超级好用之Bloom Filter与LRU Cache

今日目标：

1：完成实战面试题目

2：能说出布隆过滤器的工作原理

面试实战

146. LRU 缓存机制

```java
class LRUCache {

    int capacity;
    int size;
    Map<Integer,Node> hashtable;

    Node head; //链表头
    Node tail;//链表尾

    public LRUCache(int capacity) {
        hashtable = new HashMap();
        this.capacity = capacity;
        this.size = 0;
        head = new Node();//哨兵头
        tail = new Node();//哨兵尾
        //头尾先接上
        head.next = tail;
        tail.prev = head;
    }
    //获取元素
    public int get(int key) {
        //判断hashtable中是否存在key
        Node node = hashtable.get(key);
        if (node==null) {
            return -1;
        }
        //将该节点移动到链表头（哨兵后面）
        moveToHead(node);
        return node.value;
    }
    private void moveToHead(Node node) {
        //分两步走：第一步：将该节点删除,第二步：将该节点添加到头哨兵后面
        removeNode(node);
        addToHead(node);
    }
    //从双向链表中删除该节点
    private void removeNode(Node node) {
        //拿到前驱和后继
        Node prevNode = node.prev;
        Node nextNode = node.next;
        //前驱指向后继
        prevNode.next = nextNode;
```

```java
            //后继指向前驱
            nextNode.prev = prevNode;

            //清理node的prev和next指针
            node.prev = null;
            node.next = null;
        }
        //将该节点添加到链表头部（头哨兵后面）
        private void addToHead(Node node) {
            //先拿到原本头部元素(头哨兵后面的第一个元素)
            Node firstNode = head.next;
            //在哨兵和firstNode中插入node
            head.next = node;
            node.next = firstNode;
            firstNode.prev = node;
            node.prev = head;
        }
        //插入元素
        public void put(int key, int value) {
            //判断key是否在hashtable中
            Node node = hashtable.get(key);
            if (node!=null) {
                //更新value
                node.value = value;
                //将该节点移动到头部
                moveToHead(node);
                return;
            }

            //创建新节点
            node = new Node(key,value);
            //添加哈希表中
            hashtable.put(key,node);
            //将该新节点添加到头部
            addToHead(node);
            //元素个数加1
            this.size ++;
            //判断是否超过容量大小
            if (this.size > this.capacity) {
                //移除最少使用的元素(尾元素，尾哨兵的前一个)
                Node n = removeTail();
                //从哈希表中移除对应的key/value
                hashtable.remove(n.key);
                //元素个数减少
                this.size--;
            }

        }
        //干掉尾哨兵的前一个元素
        private Node removeTail() {
            Node node = tail.prev;
            Node prevNode = node.prev;
            //从双向链表中拿掉node
            prevNode.next = tail;
            tail.prev = prevNode;
            node.prev = null;
            node.next = null;
            return node;
```

```
    }

    class Node {
        int key;
        int value;

        Node prev;
        Node next;

        public Node() {}

        public Node(int key,int value) {
            this.key = key;
            this.value = value;
        }
    }

}

/**
 * Your LRUCache object will be instantiated and called as such:
 * LRUCache obj = new LRUCache(capacity);
 * int param_1 = obj.get(key);
 * obj.put(key,value);
 */
```

同类题目：

[面试题 16.25. LRU 缓存]

[460. LFU 缓存]