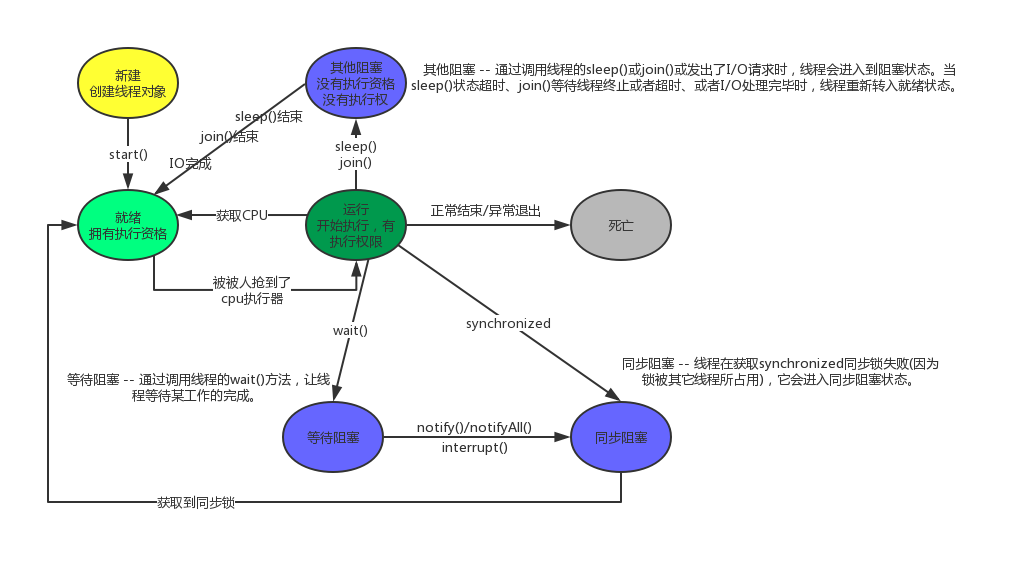
# 线程的生命周期：



接下来还是用经典的生产者和消费者的问题来解说：

先来解释一些概念：

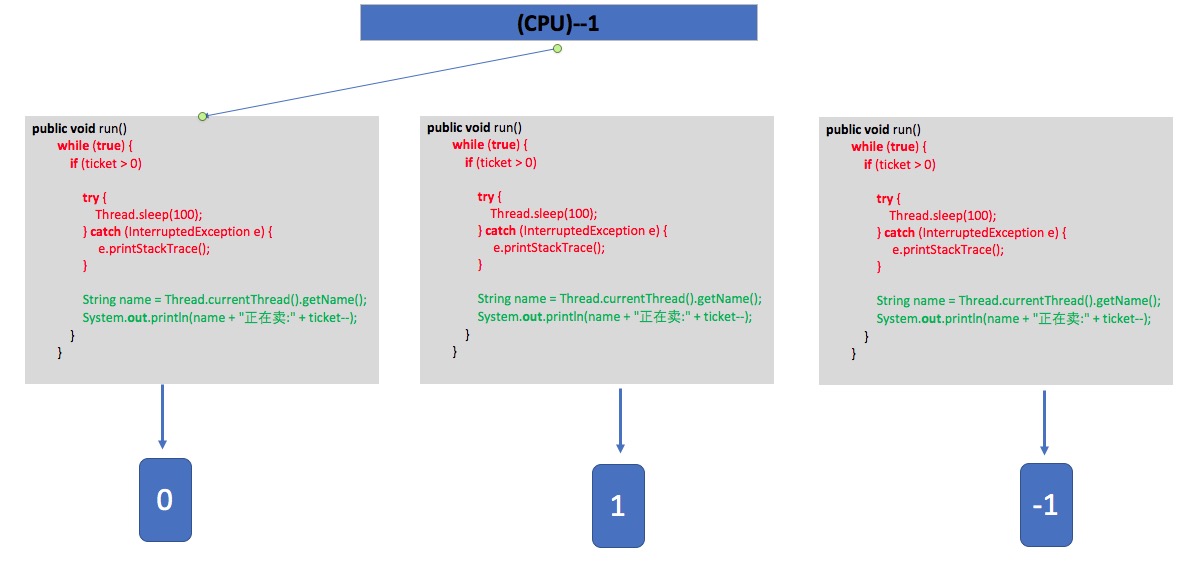
Object 的wait方法：让当前线程处于阻塞状态(等待...)，同时释放当前线程锁占用的锁。

必须要通过notify()才能唤醒这个阻塞的线程，继续进行cpu的争抢。

notify() 唤醒一个线程

notifyAll() 唤醒多个线程

==================================================================================

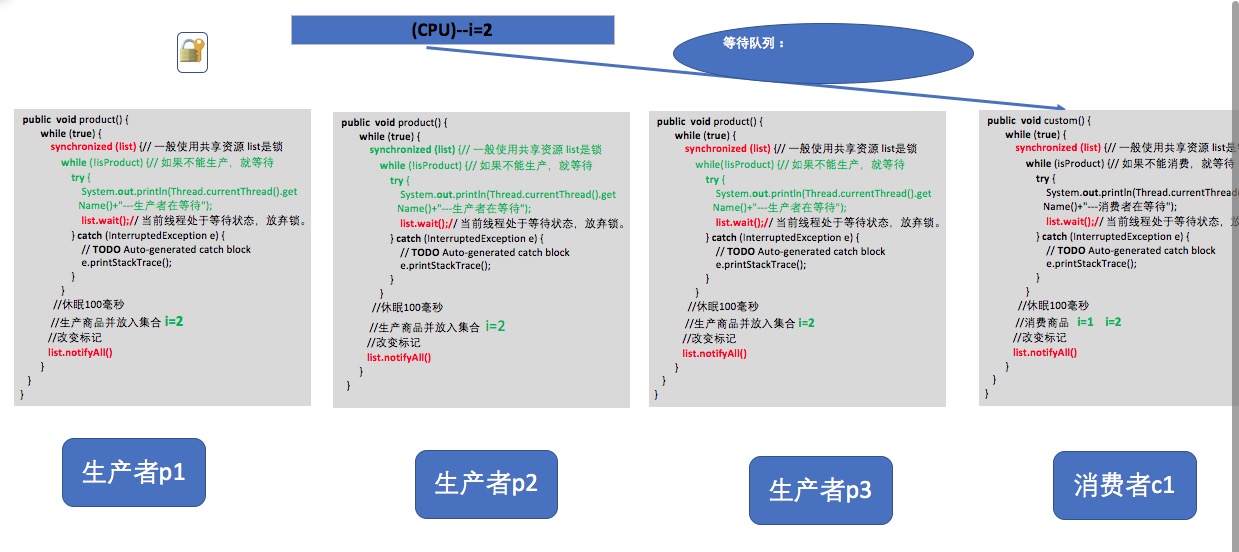


1号线程抢到cpu资源先执行，执行完上图红色部分进入了休眠，空置状态出现，2号3号开始争抢cpu；

以此类推，2号和3号也会执行完红色部分进入100毫秒的休眠。

此时，1号重新开始执行后面的绿色部分代码，这个时候会执行代码并打印“1号窗口卖出了第1张票”；按理说此时就应该停止，因为已经没有票可以再卖了。但是此时2号3号线程都是已经启动了无法关闭，在他们的休眠时间结束后，也会开始执行绿色部分的代码。打印输出“2号窗口卖出了第0张票”、“3号窗口卖出了第-1张票”，这显然是错的！

这种现象就是线程不安全导致的并发问题；

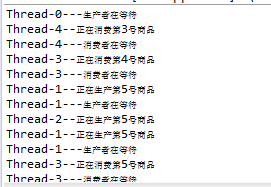


这里绿色部分代码的while循环判断如果换成了if循环判断的话，就会导致前面三个生产者进程都因为wait()等待的原因进入等待的状态，这个时候本来i=1，然后直到消费者线程抢到cpu开始运行并且执行了后面的改变标记以后再用ntifyAll()唤醒所有等待线程，前面的三个生产者线程不去再做判断而是直接往下执行，就会使得三个生产者线程都因为i++把原来得i=1都变成i=2然后输出。这样就出现了错误。

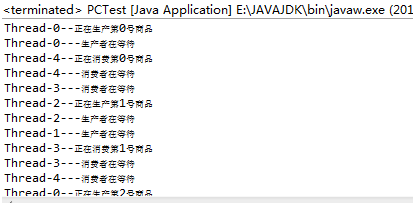
而使用了while以后，相当于在每次用ntifyAll()唤醒所有等待线程后都会让前面三个生产者线程再次执行判断，然后获得锁并执行，这样就不会出错！

执行结果如下：

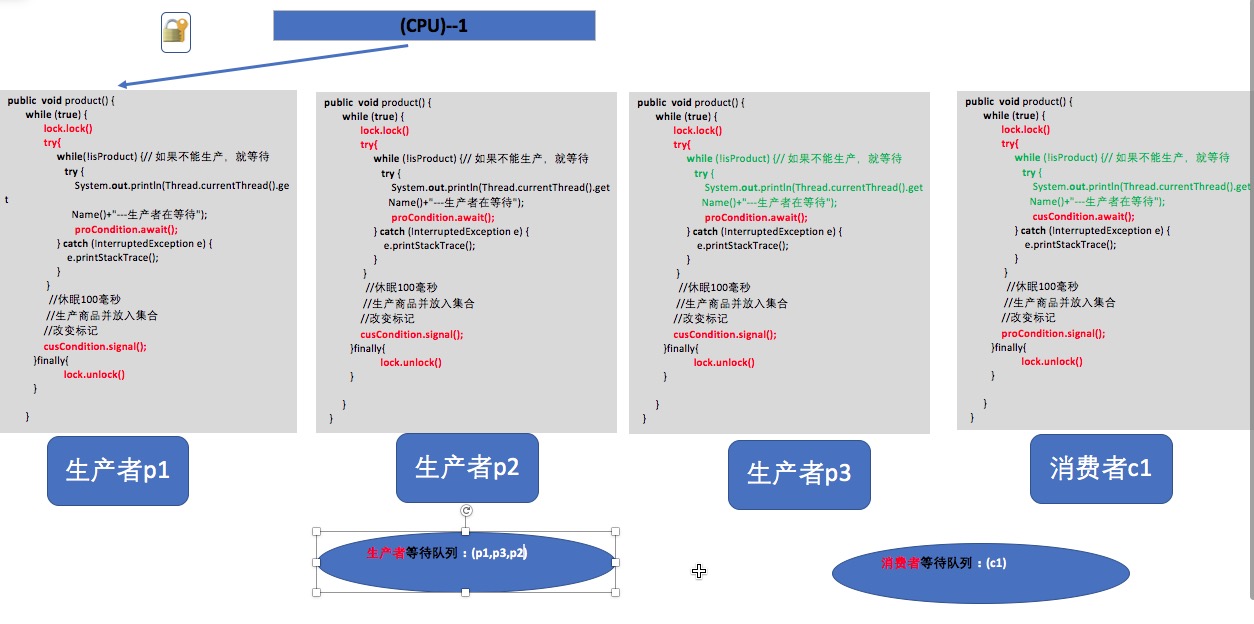
if执行结果：



while执行结果：



Lock锁



Lock锁

class TicketRannable2 implements Runnable{

private int ticket = 100;

Lock lock = new ReentrantLock();

@Override

public void run() {

//while(true) {

//同步块锁 锁（this）

//什么时候释放锁？块执行完就释放锁

//加锁

//lock.lock();//一个线程获取锁，其他线程就阻塞(等待锁的释放)

//lock.tryLock();

if(lock.tryLock()) {//一个线程获取锁，其他线程获取不到锁，就会放弃

try {

if(ticket>0) {

//模拟出票的耗时

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName()+"卖出第"+ticket--+"张票！！！");

}

} finally {

lock.unlock();//手动释放锁必须在finally中取释放锁，否则锁永远得不到锁死锁。

}

}else {

System.out.println(Thread.currentThread().getName()+"---放弃了锁");

}

//}

}

}

lock.unlock();手动释放锁最好写在finally块里面，这样就不会忘记写，因为为了不出现死锁的情况，那么解锁就千万不能忘记。

ReentantLock锁，await()，signal()方法,Condition类，tryLock和lock和lockInterruptibly的区别，读写锁

**await() 和 signall 的添加， 实际上是为我们提供了一种方便的基于同一个锁， 实现多个条件的 wait() 和 notify() 操作**

**await方法：**通过创建Condition对象来使线程wait，必须先执行lock.lock方法获得锁；

**signal方法：**condition对象的signal方法可以唤醒wait线程；

**Condition类和Object类：**

 Condition类的awiat方法和Object类的wait方法等效

 Condition类的signal方法和Object类的notify方法等效

 Condition类的signalAll方法和Object类的notifyAll方法等效