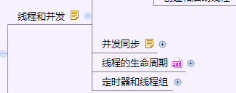
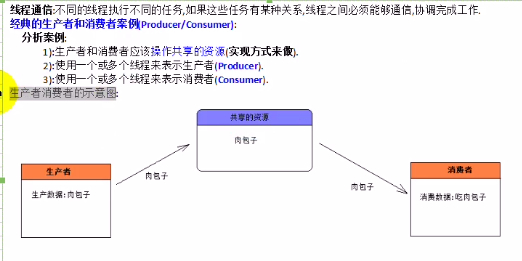
Day 15





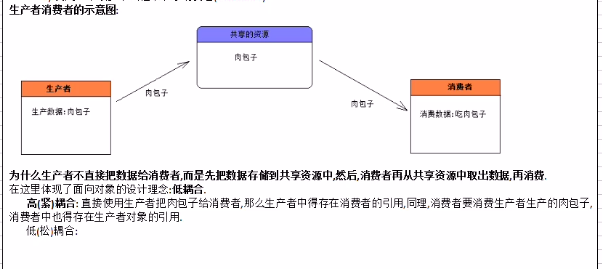
**01线程通信**



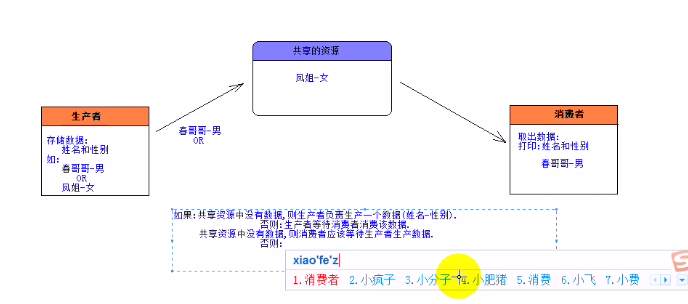


如kafka的生产者和消费者，共享的资源是消息队列中的消息；

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");  
 }  
 }  
}

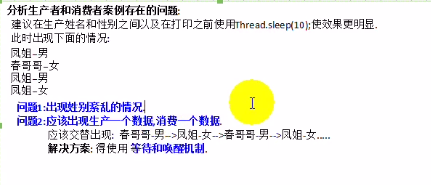
//消费者类

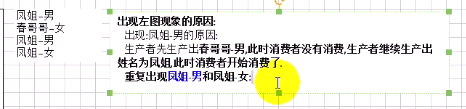
package aboutthread.producerandconsumer;  
  
/\*\*  
 \* Created by DaiYan on 2017/4/23.  
 \*/  
public class Consumer implements Runnable{  
 private SharedResource consumption;  
 public Consumer(SharedResource res){  
 this.consumption=res;  
 }  
  
 public void run(){  
 for (int i = 0; i <50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

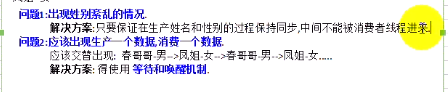




问题1的解决方案：同步代码块和同步方法或者使用锁机制来保存同步性；使得生产姓名和性别同步。

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



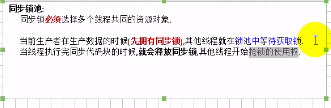
同步锁池：

同步锁必须选择多个线程共同的资源的对象

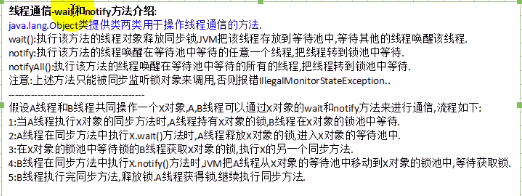
当前生产者在生产数据的时候（先拥有同步锁）， 其他线程就在锁池中等待获取锁。

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

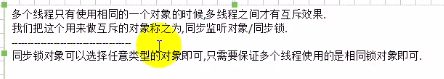


**04线程通信-wait和notify方法介绍**



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。



同步锁对象可以选择任意类型的对象，只需要保证多个线程使用的是相同的锁对象即可，

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  
 }  
}

**或者**

package com.test.demo;

/\*\*

\* Created by DaiYan on 2017/9/10.

\*/

public class SharedResource {

private String name;

private String gender;

private boolean notEmpty=false;

synchronized public void push(String name,String gender) {

try {

if (!notEmpty) {

this.name = name;

Thread.sleep(10);

this.gender = gender;

this.notEmpty = true;

this.notify();

this.wait();

}

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

synchronized public void pop() {

try {

if (notEmpty) {

Thread.sleep(10);

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

this.notEmpty = false;

this.notify();

this.wait();

}

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getGender() {

return gender;

}

public void setGender(String gender) {

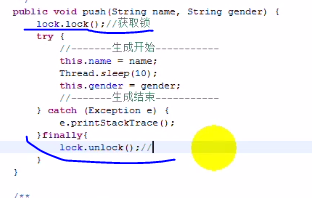
this.gender = gender;

}

}

因为没有同步锁所以lock不能调用wait和notify方法

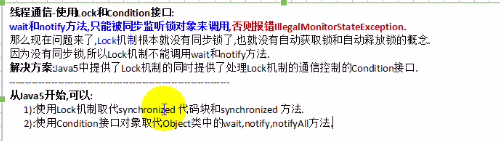
**06线程通信-使用lock和condition接口**



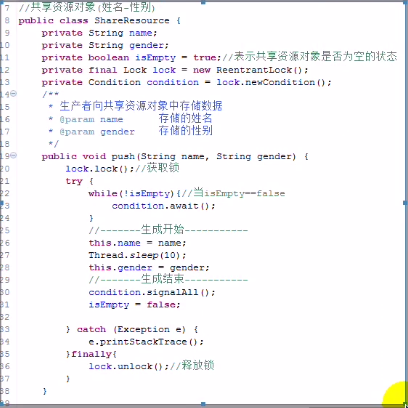
Wait和notify只能被同步监听锁对象来调用

Lock没有同步锁，所以lock机制下不能调用wait和notify方法

从java1.5开始，可以lock的机制的Condition类

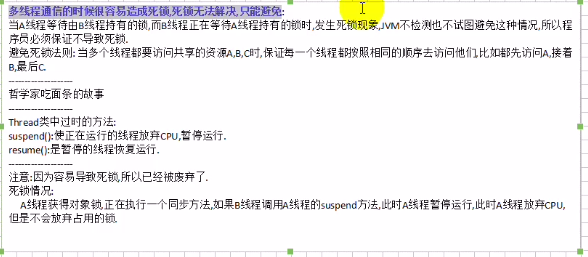


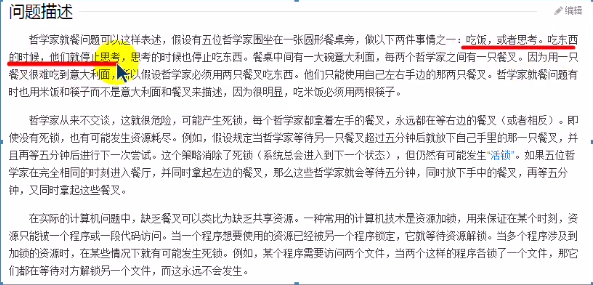
Await和signal、signalAll



**07线程通信-死锁问题**

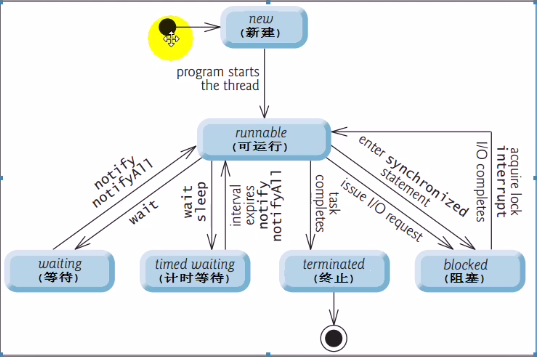
死锁只能避免，无法消除；死锁一点发生则线程就回长期处于阻塞状态

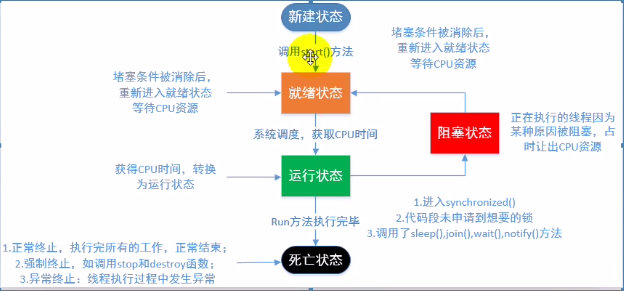


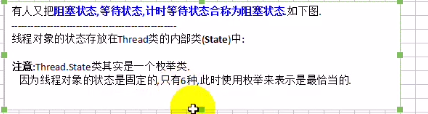


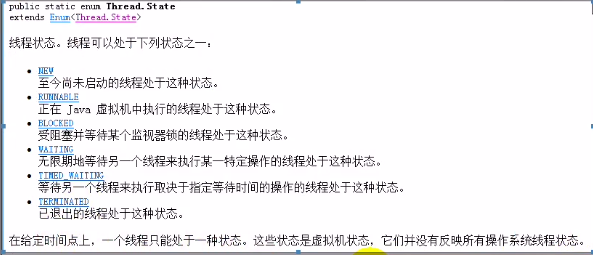
**08线程的生命周期和状态**



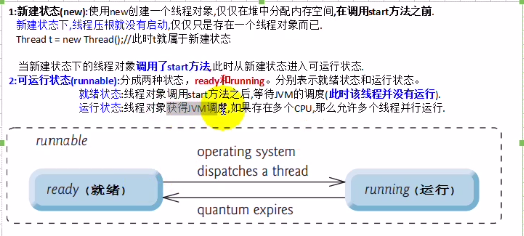




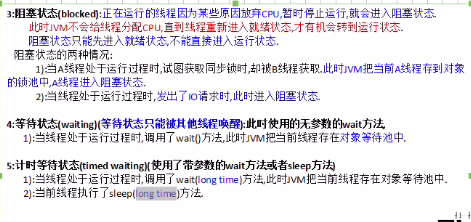




**09生命周期中的各个状态**



线程对象的start方法只能调用一次，即Thread对象只能调用一次start方法。



阻塞状态：

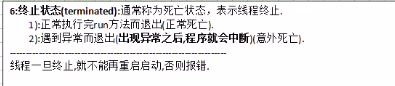
1. 未获取到锁，处于同步锁池中
2. 线程发出IO请求，处于阻塞状态

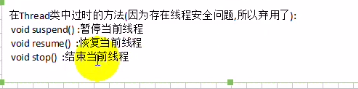
Wait方法（不带参数）会释放同步锁，而且只能被其他线程唤醒；

Wait方法（待参数）会等待一定时间后自动进入到可运行状态；

而sleep方法不会释放锁

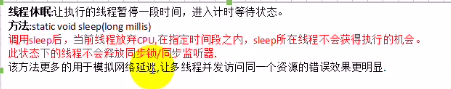
isAlive判断当前线程是否处于活跃状态，否则为终止状态。



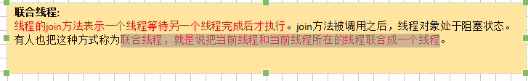


**10线程的操作-线程睡眠**

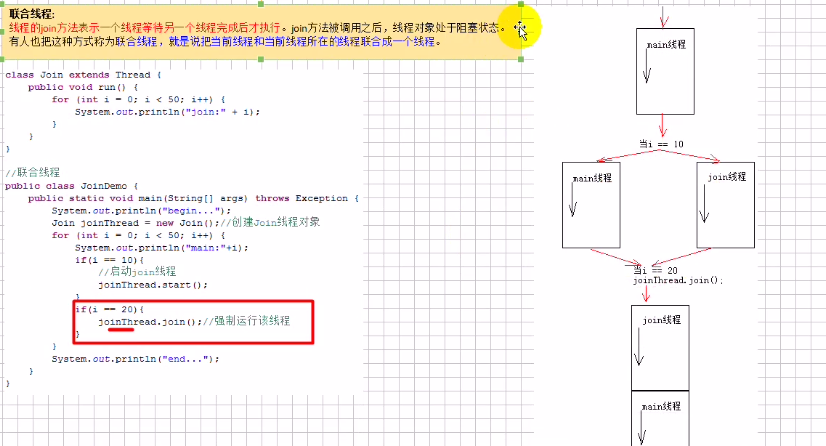


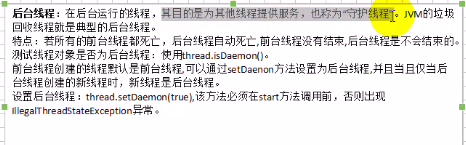


**11联合线程**



哪一个线程调用join方法，则不会释放资源直到该线程执行结束。



**12 java核心内库-线程-线程的操作-后台线程**

Java中至少有两个线程，main主线程和垃圾线程gc

setDaemon设置为后台进程

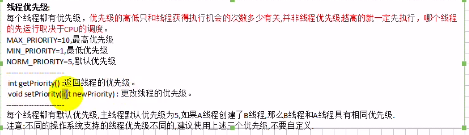
isDaemon判断是否为后台线程

前台线程创建的进程默认也是前台线程，后台线程创建的线程也是后台线程；

前台线程创建的线程可以设置为后台线程，线程要先设置后启动。

前台线程终止后，后台线程自动终止运行。

**13 java核心内库-线程-线程的操作-线程优先级**

****

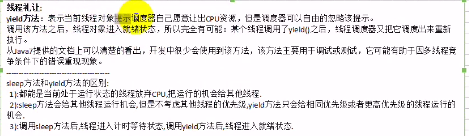
优先级越高知只是有更大的机会被jvm调度

优先级的范围为1-10，默认为5

线程优先级可以先启动后设置

子线程默认获得父线程的优先级，可以更改

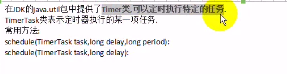
**14 java核心内库-线程-线程的操作-线程礼让**

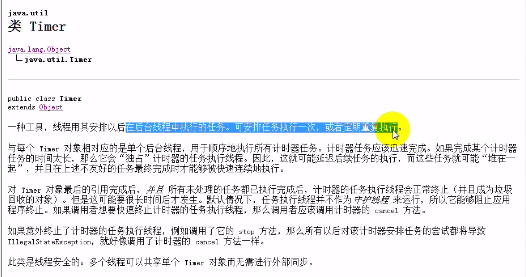
****

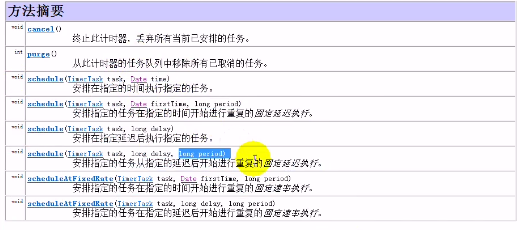
调用yield方法后，线程自动进入就绪状态，可能还会被jvm调度，开发中很少使用该方法，仅用于调试和测试。

Sleep方法被调用后进入计时等待，让出CPU，持有锁的话不会释放；计时结束后进入就绪状态

**15.java核心内库-线程-定时器和线程组**







继承TimerTask类，是一个线程

线程组：

