芋道源码 2022-04-09

幂等性（防重复提交）

`yudao-spring-boot-starter-protection` 技术组件，由它的 `idempotent` 包，提供声明式的幂等特性，可防止重复请求。例如说，用户快速的双击了某个按钮，前端没有禁用该按钮，导致发送了两次重复的请求。

```
// UserController.java

@Idempotent(timeout = 10, timeUnit = TimeUnit.SECONDS, message = "正在添加用户中,
@PostMapping("/user/create")
public String createUser(User user){
    userService.createUser(user);
    return "添加成功";
}
```

1. 实现原理

它的实现原理非常简单，针对相同参数的方法，一段时间内，有且仅能执行一次。执行流程如下：

① 在方法执行前，根据参数对应的 Key 查询是否存在。

- 如果**存在**，说明正在执行中，则进行报错。
- 如果**不在**，则计算参数对应的 Key，存储到 Redis 中，并设置过期时间，即标记正在执行中。

默认参数的 Redis Key 的计算规则由 DefaultIdempotentKeyResolver 实现，使用 MD5(方法名 + 方法参数)，避免 Redis Key 过长。

② 方法执行完成，**不会**主动删除参数对应的 Key。

如果希望会**主动**删除 Key，可以使用 《开发指南 —— 分布式锁》 提供的 `@Lock` 来实现幂等性。

🙂 从本质上来说， `idempotent` 包提供的幂等特性，本质上也是基于 Redis 实现的分布式锁。

③ 如果方法执行时间较长，超过 Key 的过期时间，则 Redis 会自动删除对应的 Key。因此，需要大概评估下，避免方法的执行时间超过过期时间。

2. `@Idempotent` 注解

`@Idempotent` 注解，声明在方法上，表示该方法需要开启幂等性。代码如下：

```java
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Idempotent {

    /**
     * 幂等的超时时间，默认为 1 秒
     *
     * 注意，如果执行时间超过它，请求还是会进来
     */
    int timeout() default 1;
    /**
     * 时间单位，默认为 SECONDS 秒
     */
    TimeUnit timeUnit() default TimeUnit.SECONDS;

    /**
     * 提示信息，正在执行中的提示
     */
    String message() default "重复请求，请稍后重试";

    /**
     * 使用的 Key 解析器
     */
    Class<? extends IdempotentKeyResolver> keyResolver() default DefaultIdempotentKeyResolver.class;
    /**
     * 使用的 Key 参数
     */
    String keyArg() default "";

}
```

① 一般情况下，只需要使用这 2 个属性即可

② 方法参数的拼接规则

① 对应的 AOP 切面是 IdempotentAspect 类，核心就 10 行左右的代码，如下图所示：

```java
@Before("@annotation(idempotent)")
public void beforePointCut(JoinPoint joinPoint, Idempotent idempotent) {
    // 获得 IdempotentKeyResolver
    IdempotentKeyResolver keyResolver = keyResolvers.get(idempotent.keyResolver());
    Assert.notNull(keyResolver, message: "找不到对应的 IdempotentKeyResolver");
    // 解析 Key
    String key = keyResolver.resolver(joinPoint, idempotent);

    // 锁定 Key。
    boolean success = idempotentRedisDAO.setIfAbsent(key, idempotent.timeout(), idempotent.timeUnit());
    // 锁定失败，抛出异常
    if (!success) {
        log.info("[beforePointCut][方法({}) 参数({}) 存在重复请求]", joinPoint.getSignature().toString(), joinPoint.getArgs());
        throw new ServiceException(GlobalErrorCodeConstants.REPEATED_REQUESTS.getCode(), idempotent.message());
    }
}
```

① 基于方法参数，计算对应的 Key

② 向 Redis 写入 Key，如果存在，则写入失败，抛出 900 业务异常

② 对应的 Redis Key 的前缀是 `idempotent:%s` ，可见 IdempotentRedisDAO 类，如下图所示：

```java
@AllArgsConstructor
public class IdempotentRedisDAO {

    private static final RedisKeyDefine IDEMPOTENT = new RedisKeyDefine( memo: "幂等操作",
            keyTemplate: "idempotent:%s", // 参数为 uuid
            STRING, String.class, RedisKeyDefine.TimeoutTypeEnum.DYNAMIC);

    private final StringRedisTemplate redisTemplate;

    public Boolean setIfAbsent(String key, long timeout, TimeUnit timeUnit) {
        String redisKey = formatKey(key);
        return redisTemplate.opsForValue().setIfAbsent(redisKey, value: "", timeout, timeUnit);
    }
```

%s 对应方法名 + 方法参数计算出来的结果

3. 使用示例

本小节，我们实现 `/admin-api/infra/test-demo/get` RESTful API 接口的幂等性。

① 在 `pom.xml` 文件中，引入 `yudao-spring-boot-starter-protection` 依赖。

```
<dependency>
    <groupId>cn.iocoder.cloud</groupId>
    <artifactId>yudao-spring-boot-starter-protection</artifactId>
</dependency>
```

② 在 `/admin-api/infra/test-demo/get` RESTful API 接口的对应方法上，添加 `@Idempotent` 注解。代码如下：

```
// TestDemoController.java

@GetMapping("/get")
@Idempotent(timeout = 10, message = "重复请求，请稍后重试")
public CommonResult<TestDemoRespVO> getTestDemo(@RequestParam("id") Long id) {
    // ... 省略代码
}
```

③ 调用 `/admin-api/infra/test-demo/get` RESTful API 接口，执行成功。

④ 再次调用 `/admin-api/infra/test-demo/get` RESTful API 接口，被幂等性拦截，执行失败。

```
{
  "code": 900,
  "data": null,
  "msg": "重复请求，请稍后重试"
}
```

← 分布式锁 数据库文档→