<https://mp.weixin.qq.com/s/pE1_gDkTpHsbJCgbptG4Qw>

# 这回，阿粉要将 synchronized 与锁的关系讲透！！！

# **synchronized 关键字**

说到锁，都会提 synchronized 。这个英文单词儿啥意思呢？翻译成中文就是「同步」的意思

一般都是使用 synchronized 这个关键字来给一段代码或者一个方法上锁，使得这段代码或者方法，在同一个时刻只能有一个线程来执行它。

synchronized 相比于 volatile 来说，用的比较灵活，你可以在方法上使用，可以在静态方法上使用，也可以在代码块上使用。

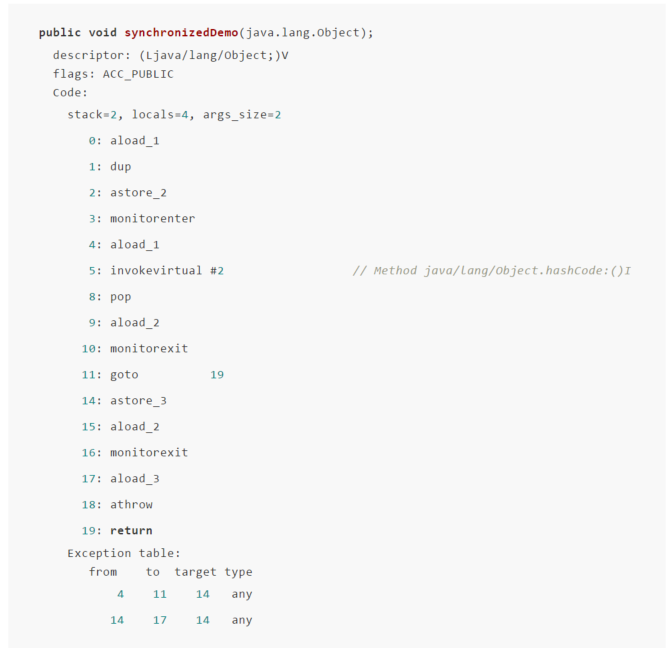
关于 synchronized 这一块大概就说到这里，阿粉今天想着重来说一下， synchronized 底层是怎么实现的

## **JVM 是如何实现 synchronized 的？**

我知道可以利用 synchronized 关键字来给程序进行加锁，但是它具体怎么实现的我不清楚呀，别急，咱们先来看个 demo :

**public** **class** **demo** {  
    **public** **void** **synchronizedDemo**(Object lock){  
  **synchronized**(lock){  
   lock.hashCode();  
  }  
 }  
}

上面是我写的一个 demo ，然后进入到 class 文件所在的目录下，使用 javap -v demo.class 来看一下编译的字节码(在这里我截取了一部分):



应该能够看到当程序声明 synchronized 代码块时，编译成的字节码会包含 monitorenter和 monitorexit 指令，这两种指令会消耗操作数栈上的一个引用类型的元素(也就是 synchronized 关键字括号里面的引用)，作为所要加锁解锁的锁对象。如果看的比较仔细的话，上面有一个 monitorenter 指令和两个 monitorexit 指令，这是 Java 虚拟机为了确保获得的锁不管是在正常执行路径，还是在异常执行路径上都能够解锁。

关于 monitorenter 和 monitorexit ，可以理解为每个锁对象拥有一个锁计数器和一个指向持有该锁的线程指针:

* 当程序执行 monitorenter 时，如果目标锁对象的计数器为 0 ，说明这个时候它没有被其他线程所占有，此时如果有线程来请求使用， Java 虚拟机就会分配给该线程，并且把计数器的值加 1
  + 目标锁对象计数器不为 0 时，如果锁对象持有的线程是当前线程， Java 虚拟机可以将其计数器加 1 ，如果不是呢？那很抱歉，就只能等待，等待持有线程释放掉
* 当执行 monitorexit 时， Java 虚拟机就将锁对象的计数器减 1 ，当计数器减到 0 时，说明这个锁就被释放掉了，此时如果有其他线程来请求，就可以请求成功

为什么采用这种方式呢？是为了允许同一个线程重复获取同一把锁。比如，一个 Java 类中拥有好多个 synchronized 方法，那这些方法之间的相互调用，不管是直接的还是间接的，都会涉及到对同一把锁的重复加锁操作。这样去设计的话，就可以避免这种情况。

# **锁**

在 Java 多线程中，所有的锁都是基于对象的。也就是说， Java 中的每一个对象都可以作为一个锁。你可能会有疑惑，不对呀，不是还有类锁嘛。但是 class 对象也是特殊的 Java 对象，所以呢，在 Java 中所有的锁都是基于对象的

在 Java6 之前，所有的锁都是"重量级"锁，重量级锁会带来一个问题，就是如果程序频繁获得锁释放锁，就会导致性能的极大消耗。为了优化这个问题，引入了"偏向锁"和"轻量级锁"的概念。所以在 Java6 及其以后的版本，一个对象有 4 种锁状态:无锁状态，偏向锁状态，轻量级锁状态，重量级锁状态。

在 4 种锁状态中，无锁状态应该比较好理解，无锁就是没有锁，任何线程都可以尝试修改，所以这里就一笔带过了。

随着竞争情况的出现，锁的升级非常容易发生，但是如果想要让锁降级，条件非常苛刻，有种你想来可以，但是想走不行的赶脚。

阿粉在这里啰嗦一句:很多文章说，锁如果升级之后是不能降级的，其实在 HotSpot JVM 中，是支持锁降级的

锁降级发生在 Stop The World 期间，当 JVM 进入安全点的时候，会检查有没有闲置的锁，如果有就会尝试进行降级

看到 Stop The World 和 安全点 可能有人比较懵，我这里简单说一下，具体还需要读者自己去探索一番.(因为这是 JVM 的内容，这篇文章的重点不是 JVM )

在 Java 虚拟机里面，传统的垃圾回收算法采用的是一种简单粗暴的方式，就是 Stop-the-world ，而这个 Stop-the-world 就是通过安全点( safepoint )机制来实现的，安全点是什么意思呢？就是 Java 程序在执行本地代码时，如果这段代码不访问 Java 对象/调用 Java 方法/返回到原来的 Java 方法，那 Java 虚拟机的堆栈就不会发生改变，这就代表执行的这段本地代码可以作为一个安全点。当 Java 虚拟机收到 Stop-the-world 请求时，它会等所有的线程都到达安全点之后，才允许请求 Stop-the-world 的线程进行独占工作

接下来就介绍一下几种锁和锁升级

## **Java 对象头**

在刚开始就说了， Java 的锁都是基于对象的，那是怎么告诉程序我是个锁呢？就不得不来说， Java 对象头每个 Java 对象都有对象头，如果是非数组类型，就用 2 个字宽来存储对象头，如果是数组，就用 3 个字宽来存储对象头。在 32 位处理器中，一个字宽是 32 位;在 64 位处理器中，字宽就是 64 位咯~对象头的内容就是下面这样:

| **长度** | **内容** | **说明** |
| --- | --- | --- |
| 32/64 bit | Mark Word | 存储对象的 hashCode 或锁信息等 |
| 32/64 bit | Class Metadata Address | 存储到对象类型数据的指针 |
| 32/64 bit | Array length | 数组的长度(如果是数组) |

咱们主要来看 Mark Word 的内容:

| **锁状态** | **29 bit/61 bit** | **1 bit 是否是偏向锁** | **2 bit 锁标志位** |
| --- | --- | --- | --- |
| 无锁 |  | 0 | 01 |
| 偏向锁 | 线程 ID | 1 | 01 |
| 轻量级锁 | 指向栈中锁记录的指针 | 此时这一位不用于标识偏向锁 | 00 |
| 重量级锁 | 指向互斥量(重量级锁)的指针 | 此时这一位不用于标识偏向锁 | 10 |
| GC 标记 |  | 此时这一位不用于标识偏向锁 | 11 |

从上面表格中，应该能够看到，是偏向锁时， Mark Word 存储的是偏向锁的线程 ID ；是轻量级锁时， Mark Word 存储的是指向线程栈中 Lock Record 的指针；是重量级锁时， Mark Word 存储的是指向堆中的 monitor 对象的指针。