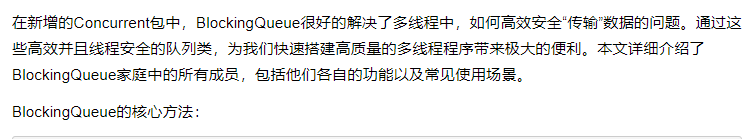
https://www.cnblogs.com/WangHaiMing/p/8798709.html

Blocking:阻塞

BlockingQueue:阻塞队列



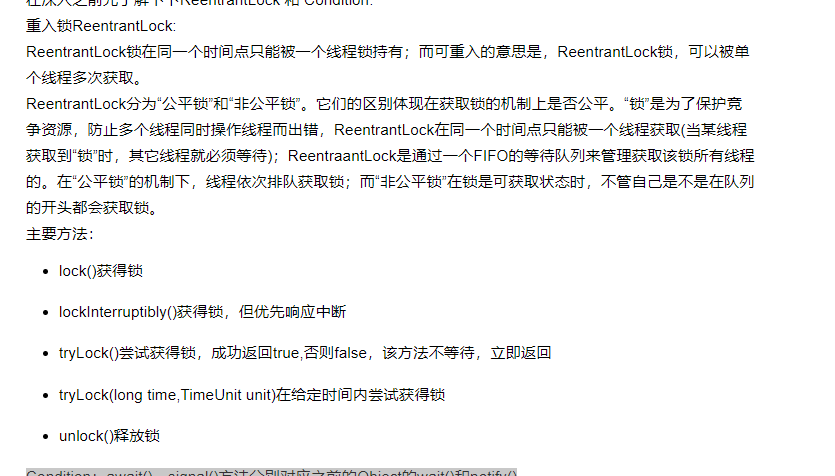
继承关系;



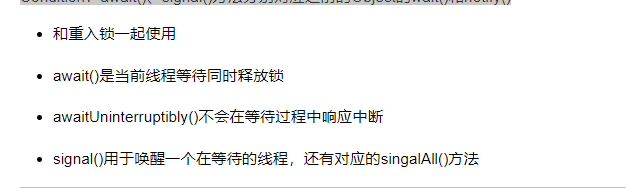
方法介绍:



重入锁ReentrantLock:



Condition：await()、signal()方法分别对应之前的Object的wait()和notify()







# 任务队列

## DelayQueue

相关的参数和方法查看本文档开头的地址.

## ArrayBlockingQueue

## LinkedBlockingQueue

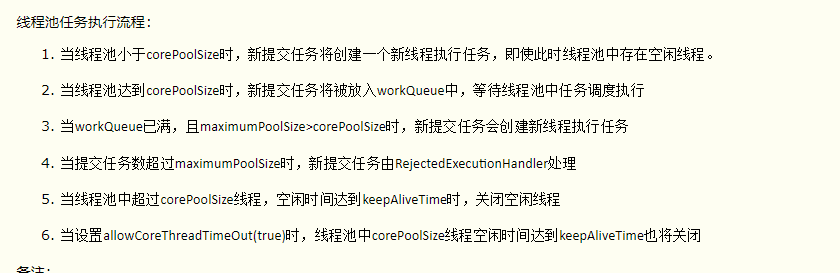
## 生产消费与BlockingQueue的关系



|  |
| --- |
| **package** blocking;  **import** java.util.concurrent.ArrayBlockingQueue;  **import** java.util.concurrent.BlockingQueue;  **public** **class** ProducterAndCustomer {  **public** **static** **void** main(String[] args)  {  **final** BlockingQueue<String> bq = **new** ArrayBlockingQueue<String>(10);  Runnable producerRunnable = **new** Runnable()  {  **int** i = 0;  **public** **void** run()  {  **while** (**true**)  {  **try**  {  System.***out***.println("我生产了一个" + i++);  bq.put(i + "");  Thread.*sleep*(1000);  }  **catch** (InterruptedException e)  {  e.printStackTrace();  }  }  }  };  Runnable customerRunnable = **new** Runnable()  {  **public** **void** run()  {  **while** (**true**)  {  **try**  {  System.***out***.println("我消费了一个" + bq.take());  Thread.*sleep*(3000);  }  **catch** (InterruptedException e)  {  e.printStackTrace();  }  }  }  };  Thread producerThread = **new** Thread(producerRunnable);  Thread customerThread = **new** Thread(customerRunnable);  producerThread.start();  customerThread.start();  }  } |

这里是用ArrayBlockingQueue保存生产者生产的内容,但是这个长度是10,刚开始的时候生产的效率高于消费者,但是等到这个ArrayBlockingQueue达到最大值的时候,只能是消费一个腾出空间,然后才能生产.

# 线程池是如何管理线程的



意思就是：corePoolSize和maximumPoolSize还有workQueue

首先线程根据corePoolSize判断是不是需要开启新的线程执行任务。

任务多的时候放在workQueue中进行等待，当workQueue中达到最大值以后，就再开启线程，但是不能超过maximumPoolSize。一旦达到最大值时，再有任务进来就执行RejectedExecutionHandler进行拒绝了。