redis

一、strings 类型及操作

1、set

设置key 对应的值为string 类型的value

```
127.0.0.1:6379[1]> set name leilanjie
OK
127.0.0.1:6379[1]> get name
"leilanjie"
```

2、setnx

设置key 对应的值为string 类型的value。如果key 已经存在，返回0，nx 是not exist 的意思

```
127.0.0.1:6379[1]> setnx name leilanjie_new
(integer) 0                                    //本次修改不生效，且返回码是0
```

3、setex

设置key 对应的值为string 类型的value，并指定此键值对应的有效期

```
127.0.0.1:6379[1]> setex haircolor 10 red
OK
127.0.0.1:6379[1]> get haircolor
"red"
127.0.0.1:6379[1]> get haircolor
(nil)                              //10s后失效
```

4、setrange

设置指定key 的value 值的子字符串

```
127.0.0.1:6379[1]> set email leilanjie@qq.com
OK
127.0.0.1:6379[1]> setrange email 10 163.com
(integer) 17
127.0.0.1:6379[1]> get email
"leilanjie@163.com"
```

5、mset

一次设置多个key 的值，成功返回ok 表示所有的值都设置了，失败返回0 表示没有任何值被设置

```
127.0.0.1:6379[1]> mset key1 leilanjie1 key2 leilanjie2
OK
```

6、msetnx

一次设置多个key 的值，成功返回ok 表示所有的值都设置了，失败返回0 表示没有任何值被设置，但是不会覆盖已经存在的key

```
127.0.0.1:6379[1]> msetnx key2 leilanjie_new key3 leilanjie3
(integer) 0
```

7、get

获取key 对应的string 值,如果key 不存在返回nil

8、getset

设置key 的值，并返回key 的旧值

```
127.0.0.1:6379[1]> getset name leilanjie_new
"leilanjie"
127.0.0.1:6379[1]> get name
"leilanjie_new"
```

9、getrange

获取指定key 的value 值的子字符串

```
127.0.0.1:6379[1]> get name
"leilanjie_new"
127.0.0.1:6379[1]> getrange name 0 6
"leilanj"              //左下标从0开始
127.0.0.1:6379[1]> getrange name -7 -1
"jie_new"             //右下标从-1开始
```

10、mget

一次获取多个key 的值，如果对应key 不存在，则对应返回nil

```
127.0.0.1:6379[1]> mget key1 key2 key3
1) "leilanjie1"
2) "leilanjie2"
3) (nil)           //key3不存在，返回nil
```

11、incr

对key 的值做加加操作,并返回新的值, incr 一个不是int 的value 会返回错误，incr 一个不存在的key，则设置key 为1

```
127.0.0.1:6379[1]> set age 20
OK
127.0.0.1:6379[1]> incr age
(integer) 21
```

12、incrby

同incr 类似，加指定值 ， key 不存在时候会设置key，并认为原来的value 是 0

```
127.0.0.1:6379[1]> incrby age 5
(integer) 26
```

13、decr

对key 的值做的是减减操作，decr 一个不存在key，则设置key 为-1

```
127.0.0.1:6379[1]> decr age
(integer) 25
```

14、decrby

同decr，减指定值

```
127.0.0.1:6379[1]> decrby age 5
(integer) 20
127.0.0.1:6379[1]> incrby age -5
(integer) 15                          //与decrby功能类似
```

15、append

给指定key 的字符串值追加value,返回新字符串值的长度

```
127.0.0.1:6379[1]> append name @163.com
(integer) 21
127.0.0.1:6379[1]> get name
"leilanjie_new@163.com"
```

16、strlen

取指定key 的value 值的长度

```
127.0.0.1:6379[1]> strlen name
(integer) 21
```

二、hashes数据类型及操作

1、hset

设置hash field 为指定值，如果key 不存在，则先创建

```
127.0.0.1:6379[1]> hset myhash field1 hello
(integer) 1
```

2、hsetnx

设置hash field 为指定值，如果key 不存在，则先创建。如果field 已经存在，返回0，nx 是not exist 的意思

```
127.0.0.1:6379[1]> hsetnx myhash field "hello"
(integer) 1
127.0.0.1:6379[1]> hsetnx myhash field "hello"
(integer) 0                                 //field已经存在，返回0
```

3、hmset

同时设置hash 的多个field

```
127.0.0.1:6379[1]> hmset myhash field1 hello field2 world
OK
```

4、hget

获取指定的hash field

```
127.0.0.1:6379[1]> hget myhash field1
"hello"
```

5、hmget

获取全部指定的hash filed

```
127.0.0.1:6379[1]> hmget myhash field1 field2 field3
1) "hello"
2) "world"
3) (nil)
```

6、hincrby

指定的hash filed 加上给定值

```
127.0.0.1:6379[1]> hset myhash field3 20
(integer) 1
127.0.0.1:6379[1]> hincrby myhash field3 -8
(integer) 12
127.0.0.1:6379[1]> hget myhash field3
"12"
```

7、hexists

测试指定field 是否存在

```
127.0.0.1:6379[1]> hexists myhash field3
(integer) 1
```

8、hlen

返回指定hash 的field 数量

```
127.0.0.1:6379[1]> hlen myhash
(integer) 4
```

9、hdel

删除指定hash的field

```
127.0.0.1:6379[1]> hdel myhash field
(integer) 1
```

10、hkeys

返回hash 的所有field

```
127.0.0.1:6379[1]> hkeys myhash
1) "field1"
2) "field2"
3) "field3"
```

11、hvals

返回hash 的所有value

```
127.0.0.1:6379[1]> hvals myhash
1) "hello"
2) "world"
3) "12"
```

12、hgetall

获取某个hash 中全部的filed 及value

```
127.0.0.1:6379[1]> hgetall myhash
1) "field1"
2) "hello"
3) "field2"
4) "world"
5) "field3"
6) "12"
```

三、lists类型及操作

1、lpush

在key 对应list 的头部添加字符串元素

```
127.0.0.1:6379> lpush mylist "world"
(integer) 1
127.0.0.1:6379> lpush mylist "hello"
(integer) 2
127.0.0.1:6379> lrange mylist 0 -1
1) "hello"
2) "world"
```

其中lrange 是用于取mylist 的内容.

2、rpush

在key 对应list 的尾部添加字符串元素

```
127.0.0.1:6379> rpush mylist2 "hello"
(integer) 1
127.0.0.1:6379> rpush mylist2 "world"
(integer) 2
127.0.0.1:6379> lrange mylist 0 -1
1) "hello"
2) "world"
```

3、linsert

在key 对应list 的特定位置之前或之后添加字符串元素

```
127.0.0.1:6379> rpush mylist3 "hello"
(integer) 1
127.0.0.1:6379> rpush mylist3 "world"
(integer) 2
127.0.0.1:6379> linsert mylist3 before "world" "there"
(integer) 3
127.0.0.1:6379> lrange mylist3 0 -1
1) "hello"
2) "there"
3) "world"
```

4、lset

设置list 中指定下标的元素值(下标从0 开始)

```
127.0.0.1:6379> rpush mylist4 "one"
(integer) 1
127.0.0.1:6379> rpush mylist4 "two"
(integer) 2
127.0.0.1:6379> lset mylist4 1 "three"
OK
127.0.0.1:6379> lrange mylist4 0 -1
1) "one"
2) "three"
```

5、lrem

从key 对应list 中删除count 个和value 相同的元素。

count>0 时，按从头到尾的顺序删除，具体如下:

```
127.0.0.1:6379> rpush mylist5 "hello"
(integer) 1
127.0.0.1:6379> rpush mylist5 "hello"
(integer) 2
127.0.0.1:6379> rpush mylist5 "foo"
(integer) 3
127.0.0.1:6379> rpush mylist5 "hello"
(integer) 4
127.0.0.1:6379> lrem mylist5 2 "hello"
(integer) 2
127.0.0.1:6379> lrange mylist5 0 -1
1) "foo"
2) "hello"
```

count<0 时，按从尾到头的顺序删除

```
127.0.0.1:6379> rpush mylist6 "hello"
(integer) 1
127.0.0.1:6379> rpush mylist6 "hello"
(integer) 2
127.0.0.1:6379> rpush mylist6 "foo"
(integer) 3
127.0.0.1:6379> rpush mylist6 "hello"
(integer) 4
127.0.0.1:6379> lrem mylist6 -2 "hello"
(integer) 2
127.0.0.1:6379> lrange mylist6 0 -1
1) "hello"
2) "foo"
```

count=0 时，删除全部

```
127.0.0.1:6379> rpush mylist7 "hello"
(integer) 1
127.0.0.1:6379> rpush mylist7 "hello"
(integer) 2
127.0.0.1:6379> rpush mylist7 "foo"
(integer) 3
127.0.0.1:6379> rpush mylist7 "hello"
(integer) 4
127.0.0.1:6379> lrem mylist7 0 "hello"
(integer) 3
127.0.0.1:6379> lrange mylist7 0 -1
1) "foo"
```

6、ltrim

保留指定key 的值范围内的数据

```
127.0.0.1:6379> rpush mylist8 "one"
(integer) 1
127.0.0.1:6379> rpush mylist8 "two"
(integer) 2
127.0.0.1:6379> rpush mylist8 "three"
(integer) 3
```

```
127.0.0.1:6379> rpush mylist8 "four"
(integer) 4
127.0.0.1:6379> ltrim mylist8 1 -1
OK
127.0.0.1:6379> lrange mylist8 0 -1
1) "two"
2) "three"
3) "four"
```

7、lpop

从list 的头部删除元素，并返回删除元素

```
127.0.0.1:6379> lrange mylist 0 -1
1) "hello"
2) "world"
3) "two"
127.0.0.1:6379> lpop mylist
"hello"
127.0.0.1:6379> lrange mylist 0 -1
1) "world"
2) "two"
```

8、rpop

从list 的尾部删除元素，并返回删除元素

```
127.0.0.1:6379> rpop mylist
"two"
127.0.0.1:6379> lrange mylist 0 -1
1) "world"
```

9、rpoplpush

从第一个list 的尾部移除元素并添加到第二个list 的头部,最后返回被移除的元素值，整个操作是原子的. 如果第一个list 是空或者不存在返回nil

```
127.0.0.1:6379> lrange mylist5 0 -1
1) "foo"
2) "hello"
127.0.0.1:6379> lrange mylist6 0 -1
1) "hello"
2) "foo"
127.0.0.1:6379> rpoplpush mylist5 mylist6
"hello"
127.0.0.1:6379> lrange mylist5 0 -1
1) "foo"
127.0.0.1:6379> lrange mylist6 0 -1
1) "hello"
2) "hello"
3) "foo"
```

10、lindex

返回名称为key 的list 中index 位置的元素

```
127.0.0.1:6379> lrange mylist6 0 -1
1) "hello"
2) "hello"
3) "foo"
127.0.0.1:6379> lindex mylist6 0
"hello"
```

11、llen

返回key 对应list 的长度

```
127.0.0.1:6379> llen mylist6
(integer) 3
```

四、sets 类型及操作

1、sadd

向名称为key 的set 中添加元素

```
127.0.0.1:6379> sadd myset "hello"
(integer) 1
127.0.0.1:6379> sadd myset "world"
(integer) 1
127.0.0.1:6379> sadd myset "world"
(integer) 0
127.0.0.1:6379> smembers myset
1) "hello"
2) "world"
```

smembers来查看myset中的所有元素

2、srem

删除名称为key 的set 中的元素member

```
127.0.0.1:6379> srem myset "world"
(integer) 1
127.0.0.1:6379> smembers myset
1) "hello"
```

3、spop

随机返回并删除名称为key 的set 中一个元素

```
127.0.0.1:6379> sadd myset1 "one"
(integer) 1
127.0.0.1:6379> sadd myset1 "two"
(integer) 1
127.0.0.1:6379> spop myset1
"two"
```

4、sdiff

返回所有给定key 与第一个key 的差集

```
127.0.0.1:6379> smembers myset1
1) "hello"
2) "one"
127.0.0.1:6379> smembers myset
1) "hello"
127.0.0.1:6379> sdiff myset myset1  //myset与myset1中不同的是空
(empty list or set)
127.0.0.1:6379> sdiff myset1 myset  //myset1与myset不同的是“one”
1) "one"
```

5、sdiffstore

返回所有给定key 与第一个key 的差集，并将结果存为另一个key

```
127.0.0.1:6379> sdiffstore myset2 myset1 myset
(integer) 1
127.0.0.1:6379> smembers myset2
1) "one"
```

6、sinter

返回所有给定key 的交集

```
127.0.0.1:6379> sinter myset myset1
1) "hello"
```

7、sinterstore

返回所有给定key 的交集，并将结果存为另一个key

```
127.0.0.1:6379> sinterstore myset2 myset myset1
(integer) 1
127.0.0.1:6379> smembers myset2
1) "hello"
```

8、sunion

返回所有给定key 的并集

```
127.0.0.1:6379> sunion myset myset1
1) "hello"
2) "one"
```

9、sunionstore

返回所有给定key 的并集，并将结果存为另一个key

```
127.0.0.1:6379> sunionstore myset3 myset myset1
(integer) 2
127.0.0.1:6379> smembers myset3
1) "hello"
2) "one"
```

10、smove

从第一个key 对应的set 中移除member 并添加到第二个对应set 中

```
127.0.0.1:6379> smove myset2 myset4 hello
(integer) 1
127.0.0.1:6379> smembers myset4
1) "hello"
```

11、scard

返回名称为key 的set 的元素个数

```
127.0.0.1:6379> scard myset4
(integer) 1
```

12、sismember

测试member 是否是名称为key 的set 的元素

```
127.0.0.1:6379> sismember myset4 hello
(integer) 1
127.0.0.1:6379> sismember myset4 hai
(integer) 0
```

13、srandmember

随机返回名称为key 的set 的一个元素，但是不删除元素

```
127.0.0.1:6379> srandmember myset3
"one"
```

五、sorted sets 类型及操作

1、zadd

向名称为key 的zset 中添加元素member，score 用于排序。如果该元素已经存在，则根据score 更新该元素的顺序

```
127.0.0.1:6379> zadd myzset 1 "one"
(integer) 1
127.0.0.1:6379> zadd myzset 2 "two"
(integer) 1
127.0.0.1:6379> zadd myzset 3 "two"
(integer) 0
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "3"
```

two 被设置了2 次，以最后一次的设置为准.

2、zrem

删除名称为key 的zset 中的元素member

```
127.0.0.1:6379> zrem myzset two
(integer) 1
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
```

3、zincrby

如果在名称为key 的zset 中已经存在元素member，则该元素的score 增加increment；否则向集合中添加该元素，其score 的值为increment

```
127.0.0.1:6379[1]> zadd myzset3 1 "one"
(integer) 1
127.0.0.1:6379[1]> zadd myzset3 2 "two"
(integer) 1
127.0.0.1:6379[1]> zincrby myzset3 2 "one"
"3"
127.0.0.1:6379[1]> zrange myzset3 0 -1 withscores
1) "two"
2) "2"
3) "one"
4) "3"
```

4、zrank

返回名称为key 的zset 中member 元素的排名(按score 从小到大排序)即下标

```
127.0.0.1:6379[1]> zrank myzset3 two
(integer) 0   //下标从零开始
```

5、zrevrank

返回名称为key 的zset 中member 元素的排名(按score 从大到小排序)即下标

```
127.0.0.1:6379[1]> zrevrank myzset3 two
(integer) 1
```

6、zrevrange

返回名称为key 的zset（按score 从大到小排序）中的index 从start 到end 的所有元素

```
127.0.0.1:6379[1]> zrevrange myzset3 0 -1 withscores
1) "one"
2) "3"
3) "two"
4) "2"
```

7、zrangebyscore

返回集合中score 在给定区间的元素

```
127.0.0.1:6379[1]> zrangebyscore myzset3 2 3 withscores
1) "two"
2) "2"
3) "one"
4) "3"
```

8、zcount

返回集合中score 在给定区间的数量

```
127.0.0.1:6379[1]> zcount myzset3 2 3
(integer) 2
```

9、zcard

返回集合中元素个数

```
127.0.0.1:6379[1]> zcard myzset3
(integer) 2
```

10、zscore

返回给定元素对应的score

```
127.0.0.1:6379[1]> zscore myzset3 one
"3"
```

11、zremrangebyrank

删除集合中排名在给定区间的元素

```
127.0.0.1:6379[1]> zremrangebyrank myzset3 1 1
(integer) 1
127.0.0.1:6379[1]> zrange myzset3 0 -1
1) "two"
```

12、zremrangebyscore

删除集合中score 在给定区间的元素

```
127.0.0.1:6379[1]> zremrangebyscore myzset3 2 2  //删除score为2的two
(integer) 1
127.0.0.1:6379[1]> zrange myzset3 0 -1
(empty list or set)      //myzset3被删除空了
```

redis常用命令

一、键值相关命令

1、keys

返回满足给定pattern 的所有key

```
127.0.0.1:6379[1]> keys *  //取出所有key
1) "myzset2"
2) "myset"
3) "myset1"
127.0.0.1:6379[1]> keys my*  //取出所有以my开头的key
1) "myzset2"
2) "myset"
3) "myset1"
```

2、exists

确认一个key 是否存在

```
3) "myset1"
127.0.0.1:6379[1]> exists myzet
(integer) 0
127.0.0.1:6379[1]> exists myzset2
(integer) 1
```

3、del

删除一个key

```
127.0.0.1:6379[1]> del myset1
(integer) 1
127.0.0.1:6379[1]> exists myset1
(integer) 0
```

4、expire

设置一个key 的过期时间(单位:秒)

```
127.0.0.1:6379[1]> expire myset 10
(integer) 1
127.0.0.1:6379[1]> ttl myset   //以秒为单位，返回给定 key 的剩余生存时间
(integer) 2
127.0.0.1:6379[1]> ttl myset
(integer) -2   // 应该为-1吧。。。。
```

5、move

将当前数据库中的key 转移到其它数据库中

```
27.0.0.1:6379[1]> set age 30
OK
127.0.0.1:6379[1]> get age
"30"
127.0.0.1:6379[1]> move age 2
(integer) 1
127.0.0.1:6379[1]> get age
(nil)                    //在数据库1中已经被移除
127.0.0.1:6379[1]> select 2
OK
127.0.0.1:6379[2]> get age
"30"                     //在数据库2中被查询
```

6、persist

移除给定key 的过期时间

```
127.0.0.1:6379[2]> expire age 300
(integer) 1
127.0.0.1:6379[2]> ttl age
(integer) 294
127.0.0.1:6379[2]> ttl age
(integer) 288
127.0.0.1:6379[2]> persist age
(integer) 1
127.0.0.1:6379[2]> ttl age
(integer) -1               //移除后，过期
```

7、randomkey

随机返回key 空间的一个key

```
127.0.0.1:6379[2]> randomkey
"sex"
```

8、rename

重命名

```
127.0.0.1:6379[2]> rename age age-new
OK
127.0.0.1:6379[2]> keys *
1) "age-new"
2) "sex"
3) "name"
```

9、type

返回值的类型

```
127.0.0.1:6379[2]> type sex
string
127.0.0.1:6379[2]> hset myhash field hello
(integer) 1
127.0.0.1:6379[2]> type myhash
hash
```

二、服务器相关命令

1、ping

测试连接是否存活

```
127.0.0.1:6379[2]> ping
PONG                              //表示连接成功
```

2、echo

在命令行打印一些内容

```
127.0.0.1:6379[2]> echo hahaha
"hahaha"
```

3、select

选择数据库。Redis 数据库编号从0~15，我们可以选择任意一个数据库来进行数据的存取

```
127.0.0.1:6379[2]> select 5
OK
127.0.0.1:6379[5]> select 16
(error) ERR invalid DB index   //最大是15
```

4、quit

退出连接

```
127.0.0.1:6379> quit
C:\Users\20190712133>
```

5、dbsize

返回当前数据库中key 的数目

```
127.0.0.1:6379> dbsize
(integer) 14
```

6、info

获取服务器的信息和统计

```
127.0.0.1:6379> info
# Server
redis_version:3.0.504
```

```
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:a4f7a6e86f2d60b3
redis_mode:standalone
os:Windows
arch_bits:64
multiplexing_api:WinSock_IOCP
process_id:7552
run_id:f6a09cdf16e1fa8fa88d156845493aed20e45ca3
tcp_port:6379
uptime_in_seconds:27151
uptime_in_days:0
hz:10
lru_clock:6704250
config_file:
......
```

7、config get

获取服务器配置信息

```
127.0.0.1:6379> config get dir
1) "dir"
2) "C:\\Users\\20190712133"
```

”config get *” 获取全部参数据的配置值

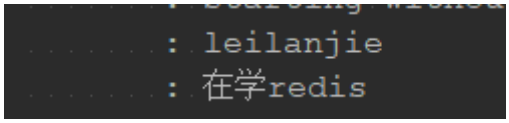
8、flushdb

删除当前选择数据库中的所有key

```
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> keys *
1) "myzset2"
127.0.0.1:6379[1]> flushdb
OK
127.0.0.1:6379[1]> keys *
(empty list or set)
```

9、flushall

删除所有数据库中的所有key



redis实现消息队列

一、发布订阅模式

消息发布：

```
    /**
```

```
     * 发布订阅
     * @param channel
     * @param message
     * @return
     */
    @RequestMapping("/view")
    @ResponseBody
    public Map<String ,Object > testChannl(@RequestParam(value = "channel")
                String channel,  @RequestParam (value = "message") String
 message){
        redisTemplate.convertAndSend(channel,message);
        Map<String, Object> map = new HashMap<String, Object>();
        map.put ( "success ", true) ;
        return map ;
    }
```

消息监听:

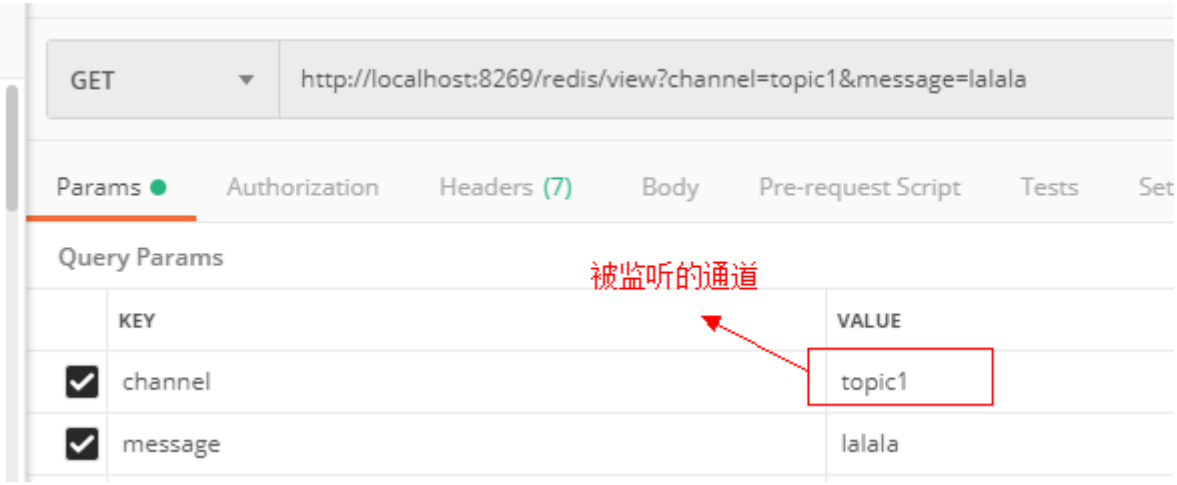
```
 /**
     * 定义Redis的监听容器
      * @return
     */
    @Bean
    public RedisMessageListenerContainer initRedisContainer () {
        RedisMessageListenerContainer container = new
RedisMessageListenerContainer();
        //Redis连接工厂
        container.setConnectionFactory(connectionFactory);
        //设置运行任务池
        container.setTaskExecutor(initTaskScheduler());
        //定义监听渠道，名称为topic1
        Topic topic = new ChannelTopic("topic1");
        //使用监听器监听Redis的消息
        container.addMessageListener(listener, topic);
        return container;
    }
```

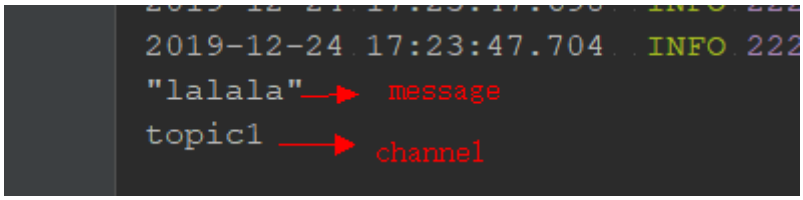
消息订阅:

```
@Component
public class RedisMessageListener implements MessageListener {

    @Override
    public void onMessage(Message message,byte[] pattern){
        //消息体
        String body = new String(message.getBody());
        //渠道名称
        String topic = new String(pattern);
        System.out.println(body);
        System.out.println(topic);
    }
}
```

测试:





二、生产者/消费者模式

原理：使用redis的list数据类型做队列，使用lpush/rpop,rpush/lpop方法实现消息的消费

生产者：

```
public class Product extends Thread{

    RedisTemplate redissss = RedisApplication.redis;
    @Override
    public void run() {
        super.run();
        int j = 101;
        for (int i = 1; i <= j; i++) {
            String s = "消息" + i;
            redissss.opsForList().leftPush("message",s);
            System.out.println("消息生产了" + s);
        }
```

生产消息

消费者：

```
lic class MessageConsumerOne extends Thread {

 RedisTemplate redissss = RedisApplication.redis;
 @Override
public void run() {
    while (true) {
        Object message = redissss.opsForList().rightPop( key: "message", timeout: 200,TimeUnit.SECONDS);
        if ( message != null) {
            System.out.println("消费者1号消费：");
            System.out.println(message);
        } else {
            System.out.println("消费者1号，队列里没有消息了");
        }
```

通过队列阻塞，实现消息消费。不会浪费资源

启动消费者与生产者线程：

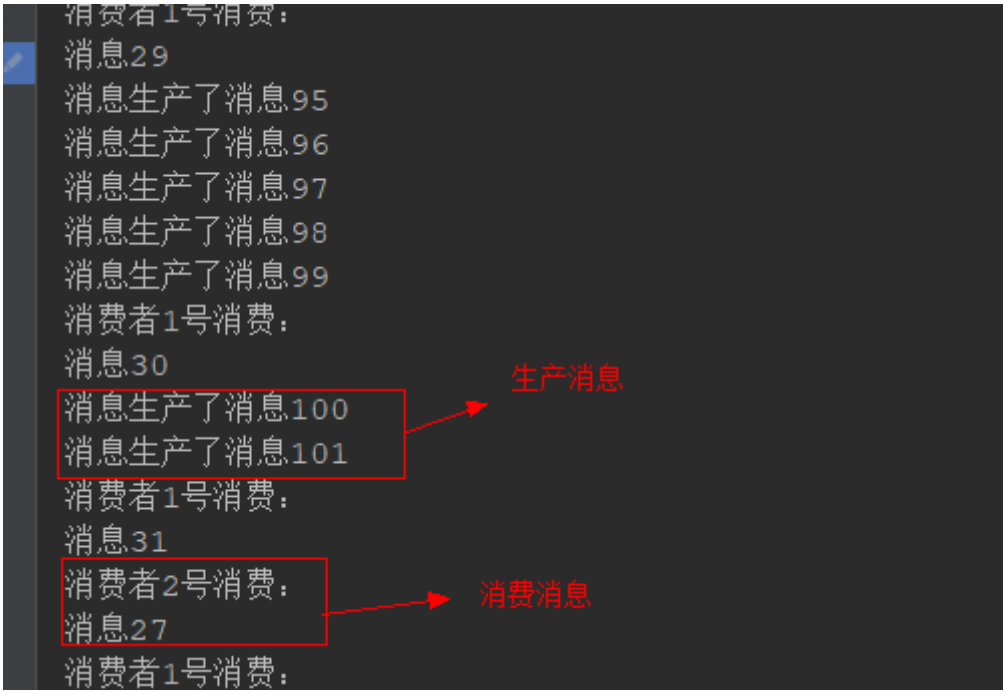
```
public static void main(String[] args) {

    SpringApplication.run(RedisApplication.class, args);
    try {
        Product product = new Product();
        MessageConsumerOne consumerOne = new MessageConsumerOne();
        MessageConsumerTwo consumerTwo = new MessageConsumerTwo();
        product.start();
        consumerOne.start();
        consumerTwo.start();
    }catch ( Exception e){
        e.printStackTrace();
    }
```

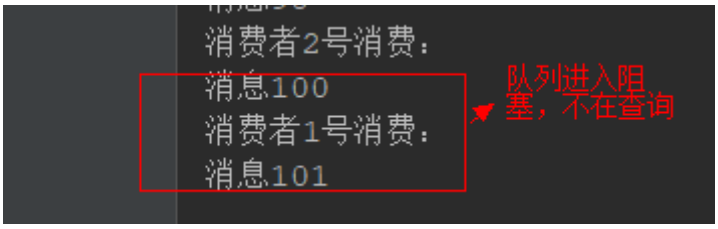
生产者

消费者

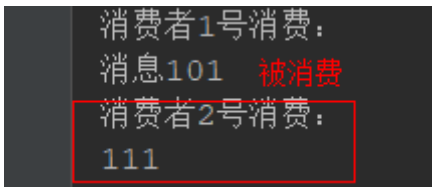
测试：



当队列无消息时，进入阻塞：



当队列中有消息时自动唤醒并消费消息：



redis常见问题

1、redisTemplate注入失败（注入后总显示null）

解决办法：

步骤一：在配置类中加入

```
@Autowired
private RedisTemplate redisTemplate;
public static RedisTemplate redis;

@PostConstruct
public void getRedisTemplate(){
    redis=this.redisTemplate;
}
```

步骤二：调用方式

```
RedisTemplate redisssss = RedisApplication.redis;
```