# 线程的生产者消费者模式🡪设计模式

信号灯法！

## 原因

用在了线程同步方法中

防止对同一资源使用锁的时候造成死锁

## 一些对比

wait 方法 --> 暂停了线程，不过这个时候它把锁释放了 是属于java包方法

sleep方法 --> 暂停了线程，不过并没有释放锁，相当于抱着锁睡觉了 属于线程包方法

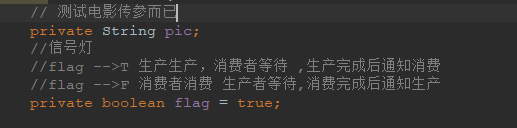
## 举个例子解释 🡪看电影与播放电影

这里还用到了标识法来中断线程 （并没有结束）

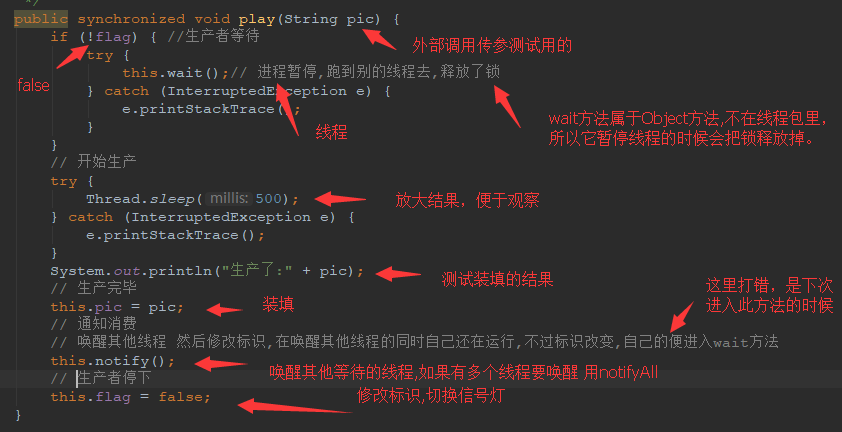
### 定义电影类

#### 创建一个信号灯标识

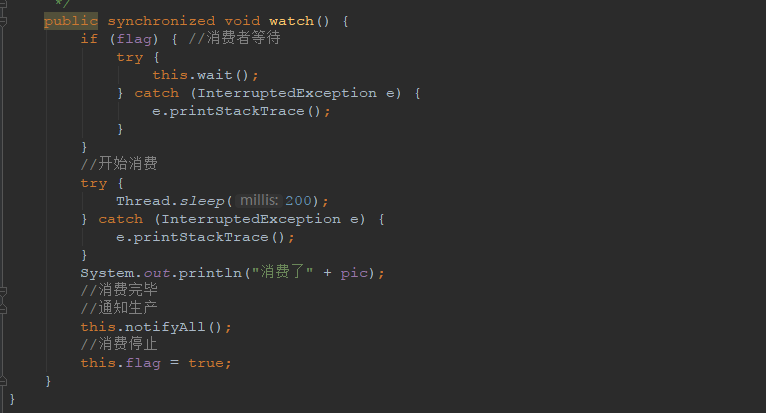
private boolean flag = true;



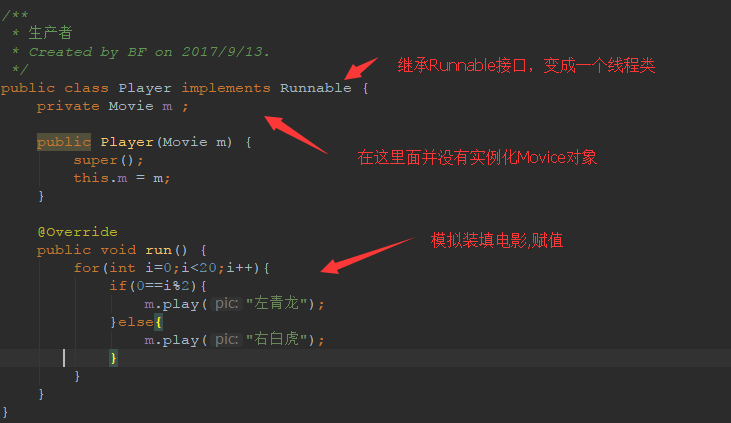
### 创建装填电影方法🡪生产者



### 创建播放电影的点发->消费者



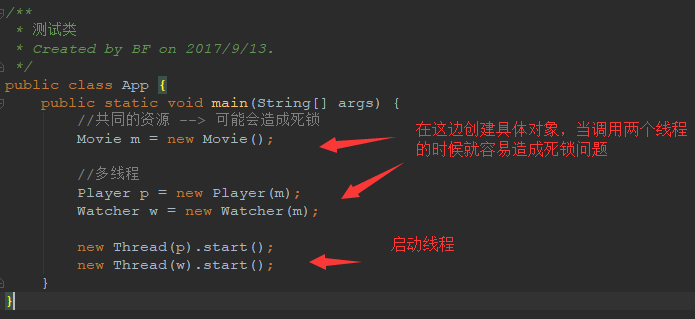
## 创建装填电影的人 🡪 生产者



## 创建看电影的人 🡪 消费者



## 创建测试类：



# 源码

## Movice

/\*\*  
 \* Created by BF on 2017/9/13.  
 \* 一个场景,共同的资源  
 \* 生产者消费者模式 信号灯法  
 \* wait() :等待，释放锁   sleep 不释放锁  
 \* notify()C:\Users\BF\AppData\Local\Temp\OOY5$4OW5H`8`9%(9$)T67M.giftifyAll():唤醒  
 \* 与 synchronized 共同使用,不可单独使用  
 \*/  
public class Movie {  
    // 测试电影传参而已  
    private String pic;  
    //信号灯  
    //flag -->T 生产生产，消费者等待 ,生产完成后通知消费  
    //flag -->F 消费者消费 生产者等待,消费完成后通知生产  
    private boolean flag = true;  
  
    /\*\*  
     \* 模拟装填 --> 生产者  
     \*  
     \* @param pic 外部传参  
     \*/  
    public synchronized void play(String pic) {  
        if (!flag) { //生产者等待  
            try {  
                this.wait();// 进程暂停,跑到别的线程去,释放了锁  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
        // 开始生产  
        try {  
            Thread.sleep(500);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        System.out.println("生产了:" + pic);  
        // 生产完毕  
        this.pic = pic;  
        // 通知消费  
        // 唤醒其他线程 然后修改标识,在唤醒其他线程的同时自己还在运行,不过标识改变,自己的便进入wait方法  
        this.notify();  
        // 生产者停下  
        this.flag = false;  
    }  
  
    /\*\*  
     \* 模拟播放 --> 消费者  
     \*/  
    public synchronized void watch() {  
        if (flag) { //消费者等待  
            try {  
                this.wait();  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
        //开始消费  
        try {  
            Thread.sleep(200);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        System.out.println("消费了" + pic);  
        //消费完毕  
        //通知生产  
        this.notifyAll();  
        //消费停止  
        this.flag = true;  
    }  
}

## Player

/\*\*

\* 生产者

\* Created by BF on 2017/9/13.

\*/

public class Player implements Runnable {

private Movie m ;

public Player(Movie m) {

super();

this.m = m;

}

@Override

public void run() {

for(int i=0;i<20;i++){

if(0==i%2){

m.play("左青龙");

}else{

m.play("右白虎");

}

}

}

}

## Watcher

/\*\*

\* 消费者

\* Created by BF on 2017/9/13.

\*/

public class Watcher implements Runnable {

private Movie m ;

public Watcher(Movie m) {

super();

this.m = m;

}

@Override

public void run() {

for(int i=0;i<20;i++){

m.watch();

}

}

}

## 测试类

/\*\*

\* 测试类

\* Created by BF on 2017/9/13.

\*/

public class App {

public static void main(String[] args) {

//共同的资源 --> 可能会造成死锁

Movie m = new Movie();

//多线程

Player p = new Player(m);

Watcher w = new Watcher(m);

new Thread(p).start();

new Thread(w).start();

}

}