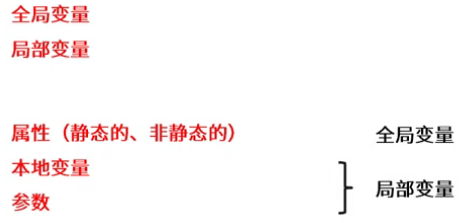
**Java之线程安全和变量可见性Volatile解密**

**一、并发中变量可见性问题**

1.什么是并发中的变量可见性问题

Q1 ：变量分为几类？



Q2：如何再多线程间共享数据？

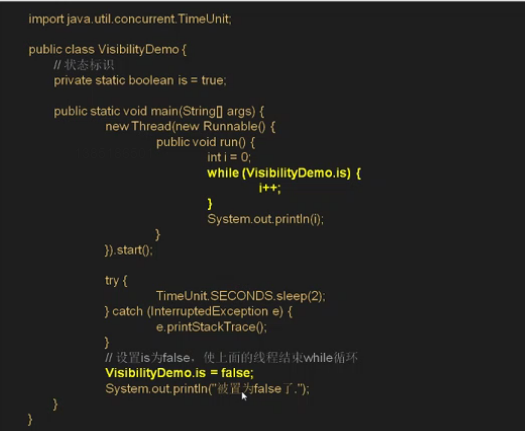
用全局变量：静态变量，或共享变量

Q3：一个变量在线程A中被改变值了，在线程B中能看到该变量的最新值吗？

不一定，分情况和场景

Demo1：示例代码

代码逻辑：通过共享变量，再一个线程中控制另一个线程的执行流程



请问：线程会停止循环吗？打印出i的值吗？ 答：不会

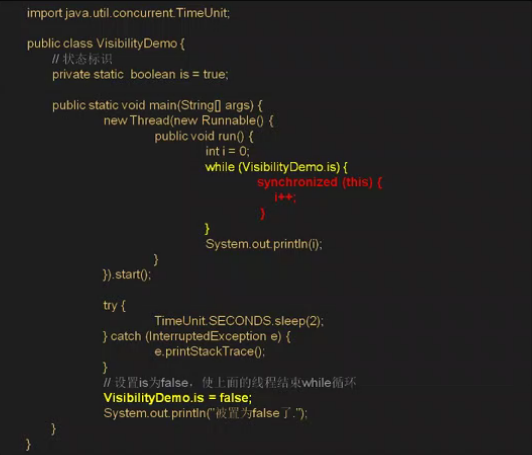
2.并发的线程能不能看到线程的最新值，这就是**并发中的变量可见性问题**。

Q1 为什么不可见？

Q2 怎么样才能可见？

3.怎么样才能可见

方式一：synchronized

’

方式二：volatile

**二、变量可见性、线程安全问题原因**

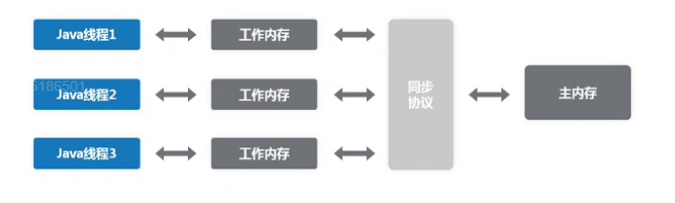
1. Java 内存模型

Java 内存模型及操作规范

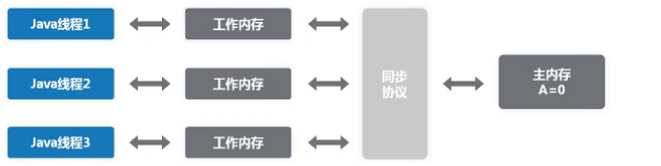
主内存：堆内存和方法区

工作内存：线程栈空间区

1. 共享内存必须存放在主内存中
2. 线程有自己的工作内存，线程只可操作自己的工作内存
3. 线程要操作共享变量，需要从主内存读取到工作内存，改变值后需从工作内存同步到主内存中。



2. Java内存模型带来的问题



思考：有变量 A，多线程并发对其累加会有什么问题？如三个线程并发操作A，大家读取A时都读取到A=0,都对A+1，再将值同步回主内存，结果是多少？ （答案：1）

**这就是线程安全问题。是否明白 线程安全问题的原因了-即JMM（Java 内存模型）？**

**如何解决线程安全问题？**

**内存模型也产生了变量可见性问题。**

3.如何让线程2使用A时看到最新值 ？

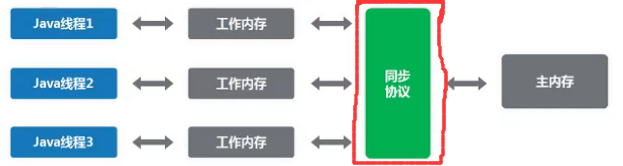
（1）线程1修改A后必须立马同步回主内存

（2）线程2使用A前需从主内存读取同步到工作内存

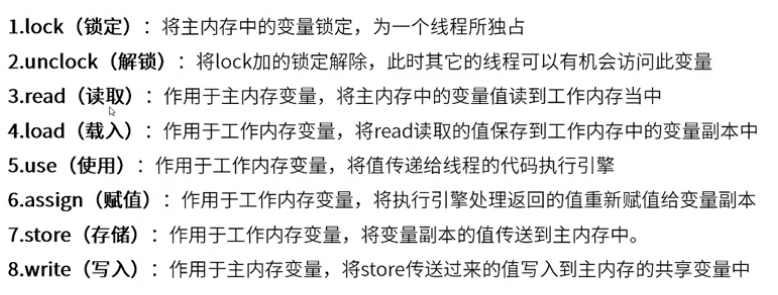
Q1：java变量使用前不会重新从主内存读取到工作内存吗？不一定

Q2：java 变量修改后不会立马同步回主内存吗？不一定

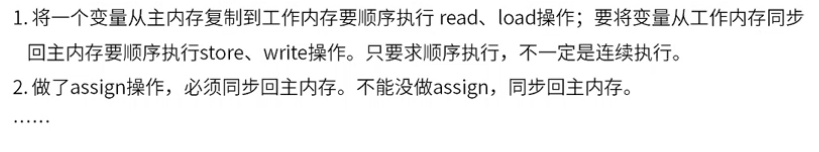
4. Java内存模型中同步交互协议



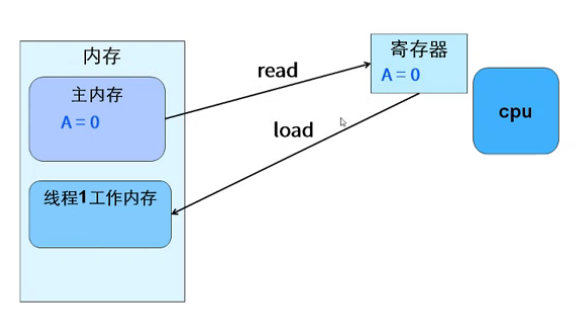
Java存模型中同步交互协议，规定了8种原子操作



同步协议中的操作规范



操作示例



5. 保证变量可见性的方式

并发中保证变量可见性的方式：



6. synchronized怎么做到可见性的

Synchronized 语义规范：

1. 进入同步块前，先清空工作内存中的共享变量，从主内存中重新加载
2. 解锁前必须把修改的共享变量同步回主内存中

Synchronized是如何做到线程安全的？

1. 锁机制 保护了共享资源，只有获得锁的线程才可操作共享资源
2. Synchronized语义规范保证了修改共享资源后，会同步回主内存，就做到了线程安全

**三、volatile关键字解密**

1. volatile怎么做到可见性

因为volatile的语义规范：

1. 使用volatie变量时，必须重新从主内存加载，并且read、load是连续的
2. 修改volatile变量后，必须立马同步回主内存，并且store、write是连续的

Volatie 能做到线程安全吗？

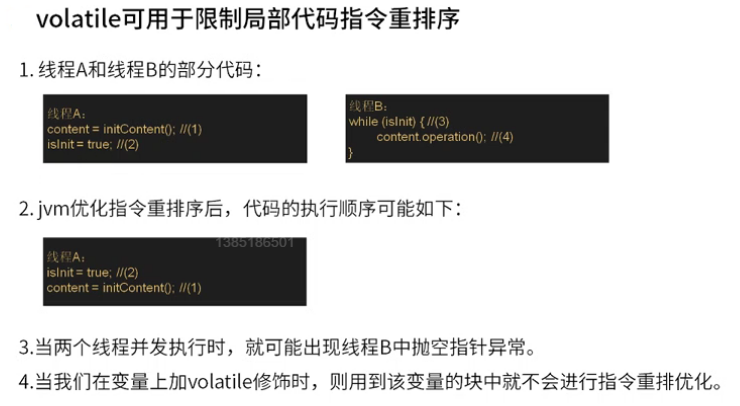
1. 不能，因为他没有锁机制，线程可并发操作共享资源

2. 为什么要使用volatile

Synchronized可以保证可见性，为什么还要用volatile?

1. 主要原因：使用volatile比synchronized简单
2. Volatile性能比synchronized稍好

3. volatile还有什么用途



.

4. volatile的使用场景

Volatile的使用范围

1. volatile只可修饰成员变量（静态的，非静态的）
2. 多线程并发下，才需要它

Volatile典型的应用场景

1. 只有一个修改者，多个使用者，要求保证可见性的场景
2. 状态标识，如示例中的介绍标识
3. 数据定期发布，多个获取者

（2）单例模式：为了线程尽量少的进入同步块，提高代码效率