37 Sentinel 基本用法

更新时间：2019-08-01 09:31:40

> 如果不想在世界上虚度一生，那就要学习一辈子。
>
> ——高尔基

前面和大家聊了 Spring Cloud Alibaba 中的 Nacos 组件，主要带着大家实现了用 Nacos 做配置中心和注册中心，本文我们来继续学习 Spring Cloud Alibaba 中的另外一个组件 Sentinel 。

Sentinel 简介

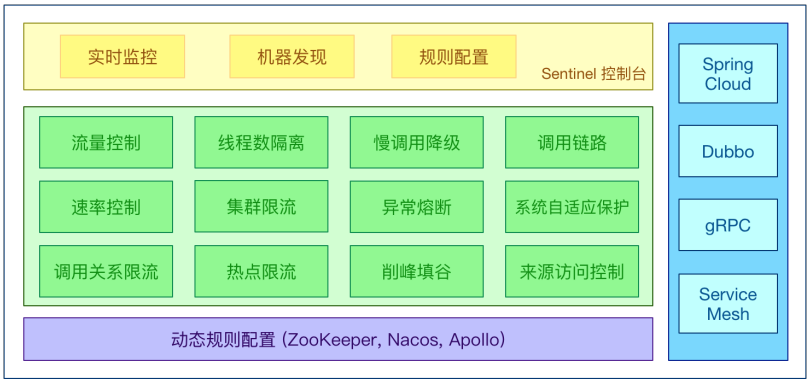
根据官方文档的介绍，Sentinel 被称作**分布式系统的流量防卫兵**，看到这个名字，我相信很多小伙伴都会想到 Hystrix ，可是前面的文章我们也已经说过，Hystrix 早已是明日黄花了，Netflix 都说了这个东西不再维护，因此我们在前面的文章中向大家介绍了 Spring Cloud 官方推荐的替代品 Resilience4j ，可是除了这个替代品 Resilience4j 之外， Sentinel 也是一个可以考虑的方案。

那么到底什么是 Sentinel 呢？它又有哪些特性呢？我们先来了解下。

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。根据官方文档的介绍，Sentinel 主要有如下特征：

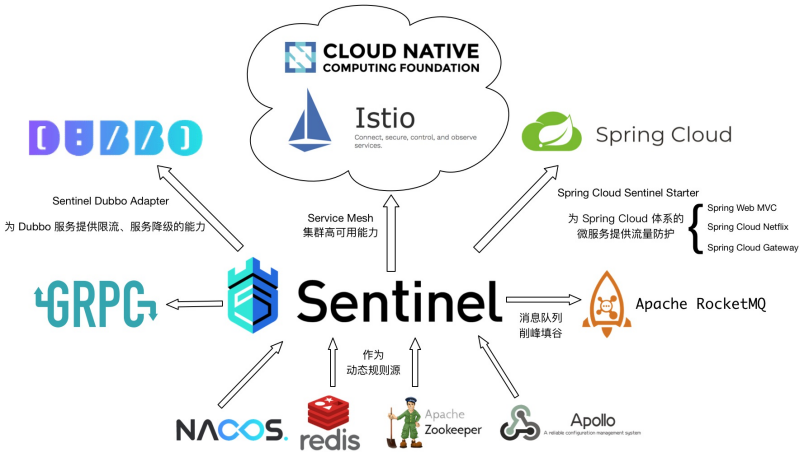
- 丰富的应用场景：Sentinel 承接了阿里巴巴近 10 年的双十一大促流量的核心场景，例如秒杀（即突发流量控制在系统容量可以承受的范围）、消息削峰填谷、集群流量控制、实时熔断下游不可用应用等。
- 完备的实时监控：Sentinel 同时提供实时的监控功能。开发者可以在控制台中看到接入应用的单台机器秒级数据，甚至 500 台以下规模的集群的汇总运行情况。
- 广泛的开源生态：Sentinel 提供开箱即用的与其它开源框架/库的整合模块，例如与 Spring Cloud、Dubbo、gRPC 的整合。开发者只需要引入相应的依赖并进行简单的配置即可快速地接入 Sentinel。
- 完善的 SPI 扩展点：Sentinel 提供简单易用、完善的 SPI 扩展接口。开发者可以通过实现扩展接口来快速地定制逻辑。例如定制规则管理、适配动态数据源等。

下面这张图完美诠释了 Sentinel 的核心功能：



可以看到，曾经 Hystrix 支持的功能，这里都有！并且支持的更多！这也是去年以来，有读者在微信上老问我 Netflix 要跑路了 ，Spring Cloud 还值不值得学？我说没问题，学你的，多少公司想给 Spring Cloud 做实现呢！不用担心，学，没错！

当然大家知道，现在不管玩什么，都讲究一个**生态**，没有生态，也就意味着其他的框架都不和你玩，那你这个框架的价值就会大打折扣！那么 Sentinel 的生态又是怎么样的呢？我们来看一张来自 Sentinel 官方文档的生态图：



从这张生态图中可以看到，Sentinel 对目前主流的服务都有很好的支持，都能方便的整合到其中，例如 Spring Cloud、Redis、Nacos、Apollo、RocketMQ 等。

从整体上来说，Sentinel 可以分为两个部分:

- 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。

好了，这是我们对 Sentinel 的简单介绍，接下来，我们就来通过一个简单的例子来感受下 Sentinel 的基本用法。

基本用法

Sentinel 仪表盘安装

为了方便看到 Sentinel 的工作效果，我们需要首先安装好 Sentinel 仪表盘，官方提供的 Sentinel 仪表盘就是一个普通的 Spring Boot 工程，我们先将之下载下来：
下载下来之后，直接按照 Spring Boot 项目启动的套路来运行即可。如下：

```
java -jar sentinel-dashboard-1.6.1.jar
```

不过官方目前发布的 jar 并非是最新版，如果大家想使用最新版的 sentinel-dashboard ，可以下载最新的源码然后编译运行：

- Sentinel Dashboard 最新源码(截止本文写作时最新源码)

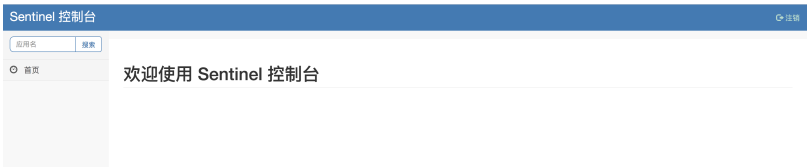
下载完成后，解压，然后执行如下命令进行打包（如果需要修改启动端口等参数，可以按照 Spring Boot 项目的方式去修改，也可以在项目启动时指定相关参数，总之，这就是一个普通的 Spring Boot 工程，所有东西都按照 Spring Boot 的套路来即可）：

```
mvn clean package
```

不过由于打包时候要下载的东西比较多，打包过程比较慢，因此需要多等一会。打包成功后，进入到解压目录的 sentinel-dashboard/target 目录下，可以看到我们刚刚生成的 jar ，然后执行 `java -jar sentinel-dashboard.jar` 命令启动项目。

当 sentinel-dashboard 启动成功后，浏览器输入 `http://localhost:8080` ，就可以看到 Sentinel 的控制台登录页面：

默认的用户名/密码都是 sentinel ，登录成功后的界面如下：



如此之后，我们的 Sentinel 控制台就算安装成功了，接下来我们就可以开始创建 Spring Boot 项目了。

整合 Spring Boot

接下来我们来在 Spring Boot 项目中整合 Sentinel。

首先创建一个 Spring Boot 项目，创建时引入 Web 依赖即可，创建成功之后，手动引入 Sentinel 依赖，最终的 pom.xml 文件如下图：

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
        <version>0.9.0.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

项目创建成功之后，首先在 application.properties 中添加如下配置：

```
spring.cloud.sentinel.transport.dashboard=localhost:8080
spring.application.name=sentinel
server.port=8081
```

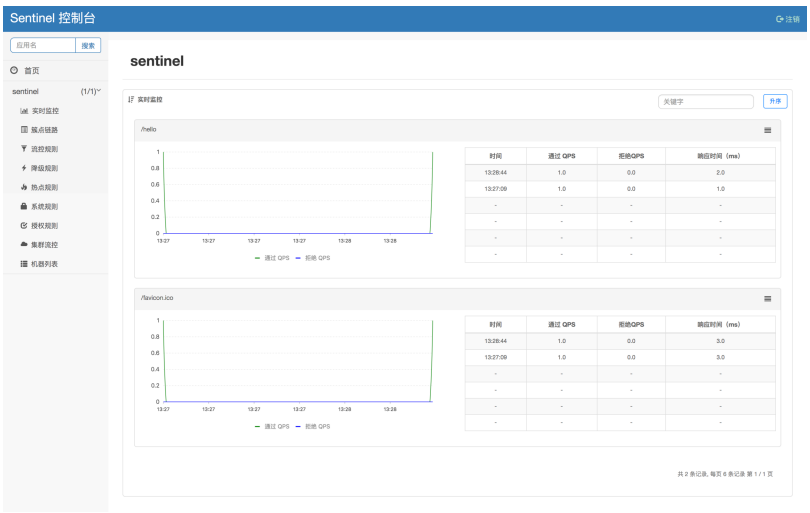
- 第一行配置表示指定 sentinel-dashborad 的位置
- 第二行和第三行配置分别表示指定服务的名字以及当前项目的端口号

配置完成后，我们再来创建一个 HelloController ，如下：

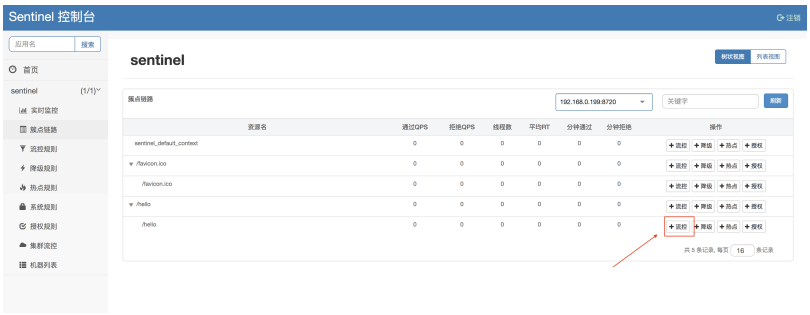
```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello() {
        return "hello spring boot!";
    }
}
```

配置玩车后，启动项目。

项目启动成功之后，我们先来访问 /hello 这个接口，访问完成之后，我们打开 sentinel-dashboard 管理页面，就可以看到我们的服务了，如下：



接下来我们来看一个简单的流量控制效果，我们点击左边的**簇点链路**按钮，然后点击右边的**流控**按钮：



然后进行流量控制配置，如下：

这里我们配置 QPS 单机阈值为 5，表示1秒内可以处理 5 个请求，超过数量的请求将进行排队等待，等待超时时间是 1 s，即 1 s 之内还没处理，就会抛出超时。

配置完成后，在流控规则中可以看到我们刚刚添加的配置：

然后我们在刚刚创建的 Spring Boot 工程中，写一个简单的单元测试：

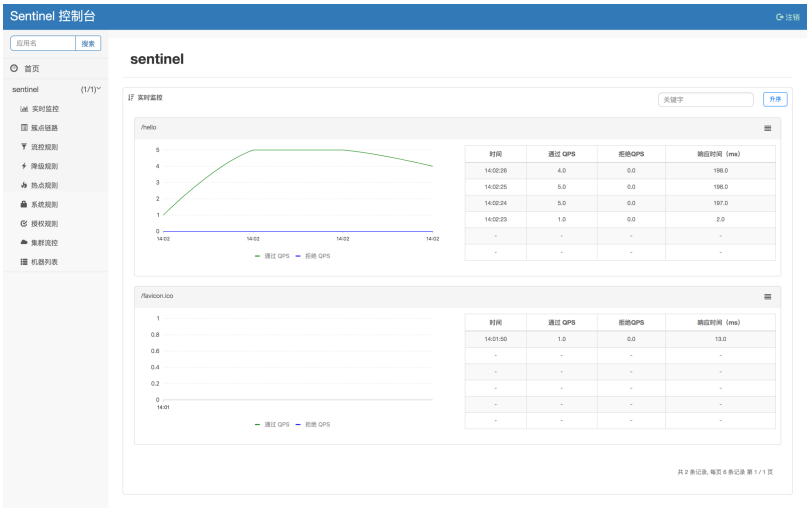
```
@RunWith(SpringRunner.class)
@SpringBootTest
public class SentinelApplicationTests {

    @Test
    public void contextLoads() {
        RestTemplate restTemplate = new RestTemplate();
        for (int i = 0; i < 15; i++) {
            String s = restTemplate.getForObject("http://localhost:8081/hello", String.class);
            System.out.println(i + ":" + s+":"+new Date());
        }
    }
}
```

执行这段代码，我们连续向刚刚的 Spring Boot 工程发送 15 个请求，查看请求日志，如下：

```
0:hello spring boot!:Sat Jun 08 14:02:23 CST 2019
1:hello spring boot!:Sat Jun 08 14:02:24 CST 2019
2:hello spring boot!:Sat Jun 08 14:02:24 CST 2019
3:hello spring boot!:Sat Jun 08 14:02:24 CST 2019
4:hello spring boot!:Sat Jun 08 14:02:24 CST 2019
5:hello spring boot!:Sat Jun 08 14:02:24 CST 2019
6:hello spring boot!:Sat Jun 08 14:02:25 CST 2019
7:hello spring boot!:Sat Jun 08 14:02:25 CST 2019
8:hello spring boot!:Sat Jun 08 14:02:25 CST 2019
9:hello spring boot!:Sat Jun 08 14:02:25 CST 2019
10:hello spring boot!:Sat Jun 08 14:02:25 CST 2019
11:hello spring boot!:Sat Jun 08 14:02:26 CST 2019
12:hello spring boot!:Sat Jun 08 14:02:26 CST 2019
13:hello spring boot!:Sat Jun 08 14:02:26 CST 2019
14:hello spring boot!:Sat Jun 08 14:02:26 CST 2019
```

从日志中我们可以看到，每秒的请求个数不超过 5 个。

我们再来看看 sentinel-dashboard 中显示的情况：



流控功能已经实现了。是不是比我们前面学过的 Resilience4j 实现流控要容易一些。

整合 Nacos

上篇文章我们学习了 Nacos ， Sentinel 中的限流规则也可以配置在 Nacos 中，我们一起来看下如何实现。

还是上面的项目，我们继续添加 Nacos 依赖：

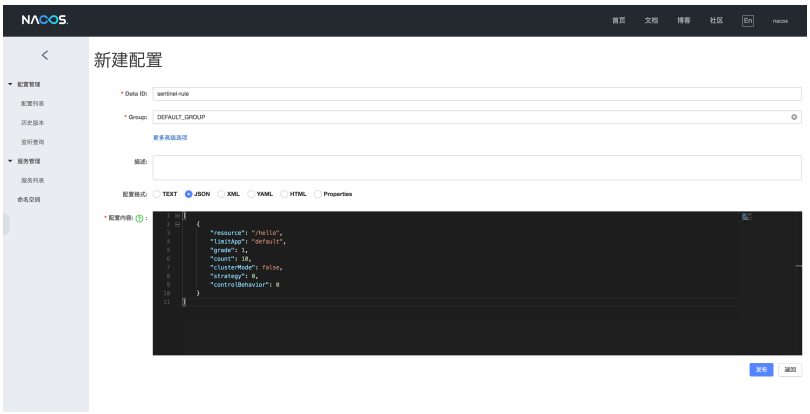
```
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-datasource-nacos</artifactId>
    <version>1.6.1</version>
</dependency>
```

然后再在 application.properties 中添加如下配置：

```
spring.cloud.sentinel.datasource.ds.nacos.server-addr=localhost:8848
spring.cloud.sentinel.datasource.ds.nacos.dataId=sentinel-rule
spring.cloud.sentinel.datasource.ds.nacos.groupId=DEFAULT_GROUP
spring.cloud.sentinel.datasource.ds.nacos.rule-type=flow
```

这里我们主要配置了四项，前三项的分别表示 Nacos 的地址，Nacos 中 dataId 的值，Nacos 中 groudId 的值。最后一项 rule-type 定义了规则类型，在 Sentinel 中，有不同的规则，例如热点参数限流、系统自适应限流等。

这里配置完成后，我们启动 Nacos ，并新建一项配置，如下：



这里的 dataId 和 Group 要和我们前面 application.properties 中配置的一致。至于配置内容，含义如下：

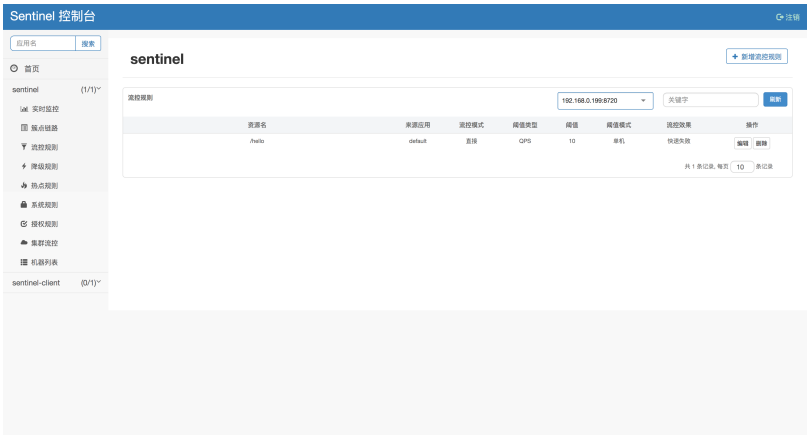
1. resource：这个指资源名称，一般来说就是我们配置的某一个接口或者某一个方法。
2. limitApp：流控针对的调用源，我们可以针对不同调用源给出不同的流控方案，default 则表示不区分调用源。
3. grade：阈值类型，0 表示并发线程数，1 表示根据 QPS。
4. count：单机阈值。
5. clusterMode：是否集群
6. strategy：流控模式
7. controlBehavior：流控效果

实际上，我们发现，这里的配置参数，和我们自己手动在 Sentinel 后台配置的参数一模一样！

这里配置完成后，我们再来重启一下我们的 Spring Boot 项目，注意观察启动日志：

```
INFO: log base dir is: /Users/sang/logs/csp/
INFO: log name use pid is: false
2019-06-08 22:13:09.172  INFO 27725 --- [           main] o.s.c.a.s.c.SentinelDataSourceHandler    : [Sentinel Starter] DataSource ds-sentinel-nacos-datasource load 1 FlowRule
2019-06-08 22:13:09.213  INFO 27725 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8081 (http) with context path ''
2019-06-08 22:13:09.218  INFO 27725 --- [           main] c.j.sentinel.SentinelApplication         : Started SentinelApplication in 2.381 seconds (JVM running for 3.103)
```

项目启动完成后，我们先在浏览器中访问项目的 /hello 接口，访问成功之后，我们刷新 Sentinel 后台管理页面，在流控规则一栏中，我们可以看到刚刚配置的流控规则：



看到这里，说明我们的配置已经成功了，接下来的测试，就和前文一模一样了，我就不再赘述了。

总结

本文主要向读者介绍了 Sentinel 的简单用法，相对于 Resilience4j，个人感觉 Sentinel 还是更加方便一些，特别是在 Hystrix 停止更新的当下，Sentinel 似乎更加耀眼啦！

参考资料

1.Sentinel 官方文档

36 Nacos 基础用法

38 微服务常见面试题分析