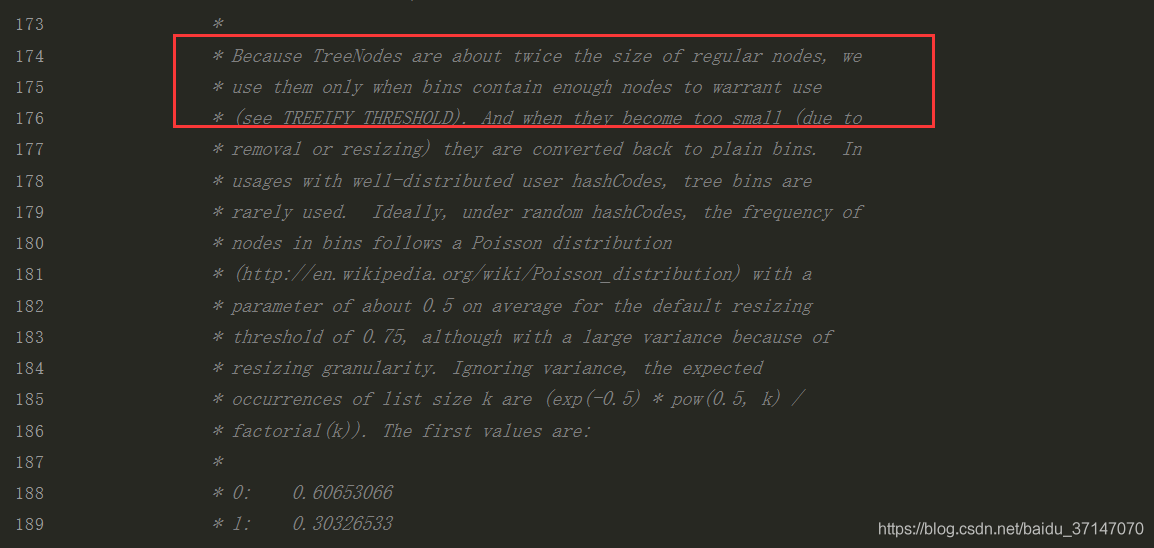
## HashMap

### JDK1.8以后的hashmap为什么在链表长度为8的时候变为红黑树

来自：<https://blog.csdn.net/baidu_37147070/article/details/98785367>

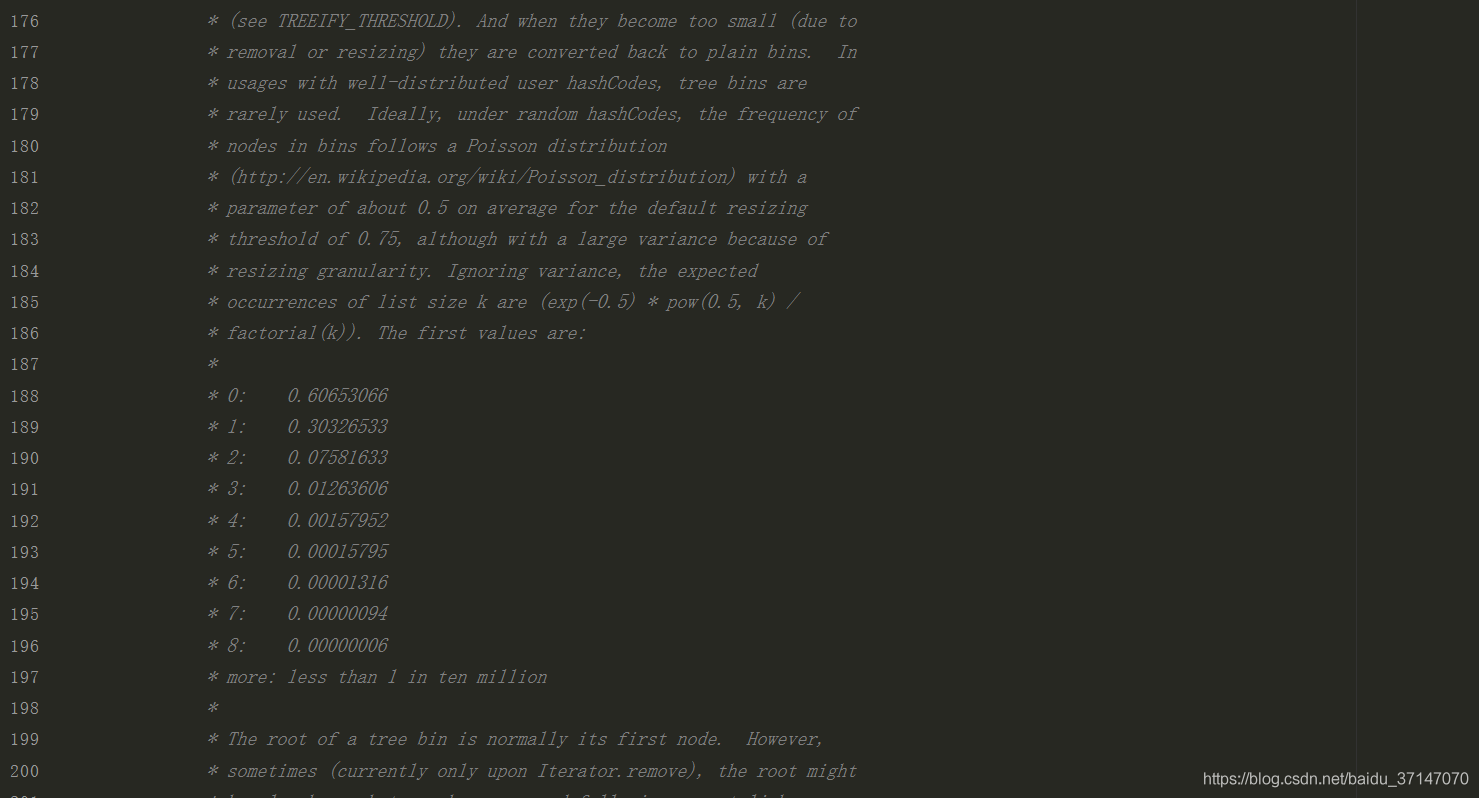
在JDK1.8以及以后的版本中，hashmap的底层结构，由原来单纯的的数组+链表，更改为链表长度为8时，开始由链表转换为红黑树，为何大刀阔斧的对hashmap采取这个改变呢，以及为何链表长度为8才转变为红黑树呢，下面结合源码一起来分析一下。

我们都知道，链表的时间复杂度是O(n)，红黑树的时间复杂度O(logn)，很显然，红黑树的复杂度是优于链表的，既然这么棒，那为什么hashmap为什么不直接就用红黑树呢，请看下图

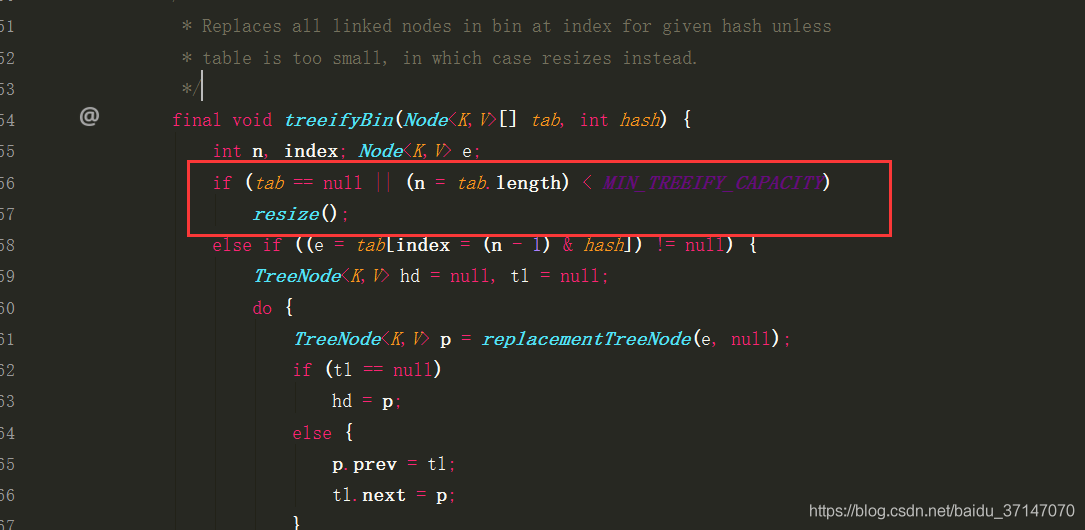


源码中的注释写的很清楚，因为树节点所占空间是普通节点的两倍，所以只有当节点足够多的时候，才会使用树节点。也就是说，节点少的时候，尽管时间复杂度上，红黑树比链表好一点，但是红黑树所占空间比较大，综合考虑，认为只能在节点太多的时候，红黑树占空间大这一劣势不太明显的时候，才会舍弃链表，使用红黑树。

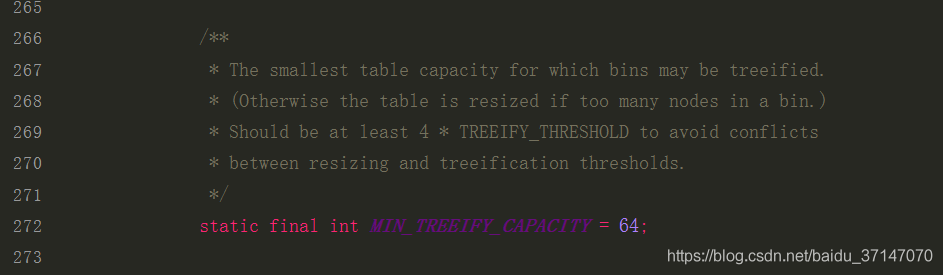
  那为什么选择8才会选择使用红黑树呢？看下图



源码上说，为了配合使用分布良好的hashCode，树节点很少使用。并且在理想状态下，受随机分布的hashCode影响，链表中的节点遵循泊松分布，而且根据统计，链表中节点数是8的概率已经接近千分之一，而且此时链表的性能已经很差了。所以在这种比较罕见和极端的情况下，才会把链表转变为红黑树。因为链表转换为红黑树也是需要消耗性能的，特殊情况特殊处理，为了挽回性能，权衡之下，才使用红黑树，提高性能。也就是大部分情况下，hashmap还是使用的链表，如果是理想的均匀分布，节点数不到8，hashmap就自动扩容了。为什么这么说呢，再看下图



在链表转变为红黑树方法中，有这样一个判断，数组长度小于MIN\_TREEIFY\_CAPACITY，就会扩容，而不是直接转变为红黑树，可不是什么链表长度为8就变为红黑树，要仔细看代码，还有别的条件，



现在回头想想，为啥用8？因为通常情况下，链表长度很难达到8，但是特殊情况下链表长度为8，哈希表容量又很大，造成链表性能很差的时候，只能采用红黑树提高性能，这是一种应对策略。