

N7-SpringBoot接口防抖

[介绍](#)

[哪一类接口需要防抖](#)

[如何确定接口是重复的](#)

[使用Redis](#)

[流程图](#)

[实现](#)

[请求锁](#)

[生成唯一 key](#)

[重复提交判断](#)

[测试](#)

介绍

所谓防抖，一是防用户手抖，二是防网络抖动。在Web系统中，表单提交是一个非常常见的功能，如果不加控制，容易因为用户的误操作或网络延迟导致同一请求被发送多次，进而生成重复的数据记录。要针对用户的误操作，前端通常会实现按钮的loading状态，阻止用户进行多次点击。而对于网络波动造成的请求重发问题，仅靠前端是不行的。为此，后端也应实施相应的防抖逻辑，确保在网络波动的情况下不会接收并处理同一请求多次。

哪一类接口需要防抖

接口防抖也不是每个接口都需要加，一般需要加防抖的接口有这几类：

- 用户输入类接口：比如搜索框输入、表单输入等，用户输入往往会频繁触发接口请求，但是每次触发并不一定需要立即发送请求，可以等待用户完成输入一段时间后再发送请求。
- 按钮点击类接口：比如提交表单、保存设置等，用户可能会频繁点击按钮，但是每次点击并不一定需要立即发送请求，可以等待用户停止点击一段时间后再发送请求。
- 滚动加载类接口：比如下拉刷新、上拉加载更多等，用户可能在滚动过程中频繁触发接口请求，但是每次触发并不一定需要立即发送请求，可以等待用户停止滚动一段时间后再发送请求。

如何确定接口是重复的

防抖也即防重复提交，那么如何确定两次接口就是重复的呢？首先，我们需要给这两次接口的调用加一个时间间隔，大于这个时间间隔的一定不是重复提交；其次，两次请求提交的参数比对，不一定要全部参数，选择标识性强的参数即可；最后，如果想做的更好一点，还可以加一个请求地址的对比。

使用Redis

流程图





@稀土掘金技术社区

实现

请求锁

@RequestLock注解定义了几个基础的属性，redis锁前缀、redis锁时间、redis锁时间单位、key分隔符。其中前面三个参数比较好理解，都是一个锁的基本信息。key分隔符是用来将多个参数合并在一起的，比如userName是张三，userPhone是123456，那么完整的key就是"张三&123456"，最后再加上redis锁前缀，就组成了一个唯一key。

```

1  import java.lang.annotation.Documented;
2  import java.lang.annotation.ElementType;
3  import java.lang.annotation.Inherited;
4  import java.lang.annotation.Retention;
5  import java.lang.annotation.RetentionPolicy;
6  import java.lang.annotation.Target;
7  import java.util.concurrent.TimeUnit;
8
9  /**
10   * @description 请求防抖锁，用于防止前端重复提交导致的错误
11   */
12  @Target(ElementType.METHOD)
13  @Retention(RetentionPolicy.RUNTIME)
14  @Documented
15  @Inherited
16  public @interface RequestLock {
17      /**
18       * redis锁前缀
19       *
20       * @return 默认为空，但不可为空
21       */
22      String prefix() default "redis";
23
24      /**
25       * redis锁过期时间
26       *
27       * @return 默认2秒
28       */
29      int expire() default 2;
30
31      /**
32       * redis锁过期时间单位
33       *
34       * @return 默认单位为秒
35       */
36      TimeUnit timeUnit() default TimeUnit.SECONDS;
37
38      /**
39       * redis key分隔符
40       *
41       * @return 分隔符
42       */
43      String delimiter() default "&";
44  }

```

生成唯一 key

Redis的效率跟key的大小息息相关，直接把参数拿来用有些时候不太合适，所以我们定一个可以选择参数作为 key 的注解

```
1  import java.lang.annotation.*;
2
3  /**
4   * @description 加上这个注解可以将参数设置为key
5   */
6  @Target({ElementType.METHOD, ElementType.PARAMETER, ElementType.FIELD})
7  @Retention(RetentionPolicy.RUNTIME)
8  @Documented
9  @Inherited
10 public @interface RequestKeyParam {
11
12 }
```

接下来生成 LockKey

```

1  import java.lang.annotation.Annotation;
2  import java.lang.reflect.Field;
3  import java.lang.reflect.Method;
4  import java.lang.reflect.Parameter;
5
6  import org.aspectj.lang.ProceedingJoinPoint;
7  import org.aspectj.lang.reflect.MethodSignature;
8  import org.springframework.util.ReflectionUtils;
9  import org.springframework.util.StringUtils;
10
11 public class RequestKeyGenerator {
12     /**
13      * 获取LockKey
14      *
15      * @param joinPoint 切入点
16      * @return
17      */
18     public static String getLockKey(ProceedingJoinPoint joinPoint) {
19         //获取连接点的方法签名对象
20         MethodSignature methodSignature = (MethodSignature)joinPoint.getSignature();
21         //Method对象
22         Method method = methodSignature.getMethod();
23         //获取Method对象上的注解对象
24         RequestLock requestLock = method.getAnnotation(RequestLock.class);
25         //获取方法参数
26         final Object[] args = joinPoint.getArgs();
27         //获取Method对象上所有的注解
28         final Parameter[] parameters = method.getParameters();
29         StringBuilder sb = new StringBuilder();
30         for (int i = 0; i < parameters.length; i++) {
31             final RequestKeyParam keyParam = parameters[i].getAnnotation(RequestKeyParam.class);
32             //如果属性不是RequestKeyParam注解，则不处理
33             if (keyParam == null) {
34                 continue;
35             }
36             //如果属性是RequestKeyParam注解，则拼接 连接符 "& + RequestKeyParam"
37             sb.append(requestLock.delimiter()).append(args[i]);
38         }
39         //如果方法上没有加RequestKeyParam注解
40         if (StringUtils.isEmpty(sb.toString())) {
41             //获取方法上的多个注解（为什么是两层数组：因为第二层数组是只有一个元素的数组）
42             final Annotation[][] parameterAnnotations = method.getParameterAnnotations();

```

```

43         //循环注解
44         for (int i = 0; i < parameterAnnotations.length; i++) {
45             final Object object = args[i];
46             //获取注解类中所有的属性字段
47             final Field[] fields = object.getClass().getDeclaredFields
48         ();
49             for (Field field : fields) {
50                 //判断字段上是否有RequestKeyParam注解
51                 final RequestKeyParam annotation = field.getAnnotation
52         (RequestKeyParam.class);
53                 //如果没有，跳过
54                 if (annotation == null) {
55                     continue;
56                 }
57                 //如果有，设置Accessible为true（为true时可以使用反射访问私有
58         变量，否则不能访问私有变量）
59                 field.setAccessible(true);
60                 //如果属性是RequestKeyParam注解，则拼接 连接符" & + RequestKeyParam"
61                 sb.append(requestLock.delimiter()).append(ReflectionUtils.getField(field, object));
62             }
63         }
64         //返回指定前缀的key
65         return requestLock.prefix() + sb;
66     }

```

> 由于`@RequestKeyParam`可以放在方法的参数上，也可以放在对象的属性上，所以这里需要进行两次判断，一次是获取方法上的注解，一次是获取对象里面属性上的注解。

重复提交判断

这里的核心代码是stringRedisTemplate.execute里面的内容，正如注释里面说的“使用RedisCallback接口执行set命令，设置锁键；设置额外选项：过期时间和SET_IF_ABSENT选项” SET_IF_ABSENT 是 RedisStringCommands.SetOption 枚举类中的一个选项，用于在执行 SET 命令时设置键值对的时候，如果键不存在则进行设置，如果键已经存在，则不进行设置。


```

1  /**
2   * @description 缓存实现
3   */
4   @Aspect
5   @Configuration
6   @Order(2)
7   public class RedisRequestLockAspect {
8
9       private final StringRedisTemplate stringRedisTemplate;
10
11       @Autowired
12       public RedisRequestLockAspect(StringRedisTemplate stringRedisTemplate
13       ) {
14           this.stringRedisTemplate = stringRedisTemplate;
15       }
16
17       @Around("execution(public * * (..)) && @annotation(org.example.test2.1
18       ock.RequestLock)")
19       public Result<Object> interceptor(ProceedingJoinPoint joinPoint) {
20           MethodSignature methodSignature = (MethodSignature)joinPoint.getSi
21           gnature();
22           Method method = methodSignature.getMethod();
23           RequestLock requestLock = method.getAnnotation(RequestLock.class);
24
25           //获取自定义key
26           final String lockKey = RequestKeyGenerator.getLockKey(joinPoint);
27           // 使用RedisCallback接口执行set命令，设置锁键；设置额外选项：过期时间和SET
28           _IF_ABSENT选项
29           final Boolean success = stringRedisTemplate.execute(
30               (RedisCallback<Boolean>) connection -> connection.set(lock
31               Key.getBytes(), new byte[0],
32               Expiration.from(requestLock.expire(), requestLock.
33               timeUnit()),
34               RedisStringCommands.SetOption.SET_IF_ABSENT));
35
36           if (!success) {
37               return Result.error(409, "您的操作太快了,请稍后重试");
38           }
39
40           try {
41               return (Result<Object>) joinPoint.proceed();
42           } catch (Throwable throwable) {
43               return Result.error(500, "系统异常");
44           }
45       }
46   }

```

测试

PrettyRawPreviewVisualizeJSON

12345

"code": 409,

"msg": "您的操作太快了,请稍后重试",

"data": null