# Redis配置文件详解

参考地址：

<http://www.jb51.net/article/60627.htm>

<http://www.redis.cn/commands.html#>

<http://www.cnblogs.com/zhang-ke/p/5981108.html>

## 01.引入redis配置文件

# Redis configuration file example.

[redis配置文件案例]

#

# Note that in order to read the configuration file, Redis must be started with the file path as first argument:

[注意：为读取配置文件，redis需要把配置文件作为第一参数。理解：当执行redis-server命令，如果不使用默认配置，而使用其他自定义配置，那么redis-server后面跟着着就是配置文件路径：

比如 ./redis-server /mnt/java/config/redis.cong]

# ./redis-server /path/to/redis.conf

# Note on units: when memory size is needed, it is possible to specify it in the usual form of 1k 5GB 4M and so forth:

[注意：当设置内存大小时，需要使用1k 5GB 4M的形式]

#

# 1k => 1000 bytes

# 1kb => 1024 bytes

# 1m => 1000000 bytes

# 1mb => 1024\*1024 bytes

# 1g => 1000000000 bytes

# 1gb => 1024\*1024\*1024 bytes

#

# units are case insensitive so 1GB 1Gb 1gB are all the same.

[单位是不区分大小写的，因此1GB 1Gb 1gB 是相同的]

########### INCLUDES [include]#################

# Include one or more other config files here. This is useful if you

[如果你有配置模板需要给所有的redis服务器使用，但有些自定义的设置，当在其他配置中包含这个通用的配置时，include就在此时使用]

# have a standard template that goes to all Redis servers but also need

# to customize a few per-server settings. Include files can include

# other files, so use this wisely.

#

# Notice option "include" won't be rewritten by command "CONFIG REWRITE"

[注意include选项不能被config rewrite命令重写。

config rewrite命令的作用是在运行时重新设置redis的参数。

如果在程序中设置int a=1， a=2， a=3，那么知晓a最后被数据3所重写；如果模板参数打算覆盖redis server配置数据，那么将include包含的文件放置在配置文件末尾；]

# from admin or Redis Sentinel. Since Redis always uses the last processed

# line as value of a configuration directive, you'd better put includes

# at the beginning of this file to avoid overwriting config change at runtime.

[如果想避免配置文件在运行时被重写，那么将include包含的文件放在配置文件首部]

#

# If instead you are interested in using includes to override configuration options, it is better to use include as the last line.

[如果你打算使用include中文件配置覆盖掉原有配置文件的某些选项，那么将include中文件放置在配置文件的末尾]

#

# include /path/to/local.conf

# include /path/to/other.conf

## 02.网络连接

############# NETWORK [网络]#######################

# By default, if no "bind" configuration directive is specified, Redis listens for connections from all the network interfaces available on the server.

[默认情况，如果没有绑定配置指令，redis会监听所有来自网络上的请求。Redis安全设置是依赖于安全环境，本身对安全设置简单。Redis安全好比温室里的花朵，经不起外界的风吹雨打]

# It is possible to listen to just one or multiple selected interfaces using the "bind" configuration directive, followed by one or more IP addresses.  
 [使用绑定配置来监听已选择的一个或者多个IP]

#

# Examples:

[多个IP之间使用空格，::1是什么含义呢？]

# bind 192.168.1.100 10.0.0.1

# bind 127.0.0.1 ::1

#

[警告：如果运行redis的服务器直接连接在网络上，那么允许任何接口进行访问是危险的，再次强调不要直接在网络上连接，需要绑定固定IP来访问redis]

# ~~~ WARNING ~~~ If the computer running Redis is directly exposed to the internet, binding to all the interfaces is dangerous and will expose the instance to everybody on the internet. So by default we uncomment the

# following bind directive, that will force Redis to listen only into the IPv4 lookback interface address (this means Redis will be able to accept connections only from clients running into the same computer it is running).

[Redis的作者对安全的提醒重复若干，其实可以拿出个实例让使用者影响深刻的，比如某次某个公司被攻击;

如果你确认你的redis允许任何机器访问，那么仅仅注释掉以下这行即可。bind 127.0.0.1]

# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES JUST COMMENT THE FOLLOWING LINE.

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

bind 127.0.0.1

# Protected mode is a layer of security protection, in order to avoid that Redis instances left open on the internet are accessed and exploited.

[保护模式是一种安全机制，为了避免redis实例被互联网访问和利用，保护模式当以下情况被开启：]

#

# When protected mode is on and if:

#

# 1) The server is not binding explicitly to a set of addresses using the "bind" directive.

[1.服务器没有明显地使用bind指令绑定若干IP地址

2.没有配置密码]

# 2) No password is configured.

#

[以上两种情景作用下会开启保护模式，保护模式是服务器只会接收本机和unix socket的访问，不接受其他访问]

# The server only accepts connections from clients connecting from the IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix domain sockets.

[默认情况下保护模式是开启的，如果你项禁止保护模式，当你确认任何机器都会访问redis或者没有密码的时候.你可以将yes修改为no即可]

# By default protected mode is enabled. You should disable it only if you are sure you want clients from other hosts to connect to Redis even if no authentication is configured, nor a specific set of interfaces are explicitly listed using the "bind" directive.

protected-mode yes

[接收指定端口连接，默认端口6379.如果将端口设置为0，那么redis将不会监听TCP连接]

# Accept connections on the specified port, default is 6379 (IANA #815344).

# If port 0 is specified Redis will not listen on a TCP socket.

port 6379

# TCP listen() backlog.

[在高并发环境中，你需要设置高的backlog来避免缓慢的客户端连接.注意Linux内核会将其在/proc/sys/net/core/somaxconn中设置个默认值，因此为了获取预期效果，需要同时提高somaxconn和tcp\_max\_syn\_backlog的数值。

具体什么用处我也不确定，高并发的时候需要处理这两个数值]

# In high requests-per-second environments you need an high backlog in order to avoid slow clients connections issues. Note that the Linux kernel will silently truncate it to the value of /proc/sys/net/core/somaxconn so make sure to raise both the value of somaxconn and tcp\_max\_syn\_backlog in order to get the desired effect.

tcp-backlog 511

# Unix socket.

[指定监听unix套接字连接路径。没有默认指定路径，因此Redis不会监听unix套接字路径]

# Specify the path for the Unix socket that will be used to listen for incoming connections. There is no default, so Redis will not listen on a unix socket when not specified.

#

# unixsocket /tmp/redis.sock

# unixsocketperm 700

[客户端和服务器端连接超时时间，即当客户端连接服务器端多少秒后管理连接，0表示不设置超时时间]

# Close the connection after a client is idle for N seconds (0 to disable)

timeout 0

# TCP keepalive.

[使用SO\_KEEPALIVE发送TCP ACKs给客户端来确认有效连接。为什么这么处理：检测出死链接；降低中间设备出问题而导致网络看似连接却已经与对端端口的问题。

什么鬼？心跳检测，防止存在死链接或者霸占连接不还的问题]

# If non-zero, use SO\_KEEPALIVE to send TCP ACKs to clients in absence of communication. This is useful for two reasons:

# 1) Detect dead peers.

# 2) Take the connection alive from the point of view of network equipment in the middle.

[在Linux中，发送ACKs的周期是特定的值，注意关闭连接需要双倍的周期时间，其他Linux中发送ACKs则取决于内核设置。以上是Linux设置]

# On Linux, the specified value (in seconds) is the period used to send ACKs.Note that to close the connection the double of the time is needed.On other kernels the period depends on the kernel configuration.

[Redis设置该数据是一个有效的时间值是300秒]

# A reasonable value for this option is 300 seconds, which is the new Redis default starting with Redis 3.2.1.

tcp-keepalive 300

## 03.常用设置

############ GENERAL [一般设置]#######################

[默认Redis不是作为守护进程在后台运行。如果你需要让它作为守护进程，那么使用yes即可。注意当redis作为守护进程时，会在/var/run/下写入redis.pid]

# By default Redis does not run as a daemon. Use 'yes' if you need it. Note that Redis will write a pid file in /var/run/redis.pid when daemonized.

daemonize no

[没有明白什么作用]

# If you run Redis from upstart or systemd, Redis can interact with your supervision tree. Options:

# supervised no - no supervision interaction

# supervised upstart - signal upstart by putting Redis into SIGSTOP mode

# supervised systemd - signal systemd by writing READY=1 to $NOTIFY\_SOCKET

# supervised auto - detect upstart or systemd method based on UPSTART\_JOB or NOTIFY\_SOCKET environment variables

# Note: these supervision methods only signal "process is ready." hey do not enable continuous liveness pings back to your supervisor.

supervised no

# If a pid file is specified, Redis writes it where specified at startup and removes it at exit.

[如果已经指定一个pid文件，那么在开启和退出的时候，redis将会进行重写它]

# When the server runs non daemonized, no pid file is created if none is specified in the configuration. When the server is daemonized, the pid file is used even if not specified, defaulting to "/var/run/redis.pid".

[当服务器不是以守护进程执行，如果没有在配置文件中设置时，是不会创建pid文件。当服务进程是守护进程，即使没有设置pid文件，那么会在/var/run/redis/redis.pid创建pid文件]

#

# Creating a pid file is best effort: if Redis is not able to create it nothing bad happens, the server will start and run normally.

[创建pid文件是好的行为，即使没有创建pid文件也没有任何关系，redis服务器会正常运行]

pidfile /var/run/redis\_6379.pid

## 04.日志信息

[指定服务器日志级别]

# Specify the server verbosity level.

# This can be one of:

# debug (a lot of information, useful for development/testing)

# verbose (many rarely useful info, but not a mess like the debug level)

# notice (moderately verbose, what you want in production probably)

# warning (only very important / critical messages are logged)

[debug提供许多有用的信息，适合开发和测试阶段；verbose比debug信息量少，比较少的有用信息；notice适合生产环境；warning仅仅提供少量的有用信息]

loglevel notice

# Specify the log file name. Also the empty string can be used to force Redis to log on the standard output. Note that if you use standard output for logging but daemonize, logs will be sent to /dev/null

[如果没有指定日志文件名称，空字符串可以强制redis写入到控制台。注意你使用控制台输入日志，同时如果作为守护进程，那么日志同时会写到/dev/null中]

logfile ""

[如果想使用系统log，那么设置syslog-enabled等于yes即可。可以设置其他的参数来满足日志要求]

# To enable logging to the system logger, just set 'syslog-enabled' to yes, and optionally update the other syslog parameters to suit your needs.

# syslog-enabled no

[指定syslog的identity，什么作用不知道]

# Specify the syslog identity.

# syslog-ident redis

[执行系统日志，必须使用user或使用local0-local7之间，具体何处使用未知]

# Specify the syslog facility. Must be USER or between LOCAL0-LOCAL7.

# syslog-facility local0

# Set the number of databases. The default database is DB 0, you can select a different one on a per-connection basis using SELECT <dbid> where dbid is a number between 0 and 'databases' -1

[设置数据库个数，默认数据库是0，你可以使用select命令 dbId来选择不同的数据库，dbId数据是0到databases-1]

databases 16

## 05.RDB模式

############# SNAPSHOTTING [快照，保存] #################

[将数据保存在硬盘中，使用格式如下：多少秒内多少修改数据进行保存，即save 间隔时间 写入次数]

# Save the DB on disk:

# save <seconds> <changes>

#

# Will save the DB if both the given number of seconds and the given number of write operations against the DB occurred.

[如果给的秒数和给出来数据改变个数发生，那么将保存在数据。案例如下：比如900秒至少1个key改变；300秒至少10个可以改变；60秒至少1000个key数据改变]

# In the example below the behaviour will be to save:

# after 900 sec (15 min) if at least 1 key changed

# after 300 sec (5 min) if at least 10 keys changed

# after 60 sec if at least 10000 keys changed

#

[注意：你可以注释掉所有的save行来不保存数据。或可以通过保存指令删除之前配置的保存，使用save ""]

# Note: you can disable saving completely by commenting out all "save" lines.It is also possible to remove all the previously configured save points by adding a save directive with a single empty string argument like in the following example:

# save ""

[900秒内至少一个key对应的数据发生改变，就进行持久化操作；300秒内至少有10个key对应的数据发生改变，就进行持久化操作；60秒内至少10000个key对应的数据发生改变，就进行持久化操作。除了改变，还有新增操作]

save 900 1

save 300 10

save 60 10000

[默认情况下，如果RDB模式保存快照失败，那么redis将停止进行写操作]

# By default Redis will stop accepting writes if RDB snapshots are enabled (at least one save point) and the latest background save failed.

[上面情况将将会强硬地通知使用者数据无法持久化到硬盘，如果不这么处理，可能没有人会注意到问题发生，可能会导致灾难性后果发生]

# This will make the user aware (in a hard way) that data is not persisting on disk properly, otherwise chances are that no one will notice and some disaster will happen.

[如果后台保存进程进行重启，那么redis将会自动允许进行持久化操作]

# If the background saving process will start working again Redis will automatically allow writes again.

[如果你使用了可靠的监控进程，监控redis服务器和持久化，那么如果保存发生问题，你可能依旧允许redis继续进行写操作，那么将下面的配置修改为no即可。stop-writes-on-bgsave-error no]

# However if you have setup your proper monitoring of the Redis server and persistence, you may want to disable this feature so that Redis will continue to work as usual even if there are problems with disk,permissions, and so forth.

stop-writes-on-bgsave-error yes

[使用RDB进行持久化时，使用LZF算法对字符串进行压缩，默认如此操作，并且是个双赢，为何？不知道。可能节省空间]

# Compress string objects using LZF when dump.rdb databases? For default that's set to 'yes' as it's almost always a win.

[如果想保存子进程来节省CPU时，那么可以设置为no，此时保存的压缩数据key和value会比较大。个人认为在主从时候使用]

# If you want to save some CPU in the saving child set it to 'no' but the dataset will likely be bigger if you have compressible values or keys.

rdbcompression yes

[在RDB文件末尾使用CRC64对RDB文件进行有效性校验，这会对性能造成10%左右的影响。因此在实现最大性能时可以禁止对RDB文件进行校验]

# Since version 5 of RDB a CRC64 checksum is placed at the end of the file. This makes the format more resistant to corruption but there is a performance hit to pay (around 10%) when saving and loading RDB files, so you can disable it for maximum performances.

# RDB files created with checksum disabled have a checksum of zero that will tell the loading code to skip the check.

rdbchecksum yes

[RDB文件保存位置和文件名称]

# The filename where to dump the DB

dbfilename dump.rdb

[工作目录，数据将会写入这个目录，带有指定文件名，比如上面我们指定的'dbfilename']

# The working directory.

# The DB will be written inside this directory, with the filename specified above using the 'dbfilename' configuration directive.

[AOF文件同样在这个目录中被创建，注意：你必须在此指定目录，而不是文件名。默认是当前目录]

# The Append Only File will also be created inside this directory. Note that you must specify a directory here, not a file name.

dir ./

## 06.主从复制

###############REPLICATION [主从复制]################

# Master-Slave replication. Use slaveof to make a Redis instance a copy of another Redis server. A few things to understand ASAP about Redis replication.

[主从复制，使用slaveof命令使1个redis实例去复制另1个redis服务器数据。ASAP?尽快，还是加速器？]

#

# 1) Redis replication is asynchronous, but you can configure a master to stop accepting writes if it appears to be not connected with at least a given number of slaves.

[1.Redis复制是异步的，当至少有一个slave连接master进行复制时，你可以在配置文件中设置master停止写入]

# 2) Redis slaves are able to perform a partial resynchronization with the master if the replication link is lost for a relatively small amount of time. You may want to configure the replication backlog size (see the next sections of this file) with a sensible value depending on your needs.

[2.slaves和master如果在短时间内容失去连接，那么slaves有能力重新和master建议异步复制连接。你可以根据需要在后面的配置中进行该项配置]

# 3) Replication is automatic and does not need user intervention. After a network partition slaves automatically try to reconnect to masters and resynchronize with them.

[3.复制是自动进行的，不需要人工干预。slaves会自动和master进行连接，如果断了连接会自动尝试连接主redis]

# slaveof <masterip> <masterport>

[比如： saveof 127.0.0.1 6378]

# If the master is password protected (using the "requirepass" configuration directive below) it is possible to tell the slave to authenticate before starting the replication synchronization process, otherwise the master will refuse the slave request.

[如果主Redis设置密码认证，那么从Redis开始复制前需要使用主Redis认证，否则将会拒绝从Redis的复制请求。通俗地说，master进行认证，那么salve将会使用master的认证去进行复制请求]

# masterauth <master-password>

[此处针对于读写分离处理情况]

# When a slave loses its connection with the master, or when the replication is still in progress, the slave can act in two different ways:

[如果从Redis失去和主Redis连接，或在已经进行复制时，从Redis将有两种运行方式：]

# 1) if slave-serve-stale-data is set to 'yes' (the default) the slave will still reply to client requests, possibly with out of date data, or the data set may just be empty if this is the first synchronization.

[1.如果设置if slave-serve-stale-data为yes，那么从Redis会会使用过时数据或者第一次异步复制过来的数据或者空字符串来恢复客户端请求。]

# 2) if slave-serve-stale-data is set to 'no' the slave will reply with an error "SYNC with master in progress" to all the kind of commands but to INFO and SLAVEOF.

[2.如果设置if slave-serve-stale-data为no，那么除了INFO和SLAVEOF命令外，对其他命令回复"SYNC with master in progress"]

slave-serve-stale-data yes

[从Redis接收写操作]

[你可以配置slave实例接受写操作，向slave实例短暂写入有用的数据是有用的，但是如果客户端向它写入数据可能因为配置导致问题的发生]

# You can configure a slave instance to accept writes or not. Writing against a slave instance may be useful to store some ephemeral data (because data written on a slave will be easily deleted after resync with the master) but may also cause problems if clients are writing to it because of a misconfiguration.

#

# Since Redis 2.6 by default slaves are read-only.

[从2.6版本slaves默认设置为只读]

# Note: read only slaves are not designed to be exposed to untrusted clients on the internet. It's just a protection layer against misuse of the instance.

[注意：只读的从库模式设计暴露给不信任的网络客户端，它知识对错误使用redis实例的保护措施而已]

# still a read only slave exports by default all the administrative commands such as CONFIG, DEBUG, and so forth. To a limited extent you can improve security of read only slaves using 'rename-command' to shadow all the administrative / dangerous commands.

[默认情况下，从库作为从属输出，比如config和debug等等，为安全着想，你可以使用rename-command来重命名危险的命令]

slave-read-only yes

[同步复制策略：硬盘或者套接字]

# Replication SYNC strategy: disk or socket.

#

-------------------------------------------------------

# WARNING: DISKLESS REPLICATION IS EXPERIMENTAL CURRENTLY

-------------------------------------------------------

[注意：无盘同步复制目前只是实验]以下没有翻译整理

# New slaves and reconnecting slaves that are not able to continue the replication process just receiving differences, need to do what is called a "full synchronization". An RDB file is transmitted from the master to the slaves.

# The transmission can happen in two different ways:

# 1) Disk-backed: The Redis master creates a new process that writes the RDB

# file on disk. Later the file is transferred by the parent process to the slaves incrementally.

# 2) Diskless: The Redis master creates a new process that directly writes the

# RDB file to slave sockets, without touching the disk at all.With disk-backed replication, while the RDB file is generated, more slaves can be queued and served with the RDB file as soon as the current child producing the RDB file finishes its work. With diskless replication instead once the transfer starts, new slaves arriving will be queued and a new transfer will start when the current one terminates.

# When diskless replication is used, the master waits a configurable amount of time (in seconds) before starting the transfer in the hope that multiple slaves will arrive and the transfer can be parallelized. With slow disks and fast (large bandwidth) networks, diskless replication works better.

repl-diskless-sync no

# When diskless replication is enabled, it is possible to configure the delay the server waits in order to spawn the child that transfers the RDB via socket to the slaves.

# This is important since once the transfer starts, it is not possible to serve new slaves arriving, that will be queued for the next RDB transfer, so the server waits a delay in order to let more slaves arrive.

# The delay is specified in seconds, and by default is 5 seconds. To disable it entirely just set it to 0 seconds and the transfer will start ASAP.

repl-diskless-sync-delay 5

# slaves send PINGs to server in a predefined interval. It's possible to interval with the repl\_ping\_slave\_period option. The default value is 10 seconds.

[从Redis会在预定时间间隔内发送PINGs给主Redis，使用repl\_ping\_slave\_period时间间隔配置来进行发送PING，默认值目前是10秒]

# repl-ping-slave-period 10

[以下选项设置复制的连接超时设置]

# The following option sets the replication timeout for:

# 1) Bulk transfer I/O during SYNC, from the point of view of slave.

# 2) Master timeout from the point of view of slaves (data, pings).

# 3) Slave timeout from the point of view of masters (REPLCONF ACK pings).

[master检测salve上次发送请求的时间超过repl-timeout时间，认为salve离线，清除该salve信息；salve检测到上次和master交互时间超过repl-timeout时间，认为master离线]

#

# It is important to make sure that this value is greater than the value specified for repl-ping-slave-period otherwise a timeout will be detected every time there is low traffic between the master and the slave.

[确认repl-timeout数据必须大于repl-ping-slave-period(上面的配置介绍)值，否则salve会一直检测到master离线]

# repl-timeout 60

[是否禁止复制tcp链接的tcp nodelay参数，可传递yes或者no。默认是no，即使用tcp nodelay。如果master设置了yes来禁止tcp nodelay设置，在把数据复制给slave的时候，会减少包的数量和更小的网络带宽。但是这也可能带来数据的延迟。默认我们推荐更小的延迟，但是在数据量传输很大的场景下，建议选择yes。

数据传输量大的时候使用yes可以减少带宽和包数量；

数据量小选择默认no，可以减少数据延迟]

# Disable TCP\_NODELAY on the slave socket after SYNC?

#

# If you select "yes" Redis will use a smaller number of TCP packets and less bandwidth to send data to slaves. But this can add a delay for the data to appear on the slave side, up to 40 milliseconds with Linux kernels using a default configuration.

#

# If you select "no" the delay for data to appear on the slave side will be reduced but more bandwidth will be used for replication.

# By default we optimize for low latency, but in very high traffic conditions or when the master and slaves are many hops away, turning this to "yes" may be a good idea.

repl-disable-tcp-nodelay no

[设置复制缓存区大小

当salves和master断开一些时间，缓存区会进行数据缓存。当从库需要重新和主库连接时，通常不需要完整的同步复制连接，局部的同步足够，仅仅传递和主机失去连接时的数据]

# Set the replication backlog size. The backlog is a buffer that accumulates slave data when slaves are disconnected for some time, so that when a slave wants to reconnect again, often a full resync is not needed, but a partial resync is enough, just passing the portion of data the slave missed while disconnected.

[更大的复制缓冲区、更长的断开时间、更晚局部数据再同步]

# The bigger the replication backlog, the longer the time the slave can be disconnected and later be able to perform a partial resynchronization.

#

# The backlog is only allocated once there is at least a slave connected.

[复制缓冲区只可以设置1次，当1个从库和主库进行连接时，默认设置1M，并且不会开启，如果没有从库连接时，不会设置复制缓存区]

# repl-backlog-size 1mb

# After a master has no longer connected slaves for some time, the backlog will be freed.The following option configures the amount of seconds that need to elapse, starting from the time the last slave disconnected, for the backlog buffer to be freed.

# A value of 0 means to never release the backlog.

#

# repl-backlog-ttl 3600

# The slave priority is an integer number published by Redis in the INFO output.

# It is used by Redis Sentinel in order to select a slave to promote into a master if the master is no longer working correctly.

#

# A slave with a low priority number is considered better for promotion, so for instance if there are three slaves with priority 10, 100, 25 Sentinel will pick the one with priority 10, that is the lowest.

#

# However a special priority of 0 marks the slave as not able to perform the role of master, so a slave with priority of 0 will never be selected by Redis Sentinel for promotion.

#

# By default the priority is 100.

slave-priority 100

# It is possible for a master to stop accepting writes if there are less than N slaves connected, having a lag less or equal than M seconds.

#

# The N slaves need to be in "online" state.

# The lag in seconds, that must be <= the specified value, is calculated from the last ping received from the slave, that is usually sent every second.

#

# This option does not GUARANTEE that N replicas will accept the write, but will limit the window of exposure for lost writes in case not enough slaves are available, to the specified number of seconds.

#

# For example to require at least 3 slaves with a lag <= 10 seconds use:

# min-slaves-to-write 3

# min-slaves-max-lag 10

#

# Setting one or the other to 0 disables the feature.

#

# By default min-slaves-to-write is set to 0 (feature disabled) and min-slaves-max-lag is set to 10.

# A Redis master is able to list the address and port of the attached slaves in different ways. For example the "INFO replication" section

# offers this information, which is used, among other tools, by

# Redis Sentinel in order to discover slave instances.

# Another place where this info is available is in the output of the "ROLE" command of a masteer.

#

# The listed IP and address normally reported by a slave is obtained in the following way:

#

# IP: The address is auto detected by checking the peer address of the socket used by the slave to connect with the master.

#

# Port: The port is communicated by the slave during the replication handshake, and is normally the port that the slave is using to list for connections.

#

# However when port forwarding or Network Address Translation (NAT) is used, the slave may be actually reachable via different IP and port pairs. The following two options can be used by a slave in order to report to its master a specific set of IP and port, so that both INFO and ROLE will report those values.

#

# There is no need to use both the options if you need to override just the port or the IP address.

#

# slave-announce-ip 5.5.5.5

# slave-announce-port 1234

## 07.安全

##################### SECURITY [安全]###################

# Require clients to issue AUTH <PASSWORD> before processing any other commands. This might be useful in environments in which you do not trust others with access to the host running redis-server.

#

# This should stay commented out for backward compatibility and because most people do not need auth (e.g. they run their own servers).

#

# Warning: since Redis is pretty fast an outside user can try up to

# 150k passwords per second against a good box. This means that you should use a very strong password otherwise it will be very easy to break.

#

# requirepass foobared

### 命令重命名

# Command renaming.

[需要在共享环境中将危险命令进行重命名，比如把config命令需要重命名成不容易猜到的命令，同时它依旧可以被内部人员使用，但不提供给客户端使用]

# It is possible to change the name of dangerous commands in a shared environment. For instance the CONFIG command may be renamed into something hard to guess so that it will still be available for internal-use tools but not available for general clients.

#

# Example:

#rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52

[如果彻底废弃掉某个命令，那么就将该命令重命名为空字符串]

# It is also possible to completely kill a command by renaming it into an empty string:

# rename-command CONFIG ""

[请注意这些被重命名的命令会被记录到AOF文件中，如果发送给从库可能导致问题发生]

# Please note that changing the name of commands that are logged into the AOF file or transmitted to slaves may cause problems.

## 08.内存策略

##################### LIMITS[限制]##################

### 最大连接个数设置

[设置最大客户端连接数量，默认情况下设置连接数量是10000.而如果redis服务器没有对该项进行配置，将允许以特定形式进行最大连接数限制，设置连接不低于32，作为redis服务器内部连接使用]

# Set the max number of connected clients at the same time. By default this limit is set to 10000 clients, however if the Redis server is not able to configure the process file limit to allow for the specified limit the max number of allowed clients is set to the current file limit minus 32 (as Redis reserves a few file descriptors for internal uses).

[一旦连接超过最大连接数那么redis服务器会关掉所有新的连接，并且同时发送max number of clients reached的错误提示]

# Once the limit is reached Redis will close all the new connections sending an error 'max number of clients reached'.

[默认最大连接数10000，此时限制应该是最大连接是32]

# maxclients 10000

### redis最大内存设置

[使用内存要超过特定大小，如果内存使用已经达到redis上限，redis将会根据已经选择的收回策略来删除keys，参考maxmemory-policy]

# Don't use more memory than the specified amount of bytes.

# When the memory limit is reached Redis will try to remove keys according to the eviction policy selected (see maxmemory-policy).

[如果redis不能根据回收策略删除keys。或者回收策略使用noeviction，当redis要执行更加占用内存的命令比如set、lpush等，那么会进行回复错误信息，同时会继续回复类似get的只读命令][参考翻译，可能翻译不准确]

# If Redis can't remove keys according to the policy, or if the policy is set to 'noeviction', Redis will start to reply with errors to commands that would use more memory, like SET, LPUSH, and so on, and will continue to reply to read-only commands like GET.

[当使用redis作为LRU缓存或者为某个实例设置内存限制是，该选项是有用的。比如当使用noeviction回收策略]

# This option is usually useful when using Redis as an LRU cache, or to set a hard memory limit for an instance (using the 'noeviction' policy).

[以下内从涉及主动问题，暂时没有翻译处理]

# WARNING: If you have slaves attached to an instance with maxmemory on, the size of the output buffers needed to feed the slaves are subtracted from the used memory count, so that network problems / resyncs will not trigger a loop where keys are evicted, and in turn the output buffer of slaves is full with DELs of keys evicted triggering the deletion of more keys, and so forth until the database is completely emptied.

# In short... if you have slaves attached it is suggested that you set a lower limit for maxmemory so that there is some free RAM on the system for slave output buffers (but this is not needed if the policy is 'noeviction').

# maxmemory <bytes>

### redis回收策略

[最大内存策略：当redis内存使用达到maxmemory时，reids如何选择哪些keys删除呢]

# MAXMEMORY POLICY: how Redis will select what to remove when maxmemory is reached. You can select among five behaviors:

#

# volatile-lru -> remove the key with an expire set using an LRU algorithm

[使用LRU算法删除设置过期时间的key]

#allkeys-lru -> remove any key according to the LRU algorithm

[使用LRU算法删除任何key]

# volatile-random -> remove a random key with an expire set

[随机删除设置过期时间的key]

# allkeys-random -> remove a random key, any key

[随机删除任意1个key]

# volatile-ttl -> remove the key with the nearest expire time (minor TTL)

[删除即将到期的key、辅以ttl]

# noeviction -> don't expire at all, just return an error on write operations

[不删除任何一个key，仅仅返回一个写错误]

# Note: with any of the above policies, Redis will return an error on write operations, when there are no suitable keys for eviction.

[注意：在使用上述任意回收策略，当在执行写操作，没有适合的key被回收时，redis将会返回错误提醒内容]

[这些写操作命令如下：]

# At the date of writing these commands are: set setnx setex append incr decr rpush lpush rpushx lpushx linsert lset rpoplpush sadd sinter sinterstore sunion sunionstore sdiff sdiffstore zadd zincrby zunionstore zinterstore hset hsetnx hmset hincrby incrby decrby getset mset msetnx exec sort

# The default is:

# maxmemory-policy noeviction

# LRU and minimal TTL algorithms are not precise algorithms but approximated algorithms (in order to save memory), so you can tune it for speed or accuracy. For default Redis will check five keys and pick the one that was used less recently, you can change the sample size using the following configuration directive.

#

# The default of 5 produces good enough results. 10 Approximates very closely true LRU but costs a bit more CPU. 3 is very fast but not very accurate.

# maxmemory-samples 5

## 09.AOF模式

############ APPEND ONLY MODE[AOF模式]##############

[根本目的是为什么使用AOF模式：默认情况下redis使用RDB模式将数据异步存储到硬盘上，这已经足够应付大部分应用。但是会存在使用redis过程中途宕机，那么会导致几分钟内数据丢失问题。]

# By default Redis asynchronously dumps the dataset on disk. This mode is good enough in many applications, but an issue with the Redis process or a power outage may result into a few minutes of writes lost (depending on the configured save points).

[AOF文件是另一种持久化模式，它提供更好的易用性。如果将aof作为数据持久化策略，那么Redis仅仅会丢失诸如服务器宕机最后1秒的数据]

# The Append Only File is an alternative persistence mode that provides much better durability. For instance using the default data fsync policy (see later in the config file) Redis can lose just one second of writes in a dramatic event like a server power outage, or a single write if something wrong with the Redis process itself happens, but the operating system is still running correctly.

[aof和rdb持久化策略可以同时使用，而不会发生任何问题]

# AOF and RDB persistence can be enabled at the same time without problems.

[如果在启动时默认开启aof模式，那么redis会加载aof文件，其中aof文件提供更好的持久化担保，丢失数据的可能性小]

# If the AOF is enabled on startup Redis will load the AOF, that is the file with the better durability guarantees.

# Please check http://redis.io/topics/persistence for more information.

[如果打算开启aof模式，那么将no修改为yes即可]

appendonly no

[修改aof文件名称，默认是appendonly.aof]

# The name of the append only file

# (default: "appendonly.aof")

appendfilename "appendonly.aof"

[异步通知操作系统及时将输入写到磁盘中，而不是在输入缓存中等待更多的数据写入写入后再将数据写入磁盘中。通常某些系统可以及时将数据持久化到硬盘上，有些系统却会将数据保存在系统的缓存中一段时间然后才将数据持久化到硬盘中]

# The fsync() call tells the Operating System to actually write data on disk instead of waiting for more data in the output buffer. Some OS will really flush data on disk, some other OS will just try to do it ASAP.

[redis提供三种解决方案：

no:不异步通知，仅仅等待操作系统将自身数据缓存进行持久化，速度快；always:每次写命令都会进行写入到aof文件中，速度慢、最安全；everysec：每秒内将通知操作系统将缓存的命令写入到aof文件，兼顾速度和安全]

# Redis supports three different modes:

# no: don't fsync, just let the OS flush the data when it wants. Faster.

# always: fsync after every write to the append only log. Slow, Safest.

# everysec: fsync only one time every second. Compromise.

[默认使用everysec，该模式可以兼顾速度和安全。]

# The default is "everysec", as that's usually the right compromise between speed and data safety. It's up to you to understand if you can relax this to

# "no" that will let the operating system flush the output buffer when it wants, for better performances (but if you can live with the idea of some data loss consider the default persistence mode that's snapshotting), or on the contrary, use "always" that's very slow but a bit safer than everysec.

# More details please check the following article:

# http://antirez.com/post/redis-persistence-demystified.html

# If unsure, use "everysec".

# appendfsync always

appendfsync everysec

# appendfsync no

# When the AOF fsync policy is set to always or everysec, and a background saving process (a background save or AOF log background rewriting) is performing a lot of I/O against the disk, in some Linux configurations

# Redis may block too long on the fsync() call. Note that there is no fix for this currently, as even performing fsync in a different thread will block our synchronous write(2) call.

#

# In order to mitigate this problem it's possible to use the following option that will prevent fsync() from being called in the main process while a BGSAVE or BGREWRITEAOF is in progress.

#

# This means that while another child is saving, the durability of Redis is the same as "appendfsync none". In practical terms, this means that it is possible to lose up to 30 seconds of log in the worst scenario (with the default Linux settings).

#

# If you have latency problems turn this to "yes". Otherwise leave it as "no" that is the safest pick from the point of view of durability.

no-appendfsync-on-rewrite no

# Automatic rewrite of the append only file.

# Redis is able to automatically rewrite the log file implicitly calling

# BGREWRITEAOF when the AOF log size grows by the specified percentage.

#

# This is how it works: Redis remembers the size of the AOF file after the latest rewrite (if no rewrite has happened since the restart, the size of the AOF at startup is used).

#

# This base size is compared to the current size. If the current size is bigger than the specified percentage, the rewrite is triggered. Also you need to specify a minimal size for the AOF file to be rewritten, this is useful to avoid rewriting the AOF file even if the percentage increase is reached but it is still pretty small.

#

# Specify a percentage of zero in order to disable the automatic AOF rewrite feature.

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

# An AOF file may be found to be truncated at the end during the Redis startup process, when the AOF data gets loaded back into memory.

# This may happen when the system where Redis is running

# crashes, especially when an ext4 filesystem is mounted without the

# data=ordered option (however this can't happen when Redis itself crashes or aborts but the operating system still works correctly).

#

# Redis can either exit with an error when this happens, or load as much data as possible (the default now) and start if the AOF file is found to be truncated at the end. The following option controls this behavior.

#

# If aof-load-truncated is set to yes, a truncated AOF file is loaded and the Redis server starts emitting a log to inform the user of the event.

# Otherwise if the option is set to no, the server aborts with an error and refuses to start. When the option is set to no, the user requires

# to fix the AOF file using the "redis-check-aof" utility before to restart the server.

#

# Note that if the AOF file will be found to be corrupted in the middle the server will still exit with an error. This option only applies when Redis will try to read more data from the AOF file but not enough bytes will be found.

aof-load-truncated yes

##################### LUA SCRIPTING ###################

# Max execution time of a Lua script in milliseconds.

#

# If the maximum execution time is reached Redis will log that a script is still in execution after the maximum allowed time and will start to reply to queries with an error.

#

# When a long running script exceeds the maximum execution time only the

# SCRIPT KILL and SHUTDOWN NOSAVE commands are available. The first can be

# used to stop a script that did not yet called write commands. The second is the only way to shut down the server in the case a write command was already issued by the script but the user doesn't want to wait for the natural

# termination of the script.

#

# Set it to 0 or a negative value for unlimited execution without warnings.

lua-time-limit 5000

##################### REDIS CLUSTER ###################

# WARNING EXPERIMENTAL: Redis Cluster is considered to be stable code, however in order to mark it as "mature" we need to wait for a non trivial percentage of users to deploy it in production.

#

# Normal Redis instances can't be part of a Redis Cluster; only nodes that are started as cluster nodes can. In order to start a Redis instance as a

# cluster node enable the cluster support uncommenting the following:

#

# cluster-enabled yes

# Every cluster node has a cluster configuration file. This file is not intended to be edited by hand. It is created and updated by Redis nodes.

# Every Redis Cluster node requires a different cluster configuration file.

# Make sure that instances running in the same system do not have overlapping cluster configuration file names.

#

# cluster-config-file nodes-6379.conf

# Cluster node timeout is the amount of milliseconds a node must be unreachable for it to be considered in failure state.

# Most other internal time limits are multiple of the node timeout.

# cluster-node-timeout 15000

# A slave of a failing master will avoid to start a failover if its data looks too old.

# There is no simple way for a slave to actually have a exact measure of its "data age", so the following two checks are performed:

#

# 1) If there are multiple slaves able to failover, they exchange messages

# in order to try to give an advantage to the slave with the best replication offset (more data from the master processed).

# Slaves will try to get their rank by offset, and apply to the start of the failover a delay proportional to their rank.

#

# 2) Every single slave computes the time of the last interaction with

# its master. This can be the last ping or command received (if the master

# is still in the "connected" state), or the time that elapsed since the

# disconnection with the master (if the replication link is currently down).

# If the last interaction is too old, the slave will not try to failover

# at all.

#

# The point "2" can be tuned by user. Specifically a slave will not perform the failover if, since the last interaction with the master, the time

# elapsed is greater than:

#

# (node-timeout \* slave-validity-factor) + repl-ping-slave-period

#

# So for example if node-timeout is 30 seconds, and the slave-validity-factor

# is 10, and assuming a default repl-ping-slave-period of 10 seconds, the

# slave will not try to failover if it was not able to talk with the master for longer than 310 seconds.

#

# A large slave-validity-factor may allow slaves with too old data to failover a master, while a too small value may prevent the cluster from being able to elect a slave at all.

#

# For maximum availability, it is possible to set the slave-validity-factor

# to a value of 0, which means, that slaves will always try to failover the master regardless of the last time they interacted with the master.

# (However they'll always try to apply a delay proportional to their offset rank).

#

# Zero is the only value able to guarantee that when all the partitions heal

# the cluster will always be able to continue.

#

# cluster-slave-validity-factor 10

# Cluster slaves are able to migrate to orphaned masters, that are masters

# that are left without working slaves. This improves the cluster ability

# to resist to failures as otherwise an orphaned master can't be failed over in case of failure if it has no working slaves.

#

# Slaves migrate to orphaned masters only if there are still at least a

# given number of other working slaves for their old master. This number

# is the "migration barrier". A migration barrier of 1 means that a slave

# will migrate only if there is at least 1 other working slave for its master

# and so forth. It usually reflects the number of slaves you want for every

# master in your cluster.

#

# Default is 1 (slaves migrate only if their masters remain with at least

# one slave). To disable migration just set it to a very large value.

# A value of 0 can be set but is useful only for debugging and dangerous

# in production.

#

# cluster-migration-barrier 1

# By default Redis Cluster nodes stop accepting queries if they detect there

# is at least an hash slot uncovered (no available node is serving it).

# This way if the cluster is partially down (for example a range of hash slots are no longer covered) all the cluster becomes, eventually, unavailable.

# It automatically returns available as soon as all the slots are covered again.

#

# However sometimes you want the subset of the cluster which is working, to continue to accept queries for the part of the key space that is still covered. In order to do so, just set the cluster-require-full-coverage

# option to no.

#

# cluster-require-full-coverage yes

# In order to setup your cluster make sure to read the documentation

# available at http://redis.io web site.

################## SLOW LOG ########################

# The Redis Slow Log is a system to log queries that exceeded a specified

# execution time. The execution time does not include the I/O operations

# like talking with the client, sending the reply and so forth,

# but just the time needed to actually execute the command (this is the only stage of command execution where the thread is blocked and can not serve other requests in the meantime).

#

# You can configure the slow log with two parameters: one tells Redis what is the execution time, in microseconds, to exceed in order for the command to get logged, and the other parameter is the length of the slow log. When a new command is logged the oldest one is removed from the

# queue of logged commands.

# The following time is expressed in microseconds, so 1000000 is equivalent to one second. Note that a negative number disables the slow log, while

# a value of zero forces the logging of every command.

slowlog-log-slower-than 10000

# There is no limit to this length. Just be aware that it will consume memory.

# You can reclaim memory used by the slow log with SLOWLOG RESET.

slowlog-max-len 128

####################### LATENCY MONITOR ################

# The Redis latency monitoring subsystem samples different operations at runtime in order to collect data related to possible sources of latency of a Redis instance.

#

# Via the LATENCY command this information is available to the user that can print graphs and obtain reports.

#

# The system only logs operations that were performed in a time equal or

# greater than the amount of milliseconds specified via the

# latency-monitor-threshold configuration directive. When its value is set to zero, the latency monitor is turned off.

#

# By default latency monitoring is disabled since it is mostly not needed if you don't have latency issues, and collecting data has a performance

# impact, that while very small, can be measured under big load. Latency

# monitoring can easily be enabled at runtime using the command

# "CONFIG SET latency-monitor-threshold <milliseconds>" if needed.

latency-monitor-threshold 0

############# EVENT NOTIFICATION ###################

# Redis can notify Pub/Sub clients about events happening in the key space.

# This feature is documented at http://redis.io/topics/notifications

#

# For instance if keyspace events notification is enabled, and a client

# performs a DEL operation on key "foo" stored in the Database 0, two messages will be published via Pub/Sub:

#

# PUBLISH \_\_keyspace@0\_\_:foo del

# PUBLISH \_\_keyevent@0\_\_:del foo

#

# It is possible to select the events that Redis will notify among a set

# of classes. Every class is identified by a single character:

#

# K Keyspace events, published with \_\_keyspace@<db>\_\_ prefix.

# E Keyevent events, published with \_\_keyevent@<db>\_\_ prefix.

# g Generic commands (non-type specific) like DEL, EXPIRE, RENAME, ...

# $ String commands

# l List commands

# s Set commands

# h Hash commands

# z Sorted set commands

# x Expired events (events generated every time a key expires)

# e Evicted events (events generated when a key is evicted for maxmemory)

# A Alias for g$lshzxe, so that the "AKE" string means all the events.

#

# The "notify-keyspace-events" takes as argument a string that is composed

# of zero or multiple characters. The empty string means that notifications

# are disabled.

#

# Example: to enable list and generic events, from the point of view of the

# event name, use:

#

# notify-keyspace-events Elg

#

# Example 2: to get the stream of the expired keys subscribing to channel

# name \_\_keyevent@0\_\_:expired use:

#

# notify-keyspace-events Ex

#

# By default all notifications are disabled because most users don't need this feature and the feature has some overhead. Note that if you don't specify at least one of K or E, no events will be delivered.

notify-keyspace-events ""

##################### ADVANCED CONFIG #################

# Hashes are encoded using a memory efficient data structure when they have a

# small number of entries, and the biggest entry does not exceed a given

# threshold. These thresholds can be configured using the following directives.

hash-max-ziplist-entries 512

hash-max-ziplist-value 64

# Lists are also encoded in a special way to save a lot of space.

# The number of entries allowed per internal list node can be specified

# as a fixed maximum size or a maximum number of elements.

# For a fixed maximum size, use -5 through -1, meaning:

# -5: max size: 64 Kb <-- not recommended for normal workloads

# -4: max size: 32 Kb <-- not recommended

# -3: max size: 16 Kb <-- probably not recommended

# -2: max size: 8 Kb <-- good

# -1: max size: 4 Kb <-- good

# Positive numbers mean store up to \_exactly\_ that number of elements

# per list node.

# The highest performing option is usually -2 (8 Kb size) or -1 (4 Kb size),

# but if your use case is unique, adjust the settings as necessary.

list-max-ziplist-size -2

# Lists may also be compressed.

# Compress depth is the number of quicklist ziplist nodes from \*each\* side of

# the list to \*exclude\* from compression. The head and tail of the list

# are always uncompressed for fast push/pop operations. Settings are:

# 0: disable all list compression

# 1: depth 1 means "don't start compressing until after 1 node into the list,

# going from either the head or tail"

# So: [head]->node->node->...->node->[tail]

# [head], [tail] will always be uncompressed; inner nodes will compress.

# 2: [head]->[next]->node->node->...->node->[prev]->[tail]

# 2 here means: don't compress head or head->next or tail->prev or tail,

# but compress all nodes between them.

# 3: [head]->[next]->[next]->node->node->...->node->[prev]->[prev]->[tail]

# etc.

list-compress-depth 0

# Sets have a special encoding in just one case: when a set is composed

# of just strings that happen to be integers in radix 10 in the range

# of 64 bit signed integers.

# The following configuration setting sets the limit in the size of the

# set in order to use this special memory saving encoding.

set-max-intset-entries 512

# Similarly to hashes and lists, sorted sets are also specially encoded in

# order to save a lot of space. This encoding is only used when the length and

# elements of a sorted set are below the following limits:

zset-max-ziplist-entries 128

zset-max-ziplist-value 64

# HyperLogLog sparse representation bytes limit. The limit includes the

# 16 bytes header. When an HyperLogLog using the sparse representation crosses

# this limit, it is converted into the dense representation.

#

# A value greater than 16000 is totally useless, since at that point the

# dense representation is more memory efficient.

#

# The suggested value is ~ 3000 in order to have the benefits of

# the space efficient encoding without slowing down too much PFADD,

# which is O(N) with the sparse encoding. The value can be raised to

# ~ 10000 when CPU is not a concern, but space is, and the data set is

# composed of many HyperLogLogs with cardinality in the 0 - 15000 range.

hll-sparse-max-bytes 3000

# Active rehashing uses 1 millisecond every 100 milliseconds of CPU time in order to help rehashing the main Redis hash table (the one mapping top-level

# keys to values). The hash table implementation Redis uses (see dict.c)

# performs a lazy rehashing: the more operation you run into a hash table

# that is rehashing, the more rehashing "steps" are performed, so if the

# server is idle the rehashing is never complete and some more memory is used

# by the hash table.

#

# The default is to use this millisecond 10 times every second in order to

# actively rehash the main dictionaries, freeing memory when possible.

#

# If unsure:

# use "activerehashing no" if you have hard latency requirements and it is

# not a good thing in your environment that Redis can reply from time to time

# to queries with 2 milliseconds delay.

#

# use "activerehashing yes" if you don't have such hard requirements but want to free memory asap when possible.

activerehashing yes

# The client output buffer limits can be used to force disconnection of clients

# that are not reading data from the server fast enough for some reason (a

# common reason is that a Pub/Sub client can't consume messages as fast as the publisher can produce them).

#

# The limit can be set differently for the three different classes of clients:

#

# normal -> normal clients including MONITOR clients

# slave -> slave clients

# pubsub -> clients subscribed to at least one pubsub channel or pattern

#

# The syntax of every client-output-buffer-limit directive is the following:

#

# client-output-buffer-limit <class> <hard limit> <soft limit> <soft seconds>

#

# A client is immediately disconnected once the hard limit is reached, or if

# the soft limit is reached and remains reached for the specified number of seconds (continuously).

# So for instance if the hard limit is 32 megabytes and the soft limit is

# 16 megabytes / 10 seconds, the client will get disconnected immediately

# if the size of the output buffers reach 32 megabytes, but will also get disconnected if the client reaches 16 megabytes and continuously overcomes

# the limit for 10 seconds.

#

# By default normal clients are not limited because they don't receive data

# without asking (in a push way), but just after a request, so only

# asynchronous clients may create a scenario where data is requested faster than it can read.

#

# Instead there is a default limit for pubsub and slave clients, since subscribers and slaves receive data in a push fashion.

#

# Both the hard or the soft limit can be disabled by setting them to zero.

client-output-buffer-limit normal 0 0 0

client-output-buffer-limit slave 256mb 64mb 60

client-output-buffer-limit pubsub 32mb 8mb 60

# Redis calls an internal function to perform many background tasks, like

# closing connections of clients in timeout, purging expired keys that are never requested, and so forth.

#

# Not all tasks are performed with the same frequency, but Redis checks for tasks to perform according to the specified "hz" value.

#

# By default "hz" is set to 10. Raising the value will use more CPU when

# Redis is idle, but at the same time will make Redis more responsive when there are many keys expiring at the same time, and timeouts may be handled with more precision.

#

# The range is between 1 and 500, however a value over 100 is usually not a good idea. Most users should use the default of 10 and raise this up to 100 only in environments where very low latency is required.

hz 10

# When a child rewrites the AOF file, if the following option is enabled

# the file will be fsync-ed every 32 MB of data generated. This is useful

# in order to commit the file to the disk more incrementally and avoid

# big latency spikes.

aof-rewrite-incremental-fsync yes