## Volatile

### Volatile的作用

1. 保证共享变量的可见性
2. 禁止指令重排

多线程并发编程围绕着三个特性来实现

1. 可见性：一个线程修改了共享变量，另一个线程能够获取这个共享变量修改之后的值
2. 原子性：一个操作或者一组操作要么全部执行，要么全部不执行
3. 有序性：程序执行的顺序按照代码的先后顺序执行

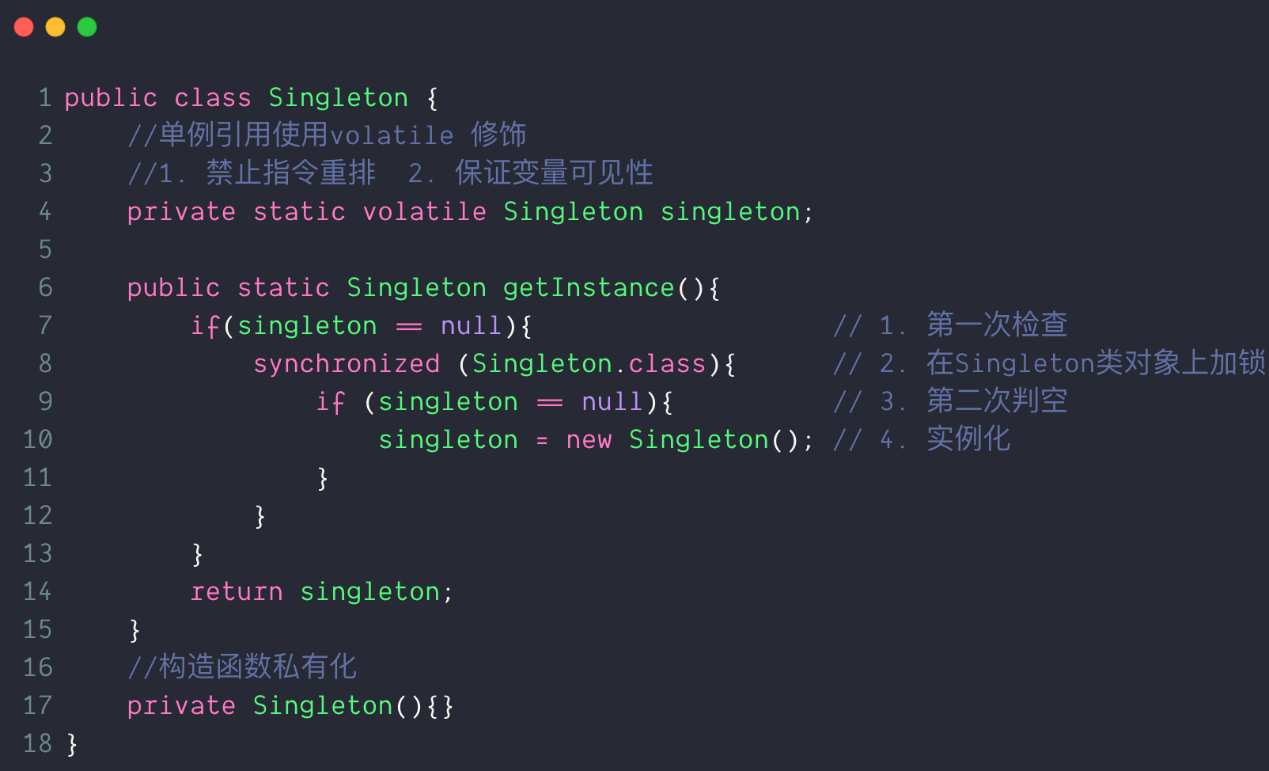
应用场景

1. 状态标志，比如在项目中定义一些变量来标识启动，初始化，是否停止等，volatile 很适合只有一个线程修改，其他线程读取的情况。volatile 变量被修改之后，对其他线程立即可见。



比如这是一个带前端交互的系统，有A、 B二个线程，用户点了停止应用按钮，A 线程调用shutdown() 方法，让变量shutdown 从false 变成 true，但是因为没有使用volatile 修饰, B 线程可能感知不到shutdown 的变化，而继续执行 doWork 内的循环，这样违背了程序的意愿：当shutdown 变量为true 时，代表应用该停下了，doWork函数应该跳出循环，不再执行。

1. 单例模式的double check实现方法



这个单例模式的几个疑问

**为什么使用volatile 修饰了singleton 引用还用synchronized 锁？**

Volatile能够保证修饰变量的可见性，但是不能保证操作的原子性；创建对象的过程可以分为3步：

步骤1：在堆内存申请一块内存空间；

步骤2：初始化申请好的内存空间；

步骤3：将内存空间的地址赋值给 singleton；

所以singleton = new Singleton(); 是一个由三步操作组成的复合操作，多线程环境下A 线程执行了第一步、第二步之后发生线程切换，B 线程开始执行第一步、第二步、第三步（因为A 线程singleton 是还没有赋值的），所以为了保障这三个步骤不可中断，可以使用synchronized 在这段代码块上加锁

**第一次检查singleton 为空后为什么内部还需要进行第二次检查？**

A 线程进行判空检查之后开始执行synchronized代码块时发生线程切换(线程切换可能发生在任何时候)，B 线程也进行判空检查，B线程检查 singleton == null 结果为true，也开始执行synchronized代码块，虽然synchronized 会让二个线程串行执行，如果synchronized代码块内部不进行二次判空检查，singleton 可能会初始化二次。

### 底层实现

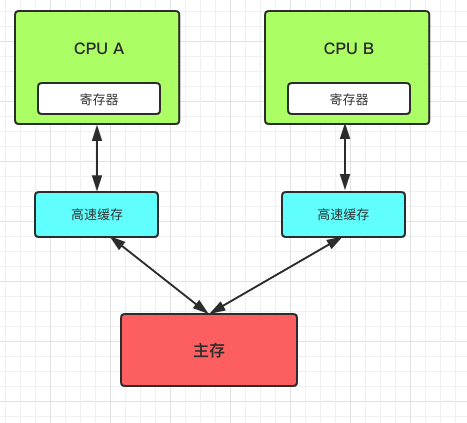
可见性

背景

因为CPU 运行速度实在太快，主存（就是内存）的数据读取速度和CPU 运算速度差了有几个数量级，因此现代计算机系统通过在CPU 和主存之前加了一层读写速度尽可能接近CPU 运行速度的高速缓存来做数据缓冲，这样缓存提前从主存获取数据，CPU 不再从主存取数据，而是从缓存取数据。这样就缓解由于主存速度太慢导致的CPU 饥饿的问题。同时CPU 内还有寄存器，一些计算的中间结果临时放在寄存器内。

为了提高处理速度，处理器不直接和内存进行数据交互，而是先将系统内存的数据读到内部缓存中（L1,L2）再进行操作，但是操作不知道何时会写到内存

Cpu,系统内存，缓存的关系图如下

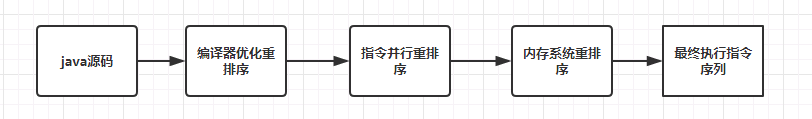


当使用volatile来修饰变量时，jvm会向处理器发送一个Lock前缀指令，将这个变量所在的缓存行的数据回写到系统内存中；

变量回写到内存中后，其他处理器缓存行中的数据仍是之前缓存的旧值；所以在多处理器下，为了保证各个处理器的缓存一致性，就会使用缓存一致性协议，每个处理器通过嗅探在总线上传播的数据来检查自己缓存的值是不是过期值，当发现自己缓存行对应的内存地址被修改，就会将当前处理器的缓存行设置为无效状态，处理器对这个数据进行修改时，会从系统内存中读取出来缓存到自己的缓存行中

重排序

1. 编译器优化重排序：编译器在不改变程序语义的前提下，可以重新安排语句的执行顺序
2. 指令级并行重排序：处理器采用指令级并行技术来将多条指令重叠执行，如果不存在数据的依赖性，处理器可以改变语句对应机器指令的执行顺序
3. 内存系统重排序：由于处理使用缓存和读写缓冲区，这使得加载和存储操作看上去是乱序的



编译器重排序发生在编译器，指令级并行重排序和内存系统重排序时处理器重排序，在多线程环境下，重排序可能会导致内存可见性问题

数据依赖性

如果2个操作访问同一个变量，且这两个操作中有一个是写操作，此时这两个操作之间就存在数据依赖性；

编译器和处理器可能会对操作做重排序时，会遵循数据依赖性，不会改变存在数据依赖关系的两个操作的执行顺序

as-if-serial:不管怎样重排序，执行结构都不能改变；编译器，Runtime和处理器都必须遵守as-if-serial,编译器和处理器都不会对存在数据依赖的操作进行重排序，因为这种排序会改变执行结果

重排序对多线程的影响