# 1 redis 简介

## 1.1 redis简介

Redis 是完全开源免费的，遵守BSD协议，是一个高性能的key-value数据库。

Redis 与其他 key - value 缓存产品有以下三个特点：

Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

Redis支持数据的备份，即master-slave模式的数据备份。

## 1.2 Redis优势

性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。

丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

原子 – Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。

丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

## 1.3 Redis与其他key-value存储有什么不同？

Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。

Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

# 2 Redis 安装

## 2.1 Window 下安装

下载地址：https://github.com/MSOpenTech/redis/releases。

Redis 支持 32 位和 64 位。这个需要根据你系统平台的实际情况选择，这里我们下载 Redis-x64-xxx.zip压缩包到 C 盘，解压后，将文件夹重新命名为 redis。

打开一个 cmd 窗口 使用cd命令切换目录到 C:\redis 运行 redis-server.exe redis.windows.conf 。

如果想方便的话，可以把 redis 的路径加到系统的环境变量里，这样就省得再输路径了，后面的那个 redis.windows.conf 可以省略，如果省略，会启用默认的。输入之后，会显示如下界面：

这时候另启一个cmd窗口，原来的不要关闭，不然就无法访问服务端了。

切换到redis目录下运行 redis-cli.exe -h 127.0.0.1 -p 6379 。

设置键值对 set myKey abc

取出键值对 get myKey

## 2.2 Linux下安装

下载地址：http://redis.io/download，下载最新文档版本。本教程使用的最新文档版本为 2.8.17，下载并安装：

$ wget http://download.redis.io/releases/redis-2.8.17.tar.gz

$ tar xzf redis-2.8.17.tar.gz

$ cd redis-2.8.17

$ make

make完后 redis-2.8.17目录下会出现编译后的redis服务程序redis-server,还有用于测试的客户端程序redis-cli,两个程序位于安装目录 src 目录下：

下面启动redis服务.

$ cd src

$ ./redis-server

注意这种方式启动redis 使用的是默认配置。也可以通过启动参数告诉redis使用指定配置文件使用下面命令启动。

$ cd src

$ ./redis-server redis.conf

redis.conf是一个默认的配置文件。我们可以根据需要使用自己的配置文件。

启动redis服务进程后，就可以使用测试客户端程序redis-cli和redis服务交互了。 比如：

$ cd src

$ ./redis-cli

redis> set foo bar

OK

redis> get foo

"bar"

## 2.3 Ubuntu 下安装

在 Ubuntu 系统安装 Redi 可以使用以下命令:

$sudo apt-get update

$sudo apt-get install redis-server

启动 Redis

$ redis-server

查看 redis 是否启动？

$ redis-cli

以上命令将打开以下终端：

redis 127.0.0.1:6379>

其中127.0.0.1 是本机 IP ，6379 是 redis 服务端口。现在我们输入 PING 命令。

redis 127.0.0.1:6379> ping

PONG

以上说明我们已经成功安装了redis。

# 3 Redis 配置

Redis 的配置文件位于 Redis 安装目录下，文件名为 redis.conf。

通过 CONFIG GET命令查看配置项。

CONFIG GET config\_name

CONFIG GET \*

通过 CONFIG SET 命令或者修改redis.conf设置配置。

CONFIG SET config\_name new\_value

配置说明：

#redis.conf

# Redis configuration file example.

# ./redis-server /path/to/redis.conf

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

#指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件

#include /path/to/local.conf

#include /path/to/other.conf

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

#Redis默认不是以守护进程的方式运行，可以通过该配置项修改，使用yes启用守护进程

daemonize yes

#3.2里的参数，是否开启保护模式，默认开启。要是配置里没有指定bind和密码。开启该参数后，redis只会本地进行访问，拒绝外部访问。要是开启了密码和bind，可以开启。否则最好关闭，设置为no。

protected-mode yes

#当Redis以守护进程运行时，会把pid写入/var/run/redis.pid文件，可以通过pidfile指定。

pidfile /var/run/redis/redis-server.pid

#指定Redis监听端口，默认端口为6379。

port 6379

#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的默认参数值设定。该内核参数默认值一般是128，对于负载很大的服务程序来说大大的不够。一是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般会将它修改为2048或者更大。在/etc/sysctl.conf中添加:net.core.somaxconn = 2048，然后在终端中执行sysctl -p。

tcp-backlog 511

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

bind 127.0.0.1

#配置unix socket来让redis支持监听本地连接。

# unixsocket /var/run/redis/redis.sock

#配置unix socket使用文件的权限

# unixsocketperm 700

# 此参数为设置客户端空闲超过timeout，服务端会断开连接，为0则服务端不会主动断开连接，不能小于0。

timeout 0

#tcp keepalive参数。如果设置不为0，就使用配置tcp的SO\_KEEPALIVE值，使用keepalive有两个好处:检测挂掉的对端。降低中间设备出问题而导致网络看似连接却已经与对端端口的问题。在Linux内核中，设置了keepalive，redis会定时给对端发送ack。检测到对端关闭需要两倍的设置值。

tcp-keepalive 0

#指定了服务端日志的级别。级别包括：debug（很多信息），verbose（许多有用），notice（适当日志级别），warn（只有非常重要的信息）

loglevel notice

#日志记录方式，默认为标准输出，如果配置Redis为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给/dev/null。

logfile /var/log/redis/redis-server.log

#是否打开记录syslog功能

# syslog-enabled no

#syslog的标识符。

# syslog-ident redis

#日志的来源、设备

# syslog-facility local0

# 设置数据库的数量，默认数据库为0，可以使用SELECT <dbid>命令在连接上指定数据库id

databases 16

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

# 快照配置

# 注释掉“save”这一行配置项就可以让保存数据库功能失效

# 设置sedis进行数据库镜像的频率。

# 900秒（15分钟）内至少1个key值改变（则进行数据库保存--持久化）

# 300秒（5分钟）内至少10个key值改变（则进行数据库保存--持久化）

# 60秒（1分钟）内至少10000个key值改变（则进行数据库保存--持久化）

save 900 1

save 300 10

save 60 10000

#当RDB持久化出现错误后，是否依然进行继续进行工作，yes：不能进行工作，no：可以继续进行工作，可以通过info中的rdb\_last\_bgsave\_status了解RDB持久化是否有错误

stop-writes-on-bgsave-error yes

#指定存储至本地数据库时是否压缩数据，默认为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变的巨大

rdbcompression yes

#是否校验rdb文件。从rdb格式的第五个版本开始，在rdb文件的末尾会带上CRC64的校验和。这跟有利于文件的容错性，但是在保存rdb文件的时候，会有大概10%的性能损耗，所以如果你追求高性能，可以关闭该配置。

rdbchecksum yes

#指定本地数据库文件名，默认值为dump.rdb

dbfilename dump.rdb

#指定本地数据库存放目录数据目录，数据库的写入会在这个目录。rdb、aof文件也会写在这个目录

dir /var/lib/redis

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

#复制选项，slave复制对应的master。设置当本机为slave服务时，设置master服务的IP地址及端口，在Redis启动时，它会自动从master进行数据同步

# slaveof <masterip> <masterport>

#如果master设置了requirepass，那么slave要连上master，需要有master的密码才行。masterauth就是用来配置master的密码，这样可以在连上master后进行认证。

# masterauth <master-password>

#当从库同主机失去连接或者同步正在进行，从机库有两种运行方式：1) 如果slave-serve-stale-data设置为yes(默认设置)，从库会继续响应客户端的请求。2) 如果slave-serve-stale-data设置为no，除去INFO和SLAVOF命令之外的任何请求都会返回一个错误”SYNC with master in progress”。

slave-serve-stale-data yes

#作为从服务器，默认情况下是只读的（yes），可以修改成NO，用于写（不建议）。

slave-read-only yes

#是否使用socket方式复制数据。目前redis复制提供两种方式，disk和socket。如果新的slave连上来或者重连的slave无法部分同步，就会执行全量同步，master会生成rdb文件。有2种方式：disk方式是master创建一个新的进程把rdb文件保存到磁盘，再把磁盘上的rdb文件传递给slave。socket是master创建一个新的进程，直接把rdb文件以socket的方式发给slave。disk方式的时候，当一个rdb保存的过程中，多个slave都能共享这个rdb文件。socket的方式就的一个个slave顺序复制。在磁盘速度缓慢，网速快的情况下推荐用socket方式。

repl-diskless-sync no

#diskless复制的延迟时间，防止设置为0。因为一旦复制开始，节点不会再接收新slave的复制请求直到备份完成。所以最好等待一段时间，等更多的slave连上来。

repl-diskless-sync-delay 5

#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

#复制连接超时时间。master和slave都有超时时间的设置。master检测到slave上次发送的时间超过repl-timeout，即认为slave离线，清除该slave信息。slave检测到上次和master交互的时间超过repl-timeout，则认为master离线。需要注意的是repl-timeout需要设置一个比repl-ping-slave-period更大的值，不然会经常检测到超时。

# repl-timeout 60

#是否禁止复制tcp链接的tcp nodelay参数，可传递yes或者no。默认是no，即使用tcp nodelay。如果master设置了yes来禁止tcp nodelay设置，在把数据复制给slave的时候，会减少包的数量和更小的网络带宽。但是这也可能带来数据的延迟。默认我们推荐更小的延迟，但是在数据量传输很大的场景下，建议选择yes。

repl-disable-tcp-nodelay no

#复制缓冲区大小，这是一个环形复制缓冲区，用来保存最新复制的命令。这样在slave离线的时候，不需要完全复制master的数据，如果可以执行部分同步，只需要把缓冲区的部分数据复制给slave，就能恢复正常复制状态。缓冲区的大小越大，slave离线的时间可以更长，复制缓冲区只有在有slave连接的时候才分配内存。没有slave的一段时间，内存会被释放出来，默认1m。

# repl-backlog-size 5mb

#master没有slave一段时间会释放复制缓冲区的内存，repl-backlog-ttl用来设置该时间长度。单位为秒。

# repl-backlog-ttl 3600

#当master不可用，Sentinel会根据slave的优先级选举一个master。最低的优先级的slave，当选master。而配置成0，永远不会被选举。

slave-priority 100

#redis提供了可以让master停止写入的方式，如果配置了min-slaves-to-write，健康的slave的个数小于N，mater就禁止写入。master最少得有多少个健康的slave存活才能执行写命令。这个配置虽然不能保证N个slave都一定能接收到master的写操作，但是能避免没有足够健康的slave的时候，master不能写入来避免数据丢失。设置为0是关闭该功能。默认为0

# min-slaves-to-write 3

#延迟小于min-slaves-max-lag秒的slave才认为是健康的slave。吗，默认为10

# min-slaves-max-lag 10

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

# 设置Redis连接密码，如果配置了连接密码，客户端在连接Redis时需要通过AUTH <password>命令提供密码，这让redis可以使用在不受信任的网络中。为了保持向后的兼容性，可以注释该命令，因为大部分用户也不需要认证。使用requirepass的时候需要注意，因为redis太快了，每秒可以认证15w次密码，简单的密码很容易被攻破，所以最好使用一个更复杂的密码。默认关闭。

# requirepass foobared

#把危险的命令给修改成其他名称。比如CONFIG命令可以重命名为一个很难被猜到的命令，这样用户不能使用，而内部工具还能接着使用。

# rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52

#设置成一个空的值，可以禁止一个命令

# rename-command CONFIG ""

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

# 设置能连上redis的最大客户端连接数量。默认是10000个客户端连接。由于redis不区分连接是客户端连接还是内部打开文件或者和slave连接等，所以maxclients最小建议设置到32。如果超过了maxclients，redis会给新的连接发送’max number of clients reached’，并关闭连接。

# maxclients 10000

#redis配置的最大内存容量。当内存满了，需要配合maxmemory-policy策略进行处理。注意slave的输出缓冲区是不计算在maxmemory内的。所以为了防止主机内存使用完，建议设置的maxmemory需要更小一些。

# maxmemory <bytes>

#内存容量超过maxmemory后的处理策略。

#volatile-lru：利用LRU算法移除设置过过期时间的key。

#volatile-random：随机移除设置过过期时间的key。

#volatile-ttl：移除即将过期的key，根据最近过期时间来删除（辅以TTL）

#allkeys-lru：利用LRU算法移除任何key。

#allkeys-random：随机移除任何key。

#noeviction：不移除任何key，只是返回一个写错误。

#上面的这些驱逐策略，如果redis没有合适的key驱逐，对于写命令，还是会返回错误。redis将不再接收写请求，只接收get请求。写命令包括：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。

# maxmemory-policy noeviction

#lru检测的样本数。使用lru或者ttl淘汰算法，从需要淘汰的列表中随机选择sample个key，选出闲置时间最长的key移除。

# maxmemory-samples 5

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

#默认redis使用的是上面的save策略持久化，这种方式在许多应用中已经足够用了。但是redis如果中途宕机，会导致可能有几分钟的数据丢失，Append Only File是另一种持久化方式，可以提供更好的持久化特性。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都会先把这个文件的数据读入内存里，先忽略RDB文件。默认为no。

appendonly no

#指定更新日志文件名，默认为appendonly.aof

appendfilename "appendonly.aof"

#aof持久化策略的配置

#no表示不执行fsync，由操作系统保证数据同步到磁盘，速度最快。

#always表示每次写入都执行fsync，以保证数据同步到磁盘。

#everysec表示每秒执行一次fsync，可能会导致丢失这1s数据。

appendfsync everysec

# 在aof重写或者写入rdb文件的时候，会执行大量IO，此时对于everysec和always的aof模式来说，执行fsync会造成阻塞过长时间，no-appendfsync-on-rewrite字段设置为默认设置为no。如果对延迟要求很高的应用，这个字段可以设置为yes，否则还是设置为no，这样对持久化特性来说这是更安全的选择。设置为yes表示rewrite期间对新写操作不fsync,暂时存在内存中,等rewrite完成后再写入，默认为no，建议yes。Linux的默认fsync策略是30秒。可能丢失30秒数据。

no-appendfsync-on-rewrite no

#aof自动重写配置。当目前aof文件大小超过上一次重写的aof文件大小的百分之多少进行重写，即当aof文件增长到一定大小的时候Redis能够调用bgrewriteaof对日志文件进行重写。当前AOF文件大小是上次日志重写得到AOF文件大小的二倍（设置为100）时，自动启动新的日志重写过程。

auto-aof-rewrite-percentage 100

#设置允许重写的最小aof文件大小，避免了达到约定百分比但尺寸仍然很小的情况还要重写

auto-aof-rewrite-min-size 64mb

#aof文件可能在尾部是不完整的，当redis启动的时候，aof文件的数据被载入内存。重启可能发生在redis所在的主机操作系统宕机后，尤其在ext4文件系统没有加上data=ordered选项（redis宕机或者异常终止不会造成尾部不完整现象。）出现这种现象，可以选择让redis退出，或者导入尽可能多的数据。如果选择的是yes，当截断的aof文件被导入的时候，会自动发布一个log给客户端然后load。如果是no，用户必须手动redis-check-aof修复AOF文件才可以。

aof-load-truncated yes

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

# 如果达到最大时间限制（毫秒），redis会记个log，然后返回error。当一个脚本超过了最大时限。只有SCRIPT KILL和SHUTDOWN NOSAVE可以用。第一个可以杀没有调write命令的东西。要是已经调用了write，只能用第二个命令杀。

lua-time-limit 5000

#在aof重写的时候，如果打开了aof-rewrite-incremental-fsync开关，系统会每32MB执行一次fsync。这对于把文件写入磁盘是有帮助的，可以避免过大的延迟峰值。

aof-rewrite-incremental-fsync yes

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

#集群开关，默认是不开启集群模式。

# cluster-enabled yes

#集群配置文件的名称，每个节点都有一个集群相关的配置文件，持久化保存集群的信息。这个文件并不需要手动配置，这个配置文件有Redis生成并更新，每个Redis集群节点需要一个单独的配置文件，请确保与实例运行的系统中配置文件名称不冲突

# cluster-config-file nodes-6379.conf

#节点互连超时的阀值。集群节点超时毫秒数

# cluster-node-timeout 15000

#在进行故障转移的时候，全部slave都会请求申请为master，但是有些slave可能与master断开连接一段时间了，导致数据过于陈旧，这样的slave不应该被提升为master。该参数就是用来判断slave节点与master断线的时间是否过长。判断方法是：

#比较slave断开连接的时间和(cluster-node-timeout \* cluster-slave-validity-factor) + repl-ping-slave-period

#如果节点超时时间为三十秒, 并且cluster-slave-validity-factor为10,假设默认的repl-ping-slave-period是10秒，即如果超过310秒slave将不会尝试进行故障转移

# cluster-slave-validity-factor 10

#master的slave数量大于该值，slave才能迁移到其他孤立master上，如这个参数若被设为2，那么只有当一个主节点拥有2 个可工作的从节点时，它的一个从节点会尝试迁移。

# cluster-migration-barrier 1

#默认情况下，集群全部的slot有节点负责，集群状态才为ok，才能提供服务。设置为no，可以在slot没有全部分配的时候提供服务。不建议打开该配置，这样会造成分区的时候，小分区的master一直在接受写请求，而造成很长时间数据不一致。

# cluster-require-full-coverage yes

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

#slowlog是用来记录redis运行中执行比较慢的命令耗时。当命令的执行超过了指定时间，就记录在slow log中，slog log保存在内存中，所以没有IO操作。

#执行时间比slowlog-log-slower-than大的请求记录到slowlog里面，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

#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。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

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

#延迟监控功能是用来监控redis中执行比较缓慢的一些操作，用LATENCY打印redis实例在跑命令时的耗时图表。只记录大于等于下边设置的值的操作。0的话，就是关闭监视。默认延迟监控功能是关闭的，如果你需要打开，也可以通过CONFIG SET命令动态设置。

latency-monitor-threshold 0

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

#键空间通知使得客户端可以通过订阅频道或模式，来接收那些以某种方式改动了 Redis 数据集的事件。因为开启键空间通知功能需要消耗一些 CPU ，所以在默认配置下，该功能处于关闭状态。

#notify-keyspace-events 的参数可以是以下字符的任意组合，它指定了服务器该发送哪些类型的通知：

##K 键空间通知，所有通知以 \_\_keyspace@\_\_ 为前缀

##E 键事件通知，所有通知以 \_\_keyevent@\_\_ 为前缀

##g DEL 、 EXPIRE 、 RENAME 等类型无关的通用命令的通知

##$ 字符串命令的通知

##l 列表命令的通知

##s 集合命令的通知

##h 哈希命令的通知

##z 有序集合命令的通知

##x 过期事件：每当有过期键被删除时发送

##e 驱逐(evict)事件：每当有键因为 maxmemory 政策而被删除时发送

##A 参数 g$lshzxe 的别名

#输入的参数中至少要有一个 K 或者 E，否则的话，不管其余的参数是什么，都不会有任何 通知被分发。详细使用可以参考http://redis.io/topics/notifications

notify-keyspace-events ""

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

#数据量小于等于hash-max-ziplist-entries的用ziplist，大于hash-max-ziplist-entries用hash

hash-max-ziplist-entries 512

#value大小小于等于hash-max-ziplist-value的用ziplist，大于hash-max-ziplist-value用hash。

hash-max-ziplist-value 64

#数据量小于等于list-max-ziplist-entries用ziplist，大于list-max-ziplist-entries用list。

list-max-ziplist-entries 512

#value大小小于等于list-max-ziplist-value的用ziplist，大于list-max-ziplist-value用list。

list-max-ziplist-value 64

#数据量小于等于set-max-intset-entries用iniset，大于set-max-intset-entries用set。

set-max-intset-entries 512

#数据量小于等于zset-max-ziplist-entries用ziplist，大于zset-max-ziplist-entries用zset。

zset-max-ziplist-entries 128

#value大小小于等于zset-max-ziplist-value用ziplist，大于zset-max-ziplist-value用zset。

zset-max-ziplist-value 64

#value大小小于等于hll-sparse-max-bytes使用稀疏数据结构（sparse），大于hll-sparse-max-bytes使用稠密的数据结构（dense）。一个比16000大的value是几乎没用的，建议的value大概为3000。如果对CPU要求不高，对空间要求较高的，建议设置到10000左右。

hll-sparse-max-bytes 3000

#Redis将在每100毫秒时使用1毫秒的CPU时间来对redis的hash表进行重新hash，可以降低内存的使用。当你的使用场景中，有非常严格的实时性需要，不能够接受Redis时不时的对请求有2毫秒的延迟的话，把这项配置为no。如果没有这么严格的实时性要求，可以设置为yes，以便能够尽可能快的释放内存。

activerehashing yes

#对客户端输出缓冲进行限制可以强迫那些不从服务器读取数据的客户端断开连接，用来强制关闭传输缓慢的客户端。

#对于normal client，第一个0表示取消hard limit，第二个0和第三个0表示取消soft limit，normal client默认取消限制，因为如果没有寻问，他们是不会接收数据的。

client-output-buffer-limit normal 0 0 0

#对于slave client和MONITER client，如果client-output-buffer一旦超过256mb，又或者超过64mb持续60秒，那么服务器就会立即断开客户端连接。

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

#对于pubsub client，如果client-output-buffer一旦超过32mb，又或者超过8mb持续60秒，那么服务器就会立即断开客户端连接。

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

#redis执行任务的频率为1s除以hz。

hz 10

###########################virtual memory################################

#指定是否启用虚拟内存机制，默认值为no，VM机制将数据分页存放，由Redis将访问量较少的页即冷数据swap到磁盘上，访问多的页面由磁盘自动换出到内存中

vm-enabled no

#虚拟内存文件路径，默认值为/tmp/redis.swap，不可多个Redis实例共享

vm-swap-file /tmp/redis.swap

#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 swap文件分成了很多的page，一个对象可以保存在多个page上面，但一个page上不能被多个对象共享，vm-page-size是要根据存储的 数据大小来设定的，建议如果存储很多小对象，page大小最好设置为32或者64bytes；如果存储很大大对象，则可以使用更大的page，如果不 确定，就使用默认值

vm-page-size 32

#设置swap文件中的page数量，由于页表（一种表示页面空闲或使用的bitmap）是在放在内存中的，，在磁盘上每8个pages将消耗1byte的内存。

vm-pages 134217728

#设置访问swap文件的线程数,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。默认值为4

vm-max-threads 4

#设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启

glueoutputbuf yes

# 4 Redis 数据类型

Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zset(sorted set：有序集合)。

## 4.1 String（字符串）

string是redis最基本的类型，你可以理解成与Memcached一模一样的类型，一个key对应一个value。

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个键最大能存储512MB。

## 4.2 Hash（哈希）

Redis hash 是一个键名对集合。类似于C++的map。

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

每个 hash 可以存储 232 -1 键值对（40多亿）。

## 4.3 List（列表）

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）。

列表最多可存储 232 - 1 元素 (4294967295, 每个列表可存储40多亿)。

## 4.4 Set（集合）

Redis的Set是string类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

注意：根据集合内元素的唯一性，重复插入的元素将被忽略。

集合中最大的成员数为 232 - 1(4294967295, 每个集合可存储40多亿个成员)。

## 4.5 zset(有序集合)

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。

# 5 连接redis 服务器

## 5.1 本地连接

语法

$ redis-cli

## 5.2 远程连接

如果需要在远程 redis 服务上执行命令，同样我们使用的也是 redis-cli 命令。语法：

$ redis-cli -h host -p port -a password

# 6 Redis 键命令

Redis 键命令用于管理 redis 的键。

Redis 键命令的基本语法如下：

COMMAND KEY\_NAME

|  |  |
| --- | --- |
| 命令 | 功能 |
| DEL key | 该命令用于在 key 存在时删除 key。 |
| DUMP key | 序列化给定 key ，并返回被序列化的值。 |
| EXISTS key | 检查给定 key 是否存在。 |
| EXPIRE key seconds | 为给定 key 设置过期时间。 |
| EXPIREAT key timestamp | EXPIREAT 的作用和 EXPIRE 类似，都用于为 key 设置过期时间。 不同在于 EXPIREAT 命令接受的时间参数是 UNIX 时间戳(unix timestamp)。 |
| PEXPIRE key milliseconds | 设置 key 的过期时间以毫秒计。 |
| PEXPIREAT key milliseconds-timestamp | 设置 key 过期时间的时间戳(unix timestamp) 以毫秒计 |
| KEYS pattern | 查找所有符合给定模式( pattern)的 key 。 |
| MOVE key db | 将当前数据库的 key 移动到给定的数据库 db 当中。 |
| PERSIST key | 移除 key 的过期时间，key 将持久保持。 |
| PTTL key | 以毫秒为单位返回 key 的剩余的过期时间。 |
| TTL key | 以秒为单位，返回给定 key 的剩余生存时间(TTL, time to live)。 |
| RANDOMKEY | 从当前数据库中随机返回一个 key 。 |
| RENAME key newkey | 修改 key 的名称 |
| RENAMENX key newkey | 仅当 newkey 不存在时，将 key 改名为 newkey 。 |
| TYPE key | 返回 key 所储存的值的类型。 |

# 7 Redis 基本命令

## 7.1 字符串(String)

Redis 字符串数据类型的相关命令用于管理 redis 字符串值，基本语法如下：

COMMAND KEY\_NAME

|  |  |
| --- | --- |
| 命令 | 功能 |
| SET key value | 设置指定 key 的值 |
| GET key | 获取指定 key 的值。 |
| GETRANGE key start end | 返回 key 中字符串值的子字符 |
| GETSET key value | 将给定 key 的值设为 value ，并返回 key 的旧值(old value)。 |
| GETBIT key offset | 对 key 所储存的字符串值，获取指定偏移量上的位(bit)。 |
| MGET key1 [key2..] | 获取所有(一个或多个)给定 key 的值。 |
| SETBIT key offset value | 对 key 所储存的字符串值，设置或清除指定偏移量上的位(bit)。 |
| SETEX key seconds value | 将值value关联到key,并将key的过期时间设为 seconds (以秒为单位)。 |
| SETNX key value | 只有在 key 不存在时设置 key 的值。 |
| SETRANGE key offset value | 用 value 参数覆写给定 key 所储存的字符串值，从偏移量 offset 开始。 |
| STRLEN key | 返回 key 所储存的字符串值的长度。 |
| MSET key value [key value ...] | 同时设置一个或多个 key-value 对。 |
| MSETNX key value [key value ...] | 同时设置一个或多个 key-value 对，当且仅当所有给定 key 都不存在。 |
| PSETEX key milliseconds value | 这个命令和 SETEX 命令相似，但它以毫秒为单位设置 key 的生存时间，而不是像 SETEX 命令那样，以秒为单位。 |
| INCR key | 将 key 中储存的数字值增一。 |
| INCRBY key increment | 将 key 所储存的值加上给定的增量值（increment） 。 |
| INCRBYFLOAT key increment | 将 key 所储存的值加上给定的浮点增量值（increment） 。 |
| DECR key | 将 key 中储存的数字值减一。 |
| DECRBY key decrement | key 所储存的值减去给定的减量值（decrement） 。 |
| APPEND key value | 如果 key 已经存在并且是一个字符串， APPEND 命令将 value 追加到 key 原来的值的末尾。 |

## 7.2 哈希(Hash)

Redis hash 是一个string类型的field和value的映射表，hash特别适合用于存储对象。

Redis 中每个 hash 可以存储 232 - 1 键值对（40多亿）。

|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| HDEL key field2 [field2] | 删除一个或多个哈希表字段 |
| HEXISTS key field | 查看哈希表 key 中，指定的字段是否存在。 |
| HGET key field | 获取存储在哈希表中指定字段的值。 |
| HGETALL key | 获取在哈希表中指定 key 的所有字段和值 |
| HINCRBY key field increment | 为哈希表 key 中的指定字段的整数值加上增量 increment 。 |
| HINCRBYFLOAT key field increment | 为哈希表 key 中的指定字段的浮点数值加上增量 increment 。 |
| HKEYS key | 获取所有哈希表中的字段 |
| HLEN key | 获取哈希表中字段的数量 |
| HMGET key field1 [field2] | 获取所有给定字段的值 |
| HMSET key field1 value1 [field2 value2 ] | 同时将多个 field-value (域-值)对设置到哈希表 key 中。 |
| HSET key field value | 将哈希表 key 中的字段 field 的值设为 value 。 |
| HSETNX key field value | 只有在字段 field 不存在时，设置哈希表字段的值。 |
| HVALS key | 获取哈希表中所有值 |
| HSCAN key cursor [MATCH pattern] [COUNT count] | 迭代哈希表中的键值对。 |

## 7.3 列表(List)

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

一个列表最多可以包含 232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。

|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| BLPOP key1 [key2 ] timeout | 移出并获取列表的第一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| BRPOP key1 [key2 ] timeout | 移出并获取列表的最后一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| BRPOPLPUSH source destination timeout | 从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| LINDEX key index | 通过索引获取列表中的元素 |
| LINSERT key BEFORE|AFTER pivot value | 在列表的元素前或者后插入元素 |
| LLEN key | 获取列表长度 |
| LPOP key | 移出并获取列表的第一个元素 |
| LPUSH key value1 [value2] | 将一个或多个值插入到列表头部 |
| LPUSHX key value | 将一个值插入到已存在的列表头部 |
| LRANGE key start stop | 获取列表指定范围内的元素 |
| LREM key count value | 移除列表元素 |
| LSET key index value | 通过索引设置列表元素的值 |
| LTRIM key start stop | 对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。 |
| RPOP key | 移除并获取列表最后一个元素 |
| RPOPLPUSH source destination | 移除列表的最后一个元素，并将该元素添加到另一个列表并返回 |
| RPUSH key value1 [value2] | 在列表中添加一个或多个值 |
| RPUSHX key value | 为已存在的列表添加值 |

## 7.4 集合(Set)

Redis的Set是string类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。

Redis 中 集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| SADD key member1 [member2] | 向集合添加一个或多个成员 |
| SCARD key | 获取集合的成员数 |
| SDIFF key1 [key2] | 返回给定所有集合的差集 |
| SDIFFSTORE destination key1 [key2] | 返回给定所有集合的差集并存储在 destination 中 |
| SINTER key1 [key2] | 返回给定所有集合的交集 |
| SINTERSTORE destination key1 [key2] | 返回给定所有集合的交集并存储在 destination 中 |
| SISMEMBER key member | 判断 member 元素是否是集合 key 的成员 |
| SMEMBERS key | 返回集合中的所有成员 |
| SMOVE source destination member | 将 member 元素从 source 集合移动到 destination 集合 |
| SPOP key | 移除并返回集合中的一个随机元素 |
| SRANDMEMBER key [count] | 返回集合中一个或多个随机数 |
| SREM key member1 [member2] | 移除集合中一个或多个成员 |
| SUNION key1 [key2] | 返回所有给定集合的并集 |
| SUNIONSTORE destination key1 [key2] | 所有给定集合的并集存储在 destination 集合中 |
| SSCAN key cursor [MATCH pattern] [COUNT count] | 迭代集合中的元素 |

## 7.5 有序集合(sorted set)

Redis 有序集合和集合一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。有序集合的成员是唯一的,但分数(score)却可以重复。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。 集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| ZADD key score1 member1 [score2 member2] | 向有序集合添加一个或多个成员，或者更新已存在成员的分数 |
| ZCARD key | 获取有序集合的成员数 |
| ZCOUNT key min max | 计算在有序集合中指定区间分数的成员数 |
| ZINCRBY key increment member | 有序集合中对指定成员的分数加上增量 increment |
| ZINTERSTORE destination numkeys key [key ...] | 计算给定的一个或多个有序集的交集并将结果集存储在新的有序集合 key 中 |
| ZLEXCOUNT key min max | 在有序集合中计算指定字典区间内成员数量 |
| ZRANGE key start stop [WITHSCORES] | 通过索引区间返回有序集合成指定区间内的成员 |
| ZRANGEBYLEX key min max [LIMIT offset count] | 通过字典区间返回有序集合的成员 |
| ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT] | 通过分数返回有序集合指定区间内的成员 |
| ZRANK key member | 返回有序集合中指定成员的索引 |
| ZREM key member [member ...] | 移除有序集合中的一个或多个成员 |
| ZREMRANGEBYLEX key min max | 移除有序集合中给定的字典区间的所有成员 |
| ZREMRANGEBYRANK key start stop | 移除有序集合中给定的排名区间的所有成员 |
| ZREMRANGEBYSCORE key min max | 移除有序集合中给定的分数区间的所有成员 |
| ZREVRANGE key start stop [WITHSCORES] | 返回有序集中指定区间内的成员，通过索引，分数从高到底 |
| ZREVRANGEBYSCORE key max min [WITHSCORES] | 返回有序集中指定分数区间内的成员，分数从高到低排序 |
| ZREVRANK key member | 返回有序集合中指定成员的排名，有序集成员按分数值递减(从大到小)排序 |
| ZSCORE key member | 返回有序集中，成员的分数值 |
| ZUNIONSTORE destination numkeys key [key ...] | 计算给定的一个或多个有序集的并集，并存储在新的 key 中 |
| ZSCAN key cursor [MATCH pattern] [COUNT count] | 迭代有序集合中的元素（包括元素成员和元素分值） |

## 7.6 HyperLogLog

Redis 在 2.8.9 版本添加了 HyperLogLog 结构。

Redis HyperLogLog 是用来做基数统计的算法，HyperLogLog 的优点是，在输入元素的数量或者体积非常非常大时，计算基数所需的空间总是固定的、并且是很小的。

在 Redis 里面，每个 HyperLogLog 键只需要花费 12 KB 内存，就可以计算接近 264 个不同元素的基数。这和计算基数时，元素越多耗费内存就越多的集合形成鲜明对比。

但是，因为 HyperLogLog 只会根据输入元素来计算基数，而不会储存输入元素本身，所以 HyperLogLog 不能像集合那样，返回输入的各个元素。

什么是基数?

比如数据集 {1, 3, 5, 7, 5, 7, 8}， 那么这个数据集的基数集为 {1, 3, 5 ,7, 8}, 基数(不重复元素)为5。 基数估计就是在误差可接受的范围内，快速计算基数。

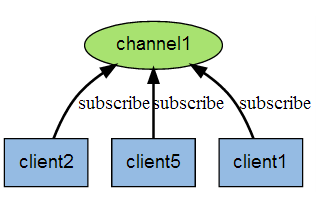
|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| PFADD key element [element ...] | 添加指定元素到 HyperLogLog 中。 |
| PFCOUNT key [key ...] | 返回给定 HyperLogLog 的基数估算值。 |
| PFMERGE destkey sourcekey [sourcekey ...] | 将多个 HyperLogLog 合并为一个 HyperLogLog |

## 7.7 发布订阅（pub/sub）

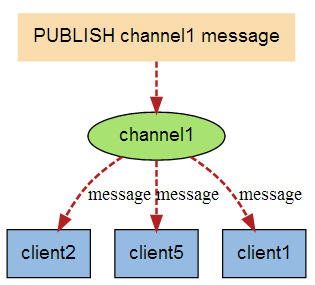
Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

Redis 客户端可以订阅任意数量的频道。

下图展示了频道 channel1，以及订阅这个频道的三个客户端 client2 、 client5 和 client1 之间的关系：



当有新消息通过 PUBLISH 命令发送给频道 channel1 时，这个消息就会被发送给订阅它的三个客户端：



|  |  |  |
| --- | --- | --- |
| 序号 | 命令及描述 | |
| PSUBSCRIBE pattern [pattern ...] | 订阅一个或多个符合给定模式的频道。 | |
| PUBSUB subcommand [argument [argument ...]] | | 查看订阅与发布系统状态。 |
| PUBLISH channel message | 将信息发送到指定的频道。 | |
| PUNSUBSCRIBE [pattern [pattern ...]] | 退订所有给定模式的频道。 | |
| SUBSCRIBE channel [channel ...] | 订阅给定的一个或多个频道的信息。 | |
| UNSUBSCRIBE [channel [channel ...]] | 指退订给定的频道。 | |

## 7.8 事物

Redis 事务可以一次执行多个命令， 并且带有以下两个重要的保证：

事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

一个事务从开始到执行会经历以下三个阶段：开始事务；命令入队；执行事务。

|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| DISCARD | 取消事务，放弃执行事务块内的所有命令。 |
| EXEC | 执行所有事务块内的命令。 |
| MULTI | 标记一个事务块的开始。 |
| UNWATCH | 取消 WATCH 命令对所有 key 的监视。 |
| WATCH key [key ...] | 监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。 |

## 7.9 脚本

Redis 脚本使用 Lua 解释器来执行脚本。 Reids 2.6 版本通过内嵌支持 Lua 环境。执行脚本的常用命令为 EVAL。Eval 命令的基本语法如下：

EVAL script numkeys key [key ...] arg [arg ...]

|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| EVAL script numkeys key [key ...] arg [arg ...] | 执行 Lua 脚本。 |
| EVALSHA sha1 numkeys key [key ...] arg [arg ...] | 执行 Lua 脚本。 |
| SCRIPT EXISTS script [script ...] | 查看指定的脚本是否已经被保存在缓存当中。 |
| SCRIPT FLUSH | 从脚本缓存中移除所有脚本。 |
| SCRIPT KILL | 杀死当前正在运行的 Lua 脚本。 |
| SCRIPT LOAD script | 将脚本script添加到脚本缓存中但并不立即执行这个脚本。 |

## 7.10 连接

Redis 连接命令主要是用于连接 redis 服务。

|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| AUTH password | 验证密码是否正确 |
| ECHO message | 打印字符串 |
| PING | 查看服务是否运行 |
| QUIT | 关闭当前连接 |
| SELECT index | 切换到指定的数据库 |

## 7.11 服务器

Redis 服务器命令主要是用于管理 redis 服务。

|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| BGREWRITEAOF | 异步执行一个 AOF（AppendOnly File） 文件重写操作 |
| BGSAVE | 在后台异步保存当前数据库的数据到磁盘 |
| CLIENT KILL [ip:port] [ID client-id] | 关闭客户端连接 |
| CLIENT LIST | 获取连接到服务器的客户端连接列表 |
| CLIENT GETNAME | 获取连接的名称 |
| CLIENT PAUSE timeout | 在指定时间内终止运行来自客户端的命令 |
| CLIENT SETNAME connection-name | 设置当前连接的名称 |
| CLUSTER SLOTS | 获取集群节点的映射数组 |
| COMMAND | 获取 Redis 命令详情数组 |
| COMMAND COUNT | 获取 Redis 命令总数 |
| COMMAND GETKEYS | 获取给定命令的所有键 |
| TIME | 返回当前服务器时间 |
| COMMAND INFO command-name [command-name ...] | 获取指定 Redis 命令描述的数组 |
| CONFIG GET parameter | 获取指定配置参数的值 |
| CONFIG REWRITE | 对启动 Redis 服务器时所指定的 redis.conf 配置文件进行改写 |
| CONFIG SET parameter value | 修改 redis 配置参数，无需重启 |
| CONFIG RESETSTAT | 重置 INFO 命令中的某些统计数据 |
| DBSIZE | 返回当前数据库的 key 的数量 |
| DEBUG OBJECT key | 获取 key 的调试信息 |
| DEBUG SEGFAULT | 让 Redis 服务崩溃 |
| FLUSHALL | 删除所有数据库的所有key |
| FLUSHDB | 删除当前数据库的所有key |
| INFO [section] | 获取 Redis 服务器的各种信息和统计数值 |
| LASTSAVE | 返回最近一次 Redis 成功将数据保存到磁盘上的时间，以 UNIX 时间戳格式表示 |
| MONITOR | 实时打印出 Redis 服务器接收到的命令，调试用 |
| ROLE | 返回主从实例所属的角色 |
| SAVE | 异步保存数据到硬盘 |
| SHUTDOWN [NOSAVE] [SAVE] | 异步保存数据到硬盘，并关闭服务器 |
| SLAVEOF host port | 将当前服务器转变为指定服务器的从属服务器(slave server) |
| SLOWLOG subcommand [argument] | 管理 redis 的慢日志 |
| SYNC | 用于复制功能(replication)的内部命令 |

# 8 Redis 进阶命令

## 8.1 数据备份与恢复

SAVE 命令用于创建当前数据库的备份，创建 redis 备份文件也可以使用命令 BGSAVE，该命令在后台执行。

如果需要恢复数据，只需将备份文件 (dump.rdb) 移动到 redis 安装目录并启动服务即可。获取 redis 目录可以使用 CONFIG GETdir命令。

## 8.2 安全

我们可以通过 redis 的配置文件设置密码参数，这样客户端连接到 redis 服务就需要密码验证，这样可以让你的 redis 服务更安全。

以通过CONFIG GET requirepass查看是否设置了密码验证。

默认情况下 requirepass 参数是空的，这就意味着你无需通过密码验证就可以连接到 redis 服务。

可以通过CONFIG set requirepass “newpassword”设置密码。设置密码后，客户端连接 redis 服务就需要密码验证，否则无法执行命令。

密码验证基本语法格式如下：AUTH password。

## 8.3 性能测试

Redis 性能测试是通过同时执行多个命令实现的。redis 性能测试的基本命令如下：

redis-benchmark [option] [option value]

|  |  |  |
| --- | --- | --- |
| 选项 | 描述 | 默认值 |
| -h | 指定服务器主机名 | 127.0.0.1 |
| -p | 指定服务器端口 | 6379 |
| -s | 指定服务器 socket |  |
| -c | 指定并发连接数 | 50 |
| -n | 指定请求数 | 10000 |
| -d | 以字节的形式指定 SET/GET 值的数据大小 | 2 |
| -k | 1=keep alive 0=reconnect | 1 |
| -r | SET/GET/INCR 使用随机 key, SADD 使用随机值 |  |
| -P | 通过管道传输 <numreq> 请求 | 1 |
| -q | 强制退出 redis。仅显示 query/sec 值 |  |
| --csv | 以 CSV 格式输出 |  |
| -l | 生成循环，永久执行测试 |  |
| -t | 仅运行以逗号分隔的测试命令列表。 |  |
| -I | Idle 模式。仅打开 N 个 idle 连接并等待。 |  |

## 8.4 客户端连接

Redis 通过监听一个 TCP 端口或者 Unix socket 的方式来接收来自客户端的连接，当一个连接建立后，Redis 内部会进行以下一些操作：

首先，客户端 socket 会被设置为非阻塞模式，因为 Redis 在网络事件处理上采用的是非阻塞多路复用模型。

然后，为这个 socket 设置 TCP\_NODELAY 属性，禁用 Nagle 算法

然后，创建一个可读的文件事件用于监听这个客户端 socket 的数据发送

在 Redis2.4 中，最大连接数是被直接硬编码在代码里面的，而在2.6版本中这个值变成可配置的。

maxclients 的默认值是 10000，你也可以在 redis.conf 中或者通过CONFIG SET maxclients value对这个值进行修改，或者在服务启动时设置最大连接数为 100000：redis-server --maxclients 100000。

管道技术

Redis是一种基于客户端-服务端模型以及请求/响应协议的TCP服务。这意味着通常情况下一个请求会遵循以下步骤：

客户端向服务端发送一个查询请求，并监听Socket返回，通常是以阻塞模式，等待服务端响应。

服务端处理命令，并将结果返回给客户端。

## 8.5 管道技术

Redis 管道技术可以在服务端未响应时，客户端可以继续向服务端发送请求，并最终一次性读取所有服务端的响应。这个和Linux的管道类似，命令1 [|命令2] [|命令2][……]。

管道技术最显著的优势是提高了 redis 服务的性能。

## 8.6 分区

分区是分割数据到多个Redis实例的处理过程，因此每个实例只保存key的一个子集。

### 8.6.1 分区的优势

（1）通过利用多台计算机内存的和值，允许我们构造更大的数据库。

（2）通过多核和多台计算机，允许我们扩展计算能力；

（3）通过多台计算机和网络适配器，允许我们扩展网络带宽。

### 8.6.2 分区的不足

（1）涉及多个key的操作通常是不被支持的。举例来说，当两个set映射到不同的redis实例上时，你就不能对这两个set执行交集操作。

（2）涉及多个key的redis事务不能使用。

（3）当使用分区时，数据处理较为复杂，比如你需要处理多个rdb/aof文件，并且从多个实例和主机备份持久化文件。

（4）增加或删除容量也比较复杂。redis集群大多数支持在运行时增加、删除节点的透明数据平衡的能力，但是类似于客户端分区、代理等其他系统则不支持这项特性。然而，一种叫做presharding的技术对此是有帮助的。

### 8.6.3 分区类型

Redis 有两种类型分区。 假设有4个Redis实例 R0，R1，R2，R3，和类似user:1，user:2这样的表示用户的多个key，对既定的key有多种不同方式来选择这个key存放在哪个实例中。也就是说，有不同的系统来映射某个key到某个Redis服务。

（1）范围分区

最简单的分区方式是按范围分区，就是映射一定范围的对象到特定的Redis实例。

比如，ID从0到10000的用户会保存到实例R0，ID从10001到 20000的用户会保存到R1，以此类推。

这种方式是可行的，并且在实际中使用，不足就是要有一个区间范围到实例的映射表。这个表要被管理，同时还需要各种对象的映射表，通常对Redis来说并非是好的方法。

（2）哈希分区

另外一种分区方法是hash分区。这对任何key都适用，也无需是object\_name:这种形式，像下面描述的一样简单：

用一个hash函数将key转换为一个数字，比如使用crc32 hash函数。对key foobar执行crc32(foobar)会输出类似93024922的整数。

对这个整数取模，将其转化为0-3之间的数字，就可以将这个整数映射到4个Redis实例中的一个了。93024922 % 4 = 2，就是说key foobar应该被存到R2实例中。注意：取模操作是取除的余数，通常在多种编程语言中用%操作符实现。

# 9 Redis C/.C++ API

## 9.1 C API

函数原型：

redisContext \*redisConnect(const char \*ip, int port)

说明：该函数用来连接redis数据库，参数为数据库的ip地址和端口，一般redis数据库的端口为6379

该函数返回一个结构体redisContext。

函数原型：

void \*redisCommand(redisContext \*c, const char \*format, ...);

说明：该函数执行命令，就如sql数据库中的SQL语句一样，只是执行的是redis数据库中的操作命令，第一个参数为连接数据库时返回的redisContext，剩下的参数为变参，就如C标准函数printf函数一样的变参。返回值为void\*，一般强制转换成为redisReply类型的进行进一步的处理。

函数原型

void freeReplyObject(void \*reply);

说明：释放redisCommand执行后返回的redisReply所占用的内存

函数原型：

void redisFree(redisContext \*c);

说明：释放redisConnect()所产生的连接。

APPEND key value

启用时间：自2.0.0起可用。

时间复杂度： O（1）。假设附加值小，已经存可以是任何大小，则摊销的时间复杂度是O（1），因为Redis使用的动态字符串库将使每次重新分配的可用空间翻番。

功能：如果key已经存在并且是字符串，则该命令将追加value到字符串的末尾。如果key不存在，它将被创建并设置为空字符串，因此APPEND在这种特殊情况下 将类似于SET。

返回值：整数，追加操作后的字符串长度。

AUTH password

启用时间：自1.0.0起可用。

功能： 在启用密码保护的Redis服务器上请求验证。在允许客户端执行命令之前，可以指示Redis要求密码。这是使用配置文件中的requirepass指令设置的。

返回值：ssword与配置文件中的密码相匹配，服务器将回复OK状态码并开始接受命令。否则，返回错误，客户端需要尝试新的密码。

注意：由于Redis的性能很高，可以在很短的时间内并行尝试大量的密码，因此请确保生成一个强大而非常长的密码，以便这种攻击是不可行的。

BGREWRITEAOF

启用时间：自1.0.0起可用。

功能：启用Append Only File 重写进程。重写将创建当前的“Append Only File”的小型优化版本。如果BGREWRITEAOF失败，没有数据丢失，旧的AOF文件将不被修改。如果没有后台进程持续执行，则重写将仅由Redis触发。特别的：

如果Redis子进程正在磁盘上创建快照，则AOF重写将被调度，但不会启动，直到生成RDB文件的子程序终止。在这种情况下，BGREWRITEAOF仍然会返回一个OK代码，但会返回一个适当的消息。可以查看redis2.6 的INFO命令来检查AOF重写是否调用。

如果AOF重写已经在进行中，则命令返回错误，并且将不再安排AOF重写。

自Redis 2.4由Redis自动触发AOF重写，但 BGREWRITEAOF命令可以随时触发重写。

返回值：始终返回OK。

BGSAVE

启用时间：自1.0.0起可用。

功能：在后台保存数据库。redis立即返回ok。Redis 创建子进程，父级继续为客户端服务，子进程将数据库保存在磁盘上，然后退出。客户端可以使用LASTSAVE 命令检查操作是否成功。

返回值：返回OK。

BITCOUNT key [start end]

启用时间：自2.6.0起可用。

时间复杂度：O（N）

功能：

计算字符串中的设置位数（人口计数）。默认情况下，检查字符串的所有字节。通过附加参数start和end指定计数区间。像GETRANGE命令一样，起始和结束可以包含负值，以便从字符串的末尾开始索引字节，其中-1是最后一个字节，-2是倒数第二个等等。不存在的键被视为空字符串，因此命令将返回零。

返回值：整数