【第60话：在你的项目中如何进行限流】

Hello 小伙伴们，这节课给大家讲解下，面试官问：“在你的项目中如何进行限流”，我们应该如何回答？

限流都出现在高并发场景。通过限流，我们可以很好地控制系统的QPS（Queries per Second），从而达到保护系统的目的。

我们需要明确一点，需要进行限流的项目基本上都是分布式项目。都需要进行限流了，架构师还设定项目为单体架构，那就说不过去了。目前分布式项目稍微老一点项目使用SOA，相对新一些的项目使用的都是微服务架构。那我们就说一说微服务架构中限流方案。

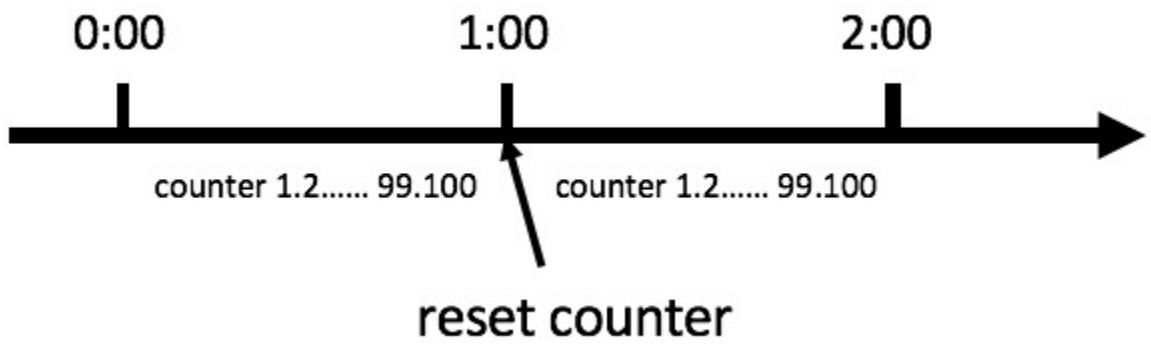
Java中微服务架构首选技术方案就是Spring Cloud，而Spring Cloud里面能够做限流的组件：

- Netflix Zuul：网飞的路由组件。使用Zuul+RateLimiter或里面的过滤器实现限流。但是Zuul这个组件在目前企业已经很少使用了。
- Gateway：API 网关。Gateway是目前使用频率比较高的组件，里面自带令牌桶算法实现限流。
- Alibaba Sentinel：带有可视化管理界面的限流组件。有着淘宝双11作为背书，是一个不错的限流组件。

所以我们在和面试官回答时看看自己项目使用了上面的哪个组件。为了让小伙伴们能够更加清晰理解到位，我们先来说说常见的限流算法。

计数器算法

以QPS（每秒查询率Queries-per-second）为100举例。
从第一个请求开始计时。每个请求让计数器加一。当到达100以后，其他的请求都拒绝。到下一秒开始时，计数清零，重新开始计数。



这种算法的问题是：如果1秒钟内前200ms请求数量已经到达了100，后面800ms中到来的所有请求都会被拒绝，这种情况称为“突刺现象”。

漏桶算法

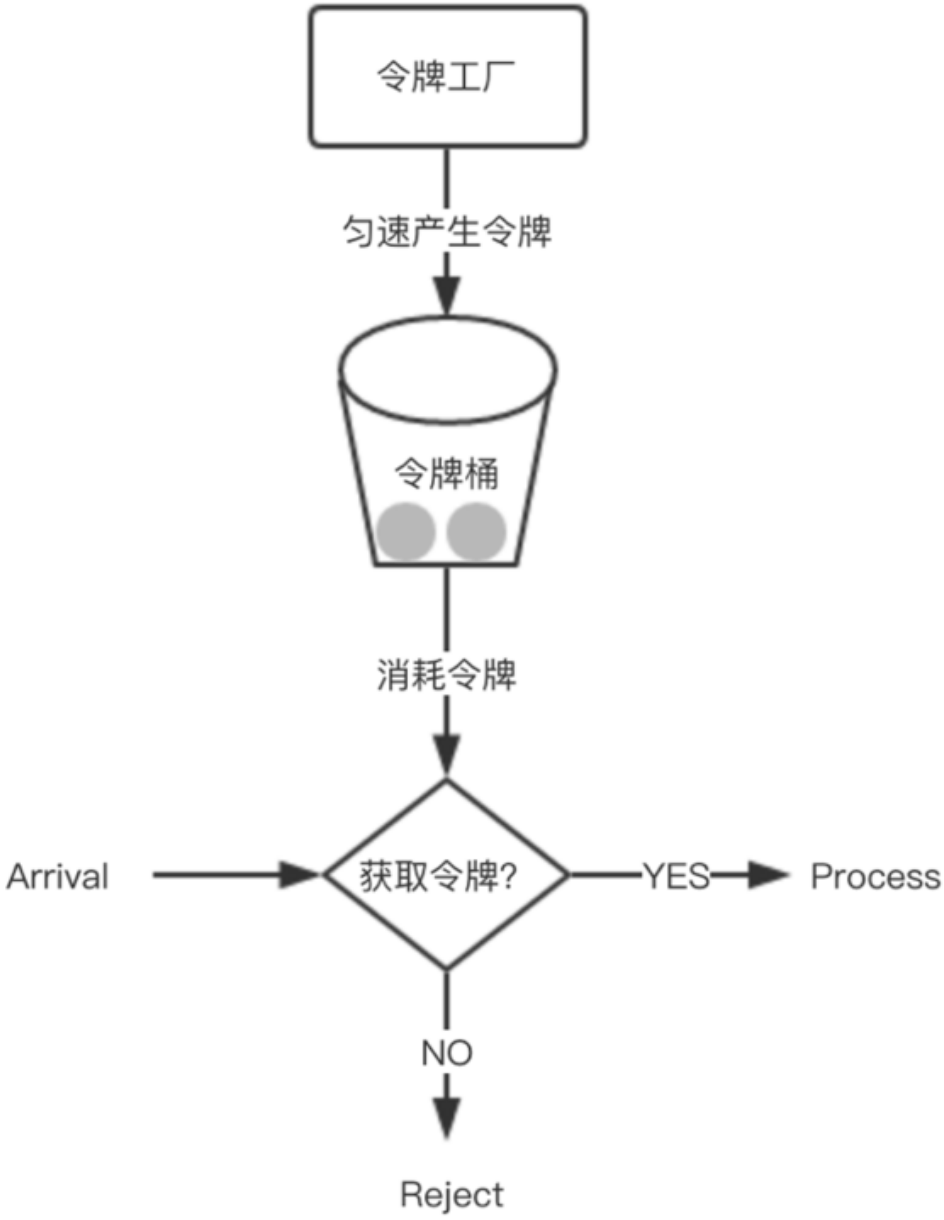
漏桶算法可以解决突刺现象。
和生活中边玩手机边充电的情况一样，假定玩手机时耗电量是一定的（即服务器处理能力），手机电池容量有上限（请求缓存队列容量有上限），充电方式不同则充电速率不同（请求并不均匀，有瞬间到来大量请求的可能），导致电池充满的时间也不同（请求缓存队列已满），电池充满了，即使继续插电玩手机，也不会有充电现象（后续请求拒绝处理）。假设电池充满，继续充电，手机会爆炸，那就需要时刻关注电池电量了。

令牌桶算法

令牌桶算法可以说是对漏桶算法的一种改进。

在桶中放令牌，请求获取令牌后才能继续执行。如果桶中没有令牌，请求可以选择进行等待或者直接拒绝。

由于桶中令牌是按照一定速率放置的，所以可以一定程度解决突发访问。如果桶中令牌最多有100个，QPS最大为100。



Gateway中使用RequestRateLimiter实现限流

RequestRateLimiter是基于Redis和Lua脚本实现的令牌桶算法。既然基于Redis记录令牌数据，那么应该有Spring Data Redis相关依赖。

在Gateway项目中不要忘记Spring-Data-Redis的依赖

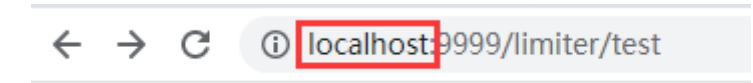
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

新建密钥解析器，com.bjsxt.resolver.MyKeyResolver

```
/**
 * 限流过滤器配置必要的类型。
```

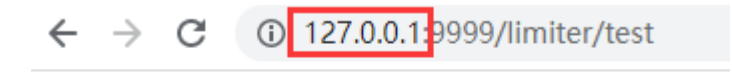
```
 * 令牌桶算法中令牌工厂的组件之一，用于生成一个个与客户端对应的令牌key。 提供一个能对应客户端的数据。
 * 如：IP，用户登录名等。
 *
 * 当前类型对象，需要Spring容器管理。
 */
@Component
public class MyKeyResolver implements KeyResolver {
    @Override
    public Mono<String> resolve(ServerWebExchange exchange) {
        String ip = exchange.getRequest() // 获取请求对象
                .getRemoteAddress() // 获取客户端地址对象 InetSocketAddress
                .getAddress() // 获取客户端地址对象 InetAddress
                .getHostAddress(); // 获取客户端的主机地址（IP或唯一的主机名）
        return Mono.just(ip); // 创建返回结果对象
    }
}
```

解析器的返回结果，会影响到Redis中记录的令牌key。具体如下：



服务器AppService返回结果

```
127.0.0.1:6379> keys *request*
1) "request_rate_limiter.{0:0:0:0:0:0:0:1}.timestamp"
2) "request_rate_limiter.{0:0:0:0:0:0:0:1}.tokens"
```



服务器AppService返回结果

```
127.0.0.1:6379> keys *request*
1) "request_rate_limiter.{127.0.0.1}.timestamp"
2) "request_rate_limiter.{127.0.0.1}.tokens"
```

编辑配置文件

```
server:
  port: 9999
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
spring:
  application:
    name: cloud-gateway
  cloud:
    gateway:
      discovery:
        locator:
          enabled: false
```

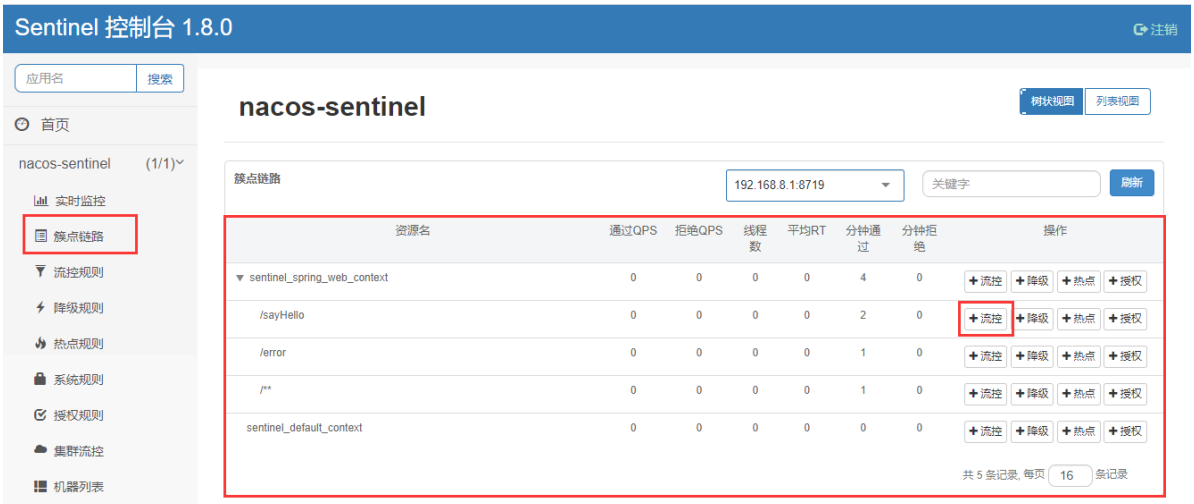
```
          lower-case-service-id: true
      routes:
        - id: rateLimiter
          uri: lb://application-service
          predicates:
            - Path=/limiter/**
          filters:
            - StripPrefix=1
            - name: RequestRateLimiter
              args:
                keyResolver: '#{@myKeyResolver}' # 表达式， #{} 从容器找对象，
@beanId
                # redis-rate-limiter是用于做令牌校验，和令牌生成的类型。gateway框架提供
了基于Redis的实现。
                redis-rate-limiter.replenishRate: 1 # 每秒令牌生成速率
                redis-rate-limiter.burstCapacity: 2 # 令牌桶容量上限
```

启动并测试

使用JMeter访问 http://localhost:9999/limiter/test 若干次，结果是，第一秒可处理2个请求（令牌桶上限），后续每秒可以处理1个请求（令牌生成速率）。

Sentinel实现限流

簇点链路显示了当前服务中所有URL.并可以对每个URL进行流控、降级、热点、授权操作。



流控规则选项解释

资源名：标识资源的唯一名称，默认为请求路径，也可以在客户端中使用@SentinelResource配置。

针对来源：Sentinel可以针对服务调用者进行限流，填写微服务名称即spring.application.name，默认为default，不区分来源。

阈值类型：

QPS（Queries-per-second，每秒钟的请求数量）。当调用该api的QPS达到阈值的时候，进行限流。

线程数：当调用该api的线程数达到阈值的时候，进行限流。

单机阈值:定义QPS或线程数的具体控制阈值。

是否集群:默认不集群。

流控模式

直接:当api调用达到限流条件的时，直接限流。

关联:当关联的资源请求达到阈值的时候，限流自己。

链路:只记录指定链路上的流量（指定资源从入口资源进来的流量，如果达到阈值，则进行限流）。

流控效果

快速失败:直接失败。

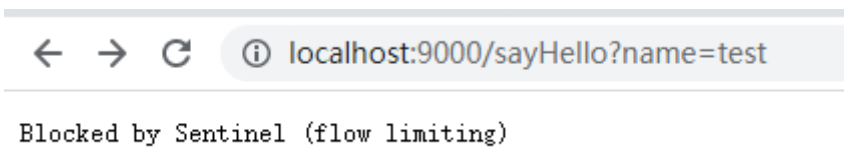
Warm Up（预热）:根据codeFactor（冷加载因子，默认值为3）的值，从阈值/codeFactor（除法），经过预热时长，才达到设置的QPS阈值。

排队等待:匀速排队，让请求匀速通过，阈值类型必须设置为QPS，否则无效。

快速失败方案

按照下图所示，填写单机阈值为2，并点击新增按钮。会在Sentinel Dashboard中流控规则中出现我们建立的规则。

在浏览器中快速访问 http://localhost:9000/sayHello?name=test 3次，当1秒内访问超过2次后会直接决绝。

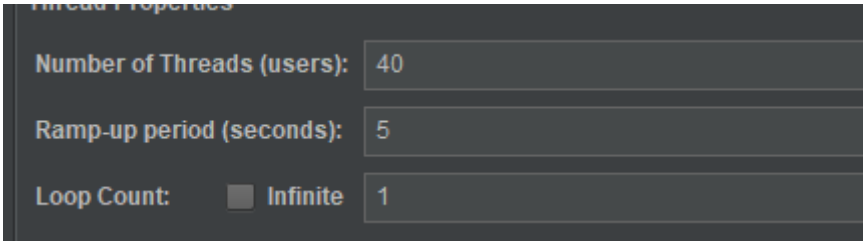


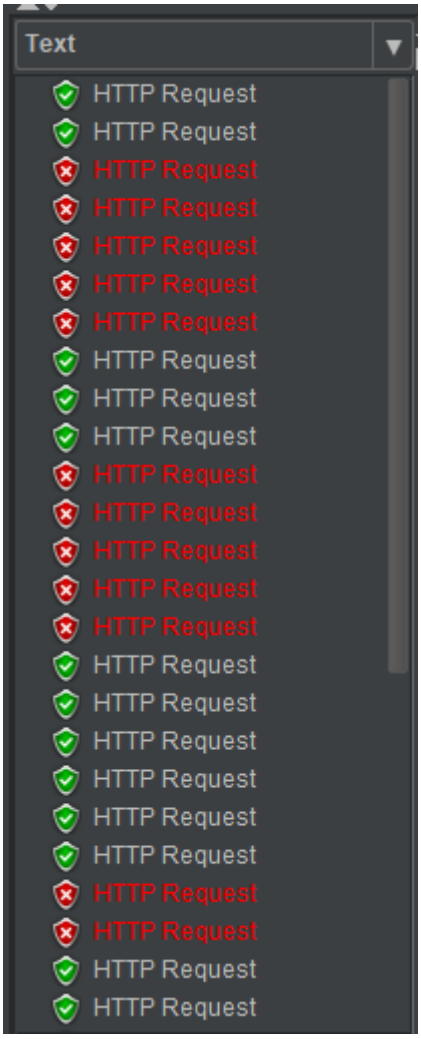

```
Blocked by Sentinel (flow limiting)
```

预热（warm up）方案

warm up 模式可以防止突然出现大量请求。按照下图单机阈值设置6，预热时长内（1秒内）QPS限制是：阈值/3，经过预热时长QPS限制到达阈值。会在Sentinel Dashboard中流控规则中出现我们建立的规则。

使用Apache jmeter进行测试。设置5秒40个请求。会发现前期只能访问2个其他请求直接决绝，后面逐渐能访问6个。





排队等待方案

排队等待，只要请求不超过设置的超时时间，就会一直等待。除非已经超过超时时间了。会在Sentinel Dashboard中流控规则中出现我们建立的规则。设置如下图

修改控制器，方法中增加休眠时间，测试：

```
@GetMapping("/sayHello")
public String sayHello(String name){
    System.out.println("sentinel name = " + name);
     try {
        Thread.sleep(500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return "您好：" + name;
}
```