需求：模拟面包生产过程，使用notify方法和wait方法让双方进行通信协调。

分析：1.创建一个面包类

2.创建一个容器类，含有一个面包的数组，能为生产的面包添加标签能加入生产的面包，移除出售的面包。

3.创建生产者线程，含有一个容器实例，run()中当面包余量大于5时，就唤醒消费线程，并阻塞该线程，否则就生产，并加入容器。

4.创建消费者线程，含有一个容器实例，run()中当面包余量小于1时，就唤醒生产线程，并阻塞该线程，否则就消费，并移除容器。

源码：

面包类：

**package** xhu.edu.net.Bread;

**public** **class** Bread {

**private** String chefString;

**private** String numString;

**public** Bread(String chefString, String numString) {

**super**();

**this**.chefString = chefString;

**this**.numString = numString;

}

**public** String getChefString() {

**return** chefString;

}

**public** **void** setChefString(String chefString) {

**this**.chefString = chefString;

}

**public** String getNumString() {

**return** numString;

}

**public** **void** setNumString(String numString) {

**this**.numString = numString;

}

@Override

**public** String toString() {

**return** "厨师：" + chefString + ", 编号" + numString;

}

}

容器类：

**package** xhu.edu.net.Bread;

/\*

\* 问题：是采用实时创建线程好点还是采用唤醒线程好点？

\* 比如说这里如果我们采用容器统一进行控制，到一定数量的时候进行创建线程，执行相应的操作。

\* 那生产者线程和消费者者线程就不用进行大量的循环判断了。

\*/

**import** java.util.List;

//容器类作为一个通知类，有一定数量的时候就进行通知消费者售卖，少于一定的时候就通知生产者进行生产。并对每个新生的面包进行编号

**public** **class** Container {

**private** List<Bread> breadList;

**private** **int** id = 0;

**public** Container(List<Bread> breadList) {

**super**();

**this**.breadList = breadList;

}

//获取容器中面包的余量

**public** **int** breadMargin() {

**return** breadList.size();

}

//添加面包

**public** **void** add(Bread bread) {

bread.setNumString(breadNum());

breadList.add(bread);

System.***out***.println("生产：" + bread.toString());

}

//移除一个面包

**public** **void** remove() {

Bread bread = breadList.remove(0);

System.***out***.println("消费：" + bread.toString());

}

//为面包加标签

**private** String breadNum() {

id++;

**return** "xsf-ml:xo" + id;

}

}

生产线程：

**package** xhu.edu.net.Bread;

**public** **class** Producer **implements** Runnable {

Container container;

**public** Producer(Container container) {

**super**();

**this**.container = container;

}

@Override

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

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

**synchronized** (container) {

// 如果容器余量大于0就阻塞生产线程(这个地方如果没有同步锁，那就会出现线程该醒不醒该等不等)

**int** num = container.breadMargin();

**if** (num >= 5) {

**try** {

container.notifyAll();// 唤醒消费线程，但是问题也会唤醒生产者线程，那这样会浪费大量的时间和性能，如何来避免这个问题？

container.wait();

} **catch** (InterruptedException e) {

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

e.printStackTrace();

}

} **else** {

container.add(**new** Bread(Thread.*currentThread*().getName(), **null**));

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e1) {

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

e1.printStackTrace();

}

}

}

}

}

}

消费线程：

**package** xhu.edu.net.Bread;

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

Container container;

**public** Consumer(Container container) {

**super**();

**this**.container = container;

}

@Override

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

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

**synchronized** (container) {

**int** num = container.breadMargin();

**if** (num < 1) {

container.notifyAll();// 如果余量小于1，就阻塞消费线程唤醒生产线程

**try** {

container.wait();

} **catch** (InterruptedException e) {

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

e.printStackTrace();

}

} **else** {

System.***out***.print(Thread.*currentThread*().getName() + " ");

container.remove();

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e1) {

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

e1.printStackTrace();

}

}

}

}

}

}

测试类：

**package** xhu.edu.net.Bread;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** Test {

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

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

List<Bread> list = **new** ArrayList<Bread>();

Container container = **new** Container(list);

Producer producer = **new** Producer(container);

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

// 开启生产线程

**new** Thread(producer, "张氏").start();

**new** Thread(producer, "黎氏").start();

**new** Thread(producer, "明氏").start();

**new** Thread(producer, "谢氏").start();

// 开启消费线程

**new** Thread(consumer, "李铭").start();

**new** Thread(consumer, "明亮").start();

**new** Thread(consumer, "噬恶").start();

**new** Thread(consumer, "可达").start();

**new** Thread(consumer, "爱卿").start();

}

}

运行截图：

