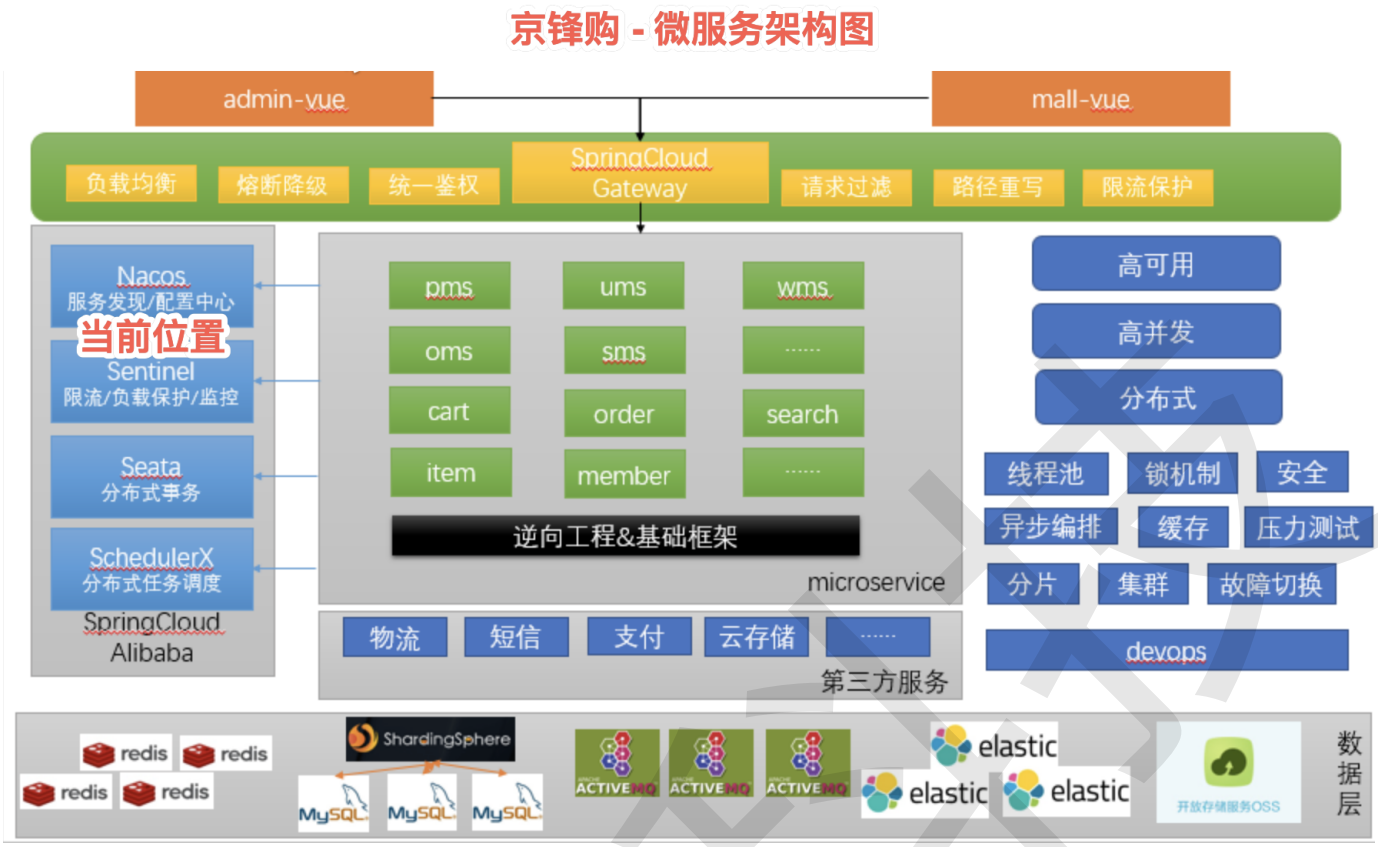
微服务项目-京锋购 03 - Spring Cloud Alibaba Sentinel

京锋购

京东 X 砺锋 = 京锋购商城



Spring Cloud Alibaba Sentinel

分布式流量哨兵。

Sentinel 是什么?

随着微服务的流行，服务和服务之间的稳定性变得越来越重要。Sentinel 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保护服务的稳定性。

Sentinel 的历史：

- 2012 年，Sentinel 诞生，主要功能为入口流量控制。

- 2013-2017 年，Sentinel 在阿里巴巴集团内部迅速发展，成为基础技术模块，覆盖了所有的核心场景。Sentinel 也因此积累了大量的流量归整场景以及生产实践。
- 2018 年，Sentinel 开源，并持续演进。
- 2019 年，Sentinel 朝着多语言扩展的方向不断探索，推出 C++ 原生版本，同时针对 Service Mesh 场景也推出了 Envoy 集群流量控制支持，以解决 Service Mesh 架构下多语言限流的问题。
- 2020 年，推出 Sentinel Go 版本，继续朝着云原生方向演进。

Sentinel 分为两个部分:

- 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。
- 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。

Sentinel 可以简单的分为 Sentinel 核心库和 Dashboard。

核心库不依赖 Dashboard，但是结合 Dashboard 可以取得最好的效果。

基本概念及作用

基本概念：

- **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，或由应用程序调用的其它应用提供的服务，甚至可以是一段代码。
  - 在接下来的文档中，我们都会用资源来描述代码块。
  - 只要通过 Sentinel API 定义的代码，就是资源，能够被 Sentinel 保护起来。
  - 大部分情况下，可以使用方法签名，URL，甚至服务名称作为资源名来标示资源。
- **规则**
  - 围绕资源的实时状态设定的规则，可以包括流量控制规则、熔断降级规则以及系统保护规则。
  - 所有规则可以动态实时调整。

主要作用：

1. 流量控制
2. 熔断降级
3. 系统负载保护

我们说的资源，可以是任何东西，服务，服务里的方法，甚至是一段代码。使用 Sentinel 来进行资源保护，主要分为几个步骤:

1. 定义资源
2. 定义规则
3. 检验规则是否生效

先把可能需要保护的资源定义好，之后再配置规则。

也可以理解为，只要有了资源，我们就可以在任何时候灵活地定义各种流量控制规则。

在编码的时候，只需要考虑这个代码是否需要保护，如果需要保护，就将之定义为一个资源。

快速开始

官方文档：https://github.com/alibaba/spring-cloud-alibaba/wiki/Sentinel

搭建Dashboard控制台

您可以从 release 页面 下载最新版本的控制台 jar 包。
https://github.com/alibaba/Sentinel/releases

下载的jar包，copy到一个没有空格或者中文的路径下，打开dos窗口切换到jar包所在目录。
执行：

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

如若8080端口冲突，可使用 -Dserver.port=新端口 进行设置。

启动后，可以在浏览器中访问 sentinel 控制台。
http://localhost:8080/
进入登录页面，管理页面用户名和密码：sentinel/sentinel

此时页面为空，这是因为还没有监控任何服务。

另外，sentinel是懒加载的，如果服务没有被访问，也看不到该服务信息。

改造nacos-consumer

1. 引入 sentinel 依赖

使用 group ID 为 `com.alibaba.cloud` 和 artifact ID 为 `spring-cloud-starter-alibaba-sentinel` 的 starter。

XML 复制代码

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

2. 在 application.yml 中添加配置：

YAML 复制代码

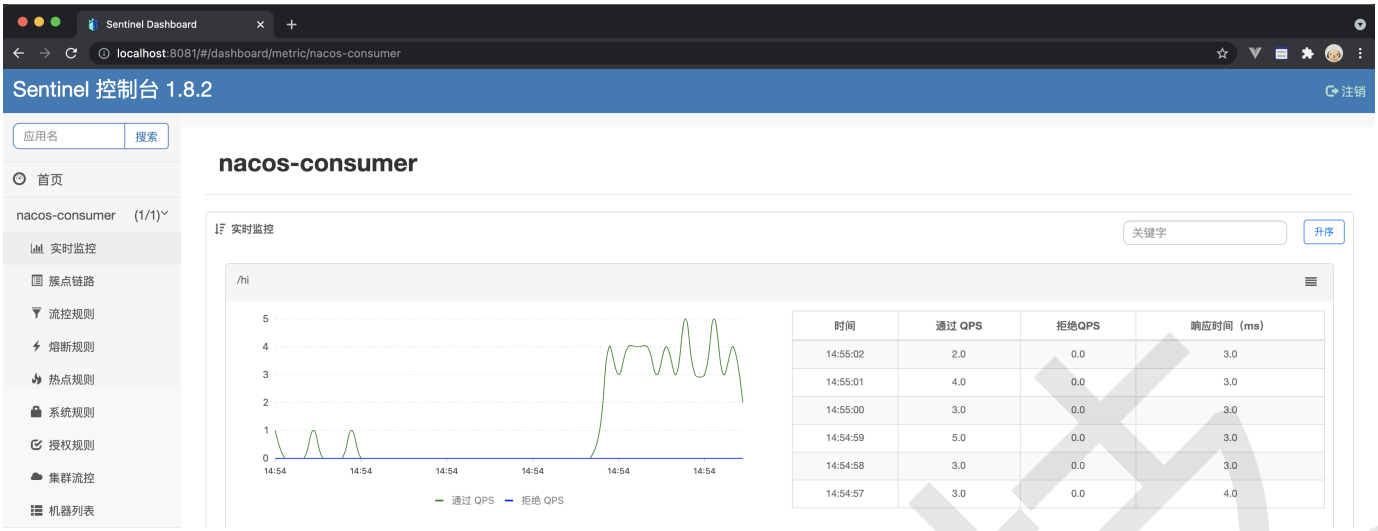
```yaml
# 指定dashboard地址
spring:
  cloud:
    sentinel:
      transport:
        port: 8719
        dashboard: localhost:8080
```

这里的 spring.cloud.sentinel.transport.port 端口配置会在应用对应的机器上启动一个 Http Server，该 Server 会与 Sentinel 控制台做交互。

比如 Sentinel 控制台添加了一个限流规则，会把规则数据 push 给这个 Http Server 接收，Http Server 再将规则注册到 Sentinel 中。

重启 nacos-consumer 工程，在浏览器中反复访问：http://localhost:18080/hi

再次查看sentinel控制台页面：



整合 Feign 组件

Sentinel 适配了 Feign 组件。使用分三步：

引入依赖

引入feign及sentinel的依赖

开启 sentinel 监控功能

YAML 复制代码

```yaml
feign:
  sentinel:
    enabled: true
```

新增接口

在 feign 新增 ProviderClient 接口，用于指定熔断类：

Java 复制代码

```java
package com.rushuni.nacosconsumer.feign;

import com.rushuni.nacosconsumer.fallback.ProviderFallback;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;

/**
 * @author rushuni
 * @date 2021/08/20
 */
@FeignClient(value = "nacos-provider", fallback = ProviderFallback.class)
public interface ProviderClient {

    @GetMapping("hello")
    String hello();
}
```

实现熔断类

消费模块中，添加 feign 接口的熔断类 ProviderFallback：

Java 复制代码

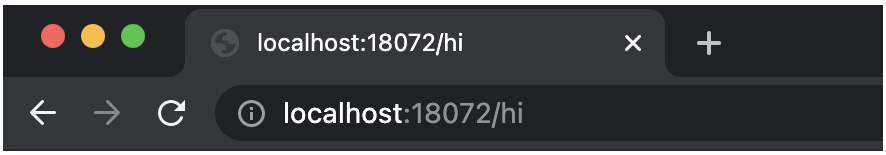
```java
@Component
public class ProviderFallback implements ProviderClient {

    @Override
    public String hello() {
        return "现在服务器忙，请稍后再试！";
    }
}
```

测试之前，先在服务提供方的 controller 方法中添加异常：

```java
@GetMapping("hello")
public String hello(){
    int i = 1 / 0;
    return "hello " + name + ",
}
```

再重启 nacos-provider 和 nacos-consumer 服务。在浏览器中地址栏访问接口进行测试：



现在服务器忙，请稍后再试！