synchronized各种用法/特性总结

# synchronized锁对象：

## synchronizedd锁对象的两种方式：

一种是在方法里面使用synchronizedd代码块：

**public void** testSyncObj() {  
 **synchronized** (**this**) {  
 System.***out***.println(**"锁对象的方式一：synchronized代码块"**);  
 }  
}

还有一种是直接定义在方法修饰符上：

**public synchronized void** testSyncObj2() {  
 System.***out***.println(**"锁对象的方式二：synchronized修饰方法"**);  
}

以上两种情况，关键字synchronized取得的是对象锁，而不是把一段代码或方法当做锁，这个一定要记住。锁对象的时候，哪个线程先执行带synchronized关键字的方法或代码块，哪个线程就持有该对象所属方法的锁，其他线程只能呈等待状态，前提是多个线程访问的是同一个对象！如果多个线程访问的是多个对象，那么synchronized不会起作用，因为他们根本不存在同时争抢某一个对象的锁。

还需要记住，对象中没有加synchronized修饰的方法是可以在任何时间点调用的。

## 脏读

虽然在赋值时进行了同步，但是在取值时有可能出现一些意想不到的惊喜，这种现象就是脏读（DirtyRead）。发生脏读的情况是在取实例变量时，此值已经被其他线程更改过了。

出现脏读很明显是同步没有做到位，只做了一部分，没有顾及另一部分。在写的时候做了同步，在读的时候却没有，而上面讲到过，对象出了synchronized之外的其他方法可以随时调用，所以我们在读的时候也需要加上synchronized修饰，这样就能保证读和写一致。

下面解释一下某个对象被多个线程争抢的情况：

*1）当A线程调用anyObject对象加入synchronized关键字的X方法时，A线程就获得了X方法所在对象的锁，所以其他线程必须等待A线程执行完这个方法之后才可以争抢到对象锁执行X方法。但是B线程可以随意调用其他的非synchronized修饰的方法。*

*2）当A线程调用anyObject对象加入synchronized关键字的X方法时，A线程就获得了X方法所在对象的锁，所以其他线程必须等待A线程执行完这个方法之后才可以争抢到对象锁执行X方法。 而B线程如果调用声明了synchronized关键字的非X方法时，必须等待A线程将X方法执行完，也就是说必须等A释放了对象的锁之后，才可以调用。这时候A已经执行了一个完整的任务，不存在脏读了。*

## synchronized锁重入

关键字synchronized具有锁重入的功能，也就是在使用synchronized时，当一个线程得到一个对象锁后，再次请求此对象的锁时是可以再次得到该对象的锁的。**这也证明在一个synchronized方法/块内调用其他synchronized方法/块是永远可以得到锁的**。

“可重入锁”的特性：

**自己可以再次获取自己的内部锁**。比如某个线程获得了某个对象的锁，此时还没有释放这个对象的锁，那么这个线程此时再调用其他同步块或同步方法的时候还是可以继续得到这个对象的锁的，不然不就死锁了吗。

当存在父子集成关系时，子类是完全可以通过“可重入锁”调用父类的同步块或同步方法的

## 自动释放锁

当一个线程执行的代码出现异常时，其所持有的锁会自动释放。

## 同步不具有继承性

同步不可以继承。父类有synchronized修饰的方法，子类还需要加入synchronized修饰，不然无法做到同步。

## synchronized同步语句块比synchronized修饰方法的优势

用关键字synchronized修饰方法简单粗暴，可以很方便的达到同步效果，但是修饰方法往往是有弊端的：一是对性能的影响，方法体通常很长，其他对象都要等得到锁的那个对象释放锁可能是一个漫长的过程。二是灵活性没有锁代码块好，synchronized修饰方法就是给方法所在的对象上锁，而synchronized代码块有一个参数，可以传一个对象进去，这个对象可以是this对象，也可以是其他对象，传入的对象也就是需要上锁的对象。

当一个线程访问某个对象的同步代码块时，另一个线程仍然可以访问这个对象中非同步代码块部分。也就是说：不在synchronized代码块中就是异步执行，在synchronized代码块中就是同步执行。

但是还有一个问题，在一个方法里面，如果同步代码块在中间，那么多个线程调用的时序是怎样的呢？答案是，在synchronized同步代码块之前的代码会异步执行，不受时间控制，但是在synchronized中的代码以及synchronized代码块之后的代码都需要排队的。

## synchronized代码块间的同步性

在使用同步synchronized(this) 代码块的时候需要注意，当一个线程访问一个对象的synchronized同步代码块的时候，其他线程在这个时候也访问这个对象中的其他synchronized修饰的方法/代码块都将被阻塞，直到第一个线程释放锁。这说明这些synchronized使用的是同一个对象监视器，监视器就是this对象（如果不是传入的this，同理监视器就是其他传入的对象）。

## synchronized(obj) 将任意对象作为对象监视器

在上面讲述的概念中，基本都是用this作为对象监视器（synchronized修饰方法，或者synchronized (this) ），这种用法最常用，也最直接粗暴，但极端情况下，我们也可以传入其他对象作为对象监视器。这个“其他对象”可以是任意对象，大多数是实例变量或方法的参数，使用格式为：synchronized (obj) ---obj可以是任意对象。在某些需要共享一个特殊对象或数据时，可以这样做同步。

将任意对象作为对象监视器有以下特性：

当指定了obj(任意对象)做为对象监视器，并且这个时候有多个线程同时争抢这个对象监视器的锁（即要访问这个对象监视器的synchronized代码块），那么同时只有一个线程可以执行synchronized(obj) 同步代码块中的代码。

将任意对象作为对象监视器有以下优点：

如果一个类中有很多synchronized方法，这无疑是非常影响效率的，但如果用同步代码块锁非this对象，则synchronized(非this) 代码块中的程序与这个类中的同步方法是异步的，因为同步方法是锁this对象。这样可以大大提高效率。

synchronized(非this) 与synchronized(this) 性质原理很相似，可以一样去理解，**我们只需要牢牢抓住谁是对象监视器**，也就是括号里面传入的对象是谁。弄清了对象监视器之后，再看**是不是有多个线程来争抢这个对象监视器**（被上锁的对象），如果有，则会同步执行。一定要切记锁的是哪个对象，有时候多个线程可能会实例化多个对象监视器，这样也就不存在争抢问题了，自然不会同步执行，下面我会讲一下锁对象和锁class的区别。

# synchronized锁静态方法及synchronized(class) 代码块

关键字synchronized还可以修饰静态方法，如果修饰静态方法，那就表示给当前Java类进行上锁（也就是Class）。

用法实例：

锁静态代码块：

**public static void** testSyncStaticBlock() {  
 **synchronized** (SyncStaticTest**.class**) {  
 System.***out***.println(**"锁静态代码块"**);  
 }  
}

锁静态方法：

**synchronized public static void** testSyncStaticMethod() {  
 System.***out***.println(**"锁静态方法"**);  
}

给Class类上锁和给this对象上锁是有本质不同的，举个例子说明一下：

条件：

假设有一个Java类ClassA, 里面定义了synchronized修饰的静态方法staticMethodA, 和非静态方法methodB。

用ClassA 创建两个实例ClassObjA 和 ClassObjB。

创建两个线程threadA和threadB。

实验：

1. 用threadA调用classObjA的methodB方法，同时用threadB调用classObjB的methodB方法，发现执行并不是同步的，因为在动态方法中，实例classObjeA的methodB方法的this是指的classObjA，classObjeB的methodB方法this是指的classObjB，上面反复提到过，我们一定要弄清楚对象监视器的指针是指向谁，这很明显是两个对象，压根不存在对象争抢的问题，所以当然不会同步执行。

2. 用threadA和threadB都调用ClassA.staticMethodA方法, 发现是同步执行的，因为synchronized锁的是静态代码块，所以跟对象就没什么关系了，不管哪个线程，只要你调用我ClassA的这个静态方法，都必须排队。

上面两个实验可以很清晰的看到锁class静态代码块和锁对象的区别。

*插个广告：下节我们将介绍volatile的各种用法。*