Redis配置文件解读

# 初衷：

无论是Redis的使用还是源码分析，始终跳不过的就是Redis配置文件的解读。由于当前网络环境中仍没有比较好的这方面的翻译文档，所以这里对其进行一个翻译。

# 出现位置：

Redis的默认配置文件是redis.conf。其位置在根目录下。如果需要使用配置文件，在启动server前还需将配置文件拷贝到待启动的文件夹下。如果需要自定义配置，则可进行相应的修改。当然，Redis也支持指定自定义的配置文件。

# 参考：

<http://wenku.baidu.com/link?url=3guczIcXwx_7JVS09DNs54XIvyZL2uT22H5Bm_F4ULrXbTtbETk5dDEahpzScPdEbztmmoMHwW1eyuqZ9ukQSG3inVgSollc4Sw6Z3HKJDy> redis.conf配置及说明

<https://github.com/chaopeng/redis-conf/blob/master/redis.conf> Redis配置文件翻译V2.6.10（2013）

<https://my.oschina.net/u/568779/blog/308129> Redis配置文件翻译V2.8（2014）

<http://www.aspku.com/database/redis/55733.html> Redis2.8配置文件中文翻译版（2015）

<http://www.cnblogs.com/xiaozong/p/5670333.html> Redis配置文件翻译 V3.2版本（2016）

# Redis配置文件解读

## 指定配置文件：

如果想要使用到配置文件，Redis服务必须以配置文件的路径作为第一个参数启动。如：./redis-server /path/to/redis.conf

## 单位说明：

当需要指定内存大小时，可能会使用到不同的单位，如1k、5GB、4M等，这里给出其单位含义：

**1k => 1000 bytes**

**1kb => 1024 bytes**

1m => 1000000 bytes

1mb => 1024\*1024 bytes

**1g => 1000000000 bytes**

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

指定单位是**大小写不敏感**。如1GB、1gB、1Gb是一样的。

## include使用：

用于include一个或多个配置文件。

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

当需要在一个标准的通用配置模板上进行一些个性化定制，则可以使用include 关键字来include这些个性化配置文件。

注意：虽然admin 或Redis Sentinel下执行的“CONFIG REWRITE”命令（Redis 2.8 引入的命令）会重写配置，但并不包含“include”关键字。也就是说“CONFIG REWRITE”覆盖“include”相关内容。

由于redis以最后的配置作为直接配置，所以建议将include命令放置在配置文件的最前面以防止配置被覆盖。

但是如果打算使用另外的配置文件来覆盖当前文件的部分或全部配置，那么则可将include命令放置到该文件的末尾。

Redis这里使用了**最后生效原则**，即**最后被解析的配置将作为最后的配置**。

格式如下：

include /path/to/local.conf

include /path/to/other.conf

## 网络配置：

对服务器网络相关的参数进行一个配置。

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

### 1 bind命令

我们知道，一台服务器上可能有多个网络接口，所以如果没有使用bind指定接口，Redis将监听该机器上的所有网络接口的连接请求。

如果仅需要监听一个或多个指定的接口，则可以使用“bind”命令来指定接口。

实例如下：

bind 192.168.1.100 10.0.0.1

bind 127.0.0.1 ::1

~~~ WARNING ~~~ 如果运行Redis服务的机器直接暴漏在Internet中，那么绑定所有的接口是一件危险的事。因为这样会将Redis服务暴漏给Internet中的每一个人。所以默认情况下，使用bind 127.0.0.1命令强制Redis监听IPv4环回接口地址，也就是说Redis仅接受本机的客户端请求。

服务器可以有一个网络接口（通常表述为网卡），或者多个。假设某机器上有两个网卡，分别为192.168.205.5 和192.168.205.6，如果bind 192.168.205.5，那么只有该网卡地址接受外部请求，如果不绑定，则两个网卡口都接受请求。

如果需要进行外网访问则需注销该命令行。在配置文件中，“#”代表注释。

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

### 2 保护模式

保护模式是Redis提供的安全防护层。设立该层是为了避免网络对未关闭Redis实例的随意访问。

该模式需要开启，当：

1） 没有使用“bind”命令明确需要绑定的地址；

2） 没有为Redis配置密码。

该模式启动后，服务器仅能接受使用IPv4、IPv6环回地址（127.0.0.1或::1）和本地Socket的客户端的连接请求。

默认情况下，保护模式是开启的。建议只有在已明确待连接的客户端无需授权或无需使用bind指定特定的接口时才关闭该模式。

使用如下：

protected-mode yes

### 3 端口

在指定的端口上进行监听，默认是 6379。当端口设置为0时，Redis就不会在TCP socket上进行监听。

使用如下：

port 6379

### 4 TCP listen() backlog设置

在一个并发量高的环境中，需要指定一个比较大的backlog值来避免慢连接情况的发生。注意，linux内核会默认使用/proc/sys/net/core/somaxconn值来减小backlog实际值。因此为了获得期望的值，需要确保增大 somaxconn 和 tcp\_max\_syn\_backlog 这两个值。

建议配置：

tcp-backlog 511

### 5 Unix socket

指定Unix socket路径来进行连接监听。默认是不指定，因此redis不会在Unix socket上进行监听。

使用方法如下：

unixsocket /tmp/redis.sock

unixsocketperm 700

### 6 Client timeout

当client在空闲N秒后，关闭该连接（0表示不处理空闲连接，默认方式）

实例：

timeout 0

### 7 TCP keepalive时间

当该值非零时，如果通信缺失，Redis会使用SO\_KEEPALIVE发送TCP ACKs给客户端。这样做的好处有二：

1. 检测已经死亡对端。（TCP关闭存在无法完成4次握手的情况，如断电，断网，数据丢失等等）
2. 保存已有连接的活性。

在Linux中，该指定时间是一次发送ACKs的时间片。对于其他内核系统，其时间片大小与内核配置有关。

一个比较合理的值是300 seconds。Redis 3.2.1版本之后默认指定该值为300 seconds。

实例如下：

tcp-keepalive 300

## 通用配置：

对一些通用参数进行配置。

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

### 1 daemon

默认情况下，Redis并不是一个守护进程，如果需要将Redis设置成守护进程，则可以使用daemonize yes进行配置。注意：当Redis作为守护进程时， 其pid 文件为 /var/run/redis.pid。

使用如下：

daemonize no

### 2 supervision

Redis 3.2新增命令。如果需要在机器启动（upstart模式 或systemd模式）时就启动Redis服务器，可以通过该选项来配置Redis。

支持的模式：

supervised **no**  – 无，不会与supervised tree进行交互

supervised **upstart** – 将Redis服务器添加到SIGSTOP 模式中

supervised **systemd** – 将READY=1 写入 $NOTIFY\_SOCKET

supervised **auto** - 根据环境变量UPSTART\_JOB 或NOTIFY\_SOCKET检测upstart 还是 systemd

注意，上述supervision方法（upstart或systemd）仅发出“程序已就绪”信号，不会继续给supervisor返回ping回复。

默认是不开启：

supervised no

### 3 pid文件

如果指定了pid文件，Redis会在启动时写该pid文件，在退出时删除该文件。

当Redis服务器已守护进程启动时，如果指定了配置文件，则直接使用，如果没有指定，则创建/var/run/redis.pid作为配置文件。

使用如下：

pidfile /var/run/redis\_6379.pid

### 4 日志级别

指定服务器的verbosity级别。Redis提供四种级别：

debug ---- 包含大量信息，用于开发和测试

verbose ---- 包含一些稀有的有用信息，但没有debug级别混乱

notice ---- 适量提示信息，用于生产环境

warning ---- 只包含非常重要和关键的信息

默认是notice级别：

loglevel notice

### 5 日志文件名称

指定日志文件名称。指定为空时将输出到标准输出设备中。如果Redis以守护进程启动，当日志文件名称为空时，日志将会输出到 /dev/null。

默认设置如下：

logfile ""

### 6 写系统日志

**6.1允许写系统日志**

将syslog-enabled设置为yes时， 允许将Redis服务日志记录到系统日志中。此外，还

可以使用更多的日志参数来满足特定要求。

实例如下：

syslog-enabled no

**6.2指定在系统日志身份**

一旦enable写入系统日志，可以指定服务在系统日志身份。实例如下：

syslog-ident redis

**6.3指定系统日志能力级别**

系统日志级别必须是 LOCAL0 到 LOCAL7 之间（闭区间）的值。实例如下：

syslog-facility local0

### 7 数据库数量

设置数据库的数量。默认使用0号数据库。可以在每一个连接上使用SELECT <dbid> 来指定另外的数据库，但是这个值必须在 0到 'database'-1之间。

实例如下：

databases 16

## 快照配置：

Redis提供快照功能，记录某一时刻数据库中保存的数据。

################################ SNAPSHOTTING ################################

### 1 DB持久化

将DB数据保存到磁盘。格式为：save <seconds> <changes>。当在规定的时间内seconds，如果写磁盘的次数超过了changes，则将DB数据保存到磁盘。如save 900 1表示在900秒内，如果对DB进行了至少一次写操作，则将DB数据持久化到磁盘上。

注意，可以通过注销所有save命令或将在所有save命令后追加save置空（save “”）命令来禁用save功能。使用示例：

save 900 1

save 300 10

save 60 10000

### 2 bgsave-error

默认情况下，在发生RDB快照或BGSAVE执行失败的那一刻，Redishi执行接收写请求。这会使用户察觉（通常比较困难）到数据没有正确的持久化到磁盘。否则有可能出现不被察觉的灾难性后果。

当后台BGSAVE程序可以再次开始工作时，Reidis会再次自动允许写入。

如果已经对Server和服务器持久化建立了正确的监控，那么当你禁用该功能后，即使磁盘、持久化等出现问题，Redis也能继续提供服务。

实例如下：

stop-writes-on-bgsave-error yes

### 3 rdbcompression

默认情况下，在dump RDB文件时，Redis采用LZF（一种高效的压缩算法）算法进行字符串对象数据的压缩，其性能较高。虽然LZF算法会消耗部分CPU性能，但是其数据压缩能够较高，所以建议不要关闭：

rdbcompression yes

### 4 RDB文件名称

该命令用于定义dump DB文件的文件名。格式如下：

dbfilename dump.rdb

### 5 工作目录

指定RDB文件所在目录。该目录也是AOF文件所在目录。注意，这里是指定一个目录而不是文件名。默认是当前目录，格式如下：

dir ./

## 复制配置：

Redis提供主从复制功能保证数据的可靠性。

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

### 1 主从关系建立

Redis主从复制。单机模式下，Redis支持使用slaveof命令从另一个Redis服务器的拷贝中来创建一个实例。集群模式下则使用cluster replicate <master-id>命令。Redis复制使用前须知：

1. Redis复制是异步复制，但是可以配置连接的从节点数量。
2. 当连接断开，Redis从节点支持部分重同步（psync）功能来保证主从节点数据同步。
3. 复制过程是一个自动化过程，无需人工干预。当出现网络分区后，从节点会自动尝试建立与主节点的连接，并尝试同步。

建立主从连接的命令如下：

slaveof <masterip> <masterport>

### 2 授权密码

当主节点开启密码保护时（通过配置"requirepass" 命令），从节点必须在开始复制同步前进行授权操作，否则其请求不会被接受。

命令如下：

masterauth <master-password>

注意，该命令只有在主节点开启密码保护时才生效。

### 3 从节点是否支持脏读

当主从断开连接或主从进行复制时，从节点对外拥有两种策略：

1. 如果“slave-serve-stale-data”参数设置成“yes”（默认情况下），则从节点可以响应客户端请求，尽管可能会恢复过期数据或空数据（如果主从第一次进行同步）。
2. 如果“slave-serve-stale-data”参数设置成“no”，则从节点会对除INFO 和SLAVEOF之外的命令返回"SYNC with master in progress"信息。

默认设置如下：

slave-serve-stale-data yes

### 4 从节点是否支持写入

指定从节点是否支持写操作。当需要存储一些临时数据时，让从节点支持写操作很有用。这样一来，就可以通过主从同步轻松的将已写入从节点的数据删除。但是，如果配置出错，也会出现客户端写入出错等问题。

注意，从Redis 2.6之后，从节点默认是只读的。

提示，只读的从服务器并不是设计给非信任的互联网客户端提供服务。这种模式的服务器设置只是一个用来防止对服务器实例进行误操作的保护层。默认情况下，只读从服务器能够输出管理员命令，如CONFIG、 DEBUG等。如果想限制只读从节点输出的管理型命令，可以通过'rename-command' 命令来隐藏这些管理员命令或危险命令。

# 提高它的安全性，使得她作为一个影子来执行管理或者危险的命令

默认设置如下：

slave-read-only yes

### 5 复制策略：有无磁盘

主从节点的数据同步策略有两种：磁盘同步、套接字同步。

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

WARNING: 无磁盘复制（DISKLESS REPLICATION I）当前仍处于实验阶段

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

#

# 对于新连接的slaves或断开重连的slaves将无法执行“部分同步”，需要进行一次完全同# 步。当进行完全同步时，主节点将传播一个RDB文件给从节点。该RDB文件的传播方式# 有两种：

# 1）基于磁盘：Redis主节点创建一个新进程将RDB文件写到磁盘，然后将生成的RDB文件传播给从节点。

# 2）无磁盘：Redis主节点创建一个新进程直接将RDB文件写到slaves的套接字中，RDB文件无需落盘。

# 基于磁盘的复制，一旦RDB文件生成，多个slaves将排队等待并可以共享该文件。而无磁盘复制一旦开始传输数据，新slaves到来后将会排队等待。

# 在使用无磁盘复制时，主节点在开始传输同步数据前将根据配置的时间进行等待，从而实现多个从节点的并发传输。

# 在磁盘速度缓慢且网络速度很快（高带宽）时，无磁盘复制效率更高。默认情况下，无磁盘复制同步关闭。配置如下：

repl-diskless-sync no

### 6 无磁盘复制等待时间

无磁盘复制前，主节点需要等待的时间。该配置在启用无磁盘复制时将生效。

# 由于一旦开启一次数据传输，其余slaves将排队等待，所以最好让主节点等待一段时间，这样主节点就可对多个slaves并发传播数据。

# 等待的单位是秒（second），默认是5秒。一旦将其设置为0，主节点将会马上开始数据传输。默认配置如下：

repl-diskless-sync-delay 5

### 7 slaves定时向master发送PING的时间片

默认情况下，slaves每10秒向master发送一次PING消息。可以根据网络等因素进行设置。该配置默认在配置文件中关闭：

# repl-ping-slave-period 10

### 8 复制超时阈值

# 以下情境将使用到复制超时阈值：

# 1) 从节点在执行SYNC期间，检测块文件传输超时

# 2) 从节点检测主节点离线（data、pings）

# 3) 主节点检测从节点离线（REPLCONF ACK）

# 必须要确保复制超时阈值（repl-timeout）大于slaves定时向master发送PING的时间片（repl-ping-slave-period），否则将总会检测到复制超时（当slave发送PING的时间片大于复制超时阈值时，slave还未发送ping就会被定性为复制超时）。使用格式如下：

# repl-timeout 60

### 9 TCP\_NODELAY功能

# 执行完SYNC后，是否要禁用TCP\_NODELAY。

#当禁用该功能后，Redis会使用占用更少带宽的小TCP包向从节点发送数据。但是这样做将会增大从节点端数据传输延时。在Linux下禁用TCP\_NODELAY功能将导致40 微秒的延迟。

# 当启动该功能后，在进行复制时将会减少数据传输延迟，但是会占用更大的带宽。

# 默认情况下，我们优先选择低延迟，但是在高速网络或主从节点存在多hops路径时，建议禁用TCP\_NODELAY功能。

# 默认开启TCP\_NODELAY功能。格式如下：

repl-disable-tcp-nodelay no

### 10 复制积压缓冲区大小

# 设置复制积压缓冲区（replication backlog）大小。当slaves断开与节点连接后，Redis使用复制积压缓冲区记录需要未发送给slave的数据。当从节点重连后，仅需执行一次部分同步，将从节点缺失数据补全。

# 复制积压缓冲区（replication backlog）越大，Redis可以支持的slave离线时间就越长。复制积压缓冲区用于部分重同步。

# 复制缓冲区只有在有slave连接时才分配内存。没有slave时，该内存会被释放出来，默认大小为1m。格式如下：

# repl-backlog-size 1mb

### 11 复制积压缓冲区释放时间片

# 当主节点不再有新连接的从节点后，复制积压缓冲区将会被释放。为避免因从节点频繁掉线后上线而频繁的进行复制积压缓冲区的释放与申请，Redis提供复制积压缓冲区释放时间片（repl-backlog-ttl）参数，保证主节点在检测到从节点掉线后的规定时间内不会释放该缓冲区。

# 值为零时表示不会释放该复制积压缓冲区。

# 单位为秒，配置如下：

# repl-backlog-ttl 3600

### 12 从节点优先级

# 使用整数表示从节点优先级。

# 当主节点无法正常工作后，Sentinel将使用该优先级在从节点中推选出新的主节点。

# 优先级对应的整数值越小，被推选成主节点的可能性更大。但是当优先级的值为零时表示该从节点不具备成为主节点的身份。

#　默认优先级为100。配置形式如下：

slave-priority 100

### 13 从节点连接数及从节点延时设置

# 当主节点的已连接从节点数小于N且这些从节点延迟均大于M秒，该主节点将停止接收写请求。

# 从节点处于“online”状态，当且仅当延迟（通过计算距离上一次接收从节点的ping消息的时间间隔获得）小于指定的阈值。

# 这个选项配置不是用来保证N个部分接收写信息，而是为了在没有足够的从节点可用时，限制写丢失。

# 如需要至少需要3个从节点并在10s内可用，则设置：

# min-slaves-to-write 3

# min-slaves-max-lag 10

# 一旦对这两个中的一个赋值为零，则该功能失效。

# 默认min-slaves-to-write 参数设置为0，即该功能默认不启用。

### 14 从节点指定的IP和port

# Redis主节点可以通过多种途径显示已连接从节点的IP和port。如Sentinel 可以使用“INFO replication”命令来发现从节点实例；Master可以使用“ROLE”命令显示从节点IP和port信息等。

# slave获取IP和port的方式是：

# IP：自动检测获取。当从节点连接主节点时，通过检查对应套接字地址获取。

# Port：从节点在复制中和主节点握手时需要使用到port。通常情况下，port即为连接时的port。

# 但是，当发生端口转发（port forwarding，转发一个网络端口从一个网络节点到另一个网络节点的行为）或使用NAT（Network Address Translation，网络地址转换）技术是，从节点需要被分配不同IP和port后才能被访问。

# 接下来的两个配置用来设置从节点的IP和port，用来告知主节点所指定的IP和port，这样INFO和ROLE 才能继续返回结果。

# 当需要重写IP和port时，则无需配置该选项。

# 配置格式如下：

#

# slave-announce-ip 5.5.5.5

# slave-announce-port 1234

## 安全配置：

Redis提供一些安全策略，尽量保证访问安全性。

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

### 1 认证密码

# Server在处理客户端命令前，该客户端需要提供提供认证密码。这在非可信网络环境中很有用。

# 为减少后台执行复杂度，这个选项一般都会被注释掉。因为大多数用户不需要授权。（如用户使用自己的服务器）

#

# Warning: 由于Redis执行高效，所以外部用户每秒可以尝试认证15w次。也就是说，为避免密码被快送攻破，用户需要使用一个极其复杂的密码。

# 设置认证密码格式如下：

# requirepass foobared

### 2 命令重命名

# 重命名命令。

# 在一个共享环境中有必要对危险命令进行重命令，从而避免危险命令的滥用、无用。

# 如给CONFIG命令重新设置一个难以猜测的命令，这样这个命令就很难被普通用户使用的，但仍能被内部工具使用。

# 如：

#

# rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52

# 当然，有时需要禁用一些命令。可以通过将命令置空实现：

#

# rename-command CONFIG ""

# 注意，一定要避免重命名那些写AOF文件或传输数据给slaves的命令，否则将会导致各种难以预料的错误。

## 限制配置：

对一些参数范围进行限定。

################################### LIMITS ####################################

### 1 最大客户端连接数

设置某一时刻客户端的并发连接数。默认情况下，限制上限是1w。但是，如果Redis服务器没有配置进程文件limit，那么可允许连接的最大客户端个数将被设置为当前文件限制的最小值32（Redis会保留一部分文件给内部用户）。

# 一旦得到了连接上限，Redis将关闭所有新连接并发送错误“max number of clients reached”提示。连接上限配置格式如下：

# maxclients 10000

### 2 最大可用内存

# 不要再内存超过指定限制时仍然使用内存。

# 当达到内存上限时，Redis会根据选定的过期键策略移除一些key。

# 如果根据过期键策略仍不能移除一些键或者过期键策略设置成“noeviction”（不启用过期键策略），那么Redis会向如SET、LPUSH等使用内存的命令返回错误，向诸如GET等读命令正常返回结果。

# 这个配置通常在将Redis当过LRU 缓存或对以设置硬性的内存上限的Redis很适用。

#

# WARNING: slaves的输出缓冲区不在主节点的maxmemory计算中，所以设置的maxmemory不宜过大。如果过大，可能导致主机的剩余内存过小，从而不能预留足够的内存用于创建slaves的输出缓冲区。

# 简言之，如果当前节点存在已连接的从节点，建立设置一个较小的maxmemory上限，这样系统就可以有多余的RAM用与创建从节点输出缓存。（当过期键策略设置成'noeviction'时，则没有必要这么做）

# 设置格式如下：

#

# maxmemory <bytes>

### 3 内存淘汰策略

# MAXMEMORY POLICY: 当达到maxmemory时，可以采用的内存淘汰策略。

# Redis提供五种内存淘汰策略：

# volatile-lru：利用LRU算法移除过期keys。

# allkeys-lru：利用LRU算法移除keys。

# volatile-random：随机移除过期keys。

# allkeys-random：随机移除keys。

# volatile-ttl：按照最近过期时间来删除（辅以TTL），移除即将过期的keys。

# noeviction：不移除任何key，只是返回一个写错误。

#

# Note: 不管采用了上述何种淘汰策略，当没有合适的键进行移除时，Redis仍会返回写错误。

# 这些写命令是： 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

#

# 默认内存淘汰策略是’noeviction’。

#

### 4 最大样例检测数

# LRU算法和最小TTL算法都不是精确算法，所以可以对其进行执行速度或精确度上的判定。

# 默认情况下，Redis会从五个键中选择一个最近最久未使用的键进行淘汰。可以通过配置该选择设置检测基数。

# 一般情况下，五个检测样本可以获得足够好的结果。10个样本的结果更接近LRU算法，但是会消耗更多的CPU。3个样本可以获得较高的执行速度但是不够精确。

# 配置格式如下：

# maxmemory-samples 5

## AOF配置：

AOF持久化模式对应的一些配置。

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

### 1 是否开启AOF持久化功能

# 默认情况下，Redis会异步将数据集快照到磁盘上。尽管这种模式对许多应用友好，但是当Redis进程崩溃或发生掉电时，几分钟内的写信息将丢失（根据快照执行的粒度）。

# AOF作为一种可替换的持久化策略，能够提供更好的耐久性。如使用默认的fsync策略，Redis仅会丢失1s的写信息，当发生突发事件（服务器掉电、服务器进程崩溃但OS运行正常）

# AOF持久化和RDB持久化可以同时开启。如果在启动Redis时已经存在AOF文件，则会直接加载AOF文件（考虑到AOF文件相比RDB文件有更好的耐久性）。

#

# 关于AOF的更多讯息见： http://redis.io/topics/persistence for more information.

# 默认AOF关闭，配置如下：

appendonly no

### 2 指定AOF文件名称

# 指定AOF文件名称，默认是appendonly.aof。配置如下：

appendfilename "appendonly.aof"

### 3 fsync()系统函数调用频率

# 调用fsync()系统函数用来告知OS将数据写入磁盘，而不是在输出缓冲区等待数据。有些OS将直接flush数据到磁盘，有些其他的OS仅会尝试去flush。

# Redis支持三种fsync()调用模式：

# no：不执行fsync,由OS决定flush数据的频率。高效。Linux下默认是每30s执行一次flush。

# always：每写入一次AOF就调用一次fsync。慢，最安全。

# everysec：每秒调用一次fsync。适中。

#

# 默认fsync的调用频率是“everysec”，这种策略在执行速度和数据安全进行折中。

# 当可以容忍一定程度数据丢失并期望更高的性能时，可以使用“no”策略（由操作系统决定flush的频率）。相反的，如果不能容忍数据丢失，可以使用“always”获得更好的安全性，尽管执行更慢。

#更多AOF信息可以参考：

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

#

# 在无法确定fsync调用频率时，推荐使用“everysec”策略。

# 在开启AOF持久化功能后，该配置才会生效。

# 配置设置如下：

# appendfsync always

appendfsync everysec

# appendfsync no

### 4 AOF rewrite与fsync

# 当AOF执行fsync的策略是always和everysec时，如果此时有一个后台进程（BGSAVE进程或AOF rewrite进程）正在执行大量的I/O操作到磁盘，在一些Linux系统中，执行fsync会造成较长的阻塞。当前对这种情况还没有很好的解决策略，即使在不同的线程中执行fsync也会导致调用同步write(2)阻塞。

# 为了缓解上述问题，可以通过配置下述选项来避免在主线程调用fsync()时执行BGSAVE或 BGREWRITEAOF带来的阻塞。

# 也就是说，默认情况下，当子进程执行BGSAVE或BGREWRITEAOF时，Redis的耐久性将默认转变成"appendfsync none"。在实际的应用中就意味着在最坏的场景下将丢失30s的数据，即使配置了fsync调用频率为always或everysec。（默认情况下，Linux每30s自动调用一次fsync将缓存数据flush到磁盘）

# 如果当前应用已考虑延迟问题，则将该配置设置成“yes”。否则使用默认配置（“no”），这是从耐久性角度考虑的最安全的选择。

no-appendfsync-on-rewrite no # 该配置等价于appendfsync-on-rewrite（在rewrite时仍执行fsync）

### 4 AOF rewrite触发时机

# 设置AOF 重写的触发条件。

# 当AOF日志按照指定的比例增长时，可以通过调用BGREWRITEAOF执行自动的AOF rewrite。

# 工作原理：Redis通过对比上一次执行rewrite时AOF文件的大小与当前AOF文件大小（在重启时将没有上一次执行rewrite的记录，这时将使用startup时的AOF文件大小），决定是否进行rewrite。

# 如果当前AOF对于上一次执行rewrite的AOF文件的增长比率大于指定的比率，将会触发一次rewrite。

# 当然，还需指定一个AOF进行重写的最小单位。这样做可以避免增长比率已经达到要求，但对应的AOF仍很小的情况（这种情况下没有必要进行rewrite）的发生。

# 如果想要关闭自动AOF rewrite功能，可将进行rewrite要求的增长比率设置0。

# 默认当AOF大于64MB且相比于上一次rewrite，AOF以扩充了两倍时会触发一次rewrite执行。默认配置如下：

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

### 5 AOF 文件不完整

# 在将AOF文件加载到内存时（重启Redis），可能会出现AOF被截断的情况。

# 如当Redis运行所在系统突然崩溃（当ext4文件系统在安装时没有配置成数据按序存储），会出现AOF被截断情况。

# 如果Redis程序发生崩溃或异常，但操作系统仍能正常工作，则不会出现AOF被截断的情况。

# 出现AOF被截断后，Redis要么直接退出并返回错误，要么加载被截断AOF中尽可能多的数据（当前默认方式）。

# 可以通过aof-load-truncated选项进行配置

# 当aof-load-truncated设置成“yes”，Redis仍会加载一个被截断的AOF文件，同时向用户报告AOF文件被截断。如果设置成“no”，Redis会直接返回错误并拒绝启动，这时用户需要使用"redis-check-aof"程序修复AOF，只有这样才能重启Server。

# 注意，如果AOF文件在执行一半时就出现问题，即使设置aof-load-truncated为 “yes”，Redis也会直接退出并返回错误。

# 这个配置仅在Redis尝试从AOF文件读更多数据但发现没有足够字计数存在时有意义。

# 默认开启该功能。

aof-load-truncated yes

## LUA脚本处理：

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

### 1 Lua脚本执行超时阈值

# 设置Lua脚本执超时的时间上限，单位是毫秒，milliseconds。

# 当Lua脚本执行超时，Redis会记录脚本执行之后的结果（超时后）并向查询返回错误。

# 当一个长时脚本执行时间超过最大执行时间时，只有SCRIPT KILL和 SHUTDOWN NOSAVE 命令可用。停止这类脚本运行的第一个方法是调用一个非写命令。第二种方法是shut down 这个server，如果已经发送了一个写命令但用户并不想等待脚本自然终止。

# 如果想不限制脚本的执行时间并且不需要返回warning，可以将该参数设置成0或负数。

# 默认配置如下：

lua-time-limit 5000

## Redis 集群配置：

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

#

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# WARNING EXPERIMENTAL: 尽管当前Redis Cluster代码已经稳定，但是为了将这部分代码水准标记为“mature”，仍需要一批有分量的用户将该功能应用到生产环境。

即Redis Cluster功能仍需要生产实践的考验。

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

### 1 启动Redis集群功能

# 正常情况下，启动的Redis实例为非集群模式。只有当节点配置成集群模式时才能成为集群节点。如需以集群模式启动，取消下述配置的注释即可：

#

# cluster-enabled yes

### 2 集群配置文件命令

# 每个集群节点都有一个集群配置文件。该文件不是用来让用户编辑，而是持久化集群信息。

# 该文件由集群节点创建并更新。每个Redis 集群节点都需要有唯一的集群配置文件。所以在同一系统创建的多个Redis实例需要确保不存在配置文件名相同的情况发生（这样会导致集群配置文件重载）。

# 集群配置文件命令格式如下：

#

# cluster-config-file nodes-6379.conf

### 3 集群节点超时阈值

# 集群节点超时阈值用来作为节点不可达并被标记为失效状态的超时上限，单位是毫秒（millisecond）

# 大部分其他内部时间将其基本参考数。

# 设置方式如下：

# cluster-node-timeout 15000

### 4 从节点可发起故障转移的判定因子（slave-validity-factor）

# 当主节点失效后，Redis避免让存储数据过旧的从节点发起故障转移。

# 没有简单的方式可以直接准确判定从节点的”data age”，可以通过下面两个方面的检测实现：

# 1) 如果有多个从节点可以发起故障转移，可以让他们交换信息以选出数据状态最节点主节点的从节点。从节点可以通过offset进行排名并通过该排名延迟发起故障转移的时机。

# 2) 每个独立的从节点计算最近一次与主节点交互的时间。这里的交互可以是最近一次PING、最近一次接收到来自主节点的命令、与主节点断开连接的时间（当复制链接已经down掉时）。如果最近一次与主节点的交互已经足够久远，那么这个从节点将放弃进行故障转移。

#

# 在2)中的时间阈值可以由用户设定。特别地，当从节点的最近一次与主节点的交互远大于(node-timeout \* slave-validity-factor) + repl-ping-slave-period 时，这个从节点将不会执行故障转移。

# 例如假设node-timeout为30秒，slave-validity-factor参数为10，repl-ping-slave-period 为10秒，当从节点距离最近一次与主节点的交互时间大于310秒（30\*10+10）时，该从节点不能进行故障转移。

# 一个过大的从节点有效因子（slave-validity-factor）会允许存储过旧数据的从节点进行故障转移，而一个过小的从节点有效因子将会妨碍集群选择从节点成为新的主节点。

# 所以合理的设置从节点有效因子很重要。

# 为了获得最大的可用性，可以将从节点有效因子（slave-validity-factor）赋值为0。也就是说，从节点忽略距离最近一次与主节点交互的时间段，则是直接点尝试发起故障转移。（但是这种策略下，这些从节点仍会根据offset的排名来推迟发起故障转移的时间）

# 从节点有效因子（slave-validity-factor）值为0是唯一可以保证网络分区消失后，集群仍继续工作的值。

# 设置格式如下：

# cluster-slave-validity-factor 10

### 5 从节点迁移屏蔽因子（cluster-migration-barrier）

# Redis集群支持将从节点迁移到孤立主节点（orphaned masters），没有可以工作从节点的主节点）。该功能减少了集群孤立主节点故障但没有可工作从节点进行故障转移的情况的发生。

# 从节点可以迁移到孤立主节点当且仅当原来的主节点的剩余可工作从节点个数大于等于指定的可工作从节点数。这个数称为从节点迁移屏蔽因子（migration barrier）。当迁移屏蔽因子设置为1时，当且仅当主节点拥有至少两个可工作的从节点才允许其中从节点迁移到孤立主节点（orphaned masters）。该因子通常用来表明使用者需要为集群中的主节点配置从节点个数。

# 默认迁移屏蔽因子是1（从节点可以执行迁移当前仅当其主节点在该节点迁移后仍保有至少一个从节点）。如果想关闭该功能，只需将该参数设置成一个极大值即可。

# 允许将迁移屏蔽因子置零。这种行为仅在调试时有用，且在生产环境中存在极大风险。

# 配置格式如下：

# cluster-migration-barrier 1

### 6 是否支持集群部分可用

# 默认情况下，Redis集群将停止接收客户端请求（停止服务）当集群检测到存在哈希槽没有对应负责的节点。也就是说，如果集群部分down（如有一部分哈希槽没有对应的节点），整个集群最终将会不可用。（集群信息传播遵循最终一致性）

# 当所有的槽都再次有对应负责的节点后，集群将会自动再次可用。

# 但是有时希望即使集群只有部分槽有对应的节点，集群也能继续接受客户端请求并处理对应的键空间。为了达到上述目的，将ecluster-require-full-coverage 设置为“no”即可。

# 配置格式如下：

# cluster-require-full-coverage yes

# 创建集群的指导文档在<http://redis.io> 网站可以获得。

#

## 慢日志配置：

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

### 1 命令执行超时上限

# Redis慢日志系统用来记录执行时间较长的查询。这里的执行时间（“execution time”）不包括IO操作时间，如接收客户端的请求，返回请求结果等，而是实际执行命令的时间（此时线程处于阻塞状态，仅能执行该命令，不能同时处理其他请求）

# 可以使用两个参数配置慢日志：一个参数告知Redis执行时间超时阈值（单位是微秒，microseconds），这样一旦某个执行时间超过指定上限，将会被记录到慢日志中；另一个参数是慢日志的长度。慢日志使用环式结构存储超时命令。（当慢日志满后，新命令添加进去后，最老的命令将被踢出）

# 单位是微秒（microsecond，106微秒等于1秒）。当该值为负数时表示禁用slow log功能。当该值为零时，表示强制使用slow log记录每一条命令。

# 默认慢日志功能是开启的，slowlog-log-slower-than时间上限是104微秒。

slowlog-log-slower-than 10000

### 2 慢日志长度

# slow log保存在内存中，只要内存容量足够，可以随意设定慢日志长度。可以使用SLOWLOG RESET命令重新设置慢日志长度。

# 默认设置如下：

slowlog-max-len 128

## 延迟监测配置：

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

### 1 设置操作延迟判定上限

# Redis 延迟监测自系统通过对执行期间的操作的检测来收集与延迟相关的数据。

# 通过使用LATENCY命令，Redis用户可以获得延迟相关的图形、报告等信息。

# 延迟系统只会记录大于等于设置的latency-monitor-threshold值的操作。当该值为零时，

# 则表明关闭latency monitor。

#

# 默认情况下，latency monitor功能是关闭的，因为大多数场景下并不需要该功能。

# latency monitor可以在Redis运行时通过"CONFIG SET latency-monitor-threshold

# <milliseconds>"启动。

# latency monitor设置格式如下：

latency-monitor-threshold 0

## 事件通知配置：

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

### 1 设置事件通知

# Redis 可以通知那些已Pub/Sub客户端键空间发生的事件。

# 该功能对应的文档是http://redis.io/topics/notifications

# 例如，如果开启键空间通知功能且一个客户端对0号数据库上的“foo”key执行DEL操作，那么Redis将使用Pub/Sub发送两条消息：

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

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

#

# Redis对通知的事件进行了分类，每一类都使用唯一的字符标记：

# K 键空间（Key）通知，前缀为：\_\_keyspace@<db>\_\_

# E 键事件（Event）通知，前缀为：\_\_keyevent@<db>\_\_

# 对于所有命令类型和非键事件，均使用小写字母表示，且没有前缀。

# g 一般命令（Generic commands），如DEL, EXPIRE, RENAME等

# $ 字符串（String）命令

# l 列表（**l**ist）命令

# s 集合（set）命令

# h 哈希（hash）命令

# z 有序集合（sorted set）命令

# x 过期事件（过期键产生的事件）

# e 驱逐事件（因maxmemory而驱逐的事件）

# A g$lshzxe等类型的别称（Alias），如此一来就可以使用"AKE"代表所有的事件类型

#

# notify-keyspace-events 可以指定多个字符组成的字符串或空串。其中空串代表关闭通知功能。

# 例1：为了开启List事件和Genetic事件（从事件名称分类来说），可以使用如下设置：

#

# notify-keyspace-events Elg

#

# 例2：为了获取过期键的信息并发送到订阅的频道，即\_\_keyevent@0\_\_:expired use信息，可设置如下：

#

# notify-keyspace-events Ex

#

# 默认情况下，事件通知功能是关闭的，因为大多数用户并不需要这个功能且这个功能会带来额外的性能开销。

# 注意，如果没有指定键空间（K）通知还是键事件（E）通知，那么任何事件通知都不会被传送。

# 默认设置如下：

notify-keyspace-events ""

## 高级配置：

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

### 1 Hash类型

#　Hash数据类型的底层实现有压缩链表（ziplist）和哈希表（hash）。当且仅当存储的数据量小于hash-max-ziplist-entries且节点占用的容量小于hash-max-ziplist-value时才使用小数据量存储高效的ziplist结构存储。否则，使用哈希结构存储。

# 可以通过下述指令设置阈值：

hash-max-ziplist-entries 512

hash-max-ziplist-value 64

### 2 List类型

# List类型也可以通过特殊的方式来节省空间。

# 每个内部list节点允许存储的entries数量可以指定为已修订最大数量或最大元素数。

# 如指定-5到-1，其含义是：

# -5: max size: 64 Kb <-- 对于普通的工作负载，不建议使用

# -4: max size: 32 Kb <-- 不建议使用

# -3: max size: 16 Kb <-- 有时不建议使用

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

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

# 整数代表每个list节点准确存储指定数量的elements

# 最高效的参数设置是-2 (8 Kb size) 或 -1 (4 Kb size)

# 但是，如果需求很特殊，则应根据需要调整参数：

list-max-ziplist-size -2

### 3 List类型压缩深度设置

# List可以实现压缩

# 压缩深度是划定quicklist、ziplist等list在压缩时的节点范围。为了进行快速的push/pop操作，不会对list的head和tail进行压缩，只会对中间节点进行压缩。参数设置如下：

#

# 0: 关闭list压缩功能

# 1: 深度为1表示只有当list添加一个节点（无论从head还是tail添加该节点）后才开始进行压缩。

# 所以对于[head]->**node->node->...->node**->[tail]

# 只有黑体部分加入才会执行压缩操作。

# 2: 深度为2

# 对于链表：[head]->[next]->**node->node->...->node**->[prev]->[tail]

# 不会压缩head 或 head->next 或 tail->prev 或 tail，而仅压缩剩余部分。

# 3: 深度为3

# [head]->[next]->[next]->**node->node->...->node**->[prev]->[prev]->[tail]

# 设置格式如下：

list-compress-depth 0

### 4 集合(intset)类型

# Set数据类型的底层实现默认是intSet，也可以是hash。

# Set使用hash编码格式当且仅当字符串组成的set变成底数为10的整数，且其值范围在64位整数中。

# 下面的配置设置用来指定set可以使用特殊编码格式的阈值：

set-max-intset-entries 512

### 5 Sorted Set类型

# Sorted Set默认使用ziplist实现，也可通过skiplist编码实现来节省空间。

# 当且仅当Sorted Set中元素值大于zset-max-ziplist-value、元素数量大于zset-max-ziplist-entries时，才使用skiplist实现Sorted Set。

# 参数设置如下：

zset-max-ziplist-entries 128

zset-max-ziplist-value 64

### 6 HyperLogLog稀疏表示

# HyperLogLog稀疏表示阈值。16位的header部分也在limit中。当使用稀疏表示的HyperLogLog存储的字节超过了指定的阈值，它将转变成稠密表示。

#

# 不建议使用大于16000的值。当小于16000时能够获得较高存储效率。

# 建议的值是3000，该值可以在较少PFADD（在执行稀疏编码时时间复杂度是O(N)）操作执行的同时获得极高空间使用收益。当CPU问题无需考虑时，可将该值提升到10000，但其值不应超过 15000。

# 设置方式如下：

hll-sparse-max-bytes 3000

### 7 rehash处理

# 激活的rehash会占用每100毫秒的1 millisecond的CPU时间来对Redis hash table（存储数据库的键值对的hash table）执行rehash。在Redis中hash table使用惰性rehash：在rehashing时，hash table中执行的操作越多，rehash执行的步骤也越多。所以当server很空闲时，rehash将很简单，hash table也会有更多的内存可以使用。

# 为了在条件许可的情况下对hash table进行rehash，从而节省内存空间，默认情况下，rehash功能会每100毫秒中1 毫秒被调用一次。

# 如果不确定：

# 使用"activerehashing no" 如果当前应用环境很注重延迟、Redis仅允许2毫秒的延迟应答。

#　使用"activerehashing yes" 如果当前应用环境不是太注重延迟且想要尽可能块的释放内存空间。

# 默认该功能开启：

activerehashing yes

### 8 客户端输出缓冲区

# 客户端输出缓冲区可以用来强制断开那些不能够快速读取服务器数据的客户端连接。（如在Pub/Sub模式中，客户端不能够快速的处理publisher发送过来的消息）

# 客户端类型可以细分为三类：

#

# normal -> 普通客户端（包括MONITOR客户端）

# slave -> 从节点客户端

# pubsub ->订阅至少一个pubsub通道或模式的客户端

#

# client-output-buffer-limit 设置的通用格式如下：

#

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

#

# 一旦hard limit达到，客户端将直接断开连接。如果soft limit 达到，客户端连接将会持续soft seconds 后才断开连接。

#例如，当hard limit 是 32 MB（megabytes）、soft limit 在10 秒内持续超过16MB，如果客户端输出缓冲区（clients output buffer）超过32 MB 或客户客户端输出缓冲区（clients output buffer）超过16MB，并在接下来的10秒都高于16MB，客户端连接将马上断开。

# 默认情况下，不需对normal级别的clients进行约束因为这些客户端如果没有发起询问就不会接受数据。所以，只需对异步客户端进行约束因为异步客户端会出现请求速度大于read速度的情况。

# 因为订阅者和从节点使用推的方式接受数据，所以需要对pubsub 客户端和slave客户端设置默认的客户端输出缓冲区约束。

# 将hard limit 或 soft limit 置零表示关闭对应的功能。

client-output-buffer-limit normal 0 0 0

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

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

### 9 Redis心跳频率

# Redis调用一个内部函数来执行后台任务，如在timeout时关闭客户端连接，清除从未被请求的过期键，等等。

# 虽然并不是所有的tasks都使用同样的频率执行，但是Redis会根据指定的频率值来检测tasks的执行。

# 默认设置的值为10，即每秒执行10次。在Redis处于空闲提升该值时，将会消耗更多的CPU。但是，提升该值也会使Redis更精确的处理超时问题，并检测到更多的过期键。

#

# 该值设置范围是1到500。但是不建议将其设置大于100。大多数的用户建议使用默认的值（10），并根据应用环境的低延迟需求适当提升该值（峰值建议不要大于100）。

# 格式如下：

hz 10

### 10 AOF重写与磁盘同步

# 子进程在执行重写AOF文件时，如果启动该功能，则数据每增长32MB就进行一次文件磁盘同步。该功能对加快文件同步到磁盘、避免大的延迟峰值有很大帮助。

# 默认该功能启动。格式如下：

aof-rewrite-incremental-fsync yes