[**#**](https://pig4cloud.com/#pom-%E4%BE%9D%E8%B5%96) **POM 依赖**

这里一定要注意，是网关引入的redis-reactive，背压模式的redis。

<!--基于 reactive stream 的redis -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis-reactive</artifactId>

</dependency>

[**#**](https://pig4cloud.com/#%E9%85%8D%E7%BD%AE%E6%8C%89%E7%85%A7%E8%AF%B7%E6%B1%82ip-%E7%9A%84%E9%99%90%E6%B5%81) **配置按照请求IP 的限流**

spring:

cloud:

gateway:

routes:

- id: requestratelimiter\_route

uri: lb://pigx-upms

order: 10000

predicates:

- Path=/admin/\*\*

filters:

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 1 # 令牌桶的容积

redis-rate-limiter.burstCapacity: 3 # 流速 每秒

key-resolver: "#{@remoteAddrKeyResolver}" #SPEL表达式去的对应的bean

- StripPrefix=1

* 配置bean，多维度限流量的入口 对应上边key-resolver 【我自己哦

/\*\*

\* 自定义限流标志的key，多个维度可以从这里入手

\* exchange对象中获取服务ID、请求信息，用户信息等

\*/

@Bean

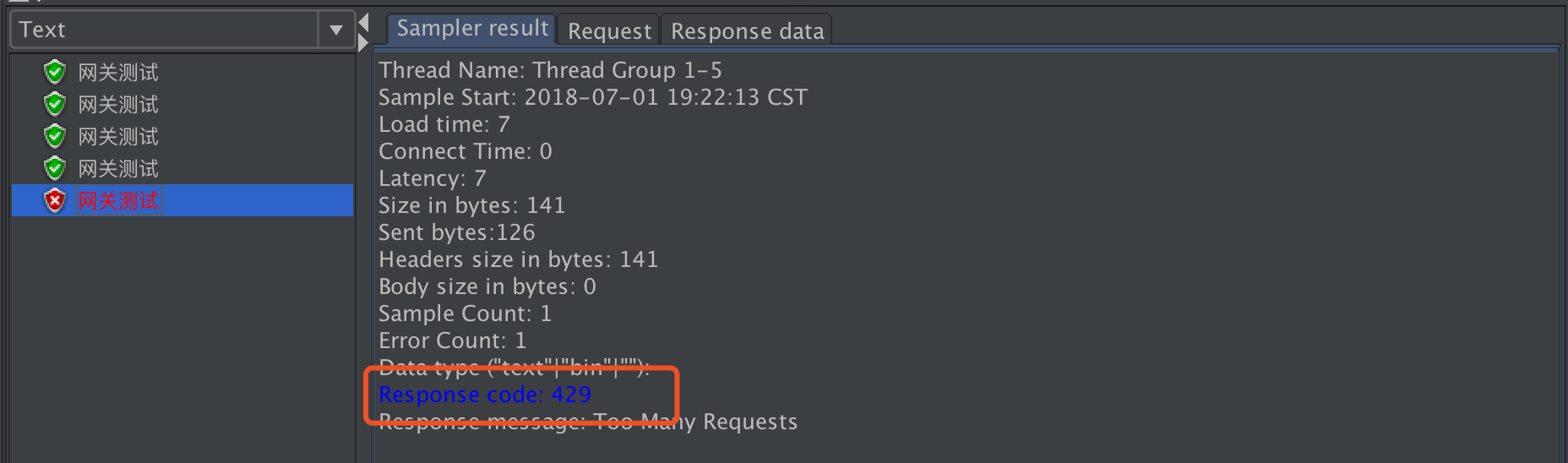
KeyResolver remoteAddrKeyResolver() {

return exchange -> Mono.just(exchange.getRequest().getRemoteAddress().getHostName());

}

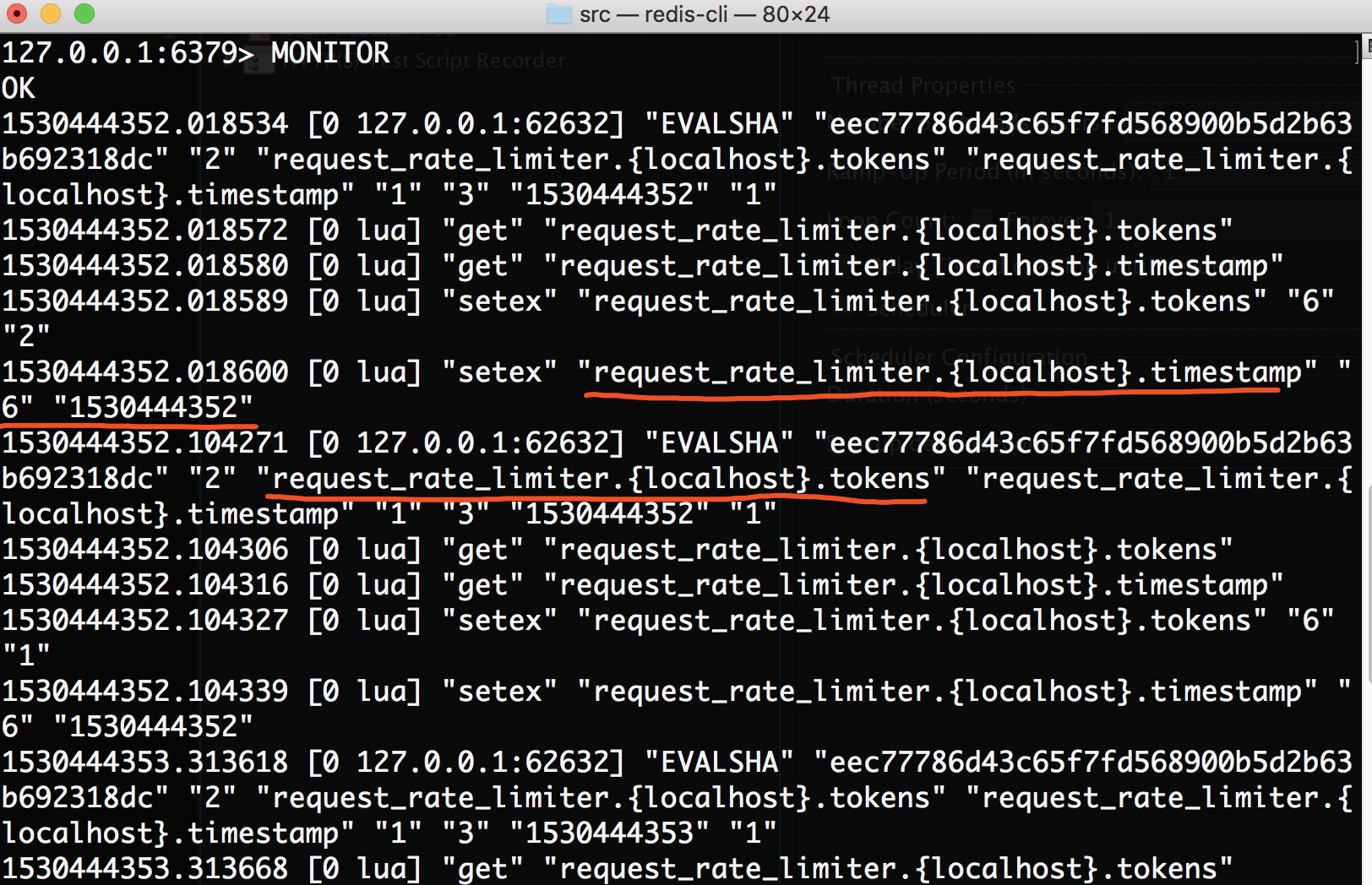
OK 完成。

[**#**](https://pig4cloud.com/#%E5%8E%8B%E5%8A%9B%E6%B5%8B%E8%AF%95) **压力测试**

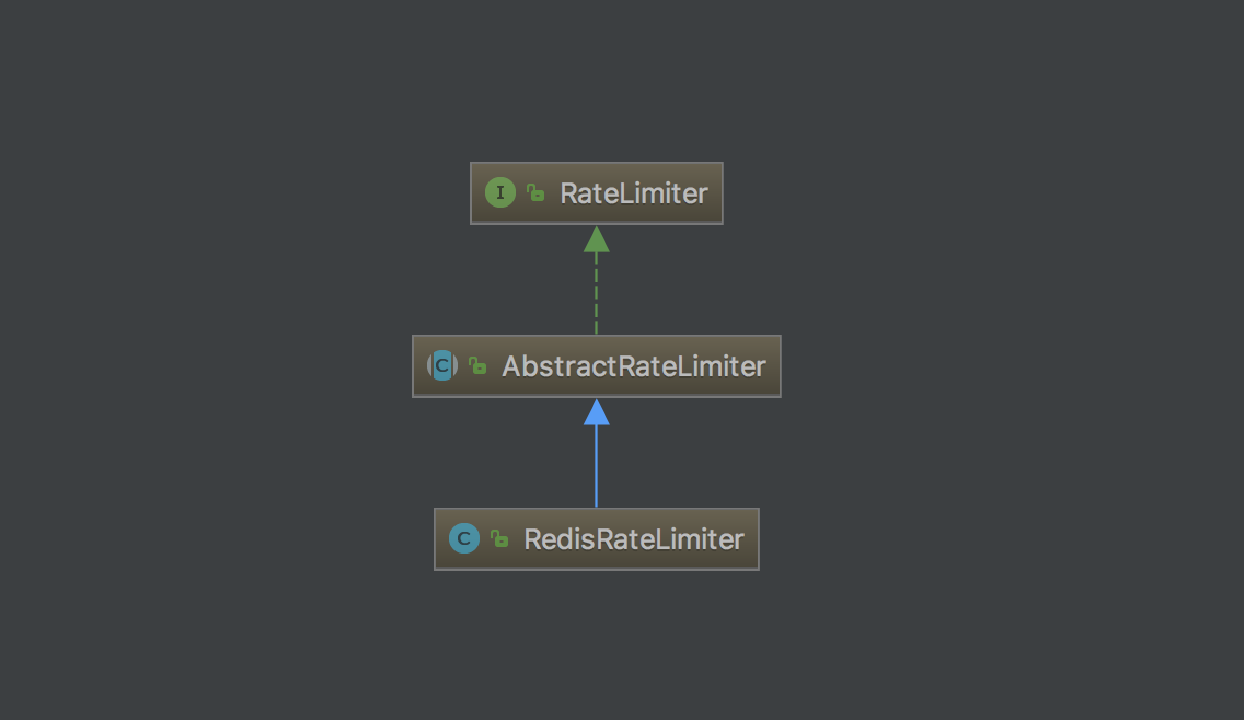
并发5个线程。  


[**#**](https://pig4cloud.com/#redis-%E6%95%B0%E6%8D%AE%E5%8F%98%E5%8C%96) **Redis 数据变化**

我们使用redis的**monitor** 命令，实时查看redis 的操作情况。  
会发现在redis中会操作两个key

* request\_rate\_limiter.{xxx}.timestamp
* request\_rate\_limiter.{xxx}.tokens 

[**#**](https://pig4cloud.com/#%E5%AE%9E%E7%8E%B0%E5%8E%9F%E7%90%86) **实现原理**



Spring Cloud Gateway 默认实现 Redis限流，如果扩展只需要实现ratelimter接口即可。

[**#**](https://pig4cloud.com/#redisratelimter-%E7%9A%84%E6%A0%B8%E5%BF%83%E4%BB%A3%E7%A0%81%EF%BC%8C%E5%88%A4%E6%96%AD%E6%98%AF%E5%90%A6%E5%8F%96%E5%88%B0%E4%BB%A4%E7%89%8C%E7%9A%84%E5%AE%9E%E7%8E%B0%EF%BC%8C%E9%80%9A%E8%BF%87%E8%B0%83%E7%94%A8-redis%E7%9A%84lua-%E8%84%9A%E6%9C%AC%E3%80%82) **RedisRateLimter 的核心代码，判断是否取到令牌的实现，通过调用 redis的LUA 脚本。**

public Mono<Response> isAllowed(String routeId, String id) {

Config routeConfig = getConfig().getOrDefault(routeId, defaultConfig);

int replenishRate = routeConfig.getReplenishRate();

int burstCapacity = routeConfig.getBurstCapacity();

try {

List<String> keys = getKeys(id);

returns unixtime in seconds.

List<String> scriptArgs = Arrays.asList(replenishRate + "", burstCapacity + "",

Instant.now().getEpochSecond() + "", "1");

// 这里是核心，执行redis 的LUA 脚本。

Flux<List<Long>> flux =

this.redisTemplate.execute(this.script, keys, scriptArgs);

return flux.onErrorResume(throwable -> Flux.just(Arrays.asList(1L, -1L)))

.reduce(new ArrayList<Long>(), (longs, l) -> {

longs.addAll(l);

return longs;

}) .map(results -> {

boolean allowed = results.get(0) == 1L;

Long tokensLeft = results.get(1);

Response response = new Response(allowed, getHeaders(routeConfig, tokensLeft));

if (log.isDebugEnabled()) {

log.debug("response: " + response);

}

return response;

});

}

catch (Exception e) {

log.error("Error determining if user allowed from redis", e);

}

return Mono.just(new Response(true, getHeaders(routeConfig, -1L)));

}

[**#**](https://pig4cloud.com/#lua-%E8%84%9A%E6%9C%AC) **LUA 脚本**

