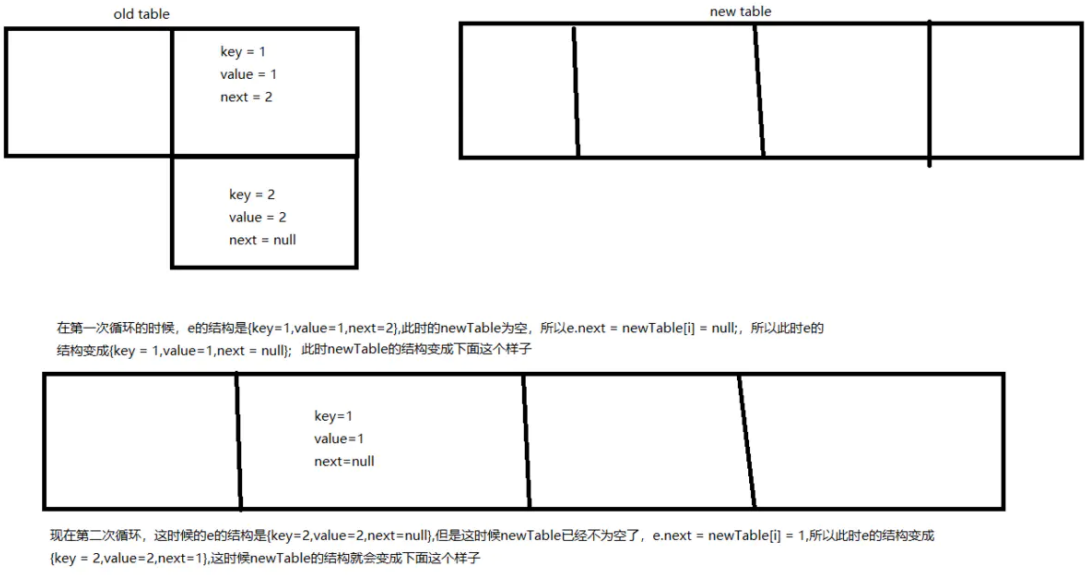
JDK1.7HashMap环链的形成原理(附图)

HashMap在1.8相对1.7做了很多改进，比如红黑树，还有今天要说的环链的形成，之前看别人博客只是说到了1.7版本HashMap在扩容的时候会形成环链，但是没有说到具体原因。

```
*/
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;
        }
    }
}
```

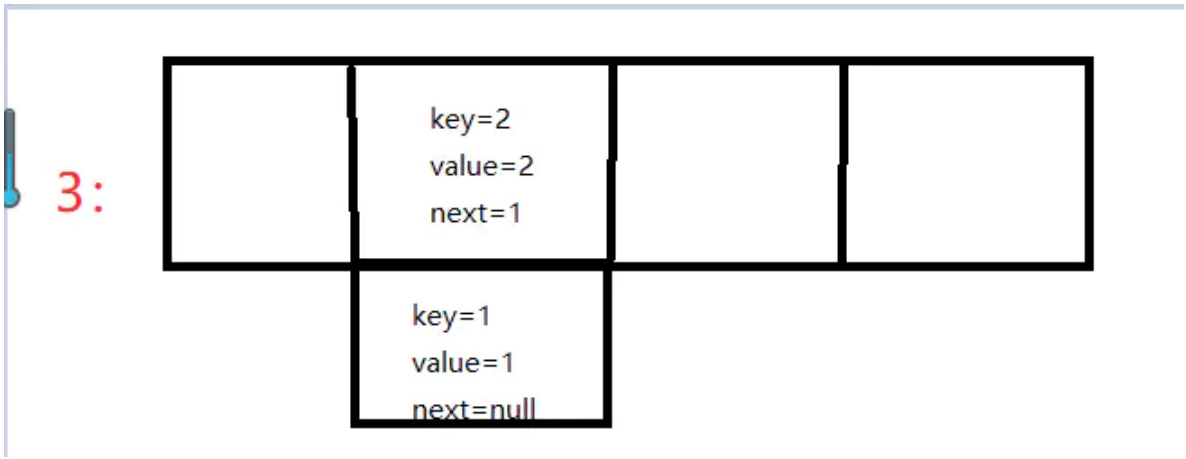
这是jdk1.7中HashMap扩容的源代码，我们的分析也是从这开始的。HashMap是线程不安全的，假如此时有二个线程Thread-1和Thread-2同时进入到这个扩容方法，但是这个时候Thread-2阻塞住了，也就是卡在这没有往下执行。Thrwad-1继续执行下去。



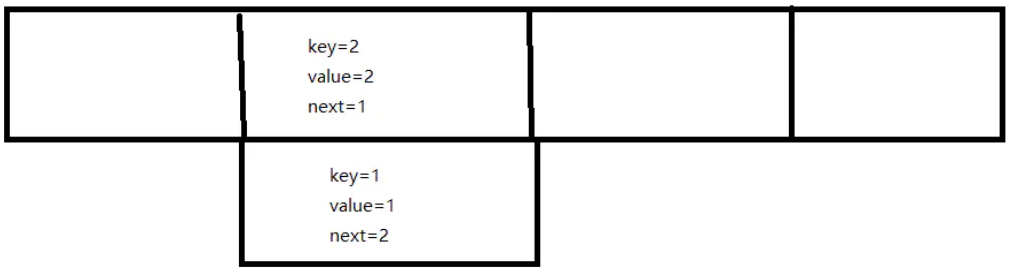
在第一次循环的时候，e的结构是{key=1,value=1,next=2},此时的newTable为空，所以e.next = newTable[i] = null;，所以此时e的结构变成{key = 1,value=1,next = null}；此时newTable的结构变成下面这个样子

现在第二次循环，这时候的e的结构是{key=2,value=2,next=null},但是这时候newTable已经不为空了，e.next = newTable[i] = 1,所以此时e的结构变成{key = 2,value=2,next=1},这时候newTable的结构就会变成下面这个样子

第二幅图文字我在这重复一下：但是这时候Thread-2苏醒了，此时的Thread-2遍历的还是之前的那个old table，所以Thread-2来做第一次循环的时候第一个元素e的结构是{key=1,value=1,next=2},然后Thread-2把这个新的元素移到新的table上去，但是此时的newtable的结构如下



所以在执行e.next = newTable[i] = 2,这个时候这个元素的结构就变成{key=1,value=1,next=2},然后Thread-2把这个元素放到newTable上去，newTable的结构就会变成下面这样



这时候二个元素的下一个节点指向了对方，所以就造成了环链。