多线程中存在的死锁现象

1. 死锁：一种很常见的情况就是**发生在同步的嵌套情况下**。
2. 面试时：可能会让写出一个**死锁**示例。

最简单的例子就是：**先给钱再交货，和先交货再给钱**。

钱和货都是两个锁。**买方法需要先判断货对象，再拿钱；而卖方法需要先判断钱，再出货。这样hen容易死锁了。**

**Buy：**

**public** **class** Buy **implements** Runnable {

Object money = **null**;

Object goods = **null**;

**public** Buy( Object money,Object goods) {

**this**.money = money;

**this**.goods = goods;

}

**public** **void** buy() {

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

}

**synchronized** (goods) {

**synchronized** (money) {

System.***out***.println("buy");

}

}

}

@Override

**public** **void** run() {

**this**.buy();

} }

Shou：

**public** **class** Shou **implements** Runnable {

Object lock1 = **null**;

Object lock2 = **null**;

**public** Shou(Object lock1,Object lock2) {

**this**.lock1 = lock1;

**this**.lock2 = lock2;

}

**public** **void** mai() {

**synchronized** (lock1) {

**try** {

Thread.*sleep*(1000);//增加死锁的几率

} **catch** (InterruptedException e) {

}

**synchronized** (lock2) {

System.***out***.println("mai");

}

}

}

@Override

**public** **void** run() {

**this**.mai();

}}

Test：

**public** **static** **void** main(String[] args) {

Object money = **new** Object();

Object goods = **new** Object();

Buy buy = **new** Buy(money, goods);

Shou shou = **new** Shou(money, goods);

**new** Thread(buy).start();

**new** Thread(shou).start();

}

结果：很容易死锁。但是也有不死锁的情况。

1. 示例：

public class DeadLockDemo implements Runnable {

boolean flag;

public DeadLockDemo(boolean flag) {

super();

this.flag = flag;

}

public void run() {

if(flag) {

**synchronized (MyLock.lockA) {//外同步**

System.out.println(Thread.currentThread().getName()+" 外");

**synchronized(MyLock.lockB)** {//内同步

System.out.println(Thread.currentThread().getName()+" 内");

}

}

}

else

{

**synchronized (MyLock.lockB) {//外同步**

System.out.println(Thread.currentThread().getName()+" 外");

**synchronized(MyLock.lockA) {**//内同步

System.out.println(Thread.currentThread().getName()+" 内");

}

}

}

}

}

两把锁：

public class MyLock {

public static final Object lockA = new Object();

public static final Object lockB = new Object();

}

主方法：

public static void main(String[] args) {

DeadLockDemo dl = new DeadLockDemo(false);

DeadLockDemo d2 = new DeadLockDemo(true);

Thread th1 = new Thread(dl,"线程1");

Thread th2 = new Thread(d2,"线程2");

th1.start();

th2.start();

}

直接就***死锁***。

**解析： 开启了两个子线程（run方法中分别是两个同步的嵌套使用）：对于t1线程而言，首先是A锁的钥匙，然后是B锁的钥匙，；t2线程正好相反，首先是B的钥匙，然后是A的钥匙。首先，线程t1 利用A钥匙打开，紧接着t2利用B钥匙打开，这样T1**

**拥有A钥匙，需要B钥匙才可进入第二层同步，T2拥有B钥匙需要A钥匙才可以进入第二层同步，就这样，各自都不能得到自己的东西，又不释放已有的东西，产生死锁现象。**

1. 示例2：死锁

public class DeadLockDemo implements Runnable {

boolean flag;

static Object obj = new Object();

public DeadLockDemo(boolean flag) {

super();

this.flag = flag;

}

public void run() {

if (flag) {

while (true)

synchronized (DeadLockDemo.class) {**//需要class锁**

synchronized(obj) {//需要obj锁

System.out.println(Thread.currentThread().getName());

}

}

}

else {

while(true) {

**synchronized(obj) {**//需要obj锁

**show1();//需要class锁**

}

}

}

}

private static synchronized void show1() {

System.out.println(Thread.currentThread().getName());

}

}

DeadLockDemo dl = new DeadLockDemo(false);

DeadLockDemo d2 = new DeadLockDemo(true);

Thread th1 = new Thread(dl,"线程1");

Thread th2 = new Thread(d2,"线程2");

th1.start(); th2.start();