字典

1. 字典（dict）又称为散列表，是一种用来存储键值对的数据结构；
2. C语言没有内置这种数据结构，所以Redis构建了自己的字典实现。

字典在Redis中应用十分广泛：

1. Redis数据库的底层就是采用字典实现的；
2. 字典也是集合、哈希类型的底层实现之一；
3. Redis的哨兵模式，是以字典存储所有主从节点的。

Redis字典的实现主要涉及三个结构体：字典、哈希表、哈希表节点。其中，每个哈希表节点保存一个键值对，每个哈希表由多个哈希表节点构成，而字典则是对哈希表的进一步封装。

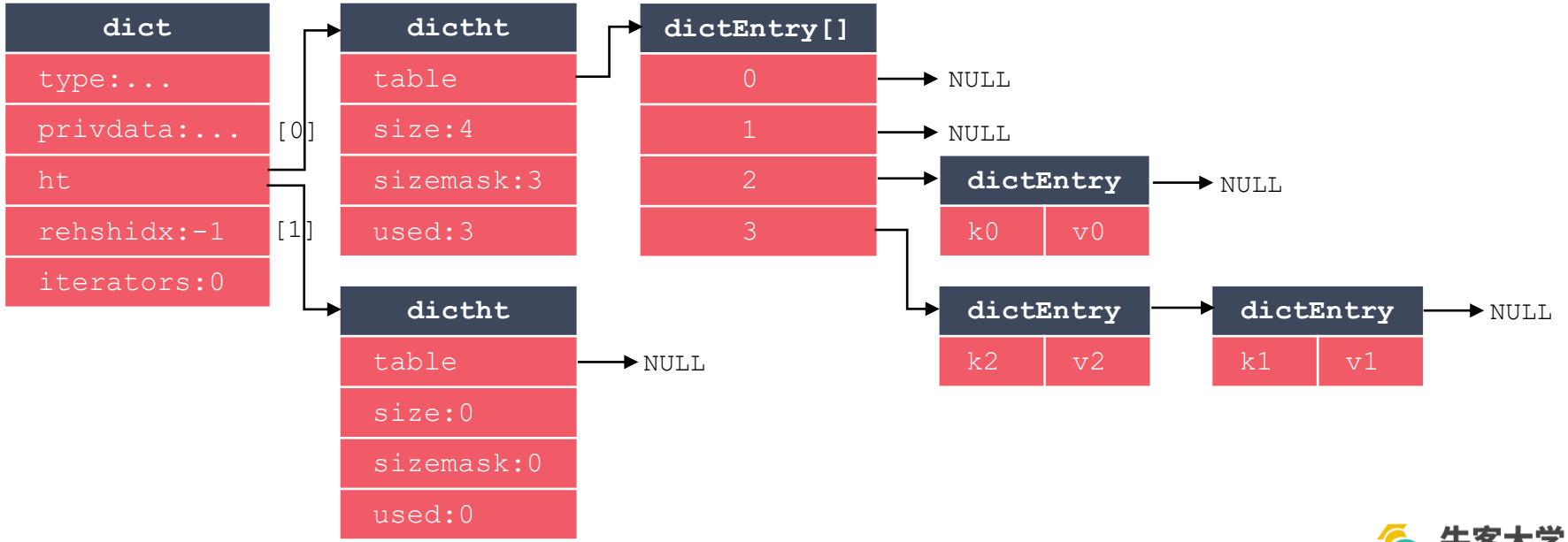
```
typedef struct dict {
    dictType *type;           // 字典类型，内含若干特定的操作函数；
    void *privdata;           // 该字典持有的私有数据；
    dictht ht[2];             // 哈希表数组，固定长度为2；
    long rehashidx;           // rehash标识，存储rehash的偏移量，默认-1；
    unsigned long iterators; // 记录绑定在此字典上的，正在运行的迭代器数量；
} dict;
```

```
typedef struct dictht {
    // 节点数组
    dictEntry **table;
    // 数组大小
    unsigned long size;
    // 掩码（size-1）
    unsigned long sizemask;
    // 已有节点数量
    unsigned long used;
} dictht;
```

```
typedef struct dictEntry {
    void *key;                // 键
    union {
        void *val;
        uint64_t u64;
        int64_t s64;
        double d;
    } v;                      // 值
    struct dictEntry *next;   // 下一节点
} dictEntry;
```

dict.h

Redis字典的实现主要涉及三个结构体：字典、哈希表、哈希表节点。其中，每个哈希表节点保存一个键值对，每个哈希表由多个哈希表节点构成，而字典则是对哈希表的进一步封装。



向字典中添加新的键值对时，程序需要先根据键计算出哈希值，再根据哈希值计算出索引值，最后将此键值对封装在哈希表节点中，放到节点数组的指定索引上，关键步骤参考如下代码：

```
// 使用哈希函数，计算键的哈希值
hash = dict->type->hashFunction(key);
// 使用哈希值和掩码，计算索引值
// 等价于哈希值与哈希表容量取余，但位运算效率更高
index = hash & dict->ht[x].sizemask;
```

键冲突问题：

1. 当多个键被分配到了节点数组的同一个索引上时，则这些键发生了冲突；
2. Redis采用链表来解决键冲突，即使用next指针将这些节点链接起来，形成单向链表；
3. Redis的哈希表节点没有设计表尾指针，每次添加时都是将新节点插入到表头的位置。

- REHASH的基本概念

当哈希表保存的键值对数量过多或过少时，需要对哈希表的大小进行扩展或收缩操作，在Redis中，扩展和收缩哈希表是通过REHASH实现的，执行REHASH的大致步骤如下：

1. 为字典的ht[1]哈希表分配内存空间

   如果执行的是扩展操作，则ht[1]的大小为第1个大于等于ht[0].used*2的2^n；

   如果执行的是收缩操作，则ht[1]的大小为第1个大于等于ht[0].used的2^n；

2. 将存储在ht[0]中的数据迁移到ht[1]上

   重新计算键的哈希值和索引值，然后将键值对放置到ht[1]哈希表的指定位置上；

3. 将字典的ht[1]哈希表晋升为默认哈希表

   迁移完成后，清空ht[0]，再交换ht[0]和ht[1]的值，为下一次REHASH做准备。

- REHASH的触发条件

  当满足以下任何一个条件时，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bgrewriteof命令，并且哈希表的负载因子大于等于1;
  2. 服务器目前正在执行bgsave或bgrewriteof命令，并且哈希表的负载因子大于等于5;

  其中，负载因子可以通过如下公式计算:

  ```
  load_factor = ht[0].used / ht[0].size;
  ```

  另外，当哈希表的负载因子小于0.1时，程序会自动开始对哈希表执行收缩操作。

- REHASH的详细步骤

  为了避免REHASH对服务器性能造成影响，REHASH操作不是一次性地完成的，而是分多次、渐进式地完成的。渐进式REHASH的详细过程如下：

  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. 在字典中的索引计数器rehashidx设置为0，表示REHASH操作正式开始；
  3. 在REHASH期间，每次对字典执行添加、删除、修改、查找操作时，程序除了执行指定的操作外，还会顺带将ht[0]中位于rehashidx上的所有键值对迁移到ht[1]中，再将rehashidx的值加1；
  4. 随着字典不断被访问，最终在某个时刻，ht[0]上的所有键值对都被迁移到ht[1]上，此时程序将rehashidx属性值设置为-1，标识REHASH操作完成。

- REHASH期间的访问

  REHSH期间，字典同时持有两个哈希表，此时的访问将按照如下原则处理：

  1. 新添加的键值对，一律被保存到ht[1]中；
  2. 删除、修改、查找等其他操作，会在两个哈希表上进行，即程序先尝试去ht[0]中访问要操作的数据，若不存在则到ht[1]中访问，再对访问到的数据做相应的处理。

牛客大学
-专业求职辅导-

THANKS

关注【牛客大学】公众号
回复“牛客大学”获取更多求职资料