Redis 笔记

Redis 的底层结构分为：SDS，连表，字典(字典里是hash有2个表)，跳跃表，整数集合，压缩列表

第8章对象

注意：type返回的是数据库健对应值对象的类型，而不是健对象的类型。Redis的健总是一个字符串对象，而值可以是字符串对象、列表对象、哈希对象、集合对象或有序集合对象。

使用OBJECT ENCODING key_name是返回一个数据库健的值对象的编码。

字符串对像

OBJECT ENCODING key_name

字符串对像的编码可以是int、raw或embstr。

int：如果一个字串对像保存的是整数值，并这个数值可以用Longo类型来表示，那么字符串对像会将整数值保存在字符串对像结构的ptr属性里并把字符串对像的编码设置为int。如：SET story 10086这时用OBJECT ENCODEING查看是Int。

raw：如果字符串对像保存的是一个字符串值，并这个值长度大于32字节，那么字符串对像将使用一个简单动态字符串SDS来保存这个值，并将对像的编码设置为raw。如：SET story hello word hi oh my gad。

embstr：如果字符串对像保存的是一个字符串值，并这个字符串值的长度小于等于32字节，那么字符串对像将使用embstr编码的方式保存。如：SET story hello

embstr是只读的，更改就会变raw,是专门用于保存短字符串的一种优化编码方式，这么编码和raw编码一样，都使用redisObject结构和sdshdr结构来表示字符串对像，但raw编码会调用两次mwdhwv配函数来创建，而embstr则通过一次内存分配函数来分配一块连续的空间。embstr编码的字符串对像在执行命令时产生的效果和raw的对像是一样的，但使用embstr保存短字符带来以下好处：

1.embstr内存分配次数从raw编码的两次降为1次。

2.释放embstr时只用一次内存释放，而raw要两次。

3.因为embstr所有数据都保存在一块连续的的内存里，所以能更好的利用缓存带来的优势。

如long double 在Redis里也是字符串，保存一个浮点数到字符串对像里，程序会先将浮点数换在字符串，然后再保存转换所得的字符串值。如SET pi 3.21它是用embstr保存的。如果是INCRBYFLOAT pi 2.0这时就会是5.14这时候是先把保存在字符串对像里的字符串值换回浮点数值，执行操作然后再把操作所得的浮点值换回字twfk串，并继续保存在字符串对像里。

可以用long类型保存的整数int,可以用long double类型保存的浮点数embitter或raw，字符串值，或因为长度太大而没法用long类型表示的整数，或因为长度太大没法用long duble类型表示的浮点数embstr或raw

列表对像

列表对像的编码可以是ziplist或linkedlist

ziplist编码的列表对象使用压缩列表作为底层实现。

RPUSH number 1 "three" 5如果number健的值对像是用zippiest编码，就是如下：

redisObject
type REDIS_LIST
encoding REDIS_ENCODING_ZIPLIST
ptr->zlbytes|zltail|zllen|1|"three"|5|zlend
另一方面，linkedlist编码的列表对像使用双向链表作为底层实现，每个双端链表节点都保存了一个字符串对像，而每个字符串对像都保存了一个列表元素。如果用上面的例子来讲，如果用的是linkedlist就是：
redisObject
type REDIS_LIST
encoding REDIS_ENCODING_ZIPLIST
ptr--> stringObject 1—>stringObject "three"—>stringObject 5
注意linkedlist编码的列表对像在底层的双端链表的双端链表结构中包含了多个字符串对像，这种嵌套字符串对像的行为在哈希对像、集合对像和有序集合对像中都会出现，字符串对像是Redis五种类型的对像中唯一一种会被其它四种类型对像嵌套的对像。上面的stringObject three是以下的完整版：
redisObject |
sdshdr |
type REDIS_STRING | encodeing REDIS_ENCODING_EMBSTR | prt | …| free 0 | len 5 | buf |t|h|r|e|e|\0|

编码转换

满足下面两个条件时列表对象会使用ziplist编码如果不满足就会使用linkedlist：
1.列表对象保存的所有字符串元素长度小于64字节。
2.列表对象保存的元素数量小于512个。
以上两个值是可以修改的，可以在配置文件中list-max-ziplist-value和list-max-ziplist-entries进行修改。当使用ziplist编码所要的两个条件其中一个不满足就会被转换，原来保存在压缩列表里的所有列表元素都会被转移并保存到双向连表里，对像的编码也会从ziplist变为lindedlist。

哈希对象

哈希对象的编码可以是ziplist或hashtable。
ziplist当有新的健值要加入哈希对象时，会先保存了健的压缩列表节点推入到压缩列表的表尾，然后把保存了值的压缩列表节点推入到压缩列表的尾部。所以保存了同一健值对的两个节点总是紧挨在一起，保存健的节点在前，保存值的节点在后。而先添加到哈希对象中的健值会被放在压压缩列表的表头方向，而后来添加到哈希对象中的健值对被放在压缩列表的表尾。

编码转换

哈希对象当同时满足以下两个条件时，会使用ziplist编码
1.哈希对象保存的所有健值对的健和值的字条串长度都小于64字节；
2.哈希对象的保存的健值对数量小于512个；

如果不满足这两个条件，就会使用hashtable编码。根据hash-max-ziplist-value和hash-max-ziplist-entries可以根据这两个参数来设置。
集合对象
集合对象的编码可以是intset或hashtable
intset编码的集合对象使用整数集合作为底层实现，集合对象包含的所有元素都被保存在整数集合里。如SADD 1 3 5是intset。如图:
ptr|—>intset列表里会有个contents | —>1 3 5
如果是SADD fruits "apple" "banana" "cherry"这就是hashtable如图:
ptr—>dict列表里面全是stringObject cherry-> stringObject apple->stringObject banana……
编码转换
当集合对象可以同时满足以下两个条件时，对象使用intset编码:
1.集合对象保存的所有元素都是整数值
2.集合对象保存的元素数量不超过512个
如果不满足就会使用hashtable，第二个上限的值可以用set-max-inset-entries来配置。
有序集合对象
有序集合的编码可以是ziplist或skiplist
ziplist的压缩列表对象使用压缩列表作为底层实现，每个元素挨在一起，第一个节点保存元素成员(member)，第二个保存元素的分值(score)。
压缩列表内的集合按分值从小到大进行排序，分小的被放置在靠近表头的方向。分大的相反。如:
ZADD price 8.5 apple 5.0 banana 6.0 cherry如果是zippiest那么如下:
ptr—>压缩列表("banana" 5.0 | "cherry" 6.0 | "apple" 8.5)
编码转换
当有序集合对象可以同时满足以下条件时就会使用ziplist编码:
1.有序集合保存的元素量小于128个。
2.有序集合保存的所有元素成员的长度都小于64字节。
如果不能满足以上将使用skiplist编码。zset-max-ziplist-entries和zset-max-ziplist-value可以控制以上的两个参数。
类型检查与命令多态
Redis中有两种类型的命令第一种是对任何类型的健执行。如：DEL、EXPIRE、RENAME、TYPE和OBJECT等。
而另一种就只能对特定的健执行比如:
SET、GET、APPEND、STRLEN只能对字符串健执行
HDEL、HSET、HGET、HLEN只能对哈希健执行
RPUSH、LPOP、LINSERT、LLEN只能对列表执行
SADD、SPOP、SINTER、SCARD等只能对集合健执行
ZADD、ZCARD、ZRANK、ZSCORE……
内存回收：对象的引用计数器信息会随着对象的使用状态而不断变化:

1.在创建一个新对象时，引用计数器的值会被初始为1
2.当对象对一个新程序使用时，它的引用计数值为被+1
3.当对象不再被一个程序使用，它的引用数值会被-1
4.当对象的引用计数值为0时，对象所占用的内存就会被释放
对象享：假设A创健了一个包含整数值100的字符串值作为对象，如果B也创健了一个同样的数值100字符串对象，那么服务器会有以下两种做法：
1.为健B新创健一个包含整数数值100的字符串对象
2让健A和健B共享一个字符串对象
这两种啃定是第二种比较好，在Redis中让多个健共享一个值对象要执行以下两个步骤：
1.将数据库健的值批值指向一个现有的值对象
2.将被共享的值对象的引用计数增1
OBJECT REFCOUNT key_name
如果数据库中保存了数值100的健不只有A和B而是有一百个，那么就会节约了很多空间。目前Redis会在初始化时创健一万个字符串对象，包含了从0到9999的所有整数值，当服务器需要用到0到9999的对象时就会用这些共享对象，而不创健。比如：我们创健一个值为100的健A并用OBJECT REFCOUNT来看就会看到它的值对象引用的计数。
OBJECT IDLETIME key_name：对象的空转时长：除了type、encoding、ptr和refcount四个属于外，还有一个包含最后一个redisObject属性，它记录了对象最后一次被程序访问的时间。除了被OBJECT IDLETIME命令打印出来，健的空转时长还有另外一项作用：如果服务器打开了maxmemory选项，并服务器用于回收内存的算法为volatile-lru或allkeys-lru那么当服务器占用的内存数超过了maxmemory选项所设置的上限值时，空转时间较高的那部分健会优先被服务器释放，从而回收内存
配制文件的maxmemory选项和maxmemory-policy选项的说明介绍了关于这的信息。
其它健空间操作
FLUSHDB 删除健空间中的所有健值对、RANDOMKEY 随机反回一个健、DBSIZE 返回健空间中健值对的数量，还有EXISTI、RENAME、KEYS等这些都是能过对健空间进行操作来实现的。
读写健空间时的维护操作
当使用Redis命令对数据库进行读写时，服务器不仅会对健空间执行指定的操作，还会执行一些额外的维护操作，包括：
1.读取一个健后(计和写操作都要对健进行读取)服务器会根据是否存在来更新服务器健空间的命中(hit)和不命中(miss)次数可keyspace_hits和keyspace_misses属性中查看。
2.读取一个健后服务器会更新新健的LUR时间，这个值是OBJECT IDLETIME查看的。
3.如果发现过期，那么会先删除过期健然后再其它操作
4.如果客户端使用WATCH监视了某一健，那么在对被监视健进行修改后会将这个健标为脏，从而让事务注意到它
5.服务器每修改一个健都会对脏健计数器的值+1，这个计数器会触发服务器的持久化以及复制操作

6.如果服务器开了数据库通知功能，那么在健进行修改后，服务器将按配置发送相应的数据库通知
设置过期时间
SETEX:能财时对字符串进行设置 SETEX shixiaowen 5 hello设置过期为5秒
EXPIRE key ttl 将健key的生存时间设置为ttl 秒
PEXPIRE key ttl 将健key生存时间设置为毫秒
EXPORT key timestamp 设置健过期时间为timestamp所指定的秒数时间戳
PERSIST key可以移除一个健的过期时间，它是PEXPIREAT命令的反操作
TTL 返回健过期时间差秒，PTTL返回的也是过期时间但是单位为毫秒。
Redis过期健策略有
定时删除：在设置健过期时同时创健一个定时器。缺点占用CPU时间。
惰性删除：每次获取健时再进行删除。缺点占用内存，因为过期后要读取操作才能释放。
定期删除：每隔一段时间程序就会对数据库进行一次检查。是折中了以上两点。
Redis使用了惰性删除和定期删除两种策略。这样服务器可以很好地配合使用CPU时间和避免内存浪费之间取得平衡。
notify-keyspace-events选项决定了服务器所发送通知的类型：
1.想让服务器发送所有类型的健空间通知和健事件通知，可以将选项的值设为AKE
2.想让服务器发送所有类型的健空间通知，可以把选项的值设为AK
3.想让服务器发送所有类型的健事件通知，可以把选项值设为AE
4.想让服务器发送和字符串健有关的健空间通知，可以设为KS
5.想让服务器发送所和列表健有关的健事件通知，可设为El
Redis内存数据集大小上升到一定大小时，就会施行数据淘汰策略。在配置文件里的配制如下：
volatile-lru:从已设置过期时间的数据集(server.db[i].expires)中选择最近最少使用的数据淘汰。
volatile-ttl:从已设置过期时间的数据集(server.db[i].expires)中选择将要过期的数据进行淘汰。
volatile-random:从已设置过期时间的数据集(server.db[i].expires)中任意选择进行淘汰。
allkeys-lru:从数据集(server.db[i].expires)中选择最近最少使用的数据淘汰。
alleys-random:从数据集(server.db[i].expires)中任意选择数据淘汰。
no-enviction:禁止驱逐数据。

持久化

RDB持久化
RDB文件的创健与载入，SAVE和BGSAVE可以生成RBD文件，前者会阻塞客户端的请求，后者不会，后者会派生一个子进程然后去创建。RDB的载入是在服务器启动时检测如果有它会自动载入。因为AOF文件的更新频率通常比RDB文件的更新频率高所以：
1.如果开启AOF时，服务器会优先使用AOF文件来还原数据。
2.只有在AOF持久化功能处于关闭时，服务器才会用RDB来还原数据。
SAVE BGSAVE BGREWRITEAOF(也是用子进程)这几个命令不能同时进行。

dirty计数器和last save属性
除了saveparams数组外，服务器状态还维持着一个dirty计数器，和一个lastsave属性：
1.dirty计数器记录距离上一次成功执行SAVE命令或BGSAVE命令后服务器对数据库状态(服务器中的所有数据库)进行了多少次修改(包括写入、删除、更新等)
2.lastsave属性是一个UNIX时间戳，记录了服务器上次成功执行SAVE命令或BGSAVE的时间。
如SET masseg "hello" 这时dirty计数器会加1,如果向一个集合加3个元素那么dirty会加3
RDB 文件结构(RDB文件保存的是二进制)
|REDES | db_version | databases0 | databases1 | EOF | check_sum
5字节 版本号 数据库如果为空则没有 载入RDB时会将载入数据所计算的检验和它进行对比，看看是否有损坏的情况出现。
AOF持久化
RDB是以保存数据库的健值对来记录的，而AOF是通过保存服务器所执行的写命令来记录数据库状态的。
AOF的是以命令追加(append)、文件写入、文件同步(sync)三个步骤来实现的。
命令追加：如SET KEY VALUE执行命令后会把内容追加到sof_buf缓冲区中的末尾。
AOF文件的写入与同步
AOF服务器进程就是一个事件循环loop，这个循环中的文件事件负责接收客户端的命令请求，以及向客户端发送命令回复，而时间事件则负责像serverCron函数这样需要定时运行的函数。因为服务器在处理文件事件时可能会执行写命令，使得一些内容被追加到aof_buf缓冲区里，所以在服务器每次结束一个事件循环之前，都会调用flushAppendOnlyFile函数，考虑是否要把aof_buf缓冲区的内容写入和保存到AOF文件里面。
appendfsync：控制函数flushAppendOnlyFile的行为。参数有：
always:将aof_buf缓冲区中的所有内容写入并同步到AOF文件。 效率最低，但是最保险
everysec:将aof_buf缓冲区中的所有内容写入到AOF文件，如果上次同步AOF文件的时间距离现超过一秒钟，那么再次对AOF文件进行同步，并这个同步操作是由一个线程专门负责执行。 比上一个好点，就算宕机也只丢1秒
no:将aof_buf缓冲区的所有内容写到AOF文件，但不对AOF文件进行同步，何时同步由操作系统决定。
AOF的文件载入与数据还原：
因为里面全是所有写的命令，所以只要读入并重新执行一下就可以，步骤如下：
服务器启动—>创健为客户端—>从AOF文件中分析并取出一条写命令—>使用伪客户端执行写命令—>AOF文件中的所有写命令都OK—>（载入完毕）如果没有OK那再回到—>从AOF文件中分析并取出一条写命令
伪客户端(fake client)：因为Redis只能在客户端上下文中执行，而载入AOF文件时使用的命令直接来源于AOF文件而不是网络连接，所以服务器使用了一个没有网络连接

的伙客户端来执行AOF文件保存的命令，和客户端执行的命令是一样的。

AOF重写原理

因为AOF是记的命令比如rpush list 里写了很多数据就会很多条，文件也会很大AOF重写就是要整理这些多余的命令。

AOF重写并不是在原来的AOF文件里做操作，是获取现有数据库的所有状态来进行操作的。为了不阻塞客户端的请求Redis会派生出来一个子进程进行来进行重写，写完后再通知父进程，这时候如果客户端有命令在写入有可能会造成数据不一致所以在重写的时候会同时往AOF缓冲区和AOF重写缓冲区进行命令的追加来保证数据的一致性。

客户端

Redis是典型的一对多的服务程序，一个服务器可以与多个客户端建立网络连接，每个客户端可以向服务器发送命令请求，而服务器则接收并处理客户端发送的命令请求，并向客户端返回命令回复。通过IO多路复用技术实现的文件事件处理器，它使用单线程单进程的方式来处理命令请求，并与多个客户端进行网络通信。

客户端属性分为：

1.通用属性 这些属性很少与特定功能相当，无论客户端执行的是什么工作，它们都要用到这些属性。

2.特定功能属性 如操作数据库时要用的db属性和dictid属性，执行事务时要用到的mstate属性，和执行WATCH命令要用到的watched_keys属性等。

WATCH：watch命令可以监控一个或多个键值的变化，一旦其中一个键被改变，之后的事务就不会执行，而且监控会一直持续到exec命令。如下

```
127.0.0.1:6379> set key 1
OK
127.0.0.1:6379> watch key
OK
127.0.0.1:6379> set key 2
OK
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set key 3
QUEUED
127.0.0.1:6379> exec
(nil)
127.0.0.1:6379> get key
“2” 值还是2并不是3
```

CLIENT list:可以列出目前所有连接到服务器的普通客户端如下：

```
addr=10.132.33.219:35848 fd=40 name= age=0 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 events=r cmd=auth
addr=10.160.4.245:38448 fd=42 name= age=0 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 events=r cmd=auth
```

fd：记录了客户端正在使用的套接字描述符，值可以是-1或大于-1的整数。-1为伪客户端(fake client)来原于AOF文件或者Lua脚本，而不是网络，所以不需要套接字连接。目前redis服务器会在两个地方用到伪客户端(fake client)一个是载入AOF文件并还原数据库状态，另一个则用于执行Lua脚本中包含redis命令。大于-1的就是正常的套接字连接。

name：在默认下一个连接到服务器的客户端是没有名字的，可以为自己的连接设置一个名字CLIENT setname my_name就可以了。

idle：客户端最后一次与服务器互动的空转时间。
输入缓冲区：记录了客户端发送的命令请求，缓冲区大小不能超过1GB.
输出缓冲区：有二个，一个是固定大小，一个是可变。
固定的是保存那些简短的的字符串值、整数值、错误回复等。
可变大小缓冲区用于保存那些较大的回复，如一个很长的字符串值，一个由很多项组成的列表，一个包含了很多元素的集合等
客户端的固定大小缓冲区：是由buf和bufpos两个属性组成。buf是一个大小为REDIS_REPLY_CHUNK_BYTES字节的字节数组，而bufpos则记录的buf数据目前已使用的字节数量。REDIS_REPLY_CHUNK_BYTES默认值为16*1024也就是说buf数组的默认值为16KB。
可变大小缓冲区由reply链表和一个或多个字符串对象组成，通过链表来连接多个字符串对象，服务器可以为客户端保存一个非常长的命令回复，而不用受到固定大小缓冲区16KB的限制。
客户端的创建与关闭
创建普通客户端：如C1和C2两个普通客户端正在连接服务器，有一个新的C3连接后，服务器会把C3所对应的客户端加到clients链 表的尾部。c1->c2->c3
关闭普通客户端会有很多原因被关闭如下：
1.如果客户端进程退出或被杀死，那客户端和服务器间的网络连接会被关闭。
2.如果客户端向服务器发送了带有不符合的协议格式的命令请求，那这个客户端也会被服务器关闭。
3.如果客户端成为了CLIENT KILL命令的目标，那么也会被关闭。
4.如果用户为服务器设置了timeout配置选项，那当客户端的空转时间超过了timeout的值时，也会被关。不过timeout也有一些例外。如：客户端是主服务器(打开了REDIS_MASTER标志)，从服务器(打开了REDIS_SLAVE标志)，正在被BLPOP命令阻塞(打开了REDIS_BLOCKED标志)或正在执行SUBSCRIBE、PSUBSCRIBE等订阅命令，那么就算客户端的空转时间超过了timeout的值，也不会被服务器关闭。
5.如果客户端发送的命令请求大小超过了输入缓冲区的限制大小(默认为1GB)，也会被关闭。
6.如果要发送给客户端的命令回复的大小超过了输出缓冲区的限制大小，也会被服务器关闭。
前面介绍输出缓冲区提到可变大小缓冲区由一个链表和任意多个字符串对象组成，理论上来说，这个缓冲区可以保存任意长的命令回复，但是为了避免客户端的回复过大，占用太多服务器资源，服务器会时间检查客户端的输出缓冲区的大小，并在缓冲区的大小超出范围时执行相应操作。服务器使用两种模式来限制客户端输出缓冲区的大小：
1.硬性限制(hard limit)：如果输出缓冲区的大小超过了硬性限制所设置的大小，那么服务器会立即关闭客户端。
2.软性限制(soft limit)：如果输出缓冲区的大小超过了软性限制所设置的大小，但还没超过硬性限制，那么服务器使用客户端状态结构的obuf_soft_limit_reached_time属性记录下客户端到达软限制的起始时间；之后服务器会继续监视客户端，如果输出缓冲区的大小一直招出软性限制，并持续时间超过服务器设定的时长，那么服务器将关闭客户端；相反地，如果输出缓冲区的大小在指定时间之内，不再超出软性限制，那么客户端就不会被关闭。并obuf_soft_limit_reached_time属性的值也会被清零。使用client-output-buffer-limit选项可以为普通客户端、从服务器客户端、执行发布与定

阅功能的客户端分别设置不同的软性限制和硬性限制，格式为(client-output-buffer-limit<class> <hard limit> <soft limit> <soft seconds>)如下：

```
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60
```

第一行设置将普通客户端的感性限制和软性限制都设置为0，表示不限制客户端的输出缓冲区大小。
第二行设置将从服务器客户端的硬性限制设置为256MB，而软性限制为64MB，软性限制时间为60秒。
第三行设置执行发布与订阅功能的客户端的感性限制为32MB，软性为8MB，软性限制时长为60秒。

重点回顾：

服务器状态结构用clients链表连接起多个客户端状态，新加的会放到链表尾部。
命令的参数和参数个数会被记录在客户端状态的argv和argc属性里，而cmd属性则记录了客户端要执命令的函数。
客户端有固定大小缓冲区和可变大小缓冲区两种缓冲区可用，其中固定大小缓冲区最大大小为16KB，而可变大小缓冲区的最大大小不能超过服务器的硬性限制值。
输出缓冲区限制值有两种，如果输出缓冲区大小超过了服务器设置的硬性限制，那么客户端会被立即关闭；除此之外，如果客户端在一定时间内，一直超过服务器设置的软性限制，那么客户端也会被关闭。
当一个客户端通过网络连接连接上服务器时，服务器会为这个客户端创健相应的客户端状态。网络连接关闭、发送了不合协议格式的命令请求、成为CLIENT KILL命令的目标、空转时间超时、输出缓冲区的大小超出限制，这些原因都会造成客户端被关闭。
处理Lua脚本的伪客户端在服务器初始化创建，这个客户端会一直存在，直到服务器关闭。
载入AOF文件时使用的伪客户端在载入工作开始时动态创建，载入工作完毕后关闭。

服务器

在执行命令执行时(预备操作)
如果maxmemory开启，在执行命令前会先检查服务器内存占用情况，并在有需要时进行内存回收，如果回收失败那么不再执行后续步骤，向客户端返回一个错误。
如果服务器上一次执行BGSAVE命令时出错，并打开了stop-writes-on-bgsave-error，而且服务器即将要执行的命令是一个写命令，那么服务器将拒绝执行这个命令，并向客户端返回一个错误。
如果客户端正在用SUBSCRIBE命令订阅频道，或正在用PSUBSCRIBE命令订阅模式，那么服务器只会执行客户端发来的SUBSCRIBE、PSUBSCRIBE、UNSUBSCRIBE、PUNSUBSCRIBE四个命令，其它命令都会被服务器拒绝。
如果服务器正在进行数据载入，那客户端发送的命令必须带有l标识(比如INFO、SHUTDOWN、PUBLISH等)才会被服务器执行，其他命令都会被服务器拒绝。
如果服务器因为执行Lua脚本而超时，并进入阻塞状态，那么服务器只会执行客户端发来的SHUTDOWN nosave命令和SCRIPT KILL命令。
如果客户端正在执行事务，那么服务器只会执行客户端发来的EXEC、DISCARD、MULTI、WATCH四个命令。
如果服务器打开了监视功能，那么服务器会将要执行的命令和参数等信息发送给监视器。当完成以上预备操作后，服务器就可以开始真正的执行命令了。

以上是列出了单机服务器在执行命令时的检查操作，如果在复制或集群模式下预备操作会更多。
instantaneous_ops_per_sec:在最近的一秒种内，服务器大约处理的命令。
of_rewrite_scheduled:记录了服务器是否延迟了BGREWRITEAOF命令。在服务器执行BGSAVE命令期间，如果客户端向服务器发来BGREWRITEAOF命令，那么服务器将BGREWRITEAOF命令的执行时间延迟到BGSAVE命令执行完毕后。
Redis复制
主从是从往主上发SLAVEOF进行第一次同步生成RDB文件再返回（BGSAVE）。2.8前如果断线后再连上会再次生成RDB再返回。在2.8后有了PSYNC进行部份同步，部份同步由三个部分结成：
1.主服务器的复制偏移量(replication offset)和从服务器的复制偏移量。
2.主服务器的复制积压缓冲区(replication backlog)。默认为1MB是一个先进先出的队列，每次同步都会写这里面一次。
3.服务器的运行ID(run ID)。
如果断线后从服会发送PSYNC命令，主服会在接收到的偏移量来进行同步。如果数据在积压缓冲里就进行部份同步，如果不在就为全量同步。
repl-lacklog-size:控制复制积压缓冲区的大小，为了安全起见把积压缓冲区的大小设为2*second*write_size_per_second。
min-slaves-to-write 3
min-slaves-max-lag 10
以上两个选项如果在master设置后，意思就是在从服务器的数量少于3个，或三个从服务器的延迟lag值都大于或等于10秒时，主服务器将拒绝执行写命令，这里的延迟值就是INFO replication命令的lag值。
集群
Redis集群是Redis提供的分布式数据库方案，集群通过分片来进行数据共享，并提供复制和故障转移功能。
节点:一个redis集群通过由多个节点node组成，
CLUSTER MEET：用来串连起各个节点。如：CLUSTER MEET <ip> <port>
CLUSTER NODE：可以查看集群所包含的所有节点。
CLUSTER REPLICATE <node_id>：向一个节点发送它可以让接收命令的节点成为node_id所指定节点的从节点，并开始对主节点进行复制。
启动节点
cluster-enabled：redis会根据这个配置来决定是否开启服务器的集群模式。
槽指派
redis集群通过分片的方式来保存数据库中的健值对：集群的整个数据库被分为16384个槽(slot)，数据库中的每个健都属于这16384个槽的其中一个，集群中的每个节点可以处理0个或最多16384个槽。
当数据库中的16384个槽都有节点在生理时，集群处于上线状态(ok)。相反地，如果数据库中没有任何一个槽没有得到处理，那么集群处于下线状态(fail)。
CLUSTER INFO：可以看见很多槽信息。
CLUSTER ADDSLOTS：可以将一个或多个槽指派给节点负责。如：CLUSTER ADDSLOTS 0 1 2 ……
CLUSTER NODES：可以显示出所有节点

slots属性是一个二进制位数组。这个数组的长度为16384/8=2048个字节，共包含16384个二进制位。
redis以0为起始索引16384为终止索引，如果slots数组在索引i上的二进制位值为1，那么表示节负责处理槽i。反之则不处理。取出slots数组中的任意一个二进制位的复杂度都为O(1)
CLUSTER KEYSLOT <key>：用于计算KEY属于哪个槽。是基于CRC16校验
发布与订阅
SUBSCRIBE：负责将客户端和被订阅的频道关联到字典里。
UNSUBSCRIBE：则相反。
PUBLISH <channel> <message>:将消息发送给模式订阅者的方法。PUBLISH "news.it" “hello”
PSUBSCRIBE:
PPUBLISH:
UNSUBSCRIBE "news.*" :订阅多个匹配。
PUBSUB CHANNELS [pattern]:用于返回服务器当前被订阅的频道，其中pattern参数是可选的。如果没有则返回所有。
PUBSUB CHANNELS "news.[is]*" 会显示出news.*的所有
PUBSUB NUMSUB news.it news.sport news.busies：会获得像PUBSUB CHANNELS的值一样。
PUBSUB NUMPAT:返回服务器当前被订阅模式的数量。
事务
redis通过MULTI、EXEC、WATCH命令来实现事务(transaction)功能。事务提供了一种将多个命令请求打包，然后一次性、按顺序地执行多个命令的机制，并在事务执行期间，服务器不会中断事务而改去执行其他客户端的命令请求，它会将事务中的所有命令都执行完毕，然后才去处理其他客户端的命令请求。
以下是一个事务执行的过程，该事务首先以一个MULTI命令开始，接着将多个命令放入事务，最后由EXEC将这个事务提交给服务器执行。如下：
MULTI
set name hello
get name
set author biubiu
EXEC
WATCH：命令是一个乐观锁(optimistic locking)，它可以在EXEC命令执行前，监视任意数据库健，并在EXEC命令执行时，检查被监视的健是否至少有一个已经被修改过了，如果是，服务器将拒绝执行事务，并向客户端返回代表事务执行失败时的回复。
SORT：排序字母用ALPHA。如:
sadd mykey a c b e p
smembers mykey a c b e p
sort mykey alpha limit 0 3
sort mykey desc
以序号为权重，对有序集合中的元素排序
zadd mytest 3.0 jack 3.5 peter 4.0 tom

```
127.0.0.1:56379[10]> zrange mytest 0 -1
1) "jack"
2) "peter"
3) "tom"
```
mset peter_number 1 tom_number 2 jack_number 3
```
127.0.0.1:56379[10]> sort test by *_number alpha
1) "peter"
2) "tom"
3) "jack"
```
如果要获取test里面已排好序值的某些值可以用GET来实现，如：工获得test里面姓名的全称可以如下：
```
127.0.0.1:56379[10]> mset jack-name "jack snow" tom-name "tom smith" peter-name "peter white"
```
在原来的test里有
```
127.0.0.1:56379[10]> zrange mytest 0 -1
1) "jack"
2) "peter"
3) "tom"
127.0.0.1:56379[10]> sort test alpha get *-name
1) "jack snow"
2) "peter white"
3) "tom smith"
```
这样就取到了全名。还可以多个GET，随着GET增加，命令查找的次数也会增加。比如根据上面的还要获取生日。如下：
```
mset jack-birth 1998 tom-birth 1984 peter-birth 1887
127.0.0.1:56379[10]> sort test alpha get *-name get *-birth
1) "jack snow"
2) "1998"
3) "peter white"
4) "1887"
5) "tom smith"
6) "1984"
```
这样就取到了生日
STORE: 可以把排序的结果放在一个新的key里。如下
```
127.0.0.1:56379[10]> sort test alpha get *-name get *-birth store hello
(integer) 6
127.0.0.1:56379[10]> lrange hello 0 -1
1) "jack snow"
2) "1998"
3) "peter white"
4) "1887"
5) "tom smith"
6) “1984"
```
慢查询
slowlog-log-slower-than：记录超过指定时间的命令会被记到日志上。单位为微秒(1秒等于1000000微秒)
slowlog-max-len：指定服务器最多保存多少条慢查询日志。以先进先出的方式删除。
CONFIG SET 命令：CONFIG SET slowlog-max-slower-than 0，把配制文件的选项重新设置。

SLOWLY GET：查看服务器所保存慢查询日志
MONITOR:监控