十五分钟介绍 Redis数据结构

作者：nosqlfan on 星期六, 十月 8, 2011 · *9条评论* 【阅读：102,138 次】

下面是一个对Redis官方文档《A fifteen minute introduction to Redis data types》一文的翻译，如其题目所言，此文目的在于让一个初学者能通过15分钟的简单学习对Redis的数据结构有一个了解。

> Redis是一种面向“键/值”对类型数据的分布式NoSQL数据库系统，特点是高性能，持久存储，适应高并发的应用场景。它起步较晚，发展迅速，目前已被许多大型机构采用，比如Github，看看谁在用它。
> 本文翻译自Redis的一篇官方文档：A fifteen minute introduction to Redis data types
> 方便感兴趣的朋友，快速介绍Redis的数据类型。
>
> 中英文对照，如有疏漏敬请留言，某些关键词不译，便于阅读。
>
> ————————————————————————————————————
>
> 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 **string**
> - 二进制安全的 字符串列表 list of string
> - 二进制安全的 字符串集合 set of string，换言之：它是一组无重复未排序的element。可以把它看成Ruby中的 hash–其key等于element，value都等于’true‘。
> - 有序集合**sorted set** of string，类似于集合set，但其中每个元素都和一个浮点数score（评分）关联。element根据score排序。可以把它看成Ruby中的 hash–其key等于element，value等于score，但元素总是按score的顺序排列，无需额外的排序操作。
>
> ## Redis 键
>
> 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”。

字符串类型

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）。

我们来玩儿一下字符串类型：

```
$ redis-cli set mykey "my binary safe value"
OK
$ redis-cli get mykey
my binary safe value
```

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1GB。

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：

```
$ redis-cli set counter 100
OK $ redis-cli incr counter
(integer) 101
$ redis-cli incr counter
(integer) 102
$ redis-cli incrby counter 10
(integer) 112
```

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 and 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。

模式：计数器、限速器，使用场景：全局锁

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。

列表类型

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。

使用链表实现的理由：

Redis Lists are implemented with linked lists because for a database system it is crucial to be able to add elements to a very long list in a very fast way. Another strong advantage is, as you'll see in a moment, that Redis Lists can be taken at constant length in constant time.

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。

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

```
$ redis-cli rpush messages "Hello how are you?"
OK
$ redis-cli rpush messages "Fine thanks. I'm having fun with Redis"
OK
$ redis-cli rpush messages "I should look into this NOSQL thing ASAP"
OK
$ redis-cli lrange messages 0 2
1. Hello how are you?
2. Fine thanks. I'm having fun with Redis
3. I should look into this NOSQL thing ASAP
```

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。

使用场景：聊天系统、工作队列

As you can guess from the example above, lists can be used, for instance, in order to implement a chat system. Another use is as queues in order to route messages between different processes. But the key point is that you can use Redis lists every time you require to access data in the same order they are added. This will not require any SQL ORDER BY operation, will be very fast, and will scale to millions of elements even with a toy Linux box.

关键点：以添加的顺序来访问数据

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中推入进博客评论，等等。

向Redis list压入ID而不是实际的数据

在上面的例子里，我们将“对象”（此例中是简单消息）直接压入Redis list，但通常不应这么做，由于对象可能被多次引用：例如在一个list中维护其时间顺序，在一个集合中保存它的类别，只要有必要，它还会出现在其他list中，等等。

让我们回到reddit.com的例子，将用户提交的链接（新闻）添加到list中，有更可靠的方法如下所示：

```
$ redis-cli incr next.news.id
(integer) 1
$ redis-cli set news:1:title "Redis is simple"
OK
$ redis-cli set news:1:url "http://code.google.com/p/redis"
OK
$ redis-cli lpush submitted.news 1
OK
```

标题、URL 可以通过 hash 类型

我们自增一个key，很容易得到一个独一无二的自增ID，然后通过此ID创建对象–为对象的每个字段设置一个key。最后将新对象的ID压入*submitted.news* list。

这只是牛刀小试。在命令参考文档中可以读到所有和list有关的命令。你可以删除元素，旋转list，根据索引获取和设置元素，当然也可以用LLEN得到list的长度。

Redis 集合

Redis集合是未排序的集合，其元素是二进制安全的字符串。SADD命令可以向集合添加一个新元素。和sets相关的操作也有许多，比如检测某个元素是否存在，以及实现交集，并集，差集等等。一例胜千言：

```
$ redis-cli sadd myset 1
(integer) 1
$ redis-cli sadd myset 2
(integer) 1
$ redis-cli sadd myset 3
(integer) 1
$ redis-cli smembers myset
1. 3
2. 1
3. 2
```

我向集合中添加了三个元素，并让Redis返回所有元素。如你所见它们是无序的。

现在让我们检查某个元素是否存在：

```
$ redis-cli sismember myset 3
(integer) 1
$ redis-cli sismember myset 30
(integer) 0
```

使用场景：共同好友（交集）

“3”是这个集合的成员，而“30”不是。集合特别适合表现对象之间的关系。例如用Redis集合可以很容易实现标签功能。

N:N - 多对多关系

下面是一个简单的方案：对每个想加标签的对象，用一个标签ID集合与之关联，并且对每个已有的标签，一组对象ID与之关联。

例如假设我们的新闻ID 1000被加了三个标签tag 1,2,5和77，就可以设置下面两个集合：

```
$ redis-cli sadd news:1000:tags 1
(integer) 1
$ redis-cli sadd news:1000:tags 2
(integer) 1
$ redis-cli sadd news:1000:tags 5
(integer) 1
$ redis-cli sadd news:1000:tags 77
(integer) 1
$ redis-cli sadd tag:1:objects 1000
(integer) 1
$ redis-cli sadd tag:2:objects 1000
(integer) 1
$ redis-cli sadd tag:5:objects 1000
(integer) 1
$ redis-cli sadd tag:77:objects 1000
(integer) 1
```

要获取一个对象的所有标签，如此简单：

```
$ redis-cli smembers news:1000:tags
1. 5
2. 1
3. 77
4. 2
```

而有些看上去并不简单的操作仍然能使用相应的Redis命令轻松实现。例如我们也许想获得一份同时拥有标签1, 2, 10和27的对象列表。这可以用SINTER命令来做，他可以在不同集合之间取出交集。因此为达目的我们只需：

```
$ redis-cli sinter tag:1:objects tag:2:objects tag:10:objects tag:27:objects
... no result in our dataset composed of just one object ☺ ...
```

在命令参考文档中可以找到和集合相关的其他命令，令人感兴趣的一抓一大把。一定要留意SORT命令，Redis集合和list都是可排序的。

题外话：如何为字符串获取唯一标识

在标签的例子里，我们用到了标签ID，却没有提到ID从何而来。基本上你得为每个加入系统的标签分配一个唯一标识。你也希望在多个客户端同时试着添加同样的标签时不要出现竞争的情况。此外，如果标签已存在，你希望返回他的ID，否则创建一个新的唯一

标识并将其与此标签关联。

Redis 1.4将增加Hash类型。有了它，字符串和唯一ID关联的事儿将不值一提，但如今我们如何用现有Redis命令可靠的解决它呢?

我们首先的尝试（以失败告终）可能如下。假设我们想为标签“redis”获取一个唯一ID:

- 为了让算法是二进制安全的（只是标签而不考虑utf8，空格等等）我们对标签做SHA1签名。SHA1(redis)=b840fc02d524045429941cc15f59e41cb7be6c52。
- 检查这个标签是否已与一个唯一ID关联，
  用命令*GET tag:b840fc02d524045429941cc15f59e41cb7be6c52:id*
- 如果上面的GET操作返回一个ID，则将其返回给用户。标签已经存在了。
- 否则… 用*INCR next.tag.id*命令生成一个新的唯一ID（假定它返回123456）。
- 最后关联标签和新的ID，
  SET tag:b840fc02d524045429941cc15f59e41cb7be6c52:id 123456
  并将新ID返回给调用者。

多美妙，或许更好…等等！当两个客户端同时使用这组指令尝试为标签“redis”获取唯一ID时会发生什么呢？如果时间凑巧，他们俩都会从GET操作获得nil，都将对*next.tag.id key*做自增操作，这个key会被自增两次。其中一个客户端会将错误的ID返回给调用者。幸运的是修复这个算法并不难，这是明智的版本:

- 为了让算法是二进制安全的（只是标签而不考虑utf8，空格等等）我们对标签做SHA1签名。SHA1(redis)=b840fc02d524045429941cc15f59e41cb7be6c52。
- 检查这个标签是否已与一个唯一ID关联，
  用命令*GET tag:b840fc02d524045429941cc15f59e41cb7be6c52:id*
- 如果上面的GET操作返回一个ID，则将其返回给用户。标签已经存在了。
- 否则… 用*INCR next.tag.id*命令生成一个新的唯一ID（假定它返回123456）。
- 下面关联标签和新的ID，(注意用到一个新的命令)
  SETNX tag:b840fc02d524045429941cc15f59e41cb7be6c52:id 123456。如果另一个客户端比当前客户端更快，SETNX将不会设置key。而且，当key被成功设置时SETNX返回1，否则返回0。那么…让我们再做最后一步运算。
- 如果SETNX返回1（key设置成功）则将123456返回给调用者，这就是我们的标签ID，否则执行*GET tag:b840fc02d524045429941cc15f59e41cb7be6c52:id* 并将其结果返回给调用者。

有序集合

集合是使用频率很高的数据类型，但是…对许多问题来说他们也有点儿太不讲顺序了;)因此Redis1.2引入了有序集合。他和集合非常相似，也是二进制安全的字符串集合，但是这次带有关联的score，以及一个类似LRANGE的操作可以返回有序元素，此操作只能作用于有序集合，它就是，ZRANGE 命令。

基本上有序集合从某种程度上说是SQL世界的索引在Redis中的等价物。例如在上面提到的reddit.com例子中，并没有提到如何根据用户投票和时间因素将新闻组合生成首页。我

们将看到有序集合如何解决这个问题，但最好先从更简单的事情开始，阐明这个高级数据类型是如何工作的。让我们添加几个黑客，并将他们的生日作为“score”。

```
$ redis-cli zadd hackers 1940 "Alan Kay"
(integer) 1
$ redis-cli zadd hackers 1953 "Richard Stallman"
(integer) 1
$ redis-cli zadd hackers 1965 "Yukihiro Matsumoto"
(integer) 1
$ redis-cli zadd hackers 1916 "Claude Shannon"
(integer) 1
$ redis-cli zadd hackers 1969 "Linus Torvalds"
(integer) 1
$ redis-cli zadd hackers 1912 "Alan Turing"
(integer) 1
```

对有序集合来说，按生日排序返回这些黑客易如反掌，因为他们已经是有序的。有序集合是通过一个dual-ported 数据结构实现的，它包含一个精简的有序列表和一个hash table，因此添加一个元素的时间复杂度是O(log(N))。这还行，但当我们需要访问有序的元素时，Redis不必再做任何事情，它已经是有序的了：

```
$ redis-cli zrange hackers 0 -1
1. Alan Turing
2. Claude Shannon
3. Alan Kay
4. Richard Stallman
5. Yukihiro Matsumoto
6. Linus Torvalds
```

你知道Linus比Yukihiro年轻吗

无论如何，我想反向对这些元素排序，这次就用 ZREVRANGE 代替 ZRANGE 吧：

```
$ redis-cli zrevrange hackers 0 -1
1. Linus Torvalds
2. Yukihiro Matsumoto
3. Richard Stallman
4. Alan Kay
5. Claude Shannon
6. Alan Turing
```

一个非常重要的小贴士，ZSets只是有一个“默认的”顺序，但你仍然可以用 SORT 命令对有序集合做不同的排序（但这次服务器要耗费CPU了）。要想得到多种排序，一种可选方案是同时将每个元素加入多个有序集合。

区间操作

有序集合之能不止于此，他能在区间上操作。例如获取所有1950年之前出生的人。我们用 ZRANGEBYSCORE 命令来做：

```
$ redis-cli zrangebyscore hackers -inf 1950
1. Alan Turing
2. Claude Shannon
3. Alan Kay
```

我们请求Redis返回score介于负无穷到1950年之间的元素（两个极值也包含了）。

也可以删除区间内的元素。例如从有序集合中删除生日介于1940到1960年之间的黑客。

```
$ redis-cli zremrangebyscore hackers 1940 1960
(integer) 2
```

ZREMRANGEBYSCORE 这个名字虽然不算好，但他却非常有用，还会返回已删除的元素数量。

回到Reddit的例子

最后，回到 Reddit的例子。现在我们有个基于有序集合的像样方案来生成首页。用一个有序集合来包含最近几天的新闻（用 ZREMRANGEBYSCORE 不时的删除旧新闻）。用一个后台任务从有序集合中获取所有元素，根据用户投票和新闻时间计算score，然后用新闻IDs和scores关联生成 *reddit.home.page* 有序集合。要显示首页，我们只需闪电般的调用 ZRANGE。

不时的从 *reddit.home.page* 有序集合中删除过旧的新闻也是为了让我们的系统总是工作在有限的新闻集合之上。

更新有序集合的scores

结束这篇指南之前还有最后一个小贴士。有序集合scores可以在任何时候更新。只要用 ZADD 对有序集合内的元素操作就会更新它的score（和位置），时间复杂度是 O(log(N))，因此即使大量更新，有序集合也是合适的。

这篇指南远未尽言，这只是从Redis开始的基础，欲深入之请读命令参考文档。

谢谢阅读。Salvatore。

来源：blog.1001i.com