1,关于ReentrantLock的公平锁和非公平锁



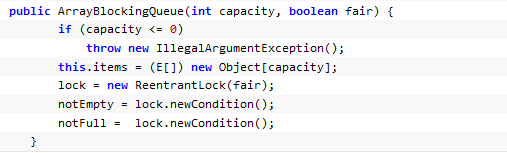
FairSync获取锁

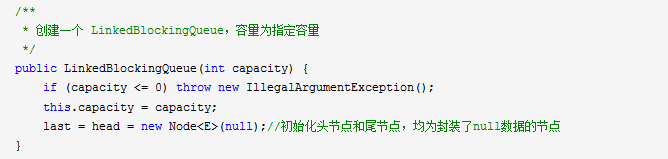


NonfairSync获取锁

ReentrantLock是基于AbstractQueuedSynchronizer的独占锁模式实现的，FairSync(公平锁同步器) 和NonfairSync (非公平锁同步器)和)都是AbstractQueuedSynchronizer的实现。在公平的锁上，线程按照他们发出请求的顺序获取锁，如果有另一个线程持有锁或者有其他线程在等待队列中等待这个所，那么新发出的请求的线程将被放入到队列中。在非公平锁上，则允许‘插队’：当一个线程请求非公平锁时，如果在发出请求的同时该锁变成可用状态，那么这个线程会跳过队列中所有的等待线程而获得锁，只有当锁被某个线程持有时，新发出请求的线程才会被放入队列中。因为FairSync 的tryAcquire(int acquires)多了需要判断当前线程是否在等待队列首部的逻辑，所以非公平锁性能略高于公平锁性能的。

2,关于ArrayBlockingQueue和LinkedBlockingQueue





ArrayBlockingQueue和LinkedBlockingQueue都是BlockingQueue的实现(另外两个实现是PriorityBlockingQueue和SynchronousQueue)。

ArrayBlockingQueue：规定大小的BlockingQueue,是基于数组实现的，其构造函数必须带一个int参数来指明其大小.其所含的对象是以FIFO排序的。

LinkedBlockingQueue：大小不定的BlockingQueue,是基于链表实现的，若其构造函数带一个规定大小的参数,生成的BlockingQueue有大小限制,若不带大小参数,所生成的BlockingQueue的大小由Integer.MAX\_VALUE来决定.其所含的对象是以FIFO排序的。

ArrayBlockingQueue插入和取出用的是同一个锁，构造参数可以指定一个布尔变量指定使用公平锁还是非公平锁。LinkedBlockingQueue实现的队列中的锁是分离的，内部使用ReentrantLock实现插入锁(putLock)和取出锁(takeLock)，因此可以让读写操作在不干扰对方的情况下，完成各自的功能，提高并发吞吐量。

LinkedBlockingQueue使用非公平锁，无法更改；

ArrayBlockingQueue可以指定使用公平锁还是非公平锁，默认是unfair

一般情况下，如果知道队列的大小，使用ArrayBlockIngQueue比较合适，因为它使用循环数组实现，会提前开辟内存空间。但是如果不知道队列未来的大小，那么使用ArrayBlockingQueue就必然会导致数组的来回复制，降低效率。但在使用LinkedBlockingQueue时，若用默认大小且当生产速度大于消费速度时候，有可能会内存溢出，因为Integer.MAX\_VALUE类似于无限大，也许还没有等到队列满阻塞产生，系统内存就有可能已被消耗殆尽了，需要注意。

3. 并发情况下，几种Queue的性能测试

并发length 次poll(),length 次 offer()

length = 100, 重复1000次取平均值

|  |  |  |
| --- | --- | --- |
| LinkedList | 0.5~0.7ms | 双向链表 |
| LinkedBlockingQueue | 0.5~0.7 ms | 单向链表(首尾指针) |
| ArrayBlockingQueue  ，非公平 | 0.5~0.7 ms | 循环数组 |
| ArrayBlockingQueue  ，公平 | 0.5~0.8 ms |  |
|  |  |  |

length = 100\*1000, 重复20次取平均值

|  |  |  |
| --- | --- | --- |
| LinkedList | 650~700 ms |  |
| LinkedBlockingQueue | 600~650 ms |  |
| ArrayBlockingQueue  ，非公平 | 650~700 ms |  |
| ArrayBlockingQueue  ，公平 | 650~700ms |  |
|  |  |  |

在并发场景下，几种queue的效率差异不明显

4. 串行情况下测试

Length=100\*1000 ，串行length次 add()和poll()

|  |  |  |
| --- | --- | --- |
| LinkedList | 14~15 ms |  |
| LinkedBlockingQueue | 56~57 ms |  |
| ArrayBlockingQueue  ，非公平 | 36~37 ms |  |
| ArrayBlockingQueue  ，公平 | 37~38ms |  |
|  |  |  |

（1）Array是事先构造好了对象，整体所需的所有引用都准备好了，使用中只是更改引用指向的位置； List在每次插入的时候，要给Queue里多一个对象，就要增加空间，所以会慢一点

（2）LinkedList也实现了Queue接口，因为没有加锁类的代码，所以更快

5.其它队列

Deque（双端队列）

可以当队列用，也可以当栈用。

应用例子：

1. A和B的任务分别在两头往中间填，一个人做完了，就替对方做

2. 原本是正常的队列，但是业务员特殊，有特例要优先执行，不排队。

ConcurrentLinkedQueue 🡪非阻塞队列，线程安全，用原子操作CAS实现

LinkedBlockingQueue 🡪 阻塞队列，线程安全，用加锁实现