Java同步块

原文链接 **作者**：Jakob Jenkov **译者**：李同杰

Java 同步块（synchronized block）用来标记方法或者代码块是同步的。Java同步块用来避免竞争。本文介绍以下内容：

Java 同步关键字（synchronized）

Java中的同步块用synchronized标记。同步块在Java中是同步在某个对象上。所有同步在一个对象上的同步块在同时只能被一个线程进入并执行操作。所有其他等待进入该同步块的线程将被阻塞，直到执行该同步块中的线程退出。

有四种不同的同步块：

1. 实例方法
2. 静态方法
3. 实例方法中的同步块
4. 静态方法中的同步块

上述同步块都同步在不同对象上。实际需要那种同步块视具体情况而定。

实例方法同步

下面是一个同步的实例方法：

```
1 public synchronized void add(int value){
2 this.count += value;
3  }
```

注意在方法声明中同步（synchronized ）关键字。这告诉Java该方法是同步的。

Java实例方法同步是同步在拥有该方法的对象上。这样，每个实例其方法同步都同步在不同的对象上，即该方法所属的实例。只有一个线程能够在实例方法同步块中运行。如果有多个实例存在，那么一个线程一次可以在一个实例同步块中执行操作。一个实例一个线程。

静态方法同步

静态方法同步和实例方法同步方法一样，也使用synchronized 关键字。Java静态方法同步如下示例：

```
public static synchronized void add(int value){
 count += value;
 }
```

同样，这里synchronized 关键字告诉Java这个方法是同步的。

静态方法的同步是指同步在该方法所在的类对象上。因为在Java虚拟机中一个类只能对应一个类对象，所以同时只允许一个线程执行同一个类中的静态同步方法。

对于不同类中的静态同步方法，一个线程可以执行每个类中的静态同步方法而无需等待。不管类中的那个静态同步方法被调用，一个类只能由一个线程同时执行。

实例方法中的同步块

有时你不需要同步整个方法，而是同步方法中的一部分。Java可以对方法的一部分进行同步。

在非同步的Java方法中的同步块的例子如下所示：

```
public void add(int value){

    synchronized(this){
       this.count += value;
    }
  }
```

示例使用Java同步块构造器来标记一块代码是同步的。该代码在执行时和同步方法一样。

注意Java同步块构造器用括号将对象括起来。在上例中，使用了“this”，即为调用add方法的实例本身。在同步构造器中用括号括起来的对象叫做监视器对象。上述代码使用监视器对象同步，同步实例方法使用调用方法本身的实例作为监视器对象。

一次只有一个线程能够在同步于同一个监视器对象的Java方法内执行。

下面两个例子都同步他们所调用的实例对象上，因此他们在同步的执行效果上是等效的。

```
public class MyClass {

```

```
    public synchronized void log1(String msg1, String msg2){
       log.writeln(msg1);
       log.writeln(msg2);
    }

    public void log2(String msg1, String msg2){
       synchronized(this){
          log.writeln(msg1);
          log.writeln(msg2);
       }
    }
 }
```

在上例中，每次只有一个线程能够在两个同步块中任意一个方法内执行。

如果第二个同步块不是同步在this实例对象上，那么两个方法可以被线程同时执行。

静态方法中的同步块

和上面类似，下面是两个静态方法同步的例子。这些方法同步在该方法所属的类对象上。

```
public class MyClass {
    public static synchronized void log1(String msg1, String msg2){
       log.writeln(msg1);
       log.writeln(msg2);
    }

    public static void log2(String msg1, String msg2){
       synchronized(MyClass.class){
          log.writeln(msg1);
          log.writeln(msg2);
       }
    }
  }
```

这两个方法不允许同时被线程访问。

如果第二个同步块不是同步在MyClass.class这个对象上。那么这两个方法可以同时被线程访问。

Java同步实例

在下面例子中，启动了两个线程，都调用Counter类同一个实例的add方法。因为同步在该方法所属的实例上，所以同时只能有一个线程访问该方法。

```
public class Counter{
     long count = 0;

     public synchronized void add(long value){
       this.count += value;
     }
  }
  public class CounterThread extends Thread{

     protected Counter counter = null;

     public CounterThread(Counter counter){
        this.counter = counter;
     }
```

```
15
16      public void run() {
17     for(int i=0; i<10; i++){
18             counter.add(i);
19          }
20       }
21   }
22   public class Example {
23
24     public static void main(String[] args){
25       Counter counter = new Counter();
26       Thread  threadA = new CounterThread(counter);
27       Thread  threadB = new CounterThread(counter);
28
29       threadA.start();
30       threadB.start();
31     }
32   }
```

创建了两个线程。他们的构造器引用同一个Counter实例。Counter.add方法是同步在实例上，是因为add方法是实例方法并且被标记上synchronized关键字。因此每次只允许一个线程调用该方法。另外一个线程必须要等到第一个线程退出add()方法时，才能继续执行方法。

如果两个线程引用了两个不同的Counter实例，那么他们可以同时调用add()方法。这些方法调用了不同的对象，因此这些方法也就同步在不同的对象上。这些方法调用将不会被阻塞。如下面这个例子所示：

```
01 public class Example {
02
03    public static void main(String[] args){
04      Counter counterA = new Counter();
05      Counter counterB = new Counter();
06      Thread  threadA = new CounterThread(counterA);
07      Thread  threadB = new CounterThread(counterB);
08
09      threadA.start();
10      threadB.start();
11    }
12  }
```

注意这两个线程，threadA和threadB，不再引用同一个counter实例。CounterA和counterB的add方法同步在他们所属的对象上。调用counterA的add方法将不会阻塞调用counterB的add方法。

萍水相逢
2013/04/16 10:34上午

关于 Java同步实例 中的第一个例子 ： 每次执行的在Counter类add方法中输出 count 结果都不一样，

public class Counter {

long count = 0;

public synchronized void add(long value){

this.count += value;

System.out.println(“count:==== “+count);

}

}

请教一下，不是应该每次执行的结果一样吗？

李同杰
2013/04/16 10:26下午

您好，过程的值有可能不一样，但是最终结果会是一样，都是90。原因在于CounterThread类中的run 方法中的for 循环，这里synchronized add方法能保证counter类中add方法对于同一个实例对象是同步的，但for循环不是同步的，就是有可能threadA执行一部分for循环，然后threadB执行一部分for循环，两个线程交替执行。也有可能threadA执行完for循环，然后threadB执行for循环。具体情况是随机的。所以会导致你看到执行结果不一致。但是最终结果都是一致的。

萍水相逢
2013/04/16 10:38上午

count:==== 0

count:==== 1

count:==== 3

count:==== 6

count:==== 10

count:==== 15

count:==== 21

count:==== 28

count:==== 36

count:==== 45

count:==== 45

count:==== 46

count:==== 48

count:==== 51

count:==== 55

count:==== 60

count:==== 66

count:==== 73

count:==== 81

count:==== 90

——-

count:==== 0

count:==== 1

count:==== 3

count:==== 3

count:==== 4

count:==== 6

count:==== 9

count:==== 13

count:==== 18

count:==== 24

count:==== 31

count:==== 39

count:==== 48

count:==== 51

count:==== 55

count:==== 60

count:==== 66

count:==== 73

count:==== 81

count:==== 90

——

count:==== 0

count:==== 0

count:==== 1

count:==== 3

count:==== 6

count:==== 10

count:==== 15

count:==== 21

count:==== 28

count:==== 36

count:==== 45

count:==== 46

count:==== 48

count:==== 51

count:==== 55

count:==== 60

count:==== 66

count:==== 73

count:==== 81

count:==== 90

fair_jm
2013/04/16 10:24下午

最后的结果是一样的 这个不是控制线程的调度啊…这个是防止竞态条件导致的与预期不同的错误..

kopcoder
2013/04/16 5:25下午

System.out.println(“count:==== “+count);

这行打印语句换成

System.out.println(Thread.currentThread().getName() + “ count:==== “+count);

就可以看出差别了。

两个线程并不会交替调用add方法，也不会一个线程执行完了才交给另外一个线程执行，所以每次执行的结果基本都不会一样。

为什么呢
2013/09/30 2:38下午

Thread-0 ==== 0

Thread-1 ==== 0

Thread-0 ==== 1

Thread-0 ==== 4

Thread-0 ==== 7

Thread-1 ==== 2

Thread-0 ==== 11

Thread-0 ==== 18

Thread-0 ==== 24

Thread-1 ==== 13

Thread-0 ==== 31

Thread-1 ==== 34

Thread-0 ==== 42

Thread-1 ==== 46

Thread-0 ==== 55

Thread-1 ==== 60

Thread-1 ==== 66

Thread-1 ==== 73

Thread-1 ==== 81

Thread-1 ==== 90

```
class Counter {
long count = 0;
public void add(long value) {
this.count += value;
System.out.println(Thread.currentThread().getName() + ” ==== ”
+ this.count);
}
}
```

add方法不加synchronized执行结果也Ok啊？

怎么回事呢？

方 腾飞
2013/10/05 3:36下午

能贴出完整的代码吗？

为什么呢
2013/10/08 10:44上午

代码就是文中的例子。

不过现在已经弄明白了,问题出在线程的run方法中，

由于循环的时间较短，cpu的处理速度够快的话，多线程的资源竞争

冲突就表现得不那么明显。

所以例子中的add方法synchronized keyword不加，虽然从代码上看确实有问题

但是结果很有可能就是一致的。

※多线程新手成长中

> 方 腾飞
> 2013/10/10 11:02下午
>
> 赞！

niannian
2013/10/11 5:59下午

对，在循环里加上个Thread.sleep(20)..就会看出区别。

niannian
2013/10/11 6:01下午

问下：

静态方法中的同步块里：

“如果第二个同步块不是同步在MyClass.class这个对象上。那么这两个方法可以同时被线程访问。”

不是同步在MyClass.class，那么还会同步在哪个类对象上呢？只有这个类才能调用这个方法不是？

> 李同杰
> 2013/10/11 7:46下午
>
> 您好：
>
> 这里是指同步在实例对象或者其他对象上，如下文中的同步实例对象的例子。

Frank
2014/03/04 8:53上午

您好，因为刚刚接触多线程，有个问题想请教一下。问题就是：为什么实例方法上的同步就是同步到了拥有该方法的对象上？静态方法的同步就是同步在该方法所在的类对象上？同步代码块同步到他们所属的对象或者是类上到是可以理解，因为synchronized关键字后面指明了当前对象以及当前类（this和MyClass.clas）。希望能帮助解答一下，因为不是太懂，谢谢了。

李同杰
2014/03/04 2:19下午

您好：

Java的同步机制是通过给对象(资源)加锁实现的，为了防止同时访问共享资源(如内存)，线程会在访问前后加锁和解锁。

静态（static）方法是属于类对象，在xx.class被JVM load的时候，xx.class就是一个对象，类型是Class，对于静态方法的同步就会同步在类对象上。而实例方法是属于实例对象，通过new关键字产生出来，比如Object a=new Object()，这时候同步就会同步在实例对象上。

钟刘旺
2014/09/11 9:29上午

Thread threadA = new CounterThread(counter); 通不过。

李同杰
2014/09/11 11:12上午

> 钟刘旺 :
>
> Thread threadA = new CounterThread(counter); 通不过。

提示什么错误？

Bingo
2014/09/26 9:56下午

够深，够细。N个赞

Jintao_Ma
2016/11/15 5:40下午

感觉不用做这么多的分类，同步目的就是保证写资源的唯一性；那么当一个资源需要的同步的时候用同步代码块，当一个方法中很多资源需要同步的时候，用同步方法更方便。

我们也不关心这个方法是不是静态的，只关心它有没有操作共享资源。

zhili
2018/05/31 11:22上午

这两个方法不允许同时被线程访问。

有歧义, 应该为:

这两个方法中的任意一个均不允许同时被线程访问。