这篇文章我们将深入的分析 synchronized 关键字在字节码层面是如何实现的

0x01 代码块级别的 synchronized

```
private Object lock = new Object();
public void foo() {
    synchronized (lock) {
        bar();
    }
}

public void bar() { }
```

编译成字节码如下

```
 public void foo();
    Code:
       0: aload_0
       1: getfield      #3                  //
Field lock:Ljava/lang/Object;
       4: dup
       5: astore_1

       6: monitorenter

       7: aload_0
       8: invokevirtual #4                  //
Method bar:()V

      11: aload_1
      12: monitorexit
      13: goto          21

      16: astore_2
      17: aload_1
      18: monitorexit
      19: aload_2
      20: athrow
      21: return
    Exception table:
       from    to  target type
           7    13    16   any
          16    19    16   any
```

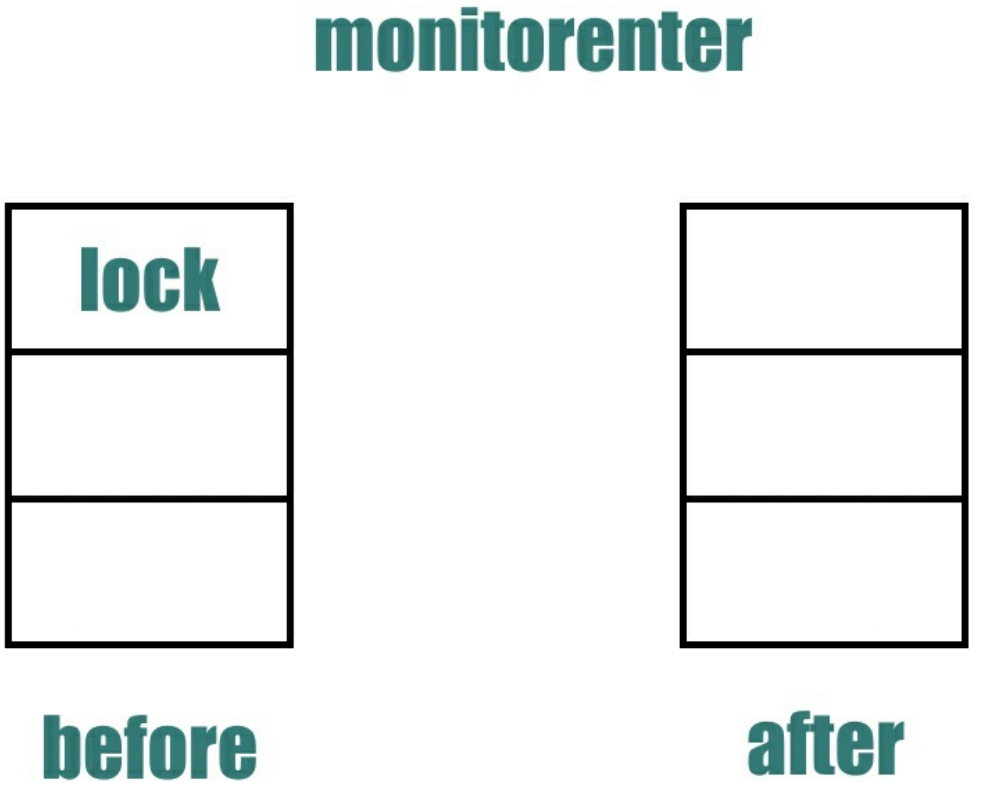
Java 虚拟机中代码块的同步是通过 monitorenter 和 monitorexit 两个支持 synchronized 关键字语意的。比如上面的字节码

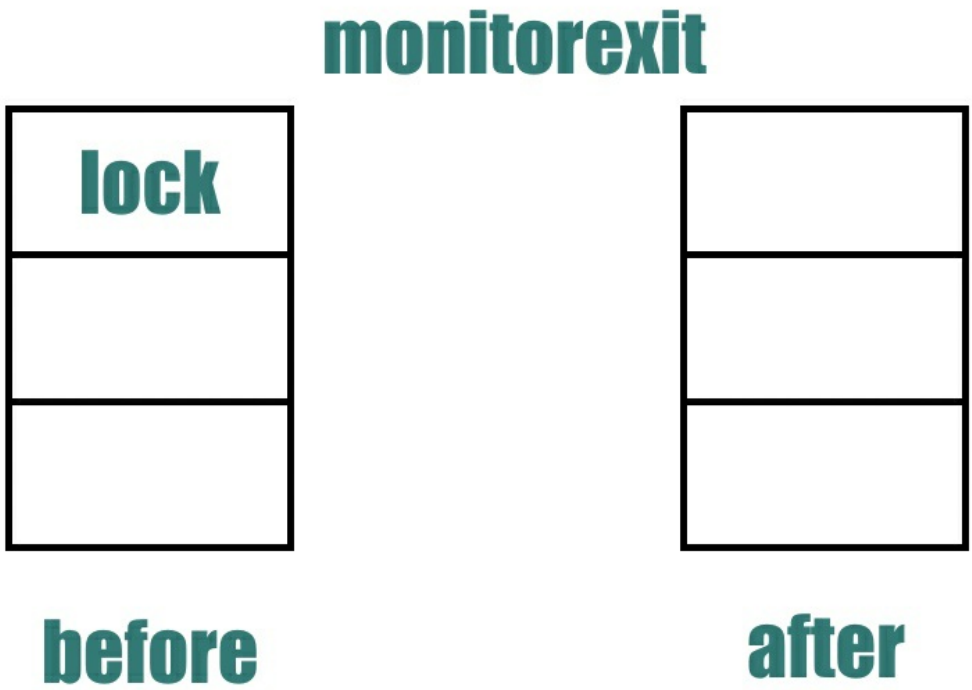
- 0 ~ 5：将 lock 对象入栈，使用 dup 指令复制栈顶元素，并

将它存入局部变量表位置 1 的地方，现在栈上还剩下一个 lock 对象

- 6：以栈顶元素 lock 做为锁，使用 monitorenter 开始同步
- 7 ~ 8：调用 bar() 方法
- 11 ~ 12：将 lock 对象入栈，调用 monitorexit 释放锁

monitorenter 对操作数栈的影响如下





- 16 ~ 20：执行异常处理，我们代码中本来没有 try-catch 的代码，为什么字节码会帮忙加上这段逻辑呢?

因为编译器必须保证，无论同步代码块中的代码以何种方式结束（正常 return 或者异常退出），代码中每次调用 monitorenter 必须执行对应的 monitorexit 指令。为了保证这一点，编译器会自动生成一个异常处理器，这个异常处理器的目的就是为了同步代码块抛出异常时能执行 monitorexit。这也是字节码中，只有一个 monitorenter 却有两个 monitorexit 的原因

可理解为这样的一段 Java 代码

```
public void _foo() throws Throwable {
    monitorenter(lock);
    try {
        bar();
    } finally {
        monitorexit(lock);
    }
}
```

根据我们之前介绍的 try-catch-finally 的字节码实现原理，复制 finally 语句块到所有可能函数退出的地方，上面的代码等价于

```
public void _foo() throws Throwable {
    monitorenter(lock);
    try {
        bar();
        monitorexit(lock);
    } catch (Throwable e) {
        monitorexit(lock);
        throw e;
    }
}
```

0x02 方法级的 synchronized

方法级的同步与上述有所不同，它是由常量池中方法的 ACC_SYNCHRONIZED 标志来隐式实现的。

```
synchronized public void testMe() {
}

对应字节码

public synchronized void testMe();
descriptor: ()V
flags: ACC_PUBLIC, ACC_SYNCHRONIZED
```

JVM 不会使用特殊的字节码来调用同步方法，当 JVM 解析方法的符号引用时，它会判断方法是不是同步的（检查方法 ACC_SYNCHRONIZED 是否被设置）。如果是，执行线程会先尝试获取锁。如果是实例方法，JVM 会尝试获取实例对象的锁，如果是类方法，JVM 会尝试获取类锁。在同步方法完成以后，不管是正常返回还是异常返回，都会释放锁

0x03 小结

这篇文章我们讲了 synchronized 关键字在字节码层面的实现细节，一起来回顾一下要点：第一，代码块级别的 synchronized 是使用 monitorenter、monitorexit 指令来实现的，monitorexit 会在所有可能退出的地方调用（正常退出、异常退出），以实现 monitorexit 一定会调用的语义。第二，方法级的 synchronized 是 JVM 隐式实现的，没有成对的 monitorenter-monitorexit 语句块。

0x04 思考

留一道作业题：monitorenter 和 monitorexit 底层做了什么？跟 Java 的对象头有什么关系?

欢迎你在留言区留言，和我一起讨论。