# Redis配置文件详解

## Redis配置文件详解

---

#redis-server绑定的网络地址
**bind 192.168.106.128**
#保护模式 - yes为不能被外网访问，只能采用回环地址访问
**protected-mode yes**
# 守护进程模式
# 默认情况下 redis 不是作为守护进程运行的，如果你想让它在后台运行，你就把它改成 yes
# 当redis作为守护进程运行的时候，它会写一个 pid 到 /var/run/redis.pid 文件里面
**daemonize yes**
# 当redis作为守护进程运行的时候，它会把 pid 默认写到 /var/run/redis.pid 文件里面，但是你可以在这里自己制定它的文件位置
**pidfile /var/run/redis_6379.pid**
# 监听端口号，默认为 6379，如果你设为 0 ，redis 将不在 socket 上监听任何客户端连接。
**port 6379**
# TCP接收队列长度,受/proc/sys/net/core/somaxconn和tcp_max_syn_backlog这两个内核参数的影响
# 在高并发的环境下,你需要把这个值调高以避免客户端连接缓慢的问题.此参数确定了TCP连接中已完成队列(完成三次握手之后)的长度,当然此值必须不大于Linux系统定义的/proc/sys/net/core/somaxconn值,默认是511,而linux的# 默认参数值是128。当系统并发量大并且客户端速度缓慢的时候，可以将这二个参数一起参考设定
**tcp-backlog 511**
# 一个客户端空闲多少秒后关闭连接(0代表禁用，永不关闭)
**timeout 0**
# 如果非零，则设置SO_KEEPALIVE选项来向空闲连接的客户端发送ACK
#单位是秒，表示将周期性的使用SO_KEEPALIVE检测客户端是否还处于健康状态，避免服务器一直阻塞，官方给出的建议值是300S
**tcp-keepalive** 300
#可以通过upstart和systemd管理Redis守护进程，这个参数是和具体的操作系统相关的。
**supervised no**
# 指定服务器调试等级
# 可能值：
# debug （大量信息，对开发/测试有用）
# verbose （很多精简的有用信息，但是不像debug等级那么多）
# notice （适量的信息，基本上是你生产环境中需要的）
# warning （只有很重要/严重的信息会记录下来）
**loglevel notice**
#指定日志文件名和保存位置
**logfile "./redis.log"**
# 设置数据库个数
# 默认数据库是 DB 0，你可以在每个连接上使用 select <dbid> 命令选择一个不同的数据库，但是 dbid 必须是一个介于 0 到 databasees - 1 之间的值
**databases 16**
# 启动redis实例时是否打印logo
**always-show-logo yes**
#根据给定的时间间隔和写入次数将数据保存到磁盘
# 900秒（15分钟）之后，至少有一条数据更新，则保存到数据文件中
# 300秒（5分钟）之后，至少有10条数据更新，则保存到数据文件中
# 60秒之后，至少有10000条数据更新，则保存到数据文件中
**save 900 1**
**save 300 10**
**save 60 10000**
#当硬盘因为权限等原因无法写入时，停止写入
**stop-writes-on-bgsave-error yes**
# 当导出到 .rdb 指定存储至本地数据库时是否压缩数据，默认是yes，redis采用LZF压缩，如果为了节省CPU时间 可以关闭该选项，但会导致数据库文件扁的巨大
**rdbcompression yes**
# 是否校验rdb文件，版本5的RDB有一个CRC64算法的校验和放在了文件的最后。这将使文件格式更加可靠。
**rdbchecksum yes**
# 指定rdb保存到本地数据库文件名

**dbfilename dump.rdb**
#作为主节点时不配置此项slaveof <masterip> <masterport>
slaveof 192.168.106.128 6379
# 工作目录
# 例如上面的 dbfilename 只指定了文件名，但是它会写入到这个目录下。这个配置项一定是个目录，而不能是文件名
**dir ./**
# 当master服务设置了密码保护时，slav服务连接master的密码
#masterauth !@#$%^&*
# 当一个slave失去和master的连接，或者同步正在进行中，slave的行为可以有两种表现：
#
# 1) 如果 slave-serve-stale-data 设置为 "yes" (默认值)，slave会继续响应客户端请求，
# 可能是正常数据，或者是过时了的数据，也可能是还没获得值的空数据。
# 2) 如果 slave-serve-stale-data 设置为 "no"，slave会回复"正在从master同步
# （SYNC with master in progress）"来处理各种请求，除了 INFO 和 SLAVEOF 命令。
**slave-serve-stale-data yes**
# 你可以配置salve实例是否接受写操作。可写的slave实例可能对存储临时数据比较有用(因为写入salve的数据在同master同步之后将很容易被删除）
**slave-read-only yes**
# 是否在slave套接字发送SYNC之后禁用 TCP_NODELAY?
# 如果你选择"yes"Redis将使用更少的TCP包和带宽来向slaves发送数据。但是这将使数据传输到slave上有延迟，Linux内核的默认配置会达到40毫秒
# 如果你选择了 "no" 数据传输到salve的延迟将会减少但要使用更多的带宽
**repl-disable-tcp-nodelay no**
# slave的优先级是一个整数展示在Redis的Info输出中。如果master不再正常工作了，哨兵将用它来选择一个slave提升为master
# 优先级数字小的salve会优先考虑提升为master，所以例如有三个slave优先级分别为10，100，25，哨兵将挑选优先级最小数字为10的slave。
# 0作为一个特殊的优先级，标识这个slave不能作为master，所以一个优先级为0的slave永远不会被哨兵挑选提升为master
**slave-priority 100**
# 密码验证
# 警告：因为Redis太快了，所以外面的人可以尝试每秒150k的密码来试图破解密码。这意味着你需要一个高强度的密码，否则破解太容易了
**requirepass !@#$%^&***
# redis实例最大占用内存，不要用比设置的上限更多的内存。一旦内存使用达到上限，Redis会根据选定的回收策略（参见：maxmemmory-policy）删除key
**maxmemory 3gb**
# 最大内存策略：如果达到内存限制了，Redis如何选择删除key。你可以在下面五个行为里选：
# volatile-lru -> 根据LRU算法删除带有过期时间的key。
# allkeys-lru -> 根据LRU算法删除任何key。
# volatile-random -> 根据过期设置来随机删除key, 具备过期时间的key。
# allkeys->random -> 无差别随机删, 任何一个key。
# volatile-ttl -> 根据最近过期时间来删除（辅以TTL）, 这是对于有过期时间的key
# noeviction -> 谁也不删，直接在写操作时返回错误。
**maxmemory-policy volatile-lru**
##################
lazyfree-lazy-eviction no
lazyfree-lazy-expire no
lazyfree-lazy-server-del no
slave-lazy-flush no
repl-diskless-sync no
list-max-ziplist-size -2
list-compress-depth 0
aof-use-rdb-preamble no
##################
#指出是否在每次更新操作后进行日志记录，如果不开启，可能会在断电时导致一段时间内的数据丢失，
#因为redis本身同步数据文件是按上面的save条件来同步的，所以有的数据会在一段 时间内只存在于内存中。
**appendonly no**
# aof文件名
**appendfilename "appendonly.aof"**
# fsync() 系统调用告诉操作系统把数据写到磁盘上，而不是等更多的数据进入输出缓冲区。
# 有些操作系统会真的把数据马上刷到磁盘上；有些则会尽快去尝试这么做。

#
# Redis支持三种不同的模式：
#
# no：不要立刻刷，只有在操作系统需要刷的时候再刷。比较快。
# always：每次写操作都立刻写入到aof文件。慢，但是最安全。
# everysec：每秒写一次。折中方案。
**appendfsync everysec**
# 如果AOF的同步策略设置成 "always" 或者 "everysec"，并且后台的存储进程（后台存储或写入AOF
# 日志）会产生很多磁盘I/O开销。某些Linux的配置下会使Redis因为 fsync()系统调用而阻塞很久。
# 注意，目前对这个情况还没有完美修正，甚至不同线程的 fsync() 会阻塞我们同步的write(2)调用。
#
# 为了缓解这个问题，可以用下面这个选项。它可以在 BGSAVE 或 BGREWRITEAOF 处理时阻止主进程进行fsync()。
#
# 这就意味着如果有子进程在进行保存操作，那么Redis就处于"不可同步"的状态。
# 这实际上是说，在最差的情况下可能会丢掉30秒钟的日志数据。（默认Linux设定）
#
# 如果你有延时问题把这个设置成"yes"，否则就保持"no"，这是保存持久数据的最安全的方式。
**no-appendfsync-on-rewrite yes**
# 自动重写AOF文件
**auto-aof-rewrite-percentage 100**
**auto-aof-rewrite-min-size 64mb**
# AOF文件可能在尾部是不完整的（这跟system关闭有问题，尤其是mount ext4文件系统时
# 没有加上data=ordered选项。只会发生在os死时，redis自己死不会不完整）。
# 那redis重启时load进内存的时候就有问题了。
# 发生的时候，可以选择redis启动报错，并且通知用户和写日志，或者load尽量多正常的数据。
# 如果aof-load-truncated是yes，会自动发布一个log给客户端然后load（默认）。
# 如果是no，用户必须手动redis-check-aof修复AOF文件才可以。
# 注意，如果在读取的过程中，发现这个aof是损坏的，服务器也是会退出的，
# 这个选项仅仅用于当服务器尝试读取更多的数据但又找不到相应的数据时。
**aof-load-truncated yes**
# Lua 脚本的最大执行时间，毫秒为单位
**lua-time-limit 5000**
# Redis慢查询日志可以记录超过指定时间的查询
**slowlog-log-slower-than 10000**
# 这个长度没有限制。只是要主要会消耗内存。你可以通过 SLOWLOG RESET 来回收内存。
**slowlog-max-len 128**
# redis延时监控系统在运行时会采样一些操作，以便收集可能导致延时的数据根源。
# 通过 LATENCY命令 可以打印一些图样和获取一些报告，方便监控
# 这个系统仅仅记录那个执行时间大于或等于预定时间（毫秒）的操作，
# 这个预定时间是通过latency-monitor-threshold配置来指定的，
# 当设置为0时，这个监控系统处于停止状态
**latency-monitor-threshold 0**
# Redis能通知 Pub/Sub 客户端关于键空间发生的事件，默认关闭
**notify-keyspace-events ""**
# 当hash只有少量的entry时，并且最大的entry所占空间没有超过指定的限制时，会用一种节省内存的
# 数据结构来编码。可以通过下面的指令来设定限制
**hash-max-ziplist-entries 512**
**hash-max-ziplist-value 64**
# 与hash似，数据元素较少的list，可以用另一种方式来编码从而节省大量空间。
# 这种特殊的方式只有在符合下面限制时才可以用
**list-max-ziplist-entries 512**
**list-max-ziplist-value 64**
# set有一种特殊编码的情况：当set数据全是十进制64位有符号整型数字构成的字符串时。
# 下面这个配置项就是用来设置set使用这种编码来节省内存的最大长度。
**set-max-intset-entries 512**
# 与hash和list相似，有序集合也可以用一种特别的编码方式来节省大量空间。
# 这种编码只适合长度和元素都小于下面限制的有序集合
**zset-max-ziplist-entries 128**
**zset-max-ziplist-value 64**
# HyperLogLog稀疏结构表示字节的限制。该限制包括
# 16个字节的头。当HyperLogLog使用稀疏结构表示
# 这些限制，它会被转换成密度表示。

```
# 值大于16000是完全没用的，因为在该点
# 密集的表示是更多的内存效率。
# 建议值是3000左右，以便具有的内存好处, 减少内存的消耗
hll-sparse-max-bytes 3000
# 启用哈希刷新，每100个CPU毫秒会拿出1个毫秒来刷新Redis的主哈希表（顶级键值映射表）
activerehashing yes
# 客户端的输出缓冲区的限制，可用于强制断开那些因为某种原因从服务器读取数据的速度不够快的客户端
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60
# 默认情况下，“hz”的被设定为10。提高该值将在Redis空闲时使用更多的CPU时，但同时当有多个key
# 同时到期会使Redis的反应更灵敏，以及超时可以更精确地处理
hz 10
# 当一个子进程重写AOF文件时，如果启用下面的选项，则文件每生成32M数据会被同步
aof-rewrite-incremental-fsync yes
# # 设置连接的客户端的最大数量，一旦达到限制，Redis将关闭所有新连接，发送一个错误“最大客户端数量
#maxclients 10000
```

---

## 哨兵配置文件详解 {哨兵所需配置项}

---

```
#sentinel端口
port 26379
#工作路径
dir "/usr/redis-cluster/redis-master-1/sentinel_data""
# 守护进程模式
daemonize yes
#关闭保护模式 否则只能在回环地址访问
protected-mode no
# 指明日志文件名
logfile "./sentinel.log"
#redis哨兵id
sentinel myid 5ddea8652fc4f4255081bc0f67527627f63f133F
#哨兵监控的master，监控主节点地址 sentinel monitor <groupName> <ip> <port> <quorum> [quorum数量 为最少有几个哨兵同意]（只要同意 Sentinel 的数量不达标，自动故障迁移就不会执行）
sentinel monitor mymaster 192.168.106.128 6379 2
#如果服务器在给定的毫秒数之内， 没有返回 Sentinel 发送的 PING 命令的回复， 或者返回一个错误， 那么 Sentinel 将这个服务器标记为主观下线（subjectively down，简称 SDOWN ）。
sentinel down-after-milliseconds mymaster 15000
#若sentinel在该配置值内未能完成failover操作（即故障时master/slave自动切换），则认为本次failover失败。
sentinel failover-timeout mymaster 18000
#设置master和slaves验证密码
sentinel auth-pass mymaster password
#指定了在执行故障转移时， 最多可以有多少个从服务器同时对新的主服务器进行同步 你可以通过将这个值设为 1 来保证每次只有一个从服务器处于不能处理命令请求的状态
sentinel parallel-syncs mymaster 1
#发生故障转移次数
sentinel config-epoch mymaster3 0
#被选举leader的次数
sentinel leader-epoch mymaster3 0
```

---

## 连接Redis哨兵

---

```
#连接哨兵 - 查看哨兵状态 [h为host 哨兵ip,p为port 哨兵端口号]
$>src/redis-cli -h 192.168.106.128 -p 26379
#查看master的状态
```

**sentinel master mymaster**
#查看salves的状态
**SENTINEL slaves mymaster**
#查看哨兵的状态
SENTINEL sentinels mymaster
#获取当前master的地址
SENTINEL get-master-addr-by-name mymaster
#查看哨兵信息 - 哨兵所监测的 主服务器状态
**info sentinel**

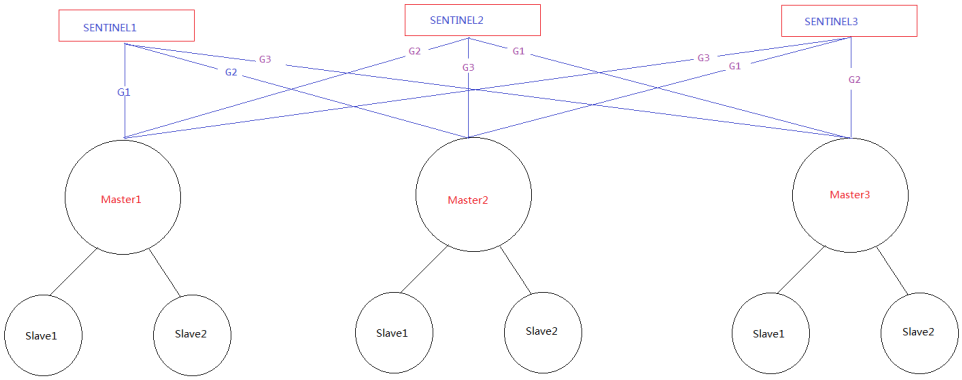
---

## 连接Redis server

#连接redis-server 查看server {master,slave}状态 // [h为host redis-server ip,p为port redis-server端口号]
$>src/redis-cli -h 192.168.106.128 -p 6379
#查看redis-server详细信息
info replication

redis-4.0.1.tar.gz
1.63MB

---




---