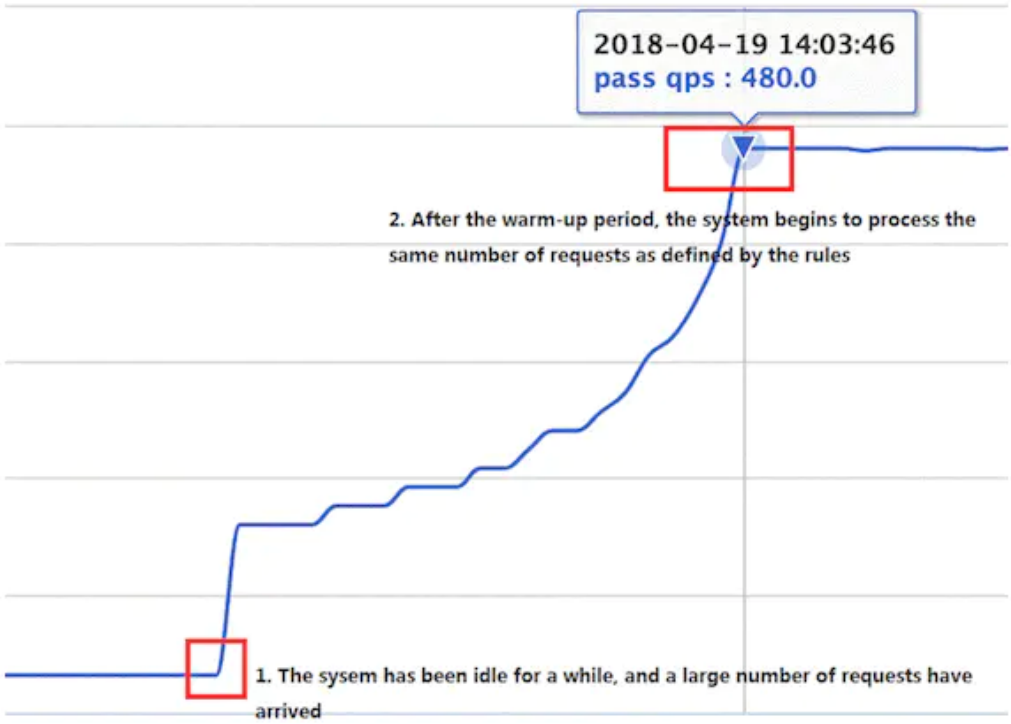
概述

`Sentinel`译为“哨兵”，顾名思义，面对您后台的大量服务/微服务，前置一个哨兵，但面对大量请求时，让后台服务有序被调用，但某些服务的不可用时，采用服务熔断降级等措施，让系统仍能平稳运行，不至于造成系统雪崩，典型应用场景：

1. MQ中消息在某些时间段（比如行情交易的高峰期，秒杀期等）消息并发量非常大时，通过`Sentinel`起到“削峰填谷”的作用；
2. 某个业务服务非常复杂，需要调用大量微服务，其中某服务不可用时，不影响整体业务运行，如提交某个订单，需要调用诸如验证库存，验证优惠金额，支付，验证手机号等，其中验证手机号服务不可用时，采用降级的方式让其通过，不影响整个提交订单的业务；
3. 上述订单业务提交时，依赖的下游应用控制线程数，请求上下文超过阈值时，新的请求立即拒绝，即针对流控，可基于QPS或线程数在某些业务场景下，都会有用，如下就是一个qps瞬时拉大时，通过流量缓慢增加，避免系统被压垮的情况：



image

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 具有以下特征:

- **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、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。您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。您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`扩展点。您可以通过实现扩展点，快速的定制逻辑。例如定制规则管理、适配数据源等。

`Sentinel`分为两部分：

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，该模块目前基于SpringBoot运行；

部署

在使用`Sentinel`之前，我们首先需部署`Sentinel Dashborad`，下载最新版的`Sentinel Dashborad`，通过以下命令运行：

```
java -Dserver.port=8088 -jar sentinel-dashboard-1.3.0.jar
```

通过`server.port`执行程序运行端口号，通过`http://localhost:8088`打开控制台，如下：

image

至此部署完成，非常简单！

项目中使用Sentinel

Sentinel针对各个主流框架都提供了适配（包括Servlet，Dubbo，SpringBoot/SpringCloud，gRPC，RocketMQ等），本文以`SpringBoot2`举例（通过笔者测试发现，SpringBoot 1.x支持不好，自定义流控规则不可用），首先我们需要在`SpringBoot2`的配置文件中指定Sentinel连接的控制台地址和项目名，即`application.yml`文件，如下：

```yaml
project:
  name: 在控制台显示的项目名
spring:
  cloud:
    sentinel:
      transport:
        dashboard: 192.168.1.154:8088
```

在项目中加入依赖：

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
    <version>0.2.0.RELEASE</version>
</dependency>
```

启动`SpringBoot2`后，可以在簇点链路页面看到项目的各个服务（首次访问服务时会被出现在列表中，这些服务在Sentinel中被称为资源），接着，你就可以针对这些资源进行流控，降级，热点，授权等操作。

流量控制与规则扩展

在控制台可通过**流控规则**菜单定义某资源的流控规则，不过这个定义只在内存中，但控制台重启后，随之消失，所以我们一般在项目中通过配置文件来定义流控规则，编写一个流控数据源，如下：

```
package com.sumscope.study.springboot2.service;

import java.net.URLDecoder;
import java.util.List;

import com.alibaba.csp.sentinel.datasource.Converter;
import com.alibaba.csp.sentinel.datasource.ReadableDataSource;
import com.alibaba.csp.sentinel.init.InitFunc;
import com.alibaba.csp.sentinel.datasource.FileRefreshableDataSource;
import com.alibaba.csp.sentinel.property.PropertyListener;
import com.alibaba.csp.sentinel.property.SentinelProperty;
import com.alibaba.csp.sentinel.slots.block.Rule;
import com.alibaba.csp.sentinel.slots.block.degrade.DegradeRule;
import com.alibaba.csp.sentinel.slots.block.degrade.DegradeRuleManager;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
import com.alibaba.csp.sentinel.slots.system.SystemRule;
import com.alibaba.csp.sentinel.slots.system.SystemRuleManager;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.TypeReference;
public class FileDataSourceInit implements InitFunc{
    private Converter<String, List<FlowRule>> flowRuleListParser = source ->
JSON.parseObject(source,
        new TypeReference<List<FlowRule>>() {});
    private Converter<String, List<DegradeRule>> degradeRuleListParser = source
-> JSON.parseObject(source,
        new TypeReference<List<DegradeRule>>() {});
```

```
    private Converter<String, List<SystemRule>> systemRuleListParser = source ->
JSON.parseObject(source,
        new TypeReference<List<SystemRule>>() {});
    @Override
    public void init() throws Exception {
        ClassLoader classLoader = getClass().getClassLoader();
        String flowRulePath =
URLDecoder.decode(classLoader.getResource("FlowRule.json").getFile(), "UTF-8");
        String degradeRulePath =
URLDecoder.decode(classLoader.getResource("DegradeRule.json").getFile(), "UTF-
8");
        String systemRulePath =
URLDecoder.decode(classLoader.getResource("SystemRule.json").getFile(), "UTF-
8");

        // Data source for FlowRule
        FileRefreshableDataSource<List<FlowRule>> flowRuleDataSource = new
FileRefreshableDataSource<>(
            flowRulePath, flowRuleListParser);
        FlowRuleManager.register2Property(flowRuleDataSource.getProperty());

        // Data source for DegradeRule
        FileRefreshableDataSource<List<DegradeRule>> degradeRuleDataSource
            = new FileRefreshableDataSource<>(
            degradeRulePath, degradeRuleListParser);

 DegradeRuleManager.register2Property(degradeRuleDataSource.getProperty());

        // Data source for SystemRule
        FileRefreshableDataSource<List<SystemRule>> systemRuleDataSource
            = new FileRefreshableDataSource<>(
            systemRulePath, systemRuleListParser);
        SystemRuleManager.register2Property(systemRuleDataSource.getProperty());
    }
}
```

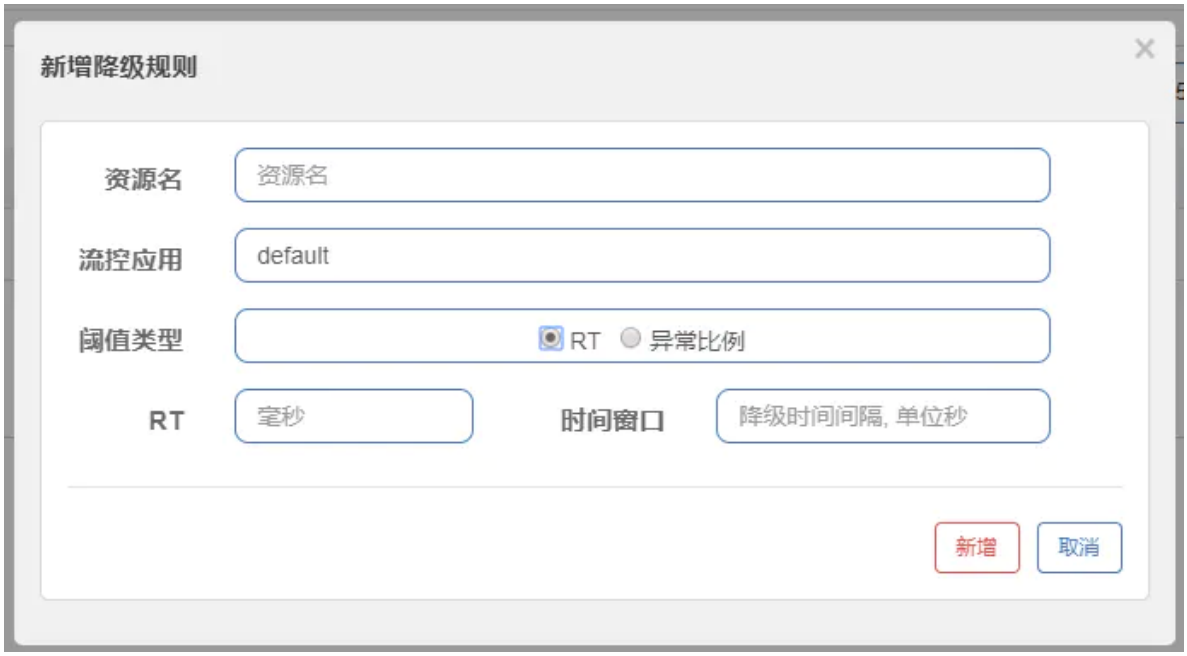
然后在项目的`resources`下新增目录`META-INF\services\com.alibaba.csp.sentinel.init.InitFunc`，并填写以上类的完成类名如`com.sumscope.study.springboot2.service.FileDataSourceInit`即可，这样，在您的classpath目录下通过`DegradeRule.json`,`FlowRule.json`,`SystemRule.json`分别来定义降级规则，流控规则和系统规则，比如我们定义一个流控规则，让test资源QPS为1，即1秒钟最多调用1次，如下：

```
[
  {
    "resource": "test",
    "controlBehavior": 0,
    "count": 1,
    "grade": 1,
    "limitApp": "default",
    "strategy": 0
  }
]
```

> 在Java代码中，通过`@SentinelResource("test")`来定义服务对应的资源名，如果不指数，URI即为资源名。

熔断降级

降级规则定义如下：



image

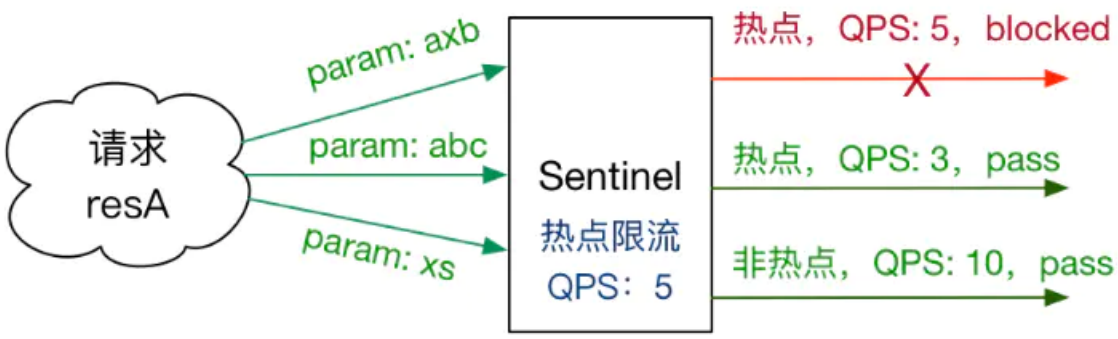
可基于RT（平均响应时间）或异常比例两种方式来定义，其中RT单位为毫秒。

指定RT时，当资源的平均响应时间超过阈值（DegradeRule 中的 count，以 ms 为单位）之后，资源进入准降级状态。接下来如果持续进入 5 个请求，它们的 RT 都持续超过这个阈值，那么在接下的时间窗口（DegradeRule 中的 timeWindow，以 s 为单位）之内，对这个方法的调用都会自动地返回。

指定异常时，当资源的每秒异常总数占通过总数的比值超过阈值（DegradeRule 中的 count）之后，资源进入降级状态，即在接下的时间窗口（DegradeRule 中的 timeWindow，以 s 为单位）之内，对这个方法的调用都会自动地返回。

热点参数限流

也可对经常访问的数据进行限流，如某个商品或某个用户等，如下图：



image

`Sentinel` 利用 `LRU` 策略，结合底层的滑动窗口机制来实现热点参数统计。LRU 策略可以统计单位时间内，最近最常访问的热点参数，而滑动窗口机制可以帮助统计每个参数的 QPS，热点参数限流目前只支持QPS模式。

黑白名单

可通过定义策略（黑名单或白名单）限定资源的调用方是否让其通过，以下是代码定义白名单规则：

```
AuthorityRule rule = new AuthorityRule();
rule.setResource("test");
rule.setStrategy(RuleConstant.AUTHORITY_WHITE);
rule.setLimitApp("appA,appB");
AuthorityRuleManager.loadRules(Collections.singletonList(rule));
```

实时监控

在控制台我们可以实时看到每个资源的qps情况如下图：

时间	p_qps	b_qps	rt(ms)
15:44:22	4.0	0.0	0.0
15:44:21	6.0	0.0	0.0
15:44:20	5.0	0.0	0.0
15:44:19	1.0	0.0	0.0
-	-	-	-
-	-	-	-