链滴

SpringCloud Alibaba 微服务实战五 - 限流熔断

作者：jianzh5

原文链接：https://ld246.com/article/1576035826460

来源网站：链滴

导读：本篇作为SpringCloud Alibaba微服务实战系列的第五篇，主要内容是使用Sentinel给微服务上限流熔断功能，防止异常情况拖垮应用服务。系列文章，欢迎持续关注。

简介

Sentinel是面向分布式服务框架的轻量级流量控制框架,主要以流量为切入点,从流量控制,熔断降级,系负载保护等多个维度来维护系统的稳定性。

在SpringCloud体系中，sentinel主要是为了替换原Hystrix的功能，与Hystrix相比，sentinel的隔离别更加精细，提供的Dashboard可以在线更改限流熔断规则，而且使用也越加方便。要了解更多详细息请移步至Sentinel官网。

基础准备

要使用Sentinel提供的限流熔断能力，需要先做如下准备：

- 安装Sentinel

这部分内容我已经在第一期SpringCloud Alibaba微服务实战一 - 基础环境准备中提过，大家可以翻查看。

- 引入Sentinel

在需要配置限流熔断服务的POM文件中引入Sentinel组件

```
<!--Sentinel-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
</dependency>
```

- 自定义资源　@SentinelResource

我们只需要在相关方法上加上@SentinelResource注解，让其可以成为sentinel识别的资源即可。如：

```
@GetMapping("/account/getByCode/{accountCode}")
@SentinelResource(value = "getByCode")
public ResultData<AccountDTO> getByCode(@PathVariable(value = "accountCode") String a
ccountCode){
    log.info("get account detail,accountCode is :{}",accountCode);
    AccountDTO accountDTO = accountService.selectByCode(accountCode);
    return ResultData.success(accountDTO);
}
```

- 在配置文件中添加sentinel的服务端地址

```
server:
  port: 8010
spring:
  application:
    name: account-service
  cloud:
    nacos:
      discovery:
        server-addr: 192.168.0.107:8848/
    sentinel:
      transport:
      # sentinel服务端地址
        dashboard: 192.168.0.107:8858
      # 取消延迟加载
      eager: true
```

经过以上几步我们准备好了使用Sentinel的基础环境，接下来我们看看限流熔断的具体配置。

限流

概念说明

生产者accout-service是一个核心服务，我们通过压测得出服务的最大负载能力为60。如果某个时间account-service的请求数飙升达到了600，那服务肯定就直接gg了。所以为了保护我们的accout-service我们会给它配置一个限流规则，如果每秒钟有超过60的请求那不好意思我直接丢掉不处理了，然后丢消费者一个异常，想拖垮我，哼，没门！。

总而言之，限流是通过限制调用方对自己的调用，起到保护自己系统的效果。

限流配置

理想是丰满的，现实是骨感的。由于本人对Jmeter之类的压测工具不是很精通所以为了方便测试，我就将accout-service的QPS单机阈值设置成5，如果每秒QPS超过5，直接丢弃。

这里的资源名就是我们使用@SentinelResource注解自定义的资源。

打开浏览器，快速刷新浏览器，当每秒请求书超过5时会看到如下错误：

在后端服务日志中你会看到如下的错误日志：

```
2019-12-10 14:22:31,948 ERROR [dispatcherServlet]:175 - Servlet.service() for servlet [dispatc
erServlet] in context with path [] threw exception [Request processing failed; nested exception
is java.lang.reflect.UndeclaredThrowableException] with root cause
com.alibaba.csp.sentinel.slots.block.flow.FlowException: null
```

不要慌，这说明我们的目的达到了，限流成功！

自定义异常

我们可以通过@SentinelResource中添加blockHandler参数，给其添加自定义异常方法。如：

```
@GetMapping("/account/getByCode/{accountCode}")
@SentinelResource(value = "getByCode",blockHandler = "handleException")
public ResultData<AccountDTO> getByCode(@PathVariable(value = "accountCode") String a
ccountCode){
    log.info("get account detail,accountCode is :{}",accountCode);
    AccountDTO accountDTO = accountService.selectByCode(accountCode);
    return ResultData.success(accountDTO);
```

```
}
/**
 * 自定义异常策略
 * 返回值和参数要跟目标函数一样，参数可以追加BlockException
 */
public ResultData<AccountDTO> handleException(String accountCode,BlockException except
on){
    log.info("flow exception{}",exception.getClass().getCanonicalName());
    return ResultData.fail(900,"达到阈值了,不要再访问了!");
}
```

注意，自定义的异常方法的参数和返回值要跟目标方法一样，参数可以追加BlockException

效果如下：

比之前的那个错误页优雅多了有木有！

持久化配置

由于Sentinel的配置默认是放在内存中的，每当应用重启或者sentinel重启都会丢失数据，我们这里用Nacos作为配置中心持久化限流配置。

- 修改pom文件，引入 sentinel-datasource-nacos组件

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

- 修改application.yml，配置sentinel的数据源

```
spring:
  cloud:
    sentinel:
      datasource:
        ds:
          nacos:
            server-addr: 10.0.10.48:8848
            data-id: ${spring.application.name}-sentinel
            group-id: DEFAULT_GROUP
            rule-type: flow
```

- 在nacos中建立限流配置 account-service-sentinel(配置格式设置成json)

```
[
    {
        "resource": "getByCode",
```

```
        "limitApp": "default",
        "grade": 1,
        "count": 3,
        "strategy": 0,
        "controlBehavior": 0,
        "clusterMode": false
    }
]
```

可以看到上面配置规则是一个数组类型，数组中的每个对象是针对每一个保护资源的配置对象，每个象中的属性解释如下：

resource：资源名，即限流规则的作用对象

limitApp：流控针对的调用来源，若为 default 则不区分调用来源

grade：限流阈值类型（QPS 或并发线程数）；0代表根据并发数量来限流，1代表根据QPS来进行流控制

count：限流阈值

strategy：调用关系限流策略

controlBehavior：流量控制效果（直接拒绝、Warm Up、匀速排队）

clusterMode：是否为集群模式

- 进入 sentinel查看dashboard，发现sentinel自动获取nacos的配置

- 频繁刷新浏览器调用接口，验证接口是否正常限流

熔断

概念说明

消费者order-service需要先调用product-service获取具体的product，然后再处理其他的业务逻辑但是这个product-service接口不是很稳定，经常抛出异常；或者是响应缓慢，导致order-service的应变慢；如果置之不理，order-service可能会被product-service拖垮。这时候为了保护order-service我们需要对product-service接口进行熔断。

原文链接：SpringCloud Alibaba 微服务实战五 - 限流熔断

一言以蔽之：熔断是通过限制自己对外部系统的调用， 起到节约响应时间、维护链路稳定的作用。

熔断配置

Sentinel中的熔断降级有三个降级策略：

- RT(平均响应时间)：

当资源的平均响应时间超过阈值之后，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 T 都持续超过这个阈值，那么在接下的时间窗口之内，对这个方法的调用都会自动抛出 DegradeExce tion 异常。在下一个时间窗口到来时, 会接着再放入5个请求, 再重复上面的判断.

- 异常比例

当资源的每秒异常总数占通过量的比值超过阈值之后，资源进入降级状态，即在接下的时间窗口之内对这个方法的调用都会自动地抛出DegradeException异常。异常比率的阈值范围是 [0.0, 1.0]，代表 % - 100%。

- 异常数

当资源近 1 分钟的异常数目超过阈值之后会进行熔断。

首先我们对原接口进行改造，让其直接抛出Runtimeexception：

```
@GetMapping("/product/getByCode/{productCode}")
@SentinelResource(value = "/product/getByCode",fallback = "fallbackHandler")
public ResultData<ProductDTO> getByCode(@PathVariable String productCode){
    log.info("get product detail,productCode is :{}",productCode);
    ProductDTO productDTO = productService.selectByCode(productCode);
    throw new RuntimeException("error");
//        return ResultData.success(productDTO);
}
```

这里我们将product-service设置如下的熔断规则：

如果/product/getByCode的异常率超过50%，那么接下来2秒内直接触发熔断降级，默认情况会抛出egradeException异常，如：

```
2019-12-10 19:35:53,764 ERROR [dispatcherServlet]:175 - Servlet.service() for servlet [dispatc
erServlet] in context with path [] threw exception [Request processing failed; nested exception
```

```
is java.lang.reflect.UndeclaredThrowableException] with root cause
com.alibaba.csp.sentinel.slots.block.degrade.DegradeException: null
```

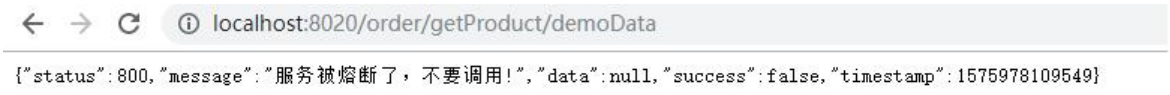
自定义异常

自定义熔断异常跟限流异常类似，我们使用fallback属性指定自定义异常的方法，如：

```
@SentinelResource(value = "/product/getByCode",fallback = "fallbackHandler")
public ResultData<ProductDTO> getByCode(@PathVariable String productCode){
 ...
}
/**
 * 自定义熔断异常
 * 返回值和参数要跟目标函数一样
 */
public ResultData<ProductDTO> fallbackHandler(String productCode){
    return ResultData.fail(800,"服务被熔断了，不要调用!");
}
```

注意，自定义的异常方法的参数和返回值要跟目标方法一样

效果如下：



持久化配置

- 引入　sentinel-datasource-nacos组件，跟限流一样配置即可
- 修改application.yml，配置sentinel的数据源

```
spring:
  cloud:
    sentinel:
      datasource:
        ds:
          nacos:
            server-addr: 192.168.0.106:8848
            data-id: ${spring.application.name}-sentinel-degrade
            group-id: DEFAULT_GROUP
            rule-type: degrade
```

- 在nacos中建立配置文件　product-service-sentinel-degrade,做如下配置

```
[
    {
    "resource": "/product/getByCode",
    "count": 0.5,
    "grade": 1,
    "passCount": 0,
```

```
    "timeWindow": 2
  }
]
```

可以看到上面配置规则是一个数组类型，数组中的每个对象是针对每一个保护资源的配置对象，每个象中的属性解释如下：

resource：资源名，即降级规则的作用对象

count：阈值

grade：降级模式 0:RT 1:异常比例 2:异常数

timeWindow：时间窗口（单位秒）

- 进入sentinel查看dashboard，发现sentinel自动获取nacos的配置

血与泪

大家在使用sentinel过程中如果出现Failed to fetch metric from 的错误,具体表现如下：

```
Failed to fetch metric from <http://192.168.136.1:8719/metric?startTime=1563865044000&endTime=1563865050000&refetch=false>
 (ConnectionException: Connection refused: no further information)
```

这个时候你需要去检查下sentinel控制台的服务列表，确认是否跟你ip一致。（我之前是装过虚拟机，entinel一致抓取的是我虚拟的ip，不知道为什么。。。）

如果发现监听的地址不对的话，可以在sentinel客户端配置中加入客户端ip配置

```
spring:
  cloud:
    sentinel:
      transport:
        client-ip: 192.168.0.108
```

至此我们已经给我们的微服务加上了限流熔断保护，再也不用担心异常流量的冲击，下游系统不稳定致自身服务不可用了。那么本期的“SpringCloud Alibaba微服务实战五 - 限流熔断”篇也就该结束，咱们下期有缘再见！

再见之前让我再求一波关注吧！

系列文章

- SpringCloud Alibaba微服务实战一 - 基础环境准备
- SpringCloud Alibaba微服务实战二 - 服务注册
- SpringCloud Alibaba微服务实战三 - 服务调用
- SpringCloud Alibaba微服务实战四 - 版本管理

原文链接：SpringCloud Alibaba 微服务实战五 - 限流熔断