Redis学习笔记-基础篇

参考资料：《Redis设计与实现》、《Redis使用手册》

1. 五大数据结构

先来看看对象的编码

表 8-3　对象的编码

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

1.1 字符串

1.1.1 数据结构

字符串的编码可以说int，raw或者embstr

int

> 如果一个字符串保存的是整数值，并且该整数值可以用long来表示，那么字符串对象会将整数保存在字符串对象结构的ptr属性里。

```
> SET number 10086
```

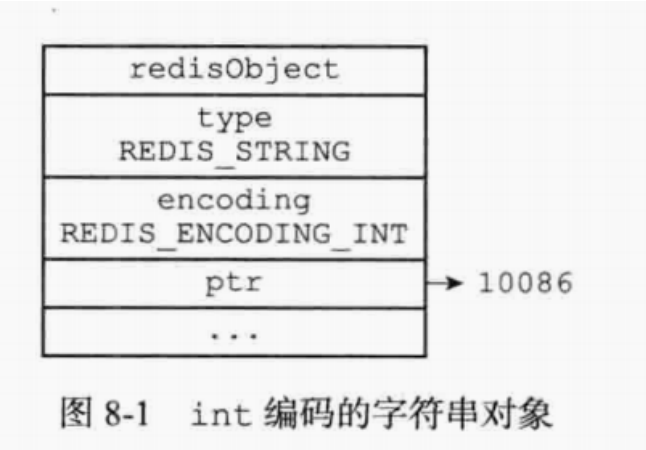

图 8-1　int 编码的字符串对象

raw

> 如果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字节，那么将使用一个简单字符串（SDS）来保存这个字符串的值

```
> SET story "Long, long ago there lived a king"
```

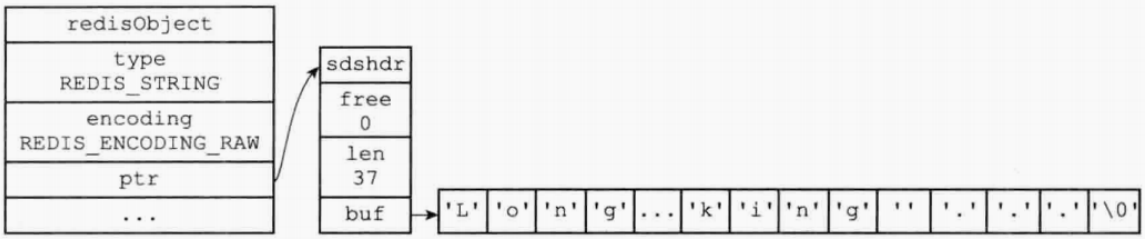

图 8-2　raw 编码的字符串对象

该结构分两次分别创建了redisObject和sdshdr，使用两次内存分配，释放的时候也要调用两次内存释放。

embstr

> 如果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32字节，那么将使用embstr来保存这个字符串的值

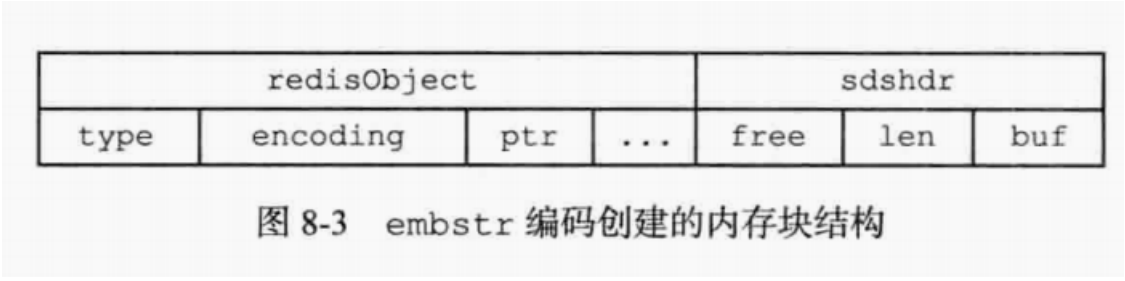

图 8-3　embstr 编码创建的内存块结构

好处（与raw相比）：

1. 只调用一次内存分配函数来分配一块连续的内存空间
2. 只调用一次内存释放函数

```
> SET msg "hello"
```

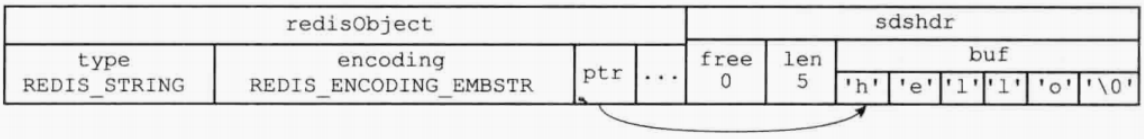

图 8-4　embstr 编码的字符串对象

总结：

表 8-6　字符串对象保存各类型值的编码方式

值	编码
可以用 long 类型保存的整数	int
可以用 long double 类型保存的浮点数	embstr 或者 raw
字符串值，或者因为长度太大而没办法用 long 类型表示的整数，又或者因为长度太大而没办法用 long double 类型表示的浮点数	embstr 或者 raw

1.1.2 命令

```
#查看该键是否存在
EXISTS key

#设置字符串（如果已经有该key，则覆盖，无视类型）
SET key value
```

```
#获取字符串
GET key

#获取多个字符串（如果某个键不存在，该值就返回nil，其他存在的值返回正常）
MGET key1 key2

#设置多个字符串
MSET key1 v1 key2 v2

#范围获取值（获取子字符串）（也包含end）
GETRANGE KEY_NAME start end
GETRANGE key 0 -1
GETRANGE key 0 3

#返回字符串长度（不存在返回0）
STRLEN key

#范围设置值，返回被修改后的字符串长度
SETRANGE  KEY_NAME OFFSET VALUE
    redis 127.0.0.1:6379> SET key1 "Hello World"
    OK
    redis 127.0.0.1:6379> SETRANGE key1 6 "Redis"
    (integer) 11
    redis 127.0.0.1:6379> GET key1
    "Hello Redis"

#设置多个字符串（不存在才设置），如果有一个失败，返回0，所有设置无效（原子性操作）
MSETNX key1 v1 key2 v2

#指定key不存在时，为key设置指定的值。成功：1，失败：0
SETNX key value

#设置值和过期时间（单位为SECONDS）。若该键已存在，则覆盖
SETEX KEY_NAME TIMEOUT VALUE
SETEX key 100 value

#设置值和过期时间（单位为MILLISECONDS）。若该键已存在，则覆盖
PSETEX key 100 value

#返回key的旧值，没有就返回nil
GETSET key value

#自增1。如果key不存在，先将key的值设为0，再自增。 DECR同理
INCR key

#自增n。如果key不存在，先将key的值设为0，再自增n。 DECRBY同理
INCRBY key n

#加上某个浮点数，如果key不存在，先将key的值设为0，再加。 没有DECRBYFLOAR
INCRBYFLOAT key 3.14

#追加
APPEND key value_A

#位操作
SETBIT KEY offset 0/1
SETBIT key 10086 1
```

```
#获取key里第10086个字节的位，若超出或者不存在该key返回0，或者该位本来就为0
GETBIT key 10086

#获取位为1的个数
BIGCOUNT key
#获取第一个字节的所有位为1的个数
BIGCOUNT key 0 1

#Bitmaps的运算。bitop是一个复合操作， 它可以做多个Bitmaps的and（交集） 、 or（并集） 、 not（非） 、 xor（异或） 操作并将结果保存在destkey中。
bitop op destkey key[key....]
```

SETBIT&GETBIT

表 8-7　字符串命令的实现

命令	int 编码的实现方法	embstr 编码的实现方法	raw 编码的实现方法
SET	使用 int 编码保存值	使用 embstr 编码保存值	使用 raw 编码保存值
GET	拷贝对象所保存的整数值，将这个拷贝转换成字符串值，然后向客户端返回这个字符串值	直接向客户端返回字符串值	直接向客户端返回字符串值
APPEND	将对象转换成 raw 编码，然后按 raw 编码的方式执行此操作	将对象转换成 raw 编码，然后按 raw 编码的方式执行此操作	调用 sdscatlen 函数，将给定字符串追加到现有字符串的末尾
INCRBYFLOAT	取出整数值并将其转换成 long double 类型的浮点数，对这个浮点数进行加法计算，然后将得出的浮点数结果保存起来	取出字符串值并尝试将其转换成 long double 类型的浮点数，对这个浮点数进行加法计算，然后将得出的浮点数结果保存起来。如果字符串值不能被转换成浮点数，那么向客户端返回一个错误	取出字符串值并尝试将其转换成 long double 类型的浮点数，对这个浮点数进行加法计算，然后将得出的浮点数结果保存起来。如果字符串值不能被转换成浮点数，那么向客户端返回一个错误
INCRBY	对整数值进行加法计算，得出的计算结果会作为整数被保存起来	embstr 编码不能执行此命令，向客户端返回一个错误	raw 编码不能执行此命令，向客户端返回一个错误
DECRBY	对整数值进行减法计算，得出的计算结果会作为整数被保存起来	embstr 编码不能执行此命令，向客户端返回一个错误	raw 编码不能执行此命令，向客户端返回一个错误
STRLEN	拷贝对象所保存的整数值，将这个拷贝转换成字符串值，计算并返回这个字符串值的长度	调用 sdslen 函数，返回字符串的长度	调用 sdslen 函数，返回字符串的长度
SETRANGE	将对象转换成 raw 编码，然后按 raw 编码的方式执行此命令	将对象转换成 raw 编码，然后按 raw 编码的方式执行此命令	将字符串特定索引上的值设置为给定的字符
GETRANGE	拷贝对象所保存的整数值，将这个拷贝转换成字符串值，然后取出并返回字符串指定索引上的字符	直接取出并返回字符串指定索引上的字符	直接取出并返回字符串指定索引上的字符

1.2 列表

1.2.1 数据结构

列表对象的编码可以是ziplist或者linkedlist（都是头插法）

ziplist

> ziplist的编码的对象使用压缩列表作为底层实现，每个压缩列表节点保存了一个对象
>
> 当满足：

> 1. 列表对象保存的所有字符串元素的长度都小于64字节
> 2. 列表对象保存的元素数量小于512个
>
> 时，使用ziplist，否则使用linkedlist
>
> ***以上参数可以在配置文件的list-max-ziplist-value和list-max-ziplist-entries中修改***

要注意压缩列表的连锁更新问题（具体的数据结构信息可以在《Redis设计与实现》第七章查看）

```
RPUSH numbers 1 "three" 5
```

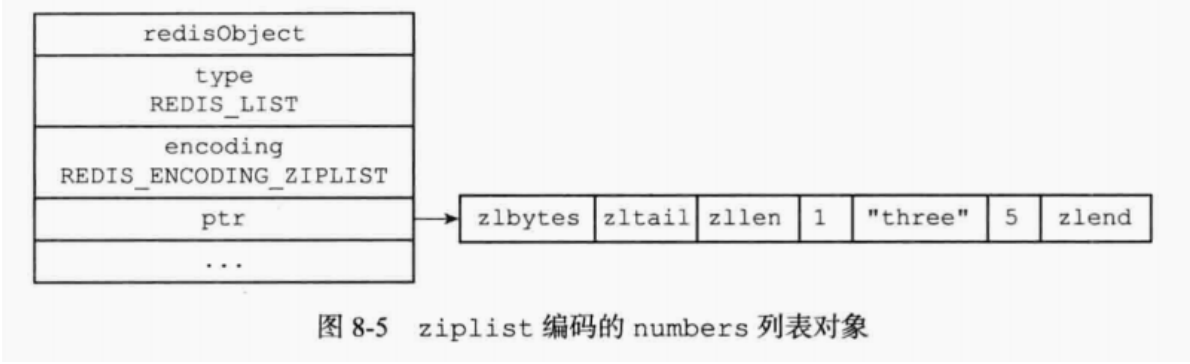

图 8-5　ziplist 编码的 numbers 列表对象

linkedlist

> linkedlist编码的列表对象使用双向链表作为底层实现，每个链表节点都保存了一个字符串对象，而每个字符串对象都保存了一个列表元素
>
> 不使用ziplist时，就使用linkedlist

```
RPUSH numbers 1 "three" 5
```

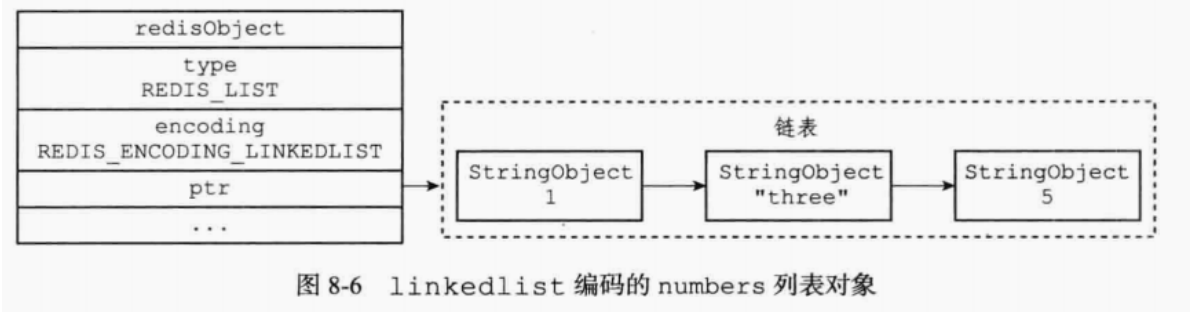

图 8-6　linkedlist 编码的 numbers 列表对象

1.2.2 命令

```
#从左到右在表头插入一个或多个值（返回列表的元素值，可以加入重复元素）
LPUSH key v1 v2

#从左到右在表头插入一个或多个值（列表为空或者key不存在时不允许插入）
LPUSHX key v1 v2

#从左到右在表尾插入一个或多个值（返回列表的元素值，可以加入重复元素）
RPUSH key v1 v2

#从左到右在表尾插入一个或多个值（列表为空或者key不存在时不允许插入）
RPUSHX key v1 v2

#移除并返回表头元素
LPOP key
```

```
#移除并返回表尾元素
RPOP key

#一个列表的表尾弹出一个元素插入到另一个列表的表头
RPOPLPUSH source destination

#根据参数 count 的值，移除列表中与参数 value 相等的元素。
LREM key count value
#当count = 0时，移除列表中所有与该值相等的元素
LREM key 0 v
#当count > 0时，从左到右移除列表count个与该值相等的元素
#当count < 0时，从右到左移除列表|count|个与该值相等的元素

#返回链表key的长度
LLEN key

#根据下标获取列表元素
LINDEX key 1
LINDEX key -1

#将值 value 插入到列表 key 当中，位于值 pivot 之前或之后。（列表不存在或pivot不存在返回0/-1）
LINSERT key BEFORE|AFTER pivot value
#将v2插在v1之前
LINSERT key BEFORE v1 v2

#根据下标设置值将列表 key 下标为 index 的元素的值设置为 value 。
LSET key index value

#返回列表 key 中指定区间内的元素，区间以偏移量 start 和 stop (也包含)指定。
LRANGE key start stop

#对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。
LTRIM key start stop

#当给定列表内没有任何元素可供弹出的时候，连接将被 BLPOP 命令阻塞，直到等待超时或发现可弹出元素为止。（单位：SECONDS）
#当给定多个 key 参数时，按参数 key 的先后顺序依次检查各个列表，弹出第一个非空列表的头元素
BLPOP key [key …] timeout

#与BLPOP一致，不过弹出的是尾部元素
BRPOP key [key …] timeout

#source列表的尾部弹出一个元素，加入到destination列表的头部，超时阻塞
BRPOPLPUSH source destination timeout
```

表 8-8　列表命令的实现

命令	ziplist 编码的实现方法	linkedlist 编码的实现方法
LPUSH	调用 ziplistPush 函数，将新元素推入到压缩列表的表头	调用 listAddNodeHead 函数，将新元素推入到双端链表的表头
RPUSH	调用 ziplistPush 函数，将新元素推入到压缩列表的表尾	调用 listAddNodeTail 函数，将新元素推入到双端链表的表尾
LPOP	调用 ziplistIndex 函数定位压缩列表的表头节点，在向用户返回节点所保存的元素之后，调用 ziplistDelete 函数删除表头节点	调用 listFirst 函数定位双端链表的表头节点，在向用户返回节点所保存的元素之后，调用 listDelNode 函数删除表头节点
RPOP	调用 ziplistIndex 函数定位压缩列表的表尾节点，在向用户返回节点所保存的元素之后，调用 ziplistDelete 函数删除表尾节点	调用 listLast 函数定位双端链表的表尾节点，在向用户返回节点所保存的元素之后，调用 listDelNode 函数删除表尾节点
LINDEX	调用 ziplistIndex 函数定位压缩列表中的指定节点，然后返回节点所保存的元素	调用 listIndex 函数定位双端链表中的指定节点，然后返回节点所保存的元素
LLEN	调用 ziplistLen 函数返回压缩列表的长度	调用 listLength 函数返回双端链表的长度
LINSERT	插入新节点到压缩列表的表头或者表尾时，使用 ziplistPush 函数；插入新节点到压缩列表的其他位置时，使用 ziplistInsert 函数	调用 listInsertNode 函数，将新节点插入到双端链表的指定位置
LREM	遍历压缩列表节点，并调用 ziplistDelete 函数删除包含了给定元素的节点	遍历双端链表节点，并调用 listDelNode 函数删除包含了给定元素的节点
LTRIM	调用 ziplistDeleteRange 函数，删除压缩列表中所有不在指定索引范围内的节点	遍历双端链表节点，并调用 listDelNode 函数删除链表中所有不在指定索引范围内的节点
LSET	调用 ziplistDelete 函数，先删除压缩列表指定索引上的现有节点，然后调用 ziplistInsert 函数，将一个包含给定元素的新节点插入到相同索引上面	调用 listIndex 函数，定位到双端链表指定索引上的节点，然后通过赋值操作更新节点的值

1.3 哈希

1.3.1 数据结构

哈希对象的编码可以是ziplist或者hashtable。

ziplist

> 当
>
> 1. 哈希对象保存所有值键值对的键和值的字符串长度都小于64字节
> 2. 哈希对象保存的键值对数量小于512个
>
> 使用ziplist，否则使用hashtable。
>
> ***以上参数可以在配置文件的hash-max-ziplist-value和hash-max-ziplist-enties中修改***

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

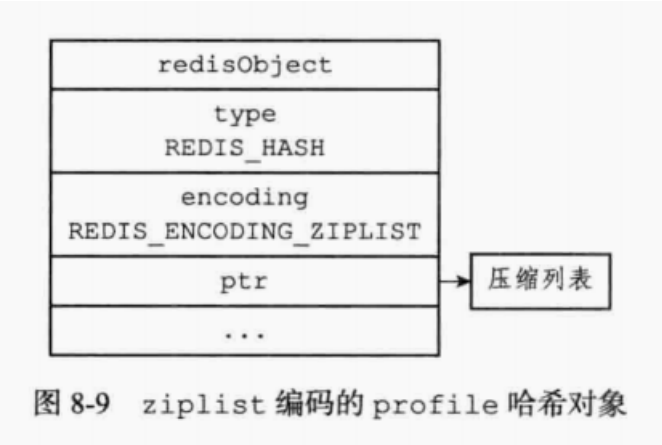

图 8-9　ziplist 编码的 profile 哈希对象

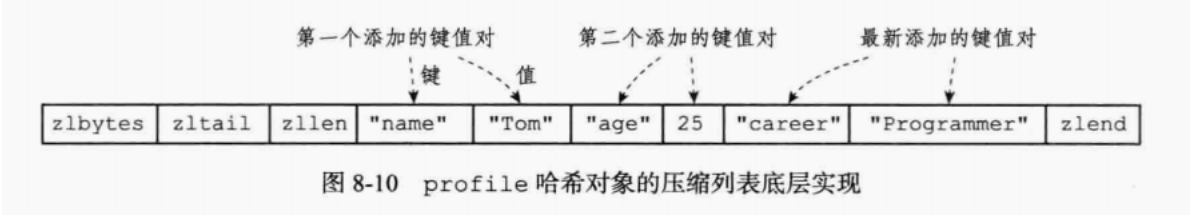

图 8-10　profile 哈希对象的压缩列表底层实现

hashtable

> hashtable编码的哈希对象使用字典作为底层实现，哈希对象的每个键值对都使用一个字典键值对来保存

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

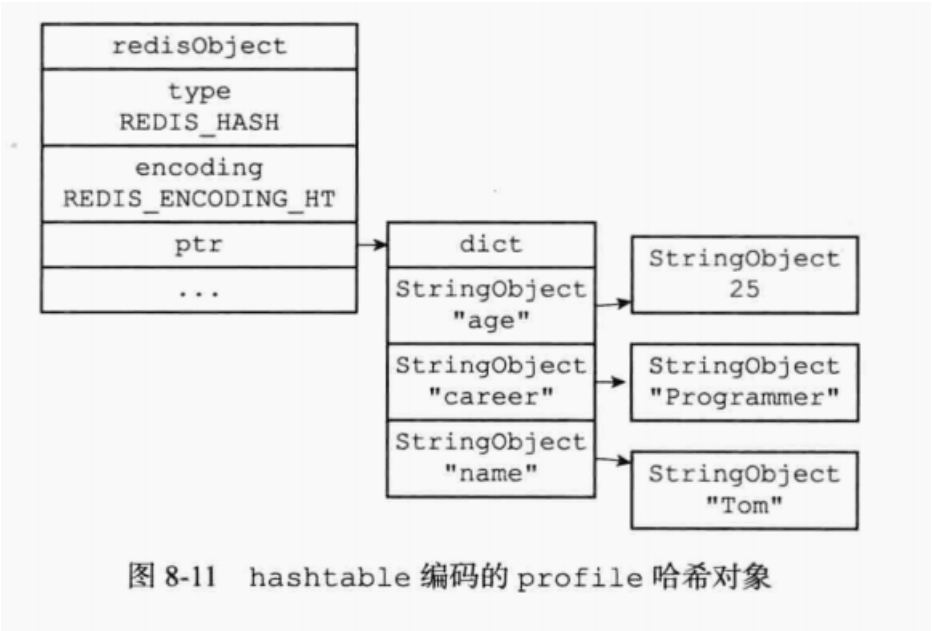

图 8-11　hashtable 编码的 profile 哈希对象

对象排序无序

1.3.2 命令

```
#将哈希表 hash 中域 field 的值设置为 value
HSET hash field value

#当且仅当域 field 尚未存在于哈希表的情况下， 将它的值设置为 value
HSETNX hash field value
```

```
#返回给定域的值
HGET hash field

#检查给定域 field 是否存在于哈希表 hash 当中。
HEXISTS hash field

#删除哈希表 key 中的一个或多个指定域，不存在的域将被忽略。
HDEL hash field1 [field2...]

#返回哈希表 key 中域的数量。
HLEN key

#返回哈希表 key 中， 与给定域 field 相关联的值的字符串长度（string length）
HSTRLEN key field

#为哈希表 key 中的域 field 的值加上增量 increment
HINCRBY key field increment

#为哈希表 key 中的域 field 加上浮点数增量 increment
HINCRBYFLOAT key field increment

#同时将一个或多个 field-value (域-值)对设置到哈希表 key 中。
HMSET key field value [field value..]

#返回哈希表 key 中，一个或多个给定域的值。
HMGET key field [field..]

#返回哈希表 key 中的所有域（慎用！）
HKEYS key

#返回哈希表 key 中所有域的值
HVALS key

#返回哈希表 key 中，所有的域和值
HGETALL key

#HSCAN key cursor [MATCH pattern] [COUNT count]
```

关于SCAN

表 8-9　哈希命令的实现

命令	ziplist 编码实现方法	hashtable 编码的实现方法
HSET	首先调用 ziplistPush 函数，将键推入到压缩列表的表尾，然后再次调用 ziplistPush 函数，将值推入到压缩列表的表尾	调用 dictAdd 函数，将新节点添加到字典里面
HGET	首先调用 ziplistFind 函数，在压缩列表中查找指定键所对应的节点，然后调用 ziplistNext 函数，将指针移动到键节点旁边的值节点，最后返回值节点	调用 dictFind 函数，在字典中查找给定键，然后调用 dictGetVal 函数，返回该键所对应的值
HEXISTS	调用 ziplistFind 函数，在压缩列表中查找指定键所对应的节点，如果找到的话说明键值对存在，没找到的话就说明键值对不存在	调用 dictFind 函数，在字典中查找给定键，如果找到的话说明键值对存在，没找到的话就说明键值对不存在
HDEL	调用 ziplistFind 函数，在压缩列表中查找指定键所对应的节点，然后将相应的键节点、以及键节点旁边的值节点都删除掉	调用 dictDelete 函数，将指定键所对应的键值对从字典中删除掉
HLEN	调用 ziplistLen 函数，取得压缩列表包含节点的总数量，将这个数量除以 2，得出的结果就是压缩列表保存的键值对的数量	调用 dictSize 函数，返回字典包含的键值对数量，这个数量就是哈希对象包含的键值对数量
HGETALL	遍历整个压缩列表，用 ziplistGet 函数返回所有键和值（都是节点）	遍历整个字典，用 dictGetKey 函数返回字典的键，用 dictGetVal 函数返回字典的值

1.4 集合

集合编码可以是intset和hashtable

1.4.1 数据结构

intset

> 集合对象保存的所有元素都是整数时
>
> 集合对象保存的元素数量不超过512个时

第二个条件是可以改变的，具体请看配置文件中关于set-max-intset-entries选项

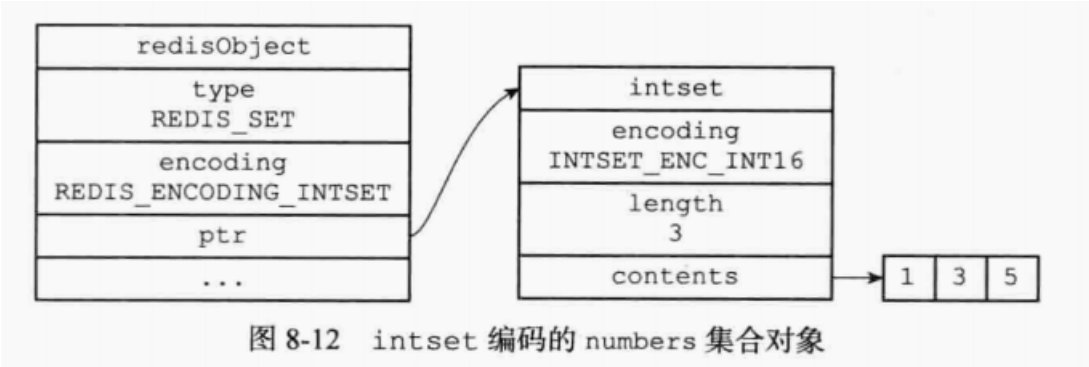

图 8-12　intset 编码的 numbers 集合对象

hashtable

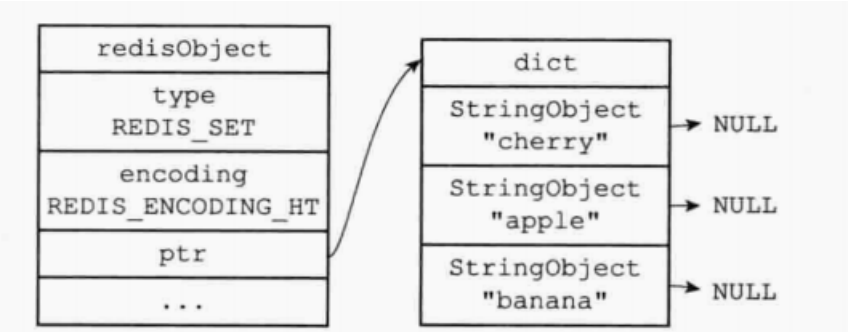

图 8-13　hashtable 编码的 fruits 集合对象

1.4.2 命令

```
# 将一个或多个 member 元素加入到集合 key 当中，已经存在于集合的 member 元素将被忽略。
SADD key member [member …]

#判断 member 元素是否集合 key 的成员。
SISMEMBER key member

# 返回集合 key 中的所有成员。
SMEMBERS key

#移除并返回集合中的一个随机元素。
SPOP key

# 如果命令执行时，只提供了 key 参数，那么返回集合中的一个随机元素
# 当count参数大于0时，返回count个随机不重复的元素的数组，若count大于集合的元素个数，则返回整个结合
# 当count参数小于0，返回|count|随机重复个元素的数组。若count大于集合的个数，也返回count个。
SRANDMEMBER key [count]

# 移除集合 key 中的一个或多个 member 元素，不存在的 member 元素会被忽略。
SREM key member [member …]

# 将 member 元素从 source 集合移动到 destination 集合。
SMOVE source destination member

# 返回集合 key 的基数(集合中元素的数量)。
SCARD key

# 返回一个集合的全部成员，该集合是所有给定集合的交集。
SINTER key [key …]

# 这个命令类似于 SINTER key [key …] 命令，但它将结果保存到 destination 集合，而不是简单地返回结果集。
SINTERSTORE destination key [key …]

# 返回一个集合的全部成员，该集合是所有给定集合的并集。
SUNION key [key …]

# 将结果保存到 destination 集合，而不是简单地返回结果集。
SUNIONSTORE destination key [key …]

# 返回一个集合的全部成员，该集合是所有给定集合之间的差集。
SDIFF key [key …]
```

```
# 将结果保存到 destination 集合，而不是简单地返回结果集。
SDIFFSTORE destination key [key …]
```

表 8-10　集合命令的实现方法

命令	intset 编码的实现方法	hashtable 编码的实现方法
SADD	调用 intsetAdd 函数，将所有新元素添加到整数集合里面	调用 dictAdd，以新元素为键，NULL 为值，将键值对添加到字典里面

（续）

命令	intset 编码的实现方法	hashtable 编码的实现方法
SCARD	调用 intsetLen 函数，返回整数集合所包含的元素数量，这个数量就是集合对象所包含的元素数量	调用 dictSize 函数，返回字典所包含的键值对数量，这个数量就是集合对象所包含的元素数量
SISMEMBER	调用 intsetFind 函数，在整数集合中查找给定的元素，如果找到了说明元素存在于集合，没找到则说明元素不存在于集合	调用 dictFind 函数，在字典的键中查找给定的元素，如果找到了说明元素存在于集合，没找到则说明元素不存在于集合
SMEMBERS	遍历整个整数集合，使用 intsetGet 函数返回集合元素	遍历整个字典，使用 dictGetKey 函数返回字典的键作为集合元素
SRANDMEMBER	调用 intsetRandom 函数，从整数集合中随机返回一个元素	调用 dictGetRandomKey 函数，从字典中随机返回一个字典键
SPOP	调用 intsetRandom 函数，从整数集合中随机取出一个元素，在将这个随机元素返回给客户端之后，调用 intsetRemove 函数，将随机元素从整数集合中删除掉	调用 dictGetRandomKey 函数，从字典中随机取出一个字典键，在将这个随机字典键的值返回给客户端之后，调用 dictDelete 函数，从字典中删除随机字典键所对应的键值对
SREM	调用 intsetRemove 函数，从整数集合中删除所有给定的元素	调用 dictDelete 函数，从字典中删除所有键为给定元素的键值对

1.5 有序集合

有序集合可以是ziplist和skiplist

1.5.1 数据结构

ziplist

> 压缩列表内的集合元素按分值从小到大进行排序，分值较小的被放置在靠近表头的方向，分值较小的被放置在靠近表尾的方向。
>
> 当有序集合保存的元素数量小于128个
>
> 有序集合保存的所有元素的长度都小于64字节

以上两个数值是可以修改的，具体在配置文件的zset-max-ziplist-entries选项和zset-max-ziplist-value选修

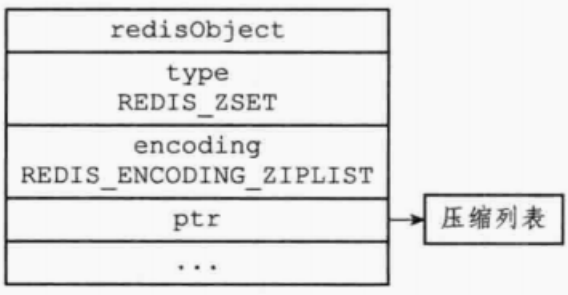

图 8-14　ziplist 编码的有序集合对象

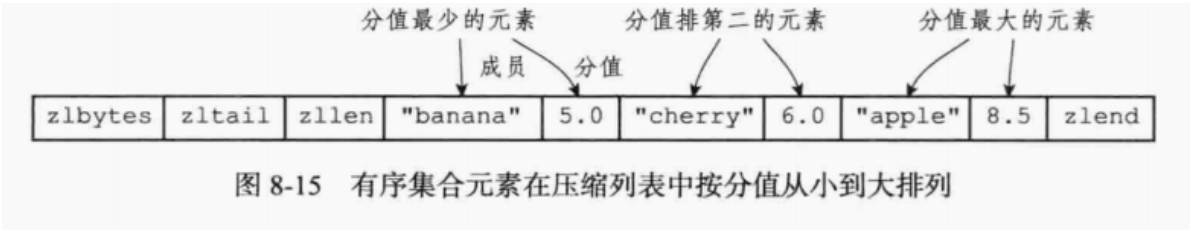

图 8-15　有序集合元素在压缩列表中按分值从小到大排列

skiplist

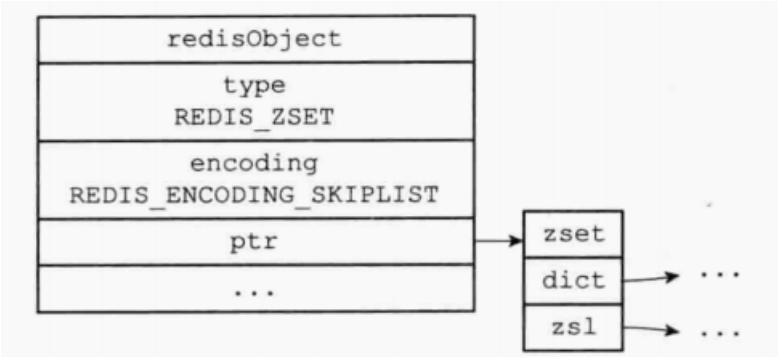

图 8-16　skiplist 编码的有序集合对象

dict字典为有序集合创建了一个从成员到分值的映射。

zsl跳跃表按分值从小到大保存了所有元素。

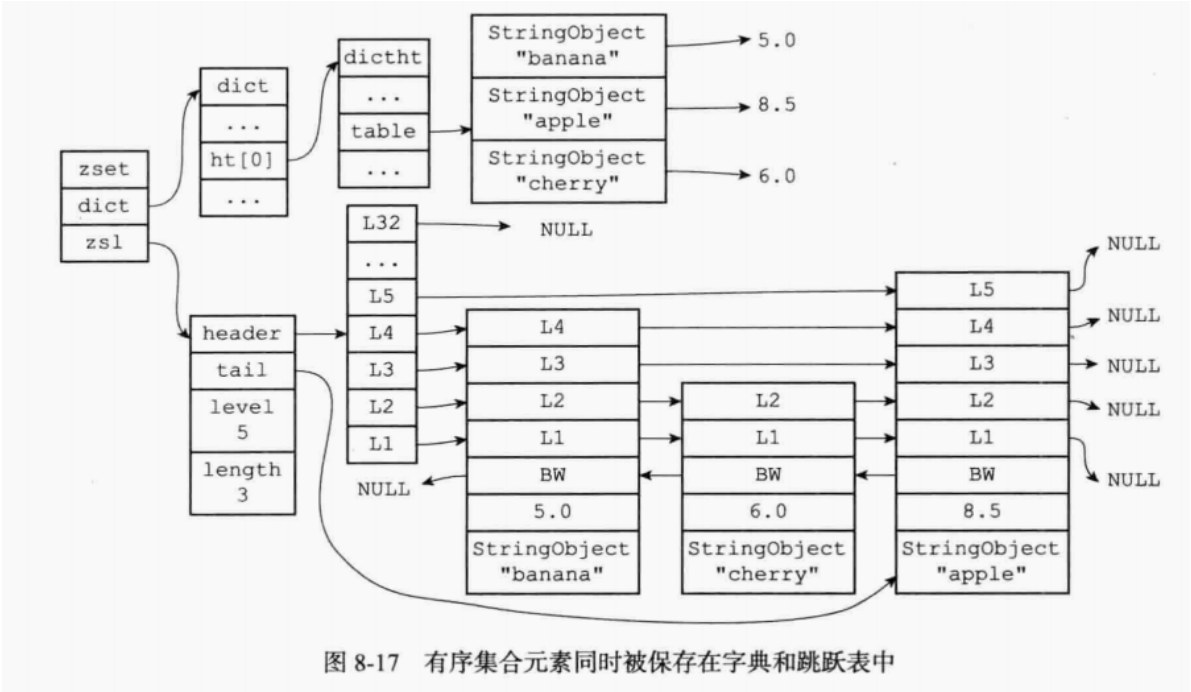

图 8-17　有序集合元素同时被保存在字典和跳跃表中

实际上，字典和跳跃表会共享元素的成员和分值，所以不会造成数据重复。

为什么有序集合需要同时使用跳跃表和字典来实现?

在理论上，有序集合可以单独使用字典或者跳跃表的其中一种数据结构来实现，但无论单独使用字典还是跳跃表，在性能上对比起同时使用字典和跳跃表都会有所降低。举个例子，如果我们只使用字典来实现有序集合，那么虽然以 $O(1)$ 复杂度查找成员的分值这一特性会被保留，但是，因为字典以无序的方式来保存集合元素，所以每次在执行范围型操作——比如 *ZRANK*、*ZRANGE* 等命令时，程序都需要对字典保存的所有元素进行排序，完成这种排序需要至少 $O(N\log N)$ 时间复杂度，以及额外的 $O(N)$ 内存空间（因为要创建一个数组来保存排序后的元素）。

另一方面，如果我们只使用跳跃表来实现有序集合，那么跳跃表执行范围型操作的所有优点都会被保留，但因为没有了字典，所以根据成员查找分值这一操作的复杂度将从 $O(1)$ 上升为 $O(\log N)$。因为以上原因，为了让有序集合的查找和范围型操作都尽可能快地执行，Redis 选择了同时使用字典和跳跃表两种数据结构来实现有序集合。

1.5.2 命令

```
# 将一个或多个 member 元素及其 score 值加入到有序集 key 当中。
ZADD key score member [[score member] [score member] …]

# 返回有序集 key 中，成员 member 的 score 值。
ZSCORE key member

# 为有序集 key 的成员 member 的 score 值加上增量 increment 。
ZINCRBY key increment member

# 返回有序集 key 的基数。
ZCARD key

# 返回有序集 key 中， score 值在 min 和 max 之间(默认包括 score 值等于 min 或 max )的成员的数量。
ZCOUNT key min max

# 返回有序集 key 中，指定区间内的成员。 其中成员的位置按 score 值递增(从小到大)来排序。
ZRANGE key start stop [WITHSCORES]

# 从大到小排序
ZREVRANGE key start stop [WITHSCORES]

# 返回有序集 key 中，所有 score 值介于 min 和 max 之间(包括等于 min 或 max )的成员。有序集成员按 score 值递增(从小到大)次序排列。
# -inf +inf 正负无穷
ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]

# 返回有序集 key 中， score 值介于 max 和 min 之间(默认包括等于 max 或 min )的所有的成员。有序集成员按 score 值递减(从大到小)的次序排列。
ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]

# 返回有序集 key 中成员 member 的排名。其中有序集成员按 score 值递增(从小到大)顺序排列。
ZRANK key member

# 返回有序集 key 中成员 member 的排名。其中有序集成员按 score 值递减(从大到小)排序。
ZREVRANK key member

# 移除有序集 key 中的一个或多个成员，不存在的成员将被忽略。
ZREM key member [member …]
```

```
# 移除有序集 key 中，指定排名(rank)区间内的所有成员。
ZREMRANGEBYRANK key start stop

# 移除有序集 key 中，所有 score 值介于 min 和 max 之间(包括等于 min 或 max )的成员。
ZREMRANGEBYSCORE key min max

# 对于一个所有成员的分值都相同的有序集合键 key 来说， 这个命令会返回该集合中， 成员介于 min 和 max 范围内的元素数量。
ZLEXCOUNT key min max
```

表 8-11　有序集合命令的实现方法

命令	ziplist 编码的实现方法	zset 编码的实现方法
ZADD	调用 ziplistInsert 函数，将成员和分值作为两个节点分别插入到压缩列表	先调用 zslInsert 函数，将新元素添加到跳跃表，然后调用 dictAdd 函数，将新元素关联到字典
ZCARD	调用 ziplistLen 函数，获得压缩列表包含节点的数量，将这个数量除以 2 得出集合元素的数量	访问跳跃表数据结构的 length 属性，直接返回集合元素的数量
ZCOUNT	遍历压缩列表，统计分值在给定范围内的节点的数量	遍历跳跃表，统计分值在给定范围内的节点的数量
ZRANGE	从表头向表尾遍历压缩列表，返回给定索引范围内的所有元素	从表头向表尾遍历跳跃表，返回给定索引范围内的所有元素
ZREVRANGE	从表尾向表头遍历压缩列表，返回给定索引范围内的所有元素	从表尾向表头遍历跳跃表，返回给定索引范围内的所有元素
ZRANK	从表头向表尾遍历压缩列表，查找给定的成员，沿途记录经过节点的数量，当找到给定成员之后，途经节点的数量就是该成员所对应元素的排名	从表头向表尾遍历跳跃表，查找给定的成员，沿途记录经过节点的数量，当找到给定成员之后，途经节点的数量就是该成员所对应元素的排名
ZREVRANK	从表尾向表头遍历压缩列表，查找给定的成员，沿途记录经过节点的数量，当找到给定成员之后，途经节点的数量就是该成员所对应元素的排名	从表尾向表头遍历跳跃表，查找给定的成员，沿途记录经过节点的数量，当找到给定成员之后，途经节点的数量就是该成员所对应元素的排名
ZREM	遍历压缩列表，删除所有包含给定成员的节点，以及被删除成员节点旁边的分值节点	遍历跳跃表，删除所有包含了给定成员的跳跃表节点。并在字典中解除被删除元素的成员和分值的关联
ZSCORE	遍历压缩列表，查找包含了给定成员的节点，然后取出成员节点旁边的分值节点保存的元素分值	直接从字典中取出给定成员的分值