

生产者消费者模式（lcok锁）：

1. 需要锁（使用lock.lock()进行加锁,使用lock.unLock()去掉锁）
2. 使用condition.await()是线程进入阻塞状态
3. 使用condition.signal()唤醒线程
4. 有两个等待队列，一个生产者队列，一个消费者队列，生产者只唤醒消费者队列中的线程，同样消费者只唤醒生产者队列中的线程（FIFO双向列表）

注：锁必须释放掉，否则会产生死锁

实例：

**package** 生产者消费者模式;

/\*\*

\* **@author** mo

\* **@version** 2019年5月20日 下午7:05:58

\* **@description**

\*/

**import** java.util.ArrayList;

**import** java.util.concurrent.locks.Condition;

**import** java.util.concurrent.locks.Lock;

**import** java.util.concurrent.locks.ReentrantLock;

**class** Storage2{

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

**int** i=0;

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

Lock lock=**new** ReentrantLock();

Condition proCondition=lock.newCondition();

Condition cusCondition=lock.newCondition();

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

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

lock.lock();

**try** {

**while**(!isProduct) {

**try** {

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

proCondition.await();

} **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**;

cusCondition.signal();

} **finally** {

lock.unlock();

}

}

}

**public** **void** custom() {

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

lock.lock();

**while**(isProduct) {

**try** {

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

cusCondition.await();

} **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**;

proCondition.signal();

}

}

}

**class** Product2 **extends** Thread{

Storage2 storage2;

**public** Product2(Storage2 storage2) {

**this**.storage2=storage2;

}

@Override

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

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

storage2.product();

}

}

**class** Custom2 **extends** Thread{

Storage2 storage2;

**public** Custom2(Storage2 storage2) {

**this**.storage2=storage2;

}

@Override

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

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

storage2.custom();

}

}

**public** **class** Test02 {

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

Storage2 storage2=**new** Storage2();

Thread p1=**new** Product2(storage2);

Thread p2=**new** Product2(storage2);

Thread p3=**new** Product2(storage2);

Thread c1=**new** Custom2(storage2);

Thread c2=**new** Custom2(storage2);

Thread c3=**new** Custom2(storage2);

p1.start();

p2.start();

p3.start();

c1.start();

c2.start();

c3.start();

}

}