一、功能说明

该文档的目的是是介绍一种可实施且通用的分布式锁方案, 以及工具包: tools-api的扩展用法。

背景

- 当某个业务(订单)的操作比较复杂, 比如: 客户端, 商家端, 运营端, 系统定时任务等多处可能有并发问题时
  - **常规做法**: 在每个端的接口上加上分布式锁代码, 但是这样手工编码的方式有代码冗余
  - **变量锁的方式**: 只需要在每个接口上添加注解即可达到手工一样的效果, 减少代码冗余

二、工具使用

- 添加依赖: pom.xml

```
<dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>tools-api</artifactId>
    <version>4.20.1</version>
</dependency>
```

- 启用工具

> 在启动类上或其他配置类上扫描包: cn.gmlee.tools

```
@SpringBootApplication
@ComponentScan("cn.gmlee.tools")
public class XxxApp {
    public static void main(String[] args) {
        SpringApplication.run(XxxApp.class, args);
    }
}
```

- 添加注解: @VariableLock

```
@VariableLock({"key", "val"})
@GetMapping("/lock1")
public JsonResult lock1(String key, String val) {
    return JsonResult.OK;
}
```

三、工具原理

逻辑

- 工具对添加了@VariableLock注解的业务方法进行拦截
- 在进入方法前对指定的请求参数进行加锁处理
- 在方法完成后对指定的请求参数进行解锁处理

代码

- 切面: cn.gmlee.tools.api.aop.VariableLockAspect
- 锁服务: cn.gmlee.tools.api.lock.VariableLockServer
    - Redis依赖丢失时采用内存锁(仅适用单机版): MemoryVariableLockServer
    - Redis依赖存在时采用Redis锁(适用集群版): RedisVariableLockServer

数据

- 加锁时会将参数值绑定到当前线程, 业务完成后删除。

四、使用实例

代码

```
@VariableLock({"key", "val"})
@GetMapping("/lock1")
public JsonResult lock1(String key, String val) {
    return JsonResult.OK;
}

@VariableLock({"key", "val"})
@PostMapping("/lock2")
public JsonResult lock2(String key, String val) {
    return JsonResult.OK;
}

@VariableLock({"key", "val"})
@PostMapping("/lock3")
public JsonResult lock3(Kv<String, String> kv) {
    return JsonResult.OK;
}

@VariableLock({"key", "val"})
@PostMapping("/lock4")
public JsonResult lock4(@RequestBody Kv<String, String> kv) {
    return JsonResult.OK;
}
```

日志

```
2023-05-30 17:04:19.562  INFO 24516 --- [  XNIO-1 task-1]
c.gmlee.tools.api.lock.RedisDataLockServer  : 【变量锁】加锁完成: true
tools:api:variableLock:key_123-_val_-321 123-,-321
2023-05-30 17:04:19.574  INFO 24516 --- [  XNIO-1 task-1]
c.gmlee.tools.api.lock.RedisDataLockServer  : 【变量锁】解锁完成: true
tools:api:variableLock:key_123-_val_-321 123-,-321
```