2、生产者-消费者模式的三种实现方式

1、背景

生产者生产数据到**缓冲区**中，消费者从**缓冲区**中取数据。

如果**缓冲区**已经满了，则生产者线程阻塞；

如果**缓冲区**为空，那么消费者线程阻塞。

2、方式一：synchronized、wait和notify

```
1 package producerConsumer;
2 //wait 和 notify
3 public class ProducerConsumerWithWaitNofity {
4     public static void main(String[] args) {
5         Resource resource = new Resource();
6         //生产者线程
7         ProducerThread p1 = new ProducerThread(resource);
8         ProducerThread p2 = new ProducerThread(resource);
9         ProducerThread p3 = new ProducerThread(resource);
10        //消费者线程
11        ConsumerThread c1 = new ConsumerThread(resource);
12        //ConsumerThread c2 = new ConsumerThread(resource);
13        //ConsumerThread c3 = new ConsumerThread(resource);
14
15        p1.start();
16        p2.start();
17        p3.start();
18        c1.start();
19        //c2.start();
20        //c3.start();
21    }
22
23
24
25 }
26 /**
27  * 公共资源类
28  * @author
29  *
30  */
31 class Resource{//重要
32     //当前资源数量
33     private int num = 0;
34     //资源池中允许存放的资源数目
35     private int size = 10;
36
37     /**
38      * 从资源池中取走资源
39      */
40     public synchronized void remove(){
41         if(num > 0){
42             num--;
43             System.out.println("消费者" + Thread.currentThread().getName() +
44                     "消耗一件资源，" + "当前线程池有" + num + "个");
45             notifyAll();//通知生产者生产资源
46         }else{
47             try {
48                 //如果没有资源，则消费者进入等待状态
49                 wait();
50                 System.out.println("消费者" + Thread.currentThread().getName() + "线程进入等
```

公告

昵称：fankongkong
园龄：1年11个月
粉丝：5
关注：1
+加关注

‹ 2018年4月 ›

日	一	二	三	四	五	六
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

剑指offer(45)
排序(9)
Android开发艺术探索(9)
多线程(6)
设计模式(6)
算法(5)
集合(4)
Android(4)
Java基础(4)
Java虚拟机(3)
更多

随笔分类

Android基础(18)
Android项目(4)
Java基础(32)
Java源码(5)
笔试面试
笔试算法题(1)
操作系统(1)
常用设计模式(4)
常用算法(65)
大数据
多线程(6)
机器学习
计算机网络(1)
历史

```
待状态");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        /**
         * 向资源池中添加资源
         */
        public synchronized void add(){
            if(num < size){
                num++;
                System.out.println(Thread.currentThread().getName() + "生产一件资源，当前资源池有"
                + num + "个");
                //通知等待的消费者
                notifyAll();
            }else{
                //如果当前资源池中有10件资源
                try{
                    wait();//生产者进入等待状态，并释放锁
                    System.out.println(Thread.currentThread().getName()+"线程进入等待");
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        }
    }
    /**
     * 消费者线程
     */
    class ConsumerThread extends Thread{
        private Resource resource;
        public ConsumerThread(Resource resource){
            this.resource = resource;
        }
        @Override
        public void run() {
            while(true){
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                resource.remove();
            }
        }
    }
    /**
     * 生产者线程
     */
    class ProducerThread extends Thread{
        private Resource resource;
        public ProducerThread(Resource resource){
            this.resource = resource;
        }
        @Override
        public void run() {
            //不断地生产资源
            while(true){
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                resource.add();
            }
        }

    }
```

3、方式二：lock和condition的await、signalAll

```
package producerConsumer;

```


随笔档案

最新评论

1. Re:2、生产者-消费者模式的三种实现方式
@924249386可以的，昨天在修改论文，没注意。...
--fankongkong

2. Re:2、生产者-消费者模式的三种实现方式
哈哈，答案已找到，如果run 方法中不加入while(true) 的话，run方法运行结束后，线程会结束，加入的话，run方法会持续进行，加入适当的条件，会退出无限循环的
--924249386

3. Re:2、生产者-消费者模式的三种实现方式
朋友，你好，发现方式一中有个小问题类ConsumerThread和类ProducerThread中的run 方法，加while(true) {}和不加有不同的运行结果，加while(true) {}程.....
--924249386

阅读排行榜

```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
/**
 * 使用Lock 和 Condition解决生产者消费者问题
 * @author tangzhijing
 *
 */
public class LockCondition {
        public static void main(String[] args) {
            Lock lock = new ReentrantLock();
            Condition producerCondition = lock.newCondition();
            Condition consumerCondition = lock.newCondition();
            Resource2 resource = new Resource2(lock,producerCondition,consumerCondition);

            //生产者线程
            ProducerThread2 producer1 = new ProducerThread2(resource);

            //消费者线程
            ConsumerThread2 consumer1 = new ConsumerThread2(resource);
            ConsumerThread2 consumer2 = new ConsumerThread2(resource);
            ConsumerThread2 consumer3 = new ConsumerThread2(resource);

            producer1.start();
            consumer1.start();
            consumer2.start();
            consumer3.start();
        }
}
/**
 * 消费者线程
 */
class ConsumerThread2 extends Thread{
    private Resource2 resource;
    public ConsumerThread2(Resource2 resource){
        this.resource = resource;
        //setName("消费者");
    }
    public void run(){
        while(true){
            try {
                Thread.sleep((long) (1000 * Math.random()));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            resource.remove();
        }
    }
}
/**
 * 生产者线程
 * @author tangzhijing
 *
 */
class ProducerThread2 extends Thread{
    private Resource2 resource;
    public ProducerThread2(Resource2 resource){
        this.resource = resource;
        setName("生产者");
    }
    public void run(){
        while(true){
                try {
                    Thread.sleep((long) (1000 * Math.random()));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                resource.add();
        }
    }
}
/**
 * 公共资源类
 * @author tangzhijing
 *
 */
class Resource2{
    private int num = 0;//当前资源数量
    private int size = 10;//资源池中允许存放的资源数目
    private Lock lock;
```


评论排行榜

推荐排行榜

```
    private Condition producerCondition;
    private Condition consumerCondition;
    public Resource2(Lock lock, Condition producerCondition, Condition consumerCondition)
{
        this.lock = lock;
        this.producerCondition = producerCondition;
        this.consumerCondition = consumerCondition;

    }
    /**
     * 向资源池中添加资源
     */
    public void add(){
        lock.lock();
        try{
            if(num < size){
                num++;
                System.out.println(Thread.currentThread().getName() +
                        "生产一件资源,当前资源池有" + num + "个");
                //唤醒等待的消费者
                consumerCondition.signalAll();
            }else{
                //让生产者线程等待
                try {
                    producerCondition.await();
                    System.out.println(Thread.currentThread().getName() + "线程进入等待");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }finally{
            lock.unlock();
        }
    }
    /**
     * 从资源池中取走资源
     */
    public void remove(){
        lock.lock();
        try{
            if(num > 0){
                num--;
                System.out.println("消费者" + Thread.currentThread().getName()
                        + "消耗一件资源," + "当前资源池有" + num + "个");
                producerCondition.signalAll();//唤醒等待的生产者
            }else{
                try {
                    consumerCondition.await();
                    System.out.println(Thread.currentThread().getName() + "线程进入等待");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }//让消费者等待
            }
        }finally{
            lock.unlock();
        }
    }

}
```

4、方式三：BlockingQueue

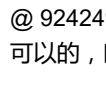
```
package producerConsumer;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

//使用阻塞队列BlockingQueue解决生产者消费者
public class BlockingQueueConsumerProducer {
    public static void main(String[] args) {
        Resource3 resource = new Resource3();
        //生产者线程
        ProducerThread3 p = new ProducerThread3(resource);
        //多个消费者
        ConsumerThread3 c1 = new ConsumerThread3(resource);
```

```
        ConsumerThread3 c2 = new ConsumerThread3(resource);
        ConsumerThread3 c3 = new ConsumerThread3(resource);

        p.start();
        c1.start();
        c2.start();
        c3.start();
    }
}
/**
 * 消费者线程
 * @author tangzhijing
 *
 */
class ConsumerThread3 extends Thread {
    private Resource3 resource3;

    public ConsumerThread3(Resource3 resource) {
        this.resource3 = resource;
        //setName("消费者");
    }

    public void run() {
        while (true) {
            try {
                Thread.sleep((long) (1000 * Math.random()));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            resource3.remove();
        }
    }
}
/**
 * 生产者线程
 * @author tangzhijing
 *
 */
class ProducerThread3 extends Thread{
    private Resource3 resource3;
    public ProducerThread3(Resource3 resource) {
        this.resource3 = resource;
        //setName("生产者");
    }

    public void run() {
        while (true) {
            try {
                Thread.sleep((long) (1000 * Math.random()));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            resource3.add();
        }
    }
}
class Resource3{
    private BlockingQueue resourceQueue = new LinkedBlockingQueue(10);
    /**
     * 向资源池中添加资源
     */
    public void add(){
        try {
            resourceQueue.put(1);
            System.out.println("生产者" + Thread.currentThread().getName()
                    + "生产一件资源," + "当前资源池有" + resourceQueue.size() +
                    "个资源");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    /**
     * 向资源池中移除资源
     */
    public void remove(){
        try {
            resourceQueue.take();
            System.out.println("消费者" + Thread.currentThread().getName() +
                    "消耗一件资源," + "当前资源池有" + resourceQueue.size()
                    + "个资源");
```

```
94          } catch (InterruptedException e) {
95              e.printStackTrace();
96          }
97      }
98  }
```

分类: Java基础 , 常用设计模式

标签: 多线程 , 设计模式



fankongkong
关注 - 1
粉丝 - 5
+加关注

2 0

« 上一篇：堆排序
» 下一篇：死锁

posted @ 2017-08-20 12:10 fankongkong 阅读(6258) 评论(3) 编辑 收藏

评论列表

#1楼 2018-03-22 23:13 **924249386**

朋友，你好，发现方式一中有个小问题
类ConsumerThread和类ProducerThread中的run 方法，加while(true) {}和不加有不同的运行结果，加while(true) {}程序会正常运行，不加的话程序会提前结束。这个while(true) {}有什么玄机吗?

支持(0) 反对(0)

#2楼 2018-03-22 23:23 **924249386**

哈哈，答案已找到，如果run 方法中不加入while(true) 的话，run方法运行结束后，线程会结束，加入的话，run方法会持续进行，加入适当的条件，会退出无限循环的

支持(1) 反对(0)

#3楼[楼主] 2018-03-23 12:18 **fankongkong**

@ 924249386
可以的，昨天在修改论文，没注意。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

» 更多知识库文章...