redis3.2 最新版本启动配置文件redis.conf详细说明

Redis启动的时候，可以指定配置文件，如下：

```
/usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf &
```

Redis.conf文件内容详细说明：

[python] view plain copy print ?

```
# 默认redis不是以后台进程的方式启动，如果需要在后台运行，需要将这个值设置成yes
# 以后台方式启动的时候，redis会写入默认的进程文件/var/run/redis.pid
daemonize yes

# redis启动的进程路径
pidfile/var/run/redis.pid

# 启动进程端口号，这里最好不要使用默认的6379，容易被攻击
port 7179

tcp-backlog 511

# 配置redis监听到的ip地址，可以是一个也可以多个
bind 127.0.0.110.254.3.42

# redis的sock路径
unixsocket/tmp/redis.sock
unixsocketperm 755

# 超时时间
timeout 0

#指定TCP连接是否为长连接,"侦探"信号有server端维护。默认为0.表示禁用
tcp-keepalive 0

# 日志级别，log 等级分为4 级，debug,verbose,notice, 和warning。生产环境下一般开启notice
loglevel notice

# 日志文件地址
logfile"/usr/local/redis/logs/redis.log"



# 设置数据库的个数，可以使用SELECT 命令来切换数据库。默认使用的数据库是0号库。默认16个库
databases 16

#RDB方式的持久化是通过快照（snapshotting）完成的，当符合一定条件时Redis会自动将内存中的所有数据进行快照并存储在硬盘上。进行快照的条件可以由用户在配置文件中自定义，由两个参数构成：时间和改动的键的个数。当在指定的时间内被更改的键的个数大于指定的数值时就会进行快照。RDB是Redis默认采用的持久化方式，在配置文件中已经预置了3个条件：
save 900 1    # 900秒内有至少1个键被更改则进行快照
save 300 10   # 300秒内有至少10个键被更改则进行快照
save 60 10000  # 60秒内有至少10000个键被更改则进行快照

# 持久化数据存储目录
dir/usr/local/redis/data



#当持久化出现错误时，是否依然继续进行工作，是否终止所有的客户端write请求。默认设置"yes"表示终止，一旦snapshot数据保存故障，那么此server为只读服务。如果为"no"，那么此次snapshot将失败，但下一次snapshot不会受到影响，不过如果出现故障,数据只能恢复到"最近一个成功点"
stop-writes-on-bgsave-errorno

#在进行数据镜像备份时，是否启用rdb文件压缩手段，默认为yes。压缩可能需要额外的cpu开支，不过这能够有效的减小rdb文件的大，有利于存储/备份/传输/数据恢复
rdbcompression yes

#checksum文件检测，读取写入的时候rdb文件checksum，会损失一些性能
rdbchecksum yes

#镜像备份文件的文件名，默认为dump.rdb
dbfilename dump.rdb

#当主master服务器挂机或主从复制在进行时，是否依然可以允许客户访问可能过期的数据。在"yes"情况下,slave继续向客户端提供只读服务,有可能此时的数据已经过期；在"no"情况下，任何向此server发送的数据请求服务(包括客户端和此server的slave)都将被告知"error"
slave-serve-stale-datayes

# 如果是slave库，只允许只读，不允许修改
slave-read-only yes



#slave与master的连接,是否禁用TCPnodelay选项。"yes"表示禁用,那么socket通讯中数据将会以packet方式发送(packet大小受到socket buffer限制)。可以提高socket通讯的效率(tcp交互次数),但是小数据将会被buffer,不会被立即发送,对于接受者可能存在延迟。"no"表示开启tcp nodelay选项,任何数据都会被立即发送,及时性较好,但是效率较低，建议设为no，在高并发或者主从有大量操作的情况下，设置为yes
repl-disable-tcp-nodelayno
```

```
#适用Sentinel模块(unstable,M-S集群管理和监控),需要额外的配置文件支持。slave的权重值,默认100.当master失效后,Sentinel将会从slave列表中找到权重值最低(>0)的slave,并提升为master。如果权重值为0,表示此slave为"观察者",不参与master选举
slave-priority 100

#限制同时连接的客户数量。当连接数超过这个值时，redis 将不再接收其他连接请求，客户端尝试连接时将收到error 信息。默认为10000，要考虑系统文件描述符限制，不宜过大，浪费文件描述符，具体多少根据具体情况而定
maxclients 10000

#redis-cache所能使用的最大内存(bytes),默认为0,表示"无限制",最终由OS物理内存大小决定(如果物理内存不足,有可能会使用swap)。此值尽量不要超过机器的物理内存尺寸,从性能和实施的角度考虑,可以为物理内存3/4。此配置需要和"maxmemory-policy"配合使用,当redis中内存数据达到maxmemory时,触发"清除策略"。在"内存不足"时,任何write操作(比如set,lpush等)都会触发"清除策略"的执行。在实际环境中,建议redis的所有物理机器的硬件配置保持一致(内存一致),同时确保master/slave中"maxmemory""policy"配置一致。
maxmemory 0



#内存过期策略，内存不足"时,数据清除策略,默认为"volatile-lru"。
#volatile-lru  ->对"过期集合"中的数据采取LRU(近期最少使用)算法.如果对key使用"expire"指令指定了过期时间,那么此key将会被添加到"过期集合"中。将已经过期/LRU的数据优先移除.如果"过期集合"中全部移除仍不能满足内存需求,将OOM.
#allkeys-lru ->对所有的数据,采用LRU算法
#volatile-random ->对"过期集合"中的数据采取"随即选取"算法,并移除选中的K-V,直到"内存足够"为止. 如果如果"过期集合"中全部移除全部移除仍不能满足,将OOM
#allkeys-random ->对所有的数据,采取"随机选取"算法,并移除选中的K-V,直到"内存足够"为止
#volatile-ttl ->对"过期集合"中的数据采取TTL算法(最小存活时间),移除即将过期的数据.
#noeviction ->不做任何干扰操作,直接返回OOM异常
#另外，如果数据的过期不会对"应用系统"带来异常,且系统中write操作比较密集,建议采取"allkeys-lru"
maxmemory-policyvolatile-lru

# 默认值5，上面LRU和最小TTL策略并非严谨的策略，而是大约估算的方式，因此可以选择取样值以便检查
maxmemory-samples 5

#默认情况下，redis 会在后台异步的把数据库镜像备份到磁盘，但是该备份是非常耗时的，而且备份也不能很频繁。所以redis 提供了另外一种更加高效的数据库备份及灾难恢复方式。开启append only 模式之后，redis 会把所接收到的每一次写操作请求都追加到appendonly.aof 文件中，当redis 重新启动时，会从该文件恢复出之前的状态。但是这样会造成appendonly.aof 文件过大，所以redis 还支持了BGREWRITEAOF 指令，对appendonly.aof 进行重新整理。如果不经常进行数据迁移操作，推荐生产环境下的做法为关闭镜像，开启appendonly.aof，同时可以选择在访问较少的时间每天对appendonly.aof 进行重写一次。
#另外，对master机器,主要负责写，建议使用AOF,对于slave,主要负责读，挑选出1-2台开启AOF，其余的建议关闭
appendonly yes

#aof文件名字，默认为appendonly.aof
appendfilename"appendonly.aof"

# 设置对appendonly.aof 文件进行同步的频率。always表示每次有写操作都进行同步，everysec 表示对写操作进行累积，每秒同步一次。no不主动fsync，由OS自己来完成。这个需要根据实际业务场景进行配置
appendfsync everysec

# 在aof rewrite期间,是否对aof新记录的append暂缓使用文件同步策略,主要考虑磁盘IO开支和请求阻塞时间。默认为no,表示"不暂缓",新的aof记录仍然会被立即同步
no-appendfsync-on-rewriteno

#当Aof log增长超过指定比例时，重写logfile，设置为0表示不自动重写Aof 日志，重写是为了使aof体积保持最小，而确保保存最完整的数据。
auto-aof-rewrite-percentage100
#触发aof rewrite的最小文件尺寸
auto-aof-rewrite-min-size64mb

#lua脚本执行的最大时间，单位毫秒
lua-time-limit 5000



#慢日志记录，单位微妙，10000就是10毫秒值，如果操作时间超过此值,将会把command信息"记录"起来.(内存,非文件)。其中"操作时间"不包括网络IO开支,只包括请求达到server后进行"内存实施"的时间."0"表示记录全部操作
slowlog-log-slower-than10000

#"慢操作日志"保留的最大条数,"记录"将会被队列化,如果超过了此长度,旧记录将会被移除。可以通过"SLOWLOG<subcommand> args"查看慢记录的信息(SLOWLOG get 10,SLOWLOG reset)
slowlog-max-len 128
notify-keyspace-events""

#hash类型的数据结构在编码上可以使用ziplist和hashtable。ziplist的特点就是文件存储(以及内存存储)所需的空间较小,在内容较小时,性能和hashtable几乎一样.因此redis对hash类型默认采取ziplist。如果hash中条目的条目个数或者value长度达到阀值,将会被重构为hashtable。
#这个参数指的是ziplist中允许存储的最大条目个数，，默认为512，建议为128
hash-max-ziplist-entries512
#ziplist中允许条目value值最大字节数，默认为64，建议为1024
hash-max-ziplist-value64

#同上
list-max-ziplist-entries512
list-max-ziplist-value64

#intset中允许保存的最大条目个数,如果达到阀值,intset将会被重构为hashtable
set-max-intset-entries512

#zset为有序集合,有2中编码类型:ziplist,skiplist。因为"排序"将会消耗额外的性能,当zset中数据较多时,将会被重构为skiplist。
zset-max-ziplist-entries128
#zset中允许条目value值最大字节数，默认为64，建议为1024
zset-max-ziplist-value64



#是否开启顶层数据结构的rehash功能,如果内存允许,请开启。rehash能够很大程度上提高K-V存取的效率
activerehashing yes

#客户端buffer控制。在客户端与server进行的交互中,每个连接都会与一个buffer关联,此buffer用来队列化等待被client接受的响应信息。如果client不能及时的消费响应信息,那么buffer将会被不断积压而给server带来内存压力.如果buffer中积压的数据达到阀值,将会导致连接被关闭,buffer被移除。
```

```
#buffer控制类型包括:normal -> 普通连接；slave->与slave之间的连接；pubsub ->pub/sub类型连接，此类型的连接，往往会产生此种问题;因为pub端会密集的发布消息,但是sub端可能消费不足.指令格式:client-output-buffer-limit <class> <hard><soft><seconds>",其中hard表示buffer最大值,一旦达到阀值将立即关闭连接;soft表示"容忍值",它和seconds配合,如果buffer值超过soft且持续时间达到了seconds,也将立即关闭连接,如果超过了soft但是在seconds之后，buffer数据小于了soft,连接将会被保留.其中hard和soft都设置为0,则表示禁用buffer控制.通常hard值大于soft.
client-output-buffer-limitnormal 0 0 0
client-output-buffer-limitslave 256mb 64mb 60
client-output-buffer-limitpubsub 32mb 8mb 60



#Redis server执行后台任务的频率,默认为10,此值越大表示redis对"间歇性task"的执行次数越频繁(次数/秒)。"间歇性task"包括"过期集合"检测、关闭"空闲超时"的连接等,此值必须大于0且小于500。此值过小就意味着更多的cpu周期消耗,后台task被轮询的次数更频繁。此值过大意味着"内存敏感"性较差。建议采用默认值。
hz 10

#当一个child在重写AOF文件的时候，如果aof-rewrite-incremental-fsync值为yes生效，那么这个文件会以每次32M数据的来被同步，这大量新增提交到磁盘是有用的，并且能避免高峰延迟。
aof-rewrite-incremental-fsyncyes
```