Java性能调优实战
刘超
金山软件西山居技术经理

查看详情 5970 人已学习

加餐 | 什么是数据的强、弱一致性?

刘超 2019-07-06

00:00 07:54
讲述：李良 大小：7.25M

你好，我是刘超。

在第 17 讲讲解并发容器的时候，我提到了“强一致性”和“弱一致性”。很多同学留言表示对这个概念没有了解或者比较模糊，今天这讲加餐就来详解一下。

说到一致性，其实在系统的很多地方都存在数据一致性的相关问题。除了在并发编程中保证共享变量数据的一致性之外，还有数据库的 ACID 中的 C（Consistency 一致性）、分布式系统的 CAP 理论中的 C（Consistency 一致性）。下面我们主要讨论的就是“并发编程中共享变量的一致性”。

在并发编程中，Java 是通过共享内存来实现共享变量操作的，所以在多线程编程中就会涉及到数据一致性的问题。

我先通过一个经典的案例来说明下多线程操作共享变量可能出现的问题，假设我们有两个线程（线程 1 和线程 2）分别执行下面的方法，x 是共享变量：

复制代码

```
// 代码 1
public class Example {
    int x = 0;
    public void count() {
        x++;                    //1
        System.out.println(x)//2
    }
}

```

线程1 调用 count	线程2 调用count
x++;	x++;

如果两个线程同时运行，两个线程的变量的值可能会出现以下三种结果：

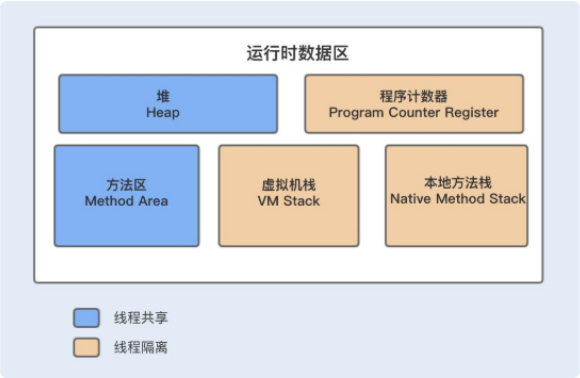
结果1	结果2	结果3
1,1	2,1	1,2

Java 存储模型

2,1 和 1,2 的结果我们很好理解，那为什么会出现以上 1,1 的结果呢?

我们知道，Java 采用共享内存模型来实现多线程之间的信息交换和数据同步。在解释为什么会出现这样的结果之前，我们先通过下图来简单了解下 Java 的内存模型（第 21 讲还会详解），程序

在运行时，局部变量将会存放在虚拟机栈中，而共享变量将会被保存在堆内存中。



由于局部变量是跟随线程的创建而创建，线程的销毁而销毁，所以存放在栈中，由上图我们可知，Java 栈数据不是所有线程共享的，所以不需要关心其数据的一致性。

共享变量存储在堆内存或方法区中，由上图可知，堆内存和方法区的数据是线程共享的。而堆内存中的共享变量在被不同线程操作时，会被加载到自己的工作内存中，也就是 CPU 中的高速缓存。

CPU 缓存可以分为一级缓存（L1）、二级缓存（L2）和三级缓存（L3），每一级缓存中所储存的全部数据都是下一级缓存的一部分。当 CPU 要读取一个缓存数据时，首先会从一级缓存中查找；如果没有找到，再从二级缓存中查找；如果还是没有找到，就从三级缓存或内存中查找。

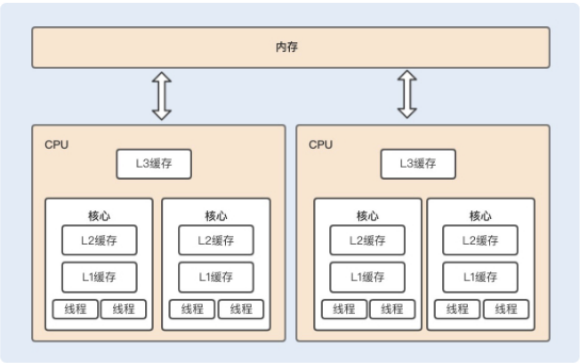
如果是单核 CPU 运行多线程，多个线程同时访问进程中的共享数据，CPU 将共享变量加载到高速缓存后，不同线程在访问缓存数据的时候，都会映射到相同的缓存位置，这样即使发生线程的切换，缓存仍然不会失效。

如果是多核 CPU 运行多线程，每个核都有一个 L1 缓存，如果多个线程运行在不同的内核上访问共享变量时，每个内核的 L1 缓存将会缓存一份共享变量。

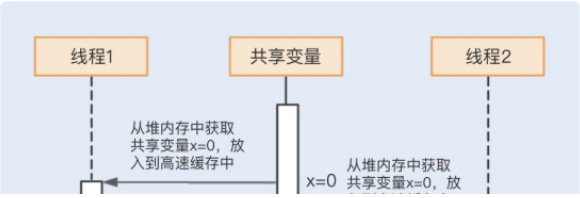
假设线程 A 操作 CPU 从堆内存中获取一个缓存数据，此时堆内存中的缓存数据值为 0，该缓存数据会被加载到 L1 缓存中，在操作后，缓存数据的值变为 1，然后刷新到堆内存中。

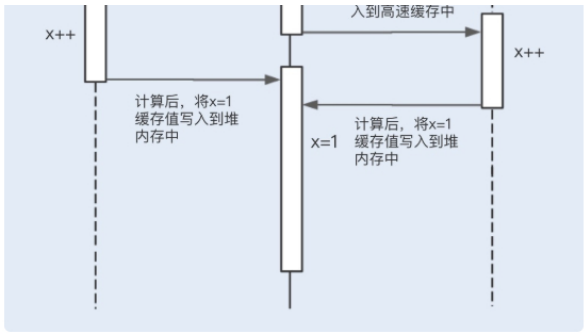
在正好刷新到堆内存中之前，又有另外一个线程 B 将堆内存中为 0 的缓存数据加载到了另外一个内核的 L1 缓存中，此时线程 A 将堆内存中的数据刷新到了 1，而线程 B 实际拿到的缓存数据的值为 0。

此时，内核缓存中的数据和堆内存中的数据就不一致了，且线程 B 在刷新缓存到堆内存中的时候也将覆盖线程 A 中修改的数据。这时就产生了数据不一致的问题。



了解完内存模型之后，结合以上解释，我们就可以回过头来看看第一段代码中的运行结果是如何产生的了。看到这里，相信你可以理解图中 1,1 的运行结果了。





重排序

除此之外，在 Java 内存模型中，还存在重排序的问题。请看以下代码：

复制代码

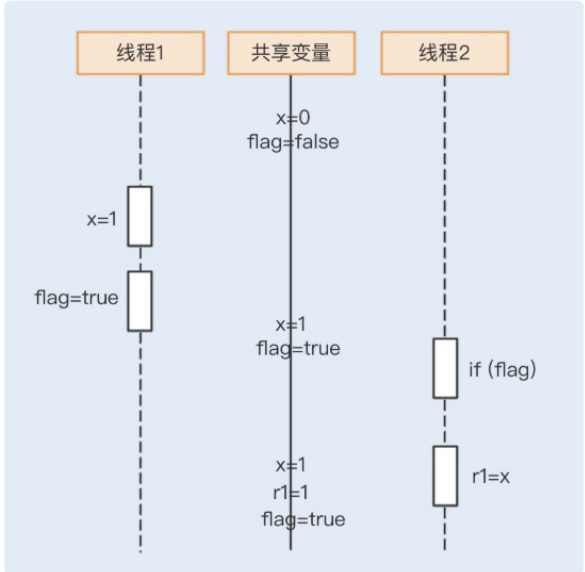
```
// 代码 1
public class Example {
    int x = 0;
    boolean flag = false;
    public void writer() {
        x = 1;                //1
        flag = true;          //2
    }

    public void reader() {
        if (flag) {           //3
            int r1 = x;       //4
            System.out.println(r1==x)
        }
    }
}

```

线程1 调用writer	线程2 调用reader
x = 1;	if (flag)
flag = true	int r1 = x;

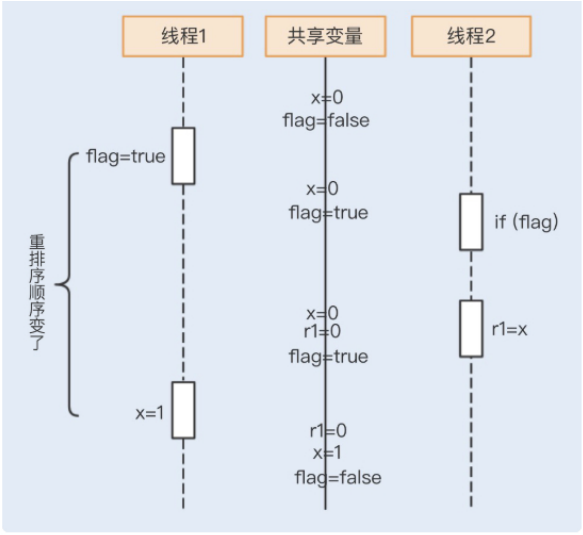
如果两个线程同时运行，线程 2 中的变量的值可能会出现以下两种可能：

结果1	结果2
r1=0	r1=1

现在一起来看看 r1=1 的运行结果，如下图所示：



那 r1=0 又是怎么获取的呢？我们再来看一个时序图：



在不影响运算结果的前提下，编译器有可能会改变顺序代码的指令执行顺序，特别是在一些可以优化的场景。

例如，在以下案例中，编译器为了尽可能地减少寄存器的读取、存储次数，会充分复用寄存器的存储值。如果没有进行重排序优化，正常的执行顺序是步骤 1\2\3，而在编译期间进行了重排序优化之后，执行的步骤有可能就变成了步骤 1/3/2 或者 2/1/3，这样就能减少一次寄存器的存取次数。

复制代码

```
int x = 1;// 步骤 1：加载 x 变量的内存地址到寄存器中，加载 1 到寄存器中，CPU 通过 mov 指令
boolean flag = true; // 步骤 2 加载 flag 变量的内存地址到寄存器中，加载 true 到寄存器中，
int y = x + 1;// 步骤 3 重新加载 x 变量的内存地址到寄存器中，加载 1 到寄存器中，CPU 通过 r

```

在 JVM 中，重排序是十分重要的一环，特别是在并发编程中。可 JVM 要是能对它们进行任意排序的话，也可能会给并发编程带来一系列的问题，其中就包括了一致性的问题。

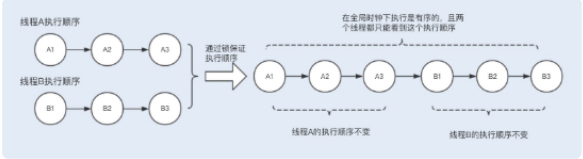
Happens-before 规则

为了解决这个问题，Java 提出了 Happens-before 规则来规范线程的执行顺序：

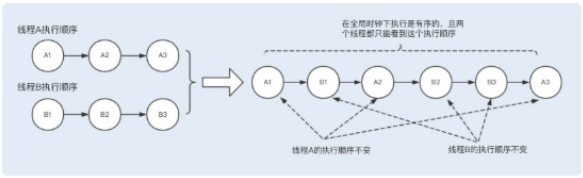
- 程序次序规则：在单线程中，代码的执行是有序的，虽然可能会存在运行指令的重排序，但最终执行的结果和顺序执行的结果是一致的；
- 锁定规则：一个锁处于被一个线程锁定占用状态，那么只有当这个线程释放锁之后，其它线程才能再次获取锁操作；
- volatile 变量规则：如果一个线程正在写 volatile 变量，其它线程读取该变量会发生在写入之后；
- 线程启动规则：Thread 对象的 start() 方法先行发生于此线程的其它每一个动作；
- 线程终结规则：线程中的所有操作都先行发生于对此线程的终止检测；
- 对象终结规则：一个对象的初始化完成先行发生于它的 finalize() 方法的开始；
- 传递性：如果操作 A happens-before 操作 B，操作 B happens-before 操作 C，那么操作 A happens-before 操作 C；
- 线程中断规则：对线程 interrupt() 方法的调用先行发生于被中断线程的代码检测到中断事件的发生。

结合这些规则，我们可以将一致性分为以下几个级别：

严格一致性（强一致性）：所有的读写操作都按照全局时钟下的顺序执行，且任何时刻线程读取到的缓存数据都是一样的，Hashtable 就是严格一致性；



顺序一致性：多个线程的整体执行可能是无序的，但对于单个线程而言执行是有序的，要保证任何一次读都能读到最近一次写入的数据，volatile 可以阻止指令重排序，所以修饰的变量的程序属于顺序一致性；



弱一致性：不能保证任何一次读都能读到最近一次写入的数据，但能保证最终可以读到写入的数据，单个写锁 + 无锁读，就是弱一致性的一种实现。

今天的加餐到这里就结束了，如有疑问，欢迎留言给我。也欢迎你点击“请朋友读”，把今天的内容分享给身边的朋友，邀请他一起学习。

胡峣

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Command + Enter 发表　　0/2000字　　提交留言

精选留言(13)

Liam

老师好，请教一个问题：

文中举例，数据不一致是多核CPU的高速缓存不一致导致的，是否意味着单核CPU多线程操作就不会发生数据不一致呢

作者回复: 也会的，线程安全除了要保证可见性，还需要保证原子性、有序性。

2019-07-06　　4

Lost In The Echo。

老师，请问强一致性和顺序一致性有什么区别吗?

作者回复: 顺序一致性是指单个线程的执行的顺序性，强一致性则指的是多个线程在全局时钟下的执行的顺序性。

2019-07-06 3

东方奇骥

上面例子，flag加volatile修饰，根据happens before中的顺序性选择和volatile的原则，就能保证另一个线程读到写入的值了。

作者回复: 对的，volatile除了可以保证变量的可见性，可以阻止局部指令重排序。

2019-07-06 1

青梅煮酒

老师，请问一下，每核CPU都有自己的L1和L2，那么L1和L2的主要区别是什么呢？为什么不能合到一起呢？

作者回复: L1\L2\L3三个缓存的作用和实现的技术是不一样的，L1的内存大小是非常有限的，所以很多时候在L1获取缓存数据的命中率非常低。为了提高CPU读取的速率，在L1没有命中的缓存，可以进入到L2进行获取，L2的容量要比L1大，但离CPU核心更远。但还是能提高CPU读取缓存数据的速率。

2019-07-16

明翼

早看到就好了😬，老师请教下这么多知识点你是怎么记住的？

作者回复: 平时善于做笔记，除此之外，尝试将自己学到的知识点分享给其他人。

2019-07-10

面朝大海

int x = 1;// 步骤 1：加载 x 变量的内存地址到寄存器中，加载 1 到寄存器中，CPU 通过 mov 指令把 1 写入到寄存器指定的内存中
boolean flag = true; // 步骤 2 加载 flag 变量的内存地址到寄存器中，加载 true 到寄存器中，CPU 通过 mov 指令把 1 写入到寄存器指定的内存中
int y = x + 1;// 步骤 3 重新加载 a 变量的内存地址到寄存器中，加载 1 到寄存器中，CPU 通过 mov 指令把 1 写入到寄存器指定的内存中

2019-07-08

TWO STRINGS

老师您好，都说concurrenthashmap的get是弱一致性，但我不理解啊，volatile 修饰的变量读操作为什么会读不到最新的数据?

作者回复: 我们知道Node<k,v>以及Node<k,v>的value是volatile修饰的，所以在一个线程对其进行修改后，另一个线程可以马上看到。
如果是一个新Node，那么就不能马上看到，虽然Node的数组table被volatile修饰，但是这样只是代表table的引用地址如果被修改，其他线程可以立马看到，并不代表table里的数据被修改立马可以看到。

2019-07-08

-W.LI-

老师好volatile+cas是强一致性么?。L1直接刷回主存，L2和L3需要做什么操作么?开头说每一级都是上一级的子集来着。

作者回复: cas+volatile可以解决单个变量的强一致性问题。

2019-07-07

云封

老师，请问下，如果不存在操作共享变量的情况或者把共享产量存在redis中，多线程结果就不会发生由于指令重排而导致结果不一致的情况。

作者回复: 指令重排序不一定是由于共享变量导致的，这块需要结合具体的场景分析。

2019-07-07

Jxin

请问老师，指令重排优化会受多线程影响吗？感觉应该不会出现赋值为true和x=1这两条指令对换位置。因为从单线程来看这没有指令重排的价值，所以感觉不会做重排优化。而如果重排优化会受多线程影响，那么场景1的r1==1应该是赋值为true，然后进入了if逻辑，接着优先执行x=1才导致的r1==1的结果。布尔赋值为true和if判断应该要紧挨着，减少一次寄存器加载该临时变量值。也就是老师那个场景1不会出现。

作者回复: 这里只是假设，有专门一个指令重排序的例子。

2019-07-06

-W.LI-

老师容我问一个很基础的问题!父类private的属性会被子类继承么?子类创建的时候JVM给子类分配内存的时候，我看书上有说父类的属性会排在子类前面有可能穿插。可是没写是否会给子类分配父类的私有属性内存空间。子类创建的时候，会默认调用父类的无参构造器。这时候就会实例化一个父类对象么?(如果父类没有无参构造器会报错或者需要显式调用父类的有参构造器)。如果每次实力子类对象的时候都会先创建一个父类对象的话，滥用继承。就会浪费很多内存是么?对象头就需要8字节了。

2019-07-06

密码123456

单核也会有问题的，还有重排序。

作者回复: 会有重排序问题

2019-07-06

nightmare

点赞666

2019-07-06