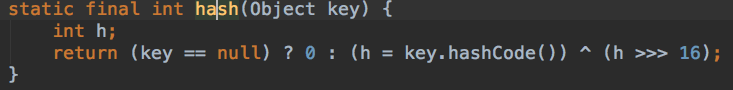
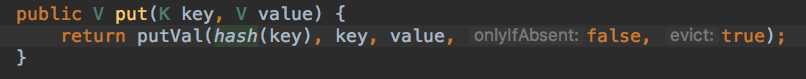
HashMap源码分析

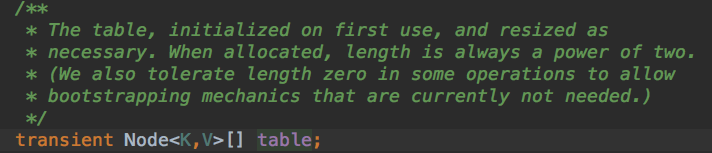
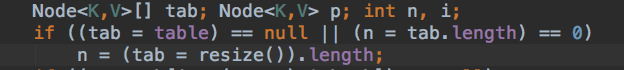
一、总述

HashMap是工作中经常使用的数据结构，也是面试中经常问到的知识点，掌握HashMap的工作原理，帮助更好的使用HashMap，并在面试中获得更多机会，下面我们从源码层面分析HashMap的工作原理。

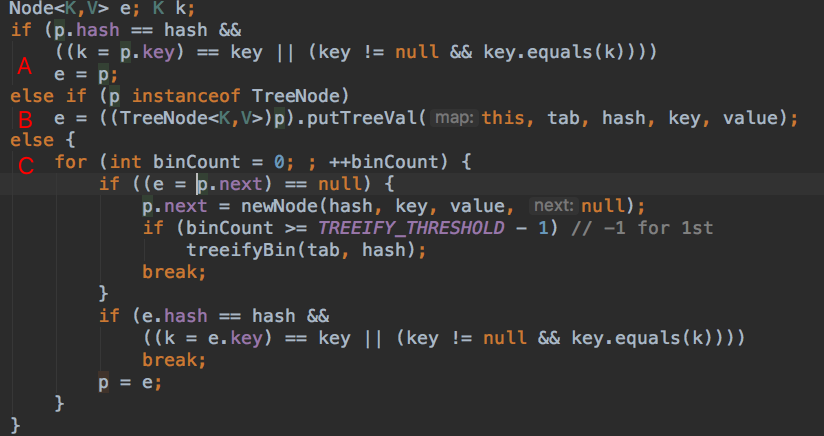
二、Put()

对单一功能的源码分析不仅会涉及到当前功能代码，还会辐射到相关的代码，如果能将一个主流程分析清楚，其他主流程也会更容易。

进入put方法，其中hash()调用了key本身的hashCode方法，产生int，并将hashCode的高16位和低16位相互异或。原因是哈希表长度为2的幂次，因此hashCode高位值在计算元素放置的位点（index calculating）时被忽略，只有低于或等于哈希表长度掩码（高位全0，低位全1）的位才生效，因此要让高位参与决策，又要权衡时间、效果，异或是较好的选择。

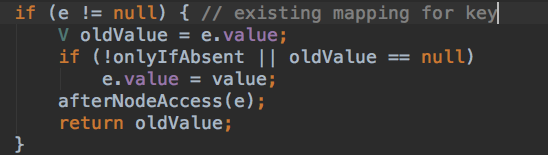
这里也可以看出，HashMap支持null作为key。

进入putVal方法，该方法存放KV，并返回被覆盖的value（如果key存在，否则返回null）。HashMap的本质是Node数组，Node中包含hash、key、value、next。本质上Node是链表元素。注释中写明，table在第一次使用时初始化，并在需要的时候扩容。另外不仅是HashMap中，JDK的很多代码中都会将成员变量先拷贝一份为局部变量，例如tab = table，原因是局部变量存放在栈中，成员变量存放在堆中，访问栈会更快。当初次访问或哈希表大小为0时进行扩容，即首次使用。扩容过程后续会分析【分支一】。

n - 1为哈希掩码，高位全0，低位全1，与操作，只保留hash的低位。如果tab的位点为空，则放入一个Node。否则p不等于null。

如果位点不是null，总会检查p，即头结点是否与插入的key相同，判断条件时hash相等且key相等。如果头结点不同会检查后续链表节点和树的子节点。注意此时并没有完成覆盖，而是将老节点暂存在e中。这就是代码A，接着看代码C，对于HashMap的源码分析，我们总是先读链表，后读红黑树。【分支二】

需要注意的是，key不同但hash相同的多个key，不会覆盖，而是保留。所以如果hashCode方法设计不够随机，更容易产生hash冲突。只有key相同，hash相同时才会覆盖。判断key相同的方法是引用的同一个对象或符合equals方法。

同样继续遍历链表，如果已经到达尾部，则将元素插入链尾。遍历过程中也会对比key是否相同，判断条件跟A相同。如果有相同的说明需要覆盖，同样暂存在e中。

当e不是null，即哈希表中已经存在key，则替换value，返回老值。接着递增modCount，并判断是否需要扩容。个人认为代码A没必要，可以合并到遍历链表中。接着回到【分支二】。

红黑树是一颗二叉搜索平衡树，查找时利用了二叉搜索树折半查找的特性，虽然这些Node hash冲突（hash可能不同，但位点相同），但hash值并不一定相同，首先可以根据hash值的大小判断位置，使用了二叉搜索树的特性（如果hash>Node.hash则向右子树寻找，如果hash<Node.hash则向左子树寻找），如果hash相同则会判断key是否相同，如果两者都相同即已经存在这个key。如果不相同，则会尝试调用compareTo方法，比较两者大小。前提是Key的类型实现了Compareable接口。如果没实现Compareable接口，或compareTo返回了0，即两个对象相等