参考文档：

<https://www.cnblogs.com/fanguangdexiaoyuer/p/10743619.html>

happens-before规则

**Java内存模型具备一些先天的“有序性”**

下面就来具体介绍下happens-before原则（先行发生原则）：

* **程序次序规则**：一个线程内，按照代码顺序，书写在前面的操作先行发生于书写在后面的操作
* **锁定规则**：一个unLock操作先行发生于后面对同一个锁额lock操作
* **volatile变量规则**：对一个变量的写操作先行发生于后面对这个变量的读操作
* **传递规则**：如果操作A先行发生于操作B，而操作B又先行发生于操作C，则可以得出操作A先行发生于操作C
* 线程启动规则：Thread对象的start()方法先行发生于此线程的每个一个动作
* 线程中断规则：对线程interrupt()方法的调用先行发生于被中断线程的代码检测到中断事件的发生
* 线程终结规则：线程中所有的操作都先行发生于线程的终止检测，我们可以通过Thread.join()方法结束、Thread.isAlive()的返回值手段检测到线程已经终止执行
* 对象终结规则：一个对象的初始化完成先行发生于他的finalize()方法的开始

一旦一个共享变量（类的成员变量、类的静态成员变量）被volatile修饰之后，那么就具备了两层语义：

　　1）保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的。

　　2）禁止进行指令重排序。

volatile修饰的变量，更新后会立即写入主存，写入主存的操作会导致变量在所有线程中的缓存失效。从而来达到可见性。

volatile不能当做锁用

atomic是利用CAS来实现原子性操作的（Compare And Swap），**CAS实际上是利用处理器提供的CMPXCHG指令实现的**，而处理器执行**CMPXCHG**指令是一个原子性操作。

volatile关键字禁止指令重排序有两层意思：

　　1）当程序执行到volatile变量的读操作或者写操作时，在其前面的操作的更改肯定全部已经进行，且结果已经对后面的操作可见；在其后面的操作肯定还没有进行；

　　2）在进行指令优化时，不能将在对volatile变量访问的语句放在其后面执行，也不能把volatile变量后面的语句放到其前面执行。

**volatile的原理和实现机制（内存屏障）**

前面讲述了源于volatile关键字的一些使用，下面我们来探讨一下volatile到底如何保证可见性和禁止指令重排序的。

　　下面这段话摘自《深入理解Java虚拟机》：

　　“观察加入volatile关键字和没有加入volatile关键字时所生成的汇编代码发现，**加入volatile关键字时，会多出一个lock前缀指令**”

　　lock前缀指令实际上相当于一个**内存屏障**（也成内存栅栏），内存屏障会提供3个功能：

　　1）它确保指令重排序时不会把其后面的指令排到内存屏障之前的位置，也不会把前面的指令排到内存屏障的后面；即在执行到内存屏障这句指令时，在它前面的操作已经全部完成；

　　2）它会强制将对缓存的修改操作立即写入主存；

　　3）如果是写操作，它会导致其他CPU中对应的缓存行无效。

volatile 保证变量可见性。当一个线程修改了共享变量，另一个线程能读取到新的值。

cpu缓存 数据总线 内存

cpu1 cpu2 cpu3 cpu4

cache1 cache2 cache3 cache4

------------------------------------------------------------------- 总线

============================================ 内存

没有volatile的变量，不确定什么时候将缓存中的数据写入内存

1、可见性

2、禁止指令重排——有序性