# 对象

Redis 用到的所有主要数据结构包括简单动态字符串（ SDS ）、双端链表、字典、压缩列表、整数集合等等。

Redis 并没有直接使用这些数据结构来实现键值对数据库，而是基于这些数据结构创建了一个对象系统，这个系统包含字符串对象、列表对象、哈希对象、集合对象和有序集合对象这五种类型的对象，每种对象都用到了至少一种我们前面所介绍的数据结构。

通过这五种不同类型的对象， Redis 可以在执行命令之前，根据对象的类型来判断一个对象是否可以执行给定的命令。使用对象的另一个好处是，我们可以针对不同的使用场景，为对象设置多种不同的数据结构实现，从而优化对象在不同场景下的使用效率。

除此之外， Redis 的对象系统还实现了基于引用计数技术的内存回收机制，当程序不再使用某个对象的时候，这个对象所占用的内存就会被自动择放；另外， Redis 还通过引用计数技术实现了对象共享机制，这一机制可以在适当的条件下，通过让多个数据库键共事同一个对象来节约内存。

最后， Redis 的对象带有访问时间记录信息，该信息可以用于计算数据库键的空转时长，在服务器启用了maxmemory 功能的情况下，空转时长较大的那些键可能会优先被服务器删除。

## 对象的类型与编码

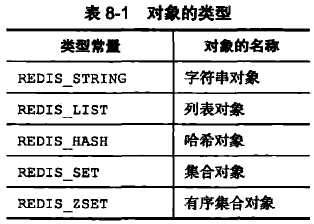
Redis 使用对象来表示数据库中的键和值，每次当我们在Redis 的数据库中新创建一个键值对时，我们至少会创建两个对象，一个对象用作键值对的键（键对象），另一个对象用作键值对的值（值对象）。

Red is 中的每个对象都由一个redisObject 结构表示，该结构中和保存数据有关的三个属性分别是type 属性、encoding 属性和ptr 属性：

|  |
| --- |
| typedef struct redisObject {  // 类型  unsigned type:4;  // 编码  unsigned encoding:4;  // 指向底层实现数据结构的指针  void \*ptr;  // ...  } robj; |

### 类型

对象的type 属性记录了对象的类型，这个属性的值可以是表8-1 列出的常量的其中一个。



对于Redis 数据库保存的键值对来说，键总是一个字符串对象，而值则可以是字符串对象、列表对象、哈希对象、集合对象或者有序集合对象的其中一种，因此：

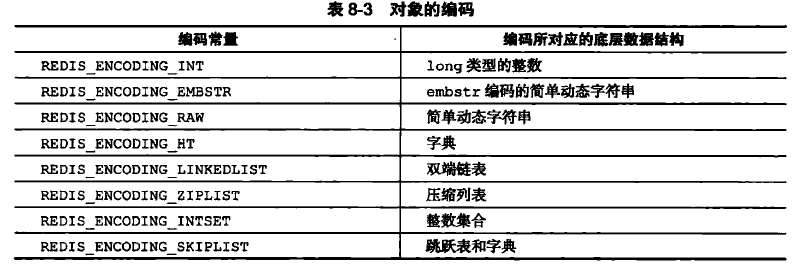
* 当我们称呼一个数据库键为“字符串键”时，我们指的是“这个数据库键所对应的值为字符串对象”；
* 当我们称呼一个键为“列表键”时，我们指的是“这个数据库键所对应的值为列表对象”。

TYPE 命令的实现方式也与此类似，当我们对一个数据库键执行TYPE 命令时，命令返回的结果为数据库键对应的值对象的类型，而不是键对象的类型。表8-2 列出了TYPE 命令在面对不同类型的值对象时所产生的输出。

### 编码和底层实现

对象的ptr 指针指向对象的底层实现数据结构，而这些数据结构由对象的encoding属性决定。

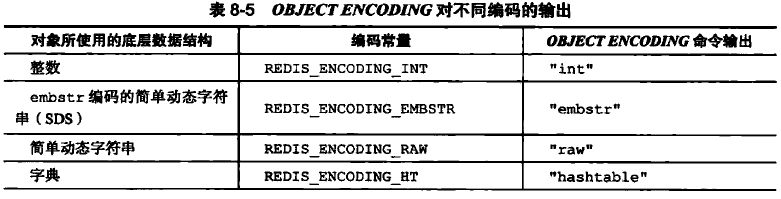
encoding 属性记录了对象所使用的编码，也即是说这个对象使用了什么数据结构作为对象的底层实现，这个属性的值可以是表8-3 列出的常量的其中一个。

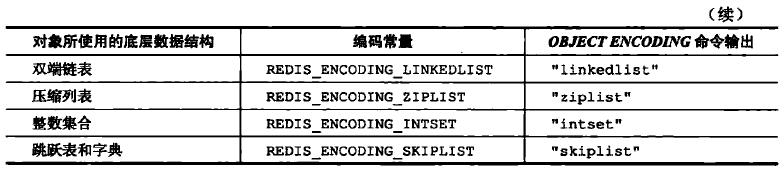


每种类型的对象都至少使用了两种不同的编码，表8-4 列出了每种类型的对象可以使用的编码。



使用OBJECT ENCODING 命令可以查看一个数据库键的值对象的编码，表8-5 列出了不同编码的对象所对应的OBJECT ENCODING 命令输出。





通过encoding 属性来设定对象所使用的编码，而不是为特定类型的对象关联一种固定的编码，极大地提升了Redis 的灵活性和效率，因为Redis 可以根据不同的使用场景来为一个对象设置不同的编码，从而优化对象在某一场景下的效率。

举个例子，在列表对象包含的元素比较少时， Redis 使用压缩列表作为列表对象的底层实现：

* 因为压缩列表比双端链表更节约内存，并且在元素数量较少时，在内存中以连续块方式保存的压缩列表比起双端链表可以更快被载入到缓存中；
* 随着列表对象包含的元素越来越多，使用压缩列表来保存元素的优势逐渐消失时，对象就会将底层实现从压缩列表转向功能更强、也更适合保存大量元素的双端链表上面；

其他类型的对象也会通过使用多种不同的编码来进行类似的优化。

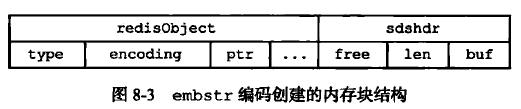
## 字符串对象

字符串对象的编码可以是int、raw 或者embstr 。如果一个字符串对象保存的是整数值，并且这个整数值可以用long 类型来表示，那么字符串对象会将整数值保存在字符串对象结构的ptr 属性里面（将void \*转换成long )，并将字符串对象的编码设置为int。

如果字符串对象保存的是一个字符串值，并且这个字符串值的长度大于32 宇节，那么字符串对象将使用一个简单动态字符串（ SDS ）来保存这个字符串值，并将对象的编码设置为raw。

如果字符串对象保存的是一个字符串值，并且这个字符串值的长度小于等于32 字节，那么字符串对象将使用ernbstr 编码的方式来保存这个字符串值。

ernbstr 编码是专门用于保存短字符串的一种优化编码方式，这种编码和raw 编码一样，都使用redisObject 结构和sdshdr 结构来表示字符串对象，但raw 编码会调用两次内存分配函数来分别创建redisObject 结构和sdshdr 结构，而embstr 编码则通过调用一次内存分配函数来分配一块连续的空间，空间中依次包含redisObject 和sdshdr两个结构，如图8-3 所示。



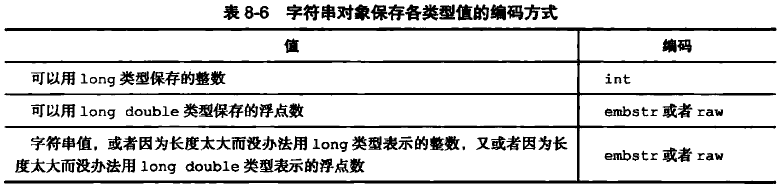
ernbstr 编码的字符串对象在执行命令时，产生的效果和raw 编码的字符串对象执行命令时产生的效果是相同的，但使用ernbstr 编码的字符串对象来保存短字符串值有以下好处：

* ernbstr 编码将创建字符串对象所需的内存分配次数从raw 编码的两次降低为一次。
* 释放ernbstr 编码的字符串对象只需要调用一次内存释放函数，而释放raw 编码的字符串对象需要调用两次内存释放函数。
* 因为ernbstr 编码的字符串对象的所有数据都保存在一块连续的内存里面，所以这种编码的字符串对象比起raw 编码的字符串对象能够更好地利用缓存带来的优势。

最后要说的是，可以用long double 类型表示的浮点数在Redis 中也是作为宇符串值来保存的。如果我们要保存一个浮点数到字符串对象里面，那么程序会先将这个浮点数转换成字符串值，然后再保存转换所得的字符串值。

在有需要的时候，程序会将保存在字符串对象里面的字符串值转换回浮点数值，执行某些操作，然后再将执行操作所得的撑点数值转换回字符串值，并继续保存在字符串对象里面。

表8-6 总结并列出了宇符串对象保存各种不同类型的值所使用的编码方式。



### 编码的转换

int 编码的字符串对象和embstr 编码的字符串对象在条件满足的情况下，会被转换为raw 编码的字符串对象。

对于int 编码的字符串对象来说，如果我们向对象执行了一些命令，使得这个对象保存的不再是整数值，而是一个字符串值，那么字符串对象的编码将从int 变为raw 。

另外，因为Redis 没有为embstr 编码的字符串对象编写任何相应的修改程序（只有int 编码的字符串对象和raw 编码的字符串对象有这些程序），所以embstr 编码的字符串对象实际上是只读的。当我们对embstr 编码的字符串对象执行任何修改命令时，程序会先将对象的编码从embstr 转换成raw ，然后再执行修改命令。因为这个原因， embstr编码的字符串对象在执行修改命令之后，总会变成一个主aw 编码的字符串对象。

### 字符串命令的实现

因为字符串键的值为字符串对象，所以用于字符串键的所有命令都是针对字符串对象来构建的，表8-7 列举了其中一部分字符串命令，以及这些命令在不同编码的字符串对象下的实现方法。



## 列表对象

列表对象的编码可以是ziplist 或者linkedlist 。ziplist 编码的列表对象使用压缩列表作为底层实现，每个压缩列表节点（ entry）保存了一个列表元素。另一方面， linkedlist 编码的列表对象使用双端链表作为底层实现，每个双端链表节点（ node ）都保存了一个字符串又如象，而每个字符串对象都保存了一个列表元素。

注意， linkedlist 编码的列表对象在底层的双端链表结构中包含了多个字符串对象，这种嵌套字符串对象的行为在哈希对象、集合对象和有序集合对象中都会出现，字符串对象是Redis 五种类型的对象中唯一一种会被其他四种类型对象嵌套的对象。

### 编码转换

当列表对象可以同时满足以下两个条件时，列表对象使用ziplist 编码：

* 列表对象保存的所有字符串元素的长度都小于64 字节；
* 列表对象保存的元素数量小于512 个；不能满足这两个条件的列表对象需要使用linkedlist 编码。

以上两个条件的上限值是可以修改的，具体请看配直文件中关于list-max-ziplistvalue选项和list-max-ziplist-entries 选项的说明。

对于使用ziplist 编码的列表对象来说，当使用ziplist 编码所需的两个条件的任意一个不能被满足时，对象的编码转换操作就会被执行，原本保存在压缩列表里的所有列表元素都会被转移并保存到双端链表里面，对象的编码也会从ziplist 变为linkedlist 。

### 列表命令的实现

因为列表键的值为列表对象，所以用于列表键的所有命令都是针对列表对象来构建的，表8-8 列出了其中一部分列表键命令，以及这些命令在不同编码的列表对象下的实现方法。



## 哈希对象

哈希对象的编码可以是ziplist 或者hashtable 。ziplist 编码的哈希对象使用压缩列表作为底层实现，每当有新的键值对要加入到哈希对象时，程序会先将保存了键的压缩列表节点推入到压缩列表表尾，然后再将保存了值的

压缩列表节点推入到压缩列表表尾，因此：

* 保存了同一键值对的两个节点总是紧挨在一起，保存键的节点在前，保存值的节点在后；
* 先添加到哈希对象中的键值对会被放在压缩列表的表头方向，而后来添加到哈希对象中的键值对会被放在压缩列表的表尾方向。

另一方面， hash table 编码的哈希对象使用宇典作为底层实现，哈希对象中的每个键值对都使用一个字典键值对来保存z

* 字典的每个键都是一个字符串对象，对象中保存了键值对的键。
* 字典的每个值都是一个字符串对象，对象中保存了键值对的值。

### 编码转换

当哈希对象可以同时满足以下两个条件时，哈希对象使用ziplist 编码：

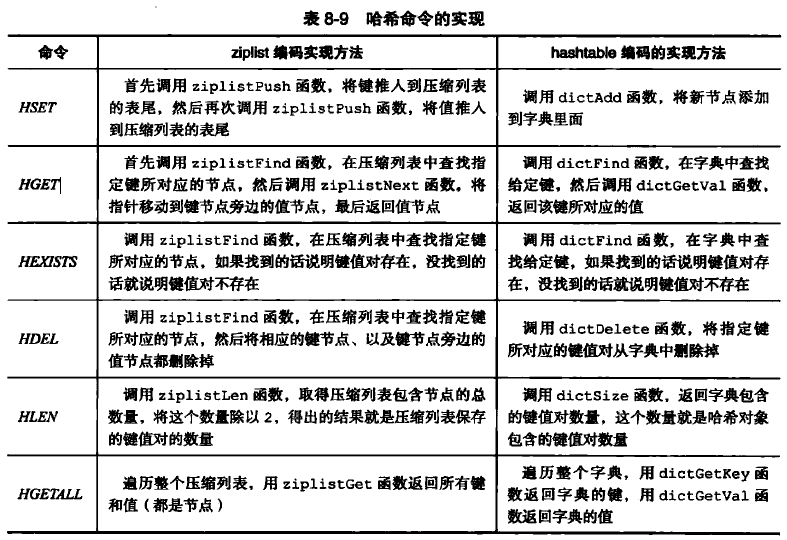
* 哈希对象保存的所有键值对的键和值的字符串长度都小于64 字节E
* 哈希对象保存的键值对数量小于512 个；不能满足这两个条件的晗希对象需要使用hash table 编码。

注意：这两个条件的上限值是可以修改的，具体请看配置文件中关于hash-max-ziplistvalue选项和hash-max-ziplist-entries 选项的说明。

对于使用ziplist 编码的列表对象来说，当使用ziplist 编码所需的两个条件的任意一个不能被满足时，对象的编码转换操作就会被执行，原本保存在压缩列表里的所有键值对都会被转移并保存到字典里面，对象的编码也会从ziplist 变为hash table 。

### 哈希命令的实现

因为哈希键的值为晴希对象，所以用于哈希键的所有命令都是针对哈希对象来构建的，表8-9 列出了其中一部分哈希键命令，以及这些命令在不同编码的哈希对象下的实现方法。



## 集合对象

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里面。另一方面， hash table 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，每个字符串对象包含了一个集合元素，而字典的值则全部被设置为NULL。

### 编码的转换

当集合对象可以同时满足以下两个条件时，对象使用intset 编码：

* 集合对象保存的所有元素都是整数值；
* 集合对象保存的元素数量不超过512 个。

不能满足这两个条件的集合对象需要使用hash table 编码。

注意：第二个条件的上限值是可以修改的，具体请看配置文件中关于set-max-intsetentries选项的说明。

对于使用intset 编码的集合对象来说，当使用intset 编码所需的两个条件的任意一个不能被满足时，就会执行对象的编码转换操作，原本保存在整数集合中的所有元素都会被转移并保存到字典里面，并且对象的编码也会从intset 变为hash table 。

### 集合命令的实现

因为集合键的值为集合对象，所以用于集合键的所有命令都是针对集合对象来构建的，表8-10 列出了其中一部分集合键命令，以及这些命令在不同编码的集合对象下的实现方法。

