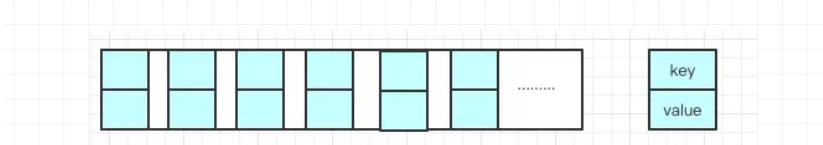
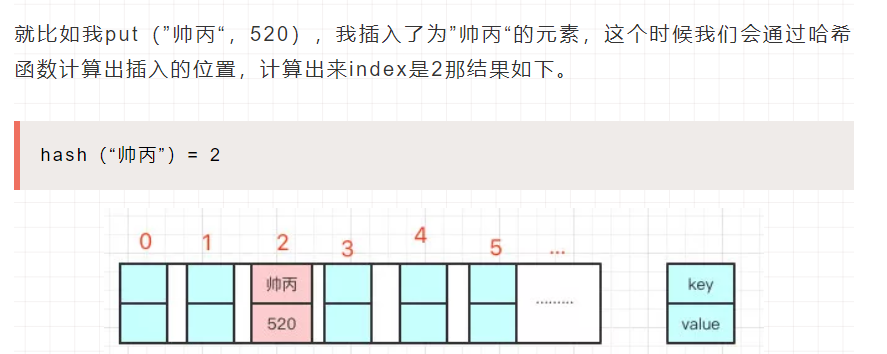


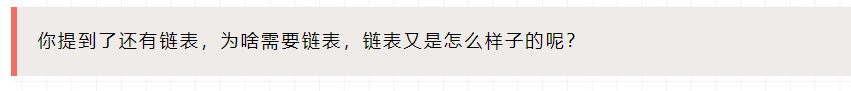
HashMap在我们的开发中用的非常多的，他的底层是由数组和链表组成的。

**数组：**在数组中存在着key-value这样的实例。在java7中我叫他Entry，在java8中我们称他为Node。

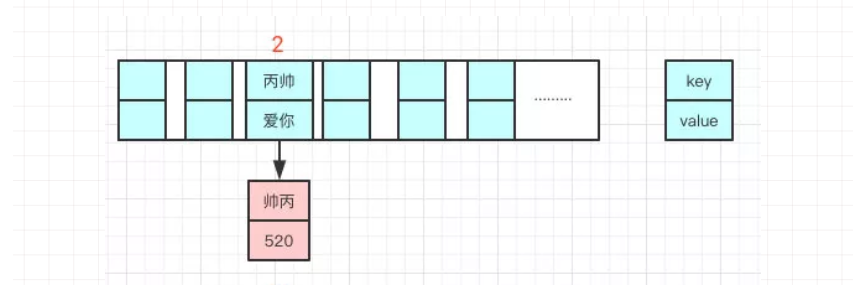


在这个数组中一开始全部都是null，当我们进行put的时候根据key的hash去计算他的index（位置）



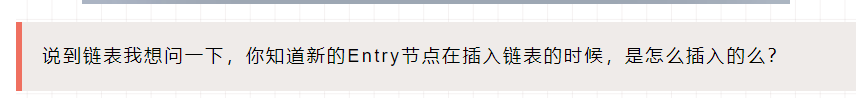


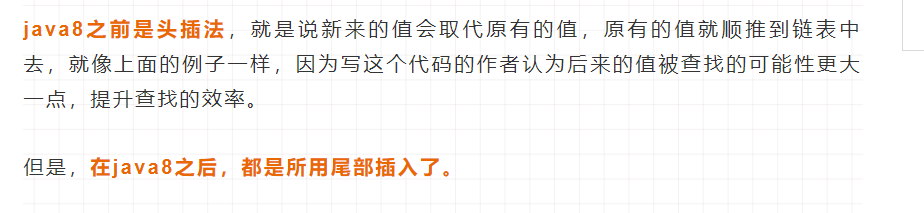
因为我们每次进行put的时候是通过key的hash去计算index（位置）就有可能导致hash碰撞【即不同的key，会在相同位置】，在java8以前采用头插法，java8以后采用的是尾插法



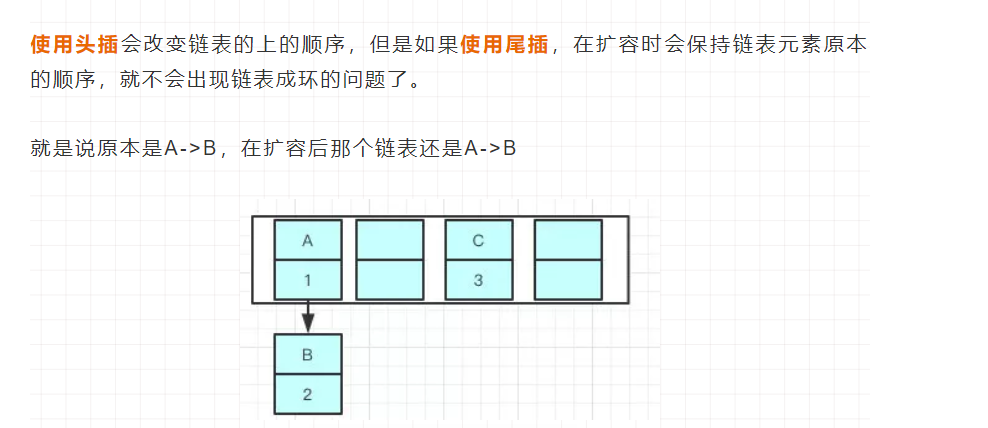
每一个Node都会保存自身的key、value、hash以及下一个Node节点

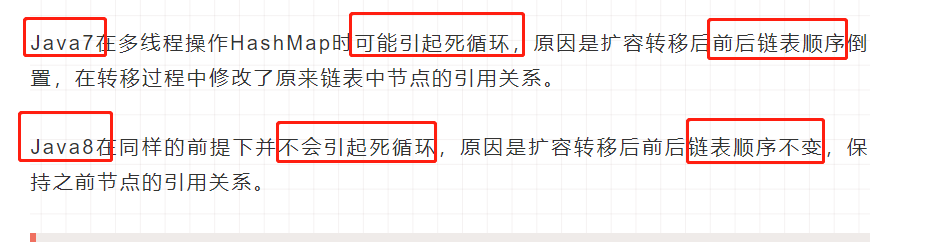


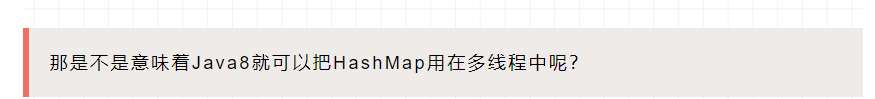




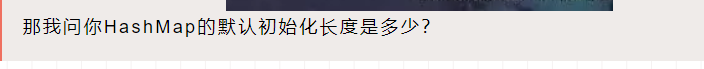








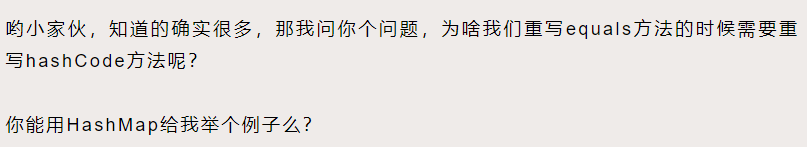
我认为即使是java8不会在多线程的情况下产生死循环，但是你看他的源代码的中put以及get的方法你就会了解到他们这两个的方法其实是没有加锁，所以你无法保证上一秒put的value值和下一秒get的value是一样。所以说在java8中使用HashMap的多线程情况下也是不安全的。



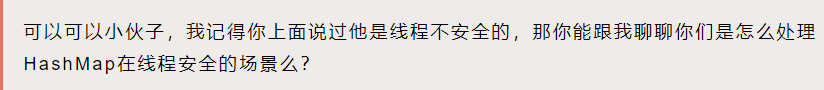
我记得我在看源码的时候初始化大小是16



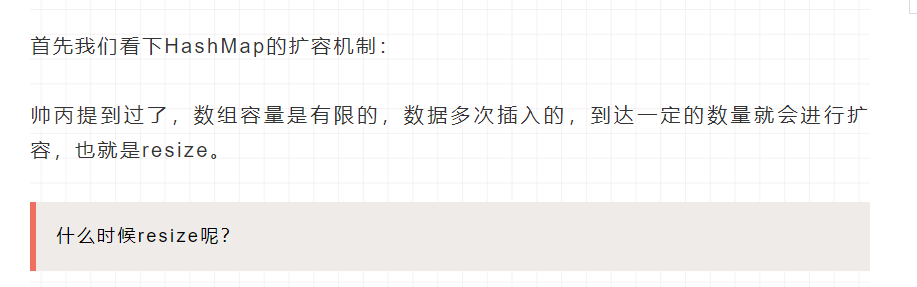
通过计算hash的规则发现HashMap的初始化的大小在16的情况下，产生碰撞的几率比较小。

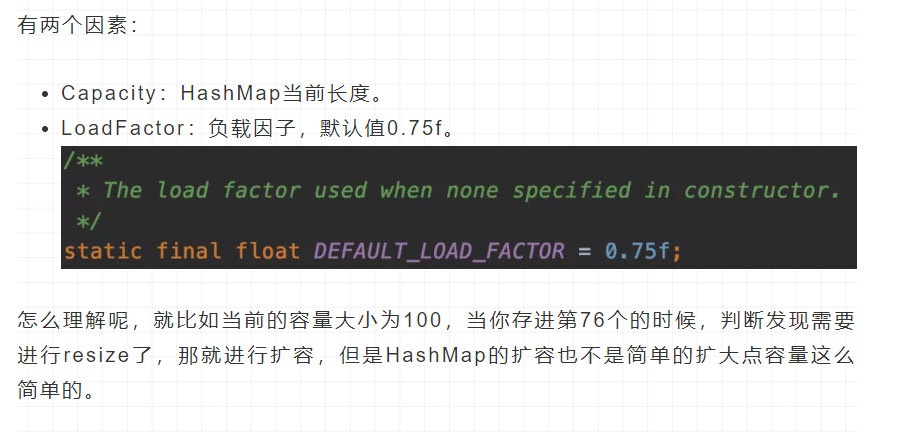


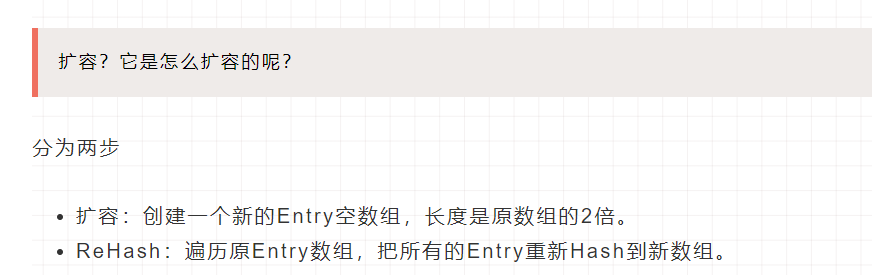


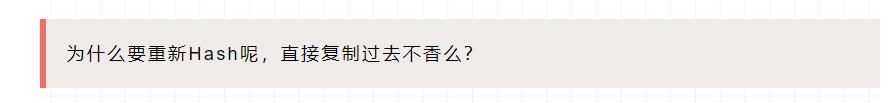


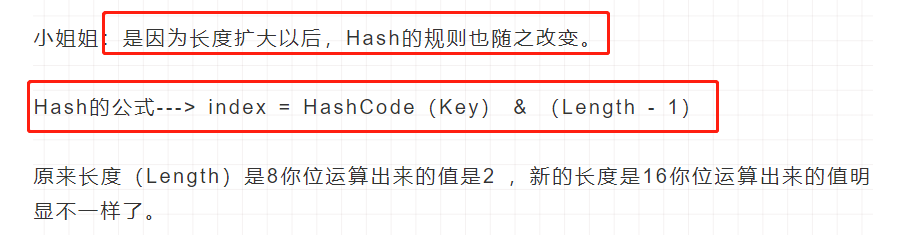


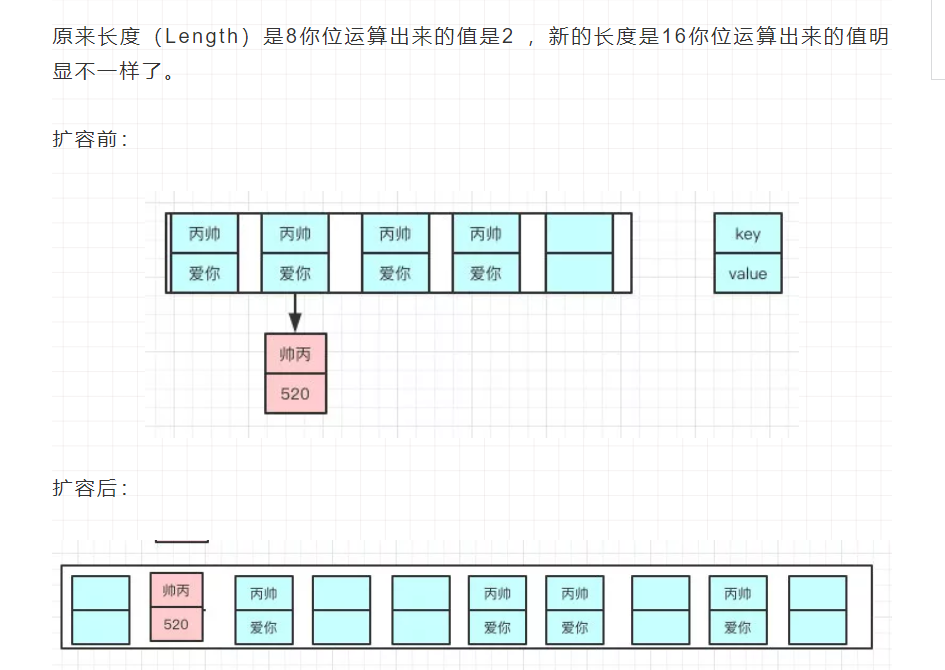






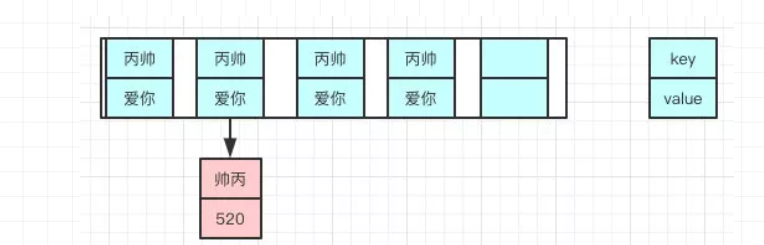




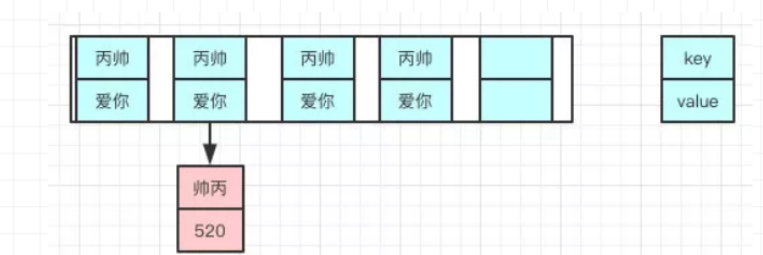




1. HashMap的底层数据结构？
   1. 数组和链表
2. HashMap的存取原理？
   1. Put
      1. 当我使用HashMap中的put(key,value)方法的时候，首先会调用一个key.hashCode()返回一个hash值，
      2. 从而确定Entry的位置（index）,当这个index上次不存在key的时候，会直接添加进去。
      3. 当这个位置上面存在key的时候，会使equal的方法判断这两个key是否相同
      4. 相同的话会把原来的key对应的value值，替换成新添加进来的值。
      5. 不相同的话，在java8之前使用头插法，将原来的key做成下一个节点，新的key放在原来key的位置。Java8以后采用尾插法查用新的key形成链表



* 1. Get（key）
     1. 首先会调用key的hashCode的方法返回hash值，从而确定index，再调用equal的方法判断链表中的那个值。
  2. Java7和Java8的区别？
     1. Java7采用的是头插法
     2. Java8采用的是尾插法
  3. 为啥会线程不安全？
     1. 因为HashMap中put以及get方法都没有加锁，所以无法保证上一秒所put的值和下一秒所get的值是一样。
  4. 有什么线程安全的类代替么?
     1. HashTable【在所有的方法上面都加了锁，所以并行度比较低】
     2. CurrentHashMap
  5. 默认初始化大小是多少？为啥是这么多？为啥大小都是2的幂？
     1. 16
     2. 这是为了**实现均匀分布，**这么减少hash碰撞
  6. HashMap的扩容方式？负载因子是多少？为什是这么多？
     1. 扩容方式
        1. 首先原来的HashMap的长度扩大为2倍
        2. 遍历所有Entry的值并且重新hash赋值给新的HashMap
        3. 重新hash是因为hash的计算规则跟HashMap长度有关，得重新hash再赋值给新的HashMap中.
     2. 0.75
     3. 尽可能的减少hash碰撞和增加HashMap的空间使用率
  7. HashMap的主要参数都有哪些？
     1. 默认的初始化容量
     2. 容量的最大值
     3. 默认的加载因子
  8. HashMap是怎么处理hash碰撞的？
     1. 将碰撞的hash对应key在原来的key上面组成链表



* 1. hash的计算规则？



参考：<https://mp.weixin.qq.com/s/0Gf2DzuzgEx0i3mHVvhKNQ>