valotile关键字

一、java的内存模型

1、内存模型简介

（1）不同的平台，内存模型是不一样的，但是jvm的内存模型规范是统一的

（2）线程安全无非是要控制多个线程对某个资源的有序访问或修改。总结java的内存模型，要解决两个主要的问题：可见性和有序性。

（3）我们都知道计算机有高速缓存的存在，处理器并不是每次处理数据都是取内存的。JVM定义了自己的内存模型，屏蔽了底层平台内存管理细节

2、可见性

前提：Java内存模型（JMM）规定了jvm有主内存，主内存是多个线程共享的。当new一个对象的时候，也是被分配在主内存中，每个线程都有自己的工作内存，工作内存存储了主存的某些对象的副本。

当线程操作某个对象时，执行顺序如下：

(1) 从主存复制变量到当前工作内存 (read and load)

(2) 执行代码，改变共享变量值 (use and assign)

(3) 用工作内存数据刷新主存相关内容 (store and write)

synchronize和valotile

（1）synchronize

一个线程执行临界区代码过程如下：

1 获得同步锁

2 清空工作内存

3 从主存拷贝变量副本到工作内存

4 对这些变量计算

5 将变量从工作内存写回到主存

6 释放锁

可见，synchronized既保证了多线程的并发有序性，又保证了多线程的内存可见性。

（2）valotile

是轻量级的同步，为什么这么说，因为volatile只能保证多线程的内存可见性，不能保证多线程的执行有序性。任何被volatile修饰的变量，都不拷贝副本到工作内存，任何修改都及时写在主存。因此对于Valatile修饰的变量的修改，所有线程马上就能看到。

二、Volatile的官方定义

Java语言规范第三版中对volatile的定义如下： java编程语言允许线程访问共享变量，为了确保共享变量能被准确和一致的更新，线程应该确保通过排他锁单独获得这个变量。

Java语言提供了volatile，在某些情况下比锁更加方便。如果一个字段被声明成volatile，java线程内存模型确保所有线程看到这个变量的值是一致的。

三、volatile的原理和实现机制

　　前面讲述了源于volatile关键字的一些使用，下面我们来探讨一下volatile到底如何保证可见性和禁止指令重排序的。

　　下面这段话摘自《深入理解Java虚拟机》：

　　“观察加入volatile关键字和没有加入volatile关键字时所生成的汇编代码发现，加入volatile关键字时，会多出一个lock前缀指令”

　　lock前缀指令实际上相当于一个内存屏障（也成内存栅栏），内存屏障会提供3个功能：

　　1）它确保指令重排序时不会把其后面的指令排到内存屏障之前的位置，也不会把前面的指令排到内存屏障的后面；即在执行到内存屏障这句指令时，在它前面的操作已经全部完成；

　　2）它会强制将对缓存的修改操作立即写入主存；

　　3）如果是写操作，它会导致其他CPU中对应的缓存行无效。

四、下面就来具体介绍下happens-before原则（先行发生原则）：

•程序次序规则：一个线程内，按照代码顺序，书写在前面的操作先行发生于书写在后面的操作

•锁定规则：一个unLock操作先行发生于后面对同一个锁额lock操作

•volatile变量规则：对一个变量的写操作先行发生于后面对这个变量的读操作

•传递规则：如果操作A先行发生于操作B，而操作B又先行发生于操作C，则可以得出操作A先行发生于操作C

•线程启动规则：Thread对象的start()方法先行发生于此线程的每个一个动作

•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

•线程终结规则：线程中所有的操作都先行发生于线程的终止检测，我们可以通过Thread.join()方法结束、Thread.isAlive()的返回值手段检测到线程已经终止执行

•对象终结规则：一个对象的初始化完成先行发生于他的finalize()方法的开始

四、volatile 变量的条件

您只能在有限的一些情形下使用 volatile 变量替代锁。要使 volatile 变量提供理想的线程安全，必须同时满足下面两个条件：

对变量的写操作不依赖于当前值。

该变量没有包含在具有其他变量的不变式中。

五、使用场景

1.volatile最适用一个线程写，多个线程读的场合。

如果有多个线程并发写操作，仍然需要使用锁或者线程安全的容器或者原子变量来代替。(摘自Netty权威指南)

模式 #1：状态标志

模式 #2：一次性安全发布（单例）

private volatile static Singleton instace;

public static Singleton getInstance(){

//第一次null检查

if(instance == null){

synchronized(Singleton.class) { //1

//第二次null检查

if(instance == null){ //2

instance = new Singleton();//3

}

}

}

return instance;

}

模式 #3：独立观察（independent observation）

安全使用 volatile 的另一种简单模式是：定期 “发布” 观察结果供程序内部使用。【例如】假设有一种环境传感器能够感觉环境温度。一个后台线程可能会每隔几秒读取一次该传感器，并更新包含当前文档的 volatile 变量。然后，其他线程可以读取这个变量，从而随时能够看到最新的温度值。

使用该模式的另一种应用程序就是收集程序的统计信息。【例】如下代码展示了身份验证机制如何记忆最近一次登录的用户的名字。将反复使用lastUser 引用来发布值，以供程序的其他部分使用。

public class UserManager {

public volatile String lastUser; //发布的信息

public boolean authenticate(String user, String password) {

boolean valid = passwordIsValid(user, password);

if (valid) {

User u = new User();

activeUsers.add(u);

lastUser = user;

}

return valid;

}

}

模式 #4：“volatile bean” 模式

volatile bean 模式的基本原理是：很多框架为易变数据的持有者（例如 HttpSession）提供了容器，但是放入这些容器中的对象必须是线程安全的。

在 volatile bean 模式中，JavaBean 的所有数据成员都是 volatile 类型的，并且 getter 和 setter 方法必须非常普通——即不包含约束！

@ThreadSafe

public class Person {

private volatile String firstName;

private volatile String lastName;

private volatile int age;

public String getFirstName() { return firstName; }

public String getLastName() { return lastName; }

public int getAge() { return age; }

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public void setAge(int age) {

this.age = age;

}

}

模式 #5：开销较低的“读－写锁”策略

如果读操作远远超过写操作，您可以结合使用内部锁和 volatile 变量来减少公共代码路径的开销。

如下显示的线程安全的计数器，使用 synchronized 确保增量操作是原子的，并使用 volatile 保证当前结果的可见性。如果更新不频繁的话，该方法可实现更好的性能，因为读路径的开销仅仅涉及 volatile 读操作，这通常要优于一个无竞争的锁获取的开销。

@ThreadSafe

public class CheesyCounter {

// Employs the cheap read-write lock trick

// All mutative operations MUST be done with the 'this' lock held

@GuardedBy("this") private volatile int value;

//读操作，没有synchronized，提高性能

public int getValue() {

return value;

}

//写操作，必须synchronized。因为x++不是原子操作

public synchronized int increment() {

return value++;

}

}

使用锁进行所有变化的操作，使用 volatile 进行只读操作。

其中，锁一次只允许一个线程访问值，volatile 允许多个线程执行读操作

六、**volatile的优势**

1）与锁相比，volatile变量是一和更轻量级的同步机制，因为在使用这些变量时不会发生上下文切换和线程调度等操作

2）volatile变量也存在一些局限：不能用于构建原子的复合操作，因此当一个变量依赖旧值时就不能使用volatile变量