1. Java内存模型
2. 概述

Java内存模型（Java Memory Model, JMM），用来屏蔽各种硬件和操作系统的内存访问差异，已实现Java程序在各种平台下都能够达到一致的内存访问效果。

1. 主内存与工作内存

Java内存模型的主要目的是定义程序中各种变量的访问规则，这里所说的变量指的是实例变量，静态字段和构成数组对象的元素，但是不包含局部变量和方法参数，因为后者是线程私有的，不会被共享。

JMM规定了所有的变量都存储在主内存中，每条线程有自己的工作内存，工作内存中保存着该线程使用的变量的主内存副本，线程对变量的所有操作都必须在工作内存中进行，而不能直接读取主内存汇总的数据。不同的线程之间也无法直接访问对方工作内存中的变量，线程间变量值的传递需要通过主内存来完成。

主内从对应于物理硬件的内存，而为了获取更好的运行速度，虚拟机可能会让工作内存优先存储于寄存器和高速缓存中，因为程序运行时主要访问的是工作内存。

1. 内存的交互操作

JMM中定义了8中操作来完成内存之间的交互，每一种操作都是原子性的，不可再分，如下所示：

1. lock

作用于主内存的变量，标识为线程独占。

1. unlock

作用于主内存的变量，释放锁定状态的变量。

1. read

作用于主内存变量，将变量的值从主内存传输到线程的工作内存中，以便后续的load动作使用。

1. load

作用于工作内存变量，将read操作得到的变量放入工作内存的变量副本中。

1. use

作用于工作内存的变量，将该变量传递给执行引擎，当需要使用该变量时执行该操作。

1. assign

作用于工作内存变量，把一个从执行引擎接受的值赋给工作内存的变量。

1. store

作用于工作内存变量，将该变量传送到主内存中，以便write操作使用。

1. write

作用于主内存变量，将store操作得到的变量放入主内存的变量中。

如果要把一个变量从主内存拷贝到工作内存中，按顺序执行read和load操作；

如果将变量从工作内存同步到主内存中，按顺序执行store和write操作。注意，这两个操作都是按照顺序，而非必须连续。

1. volatile

关键字volatile时Java虚拟机提供的最轻量级的同步机制，它主要解决的是变量的可见性，即当一条线程修改了这个变量的值，其他线程可以立即得知修改后的新值。但是，它无法保证变量的原子性，在变量从工作内存同步会主内存的过程中，其他线程可能已经将该值给修改了，因此要保证该变量的原子性，还需要通过加锁来保证。

volatile关键字还能够禁止指令重排序优化，普通的变量不能保证变量赋值操作的顺序与程序代码中的执行顺序一致，它可能在指令层面已经被优化过了，导致了指令执行顺序是混乱的，但是最终结果是一致的，这种优化对于高并发可能会造成混乱，因此通过volatile关键字来禁止这种优化，通过插入内存屏障指令保证后面的指令不能重排序到内存屏障之前的位置，这也就让volatile修饰的变量的写操作比普通变量慢一些，因为有许多内存屏障指令，不过它的总开销仍然比锁要低。

因此，在工作内存中，每次使用volatile修饰的变量都会先从主内存中刷新最新的值；每次修改该变量后必须立即同步会主内存中。