# Redis必备命令简介

Redis必备命令包括Redis的键命令、HyperLogLog命令、脚本命令、连接命令、服务器命令等。

# 键命令

1. Redis的键命令主要用于管理Redis的键，如删除键、查询键、修改键以及设置键等。
2. Redis的键不仅仅指的是String类型中键值对的键，而哈希表名、列表名、集合名、有序集合名也都是键。

## 查询键

### exists

exist命令用于判断键是否存在于当前数据库，命令格式如下：

**exists key**

key表示键；

返回值：键存在，则返回1；键不存在，则返回0。

### keys

keys命令用于根据给定的匹配模式（类似于正则表达式）列出该数据库中所有的键，命令如下：

**keys pattern**

pattern表示匹配模式；

如：

keys \* ：表示查找所有的键；

keys r?dis：表示匹配redis、radis、rbdis等键；

keys r\*dis：表示匹配reedis、radfsdis等键；

keys r[ab]dis：表示可以匹配radis或者rbdis键；

遇到特殊符号时需要使用“\”进行转义。

返回值：返回匹配到的键列表。

### type

type命令用于返回键对应的值的类型。命令如下：

**type key**

key表示键

返回值：

如果键不存在，则返回none；

如果键对应的值是字符串类型的，则返回string；

如果键对应的值是列表类型的，则返回list；

如果键对应的值是哈希类型的，则返回hash；

如果键对应的值是集合类型的，则返回set；

如果键对应的值是有序集合类型的，则返回zset；

### randomkey

randomkey用于随机返回一个键，命令如下：

**randomkey**

返回值：如果这个数据库中存在键，则随机返回一个键（名）；如果这个数据库为空，则返回nil。

### object

object命令用于查看键的相关信息，命令如下：

**object subcommand [arguments [arguments] ]**

object命令用于从内部查看键的Redis对象。通常用在排错或者为了节省空间而对key使用特殊编码的情况下。如果要用Redis来实现缓存相关功能，则可以使用object命令来决定是否清除键。

object命令有如下子命令：

**object refcount key：**返回给定键引用所存储的值的次数，多用于排错；

**object encoding key：**返回给定键所存储的值所使用的底层数据结构；可以通过返回的值来判断该键属于哪种数据类型。

Redis对象具有多种编码格式：

·字符串：被编码为raw或int

·哈希表：被编码为zipmap或hashtable

·列表：被编码为ziplist或linkedlist

·集合：被编码为intset或hashtable

·有序集合：被编码为ziplist或skiplist

**object idletime key：**返回给定键自存储依赖的空闲时间，以秒为单位；

## 修改键

### rename

rename命令用于修改键的名称，命令如下：

**rename key newkey**

key表示要修改名称的键；

newkey表示键的新名称；

如果要修改名称的键不存在，则返回错误；

如果指定newkey已存在，则删除这个键，然后再将名称赋给要修改名称的键；

返回值：改名成功则返回OK，否则返回错误。

### renamenx

renamenx命令用于修改键的名称，但只有当新名称不存在时才可以修改成功。命令如下：

**renamenx key newkey**

key表示要修改名称的键；

newkey表示键的新名称；

如果key不存在，则返回错误；

如果newkey已存在，则返回0；

返回值：修改成功返回1。

## 键的序列化

### dump命令

dump命令用于序列化给定的键，命令如下：

**dump key**

key表示要被序列化的键；

使用dump命令序列化生存的值具有以下特点：

·这个值具有64位的校验和，用于检测错误。restore命令在进行反序列化之前，会先检查校验和；

·这个值的编码格式与rdb文件的编码格式保持一致；

·rdb版本会在编码在序列化值中，如果redis的版本不同，那么这个rdb文件会存在不兼容，redis也就无法对这个值进行反序列化；

·这个序列化的值中没有键的生存时间信息。

键被序列化后，会在Redis的安装目录中生成相应的文件，文件后缀名为rdb。

如果指定的键不存在，则返回nil；

返回值：执行成功后，返回被序列化后的值。

### restore

restore命令用于将一个给定的序列化值进行反序列化，并为它关联给定的键。命令如下：

**resotre key ttl serialized-value [replace]**

key表示键；

ttl表示是否为键设置生存时间，为0则不设置；

serialized-value表示序列化值；

restore命令在执行反序列化操作之前，会先对这个序列化的rdb版本和校验数据进行检查，如果rdb版本不相同或者数据不完整，那么restore命令会拒绝进行反序列化，并返回一个错误。

如果键已经存在，且设置了replace，那么使用反序列化得出的值来替换键的旧值；

如果没有设置replace参数，则会返回一个错误。

返回值：反序列成功，返回OK。

## 键的生存时间

### pttl

pttl命令用于获取键剩下的生存时间，以毫秒为单位。命令如下：

**pttl key**

key表示键；

返回值：

执行成功后返回键剩下的生存时间，单位为毫秒；

如果键不存在，则返回-2；

如果键存在但没有设置生存时间，则返回-1；

### ttl

ttl命令用于获取键剩下的生存时间，以秒为单位。命令如下：

**ttl key**

key表示键；

返回值：

执行成功后返回键剩下的生存时间，单位为秒；

如果键不存在，则返回-2；

如果键存在但没有设置生存时间，则返回-1；

### expire

expire命令用于设置键的生存时间，以秒为单位。命令如下：

**expire key seconds**

key表示键；

seconds表示秒数；

为键设置生存时间后，当生存时间结束，该键将被删除；

如果设置的生存时间为0，则表示删除该键；

如果修改了键的名称，设置的生存时间不会失效；

返回值：

设置成功后返回1；

如果key不存在或Redis版本过低，则返回0；

### pexpire

pexpire命令用于设置键的生存时间，以毫秒为单位。命令如下：

**pexpire key millseconds**

key表示键；

millseconds表示毫秒数；

返回值：

设置成功返回1；

设置失败（key不存在）返回0；

### expireat

expireat命令用于为键设置生存时间，以UNIX时间戳的形式设置。命令如下：

**expireat key timestamp**

返回值：设置成功返回1，当键不存在或不能为键设置生存时间时，返回0；

### pexpireat

pexpireat命令用于以毫秒为单位设置键的过期UNIX时间戳。命令如下：

**pexpireat key timestamp**

key表示键；

timestamp表示时间戳；

返回值：设置成功返回1；当键不存在或不能为键设置生存时间时，返回0；

## 键的转移

### move

move命令用于将键从当前数据库中剪切粘贴到本地的其他数据库中。命令如下：

**move key db**

key表示要转移的键；

db表示目标数据库；

返回值：

转移成功返回1；

如果要转移键不存在或要转移的键在目标数据库中已存在，则命令无效；

### migrate

migrate命令用于将键从当前数据库复制粘贴或剪切粘贴到远程的目标数据库中。命令如下：

**migrate host port key db timeout [copy] [replace]**

host表示ip地址；

post表示端口号；

key表示要转移的键；

db表示远程的目标数据库；

timeout用于指定超时时间；

copy表示复制粘贴，本地数据库中的该键不会被删除；

replace表示是否覆盖目标数据库中已存在的该键；

成功转移后，会将本地数据库中该键删除，除非使用了copy参数；

migrate具有原子性，所以在执行的时候会阻塞进行转移的两个数据库，直到转移成功，或者转移失败，又或者是等待超时；

migrate命令需要在指定的超时时间内完成，如果没有完成，那么该命令将终止执行，并返回一个特殊的错误IOERR，但键还是依然存在于本地数据库中的，不会丢失；

返回值：转移成功返回OK

### sort

sort命令主要用于排序。命令如下：

**sort key [by pattern] [LIMIT offset count] [GET pattern [get pattern……]**

**[asc|desc] [alpha] [store destination]**

key表示键；

1. 排序默认以数字作为排序根据，值会被解释为double类型浮点数，然后进行比较；

2）由于sort命令的可选参数众多，在后面的的章节中详细讲解sort命令的使用。

返回值：

如果没有使用store参数，那么排序结果将会以列表的形式返回；

如果使用了store参数，则返回排序结果的元素数量。

## 删除键

### del

del命令用于删除指定的一个或多个键。命令如下：

**del key [key…]**

key表示要删除的键；

返回值：返回成功被删除键的数量。

注：如果要清空当前数据库中所有的键，则执行flushall命令，如下：



### persist

persist命令用于删除键的生存时间，将键变为永久键，不会过期。命令如下：

**persist key**

key表示键；

返回值：成功删除后，返回1；如果键存在或并没有生存时间，则返回0。

# 数据库命令

## select

select用于切换数据库。命令如下：

**select index**

index表示数据库索引号，从0开始。

如果客户端连接上服务器之后没有通过select切换数据库，则默认使用0号数据库。

Redis默认有16个数据库，如果想要增加数据库，可以在Redis配置文件中配置databases 参数。

返回值：返回OK，如果索引值超过了数据库的最大索引（Redis默认为16个数据库，

可以在配置文件中修改）值则返回错误。

## dbsize

dbsize命令用于统计当前数据库中键的数量。命令如下：

**dbsize**

返回值：返回当前数据库中键的数量。

# HyperLogLog命令

HyperLogLog是Redis的用来做基数统计的一个算法。

1. 当Redis数据库中的数据量非常庞大时，使用HyperLogLog命令来计算相关基数时，它具有所需空间固定、所占空间小的优点。每个HyperLogLog键只需要耗费12KB的内存，就可以计算接近2^64个不同元素的基数。
2. 基数是什么呢？例如有数据集{1,3,5,8,5,9}，去掉重数据之后，得到的基数集是{1,3,5,8,9}，这个基数集的基数就是5。
3. HyperLogLog不会存储输入的元素，它仅仅根据输入的元素来计算基数，因此它不会返回输入的元素。
4. 利用HyperLogLog，用户可以使用较小的固定大小的内存来存储集合中的唯一元素，每个HyperLogLog只需要使用12KB的内存，以及几个字节的内存来存储键本身。

## pfadd

pfadd命令用于向HyperLogLog中添加一个或多个元素。命令如下：

**pfadd key elements [elements……]**

key表示HyperLogLog键；

elements表示要添加的元素，可以是多个；

执行pfadd命令后，HyperLogLog会添加它不包含的元素，来反映一个不同的唯一元素数量，这个数量就是集合的基数；

如果给定的HyperLogLog键不存在，则先会创建一个空的，在执行pfadd命令。

返回值：如果HyperLogLog内部更新了，则返回1，否则返回0。

## pfcount

pfcount命令用于返回HyperLogLog的近似基数。命令如下：

**pfcount key [key……]**

key表示HyperLogLog键；

当给定的HyperLogLog键只有一个，则返回存储它的近似基数。如果HyperLogLog不存在，则返回0；

当给定的HyperLogLog键有多个，则返回给定HyperLogLog的并集的近似基数，这个近似基数是通过将所有给定的HyperLogLog合并到一个临时的HyperLogLog中计算出来的。

利用HyperLogLog，用户可以使用较小且固定大小的内存来存储集合中的唯一元素，每个HyperLogLog只需要使用12KB的内存，以及几个字节的内存来存储键本身。

使用pf命令获得的基数并不是准确的，它是一个带有0.81%标准错误的近似值。

返回值：返回一个整数，这个整数是给定HyperLogLog包含的唯一元素的近似数量。

## pfmerge

pfmerge命令用于将多个HyperLogLog键合并为一个新的HyperLogLog键。命令如下：

**pfmerge destkey sourcekey [soucekey……]**

destkey是用于接收合并产生的HyperLogLog键，如果不存在，则创建；

sourcekey表示将要被合并的HyperLogLog键；

返回值：执行成功返回OK。

# 脚本命令

1. 使用Redis脚本可以一次性将多个请求命令发送出去，以减少网络的开销；
2. 使用Redis脚本命令实现原子操作，Redis会将整个脚本作为一个整体执行，中间不会有其他命令被执行，以此来保证原子性；
3. 使用Redis脚本可以达到复用的目的，因为Redis会永久保存客户端发送的脚本，所以其他客户端可以直接复用这个脚本。
4. Redis脚本使用Lua解释器来执行。

## 缓存中的Lua脚本

### script load

script load命令用于将脚本添加到服务器的缓存中。命令如下：

**script load script**

script表示脚本；

1. 使用script load命令将脚本script添加到脚本缓存中，但是并不会立即执行这个脚本；
2. 如果给定的脚本在缓存中已存在，那么这个命令什么也不做；
3. 脚本被加入缓存中后，可以通过evalsha命令使用脚本的SHA1校验和来调用这个脚本；
4. Lua脚本可以在Redis的缓存中长时间保存，直到遇到script flush命令为止；

返回值：成功执行后，返回给定脚本的SHA1校验和。

### script exists

script exists命令用于判断给定的脚本是否已经存在于缓存中。命令如下：

**script exists shal [shal…..]**

shal表示脚本的SHA1校验和；

返回值：成功执行后，返回一个列表，包含0和1，0表示缓存中不存在，1表示缓存中已存在，返回的这个列表中的元素和给定的SHA1校验和顺序位置一一对应。

## 对Lua脚本求值

### eval命令

eval命令用于对Lua脚本求值。命令如下：

**eval script numkeys key [key……] arg [arg…..]**

script是一段Lua脚本，它运行在Redis的服务器中；

numkeys用于指定键名参数的个数；

键名参数key，表示脚本中引用了哪些键，如果一个也没有引用，则numkeys参数值应该为0；在Lua脚本中，可以使用全局变量keys数组（下标从1开始）来访问这些键名参数；

参数arg [arg……]是附加参数。在Lua脚本中，可以使用argv数组（下标从1开始）来访问这些附加参数；

### evalsha

evalsha命令用于根据给定的SHA1校验码来对缓存在服务器中的脚本求值。命令如下：

**evalsha sha1 numkeys key [key……] arg [arg……]**

使用script load命令可以将脚本缓存到服务器中；

shal表示脚本的SHA1校验码；

返回值：以列表的形式返回脚本中代码的执行结果；

## 杀死或清除脚本

### script kill

script kill命令用于杀死一个正在运行的脚本。命令如下：

**script kill**

1. 针对一个正在运行且没有执行过任何写操作的Lua脚本，可以使用script kill命令来杀死它；
2. script kill命令主要用于终止运行时间过长的脚本，执行该命令后，当前正在运行的脚本会被杀死，执行这个脚本的客户端会从eval命令的阻塞当中退出，并收到一个错误的返回值；
3. 如果这个脚本执行过写操作，那么使用script kill命令是无法杀死它的；如果你非要杀死这个运行中的Lua脚本，则可以使用shutdown nosave命令来直接关闭整个Redis进程，进而停止这个脚本的运行，并防止不完整的数据写入数据库中；

返回值：成功杀死脚本，则返回OK；失败则返回错误信息；

### script flush

script flush命令用于清除Redis服务器中的所有Lua脚本缓存。命令如下：

**script flush**

返回值：总是返回OK；

# 连接命令

Redis的连接命令主要用于连接Redis的服务，如查看服务器状态、切换数据库等。

## auth

auth命令用于解锁密码。命令如下：

**auth password**

1. 我们可以通过修改配置文件中的requirepass项的值，来为Redis设置密码，进行来保护Redis服务器。命令为：

**config set requirepass password**

1. 为Redis设置密码后，客户端在连接服务端后需要使用auth命令来解锁密码，解锁成功后才可以使用Redis的其他命令。
2. 如果auth命令给定的password和配置文件设置的密码相同，则服务器会返回OK，否则返回错误信息，并要求客户端重新输入。

返回值：密码匹配成功返回OK，否则返回错误信息；

## quit

quit命令用于断开与服务端的连接。命令如下：

**quit**

返回值：总是返回OK

## ping

ping命令用于查看Redis服务器是否正常运行。命令如下：

**ping**

使用客户端向Redis服务端发出一个ping命令，如果服务器正常运行，则会返回一个pong；

我们常常使用ping命令来测试客户端与服务器的连接是否正常，或者用于测试服务器的延迟值；

返回值：客户端与服务器连接正常，则返回pong；

## echo

echo命令用于输出打印信息，主要在测试时使用。命令如下：

**echo message**

message表示要设置的信息；

返回值：返回设置的信息；

# 服务器命令

## 管理客户端

### client list

client list命令用于获取客户端的相关信息。命令如下：

**client list**

返回值：返回该客户端的相关信息，由多个键值对组成，每个键值对之间使用空格分开。

返回结果中键的含义如下：  
 id：表示客户端编号；

addr：表示客户端的地址，由IP地址和端口号组成；

fd：表示套接字所使用的文件描述符；

age：表示已连接的时长，单位为秒；

idle：表示连接空闲时长，单位为秒；

flags：表示客户端flag（见下文）；

db：表示当前正在使用的数据库；

sub：表示已订阅的消息频道数量；

psub：表示已订阅模式的数量；

multi：表示在事务中被执行的命令数量；

qbuf：表示查询缓冲区的长度，0表示没有分配查询缓冲区，单位为字节；

qbuf-free：表示查询缓冲区剩余空间的长度，0表示没有剩余空间，单位为字节；

obl：表示输出缓冲区的长度，0表示没有分配输出缓冲区，单位为字节；

oll：表示输出列表中包含的对象数量。如果输出缓冲区没有剩余空间，则命令为回复会以字符串对象的形式被添加到这个队列中；

omen：表示输出缓冲区和输出列表占用的内存总量；

event：表示文件描述符事件，值有以下两种情况：

·r：在loop事件中，表示套接字是可读的；

·w：在loop事件中，表示套接字是可写的；

cmd：表示最近一次执行的命令；

flag键对应的值由以下情况：

O：表示客户端是monitor模式的附属节点（slave）

S：客户端时一般模式（normal）的附属节点；

M：客户端时主节点（master）；

x：客户端正在执行事务；

b：客户端正在等待阻塞事件；

i：客户端正在等待VM I/O操作（已废弃）

d：一个受监视的键已被修改，exec命令将执行失败；

c：在将回复完整的写出来之后，关闭连接；

u：客户端未被阻塞；

A：尽可能快地关闭连接；

N：未设置任何flag

### client getname

client getname命令用于获取客户端的名称。命令如下：

**client getname**

客户端的名称表示的是本次连接服务器后，客户端通过client setname命令设置的名称；

返回值：返回客户端设置的名称；如果客户端没有设置名称，则返回nil。

### client setname

client setname命令用于设置客户端的名称。命令如下：

**client setname connection-name**

connection-name表示要设置的名称；

返回值：设置成功返回OK。

### client pause

client pause命令用于在指定的时间内停止运行来自客户端的命令。命令如下：

**client pause timeout**

timeout表示要指定的时间，单位为毫秒；

在指定的时间内，客户端输入的命令执行时会进入阻塞状态，当等待时间超时，再执行输入的命令。

返回值：成功执行返回OK，如果timeout类型非法，则返回错误；

### client kill

client kill命令用于关闭一个客户端的连接。命令如下：

**client kill ip:port**

ip表示IP地址；

post表示端口号；

Redis是单线程的，当一个Redis命令正在执行时，不会有客户端被断开连接，它只是关闭了客户端，并不会断开与客户端的连接。比如，如果设置了密码，则使用client kill命令关闭客户端的连接后，该客户端则又需要使用auth命令来解锁密码。

返回值：如果指定的客户端存在，且成功关闭，则返回OK。

## 查看Redis服务器信息

### command

command命令用于返回Redis所有命令的描述信息。命令如下：

**command**

返回值：以列表的形式返回Redis的所有命令的描述信息。

### command count

command count命令用于统计Redis命令的个数。命令如下：

**command count**

返回值：返回Redis命令的个数。

### command getkeys

command getkeys命令用于获取指定的键值对中所有的键。命令如下：

**command getkeys key**

参数key不是键的名称，而是一条创建键语句，如set name zhangsan；

返回值：以列表的形式返回给定键值对的所有键。

### command info

command info命令用于获取指定Redis命令的描述信息。命令如下：

**command info key [key…..]**

key表示Redis的命令，如set、get等，可以有多个；

返回值：以列表的形式返回给定命令的描述信息；

### info

info命令用于查看Redis服务器的各种信息以及统计相关数值。命令如下：

**into [section]**

section为可选参数，如果没有指定，则默认查看服务器的所有信息；如果指定，则查看服务器指定部分的信息，它的取值如下：

·server：该部分说明了Redis的服务器信息；

·clients：该部分记录了已连接客户端的信息；

·memory：该部分记录了Redis服务器的内存相关信息；

·persitence：该部分记录了与持久化（RDB持久化和AOF持久化）相关的信息；

·stats：该部分记录了相关的统计信息；

·replication：该部分记录了Redis数据库主从复制信息；

·cpu：该部分记录了CPU的计算量统计信息；

·commandstats：该部分记录了Redis各种命令的执行统计信息，如执行命令消耗的CPU时间、执行次数等。

·cluster：该部分记录了与Redis集群相关的信息；

·keyspace：该部分记录了与Redis数据库相关的统计信息，如键的数量。

返回值：返回服务器所有的信息或指定部分的信息。

### lastsave

lastsave命令用于返回最近一次保存数据的时间。命令如下：

**lastsave**

返回值：返回一个UNIX时间戳。

### monitor

monitor命令常用于调试，能够实时打印出Redis服务器接收到的命令。命令如下：

**monitor**

在客户端A上使用了monitor命令后，进入阻塞状态，其他客户端执行的命令会在客户端A上打印出来，并且能够显示其他客户端的IP和端口号，以及执行命令的时间。

返回值：执行成功后返回OK，并进入阻塞状态。

### time

time命令用于获取当前服务器的时间。命令如下：

**time**

返回值：返回包含两个字符串的列表，第一个字符串当前时间，为UNIX时间戳形式；第二个字符串是在当前这一秒内逝去的微秒数。

## 修改并查看相关配置

### config set

config set命令用于修改Redis服务器的配置，并且修改之后不需要重启服务器就能生效。命令如下：

**config set parameter value**

parameter表示要修改的配置参数对应的名称，它都可以在Redis的配置文件redis.conf中找到；

value表示配置参数对应的值；

返回值：成功执行返回OK，否则返回错误信息；

### config get

config get命令用于根据根据给定的配置参数的值。命令如下：

**config get parameter**

parameter表示配置参数，它都可以在Redis的配置文件redis.conf中找到；如果parameter值为\*，则表示列出Redis服务器的所有配置参数及参数值；如果parameter值为a\*，则表示列出Redis服务器的以a开头的配置参数及参数值。

返回值：返回给定配置参数的参数名及对应的值；

### config resetstat

config resetstat命令用于重置info命令中的一些统计数据。命令如下：

**config resetstat**

它能够重置的数据有：

·Keyspace hits：表示键空间命中次数；

·Keyspace misses：表示键空间不命中次数；

·Number of commands processed：表示执行命令的次数；

·Number of connections received：表示连接服务器的次数；

·Number of expired keys：表示过期key的数量；

·Number of rejected conections：表示被拒绝的连接数量；

·Latest fork(2) time：表示最后执行fork(2)的时间；

·The aof\_delayed\_fsync counter：表示aof\_delayed\_fsync计数器的值；

返回值：总是返回OK。

### config rewrite

config rewrite命令用于改写Redis配置文件，将当前客户端正在使用的配置信息保存到redis.conf文件中。命令如下：  
 **config rewrite**

命令：修改成功，则返回OK；修改失败，返回错误信息。

（此命令有待完善）

## 数据持久化

### bgrewriteaof

bgrewriteof命令用于执行AOF文件重写操作，创建一个当前AOF文件的优化版本。命令如下：

**bgrewriteaof**

1. AOF文件记录了所有执行过的命令，是Redis数据持久化的方式之一。
2. 如果bgrewriteaof命令执行失败，AOF文件数据并不会丢失，因为旧的AOF文件在bgrewriteaof命令执行成功之前是不会被修改的，所以不存在数据丢失的问题。
3. 在Redis的版本中（Redis 2.4以后），AOF文件的重写将由Redis自动触发的，使用bgrewriteaof命令只是手动触发重写操作。

返回值：返回一条反馈信息。

### save

save命令用于将数据保存到磁盘中，以同步的方式。命令如下：

**save**

1. save命令的执行是一个同步保存操作，它以RDB文件的形式将当前Redis中的所有数据快照保存到磁盘中
2. 在生产环境中，不建议使用save命令来保存数据，因为它在执行时会阻塞所有客户端，所以推荐使用bgsave命令以异步的方式执行保存数据的操作，如果后台子进程保存数据失败，或者出现其他问题，则可以使用save命令来做最后的保存。

返回值：执行成功（数据保存成功），返回OK。

### bgsave

bgsave命令用于将数据保存到磁盘中，以异步的方式。命令如下：

**bgsave**

1. bgsave的执行原理：执行bgsave命令后返回OK，之后Redis启动一个新的子进程，原来的Redis进程（父进程）继续执行客户端输入的其他命令，而子进程则负责将数据保存到磁盘中，然后退出。
2. 我们可以使用lastsave命令来查看相关信息，进而判断bgsave命令是否将数据保存成功。

返回值：返回一条提示信息。

## 实现主从服务

### sync

sync命令是Redis复制功能的内部命令。命令如下：

**sync**

返回值：没有明确的返回值。

（有待完善）

### psync

psync命令也是Redis复制功能的内部命令。命令如下：

**psync <master\_runid> <offset>**

返回值：没有明确的返回值。

（有待完善）

### slaveof

slaveof命令用于动态的修改复制功能的行为。命令如下：

**slaveof host post**

1. 我们可以利用slaveof命令来修改当前服务器，使其转变为指定服务器的从属服务器。如果当前服务器是某个主服务器的从属服务器，则执行slaveof命令后，会使当前服务器停止对旧主服务器的同步，并且将旧数据丢弃，然后开始对新主服务器数据进行同步。
2. 如果想在同步时不丢失数据集，则可以使用slaveof no one命令，该命令执行后不会丢弃同步数据集。当主服务器出现故障时，我们可以利用该命令将从属服务器用作新的主服务器，实现数据不丢失，不间断运行。

（有待完善）

### role

role命令用于查看主从服务器的角色。命令如下：

**role**

角色有3种：master、slave、sentinel。

返回值：成功执行后返回一个数组，该数组中记录的是主从服务器的角色。

## 服务器管理

### slowlog

slowlog命令用于管理Redis的慢日志。命令如下：

**slowlog subcommand [argument]**

1. Slow log（慢查询）是Redis的日志系统，用于记录查询执行时间；查询执行时间指的是执行一个查询命令所耗费的时间，它不包括客户端响应、发送信息等I/O操作；Slow log保存在内存里面，读写速递非常快；
2. slowlog命令有多个子命令，每个子命令的作用不一样，返回值也不同；关于slowlog命令的其他相关用法，在后面的章节中会详细讲解。

（有待完善）

### shutdown

shutdown命令具有多种作用，如下：

·关闭Redis服务器；

·关闭（停止）所有客户端；

·在AOF选项被打开的情况下，执行SHUTDOWN命令将会更新AOF文件；

·如果Redis服务器中至少存在一个保存点在等待，则在执行shutdown命令的同时将会执行save命令；

命令如下：

**shutdown [save|nosave]**

1. 在持久化被打开的情况下，执行shutdown命令，服务器会被正常关闭，且不会丢失数据；
2. shutdown save命令会强制Redis数据库执行保存命令，即使没有设置保存点，也会执行，同样服务器也会被正常关闭，且不会丢失数据；
3. shutdown nosave命令的作用域shutdown save命令的作用刚好相反，它会阻止Redis数据库执行保存操作，即使设置了一个或多个保存点，也会阻止；同样服务器会被正常关闭。

返回值：

执行成功什么也不返回，服务器被关闭，客户端提示没有连接到服务器；

执行失败返回错误信息；

（有待完善）