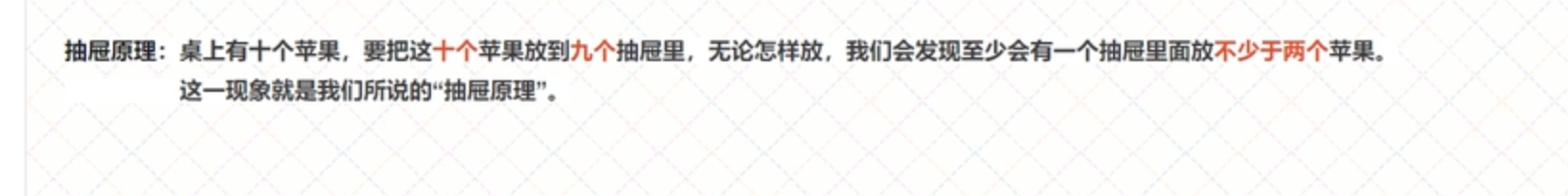
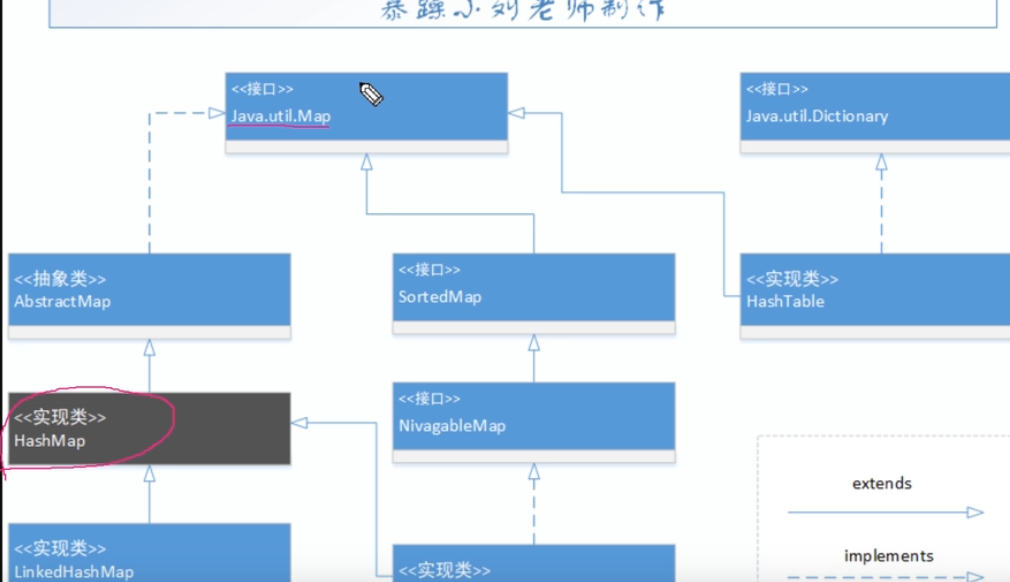
# HashMap

16 64 8 0.75 tableforsize 2的次方

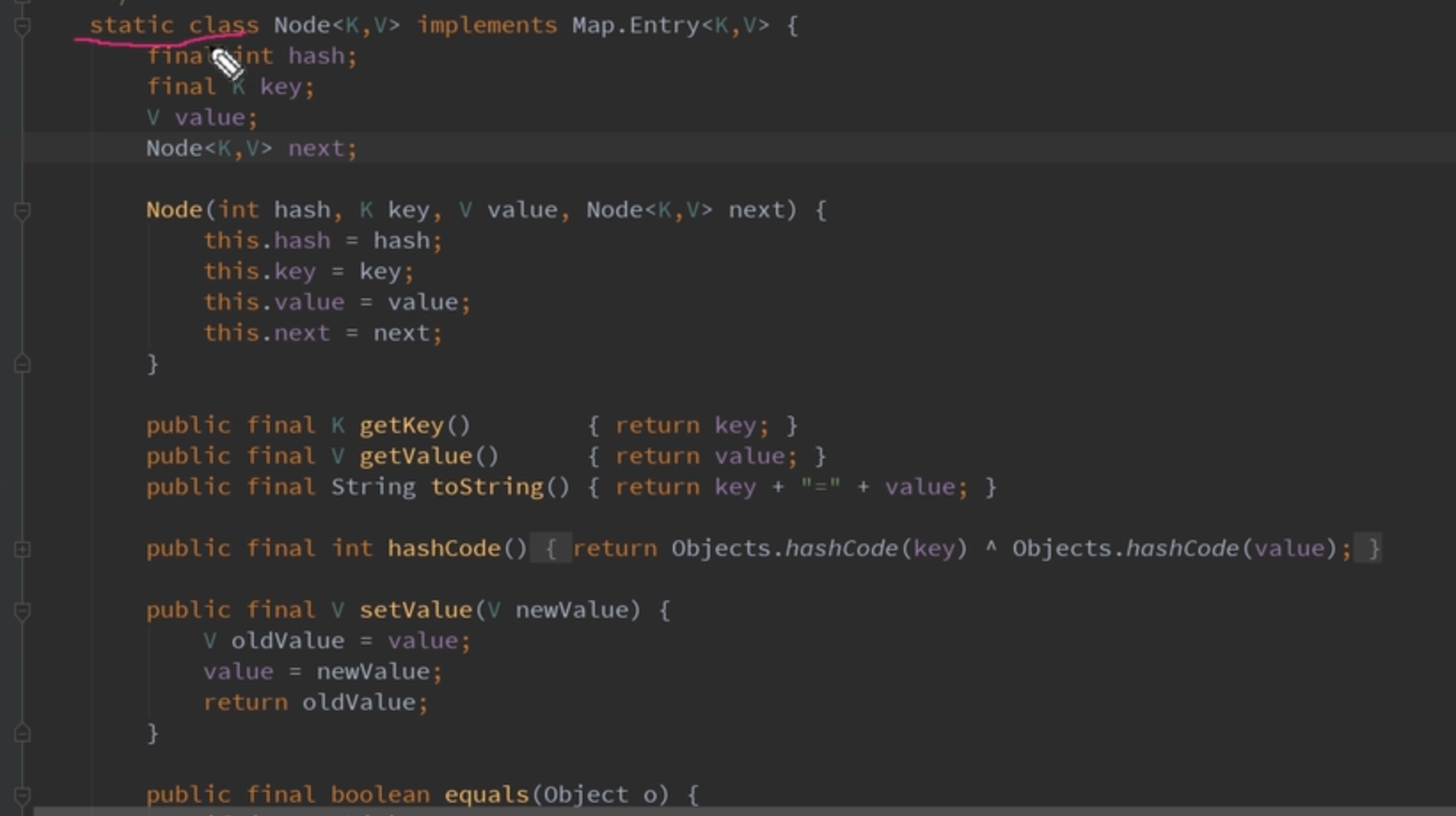
HashMap 要兼顾的是put get 方法的性能 和内存空间

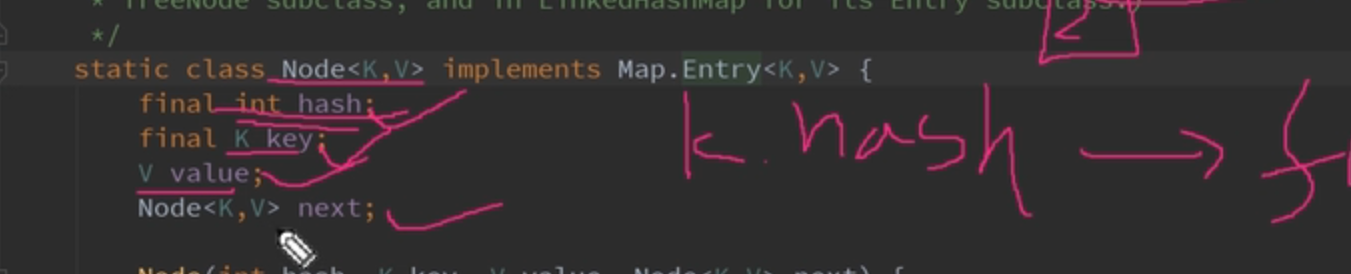




继承体系

Node 是hashMap 中的 静态内部类



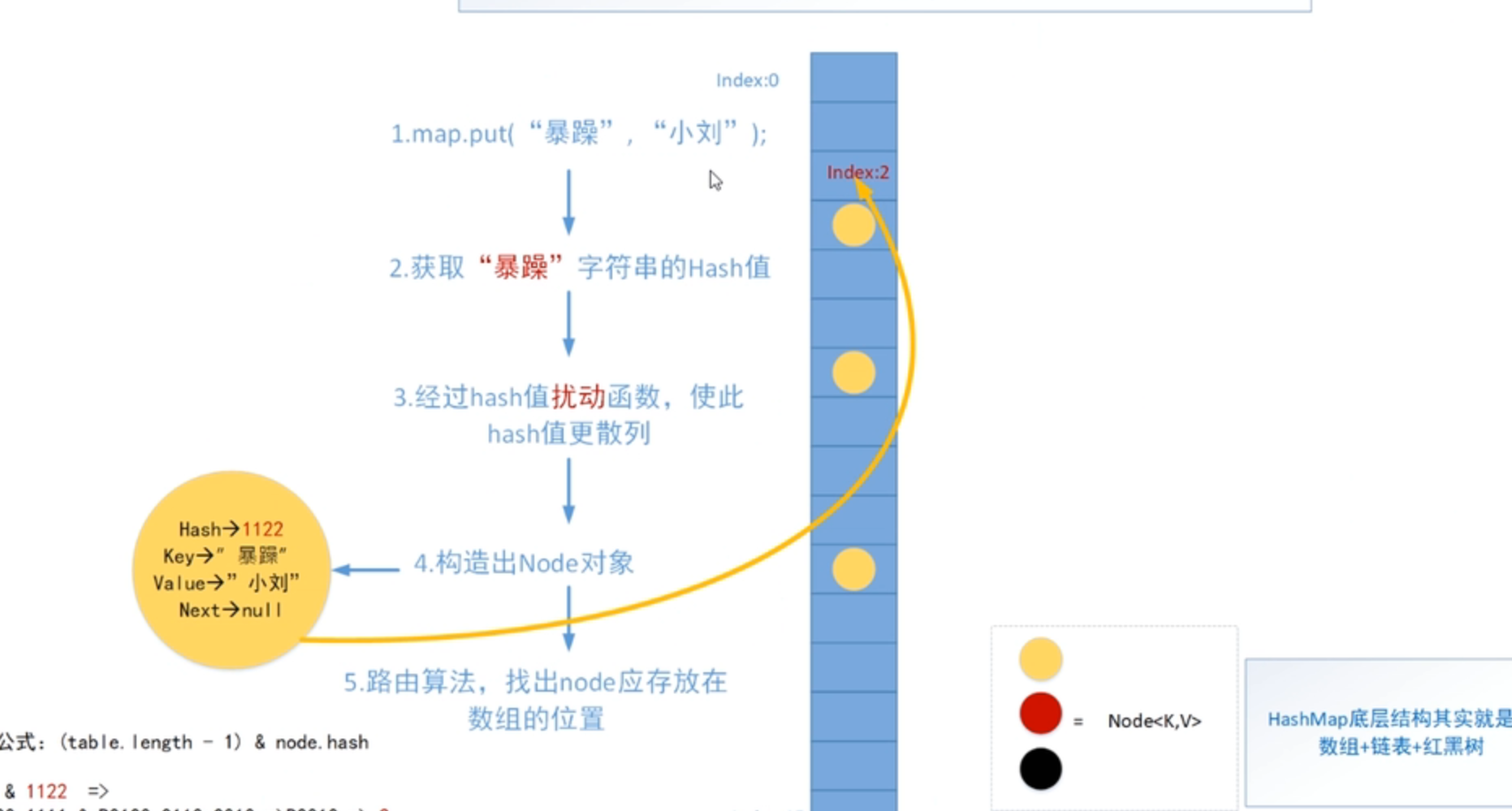


静态内部类

Node 就是散列表中的元素

数组加链表 加红黑树

桶位计算公式 h



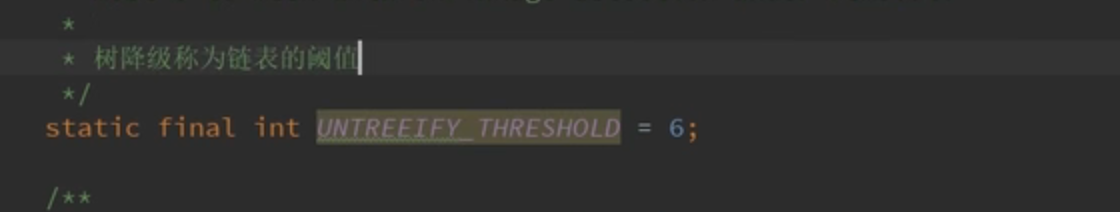
解决链化问题 可以提高查找效率

扩容是为了提高查找性能

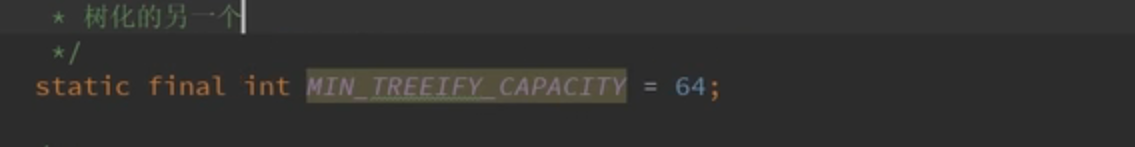
默认16

缺省负载因子0.75

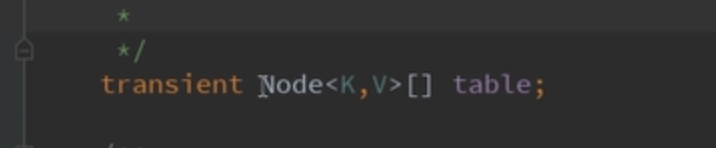
Treeify 8



树化阈值有两个

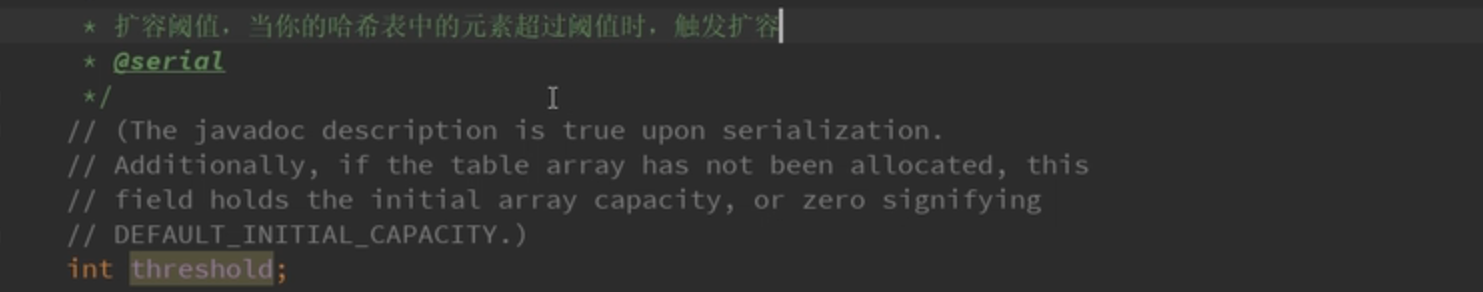


散列表结构

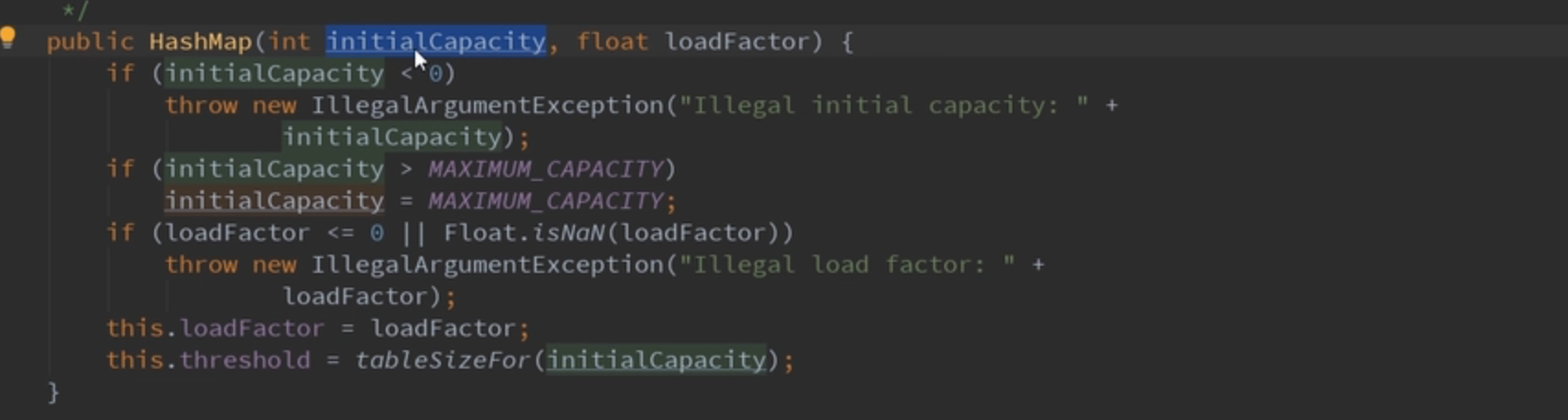


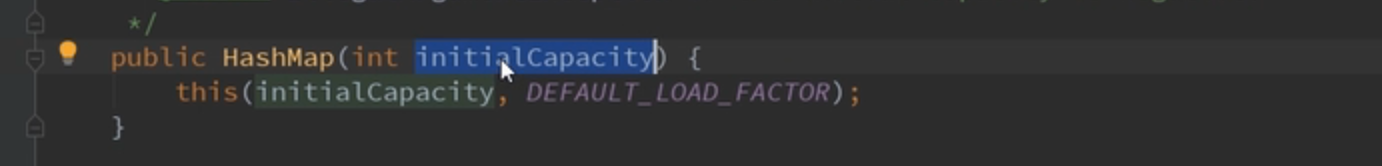


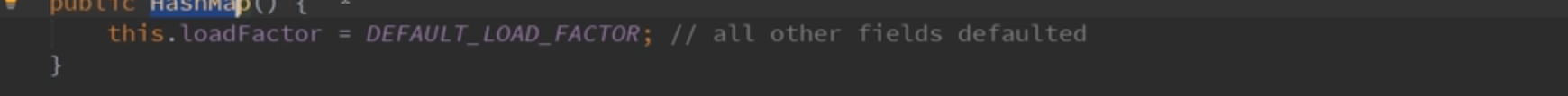


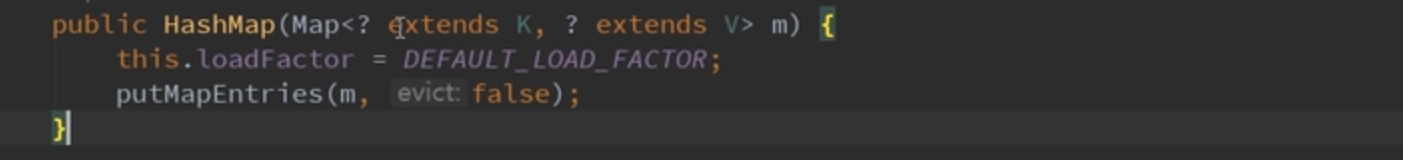


## 构造方法



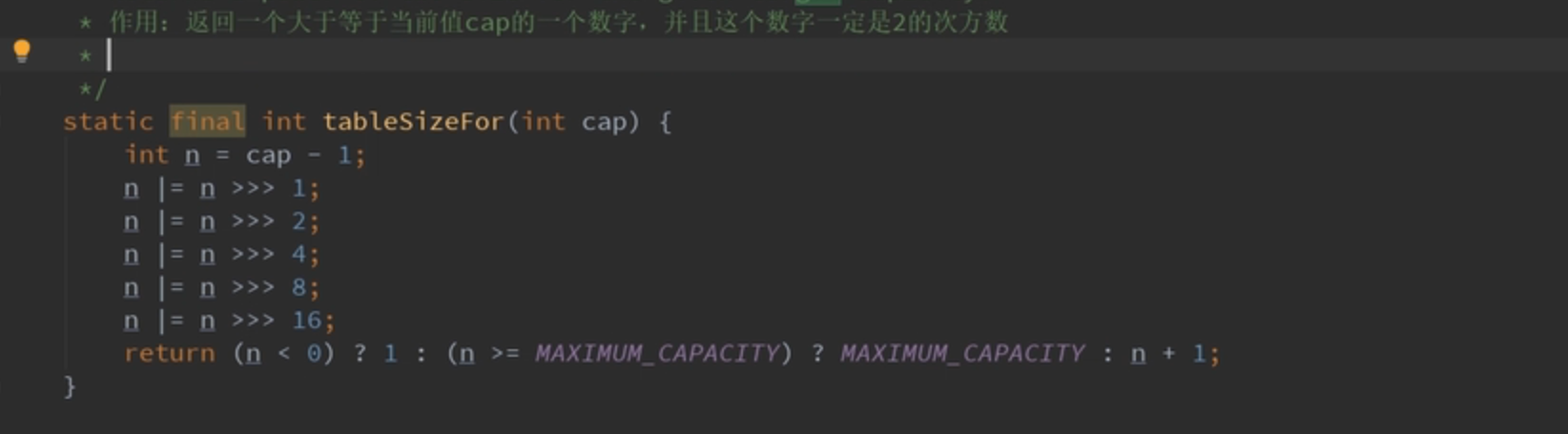




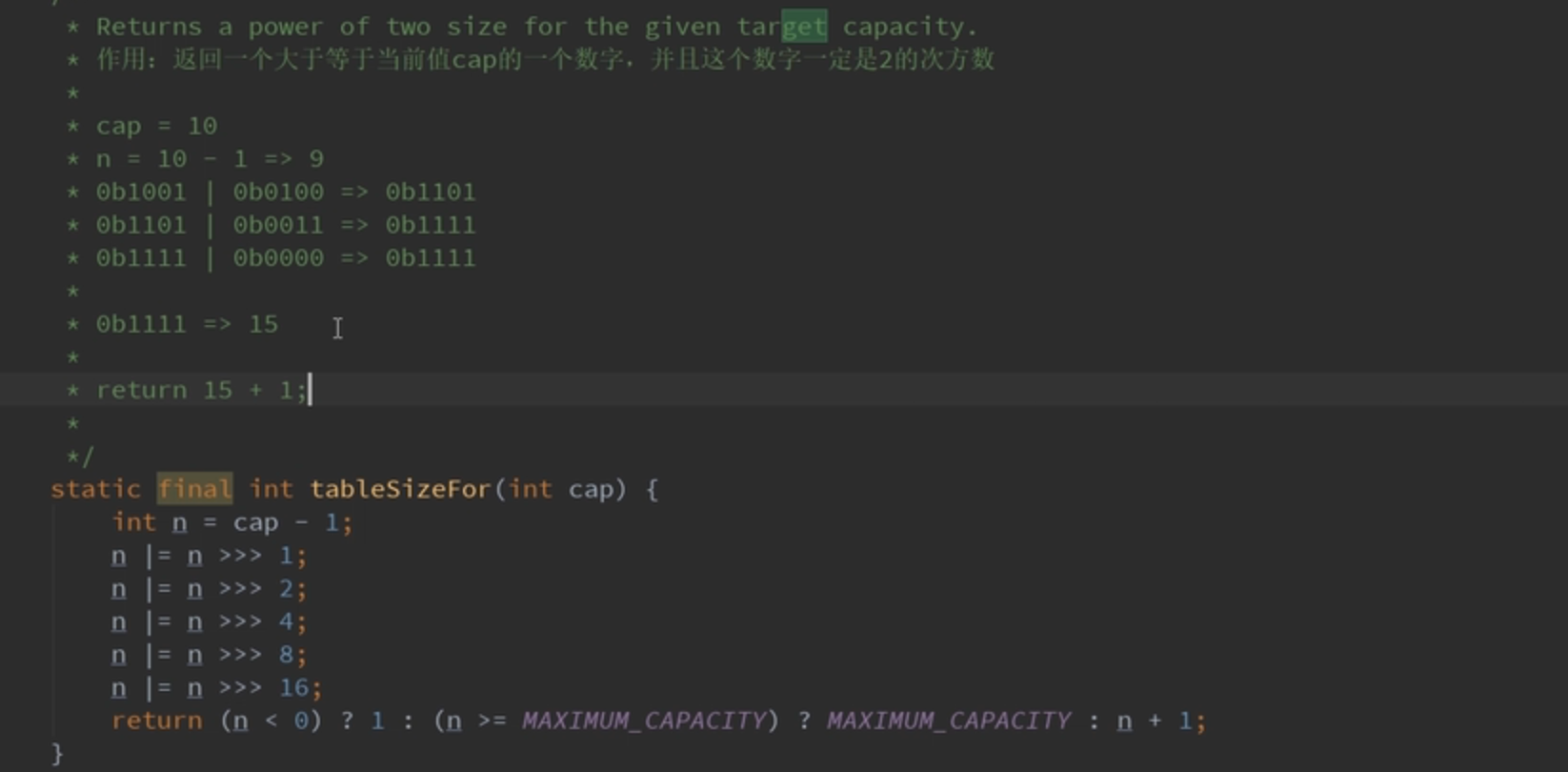


## 初始Threshold 计算方式





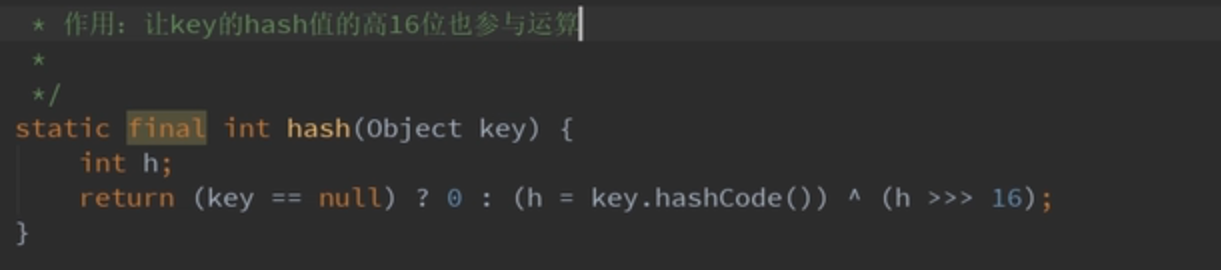
根据大小 跟定初始的table size

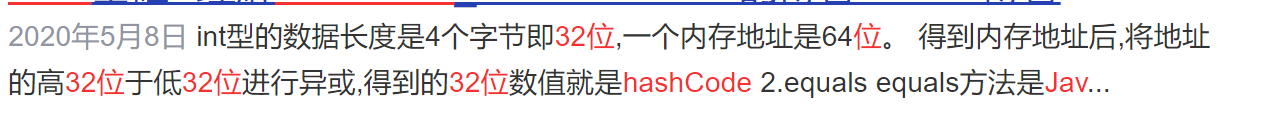


扩容阈值只能是2的次方数

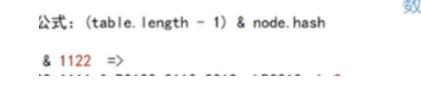
## Key 的最终hash值

使得高位16位置与地位16位置 进行异或运算 使得三十二位都参与 桶位置的计算



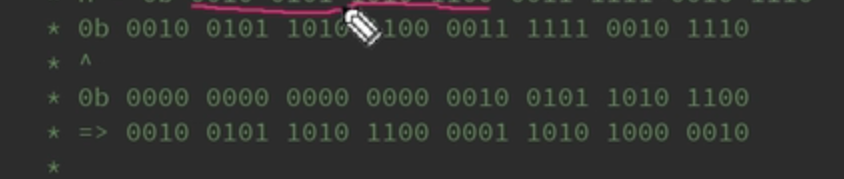


木桶位置计算公式是



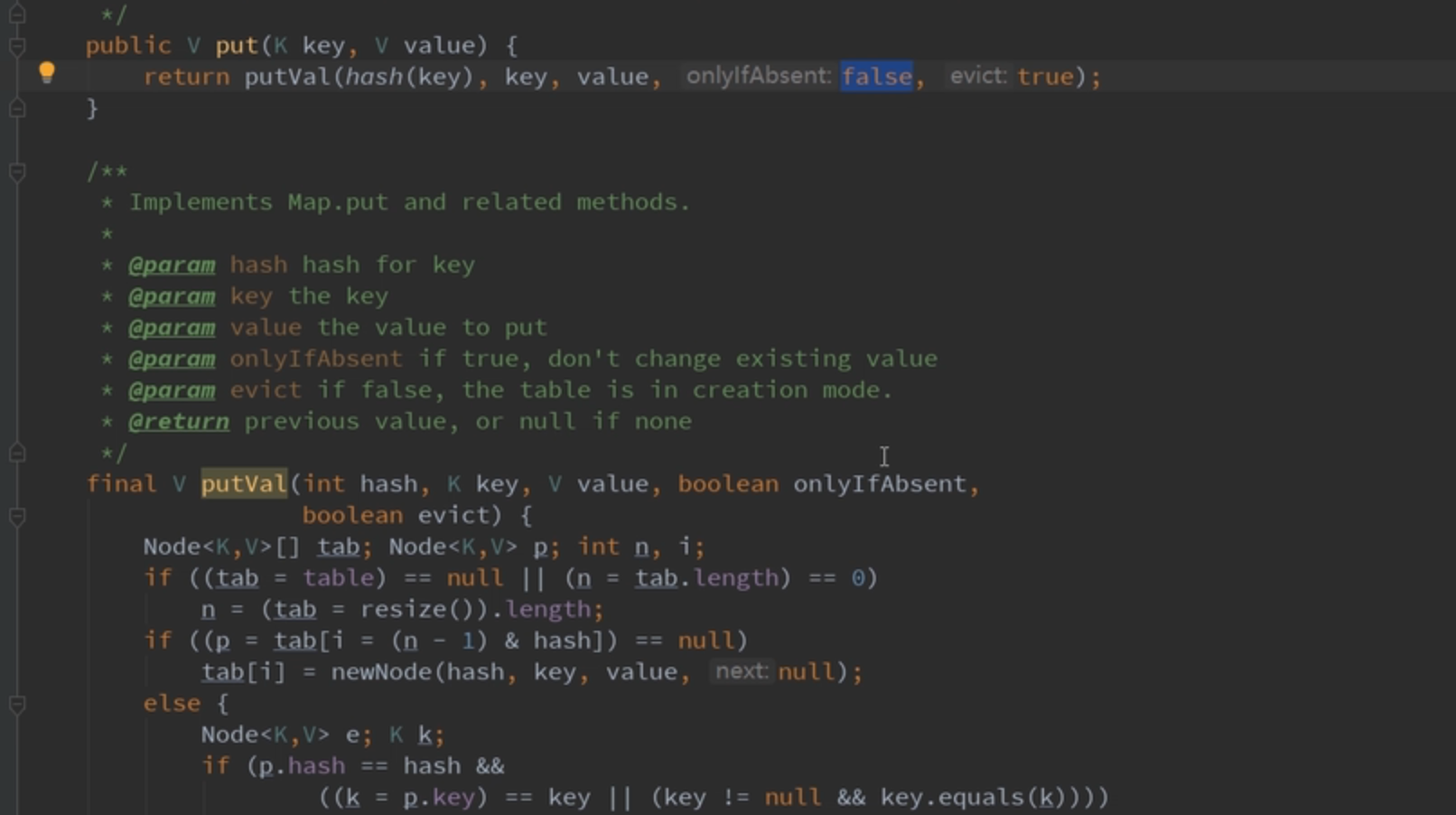
H 计算公式 hash 值与向右移动 的16位 进行亦或运算

把高位的hash 值也参与到桶的位置的计算dang'zh

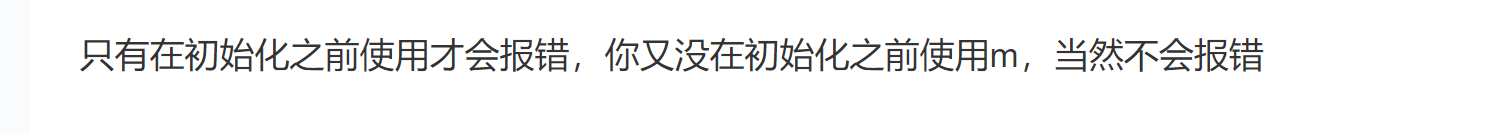


Putvalue

方法

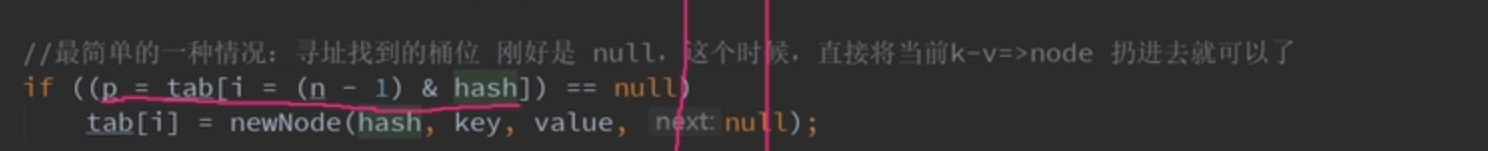


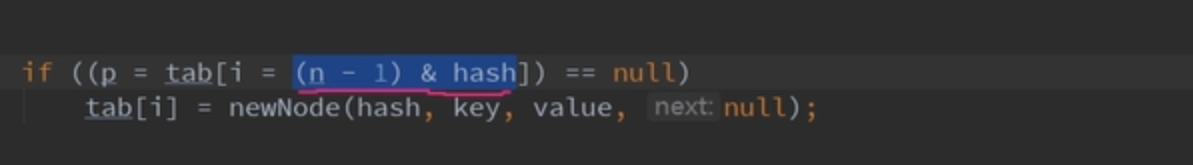
Java 局部变量没有报错



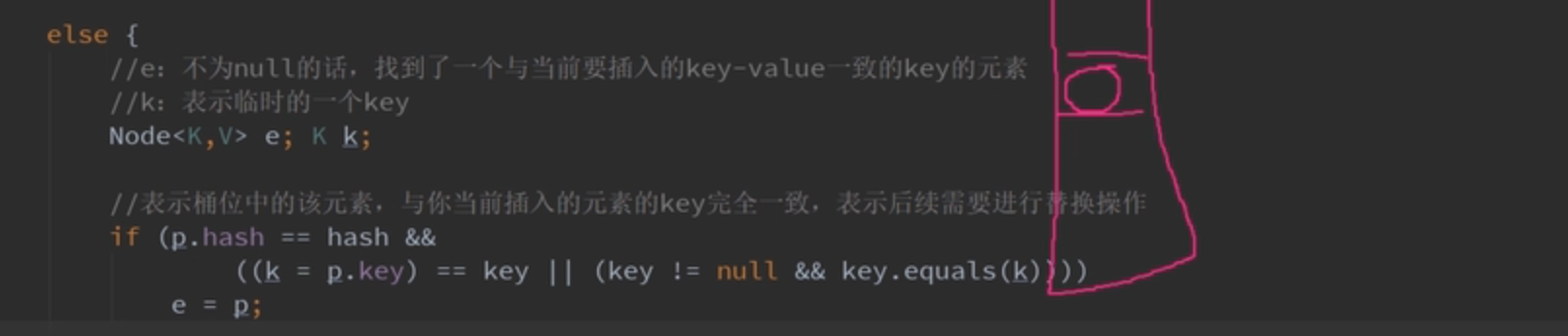
计算桶位置

如果没有数据 直接放入

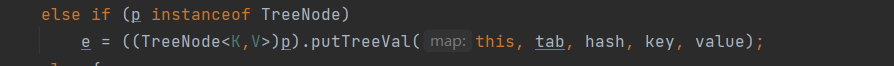




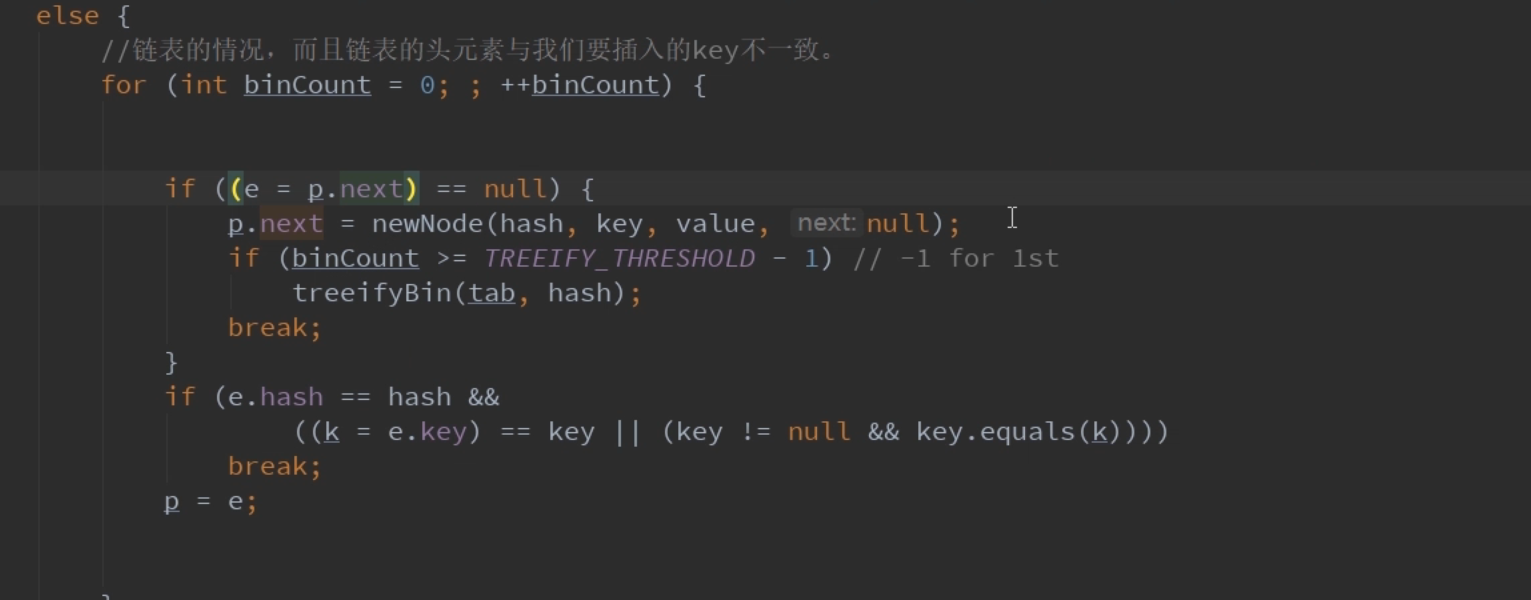
如果桶位置相同 将node 临时存入 node e



如果根据 算出的index 是树，放入tree



否则是链表



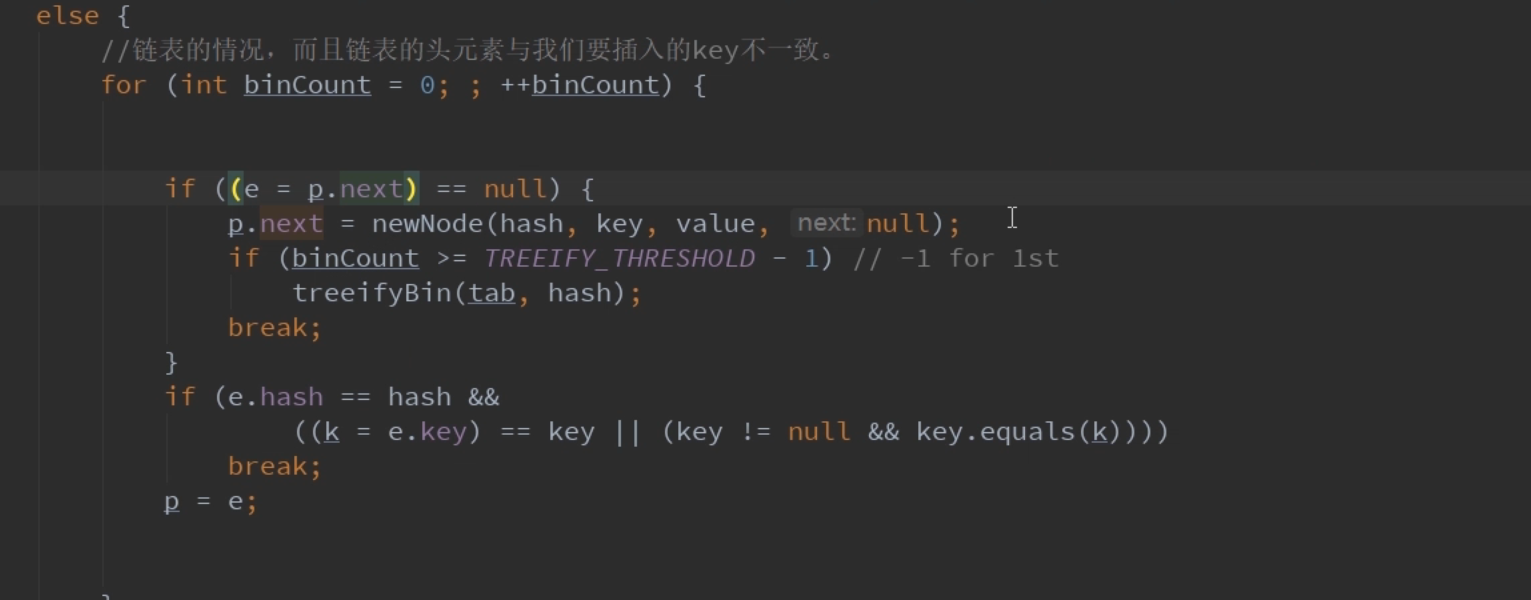
找到e 之后替换



遍历到链表最后一个

加入，加完之后 校验，如果

触发树化操作

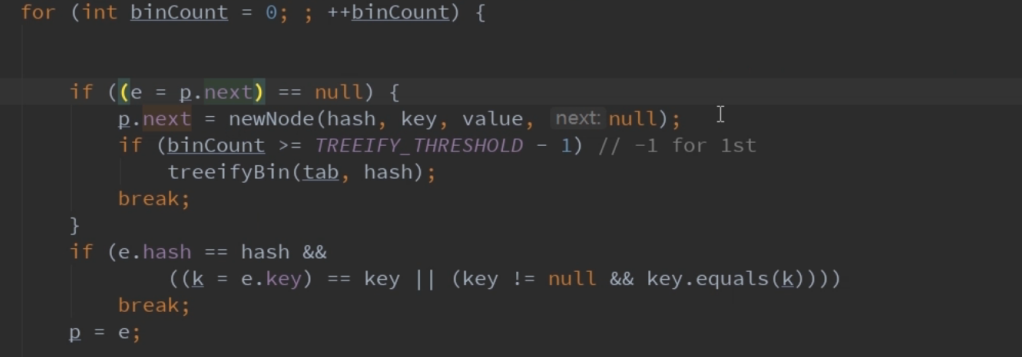


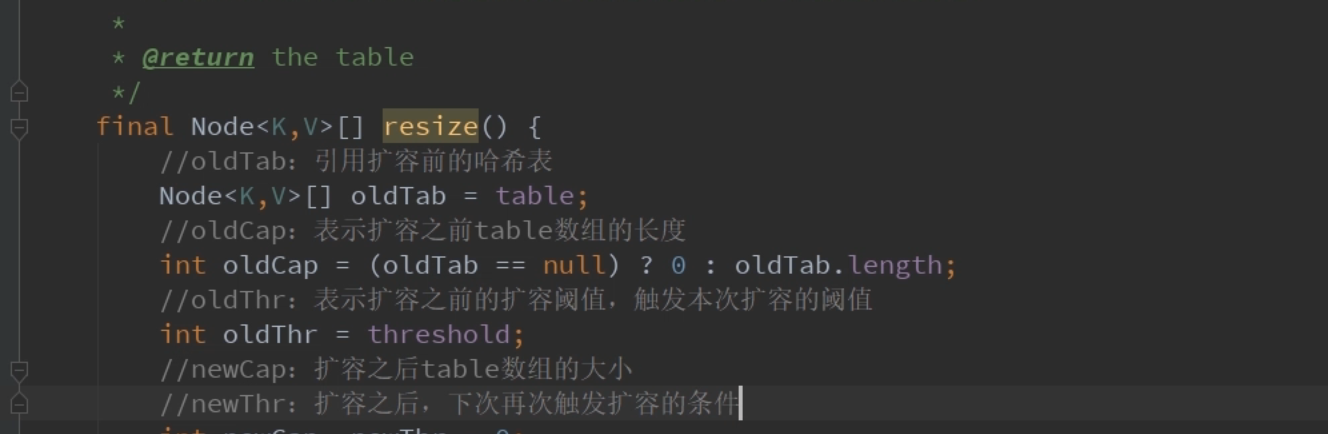
# Resize 方法详解

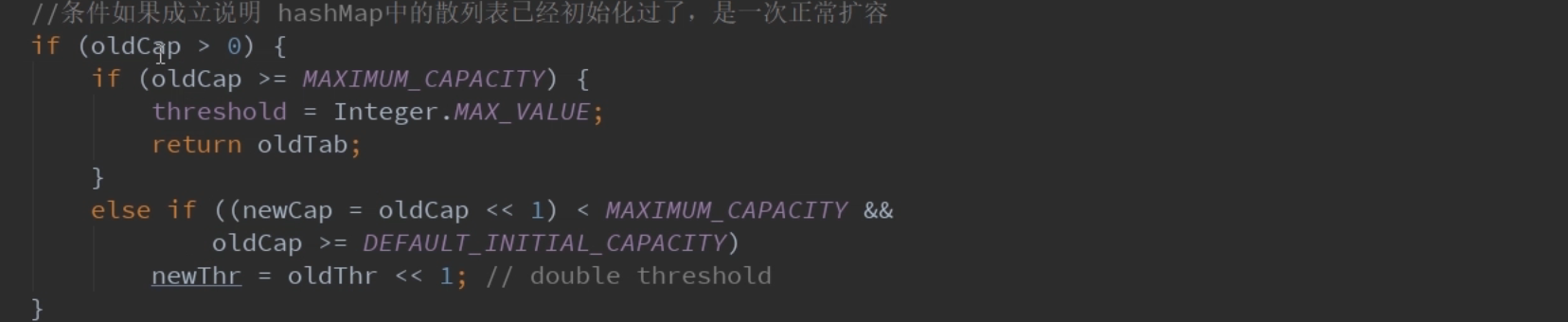
为什么需要扩容

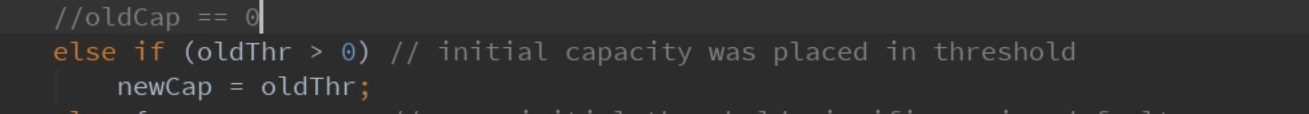
性能 防止链表过长

## newCap newThr

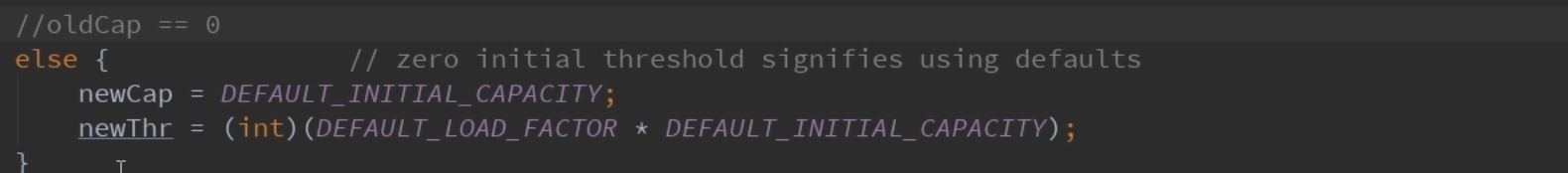


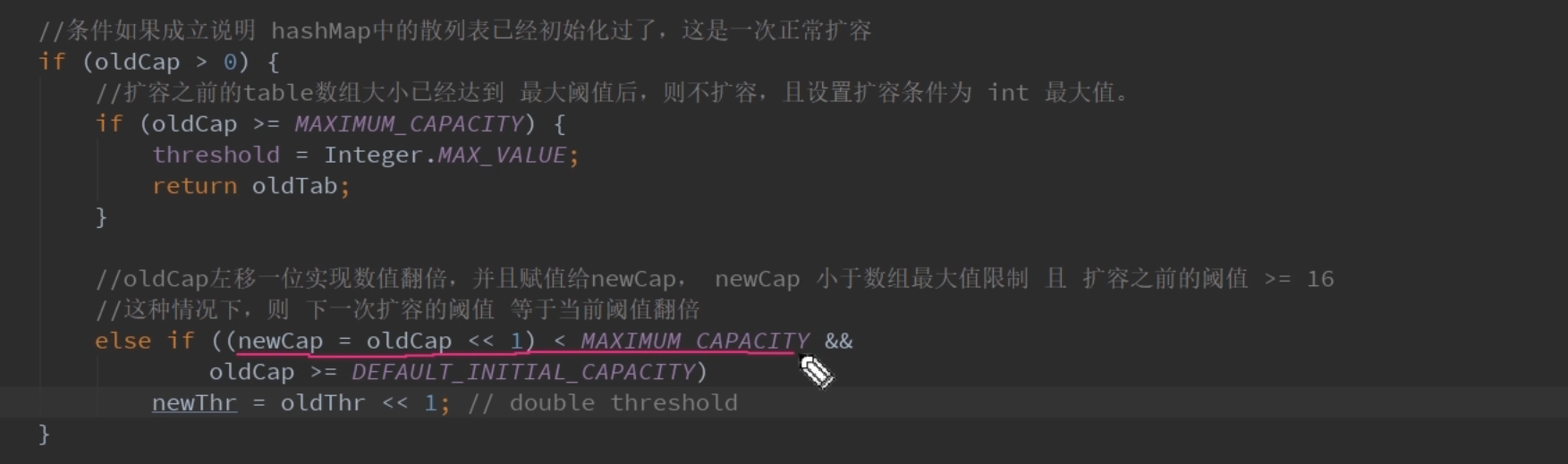


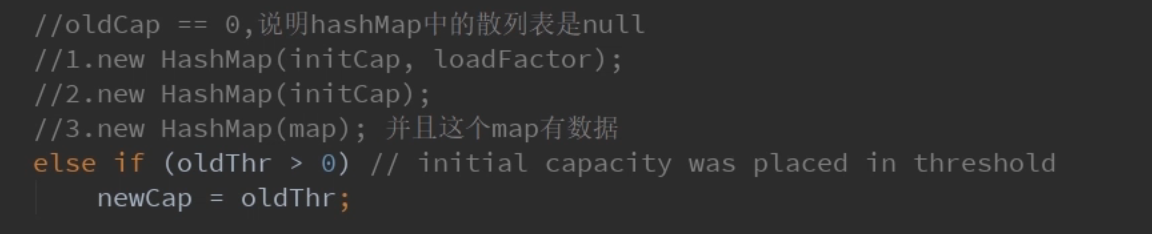


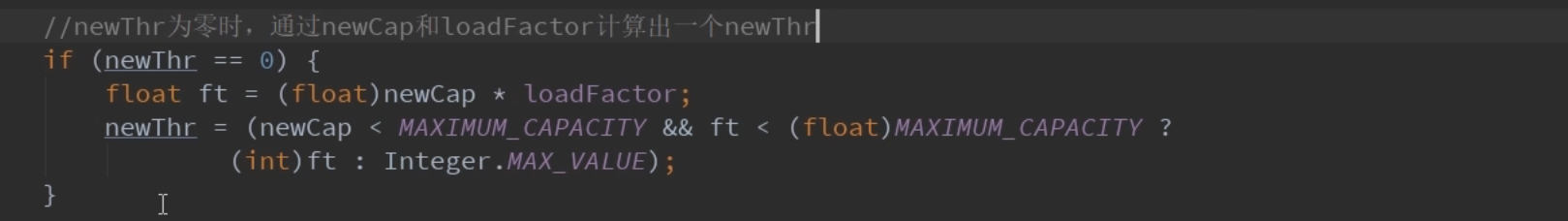


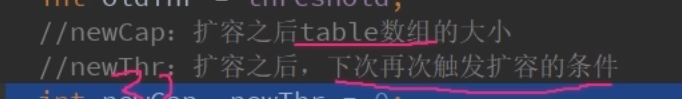
这里写错了 是Threshold = 0



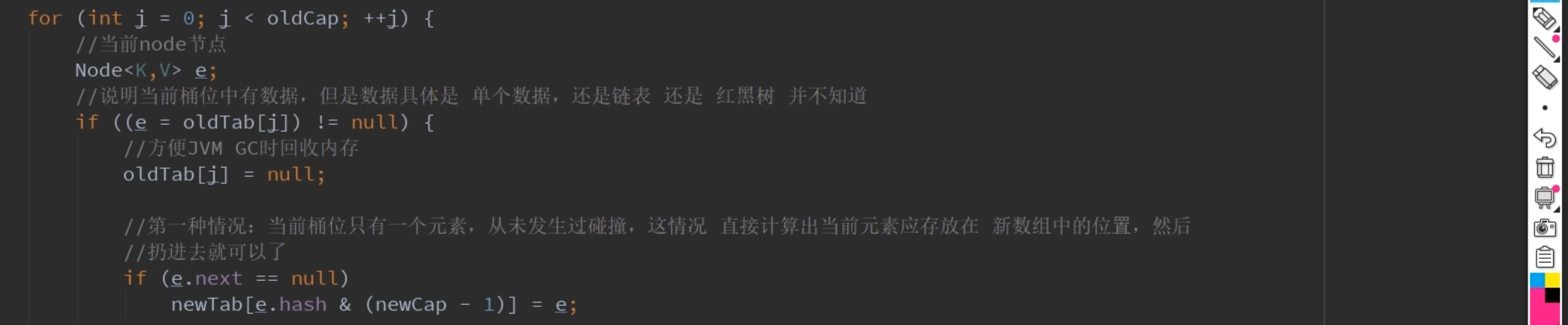


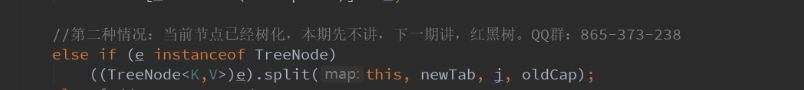




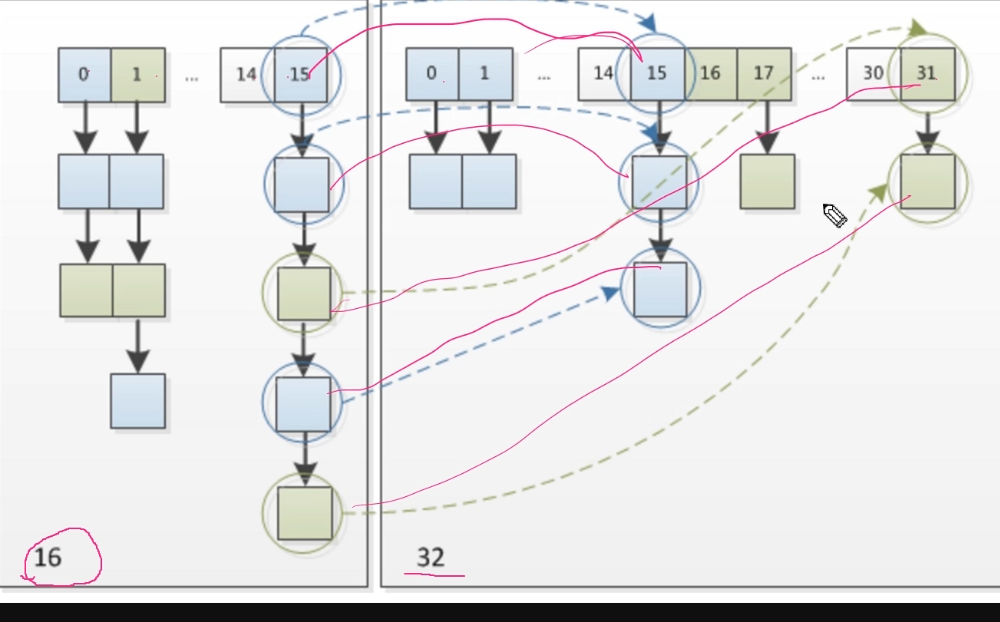


## 将原数据复制到新扩容的列表中





### 扩容中链表的处理



# 为什么是2的n次方：

性能考虑，位运算最快。

为了更快的性能。Hash 计算node 需要防止的table 索引位置 是与与运算，更快，

扩容时候 也是通过位运算 而不是取余或者乘法， 位运算最快。