第12 章 java内存模型与线程

计算机的运行速度与它的存储和通信子系统速度的差距太大，大量时间花费在磁盘IO、网络通信或者数据库访问上。所以计算机可以并行处理多任务。

衡量一个服务性能的好坏，每秒事务处理数是最重要的指标之一，它代表着一秒内服务端平均能响应的请求总数。

物理机并发🡪jvm虚拟机并发

计算机的存储设备与处理器的运算速度有几个数量级的差异，现代计算机不得不加入一层缓存来作为内存与处理器之间的缓冲：将运算需要得数据复制到缓存中，让运算能快速进行，当运算结束后在从缓存同步回内存之中，这样处理器就无需等待缓慢的内存读写了。

基于高速缓存的存储交互解决类处理器和内存的速度矛盾，但是带来了新的问题：数据一致性。

多处理器每个都有自己的高速缓存，而且同时共享同一主存。

除了增加高速缓存之外，为了使得处理器内部的运算单元能尽量被充分利用，处理器可能对输入代码进行乱序执行优化（即指令重排），处理器会在计算之后将乱序执行的结果进行重组，保证该结果与顺序执行的结果是一致的。

Java内存模型

主内存与工作内存：

Java内存模型规定了所有的变量都存储在主内存中（虚拟机内存的一部分）。每条线程还有自己的工作内存，线程的工作内存保存了被该线程使用到的变量的主内存副本拷贝。线程对变量的所有操作（读取、赋值等）都必须在工作内存中进行，而不能直接读写主内存中的变量。不同的线程之间也无法直接访问对方工作内存中的变量，线程间变量指的传递均需要通过主内存来完成。

局部变量与方法参数时线程私有的，不会被共享，自然就不存在竞争的问题。

这里讲的主内存、工作内存与前面讲的java内存区域划分为堆、栈和方法区是在不同层次上的内存划分。

为了获得更高的运行速度，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读写的是工作内存。

内存间交互操作

Java内存模型中定义了一下8种操作来完成主内存和虚拟内存之间的具体交互协议（即如何拷贝变量到工作内存和如何将工作内存同步到主内存）：

1. Lock
2. Unlock
3. Read
4. Load
5. Use
6. Assign
7. Store
8. write

主内存变量拷贝到工作内存：read🡪load

工作内存同步到主内存: store🡪write

一个变量在同一时刻只允许一条县城关对其执行lock操作，但lock操作可以被同一条线程重复多次，多次执行lock后，只有执行相同次数的unlock操作，变量才会被释放锁。

对于voltaile型变量的特殊规则：

最轻量级的同步机制。

1. 可见性：当一个线程修改了这个变量的值，新值对于其他线程来说是可以立即得知的

由于volatile变量只能保证可见性，在不符合以下两个规则的运算场景中，我们让然要通过加锁来保证原子性：

运算结果并不依赖变量的当前值，或者能够确保只有单一的线程修改变量的值

变量不需要与其他的状态变量共同参与不变约束

使用volatile的第二个用语义是以为了禁止指令重排序优化。

1. 原子性

Volatile 的读操作的性能与普通变量没有什么差别，但是写操作相对的要慢一些。

对long和double型变量的特殊规则：64位

非原子性协定

原子性、可见性和有序性：java内存模型是围绕着在并发过程中如何处理原子性、可见性和有序性这3个特征来建立的。

除了volatile之外，synchronized和final能够实现可见性

Java语言提供了volatile和synchronized两个关键字来保证线程之间操作的有序性。

先行发生原则：它是判断数据是否存在竞争、线程是否安全的主要依据。

Java与线程

并发不一定要依赖多线程（如PHP中很常见的多进程并发）

线程是CPU调度的基本单位

实现线程主要有三种方式：

1. 使用内核线程实现
2. 使用用户线程实现

一对多线程模型

1. 使用用户线程加轻量级进程混合实现