


Java AbstractQueuedSynchronizer源码阅读2-addWaiter()



作者

lzwang2 (/u/ba70fb78fecf)

+关注

2016.05.06 15:07 字数 1734 阅读 163 评论 3 喜欢 0

(/u/ba70fb78fecf)

AbstractQueuedSynchronizer既然是同步器实现框架，关键便在于处理好多线程运行时的
问题。通过Java AbstractQueuedSynchronizer源码阅读1-基于队列的同步器框架
(<http://www.jianshu.com/p/9ebca222513b>)，可以了解到addWaiter()的功能是将Node入
队，那么addWaiter()是如何保证多线程运行下入队操作的正确性的呢？

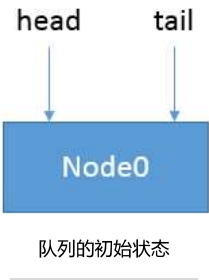
addWaiter()使用了原子性方法compareAndSetTail()。为方便叙述，将addWater()的代码
粘贴如下：

```
private Node addWaiter(Node mode) {  
    Node node = new Node(Thread.currentThread(), mode); //新建与一个当前线程关联的node  
    // Try the fast path of enq; backup to full enq on failure  
    Node pred = tail;  
    if (pred != null) { //如果tail不为空  
        node.prev = pred; //将新建的node加入到队尾  
        if (compareAndSetTail(pred, node)) { //调用CAS (CompareAndSet) 重新设置tail  
            pred.next = node;  
            return node;  
        }  
    }  
    //如果入队失败了，则调用enq()  
    enq(node);  
    return node;  
}
```

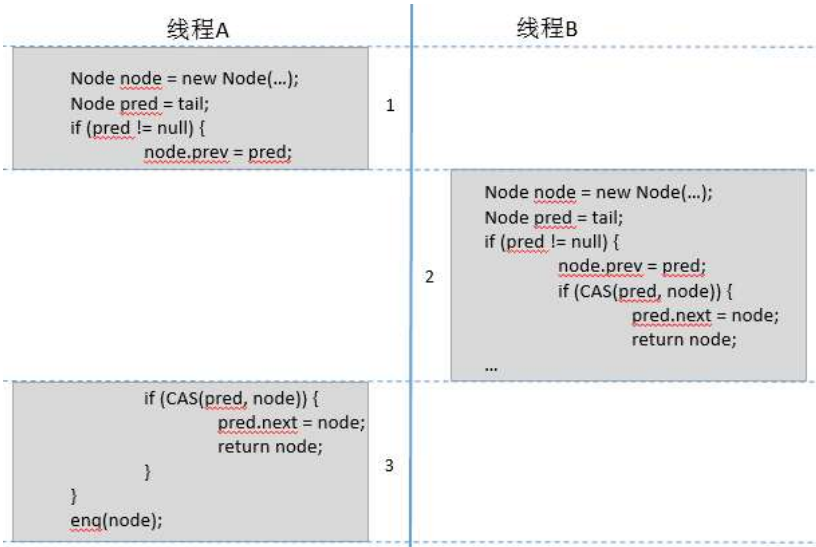
compareAndSetTail(pred, node)会比较pred和tail是否指向同一个节点，如果是，才将tail
更新为node。为何不是直接赋值，而要多做一步比较操作呢？那是因为虽然当前线程在
声明pred时，为pred赋值了tail，但tail可能会被其他线程改变，而当前线程的本地变量
pred是不会感知到这个改变的。

这里，通过模拟多线程执行来加深这个理解。

假设队列的初始状态如下，只有一个Node（称为Node0），队列的head和tail都指向
Node0。

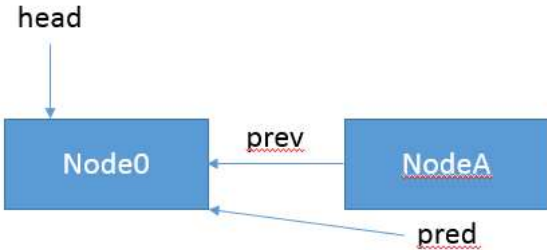


现在有线程A和线程B同时调用addWaiter()，要向该队列插入新的Node。假设线程A和线
程B的执行流程如下图所示（省略了部分代码，并对代码采取了简写）：



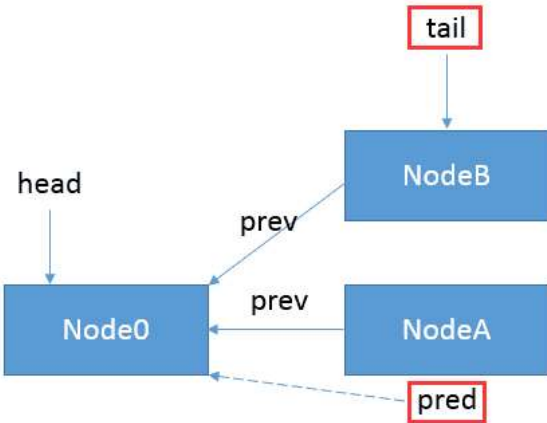
线程A和线程B的执行流程

第1步：线程A新建了Node（称为NodeA），并开始入队。但是，线程A仅来的及将NodeA的prev指针赋值为tail，便被调度出处理器。此时队列的状态如下图所示，NodeA的prev指向了Node0。图中还另外标注了线程A的局部变量pred，也是指向Node0的。



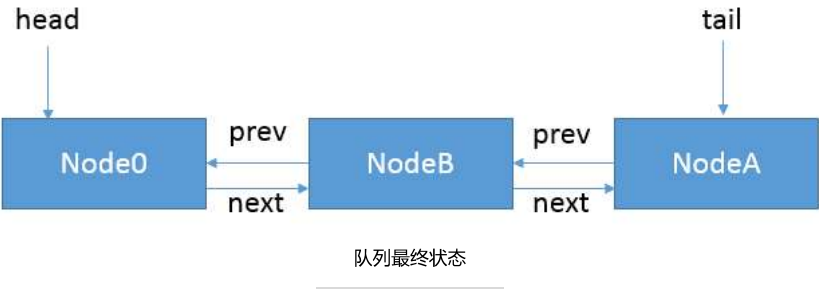
第1步

第2步：紧接着，线程B开始占用处理器，执行第2步。线程2也新建了一个Node（称为NodeB），并且线程B成功的执行完addWaiter()，将NodeB入队。此时队列的状态如下图所示，NodeB的prev也指向了Node0。此处关键要注意的是，tail此时已经指向了NodeB，但是线程A的pred依然指向Node0。

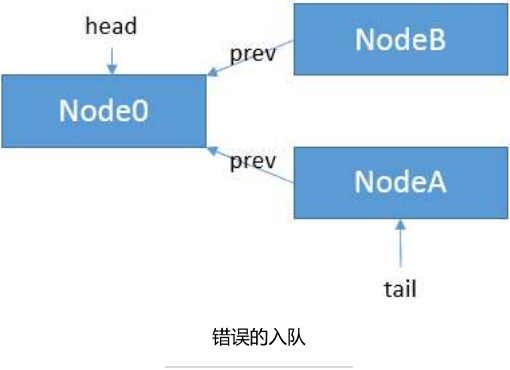


第2步

第3步：线程B在入队完成后，线程A又开始占用处理器执行。线程A调用compareAndSetTail(pred, tail)，发现pred和tail并不是指向同一个Node的，该方法会返回false，线程A尝试快速入队失败。之后，线程A会调用enq()，重新获取tail，并不停尝试入队直到成功。这里不再展开enq()方法。最终队列的状态会如下所示，线程B因为先于线程A成功调用了compareAndSetTail()，而位于A的前面。



如果使用的不是CAS方法，而是直接采用赋值的方式（即将compareAndSetTail(pred, node)换成tail=node），则在第3步时，会得到下面这个错误的结果。



所以说，入队的同步关键在于原子性的compareAndSetTail()方法。它保证了每个线程能够完整的执行下面两个操作：

- 1. 设置prev，将自己链接到队尾；
- 2. 将tail更新为自己。

这使得队列中的tail和prev指针总是可靠的，用户在任何时候都可以使用tail和prev去访问队列。

提出一些问题

为何不将如下入队的两步关键性操作封装为原子性操作

- 1. 设置prev，将自己链接到队尾；
- 2. 将tail更新为自己。

如果将这两步封装为原子性操作，那么正确的入队就可以一次性完成。而原本的实现中，CAS失败后，还需要再重试。但是，AbstractQueuedSynchronizer本身是要实现原子性的操作，而其本身又依赖原子性的操作，感觉有点像是先有鸡还是先有蛋的问题了。其实，CAS的原子性是依赖机器指令实现的，但是机器指令无法支持以上两步执行的原子性。AbstractQueuedSynchronizer采取的方法是，依赖原子性的CAS以及循环，来实现上述两步的原子性。

为何addWaiter()实现时，是按照下面这个顺序

- 1. 设置prev指针；
- 2. compareAndSetTail()；
- 3. 设置next指针；

而不是3-2-1或是1-3-2这种顺序呢？

对于3-2-1，我认为，其实和1-2-3的本质是一样的，只是代码实现时，选择prev指针而已。1-2-3这种方式保证了prev的可靠性，可以看到AbstractQueuedSynchronizer中一些需要遍历队列的方法，如getQueuedThreads()，都是使用的prev。

对于1-3-2，我们可以按照上面的图“线程A和线程B的执行流程”，再推演一遍，可以发现，线程A在快速入队时，对NodeA的prev指针和next指针的设置都浪费了。而1-2-3的顺序下，仅是浪费了设置prev指针这一步。

总而言之，addWaiter()的实现保证了prev指针的可靠性。

那么，next指针既然不可靠，那为何还需要呢？prev指针不是已经保证了队列的可访问性了么？引用源码中对Node的注释。

We also use "next" links to implement blocking mechanics. The thread id for each node is kept in its own node, so a predecessor signals the next node to wake up by traversing next link to determine which thread it is. Determination of successor must avoid races with newly queued nodes to set the "next" fields of their predecessors. This is solved when necessary by checking backwards from the atomically updated "tail" when a node's successor appears to be null. (Or, said differently, the next-links are an optimization so that we don't usually need a backward scan.)

这段话的大意是说某个节点在释放锁时，需要唤醒其后继节点。如果没有next指针，那么每次都需要从tail往前遍历，next指针则优化了这一操作。但是要注意的是，因为next指针并不可靠，所以有时候next指针会是null，此时，依然需要依赖tail指针向前回溯，以找到期望的节点。

IT (/nb/4985795)

举报文章 © 著作权归作者所有



lzwang2 (/u/ba70fb78fecf)

写了 44955 字，被 7 人关注，获得了 10 个喜欢
(/u/ba70fb78fecf)

+ 关注

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

赞赏支持

♡ 喜欢 (/sign_in) | 0



更多分享

(http://cwb.assets.jianshu.io/notes/images/3833076/v



登录 (/sign_in) 后发表评论

3条评论

只看作者

按喜欢排序 按时间正序 按时间倒序



晨水无尘 (/u/6c6cb8b2741f)

2楼 · 2016.12.12 15:03


(/u/6c6cb8b2741f)

我用enq方法测试如果是 node0 后面两个线程哪个先执行enq方法return t 那一步 谁就排在前面 不知道 我解释这个对不对 和你这个月应该一样


👍 赞 💬 回复


lzwang2 (/u/ba70fb78fecf) : @晨水无尘 (/users/6c6cb8b2741f) O_O 没想到有人回复了。。。我又重温了下，关键在于这个方法：compareAndSetTail，谁成功的执行了这个方法谁就先入队了。一个node要入队就是要将自己加到队尾成为tail，所以如果谁成功的先将自己set为了tail，谁就先入队了。

我也不过小白一个，没有加什么群。。。交流少的很。。。只不过有次三分钟热度，看了下这东西。。。

2016.12.16 22:57  回复

晨水无尘 (/u/6c6cb8b2741f) : @lzwang2 (/users/ba70fb78fecf) 觉得你分析的不错 你有qq 或者什么吗 可以多交流

2016.12.23 11:01  回复

 添加新评论