死锁

原文链接 **作者：**Jakob Jenkov **译者：**申章 **校对：**丁一

死锁是两个或更多线程阻塞着等待其它处于死锁状态的线程所持有的锁。死锁通常发生在多个线程同时但以不同的顺序请求同一组锁的时候。

例如，如果线程1锁住了A，然后尝试对B进行加锁，同时线程2已经锁住了B，接着尝试对A进行加锁，这时死锁就发生了。线程1永远得不到B，线程2也永远得不到A，并且它们永远也不会知道发生了这样的事情。为了得到彼此的对象（A和B），它们将永远阻塞下去。这种情况就是一个死锁。

该情况如下：

```
Thread 1  locks A, waits for B

Thread 2  locks B, waits for A
```

这里有一个TreeNode类的例子，它调用了不同实例的synchronized方法：

```
01 public class TreeNode {
02     TreeNode parent   = null;
03     List children = new ArrayList();
04
05     public synchronized void addChild(TreeNode child){
06         if(!this.children.contains(child)) {
07             this.children.add(child);
08             child.setParentOnly(this);
09         }
10     }
11
12     public synchronized void addChildOnly(TreeNode child){
13         if(!this.children.contains(child){
14             this.children.add(child);
15         }
16     }
17
18     public synchronized void setParent(TreeNode parent){
19         this.parent = parent;
20         parent.addChildOnly(this);
21     }
22
23     public synchronized void setParentOnly(TreeNode parent){
24         this.parent = parent;
25     }
26 }
```

如果线程1调用parent.addChild(child)方法的同时有另外一个线程2调用child.setParent(parent)方法，两个线程中的parent表示的是同一个对象，child亦然，此时就会发生死锁。下面的伪代码说明了这个过程：

```
Thread 1: parent.addChild(child); //locks parent
         --> child.setParentOnly(parent);

Thread 2: child.setParent(parent); //locks child
         --> parent.addChildOnly()
```

首先线程1调用parent.addChild(child)。因为addChild()是同步的，所以线程1会对parent对象加锁以不让其它线程访问该对象。

然后线程2调用child.setParent(parent)。因为setParent()是同步的，所以线程2会对child对象加锁以不让其它线程访问该对象。

现在child和parent对象被两个不同的线程锁住了。接下来线程1尝试调用child.setParentOnly()方法，但是由于child对象现在被线程2锁住的，所以该调用会被阻塞。线程2也尝试调用parent.addChildOnly()，但是由于parent对象现在被线程1锁住，导致线程2也阻塞在该方法处。现在两个线程都被阻塞并等待着获取另外一个线程所持有的锁。

注意：像上文描述的，这两个线程需要同时调用parent.addChild(child)和child.setParent(parent)方法，并且是同一个parent对象和同一个child对象，才有可能发生死锁。上面的代码可能运行一段时间才会出现死锁。

这些线程需要**同时**获得锁。举个例子，如果线程1稍微领先线程2，然后成功地锁住了A和B两个对象，那么线程2就会在尝试对B加锁的时候被阻塞，这样死锁就不会发生。因为线程调度通常是不可预测的，因此没有一个办法可以准确预测**什么时候**死锁会发生，仅仅是**可能会**发生。

更复杂的死锁

死锁可能不止包含2个线程，这让检测死锁变得更加困难。下面是4个线程发生死锁的例子：

```
Thread 1  locks A, waits for B
Thread 2  locks B, waits for C
Thread 3  locks C, waits for D
Thread 4  locks D, waits for A
```

线程1等待线程2，线程2等待线程3，线程3等待线程4，线程4等待线程1。

数据库的死锁

更加复杂的死锁场景发生在数据库事务中。一个数据库事务可能由多条SQL更新请求组成。当在一个事务中更新一条记录，这条记录就会被锁住避免其他事务的更新请求，直到第一个事务结束。同一个事务中每一个更新请求都可能会锁住一些记录。

当多个事务同时需要对一些相同的记录做更新操作时，就很有可能发生死锁，例如：

```
Transaction 1, request 1, locks record 1 for update
Transaction 2, request 1, locks record 2 for update
Transaction 1, request 2, tries to lock record 2 for update.
Transaction 2, request 2, tries to lock record 1 for update.
```

因为锁发生在不同的请求中，并且对于一个事务来说不可能提前知道所有它需要的锁，因此很难检测和避免数据库事务中的死锁。

十三月
2014/08/16 3:39下午

如果线程1调用parent.addChild(child)方法的同时有另外一个线程2调用child.setParent(parent)方法，两个线程中的parent表示的是同一个对象，child亦然，此时就会发生死锁。

这个说法是错误的吧，addChild只对parent对象加锁，并没有对child对象加锁，所以线程1是不需要获取child对象的锁的，所以不会有死锁发生才对。

public synchronized void addChild(TreeNode child){

synchronized (child){

if(!this.children.contains(child)) {

this.children.add(child);

child.setParentOnly(this);

}

}

}

只有这样才活造成死锁吧。不知我理解的是否正确

sandaobusi
2015/01/30 9:49上午

child.setParentOnly(this);

线程1会在这句话对child对象加锁

maleking
2019/06/06 10:51上午

挖坟回复：

addChild方法被调用，前提是有一个TreeNode类型的parent对象来调用的，即：parent.addChild(child)。因为addChild方法是同步的，所以线程1获取了parent对象锁，即parent被锁了。接下来会继续执行，到了child.setParentOnly(this)，同样，有一个child对象，调用它的同步方法setParentOnly，那么此时需要获取child对象锁才能进行。在此之前，线程2已经执行了child.setParent同步方法，获得了child的对象锁。这个时候，线程1再想获得child对象锁已经不可能了。所以互相等待，导致死锁的发生。

回复的目的是为了看看自己是否理解正确，其他人看到我的理解如果发现不正确，烦请指正啊。谢谢！

cenyol
2017/03/16 10:10上午

个人对锁机制的大致理解是：

关键在于明白哪个代码块是被上锁的区域(理解为房间)，哪个对象是作为开锁的钥匙。就能很好的解释了。

对于synchronized代码块来说，它自己就是被锁的区域(房间)，而钥匙就是synchronized(obj){}这个块的括号里面的那个obj，并且这个obj钥匙在任何一个时刻只能被一个线程拥有占用。

记住一个被锁的房间，只能有一把钥匙可以开，而一把钥匙在任何时刻只能有一个线程占有。最最最关键的是明白哪个是被锁的区域，哪个对象是作为这个锁的区域钥匙，问题就不大了。

对于实例对象的synchronized方法来说，方法自己是被锁的区域，而实例这个对象是钥匙，就是那个this关键字。

对于类的静态sync方法(static synchronized)来说，比如类名为MyClass，被锁区域也是这个方法自己，而钥匙则是MyClass.class

得钥匙者得天下！！！找钥匙，找钥匙，找钥匙！

更多请见这里：https://cenyol.com/the-granularity-of-synchronized-in-java/