参考：http://blog.csdn.net/m\_xiaoer/article/details/73274642

[**关于synchronized的Monitor Object机制的研究**](http://blog.csdn.net/m_xiaoer/article/details/73274642)

这是一道面试题

* Synchronized 介绍
* monitor Object 设计模式
* Monitor(java多线程同步机制)
* Refer

synchronized介绍

synchronized关键字通过修饰一个方法或声明一个代码块，从而产生一个同步对象锁以及对应的同步代码块。

每当有线程要对该同步代码块进行访问时，线程就会首先尝试去获取该对象锁，并在成功获取到对象锁后，对该同步代码块进行正常访问，在同步代码块访问过程中，线程会一直持有该对象锁直到同步代码块访问完毕才会释放。

在上述线程持有同步锁并进行同步代码块访问过程中，其它线程将无法获得该对象锁，也无法访问该同步代码，这些线程都会被阻塞直到上述线程访问完毕。syschronized关键字，通过以上措施，确保每次只有一个线程能持有对象锁并对同步代码块进行访问，并在访问结束之前，不会有其它线程对其进行访问。

也就说，即使同步代码块在执行过程中遭遇线程调度，其它线程也无法访问该同步代码块，直到该线程被重新调度并完成同步代码块的访问并释放对象锁。

这样就保证了线程对同步代码块访问的连续性不受线程调度而中断。

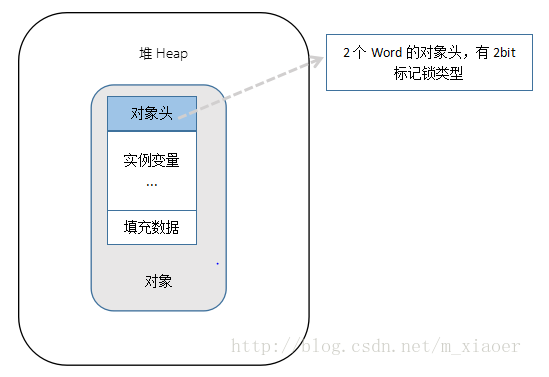
**synchronized被称为重量级的锁，它的同步包括：**

* 对于普通方法同步，锁是当前实例对象
* 对于静态方法同步，锁是当前类的 Class 对象
* 对于方法块同步，锁是 Synchronized 括号里的对象

上述都是对象级别的锁，当一个线程访问对象中的同步方法时，会获取到对象级别的锁，由于 Synchronized 内部是可重入的互斥锁，所以线程可再次重入用 Synchronized 修饰的方法，但当其它线程执行同一个对象的带有 Synchronized 的方法时，会被阻塞，即使和以持有对象锁的线程执行的相同对象的不同 Synchronized 方法。因为锁是对象级别的。比如线程 A、B。对象 Foo 有同步方法 M、N。线程 A 首先执行同步方法 M 时就会获取对象锁，此时 B 不能执行同一把对象锁修饰的方法 M、N。除非 A 释放锁。又因为锁是可重入的，所以 A 可以继续执行 M，N 方法。可重入锁一定程度上避免了死锁的问题，内部是关联一个计数器，加一次锁计数器值加一，为零时释放锁。

**如何理解锁是“对象”?**

Java 编程语言中号称一切皆对象。当我们 new 一个对象的时候 JVM 会给 heap 中分配对象。如下图：



(对象头)这个头包括两个部分，第一部分用于存储自身运行时的数据例如GC标志位、哈希码、锁状态 等信息。第二部分存放指向方法区类静态数据的指针。锁状态就是用来同步操作的 bit 位。因为锁信息是存储在对象上的，所以就不难理解锁是对象这句话了。

**那么 Java 为什么要将锁内置到对象中呢？这要从 monitor Object 设计模式说起。**

monitor Object 设计模式

**问题描述：**

我们在开发并发的应用时，经常需要设计这样的对象，该对象的方法会在多线程的环境下被调用，而这些方法的执行都会改变该对象本身的状态。为了防止竞争条件 (race condition) 的出现，对于这类对象的设计，需要考虑解决以下问题：

* 在任一时间内，只有唯一的公共的成员方法，被唯一的线程所执行。
* 对于对象的调用者来说，如果总是需要在调用方法之前进行拿锁，而在调用方法之后进行放锁，这将会使并发应用编程变得更加困难。
* 如果一个对象的方法执行过程中，由于某些条件不能满足而阻塞，应该允许其它的客户端线程的方法调用可以访问该对象。

我们使用 Monitor Object 设计模式来解决这类问题：将被客户线程并发访问的对象定义为一个 monitor 对象。客户线程仅能通过 monitor 对象的同步方法才能使用 monitor 对象定义的服务。为了防止陷入竞争条件，在任一时刻只能有一个同步方法被执行。每一个 monitor 对象包含一个 monitor 锁，被同步方法用于串行访问对象的行为和状态。此外，同步方法可以根据一个或多个与 monitor 对象相关的 monitor conditions 来决定在何种情况下挂起或恢复他们的执行。

**结构**

在 Monitor Object 模式中，主要有四种类型的参与者：

* 监视者对象 (Monitor Object): 负责定义公共的接口方法，这些公共的接口方法会在多线程的环境下被调用执行。
* 同步方法：这些方法是监视者对象所定义。为了防止竞争条件，无论是否同时有多个线程并发调用同步方法，还是监视者对象含有多个同步方法，在任一时间内只有监视者对象的一个同步方法能够被执行。
* 监视锁 (Monitor Lock): 每一个监视者对象都会拥有一把监视锁。
* 监视条件 (Monitor Condition): 同步方法使用监视锁和监视条件来决定方法是否需要阻塞或重新执行。

**执行序列图**

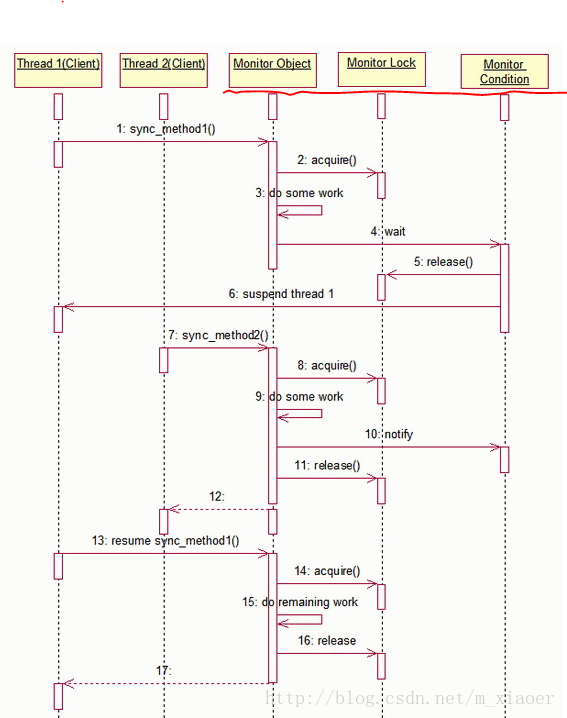
在监视者对象模式中，在参与者之间将发生如下的协作过程：

1、同步方法的调用和串行化。当客户线程调用监视者对象的同步方法时，必须首先获取它的监视锁。只要该监视者对象有其他同步方法正在被执行，获取操作便不会成功。在这种情况下，客户线程将被阻塞直到它获取监视锁。当客户线程成功获取监视锁后，进入临界区，执行方法实现的服务。一旦同步方法完成执行，监视锁会被自动释放，目的是使其他客户线程有机会调用执行该监视者对象的同步方法。

2、同步方法线程挂起。如果调用同步方法的客户线程必须被阻塞或是有其他原因不能立刻进行，它能够在一个监视条件上等待，这将导致该客户线程暂时释放监视锁，并被挂起在监视条件上。

3、监视条件通知。一个客户线程能够通知一个监视条件，目的是为了让一个前期使自己挂起在一个监视条件上的同步方法线程恢复运行。

4、同步方法线程恢复。一旦一个早先被挂起在监视条件上的同步方法线程获取通知，它将继续在最初的等待监视条件的点上执行。在被通知线程被允许恢复执行同步方法之前，监视锁将自动被获取。图中描述了监视者对象的动态特性。



其实， monitor object 设计模式执行时序图中的红线部分 Monitor Object、Monitor Lock、Monitor Condition 三者就是 Java Object！！ Java 将该模式内置到语言层面，对象加 Synchronized 关键字，就能确保任何对它的方法请求的同步被透明的进行，而不需要调用者的介入。

这也就是为什么 Java 所有对象的基类 Object 中会有 wait()、notify()、notifyAll() 方法了。

Monitor(java多线程同步机制)

为了达到同步，java在一个监控器(Monitor)的基础上实现了一个巧妙的方案。

监控器是一个控制机制，可以认为是一个很小的、只能容纳一个线程的盒子。

一旦一个线程进入监控器，其它的线程必须等待，直到那个线程退出监控为止。

通过这种方式，一个监控器可以保证共享资源在同一时刻只可被一个线程使用，这种方式称为同步。

一旦一个线程进入一个实例的任何同步方法，别的线程将不能进入该同一实例的其它同步方法，但是该实例的非同步方法仍然能够被调用。

Refer

1、[探索 Java 同步机制](https://www.ibm.com/developerworks/cn/java/j-lo-synchronized/)

2、 [Java中的synchronized、Object.wait()、Object.notify()/notifyAll()原理](http://blog.csdn.net/achellies/article/details/7094467)

3、 [Java 锁机机制——浅析 Synchronized](http://blog.csdn.net/wwh578867817/article/details/52004265)