volatile：可见性、有序性

- Java内存模型 JMM为了实现变量的可见性、有序性(禁止指令重排)而定义的一个关键字

可见性?

CPU和内存之间处理速度的不匹配，因此CPU和内存之间存在高速缓存，JMM将这个抽象成内存-线程工作内存-线程这种模型；假设多个线程都对内存中变量A进行拷贝到了各自的工作内存，那么当某个线程对变量A进行修改后,写到其自己的工作内存中，其它线程并不知道这个变量发生了改变，这个现象叫做可见性。

有序性?

执行程序时，为了提高性能，编译器和处理器会对指令进行重排序，重排序分3种类型：

- 编译器优化的重排序。编译器在不改变单线程程序语义的前提下，可以重新安排语句的执行顺序.
- 指令级并行的重排序。现代处理器采用了指令级并行技术（Instruction-LevelParallelism，ILP）来将多条指令重叠执行．如果不存在数据依赖性，处理器可以改变语句对应 机器指令的执行顺序.
- 内存系统的重排序。由于处理器使用缓存和读/写缓冲区，这使得加载和存储操作看上去可能是在乱序执行.

volatile实现可见性和有序性的底层原理：

在读写volatile变量时,对该指令加lock前缀(汇编指令),lock指令会在相应的读写指令前后加内存屏障,以保证 "工作区变量的缓存行被刷到内存中"，并且保证读的变量是"从内存中直接读取"，并且会禁止指令重排序;

面试掌握内容

更准确的理解

上面对volatile的理解是基于<<并发编程的艺术>>这本书的解释，它对volatile变量的语义解释是准确的，但是底层原理的描述，据资料可能是错误的;该书指出lock指令是利用CPU的MESI协议来完成变量可见性，即缓存一致性,那么是否CPU对非volatile变量就不开启MESI协议呢?

实质上CPU始终开启MESI,但是由于MESI为了保证缓存一致性，当某个CPU共享缓存中变量被修改时，需要对其它拥有这个缓存的CPU-0发送read invalid信号，其它CPU-x收到信号后,要设置缓存行invalid,发invalid- ack. CPU-0要接收到所有CPU-x的invalid-ack,才会修改其缓存，这样MESI协议是能保存缓存一致性的,但是由于这样需要CPU-0进行同步等待，效率低，因此设立了storebuffer和invalidqueue，cpu-0直接将变量修改值写到storebuffer就进行其它操作，这样就不用同步了，但是这样带来了缓存不一致，也就是storebuffer的操作延迟，CPU-0的修改对其它CPU-x不可见．综上,MESI并不能完全保证缓存一致性;

> lock的操作，通过对指令加读写屏障，将storebuffer刷到缓存，清空invalidqueue，来借助MESI保证缓存一致性；因此，valid实质增强MESI保证缓存一致性，而不是刷回内存，而是刷新缓存。但是由于MESI会保证缓存一致(因为storebuffer invalidqueue已经清空了)，其语义上相当于写入内存，和从内存读取最新值。

---

volatile为什么不保证原子性?

```
线程1(CPU1) {x++;} 线程2(CPU2) {x++;}
```

假设x首先在内存中，线程1读取x,缓存变为E，线程2读取x，CPU1-CPU2缓存变为S;

CPU1先执行 x+1 =1 ->temp(寄存器)[CPU2也读取x=0,此时缓存都为S有效],尝试写入CPU1缓存，通知CPU2缓存失效，此时CPU2中缓存更新为X=1,由于CPU2中**已经将x读入寄存器**则可直接执行x+1 = 1,也写入缓存，此时内存最后保留的x=1，失去原子性;

volatile不保证原子性，那它的使用场景?

- 可见性
  一个线程修改变量，其它线程读变量(例如作为条件判断)
- 有序性
  `instance = new CheckManager();`

  上面的对象初始化有三个操作(分配空间、初始化、返回对象内存空间首地址)，volatile禁止指令重排，防止先返回未初始化完成的地址

参考资料

- <<Java并发编程的艺术>>
- https://blog.csdn.net/weixin_42762133/article/details/105264806
- https://www.zhihu.com/question/296949412/answer/747494794
- https://blog.csdn.net/wll1228/article/details/107775825?spm=1001.2014.3001.5501
- https://blog.csdn.net/wll1228/article/details/107775976?spm=1001.2014.3001.5501
- https://www.jianshu.com/p/c9c77d771221