**Redis常用配置项详解**

Redis 的配置文件位于 Redis 安装目录下，文件名为 redis.conf。

也可以通过 CONFIG 命令查看或设置配置项。

命令如下:

CONFIG GET pidfile

**配置项详细说明:**

**daemonize**默认为no，redis默认不是在后台运行的，如果改为yes，则在后台运行

**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

**port**  
    监听端口

**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。默认注释，不开启

**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

**tcp-keepalive**    指定TCP连接是否为长连接,"侦探"信号有server端维护。默认为0表示禁用

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

**logfile stdout**    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

**databases 16**    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号库。默认16个库

**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。用来描述"在多少秒期间至少多少个变更操作"触发snapshot数据保存动作  
    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  
    }

**stop-writes-on-bgsave-error**    当持久化出现错误时，是否依然继续进行工作，是否终止所有的客户端write请求。默认设置"yes"表示终止，一旦snapshot数据保存故障，那么此server为只读服务。如果为"no"，那么此次snapshot将失败，但下一次snapshot不会受到影响，不过如果出现故障,数据只能恢复到"最近一个成功点"

**rdbcompression**    在进行数据镜像备份时，是否启用rdb文件压缩手段，默认为yes。压缩可能需要额外的cpu开支，不过这能够有效的减小rdb文件的大，有利于存储/备份/传输/数据恢复

**rdbchecksum**    是否进行校验和，是否对rdb文件使用CRC64校验和,默认为"yes"，那么每个rdb文件内容的末尾都会追加CRC校验和，利于第三方校验工具检测文件完整性

**dbfilename dump.rdb**    镜像备份文件的文件名，默认为 dump.rdb

**dir ./**    数据库镜像备份的文件rdb/AOF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 <masterip> <masterport>**    设置该数据库为其他数据库的从数据库，并为其指定master信息。

**masterauth**    当主数据库连接需要密码验证时，在这里指定

**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"

**slave-read-only yes**    slave是否为"只读"，强烈建议为"yes"

**repl-ping-slave-period 10**    slave向指定的master发送ping消息的时间间隔(秒)，默认为10

**repl-timeout 60**    slave与master通讯中,最大空闲时间,默认60秒.超时将导致连接关闭

**repl-disable-tcp-nodelay no**    slave与master的连接,是否禁用TCP nodelay选项。"yes"表示禁用,那么socket通讯中数据将会以packet方式发送(packet大小受到socket buffer限制)。  
    可以提高socket通讯的效率(tcp交互次数),但是小数据将会被buffer,不会被立即发送,对于接受者可能存在延迟。"no"表示开启tcp nodelay选项,任何数据都会被立即发送,及时性较好,但是效率较低，建议设为no

**slave-priority 100**    适用Sentinel模块(unstable,M-S集群管理和监控),需要额外的配置文件支持。slave的权重值,默认100.当master失效后,Sentinel将会从slave列表中找到权重值最低(>0)的slave,并提升为master。如果权重值为0,表示此slave为"观察者",不参与master选举

**requirepass foobared**    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。

**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

**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。默认为10000，要考虑系统文件描述符限制，不宜过大，浪费文件描述符，具体多少根据具体情况而定

**maxmemory <bytes>**    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。

**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.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"

**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

**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

**appendfilename appendonly.aof**    aof文件名字，默认为appendonly.aof

**appendfsync always/everysec/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

**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

**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。

**auto-aof-rewrite-min-size 64mb**    触发aof rewrite的最小文件尺寸

**lua-time-limit 5000**    lua脚本运行的最大时间

**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

**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)

**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

**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

**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