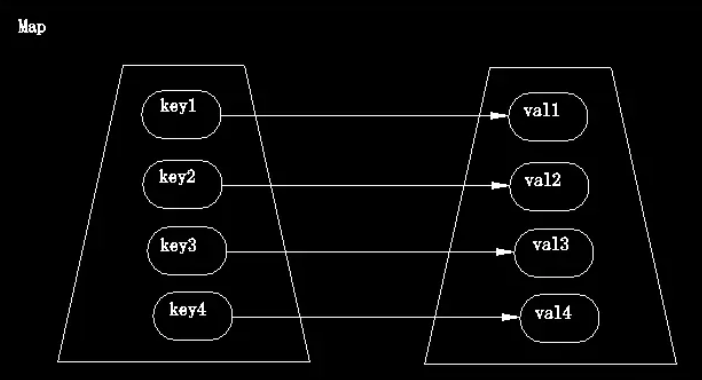
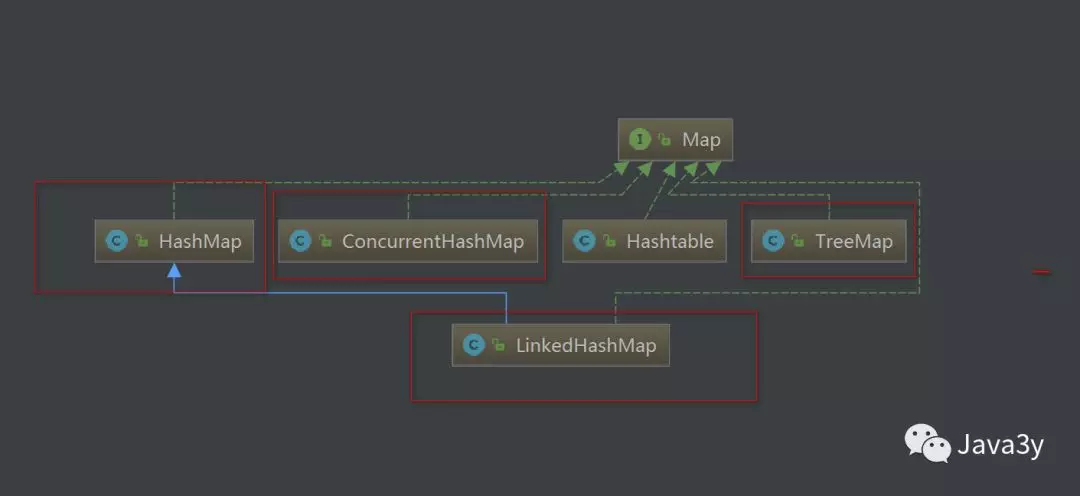
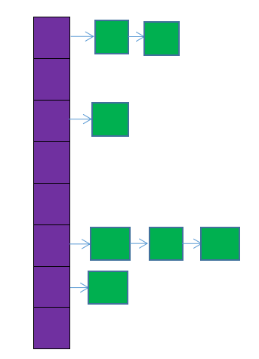
**Map接口**HashMap、Hashtable





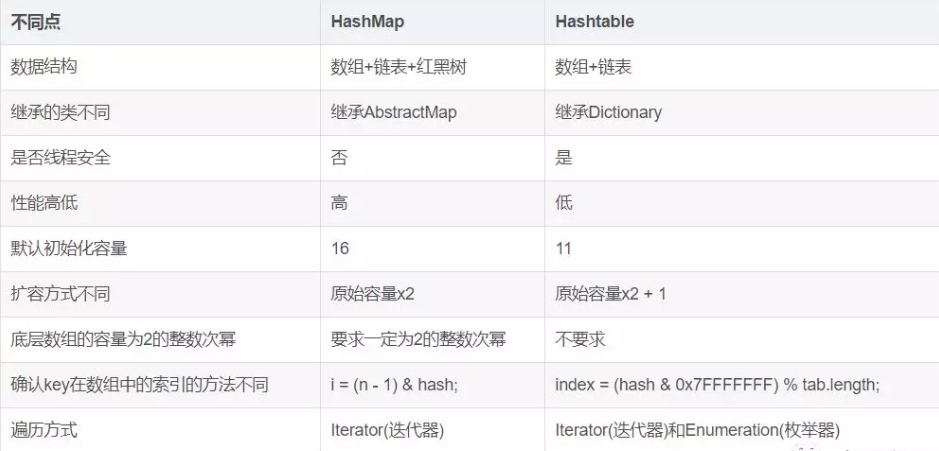


紫色部分代表哈希表（是Node[]数组）

数组的每个元素都是一个单链表的头节点

数组中的每个桶存放的是一条链表，链表中的每个节点用来存放键值对

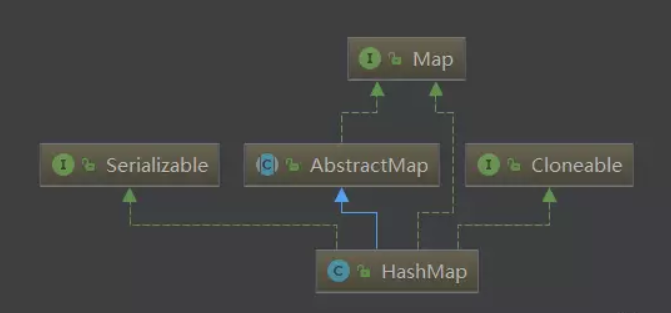
**HashMap和Hashtable相比**



Hashmap的key/value允许为null

都无序（hash本无序）

HashMap



1. HashMap为什么要选择Entry数组来存放key-value?



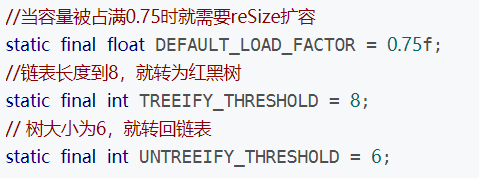
因为Entry实现的Map.Entry接口里面定义了getKey(),getValue()，setKey(),setValue()等方法方便操作

1. 并不是桶子上有8位元素的时候它就能变成红黑树，它得同时满足我们的散列表容量大于64才行的~



1. 成员属性

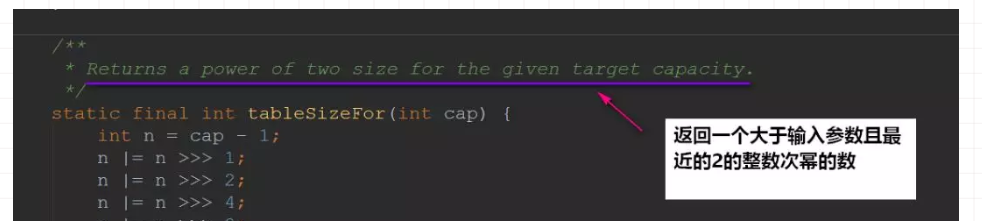
* size表示HashMap中存放KV的数量
* threshold临界值：capacity \* loadFactory，当size > threshold时，会扩容
* 装载因子 ：0.75f
* capacity：HashMap容量，默认大小16



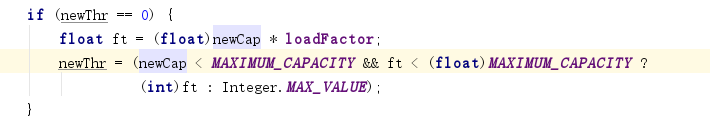
1. 常用方法

* **tableSizeFor()** <https://www.cnblogs.com/yang4869/p/11078816.html>





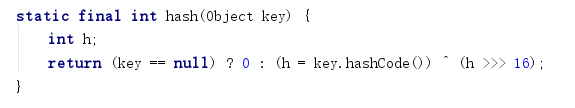
这里仅仅是一个初始化，当创建哈希表的时候，它会重新赋值



* **Hash()**

HashMap并不是直接拿key的哈希值来用的，它会把key的哈希值与其高16位异或

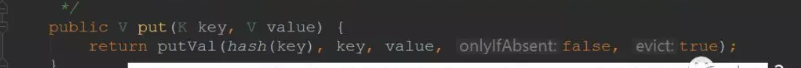
异或^操作，使得将元素放入哈希表的时候增加一定的随机性相同为0，不同为1

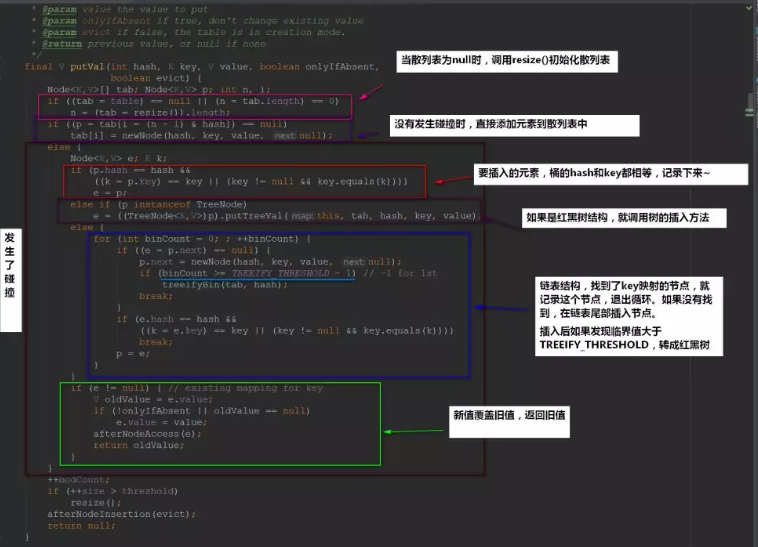


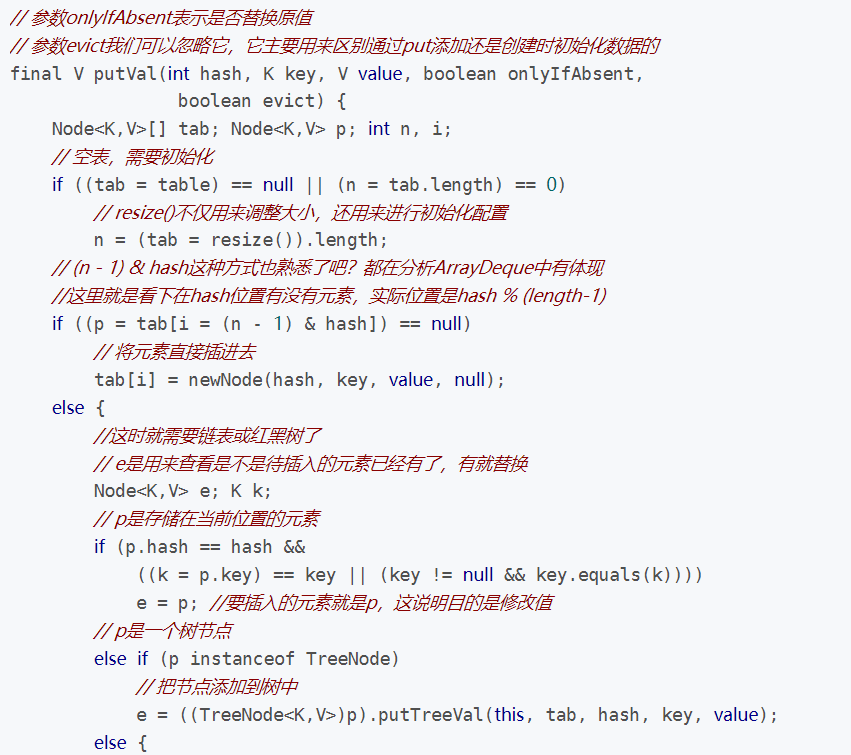
**碰撞：**

n为表长度

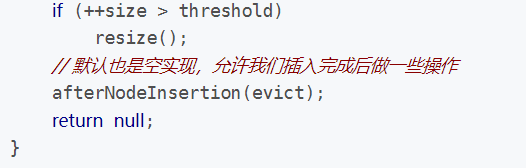
* **Put()**



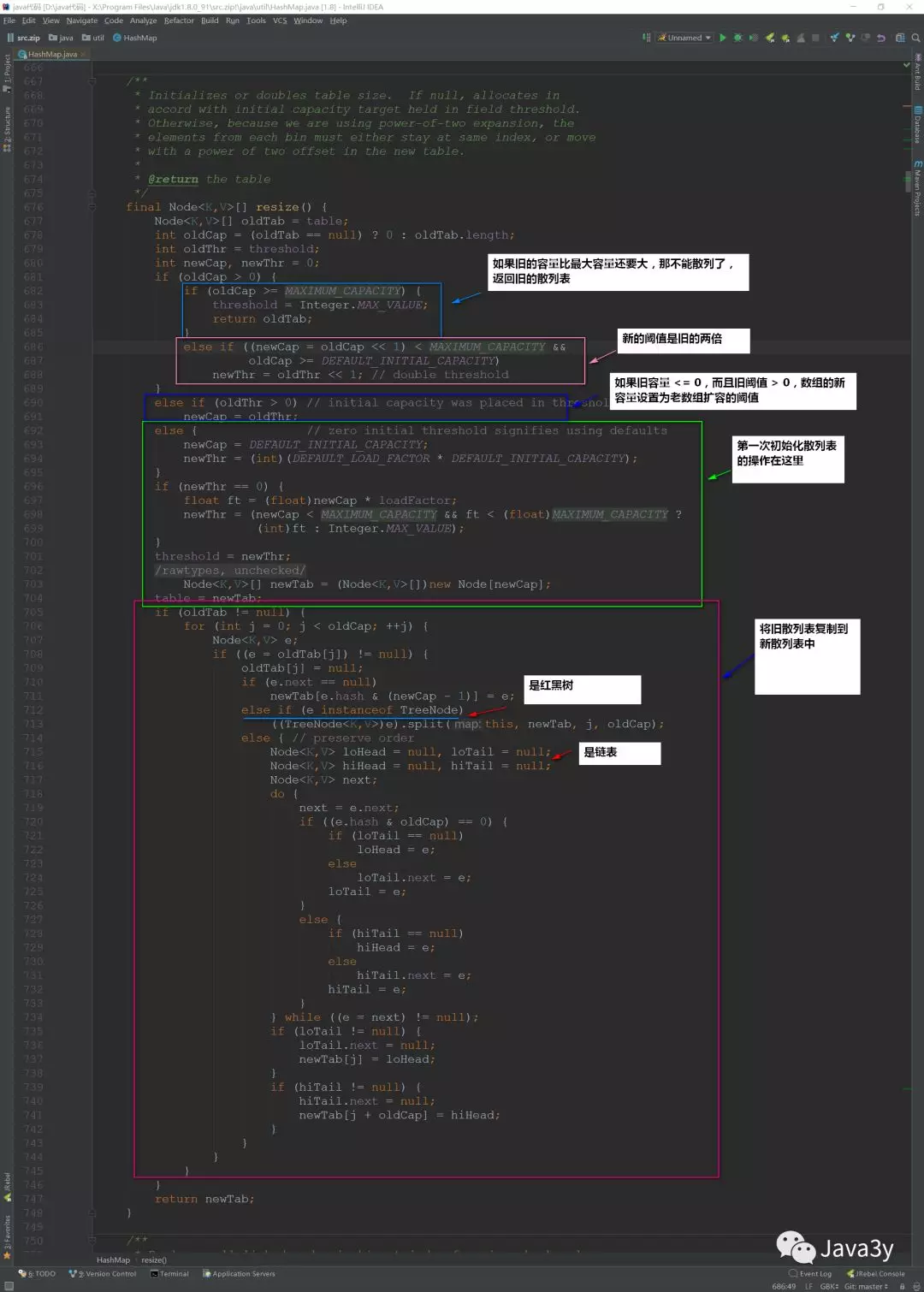




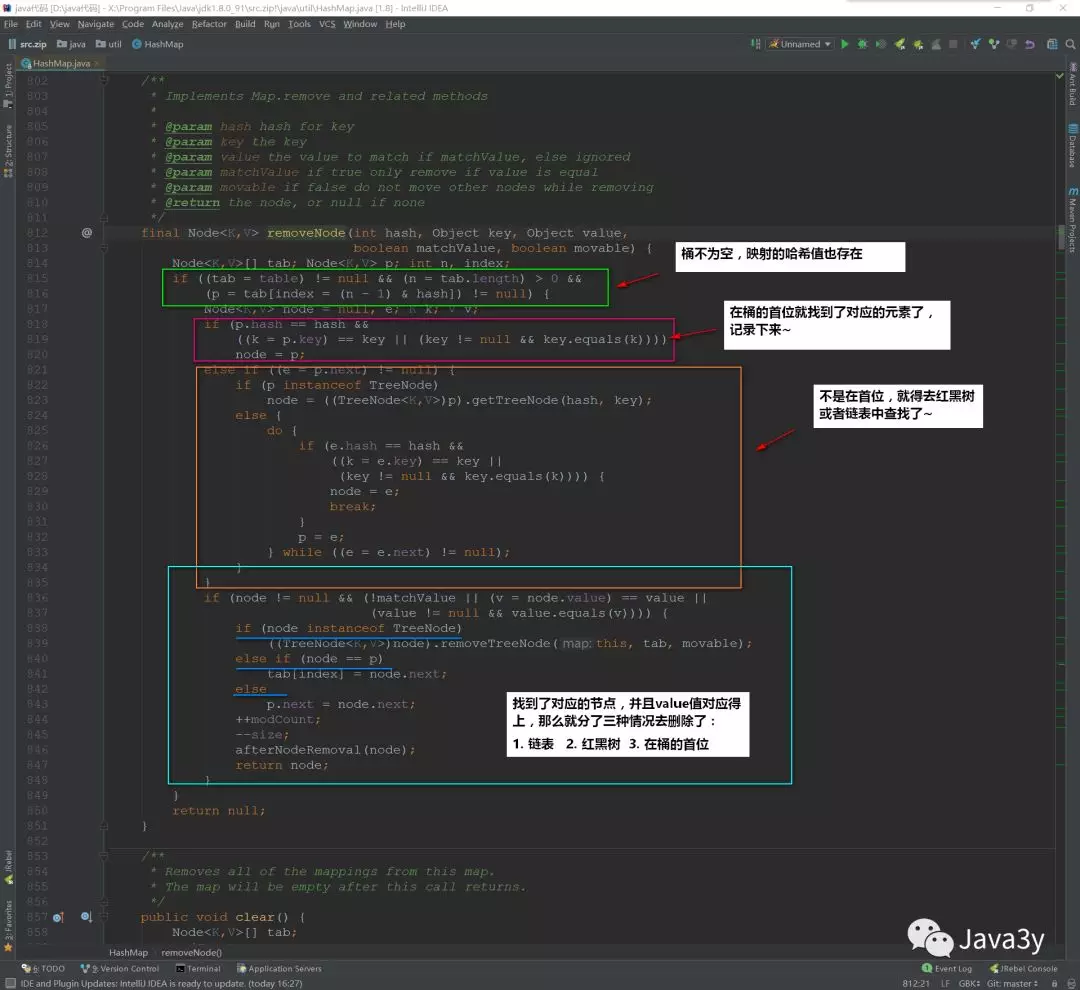




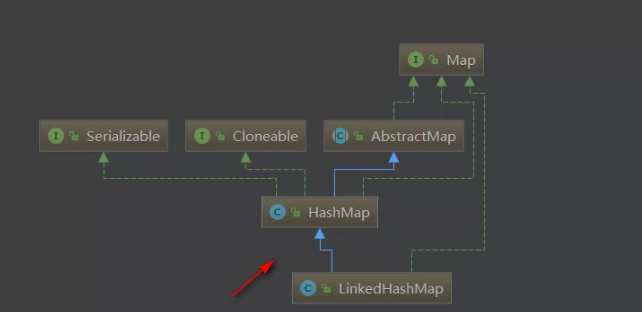
* **resize()**



* **Remove()**



# LinkedHashMap



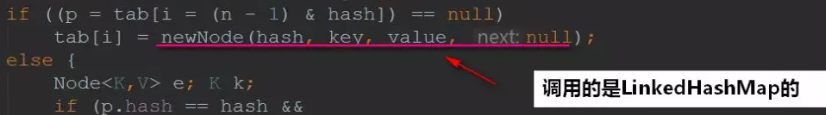
1. 底层是哈希表和双向链表



**哈希表(Hash Table)，是一种「数据结构」**

1. Put方法与父类HashMap一样只是重写了newNode()方法

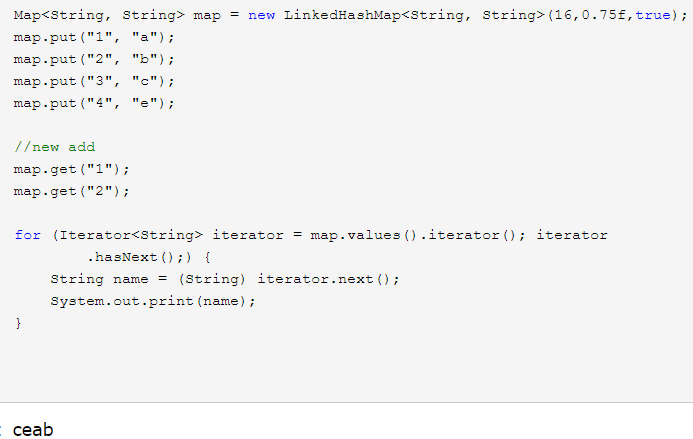




1. 两种遍历方式
2. accessOrder

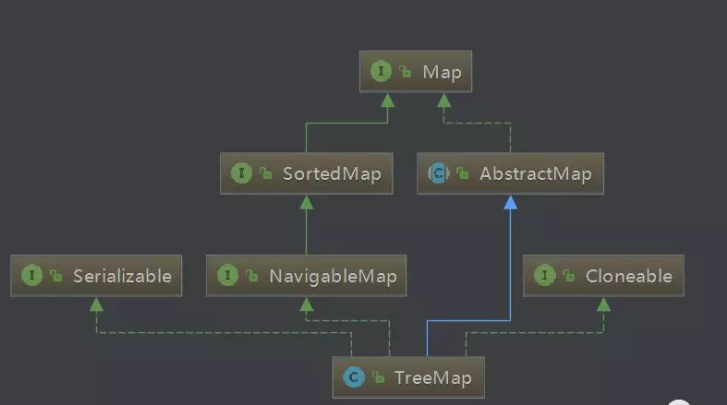
false： 基于插入顺序

true： 基于访问顺序LRU(最近最少使用)算法



get一个元素后，这个元素被加到最后(使用了LRU 最近最少被使用的调度算法)

# TreeMap



1. TreeMap实现了NavigableMap接口，而NavigableMap接口继承着SortedMap接口，致使我们的TreeMap是有序的！

Comparator和Comparable出现的频率是很高的，因为TreeMap实现有序要么就是外界传递进来Comparator对象，要么就使用默认key的Comparable接口(实现自然排序)

1. TreeMap底层是红黑树
2. 非同步, key不能为null
3. 如果在构造方法中传递了Comparator对象，那么就会以Comparator对象的方法进行比较。否则，则使用Comparable的compareTo(T o)方法来比较。