

volatile可见性详解

volatile是java虚拟机提供的轻量级的同步机制具有以下特点：

- 1.1.保证可见性
- 2.2.不保证原子性
- 1.3.禁止指令重排

接下来我们先看volatile的可见性的特征，以及底层原理的讲解，那么在研究volatile底层原理之前，我们接下来先要研究一个新的知识-JMM，好，这一题还没讲完呢，猴哥你特么又扯出来一个新的知识点JMM呢？这个时候请按照老师的要求来进行学习，听好去年甚至前年，17 18年面试题上一道必考的题目JVM，这个是java虚拟机，只要是java程序员，地球人都知道，但是最近这一两年，由于这道JVM的题目，比如说一言不合就让你画一个JVM的内存图，大家都背过学过，那么用人单位他没有办法挑出最好的java工程师，所以说题目升级，尤其现在非常注重，你在上一家当前家公司，做的是高并发项目还是单机版的系统，那么如果是高并发的系统，自然而然问题频出，为了解决这些问题，你不得不去研究底层，一研究底层，自然而然会来一个东西叫JMM（java内存模型）不是java虚拟机，现在大厂基本上都会跟你聊聊JMM，如果你说不知道，那么没办法你一定没有干过高并发，一定没有做过JUC编程，你也就是传说中的增删改查程序员，你的这个业务也就这么回事，不是找不到工作，而是很难突破18,20甚至25，那么现在我们要唠唠什么叫java内存模型，它跟我们的volatile有什么关系？首先这道题是谈谈你对volatile的理解。

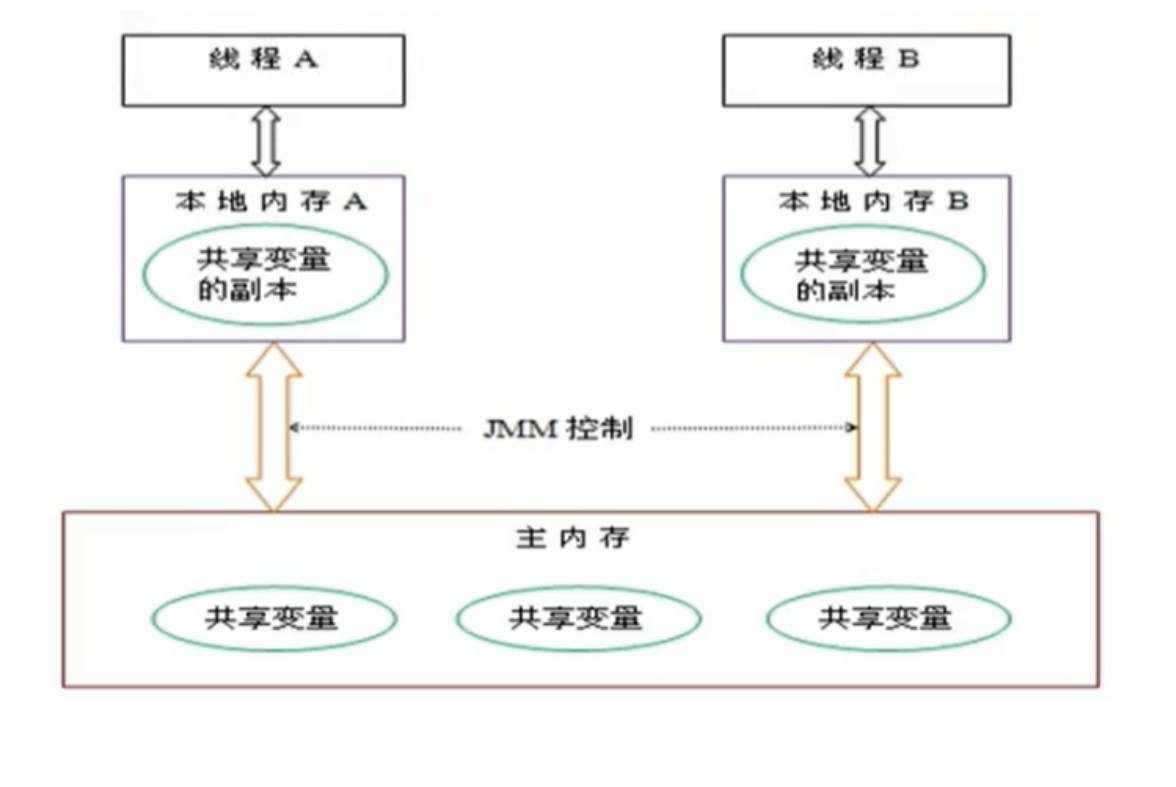
我们首先对于JMM，我们有一个深度的解析，看一下下面横线之间的文字，简单通读一遍：

JMM（java内存模型Java Memory Model，简称JMM）本身是一个抽象的概念**并不真实存在**，他描述的是一组规则或规范，通过这组规范定义了程序中各个变量（包括实例字段，静态字段和构成数组对象的元素）的访问方式

JMM关于同步的规定：

- 1、线程解锁前，必须吧共享变量的值刷新回**主内存**
- 2、线程加锁前，必须读取主内存的最新值到**自己的工作内存**
- 3、加锁解锁是同一把锁

由于JVM运行程序的实体是线程，而每一个线程创建时JVM都会为其创建一个工作内存（有些地方称为栈空间），工作内存是每个线程的私有数据区域，而java内存模型中规定所有的变量都存储在主内存，主内存是共享内存区域，所有的线程都可以访问，**但线程对变量的操作（读取赋值等）必须在工作内存中进行，首先要将变量从主内存拷贝到自己的工作内存空间，然后对变量进行操作，操作完成后再将变量写回主内存**，不能直接操作主内存中的变量，各个线程中的工作内存中存储着主内存的**变量副本拷贝**，因此不同的线程间无法访问对方的工作内存，线程间的通信（传值）必须通过主内存完成，其简要访问过程如下图：

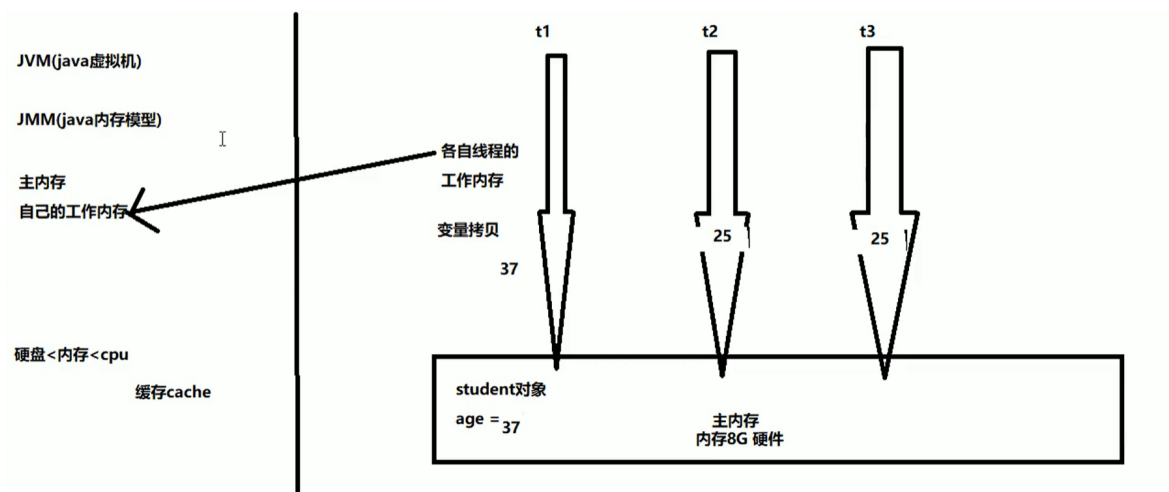


好大家先把这一段话囫圇吞枣的通读一遍，OK相信大家读完之后，不是一脸懵逼就是满脸懵逼，反正吧都是中文，也理解什么不全，也吃透不了全部的意思，但是你要说一点不懂吧，感觉也能摸到一点边，那么这个时候，看大家满脸懵逼的状态，别读了，第一遍，老师的要求第一遍，学三遍，干吗呢？这一边读完了，那么我就告诉你假设这就是一本书，我会向大家证明为什么你现在必须跟着猴哥学，为什么自己自学、看看书成为高手，那是不可能的，没有老师去带你，你不开挂，那么基本上你的成长有限，OK，咱们开始：

经过上面文字的阅读，出现了几个稍微有点晦涩的名字：主内存、自己的工作内存

首先读完的同学不管懂了多少，别读了，咱们开工，我们先解决JMM模型，好，那么首先我们大家都明白，对于我们工作当中数据的传递和存储，它的速率基本上是这样，那就是 硬盘<内存，不用多讲，我们的MySQL数据库装在硬盘上，但是数据的存储要大规模的存取之后，我们都明白磁盘有一个硬件绕不开叫IO，是纯硬件的，它有它的上线和限制，达到了一定的阈值以后，很难在突破，除非是硬件上突破，好，但是我们的数据不会因为你存储在硬盘上她就饶了你，尤其是现在大数据时代，要求数据的读写存储都得是很快，最后我们明白了一个道理，部分数据就不要存储在硬盘上了，放哪？内存里面，看都是一个硬件对应一个软件，那这个内存是什么鬼？那么大家之前咱们学过分布式的内存数据库Redis，大家都明白内存的读取速度是不是快过硬盘？好，那么接下来比内存更快的呢？？？这时候就是CPU，这时候有的同学说：昂，那我懂了，将数据存在CPU中 错大错特错，CPU是负责计算的不管存储，这个时候干吗呢？有这么一个情况，假设CPU比内存快，那么在这中间，这段时间读取速度，CPU的计算能力超强，算完了但是数据还运不上来，那怎么办呢？CPU就这么一直耗着？那不行。这个是我们一定要明白，在CPU和内存读取这存储之间还有一点空间，那么这个空间也就是所谓的缓存。

如果CPU的速度大于内存，这是必须的，如果CPU一直闲着，是不是会导致CPU性能下降？干脆一部分数据计算好了，我们先把它保存在缓存中去，可以抽象的理解为：首先我们的数据在内存里面，每位同学现在在你们的内存插槽里面也都一条内存条，比如你的内存是8G，这就是我们插在主板上的一个硬件，注意：这个硬件就俗称我们的**主内存**。我们每new一个对象就是般存在这个主内存中。



比如说我们这个主内存中有一个Student对象age=25 我们这个在物理内存中这个对象仅此一份，接下来，到了高并发的时代，多个线程，就像我们买票一样，假设我们都要改他的年龄，可不可以多线程来改这个年龄？完全可以的。那么大家看，由于CPU太快，那么在缓存这里，有三个线程；都要操作age,是不是直接操作这个对象呢？？肯定不是。首先每一个线程都会将主内存中的这一份25 拷贝到自己线程一份这个东西就是各自线程的工作内存，就是说线程要去改，不是直接去修改主物理内存中的这个数据，而是线程一种变量的拷贝，将这个变量25 各自拷贝回各自的工作中去，也就是三个线程每个线程中都有一份25，然后第一个线程t1将25 修改为27，改完了，但是线程之间没有办法横向 t2和t3并不知道t1将25改成了37，接下来，在t1线程中将修改后的37写回给主内存，但是不好意思，这个时候t2和t3并不知道主物理内存的值已经从25修改成了37，我们必须要有有一种机制，只要有一个线程修改完自己的工作空间的值并写回到主内存中，要及时通知其他线程，这样及时通知的情况就俗称JMM内存模型中的第一个重要特性俗称**可见性**，能理解？？也就是说某一个线程你修改了值并写回了主内存以后，另外的线程要马上能够知道，我手上的这个25 已经不是最新值了，、作废，需要重新回到主内存去拿到最新的值，能理解？？这时候请大家跟着老师熟悉什么叫可见性？即一个线程修改了主物理内存的值，主物理内存的值被修改，其他线程马上获得通知，假设今天晚上我们需要加课，只要侯哥跟班主任一说，班主任是不是马上在微信群里面@所有人大家是不是马上立刻就能看到知道今晚要加课？否则今晚你就可能请假走人了。那么也就是说只要有变动大家立马可见，立刻收到最新消息这个机制叫**可见性**。

OK此时再次阅读一遍，横线之间的内容，看看什么效果？

可见性的代码验证说明：

```
package com.lagou.test;
import java.util.concurrent.TimeUnit;
class MyData{
    volatile int number = 0;
    public void addTo60(){
        this.number = 60;
    }
}
/**
 * 验证volatile的可见性
 * 1.1 假如int number = 0 ; number变量之前根本没有添加volatile关键字修饰
 */
public class VolatileDemo {
```

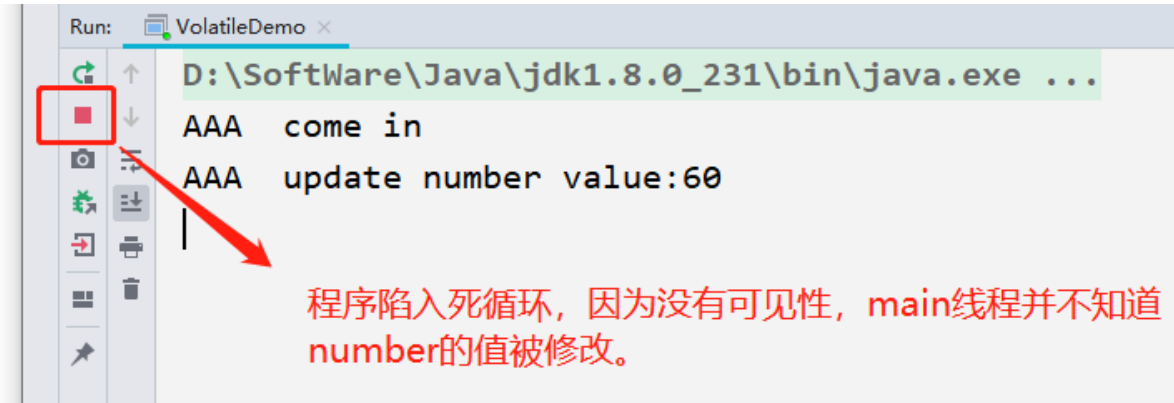
```

public static void main(String[] args) {//main是一切方法的运行入口
    MyData myData = new MyData();//资源类
    new Thread(() ->{
        System.out.println(
            Thread.currentThread().getName()+"\t come in");
        //暂停一会儿线程
        try {
            TimeUnit.SECONDS.sleep(3);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        myData.addTo60();
        System.out.println(
            Thread.currentThread().getName()+
            "\t update number value:"+ myData.number);
    }, "AAA").start();
    //第二个线程就是我们的main线程
    while (myData.number == 0){
        //main线程就一直在这里等待，直到number值不再等于0
    }
    System.out.println(Thread.currentThread().getName()+
        "\t mission is over, main get number
value:"+myData.number);
}
}

```

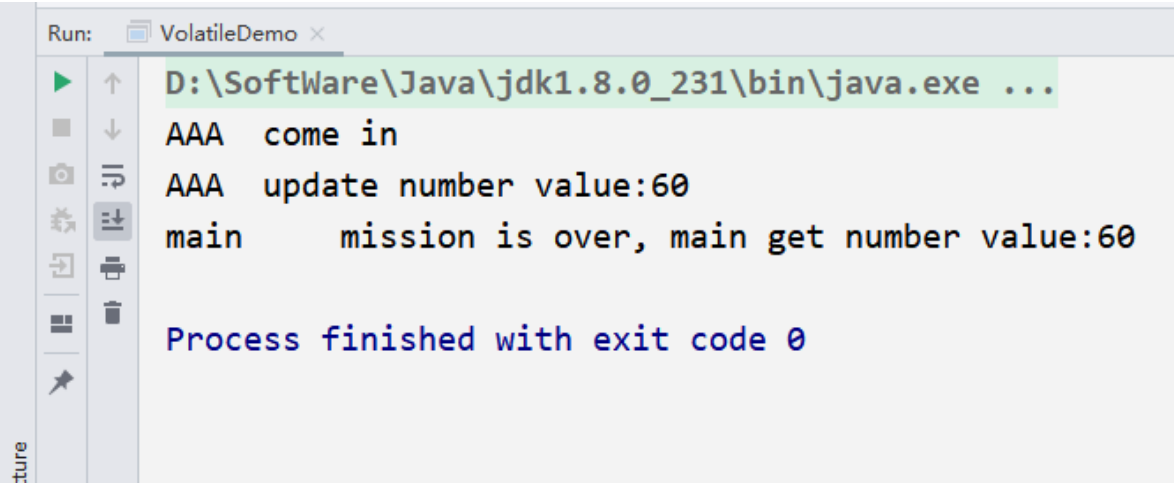
解释说明:

资源类MyData中的number变量如果没有volatile修饰，则运行结果:



程序陷入死循环，因为没有可见性，main线程并不知道number的值被修改。

如果加上volatile修饰，则运行结果:



Process finished with exit code 0

因为有了volatile修饰，具有了可见性，AAA线程中将number的修改之后，会立刻通知main线程，number的值修改为了60，则退出死循环，并打印“main mission is over, main get number value:60”