JMM（Java内存模型详解）

1. 内存模型 (java memory model):

内存模型描述的是程序中各变量（实例域、静态域和数组元素）之间的关系，以及在实际计算机系统中将变量存储到内存和从内存取出变量这样的低层细节.

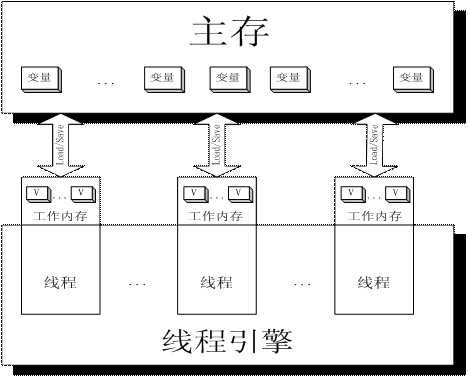
不同平台间的处理器架构将直接影响内存模型的结构.

在C或C++中, 可以利用不同操作平台下的内存模型来编写并发程序. 但是, 这带给开发人员的是, 更高的学习成本.相比之下, Java利用了自身虚拟机的优势, 使内存模型不束缚于具体的处理器架构, 通过Java内存模型真正实现了跨平台.(针对hotspot jvm, jrockit等不同的jvm, 内存模型也会不相同).

1. 内存模型的特征:
   1. Visibility 可见性(多核,多线程间数据的共享)
   2. Ordering 有序性(对内存进行的操作应该是有序的)
2. Java内存模型(java memory model):

jvm系统中存在一个主内存(Main Memory或Java Heap Memory)，Java中所有变量都储存在主存中，对于所有线程都是共享的。

每条线程都有自己的工作内存(Working Memory)，工作内存中保存的是主存中某些变量的拷贝，线程对所有变量的操作都是在工作内存中进行，线程之间无法相互直接访问，变量传递均需要通过主存完成。



其中, 工作内存里的变量, 在多核处理器下, 将大部分储存于处理器高速缓存中, 高速缓存在不经过内存时, 也是不可见的.

JMM怎么体现可视性(Visibility) ?

在JMM中, 通过并发线程修改变量值, 必须将线程变量同步回主存后, 其他线程才能访问到.

JMM怎么体现有序性(Ordering) ?

通过Java提供的同步机制或volatile关键字, 来保证内存的访问顺序.

1. 缓存一致性（cache coherency）

什么是缓存一致性?

它是一种管理多处理器系统的高速缓存区结构，其可以保证数据在高速缓存区到内存的传输中不会丢失或重复。(来自wikipedia)

举例理解:

假如有一个处理器有一个更新了的变量值位于其缓存中，但还没有被写入主内存，这样别的处理器就可能会看不到这个更新的值.

解决缓存一致性的方法?

a, 顺序一致性模型:

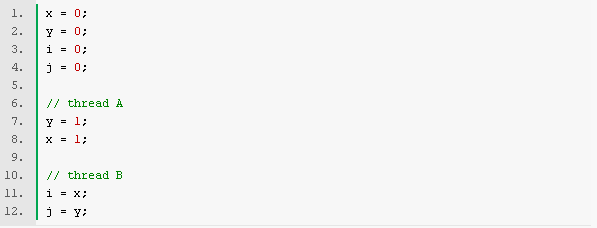
要求某处理器对所改变的变量值立即进行传播, 并确保该值被所有处理器接受后, 才能继续执行其他指令.

b, 释放一致性模型: (类似jmm cache coherency)

允许处理器将改变的变量值延迟到释放锁时才进行传播.

Java内存模型的缓存一致性模型 - "happens-before ordering(先行发生排序)"

一般情况下的示例程序:



在如上程序中, 如果线程A,B在无保障情况下运行, 那么i,j各会是什么值呢?

答案是, 不确定. (00,01,10,11都有可能出现)，这里没有使用Java同步机制, 所以Java内存模型有序性和可见性都无法得到保障.

happens-before ordering( 先行发生排序) 如何避免这种情况?

排序原则已经做到:

a, 在程序顺序中, 线程中的每一个操作, 发生在当前操作后面将要出现的每一个操作之前.

b, 对象监视器的解锁发生在等待获取对象锁的线程之前.

c, 对volitile关键字修饰的变量写入操作, 发生在对该变量的读取之前.

d, 对一个线程的 Thread.start() 调用 发生在启动的线程中的所有操作之前.

e, 线程中的所有操作 发生在从这个线程的 Thread.join()成功返回的所有其他线程之前.

为了实现 happends-before ordering原则, Java及JDK提供的工具:

a, synchronized关键字

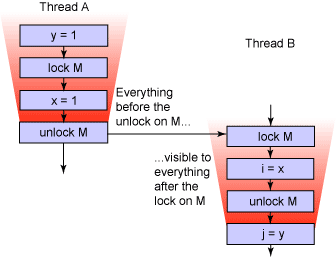
b, volatile关键字

c, final变量

d, java.util.concurrent.locks包(since jdk 1.5)

e, java.util.concurrent.atmoic包(since jdk 1.5)

使用了happens-before ordering的例子:



(1) 获取对象监视器的锁(lock)

(2) 清空工作内存数据, 从主存复制变量到当前工作内存, 即同步数据 (read and load)

(3) 执行代码，改变共享变量值 (use and assign)

(4) 将工作内存数据刷回主存 (store and write)

(5) 释放对象监视器的锁 (unlock)

注意: 其中4,5两步是同时进行的.

这边最核心的就是第二步, 他同步了主内存,即前一个线程对变量改动的结果,可以被当前线程获知!(利用了happens-before ordering原则)

1. 对比之前的例子：

如果多个线程同时执行一段未经锁保护的代码段，很有可能某条线程已经改动了变量的值，但是其他线程却无法看到这个改动，依然在旧的变量值上进行运算，最终导致不可预料的运算结果。

参考为完：<http://kenwublog.com/explain-java-memory-model-in-detail>

待细读：<http://en.wikipedia.org/wiki/Double-checked_locking>

来源：

<http://kenwublog.com/explain-java-memory-model-in-detail>

<http://en.wikipedia.org/wiki/Double-checked_locking>