线程互斥同步（synchronized，ReentrantLock）

一、基本问题

1、问题引出

在多线程编程中，有可能会出现同时访问同一个资源的情况，这种资源可以是各种类型的的资源，由于每个线程执行的过程是不可控的，所以很可能导致最终的结果与实际上的愿望相违背或者直接导致程序出错。

2、解决方法

Java 提供了两种锁机制来控制多个线程对共享资源的互斥访问，第一个是 JVM 实现的 synchronized，而另一个是 JDK 实现的 ReentrantLock。

3、二者比较

1）锁的实现

synchronized 是 JVM 实现的，而 ReentrantLock 是 JDK 实现的。

2） 性能

新版本 Java 对 synchronized 进行了很多优化，例如自旋锁等，synchronized 与 ReentrantLock 大致相同。

3） 等待可中断

当持有锁的线程长期不释放锁的时候，正在等待的线程可以选择放弃等待，改为处理其他事情。

ReentrantLock 可中断，而 synchronized 不行。

4.）公平锁

公平锁是指多个线程在等待同一个锁时，必须按照申请锁的时间顺序来依次获得锁。

synchronized 中的锁是非公平的，ReentrantLock 默认情况下也是非公平的，但是也可以是公平的。

5.）锁绑定多个条件

一个 ReentrantLock 可以同时绑定多个 Condition 对象。

4、使用选择

除非需要使用 ReentrantLock 的高级功能，否则优先使用 synchronized。这是因为 synchronized 是 JVM 实现的一种锁机制，JVM 原生地支持它，而 ReentrantLock 不是所有的 JDK 版本都支持。并且使用 synchronized 不用担心没有释放锁而导致死锁问题，因为 JVM 会确保锁的释放。

二、方法实现

synchronized

1、同步代码块

1）基本步骤

```
synchronized （同步监视器、锁）{
      //操作共享数据的代码
      //需要被同步的代码
}
```

2）实例

```
public class text {
      public static void main(String[] args) {
         MyThread a = new MyThread("A");
         MyThread b = new MyThread("B");
         MyThread c = new MyThread("C");
         a.start();
         b.start();
         c.start();
      }
```

```
    public static class MyThread extends Thread {
        private static int count = 10;
        public MyThread(String name) {
            super();
            this.setName(name);
        }

    public void run() {
        super.run();
        while (count > 0) {  //B,C会在此等待，等A结束会继续运行while里的
            //当票数为1时，若A抢到资源
            if(count<0) return;
            synchronized (MyThread.class) {
            System.out.println("由 " + MyThread.currentThread().getName()
                    + " 计算，count=" + count--);
}
```

注意：Runable天生共享锁，而Thread中需要用static对象或者this关键字或者当前类（window。class）来充当唯一锁

2、同步方法

PS: 同步方法的对象锁是当前对象 即 this

静态方法的对象锁为当前类.class

然后可以进行this.notify()，类.class.notify ()

```
//同步方法的对象锁是当前对象 即 this
//静态方法的对象锁为当前类.class
public  synchronized void method(){
}
```

1）实例

把上面的公共代码提取出来

```
private static synchronized  void sold(){
    System.out.println("由 " + MyThread.currentThread().getName()
            + " 计算，count=" + count--);
      //将该锁唤醒
       MyThread.class.notify();
       //等待
       try {
            MyThread.class.wait();
            } catch (InterruptedException e) {
            e.printStackTrace();
```

```
    }
}
```

ReentrantLock

1、基本介绍

ReentrantLock 是 java.util.concurrent（J.U.C）包中的锁

2、Lock

lock(); 获取锁

unlock(); 释放锁

```
 lock(); 获取锁
try{
   操作共享数据的代码
  } finally {
   //释放锁
   unlock();
  }
```

实例：

```
  public void set(String name){
        lock.lock();//锁住此语句与lock.unlock()之间的代码
        try{
            while(flag)
                condition_pro.await(); //生产者线程在conndition_pro对象上等待
            this.name=name+"---"+count++;
            System.out.println(Thread.currentThread().getName()+"...生产
者..."+this.name);
            flag=true;
             condition_con.signalAll();
        }
        finally{
            lock.unlock(); //unlock()要放在finally块中。
        }
    }
```