**Deadlock**

* [Thread Deadlock](http://tutorials.jenkov.com/java-concurrency/deadlock.html#thread-deadlock)
* [Database Deadlocks](http://tutorials.jenkov.com/java-concurrency/deadlock.html#database-deadlocks)

**Thread Deadlock**

死锁是当两个或更多的线程被阻塞等待获取死锁中某些其他线程持有的锁。死锁可能当多个线程需要相同锁，在同一时刻，但以不同的顺序获取他们时发生。

举例来说，如果线程1锁A，尝试锁B，线程2已经锁了B，尝试锁A，一个死锁生成了。线程1不能获取B，线程2不能获取A。额外，它们都将不知道。它们将会在各自的对象A和B上一直保持阻塞。这种情形就是死锁。

情形说明如下：

Thread 1 locks A, waits for B

Thread 2 locks B, waits for A

下面是一个在不同实例中调用同步方法的TreeNode类的示例：

public class TreeNode {

TreeNode parent = null;

List children = new ArrayList();

public synchronized void addChild(TreeNode child){

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

this.children.add(child);

child.setParentOnly(this);

}

}

public synchronized void addChildOnly(TreeNode child){

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

this.children.add(child);

}

}

public synchronized void setParent(TreeNode parent){

this.parent = parent;

parent.addChildOnly(this);

}

public synchronized void setParentOnly(TreeNode parent){

this.parent = parent;

}

}

如果一个线程(1)调用parent.addChild(child)方法，同时另一个线程(2)调用child.setParent(parent)方法，那么在相同的父实例和子实例上，可能会出现死锁。下面是一些伪代码来说明这一点：

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，线程1有效地锁定其他线程对parent对象的访问。

然后线程2调用child.setParent(parent)。由于setParent()是同步的，线程2有效地锁定其他线程对child对象的访问。

现在，子对象和父对象都被两个不同的线程锁定。接下来的线程1尝试调用child.setParentOnly()方法，但是child对象被线程2锁定，因此方法调用阻塞。线程2同样尝试调用parent.addChildOnly()，但是parent对象被线程1锁定，导致线程2阻塞该方法调用。现在，两个线程都被阻塞，等待获得另一个线程所持有的锁。

注意：这两个线程必须如上所述同时调用parent.addChild(child)和child.setParent(parent)，并且在相同的两个父和子实例上才能发生死锁。上面的代码可以执行很长一段时间，直到它突然死锁。

线程真的需要\*同时\*取锁。例如，如果线程1稍微领先于线程2，因此锁定A和B，那么线程2在试图锁定B时就已经被阻塞了。由于线程调度常常是不可预知的，所以无法预测\*什么时候\*发生死锁。只能知道它\*能\*发生。

**More Complicated Deadlocks**

死锁还可以包括两个以上的线程。这使得检测更加困难。这里有一个例子，其中四个线程死锁：

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。

**Database Deadlocks**

发生死锁的更复杂的情况是数据库事务。数据库事务可以由许多SQL更新请求组成。在事务期间更新记录时，该记录被锁定以用于其他事务的更新，直到第一个事务完成为止。因此，同一事务内的每个更新请求可能会在数据库中锁定一些记录。

如果多个事务在需要更新相同记录的同时运行，那么它们就有可能陷入死锁。

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.

由于锁是在不同的请求中获取的，并且并非给定事务所需的所有锁都是提前知道的，因此很难检测或防止数据库事务中的死锁。