21 如何看到 synchronized 背后的“monitor 锁”？

本课时我们研究下 synchronized 背后的 monitor 锁。

获取和释放 monitor 锁的时机

我们都知道，最简单的同步方式就是利用 synchronized 关键字来修饰代码块或者修饰一个方法，那么这部分被保护的代码，在同一时刻就最多只有一个线程可以运行，而 synchronized 的背后正是利用 monitor 锁实现的。所以首先我们来看下获取和释放 monitor 锁的时机，每个 Java 对象都可以用作一个实现同步的锁，这个锁也被称为内置锁或 monitor 锁，获得 monitor 锁的唯一途径就是进入由这个锁保护的同步代码块或同步方法，线程在进入被 synchronized 保护的代码块之前，会自动获取锁，并且无论是正常路径退出，还是通过抛出异常退出，在退出的时候都会自动释放锁。

我们首先来看一个 synchronized 修饰方法的代码的例子：

```
public synchronized void method() {
    method body
}
```

我们看到 method() 方法是被 synchronized 修饰的，为了方便理解其背后的原理，我们把上面这段代码改写为下面这种等价形式的伪代码。

```
public void method() {
    this.intrinsicLock.lock();
    try{
        method body
    }
    finally {
        this.intrinsicLock.unlock();
```

```
    }
}
```

在这种写法中，进入 method 方法后，立刻添加内置锁，并且用 try 代码块把方法保护起来，最后用 finally 释放这把锁，这里的 intrinsicLock 就是 monitor 锁。经过这样的伪代码展开之后，相信你对 synchronized 的理解就更加清晰了。

用 javap 命令查看反汇编的结果

JVM 实现 synchronized 方法和 synchronized 代码块的细节是不一样的，下面我们就分别来看一下两者的实现。

同步代码块

首先我们来看下同步代码块的实现，如代码所示。

```
public class SynTest {
    public void synBlock() {
        synchronized (this) {
            System.out.println("lagou");
        }
    }
}
```

在 SynTest 类中的 synBlock 方法，包含一个同步代码块，synchronized 代码块中有一行代码打印了 lagou 字符串，下面我们来通过命令看下 synchronized 关键字到底做了什么事情：首先用 cd 命令切换到 SynTest.java 类所在的路径，然后执行 javac SynTest.java，于是就会产生一个名为 SynTest.class 的字节码文件，然后我们执行 javap -verbose SynTest.class，就可以看到对应的反汇编内容。

关键信息如下：

```
  public void synBlock();
    descriptor: ()V
    flags: ACC_PUBLIC
```

```
Code:
  stack=2, locals=3, args_size=1
     0: aload_0
     1: dup
     2: astore_1
     3: monitorenter
     4: getstatic     #2                  // Field java/lang/System.out:Ljava/i
     7: ldc           #3                  // String lagou
     9: invokevirtual #4                  // Method java/io/PrintStream.println:(L
    12: aload_1
    13: monitorexit
    14: goto          22
    17: astore_2
    18: aload_1
    19: monitorexit
    20: aload_2
    21: athrow
    22: return
```

从里面可以看出，synchronized 代码块实际上多了 monitorenter 和 monitorexit 指令，标红的第3、13、19行指令分别对应的是 monitorenter 和 monitorexit。这里有一个 monitorenter，却有两个 monitorexit 指令的原因是，JVM 要保证每个 monitorenter 必须有与之对应的 monitorexit，monitorenter 指令被插入到同步代码块的开始位置，而 monitorexit 需要插入到方法正常结束处和异常处两个地方，这样就可以保证抛异常的情况下也能释放锁

可以把执行 monitorenter 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，我们来具体看一下 monitorenter 和 monitorexit 的含义：

- monitorenter

执行 monitorenter 的线程尝试获得 monitor 的所有权，会发生以下这三种情况之一：

a. 如果该 monitor 的计数为 0，则线程获得该 monitor 并将其计数设置为 1。然后，该线程就是这个 monitor 的所有者。

b. 如果线程已经拥有了这个 monitor ，则它将重新进入，并且累加计数。

c. 如果其他线程已经拥有了这个 monitor，那个这个线程就会被阻塞，直到这个 monitor 的计数变成为 0，代表这个 monitor 已经被释放了，于是当前这个线程就会再次尝试获取这个 monitor。

- monitorexit monitorexit 的作用是将 monitor 的计数器减 1，直到减为 0 为止。代表这个 monitor 已经被释放了，已经没有任何线程拥有它了，也就代表着解锁，所以，其他正在等待这个 monitor 的线程，此时便可以再次尝试获取这个 monitor 的所有权。

同步方法

从上面可以看出，同步代码块是使用 monitorenter 和 monitorexit 指令实现的。而对于 synchronized 方法，并不是依靠 monitorenter 和 monitorexit 指令实现的，被 javap 反汇编后可以看到，synchronized 方法和普通方法大部分是一样的，不同在于，这个方法会有一个叫作 ACC_SYNCHRONIZED 的 flag 修饰符，来表明它是同步方法。

同步方法的代码如下所示。

```
public synchronized void synMethod() {

}
```

对应的反汇编指令如下所示。

```
public synchronized void synMethod();
  descriptor: ()V
  flags: ACC_PUBLIC, ACC_SYNCHRONIZED
  Code:
    stack=0, locals=1, args_size=1
       0: return
    LineNumberTable:
      line 16: 0
```

可以看出，被 synchronized 修饰的方法会有一个 ACC_SYNCHRONIZED 标志。当某个线程要访问某个方法的时候，会首先检查方法是否有 ACC_SYNCHRONIZED 标志，如果有则需要先获得 monitor 锁，然后才能开始执行方法，方法执行之后再释放 monitor 锁。其他方面， synchronized 方法和刚才的 synchronized 代码块是很类似的，例如这时如果其他线程来请求执行方法，也会因为无法获得 monitor 锁而被阻塞。

好了，本课时的内容就全部讲完了，本课时我们讲解了获取和释放 monitor 的时机，以及被 synchronized 修饰的等价代码，然后我们还利用 javac 和 javap 命令查看了 synchronized 代码块以及 synchronized 方法所对应的的反汇编指令，其中同步代码块是利用 monitorenter 和 monitorexit 指令实现的，而同步方法则是利用 flags 实现的。

上一页 下一页