大家好，很高兴我们可以继续学习交流Java高频面试题。在上一小节中，我们介绍了一些多线程并发编程的基础高频考察知识点，本小节，我们继续来交流学习多线程的相关知识点，**主要包括原子性，可见性，有序性；常用的同步锁synchronized关键字，轻量级锁volatile关键字以及显式锁ReentrantLock等。**

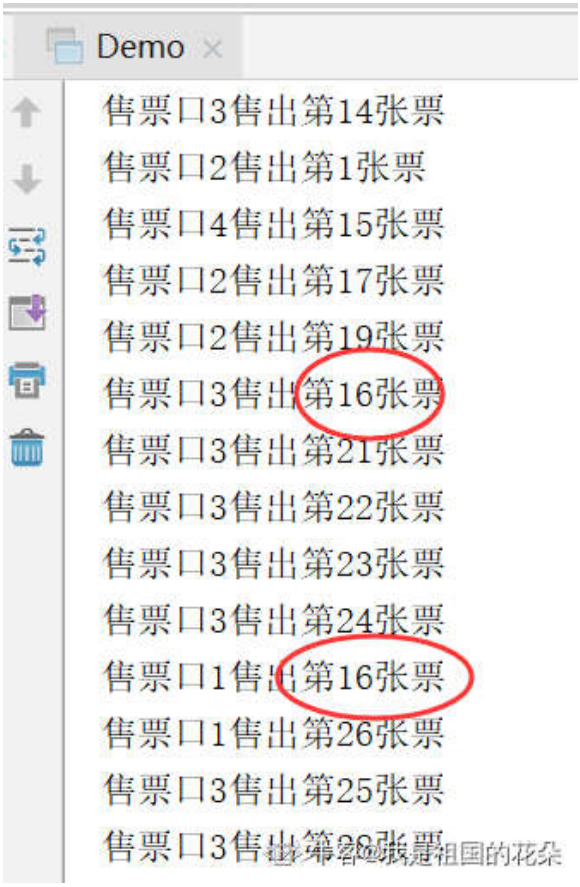
我们知道，多线程并发利用了**CPU轮询时间片**的特点，在一个线程进入阻塞状态时，可以快速切换到其余线程执行其余操作。CPU轮询时间片有利于提高其资源的利用率，最大限度的利用系统提供的处理能力，有效减少了用户的等待响应时间。但是**多线程并发编程也存在着线程活性故障以及如何保证线程安全的问题。**

在上一小节中，我们阐述了何为线程活性故障。本小节中，主要对线程安全相关知识点进行阐述。我们先来看一个线程安全的经典问题：**多个窗口售票问题。**

Demo展示如下：

```
package niuke.thread;

public class Demo {

    public static void main(String[] args) {

        TicketSale ticketSale = new TicketSale();
        Thread Sale1 = new Thread(ticketSale, "售票口1");
        Thread Sale2 = new Thread(ticketSale, "售票口2");
        Thread Sale3 = new Thread(ticketSale, "售票口3");
        Thread Sale4 = new Thread(ticketSale, "售票口4");
        // 启动线程，开始售票
        Sale1.start();
        Sale2.start();
        Sale3.start();
        Sale4.start();
    }
}

class TicketSale implements Runnable {
    int ticketSum = 100;

    @Override
    public void run() {
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        // 有余票，就卖
        while (ticketSum > 0) {
            System.out.println(Thread.currentThread().getName() + "售出第" + (100 - ticketSu
            ticketSum--;
        }
        System.out.println(Thread.currentThread().getName() + "表示没有票了");
    }
}
```

程序输出部分截图如下：



14

由图中可以看出，在多线程并发情况下，出现了同一张票被多个窗口卖出的情况，也就是出现了线程安全的问题。多线程环境下的线程安全主要体现在**原子性，可见性与有序性**方面。接下来，我们依次介绍三大特性。

（1）原子性，可见性与有序性：

答：多线程环境下的线程安全主要体现在**原子性，可见性与有序性**方面。

原子性：

定义：对于涉及到共享变量访问的操作，若该操作从执行线程以外的任意线程来看是不可分割的，那么该操作就是原子操作，该操作具有原子性。即，其它线程不会“看到”该操作执行了部分的中间结果。

举例：银行转账流程中，A账户减少了100元，那么B账户就会多100元，这两个动作是一个原子操作。我们不会看到A减少了100元，但是B余额保持不变的中间结果。

原子性的实现方式：

- 利用**锁的排他性**，保证同一时刻只有一个线程在操作一个共享变量
- 利用**CAS（Compare And Swap）**保证
- Java**语言规范中**，保证了除long和double型以外的任何变量的写操作都是原子操作
- Java**语言规范**中又规定，volatile关键字修饰的变量可以保证其写操作的原子性

关于原子性，你应该注意的地方：

- 原子性针对的是多个线程的共享变量，所以对于局部变量来说不存在共享问题，也就无所谓是否是原子操作
- 单线程环境下讨论是否是原子操作没有意义

- volatile关键字仅仅能保证变量写操作的原子性，不保证复合操作，比如说读写操作的原子性

可见性：

定义：可见性是指一个线程对于共享变量的更新，对于后续访问该变量的线程是否可见的问题。

为了阐述可见性问题，我们先来简单介绍**处理器缓存**的概念。

现代处理器处理速度远大于主内存的处理速度，所以在主内存和处理器之间加入了寄存器，高速缓存，写缓冲器以及无效化队列等部件来加速内存的读写操作。也就是说，我们的处理器可以和这些部件进行读写操作的交互，这些部件可以称为处理器缓存。

处理器对内存的读写操作，其实仅仅是与处理器缓存进行了交互。一个处理器的缓存上的内容无法被另外一个处理器读取，所以另外一个处理器必须通过缓存一致性协议来读取的其他处理器缓存中的数据，并且同步到自己的处理器缓存中，这样保证了其余处理器对该变量的更新对于另外处理器是可见的。

在单处理器中，为什么也会出现可见性的问题呢?

单处理器中，由于是多线程并发编程，所以会存在线程的上下文切换，线程会将对变量的更新当作上下文存储起来，导致其余线程无法看到该变量的更新。所以单处理器下的多线程并发编程也会出现可见性问题的。

可见性如何保证?

- 当前处理器需要**刷新处理器缓存**，使得其余处理器对变量所做的更新可以同步到当前的处理器缓存中
- 当前处理器对共享变量更新之后，需要**冲刷处理器缓存**，使得该更新可以被写入处理器缓存中

有序性：

定义：有序性是指一个处理器上运行的线程所执行的内存访问操作在另外一个处理器上运行的线程来看是否有序的问题。

重排序：
为了提高程序执行的性能，Java编译器在其认为不影响程序正确性的前提下，可能会对源代码顺序进行一定的调整，导致程序运行顺序与源代码顺序不一致。

重排序是对内存读写操作的一种优化，在单线程环境下不会导致程序的正确性问题，但是多线程环境下可能会影响程序的正确性。

重排序举例：
Instance instance = new Instance()都发生了啥?
具体步骤如下所示三步：

- 在堆内存上分配对象的内存空间
- 在堆内存上初始化对象
- 设置instance指向刚分配的内存地址

第二步和第三步可能会发生重排序，导致引用型变量指向了一个不为null但是也不完整的对象。（**在多线程下的单例模式中，我们必须通过volatile来禁止指令重排序**）

解析：

14

- **原子性**是一组操作要么完全发生，要么没有发生，其余线程不会看到中间过程的存在。注意，**原子操作+原子操作不一定还是原子操作。**
- **可见性**是指一个线程对共享变量的更新**对于另外一个线程是否可见**的问题。
- **有序性**是指一个线程对共享变量的更新在其余线程看起来是**按照什么顺序执行**的问题。
- 可以这么认为，**原子性 + 可见性 -> 有序性**

（2）谈谈你对synchronized关键字的理解。

答： synchronized是Java中的一个关键字，是一个内部锁。它可以使用在方法上和方法块上，表示同步方法和同步代码块。在多线程环境下，同步方法或者同步代码块在同一时刻只允许有一个线程在执行，其余线程都在等待获取锁，也就是实现了整体并发中的局部串行。

内部锁底层实现：

- 进入时，执行**monitorenter**，将计数器+1，释放锁**monitorexit**时，计数器-1
- 当一个线程判断到计数器为0时，则当前锁空闲，可以占用；反之，当前线程进入等待状态

synchronized内部锁对原子性的保证：

锁通过互斥来保障原子性，互斥是指一个锁一次只能被一个线程所持有，所以，临界区代码只能被一个线程执行，即保障了原子性。

synchronized内部锁对可见性的保证：

synchronized内部锁通过写线程**冲刷处理器缓存**和读线程**刷新处理器缓存**保证可见性。

- 获得锁之后，需要**刷新处理器缓存**，使得前面写线程所做的更新可以同步到本线程。
- 释放锁需要**冲刷处理器缓存**，使得当前线程对共享数据的改变可以被推送到下一个线程处理器的高速缓冲中。

synchronized内部锁对有序性的保证：

由于原子性和可见性的保证，使得写线程**在临界区中**所执行的一系列操作在读线程所执行的临界区**看起来像是完全按照源代码顺序执行的**，即保证了有序性。

解析：

synchronized是Java中的关键字，其实就是一个内部锁，**关于内部锁的考察也是Java面试中的高频考点。**内部锁可以使用在方法上和代码块上，被内部锁修饰的区域又叫做临界区。如下所示：

```
1  package niuke;
2
3  public class SynchronizedTest {
4
5      public static void main(String[] args) {
6
```

```
        synchronized (SynchronizedTest.class){
            System.out.println("这是一个同步方法块");
        }

    }

    public synchronized void test(){
        System.out.println("这是一个同步方法，因为在方法上使用了synchronized关键字");
    }
}
```

接下来，我们继续介绍其相关知识点。锁做为一种资源，**JVM对资源的调度分为公平调度和非公平调度方式**。

公平调度方式：

按照**申请的先后顺序**授予资源的独占权。

非公平调度方式：

在该策略中，资源的持有线程释放该资源的时候，等待队列中一个线程会被唤醒，而该线程从被唤醒到其继续执行可能需要一段时间。在该段时间内，**新来的线程（活跃线程）**可以先被授予该资源的独占权。

如果新来的线程占用该资源的时间不长，那么它完全有可能在被唤醒的线程继续执行前释放相应的资源，从而不影响该被唤醒的线程申请资源。

优缺点分析：

非公平调度策略：

- 优点：吞吐率较高，单位时间内可以为更多的申请者调配资源
- 缺点：资源申请者申请资源所需的时间偏差可能较大，并可能出现线程饥饿的现象

公平调度策略：

- 优点：线程申请资源所需的时间偏差较小；不会出现线程饥饿的现象；适合在资源的持有线程占用资源的时间相对长或者资源的平均申请时间间隔相对长的情况下，或者对资源申请所需的时间偏差有所要求的情况下使用；
- 缺点：吞吐率较小

接下来，我们一起来看看JVM对synchronized内部锁的调度方式吧。

JVM对synchronized内部锁的调度：

JVM对内部锁的调度是一种**非公平的调度方式**，JVM会给每个内部锁分配一个**入口集（Entry Set）**，用于记录等待获得相应内部锁的线程。当锁被持有的线程释放的时候，该锁的入口集中的任意一个线程将会被唤醒，从而得到再次申请锁的机会。被唤醒的线程等待占用处理器运行时可能还有其他新的活跃线程与该线程抢占这个被释放的锁.

（3）谈谈你对volatile关键字的理解。

答：volatile关键字是一个轻量级的锁，**可以保证可见性和有序性，但是不保证原子性。**

解析：

- volatile 可以保证**主内存和工作内存直接产生交互**，进行读写操作，保证可见性
- volatile 仅能保证变量**写操作的原子性**，不能保证读写操作的原子性。
- volatile可以**禁止指令重排序**（通过插入内存屏障），典型案例是在单例模式中使用。

14

volatile变量的开销：

volatile不会导致线程上下文切换，但是其读取变量的成本较高，因为其每次都需要从高速缓存或者主内存中读取，无法直接从寄存器中读取变量。

volatile在什么情况下可以替代锁?

volatile是一个轻量级的锁，适合多个线程**共享一个状态变量**，锁适合多个线程**共享一组状态变量**。可以**将多个线程共享的一组状态变量合并成一个对象**，用一个volatile变量来引用该对象，从而替代锁。

（4）ReentrantLock和synchronized的区别：

答：ReentrantLock**是显示锁**，其提供了一些内部锁不具备的特性，但并不是内部锁的替代品。**显式锁支持公平和非公平的调度方式**，默认采用非公平调度。

synchronized 内部锁简单，但是不灵活。显示锁支持在一个方法内申请锁，并且在另一个方法里释放锁。**显示锁定义了一个tryLock（）方法，尝试去获取锁**，成功返回true，失败并不会导致其执行的线程被暂停而是直接返回false，即可以**避免死锁**。

总结：

在本小节中，主要讲解了**与线程安全相关的原子性，可见性以及有序性。另外对常用关键字synchronized内部锁以及轻量级锁volatile，ReentrantLock显示锁**都进行了介绍。本小节所述均属于线程安全相关的基础知识点，**在面试中出现频率很高**，希望大家可以加强理解与掌握，在面试中务必准确阐述。下一小节中，我们将重点讲述**线程池以及阻塞队列**等知识点。

限于作者水平，文章中难免会有不妥之处。大家在学习过程中遇到我没有表达清楚或者表述有误的地方，欢迎随时在文章下边指出，我会及时关注，随时改正。另外，大家有任何话题都可以在下边留言，我们一起交流探讨。

讨论 评论

刘畅201904211606816 1#

打卡

发表于 2019-12-31 18:33:21 赞(0) 回复(0)

ljn1018 2#

打卡

发表于 2020-01-02 16:22:28 赞(0) 回复(0)

凌锋201903141546714 3#

volatile那得底层实现还是不太懂

发表于 2020-01-04 20:39:40 赞(0) 回复(3)

14

我是祖国的花朵 作者 ： volatile变量的读写操作底层配合使用了内存屏障，包括释放屏障和存储屏障，加载屏障和获取屏障，以此来保证有序性和可见性。

2020-01-04 20:54:37 赞(0) 回复(0)

凌锋201903141546714 ： 有的博文说的LOCK前缀又是什么情况?

2020-01-05 10:46:00 赞(0) 回复(0)

Fleer 回复 凌锋201903141546714 ： java中volatile修饰的变量汇编之后会出现LOCK前缀，某个线程对volatile变量进行写操作，JVM就将这个变量的缓存数据写回到主内存，通过缓存一致性协议当处理器发现本地缓存失效，即会从主内存中读取最新值 此条保证了volatile变量的可见性

2020-02-25 16:32:31 赞(0) 回复(0)

请输入你的观点

回复

柳杰201905011049420 4#

我也不太懂，看得云里雾里

发表于 2020-01-15 16:01:48 赞(0) 回复(0)

牧水s 5#

打卡...博主只是介绍了关于这些方面一些比较基础的知识。其实每个点都是可以引申很多知识的，需要自己查资料😁

发表于 2020-01-15 17:19:00 赞(2) 回复(0)

找不到钥匙只好撞门 6#

synchronized关键字对有序性的保证应该和添加内存屏障禁止重排序有关?

发表于 2020-02-05 20:43:32 赞(0) 回复(1)

我是祖国的花朵 作者 ： 是得，这么理解没错。

2020-02-08 17:40:01 赞(0) 回复(0)

请输入你的观点

回复

星如月勿忘初心 7#

打卡

发表于 2020-02-17 16:12:38 赞(0) 回复(0)

禾田誉子 8#

看了网上的一些解释，都说volatile关键字并不能保证操作的原子性。所以博主前文提到的"Java语言规范中又规定，volatile关键字修饰的变量可以保证其写操作的原子性"是否严谨？ 14

发表于 2020-02-25 11:48:33 赞(0) 回复(13)

可乐酱酱： volatile不能保证原子性 博主说错了

2020-02-25 19:05:20 赞(0) 回复(0)

禾田誉子 回复 可乐酱酱： 看了博主文章后面的说法，提的都是“可以保证写操作原子性，不能保证读写操作原子性”，我不能理解

2020-02-26 10:14:24 赞(0) 回复(0)

可乐酱酱： 我20个线程，循环1000次 +不到20000啊

2020-02-26 10:16:42 赞(0) 回复(0)

可乐酱酱 回复 可乐酱酱： 这个应该算写吧

2020-02-26 10:17:16 赞(0) 回复(0)

禾田誉子 回复 可乐酱酱： 我不懂你这个的意思...等等看博主有没出来解释吧

2020-02-26 15:34:23 赞(0) 回复(0)

Ruoji55555： 今天刚好看书看到这一节。 "volatile关键字被称为轻量级锁，在原子性方面它仅能保障写volatile变量操作的原子性，但没有锁的排他性"--《java多线程编程实战指南》

2020-02-26 16:09:24 赞(0) 回复(0)

Fleer： volatile修饰变量如果进行复合的操作是无法保证原子性 例如自加或者自减 如果是原子操作是可以的 个人拙见

2020-02-26 18:53:49 赞(0) 回复(0)

我是祖国的花朵 N 作者： 在文中，我们也说到了，volatile变量可以保证写操作和读操作的原子性，但是不可以保证读写操作的原子性。所以，我们在回答这个问题的时候，可以说volatile可以保证可见性和有序性，但是无法保证变量的原子性。能否保证其实都是有前提条件的。

2020-02-26 20:05:53 赞(1) 回复(0)

我是祖国的花朵 N 作者 回复 可乐酱酱： 你好，不保证原子性是有条件的，单独的写操作和读操作是可以保证原子性的。

2020-02-26 20:06:31 赞(0) 回复(0)

我是祖国的花朵 N 作者 回复 Ruoji55555： 黄文海老师的吗？这本书推荐

2020-02-26 20:07:23 赞(0) 回复(0)

首页 上一页 1 2 下一页 尾页

请输入你的观点

回复

14

牛客248955670号 9#

打开

发表于 2020-02-25 16:08:41 赞(0) 回复(0)

CuCl2 10#

我好菜啊T^T

发表于 2020-03-07 10:35:36 赞(0) 回复(0)

怡乐未央 11#

讲的是啥啊

发表于 2020-04-13 19:25:54 赞(0) 回复(0)

现在的菜鸡，未来的顶尖技术总监 12#

楼主你好，在

- Java**语言规范中**，保证了除long和double型以外的任何变量的写操作都是原子操作　　这句话中，所说的变量是其余的6种基本变量吧

发表于 2020-05-22 10:36:24 赞(0) 回复(1)

我是祖国的花朵 作者 ： 其他变量还包括引用变量的

2020-05-22 13:13:01 赞(0) 回复(0)

请输入你的观点

回复

Shawn_Liu 13#

05/31 打卡

发表于 2020-06-01 10:50:17 赞(0) 回复(0)

sky爱吃青菜 14#

ReentrantLock和synchronized的区别：

（1）是否支持公平锁
（2）显示加锁和释放锁
（3）是否可以中断
（4）是否可以获取锁状态，例如**tryLock（）**

发表于 2020-06-21 12:29:31 赞(0) 回复(0)

14