重入锁死

原文链接 **作者：**Jakob Jenkov **译者：**刘晓日 **校对：**丁一

重入锁死与死锁和嵌套管程锁死非常相似。锁和读写锁两篇文章中都有涉及到重入锁死的问题。

当一个线程重新获取锁，读写锁或其他不可重入的同步器时，就可能发生重入锁死。可重入的意思是线程可以重复获得它已经持有的锁。Java的synchronized块是可重入的。因此下面的代码是没问题的：

（译者注：这里提到的锁都是指的不可重入的锁实现，并不是Java类库中的Lock与ReadWriteLock类）

```
public class Reentrant{
    public synchronized outer(){
        inner();
    }

    public synchronized inner(){
        //do something
    }
}
```

注意outer()和inner()都声明为synchronized，这在Java中这相当于synchronized(this)块（*译者注：这里两个方法是实例方法，synchronized的实例方法相当于在this上加锁，如果是static方法，则不然，更多阅读：哪个对象才是锁？*）。如果某个线程调用了outer()，outer()中的inner()调用是没问题的，因为两个方法都是在同一个管程对象(即this)上同步的。如果一个线程持有某个管程对象上的锁，那么它就有权访问所有在该管程对象上同步的块。这就叫可重入。若线程已经持有锁，那么它就可以重复访问所有使用该锁的代码块。

下面这个锁的实现是不可重入的：

```
public class Lock{
    private boolean isLocked = false;
    public synchronized void lock()
        throws InterruptedException{
        while(isLocked){
            wait();
        }
        isLocked = true;
    }

    public synchronized void unlock(){
        isLocked = false;
        notify();
    }
}
```

如果一个线程在两次调用lock()间没有调用unlock()方法，那么第二次调用lock()就会被阻塞，这就出现了重入锁死。

避免重入锁死有两个选择：

1. 编写代码时避免再次获取已经持有的锁
2. 使用可重入锁

至于哪个选择最适合你的项目，得视具体情况而定。可重入锁通常没有不可重入锁那么好的表现，而且实现起来复杂，但这些情况在你的项目中也许算不上什么问题。无论你的项目用锁来实现方便还是不用锁方便，可重入特性都需要根据具体问题具体分析。

忍释
2015/03/05 6:38下午

对于可重入锁避免死锁解释的很到位~

jzh0535
2016/06/21 9:00下午

mark!