常规

daemonize no
Redis默认是不作为守护进程来运行的。你可以把这个设置为"yes"让它作为守护进程来运行。
注意，当作为守护进程的时候，Redis会把进程ID写到 /var/run/redis.pid

pidfile /var/run/redis.pid
当以守护进程方式运行的时候，Redis会把进程ID默认写到 /var/run/redis.pid。你可以在这里修改路径。

port 6379
接受连接的特定端口，默认是6379。
如果端口设置为0，Redis就不会监听TCP套接字。

bind 127.0.0.1
如果你想的话，你可以绑定单一接口；如果这里没单独设置，那么所有接口的连接都会被监听。

unixsocket /tmp/redis.sock
unix指定监听socket,指定用来监听连接的unxi套接字的路径。这个没有默认值，所以如果你不指定的话，Redis就不会通过unix套接字来监听。

unixsocketperm 755
#当指定监听为socket时，可以指定其权限为755

timeout 0
#一个客户端空闲多少秒后关闭连接。(0代表禁用，永不关闭)

loglevel verbose
设置服务器调试等级。
可能值：
debug （很多信息，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程度）
warning （只有很重要/严重的信息会记录下来）

logfile stdout
指明日志文件名。也可以使用"stdout"来强制让Redis把日志信息写到标准输出上。
注意：如果Redis以守护进程方式运行，而你设置日志显示到标准输出的话，那么日志会发送到 /dev/null

syslog-enabled no
要使用系统日志记录器很简单，只要设置 "syslog-enabled" 为 "yes" 就可以了。
然后根据需要设置其他一些syslog参数就可以了。

syslog-ident redis
指明syslog身份

syslog-facility local0
指明syslog的设备。必须是一个用户或者是 LOCAL0 ~ LOCAL7 之一

databases 16
设置数据库个数。默认数据库是 DB 0，你可以通过SELECT WHERE dbid（0 ~ 'databases' - 1）来为每个连接使用不同的数据库。

快照

```
save 900 1
save 300 10
save 60 10000
# 把数据库存到磁盘上:
# save
# 会在指定秒数和数据变化次数之后把数据库写到磁盘上。
#
# 下面的例子将会进行把数据写入磁盘的操作:
# 900秒（15分钟）之后，且至少1次变更
# 300秒（5分钟）之后，且至少10次变更
# 60秒之后，且至少10000次变更
#
# 注意：你要想不写磁盘的话就把所有 "save" 设置注释掉就行了。

rdbcompression yes

# 当导出到 .rdb 数据库时是否用LZF压缩字符串对象。
# 默认设置为 "yes"，所以几乎总是生效的。
# 如果你想节省CPU的话你可以把这个设置为 "no"，但是如果你有可压缩的key的话，那数据文件就会更大了。

dbfilename dump.rdb
# 数据库的文件名

dir ./
# 工作目录
# 数据库会写到这个目录下，文件名就是上面的 "dbfilename" 的值。
# 累加文件也放这里。
# 注意你这里指定的必须是目录，不是文件名。

################################################ 同步 ################################################

slaveof
# 主从同步。通过 slaveof 配置来实现Redis实例的备份。
# 注意，这里是本地从远端复制数据。也就是说，本地可以有不同的数据库文件、绑定不同的IP、监听不同的端口。

masterauth
# 如果master设置了密码（通过下面的 "requirepass" 选项来配置），那么slave在开始同步之前必须进行身份验证，否则它的同步请求会被拒绝。

slave-serve-stale-data yes
# 当一个slave失去和master的连接，或者同步正在进行中，slave的行为有两种可能：
#
# 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，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。

repl-ping-slave-period 10
# slave根据指定的时间间隔向服务器发送ping请求。
# 时间间隔可以通过 repl_ping_slave_period 来设置。
# 默认10。

repl-timeout 60
# 下面的选项设置了大块数据I/O、向master请求数据和ping响应的过期时间。
# 默认值60秒。
# 一个很重要的事情是：确保这个值比 repl-ping-slave-period 大，否则master和slave之间的传输过期时间
```

比预想的要短。

安全

requirepass foobared
要求客户端在处理任何命令时都要验证身份和密码。
这在你信不过来访者时很有用。
为了向后兼容的话，这段应该注释掉。而且大多数人不需要身份验证（例如：它们运行在自己的服务器上。）
警告：因为Redis太快了，所以居心不良的人可以每秒尝试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。

rename-command CONFIG ""
命令重命名
在共享环境下，可以为危险命令改变名字。比如，你可以为 CONFIG 改个其他不太容易猜到的名字，这样你自己仍然可以使用，而别人却没法做坏事了。
例如:
rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52
甚至也可以通过给命令赋值一个空字符串来完全禁用这条命令：

################################### 限制 ###################################

maxclients 128
设置最多同时连接客户端数量。
默认没有限制，这个关系到Redis进程能够打开的文件描述符数量。
特殊值"0"表示没有限制。
一旦达到这个限制，Redis会关闭所有新连接并发送错误"达到最大用户数上限（max number of clients reached）"

maxmemory
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。
如果因为删除策略问题Redis无法删除key，或者策略设置为 "noeviction"，Redis会回复需要更多内存的错误信息给命令。
例如，SET,LPUSH等等。但是会继续合理响应只读命令，比如：GET。
在使用Redis作为LRU缓存，或者为实例设置了硬性内存限制的时候（使用 "noeviction" 策略）的时候，这个选项还是满有用的。
警告：当一堆slave连上达到内存上限的实例的时候，响应slave需要的输出缓存所需内存不计算在使用内存当中。
这样当请求一个删除掉的key的时候就不会触发网络问题 / 重新同步的事件，然后slave就会收到一堆删除指令，直到数据库空了为止。
简而言之，如果你有slave连上一个master的话，那建议你把master内存限制设小点儿，确保有足够的系统内存用作输出缓存。
（如果策略设置为"noeviction"的话就不无所谓了）

maxmemory-policy volatile-lru
内存策略：如果达到内存限制了，Redis如何删除key。你可以在下面五个策略里面选：
#
volatile-lru -> 根据LRU算法生成的过期时间来删除。
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。
allkeys->random -> 无差别随机删。
volatile-ttl -> 根据最近过期时间来删除（辅以TTL）
noeviction -> 谁也不删，直接在写操作时返回错误。
#
注意：对所有策略来说，如果Redis找不到合适的可以删除的key都会在写操作时返回一个错误。
#

这里涉及的命令：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
#
默认值如下：

maxmemory-samples 3
LRU和最小TTL算法的实现都不是很精确，但是很接近（为了省内存），所以你可以用样例做测试。
例如：默认Redis会检查三个key然后取最旧的那个，你可以通过下面的配置项来设置样本的个数。

############################## 纯累加模式 ##############################

appendonly no
默认情况下，Redis是异步的把数据导出到磁盘上。这种情况下，当Redis挂掉的时候，最新的数据就丢了。
如果不希望丢掉任何一条数据的话就该用纯累加模式：一旦开启这个模式，Redis会把每次写入的数据在接收后都写入 appendonly.aof 文件。
每次启动时Redis都会把这个文件的数据读入内存里。
#
注意，异步导出的数据库文件和纯累加文件可以并存（你得把上面所有"save"设置都注释掉，关掉导出机制）。
如果纯累加模式开启了，那么Redis会在启动时载入日志文件而忽略导出的 dump.rdb 文件。
#
重要：查看 BGREWRITEAOF 来了解当累加日志文件太大了之后，怎么在后台重新处理这个日志文件。

appendfilename appendonly.aof
纯累加文件名字（默认："appendonly.aof"）

appendfsync always
appendfsync everysec
appendfsync no
fsync() 请求操作系统马上把数据写到磁盘上，不要再等了。
有些操作系统会真的把数据马上刷到磁盘上；有些则要磨蹭一下，但是会尽快去做。
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。折衷方案。
默认的 "everysec" 通常来说能在速度和数据安全性之间取得比较好的平衡。
如果你真的理解了这个意味着什么，那么设置"no"可以获得更好的性能表现（如果丢数据的话，则只能拿到一个不是很新的快照）；
或者相反的，你选择 "always" 来牺牲速度确保数据安全、完整。
如果拿不准，就用 "everysec"

no-appendfsync-on-rewrite no
如果AOF的同步策略设置成 "always" 或者 "everysec"，那么后台的存储进程（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() 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() 会阻塞我们的 write(2) 请求。
#
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 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）
#
如果你有延迟的问题那就把这个设为 "yes"，否则就保持 "no"，这是保存持久数据的最安全的方式。

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
自动重写AOF文件
如果AOF日志文件大到指定百分比，Redis能够通过 BGREWRITEAOF 自动重写AOF日志文件。
#
工作原理：Redis记住上次重写时AOF日志的大小（或者重启后没有写操作的话，那就直接用此时的AOF文件），
基准尺寸和当前尺寸做比较。如果当前尺寸超过指定比例，就会触发重写操作。
#
你还需要指定被重写日志的最小尺寸，这样避免了达到约定百分比但尺寸仍然很小的情况还要重写。
指定百分比为0会禁用AOF自动重写特性。

################################## 慢查询日志 ###################################

slowlog-log-slower-than 10000
Redis慢查询日志可以记录超过指定时间的查询。运行时间不包括各种I/O时间。
例如：连接客户端，发送响应数据等。只计算命令运行的实际时间（这是唯一一种命令运行线程阻塞而无法同时为其他请求服务的场景）
#
你可以为慢查询日志配置两个参数：一个是超标时间，单位为微妙，记录超过个时间的命令。
另一个是慢查询日志长度。当一个新的命令被写进日志的时候，最老的那个记录会被删掉。
#
下面的时间单位是微秒，所以1000000就是1秒。注意，负数时间会禁用慢查询日志，而0则会强制记录所有命令。

slowlog-max-len 128
这个长度没有限制。只要有足够的内存就行。你可以通过 SLOWLOG RESET 来释放内存。（译者注：日志居然是在内存里的Orz）

################################## 虚拟内存 ###################################

警告！虚拟内存在Redis 2.4是反对的。
非常不鼓励使用虚拟内存！！

vm-enabled no
vm-enabled yes
虚拟内存可以使Redis在内存不够的情况下仍然可以将所有数据序列保存在内存里。
为了做到这一点，高频key会调到内存里，而低频key会转到交换文件里，就像操作系统使用内存页一样。
#
要使用虚拟内存，只要把 "vm-enabled" 设置为 "yes"，并根据需要设置下面三个虚拟内存参数就可以了。

vm-swap-file /tmp/redis.swap
这是交换文件的路径。估计你猜到了，交换文件不能在多个Redis实例之间共享，所以确保每个Redis实例使用一个独立交换文件。
最好的保存交换文件（被随机访问）的介质是固态硬盘（SSD）。
* **警告** * 如果你使用共享主机，那么默认的交换文件放到 /tmp 下是不安全的。
创建一个Redis用户可写的目录，并配置Redis在这里创建交换文件。

vm-max-memory 0
"vm-max-memory" 配置虚拟内存可用的最大内存容量。
如果交换文件还有空间的话，所有超标部分都会放到交换文件里。
#
"vm-max-memory" 设置为0表示系统会用掉所有可用内存。
这默认值不咋地，只是把你能用的内存全用掉了，留点余量会更好。
例如，设置为剩余内存的60%-80%

vm-page-size 32
Redis交换文件是分成多个数据页的。
一个可存储对象可以被保存在多个连续页里，但是一个数据页无法被多个对象共享。
所以，如果你的数据页太大，那么小对象就会浪费掉很多空间。
如果数据页太小，那用于存储的交换空间就会更少（假定你设置相同的数据页数量）
#
如果你使用很多小对象，建议分页尺寸为64或32个字节。
如果你使用很多大对象，那就用大一些的尺寸。
如果不确定，那就用默认值 :)

vm-pages 134217728
交换文件里数据页总数。
根据内存中分页表（已用/未用的数据页分布情况），磁盘上每8个数据页会消耗内存里1个字节。
#
交换区容量 = vm-page-size * vm-pages
#
根据默认的32字节的数据页尺寸和134217728的数据页数来算，Redis的数据页文件会占4GB，而内存里的分页表会消耗16MB内存。
#
为你的应验程序设置最小且够用的数字比较好，下面这个默认值在大多数情况下都是偏大的。

vm-max-threads 4
同时可运行的虚拟内存I/O线程数。
这些线程可以完成从交换文件进行数据读写的操作，也可以处理数据在内存与磁盘间的交互和编码/解码处理。
多一些线程可以一定程度上提高处理效率，虽然I/O操作本身依赖于物理设备的限制，不会因为更多的线程而提高单次读写操作的效率。
#
特殊值0会关闭线程级I/O，并会开启阻塞虚拟内存机制。

################################ 高级配置 ################################

hash-max-zipmap-entries 512
hash-max-zipmap-value 64
当有大量数据时，适合用哈希编码（需要更多的内存），元素数量上限不能超过给定限制。
你可以通过下面的选项来设定这些限制：

list-max-ziplist-entries 512
list-max-ziplist-value 64
与哈希相类似，数据元素较少的情况下，可以用另一种方式来编码从而节省大量空间。
这种方式只有在符合下面限制的时候才可以用：

set-max-intset-entries 512
还有这样一种特殊编码的情况：数据全是64位无符号整型数字构成的字符串。
下面这个配置项就是用来限制这种情况下使用这种编码的最大上限的。

```
zset-max-ziplist-entries 128
zset-max-ziplist-value 64
# 与第一、第二种情况相似，有序序列也可以用一种特别的编码方式来处理，可节省大量空间。
# 这种编码只适合长度和元素都符合下面限制的有序序列：

activerehashing yes
# 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）。
# redis所用的哈希表实现（见dict.c）采用延迟哈希刷新机制：你对一个哈希表操作越多，哈希刷新操作就越频繁；
# 反之，如果服务器非常不活跃那么也就是用点内存保存哈希表而已。
#
# 默认是每秒钟进行10次哈希表刷新，用来刷新字典，然后尽快释放内存。
#
# 建议：
# 如果你对延迟比较在意的话就用 "activerehashing no"，每个请求延迟2毫秒不太好嘛。
# 如果你不太在意延迟而希望尽快释放内存的话就设置 "activerehashing yes"。

client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60
# 客户端输出缓存限制强迫断开读取速度比较慢的客户端
# 有三种类型的限制
# normal -> 正常的客户端包括监控客户端
# slave -> 从客户端
# pubsub -> 客户端至少订阅了一个频道或者模式
# 客户端输出缓存限制语法如下（时间单位：秒）
# client-output-buffer-limit <类别> <强制限制> <软性限制> <软性时间>
# 达到强制限制缓存大小，立刻断开链接。
# 达到软性限制，仍然会有软性时间大小的链接时间
# 默认正常客户端无限制，只有请求后，异步客户端数据请求速度快于它能读取数据的速度
# 订阅模式和主从客户端又默认限制，因为它们都接受推送。
# 强制限制和软性限制都可以设置为0来禁用这个特性

hz 10
# 设置Redis后台任务执行频率，比如清除过期键任务。
# 设置范围为1到500，默认为10.越大CPU消耗越大，延迟越小。
# 建议不要超过100

aof-rewrite-incremental-fsync yes
# 当子进程重写AOF文件，以下选项开启时，AOF文件会每产生32M数据同步一次。
# 这有助于更快写入文件到磁盘避免延迟

################################## 包含 ###################################

include /path/to/local.conf
include /path/to/other.conf
# 包含一个或多个其他配置文件。
# 这在你有标准配置模板但是每个redis服务器又需要个性设置的时候很有用。
# 包含文件特性允许你引人其他配置文件，所以好好利用吧。
```