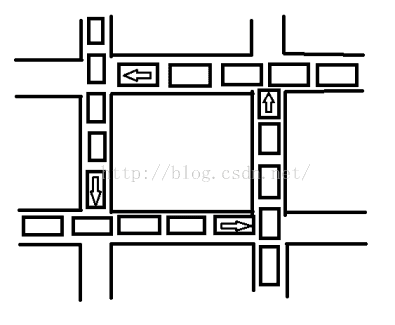
**Java线程死锁及解决方案**

要了解线程死锁，首先要明白什么是死锁

死锁

通俗点讲：死锁就是两个或两个以上的进程或线程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。

用简单一点的例子来说吧



比如这个交通堵塞的例子，从图中可以看到四个方向行驶的汽车互相阻塞，如果没有任何一个方向的汽车退回去，那么将形成一个死锁

上述图中有产生死锁的四个原因：

**1.互斥条件：**一个资源每次只能被一个线程使用。图上每条路上只能让一个方向的汽车通过，故满足产生死锁的条件之一

**2.请求与保持条件：**一个进程因请求资源而阻塞时，对已获得的资源保持不放。可以看出，图上每个方向的汽车都在等待其他方向的汽车撤走，故满足产生死锁的条件之二

**3.不剥夺条件：**进程已获得的资源，在未使用完之前，不能强行剥夺。这里假设没有交警，那么没有人能强行要求其他方向上的汽车撤离，故满足产生死锁的条件之三

**4.循环等待条件：**若干进程或线程之间形成一种头尾相接的循环等待资源关系。这个在图中很直观地表达出来了

死锁Java代码小例子

**[html]** [view plain](https://blog.csdn.net/dream_18/article/details/52645237) [copy](https://blog.csdn.net/dream_18/article/details/52645237)

1. package huaxin2016\_9\_9;
3. public class ThreadDeadlock {
4. public static void main(String[] args) throws InterruptedException {
5. Object obj1 = new Object();
6. Object obj2 = new Object();
7. Object obj3 = new Object();
8. //新建三个线程
9. Thread t1 = new Thread(new SyncThread(obj1, obj2), "t1");
10. Thread t2 = new Thread(new SyncThread(obj2, obj3), "t2");
11. Thread t3 = new Thread(new SyncThread(obj3, obj1), "t3");
12. //让线程依次开始
13. t1.start();
14. //让线程休眠
15. Thread.sleep(5000);
16. t2.start();
17. Thread.sleep(5000);
18. t3.start();
19. }
20. }
21. class SyncThread implements Runnable{
22. private Object obj1;
23. private Object obj2;
24. //构造函数
25. public SyncThread(Object o1, Object o2){
26. this.obj1=o1;
27. this.obj2=o2;
28. }
29. @Override
30. public void run() {
31. //获取并当前运行线程的名称
32. String name = Thread.currentThread().getName();
33. System.out.println(name + " acquiring lock on "+obj1);

36. synchronized (obj1) {
37. System.out.println(name + " acquired lock on "+obj1);
38. work();
39. System.out.println(name + " acquiring lock on "+obj2);
40. synchronized (obj2) {
41. System.out.println(name + " acquired lock on "+obj2);
42. work();
43. }
44. System.out.println(name + " released lock on "+obj2);
45. }
46. System.out.println(name + " released lock on "+obj1);
47. System.out.println(name + " finished execution.");
48. }
49. private void work() {
50. try {
51. Thread.sleep(30000);
52. }
53. catch (InterruptedException e) {
54. e.printStackTrace();
55. }
56. }
57. }

上述死锁小例子运行结果为

**[html]** [view plain](https://blog.csdn.net/dream_18/article/details/52645237) [copy](https://blog.csdn.net/dream_18/article/details/52645237)

1. t1 acquiring lock on java.lang.Object@675d5ed4
2. t1 acquired lock on java.lang.Object@675d5ed4
3. t2 acquiring lock on java.lang.Object@7943f708
4. t2 acquired lock on java.lang.Object@7943f708
5. t3 acquiring lock on java.lang.Object@46767615
6. t3 acquired lock on java.lang.Object@46767615
7. t1 acquiring lock on java.lang.Object@7943f708
8. t2 acquiring lock on java.lang.Object@46767615
9. t3 acquiring lock on java.lang.Object@675d5ed4

可以很直观地看到，**t1、t2、t3都在要求资源，却都保持自己的资源，故而引起死锁**

解决方案：

1.打破互斥条件，我们需要允许进程同时访问某些资源，这种方法受制于实际场景，不太容易实现条件；

2.打破不可抢占条件，这样需要允许进程强行从占有者那里夺取某些资源，或者简单一点理解，占有资源的进程不能再申请占有其他资源，**必须释放手上的资源之后才能发起申请**，这个其实也很难找到适用场景；(一点点释放，一点点请求到。)

3. **进程在运行前申请得到所有的资源，否则该进程不能进入准备执行状态。**这个方法看似有点用处，但是它的缺点是可能导致资源利用率和进程并发性降低

4.  避免出现资源申请环路，即对资源事先**分类编号，按号分配**。这种方式可以有效提高资源的利用率和系统吞吐量，但是增加了系统开销，增大了进程对资源的占用时间。