**Reentrance Lockout**

重入锁定是一种类似于死锁([**deadlock**](http://tutorials.jenkov.com/java-concurrency/deadlock.html))和嵌套监视器锁定([**nested monitor lockout**](http://tutorials.jenkov.com/java-concurrency/nested-monitor-lockout.html))的情况。重入锁定也在[**Locks**](http://tutorials.jenkov.com/java-concurrency/locks.html)和[**Read / Write Locks**](http://tutorials.jenkov.com/java-concurrency/read-write-locks.html)文中讲到。

重入锁定可能在线程重进入一个锁([**Lock**](http://tutorials.jenkov.com/java-concurrency/locks.html))、读写锁([**ReadWriteLock**](http://tutorials.jenkov.com/java-concurrency/read-write-locks.html))或者某个其他的非可重入的同步器时发生。可重入意味着一个已经持有一个锁的线程可以重复持有它。Java的同步块是可重入的。因此接下来的代码将没问题地工作：

public class Reentrant{

public synchronized outer(){

inner();

}

public synchronized inner(){

//do something

}

}

注意outer()和inner()都声明同步，在Java中等价于一个synchronized(this)块。如果一个线程调用outer()，调用inner()从outer()内部是没问题的，因为两个方法(或块)在相同的指示器对象 (“this”) 上同步。如果一个线程已经在一个指示器对象上持有锁，它有对同步在相同指示器对象上的全部块有访问。这叫做可重入。这个线程可以重进入任一它已持有锁的代码块。

接下来的Lock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();

}

}

如果一个线程调用lock()两次而在其期间不调用unlock()，第二个对lock()的调用将阻塞。重入锁定发生了。

为了避免重入锁定你必须有两项选择：

1. 避免写重进入锁的代码
2. 使用可重入锁

这些选项的哪一项最适合你的工程依赖于你的具体情况。可重入锁经常不如不可重入锁性能好，并且它们更难实现，但是这可能在你的情形中不必然是一个问题。当有或没有锁可重入，你的代码是否更容易实现必须逐一确定。