Synchronized的实现原理（一）

2017-12-02 Hollis Hollis

`synchronized`，是Java中用于解决并发情况下数据同步访问的一个很重要的关键字。当我们想要保证一个共享资源在同一时间只会被一个线程访问到时，我们可以在代码中使用`synchronized`关键字对类或者对象加锁。那么，本文来介绍一下`synchronized`关键字的实现原理是什么。在阅读本文之间，建议先看下Java虚拟机是如何执行线程同步的 。

反编译

众所周知，在Java中，`synchronized`有两种使用形式，同步方法和同步代码块。代码如下：

```
/**
 * @author Hollis 17/11/9.
 */
public class SynchronizedTest {

    public synchronized void doSth(){
        System.out.println("Hello World");
    }

    public void doSth1(){
        synchronized (SynchronizedTest.class){
            System.out.println("Hello World");
        }
    }
}
```

我们先来使用Javap来反编译以上代码，结果如下（部分无用信息过滤掉了）：

```
  public synchronized void doSth();
    descriptor: ()V
    flags: ACC_PUBLIC, ACC_SYNCHRONIZED
    Code:
      stack=2, locals=1, args_size=1
         0: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
         3: ldc           #3                  // String Hello World
         5: invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/Str
ing;)V
         8: return

  public void doSth1();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=3, args_size=1
         0: ldc           #5                  // class com/hollis/SynchronizedTest
         2: dup
         3: astore_1
         4: monitorenter
         5: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
         8: ldc           #3                  // String Hello World
        10: invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/Str
ing;)V
        13: aload_1
        14: monitorexit
        15: goto          23
        18: astore_2
        19: aload_1
```

```
      20: monitorexit
      21: aload_2
      22: athrow
      23: return
```

反编译后，我们可以看到Java编译器为我们生成的字节码。在对于 `doSth` 和 `doSth1` 的处理上稍有不同。也就是说。JVM对于同步方法和同步代码块的处理方式不同。

对于同步方法，JVM采用 `ACC_SYNCHRONIZED` 标记符来实现同步。 对于同步代码块。JVM采用 `monitorenter` 、 `monitorexit` 两个指令来实现同步。

关于这部分内容，在JVM规范中也可以找到相关的描述。

同步方法

The Java® Virtual Machine Specification中有关于方法级同步的介绍：

> Method-level synchronization is performed implicitly, as part of method invocation and return. A synchronized method is distinguished in the run-time constant pool's method*info structure by the ACC*SYNCHRONIZED flag, which is checked by the method invocation instructions. When invoking a method for which ACC_SYNCHRONIZED is set, the executing thread enters a monitor, invokes the method itself, and exits the monitor whether the method invocation completes normally or abruptly. During the time the executing thread owns the monitor, no other thread may enter it. If an exception is thrown during invocation of the synchronized method and the synchronized method does not handle the exception, the monitor for the method is automatically exited before the exception is rethrown out of the synchronized method.

主要说的是： 方法级的同步是隐式的。同步方法的常量池中会有一个 `ACC_SYNCHRONIZED` 标志。当某个线程要访问某个方法的时候，会检查是否有 `ACC_SYNCHRONIZED` ，如果有设置，则需要先获得监视器锁，然后开始执行方法，方法执行之后再释放监视器锁。这时如果其他线程来请求执行方法，会因为无法获得监视器锁而被阻断住。值得注意的是，如果在方法执行过程中，发生了异常，并且方法内部并没有处理该异常，那么在异常被抛到方法外面之前监视器锁会被自动释放。

同步代码块

同步代码块使用 `monitorenter` 和 `monitorexit` 两个指令实现。 The Java® Virtual Machine Specification 中有关于这两个指令的介绍：

monitorenter

> Each object is associated with a monitor. A monitor is locked if and only if it has an owner. The thread that executes monitorenter attempts to gain ownership of the monitor associated with objectref, as follows:
>
> > If the entry count of the monitor associated with objectref is zero, the thread enters the monitor and sets its entry count to one. The thread is then the owner of the monitor.
> >
> > If the thread already owns the monitor associated with objectref, it reenters the monitor, incrementing its entry count.
> >
> > If another thread already owns the monitor associated with objectref, the thread blocks until the monitor's entry count is zero, then tries again to gain ownership.

monitorexit

> The thread that executes monitorexit must be the owner of the monitor associated with the instance referenced by objectref.
>
> The thread decrements the entry count of the monitor associated with objectref. If as a result the value of the entry count is zero, the thread exits the monitor and is no longer its owner. Other threads that are blocking to enter the monitor are allowed to attempt to do so.

大致内容如下： 可以把执行 `monitorenter` 指令理解为加锁，执行 `monitorexit` 理解为释放锁。 每个对象维护着一个记录着被锁次数的计数器。未被锁定的对象的该计数器为0，当一个线程获得锁（执行 `monitorenter` ）后，该计数器自增变为 1 ，当同一个线程再次获得该对象的锁的时候，计数器再次自增。当同一个线程释放锁（执行 `monitorexit` 指令）的时候，计数器再自减。当计数器为0的时候。锁将被释放，其他线程便可以获得锁。

总结

> 同步方法通过 `ACC_SYNCHRONIZED` 关键字隐式的对方法进行加锁。当线程要执行的方法被标注上 `ACC_SYNCHRONIZED` 时，需要先获得锁才能执行该方法。
>
> 同步代码块通过 `monitorenter` 和 `monitorexit` 执行来进行加锁。当线程执行到 `monitorenter` 的时候要先获得所锁，才能执行后面的方法。当线程执行到 `monitorexit` 的时候则要释放锁。
>
> 每个对象自身维护这一个被加锁次数的计数器，当计数器数字为0时表示可以被任意线程获得锁。当计数器不为0时，只有获得锁的线程才能再次获得锁。即可重入锁。

至此，我们大致了解了Synchronized的原理。但是还有几个问题并没有介绍清楚，比如，Monitor到底是什么？对象的锁的状态保存在哪里？ 别急，后面会再介绍。

Top Comments

Write a comment

Fighting more
交流交流也能加深印象啊，谢谢
1-31

Reply by author
不客气。多多交流。
1-31

Fighting more
"是一个字节码的二进制文件"？就是字节码文件吧，只有JVM能读懂，然后通过JVM转换为二进制语言（也就是机器语言）。
1-31

Reply by author
对的，完全正确的。编译是javac的工作。javap可不能抢了他的活。哈哈哈。
1-31