Redis是一个基于内存的数据库，所有的数据都存储在内存中。所以如何优化存储，减少内存空间占用是一个非常重要的话题。精简键名和键值是最直观的减少内存占用的方式，如将键名very.important.person:20改成VIP:20。

但有时仅凭精简键名和键值所减少的空间并不足以满足需求，这时就需要根据Redis内部编码规则来节省更多的空间。

Redis为每种数据类型都提供了两种内部编码方式，以散列类型为例，散列类型是通过散列表实现的，这样就可以实现O(1)时间复杂度的查找、赋值操作，然而当键中元素很少的时候，O(1)的操作并不会比O(n)有明显的性能提高，所以这种情况下Redis会采用一种节约内存但性能稍差（获取元素的时间复杂度为O(n)）的内部编码方式。

内部编码方式的选择对于使用者来说是透明的，Redis会根据实际情况自动调整。当键中元素变多时Redis会自动将该键的内部编码方式转换成散列表。如果想查看一个键的内部编码方式可以使用”object encoding”命令，例如：

127.0.0.1:6379> lpush list a

(integer) 2

127.0.0.1**:**6379**>** object encoding list

"ziplist"

Redis的每个value都是使用一个redisObject结构体保存的，redisObject的定义如下：

**typedef** struct redisObject

**{**

unsigned type**:**4**;**

unsigned encoding**:**4**;**

unsigned lru**:**24**;** */\* lru time (relative to server.lruclock) \*/*

int refcount**;**

void **\***ptr**;**

**}** robj**;**

其中type字段表示的是value的数据类型，取值可以是如下内容：

*/\* Object types \*/*

#define REDIS\_STRING 0

#define REDIS\_LIST 1

#define REDIS\_SET 2

#define REDIS\_ZSET 3

#define REDIS\_HASH 4

encoding字段表示的就是value的内部编码方式，取值可以是：

#define REDIS\_ENCODING\_RAW 0 /\* Raw representation \*/

#define REDIS\_ENCODING\_INT 1 /\* Encoded as integer \*/

#define REDIS\_ENCODING\_HT 2 /\* Encoded as hash table \*/

#define REDIS\_ENCODING\_ZIPMAP 3 /\* Encoded as zipmap \*/

#define REDIS\_ENCODING\_LINKEDLIST 4 /\* Encoded as regular linked list \*/

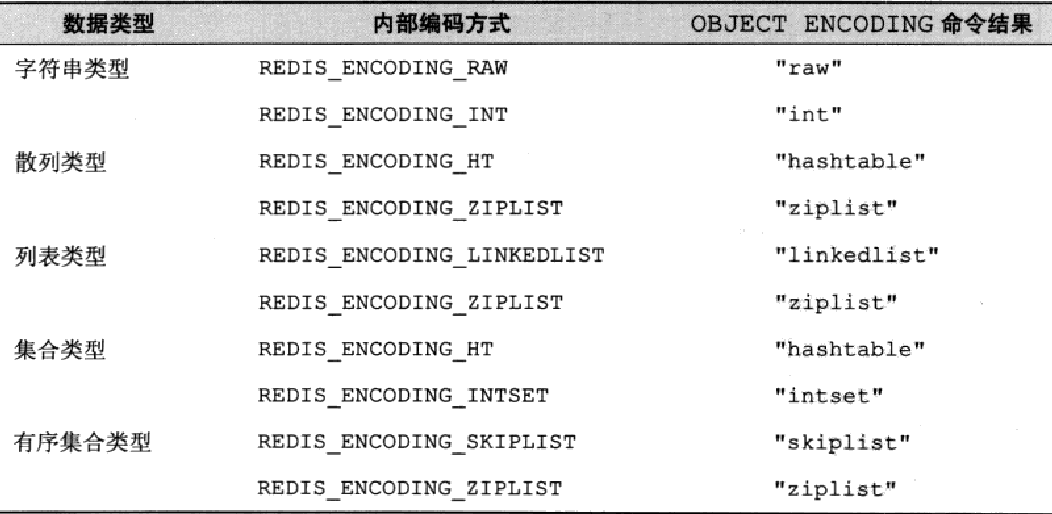
#define REDIS\_ENCODING\_ZIPLIST 5 /\* Encoded as ziplist \*/

#define REDIS\_ENCODING\_INTSET 6 /\* Encoded as intset \*/

#define REDIS\_ENCODING\_SKIPLIST 7 /\* Encoded as skiplist \*/

#define REDIS\_ENCODING\_EMBSTR 8 /\* Embedded sds string encoding \*/

各个数据类型可能采用的内部编码方式以及相应的object encoding命令执行结果如下表所示：



**一：字符串类型**

Redis使用一个sdshdr类型的变量来存储字符串，而redisObject的ptr字段指向的是该变量的地址。sdshdr的定义如下：

struct sdshdr

**{**

unsigned int len**;**

unsigned int free**;**

char buf**[];**

**};**

注意，sizeof(struct sdshdr) == 8，也就是说，最后的buf成员未计入。

其中len字段表示的是字符串的长度，free字段表示buf中的剩余空间，而buf字段存储的才是字符串的内容。

所以当执行”set key foobar”时，存储键值需要占用的空间是：

sizeof(redisObject) + sizeof(sdshdr) + strlen(“foobar”) = 16 + 8 + 6 = 30，如下图所示：



当键值内容可以用一个64位有符号整数表示时，Redis会将键值转换成long类型来存储。如”set key 12345”，实际占用的空间是sizeof(redisObject) = 16字节，比存储”foobar”节省了一半的存储空间，如下图所示：



redisObject中的refcount字段存储的是该键值被引用的数量，即一个键值可以被多个键引用。Redis启动后会预先建立10000个redis0bject类型变量作为共享对象，分别存储从0到9999这些数字。如果要设置的字符串键值在这10000个数字内（如”set key1 123”），则可以直接引用共享对象而不用再建立一个redisObject了，也就是说存储键值占用的空间是0字节，当执行了”set key1 123”和”set key2 123”后，key1和key2两个键都直接引用了一个己经建立好的共享对象，节省了存储空间。如下图所示：



由此可见，使用字符串类型键存储对象ID这种小数字是非常节省存储空问的，Redis只需存储键名和一个对共享对象的引用即可。

注意，当通过配置文件参数maxmemory设置了Redis可用的最大空间大小时，Redis不会使用共享对象，因为对于每一个键值都需要使用一个RedisObject来记录其LRU信息。

**二：散列类型**

散列类型的内部编码方式可能是REDIS\_ENCODING\_HT或REDIS\_ENCODING\_ZIPLIST。在配置文件中可以定义使用REDIS\_ENCODING\_ZIPLIST方式编码散列类型的时机：

hash**-**max**-**ziplist**-**entries 512

hash**-**max**-**ziplist**-**value 64

当散列类型键的字段个数少于hash-max-ziplist-entries，且每个字段名和字段值的长度都小于hash-max-ziplist-value时（字节），Redis就会使用REDIS\_ENCODING\_ZIPLIST来存储该键，否则就会使用REDIS\_ENCODING\_HT，转换过程是透明的，每当键值变更后Redis都会自动判断是否满足条件来完成转换。

REDIS\_ENCODING\_HT编码即散列表，可以实现O(1)时间复杂度的赋值取值等操作，其字段和字段值都是使用redisObject存储的，所以前面讲到的字符串类型键值的优化方法同样适用于散列类型键的字段和字段值。

Redis的key-value存储也是通过散列表实现的，与REDIS\_ENCODING\_HT编码方式类似，但键名并非使用redisObject存储，所以健名”123456”并不会比”abcdef”占用更少的空间。之所以不对键名进行优化是因为绝大多数情况下键名都不会是纯数字。

REDIS\_ENCODING\_ZIPLIST编码类型是一种紧凑的编码格式，它牺牲了部分读取性能以换取极高的空间利用率，适合在元素较少时使用。该编码类型同样还在列表类型和有序集合类型中使用。REDIS\_ENCODING\_ZIPLIST编码结构如下图所示：



其中zlbytes是uint32\_t类型，表示整个结构占用的空间。zltail也是uint32\_t类型。表示到最后一个元素的偏移，zltail使得程序可以直接定位到尾部元素而无需遍历整个结构，执行从尾部弹出（对列表类型而言）等操作时速度更快。zllen是uint16\_t类型，存储元索的数量。zlend是一个单字节标识，标记结构的末尾，值永远是255。

在REDIS\_ENCODING\_ZIPLIST中每个元素由4个部分组成。第1部分用来存储前一个元素的大小以实现倒序查找；第2,3个部分分别是元素的编码类型和元素大小，第四个部分是元素的实际内容。

使用REDIS\_ENCODING\_ZIPLIST编码存储散列类型时元素的排列方式是：元素1存储字段1，元素2存储字段值1，依次类推，如下图所示：



例如，执行命令”hset hkey foo bar”命令后，下次需要执行”hset hkey foo anthervalue”时Redis需要从头开始找到值为foo的元素（查找时每次都会跳过一个元素以保证只查找字段名），找到后删除其下一个元素，井将新值anothervalue插入。删除和插入都需要移动后面的内存数据，并且查找操作也需要遍历才能完成，可想而知当散列键中数据多时性能将很低，所以不宜将hash-max-ziplist-entries和hash-max-ziplist-value两个参数设置得很大。

**三：列表类型**

列表类型的内部编码方式可能是REDIS\_ENCODING\_LINKEDLIST或REDIS\_ENCODING\_ZIPLIST。同样，也可以在配置文件中定义使用REDIS\_ENCODING\_ZIPLIST方式编码的时机：

list**-**max**-**ziplist**-**entries 512

list**-**max**-**ziplist**-**value 64

具体转换方式和散列类型一样，这里不再赘述。

REDIS\_ENCODING\_LINKEDLIST编码方式即双向链表，链表中的每个元素是用redisObject方式存储的，所以此种编码方式下元素值的优化方法与字符串类型的键值相同。

使用REDIS\_ENCODING\_ZIPLIST编码方式时具体的表现和散列类型一样，由于该编码方式同样支持倒序访问，所以采用此种编码方式时获取两端的数据依然较快。

**四：集合类型**

集合类型的内部编码方式可能是REDIS\_ENCODING\_HT或REDIS\_ENCODING\_INTSET。当集合中的所有元素都是整数且元素的个数小于配置文件中的set-max-intset-entries（默认是512）时，Redis会使用REDIS\_ENCODING\_INTSET编码存储该集合，否则会使用REDIS\_ENCODING\_HT来存储。

REDIS\_ENCODING\_INTSET编码存储结构体intset的定义是：

**typedef** struct intset **{**

uint32\_t encoding**;**

uint32\_t length**;**

int8\_t contents**[];**

**}** intset**;**

其中contents存储的是集合中的元素值，根据encoding的不同，每个元素占用的字节大小不同。默认的encoding是INTSET\_ENC\_INT16（即2个字节），当新增加的整数元素无法使用2个字节表示时，Redis会将该集合的encoding升级为INTSET\_ENC\_INT32 （即4个字节），并调整之前所有元素的位置和长度，同样集合的encoding还可升级为INTSET\_ENC\_INT64（即8个字节）。

REDIS\_ENCODING\_INTSET编码以有序的方式存储元素（所以使用smembers命令获得的结果是有序的），使得可以使用二分算法查找元素。然而无论是添加还是删除元素，Redis都需要调整后面元素的内存位置，所以当集合中的元素太多时性能较差。

当新增加的元素不是整数，或集合中的元素数量超过了set-max-intset-entries参数时，Redis会自动将该集合的存储结构转换成REDIS\_ENCODING\_HT。

注意当集合的存储结构转换成REDIS\_ENCODING\_HT后，即使将集合中的所有非整数元素删除，Redis也不会自动将存储结构转换回REDIS\_ENCODING\_INTSET。因为如果要支持自动回转，就意味着Redis在每次删除元素时都需要遍历集合中的健来判断是否可以转换回原来的编码，这会使得删除元素变成了时间复杂度为O(n)的操作。

**五：有序集合类型**

有序集合类型的内部编码方式可能是REDIS\_ENCODING\_SKIPLIST或REDIS\_ENCODING\_ZIPLIST。同样在配置文件中可以定义使用REDIS\_ENCODING\_ZIPLIST方式编码的时机：

zset**-**max**-**ziplist**-**entries 128

zset**-**max**-**ziplist**-**value 64

具体转换规则和散列类型及列表类型一样，不再赘述。

当编码方式是REDIS\_ENCODING\_SKIPLIST时，Redis使用散列表和跳跃列表(skip

list)两种数据结构来存储有序集合类型键值，其中散列表用来存储元素值与元素分数的映射关系以实现O(1)时间复杂度的zscore等命令。跳跃列表用来存储元素的分数及其到元素值的映射以实现排序的功能。Redis对跳跃列表的实现进行了几点修改，其中包括允许跳跃列表中的元素（即分数）相同，还有为跳跃列表每个节点增加了指向前一个元素的指针以实现倒序查找。

采用此种编码方式时，元素值是使用redisObject存储的，所以可以使用字符串类型键值的优化方式优化元素值，而元素的分数是使用double类型存储的。

使用REDIS\_ENCODING\_ZIPLIST编码时，有序集合存储的方式按照“元素1的值，元

素1的分数，元素2的值，元素2的分数”的顺序排列，并且分数是有序的。