线程安全及不可变性

原文链接 作者：Jakob Jenkov 译者：高嵩 校对：丁一

当多个线程同时访问同一个资源，并且其中的一个或者多个线程对这个资源进行了写操作，才会产生**竞态条件**。多个线程同时读同一个资源不会产生竞态条件。

我们可以通过创建不可变的共享对象来保证对象在线程间共享时不会被修改，从而实现线程安全。如下示例：

```
01  public class ImmutableValue{
02      private int value = 0;
03
04      public ImmutableValue(int value){
05          this.value = value;
06      }
07
08      public int getValue(){
09          return this.value;
10      }
11  }
```

请注意ImmutableValue类的成员变量value是通过构造函数赋值的，并且在类中没有set方法。这意味着一旦ImmutableValue实例被创建，value变量就不能再被修改，这就是不可变性。但你可以通过getValue()方法读取这个变量的值。

（译者注：注意，“不变”（Immutable）和“只读”（Read Only）是不同的。当一个变量是“只读”时，变量的值不能直接改变，但是可以在其它变量发生改变的时候发生改变。比如，一个人的出生年月日是“不变”属性，而一个人的年龄便是“只读”属性，但是不是“不变”属性。随着时间的变化，一个人的年龄会随之发生变化，而一个人的出生年月日则不会变化。这就是“不变”和“只读”的区别。（摘自《Java与模式》第34章））

如果你需要对ImmutableValue类的实例进行操作，可以通过得到value变量后创建一个新的实例来实现，下面是一个对value变量进行加法操作的示例：

```
01  public class ImmutableValue{
02      private int value = 0;
03
04      public ImmutableValue(int value){
05          this.value = value;
06      }
07
08      public int getValue(){
09          return this.value;
10      }
11
12      public ImmutableValue add(int valueToAdd){
13          return new ImmutableValue(this.value + valueToAdd);
14      }
```

```
15  }
```

请注意add()方法以加法操作的结果作为一个新的ImmutableValue类实例返回，而不是直接对它自己的value变量进行操作。

引用不是线程安全的！

重要的是要记住，即使一个对象是线程安全的不可变对象，指向这个对象的引用也可能不是线程安全的。看这个例子：

```
01  public void Calculator{
02      private ImmutableValue currentValue = null;
03
04      public ImmutableValue getValue(){
05          return currentValue;
06      }
07
08      public void setValue(ImmutableValue newValue){
09          this.currentValue = newValue;
10      }
11
12      public void add(int newValue){
13          this.currentValue = this.currentValue.add(newValue);
14      }
15  }
```

Calculator类持有一个指向ImmutableValue实例的引用。注意，通过setValue()方法和add()方法可能会改变这个引用。因此，即使Calculator类内部使用了一个不可变对象，但Calculator类本身还是可变的，因此Calculator类不是线程安全的。换句话说：ImmutableValue类是线程安全的，但使用它的类不是。当尝试通过不可变性去获得线程安全时，这点是需要牢记的。

要使Calculator类实现线程安全，将getValue()、setValue()和add()方法都声明为同步方法即可。