# Redis for Python开发手册

## redis基本命令 String

### Set

set(name, value, ex=None, px=None, nx=False, xx=False)

在Redis中设置值，默认，不存在则创建，存在则修改

参数说明：

ex，过期时间（秒）

px，过期时间（毫秒）

nx，如果设置为True，则只有name不存在时，当前set操作才执行

xx，如果设置为True，则只有name存在时，当前set操作才执行

1.ex，过期时间（秒） 这里过期时间是3秒，3秒后p，键food的值就变成None

import redis

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, decode\_responses=True)

r = redis.Redis(connection\_pool=pool)

r.set('food', 'mutton', ex=3) # key是"food" value是"mutton" 将键值对存入redis缓存

print(r.get('food')) # mutton 取出键food对应的值

2.px，过期时间（豪秒） 这里过期时间是3豪秒，3毫秒后，键foo的值就变成None

import redis

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, decode\_responses=True)

r = redis.Redis(connection\_pool=pool)

r.set('food', 'beef', px=3)

print(r.get('food'))

3.nx，如果设置为True，则只有name不存在时，当前set操作才执行 （新建）

import redis

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, decode\_responses=True)

r = redis.Redis(connection\_pool=pool)

print(r.set('fruit', 'watermelon', nx=True)) # True--不存在

# 如果键fruit不存在，那么输出是True；如果键fruit已经存在，输出是None

4.xx，如果设置为True，则只有name存在时，当前set操作才执行 （修改）

print((r.set('fruit', 'watermelon', xx=True))) # True--已经存在

# 如果键fruit已经存在，那么输出是True；如果键fruit不存在，输出是None

5.setnx(name, value)

设置值，只有name不存在时，执行设置操作（添加）

print(r.setnx('fruit1', 'banana')) # fruit1不存在，输出为True

### setex

setex(name, value, time)

设置值

参数：

time，过期时间（数字秒 或 timedelta对象）

import redis

import time

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, decode\_responses=True)

r = redis.Redis(connection\_pool=pool)

r.setex("fruit2", "orange", 5)

time.sleep(5)

print(r.get('fruit2')) # 5秒后，取值就从orange变成None

### psetex

psetex(name, time\_ms, value)

设置值

参数：

time\_ms，过期时间（数字毫秒 或 timedelta对象）

r.psetex("fruit3", 5000, "apple")

time.sleep(5)

print(r.get('fruit3')) # 5000毫秒后，取值就从apple变成None

8.mset(args, \*kwargs)

批量设置值

如：

r.mget({'k1': 'v1', 'k2': 'v2'})

r.mset(k1="v1", k2="v2") # 这里k1 和k2 不能带引号 一次设置对个键值对

print(r.mget("k1", "k2")) # 一次取出多个键对应的值

print(r.mget("k1"))

### mget

mget(keys, \*args)

批量获取

如：

print(r.mget('k1', 'k2'))

print(r.mget(['k1', 'k2']))

print(r.mget("fruit", "fruit1", "fruit2", "k1", "k2")) # 将目前redis缓存中的键对应的值批量取出来

### getset

getset(name, value)

设置新值并获取原来的值

print(r.getset("food", "barbecue")) # 设置的新值是barbecue 设置前的值是beef1

### getrange

getrange(key, start, end)

获取子序列（根据字节获取，非字符）

参数：

name，Redis 的 name

start，起始位置（字节）

end，结束位置（字节）

如： “君惜大大” ，0-3表示 “君”

r.set("cn\_name", "君惜大大") # 汉字

print(r.getrange("cn\_name", 0, 2)) # 取索引号是0-2 前3位的字节 君 切片操作 （一个汉字3个字节 1个字母一个字节 每个字节8bit）

print(r.getrange("cn\_name", 0, -1)) # 取所有的字节 君惜大大 切片操作

r.set("en\_name","junxi") # 字母

print(r.getrange("en\_name", 0, 2)) # 取索引号是0-2 前3位的字节 jun 切片操作 （一个汉字3个字节 1个字母一个字节 每个字节8bit）

print(r.getrange("en\_name", 0, -1)) # 取所有的字节 junxi 切片操作

### setrange

setrange(name, offset, value)

修改字符串内容，从指定字符串索引开始向后替换（新值太长时，则向后添加）

参数：

offset，字符串的索引，字节（一个汉字三个字节）

value，要设置的值

r.setrange("en\_name", 1, "ccc")

print(r.get("en\_name")) # jccci 原始值是junxi 从索引号是1开始替换成ccc 变成 jccci2

### setbit

setbit(name, offset, value)

对name对应值的二进制表示的位进行操作

参数：

name，redis的name

offset，位的索引（将值变换成二进制后再进行索引）

value，值只能是 1 或 0

注：如果在Redis中有一个对应： n1 = "foo"，

那么字符串foo的二进制表示为：01100110 01101111 01101111

所以，如果执行 setbit('n1', 7, 1)，则就会将第7位设置为1，

那么最终二进制则变成 01100111 01101111 01101111，即："goo"

扩展，转换二进制表示：

source = "郭加磊"

source = "foo"

for i in source:

num = ord(i)

print bin(num).replace('b','')

特别的，如果source是汉字 "郭加磊"怎么办？

答：对于utf-8，每一个汉字占 3 个字节，那么 "郭加磊" 则有 9个字节

对于汉字，for循环时候会按照 字节 迭代，那么在迭代时，将每一个字节转换 十进制数，然后再将十进制数转换成二进制

11100110 10101101 10100110 11100110 10110010 10011011 11101001 10111101 10010000

### getbit

getbit(name, offset)

获取name对应的值的二进制表示中的某位的值 （0或1）

print(r.getbit("foo1", 0)) # 0 foo1 对应的二进制 4个字节 32位 第0位是0还是1

### bitcount

bitcount(key, start=None, end=None)

获取name对应的值的二进制表示中 1 的个数

参数：

key，Redis的name

start 字节起始位置

end，字节结束位置

print(r.get("foo")) # goo1 01100111

print(r.bitcount("foo",0,1)) # 11 表示前2个字节中，1出现的个数

### bittop

bitop(operation, dest, \*keys)

获取多个值，并将值做位运算，将最后的结果保存至新的name对应的值

参数：

operation,AND（并） 、 OR（或） 、 NOT（非） 、 XOR（异或）

dest, 新的Redis的name

\*keys,要查找的Redis的name

如：

bitop("AND", 'new\_name', 'n1', 'n2', 'n3')

获取Redis中n1,n2,n3对应的值，然后讲所有的值做位运算（求并集），然后将结果保存 new\_name 对应的值中

r.set("foo","1") # 0110001

r.set("foo1","2") # 0110010

print(r.mget("foo","foo1")) # ['goo1', 'baaanew']

print(r.bitop("AND","new","foo","foo1")) # "new" 0 0110000

print(r.mget("foo","foo1","new"))

source = "12"

for i in source:

num = ord(i)

print(num) # 打印每个字母字符或者汉字字符对应的ascii码值 f-102-0b100111-01100111

print(bin(num)) # 打印每个10进制ascii码值转换成二进制的值 0b1100110（0b表示二进制）

print bin(num).replace('b','') # 将二进制0b1100110替换成01100110

### strlen

strlen(name)

返回name对应值的字节长度（一个汉字3个字节）

print(r.strlen("foo")) # 4 'goo1'的长度是4

### incr

incr(self, name, amount=1)

自增 name对应的值，当name不存在时，则创建name＝amount，否则，则自增。

参数：

name,Redis的name

amount,自增数（必须是整数）

注：同incrby

r.set("foo", 123)

print(r.mget("foo", "foo1", "foo2", "k1", "k2"))

r.incr("foo", amount=1)

print(r.mget("foo", "foo1", "foo2", "k1", "k2"))

应用场景 – 页面点击数

假定我们对一系列页面需要记录点击次数。例如论坛的每个帖子都要记录点击次数，而点击次数比回帖的次数的多得多。如果使用关系数据库来存储点击，可能存在大量的行级锁争用。所以，点击数的增加使用redis的INCR命令最好不过了。

当redis服务器启动时，可以从关系数据库读入点击数的初始值（12306这个页面被访问了34634次）

r.set("visit:12306:totals", 34634)

print(r.get("visit:12306:totals"))

每当有一个页面点击，则使用INCR增加点击数即可。

r.incr("visit:12306:totals")

r.incr("visit:12306:totals")

页面载入的时候则可直接获取这个值

print(r.get("visit:12306:totals"))

### incrfloat

incrbyfloat(self, name, amount=1.0)

自增 name对应的值，当name不存在时，则创建name＝amount，否则，则自增。

参数：

name,Redis的name

amount,自增数（浮点型）

r.set("foo1", "123.0")

r.set("foo2", "221.0")

print(r.mget("foo1", "foo2"))

r.incrbyfloat("foo1", amount=2.0)

r.incrbyfloat("foo2", amount=3.0)

print(r.mget("foo1", "foo2"))

### decr

decr(self, name, amount=1)

自减 name对应的值，当name不存在时，则创建name＝amount，否则，则自减。

参数：

name,Redis的name

amount,自减数（整数)

r.decr("foo4", amount=3) # 递减3

r.decr("foo1", amount=1) # 递减1

print(r.mget("foo1", "foo4"))

### append

append(key, value)

在redis name对应的值后面追加内容

参数：

key, redis的name

value, 要追加的字符串

r.append("name", "haha") # 在name对应的值junxi后面追加字符串haha

print(r.mget("name"))

## hash操作

### hset

1.单个增加–修改(单个取出)–没有就新增，有的话就修改

hset(name, key, value)

name对应的hash中设置一个键值对（不存在，则创建；否则，修改）

参数：

name，redis的name

key，name对应的hash中的key

value，name对应的hash中的value

注：

hsetnx(name, key, value),当name对应的hash中不存在当前key时则创建（相当于添加）

import redis

import time

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, decode\_responses=True)

r = redis.Redis(connection\_pool=pool)

r.hset("hash1", "k1", "v1")

r.hset("hash1", "k2", "v2")

print(r.hkeys("hash1")) # 取hash中所有的key

print(r.hget("hash1", "k1")) # 单个取hash的key对应的值

print(r.hmget("hash1", "k1", "k2")) # 多个取hash的key对应的值

r.hsetnx("hash1", "k2", "v3") # 只能新建

print(r.hget("hash1", "k2"))

### hmset

批量增加（取出）

hmset(name, mapping)

在name对应的hash中批量设置键值对

参数：

name，redis的name

mapping，字典，如：{‘k1’:’v1’, ‘k2’: ‘v2’}

如：

r.hmset("hash2", {"k2": "v2", "k3": "v3"})

hget(name,key)

在name对应的hash中获取根据key获取value

### hmget

hmget(name, keys, \*args)

在name对应的hash中获取多个key的值

参数：

name，reids对应的name

keys，要获取key集合，如：[‘k1’, ‘k2’, ‘k3’]

\*args，要获取的key，如：k1,k2,k3

如：

print(r.hget("hash2", "k2")) # 单个取出"hash2"的key-k2对应的value

print(r.hmget("hash2", "k2", "k3")) # 批量取出"hash2"的key-k2 k3对应的value --方式1

print(r.hmget("hash2", ["k2", "k3"])) # 批量取出"hash2"的key-k2 k3对应的value

### hgetall

取出所有的键值对

hgetall(name)

获取name对应hash的所有键值

print(r.hgetall("hash1"))

### hlen

得到所有键值对的格式 hash长度

hlen(name)

获取name对应的hash中键值对的个数

print(r.hlen("hash1"))

### hkeys

得到所有的keys（类似字典的取所有keys）

hkeys(name)

获取name对应的hash中所有的key的值

print(r.hkeys("hash1"))

### hvals

得到所有的value（类似字典的取所有value）

hvals(name)

获取name对应的hash中所有的value的值

print(r.hvals("hash1"))

### hexists

判断成员是否存在（类似字典的in）

hexists(name, key)

检查name对应的hash是否存在当前传入的key

print(r.hexists("hash1", "k4")) # False 不存在

print(r.hexists("hash1", "k1")) # True 存在

### hdel

删除键值对

hdel(name,\*keys)

将name对应的hash中指定key的键值对删除

print(r.hgetall("hash1"))

r.hset("hash1", "k2", "v222") # 修改已有的key k2

r.hset("hash1", "k11", "v1") # 新增键值对 k11

r.hdel("hash1", "k1") # 删除一个键值对

print(r.hgetall("hash1"))

#### hincrby

自增自减整数(将key对应的value–整数 自增1或者2，或者别的整数 负数就是自减)

hincrby(name, key, amount=1)

自增name对应的hash中的指定key的值，不存在则创建key=amount

参数：

name，redis中的name

key， hash对应的key

amount，自增数（整数）

r.hset("hash1", "k3", 123)

r.hincrby("hash1", "k3", amount=-1)

print(r.hgetall("hash1"))

r.hincrby("hash1", "k4", amount=1) # 不存在的话，value默认就是1

print(r.hgetall("hash1"))

### hincrbyfloat

自增自减浮点数(将key对应的value–浮点数 自增1.0或者2.0，或者别的浮点数 负数就是自减)

hincrbyfloat(name, key, amount=1.0)

自增name对应的hash中的指定key的值，不存在则创建key=amount

参数：

name，redis中的name

key， hash对应的key

amount，自增数（浮点数）

自增name对应的hash中的指定key的值，不存在则创建key=amount

r.hset("hash1", "k5", "1.0")

r.hincrbyfloat("hash1", "k5", amount=-1.0) # 已经存在，递减-1.0

print(r.hgetall("hash1"))

r.hincrbyfloat("hash1", "k6", amount=-1.0) # 不存在，value初始值是-1.0 每次递减1.0

print(r.hgetall("hash1"))

### hscan

取值查看–分片读取

hscan(name, cursor=0, match=None, count=None)

增量式迭代获取，对于数据大的数据非常有用，hscan可以实现分片的获取数据，并非一次性将数据全部获取完，从而放置内存被撑爆

参数：

name，redis的name

cursor，游标（基于游标分批取获取数据）

match，匹配指定key，默认None 表示所有的key

count，每次分片最少获取个数，默认None表示采用Redis的默认分片个数

如：

第一次：cursor1, data1 = r.hscan(‘xx’, cursor=0, match=None, count=None)

第二次：cursor2, data1 = r.hscan(‘xx’, cursor=cursor1, match=None, count=None)

…

直到返回值cursor的值为0时，表示数据已经通过分片获取完毕

print(r.hscan("hash1"))

### hscan\_iter

hscan\_iter(name, match=None, count=None)

利用yield封装hscan创建生成器，实现分批去redis中获取数据

参数：

match，匹配指定key，默认None 表示所有的key

count，每次分片最少获取个数，默认None表示采用Redis的默认分片个数

如：

for item in r.hscan\_iter('hash1'):

print(item)

print(r.hscan\_iter("hash1")) # 生成器内存地址

## list列表操作

### lpush

增加（类似于list的append，只是这里是从左边新增加）–没有就新建

lpush(name,values)

在name对应的list中添加元素，每个新的元素都添加到列表的最左边

如：

import redis

import time

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, decode\_responses=True)

r = redis.Redis(connection\_pool=pool)

r.lpush("list1", 11, 22, 33)

print(r.lrange('list1', 0, -1))

保存顺序为: 33,22,11

扩展：

r.rpush("list2", 11, 22, 33) # 表示从右向左操作

print(r.llen("list2")) # 列表长度

print(r.lrange("list2", 0, 3)) # 切片取出值，范围是索引号0-3

### rpush

增加（从右边增加）–没有就新建

r.rpush("list2", 44, 55, 66) # 在列表的右边，依次添加44,55,66

print(r.llen("list2")) # 列表长度

print(r.lrange("list2", 0, -1)) # 切片取出值，范围是索引号0到-1(最后一个元素)

### lpushx

往已经有的name的列表的左边添加元素，没有的话无法创建

lpushx(name,value)

在name对应的list中添加元素，只有name已经存在时，值添加到列表的最左边

更多：

r.lpushx("list10", 10) # 这里list10不存在

print(r.llen("list10")) # 0

print(r.lrange("list10", 0, -1)) # []

r.lpushx("list2", 77) # 这里"list2"之前已经存在，往列表最左边添加一个元素，一次只能添加一个

print(r.llen("list2")) # 列表长度

print(r.lrange("list2", 0, -1)) # 切片取出值，范围是索引号0到-1(最后一个元素

### rpushx

往已经有的name的列表的右边添加元素，没有的话无法创建

r.rpushx("list2", 99) # 这里"foo\_list1"之前已经存在，往列表最右边添加一个元素，一次只能添加一个

print(r.llen("list2")) # 列表长度

print(r.lrange("list2", 0, -1)) # 切片取出值，范围是索引号0到-1(最后一个元素)

### linsert

新增（固定索引号位置插入元素）

linsert(name, where, refvalue, value))

在name对应的列表的某一个值前或后插入一个新值

参数：

name，redis的name

where，BEFORE或AFTER

refvalue，标杆值，即：在它前后插入数据

value，要插入的数据

r.linsert("list2", "before", "11", "00") # 往列表中左边第一个出现的元素"11"前插入元素"00"

print(r.lrange("list2", 0, -1)) # 切片取出值，范围是索引号0-最后一个元素

### lset

修改（指定索引号进行修改）

r.lset(name, index, value)

对name对应的list中的某一个索引位置重新赋值

参数：

name，redis的name

index，list的索引位置

value，要设置的值

r.lset("list2", 0, -11) # 把索引号是0的元素修改成-11

print(r.lrange("list2", 0, -1))

### lrem

删除（指定值进行删除）

r.lrem(name, value, num)

在name对应的list中删除指定的值

参数：

name，redis的name

value，要删除的值

num， num=0，删除列表中所有的指定值；

num=2,从前到后，删除2个； num=1,从前到后，删除左边第1个

num=-2,从后向前，删除2个

r.lrem("list2", "11", 1) # 将列表中左边第一次出现的"11"删除

print(r.lrange("list2", 0, -1))

r.lrem("list2", "99", -1) # 将列表中右边第一次出现的"99"删除

print(r.lrange("list2", 0, -1))

r.lrem("list2", "22", 0) # 将列表中所有的"22"删除

print(r.lrange("list2", 0, -1))

### lpop

删除并返回

lpop(name)

在name对应的列表的左侧获取第一个元素并在列表中移除，返回值则是第一个元素

更多：

rpop(name) 表示从右向左操作

r.lpop("list2") # 删除列表最左边的元素，并且返回删除的元素

print(r.lrange("list2", 0, -1))

r.rpop("list2") # 删除列表最右边的元素，并且返回删除的元素

print(r.lrange("list2", 0, -1))

### ltrim

删除索引之外的值

ltrim(name, start, end)

在name对应的列表中移除没有在start-end索引之间的值

参数：

name，redis的name

start，索引的起始位置

end，索引结束位置

r.ltrim("list2", 0, 2) # 删除索引号是0-2之外的元素，值保留索引号是0-2的元素

print(r.lrange("list2", 0, -1))

### lindex

取值（根据索引号取值）

lindex(name, index)

在name对应的列表中根据索引获取列表元素

print(r.lindex("list2", 0)) # 取出索引号是0的值

### rpoplpush

移动 元素从一个列表移动到另外一个列表

rpoplpush(src, dst)

从一个列表取出最右边的元素，同时将其添加至另一个列表的最左边

参数：

src，要取数据的列表的name

dst，要添加数据的列表的name

r.rpoplpush("list1", "list2")

print(r.lrange("list2", 0, -1))

### brpoplpush

移动 元素从一个列表移动到另外一个列表 可以设置超时

brpoplpush(src, dst, timeout=0)

从一个列表的右侧移除一个元素并将其添加到另一个列表的左侧

参数：

src，取出并要移除元素的列表对应的name

dst，要插入元素的列表对应的name

timeout，当src对应的列表中没有数据时，阻塞等待其有数据的超时时间（秒），0 表示永远阻塞

r.brpoplpush("list1", "list2", timeout=2)

print(r.lrange("list2", 0, -1))

### blpop

一次移除多个列表

blpop(keys, timeout)

将多个列表排列，按照从左到右去pop对应列表的元素

参数：

keys，redis的name的集合

timeout，超时时间，当元素所有列表的元素获取完之后，阻塞等待列表内有数据的时间（秒）, 0 表示永远阻塞

更多：

r.brpop(keys, timeout) 同blpop，将多个列表排列,按照从右像左去移除各个列表内的元素

r.lpush("list10", 3, 4, 5)

r.lpush("list11", 3, 4, 5)

while True:

r.blpop(["list10", "list11"], timeout=2)

print(r.lrange("list10", 0, -1), r.lrange("list11", 0, -1))

### 自定义增量迭代

由于redis类库中没有提供对列表元素的增量迭代，如果想要循环name对应的列表的所有元素，那么就需要：

获取name对应的所有列表

循环列表

但是，如果列表非常大，那么就有可能在第一步时就将程序的内容撑爆，所有有必要自定义一个增量迭代的功能：

def list\_iter(name):

"""

自定义redis列表增量迭代

:param name: redis中的name，即：迭代name对应的列表

:return: yield 返回 列表元素

"""

list\_count = r.llen(name)

for index in range(list\_count):

yield r.lindex(name, index)

# 使用

for item in list\_iter('list2'): # 遍历这个列表

print(item)

## 集合set操作

### sadd

新增

sadd(name,values)

name对应的集合中添加元素

r.sadd("set1", 33, 44, 55, 66) # 往集合中添加元素

print(r.scard("set1")) # 集合的长度是4

print(r.smembers("set1")) # 获取集合中所有的成员

### scard

获取元素个数 类似于len

scard(name)

获取name对应的集合中元素个数

print(r.scard("set1")) # 集合的长度是4

### smembers

获取集合中所有的成员

smembers(name)

获取name对应的集合的所有成员

print(r.smembers("set1")) # 获取集合中所有的成员

获取集合中所有的成员–元组形式

sscan(name, cursor=0, match=None, count=None)

print(r.sscan("set1"))

1

获取集合中所有的成员–迭代器的方式

sscan\_iter(name, match=None, count=None)

同字符串的操作，用于增量迭代分批获取元素，避免内存消耗太大

for i in r.sscan\_iter("set1"):

print(i)2

### sdiff

差集

sdiff(keys, \*args)

在第一个name对应的集合中且不在其他name对应的集合的元素集合

r.sadd("set2", 11, 22, 33)

print(r.smembers("set1")) # 获取集合中所有的成员

print(r.smembers("set2"))

print(r.sdiff("set1", "set2")) # 在集合set1但是不在集合set2中

print(r.sdiff("set2", "set1")) # 在集合set2但是不在集合set1中

### sdiffstore

差集–差集存在一个新的集合中

sdiffstore(dest, keys, \*args)

获取第一个name对应的集合中且不在其他name对应的集合，再将其新加入到dest对应的集合中

r.sdiffstore("set3", "set1", "set2") # 在集合set1但是不在集合set2中

print(r.smembers("set3")) # 获取集合3中所有的成员

### sinter

交集

sinter(keys, \*args)

获取多一个name对应集合的交集

print(r.sinter("set1", "set2")) # 取2个集合的交集

### sinterstore

交集–交集存在一个新的集合中

sinterstore(dest, keys, \*args)

获取多一个name对应集合的并集，再将其加入到dest对应的集合中

print(r.sinterstore("set3", "set1", "set2")) # 取2个集合的交集

print(r.smembers("set3"))

### sunion

并集

sunion(keys, \*args)

获取多个name对应的集合的并集

print(r.sunion("set1", "set2")) # 取2个集合的并集1

并集–并集存在一个新的集合

sunionstore(dest,keys, \*args)

获取多一个name对应的集合的并集，并将结果保存到dest对应的集合中

print(r.sunionstore("set3", "set1", "set2")) # 取2个集合的并集

print(r.smembers("set3"))

### sismember

判断是否是集合的成员 类似in

sismember(name, value)

检查value是否是name对应的集合的成员，结果为True和False

print(r.sismember("set1", 33)) # 33是集合的成员

print(r.sismember("set1", 23)) # 23不是集合的成员

### smove

移动

smove(src, dst, value)

将某个成员从一个集合中移动到另外一个集合

r.smove("set1", "set2", 44)

print(r.smembers("set1"))

print(r.smembers("set2"))

### spop

删除–随机删除并且返回被删除值

spop(name)

从集合移除一个成员，并将其返回,说明一下，集合是无序的，所有是随机删除的

print(r.spop("set2")) # 这个删除的值是随机删除的，集合是无序的

print(r.smembers("set2"))

11.删除–指定值删除

srem(name, values)

在name对应的集合中删除某些值

print(r.srem("set2", 11)) # 从集合中删除指定值 11

print(r.smembers("set2"))

## 有序集合sort set操作

redis基本命令 有序set

Set操作，Set集合就是不允许重复的列表，本身是无序的

有序集合，在集合的基础上，为每元素排序；元素的排序需要根据另外一个值来进行比较，

所以，对于有序集合，每一个元素有两个值，即：值和分数，分数专门用来做排序。

### zadd

新增

zadd(name, args, \*kwargs)

在name对应的有序集合中添加元素

如：

import redis

import time

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, decode\_responses=True)

r = redis.Redis(connection\_pool=pool)

r.zadd("zset1", n1=11, n2=22)

r.zadd("zset2", 'm1', 22, 'm2', 44)

print(r.zcard("zset1")) # 集合长度

print(r.zcard("zset2")) # 集合长度

print(r.zrange("zset1", 0, -1)) # 获取有序集合中所有元素

print(r.zrange("zset2", 0, -1, withscores=True)) # 获取有序集合中所有元素和分数2

### zcard

获取有序集合元素个数 类似于len

zcard(name)

获取name对应的有序集合元素的数量

print(r.zcard("zset1")) # 集合长度1

### zrange

获取有序集合的所有元素

r.zrange( name, start, end, desc=False, withscores=False, score\_cast\_func=float)

按照索引范围获取name对应的有序集合的元素

参数：

name，redis的name

start，有序集合索引起始位置（非分数）

end，有序集合索引结束位置（非分数）

desc，排序规则，默认按照分数从小到大排序

withscores，是否获取元素的分数，默认只获取元素的值

score\_cast\_func，对分数进行数据转换的函数

3-1 从大到小排序(同zrange，集合是从大到小排序的)

zrevrange(name, start, end, withscores=False, score\_cast\_func=float)

print(r.zrevrange("zset1", 0, -1)) # 只获取元素，不显示分数

print(r.zrevrange("zset1", 0, -1, withscores=True)) # 获取有序集合中所有元素和分数,分数倒序

3-2 按照分数范围获取name对应的有序集合的元素

zrangebyscore(name, min, max, start=None, num=None, withscores=False, score\_cast\_func=float)

for i in range(1, 30):

element = 'n' + str(i)

r.zadd("zset3", element, i)

print(r.zrangebyscore("zset3", 15, 25)) # # 在分数是15-25之间，取出符合条件的元素

print(r.zrangebyscore("zset3", 12, 22, withscores=True)) # 在分数是12-22之间，取出符合条件的元素（带分数）

3-3 按照分数范围获取有序集合的元素并排序（默认从大到小排序）

zrevrangebyscore(name, max, min, start=None, num=None, withscores=False, score\_cast\_func=float)

print(r.zrevrangebyscore("zset3", 22, 11, withscores=True)) # 在分数是22-11之间，取出符合条件的元素 按照分数倒序1

3-4 获取所有元素–默认按照分数顺序排序

zscan(name, cursor=0, match=None, count=None, score\_cast\_func=float)

print(r.zscan("zset3"))1

3-5 获取所有元素–迭代器

zscan\_iter(name, match=None, count=None,score\_cast\_func=float)

for i in r.zscan\_iter("zset3"): # 遍历迭代器

print(i)

### zcount

zcount(name, min, max)

获取name对应的有序集合中分数 在 [min,max] 之间的个数

print(r.zrange("zset3", 0, -1, withscores=True))

print(r.zcount("zset3", 11, 22))

### zincrby

自增

zincrby(name, value, amount)

自增name对应的有序集合的 name 对应的分数

r.zincrby("zset3", "n2", amount=2) # 每次将n2的分数自增2

print(r.zrange("zset3", 0, -1, withscores=True))

### zrank

获取值的索引号

zrank(name, value)

获取某个值在 name对应的有序集合中的索引（从 0 开始）

更多：

zrevrank(name, value)，从大到小排序

print(r.zrank("zset3", "n1")) # n1的索引号是0 这里按照分数顺序（从小到大）

print(r.zrank("zset3", "n6")) # n6的索引号是1

print(r.zrevrank("zset3", "n1")) # n1的索引号是29 这里安照分数倒序（从大到小）

### zrem

删除–指定值删除

zrem(name, values)

删除name对应的有序集合中值是values的成员

r.zrem("zset3", "n3") # 删除有序集合中的元素n3 删除单个

print(r.zrange("zset3", 0, -1))

### zremrangebyrank

删除–根据排行范围删除，按照索引号来删除

zremrangebyrank(name, min, max)

根据排行范围删除

r.zremrangebyrank("zset3", 0, 1) # 删除有序集合中的索引号是0, 1的元素

print(r.zrange("zset3", 0, -1))

zremrangebyscore(name, min, max)

根据分数范围删除

r.zremrangebyscore("zset3", 11, 22) # 删除有序集合中的分数是11-22的元素

print(r.zrange("zset3", 0, -1))

### zscore

获取值对应的分数

zscore(name, value)

获取name对应有序集合中 value 对应的分数

print(r.zscore("zset3", "n27")) # 获取元素n27对应的分数271

## 其他常用操作

### delete

delete (\*names)

根据删除redis中的任意数据类型（string、hash、list、set、有序set）

r.delete("gender") # 删除key为gender的键值对1

### exists

检查名字是否存在

exists(name)

检测redis的name是否存在，存在就是True，False 不存在

print(r.exists("zset1"))1

### keys

模糊匹配

keys(pattern=’‘)

根据模型获取redis的name

更多：

KEYS 匹配数据库中所有 key 。

KEYS h?llo 匹配 hello ， hallo 和 hxllo 等。

KEYS h\*llo 匹配 hllo 和 heeeeello 等。

KEYS h[ae]llo 匹配 hello 和 hallo ，但不匹配 hillo

print(r.keys("foo\*"))

1

### expire

设置超时时间

expire(name ,time)

为某个redis的某个name设置超时时间

r.lpush("list5", 11, 22)

r.expire("list5", time=3)

print(r.lrange("list5", 0, -1))

time.sleep(3)

print(r.lrange("list5", 0, -1))

### rname

重命名

rename(src, dst)

对redis的name重命名

r.lpush("list5", 11, 22)

r.rename("list5", "list5-1")

### randomkey

随机获取name

randomkey()

随机获取一个redis的name（不删除）

print(r.randomkey())

1

### type

获取类型

type(name)

获取name对应值的类型

print(r.type("set1"))

print(r.type("hash2"))

### scan

查看所有元素

scan(cursor=0, match=None, count=None)

print(r.hscan("hash2"))

print(r.sscan("set3"))

print(r.zscan("zset2"))

print(r.getrange("foo1", 0, -1))

print(r.lrange("list2", 0, -1))

print(r.smembers("set3"))

print(r.zrange("zset3", 0, -1))

print(r.hgetall("hash1"))

### scan\_iter

查看所有元素–迭代器

scan\_iter(match=None, count=None)

for i in r.hscan\_iter("hash1"):

print(i)

for i in r.sscan\_iter("set3"):

print(i)

for i in r.zscan\_iter("zset3"):

print(i)8

other 方法

print(r.get('name')) # 查询key为name的值

r.delete("gender") # 删除key为gender的键值对

print(r.keys()) # 查询所有的Key

print(r.dbsize()) # 当前redis包含多少条数据

r.save() # 执行"检查点"操作，将数据写回磁盘。保存时阻塞

# r.flushdb() # 清空r中的所有数据

### 管道（pipeline）

redis默认在执行每次请求都会创建（连接池申请连接）和断开（归还连接池）一次连接操作，

如果想要在一次请求中指定多个命令，则可以使用pipline实现一次请求指定多个命令，并且默认情况下一次pipline 是原子性操作。

管道（pipeline）是redis在提供单个请求中缓冲多条服务器命令的基类的子类。它通过减少服务器-客户端之间反复的TCP数据库包，从而大大提高了执行批量命令的功能。

import redis

import time

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, decode\_responses=True)

r = redis.Redis(connection\_pool=pool)

# pipe = r.pipeline(transaction=False) # 默认的情况下，管道里执行的命令可以保证执行的原子性，执行pipe = r.pipeline(transaction=False)可以禁用这一特性。

# pipe = r.pipeline(transaction=True)

pipe = r.pipeline() # 创建一个管道

pipe.set('name', 'jack')

pipe.set('role', 'sb')

pipe.sadd('faz', 'baz')

pipe.incr('num') # 如果num不存在则vaule为1，如果存在，则value自增1

pipe.execute()

print(r.get("name"))

print(r.get("role"))

print(r.get("num"))

管道的命令可以写在一起，如：

pipe.set('hello', 'redis').sadd('faz', 'baz').incr('num').execute()

print(r.get("name"))

print(r.get("role"))

print(r.get("num"))