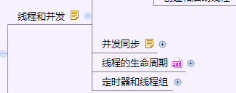
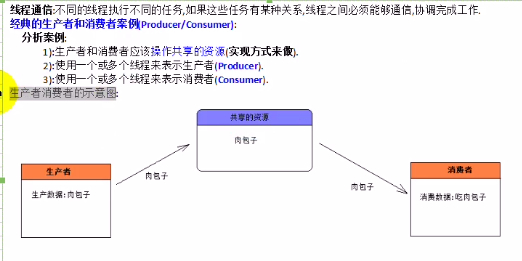
Day 15





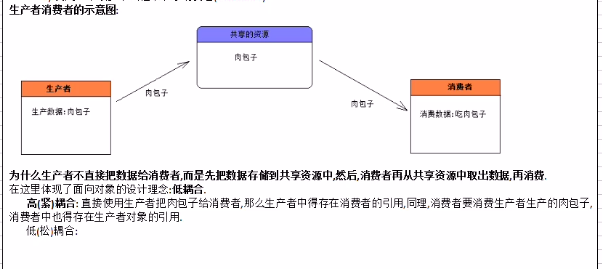
**01线程通信**



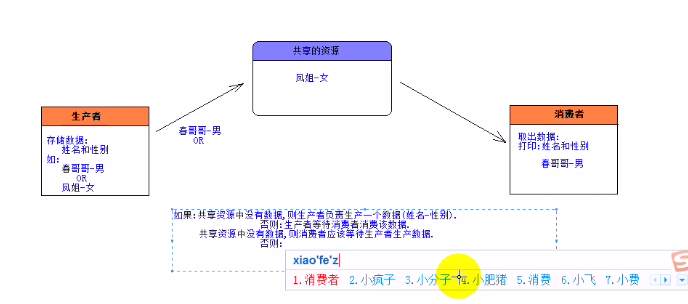


如kafka的生产者和消费者，共享的资源是消息队列中的消息；

之所以生产者和消费者之间使用共享资源来交互是为了低耦合，使用一个中间对象来屏蔽两者之间的交互，从而达到低耦合，否则生产者对象需要包含消费者的引用，而消费者的对象中也要包含生产者的引用。







消息队列：中间件

**02线程通信-生产者和消费者案例**

**//共享资源类**

package aboutthread.producerandconsumer;  
  
/\*\*  
 \* Created by DaiYan on 2017/4/23.  
 \*/  
public class SharedResource {  
 private String name;  
 private String gender;  
  
 public void push(String name,String gender){  
 this.name=name;

try{  
 Thread.sleep(10);  
}catch(InterruptedException e){  
 e.printStackTrace();  
}

this.gender=gender;  
 }  
  
 public void popup(){

try{  
 Thread.sleep(10);  
}catch(InterruptedException e){  
 e.printStackTrace();  
}

System.out.println(this.name+"-"+this.gender);  
 }  
}

//生产者类

package aboutthread.producerandconsumer;  
  
/\*\*  
 \* Created by DaiYan on 2017/4/23.  
 \*/  
public class Producer implements Runnable{  
 private SharedResource product;  
 public Producer(SharedResource res){  
 this.product=res;  
 }  
  
 public void run(){  
 for (int i = 0; i <50 ; i++) {  
 if(i%2==0)  
 product.push("春哥哥","男");  
 else  
 product.push("凤姐","女");  
 }  
 }  
}

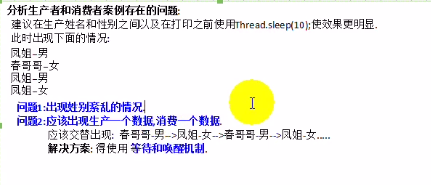
//消费者类

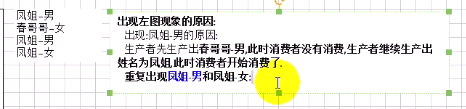
package aboutthread.producerandconsumer;  
  
/\*\*  
 \* Created by DaiYan on 2017/4/23.  
 \*/  
public class Consumer implements Runnable{  
 private SharedResource consumption;  
 public Consumer(SharedResource res){  
 this.consumption=res;  
 }  
  
 public void run(){  
 for (int i = 0; i <50 ; i++) {  
 this.consumption.popup();  
 }  
 }  
}

**//测试代码**

package aboutthread.producerandconsumer;  
  
/\*\*  
 \* Created by DaiYan on 2017/4/23.  
 \*/  
public class TestDemo {  
 public static void main(String[] args){  
 //创建共享的资源对象  
 SharedResource res=new SharedResource();  
  
 //启动生产者线程  
 new Thread(new Producer(res)).start();  
  
 //启动消费者线程  
 new Thread((new Consumer(res))).start();  
 }  
}

**03线程通信-解决性别稳定问题**

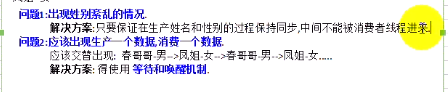




问题1的解决方案：同步代码块和同步方法或者使用锁机制来保存同步性；使得生产姓名和性别同步。

同步代码块/同步方法或者锁机制来保持同步性

问题2的解决方案：使用等待和唤醒



同步锁池：

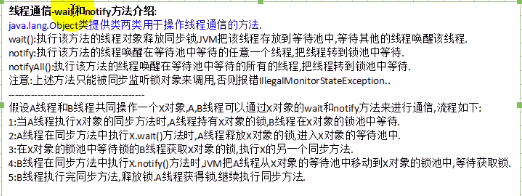
同步锁必须选择多个线程共同的资源的对象

当前生产者在生产数据的时候（先拥有同步锁）， 其他线程就在锁池中等待获取锁。

当线程执行完同步代码块的时候，就会释放同步锁，其他线程就会抢锁对象。

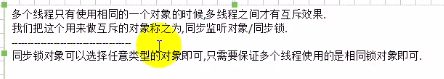
在某个对象上面锁只有一把

**04线程通信-wait和notify方法介绍**



锁池和等待池：等待池中的线程没有同步监听器，需要唤醒才能到锁池，才能抢锁。

监听对象拥有锁池和等待池。



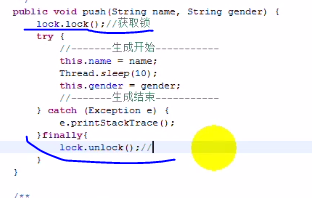
同步锁对象需要保证是相同的锁对象，也只有同步锁对象才能调用wait和notify方法

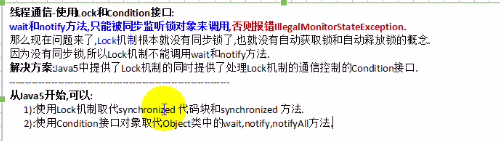


**05线程通信-解决重复生成或重复消费问题**public class SharedResource {  
 private String name;  
 private String gender;  
 private boolean isEmpty=true;//用来表示对象是否已经被生产或消费  
  
 synchronized public void push(String name,String gender){  
 if(!this.isEmpty)  
 try {  
 this.wait();  
 }catch(InterruptedException e){  
 e.printStackTrace();  
 //log  
 }  
  
 this.name=name;  
 try{  
 Thread.sleep(10);//当前线程等待，其他线程能够操作  
 }catch(InterruptedException e){  
 e.printStackTrace();  
 }  
 this.gender=gender;  
 this.isEmpty=false;  
 this.notify();  
 }  
  
 synchronized public void popup(){  
 if(isEmpty)  
 try {  
 this.wait(); //使用同步锁对象来调用，当前线程释放锁，进入到等待池中等待唤醒  
 }catch(InterruptedException e){  
 e.printStackTrace();  
 //log  
 }  
 try{  
 Thread.sleep(10);  
 }catch(InterruptedException e){  
 e.printStackTrace();  
 }  
 System.out.println(this.name+"-"+this.gender);  
 this.isEmpty=true;  
 this.notify();//使用当前同步锁来调用，唤醒其他的锁，当前线程和其他锁进入到锁池中等待jvm调用  
 }  
}

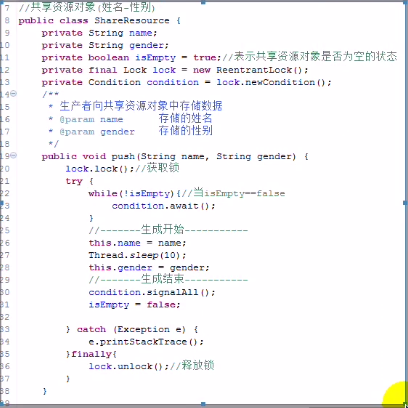
因为没有同步锁所以lock不能调用wait和notify方法

**06线程通信-使用lock和condition接口**

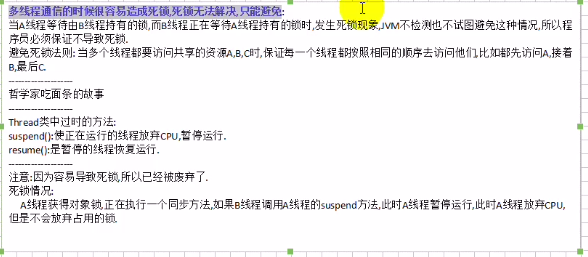


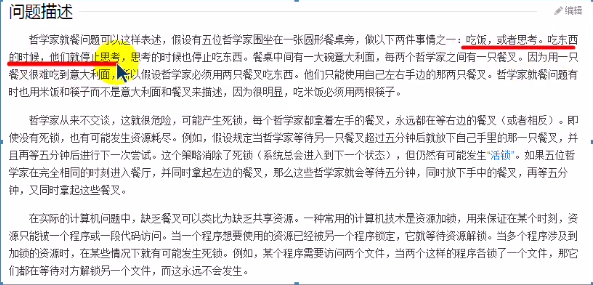


Await和signal、signalAll



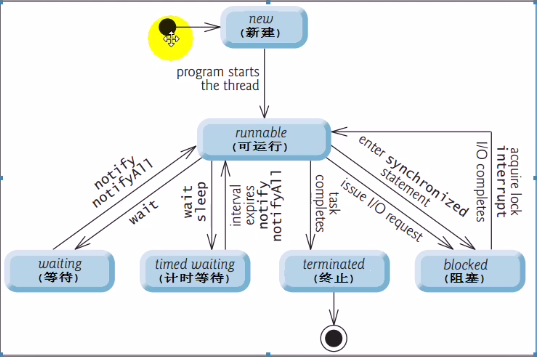
**07线程通信-死锁问题**

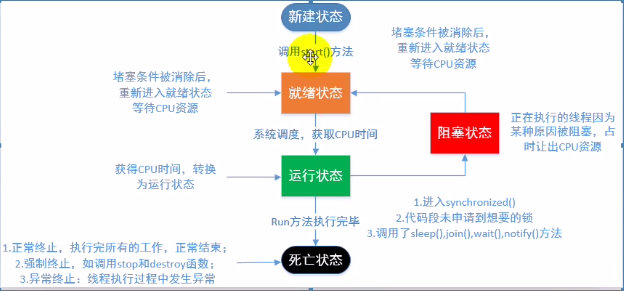


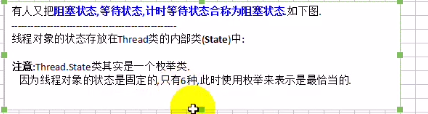


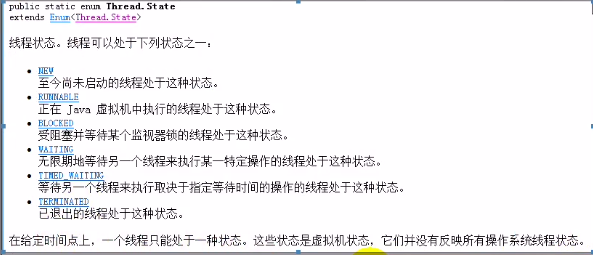
**08线程的生命周期和状态**



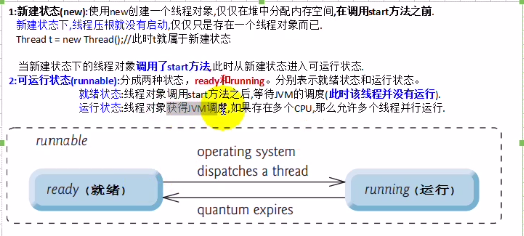




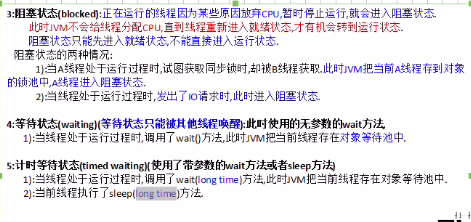


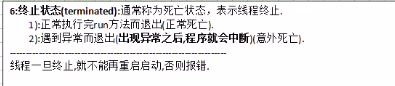


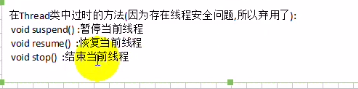
**09生命周期中的各个状态**



线程对象的start方法只能调用一次

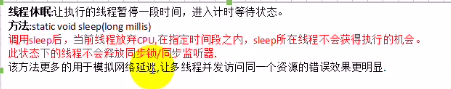




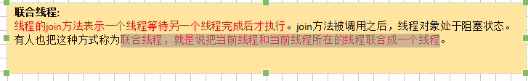


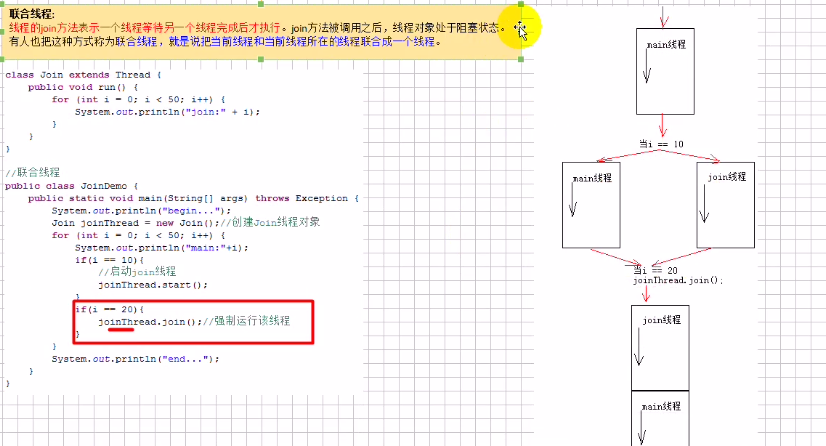
10线程的操作-线程睡眠





**11联合线程**





12后台线程

