链滴

学习笔记 |HashMap 和 ConcurrentHashMap

作者：ellenbboe

原文链接：https://ld246.com/article/1563872298013

来源网站：链滴

<h2 id="HashMap">HashMap</h2>
不支持并发操作
HashMap 里面是数组,数组中的每一个元素都是单向链表
<h3 id="特征值">特征值</h3>
capacity:当前数组容量,始终保持 2^n,可以扩容,扩容后数组大小为当前的 2 倍
loadFactor:负载因子，默认为 0.75,负载因子越接近 1,数组就越密,查找效率低,小就越疏,数组的利用率就低
threshold:扩容的阈值,等于 capacity*loadFactor
在计算 key 的哈希值的之后让 key 的 hash 值与高 16 位进行异或运算,减少了 hash 冲突
扩容:当前的 size 已经达到了阈值，并且要插入的数组位置上已经有元素,将容扩大为原来的两倍,并将原来的数组迁移到现有数组中,迁移过程中，会将原来 table[i] 中的链表的所节点，分拆到新的数组的 newTable[i] 和 newTable[i + oldLength] 位置上。如原来数组长度是 16 那么扩容后，原来 table[0] 处的链表中的所有元素会被分配到新数组中 newTable[0] 和 newTable[16 这两个位置
默认容量:16
<h3 id="Java7中的HashMap">Java7 中的 HashMap</h3>
<h4 id="Put方法">Put 方法</h4>

放入第一个元素时初始化数组大小
根据 key 存放 value

key 为 null,这个 value 放到 table[0] 中
key 非 null,得到 hash 的 key 值,通过 indexFor 方法计算出数组的下标,遍历一下对应下标的链表假如有重复(key 的判重),直接覆盖并返回旧值,否则就将 value 添加到链表上

<h4 id="初始化数据">初始化数据</h4>
大小为大于输入数字的最小 2 的 n 次方,不输的话默认是 16
阈值等于 capacity*loadFactor
<h4 id="计算对应的数组下标">计算对应的数组下标</h4>
取 key 的 hash 值的低 n 位(n 根据数组大小确定),假如数组大小为 32.那么取 hash 的低 5 位
<h4 id="添加节点到链表">添加节点到链表</h4>
将新值放入到链表的表头(注意是表头)
假如当前的容量大于阈值并且要存放的数组位置有值了,那么就要扩容,扩容后重新计算存放的位置
<h4 id="Get方法">Get 方法</h4>

计算 hash 值:得到 key 的 hash 值
获取数组的下标:hash 的低 n 位
遍历下标对应的链表

<h3 id="Java8中的HashMap">Java8 中的 HashMap</h3>
采用数组 + 链表 + 红黑树
链表中元素有 8 个并且散列表容量大于 64 时会自动转化为红黑树
根据第一个元素是 Node 或者是 TreeNode 来确定是链表还是红黑树
<h4 id="Put方法-">Put 方法</h4>

放入第一个元素时初始化数组大小(初始化大小到默认的 16 或者自定义大小)
通过 Hash 找到数组的位置

假如该位置没有值,就放入其中

已经有元素的话,判断第一个元素的 key 和插入值的 key 是否相等,假如相等就覆盖并返回旧值</l
>
判断是红黑树还是链表,红黑树的话就调用红黑树的插入方法,链表的话插入到链表的最后面(假如
入的值是第八个,链表会调用 treeifyBin 方法变化成红黑树)
假如插入后超过阈值就进行扩容(resize)

扩容的时候假如是链表,会被拆分两条链表
<h4 id="Get方法-">Get 方法</h4>

计算 key 的 Hash 值,根据 hash 值找到数组的下标
假如是链表,遍历链表,假如是红黑树就遍历红黑树

<h2 id="ConcurrentHashMap">ConcurrentHashMap</h2>
并发安全
<h3 id="特征值-">特征值</h3>
concurrencyLevel:并发数,Segement 默认是 16,表示最多可以同时支持 16
个线程并发写
initialCapacity:整个 ConcurrentHashMap 的容量,Segment 数组不可以扩容<
r>
loadFactor:为每个 Segement 内部使用
segementshift:2 的 sshift 次方等于 ssize，segmentShift=32-sshif
;segementMask=ssize-1(确保低位都是 1 来使得获得的 hashcode 均匀分布
不允许 key 和 value 是 null
<h3 id="Java7中的ConcurrentHashMap">Java7 中的 ConcurrentHashMap</h3>
ConcurrentHashMap 是一个 Segement 数组,可以对单个 Segement 加锁
Segment 数组不可以扩容,默认每个 Segement 容量大小为 2
初始化完成后得到一个 Segement 数组,只初始化了一个 Segement[0]
Segement 内部是数组 HashEntry+ 链表
每一个 segement 对象都有一个 count 数(volicate)来统计内部的 entry 数量,加入 count 大于阈值
扩容成原来的两倍;统计 size 的时候前两次不加锁(如果出来的数据变了的话),在将 remove 和 put 方
锁住进行统计
<h4 id="Put方法--">Put 方法</h4>

计算 key 的 hash 值
根据 Hash 值找到 Segement 位置下标 j(计算方法:(hash >>> segmentShift) &
egmentMask)
对 Segement[j]初始化,根据当前的 Segement[0]来初始化 Segement[j],并发操作使用 CAS 进
控制
将新值插入到对应的 Segement 槽中
0. 获取 Segement 的独占锁,循环调用 tryLock 获取

通过 key 的 Hash 值得到数组下标(table.length - 1) & hash
得到该位置的链表表头,遍历链表查看是否有重复值,有的话覆盖旧值并返回旧值
将改新值节点设置为表头,假如超过了该 Segement 阈值则需要扩容(Rehash),然后在进行插入操作
由于 entry 的 next 是 final,所以只能在表头插入元素
扩容是对 Segement 内的 HashEntry 数组进行扩容,扩容成两倍,扩容后,将原数组位置 i 处的链表<str
ng>拆分到 新数组位置 i 和 i+oldCap 两个位置

<h4 id="Get方法--">Get 方法</h4>
get 方法不用加锁,所用的变量都是 volilate 修饰,volatile 可以保证内存可见性，所以不会读取到

期数据。

计算 Hash 值,找到 Segement(槽)
根据 Hash 值在 Segement 内部数组中找到下标
遍历链表

remove()元素后该元素后面的元素顺序不变,前面的会变成倒序(是从表头插入的)
<h3 id="Java8中的ConcurrentHashMap">Java8 中的 ConcurrentHashMap</h3>
加入了红黑树
构造函数初始化(设置 sizeCtl)
通过提供的初始化容量大小设置 sizeCtl
sizeCtl:(1.5 * initialCapacity + 1),然后向上取最近的 2 的 n 次方
sizeCtl 数值:-1(初始化) ,-n(有 n-1 个线程正在扩容),正数(下一次要扩容的大小)或 0(没有初始化)
<h4 id="初始化数组">初始化数组</h4>
通过 CAS 将 sizeCtl 设置成-1,通过判断 sizeCtl 来判断是否被初始化,并设置默认值
<h4 id="链表转红黑树">链表转红黑树</h4>
对链表加锁,遍历链表,建立红黑树,并设置到对应的位置.不一定转化成红黑树,有可能就是数组扩容
/p>
<h4 id="数据迁移">数据迁移</h4>
将旧的数组大小分配给多个线程
数组第一个元素的 hash 值假如是 MOVED(-1),表明正在迁移,迁移要判断是链表还是红黑树
迁移可以多线程并发,线程加入使 sizectl 自增,线程退出后 sizectl 自减
数据迁移的时候要得到当前数组位置元素的 Sychronized 锁
<h4 id="Put方法---">Put 方法</h4>

假如数组表为 null,则初始化数组
通过 key 的 Hash 值得到数组的下标位置

数组头元素位置是空的.那就使用 CAS 将新值写入
数组头元素位置非空,判断是否是链表,如果是链表,就遍历链表,查找重复值,有重复值就覆盖并返回值,没有重复值就将新的节点放到最后;如果是红黑树,就按红黑树的插入方法插入

插入后判断链表是否大于 8 并且数组长度是否大于 64,只有前面两项都满足时才会转化成红黑树,则就是扩容数组

<h4 id="Get方法---">Get 方法</h4>

计算 hash 值
通过 Hash 找到数组的下标位置
在链表或者红黑树中查找数据
