第十三章 Lock与ReentrantLock

1. 在Java5.0之后，除了synchronized和volatile，还增加了ReentrantLock机制。与之前两种不同的是，ReentrantLock并不是一种内置锁。

#### 一．Lock与ReentrantLock

1. 显示锁

与内置锁不同的是，Lock提供了一种无条件的、可轮询的、定时的以及可中断的锁操作，所有加锁和解锁的方法都是显示的。

2. ReentrantLock和Synchronized具有相同的互斥性和内存可见性。

3. 内置锁的缺点

（1）无法中断一个正在等待获取的线程。

（2）无法在请求获取一个锁时无限等待下去。

（3）无法实现非阻塞结构的加锁规则。

（4）必须在获取该锁的代码块中释放。

4. 显示锁的缺点

（1）如果没有finally来释放lock，就很危险。

5. 轮询锁与定时锁

通过tryLock来实现。可以避免死锁的发生。

如果获取锁成功则返回true，如果没有获取锁则返回false并不会一直等待。

定时锁可以在指定时间内给出结果，如果没有，那么程序就会提前结束。

6. 可中断的锁获取操作

7. 非块结构的加锁

在内置锁中，锁的获取和释放等操作都是基于代码块的------释放锁的操作总是与获取锁的操作处于同一个代码块，而不考虑控制权如何退出改代码块。

#### 二．性能考虑因素

1. 在Java5.0中内置所的性能不如ReentrantLock，但是在java6中，二者情况基本相当。

2. 性能是一个不断变化的指标，如果在昨天的测试基准中发现X比Y更快，那么在今天就可能已经过时了。

#### 三．公平性

1. ReentrantLock可以创建公平性的锁和非公平的锁（默认非公平）。

公平锁：线程将按照它们发出请求的顺序来获得锁，等待获取的锁被放入一个队列中；

非公平锁：允许线程“插队”，如果在线程发出请求的同时该锁的状态变为可用，那么这个线程将跳过队列中所有的等待线程并获得这个锁。只有当锁被某个线程持有时，新发出的请求的线程才会被放入队列。

2. 非公平所的性能高于公平锁：

原因：恢复被挂起的线程与该线程真正开始运行之间存在着严重的延迟。

#### 四．在synchronized和ReentrantLock之间进行选择

1. 在一些内置锁无法满足的情况下，ReentrantLock可以作为一种高级工具。当需要一些高级功能时才应该使用ReentrantLock，这些功能包括：可定时的、可轮询的与可中断的锁获取操作，公平队列，以及非块结构的锁。否则，还是应该优先使用synchronized。

#### 五．读写锁

1. 写线程降级为读线程是可以的，但是从读线程升级为写线程是不可以的（会导致死锁）。