**1 daemonize  no**

 是否后台运行，默认redis不是在后台运行的，一般启动会改为yes。

**2 pidfile  var/lib/redis/6379/run/redis.pid**

 进程文件及路径设置。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。当一台server同时运行多个redis实例时，需要指定不同的pid文件和端口。

**3 port**  
端口设置，默认6379。

**4 #bind 127.0.0.1**

 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，在生产环境中为了安全最好设置该项。一般启用。

**5 timeout 0**

 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。默认为0表示禁用。

**6 tcp-keepalive 0**

指定TCP连接是否为长连接。“侦探”信号由server端维护。默认为0表示禁用。

**7 loglevel notice**

 log等级。等级分为debug, verbose, notice和warning。默认一般开启notice。

**8 logfile “var/lib/redis/6379/log/redis.log”**

 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，修改为日志文件目录。

**9 databases 16**

设置数据库的个数，可以使用select命令来切换db（cluster模式下此项设置无效，select命令无效，集群默认db0）。默认使用的数据库是0号库。默认16个库

**10**

**save 900 1  
save 300 10  
save 60 10000**

保存数据快照的频率，即将数据持久化到dump.rdb文件中的频度。根据时间维度及操作数维度来触发快照持久化。

默认设置，意思是：

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

}

**11 stop-writes-on-bgsave-error yes**

当持久化出现错误时，是否依然继续进行工作。是否停止客户端write请求。默认为“yes”。Redis作者考虑当持久化异常出现时，停止写请求，异常才会被人为的关注到。如果系统做了其他方面的监控可以把这个功能关闭。如果设为“no”，就算本次快照保存失败，下次保存不受影响，也不会影响Redis本身读写状态。但数据只能恢复到最近一次成功的节点（快照原理）。

**12 rdbcompression yes**

在进行数据镜像备份时，是否压缩RDB文件，默认为yes。压缩需要额外的CPU开支。

**13 rdbchecksum yes**

对RBD文件进行CRC64校验，利于保证文件正确性和完整性。该功能使持久化文件格式更难被污染，但是读写RDB文件时会损失大约10%性能。

**14 dbfilename dump.rdb**

镜像备份文件的文件名，默认为dump.rdb

**15 dir var/lib/redis/6379/persistent/**

数据库镜像备份的文件存放置的路径。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。

**16 # slaveof <masterip> <masterport>**

设置该数据库为其他数据库的从数据库。

**17 masterauth**

当主数据库连接密码验证。

**18 slave-serve-stale-data yes**

当主master服务器挂机或主从复制在进行时，是否依然可以允许客户访问可能过期的数据。在"yes"情况下,slave继续向客户端提供只读服务,有可能此时的数据已经过期；在"no"情况下，任何向此server发送的数据请求服务(包括客户端和此server的slave)都将被告知"error"

**19 slave-read-only yes**

slave是否只读，一般设置为yes

**20 #repl-ping-slave-period 10**

slave向master发送ping消息的间隔（秒），master返回pong，keep alive功能。

**21 #repl-timeout 60**

slave与master通讯超时时间，默认60秒。

**22 repl-disable-tcp-nodelay no**

slave与master的连接是否禁用TCP nodelay。默认no

如果选yes，redis会使用较小的TCP包传输数据到slave。该方式使用较小的网络带宽，slave同步到数据会有一定的延时。

如果选no，表示开启TCP nodelay选项，slave收到同步数据较快，使用网络带宽较大。

**23 repl-backlog-size 32mb**  
增量缓存文件大小。防止slave闪短触发全同步，根据实际redis用量可以调整大小。

**24 slave-priority 100**

slave权重值。当触发主从切换时，会从slave列表中找到权重值最低（>0）的slave提升为master。如果权重值是0，表示此slave为观察者不参与master选举。

**25 #requirepass foobared**

是否需要安全验证。绝大部分公司都不使用密码。由于redis速度相当快，很容易被暴力破解。

**25 # rename-command CONFIG 3ed984507a5dcd722aeade310065ce5d**   (方式:MD5('CONFIG^!'))

重命名指令,对于一些与"server"控制有关的指令,可能不希望远程客户端(非管理员用户)链接随意使用,那么就可以把这些指令重命名为"难以阅读"的其他字符串

**26 # maxclients 10000**

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。默认值为10000，大小应根据应用及服务器实际情况调整。

**27 maxcmemory 10000**

Reids最大使用内存值。

内存值大小应考虑单机最大内存，slave最大内存，及触发持久化是内存缓冲区。

redis-cache所能使用的最大内存(bytes),默认为0,表示"无限制",最终由OS物理内存大小决定(如果物理内存不足,有可能会使用swap)。此值尽量不要超过机器的物理内存尺寸,从性能和实施的角度考虑,可以为物理内存3/4。此配置需要和"maxmemory-policy"配合使用,当redis中内存数据达到maxmemory时,触发"清除策略"。在"内存不足"时,任何write操作(比如set,lpush等)都会触发"清除策略"的执行。在实际环境中,建议redis的所有物理机器的硬件配置保持一致(内存一致),同时确保master/slave中"maxmemory""policy"配置一致。

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，内存还不够用，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。

**28 # maxmemory-policy volatile-lru**

内存不足"时,数据清除策略,默认为"volatile-lru"。

volatile-lru  ->对"过期集合"中的数据采取LRU(近期最少使用)[算法](http://lib.csdn.net/base/datastructure).如果对key使用"expire"指令指定了过期时间,那么此key将会被添加到"过期集合"中。将已经过期/LRU的数据优先移除.如果"过期集合"中全部移除仍不能满足内存需求,将OOM.  
allkeys-lru ->对所有的数据,采用LRU算法  
volatile-random ->对"过期集合"中的数据采取"随即选取"算法,并移除选中的K-V,直到"内存足够"为止. 如果如果"过期集合"中全部移除全部移除仍不能满足,将OOM  
allkeys-random ->对所有的数据,采取"随机选取"算法,并移除选中的K-V,直到"内存足够"为止  
volatile-ttl ->对"过期集合"中的数据采取TTL算法(最小存活时间),移除即将过期的数据.  
noeviction ->不做任何干扰操作,直接返回OOM异常  
另外，如果数据的过期不会对"应用系统"带来异常,且系统中write操作比较密集,建议采取"allkeys-lru"

**29 # maxmemory-samples 3**

默认值3，上面LRU和最小TTL策略并非严谨的策略，而是大约估算的方式，因此可以选择取样值以便检查

**29 appendonly no**

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。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。如果不经常进行数据迁移操作，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。

另外，对master机器,主要负责写，建议使用AOF,对于slave,主要负责读，挑选出1-2台开启AOF，其余的建议关闭

**31 appendfilename appdonly.aof**  
AOF文件名

**31**

**# appendfsync always  
appendfsync everysec  
# appendfsync no**

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。no不主动fsync，由OS自己来完成。这个需要根据实际业务场景进行配置

**32 no-appendfsync-on-rewrite no**

在aof rewrite期间,是否对aof新记录的append暂缓使用文件同步策略,主要考虑磁盘IO开支和请求阻塞时间。默认为no,表示"不暂缓",新的aof记录仍然会被立即同步

**33 auto-aof-rewrite-percentage 100**

 当Aof log增长超过指定比例时，重写log file， 设置为0表示不自动重写Aof 日志，重写是为了使aof体积保持最小，而确保保存最完整的数据。

**34 auto-aof-rewrite-min-size 64mb**

触发aof rewrite的最小文件尺寸

**35 lua-time-limit 5000**

lua脚本运行的最大时间

**36 slowlog-log-slower-than 10000**

"慢操作日志"记录,单位:微秒(百万分之一秒,1000 \* 1000),如果操作时间超过此值,将会把command信息"记录"起来.(内存,非文件)。其中"操作时间"不包括网络IO开支,只包括请求达到server后进行"内存实施"的时间."0"表示记录全部操作

**37 slowlog-max-len 128**

"慢操作日志"保留的最大条数,"记录"将会被队列化,如果超过了此长度,旧记录将会被移除。可以通过"SLOWLOG <subcommand> args"查看慢记录的信息(SLOWLOG get 10,SLOWLOG reset)

**38 hash-max-ziplist-entries 512**

hash类型的数据结构在编码上可以使用ziplist和hashtable。ziplist的特点就是文件存储(以及内存存储)所需的空间较小,在内容较小时,性能和hashtable几乎一样.因此redis对hash类型默认采取ziplist。如果hash中条目的条目个数或者value长度达到阀值,将会被重构为hashtable。

这个参数指的是ziplist中允许存储的最大条目个数，，默认为512，建议为128

**hash-max-ziplist-value 64**

ziplist中允许条目value值最大字节数，默认为64，建议为1024

**39**

**list-max-ziplist-entries 512**

**list-max-ziplist-value 64**

对于list类型,将会采取ziplist,linkedlist两种编码类型。解释同上。

**40 set-max-intset-entries 512**

intset中允许保存的最大条目个数,如果达到阀值,intset将会被重构为hashtable

**41**

**zset-max-ziplist-entries 128  
zset-max-ziplist-value 64**

zset为有序集合,有2中编码类型:ziplist,skiplist。因为"排序"将会消耗额外的性能,当zset中数据较多时,将会被重构为skiplist。

**42 activerehashing yes**

是否开启顶层数据结构的rehash功能,如果内存允许,请开启。rehash能够很大程度上提高K-V存取的效率

**43**

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

客户端buffer控制。在客户端与server进行的交互中,每个连接都会与一个buffer关联,此buffer用来队列化等待被client接受的响应信息。如果client不能及时的消费响应信息,那么buffer将会被不断积压而给server带来内存压力.如果buffer中积压的数据达到阀值,将会导致连接被关闭,buffer被移除。

buffer控制类型包括:normal -> 普通连接；slave ->与slave之间的连接；pubsub ->pub/sub类型连接，此类型的连接，往往会产生此种问题;因为pub端会密集的发布消息,但是sub端可能消费不足.  
指令格式:client-output-buffer-limit <class> <hard> <soft> <seconds>",其中hard表示buffer最大值,一旦达到阀值将立即关闭连接;  
soft表示"容忍值",它和seconds配合,如果buffer值超过soft且持续时间达到了seconds,也将立即关闭连接,如果超过了soft但是在seconds之后，buffer数据小于了soft,连接将会被保留.  
其中hard和soft都设置为0,则表示禁用buffer控制.通常hard值大于soft.

**44 hz 10**

Redis server执行后台任务的频率,默认为10,此值越大表示redis对"间歇性task"的执行次数越频繁(次数/秒)。"间歇性task"包括"过期集合"检测、关闭"空闲超时"的连接等,此值必须大于0且小于500。此值过小就意味着更多的cpu周期消耗,后台task被轮询的次数更频繁。此值过大意味着"内存敏感"性较差。建议采用默认值。