## volatile 与 Java内存模型

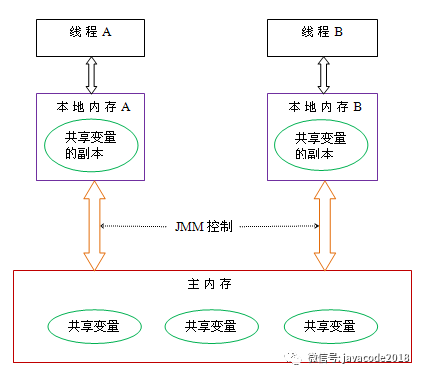


运行上面代码，会发现程序无法终止。

线程t1的run()方法中有个循环，通过flag来控制循环是否结束，主线程中休眠了1秒，将flag置为false，按说此时线程t1会检测到flag为false，打印“线程t1停止了”，为何和我们期望的结果不一样呢？运行上面的代码我们可以判断，t1中看到的flag一直为true，主线程将flag置为false之后，t1线程中并没有看到，所以一直死循环。

那么t1中为什么看不到被主线程修改之后的flag？

要解释这个，我们需要先了解一下java内存模型（JMM），Java线程之间的通信由Java内存模型（本文简称为JMM）控制，**JMM决定一个线程对共享变量的写入何时对另一个线程可见**。从抽象的角度来看，JMM定义了线程和主内存之间的抽象关系：线程之间的共享变量存储在主内存（main memory）中，每个线程都有一个私有的本地内存（local memory），本地内存中存储了该线程以读/写共享变量的副本。本地内存是JMM的一个抽象概念，并不真实存在。它涵盖了缓存，写缓冲区，寄存器以及其他的硬件和编译器优化。Java内存模型的抽象示意图如下：

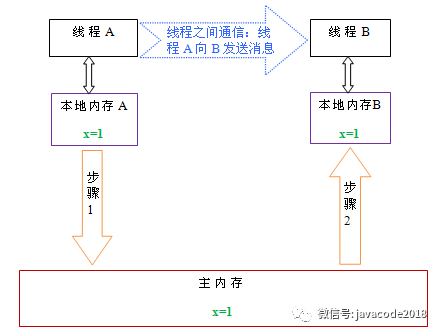


从上图中可以看出，线程A需要和线程B通信，必须要经历下面2个步骤：

首先，线程A把本地内存A中更新过的共享变量刷新到主内存中去

然后，线程B到主内存中去读取线程A之前已更新过的共享变量

下面通过示意图来说明这两个步骤：



如上图所示，本地内存A和B有主内存中共享变量x的副本。假设初始时，这三个内存中的x值都为0。线程A在执行时，把更新后的x值（假设值为1）临时存放在自己的本地内存A中。当线程A和线程B需要通信时，线程A首先会把自己本地内存中修改后的x值刷新到主内存中，此时主内存中的x值变为了1。随后，线程B到主内存中去读取线程A更新后的x值，此时线程B的本地内存的x值也变为了1。从整体来看，这两个步骤实质上是线程A在向线程B发送消息，而且这个通信过程必须要经过主内存。**JMM通过控制主内存与每个线程的本地内存之间的交互，来为java程序员提供内存可见性保证。**

对JMM了解之后，我们再看看文章开头的问题，线程t1中为何看不到被主线程修改为false的flag的值，有两种可能:

1. 主线程修改了flag之后，未将其刷新到主内存，所以t1看不到
2. 主线程将flag刷新到了主内存，但是t1一直读取的是自己工作内存中flag的值，没有去主内存中获取flag最新的值

对于上面2种情况，有没有什么办法可以解决？

是否有这样的方法：线程中修改了工作内存中的副本之后，立即将其刷新到主内存；工作内存中每次读取共享变量时，都去主内存中重新读取，然后拷贝到工作内存。

java帮我们提供了这样的方法，使用volatile修饰共享变量，就可以达到上面的效果，**被volatile修改的变量有以下特点：**

**1、线程中读取的时候，每次读取都会去主内存中读取共享变量最新的值，然后将其复制到工作内存**

**2、线程中修改了工作内存中变量的副本，修改之后会立即刷新到主内存**

我们修改一下开头的示例代码：



使用volatile修饰flag变量，然后运行一下程序，输出：



这下程序可以正常停止了。

volatile解决了共享变量在多线程中可见性的问题，可见性是指一个线程对共享变量的修改，对于另一个线程来说是否是可以看到的。