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：

```
public class ImmutableValue{
        private int value = 0;

        public ImmutableValue(int value){
                this.value = value;
        }

        public int getValue(){
                return this.value;
        }
}
```

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：

```
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);
        }
}
```

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：

```
public void 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。

夕水溪下
2013/03/04 5:56下午

总结：同时读不会引起线程安全问题。

m1llerma
2013/05/27 7:20下午

最后的currentValue字段有volatile语义就可以了

劳希
2013/07/06 2:34下午

> m1llerma :
>
> 最后的currentValue字段有volatile语义就可以了

setValue()方法确实currentValue字段有volatile语义就可以了,但是add方法中用到了volatile的当前值,所以不能保证线程安全

> 夕水溪下

2013/12/18 4:20下午

java会保证对引用的读写是原子的，但add方法中有对currentValue的再次引用，也就相当于i++之类的语法，所以还需要做同步，只有volatile是不行的。

淡蓝的晴
2014/06/14 9:49上午

你好，我还有很多没看懂的地方。我想问些问题如果有错，请帮纠正
最近看java并发实战的时候看到了类似的问题，也是先创建一个不可变的类，然后在需要被多线程访问的类里面创建 这个不可变类的对象并用volatile修饰。书中说这样做是线程安全的，因为之前这本书还介绍了volatile的线程安全特性主要是在可见性上。我是不是可以理解为这个可见性，主要是针对构造上的可见。所以在setValue和add方法上volatile的变量没有意义？

每天打老虎
2014/06/17 5:48下午

voltile 可见性指的是，一个线程修改了这个变量的值，会立马写回主存储器中，这样就能被另一个线程看到这个变量的值的变化（多线程是并发并不是并行）。 Doug Lea并发编程实战中写过 voltile 只能保证可见性，并不能保证线程安全。线程安全还是要靠监视器保证。volitile 可见性是确保编译器不会对代码做寄存器优化（关于什么乱序和栅栏方面的东西，喜欢做研究的可以到处找找）。 具体的代码我没看，我只是路过。

dongzh
2014/10/07 4:53下午

请问，同步能保证可见性么？为啥并发实战里讲volatile可以提供弱同步，同步跟可见到底啥关系呢？

方 腾飞
2014/06/21 1:45下午

可见性是指一个线程修改了变量A，对于另外一个线程是绝对可见的。

zhengJackson
2013/11/04 11:31下午

深入浅出！

silence
2014/01/23 7:49下午

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。

—请问：仅仅将setValue()和add()方法声明为同步方法不可以吗？为什么还需要getValue()也声明为同步的。这里的getValue()方法我指的是ImmutableValue类中的getValue()方法

> silence
> 2014/01/23 7:52下午
>
> 嘿嘿，不好意思，之前没看到Calculator类也有getValue方法，sorry，我还以为指的是ImmutableValue类中的getValue()方法

AnyStretch
2014/08/14 6:08下午

有点疑惑，在ImmutableValue中如果value不是final的话，那么再构造过程中是不是会发送重排序呢？此时线程安全又如何保证呢？

> dongzh
> 2014/10/07 3:31下午
>
> 并发实战里有讲建议所有域都应该是final，但不是必须，并有注解说要对类的良性数据竞争情况精确分析，不太明白，期待高手解答。

muyu
2015/02/09 9:48下午

文章写得很深入，但是Calculator类的代码有问题吧，开头应该把void换成class吧。。。

Darker
2018/11/13 10:10下午

有个不太明白的地方，望解答。

Calculator这个

Darker

2018/11/13 10:15下午

有个不太明白的地方，望解答。

Calculator这个类为什么要给出一个get方法，而且还会出现空指针异常。