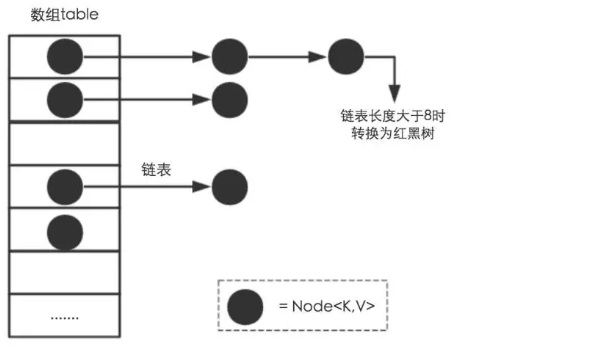
HashMap的数据结构



默认的初始容量-必须是2的幂。

**static final int *DEFAULT\_INITIAL\_CAPACITY*** = 1 << 4;

默认的装载因子

**static final float *DEFAULT\_LOAD\_FACTOR*** = 0.75f;

当链表长度大于8时转换为红黑树

**static final int *TREEIFY\_THRESHOLD*** = 8;

当红黑树节点小于6时再转为链表

**static final int *UNTREEIFY\_THRESHOLD*** = 6;

实际存储key-value的数组

**transient** Node<K,V>[] **table**;

**static class** Node<K,V> **implements** Map.Entry<K,V> {  
 **final int hash**; 用来定位数组索引的位置  
 **final** K **key**;  
 V **value**;  
 Node<K,V> **next**; 链表的下一个Node

}

Node是HashMap的一个内部类，实现了Map.Entry接口，本质就是一个映射(键值对)

实际存在的键值对数量

**transient int size**;

所能容纳的的最多键值对

**int threshold**;

Node<K,V>[] table的默认初始长度为16，Load factor负载因子的默认值为0.75

threshold是HashMap所能容纳的最大键值对数量。

threshold= table.length \* Load factor;

在数组定义好长度之后，负载因子越大，所能容纳的键值对更多。

向HashMap中存取元素，首先确定哈希桶数组索引位置

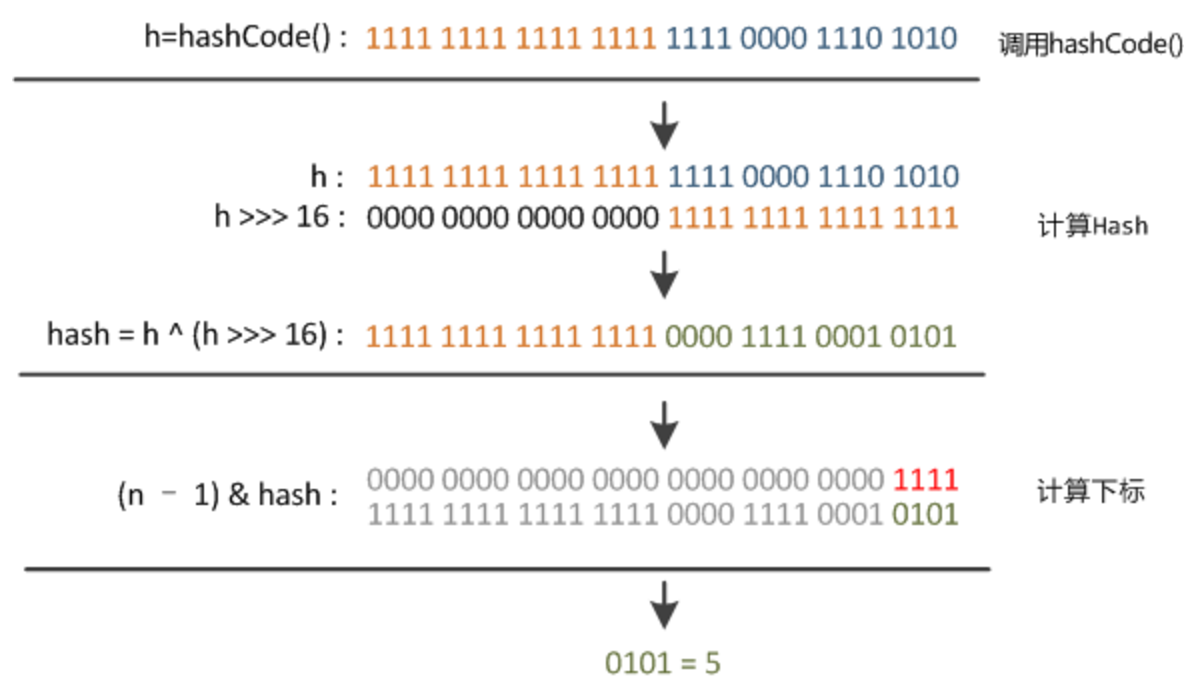
static final int hash(Object key) {

int h;

return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);

}

取key的hashCode、高位运算、取模运算



通过hashCode的高16位异或低16位，这样在数组的length比较小的时候，也能保证高低bit都参与到Hash的计算中，可以减少碰撞的几率

HashMap中put键值对过程

table为空或者length==0扩容

if ((tab = table) == null || (n = tab.length) == 0)

n = (tab = resize()).length;

取模运算得到要插入的索引，table[i] == null直接插入

if ((p = tab[i = (n - 1) & hash]) == null)

tab[i] = newNode(hash, key, value, null);

如果不为空，并且key存在，直接覆盖value

if (p.hash == hash && ((k = p.key) == key || (key != null && key.equals(k))))

e = p;

如果key不存在，table[i]是treeNode,向红黑树插入键值对

else if (p instanceof TreeNode)

e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);

table[i]不是treeNode，遍历链表插入数据，插入之后当链表的长度大于8时，将链表转为红黑树。如果链表上有存在的key，直接覆盖value。

else {

for (int binCount = 0; ; ++binCount) {

if ((e = p.next) == null) {

p.next = newNode(hash, key, value, null);

if (binCount >= TREEIFY\_THRESHOLD - 1) // -1 for 1st

treeifyBin(tab, hash);

break;

}

if (e.hash == hash &&

((k = e.key) == key || (key != null && key.equals(k))))

break;

p = e;

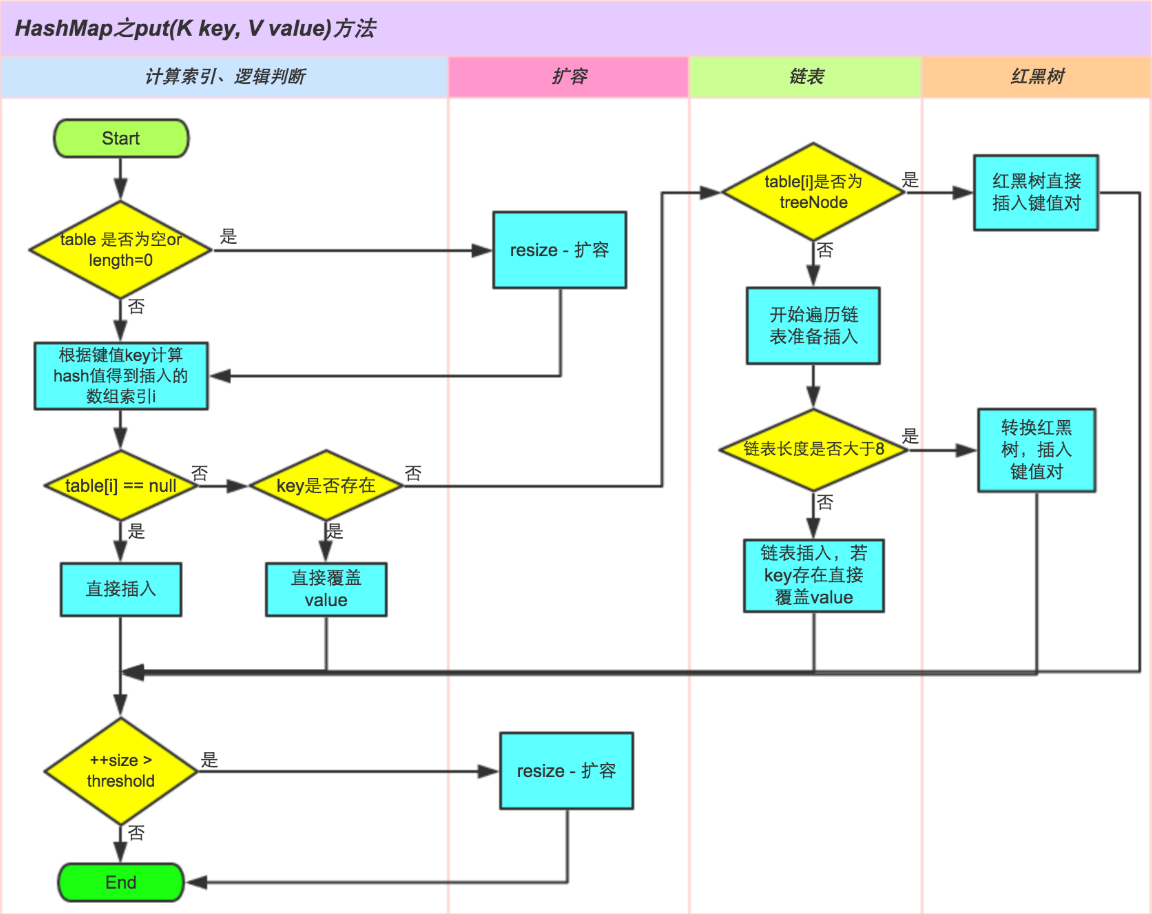
}

}

插入成功，HashMap的键值对数量+1，如果大于所能容忍的最多键值对数量进行扩容

if (++size > threshold)

resize();



Hashmap扩容机制

重新计算容量，向HashMap中不停的加入元素，如果超过所能容忍的最多键值对数量进行扩容。扩大数组的长度，以便于装入更多的元素。