**生产者消费者问题**

等待进程被唤醒的时候从被等待的地方开始执行

进程等待和唤醒必须由加锁的那个对象来做

1. 消费者最后一次循环把生产者等死的问题

**问题原因**：

当最后一个消费者执行到最后一次的时候，假如有一个生产者还有2次。

消费者抢到执行权，发现没有货，于是等待。生产者开始执行，产品加一，然后唤醒消费者。消费者再次抢到执行权，从等待的地方开始执行，它执行不到else里面的内容，并没有唤醒等待的生产者，就直接结束了，并没有唤醒消费和唤醒生产者，然后循环结束了。在此时，所有消费者全部执行结束，但是还有生产者在等待消费者唤醒，所以就等死了。

**解决方案**：

把production()和consume()方法里的else拿掉，但是仍然出现其他问题

1. 虚假唤醒问题

**问题原因**：

一个生产者生产出一个商品，却唤醒了两个消费者，消费者被唤醒后，同时开始消费。

消费者开始执行，发现没有货，就等待了，又有一个消费者执行，也等待。生产者开始执行，生产一个商品，并唤醒所有消费者。两个消费者不管谁先抢到执行权，都是从等待的地方开始执行，然后就都开始消费商品，但商品只有一个，个数就变成了负数

**解决方案**：

把production()和consume()方法里的if换成while，每次等待被唤醒后，再次执行对商品个数的判定，如果商品个数小于等于0再次等待

1. 消费者已不再消费，生产者却依然在等待

**问题原因**：

消费者只消费10次，生产者却生产20次，消费者全部结束，生产者却依然还在等待生产。

**解决方案**：

还没有

//会出现两个问题，1.消费者最后一次执行把生产者等死了 2.虚假唤醒

**public** **class** TestProducterAndConsumer {

**public** **static** **void** main(String[] args) {

Market market = **new** Market(0, 1);

Producter producter = **new** Producter(market);

Consumer consumer = **new** Consumer(market);

**new** Thread(producter, "生产者1").start();

**new** Thread(producter, "生产者2").start();

**new** Thread(producter, "生产者3").start();

// new Thread(consumer, "消费者1").start();

// new Thread(consumer, "消费者2").start();

**new** Thread(consumer, "消费者3").start();

}

}

**class** Market {

**private** **int** count;

**private** **int** space;

**public** Market() {

**super**();

setCount(0);

setSpace(30);

}

**public** Market(**int** count, **int** space) {

**super**();

**this**.count = count;

**this**.space = space - count;

}

**public** **synchronized** **void** production() {

**if** (space <= 0) {

System.***out***.println(Thread.*currentThread*().getName() + "：没有仓位了，我等一下");

**try** {

**this**.wait();

} **catch** (InterruptedException e) {

// **TODO** 自动生成的 catch 块

e.printStackTrace();

}

} **else** {

--space;

++count;

**this**.notifyAll();

System.***out***.println(Thread.*currentThread*().getName() + "：生产一件商品 剩余仓位：" + space + " 剩余商品：" + count);

}

}

**public** **synchronized** **void** consume() {

**if** (count <= 0) {

System.***out***.println(Thread.*currentThread*().getName() + "：没有商品了，我等一下");

**try** {

**this**.wait();

} **catch** (InterruptedException e) {

// **TODO** 自动生成的 catch 块

e.printStackTrace();

}

} **else** {

++space;

--count;

**this**.notifyAll();

System.***out***.println(Thread.*currentThread*().getName() + "：消费一件商品 剩余仓位：" + space + " 剩余商品：" + count);

}

}

**public** **int** getCount() {

**return** count;

}

**public** **void** setCount(**int** count) {

**this**.count = count;

}

**public** **int** getSpace() {

**return** space;

}

**public** **void** setSpace(**int** space) {

**this**.space = space;

}

}

**class** Producter **implements** Runnable {

Market market;

**public** Producter(Market market) {

**super**();

**this**.market = market;

}

@Override

**public** **void** run() {

// **TODO** 自动生成的方法存根

**for** (**int** i = 0; i < 100; i++) {

**try** {

Thread.*sleep*(20);

} **catch** (InterruptedException e) {

// **TODO** 自动生成的 catch 块

e.printStackTrace();

}

market.production();

}

}

}

**class** Consumer **implements** Runnable {

Market market;

**public** Consumer(Market market) {

**super**();

**this**.market = market;

}

@Override

**public** **void** run() {

// **TODO** 自动生成的方法存根

**for** (**int** i = 0; i < 100; i++) {

**try** {

Thread.*sleep*(20);

} **catch** (InterruptedException e) {

// **TODO** 自动生成的 catch 块

e.printStackTrace();

}

market.consume();

}

}

}