redis之lua脚本学习

在实际工作过程中，可以使用lua脚本来解决一些需要保证原子性的问题，而且lua脚本可以缓存在redis服务器上，势必会增加性能。不过lua也会有很多限制，在使用的时候要注意。

demo

/\*\*

\* lua脚本

\*/

@Test

public void script() throws InterruptedException {

/\*

\* 其中 "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}" 是被求值的 Lua 脚本，数字 2 指定了键名参数的数量， key1 和 key2 是键名参数，分别使用 KEYS[1] 和 KEYS[2] 访问，而最后的 first 和 second 则是附加参数，可以通过 ARGV[1] 和 ARGV[2] 访问它们。

\* 注意，这里一些操作不适用于redis-cluster，主要还是因为不同的key被分配到了不同的slot中

\*/

Object eval = jedisCluster.eval("return {KEYS[1],ARGV[1],ARGV[2]}", 1, "lua", "key1", "dd");

System.out.println(eval);

//脚本里使用的所有键都应该由 KEYS 数组来传递：

//因为：所有的 Redis 命令，在执行之前都会被分析，籍此来确定命令会对哪些键进行操作。因此，对于 EVAL 命令来说，必须使用正确的形式来传递键，才能确保分析工作正确地执行

System.out.println(jedisCluster.eval("return redis.call('set', KEYS[1], ARGV[1])", 1, "luaTest", "cv"));

System.out.println(jedisCluster.get("luaTest"));

//注意这里需要指定KEY，因为这里lua脚本也是和slot挂钩的

String scriptLoad = jedisCluster.scriptLoad("return redis.call('get', KEYS[1])", "luaTest");//加载脚本

System.out.println(scriptLoad);//返回的SHA1校验和，后续可以直接使用这个进行操作。

System.out.println(jedisCluster.scriptExists(scriptLoad, "luaTest"));//检查是否存在

System.out.println(jedisCluster.evalsha(scriptLoad, 1, "luaTest"));//执行lua脚本

System.out.println(jedisCluster.scriptFlush("luaTest".getBytes()));//删除KEY as 上的所有lua脚本

System.out.println(jedisCluster.scriptExists(scriptLoad, "luaTest"));

System.out.println(jedisCluster.evalsha(scriptLoad, 1, "luaTest"));//脚本已经删除，返回错误：NOSCRIPT No matching script. Please use EVAL.

}

/\*\*

\* redis中的lua脚本做了很多限制，防止随机性的发生。比如lua脚本中返回的总是有序的集合。

\* 详情见 http://doc.redisfans.com/script/eval.html - 纯函数脚本

\*/

@Test

public void scriptFuc() throws InterruptedException {

String key = "luaTest";

System.out.println(jedisCluster.del(key));

System.out.println(jedisCluster.sadd(key, "10","3","7","40","6"));

System.out.println(jedisCluster.smembers(key));//这里怎么返回的值是有序的？ [3, 6, 7, 10, 40]

System.out.println(jedisCluster.eval("return redis.call('smembers', KEYS[1])", 1, key));//根据字母序排序 [10, 3, 40, 6, 7]

}

spring boot中使用lua脚本

前提：首先要引入spring-boot-starter-data-redis依赖，使用redisTemplate。该配置项不在本文讲述，只讲述lua脚本的使用。

编写lua脚本

该脚本功能：先检查redis中某个key的值是否与期望的值V1一致，如果一致则将其修改为新的值V2并返回true，否则返回false。其实就是CAS。

lua:

local current = redis.call('GET', KEYS[1])

if current == ARGV[1]

then redis.call('SET', KEYS[1], ARGV[2])

return true

end

return false

[ 注意：lua脚本中的变量都要是local的，不可以是全局变量。否则会报错。详见[http://doc.redisfans.com/script/eval.html#id6](http://doc.redisfans.com/script/eval.html" \l "id6) ]

使用DefaultRedisScript加载lua脚本

我们应该在应用上下文中配置一个DefaultRedisScript的单例，避免在每个脚本执行的时候重复创建脚本的SHA1。

@Bean

public DefaultRedisScript<Boolean> redisScript() {

DefaultRedisScript<Boolean> redisScript = new DefaultRedisScript<>();

redisScript.setScriptSource(new ResourceScriptSource(new ClassPathResource("lua/checkandset.lua")));

redisScript.setResultType(Boolean.class);

return redisScript;

}

java代码中引用

@Autowired

DefaultRedisScript<Boolean> redisScript;

/\*\*

\* 测试redis lua

\*

\* @return

\*/

@RequestMapping(value = "testredislua", method = {RequestMethod.GET})

@ResponseBody

public Object testredislua() {

String key = "testredislua";

redisTemplate.delete(key);

redisTemplate.opsForValue().set(key, "hahaha");

String s = redisTemplate.opsForValue().get(key);

log.info(s);

redisTemplate.execute(redisScript, Collections.singletonList(key), "hahaha", "3333");

s = redisTemplate.opsForValue().get(key);

log.info(s);

return null;

}