**redis常用命令、常见错误、配置技巧等分享**

转载  2015-02-02   投稿：junjie   [[http://www.jb51.net/skin/2018/images/text-message.png](http://www.jb51.net/article/60627.htm#comments) 我要评论](http://www.jb51.net/article/60627.htm#comments)

这篇文章主要介绍了redis常用命令、常见错误、配置技巧等分享,本文分享了12条redis知识,需要的朋友可以参考下

**1. redis查看当前所有的key**

复制代码代码如下:

KEYS \*

**2. 查看当前redis的配置信息**

复制代码代码如下:

CONFIG GET \*

**3. MISCONF Redis is configured to save RDB snapshots, but is currently not able to persist on disk. Commands that may modify the data set are disabled. Please check Redis logs for details about the error.**

强制停止redis快照导致，redis运行用户没有权限写rdb文件或者磁盘空间满了，解决办法：

复制代码代码如下:

config set stop-writes-on-bgsave-error no

例如：

复制代码代码如下:

set 'name' 'shenhui'  
-MISCONF Redis is configured to save RDB snapshots, but is currently not able to persist on disk. Commands that may modify the data set are disabled. Please check Redis logs for details about the error.  
config set stop-writes-on-bgsave-error no  
+OK  
set 'name' 'shenhui'  
+OK

**4. redis 127.0.0.1:6379> CONFIG SET logfile "/var/log/redis/redis-server.log"  
(error) ERR Unsupported CONFIG parameter: logfile**

logfile 不能通过set动态设置

**5.(error) OOM command not allowed when used memory >**设置了maxmemory的选项,redis内存使用达到上限。  
可以通过设置LRU算法来删除部分key,释放空间。  
默认是按照过期时间的,如果set时候没有加上过期时间就会导致数据写满maxmemory。  
如果不设置maxmemory或者设置为0 64位系统不限制内存，32位系统最多使用3GB内存。

volatile-lru -> 根据LRU算法生成的过期时间来删除。  
allkeys-lru -> 根据LRU算法删除任何key。  
volatile-random -> 根据过期设置来随机删除key。  
allkeys->random -> 无差别随机删。  
volatile-ttl -> 根据最近过期时间来删除（辅以TTL）  
noeviction -> 谁也不删，直接在写操作时返回错误。

**6. reids日志位置**

logfile 日志记录方式，默认值为stdout，如果设置为stdout且以守护进程方式运行，那么日志会被重定向到/dev/null,也就是不记日志。

**7. reids配置参数详解**

复制代码代码如下:

#daemonize no  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  
daemonize yes  
#  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  
pidfile /var/run/redis\_6379.pid  
#  指定 redis 运行的端口，默认是 6379  
port 6379  
#  在高并发的环境中，为避免慢客户端的连接问题，需要设置一个高速后台日志  
tcp-backlog 511  
#  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  
# bind 192.168.1.100 10.0.0.1  
# bind 127.0.0.1  
#  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  
# 0 是关闭此设置  
timeout 0  
# TCP keepalive  
#  在 Linux 上，指定值（秒）用于发送 ACKs 的时间。注意关闭连接需要双倍的时间。默认为 0 。  
tcp-keepalive 0  
#  指定日志记录级别，生产环境推荐 notice  
# Redis 总共支持四个级别： debug 、 verbose 、 notice 、 warning ，默认为 verbose  
# debug     记录很多信息，用于开发和测试  
# varbose   有用的信息，不像 debug 会记录那么多  
# notice    普通的 verbose ，常用于生产环境  
# warning   只有非常重要或者严重的信息会记录到日志  
loglevel notice  
#  配置 log 文件地址  
#  默认值为 stdout ，标准输出，若后台模式会输出到 /dev/null 。  
logfile /var/log/redis/redis.log  
#  可用数据库数  
#  默认值为 16 ，默认数据库为 0 ，数据库范围在 0- （ database-1 ）之间  
databases 16  
################################ 快照#################################  
#  保存数据到磁盘，格式如下 :  
#   save    
#    指出在多长时间内，有多少次更新操作，就将数据同步到数据文件 rdb 。  
#    相当于条件触发抓取快照，这个可以多个条件配合  
#    比如默认配置文件中的设置，就设置了三个条件  
#   save 900 1  900 秒内至少有 1 个 key 被改变  
#   save 300 10  300 秒内至少有 300 个 key 被改变  
#   save 60 10000  60 秒内至少有 10000 个 key 被改变  
# save 900 1  
# save 300 10  
# save 60 10000  
#  后台存储错误停止写。  
stop-writes-on-bgsave-error yes  
#  存储至本地数据库时（持久化到 rdb 文件）是否压缩数据，默认为 yes  
rdbcompression yes  
# RDB 文件的是否直接偶像 chcksum  
rdbchecksum yes  
#  本地持久化数据库文件名，默认值为 dump.rdb  
dbfilename dump.rdb  
#  工作目录  
#  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，  
#  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。  
# AOF 文件也会存放在这个目录下面  
#  注意这里必须制定一个目录而不是文件  
dir /var/lib/redis-server/  
################################# 复制 #################################  
#  主从复制 . 设置该数据库为其他数据库的从数据库 .  
#  设置当本机为 slav 服务时，设置 master 服务的 IP 地址及端口，在 Redis 启动时，它会自动从 master 进行数据同步  
# slaveof   
#  当 master 服务设置了密码保护时 ( 用 requirepass 制定的密码 )  
# slave 服务连接 master 的密码  
# masterauth   
#  当从库同主机失去连接或者复制正在进行，从机库有两种运行方式：  
# 1)  如果 slave-serve-stale-data 设置为 yes( 默认设置 ) ，从库会继续响应客户端的请求  
# 2)  如果 slave-serve-stale-data 是指为 no ，出去 INFO 和 SLAVOF 命令之外的任何请求都会返回一个  
#     错误 "SYNC with master in progress"  
slave-serve-stale-data yes  
#  配置 slave 实例是否接受写。写 slave 对存储短暂数据（在同 master 数据同步后可以很容易地被删除）是有用的，但未配置的情况下，客户端写可能会发送问题。  
#  从 Redis2.6 后，默认 slave 为 read-only  
slaveread-only yes  
#  从库会按照一个时间间隔向主库发送 PINGs. 可以通过 repl-ping-slave-period 设置这个时间间隔，默认是 10 秒  
# repl-ping-slave-period 10  
# repl-timeout  设置主库批量数据传输时间或者 ping 回复时间间隔，默认值是 60 秒  
#  一定要确保 repl-timeout 大于 repl-ping-slave-period  
# repl-timeout 60  
#  在 slave socket 的 SYNC 后禁用 TCP\_NODELAY  
#  如果选择“ yes ” ,Redis 将使用一个较小的数字 TCP 数据包和更少的带宽将数据发送到 slave ， 但是这可能导致数据发送到 slave 端会有延迟 , 如果是 Linux kernel 的默认配置，会达到 40 毫秒 .  
#  如果选择 "no" ，则发送数据到 slave 端的延迟会降低，但将使用更多的带宽用于复制 .  
repl-disable-tcp-nodelay no  
#  设置复制的后台日志大小。  
#  复制的后台日志越大， slave 断开连接及后来可能执行部分复制花的时间就越长。  
#  后台日志在至少有一个 slave 连接时，仅仅分配一次。  
# repl-backlog-size 1mb  
#  在 master 不再连接 slave 后，后台日志将被释放。下面的配置定义从最后一个 slave 断开连接后需要释放的时间（秒）。  
# 0 意味着从不释放后台日志  
# repl-backlog-ttl 3600  
#  如果 master 不能再正常工作，那么会在多个 slave 中，选择优先值最小的一个 slave 提升为 master ，优先值为 0 表示不能提升为 master 。  
slave-priority 100  
#  如果少于 N 个 slave 连接，且延迟时间 <=M 秒，则 master 可配置停止接受写操作。  
#  例如需要至少 3 个 slave 连接，且延迟 <=10 秒的配置：  
# min-slaves-to-write 3  
# min-slaves-max-lag 10  
#  设置 0 为禁用  
#   默认 min-slaves-to-write 为 0 （禁用）， min-slaves-max-lag 为 10  
################################## 安全 ###################################  
#  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  
# requirepass foobared  
#  命令重命名 .  
#  在一个共享环境下可以重命名相对危险的命令。比如把 CONFIG 重名为一个不容易猜测的字符。  
#  举例 :  
# rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52  
#  如果想删除一个命令，直接把它重命名为一个空字符 "" 即可，如下：  
# rename-command CONFIG ""  
################################### 约束###################################  
#设置同一时间最大客户端连接数，默认无限制，   
#Redis 可以同时打开的客户端连接数为 Redis 进程可以打开的最大文件描述符数，  
#如果设置  maxclients 0 ，表示不作限制。  
#当客户端连接数到达限制时， Redis 会关闭新的连接并向客户端返回 max number of clients reached 错误信息  
# maxclients 10000  
#  指定 Redis 最大内存限制， Redis 在启动时会把数据加载到内存中，达到最大内存后， Redis 会按照清除策略尝试清除已到期的 Key  
#  如果 Redis 依照策略清除后无法提供足够空间，或者策略设置为 ”noeviction” ，则使用更多空间的命令将会报错，例如 SET, LPUSH 等。但仍然可以进行读取操作  
#  注意： Redis 新的 vm 机制，会把 Key 存放内存， Value 会存放在 swap 区  
#  该选项对 LRU 策略很有用。  
# maxmemory 的设置比较适合于把 redis 当作于类似 memcached 的缓存来使用，而不适合当做一个真实的 DB 。  
#  当把 Redis 当做一个真实的数据库使用的时候，内存使用将是一个很大的开销  
# maxmemory   
#  当内存达到最大值的时候 Redis 会选择删除哪些数据？有五种方式可供选择  
# volatile-lru ->  利用 LRU 算法移除设置过过期时间的 key (LRU: 最近使用  Least RecentlyUsed )  
# allkeys-lru ->  利用 LRU 算法移除任何 key  
# volatile-random ->  移除设置过过期时间的随机 key  
# allkeys->random -> remove a randomkey, any key  
# volatile-ttl ->  移除即将过期的 key(minor TTL)  
# noeviction ->  不移除任何可以，只是返回一个写错误  
#  注意：对于上面的策略，如果没有合适的 key 可以移除，当写的时候 Redis 会返回一个错误  
#  默认是 :  volatile-lru  
# maxmemory-policy volatile-lru    
# LRU  和  minimal TTL 算法都不是精准的算法，但是相对精确的算法 ( 为了节省内存 ) ，随意你可以选择样本大小进行检测。  
# Redis 默认的灰选择 3 个样本进行检测，你可以通过 maxmemory-samples 进行设置  
# maxmemory-samples 3  
############################## AOF###############################  
#  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。  
#  你可以同时开启 asynchronous dumps 和  AOF  
appendonly no  
# AOF 文件名称  ( 默认 : "appendonly.aof")  
# appendfilename appendonly.aof  
# Redis 支持三种同步 AOF 文件的策略 :  
# no:  不进行同步，系统去操作  . Faster.  
# always: always 表示每次有写操作都进行同步 . Slow, Safest.  
# everysec:  表示对写操作进行累积，每秒同步一次 . Compromise.  
#  默认是 "everysec" ，按照速度和安全折中这是最好的。  
#  如果想让 Redis 能更高效的运行，你也可以设置为 "no" ，让操作系统决定什么时候去执行  
#  或者相反想让数据更安全你也可以设置为 "always"  
#  如果不确定就用  "everysec".  
# appendfsync always  
appendfsync everysec  
# appendfsync no  
# AOF 策略设置为 always 或者 everysec 时，后台处理进程 ( 后台保存或者 AOF 日志重写 ) 会执行大量的 I/O 操作  
#  在某些 Linux 配置中会阻止过长的 fsync() 请求。注意现在没有任何修复，即使 fsync 在另外一个线程进行处理  
#  为了减缓这个问题，可以设置下面这个参数 no-appendfsync-on-rewrite  
no-appendfsync-on-rewrite no  
# AOF  自动重写  
#  当 AOF 文件增长到一定大小的时候 Redis 能够调用  BGREWRITEAOF  对日志文件进行重写  
#  它是这样工作的： Redis 会记住上次进行些日志后文件的大小 ( 如果从开机以来还没进行过重写，那日子大小在开机的时候确定 )  
#  基础大小会同现在的大小进行比较。如果现在的大小比基础大小大制定的百分比，重写功能将启动  
#  同时需要指定一个最小大小用于 AOF 重写，这个用于阻止即使文件很小但是增长幅度很大也去重写 AOF 文件的情况  
#  设置  percentage 为 0 就关闭这个特性  
auto-aof-rewrite-percentage 100  
auto-aof-rewrite-min-size 64mb  
################################ LUASCRIPTING #############################  
# 一个 Lua 脚本最长的执行时间为 5000 毫秒（ 5 秒），如果为 0 或负数表示无限执行时间。  
lua-time-limit 5000  
################################LOW LOG################################  
# Redis Slow Log  记录超过特定执行时间的命令。执行时间不包括 I/O 计算比如连接客户端，返回结果等，只是命令执行时间  
#  可以通过两个参数设置 slow log ：一个是告诉 Redis 执行超过多少时间被记录的参数 slowlog-log-slower-than( 微妙 ) ，  
#  另一个是 slow log 的长度。当一个新命令被记录的时候最早的命令将被从队列中移除  
#  下面的时间以微妙为单位，因此 1000000 代表一秒。  
#  注意指定一个负数将关闭慢日志，而设置为 0 将强制每个命令都会记录  
slowlog-log-slower-than 10000  
#  对日志长度没有限制，只是要注意它会消耗内存  
#  可以通过  SLOWLOG RESET 回收被慢日志消耗的内存  
#  推荐使用默认值 128 ，当慢日志超过 128 时，最先进入队列的记录会被踢出  
slowlog-max-len 128  
################################  事件通知  #############################  
#  当事件发生时， Redis 可以通知 Pub/Sub 客户端。  
#  可以在下表中选择 Redis 要通知的事件类型。事件类型由单个字符来标识：  
# K     Keyspace 事件，以 \_keyspace@\_ 的前缀方式发布  
# E     Keyevent 事件，以 \_keysevent@\_ 的前缀方式发布  
# g     通用事件（不指定类型），像 DEL, EXPIRE, RENAME, …  
# $     String 命令  
# s     Set 命令  
# h     Hash 命令  
# z     有序集合命令  
# x     过期事件（每次 key 过期时生成）  
# e     清除事件（当 key 在内存被清除时生成）  
# A     g$lshzxe 的别称，因此 ”AKE” 意味着所有的事件  
# notify-keyspace-events 带一个由 0 到多个字符组成的字符串参数。空字符串意思是通知被禁用。  
#  例子：启用 list 和通用事件：  
# notify-keyspace-events Elg  
#  默认所用的通知被禁用，因为用户通常不需要改特性，并且该特性会有性能损耗。  
#  注意如果你不指定至少 K 或 E 之一，不会发送任何事件。  
notify-keyspace-events “”  
##############################  高级配置  ###############################  
#  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  
# Redis Hash 对应 Value 内部实际就是一个 HashMap ，实际这里会有 2 种不同实现，  
#  这个 Hash 的成员比较少时 Redis 为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的 HashMap 结构，对应的 valueredisObject 的 encoding 为 zipmap,  
#  当成员数量增大时会自动转成真正的 HashMap, 此时 encoding 为 ht 。  
hash-max-zipmap-entries 512  
hash-max-zipmap-value 64    
#  和 Hash 一样，多个小的 list 以特定的方式编码来节省空间。  
# list 数据类型节点值大小小于多少字节会采用紧凑存储格式。  
list-max-ziplist-entries 512  
list-max-ziplist-value 64  
# set 数据类型内部数据如果全部是数值型，且包含多少节点以下会采用紧凑格式存储。  
set-max-intset-entries 512  
#  和 hashe 和 list 一样 , 排序的 set 在指定的长度内以指定编码方式存储以节省空间  
# zsort 数据类型节点值大小小于多少字节会采用紧凑存储格式。  
zset-max-ziplist-entries 128  
zset-max-ziplist-value 64  
# Redis 将在每 100 毫秒时使用 1 毫秒的 CPU 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  
activerehashing yes  
# 客户端的输出缓冲区的限制，因为某种原因客户端从服务器读取数据的速度不够快，  
# 可用于强制断开连接（一个常见的原因是一个发布 / 订阅客户端消费消息的速度无法赶上生产它们的速度）。  
#  可以三种不同客户端的方式进行设置：  
# normal ->  正常客户端  
# slave  -> slave 和 MONITOR 客户端  
# pubsub ->  至少订阅了一个 pubsub channel 或 pattern 的客户端  
#  每个 client-output-buffer-limit 语法 :  
# client-output-buffer-limit     
#  一旦达到硬限制客户端会立即断开，或者达到软限制并保持达成的指定秒数（连续）。  
#  例如，如果硬限制为 32 兆字节和软限制为 16 兆字节 /10 秒，客户端将会立即断开  
#  如果输出缓冲区的大小达到 32 兆字节，客户端达到 16 兆字节和连续超过了限制 10 秒，也将断开连接。  
#  默认 normal 客户端不做限制，因为他们在一个请求后未要求时（以推的方式）不接收数据，  
#  只有异步客户端可能会出现请求数据的速度比它可以读取的速度快的场景。  
#  把硬限制和软限制都设置为 0 来禁用该特性  
client-output-buffer-limit normal 0 0 0  
client-output-buffer-limit slave 256mb 64mb60  
client-output-buffer-limit pubsub 32mb 8mb60  
# Redis 调用内部函数来执行许多后台任务，如关闭客户端超时的连接，清除过期的 Key ，等等。  
#  不是所有的任务都以相同的频率执行，但 Redis 依照指定的“ Hz ”值来执行检查任务。  
#  默认情况下，“ Hz ”的被设定为 10 。  
#  提高该值将在 Redis 空闲时使用更多的 CPU 时，但同时当有多个 key 同时到期会使 Redis 的反应更灵敏，以及超时可以更精确地处理。  
#  范围是 1 到 500 之间，但是值超过 100 通常不是一个好主意。  
#  大多数用户应该使用 10 这个预设值，只有在非常低的延迟的情况下有必要提高最大到 100 。  
hz 10    
#  当一个子节点重写 AOF 文件时，如果启用下面的选项，则文件每生成 32M 数据进行同步。  
aof-rewrite-incremental-fsync yes

**8.Redis官方文档对VM的使用建议:**

当你的key很小而value很大时,使用VM的效果会比较好.因为这样节约的内存比较大.  
当你的key不小时,可以考虑使用一些非常方法将很大的key变成很大的value,比如你可以考虑将key,value组合成一个新的value.  
最好使用linux ext3 等对稀疏文件支持比较好的文件系统保存你的swap文件.  
vm-max-threads这个参数,可以设置访问swap文件的线程数,设置最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的.可能会造成比较长时间的延迟,但是对数据完整性有很好的保证.  
有了VM功能,Redis终于摆脱了受内存容量限制的噩梦了,似乎我们可以称其为Redis数据库了,我们还可以想象又有多少新的用法可以产生.当然,希望这一功能不会对Redis原有的非常牛B的内存存储性能有所影响.

**9. redis修改持久化路径和日志路径**

复制代码代码如下:

vim redis.conf  
logfile /data/redis\_cache/logs/redis.log #日志路径

dir /data/redis\_cache #持久化路径，修改后 记得要把dump.rdb持久化文件拷贝到/data/redis\_cache下

先杀掉redis，拷贝dump.rdb，启动

**10. 清redis缓存**

复制代码代码如下:

./redis-cli    #进入  
dbsize  
flushall     #执行  
exit

**11. 删除redis当前数据库中的所有Key**

复制代码代码如下:

flushdb

**12.删除redis所有数据库中的key**

复制代码代码如下:

flushall

**您可能感兴趣的文章:**

1. [redis常用命令小结](http://www.jb51.net/article/51625.htm)
2. [Redis总结笔记（一）：安装和常用命令](http://www.jb51.net/article/59429.htm)
3. [Redis操作命令总结](http://www.jb51.net/article/61793.htm)
4. [Redis教程(七)：Key操作命令详解](http://www.jb51.net/article/65256.htm)
5. [Redis中主键失效的原理及实现机制剖析](http://www.jb51.net/article/68254.htm)
6. [Redis禁用命令、危险命令及规避方法](http://www.jb51.net/article/68310.htm)
7. [Redis sort 排序命令详解](http://www.jb51.net/article/69131.htm)
8. [Redis使用Eval多个键值自增的操作实例](http://www.jb51.net/article/96489.htm)
9. [详解Redis命令和键\_动力节点Java学院整理](http://www.jb51.net/article/122265.htm)