**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。

**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。

内存数据库

**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。

# Redis 安装

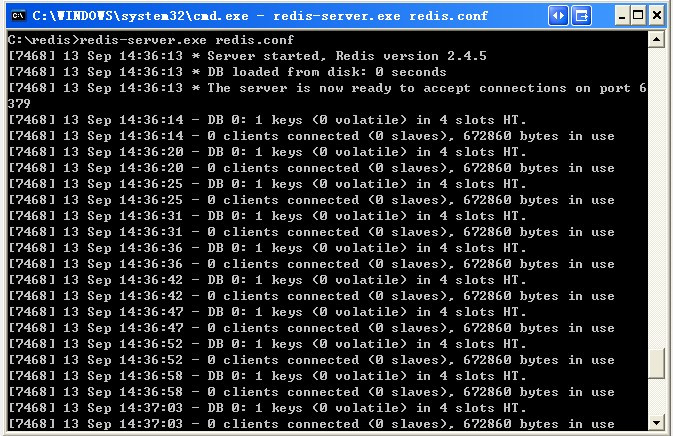
## Window 下安装

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

下载到的Redis支持32bit和64bit。根据自己实际情况选择，将64bit的内容cp到自定义盘符安装目录取名redis。 如 C:\reids

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.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.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：

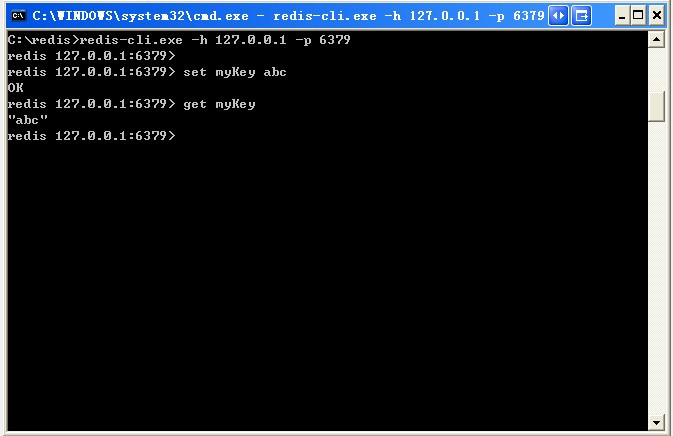


这时候另启一个cmd窗口，原来的不要关闭，不然就无法访问服务端了。

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**



## 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"

## 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>

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。

# Redis 配置

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

你可以通过 **CONFIG** 命令查看或设置配置项。

### 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：

### 实例

redis 127.0.0.1:6379> CONFIG GET \*

1) "dbfilename"

2) "dump.rdb"

3) "requirepass"

4) ""

5) "masterauth"

6) ""

7) "unixsocket"

8) ""

9) "logfile"

10) ""

11) "pidfile"

12) "/var/run/redis.pid"

13) "maxmemory"

14) "0"

15) "maxmemory-samples"

16) "3"

17) "timeout"

18) "0"

19) "tcp-keepalive"

20) "0"

21) "auto-aof-rewrite-percentage"

22) "100"

23) "auto-aof-rewrite-min-size"

24) "67108864"

25) "hash-max-ziplist-entries"

26) "512"

27) "hash-max-ziplist-value"

28) "64"

29) "list-max-ziplist-entries"

30) "512"

31) "list-max-ziplist-value"

32) "64"

33) "set-max-intset-entries"

34) "512"

35) "zset-max-ziplist-entries"

36) "128"

37) "zset-max-ziplist-value"

38) "64"

39) "hll-sparse-max-bytes"

40) "3000"

41) "lua-time-limit"

42) "5000"

43) "slowlog-log-slower-than"

44) "10000"

45) "latency-monitor-threshold"

46) "0"

47) "slowlog-max-len"

48) "128"

49) "port"

50) "6379"

51) "tcp-backlog"

52) "511"

53) "databases"

54) "16"

55) "repl-ping-slave-period"

56) "10"

57) "repl-timeout"

58) "60"

59) "repl-backlog-size"

60) "1048576"

61) "repl-backlog-ttl"

62) "3600"

63) "maxclients"

64) "4064"

65) "watchdog-period"

66) "0"

67) "slave-priority"

68) "100"

69) "min-slaves-to-write"

70) "0"

71) "min-slaves-max-lag"

72) "10"

73) "hz"

74) "10"

75) "no-appendfsync-on-rewrite"

76) "no"

77) "slave-serve-stale-data"

78) "yes"

79) "slave-read-only"

80) "yes"

81) "stop-writes-on-bgsave-error"

82) "yes"

83) "daemonize"

84) "no"

85) "rdbcompression"

86) "yes"

87) "rdbchecksum"

88) "yes"

89) "activerehashing"

90) "yes"

91) "repl-disable-tcp-nodelay"

92) "no"

93) "aof-rewrite-incremental-fsync"

94) "yes"

95) "appendonly"

96) "no"

97) "dir"

98) "/home/deepak/Downloads/redis-2.8.13/src"

99) "maxmemory-policy"

100) "volatile-lru"

101) "appendfsync"

102) "everysec"

103) "save"

104) "3600 1 300 100 60 10000"

105) "loglevel"

106) "notice"

107) "client-output-buffer-limit"

108) "normal 0 0 0 slave 268435456 67108864 60 pubsub 33554432 8388608 60"

109) "unixsocketperm"

110) "0"

111) "slaveof"

112) ""

113) "notify-keyspace-events"

114) ""

115) "bind"

116) ""

## 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：

1. 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 no**

2. 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

**pidfile /var/run/redis.pid**

3. 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

**port 6379**

4. 绑定的主机地址

**bind 127.0.0.1**

5.当 客户端闲置多长时间后关闭连接，如果指定为0，表示关闭该功能

**timeout 300**

6. 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

**loglevel verbose**

7. 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 stdout**

8. 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**

9. 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

**save <seconds> <changes>**

    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. 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**

11. 指定本地数据库文件名，默认值为dump.rdb

**dbfilename dump.rdb**

12. 指定本地数据库存放目录

**dir ./**

13. 设置当本机为slav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>**

14. 当master服务设置了密码保护时，slav服务连接master的密码

**masterauth <master-password>**

15. 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

**requirepass foobared**

16. 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

**maxclients 128**

17. 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

**maxmemory <bytes>**

18. 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

**appendonly no**

19. 指定更新日志文件名，默认为appendonly.aof

**appendfilename appendonly.aof**

20. 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（折衷，默认值）

**appendfsync everysec**

21. 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）

**vm-enabled no**

22. 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**

23. 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**

24. 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

**vm-page-size 32**

25. 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**

26. 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**

27. 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**

28. 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

**hash-max-zipmap-entries 64**

**hash-max-zipmap-value 512**

29. 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）

**activerehashing yes**

30. 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**

# 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）

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。

### 实例

redis 127.0.0.1:6379> SET name "runoob"

OK

redis 127.0.0.1:6379> GET name

"runoob"

在以上实例中我们使用了 Redis 的 **SET** 和 **GET** 命令。键为 name，对应的值为 **runoob**。

**注意：**一个键最大能存储512MB。

## Hash（哈希）

Redis hash 是一个键值对集合。

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。

### 实例

127.0.0.1:6379> HMSET user:1 username runoob password runoob points 200

OK

127.0.0.1:6379> HGETALL user:1

1) "username"

2) "runoob"

3) "password"

4) "runoob"

5) "points"

6) "200"

以上实例中 hash 数据类型存储了包含用户脚本信息的用户对象。 实例中我们使用了 Redis **HMSET, HGETALL** 命令，**user:1** 为键值。

每个 hash 可以存储 232 -1 键值对（40多亿）。

## 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）。

### 实例

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"

redis 127.0.0.1:6379>

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

## 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)。

### sadd 命令

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,key对应的set不存在返回错误。

sadd key member

### 实例

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) "rabitmq"

2) "mongodb"

3) "redis"

**注意：**以上实例中 rabitmq 添加了两次，但根据集合内元素的唯一性，第二次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)。

## zset(sorted set：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。

### zadd 命令

添加元素到集合，元素在集合中存在则更新对应score

zadd key score member

### 实例

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) "redis"

2) "mongodb"

3) "rabitmq"

# Redis 命令

Redis 命令用于在 redis 服务上执行操作。

要在 redis 服务上执行命令需要一个 redis 客户端。Redis 客户端在我们之前下载的的 redis 的安装包中。

### 语法

Redis 客户端的基本语法为：

$ redis-cli

### 实例

以下实例讲解了如何启动 redis 客户端：

启动 redis 客户端，打开终端并输入命令 **redis-cli**。该命令会连接本地的 redis 服务。

$redis-cli

redis 127.0.0.1:6379>

redis 127.0.0.1:6379> PING

PONG

在以上实例中我们连接到本地的 redis 服务并执行 **PING** 命令，该命令用于检测 redis 服务是否启动。

## 在远程服务上执行命令

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

### 实例

以下实例演示了如何连接到主机为 127.0.0.1，端口为 6379 ，密码为 mypass 的 redis 服务上。

$redis-cli -h 127.0.0.1 -p 6379 -a "mypass"

redis 127.0.0.1:6379>

redis 127.0.0.1:6379> PING

PONG

# Redis 键(key)

Redis 键命令用于管理 redis 的键。

### 语法

Redis 键命令的基本语法如下：

redis 127.0.0.1:6379> COMMAND KEY\_NAME

### 实例

redis 127.0.0.1:6379> SET runoobkey redis

OK

redis 127.0.0.1:6379> DEL runoobkey

(integer) 1

在以上实例中 **DEL** 是一个命令， **runoobkey** 是一个键。 如果键被删除成功，命令执行后输出 **(integer) 1**，否则将输出 **(integer) 0**

## Redis keys 命令

下表给出了与 Redis 键相关的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [DEL key](http://www.runoob.com/redis/keys-del.html) 该命令用于在 key 存在时删除 key。 |
| 2 | [DUMP key](http://www.runoob.com/redis/keys-dump.html)  序列化给定 key ，并返回被序列化的值。 |
| 3 | [EXISTS key](http://www.runoob.com/redis/keys-exists.html)  检查给定 key 是否存在。 |
| 4 | [EXPIRE key](http://www.runoob.com/redis/keys-expire.html) seconds 为给定 key 设置过期时间。 |
| 5 | [EXPIREAT key timestamp](http://www.runoob.com/redis/keys-expireat.html)  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)。 |
| 6 | [PEXPIRE key milliseconds](http://www.runoob.com/redis/keys-pexpire.html)  设置 key 的过期时间以毫秒计。 |
| 7 | [PEXPIREAT key milliseconds-timestamp](http://www.runoob.com/redis/keys-pexpireat.html)  设置 key 过期时间的时间戳(unix timestamp) 以毫秒计 |
| 8 | [KEYS pattern](http://www.runoob.com/redis/keys-keys.html)  查找所有符合给定模式( pattern)的 key 。 |
| 9 | [MOVE key db](http://www.runoob.com/redis/keys-move.html)  将当前数据库的 key 移动到给定的数据库 db 当中。 |
| 10 | [PERSIST key](http://www.runoob.com/redis/keys-persist.html)  移除 key 的过期时间，key 将持久保持。 |
| 11 | [PTTL key](http://www.runoob.com/redis/keys-pttl.html)  以毫秒为单位返回 key 的剩余的过期时间。 |
| 12 | [TTL key](http://www.runoob.com/redis/keys-ttl.html)  以秒为单位，返回给定 key 的剩余生存时间(TTL, time to live)。 |
| 13 | [RANDOMKEY](http://www.runoob.com/redis/keys-randomkey.html)  从当前数据库中随机返回一个 key 。 |
| 14 | [RENAME key newkey](http://www.runoob.com/redis/keys-rename.html)  修改 key 的名称 |
| 15 | [RENAMENX key newkey](http://www.runoob.com/redis/keys-renamenx.html)  仅当 newkey 不存在时，将 key 改名为 newkey 。 |
| 16 | [TYPE key](http://www.runoob.com/redis/keys-type.html)  返回 key 所储存的值的类型。 |

# Redis 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：

### 语法

redis 127.0.0.1:6379> COMMAND KEY\_NAME

### 实例

redis 127.0.0.1:6379> SET runoobkey redis

OK

redis 127.0.0.1:6379> GET runoobkey

"redis"

在以上实例中我们使用了 **SET** 和 **GET** 命令，键为 **runoobkey**。

## Redis 字符串命令

下表列出了常用的 redis 字符串命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [SET key value](http://www.runoob.com/redis/strings-set.html)  设置指定 key 的值 |
| 2 | [GET key](http://www.runoob.com/redis/strings-get.html)  获取指定 key 的值。 |
| 3 | [GETRANGE key start end](http://www.runoob.com/redis/strings-getrange.html)  返回 key 中字符串值的子字符 |
| 4 | [GETSET key value](http://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](http://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..]](http://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](http://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](http://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](http://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](http://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](http://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 ...]](http://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 ...]](http://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](http://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](http://www.runoob.com/redis/strings-incr.html) 将 key 中储存的数字值增一。 |
| 16 | [INCRBY key increment](http://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](http://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](http://www.runoob.com/redis/strings-decr.html) 将 key 中储存的数字值减一。 |
| 19 | [DECRBY key decrement](http://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](http://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 value 追加到 key 原来的值的末尾。 |

# Redis 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）。

### 实例

127.0.0.1:6379> HMSET runoobkey name "redis tutorial" description "redis basic commands for caching" likes 20 visitors 23000

OK

127.0.0.1:6379> HGETALL runoobkey

1) "name"

2) "redis tutorial"

3) "description"

4) "redis basic commands for caching"

5) "likes"

6) "20"

7) "visitors"

8) "23000"

在以上实例中，我们设置了 redis 的一些描述信息(name, description, likes, visitors) 到哈希表的 **runoobkey** 中。

## Redis hash 命令

下表列出了 redis hash 基本的相关命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [HDEL key field2 [field2]](http://www.runoob.com/redis/hashes-hdel.html)  删除一个或多个哈希表字段 |
| 2 | [HEXISTS key field](http://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](http://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](http://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](http://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](http://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](http://www.runoob.com/redis/hashes-hkeys.html)  获取所有哈希表中的字段 |
| 8 | [HLEN key](http://www.runoob.com/redis/hashes-hlen.html)  获取哈希表中字段的数量 |
| 9 | [HMGET key field1 [field2]](http://www.runoob.com/redis/hashes-hmget.html)  获取所有给定字段的值 |
| 10 | [HMSET key field1 value1 [field2 value2 ]](http://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](http://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](http://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](http://www.runoob.com/redis/hashes-hvals.html)  获取哈希表中所有值 |
| 14 | HSCAN key cursor [MATCH pattern] [COUNT count]  迭代哈希表中的键值对。 |

# Redis 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导列表的头部（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)。

### 实例

redis 127.0.0.1:6379> LPUSH runoobkey redis

(integer) 1

redis 127.0.0.1:6379> LPUSH runoobkey mongodb

(integer) 2

redis 127.0.0.1:6379> LPUSH runoobkey mysql

(integer) 3

redis 127.0.0.1:6379> LRANGE runoobkey 0 10

1) "mysql"

2) "mongodb"

3) "redis"

在以上实例中我们使用了 **LPUSH** 将三个值插入了名为 **runoobkey** 的列表当中。

### Redis 列表命令

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](http://www.runoob.com/redis/lists-blpop.html)  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](http://www.runoob.com/redis/lists-brpop.html)  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](http://www.runoob.com/redis/lists-brpoplpush.html)  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](http://www.runoob.com/redis/lists-lindex.html)  通过索引获取列表中的元素 |
| 5 | [LINSERT key BEFORE|AFTER pivot value](http://www.runoob.com/redis/lists-linsert.html)  在列表的元素前或者后插入元素 |
| 6 | [LLEN key](http://www.runoob.com/redis/lists-llen.html)  获取列表长度 |
| 7 | [LPOP key](http://www.runoob.com/redis/lists-lpop.html)  移出并获取列表的第一个元素 |
| 8 | [LPUSH key value1 [value2]](http://www.runoob.com/redis/lists-lpush.html)  将一个或多个值插入到列表头部 |
| 9 | [LPUSHX key value](http://www.runoob.com/redis/lists-lpushx.html)  将一个或多个值插入到已存在的列表头部 |
| 10 | [LRANGE key start stop](http://www.runoob.com/redis/lists-lrange.html)  获取列表指定范围内的元素 |
| 11 | [LREM key count value](http://www.runoob.com/redis/lists-lrem.html)  移除列表元素 |
| 12 | [LSET key index value](http://www.runoob.com/redis/lists-lset.html)  通过索引设置列表元素的值 |
| 13 | [LTRIM key start stop](http://www.runoob.com/redis/lists-ltrim.html)  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](http://www.runoob.com/redis/lists-rpop.html)  移除并获取列表最后一个元素 |
| 15 | [RPOPLPUSH source destination](http://www.runoob.com/redis/lists-rpoplpush.html)  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]](http://www.runoob.com/redis/lists-rpush.html)  在列表中添加一个或多个值 |
| 17 | [RPUSHX key value](http://www.runoob.com/redis/lists-rpushx.html)  为已存在的列表添加值 |

# Redis 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)。

### 实例

redis 127.0.0.1:6379> SADD runoobkey redis

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mongodb

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mysql

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mysql

(integer) 0

redis 127.0.0.1:6379> SMEMBERS runoobkey

1) "mysql"

2) "mongodb"

3) "redis"

在以上实例中我们通过 **SADD** 命令向名为 **runoobkey** 的集合插入的三个元素。

## Redis 集合命令

下表列出了 Redis 集合基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [SADD key member1 [member2]](http://www.runoob.com/redis/sets-sadd.html)  向集合添加一个或多个成员 |
| 2 | [SCARD key](http://www.runoob.com/redis/sets-scard.html)  获取集合的成员数 |
| 3 | [SDIFF key1 [key2]](http://www.runoob.com/redis/sets-sdiff.html)  返回给定所有集合的差集 |
| 4 | [SDIFFSTORE destination key1 [key2]](http://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]](http://www.runoob.com/redis/sets-sinter.html)  返回给定所有集合的交集 |
| 6 | [SINTERSTORE destination key1 [key2]](http://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](http://www.runoob.com/redis/sets-sismember.html)  判断 member 元素是否是集合 key 的成员 |
| 8 | [SMEMBERS key](http://www.runoob.com/redis/sets-smembers.html)  返回集合中的所有成员 |
| 9 | [SMOVE source destination member](http://www.runoob.com/redis/sets-smove.html)  将 member 元素从 source 集合移动到 destination 集合 |
| 10 | [SPOP key](http://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]](http://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]](http://www.runoob.com/redis/sets-srem.html)  移除集合中一个或多个成员 |
| 13 | [SUNION key1 [key2]](http://www.runoob.com/redis/sets-sunion.html)  返回所有给定集合的并集 |
| 14 | [SUNIONSTORE destination key1 [key2]](http://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]](http://www.runoob.com/redis/sets-sscan.html)  迭代集合中的元素 |

**Redis 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 2^64 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。

**实例**

以下实例演示了 HyperLogLog 的工作过程：

redis 127.0.0.1:6379> PFADD runoobkey "redis"

1) (integer) 1

redis 127.0.0.1:6379> PFADD runoobkey "mongodb"

1) (integer) 1

redis 127.0.0.1:6379> PFADD runoobkey "mysql"

1) (integer) 1

redis 127.0.0.1:6379> PFCOUNT runoobkey

(integer) 3

**Redis HyperLogLog 命令**

下表列出了 redis HyperLogLog 的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [PFADD key element [element ...]](http://www.runoob.com/redis/hyperloglog-pfadd.html)  添加指定元素到 HyperLogLog 中。 |
| 2 | [PFCOUNT key [key ...]](http://www.runoob.com/redis/hyperloglog-pfcount.html)  返回给定 HyperLogLog 的基数估算值。 |
| 3 | [PFMERGE destkey sourcekey [sourcekey ...]](http://www.runoob.com/redis/hyperloglog-pfmerge.html)  将多个 HyperLogLog 合并为一个 HyperLogLog |

**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"

1) "message"

2) "redisChat"

3) "Learn redis by runoob.com"

**Redis 发布订阅命令**

下表列出了 redis 发布订阅常用命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [PSUBSCRIBE pattern [pattern ...]](http://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 ...]]](http://www.runoob.com/redis/pub-sub-pubsub.html)  查看订阅与发布系统状态。 |
| 3 | [PUBLISH channel message](http://www.runoob.com/redis/pub-sub-publish.html)  将信息发送到指定的频道。 |
| 4 | [PUNSUBSCRIBE [pattern [pattern ...]]](http://www.runoob.com/redis/pub-sub-punsubscribe.html)  退订所有给定模式的频道。 |
| 5 | [SUBSCRIBE channel [channel ...]](http://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 ...]]](http://www.runoob.com/redis/pub-sub-unsubscribe.html)  指退订给定的频道。 |

**Redis 事务**

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。

**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"

**Redis 事务命令**

下表列出了 redis 事务的相关命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [DISCARD](http://www.runoob.com/redis/transactions-discard.html)  取消事务，放弃执行事务块内的所有命令。 |
| 2 | [EXEC](http://www.runoob.com/redis/transactions-exec.html)  执行所有事务块内的命令。 |
| 3 | [MULTI](http://www.runoob.com/redis/transactions-multi.html)  标记一个事务块的开始。 |
| 4 | [UNWATCH](http://www.runoob.com/redis/transactions-unwatch.html)  取消 WATCH 命令对所有 key 的监视。 |
| 5 | [WATCH key [key ...]](http://www.runoob.com/redis/transactions-watch.html)  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。 |

# Redis 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**。

### 语法

Eval 命令的基本语法如下：

redis 127.0.0.1:6379> EVAL script numkeys key [key ...] arg [arg ...]

### 实例

以下实例演示了 redis 脚本工作过程：

redis 127.0.0.1:6379> EVAL "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}" 2 key1 key2 first second

1) "key1"

2) "key2"

3) "first"

4) "second"

## Redis 脚本命令

下表列出了 redis 脚本常用命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [EVAL script numkeys key [key ...] arg [arg ...]](http://www.runoob.com/redis/scripting-eval.html)  执行 Lua 脚本。 |
| 2 | [EVALSHA sha1 numkeys key [key ...] arg [arg ...]](http://www.runoob.com/redis/scripting-evalsha.html)  执行 Lua 脚本。 |
| 3 | [SCRIPT EXISTS script [script ...]](http://www.runoob.com/redis/scripting-script-exists.html)  查看指定的脚本是否已经被保存在缓存当中。 |
| 4 | [SCRIPT FLUSH](http://www.runoob.com/redis/scripting-script-flush.html)  从脚本缓存中移除所有脚本。 |
| 5 | [SCRIPT KILL](http://www.runoob.com/redis/scripting-script-kill.html)  杀死当前正在运行的 Lua 脚本。 |
| 6 | [SCRIPT LOAD script](http://www.runoob.com/redis/scripting-script-load.html)  将脚本 script 添加到脚本缓存中，但并不立即执行这个脚本。 |

# Redis 连接

Redis 连接命令主要是用于连接 redis 服务。

### 实例

以下实例演示了客户端如何通过密码验证连接到 redis 服务，并检测服务是否在运行：

redis 127.0.0.1:6379> AUTH "password"

OK

redis 127.0.0.1:6379> PING

PONG

## Redis 连接命令

下表列出了 redis 连接的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [AUTH password](http://www.runoob.com/redis/connection-auth.html)  验证密码是否正确 |
| 2 | [ECHO message](http://www.runoob.com/redis/connection-echo.html)  打印字符串 |
| 3 | [PING](http://www.runoob.com/redis/connection-ping.html)  查看服务是否运行 |
| 4 | [QUIT](http://www.runoob.com/redis/connection-quit.html)  关闭当前连接 |
| 5 | [SELECT index](http://www.runoob.com/redis/connection-select.html)  切换到指定的数据库 |

# Redis 服务器

Redis 服务器命令主要是用于管理 redis 服务。

### 实例

以下实例演示了如何获取 redis 服务器的统计信息：

redis 127.0.0.1:6379> INFO

# Server

redis\_version:2.8.13

redis\_git\_sha1:00000000

redis\_git\_dirty:0

redis\_build\_id:c2238b38b1edb0e2

redis\_mode:standalone

os:Linux 3.5.0-48-generic x86\_64

arch\_bits:64

multiplexing\_api:epoll

gcc\_version:4.7.2

process\_id:3856

run\_id:0e61abd297771de3fe812a3c21027732ac9f41fe

tcp\_port:6379

uptime\_in\_seconds:11554

uptime\_in\_days:0

hz:10

lru\_clock:16651447

config\_file:

# Clients

connected\_clients:1

client-longest\_output\_list:0

client-biggest\_input\_buf:0

blocked\_clients:0

# Memory

used\_memory:589016

used\_memory\_human:575.21K

used\_memory\_rss:2461696

used\_memory\_peak:667312

used\_memory\_peak\_human:651.67K

used\_memory\_lua:33792

mem\_fragmentation\_ratio:4.18

mem\_allocator:jemalloc-3.6.0

# Persistence

loading:0

rdb\_changes\_since\_last\_save:3

rdb\_bgsave\_in\_progress:0

rdb\_last\_save\_time:1409158561

rdb\_last\_bgsave\_status:ok

rdb\_last\_bgsave\_time\_sec:0

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:24

total\_commands\_processed:294

instantaneous\_ops\_per\_sec:0

rejected\_connections:0

sync\_full:0

sync\_partial\_ok:0

sync\_partial\_err:0

expired\_keys:0

evicted\_keys:0

keyspace\_hits:41

keyspace\_misses:82

pubsub\_channels:0

pubsub\_patterns:0

latest\_fork\_usec:264

# 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:10.49

used\_cpu\_user:4.96

used\_cpu\_sys\_children:0.00

used\_cpu\_user\_children:0.01

# Keyspace

db0:keys=94,expires=1,avg\_ttl=41638810

db1:keys=1,expires=0,avg\_ttl=0

db3:keys=1,expires=0,avg\_ttl=0

## Redis 服务器命令

下表列出了 redis 服务器的相关命令:

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [BGREWRITEAOF](http://www.runoob.com/redis/server-bgrewriteaof.html)  异步执行一个 AOF（AppendOnly File） 文件重写操作 |
| 2 | [BGSAVE](http://www.runoob.com/redis/server-bgsave.html)  在后台异步保存当前数据库的数据到磁盘 |
| 3 | [CLIENT KILL [ip:port] [ID client-id]](http://www.runoob.com/redis/server-client-kill.html)  关闭客户端连接 |
| 4 | [CLIENT LIST](http://www.runoob.com/redis/server-client-list.html)  获取连接到服务器的客户端连接列表 |
| 5 | [CLIENT GETNAME](http://www.runoob.com/redis/server-client-getname.html)  获取连接的名称 |
| 6 | [CLIENT PAUSE timeout](http://www.runoob.com/redis/server-client-pause.html)  在指定时间内终止运行来自客户端的命令 |
| 7 | [CLIENT SETNAME connection-name](http://www.runoob.com/redis/server-client-setname.html)  设置当前连接的名称 |
| 8 | [CLUSTER SLOTS](http://www.runoob.com/redis/server-cluster-slots.html)  获取集群节点的映射数组 |
| 9 | [COMMAND](http://www.runoob.com/redis/server-command.html)  获取 Redis 命令详情数组 |
| 10 | [COMMAND COUNT](http://www.runoob.com/redis/server-command-count.html)  获取 Redis 命令总数 |
| 11 | [COMMAND GETKEYS](http://www.runoob.com/redis/server-command-getkeys.html)  获取给定命令的所有键 |
| 12 | [TIME](http://www.runoob.com/redis/server-time.html)  返回当前服务器时间 |
| 13 | [COMMAND INFO command-name [command-name ...]](http://www.runoob.com/redis/server-command-info.html)  获取指定 Redis 命令描述的数组 |
| 14 | [CONFIG GET parameter](http://www.runoob.com/redis/server-config-get.html)  获取指定配置参数的值 |
| 15 | [CONFIG REWRITE](http://www.runoob.com/redis/server-config-rewrite.html)  对启动 Redis 服务器时所指定的 redis.conf 配置文件进行改写 |
| 16 | [CONFIG SET parameter value](http://www.runoob.com/redis/server-config-set.html)  修改 redis 配置参数，无需重启 |
| 17 | [CONFIG RESETSTAT](http://www.runoob.com/redis/server-config-resetstat.html)  重置 INFO 命令中的某些统计数据 |
| 18 | [DBSIZE](http://www.runoob.com/redis/server-dbsize.html)  返回当前数据库的 key 的数量 |
| 19 | [DEBUG OBJECT key](http://www.runoob.com/redis/server-debug-object.html)  获取 key 的调试信息 |
| 20 | [DEBUG SEGFAULT](http://www.runoob.com/redis/server-debug-segfault.html)  让 Redis 服务崩溃 |
| 21 | [FLUSHALL](http://www.runoob.com/redis/server-flushall.html)  删除所有数据库的所有key |
| 22 | [FLUSHDB](http://www.runoob.com/redis/server-flushdb.html)  删除当前数据库的所有key |
| 23 | [INFO [section]](http://www.runoob.com/redis/server-info.html)  获取 Redis 服务器的各种信息和统计数值 |
| 24 | [LASTSAVE](http://www.runoob.com/redis/server-lastsave.html)  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 |
| 25 | [MONITOR](http://www.runoob.com/redis/server-monitor.html)  实时打印出 Redis 服务器接收到的命令，调试用 |
| 26 | [ROLE](http://www.runoob.com/redis/server-role.html)  返回主从实例所属的角色 |
| 27 | [SAVE](http://www.runoob.com/redis/server-save.html)  异步保存数据到硬盘 |
| 28 | [SHUTDOWN [NOSAVE] [SAVE]](http://www.runoob.com/redis/server-shutdown.html)  异步保存数据到硬盘，并关闭服务器 |
| 29 | [SLAVEOF host port](http://www.runoob.com/redis/server-slaveof.html)  将当前服务器转变为指定服务器的从属服务器(slave server) |
| 30 | [SLOWLOG subcommand [argument]](http://www.runoob.com/redis/server-showlog.html)  管理 redis 的慢日志 |
| 31 | [SYNC](http://www.runoob.com/redis/server-sync.html)  用于复制功能(replication)的内部命令 |

# Redis 数据备份与恢复

Redis **SAVE** 命令用于创建当前数据库的备份。

### 语法

redis Save 命令基本语法如下：

redis 127.0.0.1:6379> SAVE

### 实例

redis 127.0.0.1:6379> SAVE

OK

该命令将在 redis 安装目录中创建dump.rdb文件。

## 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) "/usr/local/redis/bin"

以上命令 **CONFIG GET dir** 输出的 redis 安装目录为 /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。

### 实例

127.0.0.1:6379> BGSAVE

Background saving started

# 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。

### 实例

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"

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。

### 语法

**AUTH** 命令基本语法格式如下：

127.0.0.1:6379> AUTH password

### 实例

127.0.0.1:6379> AUTH "runoob"

OK

127.0.0.1:6379> SET mykey "Test value"

OK

127.0.0.1:6379> GET mykey

"Test value"

# Redis 性能测试

Redis 性能测试是通过同时执行多个命令实现的。

### 语法

redis 性能测试的基本命令如下：

redis-benchmark [option] [option value]

### 实例

以下实例同时执行 10000 个请求来检测性能：

redis-benchmark -n 10000

PING\_INLINE: 141043.72 requests per second

PING\_BULK: 142857.14 requests per second

SET: 141442.72 requests per second

GET: 145348.83 requests per second

INCR: 137362.64 requests per second

LPUSH: 145348.83 requests per second

LPOP: 146198.83 requests per second

SADD: 146198.83 requests per second

SPOP: 149253.73 requests per second

LPUSH (needed to benchmark LRANGE): 148588.42 requests per second

LRANGE\_100 (first 100 elements): 58411.21 requests per second

LRANGE\_300 (first 300 elements): 21195.42 requests per second

LRANGE\_500 (first 450 elements): 14539.11 requests per second

LRANGE\_600 (first 600 elements): 10504.20 requests per second

MSET (10 keys): 93283.58 requests per second

redis 性能测试工具可选参数如下所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **选项** | **描述** | **默认值** |
| 1 | **-h** | 指定服务器主机名 | 127.0.0.1 |
| 2 | **-p** | 指定服务器端口 | 6379 |
| 3 | **-s** | 指定服务器 socket |  |
| 4 | **-c** | 指定并发连接数 | 50 |
| 5 | **-n** | 指定请求数 | 10000 |
| 6 | **-d** | 以字节的形式指定 SET/GET 值的数据大小 | 2 |
| 7 | **-k** | 1=keep alive 0=reconnect | 1 |
| 8 | **-r** | SET/GET/INCR 使用随机 key, SADD 使用随机值 |  |
| 9 | **-P** | 通过管道传输 <numreq> 请求 | 1 |
| 10 | **-q** | 强制退出 redis。仅显示 query/sec 值 |  |
| 11 | **--csv** | 以 CSV 格式输出 |  |
| 12 | **-l** | 生成循环，永久执行测试 |  |
| 13 | **-t** | 仅运行以逗号分隔的测试命令列表。 |  |
| 14 | **-I** | Idle 模式。仅打开 N 个 idle 连接并等待。 |  |

### 实例

以下实例我们使用了多个参数来测试 redis 性能：

redis-benchmark -h 127.0.0.1 -p 6379 -t set,lpush -n 10000 -q

SET: 146198.83 requests per second

LPUSH: 145560.41 requests per second

以上实例中主机为 127.0.0.1，端口号为 6379，执行的命令为 set,lpush，请求数为 10000，通过 -q 参数让结果只显示每秒执行的请求数。

# Redis 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

## 最大连接数

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 中对这个值进行修改。

config get maxclients

1) "maxclients"

2) "10000"

### 实例

以下实例我们在服务启动时设置最大连接数为 100000：

redis-server --maxclients 100000

## 客户端命令

|  |  |  |
| --- | --- | --- |
| **S.N.** | **命令** | **描述** |
| 1 | **CLIENT LIST** | 返回连接到 redis 服务的客户端列表 |
| 2 | **CLIENT SETNAME** | 设置当前连接的名称 |
| 3 | **CLIENT GETNAME** | 获取通过 CLIENT SETNAME 命令设置的服务名称 |
| 4 | **CLIENT PAUSE** | 挂起客户端连接，指定挂起的时间以毫秒计 |
| 5 | **CLIENT KILL** | 关闭客户端连接 |

# 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

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。

### 实例

查看 redis 管道，只需要启动 redis 实例并输入以下命令：

$(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

以上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

## 管道技术的优势

管道技术最显著的优势是提高了 redis 服务的性能。

### 一些测试数据

在下面的测试中，我们将使用Redis的Ruby客户端，支持管道技术特性，测试管道技术对速度的提升效果。

require 'rubygems'

require 'redis'

def bench(descr)

start = Time.now

yield

puts "#{descr} #{Time.now-start} seconds"

end

def without\_pipelining

r = Redis.new

10000.times {

r.ping

}

end

def with\_pipelining

r = Redis.new

r.pipelined {

10000.times {

r.ping

}

}

end

bench("without pipelining") {

without\_pipelining

}

bench("with pipelining") {

with\_pipelining

}

从处于局域网中的Mac OS X系统上执行上面这个简单脚本的数据表明，开启了管道操作后，往返时延已经被改善得相当低了。

without pipelining 1.185238 seconds

with pipelining 0.250783 seconds

如你所见，开启管道后，我们的速度效率提升了5倍。

# Redis 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。

### 分区的优势

* 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。
* 通过多核和多台计算机，允许我们扩展计算能力；通过多台计算机和网络适配器，允许我们扩展网络带宽。

### 分区的不足

redis的一些特性在分区方面表现的不是很好：

* 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。
* 涉及多个key的redis事务不能使用。
* 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。
* 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。

## 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。

### 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。

### 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。

**Java 使用 Redis**

**安装**

开始在 Java 中使用 Redis 前， 我们需要确保已经安装了 redis 服务及 Java redis 驱动，且你的机器上能正常使用 Java。 Java的安装配置可以参考我们的 [Java开发环境配置](http://www.runoob.com/redis/java-environment-setup.html)接下来让我们安装 Java redis 驱动：

* 首先你需要下载驱动包，[**下载 jedis.jar**](http://repo1.maven.org/maven2/redis/clients/jedis/2.1.0/jedis-2.1.0-sources.jar)，确保下载最新驱动包。
* 在你的classpath中包含该驱动包。

**连接到 redis 服务**

import redis.clients.jedis.Jedis;

public class RedisJava {

public static void main(String[] args) {

//连接本地的 Redis 服务

Jedis jedis = new Jedis("localhost");

System.out.println("Connection to server sucessfully");

//查看服务是否运行

System.out.println("Server is running: "+jedis.ping());

}

}

编译以上 Java 程序，确保驱动包的路径是正确的。

$javac RedisJava.java

$java RedisJava

Connection to server sucessfully

Server is running: PONG

Redis Java String Example

**Redis Java String(字符串) 实例**

import redis.clients.jedis.Jedis;

public class RedisStringJava {

public static void main(String[] args) {

//连接本地的 Redis 服务

Jedis jedis = new Jedis("localhost");

System.out.println("Connection to server sucessfully");

//设置 redis 字符串数据

jedis.set("runoobkey", "Redis tutorial");

// 获取存储的数据并输出

System.out.println("Stored string in redis:: "+ jedis.get("runoobkey"));

}

}

编译以上程序。

$javac RedisStringJava.java

$java RedisStringJava

Connection to server sucessfully

Stored string in redis:: Redis tutorial

**Redis Java List(列表) 实例**

import redis.clients.jedis.Jedis;

public class RedisListJava {

public static void main(String[] args) {

//连接本地的 Redis 服务

Jedis jedis = new Jedis("localhost");

System.out.println("Connection to server sucessfully");

//存储数据到列表中

jedis.lpush("tutorial-list", "Redis");

jedis.lpush("tutorial-list", "Mongodb");

jedis.lpush("tutorial-list", "Mysql");

// 获取存储的数据并输出

List<String> list = jedis.lrange("tutorial-list", 0 ,5);

for(int i=0; i<list.size(); i++) {

System.out.println("Stored string in redis:: "+list.get(i));

}

}

}

编译以上程序。

$javac RedisListJava.java

$java RedisListJava

Connection to server sucessfully

Stored string in redis:: Redis

Stored string in redis:: Mongodb

Stored string in redis:: Mysql

**Redis Java Keys 实例**

import redis.clients.jedis.Jedis;

public class RedisKeyJava {

public static void main(String[] args) {

//连接本地的 Redis 服务

Jedis jedis = new Jedis("localhost");

System.out.println("Connection to server sucessfully");

// 获取数据并输出

List<String> list = jedis.keys("\*");

for(int i=0; i<list.size(); i++) {

System.out.println("List of stored keys:: "+list.get(i));

}

}

}

编译以上程序。

$javac RedisKeyJava.java

$java RedisKeyJava

Connection to server sucessfully

List of stored keys:: tutorial-name

List of stored keys:: tutorial-list