虚机synchronized析

当声明synchronized代码块时，编译而成的字节码将包含monitorenter和monitorexit指令。这两个指令都会消耗操作数栈上的一个引用类型元素，也就是synchronized关键字括号内的引用，作为所要加锁解锁的对象。

## Synchronized

抽象级锁

1.

**public void** foo(**Object** lock) {  
 // 代码块  
 **synchronized** (lock) {  
 lock.hashCode();  
 }  
}

$javap –v TestV1

public class cn.lxj.jvm40courses.course14.TestV1 {

public cn.lxj.jvm40courses.course14.TestV1();

Code:

0: aload\_0

1: invokespecial #1 // Method java/lang/Object."<init>":()V

4: return

public void foo(java.lang.Object);

Code:

0: aload\_1

1: dup

2: astore\_2

3: monitorenter

4: aload\_1

5: invokevirtual #2 // Method java/lang/Object.hashCode:()I

8: pop

9: aload\_2

10: monitorexit

11: goto 19

14: astore\_3

15: aload\_2

16: monitorexit

17: aload\_3

18: athrow

19: return

Exception table:

from to target type

4 11 14 any

14 17 14 any

}

2.

**public synchronized void** foo2(**Object** lock) {  
 lock.hashCode();  
}

public synchronized void foo2(java.lang.Object);

descriptor: (Ljava/lang/Object;)V

flags: ACC\_PUBLIC, ACC\_SYNCHRONIZED

Code:

stack=1, locals=2, args\_size=2

0: aload\_1

1: invokevirtual #2 // Method java/lang/Object.hashCode:()I

4: pop

5: return

LineNumberTable:

line 17: 0

line 18: 5

LocalVariableTable:

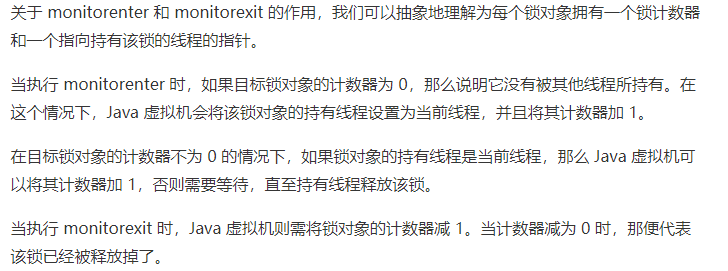
Start Length Slot Name Signature

0 6 0 this Lcn/lxj/jvm40courses/course14/TestV1;

0 6 1 lock Ljava/lang/Object;

这里的**monitorenter**和**monitorexit**操作对应的锁对象是隐式的。

对于实例方法来说，这两个操作对应的锁对象是this，对于静态方法来说，这两个操作对应的锁对象是所在类的Class实例。



## 重量级锁

重量级锁是java虚机中最为基础的锁实现。在这种状态下，java虚机会阻塞加锁失败的线程，并且在目标锁被释放的时候，唤醒这些线程。

Java虚机的阻塞以及唤醒，都是依靠操作系统来完成的。对于符合posix接口的操作系统来说（macos和绝大部分linux），上述操作是通过pthread（mutex）来实现的。此外，这些操作将涉及系统调用，需要从操作系统的用户态切换至内核态，开销非常大。

避免昂贵的阻塞和唤醒操作，虚机会在进入阻塞状态之前，以及被唤醒后竞争不到锁的情况下，进入自旋状态——在处理器上空跑并且轮询锁是否被释放，如果此时锁恰好被释放，那么当前线程无需阻塞，直接获得这把锁。

自旋会浪费大量的处理资源。

自适应自旋，避免过长时间的线程自旋，根据时长，来选择阻塞和自旋。根据以往自旋等待时是否能够获得锁来动态调整自旋的时间（循环数目）

自旋状态的副作用：

不公平的锁机制，处于阻塞状态的线程，无法立即竞争被释放的锁。

处于自旋状态的线程，很有可能获得这把锁。

## 轻量级锁

在不同的时间段请求同一把锁，也就是说没有锁竞争。采用轻量级锁，避免重量级锁的阻塞以及唤醒。

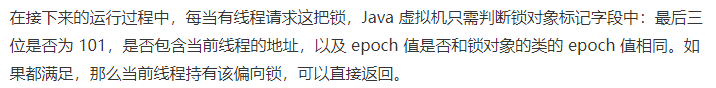
在Java对象头的标记字段，其最后两位被用来表示对象的锁状态。

1. 代表轻量锁，01-代表无锁（或偏向锁），10-代表重量级锁，11则跟垃圾回收算法的标记有关

## 偏向锁

从始至终只有一个线程请求某一把锁。

在线程进行加锁的时候，如果该锁对象支持偏向锁，那么java虚机会通过CAS操作，将当前线程的地址记录在锁对象的标记字段之中，并且将标记字段的最后三位设置为101.



Epoch值：当请求加锁的线程和锁对象标记的字段保持线程地址不匹配的时候（而且epoch值相等，如果不等，那么当前线程可以将该锁重偏向自己），java虚机需要撤销该偏向锁。撤销很麻烦，要求持有这个偏向锁的线程到达安全点，再将偏向锁替换为轻量级锁。

