Redis 数据类型介绍

你也许已经知道Redis并不是简单的key-value存储，实际上他是一个数据结构服务器，支持不同类型的值。也就是说，你不必仅仅把字符串当作键所指向的值。下列这些数据类型都可作为值类型：

- 二进制安全的字符串
- Lists: 按插入顺序排序的字符串元素的集合。他们基本上就是*链表（linked lists）*。
- Sets: 不重复且无序的字符串元素的集合。
- Sorted sets,类似Sets,但是每个字符串元素都关联到一个叫*score*浮动数值（floating number value）。里面的元素总是通过score进行着排序，所以不同的是，它是可以检索的一系列元素。（例如你可能会问：给我前面10个或者后面10个元素）。
- Hashes,由field和关联的value组成的map。field和value都是字符串的。这和Ruby、Python的hashes很像。
- Bit arrays (或者说 simply bitmaps): 通过特殊的命令，你可以将 String 值当作一系列 bits 处理：可以设置和清除单独的 bits，数出所有设为 1 的 bits 的数量，找到最前的被设为 1 或 0 的 bit，等等。
- HyperLogLogs: 这是被用于估计一个 set 中元素数量的概率性的数据结构。别害怕，它比看起来的样子要简单…参见本教程的 HyperLogLog 部分。D

学习这些数据类型的原理，以及如何使用它们解决 command reference 中的特定问题，并不总是不关紧要的。所以，本文档是一个关于 Redis 数据类型和它们最常见特性的导论。 在所有的例子中，我们将使用 `redis-cli` 工具。它是一个简单而有用的命令行工具，用于向 Redis 服务器发出命令。

Redis keys

Redis key值是二进制安全的，这意味着可以用任何二进制序列作为key值，从形如”foo”的简单字符串到一个JPEG文件的内容都可以。空字符串也是有效key值。

关于key的几条规则：

- 太长的键值不是个好主意，例如1024字节的键值就不是个好主意，不仅因为消耗内存，而且在数据中查找这类键值的计算成本很高。
- 太短的键值通常也不是好主意，如果你要用”u:1000:pwd”来代替”user:1000:password”，这没有什么问题，但后者更易阅读，并且由此增加的空间消耗相对于key object和value object本身来说很小。当然，没人阻止您一定要用更短的键值节省一丁点儿空间。
- 最好坚持一种模式。例如：”object-type:id:field”就是个不错的注意，像这样”user:1000:password”。我喜欢对多单词的字段名中加上一个点，就像这样：”comment:1234:reply.to”。

Redis Strings

这是最简单Redis类型。如果你只用这种类型，Redis就像一个可以持久化的memcached服务器（注：memcache的数据仅保存在内存中，服务器重启后，数据将丢失）。

我们用redis-cli来玩一下字符串类型：

```
> set mykey somevalue
OK
> get mykey
"somevalue"
```

正如你所见到的，通常用SET command 和 GET command来设置和获取字符串值。

值可以是任何种类的字符串（包括二进制数据），例如你可以在一个键下保存一副jpeg图片。值的长度不能超过512 MB。

SET 命令有些有趣的操作，例如，当key存在时SET会失败，或相反的，当key不存在时它只会成功。

```
> set mykey newval nx
(nil)
> set mykey newval xx
OK
```

虽然字符串是Redis的基本值类型，但你仍然能通过它完成一些有趣的操作。例如：原子递增：

```
> set counter 100
OK
> incr counter
(integer) 101
> incr counter
(integer) 102
> incrby counter 50
(integer) 152
```

INCR 命令将字符串值解析成整型，将其加一，最后将结果保存为新的字符串值，类似的命令有INCRBY, DECR 和 DECRBY。实际上他们在内部就是同一个命令，只是看上去有点儿不同。

INCR是原子操作意味着什么呢？就是说即使多个客户端对同一个key发出INCR命令，也决不会导致竞争的情况。例如如下情况永远不可能发生：『客户端1和客户端2同时读出“10”，他们俩都对其加到11，然后将新值设置为11』。最终的值一定是12，read-increment-set操作完成时，其他客户端不会在同一时间执行任何命令。

对字符串，另一个的令人感兴趣的操作是GETSET命令，行如其名：他为key设置新值并且返回原值。这有什么用处呢？例如：你的系统每当有新用户访问时就用INCR命令操作一个Redis key。你希望每小时对这个信息收集一次。你就可以GETSET这个key并给其赋值0并读取原值。

为减少等待时间，也可以一次存储或获取多个key对应的值，使用MSET和MGET命令:

```
> mset a 10 b 20 c 30
OK
> mget a b c
1) "10"
2) "20"
3) "30"
```

MGET 命令返回由值组成的数组。

修改或查询键空间

有些指令不是针对任何具体的类型定义的，而是用于和整个键空间交互的。因此，它们可被用于任何类型的键。

使用EXISTS命令返回1或0标识给定key的值是否存在，使用DEL命令可以删除key对应的值，DEL命令返回1或0标识值是被删除(值存在)或者没被删除(key对应的值不存在)。

```
> set mykey hello
OK
> exists mykey
(integer) 1
> del mykey
(integer) 1
> exists mykey
(integer) 0
```

TYPE命令可以返回key对应的值的存储类型：

```
> set mykey x
OK
> type mykey
string
> del mykey
(integer) 1
> type mykey
none
```

Redis超时:数据在限定时间内存活

在介绍复杂类型前我们先介绍一个与值类型无关的Redis特性:超时。你可以对key设置一个超时时间，当这个时间到达后会被删除。精度可以使用毫秒或秒。

```
> set key some-value
OK
> expire key 5
(integer) 1
> get key (immediately)
"some-value"
> get key (after some time)
(nil)
```

上面的例子使用了EXPIRE来设置超时时间(也可以再次调用这个命令来改变超时时间，使用PERSIST命令去除超时时间)。我们也可以在创建值的时候设置超时时间:

```
> set key 100 ex 10
OK
> ttl key
(integer) 9
```

TTL命令用来查看key对应的值剩余存活时间。

Redis Lists

要说清楚列表数据类型，最好先讲一点儿理论背景，在信息技术界List这个词常常被使用不当。例如”Python Lists”就名不副实（名为Linked Lists），但他们实际上是数组（同样的数据类型在Ruby中叫数组）

一般意义上讲，列表就是有序元素的序列：10,20,1,2,3就是一个列表。但用数组实现的List和用Linked List实现的List，在属性方面大不相同。

Redis lists基于Linked Lists实现。这意味着即使在一个list中有数百万个元素，在头部或尾部添加一个元素的操作，其时间复杂度也是常数级别的。用LPUSH 命令在十个元素的list头部添加新元素，和在千万元素list头部添加新元素的速度相同。

那么，坏消息是什么？在数组实现的list中利用索引访问元素的速度极快，而同样的操作在linked list实现的list上没有那么快。

Redis Lists用linked list实现的原因是：对于数据库系统来说，至关重要的特性是：能非常快的在很大的列表上添加元素。另一个重要因素是，正如你将要看到的：Redis lists能在常数时间取得常数长度。

如果快速访问集合元素很重要，建议使用可排序集合(sorted sets)。可排序集合我们会随后介绍。

Redis lists 入门

LPUSH 命令可向list的左边（头部）添加一个新元素，而RPUSH命令可向list的右边（尾部）添加一个新元素。最后 LRANGE 命令可从list中取出一定范围的元素:

```
> rpush mylist A
(integer) 1
> rpush mylist B
(integer) 2
> lpush mylist first
(integer) 3
> lrange mylist 0 -1
1) "first"
2) "A"
3) "B"
```

注意:LRANGE 带有两个索引，一定范围的第一个和最后一个元素。这两个索引都可以为负来告知Redis从尾部开始计数，因此-1表示最后一个元素，-2表示list中的倒数第二个元素，以此类推。

上面的所有命令的参数都可变，方便你一次向list存入多个值。

```
> rpush mylist 1 2 3 4 5 "foo bar"
(integer) 9
> lrange mylist 0 -1
1) "first"
2) "A"
3) "B"
4) "1"
5) "2"
6) "3"
7) "4"
8) "5"
9) "foo bar"
```

还有一个重要的命令是pop,它从list中删除元素并同时返回删除的值。可以在左边或右边操作。

```
> rpush mylist a b c
(integer) 3
> rpop mylist
"c"
> rpop mylist
"b"
> rpop mylist
"a"
```

我们增加了三个元素，并弹出了三个元素，因此，在这最后 列表中的命令序列是空的，没有更多的元素可以被弹出。如果我们尝试弹出另一个元素，这是我们得到的结果：

```
> rpop mylist
(nil)
```

当list没有元素时，Redis 返回了一个NULL。

List的常用案例

正如你可以从上面的例子中猜到的，list可被用来实现聊天系统。还可以作为不同进程间传递消息的队列。关键是，你可以每次都以原先添加的顺序访问数据。这不需要任何SQL ORDER BY 操作，将会非常快，也会很容易扩展到百万级别元素的规模。

例如在评级系统中，比如社会化新闻网站 reddit.com，你可以把每个新提交的链接添加到一个list，用LRANGE可简单的对结果分页。

在博客引擎实现中，你可为每篇日志设置一个list，在该list中推入博客评论，等等。

Capped lists

可以使用LTRIM把list从左边截取指定长度。

```
> rpush mylist 1 2 3 4 5
(integer) 5
> ltrim mylist 0 2
OK
> lrange mylist 0 -1
1) "1"
2) "2"
3) "3"
```

List上的阻塞操作

可以使用Redis来实现生产者和消费者模型，如使用LPUSH和RPOP来实现该功能。但会遇到这种情景：list是空，这时候消费者就需要轮询来获取数据，这样就会增加redis的访问压力、增加消费端的cpu时间，而很多访问都是无用的。为此redis提供了阻塞式访问 BRPOP 和 BLPOP 命令。 消费者可以在获取数据时指定如果数据不存在阻塞的时间，如果在时限内获得数据则立即返回，如果超时还没有数据则返回null, 0表示一直阻塞。

同时redis还会为所有阻塞的消费者以先后顺序排队。

如需了解详细信息请查看 RPOPLPUSH 和 BRPOPLPUSH。

key 的自动创建和删除

目前为止，在我们的例子中，我们没有在推入元素之前创建空的 list，或者在 list 没有元素时删除它。在 list 为空时删除 key，并在用户试图添加元素（比如通过 LPUSH）而键不存在时创建空 list，是 Redis 的职责。

这不光适用于 lists，还适用于所有包括多个元素的 Redis 数据类型 – Sets, Sorted Sets 和 Hashes。

基本上，我们可以用三条规则来概括它的行为：

1. 当我们向一个聚合数据类型中添加元素时，如果目标键不存在，就在添加元素前创建空的聚合数据类型。
2. 当我们从聚合数据类型中移除元素时，如果值仍然是空的，键自动被销毁。
3. 对一个空的 key 调用一个只读的命令，比如 LLEN （返回 list 的长度），或者一个删除元素的命令，将总是产生同样的结果。该结果和对一个空的聚合类型做同个操作的结果是一样的。

规则 1 示例：

```
> del mylist
(integer) 1
> lpush mylist 1 2 3
(integer) 3
```

但是，我们不能对存在但类型错误的 key 做操作： > set foo bar OK > lpush foo 1 2 3 (error) WRONGTYPE Operation against a key holding the wrong kind of value > type foo string

规则 2 示例:

```
> lpush mylist 1 2 3
(integer) 3
> exists mylist
(integer) 1
> lpop mylist
"3"
> lpop mylist
"2"
> lpop mylist
"1"
> exists mylist
(integer) 0
```

所有的元素被弹出之后， key 不复存在。

规则 3 示例:

```
> del mylist
(integer) 0
> llen mylist
(integer) 0
> lpop mylist
(nil)
```

Redis Hashes —

Redis hash 看起来就像一个 “hash” 的样子，由键值对组成：

```
> hmset user:1000 username antirez birthyear 1977 verified 1
OK
> hget user:1000 username
"antirez"
> hget user:1000 birthyear
"1977"
> hgetall user:1000
1) "username"
2) "antirez"
3) "birthyear"
4) "1977"
5) "verified"
6) "1"
```

Hash 便于表示 *objects*，实际上，你可以放入一个 hash 的域数量实际上没有限制（除了可用内存以外）。所以，你可以在你的应用中以不同的方式使用 hash。

HMSET 指令设置 hash 中的多个域，而 HGET 取回单个域。HMGET 和 HGET 类似，但返回一系列值：

```
> hmget user:1000 username birthyear no-such-field
1) "antirez"
2) "1977"
3) (nil)
```

也有一些指令能够对单独的域执行操作，比如 HINCRBY：

```
> hincrby user:1000 birthyear 10
(integer) 1987
> hincrby user:1000 birthyear 10
(integer) 1997
```

你可以在文档中找到 hash 指令的完整列表。

值得注意的是，小的 hash 被用特殊方式编码，非常节约内存。

Redis Sets —

Redis Set 是 String 的无序排列。SADD 指令把新的元素添加到 set 中。对 set 也可做一些其他的操作，比如测试一个给定的元素是否存在，对不同 set 取交集，并集或差，等等。

```
> sadd myset 1 2 3
(integer) 3
> smembers myset
1. 3
2. 1
3. 2
```

现在我已经把三个元素加到我的 set 中，并告诉 Redis 返回所有的元素。可以看到，它们没有被排序 —— Redis 在每次调用时可能按照任意顺序返回元素，因为对于元素的顺序并没有规定。

Redis 有检测成员的指令。一个特定的元素是否存在?

```
> sismember myset 3
(integer) 1
> sismember myset 30
(integer) 0
```

“3” 是 set 的一个成员，而 “30” 不是。

Sets 适合用于表示对象间的关系。 例如，我们可以轻易使用 set 来表示标记。

一个简单的建模方式是，对每一个希望标记的对象使用 set。这个 set 包含和对象相关联的标签的 ID。

假设我们想要给新闻打上标签。 假设新闻 ID 1000 被打上了 1,2,5 和 77 四个标签，我们可以使用一个 set 把 tag ID 和新闻条目关联起来：

```
> sadd news:1000:tags 1 2 5 77
(integer) 4
```

但是，有时候我可能也会需要相反的关系：所有被打上相同标签的新闻列表：

```
> sadd tag:1:news 1000
(integer) 1
> sadd tag:2:news 1000
(integer) 1
> sadd tag:5:news 1000
(integer) 1
> sadd tag:77:news 1000
(integer) 1
```

获取一个对象的所有 tag 是很方便的：

```
> smembers news:1000:tags
1. 5
2. 1
3. 77
4. 2
```

注意：在这个例子中，我们假设你有另一个数据结构，比如一个 Redis hash，把标签 ID 对应到标签名称。

使用 Redis 命令行，我们可以轻易实现其它一些有用的操作。比如，我们可能需要一个含有 1, 2, 10, 和 27 标签的对象的列表。我们可以用 SINTER 命令来完成这件事。它获取不同 set 的交集。我们可以用：

```
> sinter tag:1:news tag:2:news tag:10:news tag:27:news
... results here ...
```

不光可以取交集，还可以取并集，差集，获取随机元素，等等。

获取一个元素的命令是 SPOP，它很适合对特定问题建模。比如，要实现一个基于 web 的扑克游戏，你可能需要用 set 来表示一副牌。假设我们用一个字符的前缀来表示不同花色：

```
>  sadd deck C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 CJ CQ CK
   D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 DJ DQ DK H1 H2 H3
   H4 H5 H6 H7 H8 H9 H10 HJ HQ HK S1 S2 S3 S4 S5 S6
   S7 S8 S9 S10 SJ SQ SK
   (integer) 52
```

现在，我们想要给每个玩家 5 张牌。SPOP 命令删除一个随机元素，把它返回给客户端，因此它是完全合适的操作。

但是，如果我们对我们的牌直接调用它，在下一盘我们就需要重新充满这副牌。开始，我们可以复制 deck 键中的内容，并放入 game:1:deck 键中。

这是通过 SUNIONSTORE 实现的，它通常用于对多个集合取并集，并把结果存入另一个 set 中。但是，因为一个 set 的并集就是它本身，我可以这样复制我的牌：

```
> sunionstore game:1:deck deck
(integer) 52
```

现在，我已经准备好给 1 号玩家发五张牌：

```
> spop game:1:deck
"C6"
> spop game:1:deck
"CQ"
> spop game:1:deck
"D1"
> spop game:1:deck
"CJ"
> spop game:1:deck
"SJ"
```

One pair of jacks, not great…

Now it's a good time to introduce the set command that provides the number of elements inside a set. This is often called the *cardinality of a set* in the context of set theory, so the Redis command is called SCARD.

```
> scard game:1:deck
(integer) 47
```

The math works: 52 - 5 = 47.

When you need to just get random elements without removing them from the set, there is the SRANDMEMBER command suitable for the task. It also features the ability to return both repeating and non-repeating elements.

Redis Sorted sets —

Sorted sets are a data type which is similar to a mix between a Set and a Hash. Like sets, sorted sets are composed of unique, non-repeating string elements, so in some sense a sorted set is a set as well.

However while elements inside sets are not ordered, every element in a sorted set is associated with a floating point value, called *the score* (this is why the type is also similar to a hash, since every element is mapped to a value).

Moreover, elements in a sorted sets are *taken in order* (so they are not ordered on request, order is a peculiarity of the data structure used to represent sorted sets). They are ordered according to the following rule:

- If A and B are two elements with a different score, then A > B if A.score is > B.score.
- If A and B have exactly the same score, then A > B if the A string is lexicographically greater than the B string. A and B strings can't be equal since sorted sets only have unique elements.

Let's start with a simple example, adding a few selected hackers names as sorted set elements, with their year of birth as "score".

```
> zadd hackers 1940 "Alan Kay"
(integer) 1
> zadd hackers 1957 "Sophie Wilson"
(integer 1)
> zadd hackers 1953 "Richard Stallman"
(integer) 1
> zadd hackers 1949 "Anita Borg"
(integer) 1
> zadd hackers 1965 "Yukihiro Matsumoto"
(integer) 1
> zadd hackers 1914 "Hedy Lamarr"
(integer) 1
> zadd hackers 1916 "Claude Shannon"
(integer) 1
> zadd hackers 1969 "Linus Torvalds"
(integer) 1
> zadd hackers 1912 "Alan Turing"
(integer) 1
```

As you can see ZADD is similar to SADD, but takes one additional argument (placed before the element to be added) which is the score. ZADD is also variadic, so you are free to specify multiple score-value pairs, even if this is not used in the example above.

With sorted sets it is trivial to return a list of hackers sorted by their birth year because actually *they are already sorted*.

Implementation note: Sorted sets are implemented via a dual-ported data structure containing both a skip list and a hash table, so every time we add an element Redis performs an O(log(N)) operation. That's good, but when we ask for sorted elements Redis does not have to do any work at all, it's already all sorted:

```
> zrange hackers 0 -1
1) "Alan Turing"
2) "Hedy Lamarr"
3) "Claude Shannon"
4) "Alan Kay"
5) "Anita Borg"
6) "Richard Stallman"
7) "Sophie Wilson"
8) "Yukihiro Matsumoto"
9) "Linus Torvalds"
```

Note: 0 and -1 means from element index 0 to the last element (-1 works here just as it does in the case of the LRANGE command).

What if I want to order them the opposite way, youngest to oldest? Use ZREVRANGE instead of ZRANGE:

```
> zrevrange hackers 0 -1
1) "Linus Torvalds"
2) "Yukihiro Matsumoto"
3) "Sophie Wilson"
4) "Richard Stallman"
5) "Anita Borg"
6) "Alan Kay"
7) "Claude Shannon"
8) "Hedy Lamarr"
9) "Alan Turing"
```

It is possible to return scores as well, using the WITHSCORES argument:

```
> zrange hackers 0 -1 withscores
1) "Alan Turing"
2) "1912"
3) "Hedy Lamarr"
4) "1914"
5) "Claude Shannon"
6) "1916"
7) "Alan Kay"
8) "1940"
9) "Anita Borg"
10) "1949"
11) "Richard Stallman"
12) "1953"
13) "Sophie Wilson"
14) "1957"
15) "Yukihiro Matsumoto"
16) "1965"
17) "Linus Torvalds"
18) "1969"
```

Operating on ranges

Sorted sets are more powerful than this. They can operate on ranges. Let's get all the individuals that were born up to 1950 inclusive. We use the ZRANGEBYSCORE command to do it:

```
> zrangebyscore hackers -inf 1950
1) "Alan Turing"
2) "Hedy Lamarr"
3) "Claude Shannon"
4) "Alan Kay"
5) "Anita Borg"
```

We asked Redis to return all the elements with a score between negative infinity and 1950 (both extremes are included).

It's also possible to remove ranges of elements. Let's remove all the hackers born between 1940 and 1960 from the sorted set:

```
> zremrangebyscore hackers 1940 1960
(integer) 4
```

`ZREMRANGEBYSCORE` is perhaps not the best command name, but it can be very useful, and returns the number of removed elements.

Another extremely useful operation defined for sorted set elements is the get-rank operation. It is possible to ask what is the position of an element in the set of the ordered elements.

```
> zrank hackers "Anita Borg"
(integer) 4
```

The `ZREVRANK` command is also available in order to get the rank, considering the elements sorted a descending way.

Lexicographical scores

With recent versions of Redis 2.8, a new feature was introduced that allows getting ranges lexicographically, assuming elements in a sorted set are all inserted with the same identical score (elements are compared with the C `memcmp` function, so it is guaranteed that there is no collation, and every Redis instance will reply with the same output).

The main commands to operate with lexicographical ranges are `ZRANGEBYLEX`, `ZREVRANGEBYLEX`, `ZREMRANGEBYLEX` and `ZLEXCOUNT`.

For example, let's add again our list of famous hackers, but this time use a score of zero for all the elements:

```
> zadd hackers 0 "Alan Kay" 0 "Sophie Wilson" 0 "Richard Stallman" 0
  "Anita Borg" 0 "Yukihiro Matsumoto" 0 "Hedy Lamarr" 0 "Claude Shannon"
  0 "Linus Torvalds" 0 "Alan Turing"
```

Because of the sorted sets ordering rules, they are already sorted lexicographically:

```
> zrange hackers 0 -1
1) "Alan Kay"
2) "Alan Turing"
3) "Anita Borg"
4) "Claude Shannon"
5) "Hedy Lamarr"
6) "Linus Torvalds"
7) "Richard Stallman"
8) "Sophie Wilson"
9) "Yukihiro Matsumoto"
```

Using ZRANGEBYLEX we can ask for lexicographical ranges:

```
> zrangebylex hackers [B [P
1) "Claude Shannon"
2) "Hedy Lamarr"
3) "Linus Torvalds"
```

Ranges can be inclusive or exclusive (depending on the first character), also string infinite and minus infinite are specified respectively with the + and – strings. See the documentation for more information.

This feature is important because it allows us to use sorted sets as a generic index. For example, if you want to index elements by a 128-bit unsigned integer argument, all you need to do is to add elements into a sorted set with the same score (for example 0) but with an 8 byte prefix consisting of **the 128 bit number in big endian**. Since numbers in big endian, when ordered lexicographically (in raw bytes order) are actually ordered numerically as well, you can ask for ranges in the 128 bit space, and get the element's value discarding the prefix.

If you want to see the feature in the context of a more serious demo, check the Redis autocomplete demo.

Updating the score: leader boards

Just a final note about sorted sets before switching to the next topic. Sorted sets' scores can be updated at any time. Just calling ZADD against an element already included in the sorted set will update its score (and position) with O(log(N)) time complexity. As such, sorted sets are suitable when there are tons of updates.

Because of this characteristic a common use case is leader boards. The typical application is a Facebook game where you combine the ability to take users sorted by their high score, plus the get-rank operation, in order to show the top-N users, and the user rank in the leader board (e.g., "you are the #4932 best score here").

Bitmaps —

Bitmaps are not an actual data type, but a set of bit-oriented operations defined on the String type. Since strings are binary safe blobs and their maximum length is 512 MB, they are suitable to set up to 2^32 different bits.

Bit operations are divided into two groups: constant-time single bit operations, like setting a bit to 1 or 0, or getting its value, and operations on groups of bits, for example counting the number of set bits in a given range of bits (e.g., population counting).

One of the biggest advantages of bitmaps is that they often provide extreme space savings when storing information. For example in a system where different users are represented by incremental user IDs, it is possible to remember a single bit information (for example, knowing whether a user wants to receive a newsletter) of 4 billion of users using just 512 MB of memory.

Bits are set and retrieved using the SETBIT and GETBIT commands:

```
> setbit key 10 1
(integer) 1
> getbit key 10
(integer) 1
> getbit key 11
(integer) 0
```

The SETBIT command takes as its first argument the bit number, and as its second argument the value to set the bit to, which is 1 or 0. The command automatically enlarges the string if the addressed bit is outside the current string length.

GETBIT just returns the value of the bit at the specified index. Out of range bits (addressing a bit that is outside the length of the string stored into the target key) are always considered to be zero.

There are three commands operating on group of bits:

1. BITOP performs bit-wise operations between different strings. The provided operations are AND, OR, XOR and NOT.
2. BITCOUNT performs population counting, reporting the number of bits set to 1.
3. BITPOS finds the first bit having the specified value of 0 or 1.

Both BITPOS and BITCOUNT are able to operate with byte ranges of the string, instead of running for the whole length of the string. The following is a trivial example of BITCOUNT call:

```
> setbit key 0 1
(integer) 0
> setbit key 100 1
(integer) 0
> bitcount key
(integer) 2
```

Common user cases for bitmaps are:

- Real time analytics of all kinds.
- Storing space efficient but high performance boolean information associated with object IDs.

For example imagine you want to know the longest streak of daily visits of your web site users. You start counting days starting from zero, that is the day you made your web site public, and set a bit with SETBIT every time the user visits the web site. As a bit index you simply take the current unix time, subtract the initial offset, and divide by 3600*24.

This way for each user you have a small string containing the visit information for each day. With BITCOUNT it is possible to easily get the number of days a given user visited the web site, while with a few BITPOS calls, or simply fetching and analyzing the bitmap client-side, it is possible to easily compute the longest streak.

Bitmaps are trivial to split into multiple keys, for example for the sake of sharding the data set and because in general it is better to avoid working with huge keys. To split a bitmap across different keys instead of setting all the bits into a key, a trivial strategy is just to store M bits per key and obtain the key name with `bit-number/M` and the Nth bit to address inside the key with `bit-number MOD M`.

HyperLogLogs —

A HyperLogLog is a probabilistic data structure used in order to count unique things (technically this is referred to estimating the cardinality of a set). Usually counting unique items requires using an amount of memory proportional to the number of items you want to count, because you need to remember the elements you have already seen in the past in order to avoid counting them multiple times. However there is a set of algorithms that trade memory for precision: you end with an estimated measure with a standard error, in the case of the Redis implementation, which is less than 1%. The

magic of this algorithm is that you no longer need to use an amount of memory proportional to the number of items counted, and instead can use a constant amount of memory! 12k bytes in the worst case, or a lot less if your HyperLogLog (We'll just call them HLL from now) has seen very few elements.

HLLs in Redis, while technically a different data structure, is encoded as a Redis string, so you can call GET to serialize a HLL, and SET to deserialize it back to the server.

Conceptually the HLL API is like using Sets to do the same task. You would SADD every observed element into a set, and would use SCARD to check the number of elements inside the set, which are unique since SADD will not re-add an existing element.

While you don't really *add items* into an HLL, because the data structure only contains a state that does not include actual elements, the API is the same:

- Every time you see a new element, you add it to the count with PFADD.
- Every time you want to retrieve the current approximation of the unique elements *added* with PFADD so far, you use the PFCOUNT.

```
> pfadd hll a b c d
(integer) 1
> pfcount hll
(integer) 4
```

An example of use case for this data structure is counting unique queries performed by users in a search form every day.

Redis is also able to perform the union of HLLs, please check the full documentation for more information.

Other notable features

There are other important things in the Redis API that can't be explored in the context of this document, but are worth your attention:

- It is possible to iterate the key space of a large collection incrementally.
- It is possible to run Lua scripts server side to win latency and bandwidth.
- Redis is also a Pub-Sub server.

Learn more

This tutorial is in no way complete and has covered just the basics of the API. Read the command reference to discover a lot more.

Thanks for reading, and have fun hacking with Redis!

关于data-types-intro互动的最新评论

_目_ 发布于 2017-7-18 09:01:55

楼主，什么叫数据结构服务器?

lizhiyong2204 发布于 2017-6-13 14:26:38

这里描述错误

lizhiyong2204 发布于 2017-6-13 14:24:45

http://wx3.sinaimg.cn/mw690/4dbbaa1cgy1fgjk0l5b95j20uf07i0to.jpg

geelou 发布于 2016-11-29 13:16:50

没明白，没看出问题

伟勤 发布于 2016-11-24 16:02:08

有一处文章描述和实际操作不符合，请看下。

geelou 发布于 2016-8-31 18:09:50

data-types-intro互动

发表评论