牛客大学高薪加成系列课

跳跃表

牛客大学

- 专业求职辅导 -

1. 有序集合的底层，可以采用数组、链表、平衡树等结构来实现：

   数组不便于元素的插入和删除，链表的查询效率低，平衡树/红黑树效率高但实现复杂；

2. Redis采用跳跃表（skiplist）作为有序集合的一种实现方案：

   跳跃表的查找复杂度为平均O(logN)，最坏O(N)，效率堪比红黑树，却远比红黑树实现简单；

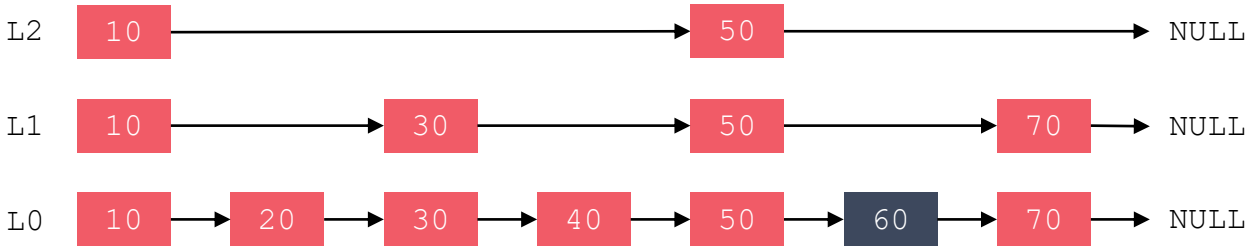
在Redis中，跳跃表的另一个应用是作为集群节点的内部数据结构！

有序链表插入、删除的复杂度为O(1)，而查找的复杂度为O(N)；

例：若要查找值为60的元素，需要从第1个元素依次向后比较，共需比较6次才行；

从有序链表中选取部分节点，组成一个新链表，并以此作为原始链表的一级索引；

从一级索引中选取部分节点，组成一个新链表，并以此作为原始链表的二级索引；



1．优先从高层开始查找；

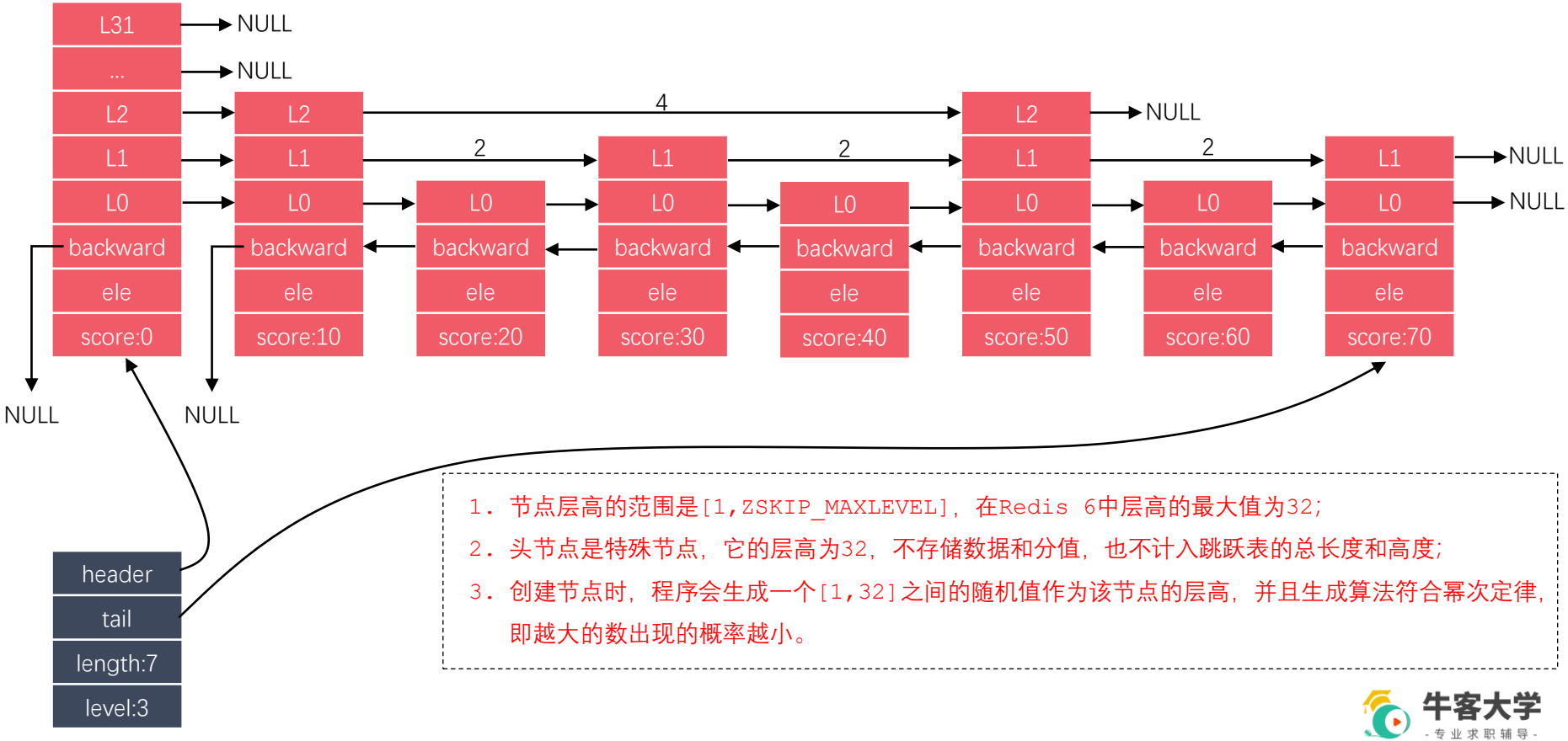
2．若next节点值大于目标值，或next指针指向NULL，则从当前节点下降一层继续向后查找；

1. 跳跃表的实现主要涉及2个结构体：zskiplist、zskiplistNode；
2. 3.0版本及以前，它们被定义在redis.h中，3.0版本之后，它们被定义在server.h中。

```
typedef struct zskiplistNode {
    // 节点数据
    sds ele;
    // 节点分值
    double score;
    // 后退指针
    struct zskiplistNode *backward;
    // 层级数组（各节点不一样）
    struct zskiplistLevel {
        // 前进指针
        struct zskiplistNode *forward;
        // 跨度（节点间的距离，用于计算排名）
        unsigned long span;
    } level[];
} zskiplistNode;
```

```
typedef struct zskiplist {
    // 表头指针、表尾指针
    struct zskiplistNode *header, *tail;
    // 跳跃表的长度（除表头之外的节点总数）
    unsigned long length;
    // 跳跃表的高度（除表头之外的最高层数）
    int level;
} zskiplist;
```

02 / 跳跃表的实现



1. 节点层高的范围是[1,ZSKIP_MAXLEVEL]，在Redis 6中层高的最大值为32；
2. 头节点是特殊节点，它的层高为32，不存储数据和分值，也不计入跳跃表的总长度和高度；
3. 创建节点时，程序会生成一个[1,32]之间的随机值作为该节点的层高，并且生成算法符合幂次定律，即越大的数出现的概率越小。

1. 跳跃表由多层构成，它的每一层都是一个有序链表，数据依次递增；
2. 跳跃表有一个头节点，它是一个32层的结构，内部不存储实际数据；
3. 跳跃表包含有头尾指针，分别指向跳跃表的第一个和最后一个节点；
4. 除头节点外，层数最多的节点的层高为跳跃表的高度；
5. 除头节点外，一个元素在上层有序链表中出现，则它一定能够会在下层有序链表中出现；
6. 跳跃表每层的最后一个节点指向NULL；
7. 最底层的有序链表包含所有的节点，最底层的节点个数为跳跃表的长度；
8. 每个节点包含一个后退指针，头节点和第一个节点指向NULL，其他节点指向最底层的前一节点。

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