redis (http://39.106.3.150/tags/#redis)

深入Redis内存模型

Posted by Palmer on 07-13，2019

基本介绍

Redis是一个开源（BSD许可），内存存储的数据结构服务器，可用作数据库，高速缓存和消息队列代理。它支持字符串、哈希表、列表、集合、有序集合，位图，hyperloglogs等数据类型。内置复制、Lua脚本、LRU收回、事务以及不同级别磁盘持久化功能，同时通过Redis Sentinel提供高可用，通过Redis Cluster提供自动分区。

Redis是一个key-value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

Redis 是一个高性能的key-value数据库。 redis的出现，很大程度补偿了memcached这类key/value存储的不足，在部 分场合可以对关系数据库起到很好的补充作用。它提供了Java，C/C++，C#，PHP，JavaScript，Perl，Object-C，Python，Ruby，Erlang等客户端，使用很方便。

Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得Redis可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

redis的优点：

1、读写性能优异，从内存当中进行IO读写速度快。

2、支持数据持久化，支持AOF和RDB两种持久化方式(由于Redis的数据都存放在内存中，如果没有配置持久化，redis重启后数据就全丢失了，于是需要开启redis的持久化功能，将数据保存到磁 盘上，当redis重启后，可以从磁盘中恢复数据。redis提供两种方式进行持久化，一种是RDB持久化:指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork一个子进程，

先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。还有一种是AOF持久化：以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。）

3、支持主从复制，主机会自动将数据同步到从机，可以进行读写分离。

4、数据结构丰富：除了支持string类型的value外还支持string、hash、set、sortedset、list等数据结构。

5、Redis是单线程多CPU，这样速度更快。因为单线程，没有线程切换的开销，不需要考虑加锁释放锁，也就没有死锁的问题。单线程-多路复用IO模型。效率高。

redis的缺点：

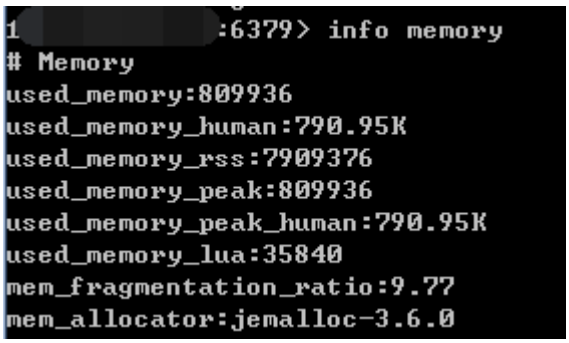
1、主从同步，如果主机宕机，宕机前有一部分数据没有同步到从机，会导致数据不一致。

2、主从同步，数据同步会有延迟。

3、读写分离，主机写的负载量太大，也会导致主机的宕机

Redis内存统计

在客户端通过redis-cli连接服务器后（后面如无特殊说明，客户端一律使用redis-cli），通过info命令可以查看内存使用情况：



info命令可以显示redis服务器的许多信息，包括服务器基本信息、CPU、内存、持久化、客户端连接信息等等；memory是参数，表示只显示内存相关的信息。

返回结果中比较重要的几个说明如下：

（1）**used_memory**：Redis分配器分配的内存总量（单位是字节），包括使用的虚拟内存（即swap）；Redis分配器后面会介绍。used_memory_human只是显示更友好。

（2）**used_memory_rss**：Redis进程占据操作系统的内存（单位是字节），与top及ps命令看到的值是一致的；除了分配器分配的内存之外，used_memory_rss还包括进程运行本身需要的内存、内存碎片等，但是不包括虚拟内存。因此，used_memory和used_memory_rss，前者是从Redis角度得到的量，后者是从操作系统角度得到的量。二者之所以有所不同，一方面是因为内存碎片和Redis进程运行需要占用内存，使得前者可能比后者小，另一方面虚拟内存的存在，使得前者可能比后者大。由于在实际应用中，Redis的数据量会比较大，此时进程运行占用的内存与Redis

数据量和内存碎片相比，都会小得多；因此used_memory_rss和used_memory的比例，便成了衡量Redis内存碎片率的参数；这个参数就是mem_fragmentation_ratio。

（3）**mem_fragmentation_ratio**：内存碎片比率，该值是used_memory_rss / used_memory的比值。mem_fragmentation_ratio一般大于1，且该值越大，内存碎片比例越大。mem_fragmentation_ratio<1，说明Redis使用了虚拟内存，由于虚拟内存的媒介是磁盘，比内存速度要慢很多，当这种情况出现时，应该及时排查，如果内存不足应该及时处理，如增加Redis节点、增加Redis服务器的内存、优化应用等。一般来说，mem_fragmentation_ratio在1.03左右是比较健康的状态（对于jemalloc来说）；上面截图中的mem_fragmentation_ratio值很大，是因为还没有向Redis中存入数据，Redis进程本身运行的内存使得used_memory_rss 比used_memory大得多。

（4）**mem_allocator**：Redis使用的内存分配器，在编译时指定；可以是 libc 、jemalloc或者tcmalloc，默认是jemalloc；截图中使用的便是默认的jemalloc。

附上：Redis命令大全 (http://doc.redisfans.com/)

Redis内存部分

Redis作为内存数据库，在内存中存储的内容主要是数据（键值对）；通过前面的叙述可以知道，除了数据以外，Redis的其他部分也会占用内存。

Redis的内存占用主要可以划分为以下几个部分：

1、**数据**。作为数据库，数据是最主要的部分；这部分占用的内存会统计在used_memory中。Redis使用键值对存储数据，其中的值（对象）包括5种类型，即字符串、哈希、列表、集合、有序集合。这5种类型是Redis对外提供的，实际上，在Redis内部，每种类型可能有2种或更多的内部编码实现；此外，Redis在存储对象时，并不是直接将数据扔进内存，而是会对对象进行各种包装：如redisObject、SDS等；这篇文章后面将重点介绍Redis中数据存储的细节。

2、**进程本身运行需要的内存**。Redis主进程本身运行肯定需要占用内存，如代码、常量池等等；这部分内存大约几兆，在大多数生产环境中与Redis数据占用的内存相比可以忽略。这部分内存不是由jemalloc分配，因此不会统计在used_memory中。

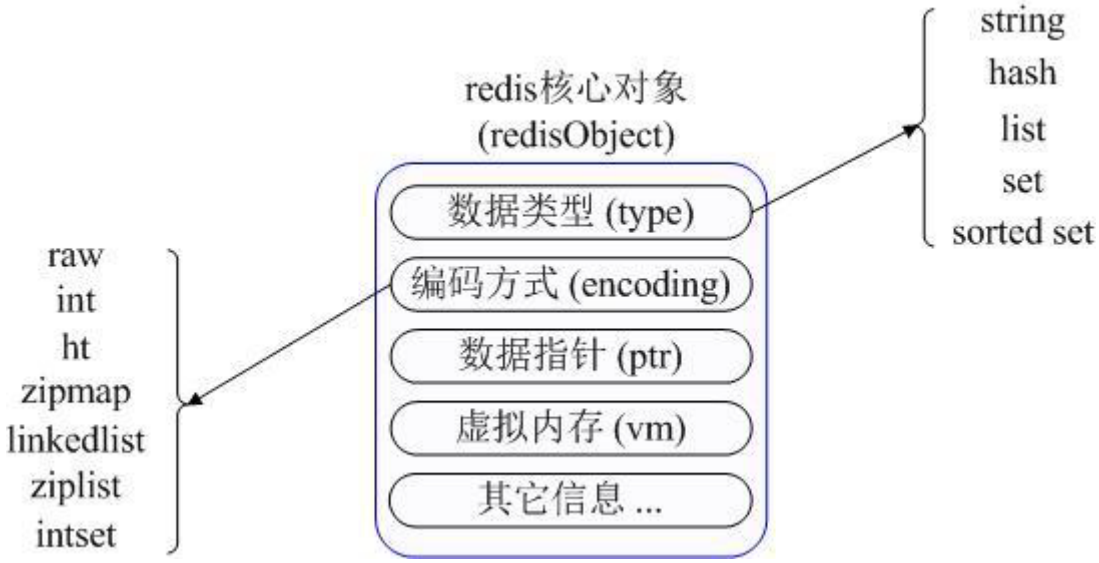
补充说明：除了主进程外，Redis创建的子进程运行也会占用内存，如Redis执行AOF、RDB重写时创建的子进程。当然，这部分内存不属于Redis进程，也不会统计在used_memory和used_memory_rss中。

3、**缓冲内存**。缓冲内存包括客户端缓冲区、复制积压缓冲区、AOF缓冲区等；其中，客户端缓冲存储客户端连接的输入输出缓冲；复制积压缓冲用于部分复制功能；AOF缓冲区用于在进行AOF重写时，保存最近的写入命令。在了解相应功能之前，不需要知道这些缓冲的细节；这部分内存由jemalloc分配，因此会统计在used_memory中。

4、**内存碎片**。内存碎片是Redis在分配、回收物理内存过程中产生的。例如，如果对数据的更改频繁，而且数据之间的大小相差很大，可能导致redis释放的空间在物理内存中并没有释放，

但redis又无法有效利用，这就形成了内存碎片。内存碎片不会统计在used_memory中。内存碎片的产生与对数据进行的操作、数据的特点等都有关；此外，与使用的内存分配器也有关系：如果内存分配器设计合理，可以尽可能的减少内存碎片的产生。后面将要说到的jemalloc便在控制内存碎片方面做的很好。如果Redis服务器中的内存碎片已经很大，可以通过安全重启的方式减小内存碎片：因为重启之后，Redis重新从备份文件中读取数据，在内存中进行重排，为每个数据重新选择合适的内存单元，减小内存碎片。

Redis对象的类型与编码



Redis使用前面说的五大数据类型来表示键和值，每次在Redis数据库中创建一个键值对时，至少会创建两个对象，一个是键对象，一个是值对象，而Redis中的每个对象都是由 redisObject 结构来表示。redisObject主要的信息包括数据类型（type）、编码方式(encoding)、数据指针（ptr）、虚拟内存（vm）等。type代表一个value对象具体是何种数据类型，encoding是不同数据类型在redis内部式。

```
typedef struct redisObject{
     //类型
     unsigned type:4;
     //编码
     unsigned encoding:4;
     //指向底层数据结构的指针
     void *ptr;
     //引用计数
     int refcount;
     //记录最后一次被程序访问的时间
     unsigned lru:22;

}robj
```

对象的type属性记录了对象的类型，这个类型就是前面讲的五大数据类型：

（1）**type**

type字段表示对象的类型，占4个比特；目前包括REDIS_STRING(字符串)、REDIS_LIST (列表)、REDIS_HASH(哈希)、REDIS_SET(集合)、REDIS_ZSET(有序集合)。

当我们执行type命令时，便是通过读取RedisObject的type字段获得对象的类型；如下图所示：

```
127.0.0.1:6379> set mystring helloredis
OK
127.0.0.1:6379> type mystring
string
127.0.0.1:6379> sadd myset member1 member2 member3
(integer) 3
127.0.0.1:6379> type myset
set
```

（2）**encoding**

encoding表示对象的内部编码，占4个比特。

对于Redis支持的每种类型，都有至少两种内部编码，例如对于字符串，有int、embstr、raw三种编码。通过encoding属性，Redis可以根据不同的使用场景来为对象设置不同的编码，大大提高了Redis的灵活性和效率。以列表对象为例，有压缩列表和双端链表两种编码方式；如果列表中的元素较少，Redis倾向于使用压缩列表进行存储，因为压缩列表占用内存更少，而且比双端链表可以更快载入；当列表对象元素较多时，压缩列表就会转化为更适合存储大量元素的双端链表。

通过object encoding命令，可以查看对象采用的编码方式，如下图所示：

```
127.0.0.1:6379> set key1 33
OK
127.0.0.1:6379> object encoding key1
"int"
127.0.0.1:6379> set key2 helloworld
OK
127.0.0.1:6379> object encoding key2
"embstr"
```

（3）**lru**

lru记录的是对象最后一次被命令程序访问的时间，占据的比特数不同的版本有所不同（如4.0版本占24比特，2.6版本占22比特）。

通过对比lru时间与当前时间，可以计算某个对象的空转时间；object idletime命令可以显示该空转时间（单位是秒）。object idletime命令的一个特殊之处在于它不改变对象的lru值。

lru值除了通过object idletime命令打印之外，还与Redis的内存回收有关系：如果Redis打开了maxmemory选项，且内存回收算法选择的是volatile-lru或allkeys—lru，那么当Redis内存占用超过maxmemory指定的值时，Redis会优先选择空转时间最长的对象进行释放。

（4）**refcount**

refcount与共享对象

refcount记录的是该对象被引用的次数，类型为整型。refcount的作用，主要在于对象的引用计数和内存回收。当创建新对象时，refcount初始化为1；当有新程序使用该对象时，refcount加1；当

对象不再被一个新程序使用时，refcount减1；当refcount变为0时，对象占用的内存会被释放。

Redis中被多次使用的对象(refcount>1)，称为共享对象。Redis为了节省内存，当有一些对象重复出现时，新的程序不会创建新的对象，而是仍然使用原来的对象。这个被重复使用的对象，就是共享对象。目前共享对象仅支持整数值的字符串对象。

共享对象的具体实现

Redis的共享对象目前只支持整数值的字符串对象。之所以如此，实际上是对内存和CPU（时间）的平衡：共享对象虽然会降低内存消耗，但是判断两个对象是否相等却需要消耗额外的时间。对于整数值，判断操作复杂度为O(1)；对于普通字符串，判断复杂度为O(n)；而对于哈希、列表、集合和有序集合，判断的复杂度为O(n^2)。

虽然共享对象只能是整数值的字符串对象，但是5种类型都可能使用共享对象（如哈希、列表等的元素可以使用）。

就目前的实现来说，Redis服务器在初始化时，会创建10000个字符串对象，值分别是0~9999的整数值；当Redis需要使用值为0~9999的字符串对象时，可以直接使用这些共享对象。10000这个数字可以通过调整参数

REDIS_SHARED_INTEGERS（4.0中是OBJ_SHARED_INTEGERS）的值进行改变。

共享对象的引用次数可以通过object refcount命令查看，如下图所示。命令执行的结果页佐证了只有0~9999之间的整数会作为共享对象。

```
127.0.0.1:6379> set mystring helloredis
OK
127.0.0.1:6379> object idletime mystring
(integer) 9
127.0.0.1:6379> object idletime mystring
(integer) 12
127.0.0.1:6379> object idletime mystring
(integer) 82
127.0.0.1:6379> get mystring
"helloredis"
127.0.0.1:6379> object idletime mystring
(integer) 4
```

（5）**ptr**

ptr指针指向具体的数据，如前面的例子中，set hello world，ptr指向包含字符串world的SDS。

（6）**总结**

综上所述，redisObject的结构与对象类型、编码、内存回收、共享对象都有关系；一个redisObject对象的大小为16字节：

**encoding 属性和 prt 指针*

4bit+4bit+24bit+4Byte+8Byte=16Byte。

对象	对象 type 属性的值	TYPE 命令的输出
字符串对象	REDIS_STRING	"string"
列表对象	REDIS_LIST	"list"
哈希对象	REDIS_HASH	"hash"
集合对象	REDIS_SET	"set"
有序集合对象	REDIS_ZSET	"zset"

编码常量	编码所对应的底层数据结构
REDIS_ENCODING_INT	long 类型的整数
REDIS_ENCODING_EMBSTR	embstr 编码的简单动态字符串
REDIS_ENCODING_RAW	简单动态字符串
REDIS_ENCODING_HT	字典
REDIS_ENCODING_LINKEDLIST	双端链表
REDIS_ENCODING_ZIPLIST	压缩列表
REDIS_ENCODING_INTSET	整数集合
REDIS_ENCODING_SKIPLIST	跳跃表和字典

类 型	编 码	对 象
REDIS_STRING	REDIS_ENCODING_INT	使用整数值实现的字符串对象
REDIS_STRING	REDIS_ENCODING_EMBSTR	使用 embstr 编码的简单动态字符串实现的字符串对象
REDIS_STRING	REDIS_ENCODING_RAW	使用简单动态字符串实现的字符串对象
REDIS_LIST	REDIS_ENCODING_ZIPLIST	使用压缩列表实现的列表对象
REDIS_LIST	REDIS_ENCODING_LINKEDLIST	使用双端链表实现的列表对象
REDIS_HASH	REDIS_ENCODING_ZIPLIST	使用压缩列表实现的哈希对象
REDIS_HASH	REDIS_ENCODING_HT	使用字典实现的哈希对象
REDIS_SET	REDIS_ENCODING_INTSET	使用整数集合实现的集合对象
REDIS_SET	REDIS_ENCODING_HT	使用字典实现的集合对象
REDIS_ZSET	REDIS_ENCODING_ZIPLIST	使用压缩列表实现的有序集合对象
REDIS_ZSET	REDIS_ENCODING_SKIPLIST	使用跳跃表和字典实现的有序集合对象

Redis五大数据类型

字符串对象

字符串是Redis最基本的数据类型，不仅所有key都是字符串类型，其它几种数据类型构成的元素也是字符串。注意字符串的长度不能超过512M。

①、编码

字符串对象的编码可以是int，raw或者embstr。

1、int 编码：保存的是可以用 long 类型表示的整数值。

2、raw 编码：保存长度大于44字节的字符串（redis3.2版本之前是39字节，之后是44字节）。

3、embstr 编码：保存长度小于44字节的字符串（redis3.2版本之前是39字节，之后是44字节）。

由上可以看出，int 编码是用来保存整数值，raw编码是用来保存长字符串，而embstr是用来保存短字符串。其实 embstr 编码是专门用来保存短字符串的一种优化编码，raw 和 embstr 的区别：

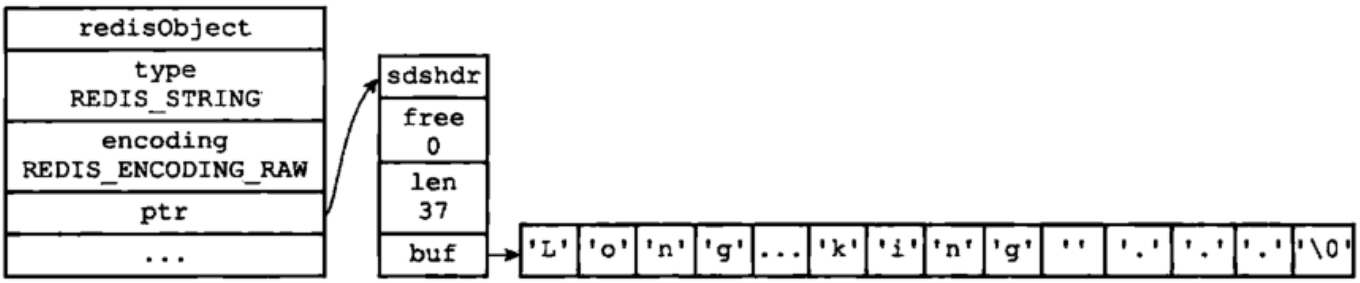


图 8-2 raw 编码的字符串对象

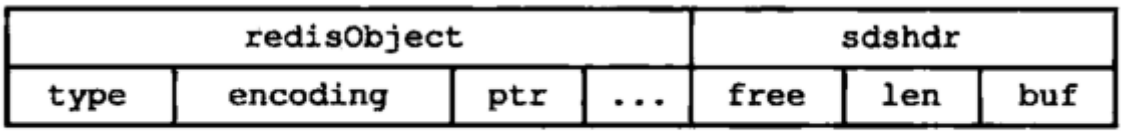


图 8-3 embstr 编码创建的内存块结构

embstr与raw都使用redisObject和sds保存数据，区别在于，embstr的使用只分配一次内存空间（因此redisObject和sds是连续的），而raw需要分配两次内存空间（分别为redisObject和sds分配空间）。因此与raw相比，embstr的好处在于创建时少分配一次空间，删除时少释放一次空间，以及对象的所有数据连在一起，寻找方便。而embstr的坏处也很明显，如果字符串的长度增加需要重新分配内存时，整个redisObject和sds都需要重新分配空间，因此redis中的embstr实现为只读。

Redis中对于浮点数类型也是作为字符串保存的，在需要的时候再将其转换成浮点数类型。

②、编码的转换

当 int 编码保存的值不再是整数，或大小超过了long的范围时，自动转化为raw。

对于 embstr 编码，由于 Redis 没有对其编写任何的修改程序（embstr 是只读的），在对embstr对象进行修改时，都会先转化为raw再进行修改，因此，只要是修改embstr对象，修改后的对象一定是raw的，无论是否达到了44个字节。

列表对象

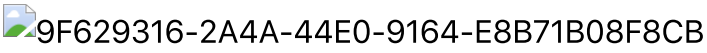
list 列表，它是简单的字符串列表，按照插入顺序排序，你可以添加一个元素到列表的头部（左边）或者尾部（右边），它的底层实际上是个链表结构。

①、编码

列表对象的编码可以是 ziplist(压缩列表) 和 linkedlist(双端链表)。 关于链表和压缩列表的特性可以看我前面的这篇博客。

比如我们执行以下命令，创建一个 key = ‘numbers’，value = ‘1 three 5’ 的三个值的列表。

```
rpush numbers 1 "three" 5
```



②、编码转换

当同时满足下面两个条件时，使用ziplist（压缩列表）编码：

1、列表保存元素个数小于512个

2、每个元素长度小于64字节

不能满足这两个条件的时候使用 linkedlist 编码。上面两个条件可以在redis.conf 配置文件中的 list-max-ziplist-value选项和 list-max-ziplist-entries 选项进行配置。

哈希对象

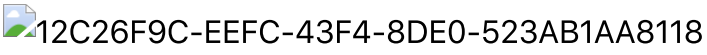
哈希对象的键是一个字符串类型，值是一个键值对集合。

①、编码

哈希对象的编码可以是 ziplist 或者 hashtable。

当使用ziplist，也就是压缩列表作为底层实现时，新增的键值对是保存到压缩列表的表尾。比如执行以下命令：

```
hset profile name "Tom"
hset profile age 25
hset profile career "Programmer"
```



hashtable 编码的哈希表对象底层使用字典数据结构，哈希对象中的每个键值对都使用一个字典键值对。

在前面介绍压缩列表时，我们介绍过压缩列表是Redis为了节省内存而开发的，是由一系列特殊编码的连续内存块组成的顺序型数据结构，相对于字典数据结构，压缩列表用于元素个数少、元素长度小的场景。其优势在于集中存储，节省空间。

②、编码转换

和上面列表对象使用 ziplist 编码一样，当同时满足下面两个条件时，使用ziplist（压缩列表）编码：

1、列表保存元素个数小于512个

2、每个元素长度小于64字节

不能满足这两个条件的时候使用 hashtable 编码。第一个条件可以通过配置文件中的 set-max-intset-entries 进行修改。

集合对象

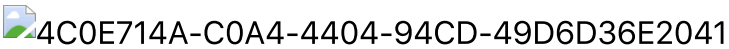
集合对象 set 是 string 类型（整数也会转换成string类型进行存储）的无序集合。注意集合和列表的区别：集合中的元素是无序的，因此不能通过索引来操作元素；集合中的元素不能有重复。

①、编码

集合对象的编码可以是 intset 或者 hashtable。

intset 编码的集合对象使用整数集合作为底层实现，集合对象包含的所有元素都被保存在整数集合中。

hashtable 编码的集合对象使用 字典作为底层实现，字典的每个键都是一个字符串对象，这里的每个字符串对象就是一个集合中的元素，而字典的值则全部设置为 null。这里可以类比Java集合中HashSet 集合的实现，HashSet 集合是由 HashMap 来实现的，集合中的元素就是 HashMap 的 key，而 HashMap 的值都设为 null。



②、编码转换

当集合同时满足以下两个条件时，使用 intset 编码：

1、集合对象中所有元素都是整数

2、集合对象所有元素数量不超过512

不能满足这两个条件的就使用 hashtable 编码。第二个条件可以通过配置文件的 set-max-intset-entries 进行配置。

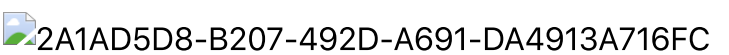
有序集合对象

和上面的集合对象相比，有序集合对象是有序的。与列表使用索引下标作为排序依据不同，有序集合为每个元素设置一个分数（score）作为排序依据。

①、编码

有序集合的编码可以是 ziplist 或者 skiplist。

ziplist 编码的有序集合对象使用压缩列表作为底层实现，每个集合元素使用两个紧挨在一起的压缩列表节点来保存，第一个节点保存元素的成员，第二个节点保存元素的分值。并且压缩列表内的集合元素按分值从小到大的顺序进行排列，小的放置在靠近表头的位置，大的放置在靠近表尾的位置。



字典的键保存元素的值，字典的值则保存元素的分值；跳跃表节点的 object 属性保存元素的成员，跳跃表节点的 score 属性保存元素的分值。

这两种数据结构会通过指针来共享相同元素的成员和分值，所以不会产生重复成员和分值，造成内存的浪费。

说明：其实有序集合单独使用字典或跳跃表其中一种数据结构都可以实现，但是这里使用两种数据结构组合起来，原因是假如我们单独使用 字典，虽然能以 O(1) 的时间复杂度查找成员的分值，但是因为字典是以无序的方式来保存集合元素，所以每次进行范围操作的时候都要进行排序；假如我们单独使用跳跃表来实现，虽然能执行范围操作，但是查找操作有 O(1)的复杂度变为了O(logN)。因此Redis使用了两种数据结构来共同实现有序集合。

②、编码转换

当有序集合对象同时满足以下两个条件时，对象使用 ziplist 编码：

1、保存的元素数量小于128；

2、保存的所有元素长度都小于64字节。

不能满足上面两个条件的使用 skiplist 编码。以上两个条件也可以通过Redis配置文件zset-max-ziplist-entries 选项和 zset-max-ziplist-value 进行修改。

← PREVIOUS POST (HTTP://39.106.3.150/ARCHIVES/JAVA-8)

NEXT POST → (HTTP://39.106.3.150/ARCHIVES/JAVATRAC

FRIENDS


(https://github.com/PowehiEdge)