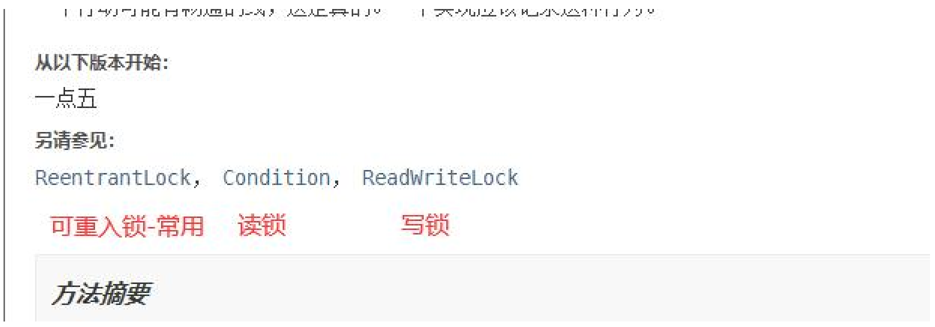
链滴

Lock 锁重点（Juc-02）

Lock接口

```
Lock l = ...;
l.lock();
try {
  // access the resource protected by this lock
} finally {
  l.unlock();
}
```



[illegible]

从以下版本开始:

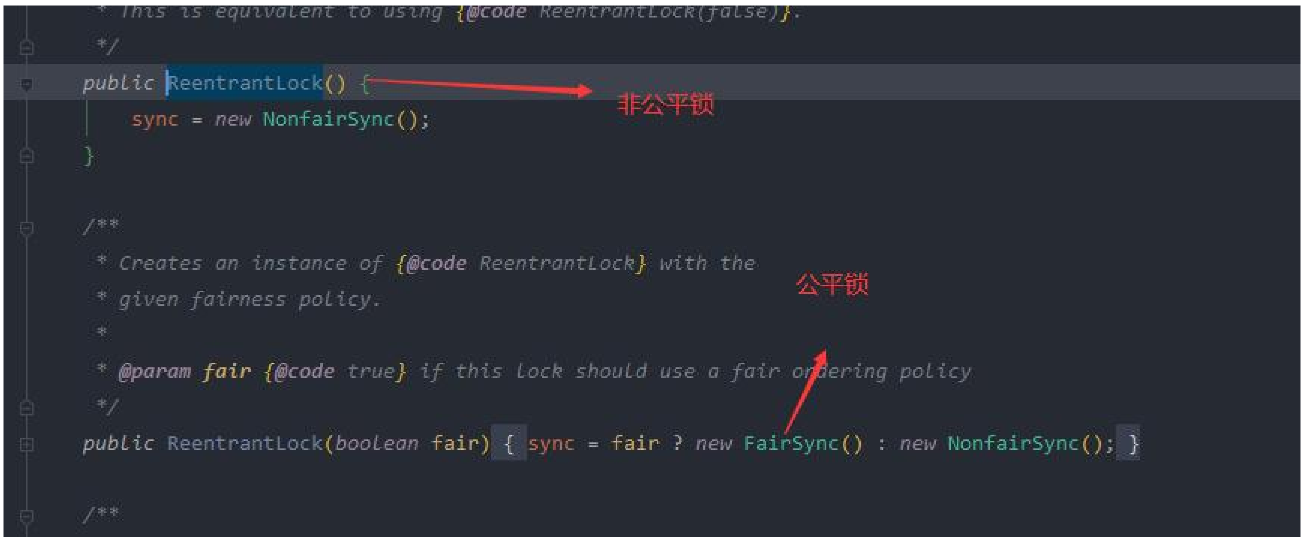
一点五

另请参见:

ReentrantLock, Condition, ReadWriteLock

可重入锁-常用　读锁　　　写锁

方法摘要



```
     * This is equivalent to using {@code ReentrantLock(false)}.
     */
    public ReentrantLock() {
        sync = new NonfairSync();
    }

    /**
     * Creates an instance of {@code ReentrantLock} with the
     * given fairness policy.
     *
     * @param fair {@code true} if this lock should use a fair ordering policy
     */
    public ReentrantLock(boolean fair) { sync = fair ? new FairSync() : new NonfairSync(); }

    /**
```

公平锁：十分的公平，可以先来后到

非公平锁：十分不公平，可以插队 (默认)

```
public class SaleTicketDemo02 {
    public static void main(String[] args) {
        Ticket2 ticket = new Ticket2();
        new Thread(()->{for (int i = 0; i <60 ; i++) ticket.sale(); },"A").start();
        new Thread(()->{for (int i = 0; i <60 ; i++) ticket.sale(); },"B").start();
        new Thread(()->{for (int i = 0; i <60 ; i++) ticket.sale(); },"C").start();
    }
}
/*
* 1、new ReentrantLock()
* 2.lock.lock() 加锁
* 3、 finally=>lock.unlock() 解锁
* */
//资源类OOP
class Ticket2 {
    private int number = 50;
    Lock lock = new ReentrantLock();
    public void sale() {
        lock.lock();
        try {
            if(number>0){
                System.out.println(Thread.currentThread().getName()+"买到了第："+number-- +"票"
;
            }
        }catch (Exception e){
        }finally {
            lock.unlock();
        }
    }
}
```

Synchronized和Lock区别

- synchronized 内置的java的关键子，Lock是一个Java类

- synchronized 无法判断获取所得状态，Lock可以判断是否获取到了锁
- synchronized 会自动释放锁，lock必须要手动释放锁！如果不释放锁那么就是个死锁
- synchronized 线程1（获得锁，阻塞）、线程2（等待，傻乎乎的等着），Lock锁就不一定会等下去
- synchronized 可重入锁，不可以中断，非公平的；Lock，可重入锁，可以判断锁，非公平的锁，以自己设置
- synchronized 适合锁少量的代码同步问题，Lock适合锁大量的 同步代码！

生产者和消费者问题synchronized版

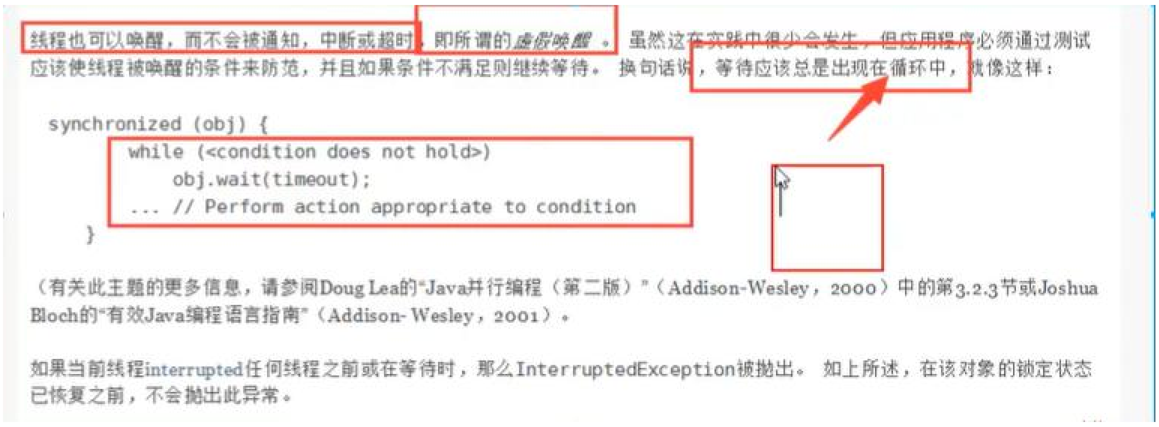
进行线程之间的通信

下面是两个线程之间的通信

```
package net.yscxy.pc;

/**
 * @Author WangFuKun
 * @create 2020/11/19 20:46
 */
/*
* 线程之间的通信问题，生产者和消费者问题！ 等待唤醒，通知唤醒
* 线程交替执行A B 操作同一个变量 num = 0
* */
public class A {
    public static void main(String[] args) {
        Data data = new Data();
        new Thread(()->{
            for (int i = 0; i < 10; i++) {
                try {
                    data.decrement();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"A").start();
        new Thread(()->{
            for (int i = 0; i < 10; i++) {
                try {
                    data.increment();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"B").start();
    }
}
//判断、等待、通知
class Data{
    private int number = 0;
    //+1
    public synchronized void increment() throws InterruptedException {
        if (number!=0){
            this.wait();
        }
        number ++;
        System.out.println(Thread.currentThread().getName()+"->"+number);
        //通知其他线程，我+1完成了
        this.notify();
    }
    //-1
    public synchronized void decrement() throws InterruptedException {
        if(number ==0){
            //等待
            this.wait();
        }
        number --;
        System.out.println(Thread.currentThread().getName()+"->"+number);
        //通知其他线程我-1完成了
        this.notify();
    }
}
```

问题出现，如果是多个线程的话就会出现问题！，例如纯在ABCD四个线程就会出现问题(虚假唤醒！)



线程也可以唤醒，而不会被通知，中断或超时，即所谓的*虚假唤醒*。虽然这在实践中很少会发生，但应用程序必须通过测试应该使线程被唤醒的条件来防范，并且如果条件不满足则继续等待。换句话说，等待应该总是出现在循环中，就像这样：

```
  synchronized (obj) {
         while (<condition does not hold>)
             obj.wait(timeout);
         ... // Perform action appropriate to condition
     }
```

（有关此主题的更多信息，请参阅Doug Lea的"Java并行编程（第二版）"（Addison-Wesley，2000）中的第3.2.3节或Joshua Bloch的"有效Java编程语言指南"（Addison- Wesley，2001）。

如果当前线程interrupted任何线程之前或在等待时，那么InterruptedException被抛出。如上所述，在该对象的锁定状态已恢复之前，不会抛出此异常。

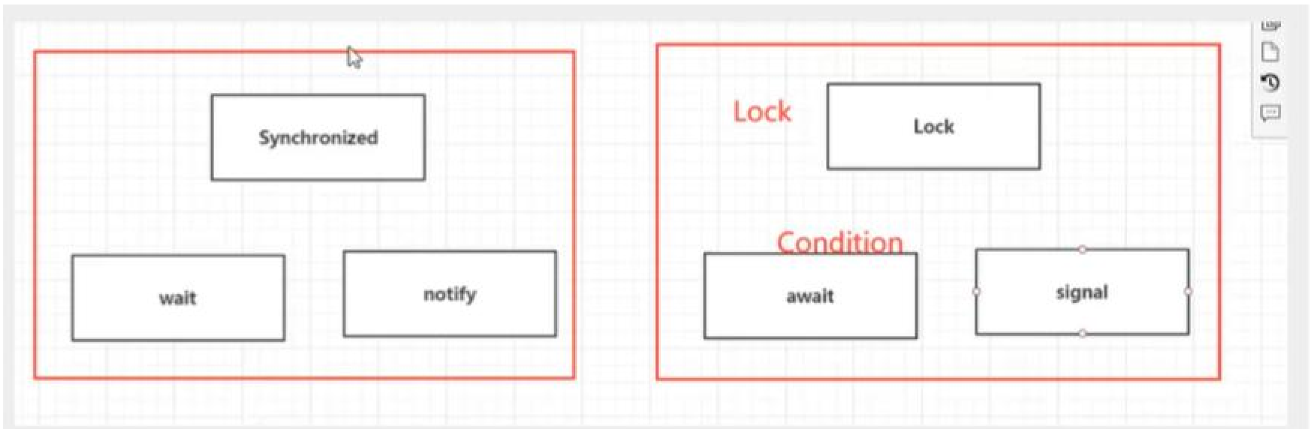
解决方案

if 换成 while，类似下面这样

```
package net.yscxy.pc;

/**
 * @Author WangFuKun
 * @create 2020/11/19 20:46
 */
/*
* 线程之间的通信问题，生产者和消费者问题！ 等待唤醒，通知唤醒
* 线程交替执行A B 操作同一个变量 num = 0
* */
public class A {
    public static void main(String[] args) {
        Data data = new Data();
        new Thread(()->{
            for (int i = 0; i < 100; i++) {
                try {
                    data.decrement();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"A").start();
        new Thread(()->{
            for (int i = 0; i < 100; i++) {
                try {
                    data.increment();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"B").start();
        new Thread(()->{
            for (int i = 0; i < 100; i++) {
                try {
                    data.decrement();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"C").start();
        new Thread(()->{
            for (int i = 0; i < 100; i++) {
                try {
                    data.decrement();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"D").start();
    }
}
//判断、等待、通知
class Data{
    private int number = 0;
    //+1
    public synchronized void increment() throws InterruptedException {
        while (number!=0){
            this.wait();
        }
        number ++;
        System.out.println(Thread.currentThread().getName()+"->"+number);
        //通知其他线程，我+1完成了
        this.notify();
    }
    //-1
    public synchronized void decrement() throws InterruptedException {
        while(number ==0){
            //等待
            this.wait();
        }
        number --;
        System.out.println(Thread.currentThread().getName()+"->"+number);
        //通知其他线程我-1完成了
        this.notify();
    }
}
```

JUC版本的生产者和消费者问题



代码实现

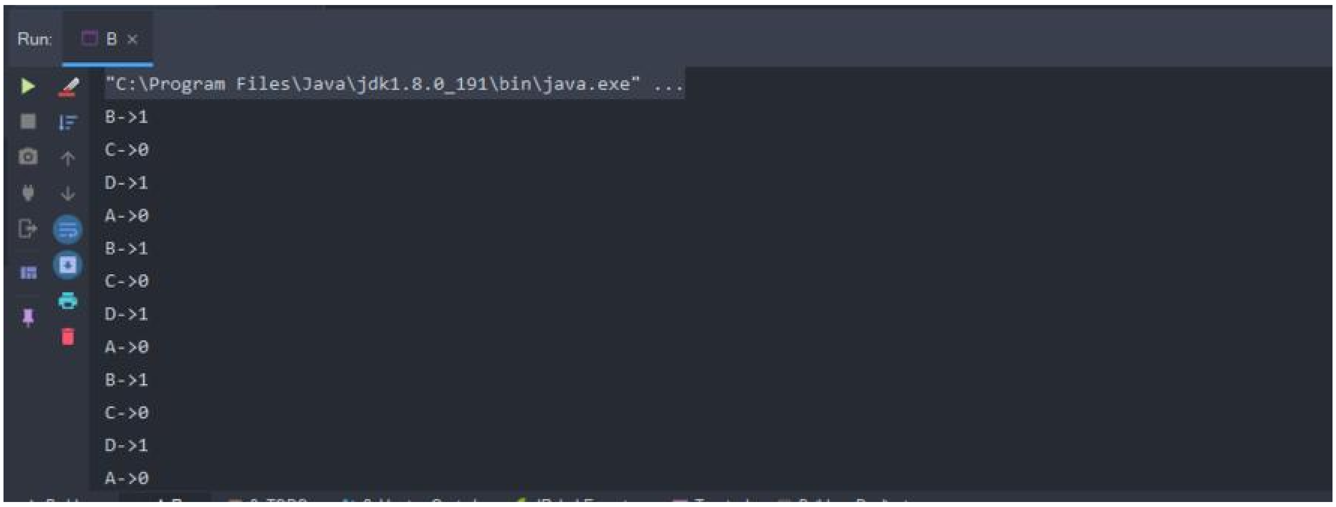
```
package net.yscxy.pc;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * @Author WangFuKun
 * @create 2020/11/19 20:46
 */
/*
* 线程之间的通信问题，生产者和消费者问题！ 等待唤醒，通知唤醒
* 线程交替执行A B 操作同一个变量 num = 0
* */
public class B {
    public static void main(String[] args) {
        Data2 data = new Data2();
        new Thread(()->{
            for (int i = 0; i < 10; i++) {
                try {
                    data.decrement();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"A").start();
        new Thread(()->{
            for (int i = 0; i < 10; i++) {
                try {
                    data.increment();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"B").start();
        new Thread(()->{
            for (int i = 0; i < 10; i++) {
                try {
                    data.decrement();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"C").start();
        new Thread(()->{
            for (int i = 0; i < 10; i++) {
                try {
                    data.increment();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },"D").start();
    }
}

//判断、等待、通知
class Data2{
    private int number = 0;
    Lock lock = new ReentrantLock();
    Condition condition = lock.newCondition();
    //+1
    public  void increment() throws InterruptedException {
        lock.lock();
        try {
            //业务代码
            while(number !=0){
                //等待
                condition.await();
            }
            number ++;
            System.out.println(Thread.currentThread().getName()+"->"+number);
            //通知其他线程，我+1完成了
            condition.signalAll();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }

    }
    //-1
    public  void decrement() throws InterruptedException {
        lock.lock();
        try {
            while(number ==0){
                //等待
                condition.await();
            }
            number --;
            System.out.println(Thread.currentThread().getName()+"->"+number);
            condition.signalAll();//唤醒全部
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
}
```



```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
B->1
C->0
D->1
A->0
B->1
C->0
D->1
A->0
B->1
C->0
D->1
A->0
```

Condition进准的通知和和唤醒线程

A、B、C、D有序执行

代码测试

```
package net.yscxy.pc;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * @Author WangFuKun
 * @create 2020/11/19 20:46
 */
/*
 * 让线程有序执行,A执行完成调用B，B执行完调用C，C执行完调用A
 * */
```

```
public class C {
    public static void main(String[] args) {
        Data3 data = new Data3();
        new Thread(() -> {
            for (int i = 0; i <10 ; i++) {
                data.PrintA();
            }
        },"A").start();
        new Thread(() -> {
            for (int i = 0; i <10 ; i++) {
                data.PrintB();
            }
        },"B").start();
        new Thread(() -> {   for (int i = 0; i <10 ; i++) {
            data.PrintC();
        } },"C").start();

    }
}

//判断、等待、通知
class Data3 {
    private int number = 1;
    Lock lock = new ReentrantLock();
    Condition condition1 = lock.newCondition();
    Condition condition2 = lock.newCondition();
    Condition condition3 = lock.newCondition();


    public void PrintA() {
        lock.lock();
        try {
            //业务、判断、执行、通知
            while (number!=1){
                //等待
                condition1.await();
            }
            System.out.println(Thread.currentThread().getName()+"-->AAAAAAAAAA");
            //唤醒指定的人,唤醒2
            number = 2;
            condition2.signal();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }

    }
    public void PrintB() {
        lock.lock();
        try {
            //业务、判断、执行、通知
            while (number!=2){
                //等待
                condition2.await();
            }
            System.out.println(Thread.currentThread().getName()+"-->BBBBBBBBB");
            //唤醒指定的人,唤醒2
            number = 3;
            condition3.signal();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
    public void PrintC() {
        lock.lock();
        try {
            //业务、判断、执行、通知
            while (number!=3){
                //等待
                condition3.await();
            }
            System.out.println(Thread.currentThread().getName()+"-->CCCCCCCCC");
            //唤醒指定的人,唤醒2
            number = 1;
            condition1.signal();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
}
```

8锁现象

如何判断锁的是谁，锁的到底是谁

new this 锁的是具体的一个手机

static Class唯一的一个模板