转 从Java内存模型的角度思考线程安全与并发

2017年03月23日 15:08:29 阅读数：1793 更多

并发的两个关键问题

1、线程之间如何通信

2、线程之间如何同步

通信是指线程之间以何种机制来交换信息，在命令式编程中，通信机制有两种：共享内存和消息传递；JAVA的并发采用的是共享内存，线程之间的通信总是隐式进行。

同步指程序中用于控制不同线程间操作发生相对顺序的机制，在共享内存并发模型中，同步是显式进行的。

JAVA的内存模型

1、共享变量：分配在堆内存中的元素都是共享变量，包括实例域、静态域、数组元素。

2、非共享变量：分配在栈上的都是非共享变量，主要指的是局部变量。该变量为线程私有，不会在线程之间共享，也不存在内存可见性的问题。

1、图中的主内存用于存储共享变量，主内存是所有线程所共有的。

2、本地内存是一个抽象概念，不像主内存是真实存在的，每一个线程都有一个本地内存，用于存放该线程所使用的共享变量的副本。

如果线程A和线程B需要进行通信，则必须经过以下两个过程：

1、线程A把本次内存中修改过的共享变量刷新到主内存中。

2、线程B到主内存中去读取线程A已经更新过的共享变量

这个过程与计算机网络的7层模型的过程特性类似，都必须先经过从上到下，再经过底层的物理链路，最后从下到上完成一次通信。

一、内存间交互操作

内存间交互主要指工作内存（本地内存）与主内存之间的交互，即一个变量如何从主内存拷贝到工作内存，如何从工作内存刷新到主内存的一些实现细节。J 存模型定义以下八种操作来完成：

1、lock：作用于主内存，把一个变量标识为某个线程独占状态。

2、unlock：作用于主内存，把一个处于锁定状态的变量释放，释放后变量可以被其他线程锁定。

3、read：作用于主内存，把一个变量从主内存传输到工作内存中，用于后面的load操作。

4、load：作用于工作内存，把read操作从主内存中得到的变量值放入工作内存的变量副本中。

5、use：作用于工作内存，把变量值传递给执行引擎，每当虚拟机需要使用变量的字节码指令时将会执行这个操作。

6、assign：作用于工作内存，把从执行引擎接收到的值赋值给工作内存的变量，每当虚拟机遇到需要给该变量赋值的字节码指令时执行这个操作。

7、store：作用于工作内存，把工作内存中的一个变量值传到的主内存，以便后续的write操作。

8、write：作用于主内存，把store操作从工作内存中获取的值赋值给主内存中的变量

对于这8个操作，有如下的一个原则：

1、不允许read和load，store和write操作单独出现。

2、不允许一个线程丢弃它最近的assign操作，即变量在工作内存中的更新需要同步到主内存中。

3、不允许线程无原因地（没有发生过任何assign操作）把数据同步到主内存。

4、一个新的变量只能在主内存中产生，不能在工作内存中直接使用未被初始化的变量。

5、一个变量在同一时刻只能被一个线程lock，并且lock和unlock需要成对出现。

6、如果对一个变量执行lock操作，将会清空工作内存中此变量的值，在执行引擎使用这个变量前需要执行load或者assgin操作。

7、对一个变量执行unclock之前，必须把此变量同步到主内存中。

二、重排序

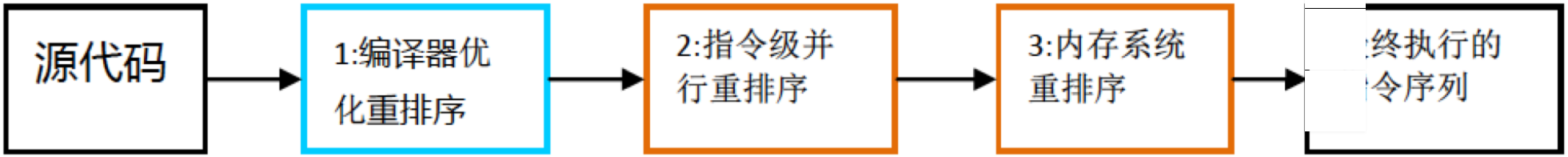
重排序是指编译器和处理器为了优化程序性能而对指令序列进行重新排序的一种手段。重排序分为3类：

1、编译器优化重排序：编译器在不改变单线程语义的前提下，可以重新安排语句的执行顺序。

2、指令级并行的重排序：如果不存在数据依赖性，处理器可以改变语句对应机器指令的执行顺序。

3、内存系统重排序：由于处理器使用缓存和缓冲区，使得加载和存储操作看上去可能是乱序执行。

从Java源代码到最终实际执行的指令序列，会经过下面三种重排序：



数据依赖性：

```
a = 1;      //1
a = 2;     //2
```

上面展示的是写后写的操作，其中1、2两步的顺序打乱，会改变程序执行的结果；这种就是数据有依赖性性。

```
double pi = 3.14;              //A
double r = 1.0;                //B
double area = pi * r * r;      //C
```

这里可以看到，A和C、B和C有数据依赖，其中A和B没有数据依赖，这样编译器和处理器就可以对AB进行重排序。

上面AB这种场景就是符合as-if-serial语义的：不管怎么进行重排序，单线程程序的执行结果不能被改变。编译器和处理器在重排序的时候都必须遵守as-if-serial语义。

三、内存屏障

内存屏障又称内存栅栏，是一种CPU指令，用于控制特定条件下的重排序和内存可见性问题。由于现在操作系统都是多处理器，而每一个处理器都有自己的缓存，并且这些缓存都不是实时与内存进行交互。这样就会导致不同CPU上缓存的数据不一致问题，在多线程的程序中，就会出现一些异常行为。而操作系统底层就提供了内存屏障来解决这些问题。目

前有4种屏障：

1、LoadLoad屏障：

对于语句Load1；LoadLoad；Load2，在Load2及后续读取操作要读取的数据被访问前，保证Load1要读取的数据被读取完毕。

2、StoreStore屏障：

对于语句Store1；StoreStore；Store2，在Store2及后续写入操作之前，保证Store1的写入操作对其他处理器可见。

3、LoadStore屏障：

对于语句Load1；LoadStore；Store2，在Store2及后续写入操作之前，保证Load1要读取的数据被读取完毕。

4、StoreLoad屏障：

对于语句Store1；StoreLoad；Load2，在Load2及后续所有读取操作执行前，保证Store1的写入对所有处理器可见。

java对内存屏障的使用有以下常见的两种：

1、使用volatile修饰变量，则对变量的写操作，会插入StoreLoad屏障。

2、使用Synchronized关键字包住的代码区域，当线程进入到该区域读取变量的信息时，保证读到的是最新的值，这是因为在同步区内对变量的写入操作，在离开同步区时将当前线程内数据刷新到主内存中。而对数据的读取也不能从缓存中获取，只能从主内存中读取，保证了数据的有效性。这就是插入了StoreStore屏障。

四、volatile内存语义与实现

关键字volatile是java虚拟机提供的轻量级的同步机制，只能用来修改变量，在多线程情况下保证了变量的可见性，但是不保证变量的原子性。volatitle修饰的变量具有以下两种特性：

1、可见性

保证此变量对所有线程的可见性，这里的可见性是指当一个线程修改了该变量的值，修改后的值对其他所有线程都是立即可知的。而普通变量则做不到这一点，因为普通变量需要将修改的值从工作内存同步到主内存后才能被其他线程可见。

volatile变量也可以在各个线程的工作内存中存在数据不一致的情况，但是由于每次使用之前都要进行刷新，执行引擎看不到不一致的情况，因此可以认为不存在不一致的问题。

前面提高volatile不能保证原子性，下面看这段代码：

复制代码

```
public class VolatileTest
{
    private static volatile int rac = 0;

    private static final int threadCnt = 20;

    public static void increase()
    {
        rac++;
    }

    public static void main(String[] args)
    {
        Thread[] thread = new Thread[threadCnt];
        for(int i=0; i<threadCnt; i++)
        {
            thread[i] = new Thread(new Runnable()
            {

                @Override
                public void run()
                {
                    for(int j=0; j < 10000; j++)
                    {
                        increase();
                    }

              }});
            thread[i].start();
        }

        while(Thread.activeCount() > 1)
        {
```

```
            Thread.yield();
        }

        System.out.println(rac);
    }
}
```

复制代码

这段代码如果是正确并发的话，得到的结果应该是200000，但是我们得到的结果都是比这个值要小，且每次的结果都不一样。（这里每条线程的自增操作次数要足够大，10000就可以，因为如果太小，20个线程就会顺序执行没有并发，得到的是正确的结果，不会出现我们要制造的那种场景）。

我们利用下面的命令，得到increase方法的字节码如下：

可以看到在自增运算rac++是由四条字节码指令构成，这里就可以知道为什么会出现异常的原因了：第一步中getstatic指令将rac的值取到操作栈顶时，volatile保证了rac的值在此时是正确的，但是在执行第二、三步的时候，其他线程可能已经将rac的值增加了，那么在栈顶的rac的值就是过期的数据，在最后调用putstatic指令将rac同步到主内存中时，rac的值就偏小；

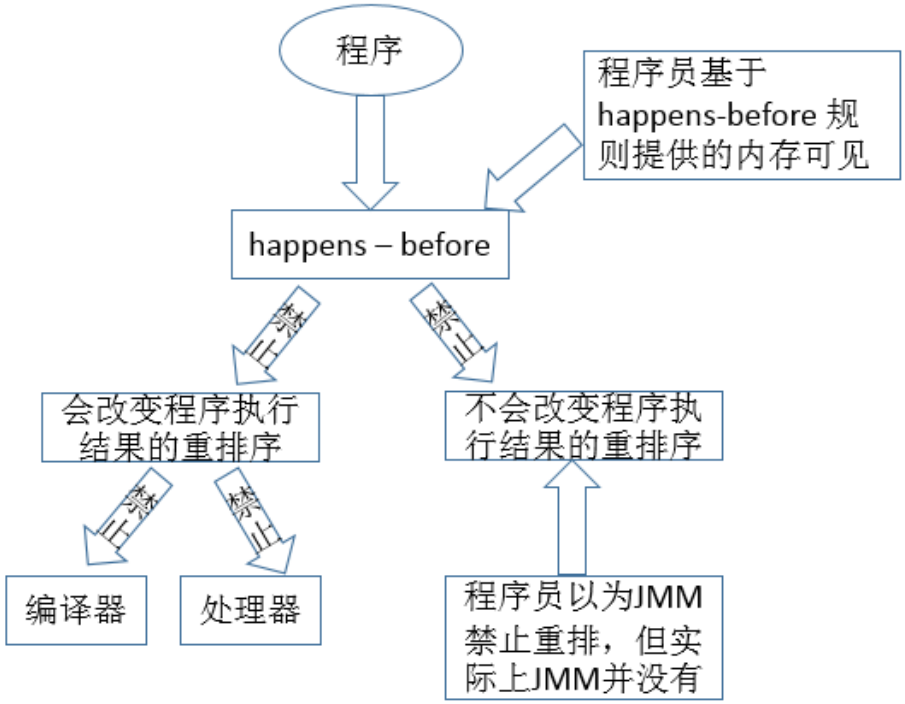
2、禁止指令重排序优化

五、happens-before

JMM对两种不同性质的重排序，采取了不同的策略：

1、对于会改变程序执行结果的重排序，JMM要求编译器和处理器必须禁止这种重排序。

2、对于不会改变程序执行结果的重排序，JMM对编译器和处理器不作要求（JMM允许这种重排序）



happens-before规则：

1、一个线程中的每个操作happens-before于该线程中的任意后续操作。

2、对一个锁的解锁happens-before与随后对这个锁的加锁。

3、对一个volatile域的写happens-before于任意后续对这个volatile域的读。

4、如果A happens-before B，且B happens-before C， 那么A happens-before C

5、如果线程A执行操作ThreadB.start()(启动B线程)，那么A线程的ThreadB.start()操作happens-before于线程B中的任意操作。

6、如果线程A执行操作ThreadB.join()并且成功返回，那么线程B中的任意操作happens-before于线程A从ThreadB.join()操作成功返回。

参考：http://www.cnblogs.com/dongguacai/p/5970076.html

线程同步(1):原子操作,内存屏障,锁综述


6651

原子操作,内存屏障,锁 1.原理：CPU提供了原子操作、关中断、锁内存总线,内存屏障等机制；OS基于这几个CPU硬件机制，就能够…

Java多线程之内存可见性和原子性：Synchronized和Volatile的比较


382

在刷题时，碰到一题：关于volatile关键字的说法错误的是： A. 能保证线程安全 B volatile关键字用在多线程同步中，可保证读取的可见…

从 JVM 内存模型谈线程安全


587

作为一个三个多月没有去工作的独立开发者而言,今天去小米面试了一把.怎么说呢,无论你水平如何,请确保在面试之前要做准备,就像其…

java线程安全总结


491

最近想将java基础的一些东西都整理整理，写下来，这是对知识的总结，也是一种乐趣。已经拟好了提纲，大概分为这几个主题： java…

相关热词 javaⅱ 与java java的～ java java和--

0

个人资料


小飞鹤 关注

原创	粉丝	喜欢	评论
422	746	297	196

等级： 博客 访问： 157万+

积分： 1万+ 排名： 693

勋章：

0

最新文章

AQS与JUC中的锁实现原理

JDK1.8源码分析之AbstractQueuedSynchronizer

基于Docker实现MySQL的主从复制和SpringBoot2+MyBatis的动态切换数据源的读写分离

LinkedBlockingQueue原理分析---基于JDK8

MongoDB数据导入与导出

博主专栏

Java高级学习

阅读量：72663 46 篇

Spring全栈高级技术

阅读量：0 0 篇

Java技术栈架构

阅读量：173929 28 篇

JVM和Java源码原理分析

阅读量：0 0 篇

Spring Boot 与 微服务实践

阅读量：100967 14 篇

个人分类

Maven	6篇
Mybatis	8篇
Android开发详解	132篇
JavaEE	54篇
感悟	32篇

展开

归档

2018年8月	2篇
2018年5月	1篇
2018年1月	2篇
2017年11月	3篇
2017年9月	8篇

展开

热门文章

消息队列的使用场景
阅读量：48348

Spring整合Shiro做权限控制模块详细案例分析
阅读量：37970

使用QRCode.jar生成和解析二维码(Maven版)
阅读量：33035

0

Spring MVC中使用 Swagger2 构建Restful API
阅读量：28278

Java网络编程详解
阅读量：27688

最新评论

Dubbo原理简单分析
tr1912：博主总结的很好，学习了

微服务的一种开源实现方式——dub...
ldb987：很詳細，感謝博主分享，。

开发技术选型参考
JYL15732624861：感谢分享

使用QRCode.jar生成和解析...
baidu_34036884：你好，请问如果公司没有自己的maven仓库的话，可以用这种公共仓库没有的jar包吗?

Spring整合Shiro做权限控...
qq_24607837：[reply]yjsl__[/reply] 15年的提问都没有回复，估计这货都不干这一行了

联系我们

请扫描二维码联系客服

webmaster@csdn.net

400-660-0108

QQ客服 客服论坛