线程死锁

1、什么叫线程的死锁

所谓死锁是指多个线程因竞争资源而造成的一种僵局（互相等待），若无外力作用，这些进程都将无法向前推进。

现实生活中的举例1：

张三手上有一支笔，但是没有画板。

李四手上有画板，但是没有笔。

他俩都想画画。

张三等着李四将画板给他，李四等着张三将笔给他。

此时：张三和李四，就互相等待。陷入僵局。两个都无法画画。

现实生活中的举例2：

十字路口。

十字路口，由红绿灯控制通行的。

举例3：

先看生活中的一个实例，两个人面对面过独木桥，甲和乙都已经在桥上走了一段距离，即占用了桥的资源，甲如果想通过独木桥的话，乙必须退出桥面让出桥的资源，让甲通过，但是乙不服，为什么让我先退出去，我还想先过去呢，于是就僵持不下，导致谁也过不了桥，这就是死锁。

对应在线程中：

A线程（张三）持有A对象（笔）的锁，并且又想去获取B对象（画板）的锁。

B线程（李四）持有B对象（画板）的锁，并且又想去获取A对象（笔）的锁。

A线程持有A对象的锁，B线程持有B对象，A在等待B释放B对象的锁，B线程等待A线程释放A对象的锁。

陷入无限等待。

这就是死锁。

2、造成的后果：

陷入死锁的线程，全部卡死，无法推进。

3、死锁产生的必要条件：

产生死锁必须同时满足以下四个条件，只要其中任一条件不成立，死锁就不会发生。

（1）互斥条件：进程要求对所分配的资源（如打印机）进行排他性控制，即在一段时间内某资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待。

（2）不剥夺条件：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放（只能是主动释放)。

（3）请求和保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

（4）循环等待条件：存在一种进程资源的循环等待链，链中每一个进程已获得的资源同时被链中下一个进程所请求。即存在一个处于等待状态的进程集合{Pl, P2, ..., pn}，其中Pi等 待的资源被P(i+1)占有（i=0, 1, ..., n-1)，Pn等待的资源被P0占有，如图1所示。

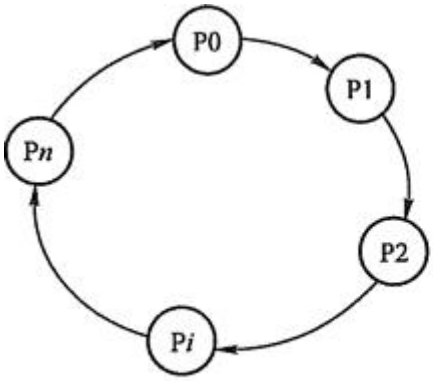

图1　循环等待

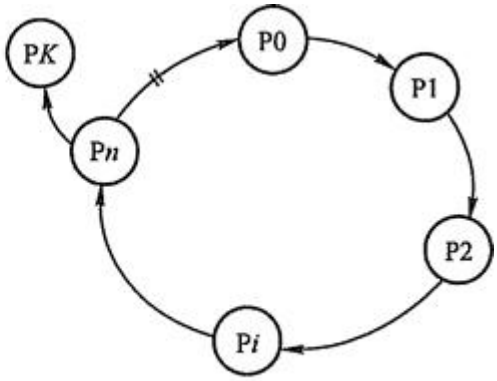

图2　满足条件但无死循环

```
//两个线程互相等待
public class DeadLock implements Runnable{

    private static Object obj1 = new Object();
    private static Object obj2 = new Object();
    private boolean flag;

    public DeadLock(boolean flag){
        this.flag = flag;
    }

    @Override
    public void run(){
        System.out.println(Thread.currentThread().getName() + "运行");

        if(flag){
            synchronized(obj1){
                System.out.println(Thread.currentThread().getName() + "已经锁住obj1");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                synchronized(obj2){
                    // 执行不到这里
                    System.out.println("1秒钟后，"+Thread.currentThread().getName()
                                + "锁住obj2");
                }
            }
        }else{
            synchronized(obj2){
                System.out.println(Thread.currentThread().getName() + "已经锁住obj2");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                synchronized(obj1){
                    // 执行不到这里
                    System.out.println("1秒钟后，"+Thread.currentThread().getName()
```

```
                                + "锁住obj1");
                }
            }
        }
    }

}


package com.demo.test;

public class DeadLockTest {

     public static void main(String[] args) {

         Thread t1 = new Thread(new DeadLock(true), "线程1");
         Thread t2 = new Thread(new DeadLock(false), "线程2");

         t1.start();
         t2.start();
    }
}

//结果
线程1运行
线程1已经锁住obj1
线程2运行
线程2已经锁住obj2

//结果解析
线程1锁住了obj1（甲占有桥的一部分资源），线程2锁住了obj2（乙占有桥的一部分资源），线程1企图锁住obj2（甲让乙退出桥面，乙不从），进入阻塞，线程2企图锁住obj1（乙让甲退出桥面，甲不从），进入阻塞，死锁了。

    从这个例子也可以反映出，死锁是因为多线程访问共享资源，由于访问的顺序不当所造成的，通常是一个线程锁定了一个资源A，而又想去锁定资源B；在另一个线程中，锁定了资源B，而又想去锁定资源A以完成自身的操作，两个线程都想得到对方的资源，而不愿释放自己的资源，造成两个线程都在等待，而无法执行的情况。
```

```
//多个线程，循环等待，线程闭环
public class SyncThread implements Runnable{

    private Object obj1;
    private Object obj2;

    public SyncThread(Object o1, Object o2){
        this.obj1=o1;
        this.obj2=o2;
    }

    @Override
    public void run() {
        String name = Thread.currentThread().getName();
```

```
        synchronized (obj1) {
            System.out.println(name + " acquired lock on "+obj1);
            work();
            synchronized (obj2) {
                System.out.println("After, "+name + " acquired lock on "+obj2);
                work();
            }
            System.out.println(name + " released lock on "+obj2);
        }
        System.out.println(name + " released lock on "+obj1);
        System.out.println(name + " finished execution.");
    }

    private void work() {
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}



public class ThreadDeadTest {

    public static void main(String[] args) throws InterruptedException {
        Object obj1 = new Object();
        Object obj2 = new Object();
        Object obj3 = new Object();

        Thread t1 = new Thread(new SyncThread(obj1, obj2), "t1");
        Thread t2 = new Thread(new SyncThread(obj2, obj3), "t2");
        Thread t3 = new Thread(new SyncThread(obj3, obj1), "t3");

        t1.start();
        Thread.sleep(1000);
        t2.start();
        Thread.sleep(1000);
        t3.start();

    }
}

//结果
t1 acquired lock on java.lang.Object@5e1077
t2 acquired lock on java.lang.Object@1db05b2
t3 acquired lock on java.lang.Object@181ed9e

//结果分析
在这个例子中，形成了一个锁依赖的环路。以t1为例，它先将第一个对象锁住，但是当它试着向第二个对象获取锁时，它就会进入等待状态，因为第二个对象已经被另一个线程锁住了。这样以此类推，t1依赖t2锁住的对象obj2，t2依赖t3锁住的对象obj3，而t3依赖t1锁住的对象obj1，从而导致了死锁。在线程引起死锁的过程中，就形成了一个依赖于资源的循环。
```

4、如何避免死锁

在有些情况下死锁是可以避免的。下面介绍三种用于避免死锁的技术：

- 加锁顺序（线程按照一定的顺序加锁）
- 加锁时限（线程尝试获取锁的时候加上一定的时限，超过时限则放弃对该锁的请求，并释放自己占有的锁）
  tryLock
- 死锁检测

参考资料：https://www.cnblogs.com/xiaoxi/p/8311034.html