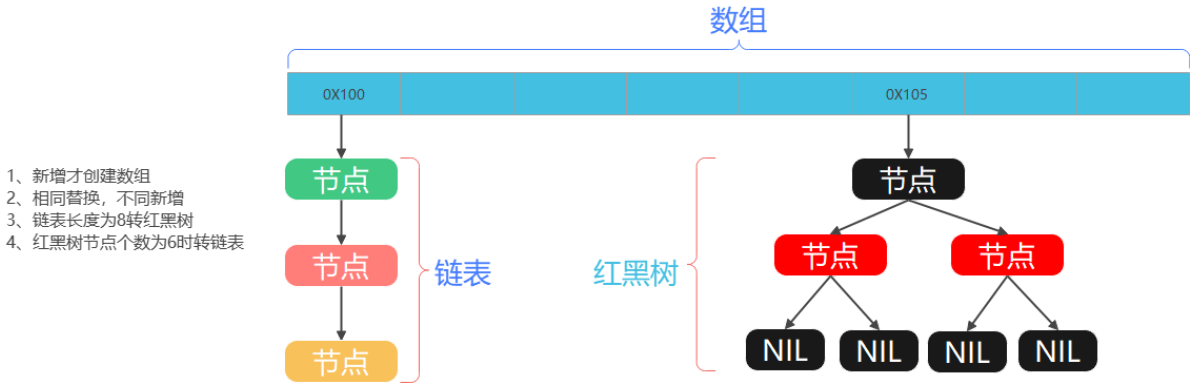
【第02话：怎么每次面试官都爱问HashMap底层原理呢】

HashMap底层原理(月薪1万的回答示范)



HashMap底层是数组+链表+红黑树。新增数据时创建数组，默认长度为16。当元素个数大于总容量的75%时，数组扩容2倍。最大为2的30次方。

每次新增时都会对Key做Hash运算，根据Hash值及数组容量确定节点在数组中放置位置。如果数组中这个位置没有内容则新建一个节点放在数组中。如果这个位置有内容了，会判断这个位置中是否已经存在这个值，如果存在会替换。如果不存在会新建一个节点，并把节点放这个位置对应的链表最后。

当链表长度大于等于8时，会判断数组长度是否大于等于64.如果数组长度小于64则会对数组进行扩容。如果数组长度大于等于64，则会把链表转换为红黑树。

当删除元素时，如果红黑树中元素个数小于等于6会把红黑树转换为链表。

以上就是Hashmap底层原理，月薪1万的回答示范。

HashMap底层原理(月薪1.5万的回答示范)

从Java8开始HashMap的底层是数组+链表+红黑树。而Java 7之前HashMap底层是数组+链表，没有把链表转换为红黑树这个规则。

当实例化HashMap时，在Java 7中会默认实例化一个长度为16的数组，并设置扩容因子为0.75。而从Java 8开始只是设置了扩容因子为0.75，变成了在第一次新增时创建默认长度为16的数组。

当数组需要扩容时，Java7中只会判断元素个数是否到达总容量的75%，达到后会对数组中元素扩容，并根据元素Hash值重新放置到新数组中。而从Java 8 开始数组扩容有两种情况，第一种和Java7相同，判断元素个数是否到达75%，开始扩容值。第二种是数组长度小于64，但链表长度已经到达8时进行扩容。

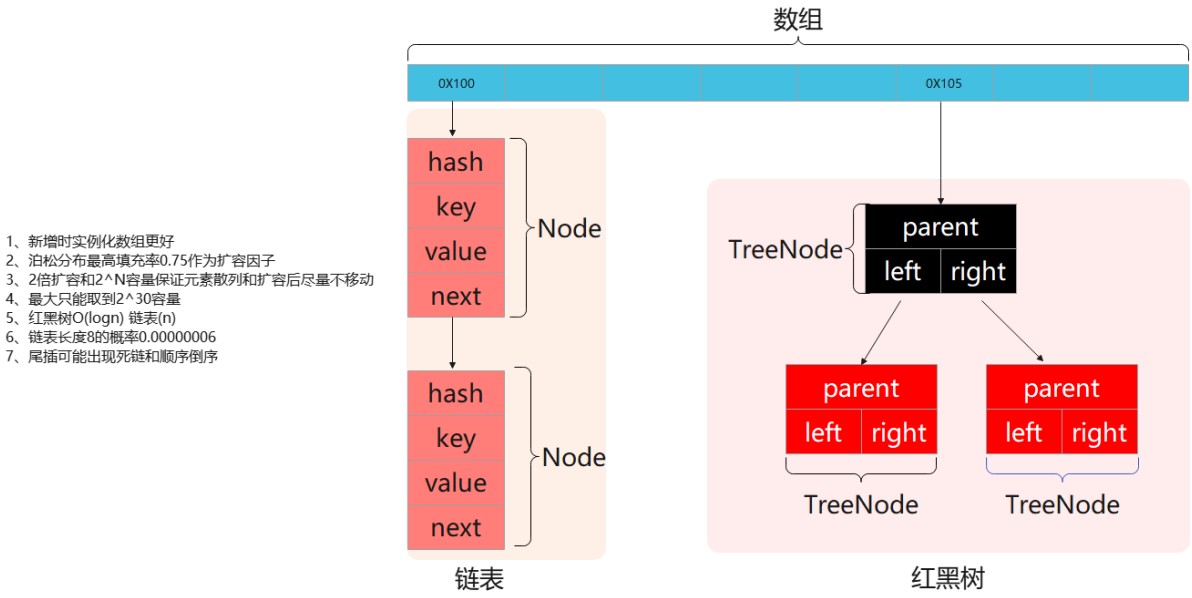
当新增元素时，如果元素已经存在会覆盖掉之前的内容。如果元素不存在，在Java 7 中会创建一个Entry对象，根据Key的Hash值，确定对象放置位置。当发生Hash碰撞后会形成链表，且使用头插法把对象插入到链表中。而从Java 8之后，元素类型变成Entry的子类Node类型对象。且不再使用头插法，而变成了尾插法。如果链表长度大于等于8，且数组长度大于等于64，会把链表转换为红黑树，来提升查询的性能。转化时元素类型变成了Node的子类TreeNode类型。这也是从Java 8开始HashMap底层最大的变化了。

当删除元素时，会对元素Key做Hash计算后定位到数组的某个位置，然后遍历这个位置对应的链表或红黑树，找到元素后进行删除。如果这个位置是红黑树，删除元素后，红黑树中元素个数小于等于6会把红黑树转换为链表。HashMap只有扩容，并没有缩容。

当查询元素时，会对Key做Hash运算，定位到数组中某个位置。然后遍历链表或红黑树，直到找到这个元素，找到后返回元素的地址。

以上就是HashMap底层原理，月薪1.5万的回答示范。

HashMap底层原理(月薪2万+的回答示范)



Java中HashMap是对数据结构中散列表的具体实现。使用链式地址法来解决Hash碰撞。

当实例化一个HashMap对象时，Java8之前是可以让PG(Programer)指定数组初始容量。如果没有指定初始容量则会同时创建一个长度为16的数组。从Java 8开始PG没有指定初始容量时，并不会创建数组。这是Oracle公司考虑到既然实例化HashMap时并没有添加元素，也就没有必要创建底层数组，什么时候新增什么时候创建更加合适。这也是SQO（Software Quality Optimization，软件质量最优化）非常重要的一个思想 。和Hibernate或MyBatis中的延迟加载机制有着异曲同工之妙。

HashMap扩容是根据总容量扩容因子0.75进行判断的，每次扩容2倍且容量保证是2的N次方，最大上限为2的30次方。扩容因子越大发生Hash碰撞概率越高，扩容因子越低发生碰撞的概率越小，但是太小的扩容因子会导致频繁扩容，也会影响性能。所以根据泊松分布计算出的最高填充率为0.75。而每次扩容2倍是且保证是2的N次方是为了保证扩容后元素尽量散列，避免不必要的Hash冲突。同时保证元素只有50%的概率改变位置，优化性能。也是因为这点，所以HashMap最大容量只能是2的30次方。毕竟数组最大长度为Integer.MAX_VALUE，即2的31次方-1，能保证最大的2的N次方只能是2的30次方了。

当新增元素时，元素类型为Node，里面存储key的hash、key值、value值、下一个节点地址。如果是红黑树，除了这四个值还有和它关联节点的值，parent、left、right等。从Java8开始最大的变化就是链表长度为8且数组长度为64时转换由链表转换为红黑树。这是因为红黑树查询、增加、删除的时间复杂度为O(logn),链表查询、删除时间复杂度O(n),插入为O(1)。添加元素的效率红黑树要低于链表。查询删除效率红黑树要高于链表。也是说红黑树时间复杂度时log(8)=3，而链表时n/2=4,这时有必要转换。

根据泊松分布计算，链表长度达到8的概率只有0.00000006，可以说是超级小概率事件了。比北京小汽车摇号概率0.0003还低。虽然低，但是也有可能发生，所以红黑树和链表转换只是在极端情况下更优方案，也不需要过分担心红黑树和链表转换影响性能的问题。

同时Java8中链表是尾插法，而Java 7是头插法。主要原因是Java 7中尾插法，在多线程下可能产生死链问题，且扩容后链表元素顺序倒序。而Java8的尾插法没有对应问题，扩容后顺序是不变的。

而删除时就是根据Key的Hash值查找到节点，进行删除即可。还是一个性能考虑问题，如果数组中红黑树中节点数量小于等于6转换为红黑树。之所以是6不是8，因为log(6)=2.6，6/2=3，性能相差不大了。可以转换回链表了。

查询时就是正常的根据Key的Hash值定位到数组后遍历链表或查找红黑树就可以了。

HashMap底层原理(月薪2万+的回答示范)