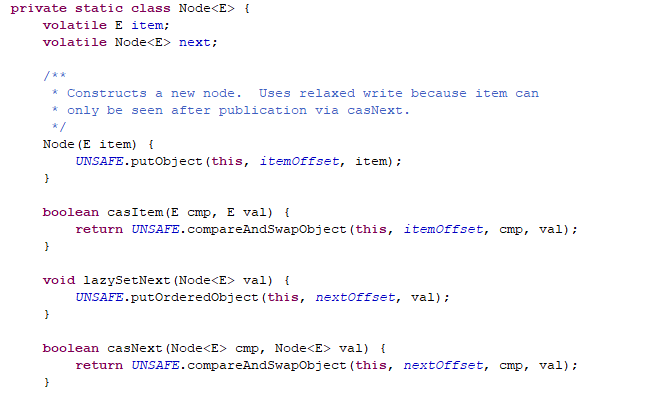
# ConcurrentLinkedQueue(jdk1.7.0\_67)

ConcurrentLinkedQueue是一个无界的线程安全的队列，内部实现是基于一个先进先出的链表。链表是通过内部类Node实现的，先看一下这个内部类。



item和next属性都被声明为volatile，通过casItem和casNext更新属性的值，确保了多线程下的线程安全。

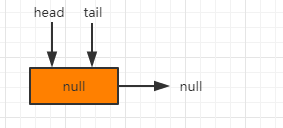
ConcurrentLinkedQueue不允许null值的插入。

# 1.构造器

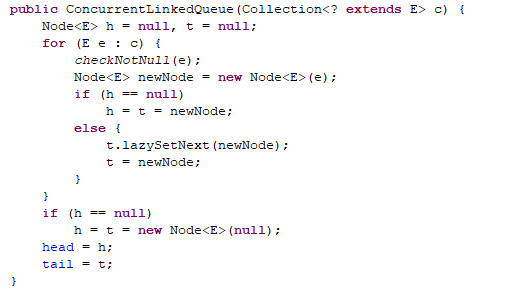
## 1.1默认构造器



初始化了head，tail节点。这是ConcurrentLinkedQueue的内部构造如下所示：



## 1.2其他构造器

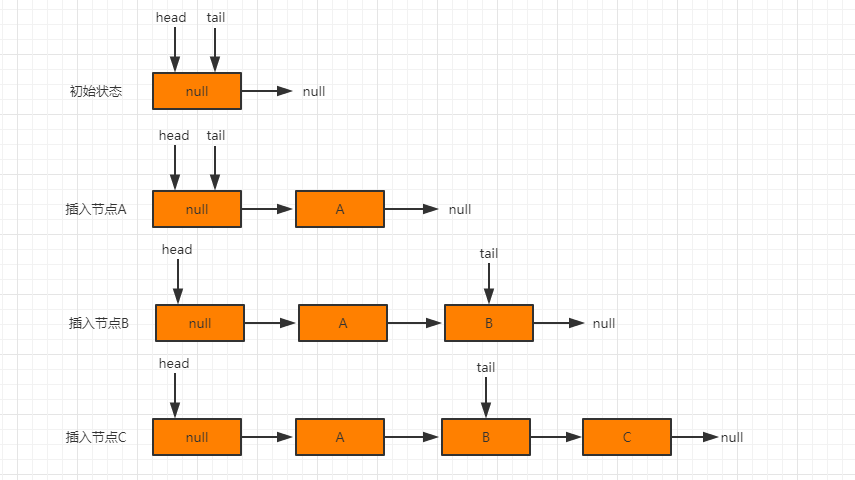


遍历集合c中所有的元素e，首先检查元素非null，如果元素为null，抛出异常，否则将元素e添加到链表的尾部，h为头节点，t为尾节点，更新尾节点t。然后判断h为null，即c为空集合，就创建一个null节点，将h，t指向它。最后更新head，tail属性。

# 2.方法

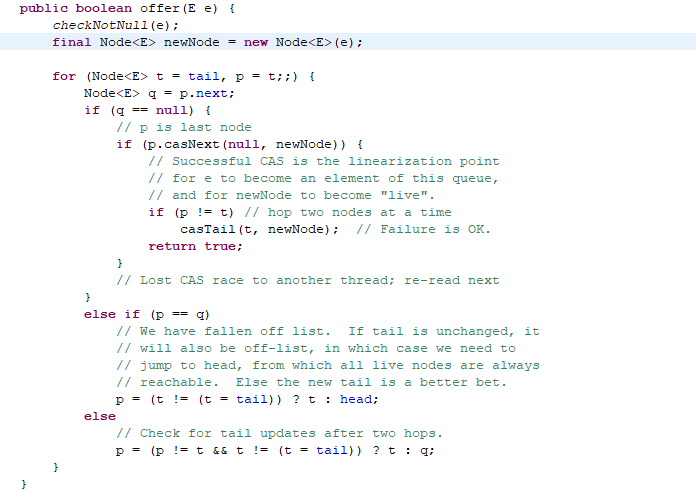
## 2.1插入方法

首先看一下ConcurrentLinkedList插入节点的流程，如下图所示：

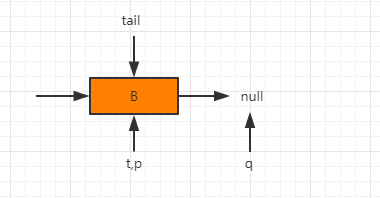


可以看到tail属性并不是一直指向尾节点的，这里被设计为没插入两个节点更新一次tail属性，这样做得目的是为了减少CAS更新tail属性的次数，从而提高了性能。

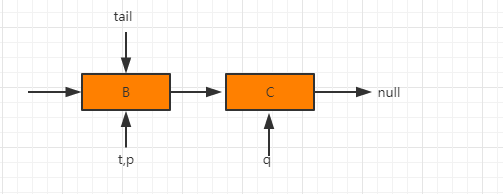
### 2.1.1offer



1. 先看一下q==null的情况，



假设casNext调用成功，此时p==t直接返回。假设有这样一种情况，线程A和线程B都执行到了q==null，这时线程B抢先一步执行了casNext并成功退出，此时线程A在执行casNext将会失败，并重新进入循环，这时候线程A看的内部状态如下图所示：

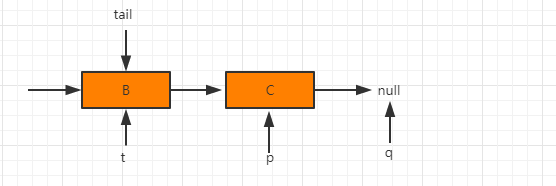


这时变成了第二种情况。

1. q!=null并且p!=q，这时将执行else下的代码。

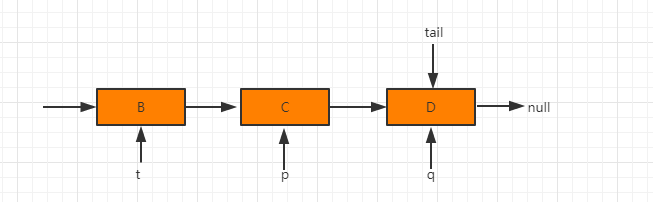


此时p==t所以p=q，继续循环，这时内部状态如下图：



这时候变成了情况三。

1. q==null并且p!=t，假设casNext成功，这时需要重新设置tail属性值。这里更新tail失败也没关系，因为必然是其他线程成功更新了tail。假设casNext不成功，即有其他线程插入了节点，此时内部状态如下图所示：

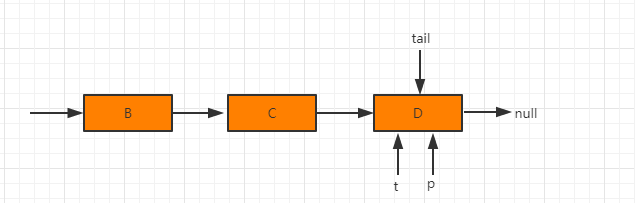


这时候变成了情况四。

1. q!=null并且p!=q，这时执行以下代码。

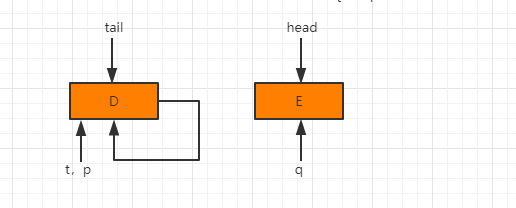


p!=t&&t!=tail返回true，p=t，t=tail将p，t重新指向了tail节点。此时状态如下所示：



然后继续循环。q=null，这时又变成了情况一。

1. 还有p==q这条分支没走。当并发删除的时候可能出现以下情况。

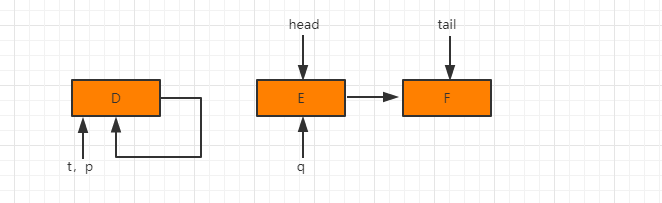


这时候执行p==q分支中的代码。



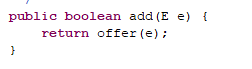
此时t==tail，p=head继续循环。

另一种情况t!=tail。



这时候将t=tail，p=t继续循环。

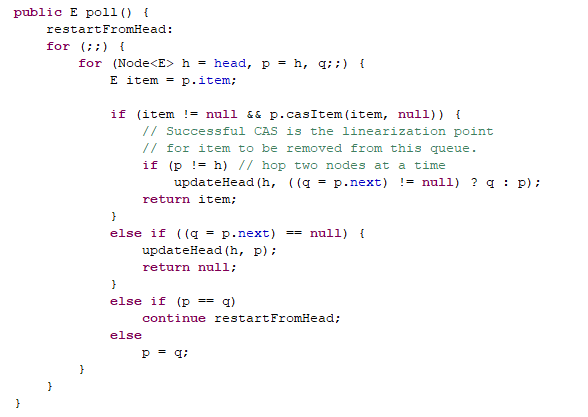
### 2.1.2add



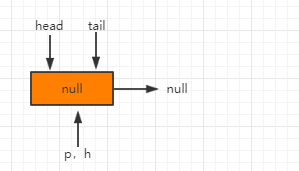
调用了offer方法。

## 2.2删除方法

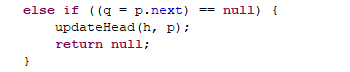
### 2.2.1poll

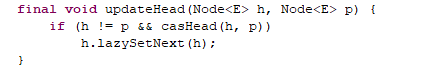


1. 先看第一种情况，当未插入元素时候。此时状态为下图所示。



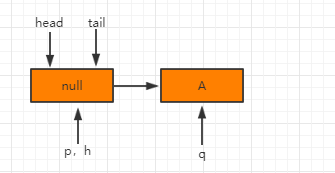
此时item==null，q=p.next==null，然后执行以下代码





h==p，所以直接返回null。

假设在判断q=p.next==null之前，正好有其他线程插入了新的节点，这时候内部的状态如下所示。

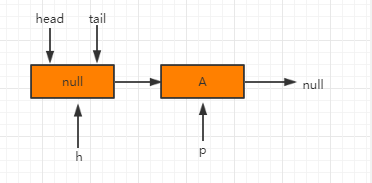


然后当前线程判断q==null失败，这时进入第二种情况。

1. p!=null，执行以下代码。

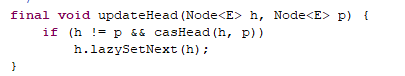


状态如下图所示。

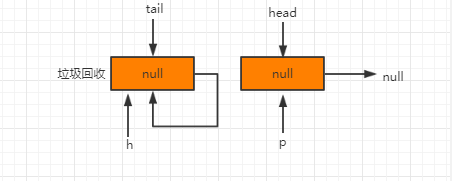


这样进入了情况三。

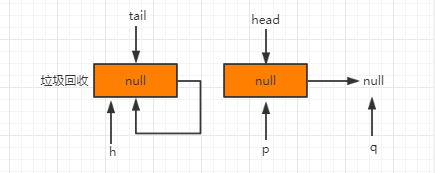
1. item!=null，假设casItem成功，此时p!=h将调用updateHead重新设置head属性。



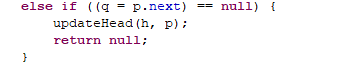
这里传入的h为原head节点，p为A节点。此时h!=p，假设casHead成功，调用lazySetNext延迟将原head节点的next指向自己。形成以下状态。这里casHead失败没关系，因为必然是其他线程重新设置了head。



假设casItem失败，必然是其他线程更新了该节点，此时状态为。

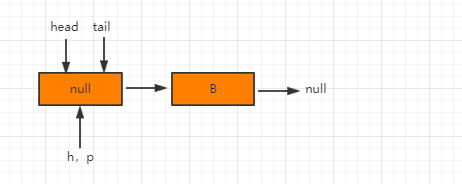


这时候将执行。

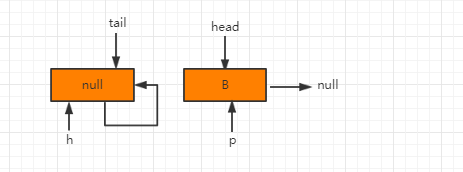


返回null。

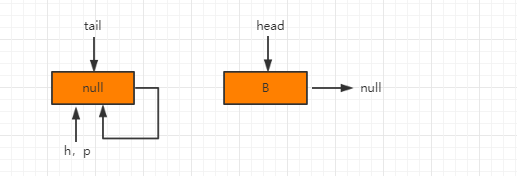
1. 还有p==q这个分支没有走。假设有两个线程A和B，线程A和线程B都看到如下的状态。



这时线程B继续执行，直到删除节点。线程B看到的状态如下图。

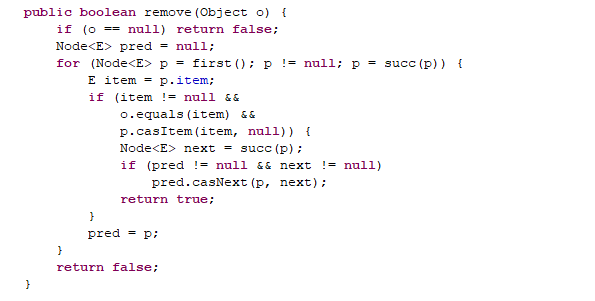


而线程A看到的状态如下图。

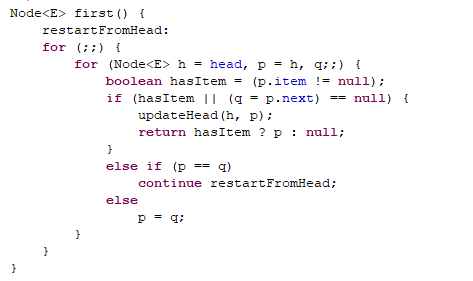


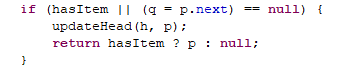
然后线程A继续执行。q=p.next也就是p==q，调到外层循环继续执行。

### 2.2.2remove



遍历整个ConcurrentLinkedQueue，查找到元素就删除，返回true，否则返回false。主要看一下first跟succ这两个方法，first获取第一个节点，succ获取后继节点。





hasItem为true，表示p为第一个元素，将head更新为p，并放回p。

q=p.next == null为true，表示初始情况，没有节点插入，返回null。

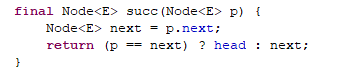


这种情况是正好有一个线程插入了新的节点。



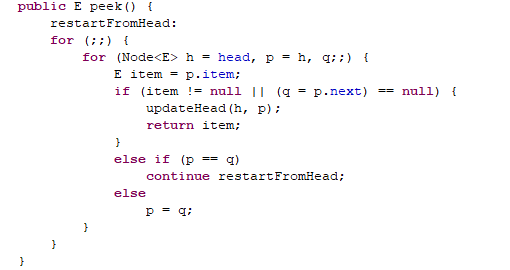
这种情况是有其他线程删除了节点。

下面看一下succ方法。



## 2.3获取元素

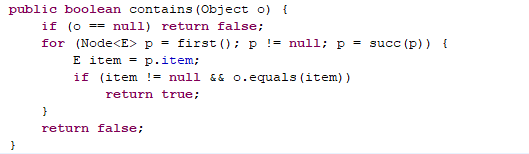
### 2.3.1peek



这个方法跟first方法类似，第一次peek后会将head指向第一元素节点。

## 2.4其他方法

### 2.4.1contains



遍历整个ConcurrentLinkedQueue，查找元素。

### 2.4.2isEmpty



判断第一个元素是否为null。

### 2.4.3size

