|  |
| --- |
| Neusoft |
| CentOS  &  Redis |
| Redis 环境搭建 |

|  |
| --- |
| 成阳阳  2015/5/15 |
|  |

# CentOS 7 安装Redis

准备  
  
1. redis 3.0.1版  
    [http://download.redis.io/releases/redis-2.6.16.tar.gz](http://download.redis.io/releases/redis-2.6.16.tar.gz" \t "_blank)  
安装计划及具体步骤笔记  
  
  
一、计划  
  
1. 下载redis稳定版本  
2. 解压并安装redis，并放入合理的目录中  
3. 手工启动redis，测试redis是否运行正常  
4. 将redis加入开机自启动序列  
  
二、具体步骤笔记  
  
1. 默认安装的CentOS系统自带wget小工具，所以我们利用它来在linux系统里下载redis  
（1）获取响应权限：su root  
（2）规划好下载的目录：mkdir /usr/local/soft/  
（3）进入新建的目录，准备下载：cd /usr/local/soft/  
（4）下载redis最新稳定版：wget <http://download.redis.io/releases/redis-3.0.1.tar.gz>  
  
2.  解压并安装redis，并放入合理的目录中  
（1）解压tar包：tar zxvf redis-3.0.1.tar.gz  
（2）进入解压后出现的新目录：cd redis-3.0.1  
（3）编译源码：make  
（4）校验编译是否正确：make test  
        此时会有错误提示：You need tcl 8.5 or newer in order to run the Redis test  
（5）根据以上错误提示，安装ActiveTcl 8.5 以上版本（我选的是最新的8.6版本，约20多MB，URL：[http://downloads.activestate.com/ActiveTcl/releases/](http://downloads.activestate.com/ActiveTcl/releases/" \t "_blank)）

下载后 解压 运行install.sh 安装

创建logs文件夹

Redis主配置文件 redis.conf需改

|  |
| --- |
| #后台运行  daemonize yes  #log文件目录  logfile “../logs/server.out” |

编写运行脚本

|  |
| --- |
| ../src/redis-server ../redis.conf  tail -fn 100 ../logs/server.out |

停止脚本

|  |
| --- |
| ../src/redis-cli shutdown |

配置slave服务器很简单，只需要在配置文件redis.conf中加入如下配置

slaveof 192.168.1.1 6379 #指定master的ip和端口

3. 手工启动redis，测试redis是否运行正常  
（1）在当前redis目录中输入命令启动Redis：src/redis-server  
（2）检查进程是否有Redis了：ps aux | grep redis，可以看见进程列表中有一个叫“src/redis-server”的进程了  
（3）启动redis自带的客户端，进行测试：  
        # src/redis-cli -- 启动客户端并连接本地Redis  
        # set foo bar -- 提示 “ok”说明设置键值正常。  
        # get foo -- 能够提示返回正确的“bar”，说明运行正常。  
        # quit -- 退出客户端

主从复制:让多个slave server拥有和master server相同的数据库副本。

特点

1.master可以有多个slave

2.除了多个slave连到相同的master外，slave也可以连接其他slave形成图状结构

3.主从复制不会阻塞master。也就是说当一个或多个slave与master进行初次同步数据时，master可以继续处理client发来的请求。

相反slave在初次同步数据时则会阻塞不能处理client的请求。

4.主从复制可以用来提高系统的可伸缩性,我们可以用多个slave 专门用于client的读请求，

如sort操作可以使用slave来处理。也可以用来做简单的数据冗余

5.可以在master禁用数据持久化，只需要注释掉master 配置文件中的所有save配置，然后只在slave上配置数据持久化

主从复制的过程

当设置好slave服务器后，slave会建立和master的连接，然后发送sync命令。

无论是第一次同步建立的连接还是连接断开后的重新连 接，master都会启动一个后台进程，将数据库快照保存到文件中，

同时master主进程会开始收集新的写命令并缓存起来。后台进程完成写文件 后，master就发送文件给slave，slave将文件保存到磁盘上，

然后加载到内存恢复数据库快照到slave上。接着master就会把缓存的命 令转发给slave。

而且后续master收到的写命令都会通过开始建立的连接发送给slave。

从master到slave的同步数据的命令和从 client发送的命令使用相同的协议格式。当master和slave的连接断开时slave可以自动重新建立连接。

如果master同时收到多个 slave发来的同步连接命令，只会使用启动一个进程来写数据库镜像，然后发送给所有slave。

三、配置Redis

使用配置文件启动：src/redis-server redis.conf

主要配置项：

Redis支持很多的参数，但都有默认值。

　　●daemonize:

　　默认情况下，redis不是在后台运行的，如果需要在后台运行，把该项的值更改为yes。

　　●pidfile

　　当Redis在后台运行的时候，Redis默认会把pid文件放在/var/run/redis.pid，你可以配置到其他地址。当运行多个redis服务时，需要指定不同的pid文件和端口。

　　●bind

　　指定Redis只接收来自于该IP地址的请求，如果不进行设置，那么将处理所有请求，在生产环境中最好设置该项。

　　●port

　　监听端口，默认为6379。

　　●timeout

　　设置客户端连接时的超时时间，单位为秒。当客户端在这段时间内没有发出任何指令，那么关闭该连接。

　　●loglevel

　　log等级分为4级，debug, verbose, notice, 和warning。生产环境下一般开启notice。

　　●logfile

　　配置log文件地址，默认使用标准输出，即打印在命令行终端的窗口上。

　　●databases

　　设置数据库的个数，可以使用SELECT 命令来切换数据库。默认使用的数据库是0。

　　●save

　　设置Redis进行数据库镜像的频率。

　　if(在60秒之内有10000个keys发生变化时){

　　进行镜像备份

　　}else if(在300秒之内有10个keys发生了变化){

　　进行镜像备份

　　}else if(在900秒之内有1个keys发生了变化){

　　进行镜像备份

　　}

　　●rdbcompression

　　在进行镜像备份时，是否进行压缩。

　　●dbfilename

　　镜像备份文件的文件名。

　　●dir

　 　数据库镜像备份的文件放置的路径。这里的路径跟文件名要分开配置是因为Redis在进行备份时，先会将当前数据库的状态写入到一个临时文件中，等备份完 成时，再把该该临时文件替换为上面所指定的文件，而这里的临时文件和上面所配置的备份文件都会放在这个指定的路径当中。

　　●slaveof

　　设置该数据库为其他数据库的从数据库。

　　●masterauth

　　当主数据库连接需要密码验证时，在这里指定。

　　●requirepass

　　设置客户端连接后进行任何其他指定前需要使用的密码。警告：因为redis速度相当快，所以在一台比较好的[服务器](http://product.pcpop.com/Server/10734_1.html" \o "服务器" \t "_blank)下，一个外部的用户可以在一秒钟进行150K次的密码尝试，这意味着你需要指定非常非常强大的密码来防止暴力破解。

　　●maxclients

　　限制同时连接的客户数量。当连接数超过这个值时，redis将不再接收其他连接请求，客户端尝试连接时将收到error信息。

　　●maxmemory

　　设置redis能够使用的最大[内存](http://product.pcpop.com/Memory/10734_1.html" \o "内存" \t "_blank)。 当内存满了的时候，如果还接收到set命令，redis将先尝试剔除设置过expire信息的key，而不管该key的过期时间还没有到达。在删除时，将 按照过期时间进行删除，最早将要被过期的key将最先被删除。如果带有expire信息的key都删光了，那么将返回错误。这样，redis将不再接收写 请求，只接收get请求。maxmemory的设置比较适合于把redis当作于类似memcached的缓存来使用。

　　●appendonly

　 　默认情况下，redis会在后台异步的把数据库镜像备份到磁盘，但是该备份是非常耗时的，而且备份也不能很频繁，如果发生诸如拉闸限电、拔插头等状况， 那么将造成比较大范围的数据丢失。所以redis提供了另外一种更加高效的数据库备份及灾难恢复方式。开启append only模式之后，redis会把所接收到的每一次写操作请求都追加到appendonly.aof文件中，当redis重新启动时，会从该文件恢复出之 前的状态。但是这样会造成appendonly.aof文件过大，所以redis还支持了BGREWRITEAOF指令，对appendonly.aof 进行重新整理。所以我认为推荐生产环境下的做法为关闭镜像，开启appendonly.aof，同时可以选择在访问较少的时间每天对 appendonly.aof进行重写一次。

　　●appendfsync

　　设置对appendonly.aof文件进行同步的频率。always表示每次有写操作都进行同步，everysec表示对写操作进行累积，每秒同步一次。这个需要根据实际业务场景进行配置。

　　●vm-enabled

　 　是否开启虚拟内存支持。因为redis是一个内存数据库，而且当内存满的时候，无法接收新的写请求，所以在redis 2.0中，提供了虚拟内存的支持。但是需要注意的是，redis中，所有的key都会放在内存中，在内存不够时，只会把value值放入交换区。这样保证 了虽然使用虚拟内存，但性能基本不受影响，同时，你需要注意的是你要把vm-max-memory设置到足够来放下你的所有的key。

　　●vm-swap-file

　　设置虚拟内存的交换文件路径。

　　●vm-max-memory

　　这里设置开启虚拟内存之后，redis将使用的最大物理内存的大小。默认为0，redis将把他所有的能放到交换文件的都放到交换文件中，以尽量少的使用物理内存。在生产环境下，需要根据实际情况设置该值，最好不要使用默认的0。

　　●vm-page-size

　　设置虚拟内存的页大小，如果你的value值比较大，比如说你要在value中放置博客、新闻之类的所有文章内容，就设大一点，如果要放置的都是很小的内容，那就设小一点。

　　●vm-pages

　　设置交换文件的总的page数量，需要注意的是，page table信息会放在物理内存中，每8个page就会占据RAM中的1个byte。总的虚拟内存大小 = vm-page-size \* vm-pages。

　　●vm-max-threads

　　设置VM IO同时使用的线程数量。因为在进行内存交换时，对数据有编码和解码的过程，所以尽管IO设备在硬件上本上不能支持很多的并发读写，但是还是如果你所保存的vlaue值比较大，将该值设大一些，还是能够提升性能的。

　　●glueoutputbuf

　　把小的输出缓存放在一起，以便能够在一个TCP packet中为客户端发送多个响应，具体原理和真实效果我不是很清楚。所以根据注释，你不是很确定的时候就设置成yes。

　　●hash-max-zipmap-entries

　　在redis 2.0中引入了hash数据结构。当hash中包含超过指定元素个数并且最大的元素没有超过临界时，hash将以一种特殊的编码方式(大大减少内存使用)来存储，这里可以设置这两个临界值。

　　●activerehashing

　　开启之后，redis将在每100毫秒时使用1毫秒的[CPU](http://product.it168.com/list/b/0217_1.shtml" \o "CPU" \t "_blank)时 间来对redis的hash表进行重新hash，可以降低内存的使用。当你的使用场景中，有非常严格的实时性需要，不能够接受Redis时不时的对请求有 2毫秒的延迟的话，把这项配置为no。如果没有这么严格的实时性要求，可以设置为yes，以便能够尽可能快的释放内存。

四、操作Redis

1、插入数据

　　redis 127.0.0.1:6379> set name wwl  
  
　　OK

　　设置一个key-value对。

　　2、查询数据

　　redis 127.0.0.1:6379> get name  
  
　　"wwl"

　　取出key所对应的value。

　　3、删除键值

　　redis 127.0.0.1:6379> del name

　　删除这个key及对应的value。

　　4、验证键是否存在

　　redis 127.0.0.1:6379> exists name  
  
　　(integer) 0

　　其中0，代表此key不存在;1代表存在。

五、各类型的基本操作

1）strings类型及操作

　　string是最简单的类型，你可以理解成与Memcached是一模一样的类型，一个key对应一个value，其上支持的操作与Memcached的操作类似。但它的功能更丰富。

　　string类型是二进制安全的。意思是redis的string可以包含任何数据，比如jpg图片或者序列化的对象。从内部实现来看其实string可以看作byte数组，最大上限是1G字节，下面是string类型的定义:

　　struct sdshdr {  
  
　　long len;  
  
　　long free;  
  
　　char buf[];  
  
　　};

　　len是buf数组的长度。

　　free是数组中剩余可用字节数，由此可以理解为什么string类型是二进制安全的了，因为它本质上就是个byte数组，当然可以包含任何数据了

　　buf是个char数组用于存贮实际的字符串内容，其实char和c#中的byte是等价的，都是一个字节。

　　另外string类型可以被部分命令按int处理.比如incr等命令，如果只用string类型，redis就可以被看作加上持久化特性的memcached。当然redis对string类型的操作比memcached还是多很多的，具体操作方法如下：

　　1、set

　　设置key对应的值为string类型的value。

　　例如我们添加一个name= HongWan的键值对，可以这样做:

　　redis 127.0.0.1:6379> set name HongWan  
  
　　OK  
  
　　redis 127.0.0.1:6379>

　　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。

　　例如我们添加一个name= HongWan\_new的键值对，可以这样做:

　　redis 127.0.0.1:6379> get name  
  
　　"HongWan"  
  
　　redis 127.0.0.1:6379> setnx name HongWan\_new  
  
　　(integer) 0  
  
　　redis 127.0.0.1:6379> get name  
  
　　"HongWan"  
  
　　redis 127.0.0.1:6379>

　　由于原来name有一个对应的值，所以本次的修改不生效，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。

　　例如我们添加一个haircolor= red的键值对，并指定它的有效期是10秒，可以这样做:

　　redis 127.0.0.1:6379> setex haircolor 10 red  
  
　　OK  
  
　　redis 127.0.0.1:6379> get haircolor  
  
　　"red"  
  
　　redis 127.0.0.1:6379> get haircolor  
  
　　(nil)  
  
　　redis 127.0.0.1:6379>

　　可见由于最后一次的调用是10秒以后了，所以取不到haicolor这个键对应的值。

　　4、setrange

　　设置指定key的value值的子字符串。

　　例如我们希望将HongWan的126邮箱替换为gmail邮箱，那么我们可以这样做:

　　redis 127.0.0.1:6379> get name  
  
　　"HongWan@126.com"  
  
　　redis 127.0.0.1:6379> setrange name 8 gmail.com  
  
　　(integer) 17  
  
　　redis 127.0.0.1:6379> get name  
  
　　"HongWan@gmail.com"  
  
　　redis 127.0.0.1:6379>

　　其中的8是指从下标为8(包含8)的字符开始替换

　　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。

　　redis 127.0.0.1:6379> mset key1 HongWan1 key2 HongWan2  
  
　　OK  
  
　　redis 127.0.0.1:6379> get key1  
  
　　"HongWan1"  
  
　　redis 127.0.0.1:6379> get key2  
  
　　"HongWan2"  
  
　　redis 127.0.0.1:6379>

　　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。

　　redis 127.0.0.1:6379> get key1  
  
　　"HongWan1"  
  
　　redis 127.0.0.1:6379> get key2  
  
　　"HongWan2"  
  
　　redis 127.0.0.1:6379> msetnx key2 HongWan2\_new key3 HongWan3  
  
　　(integer) 0  
  
　　redis 127.0.0.1:6379> get key2  
  
　　"HongWan2"  
  
　　redis 127.0.0.1:6379> get key3  
  
　　(nil)

　　可以看出如果这条命令返回0，那么里面操作都会回滚，都不会被执行。

　　7、get

　　获取key对应的string值,如果key不存在返回nil。

　　例如我们获取一个库中存在的键name，可以很快得到它对应的value

　　redis 127.0.0.1:6379> get name  
  
　　"HongWan"  
  
　　redis 127.0.0.1:6379>

　　我们获取一个库中不存在的键name1，那么它会返回一个nil以表时无此键值对

　　redis 127.0.0.1:6379> get name1  
  
　　(nil)  
  
　　redis 127.0.0.1:6379>

　　8、getset

　　设置key的值，并返回key的旧值。

　　redis 127.0.0.1:6379> get name  
  
　　"HongWan"  
  
　　redis 127.0.0.1:6379> getset name HongWan\_new  
  
　　"HongWan"  
  
　　redis 127.0.0.1:6379> get name  
  
　　"HongWan\_new"  
  
　　redis 127.0.0.1:6379>

　　接下来我们看一下如果key不存的时候会什么样儿?

　　redis 127.0.0.1:6379> getset name1 aaa  
  
　　(nil)  
  
　　redis 127.0.0.1:6379>

　　可见，如果key不存在，那么将返回nil

　　9、getrange

　　获取指定key的value值的子字符串。

　　具体样例如下:

　　redis 127.0.0.1:6379> get name  
  
　　"HongWan@126.com"  
  
　　redis 127.0.0.1:6379> getrange name 0 6  
  
　　"HongWan"  
  
　　redis 127.0.0.1:6379>

　　字符串左面下标是从0开始的

　　redis 127.0.0.1:6379> getrange name -7 -1  
  
　　"126.com"  
  
　　redis 127.0.0.1:6379>

　　字符串右面下标是从-1开始的

　　redis 127.0.0.1:6379> getrange name 7 100  
  
　　"@126.com"  
  
　　redis 127.0.0.1:6379>

　　当下标超出字符串长度时，将默认为是同方向的最大下标

　　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。

　　具体样例如下:

　　redis 127.0.0.1:6379> mget key1 key2 key3  
  
　　1) "HongWan1"  
  
　　2) "HongWan2"  
  
　　3) (nil)  
  
　　redis 127.0.0.1:6379>

　　key3由于没有这个键定义，所以返回nil。

　　11、incr

　　对key的值做加加操作,并返回新的值。注意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

　　redis 127.0.0.1:6379> set age 20  
  
　　OK  
  
　　redis 127.0.0.1:6379> incr age  
  
　　(integer) 21  
  
　　redis 127.0.0.1:6379> get age  
  
　　"21"  
  
　　redis 127.0.0.1:6379>

　　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

　　redis 127.0.0.1:6379> get age  
  
　　"21"  
  
　　redis 127.0.0.1:6379> incrby age 5  
  
　　(integer) 26  
  
　　redis 127.0.0.1:6379> get name  
  
　　"HongWan@gmail.com"  
  
　　redis 127.0.0.1:6379> get age  
  
　　"26"  
  
　　redis 127.0.0.1:6379>

　　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

　　redis 127.0.0.1:6379> get age  
  
　　"26"  
  
　　redis 127.0.0.1:6379> decr age  
  
　　(integer) 25  
  
　　redis 127.0.0.1:6379> get age  
  
　　"25"  
  
　　redis 127.0.0.1:6379>

　　14、decrby

　　同decr，减指定值。

　　redis 127.0.0.1:6379> get age  
  
　　"25"  
  
　　redis 127.0.0.1:6379> decrby age 5  
  
　　(integer) 20  
  
　　redis 127.0.0.1:6379> get age  
  
　　"20"  
  
　　redis 127.0.0.1:6379>

　　decrby完全是为了可读性，我们完全可以通过incrby一个负值来实现同样效果，反之一样。

　　redis 127.0.0.1:6379> get age  
  
　　"20"  
  
　　redis 127.0.0.1:6379> incrby age -5  
  
　　(integer) 15  
  
　　redis 127.0.0.1:6379> get age  
  
　　"15"  
  
　　redis 127.0.0.1:6379>

　　15、append

　　给指定key的字符串值追加value,返回新字符串值的长度。例如我们向name的值追加一个@126.com字符串，那么可以这样做:

　　redis 127.0.0.1:6379> append name @126.com  
  
　　(integer) 15  
  
　　redis 127.0.0.1:6379> get name  
  
　　"HongWan@126.com"  
  
　　redis 127.0.0.1:6379>

　　16、strlen

　　取指定key的value值的长度。

　　redis 127.0.0.1:6379> get name  
  
　　"HongWan\_new"  
  
　　redis 127.0.0.1:6379> strlen name  
  
　　(integer) 11  
  
　　redis 127.0.0.1:6379> get age  
  
　　"15"  
  
　　redis 127.0.0.1:6379> strlen age  
  
　　(integer) 2  
  
　　redis 127.0.0.1:6379>

2）hash

Redis hash是一个[string类型](http://tech.it168.com/a2011/0901/1241/000001241205.shtml" \t "_blank)的field和value的映射表.它的添加、删除操作都是O(1)(平均)。hash特别适合用于存储对象。相较于将对象的每个字段存成单个string类型。将一个对象存储在hash类型中会占用更少的[内存](http://product.it168.com/list/b/0205_1.shtml" \o "内存" \t "_blank)，并且可以更方便的存取整个对象。省[内存](http://product.pcpop.com/Memory/10734_1.html" \o "内存" \t "_blank)的 原因是新建一个hash对象时开始是用zipmap(又称为small hash)来存储的。这个zipmap其实并不是hash table，但是zipmap相比正常的hash实现可以节省不少hash本身需要的一些元数据存储开销。尽管zipmap的添加，删除，查找都是 O(n)，但是由于一般对象的field数量都不太多。所以使用zipmap也是很快的,也就是说添加删除平均还是O(1)。如果field或者 value的大小超出一定限制后，Redis会在内部自动将zipmap替换成正常的hash实现. 这个限制可以在配置文件中指定

　　hash-max-zipmap-entries 64 #配置字段最多64个。

　　hash-max-zipmap-value 512 #配置value最大为512字节。

**1、hset**

　　设置hash field为指定值，如果key不存在，则先创建。

　　redis 127.0.0.1:6379> hset myhash field1 Hello  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379>

**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。

　　redis 127.0.0.1:6379> hsetnx myhash field "Hello"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> hsetnx myhash field "Hello"  
  
　　(integer) 0  
  
　　redis 127.0.0.1:6379>

　　第一次执行是成功的，但第二次执行相同的命令失败，原因是field已经存在了。

**3、hmset**

　　同时设置hash的多个field。

　　redis 127.0.0.1:6379> hmset myhash field1 Hello field2 World  
  
　　OK  
  
　　redis 127.0.0.1:6379>

**4、hget**

　　获取指定的hash field。

　　redis 127.0.0.1:6379> hget myhash field1  
  
　　"Hello"  
  
　　redis 127.0.0.1:6379> hget myhash field2  
  
　　"World"  
  
　　redis 127.0.0.1:6379> hget myhash field3  
  
　　(nil)  
  
　　redis 127.0.0.1:6379>

　　由于数据库没有field3，所以取到的是一个空值nil。

**5、hmget**

　　获取全部指定的hash filed。

　　redis 127.0.0.1:6379> hmget myhash field1 field2 field3  
  
　　1) "Hello"  
  
　　2) "World"  
  
　　3) (nil)  
  
　　redis 127.0.0.1:6379>

　　由于数据库没有field3，所以取到的是一个空值nil。

**6、hincrby**

　　指定的hash filed 加上给定值。

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

　　在本例中我们将field3的值从20降到了12，即做了一个减8的操作。

**7、hexists**

　　测试指定field是否存在。

　　redis 127.0.0.1:6379> hexists myhash field1  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> hexists myhash field9  
  
　　(integer) 0  
  
　　redis 127.0.0.1:6379>

　　通过上例可以说明field1存在，但field9是不存在的。

**8、hlen**

　　返回指定hash的field数量。

　　redis 127.0.0.1:6379> hlen myhash  
  
　　(integer) 4  
  
　　redis 127.0.0.1:6379>

　　通过上例可以看到myhash中有4个field。

**9、hdel**

　　返回指定hash的field数量。

　　redis 127.0.0.1:6379> hlen myhash  
  
　　(integer) 4  
  
　　redis 127.0.0.1:6379> hdel myhash field1  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> hlen myhash  
  
　　(integer) 3  
  
　　redis 127.0.0.1:6379>

**10、hkeys**

　　返回hash的所有field。

　　redis 127.0.0.1:6379> hkeys myhash  
  
　　1) "field2"  
  
　　2) "field"  
  
　　3) "field3"  
  
　　redis 127.0.0.1:6379>

　　说明这个hash中有3个field。

**11、hvals**

　　返回hash的所有value。

　　redis 127.0.0.1:6379> hvals myhash  
  
　　1) "World"  
  
　　2) "Hello"  
  
　　3) "12"  
  
　　redis 127.0.0.1:6379>

　　说明这个hash中有3个field。

**12、hgetall**

　　获取某个hash中全部的filed及value。

　　redis 127.0.0.1:6379> hgetall myhash  
  
　　1) "field2"  
  
　　2) "World"  
  
　　3) "field"  
  
　　4) "Hello"  
  
　　5) "field3"  
  
　　6) "12"  
  
　　redis 127.0.0.1:6379>

　　可见，一下子将myhash中所有的field及对应的value都取出来了。

3）list

Redis的list类型其实就是一个每个子元素都是string类型的双向链表。链表的最大长度是(2的32次方)。我们可以通过push,pop操作从链表的头部或者尾部添加删除元素。这使得list既可以用作栈，也可以用作队列。

　　有意思的是list的pop操作还有阻塞版本的，当我们[lr]pop一个 list对象时，如果list是空，或者不存在，会立即返回nil。但是阻塞版本的b[lr]pop可以则可以阻塞，当然可以加超时时间，超时后也会返回 nil。为什么要阻塞版本的pop呢，主要是为了避免轮询。举个简单的例子如果我们用list来实现一个工作队列。执行任务的thread可以调用阻塞版 本的pop去获取任务这样就可以避免轮询去检查是否有任务存在。当任务来时候工作线程可以立即返回，也可以避免轮询带来的延迟。说了这么多，接下来看一下 实际操作的方法吧：

**1、lpush**

　　在key对应list的头部添加字符串元素：

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

　　在此处我们先插入了一个world，然后在world的头部插入了一个hello。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：

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

　　在此处我们先插入了一个hello，然后在hello的尾部插入了一个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：

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

　　在此处我们先插入了一个hello，然后在hello的尾部插入了一个world，然后又在world的前面插入了there。

**4、lset**

　　设置list中指定下标的元素值(下标从0开始)：

redis 127.0.0.1:6379> rpush mylist4 "one"  
(integer) 1  
redis 127.0.0.1:6379> rpush mylist4 "two"  
(integer) 2  
redis 127.0.0.1:6379> rpush mylist4 "three"  
(integer) 3  
redis 127.0.0.1:6379> lset mylist4 0 "four"  
OK  
redis 127.0.0.1:6379> lset mylist4 -2 "five"  
OK  
redis 127.0.0.1:6379> lrange mylist4 0 -1  
1) "four"  
2) "five"  
3) "three"  
redis 127.0.0.1:6379>

　　在此处我们依次插入了one,two,three，然后将标是0的值设置为four，再将下标是-2的值设置为five。

**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：

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

　　count<0时，按从尾到头的顺序删除，具体如下：

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

　　count=0时，删除全部，具体如下：

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

**6、ltrim**

　　保留指定key 的值范围内的数据：

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

**7、lpop**

　　从list的头部删除元素，并返回删除元素：

redis 127.0.0.1:6379> lrange mylist 0 -1  
1) "hello"  
2) "world"  
redis 127.0.0.1:6379> lpop mylist  
"hello"  
redis 127.0.0.1:6379> lrange mylist 0 -1  
1) "world"  
redis 127.0.0.1:6379>

**8、rpop**

　　从list的尾部删除元素，并返回删除元素：

redis 127.0.0.1:6379> lrange mylist2 0 -1  
1) "hello"  
2) "world"  
redis 127.0.0.1:6379> rpop mylist2  
"world"  
redis 127.0.0.1:6379> lrange mylist2 0 -1  
1) "hello"  
redis 127.0.0.1:6379>

**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：

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

**10、lindex**

　　返回名称为key的list中index位置的元素：

redis 127.0.0.1:6379> lrange mylist5 0 -1  
1) "three"  
2) "foo"  
redis 127.0.0.1:6379> lindex mylist5 0  
"three"  
redis 127.0.0.1:6379> lindex mylist5 1  
"foo"  
redis 127.0.0.1:6379>

**11、llen**

　　返回key对应list的长度：

redis 127.0.0.1:6379> llen mylist5  
(integer) 2  
redis 127.0.0.1:6379>

4）sets

Redis的set是string类型的无序集合。set元素最大可以包含(2的32次方)个元素。

　　set的是通过hash table实现的，所以添加、删除和查找的复杂度都是O(1)。hash table会随着添加或者删除自动的调整大小。需要注意的是调整hash table大小时候需要同步(获取写锁)会阻塞其他读写操作，可能不久后就会改用跳表(skip list)来实现，跳表已经在sorted set中使用了。关于set集合类型除了基本的添加删除操作，其他有用的操作还包含集合的取并集(union)，交集(intersection)，差集 (difference)。通过这些操作可以很容易的实现sns中的好友推荐和blog的tag功能。下面详细介绍set相关命令：

**1、sadd**

　　向名称为key的set中添加元素：

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

　　本例中，我们向myset中添加了三个元素，但由于第三个元素跟第二个元素是相同的，所以第三个元素没有添加成功，最后我们用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：

　　redis 127.0.0.1:6379> sadd myset2 "one"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> sadd myset2 "two"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> sadd myset2 "three"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> srem myset2 "one"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> srem myset2 "four"  
  
　　(integer) 0  
  
　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379>

　　本例中，我们向myset2中添加了三个元素后，再调用srem来删除one和four，但由于元素中没有four所以，此条srem命令执行失败。

**3、spop**

　　随机返回并删除名称为key的set中一个元素：

　　redis 127.0.0.1:6379> sadd myset2 "one"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> sadd myset2 "two"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> sadd myset2 "three"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> srem myset2 "one"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> srem myset2 "four"  
  
　　(integer) 0  
  
　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379>

　　本例中，我们向myset3中添加了三个元素后，再调用spop来随机删除一个元素，可以看到three元素被删除了。

　　4、sdiff

　　返回所有给定key与第一个key的差集：

　　redis 127.0.0.1:6379> sadd myset2 "one"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> sadd myset2 "two"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> sadd myset2 "three"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> srem myset2 "one"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> srem myset2 "four"  
  
　　(integer) 0  
  
　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379>

　　本例中，我们可以看到myset2中的元素与myset3中不同的只是three，所以只有three被查出来了，而不是three和one，因为one是myset3的元素。

　　我们也可以将myset2和myset3换个顺序来看一下结果：

　　redis 127.0.0.1:6379> sdiff myset3 myset2  
  
　　1) "one"  
  
　　redis 127.0.0.1:6379>

　　这个结果中只显示了，myset3中的元素与myset2中不同的元素。

**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：

　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379> smembers myset3  
  
　　1) "two"  
  
　　2) "one"  
  
　　redis 127.0.0.1:6379> sdiffstore myset4 myset2 myset3  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> smembers myset4  
  
　　1) "three"  
  
　　redis 127.0.0.1:6379>

**6、sinter**

　　返回所有给定key的交集：

　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379> smembers myset3  
  
　　1) "two"  
  
　　2) "one"  
  
　　redis 127.0.0.1:6379> sinter myset2 myset3  
  
　　1) "two"  
  
　　redis 127.0.0.1:6379>

　　通过本例的结果可以看出, myset2和myset3的交集two被查出来了。

**7、sinterstore**

　　返回所有给定key的交集，并将结果存为另一个key

　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379> smembers myset3  
  
　　1) "two"  
  
　　2) "one"  
  
　　redis 127.0.0.1:6379> sinterstore myset5 myset2 myset3  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> smembers myset5  
  
　　1) "two"  
  
　　redis 127.0.0.1:6379>

　　通过本例的结果可以看出, myset2和myset3的交集被保存到myset5中了

**8、sunion**

　　返回所有给定key的并集

　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379> smembers myset3  
  
　　1) "two"  
  
　　2) "one"  
  
　　redis 127.0.0.1:6379> sunion myset2 myset3  
  
　　1) "three"  
  
　　2) "one"  
  
　　3) "two"  
  
　　redis 127.0.0.1:6379>

　　通过本例的结果可以看出, myset2和myset3的并集被查出来了

**9、sunionstore**

　　返回所有给定key的并集，并将结果存为另一个key

　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379> smembers myset3  
  
　　1) "two"  
  
　　2) "one"  
  
　　redis 127.0.0.1:6379> sunionstore myset6 myset2 myset3  
  
　　(integer) 3  
  
　　redis 127.0.0.1:6379> smembers myset6  
  
　　1) "three"  
  
　　2) "one"  
  
　　3) "two"  
  
　　redis 127.0.0.1:6379>

　　通过本例的结果可以看出, myset2和myset3的并集被保存到myset6中了

**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

　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "three"  
  
　　2) "two"  
  
　　redis 127.0.0.1:6379> smembers myset3  
  
　　1) "two"  
  
　　2) "one"  
  
　　redis 127.0.0.1:6379> smove myset2 myset7 three  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> smembers myset7  
  
　　1) "three"  
  
　　redis 127.0.0.1:6379>

　　通过本例可以看到，myset2的three被移到myset7中了

**11、scard**

　　返回名称为key的set的元素个数

　　redis 127.0.0.1:6379> scard myset2  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379>

　　通过本例可以看到，myset2的成员数量为1

**12、sismember**

　　测试member是否是名称为key的set的元素

　　redis 127.0.0.1:6379> smembers myset2  
  
　　1) "two"  
  
　　redis 127.0.0.1:6379> sismember myset2 two  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> sismember myset2 one  
  
　　(integer) 0  
  
　　redis 127.0.0.1:6379>

　　通过本例可以看到，two是myset2的成员，而one不是。

**13、srandmember**

　　随机返回名称为key的set的一个元素，但是不删除元素

　　redis 127.0.0.1:6379> smembers myset3  
  
　　1) "two"  
  
　　2) "one"  
  
　　redis 127.0.0.1:6379> srandmember myset3  
  
　　"two"  
  
　　redis 127.0.0.1:6379> srandmember myset3  
  
　　"one"  
  
　　redis 127.0.0.1:6379>

5）sorted sets

和set一样sorted set也是string类型元素的集合，不同的是每个元素都会关联一个double类型的score。sorted set的实现是skip list和hash table的混合体。

　　当元素被添加到集合中时，一个元素到score的映射被添加到hash table中，所以给定一个元素获取score的开销是O(1),另一个score到元素的映射被添加到skip list，并按照score排序，所以就可以有序的获取集合中的元素。添加，删除操作开销都是O(log(N))和skip list的开销一致,redis的skip list实现用的是双向链表,这样就可以逆序从尾部取元素。sorted set最经常的使用方式应该是作为索引来使用.我们可以把要排序的字段作为score存储，对象的id当元素存储。下面是sorted set相关命令

**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

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

　　本例中我们向myzset中添加了one和two，并且two被设置了2次，那么将以最后一次的设置为准，最后我们将所有元素都显示出来并显示出了元素的score。

**2、zrem**

　　删除名称为key的zset中的元素member

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

　　可以看到two被删除了

**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

　　redis 127.0.0.1:6379> zadd myzset2 1 "one"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> zadd myzset2 2 "two"  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> zincrby myzset2 2 "one"  
  
　　"3"  
  
　　redis 127.0.0.1:6379> zrange myzset2 0 -1 withscores  
  
　　1) "two"  
  
　　2) "2"  
  
　　3) "one"  
  
　　4) "3"  
  
　　redis 127.0.0.1:6379>

　　本例中将one的score从1增加了2，增加到了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

　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　7) "five"  
  
　　8) "5"  
  
　　redis 127.0.0.1:6379> zrank myzset3 two  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379>

　　本例中将two的下标是1，我这里取的是下标，而不是score

**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

　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　7) "five"  
  
　　8) "5"  
  
　　redis 127.0.0.1:6379> zrank myzset3 two  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379>

　　按从大到小排序的话two是第三个元素，下标是2

**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

　　redis 127.0.0.1:6379> zrevrange myzset3 0 -1 withscores  
  
　　1) "five"  
  
　　2) "5"  
  
　　3) "three"  
  
　　4) "3"  
  
　　5) "two"  
  
　　6) "2"  
  
　　7) "one"  
  
　　8) "1"  
  
　　redis 127.0.0.1:6379>

　　首先按score从大到小排序，再取出全部元素

**7、zrangebyscore**

　　返回集合中score在给定区间的元素

　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　7) "five"  
  
　　8) "5"  
  
　　redis 127.0.0.1:6379> zrangebyscore myzset3 2 3 withscores  
  
　　1) "two"  
  
　　2) "2"  
  
　　3) "three"  
  
　　4) "3"  
  
　　redis 127.0.0.1:6379>

　　本例中，返回了score在2~3区间的元素

**8、zcount**

　　返回集合中score在给定区间的数量

　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　7) "five"  
  
　　8) "5"  
  
　　redis 127.0.0.1:6379> zcount myzset3 2 3  
  
　　(integer) 2  
  
　　redis 127.0.0.1:6379>

　　本例中，计算了score在2~3之间的元素数目

**9、zcard**

　　返回集合中元素个数

　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　7) "five"  
  
　　8) "5"  
  
　　redis 127.0.0.1:6379> zcard myzset3  
  
　　(integer) 4  
  
　　redis 127.0.0.1:6379>

　　从本例看出myzset3这个集全的元素数量是4

**10、zscore**

　　返回给定元素对应的score

　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　7) "five"  
  
　　8) "5"  
  
　　redis 127.0.0.1:6379> zscore myzset3 two  
  
　　"2"  
  
　　redis 127.0.0.1:6379>

　　此例中我们成功的将two的score取出来了。

**11、zremrangebyrank**

　　删除集合中排名在给定区间的元素

　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　7) "five"  
  
　　8) "5"  
  
　　redis 127.0.0.1:6379> zremrangebyrank myzset3 3 3  
  
　　(integer) 1  
  
　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　redis 127.0.0.1:6379>

　　在本例中我们将myzset3中按从小到大排序结果的下标为3的元素删除了。

**12、zremrangebyscore**

　　删除集合中score在给定区间的元素

　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "one"  
  
　　2) "1"  
  
　　3) "two"  
  
　　4) "2"  
  
　　5) "three"  
  
　　6) "3"  
  
　　redis 127.0.0.1:6379> zremrangebyscore myzset3 1 2  
  
　　(integer) 2  
  
　　redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores  
  
　　1) "three"  
  
　　2) "3"  
  
　　redis 127.0.0.1:6379>

　　在本例中我们将myzset3中按从小到大排序结果的score在1~2之间的元素删除了。