Redis

# 依赖:

<**dependency**>  
 <**groupId**>redis.clients</**groupId**>  
 <**artifactId**>jedis</**artifactId**>  
 <**version**>2.9.0</**version**>  
</**dependency**>

# 配置:

**spring.redis.host**=**47.100.138.136  
spring.redis.port**=**6379  
spring.redis.database**=**0**

# 工具类:

**public class** RedisUtil {  
 **private** JedisPool **jedisPool**;  
 **public void** initPool(String host,**int** port ,**int** database){  
 JedisPoolConfig poolConfig = **new** JedisPoolConfig();  
 poolConfig.setMaxTotal(100);  
 poolConfig.setMaxIdle(30);  
 poolConfig.setMinIdle(10);  
 poolConfig.setBlockWhenExhausted(**true**);  
 poolConfig.setMaxWaitMillis(10000);  
 poolConfig.setTestOnBorrow(**true**);  
 poolConfig.setTestOnReturn(**true**);  
 **jedisPool**=**new** JedisPool(poolConfig,host,port,20000,**"root"**);  
 }  
 **public synchronized** Jedis getJedis() {  
 Jedis jedis = **null**;  
 **if** (**jedisPool** != **null**) {  
 **try** {  
 **if** (jedis == **null**) {  
 jedis = **jedisPool**.getResource();  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 **return** jedis;  
 }  
  
 */\*\*  
 \* 回收Jedis对象资源  
 \** ***@param jedis*** *\*/* **public synchronized void** returnResource(Jedis jedis) {  
 **if** (jedis != **null**) {  
 **jedisPool**.returnResource(jedis);  
 }  
 }  
  
 */\*\*  
 \* Jedis对象出异常的时候，回收Jedis对象资源  
 \** ***@param jedis*** *\*/* **public synchronized void** returnBrokenResource(Jedis jedis) {  
 **if** (jedis != **null**) {  
 **jedisPool**.returnBrokenResource(jedis);  
 }  
  
 }  
}

# 配置类:

@Configuration  
**public class** RedisConfig {  
 *//读取配置文件中的redis的ip地址* @Value(**"${spring.redis.host:disabled}"**)  
 **private** String **host** = **"47.100.138.136"**;  
 @Value(**"${spring.redis.port:0}"**)  
 **private int port** = 6379;  
 @Value(**"${spring.redis.database:0}"**)  
 **private int database**;  
 @Bean  
 **public** RedisUtil getRedisUtil(){  
 **if**(**host**.equals(**"disabled"**)){  
 **return null**;  
 }  
 RedisUtil redisUtil=**new** RedisUtil();  
 redisUtil.initPool(**host**,**port**,**database**);  
 **return** redisUtil;  
 }  
}

# 高性能分布式锁—Redisson

## 依赖:

<**dependency**>  
 <**groupId**>org.redisson</**groupId**>  
 <**artifactId**>redisson</**artifactId**>  
 <**version**>3.10.5</**version**>  
</**dependency**>

## 配置:

@Configuration  
**public class** RedissonConfig {  
  
 @Value(**"${spring.redis.host:0}"**)  
 **private** String **host**;  
  
 @Value(**"${spring.redis.port:6379}"**)  
 **private** String **port**;  
  
 @Bean  
 **public** RedissonClient redissonClient(){  
 Config config = **new** Config();  
 config.useSingleServer().setAddress(**"redis://"**+**host**+**":"**+**port**);  
 RedissonClient redisson = Redisson.*create*(config);  
 **return** redisson;  
 }  
}

Spring Data Redis

(在Spring Data Redis中默认使用的是Lettuce作为redis框架)

Jedis：是Redis的Java实现客户端，提供了比较全面的Redis命令的支持，

　　Redisson：实现了分布式和可扩展的Java数据结构。

　　Lettuce：高级Redis客户端，用于线程安全同步，异步和响应使用，支持集群，Sentinel，管道和编码器

# 依赖:

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-redis</**artifactId**>  
</**dependency**>  
*<!--Lettuce对象池-->*<**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-pool2</**artifactId**>  
</**dependency**>

# 配置信息:

**spring.redis.host**=**47.100.138.136  
spring.redis.port**=**6379  
spring.redis.password**=**root  
spring.redis.database**=**0  
spring.redis.lettuce.pool.max-idle**=**300  
spring.redis.lettuce.pool.min-idle**=**10  
spring.redis.lettuce.pool.max-active**=**600  
spring.redis.lettuce.pool.max-wait**=**1000  
spring.redis.timeout**=**300**

# 工具类:

@Component  
**public final class** RedisUtil {  
  
 @Autowired  
 **private** RedisTemplate<String, Object> **redisTemplate**;  
  
 *// =============================common============================  
  
 /\*\*  
 \* 指定缓存失效时间  
 \*  
 \** ***@param key*** *键  
 \** ***@param time*** *时间(秒)  
 \** ***@return*** *\*/* **public boolean** expire(String key, **long** time) {  
 **try** {  
 **if** (time > 0) {  
 **redisTemplate**.expire(key, time, TimeUnit.***SECONDS***);  
 }  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 根据key 获取过期时间  
 \*  
 \** ***@param key*** *键 不能为null  
 \** ***@return*** *时间(秒) 返回0代表为永久有效  
 \*/* **public long** getExpire(String key) {  
 **return redisTemplate**.getExpire(key, TimeUnit.***SECONDS***);  
 }  
  
 */\*\*  
 \* 判断key是否存在  
 \*  
 \** ***@param key*** *键  
 \** ***@return*** *true 存在 false不存在  
 \*/* **public boolean** hasKey(String key) {  
 **try** {  
 **return redisTemplate**.hasKey(key);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 删除缓存  
 \*  
 \** ***@param key*** *可以传一个值 或多个  
 \*/* **public void** del(String... key) {  
 **if** (key != **null** && key.**length** > 0) {  
 **if** (key.**length** == 1) {  
 **redisTemplate**.delete(key[0]);  
 } **else** {  
 **redisTemplate**.delete(CollectionUtils.*arrayToList*(key));  
 }  
 }  
 }  
  
 *// ============================String=============================  
  
 /\*\*  
 \* 普通缓存获取  
 \*  
 \** ***@param key*** *键  
 \** ***@return*** *值  
 \*/* **public** Object get(String key) {  
 **return** key == **null** ? **null** : **redisTemplate**.opsForValue().get(key);  
 }  
  
 */\*\*  
 \* 普通缓存放入  
 \*  
 \** ***@param key*** *键  
 \** ***@param value*** *值  
 \** ***@return*** *true成功 false失败  
 \*/* **public boolean** set(String key, Object value) {  
 **try** {  
 **redisTemplate**.opsForValue().set(key, value);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
  
 }  
  
 */\*\*  
 \* 普通缓存放入并设置时间  
 \*  
 \** ***@param key*** *键  
 \** ***@param value*** *值  
 \** ***@param time*** *时间(秒) time要大于0 如果time小于等于0 将设置无限期  
 \** ***@return*** *true成功 false 失败  
 \*/* **public boolean** set(String key, Object value, **long** time) {  
 **try** {  
 **if** (time > 0) {  
 **redisTemplate**.opsForValue().set(key, value, time, TimeUnit.***SECONDS***);  
 } **else** {  
 set(key, value);  
 }  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 递增  
 \*  
 \** ***@param key*** *键  
 \** ***@param delta*** *要增加几(大于0)  
 \** ***@return*** *\*/* **public long** incr(String key, **long** delta) {  
 **if** (delta < 0) {  
 **throw new** RuntimeException(**"递增因子必须大于0"**);  
 }  
 **return redisTemplate**.opsForValue().increment(key, delta);  
 }  
  
 */\*\*  
 \* 递减  
 \*  
 \** ***@param key*** *键  
 \** ***@param delta*** *要减少几(小于0)  
 \** ***@return*** *\*/* **public long** decr(String key, **long** delta) {  
 **if** (delta < 0) {  
 **throw new** RuntimeException(**"递减因子必须大于0"**);  
 }  
 **return redisTemplate**.opsForValue().increment(key, -delta);  
 }  
  
 *// ================================Map=================================  
  
 /\*\*  
 \* HashGet  
 \*  
 \** ***@param key*** *键 不能为null  
 \** ***@param item*** *项 不能为null  
 \** ***@return*** *值  
 \*/* **public** Object hget(String key, String item) {  
 **return redisTemplate**.opsForHash().get(key, item);  
 }  
  
 */\*\*  
 \* 获取hashKey对应的所有键值  
 \*  
 \** ***@param key*** *键  
 \** ***@return*** *对应的多个键值  
 \*/* **public** Map<Object, Object> hmget(String key) {  
 **return redisTemplate**.opsForHash().entries(key);  
 }  
  
 */\*\*  
 \* 获取hashKey对应的所有值  
 \*  
 \** ***@param key*** *键  
 \** ***@return*** *对应的多个值  
 \*/* **public** List<Object> hmgetValues(String key) {  
 **return redisTemplate**.opsForHash().values(key);  
 }  
  
 */\*\*  
 \* 获取hashKey对应的所有键  
 \*  
 \** ***@param key*** *键  
 \** ***@return*** *对应的多个键  
 \*/* **public** Set<Object> hmgetKeys(String key) {  
 **return redisTemplate**.opsForHash().keys(key);  
 }  
  
 */\*\*  
 \* HashSet  
 \*  
 \** ***@param key*** *键  
 \** ***@param map*** *对应多个键值  
 \** ***@return*** *true 成功 false 失败  
 \*/* **public boolean** hmset(String key, Map<String, Object> map) {  
 **try** {  
 **redisTemplate**.opsForHash().putAll(key, map);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* HashSet 并设置时间  
 \*  
 \** ***@param key*** *键  
 \** ***@param map*** *对应多个键值  
 \** ***@param time*** *时间(秒)  
 \** ***@return*** *true成功 false失败  
 \*/* **public boolean** hmset(String key, Map<String, Object> map, **long** time) {  
 **try** {  
 **redisTemplate**.opsForHash().putAll(key, map);  
 **if** (time > 0) {  
 expire(key, time);  
 }  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 向一张hash表中放入数据,如果不存在将创建  
 \*  
 \** ***@param key*** *键  
 \** ***@param item*** *项  
 \** ***@param value*** *值  
 \** ***@return*** *true 成功 false失败  
 \*/* **public boolean** hset(String key, String item, Object value) {  
 **try** {  
 **redisTemplate**.opsForHash().put(key, item, value);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 向一张hash表中放入数据,如果不存在将创建  
 \*  
 \** ***@param key*** *键  
 \** ***@param item*** *项  
 \** ***@param value*** *值  
 \** ***@param time*** *时间(秒) 注意:如果已存在的hash表有时间,这里将会替换原有的时间  
 \** ***@return*** *true 成功 false失败  
 \*/* **public boolean** hset(String key, String item, Object value, **long** time) {  
 **try** {  
 **redisTemplate**.opsForHash().put(key, item, value);  
 **if** (time > 0) {  
 expire(key, time);  
 }  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 删除hash表中的值  
 \*  
 \** ***@param key*** *键 不能为null  
 \** ***@param item*** *项 可以使多个 不能为null  
 \*/* **public void** hdel(String key, Object... item) {  
 **redisTemplate**.opsForHash().delete(key, item);  
 }  
  
 */\*\*  
 \* 判断hash表中是否有该项的值  
 \*  
 \** ***@param key*** *键 不能为null  
 \** ***@param item*** *项 不能为null  
 \** ***@return*** *true 存在 false不存在  
 \*/* **public boolean** hHasKey(String key, String item) {  
 **return redisTemplate**.opsForHash().hasKey(key, item);  
 }  
  
 */\*\*  
 \* hash递增 如果不存在,就会创建一个 并把新增后的值返回  
 \*  
 \** ***@param key*** *键  
 \** ***@param item*** *项  
 \** ***@param by*** *要增加几(大于0)  
 \** ***@return*** *\*/* **public double** hincr(String key, String item, **double** by) {  
 **return redisTemplate**.opsForHash().increment(key, item, by);  
 }  
  
 */\*\*  
 \* hash递减  
 \*  
 \** ***@param key*** *键  
 \** ***@param item*** *项  
 \** ***@param by*** *要减少记(小于0)  
 \** ***@return*** *\*/* **public double** hdecr(String key, String item, **double** by) {  
 **return redisTemplate**.opsForHash().increment(key, item, -by);  
 }  
 *// ============================set=============================  
  
 /\*\*  
 \* 根据key获取Set中的所有值  
 \*  
 \** ***@param key*** *键  
 \** ***@return*** *\*/* **public** Set<Object> sGet(String key) {  
 **try** {  
 **return redisTemplate**.opsForSet().members(key);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return null**;  
 }  
 }  
  
 */\*\*  
 \* 根据value从一个set中查询,是否存在  
 \*  
 \** ***@param key*** *键  
 \** ***@param value*** *值  
 \** ***@return*** *true 存在 false不存在  
 \*/* **public boolean** sHasKey(String key, Object value) {  
 **try** {  
 **return redisTemplate**.opsForSet().isMember(key, value);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 将数据放入set缓存  
 \*  
 \** ***@param key*** *键  
 \** ***@param values*** *值 可以是多个  
 \** ***@return*** *成功个数  
 \*/* **public long** sSet(String key, Object... values) {  
 **try** {  
 **return redisTemplate**.opsForSet().add(key, values);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 */\*\*  
 \* 将set数据放入缓存  
 \*  
 \** ***@param key*** *键  
 \** ***@param time*** *时间(秒)  
 \** ***@param values*** *值 可以是多个  
 \** ***@return*** *成功个数  
 \*/* **public long** sSetAndTime(String key, **long** time, Object... values) {  
 **try** {  
 Long count = **redisTemplate**.opsForSet().add(key, values);  
 **if** (time > 0) {  
 expire(key, time);  
 }  
 **return** count;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 */\*\*  
 \* 获取set缓存的长度  
 \*  
 \** ***@param key*** *键  
 \** ***@return*** *\*/* **public long** sGetSetSize(String key) {  
 **try** {  
 **return redisTemplate**.opsForSet().size(key);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 */\*\*  
 \* 移除值为value的  
 \*  
 \** ***@param key*** *键  
 \** ***@param values*** *值 可以是多个  
 \** ***@return*** *移除的个数  
 \*/* **public long** setRemove(String key, Object... values) {  
 **try** {  
 Long count = **redisTemplate**.opsForSet().remove(key, values);  
 **return** count;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 *// ===============================list=================================  
  
 /\*\*  
 \* 获取list缓存的内容  
 \*  
 \** ***@param key*** *键  
 \** ***@param start*** *开始  
 \** ***@param end*** *结束 0 到 -1代表所有值  
 \** ***@return*** *391  
 \*/* **public** List<Object> lGet(String key, **long** start, **long** end) {  
 **try** {  
 **return redisTemplate**.opsForList().range(key, start, end);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return null**;  
 }  
 }  
  
 */\*\*  
 \* 获取list缓存的长度  
 \*  
 \** ***@param key*** *键  
 \** ***@return*** *\*/* **public long** lGetListSize(String key) {  
 **try** {  
 **return redisTemplate**.opsForList().size(key);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 */\*\*  
 \* 通过索引 获取list中的值  
 \*  
 \** ***@param key*** *键  
 \** ***@param index*** *索引 index>=0时， 0 表头，1 第二个元素，依次类推；index<0时，-1，表尾，-2倒数第二个元素，依次类推  
 \** ***@return*** *\*/* **public** Object lGetIndex(String key, **long** index) {  
 **try** {  
 **return redisTemplate**.opsForList().index(key, index);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return null**;  
 }  
 }  
  
 */\*\*  
 \* 将list放入缓存  
 \*  
 \** ***@param key*** *键  
 \** ***@param value*** *值  
 \** ***@return*** *\*/* **public boolean** lSet(String key, Object value) {  
 **try** {  
 *//redisTemplate.opsForList().leftPush(key, value); //左压栈* **redisTemplate**.opsForList().rightPush(key, value);*//右压栈* **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 将list放入缓存  
 \*  
 \** ***@param key*** *键  
 \** ***@param value*** *值  
 \** ***@param time*** *时间(秒)  
 \** ***@return*** *\*/* **public boolean** lSet(String key, Object value, **long** time) {  
 **try** {  
 **redisTemplate**.opsForList().rightPush(key, value);  
 **if** (time > 0) {  
 expire(key, time);  
 }  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
  
 */\*\*  
 \* 将list放入缓存  
 \*  
 \** ***@param key*** *键  
 \** ***@param value*** *值  
 \** ***@return*** *\*/* **public boolean** lSet(String key, List<Object> value) {  
 **try** {  
 **redisTemplate**.opsForList().rightPushAll(key, value);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 将list放入缓存  
 \*  
 \** ***@param key*** *键  
 \** ***@param value*** *值  
 \** ***@param time*** *时间(秒)  
 \** ***@return*** *\*/* **public boolean** lSet(String key, List<Object> value, **long** time) {  
 **try** {  
 **redisTemplate**.opsForList().rightPushAll(key, value);  
 **if** (time > 0) {  
 expire(key, time);  
 }  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 根据索引修改list中的某条数据  
 \*  
 \** ***@param key*** *键  
 \** ***@param index*** *索引  
 \** ***@param value*** *值  
 \** ***@return*** *\*/* **public boolean** lUpdateIndex(String key, **long** index, Object value) {  
 **try** {  
 **redisTemplate**.opsForList().set(key, index, value);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 移除N个值为value  
 \*  
 \** ***@param key*** *键  
 \** ***@param count*** *移除多少个  
 \** ***@param value*** *值  
 \** ***@return*** *移除的个数  
 \*/* **public long** lRemove(String key, **long** count, Object value) {  
 **try** {  
 Long remove = **redisTemplate**.opsForList().remove(key, count, value);  
 **return** remove;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 }  
}

# 配置类

@Configuration  
@AutoConfigureAfter(RedisAutoConfiguration.**class**)  
**public class** RedisConfig {  
  
 @Value(**"${spring.redis.database:0}"**)  
 **private int database**;  
  
 @Value(**"${spring.redis.host}"**)  
 **private** String **host**;  
  
 @Value(**"${spring.redis.password}"**)  
 **private** String **password**;  
  
 @Value(**"${spring.redis.port}"**)  
 **private int port**;  
  
 @Value(**"${spring.redis.timeout}"**)  
 **private long timeout**;  
  
 @Value(**"${spring.redis.lettuce.pool.max-idle}"**)  
 **private int maxIdle**;  
  
 @Value(**"${spring.redis.lettuce.pool.min-idle}"**)  
 **private int minIdle**;  
  
 @Value(**"${spring.redis.lettuce.pool.max-active}"**)  
 **private int maxActive**;  
  
 @Value(**"${spring.redis.lettuce.pool.max-wait}"**)  
 **private long maxWait**;  
  
 */\*\*  
 \* 基础配置  
 \*/* @Bean  
 **public** LettuceConnectionFactory lettuceConnectionFactory(GenericObjectPoolConfig genericObjectPoolConfig) {  
  
 RedisStandaloneConfiguration redisStandaloneConfiguration = **new** RedisStandaloneConfiguration();  
 redisStandaloneConfiguration.setDatabase(**database**);  
 redisStandaloneConfiguration.setHostName(**host**);  
 redisStandaloneConfiguration.setPort(**port**);  
 redisStandaloneConfiguration.setPassword(RedisPassword.*of*(**password**));  
  
 LettuceClientConfiguration clientConfig = LettucePoolingClientConfiguration.*builder*()  
 .commandTimeout(Duration.*ofMillis*(**timeout**))  
 .poolConfig(genericObjectPoolConfig)  
 .build();  
  
 LettuceConnectionFactory factory = **new** LettuceConnectionFactory(redisStandaloneConfiguration, clientConfig);  
 **return** factory;  
  
 }  
  
 */\*\*  
 \* GenericObjectPoolConfig 连接池配置  
 \*/* @Bean  
 **public** GenericObjectPoolConfig genericObjectPoolConfig() {  
 GenericObjectPoolConfig genericObjectPoolConfig = **new** GenericObjectPoolConfig();  
 genericObjectPoolConfig.setMaxIdle(**maxIdle**);  
 genericObjectPoolConfig.setMinIdle(**minIdle**);  
 genericObjectPoolConfig.setMaxTotal(**maxActive**);  
 genericObjectPoolConfig.setMaxWaitMillis(**maxWait**);  
 **return** genericObjectPoolConfig;  
 }  
  
 */\*\*  
 \* 默认情况下的模板只能支持RedisTemplate<String,String>，只能存字符串。  
 \* 这时需要自定义模板，当自定义模板后又想存储String字符串时，可以使用StringRedisTemplate的方式。  
 \*/* @Bean  
 **public** RedisTemplate<String, Object> redisCacheTemplate(LettuceConnectionFactory redisConnectionFactory) {  
  
 RedisTemplate<String, Object> template = **new** RedisTemplate<String, Object>();  
 template.setConnectionFactory(redisConnectionFactory);  
 Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = **new** Jackson2JsonRedisSerializer(Object.**class**);  
 ObjectMapper om = **new** ObjectMapper();  
 om.setVisibility(PropertyAccessor.***ALL***, JsonAutoDetect.Visibility.***ANY***);  
 om.enableDefaultTyping(ObjectMapper.DefaultTyping.***NON\_FINAL***);  
 jackson2JsonRedisSerializer.setObjectMapper(om);  
 StringRedisSerializer stringRedisSerializer = **new** StringRedisSerializer();  
 *// key采用String的序列化方式* template.setKeySerializer(stringRedisSerializer);  
 *// hash的key也采用String的序列化方式* template.setHashKeySerializer(stringRedisSerializer);  
 *// value序列化方式采用jackson* template.setValueSerializer(jackson2JsonRedisSerializer);  
 *// hash的value序列化方式采用jackson* template.setHashValueSerializer(jackson2JsonRedisSerializer);  
 template.afterPropertiesSet();  
 **return** template;  
  
 }  
}

# 调用:

@Autowired  
**private** RedisTemplate<String,Object> **redisTemplate**;  
  
@Autowired  
**private** StringRedisTemplate **stringRedisTemplate**;  
  
@Autowired  
**private** RedisUtil **redisUtil**;