



sco5282

关注

【限流】4 种常见的限流实现方案

原创 sco5282 于 2023-07-22 16:44:37 发布 阅读量2.8k 收藏 20 点赞数 5
分类专栏: SpringBoot Java 一统天下 文章标签: spring boot java



SpringBoot 同时被 2 个专栏收录

7 订阅 36 篇文章

在微服务应用中，考虑到技术栈的组合，团队人员的开发水平，以及易维护性等因素，一个比较通用的做法是，利用 AOP 技术 + 自定义注解实现法或接口进行限流。

下面基于这个思路来分别介绍下几种常用的限流方案的实现：

- 基于 guava 限流实现（单机版）
- 基于 sentinel 限流实现（分布式版）
- 基于 redis+lua 限流实现（分布式版）
- 网关限流（分布式版）
- 自定义 starter 限流实现

1. 基于 guava 限流实现（单机版）

guava 为谷歌开源的一个比较实用的组件，利用这个组件可以帮助开发人员完成常规的限流操作，接下来看具体的实现步骤

1、引入依赖：

```
1 <dependency>
2   <groupId>com.google.guava</groupId>
3   <artifactId>guava</artifactId>
4   <version>30.1-jre</version>
5 </dependency>
```

2、自定义限流注解

自定义一个限流用的注解，后面在需要限流的方法或接口上面只需添加该注解即可

```
1 @Documented
2 @Target(value = ElementType.METHOD)
3 @Retention(value = RetentionPolicy.RUNTIME)
4 public @interface GuavaLimitRateAnnotation {
5     // 限制类型
6     String limitType();
7
8     // 每秒 5 个请求
9     double limitCount() default 5d;
10 }
```

3、限流 AOP 类

通过AOP前置通知的方式拦截添加了上述自定义限流注解的方法，解析注解中的属性值，并以该属性值作为 guava 提供的限流参数，该类为整个实现

```
1 @Aspect
2 @Component
3 public class GuavaLimitRateAspect {
4
5     private static Logger logger = LoggerFactory.getLogger(GuavaLimitRateAspect.class);
6
7     @Before("execution(@GuavaLimitRateAnnotation * *(..))")
8     public void limit(JoinPoint joinPoint) {
9         // 1. 获取当前方法
10        Method currentMethod = get_currentMethod(joinPoint);
11        if (Objects.isNull(currentMethod)) {
12            return;
13        }
14        // 2. 从方法注解定义上获取限流的类型
15        String limitType = currentMethod.getAnnotation(GuavaLimitRateAnnotation.class)
```



sco5282

关注

```

16     double limitCount = currentMethod.getAnnotation(GuavaLimitRateAnnotation.class).limitCount();
17     // 3.使用guava的令牌桶算法获取一个令牌, 获取不到先等待
18     RateLimiter rateLimiter = RateLimitHelper.getRateLimiter(limitType, limitCount);
19     boolean b = rateLimiter.tryAcquire();
20     if (b) {
21         System.out.println("获取到令牌");
22     } else {
23         HttpServletResponse resp = ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getResponse();
24         JSONObject jsonObject = new JSONObject();
25         jsonObject.put("success", false);
26         jsonObject.put("msg", "限流中");
27         try {
28             output(resp, jsonObject.toJSONString());
29         } catch (Exception e) {
30             logger.error("error,e:{", e);
31         }
32     }
33 }
34
35 public void output(HttpServletResponse response, String msg) throws IOException {
36     response.setContentType("application/json;charset=UTF-8");
37     ServletOutputStream outputStream = null;
38     try {
39         outputStream = response.getOutputStream();
40         outputStream.write(msg.getBytes("UTF-8"));
41     } catch (IOException e) {
42         e.printStackTrace();
43     } finally {
44         assert outputStream != null;
45         outputStream.flush();
46         outputStream.close();
47     }
48 }
49
50 private Method getCurrentMethod(JoinPoint joinPoint) {
51     Method[] methods = joinPoint.getTarget().getClass().getMethods();
52     Method target = null;
53     for (Method method : methods) {
54         if (method.getName().equals(joinPoint.getSignature().getName())) {
55             target = method;
56             break;
57         }
58     }
59     return target;
60 }
61
62 }

```

其中限流的核心 API 即为 `RateLimiter` 这个对象, 涉及到的 `RateLimitHelper` 类如下:

```

1 public class RateLimitHelper {
2
3     private RateLimitHelper(){}
4
5     private static Map<String, RateLimiter> rateMap = new HashMap<>();
6
7     public static RateLimiter getRateLimiter(String limitType, double limitCount) {
8         RateLimiter rateLimiter = rateMap.get(limitType);
9         if (rateLimiter == null) {
10             rateLimiter = RateLimiter.create(limitCount);
11             rateMap.put(limitType, rateLimiter);
12         }
13         return rateLimiter;
14     }
15
16 }

```

4、测试



sco5282

关注

```
1 @RestController
2 @RequestMapping("/limit")
3 public class LimitController {
4
5     @GetMapping("/limitByGuava")
6     @GuavaLimitRateAnnotation(limitType = "测试限流", limitCount = 1)
7     public String limitByGuava() {
8         return "limitByGuava";
9     }
10
11 }
```

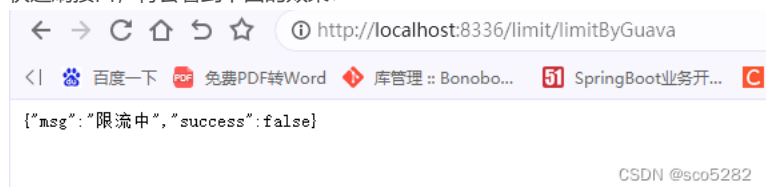
在接口中为了模拟出效果，我们将参数设置的非常小，即QPS为1，可以预想当每秒请求超过1时将会出现被限流的提示，启动工程并验证接口，每秒请求，可以正常得到结果，效果如下：



limitByGuava

CSDN @sco5282

快速刷接口，将会看到下面的效果：



CSDN @sco5282

2. 基于 sentinel 限流实现（分布式版）

sentinel 通常是需结合 springcloud-alibaba 框架一起实用的，而且与框架集成之后，可以配合控制台一起使用达到更好的效果，实际上，sentinel 提供了相对原生的 SDK 可供使用，接下来就以这种方式进行整合

1、引入依赖

```
1 <dependency>
2     <groupId>com.alibaba.csp</groupId>
3     <artifactId>sentinel-core</artifactId>
4     <version>1.8.0</version>
5 </dependency>
```

2、自定义限流注解

```
1 @Documented
2 @Target(value = ElementType.METHOD)
3 @Retention(value = RetentionPolicy.RUNTIME)
4 public @interface SentinelLimitRateAnnotation {
5
6     // 限制类型
7     String resourceName();
8
9     // 每秒 5 个
10    int limitCount() default 5;
11
12 }
```

3、自定义AOP类实现限流

该类的实现思路与上述使用guava类似，不同的是，这里使用的是sentinel原生的限流相关的API，对此不够属性的可以查阅官方的文档进行学习，这里来说了

```
1 @Aspect
2 @Component
3 public class SentinelLimitRateAspect {
4
5     @Pointcut(value = "@annotation(com.hcr.sbes.limit.sentinel.SentinelLimitRateAnnotation)")
6     public void rateLimit() {
7
8     }
```



sco5282

关注

```

/
8    }
9
10   @Around("rateLimit()")
11   public Object around(ProceedingJoinPoint joinPoint) {
12       // 1. 获取当前方法
13       Method currentMethod = getCurrentMethod(joinPoint);
14       if (Objects.isNull(currentMethod)) {
15           return null;
16       }
17       // 2. 从方法注解定义上获取限流的类型
18       String resourceName = currentMethod.getAnnotation(SentinelLimitRateAnnotation.class).resourceName();
19       if (StringUtils.isEmpty(resourceName)) {
20           throw new RuntimeException("资源名称为空");
21       }
22       int limitCount = currentMethod.getAnnotation(SentinelLimitRateAnnotation.class).limitCount();
23       // 3. 初始化规则
24       initFlowRule(resourceName, limitCount);
25       Entry entry = null;
26       Object result = null;
27       try {
28           entry = SphU.entry(resourceName);
29           try {
30               result = joinPoint.proceed();
31           } catch (Throwable throwable) {
32               throwable.printStackTrace();
33           }
34       } catch (BlockException ex) {
35           // 资源访问阻止, 被限流或被降级, 在此处进行相应的处理操作
36           System.out.println("blocked");
37           return "被限流了";
38       } catch (Exception e) {
39           Tracer.traceEntry(e, entry);
40       } finally {
41           if (entry != null) {
42               entry.exit();
43           }
44       }
45       return result;
46   }
47
48   private static void initFlowRule(String resourceName, int limitCount) {
49       List<FlowRule> rules = new ArrayList<>();
50       FlowRule rule = new FlowRule();
51       // 设置受保护的资源
52       rule.setResource(resourceName);
53       // 设置流控规则 QPS
54       rule.setGrade(RuleConstant.FLOW_GRADE_QPS);
55       // 设置受保护的资源阈值
56       rule.setCount(limitCount);
57       rules.add(rule);
58       // 加载配置好的规则
59       FlowRuleManager.loadRules(rules);
60   }
61
62   private Method getCurrentMethod(JoinPoint joinPoint) {
63       Method[] methods = joinPoint.getTarget().getClass().getMethods();
64       Method target = null;
65       for (Method method : methods) {
66           if (method.getName().equals(joinPoint.getSignature().getName())) {
67               target = method;
68               break;
69           }
70       }
71       return target;
72   }
73
74 }

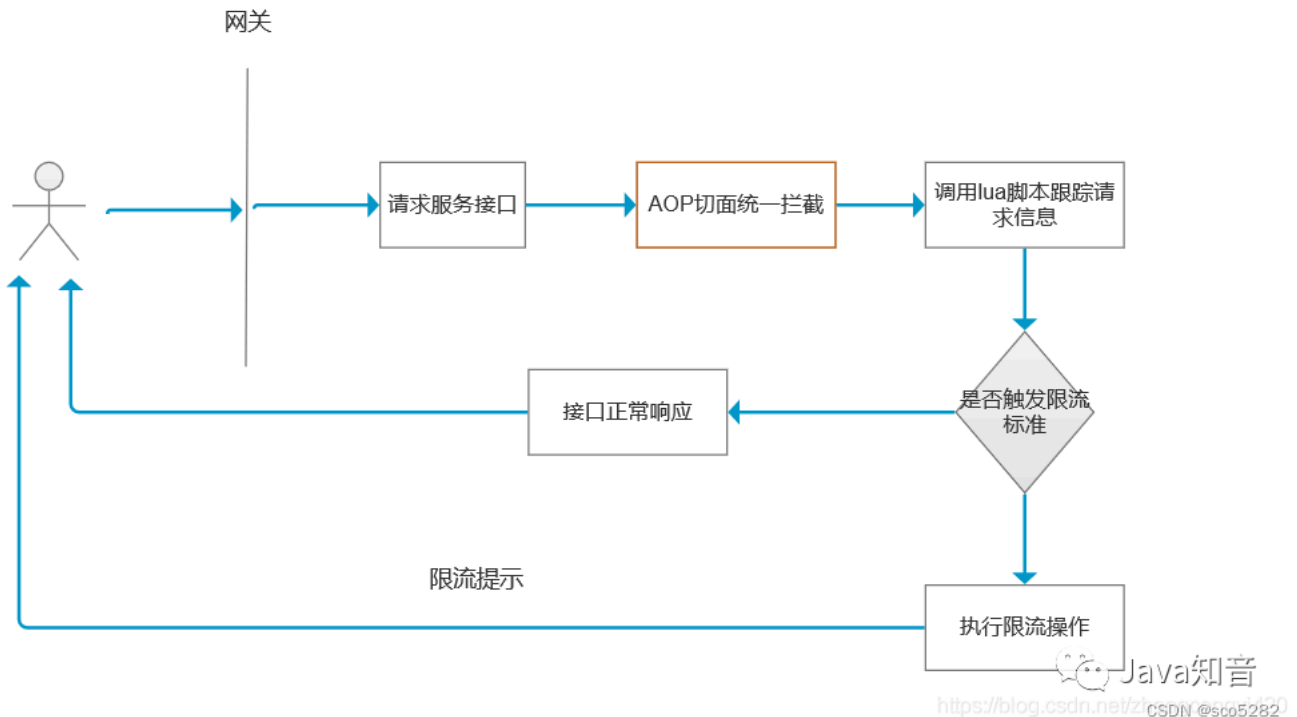
```

```
1 @GetMapping("/limitBySentinel")
2 @SentinelLimitRateAnnotation(resourceName = "测试限流2", limitCount = 1)
3 public String limitBySentinel() {
4     return "limitBySentinel";
5 }
```

3. 基于 redis+lua 限流实现（分布式版）

redis是线程安全的，天然具有线程安全的特性，支持原子性操作，限流服务不仅需要承接超高QPS，还要保证限流逻辑的执行层面具备线程安全的特性。Redis这些特性做限流，既能保证线程安全，也能保证性能。

基于redis的限流实现完整流程如下图：



1. 编写 lua 脚本，指定入参的限流规则，比如对特定的接口限流时，可以根据某个或几个参数进行判定，调用该接口的请求，在一定的时间窗口内！数；
2. 既然是限流，最好能够通用，可将限流规则应用到任何接口上，那么最合适的方式就是通过自定义注解形式切入；
3. 提供一个配置类，被 spring 的容器管理，redisTemplate 中提供了 DefaultRedisScript这个 bean；
4. 提供一个能动态解析接口参数的类，根据接口参数进行规则匹配后触发限流；

1、引入 redis 依赖

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

2、自定义注解

```
1 @Documented
2 @Target(value = ElementType.METHOD)
3 @Retention(value = RetentionPolicy.RUNTIME)
4 public @interface RedisLimitAnnotation {
5
6     /**
7      * key
8      */
9     String key() default "";
10 }
```



sco5282

关注

```

11  /**
12   * Key的前缀
13   */
14   String prefix() default "";
15
16   /**
17   * 限流时间内限流次数
18   */
19   int count();
20
21   /**
22   * 限流时间, 单位秒
23   */
24   int period();
25
26   /**
27   * 限流的类型(接口、请求ip、用户自定义key)
28   */
29   LimitTypeEnum limitType() default LimitTypeEnum.INTERFACE;
30
31 }

```

LimitTypeEnum：枚举类，定义限流类型

```

1  @Getter
2  public enum LimitTypeEnum {
3
4      // 默认限流策略, 针对某一个接口进行限流
5      INTERFACE
6      ,
7
8      // 根据IP地址进行限流
9      IP
10     ,
11
12     // 自定义的Key
13     CUSTOMER
14     ;
15
16 }

```

3、自定义 redis 配置类：解决 redis 序列化与读取 lua 脚本

```

1  @Configuration
2  public class RedisConfiguration {
3
4      @Bean
5      public DefaultRedisScript<Number> redisLuaScript() {
6          DefaultRedisScript<Number> redisScript = new DefaultRedisScript<>();
7          redisScript.setScriptSource(new ResourceScriptSource(new ClassPathResource("lua\\limit.lua")));
8          // 设置Lua脚本返回值类型 需要同Lua脚本中返回值一致
9          redisScript.setResultType(Number.class);
10         return redisScript;
11     }

```

4、自定义限流 AOP 类：进行限流操作

```

1  @Aspect
2  @Component
3  public class RedisLimitAspect {
4
5      private static final Logger logger = LoggerFactory.getLogger(RedisLimitAspect.class);
6
7      @Autowired
8      private RedisTemplate<String, Object> redisTemplate;
9
10 }

```



sco5282

关注

```

9
10
11 @Autowired

```

IpUtil : 获取 ip

```

1 public class IpUtil {
2
3     public static String getIpAddr(HttpServletRequest request) {
4         String ip = request.getHeader("x-forwarded-for");
5         if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
6             ip = request.getHeader("Proxy-Client-IP");
7         }
8         if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
9             ip = request.getHeader("WL-Proxy-Client-IP");
10        }
11        if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {

```

该类要做的事情和上面的两种限流措施类似，不过在这里核心的限流是通过读取lua脚本，通过参数传递给lua脚本实现的

5、自定义 lua 脚本：保证redis中操作原子性

在工程的 `resources/lua` 目录下，添加如下的 lua 脚本

```

1  -- 定义变量: redis中key值、规定的时间段内访问次数、redis中过期时间、当前访问次数
2
3  local key = KEYS[1]
4  local limit = tonumber(ARGV[1])
5  local count = tonumber(ARGV[2])
6  local current = tonumber(redis.call('get', key) or "0")
7
8  if current + 1 > limit then
9      return 0
10 end
11 -- 没有超阈值，将当前访问数量+1,
12 current = redis.call("INCRBY", key, "1")
13 if tonumber(current) == 1 then
14     -- 设置过期时间
15     redis.call("expire", key, count)
16 end
17 return tonumber(current)

```

6、测试接口

```

1 @RestController
2 @RequestMapping("/limit")
3 public class LimitController {
4
5     @GetMapping("/limitByRedis")
6     @RedisLimitAnnotation(key = "limitByRedis", period = 1, count = 1, limitType = LimitTypeEnum.IP)
7     public String limitByRedis() {
8         return "limitByRedis";
9     }
10
11 }

```

JAVA自定义注解实现接口/ip限流

4. 基于网关 gateway 限流（分布式版）

gateway：使用的是 Redis 加 lua 脚本的方式实现的令牌桶



sco5282

关注

暂不作介绍

5. 自定义starter限流实现

上面通过案例介绍了几种常用的限流实现，可以看到这些限流的实现都是在具体的工程模块中嵌入的。

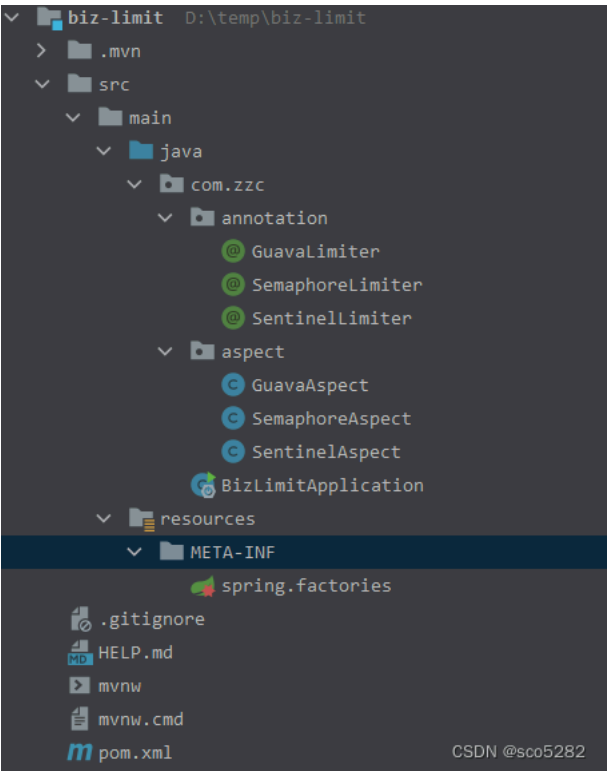
事实上，在真实的微服务开发中，一个项目可能包含了众多的微服务模块，为了减少重复造轮子，避免每个微服务模块中单独实现，可以考虑将限流封装成一个 SDK，即作为一个 springboot 的 starter 的方式被其他微服务模块进行引用即可。

这也是目前很多生产实践中比较通用的做法，接下来看看具体的实现吧。

5.1 准备

创建一个空的springboot工程，工程目录结构如下图，目录说明：

- annotation：存放自定义的限流相关的注解；
- aspect：存放不同的限流实现，比如基于guava的aop，基于sentinel的aop实现等；
- spring.factories：自定义待装配的aop实现类



5.2 代码实战

5.2.1 导入基础的依赖

这里包括如下几个必须的依赖：

- spring-boot-starter;
- guava;
- spring-boot-autoconfigure;
- sentinel-core;

```
<dependencies>
  <dependency>
1    <groupId>org.springframework.boot</groupId>
2    <artifactId>spring-boot-starter</artifactId>
3  </dependency>
4  <dependency>
5    <groupId>org.springframework.boot</groupId>
6    <artifactId>spring-boot-starter-web</artifactId>
7  </dependency>
8
```



sco5282

关注

```

9
10
11     <dependency>
12         <groupId>org.aspectj</groupId>
13         <artifactId>aspectjweaver</artifactId>
14     </dependency>
15     <dependency>
16         <groupId>com.google.guava</groupId>
17         <artifactId>guava</artifactId>
18         <version>31.1-jre</version>
19     </dependency>
20     <dependency>
21         <groupId>com.alibaba.csp</groupId>
22         <artifactId>sentinel-core</artifactId>
23         <version>1.8.0</version>
24     </dependency>
25     <dependency>
26         <groupId>org.projectlombok</groupId>
27         <artifactId>lombok</artifactId>
28         <optional>true</optional>
29     </dependency>
30     <dependency>
31         <groupId>com.alibaba</groupId>
32         <artifactId>fastjson</artifactId>
33         <version>1.2.78</version>
34     </dependency>
35 </dependencies>
36
37 <!--springboot打包-->
38 <!--<build>
39     <plugins>
40         <plugin>
41             <groupId>org.springframework.boot</groupId>
42             <artifactId>spring-boot-maven-plugin</artifactId>
43         </plugin>
44     </plugins>
45 </build-->
46
47 <!--maven打包-->
48 <build>
49     <plugins>
50         <plugin>
51             <groupId>org.apache.maven.plugins</groupId>
52             <artifactId>maven-compiler-plugin</artifactId>
53             <version>3.8.1</version>
54             <configuration>
55                 <source>1.8</source>
56                 <target>1.8</target>
57             </configuration>
58         </plugin>
59     </plugins>
60 </build>

```

【注意】：这里需要替换成 `maven` 打包工具。

5.2.2 自定义注解

目前该SDK支持三种限流方式，即后续其他微服务工程中可以通过添加这三种注解即可实现限流，分别是基于guava的令牌桶，基于sentinel的限流，1带的Semaphore限流，三个自定义注解类如下：

令牌桶：

```

1 @Target(value = ElementType.METHOD)
2 @Retention(value = RetentionPolicy.RUNTIME)
3 public @interface GuavaLimiter {
4
5     int value() default 50;
6
7 }

```



sco5282

关注

Semaphore:

```
1 @Target(value = ElementType.METHOD)
2 @Retention(value = RetentionPolicy.RUNTIME)
3 public @interface SemaphoreLimiter {
4
5     int value() default 50;
6
7 }
```

sentinel:

```
1 @Target(value = ElementType.METHOD)
2 @Retention(value = RetentionPolicy.RUNTIME)
3 public @interface SentinelLimiter {
4
5     String resourceName();
6
7     int limitCount() default 50;
8
9 }
```

5.2.3 限流实现 AOP 类

具体的限流在 AOP 中进行实现，思路和上一章类似，即通过环绕通知的方式，先解析那些添加了限流注解的方法，然后解析里面的参数，进行限流

基于 guava 的 aop 实现:

```
@Slf4j
@Aspect
@Component
public class GuavaAspect {

    private final Map<String, RateLimiter> rateLimiters = new ConcurrentHashMap<>();

    @Pointcut("@annotation(com.zzc.annotation.GuavaLimiter)")
    public void aspect() {

    }

    @Around(value = "aspect()")
    public Object around(ProceedingJoinPoint point) throws Throwable {
        log.debug("准备限流");
        Object target = point.getTarget();
        String targetName = target.getClass().getName();
        String methodName = point.getSignature().getName();
        Object[] arguments = point.getArgs();
        Class<?> targetClass = Class.forName(targetName);
        Class<?>[] argTypes = ReflectUtils.getClasses(arguments);
        Method method = targetClass.getDeclaredMethod(methodName, argTypes);
        // 获取目标method上的限流注解@Limiter
        GuavaLimiter limiter = method.getAnnotation(GuavaLimiter.class);
        RateLimiter rateLimiter = null;
        Object result = null;
        if (Objects.isNull(limiter)) {
            return point.proceed();
        }
        // 以 class + method + parameters为key, 避免重载、重写带来的混乱
        String key = targetName + "." + methodName + Arrays.toString(argTypes);
        rateLimiter = rateLimiters.get(key);
        if (null == rateLimiter) {
            // 获取限定的流量
            // 为了防止并发
            rateLimiters.putIfAbsent(key, RateLimiter.create(limiter.value()));
            rateLimiter = rateLimiters.get(key);
        }
        boolean b = rateLimiter.tryAcquire();
        if(b) {
            log.debug("得到令牌, 准备执行业务");
            return point.proceed();
        }
    }
}
```



sco5282

关注

```
40
41
42     } else {
43         HttpServletResponse resp = ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getResponse();
44         JSONObject jsonObject = new JSONObject();
45         jsonObject.put("success", false);
46         jsonObject.put("msg", "限流中");
47         try {
48             output(resp, jsonObject.toJSONString());
49         } catch (Exception e) {
50             e.printStackTrace();
51         }
52     }
53     log.debug("退出限流");
54     return result;
55 }
56
57 public void output(HttpServletResponse response, String msg) throws IOException {
58     response.setContentType("application/json;charset=UTF-8");
59     ServletOutputStream outputStream = null;
60     try {
61         outputStream = response.getOutputStream();
62         outputStream.write(msg.getBytes("UTF-8"));
63     } catch (IOException e) {
64         e.printStackTrace();
65     } finally {
66         outputStream.flush();
67         outputStream.close();
68     }
69 }
70 }
```

基于 Semaphore 的 aop 实现:

```
1  @Slf4j
2  @Aspect
3  @Component
4  public class SemaphoreAspect {
5
6      private final Map<String, Semaphore> semaphores = new ConcurrentHashMap<>();
7
8      @Pointcut("@annotation(com.zzc.annotation.SemaphoreLimiter)")
9      public void aspect() {
10
11      }
12
13      @Around(value = "aspect()")
14      public Object around(ProceedingJoinPoint point) throws Throwable {
15          log.debug("进入限流aop");
16          Object target = point.getTarget();
17          String targetName = target.getClass().getName();
18          String methodName = point.getSignature().getName();
19          Object[] arguments = point.getArgs();
20          Class<?> targetClass = Class.forName(targetName);
21          Class<?>[] argTypes = ReflectUtils.getClasses(arguments);
22          Method method = targetClass.getDeclaredMethod(methodName, argTypes);
23          // 获取目标method上的限流注解@Limiter
24          SemaphoreLimiter limiter = method.getAnnotation(SemaphoreLimiter.class);
25          Object result = null;
26          if (Objects.isNull(limiter)) {
27              return point.proceed();
28          }
29          // 以 class + method + parameters为key, 避免重载、重写带来的混乱
30          String key = targetName + "." + methodName + Arrays.toString(argTypes);
31          // 获取限定的流量
32          Semaphore semaphore = semaphores.get(key);
33          if (null == semaphore) {
34              semaphores.putIfAbsent(key, new Semaphore(limiter.value()));
35              semaphore = semaphores.get(key);
36          }
37      }
```



sco5282

关注

```

38         try {
39             semaphore.acquire();
40             result = point.proceed();
41         } finally {
42             if (null != semaphore) {
43                 semaphore.release();
44             }
45         }
46         log.debug("退出限流");
47         return result;
48     }
49 }

```

基于 sentinel 的 aop 实现:

```

@Aspect
@Component
1 public class SentinelAspect {
2
3     @Pointcut(value = "@annotation(com.zzc.annotation.SentinelLimiter)")
4     public void rateLimit() {
5
6     }
7
8     @Around("rateLimit()")
9     public Object around(ProceedingJoinPoint joinPoint) {
10         //1、获取当前的调用方法
11         Method currentMethod = getCurrentMethod(joinPoint);
12         if (Objects.isNull(currentMethod)) {
13             return null;
14         }
15         //2、从方法注解定义上获取限流的类型
16         String resourceName = currentMethod.getAnnotation(SentinelLimiter.class).resourceName();
17         if (StringUtils.isEmpty(resourceName)) {
18             throw new RuntimeException("资源名称为空");
19         }
20         int limitCount = currentMethod.getAnnotation(SentinelLimiter.class).limitCount();
21         initFlowRule(resourceName, limitCount);
22
23         Entry entry = null;
24         Object result = null;
25         try {
26             entry = SphU.entry(resourceName);
27             try {
28                 result = joinPoint.proceed();
29             } catch (Throwable throwable) {
30                 throwable.printStackTrace();
31             }
32         } catch (BlockException ex) {
33             // 资源访问阻止, 被限流或被降级
34             // 在此处进行相应的处理操作
35             System.out.println("blocked");
36             return "被限流了";
37         } catch (Exception e) {
38             Tracer.traceEntry(e, entry);
39         } finally {
40             if (entry != null) {
41                 entry.exit();
42             }
43         }
44         return result;
45     }
46
47     private static void initFlowRule(String resourceName, int limitCount) {
48         List<FlowRule> rules = new ArrayList<>();
49         FlowRule rule = new FlowRule();
50         //设置受保护的资源
51         rule.setResource(resourceName);
52         //设置流控规则 QPS
53     }

```



sco5282

关注

```
54
55
56     rule.setGrade(RuleConstant.FLOW_GRADE_QPS);
57     //设置受保护的资源阈值
58     rule.setCount(limitCount);
59     rules.add(rule);
60     //加载配置好的规则
61     FlowRuleManager.loadRules(rules);
62 }
63
64 private Method getCurrentMethod(JoinPoint joinPoint) {
65     Method[] methods = joinPoint.getTarget().getClass().getMethods();
66     Method target = null;
67     for (Method method : methods) {
68         if (method.getName().equals(joinPoint.getSignature().getName())) {
69             target = method;
70             break;
71         }
72     }
73     return target;
74 }
75 }
```

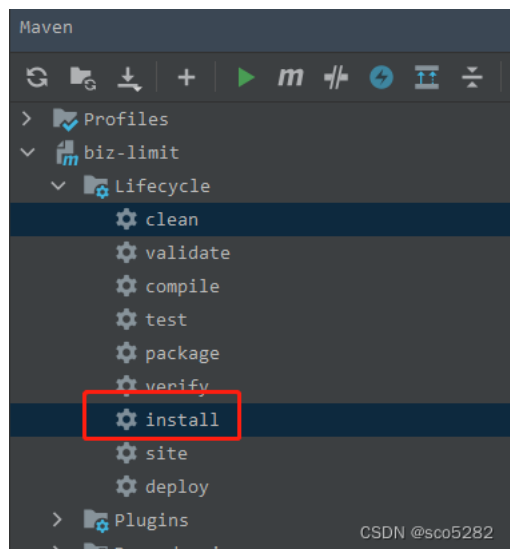
5.2.4 配置自动装配 AOP 实现

在resources目录下创建上述的spring.factories文件，内容如下，通过这种方式配置后，其他应用模块引入了当前的SDK的jar之后，就可以实现开箱即

```
1 | org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
2 |   com.zzc.aspect.GuavaAspect,\
3 |   com.zzc.aspect.SemaphoreAspect,\
4 |   com.zzc.aspect.SentinelAspect
```

5.2.5 将工程打成 jar 进行安装

mvn-install 命令



5.2.6 在其他的工程中引入上述 SDK

```
1 | <dependency>
2 |   <groupId>com.zzc</groupId>
3 |   <artifactId>biz-limit</artifactId>
4 |   <version>0.0.1-SNAPSHOT</version>
5 | </dependency>
```

5.2.7 编写测试接口

```
1 | @RestController
2 | @RequestMapping("/limit")
3 | public class LimitController {
4 |
5 | }
```



sco5282

关注

```
5    @GuavaLimiter(1)
6    @GetMapping("/guavaLimiter")
7    public String guavaLimiter(){
8        return "guavaLimiter";
9    }
10
11 }
```

上述通过starter的方式实现了一种更优雅的限流集成方式，也是生产中比较推荐的一种方式，不过当前的案例还比较粗糙，需要使用的同学还需根据情况完善里面的逻辑，进一步的封装以期得到更好的效果。

文章知识点与官方知识档案匹配，可进一步学习相关知识

Java技能树 首页 概览 149908 人正在系统学习中

限流实现方案 lsblsb的
限流实现方案 api限流实现

Java限流实现
NULL 博文链接：<https://bijian1013.iteye.com/blog/2382409>

高并发接口限流方案_高并发接口设计
网关层面的限流、或者接口调用的限流,都可以使用令牌桶算法,像 Google 的Guava,和Redisson 的限流,都用到了令牌桶算法。我认为,限流的本质是实现系统保护,最终选择

一文详解 Java 限流接口实现 java 限流实现
{ "code": -1, "message": "触发接口限流,请重试", "data": "fail"} 复制代码 四、总结 本文介绍的实现方式属于应用级限制,应用级限流方式只是单应用内的请求限流,不能进行全局

基于Redis实现分布式应用限流的方法
本篇文章主要介绍了基于 Redis 实现分布式应用限流的方法，小编觉得挺不错的，现在分享给大家，也给大家做个参考。一起跟随小编过来看看吧

实现接口的限流各种算法 zhiyikeji的
当然在现在的分布式架构下，我们可以通过服务的快速熔断，降级，隔离，限流等方式来应对单个微服务崩溃而导致整个服务不可用的情况，如果你使用的是SpringCloud

接口限流设计_接口流量控制
接口限流 一、常见限流算法 流量控制算法也叫流控算法、限流算法,主要是为了解决在面对不确定的不稳定的流量冲击下,能够确保系统的稳定运行,如果系统对流量不进行

实现接口限流的四种算法_接口限流 c++ 实现
其中窗口是在不断地滑动的,也就是说在这可变的一分钟内只有5次请求可以被处理,实现了真正的接口限流。 代码: @Controllerpublic class WindowController{@Autowired p

【Spring Cloud】Gateway 服务网关限流 最新发布 阿Q的
网关是所有请求的公共入口，所以可以在网关进行限流，而且限流的方式也很多，我们本次采用前面学过的 Sentinel 组件来实现网关的限流。

限流是如何实现的 qq_54353206的
限流是限制到达系统的并发请求数量，保证系统能够正常响应部分用户请求，而对于超过限制的流量，则通过拒绝服务的方式保证整体系统的可用性。

SpringBoot项目实战 - API接口限流
对接口限流的目的是通过对并发访问/请求进行限速,或者对一个时间窗口内的请求进行限速来保护系统,一旦达到限制速率则可以拒绝服务、排队或等待、降级等处理。 1.1

如何优雅的实现接口限流?_有什么便捷的方式能够让接口进行限流-CSDN...
首先限流,其实解决方案有很多,比如通过nginx配置,通过gateway网关进行限流,比如Spring Cloud GateWay整合熔断器实现限流 但是以上都是全局的,如何灵活的针对某些接

人人都能看懂的 6 种限流实现方案 每天叫醒自己的不是闹铃 是娟 是梦 是理
车辆限行就是一种生活中很常见的限流策略，它给我们美好的生活环境带来了一丝改善，并且快速增长的私家车已经给我们的交通 带来了巨大的“负担”，如果再不限行，

Sentinel实现限流
你可以设置不同的限流策略，例如固定窗口限流、滑动窗口限流等，这些策略基于不同的算法实现，如计数器算法、令牌桶算法和漏桶算法。这些算法可以帮助Sentinel在

限流算法及接口实现_流控算法
固定窗口限流算法(Fixed Window Rate Limiting Algorithm)是一种最简单的限流算法,其原理是在固定时间窗口(单位时间)内限制请求的数量。 3.1.2、原理 固定窗口是最简

Java并发系列之 第一篇:接口限流算法:漏桶算法&令牌桶算法
漏桶算法是一种简单但有效的接口限流算法。它的原理类似于一个漏水的桶,请求进来后按固定速率处理。当请求进来时,先看桶里有没有剩余的水,如果有,则处理请求并让

nginx限流方案的实现(三种方式)
本文将详细介绍Nginx实现限流的三种方式: `limit_conn_zone`、`limit_req_zone`以及`ngx_http_upstream_module`。 ### 1. `limit_conn_zone` `limit_conn_zone` 是Ngin

人人都能看懂的 6 种限流实现方案(纯干货)
限流的实现方案多种多样，本文将介绍六种常见的限流实现方案，并通过示例代码详细说明。 一、合法性验证限

限流功能的实现

 sco5282 关注

最常见的限流方式 固定窗口, 滑动窗口, 漏桶算法和令牌桶 假设限制10s最多请求二十次, redis_key为用户_id:api 固定窗口: 即固定死10s时间段, 在这期间只接受二十次请求

限流的几种实现 qq_40369829的
算法 流量限制 计数器 统计单位时间内请求数。超过直接拒绝。 特点: 实现简单。单位时间一开始就消耗完, 剩余时间都会拒绝, 即突刺消耗。 滑动窗口 计数器的精细

聊聊互联网限流方案 Dac
一位大神写的: 特此膜拜 http://www.dczou.com/viemall/852.html

6种常见的限流方案 开发者
限流的分类: 1) 合法性验证限流: 比如验证码、IP 黑名单等, 这些手段可以有效的防止恶意攻击和爬虫采集; 2) 容器限流: 比如 Tomcat、Nginx 等限流手段, 其中 Tc

限流实现(1)
在实际业务中, 经常会碰到突发流量的情况。如果公司基础架构做的不好, 服务无法自动扩容缩容, 在突发高流量情况下, 服务会因为压力过大而崩溃。更恐怖的是, 服务

限流实现详解 郑某某的
限流实现详解

熔断限流降级实现框架记忆如何使用java实现
熔断、限流、降级在Java中的实现, 可以使用如下框架: 1. Hystrix: 由Netflix开源的一款面向分布式系统的容错框架, 提供了线程池隔离、信号量隔离、熔断、降级、限

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00
公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心
家长监护 网络110报警服务 中国互联网举报中心 Chrome商店下载 账号管理规范 版权与免责声明 版权申诉 出版物许可证 营业执照
©1999-2024北京创新乐知网络技术有限公司



sco5282

年龄3年 暂无认证

139

4万+

3万+

36万+



原创

周排名

总排名

访问

等级

2228

550

762

64

1963

积分

粉丝

获赞

评论

收藏



















私信

关注

AI圈早知道，每日最新动态

了解全球AI新鲜事！

立即参与

大额流量券免费送

发布一篇就可获得！

去查看

搜博主文章

热门文章

- 【SpringBoot】三种方式，教你读取 jar 包中的 resources 目录下的文件 30569
- 【Excel】使用 SpringBoot 实现 Excel 文件的导入与导出 28531
- 【Thread】线程池的 7 种创建方式及自定义线程池 22443

 sco5282

关注

【SpringBoot】启动后执行方法的五种方式  15666

【HttpClient】在 SpringBoot 中使用 HttpClient 实现 HTTP 请求  13535

分类专栏

	SpringCloud Alibaba实战	7篇
	Java	39篇
	SpringBoot	36篇
	Spring	1篇
	并发编程	14篇
	消息队列	

最新评论

- 【重试】Java 中的 7 种重试机制
FBI首席执行官: 第一种, 如果不报错, 重试机制就是最大的问题, 没有异常的时候 ...
- 【File】使用 SpringBoot 实现文件的上传...
stghsiofhaseiou: 针对法发噶好贵哦阿红i回家给
- 【SpringBoot】三种方式, 教你读取 jar ...
练级中: cannot be resolved to absolute file path because it does not reside in the fi ...
- 【SpringCloud Alibaba】(四) 使用 Fei...
党同学: 讲的非常详细, 而且通俗易懂, 看得出博主的良苦用心。🌹
- 【SpringBoot】SpringBoot 中使用自定...
qq_46653265: 你这对吧, 注解应该是validated而不是, vaild (这个是java自带的 ...

最新文章

- 十一: 深入理解 Semaphore —— 信号量
- 十: 深入理解 CyclicBarrier—— 栅栏锁
- 九: 深入理解 CountDownLatch —— 闭锁/倒计时锁
- 2024年 13篇 2023年 40篇
- 2022年 23篇 2021年 64篇

目录



sco5282

关注

5.2.3 限流实现 AOP 类

5.2.4 配置自动装配 AOP 实现

5.2.5 将工程打成 jar 进行安装

5.2.6 在其他的工程中引入上述 ...



sco5282

关注