

并发编程——ConcurrentHashMap#transfer() 扩容逐行分析



莫那一鲁道 (/u/434239413000) + 关注

 0.6

2018.05.16 17:21*

字数 1666

阅读 2202

评论 3

喜欢 9

(/u/434239413000)

前言

ConcurrentHashMap 是并发中的重中之重，也是最常用的数据结构，之前的文章中，我们介绍了 putVal 方法。并发编程之 ConcurrentHashMap (JDK 1.8) putVal 源码分析 (<https://www.jianshu.com/p/77fda250bddf>)。其中分析了 initTable 方法和 putVal 方法，但也留下了一句话：

这篇文章仅仅是 ConcurrentHashMap 的开头，关于 ConcurrentHashMap 里面的精华太多，值得我们好好学习。

说道精华，他的扩容方法绝对是精华，要知道，ConcurrentHashMap 扩容是高度并发的。

今天来逐行分析源码。

先说结论

首先说结论。源码加注释我会放在后面。该方法的执行逻辑如下：

1. 通过计算 CPU 核心数和 Map 数组的长度得到每个线程 (CPU) 要帮助处理多少个桶，并且这里每个线程处理都是平均的。默认每个线程处理 16 个桶。因此，如果长度是 16 的时候，扩容的时候只会有一个线程扩容。
2. 初始化临时变量 nextTable。将其在原有基础上扩容两倍。
3. 死循环开始转移。多线程并发转移就是在这个死循环中，根据一个 finishing 变量来判断，该变量为 true 表示扩容结束，否则继续扩容。

3.1 进入一个 while 循环，分配数组中一个桶的区间给线程，默认是 16。从大到小进行分配。当拿到分配值后，进行 i-- 递减。这个 i 就是数组下标。（其中有一个 bound 参数，这个参数指的是该线程此次可以处理的区间的最小下标，超过这个下标，就需要重新领取区间或者结束扩容，还有一个 advance 参数，该参数指的是是否继续递减转移下一个桶，如果为 true，表示可以继续向后推进，反之，说明还没有处理好当前桶，不能推进）

3.2 出 while 循环，进 if 判断，判断扩容是否结束，如果扩容结束，清空临时变量，更新 table 变量，更新扩容阈值。如果没完成，但已经无法领取区间（没了），该线程退出该方法，并将 sizeCtl 减一，表示扩容的线程少一个了。如果减完这个数以后，sizeCtl 回归了初始状态，表示没有线程再扩容了，该方法所有的线程扩容结束了。（这里主要是判断扩容任务是否结束，如果结束了就让线程退出该方法，并更新相关变量）。然后检查所有的桶，防止遗漏。

3.3 如果没有完成任务，且 i 对应的槽位是空，尝试 CAS 插入占位符，让 putVal 方法的线程感知。

(/apps/redirect?utm_source=side-banner-click)

(<https://log-yex.youdao.com/ct?slot=30edd91dd86378ef7-487f-ad90-2c701414a735&yexigAwM0ITFbTaM36a1VNtTUBPvnBppYIVCa9FhKGM1CxoZk1hlclick.youdao.com%2l8ef7-487f-ad90-2c701414a735&iid=9>)

- 3.4 如果 i 对应的槽位不是空，且有了占位符，那么该线程跳过这个槽位，处理下一个槽位。
- 3.5 如果以上都不是，说明这个槽位有一个实际的值。开始同步处理这个桶。
- 3.6 到这里，都还没有对桶内数据进行转移，只是计算了下标和处理区间，然后一些完成状态判断。同时，如果对应下标内没有数据或已经被占位了，就跳过了。
4. 处理每个桶的行为都是同步的。防止 putVal 的时候向链表插入数据。
- 4.1 如果这个桶是链表，那么就将这个链表根据 length 取于拆成两份，取于结果是 0 的放在新表的低位，取于结果是 1 放在新表的高位。
- 4.2 如果这个桶是红黑数，那么也拆成 2 份，方式和链表的方式一样，然后，判断拆分过的树的节点数量，如果数量小于等于 6，改造成链表。反之，继续使用红黑树结构。
- 4.3 到这里，就完成了了一个桶从旧表转移到新表的过程。

好，以上，就是 transfer 方法的总体逻辑。还是挺复杂的。再进行精简，分成 3 步骤：

1. 计算每个线程可以处理的桶区间。默认 16。
2. 初始化临时变量 nextTable，扩容 2 倍。
3. 死循环，计算下标。完成总体判断。
4. 1 如果桶内有数据，同步转移数据。通常会像链表拆成 2 份。

大体就是上的的 3 个步骤。

再看看源码和注释。

再看源码分析

源码加注释：

```
/**
 * Moves and/or copies the nodes in each bin to new table. See
 * above for explanation.
 *
 * * transferIndex 表示转移时的下标，初始为扩容前的 length。
 *
 * * 我们假设长度是 32
 */
private final void transfer(Node<K,V>[] tab, Node<K,V>[] nextTab) {
    int n = tab.length, stride;
    // 将 length / 8 然后除以 CPU核心数。如果得到的结果小于 16，那么就使用 16。
    // 这里的目的是让每个 CPU 处理的桶一样多，避免出现转移任务不均匀的现象，如果桶较少的话，默认一个 CPU（一个线程）处理 16 个桶
    if ((stride = (Ncpu > 1) ? (n >>> 3) / Ncpu : n) < MIN_TRANSFER_STRIDE)
        stride = MIN_TRANSFER_STRIDE; // subdivide range 细分范围 stride: TODO
    // 新的 table 尚未初始化
    if (nextTab == null) { // initiating
        try {
            // 扩容 2 倍
            Node<K,V>[] nt = (Node<K,V>[])new Node<?,?>[n << 1];
            // 更新
            nextTab = nt;
        } catch (Throwable ex) { // try to cope with OOME
            // 扩容失败，sizeCtl 使用 int 最大值。
            sizeCtl = Integer.MAX_VALUE;
            return; // 结束
        }
        // 更新成员变量
```

(/apps/redirect?utm_source=side-banner-click)

(https://log-yex.youdao.com/ct?slot=30edd91dd86378ef7-487f-ad90-2c701414a735&yexigAwM0ITFbTaM36a1VNtTUBPvnBppYIVCa9FhKGM1CxoZk1hlclick.youdao.com%2l8ef7-487f-ad90-2c701414a735&iid=9

```

        nextTable = nextTab;

        // 更新转移下标, 就是 老的 tab 的 length
        transferIndex = n;
    }

    // 新 tab 的 length
    int nextn = nextTab.length;

    // 创建一个 fwd 节点, 用于占位。当别的线程发现这个槽位中是 fwd 类型的节点, 则跳过这个节点。

    ForwardingNode<K,V> fwd = new ForwardingNode<K,V>(nextTab);

    // 首次推进为 true, 如果等于 true, 说明需要再次推进一个下标 (i--), 反之, 如果是 false, 那么就不能推进下标, 需要将当前的下标处理完毕才能继续推进

    boolean advance = true;

    // 完成状态, 如果是 true, 就结束此方法。

    boolean finishing = false; // to ensure sweep before committing nextTab

    // 死循环, i 表示下标, bound 表示当前线程可以处理的当前桶区间最小下标

    for (int i = 0, bound = 0;;) {
        Node<K,V> f; int fh;

        // 如果当前线程可以向后推进; 这个循环就是控制 i 递减。同时, 每个线程都会进入这里取得自己需要转移的桶的区间

        while (advance) {
            int nextIndex, nextBound;

            // 对 i 减一, 判断是否大于等于 bound (正常情况下, 如果大于 bound 不成立, 说明该线程上次领取的任务已经完成了。那么, 需要在下面继续领取任务)

            // 如果对 i 减一大于等于 bound (还需要继续做任务), 或者完成了, 修改推进状态为 false, 不能推进了。任务成功后修改推进状态为 true。

            // 通常, 第一次进入循环, i-- 这个判断会无法通过, 从而走下面的 nextIndex 赋值操作 (获取最新的转移下标)。其余情况都是: 如果可以推进, 将 i 减一, 然后修改成不可推进。如果 i 对应的桶处理成功了, 改成可以推进。

            if (--i >= bound || finishing)
                advance = false; // 这里设置 false, 是为了防止在没有成功处理一个桶的情况下却进行了推进

            // 这里的目的是: 1. 当一个线程进入时, 会选取最新的转移下标。2. 当一个线程处理完自己的区间时, 如果还有剩余区间的没有别的线程处理。再次获取区间。

            else if ((nextIndex = transferIndex) <= 0) {
                // 如果小于等于0, 说明没有区间了, i 改成 -1, 推进状态变成 false, 不再推进, 表示, 扩容结束了, 当前线程可以退出了

                // 这个 -1 会在下面的 if 块里判断, 从而进入完成状态判断

                i = -1;

                advance = false; // 这里设置 false, 是为了防止在没有成功处理一个桶的情况下却进行了推进

            } // CAS 修改 transferIndex, 即 length - 区间值, 留下剩余的区间值供后面的线程使用

            else if (U.compareAndSwapInt
                    (this, TRANSFERINDEX, nextIndex,
                     nextBound = (nextIndex > stride ?
                                  nextIndex - stride : 0))) {
                bound = nextBound; // 这个值就是当前线程可以处理的最小当前区间最小下标

                i = nextIndex - 1; // 初次对 i 赋值, 这个就是当前线程可以处理的当前区间的最大下标

                advance = false; // 这里设置 false, 是为了防止在没有成功处理一个桶的情况下却进行了推进, 这样会导致漏掉某个桶。下面的 if (tabAt(tab, i) == f) 判断会出现这样的情况。

            }

            // 如果 i 小于0 (不在 tab 下标内, 按照上面的判断, 领取最后一段区间的线程扩容结束)

            // 如果 i >= tab.length (不知道为什么这么判断)

            // 如果 i + tab.length >= nextTable.length (不知道为什么这么判断)

            if (i < 0 || i >= n || i + n >= nextn) {
                int sc;

                if (finishing) { // 如果完成了扩容

```

(/apps/redirect?utm_source=side-banner-click)

(https://log-yex.youdao.com/ct?slot=30edd91dd86378ef7-487f-ad90-2c701414a735&yexi:gAwM0ITFbTaM36a1VNtTUBPvnBppYIVCa9FhKGM1CxoZk1hlclick.youdao.com%2l8ef7-487f-ad90-2c701414a735&iid=9

```

        nextTable = null; // 删除成员变量
        table = nextTab; // 更新 table
        sizeCtl = (n << 1) - (n >>> 1); // 更新阈值
        return; // 结束方法。
    } // 如果没完成
    if (U.compareAndSwapInt(this, SIZECTL, sc = sizeCtl, sc - 1)) { // 尝试将 sc
-1. 表示这个线程结束帮助扩容了，将 sc 的低 16 位减一。
        if ((sc - 2) != resizeStamp(n) << RESIZE_STAMP_SHIFT) // 如果 sc - 2 不
等于标识符左移 16 位。如果他们相等了，说明没有线程在帮助他们扩容了。也就是说，扩容结束了。
            return; // 不相等，说明没结束，当前线程结束方法。
        finishing = advance = true; // 如果相等，扩容结束了，更新 finishing 变量
        i = n; // 再次循环检查一下整张表
    }
}

else if ((f = tabAt(tab, i)) == null) // 获取老 tab i 下标位置的变量，如果是
null，就使用 fwd 占位。
    advance = casTabAt(tab, i, null, fwd); // 如果成功写入 fwd 占位，再次推进一个
下标

else if ((fh = f.hash) == MOVED) // 如果不是 null 且 hash 值是 MOVED。
    advance = true; // already processed // 说明别的线程已经处理过了，再次推进一个
下标

else { // 到这里，说明这个位置有实际值了，且不是占位符。对这个节点上锁。为什么上锁，防
止 putVal 的时候向链表插入数据
    synchronized (f) {
        // 判断 i 下标处的桶节点是否和 f 相同
        if (tabAt(tab, i) == f) {
            Node<K,V> ln, hn; // low, height 高位桶，低位桶
            // 如果 f 的 hash 值大于 0。TreeBin 的 hash 是 -2
            if (fh >= 0) {
                // 对老长度进行与运算（第一个操作数的第n位于第二个操作数的第n位如
果都是1，那么结果的第n为也为1，否则为0）
                // 由于 Map 的长度都是 2 的次方（000001000 这类的数字），那么取于
length 只有 2 种结果，一种是 0，一种是1
                // 如果是结果是0，Doug Lea 将其放在低位，反之放在高位，目的是将链
表重新 hash，放到对应的位置上，让新的取于算法能够击中他。

                int runBit = fh & n;
                Node<K,V> lastRun = f; // 尾节点，且和头节点的 hash 值取于不相等
                // 遍历这个桶
                for (Node<K,V> p = f.next; p != null; p = p.next) {
                    // 取于桶中每个节点的 hash 值
                    int b = p.hash & n;
                    // 如果节点的 hash 值和首节点的 hash 值取于结果不同
                    if (b != runBit) {
                        runBit = b; // 更新 runBit，用于下面判断 lastRun 该赋值给
ln 还是 hn。

                        lastRun = p; // 这个 lastRun 保证后面的节点与自己的取于值
相同，避免后面没有必要的循环
                    }
                }
            }
            if (runBit == 0) { // 如果最后更新的 runBit 是 0，设置低位节点
                ln = lastRun;
                hn = null;
            }
            else {
                hn = lastRun; // 如果最后更新的 runBit 是 1，设置高位节点
                ln = null;
            }
            // 再次循环，生成两个链表，lastRun 作为停止条件，这样就是避免无谓
的循环（lastRun 后面都是相同的取于结果）

```

(/apps/redirect?
utm_source=side-
banner-click)

(https://log-
yex.youdao.com/ct?
slot=30edd91dd8637
8ef7-487f-ad90-
2c701414a735&yexi:
gAwM0ITfbTaM36a1
VNtTUBPvnBppYIVC
a9FhKGM1CxoZk1hl
click.youdao.com%2l
8ef7-487f-ad90-
2c701414a735&iid=9

```

for (Node<K,V> p = f; p != lastRun; p = p.next) {
    int ph = p.hash; K pk = p.key; V pv = p.val;
    // 如果与运算结果是 0, 那么就还在低位
    if ((ph & n) == 0) // 如果是0, 那么创建低位节点
        ln = new Node<K,V>(ph, pk, pv, ln);
    else // 1 则创建高位
        hn = new Node<K,V>(ph, pk, pv, hn);
}
// 其实这里类似 hashMap
// 设置低位链表放在新链表的 i
setTabAt(nextTab, i, ln);
// 设置高位链表, 在原有长度上加 n
setTabAt(nextTab, i + n, hn);
// 将旧的链表设置成占位符
setTabAt(tab, i, fwd);
// 继续向后推进
advance = true;
} // 如果是红黑树
else if (f instanceof TreeBin) {
    TreeBin<K,V> t = (TreeBin<K,V>)f;
    TreeNode<K,V> lo = null, loTail = null;
    TreeNode<K,V> hi = null, hiTail = null;
    int lc = 0, hc = 0;
    // 遍历
    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);
        // 和链表相同的判断, 与运算 == 0 的放在低位
        if ((h & n) == 0) {
            if ((p.prev = loTail) == null)
                lo = p;
            else
                loTail.next = p;
            loTail = p;
            ++lc;
        } // 不是 0 的放在高位
        else {
            if ((p.prev = hiTail) == null)
                hi = p;
            else
                hiTail.next = p;
            hiTail = p;
            ++hc;
        }
    }
    // 如果树的节点数小于等于 6, 那么转成链表, 反之, 创建一个新的树
    ln = (lc <= UNTREEIFY_THRESHOLD) ? untreeify(lo) :
        (hc != 0) ? new TreeBin<K,V>(lo) : t;
    hn = (hc <= UNTREEIFY_THRESHOLD) ? untreeify(hi) :
        (lc != 0) ? new TreeBin<K,V>(hi) : t;
    // 低位树
    setTabAt(nextTab, i, ln);
    // 高位数
    setTabAt(nextTab, i + n, hn);
    // 旧的设置成占位符
    setTabAt(tab, i, fwd);
    // 继续向后推进

```

(/apps/redirect?
utm_source=side-
banner-click)

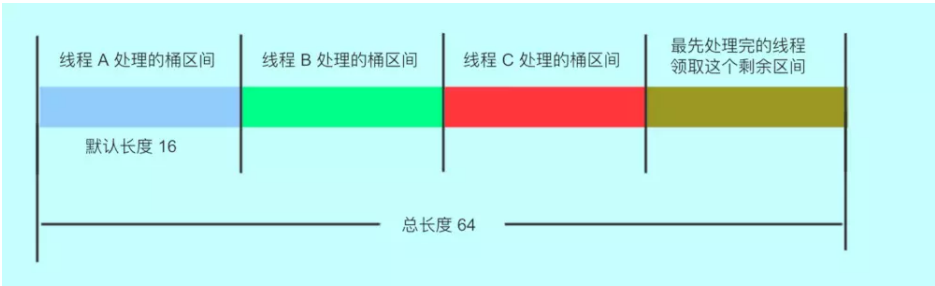
(https://log-
yex.youdao.com/ct?
slot=30edd91dd8637
8ef7-487f-ad90-
2c701414a735&yexi:
gAwM0ITFbTaM36a1
VNtTUBPvnBppYIVC
a9FhKGM1CxoZk1hl
click.youdao.com%2l
8ef7-487f-ad90-
2c701414a735&iid=9

```
        advance = true;
    }
}
}
}
}
```

代码加注释比较长，有兴趣可以逐行对照，有 2 个判断楼主看不懂为什么这么判断，知道的同学可以提醒一下。

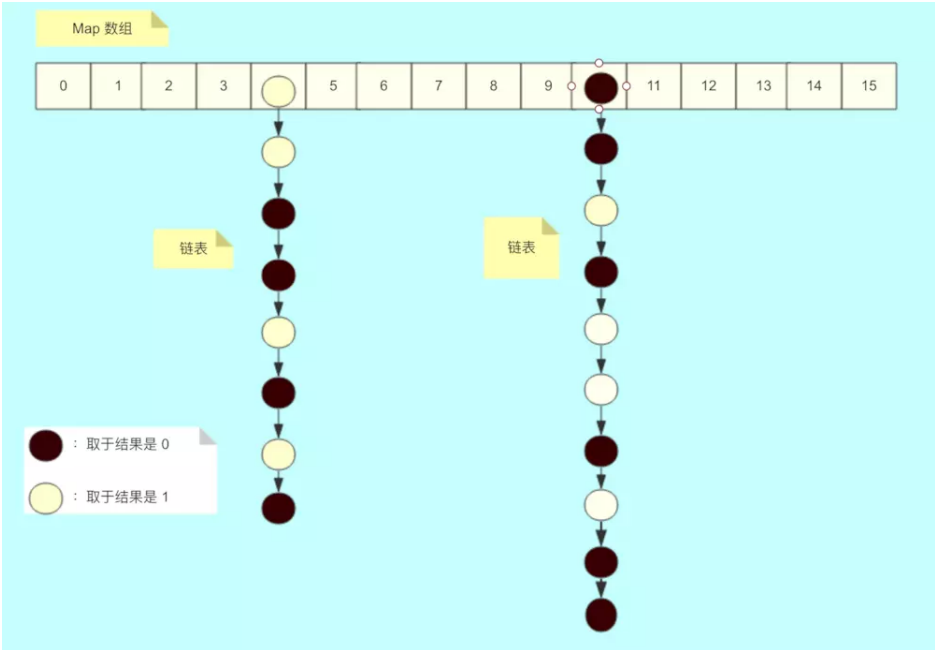
然后，说说精华的部分。

1. Cmap 支持并发扩容，实现方式是，将表拆分，让每个线程处理自己的区间。如下图：



假设总长度是 64，每个线程可以分到 16 个桶，各自处理，不会互相影响。

2. 而每个线程在处理自己桶中的数据的时候，是下图这样的：



扩容前的状态。

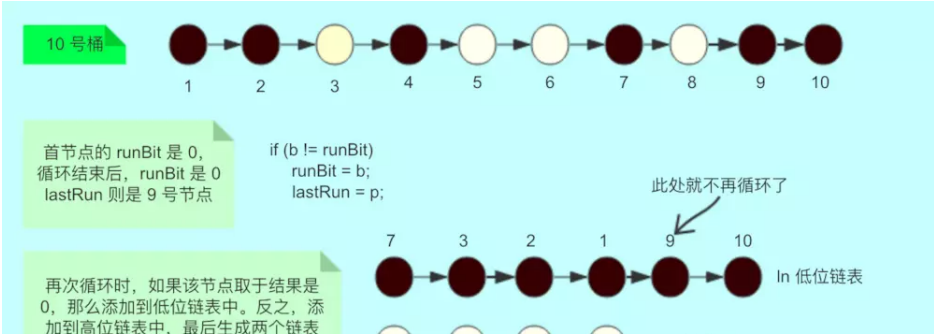
当对 4 号桶或者 10 号桶进行转移的时候，会将链表拆成两份，规则是根据节点的 hash 值取于 length，如果结果是 0，放在低位，否则放在高位。

因此，10 号桶的数据，黑色节点会放在新表的 10 号位置，白色节点会放在新桶的 26 号位置。

下图是循环处理桶中数据的逻辑：

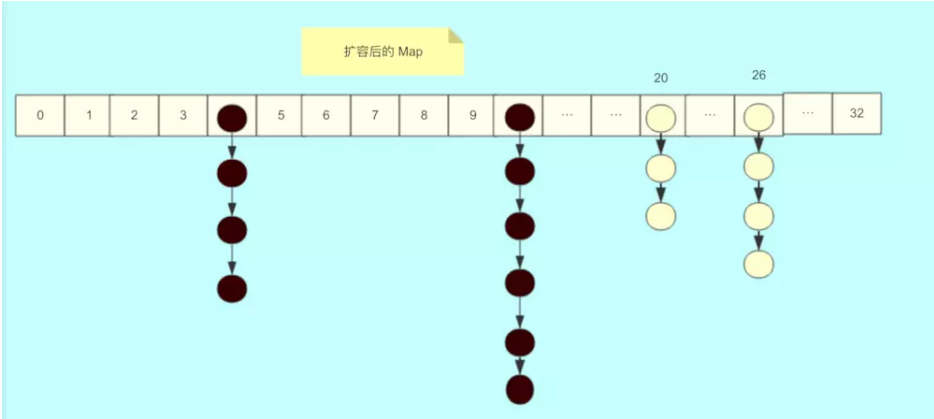
(/apps/redirect?utm_source=side-banner-click)

(https://log-yex.youdao.com/ct?slot=30edd91dd86378ef7-487f-ad90-2c701414a735&yexi:gAwM0ITFbTaM36a1VNtTUBPvnBppYIVCa9FhKGM1CxoZk1hlclick.youdao.com%2f8ef7-487f-ad90-2c701414a735&iid=9



(/apps/redirect?utm_source=side-banner-click)

处理完之后, 新桶的数据是这样的:



(https://log-yex.youdao.com/ct?slot=30edd91dd86378ef7-487f-ad90-2c701414a735&yexigAwM0ITFbTaM36a1VNtTUBPvnBppYIVCa9FhKGM1CxoZk1hlclick.youdao.com%2l8ef7-487f-ad90-2c701414a735&iid=9

总结

transfer 方法可以说很牛逼, 很精华, 内部多线程扩容性能很高,

通过给每个线程分配桶区间, 避免线程间的争用, 通过为每个桶节点加锁, 避免 putVal 方法导致数据不一致。同时, 在扩容的时候, 也会将链表拆成两份, 这点和 HashMap 的 resize 方法类似。

而如果有新的线程想 put 数据时, 也会帮助其扩容。鬼斧神工, 令人赞叹。

欢迎访问 thinkinjava.cn

赞赏支持

📖 并发编程源码分析 (/nb/12258710)

举报文章 © 著作权归作者所有



莫那一鲁道 (/u/434239413000) ♂

写了 398626 字, 被 987 人关注, 获得了 1148 个喜欢

(/u/434239413000)

+ 关注

Java 程序员 个人网站: <http://thinkinjava.cn> GitHub: <https://github.com/statels0> Stay hungry. Stay foolish.

喜欢 | 9



更多分享

https://www.jianshu.com/p/2829fe36a8dd

7/10



下载简书 App ▶

随时随地发现和创作内容



(/apps/redirect?utm_source=note-bottom-click)


(/apps/redirect?utm_source=side-banner-click) >




登录 (/sign-in?utm_source=desktop&utm_medium=not-signed-in-comment-form) 后发表评论

3条评论 只看作者

按时间倒序 按时间正序

 那时浮华染流年_2765 (/u/716252dd5040)
3楼 · 2019.06.01 15:04
(/u/716252dd5040)
为什么第9个节点不再循环了呢？

赞 回复

 ZOKE (/u/d99922588a02)
2楼 · 2019.05.07 16:19
(/u/d99922588a02)
后面链表迁移时，4号节点不见了哦


赞 回复

莫那一鲁道 (/u/434239413000)：@ZOKE (/u/d99922588a02) 画漏了 😊
2019.05.09 11:10 回复

添加新评论

(https://log-yex.youdao.com/ct?slot=30edd91dd86378ef7-487f-ad90-2c701414a735&yexigAwM0ITFbTaM36a1VNtTUBPvnBppYIVCa9FhKGM1CxoZk1hlclick.youdao.com%2f8ef7-487f-ad90-2c701414a735&iid=9

被以下专题收入，发现更多相似内容

 java基础 (/c/95ff462207a9?utm_source=desktop&utm_medium=notes-included-collection)

推荐阅读

更多精彩内容 > (/)

【华为云】热门主机，每天不到0.7元！


实名认证即可抽奖，1核2G云主机等你抽，立即认证>>

广告

(https://log-yex.youdao.com/ct?slot=f2ac00aef0eb6b673f4e4639046bc6f8&youdao_bid=d8da83ee-d2b2-45bb-80click.youdao.com%2Fclk%2Frequest.s%3Fk%3DjcmnmEPJyUt%252FOARgN8a3B4ExK4SIB42phw0DLZifsQzXd2b2-45bb-8027-b46a3f00498f&iid=%7B%221416370023083168763%22%3A1%7D&sid=17836)

Java面试宝典Beta5.0 (/p/fb7d48083e5e?utm_campaign=maleskine&ut...

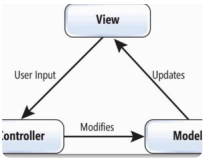
pdf下载地址：Java面试宝典 第一章内容介绍 20 第二章JavaSE基础 21 一、Java面向对象 21 1. 面向对象都有哪些特性以及你对这些特性的理解 21 2. 访问权限修饰符public、private、protected, 以及不写（默认）...

 王震阳 (/u/773a782d9d83?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/d975dc04c8e2?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

【JAVA】面试宝典 (/p/d975dc04c8e2?utm_campaign=maleskine&utm...

Java中如何实现代理机制(JDK、CGLIB) JDK动态代理：代理类和目标类实现了共同的接口，用到InvocationHandler接口。CGLIB动态代理：代理类是目标...



Y了个J (/u/5d7da7422e3e?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation) (/apps/redirect?utm_source=side-banner-click)

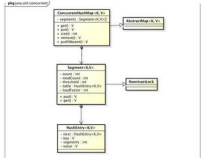
面试知识点1 (/p/044ab3dc40c3?utm_campaign=maleskine&utm_conte...

Java8张图 11、字符串不变性 12、equals()方法、hashCode()方法的区别 13、Java异常类的层次结构 14、集合类的层次结构 25、Java同步 36、别名 37、堆和栈 38、Java虚拟机运行时数据区域 3...

Miley_MOJIE (/u/46bdcdace9a2?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/7f42ba895a64?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

Java并发编程（六）阻塞队列 (/p/7f42ba895a64?utm_campaign=maleski...

相关文章Java并发编程（一）线程定义、状态和属性 Java并发编程（二）同步Java并发编程（三）volatile 域Java并发编程（四）Java内存模型 前言 在Java1.5中，并发编程大师Doug Lea给我们带来了concurrent...

刘望舒 (/u/5d38c81be78e?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation) 2c701414a735&yexi:

(/p/f9b3e76951c2?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

ConcurrentHashMap源码分析 (/p/f9b3e76951c2?utm_campaign=males...

1.ConcurrentHashMap简介 在使用HashMap时在多线程情况下扩容会出现CPU接近100%的情况，因为hashmap并不是线程安全的，通常我们可以使用在java体系中最古老的hashtable类，该类基本上所有的方法...

铁甲依然在_978f (/u/751ef2f3e0f5?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/cb1ed6dbad7f?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

他说 (/p/cb1ed6dbad7f?utm_campaign=maleskine&utm_content=note...

虽然很心动，但要克制住自己，因为你了解他，五分夸大成十分。做朋友就好，记得有这么个人，至少还有你

不想说话啦 (/u/2ca772c89dff?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/6aea0e2acb71?)




utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

别再为你的自私和懦弱、找借口了 (/p/6aea0e2acb71?utm_campaign=ma...

昨天看到一则很让人痛心的新闻：河北省邯郸市魏县一女子惨遭丈夫囚禁家暴20年，更是被迫给情妇洗衣做

饭。而更让人觉得过分的是：这位丈夫在2002年以抢劫罪被判12年，后在监狱改造10年后出狱。而这个...


 菩提子夜 (/u/9567efef138d?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/apps/redirect?
utm_source=side-
banner-click)

听雨 (/p/406b2cfd1caf?utm_campaign=maleskine&utm_content=note&...


雨，轻轻地落到了地面。我望着这雨，思绪万千。树木，探出脑袋享受雨的洗礼。鸟儿，纷纷躲进巢里。花儿落了。这场来去匆匆的雨啊。让我心情有些惆怅！

 晴儿姑娘 (/u/e36abe635cad?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

英语口语 (/p/0de48805b365?utm_campaign=maleskine&utm_content=...

我们当年学的是哑巴英语，导致现在想帮助孩子开口都难，因为我不能开口去跟孩子交流，创造不了英语对话场景，更别提孩子会去用英语和我们交流了。所以，我想，是不是应该开始学习一些日常对话，说起来...

 dongjasmine (/u/fe7f125aee6f?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/521bca78eac0?




(https://log-
yex.youdao.com/ct?
slot=30edd91dd8637
8ef7-487f-ad90-
2c701414a735&yexi:
gAwM0ITFbTaM36a1
VNtTUBPvnBppYIVC
a9FhKGM1CxoZk1hl
click.youdao.com%2l
8ef7-487f-ad90-
2c701414a735&iid=9

utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

git使用 (/p/521bca78eac0?utm_campaign=maleskine&utm_content=no...

昨天在同事电脑上操作了一把cherry-pick代码,发现很多功能不用,就慢慢忘记了,梳理了下流程图: git commit -
-amend 前提是没push cherry-pick [commit git branch -r --contains cfc3521a32b...

 gogoingmonkey (/u/35abf4e43aa5?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)