**Compare and Swap**

* [What Situations Compare And Swap is Intended to Support](http://tutorials.jenkov.com/java-concurrency/compare-and-swap.html#what-situations-compare-and-swap-is-intended-to-support)
* [Compare And Swap As Atomic Operation](http://tutorials.jenkov.com/java-concurrency/compare-and-swap.html#compare-and-swap-as-atomic-operation)

**比较和交换**(**Compare and swap**)是一种设计并发算法时使用技术。基本，比较和交换比较一个期望值与一个变量的具体值，如果变量的具体值等于期望值，交换变量值为一个新的值。比较和交换可能听起来有一点复杂，但是一旦你理解了它，它就相当简单了，所以让我进一步阐述一下这个主题

**What Situations Compare And Swap is Intended to Support**

一个在程序和并发算法中非常常发生的模式是“检查和执行”模式。检查然后执行模式发生在代码首次检查变量值然后基于那个值执行。这是一个简单例子：

class MyLock {

private boolean locked = false;

public boolean lock() {

if(!locked) {

locked = true;

return true;

}

return false;

}

}

这段代码有许多错误如果它将被用在一个多线程应用中，但请暂时忽略那些。

正如你所见，lock()方法首先检查是否locked成员变量等于false(检查)，然后如果是的话，它设置locked为true(然后执行)。

如果多个线程访问相同的MyLock实例，上面的lock()方法不将被保证可工作。如果一个线程A检查locked的值，看到它是false，线程B也在同时检查locked值。或者，事实上，在线程A设置locked为false之前的任一时间。因此，两个线程A和B都可能看到locked为false，两个都将基于那个信息执行。

为了在一个多线程应用中适当地工作，“检查和执行”操作必须是院子的。原子的意味着“检查”和“执行”两个行为都被作为原子(非可划分的)执行代码块。任一开始执行块的线程将会最终执行块而不互相交叉。没有其他的线程可以同时执行原子块。

这是代码例子，来自早前的lock()方法使用synchronized关键字转变为一个原子代码块：

class MyLock {

private boolean locked = false;

public **synchronized** boolean lock() {

if(!locked) {

locked = true;

return true;

}

return false;

}

}

现在lock()方法是同步的，以便一个时刻在同一个MyLock实例上只有一个线程能够执行它。lock()方法效力上是原子的。

原子的lock()方法实际是一个“比较和交换”的例子。lock()方法*比较*变量locked与预期的值false，如果locked等于这个预期值，它*交换*变量值为true。

**Compare And Swap As Atomic Operation**

现代CPU拥有内置的对原子比较和交换操作的支持。从Java 5开始你可以获得访问这个CPU中的函数借由java.util.concurrent.atomic包中一些新的原子类。

这是一个例子展示如何使用AtomicBoolean类实现之前展示的lock()方法：

public static class MyLock {

private AtomicBoolean locked = new AtomicBoolean(false);

public boolean lock() {

return locked.compareAndSet(false, true);

}

}

注意locked变量不再是一个boolean但是一个AtomicBoolean。这个类有一个compareAndSet()函数，该函数比较AtomicBoolean实例的值与一个预期值，如果有预期值，它用一个新值交换这个值。这种情况中它比较locked值与false，如果它是false，它设置AtomicBoolean新值为true。如果值被交换compareAndSet()方法返回true，如果没有则返回false。

使用Java 5 +而不是实现自己的比较和交换特性的优点在于，内置到Java 5 +中的比较和交换特性允许您利用应用程序正在运行的CPU的底层比较和交换特性。这使你的比较和交换代码更快。