

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 完成:

1、lock:作用于主内存,扩

2、unlock:作用于主内存

3、read:作用于主内存,

4、load:作用于工作内存,

5、use: 作用于工作内存,

6、assign:作用于工作内容

o、assign:作用于工作内1

联系我们



请扫描二维码联系客服

webmaster@csdn.net

2400-660-0108

▲ QQ客服 ● 客服论坛

关于 招聘 广告服务 网站地图 ©2018 CSDN版权所有京ICP证09002463号

☆ 百度提供搜索支持

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

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

「以被其他线程锁定。

i的load操作。

存的变量副本中。

变量的字节码指令时将会执行这个操作。

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,地工作内存中的一个发重值传到的土内存,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



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。

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; Loa

2、StoreStore屏障:

对于语句Store1; St

3、LoadStore屏障:

对于语句Load1; Loa

4、StoreLoad屏障:

对于语句Store1; St

联系我们



请扫描二维码联系客服

webmaster@csdn.net

2400-660-0108

● QQ客服 ● 客服论坛

关于 招聘 广告服务 网站地图 ©2018 CSDN版权所有京ICP证09002463号

☆ 百度提供搜索支持

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

读取的数据被访问前,保证Load1要读取的数据被读取完毕。

0

作之前,保证Store1的写入操作对其他处理器可见。

之前,保证Load1要读取的数据被读取完毕。

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

联系我们

复制代码

```
public class Volati
{
    private static

    private static

    public static v
    {
        rac++;
    }

    public static v
```

```
请扫描二维码联系客服

■ webmaster@csdn.net

2400-660-0108
```

关于 招聘 广告服务 网站地图 ©2018 CSDN版权所有京ICP证09002463号 常 百度提供搜索支持

■ QQ客服 ● 客服论坛

经营性网站备案信息 网络110报警服务 中国互联网举报中心 北京互联网违法和不良信息举报中心

```
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();
}</pre>
```

while(Thread.activeCount() > 1)

0

这段代码如果是正确并 以,因为如果太小,20个线

我们利用下面的命令,得到



导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i 欠数要足够大,10000就可不会出现我们要制造的那种场景)。

0

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)

0



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

线程一起执行的时候,一个操作一旦开始,就不会被其他...

线程都有一个私有的本地内存(local memory), 本地内...

Python正确的学习路

如何从8K提至20K月薪, 你



想对作者说点什么



Java内存模型和线程

1, 原子性: 原子性是技

Java 多线程1:多线科

Java 内存模型 线程之间的块

联系我们



请扫描二维码联系客服

webmaster@csdn.net

2400-660-0108

■ QQ客服 ● 客服论坛

广告服务 网站地图 ©2018 CSDN版权所有 京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

从面试题i = i++; 了解

☆ 百度提供搜索支持

北京互联网违法和不良信息举报中心

686

⊚ 383

0

先问大家一个问题: int i = 0; i = i ++; System.out.print(i);// 结果为 0 int i = 0; int j = i +...

JAVA并发编程2 线程安全&内存模型



2758

"你永远都不知道一个线程何时在运行!"在上一篇博客JAVA并发编程1多线程的实现方式中后面看到多线程中程序运行结果往往不确...

乍解析)

根据java内存模型理解并发出现的问题



339

原子性1、某些读写共享变量的操作如果不是原子操作,多线程并发的情况下会出现并发问题。 2、原子性实现了多个线程并发访问某...

Java知识点总结篇: Java的内存模型、线程安全、进程和线程的区别



第一部分: 进程和程序的区别 程序只是一组指令的有序集合,它本身没有任何运行的含义,它只是一个静态的实体。而进程则不同,...

纯母语 弘敦 · 1241·

孩子在这里学习英语效果真的好!

为什么在这家机构学习英语的娃都这么厉害?

线程同步(1):原子操作

原子操作,内存屏障,锁 1.原

Java多线程之内存可

在刷题时,碰到一题:关于

从 JVM 内存模型谈线

作为一个三个多月没有去工

java线程安全总结

最近想将java基础的一些东京

相关热词

javall 与java

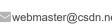
联系我们



请扫描二维码联系客服

- webmaster@csdn.net
- **2**400-660-0108

关于 招聘 广告服务 网站地图 ©2018 CSDN版权所有 京ICP证09002463号 ☆ 百度提供搜索支持



■ QQ客服 ● 客服论坛

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

内存屏障等机制; OS基于这几个CPU硬件机制, 就能够...

ile的比较

⊚ 382

0

全 B volatile关键字用在多线程同步中,可保证读取的可见...

说呢,无论你水平如何,请确保在面试之前要做准备,就像其...

491

上一种乐趣。已经拟好了提纲,大概分为这几个主题: java...

个人资料



关注

原创 422

粉丝

746

喜欢 297

评论 196

等级:

博客 つ

访问: 157万+

积分: 1万+

排名: 693

勋章: 睻 📵



https://blog.csdn.net/he90227/article/details/65443948

联系我们



请扫描二维码联系客服

webmaster@csdn.net

2400-660-0108

● QQ客服 ● 客服论坛

关于 招聘 广告服务 网站地图 ©2018 CSDN版权所有京ICP证09002463号 ☆ 百度提供搜索支持

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

基于Docker实现MySQL的主

AQS与JUC中的锁实现原理

JDK1.8源码分析之AbstractQ

gBoot2+MyBatis的动态切换数据源的读写 分离

LinkedBlockingQueue原理分析---基于JDK 8

MongoDB数据导入与导出

博主专栏

最新文章

onizer



Java高级学习

阅读量: 72663 46 篇

S_l

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

0

https://blog.csdn.net/he90227/article/details/65443948

ET (DEVAN)

阅读量: 0

0 篇



Spring Boot 与微

阅读量: 100972

个人分类

Maven

Mybatis

Android开发详解

JavaEE

咸悟

展开

联系我们



请扫描二维码联系客服

webmaster@csdn.net

2400-660-0108

▲ QQ客服 ● 客服论坛

关于 招聘 广告服务 网站地图 ©2018 CSDN版权所有京ICP证09002463号 ☆ 百度提供搜索支持

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

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日

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

https://blog.csdn.net/he90227/article/details/65443948

0

Spring MVC中使用 Swagger.

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—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,请问如果公司没有目己的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

2400-660-0108

▲ QQ客服 ● 客服论坛

关于 招聘 广告服务 网站地图 ©2018 CSDN版权所有京ICP证09002463号 ☆ 百度提供搜索支持

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

0