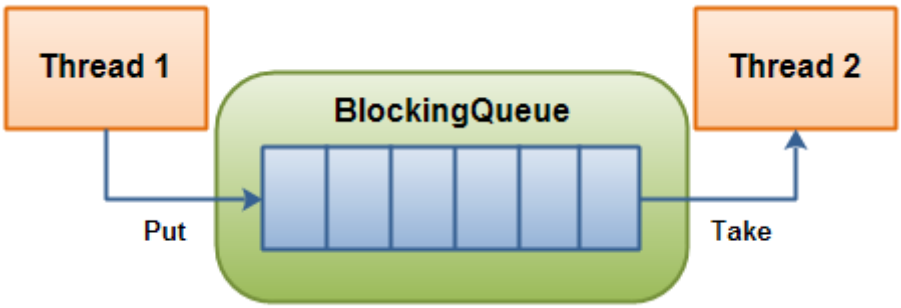
阻塞队列

原文地址 By Jakob Jenkov 翻译：寒桐 校对：方腾飞

阻塞队列与普通队列的区别在于，当队列是空的时，从队列中获取元素的操作将会被阻塞，或者当队列是满时，往队列里添加元素的操作会被阻塞。试图从空的阻塞队列中获取元素的线程将会被阻塞，直到其他的线程往空的队列插入新的元素。同样，试图往已满的阻塞队列中添加新元素的线程同样也会被阻塞，直到其他的线程使队列重新变得空闲起来，如从队列中移除一个或者多个元素，或者完全清空队列，下图展示了如何通过阻塞队列来合作：



线程1往阻塞队列中添加元素，而线程2从阻塞队列中移除元素

从5.0开始，JDK在java.util.concurrent包里提供了阻塞队列的官方实现。尽管JDK中已经包含了阻塞队列的官方实现，但是熟悉其背后的原理还是很有帮助的。

阻塞队列的实现

阻塞队列的实现类似于带上限的Semaphore的实现。下面是阻塞队列的一个简单实现

```
01 public class BlockingQueue {
02
03 private List queue = new LinkedList();
04
05 private int  limit = 10;
06
07 public BlockingQueue(int limit){
08
09 this.limit = limit;
10
11 }
12
13 public synchronized void enqueue(Object item)
14
15 throws InterruptedException  {
16
17 while(this.queue.size() == this.limit) {
```

```
18
19  wait();
20
21  }
22
23  if(this.queue.size() == 0) {
24
25  notifyAll();
26
27  }
28
29  this.queue.add(item);
30
31  }
32
33  public synchronized Object dequeue()
34
35  throws InterruptedException{
36
37  while(this.queue.size() == 0){
38
39  wait();
40
41  }
42
43  if(this.queue.size() == this.limit){
44
45  notifyAll();
46
47  }
48
49  return this.queue.remove(0);
50
51  }
52
53  }
```

必须注意到，在enqueue和dequeue方法内部，只有队列的大小等于上限（limit）或者下限（0）时，才调用notifyAll方法。如果队列的大小既不等于上限，也不等于下限，任何线程调用enqueue或者dequeue方法时，都不会阻塞，都能够正常的往队列中添加或者移除元素。

Wayne

2013/12/05 5:57下午

为什么在enqueue和dequeue方法内部，只有队列的大小等于上限（limit）或者下限（0）时，才调用notifyAll方法？

这个样子我觉得可能会增加线程间切换

假设有3个线程 a b c 而且queue size=0

a b在执行 equeue, c 在执行 dequeue

1.首先a执行equeue方法 拿到lock，执行到if判断，执行notifyAll

2.假设c抢到lock，执行dequeue方法，结果while判断，执行wait()

3.假设b于a前 先拿到了lock，然后执行equeue方法，执行if判断仍然为true，再执行notifyAll

目前是a和c都可以对这个锁形成竞争，最差的情况仍然被c抢到 执行while结果仍然wait，

接下去 无论 a 或者 b抢到lock 都可以顺利执行add操作

==========

所以我认为这句if可能会增加cpu调度频繁切换，如果去掉if语句 实际逻辑判断也不会有太大变化，但是至少能减少角度问题

> xiezc
> 2018/03/31 9:17下午
>
> 在执行了notify方法之后，当前线程不会马上释放该对象锁，呈wait状态的线程也不能马上获得该对象锁，
>
> 要等到执行notify方法的线程将程序执行完 ，也就是退出sychronized代码块后，当前线程才会释放锁，
>
> 而呈wait状态所在的线程才可以获取该对象锁。

祖龙
2014/02/24 12:46上午

假设代码中去掉两个if判断，同时此时queue.size() == 0

考虑以下情况

1. 线程a先进入dequeue方法，发现size == 0就会wait

2. 然后线程b进入enqueue方法，发现size != limit，就会不停的add直到size == limit时wait

这个时候你会发现因为没有

if(this.queue.size() == 0) {

notifyAll();

}在size == 0 的时候去通知dequeue的线程a，a已经默默的一直在wait中都饿死了！

对于

if(this.queue.size() == this.limit) {

notifyAll();

}也是同理。

francisZ
2014/05/23 11:38上午

个人觉得这边有点问题

23 – 27行：

if(this.queue.size() == 0) {

notifyAll();

}

这个时候queue里面还是什么也没有，notifyAll()唤醒其他thread之后，这些thread从queue里面什么也拿不到，结果还是去wait，所以这个时候应该在往queue里面放入一个元素之后，再notifyAll吧。

43 – 47行也有同样的问题

阿凡提的哥
2014/05/28 7:53下午

“这个时候queue里面还是什么也没有，notifyAll()唤醒其他thread之后，这些thread从queue里面什么也拿不到，结果还是去wait”你这句话不对，调用notifyAll（ ）之后，线程并未释放锁，其他线程不会被唤醒，直到 执行这句 ：this.queue.add(item); 之后，其他线程才会被唤醒，但是同时只有一个线程能拿到锁，并正常执行，剩余线程（如果有的话）会因为 while(this.queue.size() == 0){

38

39 wait();

40

41 } 又继续等待

阿凡提的哥
2014/05/28 7:55下午

修正一下—“这个时候queue里面还是什么也没有，notifyAll()唤醒其他thread之后，这些thread从queue里面什么也拿不到，结果还是去wait”你这句话不对，调用notifyAll（ ）之后，线程并未释放锁，直到 执行这句 ：this.queue.add(item); 之后，其他线程才去重新获取锁，但是同时只有一个线程能拿到锁，并正常执行，剩余线程（如果有的话）会因为 while(this.queue.size() == 0){

38

39 wait();

40

41 } 又继续等

zxpbenson
2014/08/21 11:12上午

此阻塞队列的实现方式可能会引起饥饿

需要解决公平性的问题

是会引起性能损失的可优化点

Devry
2014/10/09 9:08下午

看一下源码，更加清楚明了。看看ReentrantLock和Condition的使用。

BlackLee
2015/10/23 2:58下午

实现的问题：

没有考虑公平性，会造成饥饿。

解决或者提升方法：

既然都使用队列存储了对象了，可以在唤醒是判断队列头对象是不是自己，

按时间序列实现公平性，避免饿死。

音无麻里亚
2016/11/26 3:55下午

其实代码里的notifyAll换成notify没有任何区别吧？无论如何notifyAll()唤醒的线程，能得到锁的只有一个，而且notifyAll反而因为竞争拉低性能吧？实在是不明白，请指教

airwalkers
2019/02/14 1:38下午

notify只能唤醒一个线程，消费者和生产者可能都是多个线程，如果消费者唤醒消费者，可能出现互相等待的情况，所以notifyAll是用来防止这种情况发生的