

高性能分布式限流：Redis+Lua真香！

原创 北哥 BiggerBoy 2023-04-17 12:32 发表于北京

收录于合集
#BiggerBoy 61 #redis 11

文末附源码



微信扫一扫
关注该公众号

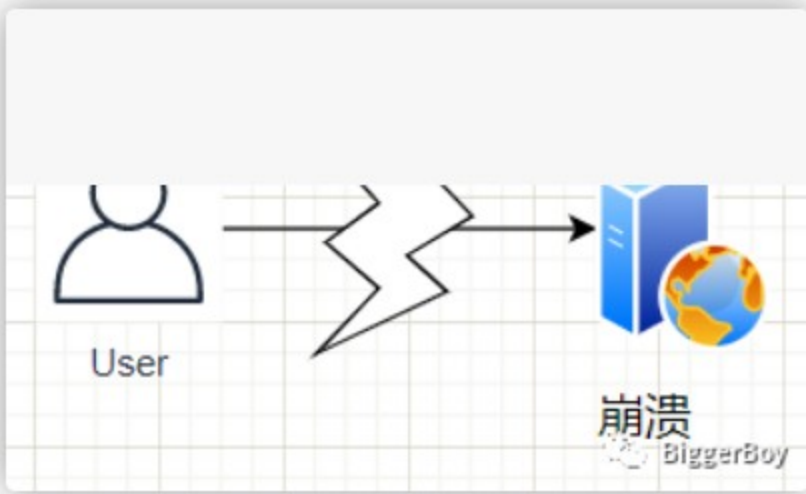
1 什么是限流？为什么要限流？

限流，这个词其实并不陌生，在我们生活中也随处可见。做核酸时，工作人员会在核酸检测点的空地上摆放着弯弯曲曲的围栏，人们排着队左拐右拐的往前移动，其实这么做的目的就是限流！因为核酸检测的窗口是有限的，一下子进那么多人，没那么多空间让人们站下，就会造成拥挤，甚至会造成事故。所以需要限流！

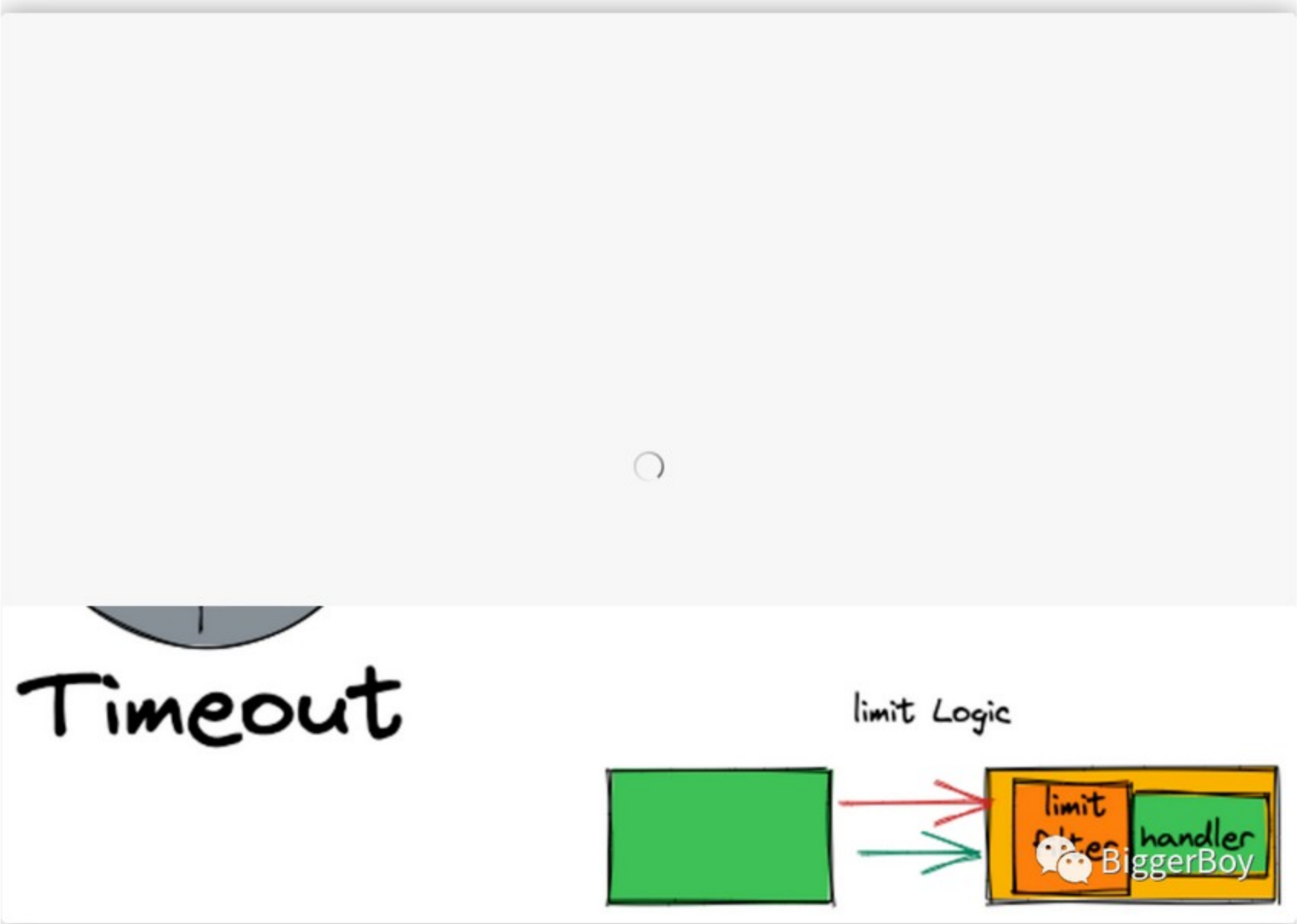


图源网络

同样的，我们的应用程序也是类似的，任何系统它处理请求的能力都是有限的，一旦请求多到超出系统的处理极限，系统就会崩溃。对于生产环境，崩溃是一个很大的生产事故，保不准就会给公司造成很大的损失，轻则赔款，重则判刑都是有可能的。所以今天我们就来聊一下如何实现高性能的限流。



重试、限流、熔断、降级被称为分布式系统高可用的四板斧。



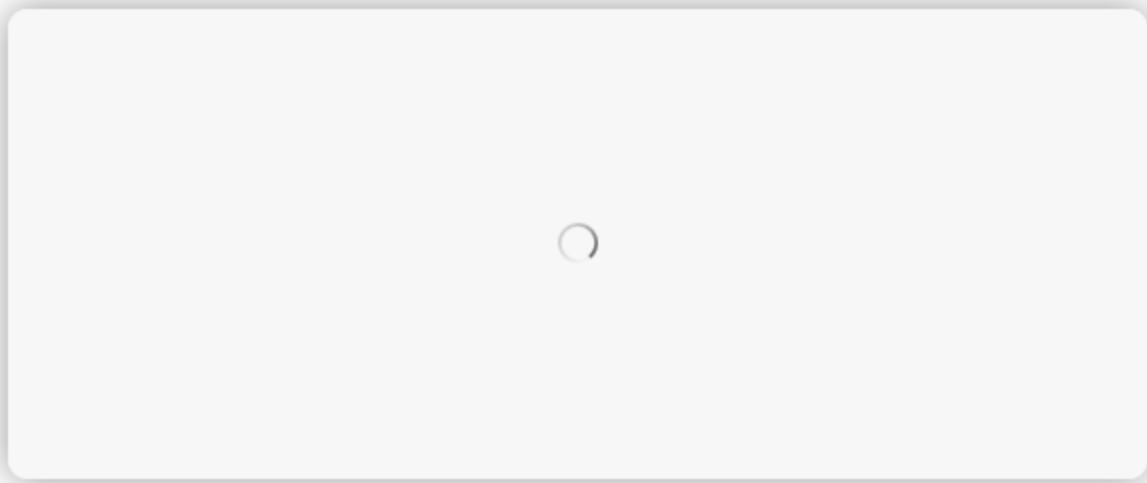
图源infoQ

- 不可避免地，
- 第一，我们一定要设置超时；
 - 第二，要在一些场景里面去考虑重试的逻辑；
 - 第三，考虑熔断的逻辑，不要被下游拖死；
 - 第四，一定要有限流的逻辑，不要被上游打死。

当今社会，互联网公司的流量巨大，系统上线前需要对系统全面的流量峰值评估，以判断系统所能承载的最大瞬时请求数，尤其是像各种秒杀促销活动，为了保证系统不被巨大的流量压垮，会事先评估系统最大请数，并设置限流逻辑，以便在系统流量到达设定的阈值时，拒绝掉这部分流量，从而确保系统不会崩溃。

限流会导致用户在短时间内（这个时间段是毫秒级的）系统不可用，假设系统设置的每秒流量阈值是100，理论上一秒内第101个及之后的请求都会被限流，相当于拒绝服务，下一秒进来的请求能正常被响应，这也就是为什么我们抢购时，一会儿能进页面一会儿

显示“请稍后”之类的提示语。相比于系统的短暂不可用，要比系统崩溃要好太多了。

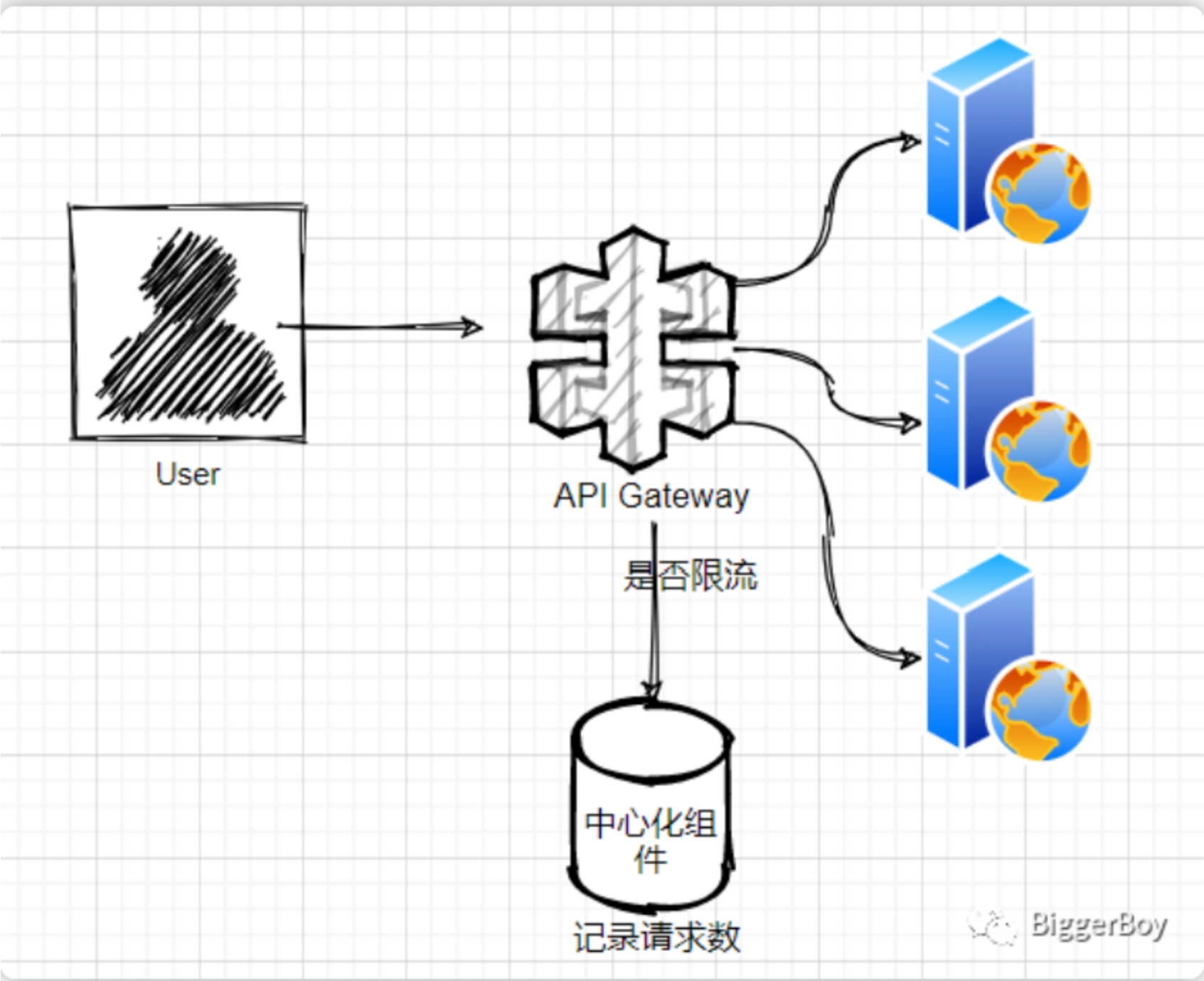


对于限流有很多方式，最经典的几种就是，计数器法、滑动窗口、漏桶法、令牌桶等，今天北哥要讲的是采用Redis + Lua脚本实现高性能的分布式限流，下面就跟着北哥来一起实战吧。

2 分布式限流

所谓的分布式限流，其实道理很简单。分布式区别于单机限流的场景，它把整个分布式集群环境中所有服务器当做一个整体来考量。比如说针对IP限流，我们限制了1个IP每秒最多10个访问，不管来自这个IP地址的请求落在了哪台机器上，只要是访问了集群中的服务节点，那么都会受到限制规则的制约。

从上面的例子不难看出，我们必须将限流信息保存在一个“中心化”的组件上，这样它就可以获取到集群中所有机器的访问状态。



目前有两个比较主流的限流方案：

1. 网关层限流。将限流规则应用在所有流量的入口处
2. 中间件限流。将限流信息存储在分布式环境中某个中间件里（比如redis），每个组件都可以从这里获取到当前时间的流量统计，从而决定是否放行还是拒绝。

3 Redis+Lua实现高性能分布式限流

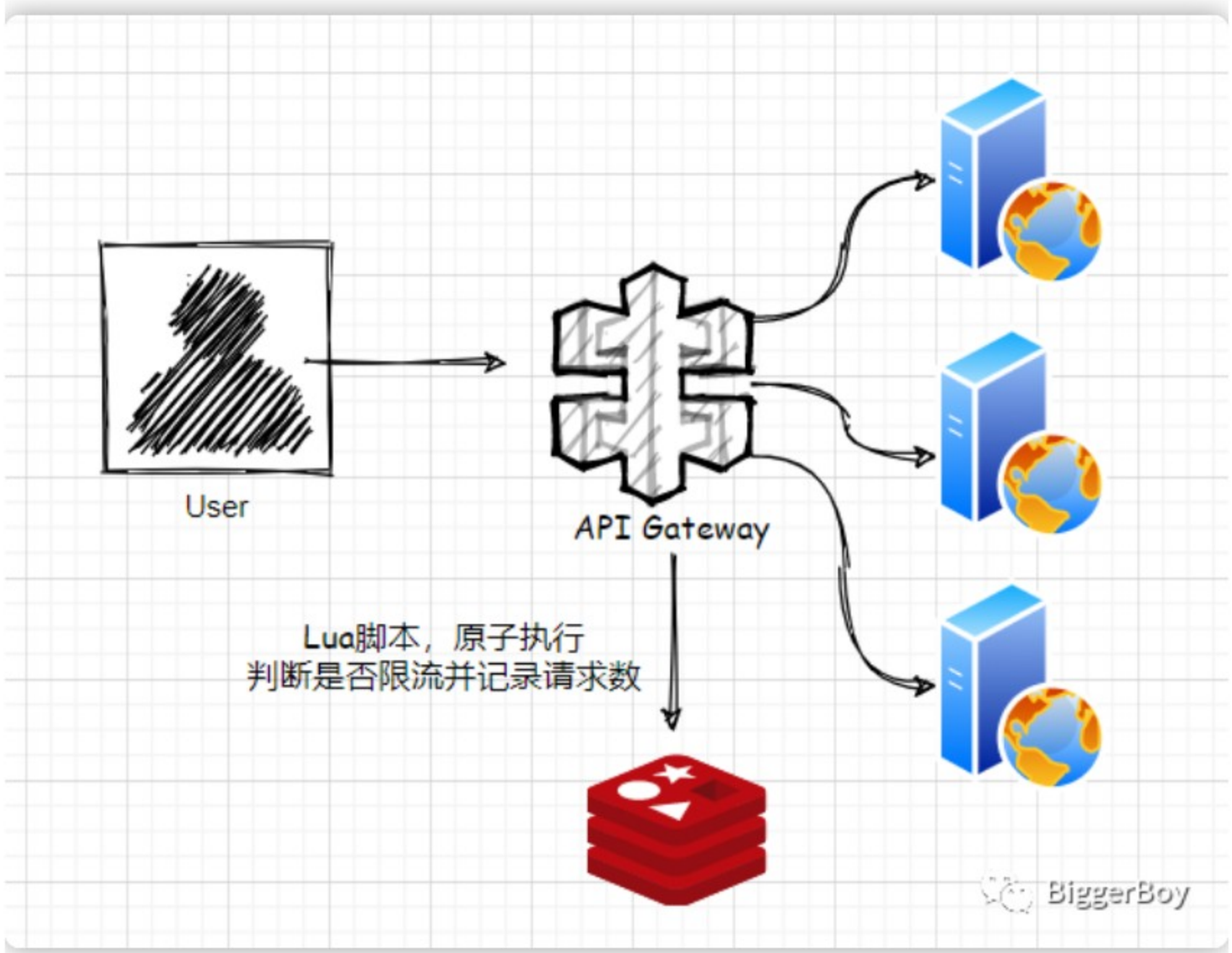
这篇文章介绍Redis+Lua实现分布式限流，很多小伙伴不知道Lua是什么，个人理解，Lua脚本和MySQL数据库的存储过程比较相似，他们执行一组命令，所有命令的执行要么全部成功或者失败，以此达到原子性。也可以把Lua脚本理解为，一段具有业务逻辑的代码块。

而Lua本身就是一种编程语言（脚本语言），Redis脚本使用Lua解释器来执行脚本。Reids 2.6 版本通过内嵌支持Lua环境。执行脚本的常用命令为 EVAL。详细参考<https://www.redis.net.cn/tutorial/3516.html>

虽然Redis官方没有直接提供限流相应的API，但却支持了Lua脚本的功能，可以使用它实现复杂的令牌桶或漏桶算法，也是分布式系统中实现限流的主要方式之一。

并且通常我们使用Redis事务时，并不是直接使用Redis自身提供的事务功能，而是使用Lua脚本。相比Redis事务，Lua脚本的优点：

- 减少网络开销：使用Lua脚本，无需向Redis发送多次请求，执行一次即可，减少网络传输
- 原子操作：Redis将整个Lua脚本作为一个命令执行，原子，无需担心并发
- 复用：Lua脚本一旦执行，会永久保存Redis中，其他客户端可复用



Lua脚本大致逻辑如下：

```
1 -- 获取调用脚本时传入的第一个key值（用作限流的 key）
2 local key = KEYS[1]
3 -- 获取调用脚本时传入的第一个参数值（限流大小）
4 local limit = tonumber(ARGV[1])
5
6 -- 获取当前流量大小
7 local curentLimit = tonumber(redis.call('get', key) or "0")
8
9 -- 是否超出限流
10 if curentLimit + 1 > limit then
11     -- 返回(拒绝)
12     return 0
13 else
14     -- 没有超出 value + 1
15     redis.call("INCRBY", key, 1)
16     -- 设置过期时间
17     redis.call("EXPIRE", key, 2)
18     -- 返回(放行)
19     return 1
20 end
```

- 通过KEYS[1] 获取传入的key参数
- 通过ARGV[1]获取传入的limit参数
- redis.call方法，从缓存中get和key相关的值，如果为null那么就返回0
- 接着判断缓存中记录的数值是否会大于限制大小，如果超出表示该被限流，返回0
- 如果未超过，那么该key的缓存值+1，并设置过期时间为1秒钟以后，并返回缓存值+1

4 实战

首先创建一个springboot项目，在pom.xml中引入依赖：


```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-data-redis</artifactId>
4 </dependency>
5 <dependency>
6   <groupId>redis.clients</groupId>
7   <artifactId>jedis</artifactId>
8 </dependency>
9 <dependency>
10  <groupId>com.google.guava</groupId>
11  <artifactId>guava</artifactId>
12  <version>21.0</version>
13 </dependency>
14 <dependency>
15  <groupId>org.springframework.boot</groupId>
16  <artifactId>spring-boot-starter-aop</artifactId>
17 </dependency>
18 <dependency>
19  <groupId>org.apache.commons</groupId>
20  <artifactId>commons-lang3</artifactId>
21  <version>3.9</version>
22 </dependency>
23 <dependency>
24  <groupId>org.projectlombok</groupId>
25  <artifactId>lombok</artifactId>
26  <optional>true</optional>
27 </dependency>
```

配置RedisTemplate

首先application.properties配置Redis连接信息

```
1 spring.redis.host=127.0.0.1
2 spring.redis.port=6379
3 spring.redis.database=0
```

然后通过@Bean配置RedisTemplate

```
1 import org.springframework.context.annotation.Bean;
2 import org.springframework.context.annotation.Configuration;
3 import org.springframework.data.redis.connection.RedisConnectionFactory;
4 import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
5 import org.springframework.data.redis.core.RedisTemplate;
6 import org.springframework.data.redis.serializer.GenericJackson2JsonRedisSerializer;
7 import org.springframework.data.redis.serializer.StringRedisSerializer;
8
9 import java.io.Serializable;
10
11 /**
12  * redis配置类.
13  *
14  * @author : 北哥 公众号: BiggerBoy
15  * @version : 1.0 2022/09/09
16  * @since : 1.0
17  */
18 @Configuration
19 public class RedisConfig {
20     @Bean
21     JedisConnectionFactory jedisConnectionFactory() {
22         return new JedisConnectionFactory();
23     }
24
25     @Bean
26     public RedisTemplate<String, Serializable> limitRedisTemplate(RedisConnectionFactory jedisConnectionFactory) {
27         RedisTemplate<String, Serializable> template = new RedisTemplate<>();
28         template.setKeySerializer(new StringRedisSerializer());
29         template.setValueSerializer(new GenericJackson2JsonRedisSerializer());
30         template.setConnectionFactory(jedisConnectionFactory);
31         return template;
32     }
33 }
```

创建自定义注解

然后我们创建自定义注解：

```
1 import com.biggerboy.redislimiter.enums.LimitType;
2 import java.lang.annotation.*;
3
4 /**
5  * redis限流注解.
6  *
7  * @author : 北哥 公众号: BiggerBoy
8  * @version : 1.0 2022/09/09
9  * @since : 1.0
10 */
11 @Target({ElementType.METHOD, ElementType.TYPE})
12 @Retention(RetentionPolicy.RUNTIME)
13 @Inherited
14 @Documented
15 public @interface MyRedisLimiter {
16     /**
17      * 缓存到Redis的key
18      */
19     String key();
20     /**
21      * Key的前缀
22      */
23     String prefix() default "limiter:";
24     /**
25      * 给定的时间范围 单位(秒)
26      * 默认1秒 即1秒内超过count次的请求将会被限流
27      */
28     int period() default 1;
29     /**
30      * 一定时间内最多访问的次数
31      */
32     int count();
33     /**
34      * 限流的维度(用户自定义key 或者 调用方ip)
35      */
36     LimitType limitType() default LimitType.CUSTOMER;
37 }
```

创建切面类RedisLimitAspect

大致逻辑是获取方法上的注解MyRedisLimiter，从注解上获取配置信息，组装keys和参数，然后调用RedisTemplate的execute方法获取当前时间内请求数，小于等于limitCount则不限流，否则限流降级处理。

```
1 /**
2  * @param pjp
3  * @author 闻北(北哥) 公众号: BiggerBoy
4  * @description 切面
5  * @date 2022-9-8 18:29:53
6  */
7 @Around("execution(public * *(..)) && @annotation(com.wenbei.annotation.MyRedisLimiter)")
8 public Object limit(ProceedingJoinPoint pjp) {
9     MethodSignature signature = (MethodSignature) pjp.getSignature();
10    Method method = signature.getMethod();
11    MyRedisLimiter limitAnnotation = method.getAnnotation(MyRedisLimiter.class);
12    LimitType limitType = limitAnnotation.limitType();
13    int limitPeriod = limitAnnotation.period();
14    int limitCount = limitAnnotation.count();
15
16    String key = getKey(limitAnnotation, limitType);
17    ImmutableList<String> keys = ImmutableList.of(StringUtils.join(limitAnnotation.prefix(), key));
18    try {
19        Number count = limitRedisTemplate.execute(redisScript, keys, limitCount, limitPeriod);
20        logger.info("try to access, this time count is {} for key: {}", count, key);
21        if (count != null && count.intValue() <= limitCount) {
22            return pjp.proceed();
23        } else {
24            demote();//降级
25            return null;
26        }
27    } catch (Throwable e) {
28        if (e instanceof RuntimeException) {
29            throw new RuntimeException(e.getLocalizedMessage());
30        }
31        throw new RuntimeException("服务器出现异常，请稍后再试");
32    }
33 }
```

到这里，最关键的Lua是如何使用的还没讲到。我们可以看到上述代码调用limitRedisTemplate.execute参数的第一个是redisScript，这便是Redis用于执行Lua脚本的重要支持。

加载Lua脚本

在切面类中，我们可以通过初始化加载Lua脚本，如下new ClassPathResource(LIMIT_LUA_PATH)

```
1 private static final String LIMIT_LUA_PATH = "limit.lua";
2 private DefaultRedisScript<Number> redisScript;
3 @PostConstruct
4 public void init() {
5     redisScript = new DefaultRedisScript<>();
6     redisScript.setResultType(Number.class);
7     ClassPathResource classPathResource = new ClassPathResource(LIMIT_LUA_PATH);
8     try {
9         classPathResource.getInputStream();//探测资源是否存在
10        redisScript.setScriptSource(new ResourceScriptSource(classPathResource.getInputStream()));
11    } catch (IOException e) {
12        logger.error("未找到文件: {}", LIMIT_LUA_PATH);
13    }
14 }
```

我们传入常量limit.lua，这是classpath下创建的脚本文件，Lua脚本如下，也很简单，就不在赘述。通常应该在limit.lua文件中放置脚本文件，这样如果需要修改脚本，仅需要修改文件重启即可。

```
1 local count
2 count = redis.call('get',KEYS[1])
3 --不超过最大值，则直接返回
4 if count and tonumber(count) > tonumber(ARGV[1]) then
5     return count;
6 end
7 --执行计算器自加
8 count = redis.call('incr',KEYS[1])
9 if tonumber(count) == 1 then
10    --从第一次调用开始限流，设置对应key的过期时间
11    redis.call('expire',KEYS[1],ARGV[2])
12 end
13 return count;
```

降级

然后在降级方法中写我们的降级逻辑，通过抛异常或往HttpServletResponse写入返回信息都可以。

```
1 /**
2  * 降级策略
3  * @author 北哥
4  * @date 2020/4/8 13:24
5  */
6 private void demote() {
7     logger.info("try to access fail, this request will be demoted");
8     //throw new RuntimeException("限流了");
9     response.setHeader("Content-Type", "text/html;charset=UTF8");
10    PrintWriter writer = null;
11    try {
12        writer = response.getWriter();
13        writer.println("访问失败，请稍后再试...");
14        writer.flush();
15    } catch (Exception e) {
16        e.printStackTrace();
17    } finally {
18        if (writer != null) {
19            writer.close();
20        }
21    }
22 }
```

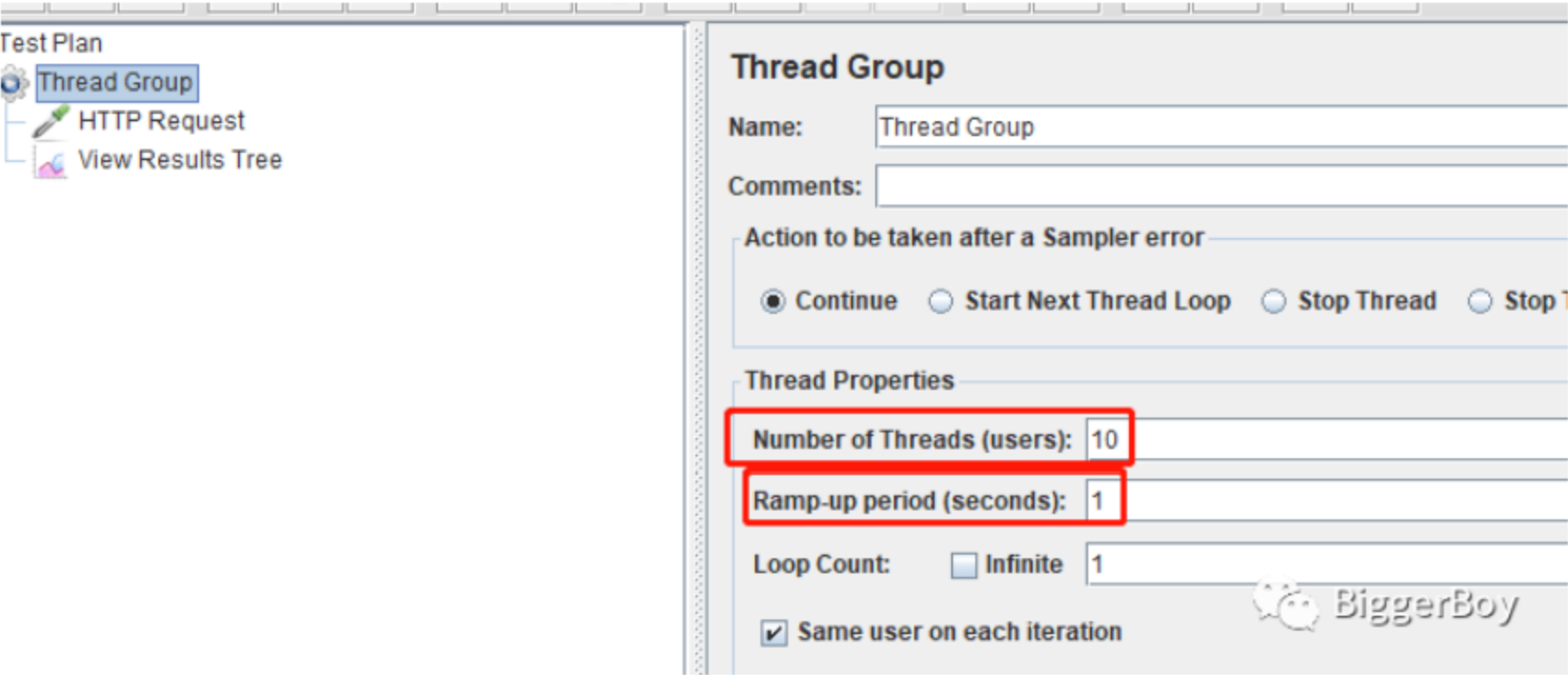
接口限流

好了，准备工作都ok了，下面我们在controller接口上加上注解，测试一下。

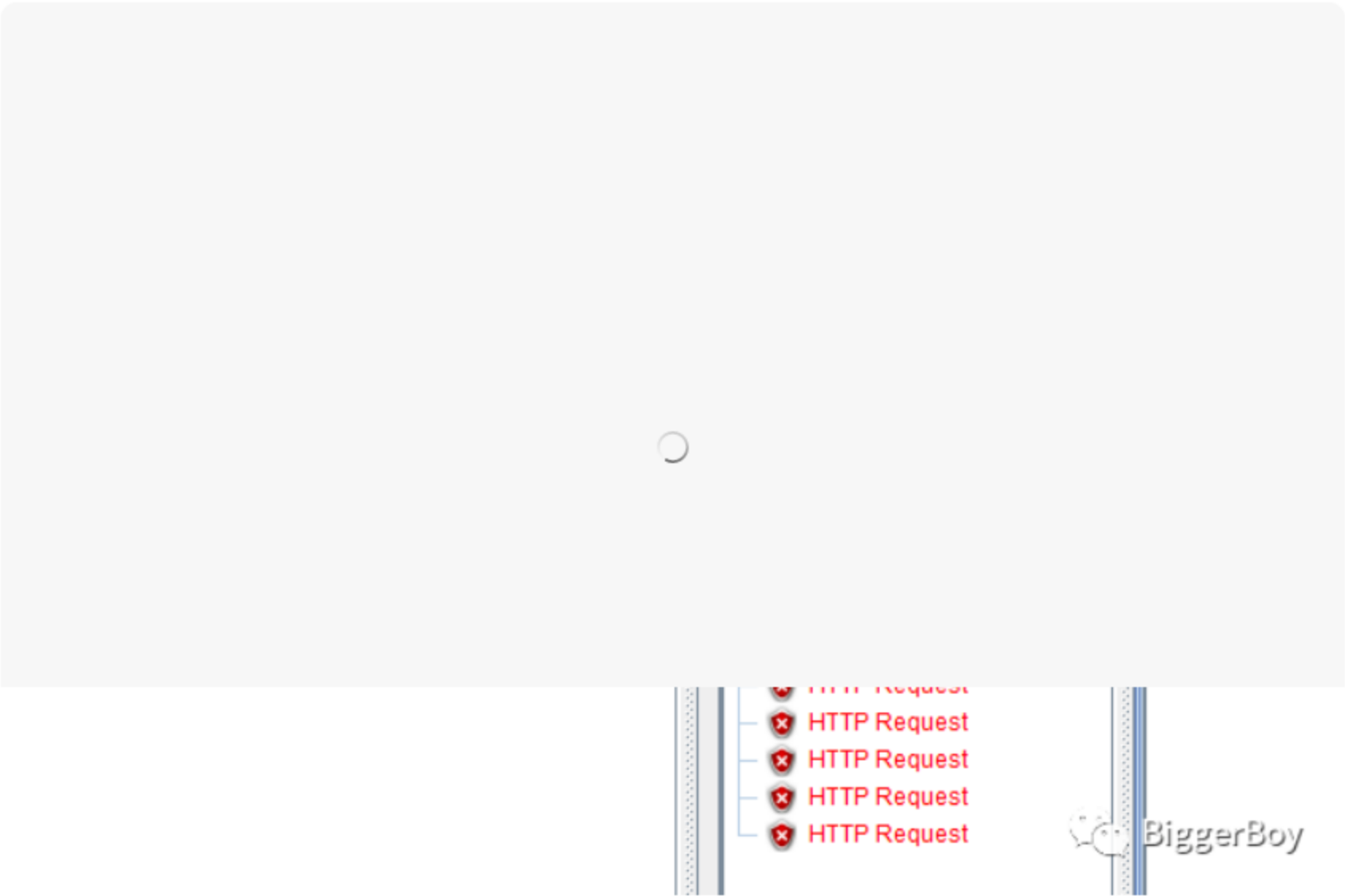
```
1  /**
2   * 测试限流controller
3   * @author 北哥 公众号: BiggerBoy
4   * @date 2022年9月8日17:02:40
5   */
6  @RestController
7  public class TestLimiterController {
8      /**
9       * @author 北哥
10      * @description
11      * @date 2020/4/8 13:42
12      */
13      @MyRedisLimiter(key = "limitTest", count = 2)
14      @RequestMapping(value = "/limitTest")
15      public Long limitTest() {
16          System.out.println("limitTest");
17          return 1L;
18      }
19
20      /**
21       * @author 北哥
22       * @description
23       * @date 2020/4/8 13:42
24       */
25      @MyRedisLimiter(key = "customer_limit_test", period = 10, count = 3, limitType = LimitType.CUSTOMER)
26      @GetMapping("/limitTest2")
27      public Integer testLimiter2() {
28          System.out.println("limitTest2");
29          return 1;
30      }
31
32      /**
33       * @author 北哥
34       * @description
35       * @date 2020/4/8 13:42
36       */
37      @MyRedisLimiter(key = "ip_limit_test", period = 10, count = 3, limitType = LimitType.IP)
38      @GetMapping("/limitTest3")
39      public Integer testLimiter3() {
40          System.out.println("limitTest3");
41          return 3;
42      }
43  }
```

测试

接口限制每秒2个请求，我们使用jmeter1秒发10个请求



结果只有前两个成功了（上述降级采用的直接抛异常，方便在这里看到限流时下面时红色的）



5 总结

以上 springboot + aop + Lua 限流实现是比较简单的，旨在让大家认识下什么是限流？如何做一个简单的限流功能，面试要知道这是个什么东西。上面虽然说了几种实现限流的方案，但选哪种还要结合具体的业务场景，不能为了用而用。在真正的场景里，不止设置一种限流规则，而是会设置多个限流规则共同作用，如连接数、访问频率、黑白名单、传输速率等。

源码地址：
<https://gitee.com/it-wenbei/redis-limiter.git>

参考：
<https://www.cnblogs.com/h1763656169/articles/16554906.html>
<https://www.redis.net.cn/tutorial/3516.html>
<https://mp.weixin.qq.com/s/kyFAWH3mVNjvurQDt4vchA>

往期文章推荐


- 大坑！隐式转换导致索引失效...
- MySQL索引知识点&常见问题汇总
- etcd的应用场景
- etcd和Zookeeper孰优孰劣？
- 联合索引在B+树上的存储结构及数据查找方式
- Redis分布式锁实战
- Mybatis第三方PageHelper插件分页原理
- MySQL索引底层原理

BiggerBoy

作者：北哥

JAVA技术
系统架构
职场干货

开发教程
面试经验
行业资讯



BiggerBoy

点个 在看 你最好看

收录于合集 #BiggerBoy 61

< 上一篇

mysql最大建议行数2000w, 靠谱吗?

下一篇 >

面试 | 如何架构高性能读服务?

喜欢此内容的人还喜欢

Python 搭配 C++ 让性能直接拉满

初代庄主



Nginx实时流量镜像

程序员实用技术分享



Redis的集群模式

IT枫斗者

