**线程并发管理**

示例：生产者与消费者问题，要生产者先行生产出了商品，才能够进行消费卖出的逻辑，如果没有生产出商品，那么就不能执行消费者行为；

定义类：

class Storage {

// 存放生产的 商品------集合

ArrayList list = new ArrayList();

int i = 0;// 商品的标号

// 定义标记---生产的标记 标记为true 表示可以生产，否则只能消费

boolean isProduct = true;

全代码导入：生产者行为

public void product() {

while (true) { //创造一个死循环，以便循环进行生产产品的行为

synchronized (list) {// 创造同步块锁一般锁的是共享资源，于是在这里list就是锁

while (!isProduct) {// 如果不能生产，就等待，此处为!isProduct，商品处于不可生产，可以消费的状态，于是先让生产者行为进入等待待唤醒状态。

try {

System.out.println(Thread.currentThread().getName()+"---生产者在等待");

list.wait();// wait()方法让当前线程进入等待状态，并且放弃锁，此时其他线程就会开始进行争抢。

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

try {

Thread.sleep(100);//生产过程，让其休眠一百毫秒执行产品的生产。

} catch (InterruptedException e) {

e.printStackTrace();

}

//输出生产产品的消息

System.out.println(Thread.currentThread().getName() + "--正在生产第" + i + "号商品");

list.add(new Integer(i));// 将生产的商品放到集合

// 生产完了改变标记，等到消费完这个商品才能继续生产商品

isProduct = false;

//唤醒所有等待的线程

list.notifyAll();//用notify()方法来唤醒等待的线程

}

}

}

全代码导入：消费者行为

public void custom() {

while (true) {

synchronized (list) {

while (isProduct) {// 如果不能消费 就等待此处为isProduct，商品处于可生产，不可以消费的状态，于是先让消费者行为进入等待待唤醒状态。

try {

System.out.println(Thread.currentThread().getName()+"---消费者在等待");

list.wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

//输出结果消费产品的消息

System.out.println(Thread.currentThread().getName() + "--正在消费第" + i + "号商品"); // 将生产的商品消费掉

list.remove(new Integer(i));

//每次卖出产品以后，产品编号要改变，从而计算卖出了多少商品，用i++。

i++;

// 消费完了改变标记，等到生产下一个才能继续消费

isProduct = true;

//唤醒一个等待的线程

list.notifyAll();

}

}

}

生产者线程：

class Producter extends Thread {//继承Thread类来写线程

private Storage storage;

//构造方法

public Producter(Storage storage) {

this.storage = storage;

}

//重写run()方法

@Override

public void run() {

storage.product();

}

}

消费者线程：

class Coustomer extends Thread {//同上

private Storage storage;

public Coustomer(Storage storage) {

this.storage = storage;

}

@Override

public void run() {

storage.custom();

}

}

执行运行的程序：

public class PCTest {

public static void main(String[] args) {

Storage s = new Storage();

Thread p1 = new Producter(s); //生产者线程1

Thread p2 = new Producter(s); //生产者线程2

Thread p3 = new Producter(s); //生产者线程3

Thread c1 = new Coustomer(s); //消费者线程1

Thread c2 = new Coustomer(s); //消费者线程2

//执行所有线程

p1.start();

p2.start();

p3.start();

c1.start();

c2.start();

}

}

【思考题】if(isProduct) 改为 while(isProduct) 就能解决问题

因为只执行一次判断的话是面对单一生产者和单一消费者的情况的，在有多个生产者消费者的多线程情况下，每一次一个线程的执行前都应该要进行一次判断。

Object类中的方法：wait()、notify()和notifyAll()

1.public final native void notify();

2.public final native void notifyAll();

3.public final native void wait(longtimeout) throws InterruptedException;

从这三个方法的文字描述可以知道以下几点信息：

　　1）wait()、notify()和notifyAll()方法是本地方法，并且为final方法，无法被重写。

　　2）调用某个对象的wait()方法能让当前线程阻塞，并且当前线程必须拥有此对象的monitor（即锁）

　　3）调用某个对象的notify()方法能够唤醒一个正在等待这个对象的monitor的线程，如果有多个线程都在等待这个对象的monitor，则只能唤醒其中一个线程；

　　4）调用notifyAll()方法能够唤醒所有正在等待这个对象的monitor的线程；