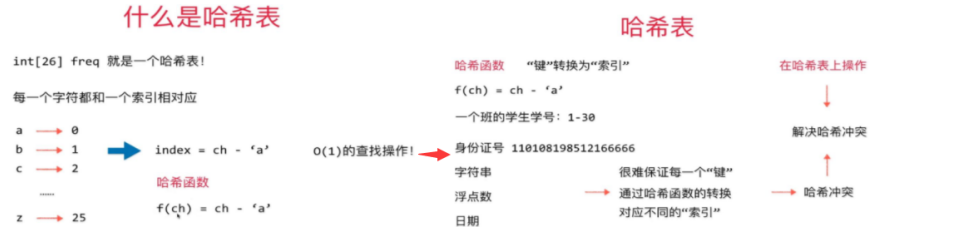
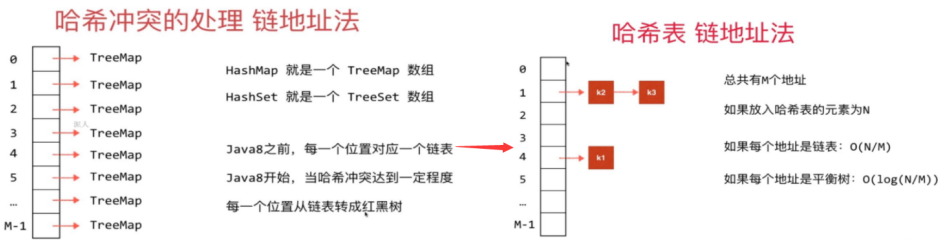
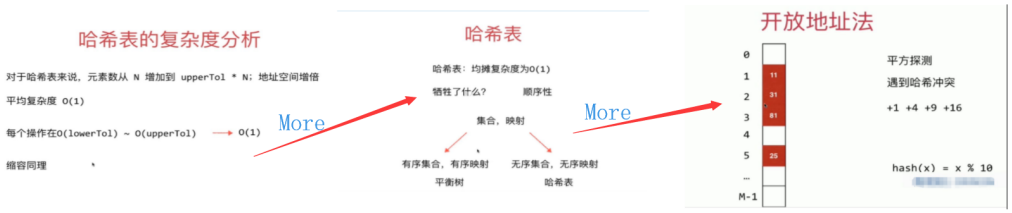
数据结构与算法 15- 哈希表

笔记本: 我的笔记
创建时间: 2020/10/11 11:08 **更新时间:** 2020/10/11 11:24
作者: liuhouer
标签: 算法

1>哈希表本质是数组，按照索引可以O(1)找到相应的元素，所以哈希操作就是一个把原来元素转化为一个索引的操作







2.实现Hash表【HashMap】

```
import java.util.Map;
import java.util.TreeMap;


public class HashTable<K, V> {


    private static final int upperTol = 10;
```

```
    private static final int lowerTol = 2;
    private static final int initCapacity = 7;


    private TreeMap<K, V>[] hashtable;
    private int size;
    private int M;


    public HashTable(int M){
        this.M = M;
        size = 0;
        hashtable = new TreeMap[M];
        for(int i = 0 ; i < M ; i ++)
            hashtable[i] = new TreeMap<>();
    }


    public HashTable(){
        this(initCapacity);
    }


    private int hash(K key){
        //key.hashCode() & 0x7fffffff 消除正数位的符号的写法【绝对值】
        return (key.hashCode() & 0x7fffffff) % M;
    }


    public int getSize(){
        return size;
    }


    public void add(K key, V value){
        TreeMap<K, V> map = hashtable[hash(key)];
        if(map.containsKey(key))
            map.put(key, value);
        else{
            map.put(key, value);
            size ++;


            if(size >= upperTol * M)
                resize(2 * M);
        }
    }


    public V remove(K key){
        V ret = null;
        TreeMap<K, V> map = hashtable[hash(key)];
        if(map.containsKey(key)){
            ret = map.remove(key);
            size --;


            if(size < lowerTol * M && M / 2 >= initCapacity)
                resize(M / 2);
        }
        return ret;
    }


    public void set(K key, V value){
        TreeMap<K, V> map = hashtable[hash(key)];
        if(!map.containsKey(key))
            throw new IllegalArgumentException(key + " doesn't exist!");


        map.put(key, value);
    }


    public boolean contains(K key){
```

```
        return hashtable[hash(key)].containsKey(key);
    }


    public V get(K key){
        return hashtable[hash(key)].get(key);
    }


    private void resize(int newM){
        TreeMap<K, V>[] newHashTable = new TreeMap[newM];
        for(int i = 0 ; i < newM ; i ++)
            newHashTable[i] = new TreeMap<>();


        int oldM = M;
        this.M = newM;
        for(int i = 0 ; i < oldM ; i ++){
            TreeMap<K, V> map = hashtable[i];
            for(K key: map.keySet())
                newHashTable[hash(key)].put(key, map.get(key));
        }


        this.hashtable = newHashTable;
    }
}
```