volatile的内存语义及实现：  
　　在JMM中，线程之间的通信采用共享内存来实现的。  
volatile内存语义是：

* 当写一个volatile变量时，JMM会把该线程对应的本地内存中的共享变量值立即刷新到主内存中；
* 当读一个volatile变量时，JMM会把该线程对应的本地内存设置为无效，直接从主内存中读取共享变量。

https://www.cnblogs.com/xrq730/p/7048693.html

volatile关键字的作用是保证变量在多线程之间的可见性，它是java.util.concurrent包的核心，没有volatile就没有这么多的并发类给我们使用。

CPU缓存的出现主要是为了解决CPU运算速度与内存读写速度不匹配的矛盾，因为CPU运算速度要比内存读写速度快得多。

访问速度的显著差异，导致CPU可能会花费很长时间等待数据到来或把数据写入内存。基于此，现在CPU大多数情况下读写都不会直接访问内存（CPU都没有连接到内存的管脚），取而代之的是CPU缓存，CPU缓存是位于CPU与内存之间的临时存储器，它的容量比内存小得多但是交换速度却比内存快得多。而缓存中的数据是内存中的一小部分数据，但这一小部分是短时间内CPU即将访问的，当CPU调用大量数据时，就可先从缓存中读取，从而加快读取速度。

当一个CPU修改缓存中的字节时，服务器中其他CPU会被通知，它们的缓存将视为无效

汇编语言lock指令的几个作用：

1. 锁总线，其它CPU对内存的读写请求都会被阻塞，直到锁释放，不过实际后来的处理器都采用锁缓存替代锁总线，因为锁总线的开销比较大，锁总线期间其他CPU没法访问内存
2. lock后的写操作会回写已修改的数据，同时让其它CPU相关缓存行失效，从而重新从主存中加载最新的数据
3. 不是内存屏障却能完成类似内存屏障的功能，阻止屏障两遍的指令重排序

（1）中写了由于效率问题，实际后来的处理器都采用锁缓存来替代锁总线，这种场景下多缓存的数据一致是通过缓存一致性协议来保证的

缓存是分段（line）的，一个段对应一块存储空间，我们称之为缓存行，它是CPU缓存中可分配的最小存储单元，大小32字节、64字节、128字节不等，这与CPU架构有关，通常来说是64字节。当CPU看到一条读取内存的指令时，它会把内存地址传递给一级数据缓存，一级数据缓存会检查它是否有这个内存地址对应的缓存段，如果没有就把整个缓存段从内存（或更高一级的缓存）中加载进来。注意，这里说的是一次加载整个缓存段，这就是上面提过的局部性原理

工作内存Work Memory其实就是对CPU寄存器和高速缓存的抽象，或者说每个线程的工作内存也可以简单理解为CPU寄存器和高速缓存。

那么当写两条线程Thread-A与Threab-B同时操作主存中的一个volatile变量i时，Thread-A写了变量i，那么：

* Thread-A发出LOCK#指令
* 发出的LOCK#指令锁总线（或锁缓存行），同时让Thread-B高速缓存中的缓存行内容失效
* Thread-A向主存回写最新修改的i

Thread-B读取变量i，那么：

* Thread-B发现对应地址的缓存行被锁了，等待锁的释放，缓存一致性协议会保证它读取到最新的值

由此可以看出，volatile关键字的读和普通变量的读取相比基本没差别，差别主要还是在变量的写操作上。

Volitile修饰的变量会被加一个lock指令，这个指令做两件事情，

1.在变量改变之后，会立刻从cpu高速缓存写到内存

2.会通知其他cpu缓存中的该变量的值设置成无效，用到该变量时会到内存中重新读取该变量的值。

Lock指令保证了缓存一致性原理。

常用的缓存一致性协议都是属于“snooping(窥探)”协议，各个核能够时刻监控自己和其他核的状态，从而统一管理协调。窥探的思想是：CPU的各个缓存是独立的，但是内存却是共享的，所有缓存的数据最终都通过总线写入同一个内存，因此CPU各个核都能“看见”总线，即各个缓存不仅在进行内存数据交换的时候访问总线，还可以时刻“窥探”总线，监控其他缓存在干什么。因此当一个缓存在往内存中写数据时，其他缓存也都能“窥探”到，从而按照一致性协议保证缓存间的同步。

volatile，可以当之无愧的被称为Java并发编程中“出现频率最高的关键字”，常用于保持内存可见性和防止指令重排序。

### **volatile如何保持内存可见性**

volatile的特殊规则就是：

* read、load、use动作必须连续出现。
* assign、store、write动作必须连续出现。

所以，使用volatile变量能够保证:

* 每次读取前必须先从主内存刷新最新的值。
* 每次写入后必须立即同步回主内存当中。

也就是说，volatile关键字修饰的变量看到的随时是自己的最新值。线程1中对变量v的最新修改，对线程2是可见的。