Redis之字典

wenmingxing 关注
2018.03.25 21:39:41 字数 2,332 阅读 6,862

> 字典本身就是很常见的数据结构之一，在Redis中，Redis数据库就是使用字典来作为底层实现的，除了用来表示数据库之外，字典还是哈希键的底层实现之一。

建议阅读：

1、字典部分源码研究见： wenmingxing Redis源码研究之dict

I、字典的实现

Redis的字典使用**哈希表**作为底层实现。

1.1 哈希表

Redis字典所使用的哈希表结构定义如下：

```
typedef struct dictht {

    // 哈希表数组
    dictEntry **table;

    // 哈希表大小
    unsigned long size;

    // 哈希表大小掩码，用于计算索引值
    // 总是等于 size - 1
    unsigned long sizemask;

    // 该哈希表已有节点的数量
    unsigned long used;

} dictht;
```

table属性是一个**数组**，数组中的每个元素都指向一个dictEntry结构的指针，每个dictEntry结构保存着一个键值对。

size属性记录了哈希表的大小，即table数组的大小，而used属性则记录了哈希表目前已有键值对的数量。

sizemask属性的值总是等于size-1，这个属性和哈希值一起决定一个键应该被放到table数组的哪个索引上。

下图展示了一个大小为4的空哈希表：

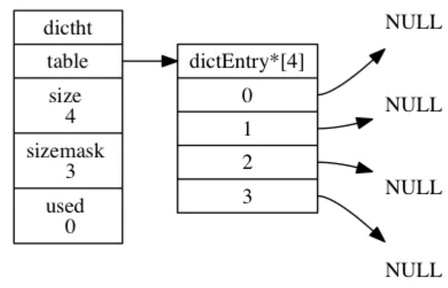


图 4-1　一个空的哈希表

1.2 哈希表节点

哈希表节点使用dictEntry结构表示，每个dictEntry结构都保存一个键值对：

```
typedef struct dictEntry {

    // 键
    void *key;

    // 值
    union {
        void *val;
        uint64_t u64;
        int64_t s64;
    } v;

    // 指向下个哈希表节点，形成链表
    struct dictEntry *next;

} dictEntry;
```

key属性保持着键值对中的键，而v属性则保存着键值对中的值，其中键值对中的值可以是一个指针，或者是一个整数。

next属性是指向另一个哈希表节点的指针，这个指针可以将多个哈希值相同的键值对连接在一起，来解决**键冲突问题**（以链表的方式解决冲突问题）。

如下图表示一个完成的哈希表：

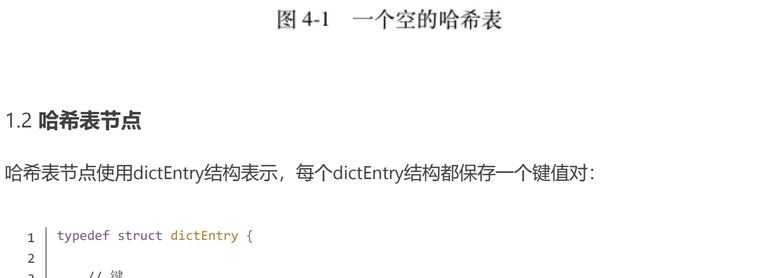


图 4-2　连接在一起的键 k1 和键 k0

1.3 字典

Redis中的字典结果如下：

```
typedef struct dict {

    // 类型特定函数
    dictType *type;

    // 私有数据
    void *privdata;

    // 哈希表
    dictht ht[2];

    // rehash 索引
    // 当 rehash 不在进行时，值为 -1
    int rehashidx; /* rehashing not in progress if rehashidx == -1 */

} dict;
```


推荐阅读

万字长文，38 图爆肝 Redis 基础
阅读 267

万字长文，38 图爆肝 Redis 基础!
阅读 92

三次给你讲清楚Redis之Redis是个啥
阅读 91

增删改查的基本操作：实现MyArrayList，你学会了吗?
阅读 68

重学Redis：Redis常用数据类型+存储结构（源码篇）
阅读 284

对的函数，Redis会为用途不同的字典设置不同类型的特定函数。
而privadata属性则保存了需要传给那些类型特定函数的可选参数。

ht**属性是一个包含了两个项的数组，数组中每个项都是一个**dictht**哈希表，一般情况下，字典只使用**ht[0]**哈希表，而**ht[1]**哈希表只对**ht[0]**哈希表进行** `rehash` **时使用。**

另一个与rehash有关的就是rehashidx属性，它积累了rehash目前的进度，如果没有进行rehash，则它的值为-1。

下图为一个普通状态下的字典结构：

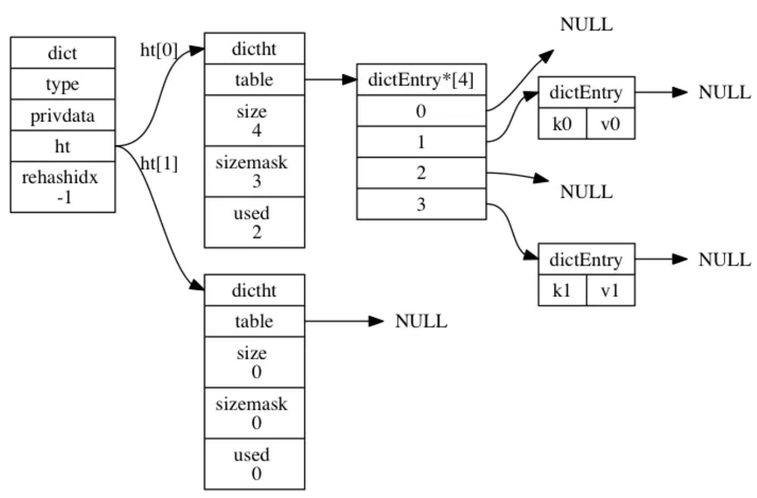


图 4-3　普通状态下的字典

II、哈希算法

将一个新的键值对添加到字典里面的时候，程序需要先根据键值对上面的键来计算出哈希值和索引值，然后再根据索引值，将包含新键值对的哈希表节点放到哈希数组的指定索引上面。

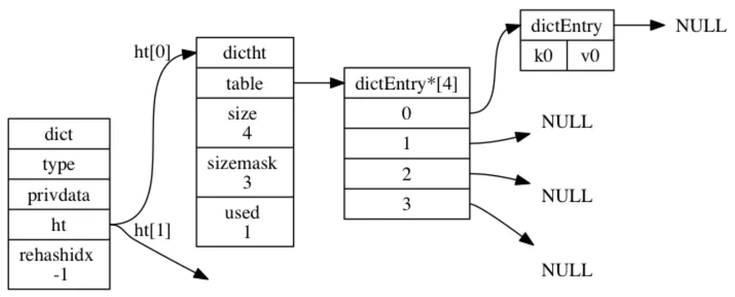
Redis计算哈希值和索引值的方法如下：

```
# 使用字典设置的哈希函数，计算键 key 的哈希值
hash = dict->type->hashFunction(key);

# 使用哈希表的 sizemask 属性和哈希值，计算出索引值
# 根据情况不同， ht[x] 可以是 ht[0] 或者 ht[1]
index = hash & dict->ht[x].sizemask;
```

下面举例说明一个完整的添加键值对<k0, v0>过程：

1. 首先程序会先使用语句 `hash = dict->type->hashFunction(k0);` 计算的处k0的哈希值。
2. 假设计算出的哈希值为8，则程序继续 `index = hash & dict->ht[0].sizemask = 8 & 3 = 0;` 计算得到k0的索引值为0，这表示包含这个键值对的节点应该放置到哈希表数组的索引0位置上。



...

图 4-5　添加键值对 k0 和 v0 之后的字典

tip： Redis使用MurmurHash2算法来计算键的哈希值。

III、解决键冲突

Redis哈希表使用**链地址法**来解决键冲突，每个哈希表节点都有一个next指针，多个哈希表节点可以用next构成一个单向链表，被分配到同一个索引上的节点可以用这个单向链表连接起来，从而解决键冲突问题。

下面的例子说明一个解决键冲突的实例：

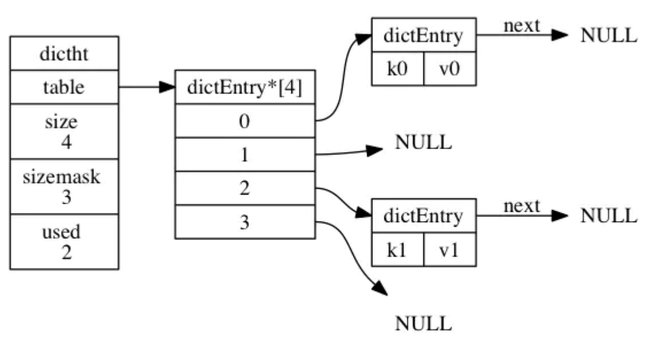


图 4-6　一个包含两个键值对的哈希表

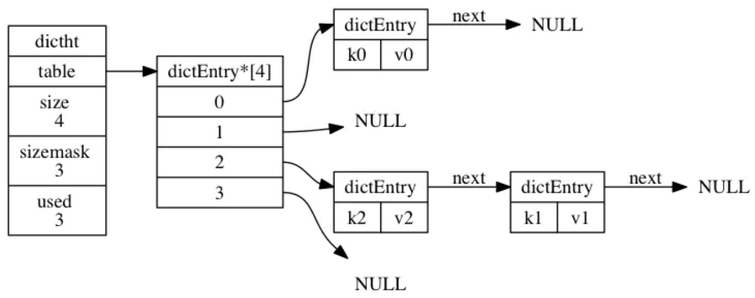


图 4-7　使用链表解决 k2 和 k1 的冲突

另外因为dictEntry节点组成的链表没有指向链表表尾的指针，为了考虑速度，**程序总是将新节点添加到链表的表头位置（这样添加节点的时间复杂度为**O(1)**）**。

IV、rehash

随着操作的不断进行，哈希表保存的键值对会逐渐增多或减少，为了让哈希表**负载因子**维持在一个合理范围之内，当哈希表保存的键值对太多或太少时，程序要对哈希表的大小进行相应的扩展或收缩。

如果执行的扩展操作，则ht[1]的大小为第一个大于等于ht[0].used*2的2^n；
如果执行的收缩操作，则ht[1]的大小为第一个大于等于ht[0].used的2^n；

2. 将保存在ht[0]中的所有键值对rehash到ht[1]上面：rehash指的是重新计算键的哈希值和索引值，然后将键值对放置到ht[1]的指定位置上。

3. 当ht[0]包含的所有键值对都迁移到ht[1]之后，释放ht[0]，将ht[1]设置为ht[0]，并在ht[1]新创建一个空白哈希表，为下一次rehash做准备。

下面为一个rehash的实例：

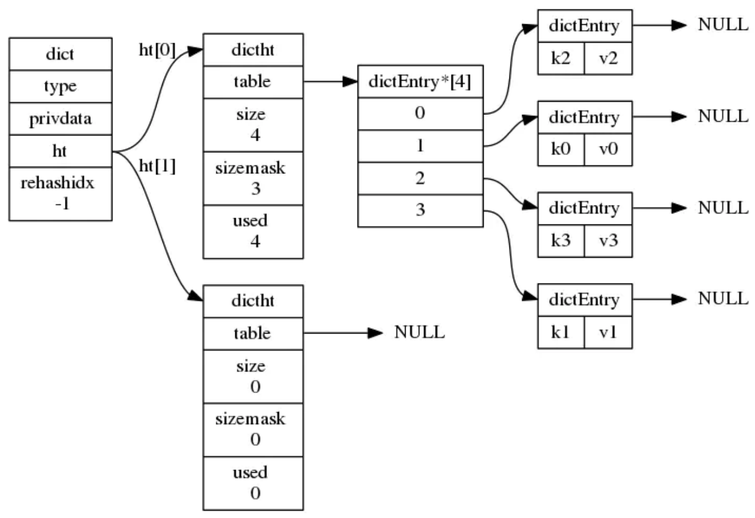


图 4-8　执行 rehash 之前的字典

2*4 = 8(2^3)：

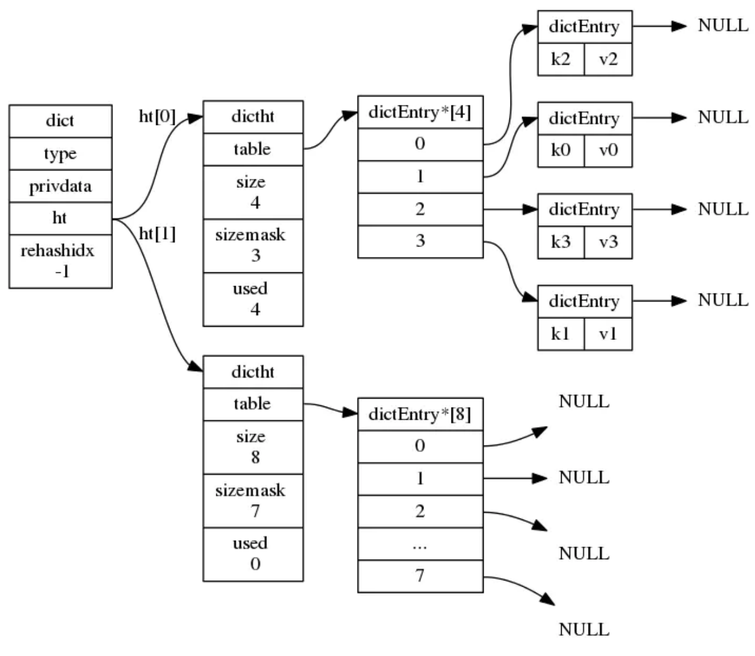


图 4-9　为字典的 ht[1] 哈希表分配空间

重新计算索引，并复制：

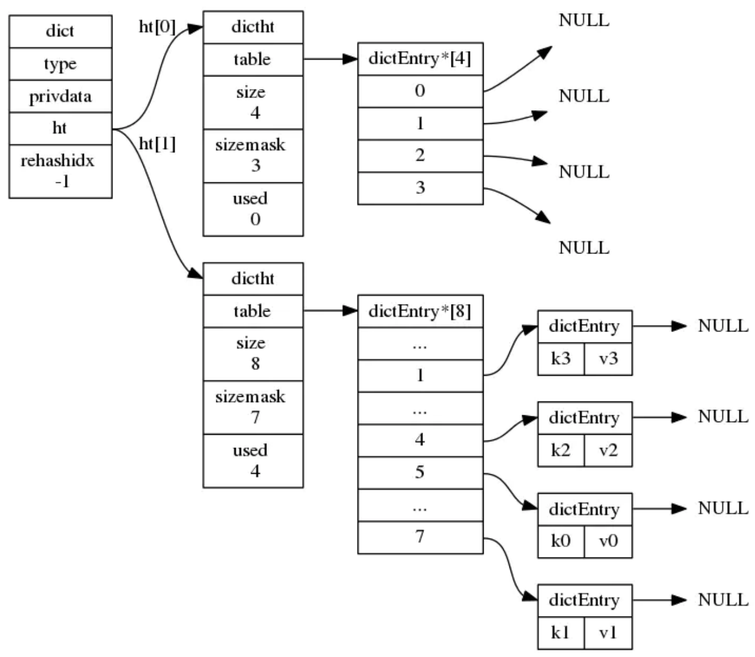


图 4-10　ht[0] 的所有键值对都已经被迁移到 ht[1]

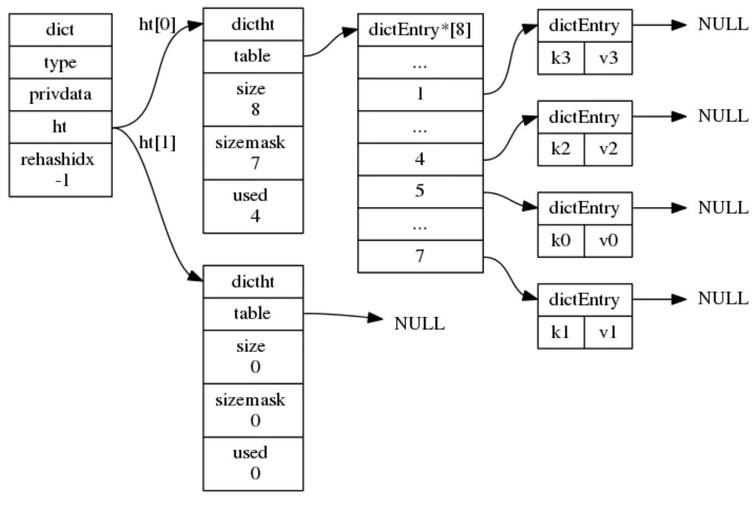


图 4-11　完成 rehash 之后的字典

哈希表的扩展与收缩

当以下条件中任意一个被满足时，程序会自动开始对哈希表执行扩展操作：

1. 服务器目前没有执行BGSAVE或BGREWRITEAOF命令，并且哈希表负载因子大于等于1。
2. 服务器正在执行BGSAVE或BGREWRITEAOF命令，并且哈希表负载因子大于等于5。

区分这两种情况的目的在于，因为执行BGSAVE与BGREWRITEAOF过程中，Redis都需要创建子进程，而大多数操作系统都采用**写时复制**技术来优化子进程使用效率，所以在子进程存在期间，服务器会提高执行扩展操作所需的负载因子，从而尽可能避免在子进程存在期间进行哈希

V、渐进式rehash

Redis中的rehash动作并不是一次性、集中式完成的，而是**分多次、渐进式的完成的。**

这样做的目的是，如果服务器中包含很多键值对，要一次性的将这些键值对全部rehash到ht[1]的话，庞大的计算量可能导致服务器在一段时间内停止服务于。

为了避免这种影响，Redis采用了**渐进式**Redis：

1. 为ht[1]分配空间，让字典同时持有ht[0]和ht[1]两个哈希表。
2. 在字典中维持一个索引计数器变量rehashidx，并将它置为0，表示rehash工作开始。
3. 在rehash进行期间，**每次对字典执行添加、删除、查找或者更新操作时，程序除了执行指定操作以外，还会顺带将**ht[0]**哈希表在**rehashidx**索引上的所有键值对**rehash**到**ht[1]**中**，当rehash工作完成之后，程序将rehashidx属性的值+1。
4. 随着字典操作的不断进行，最终在某个时间点上，ht[0]的所有键值对都被rehash到ht[1]上，这时将rehashidx属性设为-1，表示rehash完成。

渐进式rehash的好处在于其采取**分而治之**的方式，将rehash键值对所需要的计算工作均摊到字典的**每个**添加、删除、查找和更新操作上，从而避免了集中式rehash而带来的庞大计算量。

下面为一个渐进式rehash的实例：

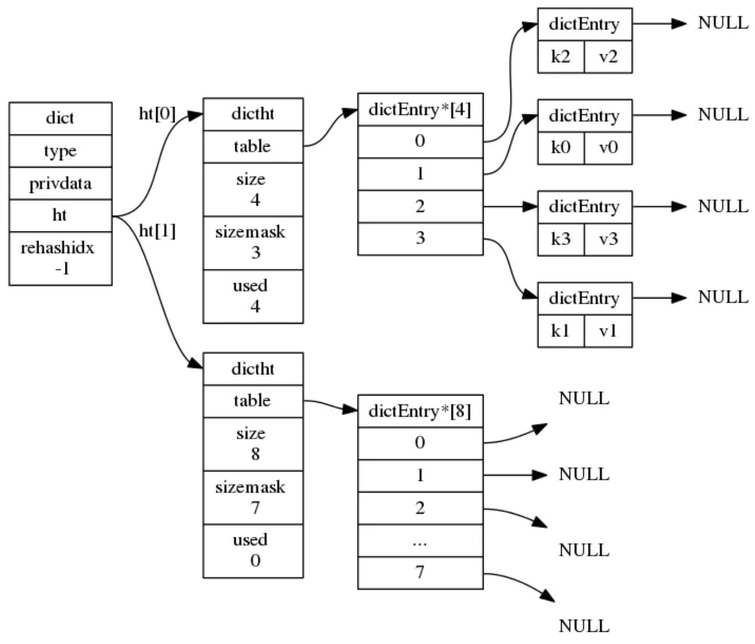


图 4-12　准备开始 rehash

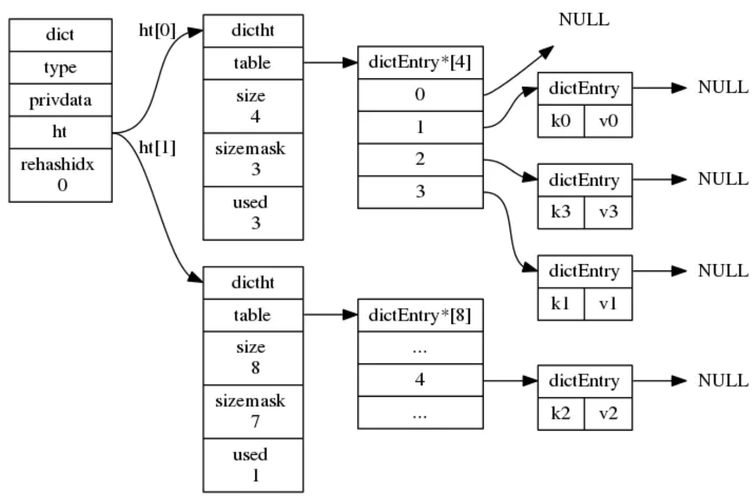


图 4-13　rehash 索引 0 上的键值对

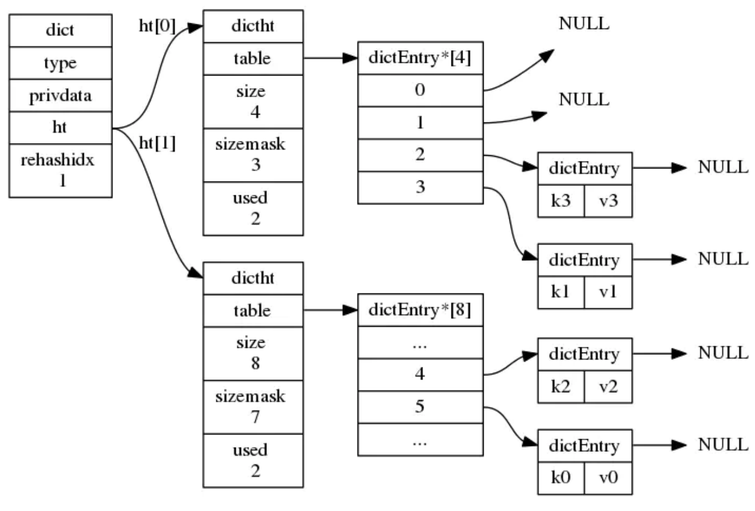


图 4-14　rehash 索引 1 上的键值对

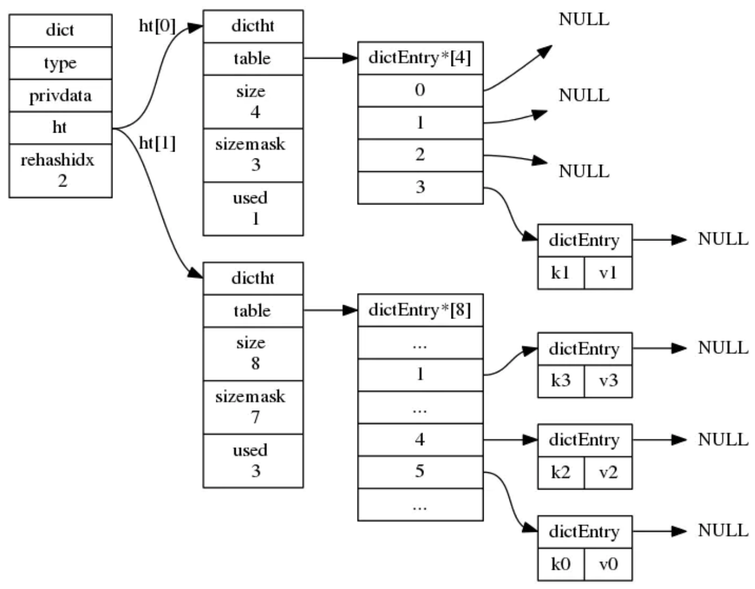


图 4-15　rehash 索引 2 上的键值对

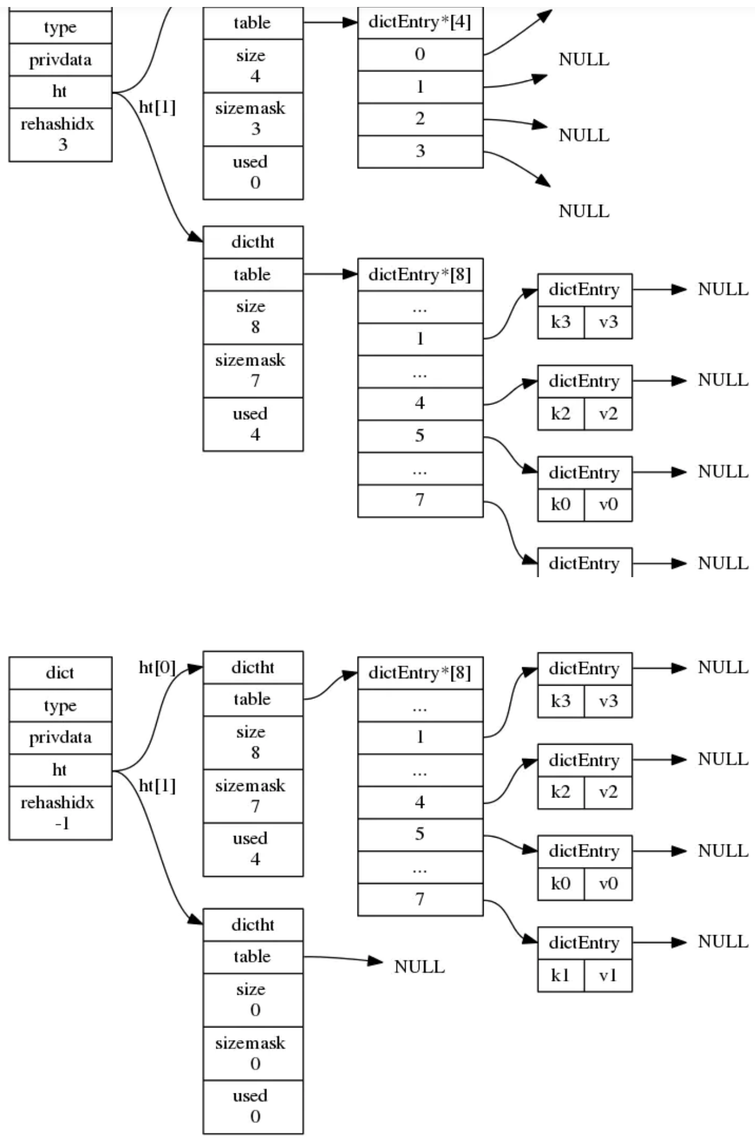


图 4-17 rehash 执行完毕

渐进式rehash**执行期间的哈希表操作**

因为在渐进式rehash的过程中，字典会同时使用ht[0]和ht[1]两个哈希表，所以在渐进式rehash进行期间，字典的删除、查找、更新等操作都是在两个表上进行的。

例如，**查找**操作会先在ht[0]上进行，如果没找到再在ht[1]上进行。

添加操作的键值对会一律保存到ht[1]中，这一措施保证ht[0]包含的键值对只会减少不会增加。

VI、字典API

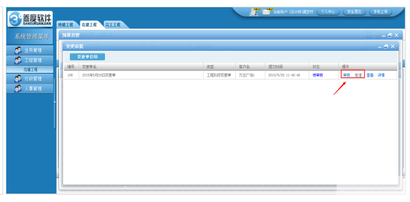
表 4-1 字典的主要操作 API

函 数	作 用	时间复杂度
dictCreate	创建一个新的字典	$O(1)$
dictAdd	将给定的键值对添加到字典里面	$O(1)$
dictReplace	将给定的键值对添加到字典里面，如果键已经存在于字典，那么用新值取代原有的值	$O(1)$
dictFetchValue	返回给定键的值	$O(1)$
dictGetRandomKey	从字典中随机返回一个键值对	$O(1)$
dictDelete	从字典中删除给定键所对应的键值对	$O(1)$
dictRelease	释放给定字典，以及字典中包含的所有键值对	$O(N)$，N 为字典包含的键值对数量

【参考】

[1] 《Redis的设计与实现》

什么叫erp管理系统



Panda2018
2019.07.26 18:16

@wenmingxing 感谢记录！

回复

添加新评论

7赞

赞赏

被以下专题收入，发现更多相似内容

我爱编程

更多好文