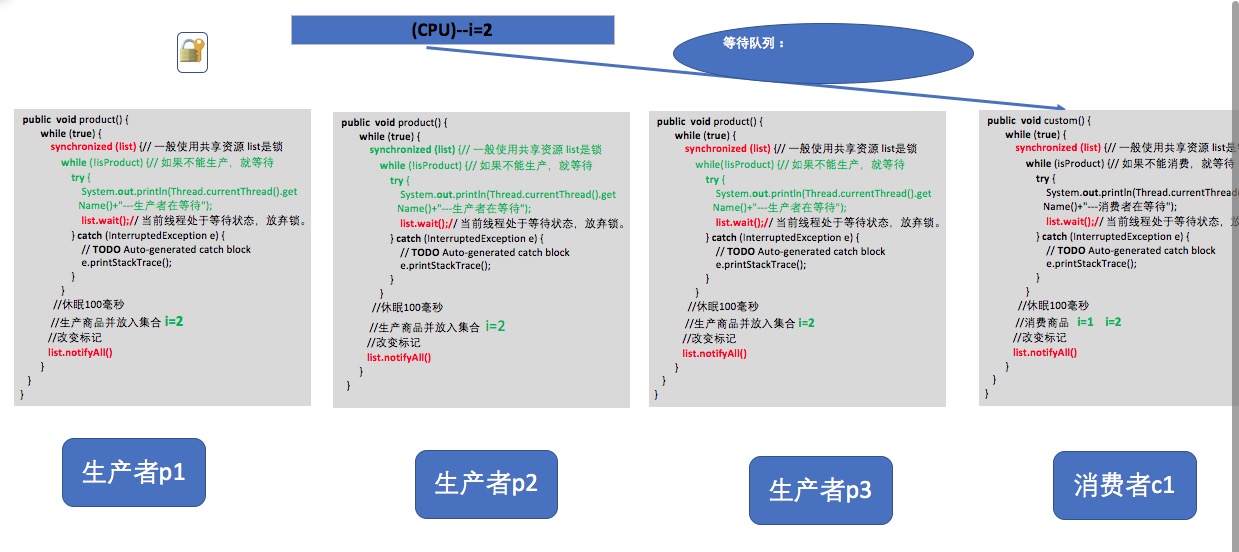
生产者消费者模式（synchronized）

1. 需要锁（这里使用synchronized）
2. 使用一个标记来控制生产和消费的行为
3. 使用Object.wait()使线程进入阻塞状态，使用notify()或者notifyAll()来唤醒被阻塞的线程



例子

**package** 生产者消费者模式;

**import** java.util.ArrayList;

/\*\*

\* **@author** mo

\* **@version** 2019年5月19日 下午3:17:08

\* **@description**

\*/

**class** Storage{

ArrayList lists=**new** ArrayList();

**int** i=0;

**boolean** isProduct=**true**;

**public** **void** product() {

**while**(**true**) {

**synchronized**(lists) {

**while**(!isProduct) {

**try** {

System.***out***.println(Thread.*currentThread*().getName()+"在等待");

lists.wait();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println(Thread.*currentThread*().getName()+"生产了第"+i+"件商品");

lists.add(**new** Integer(i));

isProduct=**false**;

lists.notifyAll();

}

}

}

**public** **void** comsume() {

**while**(**true**) {

**synchronized** (lists) {

**while**(isProduct) {

**try** {

System.***out***.println(Thread.*currentThread*().getName()+"在等待");

lists.wait();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println(Thread.*currentThread*().getName()+"消费了第"+i+"件商品");

lists.remove(**new** Integer(i));

i++;

isProduct=**true**;

lists.notifyAll();

}

}

}

}

**class** Producter **extends** Thread{

Storage storage;

**public** Producter(Storage storage) {

**this**.storage=storage;

}

@Override

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

// **TODO** Auto-generated method stub

storage.product();

}

}

**class** Consumer **extends** Thread{

Storage storage;

**public** Consumer(Storage storage) {

**this**.storage=storage;

}

@Override

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

// **TODO** Auto-generated method stub

storage.comsume();

}

}

**public** **class** Test01 {

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

Storage storage=**new** Storage();

Thread p1=**new** Producter(storage);

Thread p2=**new** Producter(storage);

Thread p3=**new** Producter(storage);

Thread c1=**new** Consumer(storage);

Thread c2=**new** Consumer(storage);

p1.start();

p2.start();

p3.start();

c1.start();

c2.start();

}

}

注：在条件判断处为什么要使用while？

因为上一个线程阻塞后，被唤醒的线程会接着lists.wait()后面开始运行，而不是从方法的开头开始运行，而且唤醒操作

可能会唤醒多于一个线程，这样就会出现生产者唤醒生产者的或者消费者唤醒消费者的情况，使用while循环判断condition条件，

可以避免这种假唤醒。