13 问题的根源—Java内存模型简介

更新时间：2019-10-10 10:27:14

> 世界上最宽阔的是海洋，比海洋更宽阔的是天空，比天空更宽阔的是人的胸怀。
>
> ——雨果

JAVA内存模型即JMM（Java Memory Model），有些人会和Java内存结构混淆。虽然两者名字很接近，但描述的为不同内容。Java内存结构描述的是JVM对内存的逻辑划分，我们在学习垃圾回收和JVM优化的时候会关心JVM内存结构。而本节所讲的JMM，实际上是一种规范。它描述了Java程序的运行行为，包括多线程操作对共享内存读取时，所能读取到的值应该遵守的规则。

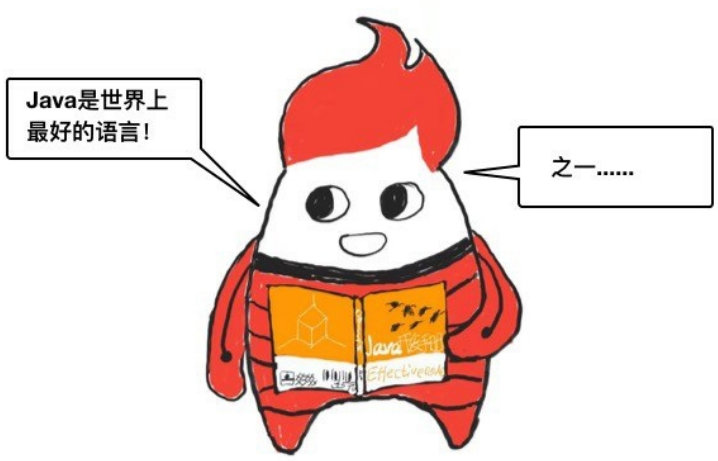
1. JMM的必要性

随着CPU的不断发展，CPU的性能越来越强大，但受迫于频率提升的困难，现代CPU架构开始向多核发展。而作为软件开发人员为了充分使用CPU的性能，越来越多的开发者会选择多线程程序开发。CPU在计算时会做一些优化，这些优化对于单线程程序来说是没有问题的，但对多线程程序则不是那么的友好。

计算机在运行时，绝大多数时间都会把对象信息保存在内存中。但是在此期间，编译器、处理器或者缓存都可能会把变量从分配的内存中取出处理再放回。比如我们在while循环中判断flag是否为true来执行一段特定的逻辑，那么编译器为了优化可能会选择把flag值取到缓存中。此时主存中的flag值可能会被其它线程所改变，但是此线程是无法感知的。直到某个特定的时机触发此线程从主存中刷新flag值。所有这些优化都是为了程序有更好的性能。在单线程的程序中，这种优化对于用户来讲是毫无感知的，不过多线程的程序中，这种优化有些时候会造成难以预料的结果。

JMM允许编译器和缓存保持对数据操作顺序优化的自由度。除非程序使用Synchronized或者volatile显式的告诉处理器需要确保可见性。这意味着如果你没有进行同步，那么多线程程序对于数据的操作，将会呈现不同的顺序。也就是前面一节讲的有序性，我们基于代码顺序对数据赋值顺序的推论，在多线程程序中可能会不成立。

在JMM之前，C和C++并没有显式的内存模型。C语言的内存模型继承自执行程序的处理器。这意味着并发的C语言程序，在不同的处理器机构上可能会呈现不一样的结果。但JMM使得Java程序能够在任何JVM上表现出一样的行为。当然，现在C和C++也有了内存模型。



2. JMM简介

JMM为程序中的所有操作定义了一定的规则，叫做Happens-Before。无论两个操作是否在同一个线程，如果要想保证操作A能看到操作B的结果，那么A、B之间一定要满足Happens-Before关系。如果两者间不满足Hapen-Before关系，JVM可以对其任意重排序。

当多个线程同时读写同一个变量，但这些操作间又没有满足Happens-Before关系，那么这些线程对此变量存在数据竞争，整个程序将会陷入混乱之中。假如我们在操作共享变量时采用了同步，那么无论有多少线程，对此变量的操作都会呈现出串行一致性。从而使得多线程的操作顺序遵守JMM约定。

3. Happens-Before规则

Happens-Before在多线程领域具有重大意义，它可以指导你如何开发多线程的程序，而不至于陷入混乱之中。你所开发的多线程程序，如果想对共享变量的操作符合你设想的顺序，那么需要依照Happens-Before原则来开发。happens-before并不是指操作A先于操作B发生，而是指操作A的结果在什么情况下可以被后面操作B所获取。下面我们就来看一下Happens-before原则。

1. 程序顺序规则。如果程序中A操作在B操作之前，那么线程中A操作将在B操作前执行。
2. 上锁原则。不同线程对同一个锁的lock操作一定在unclock前。
3. volatile变量原则。对于volatile变量的写操作会早于对其的读操作。
4. 线程启动原则。A线程中调用threadB.start()方法，那么threadB.start()方法会早于B线程中中的任何动作执行。
5. 传递规则。如果A早于B执行，B早于C执行，那么A一定早于C执行。
6. 线程中断规则：线程interrupt()方法的一定早于检测到线程的中断信号。
7. 线程终结规则：如果线程A终结了，并且导致另外一个线程B中的ThreadA.join()方法取得返回，那么线程A中所有的操作都早于线程B在ThreadA.join()之后的动作发生。

8. 对象终结规则：一个对象初始化操作肯定先于它的finalize()方法。

我们只有充分理解了happens-before原则，才能在编写多线程程序的时候，尽量避免数据的不一致性，让多线程程序在必要的时候按照我们设计的次序执行。

4. 总结

JMM是程序执行顺序的指导原则，通过JMM的约束，我们能够设计出符合我们要求的多线程程序。其实不止多线程，单线程程序一样受到JMM的约束。

}

12 什么？还有这种操作！—有序性

14 僵持不下—死锁详解