谈谈HashMap线程不安全的体现

HashMap的原理以及如何实现，之前在[JDK7与JDK8中HashMap的实现](http://my.oschina.net/hosee/blog/618953" \t "_blank)中已经说明了。

那么，为什么说HashMap是线程不安全的呢？它在多线程环境下，会发生什么情况呢？

3个情况，

1个put会同时扩容早造成死循环，

2.2个put引发扩容，另外的线程有可能get不到。

3.有可能2个同时put，导致1个丢失，被后1个put给覆盖掉了。

一种情况是直接2个线程，1存1取，A刚存完key1value1，还没等B取值，A又存完key1value2，这样B取值只能取得key1value2，key1value1就丢失了。

## 1. resize死循环

我们都知道HashMap初始容量大小为16,一般来说，当有数据要插入时，都会检查容量有没有超过设定的thredhold，如果超过，需要增大Hash表的尺寸，但是这样一来，整个Hash表里的元素都需要被重算一遍。这叫rehash，这个成本相当的大。

void transfer(Entry[] newTable, boolean rehash) {

        int newCapacity = newTable.length;

        for (Entry<K,V> e : table) {

            while(null != e) {

              Entry<K,V> next = e.next;

              if (rehash) {

               e.hash = null == e.key ? 0 : hash(e.key);

                }

              int i = indexFor(e.hash, newCapacity);

              e.next = newTable[i];

              newTable[i] = e;

              e = next;

            }

        }

}

大概看下transfer：

1. 对索引数组中的元素遍历
2. 对链表上的每一个节点遍历：用 next 取得要转移那个元素的下一个，将 e 转移到新 Hash 表的头部，使用头插法插入节点。
3. 循环2，直到链表节点全部转移
4. 循环1，直到所有索引数组全部转移

经过这几步，我们会发现转移的时候是逆序的。假如转移前链表顺序是1->2->3，那么转移后就会变成3->2->1。这时候就有点头绪了，死锁问题不就是因为1->2的同时2->1造成的吗？所以，HashMap 的死锁问题就出在这个transfer()函数上。

### 1.1 单线程 rehash 详细演示

单线程情况下，rehash 不会出现任何问题：

* 假设hash算法就是最简单的 key mod table.length（也就是数组的长度）。
* 最上面的是old hash 表，其中的Hash表的 size = 2, 所以 key = 3, 7, 5，在 mod 2以后碰撞发生在 table[1]
* 接下来的三个步骤是 Hash表 resize 到4，并将所有的 <key,value> 重新rehash到新 Hash 表的过程

如图所示：

### http://incdn1.b0.upaiyun.com/2016/10/28c8edde3d61a0411511d3b1866f0636.jpg1.2 多线程 rehash 详细演示

为了思路更清晰，我们只将关键代码展示出来

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | while(null != e) {      Entry<K,V> next = e.next;      e.next = newTable[i];      newTable[i] = e;      e = next;  } |

1. Entry<K,V> next = e.next;——因为是单链表，如果要转移头指针，一定要保存下一个结点，不然转移后链表就丢了
2. e.next = newTable[i];——e 要插入到新链表的头部，所以要先用 e.next 指向新的 Hash 表第一个元素（为什么不加到新链表最后？因为复杂度是 O（N））
3. newTable[i] = e;——现在新 Hash 表的头指针仍然指向 e 没转移前的第一个元素，所以需要将新 Hash 表的头指针指向 e（即把第一个元素换为e，原来的元素现在变成现在e指向的第二个了）
4. e = next——转移 e 的下一个结点。

**注意：**

假设这里有两个线程同时执行了put()操作，并进入了transfer()环节

while(null != e) {

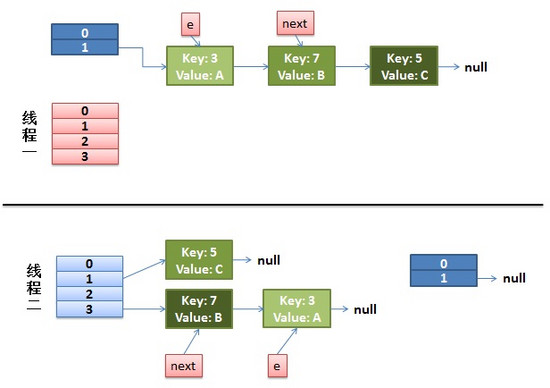
    Entry<K,V> next = e.next; //线程1执行到这里被调度挂起了

    e.next = newTable[i];

    newTable[i] = e;

    e = next;

}

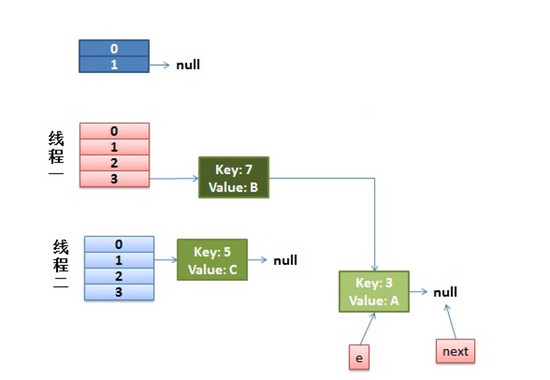
从上面的图我们可以看到，因为线程1的 e 指向了 key(3)，而 next 指向了 key(7)，在线程2 rehash 后，就指向了线程2 rehash 后的链表。

然后线程1被唤醒了：

1. 执行e.next = newTable[i]，于是 key(3)的 next 指向了线程1的新 Hash 表，因为新 Hash 表为空，所以e.next = null，
2. 执行newTable[i] = e，所以线程1的新 Hash 表第一个元素指向了线程2新 Hash 表的 key(3)。好了，e 处理完毕。
3. 执行e = next，将 e 指向 next，所以新的 e 是 key(7)

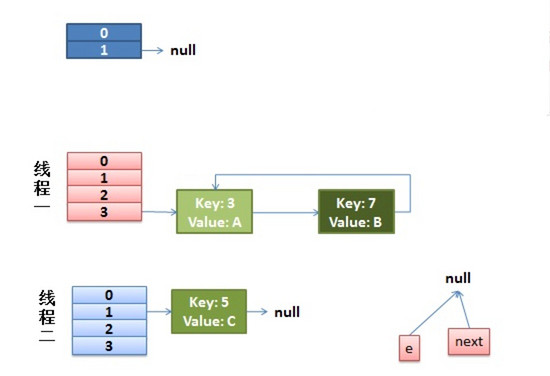
然后该执行 key(3)的 next 节点 key(7)了:

1. 现在的 e 节点是 key(7)，首先执行Entry<K,V> next = e.next,那么 next 就是 key(3)了（而不是key（5）了）
2. 执行e.next = newTable[i]，于是key(7) 的 next 就成了 key(3)
3. 执行newTable[i] = e，那么线程1的新 Hash 表第一个元素变成了 key(7)
4. 执行e = next，将 e 指向 next，所以新的 e 是 key(3)

然后又该执行 key(7)的 next 节点 key(3)了：

1. 现在的 e 节点是 key(3)，首先执行Entry<K,V> next = e.next,那么 next 就是 null
2. 执行e.next = newTable[i]，于是key(3) 的 next 就成了 key(7)
3. 执行newTable[i] = e，那么线程1的新 Hash 表第一个元素变成了 key(3)
4. 执行e = next，将 e 指向 next，所以新的 e 是 key(7)

这时候的状态如图所示：

很明显，环形链表出现了！！当然，现在还没有事情，因为下一个节点是 null，所以transfer()就完成了，等put()的其余过程搞定后，HashMap 的底层实现就是**线程1的新 Hash 表了。**

然后在该hashMap执行查找时就会陷入死循环的操作了。

还有get（）到null值和get（）数据丢失的可能。

## 多线程put的时候可能导致元素丢失

HashMap另外一个并发可能出现的问题是，可能产生元素丢失的现象。

考虑在多线程下put操作时，执行addEntry(hash, key, value, i)，如果有产生哈希碰撞，  
**导致两个线程得到同样的bucketIndex去存储，就可能会出现覆盖丢失的情况。同时存进去的位置，有一个先存的会给覆盖掉。**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | void addEntry(int hash, K key, V value, int bucketIndex) {      //多个线程操作数组的同一个位置      Entry<K,V> e = table[bucketIndex];          table[bucketIndex] = new Entry<K,V>(hash, key, value, e);          if (size++ >= threshold)              resize(2 \* table.length);      } |

### put非null元素后get出来的却是null(同个位置，又或者发生上面的情况，死循环并且有元素丢失)

在transfer方法中代码如下：

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);

}

}

}

注：src为旧数组，newTable为新数组。**src[j] = null;，把**src[j]取值给e后，把那个位置就变成null，那么同时有线程get那个旧表src的时候，就有可能取值为null了。

在这个方法里，将旧数组赋值给src，遍历src，当src的元素非null时，就将src中的该元素置null，即将旧数组中的元素置null了，也就是这一句：

if (e != null) {

src[j] = null;