```
# Redis示例配置文件.
#
# 注：为了读取此配置文件，Redis必须将此文件路径设为第一个参数启动：
#
# ./redis-server /path/to/redis.conf

# 关于单位的注意事项：当需要设置内存大小时，可以指定
# 通用的的1k 5GB 4M格式，依此类推：
#
# 1k => 1000 bytes
# 1kb => 1024 bytes
# 1m => 1000000 bytes
# 1mb => 1024*1024 bytes
# 1g => 1000000000 bytes
# 1gb => 1024*1024*1024 bytes
#
# 单位不区分大小写，因此1GB 1Gb 1gB都相同。

################################## INCLUDES ###################################

# 使用include包含一个或多个配置文件。
# 如果你有用于所有Redis服务器的标准配置模板，
# 但是也需要对少数服务器进行自定义配置的话这是十分有用的。
# include可以包含其他配置文件，所以需要谨慎使用它。
#
# 注意："include"选项不会被管理员后Sentinel的"CONFIG REWRITE"命令重写。
# 由于Redis始终将最后处理的行用作配置指令的值，因此最好将include放在此
# 配置文件的开头，以避免在运行时配置被覆盖。
#
# 相反，如果您有兴趣使用include覆盖配置选项，则最好将include作为最后一行。
#
# include /path/to/local.conf
# include /path/to/other.conf

################################## MODULES #####################################

# 在启动时加载模块(modules)。如果服务器无法加载模块则会放弃加载。
# 我们可以使用多个loadmodule命令
#
# loadmodule /path/to/my_module.so
# loadmodule /path/to/other_module.so

################################## NETWORK #####################################

# 默认情况下，如果未指定"bind"配置指令，则Redis侦听来自服务器上
# 所有可用网络接口的连接。可以使用“ bind”配置指令仅侦听一个或多
# 个所选接口，然后侦听一个或多个IP地址。
#
# 示例:
#
# bind 192.168.1.100 10.0.0.1
# bind 127.0.0.1 ::1
#
# ~~~ 警告 ~~~ 如果允许Redis的计算机直接暴露于互联网，绑定所有接口是危险的，
# 并且会把实例暴露给互联网上的所有人。因此，默认情况下，我们取消注释bind指令
```

```
# ，这将强制Redis仅监听IPv4回环接口地址（这意味着Redis将只能接受来自运行在同一
# 计算机上的客户端连接）
#
#
# 如果你确定想让你的实例监听所有端口只需要注释下面这行。
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
bind 127.0.0.1

# 保护模式是安全保护的一层，以避免利用Internet访问和打开Redis实例。
#
# 当保护模式开启时，如果：:
#
# 1) 服务器未使用"bind"
# 2) 未设置密码No password is configured.
#
# 服务器仅接受来自客户端的连接，该客户端从IPv4和IPv6回环地址
# 127.0.0.1和 ::1 ，以及Unix域套接字进行连接。
#
# 默认情况下保护模式是开启的。仅当您确定您希望其他主机的客户端（未配置身份验证）连接到Redis时，
# 才应该禁用它，也不会使用"bind"指令显式列出一组特定的接口。
protected-mode yes

# 接受指定端口上的连接，默认是6379(IANA #815344).
# 如果指定了端口0，则Redis将不会在TCP套接字上侦听。
port 6379

# TCP listen() backlog.
#
# 在每秒请求数很高的环境中，您需要大量积压，以避免客户端连接速度慢的问题。
# 请注意，Linux内核将已无提示的方式将其截断为/proc/sync/net/core/somaxconn
# 的值，因此请确保同时提高somaxconn和 tcp_max_syn_backlog的值以获取所需的效果。
tcp-backlog 511

# Unix socket.
#
# 指定用于监听进入连接的Unix socket的路径。没有默认值，所有当未指定时Redis
# 将不会再Unix socket上监听。
#
# unixsocket /tmp/redis.sock
# unixsocketperm 700

# 客户端空闲N秒后关闭连接(0为禁用)
# timeout N
timeout 0

# TCP keepalive.
#
# 如果不为0，请在没有通信的情况下使用SO_KEEPLIVE向客户端发送TCP ACK，
# 这很有用，有两个原因：
#
# 1) 检测死去的连接
# 2) Take the connection alive from the point of view of network
#    equipment in the middle.
#
# 在Linux上，指定的值（以秒为单位）是用于发送ACK的时间段。
#  注意，关闭连接需要两倍的时间。 在其他内核上，期限取决于内核配置。
#
# 此选项的合理值为300秒，这是从Redis 3.2.1开始的新Redis默认值。
```

```
tcp-keepalive 300

################################## TLS/SSL #####################################

# 默认情况下，禁用TLS / SSL。
# 要启用它，可以使用"tls-port"配置指令来定义TLS侦听端口。
# 要在默认端口上启用TLS，请使用：
#
# port 0
# tls-port 6379

# 配置X.509证书和私钥，以对连接的客户端，主服务器或群集对等服务器进行身份验证。
# 这些文件应为PEM格式。
#
# tls-cert-file redis.crt
# tls-key-file redis.key

# 配置DH参数文件以启用Diffie-Hellman（DH）密钥交换：
#
# tls-dh-params-file redis.dh

# 配置CA证书捆绑包或目录以对TLS / SSL客户端和对等方进行身份验证。
# Redis需要其中至少一项的显式配置，并且不会隐式使用系统范围的配置。
#
# tls-ca-cert-file ca.crt
# tls-ca-cert-dir /etc/ssl/certs

# 默认情况下，要求TLS端口上的客户端（包括副本服务器）使用有效
# 的客户端证书进行身份验证。
#
# 使用此指令可以禁用认证。
#
# tls-auth-clients no

# 默认情况下，Redis副本不尝试与其主服务器建立TLS连接。
#
# 使用以下指令在副本连接上启用TLS。
#
# tls-replication yes

# 默认情况下，Redis Cluster总线使用Plain TCP连接。
# 要启用总线协议，请使用以下指令：
#
# tls-cluster yes

# 明确指定要支持的TLS版本。
# 允许的值不区分大小写，包括"TLSv1"，"TLSv1.1"，"TLSv1.2"，"TLSv1.3"（OpenSSL> =
1.1.1）或任意组合。
# 要仅启用TLSv1.2和TLSv1.3，请使用：
#
# tls-protocols "TLSv1.2 TLSv1.3"

# 配置密码。有关此字符串的语法的更多信息，
# 请参见ciphers（1ssl）联机帮助页。
#
# 注意：此配置仅适用于<= TLSv1.2。
#
# tls-ciphers DEFAULT:!MEDIUM
```

```
# 配置允许的TLSv1.3密码套件。
# 有关此字符串的语法（尤其是TLSv1.3密码套件）的语法的更多信息，
# 请参见ciphers（1ssl）联机帮助页。
#
# tls-ciphersuites TLS_CHACHA20_POLY1305_SHA256

# 选择密码时，请使用服务器的首选项而不是客户端的首选项。
# 默认情况下，服务器遵循客户端的首选项。
#
# tls-prefer-server-ciphers yes

################################## GENERAL #####################################

# 默认情况下，Redis不会作为守护程序运行。 如果需要，请使用"yes"。
# 请注意，Redis守护进程将在/var/run/redis.pid中写入一个pid文件。
daemonize no

# 若从upstart或systemd运行Redis，则Redis可以与监控树交互。
#   supervised no      -无相互监督
#   supervised upstart - 使Redis进入SIGSTOP模式并监控upstart
#   supervised systemd - 通过设置READY=1到$NOTIFY_SOCKET来监控systemd
#   supervised auto    - 基于UPSTART_JOB或NOTIFY_SOCKET环境变量的监督自动检测upstart
或systemd方法
#
# 注意: 这些监督方法仅表示“过程已准备就绪”。
#       它们无法是您的监督者连续进行ping操作。
supervised no

# 如果指定了pid文件，则Redis会在启动时将其写入指定位置，然后在退出时将其删除。
#
# 当服务器以非守护进程运行时，如果在配置中未指定任何pid文件，则不会创建。
# 当服务器以守护进程运行时，即使未指定，也会使用pid文件，默认为"/var/run/redis.pid"。
#
# 创建pid是尽力而为的：如果Redis无法创建它，则不会发生任何不好的情况，服务器将正常启动并运行。
pidfile /var/run/redis_6379.pid

# 指定服务器日志级别。
# 可以是以下之一:
# debug (很多信息，对于开发/测试很有用)
# verbose (many rarely useful info, but not a mess like the debug level)
# notice (moderately verbose, what you want in production probably)
# warning (仅记录非常重要/重要的消息)
loglevel notice

# 指定日志文件名。另外，空字符串可以用来强制Redis标准输出。
# 请注意，如果您使用标准输出进行日志记录但进行守护进程，则日志将发送到/ dev / null
logfile ""

# 要启用登录到系统记录器的功能，只需将"syslog-enabled"设置为yes，然后根据需要更新其他syslog
参数。
# syslog-enabled no

# 指定系统日志标识。
# syslog-ident redis

# 指定系统日志工具。必须是USER或在LOCAL0-LOCAL7之间。
# syslog-facility local0
```

```
# 设置数据库数。默认数据库为DB 0，您可以使用SELECT <dbid>在每个连接的基础上选择一个不同的数据库，
# 其中dbid是介于0和database-1之间的数字。
databases 16

# 默认情况下，仅当开始登录到标准输出并且标准输出为TTY时，Redis才会显示ASCII艺术徽标。
# 基本上，这意味着通常仅在交互式会话中显示徽标。
#
# 但是，可以通过将以下选项设置为yes，来强制执行4.0之前的行为并始终在启动日志中显示ASCII艺术徽标。
always-show-logo yes

################################ SNAPSHOTTING  ################################
#
# 保存数据到磁盘:
#
#   save <seconds> <changes>
#
#   如果距离上一次保存数据库的时间(seconds)发生了指定的写操作次数(changes)则将保存数据库。
#
#   在下面默认配置的示例中，将会保存数据：
#   在900秒中发生了至少1次数据变化
#   在300秒中发生了至少10次数据变化
#   在60秒中发生了至少10000次数据变化
#
#   注意：您可以通过注释掉所有"保存"行来完全禁用保存。
#
#   也可以通过添加带有单个空字符串参数的save指令来删除所有先前配置的保存点，如以下示例所示：
#
#   save ""

save 900 1
save 300 10
save 60 10000

# 默认情况下，如果启用RDB快照（至少一个保存点）并且最新的background save失败，Redis将不接受写入操作。
# 这将使用户（以一种困难的方式）意识到数据无法正确地持久存储在磁盘上，否则，可能没人会注意到并且会发生一些灾难。
#
# 如果后台保存过程将再次开始工作，则Redis将自动允许再次写入。
#
# 但是，如果您设置了对Redis服务器和持久性的适当监视，则可能要禁用此功能，
# 以便即使磁盘，权限等出现问题，Redis也将继续照常工作。
stop-writes-on-bgsave-error yes

# 转储.rdb数据库时使用LZF压缩字符串对象
# 默认情况下将其设置为"yes"。
# 如果要在保存子项中保存一些CPU，请将其设置为"no"，
# 但是如果您具有可压缩的值或键，则数据集可能会更大。
rdbcompression yes

# 从RDB版本5开始，CRC64校验和位于文件的末尾。 这使该格式更能抵抗损坏，
# 但是在保存和加载RDB文件时会降低性能（约10％），因此可以禁用它以实现最佳性能。
#
# 在禁用校验和的情况下创建的RDB文件的校验和为零，这将指示加载代码跳过该校验。
rdbchecksum yes
```

```
# RDB文件的文件名
dbfilename dump.rdb

# 在未启用持久化的副本实例中删除RDB文件。默认情况下，此选项是禁用的，但是在某
# 些环境中，出于法规或其他安全方面的考虑，应将RDB文件由主数据库持久存储在磁盘
# 上以发送给副本，或将RDB文件由副本存储在磁盘上以便为初始同步加载它们，它们应
# 该被尽快删除。请注意，此选项仅在同时禁用AOF和RDB持久性的实例中起作用，否则将
# 被完全忽略。
#
# 获得相同效果的另一种方法（有时是更好的方法）是在主实例和副本实例上都使用无盘复制。
# 但是，对于副本，无盘并非始终是一种选择。
rdb-del-sync-files no

# 工作目录。
#
# 数据库将被写入该目录内，文件名使用"dbfilename"配置指令在上面指定。
#
# AOF文件也会创建在这个目录中。
#
# 注意，您必须指定一个目录，而不是文件名。
dir ./

################################# REPLICATION #################################

# 主副本复制。 使用copyof可以使Redis实例成为另一个Redis服务器的副本。
# 了解有关Redis复制的几件事。
#
#   +------------------+      +---------------+
#   |      Master      | ---> |    Replica    |
#   | (receive writes) |      |  (exact copy) |
#   +------------------+      +---------------+
#
# 1) Redis复制是异步的，但是如果它似乎未与给定数量(至少)的副本连接，则可以将主服务器配置为停止
接受写入操作。
#
# 2) 如果副本连接在相对较短的时间内丢失，则Redis副本能够与主副本执行部分重新同步。您可能要根据
需要将复制
#     积压大小（请参阅此文件的下一部分）配置为合理的值。
#
# 3) 复制是自动的，不需要用户干预。网络分区之后，副本会自动尝试重新连接到主节点并与它们重新同
步。Replication is automatic and does not need user intervention. After a
#
# replicaof <masterip> <masterport>

# 如果主服务器受密码保护(使用下面的"requirepass"配置指令)，则可以在开始复制同步过程之前告诉副
本服务器进行
# 身份验证，否则主服务器将拒绝副本请求。
#
# masterauth <master-password>
#
# 然而，这是不够的，如果使用的是Redis的访问列表（用于Redis的版本6或更大），并且默认用户不能够
运行PSYNC命令和/
# 或复制所需的其他命令。在这种情况下，最好配置一个特殊用户以用于复制，并如下配置masteruser：
#
# masteruser <username>
#
```

```
# 指定masteruser时，副本针对其主服务器进行身份验证将使用新的AUTH格式：AUTH <username>
<password>。

# 当副本断开与主数据库的连接时，或者仍在进行复制时，副本可以采取两种不同的方式进行操作：
#
# 1) 如果replica-serve-stale-data设置为"yes"（默认值），则副本仍将回复客户端请求，可能包
含过期数据，
#    或者如果这是第一次同步，则数据集可能只是空的。
#
# 2) 如果replica-serve-stale-data设置为"no"，副本将对所有除
#    INFO, replicaOF, AUTH, PING, SHUTDOWN, REPLCONF, ROLE, CONFIG,
#    SUBSCRIBE, UNSUBSCRIBE, PSUBSCRIBE, PUNSUBSCRIBE, PUBLISH, PUBSUB,
#    COMMAND, POST, HOST: and LATENCY 的命令返回"SYNC with master in progress"错误
#
replica-serve-stale-data yes

# 您可以配置副本示例接不接受写入操作。针对副本示例进行写操作可能对存储一些临时数据很有用
# (因为与主实例重新同步后，写入副本上的数据将被删除)，但是如果客户端由于配置错误而向其进
# 行写操作可能导致问题。
#
# 自Redis 2.6，默认情况下副本是只读的。
#
# 注意：只读副本并非只在向Internet上不受信任的客户端公开。它只是对实例误操作的保护层。
# 默认情况下，只读副本仍会导出所有管理命令，例如CONFIG,DEBUG等等。你可以使用'rename-
command'
# 隐藏所有管理命令/危险命令来提高只读副本的安全性
replica-read-only yes

# 副本SYNC(同步)策略: disk 或 socket.
#
# 新的和重连的副本无法进行根据差异复制的过程，需要执行"完全同步"。
# RDB文件从主数据库传输到从数据库。
#
# 有两种传播方式:
#
# 1) 磁盘复制: Redis主机创建一个新进程，将RDB文件写入磁盘。
#                                稍后，该文件由父进程以增量方式传输到副本。
# 2) 无盘复制: Redis master创建了一个新进程，直接将RDB文件写入副本套接字，而不需要接触磁盘。
#
# 使用磁盘支持的复制，在生成RDB文件时，只要当前生成RDB文件的子文件完成其工作，就可以将更多的副
本排队并与RDB文件一起提供服务。
# 如果在传输开始后使用无磁盘复制，到达的新副本将排队，当当前副本终止时，新的传输将开始。
#
# 当使用无磁盘复制时，主服务器在开始传输之前等待一段可配置的时间(以秒为单位)，希望多个副本到达并
并行传输。
#
# 对于慢磁盘和高速(大带宽)网络，无磁盘复制工作得更好。
repl-diskless-sync no
//*[@id="articleMeList-blog"]/div[2]/div[1]/h4/a
//*[@id="articleMeList-blog"]/div[2]/div[2]/h4/a
# 当启用无磁盘复制时，可以配置服务器等待的延迟，以便生成通过套接字将RDB传输到副本的子服务器。
#
# 此配置很重要，因为一旦传输开始，就不能为到达的新副本提供服务，这些副本将排队等待下一次RDB传
输，
# 因此服务器等待一段时间，以便让更多的副本到达。
#
# 延迟以秒为单位指定，默认情况下为5秒。要完全禁用它，只需将它设置为0秒，传输将尽快开始。
repl-diskless-sync-delay 5
```

```
# ------------------------------------------------------------------------
# 警告: RDB无磁盘加载是试验性的。在此设置中，副本不会立即在磁盘上存储RDB文件，所以它可能会导致
# 故障转移期间的数据丢失。在与主机的初始同步阶段，RDB无盘加载+Redis模块不能处理I/O读操作可能会
导致Redis方法I/O错误。
# 只在你需要的时候配置此传播方式.
# ------------------------------------------------------------------------
#
# 副本可以直接从套接字加载它从复制链接读取的RDB，也可以将RDB存储到文件中，
# 并在从主服务器完全获取文件后读取该文件。
#
# 在许多情况下，磁盘速度比网络慢，存储和加载RDB文件可能会增加复制时间(甚至增加Master的COW内存
和Salve的缓冲区).
# 但是，直接从套接字解析RDB文件可能意味着我们必须在收到完整的RDB文件之前刷洗当前数据库的内容。
因此，我们有以下选择:
#
# "disabled"    - 不使用无盘复制(优先将RDB文件存储到磁盘).
# "on-empty-db" - 当完全安全的情况下只使用无盘复制.
# "swapdb"      - 直接从套接字解析数据时，将当前数据库内容的副本保留在RAM中。
#                 注意：这需要足够的内存，如果没有足够的内存可以会导致OOM从而被"杀死"。
repl-diskless-load disabled

# 副本以预定义的时间间隔给主服务器发送PING命令。
# 可以修改repl-ping-replica-period更改此间隔时间。默认值为10s.
#
# repl-ping-replica-period 10

# The following option sets the replication timeout for:
#
# 1) Bulk transfer I/O during SYNC, from the point of view of replica.
# 2) Master timeout from the point of view of replicas (data, pings).
# 3) Replica timeout from the point of view of masters (REPLCONF ACK pings).
#
# 确保此值大于repl-ping-replica-period所设置的值。
# 否则，每当主服务器和副本之间的网络较慢时就会检测到超时。
#
# repl-timeout 60

# 在同步后禁用TCP_NODELAY复制套接字?
#
# 如果选择"yes",Redis将使用更少的TCP数据包和更少的带宽将数据发送至副本。
# 但这会增加数据到达副本端的延时，当使用默认配置的Linux内核时，此延时最多可达40毫秒。
#
# 如果选择"no"，将减少数据到达副本端的延迟，但将使用更多带宽进行复制。
#
# 默认情况下，我们会针对低延迟进行优化，但是在流量非常高的情况下，
# 或者当主从服务器之间跳数过多(many hops away)时，将其设置为"yes"可能是个好主意。
repl-disable-tcp-nodelay no

# 设置复制积压缓冲区大小。在副本断开连接的这段时间中，该缓冲区将累积副本数据(其实是命令)，
# 因此，当副本要重新连接时，通常不需要完全重同步，而是部分重同步。只需要传递断开连接时间中的数据
即可。
#
# 复制积压缓冲区越大，副本可以断开连接并执行部分重新同步的时间就越长。
#
# 仅当至少有一个副本连接时，才分配复制积压缓冲区。
# 默认的复制积压缓冲区大小为1MB。
#
```

```
# repl-backlog-size 1mb

# 在主服务器不再连接任何副本一段时间后将释放复制积压缓冲区。
# 以下选项配置了从断开最后一个从服务器的时间到释放缓冲区所需的秒数：
#
# 注意，从服务器永远不会释放缓冲区，因为它们可能会在后续升级为Master，需要能够与副本正确"部分重同步"
# 因此它们应该始终保留缓冲区。
#
# 0意味着永远不释放缓冲区。
#
# repl-backlog-ttl 3600

# 副本优先级是由Redis发布的一个整数，在INFO输出中可以查看到。
# 如果Master不再工作，Redis Sentinel将选择一个副本提升为Master.
#
# 优先级较低的副本被认为更适合升级，因此，例如，如果有三个优先级为10,10025的副本，
# Sentinel将选择优先级为10的副本，即优先级最低的副本。
#
# 但是，如果优先级为0，则表示该副本不能执行master的角色，因此Redis Sentinel将永远不会选择优先级为0的副本进行升级。
#
# 默认的优先级是100.
replica-priority 100

# 如果连接的副本少于N个，并且延迟小于或等于M秒，则主服务器可能会停止接受写入。
#
# N个副本必须在"online"状态.
#
# 延迟（以秒为单位）必须小于等于指定值，该延迟是根据从副本收到的最后一次ping计算得出的，通常每秒钟发送一次。
#
# 此选项并不保证N个副本将接受写操作，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 replicas
# are available, to the specified number of seconds.
#
# 例如至少3个副本，并且最大延迟时间为10秒的设置如下所示：
#
# min-replicas-to-write 3
# min-replicas-max-lag 10
#
# 将一个或另一个设置为0将禁用该功能。
#
# 默认情况下min-replicas-to-write设置为0(禁用),min-replicas-max-lay设置为10

# Redis的Master能够以不同的方式列出从服务器的地址和端口。例如使用"INFO replication"命令获取副本信息。
# Redis Sentinel使用此信息来发现副本实例。该信息可用的另一个地方是Master的"ROLE"命令的输出。
#
# 通常副本报告的IP和地址通过以下方式获取:
#
#   IP: 通过检查副本用来连接主机的套接字的对等地址，自动检测该地址。
#
#   Port: 该端口在握手期间由副本进行通信，通常是副本用来侦听连接的端口。
#
# 但是，当使用端口转发或网络地址转换(NAT)时，实际上可以通过不同的IP和端口访问该副本。
```

```
# 副本可以使用以下两个选项，以便向Master节点报告IP和端口。INFO和ROLE命令将报告这些值。
#
# 如果只需要覆盖端口或IP地址，则无需使用这两个选项。
#
# replica-announce-ip 5.5.5.5
# replica-announce-port 1234

################################ KEYS TRACKING #################################

# Redis实现了服务端对客户端缓存的辅助支持。
# 它是使用失效表来实现的，其使用1600万个插槽来记住客户端可能拥有的某些键子集。
# 它用来给客户端发送无效消息。欲了解更多相关功能，请查看此页：
#
#   https://redis.io/topics/client-side-caching
#
# 为客户端启用跟踪时，所有只读查询均假定为已缓存：这将强制Redis将信息存储在失效表中。
# 当键值信息被修改后将清除此信息。并将无效消息发送给客户端。然而，如果工作负载主要是读取，
# Redis可能会使用越来越多的内存来跟踪许多客户端获取的键。
#
# 因此，可以为失效表配置最大值。默认情况下设置为1M，一旦达到此限制，为了回收内存，即使未修改键值信息Redis
# 也会开始清除失效表中的信息，这将迫使客户端缓存失效。基本上，表的最大大小是在服务端用来跟踪有关谁缓存
# 内容的信息的内存与客户端将缓存的对象保留在内存中的能力之间进行权衡的。
#
# 如果将此值设置为0，则意味着没有限制，Redis将尽可能保留更多的键到失效表中。
# 在INFO命令输出中的"Stats"部分您可以找到关于在每个给定时刻失效表中的键值数量的信息。
#
# 注意:在广播模式下使用key tracking时，服务器端不会使用内存，因此此设置无效。
#
# tracking-table-max-keys 1000000

################################### SECURITY ###################################

# 警告：由于Redis的运行速度非常快，外部用户每秒可以在一个现代机器上尝试输入多大达100万个密码。
# 这意味着您应该使用非常安全的密码，否则密码很容易被破解。
# 请注意，由于该密码实际上是客户端跟服务器之间的共享秘钥，所以不应该被任何人记住。因为密码实际上是客户机和服务器之间的共享秘密，
# 不应该被任何人记住，所以密码可以是来自/dev/urandom的长字符串，所以使用一个长而不可猜的密码就不可能进行暴力攻击。

# Redis ACL 用户以以下格式定义:
#
#   user <username> ... acl rules ...
#
# 例如:
#
#   user worker +@list +@connection ~jobs:* on >ffa9203c493aa99
#
# 特殊的用户名"default"用于新的连接。如果此用户具有"nopass"规则，则新连接将立即被认证为"default"用户，
# 而无需通过AUTH命令提供的任何密码。否则，如果未将"default"用户标记为"nopass"，则连接将以未认证状态
# 启动，并且需要AUTH(或HELLO命令AUTH选项)才能进行认证并开始工作。
#
# 描述用户权限的ACL规则如下:
#
#  on             启用用户：可以验证为该用户。
```

```
#  off           禁用该用户：无法再与此用户进行身份验证，但是已经身份验证的连接仍然可以使用。
#
#  +<command>    允许执行该命令
#  -<command>    禁止执行该命令
#  +@<category> 允许执行此类中具有有效类别的所有命令，例如@admin,@set,@sortedset等，
#                可以查看server.c文件，其中描述和定义了Redis的命令表。特殊类别@all表示
#                当前存在于服务器中，将来将通过模块加载的所有命令。
#  +<command>|subcommand     允许其他命令的特定子命令。注意，这种形式不允许使用"-"，如-
DEBUG|SEGFAULT,
#                             只允许使用"+"开头。
#
#  allcommands  +@all的别名。注意，这意味着可以执行通过模块系统加载的所有命令。
#  nocommands   -@all的别名。
#  ~<pattern>    使用类似正则表达式的pattern模式。例如~*表示允许所有命令。可以指定多个
pattern。
#  allkeys       ~*的别名。
#  resetkeys     刷新允许的模式列表。
#  ><password>  将此密码添加到用户的有效密码列表中。
#                例如，>mypass会将"mypass"添加到列表中。 该指令清除"no pass"标志。
#  <<password>  从有效密码列表中删除此密码。
#  nopass        用户的所有密码都将被删除，并且标记该用户不需要密码：这意味着每个密码都会对这
个用户起作用。
#                如果此指令用于默认用户，则每个新连接都将立即通过默认用户进行身份验证，而无需
任何显式AUTH命令。
#                注意，"resetpass"指令将清除此条件。
#  resetpass     刷新允许的密码列表。此外，删除"nopass"状态。
#                在"resetpass"之后，用户将没有任何关联的密码，并且在添加密码前无任何进行身份
验证的方法(或稍后将其设置为"nopass")。
#  reset         执行以下操作：resetpass,resetkeys,off,-@all。用户回到创建后的初始状态。
#
# 可以按任何顺序指定ACL规则：例如，您可以先输入密码，再输入标志或密钥模式。
# 但是请注意启用和弃用规则将根据顺序改变含义。
# 例如，请参见下面的示例:
#
#   user alice on +@all -DEBUG ~* >somepassword
#
# 上述配置将允许用户"alice"使用除DEBUG命令之外的所有命令。因为+@all将所有命令添加到用户可使用
的命令集中，
# -DEBUG表示将此命令从命令集中删除。但是如果我们颠倒了这两个ACL规则的顺序含义将发生改变：
#
#   user alice on -DEBUG +@all ~* >somepassword
#
# 现在，当alice在允许的命令集中还没有命令时，DEBUG被删除，之后又添加了所有命令，因此用户将能够
执行所有操作。
#
# 基本上，ACL规则是从左到右处理的。
#
# 有关ACL配置的更多信息，请参考Redis网站，网址为https://redis.io/topics/acl。

# ACL Log
#
# ACL Log 追踪与ACL关联的失败命令和身份验证事件。
# ACL Log 可用于对 ACL 阻止的失败命令进行故障排除。
# ACL Log 存储在内存中。它的长度没有限制。你可以使用
# 也可以在下面设置最大长度。
acllog-max-len 128

# 使用外部的ACL文件
```

```
#
# 除了在此文件中配置用户信息，可以使用一个只列出用户的文件。
# 这两种方法不能混合使用，如果你在这里配置文件，同时激活外部 ACL 文件，服务器将拒绝启动。
#
# 外部ACL用户文件的格式与在redis.conf中用于描述用户的格式完全相同。
#
# aclfile /etc/redis/users.acl

# 重要提示：从Redis 6 开始，"requirepass"只是ACL 系统上的一个兼容层。
# 此选项将只为默认用户设置密码。客户端仍然像往常一样使用AUTH进行身份验证，
# 或者更显式地使用AUTH默认值进行身份验证(如果它们遵循新的协议的话):两者都可以工作。
#
# requirepass foobared

# 命令重命名(弃用)。
#
# ------------------------------------------------------------------------
# 警告：如果可能，避免使用此选项。
# 而是使用ACL从默认用户中删除命令，并将其仅放置在您出于管理目的而创建的某些admin用户中。
# ------------------------------------------------------------------------
#
# 可以在共享环境中更改危险命令的名称。例如，可以将CONFIG命令重命名为一些难以猜测的名称，
# 以便它仍可用于内部使用的工具，但不适用于一般用户。
#
# 示例:
#
# rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52
#
# 通过将命令重命名为空字符串可以完全取消命令：
#
# rename-command CONFIG ""
#
# 请注意，更改登录到AOF文件或传输到副本的命令的名称可能会导致问题。

################################### CLIENTS ####################################

# 设置最大连接客户端数量。默认情况下，设置为10000 个客户端，
# 但是，如果Redis 服务器无法将进程文件配置为指定限制值，则运行的最大客户端数为当前限制值减32
# (因为Redis保留的内部使用文件描述符很少)
#
# 达到限制后，Redis将关闭所有新连接，并发送错误消息"已达到最大客户端数"。
#
# maxclients 10000

############################## MEMORY MANAGEMENT
################################

# 将Redis内存限制设置为指定的字节数。当达到此限制值是，Redis将尝试根据所配置的逐出策略来删除键
值对。
# (参见maxmemory-policy)
#
# 如果Redis无法根据该策略删除键值对，或者如果该策略设置为'noeviction'，
# Redis则将对写操作(如SET、LPUSH等命令)回复错误，但仍响应读操作(如GET等命令)。
#
# 当将Redis作为LRU或LFU缓存，或为实例设置内存限制(使用'noeviction'策略)时，此选项很有用。
#
# 警告：如果一个设置了"maxmemory"的主节点连接了一个从节点，那么用于主从复制传递命令的输出缓冲
区占用的内存也在maxmemory当中，
```

```
# 如果当内存被占满时而出现大量的删除key的操作写到缓冲区，而缓冲区又不够，又会触发删除更多的key，这样就会造成一个死循环，
# 直到整个数据库变成空的。
#
# 简而言之...如果您附加了副本，建议您为maxmemory设置一个下限，
# 以便系统上有一些用于副本输出缓冲区的可用RAM（但是如果策略为"noeviction"，则不需要这样做）。
# 因为used memory是可以大于maxmemory的，只不过出现这种情时会导致内存回收而触发删除KEY的操作。
# 因此，如果在主从模式下，主节点的maxmemory在设置得足够大的情况下，还要给输出缓冲区留出一点空间来，
# 避免出现死循环而导致数据库被清空。不要物理内存有多少就设置多少，况且还有操作系统和其他程序在运行,一般设置为3/4。
#
# maxmemory <bytes>

# MAXMEMORY POLICY：达到maxmemory后，Redis将如何选择要删除的内容。 您可以从以下行为中选择一种：
#
# volatile-lru -> 根据LRU算法删除设置了超时属性（expire）的键，直 到腾出足够空间为止。如果没有可删除的键对象，回退到noeviction策略。
# allkeys-lru -> 根据LRU算法删除键，不管数据有没有设置超时属性， 直到腾出足够空间为止。
# volatile-lfu -> 使用近似的LFU删除过期键。
# allkeys-lfu -> 使用近似的LFU删除所有键。
# volatile-random -> 随机删除过期键，直到腾出足够空间为止。
# allkeys-random -> 随机删除所有键，直到腾出足够空间为止。
# volatile-ttl -> 根据键值对象的ttl属性，删除最近将要过期数据。如果没有，回退到noeviction策略。
# noeviction -> 默认策略，不会删除任何数据，拒绝所有写入操作并返回客户端错误信息(error)OOM command not allowed when used memory，
#                此时Redis只响应读操作。
# 推荐一个：https://cloud.tencent.com/developer/article/1530553 讲了原理和使用说明
#
# LRU means Least Recently Used   最近没有被使用的
# LFU means Least Frequently Used 最近使用频率最小的
#
# LRU、LFU和volatile-ttl都是用近似随机算法实现的。
#
# 注意: 使用上面的任何一种策略，在没有适合的键值对可以回收时，Redis对所有写操作返回错误。
#
#       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
#
# 默认策略:
#
# maxmemory-policy noeviction

# LRU、LFU和最小TTL算法不是精确算法，而是近似算法(为了节省内存)，
# 所以您可以调整它的速度或准确性。默认情况下Redis会检查5个键，并选择最近使用较少的一个，
# 你可以使用以下配置指令改变样本大小。
#
# 默认值为5会产生足够好的结果。10非常接近真实LRU，但花费更多CPU。3更快，但不是很准确。
#
# maxmemory-samples 5
```

```
# 从Redis 5开始，默认情况下，从服务器将忽略其maxmemory设置（除非在故障转移后或手动提升为Master）。
# 这意味着从服务器将不回收键值对，而是通过主服务器发送来的DEL命令进行删除。
#
# 此行为可确保主副本和副本始终保持一致，这通常是您想要的，但是，如果副本是可写的，
# 或者您希望副本具有不同的内存设置，并且您确定对副本执行的所有写操作都是幂等的，
# 那么您可以更改此默认设置（但请务必了解您的操作）。
#
# 注意，由于副本在默认情况下不会被清除，因此它最终使用的内存可能比通过maxmemory设置的内存多
# (副本上的某些缓冲区可能更大，或者数据结构有时可能占用更多内存，等等)。
# 因此，请确保监视副本，并确保它们有足够的内存，在主服务器达到配置的maxmemory设置之前不会出现真正的内存不足情况。
#
# replica-ignore-maxmemory yes

# Redis通过两种方式回收过期的键值对：访问时发现这些键值对已经过期，称为"活动的过期键值对"。
# 对键值对进行缓慢的交互式扫描，查找过期的键值对以便回收，这样就有可能释放已过期且在短时间内不会再次访问的对象的内存。
#
# 默认回收周期将试图避免内存中仍然有超过10%的过期密钥，并将试图避免消耗超过25%的总内存而给系统增加延迟。
# 然而可以通过修改默认"effort"的值改变回收效果，默认为1，最大值为10。
# 使用最大值系统将使用更多的CPU，更长的周期(并且在技术上可能引入更多的等待时间)，
# 但将减少仍然存在于系统中的已经过期的密钥。这是一个在内存，CPU和延迟之间的权衡。
#
# active-expire-effort 1

############################## LAZY FREEING ####################################

# Redis有两个删除键的原语。 一种称为DEL，它是对象的阻塞删除。 这意味着服务器停止处理新命令，
# 以便以同步方式回收与对象关联的所有内存。 如果删除的键与一个小对象相关联，
# 则执行DEL命令所需的时间非常短，可与Redis中的大多数其他O(1)或O(logN)命令相提并论。
# 但是，如果键与包含数百万个元素的聚合值关联，则服务器可能会阻塞很长时间(甚至几秒钟)以完成操作。
#
# 由于上述原因，Redis还提供了非阻塞删除原语，例如UNLINK（非阻塞DEL）以及FLUSHALL和FLUSHDB命令的ASYNC选项，
# 以便在后台回收内存。这些命令在固定时间内执行。 另一个线程将尽可能快地在后台逐渐释放对象。
#
# 用户可以控制FLUSHALL和FLUSHDB的DEL，UNLINK和ASYNC选项。
# 由应用程序来决定何时使用哪个选项是一个好主意。
# 但是，Redis服务器有时由于其他操作的副作用而必须删除键或刷新整个数据库。
# 特别是在以下情况下，Redis会独立于用户调用而删除对象:
#
# 1) 在内存回收时，由于maxmemory和maxmemory策略配置，以便为新数据腾出空间，而不会超过指定的内存限制。
#
# 2) 过期原因:必须从内存中删除一个与生存时间相关的键(请参阅EXPIRE命令)。
#
# 3) 由于将数据存储在可能已经存在的键上的副作用。例如使用RENAME重命名一个旧的键值对时可能会删除掉它。
#     同样，使用SUNIONSTORE或SORT with STORE命令也可能会删除现有键值对。
#     SET命令本身会删除指定键的所有旧内容，以便将其替换为指定的字符串。
#
# 4) 在复制期间，当从服务器与其主服务器执行完全重同步时，为了加载刚刚传输的RDB文件，
#     整个数据库的内容将被删除。
#
# 在上述所有情况下，默认删除对象的方式是阻塞的，就像如果DEL被调用。
# 但是，您可以使用以下配置指令专门配置每种情况，以便以非阻塞的方式释放内存，如调用UNLINK时。
```

```
lazyfree-lazy-eviction no
lazyfree-lazy-expire no
lazyfree-lazy-server-del no
replica-lazy-flush no

# 对于用UNLINK代替DEL困难的情况，
# 也可以使用以下配置指令将DEL命令的默认行为修改为与UNLINK完全一样：

lazyfree-lazy-user-del no

################################ THREADED I/O #################################

# Redis主要是单线程的，但是也有一些特定的线程操作，
# 如断开链接，缓慢的I/O访问和其他事情是在其他线程上执行的。
#
# 现在，还可以在不同的I/O线程中处理Redis客户端套接字的读写。
# 由于特别慢的写入速度，因此Redis用户通常使用流水线以加快每个内核的Redis性能，并生成多个实例以扩大规模。
# 使用I/O线程可以轻松地将Redis加速两次，而无需求助于实例的流水线处理或分片。
#
# 默认情况下，线程是禁用的，我们建议仅在具有至少4个或更多内核的计算机上启用它，至少保留一个备用内核。
# 使用8个以上的线程不太可能有很大帮助。我们还建议仅在实际存在性能问题时才使用线程I/O，
# Redis实例可以使用很大一部分CPU时间，否则就没有必要使用此功能。
#
# 因此，例如，如果您有四个核，请尝试使用2或3个I/O线程，
# 如果您有8个核，请尝试使用6个线程。为了启用I/O线程，请使用以下配置指令：
#
# io-threads 4
#
# 通常，将io-threads设置为1只会使用主线程。启用I / O线程后，我们仅使用线程进行写操作，
# 并将客户端缓冲区传输到套接字。但是，也可以使用以下配置指令，通过将其设置为yes，来启用读取线程和协议解析：
#
# io-threads-do-reads no
#
# 通常，线程读取没有太大帮助。
#
# 注意1：无法在运行时通过CONFIG SET更改此配置指令。
#  启用SSL时，Aso功能也不起作用。
#
# 注意2：如果您想使用redis-benchmark测试Redis速度，请确保您还以--threads选项匹配Redis thead的数量，
# 以线程模式运行基准测试本身，否则您将无法发现改进。

############################## APPEND ONLY MODE ###############################

# 默认情况下，Redis异步将数据集转储到磁盘上。
#  此模式在许多应用程序中已经足够好，但是Redis进程问题或断电可能会导致几分钟的写入丢失(取决于配置的保存点)。
#
# AOF文件是一种替代的持久性模式，可提供更好的持久性。
# 例如，使用默认数据fsync策略(请参阅配置文件中的稍后内容)，
# Redis在严重的事件(例如服务器断电)中仅会丢失一秒钟的写入，如果Redis进程本身发生问题，则可能会丢失一次写入，
# 但是系统仍在正常运行。
#
```

```
# 可以同时启用AOF和RDB持久性，而不会出现问题。
# 如果在启动时启用了AOF，则Redis将加载AOF，该文件具有更好的持久性保证。
#
# 请查看http://redis.io/topics/persistence了解更多信息。

appendonly no

# AOF文件名字 (默认名字: "appendonly.aof")

appendfilename "appendonly.aof"

# fsync()调用告诉操作系统将数据写入磁盘，而不是等待输出缓冲区中的更多数据。
# 某些操作系统确实会刷新磁盘上的数据，而另一些操作系统只会尝试尽快执行此操作。
#
# Redis支持三种不同的模式:
#
# no: 不记录进AOF文件，只让OS在需要时刷新数据即可。更快。
# always: 每次写入都记录到AOF文件中。慢，最安全。
# everysec: 每秒写入AOF文件。一般。
#
# 默认值为"everysec"，这通常是速度和数据安全性之间的正确折衷。
# 您可以了解是否可以将其放宽为"no"，以便操作系统在需要时刷新输出缓冲区，
# 以获得更好的性能(但是如果您可以忍受某些数据丢失的想法，请考虑使用默认的持久模式)
# 或者相反，使用"always"，该速度非常慢，但是比everysec更安全。
#
# 更多详细信息，请查看以下文章:
# http://antirez.com/post/redis-persistence-demystified.html
#
# 如果不确定,使用"everysec".

# appendfsync always
appendfsync everysec
# appendfsync no

# 当AOF文件同步策略设置为always或everysec，
# 并且后台保存进程（后台保存或AOF日志后台重写）对磁盘执行大量I/O时，在某些Linux配置中，Redis可能会在磁盘上阻塞太长时间。
# 请注意，目前尚无此修复程序，因为即使在其他线程中执行fsync也将阻塞我们的同步写操作。
#
# 为了减轻此问题，可以使用以下选项来防止在BGSAVE或BGREWRITEAOF进行时在主进程中调用fsync()。
#
# 这意味着当另一个子线程在保存时，Redis的持久性策略与"appendfsync none"相同。
# 这意味着在最坏的情况下(使用默认的Linux设置)可能会丢失多达30秒的日志。
#
# 如果您有延迟问题，请将其设置为"yes"。否则就保留"no"，从持久的角度来看，这是最安全的选择。

no-appendfsync-on-rewrite no

# 自动重写AOF文件。
# 当AOF文件增长到指定的大小时，Redis隐式调用BGREWRITEAOF自动重写日志文件。
#
# 重写工作流程：Redis会在最近一次重写后记住AOF文件的大小(如果自重新启动以来未发生任何重写，则使用启动时AOF的大小)。
#
# 将配置的大小与当前大小进行比较。如果当前大小大于指定的百分比，则会触发重写。
# 另外，您需要指定要重写的AOF文件的最小大小，即使达到百分比，但它仍然很小，
# 这对避免重写AOF文件很有用。
#
```

```
# 指定零百分比以禁用自动AOF重写功能。

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb

# 在Redis启动过程中将AOF数据重新加载回内存时，可能会发现AOF文件在末尾被截断。
# 当运行Redis的系统崩溃时，尤其是在没有data=ordered选项的情况下挂载ext4文件系统时，
# 可能会发生这种情况(但是，当Redis本身崩溃或中止，但操作系统仍然可以正常运行时，就不会发生这种情况)。
#
# 如果发现AOF文件最后被截断，Redis可能会退出并显示错误，
# 也可以加载尽可能多的数据(默认),以下选项控制此行为。
#
# 如果aof-load-truncated设置为yes，则将加载截短的AOF文件，Redis服务器将发送日志给用户。
# 否则，Redis将显示错误并拒绝启动。如果该选项设置为no，则用户需要在重新启动服务器之前使用"redis-check-aof"实用程序修复AOF文件。
#
# 请注意，如果在中间发现AOF文件已损坏，Redis仍将退出并出现错误。
# 仅当Redis尝试从AOF文件读取更多数据但找不到足够的字节时，此选项才适用。
aof-load-truncated yes

# 当重写AOF文件时，Redis可以在AOF文件中使用RDB前缀来更快地重写和恢复。
# 启用此选项后，重写的AOF文件由两个不同的节组成:
#
#   [RDB file][AOF tail]
#
# 加载时，Redis会识别AOF文件以"REDIS"字符串开头并加载带前缀的RDB文件，然后继续加载AOF尾部。
aof-use-rdb-preamble yes

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

# 配置Lua脚本的最大执行时间（以毫秒为单位）。
#
# 如果达到了最大执行时间，Redis会记录一个脚本在超过最大允许时间后仍在执行，并开始回复一个错误的查询。
#
# 当长时间运行的脚本超过最大执行时间时，只有KILL和SHUTDOWN NOSAVE命令可用。
# KILL命令可以用来停止尚未调用write命令的脚本。
# SHUTDOWN NOSAVE命令是在脚本已经发出了写命令但用户不想等待脚本自然终止的情况下关闭服务器的唯一方法。
#
#
# 将其设置为0或负值，可以无限制地执行而不发出警告。
lua-time-limit 5000

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

# 普通的Redis实例不能成为集群的一部分;只有作为集群节点启动的节点可以。
# 为了将Redis实例启动为集群节点，需启用集群支持，请取消注释以下内容:
#
# cluster-enabled yes

# 每个集群节点都有一个集群配置文件。此文件不应由用户编辑。它由Redis节点创建和更新。
# 每个Redis集群节点都需要一个不同的集群配置文件。
# 确保在同一系统上运行的实例没有相同的配置文件名。
#
# cluster-config-file nodes-6379.conf
```

```
# 集群节点超时时间是一个节点不可达的毫秒数，才能将其视为故障状态。
# 大多数内部时间限制是节点超时的倍数。
# cluster-node-timeout 15000

# 如果发生故障的Master副本的数据看起来太旧，它将避免启动故障转移。
#
# 对于副本来说，没有一种简单的方法可以确切地测量它的"data age"，因此执行以下两项检查:
#
# 1) 如果存在多个能够进行故障转移的副本，则它们会交换消息，
#     以便尝试利用具有最佳复制偏移量(处理了更多来自主数据库的数据)的副本来获得优势。
#     副本将尝试按偏移量分等级，并在故障转移开始时施加与它们的等级成比例的延迟。
#
# 2) 每个单个副本都会计算与其主副本之间最后一次交互的时间。
#     这可以是最后收到的ping或命令(如果主服务器仍处于"已连接"状态)，its master.
#     也可以是自从与主服务器断开连接以来经过的时间(如果副本连接当前已关闭)。
#     如果最后一次交互时间太老，则副本将完全不会尝试故障转移。
#
# 用户可以设置为第二项。
# 特别是，具体来说，如果在与主服务器的最后一次交互之后所经过的时间大于配置的时间，副本将不执行故障转移：
#
#   (node-timeout * replica-validity-factor) + repl-ping-replica-period
#
# 例如，如果节点超时为30秒，副本有效性因子为10，并且假设默认的复制副本周期为10秒，
# 那么如果与主服务器的对话时间超过310秒，副本将不会尝试进行故障转移。
#
# 较大的副本有效性因子可能会使数据太旧的副本无法对主副本进行故障转移，
# 而值太小可能会使群集根本无法选择副本。
#
# 为了获得最大可用性，可以将副本有效性因子设置为0，
# 这意味着，无论副本上次与主服务器进行交互的时间，副本将始终尝试对主服务器进行故障转移。
# (但是，他们将始终尝试应用与其偏移等级成正比的延迟)。
#
# 零是唯一能够保证当所有分区恢复正常后集群能够继续运行的值。
#
# cluster-replica-validity-factor 10

# 集群副本能够迁移到独立的主数据库，即那些没有工作副本的主数据库。
# 这样可以避免集群的单点故障能力，否则如果独立的主数据库没有可用的副本，
# 那么该主数据库无法在发生故障的情况下进行故障转移。
#
# 只有在至少有给定数量的其他工作副本连接主服务器的情况下，副本才会迁移到独立的主服务器。
# 这个数字就是"migration barrier"。migration barrier为1意味着只有在Master至少有1个工作副本时，副本才会迁移，依此类推。
# 它通常反映集群中每个主服务器的副本数量。
#
# 缺省值为1(仅当Master保留至少一个副本时，副本才会迁移)。
# 要禁用迁移，只需将其设置为非常大的值即可。可以将值设置为0，但仅用于调试和生产危险。
#
# cluster-migration-barrier 1

# 默认情况下，如果Redis Cluster节点检测到至少发现一个哈希槽未被覆盖(没有可用的节点为它服务)，它们将停止接受查询。
# 这样，如果集群部分下线(例如，部分哈希槽未被覆盖)，则所有集群最终将变得不可用。
# 再次覆盖所有哈希槽后，它将自动返回可用状态。
#
# 但是，有时您希望正在运行的集群子集继续接受对仍覆盖的部分键空间的查询。
# 为此，只需将cluster-require-full-coverage选项设置为no。
```

```
#
# cluster-require-full-coverage yes

# 设置为yes时，此选项可防止副本在主服务器发生故障时尝试对其主服务器进行故障转移。
# 但是，如果强制这样做，主服务器仍然可以执行手动故障转移。
#
# 这在不同的场景中都很有用，特别是在多个数据中心操作的情况下，
# 如果不是在完全DC故障的情况下，我们希望任何一侧永远不会被提升。
#
# cluster-replica-no-failover no

# 如果将此选项设置为yes，则允许节点在集群处于关闭状态时为读取流量提供服务，
# 只要它认为自己拥有插槽即可。
#
# 这对于两种情况很有用。
# 第一种情况是在节点故障或网络分区期间应用程序不需要数据一致性时。
# 缓存就是一个例子，只要节点拥有数据，它就应该能够为其提供服务。
#
# 第二个用例用于不满足推荐的三个分片，但希望以后启用集群模式和扩展的配置。
# 如果没有设置此选项，1或2分片配置中的主停机将导致整个集群的读/写停机，
# 而设置了此选项，则只会出现写停机。如果主机达到法定数量，插槽所有权不会自动更改。
#
# cluster-allow-reads-when-down no

# 要安装集群，请务必阅读http://redis.io上提供的文档。

############################ CLUSTER DOCKER/NAT support  ########################

# 在某些部署中，因为地址经过NAT限制或端口被转发(典型情况是Docker和其他容器)，
# Redis Cluster节点的地址发现失败。
#
# 为了使Redis Cluster在这样的环境中工作，需要一个静态配置，使得每个节点都知道其公共地址。
# 以下两个选项用于此范围，分别是：
#
# * cluster-announce-ip
# * cluster-announce-port
# * cluster-announce-bus-port
#
# 每个节点都告知节点自己的地址、客户端端口和集群消息总线端口。
# 然后，信息被发布在总线包的报头中，以便其他节点能够正确地映射发布信息的节点的地址。
#
# 如果未使用上述选项，则将使用常规的Redis集群自动检测方法。
#
# 请注意，重新映射时，总线端口可能不在客户端端口+ 10000的固定偏移处，
# 因此您可以根据重新映射的方式指定任何端口和总线端口。
# 如果未设置总线端口，通常将使用10000的固定偏移量。
#
# 示例:
#
# cluster-announce-ip 10.1.1.5
# cluster-announce-port 6379
# cluster-announce-bus-port 6380

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

# Redis Slow Log是一个用于记录超过指定执行时间的查询的系统。
# 执行时间不包括与客户端交谈，发送回复等I/O操作，而仅包括实际执行命令所需的时间
# (这是执行命令的唯一阶段，在该阶段线程被阻塞并且可以在此期间不执行其他任务)。
```

```
#
# 您可以使用两个参数配置慢日志:一个参数告诉Redis要记录慢命令需要执行超时时间(以微秒为单位),
# 另一个参数是慢日志的长度。当记录一个新命令时,旧命令将从记录的命令队列中删除。

# 时间以微秒为单位,因此1000000等于一秒。
# 请注意,负数将禁用慢速日志记录,而零值将强制记录每个命令。
slowlog-log-slower-than 10000

# 该长度没有限制。
# 请注意,它将消耗内存。您可以使用SLOWLOG RESET回收慢日志使用的内存。
slowlog-max-len 128

################################ LATENCY MONITOR ##############################
################################## 延迟监控 ###################################
#
# Redis延迟监视子系统会在运行时对不同的操作进行采样,以收集与Redis实例的潜在延迟源相关的数据。
#
# 通过LATENCY命令,可以打印图形和报告,用户可以使用此信息。
#
# 系统仅记录大于或等于delay-monitor-threshold配置指令指定的毫秒数执行的操作。
# 当其值设置为0时,等延迟监视器将关闭。
#
# 默认情况下,延迟监视处于禁用状态,因为如果您没有延迟问题,则几乎不需要它,
# 并且收集数据会对性能产生影响,尽管影响很小。但是可以在高负载下进行监视。
# 如果需要,可以在运行时使用命令"CONFIG SET delay-monitor-threshold <milliseconds>"轻
松启用延迟监视。
latency-monitor-threshold 0

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

# Redis可以将key发生的事件通知给Pub/Sub客户端。
# 此特性记录在http://redis.io/topics/notifications
#
# 例如,如果启用了键的事件通知,并且客户端对存储在数据库0中的键"foo"执行了DEL操作,
# 则将通过Pub/Sub发布两条消息:
#
# PUBLISH __keyspace@0__:foo del
# PUBLISH __keyevent@0__:del foo
#
# 可以在一组类中选择Redis将要通知的事件。每类都由一个字符标识:
#
#  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)
#  t     Stream commands
#  m     Key-miss events (Note: It is not included in the 'A' class)
#  A     Alias for g$lshzxet, so that the "AKE" string means all the events
#        (Except key-miss events which are excluded from 'A' due to their
#         unique nature).
#
#  "notify-keyspace-events"将由零个或多个字符组成的字符串作为参数。
```

```
#  空字符串表示已禁用通知。
#
#  示例1：从事件名称的角度来看，要启用列表事件和通用事件，请使用：
#
#  notify-keyspace-events Elg
#
#  示例2：获取订阅频道的过期key：name __keyevent@0__:expired 使用：
#
#  notify-keyspace-events Ex
#
#  默认情况下，所有通知都被禁用，因为大多数用户不需要此功能，并且该功能有一些开销。
#  请注意，如果您未指定K或E中的至少一个，则不会传递任何事件。
notify-keyspace-events ""

############################### GOPHER SERVER ################################

# Redis包含RFC 1436（https://www.ietf.org/rfc/rfc1436.txt）中指定的Gopher协议的实现。
#
# Gopher协议在90年代后期非常流行。
# 它是Web的替代方法，服务器和客户端的实现是如此简单，以至于Redis服务器只用100行代码就实现了。
#
# What do you do with Gopher nowadays? Well Gopher never *really* died, and
# lately there is a movement in order for the Gopher more hierarchical content
# composed of just plain text documents to be resurrected. Some want a simpler
# internet, others believe that the mainstream internet became too much
# controlled, and it's cool to create an alternative space for people that
# want a bit of fresh air.
#
# 无论如何，在Redis诞生10周年之际，我们给了它Gopher协议作为礼物。
#
# --- HOW IT WORKS? ---
#
# Redis Gopher支持使用Redis的内联协议，特别是两种非法的内联请求：
# 空请求或任何以"/"开头的请求(没有以这样的斜杠开头的Redis命令)。
# 正常的RESP2“RESP3请求完全超出了Gopher协议实现的范围，但通常也得到满足。
#
# 如果在启用Gopher后打开与Redis的连接，并向其发送"/foo"之类的字符串，
# 如果存在名为"/foo"的密钥，则将通过Gopher协议为其提供服务。
#
# 为了创建一个真正的Gopher"漏洞"(Gopher对话中Gopher站点的名称)，您可能需要类似以下的脚本：
#
#   https://github.com/antirez/gopher2redis
#
# --- 安全警告 ---
#
# 如果您打算把Redis放置在公网上，服务器的Gopher页面一定要设置密码的实例。
# 设置密码后：
#
#   1. Gopher服务器（启用后，默认情况下未启用）仍将通过Gopher提供内容。
#   2. 但是，在客户端进行身份验证之前无法调用其他命令。
#
# 因此，请使用“ requirepass”选项来保护您的实例。
#
# 要启用Gopher支持，请取消注释以下行，并将选项从no（默认）设置为yes。
#
# gopher-enabled no

############################### ADVANCED CONFIG ###############################
```

```
# 当哈希条目只有少量条目且最大条目未超过给定阈值时，将使用内存高效的数据结构对其进行编码。
hash-max-ziplist-entries 512
hash-max-ziplist-value 64

# 列表也以特殊方式编码，以节省大量空间。每个内部列表节点允许的条目数可以指定为固定的最大大小或最大元素数。
# 对于固定的最大大小，使用-5到-1，这意味着:
# -5: 最大大小: 64 Kb  <-- 正常工作负载不推荐使用
# -4: 最大大小: 32 Kb  <-- 不推荐
# -3: 最大大小: 16 Kb  <-- 可能不推荐
# -2: 最大大小: 8 Kb   <-- 优秀
# -1: 最大大小: 4 Kb   <-- 优秀
#
# 正数表示每个列表节点最多可以存储该数量个元素。
# 最高性能的选项通常是-2(8 Kb大小)或-1(4 Kb大小)，但是如果您的用例是独一无二的(特例)，请根据需要调整设置。
list-max-ziplist-size -2

# 列表也可以被压缩。
# 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: 禁用所有列表压缩
# 1: 深度1表示"头尾1个节点不压缩压缩，但之间节点进行压缩"
#    So: [head]->node->node->...->node->[tail]
#    表示除链表的头尾以外，其他链表节点都压缩
# 2: [head]->[next]->node->node->...->node->[prev]->[tail]
#    表示不要压缩head或head->next或tail->prev或tail，而是压缩它们之间的所有节点。
# 3: [head]->[next]->[next]->node->node->...->node->[prev]->[prev]->[tail]，含义以此类推
# 等等.
list-compress-depth 0
# 默认压缩深度为0，也就是说不压缩。无论如何设置，头尾是不会压缩的，比如当list被当做队列使用时，如果压缩了，还需要解压，降低了性能。


# 集合只有在以下这种情况下有一种特殊的编码:
# 当集合由恰好是基数为10的整数(64位带符号整数)组成时。
# 下面的配置设置设置了集合大小的限制，以便使用这种特殊的内存节省编码。
set-max-intset-entries 512
# 当集合(set)存放的值都是64位的无符号10进制整数时，且条目数小于512时会采用intset作为集合的底层数据结构

# 与哈希表和列表类似，排序集也进行了特殊编码，以节省大量空间。
# 仅当排序集的长度和元素低于以下限制时，才使用此编码:
zset-max-ziplist-entries 128
zset-max-ziplist-value 64
# 如果条目数小于128，且条目大小<64会使用ziplist作为有序集合的底层数据结构
#
# 这个是Redis高级功能，可以用这种数据结构统计网站的UV，能够去重，其准确度接近真实值
# HyperLogLog稀疏表示有字节的限制。这个限制包括16个字节头。
# 当使用稀疏表示的HyperLogLog超过这个限制时，它就被转换成稠密表示。
#
# 大于16000的值是完全无用的，因为在这种情况下，稠密表示的内存效率更高。
#
# 为了在过多PFADD(PFADD的稀疏编码为O(N))的情况下获得节省空间编码的好处，建议值约为3000。
```

```
# 如果不需要考虑CPU，但是需要考虑空间，并且数据集由基数在0-15000范围内的许多Hyperloglog组成，
# 那么这个值可以提高到10000。
#
# 简单点说：当去重后统计出来的值小于"hll-sparse-max-bytes"指定的值时，Redis会使用稀疏矩阵来存放，一个Key占用的空间比稠密矩阵小，
# 如果统计出来的值大于"hll-sparse-max-bytes"指定的值，那么使用稠密矩阵，此时一个Key占用的空间是12KB
# "hll-sparse-max-bytes"默认为3000，如果设置为16000以上完全是无用的，因为此时稠密矩阵效果更好
hll-sparse-max-bytes 3000

# 流节点每一项的最大大小。流数据结构是一个大节点的基树，其中编码多个项目。
# 使用此配置，可以配置单个节点的字节大小，以及在添加新流项时切换到新节点之前它可能包含的最大项数。如果以下任何设置设置为零，限制将被忽略，
# 因此，例如，可以通过设置max-bytes为0,max-entries为所需值来设置最大entires限制。
stream-node-max-bytes 4096
stream-node-max-entries 100
# 设置Stream的单个节点最大字节数和最多能有多少个条目，如果任何一个条件满足就会新增加一个节点用以保存新的数据
# 如果将任何一个配置项设置为0，表示不限制


# 每100毫秒CPU时间rehash使用1毫秒，以帮助重新哈希主Redis哈希表。
# Redis使用的哈希表实现(请参阅dict.c)渐进的rehash过程：
# 您在要进行哈希处理的哈希表中运行的操作越多，执行的哈希处理"步骤"就越多，因此，如果服务器空闲，则哈希处理将永远不会完成，
# 导致哈哈希表使用更多的内存。
#
# 默认值是每秒使用10毫秒来主动rehash，并在可能的情况下释放内存。
#
# 如果您不确定：
# 如果您有严格的延迟要求，请使用"activehashing no".
# 但是Redis可以一直以2毫秒的延迟答复查询不是一件好事。
#
# 如果您没有如此严格的要求，但希望在可能的情况下尽快释放内存，请使用"activerehashing yes"。
activerehashing yes

# 客户端输出缓冲区限制可用于强制断开由于某些原因而不能足够快地从服务器读取数据的客户端
# (一个常见的原因是Pub/Sub客户端读取消息的速度赶不上发布端生成消息的速度)
#
# 可以通过设置客户端输出缓冲区大小将待接收数据超过缓冲区大小的客户端断开
# 通常使用pub/sub的时候，客户端没有及时消费而导致超过缓冲区大小

# 可以为三种不同类别的客户设置不同的限制：
#
# normal -> 普通客户，包括MONITOR客户
# replica  -> 主从复制的
# pubsub -> pub/sub的客户端
#
# 每个client-output-buffer-limit指令的语法如下：
#
# client-output-buffer-limit <class> <hard limit> <soft limit> <soft seconds>
#
# 如果客户端输出缓冲区的大小达到了"hard limit"，服务器会立即断开连接
# 如果客户端输出缓冲区的大小达到了"soft limit"，且持续时间达到了"soft seconds"，服务器会立即断开连接
```

```
# 因此，例如，如果硬限制为32MB，软限制为16MB/10s，
# 则如果输出缓冲区的大小达到32MB，客户端将立即断开连接，但是如果客户端达到16MB并连续超过此限制10秒钟，也会断开连接。
#
# 默认情况下，普通客户端不受限制，因为它们不会在没有询问的情况下(以推送方式)接收数据，而是在请求之后才接收数据，
# 因此，只有异步客户端才能创建这样一种场景：请求数据的速度比读取数据的速度快。
#
# 相反，对于pubsub和副本客户端有一个默认的限制，因为订阅者和副本以推送方式接收数据。
#
# 硬限制或软限制都可以通过将它们设置为零来禁用。
#
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit replica 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60

# 客户端查询缓冲区会累加新命令，默认情况下，缓冲区大小是一个固定值
# 以避免协议同步失效(如客户端的错误)导致查询缓冲区出现未绑定的内存(即客户端都已经不存在了，但是它发过来的命令还在缓冲区当中)
# 但是，如果您有非常特殊的需求，例如巨大的multi/exec请求等，则可以在此处进行配置。
#
# client-query-buffer-limit 1gb

# 如果一个大容量请求(即客户端单次发送过来的字符串)被限制为512MB，我们也可以通过修改"proto-max-bulk-len"值
#
# proto-max-bulk-len 512mb

# Redis调用一个内部函数来执行许多后台任务，例如在超时时关闭客户端连接，清除从未请求的过期Key等。
#
# 注意：并非所有任务都以相同的频率执行，但是Redis会根据指定的"hz"值检查要执行的任务。
#
# 默认情况下，"hz"设置为10。提高该值将在Redis空闲时使用更多的CPU，
# 但同时当有多个键同时失效时，会使Redis响应更快，超时处理可能会更精确。
#
# 范围在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默认值10，并仅在需要非常低延迟的环境中将其提高到100。
hz 10
# 简单点说：Redis有后台任务，通过设置"hz"可提高或者降低检查这些任务是否应该执行的频率，值越大消耗的CPU越多，反之越少

# 通常，设置与连接的客户端数量成比例的HZ值是有用的。
# 例如，这对于避免每次后台任务调用处理过多的客户端，对避免延迟峰值很有用。
#
# 由于默认HZ值被保守地设置为10,Redis提供并默认启用了使用自适应HZ值的能力，当有许多连接的客户端时，HZ值将临时提高。
#
# 启用dynamic-hz后，实际配置的HZ将被用作基准，但是一旦连接了更多客户端，实际将使用配置的HZ值的倍数。
# 通过这种方式，空闲实例将使用非常少的CPU时间，而繁忙实例将更加响应。
dynamic-hz yes

# 当子进程重写AOF文件时，如果启用了以下选项，该文件将每生成32 MB的数据进行fsync。
# 这有助于将文件以增量方式提交到磁盘，并避免出现较大的延迟峰值。
aof-rewrite-incremental-fsync yes
```

```
# 当Redis保存RDB文件时，如果启用以下选项，该文件将每32 MB的数据生成fsync。
# 这有助于将文件以增量方式提交到磁盘，并避免出现较大的延迟峰值。
rdb-save-incremental-fsync yes

# Redis LFU内存淘汰策略(见maxmemory设置)可以调整。
# 但是，最好从默认设置开始，并在研究如何提高性能和键LFU如何随时间变化后才更改它们，这可以通过
OBJECT FREQ命令进行检查。
#
# Redis LFU实现中有两个可调参数：计数器对数因子和计数器衰减时间。
# 在更改两个参数之前，请务必先了解这两个参数的含义。
#
# LFU计数器每个密钥只有8位，最大值是255，因此Redis使用具有对数的概率增量。
# 给定旧计数器的值，当访问一个键时，计数器以这种方式递增：
#
# 1. 0到1之间的随机数R。
# 2. 概率P被计算为 1/(old_value*lfu_log_factor+1).
# 3. 仅当R < P时，计数器才会递增。
#
# 默认的lfu-log-factor是10。下表显示了频率计数器如何随着对数因子的访问次数变化情况：
#
# +--------+------------+------------+------------+------------+------------+
# | factor | 100 hits   | 1000 hits  | 100K hits  | 1M hits    | 10M hits   |
# +--------+------------+------------+------------+------------+------------+
# | 0      | 104        | 255        | 255        | 255        | 255        |
# +--------+------------+------------+------------+------------+------------+
# | 1      | 18         | 49         | 255        | 255        | 255        |
# +--------+------------+------------+------------+------------+------------+
# | 10     | 10         | 18         | 142        | 255        | 255        |
# +--------+------------+------------+------------+------------+------------+
# | 100    | 8          | 11         | 49         | 143        | 255        |
# +--------+------------+------------+------------+------------+------------+
#
# 注意1：上表是通过运行以下命令获得的：
#
#   redis-benchmark -n 1000000 incr foo
#   redis-cli object freq foo
#
# 注意2：计数器的初始值为5，以便让新对象有机会累积命中数。
#
# 计数器衰减时间是键计数器除以2所必须经过的时间，以分钟为单位。
#
# lfu-decaytime的默认值是1。0表示计数器每次被扫描时都会衰减。
#
# lfu-log-factor 10
# lfu-decay-time 1

############################ ACTIVE DEFRAGMENTATION #######################
#
# 什么是活动碎片整理?
# -------------------------------
#
# 活动(在线)碎片整理允许Redis服务器压缩内存中的小分配和数据回收之间的空间，从而允许收回内存。
#
# 碎片是每个分配器(但幸运的是，Jemalloc不那么常见)和某些工作负载都会发生的自然过程。
# 通常，需要重新启动服务器以减少碎片，或者至少清除所有数据并重新创建。
# 但是，由于Oran Agra为Redis 4.0实现了此功能，因此在服务器运行时，此过程可以在运行时以"热"方
式进行。
#
```

```
# 基本上，当碎片超过一定级别时(请参阅下面的配置选项)，
# Redis将通过利用某些特定的Jemalloc功能在连续的内存区域中创建值的新副本(以便了解分配是否导致碎片并分配更好的位置)，
# 同时将释放数据的旧副本。对于所有键，以增量方式重复此过程将导致碎片恢复到正常值。
#
# 重要事项：
#
# 1. 默认情况下，此功能是禁用的，并且仅当您编译Redis以使用我们随Redis的源代码提供的Jemalloc副本时才可用。
#    这是Linux构建的默认设置。
#
# 2. 如果没有碎片问题，则无需启用此功能。
#
# 3. 遇到碎片之后，可以在需要时使用命令“ CONFIG SET activedefrag yes”启用此功能。
#
# 配置参数能够微调碎片整理过程的行为。如果不确定它们的含义，最好不要更改默认值。

# 启用主动碎片整理
# activedefrag no

# Minimum amount of fragmentation waste to start active defrag
# active-defrag-ignore-bytes 100mb

# 启动主动碎片整理的最小碎片百分比，即有多少比例的碎片时开始整理
# active-defrag-threshold-lower 10

# 我们在最大程度地使用碎片的最大百分比
# active-defrag-threshold-upper 100

# 磁盘碎片整理的最小CPU百分比
# active-defrag-cycle-min 1

# 磁盘碎片整理的最大CPU百分比计算
# active-defrag-cycle-max 25

# 进行主字典扫描时处理的 set/hash/zset/list 字段的最大数量
# (就是说在进行主字典扫描时 set/hash/zset/list 的长度小于这个值才会处理，大于这个值的会放在一个列表中延迟处理)
# 因为如果某一个key过大，一次性处理完会非常耗时的
# active-defrag-max-scan-fields 1000

# 默认情况下，将启用用于清除的Jemalloc后台线程
jemalloc-bg-thread yes

# 可以将Redis的不同线程和进程固定到系统中的特定CPU，以最大化服务器的性能。
# 这不仅有助于将不同的Redis线程固定在不同的CPU中，而且还可以确保将在同一主机中运行的多个Redis实例固定到不同的CPU。
#
# 通常，您可以使用"taskset"命令执行此操作，但是在Linux和FreeBSD中，也可以直接通过Redis配置进行此操作。
#
# 您可以固定服务器/IO线程，bio线程，aof重写子进程，和bgsave子进程。指定cpu列表的语法与taskset命令相同:
#
# 设置redis服务器 io 线程cpu亲和度为0,2,4,6:
# server_cpulist 0-7:2
#
# 设置redis服务器 bio线程cpu亲和度为1,3:
```

```
# bio_cpulist 1,3
#
# 将aof重写子进程cpu亲和度设置为8,9,10,11:
# aof_rewrite_cpulist 8-11
#
# 将bgsave子进程cpu亲和度设置为1,10,11
# bgsave_cpulist 1,10-11
```