

深入分析ConcurrentHashMap1.8的扩容实现



占小狼

已关注

赞赏支持

深入分析ConcurrentHashMap1.8的扩容实现



占小狼 已关注

2 2017.03.09 17:00:49 字数 1,336 阅读 28,991

简书 占小狼 转载请注明原创出处，谢谢！

此谓知本，此谓知之至也 《礼记·大学》

- 1、深入浅出ConcurrentHashMap (1.8)
- 2、谈谈ConcurrentHashMap1.7和1.8的不同实现
- 3、ConcurrentHashMap的红黑树实现分析

ConcurrentHashMap相关的文章写了不少，有个遗留问题一直没有分析，也被好多人请教过，被搁置在一旁，即如何在并发的情况下实现数组的扩容。

什么情况会触发扩容

当往hashMap中成功插入一个key/value节点时，有可能触发扩容动作：

- 1、如果新增节点之后，所在链表的元素个数达到了阈值 8，则会调用 `treeifyBin` 方法把链表转换成红黑树，不过在结构转换之前，会对数组长度进行判断，实现如下：

```
private final void treeifyBin(Node<K,V>[] tab, int index) {
    Node<K,V> b; int n, sc;
    if (tab != null) {
        if ((n = tab.length) < MIN_TREEIFY_CAPACITY)
            tryPresize(n << 1);
        else if ((b = tabAt(tab, index)) != null && b.hash >= 0) {
            synchronized (b) {
                if (tabAt(tab, index) == b) {
                    TreeNode<K,V> hd = null, tl = null;
                    for (Node<K,V> e = b; e != null; e = e.next) {
                        TreeNode<K,V> p =
                            new TreeNode<K,V>(e.hash, e.key, e.val,
                                                null, null);
                        if ((p.prev = tl) == null)
                            hd = p;
                        else
                            tl.next = p;
                        tl = p;
                    }
                    setTabAt(tab, index, new TreeBin<K,V>(hd));
                }
            }
        }
    }
}
```

如果数组长度 n 小于阈值 `MIN_TREEIFY_CAPACITY`，默认是64，则会调用 `tryPresize` 方法把数组长度扩大到原来的两倍，并触发 `transfer` 方法，重新调整节点的位置。

推荐阅读

ConcurrentHashMap解析 (JDK1.8)
阅读 493

纯干货！JAVA细节标注+源码分析
阅读 3,035

如何设计并实现一个线程安全的 Map
? (上篇)
阅读 7,887

玛丽安的梦 (一)
阅读 277

有求乃苦，无求乃乐
阅读 112

写下你的评论...

评论65 赞103 ...

深入分析ConcurrentHashMap1.8的扩容实现

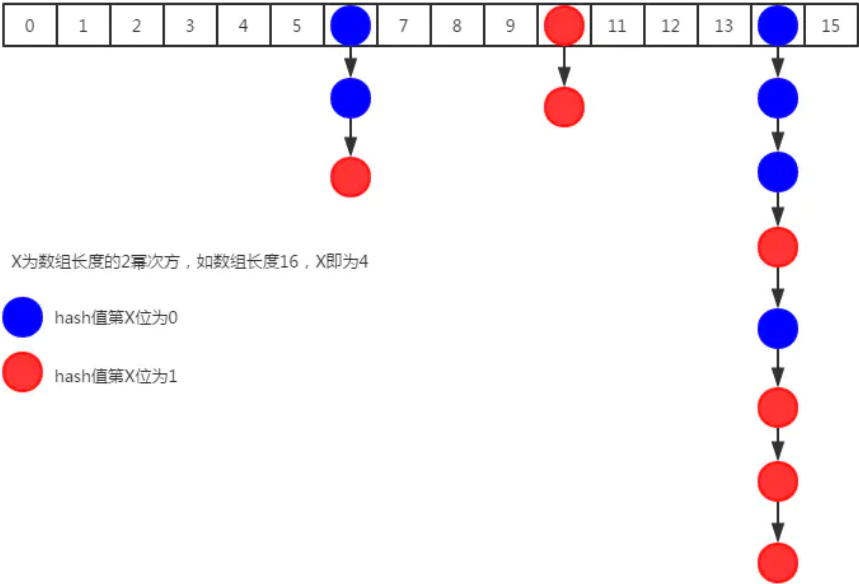
```
Node<K,V>[] nt;  
if ((sc >>> RESIZE_STAMP_SHIFT) != rs || sc == rs + 1 ||  
    sc == rs + MAX_RESIZERS || (nt = nextTable) == null ||  
    transferIndex <= 0)
```

2、新增节点之后，会调用 `addCount` 方法记录元素个数，并检查是否需要进行扩容，当数组元素个数达到阈值时，会触发 `transfer` 方法，重新调整节点的位置。

```
if (check >= 0) {  
    Node<K,V>[] tab, nt; int n, sc;  
    while (s >= (long)(sc = sizeCtl) && (tab = table) != null &&  
        (n = tab.length) < MAXIMUM_CAPACITY) {  
        int rs = resizeStamp(n);  
        if (sc < 0) {  
            if ((sc >>> RESIZE_STAMP_SHIFT) != rs || sc == rs + 1 ||  
                sc == rs + MAX_RESIZERS || (nt = nextTable) == null ||  
                transferIndex <= 0)  
                break;  
            if (U.compareAndSwapInt(this, SIZECTL, sc, sc + 1))  
                transfer(tab, nt);  
        }  
        else if (U.compareAndSwapInt(this, SIZECTL, sc,  
                                     (rs << RESIZE_STAMP_SHIFT) + 2))  
            transfer(tab, null);  
        s = sumCount();  
    }  
}
```

transfer实现

`transfer` 方法实现了在并发的情况下，高效的从原始数组往新数组中移动元素，假设扩容之前节点的分布如下，这里区分蓝色节点和红色节点，是为了后续更好的分析：



在上图中，第14个槽位插入新节点之后，链表元素个数已经达到了8，且数组长度为16，优先通过扩容来缓解链表过长的问题，实现如下：

- 1、根据当前数组长度n，新建一个两倍长度的数组 `nextTable` ；

推荐阅读

ConcurrentHashMap解析 (JDK1.8)
阅读 493

纯干货！ JAVA细节标注+源码分析
阅读 3,035

如何设计并实现一个线程安全的 Map
? (上篇)
阅读 7,887

玛丽安的梦 (一)
阅读 277

有求乃苦，无求乃乐
阅读 112



深入分析ConcurrentHashMap1.8的扩容实现



占小狼

已关注

赞赏支持

```
Node<K,V>[] nt = (Node<K,V>[])new Node<?,?>[n << 1];
```

2、初始化 `ForwardingNode` 节点，其中保存了新数组 `nextTable` 的引用，在处理完每个槽位的节点之后当做占位节点，表示该槽位已经处理过了；

```
int nextn = nextTab.length;
ForwardingNode<K,V> fwd = new ForwardingNode<K,V>(nextTab);
boolean advance = true;
boolean finishing = false; // to ensure sweep before committing nextTab
```

3、通过 `for` 自循环处理每个槽位中的链表元素，默认 `advance` 为真，通过CAS设置 `transferIndex` 属性值，并初始化 `i` 和 `bound` 值，`i` 指当前处理的槽位序号，`bound` 指需要处理的槽位边界，先处理槽位15的节点；

```
for (int i = 0, bound = 0;;) {
    Node<K,V> f; int fh;
    while (advance) {
        int nextIndex, nextBound;
        if (--i >= bound || finishing)
            advance = false;
        else if ((nextIndex = transferIndex) <= 0) {
            i = -1;
            advance = false;
        }
        else if (U.compareAndSwapInt
            (this, TRANSFERINDEX, nextIndex,
             nextBound = (nextIndex > stride ?
                          nextIndex - stride : 0))) {
            bound = nextBound;
            i = nextIndex - 1;
            advance = false;
        }
    }
}
```

4、在当前假设条件下，槽位15中没有节点，则通过CAS插入在第二步中初始化的 `ForwardingNode` 节点，用于告诉其它线程该槽位已经处理过了；

```
else if ((f = tabAt(tab, i)) == null)
    advance = casTabAt(tab, i, null, fwd);
```

5、如果槽位15已经被线程A处理了，那么线程B处理到这个节点时，取到该节点的hash值应该为 `MOVED`，值为 `-1`，则直接跳过，继续处理下一个槽位14的节点；

```
else if ((fh = f.hash) == MOVED)
    advance = true; // already processed
```

6、处理槽位14的节点，是一个链表结构，先定义两个变量节点 `ln` 和 `hn`，按我的理解应该是 `lowNode` 和 `highNode`，分别保存hash值的第X位为0和1的节点，具体实现如下：

推荐阅读

[ConcurrentHashMap解析 \(JDK1.8\)](#)

阅读 493

[纯干货！JAVA细节标注+源码分析](#)

阅读 3,035

[如何设计并实现一个线程安全的 Map ? \(上篇\)](#)

阅读 7,887

[玛丽安的梦 \(一\)](#)

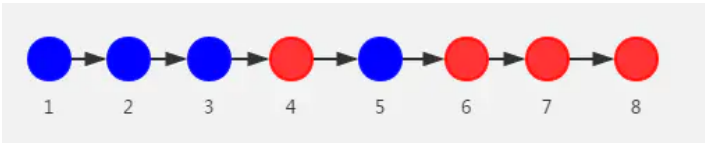
阅读 277

[有求乃苦，无求乃乐](#)

阅读 112

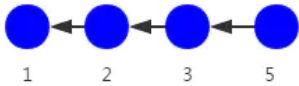
```
if (fh >= 0) {
    int runBit = fh & n;
    Node<K,V> lastRun = f;
    for (Node<K,V> p = f.next; p != null; p = p.next) {
        int b = p.hash & n;
        if (b != runBit) {
            runBit = b;
            lastRun = p;
        }
    }
    if (runBit == 0) {
        ln = lastRun;
        hn = null;
    }
}
```

使用 `fn&n` 可以快速把链表中的元素区分成两类，A类是hash值的第X位为0，B类是hash值的第X位为1，并通过 `lastRun` 记录最后需要处理的节点，A类和B类节点可以分散到新数组的槽位14和30中，在原数组的槽位14中，蓝色节点第X为0，红色节点第X为1，把链表拉平显示如下：

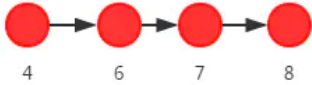


- 1、通过遍历链表，记录 `runBit` 和 `lastRun`，分别为1和节点6，所以设置 `hn` 为节点6，`ln` 为 `null`；
- 2、重新遍历链表，以 `lastRun` 节点为终止条件，根据第X位的值分别构造`ln`链表和`hn`链表：

`ln`链：和原来链表相比，顺序已经不一样了



`hn`链：



通过CAS把`ln`链表设置到新数组的*i*位置，`hn`链表设置到*i+n*的位置；

- 7、如果该槽位是红黑树结构，则构造树节点 `lo` 和 `hi`，遍历红黑树中的节点，同样根据 `hash&n` 算法，把节点分为两类，分别插入到 `lo` 和 `hi` 为头的链表中，根据 `lo` 和 `hi` 链表中的元素个数分别生成 `ln` 和 `hn` 节点，其中 `ln` 节点的生成逻辑如下：
 - (1) 如果 `lo` 链表的元素个数小于等于 `UNTREEIFY_THRESHOLD`，默认为6，则通过 `untreeify` 方法把树节点链表转化成普通节点链表；
 - (2) 否则判断 `hi` 链表中的元素个数是否等于0：如果等于0，表示 `lo` 链表中包含了所有原始节点，则设置原始红黑树给 `ln`，否则根据 `lo` 链表重新构造红黑树。

- 推荐阅读
- ConcurrentHashMap解析 (JDK1.8)
阅读 493
 - 纯干货！JAVA细节标注+源码分析
阅读 3,035
 - 如何设计并实现一个线程安全的 Map ? (上篇)
阅读 7,887
 - 玛丽安的梦（一）
阅读 277
 - 有求乃苦，无求乃乐
阅读 112

深入分析ConcurrentHashMap1.8的扩容实现

 占小狼

已关注

赞赏支持

```
TreeNode<K,V> hi = null, hiTail = null;
int lc = 0, hc = 0;
for (Node<K,V> e = t.first; e != null; e = e.next) {
    int h = e.hash;
    TreeNode<K,V> p = new TreeNode<K,V>
        (h, e.key, e.val, null, null);
    if ((h & n) == 0) {
        if ((p.prev = loTail) == null)
            lo = p;
        else
            loTail.next = p;
        loTail = p;
        ++lc;
    }
    else {
        if ((p.prev = hiTail) == null)
            hi = p;
        else
            hiTail.next = p;
        hiTail = p;
        ++hc;
    }
}
ln = (lc <= UNTREESIFY_THRESHOLD) ? untreefify(lo) :
```

最后，同样的通过CAS把 `ln` 设置到新数组的 `i` 位置，`hn` 设置到 `i+n` 位置。

 103人点赞 >






 java进阶干货




"小礼物走一走，来简书关注我"

赞赏支持

 共1人赞赏

 占小狼  如果读完觉得有收获的话，欢迎关注我的公众号：占小狼的博客 h...
总资产246 (约21.99元) 共写了17.1W字 获得17,539个赞 共25,997个粉丝

已关注




写下你的评论...



全部评论 65

只看作者

按时间倒序 按时间正序

 一只疯狂的蜗牛
33楼 01.14 09:43

对于ln和hn感觉大大说的不是很清楚，我根据自己的理解补充一下：
根据index的算法 $hash \& 2^n - 1$ 得出扩容两倍的index应该为 $hash \& (2^{n+1} - 1)$ 推导出
 $hash \& (2^{n-1} + 2^n)$ 等于 $hash \& (2^{n-1}) + hash \& 2^n$ 前者等于扩容前的index，后者由于高位是1所以其余全是0，所以结果只有0和 2^n 这两中可能。而 2^n 为链表原来的长度。所以在扩容复制的时候会将节点分为0和非0两种情况

 赞  回复



深入分析ConcurrentHashMap1.8的扩容实现



占小狼

已关注

赞赏支持

赞 回复



lzh_7dee

31楼 2020.03.27 10:38

transfer实现的6开始看不懂

赞 回复



九点半的马拉

30楼 2020.02.22 10:24

请问，在get()方法中，如果节点是红黑树怎么查啊，没看到这部分的代码

赞 回复



feifeidream

29楼 2019.07.18 00:14

感谢博主分享分析过程，明白了许多。

赞 回复



ZOKE

28楼 2019.05.08 02:48

博主，如果迁移中的 hash 桶遇到了 get 查询操作会怎么样？

赞 回复



15195819097

2019.05.09 21:58

源码里写的很清楚，在源码943行，如果能查就正常查，如果发现要查的节点的hash=-1，说明这个节点已经不在oldTable，要去nextTable查询，因此调用Node的find方法，在nextTable中查询。这也是为什么ForwardingNode的next指向了nextTable，因为便于查询。find方法在652行。

回复



豌豆逸之

2019.05.20 14:59

@ZOKE 迁移完毕后Node会被设置成ForwardingNode节点，如果另外一个线程执行的put或remove等写操作，访问到了这个ForwardingNode节点，那么就会先帮其扩容。详情参考putVal方法。

回复



ZOKE

2019.05.20 22:17

@豌豆逸之 恩，我说的是 ConcurrentHashMap 数组上的某一个 hash 桶正在迁移的过程中遇到 get 操作的情形，也就是在拼接 ln 和 hn 链的这个阶段，某个 hash 桶迁移完成后放置 fwd 节点以后的访问这个看代码很容易就能看出来。

回复

添加新评论



蹩脚的小三

27楼 2019.05.01 17:42

狼狗还是超厉害！

赞 回复

推荐阅读

ConcurrentHashMap解析 (JDK1.8)

阅读 493

纯干货！JAVA细节标注+源码分析

阅读 3,035

如何设计并实现一个线程安全的 Map ? (上篇)

阅读 7,887

玛丽安的梦 (一)

阅读 277

有求乃苦，无求乃乐

阅读 112

深入分析ConcurrentHashMap1.8的扩容实现

 占小狼

已关注

赞赏支持

👍 赞 💬 回复



NestleCaau
25楼 2018.07.04 10:56

里面有错
"通过CAS把ln链表设置到新数组的i位置， hn链表设置到i+n的位置；"
这里并没有使用CAS,源码是使用了putObjectVolatile,意思是直接把数据写回主内存

👍 赞 💬 回复



山川04
24楼 2018.05.15 12:01

5、如果槽位15已经被线程A处理了，那么线程B处理到这个节点时，取到该节点的hash值应该为MOVED，值为-1，则直接跳过，继续处理下一个槽位14的节点；
我想问一下，线程B在分配迁移槽的时候并不会分配之前分配过的啊，通过了transferindex，那么这边理论上不会出现某个线程检测到自己要处理的范围有已经被其他线程处理过的啊？不知道你知道这个条件在什么时候有用？

👍 赞 💬 回复



狂奔的龟
2018.07.04 16:54

你好，我一开始也有跟你一样的疑问，确实扩容时每个线程处理的范围不会重叠，其实这个MOVED标志是给在扩容的同时准备put的其他线程用的。put的时候如果遇到MOVED,说明正在扩容并且当前的ENTRY已经被处理过了，所以put线程加入一起扩容。如果没有遇到MOVED，说明当前ENTRY没有被处理过，即使正在扩容也不管，直接锁住当前ENTRY,先put完再判断当前是否在扩容，是的话put线程也会加入一起扩容。所以1.8中扩容和PUT操作是可以并发执行的，Doug Lea大神真是把CAS玩的出神入化。

💬 回复



Zoke
2019.05.06 22:34

@狂奔的龟 请教一个问题，如果是在put操作时锁住了某个hash桶，而正好这时候扩容线程正好迁移到该hash桶，这个时候会阻塞还是会跳过？ 如果会跳过的话直到最后一条线程的善后检查时依旧没put完的话会发生什么情况？


💬 回复


✎ 添加新评论


- 1
- 2
- 3
- 下一页


被以下专题收入，发现更多相似内容


+ 收入我的专题


 集合框架


 Java

 面试专题

 jdk1.8 ...

 JAVA

 java

 concurr... 展开更多

推荐阅读

更多精彩内容>



深入分析ConcurrentHashMap1.8的扩容实现

721 阅读 1,588 评论 0 赞 13

程序员必须掌握：数据结构与算法基础总览——数据结构篇

前言 对于绝大多数程序员来说，数据结构与算法绝对是一门非常重要但又非常难以掌握的学科。最近自己系统学习了一套数据结...

码农奋斗之路 阅读 721 评论 0 赞 13

ConcurrentHashMap的扩容

一、触发扩容的条件 addCount(put、remove、clear、computeIfAbsent、compu...

HannahLi_9f1c 阅读 124 评论 0 赞 0

并发编程之ConcurrentHashMap源码解读-1.8

上一篇文章并发编程之synchronized的前世今生
[https://www.jianshu.com/p/849...

默写流年 阅读 660 评论 2 赞 4

面试官你能不能别问我 HashMap 了？

如果你是个 Java 程序员，那一定对 HashMap 不陌生，巧的是只要你去面试，大... 概率都会被问到 HashMa...

热衷技术的Java程序员 阅读 3,277 评论 6 赞 34

推荐阅读

ConcurrentHashMap解析 (JDK1.8)

阅读 493

纯干货！ JAVA细节标注+源码分析

阅读 3,035

如何设计并实现一个线程安全的 Map ? (上篇)

阅读 7,887

玛丽安的梦 (一)

阅读 277

有求乃苦，无求乃乐

阅读 112

写下你的评论...

评论65 赞103

https://www.jianshu.com/p/f6730d5784ad

8/8