Java多线程编程

1. 工作内存与主内存

Java内存模型中，有一个主内存和许多每个线程独有的工作内存。

线程对变量的修改都发生在自己的工作内存中，如果想要让其它线程看到自己对变量的修改，就必须把工作内存的值刷新到主内存中，其它线程再通过读取主内存，才能够看到最新的修改过的值。

2. 内存可见性

内存的可见性指的是，对于一个共享变量，线程A的更新没有及时地更新到主内存中，这时发生了CPU的线程切换，开始执行线程B的任务导致，此时线程B读取到了非最新的值，再进行修改，最后导致的问题可能就是A,B中的更新操作会丢失一部分，程序错误。

所以，导致问题的根本原因在于线程A的更新操作不具备原子性，还没有把最新的值写入主内存，就发生了线程切换。

那么，为了解决原子性与刷新主内存的问题，就引入了synchronized关键词。

3. synchronized关键词

我们都知道Synchronized关键词通过给代码块或者函数加锁，可以保证原子性的问题，解决data race的问题，但是synchronized和工作内存、主内存又有什么关系呢？他是怎么保证原子性的？

Synchronized可以保证：

在线程解锁前，一定会把共享变量的最新值，刷新到主内存中。

在线程加锁时，一定会轻松工作内存中共享变量的值，并且会到主内存中读取最新的值。

这样就可以保证加锁时，共享变量一定是最新值得状态，而解锁前，最新值一定会被刷新到主内存中，被其他的线程所看到。

4. 指令重排序与多线程

程序代码的执行顺序并非总是和我们编写的代码一致，java编译器有时会主动进行一些指令的重排序，只要保证最后的结果不变即可。

指令的重排序会导致在多线程编程中出现很多奇怪的问题，如果两个线程之间通过一个共享变量进行控制依赖的话，如果没有使用volatile、synchronized等关键词，另一个线程可能永远看不到共享变量的更新，或者看的更新的时间顺序会违背代码的编写顺序。

5. volatile关键词

Volatile关键词是相比于synchronized更加弱的同步机制，能够保证对于共享变量的更新操作对于其它线程是可见的。

如果一个变量加了volatile关键词，编译器会知道他是一个共享变量，就不会讲该变量的操作与其他内存操作进行重排序。

在对volatile共享变量进行操作时，不会有任何的加锁操作，所以volatile只能保持弱一致性。

对于volatile变量的读操作，保证每一次都是从主内存中获取。

Volatile可以保证：对于共享变量的操作一定会立刻刷新到主内存中，每次操作都会从主内存中读取最新的值；禁止指令重排，在volatile操作的指令后会自动加入一条关于内存屏障的指令，java编译器不能讲屏障后面的指令重排到内存屏障的前面。