Java 多线程开发之 volatile（一）

一、前言

Java 提供了一种稍弱的同步机制，即 volatile 变量，用来确保将变量的更新操作通知到其他线程。

我们可以将 volatile 看做一个轻量级的锁，但是又与锁有些不同：

> 对于多线程，不是一种互斥关系
>
> 不能保证变量状态的“原子性操作”

二、基本概念

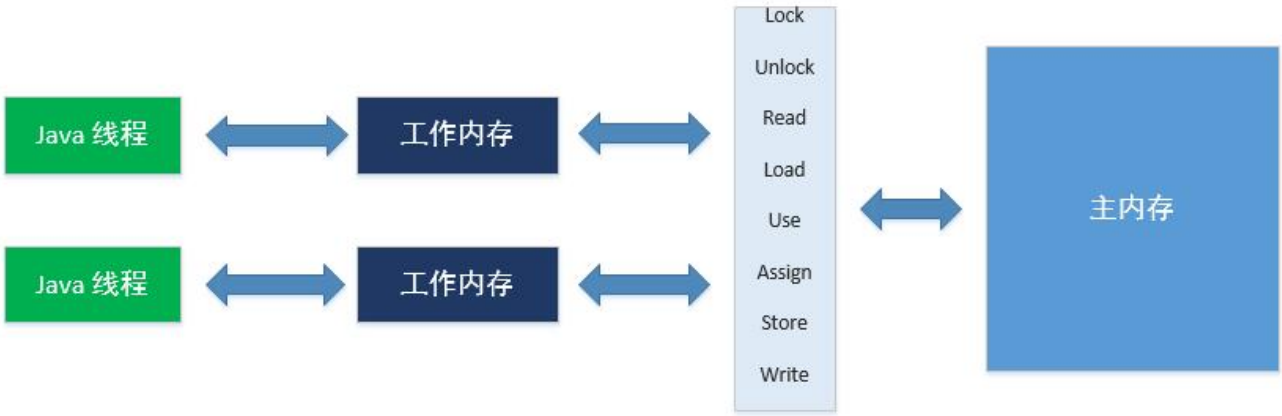
2.1 Java 内存模型

Java 内存模型规定了所有的变量（此处的变量是指成员变量和静态变量）都存储在主内存中，每条线程都有自己的工作内存。

工作内存中保存了被该线程使用到的变量的主内存副本拷贝，线程对变量的所有操作都必须在工作内存中进行，而不能直接读写主内存的变量。

不同的线程也无法直接访问对方的工作内存中的变量，它们的变量值的传递均需要通过主内存来完成。

主内存和工作内存关系如下图：



其中，主内存和工作内存通过图中 8 个操作对变量进行同步。

> lock（锁定）：　作用于主内存的变量，它把一个变量标识为一条线程独占的状态
> unlock（解锁）：作用于主内存的变量，它把一个处于锁定状态的变量释放出来，释放后的变量才可以被其他线程锁定。
> read（读取）：　作用于主内存的变量，它把一个变量的值从主内存传输到线程的工作内存中以便随后的 load 动作使用。
> load（载入）：　作用于工作内存的变量，它把 read 操作从主内存中得到的变量放入工作内存的变量副本中。
> use（使用）：　作用于工作内存的变量，它把工作内存的变量的值传递给执行引擎，每当虚拟机遇到一个需要使用到变量的值的字节码指令时将会执行这个操作。
> assign（赋值）：作用于工作内存的变量，它把从执行引擎接收到的值赋给工作内存的变量，每当虚拟机遇到一个给变量赋值的字节码指令时执行这个操作。
> store（存储）：　作用于工作内存的变量，它把工作内存中的变量的值传送到主内存中，以便随后的 write 操作使用。
> write（写入）：　作用于主内存的变量，它把 store 操作从工作内存中得到的变量的值放入主内存的变量中。

2.2 可见性

可见性是指当一个线程修改了共享变量的值，其他线程能够立即得知这个修改。

Java 内存模型是通过在变量修改后将新值同步会主内存，在变量读取前从主内存刷新变量这种依赖主内存作为传递媒介的方式实现可见性的。

volatile 有个特殊规则：保证新值能立即同步到主内存，以及每次使用变量前立即从主内存刷新。

三、案例演示与分析

3.1 案例

```
public class VolatileTest {

    public static void main(String[] args) {

        DemoThread dt = new DemoThread();
        dt.start();

        while(true) {
            if (dt.isFlag()) {
                System.out.println("主线程终止");
                break;
            }
        }

    }
}

class DemoThread extends Thread {

    private boolean flag;

    @Override
    public void run() {
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        flag = true;
        System.out.println("子线程-flag:" + isFlag());
    }

    public boolean isFlag() {
        return flag;
    }

    public void setFlag(boolean flag) {
        this.flag = flag;
    }
```

```
}
```

结果打印：

```
子线程-flag:true
```

从结果可以看出 main 线程没有中断，而是一直循环执行。

3.2 问题分析

从代码中我们可以简单的看出程序执行时产生 2 个线程（执行 main 方法的线程 和 新创建的 DemoThread 线程）。

这 2 个线程先从主内存读取 flag （值为 false），拷贝到各自的工作内存中。

当 DemoThread 线程执行 run 方法时，修改了自己工作内存的 flag 变量，并将修改的值同步回主内存中。而 main 线程执行 while 循环时，由于 while(true) 执行速度太快，导致 main 线程没有时间刷新主内存中修改的变量值，因此只能读取自己工作内存 flag 的变量，从而导致 while 循环一直进行下去。

四、解决方案

正如上文提到的概念，给声明的变量添加 volatile 关键字，保证内存可见性。

```
private volatile boolean flag;
```

再执行代码程序时，结果如下：

```
主线程终止
子线程-flag:true
```

当 DemoThread 线程修改 flag 值后立即同步回主内存，main 线程立即刷新主内存的 flag 值到工作内存中，读取 flag = true，终止 while 循环。