# Reids命令

Redis命令十分丰富，包括的命令组有Cluster、Connection、Geo、Hashes、HyperLogLog、Keys、Lists、Pub/Sub、Scripting、Server、Sets、Sorted Sets、Strings、Transactions一共14个redis命令组两百多个redis命令，下面我们一个一个来介绍如何使用Redis命令。

## APPEND key value

起始版本：2.0.0

时间复杂度：O(1)。均摊时间复杂度是O(1)， 因为redis用的动态字符串的库在每次分配空间的时候会增加一倍的可用空闲空间，所以在添加的value较小而且已经存在的 value是任意大小的情况下，均摊时间复杂度是O(1) 。

如果 key 已经存在，并且值为字符串，那么这个命令会把 value 追加到原来值（value）的结尾。 如果 key 不存在，那么它将首先创建一个空字符串的key，再执行追加操作，这种情况 APPEND 将类似于 SET 操作。

返回值

Integer reply：返回append后字符串值（value）的长度。

例子

redis> EXISTS mykey

(integer) 0

redis> APPEND mykey "Hello"

(integer) 5

redis> APPEND mykey " World"

(integer) 11

redis> GET mykey

"Hello World"

redis>

模式：节拍序列(Time series)

APPEND 命令可以用来连接一系列固定长度的样例,与使用列表相比这样更加紧凑. 通常会用来记录节拍序列. 每收到一个新的节拍样例就可以这样记录:

APPEND timeseries "fixed-size sample"

在节拍序列里, 可以很容易地访问序列中的每个元素:

STRLEN 可以用来计算样例个数.

GETRANGE 允许随机访问序列中的各个元素. 如果序列中有明确的节拍信息, 在Redis 2.6中就可以使用GETRANGE配合Lua脚本来实现一个二分查找算法.

SETRANGE 可以用来覆写已有的节拍序列.

该模式的局限在于只能做追加操作. Redis目前缺少剪裁字符串的命令, 所以无法方便地把序列剪裁成指定的尺寸. 但是, 节拍序列在空间占用上效率极好.

小贴士: 在键值中组合Unix时间戳, 可以在构建一系列相关键值时缩短键值长度,更优雅地分配Redis实例.

使用定长字符串进行温度采样的例子(在实际使用时,采用二进制格式会更好).

redis> APPEND ts "0043"

(integer) 4

redis> APPEND ts "0035"

(integer) 8

redis> GETRANGE ts 0 3

"0043"

redis> GETRANGE ts 4 7

"0035"

redis>

## AUTH password

起始版本：1.0.0

为redis服务请求设置一个密码。redis可以设置在客户端执行commands请求前需要通过密码验证。通过修改配置文件的requirepass就可以设置密码。 如果密码与配置文件里面设置的密码一致，服务端就会发会一个OK的状态码，接受客户端发送其他的请求命令，否则服务端会返回一个错误码，客户端需要尝试使用新的密码来进行连接。

注意: 因为redis的高性能能在短时间接受非常多的尝试性密码，所以请务必设置一个足够复杂的密码以防止可能的攻击。

返回值

simple-string-reply

## BGREWRITEAOF

起始版本：1.0.0

Redis BGREWRITEAOF 命令用于异步执行一个 AOF（AppendOnly File）文件重写操作。重写会创建一个当前AOF文件的体积优化版本。

即使 BGREWRITEAOF 执行失败，也不会有任何数据丢失，因为旧的AOF文件在BGREWRITEAOF 成功之前不会被修改。

AOF 重写由 Redis 自行触发， BGREWRITEAOF仅仅用于手动触发重写操作。

具体内容:

如果一个子Redis是通过磁盘快照创建的，AOF重写将会在RDB终止后才开始保存。这种情况下BGREWRITEAOF任然会返回OK状态码。从Redis 2.6起你可以通过INFO命令查看AOF重写执行情况。

如果只在执行的AOF重写返回一个错误，AOF重写将会在稍后一点的时间重新调用。

从 Redis 2.4 开始，AOF重写由 Redis 自行触发，BGREWRITEAOF仅仅用于手动触发重写操作。

请参考Redis 持久化了解更多详细信息。

返回值

simple-string-reply: 总是返回 OK。

## BGSAVE

起始版本：1.0.0

后台保存DB。会立即返回 OK 状态码。 Redis forks, 父进程继续提供服务以供客户端调用，子进程将DB数据保存到磁盘然后退出。如果操作成功，可以通过客户端命令LASTSAVE来检查操作结果。

请参考Redis 持久化了解更多详细信息。

返回值

simple-string-reply

## BITCOUNT key [start end]

起始版本：2.6.0

时间复杂度：O(N)

统计字符串被设置为1的bit数.

一般情况下，给定的整个字符串都会被进行计数，通过指定额外的 start 或 end 参数，可以让计数只在特定的位上进行。

start 和 end 参数的设置和 GETRANGE 命令类似，都可以使用负数值：比如 -1 表示最后一个位，而 -2 表示倒数第二个位，以此类推。

不存在的 key 被当成是空字符串来处理，因此对一个不存在的 key 进行 BITCOUNT 操作，结果为 0 。

返回值

Integer reply

被设置为 1 的位的数量。

例子

redis> SET mykey "foobar"

OK

redis> BITCOUNT mykey

(integer) 26

redis> BITCOUNT mykey 0 0

(integer) 4

redis> BITCOUNT mykey 1 1

(integer) 6

redis>

模式：使用 bitmap 实现用户上线次数统计

Bitmap 对于一些特定类型的计算非常有效。

假设现在我们希望记录自己网站上的用户的上线频率，比如说，计算用户 A 上线了多少天，用户 B 上线了多少天，诸如此类，以此作为数据，从而决定让哪些用户参加 beta 测试等活动 —— 这个模式可以使用 SETBIT 和 BITCOUNT 来实现。

比如说，每当用户在某一天上线的时候，我们就使用 SETBIT ，以用户名作为 key ，将那天所代表的网站的上线日作为 offset 参数，并将这个 offset 上的为设置为 1 。

举个例子，如果今天是网站上线的第 100 天，而用户 peter 在今天阅览过网站，那么执行命令 SETBIT peter 100 1 ；如果明天 peter 也继续阅览网站，那么执行命令 SETBIT peter 101 1 ，以此类推。

当要计算 peter 总共以来的上线次数时，就使用 BITCOUNT 命令：执行 BITCOUNT peter ，得出的结果就是 peter 上线的总天数。

更详细的实现可以参考博文 Fast, easy, realtime metrics using Redis bitmaps (需要翻墙)

性能

前面的上线次数统计例子，即使运行 10 年，占用的空间也只是每个用户 10\*365 比特位(bit)，也即是每个用户 456 字节。对于这种大小的数据来说， BITCOUNT 的处理速度就像 GET 和 INCR 这种 O(1) 复杂度的操作一样快。

如果你的 bitmap 数据非常大，那么可以考虑使用以下两种方法：

将一个大的 bitmap 分散到不同的 key 中，作为小的 bitmap 来处理。使用 Lua 脚本可以很方便地完成这一工作。

使用 BITCOUNT 的 start 和 end 参数，每次只对所需的部分位进行计算，将位的累积工作(accumulating)放到客户端进行，并且对结果进行缓存 (caching)。