/notifyAll();

/*
wait:
notify();
notifyAll();

都使用在同步中，因为要对持有监视器(锁)的线程操作。
所以要使用在同步中，因为只有同步才具有锁。
为什么这些操作线程的方法要定义 Object 类中呢？
因为这些方法在操作同步中线程时，都必须要标识它们所操作线程只有的锁，
只有同一个锁上的被等待线程，可以被同一个锁上 notify 唤醒。
不可以对不同锁中的线程进行唤醒。

也就是说，等待和唤醒必须是同一个锁。

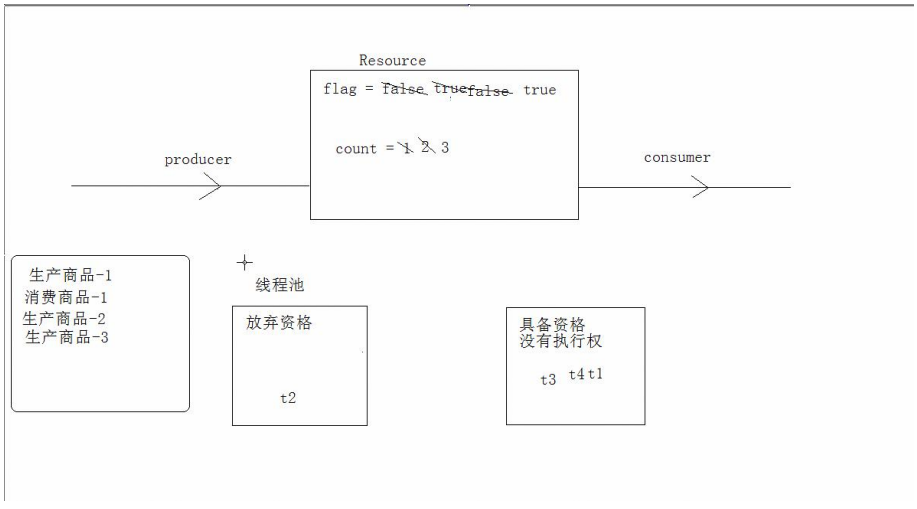
而锁可以是任意对象，所以可以被任意对象调用的方法定义 Object 类中。

*/

/*
对于多个生产者和消费者。
为什么要定义 while 判断标记。
原因：让被唤醒的线程再一次判断标记。

为什么定义 notifyAll，
因为需要唤醒对方线程。
因为只用 notify，容易出现只唤醒本方线程的情况。导致程序中的所有线程都等待。

*/



/*
JDK1.5 中提供了多线程升级解决方案。
将同步 Synchronized 替换成现实 Lock 操作。
将 Object 中的 wait，notify notifyAll，替换了 Condition 对象。
该对象可以 Lock 锁 进行获取。
该示例中，实现了本方只唤醒对方操作。

Lock:替代了 Synchronized
    lock
    unlock
    newCondition()

Condition：替代了 Object wait notify notifyAll
    await();
    signal();
    signalAll();
*/

/*
stop 方法已经过时。

如何停止线程？
只有一种，run 方法结束。
开启多线程运行，运行代码通常是循环结构。

只要控制住循环，就可以让 run 方法结束，也就是线程结束。

特殊情况：
当线程处于了冻结状态。
就不会读取到标记。那么线程就不会结束。

当没有指定的方式让冻结的线程恢复到运行状态是，这时需要对冻结进行清除。
强制让线程恢复到运行状态中来。这样就可以操作标记让线程结束。

Thread 类提供该方法 interrupt();
*/

//t1.setDaemon(true) 守护线程

*
join:
当 A 线程执行到了 B 线程的.join()方法时，A 就会等待。等 B 线程都执行完，A 才会执行。

join 可以用来临时加入线程执行。

*/