java内存模型

一.原子操作的实现原理

1.处理器实现原子操作：

处理器提供总线锁和缓存锁定两种方式来保证复杂内存操作的原子性。

- 总线锁：就是使用处理器提供一个LOCK信号，当一个处理器在总线传输信号时，其他处理器的请求将被阻塞住，那么该处理独占共享内存。所以总线锁开大。
- 缓存锁定：内存区域如果被缓存在缓存行中，且在在lock期间被锁定，当它执行锁操作回写到内存时，处理器总线不再锁定而是通过修改内部的内存地址并使用缓存一致性机制来保证操作的原子性。缓存一致性会阻止同时修改由两个以上的处理器同时缓存的内存区域数据，核心思想是当CPU写数据时，如果发现操作的变量是共享变量，即在其他CPU中也存在该变量的副本，会发出信号通知其他CPU将该变量的缓存行置为无效状态。

2.java实现原子操作

- 可以通过锁和循环CAS的方式实现原子操作
- 使用循环CAS实现原子性操作，CAS是在操作期间先比较旧值，如果旧值没有发生改变，才交换成新值，发生了变化则不交换。

CAS会产生以下几种问题：

+

- ABA问题，通过加版本号解决；
- 循环时间过长开销大，一般采用自旋方式实现；
- 只能保证一个共享变量的原子操作。

二.happens-before原则

Java内存模型控制线程之间的通信，决定了一个线程对共享变量的写入何时对另一个线程可见。它属于语言级的内存模型

为了提供内存可见性保证，JMM向程序员保证了以下hapens-before规则:

- 程序顺序规则：一个线程中的每个操作happen-before于该线程中的任意后续操作。
- 监视器锁规则：一个锁的解锁，happens-before于随后对这个锁的加锁。
- Volatile变量规则：对一个volatile域的写，happens-before于任意后续对这个域的读。
- 传递性：如果A happens-before B, 且B happens-before C 那么A happens-before C
- **start()规则：**如果线程A执行操作ThreadB.start()，那么线程A中的ThreadB.start()操作happens-before线程B中的任意操作。
- **join()规则：**如果线程A执行操作ThreadB.join()并成功返回，则线程B中的任意操作happens-before于线程A从ThreadB.join()操作成功返回

补充：

- 线程结束规则: 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
- 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
- 终结器规则: 一个对象的初始化完成先行发生于他的finalize()方法的开始

三.重排序

在Java内存模型中，允许编译器和处理器为优化程序性能对指令进行重排序，但是重排序过程不会影响到单线程程序的执行结果，却会影响到多线程并发执行的正确性， 它只会对不存在数据依赖性的指令进行重排序。

为了保证内存可见性，编译器在生成指令序列的适当位置插入内存屏障指令来禁止特定类型的处理器重排序。

四.原子性

JMM仅保证了变量的简单读取和赋值（赋值必须是将值赋值给变量，不包括变量之间的相互赋值）才是原子操作

五.Final域的内存语义

对于final域，编译器和处理器要遵守两个重排序规则：

- 在构造器函数内对一个final域的写入，与随后把这个被构造对象的引用赋值给一个引用变量，这两个操作之间不能重排序。（保证了对象引用为任何线程可见之前，对象的final域已经被正确初始化过）
- 初次读一个包含final域的对象引用，与随后初次读这个final域这两个操作不能重排序（在读final域之前，一定会先读包含这个final域的对象引用，那引用对象的final域一定被初始化过）

只要对象是正确构造的，那么不需要使用同步就可以保证线程都能看到这个final域在构造函数中被初始化之后的值。