**一、REDIS简介**

REmote DIctionary Server(Redis) 是一个由Salvatore Sanfilippo写的key-value存储系统。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是一个开源的使用ANSI C语言编写、遵守BSD协议、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。

它通常被称为数据结构服务器，因为值（value）可以是 字符串(String), 哈希(Hash), 列表(list), 集合(sets) 和 有序集合(sorted sets)等类型。

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。

启动服务：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。

启动客户端：redis-cli -h host -p port -a password，后面加上 --raw可以避免中文乱码

redis-cli.exe -h 127.0.0.1 -p 6379

redis-cli --raw

下表列出了 redis 连接的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [AUTH password](https://www.runoob.com/redis/connection-auth.html)验证密码是否正确 |
| 2 | [ECHO message](https://www.runoob.com/redis/connection-echo.html)打印字符串 |
| 3 | [PING](https://www.runoob.com/redis/connection-ping.html)查看服务是否运行，返回pong |
| 4 | [QUIT](https://www.runoob.com/redis/connection-quit.html)关闭当前连接 |
| 5 | [SELECT index](https://www.runoob.com/redis/connection-select.html)切换到指定的数据库，从0库开始 |
| 6 | shutdown 停止redis服务 |
| 7 | keys \* 获取所有key |
| 8 | randomkey 从当前库随机返回一个key |
| 9 | del [key] 删除key |
| 10 | exists [key] 判断是否存在key |
| 11 | expire key N key在N秒过期 |
| 12 | peexpire key N key在N毫秒过期 |
| 13 | object encoding key 得到key的类型，string里有三种编码，int，embstr，raw |

# 二、Redis 配置

语法

Redis CONFIG 命令格式如下：

redis 127.0.0.1:6379> CONFIG GET CONFIG\_SETTING\_NAME

实例:

redis 127.0.0.1:6379> CONFIG GET loglevel

1) "loglevel"

2) "notice"

使用 \* 号获取所有配置项：redis 127.0.0.1:6379> CONFIG GET \*

**编辑配置**

你可以通过修改 redis.conf 文件或使用 CONFIG set 命令来修改配置。

语法

CONFIG SET 命令基本语法

redis 127.0.0.1:6379> CONFIG SET CONFIG\_SETTING\_NAME NEW\_CONFIG\_VALUE

实例

redis 127.0.0.1:6379> CONFIG SET loglevel "notice"

OK

redis 127.0.0.1:6379> CONFIG GET loglevel

1) "loglevel"2) "notice"

参数说明

redis.conf 配置项说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 配置项 | 说明 |
| 1 | daemonize no | 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（Windows 不支持守护线程的配置为 no ） |
| 2 | pidfile /var/run/redis.pid | 当 Redis 以守护进程方式运行时，Redis 默认会把 pid 写入 /var/run/redis.pid 文件，可以通过 pidfile 指定 |
| 3 | port 6379 | 指定 Redis 监听端口，默认端口为 6379，作者在自己的一篇博文中解释了为什么选用 6379 作为默认端口，因为 6379 在手机按键上 MERZ 对应的号码，而 MERZ 取自意大利歌女 Alessia Merz 的名字 |
| 4 | bind 127.0.0.1 | 绑定的主机地址,*可被访问的地址，0.0.0.0，表示都可访问* |
| 5 | timeout 300 | 当客户端闲置多长秒后关闭连接，如果指定为 0 ，表示关闭该功能 |
| 6 | loglevel notice | 指定日志记录级别，Redis 总共支持四个级别：debug、verbose、notice、warning，默认为 notice |
| 7 | logfile stdout | 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 |
| 8 | databases 16 | 设置数据库的数量，默认数据库为0，可以使用SELECT 命令在连接上指定数据库id |
| 9 | save <seconds> <changes> | 指定在多长时间内，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合  Redis 默认配置文件中提供了三个条件：  **save 900 1**  **save 300 10**  **save 60 10000**  分别表示 900 秒（15 分钟）内有 1 个更改，300 秒（5 分钟）内有 10 个更改以及 60 秒内有 10000 个更改。 |
| 10 | rdbcompression 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 |
| 11 | dbfilename dump.rdb | 指定本地数据库文件名，默认值为 dump.rdb |
| 12 | dir ./ | 指定本地数据库存放目录 |
| 13 | slaveof <masterip> <masterport> | 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 |
| 14 | masterauth <master-password> | 当 master 服务设置了密码保护时，slav 服务连接 master 的密码 |
| 15 | requirepass foobared | 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，默认关闭 |
| 16 | maxclients 128 | 设置同一时间最大客户端连接数，默认无限制，Redis 可以同时打开的客户端连接数为 Redis 进程可以打开的最大文件描述符数，如果设置 maxclients 0，表示不作限制。当客户端连接数到达限制时，Redis 会关闭新的连接并向客户端返回 max number of clients reached 错误信息 |
| 17 | maxmemory <bytes> | 指定 Redis 最大内存限制，Redis 在启动时会把数据加载到内存中，达到最大内存后，Redis 会先尝试清除已到期或即将到期的 Key，当此方法处理 后，仍然到达最大内存设置，将无法再进行写入操作，但仍然可以进行读取操作。Redis 新的 vm 机制，会把 Key 存放内存，Value 会存放在 swap 区 |
| 18 | appendonly no | 指定是否在每次更新操作后进行日志记录，Redis 在默认情况下是异步的把数据写入磁盘，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 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。默认为 no |
| 19 | appendfilename appendonly.aof | 指定更新日志文件名，默认为 appendonly.aof |
| 20 | appendfsync everysec | 指定更新日志条件，共有 3 个可选值：   * **no**：表示等操作系统进行数据缓存同步到磁盘（快） * **always**：表示每次更新操作后手动调用 fsync() 将数据写到磁盘（慢，安全） * **everysec**：表示每秒同步一次（折中，默认值） |
| 21 | vm-enabled no | 指定是否启用虚拟内存机制，默认值为 no，简单的介绍一下，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（在后面的文章我会仔细分析 Redis 的 VM 机制） |
| 22 | vm-swap-file /tmp/redis.swap | 虚拟内存文件路径，默认值为 /tmp/redis.swap，不可多个 Redis 实例共享 |
| 23 | vm-max-memory 0 | 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 |
| 24 | vm-page-size 32 | 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，作者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 |
| 25 | vm-pages 134217728 | 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。 |
| 26 | vm-max-threads 4 | 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 |
| 27 | glueoutputbuf yes | 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 |
| 28 | hash-max-zipmap-entries 64  hash-max-zipmap-value 512 | 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法 |
| 29 | activerehashing yes | 指定是否激活重置哈希，默认为开启（后面在介绍 Redis 的哈希算法时具体介绍） |
| 30 | include /path/to/local.conf | 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 |

# 三、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)。

String（字符串） 粉丝数、关注数、微博数 Map<string, string>

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，string 类型的值最大能存储 512MB。

redis 127.0.0.1:6379> SET runoob "教程"

OK

redis 127.0.0.1:6379> GET runoob"教程"

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [SET key value](https://www.runoob.com/redis/strings-set.html) 设置指定 key 的值 |
| 2 | [GET key](https://www.runoob.com/redis/strings-get.html) 获取指定 key 的值。 |
| 3 | [GETRANGE key start end](https://www.runoob.com/redis/strings-getrange.html) 返回 key 中字符串值的子字符 |
| 4 | [GETSET key value](https://www.runoob.com/redis/strings-getset.html) 将给定 key 的值设为 value ，并返回 key 的旧值(old value)。 |
| 5 | [GETBIT key offset](https://www.runoob.com/redis/strings-getbit.html) 对 key 所储存的字符串值，获取指定偏移量上的位(bit)。 |
| 6 | [MGET key1 [key2..]](https://www.runoob.com/redis/strings-mget.html) 获取所有(一个或多个)给定 key 的值。 |
| 7 | [SETBIT key offset value](https://www.runoob.com/redis/strings-setbit.html) 对 key 所储存的字符串值，设置或清除指定偏移量上的位(bit)。 |
| 8 | [SETEX key seconds value](https://www.runoob.com/redis/strings-setex.html) 将值 value 关联到 key ，并将 key 的过期时间设为 seconds (以秒为单位)。 |
| 9 | [SETNX key value](https://www.runoob.com/redis/strings-setnx.html) 只有在 key 不存在时设置 key 的值。 |
| 10 | [SETRANGE key offset value](https://www.runoob.com/redis/strings-setrange.html) 用 value 参数覆写给定 key 所储存的字符串值，从偏移量 offset 开始。 |
| 11 | [STRLEN key](https://www.runoob.com/redis/strings-strlen.html) 返回 key 所储存的字符串值的长度。 |
| 12 | [MSET key value [key value ...]](https://www.runoob.com/redis/strings-mset.html) 同时设置一个或多个 key-value 对。 |
| 13 | [MSETNX key value [key value ...]](https://www.runoob.com/redis/strings-msetnx.html) 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。 |
| 14 | [PSETEX key milliseconds value](https://www.runoob.com/redis/strings-psetex.html) 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。 |
| 15 | [INCR key](https://www.runoob.com/redis/strings-incr.html) 将 key 中储存的数字值增一。 |
| 16 | [INCRBY key increment](https://www.runoob.com/redis/strings-incrby.html) 将 key 所储存的值加上给定的增量值（increment） 。 |
| 17 | [INCRBYFLOAT key increment](https://www.runoob.com/redis/strings-incrbyfloat.html) 将 key 所储存的值加上给定的浮点增量值（increment） 。 |
| 18 | [DECR key](https://www.runoob.com/redis/strings-decr.html) 将 key 中储存的数字值减一。 |
| 19 | [DECRBY key decrement](https://www.runoob.com/redis/strings-decrby.html) key 所储存的值减去给定的减量值（decrement） 。 |
| 20 | [APPEND key value](https://www.runoob.com/redis/strings-append.html) 如果 key 已经存在并且是一个字符串， APPEND 命令将指定的 value 追加到该 key 原来值（value）的末尾。 |

Hash（哈希） ？购物车Map<string, Map<Object, Object>>

Redis hash 是一个键值(key=>value)对集合。

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。

HMSET, HGET 命令，HMSET 设置了两个 field=>value 对, HGET 获取对应 field 对应的 value。

每个 hash 可以存储 232 -1 键值对（40多亿）。

redis 127.0.0.1:6379> HMSET runoob field1 "Hello" field2 "World""OK"

redis 127.0.0.1:6379> HGET runoob field1"Hello"

redis 127.0.0.1:6379> HGET runoob field2"World"

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [HDEL key field1 [field2]](https://www.runoob.com/redis/hashes-hdel.html) 删除一个或多个哈希表字段 |
| 2 | [HEXISTS key field](https://www.runoob.com/redis/hashes-hexists.html) 查看哈希表 key 中，指定的字段是否存在。 |
| 3 | [HGET key field](https://www.runoob.com/redis/hashes-hget.html) 获取存储在哈希表中指定字段的值。 |
| 4 | [HGETALL key](https://www.runoob.com/redis/hashes-hgetall.html) 获取在哈希表中指定 key 的所有字段和值 |
| 5 | [HINCRBY key field increment](https://www.runoob.com/redis/hashes-hincrby.html) 为哈希表 key 中的指定字段的整数值加上增量 increment 。 |
| 6 | [HINCRBYFLOAT key field increment](https://www.runoob.com/redis/hashes-hincrbyfloat.html) 为哈希表 key 中的指定字段的浮点数值加上增量 increment 。 |
| 7 | [HKEYS key](https://www.runoob.com/redis/hashes-hkeys.html) 获取所有哈希表中的字段 |
| 8 | [HLEN key](https://www.runoob.com/redis/hashes-hlen.html) 获取哈希表中字段的数量 |
| 9 | [HMGET key field1 [field2]](https://www.runoob.com/redis/hashes-hmget.html) 获取所有给定字段的值 |
| 10 | [HMSET key field1 value1 [field2 value2 ]](https://www.runoob.com/redis/hashes-hmset.html) 同时将多个 field-value (域-值)对设置到哈希表 key 中。 |
| 11 | [HSET key field value](https://www.runoob.com/redis/hashes-hset.html) 将哈希表 key 中的字段 field 的值设为 value 。 |
| 12 | [HSETNX key field value](https://www.runoob.com/redis/hashes-hsetnx.html) 只有在字段 field 不存在时，设置哈希表字段的值。 |
| 13 | [HVALS key](https://www.runoob.com/redis/hashes-hvals.html) 获取哈希表中所有值。 |
| 14 | [HSCAN key cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/hashes-hscan.html" \t "https://www.runoob.com/redis/_blank) 迭代哈希表中的键值对。 |

List（列表） ？秒杀、抢票 Map<string, List<string>>

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)。

redis 127.0.0.1:6379> lpush runoob redis(integer) 1

redis 127.0.0.1:6379> lpush runoob mongodb(integer) 2

redis 127.0.0.1:6379> lpush runoob rabitmq(integer) 3

redis 127.0.0.1:6379> lrange runoob 0 10

1. "rabitmq"
2. "mongodb"

3) "redis"

下表列出了列表相关的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [BLPOP key1 [key2 ] timeout](https://www.runoob.com/redis/lists-blpop.html" \t "https://www.runoob.com/redis/_blank) 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。 |
| 2 | [BRPOP key1 [key2 ] timeout](https://www.runoob.com/redis/lists-brpop.html" \t "https://www.runoob.com/redis/_blank) 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。 |
| 3 | [BRPOPLPUSH source destination timeout](https://www.runoob.com/redis/lists-brpoplpush.html" \t "https://www.runoob.com/redis/_blank) 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。 |
| 4 | [LINDEX key index](https://www.runoob.com/redis/lists-lindex.html" \t "https://www.runoob.com/redis/_blank) 通过索引获取列表中的元素 |
| 5 | [LINSERT key BEFORE|AFTER pivot value](https://www.runoob.com/redis/lists-linsert.html" \t "https://www.runoob.com/redis/_blank) 在列表的元素前或者后插入元素 |
| 6 | [LLEN key](https://www.runoob.com/redis/lists-llen.html" \t "https://www.runoob.com/redis/_blank) 获取列表长度 |
| 7 | [LPOP key](https://www.runoob.com/redis/lists-lpop.html" \t "https://www.runoob.com/redis/_blank) 移出并获取列表的第一个元素 |
| 8 | [LPUSH key value1 [value2]](https://www.runoob.com/redis/lists-lpush.html" \t "https://www.runoob.com/redis/_blank) 将一个或多个值插入到列表头部 |
| 9 | [LPUSHX key value](https://www.runoob.com/redis/lists-lpushx.html" \t "https://www.runoob.com/redis/_blank) 将一个值插入到已存在的列表头部 |
| 10 | [LRANGE key start stop](https://www.runoob.com/redis/lists-lrange.html" \t "https://www.runoob.com/redis/_blank) 获取列表指定范围内的元素 |
| 11 | [LREM key count value](https://www.runoob.com/redis/lists-lrem.html" \t "https://www.runoob.com/redis/_blank) 移除列表元素 |
| 12 | [LSET key index value](https://www.runoob.com/redis/lists-lset.html" \t "https://www.runoob.com/redis/_blank) 通过索引设置列表元素的值 |
| 13 | [LTRIM key start stop](https://www.runoob.com/redis/lists-ltrim.html" \t "https://www.runoob.com/redis/_blank) 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。 |
| 14 | [RPOP key](https://www.runoob.com/redis/lists-rpop.html" \t "https://www.runoob.com/redis/_blank) 移除列表的最后一个元素，返回值为移除的元素。 |
| 15 | [RPOPLPUSH source destination](https://www.runoob.com/redis/lists-rpoplpush.html" \t "https://www.runoob.com/redis/_blank) 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 |
| 16 | [RPUSH key value1 [value2]](https://www.runoob.com/redis/lists-rpush.html" \t "https://www.runoob.com/redis/_blank) 在列表中添加一个或多个值 |
| 17 | [RPUSHX key value](https://www.runoob.com/redis/lists-rpushx.html" \t "https://www.runoob.com/redis/_blank) 为已存在的列表添加值 |

**Set（集合） ？关注好友、在线人**Map<string, Map<Object, Object>>

Redis 的 Set 是 string 类型的无序集合。不允许重复的成员

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)。

sadd 命令：sadd key member

添加一个 string 元素到 key 对应的 set 集合中，成功返回 1，如果元素已经在集合中返回 0。

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)。

redis 127.0.0.1:6379> sadd runoob redis(integer) 1

redis 127.0.0.1:6379> sadd runoob mongodb(integer) 1

redis 127.0.0.1:6379> sadd runoob rabitmq(integer) 1

redis 127.0.0.1:6379> sadd runoob rabitmq(integer) 0

redis 127.0.0.1:6379> smembers runoob

1. "redis"
2. "rabitmq"

3) "mongodb"

下表列出了 Redis 集合基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [SADD key member1 [member2]](https://www.runoob.com/redis/sets-sadd.html) 向集合添加一个或多个成员 |
| 2 | [SCARD key](https://www.runoob.com/redis/sets-scard.html) 获取集合的成员数 |
| 3 | [SDIFF key1 [key2]](https://www.runoob.com/redis/sets-sdiff.html) 返回给定所有集合的差集 |
| 4 | [SDIFFSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sdiffstore.html) 返回给定所有集合的差集并存储在 destination 中 |
| 5 | [SINTER key1 [key2]](https://www.runoob.com/redis/sets-sinter.html) 返回给定所有集合的交集 |
| 6 | [SINTERSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sinterstore.html) 返回给定所有集合的交集并存储在 destination 中 |
| 7 | [SISMEMBER key member](https://www.runoob.com/redis/sets-sismember.html) 判断 member 元素是否是集合 key 的成员 |
| 8 | [SMEMBERS key](https://www.runoob.com/redis/sets-smembers.html) 返回集合中的所有成员 |
| 9 | [SMOVE source destination member](https://www.runoob.com/redis/sets-smove.html) 将 member 元素从 source 集合移动到 destination 集合 |
| 10 | [SPOP key](https://www.runoob.com/redis/sets-spop.html) 移除并返回集合中的一个随机元素 |
| 11 | [SRANDMEMBER key [count]](https://www.runoob.com/redis/sets-srandmember.html) 返回集合中一个或多个随机数 |
| 12 | [SREM key member1 [member2]](https://www.runoob.com/redis/sets-srem.html) 移除集合中一个或多个成员 |
| 13 | [SUNION key1 [key2]](https://www.runoob.com/redis/sets-sunion.html) 返回所有给定集合的并集 |
| 14 | [SUNIONSTORE destination key1 [key2]](https://www.runoob.com/redis/sets-sunionstore.html) 所有给定集合的并集存储在 destination 集合中 |
| 15 | [SSCAN key cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/sets-sscan.html) 迭代集合中的元素 |

zset(sorted set：有序集合)Map<string, Map<Object, Object>>

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。

zadd命令，添加元素到集合，元素在集合中存在则更新对应score

redis 127.0.0.1:6379> zadd runoob 0 redis(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 mongodb(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 rabitmq(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 rabitmq(integer) 0

redis 127.0.0.1:6379> > ZRANGEBYSCORE runoob 0 1000

1. "mongodb"
2. "rabitmq"

3) "redis"

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [ZADD key score1 member1 [score2 member2]](https://www.runoob.com/redis/sorted-sets-zadd.html) 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 |
| 2 | [ZCARD key](https://www.runoob.com/redis/sorted-sets-zcard.html) 获取有序集合的成员数 |
| 3 | [ZCOUNT key min max](https://www.runoob.com/redis/sorted-sets-zcount.html) 计算在有序集合中指定区间分数的成员数 |
| 4 | [ZINCRBY key increment member](https://www.runoob.com/redis/sorted-sets-zincrby.html) 有序集合中对指定成员的分数加上增量 increment |
| 5 | [ZINTERSTORE destination numkeys key [key ...]](https://www.runoob.com/redis/sorted-sets-zinterstore.html) 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 |
| 6 | [ZLEXCOUNT key min max](https://www.runoob.com/redis/sorted-sets-zlexcount.html) 在有序集合中计算指定字典区间内成员数量 |
| 7 | [ZRANGE key start stop [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrange.html) 通过索引区间返回有序集合指定区间内的成员 |
| 8 | [ZRANGEBYLEX key min max [LIMIT offset count]](https://www.runoob.com/redis/sorted-sets-zrangebylex.html) 通过字典区间返回有序集合的成员 |
| 9 | [ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT]](https://www.runoob.com/redis/sorted-sets-zrangebyscore.html) 通过分数返回有序集合指定区间内的成员 |
| 10 | [ZRANK key member](https://www.runoob.com/redis/sorted-sets-zrank.html) 返回有序集合中指定成员的索引 |
| 11 | [ZREM key member [member ...]](https://www.runoob.com/redis/sorted-sets-zrem.html) 移除有序集合中的一个或多个成员 |
| 12 | [ZREMRANGEBYLEX key min max](https://www.runoob.com/redis/sorted-sets-zremrangebylex.html) 移除有序集合中给定的字典区间的所有成员 |
| 13 | [ZREMRANGEBYRANK key start stop](https://www.runoob.com/redis/sorted-sets-zremrangebyrank.html) 移除有序集合中给定的排名区间的所有成员 |
| 14 | [ZREMRANGEBYSCORE key min max](https://www.runoob.com/redis/sorted-sets-zremrangebyscore.html) 移除有序集合中给定的分数区间的所有成员 |
| 15 | [ZREVRANGE key start stop [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrevrange.html) 返回有序集中指定区间内的成员，通过索引，分数从高到低 |
| 16 | [ZREVRANGEBYSCORE key max min [WITHSCORES]](https://www.runoob.com/redis/sorted-sets-zrevrangebyscore.html) 返回有序集中指定分数区间内的成员，分数从高到低排序 |
| 17 | [ZREVRANK key member](https://www.runoob.com/redis/sorted-sets-zrevrank.html) 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 |
| 18 | [ZSCORE key member](https://www.runoob.com/redis/sorted-sets-zscore.html) 返回有序集中，成员的分数值 |
| 19 | [ZUNIONSTORE destination numkeys key [key ...]](https://www.runoob.com/redis/sorted-sets-zunionstore.html) 计算给定的一个或多个有序集的并集，并存储在新的 key 中 |
| 20 | [ZSCAN key cursor [MATCH pattern] [COUNT count]](https://www.runoob.com/redis/sorted-sets-zscan.html) 迭代有序集合中的元素（包括元素成员和元素分值） |

总结：各个数据类型应用场景

| **类型** | **简介** | **特性** | **场景** |
| --- | --- | --- | --- |
| String(字符串) | 二进制安全 | 可以包含任何数据,比如jpg图片或者序列化的对象,一个键最大能存储512M | --- |
| Hash(字典) | 键值对集合,即编程语言中的Map类型 | 适合存储对象,并且可以像数据库中update一个属性一样只修改某一项属性值(Memcached中需要取出整个字符串反序列化成对象修改完再序列化存回去) | 存储、读取、修改用户属性 |
| List(列表) | 链表(双向链表) | 增删快,提供了操作某一段元素的API | 1,最新消息排行等功能(比如朋友圈的时间线) 2,消息队列 |
| Set(集合) | 哈希表实现,元素不重复 | 1、添加、删除,查找的复杂度都是O(1) 2、为集合提供了求交集、并集、差集等操作 | 1、共同好友 2、利用唯一性,统计访问网站的所有独立ip 3、好友推荐时,根据tag求交集,大于某个阈值就可以推荐 |
| Sorted Set(有序集合) | 将Set中的元素增加一个权重参数score,元素按score有序排列 | 数据插入集合时,已经进行天然排序 | 1、排行榜 2、带权重的消息队列 |

# 四、Redis 发布订阅

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：



实例：以下实例演示了发布订阅是如何工作的。在我们实例中我们创建了订阅频道名为 redisChat:

redis 127.0.0.1:6379> SUBSCRIBE redisChat

Reading messages... (press Ctrl-C to quit)1) "subscribe"2) "redisChat"3) (integer) 1

现在，我们先重新开启个 redis 客户端，然后在同一个频道 redisChat 发布两次消息，订阅者就能接收到消息。

redis 127.0.0.1:6379> PUBLISH redisChat "Redis is a great caching technique"

(integer) 1

redis 127.0.0.1:6379> PUBLISH redisChat "Learn redis by runoob.com"

(integer) 1

# 订阅者的客户端会显示如下消息

1. "message"
2. "redisChat"
3. "Redis is a great caching technique"
4. "message"
5. "redisChat"

3) "Learn redis by runoob.com"

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [PSUBSCRIBE pattern [pattern ...]](https://www.runoob.com/redis/pub-sub-psubscribe.html) 订阅一个或多个符合给定模式的频道。 |
| 2 | [PUBSUB subcommand [argument [argument ...]]](https://www.runoob.com/redis/pub-sub-pubsub.html) 查看订阅与发布系统状态。 |
| 3 | [PUBLISH channel message](https://www.runoob.com/redis/pub-sub-publish.html) 将信息发送到指定的频道。 |
| 4 | [PUNSUBSCRIBE [pattern [pattern ...]]](https://www.runoob.com/redis/pub-sub-punsubscribe.html) 退订所有给定模式的频道。 |
| 5 | [SUBSCRIBE channel [channel ...]](https://www.runoob.com/redis/pub-sub-subscribe.html) 订阅给定的一个或多个频道的信息。 |
| 6 | [UNSUBSCRIBE [channel [channel ...]]](https://www.runoob.com/redis/pub-sub-unsubscribe.html) 指退订给定的频道。 |

# 五、Redis 事务【了解】

Redis 事务可以一次执行多个命令， 并且带有以下三个重要的保证：

1批量操作在发送 EXEC 命令前被放入队列缓存。

2收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余命令依然被执行。

3在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。

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。

单个 Redis 命令的执行是原子性的，但 Redis 没有在事务上增加任何维持原子性的机制，所以 Redis 事务的执行并不是原子性的。

事务可以理解为一个打包的批量执行脚本，但批量指令并非原子化的操作，中间某条指令的失败不会导致前面已做指令的回滚，也不会造成后续的指令不做。

实例

以下是一个事务的例子， 它先以 MULTI 开始一个事务， 然后将多个命令入队到事务中， 最后由 EXEC 命令触发事务， 一并执行事务中的所有命令：

redis 127.0.0.1:6379> MULTI

OK

redis 127.0.0.1:6379> SET book-name "Mastering C++ in 21 days"

QUEUED

redis 127.0.0.1:6379> GET book-name

QUEUED

redis 127.0.0.1:6379> SADD tag "C++" "Programming" "Mastering Series"

QUEUED

redis 127.0.0.1:6379> SMEMBERS tag

QUEUED

redis 127.0.0.1:6379> EXEC

1. OK
2. "Mastering C++ in 21 days"
3. (integer) 3

4) 1) "Mastering Series"

2) "C++"

3) "Programming"

|  |  |
| --- | --- |
| 1 | [multi](https://www.runoob.com/redis/pub-sub-subscribe.html) 启动事务。标记一个事务开始 |
| 2 | discard  放弃事务，取消事务中所有命令 |
| 3 | exec  提交事务，执行所有事务内的命令 |
| 4 | unwatch  取消watch命令对所有key的监视 |
| 5 | watch key [key...]  监视一个或多个key，如果事务执行前key被其他命令所动，那么事务将被打断 |

watch监控：

悲观锁、乐观锁

# 六、Redis 数据备份与恢复

Redis SAVE 命令用于创建当前数据库的备份。该命令将在 redis 安装目录中创建dump.rdb文件。

语法

redis Save 命令基本语法如下：SAVE

redis 127.0.0.1:6379> SAVE

OK

恢复数据

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 命令，如下所示：

redis 127.0.0.1:6379> CONFIG GET dir

1. "dir"
2. 2) "/usr/local/redis/bin"

Bgsave

创建 redis 备份文件也可以使用命令 BGSAVE，该命令在后台执行。

127.0.0.1:6379> BGSAVE

Background saving started

# 七、Redis 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。

我们可以通过以下命令查看是否设置了密码验证：

127.0.0.1:6379> CONFIG get requirepass

1. "requirepass"

2) ""

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。

你可以通过以下命令来修改该参数：

127.0.0.1:6379> CONFIG set requirepass "runoob"

OK

127.0.0.1:6379> CONFIG get requirepass

1. "requirepass"

2) "runoob"

# 八、Redis 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。

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。

管道技术最显著的优势是提高了 redis 服务的性能。

以下实例中我们通过使用 PING 命令查看redis服务是否可用， 之后我们设置了 runoobkey 的值为 redis，然后我们获取 runoobkey 的值并使得 visitor 自增 3 次。

在返回的结果中我们可以看到这些命令一次性向 redis 服务提交，并最终一次性读取所有服务端的响应

$(echo -en "PING\r\n SET runoobkey redis\r\nGET runoobkey\r\nINCR visitor\r\nINCR visitor\r\nINCR visitor\r\n"; sleep 10) | nc localhost 6379

+PONG+OK

redis

:1

:2

:3

# 九、Redis 持久化persistence

RDB、AOF

RDB数据不实时，在conf中，设置save参数

Redis 默认配置文件中提供了三个条件：

**save 900 1**

**save 300 10**

**save 60 10000**

分别表示 900 秒（15 分钟）内有 1 个更改，300 秒（5 分钟）内有 10 个更改以及 60 秒内有 10000 个更改。

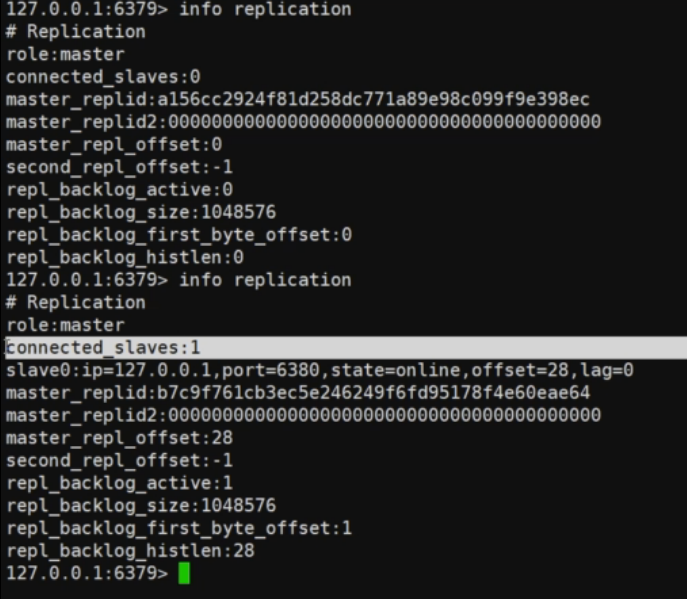
AOF模式，记录每次对服务器的操作

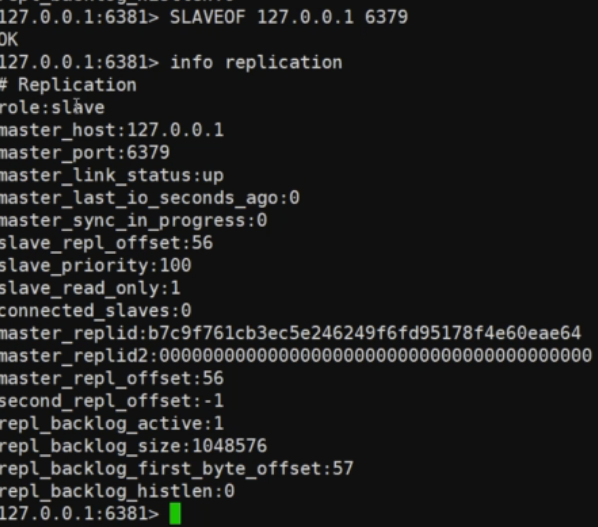
# 十、Redis 复制（中心化）【了解】

master以写为主，slave以读为主

读写分离、容灾

配从不配主，改从机的conf配置或用命令slaveof





slave从头复制，还是切入点复制？ 全复制

slave能不能写？ 不能，只能读

主机掉后，从机变成主机，还是保持从机？ 保持从机slave，还能读

主机掉后，从机可以当主机吗？可以，从机上用slaveof no one 切换成主机，其他从机需要修改新主机，或等原主机恢复。从机改主机后，可手工再改为从机。

哨兵模式sentinel，自动实现，主从切换。

# 十一、Redis集群（去中心化）

高可用：持续提供服务

高并发：同时处理请求数

redis集群数据分片：哈希槽

每个key值通过CRC16校验对 N取模确定放哪个槽

多个主master组成集群，为每个master设从机

设置集群conf：

cluster-enabled yes

cluster-config-file nodes-6379.conf

cluster-node-timeout 15000

appendonly yes

客户端连接时，加-c参数：redis -c -p XXXX

# 十二、Redis客户端工具

redis-desktop-manager

redisplus

**一、缓存穿透**

**1、概念**

缓存穿透的概念很简单，用户想要查询一个数据，发现redis内存数据库没有，也就是缓存没有命中，于是向持久层数据库查询。发现也没有，于是本次查询失败。当用户很多的时候，缓存都没有命中，于是都去请求了持久层数据库。这会给持久层数据库造成很大的压力，这时候就相当于出现了缓存穿透。

**这里需要注意和缓存击穿的区别，缓存击穿，是指一个key非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个屏障上凿开了一个洞。**

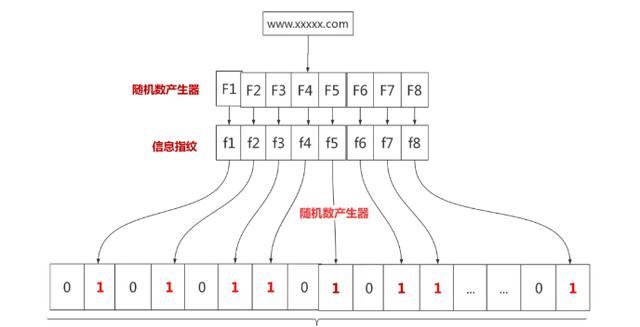
为了避免缓存穿透其实有很多种解决方案。下面介绍几种。

**2、解决方案**

**（1）布隆过滤器**

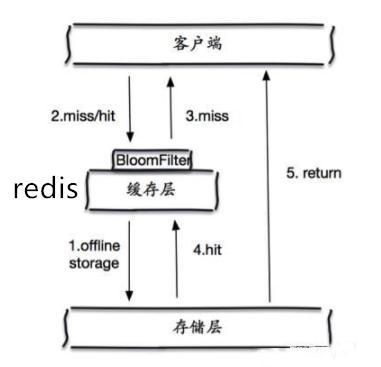
布隆过滤器是一种数据结构，垃圾网站和正常网站加起来全世界据统计也有几十亿个。网警要过滤这些垃圾网站，总不能到数据库里面一个一个去比较吧，这就可以使用布隆过滤器。假设我们存储一亿个垃圾网站地址。

可以先有一亿个二进制比特，然后网警用八个不同的随机数产生器（F1,F2, …,F8） 产生八个信息指纹（f1, f2, …, f8）。接下来用一个随机数产生器 G 把这八个信息指纹映射到 1 到1亿中的八个自然数 g1, g2, …,g8。最后把这八个位置的二进制全部设置为一。过程如下：



有一天网警查到了一个可疑的网站，想判断一下是否是XX网站，首先将可疑网站通过哈希映射到1亿个比特数组上的8个点。如果8个点的其中有一个点不为1，则可以判断该元素一定不存在集合中。

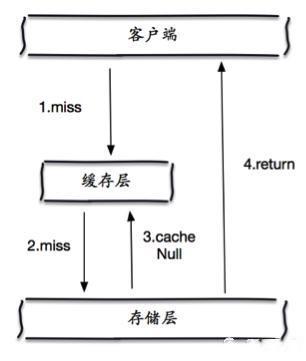
那这个布隆过滤器是如何解决redis中的缓存穿透呢？很简单首先也是对所有可能查询的参数以hash形式存储，当用户想要查询的时候，使用布隆过滤器发现不在集合中，就直接丢弃，不再对持久层查询。



这个形式很简单。

**2、缓存空对象**

当存储层不命中后，即使返回的空对象也将其缓存起来，同时会设置一个过期时间，之后再访问这个数据将会从缓存中获取，保护了后端数据源；

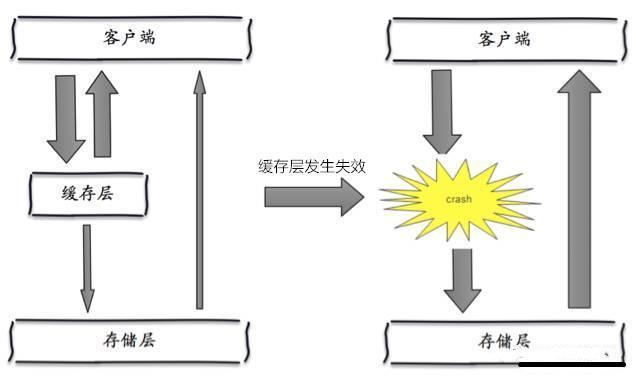


但是这种方法会存在两个问题：

如果空值能够被缓存起来，这就意味着缓存需要更多的空间存储更多的键，因为这当中可能会有很多的空值的键；即使对空值设置了过期时间，还是会存在缓存层和存储层的数据会有一段时间窗口的不一致，这对于需要保持一致性的业务会有影响。**二、缓存雪崩**

1、概念

缓存雪崩是指，缓存层出现了错误，不能正常工作了。于是所有的请求都会达到存储层，存储层的调用量会暴增，造成存储层也会挂掉的情况。



**2、解决方案**

**（1）redis高可用**

这个思想的含义是，既然redis有可能挂掉，那我多增设几台redis，这样一台挂掉之后其他的还可以继续工作，其实就是搭建的集群。

**（2）限流降级**

这个解决方案的思想是，在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个key只允许一个线程查询数据和写缓存，其他线程等待。

**（3）数据预热**

数据加热的含义就是在正式部署之前，我先把可能的数据先预先访问一遍，这样部分可能大量访问的数据就会加载到缓存中。在即将发生大并发访问前手动触发加载缓存不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀。

OK，基本上讲这块的文章几乎一样，觉得很不错，就在此基础之上修改了一下。