**Thread Safety and Immutability**

* [The Reference is not Thread Safe!](http://tutorials.jenkov.com/java-concurrency/thread-safety-and-immutability.html#reference)

只有当多个线程正在访问相同的资源，**并且**一个或多个线程**写入**资源时，才会出现竞争条件([**Race conditions**](http://tutorials.jenkov.com/java-concurrency/race-conditions-and-critical-sections.html))。如果多个线程读取相同的资源，则不会发生竞争条件([**Race conditions**](http://tutorials.jenkov.com/java-concurrency/race-conditions-and-critical-sections.html))。

通过使共享对象不可变，从而确保线程安全，可以确保线程之间共享的对象永远不会被任何线程更新。下面是一个例子：

public class ImmutableValue{

private int value = 0;

public ImmutableValue(int value){

this.value = value;

}

public int getValue(){

return this.value;

}

}

注意在构造函数中传递了ImmutableValue 实例的值。还要注意没有setter方法。一旦创建了一个ImmutableValue 实例，就不能更改它的值。它是不可变的。但是，可以使用getValue()方法读取它。

如果需要对ImmutableValue实例执行操作，可以通过返回具有由该操作生成的值的新实例来执行。下面是一个添加操作的例子：

public class ImmutableValue{

private int value = 0;

public ImmutableValue(int value){

this.value = value;

}

public int getValue(){

return this.value;

}

**public ImmutableValue add(int valueToAdd){**

**return new ImmutableValue(this.value + valueToAdd);**

**}**

}

注意，add()方法返回带有添加操作结果的新ImmutableValue实例，而不是将值添加到自身。

**The Reference is not Thread Safe!**

重要的是要记住，即使对象是不可变的，因此线程安全的，对该对象的引用可能不是线程安全的。看看这个例子：

public class Calculator{

private ImmutableValue currentValue = null;

public ImmutableValue getValue(){

return currentValue;

}

public void setValue(ImmutableValue newValue){

this.currentValue = newValue;

}

public void add(int newValue){

this.currentValue = this.currentValue.add(newValue);

}

}

Calculator 类保存对ImmutableValue 实例的引用。请注意，如何通过setValue()和add()方法更改引用。因此，即使Calculator类在内部使用不可变对象，它本身也不是不可变的，因此也不是线程安全的。换句话说，ImmutableValue 类是线程安全的，但是**使用它**是不安全的。当试图通过不变特性实现线程安全时，这是要牢记的。

为了使Calculator 类线程安全，您可以声明 getValue()、setValue()和 add()方法synchronized。那就行了。