博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

■多线程案例分析一

设计4个线程对象，两个线程执行减操作，两个线程执行加操作。

```
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		Resource res = new Resource() ;
		SubThread st = new SubThread(res) ;
		AddThread at = new AddThread(res) ;
		new Thread(at,"加法线程 - A").start() ;
		new Thread(at,"加法线程 - B").start() ;
		new Thread(st,"减法线程 - X").start() ;
		new Thread(st,"减法线程 - Y").start() ;

	}
}
class Resource {	// 定义一个操作的资源
	private int num = 0 ; // 这个要进行加减操作的数据
	private boolean flag = true ; // 加减的切换
	// flag = true：表示可以进行加法操作，但是无法进行减法操作；
	// flag = false：表示可以进行减法操作，但是无法进行加法操作。

	public synchronized void add() throws Exception { // 执行加法操作
		if (this.flag == false) {	// 现在需要执行的是减法操作，加法操作要等待
			super.wait();
		}
		Thread.sleep(100);
		this.num++;
		System.out.println("【加法操作 - " + Thread.currentThread().getName() + "】num
= " + this.num);
		this.flag = false ; // 加法操作执行完毕，需要执行减法处理
		super.notifyAll(); // 唤醒全部等待线程
	}
```

```
	public synchronized void sub() throws Exception { // 执行减法操作
		if (this.flag == true) {	// 减法操作需要等待
			super.wait();
		}
		Thread.sleep(200);
		this.num--;
		System.out.println("【减法操作 - " + Thread.currentThread().getName() + "】num
= " + this.num);
		this.flag = true ;
		super.notifyAll();
	}
}

class AddThread implements Runnable {
	private Resource resource ;
	public AddThread(Resource resource) {
		this.resource = resource ;
	}
	@Override
	public void run() {
		for (int x = 0 ; x < 50 ; x ++) {
			try {
				this.resource.add() ;
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
}
class SubThread implements Runnable {
	private Resource resource ;
	public SubThread(Resource resource) {
		this.resource = resource ;
	}
	@Override
	public void run() {
		for (int x = 0 ; x < 50 ; x ++) {
			try {
				this.resource.sub() ;
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
}
```

这一题目是一个经典的多线程的开发操作，这一个程序里面一定要考虑的核心本质在于：加一个、减一个，整体的计算结果应该是在0、-1、1之间循环出现。

■多线程案例分析二

设计一个生产电脑和搬运电脑类，要求生产出一台电脑就搬走一台电脑，如果没有新的电脑生产出来，则搬运工要等待新电脑产出；如果生产出的电脑没有搬走，则要等待电脑搬走之后再生产，并统计出生产的电脑数量。

在本程序之中实现的就是一个标准的生产者与消费者的处理模型，那么下面实现具体的程序代码。

```
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		Resource res = new Resource() ;
		new Thread(new Producer(res)).start(); ;
		new Thread(new Consumer(res)).start(); ;
	}
}
class Producer implements Runnable {
	private Resource resource ;
	public Producer(Resource resource) {
		this.resource = resource ;
	}
	public void run() {
		for (int x = 0 ; x < 50 ; x ++) {
			try {
				this.resource.make();
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	};
}
class Consumer implements Runnable {
	private Resource resource ;
	public Consumer(Resource resource) {
		this.resource = resource ;
	}
	public void run() {
		for (int x = 0 ; x < 50 ; x ++) {
			try {
				this.resource.get();
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	};
}
class Resource {
	private Computer computer ;
	public synchronized void make() throws Exception {
		if (this.computer != null) {	// 已经生产过了
```

```
                super.wait();
            }
            Thread.sleep(100);
            this.computer = new Computer("MLDN牌电脑",1.1) ;
            System.out.println("【生产电脑】" + this.computer);
            super.notifyAll();
        }
        public synchronized void get() throws Exception {
            if (this.computer == null) {     // 没有生产过
                super.wait();
            }
            Thread.sleep(10);
            System.out.println("【取走电脑】" + this.computer);
            this.computer = null ; // 已经取走了
            super.notifyAll();
        }
}

class Computer {
        private static int count = 0 ; // 表示生产的个数
        private String name ;
        private double price ;
        public Computer(String name,double price) {
            this.name = name ;
            this.price = price ;
            count ++ ;
        }
        public String toString() {
            return "【第" + count + "台电脑】" + "电脑名字：" + this.name + "、价值：" +
this.price;
        }
}
```

■多线程案例分析三

实现一个竞拍抢答程序：要求设置三个抢答者（三个线程），而后同时发出抢答指令，抢答成功者给出成功提示，未抢答成功者给出失败提示。

对于这一个多线程的操作由于里面需要牵扯到数据的返回问题，那么现在最好使用的Callable是比较方便的一种处理形式。

```
import java.util.concurrent.Callable;
import java.util.concurrent.FutureTask;
public class ThreadDemo {
        public static void main(String[] args) throws Exception {
            MyThread mt = new MyThread() ;
            FutureTask<String> taskA = new FutureTask<String>(mt) ;
            FutureTask<String> taskB = new FutureTask<String>(mt) ;
            FutureTask<String> taskC = new FutureTask<String>(mt) ;
```

```
            new Thread(taskA,"竞赛者A").start();
            new Thread(taskB,"竞赛者B").start();
            new Thread(taskC,"竞赛者C").start();
            System.out.println(taskA.get());
            System.out.println(taskB.get());
            System.out.println(taskC.get());
        }
}
class MyThread implements Callable<String> {
        private boolean flag = false ; // 抢答处理
        @Override
        public String call() throws Exception {
                synchronized(this) {        // 数据同步
                        if (this.flag == false) {     // 抢答成功
                                this.flag = true ;
                                return Thread.currentThread().getName() + "抢答成功！" ;
                        } else {
                                return Thread.currentThread().getName() + "抢答失败！" ;
                        }
                }
        }}
```