目录

[一、 对象 1](#_Toc40127844)

[(一) 简述 1](#_Toc40127845)

[1) 为什么要建立对象系统 1](#_Toc40127846)

[(二) 对象类型和编码 1](#_Toc40127847)

[1) 类型 1](#_Toc40127848)

[2) 编码和底层实现 2](#_Toc40127849)

[(三) 字符串对象 4](#_Toc40127850)

[1) Raw和embstr有什么区别 4](#_Toc40127851)

[2) 编码的转换 5](#_Toc40127852)

[3) 字符串命令的实现 5](#_Toc40127853)

[(四) 列表对象 6](#_Toc40127854)

[1) 编码转换 6](#_Toc40127855)

[2) 列表命令的实现 6](#_Toc40127856)

[(五) 哈希对象 7](#_Toc40127857)

[1) 编码转换 7](#_Toc40127858)

[2) 哈希命令实现 8](#_Toc40127859)

[(六) 集合对象 8](#_Toc40127860)

[1) 编码转换 9](#_Toc40127861)

[2) 集合命令实现 9](#_Toc40127862)

[(七) 有序集合对象 10](#_Toc40127863)

[1) 为什么有序集合同时使用跳表和字典来实现 10](#_Toc40127864)

[2) 编码的转换 10](#_Toc40127865)

[3) 有序集合命令的实现 11](#_Toc40127866)

# 对象

## 简述

Redis并没有直接使用上面的数据结构来实现键值对数据库，而是用这些数据结构创建了一个对象系统，这个系统包含了字符串对象，列表对象，哈希对象，集合对象，有序集合对象这五种类型的对象，队中对象至少使用了一种上面提到的数据结构。

### 为什么要建立对象系统

* 1. 通过这五种对象系统，Redis在执行命令之前，可以根据对象的类型判断一个对象是否可以执行给定命令
  2. 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。
  3. 节约内存，对象系统实现了基于引用计数技术的内存回收机制。在适当条件下，通过让多个数据库共享同一对象来节约内存。

## 对象类型和编码

Redis中每一个对象都由一个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、encoding和ptr：

Typedef struct redisObject{

Unsigned type:4; //类型

Unsigned encoding:4; //编码

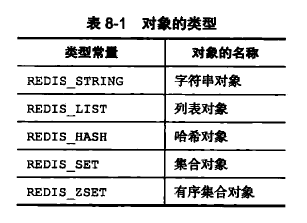
Void \*ptr ; //指向底层实现数据结构的指针

//…

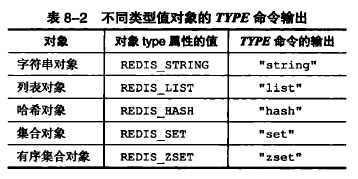
}

### 类型

Type记录了对象的类型，可以为以下其中一个:

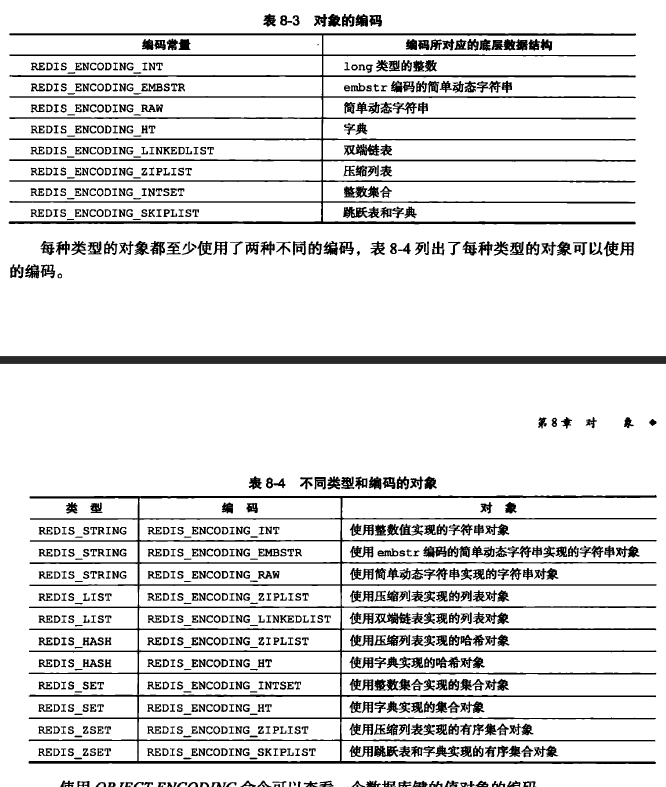


TYPE命令可以输出对象的类型



### 编码和底层实现

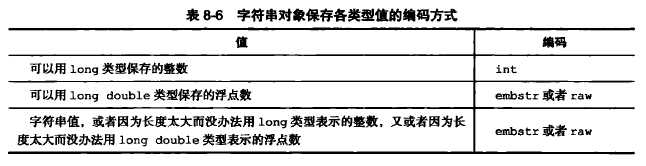
Ptr指针指向底层实现数据结构，而这些数据结构由encoding决定。Encoding是什么编码，ptr就指向对应类型的底层实现



使用OBJECT ENCODING命令可以查看键的值对象的编码



## 字符串对象

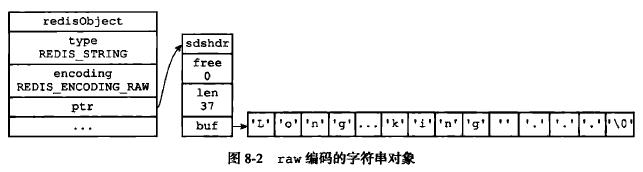


字符串对象是五种类型的对象中唯一一种会被其他四种类型对象嵌套的对象

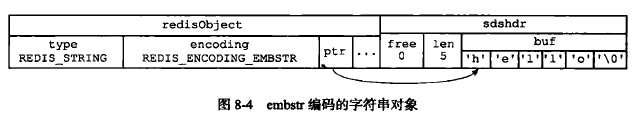
### Raw和embstr有什么区别

1. 内存结构不同

Raw:



Embstr: 相比raw的话embstr是连续的，比raw能更好利用缓存带来的优势



1. 内存的分配方式不同

Raw需要调用两次内存分配函数，分别创建redisObject结构和SDS结构；而embstr结构则只需要调用一次。

1. 内存的释放方式不同

有分配就有释放，embstr释放次数的同样逼raw少一次

1. Embstr对象是只读的，而且没有任何对应的修改程序，要修改会被转换成raw

### 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成raw编码的字符串对象。

### 字符串命令的实现



## 列表对象

列表对象的编码可以是ziplist（压缩列表）或者linkedlist（双端链表）

### 编码转换

当列表对象满足以下两个条件时，列表对象使用ziplist编码：

1. 所有字符串元素的长度都小于64字节
2. 保存元素的数量小于512个

当不能满足以上条件时则使用linkedlist编码

(注意：上述条件的上限值是可以修改的)

### 列表命令的实现

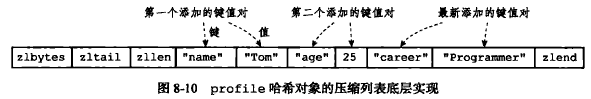


## 哈希对象

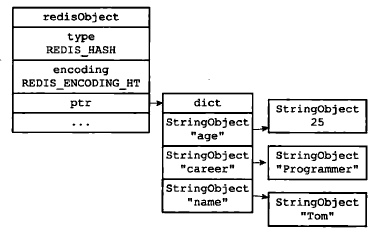
哈希对象的编码可以是ziplist或者hashtable

Ziplist编码：

新键值对加入时先将键加入表尾，再将值加入表尾



Hashtable编码：



### 编码转换

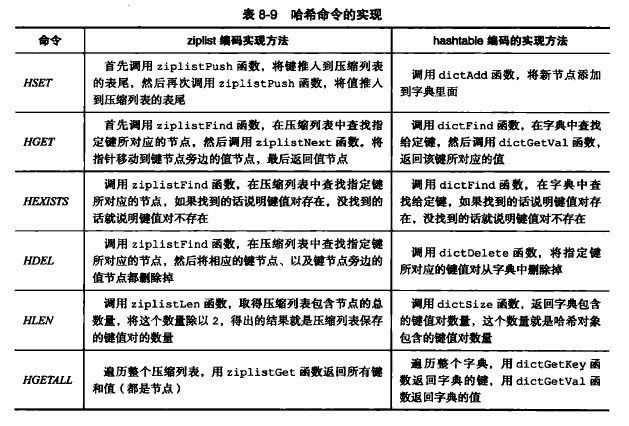
当哈希对象满足以下两个条件时，哈希对象使用ziplist编码：

1. 所有字符串元素的长度都小于64字节
2. 保存元素的数量小于512个

当不能满足以上条件时则使用hashtable’编码

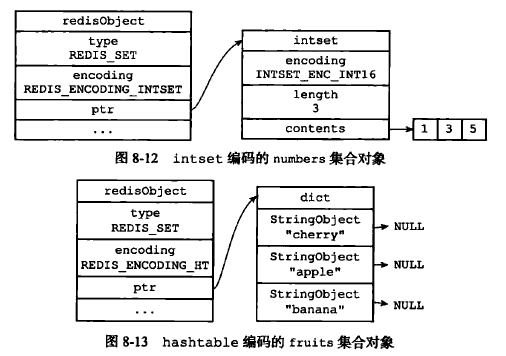
(注意：上述条件的上限值是可以修改的)

### 哈希命令实现



## 集合对象

集合对象的编码可以是intset或者是hashtable。值得注意的是hashtable编码的集合对象使用字典作为底层实现，每一个键都是字符串对象，每个字符串对象就是一个集合元素，而字典的值全为NULL。



### 编码转换

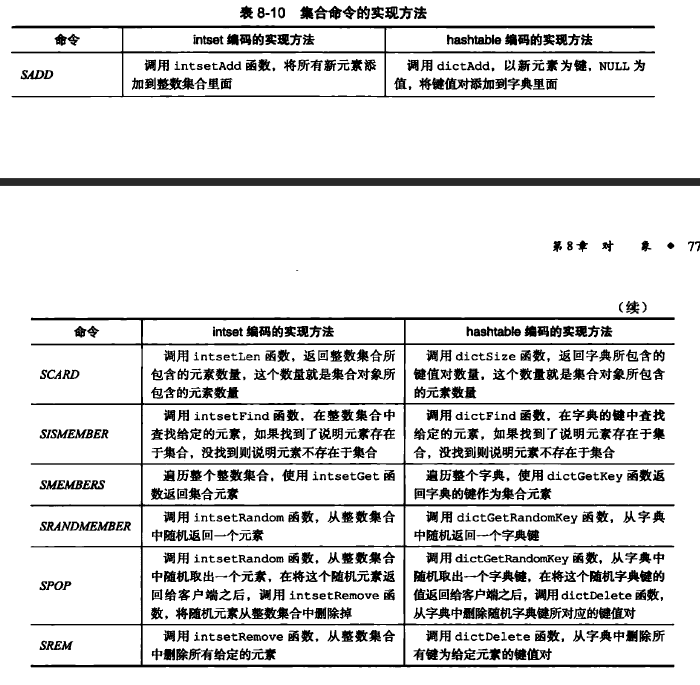
当集合对象满足以下两个条件时，集合对象使用intset编码：

1. 集合对象保存的所有元素都是整数值
2. 集合对象保存的元素数量不超过512个

当不能满足以上条件时则使用hashtable编码

(注意：第二个条件的上限值是可以修改的)

### 集合命令实现



## 有序集合对象

有序集合的编码可以是ziplist或者（skiplist 和 hashtable）。

Ziplist编码时，ziplist作为底层实现时，每个集合元素使用两个挨在一起的压缩列表节点保存，第一个是保存元素的成员，第二个保存元素的分值。

Skiplist编码时，使用zset结构作为底层实现

Typedef struct zset{

Zskiplist \*zsl; //跳表

Dict \*dict; //字典

}zset;

跳表中的每个节点都保存了一个集合元素：obj属性保存了成员，score属性保存了分值

字典中的键时成员，值就是分值。并且，两种数据结构都通过指针来共享相同元素的成员和分值

### 为什么有序集合同时使用跳表和字典来实现

原因就是单独使用的时候有序集合的性能降低。

|  |  |  |
| --- | --- | --- |
| 两种同时使用 | 只使用跳表 | 只使用字典 |
| 查询分值 | O(log(n)) | O(1) |
| 查询第k大 | O(log(n)) | O(nlog(n)) |

### 编码的转换

当有序集合对象满足以下两个条件时，有序集合对象使用ziplist编码：

1. 有序集合对象保存的所有元素的长度小于64字节
2. 有序集合对象保存的元素数量小于128个

当不能满足以上条件时则使用skiplist编码

(注意：以上条件的上限值是可以修改的)

### 有序集合命令的实现



## 类型检查和命令多态

Redis中用于操作键的命令基本可以分为两种类型：适用于任何类型的、对特定类型的键执行的

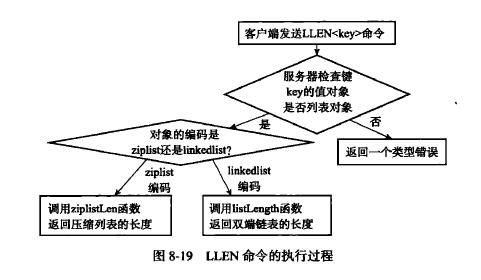
### 类型检查的实现

检查redisObject中的type属性是否为执行命令所需的类型即可。若不是则返回类型错误。

### 多态命令的实现

Redis除了会根据对象的类型判断能否执行外，还对根据对象的编码，选择正确的命令实现代码来执行命令。

以LLEN为例：



实际上第一二种类型的命令都是多态命令，

第一种是基于类型的多态——一个命令处理多种不同类型的键，

第二种是基于编码的多态——一个命令处理多种不同编码，

## 内存回收（基于引用计数的内存回收机制）

C语言并不具备自动的内存回收功能，因此redis在自己的对象系统中构建了使用引用计数的实现的内存回收机制。

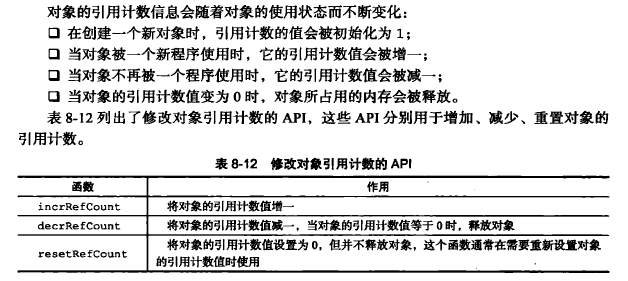
Typedef struct redisObject{

//…

//引用计数

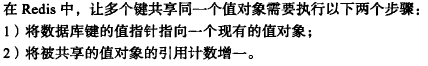
Int refcount;

}robj;



## 对象共享

对象的引用计数属性还有对象共享的作用。



### 为什么Redis 不共享包含字符串的对象？

当服务器考虑将一个共享对象设置为键的值对象时，程序需要先检查给定的共享对象和键创建的目标对象是否完全相同，只有在共享对象和目标对象完全相同的情况下，程序才会将共享对象用作键的值对象，而一个共享对象保存的值越复杂，验证共享对象和目标对象是否相同所需的复杂度就会越高，消耗的 CPU 时间也会越多：

1. 如果共享对象是保存整数值(0~9999)的字符串对象，那么验证操作的复杂度为O(1)
2. 如果共享对象是保存字符串值的字符串对象，那么验证操作的复杂度为 O(N)
3. 如果共享对象是包含了多个值(或者对象) 对象，比如列表对象或哈希对象，那么验证操作的复杂度为 O(N^2)

      因此，尽管共享更复杂的对象可以节约更多的内存，但受到 CPU 时间的限制，Redis 只对包含整数值的字符串对象进行共享

## 对象的空转时长

除了上面提到的四个属性以外，redisObject还有最后一个属性lru属性，记录了最后一次被程序访问的时间。

Typedef struct redisObject{

//…

Unsigned lru:32;

//…

}robj;

OBJECT IDLETIME命令可以打印出给定键的空转时长，这一空转时长就是通过当前的时间减去键的值对象的lru时间计算得出的。



除了可以被OBJECT IDLETIME命令打印出来之外，键的空转时长还有另外一个作用：

如果服务器打开了maxmemory选项，并且服务器用于回收内存的算法为volatile-lru或者allkeys-lru的话，那么当服务器占用的内存超过上限值时，空转时长较高的键会被服务器释放，从而回收内存。