1. 改进版

范例:加减法操作

|  |
| --- |
| package cn.gsdx.JUC;  import java.util.concurrent.locks.Condition;  import java.util.concurrent.locks.Lock;  import java.util.concurrent.locks.ReentrantLock;  public class NewAddAndSub {  public static void main(String[] args) {  Resource res = new Resource() ;  new Thread(()->{  for (int i = 0; i < 10; i++) {  try {  res.decrement();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  },"减法线程A").start();  new Thread(()->{  for (int i = 0; i < 10; i++) {  try {  res.increment();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  },"加法线程B").start();  new Thread(()->{  for (int i = 0; i < 10; i++) {  try {  res.decrement();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  },"减法线程C").start();  new Thread(()->{  for (int i = 0; i < 10; i++) {  try {  res.increment();  } catch (Exception e) {  e.printStackTrace();  }  }  },"加法线程D").start();  }  }  class Resource{  private int num = 0 ;  private Lock lock = new ReentrantLock() ;  private Condition condition = lock.newCondition() ;  public void increment() throws InterruptedException {  lock.lock();  try{  while (num != 0){  condition.await();  }  num++ ;  System.out.println(Thread.currentThread().getName()+"，线程执行加法操作:"+ num);  condition.signalAll();  }finally{  lock.unlock();  }  }  public void decrement() throws InterruptedException {  lock.lock();  try{  while(num == 0){  condition.await();  }  num--;  System.out.println(Thread.currentThread().getName() + ",线程执行减法操作:"+ num);  condition.signalAll() ;  }finally{  lock.unlock();  }  }  } |

Java在JDK.5时候加入了java.util.concurrent.locks包里面加入了如下接口用来代替synchronized , wait(),notify(),notifyAll(),这些方法，相比这些方法新方法提供更强大的锁操作和状态。

·Condition

·Lock

·ReadWriteLock

新方法的优点:

·提供精准唤醒的功能

**范例:精准唤醒**

题目:多线程之间按顺序调用，实现A->B->C，三个线程启动。

如：AA打印15次，BB打印10，CC打印5次。(10轮)

|  |
| --- |
| package cn.gsdx.JUC;  import java.util.concurrent.locks.Condition;  import java.util.concurrent.locks.Lock;  import java.util.concurrent.locks.ReentrantLock;  /\*\*  \* @author JackWen  \* 精准唤醒  \* 题目:多线程之间按顺序调用，实现A->B->C，三个线程启动。  \* 如：AA打印15次，BB打印10，CC打印5次。(10轮)  \*/  public class PreciseAwakening {  public static void main(String[] args) {  PreciseResource pr = new PreciseResource() ;  new Thread(() -> {  try {  for (int i = 0; i < 10; i++) {  pr.print15();  }  } catch (InterruptedException e) {  e.printStackTrace();  }  },"线程A").start();  new Thread(() -> {  try {  for (int i = 0; i < 10; i++) {  pr.print10();  }  } catch (InterruptedException e) {  e.printStackTrace();  }  },"线程B").start();  new Thread(() -> {  try {  for (int i = 0; i < 10; i++) {  pr.print5();  }  } catch (InterruptedException e) {  e.printStackTrace();  }  },"线程C").start();  }  }  class PreciseResource{  /\*\*  \* num为标志位，当num=1时执行A线程，2时执行B线程， 3时执行C线程  \*/  int num = 1;  private Lock lock = new ReentrantLock();  private Condition condition1 = lock.newCondition() ;  private Condition condition2 = lock.newCondition() ;  private Condition condition3 = lock.newCondition() ;  public void print15() throws InterruptedException {  lock.lock();  try{  while ( num != 1){  condition1.await();  }  for (int i = 0; i < 15; i++) {  System.out.println("AA"+i);  }  num = 2;  condition2.signal();  }finally{  lock.unlock();  }  }  public void print10() throws InterruptedException {  lock.lock();  try{  while ( num != 2){  condition1.await();  }  for (int i = 0; i < 10; i++) {  System.out.println("BB"+i);  }  num = 3;  condition3.signal();  }finally{  lock.unlock();  }  }  public void print5() throws InterruptedException {  lock.lock();  try{  while ( num != 3){  condition1.await();  }  for (int i = 0; i < 5; i++) {  System.out.println("CC"+i);  }  num = 1;  condition1.signal();  }finally{  lock.unlock();  }  }  } |

利用num进行一个标识控制,num为标志位,当num=1时执行A线程,2时执行B线程,3时执行C线程，并且lock与condition配合进行精准唤醒，使用三把锁condition1，condition2，condition3，

使每个线程拥有独立的锁，此时我们就可以使用condition1.signal() 方法进行精准唤醒。