### 键（key）

##### 登录远程服务

redis-cli -h host -p port -a password

##### 测试服务器是否正常运行

Ping

##### DEL 命令

命令用于删除已存在的键（返回1）。不存在的 key 会被忽略（返回0）。

127.0.0.1:6379> del a

(integer) 1

127.0.0.1:6379> del a123

(integer) 0

##### DUMP 命令

用于序列化给定 key ，并返回被序列化的值。

127.0.0.1:6379> SET greeting "hello, dumping world!"

OK

127.0.0.1:6379> get greeting

"hello, dumping world!"

127.0.0.1:6379> dump greeting

"\x00\x15hello, dumping world!\a\x00,\x7f\xe7\xf1%\xed(W"

127.0.0.1:6379> get greeting

"hello, dumping world!"

127.0.0.1:6379> dump a1324

(nil)

##### EXISTS 命令

用于检查给定 key 是否存在。若 key 存在返回 1 ，否则返回 0 。

127.0.0.1:6379> set test1 abc

OK

127.0.0.1:6379> exists test1

(integer) 1

127.0.0.1:6379> exists test111test

(integer) 0

127.0.0.1:6379>

##### Expire 命令

用于设置 key 的过期时间，key 过期后将不再可用。单位以秒计。

127.0.0.1:6379> exists test1

(integer) 1

127.0.0.1:6379> expire test1 10

(integer) 1

127.0.0.1:6379> exists test1

(integer) 1

127.0.0.1:6379> exists test1

(integer) 0

##### Expireat 命令

用于以 UNIX 时间戳(unix timestamp)格式设置 key 的过期时间。key 过期后将不再可用。

EXPIREAT test 1293840000

##### PEXPIRE 命令

和 EXPIRE 命令的作用类似，但是它以毫秒为单位设置 key 的生存时间，而不像 EXPIRE 命令那样，以秒为单位。

##### PEXPIREAT 命令

用于设置 key 的过期时间，以毫秒计。key 过期后将不再可用。

PEXPIREAT runoobkey 1555555555005

Ps: 与EXPIREAT相差 1000倍

##### Keys 命令

用于查找所有符合给定模式 pattern 的 key 。

\*代表着匹配任意字符（有/没有），任意个数（0-无穷）

127.0.0.1:6379> keys test29\*

127.0.0.1:6379> keys test29\*1

##### MOVE 命令

用于将当前数据库的 key 移动到给定的数据库 db 当中。

## 选中数据库1，默认是0

select 1

**move test2506 0**

##### PERSIST 命令

用于移除给定 key 的过期时间，使得 key 永不过期。

**PERSIST mykey # 移除 key 的生存时间**

##### Pttl 命令

以毫秒为单位返回 key 的剩余过期时间。

返回值

当 key 不存在时，返回 -2 。 当 key 存在但没有设置剩余生存时间时，返回 -1 。 否则，以毫秒为单位，返回 key 的剩余生存时间。

127.0.0.1:6379> pttl test1111

(integer) -1

127.0.0.1:6379> pttl test00111

(integer) -2

127.0.0.1:6379> expire test1788 60

(integer) 1

127.0.0.1:6379> pttl test1788

(integer) 54126

127.0.0.1:6379> pttl test1788

(integer) 43355

##### TTL 命令

以秒为单位返回 key 的剩余过期时间。

同pttl

##### RANDOMKEY 命令

从当前数据库中随机返回一个 key 。

127.0.0.1:6379> randomkey

"test149"

##### Rename 命令

用于修改 key 的名称 。改名成功时提示 OK ，失败时候返回一个错误。

当 OLD\_KEY\_NAME 和 NEW\_KEY\_NAME 相同，或者 OLD\_KEY\_NAME 不存在时，返回一个错误。 当 NEW\_KEY\_NAME 已经存在时， RENAME 命令将覆盖旧值。

127.0.0.1:6379> rename test2975 test2976

OK

127.0.0.1:6379> exists test2975

(integer) 0

127.0.0.1:6379> exists test2976

(integer) 1

##### Renamenx 命令

Redis Renamenx 命令用于在新的 key 不存在时修改 key 的名称 。

##### Type 命令

用于返回 key 所储存的值的类型。

127.0.0.1:6379> type person

string

127.0.0.1:6379> type test2975

none

127.0.0.1:6379> type test2976

String

### 字符串(String)

##### SET 命令

用于设置给定 key 的值。如果 key 已经存储其他值， SET 就覆写旧值，且无视类型。

127.0.0.1:6379> set test1 1

OK

##### Get 命令

用于获取指定 key 的值。如果 key 不存在，返回 nil 。如果key 储存的值不是字符串类型，返回一个错误。

127.0.0.1:6379> get test1

"1"

127.0.0.1:6379> get test01

(nil)

##### Getrange 命令

用于获取存储在指定 key 中字符串的子字符串。字符串的截取范围由 start 和 end 两个偏移量决定(包括 start 和 end 在内)。

127.0.0.1:6379> set testKey "how old are you?"

OK

127.0.0.1:6379> getrange testKey 0 2

"how"

127.0.0.1:6379> getrange testKey 0 -1

"how old are you?"

127.0.0.1:6379> getrange testKey 4 6

"old"

127.0.0.1:6379> getrange testKey 4 -1

"old are you?"

##### Getset 命令

用于设置指定 key 的值，并返回 key 的旧值。

127.0.0.1:6379> getset test2506 123

"2506"

127.0.0.1:6379> get test2506

"123"

##### Mget 命令

返回所有(一个或多个)给定 key 的值。 如果给定的 key 里面，有某个 key 不存在，那么这个 key 返回特殊值 nil 。

127.0.0.1:6379> mget test888 test1075 test756

1) "888"

2) "1075"

3) "756"

##### Setex 命令

为指定的 key 设置值及其过期时间。如果 key 已经存在， SETEX 命令将会替换旧的值。

127.0.0.1:6379> setex my123 10 123

OK

127.0.0.1:6379> get my123

"123"

127.0.0.1:6379> ttl my123

(integer) 3

127.0.0.1:6379> get my123

(nil)

##### Setnx（SET if Not eXists） 命令

在指定的 key 不存在时，为 key 设置指定的值。

127.0.0.1:6379> setnx my123 123

(integer) 1

127.0.0.1:6379> setnx my123 123

(integer) 0

##### Setrange 命令

用指定的字符串覆盖给定 key 所储存的字符串值，覆盖的位置从偏移量 offset 开始。

localhost:6379> set myKey "hollo word 123"

OK

localhost:6379> get myKey

"hollo word 123"

localhost:6379> setrange myKey 6 redis

(integer) 14

localhost:6379> get myKey

"hollo redis123"

##### Strlen 命令

用于获取指定 key 所储存的字符串值的长度。当 key 储存的不是字符串值时，返回一个错误。

localhost:6379> get myKey

"hollo redis123"

localhost:6379> strlen myKey

(integer) 14

##### Mset 命令

用于同时设置一个或多个 key-value 对。

localhost:6379> mset test199 123 test1990 234

OK

localhost:6379> get test199

"123"

localhost:6379> get test1990

"234"

##### Msetnx 命令

用于所有给定 key 都不存在时，同时设置一个或多个 key-value 对。

同mset命令

##### Psetex 命令

以毫秒为单位设置 key 的生存时间。

同setex

##### Incr 命令

将 key 中储存的数字值增一。

如果 key 不存在，那么 key 的值会先被初始化为 0 ，然后再执行 INCR 操作。

如果值包含错误的类型，或字符串类型的值不能表示为数字，那么返回一个错误。

本操作的值限制在 64 位(bit)有符号数字表示之内。

localhost:6379> incr test1992

(integer) 1993

localhost:6379> get myKey

"hollo redis123"

localhost:6379> incr myKey

(error) ERR value is not an integer or out of range

##### Incrby 命令

将 key 中储存的数字加上指定的增量值。

如果 key 不存在，那么 key 的值会先被初始化为 0 ，然后再执行 INCRBY 命令。

如果值包含错误的类型，或字符串类型的值不能表示为数字，那么返回一个错误。

本操作的值限制在 64 位(bit)有符号数字表示之内。

localhost:6379> incrby test1992 20

(integer) 2013

##### Incrbyfloat 命令

为 key 中所储存的值加上指定的浮点数增量值。

如果 key 不存在，那么 INCRBYFLOAT 会先将 key 的值设为 0 ，再执行加法操作。

localhost:6379> incrbyfloat test1992 10.2

"2023.2"

##### Decr 命令

将 key 中储存的数字值减一。

如果 key 不存在，那么 key 的值会先被初始化为 0 ，然后再执行 DECR 操作。

如果值包含错误的类型，或字符串类型的值不能表示为数字，那么返回一个错误。

本操作的值限制在 64 位(bit)有符号数字表示之内。

##### Decrby 命令

将 key 所储存的值减去指定的减量值。

如果 key 不存在，那么 key 的值会先被初始化为 0 ，然后再执行 DECRBY 操作。

如果值包含错误的类型，或字符串类型的值不能表示为数字，那么返回一个错误。

本操作的值限制在 64 位(bit)有符号数字表示之内。

##### Append 命令

用于为指定的 key 追加值。

如果 key 已经存在并且是一个字符串， APPEND 命令将 value 追加到 key 原来的值的末尾。

如果 key 不存在， APPEND 就简单地将给定 key 设为 value ，就像执行 SET key value 一样。

localhost:6379> append myKey 123

(integer) 17

localhost:6379> get myKey

"hollo redis123123"

### 哈希(Hash)

##### 入门命令

localhost:6379> hset map id 123

(integer) 1

localhost:6379> hset map name zhangsan

(integer) 1

localhost:6379> hget map id

"123"

localhost:6379> hget map name

"zhangsan"

##### Hdel 命令

用于删除哈希表 key 中的一个或多个指定字段，不存在的字段将被忽略。

localhost:6379> hdel map id

(integer) 1

##### Hexists 命令

用于查看哈希表的指定字段是否存在。

localhost:6379> hexists map id

(integer) 0

localhost:6379> hexists map name

(integer) 1

##### Hget 命令

用于返回哈希表中指定字段的值。

localhost:6379> hget map id

(nil)

localhost:6379> hget map name

"zhangsan"

##### Hgetall 命令

用于返回哈希表中，所有的字段和值。

在返回值里，紧跟每个字段名(field name)之后是字段的值(value)，所以返回值的长度是哈希表大小的两倍。

localhost:6379> hset map age 24

(integer) 1

localhost:6379> hgetall map

1) "name"

2) "zhangsan"

3) "age"

4) "24"

##### Hincrby 命令

用于为哈希表中的字段值加上指定增量值。

增量也可以为负数，相当于对指定字段进行减法操作。

如果哈希表的 key 不存在，一个新的哈希表被创建并执行 HINCRBY 命令。

如果指定的字段不存在，那么在执行命令前，字段的值被初始化为 0 。

对一个储存字符串值的字段执行 HINCRBY 命令将造成一个错误。

本操作的值被限制在 64 位(bit)有符号数字表示之内。

localhost:6379> hincrby map id 1

(integer) 1

localhost:6379> hincrby map age 2

(integer) 26

localhost:6379> hgetall map

1) "name"

2) "zhangsan"

3) "age"

4) "26"

5) "id"

6) "1"

##### Hincrbyfloat 命令

用于为哈希表中的字段值加上指定浮点数增量值。

如果指定的字段不存在，那么在执行命令前，字段的值被初始化为 0 。

localhost:6379> hincrbyfloat map age -5.5

"20.5"

localhost:6379> hget map age

"20.5"

##### Hkeys 命令

用于获取哈希表中的所有域（field）。

localhost:6379> hkeys map

1) "name"

2) "age"

3) "id"

##### Hlen 命令

用于获取哈希表中字段的数量。

localhost:6379> hlen map

(integer) 3

##### Hmget 命令

用于返回哈希表中，一个或多个给定字段的值。

如果指定的字段不存在于哈希表，那么返回一个 nil 值。

localhost:6379> hmget map id name remark

1) "1"

2) "zhangsan"

3) (nil)

##### Hmset 命令

用于同时将多个 field-value (字段-值)对设置到哈希表中。

此命令会覆盖哈希表中已存在的字段。

如果哈希表不存在，会创建一个空哈希表，并执行 HMSET 操作。

localhost:6379> hmset map id 2 name wangfang

OK

localhost:6379> hgetall map

1) "name"

2) "wangfang"

3) "age"

4) "20.5"

5) "id"

6) "2"

##### Hset 命令

用于为哈希表中的字段赋值 。

如果哈希表不存在，一个新的哈希表被创建并进行 HSET 操作。

如果字段已经存在于哈希表中，旧值将被覆盖。

localhost:6379> hset map id 10089

(integer) 0

localhost:6379> hget map id

"10089"

##### Hsetnx 命令

用于为哈希表中不存在的的字段赋值 。

如果哈希表不存在，一个新的哈希表被创建并进行 HSET 操作。

如果字段已经存在于哈希表中，操作无效。

如果 key 不存在，一个新哈希表被创建并执行 HSETNX 命令。

##### Hvals 命令

返回哈希表所有域(field)的值。

localhost:6379> hvals map

1) "wangfang"

2) "20.5"

3) "10089"

### 列表(List)

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

一个列表最多可以包含 232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。

##### 入门

localhost:6379> lpush pop a b

(integer) 2

localhost:6379> lpush pop c

(integer) 3

localhost:6379> lrange pop

(error) ERR wrong number of arguments for 'lrange' command

localhost:6379> lrange pop 0 4

1) "c"

2) "b"

3) "a"

##### Blpop 命令

移出并获取列表的第一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。单位秒

localhost:6379> lrange pop 0 4

1) "c"

2) "b"

3) "a"

localhost:6379> blpop pop 20

1) "pop"

2) "c"

localhost:6379> lrange pop 0 4

1) "b"

2) "a"

##### Brpop 命令

移出并获取列表的最后一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。

localhost:6379> lrange pop 0 10

1) "c"

2) "b"

3) "a"

localhost:6379> brpop list 10

(nil)

(10.01s)

localhost:6379> brpop pop 10

1) "pop"

2) "a"

localhost:6379> lrange pop 0 10

1) "c"

2) "b"

##### Brpoplpush 命令

从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。

localhost:6379> lrange pop 0 4

1) "4"

2) "3"

3) "2"

4) "1"

localhost:6379> brpoplpush pop newpop 1

"1"

localhost:6379> lrange pop 0 4

1) "4"

2) "3"

3) "2"

localhost:6379> lrange newpop 0 4

1. "1"

##### Lindex 命令

用于通过索引获取列表中的元素。你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。

localhost:6379> lrange pop 0 4

1) "4"

2) "3"

3) "2"

4) "1"

localhost:6379> lindex pop 0

"4"

localhost:6379> lindex pop 3

"1"

localhost:6379> lindex pop -1

"1"

localhost:6379> lindex pop -2

"2"

##### Linsert 命令

用于在列表的元素前或者后插入元素。当指定元素不存在于列表中时，不执行任何操作。

当列表不存在时，被视为空列表，不执行任何操作。

如果 key 不是列表类型，返回一个错误。

localhost:6379> lrange pop 0 5

1) "4"

2) "3"

3) "2"

4) "1"

localhost:6379> linsert pop AFTER 4 5

(integer) 5

localhost:6379> lrange pop 0 10

1) "4"

2) "5"

3) "3"

4) "2"

5) "1"

localhost:6379> linsert pop before 1 1.5

(integer) 6

localhost:6379> lrange pop 0 10

1) "4"

2) "5"

3) "3"

4) "2"

5) "1.5"

6) "1"

##### Llen 命令

用于返回列表的长度。 如果列表 key 不存在，则 key 被解释为一个空列表，返回 0 。 如果 key 不是列表类型，返回一个错误。

localhost:6379> llen pop

(integer) 6

##### Lpop 命令

用于移除并返回列表的第一个元素。

localhost:6379> lrange pop 0 10

1) "4"

2) "5"

3) "3"

4) "2"

5) "1.5"

6) "1"

localhost:6379> lpop pop

"4"

##### Lpush 命令

将一个或多个值插入到列表头部。 如果 key 不存在，一个空列表会被创建并执行 LPUSH 操作。 当 key 存在但不是列表类型时，返回一个错误。

localhost:6379> lpush pop 8 9

(integer) 7

localhost:6379> lrange pop 0 7

1) "9"

2) "8"

3) "5"

4) "3"

5) "2"

6) "1.5"

7) "1"

##### Lpushx 命令

将一个值插入到已存在的列表头部，列表不存在时操作无效。

localhost:6379> lpushx pop2 1

(integer) 0

localhost:6379> lpushx pop 1

(integer) 8

##### Lrange命令

返回列表中指定区间内的元素，区间以偏移量 START 和 END 指定。 其中 0 表示列表的第一个元素， 1 表示列表的第二个元素，以此类推。 你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。

localhost:6379> lrange pop 0 -1

1) "1"

2) "9"

3) "8"

4) "5"

5) "3"

6) "2"

7) "1.5"

8) "1"

##### Lrem 命令

根据参数 COUNT 的值，移除列表中与参数 VALUE 相等的元素。

COUNT 的值可以是以下几种：

* count > 0 : 从表头开始向表尾搜索，移除与 VALUE 相等的元素，数量为 COUNT 。
* count < 0 : 从表尾开始向表头搜索，移除与 VALUE 相等的元素，数量为 COUNT 的绝对值。
* count = 0 : 移除表中所有与 VALUE 相等的值

localhost:6379> lrange pop 0 -1

1) "1"

2) "9"

3) "8"

4) "5"

5) "3"

6) "2"

7) "1.5"

8) "1"

localhost:6379> lrem pop -2 1

(integer) 2

localhost:6379> lrange pop 0 -1

1) "9"

2) "8"

3) "5"

4) "3"

5) "2"

6) "1.5"

localhost:6379> lrem pop 0 1.5

(integer) 1

##### Lset

通过索引来设置元素的值。

当索引参数超出范围，或对一个空列表进行 LSET 时，返回一个错误。

localhost:6379> lrange pop 0 -1

1) "9"

2) "8"

3) "5"

4) "3"

5) "2"

localhost:6379> lset pop 4 1

OK

localhost:6379> lrange pop 0 -1

1) "9"

2) "8"

3) "5"

4) "3"

5) "1"

##### Ltrim

对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。

下标 0 表示列表的第一个元素，以 1 表示列表的第二个元素，以此类推。 你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。

localhost:6379> lrange pop 0 -1

1) "9"

2) "8"

3) "5"

4) "3"

5) "1"

localhost:6379> ltrim pop 1 -2

OK

localhost:6379> lrange pop 0 -1

1) "8"

2) "5"

3) "3"

##### Rpop 命令

用于移除列表的最后一个元素，返回值为移除的元素。

localhost:6379> lrange pop 0 -1

1) "8"

2) "5"

3) "3"

localhost:6379> rpop pop

"3"

localhost:6379> lrange pop 0 -1

1) "8"

2) "5"

##### Rpoplpush 命令

用于移除列表的最后一个元素，并将该元素添加到另一个列表并返回。

localhost:6379> lrange pop 0 -1

1) "8"

2) "5"

localhost:6379> lrange pop2 0 -1

(empty list or set)

localhost:6379> rpoplpush pop pop2

"5"

localhost:6379> lrange pop2 0 -1

1) "5"

localhost:6379> lrange pop 0 -1

1) "8"

##### Rpush 命令

用于将一个或多个值插入到列表的尾部(最右边)。

如果列表不存在，一个空列表会被创建并执行 RPUSH 操作。 当列表存在但不是列表类型时，返回一个错误。

localhost:6379> lrange pop 0 -1

1) "8"

localhost:6379> rpush pop 9

(integer) 2

localhost:6379> rpush pop 10

(integer) 3

localhost:6379> lrange pop 0 -1

1) "8"

2) "9"

3) "10"

##### Rpushx 命令

用于将一个值插入到已存在的列表尾部(最右边)。如果列表不存在，操作无效。

localhost:6379> lrange pop 0 -1

1) "8"

2) "9"

3) "10"

localhost:6379> rpushx pop 11

(integer) 4

localhost:6379> rpushx pop22 2

(integer) 0

localhost:6379> lrange pop 0 -1

1) "8"

2) "9"

3) "10"

4) "11"

### 集合(Set)

Redis 的 Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。

Redis 中集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。

集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

##### Sadd 命令

将一个或多个成员元素加入到集合中，已经存在于集合的成员元素将被忽略。

假如集合 key 不存在，则创建一个只包含添加的元素作成员的集合。

当集合 key 不是集合类型时，返回一个错误。

localhost:6379> sadd set 1 2 3 3

(integer) 3

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

##### Scard 命令

返回集合中元素的数量。

localhost:6379> scard set

(integer) 3

##### Sdiff 命令

返回给定集合之间的差集。不存在的集合 key 将视为空集。

差集的结果来自前面的 FIRST\_KEY ,而不是后面的 OTHER\_KEY1，也不是整个 FIRST\_KEY OTHER\_KEY1..OTHER\_KEYN 的差集。

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

localhost:6379> scard set

(integer) 3

localhost:6379> sadd set2 2 3 4

(integer) 3

localhost:6379> sdiff set set2

1) "1"

localhost:6379> sdiff set2 set

1. "4"

localhost:6379> sadd set3 2 3

(integer) 2

localhost:6379> sdiff set3 set

(empty list or set)

localhost:6379> sdiff set set3

1) "1"

##### Sdiffstore 命令

将给定集合之间的差集存储在指定的集合中。如果指定的集合 key 已存在，则会被覆盖。

localhost:6379> sdiffstore setstore set set3

(integer) 1

localhost:6379> smembers setstore

1. "1"

##### Sinter 命令

返回给定所有给定集合的交集。 不存在的集合 key 被视为空集。 当给定集合当中有一个空集时，结果也为空集(根据集合运算定律)。

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

localhost:6379> smembers set2

1) "2"

2) "3"

3) "4"

localhost:6379> sinter set set2

1) "2"

2) "3"

##### Sinterstore 命令

将给定集合之间的交集存储在指定的集合中。如果指定的集合已经存在，则将其覆盖。

localhost:6379> sinterstore setstore set set2

(integer) 2

localhost:6379> smembers setstore

1) "2"

2) "3"

##### Sismember 命令

判断成员元素是否是集合的成员。

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

localhost:6379> sismember set 1

(integer) 1

localhost:6379> sismember set 4

(integer) 0

##### Smembers 命令

返回集合中的所有的成员。 不存在的集合 key 被视为空集合。

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

##### Smove 命令

将指定成员 member 元素从 source 集合移动到 destination 集合。

SMOVE 是原子性操作。

如果 source 集合不存在或不包含指定的 member 元素，则 SMOVE 命令不执行任何操作，仅返回 0 。否则， member 元素从 source 集合中被移除，并添加到 destination 集合中去。

当 destination 集合已经包含 member 元素时， SMOVE 命令只是简单地将 source 集合中的 member 元素删除。

当 source 或 destination 不是集合类型时，返回一个错误。

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

localhost:6379> smembers set2

1) "2"

2) "3"

3) "4"

localhost:6379> smove set2 set 4

(integer) 1

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

4) "4"

localhost:6379> smembers set2

1) "2"

2) "3"

##### Spop 命令

用于移除集合中的指定 key 的一个或多个随机元素，移除后会返回移除的元素。

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

4) "4"

localhost:6379> smembers set2

1) "2"

2) "3"

localhost:6379> spop set 2

1) "3"

2) "2"

localhost:6379> smembers set

1) "1"

2) "4"

##### Srandmember 命令

用于返回集合中的一个随机元素。

从 Redis 2.6 版本开始， Srandmember 命令接受可选的 count 参数：

* 如果 count 为正数，且小于集合基数，那么命令返回一个包含 count 个元素的数组，数组中的元素各不相同。如果 count 大于等于集合基数，那么返回整个集合。
* 如果 count 为负数，那么命令返回一个数组，数组中的元素可能会重复出现多次，而数组的长度为 count 的绝对值。

该操作和 SPOP 相似，但 SPOP 将随机元素从集合中移除并返回，而 Srandmember 则仅仅返回随机元素，而不对集合进行任何改动。

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

4) "4"

localhost:6379> srandmember set

"4"

localhost:6379> srandmember set 2

1) "1"

2) "4"

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

4) "4"

##### Srem 命令

用于移除集合中的一个或多个成员元素，不存在的成员元素会被忽略。

当 key 不是集合类型，返回一个错误。

在 Redis 2.4 版本以前， SREM 只接受单个成员值。

localhost:6379> smembers set

1) "1"

2) "2"

3) "3"

4) "4"

localhost:6379> srem set 1 3

(integer) 2

localhost:6379> smembers set

1) "2"

2) "4"

##### Sunion 命令

返回给定集合的并集。不存在的集合 key 被视为空集。

localhost:6379> smembers set

1) "2"

2) "4"

localhost:6379> smembers set2

1) "2"

2) "3"

localhost:6379> sunion set set2

1) "2"

2) "3"

3) "4"

##### Sunionstore 命令

将给定集合的并集存储在指定的集合 destination 中。如果 destination 已经存在，则将其覆盖。

##### Sscan 命令

用于迭代集合中键的元素。

### 有序集合(sorted set)

##### 入门

localhost:6379> zadd test 1 redis

(integer) 1

localhost:6379> zadd test 2 mysql

(integer) 1

localhost:6379> zadd test 3 oracle

(integer) 1

localhost:6379> zadd test 4 mysql

(integer) 0

localhost:6379> zrange test 0 10

1) "redis"

2) "oracle"

3) "mysql"

localhost:6379> zrange test 0 10 withscores

1) "redis"

2) "1"

3) "oracle"

4) "3"

5) "mysql"

6) "4"

##### Zadd 命令

用于将一个或多个成员元素及其分数值加入到有序集当中。

如果某个成员已经是有序集的成员，那么更新这个成员的分数值，并通过重新插入这个成员元素，来保证该成员在正确的位置上。

分数值可以是整数值或双精度浮点数。

如果有序集合 key 不存在，则创建一个空的有序集并执行 ZADD 操作。

当 key 存在但不是有序集类型时，返回一个错误。

localhost:6379> zadd test 2 java

(integer) 1

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "oracle"

6) "3"

7) "mysql"

8) "4"

##### Zcard 命令

用于计算集合中元素的数量。

localhost:6379> zcard test

(integer) 4

##### Zcount 命令

用于计算有序集合中指定分数区间的成员数量。

localhost:6379> zcount test 1 3

(integer) 3

localhost:6379> zcount test 2 3

(integer) 2

##### Zincrby 命令

对有序集合中指定成员的分数加上增量 increment

可以通过传递一个负数值 increment ，让分数减去相应的值，比如 ZINCRBY key -5 member ，就是让 member 的 score 值减去 5 。

当 key 不存在，或分数不是 key 的成员时， ZINCRBY key increment member 等同于 ZADD key increment member 。

当 key 不是有序集类型时，返回一个错误。

分数值可以是整数值或双精度浮点数。

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "oracle"

6) "3"

7) "mysql"

8) "4"

localhost:6379> zincrby test 2 oracle

"5"

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "mysql"

6) "4"

7) "oracle"

8) "5"

##### Zinterstore 命令

计算给定的一个或多个有序集的交集，其中给定 key 的数量必须以 numkeys 参数指定，并将该交集(结果集)储存到 destination 。

默认情况下，结果集中某个成员的分数值是所有给定集下该成员分数值之和。

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "mysql"

6) "4"

7) "oracle"

8) "5"

localhost:6379> zrange test2 0 -1 withscores

1) "java"

2) "1"

3) "mysql"

4) "1"

5) "redis"

6) "1"

localhost:6379> zinterstore teststore 2 test test2

(integer) 3

localhost:6379> zrange teststore 0 -1 withscores

1) "redis"

2) "2"

3) "java"

4) "3"

5) "mysql"

6) "5"

##### Zlexcount 命令

在计算有序集合中指定字典区间内成员数量。

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "mysql"

6) "4"

7) "oracle"

8) "5"

localhost:6379> zlexcount test - +

(integer) 4

##### Zrange

返回有序集中，指定区间内的成员。

其中成员的位置按分数值递增(从小到大)来排序。

具有相同分数值的成员按字典序(lexicographical order )来排列。

如果你需要成员按

值递减(从大到小)来排列，请使用 [ZREVRANGE](http://www.runoob.com/redis/sorted-sets-zrevrange.html) 命令。

下标参数 start 和 - 都以 0 为底，也就是说，以 0 表示有序集第一个成员，以 1 表示有序集第二个成员，以此类推。

你也可以使用负数下标，以 -1 表示最后一个成员， -2 表示倒数第二个成员，以此类推。

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "mysql"

6) "4"

7) "oracle"

8) "5"

##### Zrangebylex

通过字典区间返回有序集合的成员。

localhost:6379> zrange myzset 0 -1 withscores

1) "a"

2) "0"

3) "b"

4) "0"

5) "c"

6) "0"

7) "d"

8) "0"

9) "e"

10) "0"

11) "f"

12) "0"

13) "g"

14) "0"

localhost:6379> zrangebylex myzset 0 -1 - [c

(error) ERR min or max not valid string range item

localhost:6379> zrangebylex myzset - [c

1) "a"

2) "b"

3) "c"

localhost:6379> zrangebylex myzset - (c

1) "a"

2) "b"

localhost:6379> zrangebylex myzset [c (g

1) "c"

2) "d"

3) "e"

4) "f"

localhost:6379> zadd myzset 1 h

(integer) 1

localhost:6379> zrangebylex myzset [e [h

1) "e"

2) "f"

3) "g"

4) "h"

localhost:6379>

##### Zrangebyscore

返回有序集合中指定分数区间的成员列表。有序集成员按分数值递增(从小到大)次序排列。

具有相同分数值的成员按字典序来排列(该属性是有序集提供的，不需要额外的计算)。

默认情况下，区间的取值使用闭区间 (小于等于或大于等于)，你也可以通过给参数前增加 ( 符号来使用可选的开区间 (小于或大于)。

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "mysql"

6) "4"

7) "oracle"

8) "5"

localhost:6379> zrangebyscore test (2 5

1) "mysql"

2) "oracle"

localhost:6379> zrangebyscore test (2 (5

1) "mysql"

##### Zrank

返回有序集中指定成员的排名。其中有序集成员按分数值递增(从小到大)顺序排列。

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "mysql"

6) "4"

7) "oracle"

8) "5"

localhost:6379> zrank test java

(integer) 1

localhost:6379> zrank test oracle

(integer) 3

##### Zrem 命令

用于移除有序集中的一个或多个成员，不存在的成员将被忽略。

当 key 存在但不是有序集类型时，返回一个错误。

localhost:6379> zrange myzset 0 -1 withscores

1) "a"

2) "0"

3) "b"

4) "0"

5) "c"

6) "0"

7) "d"

8) "0"

9) "e"

10) "0"

11) "f"

12) "0"

13) "g"

14) "0"

15) "h"

16) "1"

localhost:6379> zrem myzset a e

(integer) 2

localhost:6379> zrange myzset 0 -1 withscores

1) "b"

2) "0"

3) "c"

4) "0"

5) "d"

6) "0"

7) "f"

8) "0"

9) "g"

10) "0"

11) "h"

12) "1"

##### Zremrangebylex 命令

用于移除有序集合中给定的字典区间的所有成员。

localhost:6379> zrange myzset 0 -1

1) "b"

2) "c"

3) "d"

4) "f"

5) "g"

6) "h"

localhost:6379> zremrangebylex myzset [b [f

(integer) 4

localhost:6379> zrange myzset 0 -1

1) "g"

2) "h"

##### Zremrangebyrank 命令

用于移除有序集中，指定排名(rank)区间内的所有成员。

localhost:6379> zrange myzset 0 -1

1) "a"

2) "b"

3) "c"

4) "g"

5) "h"

localhost:6379> zrank myzset b

(integer) 1

localhost:6379> zrank myzset g

(integer) 3

localhost:6379> zremrangebyrank myzset 1 3

(integer) 3

localhost:6379> zrange myzset 0 -1

1) "a"

2) "h"

##### Zremrangebyscore 命令

用于移除有序集中，指定分数（score）区间内的所有成员。

localhost:6379> zrange myzset 0 -1

1) "a"

2) "h"

localhost:6379> zremrangebyscore myzset 0 0

(integer) 1

localhost:6379> zrange myzset 0 -1

1. "h"

##### Zrevrange 命令

返回有序集中，指定区间内的成员。

其中成员的位置按分数值递减(从大到小)来排列。

具有相同分数值的成员按字典序的逆序(reverse lexicographical order)排列。

除了成员按分数值递减的次序排列这一点外， ZREVRANGE 命令的其他方面和 [ZRANGE](http://www.runoob.com/redis/sorted-sets-zrange.html) 命令一样。

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "mysql"

6) "4"

7) "oracle"

8) "5"

localhost:6379> zrevrange test 0 -1 withscores

1) "oracle"

2) "5"

3) "mysql"

4) "4"

5) "java"

6) "2"

7) "redis"

8) "1"

##### Zrevrangebyscore

返回有序集中指定分数区间内的所有的成员。有序集成员按分数值递减(从大到小)的次序排列。

具有相同分数值的成员按字典序的逆序(reverse lexicographical order )排列。

localhost:6379> zrevrangebyscore test 4 2 withscores

1) "mysql"

2) "4"

3) "java"

4) "2"

##### Zrevrank 命令

返回有序集中成员的排名。其中有序集成员按分数值递减(从大到小)排序。

排名以 0 为底，也就是说， 分数值最大的成员排名为 0 。

使用 ZRANK 命令可以获得成员按分数值递增(从小到大)排列的排名。

localhost:6379> zrevrange test 0 -1 withscores

1) "oracle"

2) "5"

3) "mysql"

4) "4"

5) "java"

6) "2"

7) "redis"

8) "1"

localhost:6379> zrevrank test mysql

(integer) 1

##### Zscore 命令

返回有序集中，成员的分数值。 如果成员元素不是有序集 key 的成员，或 key 不存在，返回 nil 。

localhost:6379> zrange test 0 -1 withscores

1) "redis"

2) "1"

3) "java"

4) "2"

5) "mysql"

6) "4"

7) "oracle"

8) "5"

localhost:6379> zscore test mysql

"4"

##### Zunionstore 命令

计算给定的一个或多个有序集的并集，其中给定 key 的数量必须以 numkeys 参数指定，并将该并集(结果集)储存到 destination 。

默认情况下，结果集中某个成员的分数值是所有给定集下该成员分数值之和 。

localhost:6379> zrange programmer 0 -1 WITHSCORES

1) "peter"

2) "2000"

3) "jack"

4) "3500"

5) "tom"

6) "5000"

localhost:6379> zrange manager 0 -1 WITHSCORES

1) "peter"

2) "1500"

3) "herry"

4) "2000"

5) "mary"

6) "3500"

7) "bob"

8) "4000"

localhost:6379> zunionstore salary 2 programmer manager weights 0 2

(integer) 6

localhost:6379> ZRANGE salary 0 -1 WITHSCORES

1) "jack"

2) "0"

3) "tom"

4) "0"

5) "peter"

6) "3000"

7) "herry"

8) "4000"

9) "mary"

10) "7000"

11) "bob"

12) "8000"

### HyperLogLog

##### 入门

Redis HyperLogLog 是用来做基数统计的算法，HyperLogLog 的优点是，在输入元素的数量或者体积非常非常大时，计算基数所需的空间总是固定 的、并且是很小的。

在 Redis 里面，每个 HyperLogLog 键只需要花费 12 KB 内存，就可以计算接近 2^64 个不同元素的基 数。这和计算基数时，元素越多耗费内存就越多的集合形成鲜明对比。

但是，因为 HyperLogLog 只会根据输入元素来计算基数，而不会储存输入元素本身，所以 HyperLogLog 不能像集合那样，返回输入的各个元素。

比如数据集 {1, 3, 5, 7, 5, 7, 8}， 那么这个数据集的基数集为 {1, 3, 5 ,7, 8}, 基数(不重复元素)为5。 基数估计就是在误差可接受的范围内，快速计算基数。

localhost:6379> pfadd ptest redis oracle mysql mongodb

(integer) 1

localhost:6379> pfcount ptest

(integer) 4

localhost:6379>

##### PFMERGE 命令

将多个 HyperLogLog 合并为一个 HyperLogLog ，合并后的 HyperLogLog 的基数估算值是通过对所有 给定 HyperLogLog 进行并集计算得出的。

localhost:6379> PFADD hll1 foo bar zap a

(integer) 1

localhost:6379> PFADD hll2 a b c foo

(integer) 1

localhost:6379> PFMERGE hll3 hll1 hll2

OK

localhost:6379> PFCOUNT hll3

(integer) 6

### Redis 事务

Redis 事务可以一次执行多个命令， 并且带有以下两个重要的保证：

* 批量操作在发送 EXEC 命令前被放入队列缓存。
* 收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行。
* 在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。

一个事务从开始到执行会经历以下三个阶段：

* 开始事务。
* 命令入队。
* 执行事务。

单个 Redis 命令的执行是原子性的，但 Redis 没有在事务上增加任何维持原子性的机制，所以 Redis 事务的执行并不是原子性的。

事务可以理解为一个打包的批量执行脚本，但批量指令并非原子化的操作，中间某条指令的失败不会导致前面已做指令的回滚，也不会造成后续的指令不做。

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> set name "mmmmmm2222"

QUEUED

127.0.0.1:6379> get name

QUEUED

127.0.0.1:6379> sadd settag "ab" "cd"

QUEUED

127.0.0.1:6379> smembers settag

QUEUED

127.0.0.1:6379> exec

1) OK

2) "mmmmmm2222"

3) (integer) 2

4) 1) "cd"

2) "ab"

##### Discard 命令

用于取消事务，放弃执行事务块内的所有命令。

127.0.0.1:6379> multi

OK

127.0.0.1:6379> ping

QUEUED

127.0.0.1:6379> set abc abc

QUEUED

127.0.0.1:6379> discard

OK

##### Exec 命令

用于执行所有事务块内的命令。

127.0.0.1:6379> multi

OK

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> ping

QUEUED

127.0.0.1:6379> exec

1) (integer) 1

2) (integer) 2

3) (integer) 3

4) PONG

##### Watch 命令

用于监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断

**正常**

127.0.0.1:6379> watch abc12

OK

127.0.0.1:6379> multi

OK

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> ping

QUEUED

127.0.0.1:6379> exec

1) (integer) 4

2) (integer) 5

3) (integer) 6

4) PONG

**失败**

**开辟一个新的客户端，在watch及开启事务后，修改值**

**C:\tools\Redis-x64-3.2.100>redis-cli.exe**

**127.0.0.1:6379> set abc12 2**

**OK**

**127.0.0.1:6379>**

127.0.0.1:6379> watch abc12

OK

127.0.0.1:6379> incr abc12

(integer) 3

127.0.0.1:6379> multi

OK

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> incr abc12

QUEUED

127.0.0.1:6379> exec

(nil)

127.0.0.1:6379> get abc12

"2"

##### Unwatch 命令

用于取消 WATCH 命令对所有 key 的监视。

127.0.0.1:6379> unwatch

OK

### Redis 脚本

使用 Lua 解释器来执行脚本。 Redis 2.6 版本通过内嵌支持 Lua 环境。执行脚本的常用命令为 **EVAL**

##### Script Flush 命令

用于清除所有 Lua 脚本缓存。

C:\tools\Redis-x64-3.2.100>redis-cli.exe

127.0.0.1:6379> SCRIPT FLUSH

OK

##### Eval 命令

使用 Lua 解释器执行脚本。

127.0.0.1:6379> eval "return 'hello redis'" 0

"hello redis"

127.0.0.1:6379> eval "return {KEYS[1], ARGV[1]}" 1 a aa

1) "a"

2) "aa"

##### Redis Evalsha 命令

根据给定的 sha1 校验码，执行缓存在服务器中的脚本。

将脚本缓存到服务器的操作可以通过 SCRIPT LOAD 命令进行。

这个命令的其他地方，比如参数的传入方式，都和 EVAL 命令一样。

127.0.0.1:6379> eval "redis.call('set', KEYS[1], ARGV[1])" 1 a 12341432

(nil)

127.0.0.1:6379> get a

"12341432"

127.0.0.1:6379> SCRIPT LOAD "redis.call('set', KEYS[1], ARGV[1])"

"27d4ba7a9326bdcdabb271c0771037674735bec2"

127.0.0.1:6379> EVALSHA 27d4ba7a9326bdcdabb271c0771037674735bec2 1 bb 79

(nil)

127.0.0.1:6379> get bb

"79"

### 参数说明

redis.conf 配置项说明如下：

1. Redis默认不是以守护进程的方式运行，可以通过该配置项修改，使用yes启用守护进程

**daemonize no**

2. 当Redis以守护进程方式运行时，Redis默认会把pid写入/var/run/redis.pid文件，可以通过pidfile指定

**pidfile /var/run/redis.pid**

3. 指定Redis监听端口，默认端口为6379，作者在自己的一篇博文中解释了为什么选用6379作为默认端口，因为6379在手机按键上MERZ对应的号码，而MERZ取自意大利歌女Alessia Merz的名字

**port 6379**

4. 绑定的主机地址

**bind 127.0.0.1**

5.当 客户端闲置多长时间后关闭连接，如果指定为0，表示关闭该功能

**timeout 300**

6. 指定日志记录级别，Redis总共支持四个级别：debug、verbose、notice、warning，默认为verbose

**loglevel verbose**

7. 日志记录方式，默认为标准输出，如果配置Redis为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给/dev/null

**logfile stdout**

8. 设置数据库的数量，默认数据库为0，可以使用SELECT <dbid>命令在连接上指定数据库id

**databases 16**

9. 指定在多长时间内，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合

**save <seconds> <changes>**

    Redis默认配置文件中提供了三个条件：

**save 900 1**

**save 300 10**

**save 60 10000**

分别表示900秒（15分钟）内有1个更改，300秒（5分钟）内有10个更改以及60秒内有10000个更改。

当900秒执行1个写命令时，启动快照备份

当**300**秒执行10个写命令时，启动快照备份

当60秒执行10000个写命令时，启动快照备份

10. 指定存储至本地数据库时是否压缩数据，默认为yes，Redis采用LZF压缩，如果为了节省CPU时间，可以关闭该选项，但会导致数据库文件变的巨大

    ch**compression yes**

11. 指定本地数据库文件名，默认值为dump.rdb

**dbfilename dump.rdb**

12. 指定本地数据库存放目录

**dir ./**

13. 设置当本机为slav服务时，设置master服务的IP地址及端口，在Redis启动时，它会自动从master进行数据同步

**slaveof <masterip> <masterport>**

不再接受主服务同步，向从主机执行命令slaveof no one

14. 当master服务设置了密码保护时，slav服务连接master的密码

**masterauth <master-password>**

15. 设置Redis连接密码，如果配置了连接密码，客户端在连接Redis时需要通过AUTH <password>命令提供密码，默认关闭

**requirepass foobared**

16. 设置同一时间最大客户端连接数，默认无限制，Redis可以同时打开的客户端连接数为Redis进程可以打开的最大文件描述符数，如果设置 maxclients 0，表示不作限制。当客户端连接数到达限制时，Redis会关闭新的连接并向客户端返回max number of clients reached错误信息

**maxclients 128**

17. 指定Redis最大内存限制，Redis在启动时会把数据加载到内存中，达到最大内存后，Redis会先尝试清除已到期或即将到期的Key，当此方法处理 后，仍然到达最大内存设置，将无法再进行写入操作，但仍然可以进行读取操作。Redis新的vm机制，会把Key存放内存，Value会存放在swap区

**maxmemory <bytes>**

18. 指定是否在每次更新操作后进行日志记录，Redis在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为 redis本身同步数据文件是按上面save条件来同步的，所以有的数据会在一段时间内只存在于内存中。默认为no

**appendonly no（AOF方式）**

19. 指定更新日志文件名，默认为appendonly.aof

**appendfilename**

**appendonly.aof**

20. 指定更新日志条件，共有3个可选值：   
    **no**：表示等操作系统进行数据缓存同步到磁盘（快）   
    **always**：表示每次更新操作后手动调用fsync()将数据写到磁盘（慢，安全）   
    **everysec**：表示每秒同步一次（折中，默认值）

**appendfsync everysec**

21. 指定是否启用虚拟内存机制，默认值为no，简单的介绍一下，VM机制将数据分页存放，由Redis将访问量较少的页即冷数据swap到磁盘上，访问多的页面由磁盘自动换出到内存中（在后面的文章我会仔细分析Redis的VM机制）

**vm-enabled no**

22. 虚拟内存文件路径，默认值为/tmp/redis.swap，不可多个Redis实例共享

**vm-swap-file /tmp/redis.swap**

23. 将所有大于vm-max-memory的数据存入虚拟内存,无论vm-max-memory设置多小,所有索引数据都是内存存储的(Redis的索引数据 就是keys),也就是说,当vm-max-memory设置为0的时候,其实是所有value都存在于磁盘。默认值为0

**vm-max-memory 0**

24. Redis swap文件分成了很多的page，一个对象可以保存在多个page上面，但一个page上不能被多个对象共享，vm-page-size是要根据存储的 数据大小来设定的，作者建议如果存储很多小对象，page大小最好设置为32或者64bytes；如果存储很大大对象，则可以使用更大的page，如果不 确定，就使用默认值

**vm-page-size 32**

25. 设置swap文件中的page数量，由于页表（一种表示页面空闲或使用的bitmap）是在放在内存中的，，在磁盘上每8个pages将消耗1byte的内存。

**vm-pages 134217728**

26. 设置访问swap文件的线程数,最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的，可能会造成比较长时间的延迟。默认值为4

**vm-max-threads 4**

27. 设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启

**glueoutputbuf yes**

28. 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法

**hash-max-zipmap-entries 64**

**hash-max-zipmap-value 512**

29. 指定是否激活重置哈希，默认为开启（后面在介绍Redis的哈希算法时具体介绍）

**activerehashing yes**

30. 指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件

**include /path/to/local.conf**

redis执行save命令的时候，将禁止写入命令

stop-writes-on-bgsave-error yes

### 参数使用