPort: " + serverPort, dept);
38.         return result;
39.     }
40.
41.     @RequestMapping(value = "/dept/list", method = RequestMethod.GET)
42.     public CommonResult<List<Dept>> list() {
43.         log.info("端口: " + serverPort + "\t+ dept/list/");
44.
45.         List<Dept> depts = deptService.selectAll();
46.         CommonResult<List<Dept>> result = new CommonResult(200, "from mysql,serverPort: " + serverPort, depts);
47.         return result;
48.     }
49. }
```

11. spring-cloud-alibaba-provider-mysql-8003 的主启动类代码如下。

```

01. package net.biancheng.c;
02.
03. import org.springframework.boot.SpringApplication;
04. import org.springframework.boot.autoconfigure.SpringBootApplication;
05. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
06.
07. @SpringBootApplication
08. @EnableDiscoveryClient
09. public class SpringCloudAlibabaProviderMysql8003Application {
10.
11.     public static void main(String[] args) {
12.         SpringApplication.run(SpringCloudAlibabaProviderMysql8003Application.class, args);
13.     }
14. }
```

12. 在主工程 spring-cloud-alibaba-demo 下，创建一个名为 spring-cloud-alibaba-consumer-mysql-8803 的 Spring Boot 模块，并在其 pom.xml 添加依赖，内容如下。



```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
04.   <modelVersion>4.0.0</modelVersion>
05.   <parent>
06.     <groupId>net.biancheng.c</groupId>
07.     <version>1.0-SNAPSHOT</version>
08.     <artifactId>spring-cloud-alibaba-demo</artifactId>
09.   </parent>
10.
11.   <groupId>net.biancheng.c</groupId>
12.   <artifactId>spring-cloud-alibaba-consumer-mysql-8803</artifactId>
13.   <version>0.0.1-SNAPSHOT</version>
14.   <name>spring-cloud-alibaba-consumer-mysql-8803</name>
15.   <description>Demo project for Spring Boot</description>
16.   <properties>
17.     <java.version>1.8</java.version>
18.   </properties>
19.   <dependencies>
20.     <!--SpringCloud alibaba nacos -->
21.     <dependency>
22.       <groupId>com.alibaba.cloud</groupId>
23.       <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
24.       <exclusions>
25.         <exclusion>
26.           <groupId>org.springframework.cloud</groupId>
27.           <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
28.         </exclusion>
29.       </exclusions>
30.     </dependency>
31.     <dependency>
32.       <groupId>org.springframework.boot</groupId>
33.       <artifactId>spring-boot-starter-web</artifactId>
34.     </dependency>
35.     <!--引入 OpenFeign 的依赖-->
36.     <dependency>
37.       <groupId>org.springframework.cloud</groupId>
38.       <artifactId>spring-cloud-starter-openfeign</artifactId>
39.     </dependency>
40.     <dependency>
41.       <groupId>org.springframework.cloud</groupId>
42.       <artifactId>spring-cloud-loadbalancer</artifactId>
43.     </dependency>
44.     <dependency>
45.       <groupId>org.springframework.boot</groupId>
46.       <artifactId>spring-boot-devtools</artifactId>
```



```
47.      <scope>runtime</scope>
48.      <optional>true</optional>
49.    </dependency>
50.  <dependency>
51.    <groupId>org.projectlombok</groupId>
52.    <artifactId>lombok</artifactId>
53.    <optional>true</optional>
54.  </dependency>
55.  <dependency>
56.    <groupId>org.springframework.boot</groupId>
57.    <artifactId>spring-boot-starter-test</artifactId>
58.    <scope>test</scope>
59.  </dependency>
60.  <!--SpringCloud alibaba sentinel -->
61.  <dependency>
62.    <groupId>com.alibaba.cloud</groupId>
63.    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
64.  </dependency>
65.  <dependency>
66.    <groupId>net.biancheng.c</groupId>
67.    <artifactId>spring-cloud-alibaba-api</artifactId>
68.    <version>${project.version}</version>
69.  </dependency>
70.  <dependency>
71.    <groupId>org.springframework.boot</groupId>
72.    <artifactId>spring-boot-starter-actuator</artifactId>
73.  </dependency>
74. </dependencies>
75.
76. <build>
77.   <plugins>
78.     <plugin>
79.       <groupId>org.springframework.boot</groupId>
80.       <artifactId>spring-boot-maven-plugin</artifactId>
81.       <configuration>
82.         <excludes>
83.           <exclude>
84.             <groupId>org.projectlombok</groupId>
85.             <artifactId>lombok</artifactId>
86.           </exclude>
87.         </excludes>
88.       </configuration>
89.     </plugin>
90.   </plugins>
91. </build>
92. </project>
```



13. 在 spring-cloud-alibaba-consumer-mysql-8803 的类路径下，创建一个配置文件 application.yml，配置内容如下。

```
01. server:
02.   port: 8803
03. spring:
04.   application:
05.     name: spring-cloud-alibaba-consumer-mysql-feign
06.   cloud:
07.     nacos:
08.       discovery:
09.         server-addr: localhost:1111
10.   sentinel:
11.     transport:
12.       dashboard: localhost:8080
13.     port: 8719
14.
15. # 以下配置信息并不是默认配置，而是我们自定义的配置，目的是不在 Controller 内硬编码 服务提供者的服务名
16. service-url:
17.   nacos-user-service: http://spring-cloud-alibaba-provider-mysql #消费者要访问的微服务名称
18.
19. # 激活Sentinel对Feign的支持
20. feign:
21.   sentinel:
22.     enabled: true
```

14. 在 net.biancheng.c.service 包下，创建一个名为 DeptFeignService 的接口，代码如下。

```
01. package net.biancheng.c.service;
02.
03. import net.biancheng.c.entity.CommonResult;
04. import net.biancheng.c.entity.Dept;
05. import org.springframework.cloud.openfeign.FeignClient;
06. import org.springframework.stereotype.Component;
07. import org.springframework.web.bind.annotation.PathVariable;
08. import org.springframework.web.bind.annotation.RequestMapping;
09. import org.springframework.web.bind.annotation.RequestMethod;
10.
11. import java.util.List;
12.
13. @Component
14. @FeignClient(value = "spring-cloud-alibaba-provider-mysql", fallback = DeptFallbackService.class)
15. public interface DeptFeignService {
16.   @RequestMapping(value = "/dept/get/{id}", method = RequestMethod.GET)
17.   public CommonResult<Dept> get(@PathVariable("id") int id);
18.
```



```
19. @RequestMapping(value = "/dept/list", method = RequestMethod.GET)
20. public CommonResult<List<Dept>> list();
21. }
```

15. 在 net.biancheng.c.controller 包下，创建一个名为 DeptFeignController 的 Controller，代码如下。

```
01. package net.biancheng.c.controller;
02.
03. import com.alibaba.csp.sentinel.annotation.SentinelResource;
04. import com.alibaba.csp.sentinel.slots.block.degrade.circuitbreaker.CircuitBreaker;
05. import com.alibaba.csp.sentinel.slots.block.degrade.circuitbreaker.EventObserverRegistry;
06. import com.alibaba.csp.sentinel.util.TimeUtil;
07. import lombok.extern.slf4j.Slf4j;
08. import net.biancheng.c.entity.CommonResult;
09. import net.biancheng.c.entity.Dept;
10. import net.biancheng.c.service.DeptFeignService;
11. import org.springframework.web.bind.annotation.PathVariable;
12. import org.springframework.web.bind.annotation.RequestMapping;
13. import org.springframework.web.bind.annotation.RequestMethod;
14. import org.springframework.web.bind.annotation.RestController;
15.
16. import javax.annotation.Resource;
17. import java.text.SimpleDateFormat;
18. import java.util.Date;
19. import java.util.List;
20.
21. @RestController
22. @Slf4j
23. public class DeptFeignController {
24.     @Resource
25.     DeptFeignService deptFeignService;
26.
27.     @RequestMapping(value = "consumer/feign/dept/get/{id}", method = RequestMethod.GET)
28.     @SentinelResource(value = "fallback", fallback = "handlerFallback")
29.     public CommonResult<Dept> get(@PathVariable("id") int id) {
30.         monitor();
31.         System.out.println("----->>>主业务逻辑");
32.         CommonResult<Dept> result = deptFeignService.get(id);
33.         if (id == 6) {
34.             System.err.println("----->>>主业务逻辑，抛出非法参数异常");
35.             throw new IllegalArgumentException("IllegalArgumentException, 非法参数异常....");
36.             //如果查到的记录也是 null 也控制正异常
37.         } else if (result.getData() == null) {
38.             System.err.println("----->>>主业务逻辑，抛出空指针异常");
39.             throw new NullPointerException("NullPointerException, 该ID没有对应记录, 空指针异常");
40.     }
41. }
```



```

41.         return result;
42.     }
43.
44.     @RequestMapping(value = "consumer/feign/dept/list", method = RequestMethod.GET)
45.     public CommonResult<List<Dept>> list() {
46.         return deptFeignService.list();
47.     }
48.
49.     //处理异常的回退方法（服务降级）
50.     public CommonResult handlerFallback(@PathVariable int id, Throwable e) {
51.         System.err.println("----->>>服务降级逻辑");
52.         Dept dept = new Dept(id, "null", "null");
53.         return new CommonResult(444, "C语言中文网提醒您，服务被降级！异常信息为：" + e.getMessage(), dept);
54.     }
55.
56.     /**
57.      * 自定义事件监听器，监听熔断器状态转换
58.      */
59.     public void monitor() {
60.         EventObserverRegistry.getInstance().addStateChangeObserver("logging",
61.             (prevState, newState, rule, snapshotValue) -> {
62.                 SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
63.                 if (newState == CircuitBreaker.State.OPEN) {
64.                     // 变换至 OPEN state 时会携带触发时的值
65.                     System.err.println(String.format("%s -> OPEN at %s, 发送请求次数=%.2f", prevState.name(),
66.                         format.format(new Date(TimeUtil.currentTimeMillis())), snapshotValue));
67.                 } else {
68.                     System.err.println(String.format("%s -> %s at %s", prevState.name(), newState.name(),
69.                         format.format(new Date(TimeUtil.currentTimeMillis()))));
70.                 }
71.             });
72.     }
73. }
74. }
```

在以上代码中，我们通过 `@SentinelResource` 注解的 `fallback` 属性指定了一个 `fallback` 函数，进行熔断降级的后续处理。

使用 `@SentinelResource` 注解的 `blockHandler` 属性时，需要注意以下事项：

- 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。

## 16. spring-cloud-alibaba-consumer-mysql-8803 的主启动类代码如下。

```
01. package net.biancheng.c;
```



```
02.  
03. import org.springframework.boot.SpringApplication;  
04. import org.springframework.boot.autoconfigure.SpringBootApplication;  
05. import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
06. import org.springframework.cloud.openfeign.EnableFeignClients;  
07.  
08. @SpringBootApplication  
09. @EnableDiscoveryClient  
10. @EnableFeignClients  
11. public class SpringCloudAlibabaConsumerMysql8803Application {  
12.  
13.     public static void main(String[] args) {  
14.         SpringApplication.run(SpringCloudAlibabaConsumerMysql8803Application.class, args);  
15.     }  
16.  
17. }
```

17. 依次启动 spring-cloud-alibaba-provider-mysql-8003 和 spring-cloud-alibaba-consumer-mysql-8803，使用浏览器访问 “<http://localhost:8803/consumer/feign/dept/get/3>” , 结果如下。

```
{"code":200,"message":"from mysql,serverPort: 8003","data":{"deptNo":3,"deptName":"财务部","dbSource":"spring_cloud_db2"}}
```

18. 使用浏览器访问 “<http://localhost:8803/consumer/feign/dept/get/7>” , 结果如下。

```
{"code":444,"message":"C语言中文网提醒您，服务被降级！异常信息为：NullPointerException，该ID没有对应记录，空指针异常","data": {"deptNo":7,"deptName":"null","dbSource":"null"}}
```

19. 控制台输出如下。

```
----->>>主营业务逻辑  
----->>>主营业务逻辑  
----->>>主营业务逻辑，抛出空指针异常  
----->>>服务熔断降级逻辑
```

20. 使用浏览器访问 Sentinel 控制台，在 “簇点链路” 列表中，点击 fallback 资源的 “+熔断” 按钮，如下图。



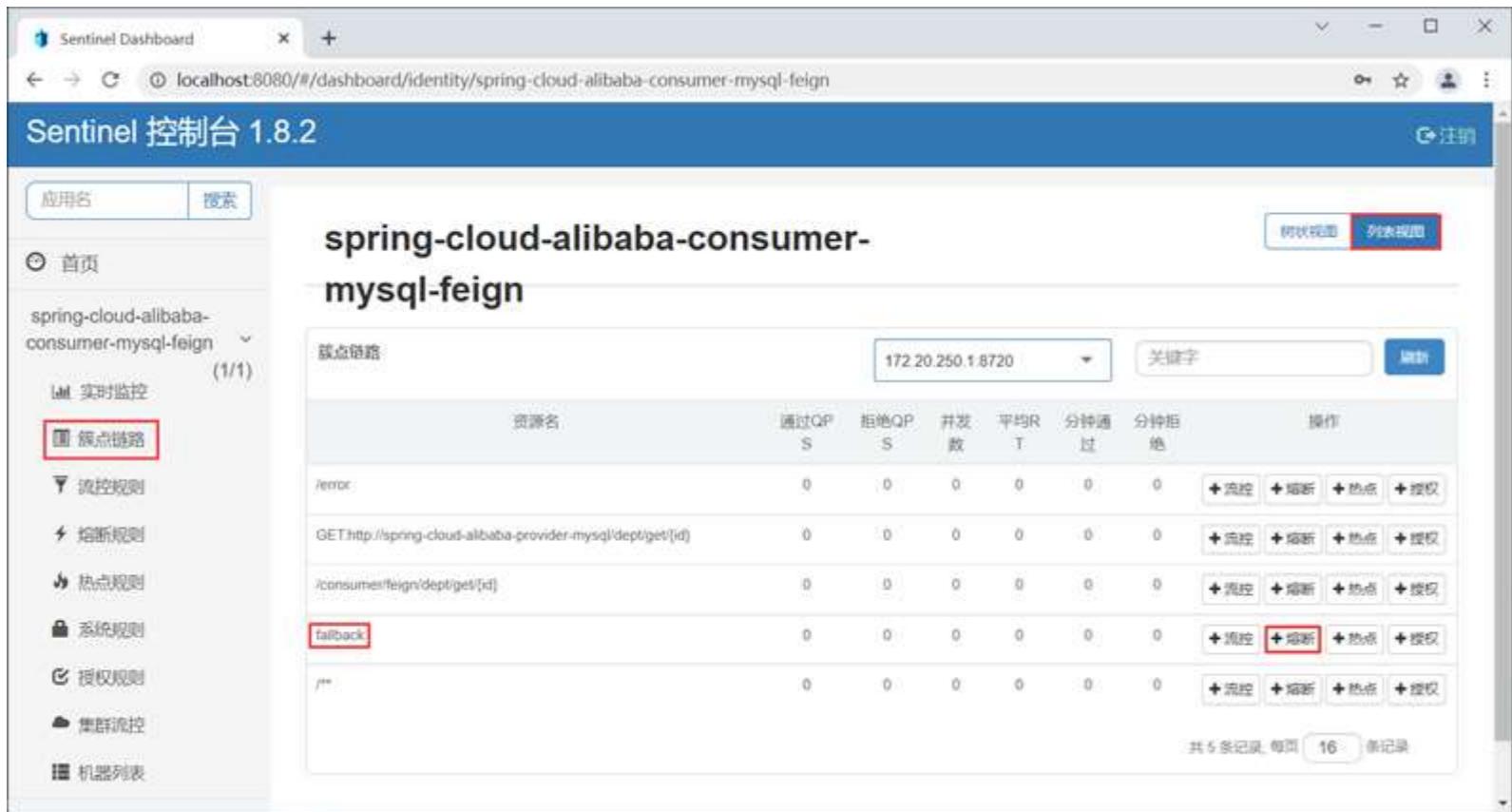


图17: Sentinel 熔断降级

21. 在“新增熔断规则”的窗口中，为名为“fallback”的资源定义以下熔断规则，如下图。



图18: Sentinel 控制台定义熔断规则

在上图中，熔断规则各属性说明如下：

- 异常数为 1；
- 统计时长为 1000 ms (即 1s)；
- 最小请求数为 2；
- 熔断时长为 5 秒；

我们为 fallback 资源定义的熔断规则为：当 1 秒钟内请求数大于 2 个，且请求异常数大于 1 时，服务被熔断，熔断的时长为 5 秒钟。

22. 使用浏览器连续访问 “<http://localhost:8803/consumer/feign/dept/get/7>” , 访问频率大于每秒 2 个请求，结果如下。

```
{"code":444,"message":"C语言中文网提醒您，服务被降级！异常信息为: null","data":{"deptNo":7,"deptName":"null","dbSource":"null"}}
```

23. 控制台输出如下。

```
----->>>主营业务逻辑  
----->>>主营业务逻辑，抛出空指针异常  
----->>>服务降级逻辑  
----->>>主营业务逻辑  
----->>>主营业务逻辑，抛出空指针异常  
----->>>服务降级逻辑  
CLOSED -> OPEN at 2021-11-17 14:06:47, 发送请求次数=2.00
```

从控制台输出可以看出，在 14 点 06 分 47 秒时，熔断器从熔断关闭状态 (CLOSED) 切换到熔断开启状态 (OPEN) 。

23. 在熔断开启状态下，使用浏览器访问 “<http://localhost:8803/consumer/feign/dept/get/4>” , 结果页面输出如下。

```
{"code":444,"message":"C语言中文网提醒您，服务被降级！异常信息为: null","data":{"deptNo":4,"deptName":"null","dbSource":"null"}}
```

24. 控制台输出如下。

```
----->>>主营业务逻辑  
----->>>主营业务逻辑，抛出空指针异常  
----->>>服务降级逻辑  
----->>>主营业务逻辑  
----->>>主营业务逻辑，抛出空指针异常  
----->>>服务降级逻辑  
CLOSED -> OPEN at 2021-11-17 14:09:19, 发送请求次数=2.00  
----->>>服务降级逻辑  
----->>>服务降级逻辑  
----->>>服务降级逻辑
```

从控制台输出可知，当熔断器处于熔断开启状态时，所有的请求都直接交给降级逻辑处理。

25. 继续使用浏览器访问 “<http://localhost:8803/consumer/feign/dept/get/4>” , 结果页面输出如下。

```
{"code":200,"message":"from mysql, serverPort: 8003","data":{"deptNo":4,"deptName":"市场部","dbSource":"spring_cloud_db2"}}
```

26. 控制台输出如下。



```

----->>>主营业务逻辑
----->>>主营业务逻辑，抛出空指针异常
----->>>服务降级逻辑
----->>>主营业务逻辑
----->>>主营业务逻辑，抛出空指针异常
----->>>服务降级逻辑
CLOSED -> OPEN at 2021-11-17 14:09:19, 发送请求次数=2.00
----->>>服务降级逻辑
----->>>服务降级逻辑
----->>>服务降级逻辑
OPEN -> HALF_OPEN at 2021-11-17 14:09:24
----->>>主营业务逻辑
HALF_OPEN -> CLOSED at 2021-11-17 14:09:24
----->>>主营业务逻辑
----->>>主营业务逻辑

```

从以上控制台输出可知，熔断器在经历了 5 秒的熔断时长后，自动切换到了探测恢复状态 (HALF-OPEN)，并在下一个请求成功的情况下，结束了熔断开启状态，切换到了熔断关闭状态 (CLOSED)。

#### 通过代码定义熔断规则

Sentinel 核心库中提供了的一个名为 DegradeRuleManager 类，我们可以通过调用它的 loadRules() 方法来定义熔断降级规则，该方法需要一个 DegradeRule 类型的 List 参数。

```

01. public static void loadRules(List<DegradeRule> rules) {
02.     try {
03.         currentProperty.updateValue(rules);
04.     } catch (Throwable var2) {
05.         RecordLog.error("[DegradeRuleManager] Unexpected error when loading degrade rules", var2);
06.     }
07. }

```

DegradeRule 类可以用来定义一条熔断规则，它包含多个与熔断规则相关的属性，如下表。

属性	说明	默认值
resource	资源名，即规则的作用对象	
grade	熔断策略，支持慢调用比例/异常比例/异常数策略	慢调用比例
count	慢调用比例模式下为慢调用临界 RT（超出该值计为慢调用）；异常比例/异常数模式下为对应的阈值	
timeWindow	熔断时长，单位为 s	
minRequestAmount	熔断触发的最小请求数，请求数小于该值时即使异常比率超出阈值也不会熔断（1.7.0 引入）	5
statIntervalMs	统计时长（单位为 ms），如 60*1000 代表分钟级（1.8.0 引入）	1000 ms
slowRatioThreshold	慢调用比例阈值，仅慢调用比例模式有效（1.8.0 引入）	



下面我们就通过一个实例，演示下如何通过代码定义熔断规则，步骤如下。

1. 在 spring-cloud-alibaba-consumer-mysql-8803 的 DeptFeignController 中，添加一个名为 initDegradeRule 的方法，代码如下。

```
01. /**
02. * 初始化熔断策略
03. */
04. private static void initDegradeRule() {
05.     List<DegradeRule> rules = new ArrayList<>();
06.     DegradeRule rule = new DegradeRule("fallback");
07.     //熔断策略为异常比例
08.     rule.setGrade(CircuitBreakerStrategy.ERROR_RATIO.getType());
09.     //异常比例阈值
10.     rule.setCount(0.7);
11.     //最小请求数
12.     rule.setMinRequestAmount(100);
13.     //统计市场，单位毫秒
14.     rule.setStatIntervalMs(30000);
15.     //熔断市场，单位秒
16.     rule.setTimeWindow(10);
17.     rules.add(rule);
18.     DegradeRuleManager.loadRules(rules);
19. }
```

2. 在 DeptFeignController 的 get() 方法中调用 initDegradeRule() 方法初始化熔断规则，代码如下。

```
01. @RequestMapping(value = "consumer/feign/dept/get/{id}", method = RequestMethod.GET)
02. @SentinelResource(value = "fallback", fallback = "handlerFallback")
03. public CommonResult<Dept> get(@PathVariable("id") int id) {
04.     initDegradeRule();
05.     monitor();
06.     System.out.println("----->>>主业务逻辑");
07.     CommonResult<Dept> result = deptFeignService.get(id);
08.     if (id == 6) {
09.         System.err.println("----->>>主业务逻辑，抛出非法参数异常");
10.         throw new IllegalArgumentException("IllegalArgumentException, 非法参数异常....");
11.         //如果查到的记录也是 null 也控制正异常
12.     } else if (result.getData() == null) {
13.         System.err.println("----->>>主业务逻辑，抛出空指针异常");
14.         throw new NullPointerException("NullPointerException, 该ID没有对应记录, 空指针异常");
15.     }
16.     return result;
17. }
```

3. 重启 spring-cloud-alibaba-consumer-mysql-8803，使用浏览器访问 “<http://localhost:8803/consumer/feign/dept/get/1>” ，结果如下。



```
"code":200,"message":"from mysql,serverPort: 8003","data":{"deptNo":1,"deptName":"开发部","dbSource":"spring_cloud_db2"}}
```

4. 使用浏览器访问 Sentinel 控制主页，点击“熔断规则”查看熔断规则列表，结果如下图。

The screenshot shows the Sentinel Control Panel (1.8.2) interface. On the left sidebar, under the '熔断规则' (Circuit Breaker Rules) section, there is a single rule listed:

资源名	熔断策略	阀值	熔断时长(s)	操作
fallback	降级比例	0.7	10s	<a href="#">编辑</a> <a href="#">删除</a>

The 'fallback' row is highlighted with a red border.

图19: Sentinel 代码定义熔断规则

从上图我们看到，通过代码也能够为资源定义熔断规则。

< 上一节

下一节 >

#### 推荐阅读

[C语言随机数生成教程，C语言rand和srand用法详解](#)

[Java项目实战：五子棋游戏（附带源码和解析）](#)

[MySQL查看存储过程](#)

[MySQL怎样将子查询修改为表连接？](#)

[C++引用（入门必读）](#)

[MySQL FROM\\_UNIXTIME函数：时间戳转日期](#)

[MySQL删除被其它表关联的主表](#)

[Django路由系统精讲](#)

[MyBatis bind标签](#)

[Scrapy爬虫框架入门教程（简明版）](#)

精美而实用的网站，分享优质编程教程，帮助有志青年。千锤百炼，只为大作；精益求精，处处斟酌；这种教程，看一眼就倾心。

[关于网站](#) | [联系我们](#) | [网站地图](#)

Copyright ©2012-2023 biancheng.net

biancheng.net

