**Java Synchronized Blocks**

* [The Java synchronized Keyword](http://tutorials.jenkov.com/java-concurrency/synchronized.html#synchronized-keyword)
* [Synchronized Instance Methods](http://tutorials.jenkov.com/java-concurrency/synchronized.html#synchronized-instance-methods)
* [Synchronized Static Methods](http://tutorials.jenkov.com/java-concurrency/synchronized.html#synchronized-static-methods)
* [Synchronized Blocks in Instance Methods](http://tutorials.jenkov.com/java-concurrency/synchronized.html#synchronized-blocks-instance-methods)
* [Synchronized Blocks in Static Methods](http://tutorials.jenkov.com/java-concurrency/synchronized.html#synchronized-blocks-static-methods)
* [Java Synchronized Example](http://tutorials.jenkov.com/java-concurrency/synchronized.html#java-synchronized-example)
* [Java Concurrency Utilities](http://tutorials.jenkov.com/java-concurrency/synchronized.html#java-concurrency-utilities)

Java同步块将方法或代码块标记为同步的。可以使用Java同步块来避免竞争条件([**race conditions**](http://tutorials.jenkov.com/java-concurrency/race-conditions-and-critical-sections.html))。

**The Java synchronized Keyword**

Java中的同步块用关键字synchronized 标记。Java中的同步块在某个对象上同步。在同一对象上同步的所有同步块只能在同一时间内执行一个线程。试图进入同步块的所有其他线程都被阻塞，直到同步块内的线程退出该块为止。

关键字synchronized 可用于标记四种不同类型的块：

1. 实例方法
2. 静态方法
3. 实例方法中的代码块
4. 静态方法中的代码块

这些块在不同的对象上同步。你需要哪种类型的同步块取决于具体情况。

**Synchronized Instance Methods**

这是一个同步实例方法：

public **synchronized** void add(int value){

this.count += value;

}

注意在方法声明中关键字synchronized的使用。这告诉Java该方法是同步的。

Java中的同步实例方法在拥有该方法的实例（对象）上同步。因此，每个实例都有同步的方法在不同的对象上同步：拥有实例。只有一个线程可以在同步实例方法中执行。如果存在不止一个实例，那么每个实例每次一个线程可以在同步实例方法内执行。每个实例一个线程。

**Synchronized Static Methods**

静态方法被标记为同步，就像使用同步关键字的实例方法一样。这里是一个Java同步静态方法示例：

public **static synchronized** void add(int value){

count += value;

}

同样，这里关键字synchronized告诉Java该方法是同步的。

同步静态方法在同步静态方法属于的类对象上同步。由于每个类中只有一个类对象存在于Java VM中，只有一个线程可以在同一个类中的静态同步方法中执行。

如果静态同步方法位于不同的类中，那么一个线程可以在每个类的静态同步方法中执行。无论它调用哪个静态同步方法，每一个类都只有一个线程。

**Synchronized Blocks in Instance Methods**

您不必同步整个方法。有时最好只对方法的一部分进行同步。方法中的Java同步块使这成为可能。

这里是一个同步的Java代码块，在一个非同步的Java方法中：

public void add(int value){

**synchronized(this){**

this.count += value;

**}**

}

此示例使用Java同步块构造来将代码块标记为同步的。这个代码现在将执行，就像它是一个同步方法一样。

注意Java同步块构造在括号中使用一个对象。在示例中使用“this”，这是add方法被调用的实例。由同步构造在括号中所取的对象称为监视器对象。该代码被称为在监视器对象上同步。同步实例方法使用它所属的对象作为监视器对象。

只有一个线程可以在同一监视器对象上同步的Java代码块内执行。

下面两个示例在调用它们的实例上都同步。因此，它们相对于同步是等价的：

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);

**}**

}

}

因此，在这个示例中只有一个线程可以在两个同步块中的任一个内执行。

如果第二个同步块是在与this不同的对象上同步的，那么一个线程一次就能够在每个方法内执行。

**Synchronized Blocks in Static Methods**

这里有两个静态方法的例子。这些方法在方法所属类的类对象上同步：

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);

}

}

}

只有一个线程可以在这两个方法中同时执行。

如果在与MyClass.class不同的对象上对第二个同步块进行同步，那么一个线程可以同时在每个方法中执行。

**Java Synchronized Example**

下面是一个启动2个线程并使它们都在Counter的相同实例上调用add方法的示例。在同一个实例上每次只有一个线程能够调用add方法，因为该方法在它所属的实例上同步。

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;

}

public void run() {

for(int i=0; i<10; i++){

counter.add(i);

}

}

}

public class Example {

public static void main(String[] args){

Counter counter = new Counter();

Thread threadA = new CounterThread(counter);

Thread threadB = new CounterThread(counter);

threadA.start();

threadB.start();

}

}

两个线程被创建。在它们的构造函数中，将相同的Counter 实例传递给它们。Counter.add()方法在实例上是同步的，因为add方法是实例方法，并且标记为synchronized。因此，一次只有一个线程调用add()方法。另一个线程将等待，直到第一个线程离开add()方法，才能自己执行方法。

如果两个线程引用了两个单独的Counter实例，那么同时调用add()方法就没有问题。调用可能已经指向不同的对象，所以调用的方法也会在不同的对象(拥有该方法的对象)上同步。因此，调用不会阻塞。这就是它的样子：

public class Example {

public static void main(String[] args){

Counter counterA = new Counter();

Counter counterB = new Counter();

Thread threadA = new CounterThread(counterA);

Thread threadB = new CounterThread(counterB);

threadA.start();

threadB.start();

}

}

注意两个线程threadA和threadA，不再引用相同的Counter实例。counterA和counterB的add方法在它们的两个自己的实例上是同步的。因此，在counterA上调用add()并不会阻塞counterB上对add()的调用。

**Java Concurrency Utilities**

synchronized 机制是Java的第一个机制，用于同步对多个线程共享的对象的访问。不过synchronized 机制不是很先进。这就是为什么Java 5得到了整整一组并发的实用工具类([**concurrency utility classes**](http://tutorials.jenkov.com/java-util-concurrent/index.html))，以帮助开发人员实现比使用synchronized所获得的更细粒度的并发控制。