1. 字典
2. 概述

一种用于保存键值对的抽象数据结构，一个键可以和一个值进行关联，每个键都是唯一的。由于C语言中没有内置字典，所以Redis构建了自己的字典实现，在Redis中的用途也很广泛，如Redis的数据库就是使用字典作为底层实现的，还有哈希键的底层实现之一等等。

1. 字典的实现

Redis的字典使用哈希表作为底层实现，一个哈希表里可以有多个哈希表节点，每个哈希表节点保存了字典中的一个键值对。

1. 哈希表

Redis字典所使用的哈希表使用dictht结构定义：

typedef struct dictht{

dictEntry \*\*table;

unsigned long size;

unsigned long sizemask;

unsigned long used;

}dictht;

table属性是一个数组，每个元素都是一个指向dictEntry结构的指针，该结构为哈希表节点，保存着一个键值对。

size属性记录了哈希表的大小，即table数组的大小；

used记录了哈希表目前已有节点的数量；

sizemask属性的值等于size-1，用于和哈希值一起决定一个键应该被放到table数组中的哪个索引之上，后面有解释

1. 哈希表节点

哈希表节点使用dictEntry结构表示，该结构保存着一个键值对。

typedef struct dictEntry{

void \*key;

union{

void \*val;

uint64\_t u64;

int64\_t s64;

}v;

struct dictEntry \*next;

}dictEntry;

key属性为键值对中的键；

v属性为键值对中的值，可以是指针或是整数，即union中的任一个；

next属性是指向另一个哈希表节点的指针，可以将多个哈希值相同的键值对链接在一起，姐姐键冲突问题。

1. 字典

字典由dict结构实现

typedef struct dict{

dictType \*type;

void \*privdata;

dictht ht[2];

int trehashidx;

}dic;

type属性和privdata属性是针对不同类型的键值对，为创建多态字典而设置的；

ht属性是一个包含两个项的数组，每个项都是一个dictht哈希表，一般情况下字典使用ht[0]哈希表，ht[1]只有在对ht[0]进行rehash时使用；

trehashidx记录了rehash目前的进度，如果没有进行rehash，则为-1

1. 哈希算法

当要将一个新的键值对添加到字典中时，首先根据键来计算出哈希值和索引值，再根据索引值，将包含新键值对的哈希表节点放到哈希表数组的指定索引上面。Redis计算哈希值的方法是dict->type->hashFunction(key)，计算出哈希值后，根据哈希表的sizemask属性和哈希值，计算出索引值：

index = hash & dict->ht[x].sizemask;

ht[x]可以是ht[0]或者ht[1]，如果是没有rehash时，都在ht[0]中，如果正在进行rehash，则新节点放到ht[1]中，旧节点依然在ht[0]中。

计算出索引值后，索引值就是该哈希表节点应该放到哈希表数组的哪个位置，如果是0就放到ht[0][0]的位置，Redis中使用MurmurHash2算法计算哈希值。

1. 键冲突问题

当有两个或以上的键被分配到哈希表数组的同一个索引之上时，称这些键发生了冲突，Redis的哈希表采用链地址法解决键冲突问题，多个哈希表节点使用next指针构成一个单向链表。因为dictEntry节点组成的链表没有指向链表表尾的指针，为了性能，采用头插法添加新的哈希表节点。

1. rehash

当哈希表保存的键值对过多或过少时，程序需要对哈希表的大小进行相应的扩展和收缩，通过rehash来完成。

1. 为字典的ht[1]哈希表分配空间，取决于要执行的操作以及ht[0]当前包含的键值对数量。如果是扩展操作，则ht[1]为第一个大于等于2倍键值对数量的2的n次幂；如果是收缩操作，那么ht[1]的大小就是第一个大于等于键值对数量的2的n次幂。
2. 将ht[0]中所有键值对rehash到ht[1]中，指的是重新计算键的哈希值和索引，将键值对放到ht[1]中对应位置。
3. 完成所有迁移后，释放ht[0]，设置ht[1]为ht[0]，并在ht[1]处新建一个空白哈希表。

哈希表的负载因子 = 哈希表已保存节点数 / 哈希表大小

当哈希表的负载因子小于0.1时，自动执行收缩操作。

整个的rehash操作并不是集中完成的，而是分多次，渐进式地完成的，因为对于大规模的哈希表，计算量吃不消。将所有的rehash需要的计算分摊到对字典的增删改查操作中。增操作只在ht[1]中，其他的操作再两个哈希表中同时进行。渐进式的rehash步骤如下：

1. 为ht[1]分配空间，让字典同时持有ht[0]和ht[1]；
2. 在字典中维持一个索引计数器变量trehashidx，设置为0，表示开始rehash；
3. 每次对字典执行增删改查时，除了执行指定操作外，还会顺带将ht[0]在trehashidx索引上的所有键值对rehash到ht[1]中；
4. ht[0]全部迁移完成，trehashidx设置为-1；