链滴

HashMap 的正确初始化姿势以及并发问题(jdk1.7)

作者：614756773

原文链接：https://ld246.com/article/1592299555158

来源网站：链滴

初始化

- 参考
- 例子：

```
// 假设我们要存放1000个元素
Map map = new HashMap(1000);
for(int i = 0; i < 1000; i++) {
    map.put(i, null);
}
// 在初始化Map时，hashmap会计算出table大小应该为1024，但是1024*0.75 = 768
// 也就是说在put第768个时就会触发扩容，导致rehash
// 所以我们应该使用 Map map = new HashMap((int)(1000 / 0.75) + 1);
// 这样初始化时table的大小为2048，2048*0.75>1000，所以我们把1000个元素都put进去也不触发rehash
```

- 关键成员变量
  - table：hash 表　最开始时为null，当第一次put后才会初始化
  - threshold：阈值　除了初始化时为2的次幂，其余时候都为table.length*loadFactor
  - loadFactor：加载因子
- 初始化流程
  - 当调　new HashMap(1000) 时，只是会指定阈值且阈值是 2 的次幂，和扩容因子
  - 此时 hash 表还是个 null
  - 当第一次 put 后才会初始化 hash 表，并且把阈值变成 0.75

```
// 初始化HashMap对象时，并没有初始化table
```

```
public HashMap(int initialCapacity, float loadFactor) {
    ......
    this.loadFactor = loadFactor;
    this.threshold = tableSizeFor(initialCapacity);
}
static final int tableSizeFor(int cap) {
    int n = cap - 1;
    n |= n >>> 1;
    n |= n >>> 2;
    n |= n >>> 4;
    n |= n >>> 8;
    n |= n >>> 16;
    return (n < 0) ? 1 : (n >= MAXIMUM_CAPACITY) ? MAXIMUM_CAPACITY : n + 1;
}

// 第一次put数据时，table为null，调用resize()
public V put(K key, V value) {
    return putVal(hash(key), key, value, false, true);
}
final V putVal(int hash, K key, V value, boolean onlyIfAbsent,
               boolean evict) {
    Node<K,V>[] tab; Node<K,V> p; int n, i;
    if ((tab = table) == null || (n = tab.length) == 0)
        n = (tab = resize()).length;
    ......
}

// resize函数中，初始化了table，并且修改了threshold的值为 newCap * loadFactor
final Node<K,V>[] resize() {
    Node<K,V>[] oldTab = table;
    int oldCap = (oldTab == null) ? 0 : oldTab.length;
    int oldThr = threshold;
    int newCap, newThr = 0;
    ......
    else if (oldThr > 0) // initial capacity was placed in threshold
        newCap = oldThr;
    ......
    if (newThr == 0) {
        float ft = (float)newCap * loadFactor;
        newThr = (newCap < MAXIMUM_CAPACITY && ft < (float)MAXIMUM_CAPACITY ?
                  (int)ft : Integer.MAX_VALUE);
    }
    threshold = newThr;
    Node<K,V>[] newTab = (Node<K,V>[])new Node[newCap];
    ......
}
```

多线程 put 时导致成环

- 参考
- JDK1.8 之前的 resize

```
void resize(int newcapacity) {
```

```
    ...
    Entry[] newTable = new Entry[newCapacity];
    transfer(newTable);
    ...
}

void transfer(Entry[] newTable) {
    Entry[] src = table;
    int newCapacity = newTable.length;
    for (int j = 0; j < src.length; j++) {
        Entry e = src[j];
        if (e != null) {
            src[j] = null;
            do {
                Entry next = e.next;
                int i = indexFor(e.hash, newCapacity);
                e.next = newTable[i];
                newTable[i] = e;
                e = next;
            } while (e != null);
        }
    }
}
```

在 jdk1.7 及之前的版本中，解决 hash 冲突使用的方式是插入链表，并且是头插法，头插法的代码下有

hash 桶的头结点是真实的节点，不是哑结点

- 两个线程 put 造成 resize 死循环的场景：
    - 假设有两个线程都在插入数据而且都同时触发扩容机制，扩容时又恰好把新数据都放在一个桶设为hash桶2
    - 假设　hash桶2 的原链表为 A->B->C
    - 首先线程 2 只执行了很少一部分代码　一个 while 循环都还没来得及完成，刚把 e.next = newable[i]执行完，此时 e：A，next：B
    - 然后线程 1 执行完毕了 resize，此时链表关系变为 C->B->A
    - 线程 2 继续执行
        - 将上一次循环执行完毕后，链表关系为 newtable[i]：A，e：B
        - 接着执行第 2 次 while 循环，链表关系为 newtable[i]：B->A，e：A　如果线程 1 没执行 e 应该是 C，但是线程 1 已经把 B 的 next 指向了 A
        - 接着执行第 3 次 while 循环，链表关系为 newtable[i]：A<->B，e：null　**死循环出现**
        - 因为 e 为 null，循环结束
    - put 虽然发生了逻辑错误，但是好歹能够执行完毕，但是当 get 的时候，就会因为 A<->B 这个环链导致无限循环，卡死

头插法

```
public class Foo {
```

```
    public static void main(String[] args) {
        Node[] table = new Node[5];
        int index = 2;

        Node node = new Node("a");
        node.next = table[index];
        table[index] = node;

        Node b = new Node("b");
        b.next = table[index];
        table[index] = b;

        Node c = new Node("c");
        c.next = table[index];
        table[index] = c;
        System.out.println(table[index]);
        // 输出结果为：  c->b->a->
    }

    static class Node{
        private String value;
        private Node next;

        Node(String value) {
            this.value = value;
        }

        @Override
        public String toString() {
            return value + " ->" + (next != null ? next.toString() : "");
        }
    }
}
```