**Redis的安装与配置详解**

**经过看好多别人写的日志总结，这里就不一一粘贴网址了**

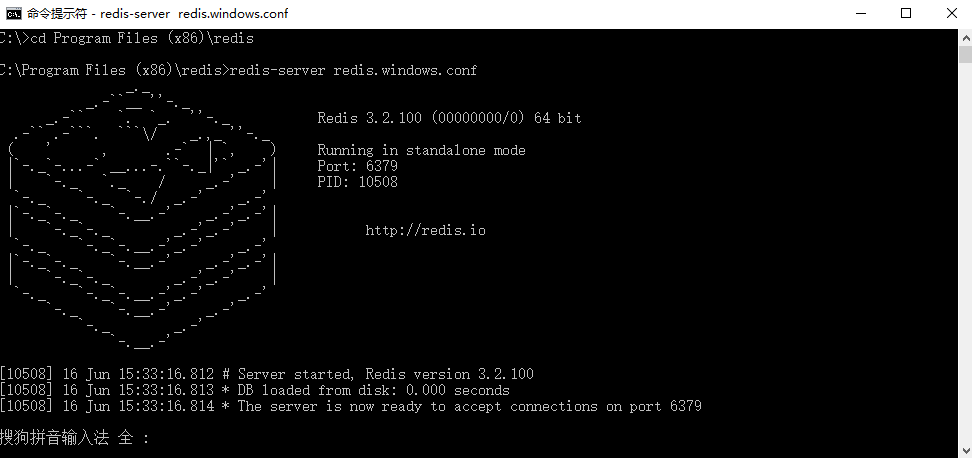
Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。从2010年3月15日起，Redis的开发工作由VMware主持。从2013年5月开始，Redis的开发由Pivotal赞助。

官方下载地址：<http://redis.io/download>，不过官方没有64位的Windows下的可执行程序，目前有个开源的托管在github上, 地址：<https://github.com/ServiceStack/redis-windows> 点击这个地方，下载文件并解压到自己的电脑目录下

|  |  |
| --- | --- |
| **文件名** | **简要** |
| redis-benchmark.exe | 基准测试 |
| redis-check-aof.exe | aof |
| redischeck-dump.exe | dump |
| redis-cli.exe | 客户端 |
| redis-server.exe | 服务器 |
| redis.windows.conf | 配置文件 |

**打开CMD 进入到刚刚解压的redis目录下**

**输入** redis-server redis.windows.conf

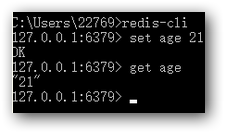


**这就说明成功了**

**再来就是如果不想这么麻烦就去设置环境变量**

**设置好之后**

**直接在cmd输入redis-cli 就启动了**

****

**虽然是启动成功了，但是你把窗口一关闭那就是把服务也关闭了，所以我们可以设置成windows下的服务**

**先用cmd进入到redis目录下，输入下面一段话 复制粘贴**

**redis-server --service-install redis.windows-service.conf --loglevel verbose**

**如果没有报错，那就说明成功了，打开windows服务就能看到**

**常用的redis命令**

卸载服务：redis-server --service-uninstall

开启服务：redis-server --service-start

停止服务：redis-server --service-stop

补充

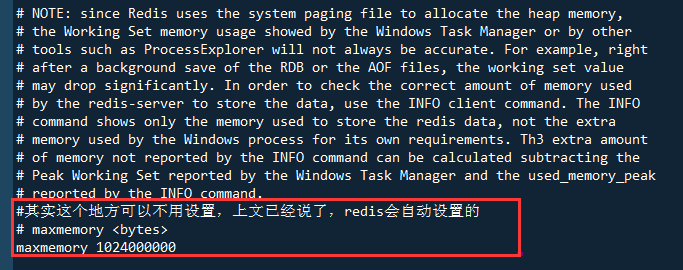
如果修改了redis.windows.conf（非redis.windows-service.conf）文件上的配置，从服务自启动，配置的信息是不生效的，如密码配置和ip绑定。

**解决方案：**

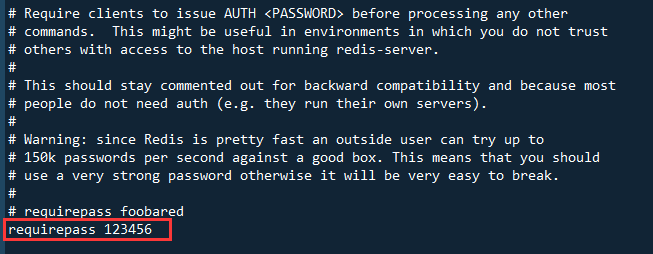
在redis.windows**-service**.conf文件（非redis.windows.conf）自定义配置信息，因为Redis安装的服务默认加载的是该文件。

然后编辑**redis.windows.conf**文件，我看网上有的教程说编辑里边**maxheap <bytes>**,但是3.x版本以后，修改的是 maxmemory **<bytes>**, bytes是字节，请自行换算，我这里改成**maxmemory 1024000000**。

其实这个地方可以不用设置，他描述上已经说了，redis会自动设置的。



设置redis的密码：找到**# requirepass foobared** 改为 **requirepass 我的密码** ，本机测试可不设置。

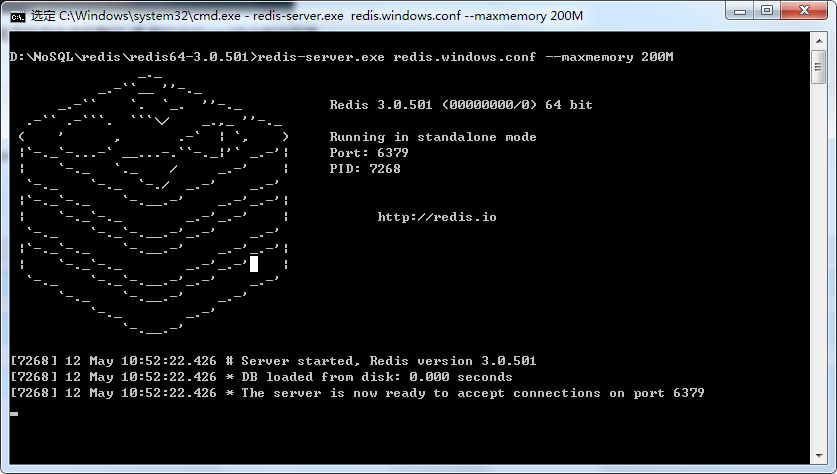


保存后打开redis的解压目录，shift+鼠标左键找到菜单上：**在此处打开命令行窗口 或** 自行运行cmd命令到此目录

命令如下：

redis-server.exe redis.windows.conf --maxmemory 200M

其中--maxmemory 200M是选填的，如果你配置文件redis.windows.conf中配置，这个就可以省去，如果没配置，这个可以加也可以不加，你开心就好。



当出现此界面的时候，说明启动成功，监听端口6379

假如出现以下错误：那么可能是你的redis有密码，如何查看redis的密码呢，进入redis目录，找到redis.windows.conf配置文件,搜索：requirepass，找到它，看看是不是有密码，这里我的密码是xwtec123

这时我们再输入auth xwtec123 回车，就可以了。

1.redis.windows.conf各项配置参数介绍

**# 默认情况下，redis不是在后台模式运行的，如果需要在后台进程运行，把该项的值更改为yes，默认为no**

**daemonize：是否以后台daemon方式运行**

**# 如redis服务以后台进程运行的时候，Redis默认会把pid写入/run/redis.pid文件组，你可以配置到其他文件路径。**

**# 当运行多个redis服务时，需要指定不同的pid文件和端口**

**pidfile：pid文件位置**

**# 指定redis监听端口，默认为6379**

**# 如果端口设置为0，Redis就不会监听TCP套接字。**

**port：监听的端口号**

**# 指定redis只接收来自于该IP地址的请求，如果不进行设置，默认将处理所有请求，**

**# 在生产环境中最好设置该项**

**bind 127.0.0.1**

**# 设置客户端连接时的超时时间，单位为秒。当客户端在这段时间内没有发出任何指令，那么关闭该连接**

**# 默认值：0代表禁用，永不关闭**

**timeout：请求超时时间**

**# 指定用来监听连接的unxi套接字的路径。这个没有默认值，所以如果不指定的话，Redis就不会通过unix套接字来监听。**

**# unixsocket /tmp/redis.sock**

**# unixsocketperm 755**

**# 指定日志记录级别**

**# Redis总共支持四个级别：debug、verbose、notice、warning，默认为verbose**

**# debug 记录很多信息，用于开发和测试**

**# varbose 很多精简的有用信息，不像debug会记录那么多**

**# notice 普通的verbose，常用于生产环境**

**# warning 只有非常重要或者严重的信息会记录到日志**

**loglevel：log信息级别**

**# 配置log文件名称和全路径地址**

**# 默认值为stdout，使用“标准输出”，默认后台模式会输出到/dev/null**

**logfile：log文件位置**

**# 可用数据库数，默认值为16，默认数据库存储在DB 0号ID库中，无特殊需求，建议仅设置一个数据库 databases 1**

**# 查询数据库使用 SELECT <dbid>**

**# dbid介于 0 到 'databases'-1 之间**

**databases：开启数据库的数量**

**save \* \*：保存快照的频率，第一个\*表示多长时间，第三个\*表示执行多少次写操作。在一定时间内执行一定数量的写操作时，自动保存快照。可设置多个条件。**

**rdbcompression：是否使用压缩**

**dbfilename：数据快照文件名（只是文件名，不包括目录）**

**dir：数据快照的保存目录（这个是目录）**

**appendonly：是否开启appendonlylog，开启的话每次写操作会记一条log，这会提高数据抗风险能力，但影响效率。**

**appendfsync：appendonlylog如何同步到磁盘（三个选项，分别是每次写都强制调用fsync、每秒启用一次fsync、不调用fsync等待系统自己同步）**

**########## REPLICATION 同步 ##########**

**#**

**# 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。**

**# 当本机为从服务时，设置主服务的IP及端口，在Redis启动时，它会自动从主服务进行数据同步**

**# slaveof <masterip> <masterport>**

**# 如果主服务master设置了密码(通过下面的 "requirepass" 选项来配置)，slave服务连接master的密码，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。**

**#当本机为从服务时，设置主服务的连接密码**

**# masterauth <master-password>**

**# 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。**

**slave-serve-stale-data yes**

**# 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秒**

**# repl-ping-slave-period 10**

**# 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。**

**# repl-timeout 60**

**########## SECURITY 安全 ##########**

**# 要求客户端在处理任何命令时都要验证身份和设置密码。**

**# 如果你不相信请求者，这个功能很有用。**

**# 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：外部使用者可以每秒尝试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。**

**# 设置连接密码**

**# requirepass foobared**

**# 命令重命名，可设置多个**

**# 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，而别人却没法知道它。**

**# 例如:**

**# rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52**

**# rename-command info info\_biran**

**# rename-command set set\_biran**

**# 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：**

**# rename-command CONFIG ""**

**########## LIMITS 限制 ##########**

**# 设置最大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)"**

**# maxclients 128**

**# 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：内存策略设置)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)**

**# 设置最大内存，达到最大内存设置后，Redis会先尝试清除已到期或即将到期的Key，当此方法处理后，任到达最大内存设置，将无法再进行写入操作。**

**# maxmemory 256000000分配256M内存**

**# maxmemory <bytes>**

**# 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-policy volatile-lru**

**# 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。**

**# maxmemory-samples 3**

**########## APPEND ONLY MODE 纯累加模式 ##########**

**# 默认情况下，Redis是异步的把数据导出到磁盘上。因为redis本身同步数据文件是按上面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，这种情况下，当Redis宕机的时候，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(此时需要把上面所有"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。**

**# 设置：yes为纯累加模式**

**appendonly no**

**# 设置纯累加文件名字及保存路径，默认："appendonly.aof"**

**# appendfilename appendonly.aof**

**# 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。**

**#**

**# 如果不确定这些模式的使用，建议使用 "everysec"**

**#**

**# appendfsync always**

**appendfsync everysec**

**# appendfsync 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。**

**no-appendfsync-on-rewrite no**

**# 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。**

**auto-aof-rewrite-percentage 100**

**auto-aof-rewrite-min-size 64mb**

**########## SLOW LOG 慢查询日志 ##########**

**# 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-log-slower-than 10000**

**# 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。**

**slowlog-max-len 128**

**########## VIRTUAL MEMORY 虚拟内存 ##########**

**### 警告！虚拟内存在Redis 2.4是反对的，因性能问题，2.4版本 VM机制彻底废弃，不建议使用此配置！！！！！！！！！！！**

**# 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-enabled no**

**# 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-swap-file /tmp/redis.swap**

**# "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，建议设置为剩余内存的60%-80%。**

**# 将所有大于vm-max-memory的数据存入虚拟内存,无论vm-max-memory设置多小,所有索引数据都是内存存储的(Redis的索引数据就是keys),也就是说,当vm-max-memory设置为0的时候,其实是所有value都存在于磁盘。默认值为0。**

**vm-max-memory 0**

**# 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-page-size 32**

**# 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-pages 134217728**

**# 同时可运行的虚拟内存I/O线程数，即访问swap文件的线程数。**

**# 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。**

**# 设置最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的.可能会造成比较长时间的延迟,但是对数据完整性有很好的保证.**

**vm-max-threads 4**

**########## ADVANCED CONFIG 高级配置 ##########**

**# 当有大量数据时，适合用哈希编码(这会需要更多的内存)，元素数量上限不能超过给定限制。**

**# Redis Hash是value内部为一个HashMap，如果该Map的成员数比较少，则会采用类似一维线性的紧凑格式来存储该Map, 即省去了大量指针的内存开销，如下2个条件任意一个条件超过设置值都会转换成真正的HashMap，**

**# 当value这个Map内部不超过多少个成员时会采用线性紧凑格式存储，默认是64,即value内部有64个以下的成员就是使用线性紧凑存储，超过该值自动转成真正的HashMap。**

**hash-max-zipmap-entries 512**

**# 当 value这个Map内部的每个成员值长度不超过多少字节就会采用线性紧凑存储来节省空间。**

**hash-max-zipmap-value 64**

**# 与hash-max-zipmap-entries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。**

**# list数据类型多少节点以下会采用去指针的紧凑存储格式**

**list-max-ziplist-entries 512**

**# list数据类型节点值大小小于多少字节会采用紧凑存储格式**

**list-max-ziplist-value 64**

**# 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。**

**set-max-intset-entries 512**

**# 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：**

**zset-max-ziplist-entries 128**

**zset-max-ziplist-value 64**

**# 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"。**

**activerehashing yes**

**########## INCLUDES 包含 ##########**

**# 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。**

**# include /path/to/local.conf**

**# include /path/to/other.conf**

修改配置后，如果配置文件涉及到中文内容记得将文件存为UTF-8编码。

2.redis-cli.exe 客户端使用

点击redis-cli.exe

测试服务器启动连接情况

127.0.0.1:6379> ping

PONG

查看服务器级别信息（测试服务器）

**127.0.0.1:6379> info**

**# Server**

**redis\_version:3.0.503**

**redis\_git\_sha1:00000000**

**redis\_git\_dirty:0**

**redis\_build\_id:d14575c6134f877**

**redis\_mode:standalone**

**os:Windows**

**arch\_bits:64**

**multiplexing\_api:WinSock\_IOCP**

**process\_id:6500**

**run\_id:f2663d82395dfda794e1a2369bca791e4297f8d3**

**tcp\_port:6379**

**uptime\_in\_seconds:13748**

**uptime\_in\_days:0**

**hz:10**

**lru\_clock:7276575**

**config\_file:**

**# Clients**

**connected\_clients:1**

**client\_longest\_output\_list:0**

**client\_biggest\_input\_buf:0**

**blocked\_clients:0**

**# Memory**

**used\_memory:692888**

**used\_memory\_human:676.65K**

**used\_memory\_rss:655240**

**used\_memory\_peak:692888**

**used\_memory\_peak\_human:676.65K**

**used\_memory\_lua:36864**

**mem\_fragmentation\_ratio:0.95**

**mem\_allocator:jemalloc-3.6.0**

**# Persistence**

**loading:0**

**rdb\_changes\_since\_last\_save:0**

**rdb\_bgsave\_in\_progress:0**

**rdb\_last\_save\_time:1500435051**

**rdb\_last\_bgsave\_status:ok**

**rdb\_last\_bgsave\_time\_sec:-1**

**rdb\_current\_bgsave\_time\_sec:-1**

**aof\_enabled:0**

**aof\_rewrite\_in\_progress:0**

**aof\_rewrite\_scheduled:0**

**aof\_last\_rewrite\_time\_sec:-1**

**aof\_current\_rewrite\_time\_sec:-1**

**aof\_last\_bgrewrite\_status:ok**

**aof\_last\_write\_status:ok**

**# Stats**

**total\_connections\_received:1**

**total\_commands\_processed:1**

**instantaneous\_ops\_per\_sec:0**

**total\_net\_input\_bytes:28**

**total\_net\_output\_bytes:7**

**instantaneous\_input\_kbps:0.00**

**instantaneous\_output\_kbps:0.00**

**rejected\_connections:0**

**sync\_full:0**

**sync\_partial\_ok:0**

**sync\_partial\_err:0**

**expired\_keys:0**

**evicted\_keys:0**

**keyspace\_hits:0**

**keyspace\_misses:0**

**pubsub\_channels:0**

**pubsub\_patterns:0**

**latest\_fork\_usec:0**

**migrate\_cached\_sockets:0**

**# Replication**

**role:master**

**connected\_slaves:0**

**master\_repl\_offset:0**

**repl\_backlog\_active:0**

**repl\_backlog\_size:1048576**

**repl\_backlog\_first\_byte\_offset:0**

**repl\_backlog\_histlen:0**

**# CPU**

**used\_cpu\_sys:0.02**

**used\_cpu\_user:0.02**

**used\_cpu\_sys\_children:0.00**

**used\_cpu\_user\_children:0.00**

**# Cluster**

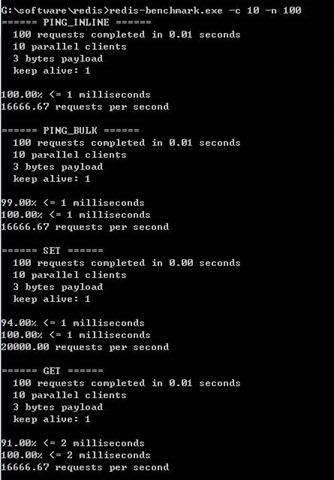
**cluster\_enabled:0**

**# Keyspace**

3.redis-benchmark 性能测试工具

默认双击打开是按照默认的测试参数进行测试，而且它自己测试跑完之后，就会自动关闭DOS窗口了。

输入如下命令后会看到如下信息，表明同时并发10个连接，总共100次操作。通俗易懂的说就是10个用户同时操作，总共每人操作10次的意思



**requests completed in 0.01 seconds （100个请求完成于0.01秒）**

**parallel clients （10个客户端并发）**

**bytes payload （每次写入3个字节）**

**keep alive: 1 （保存一个链接数）**

**100.00% <= 1 milliseconds （100%的操作小于1秒完成）**

**16666.67 requests per second （每秒完成16666.67次查询）**

命令参数说明

**redis-benchmark [-h <host>] [-p <port>] [-c <clients>] [-n <requests]> [-k <boolean>]**

**-h <hostname> 主机名 (默认 127.0.0.1)**

**-p <port> 主机端口 (默认 6379)**

**-s <socket> 主机套接字 (覆盖主机和端口)**

**-c <clients> 并发连接的数量 (默认 50)**

**-n <requests> 请求总数 (默认 10000)**

**-d <size> SET/GET数据的字节大小(默认 2)**

**-k <boolean> 1=keep alive 0=reconnect (默认 1)**

**-r <keyspacelen> SET/GET/INCR使用随机产生的key, SADD使用随机值使用这个选项 get/set keys时会用mykey\_rand:000000012456代替常量key， <keyspacelen>参数决定了随机数产生的最大值，比如，设置参数为10，那么产生的随机数范围是rand:000000000000 -rand:000000000009**

**-P <numreq> Pipeline请求的数量. 默认 1 (不使用pipeline).**

**-q 展示query/sec值**

**--csv 以CSV格式输出**

**-l 本地循环. 一直运行测试**

**-t <tests> 在运行逗号分割列表的测试. 测试的名字与产生输出的名字一样。**

**-I 空闲模式. 打开 N 个空闲连接，然后等待.**

运行示例

**对指定服务器、端口进行20个同时并发操作，总共操作100000次**

**redis-benchmark -h 192.168.1.136 -p 6379 -n 100000 -c 20**

**测试set写入操作1000000次，随机数范围在100000000**

**redis-benchmark -t set -n 1000000 -r 100000000**

**测试ping、set、get操作100000次，结果输出用csv格式**

**redis-benchmark -t ping,set,get -n 100000 –-csv**

**redis-benchmark -r 10000 -n 10000 lpush mylist ele:rand:000000000000**

4.redis-check-aof 基本用法

检查本地日志信息，加--fix参数为修复log文件

redis-check-aof.exe log.aof

5.redis-check-dump 检查数据库文件

redis-check-dump.exe dump.rdb 会输出该文件大小、使用情况。

6、 redis-check-dump 检查数据库文件

redis-check-dump.exe dump.rdb 会输出该文件大小、使用情况。

**四、影响Redis性能的因素**

**\*** 网络带宽和延迟。在执行基准测试前使用ping快速检测客户端和服务器端的延迟是一个良好的做法。对于带宽，比较好的做法是估计Gbits/s 的吞吐量和网络的理论带宽值比较。在很多实际的情况，Redis的吞吐量在网络之前会受限于CPU。

**\*** CPU也会是一个重要因素。由于单线程的，Redis受益于快速的含有巨大缓存的CPU。

**\*** 内存的速度和容量对于小的对象影响不大。但对于大于10KB的对象，可能对需要注意。通常购买昂贵的快速内存模块并不是真正的很有效。

**\*** Redis在虚拟机上运行慢。虚拟化对很多普通操作来说代价太高了，Redis并没有增加多少开销在所需的系统调用和网络中断上。

**\*** 客户端和服务器在一台机器运行，对于基准测试TCP/IP回送和UNIX域套接字都可以使用。取决于平台，但UNIX域套接字比TCP/IP回送增加50%的吞吐量。

**\*** 当大量使用 pipelining时，UNIX域套接字获得的性能好处会减少。

**\*** 当以太网访问Redis时，在数据大小小于以太网数据包的大小（大约1500字节）时，聚集命令使用 pipelining会非常有效。

**\*** 在多CPU套接字服务器，Redis的表现变得依赖于NUMA配置和处理位置。

**\*** 在高端配置，客户端连接的数量也是一个重要的因素。基于epool/kqueue模型，Redis的事件循环是相当可伸缩的。

**\*** 在高端的配置，通过调优NIC(s)配置和相关中断可能取得高吞吐量。

**\*** 根据平台，Redis编译可以使用不同的内存分配器，这可能有不同的行为在原始速度,内部和外部的碎片方面