Java volatile特性介绍

## 一句话概括：

关键字volatile的主要作用是使变量在多个线程间可见。

## 补充介绍：

了解过JVM工作原理的都应该知道，在多线程环境中，每个私有线程会拷贝公共堆栈上需要用到的变量数据，然后读取变量是从私有数据栈中读取的，这样可能造成读取的数据不是实时的，出现脏读情况。而**用volatile修饰的公共变量则每次都强制从公共堆栈中取得变量的值，而不是从私有数据栈中读取**。这样至少能保证每次读取到的共享数据是实时的。

关键字volatile主要使用场合是在多个线程中可以感知实例变量被更改了，并且可以获得最新的调用，也就是用多个线程读取共享变量时可以获得最新值的使用。

## 不支持原子性：

volatile关键字增加了实例变量在各个线程之间的可见性，但它最致命的缺点就是不支持原子性。

关键字volatile提示线程每次从共享内存中读取变量而不是从私有内存中读取，这样就保证了同步数据的可见性。但是这里需要注意的是：如果修改实例变量中的数据，比如i++,也就是i=i+1, 则这样的操作其实并不是一个原子操作，也就是非线程安全的。表达式i++的操作步骤分解如下：

1）从内存中取出i的值

2）计算i的值

3）将i的值写到内存中。

假如在第2步计算值的时候，另外一个线程也修改了i的值，那么这个时候就会出现脏读数据。所以说volatile本身并不处理数据的原子性，而是强制对数据的读写及时影响到主内存。

从Java加载变量的8个阶段来看volatile出现非线程安全的原因：

1）read和load阶段：从主内存复制变量到当前线程工作内存；

2）use和assign阶段：执行代码，改变共享变量值；

3）store和write阶段：用工作内存数据刷新主内存对应的变量的值。

对于volatile修饰的变量，JVM只是保证从主内存加载到线程工作内存的值是最新的，例如线程1和线程2在进行read和load的操作中，都从主内存加载最新的count变量，待他们read和load之后，如果此主内存的count变量再次发生变化就不会影响到线程1和2了，因为他们已经加载完毕了，这个时候就造成了私有内存和公共内存中的数据不一致，那么相应的最后计算的结果肯定也不一致，也就是出现了线程安全问题。

所以说，volatile关键字解决的是变量读时的可见性问题，但无法保证原子性。如何保证数据原子性呢，还是需要加入synchronized关键字的帮助。因此我们在设计代码的时候，如果有需要考虑原子性的问题，还是尽量用synchronized关键字去做同步，如果不需要考虑原子性，那么volatile修饰变量时一种简单高效的办法。

## volatile和synchronized关键字比较：

1）关键字volatile是线程同步的轻量级实现，所以性能上肯定比synchronized要好。不过随着JDK的更新，synchronized关键字的效率得到了很大的提升，所以一般性的用synchronized关键字还是比较多的，性能也非常好。

2）volatile只能修饰变量，而synchronized关键字可以修饰方法，代码块等，功能更全，当然操作也更麻烦。

3）多线程访问volatile不会发生阻塞，而synchronized会阻塞。

4）volatile能保证数据的可见性，但不能保证原子性。而synchronized关键字既可以保证可见性又能保证原子性，因为它会将私有内存和公共内存中的数据做同步。

5）关键字synchronized可以保证在同一时刻，只有一个线程可以执行某一个方法或者某一段代码块。它包含两个特性：互斥性和可见性，同步synchronized不仅可以解决一个线程看到对象处于不一致的状态，它还可以保证进入同步方法或者同步代码块的每个线程，都看到由同一个锁保护之前的所有最新修改效果。