## 1）内存可见性

* 内存可见性（Memory Visibility）：多个线程对同一个变量（称为：共享变量）进行操作，但是这多个线程有可能被分配到多个处理器中运行，那么编译器会对代码进行优化，当线程要处理该变量时，多个处理器会将变量从主存复制一份分别存储在自己的存储器中，等到进行完操作后，再赋值回主存。这样做的好处是提高了运行的速度，同样优化带来的问题之一是变量可见性——如果线程t1与线程t2分别被安排在了不同的处理器上面，那么t1与t2对于变量A的修改时相互不可见，如果t1给A赋值，然后t2又赋新值，那么t2的操作就将t1的操作覆盖掉了，这样会产生不可预料的结果。因此，需要保证变量的可见性（一个线程对共享变量值的修改，能够及时地被其它线程看到）。

注意：共享数据的访问权限必须定义为private。

* 可见性错误是指当读操作与写操作在不同的线程中执行时，我们无法确保执行读操作的线程能适时地看到其他线程写入的值，有时甚至根本不可能的事情。
* 我们可以通过同步来保证对象被安全地发布。除此之外我们也可以使用一种更加轻量级的volatile变量。

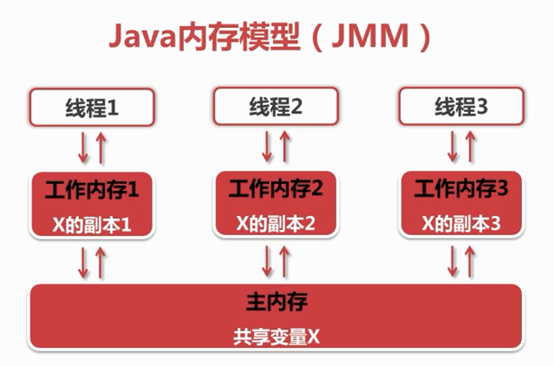
## 2）volatile关键字

Java提供了一种削弱的同步机制，即volatile变量，用来确保将变量的更新操作通知到其他线程。可以将volatile看作一个轻量级的锁，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”。
* volatile变量在每次被线程访问时，都强迫从主线程中重读该变量的值，而当该变量发生变化时，又会强迫线程将最新的值刷新到主内存中，这样任何时刻，不同的线程总能看到该变量最新的值。

*/\*\*  
 \* volatile关键字：当多个线程进行操作共享数据时，可以保证内存中的数据可见。  
 \*/***public class** TestVolatile {  
  
 */\*\*  
 \* mian线程（读flag值）、子线程1（改flag值）  
 \* <p>  
 \* JVM会为每个线程分配独立的缓存空间  
 \*  
 \** ***@param:*** *[args]  
 \** ***@return:*** *void  
 \*/* **public static void** main(String[] args) {  
 ThreadDemo threadDemo = **new** ThreadDemo();  
 **new** Thread(threadDemo).start();  
  
 **while** (**true**) {  
 **if** (threadDemo.isFlag()) {  
 System.***out***.println(**"----------------"**);  
 **break**;  
 }  
 }  
  
 }  
}  
  
**class** ThreadDemo **implements** Runnable {  
  
 **private volatile boolean flag** = **false**;  
  
 @Override  
 **public void** run() {  
  
 **try** {  
 Thread.*sleep*(200);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 **flag** = **true**;  
  
 System.***out***.println(**"flag="** + isFlag());  
 }  
  
 **public boolean** isFlag() {  
 **return flag**;  
 }  
  
 **public void** setFlag(**boolean** flag) {  
 **this**.**flag** = flag;  
 }  
}

## 多线程操作共享变量实现可见性过程JVM的内存模型如下：



所有的变量都存储在主内存中每个线程都有自己独立的工作内存，里面保存该线程使用到的变量的副本（主内存中该变量的一份拷贝）。

JVM模型两条规定：

1. 线程对共享变量的所有操作必须在自己的内存中进行，不能直接从主内存中读写。
2. 不同线程之间无法直接访问其它线程工作内存中的变量，线程变量值的传递需要通过主内存来完成。

## 在Java语言层面支持的可见性实现原理方式有synchronized和volatile

synchronized能够实现代码的原子性（同步）和内存的可见性。加锁效率低

线程执行互斥代码的过程：

1. 获得互斥锁；
2. 情况工作内存；
3. 从主内存拷贝变量的最新的值到工作内存；
4. 执行代码；
5. 将更改后的共享变量的值刷新到主内存；
6. 释放互斥锁。

volatile保证变量的可见性，不能保证变量的符合操作原子性。

写volatile变量的过程：

1. 改变线程工作内存中volatile变量副本的值；
2. 将改变后的副本值从工作内存刷新到主内存。
3. 线程读volatile变量的过程：
4. 从主内存中读取volatile变量的最新值到线程的工作内存中；
5. 从工作内存中读取volatile变量的副本。