
Java内存模型
3.1 Java内存模型的基础
并发编程模型的两个关键问题
线程之间如何通信
共享内存和消息传递
同步是指程序中用于控制不同线程间操作发生相对顺序的机制
线程之间如何同步
显示指定特定方法或片段
java内存模型的抽象数据结构
所有实例域,静态域,数组元素都存储在堆内存
线程A与线程B通信的步骤
1. 线程A把本地内存A中更新过的共享变量刷新到主内存中去
2. 线程B到主内存中去读取线程A之前修改的共享变量
JMM通过控制主内存和每个线程的本地内存之间的交互,来为java程序提供内存可见性
从源代码到指令序列的重排序
源代码->编译器优化重排序->指令级并行重排序->内存系统重排序->最终执行的指令序列
子主题 5
3.2 重排序
重排序是指编译器和处理器为了优化程序性能而对指令序列进行重排序的一种手段
3.3 顺序一致性
3.4 volatile的内存语义
volatile++不具有原子性
volatile写-读建立的happens-before关系
volatile 写-读的内存语义
volatile写的内存语义
当写一个volatile变量时,JMM会把该线程对应的本地内存中的共享变量值刷新到主内存
volatile读的内存语义
当读取一个volatile变量时,JMM会把该线程对应的本地内存置为无效.线程接下来将从主内存中读取共享变量
volatile内存语义的实现
编译器在生成字节码时,会在指令序列中插入内存屏障来禁止特定类型的处理器重排序
两者具有相同的内存语义
3.5 锁的内存语义
锁的释放-获取建立的happens-before关系
锁的释放和获取的内存语义
当线程释放锁时,JMM会把该线程对应的本地内存中的共享变量刷新到主内存中
当线程获取锁时,JMM会把该线程对应的本地内存置为无效,从而使得被监视器保护的临界区代码必须从主内存中读取共享变量
锁内存语义的实现
3.6 final域的内存语义
构造函数内对一个final域的写入与这个被构造对象的引用赋值给一个引用变量,这两个操作不能重排序
先读一个包含final域的引用,后读这个final域,这两个操作不能重排序
3.7 happens-before
JMM通过happens-before关系来向程序员提供跨线程的内存可见性保证
happens-before(正确的多线程)和as-if-serial(单线程)都是广义上的程序"顺序执行"(不影响结果的编译器和处理器重排序,不需要对程序员可见)
3.8 双重检查锁定与延迟初始化
volatile方案
基于类初始化的解决方案
3.9 java内存模型综述